# External Memory Interface Handbook

## Volume 1: Introduction and Specifications

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

EMI_INTRO-2.0

Document last updated for Altera Complete Design Suite version: 10.1
Document publication date: December 2010

Subscribe

# External Memory Interface Handbook Volume 1

## Section I. About This Handbook

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

EMI_INTRO_ABOUT-2.1

Document last updated for Altera Complete Design Suite version: 10.1
Document publication date: December 2010

Subscribe

# Contents

# 1. How to Use this Handbook

The *External Memory Interface Handbook* contains information that you require to implement an external memory interface. The handbook focuses on the Altera® solution for DDR, DDR2, DDR3 SDRAM; QDR II and QDR II+ SRAM; and RLDRAM II interfaces. The handbook is organized with a typical design flow in mind, into the following six volumes:

- "Introduction to Altera External Memory Interfaces"
- "Device, Pin, and Board Layout Guidelines"
- "Implementing Altera Memory Interface IPs"
- "Simulation, Timing Analysis, and Debugging"
- "Implementing a Custom PHY"
- "Design Flow Tutorials"

## Introduction to Altera External Memory Interfaces

This volume includes this *How to Use this Handbook* chapter, recommended design flow, and a glossary. In addition, this volume includes basic information for various memory standards.

The *Specifications* section lists a complete scorecard of Altera device family support for various memory standards.

## Device, Pin, and Board Layout Guidelines

This volume describes the initial steps of selecting the correct Altera device with the right resources and number of user I/O available for that interface.

The second section of the volume describes how you can select the correct termination and drive strength based on your board simulation for DDR, DDR2, and DDR3 SDRAM interfaces. It offers board results correlated with board simulation and the Altera recommended settings based on this correlation.

## Implementing Altera Memory Interface IPs

This volume covers the following Altera memory IP products:

- DDR and DDR2 SDRAM High Performance Controllers
- DDR and DDR2 SDRAM High Performance Controllers II
- DDR3 SDRAM High Performance Controllers
- QDR II and QDR II+ SRAM Controllers with UniPHY
- RLDRAM II Controllers with UniPHY

Each IP has two modules: the PHY and the memory controller. The DDR, DDR2, and DDR3 SDRAM high-performance controllers use the Altera ALTMEMPHY megafunction for the PHY, which you can use standalone from the controller. The QDR II, QDR II+, and RLDRAM II IP use the Altera UniPHY PHY.

The functional description of each of these modules is described separately so you do not need to read the memory controller functional description if you are creating your own custom memory controller.

The volume also contains information on how to implement the IP, including a description of the parameterization GUI, the required constraints, and latency information. The last chapter of this volume includes timing diagrams showing the memory operations that may help you debug your system or help you create a custom memory controller.

## Simulation, Timing Analysis, and Debugging

When you have implemented your external memory interface, you can use this volume for information on how to perform functional simulation, how to analyze timing, and how to debug your design.

## Implementing a Custom PHY

This volume describes the steps to create a custom PHY and offers examples on some custom PHYs that are already available for some interfaces.

## Design Flow Tutorials

This volume offers step-by-step tutorials in creating a memory interface for specific Altera development boards for debugging and testing. In addition, this volume also discusses special information such as how to implement external memory interfaces using SOPC builder or how to implement multiple memory interfaces. The design flow tutorials follow the flow in "Recommended Design Flow" on page 1–1.

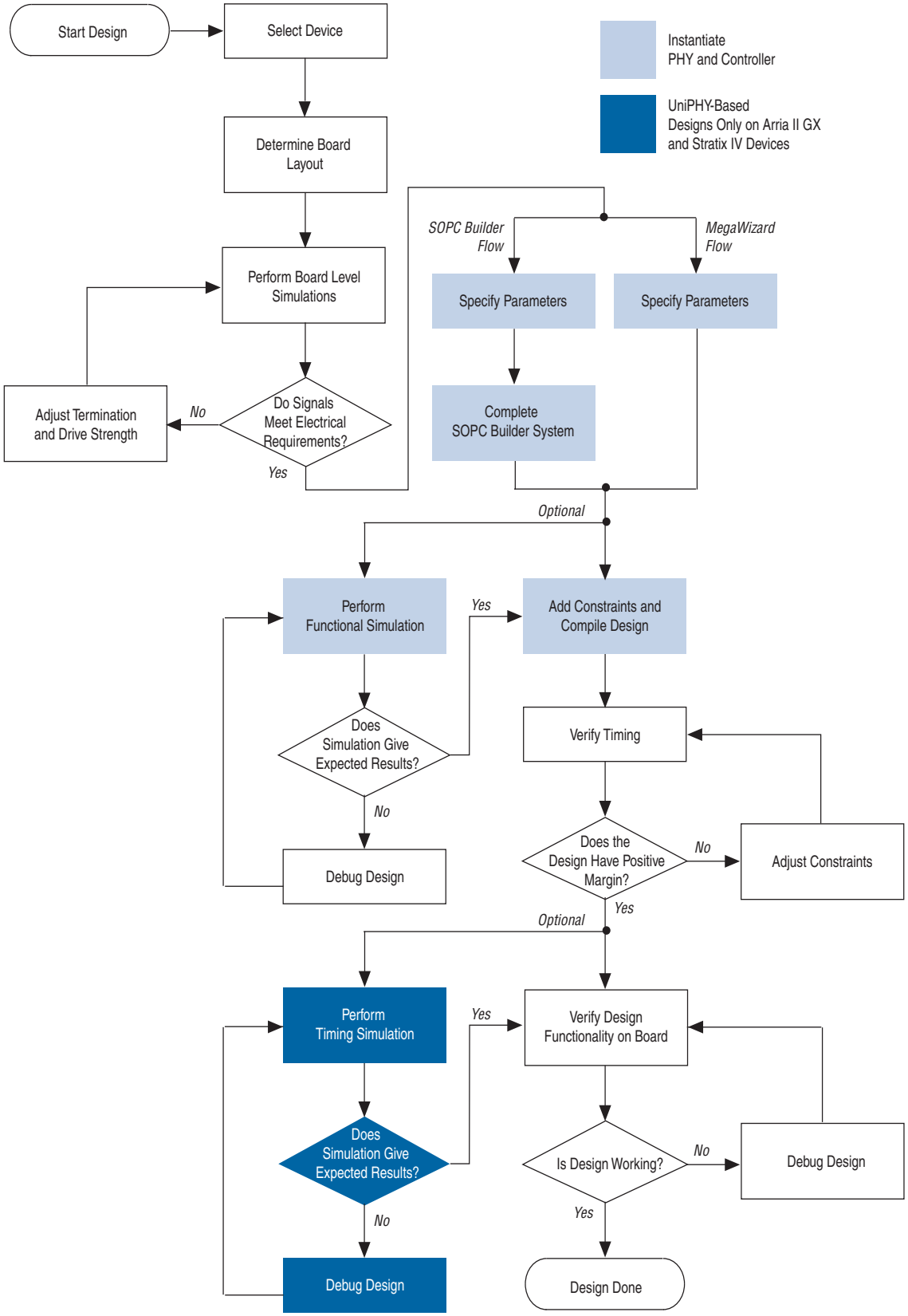
# 2. Recommended Design Flow

This chapter describes the Altera-recommended design flow for successfully implementing external memory interfaces in Altera devices. Altera recommends that you create an example top-level file with the desired pin outs and all interface IP instantiated, which enables the Quartus® II software to validate your design and resource allocation before PCB and schematic sign off. Use the "Design Checklist" on page 1–6, to verify whether you have performed all the recommended steps in creating a working and robust external memory interface.

Figure 1–1 shows the design flow to provide the fastest out-of-the-box experience with external memory interfaces in Altera devices. This topic directs you where to find information on how to perform each step of the recommended design flow. The flow assumes that you are using Altera IP to implement the external memory interface.

For design examples that follow the recommended design flow in this chapter, refer to *Volume 6: Design Flow Tutorials* of the *External Memory Interface Handbook*.

**Figure 2–1. External Memory Interfaces Design Flowchart**



# Select a Device

For more information on selecting a device, refer to the *Device and Pin Planning* section in volume 2 of the *External Memory Interface Handbook*.

# Determine Board Layout

Altera recommends prelayout SI simulations (line simulations) should take place before board layout and that you use these parameters and rules during the initial design development cycle. Advanced I/O timing and board trace models now directly impact device timing closure.

In addition, the termination scheme that you use, the drive strength setting on the FPGA, and the loading seen by the driver can directly affect the signal integrity. You must understand the tradeoffs between the different types of termination schemes and the effects of output drive strengths and loading, to choose the best possible settings for your designs.

For more information, refer to the *Board Layout Guidelines* section in volume 2 of the *External Memory Interface Handbook*.

# Perform Board-Level Simulations

To determine the correct board constraints, perform board-level simulations to see if the settings provide the optimal signal quality. With many variables that can affect the signal integrity of the memory interface, simulating the memory interface provides an initial indication of how well the memory interface performs. There are various EDA simulation tools available to perform board-level simulations. The simulations should be performed on the data, data strobe, control, command, and address signals. If the memory interface does not have good signal integrity, adjust the settings, such as drive strength setting, termination scheme or termination values to improve the signal integrity (realize that changing these settings affects the timing and it may be necessary to go back to the timing closure if these change).

For detailed information about understanding the different effects on signal integrity design, refer to the *Board Layout Guidelines* section in volume 2 of the *External Memory Interface Handbook.*

Enter topology information from your board-level simulations into the Quartus II board trace model information. The typical information required includes, but is not limited to, the following values:

- Near and far trace lengths
- Near and far trace distributed inductance
- Near and far trace distributed capacitance
- Near end deration capacitor values (if fitted)
- Far end capacitive (IC) load
- Far end termination values

### Device-Side Termination

Many Altera devices support both series and parallel OCT resistors to improve signal integrity. OCT eliminates the need for external termination resistors on the FPGA side, which simplifies board design and reduces overall board cost. You can dynamically switch between the series and parallel OCT resistor depending on whether the FPGA devices are performing a write or a read operation. The OCT features offer user-mode calibration to compensate for any variation in VT during normal operation to ensure that the OCT values remain constant. The parallel and series OCT features are available in either 25 or 50 Ω settings.

### Memory-Side Termination

The DDR2, DDR3 SDRAM, and QDR II SRAM have a dynamic parallel ODT feature that you can turn on when the FPGA is writing to the memory and turn off when the FPGA is reading from the memory. To further improve signal integrity, DDR2 SDRAM supports output drive strength control so that the driver can better match the transmission line. DDR3 SDRAM devices additionally support calibrated output impedances.

For more information on available settings of the ODT, the output drive strength features, and the timing requirements for driving the ODT pin, refer to your DDR2 or DDR3 SDRAM datasheet.

## Adjust Termination and Drive Strength

Although the recommended terminations are based on the simulations and experimental results, you must perform simulations, either using I/O buffer information specification (IBIS) or HSPICE models, to determine the quality of signal integrity on your designs.

Any changes made to the board should also be made in the board trace model in the Quartus II software.

For information on Altera-recommended terminations for memory interfaces, refer to the *Board Layout Guidelines* section in volume 2 of the *External Memory Interface Handbook*.

## Instantiate PHY and Controller

After selecting the appropriate device and memory type, create a project in the Quartus II software that targets the device and memory type.

When implementing external memory interfaces, Altera recommends that you use Altera memory interface IP, which includes a PHY that you can use with the Altera high-performance controller or with your own custom controller.

Instantiating the PHY and controller includes the following steps:

- Specify parameters
- Perform functional simulation
- Add constraints and compile design

For more information about specifying parameters, adding constraints, and compiling, refer to the *DDR and DDR2 SDRAM High-Performance Controller and ALTMEMPHY IP User Guide* section and the *DDR3 SDRAM High-Performance Controller and ALTMEMPHY IP User Guide* section in volume 3 of the *External Memory Interface Handbook*.

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

## Verify Timing

For more information about verifying timing, refer to the *Timing Analysis* section in volume 4 of the *External Memory Interface Handbook.*

## Adjust Constraints

In the timing report of the design, you can see the worst case setup and hold margin for the different paths in the design. If the setup and hold margin are unbalanced, achieve a balanced setup and hold margin by adjusting the phase setting of the clocks associated with these paths.

For example, for the address and command margin, the address and command outputs are clocked by an address and command clock that can be different with respect to the system clock, which is –30°. The system clock controls the clock outputs going to the memory. If the report timing script indicates that using the default phase setting for the address and command clock results in more hold time than setup time, adjust the address and command clock to be less negative than the default phase setting with respect to the system clock, so that there is less hold margin. Similarly, adjust the address and command clock to be more negative than the default phase setting with respect to the system clock if there is more setup margin.

For more information on adjusting constraints, refer to the *Timing Analysis* section in volume 4 of the *External Memory Interface Handbook*.

## Perform Timing Simulation

This step is optional, but recommended to ensure that the IP is working properly. This step only applies to UniPHY-based interfaces (on Arria® II GX and Stratix® IV devices only), as ALTMEMPHY-based interfaces do not support timing simulation.

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

## Verify Design Functionality

Perform system level verification to correlate the system against your design targets, using the Altera SignalTap® II logic analyzer.

For more information about using the SignalTap II analyzer, refer to the *Debugging* section in volume 4 of the *External Memory Interface Handbook*.

# Design Checklist

This topic contains a design checklist that you can use when implementing external memory interfaces in Altera devices.

| | Done | |
|---|---|---|
| | | **Select Device** |
| 1. | ☐ | Select the memory interface frequency of operation and bus width.<br>For information about selecting memory, refer to the *Memory Standard Overview* section in volume 1 of the External Memory Interface Handbook. |
| 2. | ☐ | Select the FPGA device density and package combination that you want to target.<br>For information about selecting an Altera device, refer to the *Device and Pin Planning* section in volume 2 of the *External Memory Interface Handbook*. |
| 3. | ☐ | Ensure that the target FPGA device supports the desired clock rate and memory bus width. Also the FPGA must have sufficient I/O pins for the DQ/DQS read and write groups.<br>For detailed device resource information, refer to the relevant device handbook chapter on external memory interface support.<br>For information about supported clock rates for external memory interfaces, refer to the *External Memory Interface System Specifications* section in volume 1 of the *External Memory Interface Handbook*. |
| | | **Determine Board Layout** |
| 4. | ☐ | Select the termination scheme and drive strength settings for all the memory interface signals on the memory side and the FPGA side. |
| 5. | ☐ | Ensure you apply appropriate termination and drive strength settings on all the memory interface signals, and verify using board level simulations. |
| 6. | ☐ | Use board level simulations to pick the optimal setting for best signal integrity. On the memory side, Altera recommends the use of external parallel termination on input signals to the memory (write data, address, command, and clock signals).<br>For information, refer to the *Board Layout Guidelines* section in volume 2 of the *External Memory Interface Handbook*. |
| | | **Perform Board Level Simulations** |
| 7. | ☐ | Perform board level simulations, to ensure electrical and timing margins for your memory interface |
| 8. | ☐ | Ensure you have a sufficient eye opening using simulations. Use the latest FPGA and memory IBIS models, board trace characteristics, drive strength, and termination settings in your simulation.<br>Any timing uncertainties at the board level that you calculate using simulations must be used to adjust the input timing constraints to ensure the accuracy of Quartus II timing margin reports. For example crosstalk, ISI, and slew rate deration.<br>For information, refer to the *Board Layout Guidelines* section in volume 2 of the *External Memory Interface Handbook*. |

| | Done | |
|---|---|---|
| | | **Instantiate PHY and Controller** |
| 9. | ☐ | Parameterize and instantiate the Altera external memory IP for your target memory interface.<br>You have the following three choices in implementing a memory interface in the Quartus II software:<br>■ Using the Altera memory controller and PHY.<br>■ For information about how to implement a specific memory interface, refer to *Volume 3: Implementing Altera Memory Interface IP* of the *External Memory Interface Handbook*.<br>■ The *Parameter Settings* chapter of each section describes the IP supported features page by page.<br>■ Using the Altera PHY with your own custom controller.<br>■ For information about how to implement a specific memory interface, refer to *Volume 3: Implementing Altera Memory Interface IP* of the *External Memory Interface Handbook*.<br>■ The *Parameter Settings* chapter of each section describes the IP supported features page by page. The *Functional Description* chapter describes the workings of the PHY and how you can connect a custom controller to the PHY.<br>■ Implement a custom PHY and custom controller.<br>For information about creating custom IP, refer to *Volume 5: Implementing CustomMemory Interface PHY* of the *External Memory Interface Handbook*. |
| 10. | ☐ | Ensure that you perform the following actions:<br>■ Pick the correct memory interface data rates, width, and configurations.<br>■ For DDR, DDR2, and DDR3 SDRAM interfaces, ensure that you derate the tIS, tIH, tDS, and tDH parameters, as necessary.<br>■ Include the board skew parameter for your board. |
| 11. | ☐ | Connect the PHY's local signals to your driver logic and the PHY's memory interface signals to top-level pins.<br>Ensure that the local interface signals of the PHY are appropriately connected to your own logic. If the ALTMEMPHY megafunction is compiled without these local interface connections, you may encounter compilation problems, when the number of signals exceeds the pins available on your target device.<br>For more information about the example top-level file, refer to the *Functional Description* chapter for the relevant memory controller.<br>You may also use the example top-level file as an example on how to connect your own custom controller to the Altera memory PHY. |
| | | **Perform Functional Simulation** |
| 12. | ☐ | Simulate your design using the RTL functional model.<br>Use the IP functional simulation model with your own driver logic, testbench, and a memory model, to ensure correct read and write transactions to the memory.<br>You may need to prepare the memory functional model by setting the speed grade and device bus mode.<br>For more information about simulation, refer to the *Simulation* section in volume 4 of the *External Memory Interface Handbook*. |

| Done | | |
|---|---|---|
| | | **Add Contraints** |
| 13. | ☐ | Add timing constraints. The wizard-generated **.sdc** file adds timing constraints to the interface. However, you may need to adjust these settings to best fit your memory interface configuration. |
| 14. | ☐ | Add pin settings and DQ group assignments. The wizard-generated **.tcl** file includes I/O standard and pin loading constraints to your design. |
| 15. | ☐ | Ensure that generic pin names used in the constraint scripts are modified to match your top-level pin names. The loading on memory interface pins is dependent on your board topology (memory components). |
| 16. | ☐ | Add pin location assignments. However, you need to assign the pin location assignments manually using the Pin Planner. |
| 17. | ☐ | Ensure that the example top-level file or your top-level logic is set as top-level entity. |
| 18. | ☐ | Adjust optimization techniques, to ensure the remaining unconstrained paths are routed with the highest speed and efficiency:<br>a. On the Assignments menu click **Settings**.<br>b. Select **Analysis & Synthesis Settings**.<br>c. Select **Speed** under **Optimization Technique**.<br>d. Expand **Fitter Settings**.<br>e. Turn on **Optimize Hold Timing** and select **All Paths**.<br>f. Turn on **Optimize Fast Corner Timing**.<br>g. Select **Standard Fit** under **Fitter Effort**. |
| 19. | ☐ | Provide board trace delay model. For accurate I/O timing analysis, you specify the board trace and loading information in the Quartus II software. This information should be derived and refined during your board development process of prelayout (line) simulation and finally post-layout (board) simulation. Provide the board trace information for the output and bidirectional pins through the board trace model in the Quartus II software.<br><br>For more information, refer to the *Add Constraints* chapter for the relevant memory standard in volume 3 of the *External Memory Interface Handbook* or refer to *Volume 6: Design Flow Tutorials*. |
| | | **Compile Design and Verify Timing** |
| 20. | ☐ | Compile your design and verify timing closure using all available models. |
| 21. | ☐ | Run the wizard-generated <*variation_name*>**_report_timing.tcl** file, to generate a custom timing report for each of your IP instances. Run this process across all device timing models (slow 0° C, slow 85° C, fast 0° C). |
| 22. | ☐ | If there are timing violations, adjust your constraints to optimize timing |
| 23. | ☐ | As required, adjust PLL clock phase shift settings or appropriate timing and location assignments margins for the various timing paths within the IP.<br><br>For information, refer to the *Timing Analysis* section in volume 4 of the *External Memory Interface Handbook*. |

| | Done | |
|---|---|---|
| | | **Perform Timing Simulation** |
| 24. | ☐ | Perform gate-level or timing simulation to ensure that all the memory transactions meet the timing specifications with the vendor's memory model. Timing simulation is only supported with UniPHY-based memory interfaces.<br>For more information about simulation, refer to the *Simulation* section in volume 4 of the *External Memory Interface Handbook*. |
| | | **Verify Design Functionality** |
| 25. | ☐ | Verify the functionality of your memory interface in the system<br>For more information, refer to *Volume 6: Design Flow Tutorials*. in the *External Memory Interface Handbook*. |

# 3. Glossary

This chapter lists the definitions of the terms that the Altera external memory interfaces solutions use.

Table 1–1 shows a glossary of terms.

**Table 3–1. Glossary of Terms (Part 1 of 3)**

| Term | Description |
|---|---|
| ×36 emulation | A QDR II or QDR II+ SRAM implementation to support ×36 QDR II and QDR II+ SRAM devices with Stratix III or Stratix IV devices that do not support ×36 DQS groups. Two ×18 DQS groups are combined to emulate the ×36 group. The CQ and CQn signals have to be split on the board to go to the DQS and CQn pins of the two ×18 DQS groups. |
| Advanced I/O timing | Advanced I/O timing allows the TimeQuest timing analyzer to produce enhanced timing reports based on the board layout information included in the design. Use advanced I/O timing with the board trace model. |
| Altera PHY interface (AFI) | The bus that connects calibrated PHYs with the Altera memory controller IP. |
| ALTMEMPHY megafunction | The Altera PHY IP that the high-performance controllers use. The latest generation PHY is the UniPHY IP. |
| Board trace model | The board trace model in the Quartus II software includes the board termination, trace length, and impedance in the project so that TimeQuest can produce enhanced timing reports with the included board information. You must have the advance I/O timing enabled when using he board trace model. |
| Calibration | The process of setting up the initial relationship between clocks. For example, the resynchronization window to provide the greatest timing margin in DDR, DDR2, and DDR3 SDRAM interfaces in Altera devices. The initial calibration is done only once at system reset after device initialization is complete. |
| Cascaded PLLs | A scheme whereby the clock for the PLL in the memory interface IP comes from another PLL. For more information about cascaded PLLs, refer to the *Device and Pin Planning* section in volume 2 of the *External Memory Interface Handbook*. |
| Cascaded | A signal topology, in which a signal is routed from one component to the next in series, but is buffered by each component. |
| Column address strobe (CAS) | A signal sent from a memory controller to a DRAM circuit to indicate the column address lines are valid. |
| Complementary clocks | A clocking scheme where only the rising edges of a clock and its inverted clock clock double data rate data into a device, for example on QDR II and QDR II+ SRAM interfaces. |
| Daisy-chain | A routing topology in which a single trace is routed from one component to the next, and so on, in series to the last component. |
| DDR | Double data-rate transfer, where data is latched and sent at both rising and falling edges of the (accompanying) clock. Operates at the full-rate clock frequency (twice the width of SDR data). |
| DDR3 SDRAM with leveling | A standard DDR3 SDRAM topology that requires the use of the ALTMEMPHY megafunction with read and write leveling. DDR3 SDRAM with levelling applies to any DDR3 DIMM interfaces. |
| DDR3 SDRAM without leveling | A non standard topology with synchronous DDR2-like balanced address, command, and clock layout. |

**Table 3–1. Glossary of Terms (Part 2 of 3)**

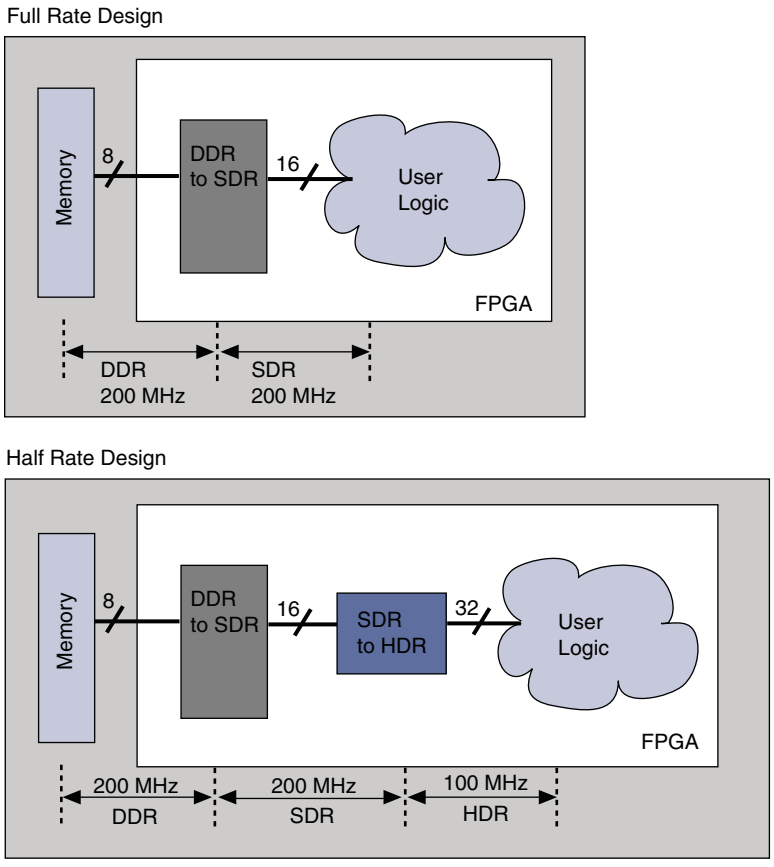
| Term | Description |
|---|---|
| Deskew | Aligning the data signals with respect to the clock signal to compensate for the skew between the data and clock signals, by using the delay chains in the IOEs. |
| Fly-by topology | When the termination is placed after the component at the end of the line. |
| Full-rate clock | Clock with a frequency that is equal to the frequency of the memory interface clock. |
| Full-rate controller | A memory controller implemented as a full-rate design. |
| Full-rate design | A variation of the memory interfaces where the clock frequency of the controller and user-interface logic is the same as the memory interface clock (refer to Figure 1–1). |
| Half data rate (HDR) | Data that changes on one edge of the half-rate clock (twice the width of DDR data and four times the width of SDR data). |
| Half-rate clock | Clock with a frequency that is half the frequency of the memory interface clock. |
| Half-rate controller | A memory controller implemented as a half-rate design. |
| Half-rate design | A variation of the memory interface where the clock frequency of the controller and user-interface logic is half of the memory interface clock frequency (refer to Figure 1–1). |
| High-performance controller | Altera memory controller that uses the ALTMEMPHY megafunction for the datapath. Altera offers the high-performance controller for DDR, DDR2, and DDR3 SDRAM interfaces. |
| High-performance controller II | Latest version of the Altera memory controller for DDR, DDR2, and DDR3 interfaces. Several new features, including command look-ahead, which improves interface efficiency. |
| Hybrid | Obsolete term for wraparound I/Os. |
| Legacy controller | The legacy integrated static datapath and controller MegaCore® functions with no support for calibration and tracking. For more information about legacy integrated static datapath and controller MegaCore functions, refer to the *DDR and DDR2 SDRAM Controller Compiler User Guide*, *QDR II SRAM Controller MegaCore Function User Guide*, and *RLDRAM II Controller MegaCore Function User Guide*. |
| Memory pessimism removal (MPR) | Memory chip calibration—when the PHY calibrates some portion of the JEDEC variation. |
| Multiple chip select and multiple rank | Multiple chip select is the general term, whereas multple rank is reserved for DIMMs. A rank refers to a group of DRAM components. Each rank has an individual CS signal. When there are *N* such ranks in a system, it is referred to as multiple rank. |
| Non-AFI | The legacy interface standard between the controller and PHY. Not recommended for new designs. |
| On-chip termination (OCT) | An FPGA device feature that eliminates the need for external resistors for termination. The ALTMEMPHY megafunction supports dynamic OCT for DDR2 and DDR3 SDRAM variations for Altera devices and static OCT for QDR II and QDR II+ SRAM and RLDRAM II variations. |
| On-die termination (ODT) | A memory vendor device feature equivalent to Altera's OCT. |
| Planar | Topology of some DIMM raw cards. |
| Quad data rate (QDR) | Two data writes and two data reads per memory clock cycle. |
| RLDRAM II | A DDR memory standard that has reduced latency and simpler bank management compared to the DDR, DDR2, or DDR3 SDRAM memory standard. |
| RLDRAM II CIO | A variant of the RLDRAM II devices that uses common I/O pins for read and write data pins. |
| RLDRAM II SIO | A variant of the RLDRAM II devices that uses separate I/O pins for read and write data pins. |
| Row address strobe (RAS) | A signal sent from a memory controller to a DRAM circuit to indicate the row address lines are valid. |
| $R_{UP}$ and $R_{DN}$ pins | Reference pins for the OCT block. |
| Sequencer | The logic block that performs calibration and tracking operations. |

**Table 3–1. Glossary of Terms (Part 3 of 3)**

| Term | Description |
|---|---|
| Signal splitter | The ablility of the Stratix III and Stratix IV DDIO output to feed both the positive and negative legs of the differential I/O pins. |
| SDR | Data that changes on one edge of the full-rate clock. |
| T topology | A tree-type topology with balanced routing as used on DDR2 SDRAM DIMMs for address and command signals. |
| Tracking | Performed as a background process during device operation, to track voltage and temperature (VT) variations to maintain the data valid window that was achieved at calibration. Only applies to DDR, DDR2, and DDR3 SDRAM interfaces. |
| UniPHY | The latest generation Altera PHY IP. |
| Wraparound interface | Previously referred to as hybrid memory interface. A memory interface where the read or write or bidirectional datapath is split across the top or bottom and left or right of the device. However, a read datapath on one edge and a write datapath on an adjacent edge is not classed as a wraparound interface. |
| ZQ calibration | The DDR commands that calibrate the DDR3 SDRAM component ODT values. |

Figure 1–1 shows the differences in the datapath width and frequency at which data is handled between full-rate and half-rate controllers.

**Figure 3–1. Full-Rate and Half-Rate Controller Description**



# External Memory Interfaces Handbook Volume 1

## Section II: Memory Standard Overviews

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

EMI_INTRO_OVER-2.0

Document last updated for Altera Complete Design Suite version: 10.1
Document publication date: December 2010

Subscribe

# Contents

# 1. Selecting your Memory Component

This chapter details some of the high-speed memory selection criteria and describes some typical applications where these memories are used. It looks at the main types of high-speed memories available, memory selection based on strengths and weaknesses, and which Altera® FPGAs these devices can interface with. It concludes with some typical application examples.

This chapter highlights the memory component's capability. The Altera IP may or may not support all of the features supported by the memory.

For the maximum supported performance supported by Altera FPGAs, refer to the *External Memory Interface System Specifications* section in volume 1 of the *External Memory Interface Handbook*.

System architects must resolve a number of complex issues in high-performance system applications that range from architecture, algorithms, and features of the available components. Typically, one of the fundamental problems in these applications is memories, as the bottlenecks and challenges of system performance often reside in its memory architecture. As higher speeds become necessary for external memories, signal integrity gets more difficult. Newer devices have added several features to overcome this issue. Altera FPGAs also support these advancements with dedicated I/O circuitry, various I/O standard support, and specialized intellectual property (IP).

## Memory Overview

The main considerations for choosing an external memory device are bandwidth, size, cost, latency, and power. Since no single memory type can excel in every area, system architects must determine the right balance for their design.

There are two common types of high-speed memories: DRAM and SRAM. DRAM devices are volatile memories offering a lower cost per bit than SRAM devices. A compact memory cell consisting of a capacitor and a single transistor makes this possible, as opposed to the six-transistor cell used in SRAM. However, as the capacitor discharges, the memory cell loses its state. This means that DRAM memory must be refreshed periodically, resulting in lower overall efficiency and more complex controllers. Generally, designers only choose DRAM where cost per bit is important.

### DDR, DDR2, and DDR3 SDRAM

The desktop computing market has positioned double data rate (DDR) SDRAM as a mainstream commodity product, which means this memory is very low-cost. DDR SDRAM is also high-density and low-power. Relative to other high-speed memories, DDR SDRAM has higher latency-they have a multiplexed address bus, which reduces the pin count (minimizing cost) at the expense of a longer and more complex bus

cycle. DDR2 SDRAM includes additional features such as increased bandwidth due to higher clock speeds, improved signal integrity on DIMMs with on-die terminations, and lower supply voltages to reduce power. DDR3 SDRAM is the latest generation of SDRAM and further increases bandwidth, lowers power, and improves signal integrity with fly-by and dynamic on-die terminations.

## RLDRAM and RLDRAM II

Reduced latency DRAM (RLDRAM) is optimized to reduce latency primarily for networking and cache applications. RLDRAM is partitioned into eight smaller banks. This partioning reduces the parasitic capacitance of the address and data lines, allowing faster accesses and reducing the probability of random access conflicts. Also, most DRAM memory types need both a row and column phase on a multiplexed address bus to support full random access, while RLDRAM supports a non-multiplexed address, saving bus cycles at the expense of more pins. RLDRAM utilizes higher operating frequencies and uses the 1.8V High-Speed Transceiver Logic (HSTL) standard with DDR data transfer to provide a very high throughput. RLDRAM II offers faster random access times, on-die termination, a delay-locked loop (DLL) for higher frequency operation, larger densities, wider data paths, and higher bus utilization compared with RLDRAM.

## QDR, QDR II, and QDR II+ SRAM

SRAMs are fundamentally different from DRAMs in that a typical SRAM memory cell consists of six transistors, while a DRAM cell consists of a transistor and a capacitor used to store a charge. Inherently, SRAM is a low-density, high-power memory device, with very low latency compared to DRAM (as the capacitor in the DRAM is slow). In most cases, SRAM latency is one clock cycle.

Quad Data Rate (QDR) SRAM has independent read and write ports that run concurrently at double data rate. QDR SRAM is true dual-port (although the address bus is still shared), which gives this memory a significantly higher bandwidth. QDR SRAM is best suited for applications where the required read/write ratio is near one-to-one. QDR II SRAM includes additional features such as increased bandwidth due to higher clock speeds, lower voltages to reduce power, and on-die termination to improve signal integrity. QDR II+ SDRAM is the latest and fastest generation.

# Memory Selection

One of the first considerations in choosing a high-speed memory is data bandwidth. Based on the system requirements, an approximate data rate to the external memory should be determined. Table 1–1 details the memory bandwidth for various technologies with the assumptions of a 32-bit data bus, operating at the maximum supported frequency in a Stratix® IV FPGA. The bandwidth column in this table includes a conservative DRAM bandwidth at 70 percent efficiency, which takes into consideration bus turnaround, refresh, burst length, and random access latency. The calculation assumes 85 % efficiency for QDR and QDR II SRAM.

**Table 1–1. Memory Bandwidth for 32-bit Wide Data Bus in Stratix IV FPGA**

| Memory | Clock Frequency (MHz) | Bandwidth for 32 bits (Gbps) | Bandwidth at % Efficiency (Gbps) *(1)* |
|---|---|---|---|
| DDR3 SDRAM | 533 | 34.1 | 23.9 |
| DDR2 SDRAM | 400 | 25.6 | 17.9 |
| DDR SDRAM | 200 | 12.8 | 9 |
| RLDRAM II | 400 | 25.6 | 17.9 |
| QDR SRAM | 200 | 25.6 | 21.8 |
| QDR II SRAM | 350 | 44.8 | 38.1 |
| QDR II+ SRAM | 350 | 44.8 | 38.1 |

**Note to Table 1–1:**

(1) 70% for DDR memories, 85% for QDR memories

You must also consider other memory attributes, including how much memory is required (density), how much latency can be tolerated, what is the power budget, and whether the system is cost sensitive. Table 1–2 is an overview of high-speed memories, and details some of the features and target markets of each technology.

**Table 1–2. Memory Selection Overview**

| Parameter | DDR3 SDRAM | DDR2 SDRAM | DDR SDRAM | RLDRAM II | QDR II/+ SRAM |
|---|---|---|---|---|---|
| Performance | 400–800 MHz | 200–400 MHz | 100–200 MHz | 200–533 MHz | 154–350 MHz |
| Altera-supported data rate | Up to 1066 Mbps | Up to 800 Mbps | Up to 400 Mbps | Up to 2132 Mbps | Up to 1400 Mbps |
| Density | 512 Mbytes–8 Gbytes, 32 Mbytes – 8 Gbytes (DIMM) | 256 Mbytes–1 Gbytes, 32 Mbytes – 4 Gbytes (DIMM) | 128 Mbytes–1 Gbytes, 32 Mbytes – 2 Gbytes (DIMM) | 288 Mbytes, 576 Mbytes | 8–72 Mbytes |
| I/O standard | SSTL-15 Class I, II | SSTL-18 Class I, II | SSTL-2 Class I, II | HSTL-1.8V/1.5V | HSTL-1.8V/1.5V |
| Data width (bits) | 4, 8, 16 | 4, 8, 16 | 4, 8, 16, 32 | 9, 18, 36 | 8, 9, 18, 36 |
| Burst length | 8 | 4, 8 | 2, 4, 8 | 2, 4, 8 | 2, 4 |
| Number of banks | 8 | 8 (>1 GB), 4 | 4 | 8 | N/A |
| Row/column access | Row before column | Row before column | Row before column | Row and column together or multiplexed option | N/A |
| CAS latency (CL) | 5, 6, 7, 8, 9, 10 | 3, 4, 5 | 2, 2.5, 3 | 4, 6, 8 | N/A |

**Table 1–2. Memory Selection Overview**

| Parameter | DDR3 SDRAM | DDR2 SDRAM | DDR SDRAM | RLDRAM II | QDR II/+ SRAM |
|---|---|---|---|---|---|
| Posted CAS additive latency (AL) | 0, CL-1, CL-2 | 0, 1, 2, 3, 4 | N/A | N/A | N/A |
| Read latency (RL) | RL = CL + AL | RL = CL + AL | RL = CL | RL = CL/CL + 1 | 1.5 clock cycles |
| On-die termination | Yes | Yes | No | Yes | Yes |
| Data strobe | Differential bidirectional strobe only | Differential or single-ended bidirectional strobe | Single-ended bidirectional strobe | Free-running differential read and write clocks | Free-running read and write clocks |
| Refresh requirement | Yes | Yes | Yes | Yes | No |
| Relative cost comparison | Presently lower than DDR2 | Less than DDR SDRAM with market acceptance | Low | Higher than DDR SDRAM, less than SRAM | Highest |
| Target market | Desktops, servers, storage, LCDs, displays, networking, and communication equipment | Desktops, servers, storage, LCDs, displays, networking, and communication equipment | Desktops, servers, storage, LCDs, displays, networking, and communication equipment | Main memory, cache memory, networking, packet processing, and traffic management | Cache memory, routers, ATM switches, packet memories, lookup, and classification memories |

Altera supports these memory interfaces, provides various IP for the physical interface and the controller, and offers many reference designs (refer to Altera's Memory Solutions Center).

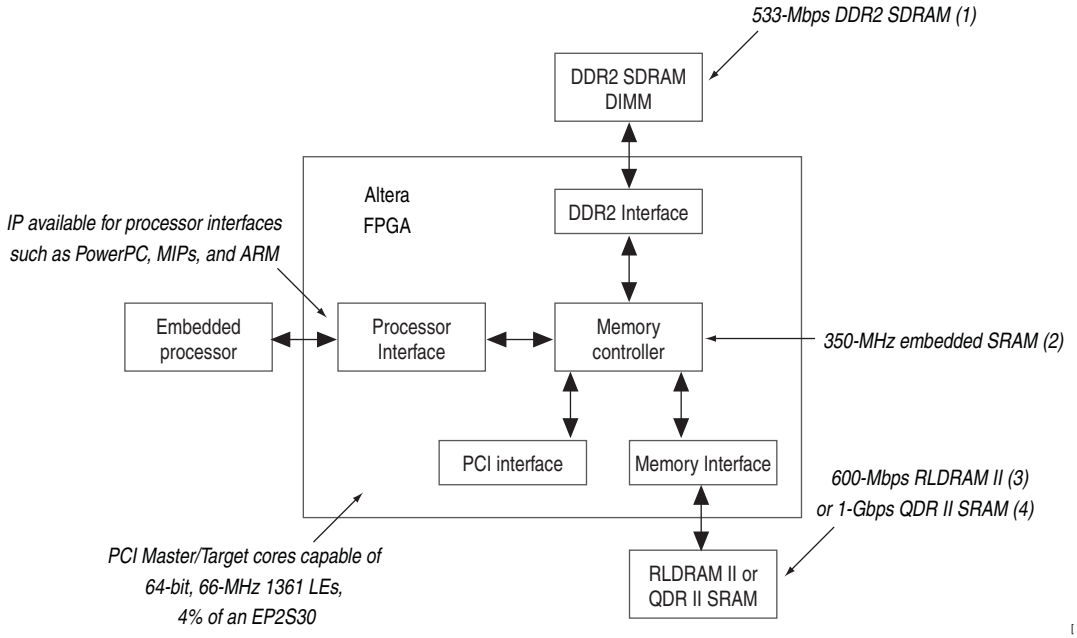
For Altera support and the maximum performance for the various high-speed memory interfaces, refer to the *External Memory Interface System Specifications* section in volume 1 of the *External Memory Interface Handbook.*

# High-Speed Memory in Embedded Processor Application Example

In embedded processor applications—any system that uses processors, excluding desktop processors—DDR SDRAM is typically used for main memory due to its very low cost, high density, and low power. Next-generation processors invest a large amount of die area to on-chip cache memory to prevent the execution pipelines from sitting idle. Unfortunately, these on-chip caches are limited in size, as a balance of performance, cost, and power must be taken into consideration. In many systems, external memories are used to add another level of cache. In high-performance systems, three levels of cache memory is common: level one (8 Kbytes is common) and level two (512 Kbytes) on chip, and level three off chip (2 Mbytes).

High-end servers, routers, and even video game systems are examples of high-performance embedded products that require memory architectures that are both high speed and low latency. Advanced memory controllers are required to manage transactions between embedded processors and their memories. Altera Arria® series and Stratix series FPGAs optimally implement advanced memory controllers by utilizing their built-in DQS (strobe) phase shift circuitry. Figure 1–1 highlights some of the features available in an Altera FPGA in an embedded application, where DDR2 SDRAM is used as the main memory and QDR II SRAM or RLDRAM II is an external cache level.

**Figure 1–1. Memory Controller Example Using FPGA**




**Notes to Figure 1–1:**

(1) 533-Mbps DDR2 SDRAM operation using dedicated DQS circuitry, post-amble circuitry, automatic phase shifting, and six registers in the I/O element: 790 LEs, 3% of an EP2S30, and four clock buffers (for a 72-bit interface).

(2) High-speed memory interfaces such as QDR II SRAM require at least four clock buffers to handle all the different clock phases and data directions.

(3) 600-Mbps RLDRAM II operation: 740 logic elements (LEs), 3% of an EP2S30, and four clock buffers (for a 36-bit wide interface).

(4) Embedded SRAM with features such as true-dual port and 350-MHz operation allows complex “store and forward” memory controller architectures.

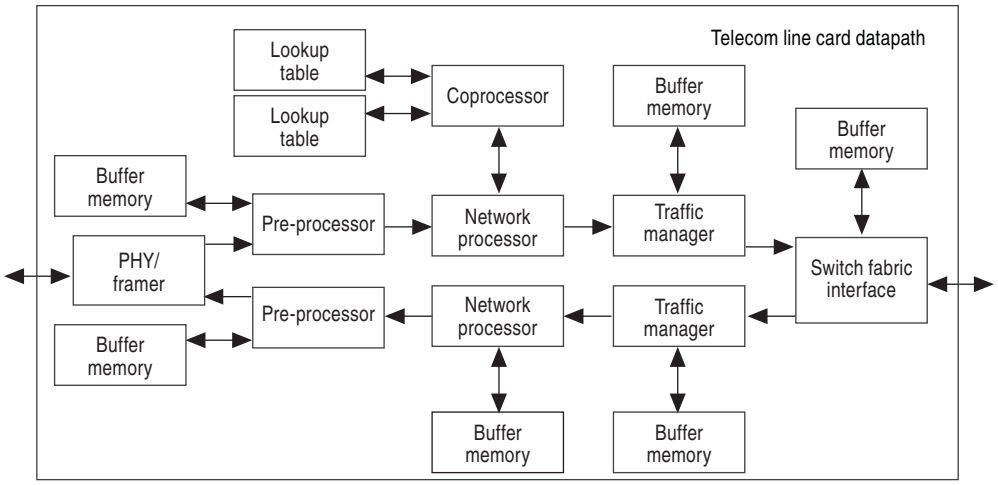
(5) The Quartus II software reports the number of adaptive look-up tables (ALUTs) that the design uses in the FPGA. The LE count is based on this number of ALUTs.

One of the target markets of RLDRAM II and QDR/QDR II SRAM is external cache memory. RLDRAM II has a read latency close to SSRAM, but with the density of SDRAM. A 16 times increase in external cache density is achievable with one RLDRAM II versus that of SSRAM. In contrast, consider QDR and QDR II SRAM for systems that require high bandwidth and minimal latency. Architecturally, the dual-port nature of QDR and QDR II SRAM allows cache controllers to handle read data and instruction fetches completely independent of writes.

# High-Speed Memory in Telecom Application Example

Because telecommunication network architectures are becoming more complex, high-end network systems are running multiple 10-Gbps line cards that connect to multi-shelf switch fabrics scaling to Terabits per second. Figure 1–2 shows an example of a typical system line interface card. These line cards offer interfaces ranging from a single-port OC-192 to multi-port Gigabit Ethernet, and consist of a number of devices, including a PHY/framer, network processors, traffic managers, fabric interface devices, and high-speed memories.

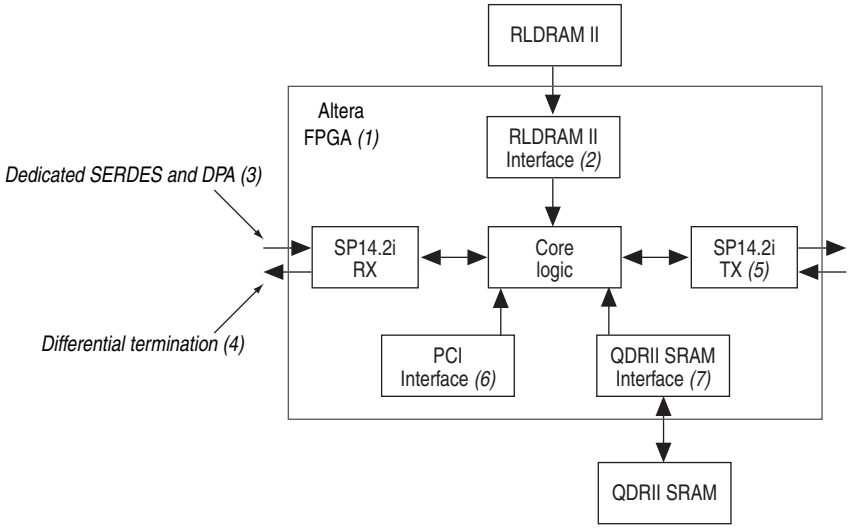
**Figure 1–2. Typical Telecom System Line Interface Card**




As packets traverse from the PHY/framer device to the switch fabric interface, they are buffered into memories, while the data path devices process headers (determining the destination, classifying packets, and storing statistics for billing) and control the flow of packets into the network to avoid congestion. Typically DDR/DDR2/DDR3 SDRAM and RLDRAM II are used for large buffer memories off network processors, traffic managers, and fabric interfaces, while QDR and QDR II SRAMs are used for look-up tables (LUTs) off preprocessors and coprocessors.

In many designs, FPGAs connect devices together for interoperability and coprocessing, implement features that are not supported by ASIC devices, or implement a device function entirely. Altera Stratix series FPGAs implement traffic management, packet processing, switch fabric interfaces, and coprocessor functions, using features such as 1 Gbps LVDS I/O, high-speed memory interface support, multi-gigabit transceivers, and IP cores. Figure 1–3 highlights some of these features in a packet buffering application where RLDRAM II is used for packet buffer memory and QDR II SRAM is used for control memory.

**Figure 1–3. FPGA Example in Packet Buffering Application**




**Notes to Figure 1–3:**

(1) As an example, 85% of the LEs still available in an EP2S90.
(2) 600-Mbps RLDRAM II operation: 740 LEs, 1% of an EP2S90, and four clock buffers (for a 36-bit wide interface).
(3) Dedicated hardware SERDES and DPA circuitry allows clean and reliable implementation of 1-Gbps LVDS.
(4) Differential termination is built in Stratix FPGAs, simplifying board layout and improving signal quality.
(5) SPI 4.2i core capable of 1 Gbps: 5178 LEs per Rx, 6087 LEs per Tx, 12% of an ES2S90, and four clock buffers (for both directions using individual buffer mode, 32-bit data path, and 10 logical ports).
(6) PCI cores capable of 64-bit 66-MHz 656 LEs, 1% of an EP2S90 for a 32-bit target
(7) 1-Gbps QDR II SRAM operation: 100 LEs, 0.1% of an EP2S90, and four clock buffers (for an 18-bit interface).
(8) Note that the Quartus II software reports the number of ALUTs that the design uses in Stratix II devices. The LE count is based on this number of ALUTs.

SDRAM is usually the best choice for buffering at high data rates due to the large amounts of memory required. Some system designers take a hybrid approach to the memory architecture, using SRAM to store the packet headers and DRAM to store the payload. The depth of the memories depends on the architecture and throughput of the system.

The buffer memory for the packet buffering application of an OC-192 line card (approximately 10 Gbps) must be able to sustain a minimum of one write and one read operation, which requires a memory bandwidth of 20 Gbps to operate at full line rate (more bandwidth is required if the headers are modified). The bandwidth requirement for memory is a key factor in memory selection (see Table 1–1). As an

example, a simple first-order calculation using RLDRAM II as buffer memory requires a bus width of 48 bits to sustain 20 Gbps (300 MHz × 2 DDR × 0.70 efficiency × 48 bits = 20.1 Gbps), which needs two RLDRAM II parts (one ×18 and one ×36). RLDRAM II also inherently includes the additional memory bits used for parity or error correction code (ECC).

QDR and QDR II SRAM have bandwidth and low random access latency advantages that make them useful for control memory in queue management and traffic management applications. Another typical implementation for this memory is billing and packet statistics, where each packet requires counters to be read from memory, incremented, and then rewritten to memory. The high bandwidth, low latency, and optimal one-to-one read/write ratio make QDR SRAM ideal for this feature.

# 2. DDR, DDR2, and DDR3 SDRAM Overview

This chapter provides an overview of DDR, DDR2, and DDR3 SDRAM in Altera devices. DDR3 SDRAM is the latest generation of DDR SDRAM technology, with improvements that include lower power consumption, higher data bandwidth, enhanced signal quality with multiple on-die termination (ODT) selection and output driver impedance control. DDR3 SDRAM brings higher memory performance to a broad range of applications, such as PCs, embedded processor systems, image processing, storage, communications, and networking. DDR2 SDRAM is the second generation of DDR SDRAM technology. DDR and DDR2 SDRAMs are available as components and modules, such as DIMMs, SODIMMs, and RDIMMs

For more information about Altera DDR3 SDRAM IP, refer to the *External Memory Interface System Specifications* section in volume 1 of the *External Memory Interface Handbook*.

## DDR SDRAM Overview

DDR SDRAM is a 2n prefetch architecture with two data transfers per clock cycle. It uses a single-ended strobe, DQS, which is associated with a group of data pins, DQ, for read and write operations. Both DQS and DQ ports are bidirectional. Address ports are shared for read and write operations.

Write and read operations are sent in bursts, DDR SDRAM supports burst lengths of 2, 4, and 8. The column address strobe (CAS) latency is the latency between when the read command is clocked into the memory and the requested data is presented at the memory pins. DDR SDRAM can have CAS latencies of 2, 2.5, and 3, depending on operating frequency.

DDR SDRAM devices use the SSTL-2 2.5V I/O standard and can hold between 64 MB and 1 GB of data. Each device is divided into four banks, and each bank has a fixed number of rows and columns. Only one row per bank can be accessed at a time. The `ACTIVE` command opens a row and the `PRECHARGE` command closes a row.

DDR SDRAM has a maximum frequency of 200 MHz or 400 Mbps per DQ pin.

## DDR2 SDRAM Overview

DDR2 SDRAM is the second generation of the DDR SDRAM standard. It is a 4n prefetch architecture (internally the memory operates at half the interface frequency) with two data transfers per clock cycle. DDR2 SDRAM can use a single-ended or differential strobe, DQS or DQSn, which is associated with a group of data pins, DQ, for read and write operations. DQS, DQSn, and DQ ports are bidirectional. Address ports are shared for read and write operations.

Write and read operations are sent in bursts, DDR2 SDRAM supports burst lengths of 4 and 8. DDR2 SDRAM supports CAS latencies of 2, 3, 4, and 5.

DDR2 SDRAM devices use the SSTL-18 1.8-V I/O standard and can hold between 256 MB and 4 GB of data. All DDR2 SDRAM devices have at least four banks, but higher-density devices (typically 1 GB and above) have eight internal banks. With more banks available, the page-to-hit ratio is twice as great when compared to DDR SDRAM. DDR2 SDRAM also allows bank interleaving, which represents a significant advantage for applications accessing random data. Bank interleaving can be extremely effective for concurrent operations and can hide the timing overhead that are otherwise required for opening and closing individual banks.

DDR2 SDRAM also supports ODT signal options of 50, 75, or 150 Ω on all DQ, DM, and DQS and DQSn signals.

DDR2 SDRAM has a maximum frequency of 533 MHz or 1,066 Mbps per DQ pin.

# DDR3 SDRAM Overview

DDR3 SDRAM is more effective at saving system power and increasing system performance than DDR2 SDRAM. DDR3 SDRAM offers lower power by using 1.5 V for the supply and I/O voltage compared to the 1.8-V supply and I/O voltage used by DDR2 SDRAM. DDR3 SDRAM also has better maximum throughput compared to DDR2 SDRAM by increasing the data rate per pin and the number of banks (to eight banks). DDR3 SDRAM also has the following additional benefits:

- Supports read/write levelling functionality required in the FPGA to interface with DDR3 DIMMs.
- Supports calibrated parallel ODT via an external resistor `RZQ` signal termination options of `RZQ`/2, `RZQ`/4, or `RZQ`/6 Ω on all `DQ`, `DM`, and `DQS` and `DQSn` signals.
- Supports controlled output driver impedance options of `RZQ`/6 or `RZQ`/7.
- Maximum frequency of 800 MHz or 1600 Mbps per `DQ` pin.
- Minimum operating frequency is 300 MHz.

The DDR3 SDRAM high-performance controller only supports local interfaces running at half the rate of the memory interface.

DDR3 SDRAMs are available as components and modules, such as DIMMs, SODIMMs, and RDIMMs.

DDR3 SDRAM is internally configured as an eight-bank DRAM. DDR3 SDRAM uses an 8n prefetch architecture to achieve high-speed operation. The 8n prefetch architecture is combined with an interface that transfers two data words per clock cycle at the I/O pins. A single read or write operation for DDR3 SDRAM consists of a single 8n-bit wide, four-clock data transfer at the internal DRAM core and two corresponding n-bit wide, one-half clock cycle data transfers at the I/O pins.

Read and write operations to the DDR3 SDRAM are burst oriented. Operation begins with the registration of an active command, which is then followed by a read or write command. The address bits registered coincident with the active command select the bank and row to be activated (BA0 to BA2 select the bank; A0 to A15 select the row). The address bits registered coincident with the read or write command select the

starting column location for the burst operation, determine if the auto precharge command is to be issued (via A10), and select burst chop (BC) of 4 or burst length (BL) of 8 mode at runtime (via A12), if enabled in the mode register. Before normal operation, the DDR3 SDRAM must be powered up and initialized in a predefined manner.

Differential strobes DQS and DQSn are mandated for DDR3 SDRAM and are associated with a group of data pins, DQ, for read and write operations. DQS, DQSn, and DQ ports are bidirectional. Address ports are shared for read and write operations.

Write and read operations are sent in bursts, DDR3 SDRAM supports BC of 4 and BL of 8. DDR3 SDRAM supports CAS latencies of 5 to 10.

DDR3 SDRAM devices use the SSTL-15 1.5-V I/O standard and can hold between 512 MB and 8 GB of data. The 1.5-V operational voltage reduces power consumption by 17% compared to DDR2 SDRAM.

All DDR3 SDRAM devices have eight internal banks. With more banks available, the page-to-hit ratio is twice that of DDR SDRAM. DDR3 SDRAM also allows bank interleaving, which represents a significant advantage for applications accessing random data. Bank interleaving can be extremely effective for concurrent operations and can hide the timing overhead that is otherwise required for opening and closing individual banks.

# DDR, DDR2 and DDR3 SDRAM Comparison

Table 2–1 compares DDR, DDR2, and DDR3 SDRAM features.

**Table 2–1. DDR, DDR2, and DDR3 SDRAM Features (Part 1 of 2)**

| Feature | DDR SDRAM | DDR2 SDRAM | DDR3 SDRAM | DDR3 SDRAM Advantage |
|---|---|---|---|---|
| Voltage | 2.5 V | 1.8 V | 1.5 V | Reduces memory system power demand by 17%. |
| Density | 64 MB to 1GB | 256 MB to 4 GB | 512 MB to 8 GB | High-density components simplify memory subsystem. |
| Internal banks | 4 | 4 and 8 | 8 | Page-to-hit ratio increased. |
| Prefetch | 2 | 4 | 8 | Lower memory core speed results in higher operating frequency and lower power operation. |
| Speed | 100 to 200 MHz | 200 to 533 MHz | 300 to 800 MHz | Higher data rate. |
| Read latency | 2, 2.5, 3 clocks | 3, 4, 5 clocks | 5, 6, 7, 8, 9, 10, and 11 | Eliminating half clock setting allows 8n prefetch architecture. |
| Additive latency *(1)* | — | 0, 1, 2, 3, 4 | 0, CL1, or CL2 | Improves command efficiency. |
| Write latency | One clock | Read latency – 1 | 5, 6, 7, or 8 | Improves command efficiency. |
| Termination | PCB, discrete to $V_{TT}$ | Discrete to $V_{TT}$ or ODT | Discrete to $V_{TT}$ or ODT parallel termination. Controlled impedance output. | Improves signaling, eases PCB layout, reduces system cost. |
| Data strobes | Single-ended | Differential or single-ended | Differential mandated | Improves timing margin. |

**Table 2–1. DDR, DDR2, and DDR3 SDRAM Features (Part 2 of 2)**

| Feature | DDR SDRAM | DDR2 SDRAM | DDR3 SDRAM | DDR3 SDRAM Advantage |
|---|---|---|---|---|
| Clock, address, and command (CAC) layout | Balanced tree | Balanced tree | Series or daisy chained | The DDR3 SDRAM read and write leveling feature allows for a much simplified PCB and DIMM layout. You can still optionally use the balanced tree topology by using the DDR3 without the leveling option. |

**Note to Table 2–1:**

(1) The Altera DDR and DDR2 SDRAM high-performance controllers do not support additive latency, but the high-performance controller II does.

# DDR, DDR2, and DDR3 SDRAM Interface Pins

This section describes the DDR, DDR2, and DDR3 SDRAM interface pins.

## Clock Signals

DDR, DDR2, and DDR3 SDRAM devices use CK and CK# signals to clock the address and command signals into the memory. Furthermore, the memory uses these clock signals to generate the DQS signal during a read through the DLL inside the memory. The SDRAM data sheet specifies the following timings:

- $t_{DQSCK}$ is the skew between the CK or CK# signals and the SDRAM-generated DQS signal
- $t_{DSH}$ is the DQS falling edge from CK rising edge hold time
- $t_{DSS}$ is the DQS falling edge from CK rising edge setup time
- $t_{DQSS}$ is the positive DQS latching edge to CK rising edge

These SDRAM have a write requirement ($t_{DQSS}$) that states the positive edge of the DQS signal on writes must be within ± 25% (± 90°) of the positive edge of the SDRAM clock input. Therefore, you should generate the CK and CK# signals using the DDR registers in the IOE to match with the DQS signal and reduce any variations across process, voltage, and temperature. The positive edge of the SDRAM clock, CK, is aligned with the DQS write to satisfy $t_{DQSS}$.

The Altera SDRAM high-performance controllers generate the CK and CK# signals using the DDR registers in the IOE with the DQS signal and reduce any variations across process, voltage, and temperature.

DDR3 SDRAM can use a daisy-chained CAC topology, the memory clock must arrive at each chip at a different time. To compensate for this flight-time skew between devices across a typical DIMM, write leveling must be employed.

## Data, Data Strobes, DM, and Optional ECC Signals

DDR SDRAM uses bidirectional single-ended data strobe (DQS); DDR3 SDRAM uses bidirectional differential data strobes. The DQSn pins in DDR2 SDRAM devices are optional but recommended for DDR2 SDRAM designs operating at more than 333 MHz. Differential DQS operation enables improved system timing due to reduced crosstalk and less simultaneous switching noise on the strobe output drivers. The DQ

pins are also bidirectional. Regardless of interface width, DDR SDRAM always operates in ×8 mode DQS groups. DQ pins in DDR2 and DDR3 SDRAM interfaces can operate in either ×4 or ×8 mode DQS groups, depending on your chosen memory device or DIMM, regardless of interface width. The ×4 and ×8 configurations use one pair of bidirectional data strobe signals, DQS and DQSn, to capture input data. However, two pairs of data strobes, UDQS and UDQS# (upper byte) and LDQS and LDQS# (lower byte), are required by the ×16 configuration devices. A group of DQ pins must remain associated with its respective DQS and DQSn pins.

The DQ signals are edge-aligned with the DQS signal during a read from the memory and are center-aligned with the DQS signal during a write to the memory. The memory controller shifts the DQ signals by –90° during a write operation to center align the DQ and DQS signals. ALTMEMPHY delays the DQS signal during a read, so that the DQ and DQS signals are center aligned at the capture register. Altera devices use a phase-locked loop (PLL) to center-align the DQS signal with respect to the DQ signals during writes and Altera devices (except Cyclone III devices) use dedicated DQS phase-shift circuitry to shift the incoming DQS signal during reads. Figure 2–1 shows an example where the DQS signal is shifted by 90° for a read from the DDR2 SDRAM.

**Figure 2–1. DQ and DQS Relationship During a DDR2 SDRAM Read in Burst-of-Four Mode**

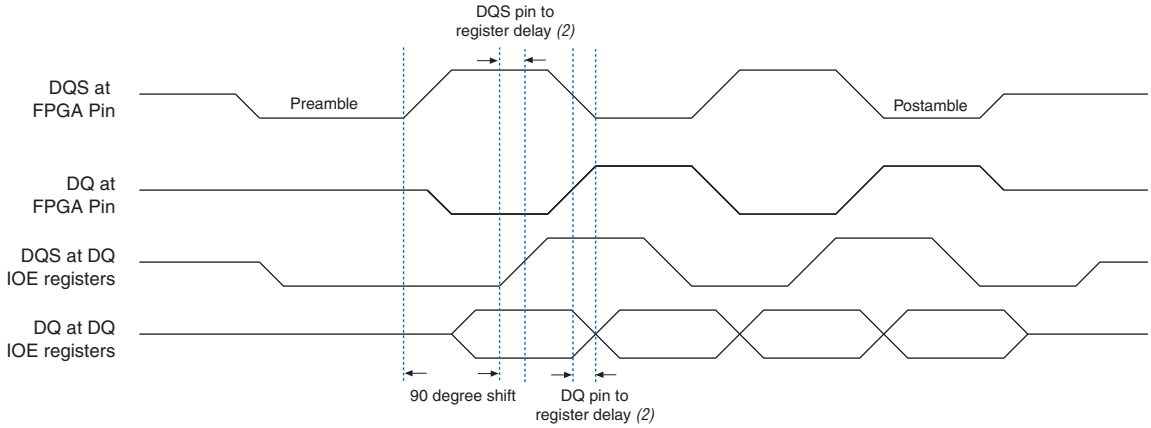
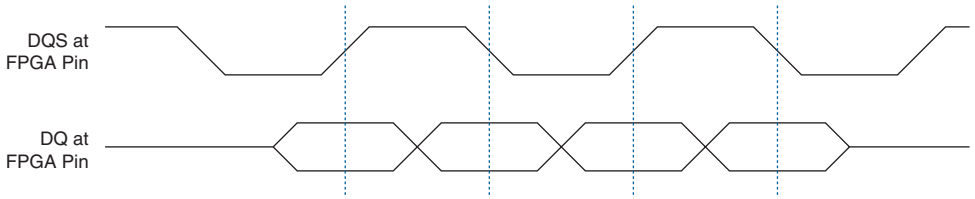



Figure 2–2 shows an example of the relationship between the data and data strobe during a burst-of-four write.

**Figure 2–2. DQ and DQS Relationship During a DDR2 SDRAM Write in Burst-of-Four Mode**




The memory device's setup ($t_{DS}$) and hold times ($t_{DH}$) for the write DQ and DM pins are relative to the edges of DQS write signals and not the CK or CK# clock. Setup and hold requirements are not necessarily balanced in in DDR2 and DDR3 SDRAM, unlike in DDR SDRAM devices.

The DQS signal is generated on the positive edge of the system clock to meet the $t_{DQSS}$ requirement. DQ and DM signals use a clock shifted –90° from the system clock, so that the DQS edges are centered on the DQ or DM signals when they arrive at the DDR2 SDRAM. The DQS, DQ, and DM board trace lengths need to be tightly matched (20 ps).

The SDRAM uses the DM pins during a write operation. Driving the DM pins low shows that the write is valid. The memory masks the DQ signals if the DM pins are driven high. While you can use any of the I/O pins in the same bank as the associated DQS and DQ pins, to generate the DM signal, Altera recommends that you use the spare DQ pin within the same DQS group as the respective data, to minimize skew.

The DM signal's timing requirements at the SDRAM input are identical to those for DQ data. The DDR registers, clocked by the –90° shifted clock, create the DM signals.

Some SDRAM modules support error correction coding (ECC) to allow the controller to detect and automatically correct error in data transmission. The 72-bit SDRAM modules contain eight extra data pins in addition to 64 data pins. The eight extra ECC pins should be connected to a single DQS or DQ group on the FPGA.

## Address and Command Signals

Address and command signals in SDRAM devices are clocked into the memory device using the CK or CK# signal. These pins operate at single data rate (SDR) using only one clock edge. The number of address pins depends on the SDRAM device capacity. The address pins are multiplexed, so two clock cycles are required to send the row, column, and bank address. The CS#, RAS, CAS, WE, CKE, and ODT pins are SDRAM command and control pins. DDR3 SDRAM has additional pins: `RESET#`, `PAR_In` and `ERR_OUT#`. The `RESET#` pin uses 1.5-V LVCMOS I/O standard, while the rest of the DDR3 SDRAM pins use the SSTL-15 I/O standard.

The DDR2 SDRAM address and command inputs do not have a symmetrical setup and hold time requirement with respect to the SDRAM clocks, CK, and CK#.

For ALTMEMPHY or Altera SDRAM high-performance controllers in Stratix III and Stratix IV devices, the address and command clock is a dedicated PLL clock output whose phase can be adjusted to meet the setup and hold requirements of the memory clock. The address and command clock is also typically half-rate, although a full-rate implementation can also be created. The command and address pins use the DDIO output circuitry to launch commands from either the rising or falling edges of the clock. The chip select (`mem_cs_n`), clock enable (`mem_cke`), and ODT (`mem_odt`) pins are only enabled for one memory clock cycle and can be launched from either the rising or falling edge of the address and command clock signal. The address and other command pins are enabled for two memory clock cycles and can also be launched from either the rising or falling edge of the address and command clock signal.

In ALTMEMPHY-based designs, the address and command clock `ac_clk_1x` is always half rate. However, because of the output enable assertion, CS#, CKE, and ODT behave like full-rate signals even in a half-rate PHY.

In Arria II GX and Cyclone III devices, the address and command clock is either shared with the `write_clk_2x` or the `mem_clk_2x` clock.

# DIMM Options

Compared to the unbuffered DIMMs (UDIMM), both single-rank and double-rank registered DIMMs (RDIMM) use only one pair of clocks and two chip selects CS#[1:0] in DDR3. An RDIMM has extra parity signals for address, RAS#, CAS#, and WE#.

Dual-rank DIMMs have the following extra signals for each side of the DIMM:

- CS# (RDIMM always has two chip selects, DDR3 uses a minimum of 2 chip selects, even on a single rank module)
- CK (only UDIMM)
- ODT signal
- CKE signal

Table 2–2 compares the UDIMM and RDIMM pin options.

**Table 2–2. UDIMM and RDIMM Pin Options**

| Pins | UDIMM Pins (Single Rank) | UDIMM Pins (Dual Rank) | RDIMM Pins (Single Rank) | RDIMM Pins (Dual Rank) |
|---|---|---|---|---|
| Data | 72 bit DQ[71:0] = {CB[7:0], DQ[63:0]} | 72 bit DQ[71:0] = {CB[7:0], DQ[63:0]} | 72 bit DQ[71:0] = {CB[7:0], DQ[63:0]} | 72 bit DQ[71:0] = {CB[7:0], DQ[63:0]} |
| Data Mask | DM[8:0] | DM[8.0] | DM[8.0] | DM[8.0] |
| Data Strobe *(1)* | DQS[8:0] and DQS#[8:0] | DQS[8:0] and DQS#[8:0] | DQS[8:0] and DQS#[8:0] | DQS[8:0] and DQS#[8:0] |
| Address | BA[2:0], A[15:0]— 2 GB: A[13:0] 4 GB: A[14:0] 8 GB: A[15:0] | BA[2:0], A[15:0]— 2 GB: A[13:0] 4 GB: A[14:0] 8 GB: A[15:0] | BA[2:0], A[15:0]— 2 GB: A[13:0] 4 GB: A[14:0] 8 GB: A[15:0] | BA[2:0], A[15:0]— 2 GB: A[13:0] 4 GB: A[14:0] 8 GB: A[15:0] |
| Clock | CK0/CK0# | CK0/CK0#, CK1/CK1# | CK0/CK0# | CK0/CK0# |
| Command | ODT, CS#, CKE, RAS#, CAS#, WE# | ODT[1:0], CS#[1:0], CKE[1:0], RAS#, CAS#, WE# | ODT, CS#[1:0], CKE, RAS#, CAS#, WE# | ODT[1:0], CS#[1:0], CKE[1:0], RAS#, CAS#, WE# |
| Parity | — | — | PAR_IN, ERR_OUT | PAR_IN, ERR_OUT |
| Other Pins | SA[2:0], SDA, SCL, EVENT#, RESET# | SA[2:0], SDA, SCL, EVENT#, RESET# | SA[2:0], SDA, SCL, EVENT#, RESET# | SA[2:0], SDA, SCL, EVENT#, RESET# |

**Note to Table 2–2:**

(1) DQS#[8:0] is optional in DDR2 SDRAM and is not supported in DDR SDRAM interfaces.

# 3. QDR II and QDR II+ SRAM Overview

This chapter provides an overview of QDR II and QDR II+ SRAM in Altera devices

Synchronous static RAM (SRAM) architectures support the high throughput requirements of communications, networking, and digital signal processing (DSP) systems. The successor to quad data rate (QDR) SRAM, QDR II+ and QDR II SRAM support higher memory bandwidth and improved timing margins and offer more flexibility in system designs.

For more information about Altera QDR II and QDR II+ SRAM IP, refer to the *External Memory Interface System Specifications* section in volume 1 of the *External Memory Interface Handbook*.

QDR II+ and QDR II SRAM can perform two data writes and two data reads per clock cycle. They use one port for writing data (`D`) and another port for reading data (`Q`). These unidirectional data ports support simultaneous reads and writes and allows back-to-back transactions without the contention issues that can occur when using a single bidirectional data bus. Write and read operations share address ports.

The QDR II SRAM devices are available in ×8, ×9, ×18, and ×36 data bus width configurations. The QDR II+ SRAM devices are available in ×9, ×18, and ×36 data bus width configurations. Write and read operations are burst-oriented. All the data bus width configurations of QDR II SRAM support burst lengths of two and four. QDR II+ SRAM supports only a burst length of four. Burst-of-two and burst-of-four for QDR II and burst-of-four for QDR II+ SRAM devices provide the same overall bandwidth at a given clock speed.

Read latency is the time between the read command being clocked into memory and the time data is presented at the memory pins. For QDR II SRAM devices, the read latency is 1.5 clock cycles, while for QDR II+ SRAM devices it is 2 or 2.5 clock cycles, depending on the memory device. Write latency is the time between the write command being clocked into memory and the time data is presented at the memory pins. For QDR II+ and burst-of-four QDR II SRAM devices, the write commands and addresses are clocked on the rising edge of clock and write latency is one clock cycle. For burst-of-two QDR II SRAM devices, the write command is clocked on the rising edge of clock and the write address is clocked on the falling edge of clock. Therefore, the write latency is zero, because the write data is presented at the same time as the write command.

Altera supports both 1.5-V and 1.8-V HSTL I/O standards for QDR II+ and QDR II SRAM interfaces. QDR II+ and QDR II SRAM interfaces use a delay-locked loop (DLL) inside the device to edge-align the data with respect to the `K` and `Kn` or `C` and `Cn` pins. You can optionally turn off the DLL, but the performance of the QDR II+ and QDR II SRAM devices is degraded. All timing specifications listed in this document assume that the DLL is on.

QDR II+ and QDR II SRAM devices also offer programmable impedance output buffers. You can set the buffers by terminating the `ZQ` pin to $V_{SS}$ through a resistor, `RQ`. The value of `RQ` should be five times the desired output impedance. The range for `RQ` should be between 175 Ω and 350 Ω with a tolerance of 10%.

# QDR II+ and QDR II SRAM Interface Pin Description

This section provides a description of the clock, control, address, and the data signals on QDR II and QDR II+ SRAM devices.

## Clock Signals

QDR II+ and QDR II SRAM devices have three pairs of clocks:

- Input clocks K and K#
- Input clocks C and C#
- Echo clocks CQ and CQ#

The positive input clock, K, is the logical complement of the negative input clock, K#. Similarly, C and CQ are complements of C# and CQ#, respectively. With these complementary clocks, the rising edges of each clock leg latch the DDR data.

The QDR II+ and QDR II SRAM devices use the K and K# clocks for write access and the C and C# clocks for read accesses only when interfacing more than one QDR II+ or QDR II SRAM device. Because the number of loads that the K and K# clocks drive affects the switching times of these outputs when a controller drives a single QDR II+ or QDR II SRAM device, C and C# are unnecessary. This is because the propagation delays from the controller to the QDR II+ or QDR II SRAM device and back are the same. Therefore, to reduce the number of loads on the clock traces, QDR II+ and QDR II SRAM devices have a single clock mode, and the K and K# clocks are used for both reads and writes. In this mode, the C and C# clocks are tied to the supply voltage ($V_{DD}$).

CQ and CQ# are the source-synchronous output clocks from the QDR II or QDR II+ SRAM device that accompanies the read data.

The Altera device outputs the K and K# clocks, data, address, and command lines to the QDR II+ or QDR II SRAM device. For the controller to operate properly, the write data (D), address (A), and control signal trace lengths (and therefore the propagation times) should be equal to the K and K# clock trace lengths.

You can generate C, C#, K, and K# clocks using any of the PLL registers via the DDR registers. Because of strict skew requirements between K and K# signals, use adjacent pins to generate the clock pair. The propagation delays for K and K# from the FPGA to the QDR II+ or QDR II SRAM device are equal to the delays on the data and address (D, A) signals. Therefore, the signal skew effect on the write and read request operations is minimized by using identical DDR output circuits to generate clock and data inputs to the memory.

## Command Signals

QDR II+ and QDR II SRAM devices use the write port select (WPSn) signal to control write operations and the read port select (RPSn) signal to control read operations. The byte write select signal (BWSn) is a third control signal that indicates to the QDR II+ or QDR II SRAM device which byte to write into the QDR II+ or QDR II SRAM device. You can use any of the FPGA's user I/O pins to generate control signals, preferably on the same side and the same bank.

## Address Signals

QDR II+ and QDR II SRAM devices use one address bus (A) for both read and write addresses. You can use any of the FPGA's user I/O pins to generate address signals, preferably on the same side and the same banks.

## Data and QVLD Signals

QDR II+ and QDR II SRAM devices use two unidirectional data buses: one for writes (D) and one for reads (Q). The read data is edge-aligned with the CQ and CQ# clocks while the write data is center-aligned with the K and K# clocks (see Figure 3–1 and Figure 3–2).

**Figure 3–1. CQ and Q Relationship During QDR II+ SRAM Read**

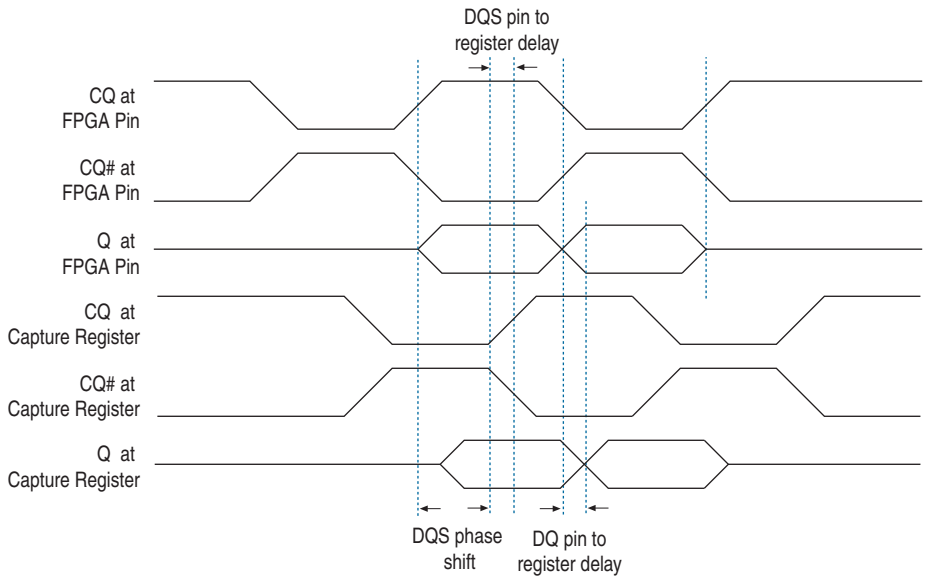
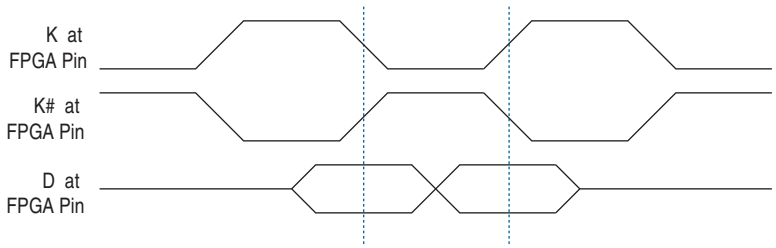



**Figure 3–2. K and D Relationship During QDR II+ SRAM Write**




QDR II+ SRAM devices also have a QVLD pin that indicates valid read data. The QVLD signal is edge-aligned with the echo clock and is asserted high for approximately half a clock cycle before data is output from memory.

The Altera QDR II+ SRAM Controller MegaCore function does not use the QVLD signal.

# 4. RLDRAM II Overview

This chapter provides an overview of RLDRAM II in Altera devices

Reduced latency DRAM II (RLDRAM II) is a DRAM-based point-to-point memory device designed for communications, imaging, and server systems requiring high density, high memory bandwidth, and low latency. The fast random access speeds in RLDRAM II devices make them a viable alternative to SRAM devices at a lower cost.

For more information about Altera RLDRAM II IP, refer to the *External Memory Interface System Specifications* section in volume 1 of the *External Memory Interface Handbook*.

There are two types of RLDRAM II devices: common I/O (CIO) and separate I/O (SIO). CIO devices share a single data I/O bus which is similar to the double data rate (DDR) SDRAM interface. SIO devices, with separate data read and write buses, have an interface similar to SRAM.

Compared to DDR SDRAM, RLDRAM II has simpler bank management and lower latency inside the memory. RLDRAM II devices are divided into eight banks instead of the typical four banks in most memory devices, providing a more efficient data flow within the device. RLDRAM II offers up to 2.4 Gigabytes per second (Gbps) aggregate bandwidth.

RLDRAM II uses a DDR scheme, performing two data transfers per clock cycle. RLDRAM II CIO devices use the bidirectional data pins (DQ) for both read and write data, while RLDRAM II SIO devices use D pins for write data (input to the memory) and Q pins for read data (output from the memory). Both types use two pairs of uni-directional free-running clocks. The memory uses DK and DK# pins during write operations, and generates QK and QK# pins during read operations. In addition, RLDRAM II uses the system clocks (CK and CK# pins) to sample commands and addresses and generate the QK and QK# read clocks. Address ports are shared for write and read operations.

The RLDRAM II SIO devices are available in ×9 and ×18 data bus width configurations, while the RLDRAM II CIO devices are available in ×9, ×18, and ×36 data bus width configurations. RLDRAM II CIO interfaces may require an extra cycle for bus turnaround time for switching read and write operations.

Write and read operations are burst oriented and all the data bus width configurations of RLDRAM II support burst lengths of two and four. In addition, RLDRAM II devices with data bus width configurations of ×9 and ×18 also support burst length of eight.

The read latency is the time between when the read command is clocked into the memory and the time data is presented at the memory pins. There is a similar latency for write operations called the write latency. The write latency is equal to the read latency plus one clock cycle. The RLDRAM devices have up to three programmable configuration settings that determine the row cycle times, read latency, and write latency of the interface at a given frequency of operation.

RLDRAM II devices use either the 1.5-V HSTL or 1.8-V HSTL I/O standard. You can use either I/O standard to interface with Altera FPGAs. Each RLDRAM II device is divided into eight banks, where each bank has a fixed number of rows and columns. Only one row per bank is accessed at a time. The memory (instead of the controller) controls the opening and closing of a row, which is similar to an SRAM interface.

RLDRAM II also offers programmable impedance output buffers and on-die termination. The programmable impedance output buffers are for impedance matching and are guaranteed to produce 25- to 60-ohm output impedance. The on-die termination is dynamically switched on during read operations and switched off during write operations. Perform an IBIS simulation to observe the effects of this dynamic termination on your system. IBIS simulation can also show the effects of different drive strengths, termination resistors, and capacitive loads on your system.

# RLDRAM II Interface Pin Description

This section describes the RLDRAM II interface pin description.

## Clock Signals

RLDRAM II devices use CK and CK# signals to clock the command and address bus in single data rate (SDR). There is one pair of CK and CK# pins per RLDRAM II device.

Instead of a strobe, RLDRAM II devices use two sets of free-running differential clocks to accompany the data. The DK and DK# clocks are the differential input data clocks used during writes while the QK or QK# clocks are the output data clocks used during reads. Even though QK and QK# signals are not differential signals according to the RLDRAM II data sheets, Micron treats these signals as such for their testing and characterization. Each pair of DK and DK#, or QK and QK# clocks are associated with either 9 or 18 data bits.

The exact clock-data relationships are as follows:

- For ×36 data bus width configuration, there are 18 data bits associated with each pair of write and read clocks. So, there are two pairs of DK and DK# pins and two pairs of QK or QK# pins.
- For ×18 data bus width configuration, there are 18 data bits per one pair of write clocks and nine data bits per one pair of read clocks. So, there is one pair of DK and DK# pins, but there are two pairs of QK and QK# pins.
- For ×9 data bus width configuration, there are nine data bits associated with each pair of write and read clocks. So, there is one pair of DK and DK# pins and one pair of QK and QK# pins each.

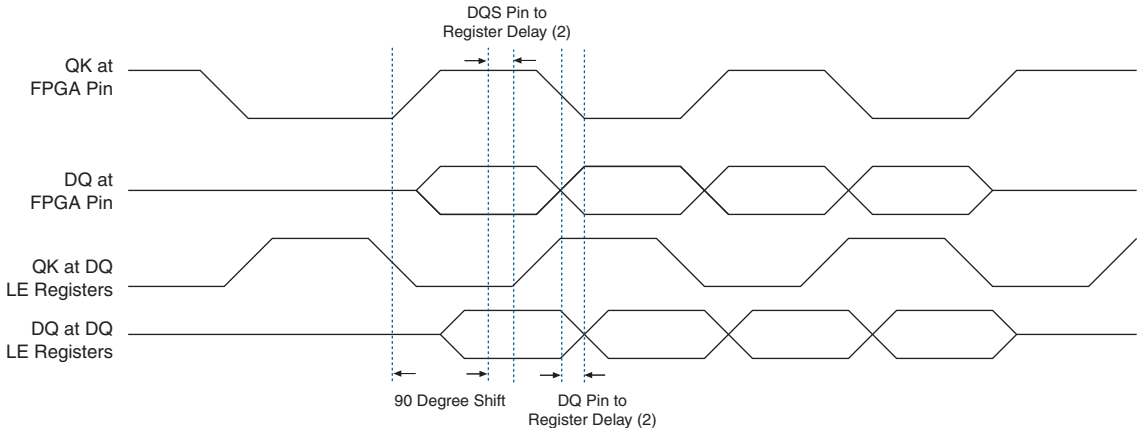
There are $t_{CKDK}$ timing requirements for skew between CK and DK or CK# and DK#.

Because of the loads on these I/O pins, the maximum frequency you can achieve depends on the number of RLDRAM II devices you are connecting to the Altera device. Perform SPICE or IBIS simulations to analyze the loading effects of the pin-pair on multiple RLDRAM II devices.

## Data, DM and QVLD Signals

The read data is edge-aligned with the QK or QK# clocks while the write data is center-aligned with the DK and DK# clocks (see Figure 4–1 and Figure 4–2). The memory controller shifts the DK or DK# signal to center align the DQ and DK or DK# signal during a write and to shift the QK signal during a read, so that read data (DQ or Q signals) and QK clock is center-aligned at the capture register. Altera devices use dedicated DQS phase-shift circuitry to shift the incoming QK signal during reads and use a PLL to center-align the DK and DK# signals with respect to the DQ signals during writes.

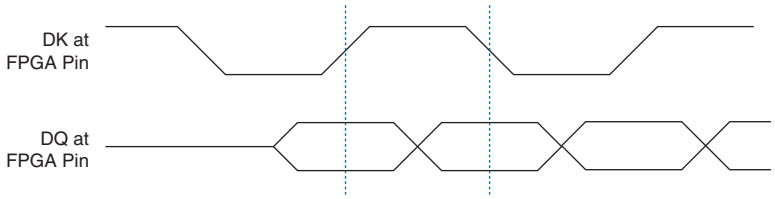
**Figure 4–1. DQ and QK Relationship During RLDRAM II Read**




**Notes to Figure 4–1:**

(1) This is an example of a 90° shift. The required phase shift for your system should be based on your timing analysis and may not be 90°.

**Figure 4–2. DQ and QK Relationship During RLDRAM II Write**




The RLDRAM II data mask (DM) pins are only used during a write. The memory controller drives the DM signal low when the write is valid and drives it high to mask the DQ signals. There is one DM pin per RLDRAM II device.

The DM timing requirements at the input to the RLDRAM II are identical to those for DQ data. The DDR registers, clocked by the write clock, create the DM signals. This reduces any skew between the DQ and DM signals.

The RLDRAM II device's setup time ($t_{DS}$) and hold ($t_{DH}$) time for the write DQ and DM pins are relative to the edges of the DK or DK# clocks. The DK and DK# signals are generated on the positive edge of system clock, so that the positive edge of CK or CK# is aligned with the positive edge of DK or DK# respectively to meet the RLDRAM II tCKDK requirement. The DQ and DM signals are clocked using a shifted clock so that the edges of DK or DK# are center-aligned with respect to the DQ and DM signals when they arrive at the RLDRAM II device.

The clocks, data, and DM board trace lengths should be tightly matched to minimize the skew in the arrival time of these signals.

RLDRAM II devices also have a QVLD pin indicating valid read data. The QVLD signal is edge-aligned with QK or QK# and is high approximately half a clock cycle before data is output from the memory.

The Altera RLDRAM II Controller IP does not use the QVLD signal.

## Commands and Addresses

The CK and CK# signals clock the commands and addresses into RLDRAM II devices. These pins operate at single data rate using only one clock edge. RLDRAM II devices have 18 to 21 address pins, depending on the data bus width configuration and burst length. RLDRAM II supports both non-multiplexed and multiplexed addressing. Multiplexed addressing allows you to save a few user I/O pins while non-multiplexed addressing allows you to send the address signal within one clock cycle instead of two clock cycles. CS#, REF#, and WE# pins are input commands to the RLDRAM II device.

The commands and addresses must meet the memory address and command setup ($t_{AS}$, $t_{CS}$) and hold ($t_{AH}$, $t_{CH}$) time requirements.

# External Memory Interface Handbook Volume 1

## Section III. System Performance Specifications

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

EMI_INTRO_SPECS-2.1

Document last updated for Altera Complete Design Suite version: 10.1
Document publication date: December 2010

Subscribe

# Contents

# 1. Maximum Clock Rates

This chapter describes the maximum clock rates supported when using the following Altera memory controllers:

- DDR SDRAM Controller
- DDR2 SDRAM Controller
- DDR3 SDRAM Controller
- QDR II and QDR II+ SRAM Controller
- RLDRAM II Controller

Specifications for HardCopy® III, HardCopy IV, and Stratix® V devices are preliminary and subject to change.

For the complete set of maximum clock rates support including row and wraparound I/Os and multiple chip-selects on Altera FPGA devices, refer to the External Memory Interface Specification Estimator tool.

## DDR SDRAM Controller

For a list of supported and unsupported features, refer to *DDR and DDR2 SDRAM Controllers with ALTMEMPHY IP User Guide*.

Table 1–1 shows the maximum clock rates supported on column I/Os on Altera FPGA devices with fastest speed grades for single chip-select DDR SDRAM Controller with ALTMEMPHY.

Table 1–1 also shows the maximum clock rates supported on column and row I/Os on HardCopy devices for single chip-select DDR SDRAM Controller with ALTMEMPHY. Wrap around clock rates are lower; to determine the maximum achievable clock rate for a wrap around interface, you must compile your design and perform timing analysis in the Quartus II software. The maximum clock rates on column and row I/Os for dual and quad chip-select interfaces are the same as -3 speed grade clock rates of the corresponding FPGA device.

For FPGA device specifications, refer to the External Memory Interface Specification Estimator tool.

**Table 1–1. DDR SDRAM Interface Maximum Clock Rate Estimates on Altera Devices (Part 1 of 2)** ***(Note 1)***

| Device | Speed Grade | Maximum Half-Rate Clock Rate (MHz) *(2)* | Maximum Full-Rate Clock Rate (MHz) *(2)* |
|---|---|---|---|
| | | Column I/Os | |
| Arria® II GX | C4<br>I3 | 200 | 200 |

**Table 1–1. DDR SDRAM Interface Maximum Clock Rate Estimates on Altera Devices (Part 2 of 2) *(Note 1)***

| Device | | Speed Grade | Maximum Half-Rate Clock Rate (MHz) *(2)* | Maximum Full-Rate Clock Rate (MHz) *(2)* |
|---|---|---|---|---|
| | | | **Column I/Os** | |
| Cyclone III, Cyclone IV GX Cyclone IV E ($V_{CC}$ = 1.2 V) | | C6 *(3)* | 167 | 167 |
| | | I7 | 150 | 150 |
| Cyclone IV E ($V_{CC}$ = 1.0 V) | | C8L | 133 *(4)* | 133 *(4)* |
| | | I8L | | |
| Stratix III | | C2 | 200 | 200 |
| | | I3 | | |
| | | | | |
| | | C4L (Vcc=0.9V) I4L (Vcc=0.9V) | | |
| Stratix IV | | C2 C2x | 200 | |
| | | I3 | | |
| Stratix IV GT | | I1 | 200 | 200 |
| | | | **Column and Row I/Os** | |
| HardCopy III HardCopy IV E | FF Flip Chip | C | 200 | 200 |
| | | I | 200 | 200 |
| | LF Flip Chip | C | 200 | 200 |
| | | I | 200 | 200 |
| | Wirebound *(4)* | C | 150 | 150 |
| | | I | 150 | 150 |
| HardCopy IV GX | FF Flip Chip | C | 200 | 200 |
| | | I | 200 | 200 |
| | LF Flip Chip | C | 200 | 200 |
| | | I | 200 | 200 |

**Notes to Table 1–1:**

(1) DDR SDRAM IP is not supported in Stratix V devices.

(2) The maximum clock rates listed are guidelines based on Altera designs. The actual achievable performance of your design is based on your system's features and must be attained through the Quartus II timing analysis of the complete design.

(3) Cyclone III LS devices do not support C6 speed grades. These frequencies are only applicable to Cyclone III non LS devices.

(4) To achieve maximum clock rate, use 200-MHz speed grade memory devices.

# DDR2 SDRAM Controller

For a list of supported and unsupported features in the DDR2 SDRAM Controller with ALTMEMPHY, refer to the *DDR and DDR2 SDRAM Controllers with ALTMEMPHY IP User Guide*. For a list of supported and unsupported features in the DDR2 SDRAM Controller with UniPHY, refer to the *DDR2 and DDR3 SDRAM Controller with UniPHY User Guide*.

Table 1–2 shows the maximum clock rates supported on column I/Os on Altera FPGA devices with the fastest speed grades for single chip-select DDR2 SDRAM Controller with ALTMEMPHY IP and UniPHY.

Table 1–2 also shows the maximum clock rates supported on column and row I/Os on HardCopy devices for single chip-select DDR2 SDRAM Controller with ALTMEMPHY and UniPHY. Wrap around clock rates are lower; to determine the maximum achievable clock rate for a wrap around interface, you must compile your design and perform timing analysis in the Quartus II software. The maximum clock rates on column and row I/Os for dual and quad chip-select interfaces are the same as -3 speed grade clock rates of the corresponding FPGA device.

For FPGA device specifications, refer to the External Memory Interface Specification Estimator tool.

**Table 1–2. DDR2 SDRAM Interface Maximum Clock Rate Estimates on Altera Devices (Part 1 of 2)**

| Device | Speed Grade | Maximum Half-Rate Clock Rate (MHz) *(1)* | Maximum Full-Rate Clock Rate (MHz) *(1)* |
|---|---|---|---|
| | | Column I/Os | |
| Arria II GX | C4 | 333 | 267 *(3)* |
| | I3 | 333 *(2)* | 233 *(3)* |
| Arria II GZ | C3 | 333 | 267 |
| | I3 | 333 | 267 |
| Cyclone III, Cyclone IV GX *(4)*, and Cyclone IV E ($V_{CC}$ = 1.2 V) | C6 *(5)* | 200 *(6)* | 167 *(6)* |
| | I7 | 167 *(6)* | 150 *(6)* |
| Cyclone IV E ($V_{CC}$ = 1.0 V) | C8L | 167 *(8)* | 150 *(7)* |
| | I8L | 150 *(9)* | 150 *(9)* |
| Stratix III | C2 | 400 | 300 *(10)* |
| | I3 | 333 | 267 *(10)* |
| Stratix III ($V_{CC}$ = 0.9 V) | C4L I4L | 200 | 167 |
| Stratix IV and Stratix V | C2 C2x | 400 | 300 *(10)* |
| | I3 | 333 | 267 *(10)* |
| Stratix IV GT | I1 | 400 | 300 *(10)* |

**Table 1–2. DDR2 SDRAM Interface Maximum Clock Rate Estimates on Altera Devices (Part 2 of 2)**

| Device | | Speed Grade | Maximum Half-Rate Clock Rate (MHz) *(1)* | Maximum Full-Rate Clock Rate (MHz) *(1)* |
|---|---|---|---|---|
| | | | Column and Row I/Os | |
| HardCopy III HardCopy IV E | FF Flip Chip *(2)* | C | 333 | 233 |
| | | I | 333 | 233 |
| | LF Flip Chip *(2)* | C | 333 | 233 |
| | | I | 267 | 233 |
| | Wirebound | C | 200 | 150 *(7)* |
| | | I | 167 *(7)* | 150 *(7)* |
| HardCopy IV GX | FF Flip Chip *(2)* | C | 333 | 233 |
| | | I | 333 | 233 |
| | LF Flip Chip *(2)* | C | 333 | 233 |
| | | I | 267 | 233 |

**Notes to Table 1–2:**

(1) The maximum clock rates listed here are guidelines based Altera designs. The actual achievable performance of your design is based on your system's features and must be attained through the Quartus II timing analysis of the complete design.

(2) To achieve the maximum clock rates, use 400-MHz memory device speed grade.

(3) To achieve the maximum clock rates of 267 MHz and 233 MHz, use 333-MHz and 267-MHz memory device speed grades respectively.

(4) In Cyclone IV GX devices, left side is not supported for external memory interface.

(5) Cyclone III LS devices do not support C6 speed grades. These frequencies are only applicable to Cyclone III non LS devices.

(6) You need 267-MHz memory device speed grade when using class I I/O standard and 333-MHz memory device speed grade when using Class II I/O standard.

(7) To achieve the maximum clock rates, use 200-MHz memory device speed grade.

(8) To achieve the maximum clock rates, use 333-MHz memory device speed grade.

(9) To achieve the maximum clock rates, use 267-MHz memory device speed grade.

(10) May require some optimization to meet core timing.

# DDR3 SDRAM Controller

For a list of supported and unsupported features in the DDR3 SDRAM Controller with ALTMEMPHY, refer to the *DDR3 SDRAM Controller with ALTMEMPHY IP User Guide*. For a list of supported and unsupported features in the DDR3 SDRAM Controller with UniPHY, refer to the *DDR2 and DDR3 SDRAM Controller with UniPHY User Guide*.

Table 1–3 shows the maximum clock rates supported on column I/Os on Altera FPGA devices with the fastest speed grades for single chip-select DDR3 SDRAM Controller with ALTMEMPHY IP and UniPHY.

Table 1–3 also shows the maximum clock rates supported on column and row I/Os on HardCopy devices for single chip-select DDR3 SDRAM Controller with ALTMEMPHY and UniPHY. Wrap around clock rates are lower; to determine the maximum achievable clock rate for a wrap around interface, you must compile your design and perform timing analysis in the Quartus II software. The maximum clock rates on column and row I/Os for dual and quad chip-select interfaces are the same as -3 speed grade clock rates of the corresponding FPGA device.

For FPGA device specifications, refer to the External Memory Interface Specification Estimator tool.

**Table 1–3. DDR3 SDRAM Interface Maximum Clock Rate Estimates on Altera Devices (Part 1 of 2)**

| Device | | Speed Grade | Maximum Half-Rate Clock Rate (MHz) *(1)*, *(2)* |
|---|---|---|---|
| | | | **Column I/Os** |
| Arria II GX *(3)* | | C4 | 400 *(4)*, *(5)* |
| | | I3 | 333 *(6)* |
| Arria II GZ | | C3 | 400 |
| | | I3 | 400 |
| Stratix III | | C2 | 533 |
| | | I3 | 400 |
| Stratix IV and Stratix V | | C2<br>C2x | 533 |
| | | I3 | 400 |
| Stratix IV GT | | I1 | 533 |
| | | | **Column and Row I/Os** |
| HardCopy III *(5)*<br>HardCopy IV E *(5)* | FF Flip Chip | C | 400 |
| | | I | 333 |
| | LF Flip Chip | C | 333 |
| | | I | 300 |
| | Wirebound | C | — |
| | | I | — |

**Table 1–3. DDR3 SDRAM Interface Maximum Clock Rate Estimates on Altera Devices (Part 2 of 2)**

| Device | | Speed Grade | Maximum Half-Rate Clock Rate (MHz) *(1)*, *(2)* |
|---|---|---|---|
| | | | Column and Row I/Os |
| HardCopy IV GX *(5)* | FF Flip Chip | C | 533 *(7) (8)* |
| | | I | 450 *(7)* |
| | LF Flip Chip | C | 400 |
| | | I | 333 |

**Notes to Table 1–3:**

(1) The maximum clock rates listed here are guidelines based on Altera designs. The actual achievable performance of your design is based on your system's features and must be attained through the Quartus II timing analysis of the complete design.

(2) For DDR3 SDRAM Controller with ALTMEMPHY, any DDR3 SDRAM interfaces operating higher than 400 MHz must use the leveling circuitry, and the memory components must be laid out like a (fly-by) DDR3 SDRAM UDIMM. The leveling circuitry is enabled by setting the **Memory Format** to **Unbuffered DIMM**. For DDR3 SDRAM Controller with UniPHY, Altera recommends that you use leveling circuitry for any DDR3 SDRAM interface.

(3) No DIMM support for Arria II GX devices.

(4) To achieve the maximum frequency, you require a low board skew within a DQS or DQ group.

(5) To achieve the stated maximum clock rate, use 533-MHz memory device speed grade.

(6) To achieve the stated maximum clock rate, use 400-MHz memory device speed grade.

(7) Stated clock rates are valid for UniPHY only; maximum achievable clock rate for ALTMEMPHY is 400 MHz.

(8) For 533-MHz clock rate, contact Altera.

# QDR II and QDR II+ SRAM Controller

For a list of supported and unsupported features in the QDR II and QDR II+ SRAM Controller with UniPHY, refer to the *QDR II and QDR II+ SRAM Controller with UniPHY IP User Guide*.

Table 1–4 shows the maximum clock rates supported on column I/Os for QDR II SRAM controllers with UniPHY on various Altera devices. Table 1–5 shows the maximum clock rates supported on column I/Os for QDR II+ SRAM controllers with UniPHY on various Altera devices with the fastest speed grade.

Table 1–4 and Table 1–5 also show the maximum clock rates supported on column and row I/Os on HardCopy devices for single chip-select QDR II and QDR II+ SRAM Controller with UniPHY. Wrap around clock rates are lower; to determine the maximum achievable clock rate for a wrap around interface, you must compile your design and perform timing analysis in the Quartus II software.

**Table 1–4. QDR II SRAM Interface Maximum Clock Rate Estimates on Altera Devices** *(Note 1)*

| Device | | Speed Grade | Maximum Half-Rate Clock Rate (MHz) *(2)* | | Maximum Full-Rate Clock Rate (MHz) *(2)* | |
|---|---|---|---|---|---|---|
| | | | ×9, ×18, and ×36 | ×36 Emulation | ×9, ×18, and ×36 | ×36 Emulation |
| Arria II GX | | C4<br>I3 | 250 | 200 | 250 | 200 |
| Arria II GZ | | C3 | 300 | 250 | 300 | 250 |
| | | I3 | | | | |
| Stratix III | | C2 | 350 | 300 | 300 | |
| | | I3 | 300 | 250 | 300 | 250 |
| Stratix III ($V_{CC}$ = 0.9 V) | | C4L<br>I4L | 250 | 200 | 167 | 167 |
| Stratix IV | | C2<br>C2x | 350 | 300 | 300 | 300 |
| | | I3 | 300 | 250 | | 250 |
| Stratix IV GT | | I1 | 350 | 300 | 300 | 300 |
| Stratix V | | C2 | 350 | — | 300 | — |
| | | I3 | 300 | — | | — |
| HardCopy III *(3)*<br>HardCopy IV E *(3)* | FF Flip Chip | C | 300 | 150 | 300 | 150 |
| | | I | | | | |
| | LF Flip Chip | C | 300 | | 300 | |
| | | I | 250 | | 250 | |
| | Wirebound | C | 150 | | | |
| | | I | | | | |
| HardCopy IV GX *(3)* | FF Flip Chip | C | 300 | 150 | 300 | 150 |
| | | I | | | | |
| | LF Flip Chip | C | | | | |
| | | I | 250 | | 250 | |

**Notes to Table 1–4:**

(1) For some configurations, to achieve the stated maximum clock rates, you may need to use a faster memory device speed grade.

(2) The maximum clock rates listed here are guidelines based on Altera designs. The actual achievable performance of your design is based on your system's features and must be attained through the Quartus II timing analysis of the complete design.

(3) For FF and LF packages in nonemulation mode, to achieve the stated maximum clock rate, use 333-MHz memory device speed grade. For wirebound packages and for LF and FF packages in emulation mode, to achieve the stated maximum clock rate, use 250-MHz memory device speed grade.

**Table 1–5. QDR II+ SRAM Interface Maximum Clock Rate Estimates on Altera Devices** *(Note 1)*

| Device | | Speed Grade | Maximum Half-Rate Clock Rate (MHz) *(2)* | | Maximum Full-Rate Clock Rate (MHz) *(2)* | |
|---|---|---|---|---|---|---|
| | | | ×9, ×18, and ×36 | ×36 Emulation | ×9, ×18, and ×36 | ×36 Emulation |
| Arria II GX | | C4<br>I3 | 250 | 200 | 250 | 200 |
| Arria II GZ | | C3 | 350 | 250 | 300 | 250 |
| | | I3 | 350 | | | |
| Stratix III | | C2 | 400 | 300 | 300 | 300 |
| | | I3 | 350 | 250 | | 250 |
| Stratix III ($V_{CC}$ = 0.9 V) | | C4L<br>I4L | 250 | | 167 | 167 |
| Stratix IV | | C2<br>C2x | 400 | 300 | | |
| | | I3 | 350 | 250 | 300 | 250 |
| Stratix IV GT | | I1 | 400 | 300 | | |
| Stratix V | | C2 | 400 | — | 300 | — |
| | | I3 | 350 | — | | — |
| HardCopy III *(3)*<br>HardCopy IV E *(3)* | FF Flip Chip | C | 350 | 150 | 300 | 150 |
| | | I | 300 | | | |
| | LF Flip Chip | C | 300 | | 300 | |
| | | I | 250 | | 250 | |
| | Wirebound | C | 150 | | | |
| | | I | | | | |
| HardCopy IV GX *(3)* | FF Flip Chip | C | 350 | 150 | 300 | 150 |
| | | I | 300 | | | |
| | LF Flip Chip | C | 350 | | | |
| | | I | 300 | | | |

**Notes to Table 1–5:**

(1) For some configurations, to achieve the stated maximum clock rates, you may need to use a faster memory device speed grade.

(2) The maximum clock rates listed here are guidelines based on Altera designs. The actual achievable performance of your design is based on your system's features and must be attained through the Quartus II timing analysis of the complete design.

(3) For FF and LF packages in nonemulation mode, to achieve the stated maximum clock rate, use 400-MHz memory device speed grade. For Wirebound packages and for LF and FF packages in emulation mode, to achieve the stated maximum clock rate, use 250-MHz memory device speed grade.

# RLDRAM II Controller

For a list of supported and unsupported features in the RLDRAM II Controller with UniPHY, refer to the *RLDRAM II Controller with UniPHY IP User Guide*.

Table 1–6 shows the maximum clock rates supported on column I/Os for RLDRAM II controllers with UniPHY on various Altera devices with the fastest speed grade.

Table 1–6 also shows the maximum clock rates supported on column and row I/Os on HardCopy devices for single chip-select RLDRAM II Controller with UniPHY. Wrap around clock rates are lower; to determine the maximum achievable clock rate for a wrap around interface, you must compile your design and perform timing analysis in the Quartus II software.

**Table 1–6. RLDRAM II Interface Maximum Clock Rate Estimates on Altera Devices *(Note 1)***

| Device | | Speed Grade | Maximum Half-Rate Clock Rate (MHz) *(2)* | Maximum Full-Rate Clock Rate (MHz) *(2)* |
|---|---|---|---|---|
| Arria II GZ | | C3 | 350 | 300 |
| | | I3 | | |
| Stratix III | | C2 | 400 *(3)* | 300 |
| | | I3 | 350 | 275 |
| Stratix IV | | C2<br>C2x | 400 | 300 |
| | | I3 | 350 | |
| Stratix IV GT | | I1 | 400 | |
| Stratix V | | C2 | 400 *(3)* | 300 |
| | | I3 | 350 | |
| HardCopy III *(4)* *(5)*<br>HardCopy IV E *(4)* *(5)* | FF Flip Chip | C | 333 | 267 |
| | | I | 300 | |
| | LF Flip Chip | C | 300 | |
| | | I | 267 | |
| | Wirebound | C | 170 | |
| | | I | | |
| HardCopy IV GX *(4)* *(5)* | FF Flip Chip | C | 350 | 267 |
| | | I | | |
| | LF Flip Chip | C | 333 | 267 |
| | | I | 300 | |

**Notes to Table 1–6:**

(1) For some configurations, to achieve the stated maximum clock rates, you may need to use a faster memory device speed grade.

(2) The maximum clock rates listed here are guidelines based on Altera designs. The actual achievable performance of your design is based on your system's features and must be attained through the Quartus II timing analysis of the complete design.

(3) To achieve the stated maximum clock rate, use 533-MHz memory device speed grade.

(4) The maximum clock rate for x36 interfaces is 170 MHz.

(5) For FF and LF packages in nonemulation mode, to achieve the stated maximum clock rate, use 533-MHz memory device speed grade. For Wirebound packages and for FF and LF packages in emulation mode, to achieve the stated maximum clock rate, use 300-MHz memory device speed grade.

This chapter provides additional information about the document and Altera.

## Revision History

The following table shows the revision history for this document.

| Date | Version | Changes |
|---|---|---|
| December 2010 | 2.2 | Updated for 10.1 release. |
| July 2010 | 2.1 | Updated for 10.0 release. |
| January 2010 | 1.1 | Corrected minor typos. |
| November 2009 | 1.0 | Initial release. |

## How to Contact Altera

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

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

**Note to Table:**

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

## Typographic Conventions

The following table shows the typographic conventions this document uses.

| Visual Cue | Meaning |
|---|---|
| **Bold Type with Initial Capital Letters** | Indicate command names, dialog box titles, dialog box options, and other GUI labels. For example, **Save As** dialog box. For GUI elements, capitalization matches the GUI. |
| **bold type** | Indicates directory names, project names, disk drive names, file names, file name extensions, software utility names, and GUI labels. For example, **\qdesigns** directory, **D:** drive, and **chiptrip.gdf** file. |
| *Italic Type with Initial Capital Letters* | Indicate document titles. For example, *Stratix IV Design Guidelines*. |

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

# External Memory Interface Handbook

# Volume 2: Device, Pin, and Board Layout Guidelines

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

EMI_PLAN-2.1

Document last updated for Altera Complete Design Suite version: 10.1
Document publication date: December 2010

Subscribe

# External Memory Interface Handbook Volume 2

## Section I. Device and Pin Planning

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

EMI_PLAN_PIN-2.1

Document last updated for Altera Complete Design Suite version: 10.1
Document publication date: December 2010

Subscribe

# Contents

# 1. Selecting a Device

This chapter discusses the planning stage, to ensure that you know what to look for when selecting the right Altera® FPGA device for your memory interface.

Use this document with "Planning Pin and Resource" on page 2–1, before you start implementing your external memory interface.

Memory controllers in Altera devices require access to dedicated I/O element (IOE) features, PLLs, and several clock networks. Altera devices are feature rich in all these areas, so you must consider detailed resource and pin planning whenever implementing complex IP or multiple IP cores. This chapter provides an overview of what to consider in such instances.

Use the Altera Product Selector to find and compare specifications and features of Altera devices.

For information about the terms used in this document, refer to the *Glossary* chapter in volume 1 of the *External Memory Interface Handbook*.

For more information about supported memory types and configurations, refer to the *External Memory Interface System Specifications* section in volume 1 of the *External Memory Interface Handbook*.

When selecting the optimal Altera device for your memory interface, consider the following topics:



## Memory Standards and Configurations

There are two common types of high-speed memories that are supported by Altera devices: DRAM and SRAM. Commonly used DRAM devices are DDR, DDR2, DDR3 SDRAM, and RLDRAM II; SRAM devices include QDR II and QDR II+ SRAM.

Different Altera devices support different memory types; not all Altera devices support all memory types and configurations. Before you start your design, you must select an Altera device, which supports the memory standard and configurations you plan to use.

In addition, Altera's FPGA devices support various data widths for different memory interfaces. The memory interface support between density and package combinations differs, so you must determine which FPGA device density and package combination best suits your application.

For more information about these memory standards, refer to the *Memory Standard Overview* section in volume 1 of the *External Memory Interface Handbook*.

## Device Pin Count

To meet the growing demand for memory bandwidth and memory data rates, memory interface systems use parallel memory channels and multiple controller interfaces. However, the number of memory channels is limited by the package pin count of the Altera devices. Hence, you must consider device pin count when you select a device; you must select a device with enough I/O pins for your memory interface requirement.

The number of device pins depends on the memory standard, the number of memory interfaces, and the memory data width. For example, a ×72 DDR3 SDRAM single-rank interface requires 125 I/O pins:

- 72 DQ pins (including ECC)
- 9 DM pins
- 9 DQS, DQSn differential pin pairs
- 17 address pins (address and bank address)
- 7 command pins (CAS, RAS, WE, CKE, ODT, reset, and CS)
- 1 CK, CK# differential pin pair

For the available number of DQS groups and the maximum number of controllers that is supported by the FPGAs for different memory types, refer to the *External Memory Interface System Specifications* section in volume 1 of the *External Memory Interface Handbook*.

Altera devices do not limit the interface widths beyond the following requirements:

- DQS, DQ, clock, and address signals of the entire interface should reside within the same bank or side of the device if possible, to achieve better performance. Although wraparound interfaces are also supported at limited frequencies.
- The maximum possible interface width in any particular device is limited by the number of DQS and DQ groups available within that bank or side.
- Sufficient regional clock networks are available to the interface PLL to allow implementation within the required number of quadrants.
- Sufficient spare pins exist within the chosen bank or side of the device to include all other clock, address, and command pin placement requirements.
- The greater the number of banks, the greater the skew. Altera recommends that you always compile a test project of your desired configuration and confirm that it meets timing requirement.

There is a constraint in Arria® II GX devices when assigning DQS and DQ pins. You are only allowed to use twelve of the sixteen I/O pins in an I/O module for DQ pin. The remaining four pins can only be used as input pin.

For DQS groups pin-out restriction format, refer to *Arria II GX Pin Connection Guidelines*.

Your pin count calculation also determines which device side to use (top or bottom, left or right, and wraparound).

## Top or Bottom and Left or Right Interfaces

Ideally any interface should wholly reside in a single bank. However, interfaces that span across multiple adjacent banks or the entire side of a device are also fully supported. Although vertical and horizontal I/O timing parameters are not identical, timing closure can be achieved on all sides of the FPGA for the maximum interface frequency.

Arria II GX, Cyclone® III, and Cyclone IV devices support interfaces spanning the top and bottom sides. In addition, Cyclone III and Cyclone IV E devices also support interfaces spanning left and right sides. Interfaces that span across sides (top and bottom, or left and right) and wraparound interfaces provide the same level of performance.

Arria II GX, Cyclone IV, and Stratix V devices do not support the left interface. There are no user I/O pins, other than the transceiver pins available in these devices.

## Wraparound Interfaces

For maximum performance, Altera recommends that data groups for external memory interfaces should always be within the same side of a device, ideally reside within a single bank. High-speed memory interfaces using top or bottom I/O bank versus left or right IO bank have different timing characteristics, so the timing margins are also different. However, Altera can support interfaces with wraparound data groups that wraparound a corner of the device between vertical and horizontal I/O banks at some speeds. Some devices wraparound interfaces are same speed as row or column interfaces.

Arria II GX, Cyclone III and Cyclone IV devices can support wraparound interface across all sides of devices that are not used for transceivers. Other Altera devices only support interfaces with data groups that wraparound a corner of the device.

# IP Support

Altera offers IP solutions to help you create your own external memory interface system. There are two parts of the external memory interface IPs:

- PHY
- Memory controller

For more information on the IP Altera offers, refer to *Volume 3: Implementing Altera Memory Interface IP* of the *External Memory Interface Handbook*.

## PHY

Altera offers external memory interface PHYs for calibration and tracking mechanisms. These PHYs support full-rate and half-rate interfaces, which you can then plug a custom controller with the PHY using the AFI. For example, the ALTMEMPHY megafunction.

The Altera UniPHY IP provides better performance and lower latency. The UniPHY IP instantiates the PLL and DLL in the top-level design to support multiple interfaces.

Alternatively, you can build a custom PHY using the ALTDLL or ALTDQ_DQS megafunctions. However, you are then responsible for determining the maximum interface frequency of your system and in creating the timing constraints required to create a robust PHY.

For more information about creating custom interfaces, refer to *Volume 5: Implementing Custom Memory Interface PHY* of the *External Memory Interface Handbook*.

## Controllers

Altera offers controllers for a full solution that you can immediately integrate with your user logic. For example, the DDR, DDR2, or DDR3 SDRAM controllers with ALTMEMPHY, or the DDR2, DDR3 SDRAM, RLDRAM II or QDR II controllers with UniPHY. These controllers include the Altera PHY IP and thus provide a simplified interface to DDR, DDR2, DDR3 SDRAM, RLDRAM II, and QDR II and QDR II+ SRAM components.

## Half- or Full-Rate SDRAM Controller

When implementing memory controllers consider whether a half-rate or a full-rate datapath is optimal for your design. Half- or full-rate designs have the following definitions:

- Half-rate designs present data to the local interface at four times the width of the actual SDRAM interface at half the SDRAM clock rate.
- Full-rate designs present data to the local interface at twice the width of the actual SDRAM interface at the full SDRAM clock rate.

Implementing half-rate memory controllers results in the highest possible SDRAM clock frequency, at the expense of latency and efficiency. You need to decide whether system efficiency or system bandwidth is more important.

This implementation is most useful when core HDL designs are difficult to implement at the higher SDRAM clock frequency, but the required SDRAM bandwidth per I/O pin is still quite high.

However, full-rate controller operations are faster with the IP operating at the same clock frequency as your system.

Consider that DDR devices can have a number of banks opened at once. Each bank has a currently selected row. Changing the column within the selected row of an open bank requires no additional bank management commands to be issued. Changing the row in an active bank, or changing the bank, both incur a protocol penalty that requires the precharge command closes the active row or bank, and the activate command opens (or activates) the new row or bank combination.

The duration of this penalty is a function of the controller clock frequency, the memory clock frequency, and the memory device characteristics. Calculating the impact of a change of memory and controller configuration on a given system is not a trivial task, as it depends on the nature of the accesses that are performed.

In this example each command takes a single clock cycle in a full-rate controller, but two clock cycles in a half-rate controller. The bank is not available for the subsequent activate command until ($t_{RP}$) after the precharge. So while the issuing of commands can be slower using a half-rate controller, the respective memory timing parameters remain the same.

Hence, when a design uses a half-rate memory controller, the control circuitry is clocked at half rate and so control operations are slower than a full-rate design. However, the memory's clock frequency and physical properties are not affected.

# Controller Efficiency

Understanding how to increase the efficiency and bandwidth of the memory controller is important when you design any external memory interface. This section discusses the factors that affect controller efficiency and the ways to increase the efficiency of the controller.

Controller efficiency varies depending on data transaction. The best way to find out the efficiency of the controller is to simulate the memory controller for your specific design.

You express controller efficiency as:

Efficiency = number of active cycles of data transfer/total number of cycles

The total number of cycles includes the number of cycles required to issue commands or other requests.

You calculate the number of active cycles of data transfer in terms of local clock cycles. For example, if the number of active cycles of data transfer is 2 memory clock cycle, you convert that to the local clock cycle which is 1.

The following cases are based on a DDR2 SDRAM high-performance controller (HPC) design targeting a Stratix® IV device that has a CAS latency of 3, and burst length of 4 on the memory side (2 cycles of data transfer), with accessed bank and row in the memory device already open. The Stratix IV device has a command latency of 9 cycles in half-rate mode. The `local_ready` signal is high.

- Case 1: The controller performs individual reads.

  Efficiency = 1/(1 + CAS + command latency) = 1/(1+1.5+9) = 1/11.5 = 8.6%

- Case 2: The controller performs 4 back to back reads.

  In this case, the number of data transfer active cycles is 8. The CAS latency is only counted once because the data coming back after the first read is continuous. Only the CAS latency for the first read has an impact on efficiency. The command latency is also counted once because the back to back read commands use the same bank and row.

  Efficiency = 4/(4 + CAS + command latency) = 4/(4+1.5+9) = 1/14.5 = 27.5%

## Factors Affecting Efficiency

To enhance controller efficiency, you must take into consideration certain factors. The two main factors that affect the efficiency of a controller are the interface standard specified by the memory vendor, and the way you transfer data.

The following sections discuss these factors in detail.

### Interface Standard

Complying to certain interface standard specifications affects controller efficiency. When interfacing the memory with the DDR2 or DDR3 SDRAM controllers, you must follow certain timing specifications and perform the following bank management operations:

- Activate

  Before you issue any read (RD) or write (WR) commands to a bank within a DDR2 SDRAM device, you must open a row in that bank using the activate (ACT) command. After you open a row, only then you issue a read or write command to that row based on the $t_{RCD}$ specification. Reading or writing to a closed row has negative impact on the efficiency as the controller has to to first activate that row and then wait until $t_{RCD}$ time to perform a read or write.

- Precharge

  To open a different row in the same bank, you must issue a precharge (PCH) command. The precharge command deactivates the open row in a particular bank or the open row in all banks. Switching a row has a negative impact on the efficiency as you must first precharge the open row, then activate the next row and wait $t_{RCD}$ time to perform any read or write operation to the row.

- Device CAS latency

  The higher the CAS latency, the less efficient an individual access. The memory device has its own read latency, which is about 12 ns to 20 ns regardless of the actual frequency of the operation. The higher the operating frequency, the higher the CAS latency is in number of cycles.

- Refresh

  A refresh, in terms of cycles, consists of the precharge command and the waiting for the auto refresh period. Based on the memory datasheet, these components require the following values:

  - $t_{RP}$ = 12 ns, 3 clock cycles for a 200-MHz operation (5 ns period for 200 MHz)
  - $t_{RFC}$ = 75 ns, 15 clock cycles for a 200-MHz operation.

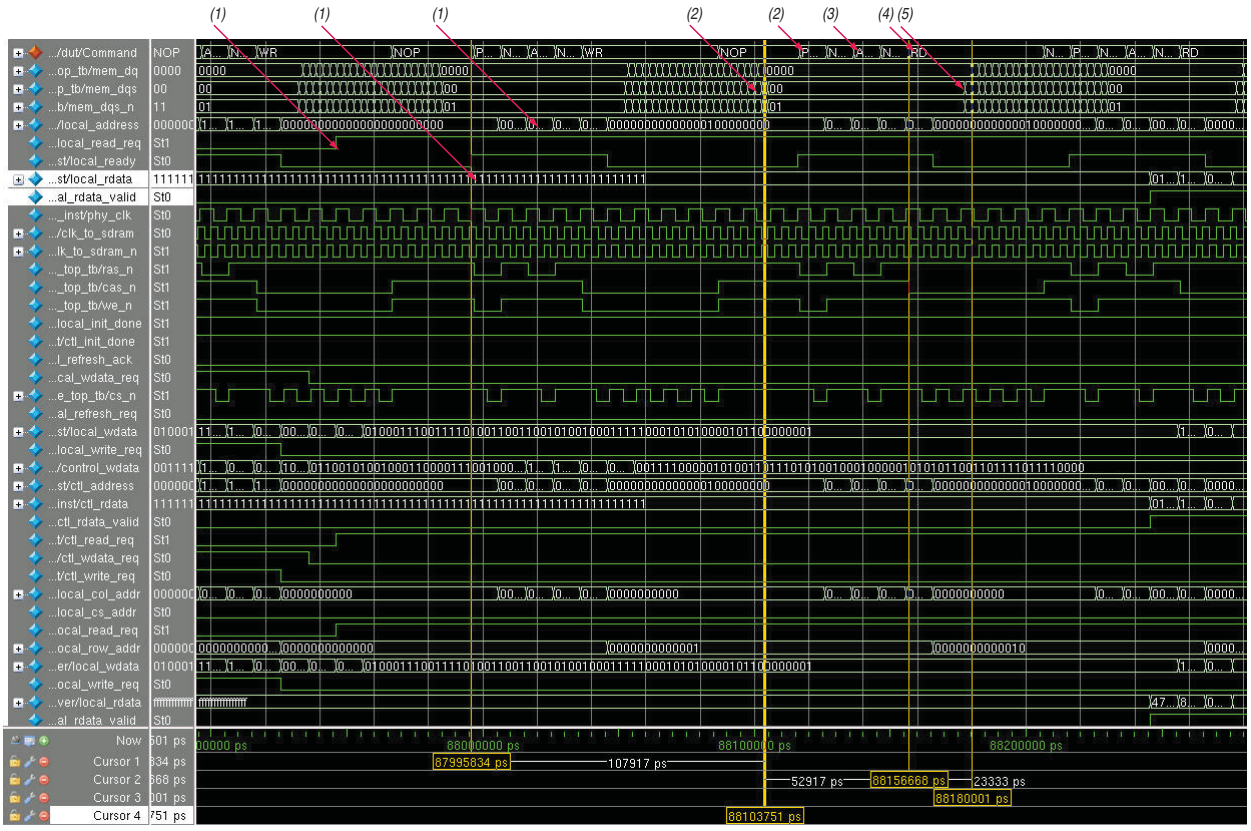
  Based on this calculation, a refresh pauses read or write operations for 18 clock cycles. So, at 200 MHz, you lose 1.15% (18 x 5 ns/7.8 us) of the total efficiency.

Figure 1–1 and Figure 1–2 show some examples of how the bank management operations affect controller efficiency.

Figure 1–1 shows a read operation where you have to change a row in a bank. This figure shows how CAS latency, and precharge and activate commands impact the efficiency.

Figure 1–1 illustrates a read after write operation. The controller changes the row address after the write to read from a different row.

**Figure 1–1. Read Operation—Changing A Row in A Bank**




The following sequence of events describes Figure 1–1.

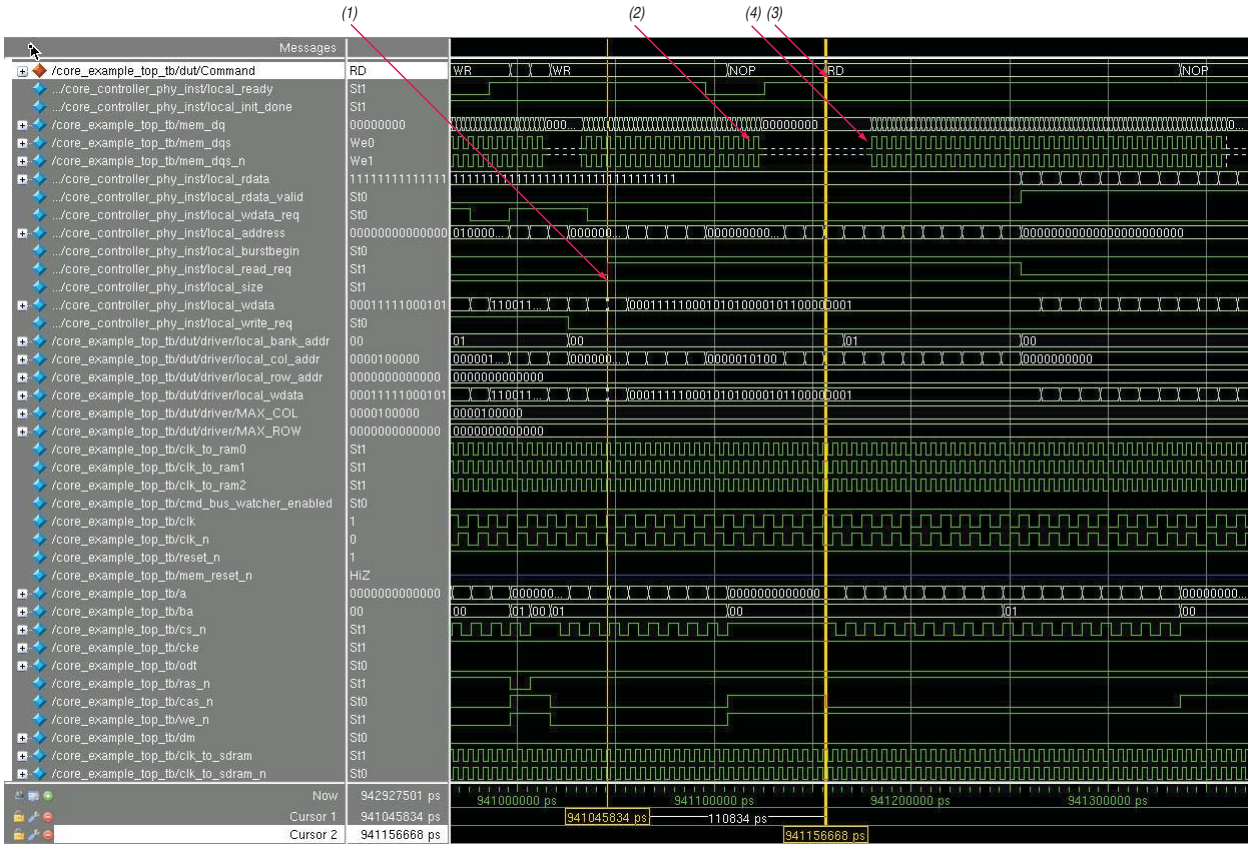
1. The `local_read_req` signal goes high, and when the `local_ready` signal goes high, the controller accepts the read request along with the address.

2. After the memory receives the last write data, the row changes for read. Now you require a precharge command to close the row opened for write. The controller waits for $t_{WR}$ time (3 memory clock cycles) to give the precharge command after the memory receives the last write data.
3. After the controller issues the precharge command, it must wait for $t_{RP}$ time to issue an activate command to open a row.
4. After the controller gives the activate command to activate the row, it needs to wait $t_{RCD}$ time to issue a read command.
5. After the memory receives the read command, it takes the memory some time to give the data on the pin. This time is known as CAS latency, which is 3 memory clock cycles in this case.

For this particular case, you need approximately 17 local clock cycles to issue a read command to the memory. Because the row in the bank changes, the read operation takes a longer time as the controller has to issue the precharge and activate commands first. You do not have to take into account $t_{WTR}$ for this case because the precharge and activate operations already exceeded $t_{WTR}$ time.

Figure 1–2 shows the case where you use the same the row and bank address when the controller switches from write to read. In this case, the read command latency is reduced.

**Figure 1–2. Changing From Write to Read—Same Row and Bank Address**




The following sequence of events describes Figure 1–2:

1. The `local_read_req` signal goes high and `local_ready` signal is high already. The controller accepts the read request along with the address.
2. When switching from write to read, the controller has to wait $t_{WTR}$ time before it gives a read command to the memory.
3. The SDRAM device receives the read command.
4. After the SDRAM device receives the read command, it takes some time to give the data on the pin. This time is called CAS latency, which is 3 memory clock cycles in this case.

For this particular case, you need approximately 11 local clock cycles to issue a read command to the memory. As the row in the bank remains the same, the controller does not have to issue the precharge and activate commands, which speeds up the read operation and in turn results in a better efficiency compared to the case in Figure 1–1.

Similarly, if you do not switch between read and write often, the efficiency of your controller improves significantly.

## Data Transfer

The way you transfer data also affects controller efficiency. The following methods reduces the efficiency of your controller:

- Individual read or write accesses.

  Performing individual read or write accesses are less efficient.

- Switching between read and write operations.

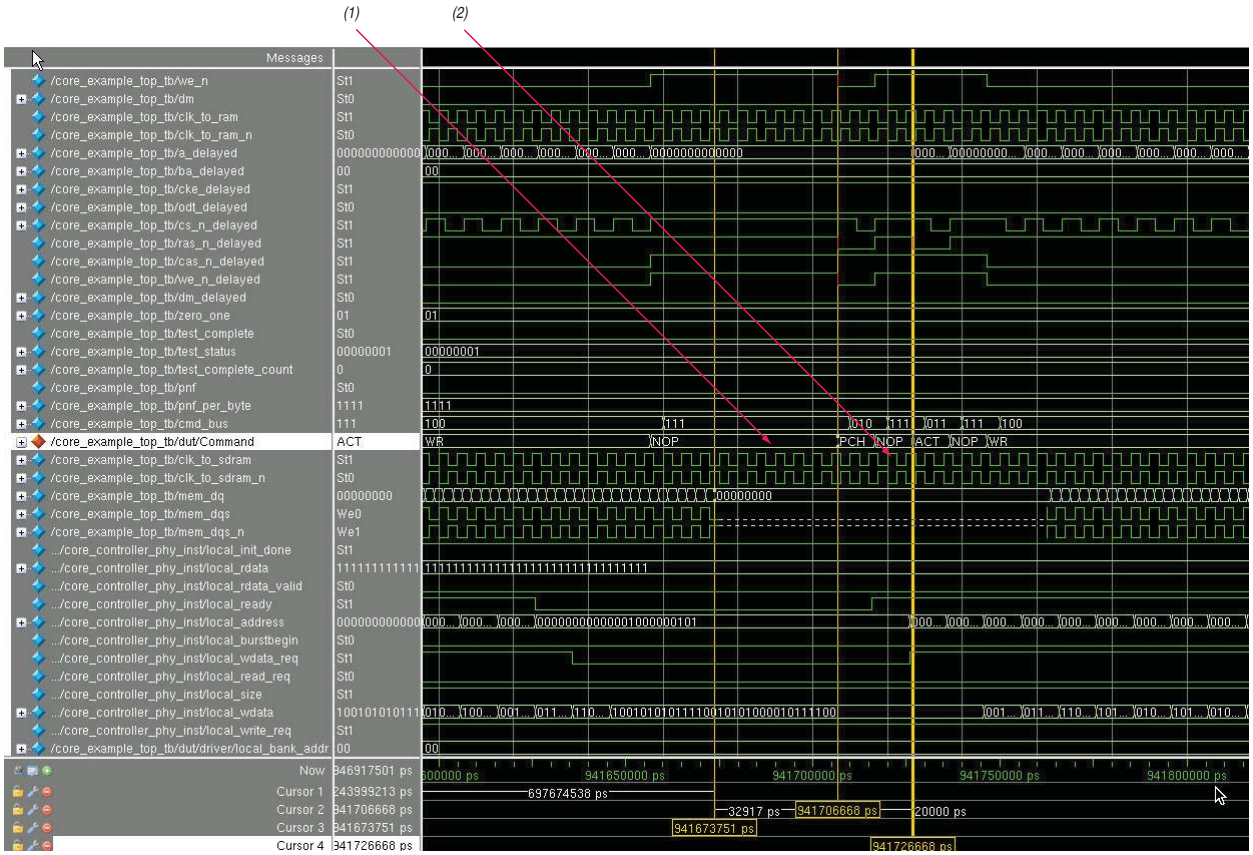
  Switching between read and write operation has a negative impact on the efficiency of the controller.

- Reading or writing from different rows.

  Performing read or write operations from different rows within a bank or in a different bank—if the bank and a row you are accessing is not already open—also affects the efficiency of your controller.

Figure 1–3 shows an example of changing the row in the same bank.

**Figure 1–3. Changing Row in the Same Bank**




The following sequence of events describes Figure 1–3:

1. You have to wait $t_{WR}$ time before giving the precharge command
2. You then wait $t_{RP}$ time to give the activate command.

## Ways to Improve Efficiency

To improve the efficiency of your controller, you can use the following methods:

- DDR2 SDRAM Controller
- Auto-precharge Commands
- Additive Latency
- Bank Interleaving
- Additive Latency and Bank Interleaving
- User-controlled Refresh
- Frequency of Operation
- Burst Length
- Series of Reads or Writes

The following sections discuss these methods in detail.

## DDR2 SDRAM Controller

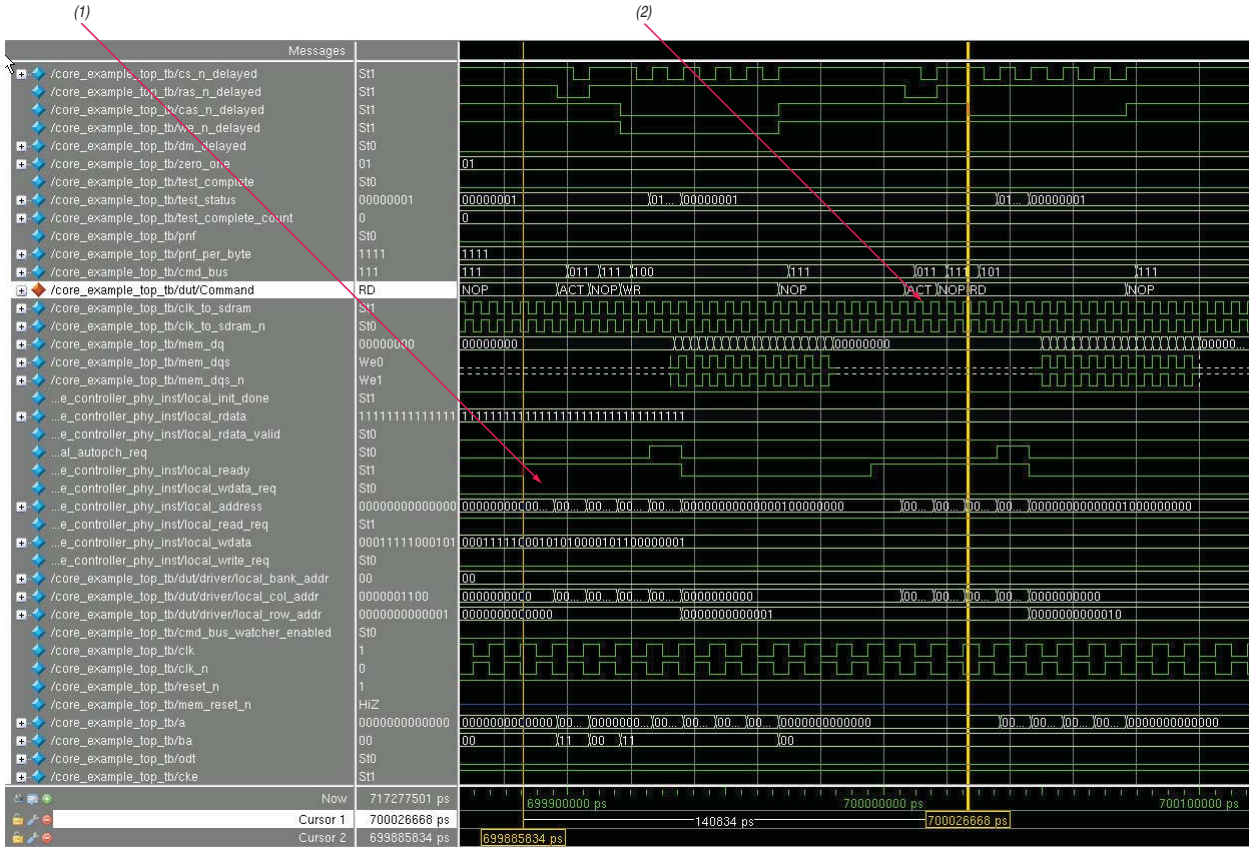
The DDR2 SDRAM controller maintains up to eight open banks; a row in each bank is open at a time. Maintaining more banks at one time helps avoid bank management commands. Make sure that you do not change a row in a bank frequently as changing the row in bank causes the bank to close and reopen to open another row in that bank.

## Auto-precharge Commands

The auto-precharge read and write commands allow you to indicate to the memory device that this read or write command is the last access to the currently opened row. The memory device automatically closes or auto-precharges the page it is currently accessing, so that the next access to the same bank is quicker. This command is useful when doing fast random access.

Figure 1–4 shows how you can improve controller efficiency using auto-precharge command.

**Figure 1–4. Improving Efficiency Using Auto-Precharge Command**




The following sequence of events describes Figure 1–4.

1. The controller accepts read request from the local side as soon as the `local_ready` signal goes high.

2. The controller gives the activate command and then gives the read command. The read command latency is approximately 14 clock cycles for this case as compared to similar case with no auto precharge which had approximately 17 clock cycles of latency (described in Figure 1–3).

When using the auto-precharge option, take note of the following two guidelines:

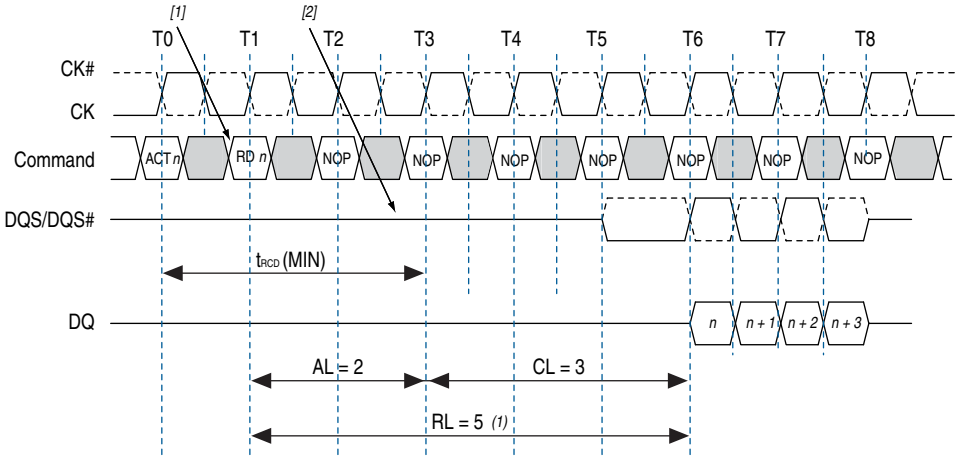
- Use auto-precharge command if you know the controller is issuing the next read or write to a particular bank to a different row.
- Auto-precharge does not improve efficiency if you auto-precharge a row only to reopen it.

## Additive Latency

Additive latency makes the command and data bus efficient for sustainable bandwidths. You may issue the commands externally but the device holds the commands internally, for the duration of additive latency, before executing to improve the system scheduling. Specifically, the delay helps to avoid collision on the command bus and gaps in data input or output bursts. Additive latency allows the controller to issue the row and column address commands—activate, and read or write—in consecutive clock cycles, so that it does not hold the column address for several ($t_{RCD}$) cycles. This gap between the activate and the read or write command can cause bubbles in the data stream.

Figure 1–5 shows an example of how additive latency is used.

**Figure 1–5. Additive Latency—Read**




The following sequence of events describes Figure 1–5.

1. The controller issues a read or write command before the $t_{RCD}$ (MIN) requirement— additive latency less than or equal to $t_{RCD}$ (MIN).
2. The controller holds the read or write command for the time stated by additive latency before issuing it internally to the SDRAM device.

   Read latency = additive latency + CAS latency

   Write latency = additive latency + CAS latency – 1 × $t_{CK}$

## Bank Interleaving

You can use bank interleaving to sustain bus efficiency when the controller misses a page, and that page is in a different bank.

Page size refers to the minimum number of column locations on any row that are accessed with a single activate command.

Without interleaving, the controller sends the address to the SDRAM device, receives the data requested, and then waits for the SDRAM device to refresh before initiating the next data transaction; thus wasting a lot of clock cycles.

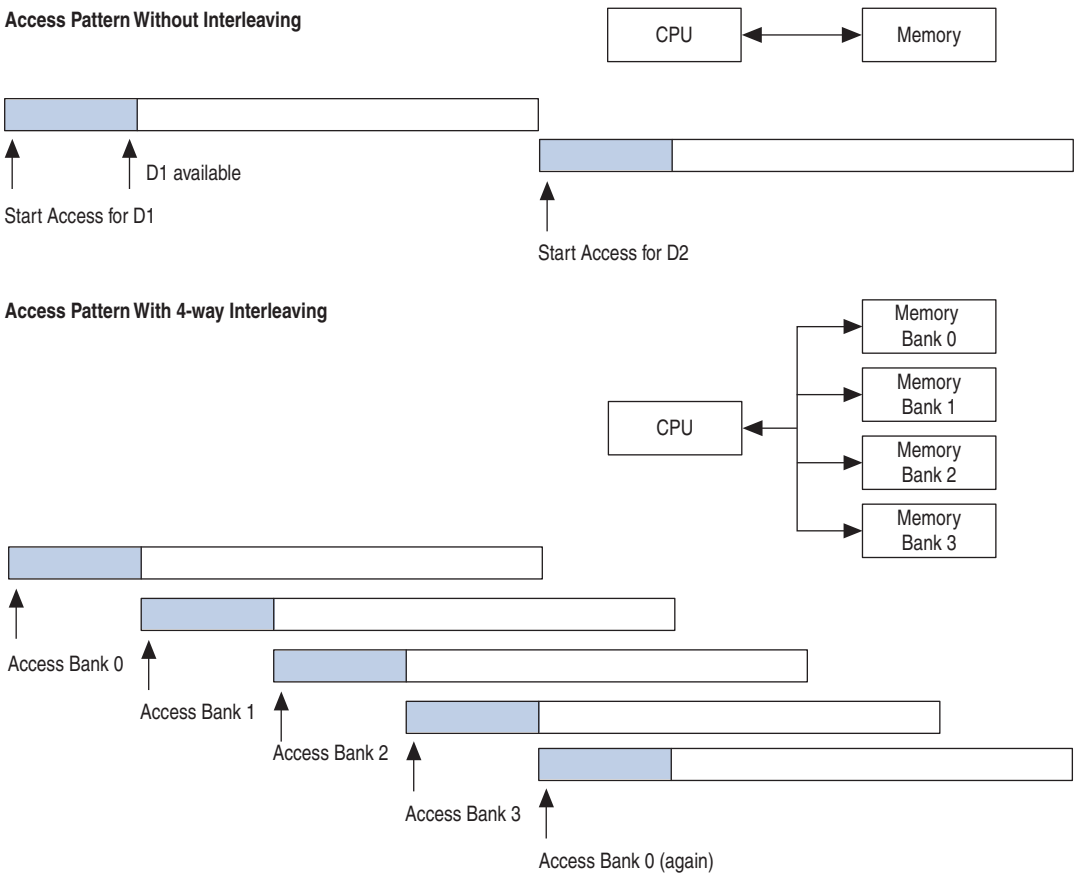
Interleaving allows banks of the SDRAM device to alternate their refresh and access cycles. One bank undergoes its refresh cycle while another is being accessed. By alternating, the controller improves its performance by masking the refresh time of each bank. If there are four banks in the system, the controller can ideally send one data request to each of the banks in consecutive clock cycles.

For example, in the first clock cycle, the CPU sends an address to Bank 0, and then sends the next address to Bank 1 in the second clock cycle, before sending the third and fourth addresses to Banks 2 and 3 in the third and fourth clock cycles respectively. The sequence is as follows:

1. Controller sends address 0 to Bank 0.
2. Controller sends address 1 to Bank 1 and receives data 0 from Bank 0.
3. Controller sends address 2 to Bank 2 and receives data 1 from Bank 1.
4. Controller sends address 3 to Bank 3 and receives data 2 from Bank 2.
5. Controller receives data 3 from Bank 3.

Figure 1–6 shows how you can increase bandwidth by using interleaving.

**Figure 1–6. Using Interleaving to Increase Bandwidth**



## Additive Latency and Bank Interleaving

Using additive latency together with bank interleaving increases the bandwidth of the controller.

Figure 1–7 shows an example of bank interleaving in a read operation without additive latency.

**Figure 1–7. Bank Interleaving—Without Additive Latency**

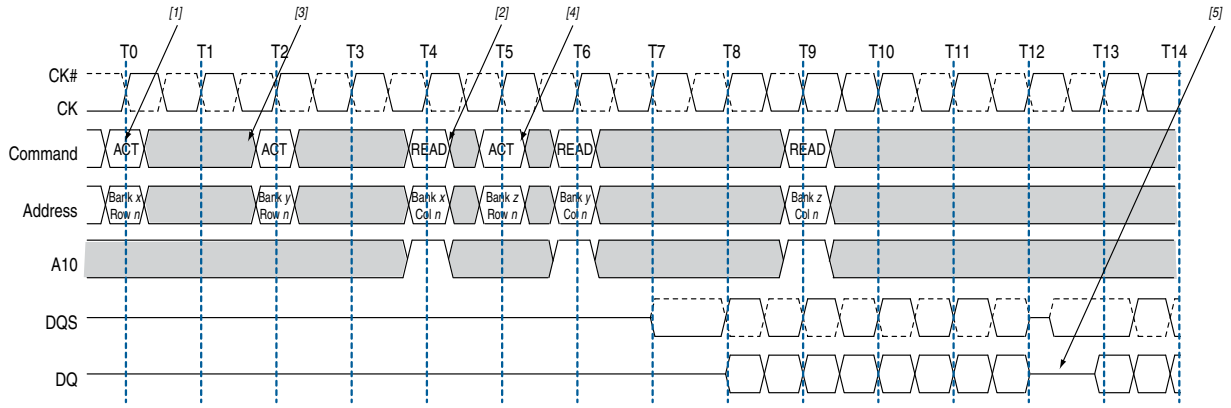



Figure 1–7 illustrates an example of DDR2 SDRAM bank interleave reads with CAS latency of 4, and burst length of 4.

The following sequence of events describes Figure 1–7.

1. The controller issues an activate command to open the bank, which activates bank *x* and the row in it.
2. After $t_{RCD}$ time, the controller issues a read with auto-precharge command to that bank.
3. Bank *y* receives an activate command after $t_{RRD}$ time.
4. The controller cannot issue an activate command to bank *z* at its optimal location because it has to wait for bank *x* to receive the read with auto-precharge command; thus delaying the activate command for one clock cycle.
5. The delay in activate command causes a gap in the output data from the DDR2 SDRAM device.

If you use additive latency of 1, the latency only affects read commands and not the timing for write commands.

Figure 1–8 shows an example of bank interleaving in a read operation with additive latency. In this configuration, the controller issues back-to-back activate and read with auto-precharge commands.

**Figure 1–8. Bank Interleaving—With Additive Latency**

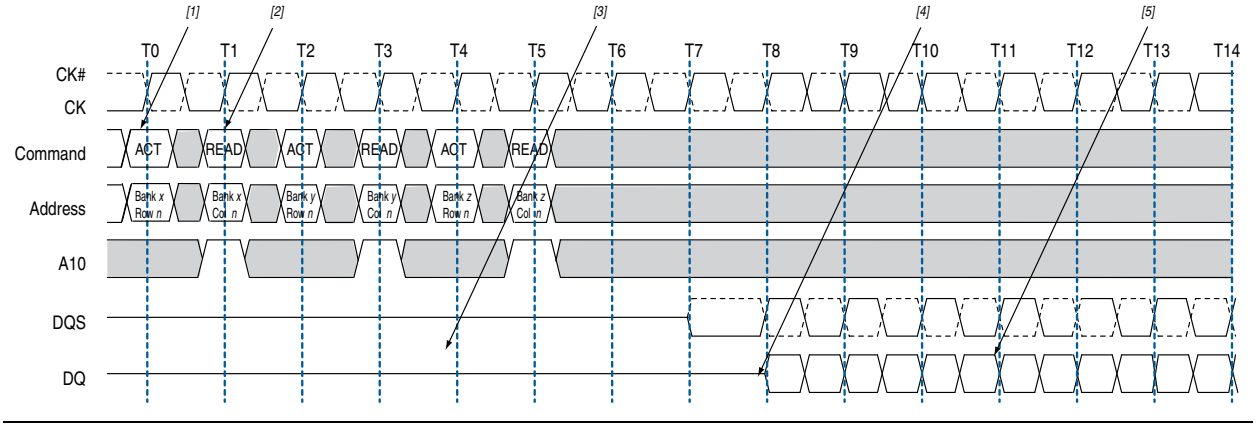



Figure 1–8 illustrates an example of a DDR2 SDRAM bank interleave reads with additive latency of 3, CAS latency of 4, and burst length of 4.

The following sequence of events describes Figure 1–8.

1. The controller issues an activate command to bank *x*.
2. The controller issues a read with auto precharge command to bank *x* right after the activate command, before waiting for the $t_{RCD}$ time.
3. The controller executes the read with auto-precharge command $t_{RCD}$ time later on the rising edge T4.
4. 4 cycles of CAS latency later, the SDRAM device issues the data on the data bus.
5. For burst length of 4, you need 2 cycles for data transfer. With 2 clocks of giving activate and read with auto-precharge commands, you get a continuous flow of output data.

Compare the following efficiency results in Figure 1–7 and Figure 1–8:

- DDR2 SDRAM bank interleave reads with no additive latency, CAS latency of 4, and burst length of 4 (Figure 1–7),

  Number of active cycles of data transfer = 6.

  Total number of cycles = 15

  Efficiency = 40%

- DDR2 SDRAM bank interleave reads with additive latency of 3, CAS latency of 4, and burst length of 4 (Figure 1–8),

  Number of active cycles of data transfer = 6.

  Total number of cycles = 14

  Efficiency = approximately 43%

The interleaving reads used with additive latency increases efficiency to approximately 3%.

Additive latency improves the efficiency of back-to-back interleaved reads or writes, but not individual random reads or writes.

## User-controlled Refresh

Use self-refresh option to control refresh operation. You can choose to perform refresh before or after you complete a read or write operation. By using the self-refresh option, you can avoid interrupting any read or write operation, thus improving the efficiency.

## Frequency of Operation

Certain frequencies of operation give you the best possible latency based on the memory parameters. The memory parameters you specify through the parameter editor in the MegaWizard™ Plug-In Manager are converted to clock cycles and rounded up.

If you are using a memory device that has $t_{RCD}$ = 20 ns and running the interface at 100 MHz, you get the following results:

- For full-rate implementation ($t_{Ck}$ = 10 ns):

  $t_{RCD}$ convert to clock cycle = 20/10 = 2.

- For half rate implementation ($t_{Ck}$ = 20 ns ):

  $t_{RCD}$ convert to clock cycle = 20/20 = 1

This frequency and parameter combination is not easy to find because there are a lot of memory parameters and frequencies for the memory device and the controller to run. Memory device parameters are optimal for the speed at which the device is designed to run; so run the device at the speed it is intended to run.

In most cases, the frequency and parameter combination is not optimal. If you are using a memory device that has $t_{RCD}$ = 20 ns and running the interface at 133 MHz, you get the following results:

- For full-rate implementation ($t_{Ck}$ = 7.5 ns):

  $t_{RCD}$ convert to clock cycle = 20/7.5 = 2.66 , rounded up to 3 clock cycles or 22.5 ns.

- For half rate implementation ($t_{Ck}$ = 15 ns ):

  $t_{RCD}$ convert to clock cycle = 20/15 = 1.33 , rounded up to 2 clock cycles or 30 ns.

There is no latency difference for this frequency and parameter combination.

## Burst Length

Burst length affects the efficiency of the controller. Using burst length of 8 gives more cycles of data transfer, compared to using burst length of 4.

For a half-rate design that has a command latency of 9 half-rate clock cycles, and a CAS latency of 3 memory clock cycles or 1.5 half rate local clock cycles, the efficiency is 9% for burst length of 4, and 16% for burst length of 8.

- Burst length of 4 (2 memory clock cycles of data transfer or 1 half-rate local clock cycle)

  Efficiency = number of active cycles of data transfer/total number of cycles

Efficiency = 1/(1 + CAS + command latency) = 1/(1 + 1.5 + 9) = 1/11.5 = 8.6% or approximately 9%

- Burst length of 8 (4 memory clock cycles of data transfer or 2 half-rate local clock cycles)

  Efficiency = number of active cycles of data transfer/total number of cycles

  Efficiency = 2/(2 + CAS + command latency) = 2/(2 + 1.5 + 9) = 2/12.5 = 16%

### Series of Reads or Writes

Performing series of reads or writes from the same bank and row increases controller efficiency.

The case shown in Figure 1–2 on page 1–8 demonstrates that a read performed from the same row takes only 14.5 clock cycles to transfer data, making the controller 27% efficient.

Do not perform random reads or random writes. When you perform reads and writes to random locations, such as changing rows in a bank or changing banks, the controller has to precharge the previous bank, and activate the row in the new bank. Even if you change the row in the same bank, the controller has to close the bank (precharge) and reopen it again just to open a new row (activate). Because of the precharge and activate commands, efficiency decreases as the controller needs more time to issue a read or write.

If you have to perform a random read or write, use additive latency and bank interleaving to increase efficiency.

The efficiency of the controller is dependent on how the data transfer is performed between the memory device and the FPGA, the memory standards specified by the memory device vendors, and the type of memory controllers.

## Bandwidth

The bandwidth is dependent on the efficiency of the memory controller that controls the data transfer from and to the memory device.

You can express bandwidth as follows:

Bandwidth = data width (bits) × data transfer rate (1/s) × efficiency

Data rate transfer (1/s) = 2 × frequency of operation (4 × for QDR SRAM interfaces)

The following example shows the bandwidth calculation for a 16-bit interface that has 70% efficiency and runs at 200 MHz frequency:

Bandwidth = 16 bits × 2 clock edges × 200 MHz × 70 % = 4.48 Gbps.

DRAM typically has an efficiency of around 70%, but when using the Altera memory controller efficiency can vary from 10 to 92%.

In QDR II+ or QDR II SRAM the IP implements two separate unidirectional write and read data buses, so the data transfer rate is four times the clock rate. The data transfer rate for a 400-MHz interface is 1, 600 Mbps. The efficiency is the percentage of time the data bus is transferring data. It is dependent on the type of memory. For example, in a QDR II+ or QDR II SRAM interface with separate write and read ports, the efficiency is 100%, when there is an equal number of read and write operations on these memory interfaces.

# External Memory Interface Features of Altera Devices

This topic describes features from the Altera device families that enable external memory interfaces.

For more information on the external memory interface circuitry in an Altera FPGA device family, refer to the *External Memory Interfaces* chapter in the relevant device family handbook.

## IOE Dedicated Features

When selecting an Altera device for your external memory system, you must select a device that is equipped with the features that suit your memory system requirements and configurations.

Altera devices have enhanced upon the IOE DDR capabilities by including the feature functionality availability directly in the IOE.

Table 1–1 shows which device families support these features.

**Table 1–1. Device IOE Dedicated Features *(Note 1)* (Part 1 of 2)**

| Features | Device Family | | | |
|---|---|---|---|---|
| | Arria II GX | Cyclone III and Cyclone IV | Arria II GZ, Stratix III, and Stratix IV | Stratix V |
| DDR input register | ✓ | — *(2)* | ✓ | ✓ |
| DDR output register | ✓ | ✓ | ✓ | ✓ |
| Synchronization registers | ✓ | — *(2)* | ✓ | ✓ *(4)* |
| Alignment registers | — | — | ✓ *(3)* | ✓ *(4)* |
| Half-rate data registers | — *(2)* | — *(2)* | ✓ | ✓ |
| DQS phase-shift circuitry | ✓ | — | ✓ | ✓ |
| DQS postamble circuitry | ✓ | — | ✓ | ✓ |
| Differential DQS signaling | ✓ | — | ✓ | ✓ |
| Read and write leveling circuitry | — | — | ✓ *(3)* | ✓ |
| Dynamic on-chip termination (OCT) control | — | — | ✓ | ✓ |
| Clock divider | — | — | ✓ | ✓ |
| Programmable delay | ✓ | ✓ | ✓ | ✓ |
| PLL | ✓ | ✓ | ✓ | ✓ |
| DLL | ✓ | — | ✓ | ✓ |

**Table 1–1. Device IOE Dedicated Features *(Note 1)* (Part 2 of 2)**

| Features | Device Family | | | |
|---|---|---|---|---|
| | Arria II GX | Cyclone III and Cyclone IV | Arria II GZ, Stratix III, and Stratix IV | Stratix V |
| Read FIFO | — | — | — | ✓ |

**Notes to Table 1–1:**

(1) For more information, refer to the *Device I/O Features* chapter in the relevant device handbook.
(2) Implemented in the FPGA core fabric.
(3) This circuitry is not applicable for Arria II GZ devices.
(4) This register is not available for Altera megafunctions that use Stratix V devices.

To use these features implement one of the Altera high-performance controllers (a complete solution) or Altera PHY IP.

Alternatively, you may access these IOE features directly using the following low-level megafunctions:

- ALTDQ_DQS and ALTDQ_DQS2 megafunctions—allow you to parameterize the following features:
  - DDR I/O registers
  - Alignment and synchronization registers
  - Half data rate registers
  - DQS bus mode
- ALTDLL megafunction—allows you to parameterize the DQS phase-shift circuitry
- ALTOCT megafunction—allows you to parameterize the IOE OCT features.
- ALTPLL megafunction—allows you to parameterize the device PLL
- ALTIOBUF megafunction—allows you to parameterize the device IO features

Altera offers no support using these low-level megafunctions as an external memory PHY to implement your memory interfaces. If you use low-level megafunctions, you are solely responsible in creating every aspect of the interface, including timing analysis and debugging.

## DDR Input and Output Registers

Arria II GZ, Stratix III, and Stratix IV devices provide DDR input and output registers on all sides. In Arria II GX, Cyclone IV, and Stratix V devices, the left side of the device is not available for interfacing. The DDR I/O structures can be directly implemented in the IOE in these devices, thus saving core logic and ensuring tight skew is easily maintained, which eases timing.

For Cyclone III and Cyclone IV devices, the DDR input registers are implemented in the core of the device. For Cyclone III and Cyclone IV devices, the read capture clock is derived from the PHY and is generated by the PLL to clock the DDR capture registers instead of using DQS read clock strobe from the memory device.

## Synchronization and Alignment Registers

Resynchronization registers resynchronize the data from memory clock domain to the memory controller system clock domain. Alignment registers align the data after read resynchronization or write leveling process.

☞ Altera IPs using Stratix V devices do not need synchronization and alignment registers.

In some devices, the synchronization registers are located in the core of the device, which makes the placement of these registers with respect to the DDR IOE critical to ensure that timing is achieved. Stratix III and Stratix IV devices have been enhanced to include the alignment and synchronization registers directly within the IOE, hence timing is now significantly improved and you are no longer concerned with ensuring critical register placement with respect to the DDR IOE. Typically, the resynchronization register is clocked via a dedicated output from the PLL. However, it may also be clocked directly from the read-leveling delay chain. The output alignment registers are typically clocked from the PLL.

If the resynchronization clock is sourced from the leveling delay chain, it may be cascaded from bank to bank, say 1A to 1B. In this configuration, memory controllers must form a single contiguous block of DQS groups that are not staggered or interleaved with another memory controller. Additionally, two PHYs cannot share the same subbank as only one leveling delay chain exists per subbank.

Arria II, Cyclone III, and Cyclone IV devices do not have leveling circuitry, so there is no need for alignment registers. Synchronization registers for Arria II devices are implemented in the IOE. These synchronization registers in Cyclone III and Cyclone IV devices are implemented in the FPGA core fabric and are clocked directly by the PLL. In Cyclone III and Cyclone IV devices, these registers are clocked by the same PLL output clock that also clocks the DDR registers.

☞ Generally, alignment and synchronization registers are optional and can be bypassed if not required. However, Altera external memory interface IP always implements these synchronization registers, regardless of interface speed. Hence latency through the PHY may not be optimal for lower frequency designs.

## Half-Rate Data Registers

As external memory interface clock speeds increase, the core $f_{MAX}$ can become the limiting factor in interface design. A common solution, which increases core $f_{MAX}$ timing problems, is to implement a half-rate architecture. This solution doubles the data width on the core side interfaces compared to a full-rate SDR solution, but also halves the required operating frequency.

Arria II GZ, Stratix III, Stratix IV, and Stratix V devices include dedicated full-rate to half-rate registers within the IOE.

Arria II GX, Cyclone III, and Cyclone IV devices implement half-rate registers in the core of the device.

## DQS Phase-Shift Circuitry

Devices that use DQS or CQ pins to clock the read data during read operation offer DQS phase-shift circuitry. This circuitry phase shifts the DQS and $CQ_n$ pins during the transaction, to obtain optimal read capture margin. DQS phase-shift circuitry consists of a DLL and phase offset control block, to further fine tune the DQS phase shift.

Cyclone III and Cyclone IV devices do not have this feature, because DQS signals are not needed during read operations at lower frequencies.

## DQS Postamble Circuitry

DQS postamble circuitry eliminates invalid DQ data capturing, because of the postamble glitches on the DQS signals through an override on DQS. This feature ensures the correct clock cycle timing of the postamble enable signal.

Altera devices only need DQS postamble circuitry to use the DQS scheme in read operation for clocking read data. Arria II, Stratix III, Stratix IV, and Stratix V devices have this feature.

As Cyclone III and Cyclone IV devices do not use DQS for capturing read data, they do not have this circuitry.

## Differential DQS Signaling

Altera devices (except Cyclone III and Cyclone IV devices) directly support differential DQS signalling and the single-ended standard supported in previous device families. DDR SDRAM only supports single-ended DQS, DDR2 SDRAM additionally includes the option of differential DQS signaling. DDR3 SDRAM only supports differential DQS signaling.

Differential DQS signaling is recommended for DDR2 SDRAM designs operating at or above 333 MHz. Differential DQS strobe operation enables improved system timing due to reduced crosstalk and less simultaneous switching noise on the strobe output drivers. You can use single-ended DQS mode for DDR2 SDRAM interfaces, but it requires more pessimistic timing data and hence results in less system timing margin.

Cyclone III and Cyclone IV devices do not support differential signalling and thus do not support DDR3 SDRAM, which requires DQS signaling.

## Read and Write Leveling

Stratix III, Stratix IV, and Stratix V I/O registers include read and write leveling circuitry to enable skew to be removed or applied to the interface on a DQS group basis. There is one leveling circuit located in each I/O subbank.

☞ ALTMEMPHY-based designs for DDR and DDR2 SDRAM (and DDR3 SDRAM without leveling) do not use leveling circuitry, as it is not needed by the memory standard. ALTMEMPHY-based designs for DDR3 SDRAM DIMMs (and components with fly-by topology like DIMMs) use leveling circuitry.

☞ UniPHY-based designs do not require read leveling circuitry during read leveling operation.

## Dynamic OCT

Arria II GZ, Stratix III, Stratix IV, and Stratix V devices support dynamic calibrated OCT. This feature allows the specified series termination to be enabled during writes, and parallel termination to be enabled during reads. These I/O features allow you to greatly simplify PCB termination schemes.

## Clock Divider

To ease data alignment, a single I/O clock divider may be used for an entire interface, as the half-rate resynchronization clock can be cascaded from DQ group to the adjacent DQ group. Hence, when using a common I/O clock divider, data alignment may be performed across the entire interface. Individual I/O clock dividers require the data alignment to be performed on a DQ group basis.

Arria II GZ, Stratix III, Stratix IV, and Stratix V devices include a dedicated I/O clock divider on a per DQS group basis, to simplify and reduce the number of clocks required. The output of this clock divider can then directly source the half-rate resynchronization clock from the full-rate version.

Arria II GX, Cyclone III, and Cyclone IV devices do not have the clock divider feature, and so must use separate PLL output.

ALTMEMPHY-based designs support both balanced (without leveling) and unbalanced (with leveling) CAC topologies. ALTMEMPHY-based designs with leveling designs use multiple I/O clock dividers on a DQ group basis and do not support balanced CAC topologies, as a dedicated resynchronization and half-rate resynchronization clock is required on a per DQS group basis. ALTMEMPHY-based designs without leveling designs use a single I/O clock divider for the whole interface to reduce PHY complexity and reduce latency. However, ALTMEMPHY-based without leveling interfaces cannot be interleaved, because of FPGA limitations.

In UniPHY-based designs, a clock divider is instantiated in the core of the device for each DQS groups.

## Programmable Delay

I/O registers include programmable delay chains that you may use to deskew interfaces. Each pin can have different delay settings, hence read and write margins can be increased as uncertainties between signals can be minimized.

The DQ-DQS offset scheme is applicable for interfacing systems using QDR II and QDR II+ SRAM, RLDRAM II, DDR, DDR2 and DDR3 SDRAM components for frequencies of 400 MHz and below. For interfacing system using DDR3 SDRAM components with frequencies above 400 MHz, a dynamic deskew scheme improves the timing margins.

For DQ-DQS offset schemes, delay-chains in the IOE shift the offset between DQ and DQS, to meet timing. The offsets employed are FPGA family, speed grade and frequency dependent.

For systems using DDR3 SDRAM interfaces at frequencies above 400 MHz, the timing margins are too small to be fixed by the static DQ-DQS offset scheme. Hence, a dynamic scheme improves the setup and hold margin at the memory component with the DLL set to 8-tap mode. Configurable delay elements and delay-chains in the IOE observe the write window in the DDR3 SDRAM components. This information configures the delays for each individual DQ and DM pins, to meet the memory component setup and hold requirements and reduce skew between DQ and DM pins in a DQS group.

## Read FIFO

Read FIFO buffer is only available for Stratix V devices. Read FIFO buffer is added to the Stratix V I/O element (IOE) to resynchronize captured data from the capture clock domain to the system clock domain. The resynchronization helps improve the performance of the Stratix V devices, and removes the need for synchronization and alignment registers.

## PLL and DLL

Altera devices use PLLs to generate the memory controller clocks. The simplest slowest speed memory controllers may only require two clocks (0° system clock and –90° write clock). However, as interface speeds increase, it becomes harder to close timing and so dedicated resynchronization, postamble, and address and command clocks are typically required. Additionally, at higher frequencies the maximum frequency becomes the bottleneck and half-rate designs are the typical solution. Thus complex half high data rate designs require typically 10 clock networks. Altera devices are well equipped to address the clocking requirements of external memory interfaces. This topic describes the availability of PLL and DLL for each of the Altera devices.

### Arria II GX Devices

Arria II GX devices offer up to six PLLs per device with seven outputs per PLL. Each corner of the device has one PLL. Another two PLLs are located at the middle of the right side of the device.

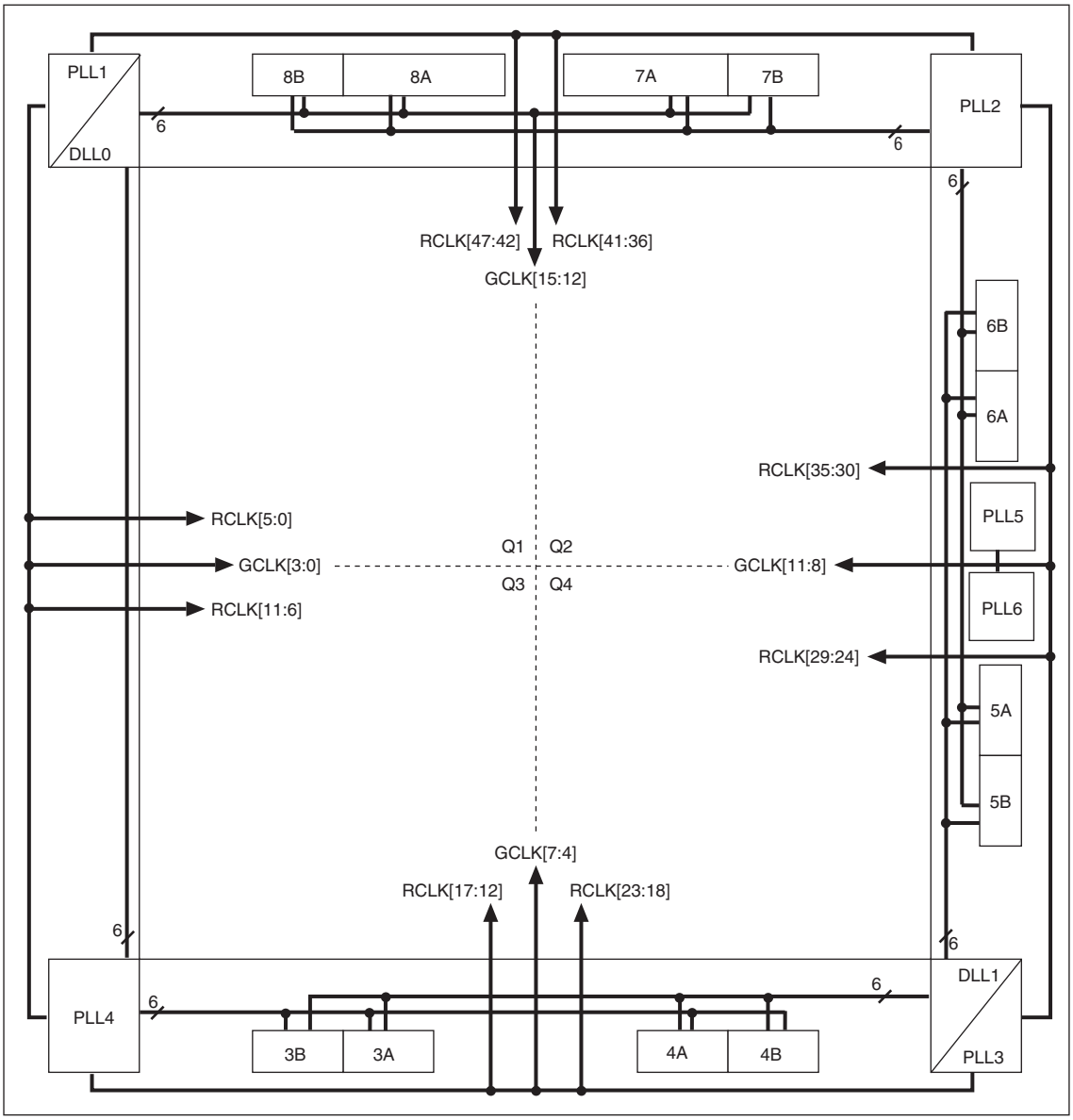
There is a total maximum of 64 clock networks provided in Arria II GX devices. 16 global clock networks and 48 regional clock networks. Each corner PLL has access to the 8 global clocks and 24 regional clocks. Each middle PLL has access to 4 global clocks and 12 regional clocks.

Arria II GX devices support two DLLs. They are located in the top-left and bottom-right corners of the device. These two DLLs can support a maximum of two different frequencies, with each DLL running at one frequency. Each DLL can have two outputs with different phase offsets, which allows one Arria II GX device to have four different DLL phase-shift settings. Each DLL can access the top, bottom, and right side of the device. Each I/O bank is accessible by two DLLs, giving more flexibility to create multiple frequencies and multiple-type interfaces. The DLL outputs the same DQS delay settings for the different sides of the device.

For more information, refer to the *Clock Networks and PLLs in Arria II GX Devices* chapter and the *External Memory Interfaces* chapter in the *Arria II GX Device Handbook*.

Figure 1–9 shows the PLL and DLL locations in Arria II GX devices.

**Figure 1–9. PLL and DLL Locations in Arria II GX Devices**




Consider the following points:

- Each corner PLL in Arria II GX devices connects to eight global clock nets and 24 regional clock nets.
- Center PLLs in Arria II GX devices connect to four maximum global clock nets and 12 regional clock nets.
- Arria II GX devices have two PLLs at the middle of right side of the device and one PLL at each of the corners of the device.
- Dual regional clock nets are created by using a regional clock net from each region. For example, a single dual regional clock net uses two regional clock nets.

- If the design uses a dedicated PLL to only generate a DLL input reference clock, the PLL mode must be set to **No Compensation** or the Quartus® II software forces this setting automatically. No compensation mode is used to minimize jitter. The PLL does not generate other outputs, so it does not need to compensate for any clock paths.
- If the design cascades PLLs, the source (upstream) PLL must have a low-bandwidth setting, while the destination (downstream) PLL must have a high-bandwidth setting.
- In Arria II GX devices, PLL_5 and PLL_6 on the right side may be cascaded to each other as you can then use the direct connection between these two PLLs instead of going through the clock network. The direct connection is required when you have to cascade PLLs for external memory interface to reduce the output clock jitter from the cascaded PLL. Cascaded PLLs are not recommended for external memory interface designs, as jitter can accumulate with the use of cascaded PLLs. The memory output clock may violate the memory device jitter specification.
- Input and output delays are only fully compensated for when the dedicated clock input pins associated with that specific PLL are used as the clock source.
- If the clock source for the PLL is not a dedicated clock pin for that specific PLL, jitter is increased, timing margin suffers, and the design may require an additional global or regional clock. Hence, dedicated PLL input clock pin is strongly commended for clock source for the PLL.

The following additional IP-specific points apply:

- ALTMEMPHY megafunctions require five global clock nets in Arria II GX devices; UniPHY IP requires three global clock networks.
- Ideally, you must pick a PLL and PLL input clock pin that are located on the same side of the device as the memory interface pins.
- You may also share the DLL and PLL static clocks for multiple memory interfaces provided the controllers are on the same side or adjacent side of the device and running at the same memory clock.
- If a single memory interface spans two right-side quadrants, a middle-side PLL must be the source for that interface.

For more information about clock networks, refer to the *Clock Networks and PLLs in Arria II GX Devices* chapter in volume 1 of the *Arria II GX Device Handbook*.

## Cyclone III and Cyclone IV Devices

There are a maximum of four PLLs in Cyclone III and Cyclone IV devices with five PLL clock outputs each. Each PLL is located in each corner of the device and drives up to 10 global clock networks. Cyclone III and Cyclone IV devices only have global clock network and offer up to 20 global clock networks.

Cyclone III and Cyclone IV devices do not have DLL circuitry. Cyclone III and Cyclone IV devices use a PLL clock generated by the PHY to clock the read capture register during the read operation. The DQS strobe from the memory component is ignored during the read operation.

For more information, refer to the *Clock Networks and PLLs in Cyclone III Devices* chapter and the *External Memory Interfaces* chapter in the *Cyclone III Device Handbook*.

Figure 1–10 shows the PLL locations and resources in Cyclone III and Cyclone IV E devices.

**Figure 1–10. PLL Locations in Cyclone III and Cyclone IV E Devices**

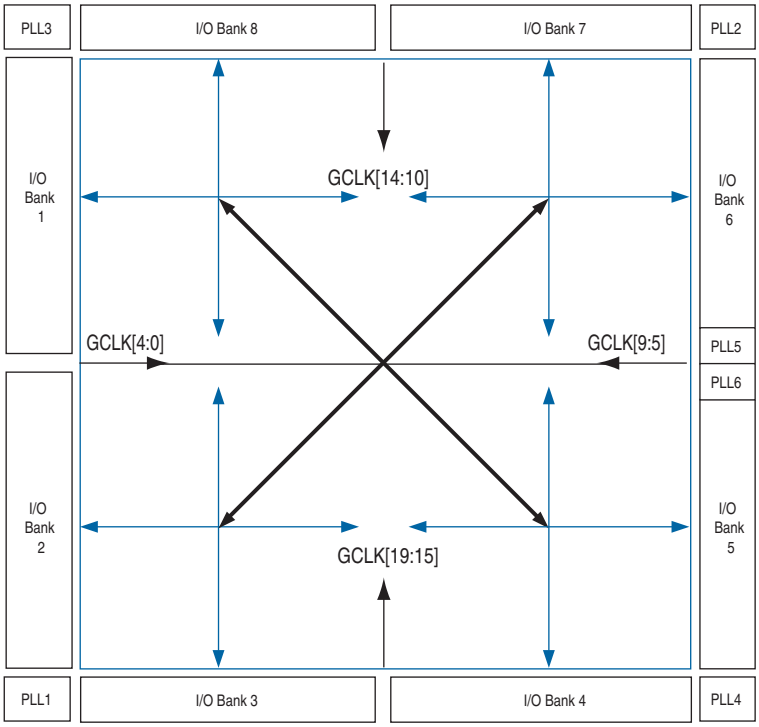
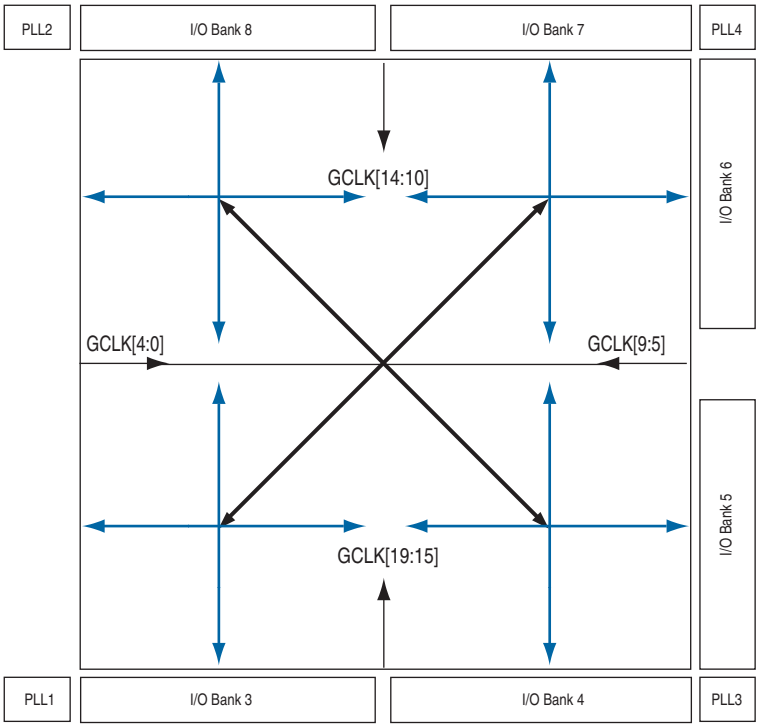


Figure 1–11 shows the PLL locations in Cyclone IV GX devices.

**Figure 1–11. PLL and DLL Locations in Cyclone IV GX Devices**




Consider the following points:

- Cyclone III and Cyclone IV devices only support global clock networks.
- Each PLL in Cyclone III and Cyclone IV devices connects to a maximum of ten global clock networks.
- Cyclone III and Cyclone IV devices have maximum of four PLLs. Each PLL is located at each corner of the device.
- You do not need to set the PLL in **With No Compensation** mode as the jitter for Cyclone III and Cyclone IV PLLs in normal mode is low. Changing the PLL compensation mode may result in inaccurate timing results.
- Cascading PLLs is not supported in Cyclone III and Cyclone IV devices.
- Input and output delays are only fully compensated for when the dedicated clock input pins associated with that specific PLL are used as the clock source.
- Ensure the clock source for the PLL is a dedicated clock pin for that specific PLL, to reduce jitter, improve timing margins, and to avoid the design requiring an additional global clock.

The following additional ALTMEMPHY megafunction-specific points apply:

- ALTMEMPHY megafunctions require four global clock nets in Cyclone III and Cyclone IV devices.

- Any PLL on any side of an Cyclone III or Cyclone IV device can support an ALTMEMPHY interface. Ideally, you must pick a PLL and PLL input clock pin that are located on the same side of the device as the memory interface pins.
- You may also share the PLL static clocks for multiple ALTMEMPHY interface, provided the controllers are on the same side or adjacent side of the device and running at the same memory clock.

For information about enabling transceiver operations for Cyclone IV GX designs, refer to the *Cyclone IV Device Family Pin Connection Guidelines*.

## Arria II GZ, Stratix III, and Stratix IV Devices

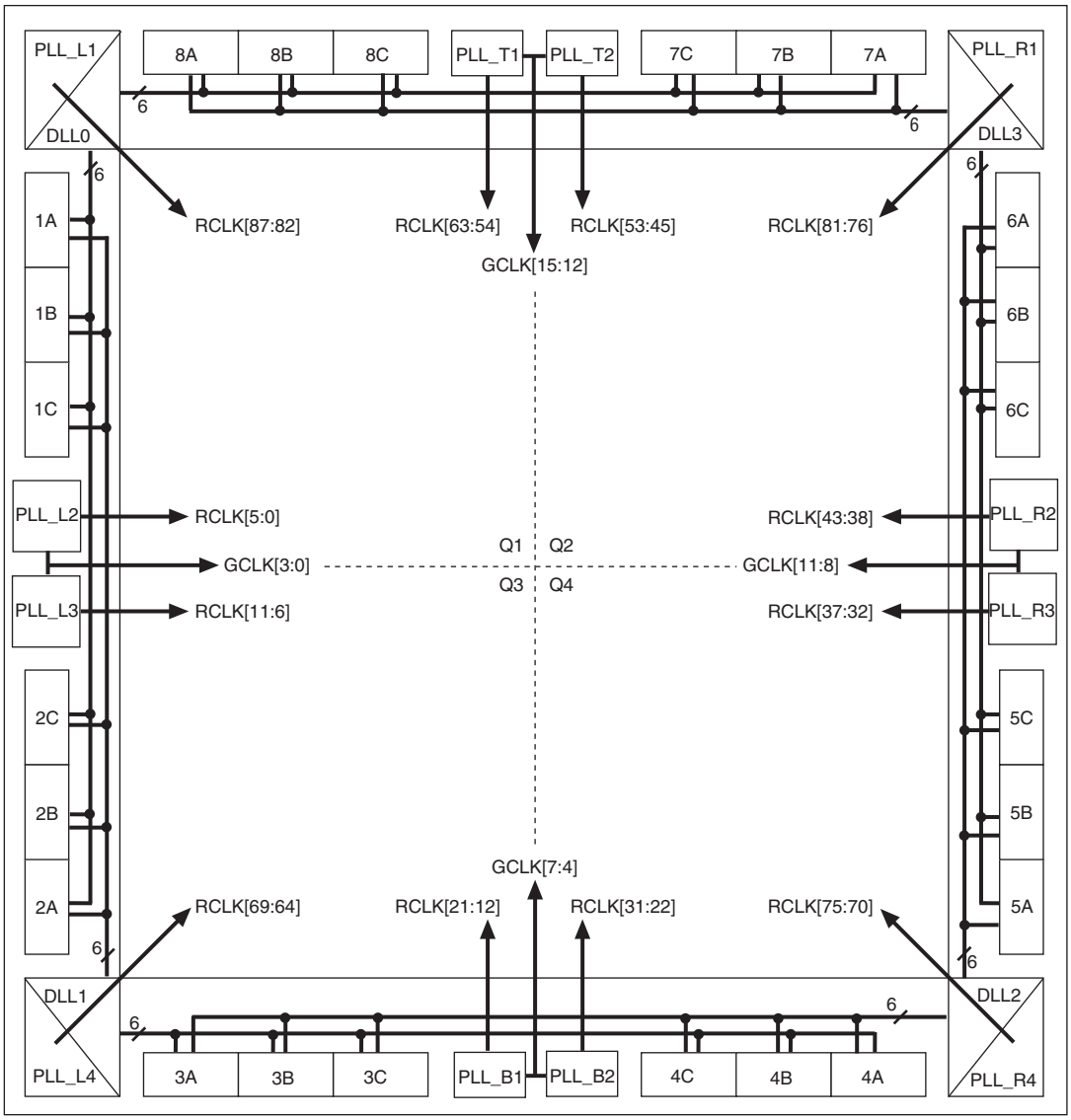
Arria II GZ, Stratix III, and Stratix IV PLLs have an increased number of outputs and global clock routing resources when compared to earlier device generations. Arria II GZ, Stratix III, and Stratix IV top and bottom PLLs feature 10 output (C) counters; left and right PLLs feature 7 output (C) counters. This increased number of PLL outputs allows for the use of dedicated clock phases. In previous Stratix II designs, clock phases had to be shared.

In general, each Arria II GZ, Stratix III, or Stratix IV PLL has access to four global clocks (GCLK) and six regional clocks (RCLK) (left and right) or ten RCLK (top and bottom).

Arria II GZ, Stratix III, and Stratix IV devices also feature four DLLs (one located in each corner of the device). Thus the FPGA can support a maximum of four unique frequencies, with each DLL running at one frequency. Each DLL can also support two different phase offsets, which allow a single Arria II GZ, Stratix III, or Stratix IV device to support eight different DLL phase shift settings. Additionally, each DLL can access the two sides adjacent to its location. Thus each I/O bank is accessible by two different DLLs, giving more flexibility when creating multiple frequency and phase shift memory interfaces.

Figure 1–12 shows PLL and DLL locations in Arria II GZ, Stratix III, and Stratix IV devices with global and regional clock resources.

**Figure 1–12. PLL and DLL Locations in Arria II GZ, Stratix III, and Stratix IV Devices**




Consider the following points:

- Each PLL connects to four maximum global clock nets.
- Top or bottom PLLs connect to ten maximum regional clock nets.
- Left or right PLLs connect to six maximum regional clock nets.
- EP4S…110 and smaller devices have only one PLL located in the middle of each side of the device
- EP3S...80 and larger devices and EP4S…180 and larger devices have two PLLs in the middle of each side of the device.

- EP3S...200 and larger devices, and also EP4S…290 and larger devices additionally have corner PLLs, which connect to six regional clock nets only. All Arria II GZ devices have two middle PLLs on each side and four corner PLLs, except for the EP2AGZ225 device, which does not have corner PLLs.
- The IP core creates dual regional clock nets by using a regional clock net from each region. For example, a single dual regional clock net uses two regional clock nets.
- If the design uses a dedicated PLL to only generate a DLL input reference clock, the PLL mode must be set to **No Compensation**, or the Quartus II software forces this setting automatically.
- If the design cascades PLLs, the source (upstream) PLL should have a low-bandwidth setting, while the destination (downstream) PLL should have a high-bandwidth setting.
- If the design requires you to cascade two PLLs, use two adjacent PLLs, as you can use a direct connection instead of going through the clock network.. The direct connection is required when you have to cascade PLLs for external memory interface to reduce the output clock jitter from the cascaded PLL. Do not use cascaded PLLs for external memory interface designs, as jitter can accumulate with the use of cascaded PLLs. The memory output clock may violate the memory device jitter specification.
- Input and output delays are only fully compensated for when the dedicated clock input pins associated with that specific PLL are used as its clock source.
- If the clock source for the PLL is not a dedicated clock pin for that specific PLL, jitter is increased, timing margin suffers, and the design may require an additional global or regional clock. Hence, a dedicated PLL input clock pin is recommended for the PLL clock source.

The following additional IP specific points apply for Arria II GZ, Stratix III, and Stratix IV devices:

- ALTMEMPHY megafunctions require one global or regional clock, and six regional clock nets. Hence seven clock nets in total are required. RLDRAM II controller with UniPHY requires one global clock net and four regional clock nets; QDR II or QDR II+ SRAM with controller UniPHY requires three regional clock nets; DDR2 or DDR3 SDRAM controller with UniPHY requires three global clock nets and four regional clock nets.
- Ideally, you should pick a PLL and a PLL input clock pin that are located on the same side of the device as the memory interface pins.
- You may also share the DLL and PLL static clocks for multiple memory interfaces, provided the controllers are on the same side or adjacent side of the device and running at the same memory clock.
- As each PLL can only connect to four global clock nets, while the memory IP requires more than four clock nets, an external memory interface design cannot cross from one side of a device to the other side. For example, an external memory interface design can only exist within a dual regional side of a device.
- If a single memory interface spans two top or bottom quadrants, a middle top or bottom PLL must be the source for that interface. The ten dual region clocks that the single interface requires should not block the design using the adjacent PLL (if available) for a second interface.

For more information on clock networks, refer to *Clock Networks and PLLs in Stratix III Devices* in the *Stratix III Device Handbook* and *Clock Networks and PLLs in Stratix IV Devices* in the *Stratix IV Device Handbook*.

For more information on multiple memory controllers, refer to *Volume 6: Design Flow Tutorials* of the *External Memory Interface Handbook*.

## Stratix V Devices

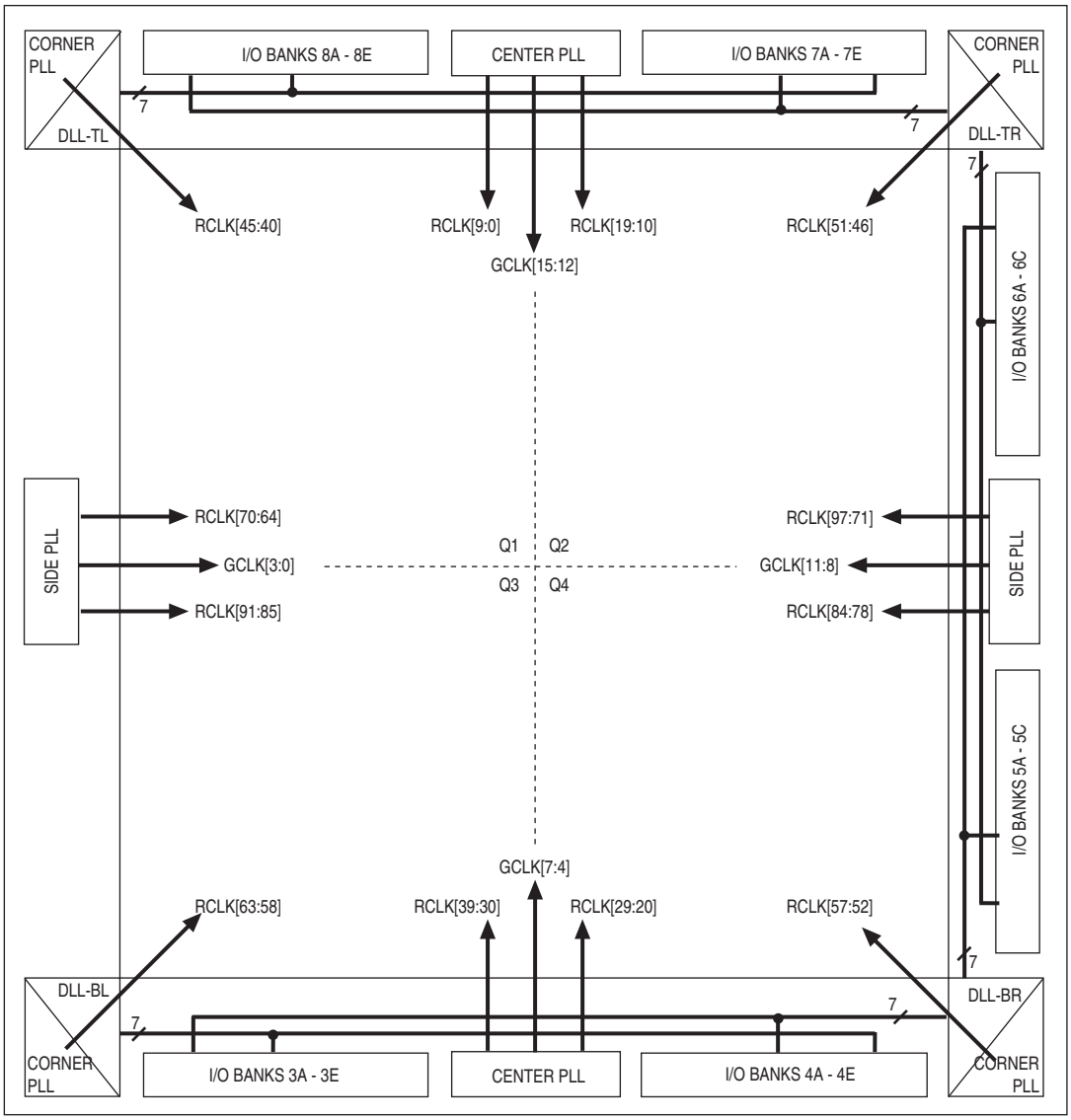
In addition to the existing integer PLL, Stratix V devices offer fractional PLL (fPLL). fPLL allows the output frequency to multiply with an integer and a fraction, providing smaller step sizes. Stratix V devices offer up to 28 fPLLs in the largest densities, with 18 output counters for each fPLL.

You can use fPLL as a single PLL with 18 outputs, or as two PLLs sharing the 18 outputs. The increased number of PLL outputs allows for the use of dedicated clock phases. Bottom center and top center PLLs have access to twelve global clocks (GCLK), while the rest of the PLLs have access to eight GCLKs.

Stratix V devices have four DLLs, one located in each corner of the device. The FPGA supports a maximum of four unique frequencies, with each DLL running at one frequency. Each DLL supports two different phase offsets, that allow a single Stratix V device to support eight different DLL phase shift settings. Each DLL also have access to the two sides adjacent to its location. Each I/O bank is accessible by two different DLLs, giving more flexibility when creating multiple frequency and phase shift memory interfaces.

Figure 1–13 shows PLL and DLL locations in Stratix V devices with global and regional clock resources.

**Figure 1–13. PLL and DLL Locations in Stratix V Devices**




For more information, refer to the Clock Networks and PLLs in Stratix V Devices chapter and the External Memory Interfaces chapter in the Stratix V Device Handbook.

Consider the following points:

- Dual regional clock nets are created by using a regional clock net from each region. For example, a single dual regional clock net uses two regional clock nets.
- If the design uses a dedicated PLL to only generate a DLL input reference clock, the PLL mode must be set to **No Compensation**, or the Quartus II software forces this setting automatically.

- If the design cascades PLLs, the source (upstream) PLL should have a low-bandwidth setting, while the destination (downstream) PLL should have a high-bandwidth setting.
- If you are required to cascade two PLLs, use two adjacent PLLs, as you can use a direct connection instead of going through the clock network.. The direct connection is required when you have to cascade PLLs for external memory interface to reduce the output clock jitter from the cascaded PLL. Cascaded PLLs are not recommended for external memory interface designs, as jitter can accumulate with the use of cascaded PLLs. The memory output clock may violate the memory device jitter specification.
- Input and output delays are only fully compensated for, when the dedicated clock input pins associated with that specific PLL are used as its clock source.
- If the clock source for the PLL is not a dedicated clock pin for that specific PLL, jitter is increased, timing margin suffers, and the design may require an additional global or regional clock. Hence, a dedicated PLL input clock pin is recommended for the PLL clock source.

The following additional IP specific points apply for Stratix V devices:

- UniPHY IP requires three global clock nets and four regional clock nets.
- Ideally, you should pick a PLL and a PLL input clock pin that are located on the same side of the device as the memory interface pins.
- You may also share the DLL and PLL static clocks for multiple memory interfaces, provided the controllers are on the same side or adjacent side of the device and running at the same memory clock.
- If a single memory interface spans two top or bottom quadrants, a middle top or bottom PLL must be the source for that interface. The ten dual region clocks that the single interface requires should not block the design using the adjacent PLL (if available) for a second interface.

For more information on clock networks, refer to the *Clock Networks and PLLs in Stratix V Devices* chapter in volume 1 of the *Stratix V Device Handbook*. For more information on multiple memory controllers, refer to *Volume 6: Design Flow Tutorials* of the *External Memory Interface Handbook*.

# 2. Planning Pin and Resource

This chapter is for board designers who need to determine the FPGA pin usage, to create the board layout for the system, as the board design process sometimes occurs concurrently with the RTL design process.

Use this document with the *External Memory Interfaces* chapter of the relevant device family handbook.

All external memory interfaces typically require the following FPGA resources:

- I/O pins
- PLL and clock network
- DLL (not applicable in Cyclone III and Cyclone IV devices)
- Other FPGA resources—for example, core fabric logic, and on-chip termination (OCT) calibration blocks

When you know the requirements for your memory interface, you can then start planning how you can architect your system. The I/O pins and internal memory cannot be shared for other applications or memory interfaces. However, if you do not have enough PLLs, DLLs, or clock networks for your application, you may share these resources among multiple memory interfaces or modules in your system.

Ideally, any interface should wholly reside in a single bank. However, interfaces that span multiple adjacent banks or the entire side of a device are also fully supported. In addition, you may also have wraparound memory interfaces, where the design uses two adjacent sides of the device and the memory interface logic resides in a device quadrant. In some cases, top or bottom bank interfaces have higher supported clock rate than left or right or wraparound interfaces.

For the maximum clock rate supported for your memory interface, refer to the *External Memory Interface System Specifications* section in volume 1 of the *External Memory Interface Handbook*.

For information about the terms used in this document, refer to the *Glossary* chapter in volume 1 of the *External Memory Interface Handbook*.

## I/O Pins

Any I/O banks that do not support transceiver operations in Arria II, Cyclone III, Cyclone IV, Stratix III, Stratix IV, and Stratix V devices, support memory interfaces. However, DQS (data strobe or data clock) and DQ (data) pins are listed in the device pin tables and fixed at specific locations in the device. You must adhere to these pin locations as these locations are optimized in routing to minimize skew and maximize margin. Always check the external memory interfaces chapters for the number of DQS and DQ groups supported in a particular device and the pin table for the actual locations of the DQS and DQ pins.

For maximum performance and best skew across the interface, each required memory interface should completely reside within a single I/O bank, or at least one side of the device. Address and command pins can be constrained in a different side of the device if there are not enough pins available. For example, you may have the read and write data pins on the top side of the device, and have the address and command pins on the left side of the device. In memory interfaces with unidirectional data, you may also have all the read data pins on the top side of the device and the write data pin on the left side of the device. However, you should not break a unidirectional pin group across multiple sides of the device. Memory interfaces typically have the following pin groups:

- Write data pin group and read data pin group
- Address and command pin group

Table 2–1 shows a summary of the number of pins required for various example memory interfaces. Table 2–1 uses series OCT with calibration, parallel OCT with calibration, or dynamic calibrated OCT, when applicable, shown by the usage of $R_{UP}$ and $R_{DN}$ pins or $R_{ZQ}$ pin.

For the memory device pin description, refer to the *Memory Standard Overview* section in volume 1 of the *External Memory Interface Handbook*.

**Table 2–1. Pin Counts for Various Example Memory Interfaces *(Note 1)*, *(2)* (Part 1 of 2)**

| Memory Interface | FPGA DQS Bus Width | Number of DQ Pins | Number of DQS Pins | Number of DM/BWSn Pins | Number of Address Pins *(3)* | Number of Command Pins | Number of Clock Pins | $R_{UP}/R_{DN}$ Pins *(4)* | $R_{ZQ}$ Pins *(11)* | Total Pins with $R_{UP}/R_{DN}$ | Total Pins with $R_{ZQ}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| DDR3 SDRAM *(5)*, *(6)* | ×4 | 4 | 2 | 0 *(7)* | 14 | 10 | 2 | 2 | 1 | 34 | 33 |
| | ×8 | 8 | 2 | 1 | 14 | 10 | 2 | 2 | 1 | 39 | 38 |
| | | 16 | 4 | 2 | 14 | 10 | 2 | 2 | 1 | 50 | 49 |
| | | 72 | 18 | 9 | 14 | 14 | 4 | 2 | 1 | 134 | 133 |
| DDR2 SDRAM *(8)* | ×4 | 4 | 1 | 1 *(7)* | 15 | 9 | 2 | 2 | 1 | 34 | 33 |
| | ×8 | 8 | 1 *(9)* | 1 | 15 | 9 | 2 | 2 | 1 | 38 | 37 |
| | | 16 | 2 *(9)* | 2 | 15 | 9 | 2 | 2 | 1 | 48 | 47 |
| | | 72 | 9 *(9)* | 9 | 15 | 12 | 6 | 2 | 1 | 125 | 124 |
| DDR SDRAM *(6)* | ×4 | 4 | 1 | 1 *(7)* | 14 | 7 | 2 | 2 | 1 | 29 | 28 |
| | ×8 | 8 | 1 | 1 | 14 | 7 | 2 | 2 | 1 | 33 | 35 |
| | | 16 | 2 | 2 | 14 | 7 | 2 | 2 | 1 | 43 | 42 |
| | | 72 | 9 | 9 | 13 | 9 | 6 | 2 | 1 | 118 | 117 |
| QDR II+ SRAM | ×9 | 18 | 2 | 1 | 19 | 3 *(10)* | 4 | 2 | 1 | 49 | 48 |
| | ×18 | 36 | 2 | 2 | 18 | 3 *(10)* | 4 | 2 | 1 | 67 | 66 |
| | ×36 | 72 | 2 | 4 | 17 | 3 *(10)* | 4 | 2 | 1 | 104 | 103 |
| QDR II SRAM | ×9 | 18 | 2 | 1 | 19 | 2 | 4 | 2 | 1 | 48 | 47 |
| | ×18 | 36 | 2 | 2 | 18 | 2 | 4 | 2 | 1 | 66 | 65 |
| | ×36 | 72 | 2 | 4 | 17 | 2 | 4 | 2 | 1 | 103 | 102 |

**Table 2–1. Pin Counts for Various Example Memory Interfaces *(Note 1)*, *(2)* (Part 2 of 2)**

| Memory Interface | FPGA DQS Bus Width | Number of DQ Pins | Number of DQS Pins | Number of DM/BWSn Pins | Number of Address Pins *(3)* | Number of Command Pins | Number of Clock Pins | $R_{UP}/R_{DN}$ Pins *(4)* | $R_{ZQ}$ Pins *(11)* | Total Pins with $R_{UP}/R_{DN}$ | Total Pins with $R_{ZQ}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **RLDRAMII CIO** | ×9 | 9 | 2 | 1 | 22 | 7 *(10)* | 4 | 2 | 2 | 47 | 46 |
| | | 18 | 2 | 1 | 21 | 7 *(10)* | 6 | 2 | 2 | 57 | 56 |
| | ×18 | 36 | 2 | 1 | 20 | 7 *(10)* | 8 | 2 | 2 | 76 | 75 |
| **RLDRAM II SIO** | ×9 | 18 | 2 | 1 | 22 | 7 *(10)* | 4 | 2 | 2 | 56 | 55 |
| | | 36 | 2 | 1 | 21 | 7 *(10)* | 6 | 2 | 2 | 75 | 74 |
| | ×18 | 72 | 2 | 1 | 20 | 7 *(10)* | 8 | 2 | 2 | 112 | 111 |

**Notes to Table 2–1:**

(1) These example pin counts are derived from memory vendor data sheets. Check the exact number of addresses and command pins of the memory devices in the configuration that you are using.

(2) PLL and DLL input reference clock pins are not counted in this calculation.

(3) The number of address pins depend on the memory device density.

(4) Some DQS or DQ pins are dual purpose and can also be required as $R_{UP}$, $R_{DN}$, or configuration pins. A DQS group is lost if you use these pins for configuration or as $R_{UP}$ or $R_{DN}$ pins for calibrated OCT. Pick $R_{UP}$ and $R_{DN}$ pins in a DQS group that is not used for memory interface purposes. You may need to place the DQS and DQ pins manually if you place the $R_{UP}$ and $R_{DN}$ pins in the same DQS group pins.

(5) The `TDQS` and `TDQS#` pins are not counted in this calculation, as these pins are not used in the memory controller.

(6) Numbers are based on 1-GB memory devices.

(7) Altera FPGAs do not support DM pins in ×4 mode with differential DQS signaling.

(8) Numbers are based on 2-GB memory devices without using differential DQS, `RDQS`, and `RDQS#` pin support.

(9) Assumes single ended DQS mode. DDR2 SDRAM also supports differential DQS, which makes these DQS and DM numbers identical to DDR3 SDRAM.

(10) The QVLD pin that indicates read data valid from the QDR II+ SRAM or RLDRAM II device, is included in this number.

(11) $R_{ZQ}$ pins are supported by Stratix V devices only.

For more information on the following topics, refer to the *External Memory Interface* chapter of the relevant FPGA family handbook.

Maximum interface width varies from device to device depending on the number of I/O pins and DQS or DQ groups available. Achievable interface width also depends on the number of address and command pins that the design requires. To ensure adequate PLL, clock, and device routing resources are available, you should always test fit any IP in the Quartus II software before PCB sign-off.

Altera devices do not limit the width of external memory interfaces beyond the following requirements:

- Maximum possible interface width in any particular device is limited by the number of DQS groups available.
- Sufficient clock networks are available to the interface PLL as required by the IP.
- Sufficient spare pins exist within the chosen bank or side of the device to include all other address and command, and clock pin placement requirements.
- The greater the number of banks, the greater the skew, hence Altera recommends that you always generate a test project of your desired configuration and confirm that it meets timing.

While you should use the Quartus II software for final pin fitting, you can estimate whether you have enough pins for your memory interface using the following steps:

1. Find out how many read data pins are associated per read data strobe or clock pair, to determine which column of the DQS and DQ group availability (×4, ×8/×9, ×16/×18, or ×32/×36) look at the pin table.
2. Check the device density and package offering information to see if you can implement the interface in one I/O bank or on one side or on two adjacent sides.

   If you target Arria II GX, Cyclone III, or Cyclone IV devices and you do not have enough I/O pins to have the memory interface on one side of the device, you may place them on the other side of the device. These device families allow a memory interface to span across the top and bottom, or left and right sides of the device. For any interface that spans across two different sides, use the wraparound interface performance.

3. Calculate the number of other memory interface pins needed, including any other clocks (write clock or memory system clock), address, command, $R_{UP}$, $R_{DN}$, $R_{ZQ}$, and any other pins to be connected to the memory components. Ensure you have enough pins to implement the interface in one I/O bank or one side or on two adjacent sides.

   The DQS groups in Arria II GX devices reside on I/O modules, each consisting of 16 I/O pins. You can only use a maximum of 12 pins per I/O modules when the pins are used as DQS or DQ pins or HSTL/SSTL output or HSTL/SSTL bidirectional pins. When counting the number of available pins for the rest of your memory interface, ensure you do not count the leftover four pins per I/O modules used for DQS, DQ, address and command pins. The leftover four pins can be used as input pins only.

   Refer to the device pinout tables and look for the blank space in the relevant DQS group column to identify the four pins that cannot be used in an I/O module.

You should always try the proposed pinouts with the rest of your design in the Quartus II software (with the correct I/O standard and OCT connections) before finalizing the pinouts, as there may be some interactions between modules that are illegal in the Quartus II software that you may not find out unless you try compiling a design and use the Quartus II Pin Planner.

## Maximum Number of Interfaces

Table 2–2 through Table 2–6 show the available device resources for DDR, DDR2, DDR3 SDRAM, RLDRAM II, and QDR II and QDR II+ SRAM controller interfaces.

Unless otherwise noted, the calculation for the maximum number of interfaces is based on independent interfaces where the address or command pins are not shared. The maximum number of independent interfaces is limited to the number of PLLs each FPGA device has.

Timing closure depends on device resource and routing utilization. For more information about timing closure, refer to the *Area and Timing Optimization Techniques* chapter in the *Quartus II Handbook*.

You need to share DLLs if the total number of interfaces exceeds the number of DLLs available in a specific FPGA device. You may also need to share PLL clock outputs depending on your clock network usage, refer to the *Device and Pin Planning* section in volume 2 of the *External Memory Interface Handbook*.

For information on the number of DQ and DQS in other packages, refer to the DQ and DQS tables in the relevant device handbook.

Table 2–2 describes the maximum number of ×8 DDR SDRAM components fit in the smallest and biggest devices and pin packages assuming the device is blank.

Each interface of size *n*, where *n* is a multiple of 8, consists of:

- *n* DQ pins (including error correction coding (ECC))
- *n*/8 DM pins
- *n*/8 DQS pins
- 18 address pins
- 6 command pins (CAS, RAS, WE, CKE, reset, and CS)
- 1 CK, CK# pin pair for up to every three ×8 DDR SDRAM components

**Table 2–2. Maximum Number of DDR SDRAM Interfaces Supported per FPGA (Part 1 of 2)**

| Device | Device Type | Package Pin Count | Maximum Number of Interfaces |
|---|---|---|---|
| Arria II GX | EP2AGX190<br>EP2AGX260 | 1,152 | Four ×8 interfaces or one ×72 interface on each side (no DQ pins on left side) |
| | EP2AGX45<br>EP2AGX65 | 358 | ■ On top side, one ×16 interface<br>■ On bottom side, one ×16 interface<br>■ On right side (no DQ pins on left side), one ×8 interface |
| Arria II GZ | EP2AGZ300<br>EP2AGZ350 | F780 | ■ On top side, three ×8 interfaces or one ×64 interface<br>■ On bottom side, three ×8 interfaces or one ×64 interface<br>■ No DQ pins on the left and right sides |
| | EP2AGZ300<br>EP2AGZ350<br>EP2AGZ225 | F1,517 | Four ×8 interfaces or one ×72 interface on each side |
| Cyclone III | EP3C120 | 780 | ■ Three ×16 interfaces on top and bottom sides<br>■ Two ×16 interfaces on right and left sides |
| | EP3C5 | 256 | ■ Two ×8 interfaces on top and bottom sides<br>■ One ×8 interface on right and left sides |

**Table 2–2. Maximum Number of DDR SDRAM Interfaces Supported per FPGA (Part 2 of 2)**

| Device | Device Type | Package Pin Count | Maximum Number of Interfaces |
|---|---|---|---|
| Cyclone IV E | EP4CE115 | 780 | ■ One ×48 interface or two ×8 interfaces on top and bottom sides<br>■ Four ×8 interfaces on right and left sides |
| | EP4CE10 | 144 | On top side, one ×8 interface with address pins wrapped around the left or right side |
| Cyclone IV GX | EP4CGX150 | 896 | ■ One ×48 interface or four ×8 interfaces on top and bottom sides<br>■ On right side, three ×8 interfaces<br>■ No DQ pins on the left side |
| | EP4CGX22 | 324 | ■ One ×8 interface on top and bottom sides<br>■ On right side, one ×8 interface with address pins wrapped around the top or bottom side<br>■ No DQ pins on the left side |
| Stratix III | EP3SL340 | 1,760 | ■ Two ×72 interfaces on top and bottom sides<br>■ One ×72 interface on right and left sides |
| | EP3SE50 | 484 | ■ Two ×8 interfaces on top and bottom sides<br>■ Three ×8 interface on right and left sides |
| Stratix IV | EP4SGX290<br>EP4SGX360<br>EP4SGX530 | 1,932 | ■ One ×72 interface on each side<br>or<br>■ One ×72 interface on each side and two additional ×72 wraparound interfaces, only if sharing DLL and PLL resources |
| | EP4SE530<br>EP4SE820 | 1,760 | |
| | EP4SGX70<br>EP4SGX110<br>EP4SGX180<br>EP4SGX230 | 780 | ■ Three ×8 interfaces or one ×64 interface on top and bottom sides<br>■ On left side, one ×48 interface or two ×8 interfaces<br>■ No DQ pins on the right side |
| Stratix V | 5SGXA3<br>5SGXA4 | 780 | ■ On top side, two ×8 interfaces<br>■ On bottom side, four ×8 interfaces or one ×72 interface<br>■ No DQ pins on left and right sides |
| | 5SGXA5<br>5SGXA7 | 1,932 | ■ Three ×72 interfaces on top and bottom sides<br>■ No DQ pins on left and right sides |
| | 5SGSB5<br>5SGSB6 | 1,932 | ■ Two ×72 interfaces on top and bottom sides<br>■ No DQ pins on left and right sides |

Table 2–3 describes the maximum number of ×8 DDR2 SDRAM components that can be fitted in the smallest and biggest devices and pin packages assuming the device is blank.

Each interface of size *n*, where *n* is a multiple of 8, consists of:

- *n* DQ pins (including ECC)

- *n*/8 DM pins
- *n*/8 DQS, DQSn pin pairs
- 18 address pins
- 7 command pins (CAS, RAS, WE, CKE, ODT, reset, and CS)
- 1 CK, CK# pin pair up to every three ×8 DDR2 components

**Table 2–3. Maximum Number of DDR2 SDRAM Interfaces Supported per FPGA**

| Device | Device Type | Package Pin Count | Maximum Number of Interfaces |
|---|---|---|---|
| Arria II GX | EP2AGX190<br>EP2AGX260 | 1,152 | Four ×8 interfaces or one ×72 interface on each side (no DQ pins on left side) |
| | EP2AGX45<br>EP2AGX65 | 358 | ■ One ×16 interface on top and bottom sides<br>■ On right side (no DQ pins on left side), one ×8 interface |
| Arria II GZ | EP2AGZ300<br>EP2AGZ350 | F780 | ■ Three ×8 interfaces or one ×64 interface on top and bottom sides<br>■ No DQ pins on the left and right sides |
| | EP2AGZ300<br>EP2AGZ350<br>EP2AGZ225 | F1,517 | Four ×8 interfaces or one ×72 interface on each side |
| Cyclone III | EP3C120 | 780 | ■ Three ×16 interfaces on top and bottom sides<br>■ Two ×16 interfaces on left and right sides |
| | EP3C5 | 256 | ■ Two ×8 interfaces on top and bottom sides<br>■ One ×8 interface on right and left sides |
| Cyclone IV E | EP4CE115 | 780 | ■ One ×48 interface or two ×8 interfaces on top and bottom sides<br>■ Four ×8 interfaces on right and left sides |
| | EP4CE10 | 144 | On top side, one ×8 interface with address pins wrapped around the left or right side |
| Cyclone IV GX | EP4CGX150 | 896 | ■ One ×48 interface or four ×8 interfaces on top and bottom sides<br>■ On right side, three ×8 interfaces<br>■ No DQ pins on the left side |
| | EP4CGX22 | 324 | ■ One ×8 interface on top and bottom sides<br>■ On right side, one ×8 interface with address pins wrapped around the top or bottom side<br>■ No DQ pins on the left side |
| Stratix III | EP3SL340 | 1,760 | ■ Two ×72 interfaces on top and bottom sides<br>■ One ×72 interface on right and left sides |
| | EP3SE50 | 484 | ■ Two ×8 interfaces on top and bottom sides<br>■ Three ×8 interfaces on right and left sides |

**Table 2–3. Maximum Number of DDR2 SDRAM Interfaces Supported per FPGA**

| Device | Device Type | Package Pin Count | Maximum Number of Interfaces |
|---|---|---|---|
| Stratix IV | EP4SGX290<br>EP4SGX360<br>EP4SGX530 | 1,932 | ■ One ×72 interface on each side<br>or<br>■ One ×72 interface on each side and two additional ×72 wraparound interfaces only if sharing DLL and PLL resources |
| | EP4SE530<br>EP4SE820 | 1,760 | |
| | EP4SGX70<br>EP4SGX110<br>EP4SGX180<br>EP4SGX230 | 780 | ■ Three ×8 interfaces or one ×64 interface on top and bottom sides<br>■ On left side, one ×48 interface or two ×8 interfaces<br>■ No DQ pins on the right side |
| Stratix V | 5SGXA3<br>5SGXA4 | 780 | ■ On top side, two ×8 interfaces<br>■ On bottom side, four ×8 interfaces or one ×72 interface<br>■ No DQ pins on left and right sides |
| | 5SGXA5<br>5SGXA7 | 1,932 | ■ Three ×72 interfaces on top and bottom sides<br>■ No DQ pins on left and right sides |
| | 5SGSB5<br>5SGsB6 | 1,932 | ■ Two ×72 interfaces on top and bottom sides<br>■ No DQ pins on left and right sides |
| | 5SGSB7<br>5SGSB8 | 1,932 | ■ Two ×72 interfaces and a ×32 interface on top and bottom sides<br>■ No DQ pins on left and right sides |

Table 2–4 describes the maximum number of ×8 DDR3 SDRAM components that can be fitted in the smallest and biggest devices and pin packages assuming the device is blank.

Each interface of size *n*, where *n* is a multiple of 8, consists of:

- *n* DQ pins (including ECC)
- *n*/8 DM pins
- *n*/8 DQS,DQSn pin pairs
- 17 address pins
- 7 command pins (CAS, RAS, WE, CKE, ODT, reset, and CS)
- 1 CK, CK# pin pair

**Table 2–4. Maximum Number of DDR3 SDRAM Interfaces Supported per FPGA**

| Device | Device Type | Package Pin Count | Maximum Number of Interfaces |
|---|---|---|---|
| Arria II GX | EP2AGX190<br>EP2AGX260 | 1,152 | Four ×8 interfaces or one ×72 interface on each side (no DQ pins on left side) |
| | EP2AGX45<br>EP2AGX65 | 358 | ■ One ×16 interface on top and bottom sides<br>■ On right side, one ×8 interface (no DQ pins on left side) |
| Arria II GZ | EP2AGZ300<br>EP2AGZ350 | F780 | ■ Three ×8 interfaces on top and bottom sides<br>■ No DQ pins on the left and right sides |
| | EP2AGZ300<br>EP2AGZ350<br>EP2AGZ225 | F1,517 | Four ×8 interfaces on each side |
| Stratix III | EP3SL340 | 1,760 | ■ Two ×72 interfaces on top and bottom sides<br>■ One ×72 interface on right and left sides |
| | EP3SE50 | 484 | ■ Two ×8 interfaces on top and bottom sides<br>■ Three ×8 interfaces on right and left sides |
| Stratix IV | EP4SGX290<br>EP4SGX360<br>EP4SGX530 | 1,932 | ■ One ×72 interface on each side<br>or<br>■ One ×72 interface on each side and 2 additional ×72 wraparound interfaces only if sharing DLL and PLL resources |
| | EP4SE530<br>EP4SE820 | 1,760 | |
| | EP4SGX70<br>EP4SGX110<br>EP4SGX180<br>EP4SGX230 | 780 | ■ Three ×8 interfaces or one ×64 interface on top and bottom sides<br>■ On left side, one ×48 interface or two ×8 interfaces (no DQ pins on right side) |
| Stratix V | 5SGXA3<br>5SGXA4 | 780 | ■ On top side, two ×8 interfaces<br>■ On bottom side, four ×8 interfaces<br>■ No DQ pins on left and right sides |
| | 5SGXA5<br>5SGXA7 | 1,932 | ■ Three ×72 interfaces on top and bottom sides<br>■ No DQ pins on left and right sides |
| | 5SGSB5<br>5SGSB6 | 1,932 | ■ Two ×72 interfaces on top and bottom sides<br>■ No DQ pins on left and right sides |
| | 5SGSB7<br>5SGSB8 | 1,932 | ■ Two ×72 interfaces and a ×32 interface on top and bottom sides<br>■ Two ×72 interfaces and a ×32 interface on right side (no DQ pins on left side) |

Table 2–5 describes the maximum number of independent QDR II+ or QDR II SRAM interfaces that can be fitted in the smallest and biggest devices and pin packages assuming the device is blank.

One interface of ×36 consists of:

- 36 Q pins
- 36 D pins
- 1 K, K# pin pairs
- 1 CQ, CQ# pin pairs
- 19 address pins
- 4 BSWn pins
- WPS, RPS

One interface of ×9 consists of:

- 9 Q pins
- 9 D pins
- 1 K, K# pin pairs
- 1 CQ, CQ# pin pairs
- 21 address pins
- 1 BWSn pin
- WPS, RPS

**Table 2–5. Maximum Number of QDR II and QDR II+ SRAM Interfaces Supported per FPGA**

| Device | Device Type | Package Pin Count | Maximum Number of Interfaces |
|---|---|---|---|
| Arria II GX | EP2AGX190<br>EP2AGX260 | 1,152 | One ×36 interface and one ×9 interface one each side |
| | EP2AGX45<br>EP2AGX65 | 358 | One ×9 interface on each side (no DQ pins on left side) |
| Arria II GZ | EP2AGZ300<br>EP2AGZ350 | F780 | ■ Three ×9 interfaces on top and bottom sides<br>■ No DQ pins on right and left sides |
| | EP2AGZ300<br>EP2AGZ350<br>EP2AGZ225 | F1,517 | ■ Two ×36 interfaces and one ×9 interface on top and bottom sides<br>■ Four ×9 interfaces on right and left sides |
| Stratix III | EP3SL340 | 1,760 | ■ Two ×36 interfaces and one ×9 interface on top and bottom sides<br>■ On left side, five ×9 interfaces on right and left sides |
| | EP3SE50<br>EP3SL50<br>EP3SL70 | 484 | ■ One ×9 interface on top and bottom sides<br>■ Two ×9 interfaces on right and left sides |

**Table 2–5. Maximum Number of QDR II and QDR II+ SRAM Interfaces Supported per FPGA**

| Device | Device Type | Package Pin Count | Maximum Number of Interfaces |
|---|---|---|---|
| Stratix IV | EP4SGX290<br>EP4SGX360<br>EP4SGX530 | 1,932 | ■ Two ×36 interfaces on top and bottom sides<br>■ One ×36 interface on right and left sides |
| | EP4SE530<br>EP4SE820 | 1,760 | |
| | EP4SGX70<br>EP4SGX110<br>EP4SGX180<br>EP4SGX230 | 780 | Two ×9 interfaces on each side (no DQ pins on right side) |
| Stratix V | 5SGXA3<br>5SGXA4 | 780 | ■ On top side, one ×36 interface (with emulated write data group) or three ×9 interfaces<br>■ On bottom side, two ×9 interfaces<br>■ No DQ pins on left and right sides |
| | 5SGXA5<br>5SGXA7 | 1,932 | ■ Two ×36 interfaces on top and bottom sides<br>■ No DQ pins on left and right sides |
| | 5SGSB5<br>5SGSB6 | 1,932 | ■ Two ×36 interfaces on top and bottom sides<br>■ No DQ pins on left and right sides |

Table 2–6 describes the maximum number of independent RLDRAM II interfaces that can be fitted in the smallest and biggest devices and pin packages assuming the device is blank.

One common I/O ×36 interface consists of:

- 36 DQ
- 1 DM pin
- 2 DK, DK# pin pairs
- 2 QK, QK# pin pairs
- 1 CK, CK# pin pair
- 24 address pins
- 1 CS# pin
- 1 REF# pin
- 1 WE# pin
- 1 QVLD pin

One common I/O ×9 interface consists of:

- 9 DQ
- 1 DM pins
- 1 DK, DK# pin pair
- 1 QK, QK# pin pair

- 1 CK, CK# pin pair
- 25 address pins
- 1 CS# pin
- 1 REF# pin
- 1 WE# pin
- 1 QVLD pin

**Table 2–6. Maximum Number of RLDRAM II Interfaces Supported per FPGA**

| Device | Device Type | Package Pin Count | Maximum Number of RLDRAM II CIO Interfaces |
|---|---|---|---|
| Arria II GZ | EP2AGZ300<br>EP2AGZ350 | F780 | ■ Three ×9 interfaces or one ×36 interface on top and bottom sides<br>■ No DQ pins on the left and right sides |
| | EP2AGZ300<br>EP2AGZ350<br>EP2AGZ225 | F1,517 | ■ Two ×36 interfaces on each side |
| Stratix III | EP3SL340 | 1,760 | ■ Four ×36 components on top and bottom sides<br>■ Three ×36 interfaces on right and left sides |
| | EP3SE50<br>EP3SL50<br>EP3SL70 | 484 | One ×9 interface on right and left sides |
| Stratix IV | EP4SGX290<br>EP4SGX360<br>EP4SGX530 | 1,932 | ■ Three ×36 interfaces on top and bottom sides<br>■ Two ×36 interfaces on right and left sides |
| | EP4SE530<br>EP4SE820 | 1,760 | ■ Three ×36 interfaces on each side |
| | EP4SGX70<br>EP4SGX110<br>EP4SGX180<br>EP4SGX230 | 780 | One ×36 interface on each side (no DQ pins on right side) |
| Stratix V | 5SGXA3<br>5SGXA4 | 780 | ■ On top side, two ×9 interfaces or one ×18 interfaces<br>■ On bottom side, three ×9 interfaces or two ×36 interfaces<br>■ No DQ pins on left and right sides |
| | 5SGXA5<br>5SGXA7 | 1,932 | ■ Four ×36 interfaces on top and bottom sides<br>■ No DQ pins on left and right sides |
| | 5SGSB5<br>5SGSB6 | 1,932 | ■ Three ×36 interfaces on top and bottom sides<br>■ No DQ pins on left and right sides |

## OCT Support for Arria II GZ, Stratix III, Stratix IV, and Stratix V Devices

This section is not applicable to Cyclone III and Cyclone IV devices as OCT is not used by the Altera IP.

If you use OCT for your memory interfaces, refer to the *Device I/O Features* chapter in the *Cyclone III* or *Cyclone IV Device Handbook*.

If your system uses any FPGA OCT calibrated series, parallel, or dynamic termination for any I/O in your design, you need a calibration block for the OCT circuitry. This calibration block is not required to be within the same bank or side of the device as the I/O pins it is serving. However, the block requires a pair of $R_{UP}$ and $R_{DN}$ or $R_{ZQ}$ pins that must be placed within an I/O bank that has the same $V_{CCIO}$ voltage as the $V_{CCIO}$ voltage of the I/O pins that use the OCT calibration block.

The $R_{ZQ}$ pin in Stratix V devices is a dual functional pin that can also be used as DQ and DQS pins in Stratix V devices when it is not used to support OCT. You can use the DQS group in ×4 mode with non-differential DQS pins if the $R_{ZQ}$ pin is part of a ×4 DQS group.

The $R_{UP}$ and $R_{DN}$ pins in Arria II GZ, Stratix III, and Stratix IV devices are dual functional pins that can also be used as DQ and DQS pins in Arria II GZ, Stratix III, and Stratix IV devices when they are not used to support OCT, giving the following impacts on your DQS groups:

- If the $R_{UP}$ and $R_{DN}$ pins are part of a ×4 DQS group, you cannot use that DQS group in ×4 mode.
- If the $R_{UP}$ and $R_{DN}$ pins are part of a ×8 DQS group, you can only use this group in ×8 mode if either of the following conditions apply:
  - You are not using DM or BWSn pins.
  - You are not using a ×8 or ×9 QDR II and QDR II+ SRAM devices, as the $R_{UP}$ and $R_{DN}$ pins may have dual purpose function as the CQn pins. In this case, pick different pin locations for $R_{UP}$ and $R_{DN}$ pins, to avoid conflict with memory interface pin placement. You have the choice of placing the $R_{UP}$ and $R_{DN}$ pins in the same bank as the write data pin group or address and command pin group.

    The QDR II and QDR II+ SRAM controller with UniPHY do not support ×8 QDR II and QDR II+ SRAM devices in the Quartus II software version 10.0.

  - You are not using complementary or differential DQS pins.

A DQS/DQ ×8/×9 group in Arria II GZ, Stratix III, and Stratix IV devices comprises 12 pins. A typical ×8 memory interface consists of one DQS, one DM, and eight DQ pins which add up to 10 pins. If you choose your pin assignment carefully, you can use the two extra pins for $R_{UP}$ and $R_{DN}$. However, if you are using differential DQS, you do not have enough pins for $R_{UP}$ and $R_{DN}$ as you only have one pin leftover. In this case, as you do not have to put the OCT calibration block with the DQS or DQ pins, you can pick different locations for the $R_{UP}$ and $R_{DN}$ pins. As an example, you can place it in the I/O bank that contains the address and command pins, as this I/O bank has the same $V_{CCIO}$ voltage as the I/O bank containing the DQS and DQ pins.

There is no restriction when using ×16/×18 or ×32/×36 DQS groups that include the ×4 groups when pin members are used as $R_{UP}$ and $R_{DN}$ pins, as there are enough extra pins that can be used as DQS or DQ pins.

You need to pick your DQS and DQ pins manually for the ×8, ×9, ×16 and ×18, or ×32 and ×36 groups, if they are using $R_{UP}$ and $R_{DN}$ pins within the group. The Quartus II software may not place these pins optimally and may give you a no-fit.

## General Pinout Guidelines

Altera recommends that you place all the pins for one memory interface (attached to one controller) on the same side of the device. For projects where I/O availability is a challenge and therefore it is necessary spread the interface on two sides, for optimal performance, place all the input pins on one side, and the output pins on an adjacent side of the device along with their corresponding source-synchronous clock.

For a unidirectional data bus as in QDR II and QDR II+ SRAM interfaces, do not split a read data pin group or a write data pin group onto two sides. It is also strongly recommended not to split the address and command group onto two sides either, especially when you are interfacing with QDR II and QDR II+ SRAM burst-length-of-two devices, where the address signals are double data rate also. Failure to adhere to these rules may result in timing failure.

In addition, there are some exceptions for the following interfaces:

- ×36 emulated QDR II and QDR II+ SRAM in Arria II, Stratix III, Stratix IV, and Stratix V devices.
- RLDRAM II CIO devices—RLDRAM II SIO pinout guidelines are covered in the general pinout guidelines.
- QDR II/+ SDRAM burst-length-of-two devices.

CAUTION

You need to compile the design in Quartus II to ensure that you are not violating signal integrity and Quartus II placement rules, which is critical when you have transceivers in the same design.

The following list gives some general guidelines on how to place pins optimally for your memory interfaces:

1. For Arria II GZ, Stratix III, Stratix IV, and Stratix V designs, if you are using OCT, the $R_{UP}$ and $R_{DN}$, or $R_{ZQ}$ pins need to be in any bank with the same I/O voltage as your memory interface signals and often use two DQS and DQ pins from a group. If you decide to place the $R_{UP}$ and $R_{DN}$, or $R_{ZQ}$ pins in a bank where the DQS and DQ groups are used, place these pins first and then see how many DQ pins you have left after, to find out if your data pins can fit in the remaining pins. Refer to "OCT Support for Arria II GZ, Stratix III, Stratix IV, and Stratix V Devices" on page 2–13.
2. Use the PLL that is on the same side of the memory interface. If the interface is spread out on two adjacent sides, you may use the PLL that is located on either adjacent side. You must use the dedicated input clock pin to that particular PLL as the reference clock for the PLL as the input of the memory interface PLL cannot come from the FPGA clock network.

3. The Altera IP uses the output of the memory interface PLL for the DLL input reference clock. Therefore, ensure you pick a PLL that can directly feed a suitable DLL.

   Alternatively, you can use an external pin to feed into the DLL input reference clock. The available pins are also listed in the *External Memory Interfaces* chapter of the relevant device family handbook. You can also activate an unused PLL clock outputs , set it at the desired DLL frequency, and route it to a PLL dedicated output pin. Connect a trace on the PCB from this output pin to the DLL reference clock pin, but be sure to include any signal integrity requirements such as terminations.

4. Read data pins require the usage of DQS and DQ group pins to have access to the DLL control signals.

   In addition, QVLD pins in RLDRAM II and QDR II+ SRAM must use DQS group pins, when the design uses the QVLD signal. None of the Altera IP uses QVLD pins as part of read capture, so theoretically you do not need to connect the QVLD pins if you are using the Altera solution. It is good to connect it anyway in case the Altera solution gets updated to use QVLD pins.

5. The read data strobe or read data clock must connect to a DQS pin. In differential clocking (DDR3/DDR2 SDRAM and RLDRAM II interfaces), connect the negative leg of the read data strobe or clock to a DQSn pin. In complementary clocking (QDR II and QDR II+ SRAM interfaces), connect the complement clock (CQn) pin to the CQn pin (and not the DQSn pin) from the pin table for controllers with 2.5 cycles read latency, and connect the CQn pin to DQS pin for controllers with 2.0 cycles read latency.
6. Write data (if unidirectional) and data mask pins (DM or BWSn) pins must use DQS groups. While the DLL phase shift is not used, using DQS groups for write data minimizes skew, and must use the SW and TCCS timing analysis methodology.
7. Assign the write data strobe or write data clock (if unidirectional) in the corresponding DQS/DQSn pin with the write data groups that place in DQ pins (except in RLDRAM II CIO devices, refer to "Pinout Rule Exceptions" on page 2–16)

   When interfacing with a DDR, or DDR2, or DDR3 SDRAM without leveling, put the three CK and CK# pairs in a single ×4 DQS group to minimize skew between clocks and maximize margin for the $t_{DQSS}$, $t_{DSS}$, and $t_{DSH}$ specifications from the memory devices.

8. Assign any address pins to any user I/O pin. To minimize skew within the address pin group, you should assign the address and command pins in the same bank or side of the device.
9. Assign the command pins to any I/O pins and assign the pins in the same bank or device side as the other memory interface pins, especially address and memory clock pins. The memory device usually uses the same clock to register address and command signals.

In QDR II and QDR II+ SRAM interfaces where the memory clock also registers the write data, assign the address and command pins in the same I/O bank or same side as the write data pins, to minimize skew.

Refer to "Pin Connection Guidelines Tables" on page 2–23 for more information about assigning memory clock pins for different device families and memory standards.

# Pinout Rule Exceptions

The following sub sections described exceptions to the rule described in the "General Pinout Guidelines" on page 2–14.

## Exceptions for ×36 Emulated QDR II and QDR II+ SRAM Interfaces in Arria II, Stratix III, Stratix IV, and Stratix V Devices

A few packages in the Arria II, Stratix III, Stratix IV, and Stratix V device families do not offer any ×32/×36 DQS groups where one read clock or strobe is associated with 32 or 36 read data pins. This limitation exists in the following I/O banks:

- All I/O banks in U358- and F572-pin packages for all Arria II GX devices
- All I/O banks in F484-pin packages for all Stratix III devices
- All I/O banks in F780-pin packages for all Arria II GZ, Stratix III, and Stratix IV devices; top and side I/O banks in F780-pin packages for all Stratix V devices
- All I/O banks in F1152-pin packages for all Arria II GZ, Stratix III, and Stratix IV devices, except EP4SGX290, EP4SGX360, EP4SGX530, EPAGZ300, and EPAGZ350 devices
- Side I/O banks in F1517- and F1760-pin packages for all Stratix III devices
- All I/O banks in F1517-pin for EP4SGX180, EP4SGX230, EP4S40G2, EP4S40G5, EP4S100G2, EP4S100G5, and EPAGZ225 devices
- Side I/O banks in F1517-, F1760-, and F1932-pin packages for all Arria II GZ and Stratix IV devices
- Side I/O banks for all Stratix V devices

This limitation limits support for ×36 QDR II and QDR II+ SRAM devices. To support these memory devices, this following section describes how you can emulate the ×32/×36 DQS groups for these devices.

CAUTION

The maximum frequency supported in ×36 QDR II and QDR II+ SRAM interfaces using ×36 emulation is lower than the maximum frequency when using a native ×36 DQS group.

The F484-pin package in Stratix III devices cannot support ×32/×36 DQS group emulation, as it does not support ×16/×18 DQS groups.

To emulate a ×32/×36 DQS group, combine two ×16/×18 DQS groups together. For ×36 QDR II and QDR II+ SRAM interfaces, the 36-bit wide read data bus uses two ×16/×18 groups; the 36-bit wide write data uses another two ×16/×18 groups or four ×8/×9 groups. The CQ and CQn signals from the QDR II and QDR II+ SRAM device traces are then split on the board to connect to two pairs of CQ/CQn pins in the FPGA. You may then need to split the QVLD pins also (if you are connecting them). These connections are the only connections on the board that you need to change for this implementation. There is still only one pair of K and Kn# connections on the board from the FPGA to the memory (see Figure 2–1). Use an external termination for the CQ/CQn signals at the FPGA end. You can use the FPGA OCT features on the other QDR II interface signals with ×36 emulation. In addition, there may be extra assignments to be added with ×36 emulation.

☞ Other QDR II and QDR II+ SRAM interface rules also apply for this implementation.

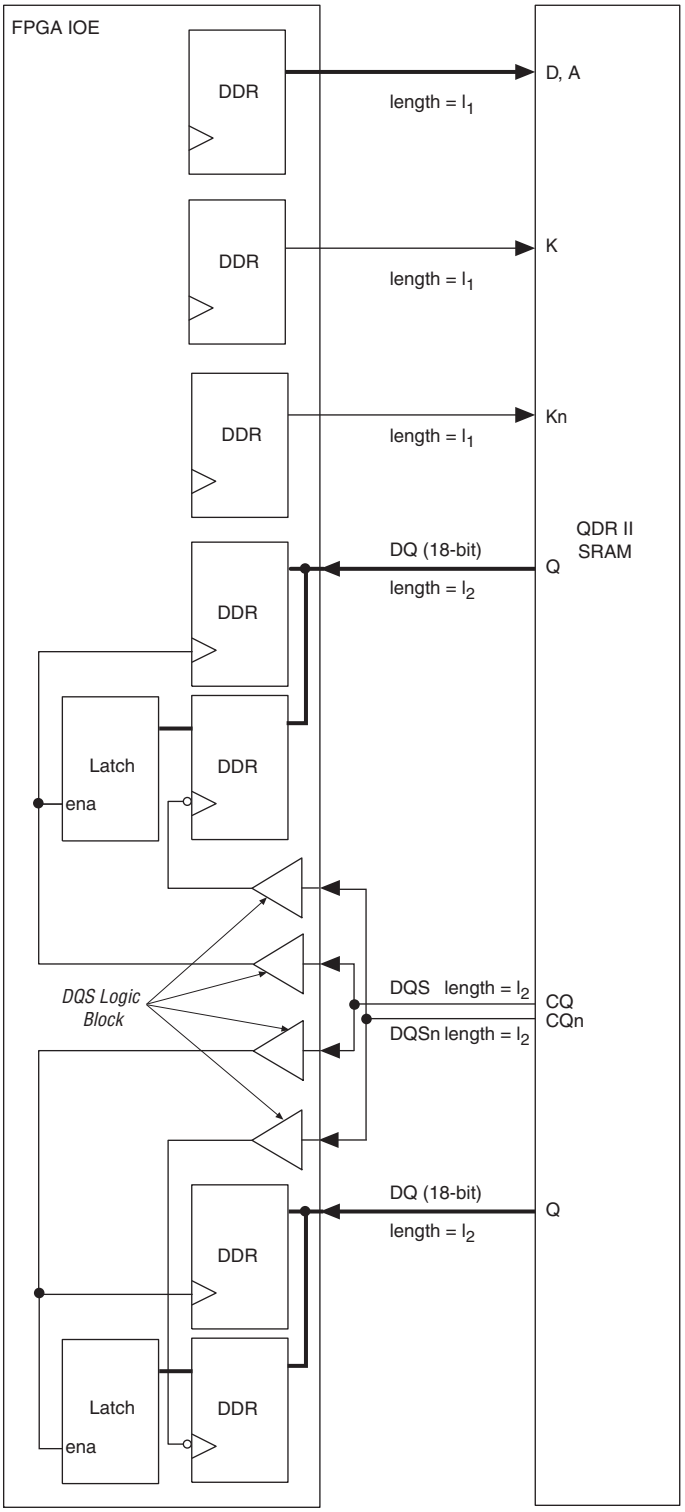
You may also combine four ×9 DQS groups (or two ×9 DQS groups and one ×18 group) on the same side of the device, if not the same I/O bank, to emulate a x36 write data group, if you need to fit the QDR II interface in a particular side of the device that does not have enough ×18 DQS groups available for write data pins. Altera does not recommend using ×4 groups as the skew may be too large, as you need eight ×4 groups to emulate the ×36 write data bits.

You cannot combine four ×9 groups to create a ×36 read data group as the loading on the CQ pin is too large and hence the signal is degraded too much.

When splitting the CQ and CQn signals, the two trace lengths that go to the FPGA pins must be as short as possible to reduce reflection. These traces must also have the same trace delay from the FPGA pin to the Y or T junction on the board. The total trace delay from the memory device to each pin on the FPGA should match the Q trace delay ($I_2$).

☞ You must match the trace delays. However, matching trace length is only an approximation to matching actual delay.

**Figure 2–1. Board Trace Connection for Emulated x36 QDR II and QDR II+ SRAM Interface**



## Timing Impact on x36 Emulation

With ×36 emulation, the CQ/CQn signals are split on the board, so these signals see two loads (to the two FPGA pins)—the DQ signals still only have one load. The difference in loading gives some slew rate degradation, and a later CQ/CQn arrival time at the FPGA pin.

The slew rate degradation factor is taken into account during timing analysis when you indicate in the UniPHY Preset Editor that you are using ×36 emulation mode. However, you must determine the difference in CQ/CQn arrival time as it is highly dependent on your board topology.

The slew rate degradation factor for ×36 emulation assumes that CQ/CQn has a slower slew rate than a regular ×36 interface. The slew rate degradation is assumed not to be more than 500 ps (from 10% to 90% $V_{CCIO}$ swing). You may also modify your board termination resistor to improve the slew rate of the ×36-emulated CQ/CQn signals. If your modified board does not have any slew rate degradation, you do not need to enable the ×36 emulation timing in the UniPHY-based controller MegaWizard interface.

For more information on how to determine the CQ/CQn arrival time skew, see "Determining the CQ/CQn Arrival Time Skew" on page 2–20. .

Because of this effect, the maximum frequency supported using x36 emulation is lower than the maximum frequency supported using a native x36 DQS group.

## Rules to Combine Groups

For information about group combining in Arria II GX devices, refer to the *External Memory Interface* chapter in the *Arria II Device Handbook*.

For devices that do not have four ×16/×18 groups in a single side of the device to form two ×36 groups for read and write data, you can form one ×36 group on one side of the device, and another ×36 group on the other side of the device. All the read groups have to be on the same edge (column I/O or row I/O) and all write groups have to be on the same type of edge (column I/O or row I/O), so you can have an interface with the read group in column I/O and the write group in row I/O. The only restriction is that you cannot combine an ×18 group from column I/O with an ×18 group from row IO to form a x36-emulated group.

For vertical migration with the ×36 emulation implementation, check if migration is possible and enable device migration in the Quartus II software.

I/O bank 1C in both Stratix III and Stratix IV devices has dual-function configuration pins. Some of the DQS pins may not be available for memory interfaces if these are used for device configuration purposes.

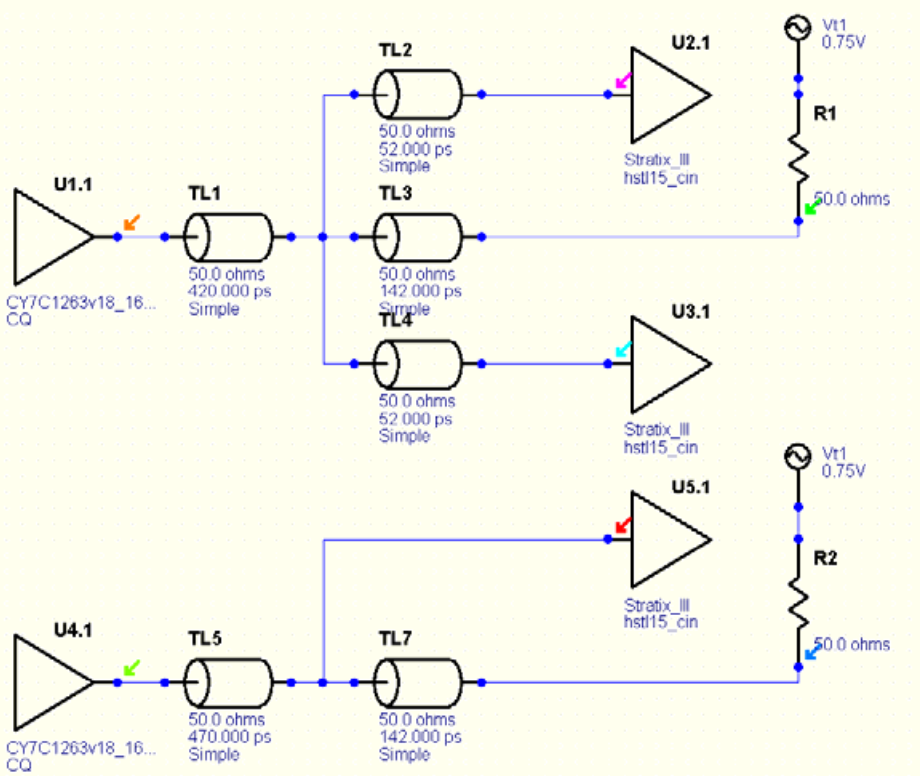
Each side of the device in these packages has four remaining ×8/×9 groups You can combine four of the remaining for the write side (only) if you want to keep the ×36 QDR II and QDR II+ SRAM interface on one side of the device, by changing the **Memory Interface Data Group** default assignment, from the default **18** to **9**.

The ALTMEMPHY megafunction does not support ×36 mode emulation wraparound interface, where the ×36 group consists of a ×18 group from the top/bottom I/O bank and a ×18 group from the side I/O banks.

## Determining the CQ/CQn Arrival Time Skew

Before compiling a design in Quartus II, you need to determine the CQ/CQn arrival time skew based on your board simulation. You then need to apply this skew in the **report_timing.tcl** file of your QDR II and QDR II+ SRAM interface in the Quartus II software. Figure 2–2 shows an example of a board topology comparing an emulated case where CQ is double-loaded and a non-emulated case where CQ only has a single load.

**Figure 2–2. Board Simulation Topology Example**

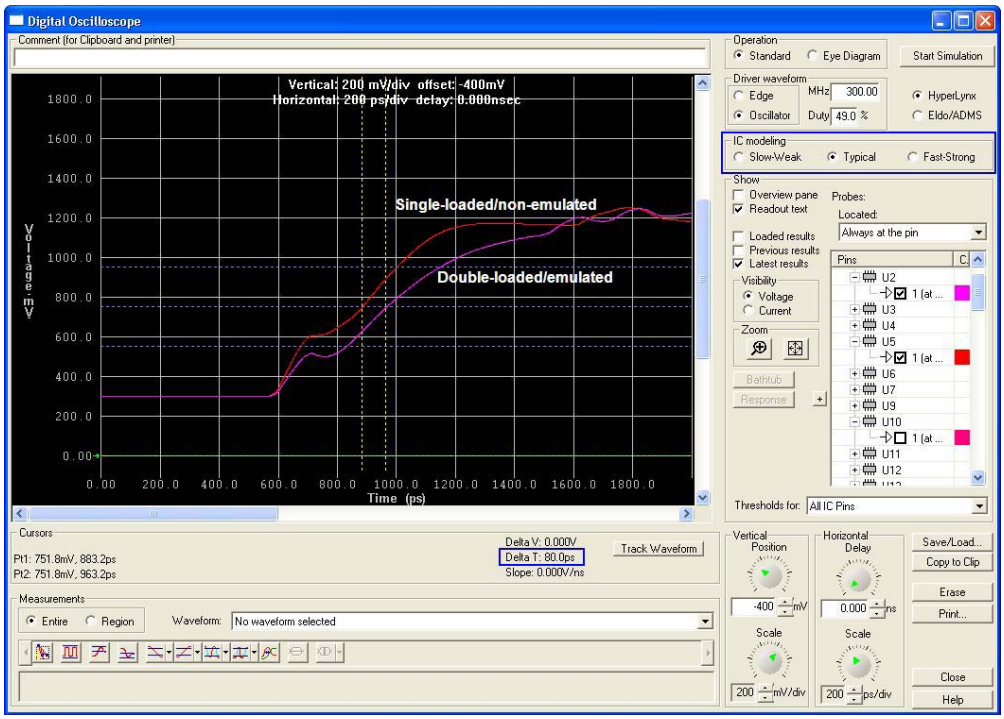



Run the simulation and look at the signal at the FPGA pin. Figure 2–3 shows an example of the simulation results from Figure 2–2. As expected, the double-loaded emulated signal, in pink, arrives at the FPGA pin later than the single-loaded signal, in red. You then need to calculate the difference of this arrival time at $V_{REF}$ level (0.75 V in this case). Record the skew and re-run the simulation in the other two cases (slow-weak and fast-strong). To pick the largest and smallest skew to be included in Quartus II timing analysis, follow these steps:

1. Open the <*variation_name*>**_report_timing.tcl** and search for `tmin_additional_dqs_variation`.
2. Set the minimum skew value from your board simulation to `tmin_additional_dqs_variation`.

3. Set the maximum skew value from your board simulation to `tmax_additional_dqs_variation`.
4. Save the **.tcl** file.

**Figure 2–3. Board Simulation Results**




## Exceptions for RLDRAM II Interfaces

RLDRAM II CIO devices have one bidirectional bus for the data, but there are two different sets of clocks: one for read and one for write. As the QK and QK# already occupies the DQS and DQSn pins needed for read, placement of DK and DK# pins are restricted due to the limited number of pins in the FPGA. This limitations causes the exceptions to the previous rules, which are discussed in the following sections.

☞ DK and DK# signals need to use DQS- and DQSn-capable pins to ensure accurate timing analysis, as the TCCS specifications are characterized using DQS and DQSn pins. As you must use the DQS and DQSn pins for the DQS group to connect to QK and QK# pins, pick a pair of DQ pins that are DQS and DQSn pins when configured as a smaller DQS group size. For example, if the interfaces uses a ×16/×18 DQS group, the DQS and DQSn pins connect to QK and QK# pins, pick differential DQ pin pairs from that DQS group that are DQS and DQSn pins for ×8/×9 DQS groups or ×4 DQS groups.

### Interfacing with ×9 RLDRAM II CIO Devices

These devices have the following pins:

- 2 pins for QK and QK# signals
- 9 DQ pins (in a ×8/×9 DQS group)

- 2 pins for DK and DK# signals
- 1 DM pin
- 1 QVLD pins
- 15 pins total

In the FPGA, the ×8/×9 DQS group consists of 12 pins: 2 for the read clocks and 10 for the data. In this case, move the QVLD (if you want to keep this connected even though this is not used in the Altera memory interface solution) and the DK and DK# pins to the adjacent DQS group. If that group is in use, move to any available user I/O pins in the same I/O bank. The DK and DK# must use DQS- and DQSn-capable pins.

### Interfacing with ×18 RLDRAM II CIO Devices

These devices have the following pins:

- 4 pins for QK/QK# signals
- 18 DQ pins (in ×8/×9 DQS group)
- 2 pins for DK/DK# signals
- 1 DM pin
- 1 QVLD pins
- 26 pins total

In the FPGA, you use two ×8/×9 DQS group totaling 24 pins: 4 for the read clocks and 20 for the data. In this case, move the DK and DK# pins to DQS- and DQSn-capable pins in the adjacent DQS group. Or if that group is in use, move to any DQS- and DQSn-capable pins in the same I/O bank.

Each ×8/×9 group has one DQ pin left over that can either use QVLD or DM, so one ×8/×9 group has the DM pin associated with that group and one ×8/×9 group has the QVLD pin associated with that group.

### Interfacing with RLDRAM II ×36 CIO Devices

These devices have the following pins:

- 4 pins for QK/QK# signals
- 36 DQ pins (in x16/x18 DQS group)
- 4 pins for DK/DK# signals
- 1 DM pins
- 1 QVLD pins
- 46 pins total

In the FPGA, you use two ×16/×18 DQS groups totaling 48 pins: 4 for the read clocks and 36 for the read data. Configure each ×16/×18 DQS group to have:

- Two QK/QK# pins occupying the DQS/DQSn pins
- Pick two DQ pins in the ×16/×18 DQS groups that are DQS and DQSn pins in the ×4 or ×8/×9 DQS groups for the DK and DK# pins
- 18 DQ pins occupying the DQ pins

- There are two DQ pins leftover that you can use for QVLD or DM pins. Put the DM pin in the group associated with `DK[1]` and the QVLD pin in the group associated with `DK[0]`.

  Check that DM is associated with `DK[1]` for your chosen memory component.

### Exceptions for QDR II and QDR II+ SRAM Burst-length-of-two Interfaces

If you are using the QDR II SRAM controller with UniPHY or creating your own interface for QDR II and QDR II+ SRAM burst-length-of-two devices, you may want to place the address pins in a DQS group also, because these pins are now double data rate too. The address pins typically do not exceed 22 bits, so you may use one ×18 DQS groups or two ×9 DQS groups on the same side of the device, if not the same I/O bank. In Stratix III, Stratix IV, and Stratix V devices, one ×18 group typically has 22 DQ bits and 2 pins for DQS/DQSn pins, while one ×9 group typically has 10 DQ bits with 2 pins for DQS/DQSn pins. Using ×4 DQS groups should be a last resort.

## Pin Connection Guidelines Tables

Table 2–7 shows the FPGA pin utilization for DDR, DDR2, and DDR3 SDRAM without leveling interfaces.

**Table 2–7. DDR, DDR2, and DDR3 SDRAM Without Leveling Interface Pin Utilization (Part 1 of 3)**

| Interface Pin Description | Memory Device Pin Name | FPGA Pin Utilization | | | |
|---|---|---|---|---|---|
| | | Arria II GX Devices | Cyclone III and Cyclone IV Devices | Arria II GZ, Stratix III, and Stratix IV Devices | Stratix V Devices |
| Data | DQ | DQ in the pin table, marked as Q in the Quartus II Pin Planner. Each DQ group has a common background color for all of the DQ and DM pins, associated with DQS (and DQSn) pins. | | | |
| Data mask | DM | | | | |
| Data strobe | DQS or DQS and DQSn (DDR2 and DDR3 SDRAM only) | DQS (S in the Quartus II Pin Planner) for single-ended DQS signaling or DQS and DQSn (S and Sbar in the Quartus II Pin Planner) for differential DQS signaling. DDR2 supports either single-ended or differential DQS signaling. However, Cyclone III and Cyclone IV devices do not support differential DQS signaling. DDR3 SDRAM mandates differential DQS signaling. | | | |
| Address and command | A[], BA[], CAS#, CKE, CS#, ODT, RAS#, WE#, RESET# | Any user I/O pin. To minimize skew, you should place address and command pins in the same bank or side of the device as the following pins: CK/CK# pins, DQ, DQS, or DM pins. The `reset#` signal is only available in DDR3 SDRAM interfaces. Altera devices use the SSTL-15 I/O standard on the `RESET#` signal to meet the voltage requirements of 1.5 V CMOS at the memory device. Altera recommends that you do not terminate the `RESET#` signal to $V_{TT}$. | | | |

**Table 2–7. DDR, DDR2, and DDR3 SDRAM Without Leveling Interface Pin Utilization (Part 2 of 3)**

| Interface Pin Description | Memory Device Pin Name | FPGA Pin Utilization | | | |
|---|---|---|---|---|---|
| | | Arria II GX Devices | Cyclone III and Cyclone IV Devices | Arria II GZ, Stratix III, and Stratix IV Devices | Stratix V Devices |
| Memory system clock | CK and CK# | If you are using single-ended DQS signaling, any unused DQ or DQS pins with DIFFOUT capability located in the same bank or on the same side as the data pins.<br>If you are using differential DQS signaling, the first CK/CK# pairs (namely mem_clk[0] or mem_clk_n[0] in the IP) must use any unused DQ or DQS pins with DIFFIO_RX or DIFFIN capability in the same bank or on the same side as the data pins. You can use either side of the device for wraparound interfaces.<br>This placement allows the mimic path that the IP VT tracking uses to go through differential I/O buffers to mimic the differential DQS signals. If there are other CK/CK# pairs (mem_clk[n:1] and mem_clk_n[n:1] where *n* <= 1), place on DIFF_OUT in the same single DQ group of adequate width to minimize skew. | Any differential I/O pin pair (DIFFIO) in the same bank or on the same side as the data pins. You can use either side of the device for wraparound interfaces.<br>mem_clk[0] and mem_clk_n[0] cannot be placed in the same row or column pad group as any of the DQ pins (Figure 2–4 and Figure 2–5). | If you are using single-ended DQS signaling, any DIFFOUT pins in the same bank or on the same side as the data pins.<br>If you are using differential DQS signaling, the first CK/CK# pairs (namely mem_clk[0] or mem_clk_n[0] in the IP) must use any unused DIFFIO_RX pins in the same bank or on the same side as the data pins. You can use either side of the device for wraparound interfaces.<br>This placement allows the mimic path that the IP VT tracking uses to go through differential I/O buffers to mimic the differential DQS signals. If there are other CK/CK# pairs (mem_clk[n:1] and mem_clk_n[n:1] where *n* <= 1), place on DIFF_OUT in the same single DQ group of adequate width to minimize skew. | If you are using single-ended DQS signaling, any DIFFOUT pins in the same bank or on the same side as the data pins.<br>If you are using differential DQS signaling, any unused pins with DIFFOUT, DIFFIO_TX,or DIFFIO_RX capability for the mem_clk[n:0] and mem_clk_n[n:0] signals (where *n* is greater than or equal to zero). |

**Table 2–7. DDR, DDR2, and DDR3 SDRAM Without Leveling Interface Pin Utilization (Part 2 of 3)**

| Interface Pin Description | Memory Device Pin Name | FPGA Pin Utilization | | | |
|---|---|---|---|---|---|
| | | Arria II GX Devices | Cyclone III and Cyclone IV Devices | Arria II GZ, Stratix III, and Stratix IV Devices | Stratix V Devices |
| Memory system clock | CK and CK# | If you are using single-ended DQS signaling, any unused DQ or DQS pins with DIFFOUT capability located in the same bank or on the same side as the data pins.<br>If you are using differential DQS signaling, the first CK/CK# pairs (namely `mem_clk[0]` or `mem_clk_n[0]` in the IP) must use any unused DQ or DQS pins with `DIFFIO_RX` or `DIFFIN` capability in the same bank or on the same side as the data pins. You can use either side of the device for wraparound interfaces.<br>This placement allows the mimic path that the IP VT tracking uses to go through differential I/O buffers to mimic the differential DQS signals. If there are other CK/CK# pairs (`mem_clk[n:1]` and `mem_clk_n[n:1]` where *n* <= 1), place on `DIFF_OUT` in the same single DQ group of adequate width to minimize skew. | Any differential I/O pin pair (`DIFFIO`) in the same bank or on the same side as the data pins. You can use either side of the device for wraparound interfaces.<br>`mem_clk[0]` and `mem_clk_n[0]` cannot be placed in the same row or column pad group as any of the DQ pins (Figure 2–4 and Figure 2–5). | If you are using single-ended DQS signaling, any `DIFFOUT` pins in the same bank or on the same side as the data pins.<br>If you are using differential DQS signaling, the first CK/CK# pairs (namely `mem_clk[0]` or `mem_clk_n[0]` in the IP) must use any unused `DIFFIO_RX` pins in the same bank or on the same side as the data pins. You can use either side of the device for wraparound interfaces.<br>This placement allows the mimic path that the IP VT tracking uses to go through differential I/O buffers to mimic the differential DQS signals. If there are other CK/CK# pairs (`mem_clk[n:1]` and `mem_clk_n[n:1]` where *n* <= 1), place on `DIFF_OUT` in the same single DQ group of adequate width to minimize skew. | If you are using single-ended DQS signaling, any `DIFFOUT` pins in the same bank or on the same side as the data pins.<br>If you are using differential DQS signaling, any unused pins with `DIFFOUT`, `DIFFIO_TX`,or `DIFFIO_RX` capability for the `mem_clk[n:0]` and `mem_clk_n[n:0]` signals (where *n* is greater than or equal to zero). |

**Table 2–7. DDR, DDR2, and DDR3 SDRAM Without Leveling Interface Pin Utilization (Part 2 of 3)**

| Interface Pin Description | Memory Device Pin Name | FPGA Pin Utilization | | | |
|---|---|---|---|---|---|
| | | Arria II GX Devices | Cyclone III and Cyclone IV Devices | Arria II GZ, Stratix III, and Stratix IV Devices | Stratix V Devices |
| Memory system clock | CK and CK# | If you are using single-ended DQS signaling, any unused DQ or DQS pins with DIFFOUT capability located in the same bank or on the same side as the data pins.<br>If you are using differential DQS signaling, the first CK/CK# pairs (namely `mem_clk[0]` or `mem_clk_n[0]` in the IP) must use any unused DQ or DQS pins with `DIFFIO_RX` or `DIFFIN` capability in the same bank or on the same side as the data pins. You can use either side of the device for wraparound interfaces.<br>This placement allows the mimic path that the IP VT tracking uses to go through differential I/O buffers to mimic the differential DQS signals. If there are other CK/CK# pairs (`mem_clk[n:1]` and `mem_clk_n[n:1]` where *n* <= 1), place on `DIFF_OUT` in the same single DQ group of adequate width to minimize skew. | Any differential I/O pin pair (`DIFFIO`) in the same bank or on the same side as the data pins. You can use either side of the device for wraparound interfaces.<br>`mem_clk[0]` and `mem_clk_n[0]` cannot be placed in the same row or column pad group as any of the DQ pins (Figure 2–4 and Figure 2–5). | If you are using single-ended DQS signaling, any `DIFFOUT` pins in the same bank or on the same side as the data pins.<br>If you are using differential DQS signaling, the first CK/CK# pairs (namely `mem_clk[0]` or `mem_clk_n[0]` in the IP) must use any unused `DIFFIO_RX` pins in the same bank or on the same side as the data pins. You can use either side of the device for wraparound interfaces.<br>This placement allows the mimic path that the IP VT tracking uses to go through differential I/O buffers to mimic the differential DQS signals. If there are other CK/CK# pairs (`mem_clk[n:1]` and `mem_clk_n[n:1]` where *n* <= 1), place on `DIFF_OUT` in the same single DQ group of adequate width to minimize skew. | If you are using single-ended DQS signaling, any `DIFFOUT` pins in the same bank or on the same side as the data pins.<br>If you are using differential DQS signaling, any unused pins with `DIFFOUT`, `DIFFIO_TX`,or `DIFFIO_RX` capability for the `mem_clk[n:0]` and `mem_clk_n[n:0]` signals (where *n* is greater than or equal to zero). |

**Table 2–7. DDR, DDR2, and DDR3 SDRAM Without Leveling Interface Pin Utilization (Part 2 of 3)**

| Interface Pin Description | Memory Device Pin Name | FPGA Pin Utilization | | | |
|---|---|---|---|---|---|
| | | Arria II GX Devices | Cyclone III and Cyclone IV Devices | Arria II GZ, Stratix III, and Stratix IV Devices | Stratix V Devices |
| Memory system clock | CK and CK# | If you are using single-ended DQS signaling, any unused DQ or DQS pins with DIFFOUT capability located in the same bank or on the same side as the data pins.<br>If you are using differential DQS signaling, the first CK/CK# pairs (namely `mem_clk[0]` or `mem_clk_n[0]` in the IP) must use any unused DQ or DQS pins with `DIFFIO_RX` or `DIFFIN` capability in the same bank or on the same side as the data pins. You can use either side of the device for wraparound interfaces.<br>This placement allows the mimic path that the IP VT tracking uses to go through differential I/O buffers to mimic the differential DQS signals. If there are other CK/CK# pairs (`mem_clk[n:1]` and `mem_clk_n[n:1]` where *n* <= 1), place on `DIFF_OUT` in the same single DQ group of adequate width to minimize skew. | Any differential I/O pin pair (`DIFFIO`) in the same bank or on the same side as the data pins. You can use either side of the device for wraparound interfaces.<br>`mem_clk[0]` and `mem_clk_n[0]` cannot be placed in the same row or column pad group as any of the DQ pins (Figure 2–4 and Figure 2–5). | If you are using single-ended DQS signaling, any `DIFFOUT` pins in the same bank or on the same side as the data pins.<br>If you are using differential DQS signaling, the first CK/CK# pairs (namely `mem_clk[0]` or `mem_clk_n[0]` in the IP) must use any unused `DIFFIO_RX` pins in the same bank or on the same side as the data pins. You can use either side of the device for wraparound interfaces.<br>This placement allows the mimic path that the IP VT tracking uses to go through differential I/O buffers to mimic the differential DQS signals. If there are other CK/CK# pairs (`mem_clk[n:1]` and `mem_clk_n[n:1]` where *n* <= 1), place on `DIFF_OUT` in the same single DQ group of adequate width to minimize skew. | If you are using single-ended DQS signaling, any `DIFFOUT` pins in the same bank or on the same side as the data pins.<br>If you are using differential DQS signaling, any unused pins with `DIFFOUT`, `DIFFIO_TX`,or `DIFFIO_RX` capability for the `mem_clk[n:0]` and `mem_clk_n[n:0]` signals (where *n* is greater than or equal to zero). |

**Table 2–7. DDR, DDR2, and DDR3 SDRAM Without Leveling Interface Pin Utilization (Part 3 of 3)**

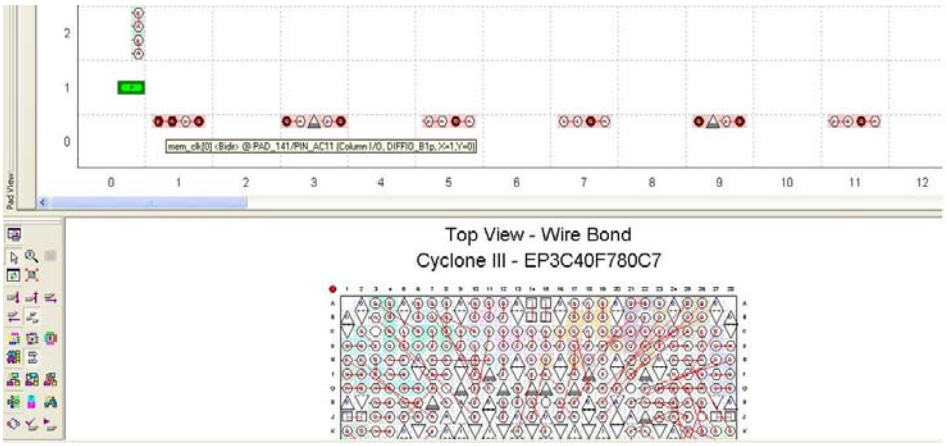
| Interface Pin Description | Memory Device Pin Name | FPGA Pin Utilization | | | |
|---|---|---|---|---|---|
| | | Arria II GX Devices | Cyclone III and Cyclone IV Devices | Arria II GZ, Stratix III, and Stratix IV Devices | Stratix V Devices |
| Memory system clock | CK and CK# | For example, DIMMs requiring three memory clock pin-pairs need to use a ×4 DQS group, where `mem_clk[0]` and `mem_clk_n[0]` pins use the `DIFFIO_RX` or `DIFFIN` pins in that group, while, `mem_clk[2:1]` and `mem_clk_n[2:1]` pins use `DIFF_OUT` pins in that DQS group. | | For example, DIMMs requiring three memory clock pin-pairs need to use a ×4 DQS group, where `mem_clk[0]` and `mem_clk_n[0]` pins use the `DIFFIO_RX` or `DIFFIN` pins in that group, while, `mem_clk[2:1]` and `mem_clk_n[2:1]` pins use `DIFF_OUT` pins in that DQS group. | |
| Clock Source | — | Dedicated PLL clock input pin with direct (not using a global clock net) connection to the PLL. Ensure that the PLL can supply the input reference clock to the DLL also, otherwise refer to alternative DLL input reference clocks ("General Pinout Guidelines" on page 2–14). | Dedicated PLL clock input pin with direct (not using a global clock net) connection to the PLL | Dedicated PLL clock input pin with direct (not using a global clock net) connection to the PLL. Ensure that the PLL can supply the input reference clock to the DLL also, otherwise refer to alternative DLL input reference clocks ("General Pinout Guidelines" on page 2–14). | Dedicated PLL clock input pin with direct (not using a global clock net) connection to the PLL. Ensure that the PLL can supply the input reference clock to the DLL also, otherwise refer to alternative DLL input reference clocks ("General Pinout Guidelines" on page 2–14). |
| Reset | — | Dedicated clock input pin to accommodate the high fan-out signal. | | | |

Figure 2–4 shows an example of assigning `mem_clk[0]` and `mem_clk_n[0]` incorrectly. As you can see, `mem_clk[0]` pin is assigned at the same column pad group as `mem_dq` pin (in column X = 1). This assignment results in the Quartus II software showing the following critical warning:

```
Register <name> fed by pin mem_clk[0] must be placed in adjacent LAB X:1
Y:0 instead of X:2 Y:0
```

Assigning the `mem_clk[0]` pin on the same row or column pad group as the DQ pin pins will also result in the failure to constrain the DDIO input nodes correctly and close timing. Hence, the Read Capture and Write timing margins computed by Time Quest may not be valid due to the violation of assumptions made by the timing scripts.

**Figure 2–4. Incorrect Placement of mem_clk[0] and mem_clk_n[0] in Cyclone III and Cyclone IV Devices.**




To eliminate this critical warning, assign the `mem_clk[0]` pin at different column or row from the data pin (Figure 2–5).

**Figure 2–5. Correct Placement of mem_clk[0] and mem_clk_n[0] in Cyclone III and Cyclone IV Devices.**

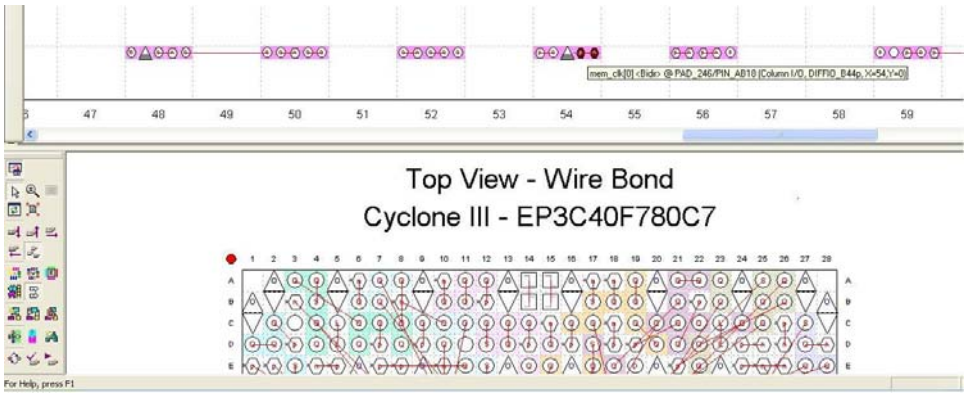


Table 2–8 shows the FPGA pin utilization for DDR3 SDRAM with leveling interfaces.

**Table 2–8. DDR3 SDRAM With Leveling Interface Pin Utilization Applicable for Stratix III, Stratix IV, and Stratix V Devices**

| Interface Pin Description | Memory Device Pin Name | FPGA Pin Utilization |
|---|---|---|
| Data | DQ | DQ in the pin table, marked as Q in the Quartus II Pin Planner. Each DQ group has a common background color for all of the DQ and DM pins, associated with DQS (and DQSn) pins. The ×4 DIMM has the following mapping between DQS and DQ pins:<br>■ DQS[0] maps to DQ[3:0]<br>■ DQS[9] maps to DQ[7:4]<br>■ DQS[1] maps to DQ[11:8]<br>■ DQS[10] maps to DQ[15:12]<br>The DQS pin index in other DIMM configurations typically increases sequentially with the DQ pin index (DQS[0]: DQ[3:0]; DQS[1]: DQ[7:4]; DQS[2]: DQ[11:8]). In this DIMM configuration, the DQS pins are indiced this way to ensure pin out is compatible with both ×4 and ×8 DIMMs. |
| Data Mask | DM | |
| Data Strobe | DQS and DQSn | DQS and DQSn (S and Sbar in the Quartus II Pin Planner) |
| Address and Command | A[], BA[], CAS#, CKE, CS#, ODT, RAS#, WE#, RESET# | Any user I/O pin. To minimize skew, you should place address and command pins in the same bank or side of the device as the following pins: CK/CK# pins, DQ, DQS, or DM pins. Altera devices use the SSTL-15 I/O standard on the `RESET#` signal to meet the voltage requirements of 1.5 V CMOS at the memory device. Altera recommends that you do not terminate this signal to $V_{TT}$. |
| Memory system clock | CK and CK# | The first CK/CK# pairs (namely `mem_clk[0]` or `mem_clk_n[0]` in the IP) must use any unused DQ or DQS pins with `DIFFIO_RX` capability pins in the same bank or on the same side as the data pins. You can use either side of the device for wraparound interfaces. This placement is to allow the mimic path used in the IP VT tracking to go through differential I/O buffers to mimic the differential DQS signals. Any other CK/CK# pairs (`mem_clk[n:1]` and `mem_clk_n [n:1]`) can use any unused DQ or DQS pins in the same bank or on the same side as the data pins.<br>For Stratix V devices, you can assign the memory clocks to any unused DQ or DQS pins with `DIFFOUT`, `DIFFIO_TX`, or `DIFFIO_RX` capability for the `mem_clk[n:0]` and `mem_clk_n[n:0]` signals (where *n* is greater than or equal to zero). |
| Clock Source | — | Dedicated PLL clock input pin with direct (not using a global clock net) connection to the PLL and optional DLL required by the interface. |
| Reset | — | Dedicated clock input pin to accommodate the high fan-out signal |

Table 2–9 shows the FPGA pin utilization for QDR II and QDR II+ SRAM interfaces.

**Table 2–9. QDR II and QDR II+ SRAM Pin Utilization for Arria II, Stratix III, Stratix IV, and Stratix V Devices**

| Interface Pin Description | Memory Device Pin Name | FPGA Pin Utilization |
|---|---|---|
| Read Clock | CQ and CQn | For QDR II or QDR II+ SRAM devices with 2.5 cycles of read latency, connect CQ to DQS pin (S in the Quartus II Pin Planner), and CQn to CQn pin (Qbar in the Quartus II Pin Planner).<br>For QDR II or QDR II+ SRAM devices with 2.0 cycles of read latency, connect CQ to CQn pin (Qbar in the Quartus II Pin Planner), and CQn to DQS pin (S in the Quartus II Pin Planner). |
| Read Data | Q | DQ pins (Q in the Quartus II Pin Planner). Ensure that you are using the DQ pins associated with the chosen read clock pins (DQS and CQn pins). QVLD pins are only available for QDR II+ SRAM dvices and note that Altera IP does not use the QVLD pin. |
| Data Valid | QVLD | |
| Memory and Write Data Clock | K and K# | DQS and DQSn pins associated with the write data pins, S and Sbar in the Quartus II Pin Planner. |
| Write Data | D | DQ pins. Ensure that you are using the DQ pins associated with the chosen memory and write data clock pins (DQS and DQS pins). |
| Byte Write Select | BWS#, NWS# | |
| Address and Control Signals | A, WPS#, RPS# | Any user I/O pin. To minimize skew, you should place address and command pins in the same bank or side of the device as the following pins: K and K# pins, DQ, DQS, BWS#, and NWS# pins. If you are using burst-length-of-two devices, place the address signals in a DQS group pin as these signals are now double data rate. |
| Clock source | — | Dedicated PLL clock input pin with direct (not using a global clock net) connection to the PLL and optional DLL required by the interface. |
| Reset | — | Dedicated clock input pin to accommodate the high fan-out signal |

Table 2–10 shows the FPGA pin utilization for RLDRAM II CIO interfaces.

**Table 2–10. RLDRAM II CIO Pin Utilization for Arria II GZ, Stratix III, Stratix IV, and Stratix V Devices**

| Interface Pin Description | Memory Device Pin Name | FPGA Pin Utilization |
|---|---|---|
| Read Clock | QK and QK# | DQS and DQSn pins (S and Sbar in the Quartus II Pin Planner) |
| Data | Q | DQ pins (Q in the Quartus II Pin Planner). Ensure that you are using the DQ pins associated with the chosen read clock pins (DQS and DQSn pins). Altera IP does not use the QVLD pin. You may leave this pin unconnected on your board. You may not be able to fit these pins in a DQS group. Refer to "Exceptions for RLDRAM II Interfaces" on page 2–21, for more information on how to place these pins. |
| Data Valid | QVLD | |
| Data Mask | DM | |

**Table 2–10. RLDRAM II CIO Pin Utilization for Arria II GZ, Stratix III, Stratix IV, and Stratix V Devices**

| Interface Pin Description | Memory Device Pin Name | FPGA Pin Utilization |
|---|---|---|
| Write Data Clock | DK and DK# | DQ pins in the same DQS group as the read data (Q) pins or in adjacent DQS group or in the same bank as the address and command pins. Refer to "Exceptions for RLDRAM II Interfaces" on page 2–21, for more details. DK/DK# must use differential output-capable pins. |
| Memory Clock | CK and CK# | Any differential output-capable pins in the same bank as the address and command pins |
| Address and Control Signals | A, BA, CS#, REF#, WE# | Any user I/O pin. To minimize skew, you should place address and command pins in the same bank or side of the device as the following pins: CK/CK# pins, DQ, DQS, and DM pins. |
| Clock source | — | Dedicated PLL clock input pin with direct (not using a global clock net) connection to the PLL and optional DLL required by the interface. |
| Reset | — | Dedicated clock input pin to accommodate the high fan-out signal |

Table 2–11 shows the FPGA pin utilization for RLDRAM II SIO interfaces.

**Table 2–11. RLDRAM II SIO Pin Utilization Applicable for Arria II GZ, Stratix III, Stratix IV, and Stratix V Devices**

| Interface Pin Description | Memory Device Pin Name | FPGA Pin Utilization |
|---|---|---|
| Read Clock | QK and QK# | DQS and DQSn pins (S and Sbar in the Quartus II Pin Planner) in the same DQS group as the respective read data (Q) pins. |
| Read Data | Q | DQ pins (Q in the Quartus II Pin Planner). Ensure that you are using the DQ pins associated with the chosen read clock (DQS and DQSn) pins. Altera does not use the QVLD pin. You may leave this pin unconnected on your board. |
| Data valid | QVLD | |
| Memory and Write Data Clock | DK and DK# | DQS and DQSn pins (S and Sbar in the Quartus II Pin Planner) in the same DQS group as the respective write data (D) pins. |
| Write Data | D | DQ pins. Ensure that you are using the DQ pins associated with the chosen write data clock (DQS and DQSn) pins. |
| Data Mask | DM | |
| Memory Clock | CK and CK# | Any differential output-capable pins in the same bank as the address and command pins |
| Address and Control Signals | A, BA, CS#, REF#, WE# | Any user I/O pin. To minimize skew, you should place address and command pins in the same bank or side of the device as the following pins: CK/CK# pins, DQ, DQS, or DM pins. |
| Clock source | — | Dedicated PLL clock input pin with direct (not using a global clock net) connection to the PLL and optional DLL required by the interface. |
| Reset | — | Dedicated clock input pin to accommodate the high fan-out signal |

# PLLs and Clock Networks

The exact number of clocks and hence PLLs required in your design depends greatly on the memory interface frequency, and the IP that your design uses.

For example, you can build simple DDR slow-speed interfaces that typically require only two clocks: system and write. You can then use the rising and falling edges of these two clocks to derive four phases (0, 90, 180, and 270°). However, as clock speeds increase, the timing margin decreases and additional clocks are required, to optimize setup and hold and meet timing. Typically, at higher clock speeds, you need to have dedicated clocks for resynchronization, and address and command paths.

In addition, some memory controller designs that use the ALTMEMPHY megafunction, use a VT tracking clock to measure and compensate for VT changes and their effects.

Altera memory interface IP uses one PLL, which generates the various clocks needed in the memory interface data path and controller, and provides the required phase shifts for the write clock and address and command clock. The PLL is instantiated when you generate the Altera memory interface MegaCore® functions or megafunctions.

By default, the PLL also generates the input reference clock for the DLL, available in all device families except for the Cyclone III and Cyclone IV devices. This method eliminates the need of an extra pin for the DLL input reference clock.

The input reference clock to the DLL can come from certain input clock pins or clock output from certain PLLs.

For the actual pins and PLLs connected to the DLLs, refer to the *External Memory Interfaces* chapter of the relevant device family handbook.

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

Table 2–12 and Table 2–13 show a comparison of the number of PLLs and dedicated clock outputs available respectively in Arria II, Cyclone III, Cyclone IV, Stratix III, Stratix IV, and Stratix V devices.

**Table 2–12. Number of PLLs Available in Altera Device Families *(Note 1)***

| Device Family | Enhanced PLLs Available |
|---|---|
| Arria II GX | 4-6 |
| Arria II GZ | 3-8 |
| Cyclone III and Cyclone IV | 2-4 |
| Stratix III | 4-12 |
| Stratix IV | 3-12 |
| Stratix V (fPLL) | 22-28 |

**Note to Table 2–12:**

(1) For more details, refer to the *Clock Networks and PLL* chapter of the respective device family handbook.

**Table 2–13. Number of Enhanced PLL Clock Outputs and Dedicated Clock Outputs Available in Altera Device Families *(Note 1)***

| Device Family | Number of Enhanced PLL Clock Outputs | Number Dedicated Clock Outputs |
|---|---|---|
| Arria II GX *(2)* | 4 clock outputs each | 1 single-ended or 1 differential pair<br>3 single-ended or 3 differential pair total *(3)* |
| Cyclone III and Cyclone IV | 5 clock outputs each | 1 single-ended or 1 differential pair total (not for memory interface use) |
| Stratix III | Left/right: 7 clock outputs<br>Top/bottom: 10 clock outputs | Left/right: 2 single-ended or 1 differential pair<br>Top/bottom: 6 single-ended or 4 single-ended and 1 differential pair |
| Arria II GZ and Stratix IV | Left/right: 7 clock outputs<br>Top/bottom: 10 clock outputs | Left/right: 2 single-ended or 1 differential pair<br>Top/bottom: 6 single-ended or 4 single-ended and 1 differential pair |
| Stratix V | 18 clock outputs each | 4 single-ended or 2 single-ended and 1 differential pair |

**Notes to Table 2–13:**

(1) For more details, refer to the *Clock Networks and PLL* chapter of the respective device family handbook.

(2) PLL_5 and PLL_6 of Arria II GX devices do not have dedicated clock outputs.

(3) The same PLL clock outputs drives three single-ended or three differential I/O pairs, which are only supported in PLL_1 and PLL_3 of the EP2AGX95, EP2AGX125, EP2AGX190, and EP2AGX260 devices.

Table 2–14 shows the number of clock networks available in the Altera device families.

**Table 2–14. Number of Clock Networks Available in Altera Device Families** ***(Note 1)***

| Device Family | Global Clock Network | Regional Clock Network |
|---|---|---|
| Arria II GX | 16 | 48 |
| Arria II GZ | 16 | 64–88 |
| Cyclone III and Cyclone IV | 10-20 | N/A |
| Stratix III | 16 | 64–88 |
| Stratix IV | 16 | 64–88 |
| Stratix V | 16 | 92 |

**Note to Table 2–14:**

(1) For more information on the number of available clock network resources per device quadrant to better understand the number of clock networks available for your interface, refer to the *Clock Networks and PLL* chapter of the respective device family handbook.

You must decide whether you need to share clock networks, PLL clock outputs, or PLLs if you are implementing multiple memory interfaces.

Table 2–15 through Table 2–17 show the number of PLL outputs and clock networks required for the memory standards using Altera IP. Table 2–18 shows the names and frequency of the clocks used.

**Table 2–15. Clock Network Usage in ALTMEMPHY-based Memory Standards**

| Device | DDR3 SDRAM | | DDR2/DDR SDRAM | | | |
|---|---|---|---|---|---|---|
| | Half-Rate | | Half-Rate | | Full-Rate | |
| | Number of full-rate clock | Number of half-rate clock | Number of full-rate clock | Number of half-rate clock | Number of full-rate clock | Number of half-rate clock |
| Arria II GX | 4 global | 2 global | 4 global | 2 global | 5 global | 1 global |
| Cyclone III and Cyclone IV | — | | 4 global | 1 global | 5 global | |
| Stratix III and Stratix IV | 1 global<br>2 regional | 2 regional | 1 regional<br>2 dual-regional | 1 global<br>2 dual-regional | 1 global<br>2 dual-regional | 2 dual-regional |

**Table 2–16. Clock Network Usage in UniPHY-based Memory Standards—DDR2 and DDR3 SDRAM *(Note 1)***

| Device | DDR3 SDRAM | | DDR2 SDRAM | |
|---|---|---|---|---|
| | Half-Rate | | Half-Rate | |
| | Number of full-rate clock | Number of half-rate clock | Number of full-rate clock | Number of half-rate clock |
| Stratix III | 3 global | 1 global<br>1 regional | 1 global<br>2 global | 1 global<br>1 regional |
| Arria II GZ and Stratix IV | 3 global | 1 global<br>1 regional | 1 regional<br>2 regional | 1 global<br>1 regional |
| Stratix V | 1 global<br>2 regional | 2 global | 1 regional<br>2 regional | 2 global |

**Note to Table 2–16:**

(1) There are two additional regional clocks, `pll_avl_clk` and `pll_config_clk` for DDR2 and DDR3 SDRAM with UniPHY memory interfaces.

**Table 2–17. Clock Network Usage in UniPHY-based Memory Standards—RLDRAM II, and QDR II and QDR II+ SRAM**

| Device | RLDRAM II | | | | QDR II/QDR II+ SRAM | | | |
|---|---|---|---|---|---|---|---|---|
| | Half-Rate | | Full-Rate | | Half-Rate | | Full-Rate | |
| | Number of full-rate clock | Number of half-rate clock | Number of full-rate clock | Number of half-rate clock | Number of full-rate clock | Number of half-rate clock | Number of full-rate clock | Number of half-rate clock |
| Arria II GX | — | — | — | — | 2 global | 2 global | 4 global | — |
| Stratix III | 2 regional | 1 global<br>1 regional | 1 global<br>2 regional | — | 1 global<br>1 regional | 2 regional | 1 global<br>2 regional | — |
| Arria II GZ and Stratix IV | 2 regional | 1 global<br>1 regional | 1 global<br>2 regional | — | 1 global<br>1 regional | 2 regional | 1 global<br>2 regional | — |

For more information about the clocks used in UniPHY-based memory standards, refer to the respective sections in *Volume 3: Implementing Altera Memory Interface IP* of the *External Memory Interface Handbook*.

**Table 2–18. Clocks Used in the ALTMEMPHY Megafunction *(Note 1)***

| Clock Name | Usage Description |
|---|---|
| phy_clk_1x | Static system clock for the half-rate data path and controller. |
| mem_clk_2x | Static DQS output clock that generates DQS, CK/CK# signals, the input reference clock to the DLL, and the system clock for the full-rate datapath and controller. |
| mem_clk_1x | This clock drives the aux_clk output or clocking DQS and as a reference clock for the memory devices. |
| write_clk_2x | Static DQ output clock used to generate DQ signals at 90º earlier than DQS signals. Also may generate the address and command signals. |
| mem_clk_ext_2x | This clock is only used if the memory clock generation uses dedicated output pins. Applicable only in HardCopy® II or Stratix II prototyping for HardCopy II designs. |
| resync_clk_2x | Dynamic-phase clock used for resynchronization and postamble paths. Currently, this clock cannot be shared by multiple interfaces. |
| measure_clk_2x/ measure_clk_1x *(2)* | Dynamic-phase clock used for VT tracking purposes. Currently, this clock cannot be shared by multiple interfaces. |
| ac_clk_2x ac_clk_1x | Dedicated static clock for address and command signals. |
| scan_clk | Static clock to reconfigure the PLL |
| seq_clk | Static clock for the sequencer logic |

**Notes to Table 2–18:**

(1) For more information on the clocks used in the ALTMEMPHY megafunction, refer to the *Clock Networks and PLL* chapter of the respective device family handbook for more details.

(2) This clock should be of the same clock network clock as the resync_clk_2x clock.

In every ALTMEMPHY solution, the measure_clk and resync_clk_2x clocks (Table 2–18) are calibrated and hence may not be shared or used for other modules in your system. You may be able to share the other statically phase-shifted clocks with other modules in your system provided that you do not change the clock network used.

Changing the clock network that the ALTMEMPHY solution uses may affect the output jitter, especially if the clock is used to generate the memory interface output pins. Always check the clock network output jitter specification in the *DC and Switching Characteristics* chapter of the device handbook, before changing the ALTMEMPHY clock network, to ensure that it meets the memory standard jitter specifications, which includes period jitter, cycle-to-cycle jitter and half duty cycle jitter.

If you need to change the resync_clk_2x clock network, you have to change the measure_clk_1x clock network also to ensure accurate VT tracking of the memory interface.

For more information about sharing clocks in multiple controllers, refer to *Volume 6: Design Flow Tutorials* of the *External Memory Interface Handbook*.

In addition, you should not change the PLL clock numbers as the wizard-generated **.sdc** file assumes certain counter outputs from the PLL (Table 2–19 through Table 2–20).

**Table 2–19. PLL Usage for DDR, DDR2, and DDR3 SDRAM Without Leveling Interfaces**

| Clock | Arria II GX Devices | Cyclone III and Cyclone IV Devices | Stratix III and Stratix IV Devices |
|---|---|---|---|
| C0 | ■ phy_clk_1x in half-rate designs<br>■ aux_half_rate_clk<br>■ PLL scan_clk | ■ phy_clk_1x in half-rate designs<br>■ aux_half_rate_clk | ■ phy_clk_1x in half-rate designs<br>■ aux_half_rate_clk<br>■ PLL scan_clk |
| C1 | ■ phy_clk_1x in full-rate designs<br>■ aux_full_rate_clk<br>■ mem_clk_2x to generate DQS and CK/CK# signals<br>■ ac_clk_2x<br>■ cs_n_clk_2x | ■ phy_clk_1x in full-rate designs<br>■ aux_full_rate_clk<br>■ mem_clk_2x to generate DQS and CK/CK# signals<br>■ ac_clk_2x<br>■ cs_n_clk_2x | ■ mem_clk_2x |
| C2 | ■ Unused | ■ write_clk_2x (for DQ)<br>■ ac_clk_2x<br>■ cs_n_clk_2x | ■ phy_clk_1x in full-rate designs<br>■ aux_full_rate_clk |
| C3 | ■ write_clk_2x (for DQ)<br>■ ac_clk_2x<br>■ cs_n_clk_2x | ■ resync_clk_2x | ■ write_clk_2x |
| C4 | ■ resync_clk_2x | ■ measure_clk_2x | ■ resync_clk_2x |
| C5 | ■ measure_clk_2x | — | ■ measure_clk_1x |
| C6 | — | — | ■ ac_clk_1x |

**Table 2–20. PLL Usage for DDR3 SDRAM With Leveling Interfaces**

| Clock | Stratix III and Stratix IV Devices |
|---|---|
| C0 | ■ phy_clk_1x in half-rate designs<br>■ aux_half_rate_clk<br>■ PLL scan_clk |
| C1 | ■ mem_clk_2x |
| C2 | ■ aux_full_rate_clk |
| C3 | ■ write_clk_2x |
| C4 | ■ resync_clk_2x |
| C5 | ■ measure_clk_1x |
| C6 | ■ ac_clk_1x |

# PLL Cascading

Arria II GZ PLLs, Stratix III PLLs, Stratix IV PLLs, Stratix V fPLLs, and the two middle PLLs in Arria II GX EP2AGX95, EP2AGX125, EP2AGX190, and EP2AGX260 devices can be cascaded using either the global or regional clock trees, or the cascade path between two adjacent PLLs.

Use this feature at your own risk. You should use faster memory devices to maximize timing margins.

Cyclone III and Cyclone IV devices do not support PLL cascading for external memory interfaces.

Both ALTMEMPHY and UniPHY IP support PLL cascading using the cascade path without any additional timing derating when the bandwidth and compensation rules are followed. The timing constraints and analysis assume that there is no additional jitter due to PLL cascading when the upstream PLL uses no compensation and low bandwidth, and the downstream PLL uses no compensation and high bandwidth.

Both ALTMEMPHY and UniPHY IP do not support PLL cascading using the global and regional clock networks. You can implement PLL cascading at your own risk without any additional guidance and specifications from Altera. The Quartus II software does issue a critical warning suggesting use of the cascade path to minimize jitter, but does not explicitly state that Altera does not support cascading using global and regional clock networks.

The Quartus II software does not issue a critical warning stating that Cyclone III and Cyclone IV ALTMEMPHY designs do not support PLL cascading; it issues the Stratix III warning message requiring use of cascade path.

Some Arria II GX devices (EP2AGX95, EP2AGX125, EP2AGX190, and EP2AGX260) have direct cascade path for two middle right PLLs. Arria II GX PLLs have the same bandwidth options as Stratix IV GX left and right PLLs.

# DLL

The Altera memory interface IP uses one DLL (except in Cyclone III and Cyclone IV devices, where this resource is not available). The DLL is located at the corner of the device and can send the control signals to shift the DQS pins on its adjacent sides for Stratix-series devices, or DQS pins in any I/O banks in Arria II GX devices.

For example, the top-left DLL can shift DQS pins on the top side and left side of the device. The DLL generates the same phase shift resolution for both sides, but can generate different phase offset to the two different sides, if needed. Each DQS pin can be configured to use or ignore the phase offset generated by the DLL.

The DLL cannot generate two different phase offsets to the same side of the device. However, you can use two different DLLs to for this functionality.

DLL reference clocks must come from either dedicated clock input pins located on either side of the DLL or from specific PLL output clocks. Any clock running at the memory frequency is valid for the DLLs.

To minimize the number of clocks routed directly on the PCB, typically this reference clock is sourced from the memory controllers PLL. In general, DLLs can use the PLLs directly adjacent to them (corner PLLs when available) or the closest PLL located in the two sides adjacent to its location.

By default, the DLL reference clock in Altera external memory IP is from a PLL output.

When designing for 780-pin packages with EP3SE80, EP3SE110, EP3SL150, EP4SE230, EP4SE360, EP4SGX180, and EP4SGX230 devices, the PLL to DLL reference clock connection is limited.

DLL2 is isolated from a direct PLL connection and can only receive a reference clock externally from pins `clk[11:4]p` in EP3SE80, EP3SE110, EP3SL150, EP4SE230, and EP4SE360 devices. In EP4SGX180 and EP4SGX230 devices, DLL2 and DLL3 are not directly connected to PLL. DLL2 and DLL3 receive a reference clock externally from pins `clk[7:4]p` and `clk[15:12]p` respectively.

Refer to the respective device handbooks for more DLL information.

The DLL reference clock should be the same frequency as the memory interface, but the phase is not important.

The required DQS capture phase is optimally chosen based on operating frequency and external memory interface type (DDR, DDR2, DDR3 SDRAM, and QDR II SRAM, or RLDRAM II). As each DLL supports two possible phase offsets, two different memory interface types operating at the same frequency can easily share a single DLL. More may be possible, depending on the phase shift required.

Altera memory IP always specifies a default optimal phase setting, to override this setting, refer to the *DDR and DDR2 SDRAM Controller with ALTMEMPHY IP User Guide* section and the *DDR3 SDRAM Controller with ALTMEMPHY IP User Guide* section in volume 3 of the *External Memory Interface Handbook*.

When sharing DLLs, your memory interfaces must be of the same frequency. If the required phase shift is different amongst the multiple memory interfaces, you can use a different delay chain in the DQS logic block or use the DLL phase offset feature.

To simplify the interface to IP connections, multiple memory interfaces operating at the same frequency usually share the same system and static clocks as each other where possible. This sharing minimizes the number of dedicated clock nets required and reduces the number of different clock domains found within the same design.

As each DLL can directly drive four banks, but each PLL only has complete C (output) counter coverage of two banks (using dual regional networks), situations can occur where a second PLL operating at the same frequency is required. As cascaded PLLs increase jitter and reduce timing margin, you are advised to first ascertain if an alternative second DLL and PLL combination is not available and more optimal.

Select a DLL that is available for the side of the device where the memory interface resides. If you select a PLL or a PLL input clock reference pin that can also serve as the DLL input reference clock, you do not need an extra input pin for the DLL input reference clock.

# Other FPGA Resources

The Altera memory interface IP uses FPGA fabric, including registers and the Memory Block to implement the memory interface.

For resource utilization examples to ensure that you can fit your other modules in the device, refer to the relevant sections in *Volume 3: Implementing Altera Memory Interface IP* of the *External Memory Interface Handbook*.

In addition, one OCT calibration block is used if you are using the FPGA OCT feature in the memory interface.The OCT calibration block uses two pins ($R_{UP}$ and $R_{DN}$), or single pin ($R_{ZQ}$) ("OCT Support for Arria II GZ, Stratix III, Stratix IV, and Stratix V Devices" on page 2–13). You can select any of the available OCT calibration block as you do not need to place this block in the same bank or device side of your memory interface. The only requirement is that the I/O bank where you place the OCT calibration block uses the same $V_{CCIO}$ voltage as the memory interface. You can share multiple memory interfaces with the same OCT calibration block if the $V_{CCIO}$ voltage is the same.

Even though Cyclone III and Cyclone IV devices support OCT, this feature is not turned on by default in the Altera IP solution.

# 3. Memory IP Planning

This chapter gives you an overview of Altera external memory controllers and PHY IP.

Altera FPGAs achieve optimal memory interface performance with external memory IP. The IP comprises two key components:

- PHY—the physical layer that guarantees the best timing margin and latency for the interface between FPGA and memories
- Controller—the state machine that guarantees efficient data transfers between FPGA and memories.

Altera provides modular memory solutions, which allow you to customize your memory interface design. You can build a custom PHY, a custom controller, or both, as desired.

Table 3–1 shows the recommended memory types and controllers that Altera offers with the PHY IP.

**Table 3–1. Altera Memory Types, PHY, and Controllers in the Quartus II Software (Part 1 of 2)**

| Quartus II Version | Memory | PHY IP | Controller IP |
|---|---|---|---|
| 10.1 | DDR/DDR2/DDR3 | ALTMEMPHY (AFI) | HPC<br>HPC II |
| | DDR2/DDR3 | UniPHY Nios-based Sequencer | HPC II |
| | QDR II/QDR II+ | UniPHY RTL Sequencer | QDR/RLD II controller |
| | RLDRAM II | UniPHY RTL Sequencer | QDR/RLD II controller |
| | Other | ALTDQ_DQS *(1)* | Custom |
| | Other | ALTDQ_DQS2 *(2)* | Custom |
| 10.0 | DDR/DDR2/DDR3 | ALTMEMPHY (AFI) | HPC<br>HPC II |
| | DDR2/DDR3 | UniPHY Nios-based Sequencer | HPC II |
| | QDR II/QDR II+ | UniPHY RTL Sequencer | QDR/RLD II controller |
| | RLDRAM II | UniPHY RTL Sequencer | QDR/RLD II controller |
| | Other | ALTDQ_DQS *(1)* | Custom |
| | Other | ALTDQ_DQS2 *(2)* | Custom |
| 9.1 | DDR/DDR2/DDR3 | ALTMEMPHY (AFI) | HPC<br>HPC II |
| | QDR II/QDR II+ | UniPHY | QDR II controller |
| | RLDRAM II | UniPHY | RLDRAM II controller |
| | Other | ALTDQ_DQS | Custom |

**Table 3–1. Altera Memory Types, PHY, and Controllers in the Quartus II Software (Part 2 of 2)**

| Quartus II Version | Memory | PHY IP | Controller IP |
|---|---|---|---|
| 9.0 | DDR/DDR2/DDR3 | ALTMEMPHY (AFI) *(3)* | High-performance controller (HPC) |
| | QDR II/QDR II+ | ALTMEMPHY | — |
| | RLDRAM II | Custom | Custom |
| | Other | ALTDQ_DQS | Custom |

**Note to Table 3–1:**

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

(2) Applicable only for Stratix V devices.

(3) AFI = Altera PHY interface

For more information about the controllers with the UniPHY or the ALTMEMPHY IP, refer to *Volume 3: Implementing Altera Memory Interface IP* of the *External Memory Interface Handbook*.
For more information about the ALTDQ_DQS megafunction, refer to the *ALTDLL and ALTDQ_DQS Megafunctions User Guide*.
For more information about the ALTDQ_DQS2 megafunction, refer to the *ALTDQ_DQS2 Megafunction User Guide*.

Altera recommends that you use the HPC II architecture with the UniPHY IP for all new designs because Altera will continue optimizing the UniPHY IP and the HPC II architecture.

# PHY IP

Altera offers two PHY IP options: ALTMEMPHY and UniPHY. The UniPHY IP is the latest PHY architecture that has several enhanced features to support the needs of high-performance applications.

Table 3–2 lists the benefits of using the UniPHY IP.

**Table 3–2. UniPHY IP Features and Benefits (Part 1 of 2)**

| Feature | Benefit |
|---|---|
| Latency | ■ The latency is half of the ALTMEMPHY IP.<br>■ The system performance for random accesses is improved more than 50%. |
| PLL, DLL and OCT sharing | Enables multiple memory interfaces on single chip. |
| Configuration support | Supports mainstream configurations for DDR2 and DDR3 SDRAM, QDR II and QDR II+ SRAM, RLDRAM II controllers with various widths, burst sizes, DIMM types, and multiple ranks through the Quartus II software. |

**Table 3–2. UniPHY IP Features and Benefits (Part 2 of 2)**

| Feature | Benefit |
|---|---|
| Calibration algorithm | Allows for higher performance through advanced calibration algorithms with improved debugging capability and easier customization. |
| Design flow | Has easier design flow with the following features:<br>■ Modular clear text code<br>■ Transparent and flexible timing model<br>■ Improved testbenches for faster development and easier debugging |

While the UniPHY IP is a more enhanced technology, a limited number of the ALTMEMPHY configurations may show some advantages in resource utilization and simulation time.

☞ Altera's next generation product families will not support the ALTMEMPHY IP.

## Resource Utilization

Use the appropriate PHY IPs based on the requirements of your design.

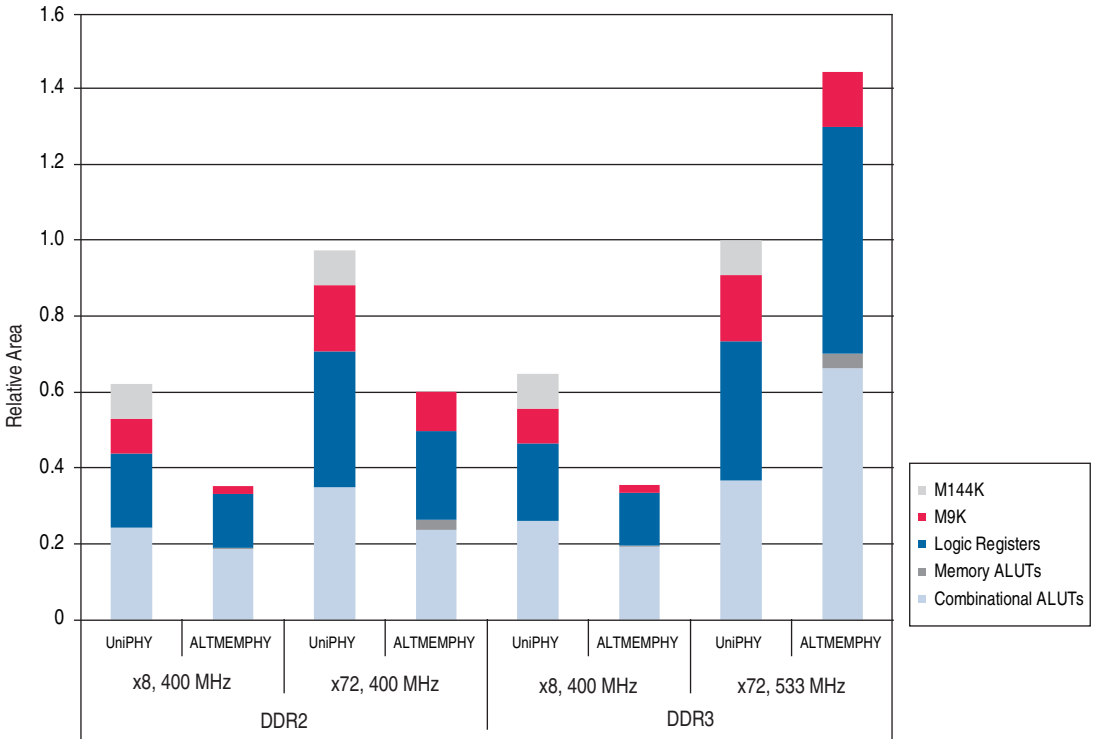
Table 3–3 lists the comparisons between the UniPHY and the ALTMEMPHY IP for resource utilization.

**Table 3–3. Resource Utilization Comparisons for the UniPHY and ALTMEMPHY IP**

| UniPHY IP | ALTMEMPHY IP |
|---|---|
| ■ The UniPHY IP is more efficient in resource utilization for wide interfaces of higher performance.<br>■ The UniPHY IP has advanced calibration algorithms which, while more robust, may consume more resources.<br>■ For example, for 533-MHz 72-bit DDR3 interface, the normalized area (LEs and internal memory), for the UniPHY IP is 35% smaller than that of the ALTMEMPHY IP. | ■ The ALTMEMPHY IP consumes less resources for smaller interfaces.<br>■ The area for the ALTMEMPHY IP is smaller for slower and narrow DDR2 interfaces because the calibration sequencer state machines for these configurations are smaller.<br>■ For example, for 400-MHz 8-bit DDR2 interface, the normalized area for the ALTMEMPHY IP is 40% smaller than that of the UniPHY IP. |

Figure 3–1 shows the comparisons between the UniPHY and the ALTMEMPHY IP for area utilization.

**Figure 3–1. Area Utilization Comparison Chart for the UniPHY and ALTMEMPHY IP**




As all new feature development focuses on the UniPHY IP only, Altera will continue to optimize area for future Quartus II releases.

For specific configurations, Altera recommends that you check the resource utilization that the Quartus II software reports. For detailed UniPHY resource utilization data, refer to the "Resource Utilization" section in the *DDR2 and DDR3 SDRAM Controller with UniPHY IP User Guide*, *RLDRAM II Controller with UniPHY IP User Guide*, and *QDR II and QDR II+ SRAM Controller with UniPHY IP User Guide* in volume 3 of the *External Memory Interface Handbook*.

## Recommended IP for Your FPGA Device

Table 3–4 shows the recommended PHY options for different FPGA devices and memory types.

**Table 3–4. Recommended PHY Options Available in the Quartus II Software 10.1**

| Family | PHY Support | | | | | |
|---|---|---|---|---|---|---|
| | DDR SDRAM | DDR2 SDRAM | DDR3 SDRAM | RLDRAM II | QDR II SRAM | QDR II+ SRAM |
| Arria II GX | ALTMEMPHY | ALTMEMPHY | ALTMEMPHY | — | UniPHY | UniPHY |
| Arria II GZ | — | UniPHY | UniPHY | UniPHY | UniPHY | UniPHY |
| Cyclone III and Cyclone IV E (1.2 V) | ALTMEMPHY | ALTMEMPHY | — | — | — | — |
| Cyclone IV E (1.0 V) | ALTMEMPHY | ALTMEMPHY | — | — | — | — |
| Cyclone IV GX | ALTMEMPHY | ALTMEMPHY | — | — | — | — |
| HardCopy III and HardCopy IV | ALTMEMPHY | UniPHY | UniPHY | UniPHY | UniPHY | UniPHY |
| Stratix III | ALTMEMPHY | UniPHY | UniPHY | UniPHY | UniPHY | UniPHY |
| Stratix IV E/GX/GT | ALTMEMPHY | UniPHY | UniPHY | UniPHY | UniPHY | UniPHY |
| Stratix V | — | UniPHY | UniPHY | UniPHY | UniPHY | UniPHY |

This chapter provides additional information about the document and Altera.

## Document Revision History

The following table shows the revision history for this document.

| Date | Version | Changes |
|---|---|---|
| December 2010 | 2.1 | ■ Added a new section: Controller Efficiency.<br>■ Added Arria II GX and Stratix V information. |
| July 2010 | 2.0 | Updated information about UniPHY-based interfaces and Stratix V devices. |
| April 2010 | 1.0 | Initial release. |

## How to Contact Altera

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

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

**Note to Table:**

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

## Typographic Conventions

The following table shows the typographic conventions this document uses.

| Visual Cue | Meaning |
|---|---|
| **Bold Type with Initial Capital Letters** | Indicate command names, dialog box titles, dialog box options, and other GUI labels. For example, **Save As** dialog box. For GUI elements, capitalization matches the GUI. |
| **bold type** | Indicates directory names, project names, disk drive names, file names, file name extensions, software utility names, and GUI labels. For example, **\qdesigns** directory, **D:** drive, and **chiptrip.gdf** file. |
| *Italic Type with Initial Capital Letters* | Indicate document titles. For example, *Stratix IV Design Guidelines*. |

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

# External Memory Interface Handbook Volume 2

## Section II. Board Planning

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

EMI_PLAN_BOARD-2.1

Document last updated for Altera Complete Design Suite version: 10.1
Document publication date: December 2010

Subscribe

# Contents

## Chapter 3. Dual-DIMM DDR2 and DDR3 SDRAM Interface Termination, Drive Strength, Loading, and Board Layout Guidelines

## Chapter 4. RLDRAM II Interface Termination and Layout Guidelines



## Chapter 5. QDR II SRAM Interface Termination and Layout Guidelines



## Chapter 6. Power Estimation Methods for External Memory Interface Designs

# 1. DDR2 SDRAM Interface Termination, Drive Strength, Loading, and Board Layout Guidelines

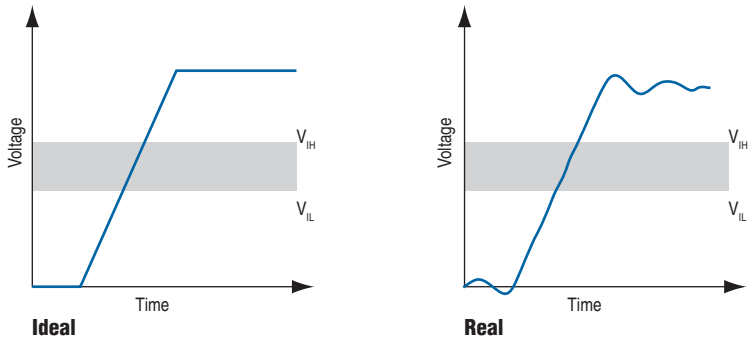
This chapter recommends the ideal termination schemes for the DDR2 SDRAM interface with Arria® GX, Arria II GX, Cyclone® II, Cyclone III, Cyclone IV, Stratix® III, and Stratix IV devices.

This chapter focuses on the following key factors that affect signal quality at the receiver:

- Proper use of termination
- Output driver drive strength setting
- Loading at the receiver
- Layout guidelines

As memory interface performance increases, board designers must pay closer attention to the quality of the signal seen at the receiver because poorly transmitted signals can dramatically reduce the overall data-valid margin at the receiver. Figure 1–1 shows the differences between an ideal and real signal seen by the receiver.

**Figure 1–1. Ideal and Real Signal at the Receiver**




In addition, this chapter compares various types of termination schemes, and their effects on the signal quality on the receiver. It also discusses the proper drive strength setting on the FPGA to optimize the signal integrity at the receiver, and the effects of different loading types, such as components versus DIMM configuration, on signal quality. Finally, this chapter provides DDR2 SDRAM layout guidelines.

The objective of this chapter to understand the trade-offs between different types of termination schemes, the effects of output drive strengths, and different loading types, so you can swiftly navigate through the multiple combinations and choose the best possible settings for your designs.

## Board Termination

In board-level design, you can use a variety of termination schemes. DDR2 adheres to the JEDEC standard of governing Stub-Series Terminated Logic (SSTL), JESD8-15a, which includes four different termination schemes.

Two commonly used termination schemes of SSTL are:

- Single parallel terminated output load with or without series resistors (Class I, as stated in JESD8-15a)
- Double parallel terminated output load with or without series resistors (Class II, as stated in JESD8-15a)

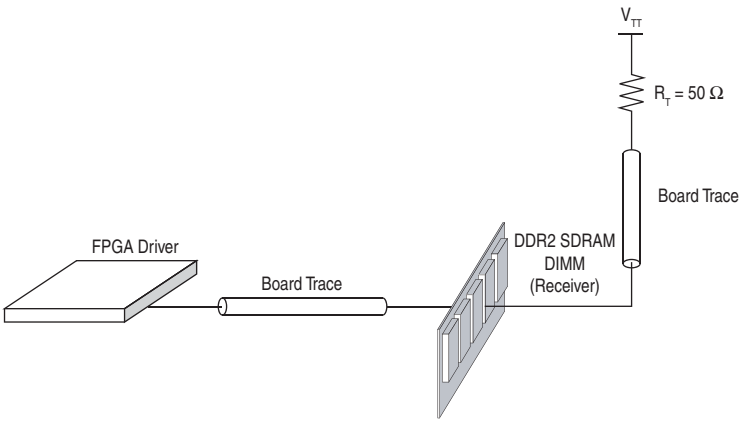
Depending on the type of signals you choose, you can use either termination scheme. Also, depending on your design's FPGA and SDRAM memory devices, you may choose external or internal termination schemes.

With the ever-increasing requirements to reduce system cost and simplify printed circuit board (PCB) layout design, you may choose not to have any parallel termination on the transmission line, and use point-to-point connections between the memory interface and the memory. In this case, you may take advantage of internal termination schemes such as on-chip termination (OCT) on the FPGA side and on-die termination (ODT) on the SDRAM side when it is offered on your chosen device.

## External Parallel Termination

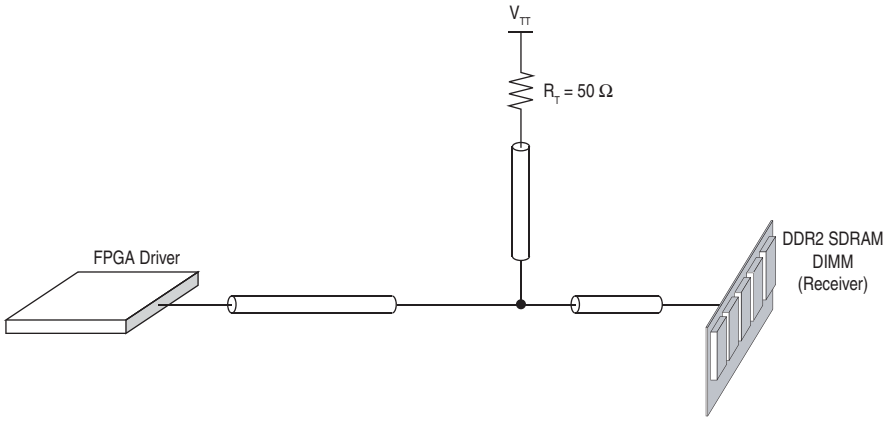
If you use external termination, you must study the locations of the termination resistors to determine which topology works best for your design. Figure 1–2 and Figure 1–3 illustrate the two most commonly used termination topologies: fly-by topology and non-fly-by topology, respectively.

**Figure 1–2. Fly-By Placement of a Parallel Resistor**



With fly-by topology (Figure 1–2), you place the parallel termination resistor after the receiver. This termination placement resolves the undesirable unterminated stub found in the non-fly-by topology. However, using this topology can be costly and complicate routing. The Stratix II Memory Board 2 uses the fly-by topology for the parallel terminating resistors placement. The Stratix II Memory Board 2 is a memory test board available only within Altera for the purpose of testing and validating Altera's memory interface.

**Figure 1–3. Non-Fly-By Placement of a Parallel Resistor**




With non-fly-by topology (Figure 1–3), the parallel termination resistor is placed between the driver and receiver (closest to the receiver). This termination placement is easier for board layout, but results in a short stub, which causes an unterminated transmission line between the terminating resistor and the receiver. The unterminated transmission line results in ringing and reflection at the receiver.

If you do not use external termination, DDR2 offers ODT and Altera® FPGAs have varying levels of OCT support. You should explore using ODT and OCT to decrease the board power consumption and reduce the required board real estate.

## On-Chip Termination

OCT technology is offered on Arria II GX, Cyclone III, Stratix III, and Stratix IV devices. Table 1–1 summarizes the extent of OCT support for each device. This table provides information about SSTL-18 standards because SSTL-18 is the supported standard for DDR2 memory interface by Altera FPGAs.

On-chip series ($R_S$) termination is supported only on output and bidirectional buffers. The value of $R_S$ with calibration is calibrated against a 25-Ω resistor for class II and 50-Ω resistor for class I connected to $R_{UP}$ and $R_{DN}$ pins and adjusted to ± 1% of 25 Ω or 50 Ω. On-chip parallel ($R_T$) termination is supported only on inputs and bidirectional buffers. The value of $R_T$ is calibrated against 100 Ω connected to the $R_{UP}$ and $R_{DN}$ pins. Calibration occurs at the end of device configuration. Dynamic OCT is supported only on bidirectional I/O buffers.

**Table 1–1. On-Chip Termination Schemes**

| Termination Scheme | SSTL-18 | FPGA Device | | | | | |
|---|---|---|---|---|---|---|---|
| | | Stratix III and Stratix IV | | Cyclone III and Cyclone IV | | Arria II GX | |
| | | Column I/O | Row I/O | Column I/O | Row I/O | Column I/O | Row I/O |
| On-Chip Series Termination without Calibration | Class I | 50 | 50 | 50 | 50 | 50 | 50 |
| | Class II | 25 | 25 | 25 | 25 | 25 | — |
| On-Chip Series Termination with Calibration | Class I | 50 | 50 | 50 | 50 | — | — |
| | Class II | 25 | 25 | 25 | 25 | — | — |
| On-Chip Parallel Termination with Calibration | Class I and Class II | 50 | 50 | — | — | — | — |

**Note to Table 1–1:**

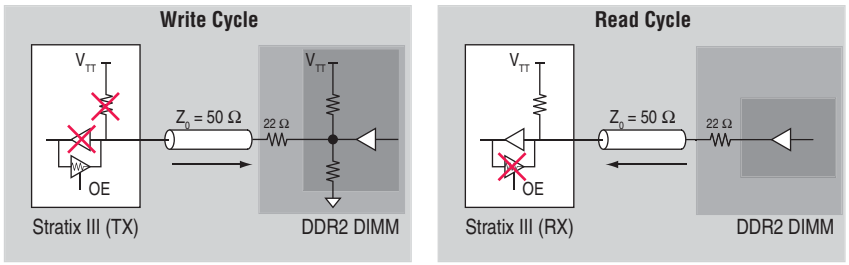
(1) Programmable Drive Strength

(2) Non-dynamic on-chip parallel termination is only supported for input pins.

The dynamic OCT scheme is only available in Stratix III and Stratix IV FPGAs. The dynamic OCT scheme enables series termination (Rs) and parallel termination (Rt) to be dynamically turned on and off during the data transfer.

The series and parallel terminations are turned on or off depending on the read and write cycle of the interface. During the write cycle, the $R_S$ is turned on and the $R_T$ is turned off to match the line impedance. During the read cycle, the $R_S$ is turned off and the $R_T$ is turned on as the Stratix III FPGA implements the far-end termination of the bus (Figure 1–4).

**Figure 1–4. Dynamic OCT for Memory Interfaces**




## Simulation and Measurement Setup

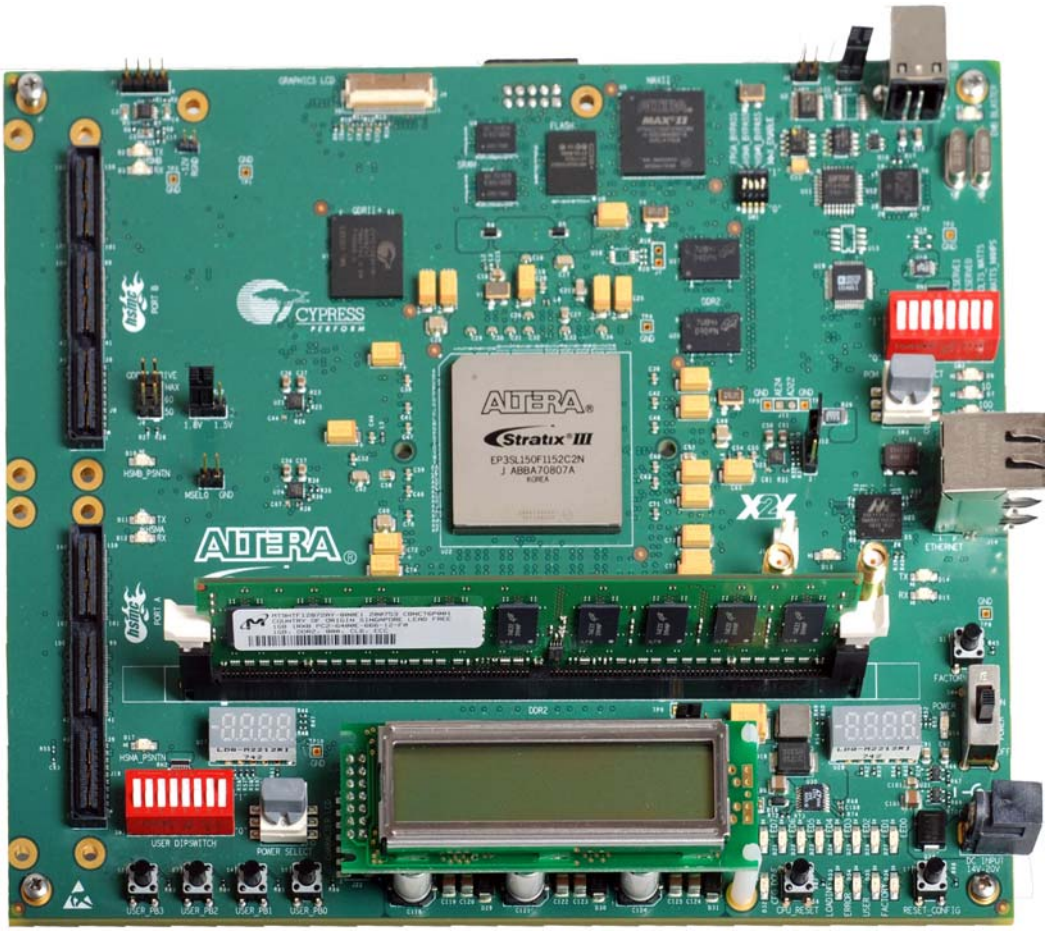
To study the different types of termination schemes properly, the simulation and measurement setups described in this chapter use two options:

- Altera Stratix III FPGA interfacing with a 400-MHz DDR2 SDRAM unbuffered DIMM (UDIMM)
- Altera Stratix II FPGA interfacing with a 267-MHz DDR2 SDRAM UDIMM

The maximum achievable frequency in Stratix II DDR2 SDRAM is 333 MHz. The 267 MHz frequency is due to the device speed grade mounted on the Stratix II Memory Board 2 board.

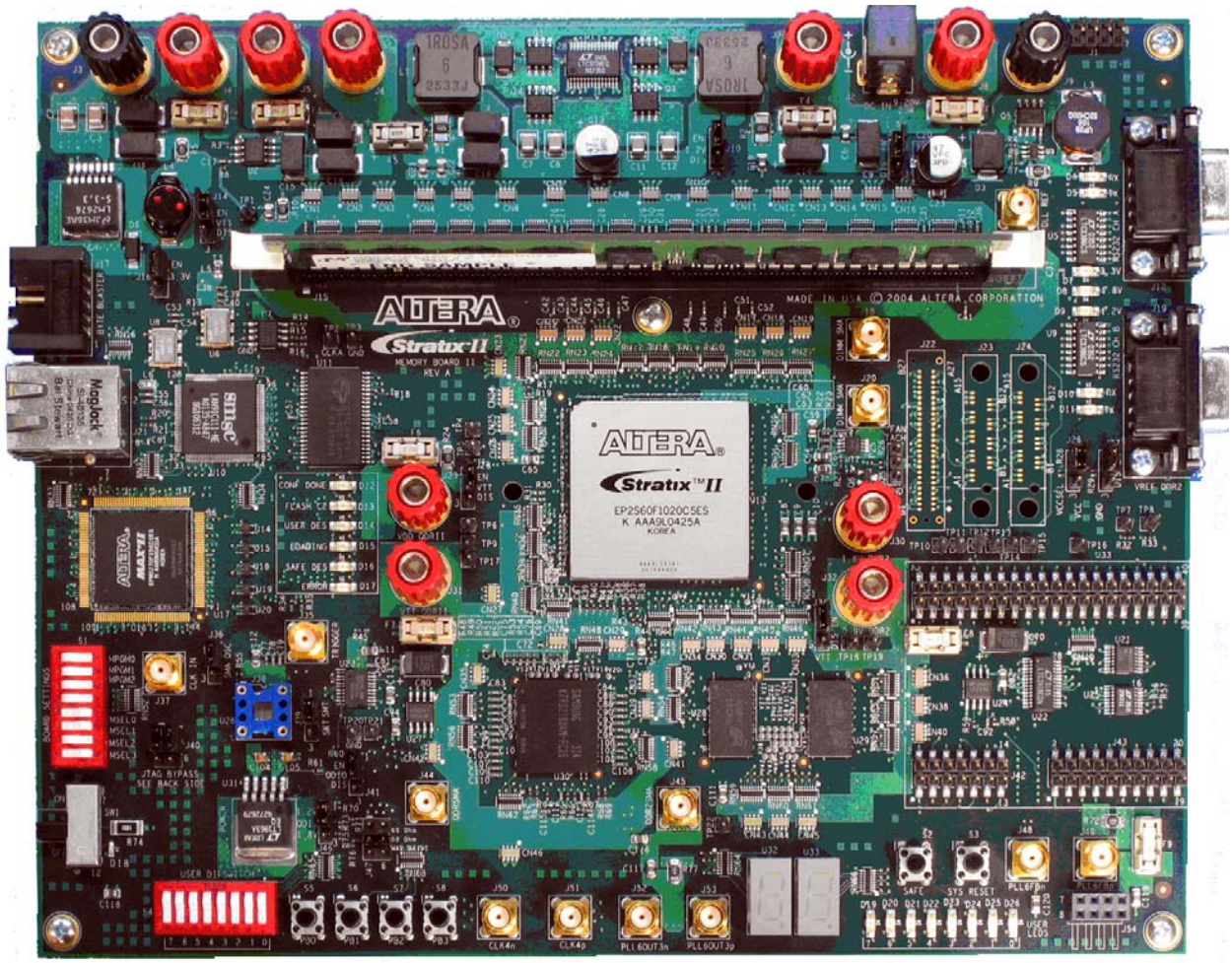
The Stratix III FPGA is interfacing with a 400-MHz DDR2 SDRAM UDIMM. This DDR2 memory interface is built on the Stratix III Host Development Kit Board (Figure 1–5). This board is available for purchase at the Altera web site.

**Figure 1–5. Stratix III Host Development Kit Board with DDR2 SDRAM DIMM Interface**



The Altera Stratix II FPGA is interfacing with a 267-MHz DDR2 SDRAM UDIMM. This DDR2 memory interface is built on the Stratix II Memory Board 2 (Figure 1–6). and is not available to customers. This board is used internally within Altera for validation and testing of memory interfaces with Stratix II devices.

**Figure 1–6. Stratix II Memory Board 2 with DDR2 SDRAM DIMM Interface**




The DDR2 SDRAM DIMM on both Stratix III and Stratix II boards contains a 22-Ω external series termination resistor for each data line, so all of the measurements and simulations must account for the effect of these series termination resistors.

To correlate the bench measurements performed on the Stratix III Host Development Kit Board and the Stratix II Memory Board 2, the simulations are performed using HyperLynx® LineSim software with IBIS models from Altera and memory vendors. Figure 1–7 and Figure 1–8 show the setup in HyperLynx used for the simulation for Stratix III and Stratix II boards. The simulation files, **Simulation Example** are on the Altera website.

**Figure 1–7. HyperLynx Setup for Stratix III Host Development Kit Board with DDR2 SDRAM DIMM Interface Simulation**

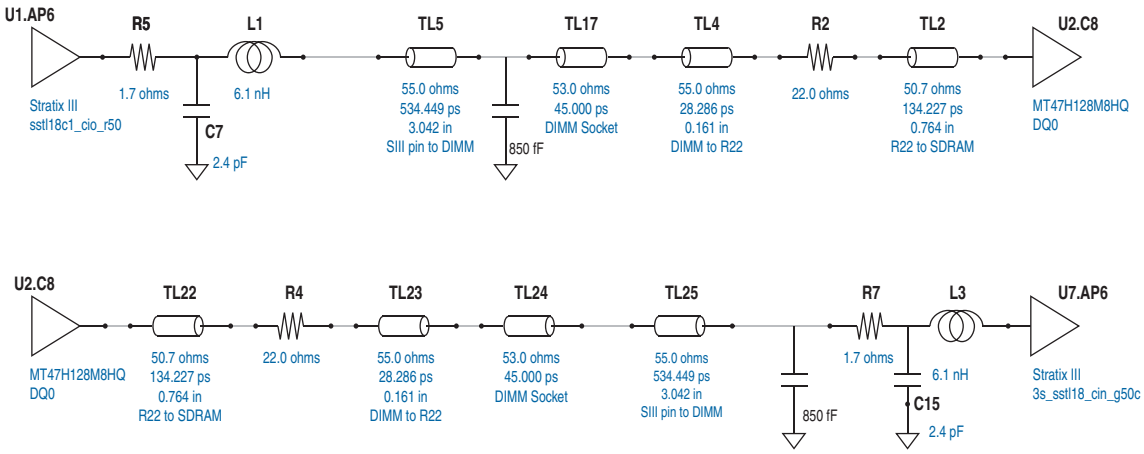
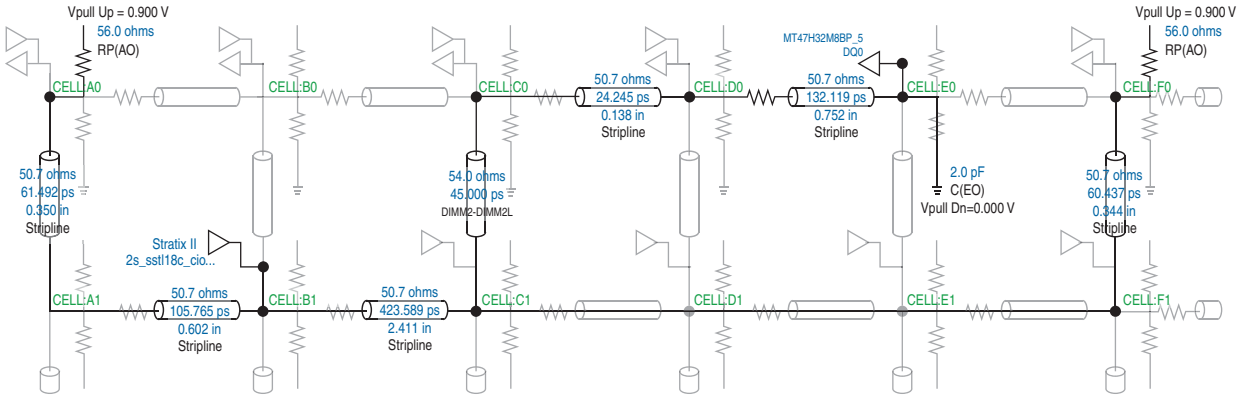



**Figure 1–8. HyperLynx Setup for Stratix II Memory Board 2 with DDR2 SDRAM DIMM Interface Simulation**




The trace length of DQ0 from DDR2 SDRAM memory interface in the FPGA to DQ0 at DDR2 SDRAM DIMM is extracted for the simulation and is approximately 3 inches long for both the Stratix III device on the Stratix III Development Kit Board and the Stratix II device on the Stratix II Memory Board 2.

In Figure 1–7, resistance, inductance, and capacitance (RLC) values of the Stratix III FPGA I/O package are extracted from **Stratix3_rlc.xls** document in the Altera Device IBIS Models available on the Altera IBIS Models page at Altera's web site. At the point of measurements, 0.85 pF of capacitance was added to represent the scope cable.

The trace information for DDR2 SDRAM DIMM on the Stratix II Memory Board 2 can be found in the *PC4300 DDR2 SDRAM Unbuffered DIMM Design Specification.*

The trace information for DDR2 SDRAM DIMM on the Stratix III Host Development Kit Board can be found in the *PC5300/6400 DDR2 SDRAM Unbuffered DIMM Design Specification*.

## Recommended Termination Schemes

Table 1–2 provides the recommended termination schemes for major DDR2 memory interface signals. Signals include data (DQ), data strobe (DQS/DQSn), data mask (DM), clocks (`mem_clk/mem_clk_n`), and address and command signals.

When interfacing with multiple DDR2 SDRAM components where the address, command, and memory clock pins are connected to more than one load, follow these steps:

1. Simulate the system to get the new slew-rate for these signals.
2. Use the derated tIS and tIH specifications from the DDR2 SDRAM datasheet based on the simulation results.
3. If timing deration causes your interface to fail timing requirements, consider signal duplication of these signals to lower their loading, and hence improve timing.

Altera uses Class I and Class II termination in this table to refer to drive strength, and not physical termination.

You must simulate your design for your system to ensure correct functionality.

**Table 1–2. Termination Recommendations (Part 1 of 3) *(Note 1)***

| Device Family | Signal Type | SSTL 18 IO Standard *(2)*, *(3)*, *(4)*, *(5)*, *(6)* | FPGA End Discrete Termination | Memory End Termination 1 Rank/DIMM | Memory I/O Standard |
|---|---|---|---|---|---|
| **Arria II GX** | | | | | |
| DDR2 component | DQ/DQS | Class I 8 mA | 50 Ω Parallel to $V_{TT}$ discrete | ODT75 *(7)* | HALF *(7)* |
| | DM | Class I 8 mA | N/A | 56 Ω parallel to $V_{TT}$ discrete | N/A |
| | Address and command | Class I MAX | N/A | | N/A |
| | Clock | Class I 8 mA | N/A | ×1 = 100 Ω differential *(10)*<br>×2 = 200 Ω differential *(11)* | N/A |
| DDR2 DIMM | DQ/DQS | Class I 8 mA | 50 Ω Parallel to $V_{TT}$ discrete | ODT75 *(7)* | FULL *(9)* |
| | DM | Class I 8 mA | N/A | 56 Ω parallel to $V_{TT}$ discrete | N/A |
| | Address and command | Class I MAX | N/A | | N/A |
| | Clock | Class I 8 mA | N/A | N/A = on DIMM | N/A |

**Table 1–2. Termination Recommendations (Part 2 of 3)** *(Note 1)*

| Device Family | Signal Type | SSTL 18 IO Standard *(2)*, *(3)*, *(4)*, *(5)*, *(6)* | FPGA End Discrete Termination | Memory End Termination 1 Rank/DIMM | Memory I/O Standard |
|---|---|---|---|---|---|
| **Cyclone III and Cyclone IV** | | | | | |
| DDR2 component | DQ/DQS | Class I 12 mA | 50 Ω Parallel to $V_{TT}$ discrete | ODT75 *(7)* | HALF *(8)* |
| | DM | Class I 12 mA | N/A | 56 Ω parallel to $V_{TT}$ discrete | N/A |
| | Address and command | Class I MAX | N/A | | N/A |
| | Clock | Class I 12 mA | N/A | ×1 = 100 Ω differential *(10)*<br>×2 = 200 Ω differential *(11)* | N/A |
| DDR2 DIMM | DQ/DQS | Class I 12 mA | 50 Ω Parallel to $V_{TT}$ discrete | ODT75 *(7)* | FULL *(9)* |
| | DM | Class I12 mA | N/A | 56 Ω parallel to $V_{TT}$ discrete | N/A |
| | Address and command | Class I MAX | N/A | | N/A |
| | Clock | Class I 12 mA | N/A | N/A = on DIMM | N/A |
| **Stratix III and Stratix IV** | | | | | |
| DDR2 component | DQ/DQS | Class I R50/P50 DYN CAL | N/A | ODT75 *(7)* | HALF *(7)* |
| | DM | Class I R50 CAL | N/A | ODT75 *(7)* | N/A |
| | Address and command | Class I R50 CAL | N/A | 56 Ω Parallel to $V_{TT}$ discrete | N/A |
| | Clock | DIFF Class I R50 NO CAL | N/A | x1 = 100 Ω differential *(10)* | N/A |
| | | DIFF Class I R50 NO CAL | N/A | X2 = 200 Ω differential *(11)* | N/A |
| | DQS DIFF recommended | DIFF Class I R50/P50 DYN CAL | N/A | ODT75 *(7)* | HALF *(7)* |
| | DQS SE *(12)* | Class I R50/P50 DYN CAL | N/A | ODT75 *(7)* | HALF *(7)* |

**Table 1–2. Termination Recommendations (Part 3 of 3)** *(Note 1)*

| Device Family | Signal Type | SSTL 18 IO Standard *(2)*, *(3)*, *(4)*, *(5)*, *(6)* | FPGA End Discrete Termination | Memory End Termination 1 Rank/DIMM | Memory I/O Standard |
|---|---|---|---|---|---|
| DDR2 DIMM | DQ/DQS | Class I R50/P50 DYN CAL | N/A | ODT75 *(7)* | FULL *(9)* |
| | DM | Class I R50 CAL | N/A | ODT75 *(7)* | N/A |
| | Address and command | Class I MAX | N/A | 56 Ω Parallel to $V_{TT}$ discrete | N/A |
| | Clock | DIFF Class I R50 NO CAL | N/A | N/A = on DIMM | N/A |
| | DQS DIFF recommended | DIFF Class I R50/P50 DYN CAL | N/A | ODT75 *(7)* | FULL *(9)* |
| | DQS SE *(12)* | Class I R50/P50 DYN CAL | N/A | ODT75 *(7)* | FULL *(9)* |

**Notes to Table 1–2:**

(1) N/A is not available.
(2) R is series resistor.
(3) P is parallel resistor.
(4) DYN is dynamic OCT.
(5) NO CAL is OCT without calibration.
(6) CAL is OCT with calibration.
(7) ODT75 vs. ODT50 on the memory has the effect of opening the eye more, with a limited increase in overshoot/undershoot.
(8) HALF is reduced drive strength.
(9) FULL is full drive strength.
(10) x1 is a single-device load.
(11) x2 is two-device load.
(12) DQS SE is single-ended DQS.

## Dynamic On-Chip Termination

The termination schemes are described in JEDEC standard JESD8-15a for SSTL 18 I/O. Dynamic OCT is available in Stratix III and Stratix IV. When the Stratix III FPGA (driver) is writing to the DDR2 SDRAM DIMM (receiver), series OCT is enabled dynamically to match the impedance of the transmission line. As a result, reflections are significantly reduced. Similarly, when the FPGA is reading from the DDR2 SDRAM DIMM, the parallel OCT is dynamically enabled.

For information about setting the proper value for termination resistors, refer to the *Stratix III Device I/O Features* chapter in the *Stratix III Device Handbook* and the *I/O Features in Stratix IV Devices* chapter in the *Stratix IV Device Handbook*..

## FPGA Writing to Memory

Figure 1–9 shows dynamic series OCT scheme when the FPGA is writing to the memory. The benefit of using dynamic series OCT is that when driver is driving the transmission line, it "sees" a matched transmission line with no external resistor termination.

**Figure 1–9. Dynamic Series OCT Scheme with ODT on the Memory**

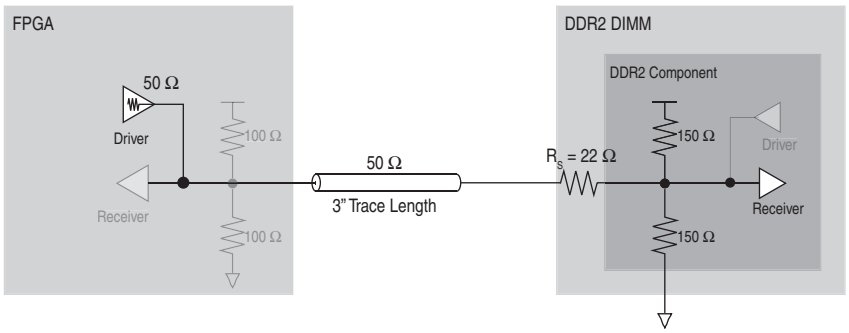


Figure 1–10 and Figure 1–11 show the simulation and measurement results of a write to the DDR2 SDRAM DIMM. The system uses Class I termination with a 50-Ω series OCT measured at the DIMM with a full drive strength and a 75 Ω ODT at the DIMM. Both simulation and bench measurements are in 200 pS/div and 200 mV/div.

**Figure 1–10. HyperLynx Simulation FPGA Writing to Memory**

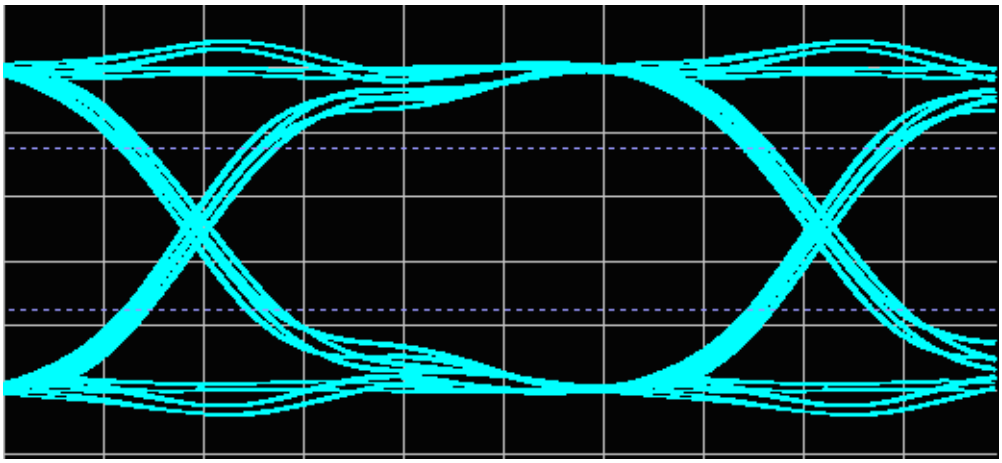


**Figure 1–11. Board Measurement, FPGA Writing to Memory**

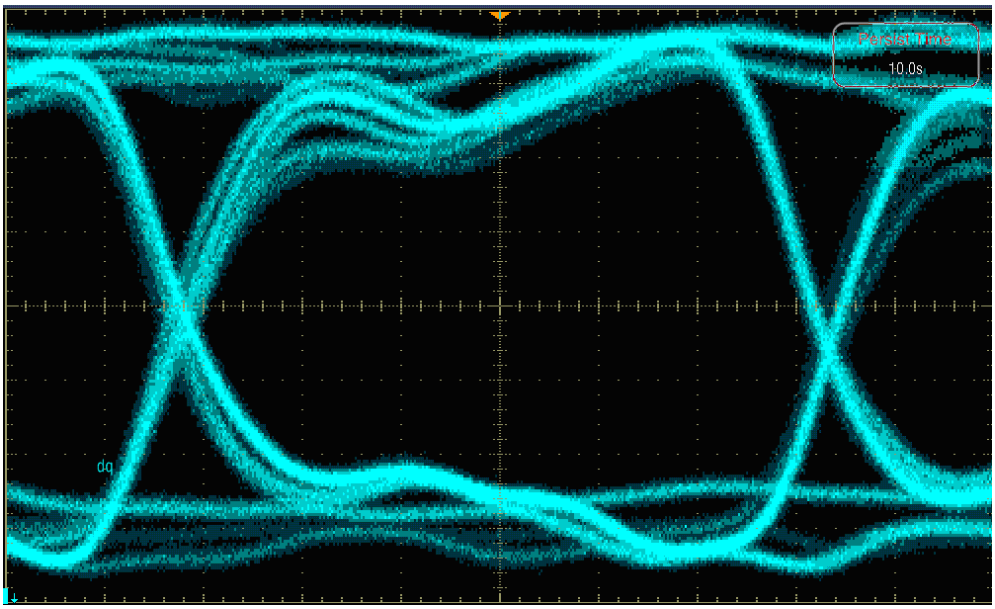


Table 1–3 summarizes the comparison between the simulation and the board measurement of the signal seen at the DDR2 SDRAM DIMM.

**Table 1–3. Signal Comparison When the FPGA is Writing to the Memory** ***(Note 1)***

| | Eye Width (ns) *(2)* | Eye Height (V) | Overshoot (V) | Undershoot (V) |
|---|---|---|---|---|
| Simulation | 1.194 | 0.740 | N/A | N/A |
| Board Measurement | 1.08 | 0.7 | N/A | N/A |

**Notes to Table 1–3:**

(1) N/A is not applicable.

(2) The eye width is measured from $V_{IH}/V_{IL}$(ac) = VREF ±250 mV to $V_{IH}/V_{IL}$(dc) = VREF ±125 mV, where $V_{IH}$ and $V_{IL}$ are determined per the JEDEC specification for SSTL-18.

The data in Table 1–3 and Figure 1–10 and Figure 1–11 suggest that when the FPGA is writing to the memory, the bench measurements are closely matched with simulation measurements. They indicate that using the series dynamic on-chip termination scheme for your bidirectional I/Os maintains the integrity of the signal, while it removes the need for external termination.

Depending on the I/O standard, you should consider the four parameters listed in Table 1–3 when designing a memory interface. Although the simulation and board measurement appear to be similar, there are some discrepancies when the key parameters are measured. Although simulation does not fully model the duty cycle distortion of the I/O, crosstalk, or board power plane degradation, it provides a good indication on the performance of the board.

For memory interfaces, the eye width is important when determining if there is a sufficient window to correctly capture the data. Regarding the eye height, even though most memory interfaces use voltage-referenced I/O standards (in this case, SSTL-18), as long as there is sufficient eye opening below and above VIL and VIH, there should be enough margin to correctly capture the data. However, because effects such as crosstalk are not taken into account, it is critical to design a system to achieve the optimum eye height, because it impacts the overall margin of a system with a memory interface.

Refer to the memory vendors when determining the over- and undershoot. They typically specify a maximum limit on the input voltage to prevent reliability issues.

## FPGA Reading from Memory

Figure 1–12 shows the dynamic parallel termination scheme when the FPGA is reading from memory. When the DDR2 SDRAM DIMM is driving the transmission line, the ringing and reflection is minimal because the FPGA-side termination 50-Ω pull-up resistor is matched with the transmission line. Figure 1–13 shows the simulation and measurement results of a read from DDR2 SDRAM DIMM. The system uses Class I termination with a 50-Ω calibrated parallel OCT measured at the FPGA end with a full drive strength and a 75-Ω ODT at the memory. Both simulation and bench measurements are in 200 pS/div and 200 mV/div.

**Figure 1–12. Dynamic Parallel OCT Scheme with Memory-Side Series Resistor**

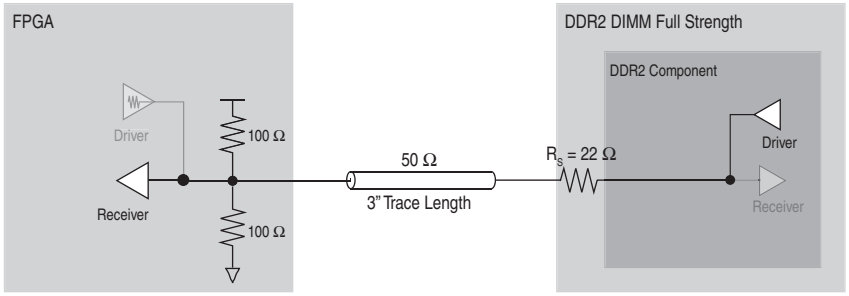


**Figure 1–13. Hyperlynx Simulation and Board Measurement, FPGA Reading from Memory**

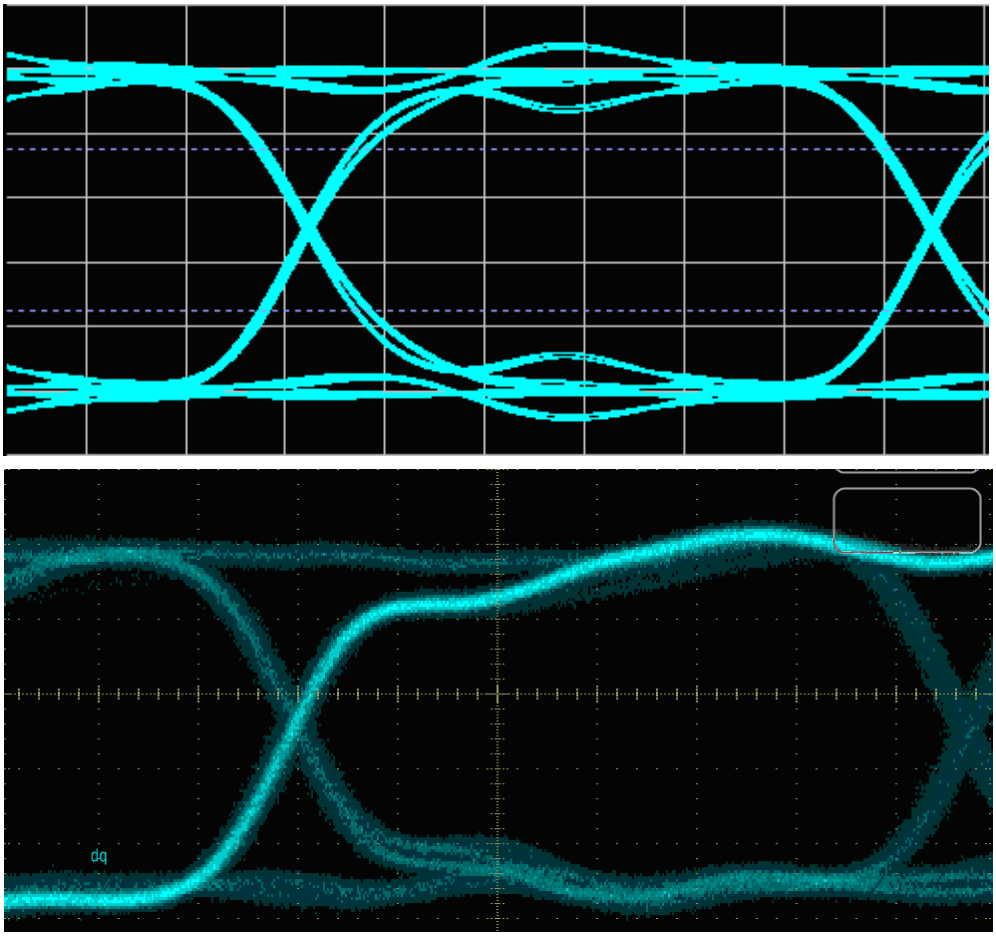



Table 1–4 summarizes the comparison between the simulation and the board measurement of the signal seen at the FPGA end.

**Table 1–4. Signal Comparison When the FPGA is Reading from the Memory *(Note 1)*, *(2)***

| | Eye Width (ns) *(3)* | Eye Height (V) | Overshoot (V) | Undershoot (V) |
|---|---|---|---|---|
| Simulation | 1.206 | 0.740 | N/A | N/A |
| Board Measurement | 1.140 | 0.680 | N/A | N/A |

**Notes to Table 1–4:**

(1) The drive strength on the memory DIMM is set to Full.

(2) N/A is not applicable.

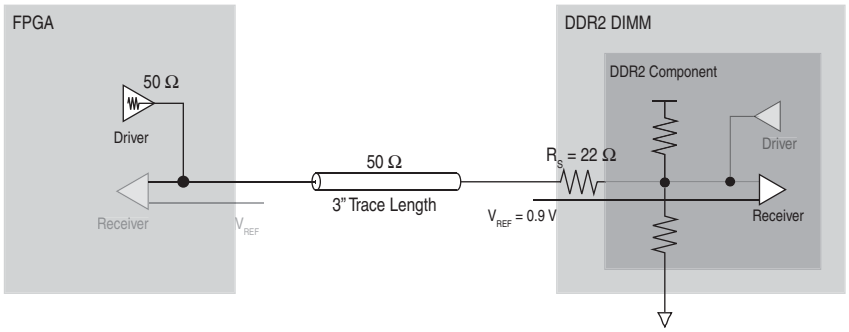
(3) The eye width is measured from $V_{IH}/V_{IL}$(ac) = VREF ±250 mV to $V_{IH}/V_{IL}$(dc) = VREF ±125 mV, in which $V_{IH}$ and $V_{IL}$ are determined per the JEDEC specification for SSTL-18.

The data in Table 1–4 and Figure 1–13 suggest that bench measurements are closely matched with simulation measurements when the FPGA is reading from the memory. They indicate that using the parallel dynamic on-chip termination scheme in bidirectional I/Os maintains the integrity of the signal, while it removes the need for external termination.

## On-Chip Termination (Non-Dynamic)

When you use the 50-Ω OCT feature in a Class I termination scheme using ODT with a memory-side series resistor, the output driver is tuned to 50 Ω, which matches the characteristic impedance of the transmission line. Figure 1–14 shows the Class I termination scheme using ODT when the 50-Ω OCT on the FPGA is turned on.

**Figure 1–14. Class I Termination Using ODT with 50-Ω OCT**




The resulting signal quality has a similar eye opening to the 8 mA drive strength setting (refer to "Drive Strength" on page 1–34) without any over- or undershoot. Figure 1–15 shows the simulation and measurement of the signal at the memory side (DDR2 SDRAM DIMM) with the drive strength setting of 50-Ω OCT in the FPGA.

**Figure 1–15. HyperLynx Simulation and Measurement, FPGA Writing to Memory**

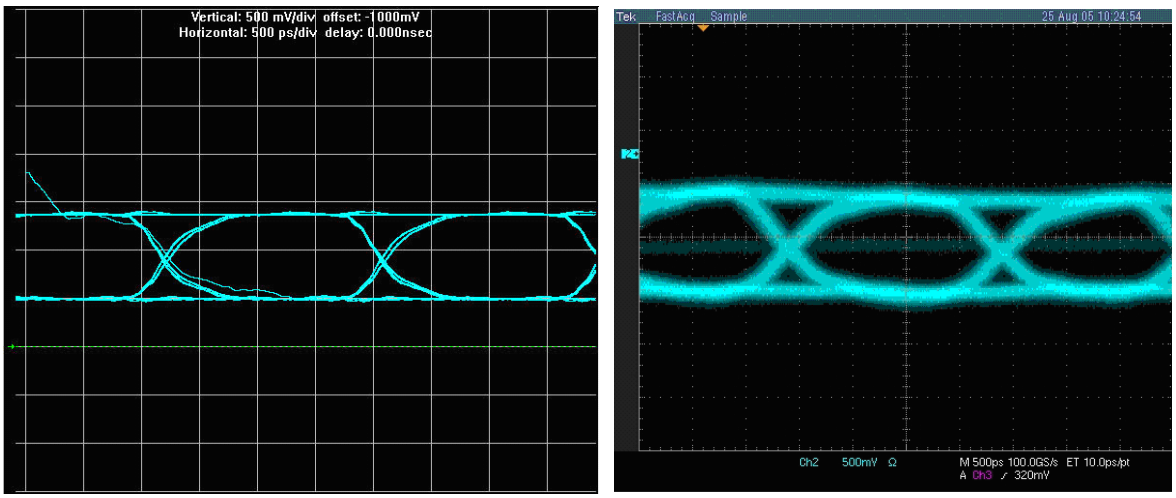


Table 1–5 shows data for the signal at the DDR2 SDRAM DIMM of a Class I scheme termination using ODT with a memory-side series resistor. The FPGA is writing to the memory with 50-Ω OCT.

**Table 1–5. Simulation and Board Measurement Results for 50-Ω OCT and 8-mA Drive Strength Settings *(Note 1)***

| | Eye Width (ns) | Eye Height (V) | Overshoot (V) | Undershoot (V) |
|---|---|---|---|---|
| **50-Ω OCT Drive Strength Setting** | | | | |
| Simulation | 1.68 | 0.82 | N/A | N/A |
| Board Measurement | 1.30 | 0.70 | N/A | N/A |

**Note to Table 1–5:**

(1) N/A is not applicable.

When you use the 50-Ω OCT setting on the FPGA, the signal quality for the Class I termination using ODT with a memory-side series resistor is further improved with lower over- and undershoot.

In addition to the 50-Ω OCT setting, Stratix II devices have a 25-Ω OCT setting that you can use to improve the signal quality in a Class II terminated transmission line. Figure 1–16 shows the Class II termination scheme using ODT when the 25-Ω OCT on the FPGA is turned on.

**Figure 1–16. Class II Termination Using ODT with 25-Ω OCT**

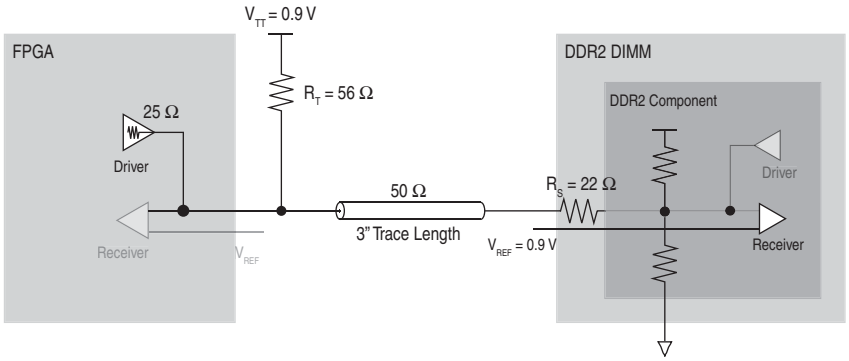


Figure 1–17 shows the simulation and measurement of the signal at the DDR2 SDRAM DIMM (receiver) with a drive strength setting of 25-Ω OCT in the FPGA.

**Figure 1–17. HyperLynx Simulation and Measurement, FPGA Writing to Memory**

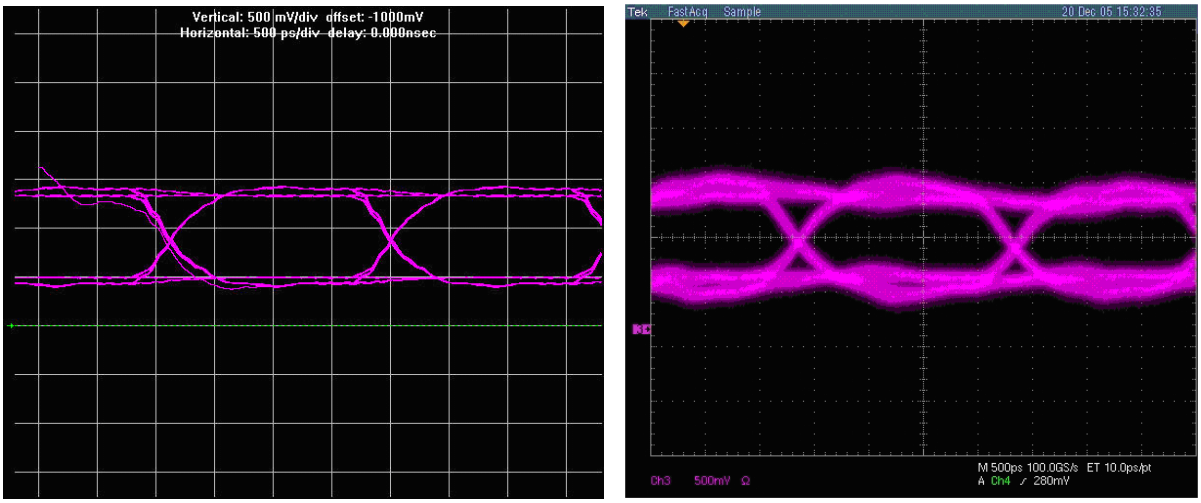



Table 1–6 shows the data for the signal at the DDR2 SDRAM DIMM of a Class II termination with a memory-side series resistor. The FPGA is writing to the memory with 25-Ω OCT.

**Table 1–6. Simulation and Board Measurement Results for 25-Ω OCT and 16-mA Drive Strength Settings *(Note 1)***

| | Eye Width (ns) | Eye Height (V) | Overshoot (V) | Undershoot (V) |
|---|---|---|---|---|
| **25-Ω OCT Drive Strength Setting** | | | | |
| Simulation | 1.70 | 0.81 | N/A | N/A |
| Board Measurement | 1.47 | 0.51 | N/A | N/A |

**Note to Table 1–6:**

(1) N/A is not applicable.

This type of termination scheme is only used for bidirectional signals, such as data (DQ), data strobe (DQS), data mask (DM), and memory clocks (CK) found in DRAMs.

## Class II External Parallel Termination

The double parallel (Class II) termination scheme is described in JEDEC standards JESD8-6 for HSTL I/O, JESD8-9b for SSTL-2 I/O, and JESD8-15a for SSTL-18 I/O. When the FPGA (driver) is writing to the DDR2 SDRAM DIMM (receiver), the transmission line is terminated at the DDR2 SDRAM DIMM. Similarly, when the FPGA is reading from the DDR2 SDRAM DIMM, the DDR2 SDRAM DIMM is now the driver and the transmission line is terminated at the FPGA (receiver). This type of termination scheme is typically used for bidirectional signals, such as data (DQ) and data strobe (DQS) signal found in DRAMs.

## FPGA Writing to Memory

Figure 1–18 shows the Class II termination scheme when the FPGA is writing to the memory. The benefit of using Class II termination is that when either driver is driving the transmission line, it sees a matched transmission line because of the termination resistor at the receiver-end, thereby reducing ringing and reflection.

**Figure 1–18. Class-II Termination Scheme with Memory-Side Series Resistor**

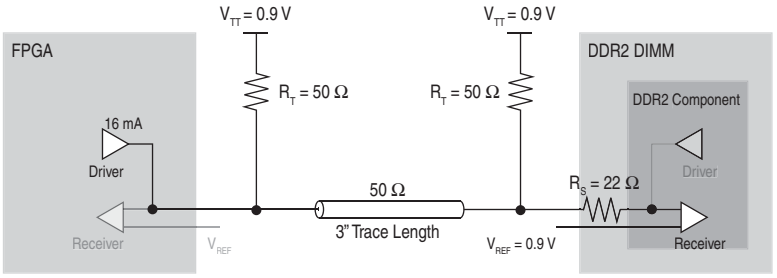
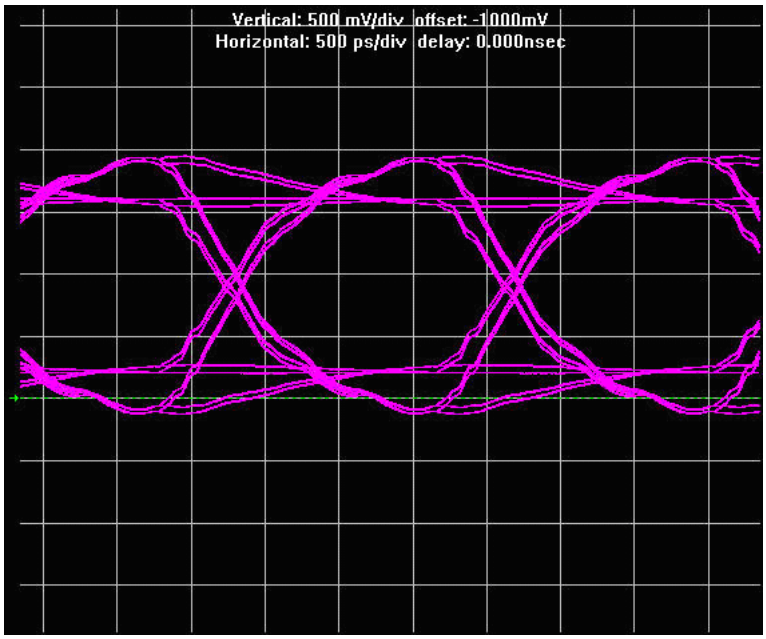



Figure 1–19 and Figure 1–20 show the simulation and measurement result of a write to the DDR2 SDRAM DIMM. The system uses Class II termination with a source-series resistor measured at the DIMM with a drive strength setting of 16 mA.

**Figure 1–19. HyperLynx Simulation, FPGA Writing to Memory**




The simulation shows a clean signal with a good eye opening, but there is slight over- and undershoot of the 1.8-V signal specified by DDR2 SDRAM. The over- and undershoot can be attributed to either overdriving the transmission line using a higher than required drive strength setting on the driver or the over-termination on the receiver side by using an external resistor value that is higher than the characteristic impedance of the transmission line. As long as the over- and undershoot do not exceed the absolute maximum rating specification listed in the memory

vendor's DDR2 SDRAM data sheet, it does not result in any reliability issues. The simulation results are then correlated with actual board level measurements. Figure 1–20 shows the measurement obtained from the Stratix II Memory Board 2. The FPGA is using a 16 mA drive strength to drive the DDR2 SDRAM DIMM on a Class II termination transmission line.

**Figure 1–20. Board Measurement, FPGA Writing to Memory**

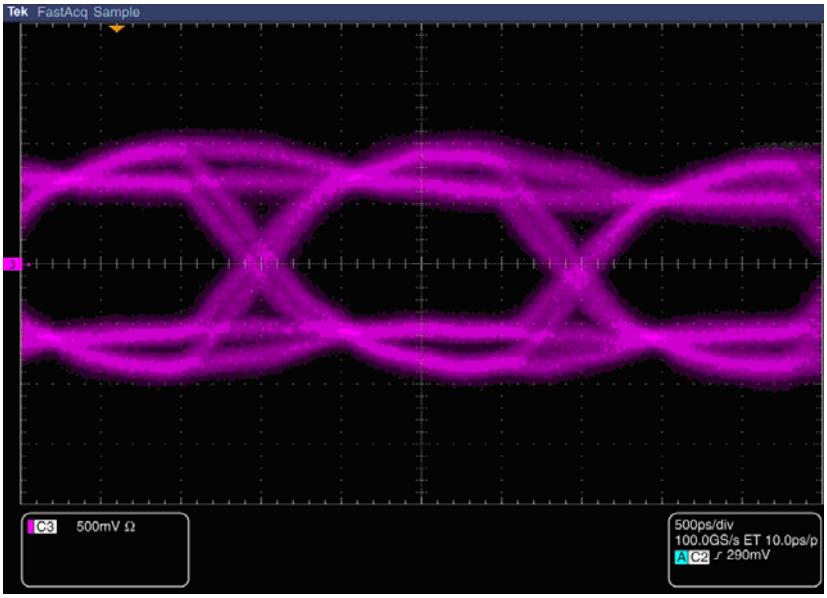



Table 1–7 summarizes the comparison between the simulation and the board measurement of the signal seen at the DDR2 SDRAM DIMM.

**Table 1–7. Signal Comparison When the FPGA is Writing to the Memory *(Note 1)***

| | Eye Width (ns) *(2)* | Eye Height (V) | Overshoot (V) | Undershoot (V) |
|---|---|---|---|---|
| Simulation | 1.65 | 1.28 | 0.16 | 0.14 |
| Board Measurement | 1.35 | 0.83 | 0.16 | 0.18 |

**Notes to Table 1–7:**

(1) The drive strength on the FPGA is set to 16 mA.

(2) The eye width is measured from $V_{REF}$ ± 125 mV where $V_{IH}$ and $V_{IL}$ are determined per the JEDEC specification for SSTL-18.

A closer inspection of the simulation shows an ideal duty cycle of 50%–50%, while the board measurement shows that the duty cycle is non-ideal, around 53%–47%, resulting in the difference between the simulation and measured eye width. In addition, the board measurement is conducted on a 72-bit memory interface, but the simulation is performed on a single I/O.

## FPGA Reading from Memory

Figure 1–21 shows the Class II termination scheme when the FPGA is reading from memory. When the DDR2 SDRAM DIMM is driving the transmission line, the ringing and reflection is minimal because of the matched FPGA-side termination pull-up resistor with the transmission line.

**Figure 1–21. Class II Termination Scheme with Memory-Side Series Resistor**

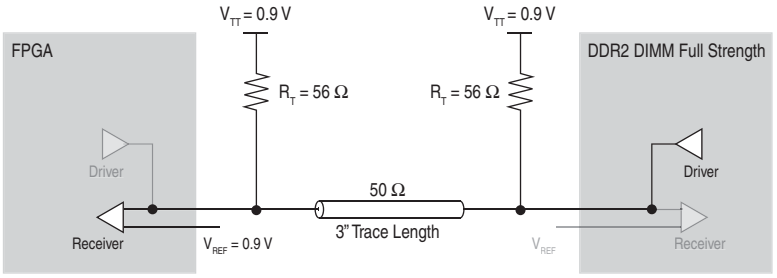


Figure 1–22 and Figure 1–23 show the simulation and measurement, respectively, of the signal at the FPGA side with the full drive strength setting on the DDR2 SDRAM DIMM. The simulation uses a Class II termination scheme with a source-series resistor transmission line. The FPGA is reading from the memory with a full drive strength setting on the DIMM.

**Figure 1–22. HyperLynx Simulation, FPGA Reading from Memory**

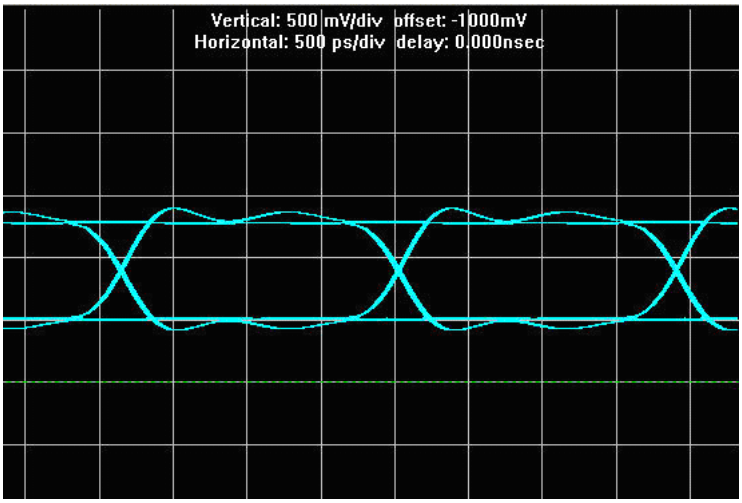



**Figure 1–23. Board Measurement, FPGA Reading from Memory**

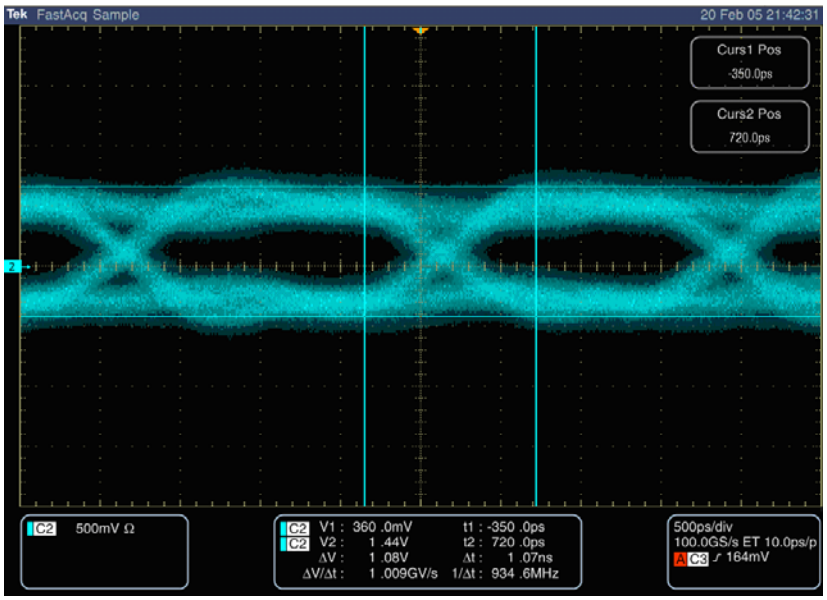


Table 1–8 summarizes the comparison between the simulation and board measurements of the signal seen by the FPGA when the FPGA is reading from memory (driver).

**Table 1–8. Signal Comparison, FPGA is Reading from Memory *(Note 1)*, *(2)***

| | Eye Width (ns) | Eye Height (V) | Overshoot (V) | Undershoot (V) |
|---|---|---|---|---|
| Simulation | 1.73 | 0.76 | N/A | N/A |
| Board Measurement | 1.28 | 0.43 | N/A | N/A |

**Note to Table 1–8:**

(1) The drive strength on the DDR2 SDRAM DIMM is set to full strength.

(2) N/A is not applicable.

Both simulation and measurement show a clean signal and a good eye opening without any over- and undershoot. However, the eye height when the FPGA is reading from the memory is smaller compared to the eye height when the FPGA is writing to the memory. The reduction in eye height is attributed to the voltage drop on the series resistor present on the DIMM. With the drive strength setting on the memory already set to full, you cannot increase the memory drive strength to improve the eye height. One option is to remove the series resistor on the DIMM when the FPGA is reading from memory (refer to the section "Component Versus DIMM" on page 1–36). Another option is to remove the external parallel resistor near the memory so that the memory driver sees less loading. For a DIMM configuration, the latter option is a better choice because the series resistors are part of the DIMM and you can easily turn on the ODT feature to use as the termination resistor when the FPGA is writing to the memory and turn off when the FPGA is reading from memory.

The results for the Class II termination scheme demonstrate that the scheme is ideal for bidirectional signals such as data strobe and data for DDR2 SDRAM memory. Terminations at the receiver eliminate reflections back to the driver and suppress any ringing at the receiver.

## Class I External Parallel Termination

The single parallel (Class I) termination scheme refers to when the termination is located near the receiver side. Typically, this scheme is used for terminating unidirectional signals (such as clocks, address, and command signals) for DDR2 SDRAM.

However, because of board constraints, this form of termination scheme is sometimes used in bidirectional signals, such as data (DQ) and data strobe (DQS) signals. For bidirectional signals, you can place the termination on either the memory or the FPGA side. This section focuses only on the Class I termination scheme with memory-side termination. The memory-side termination ensures impedance matching when the signal reaches the receiver of the memory. However, when the FPGA is reading from the memory, there is no termination on the FPGA side, resulting in impedance mismatch. This section describes the signal quality of this termination scheme.

## FPGA Writing to Memory

When the FPGA is writing to the memory (Figure 1–24), the transmission line is parallel-terminated at the memory side, resulting in minimal reflection on the receiver side because of the matched impedance seen by the transmission line. The benefit of this termination scheme is that only one external resistor is required. Alternatively, you can implement this termination scheme using an ODT resistor instead of an external resistor.

Refer to the section "Class I Termination Using ODT" on page 1–27 for more information about how an ODT resistor compares to an external termination resistor.

**Figure 1–24. Class I Termination Scheme with Memory-Side Series Resistor**

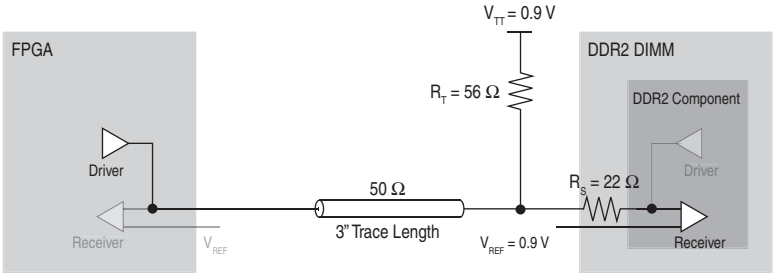



Figure 1–25 shows the simulation and measurement of the signal at the memory (DDR2 SDRAM DIMM) of Class I termination with a memory-side resistor. The FPGA writes to the memory with a 16 mA drive strength setting.

**Figure 1–25. HyperLynx Simulation and Board Measurement, FPGA Writing to Memory**

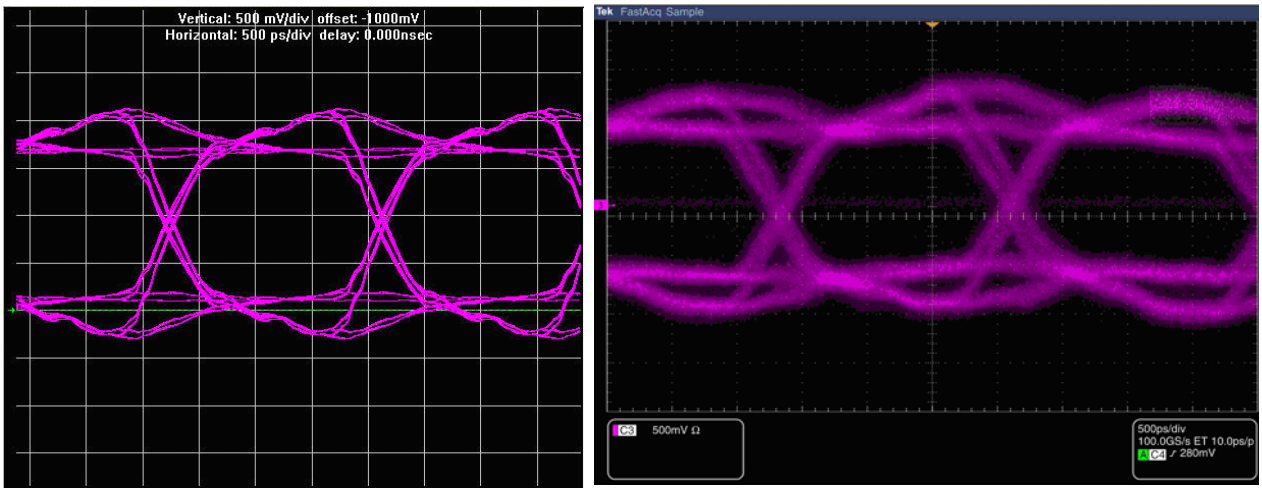


Table 1–9 summarizes the comparison of the signal at the DDR2 SDRAM DIMM of a Class I and Class II termination scheme using external resistors with memory-side series resistors. The FPGA (driver) writes to the memory (receiver).

**Table 1–9. Signal Comparison When the FPGA is Writing to Memory *(Note 1)***

| | Eye Width (ns) | Eye Height (V) | Overshoot (V) | Undershoot (V) |
|---|---|---|---|---|
| **Class I Termination Scheme With External Parallel Resistor** | | | | |
| Simulation | 1.69 | 1.51 | 0.34 | 0.29 |
| Board Measurement | 1.25 | 1.08 | 0.41 | 0.34 |
| **Class II Termination Scheme With External Parallel Resistor** | | | | |
| Simulation | 1.65 | 1.28 | 0.16 | 0.14 |
| Board Measurement | 1.35 | 0.83 | 0.16 | 0.18 |

**Note to Table 1–9:**

(1) The drive strength on the FPGA is set to 16 mA.

Table 1–9 shows the overall signal quality of a Class I termination scheme is comparable to the signal quality of a Class II termination scheme, except that the eye height of the Class I termination scheme is approximately 30% larger. The increase in eye height is due to the reduced loading "seen" by the driver, because the Class I termination scheme does not have an FPGA-side parallel termination resistor. However, increased eye height comes with a price: a 50% increase in the over- and undershoot of the signal using Class I versus Class II termination scheme. You can decrease the FPGA drive strength to compensate for the decreased loading seen by the driver to decrease the over- and undershoot.

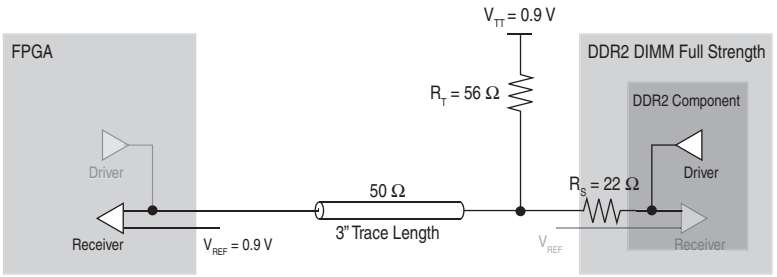
Refer to the section "Drive Strength" on page 1–34 for more information about how drive strength affects the signal quality.

## FPGA Reading from Memory

As described in the section "FPGA Writing to Memory" on page 1–24, in Class I termination, the termination is located near the receiver. However, if you use this termination scheme to terminate a bidirectional signal, the receiver can also be the driver. For example, in DDR2 SDRAM, the data signals are both receiver *and* driver.

Figure 1–26 shows a Class I termination scheme with a memory-side resistor. The FPGA reads from the memory.

**Figure 1–26. Class I Termination Scheme with Memory-Side Series Resistor**



When the FPGA reads from the memory (Figure 1–26), the transmission line is not terminated at the FPGA, resulting in an impedance mismatch, which then results in over- and undershoot. Figure 1–27 shows the simulation and measurement of the signal at the FPGA side (receiver) of a Class I termination. The FPGA reads from the memory with a full drive strength setting on the DDR2 SDRAM DIMM.

**Figure 1–27. HyperLynx Simulation and Board Measurement, FPGA Reading from Memory**

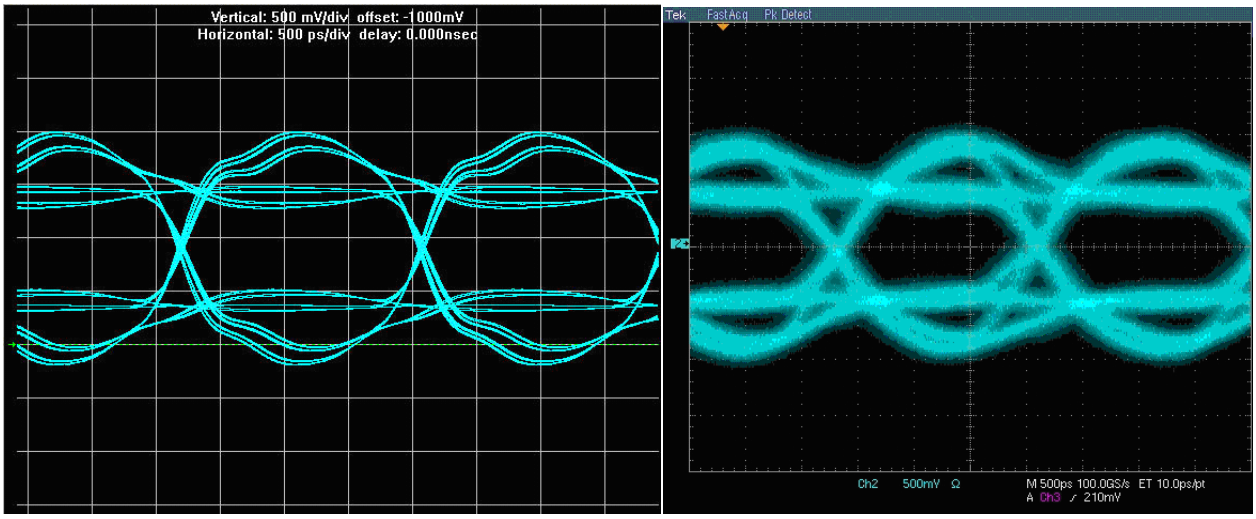



Table 1–10 summarizes the comparison of the signal "seen" at the FPGA of a Class I and Class II termination scheme using an external resistor with a memory-side series resistor. The FPGA (receiver) reads from the memory (driver).

**Table 1–10. Signal Comparison When the FPGA is Reading From Memory *(Note 1)*, *(2)***

| | Eye Width (ns) | Eye Height (V) | Overshoot (V) | Undershoot (V) |
|---|---|---|---|---|
| **Class I Termination Scheme with External Parallel Resistor** | | | | |
| Simulation | 1.73 | 0.74 | 0.20 | 0.18 |
| Board Measurement | 1.24 | 0.58 | 0.09 | 0.14 |
| **Class II Termination Scheme with External Parallel Resistor** | | | | |
| Simulation | 1.73 | 0.76 | N/A | N/A |
| Board Measurement | 1.28 | 0.43 | N/A | N/A |

**Note to Table 1–10:**

(1) The drive strength on the DDR2 SDRAM DIMM is set to full strength.

(2) N/A is not applicable.

When the FPGA reads from the memory using the Class I scheme, the signal quality is comparable to that of the Class II scheme, in terms of the eye height and width. Table 1–10 shows the lack of termination at the receiver (FPGA) results in impedance mismatch, causing reflection and ringing that is not visible in the Class II termination scheme. As such, Altera recommends using the Class I termination scheme for unidirectional signals (such as command and address signals), between the FPGA and the memory.

# Class I Termination Using ODT

Presently, ODT is becoming a common feature in memory, including SDRAMs, graphics DRAMs, and SRAMs. ODT helps reduce board termination cost and simplify board routing. This section describes the ODT feature of DDR2 SDRAM and the signal quality when the ODT feature is used.

## FPGA Writing to Memory

DDR2 SDRAM has built-in ODT that eliminates the need for external termination resistors. To use the ODT feature of the memory, you must configure the memory to turn on the ODT feature during memory initialization. For DDR2 SDRAM, set the ODT feature by programming the extended mode register. In addition to programming the extended mode register during initialization of the DDR2 SDRAM, an ODT input pin on the DDR2 SDRAM must be driven high to activate the ODT.

Refer to the respective memory data sheet for additional information about setting the ODT feature and the timing requirements for driving the ODT pin in DDR2 SDRAM.

The ODT feature in DDR2 SDRAM is controlled dynamically—it is turned on while the FPGA is writing to the memory and turned off while the FPGA is reading from the memory. The ODT feature in DDR2 SDRAM has three settings: 50Ω, 75Ω, and 150Ω. If there are no external parallel termination resistors and the ODT feature is turned on, the termination scheme resembles the Class I termination described in "Class I External Parallel Termination" on page 1–23.

Figure 1–28 shows the termination scheme when the ODT on the DDR2 SDRAM is turned on.

**Figure 1–28. Class I Termination Scheme Using ODT**

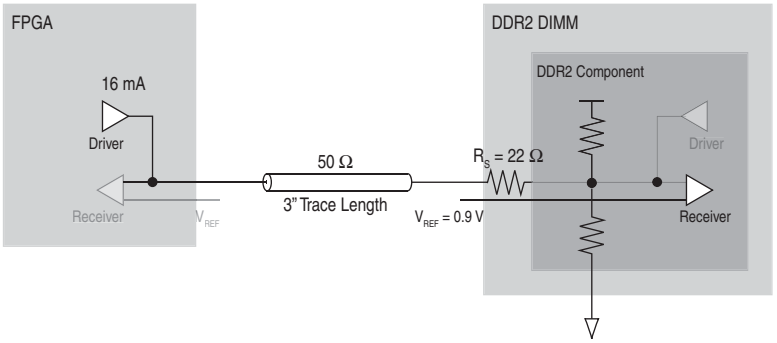


Figure 1–29 shows the simulation and measurement of the signal visible at the memory (receiver) using 50 Ω ODT with a memory-side series resistor transmission line. The FPGA writes to the memory with a 16 mA drive strength setting.

**Figure 1–29. Simulation and Board Measurement, FPGA Writing to Memory**

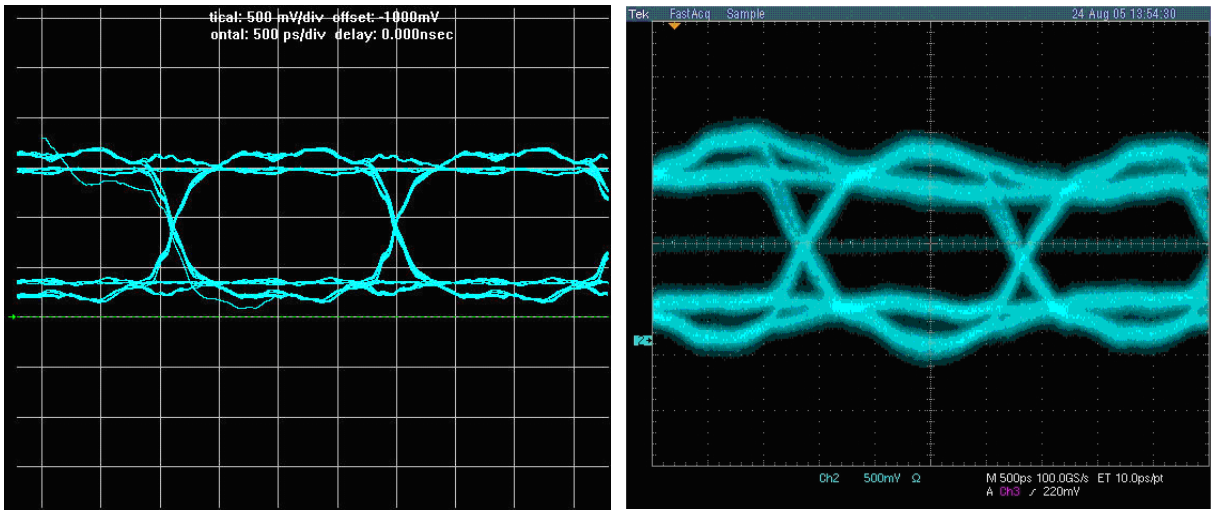



Table 1–11 summarizes the comparisons of the signal seen the DDR2 SDRAM DIMM of a Class I termination scheme using an external resistor and a Class I termination scheme using ODT with a memory-side series resistor. The FPGA (driver) writes to the memory (receiver).

**Table 1–11. Signal Comparison When the FPGA is Writing to Memory *(Note 1)*, *(2)***

| | Eye Width (ns) | Eye Height (V) | Overshoot (V) | Undershoot (V) |
|---|---|---|---|---|
| **Class I Termination Scheme with ODT** | | | | |
| Simulation | 1.63 | 0.84 | N/A | 0.12 |
| Board Measurement | 1.51 | 0.76 | 0.05 | 0.15 |
| **Class I Termination Scheme with External Parallel Resistor** | | | | |
| Simulation | 1.69 | 1.51 | 0.34 | 0.29 |
| Board Measurement | 1.25 | 1.08 | 0.41 | 0.34 |

**Note to Table 1–11:**

(1) The drive strength on the FPGA is set to 16 mA.

(2) N/A is not applicable.

When the ODT feature is enabled in the DDR2 SDRAM, the eye width is improved. There is some degradation to the eye height, but it is not significant. When ODT is enabled, the most significant improvement in signal quality is the reduction of the over- and undershoot, which helps mitigate any potential reliability issues on the memory devices.

Using memory ODT also eliminates the need for external resistors, which reduces board cost and simplifies board routing, allowing you to shrink your boards. Therefore, Altera recommends using the ODT feature on the DDR2 SDRAM memory.

### FPGA Reading from Memory

Altera's Arria GX, Arria II GX, Cyclone series, and Stratix II series of devices are not equipped with ODT. When the DDR2 SDRAM ODT feature is turned off when the FPGA is reading from the memory, the termination scheme resembles the no-parallel termination scheme illustrated by Figure 1–32 on page 1–31.

## No-Parallel Termination

The no-parallel termination scheme is described in the JEDEC standards JESD8-6 for HSTL I/O, JESD8-9b for SSTL-2 I/O, and JESD8-15a for SSTL-18 I/O. Designers who attempt series-only termination schemes such as this often do so to eliminate the need for a $V_{TT}$ power supply.

This is typically not recommended for any signals between an FPGA and DDR2 interface; however, information about this topic is included here as a reference point to clarify the challenges that may occur if you attempt to avoid parallel termination entirely.

### FPGA Writing to Memory

Figure 1–30 shows a no-parallel termination transmission line of the FPGA driving the memory. When the FPGA is driving the transmission line, the signals at the memory-side (DDR2 SDRAM DIMM) may suffer from signal degradation (for example, degradation in rise and fall time). This is due to impedance mismatch, because there is no parallel termination at the memory-side. Also, because of factors such as trace length and drive strength, the degradation seen at the receiver-end might be sufficient to result in a system failure. To understand the effects of each termination scheme on a system, perform system-level simulations before and after the board is designed.

**Figure 1–30. No-Parallel Termination Scheme**

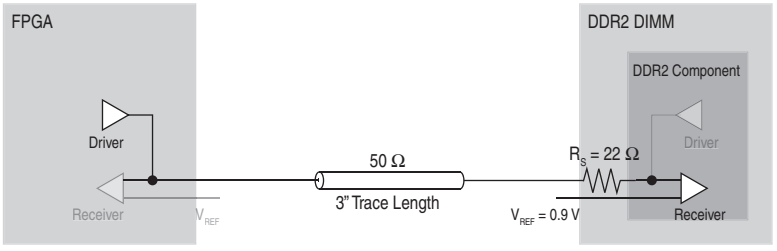
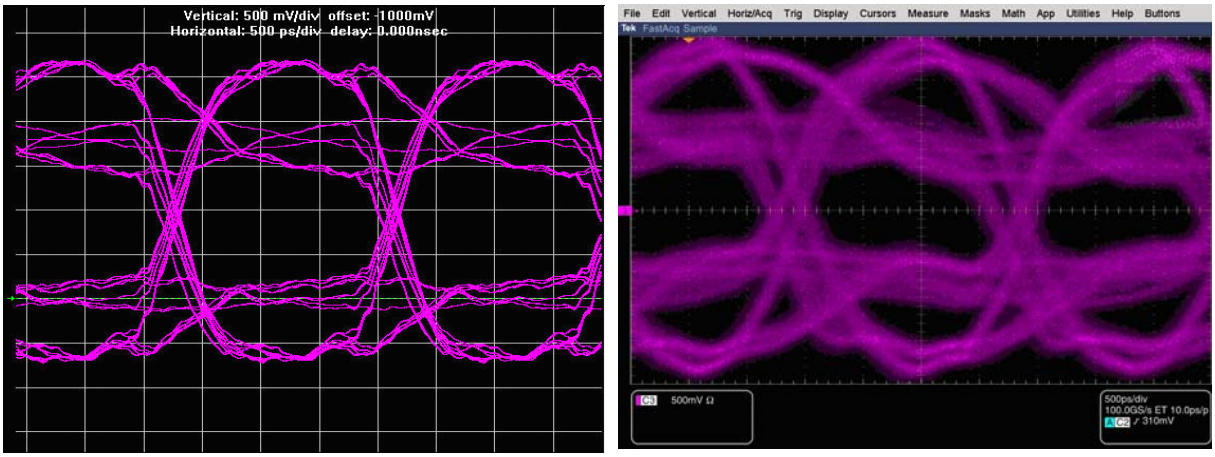


Figure 1–31 shows a HyperLynx simulation and measurement of the FPGA writing to the memory at 533 MHz with a no-parallel termination scheme using a 16 mA drive strength option. The measurement point is on the DDR2 SDRAM DIMM.

**Figure 1–31. HyperLynx Simulation and Board Measurement, FPGA Writing to Memory**




The simulated and measured signal shows that there is sufficient eye opening but also significant over- and undershoot of the 1.8-V signal specified by the DDR2 SDRAM. From the simulation and measurement, the overshoot is approximately 1 V higher than 1.8 V, and undershoot is approximately 0.8 V below ground. This over- and undershoot might result in a reliability issue, because it has exceeded the absolute maximum rating specification listed in the memory vendors' DDR2 SDRAM data sheet.

Table 1–12 summarizes the comparison of the signal visible at the DDR2 SDRAM DIMM of a no-parallel and a Class II termination scheme when the FPGA writes to the DDR2 SDRAM DIMM.

**Table 1–12. Signal Comparison When the FPGA is Writing to Memory** ***(Note 1)***

| | Eye Width (ns) | Eye Height (V) | Overshoot (V) | Undershoot (V) |
|---|---|---|---|---|
| **No-Parallel Termination Scheme** | | | | |
| Simulation | 1.66 | 1.10 | 0.90 | 0.80 |
| Board Measurement | 1.25 | 0.60 | 1.10 | 1.08 |
| **Class II Termination Scheme With External Parallel Resistor** | | | | |
| Simulation | 1.65 | 1.28 | 0.16 | 0.14 |
| Board Measurement | 1.35 | 0.83 | 0.16 | 0.18 |

**Note to Table 1–12:**

(1) The drive strength on the FPGA is set to Class II 16 mA.

Although the appearance of the signal in a no-parallel termination scheme is not clean, when you take the key parameters into consideration, the eye width and height is comparable to that of a Class II termination scheme. The major disadvantage of using a no-parallel termination scheme is the over- and undershoot. There is no termination on the receiver, so there is an impedance mismatch when the signal arrives at the receiver, resulting in ringing and reflection. In addition, the 16-mA drive strength setting on the FPGA also results in overdriving the transmission line, causing the over- and undershoot. By reducing the drive strength setting, the over- and undershoot decreases and improves the signal quality "seen" by the receiver.

For more information about how drive strength affects the signal quality, refer to "Drive Strength" on page 1–34.

## FPGA Reading from Memory

In a no-parallel termination scheme (Figure 1–32), when the memory is driving the transmission line, the resistor, $R_S$ acts as a source termination resistor. The DDR2 SDRAM driver has two drive strength settings:

- Full strength, in which the output impedance is approximately 18Ω
- Reduced strength, in which the output impedance is approximately 40Ω

When the DDR2 SDRAM DIMM drives the transmission line, the combination of the 22-Ω source-series resistor and the driver impedance should match that of the characteristic impedance of the transmission line. As such, there is less over- and undershoot of the signal visible at the receiver (FPGA).

**Figure 1–32. No-Parallel Termination Scheme, FPGA Reading from Memory**

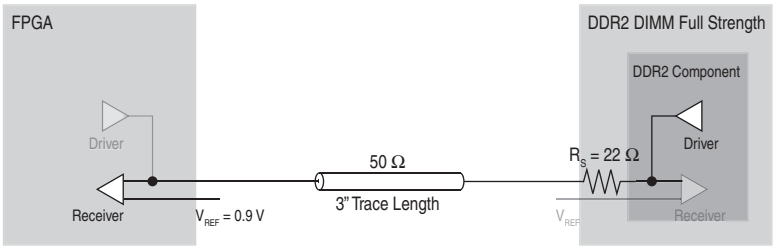


Figure 1–33 shows the simulation and measurement of the signal visible at the FPGA (receiver) when the memory is driving the no-parallel termination transmission line with a memory-side series resistor.

**Figure 1–33. HyperLynx Simulation and Board Measurement, FPGA Reading from Memory**

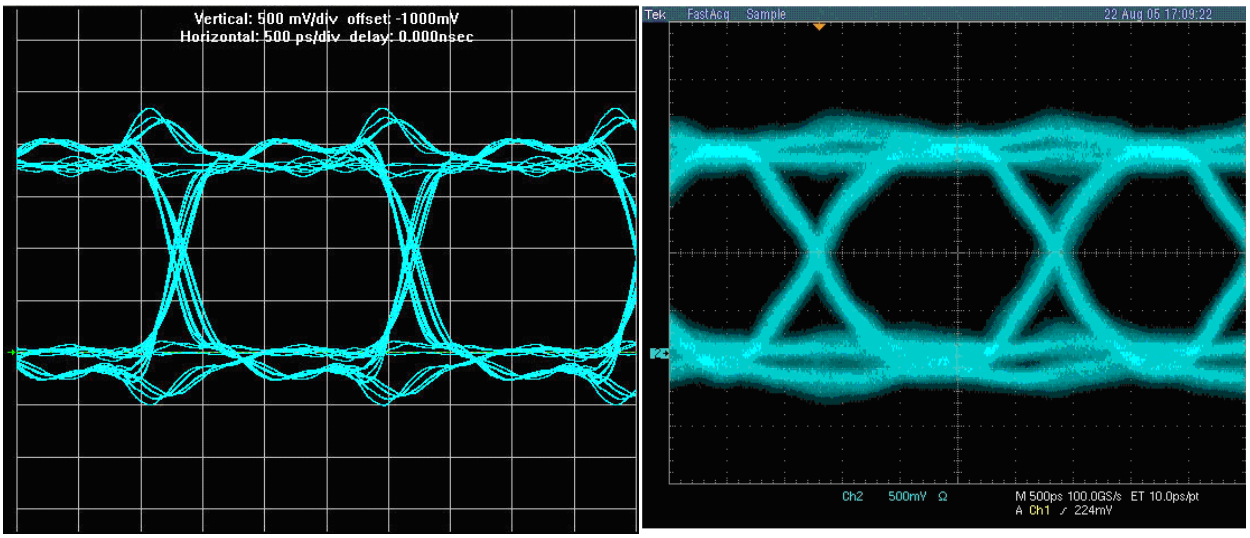



Table 1–13 summarizes the comparison of the signal seen on the FPGA with a no-parallel and a Class II termination scheme when the FPGA is reading from memory.

**Table 1–13. Signal Comparison, FPGA Reading From Memory *(Note 1)*, *(2)***

| | Eye Width (ns) | Eye Height (V) | Overshoot (V) | Undershoot (V) |
|---|---|---|---|---|
| **No-Parallel Termination Scheme** | | | | |
| Simulation | 1.82 | 1.57 | 0.51 | 0.51 |
| Board Measurement | 1.62 | 1.29 | 0.28 | 0.37 |
| **Class II Termination Scheme with External Parallel Resistor** | | | | |
| Simulation | 1.73 | 0.76 | N/A | N/A |
| Board Measurement | 1.28 | 0.43 | N/A | N/A |

**Note to Table 1–13:**

(1) The drive strength on the DDR2 SDRAM DIMM is set to full strength.

(2) N/A is not applicable.

As in the section "FPGA Writing to Memory" on page 1–29, the eye width and height of the signal in a no-parallel termination scheme is comparable to a Class II termination scheme, but the disadvantage is the over- and undershoot. There is over- and undershoot because of the lack of termination on the transmission line, but the magnitude of the over- and undershoot is not as severe when compared to that described in "FPGA Writing to Memory" on page 1–29. This is attributed to the presence of the series resistor at the source (memory side), which dampens any reflection coming back to the driver and further reduces the effect of the reflection on the FPGA side.

When the memory-side series resistor is removed (Figure 1–34), the memory driver impedance no longer matches the transmission line and there is no series resistor at the driver to dampen the reflection coming back from the unterminated FPGA side.

**Figure 1–34. No-Parallel Termination Scheme, FPGA REading from Memory**

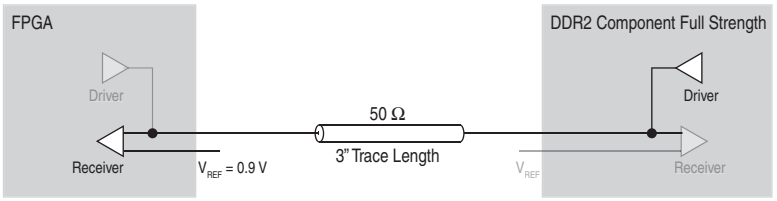



Figure 1–35 shows the simulation and measurement of the signal at the FPGA side in a no-parallel termination scheme with the full drive strength setting on the memory.

**Figure 1–35. HyperLynx Simulation and Measurement, FPGA Reading from Memory**

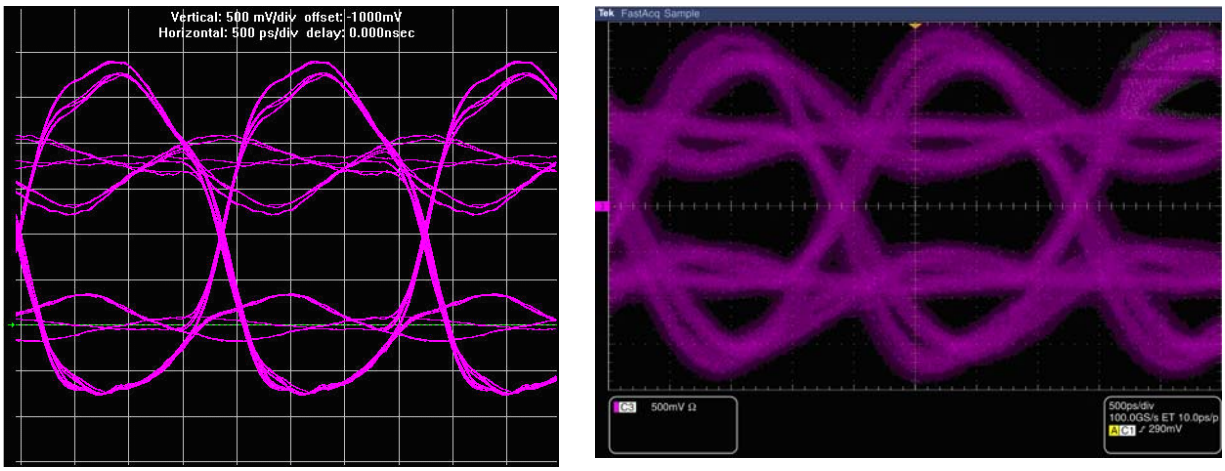



Table 1–14 summarizes the difference between no-parallel termination with and without memory-side series resistor when the memory (driver) writes to the FPGA (receiver).

**Table 1–14. No-Parallel Termination with and without Memory-Side Series Resistor *(Note 1)***

| | Eye Width (ns) | Eye Height (V) | Overshoot (V) | Undershoot (V) |
|---|---|---|---|---|
| **Without Series Resistor** | | | | |
| Simulation | 1.81 | 0.85 | 1.11 | 0.77 |
| Board Measurement | 1.51 | 0.92 | 0.96 | 0.99 |
| **With Series Resistor** | | | | |
| Simulation | 1.82 | 1.57 | 0.51 | 0.51 |
| Board Measurement | 1.62 | 1.29 | 0.28 | 0.37 |

**Note to Table 1–14:**

(1) The drive strength on the memory is set to full drive strength.

Table 1–14 highlights the effect of the series resistor on the memory side with the dramatic increase in over- and undershoot and the decrease in the eye height. This result is similar to that described in "FPGA Writing to Memory" on page 1–29. In that simulation, there is a series resistor but it is located at the receiver side (memory-side), so it does not have the desired effect of reducing the drive strength of the driver and suppressing the reflection coming back from the unterminated receiver-end. As such, in a system without receiver-side termination, the series resistor on the driver helps reduce the drive strength of the driver and dampen the reflection coming back from the unterminated receiver-end.

### Summary

This section compared the various types of termination schemes and studied the benefits and disadvantages of each scheme.

For bidirectional signals, such as the DQ and DQS signals of DDR2 SDRAM, dynamic OCT should be regarded as the ideal termination method when the feature is available; otherwise, a Class II termination scheme with fly-by topology should be regarded as the ideal termination method.

For unidirectional signals, such as the command and address signals of DDR and DDR2 SDRAM, a memory-side Class I termination scheme with fly-by topology provides the best results.

If board real estate and cost is prohibitive to placing on-board termination resistors, you can use the ODT feature on the DDR2 SDRAM memory for the memory-side Class II termination when the controller drives the transmission line. If a Stratix III series device is used, dynamic OCT can achieve a termination solution comparable to a fully discrete terminated system.

## Drive Strength

Altera's FPGA products offer numerous drive strength settings, allowing you to optimize your board designs to achieve the best signal quality. This section focuses on the most commonly used drive strength settings of 8 mA and 16 mA, as recommended by JEDEC for Class I and Class II termination schemes.

You are not restricted to using only these drive strength settings for your board designs. You should perform simulations using I/O models available from Altera and memory vendors to ensure that you use the proper drive strength setting to achieve optimum signal integrity.

## How Strong is Strong Enough?

Figure 1–20 on page 1–20 shows a signal probed at the DDR2 SDRAM DIMM (receiver) of a far-end series-terminated transmission line when the FPGA writes to the DDR2 SDRAM DIMM using a drive strength setting of 16 mA. The resulting signal quality on the receiver shows excessive over- and undershoot. To reduce the over- and undershoot, you can reduce the drive strength setting on the FPGA from 16 mA to 8 mA. Figure 1–36 shows the simulation and measurement of the FPGA with a drive strength setting of 8 mA driving a no-parallel termination transmission line.

**Figure 1–36. HyperLynx Simulation and Measurement, FPGA Writing to Memory**

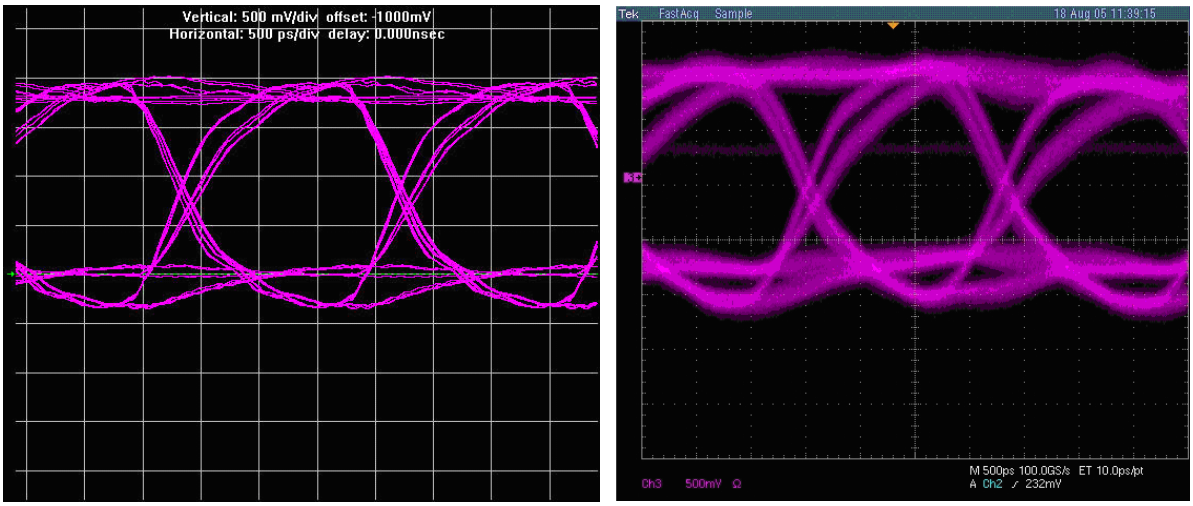



Table 1–15 compares the signals at the DDR2 SDRAM DIMM with no-parallel termination and memory-side series resistors when the FPGA is writing to the memory with 8-mA and 16-mA drive strength settings.

**Table 1–15. Simulation and Board Measurement Results for 8 mA and 16 mA Drive Strength Settings**

| | Eye Width (ns) | Eye Height (V) | Overshoot (V) | Undershoot (V) |
|---|---|---|---|---|
| **8-mA Drive Strength Setting** | | | | |
| Simulation | 1.48 | 1.71 | 0.24 | 0.35 |
| Board Measurement | 1.10 | 1.24 | 0.24 | 0.50 |
| **16-mA Drive Strength Setting** | | | | |
| Simulation | 1.66 | 1.10 | 0.90 | 0.80 |
| Board Measurements | 1.25 | 0.60 | 1.10 | 1.08 |

With a lower strength drive setting, the overall signal quality is improved. The eye width is reduced, but the eye height is significantly larger with a lower drive strength and the over- and undershoot is reduced dramatically.

To improve the signal quality further, you should use 50-Ω on-chip series termination in place of an 8mA drive strength and 25-Ω on-chip series termination in place of a 16 mA drive strength. Refer to "On-Chip Termination (Non-Dynamic)" on page 1–16 for simulation and board measurements.

### Summary

This section compared the effects of drive strength on the signal quality seen at the receiver. As shown, the drive strength setting is highly dependent on the termination scheme, so it is critical that you perform pre- and post-layout board-level simulations to determine the proper drive strength settings. However, page 1–16 through page 1–18 show that 50-Ω OCT and 25-Ω OCT drive strength settings for Class I and Class II, respectively, provide the optimum signal quality because the output driver matches the impedance "seen" by the driver. In addition, using the OCT feature in Altera's FPGA devices eliminates the need for external series resistors and simplifies board design. Finally, using the FPGA's OCT with the SDRAM ODT feature results in the best signal quality without any over- or undershoot.

## System Loading

You can use memory in a variety of forms, such as individual components or multiple DIMMs, resulting in different loading seen by the FPGA. This section describes the effect on signal quality when interfacing memories in component, dual rank, and dual DIMMs format.

### Component Versus DIMM

When using discrete DDR2 SDRAM components, the additional loading from the DDR2 SDRAM DIMM connector is eliminated and the memory-side series resistor on the DDR2 SDRAM DIMM is no longer there. You must decide if the memory-side series resistor near the DDR2 SDRAM is required.

### FPGA Writing to Memory

Figure 1–37 shows the Class II termination scheme without the memory-side series resistor when the FPGA is writing to the memory in the component format.

**Figure 1–37. Class II Termination Scheme without Memory-Side Series Resistor**

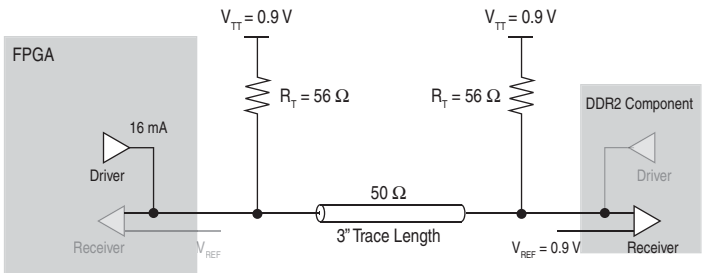


Figure 1–38 shows the simulation and measurement results of the signal seen at a DDR2 SDRAM component of a Class II termination scheme without the DIMM connector and the memory-side series resistor. The FPGA is writing to the memory with a 16-mA drive strength setting.

**Figure 1–38. HyperLynx Simulation and Measurement of the Signal, FPGA Writing to Memory**

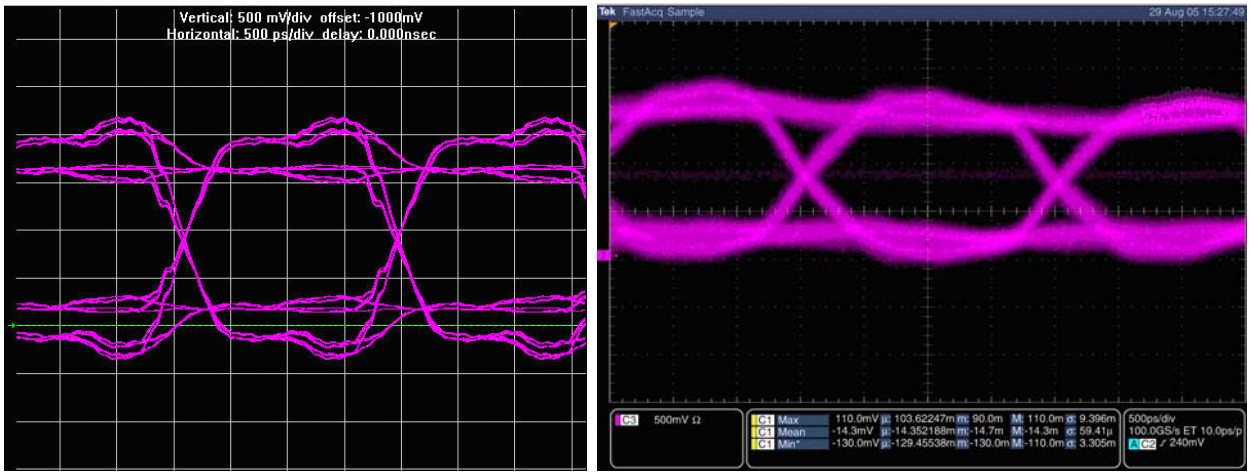



Table 1–16 compares the signal for a single rank DDR2 SDRAM DIMM and a single DDR2 SDRAM component in a Class II termination scheme when the FPGA is writing to the memory.

**Table 1–16. Simulation and Board Measurement Results for Single Rank DDR2 SDRAM DIMM and Single DDR2 SDRAM Component *(Note 1)*, *(2)***

| | Eye Width (ns) | Eye Height (V) | Overshoot (V) | Undershoot (V) | Rising Edge Rate (V/ns) | Falling Edge Rate (V/ns) |
|---|---|---|---|---|---|---|
| **Single DDR2 SDRAM Component** | | | | | | |
| Simulation | 1.79 | 1.15 | 0.39 | 0.33 | 3.90 | 3.43 |
| Measurement | 1.43 | 0.96 | 0.10 | 0.13 | 1.43 | 1.43 |
| **Single Rank DDR2 SDRAM DIMM** | | | | | | |
| Simulation | 1.65 | 0.86 | N/A | N/A | 1.71 | 1.95 |
| Measurement | 1.36 | 0.41 | N/A | N/A | 1.56 | 1.56 |

**Note to Table 1–16:**

(1) The drive strength on the FPGA is set to Class II 16 mA.

(2) N/A is not applicable.

The overall signal quality is comparable between the single rank DDR2 SDRAM DIMM and the single DDR2 SDRAM component, but the elimination of the DIMM connector and memory-side series resistor results in a more than 50% improvement in the eye height.

## FPGA Reading from Memory

Figure 1–39 shows the Class II termination scheme without the memory-side series resistor when the FPGA is reading from memory. Without the memory-side series resistor, the memory driver has less loading to drive the Class II termination. Compare this result to the result of the DDR2 SDRAM DIMM described in "FPGA Reading from Memory" on page 1–31 where the memory-side series resistor is on the DIMM.

**Figure 1–39. Class II Termination Scheme without Memory-Side Series Resistor**

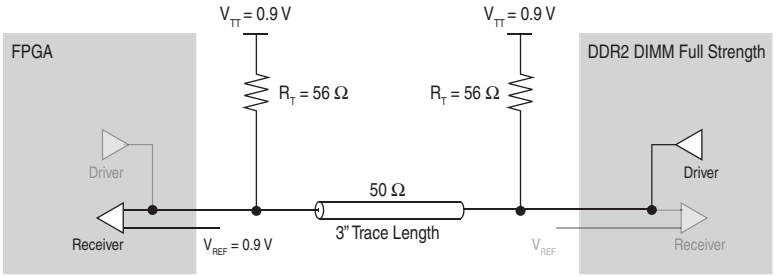



Figure 1–40 shows the simulation and measurement results of the signal seen at the FPGA. The FPGA reads from memory without the source-series resistor near the DDR2 SDRAM component on a Class II-terminated transmission line. The FPGA reads from memory with a full drive strength setting.

**Figure 1–40. HyperLynx Simulation and Measurement, FPGA Reading from the DDR2 SDRAM Component**

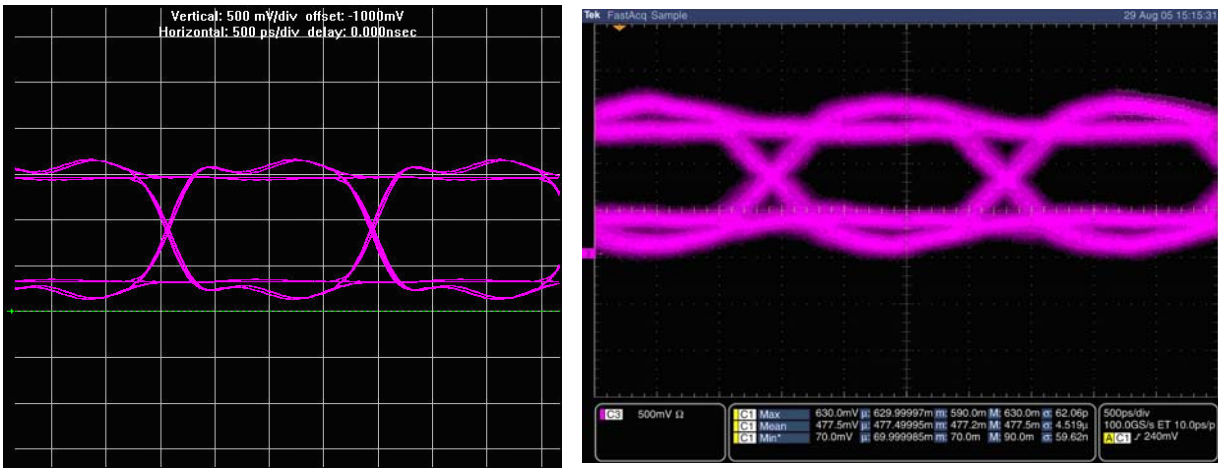


Table 1–17 compares the signal at a single rank DDR2 SDRAM DIMM and a single DDR2 SDRAM component of a Class II termination scheme. The FPGA is reading from memory with a full drive strength setting.

**Table 1–17. Simulation and Board Measurement Results of Single Rank DDR2 SDRAM DIMM and DDR2 SDRAM Component** ***(Note 1)***

| | Eye Width (ns) | Eye Height (V) | Overshoot (V) | Undershoot (V) | Rising Edge Rate (V/ns) | Falling Edge Rate (V/ns) |
|---|---|---|---|---|---|---|
| **Single DDR2 SDRAM Component** | | | | | | |
| Simulation | 1.79 | 1.06 | N/A | N/A | 2.48 | 3.03 |
| Measurement | 1.36 | 0.63 | 0.13 | 0.00 | 1.79 | 1.14 |
| **Single Rank DDR2 SDRAM DIMM** | | | | | | |
| Simulation | 1.73 | 0.76 | N/A | N/A | 1.71 | 1.95 |
| Measurement | 1.28 | 0.43 | N/A | N/A | 0.93 | 0.86 |

**Note to Table 1–17:**

(1) N/A is not applicable.

The effect of eliminating the DIMM connector and memory-side series resistor is evident in the improvement in the eye height.

## Single- Versus Dual-Rank DIMM

DDR2 SDRAM DIMMs are available in either single- or dual-rank DIMM. Single-rank DIMMs are DIMMs with DDR2 SDRAM memory components on one side of the DIMM. Higher-density DIMMs are available as dual-rank, which has DDR2 SDRAM memory components on both sides of the DIMM. With the dual-rank DIMM configuration, the loading is twice that of a single-rank DIMM. Depending on the

board design, you must adjust the drive strength setting on the memory controller to account for this increase in loading. Figure 1–41 shows the simulation result of the signal seen at a dual rank DDR2 SDRAM DIMM. The simulation uses Class II termination with a memory-side series resistor transmission line. The FPGA uses a 16-mA drive strength setting.

**Figure 1–41. HyperLynx Simulation with a 16-mA Drive Strength Setting on the FPGA**

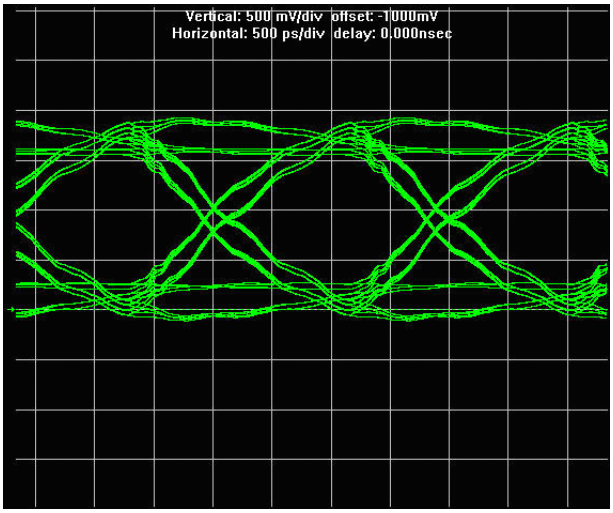



Table 1–18 compares the signals at a single- and dual-rank DDR2 SDRAM DIMM of a Class II and far-end source-series termination when the FPGA is writing to the memory with a 16-mA drive strength setting.

**Table 1–18. Simulation Results of Single- and Dual-Rank DDR2 SDRAM DIMM *(Note 1)***

| | Eye Width (ns) | Eye Height (V) | Overshoot (V) | Undershoot (V) | Rising Edge Rate (V/ns) | Falling Edge Rate (V/ns) |
|---|---|---|---|---|---|---|
| **Dual Rank DDR2 SDRAM DIMM** | | | | | | |
| Simulation | 1.34 | 1.27 | 0.12 | 0.12 | 0.99 | 0.94 |
| **Single Rank DDR2 SDRAM DIMM** | | | | | | |
| Simulation | 1.65 | 1.27 | 0.10 | 0.10 | 1.71 | 1.95 |

**Note to Table 1–18:**

(1) The drive strength on the FPGA is set to Class II 16 mA.

In a dual-rank DDR2 SDRAM DIMM, the additional loading leads to a slower edge rate, which affects the eye width. The slower edge rate leads to the degradation of the setup and hold time required by the memory as well, which must be taken into consideration during the analysis of the timing for the interface. The overall signal quality remains comparable, but eye width is reduced in the dual-rank DIMM. This reduction in eye width leads to a smaller data capture window that must be taken into account when performing timing analysis for the memory interface.

### Single DIMM Versus Multiple DIMMs

Some applications, such as packet buffering, require deeper memory, making a single DIMM interface insufficient. If you use a multiple DIMM configuration to increase memory depth, the memory controller is required to interface with multiple data strobes and the data lines instead of the point-to-point interface in a single DIMM configuration. This results in heavier loading on the interface, which can potentially impact the overall performance of the memory interface.

For detailed information about a multiple DIMM DDR2 SDRAM memory interface, refer to Chapter 3, Dual-DIMM DDR2 and DDR3 SDRAM Interface Termination, Drive Strength, Loading, and Board Layout Guidelines.

### Summary

This section compared the effects of various loading styles on the system. With larger loading, the eye width is reduced, shrinking the data capture window, which must be taken into account when performing the timing analysis for the memory interface.

## DDR2 SDRAM Design Layout Guidelines

Table 1–19 summarizes DDR2 SDRAM layout guidelines.

The following layout guidelines include several +/- length based rules. These length based guidelines for first order timing approximations if you cannot simulate the actual delay characteristic of the interface. They do not include any margin for crosstalk.

Altera recommends that you get accurate time base skew numbers for your design when you simulate the specific implementation.

**Table 1–19. DDR2 SDRAM Layout Guidelines (Part 1 of 3)** ***(Note 1)***

| Parameter | Guidelines |
|---|---|
| DIMMs | If you consider a normal DDR2 unbuffered, unregistered DIMM, essentially you are planning to perform the DIMM routing directly on your PCB. Therefore, each address and control pin routes from the FPGA (single pin) to all memory devices must be on the same side of the FPGA. |
| Impedance | ■ All signal planes must be 50-60-Ω, single-ended, ±10%<br>■ All signal planes must be 100Ω, differential ±10%<br>■ All unused via pads must be removed, because they cause unwanted capacitance |
| Decoupling Parameter | ■ Use 0.1 µF in 0402 size to minimize inductance<br>■ Make $V_{TT}$ voltage decoupling close to pull-up resistors<br>■ Connect decoupling caps between $V_{TT}$ and ground<br>■ Use a 0.1µF cap for every other $V_{TT}$ pin and 0.01µF cap for every VDD and VDDQ pin |

**Table 1–19. DDR2 SDRAM Layout Guidelines (Part 2 of 3)** ***(Note 1)***

| Parameter | Guidelines |
|---|---|
| Power | ■ GND, 2.5 V/1.8 V must be routed as planes<br>■ $V_{CCIO}$ for memories must be routed in a single split plane with at least a 20-mil (0.020 inches, or 0.508 mm) gap of separation<br>■ $V_{TT}$ must be routed as islands or 250-mil (6.35-mm) power traces<br>■ Oscillators and PLL power must be routed as islands or 100-mil (2.54-mm) power traces |
| General Routing | All specified delay matching requirements include PCB trace delays, different layer propogation velocity variance, and crosstalk. To minimize PCB layer propogation variance, Altera recommend that signals from the same net group always be routed on the same layer.<br>■ Use 45° angles (*not* 90° corners)<br>■ Avoid T-Junctions for critical nets or clocks<br>■ Avoid T-junctions greater than 250 mils (6.35 mm)<br>■ Disallow signals across split planes<br>■ Restrict routing other signals close to system reset signals<br>■ Avoid routing memory signals closer than 0.025 inch (0.635 mm) to PCI or system clocks<br>■ All data, address, and command signals must have matched length traces ± 50 ps (±0.250 inches or 6.35 mm)<br>■ All signals within a given **Byte Lane Group** should be matched length with maximum deviation of ±10 ps or approximately ±0.050 inches (1.27 mm) and routed in the same layer. |
| Clock Routing | ■ Clocks should be routed on inner layers with outer-layer run lengths held to under 500 mils (12.7 mm)<br>■ These signals should maintain a10-mil (0.254 mm) spacing from other nets<br>■ Clocks should maintain a length-matching between clock pairs of ±5 ps or approximately ±25 mils (0.635 mm)<br>■ Differential clocks should maintain a length-matching between `P` and `N` signals of ±2 ps or approximately ±10 mils (0.254 mm), routed in parallel<br>■ Space between different pairs should be at least three times the space between the differential pairs and must be routed differentially (5-mil trace, 10-15 mil space on centers) and equal to or up to 100 mils (2.54 mm) longer than signals in the `Address/Command` Group |
| Address and Command Routing | ■ Unbuffered address and command lines are more susceptible to cross-talk and are generally noisier than buffered address or command lines. Therefore, un-buffered address and command signals should be routed on a different layer than data signals (`DQ`) and data mask signals (`DM`) and with greater spacing.<br>■ Do not route differential clock (`CK`) and clock enable (`CKE`) signals close to address signals. |

**Table 1–19. DDR2 SDRAM Layout Guidelines (Part 3 of 3)** ***(Note 1)***

| Parameter | Guidelines |
|---|---|
| External Memory Routing Rules | ■ Keep the distance from the pin on the DDR2 DIMM or component to the termination resistor pack ($V_{TT}$) to less than 500 mils for DQS[x] Data Groups.<br>■ Keep the distance from the pin on the DDR2 DIMM or component to the termination resistor pack ($V_{TT}$) to less than 1000 mils for the ADR_CMD_CTL Address Group.<br>■ Parallelism rules for the DQS[x] Data Groups are as follows:<br>  ■ 4 mils for parallel runs < 0.1 inch (approximately 1× spacing relative to plane distance)<br>  ■ 5 mils for parallel runs < 0.5 inch (approximately 1× spacing relative to plane distance)<br>  ■ 10 mils for parallel runs between 0.5 and 1.0 inches (approximately 2× spacing relative to plane distance)<br>  ■ 15 mils for parallel runs between 1.0 and 6.0 inch (approximately 3× spacing relative to plane distance)<br>■ Parallelism rules for the ADR_CMD_CTL group and CLOCKS group are as follows:<br>  ■ 4 mils for parallel runs < 0.1 inch (approximately 1× spacing relative to plane distance)<br>  ■ 10 mils for parallel runs < 0.5 inch (approximately 2× spacing relative to plane distance)<br>  ■ 15 mils for parallel runs between 0.5 and 1.0 inches (approximately 3× spacing relative to plane distance)<br>  ■ 20 mils for parallel runs between 1.0 and 6.0 inches (approximately 4× spacing relative to plane distance)<br>■ All signals are to maintain a 20-mil separation from other, non-related nets.<br>■ All signals must have a total length of < 6 inches. |
| Termination Rules | ■ When pull-up resistors are used, fly-by termination configuration is recommended. Fly-by helps reduce stub reflection issues.<br>■ Pull-ups should be within 0.5 to no more than 1 inch.<br>■ Pull up is typically 56 Ω<br>■ If using resistor networks:<br>  ■ Do not share R-pack series resistors between address/command and data lines (DQ, DQS, and DM) to eliminate crosstalk within pack.<br>  ■ Series and pull up tolerances are 1–2%.<br>  ■ Series resistors are typically 10 to 20Ω<br>  ■ Address and control series resistor typically at the FPGA end of the link.<br>  ■ DM, DQS, DQ series resistor typically at the memory end of the link (or just before the first DIMM).<br>■ If termination resistor packs are used:<br>  ■ The distance to your memory device should be less than 750 mils.<br>  ■ The distance from your Altera's FPGA device should be less than 1250 mils. |

**Notes to Table 1–19:**

(1) For point-to-point and DIMM interface designs, refer to the Micron website, www.micron.com.

For more information about how the memory manufacturers route these address and control signals on their DIMMs, refer to the Cadence PCB browser from the Cadence website, at www.cadence.com. The various JEDEC example DIMM layouts are available from the JEDEC website, at www.jedec.org.

# Conclusion

This chapter provides Altera's recommendations about the termination schemes to use when interfacing Altera FPGA devices with DDR2 SDRAM devices. However, you must simulate your own design to find the best-suited termination scheme for your design.

This chapter also provides data comparisons from simulation and experimental bench results, so you can draw your own conclusions for optimum design guidelines to achieve the best signal quality.

For termination schemes, Altera recommends that receiver-side parallel termination be used for best signal quality. Therefore, for a bidirectional signal, such as data (DQ) or data strobe (DQS), the recommended termination scheme is Class II termination. You can implement Class II termination by using external parallel resistors at the FPGA and memory side, ODT and OCT at the memory and FPGA side, or a combination of the DDR2 SDRAM ODT and an external parallel resistor at the FPGA side. For a unidirectional signal, such as command or address, the recommended termination scheme is Class I termination, in which the termination is located at the memory side. If you use on-chip termination, refer to Table 1–2 on page 1–8.

Choosing the drive strength setting on the FPGA depends on the termination scheme—Class I or Class II—but the simulation results show that you should set the drive strength to match the loading the output driver sees. For Class II termination, Altera recommends using 25-Ω OCT drive strength settings. They offer the best results because the impedance of the output driver impedance matches the impedance the output driver sees. For a Class I termination scheme, Altera recommends the 50-Ω OCT drive strength setting.

Moreover, this chapter describes the effects that different memory loading styles (such as component versus DIMM) have on signal quality. From the results in "Single- Versus Dual-Rank DIMM" on page 1–39, the higher loading decreases the edge rates of the signal, thus reducing the eye width visible at the receiver. You can increase the edge rates by using a higher drive strength setting on the FPGA, but the output driver impedance decreases because the higher drive strength setting causes impedance mismatch.

Finally, this chapter provides DDR2 SDRAM layout guidelines. Although the recommendations in this chapter are based on the simulations and experimental results of the Stratix III Host Development Board and Stratix II Memory Board 2, you can apply the same general principles when determining the best termination scheme, drive strength setting, and loading style to any board designs. Even armed with this knowledge, it is still critical that you perform simulations, either using IBIS or HSPICE models, to determine the quality of signal integrity on your designs.

# References

This chapter references the following documents and websites:

- Cadence website, www.cadence.com
- JEDEC website, www.jedec.org
- *PC4300 DDR2 SDRAM Unbuffered DIMM Design Specification*
- *PC5300/6400 DDR2 SDRAM Unbuffered DIMM Design Specification*

- *Stratix III Device I/O Features* chapter in the *Stratix III Device Handbook*

# Bibliography

"High-Speed Digital Design—A Handbook of Black Magic," Howard Johnson and Martin Graham, Prentice Hall, 1993.

"Circuits Interconnects, and Packaging for VLSI," H.B. Bakoglu, Addison Wesley, 1990.

"Signal Integrity—Simplified," Eric Bogatin, Prentice Hall Modern Semiconductor Design Series, 2004.

"Handbook of Digital Techniques for High-Speed Design," Tom Granberg, Prentice Hall Modern Semiconductor Design Series, 2004.

"DDR2 Design Guide for Two-DIMM Systems," Micron Technical Note, TN-47-01, 2004.

"Termination Placement in PCB Design: How Much Does it Matter?", Doug Brooks, UltraCAD Design Inc.

"Stratix II Memory Board 2 Rev A User Guide 1.0," Altera's High-Speed/End Applications Team, 2004.

"Stratix II Memory Board 2 Layout Guidelines Rev 0.4," Altera's High-Speed/End Applications Team, 2004.

"Stratix to DDR-I Memory Devices Interface Analysis," Altera's High-Speed/End Applications Team, 2004.

"Multiconductor Transmission Line Analysis for Board-Level Digital Design," Emmanuel A. Maitre, Master Thesis, Santa Clara University, 1994.

JEDEC Standard Publication JESD79C, DDR SDRAM Specification, JEDEC Solid State Technology Association.

JEDEC Standard Publication JESD79-2, DDR2 SDRAM Specification, JEDEC Solid State Technology Association.

JEDEC Standard Publication JESD8-9B, Stub Series Termination Logic for 2.5 V (SSTL-2), JEDEC Solid State Technology Association.

JEDEC Standard Publication JESD8-15A, Stub Series Termination Logic for 1.8 V (SSTL-18), JEDEC Solid State Technology Association.

PC4300 DDR2 SDRAM Unbuffered DIMM Design Specification, Revision 0.5, Oct 30, 2003.

# 2. DDR3 SDRAM Interface Termination, Drive Strength, Loading, and Board Layout Guidelines

This chapter provides guidelines on how to improve the signal integrity of your system and layout guidelines to help you successfully implement a DDR3 SDRAM interface on your system.

DDR3 SDRAM is the third generation of the DDR SDRAM family, and offers improved power, higher data bandwidth, and enhanced signal quality with multiple on-die termination (ODT) selection and output driver impedance control while maintaining partial backward compatibility with the existing DDR2 SDRAM standard.

This chapter describes the following updated and new features in DDR3 SDRAM:

- ODT values selection
- Output driver impedance selection
- ZQ calibration
- Dynamic ODT usage

To take advantage of these new features offered by DDR3 SDRAM, Altera FPGAs—Stratix III, Stratix IV, and Stratix V—have special features to ease and expedite your implementation of DDR3 SDRAM interfaces.

The DDR3 SDRAM information in this chapter applies to both ALTMEMPHY- or UniPHY-based memory controllers unless stated otherwise.

## With Leveling or Without Leveling

Altera offers the DDR3 SDRAM PHY with or without leveling.

You can only implement non leveling DDR3 interface with the ALTMEMPHY IP. Altera strongly recommends using leveling for better signal integrity.

### With Leveling

DDR3 SDRAM DIMMs, as specified by JEDEC, always use a fly-by topology for the address, command, and clock signals. This standard DDR3 SDRAM topology requires the use of Altera DDR3 SDRAM Controller with UniPHY or ALTMEMPHY with read and write leveling.

Altera recommends that for full DDR3 SDRAM compatibility when using discrete DDR3 SDRAM components, you should mimic the JEDEC DDR3 UDIMM fly-by topology on your custom printed circuit boards (PCB).

Arria II GX devices do not support DDR3 SDRAM with read or write leveling, so standard DDR3 SDRAM DIMMs or DDR3 SDRAM components using the standard DDR3 SDRAM fly-by address, command, and clock layout topology are not supported. Refer to "Termination for DDR3 SDRAM Components (Without Leveling)" on page 2–29, for more information on how to use DDR3 SDRAM components with Arria II GX devices.

Use of the standard JEDEC DDR3 fly-by topology with leveling offers the following advantages:

- Easier layout
- Lower SSN for the memory
- Higher data rates

Refer to "Read and Write Leveling" on page 2–3 for more detailed information on read and write leveling.

### Without Leveling

Altera also supports DDR3 SDRAM components without leveling, using a nonstandard, synchronous DDR2-like balanced address, command, and clock layout topology. DDR3 SDRAM interfaces without leveling operate at lower maximum data rates compared to the standard fly-by topology. DDR3 SDRAM interfaces without leveling may be desirable for the following reasons:

- Arria II GX devices do not support read and write leveling, so DDR3 SDRAM DIMMs or topology is not supported, but the I/O electrical standard is supported.
- DDR3 SDRAM PHY IPs without leveling typically have a slightly lower PHY latency when compared to the DDR3 SDRAM PHY with leveling.
- A DDR3 SDRAM PHY IP without leveling typically requires less FPGA resources than an equivalent DDR3 SDRAM PHY IP with leveling.
- You may only require DDR2-like interface performance but want to use the lower power, potential cost, and availability benefits of DDR3 SDRAM components.

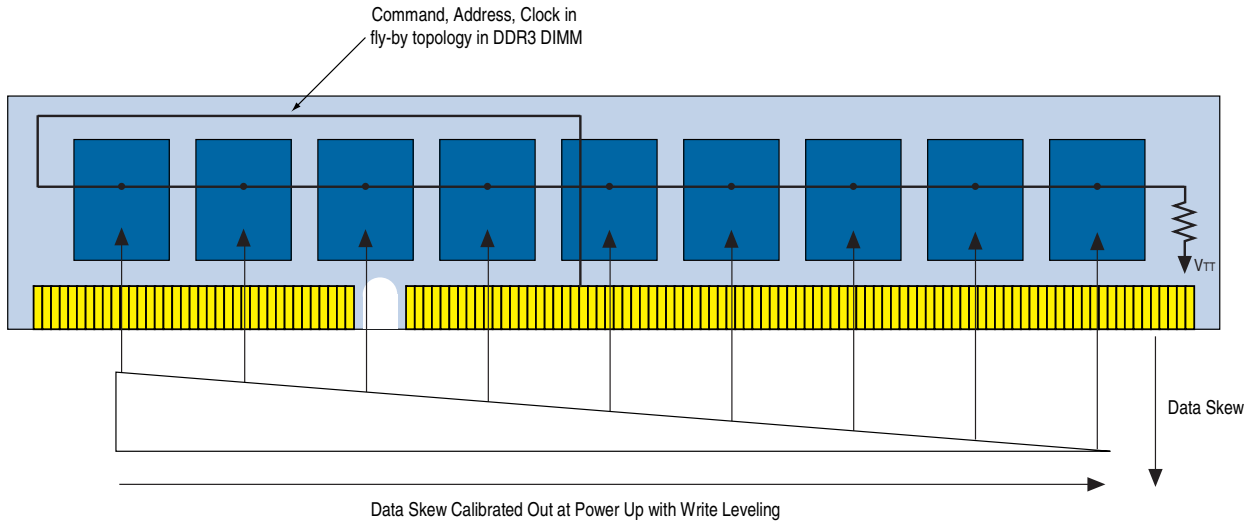
## Comparing DDR3 and DDR2

The following sections review the differences between DDR2 and DDR3 SDRAM and the changes in the features that were made to DDR3 SDRAM. Understanding these differences makes the design process for your DDR3 SDRAM interface easier.

## Read and Write Leveling

One major difference between DDR2 and DDR3 SDRAM is the use of leveling. To improve signal integrity and support higher frequency operations, the JEDEC committee defined a fly-by termination scheme used with clocks, and command and address bus signals. Fly-by topology reduces simultaneous switching noise (SSN) by deliberately causing flight-time skew between the data and strobes at every DRAM as the clock, address, and command signals traverse the DIMM (Figure 2–1).

**Figure 2–1. DDR3 DIMM Fly-By Topology Requiring Write Leveling**



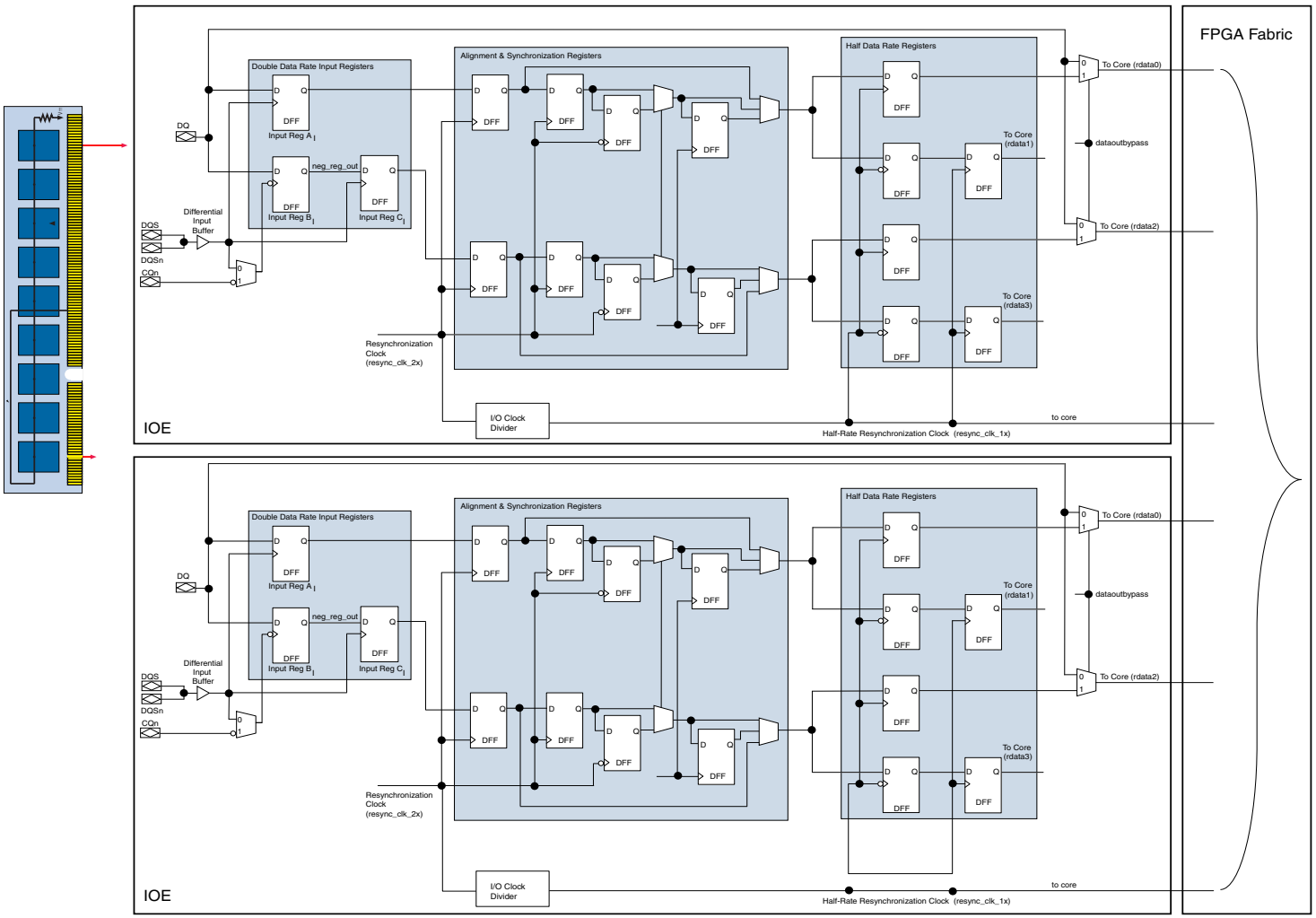

The flight-time skew caused by the fly-by topology led the JEDEC committee to introduce the write leveling feature on the DDR3 SDRAMs; thus requiring controllers to compensate for this skew by adjusting the timing per byte lane.

During a write, DQS groups launch at separate times to coincide with a clock arriving at components on the DIMM, and must meet the timing parameter between the memory clock and DQS defined as $t_{DQSS}$ of ± 0.25 $t_{CK}$.

During the read operation, the memory controller must compensate for the delays introduced by the fly-by topology. The Stratix III, Stratix IV, and Stratix V FPGAs have alignment and synchronization registers built in the I/O element (IOE) to properly capture the data.

Figure 2–2 shows two DQS groups returning from the DIMM for the same read command.

**Figure 2–2. DDR3 DIMM Fly-By Topology Requiring Read Leveling**



For information about the IOE block in Stratix III devices, refer to the *External Memory Interfaces in Stratix III Devices* chapter in volume 1 of the *Stratix III Device Handbook*.

For information about the IOE block in Stratix IV devices, refer to the *External Memory Interfaces in Stratix IV Devices* chapter in volume 1 of the *Stratix IV Device Handbook*.

For information about the IOE block in Stratix V devices, refer to the *External Memory Interfaces in Stratix V Devices* chapter in volume 1 of the *Stratix V Device Handbook*.

## Calibrated Output Impedance and ODT

In DDR2 SDRAM, there are only two drive strength settings, full or reduced, which correspond to the output impedance of 18 Ω and 40 Ω, respectively. These output drive strength settings are static settings and are not calibrated; as a result, the output impedance varies as the voltage and temperature drifts. The DDR3 SDRAM uses a programmable impedance output buffer. Currently, there are two drive strength settings, 34 Ω and 40 Ω. The 40-Ω drive strength setting is currently a reserved specification defined by JEDEC, but available on the DDR3 SDRAM, as offered by some memory vendors. Refer to the datasheet of the respective memory vendors for more information about the output impedance setting. You select the drive strength settings by programming the memory mode register defined by mode register 1 (MR1). To calibrate output driver impedance, an external precision resistor, RZQ, connects the ZQ pin and VSSQ. The value of this resistor must be 240 Ω ± 1%. If you are using a DDR3 SDRAM DIMM, RZQ is soldered on the DIMM so you do not need to layout your board to account for it. Output impedance is set during initialization. To calibrate output driver impedance after power-up, the DDR3 SDRAM needs a calibration command that is part of the initialization and reset procedure and is updated periodically when the controller issues a calibration command.

In addition to calibrated output impedance, the DDR3 SDRAM also supports calibrated parallel ODT through the same external precision resistor, RZQ, which is possible by using a merged output driver structure in the DDR3 SDRAM, which also helps to improve pin capacitance in the DQ and DQS pins. The ODT values supported in DDR3 SDRAM are 20 Ω, 30 Ω, 40 Ω, 60 Ω, and 120 Ω, assuming that RZQ is 240 Ω.

In DDR3 SDRAM, there are two commands related to the calibration of the output driver impedance and ODT. The controller often uses the first calibration command, ZQ CALIBRATION LONG (ZQCL), at initial power-up or when the DDR3 SDRAM is in a reset condition. This command calibrates the output driver impedance and ODT to the initial temperature and voltage condition, and compensates for any process variation due to manufacturing. If the controller issues the ZQCL command at initialization or reset, it takes 512 memory clock cycles to complete; otherwise, it requires 256 memory clock cycles to complete. The controller uses the second calibration command, ZQ CALIBRATION SHORT (ZQCS) during regular operation to track any variation in temperature or voltage. The ZQCS command takes 64 memory clock cycles to complete. Use the ZQCL command any time there is more impedance error than can be corrected with a ZQCS command.

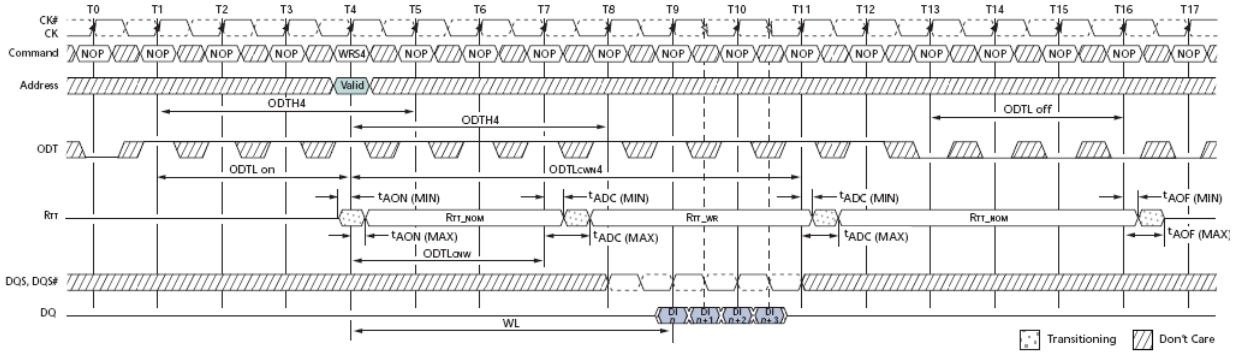
For more information about using ZQ Calibration in DDR3 SDRAM, refer to the application note by Micron, *TN-41-02 DDR3 ZQ Calibration*.

## Dynamic ODT

Dynamic ODT is a new feature in DDR3 SDRAM, and not available in DDR2 SDRAM. Dynamic ODT can change the ODT setting without issuing a mode register set (MRS) command. When you enable dynamic ODT, and there is no write operation, the DDR3 SDRAM terminates to a termination setting of RTT_NORM; when there is a write operation, the DDR3 SDRAM terminates to a setting of RTT_WR. You can preset the values of RTT_NORM and RTT_WR by programming the mode registers, MR1 and MR2.

Figure 2–3 shows the behavior of ODT when you enable dynamic ODT.

**Figure 2–3. Dynamic ODT: Behavior with ODT Asserted Before and After the Write** ***(Note 1)***




**Note to Figure 2–3:**

(1) Source: *TN-41-04 DDR3 Dynamic On-Die Termination*, Micron.

In the two-DIMM DDR3 SDRAM configuration, dynamic ODT helps reduce the jitter at the module being accessed, and minimizes reflections from any secondary modules.

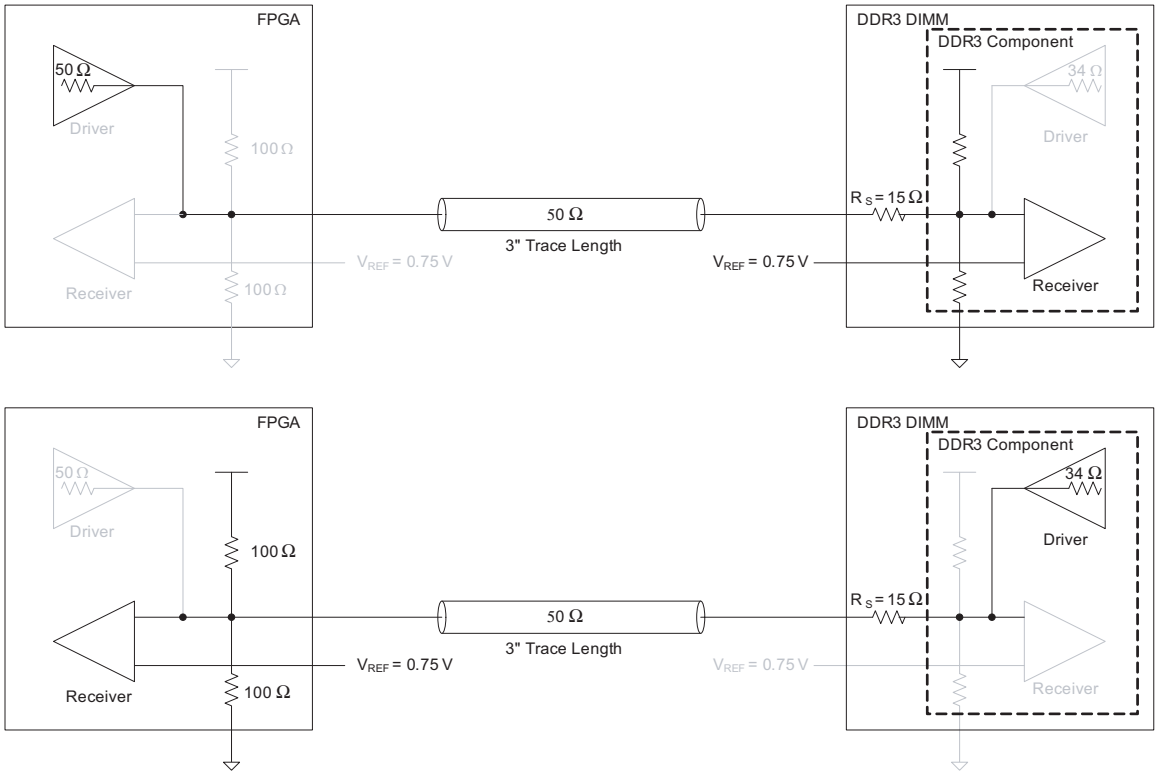
For more information about using the dynamic ODT on DDR3 SDRAM, refer to the application note by Micron, *TN-41-04 DDR3 Dynamic On-Die Termination*.

# Dynamic OCT in Stratix III and Stratix IV Devices

Stratix III and Stratix IV devices support on-off dynamic series and parallel termination for a bi-directional I/O in all I/O banks. Dynamic OCT is a new feature in Stratix III and Stratix IV FPGA devices. You enable dynamic parallel termination only when the bidirectional I/O acts as a receiver and disable it when the bidirectional I/O acts as a driver. Similarly, you enable dynamic series termination only when the bidirectional I/O acts as a driver and is disable it when the bidirectional I/O acts as a receiver. The default setting for dynamic OCT is series termination, to save power when the interface is idle—no active reads or writes.

Additionally, the dynamic control operation of the OCT is separate to the output enable signal for the buffer. Hence, UniPHY IP can only enable parallel OCT during read cycles, saving power when the interface is idle.

**Figure 2–4. Dynamic OCT Between Stratix III and Stratix IV FPGA Devices**




This feature is useful for terminating any high-performance bidirectional path because signal integrity is optimized depending on the direction of the data. In addition, dynamic OCT also eliminates the need for external termination resistors when used with memory devices that support ODT (such as DDR3 SDRAM), thus reducing cost and easing board layout.

However, dynamic OCT in Stratix III and Stratix IV FPGA devices is different from dynamic ODT in DDR3 SDRAM mentioned in previous sections and these features should not be assumed to be identical.

For detailed information about the dynamic OCT feature in the Stratix III FPGA, refer to the *Stratix III Device I/O Features* chapter in volume 1 of the *Stratix III Device Handbook*.

For detailed information about the dynamic OCT feature in the Stratix IV FPGA, refer to the *I/O Features in Stratix IV Devices* chapter in volume 1 of the *Stratix IV Device Handbook*.

# Dynamic OCT in Stratix V Devices

Stratix V devices also support dynamic OCT feature and provide more flexibility. Stratix V OCT calibration uses one RZQ pin that exists in every OCT block. You can use any one of the following as a reference resistor on the RZQ pin to implement different OCT values:

- 240-Ω reference resistor—to implement $R_S$ OCT of 34 Ω, 40 Ω, 48 Ω, 60 Ω, and 80 Ω; and $R_T$ OCT resistance of 20 Ω, 30 Ω, 40 Ω, and 120 Ω.
- 100 Ω reference resistor—to implement $R_S$ OCT of 25 Ω and 50 Ω; and $R_T$ OCT resistance of 50 Ω.

For detailed information about the dynamic OCT feature in the Stratix V FPGA, refer to the *I/O Features in Stratix V Devices* chapter in volume 1 of the *Stratix V Device Handbook*.

# Termination for DDR3 SDRAM Unbuffered and Registered DIMMs

The following sections describe the correct way to terminate a DDR3 SDRAM interface together with Stratix III, Stratix IV, and Stratix V FPGA devices.

DDR3 DIMMs have terminations on all unidirectional signals, such as memory clocks, and addresses and commands; thus eliminating the need for them on the FPGA PCB. In addition, using the ODT feature on the DDR3 SDRAM and the dynamic OCT feature of Stratix III, Stratix IV, and Stratix V FPGA devices completely eliminates any external termination resistors; thus simplifying the layout for the DDR3 SDRAM interface when compared to that of the DDR2 SDRAM interface.

## DDR3 SDRAM Unbuffered DIMM

The most common implementation of the DDR3 SDRAM interface is the unbuffered DIMM (UDIMM). You can find DDR3 SDRAM UDIMMs in many applications, especially in PC applications. You can implement a DDR3 SDRAM UDIMM interface in several permutations, such as single DIMM or multiple DIMMs, using either single-ranked or dual-ranked UDIMMs. In addition to the UDIMM's form factor, these termination recommendations are also valid for small-outline (SO) DIMMs and MicroDIMMs.

Table 2–1 outlines the different permutations of a two-slot DDR3 SDRAM interface and the recommended ODT settings on both the memory and controller when writing to memory.

**Table 2–1. DDR3 SDRAM ODT Matrix for Writes *(Note 1)* and *(2)* (Part 1 of 2)**

| Slot 1 | Slot 2 | Write To | Controller OCT *(3)* | Slot 1 | | Slot 2 | |
|---|---|---|---|---|---|---|---|
| | | | | Rank 1 | Rank 2 | Rank 1 | Rank 2 |
| DR | DR | Slot 1 | Series 50 Ω | 120 Ω *(4)* | ODT off | ODT off | 40 Ω *(4)* |
| | | Slot 2 | Series 50 Ω | ODT off | 40 Ω *(4)* | 120 Ω *(4)* | ODT off |
| SR | SR | Slot 1 | Series 50 Ω | 120 Ω *(4)* | Unpopulated | 40 Ω *(4)* | Unpopulated |
| | | Slot 2 | Series 50 Ω | 40 Ω *(4)* | Unpopulated | 120 Ω *(4)* | Unpopulated |
| DR | Empty | Slot 1 | Series 50 Ω | 120 Ω | ODT off | Unpopulated | Unpopulated |

**Table 2–1. DDR3 SDRAM ODT Matrix for Writes *(Note 1)* and *(2)* (Part 2 of 2)**

| Slot 1 | Slot 2 | Write To | Controller OCT *(3)* | Slot 1 | | Slot 2 | |
|---|---|---|---|---|---|---|---|
| | | | | Rank 1 | Rank 2 | Rank 1 | Rank 2 |
| Empty | DR | Slot 2 | Series 50 Ω | Unpopulated | Unpopulated | 120 Ω | ODT off |
| SR | Empty | Slot 1 | Series 50 Ω | 120 Ω | Unpopulated | Unpopulated | Unpopulated |
| Empty | SR | Slot 2 | Series 50 Ω | Unpopulated | Unpopulated | 120 Ω | Unpopulated |

**Notes to Table 2–1:**

(1) SR: single-ranked DIMM; DR: dual-ranked DIMM.

(2) These recommendations are taken from the *DDR3 ODT and Dynamic ODT* session of the JEDEC DDR3 2007 Conference, Oct 3-4, San Jose, CA.

(3) The controller in this case is the FPGA.

(4) Dynamic ODT is required. For example, the ODT of Slot 2 is set to the lower ODT value of 40 Ω when the memory controller is writing to Slot 1, resulting in termination and thus minimizing any reflection from Slot 2. Without dynamic ODT, Slot 2 will not be terminated.

Table 2–2 outlines the different permutations of a two-slot DDR3 SDRAM interface and the recommended ODT settings on both the memory and controller when reading from memory.

**Table 2–2. DDR3 SDRAM ODT Matrix for Reads *(Note 1)* and *(2)***

| Slot 1 | Slot 2 | Read From | Controller OCT *(3)* | Slot 1 | | Slot 2 | |
|---|---|---|---|---|---|---|---|
| | | | | Rank 1 | Rank 2 | Rank 1 | Rank 2 |
| DR | DR | Slot 1 | Parallel 50 Ω | ODT off | ODT off | ODT off | 40 Ω |
| | | Slot 2 | Parallel 50 Ω | ODT off | 40 Ω | ODT off | ODT off |
| SR | SR | Slot 1 | Parallel 50 Ω | ODT off | Unpopulated | 40 Ω | Unpopulated |
| | | Slot 2 | Parallel 50 Ω | 40 Ω | Unpopulated | ODT off | Unpopulated |
| DR | Empty | Slot 1 | Parallel 50 Ω | ODT off | ODT off | Unpopulated | Unpopulated |
| Empty | DR | Slot 2 | Parallel 50 Ω | Unpopulated | Unpopulated | ODT off | ODT off |
| SR | Empty | Slot 1 | Parallel 50 Ω | ODT off | Unpopulated | Unpopulated | Unpopulated |
| Empty | SR | Slot 2 | Parallel 50 Ω | Unpopulated | Unpopulated | ODT off | Unpopulated |

**Notes to Table 2–2:**

(1) SR: single-ranked DIMM; DR: dual-ranked DIMM.

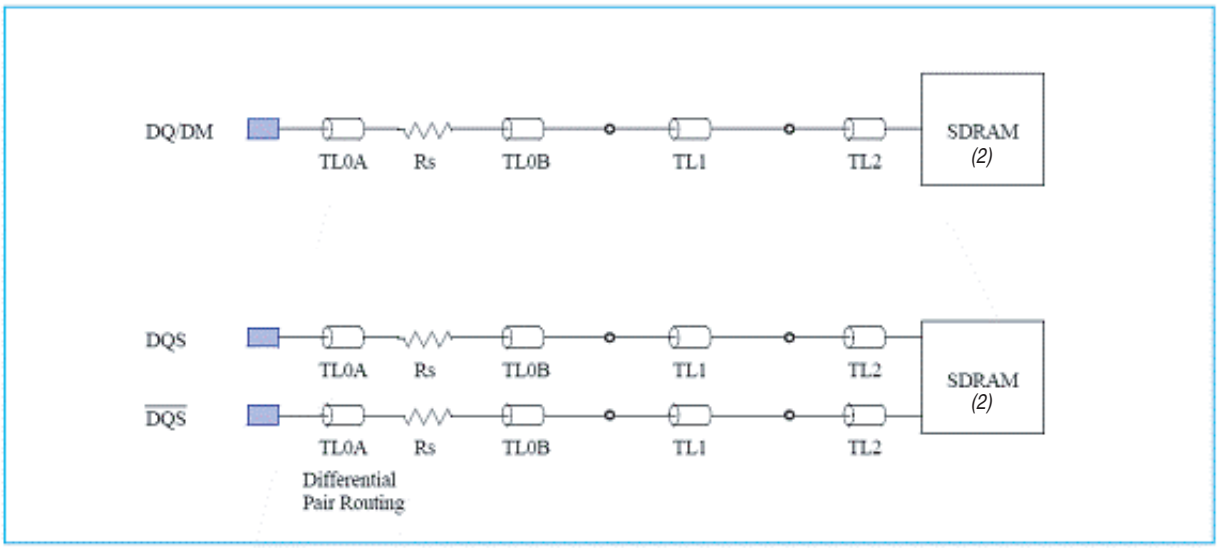
(2) These recommendations are taken from the *DDR3 ODT and Dynamic ODT* session of the JEDEC DDR3 2007 Conference, Oct 3-4, San Jose, CA.

(3) The controller in this case is the FPGA. JEDEC typically recommends 60 Ω, but this value assumes that the typical motherboard trace impedance is 60 Ω and that teh controller supports this termination. Altera recommends using a 50-Ω parallel OCT when reading from the memory.

## DQS, DQ, and DM for DDR3 SDRAM UDIMM

On a single-ranked DIMM, DQS, and DQ signals are point-to-point signals. Figure 2–5 shows the net structure for differential DQS and DQ signals. There is an external 15-Ω stub resistor, $R_S$, on each of the DQS and DQ signals soldered on the DIMM, which helps improve signal quality by dampening reflections from unused slots in a multi-DIMM configuration.

**Figure 2–5. DQ and DQS Net Structure for 64-Bit DDR3 SDRAM UDIMM** ***(Note 1)***



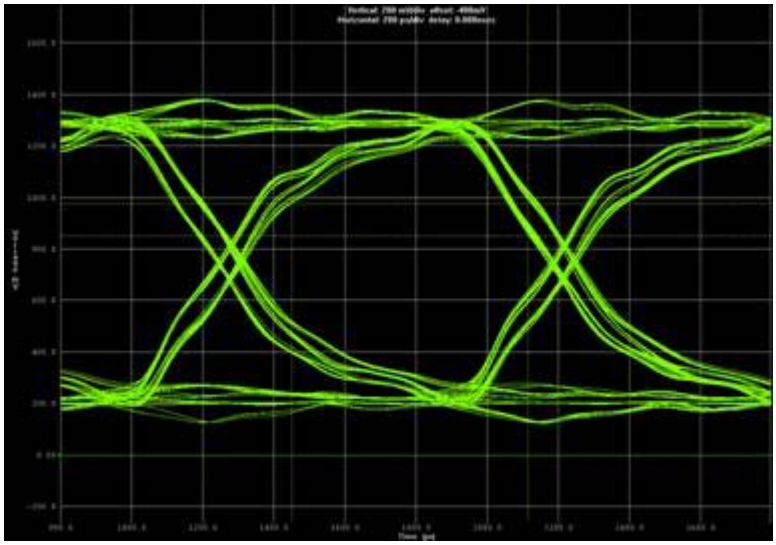

**Notes to Figure 2–5:**

(1) Source: *PC3-6400/PC3-8500/PC3-10600/PC3-12800 DDR3 SDRAM Unbuffered DIMM Design Specification*, July 2007, JEDEC Solid State Technology Association. For clarity of the signal connections in the illustration, the same SDRAM is drawn as two separate SDRAMs.

As mentioned in "Dynamic ODT" on page 2–6, DDR3 SDRAM supports calibrated ODT with different ODT value settings. If you do not enable dynamic ODT, there are three possible ODT settings available for `RTT_NORM`: 40 Ω, 60 Ω, and 120 Ω. If you enable dynamic ODT, the number of possible ODT settings available for `RTT_NORM` increases from three to five with the addition of 20 Ω and 30 Ω. Table 2–1 shows that the recommended ODT setting on the DDR3 SDRAM is 120 Ω. Trace impedance on the DIMM is 60 Ω, and over-terminating the DDR3 SDRAM components on the DIMM with 120 Ω compensates for trace impedance variation on the DIMM due to manufacturing.

Figure 2–6 shows the write-eye diagram at the DQ0 of a DDR3 SDRAM DIMM using the 120-Ω ODT setting, driven by a Stratix III or Stratix IV FPGA using a calibrated series 50-Ω OCT setting.

**Figure 2–6. Simulated Write-Eye Diagram of a DDR3 SDRAM DIMM Using a 120- Ω ODT Setting**



When over-terminating the receiver, the mismatch between load impedance and trace impedance causes ringing at the receiver (Figure 2–6). When you set the DDR3 SDRAM ODT setting to 60 Ω, you observe less ringing at the receiver (Figure 2–7).

**Figure 2–7. Simulated Write-Eye Diagram of a DDR3 SDRAM DIMM Using a 60-Ω ODT Setting**

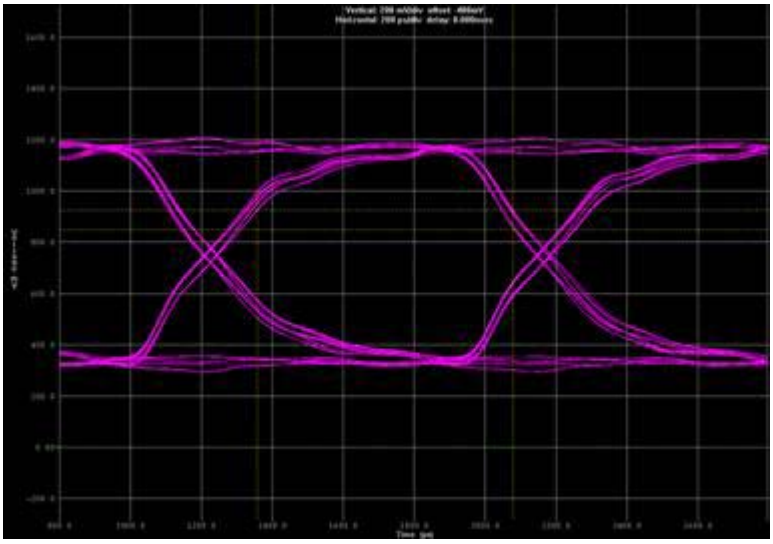


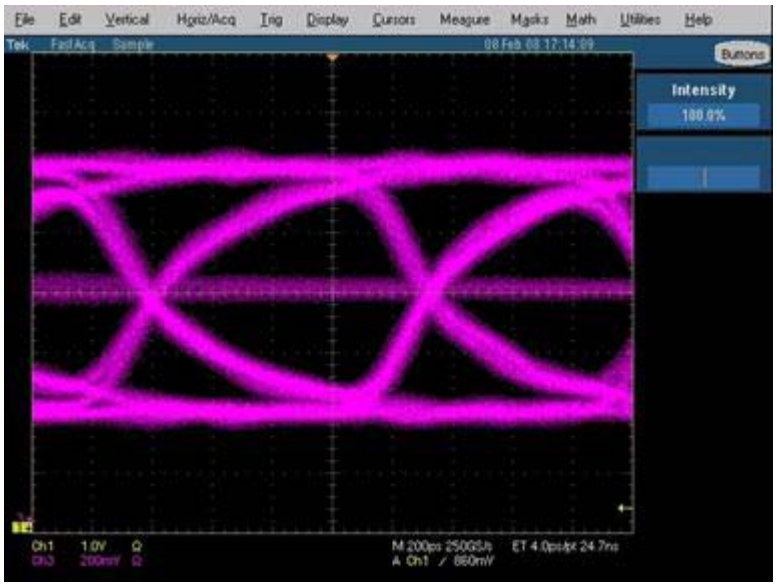
Table 2–3 compares the effects of the ODT setting on the eye diagram at the DDR3 SDRAM (receiver) when the Stratix III or Stratix IV FPGA is writing to memory.

**Table 2–3. Write-Eye Diagram Using Different ODT Setting**

| ODT | Eye Height (V) | Eye Width (ps) | Overshoot (V) | Undershoot (V) |
|---|---|---|---|---|
| 120-Ω ODT | 0.84 | 713 | — | — |
| 60-Ω ODT | 0.73 | 715 | — | — |

Although both 120-Ω and 60-Ω ODT settings result in excellent signal quality and acceptable eye opening, using 120 Ω results in a larger eye height because of under-termination, it has a minimal effect on eye width. Because the use of 60-Ω ODT results in less ringing, the 60-Ω ODT setting is used on the remaining DDR3 SDRAM DIMM testing featured in this document. Figure 2–8 shows the measured write eye diagram using Altera's Stratix III or Stratix IV memory board.

**Figure 2–8. Measured Write-Eye Diagram of a DDR3 SDRAM DIMM Using the 60-Ω ODT Setting**

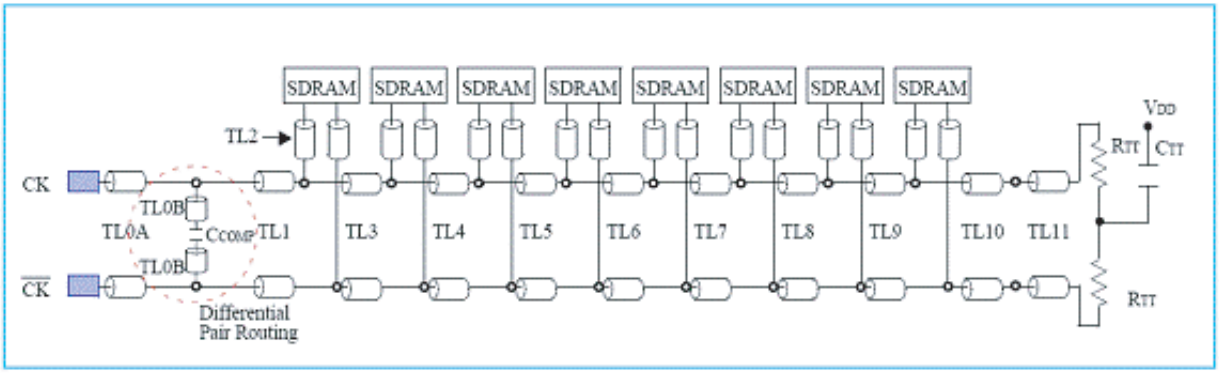



The measured eye diagram correlates well with the simulation. The faint line in the middle of the eye diagram is the effect of the refresh operation during a regular operation. Because these simulations and measurements are based on a narrow set of constraints, you must perform your own board-level simulation to ensure that the chosen ODT setting is right for your setup.

## Memory Clocks for DDR3 SDRAM UDIMM

For the DDR3 SDRAM UDIMM, you do not need to place any termination on your board because the memory clocks are already terminated on the DIMM. Figure 2–9 shows the net structure for the memory clocks and the location of the termination resistors, $R_{TT}$. The value of $R_{TT}$ is 36 Ω, which results in an equivalent differential termination value of 72 Ω. The DDR3 SDRAM DIMM also has a compensation capacitor, $C_{COMP}$ of 2.2 pF, placed between the differential memory clocks to improve signal quality.

**Figure 2–9. Clock Net Structure for a 64-Bit DDR3 SDRAM UDIMM** ***(Note 1)***




**Note to Figure 2–9:**

(1) Source: *PC3-6400/PC3-8500/PC3-10600/PC3-12800 DDR3 SDRAM Unbuffered DIMM Design Specification*, July 2007, JEDEC Solid State Technology Association

From Figure 2–9, you can see that the DDR3 SDRAM clocks are routed in a fly-by topology, as mentioned in "Read and Write Leveling" on page 2–3, resulting in the need for write-and-read leveling. Figure 2–10 shows the HyperLynx simulation of the differential clock seen at the first and last DDR3 SDRAM component on the UDIMM using the 50-Ω OCT setting on the output driver of the Stratix III or Stratix IV FPGA.

**Figure 2–10. Differential Memory Clock of a DDR3 SDRAM DIMM at the First and Last Component on the DIMM**

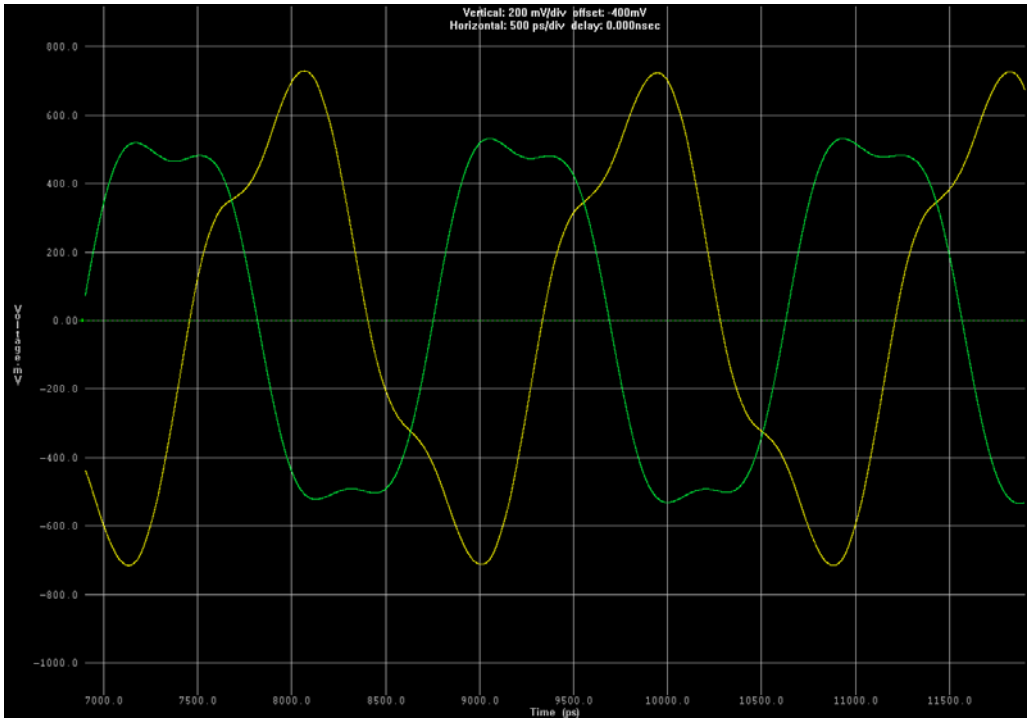
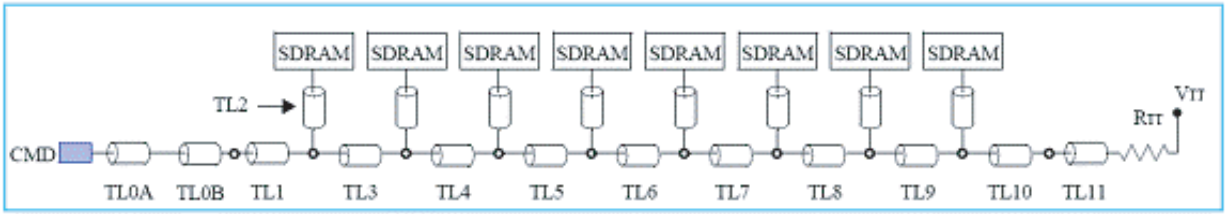



Figure 2–10 shows that the memory clock seen at the first DDR3 SDRAM component (the yellow signal) leads the memory clock seen at the last DDR3 SDRAM component (the green signal) by 1.3 ns, which is about 0.69 $t_{CK}$ for a 533 MHz operation.

## Commands and Addresses for DDR3 SDRAM UDIMM

Similar to memory clock signals, you do not need to place any termination on your board because the command and address signals are also terminated on the DIMM. Figure 2–11 shows the net structure for the command and address signals, and the location of the termination resistor, $R_{TT}$, which has an $R_{TT}$ value of 39 Ω

**Figure 2–11. Command and Address Net Structure for a 64-Bit DDR3 SDRAM Unbuffered DIMM *(Note 1)***




**Note to Figure 2–11:**

(1) Source: *PC3-6400/PC3-8500/PC3-10600/PC3-12800 DDR3 SDRAM Unbuffered DIMM Design Specification*, July 2007, JEDEC Solid State Technology Association

In Figure 2–11, observe that the DDR3 SDRAM command and address signals are routed in a fly-by topology, as mentioned in "Read and Write Leveling" on page 2–3, resulting in the need for write-and-read leveling.

Figure 2–12 shows the HyperLynx simulation of the command and address signal seen at the first and last DDR3 SDRAM component on the UDIMM, using a 25-ΩOCT setting on the output driver of the Stratix III or Stratix IV FPGA.

**Figure 2–12. Command and Address Eye Diagram of a DDR3 SDRAM DIMM at the First and Last DDR3 SDRAM Component at 533 MHz *(Note 1)***

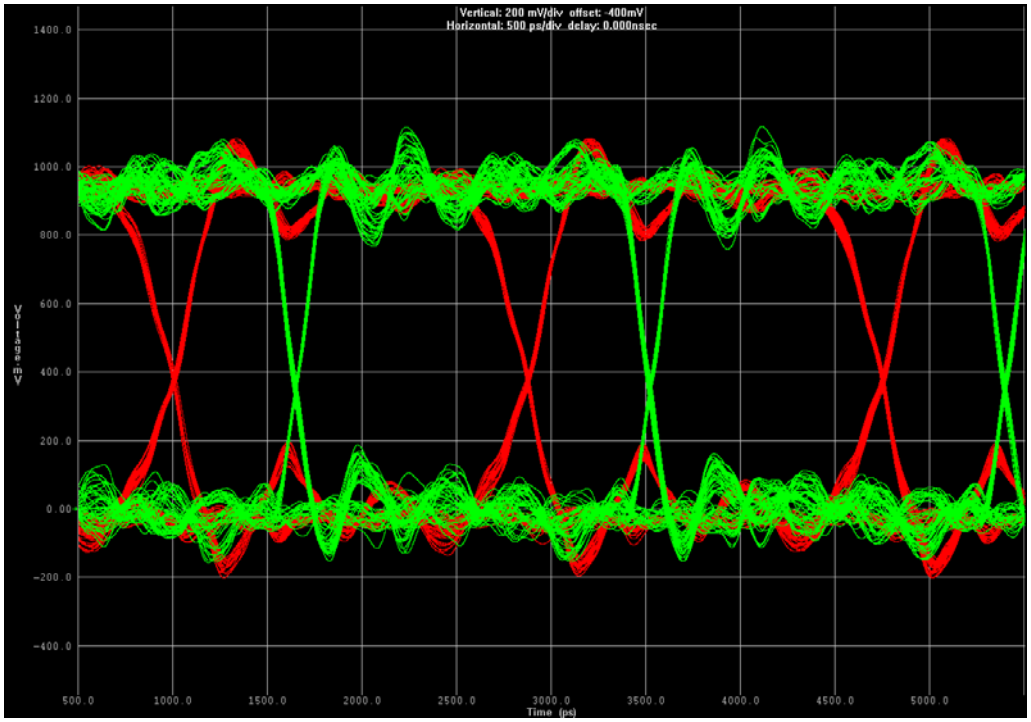


**Figure 2–12. Command and Address Eye Diagram of a DDR3 SDRAM DIMM at the First and Last DDR3 SDRAM Component at 533 MHz *(Note 1)***

**Note to Figure 2–12:**

(1) The command/address simulation is performed using a bit period of 1.875 ns.

Figure 2–12 shows that the command and address signal seen at the first DDR3 SDRAM component (the green signal) leads the command and address signals seen at the last DDR3 SDRAM component (the red signal) by 1.2 ns, which is 0.64 $t_{CK}$ for a 533-MHz operation.

## DDR3 SDRAM Registered DIMM

The difference between a registered DIMM (RDIMM) and a UDIMM is that the clock, address, and command pins of the RDIMM are registered or buffered on the DIMM before they are distributed to the memory devices. For a controller, each clock, address, or command signal has only one load, which is the register or buffer. In a UDIMM, each controller pin must drive a fly-by wire with multiple loads.

You do not need to terminate the clock, address, and command signals signals on your board, because these signals are terminated at the register. However, because of the register, these signals become point-to-point signals and have improved signal integrity making the drive strength requirements of the FPGA driver pins more relaxed.

Similar to the signals in a UDIMM, the DQS, DQ, and DM signals on a RDIMM are not registered. To terminate these signals, refer to "DQS, DQ, and DM for DDR3 SDRAM UDIMM" on page 2–10.

## Stratix III, Stratix IV, and Stratix V FPGAs

The following sections review the termination on the single-ranked single DDR3 SDRAM DIMM interface side and investigate the use of different termination features available in Stratix III, Stratix IV, and Stratix V FPGA devices to achieve optimum signal integrity for your DDR3 SDRAM interface.

### DQS, DQ, and DM for Stratix III, Stratix IV, and Stratix V FPGA

As mentioned in "Dynamic OCT in Stratix III and Stratix IV Devices" on page 2–6, Stratix III, Stratix IV, and Stratix V FPGAs support the dynamic OCT feature, which switches from series termination to parallel termination depending on the mode of the I/O buffer. Because DQS and DQ are bidirectional signals, DQS and DQ can be both transmitters and receivers. "DQS, DQ, and DM for DDR3 SDRAM UDIMM" on page 2–10 describes the signal quality of DQ, DQS, and DM when the Stratix III, Stratix IV, or Stratix V FPGA device is the transmitter with the I/O buffer set to a 50-Ω series termination.

This section details the condition when the Stratix III, Stratix IV, or Stratix V device is the receiver, the Stratix III, Stratix IV, and Stratix V I/O buffer is set to a 50-Ω parallel termination, and the memory is the transmitter. DM is a unidirectional signal, so the DDR3 SDRAM component is always the receiver.

Refer to "DQS, DQ, and DM for DDR3 SDRAM UDIMM" on page 2–10 for receiver termination recommendations and transmitter output drive strength settings.

Figure 2–13 illustrates the DDR3 SDRAM interface when the Stratix III, Stratix IV, or Stratix V FPGA device is reading from the DDR3 SDRAM using a 50-Ω parallel OCT termination on the Stratix III, Stratix IV, or Stratix V FPGA device, and the DDR3 SDRAM driver output impedance is set to 34 Ω.

**Figure 2–13. DDR3 SDRAM Component Driving the Stratix III, Stratix IV, and Stratix V FPGA Device with Parallel 50-Ω OCT Turned On**

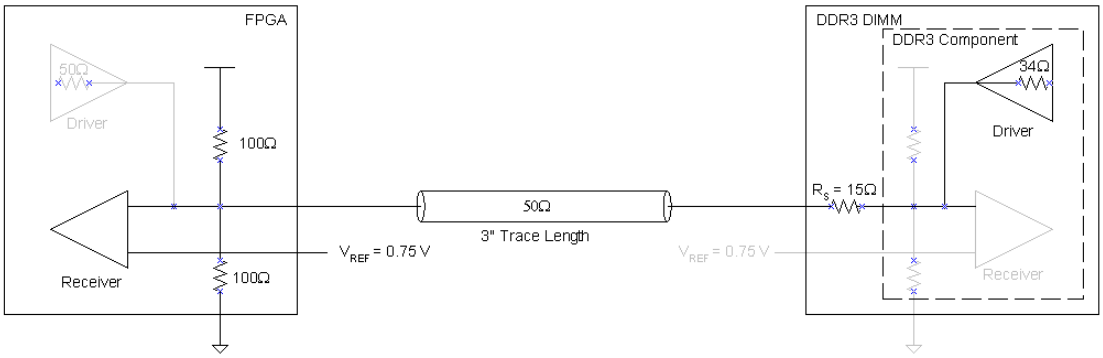
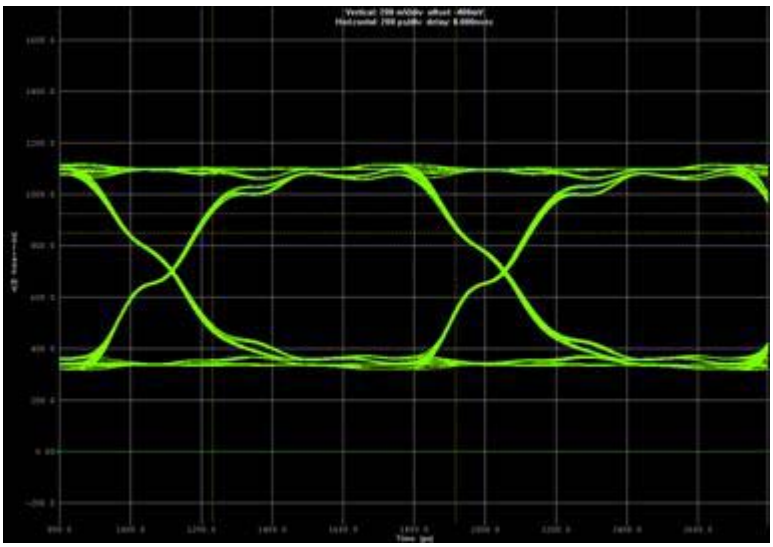



Figure 2–14 shows the simulation of a read from the DDR3 SDRAM DIMM with a 50-Ω parallel OCT setting on the Stratix III or Stratix IV FPGA device.

**Figure 2–14. Read-Eye Diagram of a DDR3 SDRAM DIMM at the Stratix III and Stratix IV FPGA Using a Parallel 50-Ω OCT Setting**



Use of the Stratix III, Stratix IV, or Stratix V parallel 50-Ω OCT feature matches receiver impedance with the transmission line characteristic impedance. This eliminates any reflection that causes ringing, and results in a clean eye diagram at the Stratix III, Stratix IV, or Stratix V FPGA.

### Memory Clocks for Stratix III, Stratix IV, and Stratix V FPGA

Memory clocks are unidirectional signals. Refer to "Memory Clocks for DDR3 SDRAM UDIMM" on page 2–13 for receiver termination recommendations and transmitter output drive strength settings.

### Commands and Addresses for Stratix III and Stratix IV FPGA

Commands and addresses are unidirectional signals. Refer to "Commands and Addresses for DDR3 SDRAM UDIMM" on page 2–15 for receiver termination recommendations and transmitter output drive strength settings.

## Termination for DDR3 SDRAM Components (With Leveling)

This section discusses terminations used to achieve optimum performance for designing the DDR3 SDRAM interface using discrete DDR3 SDRAM components.

In addition to using DDR3 SDRAM DIMM to implement your DDR3 SDRAM interface, you can also use DDR3 SDRAM components. However, for applications that have limited board real estate, using DDR3 SDRAM components reduces the need for a DIMM connector and places components closer, resulting in denser layouts.

### DDR3 SDRAM Components

The DDR3 SDRAM UDIMM is laid out to the JEDEC specification. The JEDEC specification is available from either the JEDEC Organization website (www.JEDEC.org) or from the memory vendors. However, when you are designing the DDR3 SDRAM interface using discrete SDRAM components, you may desire a layout scheme that is different than the DIMM specification. You have the following two options:

- Mimic the standard DDR3 SDRAM DIMM, using a fly-by topology for the memory clocks, address, and command signals. This options needs read and write leveling, so you must use the UniPHY IP with leveling.

  For more information on this fly-by configuration, continue reading this chapter.

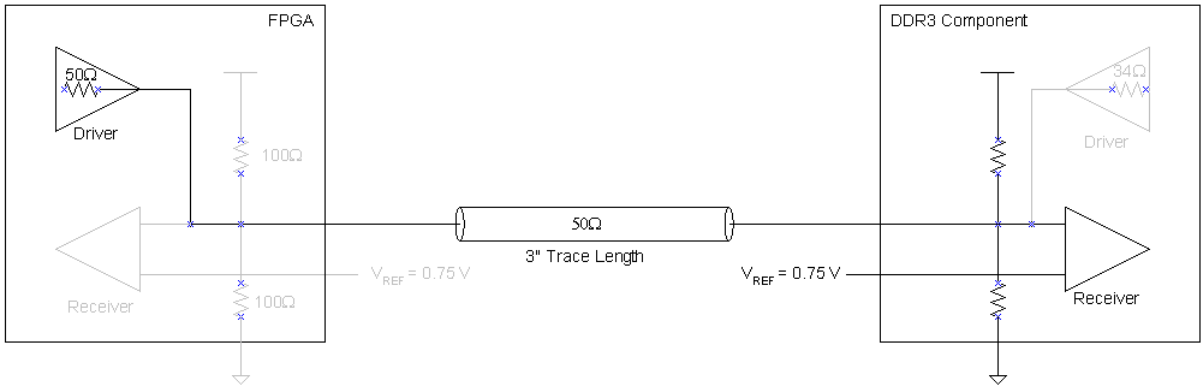
- Mimic a standard DDR2 SDRAM DIMM, using a balanced (symmetrical) tree-type topology for the memory clocks, address, and command signal. Using this topology results in unwanted stubs on the command, address, and clock, which degrades signal integrity and limits the performance of the DDR3 SDRAM interface.

  For more information on using this non-standard symmetrical configuration, refer to "Termination for DDR3 SDRAM Components (Without Leveling)" on page 2–29.

## DQS, DQ, and DM for DDR3 SDRAM Components

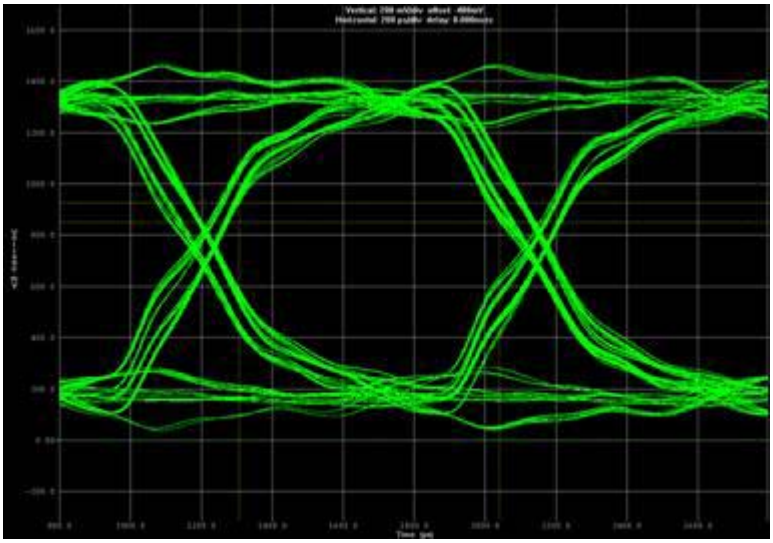
When you are laying out the DDR3 SDRAM interface using Stratix III, Stratix IV, or Stratix V devices, Altera recommends that you not include the 15-Ω stub series resistor that is on every DQS, DQ, and DM signal; unless your simulation shows that the absence of this resistor causes extra reflection. Although adding the the 15-Ω stub series resistor may help to maintain constant impendance in some cases, it also slightly reduces signal swing at the receiver. It is unlikely that by removing this resistor the waveform shows a noticeable reflection, but it is your responsibility to prove by simulating your board trace. Therefore, Altera recommends the DQS, DQ, and DM topology shown in Figure 2–15 when the Stratix III, Stratix IV, or Stratix V FPGA is writing to the DDR3 SDRAM.

**Figure 2–15. Stratix III, Stratix IV, and Stratix V FPGA Writing to a DDR3 SDRAM Components**



When you are using DDR3 SDRAM components, there are no DIMM connectors. This minimizes any impedance discontinuity, resulting in better signal integrity. Figure 2–16 shows the simulated write-eye diagram at the DQ0 of a DDR3 SDRAM component using the 120-Ω ODT setting, and driven by a Stratix III or Stratix IV FPGA using a calibrated series 50-Ω OCT setting.

**Figure 2–16. Write-Eye Diagram of a DDR3 SDRAM Component Using a 120-Ω ODT Setting**



Similarly, Figure 2–17 shows the simulated write-eye diagram at the DQ0 of a DDR3 SDRAM component using the 60-Ω ODT setting, and driven by a Stratix III or Stratix IV FPGA using a calibrated series 50-Ω OCT setting.

**Figure 2–17. Write-Eye Diagram of a DDR3 SDRAM Component Using a 60-Ω ODT Setting**

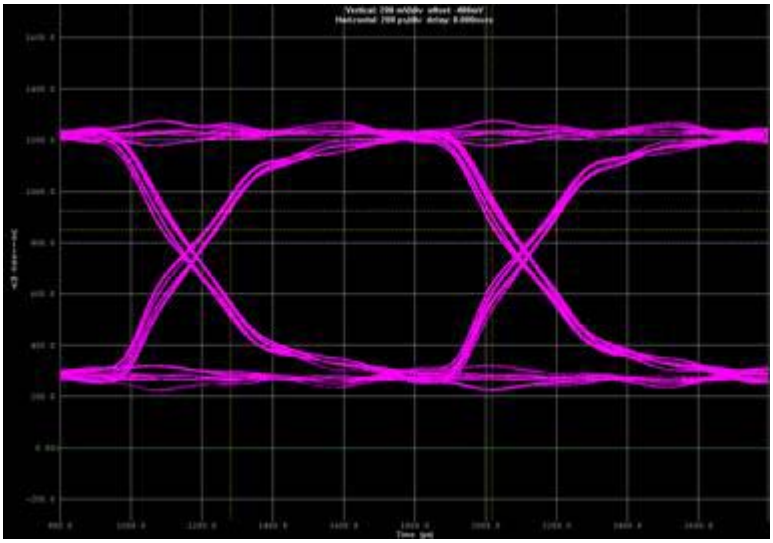


Table 2–4 compares the effects of the series stub resistor on the eye diagram at the DDR3 SDRAM (receiver) when the Stratix III or Stratix IV FPGA is writing to memory.

**Table 2–4. Simulated Write-Eye Diagram with and without $R_S$ and Using Different ODT Settings**

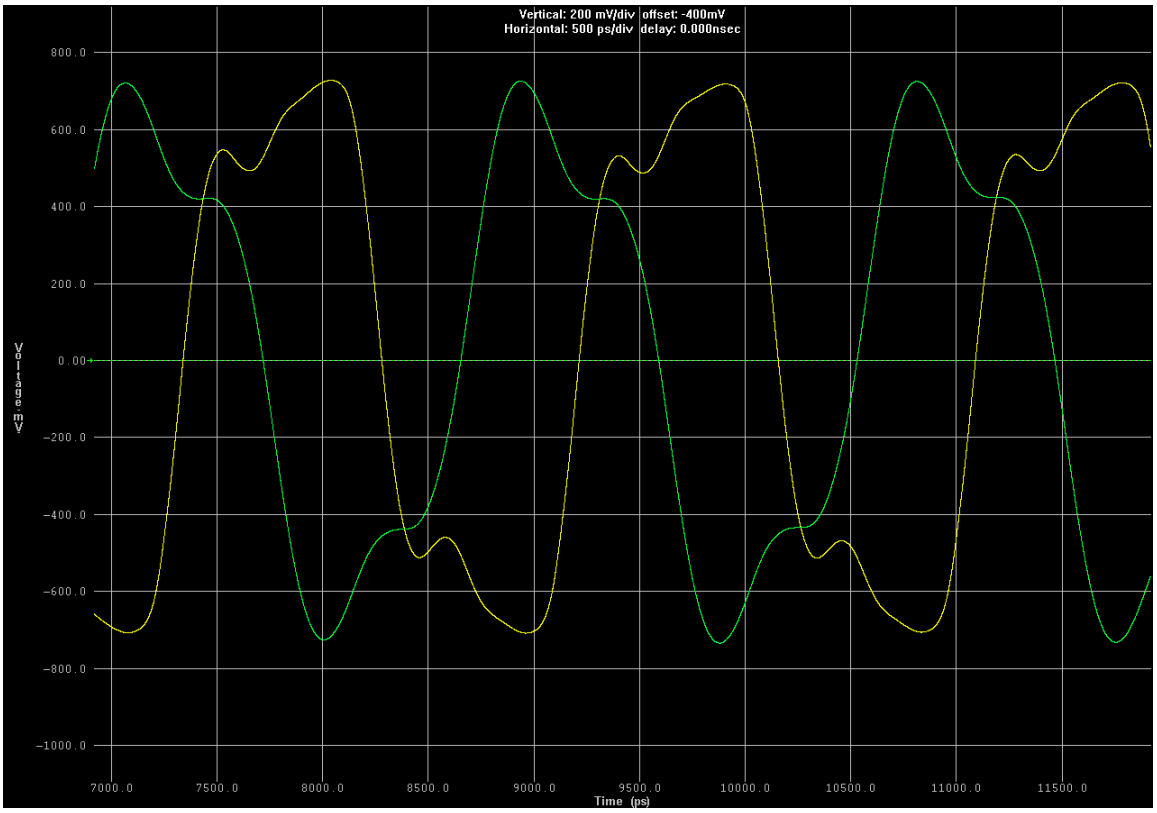
| ODT | Eye Height (V) | Eye Width (ps) | Overshoot (V) | Undershoot (V) |
|---|---|---|---|---|
| 120-Ω ODT with $R_S$ | 0.84 | 713 | — | — |
| 60-Ω ODT with $R_S$ | 0.73 | 715 | — | — |
| 120-Ω ODT without $R_S$ | 0.95 | 734 | — | — |
| 60-Ω ODT without $R_S$ | 0.83 | 737 | — | — |

Including the 15-Ω stub series resistor increases source impedance, and due to the voltage dividing effect, the voltage swing receiver drops.

## Memory Clocks for DDR3 SDRAM Components

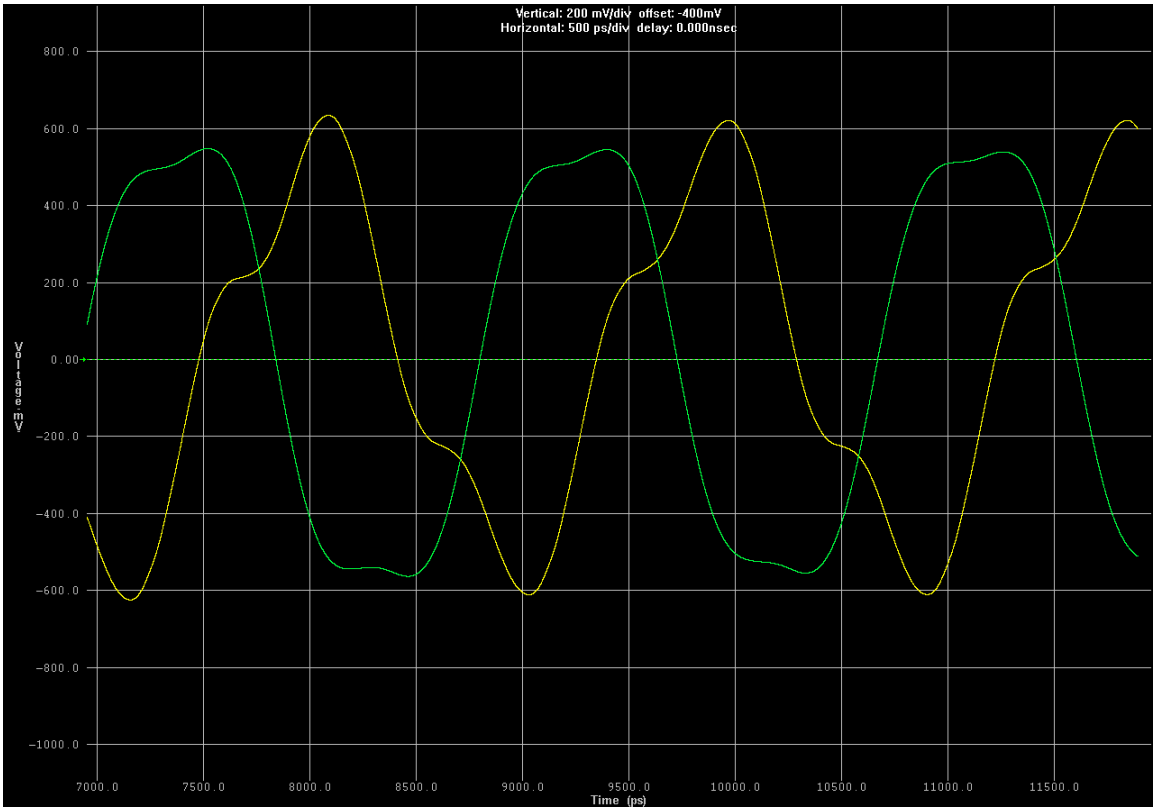
When you use DDR3 SDRAM components, you must account for the compensation capacitor and differential termination resistor between the differential memory clocks of the DIMM. Figure 2–18 shows the HyperLynx simulation of the differential clock seen at the first and last DDR3 SDRAM component using a fly-by topology on a board, without the 2.2 pF compensation capacitor using the 50-Ω OCT setting on the output driver of the Stratix III, Stratix IV, or Stratix V FPGA.

**Figure 2–18. Differential Memory Clock of a DDR3 SDRAM Component without the Compensation Capacitor at the First and Last Component Using a Fly-by Topology on a Board**



Without the compensation capacitor, the memory clocks (the yellow signal) at the first component have significant ringing, whereas, with the compensation capacitor the ringing is dampened. Similarly, the differential termination resistor needs to be included in the design. Depending on your board stackup and layout requirements, you choose your differential termination resistor value. Figure 2–19 shows the HyperLynx simulation of the differential clock seen at the first and last DDR3 SDRAM component using a fly-by topology on a board, and terminated with 100 Ω instead of the 72 Ω used in the DIMM.

**Figure 2–19. Differential Memory Clock of a DDR3 SDRAM DIMM Terminated with 100 Ω at the First and Last Component Using a Fly-by Topology on a Board**




Terminating with 100 Ω instead of 72 Ω results in a slight reduction in peak-to-peak amplitude. To simplify your design, use the terminations outlined in the JEDEC specification for DDR3 SDRAM UDIMM as your guide and perform simulation to ensure that the DDR3 SDRAM UDIMM terminations provide you with optimum signal quality.

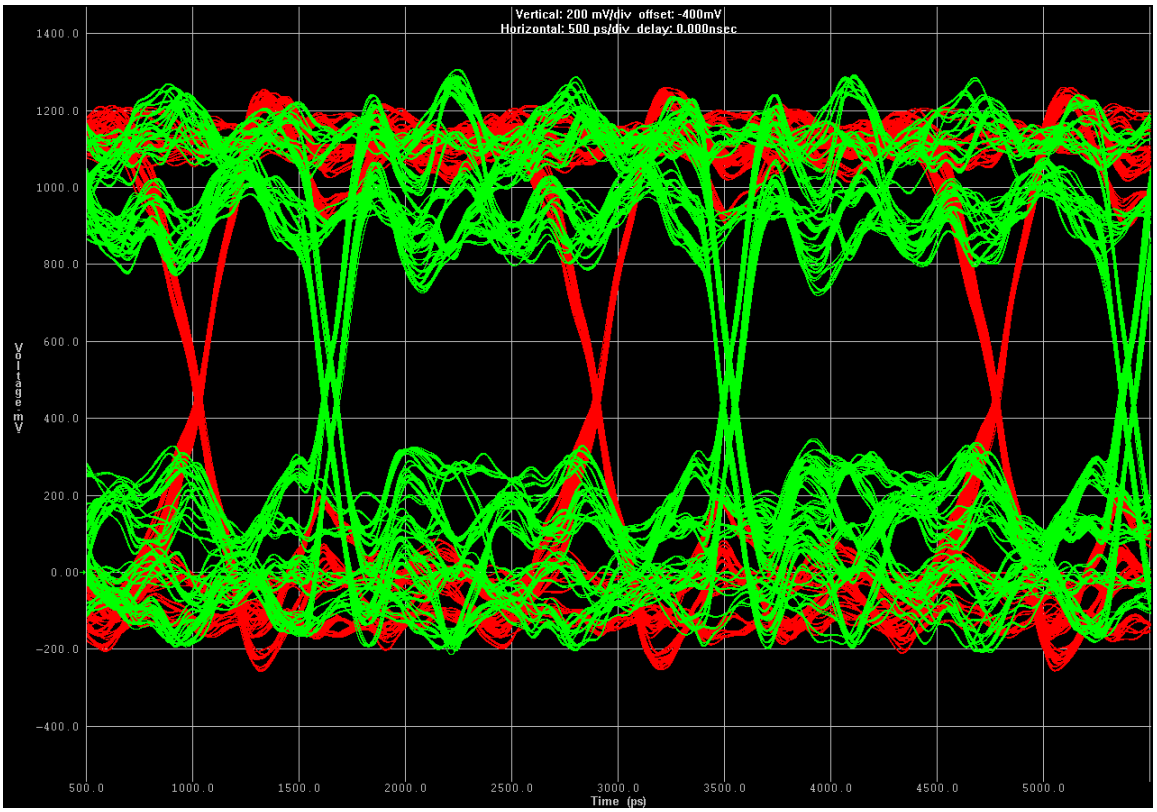
In addition to choosing the value of the differential termination, you must consider the trace length of the memory clocks. There is no specification on the flight-time skew between the first and last component when designing with DDR3 SDRAM components on your board. Altera's DDR3 UniPHY IP currently supports a flight-time skew of no more than 0.69 $t_{CK}$. If you use Altera's DDR3 UniPHY IP to create your DDR3 SDRAM interface, ensure that the flight-time skew of your memory clocks is not more than 0.69 $t_{CK}$.

☞ Refer to "Layout Considerations For DIMMs and Leveled Components" on page 2–26 for more information about layout guidelines for DDR3 SDRAM components.

## Command and Address Signals for DDR3 SDRAM

As with memory clock signals, you must account for the termination resistor on the command and address signals when you use DDR3 SDRAM components. Choose your termination resistor value depending on your board stackup and layout requirements. Figure 2–20 shows the HyperLynx simulation of the command and address seen at the first and last DDR3 SDRAM component using a fly-by topology on a board terminated with 60 Ω instead of the 39 Ω used in the DIMM.

**Figure 2–20. Command and Address Eye Diagram of a DDR3 SDRAM Component Using Fly-by Topology on a Board at the First and Last DDR3 SDRAM Component at 533 MHz, Terminated with 60 Ω**




Terminating with 60 Ω instead of 39 Ω results in eye closure in the signal at the first component (the green signal), while there is no effect on the signal at the last component (the red signal). To simplify your design with discrete DDR3 SDRAM components, use the terminations outlined in the JEDEC specification for DDR3 SDRAM UDIMM as your guide, and perform simulation to ensure that the DDR3 SDRAM UDIMM terminations provide you with the optimum signal quality.

As with memory clocks, you must consider the trace length of the command and address signals so that they match the flight-time skew of the memory clocks.

## Stratix III, Stratix IV, and Stratix V FPGAs

The following sections describe termination used on the DDR3 SDRAM component interface side and investigate using the different termination features available in Stratix III, Stratix IV, or Stratix V FPGA devices, so you can achieve optimum signal integrity for your DDR3 SDRAM interface.

### DQS, DQ, and DM Termination for Stratix III, Stratix IV, and Stratix V FPGA

Similar to the scenario highlighted in "DQS, DQ, and DM for Stratix III, Stratix IV, and Stratix V FPGA" on page 2–16, the Stratix III, Stratix IV, or Stratix V FPGA device is the receiver, the Stratix III, Stratix IV, or Stratix V I/O buffer is set to a 50-Ω parallel termination, and the memory is the transmitter. The difference between the setup in "DQS, DQ, and DM for Stratix III, Stratix IV, and Stratix V FPGA" on page 2–16 and the setup in this section is that there is no series stub resistor on the DQS, DQ, and DM signals. DM is a unidirectional signal, so the DDR3 SDRAM component is always the receiver. Refer to "DQS, DQ, and DM for DDR3 SDRAM Components" on page 2–19 for receiver termination recommendations and transmitter output drive strength settings.

Figure 2–21 illustrates the DDR3 SDRAM interface when the Stratix III, Stratix IV, or Stratix V FPGA device is reading from the DDR3 SDRAM using a 50-Ω parallel OCT termination on the Stratix III, Stratix IV, or Stratix V FPGA device and the DDR3 SDRAM driver output impedance is set to 34 Ω without the series stub resistor of 15 Ω.

**Figure 2–21. DDR3 SDRAM Component Driving the Stratix III, Stratix IV, and Stratix V FPGA Device with Parallel 50-Ω OCT Turned On**

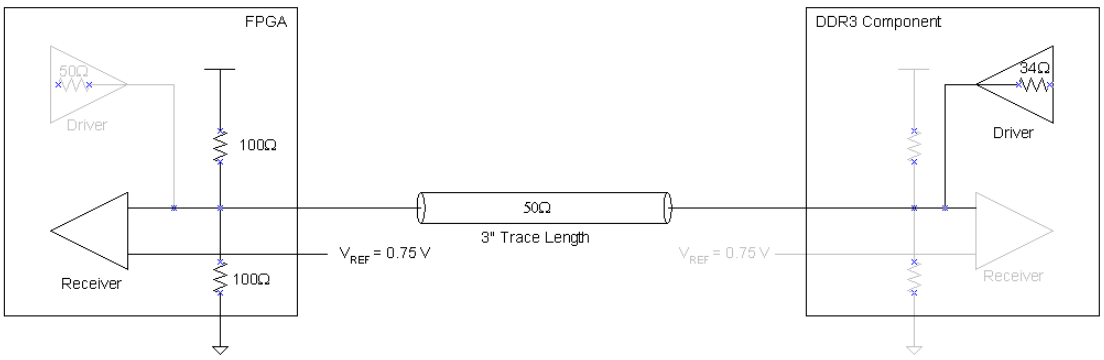


Figure 2–22 shows a simulation of a read from the DDR3 SDRAM DIMM with a 50-Ω parallel OCT setting on the Stratix III or Stratix IV FPGA device.

**Figure 2–22. Read-Eye Diagram of a DDR3 SDRAM Component at the Stratix III and Stratix IV FPGA Using a Parallel 50-W OCT Setting**

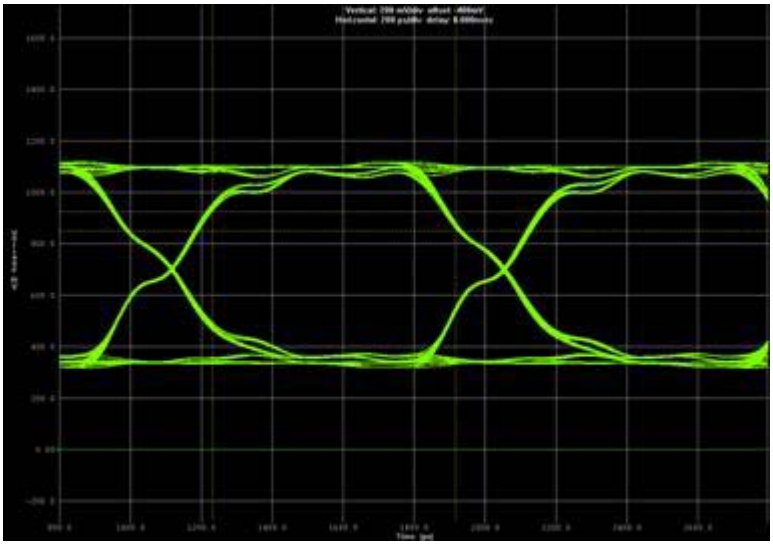


Table 2–5 compares the effects of the series stub resistor on the eye diagram at the Stratix III or Stratix IV FPGA (receiver) when the Stratix III or Stratix IV FPGA is reading from the memory.

**Table 2–5. Read-Eye Diagram with and without $R_S$ Using 50-Ω Parallel OCT**

| ODT | Eye Height (V) | Eye Width (ps) | Overshoot (V) | Undershoot (V) |
|---|---|---|---|---|
| With $R_S$ | 0.70 | 685 | — | — |
| Without $R_S$ | 0.73 | 724 | — | — |

Without the 15-Ω stub series resistor to dampen the signal, the signal at the receiver of the Stratix III or Stratix IV FPGA driven by the DDR3 SDRAM component is larger than the signal at the receiver of the Stratix III or Stratix IV FPGA driven by DDR3 SDRAM DIMM (Figure 2–13), and similar to the write-eye diagram in "DQS, DQ, and DM for DDR3 SDRAM Components" on page 2–19.

## Memory Clocks Termination for Stratix III and Stratix IV FPGA

Memory clocks are unidirectional signals. Refer to "Memory Clocks for DDR3 SDRAM Components" on page 2–21 for receiver termination recommendations and transmitter output drive strength settings.

## Command and Address Termination for Stratix III, Stratix IV, and Stratix V FPGA

Commands and addresses are unidirectional signals. Refer to "Command and Address Signals for DDR3 SDRAM" on page 2–23 for receiver termination recommendations and transmitter output drive strength setting.

# Layout Considerations For DIMMs and Leveled Components

This section discusses general layout guidelines for designing your DDR3 SDRAM interface. These layout guidelines help you plan your board layout, but are not meant as strict rules that must be adhered to. Altera recommends that you perform your own board-level simulations to ensure that the layout you choose for your board allows you to achieve your desired performance.

The layout guidelines are specifically for a DDR3 UDIMM clocked at 533 MHz.

## Trace Impedance

The layout of single-ended signal traces are to be 50 Ω and the differential signal traces are to be 100 Ω with a ±10% tolerance. Remove unused via pads as these cause unwanted capacitance.

## Decoupling

To minimize inductance, use 0.1 µF in 0402 size or smaller capacitors. Keep $V_{TT}$ voltage decoupling close to the DDR3 SDRAM components and pull-up resistors. Connect decoupling capacitors between $V_{TT}$ and $V_{DD}$ using a 0.1 µF capacitor for every other $V_{TT}$ pin. For $V_{DD}$ and $V_{DDQ}$, use 0.1 µF and 0.01 µF capacitors for every $V_{DD}$ and $V_{DDQ}$ pin.

## Power

Route the ground, 1.5 V, and 0.75 V as planes. Route $V_{CCIO}$ for memories in a single-split plane with at least a 20-mil (0.508 mm) gap of separation. Route $V_{TT}$ as islands or 250-mil (6.35 mm) power traces. Route oscillators and PLL power as islands or 100-mil (2.54 mm) power traces.

## Maximum Trace Length

Maximum trace length for all signals from FPGA to the first DIMM slot is 4.5 inches. Maximum trace length for all signals from DIMM slot to DIMM slot is 0.425 inches.

When interfacing with multiple DDR3 SDRAM components, the maximum trace length for address, command, control and clock from FPGA to first component is maximum 7 inches.

Maximum trace length for DQ, DQS, DQS#, and DM from FPGA to first component is 5 inches.

## General Routing Guidelines

Route using 45° angles and not 90° corners. Do not route critical signals across split planes. Route over appropriate $V_{CC}$ and ground planes. Avoid routing memory signals closer than 25-mil (0.635 mm) to the memory clocks. Keep the signal routing layers close to ground and power planes. All specified delay matching requirements include PCB trace delays, different layer propagation velocity variance, and crosstalk. To minimize PCB layer propagation variance, Altera recommends that you always route signals from the same net group on the same layer.

### Clock Routing Guidelines

Route clocks on inner layers with outer-layer run lengths held to under 500 mils (12.7 mm).

- 10-mil spacing for parallel runs < 0.5 inches (2× trace-to-plane distance)
- 15-mil spacing for parallel runs between 0.5 and 1.0 inches (3× trace-to-plane distance)
- 20-mil spacing for parallel runs between 1 and 6 inches (4× trace-to-plane distance)

Clocks must maintain a length matching between clock pairs of ±5 ps or approximately ± 25 mils (0.635 mm). Differential clocks need to maintain length matching between positive and negative signals of ±2 ps or approximately ± 10 mils (0.254 mm), routed in parallel. The space between differential pairs must be at least 2× the trace width of the differential pair to minimize loss and maximize interconnect density. The maximum length from the first SDRAM to the last SDRAM must be no more than 6 inches (approximately 153 mm) or 0.69 $t_{CK}$, which is the same maximum length for clocks specified by JEDEC for UDIMM. This maximum clock-length specification is only valid for UDIMM. For other DIMM configurations, check the necessary JEDEC specifications, as the maximum clock length may be different. For example, JEDEC specifies the maximum clock length for SODIMM to be 6.5 inches (approximately 166 mm).

For example, you must route differential clocks differentially (5 mil trace width, 10-15 mil space on centers, and equal in length to signals in the Address/Command Group). Take care with the via pattern used for clock traces. To avoid transmission-line-to-via mismatches, Altera recommends that your clock via pattern be a Ground-Signal-Signal-Ground (GSSG) topology (via topology: GND | CLKP | CLKN | GND).

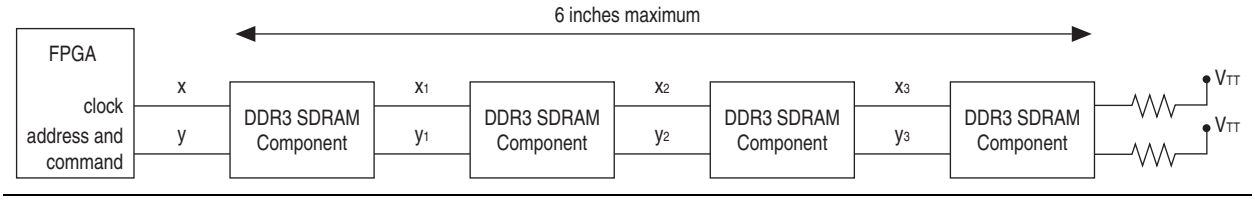
### Address and Command Routing Guidelines

Similar to the clock signals in DDR3 SDRAM, address and command signals are routed in a daisy chain topology from the first SDRAM to the last SDRAM. The maximum length from the first DRAM to the last SDRAM must be no more than 6 inches (approximately 153 mm) or 0.69 $t_{CK}$, which is the same maximum length for clocks specified by JEDEC for UDIMMs. Ensure that each net maintains the same consecutive order. UDIMMs are more susceptible to crosstalk and are generally noisier than buffered DIMMs. Route the address and command signals of UDIMMs on a different layer than DQ and DM, and with greater spacing. Do not route differential clock and clock enable signals close to address signals. Route all addresses and commands to match the clock signals to within ±25 ps or approximately ± 125 mil (± 3.175 mm) to each discrete memory component.

Figure 2–23 shows the DDR3 SDRAM routing guidelines, where:

- $x = y \pm 125$ mil
- $x + x_1 = y + y_1 \pm 125$ mil
- $x + x_1 + x_2 = y + y_1 + y_2 \pm 125$ mil

**Figure 2–23. DDR3 SDRAM Component Routing Guidelines**




### DQ, DQS, and DM Routing Guidelines

All signals within a given byte-lane group must be matched in length with a maximum deviation of ±10 ps or approximately ± 50 mils (± 1.27 mm). Ensure all signals within a given byte lane group are routed on the same layer to avoid layer to layer transmission velocity differences, which otherwise increase the skew within the group. Keep the maximum byte-lane group-to-byte group matched length deviation to ± 150 ps or ± 0.8 inches (± 20 mm). The guidelines do not require the DQS, DQ, and DM signal groups to arrive in order; signals in group 1 do not have to arrive later than group 0 or earlier than group 2.

Maintain all other signals to a spacing that is based on its parallelism with other nets:

- 5 mils for parallel runs < 0.5 inches (approximately 1× spacing relative to plane distance)
- 10 mils for parallel runs between 0.5 and 1.0 inches (approximately 2× spacing relative to plane distance)
- 15 mils for parallel runs between 1.0 and 6.0 inches (approximately 3× spacing relative to plane distance)

Do not use DDR3 deskew to correct for more than 20 ps of DQ group skew. The deskew algorithm only removes the following possible uncertainties:

- Minimum and maximum die IOE skew or delay mismatch
- Minimum and maximum device package skew or mismatch
- Board delay mismatch of 20 ps
- Memory component DQ skew mismatch

Increasing any of these four parameters runs the risk of the deskew algorithm limiting, failing to correct for the total observed system skew. If the algorithm cannot compensate without limiting the correction, timing analysis shows reduced margins.

All the trace length matching requirements are from the FPGA package ball to DDR3 package ball, which means you have to take into account trace mismatching on different DIMM raw cards.

### Termination

The previous sections use the combination of DDR3 SDRAM ODT and Stratix III, Stratix IV, or Stratix V dynamic OCT for DQS, DQS#, DQ, and DM. This practice reduces the need for external termination, and thus reduces both bill-of materials (BOM) cost and PCB size.

When using DIMMs, you have no concerns about terminations on memory clocks, addresses, and commands. If you are using components, use an external parallel termination of 40 Ω to $V_{TT}$ at the end of the fly-by daisy chain topology on the addresses and commands. For memory clocks, use an external parallel termination of 75 Ω differential at the end of the fly-by daisy chain topology on the memory clocks. Using fly-by daisy chain topology helps reduce any stub reflection. Keep the length of the traces to the termination to within 0.5 inch (14 mm). Use resistors with tolerances of 1 to 2%.

## Termination for DDR3 SDRAM Components (Without Leveling)

This section discusses the I/O standards, drive strength, termination and topologies to use so that you achieve optimum performance when designing with DDR3 SDRAM components without leveling. Altera supports the use of DDR3 SDRAM components using a PHY without leveling.

To use the PHY without leveling, you should layout the DDR3 SDRAM components on your PCB in a DDR2-like topology. Operating DDR3 SDRAM components without leveling requires tighter layout rules and the use of more complex topologies. This section discusses these termination and layout requirements.

### DDR3 SDRAM Components

This chapter describes how to implement the nonstandard DDR2-like balanced (symmetrical) topology for command, address, and clock signals. Using this alternative topology results in unwanted stubs on the address, command, and clock signals, which degrades signal integrity and limits the performance of any DDR3 SDRAM interface.

#### DQS, DQ, and DM for DDR3 SDRAM Components

The DDR3 SDRAM PHY without leveling uses the same topology and termination settings for the DQS, DQ and DM signals as the DDR3 SDRAM with leveling (refer to "DQS, DQ, and DM for DDR3 SDRAM Components" on page 2–19). However, while the topology and termination of these signals is identical, the layout rules differ, because of the balanced command, address, and clock signals. DDR3 SDRAM without leveling interfaces require much tighter DQ group to DQ group timing, refer to "Layout Considerations (without Leveling)" on page 2–34.

#### Memory Clocks for DDR3 SDRAM Components

Memory clocks in a DDR3 SDRAM interface without leveling should follow the same topology guidelines as a DDR2 SDRAM-type interface. However, DDR3 SDRAM interfaces use SSTL15 type signaling instead of SSTL18.

For more information, refer to Chapter 1, DDR2 SDRAM Interface Termination, Drive Strength, Loading, and Board Layout Guidelines.

If your DDR3 SDRAM interface connects to a single component, you can use a simple point-to-point topology a 100 Ω differential terminator at the component end of the line.

Most interfaces use two, four, or eight DDR3 SDRAM components, so you should use a balanced T-type routing pattern, where all the trace segments are balanced for each path. The total trace length to the first DDR3 SDRAM component is identical to that of the last component, hence the trace delay for each component is the same, ensuring matched timing while helping to control any reflections.

Differentially terminate clocks at the component end of the line with a 100 Ω resistor.

For more than one DDR3 SDRAM component, split the clock using a balanced T-topology.

Place the 100 Ω termination resistor at the first split in the T (refer to Figure 2–24), or increase the resistor value and place a resistor at the end of each segment at the DDR3 SDRAM component (refer to Figure 2–25).

Typically two segments require 200 Ω resistors; and four segments require 400 Ω resistors, but Altera recommends that you simulate your specific topology to ascertain the correct value.

Figure 2–24 shows the termination resistor at the first split in the T.

**Figure 2–24. Placement of the Termination Resistor—at First Split**

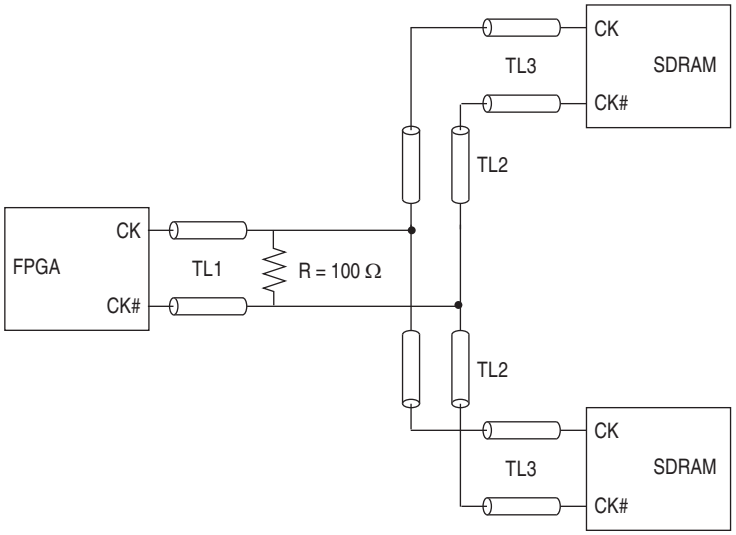
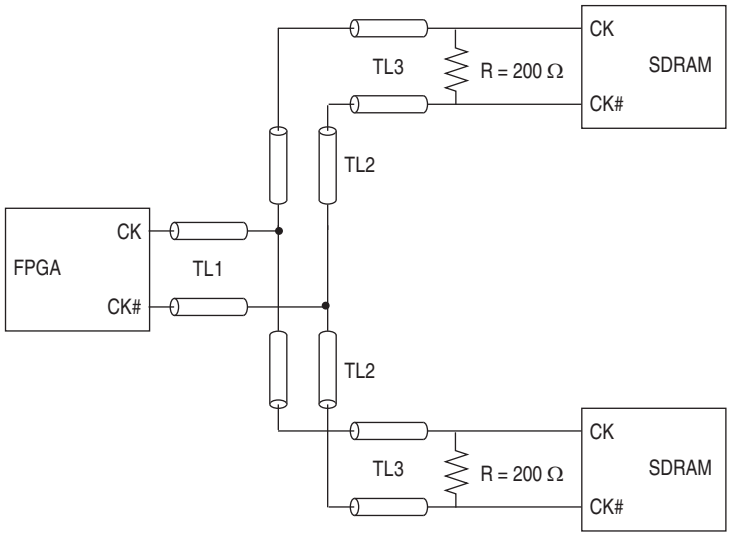


Figure 2–25 shows the termination resistor at the end of each segment at the DDR3 SDRAM component.

**Figure 2–25. Placement of the Termination Resistor—End of Each Segment**




Loading must not excessively degrade the slew rate of the memory clocks, so ideally a single differential clock pair does not drive more than four components. If a single clock pair drives eight or larger numbers of DDR3 SDRAM components, you must perform setup and hold deration to allow accurate timing analysis. Altera recommends that you simulate any proposed topology before board completion, so you can perform deration and final timing analysis. The DDR3 setup and hold deration may result in a lower than stated interface frequency to be achieved in any given device or speed grade combination.

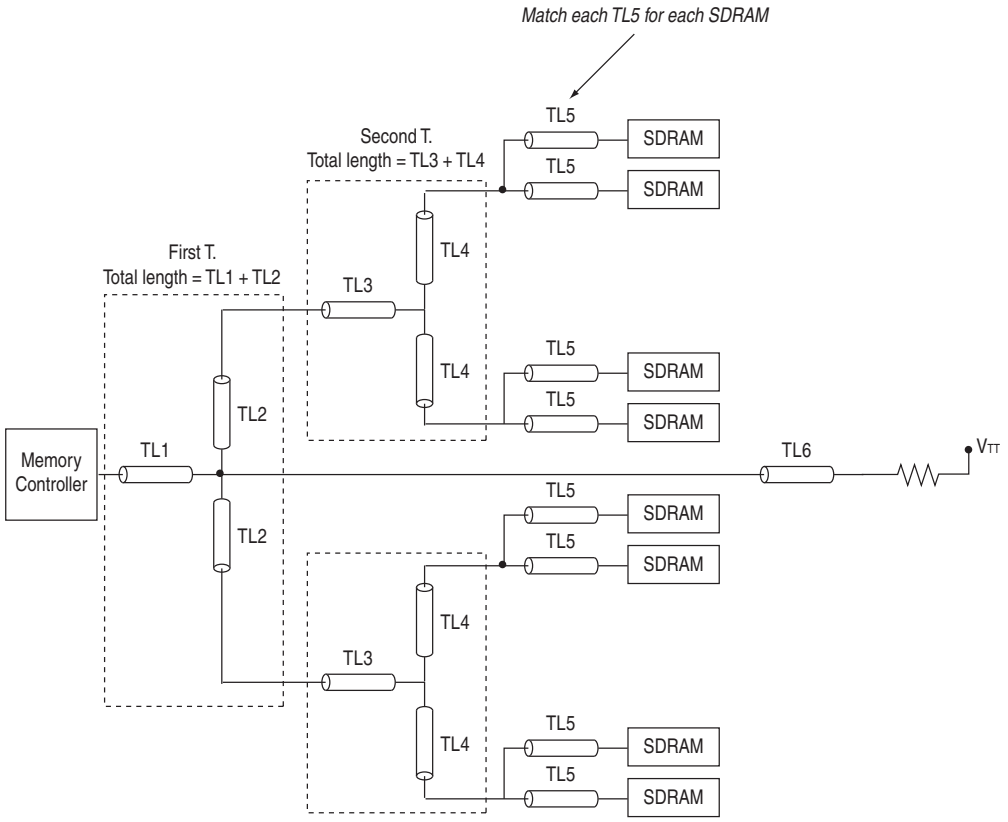
## Command and Address for DDR3 SDRAM Components

Command and address signals are similar to memory clocks in topology, so you should use a balanced T-type routing pattern, where all the trace segments are balanced for each path. The loading on the address and command signals is typically larger with eight or sixteen loads not unusual. You should mimic the topologies that JEDEC uses on DDR2 UDIMM raw cards A to C, as these topologies provide the best results.

Avoid topologies that JEDEC uses on DDR2 UDIMM raw cards D, E, and F. Raw card D topologies typically suffer from loading resonances, which reduce timing margin. Additionally, raw cards E and F are not symmetrical balanced trees, as they use a planar solution, which again can reduce timing margin.

Always terminate command and address signals with a 50 Ω resistor to $V_{TT}$. Always place this single 50 Ω resistor at the first split in the T (Figure 2–26).

**Figure 2–26. Placing the 50-Ω Resistor**




## Stratix III, Stratix IV, and Stratix V FPGAs

The following sections describe the termination used on the DDR3 SDRAM components interface side and the different termination features available in the Stratix III, Stratix IV, or Stratix V FPGA when using a PHY without leveling, so that optimum signal integrity can be achieved for your DDR3 SDRAM interface.

### DQS, DQ, and DM Termination for Stratix III, Stratix IV, and Stratix V FPGAs

It should be understood that the termination and topology for DQS, DQ, and DM signals is identical. The choice of leveling or without leveling DDR3 SDRAM PHY only affects the address, command, and clock termination schemes (refer to "DQS, DQ, and DM for Stratix III, Stratix IV, and Stratix V FPGA" on page 2–16).

Because of the different timing requirements, the layout (trace matching) constraints for DQS, DQ, and DM do differ (refer to "Layout Considerations (without Leveling)" on page 2–34).

### Memory Clocks Termination for Stratix III, Stratix IV, and Stratix V FPGA

Memory clocks are unidirectional signals. When using DDR3 SDRAM components without leveling, mimic the termination and topology used for DDR2 SDRAM components, substituting differential SSTL18 class I with differential SSTL15 class I.

For more information, refer to Chapter 1, DDR2 SDRAM Interface Termination, Drive Strength, Loading, and Board Layout Guidelines.

### Command and Address for Termination for Stratix III, Stratix IV, and Stratix V FPGAs

Commands and addresses are unidirectional signals. When using DDR3 SDRAM components without leveling, mimic the termination and topology used for DDR2 SDRAM components, substituting SSTL18 class I with SSTL15 class I.

For more information, refer to Chapter 1, DDR2 SDRAM Interface Termination, Drive Strength, Loading, and Board Layout Guidelines.

## Arria II GX FPGA

The following sections describe the termination used on the DDR3 SDRAM components interface side and the different termination features available in the Arria II GX devices when using a PHY without leveling, so that optimum signal integrity can be achieved for your DDR3 SDRAM interface without leveling.

DDR3 SDRAM component interfaces without leveling are routed identically to DDR2 SDRAM interfaces without leveling, hence DDR2 SDRAM interface recommendations apply.

### DQS, DQ and DM Termination for Arria II GX FPGAs

The termination and topology and layout of DQS, DQ, and DM signals is identical if DDR2 (differential DQS mode) is compared to DDR3 SDRAM.

DDR3 SDRAM without leveling on Arria II GX devices should be considered identical to any DDR2 SDRAM components interface.

The memory end termination (Table 2–1 and Table 2–2) still applies. But you should use the FPGA end termination settings from *AN 408: DDR2 Memory Interface Termination, Drive Strength, Loading, and Design Layout Guidelines*.

As Arria II GX devices don't feature dynamic OCT, 50 Ω parallel discrete termination to $V_{TT}$ should be used at the FPGA end of the line.

For more information, refer to "Layout Considerations (without Leveling)" on page 2–34.

### Memory Clocks Termination for Arria II GX FPGAs

Memory clocks are unidirectional signals. When using DDR3 SDRAM components without leveling, mimic the termination and topology used for DDR2 SDRAM components, substituting Differential SSTL18 Class I with Differential SSTL15 Class I.

For more information about component termination and FPGA drive strength settings, refer to "Memory Clocks for DDR3 SDRAM Components" on page 2–21 and Chapter 1, DDR2 SDRAM Interface Termination, Drive Strength, Loading, and Board Layout Guidelines.

### Command and Address for Termination for Arria II GX FPGAs

Commands and addresses are unidirectional signals. When using DDR3 SDRAM components without leveling, mimic the termination and topology used for DDR2 SDRAM components, substituting Differential SSTL18 Class I with Differential SSTL15 Class I.

For more information about component termination and FPGA drive strength settings, refer to "Command and Address Signals for DDR3 SDRAM" on page 2–23 and Chapter 1, DDR2 SDRAM Interface Termination, Drive Strength, Loading, and Board Layout Guidelines.

## Layout Considerations (without Leveling)

This section discusses general layout guidelines for designing your DDR3 SDRAM component without leveling interface. These guidelines help you plan your board layout, but are not meant as strict rules that must be adhered to. Altera recommends that you perform your own board-level simulations to ensure that your implemented topology allows you to achieve your desired performance.

For more information, refer to Chapter 1, DDR2 SDRAM Interface Termination, Drive Strength, Loading, and Board Layout Guidelines.

When mimicking DDR2 UDIMM JEDEC topologies, Altera recommends that you use only raw cards A to C, as these are balanced symmetrical topologies and result in the optimum performance. Raw cards D to F are not symmetrical and are planar solutions, so should be avoided where possible.

When following DDR2 SDRAM component guidelines, the I/O standard for DDR3 is SSTL15 and not SSTL18. DDR3 SDRAM components have enhanced ODT and output drive strength features that you can use to improve the SI performance of a DDR3 SDRAM component without leveling solution, above that of a standard DDR2 implementation.

Altera's timing analysis assumes single-ranked DDR3 SDRAM designs only. Dual or quad ranked designs require timing deration. For more information on multirank topologies and layout guidelines, refer to Chapter 3, Dual-DIMM DDR2 and DDR3 SDRAM Interface Termination, Drive Strength, Loading, and Board Layout Guidelines.

# Termination for DDR3 SDRAM Wide Interface (>72 bits)

This section discusses the different ways to lay out a wider DDR3 SDRAM interface to the FPGA. Choose the topology based on board trace simulation and the timing budget of your system.

The UniPHY IP supports up to a 144-bit wide DDR3 interface. You can either use discrete components or DIMMs to implement a wide interface (any interface wider than 72 bits). Altera recommends using leveling when you implement a wide interface with DDR3 components.

When you lay out for a wider interface, all rules and constaints discussed in the previous sections still apply. The DQS, DQ, and DM signals are point-to-point, and all the same rules discussed in "Layout Considerations For DIMMs and Leveled Components" on page 2–26 apply.

The main challenge for the design of the fly-by network topology for the clock, command, and address signals is to avoid signal integrity issues, and to make sure you route the DQS, DQ, and DM signals with the chosen topology.

## Fly-By Network Design for Clock, Command, and Address Signals

As described in "Termination for DDR3 SDRAM Components (With Leveling)" on page 2–18, the UniPHY IP requires the flight-time skew between the first DDR3 SDRAM component and the last DDR3 SDRAM component to be less than 0.69 $t_{CK}$ for memory clocks. This constraint limits the number of components you can have for each fly-by network.

If you design with discrete components, you can choose to use one or more fly-by networks for the clock, command, and address signals.

### Single Fly-by Network Topology

**Figure 2–27. Single Fly-By Network Topology**

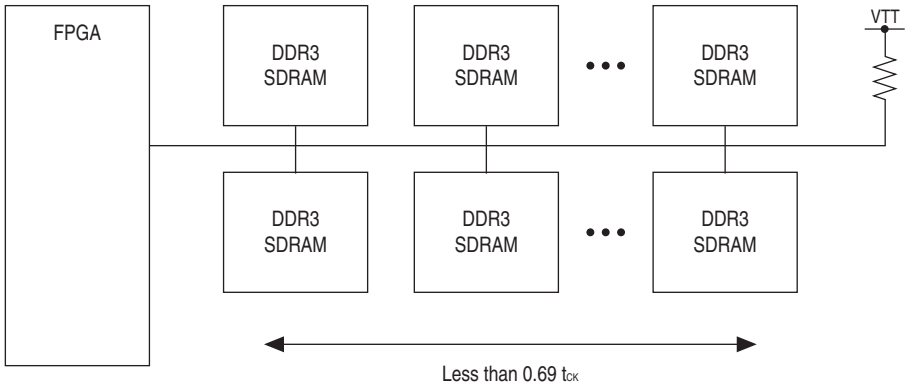


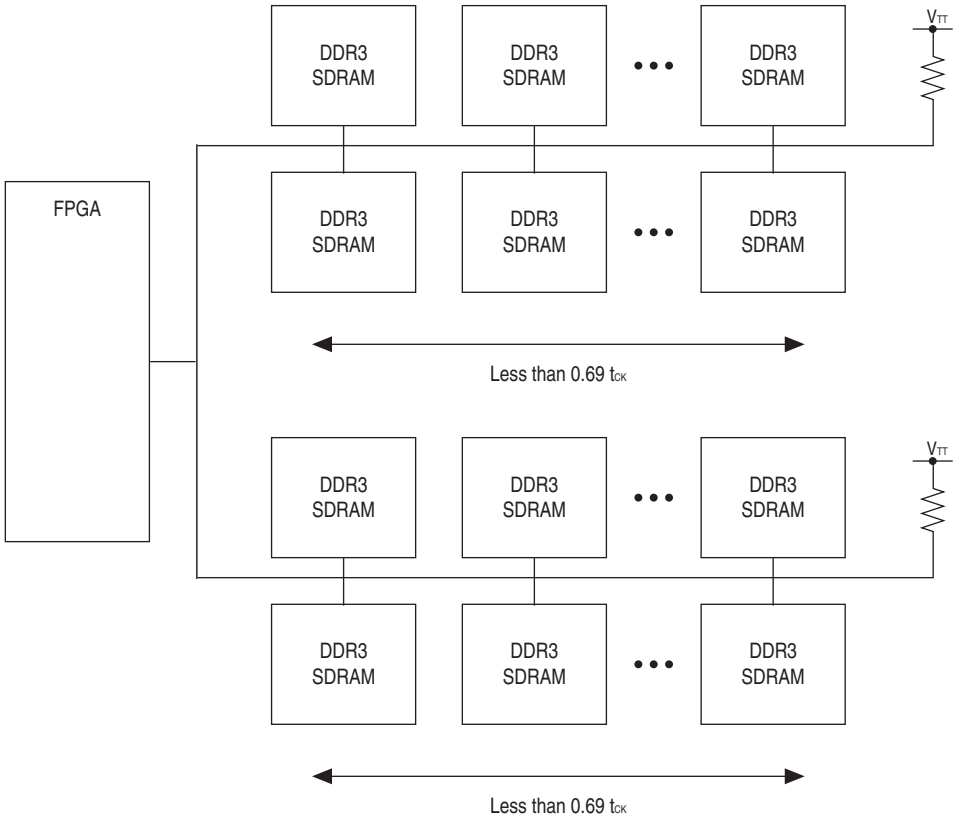

Figure 2–27 shows an example of a single fly-by network topology. Every DDR3 SDRAM component connected to the signal is a small load that causes discontinuity and degrades the signal. When using a single fly-by network topology, to minimize signal distortion, follow these guidelines:

- Use ×16 device instead ×4 or ×8 to minimize the number of devices connected to the trace.
- Keep the stubs as short as possible.
- Even with added loads from additional components, keep the total trace length short; keep the distance between the FPGA and the first DDR3 SDRAM component less than 5 inches.
- Simulate clock signals with your board trace model to ensure a decent waveform.

## Double Fly-by Network Topology

Figure 2–28 shows an example of a double fly-by network topology. This topology is not rigid but you can use it as an alternative option. The advantage of using this topology is that you can have more DDR3 SDRAM components in a system without violating the 0.69 $t_{CK}$ rule. However, as the signals branch out, the components still create discontinuity.

**Figure 2–28. Double Fly-By Network Topology**




You need to carry out some simulations to find the the location of the split, and the best impedance for the traces before and after the split. Usually the split at the FPGA pin gives good signal integrity, but it requires more traces to be routed out of the FPGA. You have to trade off routability for signal quality.

You can also consider using 2 DIMMs on each branch to replace the components. As the trade impedance on the DIMM card is 40 Ωto 60 Ωc, perform a board trace simulation to control the reflection to within the level your system can tolerate.

By using the new features of DDR3 SDRAM and the Stratix III, Stratix IV, or Stratix V FPGA, you simplify your design process for DDR3 SDRAM. Using the fly-by daisy chain topology increases the complexity of the datapath and controller design to achieve leveling, but also greatly improves performance and eases board layout for DDR3 SDRAM.

You can also use the DDR3 SDRAM components without leveling in a design if it may result in a more optimal solution, or use with devices that support the required electrical interface standard, but do not support the required read and write leveling functionality.

By using Altera FPGAs and the DDR3 SDRAM UniPHY IP, you simplify the datapath design and can take advantage of either the higher DDR3 SDRAM performance and straightforward board design in a design with leveling, or the lower power and cost performance advantages of DDR3 SDRAM components in a design without leveling.

# References

This chapter references the following documents:


- *JEDEC Standard Publication JESD79-3A, DDR3 SDRAM Specification, JEDEC Solid State Technology Association*
- *External Memory Interfaces in Stratix III Devices* chapter in volume 1 of the *Stratix III Device Handbook*
- *Stratix III Device I/O Features* chapter in volume 1 of the *Stratix III Device Handbook*
- *External Memory Interfaces in Stratix IV Devices* chapter in volume 1 of the *Stratix IV Device Handbook*
- *I/O Features in Stratix IV Devices* chapter in volume 1 of the *Stratix IV Device Handbook*
- *I/O Features in Stratix V Devices* chapter in volume 1 of the *Stratix V Device Handbook*
- *Micron Technical Note TN41-04: DDR3 Dynamic On-Die Termination Introduction*
- *Micron Technical Note TN41-08: DDR3-1066 Memory Design Guide for Two-Dimm Unbuffered Systems*
- *TN-41-02 DDR3 ZQ Calibration*, Micron
- *TN-41-04 DDR3 Dynamic On-Die Termination*, Micron
- *TN47-06: Updated JEDEC DDR2 Specifications*, Micron
- *TN47-17: DDR2 SODIMM Optimized Address/Command Nets*, Micron
- *TN47-19: DDR2 (Point-to-Point) Features and Functionality*, Micron
- *TN47-20: Point-to-Point Package Sizes and Layout Basics*, Micron
- *Consumer Electronics are Changing the Face of DRAMs*, Jody Defazio, Chip Design Magazine, June 29, 2007
- *DDR3 ODT and Dynamic ODT*, JEDEC DDR3 2007 Conference, Oct 3-4, San Jose, CA.
- *PC3-6400/PC3-8500/PC3-10600/PC3-12800 DDR3 SDRAM Unbuffered DIMM Design Specification*, July 2007, JEDEC Solid State Technology Association

# 3. Dual-DIMM DDR2 and DDR3 SDRAM Interface Termination, Drive Strength, Loading, and Board Layout Guidelines

This chapter describes guidelines for implementing dual unbuffered DIMM (UDIMM) DDR2 and DDR3 SDRAM interfaces. This chapter discusses the impact on signal integrity of the data signal with the following conditions in a dual-DIMM configuration:

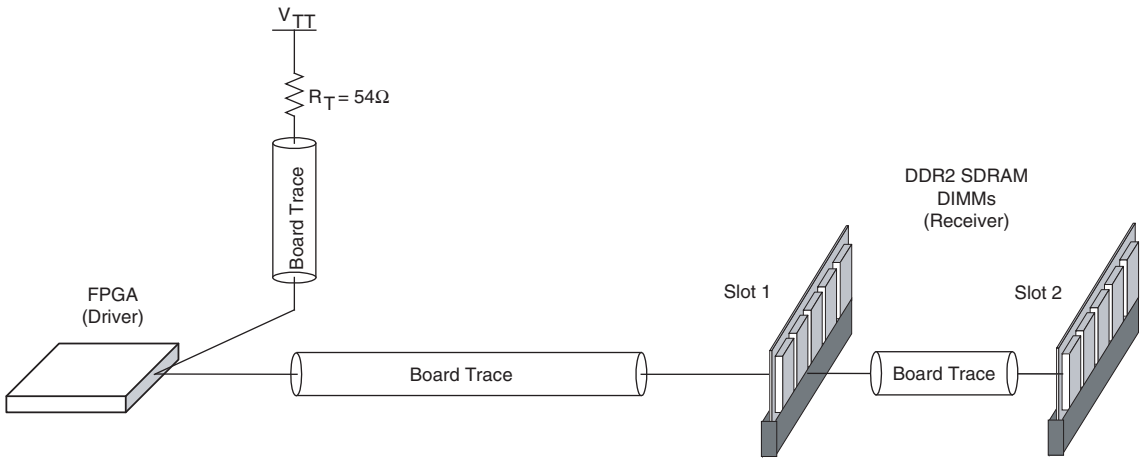
- Populating just one slot versus populating both slots
- Populating slot 1 versus slot 2 when only one DIMM is used
- On-die termination (ODT) setting of 75 Ω versus an ODT setting of 150 Ω

For detailed information about a single-DIMM DDR2 SDRAM interface, refer to Chapter 1, DDR2 SDRAM Interface Termination, Drive Strength, Loading, and Board Layout Guidelines.

## DDR2 SDRAM

This section describes guidelines for implementing a dual slot unbuffered DDR2 SDRAM interface, operating at up to 400-MHz and 800-Mbps data rates. Figure 3–1 shows a typical DQS, DQ, and DM signal topology for a dual-DIMM interface configuration using the ODT feature of the DDR2 SDRAM components.

**Figure 3–1. Dual-DIMM DDR2 SDRAM Interface Configuration** ***(Note 1)***




**Note to Figure 3–1:**

(1) The parallel termination resistor $R_T$ = 54 Ω to $V_{TT}$ at the FPGA end of the line is optional for devices that support dynamic on-chip termination (OCT).

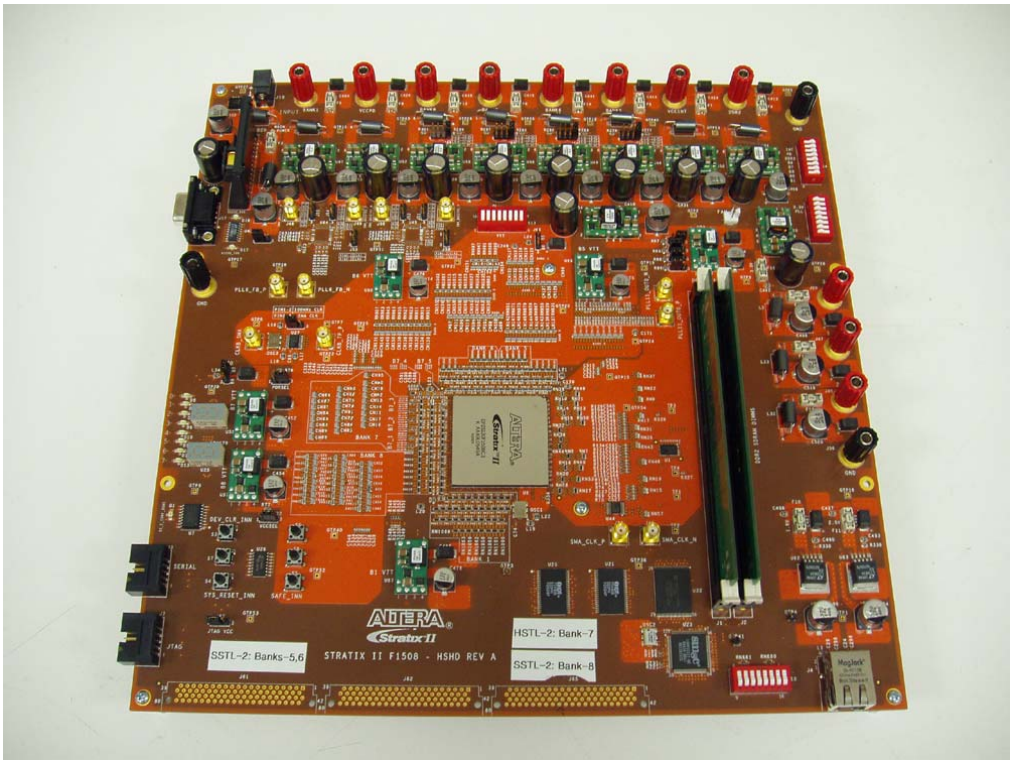
The simulations in this section use a Stratix II device-based board. Because of limitations of this FPGA device family, simulations are limited to 266 MHz and 533 Mbps so that comparison to actual hardware results can be directly made.

## Stratix II High Speed Board

To properly study the dual-DIMM DDR2 SDRAM interface, the simulation and measurement setup evaluated in the following analysis features a Stratix II FPGA interfacing with two 267-MHz DDR2 SDRAM UDIMMs. This DDR2 SDRAM interface is built on the Stratix II High-Speed High-Density Board (Figure 3–2).

For more information about the Stratix II High-Speed High-Density Board, contact your Altera representative.

**Figure 3–2. Stratix II High-Speed Board with Dual-DIMM DDR2 SDRAM Interface**



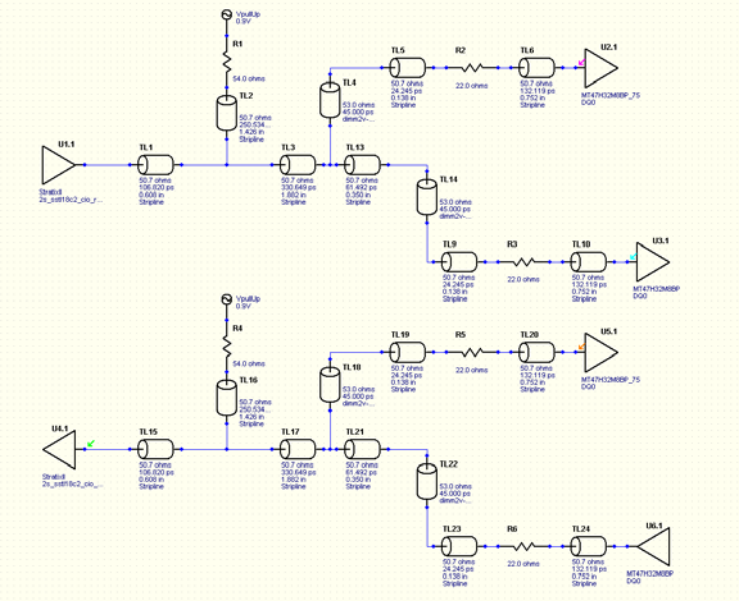

The Stratix II High-Speed Board uses a Stratix II 2S90F1508 device. For DQS, DQ, and DM signals, the board is designed without external parallel termination resistors near the DDR2 SDRAM DIMMs, to take advantage of the ODT feature of the DDR2 SDRAM components. Stratix II FPGA devices are not equipped with dynamic OCT, so external parallel termination resistors are used at the FPGA end of the line.

Stratix III and Stratix IV devices, which support dynamic OCT, do not require FPGA end parallel termination. Hence this discrete parallel termination is optional.

The DDR2 SDRAM DIMM contains a 22-Ωexternal series termination resistor for each data strobe and data line, so all the measurements and simulations need to account for the effect of these series termination resistors.

To correlate the bench measurements done on the Stratix II High Speed High Density Board, the simulations are performed using HyperLynx LineSim Software with IBIS models from Altera and memory vendors. Figure 3–3 is an example of the simulation setup in HyperLynx used for the simulation.

**Figure 3–3. HyperLynx Setup for Simulating the Stratix II High Speed High Density with Dual-DIMM DDR2 SDRAM Interface**



## Overview of ODT Control

When there is only a single-DIMM on the board, the ODT control is relatively straightforward. During write to the memory, the ODT feature of the memory is turned on; during read from the memory, the ODT feature of the memory is turned off. However, when there are multiple DIMMs on the board, the ODT control becomes more complicated.

With a dual-DIMM interface on the system, the controller has different options for turning the memory ODT on or off during read or write. Table 3–1 shows the DDR2 SDRAM ODT control during write to the memory; Table 3–2 during read from the memory. These DDR2 SDRAM ODT controls are recommended by Samsung Electronics. The JEDEC DDR2 specification was updated to include optional support for $R_{TT}$(nominal) = 50 Ω

For more information about the DDR2 SDRAM ODT controls recommended by Samsung, refer to the *Samsung DDR2 Application Note: ODT (On Die Termination) Control*.

**Table 3–1. DDR2 SDRAM ODT Control—Writes *(Note 1)***

| Slot 1 *(2)* | Slot 2 *(2)* | Write To | FPGA | Module in Slot 1 | | Module in Slot 2 | |
|---|---|---|---|---|---|---|---|
| | | | | Rank 1 | Rank 2 | Rank 3 | Rank 4 |
| DR | DR | Slot 1 | Series 50 Ω | Infinite | Infinite | 75 or 50 Ω | Infinite |
| | | Slot 2 | Series 50 Ω | 75 or 50 Ω | Infinite | Infinite | infinite |
| SR | SR | Slot 1 | Series 50 Ω | Infinite | Unpopulated | 75 or 50 Ω | Unpopulated |
| | | Slot 2 | Series 50 Ω | 75 or 50 Ω | Unpopulated | Infinite | Unpopulated |
| DR | Empty | Slot 1 | Series 50 Ω | 150 Ω | Infinite | Unpopulated | Unpopulated |
| Empty | DR | Slot 2 | Series 50 Ω | Unpopulated | Unpopulated | 150 Ω | Infinite |
| SR | Empty | Slot 1 | Series 50 Ω | 150 Ω | Unpopulated | Unpopulated | Unpopulated |
| Empty | SR | Slot 2 | Series 50 Ω | Unpopulated | Unpopulated | 150 Ω | Unpopulated |

**Note to Table 3–1:**

(1) For DDR2 at 400 MHz and 533 Mbps = 75 Ω; for DDR2 at 667 MHz and 800 Mbps = 50 Ω

(2) SR = single ranked; DR = dual ranked.

**Table 3–2. DDR2 SDRAM ODT Control—Reads *(Note 1)***

| Slot 1 *(2)* | Slot 2 *(2)* | Read From | FPGA | Module in Slot 1 | | Module in Slot 2 | |
|---|---|---|---|---|---|---|---|
| | | | | Rank 1 | Rank 2 | Rank 3 | Rank 4 |
| DR | DR | Slot 1 | Parallel 50 Ω | Infinite | Infinite | 75 or 50 Ω | Infinite |
| | | Slot 2 | Parallel 50 Ω | 75 or 50 Ω | Infinite | Infinite | Infinite |
| SR | SR | Slot 1 | Parallel 50 Ω | Infinite | Unpopulated | 75 or 50 Ω | Unpopulated |
| | | Slot 2 | Parallel 50 Ω | 75 or 50 Ω | Unpopulated | Infinite | Unpopulated |
| DR | Empty | Slot 1 | Parallel 50 Ω | Infinite | Infinite | Unpopulated | Unpopulated |
| Empty | DR | Slot 2 | Parallel 50 Ω | Unpopulated | Unpopulated | Infinite | Infinite |
| SR | Empty | Slot 1 | Parallel 50 Ω | Infinite | Unpopulated | Unpopulated | Unpopulated |
| Empty | SR | Slot 2 | Parallel 50 Ω | Unpopulated | Unpopulated | Infinite | Unpopulated |

**Note to Table 3–1:**

(1) For DDR2 at 400 MHz and 533 Mbps = 75 Ω; for DDR2 at 667 MHz and 800 Mbps = 50 Ω

(2) SR = single ranked; DR = dual ranked.

A 54-Ω external parallel termination resistor is placed on all the data strobes and data lines near the Stratix II device on the Stratix II High Speed High Density Board. Although the characteristic impedance of the transmission is designed for 50 Ω, to account for any process variation, it is advisable to underterminate the termination seen at the receiver. This is why the termination resistors at the FPGA side use 54-Ω resistors.

## DIMM Configuration

While populating both memory slots is common in a dual-DIMM memory system, there are some instances when only one slot is populated. For example, some systems are designed to have a certain amount of memory initially and as applications get more complex, the system can be easily upgraded to accommodate more memory by populating the second memory slot without re-designing the system. The following section discusses a dual-DIMM system where the dual-DIMM system only has one slot populated at one time and a dual-DIMM system where both slots are populated. ODT controls recommended by the memory vendors listed in Table 3–1 as well as other possible ODT settings will be evaluated for usefulness in an FPGA system.

## Dual-DIMM Memory Interface with Slot 1 Populated

This section focuses on a dual-DIMM memory interface where slot 1 is populated and slot 2 is unpopulated. This section examines the impact on the signal quality due to an unpopulated DIMM slot and compares it to a single-DIMM memory interface.

### FPGA Writing to Memory

In the DDR2 SDRAM, the ODT feature has two settings: 150 Ω and 75 Ω In Table 3–1, the recommended ODT setting for a dual DIMM configuration with one slot occupied is 150 Ω

On DDR2 SDRAM devices running at 333 MHz/667 Mbps and above, the ODT feature supports an additional setting of 50 Ω

Refer to the respective memory decathlete for additional information about the ODT settings in DDR2 SDRAM devices.

### Write to Memory Using an ODT Setting of 150Ω

Figure 3–4 shows a double parallel termination scheme (Class II) using ODT on the memory with a memory-side series resistor when the FPGA is writing to the memory using a 25-Ω OCT drive strength setting on the FPGA.

**Figure 3–4. Double Parallel Termination Scheme (Class II) Using ODT on DDR2 SDRAM DIMM with Memory-Side Series Resistor**

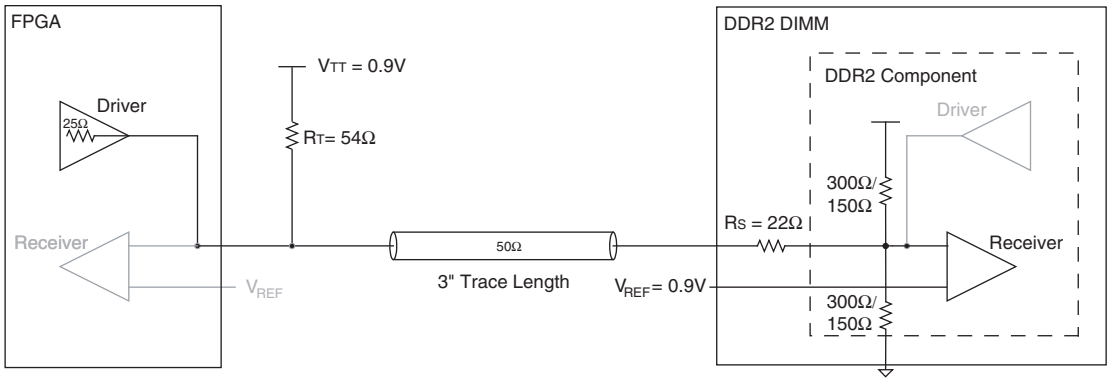


Figure 3–5 shows a HyperLynx simulation and board measurement of a signal at the memory of a double parallel termination using ODT 150 Ω with a memory-side series resistor transmission line when the FPGA is writing to the memory with a 25-Ω OCT drive strength setting.

**Figure 3–5. HyperLynx Simulation and Board Measurement of the Signal at the Memory in Slot 1 with Slot 2 Unpopulated**

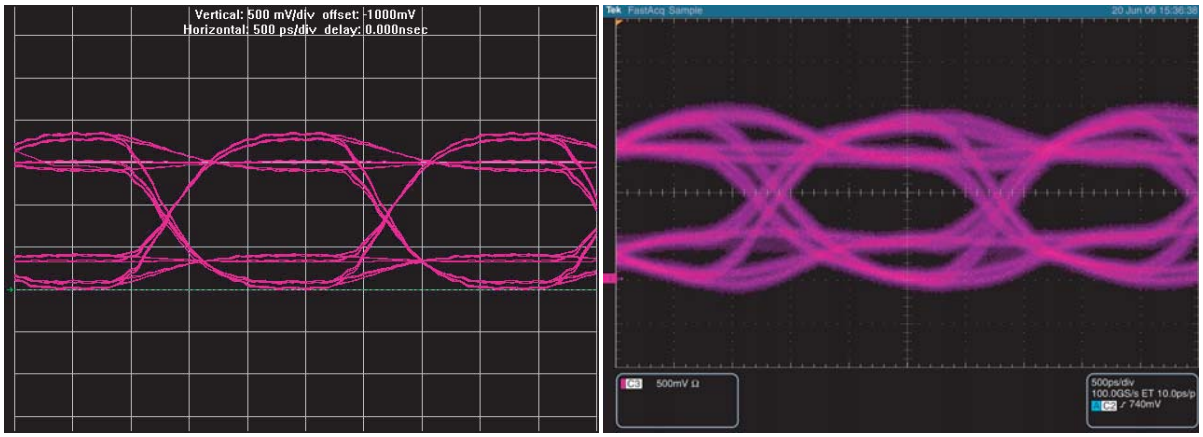



Table 3–3 summarizes the comparison between the simulation and board measurements of the signal at the memory of a single-DIMM and a dual-DIMM memory interface with slot 1 populated using a double parallel termination using an ODT setting of 150 Ω with a memory-side series resistor with a 25-Ω OCT strength setting on the FPGA.

**Table 3–3. Comparison of Signal at the Memory of a Single-DIMM and a Dual-DIMM Interface with Slot 1 Populated *(Note 1)***

| Type | Eye Width (ns) | Eye Height (V) | Overshoot (V) | Undershoot (V) | Rising Edge Rate (V/ns) | Falling Edge Rate (V/ns) |
|---|---|---|---|---|---|---|
| **Dual-DIMM memory interface with slot 1 populated** | | | | | | |
| Simulation | 1.68 | 0.97 | 0.06 | NA | 2.08 | 1.96 |
| Measurements | 1.30 | 0.63 | 0.22 | 0.20 | 1.74 | 1.82 |
| **Single-DIMM** | | | | | | |
| Simulation | 1.62 | 0.94 | 0.10 | 0.05 | 2.46 | 2.46 |
| Measurements | 1.34 | 0.77 | 0.04 | 0.13 | 1.56 | 1.39 |

**Note to Table 3–3:**

(1) The simulation and board measurements of the single-DIMM DDR2 SDRAM interface are based on the Stratix II Memory Board 2. For more information about the single-DIMM DDR2 SDRAM interface, refer to Chapter 1, DDR2 SDRAM Interface Termination, Drive Strength, Loading, and Board Layout Guidelines.

Table 3–3 indicates that there is not much difference between a single-DIMM memory interface or a dual-DIMM memory interface with slot 1 populated. The over and undershooting observed in both the simulations and board measurements can be attributed to the use of the ODT setting of 150 Ω on the memory resulting in over-termination at the receiver. In addition, there is no significant effect of the extra DIMM connector due to the unpopulated slot.

When the ODT setting is set to 75 Ω, there is no difference in the eye width and height compared to the ODT setting of 150 Ω However, there is no overshoot and undershoot when the ODT setting is set to 75 Ω, which is attributed to proper termination resulting in matched impedance seen by the DDR2 SDRAM devices.

For information about results obtained from using an ODT setting of 75 Ω refer to page 3–25.

## Reading from Memory

During read from the memory, the ODT feature is turned off. Thus, there is no difference between using an ODT setting of 150 Ω and 75 Ω As such, the termination scheme becomes a single parallel termination scheme (Class I) where there is an external resistor on the FPGA side and a series resistor on the memory side as shown in Figure 3–6.

**Figure 3–6. Single Parallel Termination Scheme (Class I) Using External Resistor and Memory-Side Series Resistor**

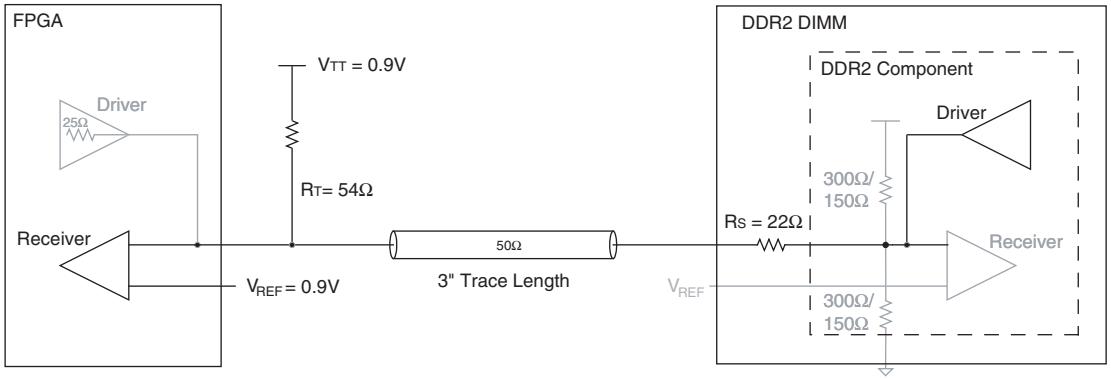



Figure 3–7 shows the simulation and board measurement of the signal at the FPGA of a single parallel termination using an external parallel resistor on the FPGA side with a memory-side series resistor with full drive strength setting on the memory.

**Figure 3–7. HyperLynx Simulation and Board Measurement of the Signal at the FPGA When Reading From Slot 1 With Slot 2 Unpopulated**

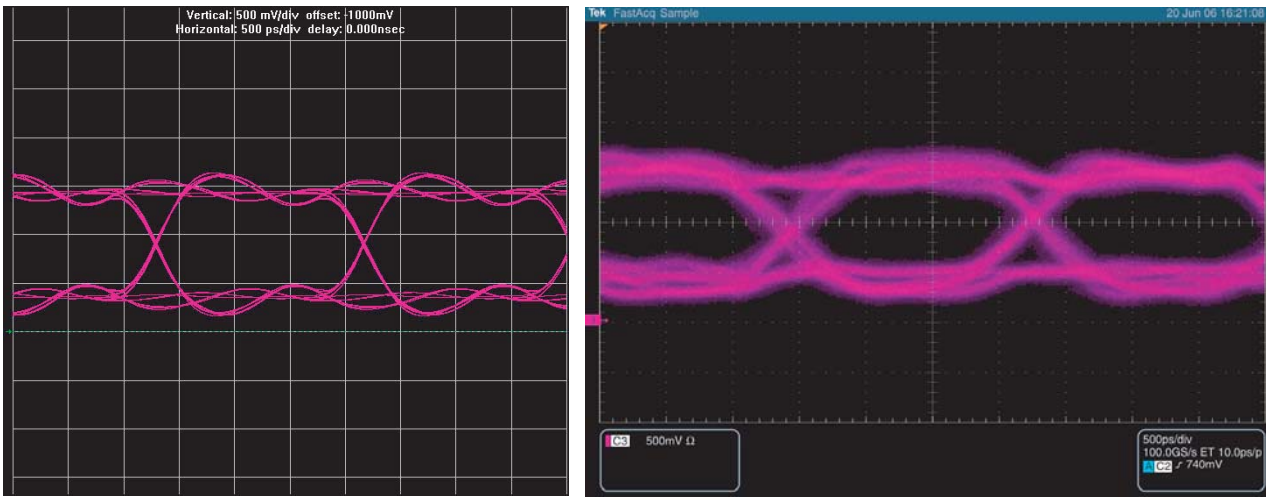


Table 3–4 summarizes the comparison between the simulation and board measurements of the signal seen at the FPGA of a single-DIMM and a dual-DIMM memory interface with a slot 1 populated memory interface using a single parallel termination using an external parallel resistor at the FPGA with a memory-side series resistor with full strength setting on the memory.

**Table 3–4. Comparison of Signal at the FPGA of a Dual-DIMM Memory Interface with Slot 1 Populated** ***(Note 1)***

| Type | Eye Width (ns) | Eye Height (V) | Overshoot (V) | Undershoot (V) | Rising Edge Rate (V/ns) | Falling Edge Rate (V/ns) |
|---|---|---|---|---|---|---|
| **Dual-DIMM memory interface with slot 1 populated** | | | | | | |
| Simulation | 1.76 | 0.80 | NA | NA | 2.29 | 2.29 |
| Measurements | 1.08 | 0.59 | NA | NA | 1.14 | 1.59 |
| **Single-DIMM[1]** | | | | | | |
| Simulation | 1.80 | 0.95 | NA | NA | 2.67 | 2.46 |
| Measurements | 1.03 | 0.58 | NA | NA | 1.10 | 1.30 |

**Note to Table 3–4:**

(1) The simulation and board measurements of the single-DIMM DDR2 SDRAM interface are based on the Stratix II Memory Board 2. For more information about the single-DIMM DDR2 SDRAM interface, refer to Chapter 1, DDR2 SDRAM Interface Termination, Drive Strength, Loading, and Board Layout Guidelines,

Table 3–4 demonstrates that there is not much difference between a single-DIMM memory interface or a dual-DIMM memory interface with only slot 1 populated. There is no significant effect of the extra DIMM connector due to the unpopulated slot.

## Dual-DIMM with Slot 2 Populated

This section focuses on a dual-DIMM memory interface where slot 2 is populated and slot 1 is unpopulated. Specifically, this section discusses the impact of location of the DIMM on the signal quality.

### FPGA Writing to Memory

The previous section focused on the dual-DIMM memory interface where slot 1 is populated resulting in the memory being located closer to the FPGA. When slot 2 is populated, the memory is located further away from the FPGA, resulting in additional trace length that potentially affects the signal quality seen by the memory. The next section explores if there are any differences between populating slot 1 and slot 2 of the dual-DIMM memory interface.

## Write to Memory Using an ODT Setting of 150Ω

Figure 3–8 shows the double parallel termination scheme (Class II) using ODT on the memory with the memory-side series resistor when the FPGA is writing to the memory using a 25-Ω OCT drive strength setting on the FPGA.

**Figure 3–8. Double Parallel Termination Scheme (Class II) Using ODT on DDR2 SDRAM DIMM with Memory-side Series Resistor**

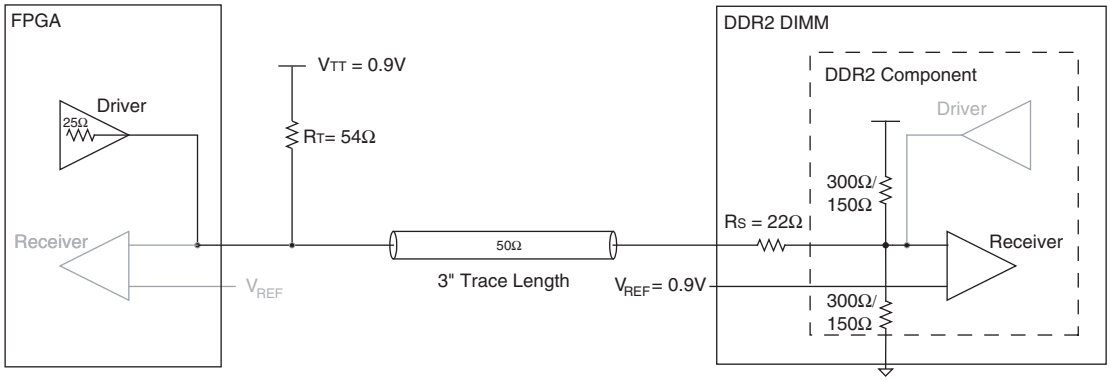



Figure 3–9 shows the simulation and board measurement of the signal at the memory of a double parallel termination using an ODT setting of 150 Ω with a memory-side series resistor transmission line when the FPGA is writing to the memory with a 25-Ω OCT drive strength setting.

**Figure 3–9. HyperLynx Simulation and Board Measurement of the Signal at the Memory in Slot 2 With Slot 1 Unpopulated**

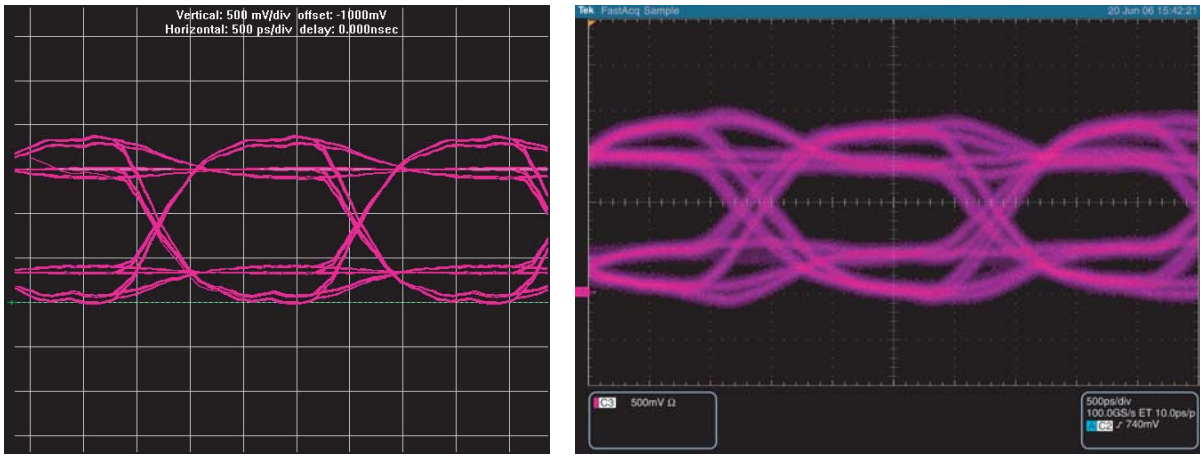


Table 3–5 summarizes the comparison between the simulation and board measurements of the signal seen at the DDR2 SDRAM DIMM of a dual-DIMM memory interface with either only slot 1 populated or only slot 2 populated using a double parallel termination using an ODT setting of 150 Ω with a memory-side series resistor with a 25-Ω OCT strength setting on the FPGA.

**Table 3–5. Comparison of Signal at the Memory of a Dual-DIMM Interface with Either Only Slot 1 Populated or Only Slot 2 Populated**

| Type | Eye Width (ns) | Eye Height (V) | Overshoot (V) | Undershoot (V) | Rising Edge Rate (V/ns) | Falling Edge Rate (V/ns) |
|---|---|---|---|---|---|---|
| **Dual-DIMM Memory Interface with Slot 2 Populated** | | | | | | |
| Simulation | 1.69 | 0.94 | 0.07 | 0.02 | 1.96 | 2.08 |
| Measurements | 1.28 | 0.68 | 0.24 | 0.20 | 1.60 | 1.60 |
| **Dual-DIMM Memory Interface with Slot 1 Populated** | | | | | | |
| Simulation | 1.68 | 0.97 | 0.06 | NA | 2.08 | 2.08 |
| Measurements | 1.30 | 0.63 | 0.22 | 0.20 | 1.74 | 1.82 |

Table 3–5 shows that there is not much difference between populating slot 1 or slot 2 in a dual-DIMM memory interface. The over and undershooting observed in both the simulations and board measurements can be attributed to the use of the ODT setting of 150 Ω on the memory, resulting in under-termination at the receiver.

When the ODT setting is set to 75 Ω, there is no difference in the eye width and height compared to the ODT setting of 150 Ω However, there is no overshoot and undershoot when the ODT setting is set to 75 Ω, which is attributed to proper termination resulting in matched impedance seen by the DDR2 SDRAM devices.

For detailed results for the ODT setting of 75 Ω, refer to page 3–26.

## Reading from Memory

During read from memory, the ODT feature is turned off, thus there is no difference between using an ODT setting of 150 Ω and 75 Ω As such, the termination scheme becomes a single parallel termination scheme (Class I) where there is an external resistor on the FPGA side and a series resistor on the memory side, as shown in Figure 3–10.

**Figure 3–10. Single Parallel Termination Scheme (Class I) Using External Resistor and Memory-Side Series Resistor**

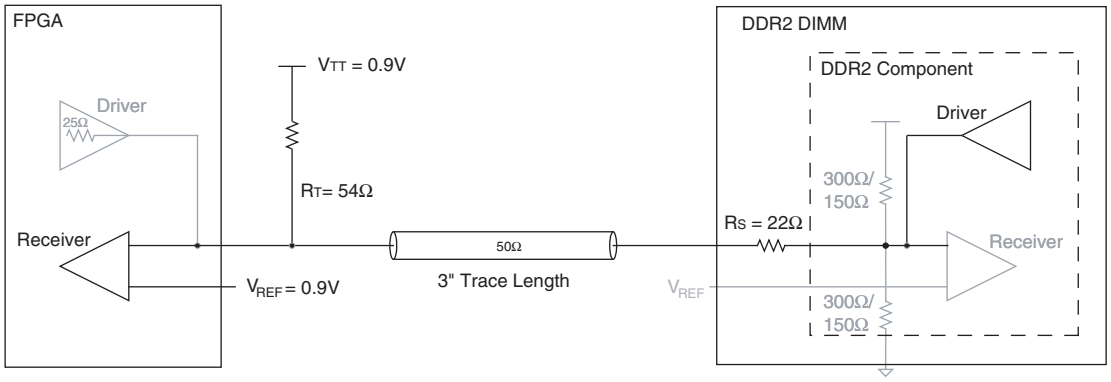


Figure 3–11 shows the simulation and board measurement of the signal at the FPGA of a single parallel termination using an external parallel resistor on the FPGA side with a memory-side series resistor with full drive strength setting on the memory.

**Figure 3–11. HyperLynx Simulation and Board Measurement of the Signal at the FPGA When Reading From Slot 2 With Slot 1 Unpopulated**

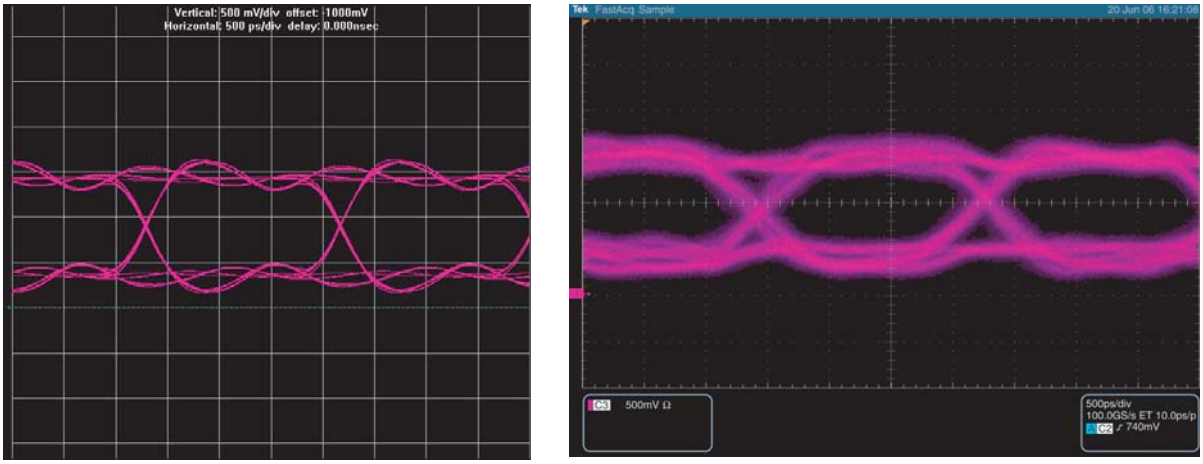



Table 3–6 summarizes the comparison between the simulation and board measurements of the signal seen at the FPGA of a dual-DIMM memory interface with either slot 1 or slot 2 populated using a single parallel termination using an external parallel resistor at the FPGA with a memory-side series resistor with full strength setting on the memory.

**Table 3–6. Comparison of the Signal at the FPGA of a Dual-DIMM Memory Interface with Either Slot 1 or Slot 2 Populated**

| Type | Eye Width (ns) | Eye Height (V) | Overshoot (V) | Undershoot (V) | Rising Edge Rate (V/ns) | Falling Edge Rate (V/ns) |
|---|---|---|---|---|---|---|
| **Slot 2 Populated** | | | | | | |
| Simulation | 1.80 | 0.80 | NA | NA | 3.09 | 2.57 |
| Measurements | 1.17 | 0.66 | NA | NA | 1.25 | 1.54 |
| **Slot 1 Populated** | | | | | | |
| Simulation | 1.80 | 0.95 | NA | NA | 2.67 | 2.46 |
| Measurements | 1.08 | 0.59 | NA | NA | 1.14 | 1.59 |

From Table 3–6, you can see the signal seen at the FPGA is similar whether the memory DIMM is located at either slot 1 or slot 2.

## Dual-DIMM Memory Interface with Both Slot 1 and Slot 2 Populated

This section focuses on a dual-DIMM memory interface where both slot 1 and slot 2 are populated. As such, you can write to either the memory in slot 1 or the memory in slot 2.

## FPGA Writing to Memory

In Table 3–1, the recommended ODT setting for a dual DIMM configuration with both slots occupied is 75 Ω Since there is an option for an ODT setting of 150 Ω, this section explores the usage of the 150 Ω setting and compares the results to that of the recommended 75 Ω

## Write to Memory in Slot 1 Using an ODT Setting of 75-Ω

Figure 3–12 shows the double parallel termination scheme (Class II) using ODT on the memory with the memory-side series resistor when the FPGA is writing to the memory using a 25-Ω OCT drive strength setting on the FPGA. In this scenario, the FPGA is writing to the memory in slot 1 and the ODT feature of the memory at slot 2 is turned on.

**Figure 3–12. Double Parallel Termination Scheme (Class II) Using ODT on DDR2 SDRAM DIMM with a Memory-Side Series Resistor**

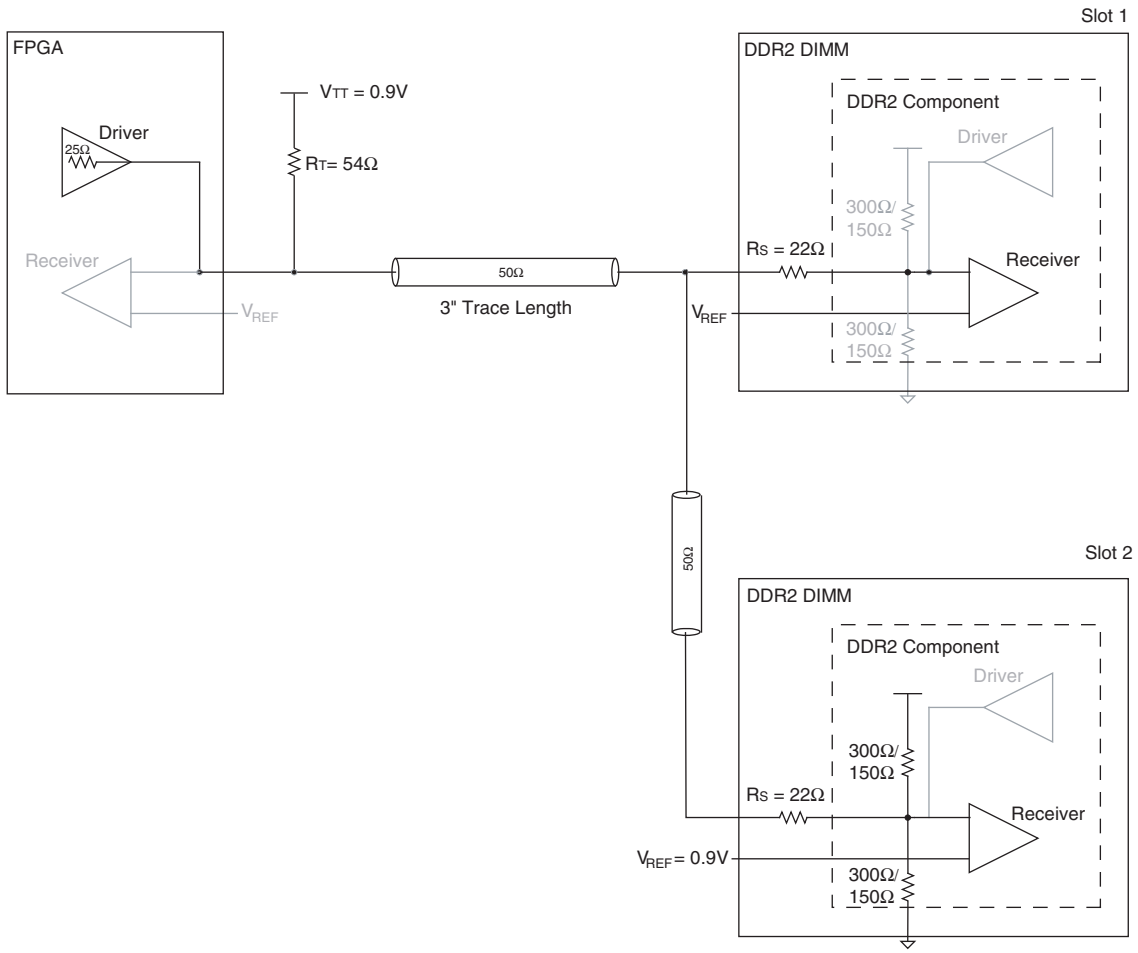


Figure 3–13 shows a HyperLynx simulation and board measurement of the signal at the memory in slot 1 of a double parallel termination using an ODT setting of 75 Ω with a memory-side series resistor transmission line when the FPGA is writing to the memory with a 25-Ω OCT drive strength setting.

**Figure 3–13. HyperLynx Simulation and Board Measurements of the Signal at the Memory in Slot 1 with Both Slots Populated**

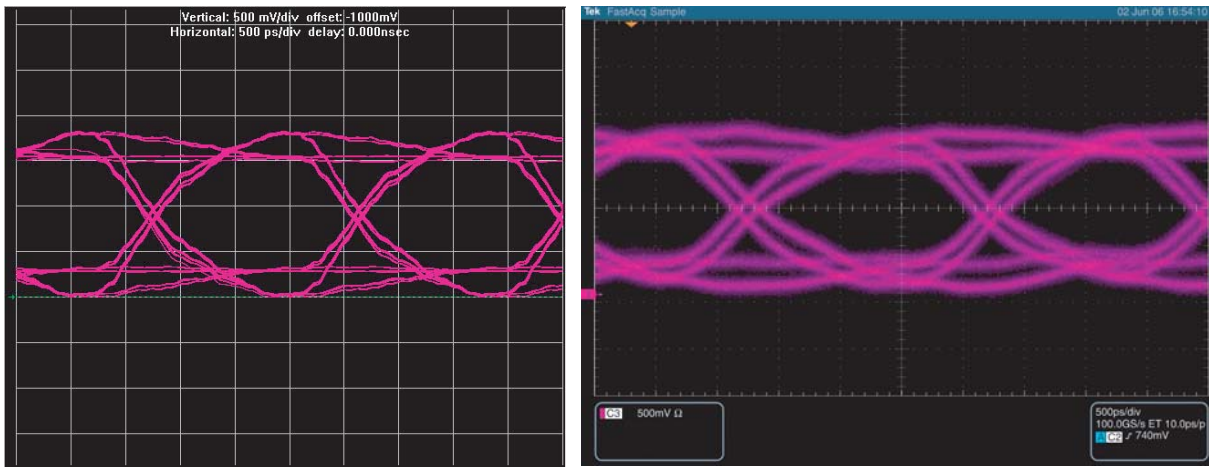



Table 3–7 summarizes the comparison of the signal at the memory of a dual-DIMM memory interface with one slot and with both slots populated using a double parallel termination using an ODT setting of 75 Ω with a memory-side series resistor with a 25-Ω OCT strength setting on the FPGA.

**Table 3–7. Comparison of the Signal at the Memory of a Dual-DIMM Interface With One Slot and With Both Slots Populated**

| Type | Eye Width (ns) | Eye Height (V) | Overshoot (V) | Undershoot (V) | Rising Edge Rate (V/ns) | Falling Edge Rate (V/ns) |
|---|---|---|---|---|---|---|
| **Dual-DIMM Interface with Both Slots Populated Writing to Slot 1** | | | | | | |
| Simulation | 1.60 | 1.18 | 0.02 | NA | 1.71 | 1.71 |
| Measurements | 0.97 | 0.77 | 0.05 | 0.04 | 1.25 | 1.25 |
| **Dual-DIMM Interface with Slot 1 Populated** | | | | | | |
| Simulation | 1.68 | 0.97 | 0.06 | NA | 2.08 | 2.08 |
| Measurements | 1.30 | 0.63 | 0.22 | 0.20 | 1.74 | 1.82 |

Table 3–7 shows that there is not much difference in the eye height between populating one slot or both slots. However, the additional loading due to the additional memory DIMM results in a slower edge rate, which results in smaller eye width and degrades the setup and hold time of the memory. This reduces the available data valid window.

When the ODT setting is set to 150 Ω, there is no difference in the eye width and height compared to the ODT setting of 75 Ω However, there is some overshoot and undershoot when the ODT setting is set to 150 Ω, which is attributed to under termination resulting in mismatched impedance seen by the DDR2 SDRAM devices.

For more information about the results obtained from using an ODT setting of 150 Ω, refer to page 3–27.

### Write to Memory in Slot 2 Using an ODT Setting of 75-Ω

In this scenario, the FPGA is writing to the memory in slot 2 and the ODT feature of the memory at slot 1 is turned on. Figure 3–14 shows the HyperLynx simulation and board measurement of the signal at the memory in slot 1 of a double parallel termination using an ODT setting of 75 Ω with a memory-side series resistor transmission line when the FPGA is writing to the memory with a 25-Ω OCT drive strength setting.

**Figure 3–14. HyperLynx Simulation and Board Measurements of the Signal at the Memory in Slot 2 With Both Slots Populated**

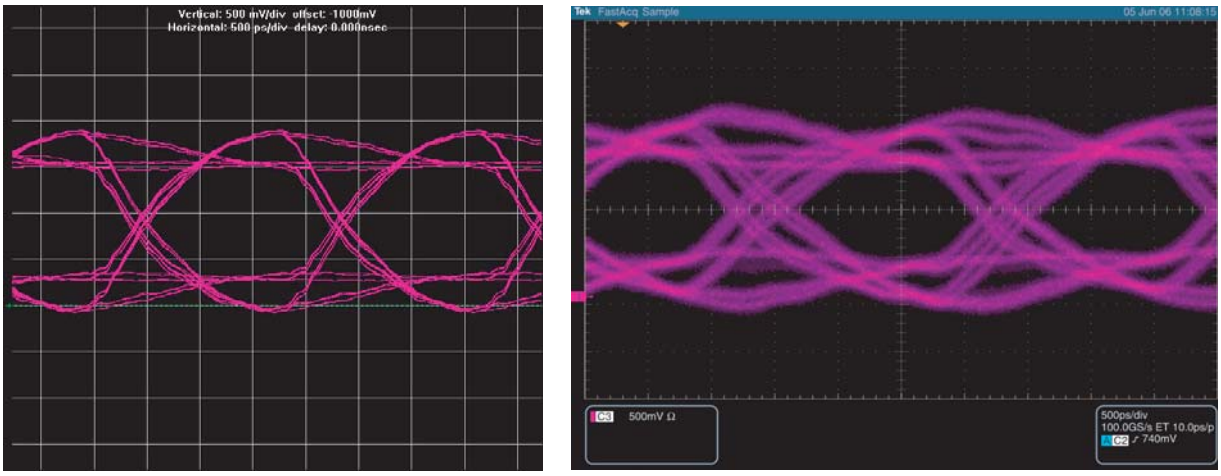



Table 3–8 summarizes the comparison of the signal at the memory of a dual-DIMM memory interface with slot 1 populated using a double parallel termination using an ODT setting of 75 Ω with a memory-side series resistor with a 25-Ω OCT strength setting on the FPGA.

**Table 3–8. Comparison of the Signal at the Memory of a Dual-DIMM Interface With Both Slots Populated**

| Type | Eye Width (ns) | Eye Height (V) | Overshoot (V) | Undershoot (V) | Rise Edge Rate (V/ns) | Falling Edge Rate (V/ns) |
|---|---|---|---|---|---|---|
| **Dual-DIMM Interface with Both Slots Populated Writing to Slot 2** | | | | | | |
| Simulation | 1.60 | 1.16 | 0.10 | 0.08 | 1.68 | 1.60 |
| Measurements | 1.10 | 0.85 | 0.16 | 0.19 | 1.11 | 1.25 |
| **Dual-DIMM Interface with Both Slots Populated Writing to Slot 1** | | | | | | |
| Simulation | 1.60 | 1.18 | 0.02 | NA | 1.71 | 1.71 |
| Measurements | 1.30 | 0.77 | 0.05 | 0.04 | 1.25 | 1.25 |

From Table 3–8, you can see that both simulations and board measurements demonstrate that the eye width is larger when writing to slot 1, which is due to better edge rate seen when writing to slot 1. The improvement on the eye when writing to slot 1 can be attributed to the location of the termination. When you are writing to slot 1, the ODT feature of slot 2 is turned on, resulting in a fly-by topology. When you are writing to slot 2, the ODT feature of slot 1 is turned on resulting in a non fly-by topology.

When the ODT setting is set to 150 Ω there is no difference in the eye width and height compared to the ODT setting of 75 Ω However, there is some overshoot and undershoot when the ODT setting is set to 150 Ω which is attributed to under termination resulting in mismatched impedance seen by the DDR2 SDRAM devices.

For more information about the results obtained from using an ODT setting of 150 Ω refer to "Write to Memory in Slot 2 Using an ODT Setting of 150 W With Both Slots Populated" on page 3–28.

## Reading From Memory

In Table 3–2, the recommended ODT setting for a dual-DIMM configuration with both slots occupied is to turn on the ODT feature using a setting of 75 Ω on the slot that is not read from. As there is an option for an ODT setting of 150 Ω this section explores the usage of the 150 Ω setting and compares the results to that of the recommended 75 Ω.

### Read From Memory in Slot 1 Using an ODT Setting of 75-Ω on Slot 2

Figure 3–15 shows the double parallel termination scheme (Class II) using ODT on the memory with the memory-side series resistor when the FPGA is reading from the memory using a full drive strength setting on the memory. In this scenario, the FPGA is reading from the memory in slot 1 and the ODT feature of the memory at slot 2 is turned on.

**Figure 3–15. Double Parallel Termination Scheme (Class II) Using External Resistor and Memory-Side Series Resistor and ODT Feature Turned On**

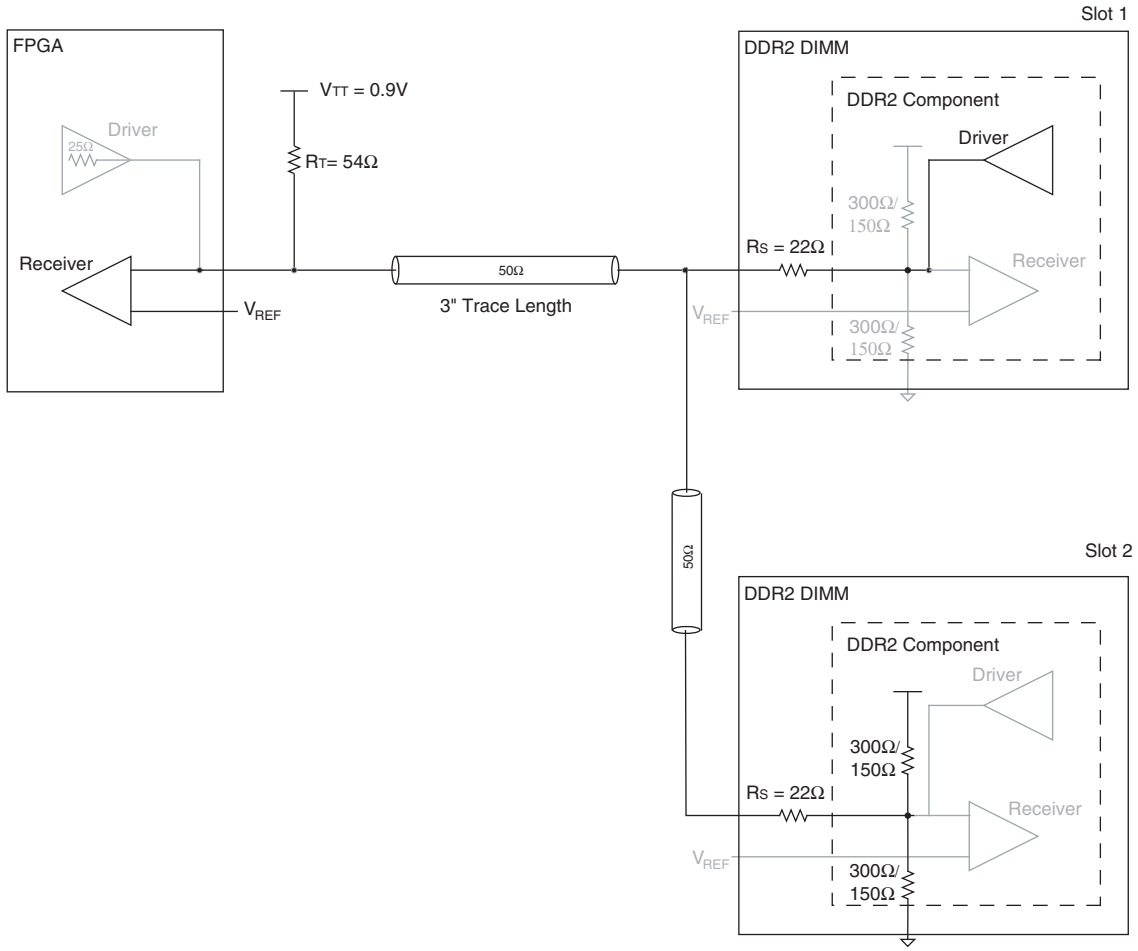
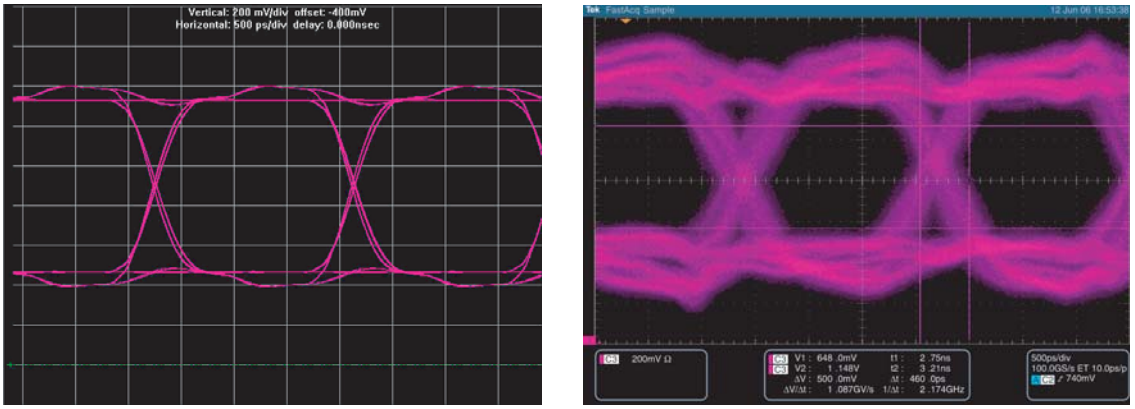


Figure 3–16 shows the simulation and board measurement of the signal at the FPGA when the FPGA is reading from the memory in slot 1 using a full drive strength setting on the memory.

**Figure 3–16. HyperLynx Simulation and Board Measurement of the Signal at the FPGA When Reading From Slot 1 With Both Slots Populated** ***(Note 1)***




**Note to Figure 3–16:**

(1) The vertical scale used for the simulation and measurement is set to 200 mV per division.

Table 3–9 summarizes the comparison between the simulation and board measurements of the signal seen at the FPGA of a dual-DIMM memory interface with both slots populated and a dual-DIMM memory interface with a slot 1 populated memory interface.

**Table 3–9. Comparison of the Signal at the FPGA of a Dual-DIMM Interface Reading From Slot 1 With One Slot and With Both Slots Populated**

| Type | Eye Width (ns) | Eye Height (V) | Overshoot (V) | Undershoot (V) | Rising Edge Rate (V/ns) | Falling Edge Rate (V/ns) |
|---|---|---|---|---|---|---|
| **Dual-DIMM with One Slot Populated with an ODT Setting of 75-Ω on Slot 2** | | | | | | |
| Simulation | 1.74 | 0.87 | NA | NA | 1.91 | 1.88 |
| Measurements | 0.86 | 0.58 | NA | NA | 1.11 | 1.09 |
| **Dual-DIMM with One Slot Populated in Slot 1 without ODT Setting** | | | | | | |
| Simulation | 1.76 | 0.80 | NA | NA | 2.29 | 2.29 |
| Measurements | 1.08 | 0.59 | NA | NA | 1.14 | 1.59 |

Table 3–9 shows that when both slots are populated, the additional loading due to the additional memory DIMM results in a slower edge rate, which results in a degradation in the eye width.

For more information about the results obtained from using an ODT setting of 150 Ω, refer to "Read from Memory in Slot 1 Using an ODT Setting of 150 W on Slot 2 with Both Slots Populated" on page 3–29.

### Read From Memory in Slot 2 Using an ODT Setting of 75-Ω on Slot 1

In this scenario, the FPGA is reading from the memory in slot 2 and the ODT feature of the memory at slot 1 is turned on.

**Figure 3–17. Double Parallel Termination Scheme (Class II) Using External Resistor and a Memory-Side Series Resistor and ODT Feature Turned On**

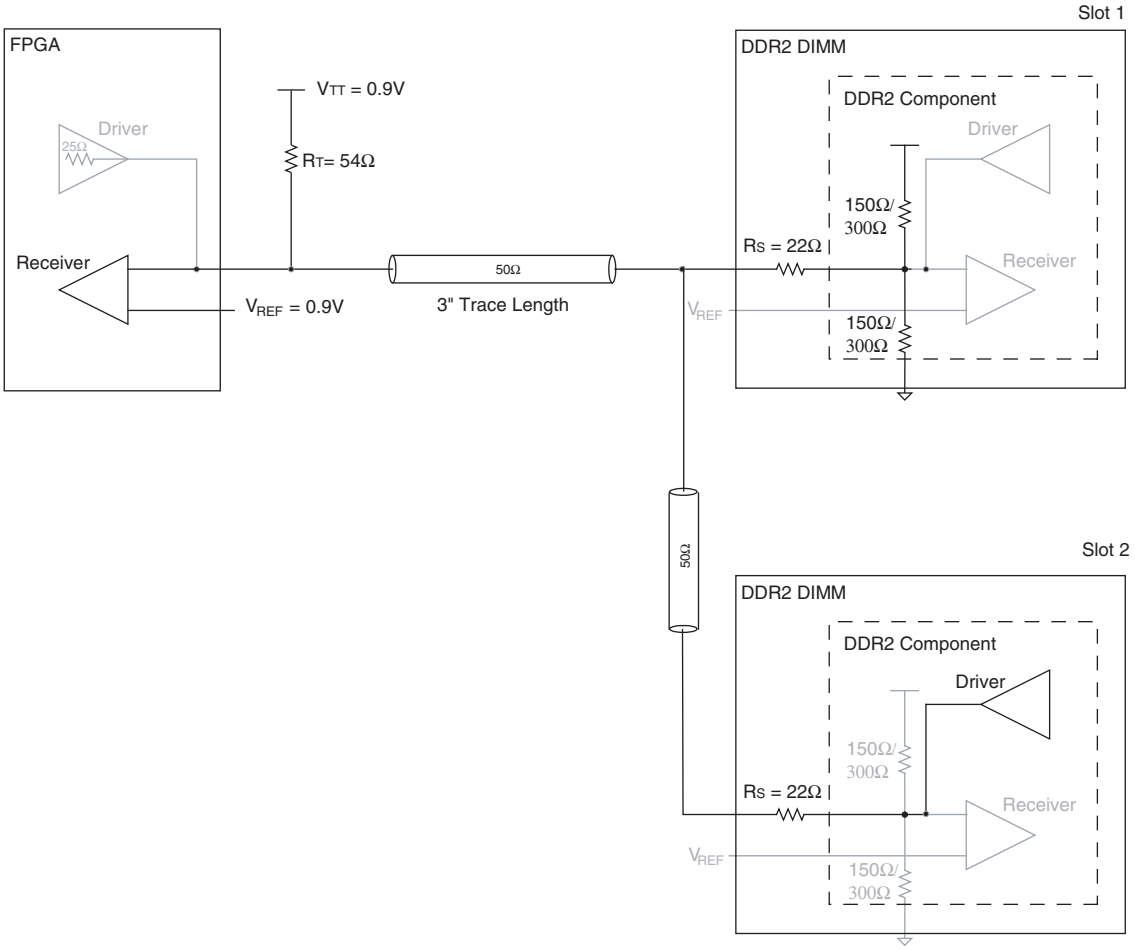
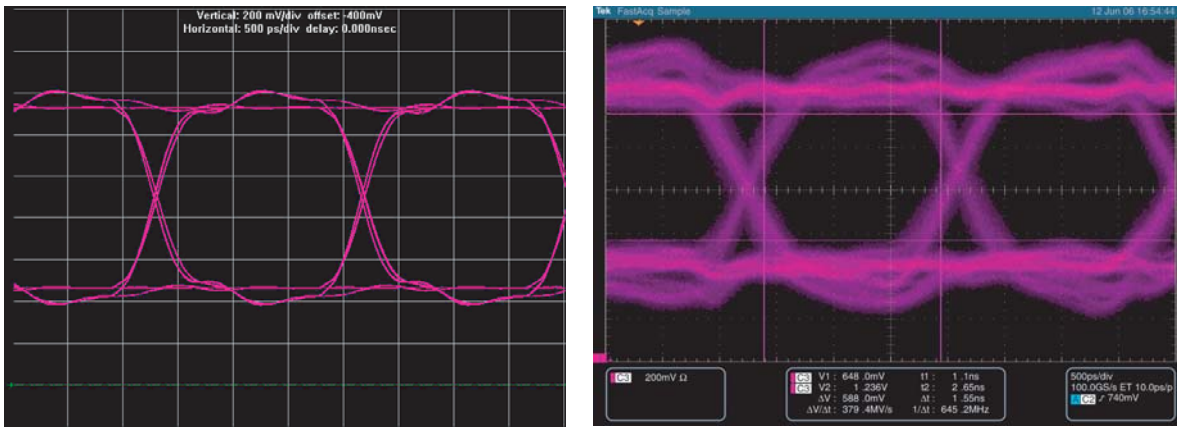


Figure 3–18 shows the HyperLynx simulation and board measurement of the signal at the FPGA of a double parallel termination using an external parallel resistor on the FPGA side with a memory-side series resistor and an ODT setting of 75 Ω with a full drive strength setting on the memory.

**Figure 3–18. HyperLynx Simulation and Board Measurements of the Signal at the FPGA When Reading From Slot 2 With Both Slots Populated *(Note 1)***




**Notes to Figure 3–18:**

(1) The vertical scale used for the simulation and measurement is set to 200 mV per division.

Table 3–10 summarizes the comparison between the simulation and board measurements of the signal seen at the FPGA of a dual-DIMM memory interface with both slots populated and a dual-DIMM memory interface with a slot 1 populated memory interface.

**Table 3–10. Comparison of the Signal at the FPGA of a Dual-DIMM Interface Reading From Slot 2 With One Slot and With Both Slots Populated**

| Type | Eye Width (ns) | Eye Height (V) | Overshoot (V) | Undershoot (V) | Rising Edge Rate (V/ns) | Falling Edge Rate (V/ns) |
|---|---|---|---|---|---|---|
| **Dual-DIMM with Both Slots Populated with an ODT Setting of 75-Ω Setting on Slot 1** | | | | | | |
| Simulation | 1.70 | 0.81 | NA | NA | 1.72 | 1.99 |
| Measurements | 0.87 | 0.59 | NA | NA | 1.09 | 1.14 |
| **Dual-DIMM with One Slot Populated in Slot 2 without an ODT Setting** | | | | | | |
| Simulation | 1.80 | 0.80 | NA | NA | 3.09 | 2.57 |
| Measurements | 1.17 | 0.66 | NA | NA | 1.25 | 1.54 |

Table 3–10 shows that when only one slot is populated in a dual-DIMM memory interface, the eye width is larger as compared to a dual-DIMM memory interface with both slots populated. This can be attributed to the loading from the DIMM located in slot 1.

When the ODT setting is set to 150 Ω, there is no difference in the signal quality compared to the ODT setting of 75 Ω

For more information about the results obtained from using an ODT setting of 150 Ω, refer to "Read From Memory in Slot 2 Using an ODT Setting of 150 W on Slot 1 With Both Slots Populated" on page 3–30.

## FPGA OCT Features

Many FPGA devices offer OCT. Depending on the chosen device family, series (output), parallel (input) or dynamic (bidirectional) OCT may be supported.

For more information specific to your device family, refer to the respective I/O features chapter in the relevant device handbook.

Use series OCT in place of the near-end series terminator typically used in both Class I or Class II termination schemes that both DDR2 and DDR3 type interfaces use.

Use parallel OCT in place of the far-end parallel termination typically used in Class I termination schemes on unidirectional input only interfaces. For example, QDR-II type interfaces, when the FPGA is at the far end.

Use dynamic OCT in place of both the series and parallel termination at the FPGA end of the line. Typically use dynamic OCT for DQ and DQS signals in both DDR2 and DDR3 type interfaces. As the parallel termination is dynamically disabled during writes, the FPGA driver only ever drives into a Class I transmission line. When combined with dynamic ODT at the memory, a truly dynamic Class I termination scheme exists where both reads and writes are always fully Class I terminated in each direction. Hence, you can use a fully dynamic bidirectional Class I termination scheme instead of a static discretely terminated Class II topology, which saves power, printed circuit board (PCB) real estate, and component cost.

### Stratix III and Stratix IV Devices

Stratix III and Stratix IV devices feature full dynamic OCT termination capability, Altera advise that you use this feature combined with the SDRAM ODT to simplify PCB layout and save power.

### Arria II GX Devices

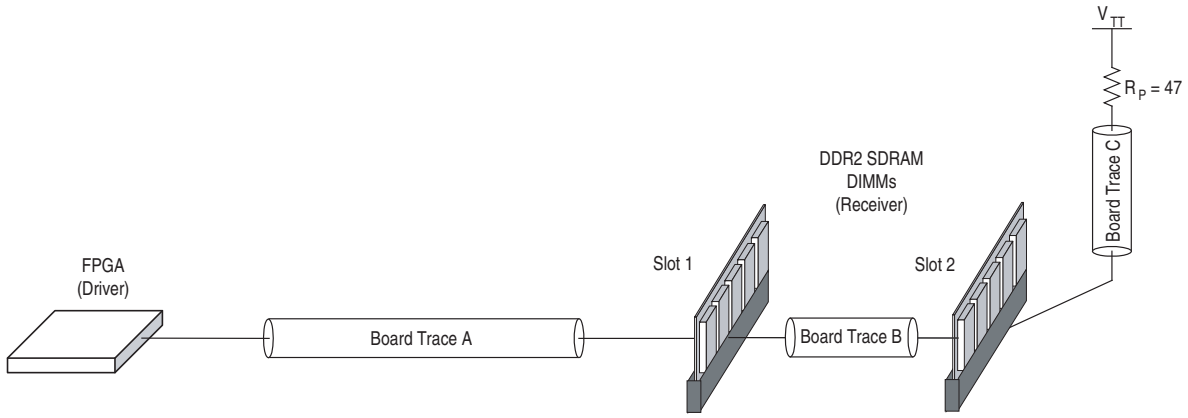
Arria II GX devices do not support dynamic OCT. Altera recommends that you use series OCT with SDRAM ODT. Use parallel discrete termination at the FPGA end of the line when necessary,

For more information, refer to Chapter 1, DDR2 SDRAM Interface Termination, Drive Strength, Loading, and Board Layout Guidelines.

# Dual-DIMM DDR2 Clock, Address, and Command Termination and Topology

The address and command signals on a DDR2 SDRAM interface are unidirectional signals that the FPGA memory controller drives to the DIMM slots. These signals are always Class-I terminated at the memory end of the line (Figure 3–19). Always place DDR2 SDRAM address and command Class-I termination after the last DIMM. The interface can have one or two DIMMs, but never more than two DIMMs total.

**Figure 3–19. Multi DIMM DDR2 Address and Command Termination Topology**




In Figure 3–19, observe the following points:

- Board trace A = 1.9 to 4.5 inches (48 to 115 mm)
- Board trace B = 0.425 inches (10.795 mm)
- Board trace C = 0.2 to 0.55 inches (5 to 13 mm)
- Total of board trace A + B + C = 2.5 to 5 inches (63 to 127 mm)
- $R_P$ = 36 to 56 Ω
- Length match all address and command signals to +250 mils (+5 mm) or +/– 50 ps of memory clock length at the DIMM.

You may place a compensation capacitor directly before the first DIMM slot 1 to improve signal quality on the address and command signal group. If you fit a capacitor, Altera recommends a value of 24 pF.

For more information, refer to *Micron TN47-01*.

## Address and Command Signals

The address and command group of signals: bank address, address, RAS#, CAS#, and WE#, operate a different toggle rate depending on whether a you implement a full-rate or half-rate memory controller.

In full-rate designs, the address and command group of signals are 1T signals, which means that the signals can change every memory clock cycle. Address and command signals are also single data rate (SDR). Hence in a full-rate PHY design, the address and command signals operate at a maximum frequency of 0.5 × the data rate. For example in a 266-MHz full rate design, the maximum address and command frequency is 133 MHz.

In half-rate designs the address and command group of signals are 2T signals, which means that the signals change only every two memory clock cycles. As the signals are also SDR, in a half-rate PHY design, the address and command signals operate at a maximum frequency of 0.25 × the data rate. For example, in a 400-MHz half-rate design, the maximum address and command frequency is 100 MHz.

## Control Group Signals

The control group of signals: chip select CS#, clock enable CKE, and ODT are always 1T regardless of whether you implement a full-rate or half-rate design. As the signals are also SDR, the control group signals operate at a maximum frequency of 0.5 × the data rate. For example, in a 400-MHz design, the maximum control group frequency is 200 MHz.

## Clock Group Signals

Depending on the specific form factor, DDR2 SDRAM DIMMs have two or three differential clock pairs, to ensure that the loading on the clock signals is not excessive. The clock signals are always terminated on the DIMMs and hence no termination is required on your PCB. Additionally, each DIMM slot is required to have its own dedicated set of clock signals. Hence clock signals are always point-to-point from the FPGA PHY to each individual DIMM slot. Individual memory clock signals should never be shared between two DIMM slots.

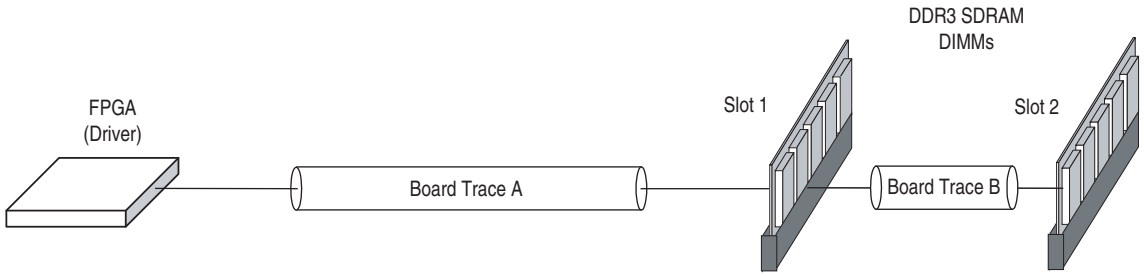
A typical two slot DDR2 DIMM design therefore has six differential memory clock pairs—three to the first DIMM and three to the second DIMM. All six memory clock pairs must be delay matched to each other to ±25 mils (±0.635 mm) and ±10 mils (±0.254 mm) for each CLK to CLK# signal.

You may place a compensation capacitor between each clock pair directly before the DIMM connector, to improve the clock slew rates. As FPGA devices have fully programmable drive strength and slew rate options, this capacitor is usually not required for FPGA design. However, Altera advise that you simulate your specific implementation to ascertain if this capacitor is required or not. If fitted the best value is typically 5 pF.

# DDR3 SDRAM

This section details the system implementation of a dual slot unbuffered DDR3 SDRAM interface, operating at up to 400 MHz and 800 Mbps data rates. Figure 3–20 shows a typical DQS, DQ, and DM, and address and command signal topology for a dual-DIMM interface configuration, using the ODT feature of the DDR3 SDRAM components combined with the dynamic OCT features available in Stratix III and Stratix IV devices.

**Figure 3–20. Multi DIMM DDR3 DQS, DQ, and DM, and Address and Command Termination Topology**




In Figure 3–20, observe the following points:

- Board trace A = 1.9 to 4.5 inches (48 to 115 mm)
- Board trace B = 0.425 inches (10.795 mm)
- This topology to both DIMMs is accurate for DQS, DQ, and DM, and address and command signals
- This topology is not correct for CLK and CLK# and control group signals (CS#, CKE, and ODT), which are always point-to-point single rank only.

## Comparison of DDR3 and DDR2 DQ and DQS ODT Features and Topology

DDR3 and DDR2 SDRAM systems are quite similar. The physical topology of the data group of signals may be considered nearly identical. The FPGA end (driver) I/O standard changes from SSTL18 for DDR2 to SSTL15 for DDR3, but all other OCT settings are identical. DDR3 offers enhanced ODT options for termination and drive-strength settings at the memory end of the line.

For more information, refer to the DDR3 SDRAM ODT matrix for writes and the DDR3 SDRAM ODT matrix for reads tables in Chapter 2, DDR3 SDRAM Interface Termination, Drive Strength, Loading, and Board Layout Guidelines.

# Dual-DIMM DDR3 Clock, Address, and Command Termination and Topology

One significant difference between DDR3 and DDR2 DIMM based interfaces is the address, command and clock signals. DDR3 uses a daisy chained based architecture when using JEDEC standard modules. The address, command, and clock signals are routed on each module in a daisy chain and feature a fly-by termination on the module. Impedance matching is required to make the dual-DIMM topology work effectively—40 to 50 Ω traces should be targeted on the main board.

## Address and Command Signals

Two UDIMMs result in twice the effective load on the address and command signals, which reduces the slew rate and makes it more difficult to meet setup and hold timing ($t_{IS}$ and $t_{IH}$). However, address and command signals operate at half the interface rate and are SDR. Hence a 400-Mbps data rate equates to an address and command fundamental frequency of 100 MHz.

## Control Group Signals

The control group signals (chip Select CS#, clock enable CKE, and ODT) are only ever single rank. A dual-rank capable DDR3 DIMM slot has two copies of each signal, and a dual-DIMM slot interface has four copies of each signal. Hence the signal quality of these signals is identical to a single rank case. The control group of signals, are always 1T regardless of whether you implement a full-rate or half-rate design. As the signals are also SDR, the control group signals operate at a maximum frequency of 0.5 × the data rate. For example, in a 400 MHz design, the maximum control group frequency is 200 MHz.

## Clock Group Signals

Like the control group signals, the clock signals in DDR3 SDRAM are only ever single rank loaded. A dual-rank capable DDR3 DIMM slot has two copies of the signal, and a dual-slot interface has four copies of the `mem_clk` and `mem_clk_n` signals.

For more information about a DDR3 two-DIMM system design, refer to Micron *TN-41-08: DDR3 Design Guide for Two-DIMM Systems*.

The Altera DDR3 ALTMEMPHY megafunction does not support the 1T address and command topology referred to in this Micron Technical Note—only 2T implementations are supported.

# Write to Memory in Slot 1 Using an ODT Setting of 75 Ω With One Slot Populated

Figure 3–21 shows the simulation and board measurement of the signal at the memory when the FPGA is writing to the memory with an ODT setting of 75 Ω and using a 25-Ω OCT drive strength setting on the FPGA.

**Figure 3–21. HyperLynx Simulation and Board Measurement of the Signal at the Memory in Slot 1 With Slot 2 Unpopulated**

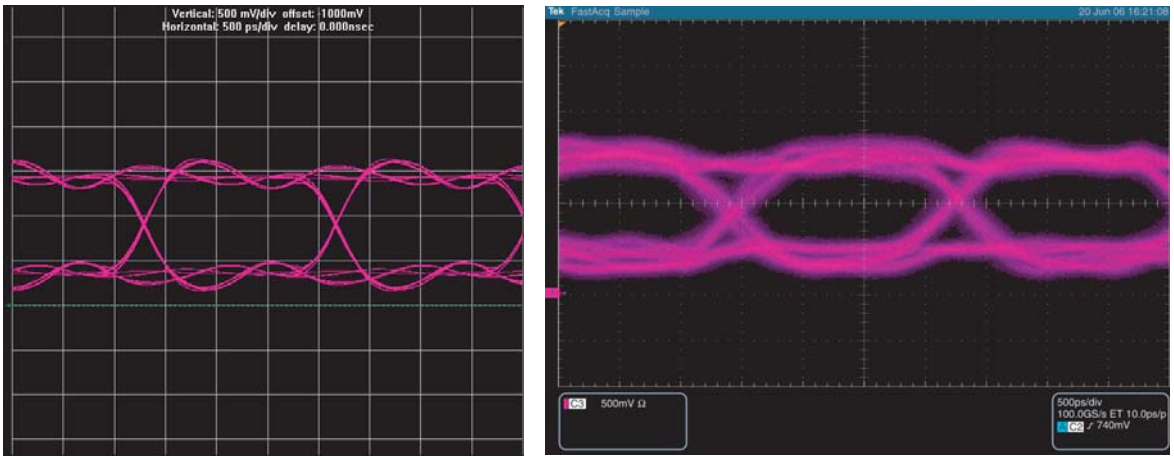



Table 3–11 summarizes the comparison between the simulation and board measurements of the signal seen at the DDR2 SDRAM of a dual-DIMM with slot 1 populated by a memory interface using a different ODT setting.

**Table 3–11. Comparison of the Signal at the Memory of a Dual-DIMM Interface With Only Slot 1 Populated and a Different ODT Setting**

| Type | Eye Width (ns) | Eye Height (V) | Overshoot (V) | Undershoot (V) | Rising Edge Rate (V/ns) | Falling Edge Rate (V/ns) |
|---|---|---|---|---|---|---|
| **ODT Setting of 75 Ω** | | | | | | |
| Simulation | 1.68 | 0.91 | NA | NA | 1.88 | 1.88 |
| Measurements | 1.28 | 0.57 | NA | NA | 1.54 | 1.38 |
| **ODT Setting of 150 Ω** | | | | | | |
| Simulation | 1.68 | 0.97 | 0.06 | NA | 2.67 | 2.13 |
| Measurements | 1.30 | 0.63 | 0.22 | 0.20 | 1.74 | 1.82 |

# Write to Memory in Slot 2 Using an ODT Setting of 75 Ω With One Slot Populated

Figure 3–22 shows the simulation and measurements result of the signal seen at the memory when the FPGA is writing to the memory with an ODT setting of 75 Ω and using a 25-Ω OCT drive strength setting on the FPGA.

**Figure 3–22. HyperLynx Simulation and Board Measurement of the Signal at the Memory in Slot 2 with Slot 1 Unpopulated**

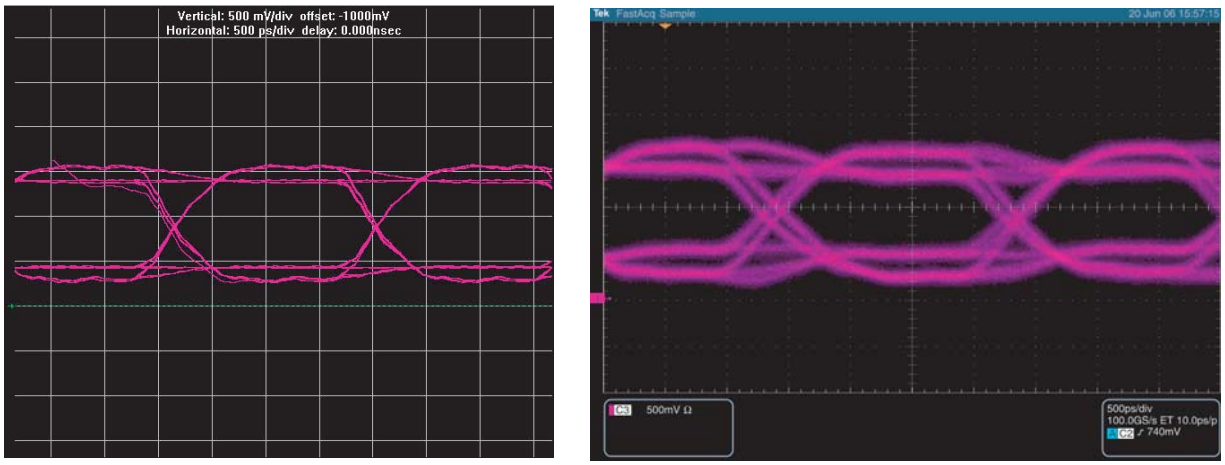



Table 3–12 summarizes the comparison of the signal at the memory of a dual-DIMM memory interface with either slot 1 or slot 2 populated using a double parallel termination using an ODT setting of 75 Ω with a memory-side series resistor with a 25-Ω OCT strength setting on the FPGA.

**Table 3–12. Comparison of Signal at the Memory of a Dual-DIMM Interface With Only Slot 2 Populated and a Different ODT Setting**

| Type | Eye Width (ns) | Eye Height (V) | Overshoot (V) | Undershoot (V) | Rising Edge Rate (V/ns) | Falling Edge Rate (V/ns) |
|---|---|---|---|---|---|---|
| **ODT Setting of 75 Ω** | | | | | | |
| Simulation | 1.68 | 0.89 | NA | NA | 1.82 | 1.93 |
| Measurements | 1.29 | 0.59 | NA | NA | 1.60 | 1.29 |
| **ODT Setting of 150 Ω** | | | | | | |
| Simulation | 1.69 | 0.94 | 0.07 | 0.02 | 1.88 | 2.29 |
| Measurements | 1.28 | 0.68 | 0.24 | 0.20 | 1.60 | 1.60 |

# Write to Memory in Slot 1 Using an ODT Setting of 150 Ω With Both Slots Populated

Figure 3–23 shows the HyperLynx simulation and board measurement of the signal at the memory in slot 1 of a double parallel termination using an ODT setting of 150 Ω on Slot 2 with a memory-side series resistor transmission line when the FPGA is writing to the memory with a 25-Ω OCT drive strength setting.

**Figure 3–23. HyperLynx Simulation and Board Measurement of the Signal at the Memory in Slot 1 With Both Slots Populated**

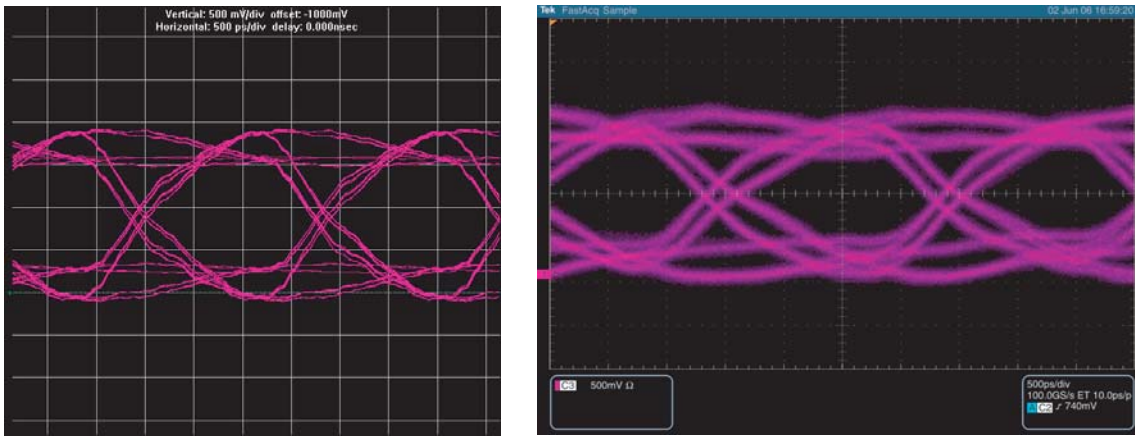



Table 3–13 summarizes the comparison between the simulation and board measurements of the signal seen at the memory in slot 1 of a dual-DIMM memory interface with both slots populated using a double parallel termination using a different ODT setting on Slot 2 with a memory-side series resistor with a 25-Ω OCT strength setting on the FPGA.

**Table 3–13. Comparison of Signal at the Memory of a Dual-DIMM Interface with Both Slots Populated and a Different ODT Setting on Slot 2**

| Type | Eye Width (ns) | Eye Height (V) | Overshoot (V) | Undershoot (V) | Rising Edge Rate (V/ns) | Falling Edge Rate (V/ns) |
|---|---|---|---|---|---|---|
| **ODT Setting of 150 Ω** | | | | | | |
| Simulation | 1.60 | 1.18 | 0.02 | NA | 1.71 | 1.71 |
| Measurements | 0.89 | 0.78 | 0.13 | 0.17 | 1.19 | 1.32 |
| **ODT Setting of 75 Ω** | | | | | | |
| Simulation | 1.60 | 1.18 | 0.02 | NA | 1.71 | 1.71 |
| Measurements | 0.97 | 0.77 | 0.05 | 0.04 | 1.25 | 1.25 |

# Write to Memory in Slot 2 Using an ODT Setting of 150 Ω With Both Slots Populated

Figure 3–24 shows the HyperLynx simulation and board measurement of the signal at the memory in slot 2 of a double parallel termination using an ODT setting of 150 Ω on slot 1 with a memory-side series resistor transmission line when the FPGA is writing to the memory with a 25-Ω OCT drive strength setting.

**Figure 3–24. HyperLynx Simulation and Board Measurements of the Signal at the Memory in Slot 2 with Both Slots Populated**

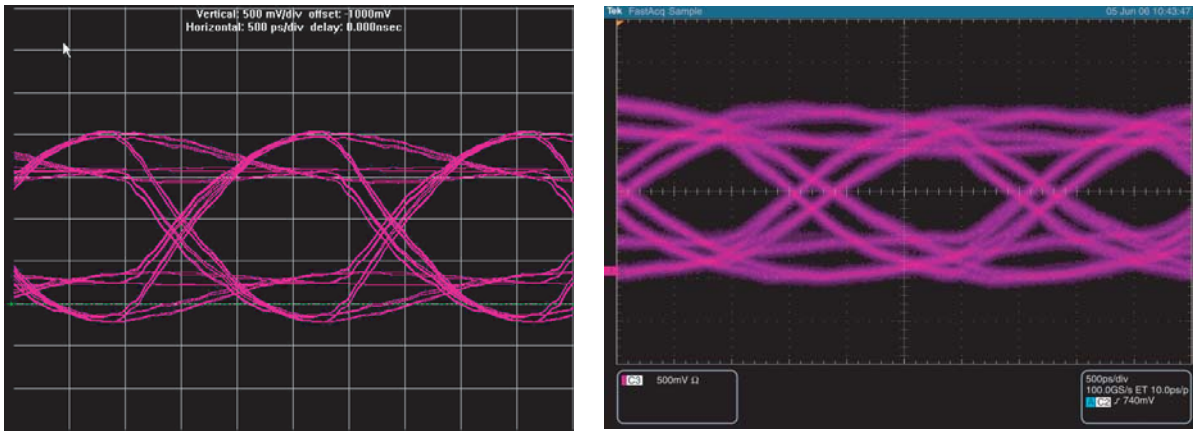



Table 3–14 summarizes the comparison between the simulation and board measurements of the signal seen at the memory of a dual-DIMM memory interface with both slots populated using a double parallel termination using a different ODT setting on Slot 1 with a memory-side series resistor with a 25-Ω OCT strength setting on the FPGA.

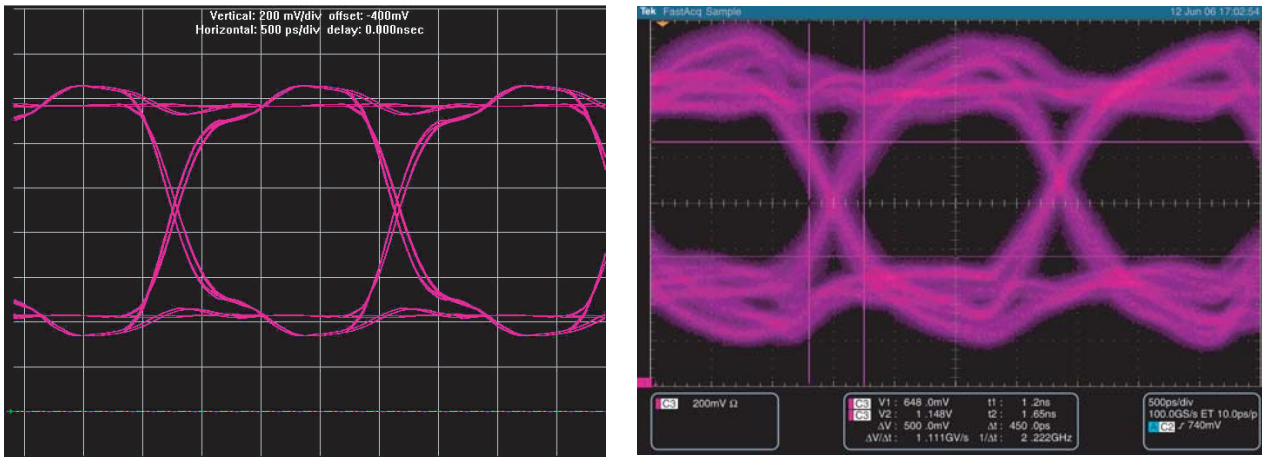
**Table 3–14. Comparison of the Signal at the Memory of a Dual-DIMM Interface With Both Slots Populated and a Different ODT Setting on Slot 1**

| Type | Eye Width (ns) | Eye Height (V) | Overshoot (V) | Undershoot (V) | Rising Edge Rate (V/ns) | Falling Edge Rate (V/ns) |
|---|---|---|---|---|---|---|
| **ODT Setting of 150 Ω** | | | | | | |
| Simulation | 1.45 | 1.11 | 0.19 | 0.17 | 1.43 | 2.21 |
| Measurements | 0.71 | 0.81 | 0.12 | 0.20 | 0.93 | 1.00 |
| **ODT Setting of 75 Ω** | | | | | | |
| Simulation | 1.60 | 1.16 | 0.10 | 0.08 | 1.68 | 1.60 |
| Measurements | 1.10 | 0.85 | 0.16 | 0.19 | 1.11 | 1.25 |

# Read from Memory in Slot 1 Using an ODT Setting of 150 Ω on Slot 2 with Both Slots Populated

Figure 3–25 shows the HyperLynx simulation and board measurement of the signal at the FPGA of a double parallel termination using an external parallel resistor on the FPGA side with a memory-side series resistor and an ODT setting of 150 Ω with a full drive strength setting on the memory.

**Figure 3–25. HyperLynx Simulation and Board Measurement of the Signal at the FPGA When Reading From Slot 1 With Both Slots Populated** ***(Note 1)***




**Note to Figure 3–25:**

(1) The vertical scale used for the simulation and measurement is set to 200 mV per division.

Table 3–15 summarizes the comparison between the simulation and board measurements of the signal seen at the FPGA of a dual-DIMM memory interface with both slots populated using a different ODT setting on Slot 2.

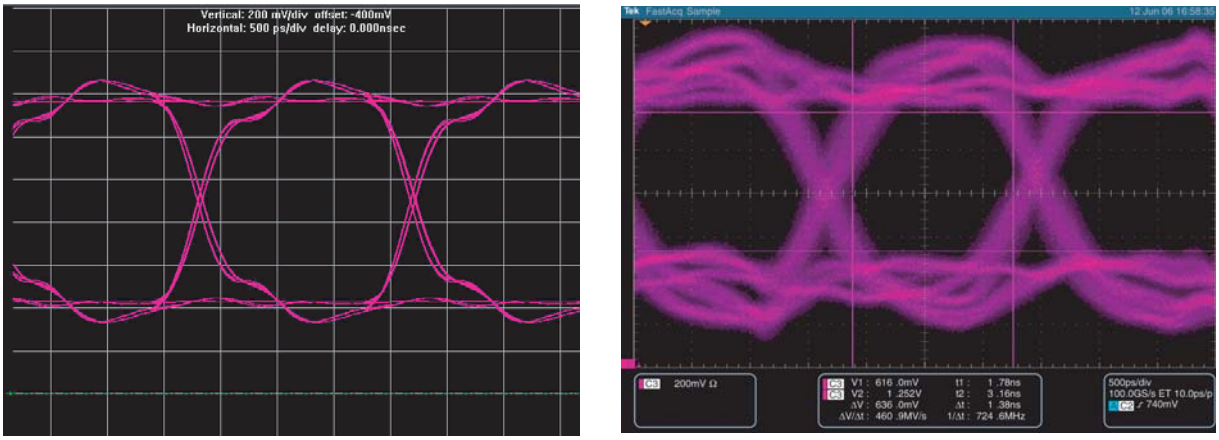
**Table 3–15. Comparison of Signal at the FPGA of a Dual-DIMM Interface With Both Slots Populated and a Different ODT Setting on Slot 2**

| Type | Eye Width (ns) | Eye Height (V) | Overshoot (V) | Undershoot (V) | Rise Edge Rate (V/ns) | Falling Edge Rate (V/ns) |
|---|---|---|---|---|---|---|
| **ODT Setting of 150 Ω** | | | | | | |
| Simulation | 1.68 | 0.77 | NA | NA | 1.88 | 1.88 |
| Measurements | 0.76 | 0.55 | NA | NA | 1.11 | 1.14 |
| **ODT Setting of 75 Ω** | | | | | | |
| Simulation | 1.74 | 0.87 | NA | NA | 1.91 | 1.88 |
| Measurements | 0.86 | 0.59 | NA | NA | 1.11 | 1.09 |

# Read From Memory in Slot 2 Using an ODT Setting of 150 Ω on Slot 1 With Both Slots Populated

Figure 3–26 shows the HyperLynx simulation board measurement of the signal seen at the FPGA of a double parallel termination using an external parallel resistor on the FPGA side with memory-side series resistor and an ODT setting of 150 Ω with a full drive strength setting on the memory.

**Figure 3–26. HyperLynx Simulation Board Measurement of the Signal at the FPGA When Reading From Slot 2 With Both Slots Populated** ***(Note 1)***




**Note to Figure 3–26:**

(1) The vertical scale used for the simulation and measurement is set to 200 mV per division.

Table 3–16 summarizes the comparison between the simulation and board measurements of the signal seen at the FPGA of a dual-DIMM memory interface with both slots populated using a different ODT setting on Slot 1.

**Table 3–16. Comparison of Signal at the FPGA of a Dual-DIMM Interface With Both Slots Populated and a Different ODT Setting on Slot 1**

| Type | Eye Width (ns) | Eye Height (V) | Overshoot (V) | Undershoot (V) | Rising Edge Rate (V/ns) | Falling Edge Rate (V/ns) |
|---|---|---|---|---|---|---|
| **ODT Setting of 150 Ω** | | | | | | |
| Simulation | 1.70 | 0.74 | NA | NA | 1.91 | 1.64 |
| Measurements | 0.74 | 0.64 | NA | NA | 1.14 | 1.14 |
| **ODT Setting of 75 Ω** | | | | | | |
| Simulation | 1.70 | 0.81 | NA | NA | 1.72 | 1.99 |
| Measurements | 0.87 | 0.59 | NA | NA | 1.09 | 1.14 |

# Conclusion

This chapter looks at single- and dual-DIMM DDR2 and DDR3 SDRAM interfaces and makes recommendations on topology and termination, to ensure optimum design guidelines and best signal quality.

In the design of any dual-DIMM interface, you should follow the memory vendor recommendations on the optimum ODT setting, slot population, and operations to the DIMM locations. In addition, this chapter recommends the OCT settings to use at the FPGA, to ensure optimum configuration.

The simulations and experiments referenced throughout this chapter show that you can achieve good signal quality, if you follow the memory vendors recommended ODT settings. Although the DDR2 simulations and experimental results in this chapter are based on the Stratix II High Speed High Density Board, you can apply the general principles to any dual-DIMM design. The addition of dynamic OCT in Stratix III and Stratix IV devices has simplified the board design further by removing the need for the previously required FPGA end discrete parallel termination.

Even though this chapter covers several combinations of ODT and OCT termination, it is critical that as a board designer you perform system specific simulations to ensure good signal integrity in your dual-DIMM SDRAM designs.

# References


- *JEDEC Standard Publication JESD79-2, DDR2 SDRAM Specification, JEDEC Solid State Technology Association*.
- *JEDEC Standard Publication JESD8-15A, Stub Series Termination Logic for 1.8 V (SSTL-18), JEDEC Solid State Technology Association.*
- Micron *TN-47-01: DDR2 Design Guide for Two-DIMM Systems*.
- Micron *TN-41-08: DDR3 Design Guide for Two-DIMM Systems*
- Micron *TN-47-17: DDR2 SODIMM Optimized Address/Command Nets Introduction*.
- Samsung Electronics *Application Note: DDR2 ODT Control*
- *High-Speed Digital Design – A Handbook of Black Magic*, Howard Johnson and Martin Graham, Prentice Hall, 1993.
- *Circuits Interconnects, and Packaging for VLSI*, H.B. Bakoglu, Addison Wesley, 1990.
- *Signal Integrity – Simplified*, Eric Bogatin, Prentice Hall Modern Semiconductor Design Series, 2004.
- *Handbook of Digital Techniques for High-Speed Design*, Tom Granberg, Prentice Hall Modern Semiconductor Design Series, 2004.
- *Termination Placement in PCB Design How Much Does it Matter?*, Doug Brooks, UltraCAD Design Inc.
- *PC4300 DDR2 SDRAM Unbuffered DIMM Design Specification, Revision 0.5, Oct 30, 2003*.

# 4. RLDRAM II Interface Termination and Layout Guidelines

This chapter provides guidelines for you to improve your system's signal integrity and layout guidelines to help successfully implement an RLDRAM II interface in your system.

The RLDRAM II Controller with UniPHY intellectual property (IP) enables you to implement Common I/O (CIO) RLDRAM II interfaces with Stratix III, Stratix IV, and Stratix V devices. You can implement Separate I/O (SIO) RLDRAM II interfaces with the ALTDQ_DQS or ALTDQ_DQS2 megafunctions.

This chapter focuses on the following key factors that affect signal integrity:

- I/O standards
- RLDRAM II configurations
- Signal terminations
- Printed circuit board (PCB) layout guidelines

## I/O Standards

RLDRAM II interface signals use one of the following JEDEC I/O signalling standards:

- HSTL-15—provides the advantages of lower power and lower emissions.
- HSTL-18—provides increased noise immunity with slightly greater output voltage swings.

To select the most appropriate standard for your interface, refer to the *Device Datasheet for Arria II Devices* chapter in the *Arria II Device Handbook*, the *Stratix III Device Datasheet: DC and Switching Characteristics* chapter in the *Stratix III Device Handbook*, the *DC and Switching Characteristics for Stratix IV Devices* chapter in the *Stratix IV Device Handbook*, or the *DC and Switching Characteristics for Stratix V Devices* chapter in the *Stratix V Device Handbook*.

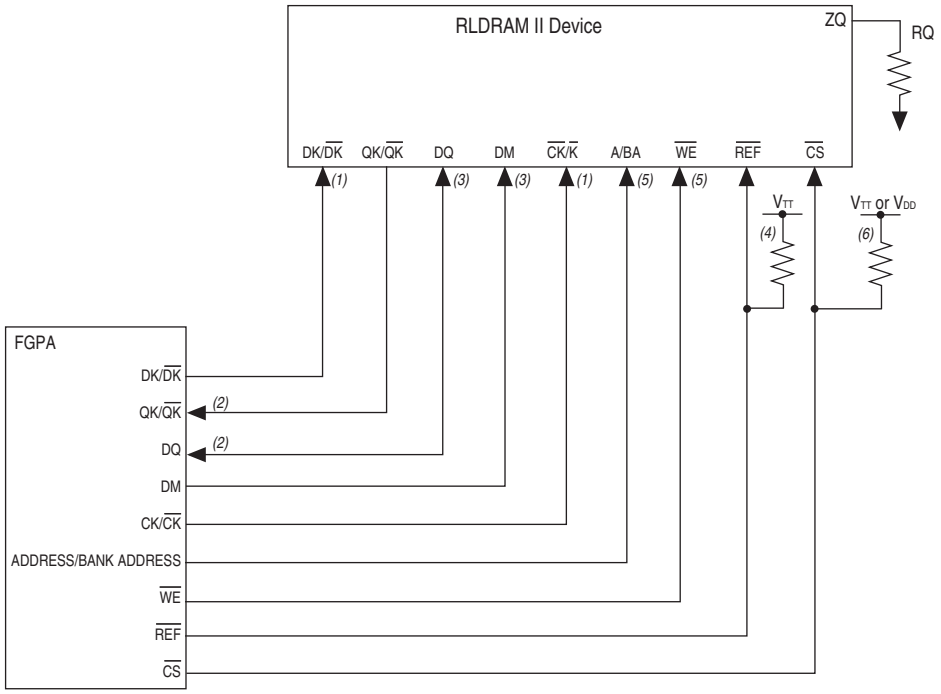
The RLDRAM II Controller with UniPHY IP defaults to HSTL 1.8 V Class I outputs and HSTL 1.8 V inputs.

## RLDRAM II Configurations

The RLDRAM II Controller with UniPHY IP supports interfaces for CIO RLDRAM II with a single device, and two devices in a width expansion configuration up to maximum width of 72 bits. This chapter focusses the layout and guidelines for CIO RLDRAM II interfaces. However, the termination and layout principles for SIO RLDRAM II interfaces are similar to CIO RLDRAM II, except that SIO RLDRAM II interfaces have unidirectional data buses.

Figure 4–1 shows the main signal connections between the FPGA and a single CIO RLDRAM II component.

**Figure 4–1. Configuration With A Single CIO RLDRAM II Component**




**Notes to Figure 4–1:**

(1) Use external differential termination on DK/DK# and CK/CK#.

(2) Use FPGA parallel on-chip termination (OCT) for terminating QK/QK# and DQ on reads.

(3) Use RLDRAM II component on-die termination (ODT) for terminating DQ and DM on writes.

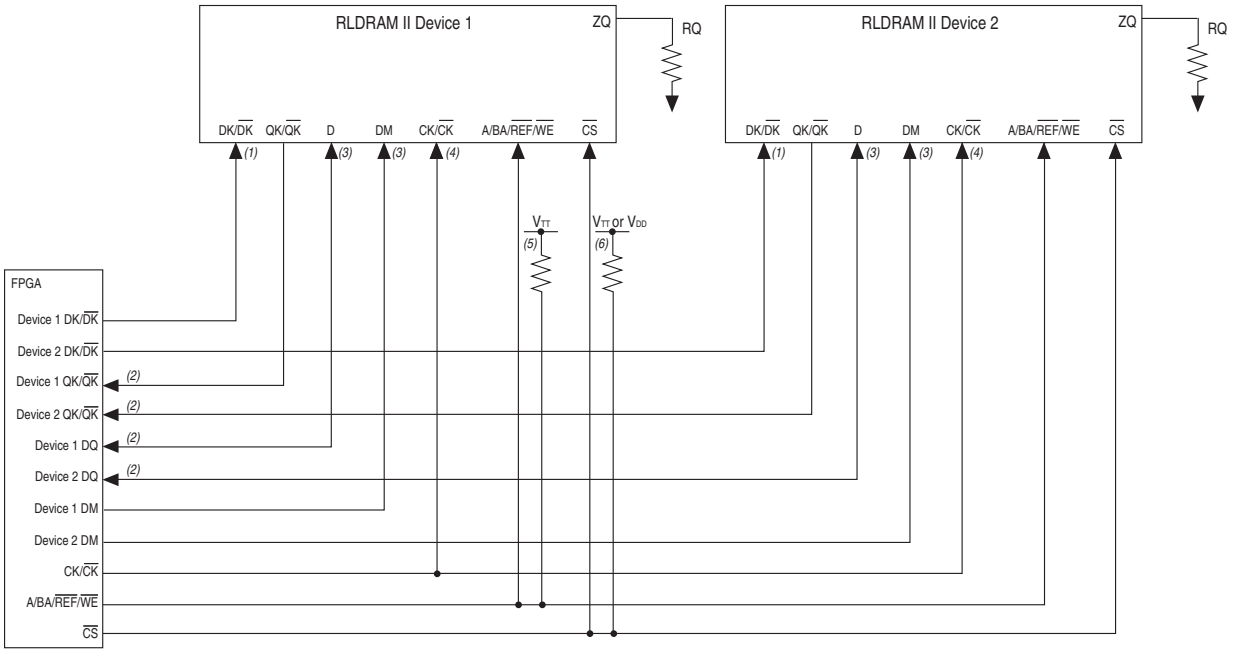
(4) Use external discrete termination with fly-by placement to avoid stubs.

(5) Use external discrete termination for this signal, as shown for REF.

(6) Use external discrete termination , as shown for REF, but you may require a pull-up resistor to $V_{DD}$ as an alternative option. Refer to the RLDRAM II device data sheet for more information about RLDRAM II power-up sequencing.

Figure 4–2 shows the main signal connections between the FPGA and two CIO RLDRAM II components in a width expansion configuration.

**Figure 4–2. Configuration With Two CIO RLDRAM II Components In A Width Expansion Configuration**




**Notes to Figure 4–2:**

(1) Use external differential termination on DK/DK#.
(2) Use FPGA parallel OCT for terminating QK/QK# and DQ on reads.
(3) Use RLDRAM II component ODT for terminating DQ and DM on writes.
(4) Use external dual 200 Ω differential termination.
(5) Use external discrete termination at the trace split of the balanced T or Y topology.
(6) Use external discrete termination at the trace split of the balanced T or Y topology, but you may require a pull-up resistor to $V_{DD}$ as an alternative option. Refer to the RLDRAM II device data sheet for more information about RLDRAM II power-up sequencing.

# Signal Terminations

Stratix III, Stratix IV, and Stratix V devices offer OCT technology.

Table 4–1 summarizes the extent of OCT support for each device.

**Table 4–1. On-Chip Termination Schemes (Part 1 of 2)**

| Termination Scheme | HSTL-15 and HSTL-18 | FPGA Device | |
|---|---|---|---|
| | | Arria II GZ, Stratix III, and Stratix IV | Stratix V |
| | | Row/Column I/O | Row/Column I/O |
| On-Chip Series Termination without Calibration | Class I | 50 | 50 |
| On-Chip Series Termination with Calibration | Class I | 50 | 50 *(1)* |

**Table 4–1. On-Chip Termination Schemes (Part 2 of 2)**

| Termination Scheme | HSTL-15 and HSTL-18 | FPGA Device | |
|---|---|---|---|
| | | Arria II GZ, Stratix III, and Stratix IV | Stratix V |
| | | Row/Column I/O | Row/Column I/O |
| On-Chip Parallel Termination with Calibration | Class I | 50 | 50 *(1)* |

**Note to Table 4–1:**

(1) Although 50 Ω is the recommended option, Stratix V devices offer a wider range of calibrated termination impedances.

On-chip series ($R_S$) termination supports output buffers, and bidirectional buffers only when they are driving output signals. On-chip parallel ($R_T$) termination supports input buffers, and bidirectional buffers only when they are input signals. RLDRAM II CIO interfaces have bidirectional data paths. The UniPHY IP uses dynamic OCT on the datapath, which switches between series OCT for memory writes and parallel OCT for memory reads .

For Arria II GZ, Stratix III, and Stratix IV devices , the HSTL Class I I/O calibrated terminations are calibrated against 50 Ω 1% resistors connected to the $R_{UP}$ and $R_{DN}$ pins in an I/O bank with the same $V_{CCIO}$ as the RLDRAM II interface. For Stratix V devices, the HSTL Class I I/O calibrated terminations are calibrated against 100 Ω or 240 Ω 1% resistors connected to the $R_{ZQ}$ pins in an I/O bank with the same $V_{CCIO}$ as the RLDRAM II interface.

The calibration occurs at the end of the device configuration.

RLDRAM II memory components have a ZQ pin which connects through a resistor RQ to ground. Typically the RLDRAM II output signal impedance is 0.2 × RQ. Refer to the RLDRAM II device data sheet for more information.

For information about OCT, refer to the *I/O Features in Arria II Devices* chapter in the *Arria II Device Handbook*, *Stratix III Device I/O Features* chapter in the *Stratix III Device Handbook*, the *I/O Features in Stratix IV Devices* chapter in the *Stratix IV Device Handbook*, or the *I/O Features in Stratix V Devices* chapter in the *Stratix V Device Handbook*.

The following section shows HyperLynx simulation eye diagrams to demonstrate signal termination options. Altera strongly recommends signal terminations to optimize signal integrity and timing margins, and to minimize unwanted emissions, reflections, and crosstalk.

All of the eye diagrams shown in this section are for a 50 Ω trace with a propagation delay of 600 ps which is approximately a 3.3-inch trace on a standard FR4 PCB. The signal I/O standard is HSTL-18.

The eye diagrams shown in this section show the best case achievable and do not take into account PCB vias, crosstalk and other degrading effects such as variations in the PCB structure due to manufacturing tolerances.

Simulate your design to ensure correct functionality.

## Outputs from the FPGA to the RLDRAM II Component

The following output signals are from the FPGA to the RLDRAM II component:

- write data (DQ on the bidirectional data signals for CIO RLDRAM II)
- data mask (DM)
- address, bank address
- command (CS, WE, and REF)
- clocks (CK/CK# and DK/DK#)

For point-to-point single-ended signals requiring external termination, Altera recommends that you place a fly-by termination by terminating at the end of the transmission line after the receiver to avoid unterminated stubs. The guideline is to place the fly-by termination within 100 ps propagation delay of the receiver.

Although not recommended, you can place the termination before the receiver, which leaves an unterminated stub. The stub delay is critical because the stub between the termination and the receiver is effectively unterminated, causing additional ringing and reflections. Stub delays should be less than 50 ps.

Altera recommends that the differential clocks, CK, CK# and DK, DK#, use a differential termination at the end of the trace of the RLDRAM II component. Alternatively, you can terminate each clock output with a parallel termination to $V_{TT}$.

The HyperLynx simulation eye diagrams show simulation cases of write data, address, and chip-select signals with termination options. All eye diagrams are shown at the connection to the receiver device die.

Figure 4–3 shows the double data rate write data using a Stratix IV Class I HSTL-18 with calibrated 50 Ω OCT output driver and the nominal RLDRAM II ODT of 150 Ω

**Figure 4–3. Write Data Simulation at 400 MHz with RLDRAM II ODT**

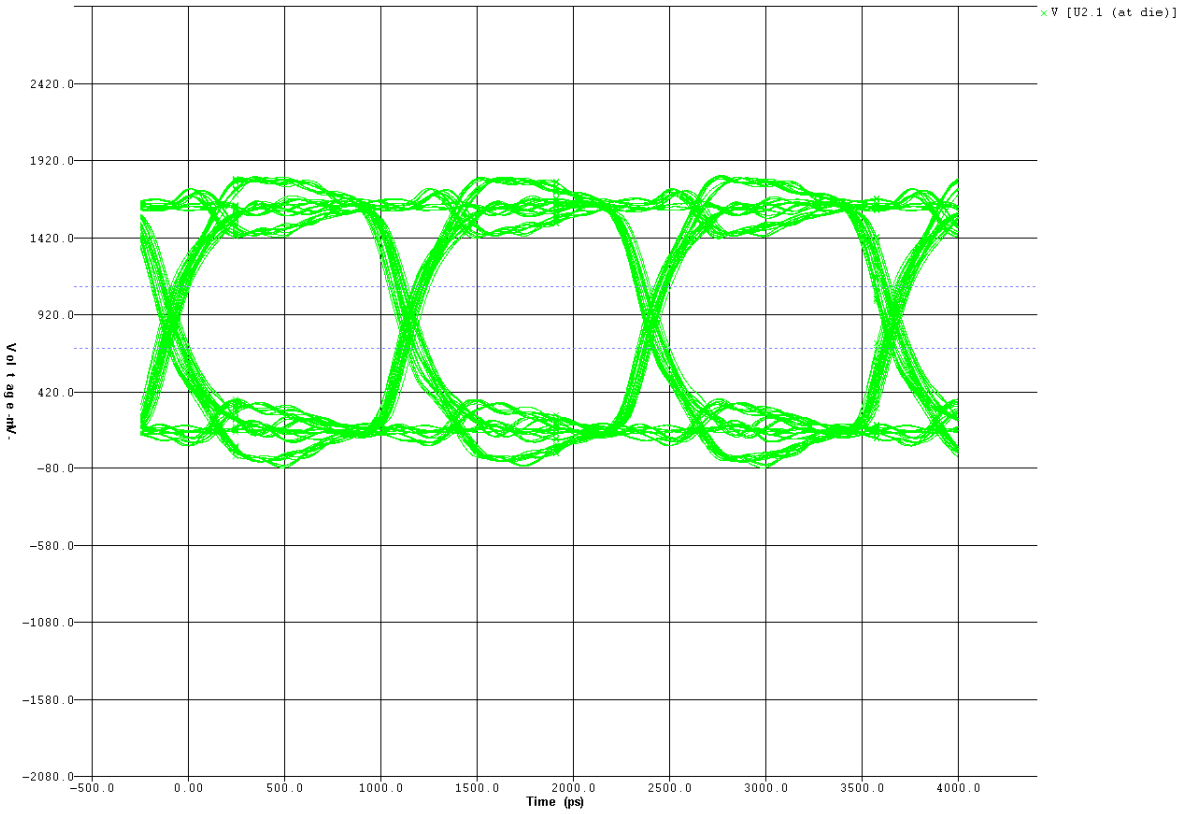


Figure 4–4 shows an address signal at a frequency of 200 MHz using Stratix IV Class I HSTL-18 with a calibrated 50 Ω OCT driver and a 100 ps fly-by 50 Ω parallel termination to $V_{TT}$.

**Figure 4–4. Address Simulation Using Stratix IV Class I HSTL-18 50 Ω Calibration Driver and Fly-by 50 Ω Parallel Termination**

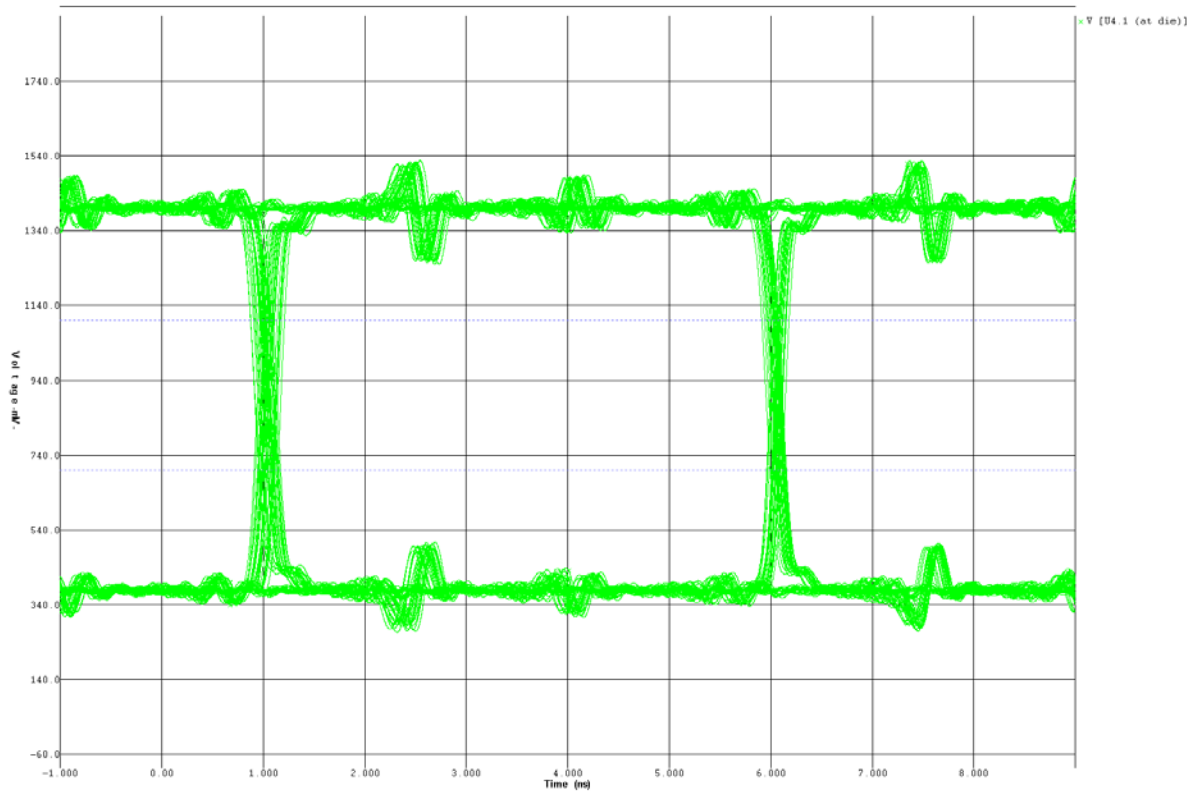


Figure 4–5 shows an address signal at a frequency of 200 MHz using Stratix IV Class I HSTL-18 12 mA driver and a 50 ps stub 50 Ω parallel termination to $V_{TT}$.

**Figure 4–5. Address Simulation Using Stratix IV Class I HSTL-18 50 Ω Calibration Driver and Stub 50 Ω Parallel Termination to $V_{TT}$**

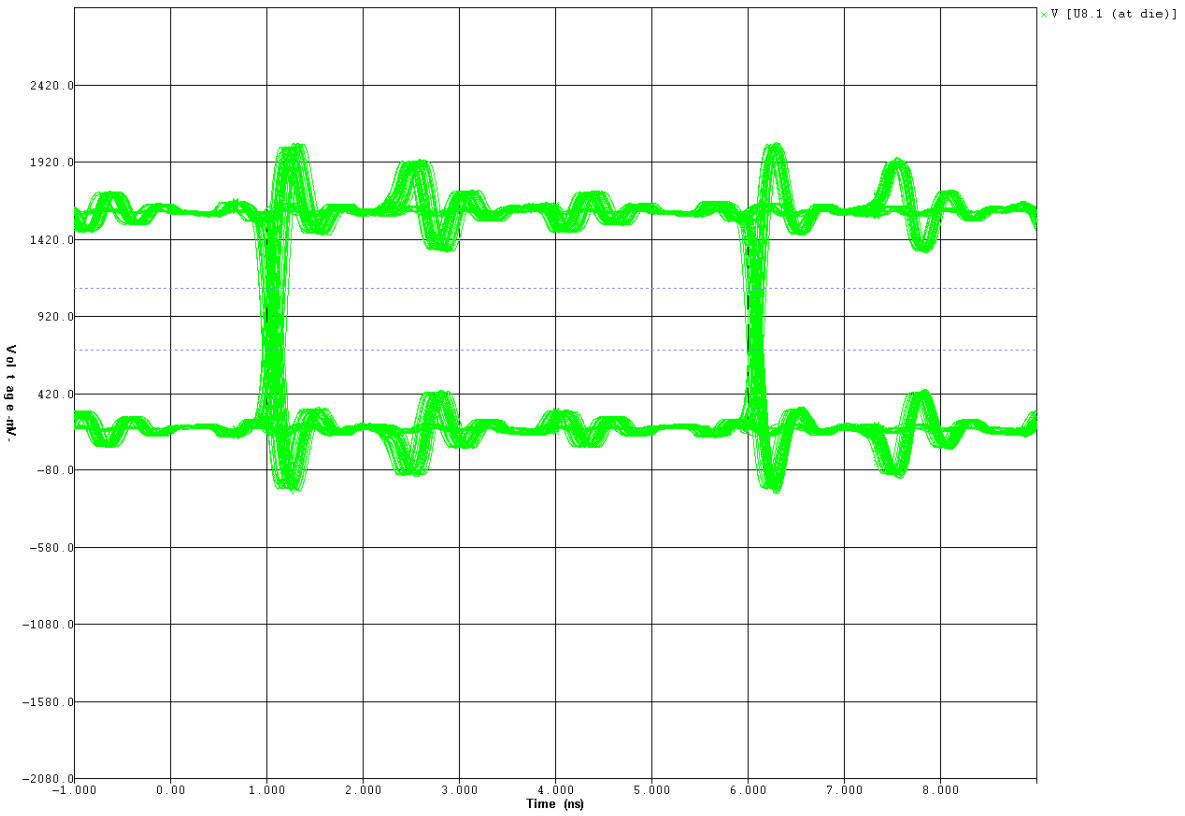
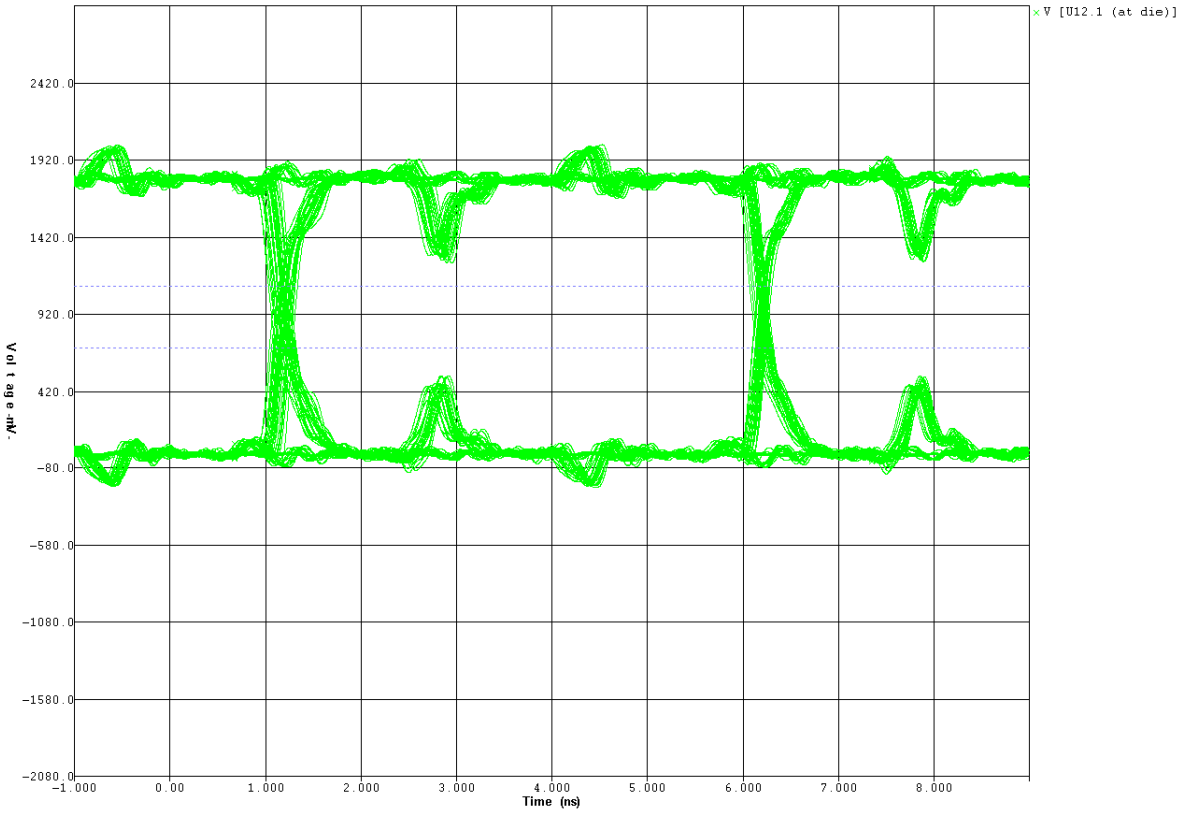


Figure 4–6 shows the chip-select signal at a frequency of 200 MHz using Stratix IV Class I HSTL-18 with a calibrated 50 Ω driver and a 10 K pull-up resistor to $V_{DD}$. The RLDRAM II power sequencing may require the chip selects to have a pull-up resistor. Refer to the RLDRAM II data sheet for further details.

**Figure 4–6. Chip-Select Simulation Using Stratix IV Class I HSTL-18 50 Ω Calibration Driver and 10 K Pull-up Resistor to $V_{DD}$**




For the RLDRAM II width expansion configuration for address and command, use the same principles recommended for "QDR II SRAM Interface Termination and Layout Guidelines" on page 5–1.

For external parallel termination recommended for a balanced T topology, refer to Figure 5–3 on page 5–4, and for HyperLynx simulation diagrams of the width expansion topology for address and command signals, refer to Figure 5–8 through Figure 5–11 on page 5–13.

## Input to the FPGA from the RLDRAM II Component

The RLDRAM II component drives the following input signals into the FPGA:

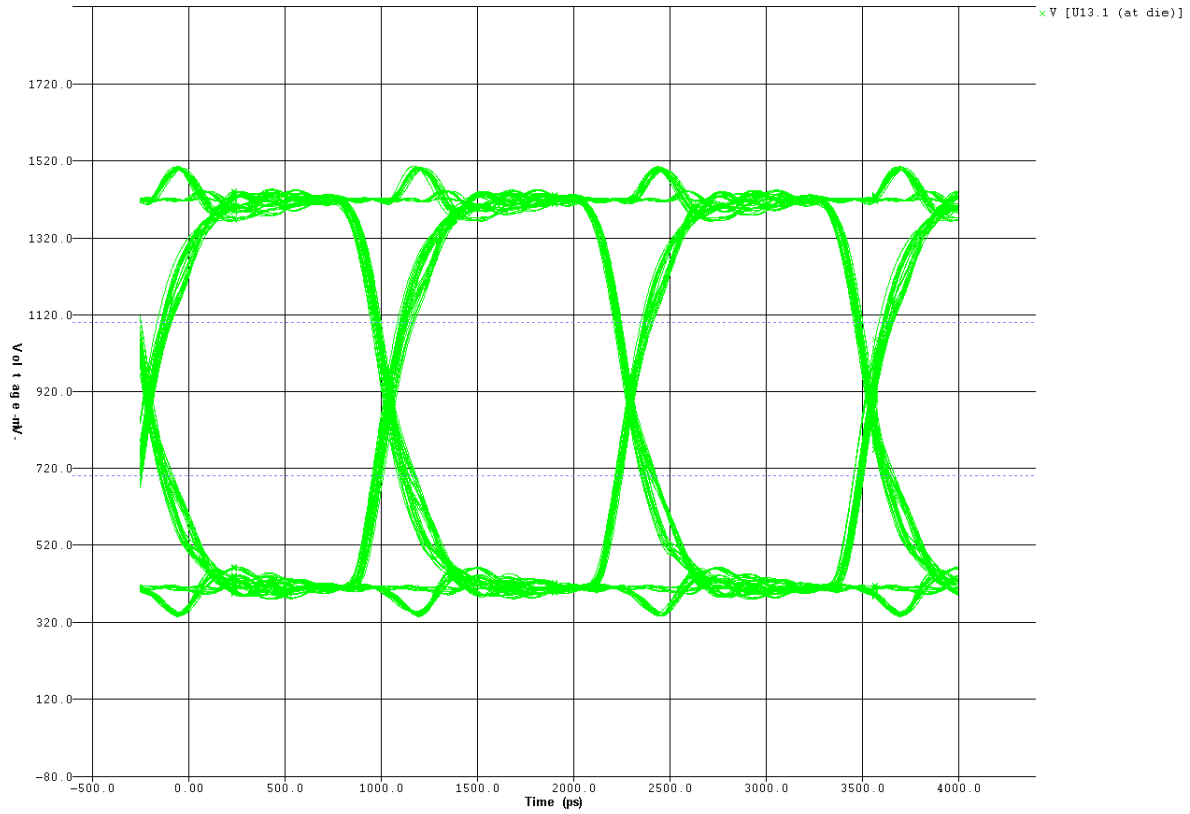
- read data (DQ on the bidirectional data signals for CIO RLDRAM II)
- read clocks (QK/QK# )

Altera recommends that you use the FPGA parallel OCT to terminate the data on reads and read clocks.

The eye diagrams are shown at the FPGA die pin, and the RLDRAM II output driver is Class I HSTL-18 using its ZQ calibration of 50 Ω The RLDRAM II read data is double data rate.

Figure 4–7 shows the ideal case of a fly-by terminated signal using 50 Ωcalibrated parallel OCT for a Stratix IV device.

**Figure 4–7. Read Data Simulation at 400 MHz with 50 ΩParallel OCT Termination**




## Termination Schemes

Table 4–2 shows the recommended termination schemes for major CIO RLDRAM II memory interface signals, which include data (DQ), data mask (DM), clocks (CK, CK#, DK, DK#, QK, and QK#), address, bank address, and command (WE#, REF#, and CS#).

**Table 4–2. Termination Recommendations for Stratix III, Stratix IV, and Stratix V Devices (Part 1 of 2)**

| Signal Type | HSTL 15/18 Standard *(1)*, *(2)*, *(3)* | Memory End Termination |
|---|---|---|
| DK/DK# Clocks | Class I R50 CAL | 100 Ω Differential |
| QK/QK# Clocks | Class I P50 CAL | ZQ50 |
| Data (Write) | Class I R50 CAL | ODT |
| Data (Read) | Class I P50 CAL | ZQ50 |
| Data Mask | Class I R50 CAL | ODT |

**Table 4–2. Termination Recommendations for Stratix III, Stratix IV, and Stratix V Devices (Part 2 of 2)**

| Signal Type | HSTL 15/18 Standard *(1)*, *(2)*, *(3)* | Memory End Termination |
|---|---|---|
| CK/CK# Clocks | Class I R50 CAL | ×1 = 100 Ω Differential *(8)*<br>×2 = 200 Ω Differential *(9)* |
| Address/Bank Address *(4)*, *(5)* | Class I Max Current | 50 Ω Parallel to $V_{TT}$ |
| Command (WE, REF) *(4)*, *(5)* | Class I Max Current | 50 Ω Parallel to $V_{TT}$ |
| Command (CS) *(4)*, *(5)*, *(6)* | Class I Max Current | 50 Ω Parallel to $V_{TT}$ or Pull-up to $V_{DD}$ |
| QVLD *(7)* | Class I P50 CAL | ZQ50 |

**Notes to Table 4–2:**

(1) R is effective series output impedance.
(2) P is effective parallel input impedance.
(3) CAL is calibrated OCT.
(4) For width expansion configuration, the address and control signals are routed to 2 devices. Recommended termination is 50 Ω parallel to $V_{TT}$ at the trace split of a balanced T or Y routing topology. Use a clamshell placement of the two RLDRAM II components to achieve minimal stub delays and optimum signal integrity. Clamshell placement is when two devices overlay each other by being placed on opposite sides of the PCB.
(5) The UniPHY default IP setting for this output is Max Current. A Class I 50 Ω output with calibration output is typically optimal in single load topologies.
(6) Altera recommends that you use a 50 Ω parallel termination to $V_{TT}$ if your design meets the power sequencing requirements of the RLDRAM II component. Refer to the RLDRAM II data sheet for further information.
(7) QVLD is not used in the RLDRAM II Controller with UniPHY implementations.
(8) ×1 is a single-device load.
(9) ×2 is a double-device load. An alternative option is to use a 100 Ω differential termination at the trace split.

☞ Altera recommends that you simulate your specific design for your system to ensure good signal integrity.

# PCB Layout Guidelines

Table 4–3 summarizes RLDRAM II general routing layout guidelines.

☞ The following layout guidelines include several +/- length based rules. These length-based guidelines are for first order timing approximations if you cannot simulate the actual delay characteristics of your PCB implementation. They do not include any margin for crosstalk.

☞ Altera recommends that you get accurate time base skew numbers when you simulate your specific implementation.

**Table 4–3. RLDRAM II Layout Guidelines (Part 1 of 2)**

| Parameter | Guidelines |
|---|---|
| Impedance | ■ All signal planes must be 50 Ω, single-ended, ±10%.<br>■ All signal planes must be 100 Ω, differential ±10%.<br>■ Remove all unused via pads, because they cause unwanted capacitance. |
| Decoupling Parameter | ■ Use 0.1 µF in 0402 size to minimize inductance.<br>■ Make $V_{TT}$ voltage decoupling close to pull-up resistors.<br>■ Connect decoupling caps between $V_{TT}$ and ground.<br>■ Use a 0.1 µF cap for every other $V_{TT}$ pin.<br>■ Verify your capacitive decoupling using the Altera Power Distribution Network (PDN) Design tool. |
| Power | ■ Route GND, 1.5 V/1.8 V as planes.<br>■ Route $V_{CCIO}$ for memories in a single split plane with at least a 20-mil (0.020 inches or 0.508 mm) gap of separation.<br>■ Route $V_{TT}$ as islands or 250-mil (6.35-mm) power traces.<br>■ Route oscillators and PLL power as islands or 100-mil (2.54-mm) power traces. |
| General Routing | ■ All specified delay matching requirements include PCB trace delays, different layer propagation, velocity variance, and crosstalk. To minimize PCB layer propagation variance, Altera recommends that signals from the same net group always be routed on the same layer. If you must route signals of the same net group on different layers with the same impedance characteristic, simulate your worst case PCB trace tolerances to ascertain actual propagation delay differences. Typical layer to layer trace delay variations are of 15 ps/inch order.<br>■ Use 45° angles (not 90° corners).<br>■ Avoid T-Junctions for critical nets or clocks.<br>■ Avoid T-junctions greater than 150 ps (approximately 500 mils, 12.7 mm).<br>■ Disallow signals across split planes.<br>■ Restrict routing other signals close to system reset signals.<br>■ Avoid routing memory signals closer than 0.025 inch (0.635 mm) to PCI or system clocks.<br>■ Match all signals within a given DQ group with a maximum skew of ±10 ps or approximately ±50 mils (0.254 mm) and route on the same layer. |
| Clock Routing | ■ Route clocks on inner layers with outer-layer run lengths held to under 150 ps (approximately 500 mils, 12.7 mm).<br>■ These signals should maintain a 10-mil (0.254 mm) spacing from other nets.<br>■ Clocks should maintain a length-matching between clock pairs of ±5 ps or approximately ±25 mils (0.635 mm).<br>■ Differential clocks should maintain a length-matching between `P` and `N` signals of ±2 ps or approximately ±10 mils (0.254 mm).<br>■ Space between different clock pairs should be at least three times the space between the traces of a differential pair. |

**Table 4–3. RLDRAM II Layout Guidelines (Part 2 of 2)**

| Parameter | Guidelines |
|---|---|
| Address and Command Routing | ■ To minimize crosstalk, route address, bank address, and command signals on a different layer than the data and data mask signals.<br>■ Do not route the differential clock signals close to the address signals.<br>■ Keep the distance from the pin on the RLDRAM II component to the stub termination resistor ($V_{TT}$) to less than 50 ps (approximately 250 mils, 6.35 mm) for the address/command signal group.<br>■ Keep the distance from the pin on the RLDRAM II component to the fly-by termination resistor ($V_{TT}$) to less than 100 ps (approximately 500 mils, 12.7 mm) for the address/command signal group. |
| External Memory Routing Rules | ■ Apply the following parallelism rules for the RLDRAM II data/address/command groups:<br>■ 4 mils for parallel runs < 0.1 inch (approximately 1× spacing relative to plane distance).<br>■ 5 mils for parallel runs < 0.5 inch (approximately 1× spacing relative to plane distance).<br>■ 10 mils for parallel runs between 0.5 and 1.0 inches (approximately 2× spacing relative to plane distance).<br>■ 15 mils for parallel runs between 1.0 and 3.3 inch (approximately 3× spacing relative to plane distance). |
| Maximum Trace Length | ■ Keep the maximum trace length of all signals from the FPGA to the RLDRAM II components to 600 ps (approximately 3,300 mils, 83.3 mm). |

Using the layout guidelines in Table 4–3, Altera recommends the following layout approach:

1. If the RLDRAM II interface has multiple DQ groups (×18 or ×36 RLDRAM II component or width expansion configuration), match all the DK/DK# and QK,QK# clocks as tightly as possible to optimize the timing margins in your design.
2. Route the DK/DK# write clock and QK/QK# read clock associated with a DQ group on the same PCB layer. Match these clock pairs to within ±5 ps.
3. Set the DK/DK# or QK,QK# clock as the target trace propagation delay for the associated data and data mask signals.
4. Route the data and data mask signals for the DQ group ideally on the same layer as the associated QK/QK# and DK/DK# clocks to within ±10 ps skew of the target clock.
5. Route the CK/CK# clocks and set as the target trace propagation delays for the address/command signal group. Match the CK/CK# clock to within ±50 ps of all the DK/DK# clocks.
6. Route the address/control signal group (address, bank address, CS, WE , and REF) ideally on the same layer as the CK/CK# clocks, to within ±20 ps skew of the CK/CK# traces.

This layout approach provides a good starting point for a design requirement of the highest clock frequency supported for the RLDRAM II interface .

Table 4–4 shows the typical margins for RLDRAM II interfaces, with the assumption that there is zero skew between the signal groups.

**Table 4–4. Typical Worst Case Margins for CIO RLDRAM II Interfaces**

| Device | Speed Grade | Frequency (MHz) | Typical Margin Address/Command (ps) | Typical Margin Write Data (ps) | Typical Margin Read Data (ps) |
|---|---|---|---|---|---|
| Stratix IV | — | 400 | — | — | — |
| Stratix V | — | 400 | — | — | — |

Other devices and speed grades typically show higher margins than those in Table 4–4.

Altera recommends that you create your project in the Quartus® II software with a fully implemented RLDRAM II Controller with UniPHY interface, and observe the interface timing margins to determine the actual margins for your design.

Although the recommendations in this chapter are based on simulations, you can apply the same general principles when determining the best termination scheme, drive strength setting, and loading style to any board designs. Even armed with this knowledge, it is still critical that you perform simulations, either using IBIS or HSPICE models, to determine the quality of signal integrity on your designs.

# 5. QDR II SRAM Interface Termination and Layout Guidelines

This chapter provides guidelines for you to improve your system's signal integrity and layout guidelines to help successfully implement a QDR II or QDR II+ SRAM interface in your system.

The QDR II and QDR II+ SRAM Controller with UniPHY intellectual property (IP) enables you to implement QDR II and QDR II+ interfaces with Arria II GX, Stratix III, and Stratix IV devices.

In this chapter, QDR II SRAM refers to both QDR II and QDR II+ SRAM unless stated otherwise.

This chapter focuses on the following key factors that affect signal integrity:

- I/O standards
- QDR II SRAM configurations
- Signal terminations
- Printed circuit board (PCB) layout guidelines

## I/O Standards

QDR II SRAM interface signals use one of the following JEDEC I/O signalling standards:

- HSTL-15—provides the advantages of lower power and lower emissions.
- HSTL-18—provides increased noise immunity with slightly greater output voltage swings.

To select the most appropriate standard for your interface, refer to the *Arria II GX Devices Data Sheet : Electrical Characteristics* chapter in the *Arria II Device Handbook*, *Stratix III Device Datasheet: DC and Switching Characteristics* chapter in the *Stratix III Device Handbook*, or the *Stratix IV Device Datasheet DC and Switching Characteristics* chapter in the *Stratix IV Device Handbook*.

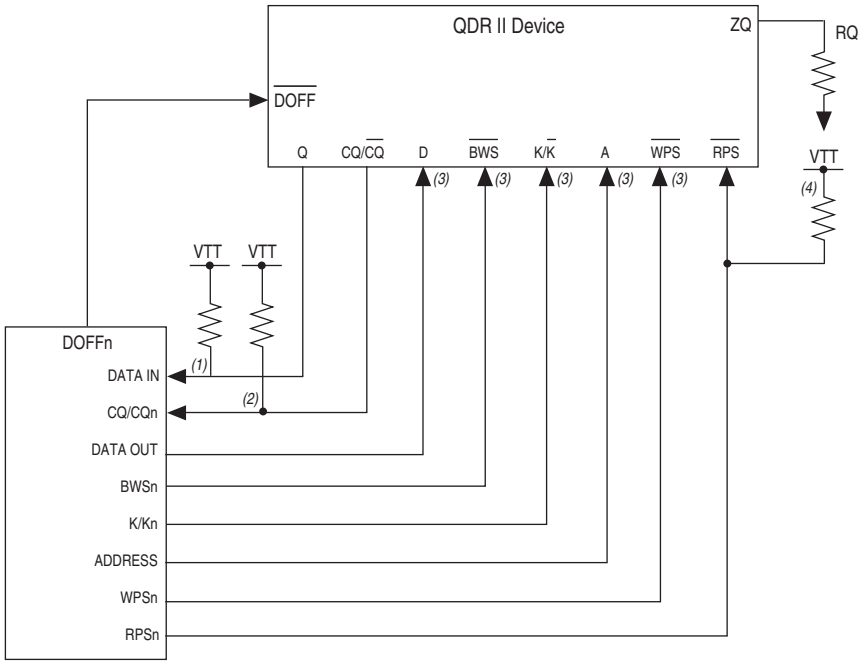
Altera QDR II SRAM Controller with UniPHY IP defaults to HSTL 1.5 V Class I outputs and HSTL 1.5 V inputs.

## QDR II SRAM Configurations

The QDR II SRAM Controller with UniPHY IP supports interfaces with a single device, and two devices in a width expansion configuration up to maximum width of 72 bits.

Figure 5–1 shows the main signal connections between the FPGA and a single QDR II SRAM component.

**Figure 5–1. Configuration With A Single QDR II SRAM Component**




**Notes to Figure 5–1:**

(1) Use external discrete termination only for data inputs targeting Arria II GX devices that do not support parallel OCT. For Stratix III and Stratix IV devices, use parallel OCT.

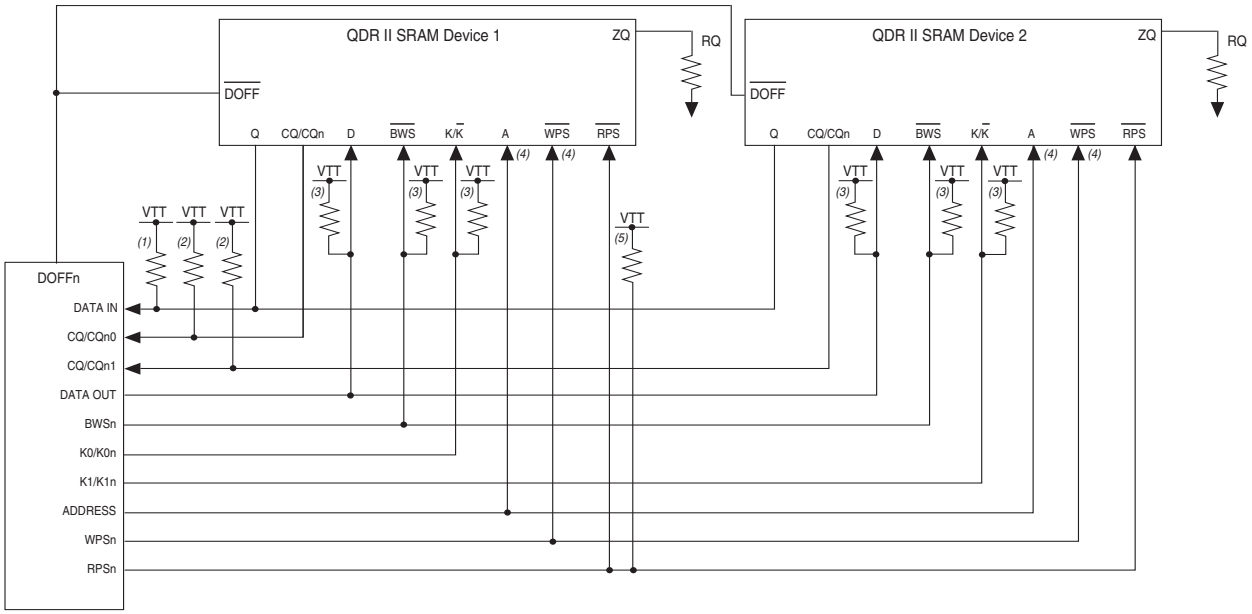
(2) Use external discrete termination only for CQ/CQ# targeting Arria II GX devices, or for any device using ×36 emulated mode.

(3) Use external discrete termination for this signal, as shown for RPS.

(4) Use external discrete termination with fly-by placement to avoid stubs.

Figure 5–2 shows the main signal connections between the FPGA and two QDR II SRAM components in a width expansion configuration.

**Figure 5–2. Configuration With Two QDR II SRAM Components In A Width Expansion Configuration**




**Notes to Figure 5–2:**

(1) Use external discrete termination only for data inputs targeting Arria II GX devices that do not support parallel OCT. For Stratix III and Stratix IV devices, use parallel OCT.

(2) Use external discrete termination only for CQ/CQ# targeting Arria II GX devices, or for any device using ×36 emulated mode.

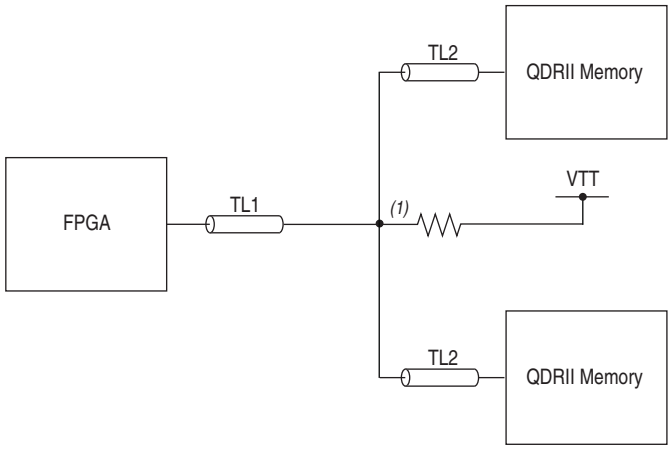
(3) Use external discrete termination for data outputs, BWSn, and K/K# clocks with fly-by placement to avoid stubs.

(4) Use external discrete termination for this signal, as shown for RPS.

(5) Use external discrete termination at the trace split of the balanced T or Y topology.

Figure 5–3 shows the detailed balanced topology recommended for the address and command signals in the width expansion configuration.

**Figure 5–3. External Parallel Termination for Balanced Topology**




**Note to Figure 5–3:**

(1) To minimize the reflections and parallel impedance discontinuity seen by the signal, place the trace split close to the QDR II SRAM memory components. Keep TL2 short so that the QDR II SRAM components appear as a lumped load.

# Signal Terminations

Arria II GX, Stratix III and Stratix IV devices offer on-chip termination (OCT) technology.

Table 5–1 summarizes the extent of OCT support for each device.

**Table 5–1. On-Chip Termination Schemes *(Note 1)***

| Termination Scheme | HSTL-15 and HSTL-18 | FPGA Device | | | |
|---|---|---|---|---|---|
| | | Stratix III and Stratix IV | | Arria II GX | |
| | | Column I/O | Row I/O | Column I/O | Row I/O |
| On-Chip Series Termination without Calibration | Class I | 50 | 50 | 50 | 50 |
| On-Chip Series Termination with Calibration | Class I | 50 | 50 | 50 | 50 |
| On-Chip Parallel Termination with Calibration | Class I | 50 | 50 | — | — |

**Note to Table 5–1:**

(1) This table provides information about HSTL-15 and HSTL-18 standards because these are the supported I/O standards for QDR II SRAM memory interfaces by Altera FPGAs.

On-chip series ($R_S$) termination is supported only on output and bidirectional buffers, while on-chip parallel ($R_T$) termination is supported only on input and bidirectional buffers. Because QDR II SRAM interfaces have unidirectional data paths, dynamic OCT is not required.

For Arria II GX, Stratix III and Stratix IV devices , the HSTL Class I I/O calibrated terminations are calibrated against 50 Ω 1% resistors connected to the $R_{UP}$ and $R_{DN}$ pins in an I/O bank with the same VCCIO as the QDRII SRAM interface. The calibration occurs at the end of the device configuration.

QDR II SRAM controllers have a ZQ pin which is connected via a resistor RQ to ground. Typically the QDR II SRAM output signal impedance is 0.2 × RQ. Refer to the QDR II SRAM device data sheet for more information.

For information about OCT, refer to the *I/O Features in Arria II GX Devices* chapter in the *Arria II GX Device Handbook*, *Stratix III Device I/O Features* chapter in the *Stratix III Device Handbook* and the *I/O Features in Stratix IV Devices* chapter in the *Stratix IV Device Handbook*.

The following section shows HyperLynx simulation eye diagrams to demonstrate signal termination options. Altera strongly recommends signal terminations to optimize signal integrity and timing margins, and to minimize unwanted emissions, reflections, and crosstalk.

All of the eye diagrams shown in this section are for a 50 Ω trace with a propagation delay of 720 ps which is approximately a 4-inch trace on a standard FR4 PCB. The signal I/O standard is HSTL-15.

For point-to-point signals, Altera recommends that you place a fly-by termination by terminating at the end of the transmission line after the receiver to avoid unterminated stubs. The guideline is to place the fly-by termination within 100 ps propagation delay of the receiver.

Although not recommended, you can place the termination before the receiver, which leaves an unterminated stub. The stub delay is critical because the stub between the termination and the receiver is effectively unterminated, causing additional ringing and reflections. Stub delays should be less than 50 ps.

The eye diagrams shown in this section show the best case achievable and do not take into account PCB vias, crosstalk and other degrading effects such as variations in the PCB structure due to manufacturing tolerances.

Simulate your design to ensure correct functionality.

## Output from the FPGA to the QDR II SRAM Component

The following output signals are from the FPGA to the QDR II SRAM component:

- write data
- byte write select (`BWSn`)
- address
- control (`WPSn` and `RPSn`)
- clocks, `K/K#`

Altera recommends that you terminate the write clocks, `K` and `K#`, with a single-ended fly-by 50 Ω parallel termination to $V_{TT}$. However, simulations show that you can consider a differential termination if the clock pair is well matched and routed differentially.

The HyperLynx simulation eye diagrams show simulation cases of write data and address signals with termination options. The QDR II SRAM write data is double data rate. The QDR II SRAM address is either double data rate (burst length of 2) or single data rate (burst length of 4).

Simulations show that lowering the drive strength does not make a significant difference to the eye diagrams. All eye diagrams are shown at the QDR II SRAM device receiver pin.

Figure 5–4 shows the fly-by terminated signal using Stratix IV Class I HSTL-15 with calibrated 50 ΩOCT output driver.

**Figure 5–4. Write Data Simulation at 400 MHz with Fly-By 50 ΩParallel Termination to $V_{TT}$**

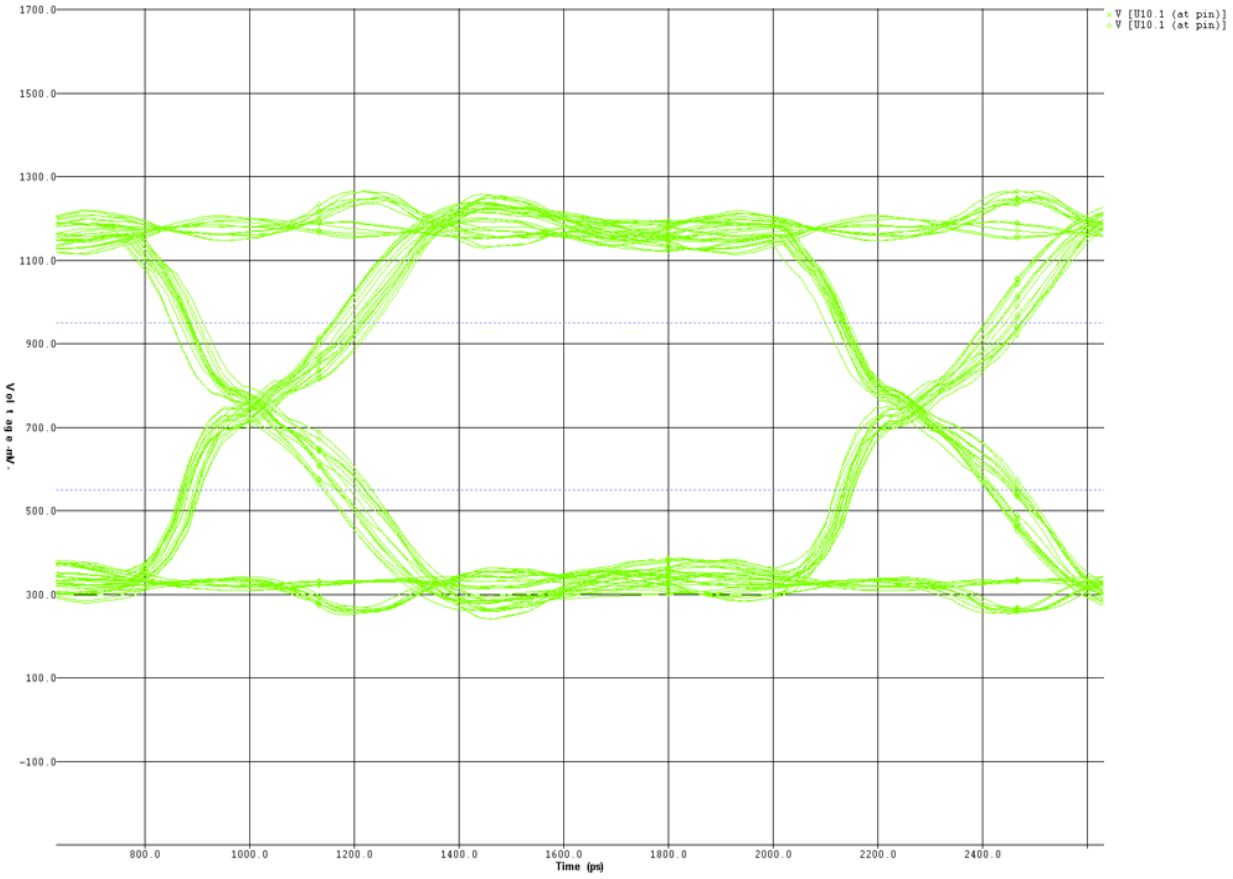


Figure 5–5 shows an unterminated signal using Stratix IV Class I HSTL-15 with a calibrated 50 Ω OCT output driver. This unterminated solution is not recommended.

**Figure 5–5. Write Data Simulation at 400 MHz with No Far-End Termination**

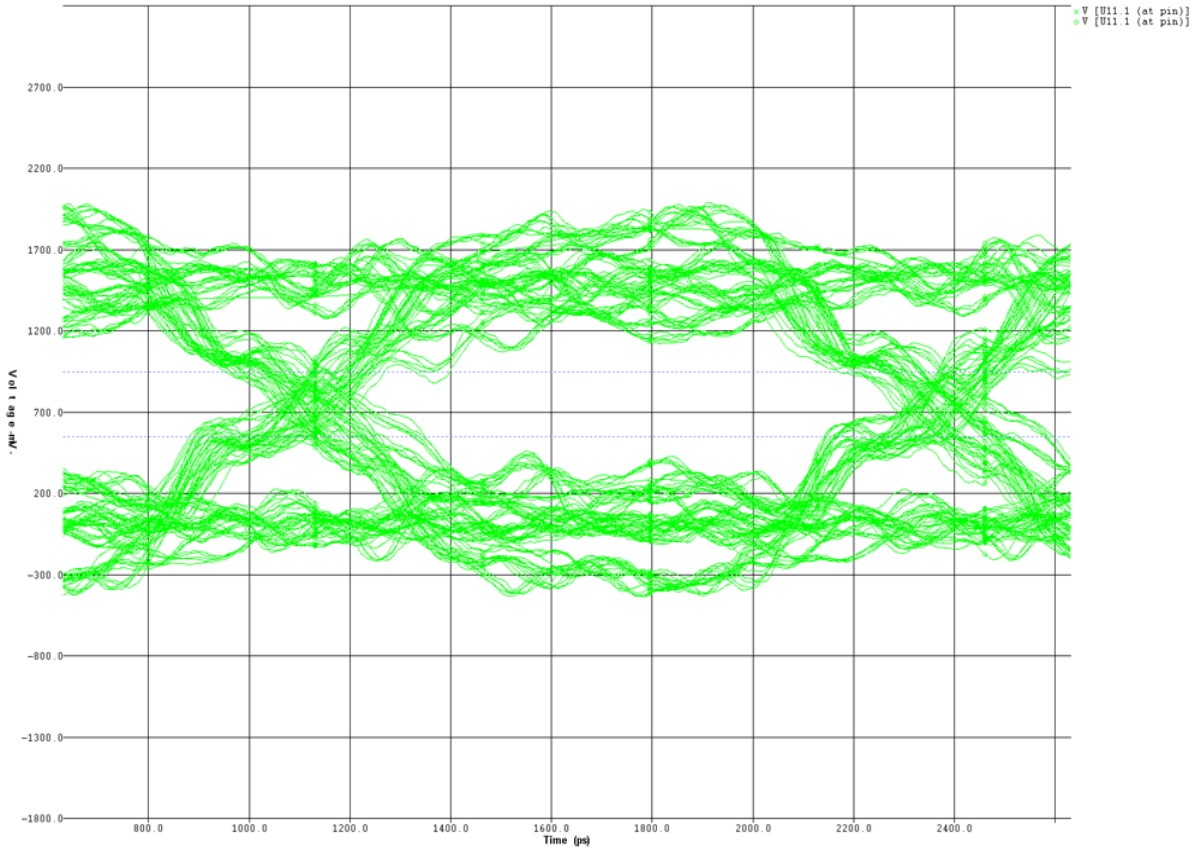


Figure 5–6 shows an unterminated signal at a lower frequency of 250 MHz using Arria II GX Class I HSTL-15 with calibrated 50 Ω OCT output driver. This unterminated solution may be passable for some systems, but is shown so that you can compare against the superior quality of the terminated signal in Figure 5–4.

**Figure 5–6. Write Data Simulation at 250 MHz with No Far-End Termination**

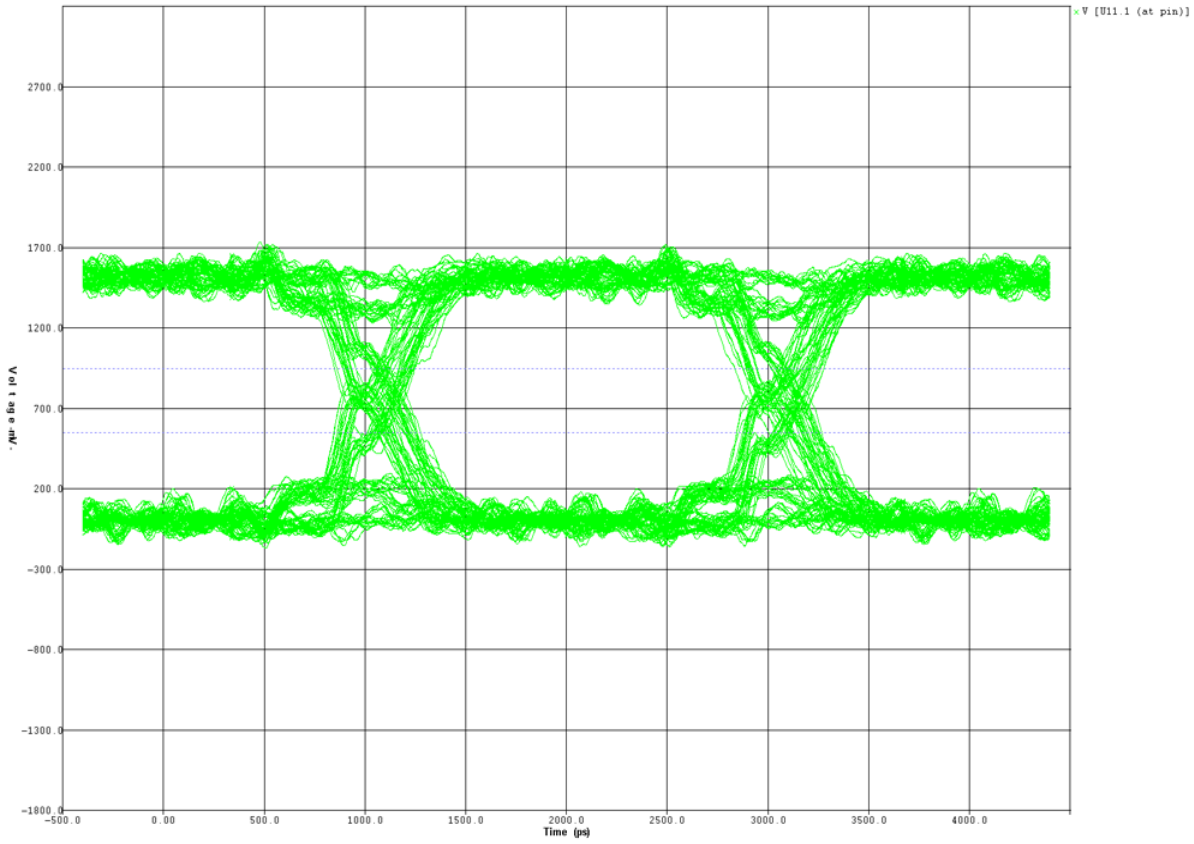


Figure 5–7 shows an unterminated signal at a frequency of 175 MHz with a point-to-point connection. QDR II SRAM interfaces using Stratix IV devices have a maximum supported frequency of 350 MHz. For QDR II SRAM with burst length of four interfaces, the address signals are effectively single date rate at 175 MHz. This unterminated solution is not recommended but can be considered. The FPGA output driver is Class I HSTL-15 with a calibrated 50 Ω OCT.

**Figure 5–7. Address Simulation for QDR II SRAM Burst Length of 4 at 175 MHz with No Far-End Termination**

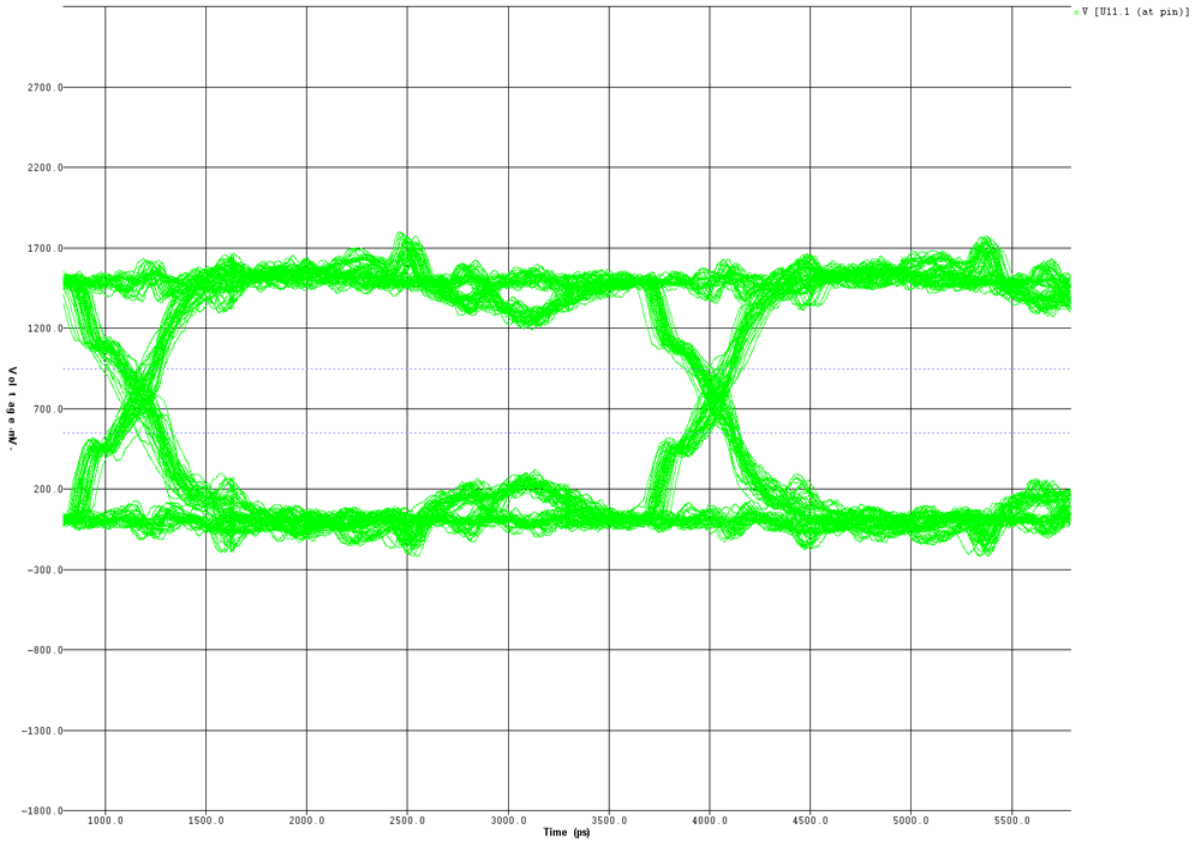
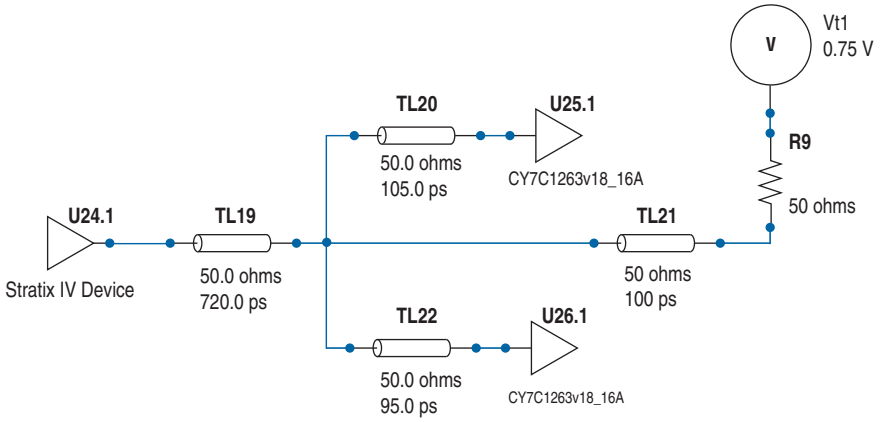


Figure 5–8 shows a typical topology, which are used for two components in width expansion mode. Altera recommends that you match the stubs TL20 and TL22, but you can allow small differences allowed to achieve acceptable signal integrity.

**Figure 5–8. Address for QDR II SRAM Burst Length of 2 in Width Expansion Mode Topology**




The eye diagrams in Figure 5–9 and Figure 5–10 use the topology shown in Figure 5–8. The eye diagram in Figure 5–11 uses the topology shown in Figure 5–8 without the $V_{TT}$ termination, R9 and TL21.

Figure 5–9 shows an address signal at a frequency of 400 MHz with parallel 50 Ω termination to $V_{TT}$ for QDR II SRAM burst length of 2 width expansion using Stratix IV Class I HSTL-15 12 mA driver and fly-by 50 Ω parallel termination to $V_{TT}$.

**Figure 5–9. Address Simulation Using Stratix IV Class I HSTL-15 12 mA Driver and Fly-by 50 Ω Parallel Termination to $V_{TT}$**

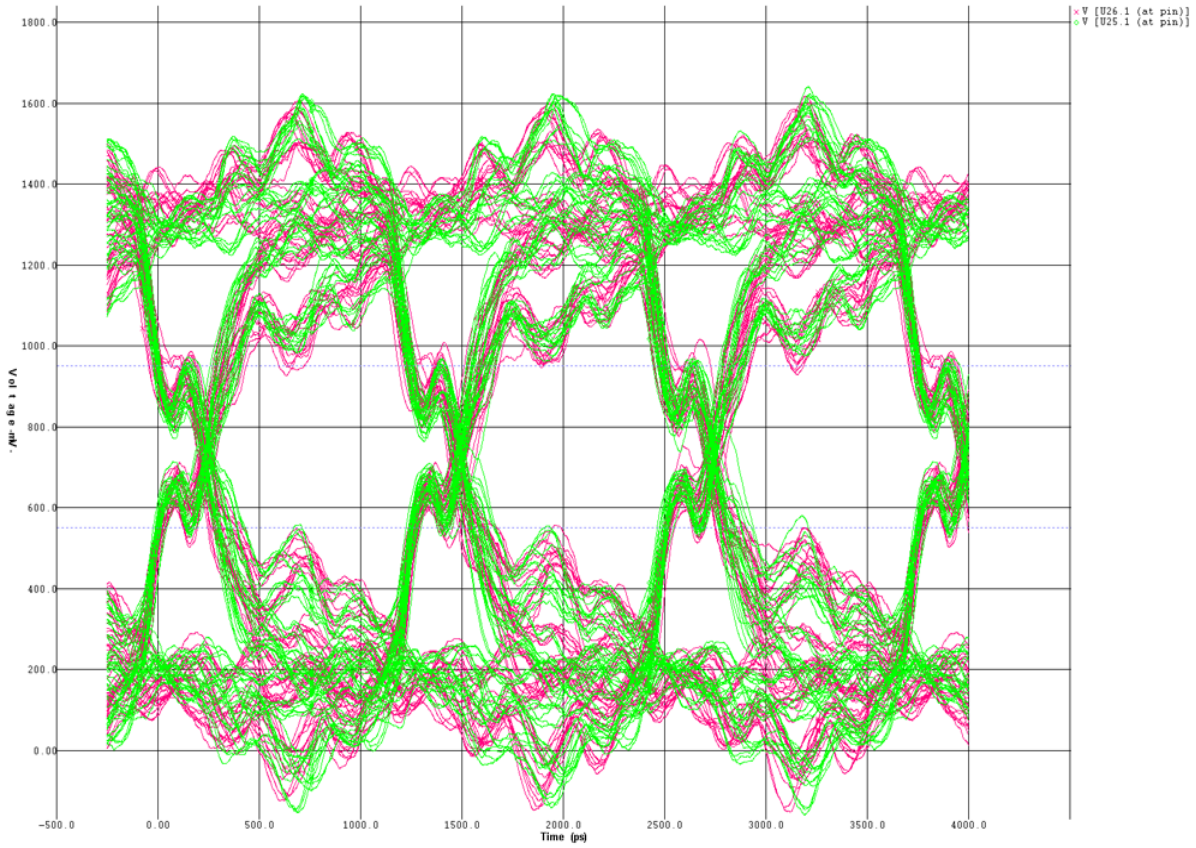


Figure 5–10 shows an address signal at a frequency of 400 MHz with parallel 50 Ω termination to $V_{TT}$ for QDR II SRAM burst length of 2 width expansion using Stratix IV Class I HSTL-15 with 50 Ω calibration driver and fly-by 50 Ω parallel termination to $V_{TT}$. The waveform eye is significantly improved compared to the maximum (12mA) drive strength case.

**Figure 5–10. Address Simulation Using Stratix IV Class I HSTL-15 50 Ω Calibration Driver and Fly-by 50 Ω Parallel Termination to $V_{TT}$**

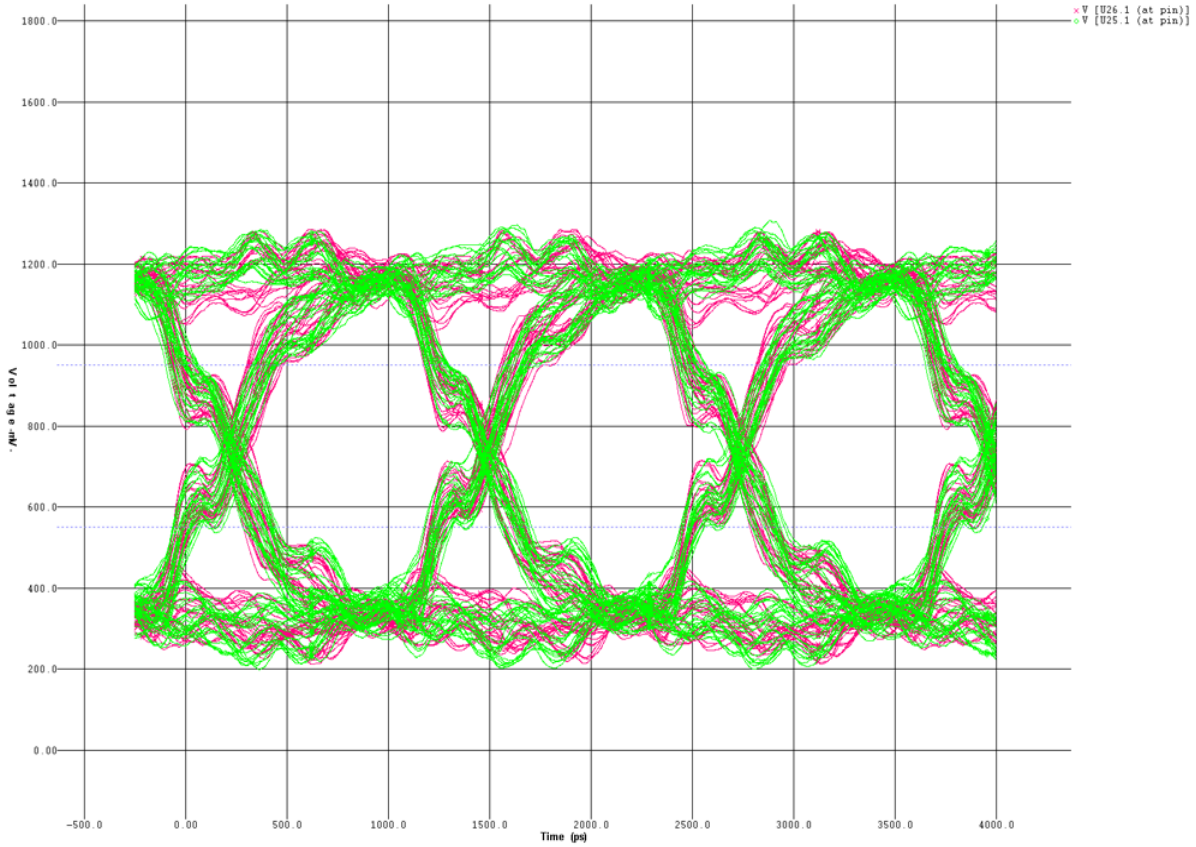
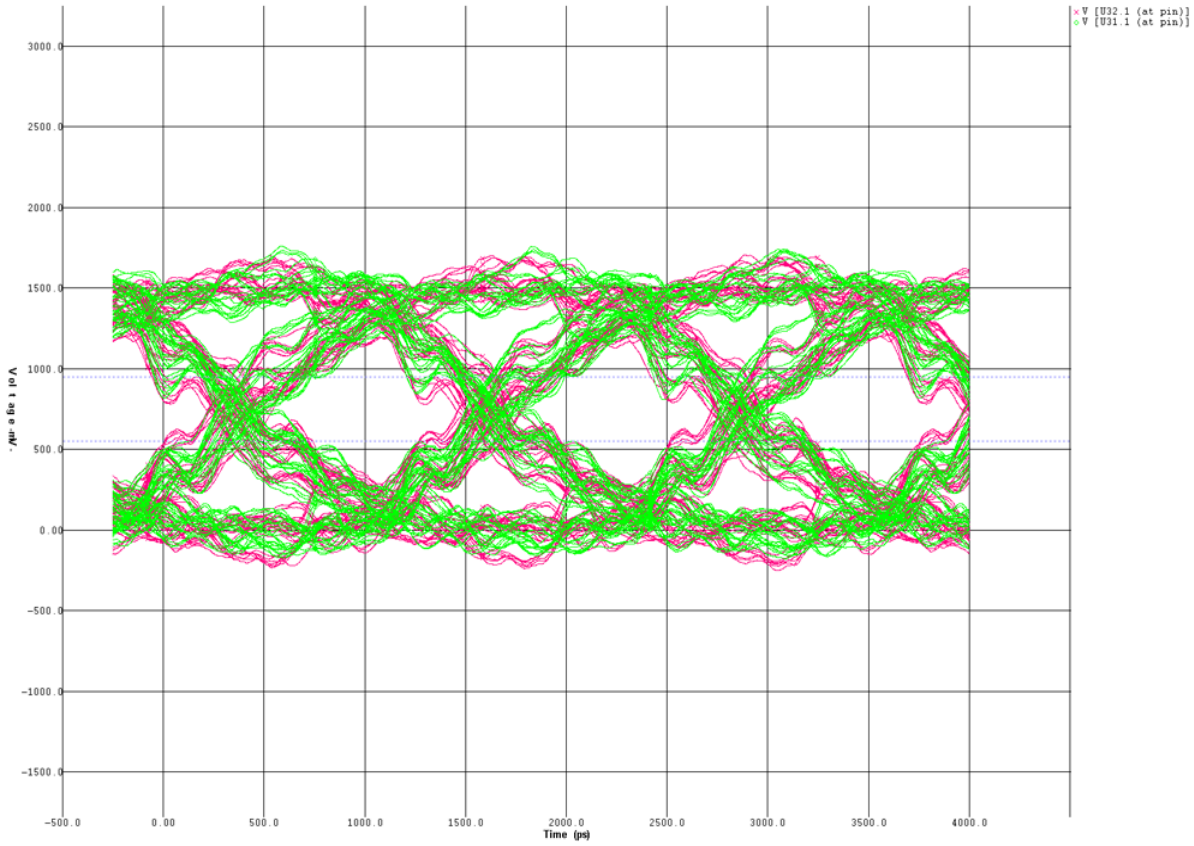


Figure 5–11 shows an unterminated address signal at a frequency of 400 MHz for QDR II SRAM burst length of 2 width expansion using Stratix IV Class I HSTL-15 with 50 Ω calibration driver. This unterminated address has small eye and is not recommended.

**Figure 5–11. Address Simulation Using Stratix IV Class I HSTL-15 50 Ω Calibration Driver and No Termination**




## Input to the FPGA from the QDR II SRAM Component

The QDR II SRAM component drives the following input signals into the FPGA:

- read data
- echo clocks, CQ/CQ#

For point-to-point signals, Altera recommends that you use the FPGA parallel OCT wherever possible. For devices that do not support parallel OCT (Arria II GX), and for ×36 emulated configuration CQ/CQ# termination, Altera recommends that you use a fly-by 50 Ω parallel termination to $V_{TT}$. Although not recommended, you can use parallel termination with a short stub of less that 50 ps propagation delay as an alternative option. The input echo clocks, CQ and CQ# must not use a differential termination.

The eye diagrams are shown at the FPGA receiver pin, and the QDR II SRAM output driver is Class I HSTL-15 using its ZQ calibration of 50 Ω The QDR II SRAM read data is double data rate.

Figure 5–12 shows the ideal case of a fly-by terminated signal using 50 Ω calibrated parallel OCT with Stratix IV device.

**Figure 5–12. Read Data Simulation at 400 MHz with 50 Ω Parallel OCT Termination**

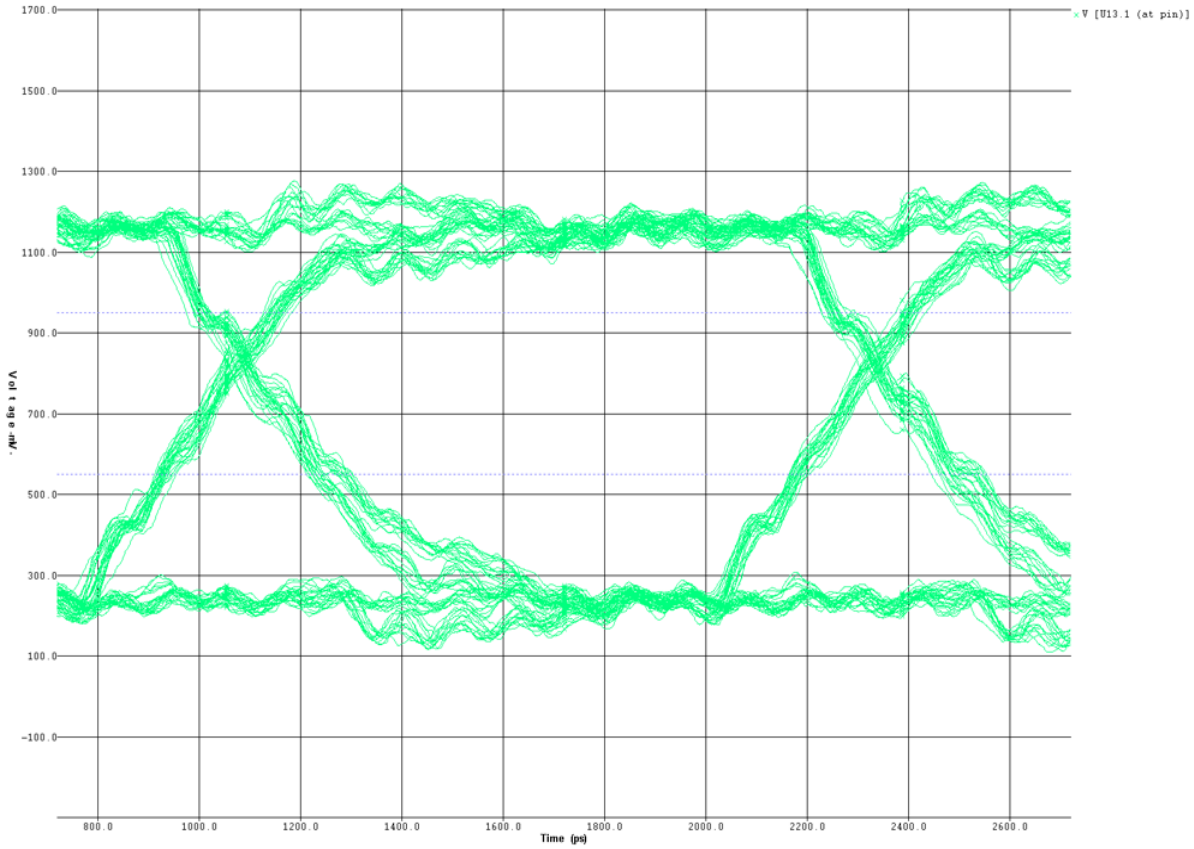


Figure 5–13 shows an external discrete component fly-by terminated signal at a lower frequency of 250 MHz using an Arria II GX device.

**Figure 5–13. Read Data Simulation at 250 MHz with Fly-By Parallel 50 Ω Termination**

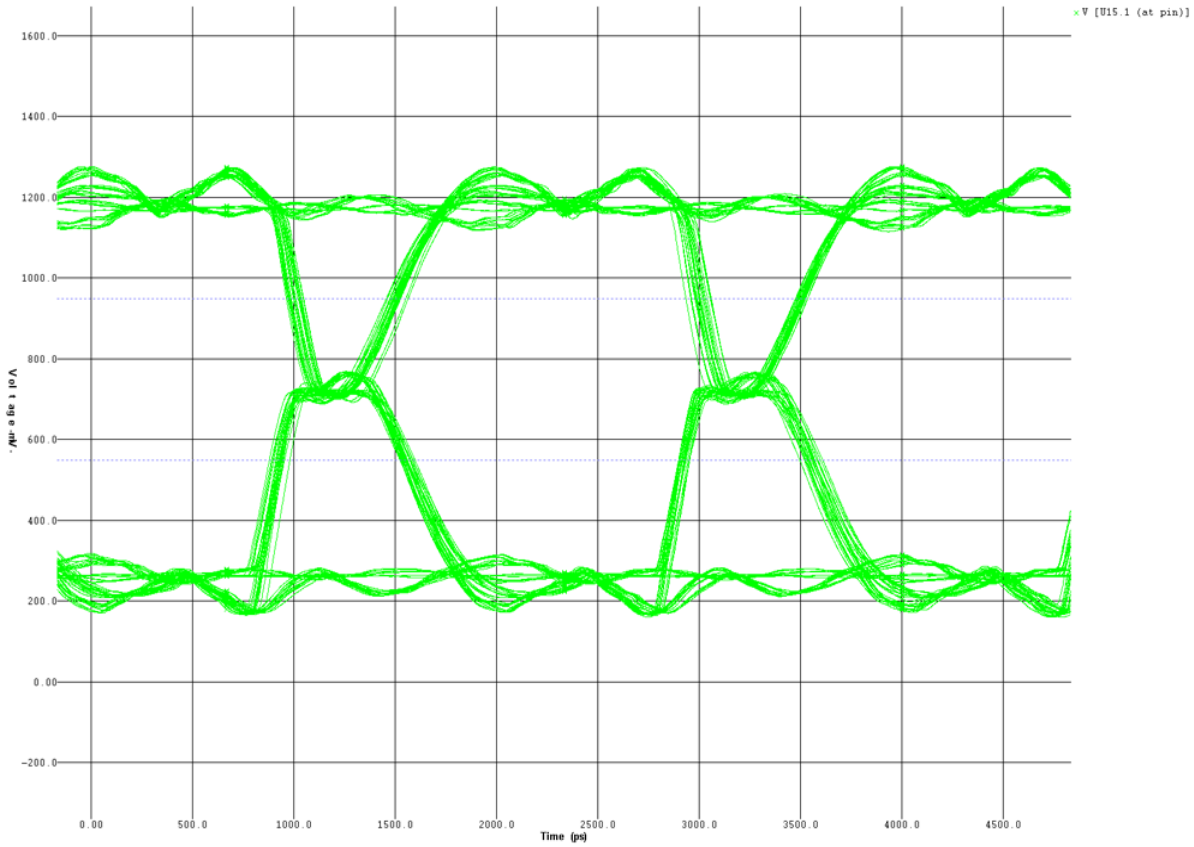
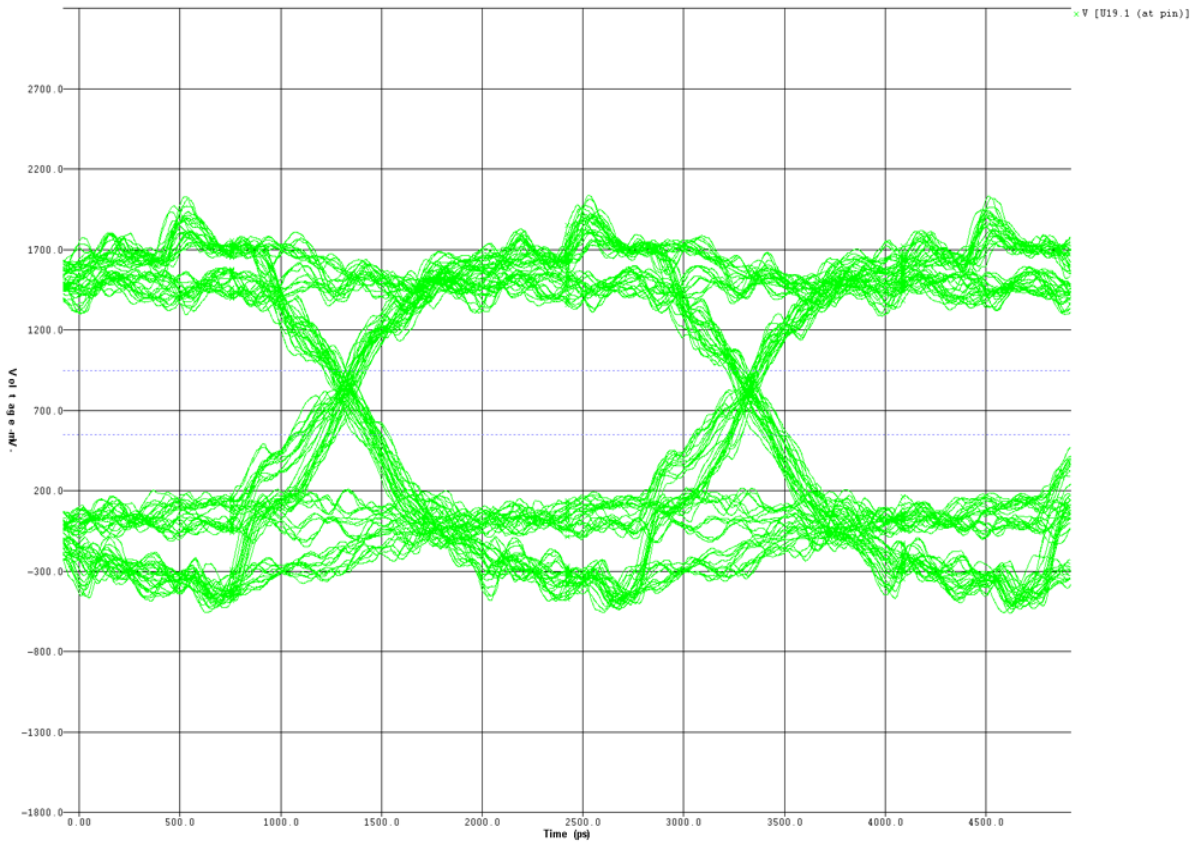


Figure 5–14 shows an unterminated signal at a lower frequency of 250 MHz using an Arria II GX device. This unterminated solution is not recommended but is shown so that you can compare against the superior quality of the terminated signal in Figure 5–13.

**Figure 5–14. Read Data Simulation at 250 MHz with No Far-End Termination**



## Termination Schemes

Table 5–2 and Table 5–3 provide the recommended termination schemes for major QDR II SRAM memory interface signals, which include write data (D), byte write select (BWS), read data (Q), clocks (K, K#, CQ, and CQ#), address and command (WPS and RPS).

**Table 5–2. Termination Recommendations for Arria II GX Devices**

| Signal Type | HSTL 15/18 Standard *(1)*, *(2)* | FPGA End Discrete Termination | Memory End Termination |
|---|---|---|---|
| K/K# Clocks | Class I R50 CAL | — | 50 Ω Parallel to $V_{TT}$ |
| Write Data | Class I R50 CAL | — | 50 Ω Parallel to $V_{TT}$ |
| BWS | Class I R50 CAL | — | 50 Ω Parallel to $V_{TT}$ |
| Address *(3)*, *(4)* | Class I Max Current | — | 50 Ω Parallel to $V_{TT}$ |
| WPS, RPS *(3)*, *(4)* | Class I Max Current | — | 50 Ω Parallel to $V_{TT}$ |
| CQ/CQ# | Class I | 50Ω Parallel to $V_{TT}$ | ZQ50 |
| CQ/CQ# ×36 emulated *(5)* | Class I | 50 Ω Parallel to $V_{TT}$ | ZQ50 |
| Read Data (Q) | Class I | 50 Ω Parallel to $V_{TT}$ | ZQ50 |
| QVLD *(6)* | — | — | ZQ50 |

**Notes to Table 5–2:**

(1) R is effective series output impedance.

(2) CAL is calibrated OCT.

(3) For width expansion configuration, the address and control signals are routed to 2 devices. Recommended termination is 50 Ω parallel to $V_{TT}$ at the trace split of a balanced T or Y routing topology. For 400 MHz burst length 2 configurations where the address signals are double data rate, it is recommended to use a clamshell placement of the two QDR II SRAM components to achieve minimal stub delays and optimum signal integrity. Clamshell placement is when two devices overlay each other by being placed on opposite sides of the PCB.

(4) The UniPHY default IP setting for this output is Max Current. A Class I 50 Ω output with calibration output is typically optimal in single load topologies.

(5) For ×36 emulated mode, the recommended termination for the CQ/CQ# signals is a 50 Ω parallel termination to $V_{TT}$ at the trace split, refer to Figure 5–15. Altera recommends that you use this termination when ×36 DQ/DQS groups are not supported in the FPGA.

(6) QVLD is not used in the QDR II or QDR II+ SRAM with UniPHY implementations.

**Table 5–3. Termination Recommendations for Stratix III and Stratix IV Devices (Part 1 of 2)**

| Signal Type | HSTL 15/18 Standard *(1)*, *(2)*, *(3)* | FPGA End Discrete Termination | Memory End Termination |
|---|---|---|---|
| K/K# Clocks | Class I R50 CAL | — | 50 Ω Parallel to $V_{TT}$ |
| Write Data | Class I R50 CAL | — | 50 Ω Parallel to $V_{TT}$ |
| BWS | Class I R50 CAL | — | 50 Ω Parallel to $V_{TT}$ |
| Address *(4)*, *(5)* | Class I Max Current | — | 50 Ω Parallel to $V_{TT}$ |
| WPS, RPS *(4)*, *(5)* | Class I Max Current | — | 50 Ω Parallel to $V_{TT}$ |
| CQ/CQ# | Class I P50 CAL | — | ZQ50 |
| CQ/CQ# ×36 emulated *(6)* | — | 50 Ω Parallel to $V_{TT}$ | ZQ50 |
| Read Data (Q) | Class I P50 CAL | — | ZQ50 |

**Table 5–3. Termination Recommendations for Stratix III and Stratix IV Devices (Part 2 of 2)**

| Signal Type | HSTL 15/18 Standard *(1)*, *(2)*, *(3)* | FPGA End Discrete Termination | Memory End Termination |
|---|---|---|---|
| QVLD *(7)* | Class I P50 CAL | — | ZQ50 |

**Notes to Table 5–3:**

(1) R is effective series output impedance.

(2) P is effective parallel input impedance.

(3) CAL is calibrated OCT.

(4) For width expansion configuration, the address and control signals are routed to 2 devices. Recommended termination is 50 Ω parallel to $V_{TT}$ at the trace split of a balanced T or Y routing topology. For 400 MHz burst length 2 configurations where the address signals are double data rate, it is recommended to use a "clam shell" placement of the two QDR II SRAM components to achieve minimal stub delays and optimum signal integrity. "Clam shell" placement is when two devices overlay each other by being placed on opposite sides of the PCB.

(5) The UniPHY default IP setting for this output is Max Current. A Class 1 50 Ω output with calibration output is typically optimal in single load topologies.

(6) For ×36 emulated mode, the recommended termination for the CQ/CQ# signals is a 50 Ω parallel termination to $V_{TT}$ at the trace split, refer to Figure 5–15. Altera recommends that you use this termination when ×36 DQ/DQS groups are not supported in the FPGA.

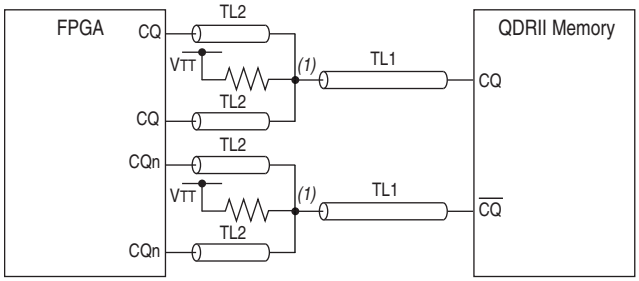
(7) QVLD is not used in the QDR II or QDR II+ SRAM Controller with UniPHY implementations.

Altera recommends that you simulate your specific design for your system to ensure good signal integrity.

The QDR II SRAM Controller with UniPHY IP does not support the on-die termination (ODT) functionality featured in some QDR II SRAM components.

For a ×36 QDR II SRAM interface that uses an emulated mode of two ×18 DQS groups in the FPGA, there are two CQ/CQ# connections at the FPGA and a single CQ/CQ# output from the QDR II SRAM device. Altera recommends that you use a balanced T topology with the trace split close to the FPGA and a parallel termination at the split, as shown in Figure 5–15.

**Figure 5–15. Emulated ×36 Mode CQ/CQn Termination Topology**




**Note to Figure 5–15:**

(1) To minimize the reflections and parallel impedance discontinuity seen by the signal, place the trace split close to the FPGA device. Keep TL2 short so that the FPGA inputs appear as a lumped load.

For more information about ×36 emulated modes, refer to *Exceptions for ×36 Emulated QDR II and QDR II+ SRAM Interfaces in Arria II GX, Stratix III, and Stratix IV Devices* in the *Device and Pin Planning* chapter in volume 2 of the *External Memory Interface Handbook*.

# PCB Layout Guidelines

Table 5–4 summarizes QDR II and QDR II SRAM general routing layout guidelines.

The following layout guidelines include several +/- length based rules. These length based guidelines are for first order timing approximations if you cannot simulate the actual delay characteristics of your PCB implementation. They do not include any margin for crosstalk.

Altera recommends that you get accurate time base skew numbers when you simulate your specific implementation.

**Table 5–4. QDR II and QDR II+ SRAM Layout Guidelines (Part 1 of 2)**

| Parameter | Guidelines |
|---|---|
| Impedance | ■ All signal planes must be 50 Ω, single-ended, ±10%.<br>■ All signal planes must be 100 Ω, differential ±10%.<br>■ Remove all unused via pads, because they cause unwanted capacitance. |
| Decoupling Parameter | ■ Use 0.1 µF in 0402 size to minimize inductance.<br>■ Make $V_{TT}$ voltage decoupling close to pull-up resistors.<br>■ Connect decoupling caps between $V_{TT}$ and ground.<br>■ Use a 0.1 µF cap for every other $V_{TT}$ pin.<br>■ Verify your capacitive decoupling using the Altera Power Distribution Network (PDN) Design tool. |
| Power | ■ Route GND, 1.5 V/1.8 V as planes.<br>■ Route $V_{CCIO}$ for memories in a single split plane with at least a 20-mil (0.020 inches or 0.508 mm) gap of separation.<br>■ Route $V_{TT}$ as islands or 250-mil (6.35-mm) power traces.<br>■ Route all oscillators and PLL power as islands or 100-mil (2.54-mm) power traces. |
| General Routing | ■ All specified delay matching requirements include PCB trace delays, different layer propagation, velocity variance, and crosstalk. To minimize PCB layer propagation variance, Altera recommends that signals from the same net group always be routed on the same layer. If signals of the same net group must be routed on different layers with the same impedance characteristic, you must simulate your worst case PCB trace tolerances to ascertain actual propagation delay differences. Typical later to later trace delay variations are of 15 ps/inch order.<br>■ Use 45° angles (not 90° corners).<br>■ Avoid T-Junctions for critical nets or clocks.<br>■ Avoid T-junctions greater than 150 ps (approximately 500 mils, 12.7 mm).<br>■ Disallow signals across split planes.<br>■ Restrict routing other signals close to system reset signals.<br>■ Avoid routing memory signals closer than 0.025 inch (0.635 mm) to PCI or system clocks. |

**Table 5–4. QDR II and QDR II+ SRAM Layout Guidelines (Part 2 of 2)**

| Parameter | Guidelines |
|---|---|
| Clock Routing | ■ Route clocks on inner layers with outer-layer run lengths held to under 150 ps (approximately 500 mils, 12.7 mm).<br>■ These signals should maintain a 10-mil (0.254 mm) spacing from other nets.<br>■ Clocks should maintain a length-matching between clock pairs of ±5 ps or approximately ±25 mils (0.635 mm).<br>■ Complementary clocks should maintain a length-matching between P and N signals of ±2 ps or approximately ±10 mils (0.254 mm).<br>■ Keep the distance from the pin on the QDR II SRAM component to stub termination resistor ($V_{TT}$) to less than 50 ps (approximately 250 mils, 6.35 mm) for the K, K# clocks.<br>■ Keep the distance from the pin on the QDR II SRAM component to fly-by termination resistor ($V_{TT}$) to less than 100 ps (approximately 500 mils, 12.7 mm) for the K, K# clocks.<br>■ Keep the distance from the pin on the FPGA component to stub termination resistor ($V_{TT}$) to less than 50 ps (approximately 250 mils, 6.35 mm) for the echo clocks, CQ, CQ#, if they require an external discrete termination.<br>■ Keep the distance from the pin on the FPGA component to fly-by termination resistor ($V_{TT}$) to less than 100 ps (approximately 500 mils, 12.7 mm) for the echo clocks, CQ, CQ#, if they require an external discrete termination. |
| External Memory Routing Rules | ■ Keep the distance from the pin on the QDR II SRAM component to stub termination resistor ($V_{TT}$) to less than 50 ps (approximately 250 mils, 6.35 mm) for the write data, byte write select and address/command signal groups.<br>■ Keep the distance from the pin on the QDR II SRAM component to fly-by termination resistor ($V_{TT}$) to less than 100 ps (approximately 500 mils, 12.7 mm) for the write data, byte write select and address/command signal groups.<br>■ Keep the distance from the pin on the FPGA (Arria II GX) to stub termination resistor ($V_{TT}$) to less than 50 ps (approximately 250 mils, 6.35 mm) for the read data signal group.<br>■ Keep the distance from the pin on the FPGA (Arria II GX) to fly-by termination resistor ($V_{TT}$) to less than 100 ps (approximately 500 mils, 12.7 mm) for the read data signal group.<br>■ Parallelism rules for the QDR II SRAM data/address/command groups are as follows:<br>  ■ 4 mils for parallel runs < 0.1 inch (approximately 1× spacing relative to plane distance).<br>  ■ 5 mils for parallel runs < 0.5 inch (approximately 1× spacing relative to plane distance).<br>  ■ 10 mils for parallel runs between 0.5 and 1.0 inches (approximately 2× spacing relative to plane distance).<br>  ■ 15 mils for parallel runs between 1.0 and 6.0 inch (approximately 3× spacing relative to plane distance). |
| Maximum Trace Length | ■ Keep the maximum trace length of all signals from the FPGA to the QDR II SRAM components to 6 inches. |

Using the layout guidelines in Table 5–4, Altera recommends the following layout approach:

1. Route the K/K# clocks and set the clocks as the target trace propagation delays for the output signal group.
2. Route the write data output signal group (write data, byte write select), ideally on the same layer as the K/K# clocks, to within ±10 ps skew of the K/K# traces.

3. Route the address/control output signal group (address, RPS, WPS), ideally on the same layer as the K/K# clocks, to within ±20 ps skew of the K/K# traces.
4. Route the CQ/CQ# clocks and set the clocks as the target trace propagation delays for the input signal group.
5. Route the read data output signal group (read data), ideally on the same layer as the CQ/CQ# clocks, to within ±10 ps skew of the CQ/CQ# traces.
6. The output and input groups do not need to have the same propagation delays, but they must have all the signals matched closely within the respective groups.

Table 5–5 and Table 5–6 show the typical margins for QDR II and QDR II+ SRAM interfaces, with the assumption that there is zero skew between the signal groups.

**Table 5–5. Typical Worst Case Margins for QDR II SRAM Interfaces of Burst Length 2**

| Device | Speed Grade | Frequency (MHz) | Typical Margin Address/Command (ps) | Typical Margin Write Data (ps) | Typical Margin Read Data (ps) |
|---|---|---|---|---|---|
| Arria II GX | I5 | 250 | ± 240 | ± 80 | ± 170 |
| Arria II GX ×36 emulated | I5 | 200 | ± 480 | ± 340 | ± 460 |
| Stratix IV | — | 350 | — | — | — |
| Stratix IV ×36 emulated | C2 | 300 | ± 320 | ± 170 | ± 340 |

**Table 5–6. Typical Worst Case Margins for QDR II+ SRAM Interfaces of Burst Length 4**

| Device | Speed Grade | Frequency (MHz) | Typical Margin Address/Command (ps) *(1)* | Typical Margin Write Data (ps) | Typical Margin Read Data (ps) |
|---|---|---|---|---|---|
| Arria II GX | I5 | 250 | ± 810 | ± 150 | ± 130 |
| Arria II GX ×36 emulated | I5 | 200 | ± 1260 | ± 410 | ± 420 |
| Stratix IV | C2 | 400 | ± 550 | ± 10 | ± 80 |
| Stratix IV ×36 emulated | C2 | 300 | ± 860 | ± 180 | ± 300 |

**Note to Table 5–6:**

(1) The QDR II+ SRAM burst length of 4 designs have greater margins on the address signals because they are single data rate.

Other devices and speed grades typically show higher margins than the ones in Table 5–5 and Table 5–6 .

Altera recommends that you create your project with a fully implemented QDR II or QDR II+ SRAM Controller with UniPHY interface, and observe the interface timing margins to determine the actual margins for your design.

Although the recommendations in this chapter are based on simulations, you can apply the same general principles when determining the best termination scheme, drive strength setting, and loading style to any board designs. Even armed with this knowledge, it is still critical that you perform simulations, either using IBIS or HSPICE models, to determine the quality of signal integrity on your designs.

# 6. Power Estimation Methods for External Memory Interface Designs

Table 6–1 shows the Altera-supported power estimation methods for external memory interfaces.

**Table 6–1. Power Estimation Methods for External Memory Interfaces**

| Method | Vector Source | ALTMEMPHY Support | UniPHY Support | Accuracy | Estimation Time *(1)* |
|---|---|---|---|---|---|
| Early power estimator (EPE) | Not applicable | ✓ | ✓ | Lowest ↓ Highest | Fastest ↓ Slowest |
| Vector-less PowerPlay power analysis (PPPA) | Not applicable | ✓ | ✓ | | |
| Vector-based PPPA | RTL simulation | ✓ | ✓ | | |
| | Zero-delay simulation *(2)* | ✓ | ✓ | | |
| | Timing simulation | *(2)* | *(2)* | | |

**Note to Table 6–1:**

(1) To decrease the estimation time, you can skip power estimation during calibration. Power consumption during calibration is typically equivalent to power consumption during user mode.

(2) Power analysis using timing simulation vectors is not supported.

When using Altera IP, you can use the zero-delay simulation method to analyze the power required for the external memory interface. Zero-delay simulation is as accurate as timing simulation for 95% designs (designs with no glitching). For a design with glitching, power may be under estimated.

For more information about zero-delay simulation, refer to the *Power Estimation and Analysis* section in the *Quartus II Handbook*.

The size of the vector file (**.vcd**) generated by zero-delay simulation of an Altera DDR3 SDRAM High-Performance Controller Example Design is 400 GB. The **.vcd** includes calibration and user mode activities. When vector generation of calibration phase is skipped, the vector size decreases to 1 GB.

To perform vector-based PPPA using zero-delay simulation, follow these steps:

1. Perform design compilation in the Quartus II software to generate your design's Netlist <*project_name*>**.vo**.

   The <*project_name*>**.vo** is generated in the last stage of a compile EDA Netlist Writer.

2. In <*project_name*>**.vo**, search for the include statement for <*project_name*>**.sdo**, comment the statement out, and save the file.

3. Create a simulation script containing device model files and libraries and design specific files:
   - Netlist file for the design, <*project_name*>**.vo**
   - RTL or netlist file for the memory device
   - Testbench RTL file
4. Compile all the files.
5. Invoke simulator with commands to generate **.vcd** files.
6. Generate **.vcd** files for the parts of the design that contribute the most to power dissipation.
7. Run simulation
8. Use the generated **.vcd** files in PPPA tool as the signal activity input file.
9. Run PPPA

For more information about estimating power, refer to the *Power Estimation and Analysis* section in the *Quartus II Handbook*

This chapter provides additional information about the document and Altera.

## Document Revision History

The following table shows the revision history for this section.

| Date | Version | Changes |
|---|---|---|
| December 2010 | 2.1 | ■ Added Arria II GX and Stratix V information.<br>■ Added new information about controller efficiency and RLDRAM II layout guidelines. |
| July 2010 | 2.0 | Updated information about UniPHY-based interfaces and Stratix V devices. |
| April 2010 | 1.0 | Initial release. |

## How to Contact Altera

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

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

**Note to Table:**

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

## Typographic Conventions

The following table shows the typographic conventions this document uses.

| Visual Cue | Meaning |
|---|---|
| **Bold Type with Initial Capital Letters** | Indicate command names, dialog box titles, dialog box options, and other GUI labels. For example, **Save As** dialog box. For GUI elements, capitalization matches the GUI. |
| **bold type** | Indicates directory names, project names, disk drive names, file names, file name extensions, software utility names, and GUI labels. For example, **\qdesigns** directory, **D:** drive, and **chiptrip.gdf** file. |
| *Italic Type with Initial Capital Letters* | Indicate document titles. For example, *Stratix IV Design Guidelines*. |

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

# External Memory Interface Handbook

# Volume 3: Implementing Altera Memory Interface IP

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

EMI_IP-2.0

Document last updated for Altera Complete Design Suite version: 10.1
Document publication date: December 2010

Subscribe

# External Memory Interface Handbook Volume 3

## Section I. DDR and DDR2 SDRAM Controllers with ALTMEMPHY IP User Guide

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

EMI_DDR_UG-2.1

Document last updated for Altera Complete Design Suite version: 10.1
Document publication date: December 2010

Subscribe

# Contents

## Chapter 6. Functional Description—High-Performance Controller

## Chapter 7. Functional Description—High-Performance Controller II



## Chapter 8. Latency

## Chapter 9. Timing Diagrams



## Additional Information

# 1. About This IP

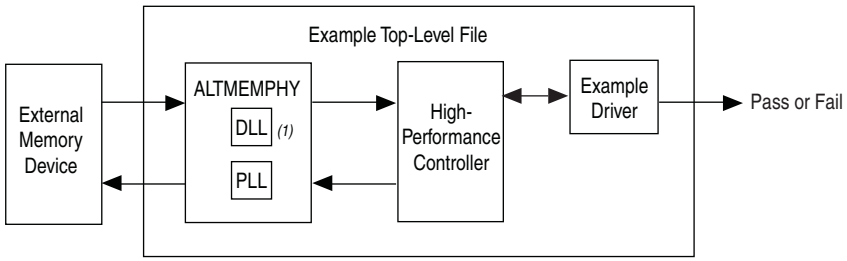
The Altera® DDR and DDR2 SDRAM Controllers with ALTMEMPHY IP provide simplified interfaces to industry-standard DDR SDRAM and DDR2 SDRAM. The ALTMEMPHY megafunction is an interface between a memory controller and the memory devices, and performs read and write operations to the memory. The DDR and DDR2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

The DDR and DDR2 SDRAM Controllers with ALTMEMPHY IP and ALTMEMPHY megafunction offer full-rate or half-rate DDR and DDR2 SDRAM interfaces. The DDR and DDR2 SDRAM Controllers with ALTMEMPHY IP offer two controller architectures: high-performance controller (HPC) and high-performance controller II (HPC II). HPC II provides higher efficiency and more advanced features.

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

Figure 1–1 shows a system-level diagram including the example top-level file that the DDR or DDR2 SDRAM Controller with ALTMEMPHY IP createS for you.

**Figure 1–1. System-Level Diagram**




**Note to Figure 1–1:**

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

The MegaWizard™ Plug-In Manager generates an example top-level file, consisting of an example driver and your DDR or DDR2 SDRAM high-performance controller custom variation. The controller instantiates an instance of the ALTMEMPHY megafunction which in turn instantiates a phase-locked loop (PLL) and DLL. You can also instantiate the DLL outside the ALTMEMPHY megafunction to share the DLL between multiple instances of the ALTMEMPHY megafunction. You cannot share a PLL between multiple instances of the ALTMEMPHY megafunction, but you may share some of the PLL clock outputs between these multiple instances.

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

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

# Release Information

Table 1–1 provides information about this release of the DDR and DDR2 SDRAM high-performance controllers and ALTMEMPHY IP.

**Table 1–1. Release Information**

| Item | Description |
|---|---|
| Version | 10.0 |
| Release Date | July 2010 |
| Ordering Codes | IP-SDRAM/HPDDR (DDR SDRAM HPC)<br>IP-SDRAM/HPDDR2 (DDR2 SDRAM HPC)<br>IP-HPMCII (HPC II) |
| Product IDs | 00BE (DDR SDRAM)<br>00BF (DDR2 SDRAM)<br>00CO (ALTMEMPHY Megafunction) |
| Vendor ID | 6AF7 |

Altera verifies that the current version of the Quartus® II software compiles the previous version of each MegaCore function. The *MegaCore IP Library Release Notes and Errata* report any exceptions to this verification. Altera does not verify compilation with MegaCore function versions older than one release. For information about issues on the DDR and DDR2 SDRAM high-performance controllers and the ALTMEMPHY megafunction in a particular Quartus II version, refer to the *Quartus II Software Release Notes*.

# Device Family Support

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

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

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

Table 1–2 shows the level of support offered by the DDR and DDR2 SDRAM Controllers with ALTMEMPHY IP to each of the Altera device families.

**Table 1–2. Device Family Support**

| Device Family | Support |
|---|---|
| Arria® GX | Final |
| Arria II GX | Preliminary |
| Cyclone® III | Final |
| Cyclone III LS | Final |
| Cyclone IV | Preliminary |
| HardCopy II | HardCopy Companion |
| HardCopy III | HardCopy Companion |
| HardCopy IV E | HardCopy Companion |
| HardCopy IV GX | HardCopy Companion |
| Stratix® II | Final |
| Stratix II GX | Final |
| Stratix III | Final |
| Stratix IV | Final |
| Other device families | No support |

# Features

The ALTMEMPHY megafunction offers the following features:

- Simple setup.
- Support for the Altera PHY Interface (AFI) for DDR and DDR2 SDRAM on all supported devices.
- Automated initial calibration eliminating complicated read data timing calculations.
- Voltage and temperature (VT) tracking that guarantees maximum stable performance for DDR and DDR2 SDRAM interfaces.
- Self-contained datapath that makes connection to an Altera controller or a third-party controller independent of the critical timing paths.
- Full-rate and half-rate DDR and DDR2 SDRAM interfaces.
- Easy-to-use parameter editor.

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

**Table 1–3. DDR and DDR2 SDRAM HPC and HPC II Features (Part 1 of 2)**

| Features | Controller Architecture | |
|---|---|---|
| | HPC | HPC II |
| Half-rate controller | ✓ | ✓ |
| Support for AFI ALTMEMPHY | ✓ | ✓ |
| Support for Avalon® Memory Mapped (MM) local interface | ✓ | ✓ |
| Support for Native local interface | ✓ | — |
| Configurable command look-ahead bank management with in-order reads and writes | — | ✓ |
| Additive latency | — | ✓ *(1)* |
| Optional support for multi-cast write for $t_{RC}$ mitigation | — | ✓ |
| Support for arbitrary Avalon burst length | — | ✓ |
| Memory burst length of 4 | ✓ | ✓ *(2)* |
| Memory burst length of 8 | — | ✓ *(3)* |
| Built-in flexible memory burst adapter | — | ✓ |
| Configurable Local-to-Memory address mappings | — | ✓ |
| Integrated half-rate bridge for low latency option | — | ✓ |
| Optional run-time configuration of size and mode register settings, and memory timing | — | ✓ |
| Partial array self-refresh (PASR) | — | ✓ |
| Support for industry-standard DDR and DDR2 SDRAM devices; and DIMMs | ✓ | ✓ |
| Optional support for self-refresh command | ✓ | ✓ |
| Optional support for user-controlled power-down command | ✓ | — |
| Optional support for automatic power-down command with programmable time-out | — | ✓ |
| Optional support for auto-precharge read and auto-precharge write commands | ✓ | — |
| Optional support for user-controller refresh | ✓ | ✓ |
| Reduced bank tracking for area optimization | — | ✓ |
| Controller variable latency | — | ✓ |
| Optional multiple controller clock sharing in SOPC Builder Flow | ✓ | ✓ |
| Integrated error correction coding (ECC) function 72-bit | ✓ | ✓ |
| Integrated ECC function 40-bit | — | ✓ |
| Support for partial-word write with optional automatic error correction | — | ✓ |
| SOPC Builder ready | ✓ | ✓ |
| Support for OpenCore Plus evaluation | ✓ | — |

**Table 1–3. DDR and DDR2 SDRAM HPC and HPC II Features (Part 2 of 2)**

| Features | Controller Architecture | |
|---|---|---|
| | HPC | HPC II |
| IP functional simulation models for use in Altera-supported VHDL and Verilog HDL simulator | ✓ | ✓ |

**Notes to Table 1–3:**

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

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

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

# Unsupported Features

The DDR and DDR2 SDRAM Controllers with ALTMEMPHY IP do not support the following features:

- Timing simulation.
- Burst length of 2.
- Partial burst and unaligned burst in ECC and non-ECC mode when DM pins are disabled.

# MegaCore Verification

Altera performs extensive random, directed tests with functional test coverage using industry-standard Denali models to ensure the functionality of the DDR and DDR2 SDRAM Controllers with ALTMEMPHY IP.

# Resource Utilization

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

## ALTMEMPHY Megafunction

Table 1–4 through Table 1–7 show the typical size of the ALTMEMPHY megafunction with the AFI in the Quartus II software version 10.0 for the following devices:

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

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

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

| PHY Rate | Memory Width (Bits) | Combinational ALUTS | Logic Registers | M9K Blocks | Memory ALUTs |
|---|---|---|---|---|---|
| Half | 8 | 1,428 | 1,179 | 2 | 18 |
| | 16 | 1,480 | 1,254 | 4 | 2 |
| | 64 | 1,787 | 1,960 | 12 | 22 |
| | 72 | 1,867 | 2,027 | 13 | 2 |
| Full | 8 | 1,232 | 975 | 0 | 35 |
| | 16 | 1,240 | 915 | 3 | 1 |
| | 64 | 1,287 | 1,138 | 7 | 41 |
| | 72 | 1,303 | 1,072 | 9 | 1 |

**Note to Table 1–4:**

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

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

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

**Note to Table 1–5:**

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

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

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

**Notes to Table 1–6:**

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

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

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

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

**Note to Table 1–7:**

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

## High-Performance Controller

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

**Table 1–8. Resource Utilization in Arria GX Devices**

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

**Table 1–9. Resource Utilization in Arria II GX Devices**

| Controller Rate | Local Data Width (Bits) | Memory Width (Bits) | Combinational ALUTs | Dedicated Logic Registers | Memory |
|---|---|---|---|---|---|
| | | | | | (M9K) |
| Half | 32 | 8 | 1,837 | 1,553 | 3 |
| | 64 | 16 | 1,894 | 1,628 | 6 |
| | 256 | 64 | 2,201 | 2,334 | 20 |
| | 288 | 72 | 2,279 | 2,401 | 22 |
| Full | 16 | 8 | 1,671 | 1400 | 1 |
| | 32 | 16 | 1,684 | 1,340 | 4 |
| | 128 | 64 | 1725 | 1,562 | 11 |
| | 144 | 72 | 1,738 | 2,497 | 14 |

**Table 1–10. Resource Utilization in Cyclone III Devices**

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

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

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

**Table 1–12. Resource Utilization in Stratix III Devices**

| Controller Rate | Local Data Width (Bits) | Memory Width (Bits) | Combinational ALUTs | Dedicated Logic Registers | Memory (M9K) |
|---|---|---|---|---|---|
| Half | 32 | 8 | 1,752 | 1,432 | 2 |
| | 64 | 16 | 1,824 | 1,581 | 3 |
| | 256 | 64 | 2,210 | 2,465 | 9 |
| | 288 | 72 | 2,321 | 2,613 | 10 |
| Full | 16 | 8 | 1,622 | 1,351 | 2 |
| | 32 | 16 | 1,630 | 1,431 | 2 |
| | 128 | 64 | 1736 | 1,897 | 5 |
| | 144 | 72 | 1,749 | 1,975 | 6 |

**Table 1–13. Resource Utilization in Stratix IV Devices**

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

## High-Performance Controller II

Table 1–14 through Table 1–17 show the typical sizes for the DDR or DDR2 SDRAM HPC II (including ALTMEMPHY) for Arria II GX, Cyclone III, Stratix III, and Stratix IV devices.

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

| Controller Rate | Local Data Width (Bits) | Memory Width (Bits) | Combinational ALUTs | Dedicated Logic Registers | Memory (M9K) |
|---|---|---|---|---|---|
| Half | 32 | 8 | 2,477 | 1,879 | 3 |
| | 64 | 16 | 2,542 | 2,050 | 5 |
| | 256 | 64 | 3,080 | 2,970 | 17 |
| | 288 | 72 | 3,203 | 3,124 | 18 |
| Full | 16 | 8 | 2,238 | 1,672 | 1 |
| | 32 | 16 | 2,311 | 1,709 | 3 |
| | 128 | 64 | 2,561 | 2,083 | 8 |
| | 144 | 72 | 2,608 | 2,088 | 10 |

**Table 1–15. Resource Utilization in Cyclone III Devices**

| Controller Rate | Local Data Width (Bits) | Memory Width (Bits) | Combinational ALUTs | Dedicated Logic Registers | Memory (M9K) |
|---|---|---|---|---|---|
| Half | 32 | 8 | 3,261 | 1,884 | 3 |
| | 64 | 16 | 3.468 | 2,080 | 5 |
| | 256 | 64 | 4,700 | 3,132 | 17 |
| | 288 | 72 | 4,917 | 3,288 | 18 |
| Full | 16 | 8 | 2,856 | 1,540 | 3 |
| | 32 | 16 | 2,954 | 1,628 | 3 |
| | 128 | 64 | 3,520 | 2,086 | 9 |
| | 144 | 72 | 3,616 | 2,163 | 10 |

**Table 1–16. Resource Utilization in Stratix III Devices**

| Controller Rate | Local Data Width (Bits) | Memory Width (Bits) | Combinational ALUTs | Dedicated Logic Registers | Memory (M9K) |
|---|---|---|---|---|---|
| Half | 32 | 8 | 2,364 | 1,797 | 2 |
| | 64 | 16 | 2,432 | 1,923 | 3 |
| | 256 | 64 | 2,837 | 2,807 | 9 |
| | 288 | 72 | 2,940 | 2,955 | 10 |
| Full | 16 | 8 | 2,187 | 1,618 | 2 |
| | 32 | 16 | 2,168 | 1,654 | 2 |
| | 128 | 64 | 2,300 | 2,120 | 5 |
| | 144 | 72 | 2,261 | 2,194 | 6 |

**Table 1–17. Resource Utilization in Stratix IV Devices**

| Controller Rate | Local Data Width (Bits) | Memory Width (Bits) | Combinational ALUTs | Dedicated Logic Registers | M9K |
|---|---|---|---|---|---|
| Half | 32 | 8 | 2,358 | 1,796 | 2 |
| | 64 | 16 | 2,425 | 1,918 | 3 |
| | 256 | 64 | 2,815 | 2,803 | 9 |
| | 288 | 72 | 2,910 | 2,922 | 10 |
| Full | 16 | 8 | 2,216 | 1,741 | 1 |
| | 32 | 16 | 2,151 | 1,656 | 2 |
| | 128 | 64 | 2,237 | 2,159 | 5 |
| | 144 | 72 | 2,274 | 2,307 | 5 |

# System Requirements

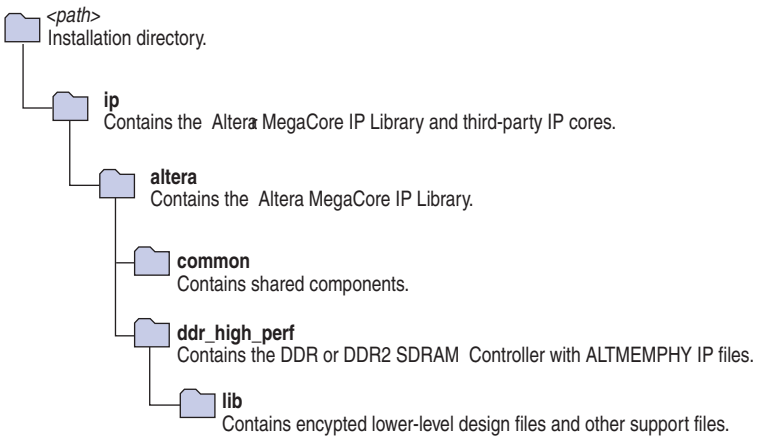
The DDR and DDR2 SDRAM Controllers with ALTMEMPHY IP are part of the MegaCore IP Library, which is distributed with the Quartus II software and downloadable from the Altera website, www.altera.com.

For system requirements and installation instructions, refer to *Altera Software Installation & Licensing*.

# Installation and Licensing

Figure 1–2 shows the directory structure after you install the DDR and DDR2 SDRAM Controllers with ALTMEMPHY IP, where <*path*> is the installation directory. The default installation directory on Windows is **c:\altera**\<*version*>; on Linux it is **/opt/altera**<*version*>.

**Figure 1–2. Directory Structure**




You need a license for the DDR and DDR2 SDRAM Controllers with ALTMEMPHY IP only when you are completely satisfied with its functionality and performance, and want to take your design to production.

To use the DDR or DDR2 SDRAM HPC, you can request a license file from the Altera web site at www.altera.com/licensing and install it on your computer. When you request a license file, Altera emails you a **license.dat** file. If you do not have Internet access, contact your local representative.

To use the DDR or DDR2 HPC II, contact your local sales representative to order a license.

## Free Evaluation

Altera's OpenCore Plus evaluation feature is only applicable to the DDR or DDR2 SDRAM HPC. With the OpenCore Plus evaluation feature, you can perform the following actions:

- Simulate the behavior of a megafunction (Altera MegaCore function or AMPP$^{SM}$ megafunction) within your system.
- Verify the functionality of your design, as well as evaluate its size and speed quickly and easily.
- Generate time-limited device programming files for designs that include MegaCore functions.
- Program a device and verify your design in hardware.

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

## OpenCore Plus Time-Out Behavior

OpenCore Plus hardware evaluation can support the following two modes of operation:

- Untethered—the design runs for a limited time
- Tethered—requires a connection between your board and the host computer. If tethered mode is supported by all megafunctions in a design, the device can operate for a longer time or indefinitely

All megafunctions in a device time-out simultaneously when the most restrictive evaluation time is reached. If there is more than one megafunction in a design, a specific megafunction's time-out behavior may be masked by the time-out behavior of the other megafunctions.

For MegaCore functions, the untethered time-out is 1 hour; the tethered time-out value is indefinite.

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

## Design Flow

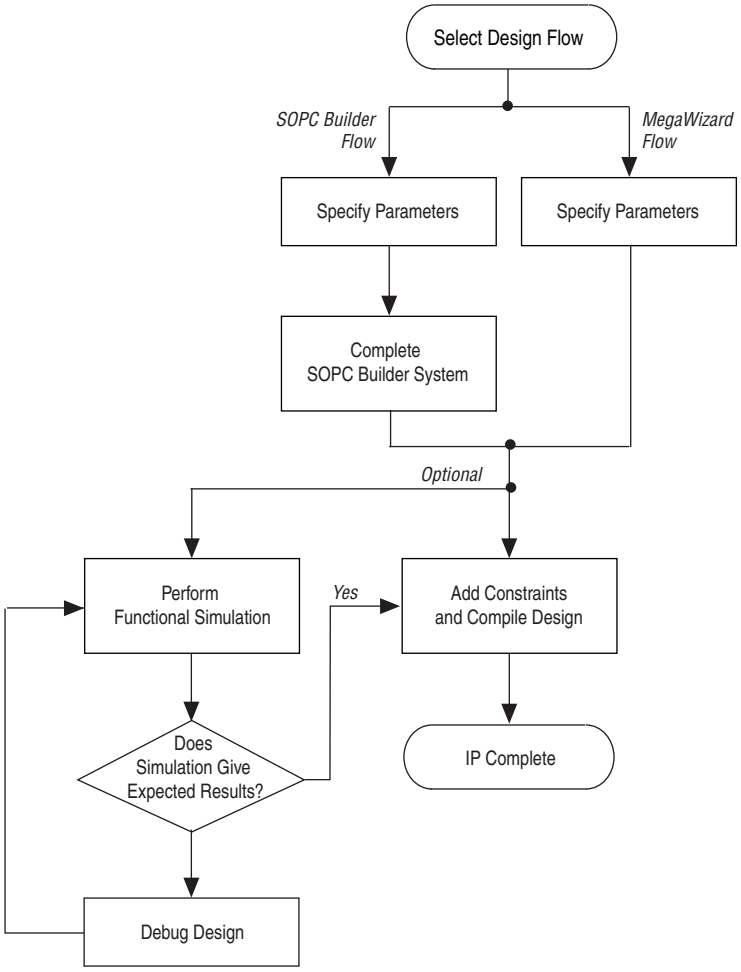
You can implement the DDR and DDR2 SDRAM Controllers with ALTMEMPHY IP using either one of the following flows:

- SOPC Builder flow
- MegaWizard Plug-In Manager flow

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

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

**Figure 2–1. Design Flow**



The SOPC Builder flow offers the following advantages:

- Generates simulation environment
- Creates custom components and integrates them via the component wizard
- Interconnects all components with the Avalon-MM interface

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

- Allows you to design directly from the DDR or DDR2 SDRAM interface to peripheral device or devices
- Achieves higher-frequency operation

# SOPC Builder Flow

The SOPC Builder flow allows you to add the DDR and DDR2 SDRAM high-performance controllers directly to a new or existing SOPC Builder system.

You can also easily add other available components to quickly create an SOPC Builder system with a DDR or DDR2 SDRAM controller, such as the Nios II processor and scatter-gather direct memory access (SDMA) controllers. SOPC Builder automatically creates the system interconnect logic and system simulation environment.

For more information about SOPC Builder, refer to volume 4 of the *Quartus II Handbook*. For more information about how to use controllers with SOPC Builder, refer to the *ALTMEMPHY Design Tutorials* section in volume 6 of the *External Memory Interface Handbook*. For more information on the Quartus II software, refer to the Quartus II Help.

## Specifying Parameters

To specify the parameters for the DDR or DDR2 SDRAM Controller with ALTMEMPHY IP, using the SOPC Builder flow, perform the following steps:

1. In the Quartus II software, create a new Quartus II project with the **New Project Wizard**.
2. On the Tools menu, click **SOPC Builder**.
3. For a new system, specify the system name and language.
4. Add **DDR or DDR2 SDRAM Controller with ALTMEMPHY** to your system from the **System Contents** tab.

   The **DDR or DDR2 SDRAM Controller with ALTMEMPHY** is in the **SDRAM** folder under the **Memories and Memory Controllers** folder.

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

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

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

6. Click **Finish** to complete parameterizing the DDR or DDR2 SDRAM Controller with ALTMEMPHY IP and add it to the system.

## Completing the SOPC Builder System

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

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

   CAUTION

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

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

   To calculate the Avalon-MM address equivalent of the memory address range `0x0` to `0x1f`, multiply the memory address by the width of the memory interface data bus in bytes. Refer to Table 2–1 for more Avalon-MM addresses.

**Table 2–1. Avalon-MM Addresses for AFI Mode**

| External Memory Interface Width | Reset Vector Offset | Exception Vector Offset |
|---|---|---|
| 8 | 0×40 | 0×60 |
| 16 | 0×80 | 0×A0 |
| 32 | 0×100 | 0×120 |
| 64 | 0×200 | 0×220 |

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

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

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

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

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

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

# MegaWizard Plug-In Manager Flow

The MegaWizard Plug-In Manager flow allows you to customize the DDR or DDR2 SDRAM Controller with ALTMEMPHY or the stand-alone PHY with the ALTMEMPHY megafunction, and manually integrate the function into your design.

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

## Specifying Parameters

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

1. In the Quartus II software, create a new Quartus II project with the **New Project Wizard**.
2. On the Tools menu, click **MegaWizard Plug-In Manager** to start the MegaWizard Plug-In Manager.
   - The DDR or DDR2 SDRAM Controller with ALTMEMPHY is in the **Interfaces** folder under the **External Memory** folder.
   - The ALTMEMPHY megafunction is in the **I/O** folder.

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

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

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

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

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

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

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

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

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

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

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

8. After you review the generation report, click **Exit** to close the MegaWizard Plug-In Manager.
9. For the high-performance controller (HPC or HPC II), set the <*variation name*>**_example_top.v** or **.vhd** file to be the project top-level design file.
   a. On the File menu, click **Open**.
   b. Browse to <*variation name*>**_example_top** and click **Open**.
   c. On the Project menu, click **Set as Top-Level Entity**.

# HardCopy Guidelines

For information about targeting a HardCopy device, refer to "HardCopy Migration Design Guidelines" in the *Getting Started* chapter in the *DDR3 SDRAM Controller with ALTMEMPHY IP User Guide*.

# Generated Files

Table 2–2 shows the ALTMEMPHY generated files.

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

| File Name | Description |
|---|---|
| alt_mem_phy_defines.v | Contains constants used in the interface. This file is always in Verilog HDL regardless of the language you chose in the MegaWizard Plug-In Manager. |
| <*variation_name*>**.ppf** | Pin planner file for your ALTMEMPHY variation. |
| <*variation_name*>**.qip** | Quartus II IP file for your ALTMEMPHY variation, containing the files associated with this megafunction. |
| <*variation_name*>**.v/.vhd** | Top-level file of your ALTMEMPHY variation, generated based on the language you chose in the MegaWizard Plug-In Manager. |
| <*variation_name*>**.vho** | Contains functional simulation model for VHDL only. |
| <*variation_name*>**_alt_mem_phy_seq_wrapper.vo/.vho** | A wrapper file, for simulation only, that calls the sequencer file, created based on the language you chose in the MegaWizard Plug-In Manager. |
| <*variation_name*>**.html** | Lists the top-level files created and ports used in the megafunction. |
| <*variation_name*>**_alt_mem_phy_seq_wrapper.v/.vhd** | A wrapper file, for compilation only, that calls the sequencer file, created based on the language you chose in the MegaWizard Plug-In Manager. |
| <*variation_name*>_alt_mem_phy_seq.vhd | Contains the sequencer used during calibration. This file is always in VHDL language regardless of the language you chose in the MegaWizard Plug-In Manager. |
| <*variation_name*>**_alt_mem_phy.v** | Contains all modules of the ALTMEMPHY variation except for the sequencer. This file is always in Verilog HDL language regardless of the language you chose in the MegaWizard Plug-In Manager. The DDR3 SDRAM sequencer is included in the <*variation_name*>**_alt_mem_phy_seq.**vhd file. |
| <*variation name*>**_alt_mem_phy_pll_**<*device*>**.ppf** | This XML file describes the MegaCore pin attributes to the Quartus II Pin Planner. |
| <*variation_name*>**_alt_mem_phy_pll.v/.vhd** | The PLL megafunction file for your ALTMEMPHY variation, generated based on the language you chose in the MegaWizard Plug-In Manager. |
| <*variation_name*>**_alt_mem_phy_delay.vhd** | Includes a delay module for simulation. This file is only generated if you choose VHDL as the language of your MegaWizard Plug-In Manager output files. |

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

| File Name | Description |
|---|---|
| <*variation_name*>**_alt_mem_phy_dq_dqs.vhd** or **.v** | Generated file that contains DQ/DQS I/O atoms interconnects and instance. Arria II GX devices only. |
| <*variation_name*>**_alt_mem_phy_dq_dqs_clearbox.txt** | Specification file that generates the <*variation_name*>**_alt_mem_phy_dq_dqs** file using the clearbox flow. Arria II GX devices only. |
| <*variation_name*>**_alt_mem_phy_pll.qip** | Quartus II IP file for the PLL that your ALTMEMPHY variation uses that contains the files associated with this megafunction. |
| <*variation_name*>**_alt_mem_phy_pll_bb.v/.cmp** | Black box file for the PLL used in your ALTMEMPHY variation. Typically unused. |
| <*variation_name*>**_alt_mem_phy_reconfig . qip** | Quartus II IP file for the PLL reconfiguration block. Only generated when targeting Arria GX, HardCopy II, Stratix II, and Stratix II GX devices. |
| <*variation_name*>**_alt_mem_phy_reconfig.v/.vhd** | PLL reconfiguration block module. Only generated when targeting Arria GX, HardCopy II, Stratix II, and Stratix II GX devices. |
| <*variation_name*>**_alt_mem_phy_reconfig_bb.v/cmp** | Black box file for the PLL reconfiguration block. Only generated when targeting Arria GX, HardCopy II, Stratix II, and Stratix II GX devices. |
| <*variation_name*>**_bb.v/.cmp** | Black box file for your ALTMEMPHY variation, depending whether you are using Verilog HDL or VHDL language. |
| <*variation_name*>**_ddr_pins.tcl** | Contains procedures used in the <*variation_name*>**_ddr_timing.sdc** and <*variation_name*>**_report_timing.tcl** files. |
| <*variation_name*>**_pin_assignments.tcl** | Contains I/O standard, drive strength, output enable grouping, DQ/DQS grouping, and termination assignments for your ALTMEMPHY variation. If your top-level design pin names do not match the default pin names or a prefixed version, edit the assignments in this file. |
| <*variation_name*>**_ddr_timing.sdc** | Contains timing constraints for your ALTMEMPHY variation. |
| <*variation_name*>**_report_timing.tcl** | Script that reports timing for your ALTMEMPHY variation during compilation. |

Table 2–3 shows the modules that are instantiated in the **<***variation_name***>_alt_mem_phy.v/.vhd** file. A particular ALTMEMPHY variation may or may not use any of the modules, depending on the memory standard that you specify.

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

| Module Name | Usage | Description |
|---|---|---|
| <*variation_name*>`_alt_mem_phy_addr_cmd` | All ALTMEMPHY variations | Generates the address and command structures. |
| <*variation_name*>`_alt_mem_phy_clk_reset` | All ALTMEMPHY variations | Instantiates PLL, DLL, and reset logic. |

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

| Module Name | Usage | Description |
|---|---|---|
| <*variation_name*>_alt_mem_phy_dp_io | All ALTMEMPHY variations | Generates the DQ, DQS, DM, and QVLD I/O pins. |
| <*variation_name*>_alt_mem_phy_mimic | DDR2/DDR SDRAM ALTMEMPHY variation | Creates the VT tracking mechanism for DDR and DDR2 SDRAM PHYs. |
| <*variation_name*>_alt_mem_phy_oct_delay | DDR2/DDR SDRAM ALTMEMPHY variation when dynamic OCT is enabled. | Generates the proper delay and duration for the OCT signals. |
| <*variation_name*>_alt_mem_phy_postamble | DDR2/DDR SDRAM ALTMEMPHY variations | Generates the postamble enable and disable scheme for DDR and DDR2 SDRAM PHYs. |
| <*variation_name*>_alt_mem_phy_read_dp | All ALTMEMPHY variations (unused for Stratix III or Stratix IV devices) | Takes read data from the I/O through a read path FIFO buffer, to transition from the resyncronization clock to the PHY clock. |
| <*variation_name*>_alt_mem_phy_read_dp_group | DDR2/DDR SDRAM ALTMEMPHY variations (Stratix III and Stratix IV devices only) | A per DQS group version of <*variation_name*>_alt_mem_phy_read_dp. |
| <*variation_name*>_alt_mem_phy_rdata_valid | DDR2/DDR SDRAM ALTMEMPHY variations | Generates read data valid signal to sequencer and controller. |
| <*variation_name*>_alt_mem_phy_seq_wrapper | All ALTMEMPHY variations | Generates sequencer for DDR and DDR2 SDRAM. |
| <*variation_name*>_alt_mem_phy_write_dp | All ALTMEMPHY variations | Generates the demultiplexing of data from half-rate to full-rate DDR data. |
| <*variation_name*>_alt_mem_phy_write_dp_fr | DDR2/DDR SDRAM ALTMEMPHY variations | A full-rate version of <*variation_name*>_alt_mem_phy_write_dp. |

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

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

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

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

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

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

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

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

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

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

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

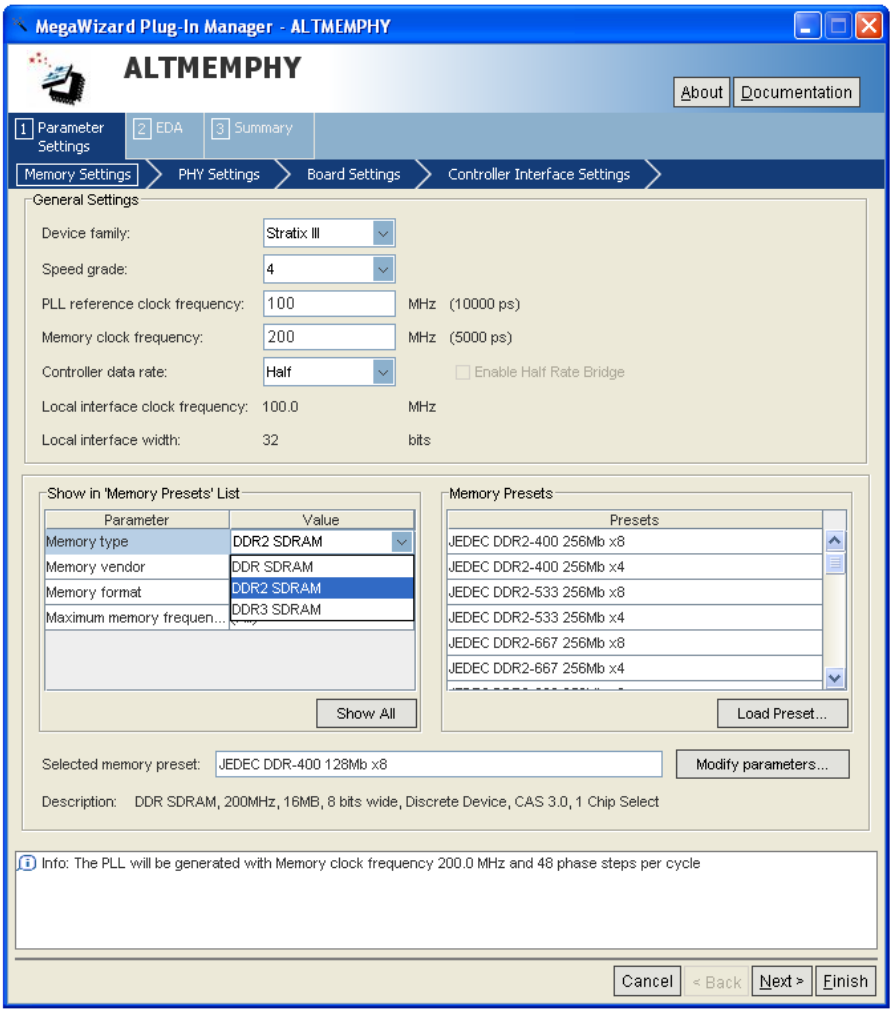
| Filename | Description |
|---|---|
| alt_ddrx_odt_gen.v | Instantiates the **alt_ddrx_ddr2_odt_gen.v** file selectively. It also controls the ODT addressing scheme. |
| alt_ddrx_state_machine.v | The main state machine of the controller. |
| alt_ddrx_timers_fsm.v | The state machine that tracks the per-bank timing parameters. |
| alt_ddrx_timers.v | Instantiates **alt_ddrx_timers_fsm.v** and contains the rank specific timing tracking logic. |
| alt_ddrx_wdata_fifo.v | The write data FIFO logic. This logic buffers the write data and byte enables from the Avalon interface. |
| alt_avalon_half_rate_bridge_constraints.sdc | Contains timing constraints if your design has the **Enable Half Rate Bridge** option turned on. |
| alt_avalon_half_rate_bridge.v | The integrated half-rate bridge logic block. |

# ALTMEMPHY Parameter Settings

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

- Memory Settings
- PHY Settings
- Board Settings

**Figure 3–1. ALTMEMPHY Parameter Settings Page**




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

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

## Memory Settings

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

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

**Table 3–1. General Settings**

| Parameter Name | Description |
|---|---|
| Device family | Targets device family (for example, Stratix III). Table 1–2 on page 1–3 shows supported device families. The device family selected here must match the device family selected on the MegaWizard page 2a. |
| Speed grade | Selects a particular speed grade of the device (for example, 2, 3, or 4 for the Stratix III device family). |
| PLL reference clock frequency | Determines the clock frequency of the external input clock to the PLL. Ensure that you use three decimal points if the frequency is not a round number (for example, 166.667 MHz or 100 MHz) to avoid a functional simulation or a PLL locking problem. |
| Memory clock frequency | Determines the memory interface clock frequency. If you are operating a memory device below its maximum achievable frequency, ensure that you enter the actual frequency of operation rather than the maximum frequency achievable by the memory device. Also, ensure that you use three decimal points if the frequency is not a round number (for example, 333.333 MHz or 400 MHz) to avoid a functional simulation or a PLL locking issue. |
| Controller data rate | Selects the data rate for the memory controller. Sets the frequency of the controller to equal to either the memory interface frequency (full-rate) or half of the memory interface frequency (half-rate). |
| Enable half rate bridge | This option is only available for HPC II.<br>Turn on to keep the controller in the memory full clock domain while allowing the local side to run at half the memory clock speed, so that latency can be reduced. |
| Local interface clock frequency | Value that depends on the memory clock frequency and controller data rate, and whether or not you turn on the **Enable Half Rate Bridge** option. |
| Local interface width | Value that depends on the memory clock frequency and controller data rate, and whether or not you turn on the **Enable Half Rate Bridge** option. |

Table 3–2 describes the options available to filter the **Memory Presets** that are displayed. This set of options is where you indicate whether you are creating a datapath for DDR or DDR2 SDRAM.

**Table 3–2. Memory Presets List**

| Parameter Name | Description |
|---|---|
| Memory type | You can filter the type of memory to display, for example, DDR2 SDRAM. The ALTMEMPHY megafunction supports DDR SDRAM and DDR2 SDRAM. |
| Memory vendor | You can filter the memory types by vendor. JEDEC is also one of the options, allowing you to choose the JEDEC specifications. If your chosen vendor is not listed, you can choose JEDEC for the DDR and DDR2 SDRAM interfaces. Then, pick a device that has similar specifications to your chosen device and check the values of each parameter. Make sure you change the each parameter value to match your device specifications. |
| Memory format | You can filter the type of memory by format, for example, discrete devices or DIMM packages. |
| Maximum frequency | You can filter the type of memory by the maximum operating frequency. |

## Using the Preset Editor to Create a Custom Memory Preset

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

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

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

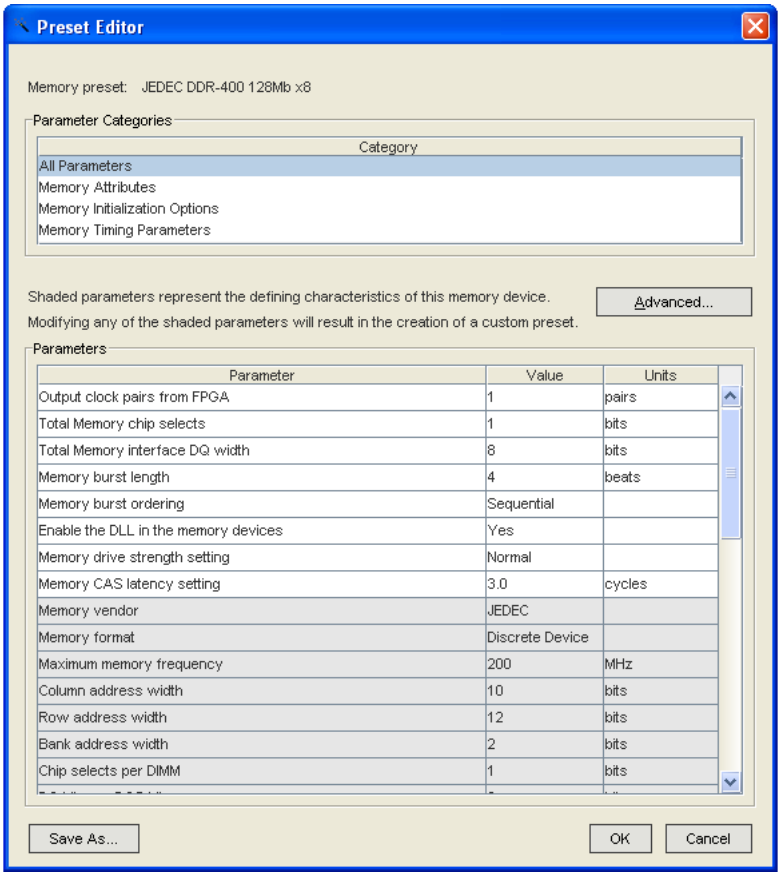
You can change the parameters with a white background to reflect your system. You can also change the parameters with a gray background so the device parameters match the device you are using. These parameters in gray background are characteristics of the chosen memory device and changing them creates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

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

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

Figure 3–2 shows the **Preset Editor** dialog box for a DDR2 SDRAM.

**Figure 3–2. DDR2 SDRAM Preset Editor**




The **Advanced** option is only available for Arria II GX and Stratix IV devices. This option shows the percentage of memory specification that is calibrated by the FPGA. The percentage values are estimated by Altera based on the process variation.

Table 3–3 through Table 3–5 describe the DDR2 SDRAM parameters available for memory attributes, initialization options, and timing parameters. DDR SDRAM has the same parameters, but their value ranges are different than DDR2 SDRAM.

**Table 3–3. DDR2 SDRAM Attributes Settings (Part 1 of 2)**

| Parameter Name | Range *(1)* | Units | Description |
|---|---|---|---|
| Output clock pairs from FPGA | 1–6 | pairs | Defines the number of differential clock pairs driven from the FPGA to the memory. More clock pairs reduce the loading of each output when interfacing with multiple devices. Memory clock pins use the signal splitter feature in Arria II GX, Stratix III, and Stratix IV devices for differential signaling. |
| Total Memory chip selects | 1, 2, 4, or 8 | bits | Sets the number of chip selects in your memory interface. The number of chip selects defines the depth of your memory. You are limited to the range shown as the local side binary encodes the chip select address. You can set this value to the next higher number if the range does not meet your specifications. However, the highest address space of the ALTMEMPHY megafunction is not mapped to any of the actual memory address. The ALTMEMPHY megafunction works with multiple chip selects and calibrates against all chip select, `mem_cs_n` signals. |
| Memory interface DQ width | 4–288 | bits | Defines the total number of DQ pins on the memory interface. If you are interfacing with multiple devices, multiply the number of devices with the number of DQ pins per device. Even though the GUI allows you to choose 288-bit DQ width, the interface data width is limited by the number of pins on the device. For best performance, have the whole interface on one side of the device. |
| Memory vendor | JEDEC, Micron, Qimonda, Samsung, Hynix, Elpida, Nanya, other | — | Lists the name of the memory vendor for all supported memory standards. |
| Memory format | Discrete Device, Unbuffered DIMM, Registered DIMM | — | Specifies whether you are interfacing with devices or modules. SODIMM is supported under unbuffered or registered DIMMs. |
| Maximum memory frequency | See the memory device datasheet | MHz | Sets the maximum frequency supported by the memory. |
| Column address width | 9–11 | bits | Defines the number of column address bits for your interface. |
| Row address width | 13–16 | bits | Defines the number of row address bits for your interface. |
| Bank address width | 2 or 3 | bits | Defines the number of bank address bits for your interface. |
| Chip selects per DIMM | 1 or 2 | bits | Defines the number of chip selects on each DIMM in your interface. |

**Table 3–3. DDR2 SDRAM Attributes Settings (Part 2 of 2)**

| Parameter Name | Range *(1)* | Units | Description |
|---|---|---|---|
| DQ bits per DQS bit | 4 or 8 | bits | Defines the number of data (DQ) bits for each data strobe (DQS) pin. |
| Precharge address bit | 8 or 10 | bits | Selects the bit of the address bus to use as the precharge address bit. |
| Drive DM pins from FPGA | Yes or No | — | Specifies whether you are using DM pins for write operation. Altera devices do not support DM pins in ×4 mode. |
| Maximum memory frequency for CAS latency 3.0 | 80–533 | MHz | Specifies the frequency limits from the memory data sheet per given CAS latency. The ALTMEMPHY parameter editor generates a warning if the operating frequency with your chosen CAS latency exceeds this number. |
| Maximum memory frequency for CAS latency 4.0 | | | |
| Maximum memory frequency for CAS latency 5.0 | | | |
| Maximum memory frequency for CAS latency 6.0 | | | |

**Note to Table 3–3:**

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

**Table 3–4. DDR2 SDRAM Initialization Options**

| Parameter Name | Range | Units | Description |
|---|---|---|---|
| Memory burst length | 4 or 8 | beats | Sets the number of words read or written per transaction.<br>Memory burst length of four equates to local burst length of one in half-rate designs and to local burst length of two in full-rate designs. |
| Memory burst ordering | Sequential or Interleaved | — | Controls the order in which data is transferred between memory and the FPGA during a read transaction. For more information, refer to the memory device datasheet. |
| Enable the DLL in the memory devices | Yes or No | — | Enables the DLL in the memory device when set to **Yes**. You must always enable the DLL in the memory device as Altera does not guarantee any ALTMEMPHY operation when the DLL is turned off. All timings from the memory devices are invalid when the DLL is turned off. |
| Memory drive strength setting | Normal or Reduced | — | Controls the drive strength of the memory device's output buffers. Reduced drive strength is not supported on all memory devices. The default option is normal. |
| Memory ODT setting | Disabled, 50, 75, 150 | W | Sets the memory ODT value. Not available in DDR SDRAM interfaces. |
| Memory CAS latency setting | 3, 4, 5, 6 | cycles | Sets the delay in clock cycles from the read command to the first output data from the memory. |

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

| Parameter Name | Range | Units | Description |
|---|---|---|---|
| $t_{INIT}$ | 0.001–1000 | µs | Minimum memory initialization time. After reset, the controller does not issue any commands to the memory during this period. |
| $t_{MRD}$ | 2–39 | ns | Minimum load mode register command period. The controller waits for this period of time after issuing a load mode register command before issuing any other commands.<br>$t_{MRD}$ is specified in ns in the DDR2 SDRAM high-performance controller and in terms of $t_{CK}$ cycles in Micron's device datasheet. Convert $t_{MRD}$ to ns by multiplying the number of cycles specified in the datasheet times $t_{CK}$, where $t_{CK}$ is the memory operation frequency and not the memory device's $t_{CK}$. |
| $t_{RAS}$ | 8–200 | ns | Minimum active to precharge time. The controller waits for this period of time after issuing an active command before issuing a precharge command to the same bank. |
| $t_{RCD}$ | 4–65 | ns | Minimum active to read-write time. The controller does not issue read or write commands to a bank during this period of time after issuing an active command. |
| $t_{RP}$ | 4–65 | ns | Minimum precharge command period. The controller does not access the bank for this period of time after issuing a precharge command. |
| $t_{REFI}$ | 1–65534 | µs | Maximum interval between refresh commands. The controller performs regular refresh at this interval unless user-controlled refresh is turned on. |
| $t_{RFC}$ | 14–1651 | ns | Minimum autorefresh command period. The length of time the controller waits before doing anything else after issuing an auto-refresh command. |
| $t_{WR}$ | 4–65 | ns | Minimum write recovery time. The controller waits for this period of time after the end of a write transaction before issuing a precharge command. |
| $t_{WTR}$ | 1–3 | $t_{CK}$ | Minimum write-to-read command delay. The controller waits for this period of time after the end of a write command before issuing a subsequent read command to the same bank. This timing parameter is specified in clock cycles and the value is rounded off to the next integer. |
| $t_{AC}$ | 300–750 | ps | DQ output access time from CK/CK# signals. |
| $t_{DQSCK}$ | 100–750 | ps | DQS output access time from CK/CK# signals. |
| $t_{DQSQ}$ | 100–500 | ps | The maximum DQS to DQ skew; DQS to last DQ valid, per group, per access. |
| $t_{DQSS}$ | 0–0.3 | $t_{CK}$ | Positive DQS latching edge to associated clock edge. |

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

| Parameter Name | Range | Units | Description |
|---|---|---|---|
| $t_{DS}$ | 10–600 | ps | DQ and DM input setup time relative to DQS, which has a derated value depending on the slew rate of the DQS (for both DDR and DDR2 SDRAM interfaces) and whether DQS is single-ended or differential (for DDR2 SDRAM interfaces). Ensure that you are using the correct number and that the value entered is referenced to $V_{REF}$(dc), not $V_{IH}$(ac) min or $V_{IL}$(ac) max. Refer to "Derating Memory Setup and Hold Timing" on page 3–9 for more information about how to derate this specification. |
| $t_{DH}$ | 10–600 | ps | DQ and DM input hold time relative to DQS, which has a derated value depending on the slew rate of the DQS (for both DDR and DDR2 SDRAM interfaces) and whether DQS is single-ended or differential (for DDR2 SDRAM interfaces). Ensure that you are using the correct number and that the value entered is referenced to $V_{REF}$(dc), not $V_{IH}$(dc) min or $V_{IL}$(dc) max. Refer to "Derating Memory Setup and Hold Timing" on page 3–9 for more information about how to derate this specification. |
| $t_{DSH}$ | 0.1–0.5 | $t_{CK}$ | DQS falling edge hold time from CK. |
| $t_{DSS}$ | 0.1–0.5 | $t_{CK}$ | DQS falling edge to CK setup. |
| $t_{IH}$ | 100–1000 | ps | Address and control input hold time, which has a derated value depending on the slew rate of the CK and CK# clocks and the address and command signals. Ensure that you are using the correct number and that the value entered is referenced to $V_{REF}$(dc), not $V_{IH}$(dc) min or $V_{IL}$(dc) max. Refer to "Derating Memory Setup and Hold Timing" on page 3–9 for more information about how to derate this specification. |
| $t_{IS}$ | 100–1000 | ps | Address and control input setup time, which has a derated value depending on the slew rate of the CK and CK# clocks and the address and command signals. Ensure that you are using the correct number and that the value entered is referenced to $V_{REF}$(dc), not $V_{IH}$(ac) min or $V_{IL}$(ac) max. Refer to "Derating Memory Setup and Hold Timing" on page 3–9 for more information about how to derate this specification. |
| $t_{QHS}$ | 100–700 | ps | The maximum data hold skew factor. |
| $t_{RRD}$ | 2.06–64 | ns | The activate command to activate time, per device, RAS to RAS delay timing parameter. |
| $t_{FAW}$ | 7.69–256 | ns | The four-activate window time, per device. |
| $t_{RTP}$ | 2.06–64 | ns | Read to precharge time. |

**Note to Table 3–5:**

(1) See the memory device data sheet for the parameter range. Some of the parameters may be listed in a clock cycle ($t_{CK}$) unit. If the MegaWizard Plug-In Manager requires you to enter the value in a time unit (ps or ns), convert the number by multiplying it with the clock period of your interface (and not the maximum clock period listed in the memory data sheet).

## Derating Memory Setup and Hold Timing

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

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

For Arria II GX and Stratix IV devices, you need not derate using the Preset Editor. You only need to enter the parameters referenced to $V_{REF}$, and the deration is done automatically when you enter the slew rate information on the **Board Settings** tab.

After derating the values, you then need to normalize the derated value because Altera input and output timing specifications are referenced to $V_{REF}$. However, JEDEC base setup time specifications are referenced to $V_{IH}/V_{IL}$ AC levels; JEDEC base hold time specifications are referenced to $V_{IH}/V_{IL}$ DC levels.

When the memory device setup and hold time numbers are derated and normalized to $V_{REF}$, update these values in the **Preset Editor** dialog box to ensure that your timing constraints are correct.

For example, according to JEDEC, 400-MHz DDR2 SDRAM has the following specifications, assuming 1V/ns DQ slew rate rising signal and 2V/ns differential slew rate:

- Base $t_{DS}$ = 50
- Base $t_{DH}$ = 125
- $V_{IH}(ac) = V_{REF} + 0.2$ V
- $V_{IH}(dc) = V_{REF} + 0.125$V
- $V_{IL}(ac) = V_{REF} – 0.2$ V
- $V_{IL}(dc) = V_{REF} – 0.125$ V

JEDEC lists two different sets of base and derating numbers for $t_{DS}$ and $t_{DH}$ specifications, whether you are using single-ended or differential DQS signaling, for any DDR2 SDRAM components with a maximum frequency up to 267 MHz. In addition, the $V_{IL}(ac)$ and $V_{IH}(ac)$ values may also be different for those devices.

The $V_{REF}$ referenced setup and hold signals for a rising edge are:

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

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

If the output slew rate of the write data is different from 1V/ns, you have to first derate the $t_{DS}$ and $t_{DH}$ values, then translate these AC/DC level specs to $V_{REF}$ specification.

For a 2V/ns DQ slew rate rising signal and 2V/ns DQS-DQSn slew rate:

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

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

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

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

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

## PHY Settings

Click **Next** or the **PHY Settings** tab to set the options described in Table 3–6. The options are available if they apply to the target Altera device.

**Table 3–6. ALTMEMPHY PHY Settings (Part 1 of 2)**

| Parameter Name | Applicable Device Families | Description |
|---|---|---|
| Use dedicated PLL outputs to drive memory clocks | HardCopy II and Stratix II (prototyping for HardCopy II) | Turn on to use dedicated PLL outputs to generate the external memory clocks, which is required for HardCopy II ASICs and their Stratix II FPGA prototypes. When turned off, the DDIO output registers generate the clock outputs.<br>When you use the DDIO output registers for the memory clock, both the memory clock and the DQS signals are well aligned and easily meets the $t_{DQSS}$ specification. However, when the dedicated clock outputs are for the memory clock, the memory clock and the DQS signals are not aligned properly and requires a positive phase offset from the PLL to align the signals together. |
| Dedicated memory clock phase | HardCopy II and Stratix II (prototyping for HardCopy II) | The required phase shift to align the CK/CK# signals with DQS/DQS# signals when using dedicated PLL outputs to drive memory clocks. |
| Use differential DQS | Arria II GX, Stratix III, and Stratix IV | Enable this feature for better signal integrity. Recommended for operation at 333 MHz or higher. An option for DDR2 SDRAM only, as DDR SDRAM does not support differential DQSS. |
| Enable external access to reconfigure PLL prior to calibration | HardCopy II, Stratix II, Stratix III, and Stratix IV (prototyping for HardCopy II) | When enabling this option for HardCopy II, Stratix II, Stratix III, and Stratix IV devices, the inputs to the ALTPLL_RECONFIG megafunction are brought to the top level for debugging purposes.<br>This option allows you to reconfigure the PLL before calibration to adjust, if necessary, the phase of the memory clock (`mem_clk_2x`) before the start of the calibration of the resynchronization clock on the read side. The calibration of the resynchronization clock on the read side depends on the phase of the memory clock on the write side. |

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

| Parameter Name | Applicable Device Families | Description |
|---|---|---|
| Instantiate DLL externally | All supported device families, except for Cyclone III devices | Use this option with Stratix III, Stratix IV, HardCopy III, or HardCopy IV devices, if you want to apply a non-standard phase shift to the DQS capture clock. The ALTMEMPHY DLL offsetting I/O can then be connected to the external DLL and the Offset Control Block.<br>As Cyclone III devices do not have DLLs, this feature is not supported. |
| Enable dynamic parallel on-chip termination | Stratix III and Stratix IV | This option provides I/O impedance matching and termination capabilities. The ALTMEMPHY megafunction enables parallel termination during reads and series termination during writes with this option checked. Only applicable for DDR and DDR2 SDRAM interfaces where DQ and DQS are bidirectional. Using the dynamic termination requires that you use the OCT calibration block, which may impose a restriction on your DQS/DQ pin placements depending on your $R_{UP}/R_{DN}$ pin locations.<br>Although DDR SDRAM does not support ODT, dynamic OCT is still supported in Altera FPGAs.<br>For more information, refer to either the *External Memory Interfaces in Stratix III Devices* chapter in volume 1 of the *Stratix III Device Handbook* or the *External Memory Interfaces in Stratix IV Devices* chapter in volume 1 of the *Stratix IV Device Handbook*. |
| Clock phase | Arria II GX, Arria GX, Cyclone III, HardCopy II, Stratix II, and Stratix II GX | Adjusting the address and command phase can improve the address and command setup and hold margins at the memory device to compensate for the propagation delays that vary with different loadings. You have a choice of 0°, 90°, 180°, and 270°, based on the rising and falling edge of the `phy_clk` and `write_clk` signals. In Stratix IV and Stratix III devices, the clock phase is set to **dedicated**. |
| Dedicated clock phase | Stratix III and Stratix IV | When you use a dedicated PLL output for address and command, you can choose any legal PLL phase shift to improve setup and hold for the address and command signals. You can set this value to between 180° and 359°, the default is 240°. However, generally PHY timing requires a value of greater than 240° for half-rate designs and 270° for full-rate designs. |
| Board skew | All supported device families except Arria II GX and Stratix IV devices | Maximum skew across any two memory interface signals for the whole interface from the FPGA to the memory (either a discrete memory device or a DIMM). This parameter includes all types of signals (data, strobe, clock, address, and command signals). You need to input the worst-case skew, whether it is within a DQS/DQ group, or across all groups, or across the address and command and clocks signals. This parameter generates the timing constraints in the **.sdc** file. |
| Autocalibration simulation options | All supported device families | Choose between **Full Calibration** (long simulation time), **Quick Calibration**, or **Skip Calibration**.<br>For more information, refer to the *Simulation* section in volume 4 of the *External Memory Interface Handbook*. |

## Board Settings

Click **Next** or the **Board Settings** tab to set the options described in Table 3–7. The board settings parameters are set to model the board level effects in the timing analysis. The options are available if you choose Arria II GX or Stratix IV device for your interface. Otherwise, the options are disabled.

**Table 3–7. ALTMEMPHY Board Settings**

| Parameter Name | Units | Description |
|---|---|---|
| Number of slots/discrete devices | — | Sets the single-rank or multi-rank configuration. |
| CK/CK# slew rate (differential) | V/ns | Sets the differential slew rate for the CK and CK# signals. |
| Addr/command slew rate | V/ns | Sets the slew rate for the address and command signals. |
| DQ/DQS# slew rate (differential) | V/ns | Sets the differential slew rate for the DQ and DQS# signals. |
| DQ slew rate | V/ns | Sets the slew rate for the DQ signals. |
| Addr/command eye reduction (setup) | ns | Sets the reduction in the eye diagram on the setup side due to the ISI on the address and command signals. |
| Addr/command eye reduction (hold) | ns | Sets the reduction in the eye diagram on the hold side due to the ISI on the address and command signals. |
| DQ eye reduction | ns | Sets the total reduction in the eye diagram on the setup side due to the ISI on the DQ signals. |
| Delta DQS arrival time | ns | Sets the increase of variation on the range of arrival times of DQS due to ISI. |
| Max skew between DIMMs/devices | ns | Sets the largest skew or propagation delay on the DQ signals between ranks, especially true for DIMMs in different slots.<br>This value affects the Resynchronization margin for the DDR2 interfaces in multi-rank configurations for both DIMMs and devices. |
| Max skew within DQS group | ns | Sets the largest skew between the DQ pins in a DQS group. This value affects the Read Capture and Write margins for the DDR2 interfaces in all configurations (single- or multi-rank, DIMM or device). |
| Max skew between DQS groups | ns | Sets the largest skew between DQS signals in different DQS groups. This value affects the Resynchronization margin for the DDR2 interfaces in both single- or multi-rank configurations. |
| Addr/command to CK skew | ns | Sets the skew or propagation delay between the CK signal and the address and command signals. The positive values represent the address and command signals that are longer than the CK signals, and the negative values represent the address and command signals that are shorter than the CK signals. This skew is used by the Quartus II software to optimize the delay of the address/command signals to have appropriate setup and hold margins for the DDR2 interfaces. |

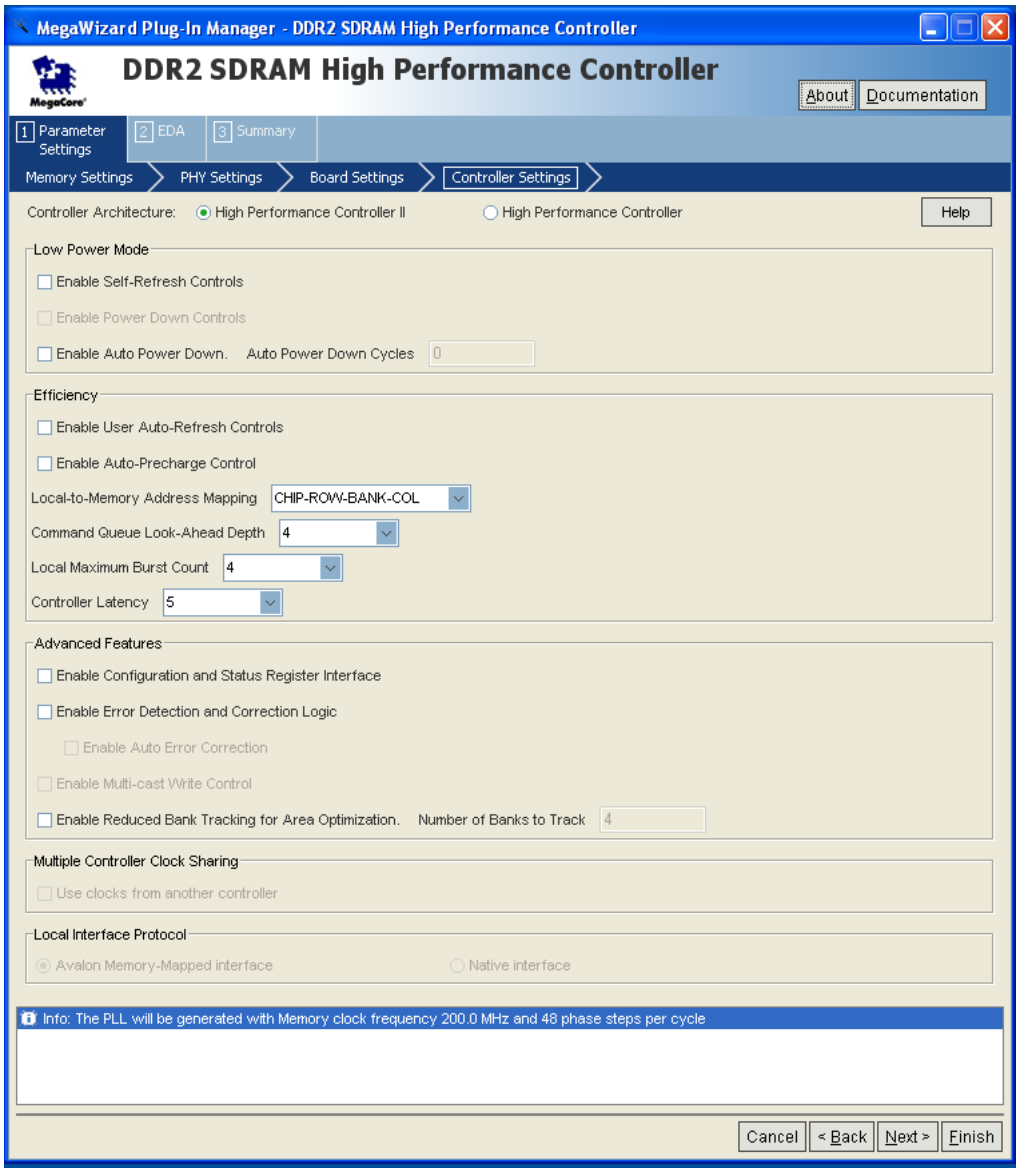
# DDR or DDR2 SDRAM Controller with ALTMEMPHY Parameter Settings

The **Parameter Settings** page in the DDR or DDR2 SDRAM Controller with ALTMEMPHY parameter editor (Figure 3–3) allows you to parameterize the following settings:

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

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

**Figure 3–3. DDR2 SDRAM Controller with ALTMEMPHY Settings**



## Controller Settings

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

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

| Parameter | Controller Architecture | Description |
|---|---|---|
| Controller architecture | — | Specifies the controller architecture. |
| Enable self-refresh controls | Both | Turn on to enable the controller to allow you to have control on when to place the external memory device in self-refresh mode, refer to "User-Controlled Self-Refresh Logic" on page 7–8. |
| Enable power down controls | HPC | Turn on to enable the controller to allow you to have control on when to place the external memory device in power-down mode. |
| Enable auto power down | HPC II | Turn on to enable the controller to automatically place the external memory device in power-down mode after a specified number of idle controller clock cycles is observed in the controller. You can specify the number of idle cycles after which the controller powers down the memory in the **Auto Power Down Cycles** field, refer to "Automatic Power-Down with Programmable Time-Out" on page 7–8. |
| Auto power down cycles | HPC II | Determines the desired number of idle controller clock cycles before the controller places the external memory device in a power-down mode. The legal range is 1 to 65,535.<br>The auto power-down mode is disabled if you set the value to 0 clock cycles. |
| Enable user auto-refresh controls | Both | Turn on to enable the controller to allow you to issue a single refresh. |
| Enable auto-precharge control | HPC | Turn on to enable the auto-precharge control on the controller top level. Asserting the auto-precharge control signal while requesting a read or write burst allows you to specify whether or not the controller should close (auto-precharge) the current opened page at the end of the read or write burst. |
| Local-to-memory address mapping | HPC II | Allows you to control the mapping between the address bits on the Avalon interface and the chip, row, bank, and column bits on the memory interface.<br>If your application issues bursts that are greater than the column size of the memory device, choose the Chip-Row-Bank-Column option. This option allows the controller to use its look-ahead bank management feature to hide the effect of changing the currently open row when the burst reaches the end of the column.<br>On the other hand, if your application has several masters that each use separate areas of memory, choose the Chip-Bank-Row-Column option. This option allows you to use the top address bits to allocate a physical bank in the memory to each master. The physical bank allocation avoids different masters accessing the same bank which is likely to cause inefficiency, as the controller must then open and close rows in the same bank. |

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

| Parameter | Controller Architecture | Description |
|---|---|---|
| Command queue look-ahead depth | HPC II | Specifies a command queue look-ahead depth value to control the number of read or write requests the look-ahead bank management logic examines, refer to "Command Queue" on page 7–5. |
| Local maximum burst count | HPC II | Specifies a burst count to configure the maximum Avalon burst count that the controller slave port accepts. |
| Controller latency | HPC II | Specifies a latency for the controller. The default latency is **5** but you have the option to choose **4** to enhance the latency performance of your design at the expense of timing closure. |
| Enable configuration and status register interface | HPC II | Turn on to enable run-time configuration and status retrieval of the memory controller. Enabling this option adds an additional Avalon-MM slave port to the memory controller top level that allows run-time reconfiguration and status retrieving for memory timing parameters, memory address size and mode register settings, and controller features. If the **Error Detection and Correction Logic** option is enabled, the same slave port also allows you to control and retrieve the status of this logic. Refer to "Configuration and Status Register (CSR) Interface" on page 7–9. |
| Enable error detection and correction logic | Both | Turn on to enable error correction coding (ECC) for single-bit error correction and double-bit error detection. Refer to "Error Correction Coding (ECC)" on page 6–6 for HPC, and "Error Correction Coding (ECC)" on page 7–9 for HPC II. |
| Enable auto error correction | HPC II | Turn on to allow the controller to perform auto correction when the ECC logic detects a single-bit error. Alternatively, you can turn off this option and schedule the error correction at a desired time for better system efficiency. Refer to "Error Correction Coding (ECC)" on page 7–9 for more information. |
| Enable multi-cast write control | HPC II | Turn on to enable the multi-cast write control on the controller top level. Asserting the multi-cast write control when requesting a write burst causes the write data to be written to all the chip selects in the memory system. When you turn on this option together with the **Enable User Auto-Refresh Controls** option, the user refresh commands are issued to all chips.<br>Multi-cast write is not supported for registered DIMM interfaces or when you turn on the **Enable Error Detection and Correction Logic** option. |
| Enable reduced bank tracking for area optimization | HPC II | Turn on to reduce the controller's resource usage. By turning on this option, you reduce the number of bank tracking blocks in the controller. Refer to "Bank Management Logic" on page 7–5 for more information. |
| Number of banks to track | HPC II | Specifies the number of bank tracking blocks you want for your design. This option is only available if you turn on the **Enable Reduced Bank Tracking for Area Optimization** option. The value for this option depends on the value you specify for the **Command Queue Look-Ahead Depth** option. Refer to "Bank Management Logic" on page 7–5 for more information. |

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

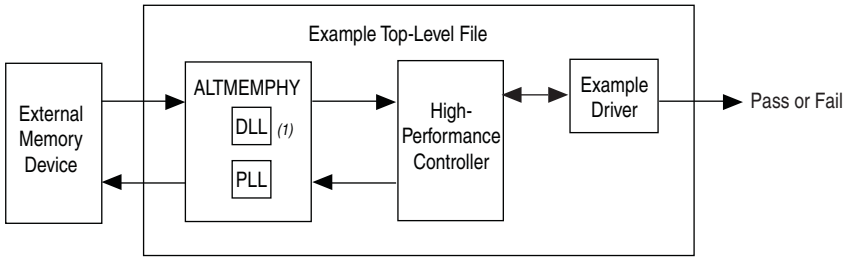
| Parameter | Controller Architecture | Description |
|---|---|---|
| Multiple controller clock sharing | Both | This option is only available in SOPC Builder Flow. Turn on to allow one controller to use the Avalon clock from another controller in the system that has a compatible PLL. This option allows you to create SOPC Builder systems that have two or more memory controllers that are synchronous to your master logic.<br>☞ This option is not for use with Cyclone III or Cyclone IV family devices. |
| Local interface protocol | HPC | Specifies the local side interface between the user logic and the memory controller. The Avalon-MM interface allows you to easily connect to other Avalon-MM peripherals.<br>The HPC II architecture supports only the Avalon-MM interface. |

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

## Compiling the Design

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

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




**Note to Figure 4–1:**

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

Before compiling a design with the ALTMEMPHY variation, you must edit some project settings, include the **.sdc** file, and make I/O assignments. I/O assignments include I/O standard, pin location, and other assignments, such as termination and drive strength settings. Some of these tasks are listed in the ALTMEMPHY **Generation** window. For most systems, Altera recommends that you use the **Advanced I/O Timing** feature by using the **Board Trace Model** command in the Quartus II software to set the termination and output pin loads for the device.

☞ You cannot compile the ALTMEMPHY variation as a stand-alone top-level design because the generated **.sdc** timing constraints file requires the ALTMEMPHY variation be part of a larger design (with a controller and/or example driver). If you want to check whether the ALTMEMPHY variation meets your required target frequency before your memory controller is ready, create a top-level file that instantiates this ALTMEMPHY variation.

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

1. Set up the TimeQuest timing analyzer:
   a. On the Assignments menu, click **Timing Analysis Settings**, select **Use TimeQuest Timing Analyzer during compilation**, and click **OK**.
   b. Add the Synopsys Design Constraints (**.sdc**) file, <*variation name*>**_phy_ddr_timing.sdc**, to your project. On the Project menu, click **Add/Remove Files in Project** and browse to select the file.
   c. Add the **.sdc** file for the example top-level design, <*variation name*>**_example_top.sdc,** to your project. This file is only required if you are using the example as the top-level design.
2. You can either use the <*variation_name*>**_pin_assignments.tcl** or the <*variation_name*>**.ppf** file to apply the I/O assignments generated by the MegaWizard Plug-In Manager. Using the **.ppf** file and the Pin Planner gives you the extra flexibility to add a prefix to your memory interface pin names. You can edit the assignments either in the Assignment Editor or Pin Planner. Use one of the following procedures to specify the I/O standard assignments for pins:

- If you have a single SDRAM interface, and your top-level pins have default naming shown in the example top-level file, run <*variation name*>**_pin_assignments.tcl**.

  or

- If your design contains pin names that do not match the design, edit the <*variation name*>**_pin_assignments.tcl** file before you run the script. To edit the **.tcl** file, perform the following steps:
  a. Open <*variation name*>**_pin_assignments.tcl** file.
  b. Based on the flow you are using, set the `sopc_mode` value to Yes or No.
     - SOPC Builder System flow:

     ```
     if {![info exists sopc_mode]} {set sopc_mode YES}
     ```

     - MegaWizard Plug-In Manager flow:

     ```
     if {![info exists sopc_mode]} {set sopc_mode NO}
     ```

  c. Type your preferred prefix in the `pin_prefix` variable. For example, to add the prefix `my_mem`, do the following:

     ```
     if {![info exists set_prefix]}{set pin_prefix "my_mem_"}
     ```

     After setting the prefix, the pin names are expanded as shown in the following:

     - SOPC Builder System flow:

     `my_mem_cs_n_from_the_`<*your instance name*>

     - MegaWizard Plug-In Manager flow:

     ```
     my_mem_cs_n[0]
     ```

> If your top-level design does not use single bit bus notation for the single-bit memory interface signals (for example, `mem_dqs` rather than `mem_dqs[0]`), in the Tcl script you should change `set single_bit {[0]}` to `set single_bit {}`.

or

- Alternatively, to change the pin names that do not match the design, you can add a prefix to your pin names by performing the following steps:
  a. On the Assignments menu, click **Pin Planner**.
  b. On the Edit menu, click **Create/Import Megafunction**.
  c. Select **Import an existing custom megafunction** and navigate to *<variation name>***.ppf**.
  d. Type the prefix you want to use in **Instance name**. For example, change **mem_addr** to **core1_mem_addr**.

3. Set the top-level entity to the top-level design.
   a. On the File menu, click **Open**.
   b. Browse to your SOPC Builder system top-level design or *<variation name>***_example_top** if you are using MegaWizard Plug-In Manager, and click **Open**.
   c. On the Project menu, click **Set as Top-Level Entity**.
4. Assign the DQ and DQS pin locations.
   a. You should assign pin locations to the pins in your design, so the Quartus II software can perform fitting and timing analysis correctly.
   b. Use either the Pin Planner or Assignment Editor to assign the clock source pin manually. Also choose which DQS pin groups should be used by assigning each DQS pin to the required pin. The Quartus II Fitter then automatically places the respective DQ signals onto suitable DQ pins within each group.

   > To avoid no-fit errors when you compile your design, ensure that you place the `mem_clk` pins to the same edge as the `mem_dq` and `mem_dqs` pins, and set an appropriate I/O standard for the non-memory interfaces, such as the clock source and the reset inputs, when assigning pins in your design. For example, for DDR SDRAM select **2.5 V** and for DDR2 SDRAM select **1.8 V**. Also select in which bank or side of the device you want the Quartus II software to place them.

5. For Stratix III and Stratix IV designs, if you are using advanced I/O timing, specify board trace models in the **Device & Pin Options** dialog box. If you are using any other device and not using advanced I/O timing, specify the output pin loading for all memory interface pins.
6. Select your required I/O driver strength (derived from your board simulation) to ensure that you correctly drive each signal or ODT setting and do not suffer from overshoot or undershoot.
7. To compile the design, on the Processing menu, click **Start Compilation**.

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

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

## Simulating the Design

During system generation, SOPC Builder optionally generates a simulation model and testbench for the entire system, which you can use to easily simulate your system in any of Altera's supported simulation tools. The MegaWizard also generates a set of ModelSim® Tcl scripts and macros that you can use to compile the testbench, IP functional simulation models, and plain-text RTL design files that describe your system in the ModelSim simulation software (refer to "Generated Files" on page 2–6).

For more information about simulating SOPC Builder systems, refer to volume 4 of the *Quartus II Handbook* and *AN 351: Simulating Nios II Embedded Processor Designs*. For more information about how to include your board simulation results in the Quartus II software and how to assign pins using pin planners, refer to *ALTMEMPHY Design Tutorials* section in volume 6 of the *External Memory Interface Handbook*.

In ALTMEMPHY variations for DDR or DDR2 SDRAM interfaces, you have the following simulation options:

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

  Skip calibration mode supports the default ALTMEMPHY parameterization with CAS latency of 3 for DDR memory, and all CAS latencies for DDR2 memory. The additive latency and registered DIMMs must be disabled for all memory types.

  Skip calibration is unavailable for DDR2 RDIMMs.

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

  You may see memory model warnings about initialization times.

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

In quick and skip calibration modes, the ALTMEMPHY megafunction ignores any delays, and assumes that all delays in the testbench and memory model are 0 ps. To successfully simulate a design with delays in the testbench and memory model, you must generate a full calibration mode model in the MegaWizard Plug-In Manager. If you are simulating your ALTMEMPHY-based design with a Denali model, Altera recommends that you use full calibration mode.

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

# 5. Functional Description—ALTMEMPHY

The ALTMEMPHY megafunction creates the datapath between the memory device and the memory controller, and user logic in various Altera devices. The ALTMEMPHY megafunction GUI helps you configure multiple variations of a memory interface. You can then connect the ALTMEMPHY megafunction variation with either a user-designed controller or with an Altera high-performance controller. In addition, the ALTMEMPHY megafunction and the Altera high-performance controllers are available for full-rate and half-rate DDR and DDR2 SDRAM interfaces.

For legacy device families not supported by the ALTMEMPHY megafunction (such as Cyclone, Cyclone II, Stratix, and Stratix GX devices), use the Altera legacy integrated static datapath and controller MegaCore functions.

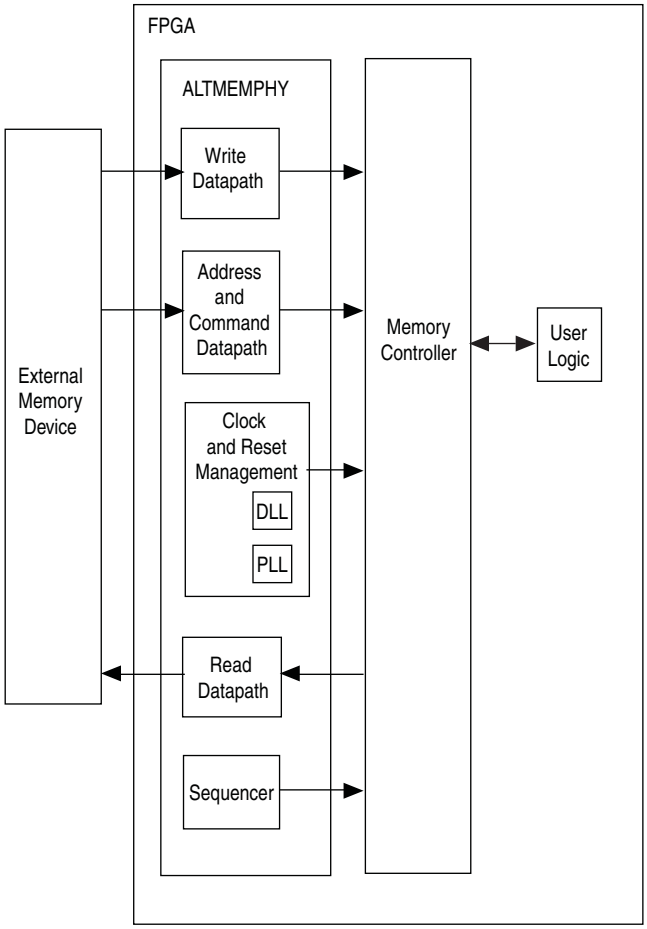
If the ALTMEMPHY megafunction does not meet your requirements, you can also create your own memory interface datapath using the ALTDLL and ALTDQ_DQS megafunctions, available in the Quartus II software. However, you are then responsible for every aspect of the interface, including timing analysis and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

This chapter describes the DDR and DDR2 SDRAM ALTMEMPHY megafunction, which uses AFI as the interface between the PHY and the controller.

# Block Description

Figure 5–1 on page 5–2 shows the major blocks of the ALTMEMPHY megafunction and how it interfaces with the external memory device and the controller. The ALTPLL megafunction is instantiated inside the ALTMEMPHY megafunction, so that you do not need to generate the clock to any of the ALTMEMPHY blocks.

**Figure 5–1. ALTMEMPHY Megafunction Interfacing with the Controller and the External Memory**




The ALTMEMPHY megafunction comprises the following blocks:

- Write datapath
- Address and command datapath
- Clock and reset management, including DLL and PLL
- Sequencer for calibration
- Read datapath

## Calibration

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

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

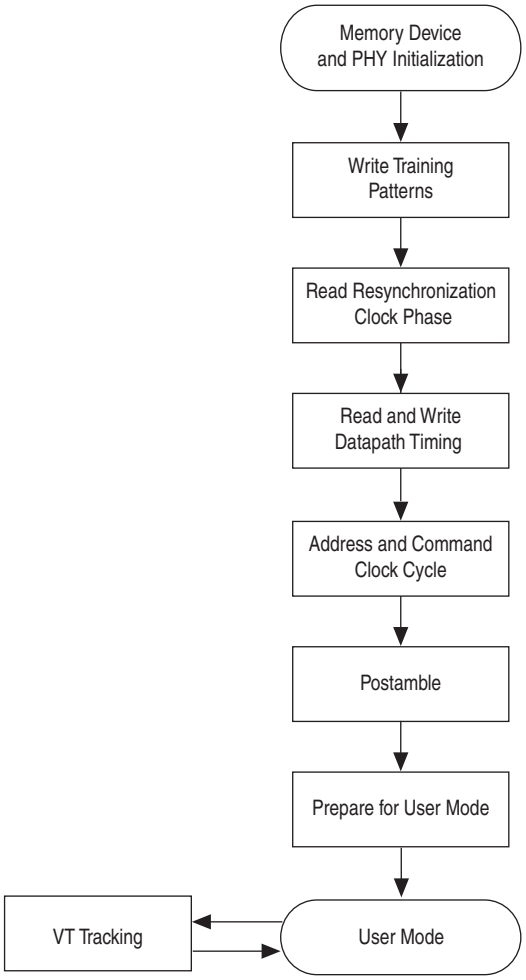
The calibration process for the DDR/DDR2 SDRAM PHY includes the following steps:

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

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

Figure 5–2 shows the calibration flow.

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




## Step 1: Memory Device Initialization

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

## Step 2: Write Training Patterns

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

- All `0`: `'b0000` - DDIO high and low bits held at `0`
- All `1`: `'b1111` - DDIO high and low bits held at `1`
- Toggle: `'b0101` - DDIO high bits held at `0` and DDIO low bits held at `1`

- Mixed: `'b0011` - DDIO high and low bits have to toggle

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

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

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

### Step 3: Read Resynchronization (Capture) Clock Phase

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

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

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

### Step 4: Read and Write Datapath Timing

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

### Step 5: Address and Command Clock Cycle

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

### Step 6: Postamble

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

Postamble is required only for DQS-based capture schemes.

## Step 7: Prepare for User Mode

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

### VT Tracking

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

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

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

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

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

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

### Mimic Path

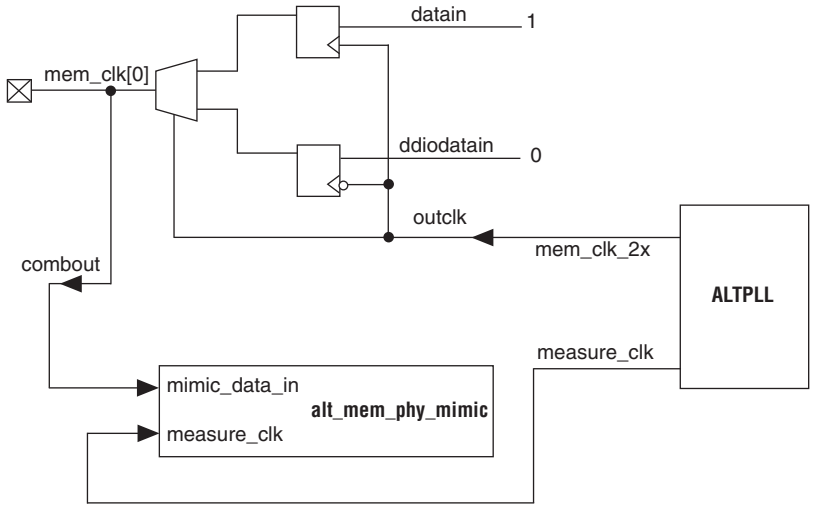
The mimic path mimics the FPGA elements of the round-trip delay, which enables the calibration sequencer to track delay variation due to VT changes during the memory read and write transactions without interrupting the operation of the ALTMEMPHY megafunction.

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

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

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

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




## Address and Command Datapath

This topic describes the address and command datapath.

### Arria GX, Arria II GX, Cyclone III, HardCopy II, Stratix II, and Stratix II GX Devices

The address and command datapath for full-rate designs is similar to half-rate designs, except that the address and command signals are all asserted for one memory clock cycle only (1T signaling).

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

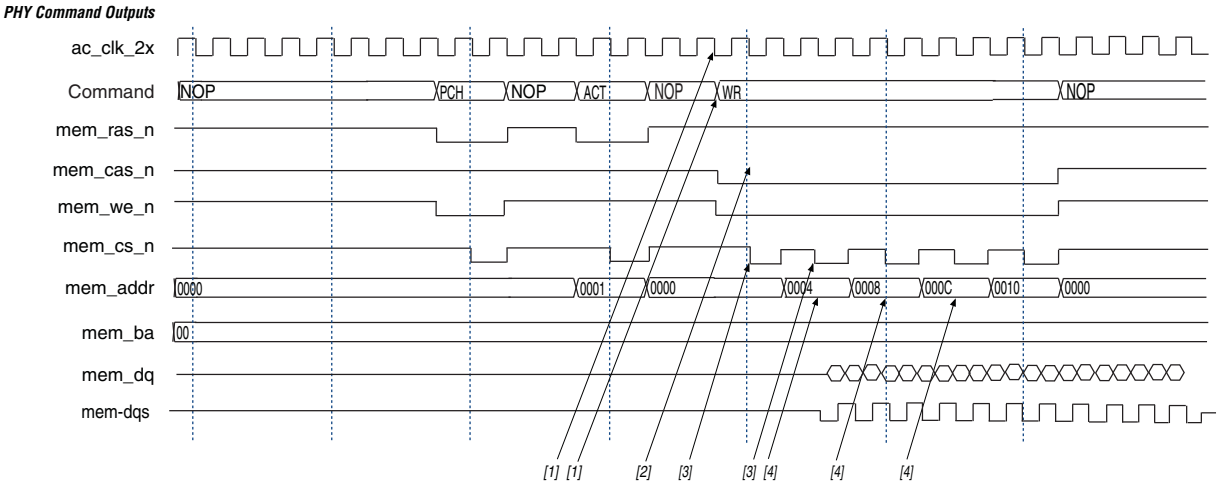
- 1T (full rate)—the duration of the address and command is a single memory clock cycle (`mem_clk_2x`, Figure 5–4). This applies to all address and command signals in full-rate designs or `mem_cs_n`, `mem_cke`, and `mem_odt` signals in half-rate designs.

- 2T (half rate)—the duration of the address and command is two memory clock cycles. For half-rate designs, the ALTMEMPHY megafunction supports only a burst size of four, which means the burst size on the local interface is always set to **1**. The size of the data is 4*n*-bits wide on the local side and is *n*-bits wide on the memory side. To transfer all the 4*n*-bits at the double data rate, two memory-clock cycles are required. The new address and command can be issued to memory every two clock cycles. This scheme applies to all address and command signals, except for `mem_cs_n`, `mem_cke`, and `mem_odt` signals in half-rate mode.

Refer to Table 5–4 in "PLL" on page 5–10 to see the frequency relationship of `mem_clk_2x` with the rest of the clocks.

Figure 5–4 shows a 1T chip select signal (`mem_cs_n`), which is active low, and disables the command in the memory device. All commands are masked when the chip-select signal is inactive. The `mem_cs_n` signal is considered part of the command code.

**Figure 5–4. Arria GX, Arria II GX, Cyclone II, HardCopy II, Stratix II, and Stratix II GX Address and Command Datapath**




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

The waveform in Figure 5–4 shows a NOP command followed by back-to-back write commands. The following sequence corresponds with the numbered items in Figure 5–4:

1. The commands are asserted either on the rising edge of `ac_clk_2x`. The `ac_clk_2x` is derived from either `mem_clk_2x` (0°), `write_clk_2x` (270°), or the inverted variations of those two clocks (for 180° and 90° phase shifts). This depends on the setting of the address and command clock in the ALTMEMPHY parameter editor. Refer to "Address and Command Datapath" on page 5–7 for illustrations of this clock in relation to the `mem_clk_2x` or `write_clk_2x` signals.
2. All address and command signals (except for `mem_cs_ns`, `mem_cke`, and `mem_odt` signals) remain asserted on the bus for two clock cycles, allowing sufficient time for the signals to settle.

3. The mem_cs_n, mem_cke, and mem_odt signals are asserted during the second cycle of the address/command phase. By asserting the chip-select signal in alternative cycles, back-to-back read or write commands can be issued.
4. The address is incremented every other ac_clk_2x cycle.

The ac_clk_2x clock is derived from either mem_clk_2x (when you choose 0° or 180° phase shift) or write_clk_2x (when you choose 90° or 270° phase shift).

The address and command clock can be 0, 90, 180, or 270° from the system clock (refer to "Address and Command Datapath" on page 5–7).

### Stratix III and Stratix IV Devices

The address and command clock in Stratix III and Stratix IV devices is one of the PLL dedicated clock outputs whose phase can be adjusted to meet the setup and hold requirements of the memory clock. The Stratix III address and command clock, ac_clk_1x, is half rate. The command and address pins use the DDIO output circuitry to launch commands from either the rising or falling edges of the clock. The chip select (cs_n) pins and ODT are only enabled for one memory clock cycle and can be launched from either the rising or falling edge of ac_clk_1x signal, while the address and other command pins are enabled for two memory clock cycles and can also be launched from either the rising or falling edge of ac_clk_1x signal.

The full-rate address and command datapath is the same as that of the half-rate address and command datapath, except that there is no full-rate to half-rate conversion in the IOE. The address and command signals are full-rate here.

## Clock and Reset Management

This topic describes the clock and reset management for specific device types.

### Arria GX, Arria II GX, HardCopy II, Stratix II, and Stratix II GX Devices

The clocking and reset block is responsible for clock generation, reset management, and phase shifting of clocks. It also has control of clock network types that route the clocks.

#### Clock Management

The clock management feature allows the ALTMEMPHY megafunction to work out the optimum resynchronization clock phase during calibration, and track the system voltage and temperature (VT) variations. Clock management is achieved by phase-shifting the clocks relative to each other.

Clock management circuitry is implemented by the following device resources:

- PLL
- PLL reconfiguration
- DLL

### PLL

The ALTMEMPHY parameter editor automatically generates an ALTPLL megafunction instance. The ALTPLL megafunction is responsible for generating the different clock frequencies and relevant phases used within the ALTMEMPHY megafunction.

The minimum PHY requirement is to have 16 phases of the highest frequency clock. The PLL uses the **With No Compensation** option to minimize jitter.

You must choose a PLL and PLL input clock pin that are located on the same side of the memory interface to ensure minimal jitter. Cascaded PLLs are not recommended for DDR/DDR2 SDRAM interfaces as jitter can accumulate with the use of cascaded PLLs causing the memory output clock to violate the memory device jitter specification. Also, ensure that the input clock to the PLL is stable before the PLL locks. If not, you must perform a manual PLL reset and relock the PLL to ensure that the phase relationship between all PLL outputs are properly set.

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

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

Table 5–4 shows the clock outputs for Arria GX, HardCopy II, Stratix II, and Stratix II GX devices.

**Table 5–1. DDR/DDR2 SDRAM Clocking in Arria GX, HardCopy II, Stratix II, and Stratix II GX Devices (Part 1 of 3)**

| Design Rate | Clock Name | Postscale Counter | Phase (Degrees) | Clock Rate | Clock Network Type | Notes |
|---|---|---|---|---|---|---|
| Half-rate | `phy_clk_1x` and `aux_half_rate_clk` | C0 | 0° | Half-Rate | Global | The only clocks parameterizable for the ALTMEMPHY megafunction. These clocks also feed into a divider circuit to provide the PLL `scan_clk` signal (for reconfiguration) that must be lower than 100 MHz. |
| | `mem_clk_2x` and `aux_full_rate_clk` | C1 | 0° | Full-Rate | Global | Clocks DQS and as a reference clock for the memory devices. |

**Table 5–1. DDR/DDR2 SDRAM Clocking in Arria GX, HardCopy II, Stratix II, and Stratix II GX Devices (Part 2 of 3)**

| Design Rate | Clock Name | Postscale Counter | Phase (Degrees) | Clock Rate | Clock Network Type | Notes |
|---|---|---|---|---|---|---|
| Full rate | aux_half_rate_clk | C0 | 0° | Half-Rate | Global | The only clock parameterizable for the ALTMEMPHY megafunction. This clock also feeds into a divider circuit to provide the PLL scan_clk signal (for reconfiguration) that must be lower than 100 MHz. |
| | phy_clk_1x *(1)*<br>and<br>mem_clk_2x<br>and<br>aux_full_rate_clk | C1 | 0° | Full-Rate | Global | Clocks DQS and as a reference clock for the memory devices. |
| Half-rate and full rate | write_clk_2x | C2 | –90° | Full-Rate | Global | Clocks the data out of the DDR I/O (DDIO) pins in advance of the DQS strobe (or equivalent). As a result, its phase leads that of the mem_clk_2x by 90°. |
| Half-rate and full rate | mem_clk_ext_2x | C3 | > 0° | Full-Rate | Dedicated | This clock is only used if the memory clock generation uses dedicated output pins. Applicable only in HardCopy II or Stratix II prototyping for HardCopy II designs. |
| Half-rate and full rate | resync_clk_2x | C4 | Calibrated | Full-Rate | Regional | Clocks the resynchronization registers after the capture registers. Its phase is adjusted to the center of the data valid window across all the DQS-clocked DDIO groups. |

**Table 5–1. DDR/DDR2 SDRAM Clocking in Arria GX, HardCopy II, Stratix II, and Stratix II GX Devices (Part 3 of 3)**

| Design Rate | Clock Name | Postscale Counter | Phase (Degrees) | Clock Rate | Clock Network Type | Notes |
|---|---|---|---|---|---|---|
| Half-rate and full rate | measure_clk_2x | C5 | Calibrated | Full-Rate | Regional *(2)* | This clock is for VT tracking. This free-running clock measures relative phase shifts between the internal clock(s) and those being fed back through a mimic path. As a result, the ALTMEMPHY megafunction can track VT effects on the FPGA and compensate for the effects. |
| Half-rate and full rate | ac_clk_2x | — | 0, 90°,180°, 270° | Full-Rate | Global | The ac_clk_2x clock is derived from either mem_clk_2x (when you choose 0° or 180° phase shift) or write_clk_2x (when you choose 90° or 270° phase shift). Refer to "Address and Command Datapath" on page 5–7 for illustrations of the address and command clock relationship with the mem_clk_2x or write_clk_2x signals. |

**Notes to Table 5–4:**

(1) In full-rate designs a _1x clock may run at full rate clock.

(2) This clock should be of the same clock network clock as the resync_clk_2x clock.

For full-rate clock and reset management refer to Table 5–4. The PLL is configured exactly in the same way as in half-rate designs. The PLL information and restriction from half-rate designs also applies.

The phy_clk_1x clock is now full-rate, despite the "1x" naming convention.

You must choose a PLL and PLL input clock pin that are located on the same side of the memory interface to ensure minimal jitter. Cascaded PLLs are not recommended for DDR/DDR2 SDRAM interfaces as jitter can accumulate with the use of cascaded PLLs causing the memory output clock to violate the memory device jitter specification. Also, ensure that the input clock to the PLL is stable before the PLL locks. If not, you must perform a manual PLL reset and relock the PLL to ensure that the phase relationship between all PLL outputs are properly set. The PLL restrictions in half-rate designs also applies to full-rate designs.

Table 5–2 shows the clock outputs that Arria II GX devices use.

**Table 5–2. DDR/DDR2 SDRAM Clocking in Arria II GX Devices (Part 1 of 2)**

| Design Rate | Clock Name *(1)* | Postscale Counter | Phase (Degrees) | Clock Rate | Clock Network Type | Notes |
|---|---|---|---|---|---|---|
| Half-rate | `phy_clk_1x` and `aux_half_rate_clk` | C0 | 0° | Half-Rate | Global | The only clock parameterizable for the ALTMEMPHY megafunction. This clock also feeds into a divider circuit to provide the PLL `scan_clk` signal (for reconfiguration) that must be lower than 100 MHz. |
| | `mem_clk_2x` and `aux_full_rate_clk` | C1 | 0° | Full-Rate | Global | Clocks DQS and as a reference clock for the memory devices. |
| Full rate | `aux_half_rate_clk` | C0 | 0° | Half-Rate | Global | The only clock parameterizable for the ALTMEMPHY megafunction. This clock also feeds into a divider circuit to provide the PLL `scan_clk` signal (for reconfiguration) that must be lower than 100 MHz. |
| | `phy_clk_1x` *(1)* and `mem_clk_2x` and `aux_full_rate_clk` | C1 | 0° | Full-Rate | Global | Clocks DQS and as a reference clock for the memory devices. |
| Half-rate and full rate | Unused | C2 | — | — | — | — |
| Half-rate and full rate | `write_clk_2x` | C3 | –90° | Full-Rate | Global | Clocks the data out of the DDR I/O (DDIO) pins in advance of the DQS strobe (or equivalent). As a result, its phase leads that of the `mem_clk_2x` by 90°. |

**Table 5–2. DDR/DDR2 SDRAM Clocking in Arria II GX Devices (Part 2 of 2)**

| Design Rate | Clock Name *(1)* | Postscale Counter | Phase (Degrees) | Clock Rate | Clock Network Type | Notes |
|---|---|---|---|---|---|---|
| Half-rate and full rate | ac_clk_2x | C3 | 90° | Full-Rate | Global | Address and command clock.<br>The ac_clk_2x clock is derived from either mem_clk_2x (when you choose 0° or 180° phase shift) or write_clk_2x (when you choose 90° or 270° phase shift). Refer tp "Address and Command Datapath" on page 5–7 for illustrations of the address and command clock relationship with the mem_clk_2x or write_clk_2x signals. |
| Half-rate and full rate | cs_n_clk_2x | C3 | 90° | Full-Rate | Global | Memory chip-select clock.<br>The cs_n_clk_2x clock is derived from ac_clk_2x. |
| Half-rate and full rate | resync_clk_2x | C4 | Calibrated | Full-Rate | Global | Clocks the resynchronization registers after the capture registers. Its phase is adjusted to the center of the data valid window across all the DQS-clocked DDIO groups. |
| Half-rate and full rate | measure_clk_2x | C5 | Calibrated | Full-Rate | Global | This clock is for VT tracking. This free-running clock measures relative phase shifts between the internal clock(s) and those being fed back through a mimic path. As a result, the ALTMEMPHY megafunction can track VT effects on the FPGA and compensate for the effects. |

**Note to Table 5–2:**

(1) In full-rate designs, a _1x clock may run at full-rate clock rate.

### PLL Reconfiguration

The ALTMEMPHY parameter editor automatically generates the PLL reconfiguration block by instantiating an ALTPLL_RECONFIG variation for Stratix II and Stratix II GX devices to match the generated ALTPLL megafunction instance. The ALTPLL_RECONFIG megafunction varies the resynchronization clock phase and the measure clock phase.

The ALTMEMPHY parameter editor does not instantiate an ALTPLL_RECONFIG megafunction for Arria II GX devices, as this device uses the dedicated phase stepping I/O on the PLL.

### DLL

A DLL instance is included in the generated ALTMEMPHY variation. When using the DQS to capture the DQ read data, the DLL center-aligns the DQS strobe to the DQ data. The DLL settings depend on the interface clock frequency.

For more information, refer to the *External Memory Interfaces* chapter in the device handbook for your target device family.

### Reset Management

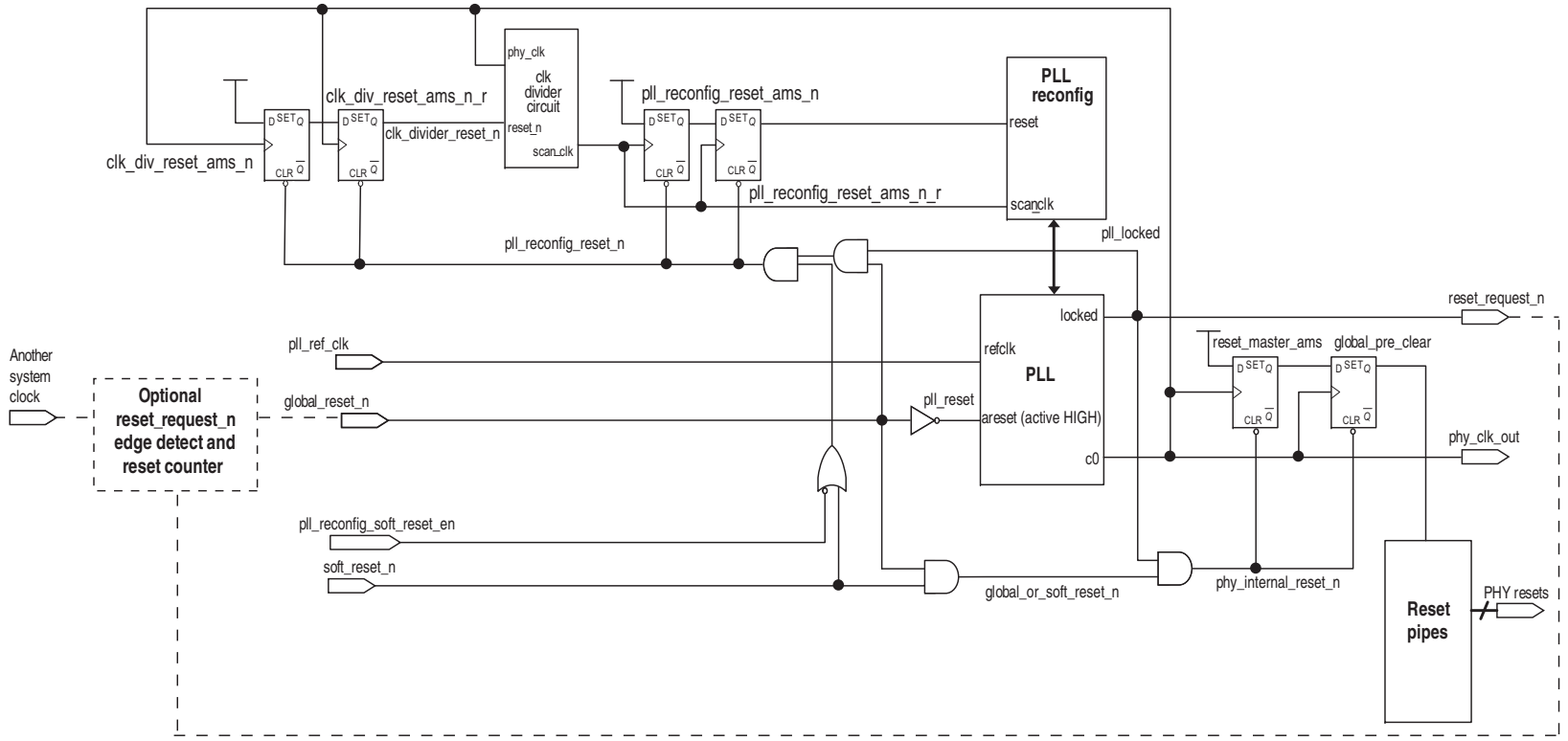
The reset management block is responsible for the following:

- Provides appropriately timed resets to the ALTMEMPHY megafunction datapaths and functional modules
- Performs the reset sequencing required for different clock domains
- Provides reset management of PLL and PLL reconfiguration functions
- Manages any circuit-specific reset sequencing

Each reset is an asynchronous assert and synchronous deassert on the appropriate clock domain. The reset management design uses a standard two-register synchronizer to avoid metastability. A unique reset metastability protection circuit for the clock divider circuit is required because the `phy_clk` domain reset metastability protection flipflops have fan-in from the `soft_reset_n` input, and so these registers cannot be used.

Figure 5–5 shows the ALTMEMPHY reset management block for Arria GX, Arria II GX, HardCopy II, Stratix II, and Stratix II GX devices. The pll_ref_clk signal goes directly to the PLL, eliminating the need for global clock network routing. If you are using the pll_ref_clk signal to feed other parts of your design, you must use a global clock network for the signal. If pll_reconfig_soft_reset_en signal is held low, the PLL reconfig is not reset during a soft reset, which allows designs targeting HardCopy II devices to hold the PHY in reset while still accessing the PLL reconfig block. However, designs targeting Arria GX, Arria II GX, or Stratix II devices are expected to tie the pll_reconfig_soft_en shell to VCC to enable PLL reconfig soft resets.

**Figure 5–5. ALTMEMPHY Reset Management Block for Arria GX, Arria II GX, Cyclone III, HardCopy II, Stratix II, and Stratix II GX Devices** ***(Note 1)***




**Note to Figure 5–5:**

(1) The reset circuit for Arria II GX and Cyclone III devices have no PLL reconfig block.

## Cyclone III Devices

Clock management circuitry is implemented using the ALTPLL megafunction.

The ALTPLL megafunction is instantiated within the ALTMEMPHY megafunction and is responsible for generating all the clocks used by the ALTMEMPHY megafunction and the memory controller.

The minimum PHY requirement is to have 48 phases of the highest frequency clock. The PLL uses Normal mode, unlike other device families. Cyclone III PLL in normal mode emits low jitter already such that you do not require to set the PLL in the **With No Compensation** option. Changing the PLL compensation mode may result in inaccurate timing results.

You must choose a PLL and PLL input clock pin that are located on the same side of the memory interface to ensure minimal jitter. Cascaded PLLs are not recommended as jitter can accumulate with the use of cascaded PLLs causing the memory output clock to violate the memory device jitter specification. Also, ensure that the input clock to the PLL is stable before the PLL locks. If not, you must perform a manual PLL reset and relock the PLL to ensure that the phase relationship between all PLL outputs are properly set.

Table 5–3 lists the clocks generated by the ALTPLL megafunction.

**Table 5–3. DDR/DDR2 SDRAM Clocking in Cyclone III Devices (Part 1 of 2) (Part 1 of 2)**

| Design Rate | Clock Name | Post-Scale Counter | Phase (Degrees) | Clock Rate | Clock Network Type | Notes |
|---|---|---|---|---|---|---|
| Half-rate | `phy_clk_1x` and `aux_half_rate_clk` | C0 | 0° | Half-Rate | Global | The only half-rate clock parameterizable for the ALTMEMPHY megafunction to be used by the controller. This clock is not used in full-rate controllers. This clock also feeds into a divider circuit to provide the PLL `scan_clk` signal for reconfiguration. |
| | `mem_clk_2x` and `aux_full_rate_clk` | C1 | 0° | Full-Rate | Global | Generates DQS signals and the memory clock and to clock the PHY in full-rate mode. |

**Table 5–3. DDR/DDR2 SDRAM Clocking in Cyclone III Devices (Part 2 of 2) (Part 2 of 2)**

| Design Rate | Clock Name | Post-Scale Counter | Phase (Degrees) | Clock Rate | Clock Network Type | Notes |
|---|---|---|---|---|---|---|
| Full rate | aux_half_rate_clk | C0 | 0° | Half-Rate | Global | The only half-rate clock parameterizable for the ALTMEMPHY megafunction to be used by the controller. This clock is not used in full-rate controllers. This clock also feeds into a divider circuit to provide the PLL scan_clk signal for reconfiguration. |
| | phy_clk_1x and mem_clk_2x and aux_full_rate_clk | C1 | 0° | Full-Rate | Global | Generates DQS signals and the memory clock and to clock the PHY in full-rate mode. |
| Half-rate and full rate | write_clk_2x | C2 | -90° | Full-Rate | Global | Clocks the data (DQ) when you perform a write to the memory. |
| Half-rate and full rate | resynch_clk_2x | C3 | Calibrated | Full-Rate | Global | A full-rate clock that captures and resynchronizes the captured read data. The capture and resynchronization clock has a variable phase that is controlled via the PLL reconfiguration logic by the control sequencer block. |
| Half-rate and full rate | measure_clk_2x | C4 | Calibrated | Full-Rate | Global | This clock is for VT tracking. This free-running clock measures relative phase shifts between the internal clock(s) and those being fed back through a mimic path. As a result, you can track VT effects on the FPGA and compensate for them. |
| Half-rate and full rate | ac_clk_2x | — | 0°, 90°, 180°, 270° | Full-Rate | Global | This clock is derived from mem_clk_2x when you choose 0° or 180° phase shift) or write_clk_2x (when you choose 90° or 270° phase shift), refer to "Address and Command Datapath" on page 5–7. |

### Reset Management

The reset management for Cyclone III devices is instantiated in the same way as it is with Stratix II devices.

## Stratix III and Stratix IV Devices

The clocking and reset block is responsible for clock generation, reset management, and phase shifting of clocks. It also has control of clock network types that route the clocks.

The ability of the ALTMEMPHY megafunction to work out the optimum phase during calibration and to track voltage and temperature variation relies on phase shifting the clocks relative to each other.

☞ Certain clocks need to be phase shifted during the ALTMEMPHY megafunction operation.

Clock management circuitry is implemented by using:

- PLL
- DLL

### PLL

The ALTMEMPHY parameter editor automatically generates an ALTPLL megafunction instance. The ALTPLL megafunction is responsible for generating the different clock frequencies and relevant phases used within the ALTMEMPHY megafunction.

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

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

For more information about the VCO frequency range and the available phase shifts, refer to the *Clock Networks and PLLs in Stratix III Devices* chapter in volume 1 of the *Stratix III Device Handbook* or the *Clock Networks and PLLs in Stratix IV Devices* chapter in volume 1 of the *Stratix IV Device Handbook*.

For Stratix IV and Stratix III devices, the PLL reconfiguration is done using the phase-shift inputs on the PLL instead of using the PLL reconfiguration megafunction. Table 5–4 shows the Stratix IV and Stratix III PLL clock outputs.

**Table 5–4. DDR2 SDRAM Clocking in Stratix IV and Stratix III Devices (Part 1 of 2)**

| Design Rate | Clock Name *(1)* | Postscale Counter | Phase (Degrees) | Clock Rate | Clock Network Type | Notes |
|---|---|---|---|---|---|---|
| Half-rate | `phy_clk_1x` and `aux_half_rate_clk` | C0 | 30 | Half-Rate | Global | The only clock parameterizable for the ALTMEMPHY megafunction. It is set to **30°** to ensure proper half-rate to full-rate transfer for write data and DQS. This clock also feeds into a divider circuit to provide the PLL `scan_clk` signal for reconfiguration. |
| | `aux_full_rate_clk` | C2 | 60 | Full-Rate | Global | The `aux_clk`. The 60° -offset maintains edge alignment with the offset on `phy_clk_1x`. |
| Full-rate | `aux_half_rate_clk` | C0 | 0 | Half-Rate | Global | The `aux_clk`. |
| | `phy_clk_1x` and `aux_full_rate_clk` | C2 | 0 | Full-Rate | Global | The only clock parameterizable for the ALTMEMPHY megafunction. This clock also feeds into a divider circuit to provide the PLL `scan_clk` signal for reconfiguration. |
| Half-rate and full-rate | `mem_clk_2x` | C1 | 0 | Full-Rate | Special | Generates `mem_clk` that provides the reference clock for the DLL. A dedicated routing resource exists from the PLL to the DLL, which you select with the regional routing resource for the `mem_clk` using the following attribute in the HDL: `(-name global_signal dual_regional _clock; -to dll~DFFIN -name global_signal off)`. If you use an external DLL, apply this attribute similarly to the external DLL. |

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

| Design Rate | Clock Name *(1)* | Postscale Counter | Phase (Degrees) | Clock Rate | Clock Network Type | Notes |
|---|---|---|---|---|---|---|
| Half-rate and full-rate | write_clk_2x | C3 | –90 | Full-Rate | Dual regional | Clocks the data out of the double data rate input/output (DDIO) pins in advance of the DQS strobe (or equivalent). As a result, its phase leads that of the mem_clk_2x clock by 90°. |
| Half-rate and full-rate | resync_clk_2x | C4 | Calibrated | Full-Rate | Dual regional | This clock feeds the I/O clock divider that then clocks the resynchronization registers after the capture registers. Its phase is adjusted in the calibration process. You can use an inverted version of this clock for postamble clocking. |
| Half-rate and full-rate | measure_clk_1x *(2)* | C5 | Calibrated | Half-Rate | Dual regional | This clock is for VT tracking. This free-running clock measures relative phase shifts between the internal clock(s) and those being fed back through a mimic path. As a result, the ALTMEMPHY megafunction can track VT effects on the FPGA and compensate for the effects. |
| Half-rate and full-rate | ac_clk_1x | C6 | Set in the GUI | Half-Rate | Dual regional | Address and command clock. |

**Notes to Table 5–4:**

(1) In full-rate designs a _1x clock may run at full-rate clock rate.

(2) This clock should be of the same clock network clock as the resync_clk_2x clock.

Clock and reset management for full-rate designs is similar to half-rate support (see Table 5–4 on page 5–20). The PLL is configured exactly in the same way as for half-rate support. The mem_clk_2x output acts as the PHY full-rate clock. Also, instead of going through the I/O clock divider, the resync_clk_2x output is now directly connected to the resynchronization registers. The rest of the PLL outputs are connected in the same way as for half-rate support.

You must choose a PLL and PLL input clock pin that are located on the same side of the device as the memory interface to ensure minimal jitter. Cascaded PLLs are not recommended as jitter can accumulate, causing the memory output clock to violate the memory device jitter specification. Also, ensure that the input clock to the PLL is stable before the PLL locks. If not, you must perform a manual PLL reset (by driving the `global_reset_n` signal low) and relock the PLL to ensure that the phase relationship between all PLL outputs are properly set. The PLL restrictions in half-rate designs also applies to full-rate designs.

### DLL

DLL settings are set depending on the memory clock frequency of operation.

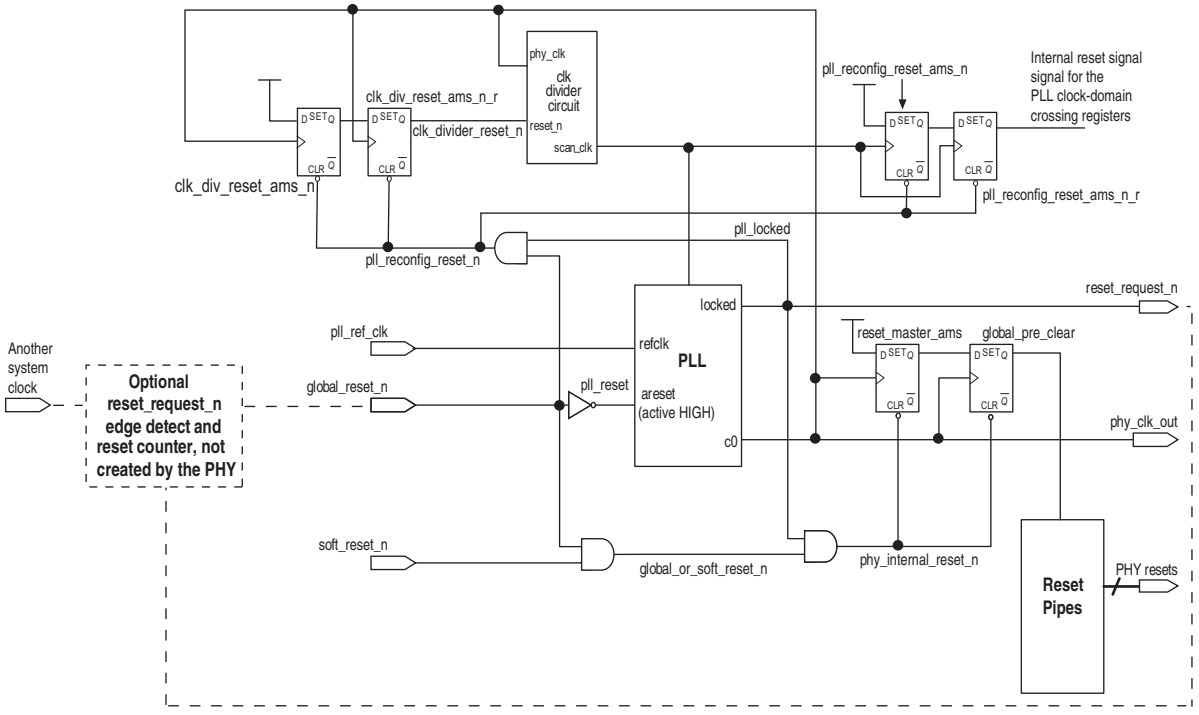
For more information on the DLL, refer to the *External Memory Interfaces in Stratix III Devices* chapter in volume 1 of the *Stratix III Device Handbook* and the *External Memory Interfaces in Stratix IV Devices* chapter in volume 1 of the *Stratix IV Device Handbook*.

### Reset Management

Figure 5–6 shows the main features of the reset management block for the DDR3 SDRAM PHY. You can use the `pll_ref_clk` input to feed the optional `reset_request_n` edge detect and reset counter module. However, this requires the `pll_ref_clk` signal to use a global clock network resource.

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

**Figure 5–6. ALTMEMPHY Reset Management Block for Stratix IV and Stratix III Devices**



# Read Datapath

This topic describes the read datapath.

## Arria GX, Arria II GX, HardCopy II, Stratix II, and Stratix II GX Devices

The following section discusses support for DDR/DDR2 SDRAM for Arria GX, Arria II GX, HardCopy II, Stratix II, and Stratix II GX devices.

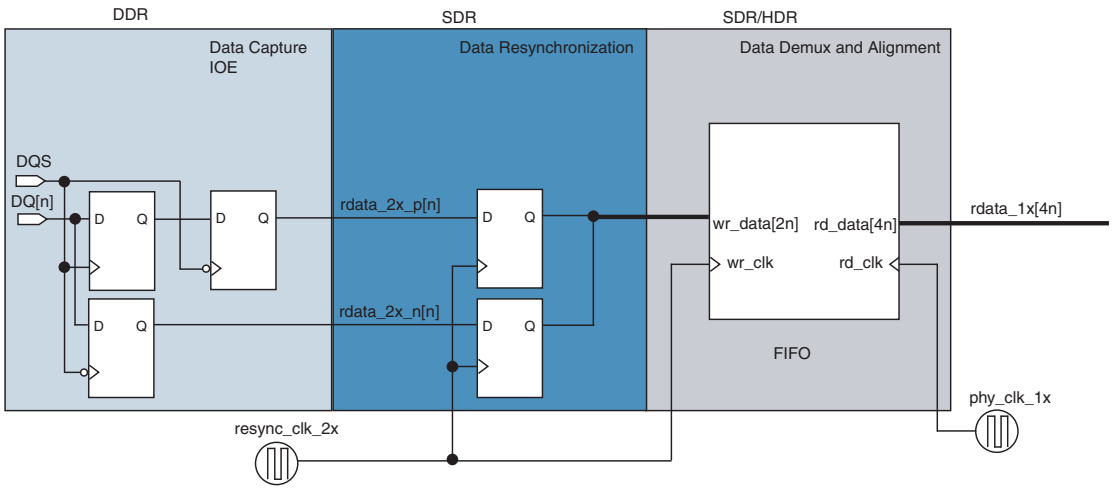
The full-rate datapath is similar to the half-rate datapath. The full-rate datapath also consists of a RAM with the same width as the data input (just like that of the half-rate), but the width on the data output of the RAM is half that of the half-rate PHY. The function of the RAM is to transfer the read data from the resynchronization clock domain to the system clock domain.

The read datapath logic is responsible for capturing data sent by the memory device and subsequently aligning the data back to the system clock domain. The following functions are performed by the read datapath:

1. Data capture and resynchronization
2. Data demultiplexing
3. Data alignment

Figure 5–7 shows the order of the functions performed by the read datapath, along with the frequency at which the read data is handled.

**Figure 5–7. DDR/DDR2 SDRAM Read Datapath in Arria GX, Arria II GX, HardCopy II, Stratix II, and Stratix II GX Devices** ***(Note 1)***




**Note to Figure 5–7:**

(1) In Arria II GX devices the resynchronization register is implemented in IOE.

### Data Capture and Resynchronization

Data capture and resynchronization is the process of capturing the read data (DQ) with the DQS strobe and re-synchronizing the captured data to an internal free-running full-rate clock supplied by the enhanced phase-locked loop (PLL).

The resynchronization clock is an intermediate clock whose phase shift is determined during the calibration stage.

Timing constraints ensure that the data resynchronization registers are placed close to the DQ pins to achieve maximum performance. Timing constraints also further limit skew across the DQ pins. The captured data (`rdata_2x_p` and `rdata_2x_n`) is synchronized to the resynchronization clock (`resync_clk_2x`), refer to Figure 5–7.

### Data Demultiplexing

Data demultiplexing is the process of SDR data into HDR data. Data demultiplexing is required to bring the frequency of the resynchronized data down to the frequency of the system clock, so that data from the external memory device can ultimately be brought into the FPGA DDR or DDR2 SDRAM controller clock domain. Before data capture, the data is DDR and *n*-bit wide. After data capture, the data is SDR and 2*n*-bit wide. After data demuxing, the data is HDR of width 4*n*-bits wide. The system clock frequency is half the frequency of the memory clock.

Demultiplexing is achieved using a dual-port memory with a 2*n*-bit wide write-port operating on the resynchronization clock (SDR) and a 4n-bit wide read-port operating on the PHY clock (HDR). The basic principle of operation is that data is written to the memory at the SDR rate and read from the memory at the HDR rate while incrementing the read- and write-address pointers. As the SDR and HDR clocks are generated, the read and write pointers are continuously incremented by the same PLL, and the 4n-bit wide read data follows the 2n-bit wide write data with a constant latency.

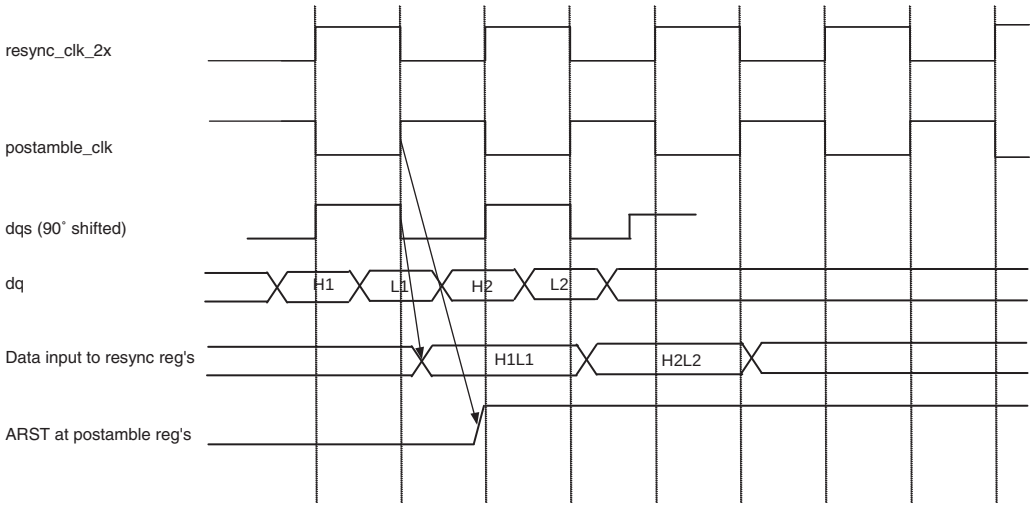
### Read Data Alignment

Data alignment is the process controlled by the sequencer to ensure the correct captured read data is present in the same half-rate clock cycle at the output of the read data DPRAM. Data alignment is implemented using either M4K or M512K memory blocks. The bottom of Figure 5–8 shows the concatenation of the read data into valid HDR data.

### Postamble Protection

The ALTMEMPHY megafunction provides the DQS postamble logic. The postamble clock is derived from the resynchronization clock and is the negative edge of the resynchronization clock. The ALTMEMPHY megafunction calibrates the resynchronization clock such that it is in the center of the data-valid window. The clock that controls the postamble logic, the postamble clock, is the negative edge of the resynchronization clock. No additional clocks are required. Figure 5–8 shows the relationship between the postamble clock and the resynchronization clock.

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




**Note to Figure 5–8:**

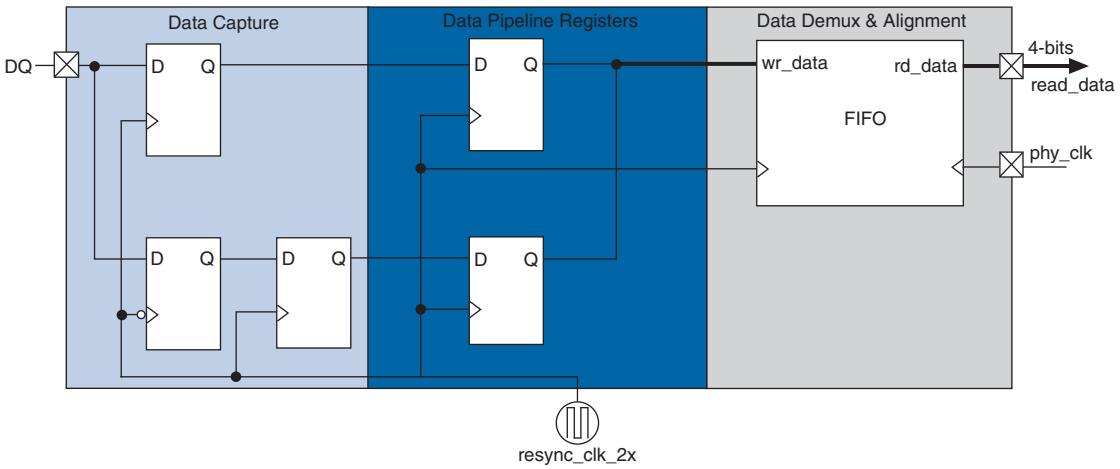
(1) `resync_clk_2x` is delayed further to allow for the I/O element (IOE) to core transition time.

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

## Cyclone III Devices

Figure 5–9 shows the Cyclone III read datapath for a single DQ pin. The diagram shows a half-rate read path where four data bits are produced for each DQ pin. Unlike Stratix II and Stratix III devices, data capture is entirely done in the core logic because the I/O element (IOE) does not contain DDIO capture registers (nonDQS capture).

**Figure 5–9. Cyclone III Read Datapath**




The full-rate read datapath for Cyclone III devices is similar to the half-rate Cyclone III implementation, except that the data is read out of the FIFO buffer with a full-rate clock instead of a half-rate clock.

### Capture and Pipelining

The DDR and DDR2 SDRAM read data is captured using registers in the Cyclone III FPGA core. These capture registers are clocked using the capture clock (`resynch_clk_2x`). The captured read data generates two data bits per DQ pin; one data bit for the read data captured by the rising edge of the capture clock and one data bit for the read data captured by the falling edge of the capture clock.

After the read data has been captured, it may be necessary to insert registers in the read datapath between the capture registers and the read data FIFO buffer to help meet timing. These registers are known as pipeline registers and are clocked off the same clock used by the capture registers, the capture clock (`resync_clk_2x`).

### Data Demultiplexing

The data demultiplexing for Cyclone III devices is instantiated in the same way as it is with Stratix II devices.

### Postamble Protection

Postamble protection circuitry is not required in the Cyclone III device implementation as DQS mode capture of the DQ data is not supported. The data capture is done using the clock (`resync_clk_2x`) generated from the ALTPLL megafunction.

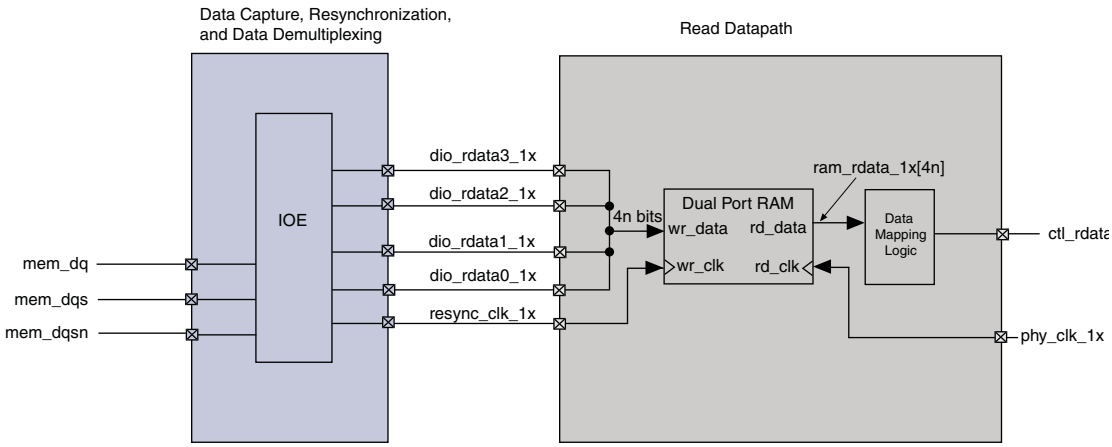
## Stratix III and Stratix IV Devices

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

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

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

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




**Note to Figure 5–10:**

(1) This figure shows a half-rate variation. For a full-rate controller, `dio_radata2_1x` and `dio_rdata3_1x` are unconnected.

### Data Capture, Resynchronization, and Demultiplexing

In Stratix IV and Stratix III devices, the smart interface module in the IOE performs the following tasks:

- Captures the data
- Resynchronizes the captured data from the DQS domain to the resynchronization clock (`resync_clk_1x`) domain
- Converts the resynchronized data into half-rate data, which is performed by feeding the resynchronized data into the HDR conversion block within the IOE, which is clocked by the half-rate version of the resynchronization clock. The `resync_clk_1x` signal is generated from the I/O clock divider module based on the `resync_clk_2x` signal from the PLL.

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

### Data Resynchronization

The read datapath block performs the following two tasks:

1. Transfers the captured read data (`rdata[n]_1x`) from the half-rate resynchronization clock (`resync_clk_1x`) domain to the half-rate system clock (`phy_clk_1x`) domain using DPRAM. Resynchronized data from the FIFO buffer is shown as `ram_data_1x`.
2. Reorders the resynchronized data (`ram_rdata_1x`) into `ctl_mem_rdata`.

The full-rate datapath is similar to the half-rate datapath, except that the resynchronization FIFO buffer converts from the full-rate resynchronization clock domain (`resync_clk_2x`) to the full-rate PHY clock domain, instead of converting it to the half-rate PHY clock domain as in half-rate designs.

### Postamble Protection

A dedicated postamble register controls the gating of the shifted DQS signal that clocks the DQ input registers at the end of a read operation. This ensures that any glitches on the DQS input signals at the end of the read postamble time do not cause erroneous data to be captured as a result of postamble glitches. The postamble path is also calibrated to determine the correct clock cycle, clock phase shift, and delay chain settings. You can see the process in simulation if you choose Full calibration (long simulation time) mode in the MegaWizard Plug-In Manager.

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

## Write Datapath

This topic describes the write datapath.

### Arria GX, Arria II GX, Cyclone III, HardCopy II, Stratix II, and Stratix II GX Devices

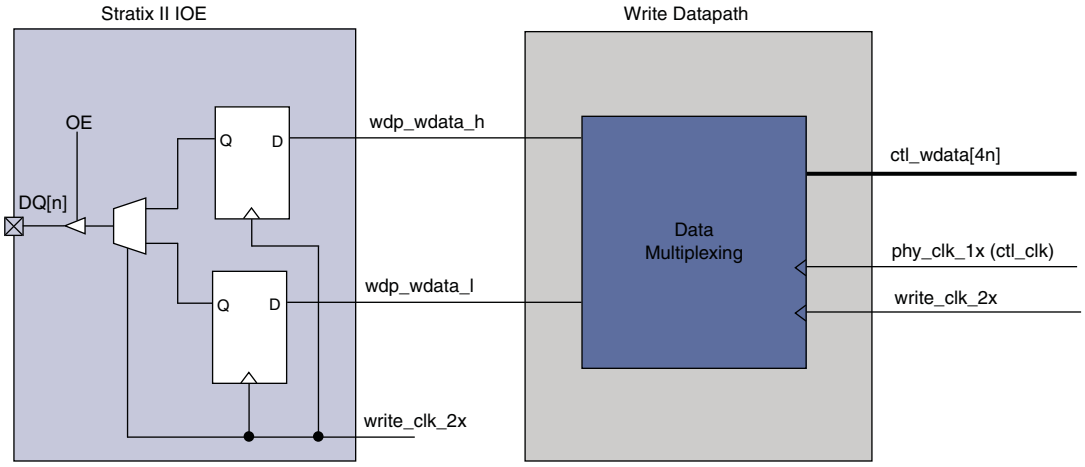
The write datapath logic efficiently transfers data from the HDR memory controller to DDR SDRAM-based memories. The write datapath logic consists of:

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

The memory controller interface outputs 4*n*-bit wide data (`ctl_wdata[4n]`) at half-rate frequency. Figure 5–11 shows that the HDR write data (`ctl_wdata[4n]`) is clocked by the half-rate clock `phy_clk_1x` and is converted into SDR which is represented by `wdp_wdata_h` and `wdp_wdata_l` and clocked by the full-rate clock `write_clk_2x`.

The DQ IOEs convert 2-n SDR bits to n-DDR bits.

**Figure 5–11. DDR/DDR2 SDRAM Write Datapath in Arria GX, Arria II GX, Cyclone III, HardCopy II, Stratix II, and Stratix II GX Devices**

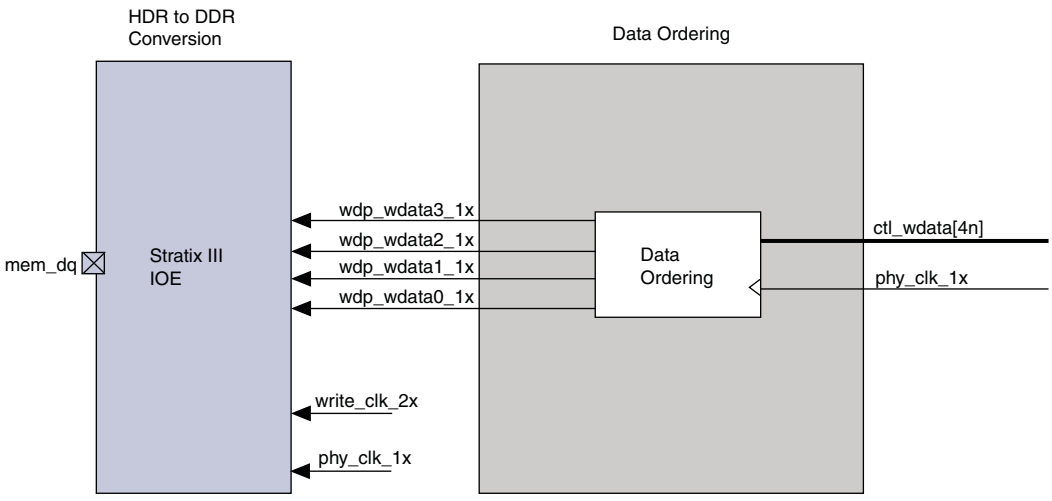



The write datapath for full-rate PHYs is similar to the half-rate PHY. The IOE block is identical to the half-rate PHY. The latency of the write datapath in the full-rate PHY is less than in the half-rate PHY because the full-rate PHY does not have the half-rate-to-full-rate conversion logic.

## Stratix III and Stratix IV Devices

The memory controller interface outputs 4 *n*-bit wide data (`ctl_wdata`) at `phy_clk_1x` frequency. The write data is clocked by the system clock `phy_clk_1x` at half data rate and reordered into HDR of width 4 *n*-bits represented in Figure 5–12 by `wdp_wdata3_1x, wdp_wdata2_1x, wdp_wdata1_1x`, and `wdp_wdata0_1x`.

**Figure 5–12. DDR and DDR2 SDRAM Write Datapath in Stratix IV and Stratix III Devices**




All of the write datapath registers in the Stratix IV and Stratix III devices are clocked by the half-rate clock, `phy_clk_1x`.

For full-rate controllers, `phy_clk_1x` runs at full rate and there are only two bits of `wdata`.

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

For more information about the Stratix III I/O structure, refer to the *External Memory Interfaces in Stratix III Devices* chapter in volume 1 of the *Stratix III Device Handbook* and the *External Memory Interfaces in Stratix IV Devices* chapter in volume 1 of the *Stratix IV Device Handbook.*

# ALTMEMPHY Signals

This section describes the ALMEMPHY megafunction ports for AFI variants.

Table 5–5 through Table 5–7 show the signals.

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

The signal lists include the following signal groups:

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

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

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

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

| Signal Name | Type | Width *(2)* | Description |
|---|---|---|---|
| mem_dqs | Bidirectional | MEM_IF_DWIDTH/ MEM_IF_DQ_PER_DQS | The memory bidirectional data strobe bus. |
| mem_dqsn | Bidirectional | MEM_IF_DWIDTH/ MEM_IF_DQ_PER_DQS | The memory bidirectional data strobe bus. |
| mem_odt | Output | MEM_IF_CS_WIDTH | The memory on-die termination control signal. |
| mem_ras_n | Output | 1 | The memory row address strobe. |
| mem_reset_n | Output | 1 | The memory reset signal. This signal is derived from the PHY's internal reset signal, which is generated by gating the global reset, soft reset, and the PLL locked signal. |
| mem_we_n | Output | 1 | The memory write enable signal. |

**Notes to Table 5–5:**

(1) Connected to I/O pads.

(2) Refer to Table 5–8 for parameter description.

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

**Table 5–6. AFI Signals (Part 1 of 4)**

| Signal Name | Type | Width *(1)* | Description |
|---|---|---|---|
| **Clocks and Resets** | | | |
| pll_ref_clk | Input | 1 | The reference clock input to the PHY PLL. |
| global_reset_n | Input | 1 | Active-low global reset for PLL and all logic in the PHY. A level set reset signal, which causes a complete reset of the whole system. The PLL may maintain some state information. |
| soft_reset_n | Input | 1 | Edge detect reset input intended for SOPC Builder use or to be controlled by other system reset logic. Causes a complete reset of PHY, but not the PLL used in the PHY. |
| reset_request_n | Output | 1 | Directly connected to the locked output of the PLL and is intended for optional use either by automated tools such as SOPC Builder or could be manually ANDed with any other system-level signals and combined with any edge detect logic as required and then fed back to the global_reset_n input.<br>Reset request output that indicates when the PLL outputs are not locked. Use this as a reset request input to any system-level reset controller you may have. This signal is always low while the PLL is locking (but not locked), and so any reset logic using it is advised to detect a reset request on a falling-edge rather than by level detection. |
| ctl_clk | Output | 1 | Half-rate clock supplied to controller and system logic. The same signal as the non-AFI phy_clk. |
| ctl_reset_n | Output | 1 | Reset output on ctl_clk clock domain. |

**Table 5–6. AFI Signals (Part 2 of 4)**

| Signal Name | Type | Width *(1)* | Description |
|---|---|---|---|
| **Other Signals** | | | |
| aux_half_rate_clk | Output | 1 | In half-rate designs, a copy of the phy_clk_1x signal that you can use in other parts of your design, same as phy_clk port. |
| aux_full_rate_clk | Output | 1 | In full-rate designs, a copy of the mem_clk_2x signal that you can use in other parts of your design. |
| aux_scan_clk | Output | 1 | Low frequency scan clock supplied primarily to clock any user logic that interfaces to the PLL and DLL reconfiguration interfaces. |
| aux_scan_clk_reset_n | Output | 1 | This reset output asynchronously asserts (drives low) when global_reset_n is asserted and de-assert (drives high) synchronous to aux_scan_clk when global_reset_n is deasserted. It allows you to reset any external circuitry clocked by aux_scan_clk. |
| **Write Data Interface** | | | |
| ctl_dqs_burst | Input | MEM_IF_DQS_WIDTH × DWIDTH_RATIO / 2 | When asserted, mem_dqs is driven. The ctl_dqs_burst signal must be asserted before ctl_wdata_valid and must be driven for the correct duration to generate a correctly timed mem_dqs signal. |
| ctl_wdata_valid | Input | MEM_IF_DQS_WIDTH × DWIDTH_RATIO / 2 | Write data valid. Generates ctl_wdata and ctl_dm output enables. |
| ctl_wdata | Input | MEM_IF_DWIDTH × DWIDTH_RATIO | Write data input from the controller to the PHY to generate mem_dq. |
| ctl_dm | Input | MEM_IF_DM_WIDTH × DWIDTH_RATIO | DM input from the controller to the PHY. |
| ctl_wlat | Output | 5 | Required write latency between address/command and write data that is issued to ALTMEMPHY controller local interface.<br>This signal is only valid when the ALTMEMPHY sequencer successfully completes calibration, and does not change at any point during normal operation.<br>The legal range of values for this signal is 0 to 31; and the typical values are between 0 and ten, 0 mostly for low CAS latency DDR memory types. |

**Table 5–6. AFI Signals (Part 3 of 4)**

| Signal Name | Type | Width *(1)* | Description |
|---|---|---|---|
| **Read Data Interface** | | | |
| ctl_doing_rd | Input | MEM_IF_DQS_WIDTH × DWIDTH_RATIO / 2 | Doing read input. Indicates that the DDR or DDR2 SDRAM controller is currently performing a read operation.<br>The controller generates ctl_doing_rd to the ALTMEMPHY megafunction. The ctl_doing_rd signal is asserted for one phy_clk cycle for every read command it issues. If there are two read commands, ctl_doing_rd is asserted for two phy_clk cycles. The ctl_doing_rd signal also enables the capture registers and generates the ctl_mem_rdata_valid signal. The ctl_doing_rd signal should be issued at the same time the read command is sent to the ALTMEMPHY megafunction. |
| ctl_rdata | Output | DWIDTH_RATIO × MEM_IF_DWIDTH | Read data from the PHY to the controller. |
| ctl_rdata_valid | Output | DWIDTH_RATIO/2 | Read data valid indicating valid read data on ctl_rdata. This signal is two-bits wide (as only half-rate or DWIDTH_RATIO = 4 is supported) to allow controllers to issue reads and writes that are aligned to either the half-cycle of the half-rate clock. |
| ctl_rlat | Output | READ_LAT_WIDTH | Contains the number of clock cycles between the assertion of ctl_doing_rd and the return of valid read data (ctl_rdata). This is unused by the Altera high-performance controllers do not use ctl_rlat. |
| **Address and Command Interface** | | | |
| ctl_addr | Input | MEM_IF_ROWADDR_WIDTH × DWIDTH_RATIO / 2 | Row address from the controller. |
| ctl_ba | Input | MEM_IF_BANKADDR_WIDTH × DWIDTH_RATIO / 2 | Bank address from the controller. |
| ctl_cke | Input | MEM_IF_CS_WIDTH × DWIDTH_RATIO / 2 | Clock enable from the controller. |
| ctl_cs_n | Input | MEM_IF_CS_WIDTH ×DWIDTH_RATIO / 2 | Chip select from the controller. |
| ctl_odt | Input | MEM_IF_CS_WIDTH × DWIDTH_RATIO / 2 | On-die-termination control from the controller. |
| ctl_ras_n | Input | DWIDTH_RATIO / 2 | Row address strobe signal from the controller. |
| ctl_we_n | Input | DWIDTH_RATIO / 2 | Write enable. |
| ctl_cas_n | Input | DWIDTH_RATIO / 2 | Column address strobe signal from the controller. |
| ctl_rst_n | Input | DWIDTH_RATIO / 2 | Reset from the controller. |
| **Calibration Control and Status Interface** | | | |
| ctl_mem_clk_disable | Input | MEM_IF_CLK_PAIR_COUNT | When asserted, mem_clk and mem_clk_n are disabled. Not supported for Cyclone III devices. |
| ctl_cal_success | Output | 1 | A 1 indicates that calibration was successful. |
| ctl_cal_fail | Output | 1 | A 1 indicates that calibration has failed. |

**Table 5–6. AFI Signals (Part 4 of 4)**

| Signal Name | Type | Width *(1)* | Description |
|---|---|---|---|
| ctl_cal_req | Input | 1 | When asserted, a new calibration sequence is started. Currently not supported. |
| ctl_cal_byte_lane_sel_n | Input | MEM_IF_DQS_WIDTH × MEM_CS_WIDTH | Indicates which DQS groups should be calibrated. |

**Note to Table 5–5:**

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

**Table 5–7. Other Interface Signals (Part 1 of 3)**

| Signal Name | Type | Width | Description |
|---|---|---|---|
| **External DLL Signals** | | | |
| dqs_delay_ctrl_export | Output | DQS_DELAY_CTL_WIDTH | Allows sharing DLL in this ALTMEMPHY instance with another ALTMEMPHY instance. Connect the dqs_delay_ctrl_export port on the ALTMEMPHY instance with a DLL to the dqs_delay_ctrl_import port on the other ALTMEMPHY instance. |
| dqs_delay_ctrl_import | Input | DQS_DELAY_CTL_WIDTH | Allows the use of DLL in another ALTMEMPHY instance in this ALTMEMPHY instance. Connect the dqs_delay_ctrl_export port on the ALTMEMPHY instance with a DLL to the dqs_delay_ctrl_import port on the other ALTMEMPHY instance. |
| dqs_offset_delay_ctrl_ width | Input | DQS_DELAY_CTL_WIDTH | Connects to the DQS delay logic when dll_import_export is set to IMPORT. Only connect if you are using a DLL offset, which can otherwise be tied to zero. If you are using a DLL offset, connect this input to the offset_ctrl_out output of the dll_offset_ctrl block. |
| dll_reference_ clk | Output | 1 | Reference clock to feed to an externally instantiated DLL. This clock is typically from one of the PHY PLL outputs. |
| **User-Mode Calibration OCT Control Signals** | | | |
| oct_ctl_rs_value | Input | 14 | OCT RS value port for use with ALT_OCT megafunction if you want to use OCT with user-mode calibration. |
| oct_ctl_rt_value | Input | 14 | OCT RT value port for use with ALT_OCT megafunction if you want to use OCT with user-mode calibration. |
| **Debug Interface Signals** *(Note 1)*, ***(Note 2)*** | | | |
| dbg_clk | Input | 1 | Debug interface clock. |
| dbg_reset_n | Input | 1 | Debug interface reset. |
| dbg_addr | Input | DBG_A_WIDTH | Address input. |
| dgb_wr | Input | 1 | Write request. |
| dbg_rd | Input | 1 | Read request. |
| dbg_cs | Input | 1 | Chip select. |
| dbg_wr_data | Input | 32 | Debug interface write data. |
| dbg_rd_data | Output | 32 | Debug interface read data. |
| dbg_waitrequest | Output | 1 | Wait signal. |

**Table 5–7. Other Interface Signals (Part 2 of 3)**

| Signal Name | Type | Width | Description |
|---|---|---|---|
| **PLL Reconfiguration Signals—Stratix III and Stratix IV Devices** | | | |
| pll_reconfig_enable | Input | 1 | This signal enables the PLL reconfiguration I/O, and is used if the user requires some custom PLL phase reconfiguration. It should otherwise be tied low. |
| pll_phasecounterselect | Input | 4 | When pll_reconfig_enable is asserted, this input is directly connected to the PLL's phasecounterselect input. Otherwise this input has no effect. |
| pll_phaseupdown | Input | 1 | When pll_reconfig_enable is asserted, this input is directly connected to the PLL's phaseupdown input. Otherwise this input has no effect. |
| pll_phasestep | Input | 1 | When pll_reconfig_enable is asserted, this input is directly connected to the PLL's phasestep input. Otherwise this input has no effect. |
| pll_phase_done | Output | 1 | Directly connected to the PLL's phase_done output. |
| **PLL Reconfiguration Signals—Stratix II Devices** | | | |
| pll_reconfig_enable | Input | 1 | Allows access to the PLL reconfiguration block. This signal should be held low in normal operation. While the PHY is held in reset (with soft_reset_n), and reset_request_n is 1, it is safe to reconfigure the PLL. To reconfigure the PLL, set this signal to 1 and use the other pll_reconfig signals to access the PLL. When finished reconfiguring set this signal to 0, and then set the soft_reset_n signal to 1 to bring the PHY out of reset. For this signal to work, the PLL_RECONFIG_PORTS_EN GUI parameter must be set to TRUE. |
| pll_reconfig_write_param | Input | 9 | Refer to the *ALTPLL_RECONFIG User Guide*, for more information. |
| pll_reconfig_read_param | Input | 9 | Refer to the *ALTPLL_RECONFIG User Guide*, for more information. |
| pll_reconfig | Input | 1 | Refer to the *ALTPLL_RECONFIG User Guide*, for more information. |
| pll_reconfig_counter_type | Input | 4 | Refer to the *ALTPLL_RECONFIG User Guide*, for more information. |
| pll_reconfig_counter_param | Input | 3 | Refer to the *ALTPLL_RECONFIG User Guide*, for more information. |
| pll_reconfig_data_in | Input | 9 | Refer to the *ALTPLL_RECONFIG User Guide* for more information. |
| pll_reconfig_busy | Output | 1 | Refer to the *ALTPLL_RECONFIG User Guide*, for more information. |
| pll_reconfig_data_out | Output | 9 | Refer to the *ALTPLL_RECONFIG User Guide*, for more information. |
| pll_reconfig_clk | Output | 1 | Synchronous clock to use for any logic accessing the pll_reconfig interface. The same as aux_scan_clk. |
| pll_reconfig_reset | Output | 1 | Resynchronised reset to use for any logic accessing the pll_reconfig interface. |

**Table 5–7. Other Interface Signals (Part 3 of 3)**

| Signal Name | Type | Width | Description |
|---|---|---|---|
| **Calibration Interface Signals—without leveling only** | | | |
| rsu_codvw_phase | Output | — | The sequencer sweeps the phase of a resynchronization clock across 360° or 720° of a memory clock cycle. Data reads from the DIMM are performed for each phase position, and a data valid window is located, which is the set of resynchronization clock phase positions where data is successfully read. The final resynchronization clock phase is set at the center of this range: the center of the data valid window or CODVW. This output is set to the current calculated value for the CODVW, and represents how many phase steps were performed by the PLL to offset the resynchronization clock from the memory clock. |
| rsu_codvw_size | Output | — | The final centre of data valid window size (rsu_codvw_size) is the number of phases where data was successfully read in the calculation of the resynchronization clock centre of data valid window phase (rsu_codvw_phase). |
| rsu_read_latency | Output | — | The rsu_read_latency output is then set to the read latency (in phy_clk cycles) using the rsu_codvw_phase resynchronization clock phase. If calibration is unsuccessful then this signal is undefined. |
| rsu_no_dvw_err | Output | — | If the sequencer sweeps the resynchronization clock across every phase and does not see any valid data at any phase position, then calibration fails and this output is set to 1. |
| rsu_grt_one_dvw_err | Output | — | If the sequencer sweeps the resynchronization clock across every phase and sees multiple data valid windows, this is indicative of unexpected read data (random bit errors) or an incorrectly configured PLL that must be resolved. Calibration has failed and this output is set to 1. |
| **Unused Signals** | | | |
| hc_scan_enable_access | Input | — | This signal is unused; you should connect this signal to logic level 0. |
| hc_scan_enable_dq | Input | — | This signal is unused; you should connect this signal to logic level 0. |
| hc_scan_enable_dm | Input | — | This signal is unused; you should connect this signal to logic level 0. |
| hc_scan_enable_dqs | Input | — | This signal is unused; you should connect this signal to logic level 0. |
| hc_scan_enable_dqs_config | Input | — | This signal is unused; you should connect this signal to logic level 0. |
| hc_scan_din | Input | — | This signal is unused; you should connect this signal to logic level 0. |
| hc_scan_update | Input | — | This signal is unused; you should connect this signal to logic level 0. |
| hc_scan_ck | Input | — | This signal is unused; you should connect this signal to logic level 0. |
| hc_scan_dout | Output | — | This signal is unused; you should connect this signal to logic level 0. |
| mem_err_out_*n* | Input | — | This signal is unused; you should connect this signal to logic level 1. |

**Notes to Table 5–7:**

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

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

Table 5–8 shows the parameters that Table 5–5 through Table 5–7 refer to.

**Table 5–8. Parameters**

| Parameter Name | Description |
|---|---|
| DWIDTH_RATIO | The data width ratio from the local interface to the memory interface. DWIDTH_RATIO of 2 means full rate, while DWIDTH_RATIO of 4 means half rate. |
| LOCAL_IF_DWIDTH | The width of the local data bus must be quadrupled for half-rate and doubled for full-rate. |
| MEM_IF_DWIDTH | The data width at the memory interface. MEM_IF_DWIDTH can have values that are multiples of MEM_IF_DQ_PER_DQS. |
| MEM_IF_DQS_WIDTH | The number of DQS pins in the interface. |
| MEM_IF_ROWADDR_WIDTH | The row address width of the memory device. |
| MEM_IF_BANKADDR_WIDTH | The bank address with the memory device. |
| MEM_IF_CS_WIDTH | The number of chip select pins in the interface. The sequencer only calibrates one chip select pin. |
| MEM_IF_DM_WIDTH | The number of mem_dm pins on the memory interface. |
| MEM_IF_DQ_PER_DQS | The number of mem_dq[] pins per mem_dqs pin. |
| MEM_IF_CLK_PAIR_COUNT | The number of mem_clk/mem_clk_n pairs in the interface. |

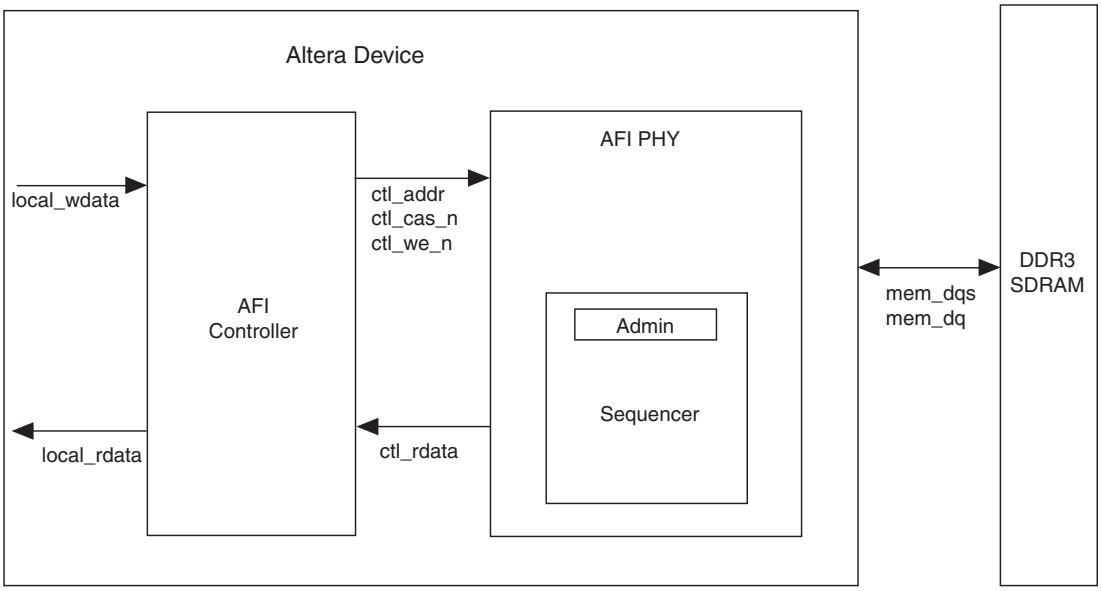
# PHY-to-Controller Interfaces

The following section describes the typical modules that are connected to the ALTMEMPHY variation and the port name prefixes each module uses. This section also describes using a custom controller. This section describes the AFI.

The AFI standardizes and simplifies the interface between controller and PHY for all Altera memory designs, thus allowing you to easily interchange your own controller code with Altera's high-performance controllers. The AFI includes an administration block that configures the memory for calibration and performs necessary mode registers accesses to configure the memory as required (these calibration processes are different). Figure 5–13 shows an overview of the connections between the PHY, the controller, and the memory device.

☞ Altera recommends that you use the AFI for new designs.

**Figure 5–13. AFI PHY Connections**




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

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

For half-rate designs, the AFI allows the controller to issue reads and writes that are aligned to either half-cycle of the half-rate `phy_clk`, which means that the datapaths can support multiple data alignments—word-unaligned and word-aligned writes and reads. Figure 5–14 and Figure 5–15 display the half-rate write operation.

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

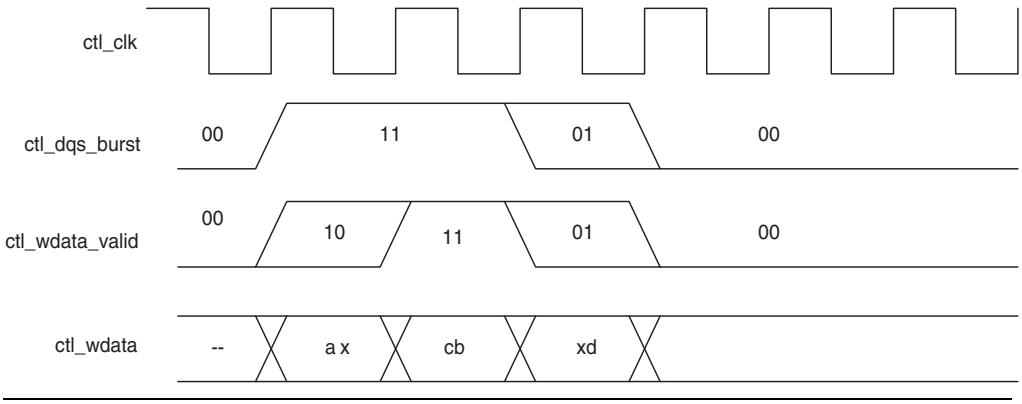



**Figure 5–15. Half-Rate Write with Word-Aligned Data**

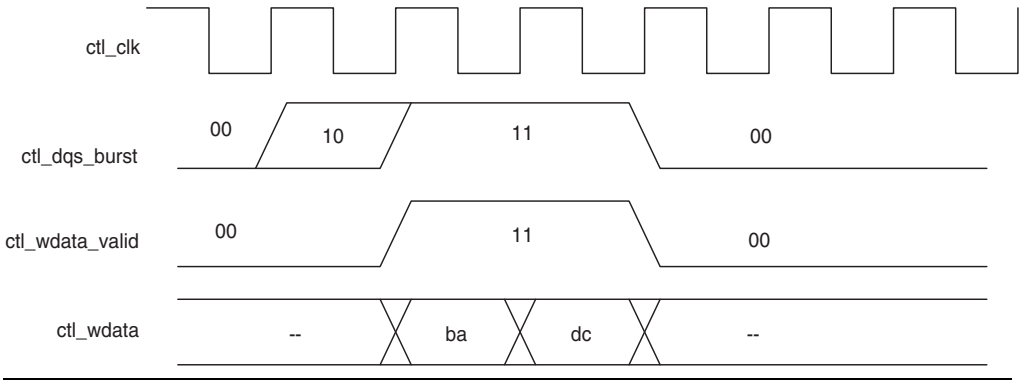
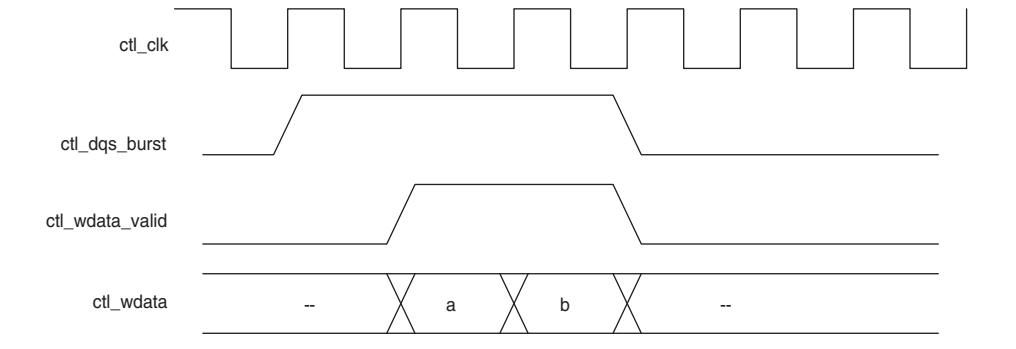



Figure 5–16 shows a full-rate write.

**Figure 5–16. Full-Rate Write**




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

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

**Figure 5–17. Full-Rate Reads**

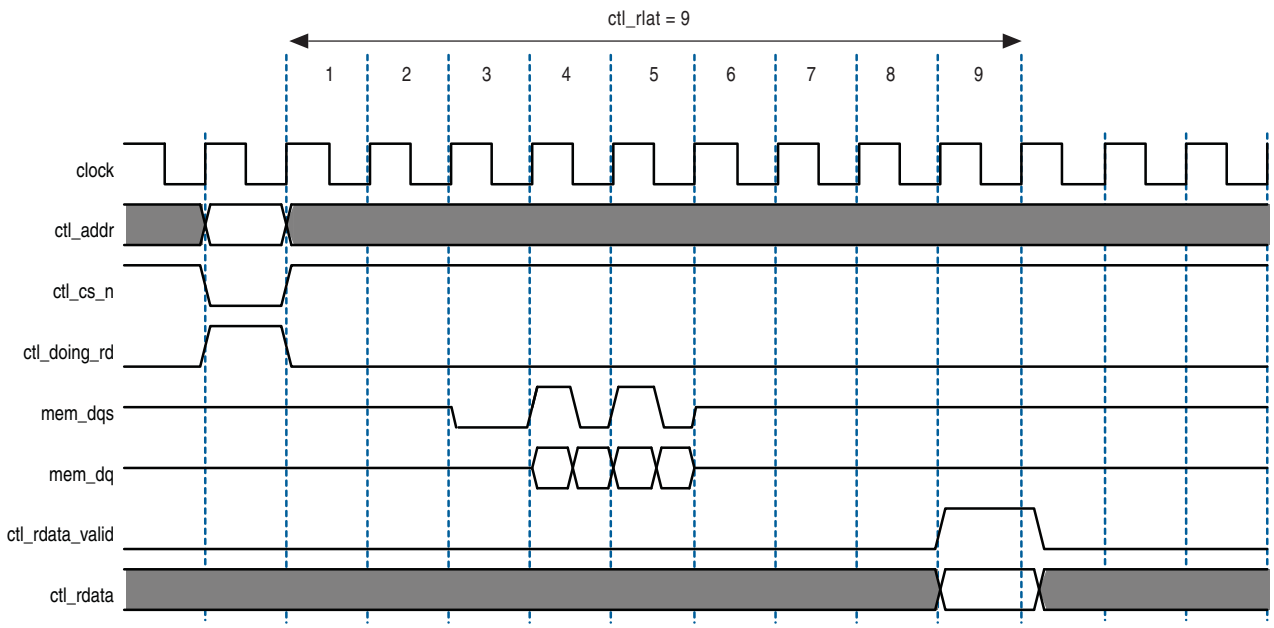



**Figure 5–18. Half-Rate Reads**

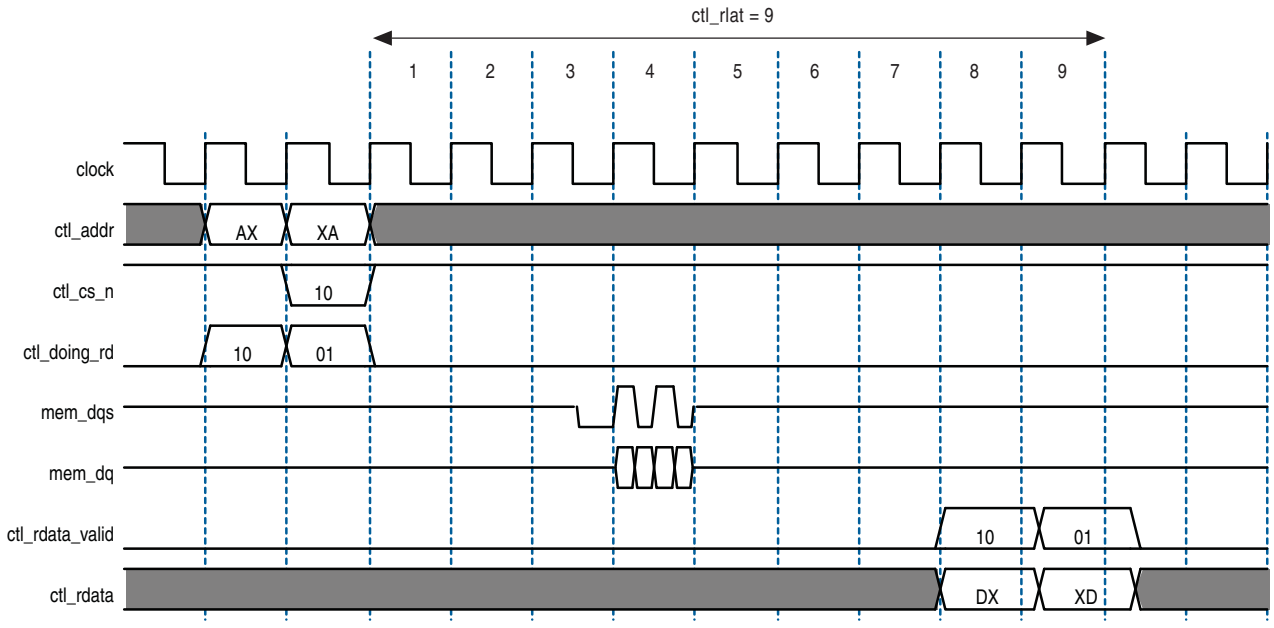



Figure 5–19 and Figure 5–20 show word-aligned writes and reads. In the following read and write examples the data is written to and read from the same address. In each example, `ctl_rdata` and `ctl_wdata` are aligned with controller clock (`ctl_clk`) cycles. All the data in the bit vector is valid at once. For comparison, refer Figure 5–21 and Figure 5–22 that show the word-unaligned writes and reads.

The ctl_doing_rd is represented as a half-rate signal when passed into the PHY. Therefore, the lower half of this bit vector represents one memory clock cycle and the upper half the next memory clock cycle. Figure 5–22 on page 5–45 shows separated word-unaligned reads as an example of two ctl_doing_rd bits are different. Therefore, for each x16 device, at least two ctl_doing_rd bits need to be driven, and two ctl_rdata_valid bits need to be interpreted.

The AFI has the following conventions:

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

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

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

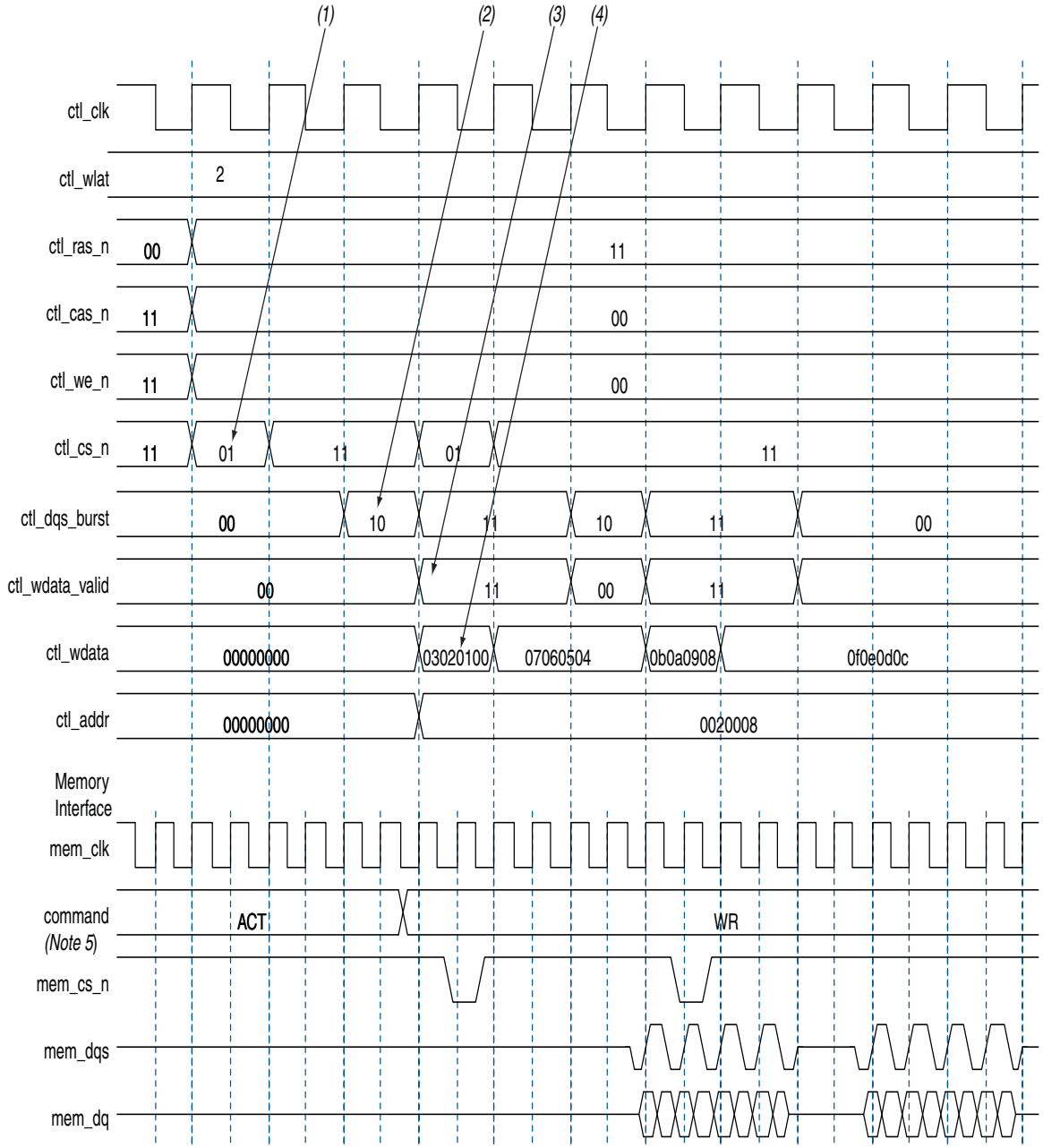
  The Altera high-performance controllers use word-aligned data only. The timing analysis script does not support word-unaligned reads and writes. Word-unaligned reads and writes are only supported on Stratix III and Stratix IV devices.

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

Figure 5–19 through Figure 5–22 assume the following general points:

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

**Figure 5–19. Word-Aligned Writes**




**Notes to Figure 5–19:**

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

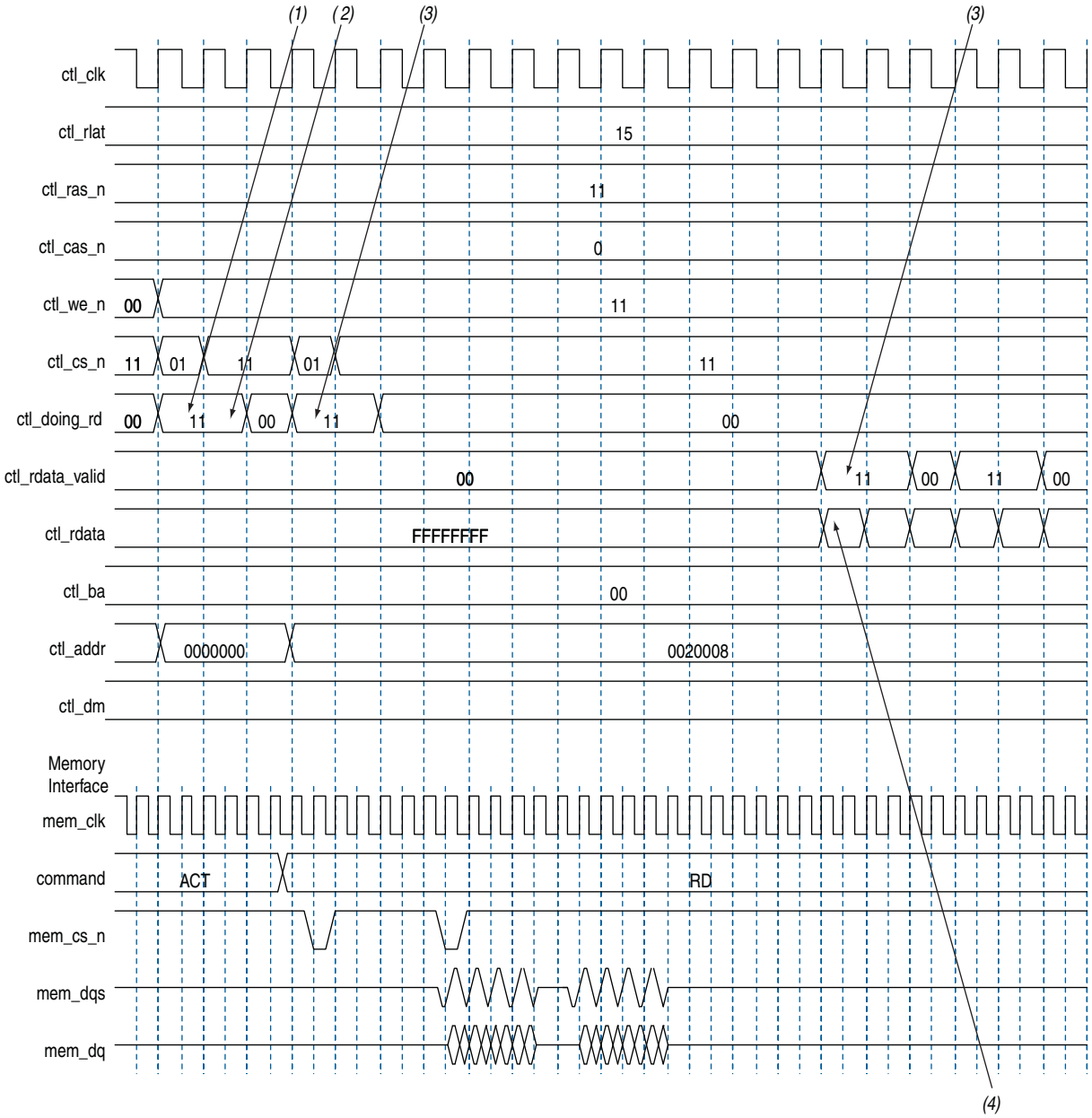
(2) The `ctl_dqs_burst` must go high one memory clock cycle before `ctl_wdata_valid`. Compare with the word-unaligned case.

(3) The `ctl_wdata_valid` is asserted two `ctl_wlat` controller clock (`ctl_clk`) cycles after chip select (`ctl_cs_n`) is asserted. The `ctl_wlat` indicates the required write latency in the system. The value is determined during calibration and is dependant upon the relative delays in the address and command path and the write datapath in both the PHY and the external DDR SDRAM subsystem. The controller must drive `ctl_cs_n` and then wait `ctl_wlat` (two in this example) `ctl_clks` before driving `ctl_wdata_valid`.

(4) Observe the ordering of write data (`ctl_wdata`). Compare this to data on the `mem_dq` signal.

(5) In all waveforms a command record is added that combines the memory pins `ras_n`, `cas_n` and `we_n` into the current command that is issued. This command is registered by the memory when chip select (`mem_cs_n`) is low. The important commands in the presented waveforms are WR = write, ACT = activate.

**Figure 5–20. Word-Aligned Reads**




**Notes to Figure 5–20:**

(1) For AFI, `ctl_doing_rd` is required to be asserted one memory clock cycle before chip select (`ctl_cs_n`) is asserted. In the half-rate `ctl_clk` domain, this requirement manifests as the controller driving `11` (as opposed to the `01`) on `ctl_doing_rd`.

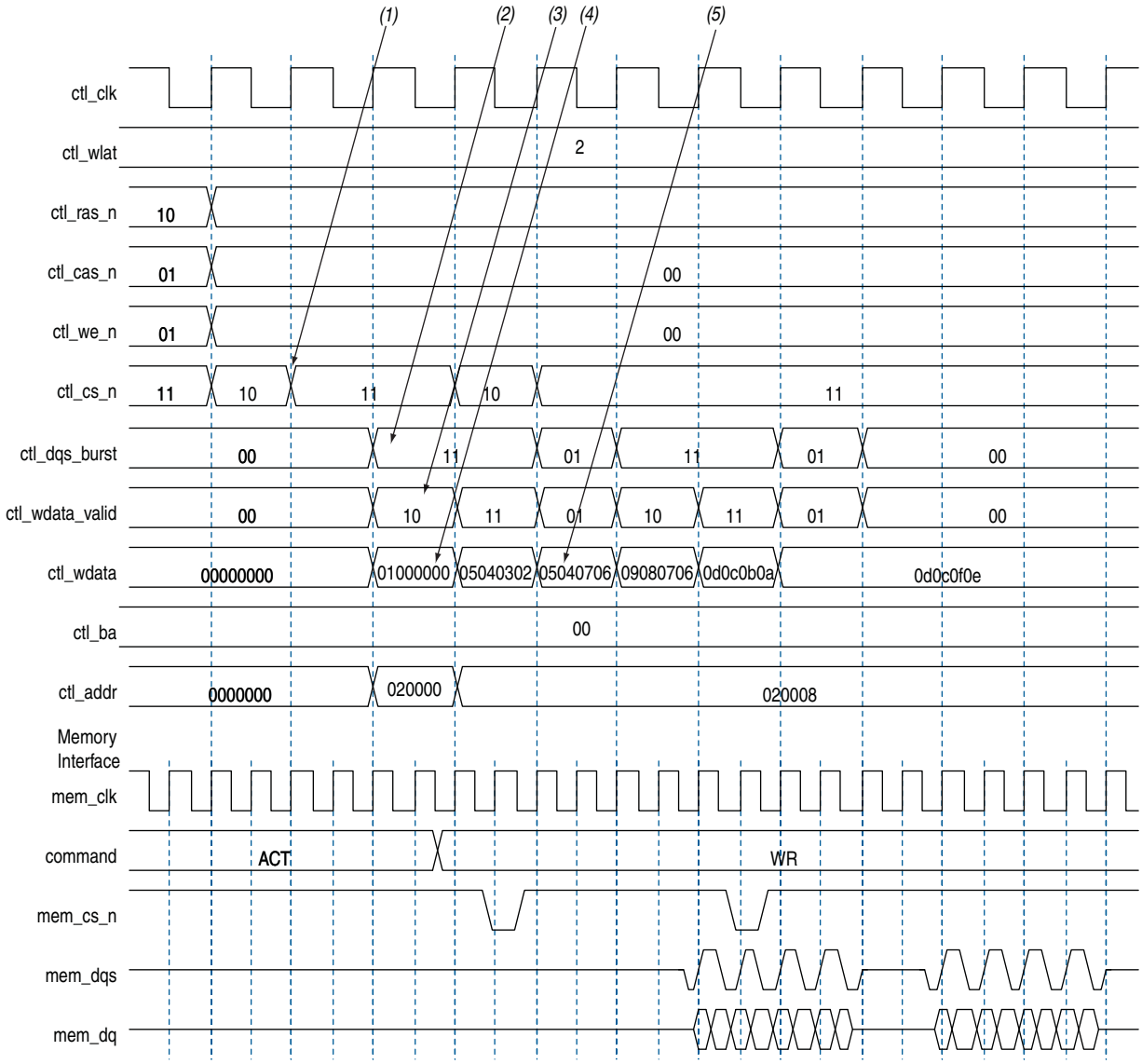
(2) AFI requires that `ctl_doing_rd` is driven for the duration of the read. In this example, it is driven to `11` for two half-rate `ctl_clks`, which equates to driving to `1`, for the four memory clock cycles of this four-beat burst.

(3) The `ctl_rdata_valid` returns 15 (`ctl_rlat`) controller clock (`ctl_clk`) cycles after `ctl_doing_rd` is asserted. Returned is when the `ctl_rdata_valid` signal is observed at the output of a register within the controller. A controller can use the `ctl_rlat` value to determine when to register to returned data, but this is unnecessary as the `ctl_rdata_valid` is provided for the controller to use as an enable when registering read data.

(4) Observe the alignment of returned read data with respect to data on the bus.

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

**Figure 5–21. Word-Unaligned Writes**




**Notes to Figure 5–21:**

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

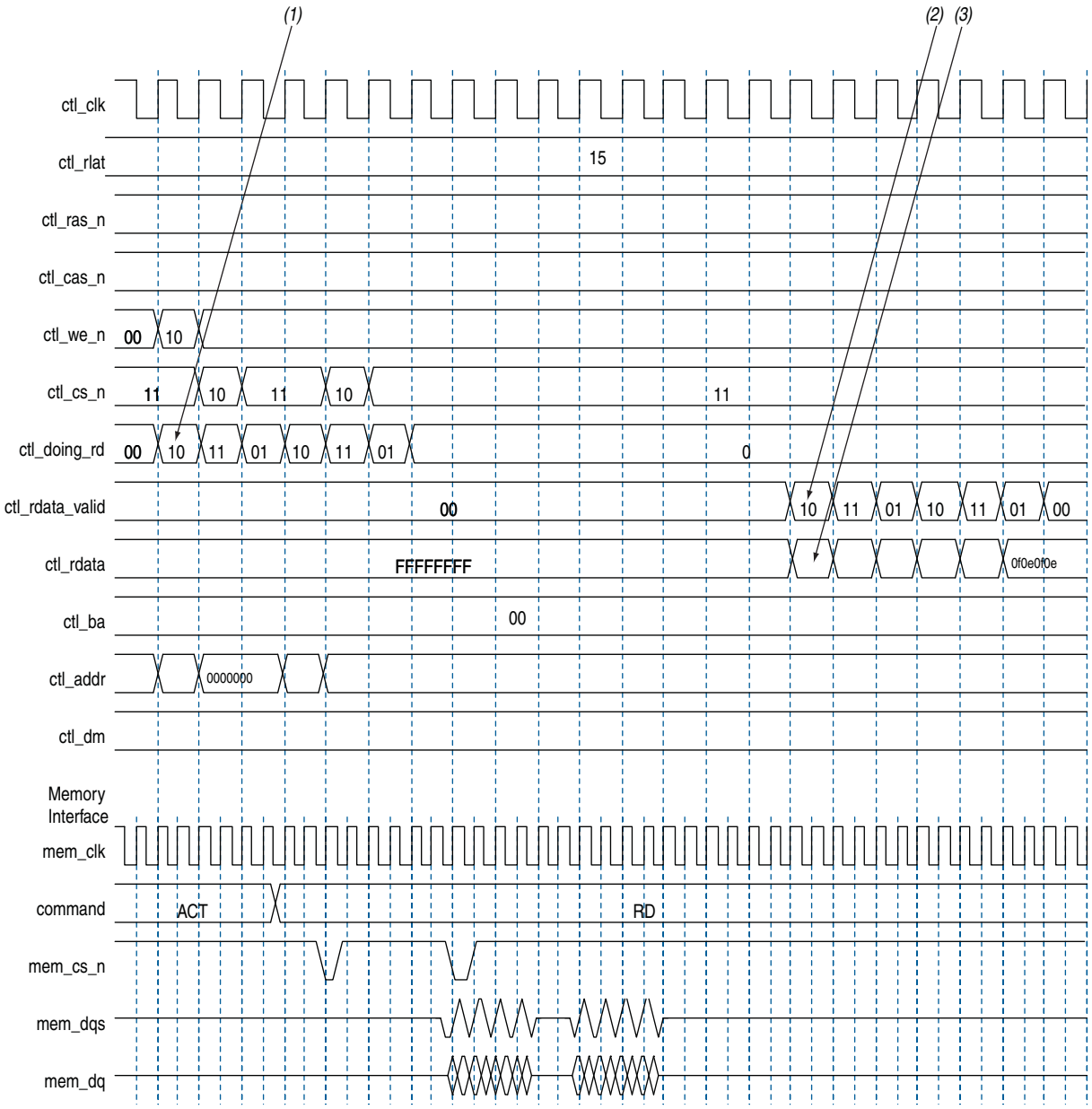
(2) As with word- aligned writes, `ctl_dqs_burst` is asserted one memory clock cycle before `ctl_wdata_valid`. You can see `ctl_dqs_burst` is 11 in the same cycle where `ctl_wdata_valid` is 10. The LSB of these two becomes the first value the signal takes in the `mem_clk` domain. You can see that `ctl_dqs_burst`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

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

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

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

**Figure 5–22. Word-Unaligned Reads**




**Notes to Figure 5–22:**

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

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

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

# Using a Custom Controller

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

## Preliminary Steps

Perform the following steps to generate the ALTMEMPHY megafunction:

1. If you are creating a custom DDR or DDR2 SDRAM controller, generate the Altera DDR or DDR2 SDRAM Controller with ALTMEMPHY IP targeting your chosen Altera memory devices.
2. Compile and verify the timing. This step is optional; refer to "Compiling and Simulating" on page 4–1.
3. If targeting a DDR or DDR2 SDRAM device, simulate the high-performance controller design.
4. Integrate the top-level ALTMEMPHY design with your controller. If you started with the high-performance controller, the PHY variation name is <*controller_name*>_phy.v/.vhd. Details about integrating your controller with Altera's ALTMEMPHY megafunction are described in the following sections.
5. Compile and simulate the whole interface to ensure that you are driving the PHY properly and that your commands are recognized by the memory device.

## Design Considerations

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

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

## Clocks and Resets

The ALTMEMPHY megafunction automatically generates a PLL instance, but you must still provide the reference clock input (`pll_ref_clk`) with a clock of the frequency that you specified in the MegaWizard Plug-In Manager. An active-low global reset input is also provided, which you can deassert asynchronously. The clock and reset management logic synchronizes this reset to the appropriate clock domains inside the ALTMEMPHY megafunction.

A clock output (half the memory clock frequency for a half-rate controller; the same as the memory clock for a full-rate controller) is provided and all inputs and outputs of the ALTMEMPHY megafunction are synchronous to this clock. For AFIs, this signal is called `ctl_clk`.

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

## Calibration Process Requirements

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

☞ You can ignore the following two warning and critical warning messages:

```
Warning: Timing Analysis for multiple chip select DDR/DDR2/DDR3-SDRAM
configurations is preliminary (memory interface has a chip select width
of 4)

Critical Warning: Read Capture and Write timing analyses may not be
valid due to violated timing model assumptions
```

## Other Local Interface Requirements

The memory burst length can be two, four, or eight for DDR SDRAM devices, and four or eight for DDR2 SDRAM devices. For a half-rate controller, the memory clock runs twice as fast as the clock provided to the local interface, so data buses on the local interface are four times as wide as the memory data bus. For a full-rate controller, the memory clock runs at the same speed as the clock provided to the local interface, so the data buses on the local interface are two times as wide as the memory data bus.

This section describes the DDR or DDR2 SDRAM high-performance controllers with the AFI.

## Address and Command Interfacing

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

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

## Handshake Mechanism Between Read Commands and Read Data

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

- Control of the postamble circuit

- Generation of ctl_rdata_valid
- Dynamic termination (Rt) control timing

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

The ALTMEMPHY provides a signal, ctl_rdata_valid, to indicate that the data on read data bus is valid. The width of this signal varies between half-rate and full-rate designs to support the option to indicate that the read data is not word aligned. Figure 5–23 and Figure 5–24 show these relationships.

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

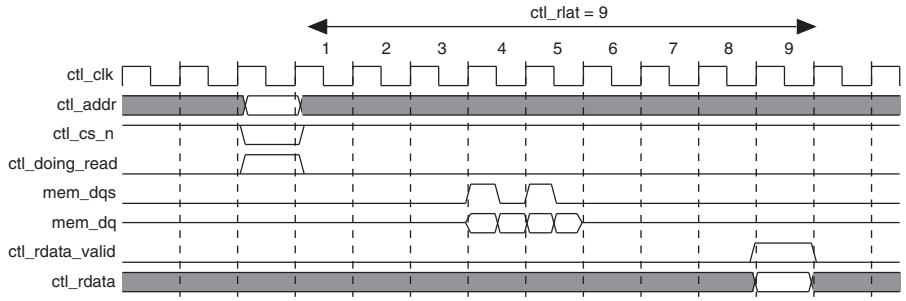
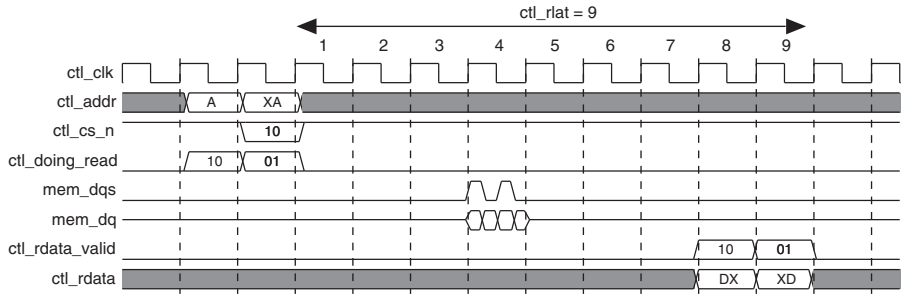



**Figure 5–24. Second Read Alignment—Half-Rate Design**



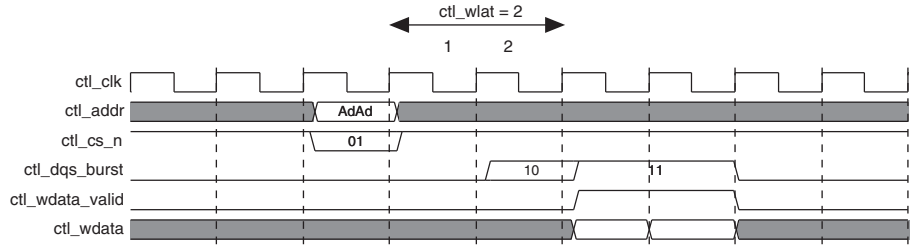

## Handshake Mechanism Between Write Commands and Write Data

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

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

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

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




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

## Partial Write Operations

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

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

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

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

## Using a Custom Controller with the TimeQuest Timing Analyzer

The Report DDR command in the TimeQuest Timing Analyzer may return the message Nothing to report when you use a custom controller.

To avoid this situation, you can edit your project's QSF file and set the value of the following line to **On**: set_global_assignment -name PROJECT_SHOW_ENTITY_NAME

Alternatively, you can perform the following procedure in the Quatrus II software GUI:

1. On the Assignments menu, click **Settings**.
2. In the **Category** list, click **Compilation Process Settings**.

3. In the **Compilation Process Settings** dialog box, click **More Settings**.
4. In the **More Compilation Process Settings** dialog box, under **Existing option settings**, set **Display entity name for node name** to **On**.

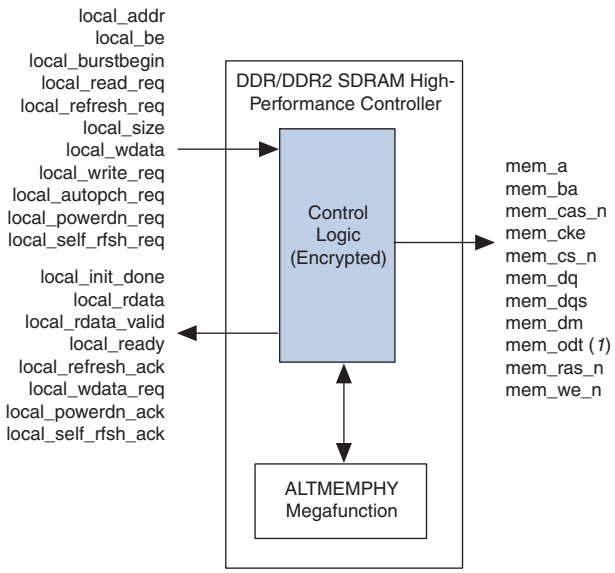
The high-performance controller (HPC) architecture instantiates encrypted control logic and the ALTMEMPHY megafunction. The controller accepts read and write requests from the user on its local interface, using either the Avalon-MM interface protocol or the native interface protocol. It converts these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

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

## Block Description

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

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

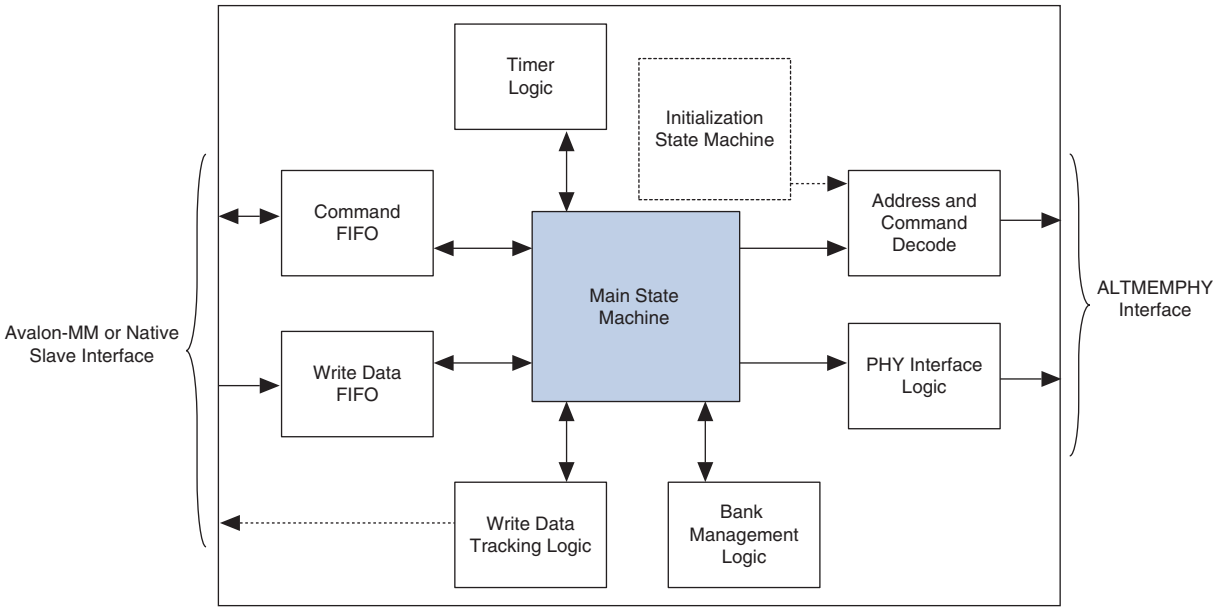



**Note to Figure 6–1:**

(1) For DDR2 SDRAM HPC only.

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

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




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

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

## Command FIFO Buffer

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

## Write Data FIFO Buffer

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

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

## Write Data Tracking Logic

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

## Main State Machine

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

## Bank Management Logic

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

## Timer Logic

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

## Initialization State Machine

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

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

## Address and Command Decode

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

## PHY Interface Logic

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

## ODT Generation Logic

The ODT generation logic (not shown in Figure 6–2) calculates when and for how long to enable the ODT outputs. It also decides which ODT bit to enable, based on the number of chip selects in the system.

- 1 DIMM (1 or 2 chip selects)

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

- 2 or more DIMMs

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

**Table 6–1. ODT**

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

## Low-Power Mode Logic

The low-power mode logic (not shown in Figure 6–2) monitors the `local_powerdn_req` and `local_self_rfsh_req` request signals. This logic also informs the user of the current low-power state via the `local_powerdn_ack` and `local_self_rfsh_ack` acknowledge signals.

HPC supports only precharge power-down mode and not active power-down mode.

## Control Logic

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

**Table 6–2. Bus Commands**

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

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

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

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

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

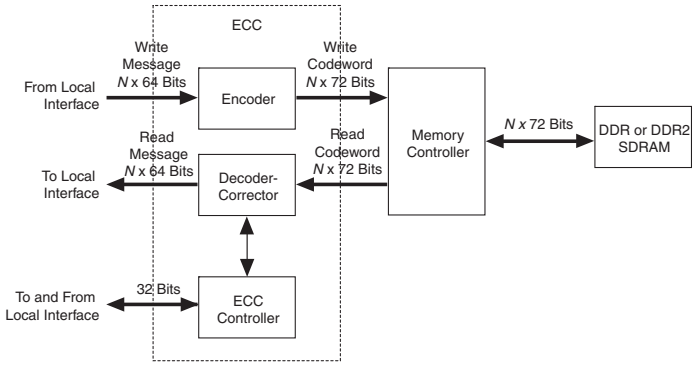
## Error Correction Coding (ECC)

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

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

Figure 6–3 shows the ECC block diagram.

**Figure 6–3. ECC Block Diagram**




The ECC comprises the following blocks:

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

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

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

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

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

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

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

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

## Interrupts

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

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

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

## Partial Writes

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

For partial writes, the ECC performs the following steps:

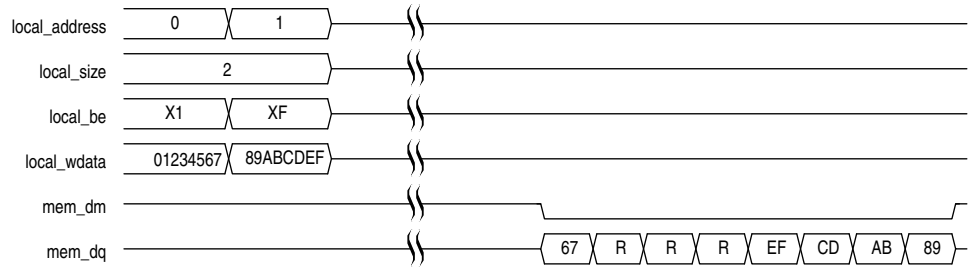
1. The ECC logic stalls further read or write commands from the Avalon-MM interface when it receives a partial write condition.
2. It simultaneously sends a self-generated read command, for the partial write address, to the memory controller.
3. Upon receiving the returned read data from the memory controller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

The following corner cases can occur:

- A single-bit error during the read phase of the read-modify-write process. In this case, the single-bit error is corrected first, the single-bit error counter is incremented and then a partial write is performed to this corrected decoded data word.
- A double-bit error during the read phase of the read-modify-write process. In this case, the double-bit error counter is incremented and an interrupt is sent through the Avalon-MM interface. The new write word is written to the memory location. A separate field in the interrupt status register highlights this condition.

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

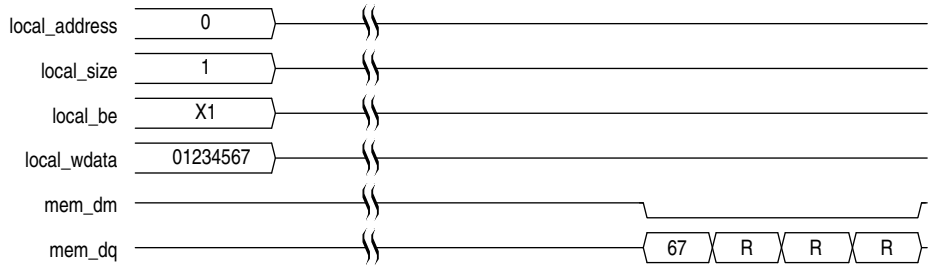
**Figure 6–4. Partial Write for HPC—Full Rate**




**Note to Figure 6–4:**

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

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




**Note to Figure 6–5:**

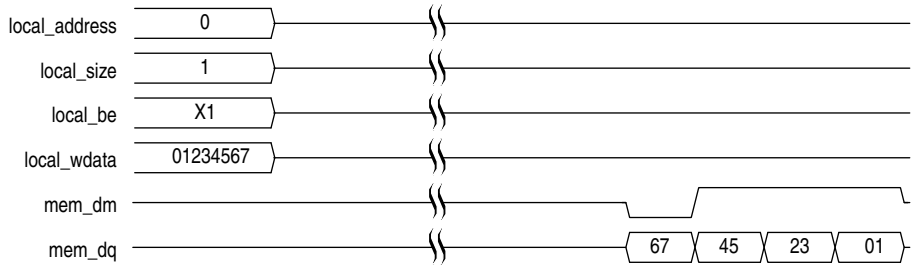
(1) R represents the internal read-back memory data during the read-modify-write process.

## Partial Bursts

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

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

**Figure 6–6. Partial Burst for HPC**



## ECC Latency

Using the ECC results in the following latency changes:

- Local Burst Length 1
- Local Burst Length 2

### Local Burst Length 1

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

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

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

**Table 6–3. Burst Lengths and Rates**

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

### Local Burst Length 2

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

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

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

## ECC Registers

Table 6–4 shows the ECC registers.

**Table 6–4. ECC Registers (Part 1 of 3)**

| Name | Address | Size (Bits) | Attribute | Default | Description |
|---|---|---|---|---|---|
| Control word specifications | 00 | 32 | R/W | 0000000F | This register contains all commands for the ECC functioning. |
| Maximum single-bit error counter threshold | 01 | 32 | R/W | 00000001 | The single-bit error counter increments (when a single-bit error occurs) until the maximum threshold, as defined by this register. When this threshold is crossed, the ECC generates an interrupt. |
| Maximum double-bit error counter threshold | 02 | 32 | R/W | 00000001 | The double-bit error counter increments (when a double-bit error occurs) until the maximum threshold, as defined by this register. When this threshold is crossed, the ECC generates an interrupt. |

**Table 6–4. ECC Registers (Part 2 of 3)**

| Name | Address | Size (Bits) | Attribute | Default | Description |
|---|---|---|---|---|---|
| Current single-bit error count | 03 | 32 | RO | 00000000 | The single-bit error counter increments (when a single-bit error occurs) until the maximum threshold. You can find the value of the count by reading this status register. |
| Current double-bit error count | 04 | 32 | RO | 00000000 | The double-bit error counter increments (when a double-bit error occurs) until the maximum threshold. You can find the value of the count by reading this status register. |
| Last or first single-bit error error address | 05 | 32 | RO | 00000000 | This status register stores the last single-bit error error address. It can be cleared using the control word clear. If bit 10 of the control word is set high, the first occurred address is stored. |
| Last or first double-bit error error address | 06 | 32 | RO | 00000000 | This status register stores the last double-bit error error address. It can be cleared using the control word clear. If bit 10 of the control word is set high, the first occurred address is stored. |
| Last single-bit error error data | 07 | 32 | RO | 00000000 | This status register stores the last single-bit error error data word. As the data word is an *M*th multiple of 64, the data word is stored in a 2*N*-deep, 32-bit wide FIFO buffer with the least significant 32-bit sub word stored first. It can be cleared individually by using the control word clear. |
| Last single-bit error syndrome | 08 | 32 | RO | 00000000 | This status register stores the last single-bit error syndrome, which specifies the location of the error bit on a 64-bit data word. As the data word is an *M*th multiple of 64, the syndrome is stored in a *N* deep, 8-bit wide FIFO buffer where each syndrome represents errors in every 64-bit part of the data word. The register gets updated with the correct syndrome depending on which part of the data word is shown on the last single-bit error error data register. It can be cleared individually by using the control word clear. |
| Last double-bit error error data | 09 | 32 | RO | 00000000 | This status register stores the last double-bit error error data word. As the data word is an *M*th multiple of 64, the data word is stored in a 2*N* deep, 32-bit wide FIFO buffer with the least significant 32-bit sub word stored first. It can be cleared individually by using the control word clear. |

**Table 6–4. ECC Registers (Part 3 of 3)**

| Name | Address | Size (Bits) | Attribute | Default | Description |
|---|---|---|---|---|---|
| Interrupt status register | 0A | 5 | RO | 00000000 | This status register stores the interrupt status in four fields (refer to Table 6–6). These status bits can be cleared by writing a 1 in the respective locations. |
| Interrupt mask register | 0B | 5 | WO | 00000001 | This register stores the interrupt mask in four fields (refer to Table 6–7). |
| Single-bit error location status register | 0C | 32 | R/W | 00000000 | This status register stores the occurrence of single-bit error for each 64-bit part of the data word in every bit (refer to Table 6–8). These status bits can be cleared by writing a 1 in the respective locations. |
| Double-bit error location status register | 0D | 32 | R/W | 00000000 | This status register stores the occurrence of double-bit error for each 64-bit part of the data word in every bit (refer to Table 6–9). These status bits can be cleared by writing a 1 in the respective locations. |

## ECC Register Bits

Table 6–5 shows the control word specification register.

**Table 6–5. Control Word Specification Register**

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

Table 6–6 shows the interrupt status register.

**Table 6–6. Interrupt Status Register**

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

Table 6–7 shows the interrupt mask register.

**Table 6–7. Interrupt Mask Register**

| Bit | Name | Description |
|---|---|---|
| 0 | Single-bit error | When 1, masks single-bit error. |
| 1 | Double-bit error | When 1, masks interrupt when double-bit error occurs during a normal or read-modify-write condition (partial write).<br>When 0, interrupt when double-bit error occurs during a normal or read-modify-write condition (partial write). |
| 2 | Maximum single-bit error | When 1, masks single-bit error maximum threshold exceeding condition. |
| 3 | Maximum double-bit error | When 1, masks double-bit error maximum threshold exceeding condition. |
| 4 | Double-bit error during read-modify-write | When 1, masks interrupt when double-bit error occurs during a read-modify-write condition (partial write).<br>When 0, interrupt when double-bit error occurs during a read-modify-write condition (partial write). |
| Others | Reserved | Reserved. |

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

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

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

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

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

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

# Example Top-Level File

The MegaWizard Plug-In Manager helps you create an example top-level file that shows you how to instantiate and connect the DDR or DDR2 SDRAM HPC. The example top-level file consists of the DDR or DDR2 SDRAM HPC, some driver logic to issue read and write requests to the controller, a PLL to create the necessary clocks, and a DLL (Stratix series only). The example top-level file is a working system that you can compile and use for both static timing checks and board tests.

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

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

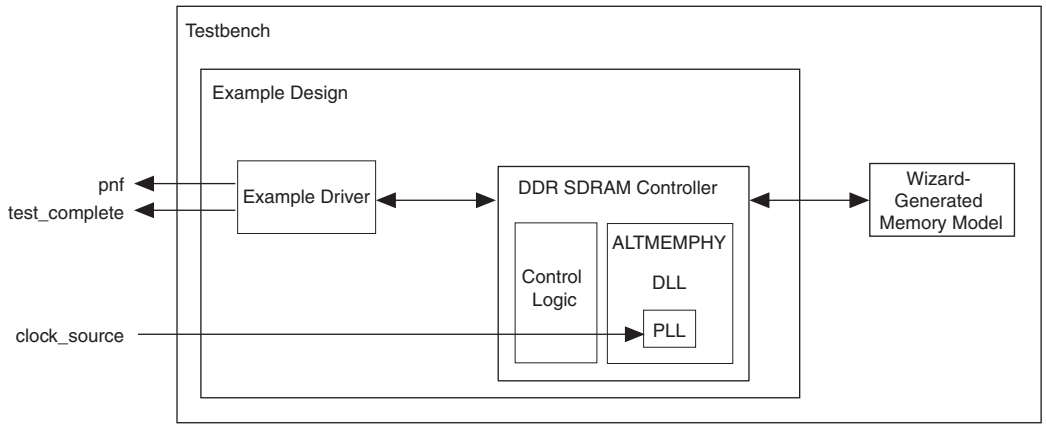



Table 6–10 describes the files that are associated with the example top-level file and the testbench.

**Table 6–10. Example Top-Level File and Testbench Files**

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

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

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

The full-array memory model (*<variation name>*_**mem model_full.v**) allocates memory for all addresses accessible by the DDR cores. This allocation makes it impossible to simulate large memory (more than 2K address spaces) designs, because simulators need more memory than what is available on a typical system.

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

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

## Example Driver

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

It performs the following tests and loops back the tests indefinitely:

- Sequential addressing writes and reads

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

- Incomplete write operation

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

- Byte enable/data mask pin operation

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

- Address pin operation

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

- Low-power mode operation

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

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

Table 6–11 shows the bit mapping for each test status.

**Table 6–11. Test Status[] Bit Mapping**

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

# Top-level Signals Description

Table 6–12 shows the clock and reset signals.

**Table 6–12. Clock and Reset Signals**

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

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

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

| Signal Name | Direction | Description |
|---|---|---|
| local_address[] | Input | Memory address at which the burst should start.<br>■ **Full rate controllers**<br>The width of this bus is sized using the following equation:<br>For one chip select: width = bank bits + row bits + column bits – 1<br>For multiple chip selects: width = chip bits + bank bits + row bits + column bits – 1<br>If the bank address is 2 bits wide, row is 13 bits wide and column is 10 bits wide, then the local address is 24 bits wide. To map local_address to bank, row and column address:<br>local_address[23:22] = bank address [1:0]<br>local_address[21:9] = row address [13:0]<br>local_address [8:0] = col_address[9:1]<br>The least significant bit (LSB) of the column address (multiples of four) on the memory side is ignored, because the local data width is twice that of the memory data bus width.<br>■ **Half rate controllers**<br>The width of this bus is sized using the following equation:<br>For one chip select: width = bank bits + row bits + column bits – 2<br>For multiple chip selects: width = chip bits + bank bits + row bits + column bits – 2<br>If the bank address is 2 bits wide, row is 13 bits wide and column is 10 bits wide, then the local address is 23 bits wide. To map local_address to bank, row and column address:<br>local_address is 23 bits wide<br>local_address[22:21] = bank address<br>local_address[20:8] = row address [13:0]<br>local_address [7:0] = col_address[9:2]<br>Two LSBs of the column address on the memory side are ignored, because the local data width is four times that of the memory data bus width.<br>You can get the information on address mapping from the *<variation_name>*_**example_top.v** or **vhd** file. |
| local_be[] | Input | Byte-enable signal, which you use to mask off individual bytes during writes. local_be is active high; mem_dm is active low.<br>To map local_wdata and local_be to mem_dq and mem_dm, consider a full-rate design with 32-bit local_wdata and 16-bit mem_dq.<br>Local_wdata = < 22334455 >< 667788AA >< BBCCDDEE ><br>Local_be = < 1100 >< 0110 >< 1010 ><br>These values map to:<br>Mem_dq = <4455><2233><88AA><6677><DDEE><BBCC><br>Mem_dm = <1 1 ><0 0 ><0 1 ><1 0 ><0 1 ><0 1 > |

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

| Signal Name | Direction | Description |
|---|---|---|
| local_burstbegin | Input | The Avalon burst begin strobe, which indicates the beginning of an Avalon burst. 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.<br>For write transactions, assert this signal at the beginning of each burst transfer and keep this signal high for one cycle per burst transfer, even if the slave has deasserted the local_ready signal. This signal is sampled at the rising edge of phy_clk when the local_write_req signal is asserted. After the slave deasserts the local_ready signal, the master keeps all the write request signals asserted until local_ready signal becomes high again.<br>For read transactions, assert this signal for one clock cycle when read request is asserted and the local_address from which the data should be read is given to the memory. After the slave deasserts local_ready (waitrequest_n in Avalon interface), the master keeps all the read request signals asserted until the local_ready signal becomes high again. |
| local_read_req | Input | Read request signal. You cannot assert the read request signal before the reset_phy_clk_n signal goes high. |
| local_refresh_req | Input | User-controlled refresh request. If **Enable User Auto-Refresh Controls** option is turned on, local_refresh_req becomes available and you are responsible for issuing sufficient refresh requests to meet the memory requirements. This option allows complete control over when refreshes are issued to the memory including ganging together multiple refresh commands. Refresh requests take priority over read and write requests unless they are already being processed. |
| local_size[] | Input | Controls the number of beats in the requested read or write access to memory, encoded as a binary number. The range of values depend on the memory burst length and whether you select full or half rate in the wizard.<br>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.<br>If you select a memory burst length 4 and full rate, the local burst length is 2 and you should set the local_size to either 1 or 2 for each read or write request. |
| local_wdata[] | Input | Write data bus. The width of local_wdata is twice that of the memory data bus for a full rate controller; four times the memory data bus for a half rate controller. |
| local_write_req | Input | Write request signal. You cannot assert the write request signal before the reset_phy_clk_n signal goes high. |
| local_autopch_req | Input | User control of precharge. If **Enable Auto-Precharge Control** is turned on, local_autopch_req becomes available and you can request the controller to issue an auto-precharge write or auto-precharge read command. These commands cause the memory to issue a precharge command to the current bank at the appropriate time without an explicit precharge command from the controller. This is particularly useful if you know the current read or write is the last one you intend to issue to the currently open row. The next time you need to use that bank, the access could be quicker as the controller does not need to precharge the bank before activating the row you wish to access. |

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

| Signal Name | Direction | Description |
|---|---|---|
| local_powerdn_req | Input | User control of the power-down feature. If **Enable Power Down Controls** option is enabled, you can request that the controller place the memory devices into a power-down state as soon as it can without violating the relevant timing parameters and responds by asserting the local_powerdn_ack signal. You can hold the memory in the power-down state by keeping this signal asserted. The controller brings the memory out of the power-down state to issue periodic auto-refresh commands to the memory at the appropriate interval if you hold it in the power-down state. You can release the memory from the power-down state at any time by deasserting the local_powerdn_ack signal once it has successfully brought the memory out of the power-down state. |
| local_self_rfsh_req | Input | User control of the self-refresh feature. If **Enable Self-Refresh Controls** option is enabled, you can request that the controller place the memory devices into a self-refresh state by asserting this signal. The controller places the memory in the self-refresh state as soon as it can without violating the relevant timing parameters and responds by asserting the local_self_rfsh_ack signal. You can hold the memory in the self-refresh state by keeping this signal asserted. You can release the memory from the self-refresh state at any time by deasserting the local_self_rfsh_req signal and the controller responds by deasserting the local__self_rfsh_ack signal once it has successfully brought the memory out of the self-refresh state. |
| phy_clk | Output | The system clock that the ALTMEMPHY megafunction provides to the user. All user inputs to and outputs from the DDR high-performance controller must be synchronous to this clock. |
| reset_phy_clk_n | Output | The reset signal that the ALTMEMPHY megafunction provides to the user. It is asserted asynchronously and deasserted synchronously to phy_clk clock domain. |
| aux_full_rate_clk | Output | An alternative clock that the ALTMEMPHY megafunction provides to the user. This clock always runs at the same frequency as the external memory interface. In half-rate mode, this clock is twice the frequency of the phy_clk and can be used whenever a 2x clock is required. In full-rate mode, this clock is driven by the same PLL output as the phy_clk signal. |
| aux_half_rate_clk | Output | An alternative clock that the ALTMEMPHY megafunction provides to the user. This clock always runs at half the frequency as the external memory interface. In full-rate mode, this clock is half the frequency of the phy_clk and can be used, for example to clock the user side of a half-rate bridge. In half-rate mode, this clock is driven by the same PLL output as the phy_clk signal. |
| dll_reference_clk | Output | Reference clock to feed to an externally instantiated DLL. |
| reset_request_n | Output | Reset request output that indicates when the PLL outputs are not locked. Use this signal as a reset request input to any system-level reset controller you may have. This signal is always low when the PLL is trying to lock, and so any reset logic using it is advised to detect a reset request on a falling edge rather than by level detection. |

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

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

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

**Table 6–14. DDR and DDR2 SDRAM Interface Signals (Part 1 of 2)**

| Signal Name | Direction | Description |
|---|---|---|
| mem_dq[] | Bidirectional | Memory data bus. This bus is half the width of the local read and write data busses. |
| mem_dqs[] | Bidirectional | Memory data strobe signal, which writes data into the 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. |

**Table 6–14. DDR and DDR2 SDRAM Interface Signals (Part 2 of 2)**

| Signal Name | Direction | Description |
|---|---|---|
| mem_a[] | Output | Memory address bus. |
| mem_ba[] | Output | Memory bank address bus. |
| mem_cas_n | Output | Memory column address strobe signal. |
| mem_cke[] | Output | Memory clock enable signals. |
| mem_cs_n[] | Output | Memory chip select signals. |
| mem_dm[] | Output | Memory data mask signal, which masks individual bytes during writes. |
| mem_odt[] | Output | Memory on-die termination control signal, for DDR2 SDRAM only. |
| mem_ras_n | Output | Memory row address strobe signal. |
| mem_we_n | Output | Memory write enable signal. |

**Note to Table 6–14:**

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

Table 6–15 shows the ECC logic signals.

**Table 6–15. ECC Logic Signals**

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

# 7. Functional Description—High-Performance Controller II

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

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

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

## Upgrading from HPC to HPC II

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

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

  For example, for Micron DDR3-800 datasheet, $t_{FAW}$=40 ns, $t_{RRD}$=10 ns, $t_{RTP}$=10 ns.
- HPC II replaces the port interface level for the AFI and Avalon interface without requiring any top-level change.

- The side-band signals differ slightly for HPC II. If you use these signals, you need to perform the following steps.
  - `local_refresh_req`

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

    The user-manual power signal is no longer supported in HPC II. Instead, you can select auto power-down on the **Controller Settings** tab in the MegaWizard Plug-In Manager, and specify the desired time-out (*n* cyles) after which the controller automatically powers down the memory.
- Because HPC II only supports a specific memory burst length, you must update the memory burst length to match the controller settings in Table 7–1.


You only can migrate your HPC designs to HPC II if you are using an Avalon-MM interface.

**Table 7–1. Burst Length Support**

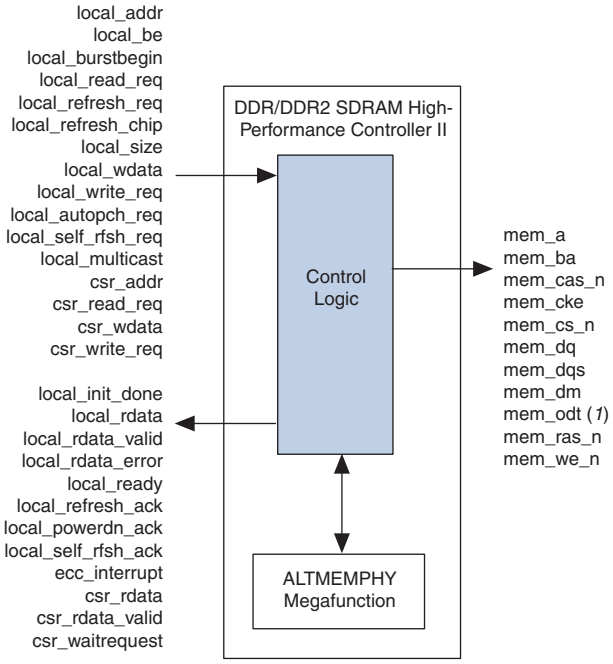
| Controller | HPC | HPC II |
|---|---|---|
| DDR | Burst length of 2 and 4 | Burst length of 4 in full-rate mode, and burst length of 8 in half-rate mode. |
| DDR2 | Burst length of 4 | |

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

# Block Description

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

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

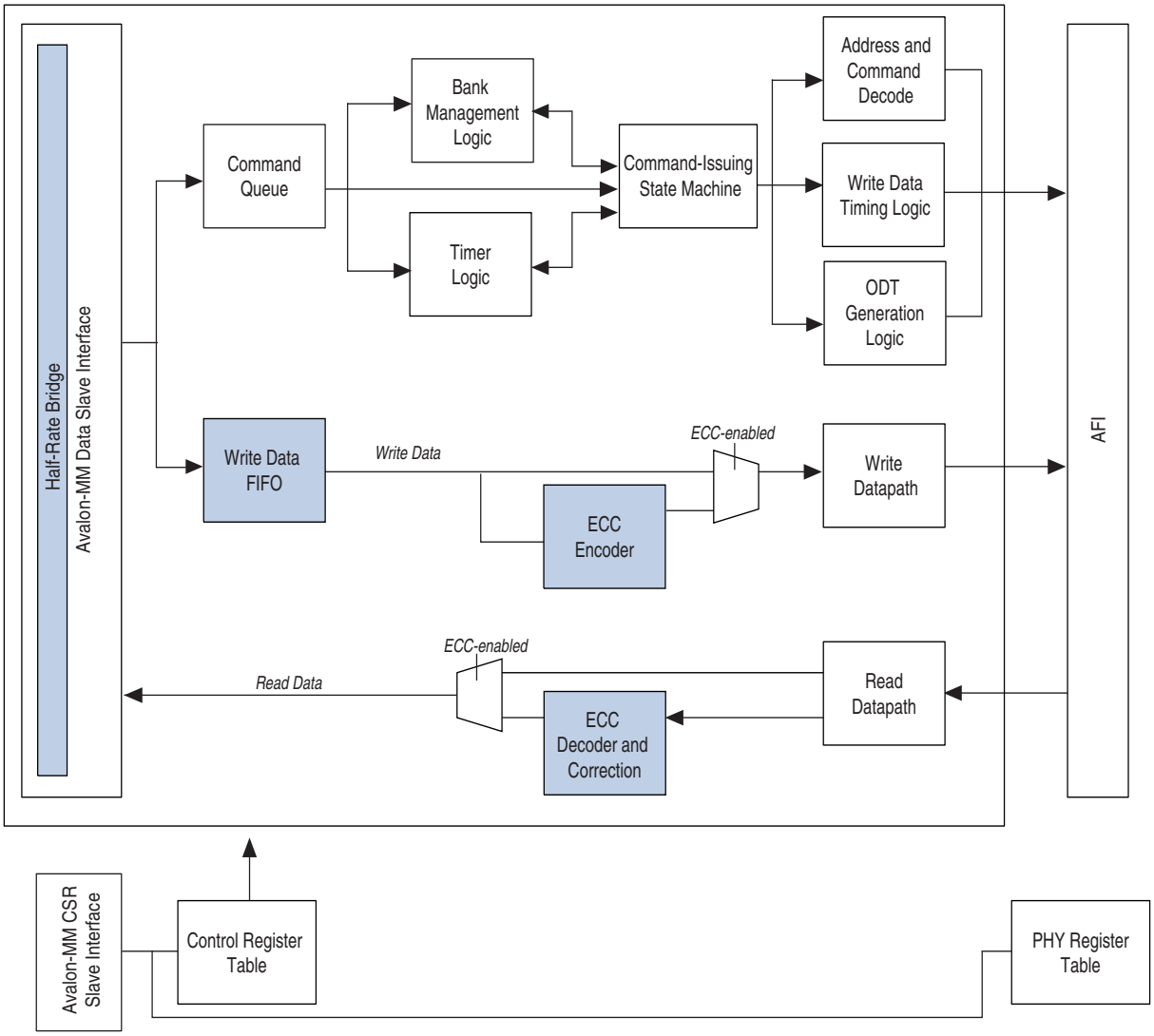



**Note to Figure 7–1:**

(1) For DDR2 SDRAM HPC II only.

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

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




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

## Avalon-MM Data Slave Interface

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

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

- For multiple chip selects:

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

- For a single chip select:

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

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

For every Avalon transaction, the number of read or write requests cannot exceed the maximum local burst count of 64. Altera recommends that you set this maximum burst count to match your system master's supported burst count.

HPC II uses word-address scheme with byte-enable instead of byte-address scheme.

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

## Write Data FIFO Buffer

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

## Command Queue

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

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

## Bank Management Logic

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

The bank management logic also includes a reduced bank tracking feature that reduces the controller's resource usage. This feature reduces the number of bank tracking blocks in the bank management logic. Each bank tracking block tracks the state of a memory bank. However, the controller only accesses a limited number of memory banks at a time. This limit is determined by the controller’s command queue look-ahead depth.

The reduced bank tracking feature provides the ability for the controller to dynamically allocate a memory bank to be tracked to one of the available bank tracking block. When a memory bank does not need to be tracked anymore, the bank tracking block is free to be allocated to another memory bank.

The reduced bank tracking feature benefits designs that have sixteen or more memory banks. For designs with fewer memory banks, this feature may not significantly reduce the controller's resource usage. Using this feature may have an impact on the memory bus efficiency, but it allows a trade-off between area and efficiency. The higher the number of banking tracking blocks, the better the efficiency.

## Timer Logic

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

## Command-Issuing State Machine

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

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

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

## Address and Command Decode Logic

When the main state machine issues a command to the memory, it asserts a set of internal signals. The address and command decode logic turns these signals into AFI specific commands and address.

This block generates the following signals:

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

## Write and Read Datapath, and Write Data Timing Logic

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

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

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

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

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

## ODT Generation Logic

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

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

**Table 7–2. ODT**

| DIMM | Chip Select per DIMM | Write or Read On | ODT Enabled (Write or Read) |
|---|---|---|---|
| 1 | Single chip select | `mem_cs[0]` | `mem_odt[0]` *(1)* |
| | Dual chip select | `mem_cs[0]` | `mem_odt[0]` *(1)* |
| | | `mem_cs[1]` | `mem_odt[1]` *(1)* |
| 2 | Single chip select | `mem_cs[0]` | `mem_odt[1]` |
| | | `mem_cs[1]` | `mem_odt[0]` |
| | Dual chip select | `mem_cs[0]` | `mem_odt[2]` |
| | | `mem_cs[1]` | `mem_odt[3]` |
| | | `mem_cs[2]` | `mem_odt[0]` |
| | | `mem_cs[3]` | `mem_odt[1]` |

**Note to Table 7–2:**

(1) The controller only drives the ODT signals during write operation.

## User-Controlled Side-Band Signals

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

### User-Refresh Commands

The user-refresh command enables the request to place the memory into refresh mode. The user-refresh control takes precedence over a read or write request. You can issue up to nine consecutive refresh commands to the selected memory chips. However, if you enable the multi-cast write feature, the user refresh commands are always issued to all chips.

### Multi-Cast Write

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

### Low-Power Mode Logic

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

#### User-Controlled Self-Refresh Logic

When you assert the `local_self_rfsh_req` signal, the controller completes all pending reads and writes before it places the memory into self-refresh mode. Once the controller places the memory into self-refresh mode, it responds by asserting the acknowledge signal, `local_self_rfsh_ack`. You can leave the memory in self-refresh mode for as long as you choose.

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

#### Automatic Power-Down with Programmable Time-Out

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



HPC II supports only precharge power-down mode and not active power-down mode.

## Configuration and Status Register (CSR) Interface

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

Refer to Table 7–10 through Table 7–24 on page 7–20 for detailed information about the register maps.

## Error Correction Coding (ECC)

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

- Hamming code ECC that encodes every 64 or 32 bits of data into 72 or 40 bits of codeword.
- A latency increase of one clock for both writes and reads.
- Detects and corrects all single-bit errors.
- Detects all double-bit errors.
- Counts the number of single-bit and double-bit errors.
- Accepts partial writes, which trigger a read-modify-write cycle, for memory devices with `dm` pins.
- Is able to inject single-bit and double-bit errors to trigger ECC correction for testing and debugging purposes.
- Generates an interrupt signal when an error occurs.

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

Enabling auto-correction allows the ECC logic to hold off all controller pending activities until the correction is completed. You can choose to disable auto-correction and schedule the correction manually when the controller is idle to ensure better system efficiency. To manually correct ECC errors, do the following:

1. When an interrupt occurs, read the `SBE_ERROR` register. When a single-bit error occurs, the `SBE_ERROR` register is equal to one.
2. Read out the `ERR_ADDR` register.

3. Correct the single-bit error by doing one of the following:
   - Issue a dummy write to the memory address stored in the ERR_ADDR register. A dummy write is a write request with the local_be signal zero, that triggers a partial write which is effectively a read-modify-write event. The partial write corrects the data at that address and writes it back.

   or

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

## Partial Writes

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

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

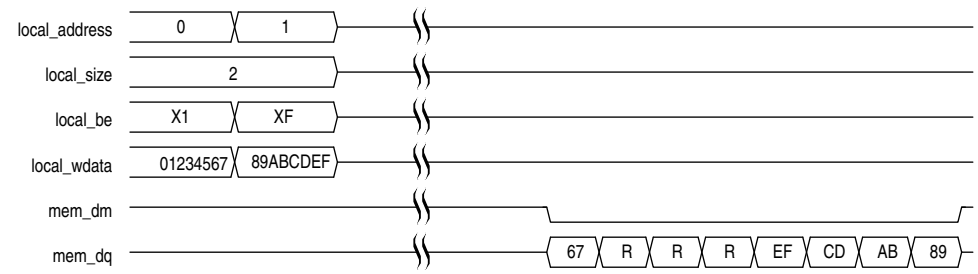
1. The ECC logic sends a read command to the partial write address.
2. Upon receiving a return data from the memory for the particular address, the ECC logic decodes the data, checks for errors, and then merges the corrected or correct dataword with the incoming information.
3. The ECC logic issues a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

The following corner cases can occur:

- A single-bit error during the read phase of the read-modify-write process. In this case, the single-bit error is corrected first, the single-bit error counter is incremented and then a partial write is performed to this corrected decoded data word.
- A double-bit error during the read phase of the read-modify-write process. In this case, the double-bit error counter is incremented and an interrupt is issued. A new write word is written back to the memory location. The ECC status register keeps track of the error information.

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

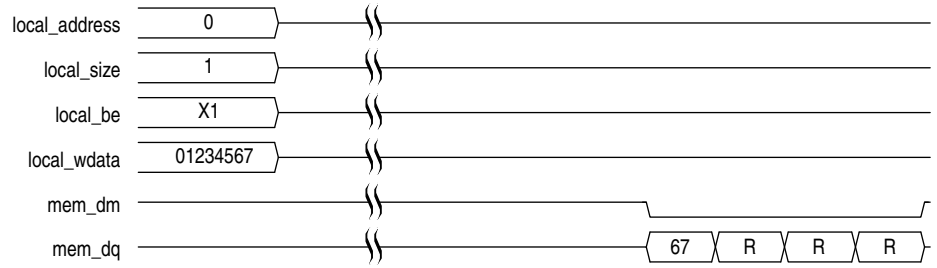
**Figure 7–3. Partial Write for HPC II—Full Rate**




**Note to Figure 7–3:**

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

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




**Note to Figure 7–4:**

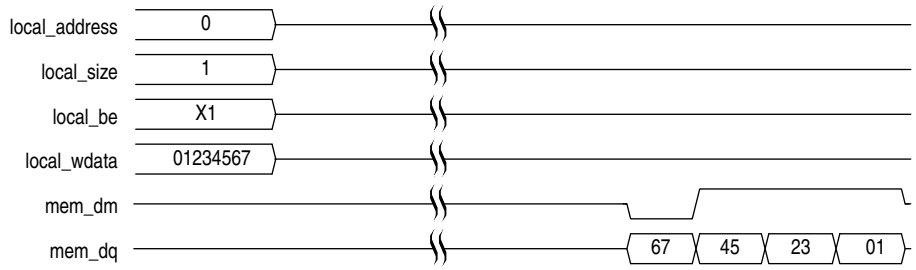
(1) R represents the internal read-back memory data during the read-modify-write process.

## Partial Bursts

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

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

**Figure 7–5. Partial Burst for HPC II**



# Example Top-Level File

The MegaWizard Plug-In Manager helps you create an example top-level file that shows you how to instantiate and connect the DDR or DDR2 SDRAM HPC II. The example top-level file consists of the DDR or DDR2 SDRAM HPC II, some driver logic to issue read and write requests to the controller, a PLL to create the necessary clocks, and a DLL (Stratix series only). The example top-level file is a working system that you can compile and use for both static timing checks and board tests.

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

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

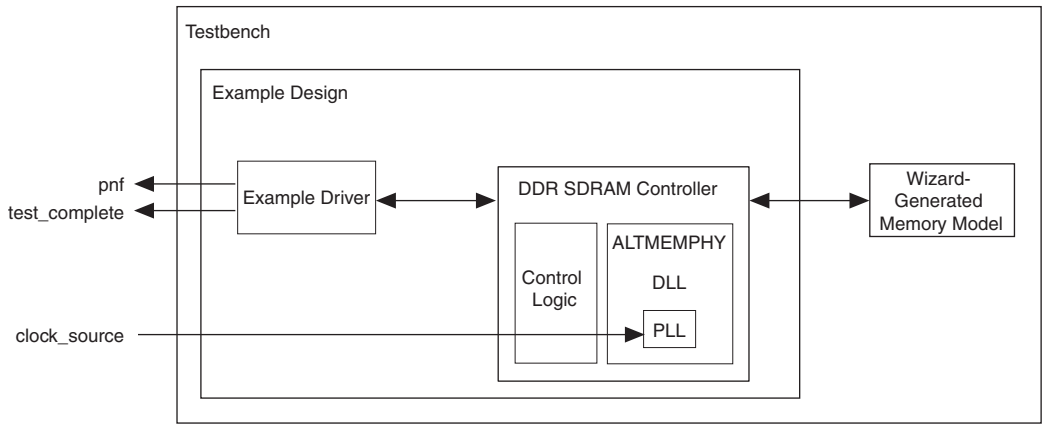



Table 7–3 describes the files that are associated with the example top-level file and the testbench.

**Table 7–3. Example Top-Level File and Testbench Files**

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

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

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

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

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

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

## Example Driver

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

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

- Sequential addressing writes and reads

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

- Incomplete write operation

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

- Byte enable/data mask pin operation

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

- Address pin operation

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

- Low-power mode operation

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

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

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

**Table 7–4. Test Status[] Bit Mapping**

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

# Top-level Signals Description

Table 7–5 shows the clock and reset signals.

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

| Name | Direction | Description |
|---|---|---|
| global_reset_n | Input | The asynchronous reset input to the controller. All other reset signals are derived from resynchronized versions of this signal. This signal holds the complete ALTMEMPHY megafunction, including the PLL, in reset while low. |
| pll_ref_clk | Input | The reference clock input to PLL. |
| phy_clk | Output | The system clock that the ALTMEMPHY megafunction provides to the user. All user inputs to and outputs from the DDR HPC II must be synchronous to this clock. |
| reset_phy_clk_n | Output | The reset signal that the ALTMEMPHY megafunction provides to the user. It is asserted asynchronously and deasserted synchronously to phy_clk clock domain. |

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

| Name | Direction | Description |
|---|---|---|
| aux_full_rate_clk | Output | An alternative clock that the ALTMEMPHY megafunction provides to the user. This clock always runs at the same frequency as the external memory interface. In half-rate mode, this clock is twice the frequency of the phy_clk and can be used whenever a 2x clock is required. In full-rate mode, this clock is driven by the same PLL output as the phy_clk signal. |
| aux_half_rate_clk | Output | An alternative clock that the ALTMEMPHY megafunction provides to the user. This clock always runs at half the frequency as the external memory interface. In full-rate mode, this clock is half the frequency of the phy_clk and can be used, for example to clock the user side of a half-rate bridge. In half-rate mode, or if the **Enable Half Rate Bridge** option is turned on, this clock is driven by the same PLL output that drives the phy_clk signal. |
| dll_reference_clk | Output | Reference clock to feed to an externally instantiated DLL. |
| reset_request_n | Output | Reset request output that indicates when the PLL outputs are not locked. Use this signal as a reset request input to any system-level reset controller you may have. This signal is always low when the PLL is trying to lock, and so any reset logic using it is advised to detect a reset request on a falling edge rather than by level detection. |
| soft_reset_n | Input | Edge detect reset input intended for SOPC Builder use or to be controlled by other system reset logic. It is asserted to cause a complete reset to the PHY, but not to the PLL used in the PHY. |
| oct_ctl_rs_value | Input | ALTMEMPHY signal that specifies the serial termination value. Should be connected to the ALT_OCT megafunction output seriesterminationcontrol. |
| oct_ctl_rt_value | Input | ALTMEMPHY signal that specifies the parallel termination value. Should be connected to the ALT_OCT megafunction output parallelterminationcontrol. |
| dqs_delay_ctrl_import | Input | Allows the use of DLL in another ALTMEMPHY instance in this ALTMEMPHY instance. Connect the export port on the ALTMEMPHY instance with a DLL to the import port on the other ALTMEMPHY instance. |

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

**Table 7–6. Local Interface Signals (Part 1 of 3)**

| Signal Name | Direction | Description |
|---|---|---|
| local_address[] | Input | Memory address at which the burst should start.<br>By default, the local address is mapped to the bank interleaving scheme. You can change the ordering via the **Local-to-Memory Address Mapping** option in the **Controller Settings** page.<br>The width of this bus is sized using the following equation:<br>■ **Full rate controllers**<br>The width of this bus is sized using the following equation:<br>For one chip select: width = row bits + bank bits + column bits – 1<br>For multiple chip selects: width = chip bits + row bits + bank bits + column bits – 1<br>If the bank address is 2 bits wide, row is 13 bits wide and column is 10 bits wide, then the local address is 24 bits wide. To map local_address to bank, row and column address:<br>local_address is 24 bits wide<br>local_address[23:11] = row address[12:0]<br>local_address[10:9] = bank address [1:0]<br>local_address [8:0] = column address[9:1]<br>The least significant bit (LSB) of the column address (multiples of four) on the memory side is ignored, because the local data width is twice that of the memory data bus width.<br>■ **Half rate controllers**<br>The width of this bus is sized using the following equation:<br>For one chip select: width = row bits + bank bits + column bits – 2<br>For multiple chip selects: width = chip bits + row bits + bank bits + column bits – 2<br>If the bank address is 2 bits wide, row is 13 bits wide and column is 10 bits wide, then the local address is 23 bits wide. To map local_address to bank, row and column address:<br>local_address is 23 bits wide<br>local_address[22:10] = row address[12:0]<br>local_address[9:8] = bank address [1:0]<br>local_address [7:0] = column address[9:2]<br>Two LSBs of the column address on the memory side are ignored, because the local data width is four times that of the memory data bus width. |
| local_be[] | Input | Byte enable signal, which you use to mask off individual bytes during writes. local_be is active high; mem_dm is active low.<br>To map local_wdata and local_be to mem_dq and mem_dm, consider a full-rate design with 32-bit local_wdata and 16-bit mem_dq.<br>Local_wdata = < 22334455 >< 667788AA >< BBCCDDEE ><br>Local_be = < 1100 >< 0110 >< 1010 ><br>These values map to:<br>Mem_dq = <4455><2233><88AA><6677><DDEE><BBCC><br>Mem_dm = <1 1 ><0 0 ><0 1 ><1 0 ><0 1 ><0 1 > |

**Table 7–6. Local Interface Signals (Part 2 of 3)**

| Signal Name | Direction | Description |
|---|---|---|
| local_burstbegin | Input | The Avalon burst begin strobe, which indicates the beginning of an Avalon burst. Unlike all other Avalon-MM signals, the burst begin signal does not stay asserted if local_ready is deasserted.<br>For write transactions, assert this signal at the beginning of each burst transfer and keep this signal high for one cycle per burst transfer, even if the slave has deasserted the local_ready signal. This signal is sampled at the rising edge of phy_clk when the local_write_req signal is asserted. After the slave deasserts the local_ready signal, the master keeps all the write request signals asserted until local_ready signal becomes high again.<br>For read transactions, assert this signal for one clock cycle when read request is asserted and the local_address from which the data should be read is given to the memory. After the slave deasserts local_ready (waitrequest_n in Avalon interface), the master keeps all the read request signals asserted until the local_ready signal becomes high again. |
| local_read_req | Input | Read request signal. You cannot assert the read request before the reset_phy_clk_n signal goes high. |
| local_refresh_req | Input | User-controlled refresh request. If **Enable User Auto-Refresh Controls** option is turned on, local_refresh_req becomes available and you are responsible for issuing sufficient refresh requests to meet the memory requirements. This option allows complete control over when refreshes are issued to the memory including grouping together multiple refresh commands. When a refresh request occurs, any commands that are currently executing in the memory device are allowed to finish, and then the refresh is executed before any other commands that are pending in the command queue. |
| local_refresh_chip | Input | Controls which chip to issue the user refresh to. This active high signal is used together with the local_refresh_req signal. This signal is as wide as the memory chip select. This signal asserts a high value to each bit that represents the refresh for the corresponding memory chip. For example: If the local_refresh_chip signal is assigned with a value of 4'b0101, the controller refreshes the memory chips 0 and 2, and memory chips 1 and 3 are not refreshed.<br>If you turn on the **Enable Multi-cast Write Control** option, this signal is ignored. |
| local_size[] | Input | Controls the number of beats in the requested read or write access to memory, encoded as a binary number. The range of supported Avalon burst lengths is 1 to 64. The width of this signal is derived based on the burst count specified in the **Local Maximum Burst Count** option. With the derived width, choose a value ranging from 1 to the local maximum burst count specified. |
| local_wdata[] | Input | Write data bus. The width of local_wdata is twice that of the memory data bus for a full rate controller; four times the memory data bus for a half rate controller. |
| local_write_req | Input | Write request signal. You cannot assert the write request signal before the reset_phy_clk_n signal goes high. |
| local_multicast | Input | In-band multi-cast write request signal. This active high signal is used together with the local_write_req signal. When this signal is asserted high, data is written to all the memory chips available. |

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

| Signal Name | Direction | Description |
|---|---|---|
| local_self_rfsh_req | Input | User control of the self-refresh feature. If **Enable Self-Refresh Controls** option is enabled, you can request that the controller place the memory devices into a self-refresh state by asserting this signal. The controller places the memory in the self-refresh state as soon as it can without violating the relevant timing parameters and responds by asserting the local_self_rfsh_ack signal. You can hold the memory in the self-refresh state by keeping this signal asserted. You can release the memory from the self-refresh state at any time by deasserting the local_self_rfsh_req signal and the controller responds by deasserting the local__self_rfsh_ack signal once it has successfully brought the memory out of the self-refresh state. |
| local_init_done | Output | When the memory initialization, training, and calibration are complete, the ALTMEMPHY sequencer asserts the ctrl_usr_mode_rdy signal to the memory controller, which then asserts this signal to indicate that the memory interface is ready to be used.<br>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.<br>This signal does not indicate that the calibration is successful. |
| local_rdata[] | Output | Read data bus. The width of local_rdata is twice that of the memory data bus for a full rate controller; four times the memory data bus for a half rate controller. |
| local_rdata_error | Output | Asserted if the current read data has an error. This signal is only available if the **Enable Error Detection and Correction Logic** option is turned on. This signal is asserted together with the local_rdata_valid signal.<br>If the controller encounters double-bit errors, no correction is made and the controller asserts this signal. |
| local_rdata_valid | Output | Read data valid signal. The local_rdata_valid signal indicates that valid data is present on the read data bus. |
| local_ready | Output | The local_ready signal indicates that the DDR or DDR2 SDRAM HPC II is ready to accept request signals. If local_ready is asserted in the clock cycle that a read or write request is asserted, that request has been accepted. The local_ready signal is deasserted to indicate that the DDR or DDR2 SDRAM HPC II cannot accept any more requests. The DDR or DDR2 SDRAM HPC II is able to buffer eight read or write requests. |
| local_refresh_ack | Output | Refresh request acknowledge, which is asserted for one clock cycle every time a refresh is issued. Even if the **Enable User Auto-Refresh Controls** option is not selected, local_refresh_ack still indicates to the local interface that the controller has just issued a refresh command. |
| local_self_rfsh_ack | Output | Self-refresh request acknowledge feature. This signal is asserted and deasserted in response to the local_self_rfsh_req signal from the user. |
| local_power_down_ack | Output | Auto power-down acknowledge signal. This signal is asserted for one clock cycle every time auto power-down is issued. |
| ecc_interrupt | Output | Interrupt signal from the ECC logic. This signal is asserted when the ECC feature is turned on, and an error is detected. This signal remains asserted until the user logic clears the error through the CSR interface. |

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

**Table 7–7. CSR Interface Signals**

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

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

**Table 7–8. DDR and DDR2 SDRAM Interface Signals**

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

**Note to Table 7–8:**

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

Table 7–9 shows the ALTMEMPHY Debug interface signals, which are located in <*variation_name*>_**phy.v/vhd** file.

**Table 7–9. ALTMEMPHY Debug Interface Signals**

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

# Register Maps Description

Table 7–10 through Table 7–24 show the register maps for the DDR and DDR2 SDRAM HPC II.

**Table 7–10. Register Map**

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

## ALTMEMPHY Register Map

The ALTMEMPHY register map allows you to control the memory components' mode register settings. To access the ALTMEMPHY register map, connect the ALTMEMPHY Debug interface signals in Table 7–9 using the Avalon-MM protocol.

After configuring the ALTMEMPHY register map, initialize a calibration request by setting bit 2 in the CSR register map address 0x100 (Table 7–13) for the mode register settings to take effect.

**Table 7–11. Address 0x005 Mode Register 0-1**

| Bit | Name | Default | Access | Description |
|---|---|---|---|---|
| 2:0 | Burst length | 8 | RO | This value is set to 4 for full-rate and 8 in half-rate for DDR or DDR2 SDRAM HPC II |
| 3 | BT | 0 | RO | This value is set to 0 because the DDR or DDR2 SDRAM HPC II only supports sequential bursts. |
| 6:4 | CAS latency | — | RW | CAS latency setting. The default value for these bits is set by the MegaWizard CAS Latency setting for your controller instance. You must set this value in the CSR interface register map 0x126 (Table 7–21) as well. |
| 7 | Reserved | 0 | — | Reserved for future use. |
| 8 | DLL | 0 | RW | Not used by the controller, but you can set and programm into the memory device mode register. |
| 11:9 | Write recovery | — | RW | Write recovery ($t_{WR}$) setting. The default value for these bits is set by the MegaWizard Write Recovery ($t_{WR}$) setting for your controller instance. You must set this value in CSR interface register map 0x126 (Table 7–21) as well. |
| 12 | PD | 0/1 | RO | This value is set to 0 because the DDR or DDR2 SDRAM HPC II only supports power-down fast exit mode. |
| 15:13 | Reserved | 0 | — | Reserved for future use. |
| 16 | DLL | 0 | RW | Not used by the controller, but you can set and program into the memory device mode register. |
| 17 | ODS | 0 | RW | |
| 18 | RTT | 0 | RW | |
| 21:19 | AL | — | RW | Additive latency setting. The default value for these bits is set by the MegaWizard Additive Latency setting for your controller instance. You must set this value in CSR interface register map 0x126 (Table 7–21) as well. |
| 22 | RTT | 0 | RW | Not used by the controller, but you can set and program into the memory device mode register. |
| 25:23 | RTT/WL/OCD | 0 | RW | |
| 26 | DQS# | 0 | RW | |
| 27 | TDQS/RDQS | 0 | RW | |
| 28 | QOFF | 0 | RW | |
| 31:29 | Reserved | 0 | — | Reserved for future use. |

**Table 7–12. Address 0x006 Mode Register 2-3**

| Bit | Name | Default | Access | Description |
|---|---|---|---|---|
| 2:0 | Reserved | 0 | — | Reserved for future use. |
| 5:3 | CWL | — | RW | CAS write latency setting. The default value for these bits is set by the MegaWizard CAS Write Latency setting for your controller instance. You must set this value in the CSR interface register map 0x126 (Table 7–21) as well. |
| 6 | ASR | 0 | RW | Not used by the controller, but you can set and program into the memory device mode register. |
| 7 | SRT/ET | 0 | RW | |
| 8 | Reserved | 0 | — | Reserved for future use. |
| 10:9 | RTT_WR | 0 | RW | Not used by the controller, but you can set and program into the memory device mode register. |
| 15:11 | Reserved | 0 | — | Reserved for future use. |
| 17:16 | MPR_RF | 0 | RW | Not used by the controller, but you can set and program into the memory device mode register. |
| 18 | MPR | 0 | RW | |
| 31:19 | Reserved | 0 | — | Reserved for future use. |

## Controller Register Map

The controller register map allows you to control the memory controller settings. To access the controller register map, connect the CSR interface signals in Table 7–7 using the Avalon-MM protocol.

**Table 7–13. Address 0x100 ALTMEMPHY Status and Control Register**

| Bit | Name | Default | Access | Description |
|---|---|---|---|---|
| 0 | CAL_SUCCESS | — | RO | This bit reports the value of the ALTMEMPHY `ctl_cal_success` output. Writing to this bit has no effect. |
| 1 | CAL_FAIL | — | RO | This bit reports the value of the ALTMEMPHY `ctl_cal_fail` output. Writing to this bit has no effect. |
| 2 | CAL_REQ | 0 | RW | Writing a 1 to this bit asserts the `ctl_cal_req` signal to the ALTMEMPHY megafunction. Writing a 0 to this bit deaasserts the signal, and the ALTMEMPHY megafunction will then initiate its calibration sequence.<br>CAUTION: You must not use this register during the ALTMEMPHY megafunction calibration. You must wait until the CAL_SUCCESS or CAL_FAIL register shows a value of 1. |
| 7:3 | Reserved | 0 | — | Reserved for future use. |
| 13:8 | CLOCK_OFF | 0 | RW | Writing a 1 to any of the bits in this register causes the appropriate `ctl_mem_clk_disable` signal to the ALTMEMPHY megafunction to be asserted, which disables the memory clock outputs. Writing a 0 to this register deasserts the signal and re-enables the memory clocks. The ALTMEMPHY megafunction supports up to 6 individual memory clocks; each bit represents each individual clock. |
| 30:14 | Reserved | 0 | — | Reserved for future use. |

**Table 7–14. Address 0x110 Controller Status and Configuration Register (Part 1 of 2)**

| Bit | Name | Default | Access | Description |
|---|---|---|---|---|
| 15:0 | AUTO_PD_CYCLES | 0x0 | RW | The number of idle clock cycles after which the controller should place the memory into power-down mode. The controller is considered to be idle if there are no commands in the command queue. Setting this register to 0 disables the auto power-down mode. The default value of this register depends on the values set during the generation of the design. |
| 16 | AUTO_PD_ACK | 1 | RO | This bit indicates that the memory is in power-down state. |

**Table 7–14. Address 0x110 Controller Status and Configuration Register (Part 2 of 2)**

| Bit | Name | Default | Access | Description |
|---|---|---|---|---|
| 17 | SELF_RFSH | 0 | RW | Setting this bit, or asserting the `local_self_rfsh` signal, causes the memory to go into self-refresh state. |
| 18 | SELF_RFSH-ACK | 0 | RO | This bit indicates that the memory is in self-refresh state. |
| 19 | Reserved | 0 | — | Reserved for future use. |
| 21:20 | ADDR_ORDER | 00 | RW | 00 - Chip, row, bank, column.<br>01 - Chip, bank, row, column.<br>10 - Reserved for future use.<br>11 - Reserved for future use. |
| 22 | REGDIMM | 0 | RW | Setting this bit to 1 enables REGDIMM support in controller. |
| 24:23 | CTRL_DRATE | 00 | RO | These bits represent controller date rate:<br>00 - Full rate.<br>01 - Half rate.<br>10 - Reserved for future use.<br>11 - Reserved for future use. |
| 30:24 | Reserved | 0 | — | Reserved for future use. |

**Table 7–15. Address 0x120 Memory Address Size Register 0**

| Bit | Name | Default | Access | Description |
|---|---|---|---|---|
| 7:0 | Column address width | — | RW | The number of column address bits for the memory devices in your memory interface. The range of legal values is 7-12. |
| 15:8 | Row address width | — | RW | The number of row address bits for the memory devices in your memory interface. The range of legal values is 12-16. |
| 19:16 | Bank address width | — | RW | The number of bank address bits for the memory devices in your memory interface. The range of legal values is 2-3. |
| 23:20 | Chip select address width | — | RW | The number of chip select address bits for the memory devices in your memory interface. The range of legal values is 0-2. If there is only one single chip select in the memory interface, set this bit to 0. |
| 31:24 | Reserved | 0 | — | Reserved for future use. |

**Table 7–16. Address 0x121 Memory Address Size Register 1**

| Bit | Name | Default | Access | Description |
|---|---|---|---|---|
| 31:0 | Data width representation (word) | — | RW | The number of DQS bits in the memory interface. This bit can be used to derive the width of the memory interface by multiplying this value by the number of DQ pins per DQS pin (typically 8). |

**Table 7–17. Address 0x122 Memory Address Size Register 2**

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

**Table 7–18. Address 0x123 Memory Timing Parameters Register 0**

| Bit | Name | Default | Access | Description |
|---|---|---|---|---|
| 3:0 | $t_{RCD}$ | — | RW | The activate to read or write a timing parameter. The range of legal values is 2-11 cycles. |
| 7:4 | $t_{RRD}$ | — | RW | The activate to activate a timing parameter. The range of legal values is 2-8 cycles. |
| 11:8 | $t_{RP}$ | — | RW | The precharge to activate a timing parameter. The range of legal values is 2-11 cycles. |
| 15:12 | $t_{MRD}$ | — | RW | The mode register load time parameter. This value is not used by the controller, as the controller derives the correct value from the memory type setting. |
| 23:16 | $t_{RAS}$ | — | RW | The activate to precharge a timing parameter. The range of legal values is 4-29 cycles. |
| 31:24 | $t_{RC}$ | — | RW | The activate to activate a timing parameter. The range of legal values is 8-40 cycles. |

**Table 7–19. Address 0x124 Memory Timing Parameters Register 1**

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

**Table 7–20. Address 0x125 Memory Timing Parameters Register 2**

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

**Table 7–21. Address 0x126 Memory Timing Parameters Register 3**

| Bit | Name | Default | Access | Description |
|---|---|---|---|---|
| 3:0 | CAS latency, $t_{CL}$ | — | RW | This value must be set to match the memory CAS latency. You must set this value in the 0x04 register map as well. |
| 7:4 | Additive latency, AL | — | RW | Additive latency setting. The default value for these bits is set in the **Memory additive CAS latency setting** in the **Preset Editor** dialog box.You must set this value in the 0x05 register map as well. |
| 11:8 | CAS write latency, CWL | — | RW | CAS write latency setting. You must set this value in the 0x06 register map as well. |
| 15:12 | Write recovery, $t_{WR}$ | — | RW | This value must be set to match the memory write recovery time ($t_{WR}$). You must set this value in the 0x04 register map as well. |
| 31:16 | Reserved | 0 | — | Reserved for future use. |

**Table 7–22. Address 0x130 ECC Control Register**

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

**Table 7–23. Address 0x131 ECC Status Register**

| Bit | Name | Default | Access | Description |
|---|---|---|---|---|
| 0 | SBE_ERROR | 0 | RO | Set to 1 when any single-bit errors occur. |
| 1 | DBE_ERROR | 0 | RO | Set to 1 when any double-bit errors occur. |
| 7:2 | Reserved | 0 | — | Reserved for future use. |
| 15:8 | SBE_COUNT | 0 | RO | Reports the number of single-bit errors that have occurred since the status register counters were last cleared. |
| 23:16 | DBE_COUNT | 0 | RO | Reports the number of double-bit errors that have occurred since the status register counters were last cleared. |
| 31:24 | Reserved | 0 | — | Reserved for future use. |

**Table 7–24. Address 0x132 ECC Error Address Register**

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

# 8. Latency

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

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

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

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

- Legacy DDR and DDR2 SDRAM controllers
- Half-rate HPC and HPC II
- Full-rate HPC and HPC II

The latency defined in this section uses the following assumptions:

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

The latency for the high-performance controllers comprises many different stages of the memory interface. Figure 8–1 on page 8–2 shows a typical memory interface read latency path showing read latency from the time a `local_read_req` signal assertion is detected by the controller up to data available to be read from the dual-port RAM (DPRAM) module.

**Figure 8–1. Typical Latency Path**

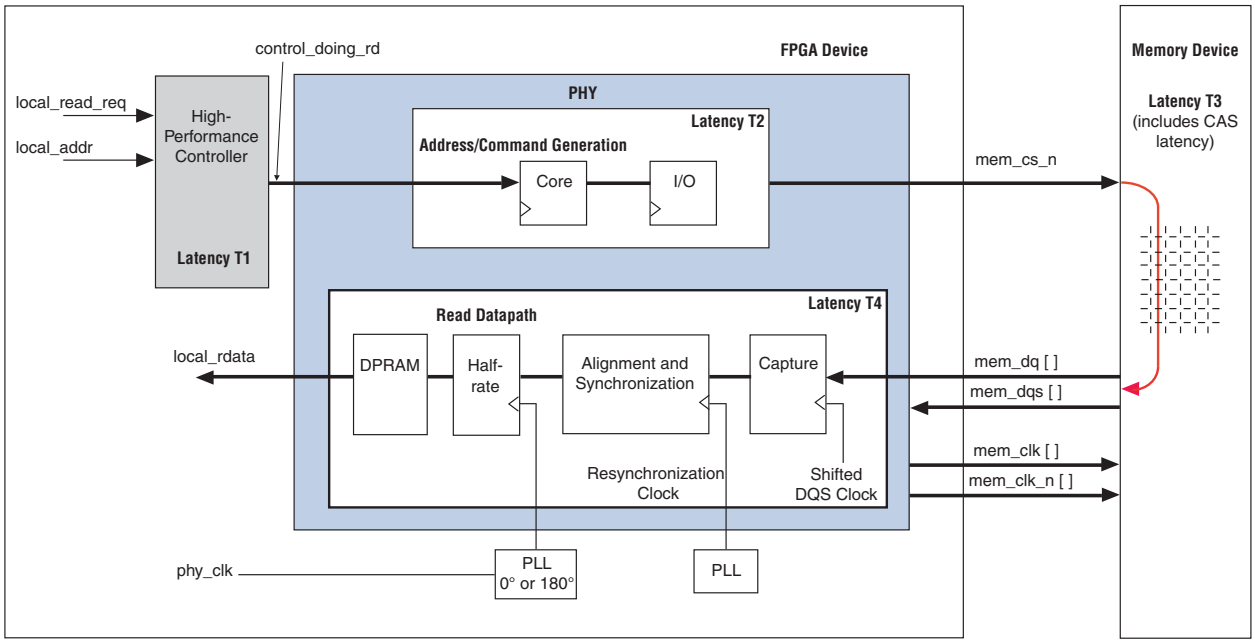



Table 8–1 shows the different stages that make up the whole read and write latency that Figure 8–1 shows.

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

| Latency Number | Latency Stage | Description |
|---|---|---|
| T1 | Controller | `local_read_req` or `local_write_req` signal assertion to `ddr_cs_n` signal assertion. |
| T2 | Command Output | `ddr_cs_n` signal assertion to `mem_cs_n` signal assertion. |
| T3 | CAS or WL | Read command to DQ data from the memory or write command to DQ data to the memory. |
| T4 | ALTMEMPHY read data input | Read data appearing on the local interface. |
| T2 + T3 | Write data latency | Write data appearing on the memory interface. |

From Figure 8–1, the read latency in the high-performance controllers is made up of four components:

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

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

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

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

Table 8–2 shows the minimum and maximum supported CAS latency for the DDR and DDR2 SDRAM high-performance controllers (HPC and HPC II).

**Table 8–2. Supported CAS Latency *(Note 1)***

| Device Family | Minimum Supported CAS Latency | | Maximum Supported CAS Latency | |
|---|---|---|---|---|
| | DDR | DDR2 | DDR | DDR2 |
| Arria GX | 3.0 | 3.0 | 3.0 | 6.0 |
| Arria II GX | 3.0 | 3.0 | 3.0 | 6.0 |
| Cyclone III | 2.0 | 3.0 | 3.0 | 6.0 |
| Cyclone IV | 2.0 | 3.0 | 3.0 | 6.0 |
| HardCopy III | 3.0 | 3.0 | 3.0 | 6.0 |
| HardCopy IV | 3.0 | 3.0 | 3.0 | 6.0 |
| Stratix II | 3.0 | 3.0 | 3.0 | 6.0 |
| Stratix III | 3.0 | 3.0 | 3.0 | 6.0 |
| Stratix IV | 3.0 | 3.0 | 3.0 | 6.0 |

**Note to Table 8–2:**

(1) The registered DIMMs, where supported, effectively introduce one extra cycle of CAS latency. For the registered DIMMs, you need to subtract 1.0 from the CAS figures to determine the minimum supported CAS latency, and add 1.0 to the CAS figures to determine the maximum supported CAS latency.

Table 8–3 through Table 8–6 show a typical latency that can be achieved in Arria GX, Arria II GX, Cyclone III, Cyclone IV, Stratix IV, Stratix III, Stratix II, and Stratix II GX devices. The exact latency for your memory controller depends on your precise configuration. You can obtain precise latency from simulation, but this figure can vary slightly in hardware because of the automatic calibration process.

The actual memory CAS and write latencies shown are halved in half-rate designs as the latency calculation is based on the local clock.

The read latency also depends on your board trace delay. The latency found in simulation can be different from that found in board testing as functional simulation does not take into account the board trace delays. For a given design on a given board, the latency may change by one clock cycle (for full-rate designs) or two clock cycles (for half-rate designs) upon resetting the board. Different boards could also show different latencies even with the same design.

The CAS and write latencies are different between DDR and DDR2 SDRAM interfaces. To calculate latencies for DDR SDRAM interfaces, use the numbers from DDR2 SDRAM listed below and replace the CAS and write latency with the DDR SDRAM values.

**Table 8–3. Typical Read Latency in HPC *(Note 1)*, *(2)***

| Device | Frequency (MHz) | Interface | Controller Latency *(3)* | Address and Command Latency | | CAS Latency *(4)* | Read Data Latency | | Total Read Latency *(5)* | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | FPGA | I/O | | FPGA | I/O | Local Clock Cycles | Time (ns) |
| Arria GX | 233 | Half-rate | 5 | 3 | 1 | 2 | 4.5 | 1 | 17 | 154 |
| | 167 | Full-rate | 4 | 2 | 1 | 4 | 5 | 1 | 17 | 108 |
| Arria II GX | 233 | Half-rate | 5 | 3 | 1 | 2.5 | 5.5 | 1 | 18 | 154 |
| | 167 | Full-rate | 4 | 2 | 1 | 4 | 6 | 1 | 18 | 114 |
| Cyclone III and Cyclone IV | 200 | Half-rate | 5 | 3 | 1 | 2 | 4.5 | 1 | 17 | 180 |
| | 167 | Full-rate | 4 | 2 | 1 | 4 | 5 | 1 | 17 | 108 |
| Stratix II and Stratix II GX | 333 | Half-rate | 5 | 3 | 1 | 2 | 4.5 | 1 | 17 | 108 |
| | 267 | Half-rate | 5 | 3 | 1 | 2 | 4.5 | 1 | 17 | 135 |
| | 200 | Full-rate | 4 | 2 | 1 | 4 | 5 | 1 | 17 | 90 |
| Stratix III and Stratix IV | 400 | Half-rate | 5 | 3 | 1 | 2.5 | 7.125 | 1.5 | 21 | 100 |
| | 267 | Full-rate | 4 | 2 | 1.5 | 4 | 7 | 1 | 20 | 71 |

**Notes to Table 8–3:**

(1) These are typical latency values using the assumptions listed in the beginning of the section. Your actual latency may be different than shown. Perform your own simulation for your actual latency.

(2) Numbers shown may have been rounded up to the nearest higher integer.

(3) The controller latency value is from the Altera high-performance controller.

(4) CAS latency is per memory device specification and is programmable in the MegaWizard Plug-In Manager.

(5) Total read latency is the sum of controller, address and command, CAS, and read data latencies.

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

| Device | Frequency (MHz) | Interface | Controller Latency *(3)* | Address and Command Latency | | CAS Latency *(4)* | Read Data Latency | | Total Read Latency *(5)* | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | FPGA | I/O | | FPGA | I/O | Local Clock Cycles | Time (ns) |
| Arria GX | 233 | Half-rate | 5 | 3 | 1 | 2 | 4.5 | 1 | 18 | 154 |
| | 167 | Full-rate | 5 | 2 | 1 | 4 | 5 | 1 | 19 | 114 |
| Arria II GX | 233 | Half-rate | 5 | 3 | 1 | 2.5 | 5.5 | 1 | 18 | 154 |
| | 167 | Full-rate | 5 | 2 | 1 | 4 | 6 | 1 | 20 | 120 |
| Cyclone III and Cyclone IV | 200 | Half-rate | 5 | 3 | 1 | 2 | 4.5 | 1 | 18 | 180 |
| | 167 | Full-rate | 5 | 2 | 1 | 4 | 5 | 1 | 19 | 114 |

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

| Device | Frequency (MHz) | Interface | Controller Latency *(3)* | Address and Command Latency | | CAS Latency *(4)* | Read Data Latency | | Total Read Latency *(5)* | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | FPGA | I/O | | FPGA | I/O | Local Clock Cycles | Time (ns) |
| Stratix II and Stratix II GX | 333 | Half-rate | 5 | 3 | 1 | 2 | 4.5 | 1 | 18 | 108 |
| | 267 | Half-rate | 5 | 3 | 1 | 2 | 4.5 | 1 | 18 | 135 |
| | 200 | Full-rate | 5 | 2 | 1 | 4 | 5 | 1 | 19 | 95 |
| Stratix III and Stratix IV | 400 | Half-rate | 5 | 3 | 1 | 2.5 | 7.125 | 1.5 | 20 | 100 |
| | 267 | Full-rate | 4 | 2 | 1.5 | 4 | 7 | 1 | 20 | 75 |

**Notes to Table 8–3:**

(1) These are typical latency values using the assumptions listed in the beginning of the section. Your actual latency may be different than shown. Perform your own simulation for your actual latency.

(2) Numbers shown may have been rounded up to the nearest higher integer.

(3) The controller latency value is from the Altera high-performance controller.

(4) CAS latency is per memory device specification and is programmable in the MegaWizard Plug-In Manager.

(5) Total read latency is the sum of controller, address and command, CAS, and read data latencies.

**Table 8–5. Typical Write Latency in HPC *(Note 1)*, *(2)***

| Device | Frequency (MHz) | Interface | Controller Latency *(3)* | Address and Command Latency | | Memory Write Latency *(4)* | Total Write Latency *(5)* | |
|---|---|---|---|---|---|---|---|---|
| | | | | FPGA | I/O | | Local Clock Cycles | Time (ns) |
| Arria GX | 233 | Half-rate | 5 | 3 | 1 | 1.5 | 12 | 103 |
| | 167 | Full-rate | 4 | 2 | 1 | 3 | 11 | 66 |
| Arria II GX | 233 | Half-rate | 5 | 3 | 1 | 2.5 | 12 | 103 |
| | 167 | Full-rate | 4 | 2 | 1 | 4 | 11 | 66 |
| Cyclone III and Cyclone IV | 200 | Half-rate | 5 | 3 | 1 | 1.5 | 12 | 120 |
| | 167 | Full-rate | 4 | 2 | 1 | 3 | 11 | 66 |
| Stratix II and Stratix II GX | 333 | Half-rate | 5 | 3 | 1 | 1.5 | 12 | 72 |
| | 267 | Half-rate | 5 | 3 | 1 | 1.5 | 12 | 90 |
| | 200 | Full-rate | 4 | 2 | 1 | 3 | 11 | 55 |
| Stratix III and Stratix IV | 400 | Half-rate | 5 | 3 | 1 | 2 | 12 | 60 |
| | 267 | Full-rate | 4 | 2 | 1.5 | 3 | 12 | 44 |

**Notes to Table 8–5:**

(1) These are typical latency values using the assumptions listed in the beginning of the section. Your actual latency may be different than shown. Perform your own simulation for your actual latency.

(2) Numbers shown may have been rounded up to the nearest higher integer.

(3) The controller latency value is from the Altera high-performance controller.

(4) Memory write latency is per memory device specification. The latency from when you provide the command to write to when you need to provide data at the memory device.

(5) Total write latency is the sum of controller, address and command, and memory write latencies.

**Table 8–6. Typical Write Latency in HPC II *(Note 1)*, *(2)* (Part 1 of 2)**

| Device | Frequency (MHz) | Interface | Controller Latency *(3)* | Address and Command Latency | | Memory Write Latency *(4)* | Total Write Latency *(5)* | |
|---|---|---|---|---|---|---|---|---|
| | | | | FPGA | I/O | | Local Clock Cycles | Time (ns) |
| Arria GX | 233 | Half-rate | 5 | 3 | 1 | 1.5 | 12 | 103 |
| | 167 | Full-rate | 5 | 2 | 1 | 3 | 12 | 72 |
| Arria II GX | 233 | Half-rate | 5 | 3 | 1 | 2.5 | 12 | 103 |
| | 167 | Full-rate | 5 | 2 | 1 | 4 | 12 | 72 |
| Cyclone III and Cyclone IV | 200 | Half-rate | 5 | 3 | 1 | 1.5 | 12 | 120 |
| | 167 | Full-rate | 5 | 2 | 1 | 3 | 12 | 72 |
| Stratix II and Stratix II GX | 333 | Half-rate | 5 | 3 | 1 | 1.5 | 12 | 72 |
| | 267 | Half-rate | 5 | 3 | 1 | 1.5 | 12 | 90 |
| | 200 | Full-rate | 5 | 2 | 1 | 3 | 12 | 60 |

**Table 8–6. Typical Write Latency in HPC II *(Note 1)*, *(2)* (Part 2 of 2)**

| Device | Frequency (MHz) | Interface | Controller Latency *(3)* | Address and Command Latency | | Memory Write Latency *(4)* | Total Write Latency *(5)* | |
|---|---|---|---|---|---|---|---|---|
| | | | | FPGA | I/O | | Local Clock Cycles | Time (ns) |
| Stratix III and Stratix IV | 400 | Half-rate | 5 | 3 | 1 | 2 | 12 | 60 |
| | 267 | Full-rate | 5 | 2 | 1.5 | 3 | 13 | 49 |

**Notes to Table 8–5:**

(1) These are typical latency values using the assumptions listed in the beginning of the section. Your actual latency may be different than shown. Perform your own simulation for your actual latency.

(2) Numbers shown may have been rounded up to the nearest higher integer.

(3) The controller latency value is from the Altera high-performance controller.

(4) Memory write latency is per memory device specification. The latency from when you provide the command to write to when you need to provide data at the memory device.

(5) Total write latency is the sum of controller, address and command, and memory write latencies.

To see the latency incurred in the IOE for both read and write paths for ALTMEMPHY variations in Stratix IV and Stratix III devices refer to the IOE figures in the *External Memory Interfaces in Stratix III Devices* chapter of the *Stratix III Device Handbook* and the *External Memory Interfaces in Stratix IV Devices* chapter of the *Stratix IV Device Handbook*.

# 9. Timing Diagrams

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

## DDR and DDR2 High-Performance Controllers

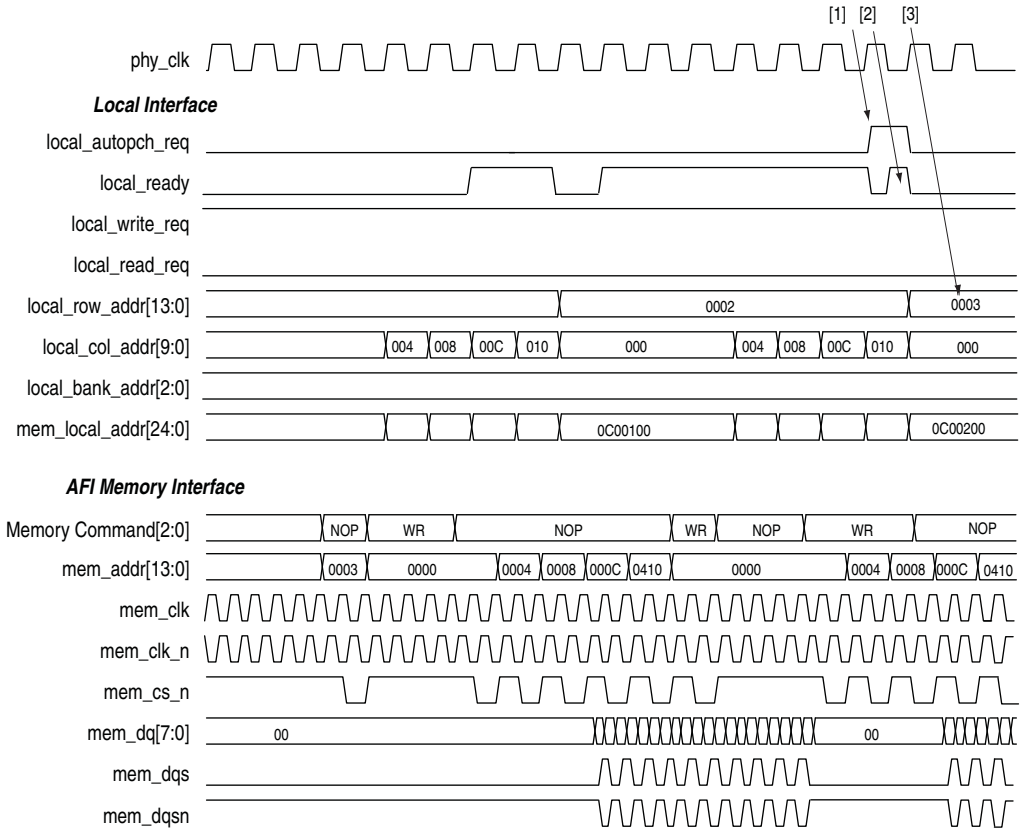
This section discusses the following timing diagrams for HPC in AFI mode:

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

## Auto-Precharge

The auto-precharge read and auto-precharge write commands allow you to indicate to the memory device that this read or write command is the last access to the currently open row. The memory device automatically closes (auto-precharges) the page it is currently accessing so that the next access to the same bank is quicker. This command is particularly useful for applications that require fast random accesses.

**Figure 9–1. Auto-Precharge Operation for HPC**




**Notes to Figure 9–1:**

(1) The auto-precharge request goes high.

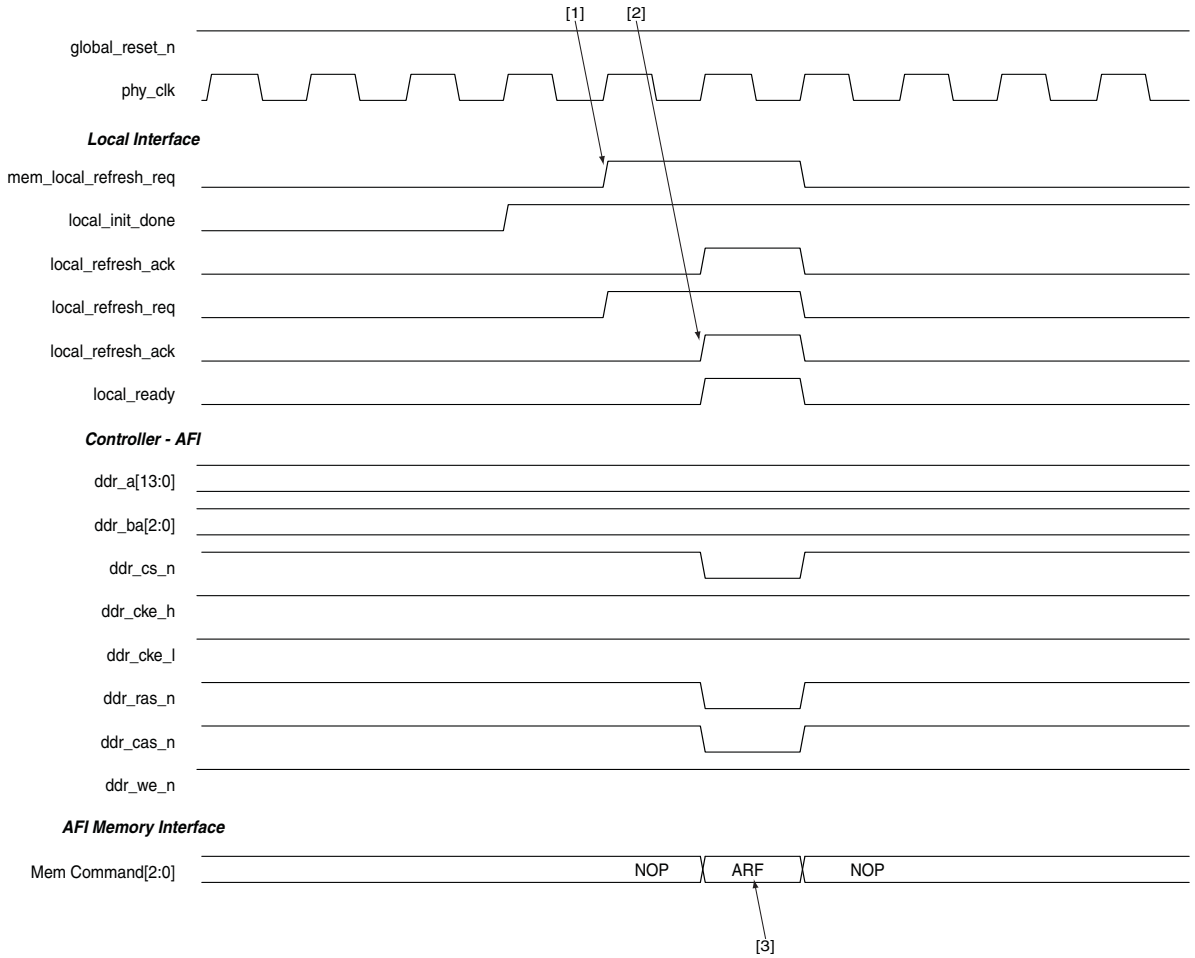
(2) The `local_ready` signal is asserted and remains high until the auto-precharge request goes low.

(3) A new row address begins.

## User Refresh

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

**Figure 9–2. User-Refresh Operation for HPC**

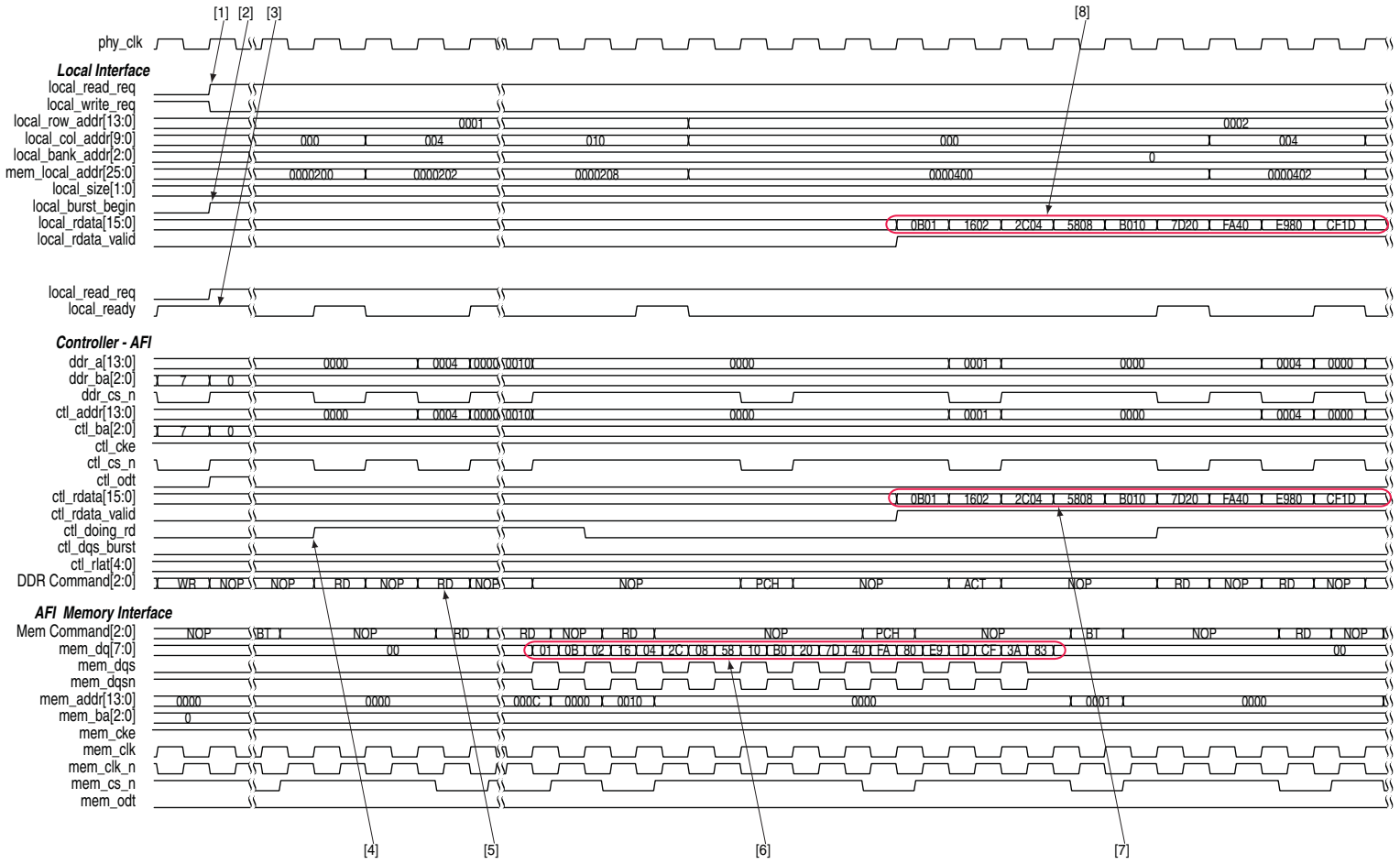



**Notes to Figure 9–2:**

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

# Full-Rate Read

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

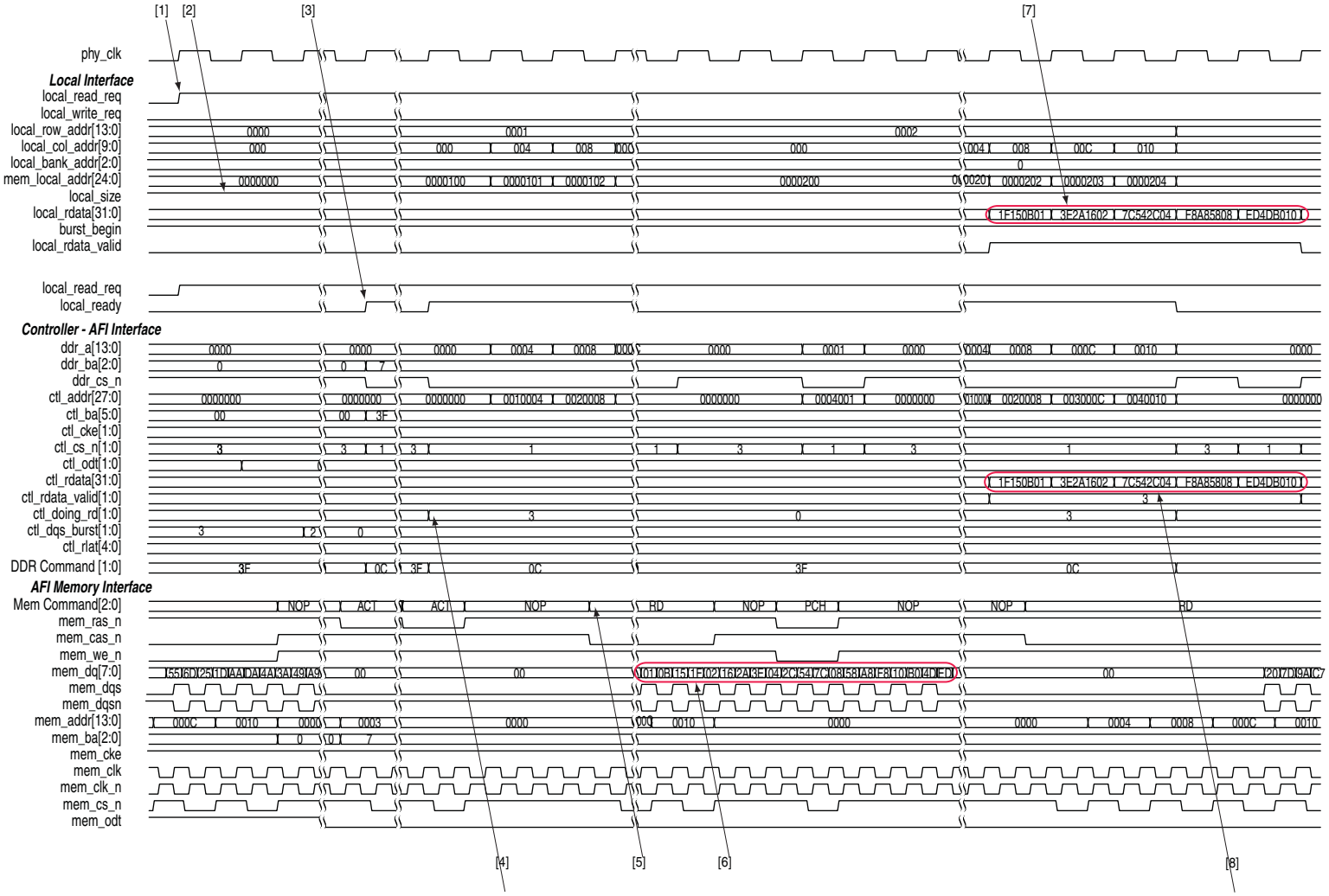


The following sequence corresponds with the numbered items in Figure 9–3:

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

# Half-Rate Read

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

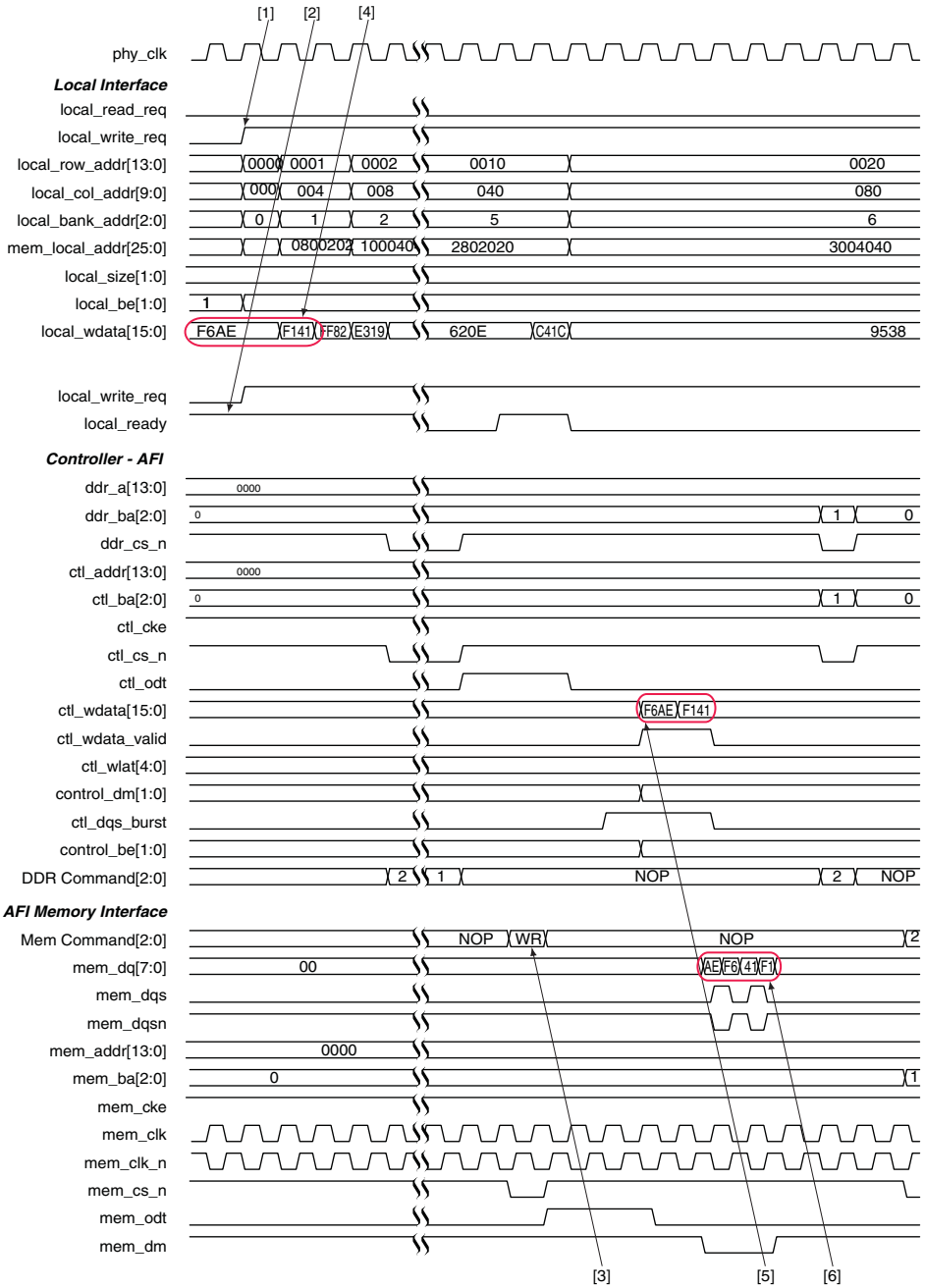


The following sequence corresponds with the numbered items in Figure 9–4:

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

## Full-Rate Write

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



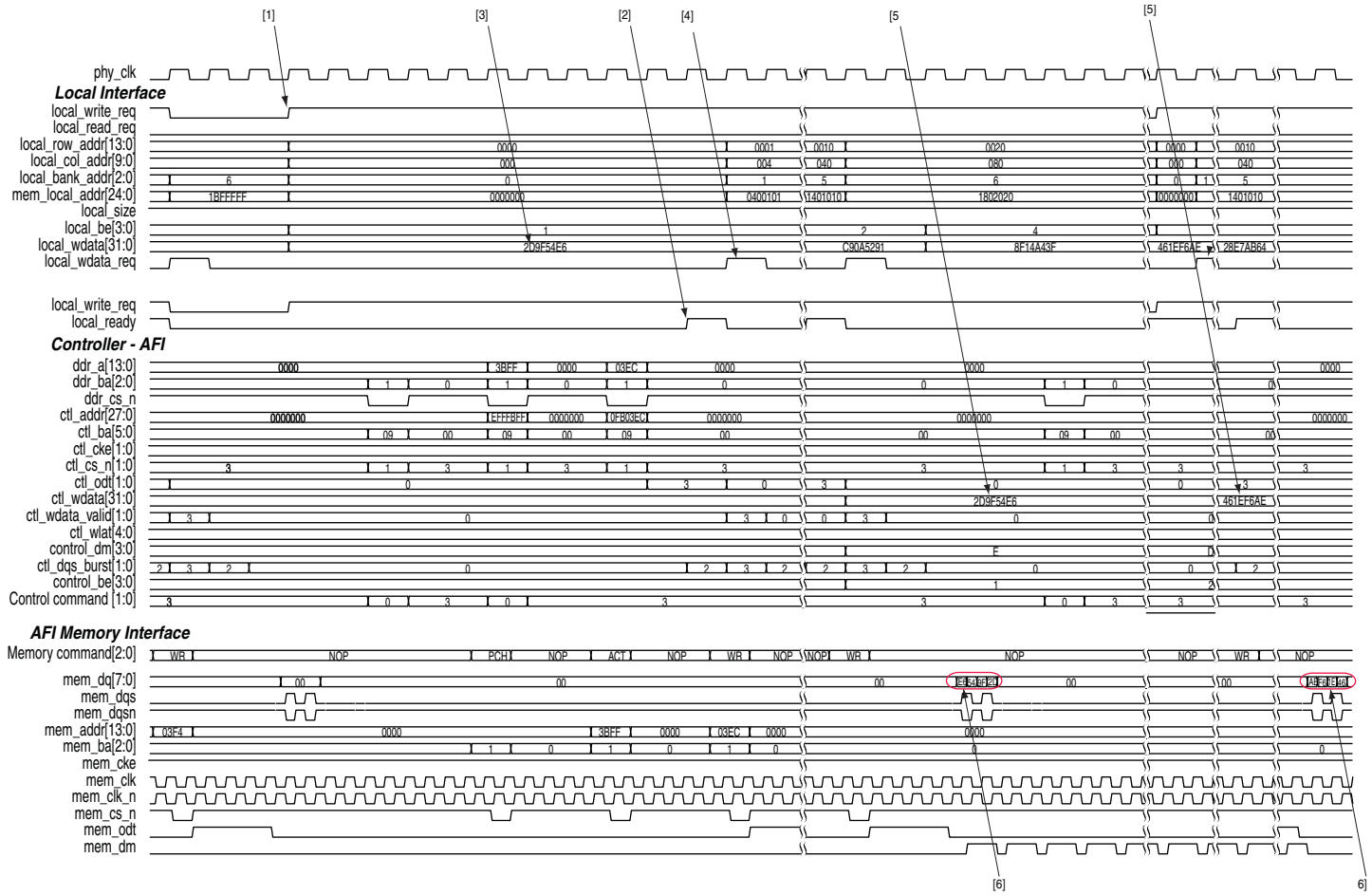

The following sequence corresponds with the numbered items in Figure 9–5:

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

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

## Half Rate Write

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

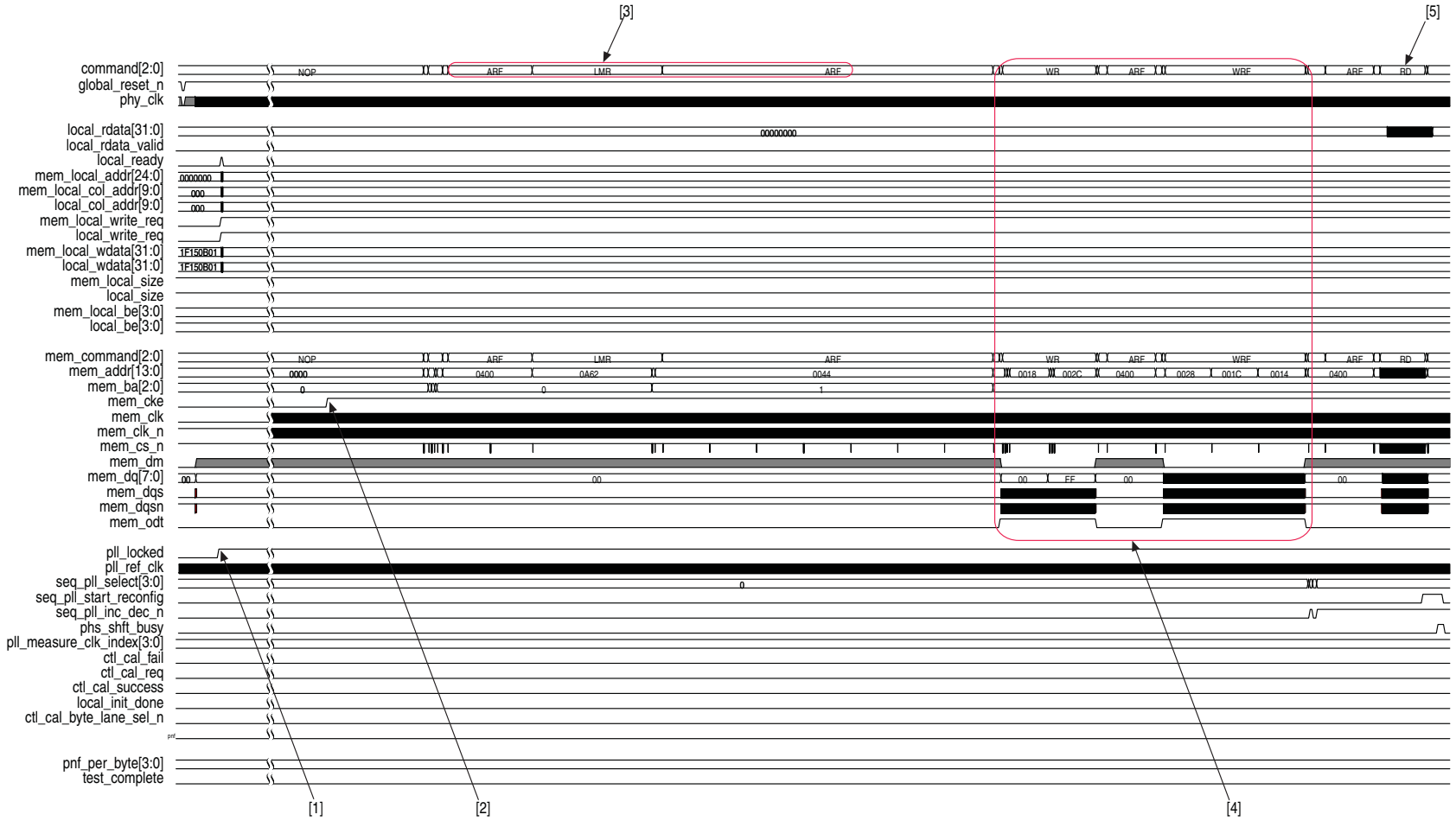


The following sequence corresponds with the numbered items in Figure 9–6:

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

# Initialization Timing

**Figure 9–7. Initialization Timing for HPC**



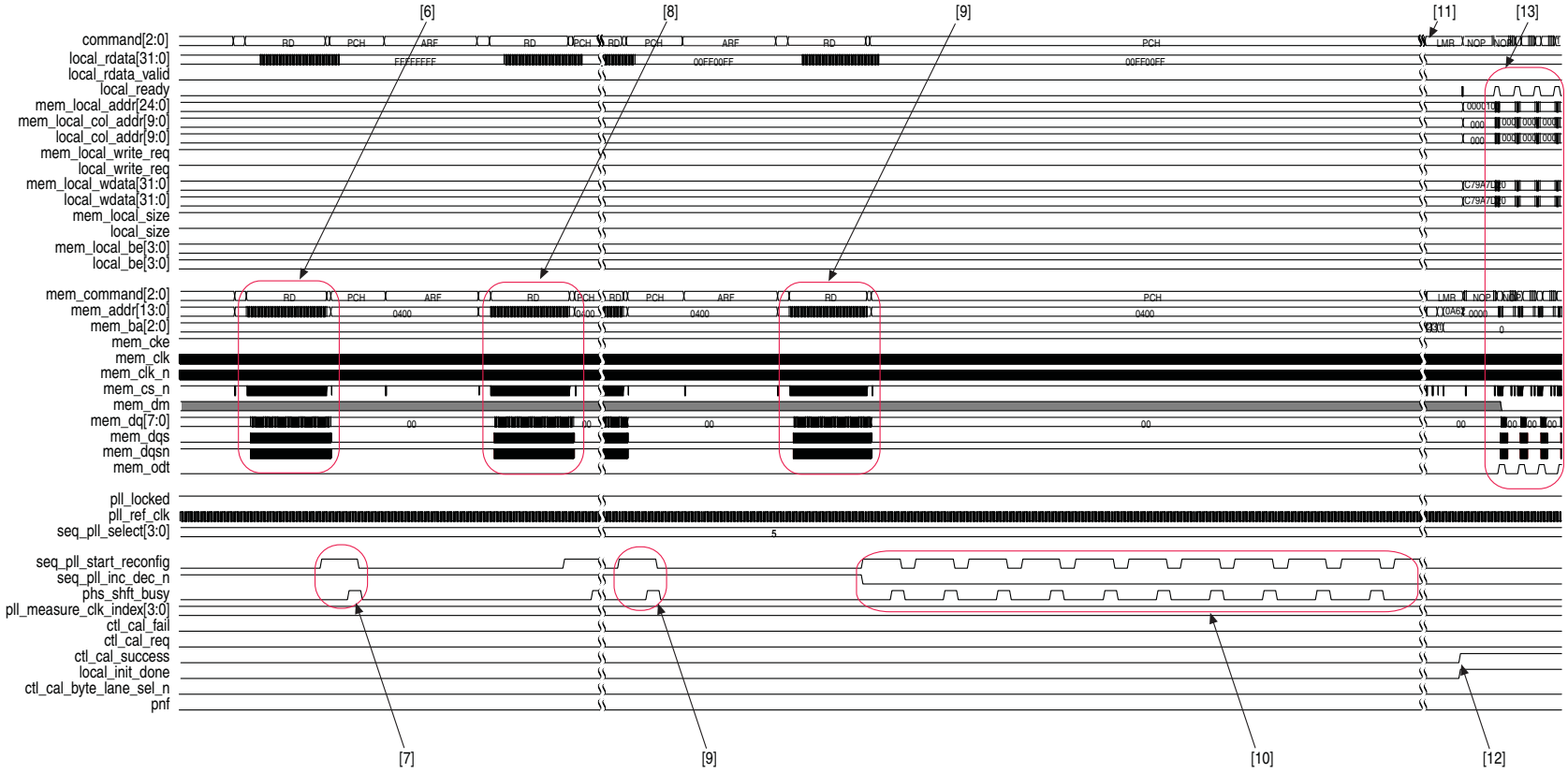

The following sequence corresponds with the numbered items in Figure 9–7:

1. The `pll_locked` signal goes high.
2. The `mem_cke` signal is asserted.

3. The first sequence of the initialization commands: PCH, LMR, PCH, ARF, LMR.
4. Write training data.
5. The first read command to read back training pattern.

## Calibration Timing

**Figure 9–8. Calibration Timing for HPC**




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

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

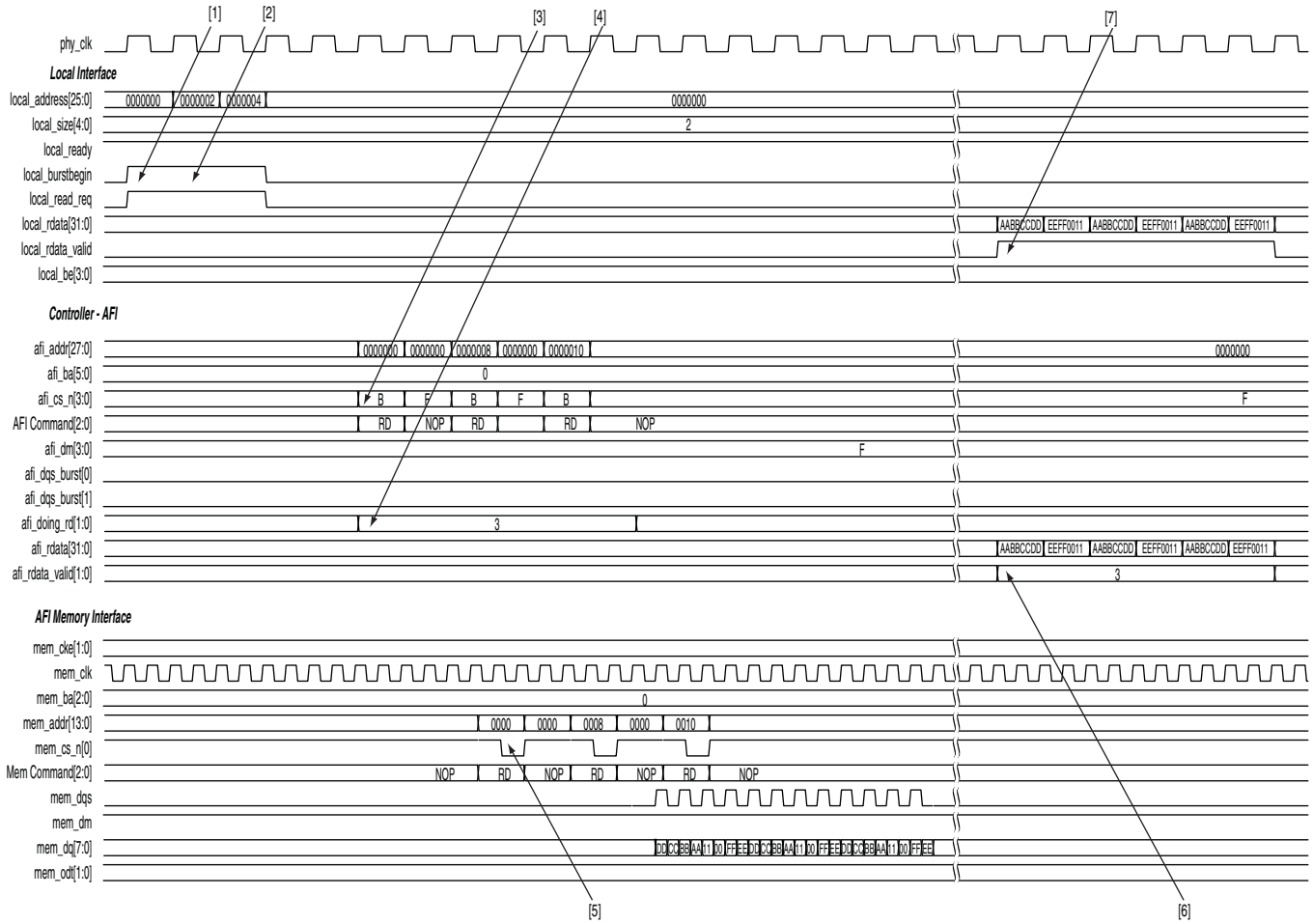
# DDR and DDR2 High-Performance Controllers II

This section discusses the following timing diagrams for HPC II:

- "Half-Rate Read"
- "Half-Rate Write"
- "Full-Rate Read"
- "Full-Rate Write"

## Half-Rate Read

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




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

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

   ```
   mem_row_address = 0x000000

   mem_col_address = 0x0000

   mem_bank_address = 0x00
   ```

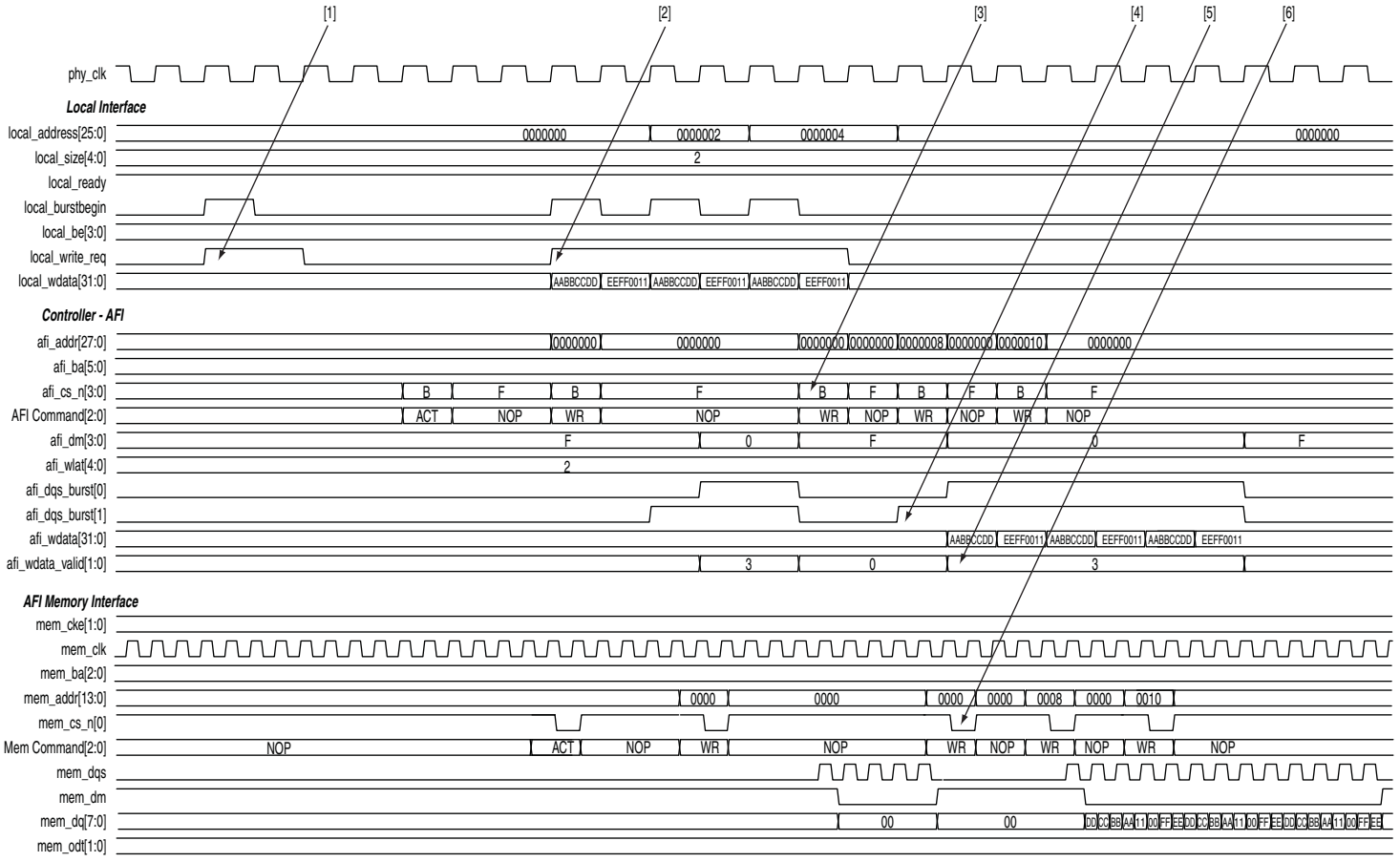
2. The user logic initiates a second read to a different memory column within the same row. The request for the second write is a burst length of 2. In this example, the user logic continues to accept commands until the command queue is full. When the command queue is full, the controller deasserts the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

   ```
   mem_row_address = 0x0000

   mem_col_address = 0x0002<<2 = 0x0008

   mem_bank_address = 0x00
   ```

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

# Half-Rate Write

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

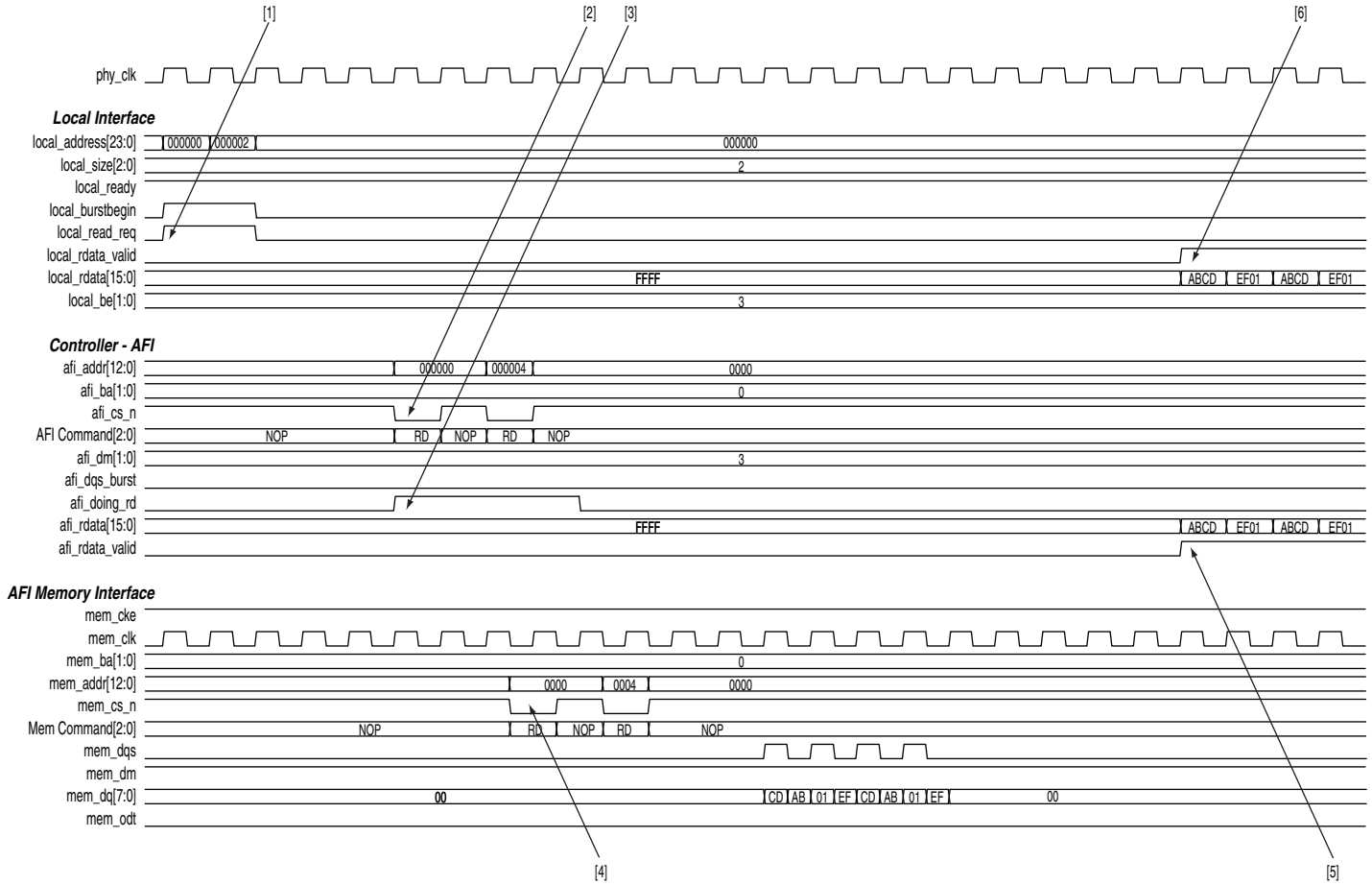


The following sequence corresponds with the numbered items in Figure 9–10:

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

# Full-Rate Read

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



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

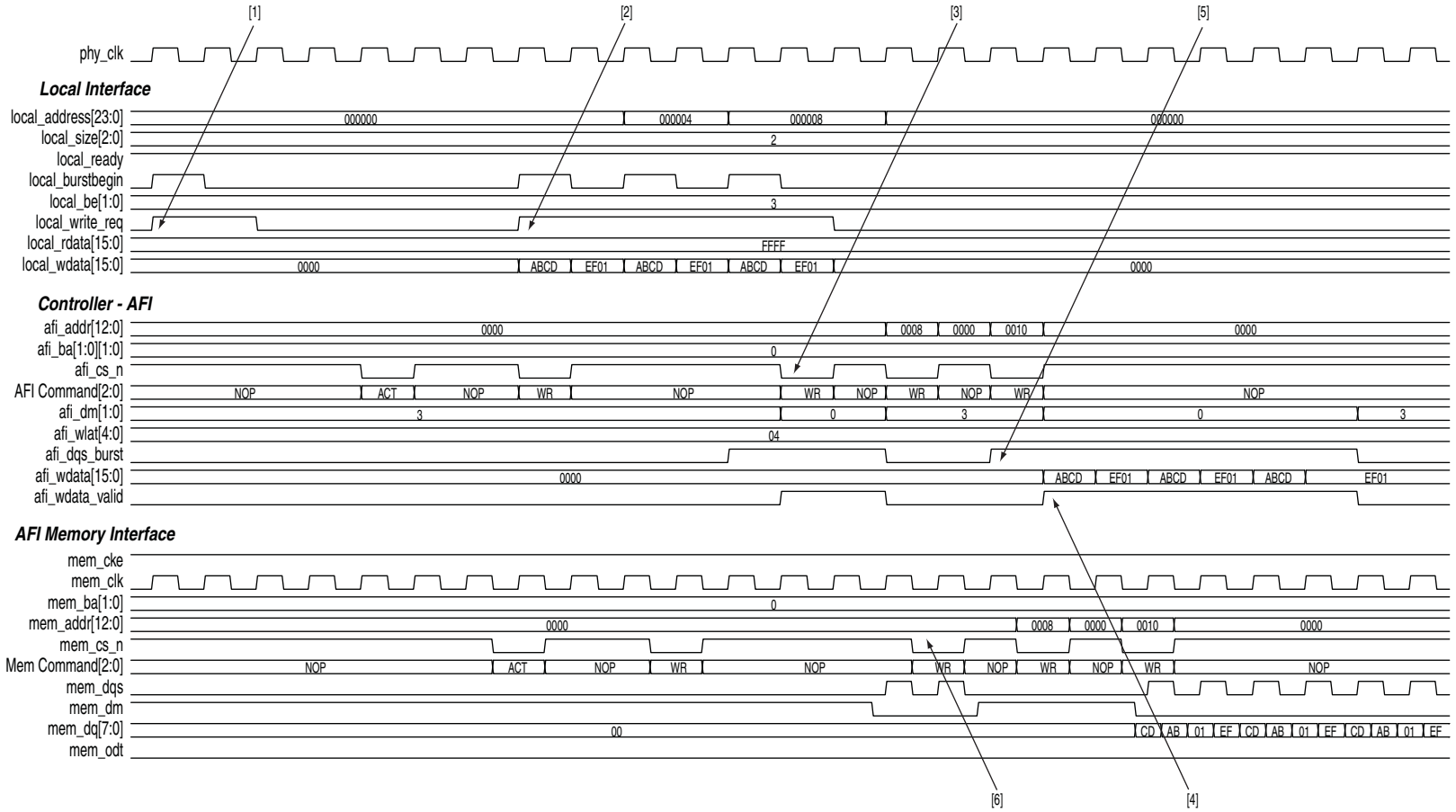
1. The user logic requests the first read by asserting `local_read_req` signal, and the size and address for this read. In this example, the request is a burst length of 2 to a local address `0x000000`. This local address is mapped to the following memory address in full-rate mode:

   ```
   mem_row_address = 0x0000

   mem_col_address = 0x0000<<2 = 0x0000

   mem_bank_address = 0x00
   ```

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

# Full-Rate Write

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



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

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

This chapter provides additional information about the document and Altera.

## Document Revision History

The following table shows the revision history for this document.

| Date | Version | Changes |
|---|---|---|
| December 2010 | 2.1 | Updated for 10.1. |
| July 2010 | 2.0 | ■ Added information for new GUI parameters: **Controller latency**, **Enable reduced bank tracking for area optimization**, and **Number of banks to track**.<br>■ Removed information about IP Advisor. This feature is removed from the DDR/DDR2 SDRAM IP support for version 10.0. |
| February 2010 | 1.3 | Corrected typos. |
| February 2010 | 1.2 | ■ Full support for Stratix IV devices.<br>■ Added timing diagrams for initialization and calibration stages for HPC. |
| November 2009 | 1.1 | Minor corrections. |
| November 2009 | 1.0 | First published. |

## How to Contact Altera

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

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

**Note to Table:**

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

# Typographic Conventions

The following table shows the typographic conventions this document uses.

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

# External Memory Interface Handbook Volume 3

# Section II. DDR3 SDRAM Controller with ALTMEMPHY IP User Guide

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

EMI_DDR3_UG-2.1

Document last updated for Altera Complete Design Suite version: 10.1
Document publication date: December 2010

Subscribe

# Contents

## Chapter 6. Functional Description—High-Performance Controller

## Chapter 7. Functional Description—High-Performance Controller II



## Chapter 8. Latency

## Chapter 9. Timing Diagrams

## Additional Information

# 1. About This IP

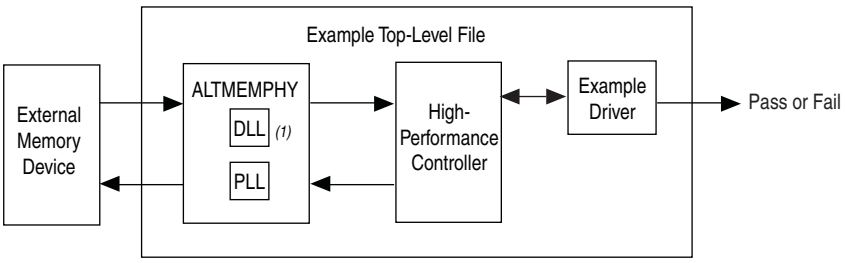
The Altera® DDR3 SDRAM Controller with ALTMEMPHY IP provides simplified interfaces to industry-standard DDR3 SDRAM. The ALTMEMPHY megafunction is an interface between a memory controller and the memory devices, and performs read and write operations to the memory. The DDR3 SDRAM Controller with ALTMEMPHY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

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

☞ DDR3 SDRAM high-performance controller denotes both HPC and HPC II unless indicated otherwise.

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

**Figure 1–1. System-Level Diagram**




**Note to Figure 1–1:**

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

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

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

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

# Release Information

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

**Table 1–1. Release Information**

| Item | Description |
|---|---|
| Version | 10.0 |
| Release Date | July 2010 |
| Ordering Codes | IP-SDRAM/DDR3 (HPC)<br>IP-HPMCII (HPC II) |
| Product IDs | 00C2 (DDR3 SDRAM)<br>00CO (ALTMEMPHY Megafunction) |
| Vendor ID | 6AF7 |

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

# Device Family Support

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

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

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

**Table 1–2. Device Family Support**

| Device Family | Support |
|---|---|
| Arria® II GX | Preliminary |
| HardCopy III | HardCopy Companion |
| HardCopy IV E | HardCopy Companion |
| HardCopy IV GX | HardCopy Companion |
| Stratix® III | Final |
| Stratix IV | Final |
| Other device families | No support |

# Features

The ALTMEMPHY megafunction offers the following features:

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

The ALTMEMPHY megafunction supports DDR3 SDRAM DIMMs with leveling and DDR3 SDRAM components without leveling:

- ALTMEMPHY with leveling is for unbuffered DIMMs (including SODIMM and MicroDIMM) or DDR3 SDRAM components up to 80-bit total data bus width with a layout like a DIMM that target Stratix III and Stratix IV devices:
  - Supports a fully-calibrated DDR3 SDRAM PHY for DDR3 SDRAM unbuffered DIMM with ×4 and ×8 devices with 300-MHz to 533-MHz frequency targets.
  - Deskew circuitry is enabled automatically for interfaces higher than 400 MHz.
  - Supports single and multiple chip selects.
- ALTMEMPHY supports DDR3 SDRAM components without leveling for Arria II GX, Stratix III, and Stratix IV devices using T-topology for clock, address, and command bus:
  - Supports multiple chip selects.
- The DDR3 SDRAM PHY with leveling $f_{MAX}$ is 533 MHz; without leveling $f_{MAX}$ is 400 MHz for single chip selects.

- No support for data-mask (DM) pins for ×4 DDR3 SDRAM DIMMs or components, so select **No** for **Drive DM pins from FPGA** when using ×4 devices.
- The ALTMEMPHY megafunction supports half-rate DDR3 SDRAM interfaces only.

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

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

| Features | Controller Architecture | |
|---|---|---|
| | HPC | HPC II |
| Half-rate controller | ✓ | ✓ |
| Support for AFI ALTMEMPHY | ✓ | ✓ |
| Support for Avalon® Memory Mapped (Avalon-MM) local interface | ✓ | ✓ |
| Support for Native local interface | ✓ | — |
| Configurable command look-ahead bank management with in-order reads and writes | — | ✓ |
| Additive latency | — | ✓ *(1)* |
| Optional support for multi-cast write for $t_{RC}$ mitigation | — | ✓ |
| Support for arbitrary Avalon burst length | — | ✓ |
| Built-in flexible memory burst adapter | — | ✓ |
| Configurable Local-to-Memory address mappings | — | ✓ |
| Integrated half-rate bridge for low latency option | — | ✓ |
| Optional run-time configuration of size and mode register settings, and memory timing | — | ✓ *(2)* |
| Partial array self-refresh (PASR) | — | ✓ |
| Support for industry-standard DDR3 SDRAM devices; and DIMMs | ✓ | ✓ |
| Optional support for self-refresh command | ✓ | ✓ |
| Optional support for user-controlled power-down command | ✓ | — |
| Optional support for automatic power-down command with programmable time-out | — | ✓ |
| Optional support for auto-precharge read and auto-precharge write commands | ✓ | — |
| Optional support for user-controller refresh | ✓ | ✓ |
| Reduced bank tracking for area optimization | — | ✓ |
| Controller variable latency | — | ✓ |
| Optional multiple controller clock sharing in SOPC Builder Flow | ✓ | ✓ |
| Integrated error correction coding (ECC) function 72-bit | ✓ | ✓ |
| Integrated ECC function 40-bit | — | ✓ |
| Support for partial-word write with optional automatic error correction | — | ✓ |
| SOPC Builder ready | ✓ | ✓ |
| Support for OpenCore Plus evaluation | ✓ | — |

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

| Features | Controller Architecture | |
|---|---|---|
| | HPC | HPC II |
| IP functional simulation models for use in Altera-supported VHDL and Verilog HDL simulator | ✓ | ✓ |

**Notes to Table 1–3:**

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

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

# Unsupported Features

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

- Timing simulation.
- Partial burst and unaligned burst in ECC and non-ECC mode when DM pins are disabled.

# MegaCore Verification

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

# Resource Utilization

The following sections show the resource utilization data for the ALTMEMPHY megafunction, and the DDR3 high-performance controllers (HPC and HPC II).

## ALTMEMPHY Megafunction

Table 1–4 and Table 1–5 show the typical size of the ALTMEMPHY megafunction with the AFI in the Quartus II software version 10.0 for the following devices:

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

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

| Memory Type | PHY Rate | Memory Width (Bits) | Combinational ALUTS | Logic Registers | M9K Blocks | Memory ALUTs |
|---|---|---|---|---|---|---|
| DDR3 SDRAM (without leveling) | Half | 8 | 1,431 | 1,189 | 2 | 18 |
| | | 16 | 1,481 | 1,264 | 4 | 2 |
| | | 64 | 1,797 | 1,970 | 12 | 22 |
| | | 72 | 1,874 | 2,038 | 13 | 2 |

**Note to Table 1–4:**

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

**Table 1–5. Resource Utilization in Stratix III and Stratix IV Devices** *(Note 1)*

| Memory Type | PHY Rate | Memory Width (Bits) | Combinational ALUTS | Logic Registers | M9K Blocks | Memory ALUTs |
|---|---|---|---|---|---|---|
| DDR3 SDRAM (400 MHz, without leveling only) | Half | 8 | 1,359 | 1,047 | 1 | 40 |
| | | 16 | 1,426 | 1,196 | 1 | 80 |
| | | 64 | 1,783 | 2,080 | 1 | 320 |
| | | 72 | 1,871 | 2,228 | 1 | 360 |
| DDR3 SDRAM (400 MHz, with leveling only) | | 8 | 3,724 | 2,723 | 2 | 80 |
| | | 16 | 4,192 | 3,235 | 2 | 160 |
| | | 64 | 6,835 | 6,487 | 5 | 640 |
| | | 72 | 7,182 | 6,984 | 5 | 720 |
| DDR3 SDRAM (533 MHz with read and write deskew, with leveling only) | | 8 | 4,098 | 2,867 | 2 | 80 |
| | | 16 | 4,614 | 3,391 | 2 | 160 |
| | | 64 | 7,297 | 6,645 | 5 | 640 |
| | | 72 | 7,641 | 7,144 | 5 | 720 |

**Note to Table 1–5:**

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

## High-Performance Controller

Table 1–6 and Table 1–7 show the typical sizes for the DDR3 SDRAM HPC (including ALTMEMPHY) for Stratix III and Stratix IV devices.

**Table 1–6. Resource Utilization in Stratix III Devices**

| Local Data Width (Bits) | Memory Width (Bits) | Combinational ALUTs | Dedicated Logic Registers | Memory (M9K) |
|---|---|---|---|---|
| 32 | 8 | 1,891 | 1,558 | 2 |
| 64 | 16 | 1,966 | 1,707 | 3 |
| 256 | 64 | 2,349 | 2,591 | 9 |
| 288 | 72 | 2,442 | 2,739 | 10 |

**Table 1–7. Resource Utilization in Stratix IV Devices**

| Local Data Width (Bits) | Memory Width (Bits) | Combinational ALUTs | Dedicated Logic Registers | Memory (M9K) |
|---|---|---|---|---|
| 32 | 8 | 1,924 | 1,580 | 2 |
| 64 | 16 | 1,987 | 1,724 | 3 |
| 256 | 64 | 2,359 | 2,584 | 9 |
| 288 | 72 | 2,449 | 2,728 | 10 |

## High-Performance Controller II

Table 1–9 through Table 1–10 show the typical sizes for the DDR3 SDRAM HPC II (including ALTMEMPHY) for Arria II GX, Stratix III, and Stratix IV devices.

**Table 1–8. Resource Utilization in Arria II GX Devices**

| Local Data Width (Bits) | Memory Width (Bits) | Combinational ALUTs | Dedicated Logic Registers | Memory (M9K) |
|---|---|---|---|---|
| 32 | 8 | 2,516 | 1,945 | 3 |
| 64 | 16 | 2,604 | 2,101 | 5 |
| 256 | 64 | 3,121 | 3,021 | 17 |
| 288 | 72 | 3,243 | 3,175 | 18 |

**Table 1–9. Resource Utilization in Stratix III Devices**

| Local Data Width (Bits) | Memory Width (Bits) | Combinational ALUTs | Dedicated Logic Registers | Memory (M9K) |
|---|---|---|---|---|
| 32 | 8 | 2,430 | 1,776 | 2 |
| 64 | 16 | 2,499 | 1,919 | 3 |
| 256 | 64 | 2,902 | 2,809 | 9 |
| 288 | 72 | 3,001 | 2,959 | 10 |

**Table 1–10. Resource Utilization in Stratix IV Devices**

| Local Data Width (Bits) | Memory Width (Bits) | Combinational ALUTs | Dedicated Logic Registers | Memory (M9K) |
|---|---|---|---|---|
| 32 | 8 | 2,427 | 1,773 | 2 |
| 64 | 16 | 2,496 | 1,914 | 3 |
| 256 | 64 | 2,887 | 2,774 | 9 |
| 288 | 72 | 2,981 | 2,924 | 10 |

# System Requirements

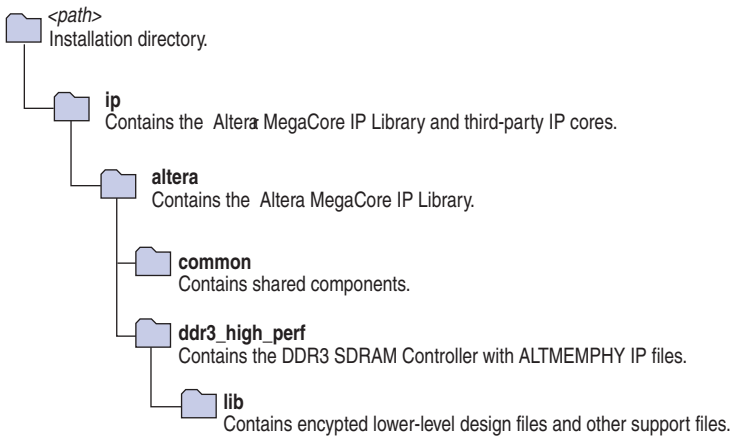
The DDR3 SDRAM Controller with ALTMEMPHY IP is a part of the MegaCore IP Library, which is distributed with the Quartus II software and downloadable from the Altera website, www.altera.com.

For system requirements and installation instructions, refer to *Altera Software Installation & Licensing*.

# Installation and Licensing

Figure 1–2 shows the directory structure after you install the DDR3 SDRAM Controller with ALTMEMPHY IP, where <*path*> is the installation directory. The default installation directory on Windows is **c:\altera**\<*version*>; on Linux it is **/opt/altera**<*version*>.

**Figure 1–2. Directory Structure**




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

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

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

## Free Evaluation

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

- Simulate the behavior of a megafunction (Altera MegaCore function or AMPP$^{SM}$ megafunction) within your system.
- Verify the functionality of your design, as well as evaluate its size and speed quickly and easily.
- Generate time-limited device programming files for designs that include MegaCore functions.
- Program a device and verify your design in hardware.

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

## OpenCore Plus Time-Out Behavior

OpenCore Plus hardware evaluation can support the following two modes of operation:

- Untethered—the design runs for a limited time
- Tethered—requires a connection between your board and the host computer. If tethered mode is supported by all megafunctions in a design, the device can operate for a longer time or indefinitely

All megafunctions in a device time-out simultaneously when the most restrictive evaluation time is reached. If there is more than one megafunction in a design, a specific megafunction's time-out behavior may be masked by the time-out behavior of the other megafunctions.

For MegaCore functions, the untethered time-out is 1 hour; the tethered time-out value is indefinite.

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

## Design Flow

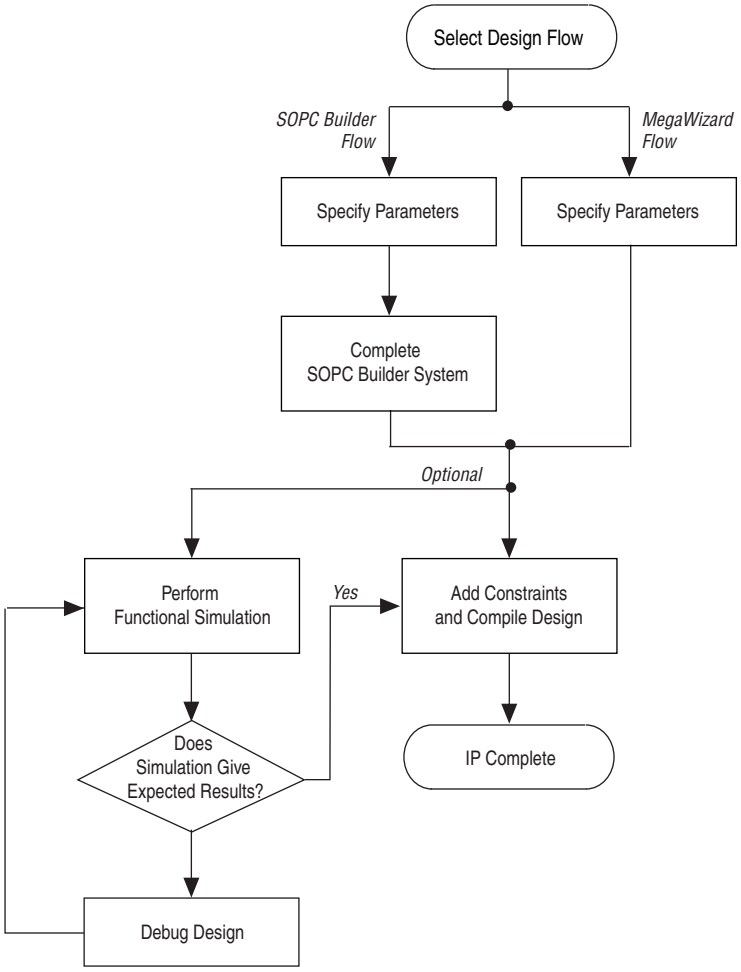
You can implement the DDR3 SDRAM Controller with ALTMEMPHY IP using either one of the following flows:

- SOPC Builder flow
- MegaWizard Plug-In Manager flow

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

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

**Figure 2–1. Design Flow**



The SOPC Builder flow offers the following advantages:

- Generates simulation environment
- Creates custom components and integrates them via the component wizard
- Interconnects all components with the Avalon-MM interface

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

- Allows you to design directly from the DDR3 SDRAM interface to peripheral device or devices
- Achieves higher-frequency operation

# SOPC Builder Flow

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

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

For more information about SOPC Builder, refer to volume 4 of the *Quartus II Handbook*. For more information about how to use controllers with SOPC Builder, refer to the *ALTMEMPHY Design Tutorials* section in volume 6 of the *External Memory Interface Handbook*. For more information on the Quartus II software, refer to the Quartus II Help.

## Specifying Parameters

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

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

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

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

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

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

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

## Completing the SOPC Builder System

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

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

   CAUTION

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

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

   To calculate the Avalon-MM address equivalent of the memory address range `0x0` to `0x47`, multiply the memory address by the width of the memory interface data bus in bytes. Refer to Table 2–1 for more Avalon-MM addresses.

**Table 2–1. Avalon-MM Addresses for AFI Mode**

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

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

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

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

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

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

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

# MegaWizard Plug-In Manager Flow

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

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

## Specifying Parameters

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

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

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

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

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

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

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

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

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

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

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

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

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

8. After you review the generation report, click **Exit** to close the MegaWizard Plug-In Manager.
9. For the high-performance controller (HPC or HPC II), set the <*variation name*>**_example_top.v** or **.vhd** file to be the project top-level design file.
   a. On the File menu, click **Open**.
   b. Browse to <*variation name*>**_example_top** and click **Open**.
   c. On the Project menu, click **Set as Top-Level Entity**.

# Generated Files

Table 2–2 shows the ALTMEMPHY generated files.

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

| File Name | Description |
|---|---|
| alt_mem_phy_defines.v | Contains constants used in the interface. This file is always in Verilog HDL regardless of the language you chose in the MegaWizard Plug-In Manager. |
| <*variation_name*>**.html** | Lists the top-level files created and ports used in the megafunction. |
| <*variation_name*>**.ppf** | Pin planner file for your ALTMEMPHY variation. |
| <*variation_name*>**.qip** | Quartus II IP file for your ALTMEMPHY variation, containing the files associated with this megafunction. |
| <*variation_name*>**.v/.vhd** | Top-level file of your ALTMEMPHY variation, generated based on the language you chose in the MegaWizard Plug-In Manager. |
| <*variation_name*>**.vho** | Contains functional simulation model for VHDL only. |
| <*variation_name*>**_alt_mem_phy_delay.vhd** | Includes a delay module for simulation. This file is only generated if you choose VHDL as the language of your MegaWizard Plug-In Manager output files. |
| <*variation_name*>**_alt_mem_phy_dq_dqs.vhd** or **.v** | Generated file that contains DQ/DQS I/O atoms interconnects and instance. Arria II GX devices only. |
| <*variation_name*>**_alt_mem_phy_dq_dqs_clearbox.txt** | Specification file that generates the <*variation_name*>**_alt_mem_phy_dq_dqs** file using the clearbox flow. Arria II GX devices only. |
| <*variation_name*>**_alt_mem_phy_pll.qip** | Quartus II IP file for the PLL that your ALTMEMPHY variation uses that contains the files associated with this megafunction. |
| <*variation_name*>**_alt_mem_phy_pll.v/.vhd** | The PLL megafunction file for your ALTMEMPHY variation, generated based on the language you chose in the MegaWizard Plug-In Manager. |
| <*variation_name*>**_alt_mem_phy_pll_bb.v/.cmp** | Black box file for the PLL used in your ALTMEMPHY variation. Typically unused. |
| <*variation_name*>_alt_mem_phy_seq.vhd | Contains the sequencer used during calibration. This file is always in VHDL language regardless of the language you chose in the MegaWizard Plug-In Manager. |
| <*variation_name*>**_alt_mem_phy_seq_wrapper.v/.vhd** | A wrapper file, for compilation only, that calls the sequencer file, created based on the language you chose in the MegaWizard Plug-In Manager. |
| <*variation_name*>**_alt_mem_phy_seq_wrapper.vo/.vho** | A wrapper file, for simulation only, that calls the sequencer file, created based on the language you chose in the MegaWizard Plug-In Manager. |

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

| File Name | Description |
|---|---|
| <*variation_name*>**_alt_mem_phy.v** | Contains all modules of the ALTMEMPHY variation except for the sequencer. This file is always in Verilog HDL language regardless of the language you chose in the MegaWizard Plug-In Manager. The DDR3 SDRAM sequencer is included in the <*variation_name*>**_alt_mem_phy_seq.**vhd file. |
| <*variation_name*>**_bb.v/.cmp** | Black box file for your ALTMEMPHY variation, depending whether you are using Verilog HDL or VHDL language. |
| <*variation_name*>**_ddr_pins.tcl** | Contains procedures used in the <*variation_name*>**_ddr_timing.sdc** and <*variation_name*>**_report_timing.tcl** files. |
| <*variation_name*>**_ddr_timing.sdc** | Contains timing constraints for your ALTMEMPHY variation. |
| <*variation_name*>**_pin_assignments.tcl** | Contains I/O standard, drive strength, output enable grouping, DQ/DQS grouping, and termination assignments for your ALTMEMPHY variation. If your top-level design pin names do not match the default pin names or a prefixed version, edit the assignments in this file. |
| <*variation_name*>**_report_timing.tcl** | Script that reports timing for your ALTMEMPHY variation during compilation. |

Table 2–3 shows the modules that are instantiated in the **<***variation_name***>_alt_mem_phy.v/.vhd** file. A particular ALTMEMPHY variation may or may not use any of the modules, depending on the memory standard that you specify.

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

| Module Name | Usage | Description |
|---|---|---|
| <*variation_name*>`_alt_mem_phy_addr_cmd` | All ALTMEMPHY variations | Generates the address and command structures. |
| <*variation_name*>`_alt_mem_phy_clk_reset` | All ALTMEMPHY variations | Instantiates PLL, DLL, and reset logic. |
| <*variation_name*>`_alt_mem_phy_dp_io` | All ALTMEMPHY variations | Generates the DQ, DQS, DM, and QVLD I/O pins. |
| <*variation_name*>`_alt_mem_phy_mimic` | DDR3 SDRAM ALTMEMPHY variation | Creates the VT tracking mechanism for DDR3 SDRAM PHYs. |
| <*variation_name*>`_alt_mem_phy_oct_delay` | DDR3 SDRAM ALTMEMPHY variation when dynamic OCT is enabled. | Generates the proper delay and duration for the OCT signals. |
| <*variation_name*>`_alt_mem_phy_postamble` | DDR3 SDRAM ALTMEMPHY variations | Generates the postamble enable and disable scheme for DDR3 PHYs. |
| <*variation_name*>`_alt_mem_phy_read_dp` | All ALTMEMPHY variations (unused for Stratix III or Stratix IV devices) | Takes read data from the I/O through a read path FIFO buffer, to transition from the resynchronization clock to the PHY clock. |

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

| Module Name | Usage | Description |
|---|---|---|
| <*variation_name*>_alt_mem_phy_read_dp_group | DDR3 SDRAM ALTMEMPHY variations (Stratix III and Stratix IV devices only) | A per DQS group version of <*variation_name*>_alt_mem_phy_read_dp. |
| <*variation_name*>_alt_mem_phy_rdata_valid | DDR3 SDRAM ALTMEMPHY variations | Generates read data valid signal to sequencer and controller. |
| <*variation_name*>_alt_mem_phy_seq_wrapper | All ALTMEMPHY variations | Generates sequencer for DDR3 SDRAM. |
| <*variation_name*>_alt_mem_phy_write_dp | All ALTMEMPHY variations | Generates the demultiplexing of data from half-rate to full-rate DDR data. |

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

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

**Table 2–4. Controller Generated Files—All High-Performance Controllers**

| Filename | Description |
|---|---|
| <*variation name*>**.bsf** | Quartus II symbol file for the MegaCore function variation. You can use this file in the Quartus II block diagram editor. |
| <*variation name*>**.html** | MegaCore function report file. |
| <*variation name*>**.v** or **.vhd** | A MegaCore function variation file, which defines a VHDL or Verilog HDL top-level description of the custom MegaCore function. Instantiate the entity defined by this file inside of your design. Include this file when compiling your design in the Quartus II software. |
| <*variation name*>**.qip** | Contains Quartus II project information for your MegaCore function variations. |
| <*variation name*>**.ppf** | XML file that describes the MegaCore pin attributes to the Quartus II Pin Planner. MegaCore pin attributes include pin direction, location, I/O standard assignments, and drive strength. If you launch IP Toolbench outside of the Pin Planner application, you must explicitly load this file to use Pin Planner. |
| <*variation name*>**_example_driver.v** or **.vhd** | Example self-checking test generator that matches your variation. |
| <*variation name*>**_example_top.v** or **.vhd** | Example top-level design file that you should set as your Quartus II project top level. Instantiates the example driver and the controller. |
| <*variation_name*>**_pin_assignments.tcl** | Contains I/O standard, drive strength, output enable grouping, and termination assignments for your ALTMEMPHY variation. If your top-level design pin names do not match the default pin names or a prefixed version, edit the assignments in this file. |

**Table 2–5. Controller Generated Files—DDR3 High-Performance Controller (HPC)**

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

**Table 2–6. Controller Generated Files—DDR3 High-Performance Controller II (HPC II)**

| Filename | Description |
|---|---|
| *<variation name>*_**alt_ddrx_controller_wrapper.v** or **.vho** | A controller wrapper that instantiates the **alt_ddrx_controller.v** file and configures the controller accordingly by the wizard. |
| alt_ddrx_addr_cmd.v | Decodes the state machine outputs into the memory address and command signals. |
| alt_ddrx_afi_block.v | Generates the read and write control signals for the AFI. |
| alt_ddrx_bank_tracking.v | Tracks which row is open in which memory bank. |
| alt_ddrx_clock_and_reset.v | Contains the clock and reset logic. |
| alt_ddrx_cmd_queue.v | Contains the command queue logic. |
| alt_ddrx_controller.v | The controller top-level file that instantiates all the sub-blocks. |
| alt_ddrx_csr.v | Contains the control and status register interface logic. |
| alt_ddrx_ddr3_odt_gen.v | Generates the on-die termination (ODT) control signal for DDR3 memory interfaces. |
| **alt_ddrx_avalon_if.v** | Communicates with the Avalon-MM interface. |
| alt_ddrx_decoder_40.v | Contains the 40 bit version of the ECC decoder logic. |
| alt_ddrx_decoder_72.v | Contains the 72 bit version of the ECC decoder logic. |
| alt_ddrx_decoder.v | Instantiates the appropriate width ECC decoder logic. |
| alt_ddrx_encoder_40.v | Contains the 40 bit version of the ECC encoder logic. |
| alt_ddrx_encoder_72.v | Contains the 72 bit version of the ECC encoder logic. |
| alt_ddrx_encoder.v | Instantiates the appropriate width ECC encoder logic. |
| alt_ddrx_input_if.v | The input input interface block. It instantiates the **alt_ddrx_cmd_queue.v**, **alt_ddrx_wdata_fifo.v**, and **alt_ddrx_avalon_if.v** files. |
| alt_ddrx_odt_gen.v | Instantiates the **alt_ddrx_ddr3_odt_gen.v** file selectively. It also controls the ODT addressing scheme. |
| alt_ddrx_state_machine.v | The main state machine of the controller. |
| alt_ddrx_timers_fsm.v | The state machine that tracks the per-bank timing parameters. |
| alt_ddrx_timers.v | Instantiates **alt_ddrx_timers_fsm.v** and contains the rank specific timing tracking logic. |
| alt_ddrx_wdata_fifo.v | The write data FIFO logic. This logic buffers the write data and byte enables from the Avalon interface. |
| alt_avalon_half_rate_bridge_constraints.sdc | Contains timing constraints if your design has the **Enable Half Rate Bridge** option turned on. |
| alt_avalon_half_rate_bridge.v | The integrated half-rate bridge logic block. |

# HardCopy Device Migration Guidelines

In HardCopy III and HardCopy IV designs where higher core performance is required and I/O performance is not a limiting factor, you can prototype your HardCopy design in a faster speed grade companion FPGA. However, this practice introduces some restrictions and limitations. For example, if you target a HardCopy device with an FPGA device as a prototype, the Quartus II Fitter restricts the VCO operating range of the PLL to the mid speed grade frequency, regardless of the actual speed grade of the FPGA that the design is targeting.

## Enabling Hardcopy Migration Performance Improvement with ALTMEMPHY

You can achieve improved performance when implementing an IP core for use with a HardCopy device by first generating your IP for a lower-speed FPGA to achieve optimal implementation, and then compiling your design for the higher-speed FPGA companion to your HardCopy device. This process is summarized below:

1. Generate your IP core, targeting a mid-speed grade FPGA.
2. Compile your design, targeting a faster speed grade FPGA.

The following sections discuss the above steps in greater detail.

### Generating Your IP Core For a Mid-speed Grade FPGA

When you parameterize and generate your controller using the ALTMEMPHY parameter editor, the PHY, PLL, and DLL are parameterized and generated together with the controller logic. Robust calibration and operation require that all of these blocks operate with matched settings. To ensure that you have matched settings, any IP that includes hard blocks should be generated in the MegaWizard Plug-In Manager targeting a mid-speed grade FPGA rather than the C2 speed grade. By targeting a mid-speed grade FPGA, you ensure that any process-dependant settings are appropriately restricted when the IP core is generated, thereby maintaining a consistent post-fit implementation throughout the compilation process. You can then compile the design for either a mid- or high-speed grade FPGA, depending on whether you want speed enhancements.

The following example illustrates this situation. Table 2–7 shows the key parameters.

**Table 2–7. PHY Sequencer Parameters (Part 1 of 2)**

| Parameter | Setting |
|---|---|
| DLL_DELAY_BUFFER_MODE | HIGH |
| DLL_DELAY_CHAIN_LENGTH | 10 |
| DQS_DELAY_CTL_WIDTH | 6 |
| DQS_OUT_MODE | DELAY_CHAIN2 |
| DQS_PHASE | 7200 |
| DQS_PHASE_SETTING | 2 |
| MEM_IF_CLK_PS | 3300 |
| MEM_IF_CLK_PS_STR | 3300 ps |
| MEM_IF_MR_0 | 4641 |

**Table 2–7. PHY Sequencer Parameters (Part 2 of 2)**

| Parameter | Setting |
|---|---|
| PLL_STEPS_PER_CYCLE | 40 |
| MEM_IF_ADDR_CMD_PHASE | 240 |

The sequencer in this example is set up to operate with 40 PLL phase steps per clock cycle. (This information appears in the message panel of the ALTMEMPHY parameter editor during generation of the IP core.)

## Compiling Your Design for a Faster Speed Grade FPGA

The ALTMEMPHY parameter editor generates PLL parameters that match the PHY requirement that the minimum PLL phase step size be one-eighth of the nominal VCO operating period. In a C2 speed grade FPGA, the VCO is configured to run at 1515 MHz with an associated phase step of 82 ps. Table 2–8 summarizes the generated PLL parameters. As shown, the PLL setup produces 40 phase steps per memory clock cycle, matching the sequencer setup. This analysis, however, does not apply when you select a HardCopy device.

**Table 2–8. Generated PLL Parameters for a C2 speed grade FPGA**

| Parameter | Setting |
|---|---|
| VCO OPERATING FREQUENCY | 1515 MHz |
| VCO PHASE SHIFT STEP | 82 ps |
| MEMORY CLOCK PERIOD | 3300 ps |
| PLL PHASE STEPS PER MEMCLK PERIOD | 40 |

The HardCopy flow targets the center of the silicon process; therefore, all hard IP blocks within the prototype FPGA must be configured accordingly to guarantee functional equivalency. When a HardCopy device is selected, the Quartus II Fitter restricts the operating range of the PLL to match the HardCopy silicon capability, regardless of the speed grade of the selected FPGA. This restriction can alter the final configuration of the PLL, producing a mismatch between the generated sequencer setup stored in RTL, and the PLL behavior generated by the Quartus II Fitter. Table 2–9 summarizes the post-fit PLL setup when HardCopy migration is selected, regardless of the chosen FPGA speed grade.

**Table 2–9. Post-fit PLL Parameters When Using a HardCopy Device**

| Parameter | Setting |
|---|---|
| VCO OPERATING FREQUENCY | 1212.1 MHz |
| VCO PHASE SHIFT STEP | 103 ps |
| MEMORY CLOCK PERIOD | 3300 ps |
| PLL PHASE STEPS PER MEMCLK PERIOD | 32 |

As shown in Table 2–9, the Quartus II Fitter restricts the VCO operating range to 1212.1 MHz, rather than the 1515 MHz of Table 2–8. This restriction produces a phase step mismatch between the PLL generated by the Quartus II Fitter and the PHY sequencer setup written in the RTL. Because the calibration process expects a common step size, the resulting design does not function properly in either the prototype FPGA or in the HardCopy device.

If you choose to prototype in a slower speed grade FPGA (C4) and target a HardCopy device, you must generate the ALTMEMPHY IP for the mid-speed grade FPGA to ensure that proper values are chosen for both the FPGA and HardCopy devices—this applies whether a performance improvement is desired or not.

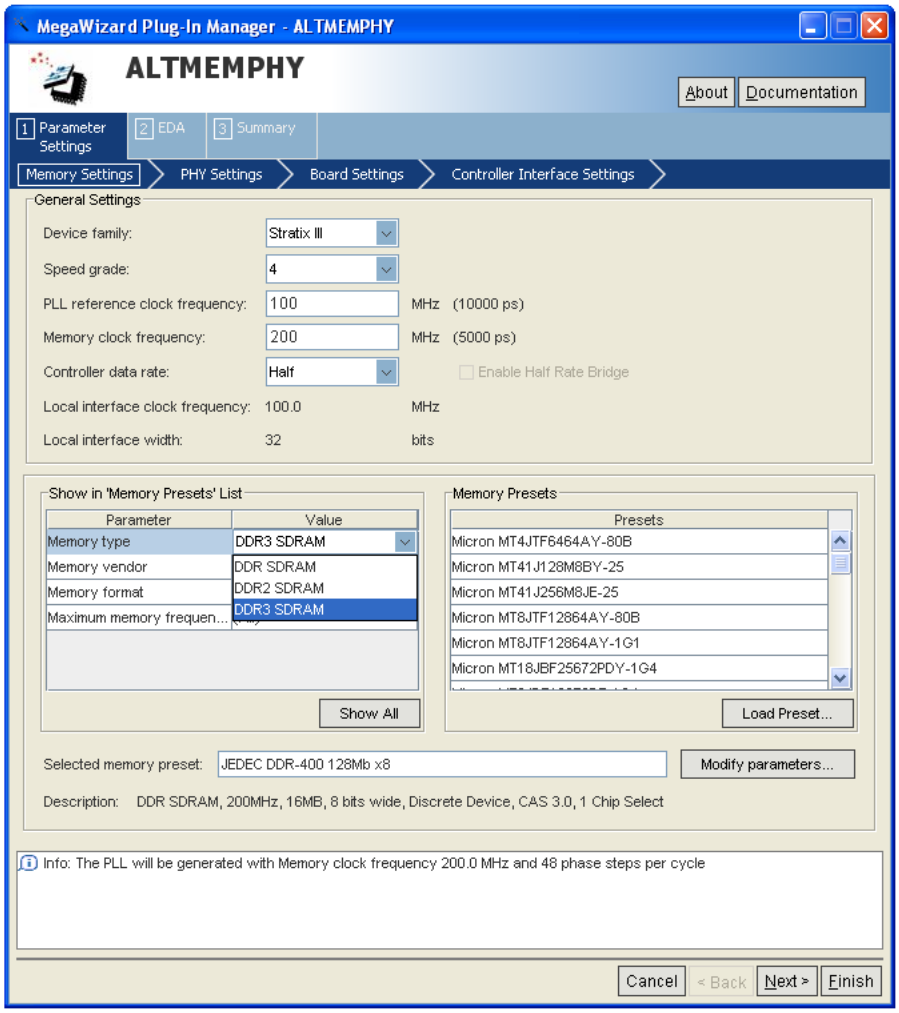
For information about HardCopy issues such as vertical I/O overhang, PLLs adjacent to I/Os, performance improvement, and timing closure, refer to HardCopy III Device I/O Features in the *HardCopy III Device Handbook, Volume 1*, and HardCopy IV Device I/O Features in the *HardCopy IV Device Handbook, Volume 1*.

# ALTMEMPHY Parameter Settings

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

- Memory Settings
- PHY Settings
- Board Settings

**Figure 3–1. ALTMEMPHY Parameter Settings Page**




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

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

## Memory Settings

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

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

**Table 3–1. General Settings**

| Parameter Name | Description |
|---|---|
| Device family | Targets device family (for example, Stratix III). Table 1–2 on page 1–3 shows supported device families. The device family selected here must match the device family selected on the MegaWizard page 2a. |
| Speed grade | Selects a particular speed grade of the device (for example, 2, 3, or 4 for the Stratix III device family). |
| PLL reference clock frequency | Determines the clock frequency of the external input clock to the PLL. Ensure that you use three decimal points if the frequency is not a round number (for example, 166.667 MHz or 100 MHz) to avoid a functional simulation or a PLL locking problem. |
| Memory clock frequency | Determines the memory interface clock frequency. If you are operating a memory device below its maximum achievable frequency, ensure that you enter the actual frequency of operation rather than the maximum frequency achievable by the memory device. Also, ensure that you use three decimal points if the frequency is not a round number (for example, 333.333 MHz or 400 MHz) to avoid a functional simulation or a PLL locking issue. |
| Controller data rate | Selects the data rate for the memory controller. Sets the frequency of the controller to equal to either the memory interface frequency (full-rate) or half of the memory interface frequency (half-rate). The full-rate option is not available for DDR3 SDRAM devices. |
| Enable half rate bridge | This option is only available for HPC II full-rate controller.<br>Turn on to keep the controller in the memory full clock domain while allowing the local side to run at half the memory clock speed, so that latency can be reduced. |
| Local interface clock frequency | Value that depends on the memory clock frequency and controller data rate, and whether or not you turn on the **Enable Half Rate Bridge** option. |
| Local interface width | Value that depends on the memory clock frequency and controller data rate, and whether or not you turn on the **Enable Half Rate Bridge** option. |

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

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

**Table 3–2. Memory Presets List**

| Parameter Name | Description |
|---|---|
| Memory type | You can filter the type of memory to display, for example, DDR3 SDRAM. |
| Memory vendor | You can filter the memory types by vendor. JEDEC is also one of the options, allowing you to choose the JEDEC specifications. If your chosen vendor is not listed, you can choose JEDEC for the DDR3 SDRAM interfaces. Then, pick a device that has similar specifications to your chosen device and check the values of each parameter. Make sure you change the each parameter value to match your device specifications. |
| Memory format | You can filter the type of memory by format, for example, discrete devices or DIMM packages. |
| Maximum frequency | You can filter the type of memory by the maximum operating frequency. |

## Using the Preset Editor to Create a Custom Memory Preset

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

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

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

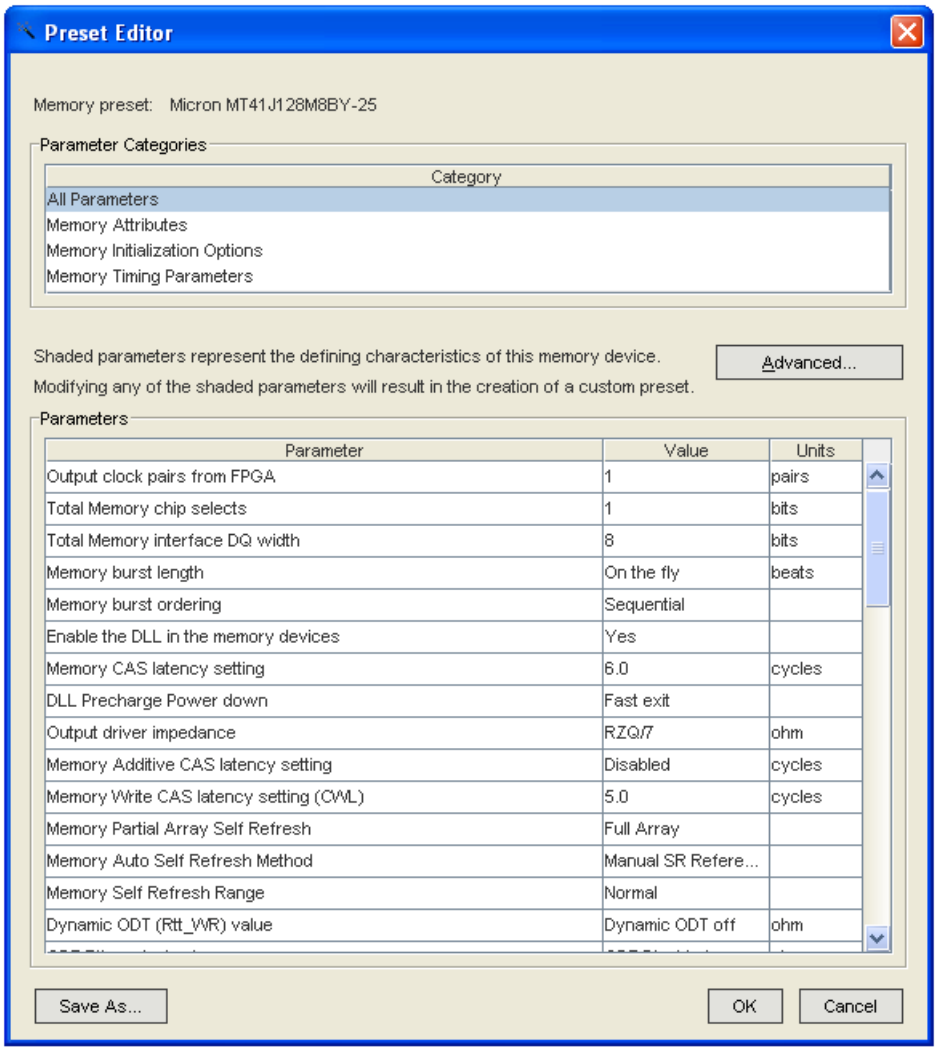
You can change the parameters with a white background to reflect your system. You can also change the parameters with a gray background so the device parameters match the device you are using. These parameters in gray background are characteristics of the chosen memory device and changing them creates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

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

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

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

**Figure 3–2. DDR3 SDRAM Preset Editor**




The **Advanced** option is only available for Arria II GX and Stratix IV devices. This option shows the percentage of memory specification that is calibrated by the FPGA. The percentage values are estimated by Altera based on the process variation.

Table 3–3 through Table 3–5 describe the DDR3 SDRAM parameters available for memory attributes, initialization options, and timing parameters.

**Table 3–3. DDR3 SDRAM Attributes Settings (Part 1 of 2)**

| Parameter Name | Range *(1)* | Units | Description |
|---|---|---|---|
| Output clock pairs from FPGA | 1–6 | pairs | Defines the number of differential clock pairs driven from the FPGA to the memory. Memory clock pins use the signal splitter feature in Arria II GX, Stratix III, and Stratix IV devices for differential signaling.<br>The ALTMEMPHY parameter editor displays an error on the bottom of the window if you choose more than one for DDR3 SDRAM interfaces. |
| Total Memory chip selects | 1, 2, 4, or 8 | bits | Sets the number of chip selects in your memory interface. The number of chip selects defines the depth of your memory. You are limited to the range shown as the local side binary encodes the chip select address. |
| Memory interface DQ width | 4–288 | bits | Defines the total number of DQ pins on the memory interface. If you are interfacing with multiple devices, multiply the number of devices with the number of DQ pins per device. Even though the GUI allows you to choose 288-bit DQ width, DDR3 SDRAM variations are only supported up to 80-bit width due to restrictions in the board layout which affects timing at higher data width. Furthermore, the interface data width is limited by the number of pins on the device. For best performance, have the whole interface on one side of the device. |
| Mirror addressing | — | — | On multiple rank DDR3 SDRAM DIMMs address signals are routed differently to each rank; referred to in the JEDEC specification as address mirroring.<br>Enter ranks with mirrored addresses in this field. There is one bit per chip select. For example, for four chip selects, enter 1011 to mirror the address on chip select #3, #1, and #0. |
| Register Control Word 0–15 for Registered DIMMs | — | bits | Register Control Word values for the DDR3 registered DIMMs. The values are available in the memory data sheet of the respective registered DIMMs. |
| Memory vendor | Elpida, JEDEC, Micron, Samsung, Hynix, Nanya, other | — | Lists the name of the memory vendor for all supported memory standards. |
| Memory format | Discrete Device, Unbuffered DIMM | — | Specifies whether you are interfacing with devices or modules. SODIMM and MicroDIMM are supported under unbuffered DIMMs. The ALTMEMPHY megafunction for DDR3 SDRAM interfaces does not support registered DIMM format. Arria II GX devices only support DDR3 SDRAM components without leveling, for example, **Discrete Device** memory format. |
| Maximum memory frequency | See the memory device datasheet | MHz | Sets the maximum frequency supported by the memory. |
| Column address width | 10–12 | bits | Defines the number of column address bits for your interface. |

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

| Parameter Name | Range *(1)* | Units | Description |
|---|---|---|---|
| Row address width | 12–16 | bits | Defines the number of row address bits for your interface. If your DDR3 SDRAM device's row address bus is 12-bit wide, set the row address width to **13** and set the 13$^{th}$ bit to logic-level low (or leave the 13$^{th}$ bit unconnected to the memory device) in the top-level file. |
| Bank address width | 3 | bits | Defines the number of bank address bits for your interface. |
| Chip selects per DIMM | 1 or 2 | bits | Defines the number of chip selects on each DIMM in your interface. Currently, calibration is done with all ranks but you can only perform timing analysis with one single-rank DIMM. |
| DQ bits per DQS bit | 4 or 8 | bits | Defines the number of data (DQ) bits for each data strobe (DQS) pin. |
| Drive DM pins from FPGA | Yes or No | — | Specifies whether you are using DM pins for write operation. Altera devices do not support DM pins with ×4 mode. |
| Maximum memory frequency for CAS latency 5.0 | 80–700 | MHz | Specifies the frequency limits from the memory data sheet per given CAS latency. The ALTMEMPHY MegaWizard Plug-In Manager generates a warning if the operating frequency with your chosen CAS latency exceeds this number. The lowest frequency supported by DDR3 SDRAM devices is 300 MHz. |
| Maximum memory frequency for CAS latency 6.0 | | | |
| Maximum memory frequency for CAS latency 7.0 | | | |
| Maximum memory frequency for CAS latency 8.0 | | | |
| Maximum memory frequency for CAS latency 9.0 | | | |
| Maximum memory frequency for CAS latency 10.0 | | | |

**Note to Table 3–3:**

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

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

| Parameter Name | Range | Units | Description |
|---|---|---|---|
| Memory burst length | 4, 8, on-the-fly | beats | Sets the number of words read or written per transaction. |
| Memory burst ordering | Sequential or Interleaved | — | Controls the order in which data is transferred between memory and the FPGA during a read transaction. For more information, refer to the memory device datasheet. |
| DLL precharge power down | Fast exit or Slow exit | — | Sets the mode register setting to disable (**Slow exit**) or enable (**Fast exit**) the memory DLL when CKE is disabled. |

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

| Parameter Name | Range | Units | Description |
|---|---|---|---|
| Enable the DLL in the memory devices | Yes or No | — | Enables the DLL in the memory device when set to **Yes**. You must always enable the DLL in the memory device as Altera does not guarantee any ALTMEMPHY operation when the DLL is turned off. All timings from the memory devices are invalid when the DLL is turned off. |
| ODT `Rtt` nominal value | ODT disable, RZQ/4, RZQ/2, RZQ/6 | W | RZQ in DDR3 SDRAM interfaces are set to 240 Ω. Sets the on-die termination (ODT) value to either 60 Ω (**RZQ/4**), 120 Ω (**RZQ/2**), or 40 Ω (**RZQ/6**). Set this to **ODT disable** if you are not planning to use ODT. For a single-ranked DIMM, set this to **RZQ/4**. |
| Dynamic ODT (`Rtt_WR`) value | Dynamic ODT off, RZQ/4, RZQ/2 | W | RZQ in DDR3 SDRAM interfaces are set to 240 Ω. Sets the memory ODT value during write operations to 60 Ω (**RZQ/4**) or 120 Ω (**RZQ/2**). As ALTMEMPHY only supports single rank DIMMs, you do not need this option (set to **Dynamic ODT off**). |
| Output driver impedance | RZQ/6 (Reserved) or RZQ/7 | W | RZQ in DDR3 SDRAM interfaces are set to 240 Ω. Sets the output driver impedance from the memory device. Some devices may not have **RZQ/6** available as an option. Be sure to check the memory device datasheet before choosing this option. |
| Memory CAS latency setting | 5.0, 6.0, 7.0, 8.0, 9.0, 10.0 | cycles | Sets the delay in clock cycles from the read command to the first output data from the memory. |
| Memory additive CAS latency setting | Disable, CL – 1, CL – 2 | cycles | Allows you to add extra latency in addition to the CAS latency setting. |
| Memory write CAS latency setting (CWL) | 5.0, 6.0, 7.0, 8.0 | cycles | Sets the delay in clock cycles from the write command to the first expected data to the memory. |
| Memory partial array self refresh | Full array, Half array {BA[2:0]=000,001, 010,011}, Quarter array {BA[2:0]=000,001}, Eighth array {BA[2:0]=000}, Three Quarters array {BA[2:0]=010,011, 100,101,110,111}, Half array {BA[2:0]=100,101, 110,111}, Quarter array {BA[2:0]=110, 111}, Eighth array {BA[2:0]=111} | — | Determine whether you want to self-refresh only certain arrays instead of the full array. According to the DDR3 SDRAM specification, data located in the array beyond the specified address range are lost if **self refresh** is entered when you use this. This option is not supported by the DDR3 SDRAM Controller with ALTMEMPHY IP, so set to **Full Array** if you are using the Altera controller. |

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

| Parameter Name | Range | Units | Description |
|---|---|---|---|
| Memory auto self refresh method | Manual SR reference (SRT) or ASR enable (Optional) | — | Sets the auto self-refresh method for the memory device. The DDR3 SDRAM Controller with ALTMEMPHY IP currently does not support the ASR option that you need for extended temperature memory self-refresh. |
| Memory self refresh range | Normal or Extended | — | Determines the temperature range for self refresh. You need to also use the optional auto self refresh option when using this option. The Altera controller currently does not support the extended temperature self-refresh operation. |

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

| Parameter Name | Range | Units | Description |
|---|---|---|---|
| Time to hold memory reset before beginning calibration | 0–1000000 | µs | Minimum time to hold the reset after a power cycle before issuing the MRS commands during the DDR3 SDRAM device initialization process. |
| $t_{INIT}$ | 0.001–1000 | µs | Minimum memory initialization time. After reset, the controller does not issue any commands to the memory during this period. |
| $t_{MRD}$ | 2–39 | ns | Minimum load mode register command period. The controller waits for this period of time after issuing a load mode register command before issuing any other commands.<br>$t_{MRD}$ is specified in ns in the DDR3 SDRAM high-performance controller and in terms of $t_{CK}$ cycles in Micron's device datasheet. Convert $t_{MRD}$ to ns by multiplying the number of cycles specified in the datasheet times $t_{CK}$, where $t_{CK}$ is the memory operation frequency and not the memory device's $t_{CK}$. |
| $t_{RAS}$ | 8–200 | ns | Minimum active to precharge time. The controller waits for this period of time after issuing an active command before issuing a precharge command to the same bank. |
| $t_{RCD}$ | 4–65 | ns | Minimum active to read-write time. The controller does not issue read or write commands to a bank during this period of time after issuing an active command. |
| $t_{RP}$ | 4–65 | ns | Minimum precharge command period. The controller does not access the bank for this period of time after issuing a precharge command. |
| $t_{REFI}$ | 1–65534 | µs | Maximum interval between refresh commands. The controller performs regular refresh at this interval unless user-controlled refresh is turned on. |
| $t_{RFC}$ | 14–1651 | ns | Minimum autorefresh command period. The length of time the controller waits before doing anything else after issuing an auto-refresh command. |
| $t_{WR}$ | 4–65 | ns | Minimum write recovery time. The controller waits for this period of time after the end of a write transaction before issuing a precharge command. |

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

| Parameter Name | Range | Units | Description |
|---|---|---|---|
| $t_{WTR}$ | 1–6 | $t_{CK}$ | Minimum write-to-read command delay. The controller waits for this period of time after the end of a write command before issuing a subsequent read command to the same bank. This timing parameter is specified in clock cycles and the value is rounded off to the next integer. |
| $t_{AC}$ | 0–750 | ps | DQ output access time. |
| $t_{DQSCK}$ | 50–750 | ps | DQS output access time from CK/CK# signals. |
| $t_{DQSQ}$ | 50–500 | ps | The maximum DQS to DQ skew; DQS to last DQ valid, per group, per access. |
| $t_{DQSS}$ | 0–0.3 | $t_{CK}$ | Positive DQS latching edge to associated clock edge. |
| $t_{DH}$ | 10–600 | ps | DQ and DM input hold time relative to DQS, which has a derated value depending on the slew rate of the differential DQS and DQ/DM signals. Ensure that you are using the correct number and that the value entered is referenced to $V_{REF}$(dc), not $V_{IH}$(dc) min or $V_{IL}$(dc) max. Refer to "Derating Memory Setup and Hold Timing" on page 3–10 for more information about how to derate this specification. |
| $t_{DS}$ | 10–600 | ps | DQ and DM input setup time relative to DQS, which has a derated value depending on the slew rate of the differential DQS signals and DQ/DM signals. Ensure that you are using the correct number and that the value entered is referenced to $V_{REF}$(dc), not $V_{IH}$(ac) min or $V_{IL}$(ac) max. Refer to "Derating Memory Setup and Hold Timing" on page 3–10 for more information about how to derate this specification. |
| $t_{DSH}$ | 0.1–0.5 | $t_{CK}$ | DQS falling edge hold time from CK. |
| $t_{DSS}$ | 0.1–0.5 | $t_{CK}$ | DQS falling edge to CK setup. |
| $t_{IH}$ | 50–1000 | ps | Address and control input hold time, which has a derated value depending on the slew rate of the CK and CK# clocks and the address and command signals. Ensure that you are using the correct number and that the value entered is referenced to $_{VREF}$(dc), not $V_{IH}$(dc) min or $V_{IL}$(dc) max. Refer to "Derating Memory Setup and Hold Timing" on page 3–10 for more information about how to derate this specification. |
| $t_{IS}$ | 65–1000 | ps | Address and control input setup time, which has a derated value depending on the slew rate of the CK and CK# clocks and the address and command signals. Ensure that you are using the correct number and that the value entered is referenced to $V_{REF}$(dc), not $V_{IH}$(ac) min or $V_{IL}$(ac) max. Refer to "Derating Memory Setup and Hold Timing" on page 3–10 for more information about how to derate this specification. |
| $t_{QHS}$ | 0–700 | ps | The maximum data hold skew factor. |
| $t_{QH}$ | 0.1–0.6 | $t_{CK}$ | DQ output hold time. |

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

| Parameter Name | Range | Units | Description |
|---|---|---|---|
| $t_{RRD}$ | 2.06–64 | ns | The activate to activate time, per device, RAS to RAS delay timing parameter. |
| $t_{FAW}$ | 7.69–256 | ns | The four-activate window time, per device. |
| $t_{RTP}$ | 2.06–64 | ns | Read to precharge time. |

**Note to Table 3–5:**

(1) See the memory device data sheet for the parameter range. Some of the parameters may be listed in a clock cycle ($t_{CK}$) unit. If the MegaWizard Plug-In Manager requires you to enter the value in a time unit (ps or ns), convert the number by multiplying it with the clock period of your interface (and not the maximum clock period listed in the memory data sheet).

## Derating Memory Setup and Hold Timing

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

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

For Arria II GX and Stratix IV devices, you need not derate using the Preset Editor. You only need to enter the parameters referenced to $V_{REF}$, and the deration is done automatically when you enter the slew rate information on the **Board Settings** tab.

After derating the values, you then need to normalize the derated value because Altera input and output timing specifications are referenced to $V_{REF}$. When the memory device setup and hold time numbers are derated and normalized to $V_{REF}$, update these values in the **Preset Editor** dialog box to ensure that your timing constraints are correct.

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

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

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

The $V_{REF}$ referenced setup and hold signals for a rising edge are:

$t_{DS}$ ($V_{REF}$) = Base $t_{DS}$ + delta $t_{DS}$ + ($V_{IH}$(ac) – $V_{REF}$)/slew_rate = 25 + 0 + 175 = 200 ps

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

If the output slew rate of the write data is different from 1V/ns, you have to first derate the $t_{DS}$ and $t_{DH}$ values, then translate these AC/DC level specs to $V_{REF}$ specification.

For a 2V/ns DQ slew rate rising signal and 2V/ns DQS-DQSn slew rate:

$t_{DS}$ ($V_{REF}$) = Base $t_{DS}$ + delta $t_{DS}$ + ($V_{IH}$(ac) – $V_{REF}$)/slew_rate = 25 + 88 + 87.5 = 200.5 ps

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

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

$t_{DS}$ ($V_{REF}$) = Base $t_{DS}$ + delta $t_{DS}$ + ($V_{IH}$(ac) – $V_{REF}$)/slew_rate = 25 + 5 + 350 = 380 ps

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

## PHY Settings

Click **Next** or the **PHY Settings** tab to set the options described in Table 3–6. The options are available if they apply to the target Altera device.

**Table 3–6. ALTMEMPHY PHY Settings (Part 1 of 2)**

| Parameter Name | Applicable Device Families | Description |
|---|---|---|
| Use dedicated PLL outputs to drive memory clocks | HardCopy II and Stratix II (prototyping for HardCopy II) | This option is disabled for DDR3 SDRAM. |
| Dedicated memory clock phase | HardCopy II and Stratix II (prototyping for HardCopy II) | This option is disabled for DDR3 SDRAM. |
| Use differential DQS | Arria II GX, Stratix III, and Stratix IV | This option is disabled for DDR3 SDRAM. |
| Enable external access to reconfigure PLL prior to calibration | HardCopy II, Stratix III, and Stratix IV (prototyping for HardCopy II) | When enabling this option for HardCopy II, Stratix III, and Stratix IV devices, the inputs to the ALTPLL_RECONFIG megafunction are brought to the top level for debugging purposes.<br>This option allows you to reconfigure the PLL before calibration to adjust, if necessary, the phase of the memory clock (`mem_clk_2x`) before the start of the calibration of the resynchronization clock on the read side. The calibration of the resynchronization clock on the read side depends on the phase of the memory clock on the write side. |

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

| Parameter Name | Applicable Device Families | Description |
|---|---|---|
| Instantiate DLL externally | All supported device families. | Use this option with Stratix III, Stratix IV, HardCopy III, or HardCopy IV devices, if you want to apply a non-standard phase shift to the DQS capture clock. The ALTMEMPHY DLL offsetting I/O can then be connected to the external DLL and the Offset Control Block. |
| Enable dynamic parallel on-chip termination (OCT) | Stratix III and Stratix IV | This option provides I/O impedance matching and termination capabilities. The ALTMEMPHY megafunction enables parallel termination during reads and series termination during writes with this option checked. Only applicable for DDR3 SDRAM interfaces where DQ and DQS are bidirectional. Using the dynamic termination requires that you use the OCT calibration block, which may impose a restriction on your DQS/DQ pin placements depending on your $R_{UP}/R_{DN}$ pin locations.<br>For more information, refer to either the *External Memory Interfaces in Stratix III Devices* chapter in volume 1 of the *Stratix III Device Handbook* or the *External Memory Interfaces in Stratix IV Devices* chapter in volume 1 of the *Stratix IV Device Handbook*. |
| Clock phase | Arria II GX | Adjusting the address and command phase can improve the address and command setup and hold margins at the memory device to compensate for the propagation delays that vary with different loadings. You have a choice of 0°, 90°, 180°, and 270°, based on the rising and falling edge of the `phy_clk` and `write_clk` signals. In Stratix IV and Stratix III devices, the clock phase is set to **dedicated**. |
| Dedicated clock phase | Stratix III and Stratix IV | When you use a dedicated PLL output for address and command, you can choose any legal PLL phase shift to improve setup and hold for the address and command signals. You can set this value to between 180° and 359° (the default is 240°). However, generally PHY timing requires a value of greater than 240° for half-rate designs. |
| Board skew | All supported device families except Arria II GX and Stratix IV devices | Maximum skew across any two memory interface signals for the whole interface from the FPGA to the memory (either a discrete memory device or a DIMM). This parameter includes all types of signals (data, strobe, clock, address, and command signals). You need to input the worst-case skew, whether it is within a DQS/DQ group, or across all groups, or across the address and command and clocks signals. This parameter generates the timing constraints in the **.sdc** file. |
| Autocalibration simulation options | All supported device families | Choose between **Full Calibration** (long simulation time), **Quick Calibration**, or **Skip Calibration**.<br>For more information, refer to the *Simulation* section in volume 4 of the *External Memory Interface Handbook*. |

## Board Settings

Click **Next** or the **Board Settings** tab to set the options described in Table 3–7. The board settings parameters are set to model the board level effects in the timing analysis. The options are available if you choose Arria II GX or Stratix IV device for your interface. Otherwise, the options are disabled.

**Table 3–7. ALTMEMPHY Board Settings**

| Parameter Name | Units | Description |
|---|---|---|
| Number of slots/discrete devices | — | Sets the single-rank or multi-rank configuration. |
| CK/CK# slew rate (differential) | V/ns | Sets the differential slew rate for the CK and CK# signals. |
| Addr/command slew rate | V/ns | Sets the slew rate for the address and command signals. |
| DQ/DQS# slew rate (differential) | V/ns | Sets the differential slew rate for the DQ and DQS# signals. |
| DQ slew rate | V/ns | Sets the slew rate for the DQ signals. |
| Addr/command eye reduction (setup) | ns | Sets the reduction in the eye diagram on the setup side due to the ISI on the address and command signals. |
| Addr/command eye reduction (hold) | ns | Sets the reduction in the eye diagram on the hold side due to the ISI on the address and command signals. |
| DQ eye reduction | ns | Sets the total reduction in the eye diagram on the setup side due to the ISI on the DQ signals. |
| Delta DQS arrival time | ns | Sets the increase of variation on the range of arrival times of DQS due to ISI. |
| Min CK/DQS skew to DIMM | ns | Sets the minimum skew between the CK signal and any DQS signal when arriving at the same DIMM over all DIMMs. |
| Max CK/DQS skew to DIMM | ns | Sets the maximum skew between the CK signal and any DQS signal when arriving at the same DIMM over all DIMMs. |
| Max skew between DIMMs/devices | ns | Sets the largest skew or propagation delay on the DQ signals between ranks, especially true for DIMMs in different slots. |
| Max skew within DQS group | ns | Sets the largest skew between the DQ pins in a DQS group. |
| Max skew between DQS groups | ns | Sets the largest skew between DQS signals in different DQS groups. |
| Addr/command to CK skew | ns | Sets the skew or propagation delay between the CK signal and the address and command signals. The positive values represent the address and command signals that are longer than the CK signals, and the negative values represent the address and command signals that are shorter than the CK signals. |

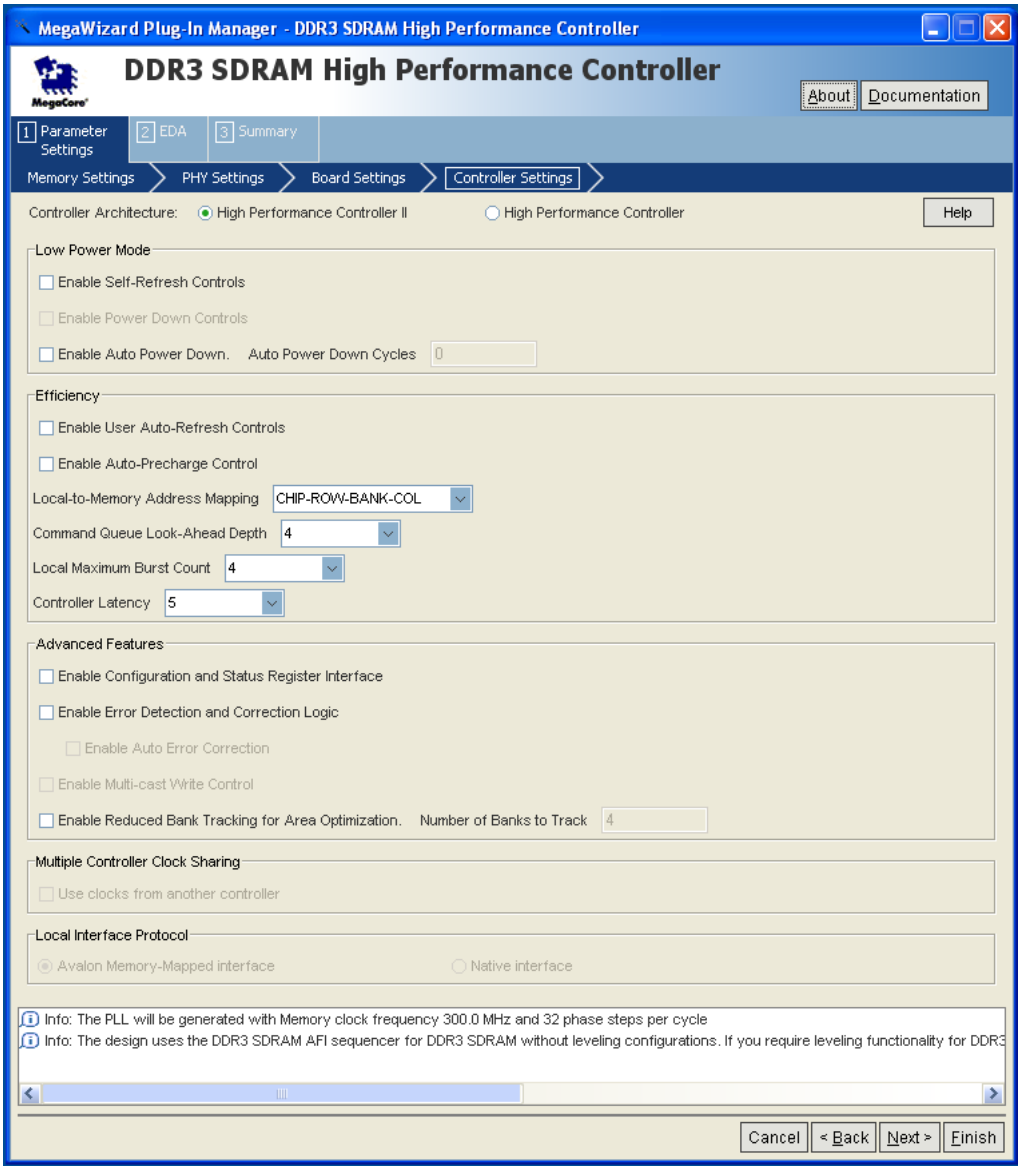
# DDR3 SDRAM Controller with ALTMEMPHY Parameter Settings

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

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

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

**Figure 3–3. DDR3 SDRAM Controller with ALTMEMPHY Settings**




## Controller Settings

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

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

| Parameter | Controller Architecture | Description |
|---|---|---|
| Controller architecture | — | Specifies the controller architecture. |
| Enable self-refresh controls | Both | Turn on to enable the controller to allow you to have control on when to place the external memory device in self-refresh mode, refer to "User-Controlled Self-Refresh Logic" on page 7–8 (HPC II). |
| Enable power down controls | HPC | Turn on to enable the controller to allow you to have control on when to place the external memory device in power-down mode. |
| Enable auto power down | HPC II | Turn on to enable the controller to automatically place the external memory device in power-down mode after a specified number of idle controller clock cycles is observed in the controller. You can specify the number of idle cycles after which the controller powers down the memory in the **Auto Power Down Cycles** field, refer to "Automatic Power-Down with Programmable Time-Out" on page 7–7. |
| Auto power down cycles | HPC II | Determines the desired number of idle controller clock cycles before the controller places the external memory device in a power-down mode. The legal range is 1 to 65,535.<br>The auto power-down mode is disabled if you set the value to 0 clock cycles. |
| Enable user auto-refresh controls | Both | Turn on to enable the controller to allow you to issue a single refresh. |
| Enable auto-precharge control | HPC | Turn on to enable the auto-precharge control on the controller top level. Asserting the auto-precharge control signal while requesting a read or write burst allows you to specify whether or not the controller should close (auto-precharge) the current opened page at the end of the read or write burst. |
| Local-to-memory address mapping | HPC II | Allows you to control the mapping between the address bits on the Avalon interface and the chip, row, bank, and column bits on the memory interface.<br>If your application issues bursts that are greater than the column size of the memory device, choose the Chip-Row-Bank-Column option. This option allows the controller to use its look-ahead bank management feature to hide the effect of changing the currently open row when the burst reaches the end of the column.<br>On the other hand, if your application has several masters that each use separate areas of memory, choose the Chip-Bank-Row-Column option. This option allows you to use the top address bits to allocate a physical bank in the memory to each master. The physical bank allocation avoids different masters accessing the same bank which is likely to cause inefficiency, as the controller must then open and close rows in the same bank. |
| Command queue look-ahead depth | HPC II | Specifies a command queue look-ahead depth value to control the number of read or write requests the look-ahead bank management logic examines, refer to "Command Queue" on page 7–4. |

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

| Parameter | Controller Architecture | Description |
|---|---|---|
| Local maximum burst count | HPC II | Specifies a burst count to configure the maximum Avalon burst count that the controller slave port accepts. |
| Controller latency | HPC II | Specifies a latency for the controller. The default latency is **5** but you have the option to choose **4** to enhance the latency performance of your design at the expense of timing closure. |
| Enable configuration and status register interface | HPC II | Turn on to enable run-time configuration and status retrieval of the memory controller. Enabling this option adds an additional Avalon-MM slave port to the memory controller top level that allows run-time reconfiguration and status retrieving for memory timing parameters, memory address size and mode register settings, and controller features. If the **Error Detection and Correction Logic** option is enabled, the same slave port also allows you to control and retrieve the status of this logic. Refer to "Configuration and Status Register (CSR) Interface" on page 7–7. |
| Enable error detection and correction logic | Both | Turn on to enable error correction coding (ECC) for single-bit error correction and double-bit error detection. Refer to "Error Correction Coding (ECC)" on page 6–5 for HPC, and "Error Correction Coding (ECC)" on page 7–7 for HPC II. |
| Enable auto error correction | HPC II | Turn on to allow the controller to perform auto correction when the ECC logic detects a single-bit error. Alternatively, you can turn off this option and schedule the error correction at a desired time for better system efficiency. Refer to "Error Correction Coding (ECC)" on page 7–7. |
| Enable multi-cast write control | HPC II | Turn on to enable the multi-cast write control on the controller top level. Asserting the multi-cast write control when requesting a write burst causes the write data to be written to all the chip selects in the memory system. When you turn on this option together with the **Enable User Auto-Refresh Controls** option, the user refresh commands are issued to all chips.<br>Multi-cast write is not supported for registered DIMM interfaces or when you turn on the **Enable Error Detection and Correction Logic** option. |
| Enable reduced bank tracking for area optimization | HPC II | Turn on to reduce the controller's resource usage. By turning on this option, you reduce the number of bank tracking blocks in the controller. Refer to "Bank Management Logic" on page 7–4 for more information. |
| Number of banks to track | HPC II | Specifies the number of bank tracking blocks you want for your design. This option is only available if you turn on the **Enable Reduced Bank Tracking for Area Optimization** option. The value for this option depends on the value you specify for the **Command Queue Look-Ahead Depth** option. Refer to "Bank Management Logic" on page 7–4 for more information. |

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

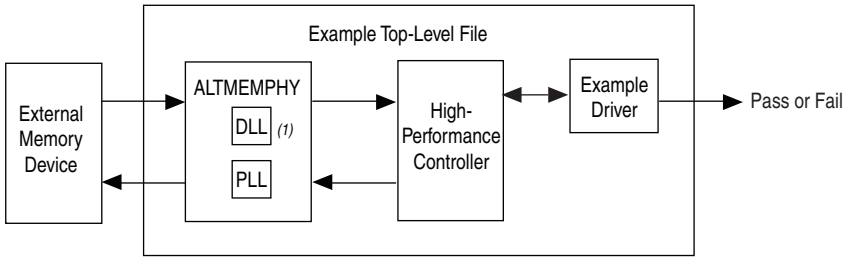
| Parameter | Controller Architecture | Description |
|---|---|---|
| Multiple controller clock sharing | Both | This option is only available in SOPC Builder Flow. Turn on to allow one controller to use the Avalon clock from another controller in the system that has a compatible PLL. This option allows you to create SOPC Builder systems that have two or more memory controllers that are synchronous to your master logic. |
| Local interface protocol | HPC | Specifies the local side interface between the user logic and the memory controller. The Avalon-MM interface allows you to easily connect to other Avalon-MM peripherals.<br>The HPC II architecture supports only the Avalon-MM interface. |

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

## Compiling the Design

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

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




**Note to Figure 4–1:**

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

Before compiling a design with the ALTMEMPHY variation, you must edit some project settings, include the **.sdc** file, and make I/O assignments. I/O assignments include I/O standard, pin location, and other assignments, such as termination and drive strength settings. Some of these tasks are listed in the ALTMEMPHY **Generation** window. For most systems, Altera recommends that you use the **Advanced I/O Timing** feature by using the **Board Trace Model** command in the Quartus II software to set the termination and output pin loads for the device.

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

1. Set up the TimeQuest timing analyzer:
   a. On the Assignments menu, click **Timing Analysis Settings**, select **Use TimeQuest Timing Analyzer during compilation**, and click **OK**.
   b. Add the Synopsys Design Constraints (**.sdc**) file, *<variation name>***_phy_ddr_timing.sdc**, to your project. On the Project menu, click **Add/Remove Files in Project** and browse to select the file.
   c. Add the **.sdc** file for the example top-level design, *<variation name>***_example_top.sdc,** to your project. This file is only required if you are using the example as the top-level design.

2. You can either use the <*variation_name*>**_pin_assignments.tcl** or the <*variation_name*>**.ppf** file to apply the I/O assignments generated by the MegaWizard Plug-In Manager. Using the **.ppf** file and the Pin Planner gives you the extra flexibility to add a prefix to your memory interface pin names. You can edit the assignments either in the Assignment Editor or Pin Planner. Use one of the following procedures to specify the I/O standard assignments for pins:

- If you have a single SDRAM interface, and your top-level pins have default naming shown in the example top-level file, run <*variation name*>**_pin_assignments.tcl**.

  or

- If your design contains pin names that do not match the design, edit the <*variation name*>**_pin_assignments.tcl** file before you run the script. To edit the **.tcl** file, perform the following steps:

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

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

     - SOPC Builder System flow:

     ```
     if {![info exists sopc_mode]} {set sopc_mode YES}
     ```

     - MegaWizard Plug-In Manager flow:

     ```
     if {![info exists sopc_mode]} {set sopc_mode NO}
     ```

  c. Type your preferred prefix in the `pin_prefix` variable. For example, to add the prefix `my_mem`, do the following:

     ```
     if {![info exists set_prefix]}{set pin_prefix "my_mem_"}
     ```

     After setting the prefix, the pin names are expanded as shown in the following:

     - SOPC Builder System flow:

     `my_mem_cs_n_from_the_`<*your instance name*>

     - MegaWizard Plug-In Manager flow:

     ```
     my_mem_cs_n[0]
     ```

     If your top-level design does not use single bit bus notation for the single-bit memory interface signals (for example, `mem_dqs` rather than `mem_dqs[0]`), in the Tcl script you should change `set single_bit {[0]}` to `set single_bit {}`.

     or

- Alternatively, to change the pin names that do not match the design, you can add a prefix to your pin names by performing the following steps:
  a. On the Assignments menu, click **Pin Planner**.
  b. On the Edit menu, click **Create/Import Megafunction**.
  c. Select **Import an existing custom megafunction** and navigate to *<variation name>***.ppf**.
  d. Type the prefix you want to use in **Instance name**. For example, change **mem_addr** to **core1_mem_addr**.

3. Set the top-level entity to the top-level design.
   a. On the File menu, click **Open**.
   b. Browse to your SOPC Builder system top-level design or *<variation name>***_example_top** if you are using MegaWizard Plug-In Manager, and click **Open**.
   c. On the Project menu, click **Set as Top-Level Entity**.
4. Assign the DQ and DQS pin locations.
   a. You should assign pin locations to the pins in your design, so the Quartus II software can perform fitting and timing analysis correctly.
   b. Use either the Pin Planner or Assignment Editor to assign the clock source pin manually. Also choose which DQS pin groups should be used by assigning each DQS pin to the required pin. The Quartus II Fitter then automatically places the respective DQ signals onto suitable DQ pins within each group.

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

   The ×4 DIMM has the following mapping between DQS and DQ pins:
   - DQS[0] maps to DQ[3:0]
   - DQS[1] maps to DQ[7:4]
   - DQS[2] maps to DQ[11:8]
   - DQS[3] maps to DQ[15:12]

   The DQS pin index in other ×4 DIMM configurations typically increases sequentially with the DQ pin index (DQS[0]: DQ[3:0]; DQS[1]: DQ[7:4]; DQS[2]: DQ[11:8])
5. For Stratix III and Stratix IV designs, if you are using advanced I/O timing, specify board trace models in the **Device & Pin Options** dialog box. If you are using any other device and not using advanced I/O timing, specify the output pin loading for all memory interface pins.

6. Select your required I/O driver strength (derived from your board simulation) to ensure that you correctly drive each signal or ODT setting and do not suffer from overshoot or undershoot.
7. To compile the design, on the Processing menu, click **Start Compilation**.

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

## Simulating the Design

During system generation, SOPC Builder optionally generates a simulation model and testbench for the entire system, which you can use to easily simulate your system in any of Altera's supported simulation tools. The MegaWizard also generates a set of ModelSim® Tcl scripts and macros that you can use to compile the testbench, IP functional simulation models, and plain-text RTL design files that describe your system in the ModelSim simulation software (refer to "Generated Files" on page 2–6).

For more information about simulating SOPC Builder systems, refer to volume 4 of the *Quartus II Handbook* and *AN 351: Simulating Nios II Embedded Processor Designs*. For more information about how to include your board simulation results in the Quartus II software and how to assign pins using pin planners, refer to *ALTMEMPHY Design Tutorials* section in volume 6 of the *External Memory Interface Handbook*.

In ALTMEMPHY variations for DDR3 SDRAM with leveling and without leveling interfaces, you have the following three simulation options:

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

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

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

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

  The additive latency and registered DIMMs must be disabled.

  Skip calibration is unavailable for DDR3 RDIMMs.

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

  Available for ×4 and ×8 DDR3 SDRAM. In quick calibration simulation mode, the sequencer only does clock cycle calibration. So there must be no delays (DDR3 DIMM modeling for example) in the testbench, because no phase calibration is performed. Quick calibration mode can be used between 300 MHz and 533 MHz. Both the generated or vendor DDR3 SDRAM simulation models support burst length on-the-fly changes during the calibration sequence.

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

  Available for ×4 and ×8 DDR3 SDRAM between 300 MHz and 533 MHz. You cannot use the wizard-generated memory model, if you select **Full Calibration**. You must use a memory-vendor provided memory model that supports write leveling calibration.

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

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

# 5. Functional Description—ALTMEMPHY

The ALTMEMPHY megafunction creates the datapath between the memory device and the memory controller, and user logic in various Altera devices. The ALTMEMPHY megafunction GUI helps you configure multiple variations of a memory interface. You can then connect the ALTMEMPHY megafunction variation with either a user-designed controller or with an Altera high-performance controller. In addition, the ALTMEMPHY megafunction and the Altera high-performance controllers are available for half-rate DDR3 SDRAM interfaces.

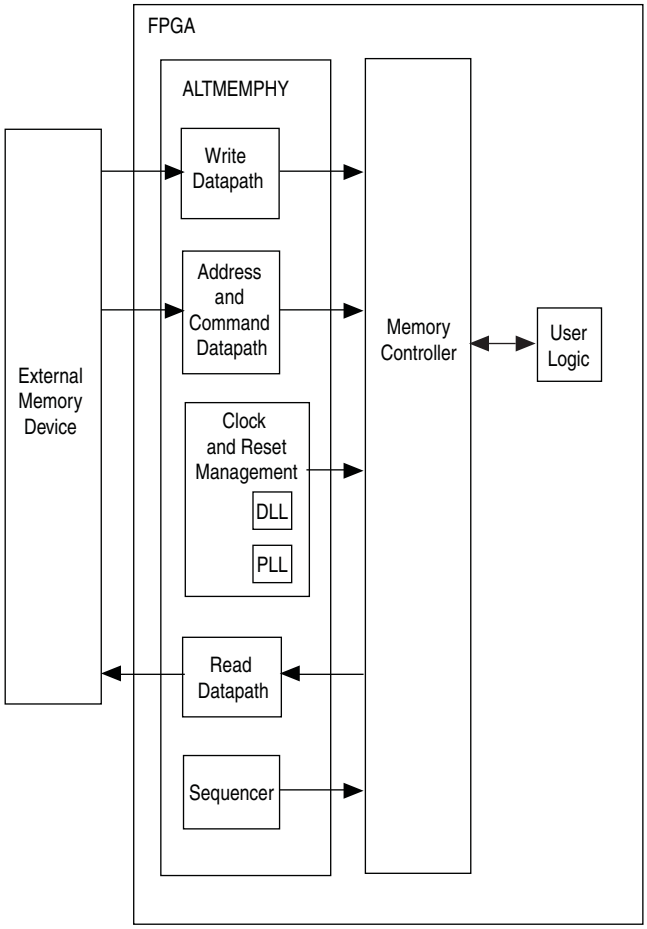
If the ALTMEMPHY megafunction does not meet your requirements, you can also create your own memory interface datapath using the ALTDLL and ALTDQ_DQS megafunctions, available in the Quartus II software. However, you are then responsible for every aspect of the interface, including timing analysis and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

This chapter describes the DDR3 SDRAM ALTMEMPHY megafunction, which uses AFI as the interface between the PHY and the controller.

# Block Description

Figure 5–1 on page 5–2 shows the major blocks of the ALTMEMPHY megafunction and how it interfaces with the external memory device and the controller. The ALTPLL megafunction is instantiated inside the ALTMEMPHY megafunction, so that you do not need to generate the clock to any of the ALTMEMPHY blocks.

**Figure 5–1. ALTMEMPHY Megafunction Interfacing with the Controller and the External Memory**




The ALTMEMPHY megafunction comprises the following blocks:

- Write datapath
- Address and command datapath
- Clock and reset management, including DLL and PLL
- Sequencer for calibration
- Read datapath

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

## Calibration

This section describes the calibration that the sequencer performs, to find the optimal clock phase for the memory interface. The calibration sequence is similar across families, but different depending on the following target memory interface:

- DDR3 SDRAM Without Leveling
- DDR3 SDRAM With Leveling

### DDR3 SDRAM Without Leveling

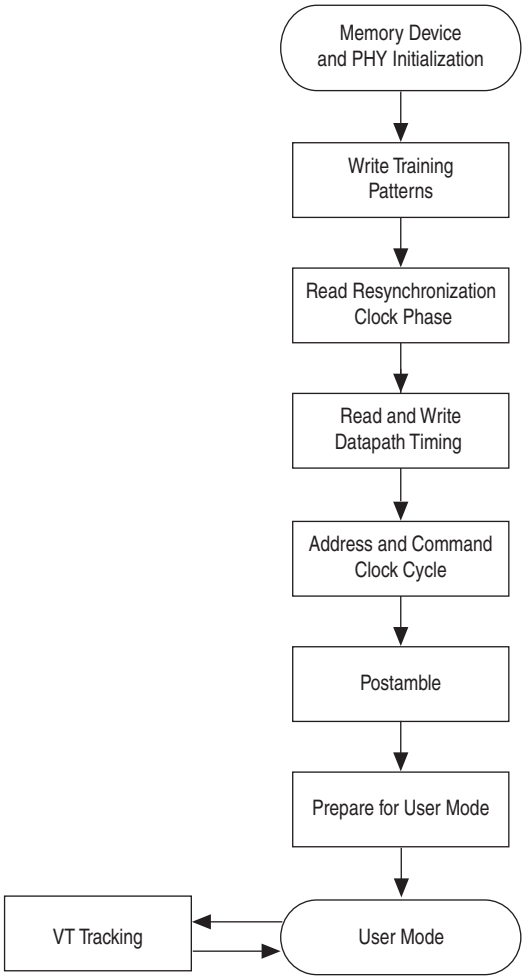
The calibration process for the DDR3 SDRAM without leveling PHY includes the following steps:

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

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

Figure 5–2 shows the calibration flow.

**Figure 5–2. Calibration Flow—Without Leveling**




## Step 1: Memory Device Initialization

This step initializes the memory device according to the DDR3 SDRAM specification. The initialization procedure includes resetting the memory device, specifying the mode registers and memory device ODT setting, and initializing the memory device DLL. Calibration requires overriding some of the user-specified mode register settings, which are reverted in "Step 7: Prepare for User Mode".

## Step 2: Write Training Patterns

In this step, a pattern is written to the memory to be read in later calibration stages. The matched trace lengths to DDR3 SDRAM devices mean that after memory initialization, write capture functions. The pattern is `0x30F5` and comprises the following separately written patterns:

- All `0`: `'b0000` - DDIO high and low bits held at `0`
- All `1`: `'b1111` - DDIO high and low bits held at `1`
- Toggle: `'b0101` - DDIO high bits held at `0` and DDIO low bits held at `1`

- Mixed: `'b0011` - DDIO high and low bits have to toggle

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

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

### Step 3: Read Resynchronization (Capture) Clock Phase

This step adjusts the phase of the resynchronization clock to determine the optimal phase that gives the greatest margin. The resynchronization clock captures the outputs of DQS capture registers (DQS is the capture clock).

To correctly calibrate resynchronization clock phase, based on a data valid window, requires 720° of phase sweep.

### Step 4: Read and Write Datapath Timing

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

### Step 5: Address and Command Clock Cycle

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

### Step 6: Postamble

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

### Step 7: Prepare for User Mode

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

#### VT Tracking

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

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

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

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

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

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

### Mimic Path

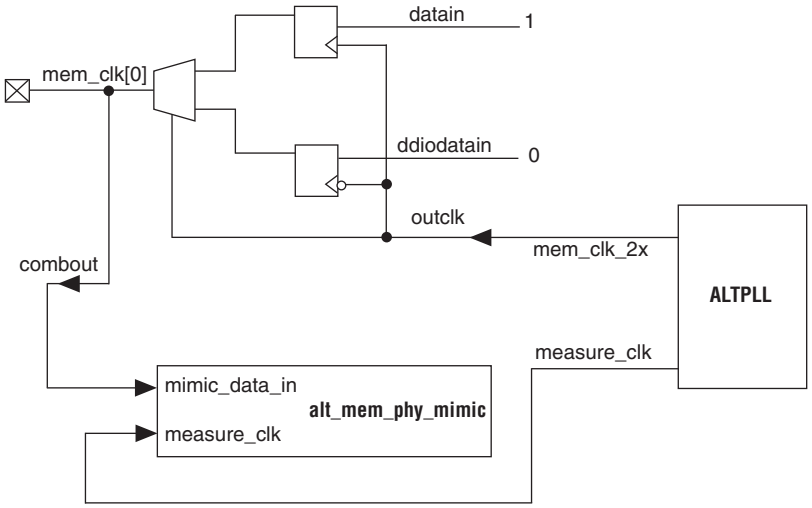
The mimic path mimics the FPGA elements of the round-trip delay, which enables the calibration sequencer to track delay variation due to VT changes during the memory read and write transactions without interrupting the operation of the ALTMEMPHY megafunction.

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

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

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

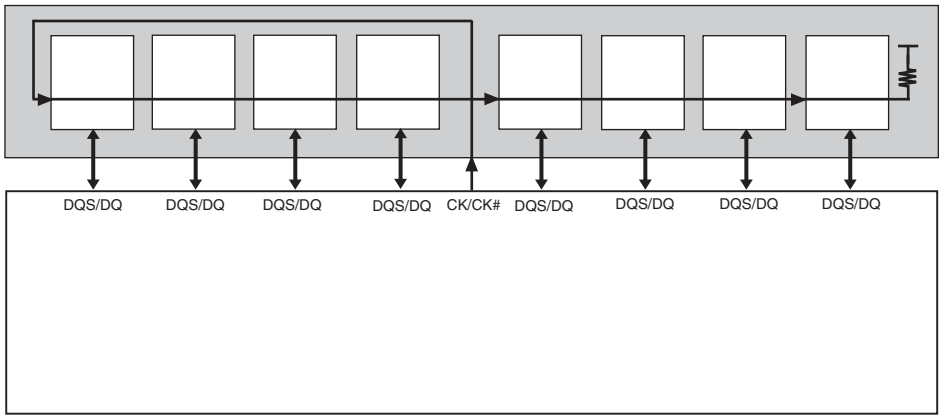
**Figure 5–3. Mimic Path**




## DDR3 SDRAM With Leveling

The calibration process for the DDR3 SDRAM PHY (with leveling) assumes an interface in an unbuffered DIMM format, where the clock uses a fly-by termination, refer to Figure 5–4.

**Figure 5–4. DDR3 SDRAM Unbuffered Module Clock Topology**




With fly-by termination, each DDR3 SDRAM device on the DIMM sees the CK/CKn edges at different times. Therefore, the sequencer must adjust the clock to launch the DQS/DQSn and DQ signals so that it is appropriately aligned to the CK/CKn signals on each device.

The DDR3 SDRAM leveling sequencer during calibration writes to the following locations:

- Banks 0, 1, and 2
- Row 0
- All columns

Bank 0 is written to for the block training pattern and clock cycle calibration (`DQ_1T` and `AC_1T`). Bank 1 is written to for write deskew (DQ). Bank 2 is written to for write deskew (DM). For each bank, only row 0 is accessed. The number of columns accessed can vary, but you should avoid writing to all columns in these banks and row 0.

The calibration process for the DDR3 SDRAM PHY with leveling includes the following steps:

- "Step 1: Memory Device Initialization"
- "Step 2: Write Leveling"
- "Step 3: Write Training Patterns"
- "Step 4: Read Resynchronization"
- "Step 5: Address and Command Path Clock Cycle"
- "Step 7: Write Clock Path Setup"
- "Step 8: Prepare for User Mode"

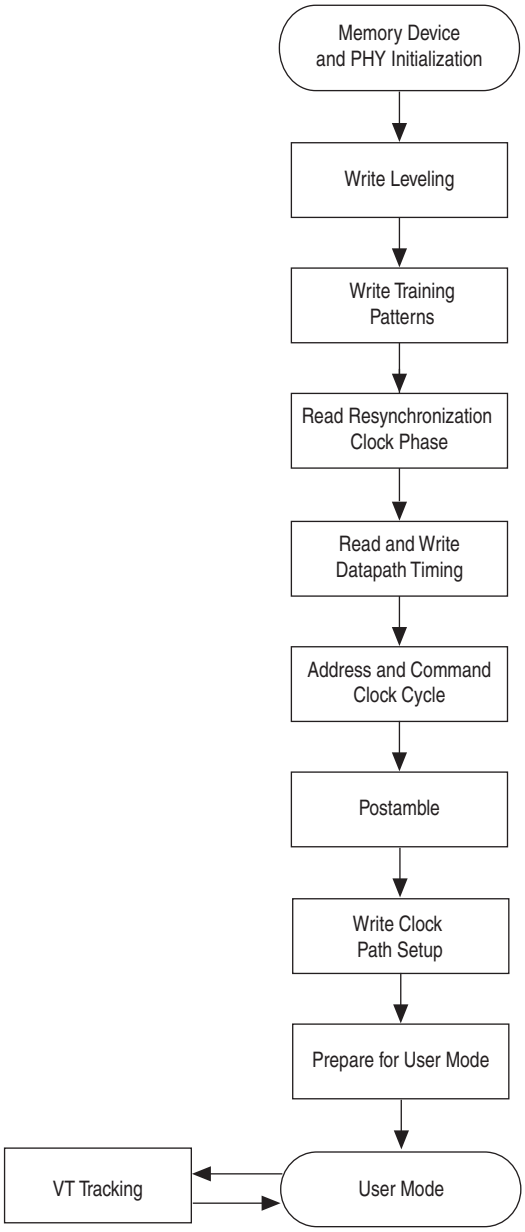
No steps can be bypassed. Therefore, even if you are using only one DDR3 SDRAM DIMM, all the calibration sequences are performed.

The calibration assumes that the skew for all the DQS launch times is one clock period maximum.

The VT tracking portion of the DDR3 SDRAM sequencer is similar to that of the DDR or DDR2 SDRAM sequencer.

Figure 5–5 shows the calibration flow.

**Figure 5–5. Calibration Flow—DDR3 SDRAM (with leveling)**




## Step 1: Memory Device Initialization

This step initializes the memory device per the DDR3 SDRAM specification. The initialization procedure includes resetting the memory device, specifying the mode registers and memory device ODT setting, and initializing the memory device DLL. Some of these settings may be different to those you set; however, these are changed to the correct values (at the end of calibration) in "Step 8: Prepare for User Mode".

On multiple rank DDR3 SDRAM DIMMs, address signals are routed differently to each rank (referred to in the JEDEC specification as address mirroring). Ranks with address mirroring can be specified in the memory Preset Editor in the **Mirror addressing** field.

RTL simulation of address mirroring is not currently supported by the memory model generated with the example testbench. To simulate successfully, you need a DDR3 DIMM model compatible with address mirroring.

### Step 2: Write Leveling

This step aligns the DQS edge with the CK edge at each memory device in the DIMM, which includes calibrating the write-leveling delay chains, programmable output delay chain, and using the $t_{DQSS}$-margin register in the DDR3 SDRAM to monitor the relationship between the DQS edge and the CK edge. The calibration uses one DQ pin per DQS group (prime DQ) for write leveling calibration.

### Step 3: Write Training Patterns

This step only allows you to write a pattern to be read later to calibrate the read path.

To satisfy the DDR3 SDRAM JEDEC specification, DQ is held constant during this step to ensure that there are no DQ timing violations. The DQS is then toggled, followed by a write command. A combination of burst length of four and burst length of eight operations ensure that it is correctly written. There are three different DQ patterns written in this step:

- All 0: 0×00, 0×00, 0×00, 0×00, 0×00, 0×00, 0×00, 0×00
- All 1: 0×FF, 0×FF, 0×FF, 0×FF, 0×FF, 0×FF, 0×FF, 0×FF
- Mixed: 0×00, 0×00, 0×00, 0×00, 0×FF, 0×FF, 0×FF, 0×FF

### Step 4: Read Resynchronization

This step adjusts the phase of the resynchronization clock to determine the optimal clock phase that gives the most margin, similar to the resynchronization calibration done in DDR and DDR2 SDRAM PHYs.

This step uses the read-leveling delay chain and the PLL reconfigurable clock output to adjust the resynchronization clock phase for each DQS group.

### Step 5: Address and Command Path Clock Cycle

This step word-aligns the read data within and between each DQS group so that data can be presented in one clock cycle.

### Step 6: Postamble

This step sets the correct clock cycle and clock phase shift for the postamble path. With the read resynchronization process, the sequencer can approximate when the postamble enable must be asserted. The sequencer then tries to incrementally assert the postamble enable signal (per DQS group) earlier until there is a read failure. This ensures the optimal clock phase for the system's postamble enable signal.

### Step 7: Write Clock Path Setup

After the sequencer has the optimum settings for read capture and resynchronization setup, the sequencer calibrates the write datapath by configuring the alignment registers in the IOE and the DQ and DQS phase shift per DQS group. This step ensures that the write data can be presented on the same clock cycle from controller, but launched at the appropriate time for each DQS group to the DDR3 SDRAM memory devices.

### Step 8: Prepare for User Mode

In this step, the sequencer sends the calibrated write latency between command and write data (the `ctl_wlat` signal) to the controller. The PHY then applies user mode register settings and performs setup for periodic VT tracking.

Deskew is automatically enabled above 400.000 MHz.

### VT Tracking

For information on VT tracking for DDR3 SDRAM with leveling, refer to "VT Tracking" on page 5–5.

### Mimic Path

For information on mimic path for DDR3 SDRAM with leveling, refer to "Mimic Path" on page 5–6.

## Address and Command Datapath

This topic discusses the address and command datapath.

### Arria II GX Devices

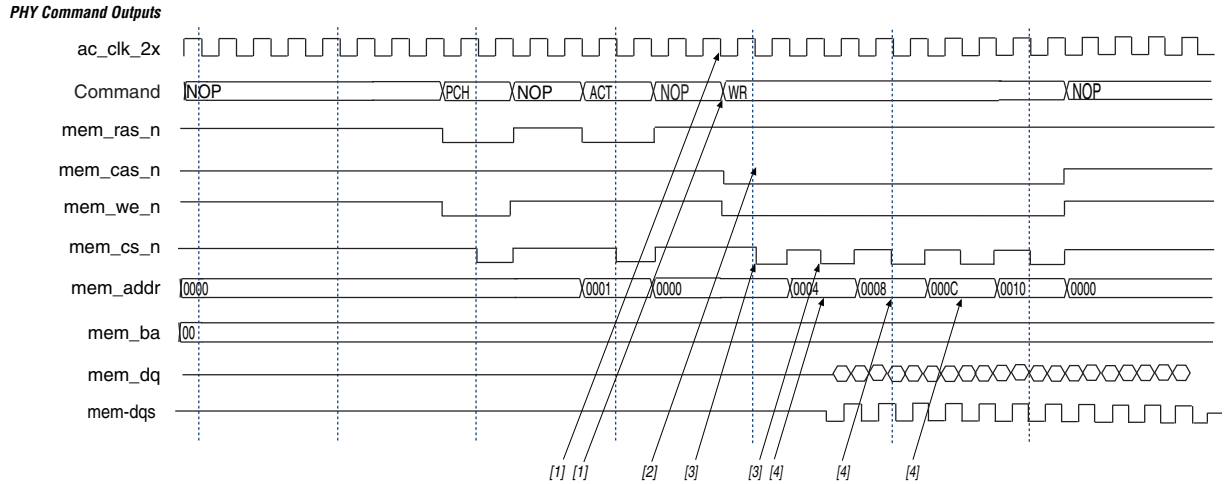
The address and command datapath is responsible for taking the address and command outputs from the controller and converting them from half-rate clock to full-rate clock. Two types of addressing are possible:

- 1T (full rate)—the duration of the address and command is a single memory clock cycle (`mem_clk_2x`, Figure 5–6). This applies to all address and command signals in full-rate designs or `mem_cs_n`, `mem_cke`, and `mem_odt` signals in half-rate designs.
- 2T (half rate)—the duration of the address and command is two memory clock cycles. For half-rate designs, the ALTMEMPHY megafunction supports only a burst size of four, which means the burst size on the local interface is always set to 1. The size of the data is 4*n*-bits wide on the local side and is *n*-bits wide on the memory side. To transfer all the 4*n*-bits at the double data rate, two memory-clock cycles are required. The new address and command can be issued to memory every two clock cycles. This scheme applies to all address and command signals, except for `mem_cs_n`, `mem_cke`, and `mem_odt` signals in half-rate mode.

Refer to Table 5–1 on page 5–14 to see the frequency relationship of `mem_clk_2x` with the rest of the clocks.

Figure 5–6 shows a 1T chip select signal (mem_cs_n), which is active low, and disables the command in the memory device. All commands are masked when the chip-select signal is inactive. The mem_cs_n signal is considered part of the command code.

**Figure 5–6. Arria II GX Address and Command Datapath**




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

The waveform in Figure 5–6 shows a NOP command followed by five back-to-back write commands. The following sequence corresponds with the numbered items in Figure 5–6.

1. The commands are asserted either on the rising edge of ac_clk_2x. The ac_clk_2x is derived from either mem_clk_2x (0°), write_clk_2x (270°), or the inverted variations of those two clocks (for 180° and 90° phase shifts). This depends on the setting of the address and command clock in the ALTMEMPHY parameter editor. Refer to "Address and Command Datapath" on page 5–11 for illustrations of this clock in relation to the mem_clk_2x or write_clk_2x signals.
2. All address and command signals (except for mem_cs_ns, mem_cke, and mem_odt signals) remain asserted on the bus for two clock cycles, allowing sufficient time for the signals to settle.
3. The mem_cs_n, mem_cke, and mem_odt signals are asserted during the second cycle of the address/command phase. By asserting the chip-select signal in alternative cycles, back-to-back read or write commands can be issued.
4. The address is incremented every other ac_clk_2x cycle.

The ac_clk_2x clock is derived from either mem_clk_2x (when you choose 0° or 180° phase shift) or write_clk_2x (when you choose 90° or 270° phase shift).

The address and command clock can be 0, 90, 180, or 270° from the system clock.

### Stratix III and Stratix IV Devices

The address and command clock is one of the PLL dedicated clock outputs whose phase can be adjusted to meet the setup and hold requirements of the memory clock. The Stratix III address and command clock, `ac_clk_1x`, is half rate. The command and address pins use the DDIO output circuitry to launch commands from either the rising or falling edges of the clock. The chip select (`mem_cs_n`), clock enable (`mem_cke`), and `mem_odt` pins are enabled on one memory clock cycle basis and can be launched from either the rising or falling edge of the `ac_clk_1x` signal, while the address and other command pins are enabled for two memory clock cycles and can also be launched from either the rising or falling edge of `ac_clk_1x` signal. It is the responsibility of the controller to maintain the relative timing of the signals.

The DDR3 SDRAM PHY generates a write latency output `ctl_wlat` that indicates the number of `ctl_clk` cycles between the write command being issued, `ctl_cs_n` asserted, and `ctl_dqs_burst` being asserted. This `ctl_wlat` signal is only valid when calibration has been successfully completed by the ALTMEMPHY sequencer and does not change at any point during normal user mode operation. The value on `ctl_wlat` includes the effect of the following as determined during calibration:

- CAS write latency (CWL)
- Additive latency
- Datapath latencies and relative phases
- Board and memory module layout
- Address and command path latency and 1T register setting which is dynamically set up to take into account any leveling effects

## Clock and Reset Management

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

### Clock Management

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

Certain clocks require phase shifting during the ALTMEMPHY megafunction operation.

You can implement clock management circuitry using PLLs and DLLs.

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

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

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

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

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

Cross-device cascading PLLs are only allowed in Stratix III devices with the following conditions:

- Upstream PLL: 0.59 MHz =< upstream PLL bandwidth < 1 MHz. The upstream PLL should use the **With No Compensation** operation mode.
- Downstream PLL: downstream PLL bandwidth > 2 MHz.

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

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

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

| Clock Name *(1)* | Postscale Counter | Phase (Degrees) | Clock Rate | Clock Network Type | Notes |
|---|---|---|---|---|---|
| `phy_clk_1x` and `aux_half_rate_clk` | C0 | 0° | Half-Rate | Global | The only clocks parameterizable for the ALTMEMPHY megafunction. These clocks also feed into a divider circuit to provide the PLL `scan_clk` signal (for reconfiguration) that must be lower than 100 MHz. |
| `mem_clk_2x` and `aux_full_rate_clk` | C1 | 0° | Full-Rate | Global | This clock is for clocking DQS and as a reference clock for the memory devices. |
| `mem_clk_1x` | C2 | 0° | Half-Rate | Global | This clock is for clocking DQS and as a reference clock for the memory devices. |
| `write_clk_2x` | C3 | –90° | Full-Rate | Global | This clock is for clocking the data out of the DDR I/O (DDIO) pins in advance of the DQS strobe (or equivalent). As a result, its phase leads that of the `mem_clk_2x` by 90°. |

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

| Clock Name *(1)* | Postscale Counter | Phase (Degrees) | Clock Rate | Clock Network Type | Notes |
|---|---|---|---|---|---|
| ac_clk_2x | C3 | –90° | Full-Rate | Global | Address and command clock.<br>The ac_clk_2x clock is derived from either mem_clk_2x (when you choose 0° or 180° phase shift) or write_clk_2x (when you choose 90° or 270° phase shift). Refer to "Address and Command Datapath" on page 5–11 for illustrations of the address and command clock relationship with the mem_clk_2x or write_clk_2x signals. |
| cs_n_clk_2x | C3 | –90° | Full-Rate | Global | Memory chip-select clock.<br>The cs_n_clk_2x clock is derived from ac_clk_2x. |
| resync_clk_2x | C4 | Calibrated | Full-Rate | Global | Clocks the resynchronization registers after the capture registers. Its phase is adjusted to the center of the data valid window across all the DQS-clocked DDIO groups. |
| measure_clk_2x | C5 | Calibrated | Full-Rate | Global | This clock is for VT tracking. This free-running clock measures relative phase shifts between the internal clock(s) and those being fed back through a mimic path. As a result, the ALTMEMPHY megafunction can track VT effects on the FPGA and compensate for the effects. |

**Note to Table 5–1:**

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

Table 5–2 shows the PLL outputs and their usage for Stratix III and Stratix IV devices.

**Table 5–2. DDR3 SDRAM Clocking Stratix IV and Stratix III Devices (Part 1 of 2)**

| Clock Name *(1)* | Postscale Counter | Phase (Degrees) | Clock Rate | Clock Network Type | Notes |
|---|---|---|---|---|---|
| `phy_clk_1x` and `aux_half_rate_clk` | C0 | –40° (with leveling) 30° (without leveling) | Half-Rate | Global | The only clock parameterizable for the ALTMEMPHY megafunction. With `phy_clk_1x` the sequencer generates another `sc_clk_dp` clock with this clock that programs the scan chains of the I/O elements. For more information on changing the clock network type, refer to the *ALTMEMPHY Design Tutorials* section in volume 6 of the *External Memory Interface Handbook*. |
| `mem_clk_2x` | C1 | 0 | Full-Rate | Special | Generates `mem_clk` that provides the reference clock for the DLL. A dedicated routing resource exists from the PLL to the DLL, which you select with the regional routing resource for the `mem_clk` using the following attribute in the HDL: `(-name global_signal dual_regional _clock; -to dll~DFFIN -name global_signal off)`. If you use an external DLL, apply this attribute similarly to the external DLL. |
| `aux_full_rate_clk` | C2 | 0° (with leveling) 60° (without leveling) | Full-Rate | None | A copy of `mem_clk_2x` that you can use in other parts of your design. |
| `write_clk_2x` | C3 | 0° (with leveling) –90° (without leveling) | Full-Rate | Regional | This clock feeds the write leveling delay chains that generate the DQ, DM, DQS, and `mem_clk` signals. |
| `resync_clk_2x` | C4 | Calibrated | Full-Rate | Regional | This clock feeds the I/O clock divider that then reads the data out of the DDIO pins. Its phase is adjusted in the calibration process. The design uses an inverted version of this clock for postamble clocking. |
| `measure_clk_1x` | C5 | Calibrated | Half-Rate | Regional *(2)* | This clock is for VT tracking. This free-running clock measures relative phase shifts between the internal clock(s) and those being fed back through a mimic path. As a result, you can track VT effects on the FPGA and compensate for the effects. |

**Table 5–2. DDR3 SDRAM Clocking Stratix IV and Stratix III Devices (Part 2 of 2)**

| Clock Name *(1)* | Postscale Counter | Phase (Degrees) | Clock Rate | Clock Network Type | Notes |
|---|---|---|---|---|---|
| ac_clk_1x | C6 | Set in the GUI | Half-Rate | Regional | Address and command clock. |

**Notes to Table 5–2:**

(1) In full-rate designs a _1x clock may run at full-rate clock rate.

(2) This clock should be of the same clock network clock as the resync_clk_1x clock.

The phase-shift inputs on the PLL perform the PLL reconfiguration. The PLL reconfiguration megafunction is not required.

## Reset Management

Figure 5–8 and Figure 5–9 show the main features of the reset management block for the DDR3 SDRAM PHY. You can use the pll_ref_clk input to feed the optional reset_request_n edge detect and reset counter module. However, this requires the pll_ref_clk signal to use a global clock network resource.

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

**Figure 5–7. ALTMEMPHY Reset Management Block for Arria II GX Devices**

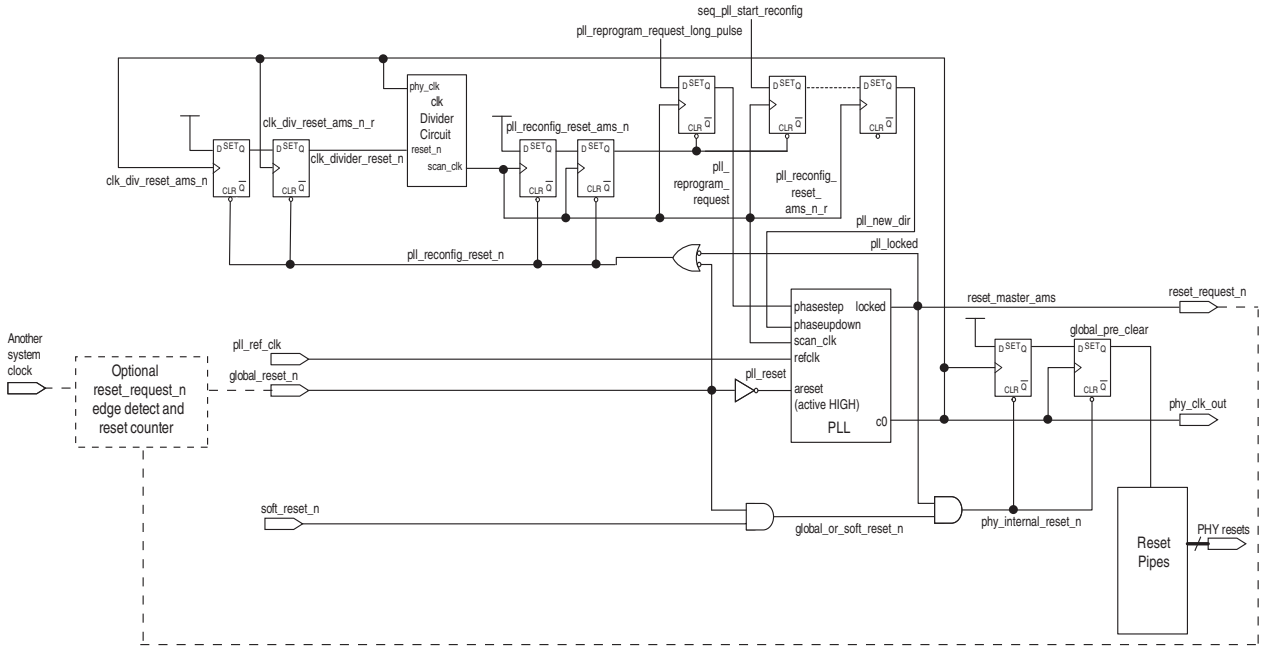
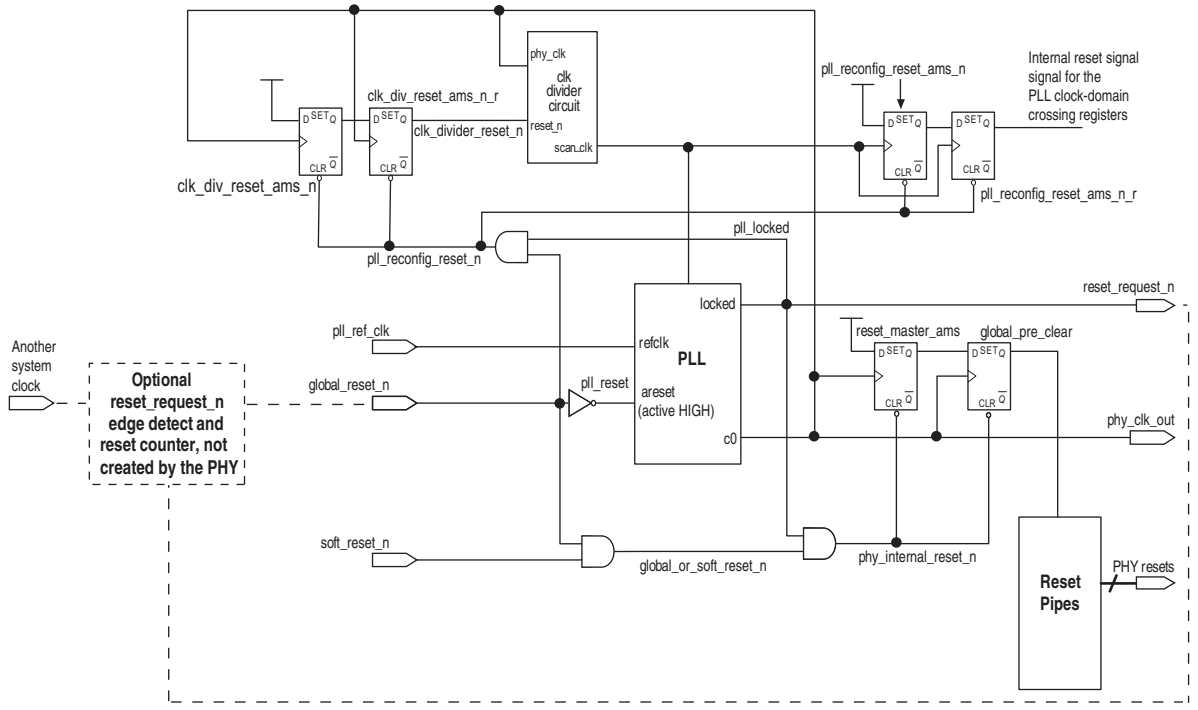


**Figure 5–8. ALTMEMPHY Reset Management Block for Stratix IV and Stratix III Devices**




# Read Datapath

This topic discusses the read datapath.

## Arria II GX Devices

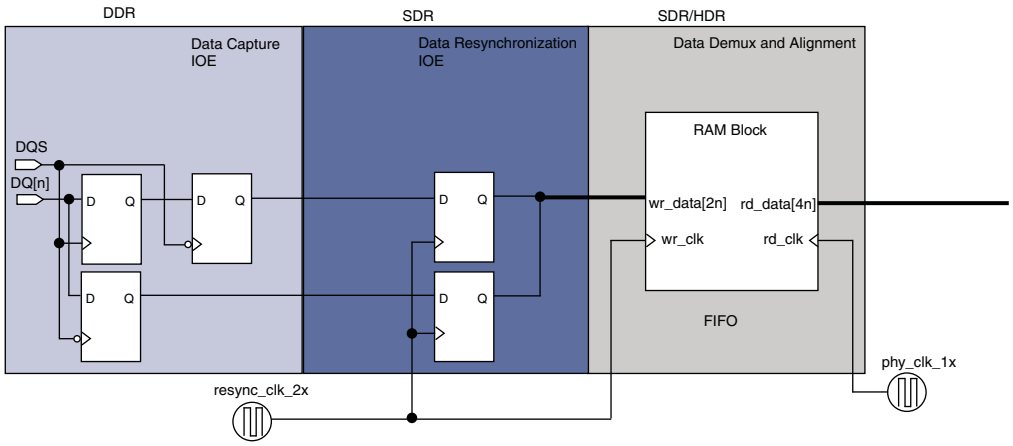
The read datapath logic captures data sent by the memory device and subsequently aligns the data back to the system clock domain. The read datapath for DDR3 SDRAM consists of the following three main blocks:

- Data capture
- Data resynchronization
- Data demultiplexing and alignment

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

Figure 5–9 shows the order of the functions performed by the read datapath and the frequency at which the read data is handled.

**Figure 5–9. DDR3 SDRAM Read Datapath in Arria II GX Devices**




### Data Capture and Resynchronization

The data capture and resynchronization registers for Arria II GX devices are implemented in the I/O element (IOE) to achieve maximum performance. Data capture and resynchronization is the process of capturing the read data (DQ) with the DQS/DQSn strobes and resynchronizing the captured data to an internal free-running full-rate clock supplied by the enhanced PLL. The resynchronization clock is an intermediate clock whose phase shift is determined during the calibration stage. The captured data (`rdata_p_captured` and `rdata_n_captured`) is synchronized to the resynchronization clock (`resync_clk_2x`), refer to Figure 5–9. For Arria II GX devices, the ALTMEMPHY instances an ALTDQ_DQS megafunction that instantiates the required IOEs for all the DQ and DQS pins.

### Data Demultiplexing

Data demultiplexing is the process of changing the SDR data into HDR data. Data demultiplexing is required to bring the frequency of the resynchronized data down to the frequency of the system clock, so that data from the external memory device can ultimately be brought into the FPGA controller clock domain. Before data capture, the data is DDR and *n*-bit wide. After data capture, the data is SDR and 2*n*-bit wide. After data demuxing, the data is HDR of width 4*n*-bits wide. The system clock frequency is half the frequency of the memory clock. Demultiplexing is achieved using a dual-port memory with a 2*n*-bit wide write-port operating on the resynchronization clock (SDR) and a 4*n*-bit wide read-port operating on the PHY clock (HDR). The basic principle of operation is that data is written to the memory at the SDR rate and read from the memory at the HDR rate while incrementing the read- and write-address pointers. As the SDR and HDR clocks are generated, the read and write pointers are continuously incremented by the same PLL, and the 4*n*-bit wide read data follows the 2*n*-bit wide write data with a constant latency

### Read Data Alignment

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

### Postamble Protection

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

## Stratix III and Stratix IV Devices

The DDR3 SDRAM controller asserts `ctl_doing_rd` to indicate that a read command is requested. The `ctl_doing_rd` signal is then used for the following purposes:

- Control of the postamble circuit
- Generation of `ctl_rdata_valid` from one bit to two bits
- Dynamic OCT control timing

The DDR3 SDRAM ALTMEMPHY then asserts the `ctl_rdata_valid` signal to indicate that the data on the read data bus is valid. The `ctl_rdata_valid` signal is two bits wide to allow controllers to issue reads and writes that are aligned to either the half-cycle of the half-rate clock.

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

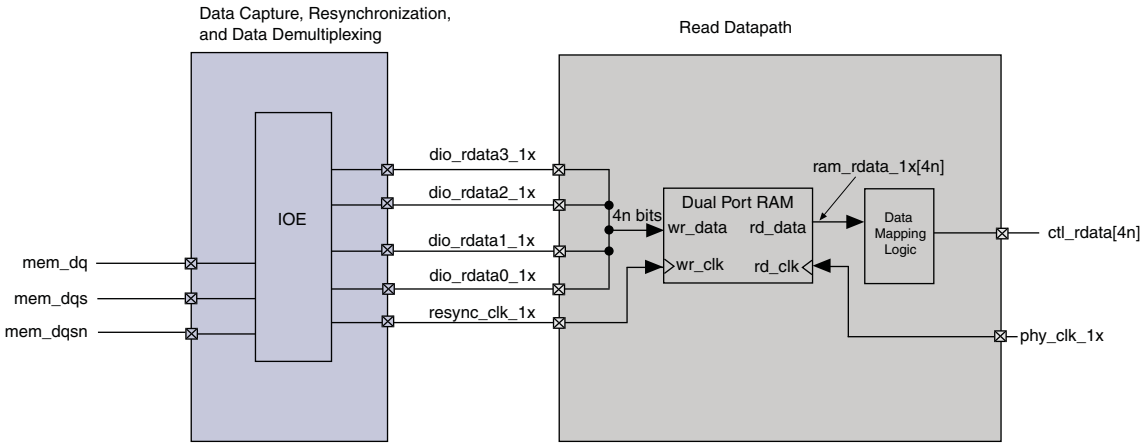
The read datapath for DDR3 SDRAM consists of two main blocks:

- Read data capture, resynchronization, and demultiplexing (in the `dp_io_siii` module)
- Read data alignment logic (in the `read_dp` module) to transfer data from the `resync_clk_2x` (half-rate resynchronization) clock domain to the `phy_clk` clock domain.

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

Figure 5–10 shows the order of the functions performed by the read datapath and the frequency at which the read data is handled.

**Figure 5–10. DDR3 SDRAM Data Capture and Read Data Mapping in Stratix IV and Stratix III Devices**




### Data Capture, Resynchronization, and Demultiplexing

The IOE in Stratix III and Stratix IV devices performs the following tasks during read operation:

- Captures the data
- Resynchronizes the captured data from the DQS domain to the resynchronization clock (`resync_clk_1x`) domain
- Converts the resynchronized data into HDR data

This operation is performed by feeding the resynchronized data into the HDR conversion block within the IOE, which is clocked by the half-rate resynchronization clock (`resync_clk_1x`). The `resync_clk_1x` signal is generated from the I/O clock divider module, based on the `resync_clk_2x` signal from the PLL.

### Read Data Storage Logic

The read block performs the following two tasks:

- Transfers the captured read data (`rdata[n]_1x`) from the half-rate resynchronization clock (`resync_clk_1x`) domain to the half-rate system clock (`phy_clk_1x`) domain using DPRAM. Resynchronized data from the DPRAM is shown as `ram_data_1x`.
- Reorders the resynchronized data (`ram_rdata_1x`) into `ctl_mem_rdata`, to be presented in the user clock domain in the same clock cycle.

### Postamble Protection

A dedicated postamble register controls the gating of the shifted DQS signal that clocks the DQ input registers at the end of a read operation. This ensures that any glitches on the DQS input signals at the end of the read postamble time do not cause erroneous data to be captured as a result of postamble glitches.

The postamble path is also calibrated to determine the correct clock cycle, clock phase shift, and delay chain settings. You can see the process in simulation if you choose **Full calibration** (long simulation time) mode in the MegaWizard Plug-In Manager.

## Write Datapath

This topic discusses the write datapath.

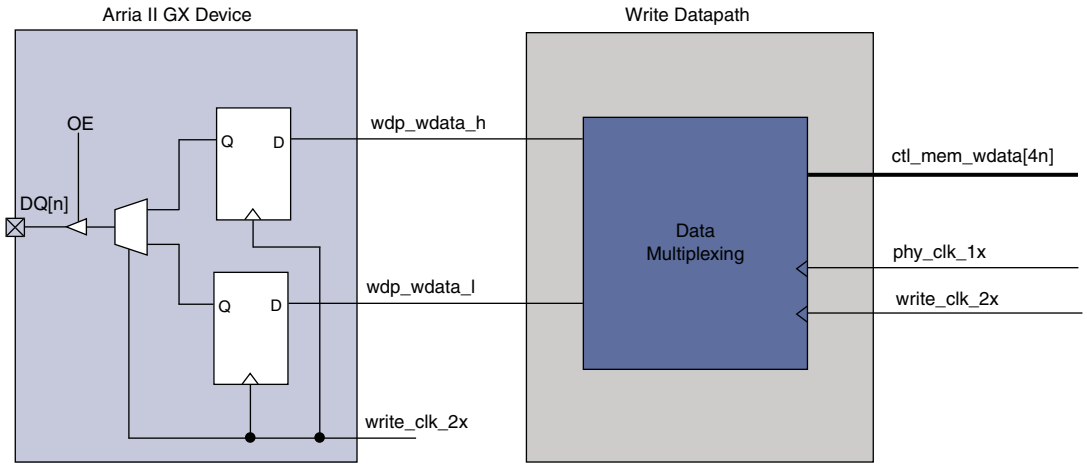
### Arria II GX Devices

The write datapath logic efficiently transfers data from the HDR memory controller to DDR3 SDRAM. The write datapath logic consists of:

- DQ and DQ output-enable logic
- DQS/DQSn and DQS/DQSn output-enable logic
- DM logic

The memory controller interface outputs 4*n*-bit wide data (`ctl_wdata[4n]`) at half-rate frequency. Figure 5-4 shows that the HDR write data (`ctl_wdata[4n]`) is clocked by the half-rate clock `phy_clk_1x` (`ctl_clk`) and is converted into SDR, which is represented by `wdp_wdata_h` and `wdp_wdata_l` and clocked by the full-rate clock `write_clk_2x`. The DQ IOEs convert 2-`n` SDR bits to `n`-DDR bits.

**Figure 5–11. DDR3 SDRAM Write Datapath in Arria II GX Devices**

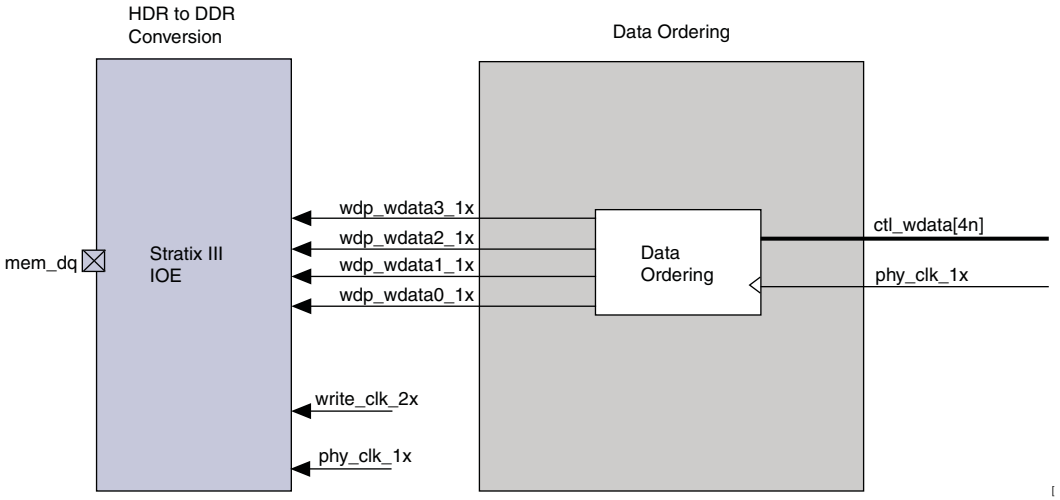



### Stratix III and Stratix IV Devices

The memory controller interface outputs four *n*-bit wide data (`ctl_wdata`) at `phy_clk_1x` frequency. The write data is clocked by the system clock `phy_clk_1x` at half-data rate (HDR) and reordered into HDR of width four with *n*-bits each, represented in Figure 5–12 by `wdp_wdata3_1x`, `wdp_wdata2_1x`, `wdp_wdata1_1x`, and `wdp_wdata0_1x`.

Figure 5–12 shows the reordered or the reordered-and-delayed HDR data is then converted to DDR data within the IOE element using both the half-rate and full-rate clocks.

**Figure 5–12. DDR3 SDRAM Write Datapath in Stratix IV and Stratix III Devices**

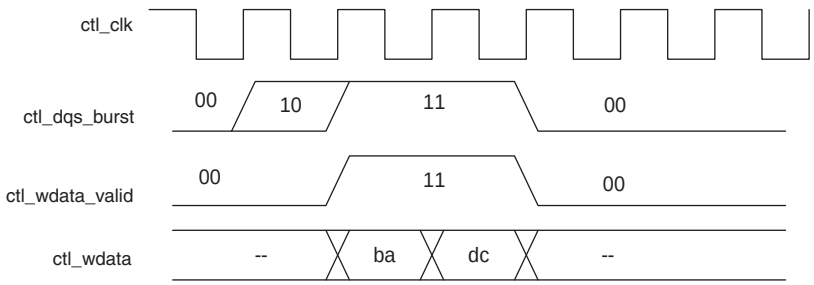



The write datapath DDIO registers are clocked by the `phy_clk_1x` clock. The `write_clk_2x` signal then clocks the alignment registers.

For more information about the I/O structure, refer to the *External Memory Interface* chapter in the respective device family handbook.

Figure 5–13 shows how the write data, `ctl_wdata` signals should be aligned from the controller during a (half rate, normally aligned) write operation. The PHY then issues the write data as ABCD where a is the first data to be written to the memory. (ABCD represent two beats of data each.) The `ctl_wdata_valid` signal in Figure 5–13 shows the output enable for the DQ and DM pins.

**Figure 5–13. Write Data Alignment from the DDR3 SDRAM Controller**




# ALTMEMPHY Signals

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

Table 5–3 through Table 5–5 show the signals.

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

The signal lists include the following signal groups:

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

**Table 5–3. Interface to the DDR3 SDRAM Devices** *(Note 1)*

| Signal Name | Type | Width *(2)* | Description |
|---|---|---|---|
| mem_addr | Output | MEM_IF_ROWADDR_WIDTH | The memory row and column address bus. |
| mem_ba | Output | MEM_IF_BANKADDR_WIDTH | The memory bank address bus. |
| mem_cas_n | Output | 1 | The memory column address strobe. |
| mem_cke | Output | MEM_IF_CS_WIDTH | The memory clock enable. |
| mem_clk | Bidirectional | MEM_IF_CLK_PAIR_COUNT | The memory clock, positive edge clock. *(3)* |
| mem_clk_n | Bidirectional | MEM_IF_CLK_PAIR_COUNT | The memory clock, negative edge clock. |
| mem_cs_n | Output | MEM_IF_CS_WIDTH | The memory chip select signal. |
| mem_dm | Output | MEM_IF_DM_WIDTH | The optional memory DM bus. |
| mem_dq | Bidirectional | MEM_IF_DWIDTH | The memory bidirectional data bus. |
| mem_dqs | Bidirectional | MEM_IF_DWIDTH/ MEM_IF_DQ_PER_DQS | The memory bidirectional data strobe bus. |
| mem_dqs_n | Bidirectional | MEM_IF_DWIDTH/ MEM_IF_DQ_PER_DQS | The memory bidirectional data strobe bus. |
| mem_odt | Output | MEM_IF_CS_WIDTH | The memory on-die termination control signal. |
| mem_ras_n | Output | 1 | The memory row address strobe. |
| mem_reset_n | Output | 1 | The memory reset signal. This signal is derived from the PHY's internal reset signal, which is generated by gating the global reset, soft reset, and the PLL locked signal. |
| mem_we_n | Output | 1 | The memory write enable signal. |
| mem_ac_parity *(4)* | Output | 1 | The address or command parity signal generated by the PHY and sent to the DIMM. |
| parity_error_n *(4)* | Output | 1 | The active-low signal that is asserted when a parity error occurs and stays asserted until the PHY is reset. |

**Table 5–3. Interface to the DDR3 SDRAM Devices *(Note 1)***

| Signal Name | Type | Width *(2)* | Description |
|---|---|---|---|
| mem_err_out_n *(4)* | Input | 1 | The signal sent from the DIMM to the PHY to indicate that a parity error has occured for a particular cycle. |

**Notes to Table 5–3:**

(1) Connected to I/O pads.

(2) Refer to Table 5–6 for parameter description.

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

(4) This signal is for Registered DIMMs only.

**Table 5–4. AFI Signals (Part 1 of 3)**

| Signal Name | Type | Width *(1)* | Description |
|---|---|---|---|
| **Clocks and Resets** | | | |
| pll_ref_clk | Input | 1 | The reference clock input to the PHY PLL. |
| global_reset_n | Input | 1 | Active-low global reset for PLL and all logic in the PHY. A level set reset signal, which causes a complete reset of the whole system. The PLL may maintain some state information. |
| soft_reset_n | Input | 1 | Edge detect reset input intended for SOPC Builder use or to be controlled by other system reset logic. Causes a complete reset of PHY, but not the PLL used in the PHY. |
| reset_request_n | Output | 1 | Directly connected to the locked output of the PLL and is intended for optional use either by automated tools such as SOPC Builder or could be manually ANDed with any other system-level signals and combined with any edge detect logic as required and then fed back to the global_reset_n input.<br>Reset request output that indicates when the PLL outputs are not locked. Use this as a reset request input to any system-level reset controller you may have. This signal is always low while the PLL is locking (but not locked), and so any reset logic using it is advised to detect a reset request on a falling-edge rather than by level detection. |
| ctl_clk | Output | 1 | Half-rate clock supplied to controller and system logic. The same signal as the non-AFI phy_clk. |
| ctl_reset_n | Output | 1 | Reset output on ctl_clk clock domain. |
| **Other Signals** | | | |
| aux_half_rate_clk | Output | 1 | In half-rate designs, a copy of the phy_clk_1x signal that you can use in other parts of your design, same as phy_clk port. |
| aux_full_rate_clk | Output | 1 | In full-rate designs, a copy of the mem_clk_2x signal that you can use in other parts of your design. |

**Table 5–4. AFI Signals (Part 2 of 3)**

| Signal Name | Type | Width *(1)* | Description |
|---|---|---|---|
| aux_scan_clk | Output | 1 | Low frequency scan clock supplied primarily to clock any user logic that interfaces to the PLL and DLL reconfiguration interfaces. |
| aux_scan_clk_reset_n | Output | 1 | This reset output asynchronously asserts (drives low) when global_reset_n is asserted and de-assert (drives high) synchronous to aux_scan_clk when global_reset_n is de-asserted. It allows you to reset any external circuitry clocked by aux_scan_clk. |
| **Write Data Interface** | | | |
| ctl_dqs_burst | Input | MEM_IF_DQS_WIDTH × DWIDTH_RATIO / 2 | When asserted, mem_dqs is driven. The ctl_dqs_burst signal must be asserted before the ctl_wdata_valid signal and must be driven for the correct duration to generate a correctly timed mem_dqs signal. |
| ctl_wdata_valid | Input | MEM_IF_DQS_WIDTH × DWIDTH_RATIO / 2 | Write data valid. Generates ctl_wdata and ctl_dm output enables. |
| ctl_wdata | Input | MEM_IF_DWIDTH × DWIDTH_RATIO | Write data input from the controller to the PHY to generate mem_dq. |
| ctl_dm | Input | MEM_IF_DM_WIDTH × DWIDTH_RATIO | DM input from the controller to the PHY. |
| ctl_wlat | Output | 5 | Required write latency between address/command and write data that is issued to ALTMEMPHY controller local interface.<br>This signal is only valid when the ALTMEMPHY sequencer successfully completes calibration, and does not change at any point during normal operation.<br>The legal range of values for this signal is 0 to 31; and the typical values are between 0 and ten, 0 mostly for low CAS latency DDR memory types. |
| **Read Data Interface** | | | |
| ctl_doing_rd | Input | MEM_IF_DQS_WIDTH × DWIDTH_RATIO / 2 | Doing read input. Indicates that the DDR3 SDRAM controller is currently performing a read operation.<br>The controller generates ctl_doing_rd to the ALTMEMPHY megafunction. The ctl_doing_rd signal is asserted for one phy_clk cycle for every read command it issues. If there are two read commands, ctl_doing_rd is asserted for two phy_clk cycles. The ctl_doing_rd signal also enables the capture registers and generates the ctl_mem_rdata_valid signal. The ctl_doing_rd signal should be issued at the same time the read command is sent to the ALTMEMPHY megafunction. |
| ctl_rdata | Output | DWIDTH_RATIO × MEM_IF_DWIDTH | Read data from the PHY to the controller. |

**Table 5–4. AFI Signals (Part 3 of 3)**

| Signal Name | Type | Width *(1)* | Description |
|---|---|---|---|
| ctl_rdata_valid | Output | DWIDTH_RATIO/2 | Read data valid indicating valid read data on ctl_rdata. This signal is two-bits wide (as only half-rate or DWIDTH_RATIO = 4 is supported) to allow controllers to issue reads and writes that are aligned to either the half-cycle of the half-rate clock. |
| ctl_rlat | Output | READ_LAT_WIDTH | Contains the number of clock cycles between the assertion of ctl_doing_rd and the return of valid read data (ctl_rdata). This signal is unused by the Altera high-performance controllers. |
| **Address and Command Interface** | | | |
| ctl_addr | Input | MEM_IF_ROWADDR_WIDTH × DWIDTH_RATIO / 2 | Row address from the controller. |
| ctl_ba | Input | MEM_IF_BANKADDR_WIDTH × DWIDTH_RATIO / 2 | Bank address from the controller. |
| ctl_cke | Input | MEM_IF_CS_WIDTH × DWIDTH_RATIO / 2 | Clock enable from the controller. |
| ctl_cs_n | Input | MEM_IF_CS_WIDTH ×DWIDTH_RATIO / 2 | Chip select from the controller. |
| ctl_odt | Input | MEM_IF_CS_WIDTH × DWIDTH_RATIO / 2 | On-die-termination control from the controller. |
| ctl_ras_n | Input | DWIDTH_RATIO / 2 | Row address strobe signal from the controller. |
| ctl_we_n | Input | DWIDTH_RATIO / 2 | Write enable. |
| ctl_cas_n | Input | DWIDTH_RATIO / 2 | Column address strobe signal from the controller. |
| ctl_rst_n | Input | DWIDTH_RATIO / 2 | Reset from the controller. |
| **Calibration Control and Status Interface** | | | |
| ctl_mem_clk_disable | Input | MEM_IF_CLK_PAIR_COUNT | When asserted, mem_clk and mem_clk_n are disabled. |
| ctl_cal_success | Output | 1 | A 1 indicates that calibration was successful. |
| ctl_cal_fail | Output | 1 | A 1 indicates that calibration has failed. |
| ctl_cal_req | Input | 1 | When asserted, a new calibration sequence is started. Currently not supported. |
| ctl_cal_byte_lane_sel_n | Input | MEM_IF_DQS_WIDTH × MEM_CS_WIDTH | Indicates which DQS groups should be calibrated. Not supported. |

**Note to Table 5–3:**

(1) Refer to Table 5–6 for parameter descriptions.

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

| Signal Name | Type | Width | Description |
|---|---|---|---|
| **External DLL Signals** | | | |
| dqs_delay_ctrl_export | Output | DQS_DELAY_CTL_WIDTH | Allows sharing DLL in this ALTMEMPHY instance with another ALTMEMPHY instance. Connect the dqs_delay_ctrl_export port on the ALTMEMPHY instance with a DLL to the dqs_delay_ctrl_import port on the other ALTMEMPHY instance. |

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

| Signal Name | Type | Width | Description |
|---|---|---|---|
| dqs_delay_ctrl_import | Input | DQS_DELAY_CTL_WIDTH | Allows the use of DLL in another ALTMEMPHY instance in this ALTMEMPHY instance. Connect the dqs_delay_ctrl_export port on the ALTMEMPHY instance with a DLL to the dqs_delay_ctrl_import port on the other ALTMEMPHY instance. |
| dqs_offset_delay_ctrl_ width | Input | DQS_DELAY_CTL_WIDTH | Connects to the DQS delay logic when dll_import_export is set to IMPORT. Only connect if you are using a DLL offset, which can otherwise be tied to zero. If you are using a DLL offset, connect this input to the offset_ctrl_out output of the dll_offset_ctrl block. |
| dll_reference_ clk | Output | 1 | Reference clock to feed to an externally instantiated DLL. This clock is typically from one of the PHY PLL outputs. |
| **User-Mode Calibration OCT Control Signals** | | | |
| oct_ctl_rs_value | Input | 14 | OCT RS value port for use with ALT_OCT megafunction if you want to use OCT with user-mode calibration. |
| oct_ctl_rt_value | Input | 14 | OCT RT value port for use with ALT_OCT megafunction if you want to use OCT with user-mode calibration. |
| **Debug Interface Signals *(Note 1)*, *(Note 2)*** | | | |
| dbg_clk | Input | 1 | Debug interface clock. |
| dbg_reset_n | Input | 1 | Debug interface reset. |
| dbg_addr | Input | DBG_A_WIDTH | Address input. |
| dgb_wr | Input | 1 | Write request. |
| dbg_rd | Input | 1 | Read request. |
| dbg_cs | Input | 1 | Chip select. |
| dbg_wr_data | Input | 32 | Debug interface write data. |
| dbg_rd_data | Output | 32 | Debug interface read data. |
| dbg_waitrequest | Output | 1 | Wait signal. |
| **PLL Reconfiguration Signals—Stratix III and Stratix IV Devices** | | | |
| pll_reconfig_enable | Input | 1 | This signal enables the PLL reconfiguration I/O, and is used if the user requires some custom PLL phase reconfiguration. It should otherwise be tied low. |
| pll_phasecounterselect | Input | 4 | When pll_reconfig_enable is asserted, this input is directly connected to the PLL's phasecounterselect input. Otherwise this input has no effect. |
| pll_phaseupdown | Input | 1 | When pll_reconfig_enable is asserted, this input is directly connected to the PLL's phaseupdown input. Otherwise this input has no effect. |
| pll_phasestep | Input | 1 | When pll_reconfig_enable is asserted, this input is directly connected to the PLL's phasestep input. Otherwise this input has no effect. |
| pll_phase_done | Output | 1 | Directly connected to the PLL's phase_done output. |
| I/O Delay Chain Signals—Stratix III, HardCopy III, and HardCopy IV Devices | | | |

**Table 5–5. Other Interface Signals (Part 3 of 4)**

| Signal Name | Type | Width | Description |
|---|---|---|---|
| hc_scan_enable_ access | Input | 1 | This signal switches the control of the levelling delay chains from the sequencer to the hc_scan_ signals. It should normally be tied low. |
| hc_scan_enable_dq | Input | MEM_IF_ DWIDTH | When hc_scan_enable_access is asserted, this bus directly connects to the ena inputs on the IO_CONFIG atoms for every DQ pin. Otherwise, this input has no effect. |
| hc_scan_enable_dm | Input | MEM_IF_D M_ DWIDTH | When hc_scan_enable_access is asserted, this bus directly connects to the ena inputs on the IO_CONFIG atoms for every DM pin. Otherwise, this input has no effect. |
| hc_scan_enable_ dqs | Input | MEM_IF_D QS_ DWIDTH | When hc_scan_enable_access is asserted, this bus directly connects to the ena inputs on the IO_CONFIG atoms for every DQS pin. Otherwise, this input has no effect. |
| hc_scan_enable_ dqs_config | Input | MEM_IF_D QS_ DWIDTH | When hc_scan_enable_access is asserted, this bus directly connects to the ena inputs on the DQS_CONFIG atoms for every DQS pin. Otherwise, this input has no effect. |
| hc_scan_din | Input | MEM_IF_D QS_ DWIDTH | When hc_scan_enable_access is asserted, this bus directly connects to the datain inputs on the IO_CONFIG and DQS_CONFIG atoms for every DQ, DM, and DQS pin. Otherwise, this input has no effect. |
| hc_scan_update | Input | MEM_IF_D QS_ DWIDTH | When hc_scan_enable_access is asserted, this bus directly connects to the update inputs on the IO_CONFIG and DQS_CONFIG atoms for every DQ, DM, and DQS pin. Otherwise, this input has no effect. |
| hc_scan_ck | Input | 1 | When hc_scan_enable_access is asserted, this bus directly connects to the clk inputs on the IO_CONFIG and DQS_CONFIG atoms for every DQ, DM, and DQS pin. Otherwise, this input has no effect. |
| hc_scan_dout | Output | MEM_IF_ DWIDTH | When hc_scan_enable_access is asserted, a multiplexer connects this bus to the relevant dataout outputs of the IO_CONFIG or DQS_CONFIG atoms for the signal group which is currently being selected via the hc_scan_enable_ signals. Otherwise, this input has no effect. |
| **Calibration Interface Signals—without leveling only** | | | |
| rsu_codvw_phase | Output | — | The sequencer sweeps the phase of a resynchronization clock across 360° or 720° of a memory clock cycle. Data reads from the DIMM are performed for each phase position, and a data valid window is located, which is the set of resynchronization clock phase positions where data is successfully read. The final resynchronization clock phase is set at the center of this range: the center of the data valid window or CODVW. This output is set to the current calculated value for the CODVW, and represents how many phase steps were performed by the PLL to offset the resynchronization clock from the memory clock. |
| rsu_codvw_size | Output | — | The final centre of data valid window size (rsu_codvw_size) is the number of phases where data was successfully read in the calculation of the resynchronization clock centre of data valid window phase (rsu_codvw_phase). |

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

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

**Notes to Table 5–5:**

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

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

Table 5–6 shows the parameters that Table 5–3 through Table 5–5 refer to.

**Table 5–6. Parameters**

| Parameter Name | Description |
|---|---|
| DWIDTH_RATIO | The data width ratio from the local interface to the memory interface. DWIDTH_RATIO of 2 means full rate, while DWIDTH_RATIO of 4 means half rate. |
| LOCAL_IF_DWIDTH | The width of the local data bus must be quadrupled for half-rate and doubled for full-rate. |
| MEM_IF_DWIDTH | The data width at the memory interface. MEM_IF_DWIDTH can have values that are multiples of MEM_IF_DQ_PER_DQS. |
| MEM_IF_DQS_WIDTH | The number of DQS pins in the interface. |
| MEM_IF_ROWADDR_WIDTH | The row address width of the memory device. |
| MEM_IF_BANKADDR_WIDTH | The bank address with the memory device. |
| MEM_IF_CS_WIDTH | The number of chip select pins in the interface. The sequencer only calibrates one chip select pin. |
| MEM_IF_DM_WIDTH | The number of mem_dm pins on the memory interface. |
| MEM_IF_DQ_PER_DQS | The number of mem_dq[] pins per mem_dqs pin. |
| MEM_IF_CLK_PAIR_COUNT | The number of mem_clk/mem_clk_n pairs in the interface. |

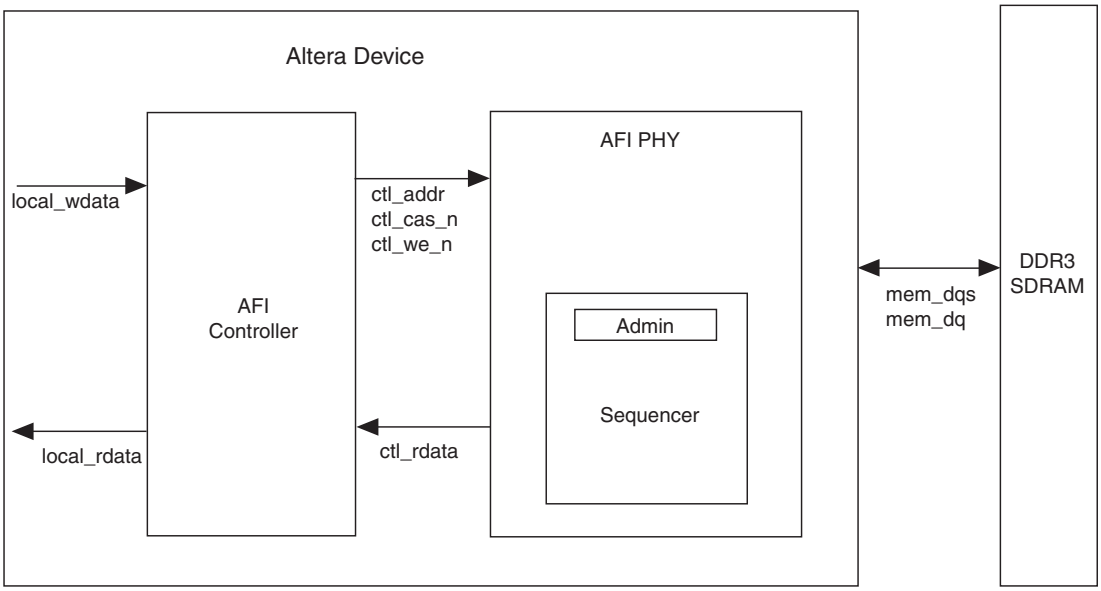
# PHY-to-Controller Interfaces

The following section describes the typical modules that are connected to the ALTMEMPHY variation and the port name prefixes each module uses. This section also describes using a custom controller. This section describes the AFI.

The AFI standardizes and simplifies the interface between controller and PHY for all Altera memory designs, thus allowing you to easily interchange your own controller code with Altera's high-performance controllers. The AFI PHY includes an administration block that configures the memory for calibration and performs necessary mode registers accesses to configure the memory as required (these calibration processes are different). Figure 5–14 shows an overview of the connections between the PHY, the controller, and the memory device.

☞ Altera recommends that you use the AFI for new designs.

**Figure 5–14. AFI PHY Connections**




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

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

For half-rate designs, the AFI allows the controller to issue reads and writes that are aligned to either half-cycle of the half-rate `phy_clk`, which means that the datapaths can support multiple data alignments—word-unaligned and word-aligned writes and reads. Figure 5–15 and Figure 5–16 display the half-rate write operation.

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

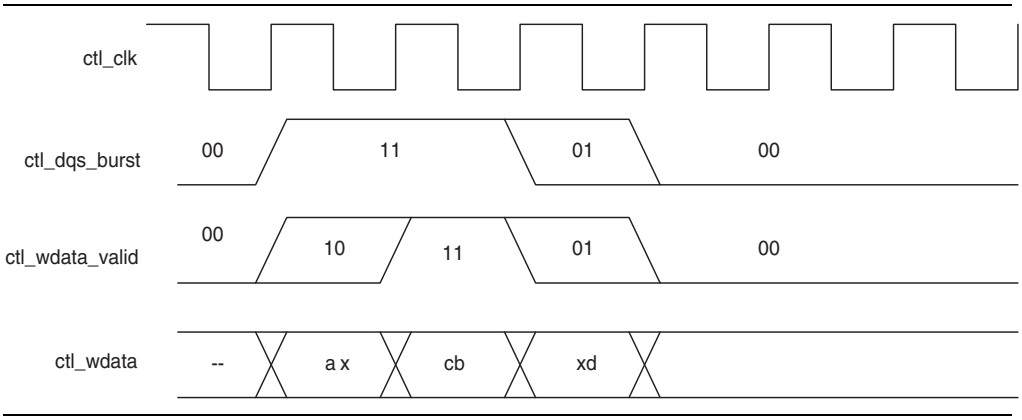
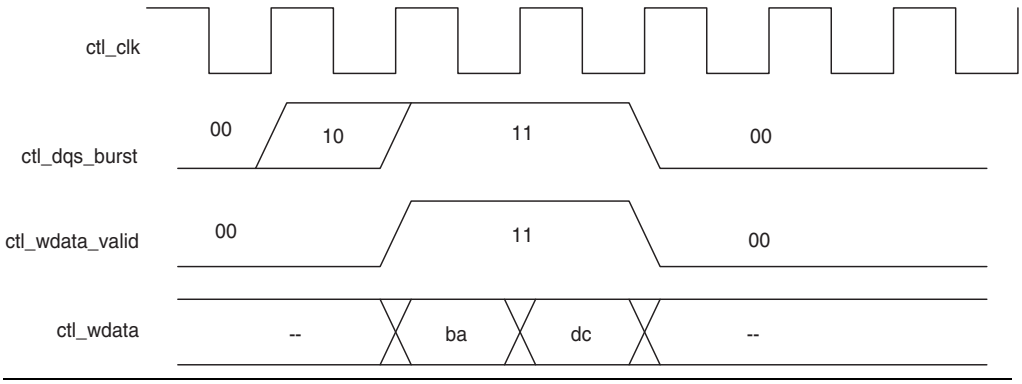



**Figure 5–16. Half-Rate Write with Word-Aligned Data**




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

Figure 5–17 and Figure 5–18 show word-aligned writes and reads. In the following read and write examples the data is written to and read from the same address. In each example, `ctl_rdata` and `ctl_wdata` are aligned with controller clock (`ctl_clk`) cycles. All the data in the bit vector is valid at once. For comparison, refer Figure 5–19 and Figure 5–20 that show the word-unaligned writes and reads.

The `ctl_doing_rd` is represented as a half-rate signal when passed into the PHY. Therefore, the lower half of this bit vector represents one memory clock cycle and the upper half the next memory clock cycle. Figure 5–20 on page 5–37 shows separated word-unaligned reads as an example of two `ctl_doing_rd` bits are different. Therefore, for each x16 device, at least two `ctl_doing_rd` bits need to be driven, and two `ctl_rdata_valid` bits need to be interpreted.

The AFI has the following conventions:

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

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

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

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

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

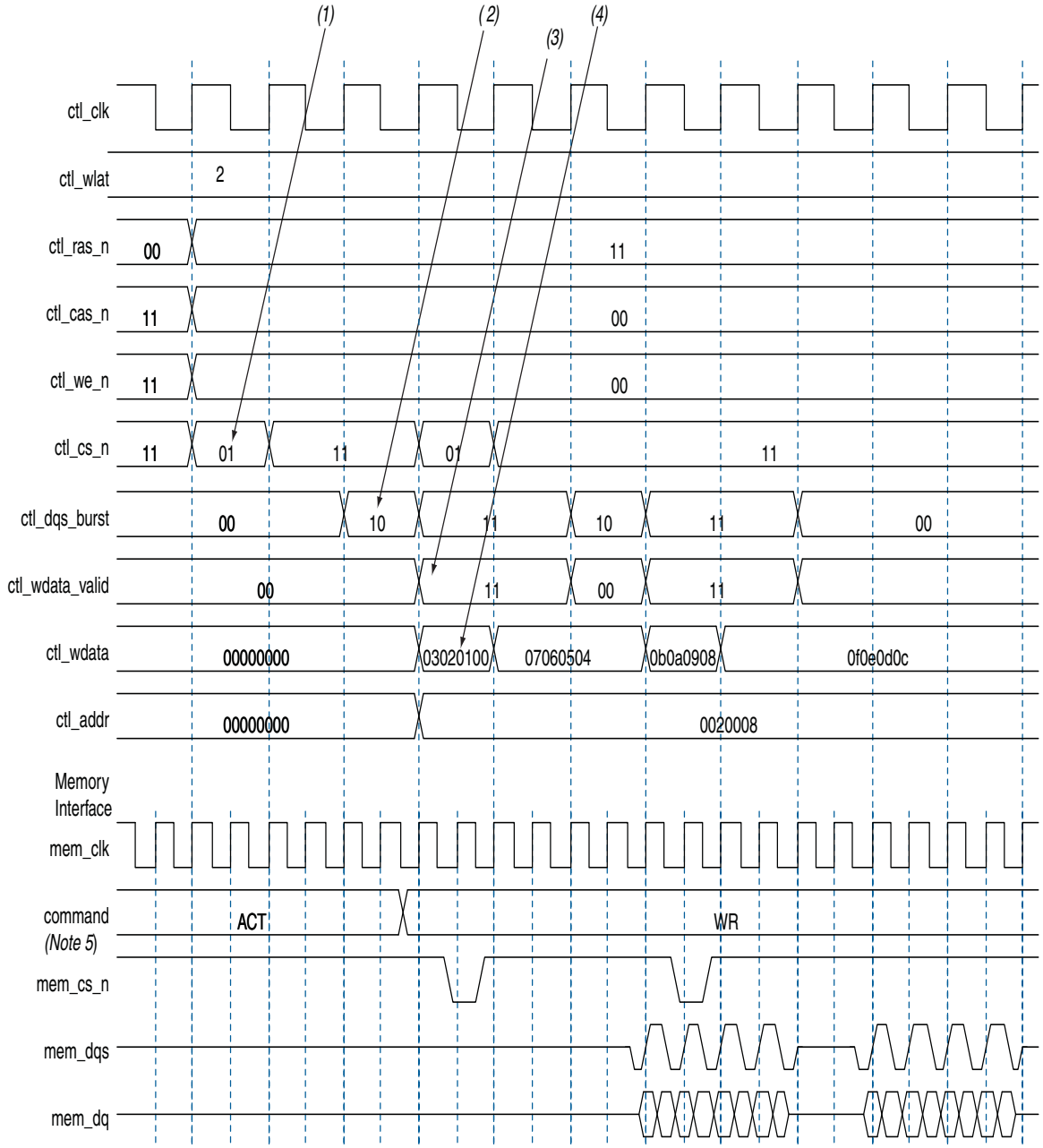
  Word-unaligned reads and writes are only supported on Stratix III and Stratix IV devices.

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

Figure 5–17 through Figure 5–20 assume the following general points:

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

**Figure 5–17. Word-Aligned Writes**




**Notes to Figure 5–17:**

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

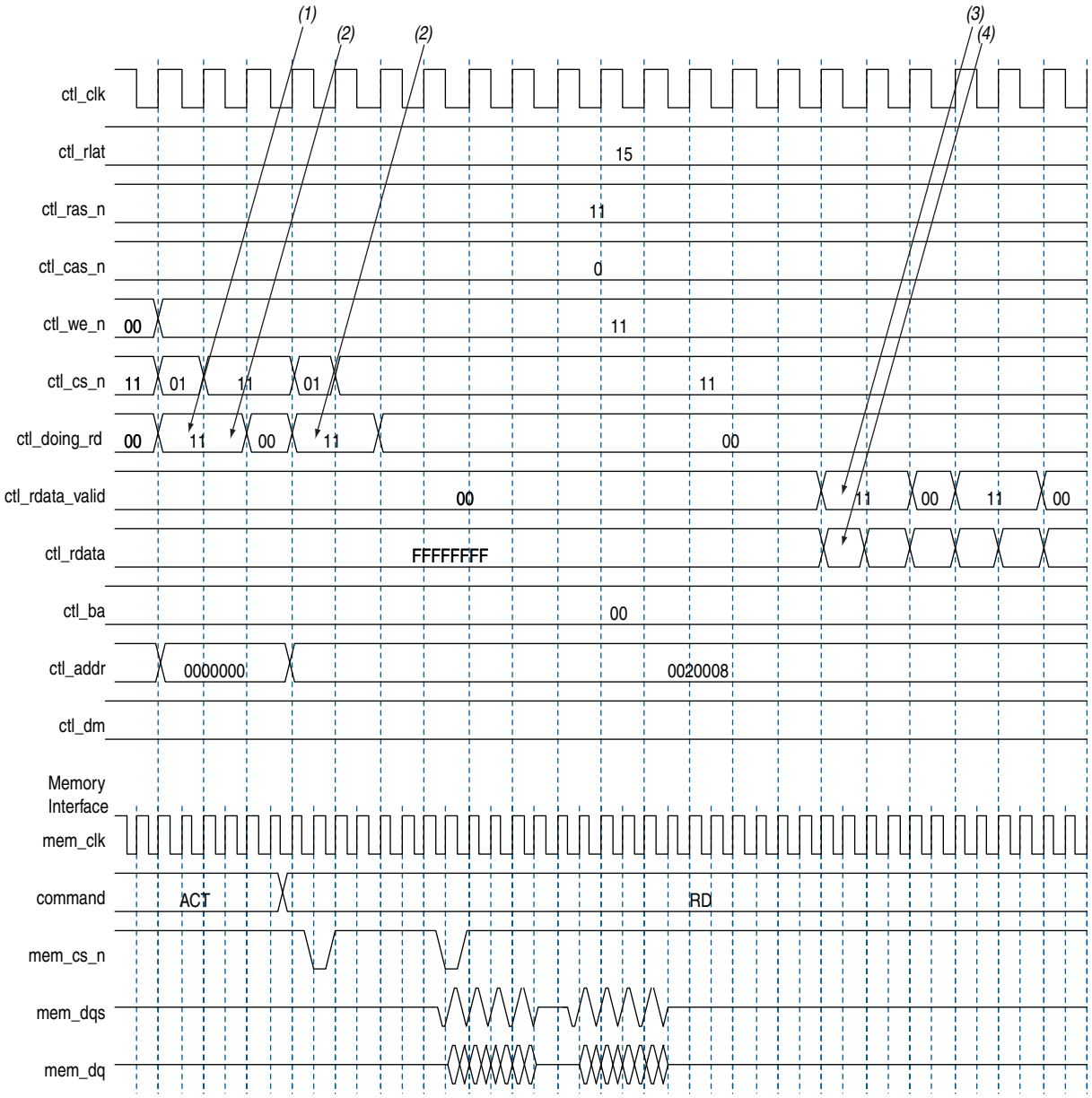
(2) The `ctl_dqs_burst` must go high one memory clock cycle before `ctl_wdata_valid`. Compare with the word-unaligned case.

(3) The `ctl_wdata_valid` is asserted two `ctl_wlat` controller clock (`ctl_clk`) cycles after chip select (`ctl_cs_n`) is asserted. The `ctl_wlat` indicates the required write latency in the system. The value is determined during calibration and is dependant upon the relative delays in the address and command path and the write datapath in both the PHY and the external DDR SDRAM subsystem. The controller must drive `ctl_cs_n` and then wait `ctl_wlat` (two in this example) `ctl_clks` before driving `ctl_wdata_valid`.

(4) Observe the ordering of write data (`ctl_wdata`). Compare this to data on the `mem_dq` signal.

(5) In all waveforms a command record is added that combines the memory pins `ras_n`, `cas_n` and `we_n` into the current command that is issued. This command is registered by the memory when chip select (`mem_cs_n`) is low. The important commands in the presented waveforms are WR = write, ACT = activate.

**Figure 5–18. Word-Aligned Reads**




**Notes to Figure 5–18:**

(1) For AFI, `ctl_doing_rd` is required to be asserted one memory clock cycle before chip select (`ctl_cs_n`) is asserted. In the half-rate `ctl_clk` domain, this requirement manifests as the controller driving `11` (as opposed to the `01`) on `ctl_doing_rd`.

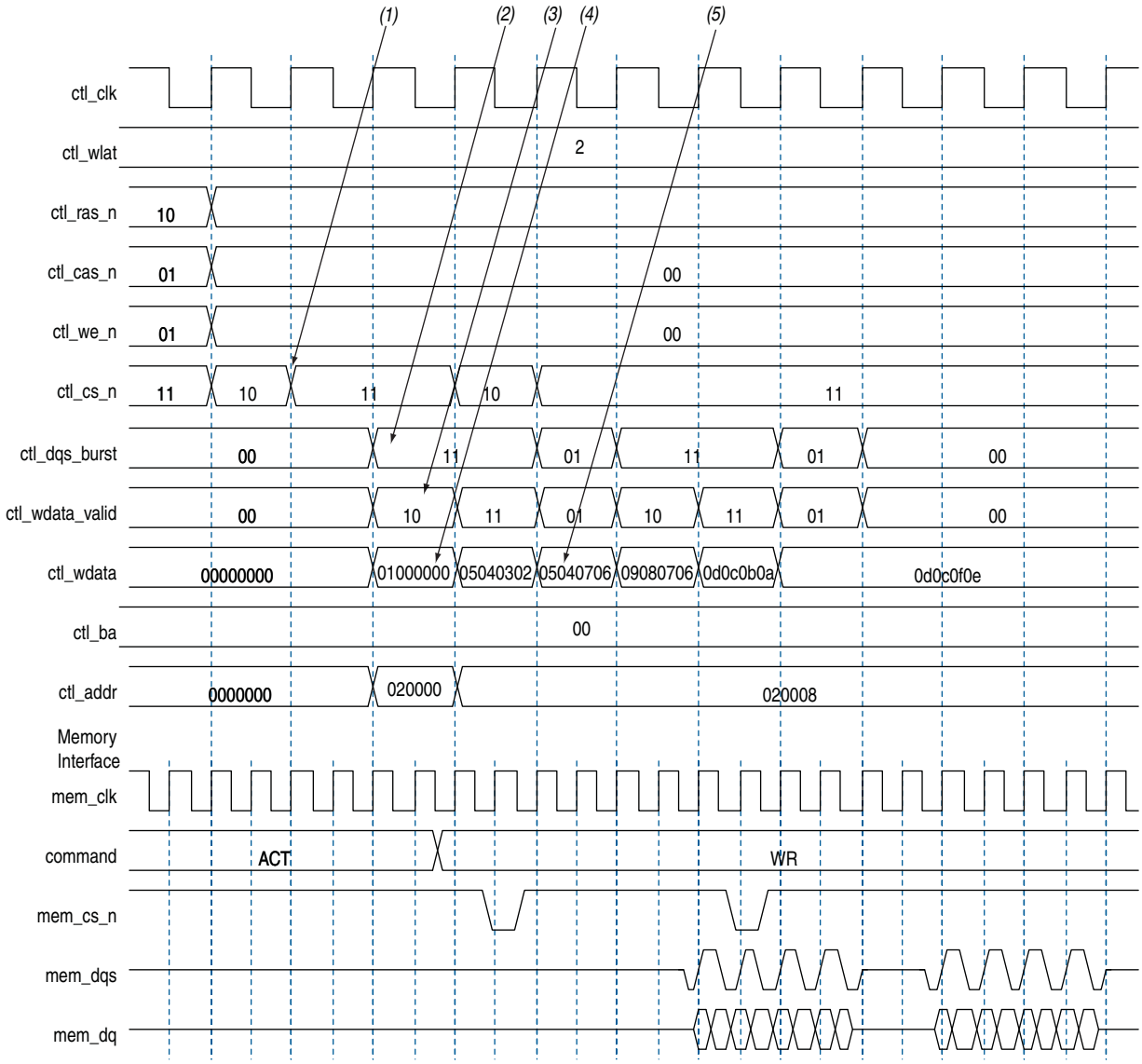
(2) AFI requires that `ctl_doing_rd` is driven for the duration of the read. In this example, it is driven to `11` for two half-rate `ctl_clks`, which equates to driving to `1`, for the four memory clock cycles of this four-beat burst.

(3) The `ctl_rdata_valid` returns 15 (`ctl_rlat`) controller clock (`ctl_clk`) cycles after `ctl_doing_rd` is asserted. Returned is when the `ctl_rdata_valid` signal is observed at the output of a register within the controller. A controller can use the `ctl_rlat` value to determine when to register to returned data, but this is unnecessary as the `ctl_rdata_valid` is provided for the controller to use as an enable when registering read data.

(4) Observe the alignment of returned read data with respect to data on the bus.

Figure 5–19 and Figure 5–20 show spaced word-unaligned writes and reads.

**Figure 5–19. Word-Unaligned Writes**




**Notes to Figure 5–19:**

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

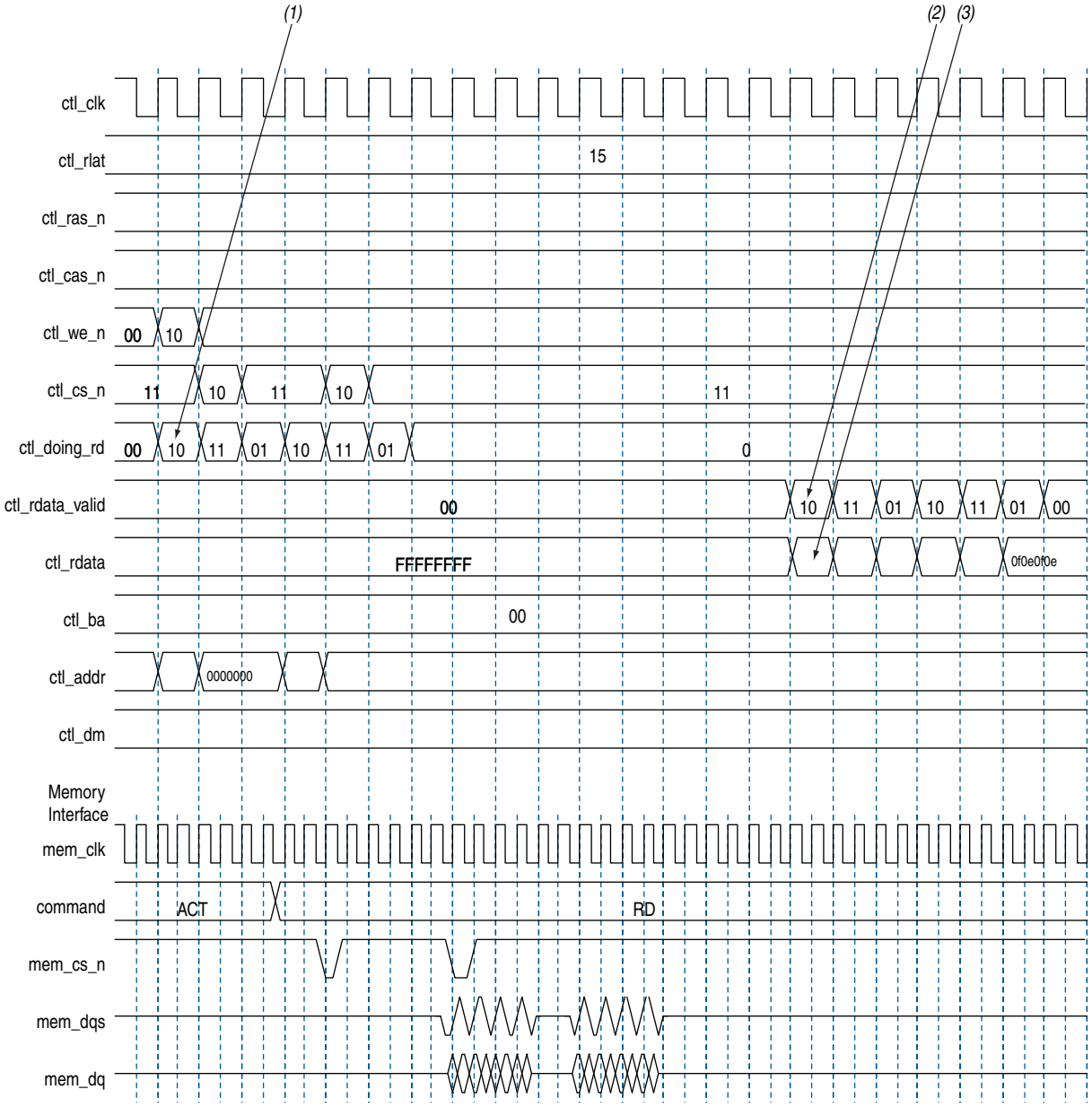
(2) As with word- aligned writes, `ctl_dqs_burst` is asserted one memory clock cycle before `ctl_wdata_valid`. You can see `ctl_dqs_burst` is 11 in the same cycle where `ctl_wdata_valid` is 10. The LSB of these two becomes the first value the signal takes in the `mem_clk` domain. You can see that `ctl_dqs_burst`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

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

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

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

**Figure 5–20. Word-Unaligned Reads**




**Notes to Figure 5–20:**

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

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

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

# Using a Custom Controller

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

## Preliminary Steps

Perform the following steps to generate the ALTMEMPHY megafunction:

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

## Design Considerations

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

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

## Clocks and Resets

The ALTMEMPHY megafunction automatically generates a PLL instance, but you must still provide the reference clock input (`pll_ref_clk`) with a clock of the frequency that you specified in the MegaWizard Plug-In Manager. An active-low global reset input is also provided, which you can deassert asynchronously. The clock and reset management logic synchronizes this reset to the appropriate clock domains inside the ALTMEMPHY megafunction.

A clock output, half the memory clock frequency for a half-rate controller, is provided and all inputs and outputs of the ALTMEMPHY megafunction are synchronous to this clock. For AFIs, this signal is called `ctl_clk`.

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

## Calibration Process Requirements

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

## Other Local Interface Requirements

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

## Address and Command Interfacing

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

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

## Handshake Mechanism Between Read Commands and Read Data

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

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

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

The ALTMEMPHY provides a signal, ctl_rdata_valid, to indicate that the data on read data bus is valid. The width of this signal varies between half-rate and full-rate designs to support the option to indicate that the read data is not word aligned. Figure 5–21 and Figure 5–22 show these relationships.

**Figure 5–21. Address and Command and Read-Path Timing—Full-Rate Design**

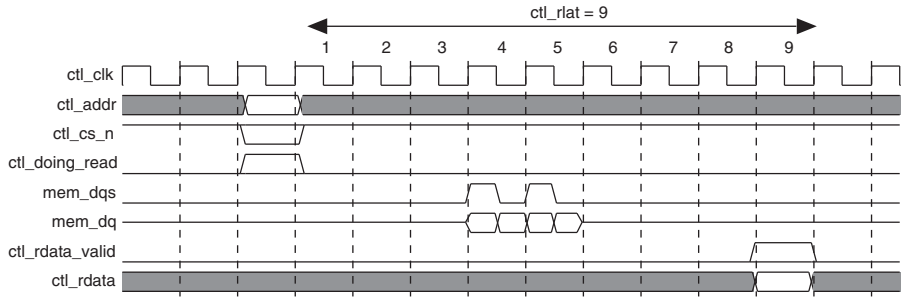
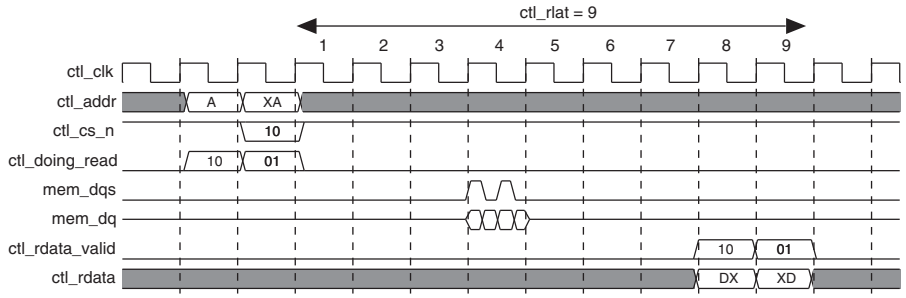



**Figure 5–22. Second Read Alignment—Half-Rate Design**



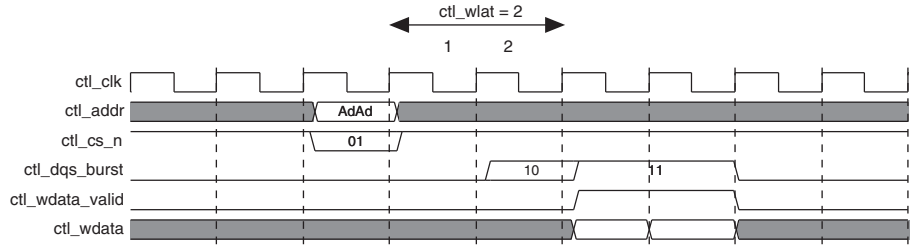

## Handshake Mechanism Between Write Commands and Write Data

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

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

The ctl_wlat signal is only valid when the calibration has been successfully completed by the ALTMEMPHY sequencer and does not change at any point during normal user mode operation. Figure 5–23 shows the operation of ctl_wlat port.

**Figure 5–23. Timing for ctl_dqs_burst, ctl_wdata_valid, Address, and Command—Half-Rate Design**




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

## Partial Writes

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

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

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

The I/O difference between Stratix III devices and other devices, and the preamble difference for DDR3 SDRAM on Arria II GX devices make it only possible to use the ctl_dqs_burst signal for the DQS enable in Stratix III devices.

The high-performance controller (HPC) architecture instantiates encrypted control logic and the ALTMEMPHY megafunction. The controller accepts read and write requests from the user on its local interface, using either the Avalon-MM interface protocol or the native interface protocol. It converts these requests into the necessary SDRAM commands, including any required bank management commands. Each read or write request on the Avalon-MM or native interface maps to one SDRAM read or write command.

The half-rate DDR3 SDRAM HPC accepts requests of size 1 or 2 on the local interface. If you request a burst size of 1, the controller issues a memory burst of 4 using the DDR3 SDRAM on-the-fly burst chop (waits for two cycles before issuing the next read or write command). If you request a burst size of 2, the controller issues a memory burst of 8 (issues the next read or write command back to back). Requests of size 2 on the local interface produce better throughput because DDR3 SDRAMs cannot accept back-to-back bursts of size 4.

The bank management logic in the controller keeps a row open in every bank in the memory system. For example, a controller configured for a dual-rank, 8-bank DDR3 SDRAM DIMM keeps an open row in each of the 16 banks. The controller allows you to request an auto-precharge read or auto-precharge write, allowing control over whether to keep that row open after the request. You can achieve maximum efficiency when you issue reads and writes to the same bank, with the last access to that bank being an auto-precharge read or write. The controller does not do any access reordering.

## Block Description

Figure 6–1 on page 6–1 shows the top-level block diagram of the DDR3 SDRAM HPC.

**Figure 6–1. DDR3 SDRAM HPC Block Diagram**

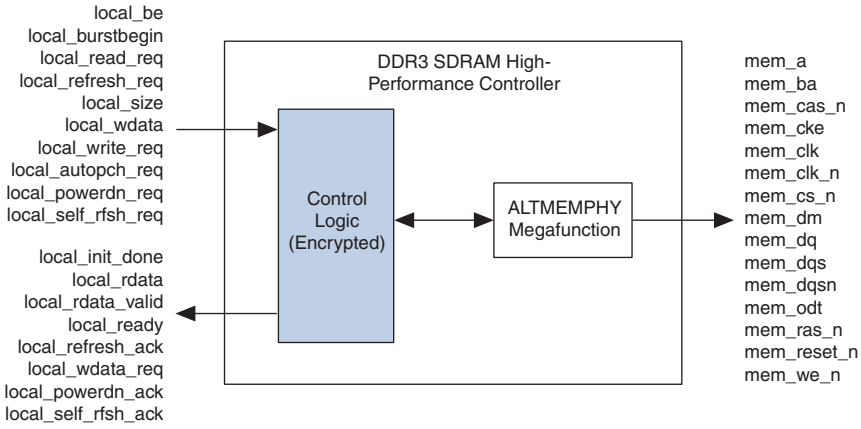
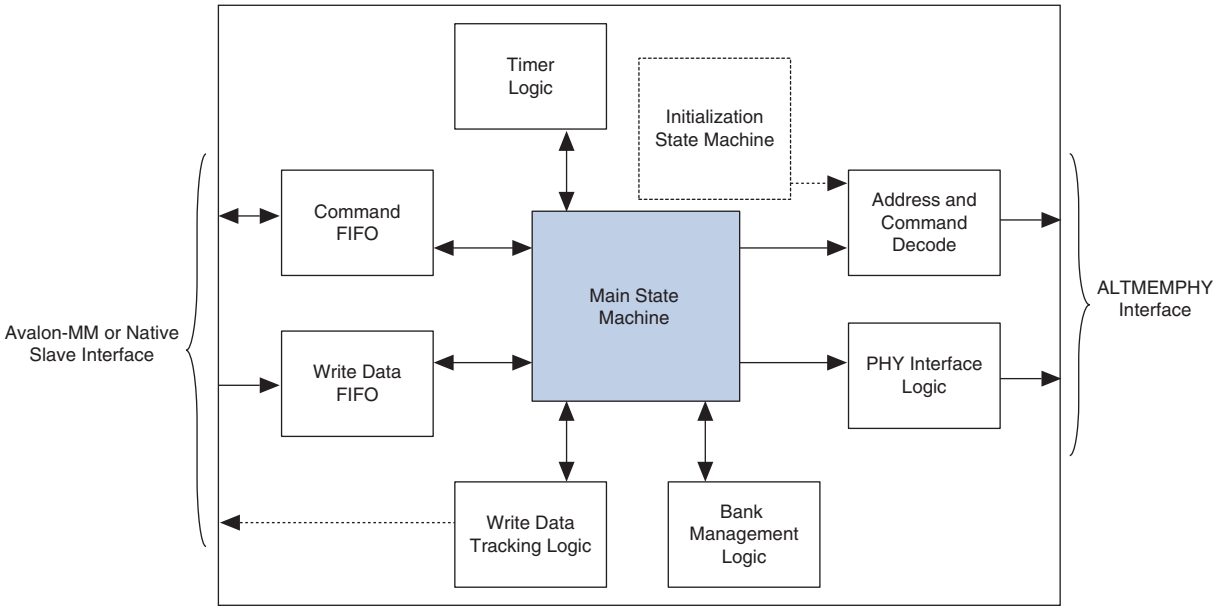


Figure 6–2 shows a block diagram of the DDR3 SDRAM HPC architecture.

**Figure 6–2. DDR3 SDRAM HPC Architecture Block Diagram**




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

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

## Command FIFO Buffer

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

## Write Data FIFO Buffer

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

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

## Write Data Tracking Logic

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

## Main State Machine

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

## Bank Management Logic

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

## Timer Logic

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

## Initialization State Machine

The initialization state machine issues the appropriate sequence of command to initialize the memory devices. It is specific to DDR3 as each memory type requires a different sequence of initialization commands.

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

## Address and Command Decode

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

## PHY Interface Logic

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

## ODT Generation Logic

The ODT generation logic (not shown in Figure 6–2) calculates when and for how long to enable the ODT outputs. It also decides which ODT bit to enable, based on the number of chip selects in the system.

- 1 DIMM (1 or 2 chip selects)

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

- 2 or more DIMMs

  In the multiple DIMM case, the appropriate ODT bit is asserted for both read and writes. Table 6–1 shows which ODT signal on the adjacent DIMM is enabled.

**Table 6–1. ODT**

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

## Low-Power Mode Logic

The low-power mode logic (not shown in Figure 6–2) monitors the `local_powerdn_req` and `local_self_rfsh_req` request signals. This logic also informs the user of the current low-power state via the `local_powerdn_ack` and `local_self_rfsh_ack` acknowledge signals.

HPC supports only precharge power-down mode and not active power-down mode.

## Control Logic

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

**Table 6–2. Bus Commands**

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

The DDR3 SDRAM HPC must open SDRAM banks before it accesses the 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 HPC closes the bank and opens it again if it needs to access a different row. The precharge (PCH) command closes only a bank.

The primary commands used to access SDRAM are read (RD) and write (WR). When the WR command is issued, the initial column address and data word is registered. When a RD command is issued, the initial address is registered. The initial data appears on the data bus 5 to 11 clock cycles later. 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 (of 6) 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. DDR3 SDRAM devices support fixed burst lengths of 4 or 8 data cycles or an on-the-fly mode where the controller can request a burst of 4 or 8 for each read or write command. This on-the-fly mode is the only mode supported. The auto-refresh command (ARF) is issued periodically to ensure data retention. This function is performed by the DDR3 SDRAM HPC.

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

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

## Error Correction Coding (ECC)

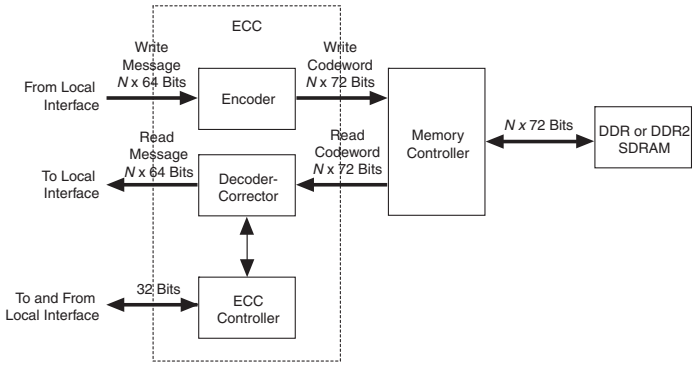
The optional ECC logic comprises an encoder and a decoder-corrector, which can detect and correct single-bit errors and detect double-bit errors. The ECC logic uses an 8-bit ECC for each 64-bit message. The ECC logic has the following features:

- Hamming code ECC logic that encodes every 64-bits of data into 72-bits of codeword with 8-bits of Hamming code parity bits

- Latency:
  - Maximum of 1 or 2 clock delay during writes
  - Minimum 1 or 3 clock delay during reads
- Detects and corrects all single-bit errors. Also the ECC logic sends an interrupt when the user-defined threshold for a single-bit error is reached.
- Detects all double-bit errors. Also, the ECC logic counts the number of double-bit errors and sends an interrupt when the user-define threshold for double-bit error is reached.
- Accepts partial writes
- Creates forced errors to check the functioning of the ECC logic
- Powers up to a ready state

Figure 6–3 shows the ECC block diagram.

**Figure 6–3. ECC Block Diagram**




The ECC comprises the following blocks:

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

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

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

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

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

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

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

When testing the DDR3 SDRAM HPC, you can turn off the ECC.

## Interrupts

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

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

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

## Partial Writes

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

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

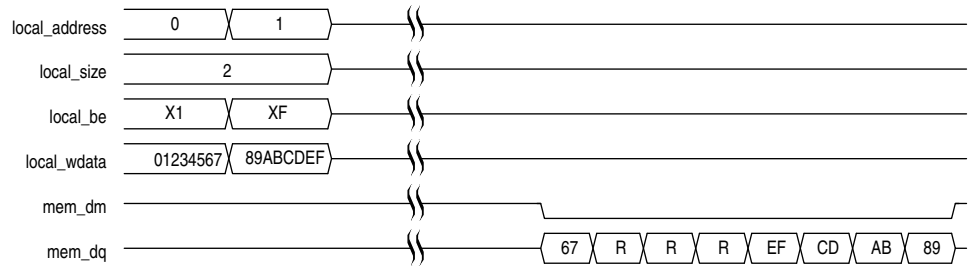
1. The ECC logic stalls further read or write commands from the Avalon-MM interface when it receives a partial write condition.
2. It simultaneously sends a self-generated read command, for the partial write address, to the memory controller.
3. Upon receiving the returned read data from the memory controller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

The following corner cases can occur:

- A single-bit error during the read phase of the read-modify-write process. In this case, the single-bit error is corrected first, the single-bit error counter is incremented and then a partial write is performed to this corrected decoded data word.
- A double-bit error during the read phase of the read-modify-write process. In this case, the double-bit error counter is incremented and an interrupt is sent through the Avalon-MM interface. The new write word is written to the memory location. A separate field in the interrupt status register highlights this condition.

Figure 6–4 shows the partial write operation for HPC. The half-rate DDR3 SDRAM HPC supports a local size of 1 and 2.

**Figure 6–4. Partial Write for HPC**




**Note to Figure 6–4:**

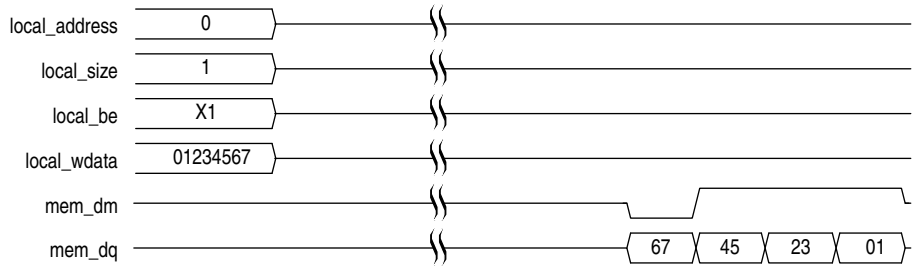
(1) R represents the internal read-back memory data during the read-modify-write process.

## Partial Bursts

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

Figure 6–5 shows the partial burst operation for HPC.

**Figure 6–5. Partial Burst for HPC**




## ECC Latency

Using the ECC results in the following latency changes:

- Local Burst Length 1
- Local Burst Length 2

### Local Burst Length 1

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

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

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

**Table 6–3. Burst Lengths and Rates**

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

### Local Burst Length 2

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

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

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

## ECC Registers

Table 6–4 shows the ECC registers.

**Table 6–4. ECC Registers (Part 1 of 3)**

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

**Table 6–4. ECC Registers (Part 2 of 3)**

| Name | Address | Size (Bits) | Attribute | Default | Description |
|---|---|---|---|---|---|
| Last or first double-bit error error address | 06 | 32 | RO | 00000000 | This status register stores the last double-bit error error address. It can be cleared using the control word clear. If bit 10 of the control word is set high, the first occurred address is stored. |
| Last single-bit error error data | 07 | 32 | RO | 00000000 | This status register stores the last single-bit error error data word. As the data word is an *M*th multiple of 64, the data word is stored in a 2*N*-deep, 32-bit wide FIFO buffer with the least significant 32-bit sub word stored first. It can be cleared individually by using the control word clear. |
| Last single-bit error syndrome | 08 | 32 | RO | 00000000 | This status register stores the last single-bit error syndrome, which specifies the location of the error bit on a 64-bit data word. As the data word is an *M*th multiple of 64, the syndrome is stored in a *N* deep, 8-bit wide FIFO buffer where each syndrome represents errors in every 64-bit part of the data word. The register gets updated with the correct syndrome depending on which part of the data word is shown on the last single-bit error error data register. It can be cleared individually by using the control word clear. |
| Last double-bit error error data | 09 | 32 | RO | 00000000 | This status register stores the last double-bit error error data word. As the data word is an *M*th multiple of 64, the data word is stored in a 2*N* deep, 32-bit wide FIFO buffer with the least significant 32-bit sub word stored first. It can be cleared individually by using the control word clear. |
| Interrupt status register | 0A | 5 | RO | 00000000 | This status register stores the interrupt status in four fields (refer to Table 6–6). These status bits can be cleared by writing a `1` in the respective locations. |
| Interrupt mask register | 0B | 5 | WO | 00000001 | This register stores the interrupt mask in four fields (refer to Table 6–7). |

**Table 6–4. ECC Registers (Part 3 of 3)**

| Name | Address | Size (Bits) | Attribute | Default | Description |
|---|---|---|---|---|---|
| Single-bit error location status register | 0C | 32 | R/W | 00000000 | This status register stores the occurrence of single-bit error for each 64-bit part of the data word in every bit (refer to Table 6–8). These status bits can be cleared by writing a 1 in the respective locations. |
| Double-bit error location status register | 0D | 32 | R/W | 00000000 | This status register stores the occurrence of double-bit error for each 64-bit part of the data word in every bit (refer to Table 6–9). These status bits can be cleared by writing a 1 in the respective locations. |

## ECC Register Bits

Table 6–5 shows the control word specification register.

**Table 6–5. Control Word Specification Register**

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

Table 6–6 shows the interrupt status register.

**Table 6–6. Interrupt Status Register**

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

Table 6–7 shows the interrupt mask register.

**Table 6–7. Interrupt Mask Register**

| Bit | Name | Description |
|---|---|---|
| 0 | Single-bit error | When 1, masks single-bit error. |
| 1 | Double-bit error | When 1, masks interrupt when double-bit error occurs during a normal or read-modify-write condition (partial write).<br>When 0, interrupt when double-bit error occurs during a normal or read-modify-write condition (partial write). |
| 2 | Maximum single-bit error | When 1, masks single-bit error maximum threshold exceeding condition. |
| 3 | Maximum double-bit error | When 1, masks double-bit error maximum threshold exceeding condition. |
| 4 | Double-bit error during read-modify-write | When 1, masks interrupt when double-bit error occurs during a read-modify-write condition (partial write).<br>When 0, interrupt when double-bit error occurs during a read-modify-write condition (partial write). |
| Others | Reserved | Reserved. |

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

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

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

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

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

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

# Example Top-Level File

The MegaWizard Plug-In Manager helps you create an example top-level file that shows you how to instantiate and connect the DDR3 SDRAM HPC. The example top-level file consists of the DDR3 HPC, some driver logic to issue read and write requests to the controller. The example top-level file is a working system that you can compile and use for both static timing checks and board tests.

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

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

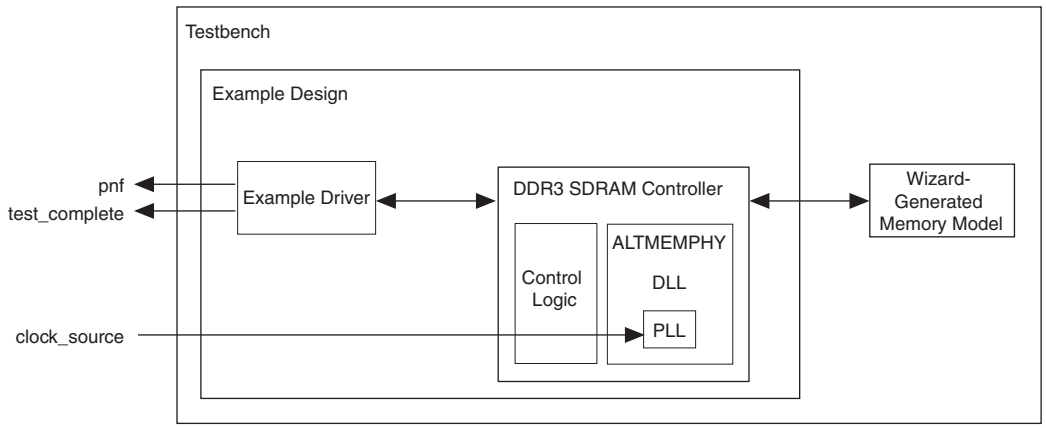



Table 6–10 describes the files that are associated with the example top-level file and the testbench.

**Table 6–10. Example Top-Level File and Testbench Files**

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

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

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

The full-array memory model (*<variation name>*_**full_mem_model.v**) allocates memory for all addresses accessible by the DDR cores. This allocation makes it impossible to simulate large memory (more than 2K address spaces) designs, because simulators need more memory than what is available on a typical system.

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

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

## Example Driver

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

It performs the following tests and loops back the tests indefinitely:

- Sequential addressing writes and reads

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

- Incomplete write operation

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

- Byte enable/data mask pin operation

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

- Address pin operation

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

- Low-power mode operation

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

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

Table 6–11 shows the bit mapping for each test status.

**Table 6–11. Test Status[] Bit Mapping**

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

# Top-level Signals Description

Table 6–12 shows the clock and reset signals.

**Table 6–12. Clock and Reset Signals (Part 1 of 2)**

| Name | Direction | Description |
|---|---|---|
| `global_reset_n` | Input | The asynchronous reset input to the controller. All other reset signals are derived from resynchronized versions of this signal. This signal holds the complete ALTMEMPHY megafunction, including the PLL, in reset while low. |
| `pll_ref_clk` | Input | The reference clock input to PLL. |
| `soft_reset_n` | Input | Edge detect reset input intended for SOPC Builder use or to be controlled by other system reset logic. It is asserted to cause a complete reset to the PHY, but not to the PLL used in the PHY. |

**Table 6–12. Clock and Reset Signals (Part 2 of 2)**

| Name | Direction | Description |
|---|---|---|
| oct_ctl_rs_value | Input | ALTMEMPHY signal that specifies the serial termination value. Should be connected to the ALT_OCT megafunction output seriesterminationcontrol. |
| oct_ctl_rt_value | Input | ALTMEMPHY signal that specifies the parallel termination value. Should be connected to the ALT_OCT megafunction output parallelterminationcontrol. |
| dqs_delay_ctrl_import | Input | Allows the use of DLL in another ALTMEMPHY instance in this ALTMEMPHY instance. Connect the export port on the ALTMEMPHY instance with a DLL to the import port on the other ALTMEMPHY instance. |

Table 6–13 on page 6–17 shows the DDR3 SDRAM HPC local interface signals.

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

| Signal Name | Direction | Description |
|---|---|---|
| local_address[] | Input | Memory address at which the burst should start. The width of this bus is sized using the following equation:<br>For one chip select:<br>width = bank bits + row bits + column bits – 2<br>For multiple chip selects:<br>width = chip bits + bank bits + row bits + column bits – 2<br>If the bank address is 3 bits wide, row is 14 bits wide and column is 10 bits wide, then the local address is 25 bits wide. To map local_address to bank, row and column address:<br>local_address is 25 bits wide<br>local_address[24:22] = bank address [2:0]<br>local_address[21:8] = row address [13:0]<br>local_address [7:0] = col_address[9:2]<br>The 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.<br>You can get the information on address mapping from the *<variation_name>*_**example_top.v** or **vhd** file. |
| local_be[] | Input | Byte enable signal, which you use to mask off individual bytes during writes. local_be is active high; mem_dm is active low.<br>To map local_wdata and local_be to mem_dq and mem_dm, consider a full-rate design with 32-bit local_wdata and 16-bit mem_dq.<br>Local_wdata = < 22334455 >< 667788AA >< BBCCDDEE ><br>Local_be = < 1100 >< 0110 >< 1010 ><br>These values map to:<br>Mem_dq = <4455><2233><88AA><6677><DDEE><BBCC><br>Mem_dm = <1 1 ><0 0 ><0 1 ><1 0 ><0 1 ><0 1 > |

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

| Signal Name | Direction | Description |
|---|---|---|
| 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.<br>For write transactions, assert this signal at the beginning of each burst transfer and keep this signal high for one cycle per burst transfer, even if the slave has deasserted the local_ready signal. After the slave deasserts the local_ready signal, the master keeps all the write request signals asserted until the local_ready signal becomes high again.<br>For read transactions, assert this signal for one clock cycle when read request is asserted and the local_address from which the data should be read is given to the memory. After the slave deasserts the local_ready signal (waitrequest_n in Avalon), the master keeps all the read request signals asserted until the local_ready signal becomes high again. |
| local_read_req | Input | Read request signal. You cannot assert the read request signal before the reset_phy_clk_n signal goes high. |
| local_refresh_req | Input | User-controlled refresh request. If **Enable user auto-refresh controls** 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 DDR3 SDRAM HPC supports burst lengths of 1 and 2 on the local side interface. |
| local_wdata[] | Input | Write data bus. The width of local_wdata is twice that of the memory data bus for a full rate controller; four times the memory data bus for a half rate controller. |
| local_write_req | Input | Write request signal. You cannot assert the write request signal before the reset_phy_clk_n signal goes high. |
| local_autopch_req | Input | User control of precharge. If **Enable Auto-Precharge Control** is turned on, local_autopch_req becomes available and you can request the controller to issue an auto-precharge write or auto-precharge read command. These commands cause the memory to issue a precharge command to the current bank at the appropriate time without an explicit precharge command from the controller. This is particularly useful if you know the current read or write is the last one you intend to issue to the currently open row. The next time you need to use that bank, the access could be quicker as the controller does not need to precharge the bank before activating the row you wish to access. |

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

| Signal Name | Direction | Description |
|---|---|---|
| local_powerdn_req | Input | User control of the power-down feature. If **Enable Power Down Controls** option is enabled, you can request that the controller place the memory devices into a power-down state as soon as it can without violating the relevant timing parameters and responds by asserting the local_powerdn_ack signal. You can hold the memory in the power-down state by keeping this signal asserted. The controller brings the memory out of the power-down state to issue periodic auto-refresh commands to the memory at the appropriate interval if you hold it in the power-down state. You can release the memory from the power-down state at any time by deasserting the local_powerdn_ack signal once it has successfully brought the memory out of the power-down state. |
| local_self_rfsh_req | Input | User control of the self-refresh feature. If **Enable Self-Refresh Controls** option is enabled, you can request that the controller place the memory devices into a self-refresh state by asserting this signal. The controller places the memory in the self-refresh state as soon as it can without violating the relevant timing parameters and responds by asserting the local_self_rfsh_ack signal. You can hold the memory in the self-refresh state by keeping this signal asserted. You can release the memory from the self-refresh state at any time by deasserting the local_self_rfsh_req signal and the controller responds by deasserting the local__self_rfsh_ack signal once it has successfully brought the memory out of the self-refresh state. |
| phy_clk | Output | The system clock that the ALTMEMPHY megafunction provides to the user. All user inputs to and outputs from the DDR HPC must be synchronous to this clock. |
| reset_phy_clk_n | Output | The reset signal that the ALTMEMPHY megafunction provides to the user. It is asserted asynchronously and deasserted synchronously to phy_clk clock domain. |
| dll_reference_clk | Output | Reference clock to feed to an externally instantiated DLL. |
| reset_request_n | Output | Reset request output that indicates when the PLL outputs are not locked. Use this signal as a reset request input to any system-level reset controller you may have. This signal is always low when the PLL is trying to lock, and so any reset logic using it is advised to detect a reset request on a falling edge rather than by level detection. |
| local_init_done | Output | When the memory initialization, training, and calibration are complete, the ALTMEMPHY sequencer asserts the ctrl_usr_mode_rdy signal to the memory controller, which then asserts this signal to indicate that the memory interface is ready to be used.<br>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.<br>This signal does not indicate that the calibration is successful. To find out if the calibration is successful, look for the calibration signal, ctl_cal_success or ctl_cal_fail. |
| local_rdata[] | Output | Read data bus. The width of local_rdata is four times the memory data bus. |
| local_rdata_error | Output | Asserted if the current read data has an error. This signal is only available if the **Enable error detection and correction logic** is turned on. |

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

| Signal Name | Direction | Description |
|---|---|---|
| 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_ready | Output | The local_ready signal indicates that the DDR3 SDRAM HPC 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 HPC cannot accept any more requests. The controller is able to buffer four read or write requests. |
| local_refresh_ack | Output | Refresh request acknowledge, which is asserted for one clock cycle every time a refresh is issued. Even if the **Enable User Auto-Refresh Controls** option is not selected, local_refresh_ack still indicates to the local interface that the controller has just issued a refresh command. |
| local_wdata_req | Output | Write data request signal, which indicates to the local interface that it should present valid write data on the next clock edge. This signal is only required when the controller is operating in **Native interface** mode. |
| local_powerdn_ack | Output | Power-down request acknowledge signal. This signal is asserted and deasserted in response to the local_powerdn_req signal from the user. |
| local_self_rfsh_ack | Output | Self refresh request acknowledge signal. This signal is asserted and deasserted in response to the local_self_rfsh_req signal from the user. |

Table 6–14 shows the DDR3 SDRAM interface signals.

**Table 6–14. DDR3 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 DDR3 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. |
| mem_ras_n | Output | Memory row address strobe signal. |
| mem_reset_n | Output | Memory reset signal. |
| mem_we_n | Output | Memory write enable signal. |

**Note to Table 6–14:**

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

Table 6–15 shows the ECC logic signals.

**Table 6–15. ECC Logic Signals**

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

# 7. Functional Description—High-Performance Controller II

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

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

- Higher efficiency with in-order read and write commands, and out-of-order bank management commands
- Run-time programmability to configure the behavior of the controller
- Integrated burst adapter supporting a range of burst sizes on the local interface
- Integrated ECC logic, supporting 40-bit and 72-bit interfaces with partial word writes and optional write back on error
- Reduced bank tracking for area optimization
- Controller variable latency to enhance the performance of your design
- Support for multi-rank UDIMM and RDIMM ports

## Upgrading from HPC to HPC II

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

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

  For example, for Micron DDR3-800 datasheet, $t_{FAW}$=40 ns, $t_{RRD}$=10 ns, $t_{RTP}$=10 ns.
- HPC II replaces the port interface level for the AFI and Avalon interface without requiring any top-level change.

- The side-band signals differ slightly for HPC II. If you use these signals, you need to perform the following steps.
  - `local_refresh_req`

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

    The user-manual power signal is no longer supported in HPC II. Instead, you can select auto power-down on the **Controller Settings** tab in the MegaWizard Plug-In Manager, and specify the desired time-out (*n* cyles) after which the controller automatically powers down the memory.
- Because HPC II only supports a specific memory burst length, you must update the memory burst length to match the controller settings. For DDR3, HPC II supports on-the-fly burst length in half-rate mode.
- Because HPC II supports arbitrary user burst length ranging from of 1 to 64, you can adjust the `max_local_size` value in HPC II. Adjusting the maximum local size value changes the width of the `local_size` signal. The maximum `local_size` signal value is $2^{n}-1$, where $n$ is the width of the `local_size` signal. HPC has a fixed `local_size` signal width of 2.

You only can migrate your HPC designs to HPC II if you are using an Avalon-MM interface.

# Block Description

Figure 7–1 shows the top-level block diagram of the DDR3 SDRAM HPC II.

**Figure 7–1. DDR3 SDRAM HPC II Block Diagram**

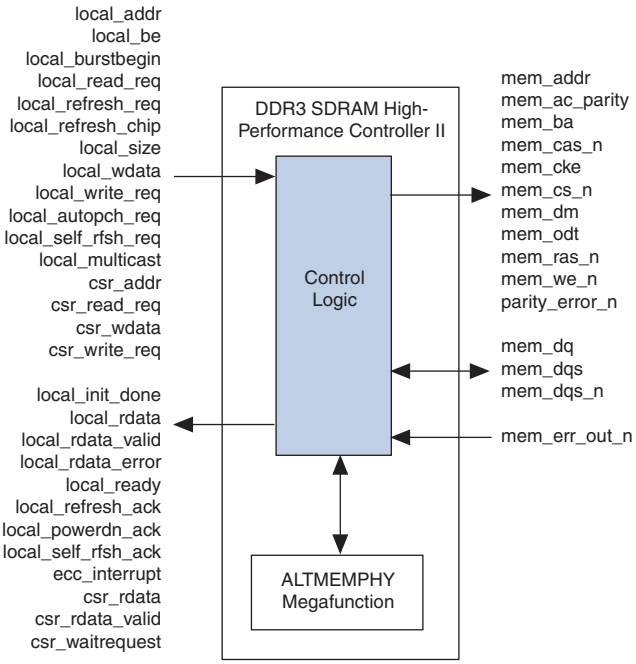
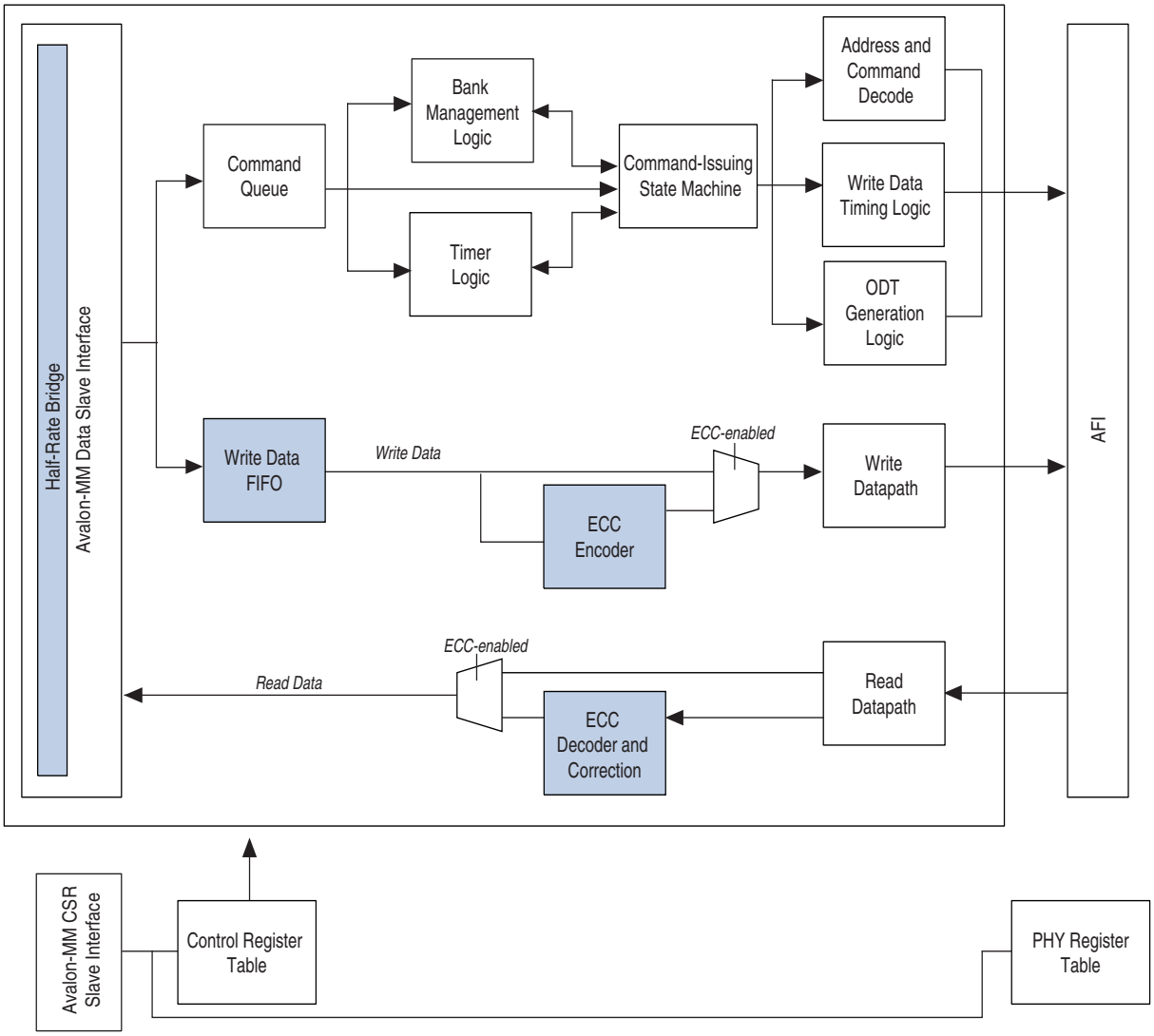


Figure 7–2 shows a block diagram of the DDR3 SDRAM HPC II architecture.

**Figure 7–2. DDR3 SDRAM HPC II Architecture Block Diagram**




The blocks in Figure 7–2 on page 7–3 are described in the following sections.

## Avalon-MM Data Slave Interface

The Avalon-MM data slave interface accepts read and write requests from the Avalon-MM master. The width of the data, `local_wdata` and `local_rdata`, is four times the width of the external memory.

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

- For multiple chip select:

  width = chip bits + row bits + bank bits + column – 2

- For single chip select:

  width = row bits + bank bits + column – 2

For every Avalon transaction, the number of read or write requests cannot exceed the the maximum local burst count of 64. Altera recommends that you set this maximum burst count to match your system master's supported burst count.

## Write Data FIFO Buffer

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

## Command Queue

The command queue allows the controller to buffer up to eight consecutive reads or writes. The command queue presents the next 4, 6, or 8 accesses to the internal logic for the look-ahead bank management. The bank management is more efficient if the look-ahead is deeper, but a deeper queue consumes more resources, and may cause maximum frequency degradation.

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

## Bank Management Logic

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

The bank management logic also includes a reduced bank tracking feature that reduces the controller's resource usage. This feature allows you to reduce the number of bank tracking blocks in the bank management logic. However, the number of bank tracking blocks used is limited because the controller is limited to the number of pages it keeps open at any given time. This limit is determined by the controller’s command queue look-ahead depth.

The reduced bank tracking feature provides the ability for the controller to dynamically allocate a memory bank to be tracked to one of the available bank tracking block. When you do not need the memory bank to be tracked anymore, the memory bank is no longer allocated.

This feature is useful if your design is highly dependent on the total number of banks. Using this feature may have an impact on the memory bus efficiency, but it allows a trade-off between area and efficiency. The higher the number of banks to track, the better the efficiency.

## Timer Logic

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

## Command-Issuing State Machine

The command-issuing state machine decides what DDR commands to issue based on the inputs from the command queue, bank management logic, and timer logic. The DDR3 SDRAM provides half-rate command-issuing state machine. The half-rate state machine supports 2T address and command, and issues "on-the-fly" memory bursts.

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

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

## Address and Command Decode Logic

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

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

## Write and Read Datapath, and Write Data Timing Logic

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

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

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

- `afi_dqs_burst`

- afi_wdata_valid
- afi_wdata
- afi_dm

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

## ODT Generation Logic

The ODT generation logic generates the necessary ODT signals for DDR3 SDRAM HPC II memory devices, based on the scheme recommended by Altera.

Table 7–1 shows which ODT signal on the adjacent DIMM is enabled.

**Table 7–1. ODT**

| DIMM | Chip Select per DIMM | Write or Read On | ODT Enabled (Write) | ODT Enabled (Read) |
|---|---|---|---|---|
| 1 | Single chip select | mem_cs[0] | mem_odt[0] | — *(1)* |
| | Dual chip select | mem_cs[0] | mem_odt[0] | — *(1)* |
| | | mem_cs[1] | mem_odt[1] | — *(1)* |
| 2 | Single chip select | mem_cs[0] | mem_odt[0] and mem_odt[1] | mem_odt[1] |
| | | mem_cs[1] | mem_odt[0] and mem_odt[1] | mem_odt[0] |
| | Dual chip select | mem_cs[0] | mem_odt[0] and mem_odt[2] | mem_odt[2] |
| | | mem_cs[1] | mem_odt[1] and mem_odt[3] | mem_odt[3] |
| | | mem_cs[2] | mem_odt[0] and mem_odt[2] | mem_odt[0] |
| | | mem_cs[3] | mem_odt[1] and mem_odt[3] | mem_odt[1] |

**Note to Table 7–1:**

(1) The controller does not drive the ODT signals during read operation.

## User-Controlled Side-Band Signals

The user-controlled side-band signals consists of the following signals.

### User-Refresh Commands

The user-refresh command enables the request to place the memory into refresh mode. The user-refresh control takes precedence over a read or write request. You can issue up to nine consecutive refresh commands to the selected memory chips. However, if you enable the multi-cast write feature, the user refresh commands are always issued to all chips.

### Multi-Cast Write

The multi-cast write request signal allows you to ask the controller to send the current write requests to all the chip selects. This means that the write data is written to all the ranks in the system. The multi-cast write feature is useful for $t_{RC}$ mitigation where you can cycle through chips to continuously read data without hitting $t_{RC}$. The multi-cast write is not supported for registered DIMM interfaces or when the ECC logic is enabled.

### Low-Power Mode Logic

There are two types of low-power mode logic: the user-controlled self-refresh logic and automatic power-down with programmable time-out logic.

#### User-Controlled Self-Refresh Logic

When you assert the `local_self_rfsh_req` signal, the controller completes all pending reads and writes before it places the memory into self-refresh mode. Once the controller places the memory into self-refresh mode, it responds by asserting the acknowledge signal, `local_self_rfsh_ack`. You can leave the memory in self-refresh mode for as long as you choose.

To bring the memory out of self-refresh mode, you must deassert the request signal, and the controller responds by deasserting the acknowledge signal when the memory is no longer in self-refresh mode.

#### Automatic Power-Down with Programmable Time-Out

The controller automatically places the memory in power-down mode to save power if the requested number of idle controller clock cycles is observed in the controller. The **Auto Power Down Cycles** parameter on the **Controller Settings** tab allows you to specify a range between 1 to 65,535 idle controller clock cycles. The counter for the programmable time-out starts when there are no user read or write requests in the command queue. Once the controller places the memory in power-down mode, it responds by asserting the acknowledge signal, `local_powerdown_ack`.

HPC II supports only precharge power-down mode and not active power-down mode.

## Configuration and Status Register (CSR) Interface

The configuration and status register interface is a 32-bit wide interface that uses the Avalon-MM interface standard. The CSR interface allows you to configure the timing parameters, address widths, and the behavior of the controller for without leveling interfaces. If you do not need this feature, you can disable it and all the programmable settings are fixed to the values configured during the generation process. This interface is synchronous to the controller clock.

The CSR interface is not fully supported for DDR3 SDRAM with leveling interfaces, such as DIMMs.

Refer to Table 7–9 through Table 7–23 in page 7–18 for detailed information about the register maps.

## Error Correction Coding (ECC)

The optional ECC logic comprises an encoder and a decoder-corrector, which can detect and correct single-bit errors, and detect double-bit errors. The ECC logic is available in two widths: 64/72 bit and 32/40 bit. The ECC logic has the following features:

- Hamming code ECC logic that encodes every 64 or 32 bits of data into 72 or 40 bits of codeword.

- A latency increase of one clock for both writes and reads.
- Detects and corrects all single-bit errors.
- Detects all double-bit errors.
- Counts the number of single-bit and double-bit errors.
- Accepts partial writes, which trigger a read-modify-write cycle, for memory devices with DM pins.
- Is able to inject single-bit and double-bit errors to trigger ECC correction for testing and debugging purposes.
- Generates an interrupt signal when an error occurs.

When a single-bit or double-bit error occurs, the ECC logic triggers the `ecc_interrupt` signal to inform you that an ECC error has occurred. When a single-bit error occurs, the ECC logic issues an internal read to the error address, and performs an internal write to write back the corrected data. When a double-bit error occurs, the ECC logic does not do any error correction but it asserts the `local_rdata_error` signal to indicate that the data is incorrect. The `local_rdata_error` signal follows the same timing as the `local_rdata_valid` signal.

Enabling auto-correction allows the ECC logic to hold off all controller pending activities until the correction is completed. You can choose to disable auto-correction and schedule the correction manually when the controller is idle to ensure better system efficiency. To manually correct ECC errors, do the following:

1. When an interrupt occurs, read the `SBE_ERROR` register. When a single-bit error occurs, the `SBE_ERROR` register is equal to one.
2. Read out the `ERR_ADDR` register.
3. Correct the single-bit error by doing one of the following:
   - Issue a dummy write to the memory address stored in the `ERR_ADDR` register. A dummy write is a write request with the `local_be` signal zero, that triggers a partial write which is effectively a read-modify-write event. The partial write corrects the data at that address and writes it back.

   or

   - Enable the `ENABLE_AUTO_CORR` register using the CSR interface and issue a read request to the memory address stored in the `ERR_ADDR` register. The read request triggers auto-error correction to the memory address stored in the `ERR_ADDR` register.

## Partial Writes

The ECC logic supports partial writes. Along with the address, data, and burst signals, the Avalon-MM interface also supports a signal vector, `local_be`, that is responsible for byte-enable. Every bit of this signal vector represents a byte on the data-bus. Thus, a logic low on any of these bits instructs the controller not to write to that particular byte, resulting in a partial write. The ECC code is calculated on all bytes of the data-bus. If any bytes are changed, the ECC code must be recalculated and the new code must be written back to the memory.

For partial writes, the ECC logic performs the following steps:

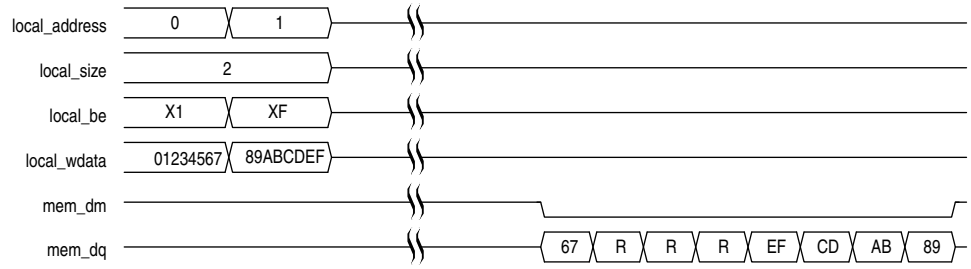
1. The ECC logic sends a read command to the partial write address.
2. Upon receiving a return data from the memory for the particular address, the ECC logic decodes the data, checks for errors, and then merges the corrected or correct dataword with the incoming information.
3. The ECC logic issues a write to write back the updated data and the new ECC code.

The following corner cases can occur:

- A single-bit error during the read phase of the read-modify-write process. In this case, the single-bit error is corrected first, the single-bit error counter is incremented and then a partial write is performed to this corrected decoded data word.
- A double-bit error during the read phase of the read-modify-write process. In this case, the double-bit error counter is incremented and an interrupt is issued. A new write word is written to the location of the error. The ECC status register keeps track of the error information.

Figure 7–3 shows the partial write operation for HPC II.

**Figure 7–3. Partial Write for HPC II**




**Note to Figure 7–3:**

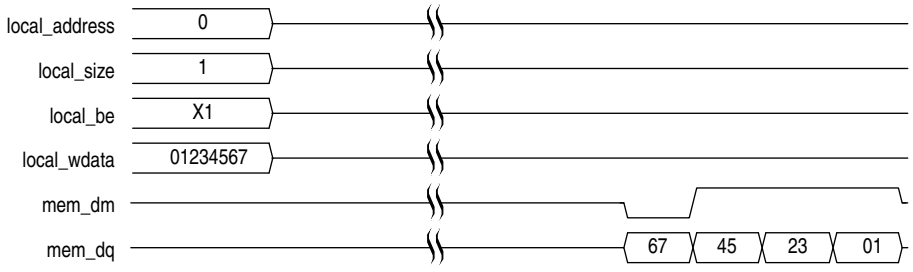
(1) R represents the internal read-back memory data during the read-modify-write process.

## Partial Bursts

DIMMs that do not have the DM pins do not support partial bursts. You must write a minimum of eight words to the memory at the same time.

Figure 7–4 shows the partial burst operation for HPC II.

**Figure 7–4. Partial Burst for HPC II**



# Example Top-Level File

The MegaWizard Plug-In Manager helps you create an example top-level file that shows you how to instantiate and connect the DDR3 SDRAM HPC II. The example top-level file consists of the DDR3 SDRAM HPC II, some driver logic to issue read and write requests to the controller, a PLL to create the necessary clocks, and a DLL (Stratix series only). The example top-level file is a working system that you can compile and use for both static timing checks and board tests.

Figure 7–5 shows the testbench and the example top-level file.

**Figure 7–5. Testbench and Example Top-Level File**

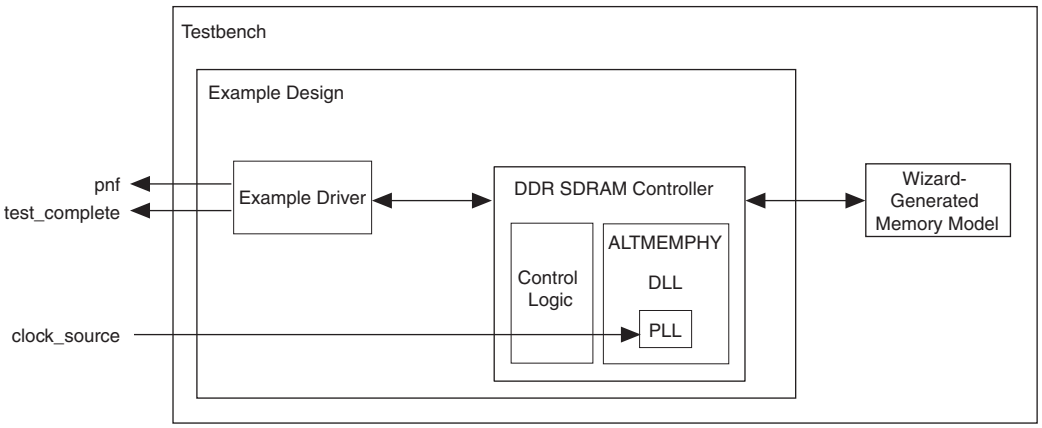



Table 7–2 describes the files that are associated with the example top-level file and the testbench.

**Table 7–2. Example Top-Level File and Testbench Files**

| Filename | Description |
|---|---|
| *<variation name>*_**example_top_tb.v** or **.vhd** | Testbench for the example top-level file. |
| *<variation name>*_**example_top.v** or **.vhd** | Example top-level file. |
| *<variation name>*_**mem_model.v** or **.vhd** | Associative-array memory model. |
| *<variation name>*_**full_mem_model.v** or **.vhd** | Full-array memory model. |
| *<variation name>*_**example_driver.v** or **.vhd** | Example driver. |
| *<variation name>* **.v** or **.vhd** | Top-level description of the custom MegaCore function. |
| *<variation name>***.qip** | Contains Quartus II project information for your MegaCore function variations. |

There are two Altera-generated memory models available—associative-array memory model and full-array memory model.

The associative-array memory model (*<variation name>*_**mem model.v**) allocates reduced set of memory addresses with a default depth of 2,048 or 2K address spaces. This allocation allows for a larger memory array compilation and simulation which enables you to easily reconfigure the depth of the associate array.

The full-array memory model (*<variation name>*_**mem model_full.v**) allocates memory for all addresses accessible by the DDR cores. This allocation makes it impossible to simulate large memory designs.

Both the memory models display similar behaviors and have the same calibration time.

The memory model, *<variation name>*_**test_component.v/vhd**, used in SOPC Builder designs, is actually a variation of the full-array memory model. To ensure your simulation works in SOPC Builder, use memory model with less than 512-Mbit capacity.

## Example Driver

The example driver is a self-checking test pattern generator for the memory interface. It uses a state machine to write and read from the memory to verify that the interface is operating correctly.

The example driver performs the following tests and loops back the tests indefinitely:

- Sequential addressing writes and reads

  The state machine writes pseudo-random data generated by a linear feedback shift register (LFSR) to a set of incrementing row, bank, and column addresses. The state machine then resets the LFSR, reads back the same set of addresses, and compares the data it receives against the expected data. You can adjust the length and pattern of the bursts that are written by changing the `MAX_ROW`, `MAX_BANK`, and `MAX_COL` constants in the example driver source code, and the entire memory space can be tested by adjusting these values. You can skip this test by setting the `test_seq_addr_on` signal to logic zero.

- Incomplete write operation

  The state machine issues a series of write requests that are less than the maximum burst size supported by your controller variation. The addresses are then read back to ensure that the controller has issued the correct signals to the memory. This test is only applicable in full-rate mode, when the local burst size is two. You can skip this test by setting the `test_incomplete_writes_on` signal to logic zero.

- Byte enable/data mask pin operation

  The state machine issues two sets of write commands, the first of which clears a range of addresses. The second set of write commands has only one byte enable bit asserted. The state machine then issues a read request to the same addresses and the data is verified. This test checks if the data mask pins are operating correctly. You can skip this test by setting the `test_dm_pin_on` signal to logic zero.

- Address pin operation

  The example driver generates a series of write and read requests starting with an all-zeros pattern, a walking-one pattern, a walking-zero pattern, and ending with an all-zeros pattern. This test checks to make sure that all the individual address bits are operating correctly. You can skip this test by setting the `test_addr_pin_on` signal to logic zero.

- Low-power mode operation

  The example driver requests the controller to place the memory into power-down and self-refresh states, and hold it in those states for the amount of time specified by the COUNTER_VALUE signal. You can vary this value to adjust the duration the memory is kept in the low-power states. This test is only available if your controller variation enables the low-power mode option.

The example driver has four outputs that allow you to observe which tests are currently running and if the tests are passing. The pass not fail (pnf) signal goes low once one or more errors occur and remains low. The pass not fail per byte (pnf_per_byte) signal goes low when there is incorrect data in a byte but goes back high again once correct data is observed in the following byte. The test_status signal indicates the test that is currently running, allowing you to determine which test has failed. The test_complete signal goes high for a single clock cycle at the end of the set of tests.

Table 7–3 shows the bit mapping for each test status.

**Table 7–3. Test Status[] Bit Mapping**

| Bit | Test |
|---|---|
| 0 | Sequential address test |
| 1 | Incomplete write test |
| 2 | Data mask pin test |
| 3 | Address pin test |
| 4 | Power-down test |
| 5 | Self-refresh test |
| 6 | Auto precharge test |

# Top-level Signals Description

Table 7–4 shows the clock and reset signals.

**Table 7–4. Clock and Reset Signals (Part 1 of 2)**

| Name | Direction | Description |
|---|---|---|
| global_reset_n | Input | The asynchronous reset input to the controller. All other reset signals are derived from resynchronized versions of this signal. This signal holds the complete ALTMEMPHY megafunction, including the PLL, in reset while low. |
| pll_ref_clk | Input | The reference clock input to PLL. |
| phy_clk | Output | The system clock that the ALTMEMPHY megafunction provides to the user. All user inputs to and outputs from the DDR3 HPC II must be synchronous to this clock. |
| reset_phy_clk_n | Output | The reset signal that the ALTMEMPHY megafunction provides to the user. It is asserted asynchronously and deasserted synchronously to phy_clk clock domain. |

**Table 7–4. Clock and Reset Signals (Part 2 of 2)**

| Name | Direction | Description |
|---|---|---|
| aux_full_rate_clk | Output | An alternative clock that the ALTMEMPHY megafunction provides to the user. This clock always runs at the same frequency as the external memory interface. In half-rate mode, this clock is twice the frequency of the phy_clk and can be used whenever a 2x clock is required. In full-rate mode, this clock is driven by the same PLL output as the phy_clk signal. |
| aux_half_rate_clk | Output | An alternative clock that the ALTMEMPHY megafunction provides to the user. This clock always runs at half the frequency as the external memory interface. In full-rate mode, this clock is half the frequency of the phy_clk and can be used, for example to clock the user side of a half-rate bridge. In half-rate mode, or if the **Enable Half Rate Bridge** option is turned on, this clock is driven by the same PLL output that drives the phy_clk signal. |
| dll_reference_clk | Output | Reference clock to feed to an externally instantiated DLL. |
| reset_request_n | Output | Reset request output that indicates when the PLL outputs are not locked. Use this signal as a reset request input to any system-level reset controller you may have. This signal is always low when the PLL is trying to lock, and so any reset logic using it is advised to detect a reset request on a falling edge rather than by level detection. |
| soft_reset_n | Input | Edge detect reset input intended for SOPC Builder use or to be controlled by other system reset logic. It is asserted to cause a complete reset to the PHY, but not to the PLL used in the PHY. |
| oct_ctl_rs_value | Input | ALTMEMPHY signal that specifies the serial termination value. Should be connected to the ALT_OCT megafunction output seriesterminationcontrol. |
| oct_ctl_rt_value | Input | ALTMEMPHY signal that specifies the parallel termination value. Should be connected to the ALT_OCT megafunction output parallelterminationcontrol. |
| dqs_delay_ctrl_import | Input | Allows the use of DLL in another ALTMEMPHY instance in this ALTMEMPHY instance. Connect the export port on the ALTMEMPHY instance with a DLL to the import port on the other ALTMEMPHY instance. |

Table 7–5 shows the DD3 SDRAM HPC II local interface signals.

**Table 7–5. Local Interface Signals (Part 1 of 3)**

| Signal Name | Direction | Description |
|---|---|---|
| local_address[] | Input | Memory address at which the burst should start.<br>By default, the local address is mapped to the bank interleaving scheme. You can change the ordering via the **Local-to-Memory Address Mapping** option in the **Controller Settings** page.<br>The width of this bus is sized using the following equation:<br>For one chip select:<br>width = row bits + bank bits + column bits – 2<br>For multiple chip selects:<br>width = chip bits + row bits + bank bits + column bits – 2<br>If the bank address is 2 bits wide, row is 13 bits wide and column is 10 bits wide, then the local address is 23 bits wide. To map local_address to bank, row and column address:<br>local_address is 23 bits wide<br>local_address[22:10] = row address[12:0]<br>local_address[9:8] = bank address [1:0]<br>local_address [7:0] = column address[9:2]<br>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_be is active high; mem_dm is active low.<br>To map local_wdata and local_be to mem_dq and mem_dm, consider a full-rate design with 32-bit local_wdata and 16-bit mem_dq.<br>Local_wdata = < 22334455 >< 667788AA >< BBCCDDEE ><br>Local_be = < 1100 >< 0110 >< 1010 ><br>These values map to:<br>Mem_dq = <4455><2233><88AA><6677><DDEE><BBCC><br>Mem_dm = <1 1 ><0 0 ><0 1 ><1 0 ><0 1 ><0 1 > |
| local_burstbegin | Input | Avalon burst begin strobe, which indicates the beginning of an Avalon burst. Unlike all other Avalon-MM signals, the burst begin signal does not stay asserted if local_ready is deasserted.<br>For write transactions, assert this signal at the beginning of each burst transfer and keep this signal high for one cycle per burst transfer, even if the slave has deasserted the local_ready signal. This signal is sampled at the rising edge of the phy_clk when the local_write_req signal is asserted. After the slave deasserts the local_ready signal, the master keeps all the write request signals asserted until local_ready signal becomes high again.<br>For read transactions, assert this signal for one clock cycle when read request is asserted and the local_address from which the data should be read is given to the memory. After the slave deasserts the local_ready signal (waitrequest_n in Avalon interface), the master keeps all the read request signals asserted until the local_ready signal becomes high again. |

**Table 7–5. Local Interface Signals (Part 2 of 3)**

| Signal Name | Direction | Description |
|---|---|---|
| local_read_req | Input | Read request signal. You cannot assert the read request signal before the reset_phy_clk_n signal goes high. |
| local_refresh_req | Input | User-controlled refresh request. If **Enable User Auto-Refresh Controls** option is turned on, local_refresh_req becomes available and you are responsible for issuing sufficient refresh requests to meet the memory requirements. This option allows complete control over when refreshes are issued to the memory including grouping together multiple refresh commands. Refresh requests take priority over read and write requests, unless the requests are already being processed. |
| local_refresh_chip | Input | Controls which chip to issue the user-refresh to. This active high signal is used together with the local_refresh_req signal. This signal is as wide as the memory chip select. This signal asserts a high value to each bit that represents the refresh for the corresponding memory chip. For example: If the local_refresh_chip signal is assigned with a value of 4'b0101, the controller refreshes the memory chips 0 and 2, and memory chips 1 and 3 are not refreshed.<br>If you turn on the **Enable Multi-cast Write Control** option, this signal is ignored. |
| local_size[] | Input | Controls the number of beats in the requested read or write access to memory, encoded as a binary number. The range of supported Avalon burst lengths is 1 to 64. The width of this signal is derived based on the burst count specified in the **Local Maximum Burst Count** option. With the derived width, you choose a value ranging from 1 to the local maximum burst count specified. |
| local_wdata[] | Input | Write data bus. The width of local_wdata is four times the memory data bus for a half rate controller. |
| local_write_req | Input | Write request signal. You cannot assert the write request signal before the reset_phy_clk_n signal goes high. |
| local_multicast | Input | In-band multi-cast write request signal. This active high signal is used together with the local_write_req signal. When this signal is asserted high, data is written to all the memory chips available. |
| local_self_rfsh_req | Input | User control of the self-refresh feature. If **Enable Self-Refresh Controls** option is enabled, you can request that the controller place the memory devices into a self-refresh state by asserting this signal. The controller places the memory in the self-refresh state as soon as it can without violating the relevant timing parameters and responds by asserting the local_self_rfsh_ack signal. You can hold the memory in the self-refresh state by keeping this signal asserted. You can release the memory from the self-refresh state at any time by deasserting the local_self_rfsh_req signal and the controller responds by deasserting the local__self_rfsh_ack signal once it has successfully brought the memory out of the self-refresh state. |
| local_init_done | Output | When the memory initialization, training, and calibration are complete, the ALTMEMPHY sequencer asserts the ctrl_usr_mode_rdy signal to the memory controller, which then asserts this signal to indicate that the memory interface is ready to be used.<br>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.<br>This signal does not indicate that the calibration is successful. |

**Table 7–5. Local Interface Signals (Part 3 of 3)**

| Signal Name | Direction | Description |
|---|---|---|
| local_rdata[] | Output | Read data bus. The width of local_rdata is twice that of the memory data bus for a full rate controller; four times the memory data bus for a half rate controller. |
| local_rdata_error | Output | Asserted if the current read data has an error. This signal is only available if the **Enable Error Detection and Correction Logic** option is turned on. This signal is asserted together with the local_rdata_valid signal.<br>If the controller encounters double-bit errors, no correction is made and the controller asserts this signal. |
| local_rdata_valid | Output | Read data valid signal. The local_rdata_valid signal indicates that valid data is present on the read data bus. |
| local_ready | Output | The local_ready signal indicates that the controller is ready to accept request signals. If the local_ready signal is asserted in the clock cycle that a read or write request is asserted, that request is accepted. The local_ready signal is deasserted to indicate that the controller cannot accept any more requests. The controller is able to buffer eight read or write requests. |
| local_refresh_ack | Output | Refresh request acknowledge, which is asserted for one clock cycle every time a refresh is issued. Even if the **Enable User Auto-Refresh Controls** option is not selected, local_refresh_ack still indicates to the local interface that the controller has just issued a refresh command. |
| local_self_rfsh_ack | Output | Self refresh request acknowledge signal. This signal is asserted and deasserted in response to the local_self_rfsh_req signal from the user. |
| local_power_down_ack | Output | Auto power-down acknowledge signal. This signal is asserted for one clock cycle every time auto power-down is issued. |
| ecc_interrupt | Output | Interrupt signal from the ECC logic. This signal is asserted when the ECC feature is turned on, and an error is detected. This signal remains asserted until the user logic clears the error through the CSR interface. |

Table 7–6 shows the DDR3 SDRAM HPC II CSR interface signals.

**Table 7–6. CSR Interface Signals (Part 1 of 2) (Part 1 of 2)**

| Signal Name | Direction | Description |
|---|---|---|
| csr_addr[] | Input | Register map address.The width of csr_addr is 16 bits. |
| csr_be[] | Input | Byte-enable signal, which you use to mask off individual bytes during writes. csr_be is active high. |
| csr_wdata[] | Input | Write data bus. The width of csr_wdata is 32 bits. |
| csr_write_req | Input | Write request signal. You cannot assert csr_write_req and csr_read_req signals at the same time. |
| csr_read_req | Input | Read request signal. You cannot assert csr_read_req and csr_write_req signals at the same time. |
| csr_rdata[] | Output | Read data bus. The width of csr_rdata is 32 bits. |

**Table 7–6. CSR Interface Signals (Part 2 of 2) (Part 2 of 2)**

| Signal Name | Direction | Description |
|---|---|---|
| csr_rdata_valid | Output | Read data valid signal. The csr_rdata_valid signal indicates that valid data is present on the read data bus. |
| csr_waitrequest | Output | The csr_waitrequest signal indicates that the HPC II is busy and not ready to accept request signals. If the csr_waitrequest signal goes high in the clock cycle when a read or write request is asserted, that request is not accepted. If the csr_waitrequest signal goes low, the HPC II is then ready to accept more requests. |

Table 7–7 shows the DDR3 SDRAM HPC II interface signals.

**Table 7–7. DDR3 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 DDR3 SDRAM and captures read data into the Altera device. |
| mem_dqs_n[] | Bidirectional | Inverted memory data strobe signal, which is used together with the mem_dqs signal to improve signal integrity. |
| mem_clk *(1)* | Bidirectional | Clock for the memory device. |
| mem_clk_n *(1)* | Bidirectional | Inverted clock for the memory device. |
| mem_addr[] | Output | Memory address bus. |
| mem_ac_parity *(2)* | Output | Address or command parity signal generated by the PHY and sent to the DIMM. |
| mem_ba[] | Output | Memory bank address bus. |
| mem_cas_n | Output | Memory column address strobe signal. |
| mem_cke[] | Output | Memory clock enable signals. |
| mem_cs_n[] | Output | Memory chip select signals. |
| mem_dm[] | Output | Memory data mask signal, which masks individual bytes during writes. |
| mem_odt | Output | Memory on-die termination control signal. |
| mem_ras_n | Output | Memory row address strobe signal. |
| mem_we_n | Output | Memory write enable signal. |
| parity_error_n *(2)* | Output | Active-low signal that is asserted when a parity error occurs and stays asserted until the PHY is reset. |
| mem_err_out_n *(2)* | Input | Signal sent from the DIMM to the PHY to indicate that a parity error has occured for a particular cycle. |

**Notes to Table 7–7:**

(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) I/Os to support the mimic path structure that the ALTMEMPHY megafunction uses.

(2) This signal is for Registered DIMMs only.

Table 7–8 shows the ALTMEMPHY Debug interface signals, which are located in <*variation_name*>_**phy.v/vhd** file.

**Table 7–8. ALTMEMPHY Debug Interface Signals**

| Signal Name | Direction | Description |
|---|---|---|
| dbg_clk | Input | Debug interface clock |
| dbg_addr | Input | Debug interface address |
| dbg_cs | Input | Debug interface chip select |
| dbg_wr | Input | Debug interface write request |
| dbg_wr_data | Input | Debug interface write data |
| dbg_rd | Input | Debug interface read request |
| dbg_rd_data | Input | Debug interface read data |
| dbg_waitrequest | Output | Debug interface wait request |

# Register Maps Description

Table 7–9 through Table 7–23 show the register maps for the DDR3 SDRAM HPC II.

**Table 7–9. Register Map**

| Address | Contents |
|---|---|
| **ALTMEMPHY Register Map** | |
| 0x005 | Mode register 0-1 |
| 0x006 | Mode register 2-3 |
| **Controller Register Map** | |
| 0x100 | ALTMEMPHY status and control register |
| 0x110 | Controller status and configuration register |
| 0x120 | Memory address size register 0 |
| 0x121 | Memory address size register 1 |
| 0x122 | Memory address size register 2 |
| 0x123 | Memory timing parameters register 0 |
| 0x124 | Memory timing parameters register 1 |
| 0x125 | Memory timing parameters register 2 |
| 0x126 | Memory timing parameters register 3 |
| 0x130 | ECC control register |
| 0x131 | ECC status register |
| 0x132 | ECC error address register |

## ALTMEMPHY Register Map

The ALTMEMPHY register map allows you to control the memory components' mode register settings. To access the ALTMEMPHY register map, connect the ALTMEMPHY Debug interface signals in Table 7–8 using the Avalon-MM protocol.

After configuring the ALTMEMPHY register map, initialize a calibration request by setting bit 2 in the CSR register map address 0x100 (Table 7–12) for the mode register settings to take effect.

☞ DDR3 SDRAM with leveling does not support the ALTMEMPHY register map. For more information about DDR3 SDRAM with leveling, refer to "DDR3 SDRAM With Leveling" on page 5–7.

**Table 7–10. Address 0x005 Mode Register 0-1 (Part 1 of 2)**

| Bit | Name | Default | Access | Description |
|---|---|---|---|---|
| 2:0 | Burst length | 8 | RO | This value is set to 8 because the DDR3 SDRAM HPC II only supports a burst length of 8. |
| 3 | BT | 0 | RO | This value is set to 0 because DDR3 SDRAM SDRAM HPC II only supports sequential bursts. |
| 6:4 | CAS latency | — | RW | CAS latency setting. The default value for these bits is set by the MegaWizard CAS Latency setting for your controller instance. You must set this value in the CSR interface register map 0x126 (Table 7–20) as well. |
| 7 | Reserved | 0 | — | Reserved for future use. |
| 8 | DLL | 0 | RW | Not used by the controller, but you can set and programm into the memory device mode register. |
| 11:9 | Write recovery | — | RW | Write recovery ($t_{WR}$) setting. The default value for these bits is set by the MegaWizard Write Recovery ($t_{WR}$) setting for your controller instance. You must set this value in CSR interface register map 0x126 (Table 7–20) as well. |
| 12 | PD | 0/1 | RO | This value is set to 0 because DDR3 SDRAM HPC II only supports power-down fast exit mode. |
| 15:13 | Reserved | 0 | — | Reserved for future use. |
| 16 | DLL | 0 | RW | Not used by the controller, but you can set and program into the memory device mode register. |
| 17 | ODS | 0 | RW | |
| 18 | RTT | 0 | RW | |
| 21:19 | AL | — | RW | Additive latency setting. The default value for these bits is set by the MegaWizard Additive Latency setting for your controller instance. You must set this value in CSR interface register map 0x126 (Table 7–20) as well. |

**Table 7–10. Address 0x005 Mode Register 0-1 (Part 2 of 2)**

| Bit | Name | Default | Access | Description |
|---|---|---|---|---|
| 22 | RTT | 0 | RW | Not used by the controller, but you can set and program into the memory device mode register. |
| 25:23 | RTT/WL/OCD | 0 | RW | |
| 26 | DQS# | 0 | RW | |
| 27 | TDQS/RDQS | 0 | RW | |
| 28 | QOFF | 0 | RW | |
| 31:29 | Reserved | 0 | — | Reserved for future use. |

**Table 7–11. Address 0x006 Mode Register 2-3**

| Bit | Name | Default | Access | Description |
|---|---|---|---|---|
| 2:0 | Reserved | 0 | — | Reserved for future use. |
| 5:3 | CWL | — | RW | CAS write latency setting. The default value for these bits is set by the MegaWizard CAS Write Latency setting for your controller instance. You must set this value in the CSR interface register map 0x126 (Table 7–20) as well. |
| 6 | ASR | 0 | RW | Not used by the controller, but you can set and program into the memory device mode register. |
| 7 | SRT/ET | 0 | RW | |
| 8 | Reserved | 0 | — | Reserved for future use. |
| 10:9 | RTT_WR | 0 | RW | Not used by the controller, but you can set and program into the memory device mode register. |
| 15:11 | Reserved | 0 | — | Reserved for future use. |
| 17:16 | MPR_RF | 0 | RW | Not used by the controller, but you can set and program into the memory device mode register. |
| 18 | MPR | 0 | RW | |
| 31:19 | Reserved | 0 | — | Reserved for future use. |

## Controller Register Map

The controller register map allows you to control the memory controller settings. To access the controller register map, connect the CSR interface signals in Table 7–6 using the Avalon-MM protocol.

**Table 7–12. Address 0x100 ALTMEMPHY Status and Control Register**

| Bit | Name | Default | Access | Description |
|---|---|---|---|---|
| 0 | CAL_SUCCESS | — | RO | This bit reports the value of the ALTMEMPHY `ctl_cal_success` output. Writing to this bit has no effect. |
| 1 | CAL_FAIL | — | RO | This bit reports the value of the ALTMEMPHY `ctl_cal_fail` output. Writing to this bit has no effect. |
| 2 | CAL_REQ | 0 | RW | Writing a 1 to this bit asserts the `ctl_cal_req` signal to the ALTMEMPHY megafunction. Writing a 0 to this bit deaasserts the signal, and the ALTMEMPHY megafunction will then initiate its calibration sequence.<br>CAUTION: You must not use this register during the ALTMEMPHY megafunction calibration. You must wait until the CAL_SUCCESS or CAL_FAIL register shows a value of 1. |
| 7:3 | Reserved | 0 | — | Reserved for future use. |
| 13:8 | CLOCK_OFF | 0 | RW | Writing a 1 to any of the bits in this register causes the appropriate `ctl_mem_clk_disable` signal to the ALTMEMPHY megafunction to be asserted, which disables the memory clock outputs. Writing a 0 to this register deasserts the signal and re-enables the memory clocks. The ALTMEMPHY megafunction supports up to 6 individual memory clocks, each bit will represent each individual clock. |
| 30:14 | Reserved | 0 | — | Reserved for future use. |

**Table 7–13. Address 0x110 Controller Status and Configuration Register (Part 1 of 2)**

| Bit | Name | Default | Access | Description |
|---|---|---|---|---|
| 15:0 | AUTO_PD_CYCLES | 0x0 | RW | The number of idle clock cycles after which the controller should place the memory into power-down mode. The controller is considered to be idle if there are no commands in the command queue. Setting this register to 0 disables the auto power-down mode. The default value of this register depends on the values set during the generation of the design. |
| 16 | AUTO_PD_ACK | 1 | RO | This bit indicates that the memory is in power-down state. |

**Table 7–13. Address 0x110 Controller Status and Configuration Register (Part 2 of 2)**

| Bit | Name | Default | Access | Description |
|---|---|---|---|---|
| 17 | SELF_RFSH | 0 | RW | Setting this bit, or asserting the `local_self_rfsh` signal, causes the memory to go into self-refresh state. |
| 18 | SELF_RFSH-ACK | 0 | RO | This bit indicates that the memory is in self-refresh state. |
| 19 | Reserved | 0 | — | Reserved for future use. |
| 21:20 | ADDR_ORDER | 00 | RW | 00 - Chip, row, bank, column.<br>01 - Chip, bank, row, column.<br>10 - Reserved for future use.<br>11 - Reserved for future use. |
| 22 | REGDIMM | 0 | RW | Setting this bit to 1 enables REGDIMM support in controller. |
| 24:23 | CTRL_DRATE | 00 | RO | These bits represent controller date rate:<br>00 - Full rate.<br>01 - Half rate.<br>10 - Reserved for future use.<br>11 - Reserved for future use. |
| 30:24 | Reserved | 0 | — | Reserved for future use. |

**Table 7–14. Address 0x120 Memory Address Size Register 0**

| Bit | Name | Default | Access | Description |
|---|---|---|---|---|
| 7:0 | Column address width | — | RW | The number of column address bits for the memory devices in your memory interface. The range of legal values is 7-12. |
| 15:8 | Row address width | — | RW | The number of row address bits for the memory devices in your memory interface. The range of legal values is 12-16. |
| 19:16 | Bank address width | — | RW | The number of bank address bits for the memory devices in your memory interface. The range of legal values is 2-3. |
| 23:20 | Chip select address width | — | RW | The number of chip select address bits for the memory devices in your memory interface. The range of legal values is 0-2. If there is only one single chip select in the memory interface, set this bit to 0. |
| 31:24 | Reserved | 0 | — | Reserved for future use. |

**Table 7–15. Address 0x121 Memory Address Size Register 1**

| Bit | Name | Default | Access | Description |
|---|---|---|---|---|
| 31:0 | Data width representation (word) | — | RW | The number of DQS bits in the memory interface. This bit can be used to derive the width of the memory interface by multiplying this value by the number of DQ pins per DQS pin (typically 8). |

**Table 7–16. Address 0x122 Memory Address Size Register 2**

| Bit | Name | Default | Access | Description |
|---|---|---|---|---|
| 7:0 | Chip select representation | — | RW | The number of chip select in binary representation.<br>For example, a design with 2 chip selects has the value of 00000011. |
| 31:8 | Reserved | 0 | — | Reserved for future use. |

**Table 7–17. Address 0x123 Memory Timing Parameters Register 0**

| Bit | Name | Default | Access | Description |
|---|---|---|---|---|
| 3:0 | $t_{RCD}$ | — | RW | The activate to read or write a timing parameter. The range of legal values is 2-11 cycles. |
| 7:4 | $t_{RRD}$ | — | RW | The activate to activate a timing parameter. The range of legal values is 2-8 cycles. |
| 11:8 | $t_{RP}$ | — | RW | The precharge to activate a timing parameter. The range of legal values is 2-11 cycles. |
| 15:12 | $t_{MRD}$ | — | RW | The mode register load time parameter. This value is not used by the controller, as the controller derives the correct value from the memory type setting. |
| 23:16 | $t_{RAS}$ | — | RW | The activate to precharge a timing parameter. The range of legal values is 4-29 cycles. |
| 31:24 | $t_{RC}$ | — | RW | The activate to activate a timing parameter. The range of legal values is 8-40 cycles. |

**Table 7–18. Address 0x124 Memory Timing Parameters Register 1**

| Bit | Name | Default | Access | Description |
|---|---|---|---|---|
| 3:0 | $t_{WTR}$ | — | RW | The write to read a timing parameter. The range of legal values is 1-10 cycles. |
| 7:4 | $t_{RTP}$ | — | RW | The read to precharge a timing parameter. The range of legal values is 2-8 cycles. |
| 15:8 | $t_{FAW}$ | — | RW | The four-activate window timing parameter. The range of legal values is 6-32 cycles. |
| 31:16 | Reserved | 0 | — | Reserved for future use. |

**Table 7–19. Address 0x125 Memory Timing Parameters Register 2**

| Bit | Name | Default | Access | Description |
|---|---|---|---|---|
| 15:0 | $t_{REFI}$ | — | RW | The refresh interval timing parameter. The range of legal values is 780-6240 cycles. |
| 23:16 | $t_{RFC}$ | — | RW | The refresh cycle timing parameter. The range of legal values is 12-88 cycles. |
| 31:24 | Reserved | 0 | — | Reserved for future use. |

**Table 7–20. Address 0x126 Memory Timing Parameters Register 3**

| Bit | Name | Default | Access | Description |
|---|---|---|---|---|
| 3:0 | CAS latency, $t_{CL}$ | — | RW | This value must be set to match the memory CAS latency. You must set this value in the 0x04 register map as well. |
| 7:4 | Additive latency, AL | — | RW | Additive latency setting. The default value for these bits is set in the **Memory additive CAS latency setting** in the **Preset Editor** dialog box. You must set this value in the 0x05 register map as well. |
| 11:8 | CAS write latency, CWL | — | RW | CAS write latency setting. You must set this value in the 0x06 register map as well. |
| 15:12 | Write recovery, $t_{WR}$ | — | RW | This value must be set to match the memory write recovery time ($t_{WR}$). You must set this value in the 0x04 register map as well. |
| 31:16 | Reserved | 0 | — | Reserved for future use. |

**Table 7–21. Address 0x130 ECC Control Register**

| Bit | Name | Default | Access | Description |
|---|---|---|---|---|
| 0 | ENABLE_ECC | 1 | RW | When 1, enables the generation and checking of ECC. |
| 1 | ENABLE_AUTO_CORR | — | RW | When 1, enables auto-correction when a single-bit error is detected. |
| 2 | GEN_SBE | 0 | RW | When 1, enables the deliberate insertion of single-bit errors, bit 0, in the data written to memory. This bit is only used for testing purposes. |
| 3 | GEN_DBE | 0 | RW | When 1, enables the deliberate insertion of double-bit errors, bits 0 and 1, in the data written to memory. This bit is only used for testing purposes. |
| 4 | ENABLE_INTR | 1 | RW | When 1, enables the interrupt output. |
| 5 | MASK_SBE_INTR | 0 | RW | When 1, masks the single-bit error interrupt. |
| 6 | MASK_DBE_INTR | 0 | RW | When 1, masks the double-bit error interrupt |
| 7 | CLEAR | 0 | RW | When 1, writing to this self-clearing bit clears the interrupt signal, and the error status and error address registers. |
| 9 | Reserved | 0 | — | Reserved for future use. |

**Table 7–22. Address 0x131 ECC Status Register (Part 1 of 2)**

| Bit | Name | Default | Access | Description |
|---|---|---|---|---|
| 0 | SBE_ERROR | 0 | RO | Set to 1 when any single-bit errors occur. |
| 1 | DBE_ERROR | 0 | RO | Set to 1 when any double-bit errors occur. |
| 7:2 | Reserved | 0 | — | Reserved for future use. |

**Table 7–22. Address 0x131 ECC Status Register (Part 2 of 2)**

| Bit | Name | Default | Access | Description |
|---|---|---|---|---|
| 15:8 | SBE_COUNT | 0 | RO | Reports the number of single-bit errors that have occurred since the status register counters were last cleared. |
| 23:16 | DBE_COUNT | 0 | RO | Reports the number of double-bit errors that have occurred since the status register counters were last cleared. |
| 31:24 | Reserved | 0 | — | Reserved for future use. |

**Table 7–23. Address 0x132 ECC Error Address Register**

| Bit | Name | Default | Access | Description |
|---|---|---|---|---|
| 31:0 | ERR_ADDR | 0 | RO | The address of the most recent ECC error. This address contains concatenation of chip, bank, row, and column addresses. |

# 8. Latency

Latency is defined using the local (user) side frequency and absolute time (ns). There are two types of latencies that exists while designing with memory controllers—read and write latencies, which have the following definitions:

- Read latency—the amount of time it takes for the read data to appear at the local interface after initiating the read request.
- Write latency—the amount of time it takes for the write data to appear at the memory interface after initiating the write request.

For a half-rate controller, the local side frequency is half of the memory interface frequency.

Altera defines read and write latencies in terms of the local interface clock frequency and by the absolute time for the memory controllers. These latencies apply to supported device families with the half-rate DDR3 high-performance controllers (HPC and HPC II).

The latency defined in this section uses the following assumptions:

- The row is already open, there is no extra bank management needed.
- The controller is idle, there is no queued transaction pending, indicated by the `local_ready` signal asserted high.
- No refresh cycles occur before the transaction.

The latency for the high-performance controller comprises many different stages of the memory interface.

Figure 8–1 shows a typical memory interface read latency path showing read latency from the time a `local_read_req` assertion is detected by the controller up to data available to be read from the dual-port RAM (DPRAM) module.

**Figure 8–1. Typical Latency Path**

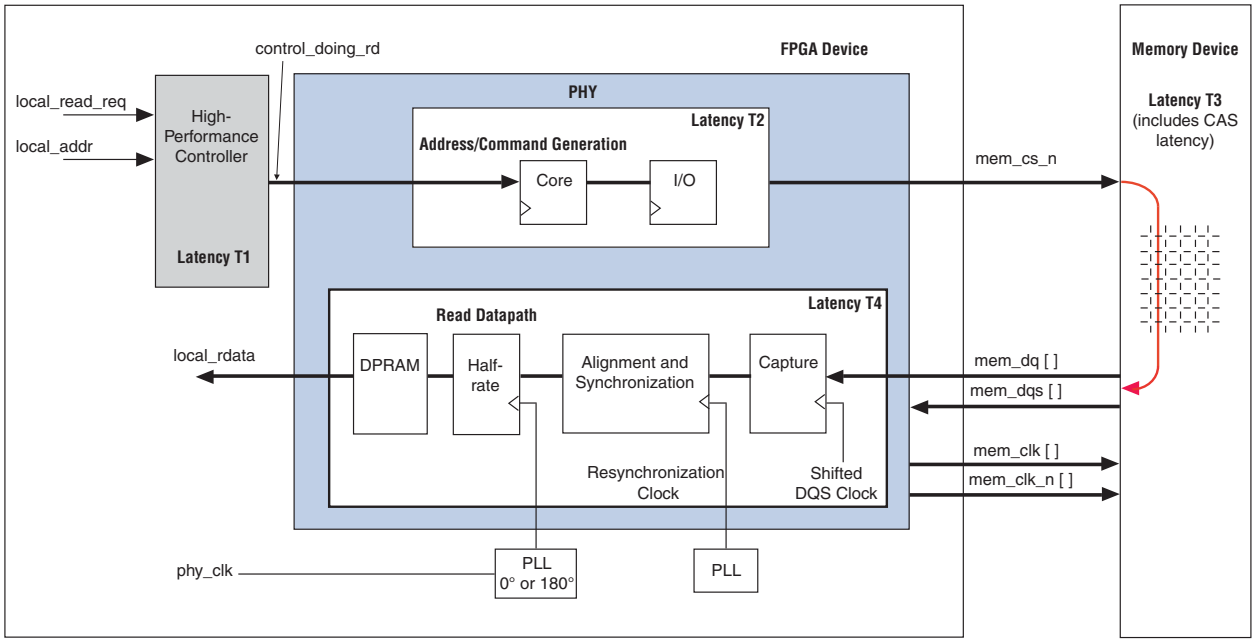



Table 8–1 shows the different stages that make up the whole read and write latency that Figure 8–1 shows.

**Table 8–1. High-Performance Controller Latency Stages and Descriptions**

| Latency Number | Latency Stage | Description |
|---|---|---|
| T1 | Controller | `local_read_req` or `local_write_req` signal assertion to `ddr_cs_n` signal assertion. |
| T2 | Command Output | `ddr_cs_n` signal assertion to `mem_cs_n` signal assertion. |
| T3 | CAS or WL | Read command to DQ data from the memory or write command to DQ data to the memory. |
| T4 | ALTMEMPHY read data input | Read data appearing on the local interface. |
| T2 + T3 | Write data latency | Write data appearing on the memory interface. |

From Figure 8–1, the read latency in the high-performance controllers is made up of four components:

read latency = controller latency (T1) + command output latency (T2) + CAS latency (T3) + PHY read data input latency (T4)

Similarly, the write latency in the high-performance controllers is made up of three components:

write latency = controller latency (T1) + write data latency (T2+T3)

You can separate the controller and ALTMEMPHY read data input latency into latency that occurred in the I/O element (IOE) and latency that occurred in the FPGA fabric.

Table 8–2 shows the read and write latency derived from the write and read latency definitions for half rate controller for Stratix III and Stratix IV devices.

The exact latency depends on your precise configuration. You should obtain precise latency from simulation, but this figure may vary in hardware because of the automatic calibration process.

**Table 8–2. Typical Latency**

| Device | Controller Rate | Frequency (MHz) | Latency Type | Total Latency | |
|---|---|---|---|---|---|
| | | | | Local Clock Cycles | Time (ns) |
| Stratix III | Half | 400 | Read | 23 | 115 |
| | | | Write | 14 | 68 |
| Stratix IV | Half | 400 | Read | 23 | 115 |
| | | | Write | 14 | 68 |

To see the latency incurred in the IOE for both read and write paths for ALTMEMPHY variations in Stratix IV and Stratix III devices refer to the IOE figures in the *External Memory Interfaces in Stratix III Devices* chapter of the *Stratix III Device Handbook* and the *External Memory Interfaces in Stratix IV Devices* chapter of the *Stratix IV Device Handbook*.

# 9. Timing Diagrams

This chapter details the timing diagrams for the DDR3 SDRAM high-performance controllers (HPC) and high-performance controllers II (HPC II).

## DDR3 High-Performance Controllers

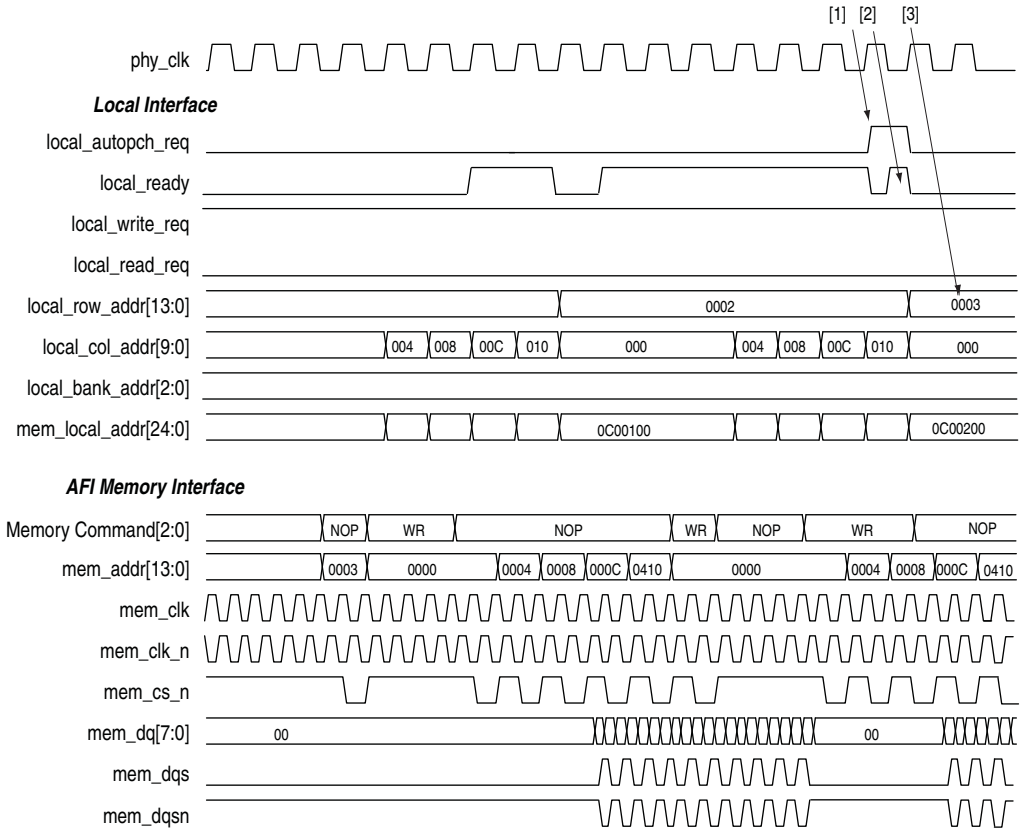
This section discusses the following timing diagrams for HPC in AFI mode:

- "Auto-Precharge"
- "User Refresh"
- "Half-Rate Read for Avalon Interface"
- "Half-Rate Write for Avalon Interface"
- "Half Rate Write for Native Interface"
- "Initialization Timing for HPC"
- "Calibration Timing for HPC"

## Auto-Precharge

The auto-precharge read and auto-precharge write commands allow you to indicate to the memory device that this read or write command is the last access to the currently open row. The memory device automatically closes (auto-precharges) the page it is currently accessing so that the next access to the same bank is quicker. This command is particularly useful for applications that require fast random accesses.

**Figure 9–1. Auto-Precharge Operation for HPC**




**Notes to Figure 9–1:**

(1) The auto-precharge request goes high.

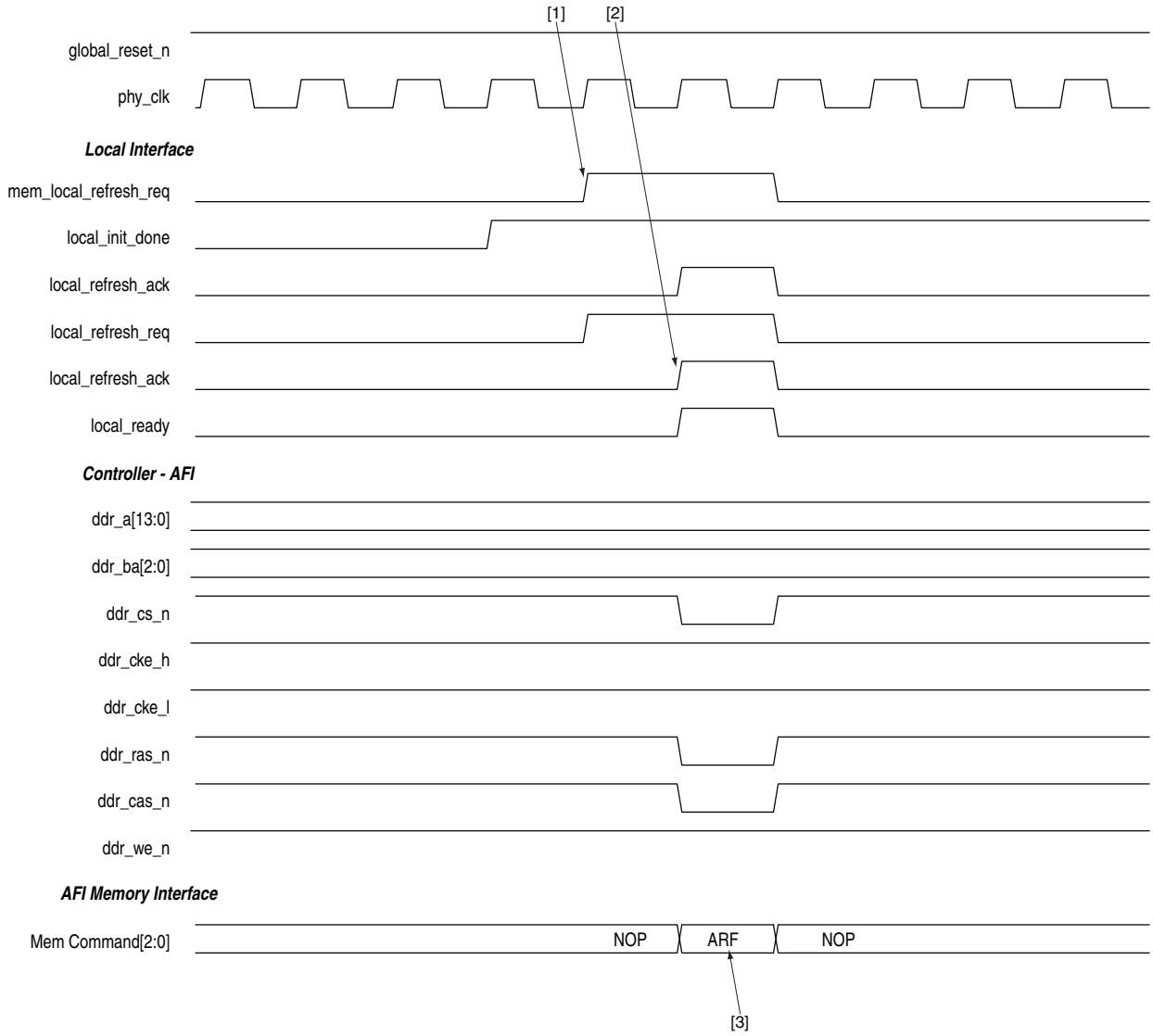
(2) The `local_ready` signal is asserted and remains high until the auto-precharge request goes low.

(3) A new row address begins.

## User Refresh

Figure 9–2 shows the user refresh control interface. This feature allows you to control when the controller issues refreshes to the memory. This feature allows better control of worst case latency and allows refreshes to be issued in bursts to take advantage of idle periods.

**Figure 9–2. User-Refresh Operation for HPC**

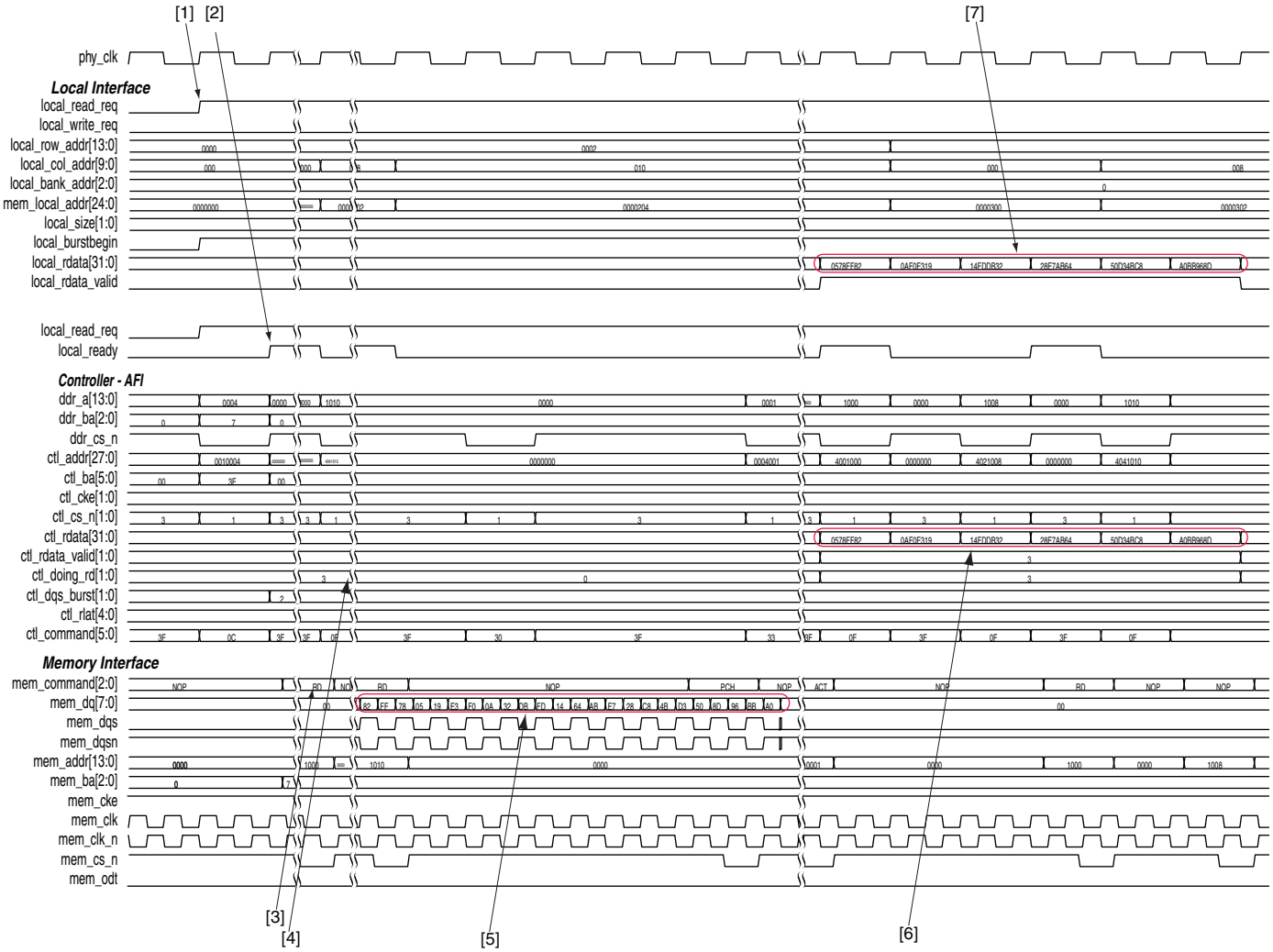



**Notes to Figure 9–2:**

(1) The local refresh request signal is asserted.
(2) The controller asserts the `local_refresh_ack` signal.
(3) The auto-refresh (ARF) command on the command bus.

# Half-Rate Read for Avalon Interface

**Figure 9–3. Half-Rate Read Operation for HPC Using Avalon-MM Interface**

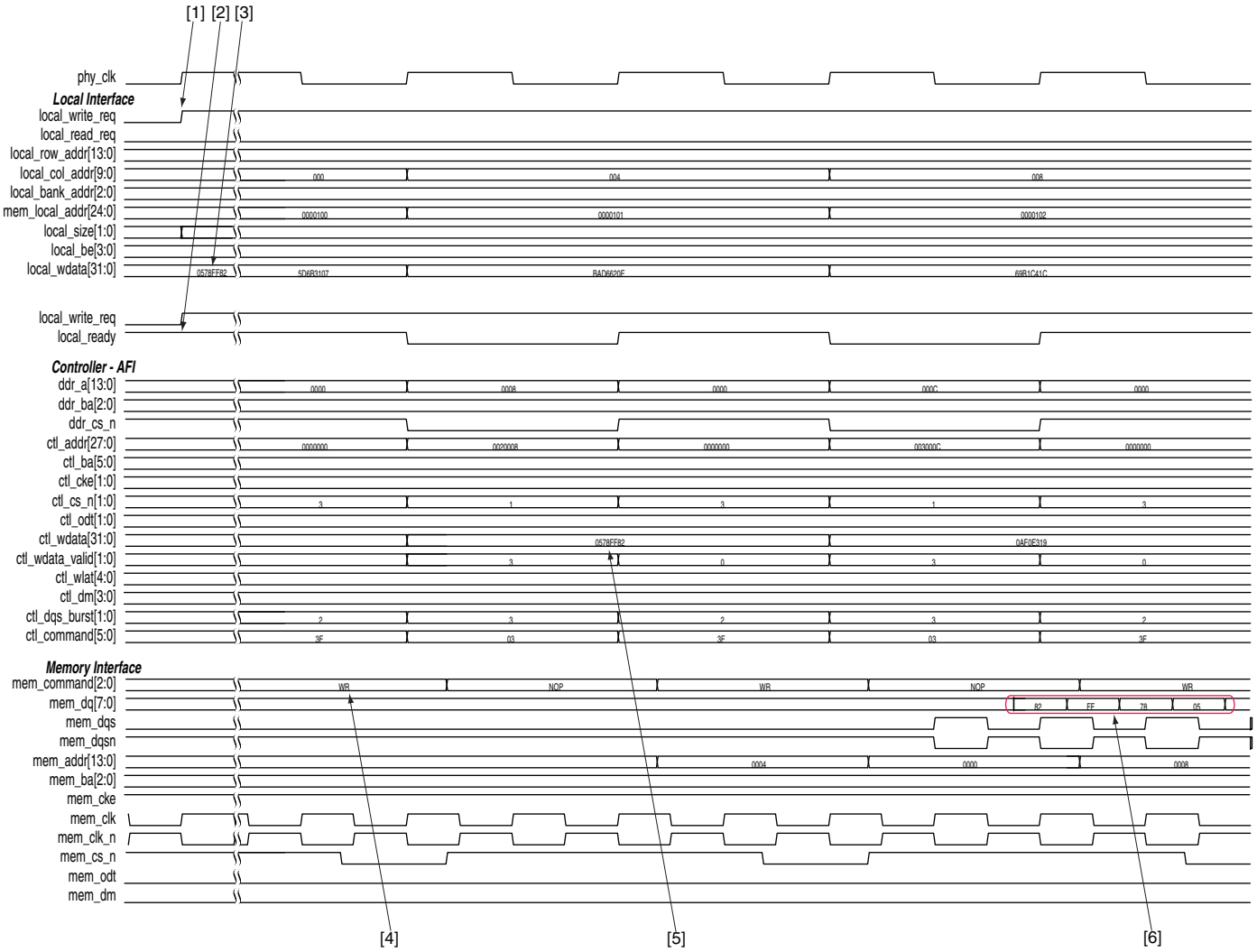


The following sequence corresponds with the numbered items in Figure 9–3:

1. The local read request signal is asserted.
2. The controller accepts the request, the `local_ready` signal is asserted.
3. The controller asserts the `ctl_doing_rd` to tell the PHY how many clock cycles of read data to expect.
4. The read command (RD) on the command bus.
5. The `mem_dqs` signal has the read data from the controller.
6. These are the data to the controller with the valid signal.
7. The controller returns the valid read data to the user logic by asserting the `local_rdata_valid` signal when there is valid local read data.

# Half-Rate Write for Avalon Interface

**Figure 9–4. Half-Rate Write Operation for HPC Using Avalon Interface**

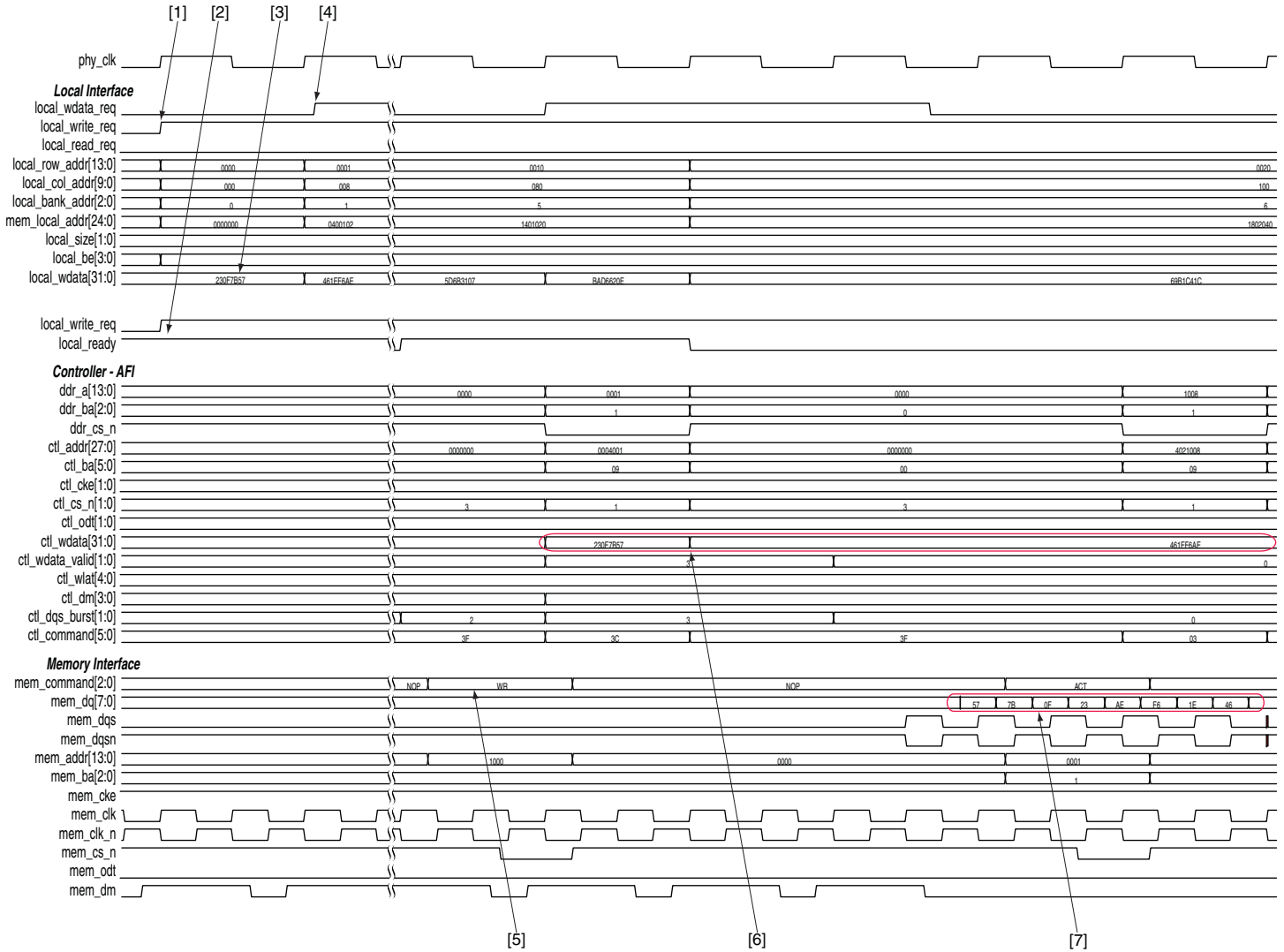


The following sequence corresponds with the numbered items in Figure 9–4:

1. The user logic requests write by asserting the `local_write_req` signal.
2. The `local_ready` signal is asserted, indicating that the controller has accepted the request.
3. The data written to the memory for the write command.
4. The write (WR) command on the command bus.
5. The valid write data on the `ctl_wdata` signal.
6. The valid data on the `mem_dq` signal goes to the controller.

# Half Rate Write for Native Interface

**Figure 9–5. Half-Rate Write Operation for HPC Using Native Interface**



The following sequence corresponds with the numbered items in Figure 9–5:

1. The user logic requests write by asserting the `local_write_req` signal.
2. The `local_ready` signal is asserted, indicating that the controller has accepted the request.
3. The data written to the memory for the write command.
4. The controller requests the user logic for the write data and byte-enables for the write by asserting the `local_wdata_req` signal.
5. The write (WR) command on the command bus.
6. The valid write data on the `ctl_wdata` signal.
7. The valid data on the `mem_dq` signal goes to the controller.

# Initialization Timing

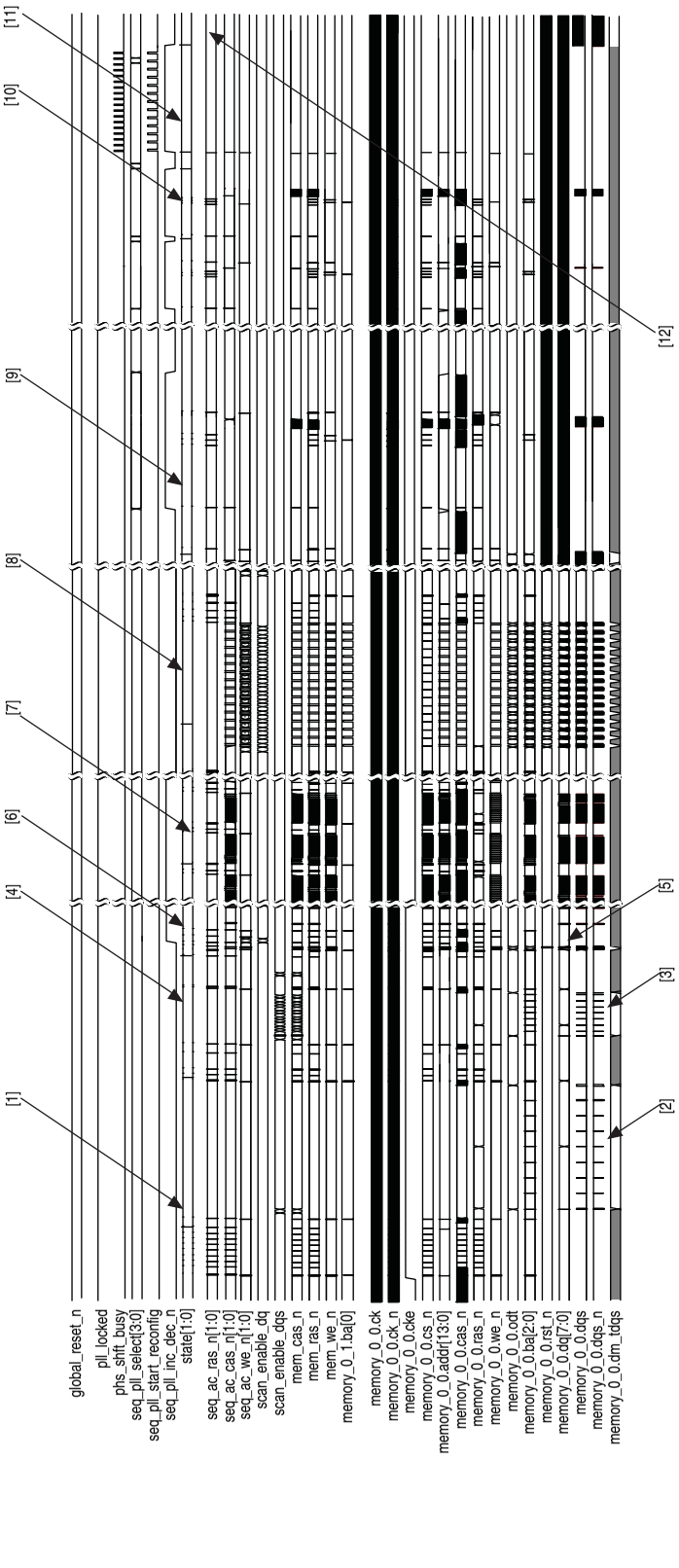
**Figure 9–6. Initialization Timing for HPC**

The following sequence corresponds with the numbered items in Figure 9–6:

1. The PHY initialization stage; wait for PLL to unlock.
2. The DRAM initialization stage; reset sequence.
3. Various SDRAM bus commands during the initialization sequence.

## Calibration Timing

**Figure 9–7. Calibration Timing for HPC**



The following sequence corresponds with the numbered items in Figure 9–7:

1. The write leveling stage.
2. The write leveling coarse phase sweep.
3. Fine T9/T10 delay chain sweep.
4. The write burst training pattern.
5. Three training patterns available at different addresses—zeroes, ones, and mixed.
6. The read path setup starts with the first operation, read deskew.
7. The read path deskew increases capture margin.
8. The write deskew stage; patterns written to RAM and read back.
9. The write datapath setup; data written to DRAM to determine latency.
10. Advertise read and write latency stage.
11. Tracking setup stage to set up mimic window.
12. Calibration successful on user mode.

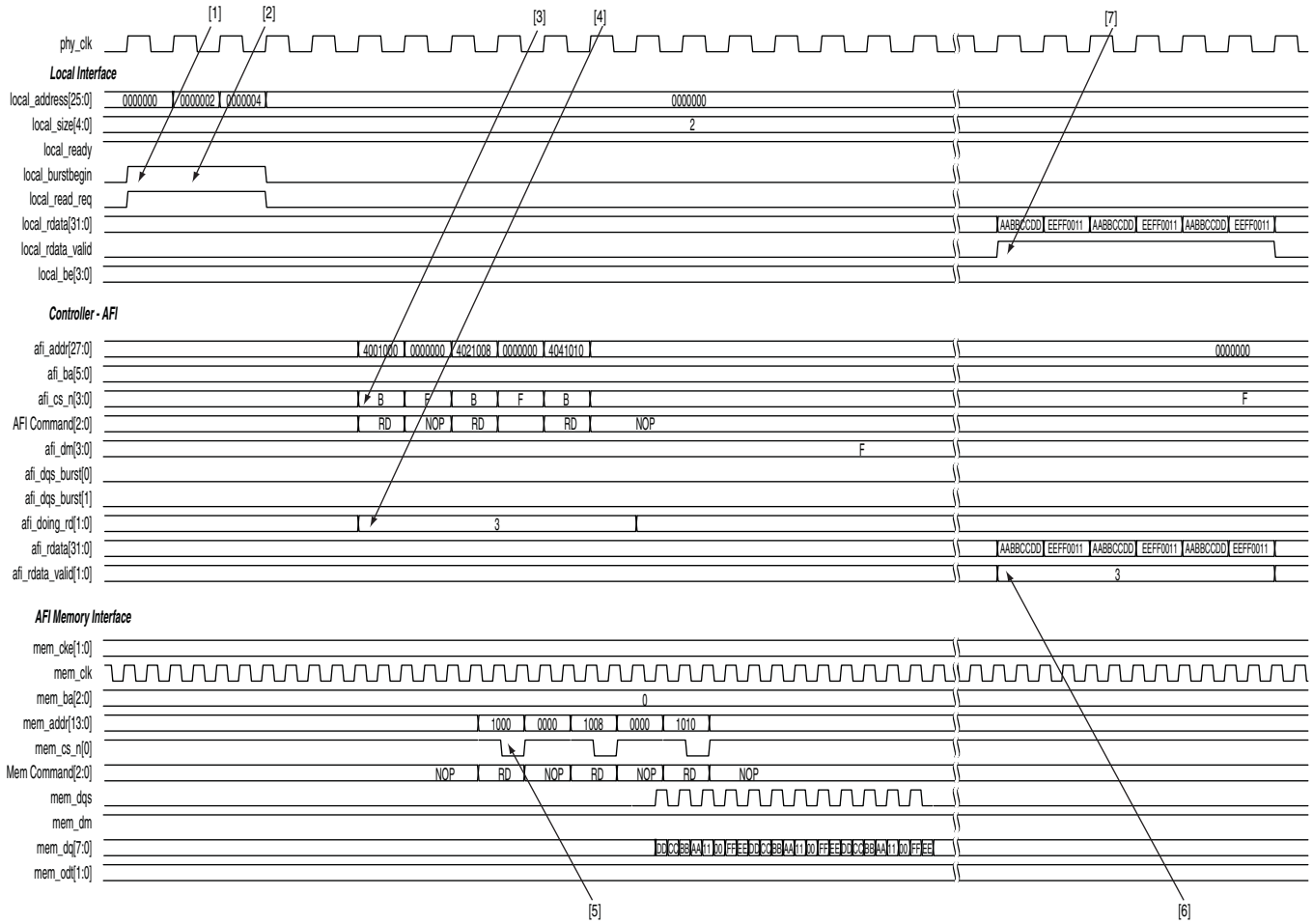
## DDR3 High-Performance Controllers II

This section discusses the following timing diagrams for HPC II:

- "Half-Rate Read (Burst-Aligned Address)"
- "Half-Rate Write (Burst-Aligned Address)"
- "Half-Rate Read (Non Burst-Aligned Address)"
- "Half-Rate Write (Non Burst-Aligned Address)"
- "Half-Rate Read With Gaps"
- "Half-Rate Write With Gaps"
- "Half-Rate Write Operation (Merging Writes)"
- "Write-Read-Write-Read Operation"

# Half-Rate Read (Burst-Aligned Address)

**Figure 9–8. Half-Rate Read Operation for HPC II—Burst-Aligned Address**



The following sequence corresponds with the numbered items in Figure 9–8:

1. The user logic requests the first read by asserting the `local_read_req` signal, and the size and address for this read. In this example, the request is a burst of length of 2 to the local address `0x000000`. This local address is mapped to the following memory address in half-rate mode:

   ```
   mem_row_address = 0x000000

   mem_col_address = 0x0000

   mem_bank_address = 0x00
   ```

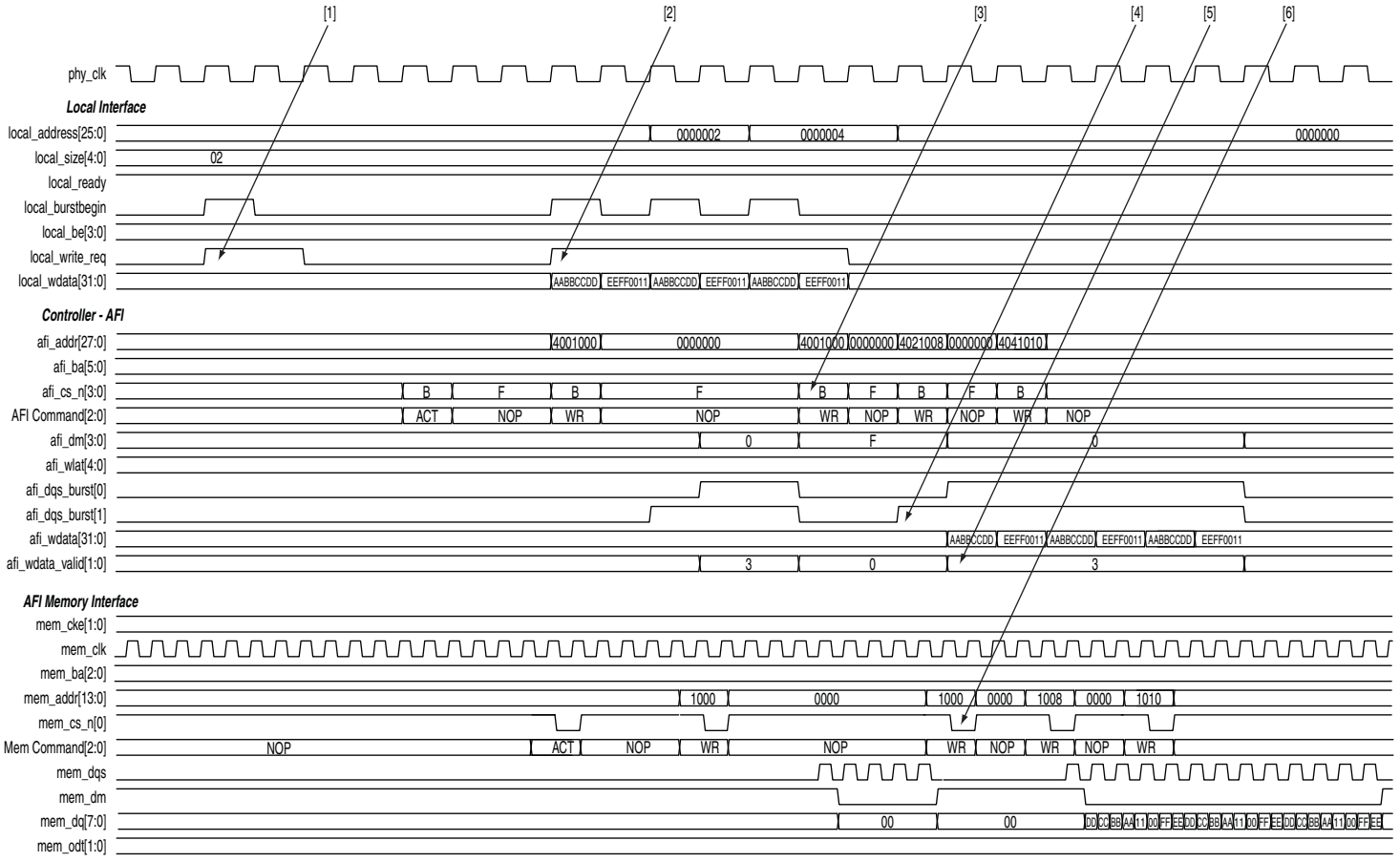
2. The user logic initiates a second read to a different memory column within the same row. The request for the second write is a burst length of 2. In this example, the user logic continues to accept commands until the command queue is full. When the command queue is full, the controller deasserts the `local_ready` signal. The starting local address 0x000002 is mapped to the following memory address in half-rate mode:

   ```
   mem_row_address = 0x0000

   mem_col_address = 0x0002<<2 = 0x0008

   mem_bank_address = 0x00
   ```

3. The controller issues the first read memory command and address signals to the ALTMEMPHY megafunction for it to send to the memory device.
4. The controller asserts the `afi_doing_rd` signal to indicate to the ALTMEMPHY megafunction the number of clock cycles of read data it must expect for the first read. The ALTMEMPHY megafunction uses the `afi_doing_rd` signal to enable its capture registers for the expected duration of memory burst.
5. The ALTMEMPHY megafunction issues the first read command to the memory and captures the read data from the memory.
6. The ALTMEMPHY megafunction returns the first data read to the controller after resynchronizing the data to the `phy_clk` domain, by asserting the `afi_rdata_valid` signal when there is valid read data on the `afi_rdata` bus.
7. The controller returns the first read data to the user by asserting the `local_rdata_valid` signal when there is valid read data on the `local_rdata` bus. If the ECC logic is disabled, there is no delay between the `afi_rdata` and the `local_rdata` buses. If there is ECC logic in the controller, there is one or three clock cycles of delay between the `afi_rdata` and `local_rdata` buses.

# Half-Rate Write (Burst-Aligned Address)

**Figure 9–9. Half-Rate Write Operation for HPC II—Burst-Aligned Address**

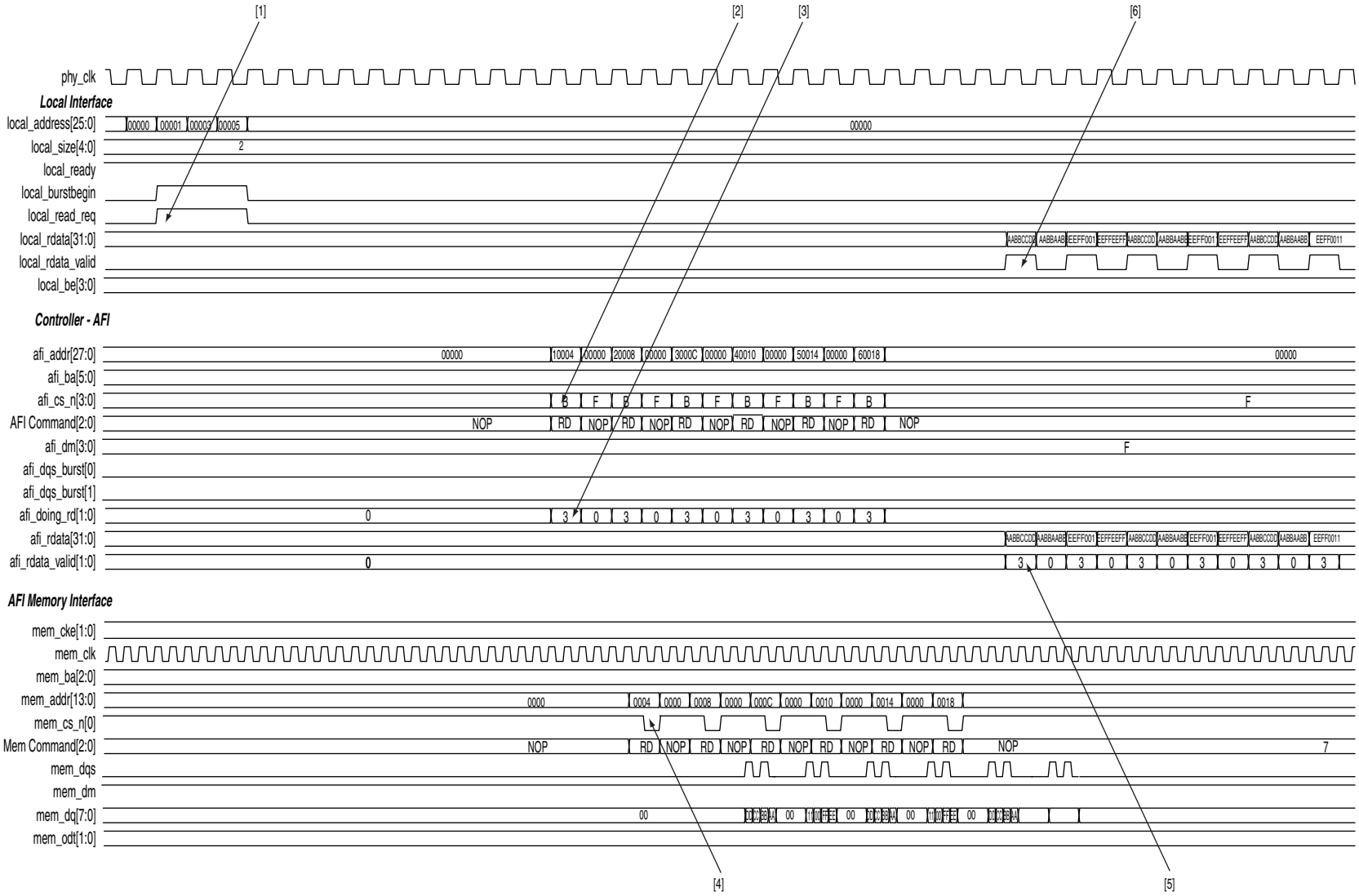


The following sequence corresponds with the numbered items in Figure 9–9:

1. The user logic asserts the first write request to row 0 so that row 0 is open before the next transaction.
2. The user logic asserts a second `local_write_req` signal with size of 2 and address of 0 (col = 0, row = 0, bank = 0, chip = 0). The `local_ready` signal is asserted along with the `local_write_req` signal, which indicates that the controller has accepted this request, and the user logic can request another read or write in the following clock cycle. If the `local_ready` signal was not asserted, the user logic must keep the write request, size, and address signals asserted until the `local_ready` signal is registered high.
3. The controller issues the necessary memory command and address signals to the ALTMEMPHY megafunction for it to send to the memory device.
4. The controller asserts the `afi_wdata_valid` signal to indicate to the ALTMEMPHY megafunction that valid write data and write data masks are present on the inputs to the ALTMEMPHY megafunction.
5. The controller asserts the `afi_dqs_burst` signals to control the timing of the DQS signal that the ALTMEMPHY megafunction issues to the memory.
6. The ALTMEMPHY megafunction issues the write command, and sends the write data and write DQS to the memory.

# Half-Rate Read (Non Burst-Aligned Address)

**Figure 9–10. Half-Rate Read Operation for HPC II—Non Burst-Aligned Address**



The following sequence corresponds with the numbered items in Figure 9–10:

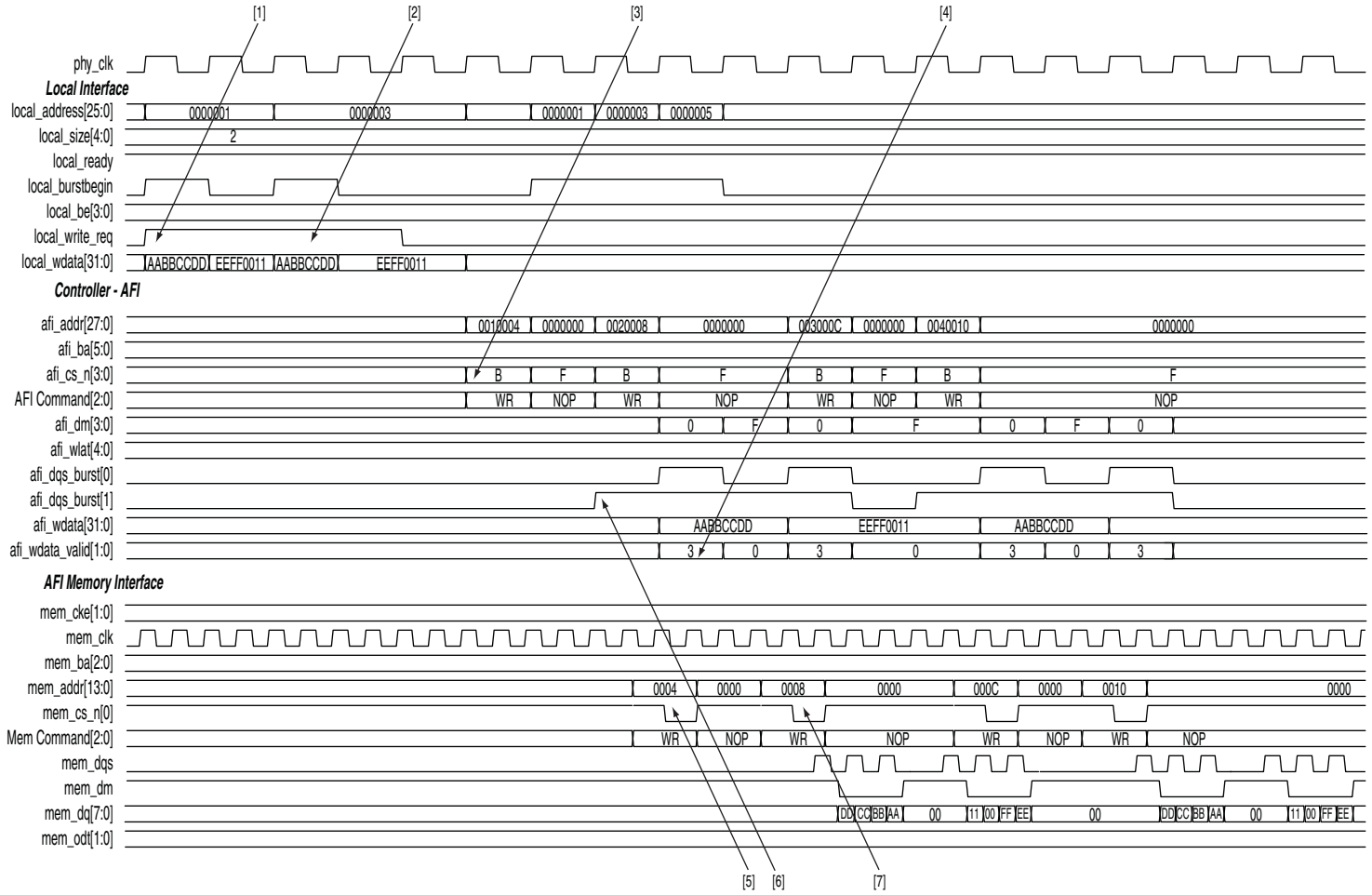
1. The user logic requests the first read by asserting the `local_read_req` signal, and the size and address for this read. In this example, the request is a burst of length of 2 to the local address `0x000001`. This local address is mapped to the following memory address in half-rate mode:

   ```
   mem_row_address = 0x0000

   mem_col_address = 0x0001<<2 = 0x0004

   mem_bank_address = 0x00
   ```

2. The controller issues the first read memory command and address signals to the ALTMEMPHY megafunction for it to send to the memory device.
3. The controller asserts the `afi_doing_rd` signal to indicate to the ALTMEMPHY megafunction the number of clock cycles of read data it must expect for the first read. The ALTMEMPHY megafunction uses the `afi_doing_rd` signal to enable its capture registers for the expected duration of memory burst.
4. The ALTMEMPHY megafunction issues the first read command to the memory and captures the read data from the memory.
5. The ALTMEMPHY megafunction returns the first data read to the controller after resynchronizing the data to the `phy_clk` domain, by asserting the `afi_rdata_valid` signal when there is valid read data on the `afi_rdata` bus.
6. The controller returns the first read data to the user by asserting the `local_rdata_valid` signal when there is valid read data on the `local_rdata` bus. If the ECC logic is disabled, there is no delay between the `afi_rdata` and the `local_rdata` buses. If there is ECC logic in the controller, there is one or three clock cycles of delay between the `afi_rdata` and `local_rdata` buses.

# Half-Rate Write (Non Burst-Aligned Address)

**Figure 9–11. Half-Rate Write Operation for HPC II—Non Burst-Aligned Address**



The following sequence corresponds with the numbered items in Figure 9–11:

1. The user logic asserts the first `local_write_req` signal with a size of 2 and an address of `0x000001`. The `local_ready` signal is asserted along with the `local_write_req` signal, which indicates that the controller has accepted this request, and the user logic can request another read or write in the following clock cycle. If the `local_ready` signal was not asserted, the user logic must keep the write request, size, and address signals asserted until the `local_ready` signal is registered high. The local address 0x000001 is mapped to the following memory address in half-rate mode:

   ```
   mem_row_address = 0x0000

   mem_col_address = 0x000001<<2 = 0x000004

   mem_bank_address = 0x00
   ```

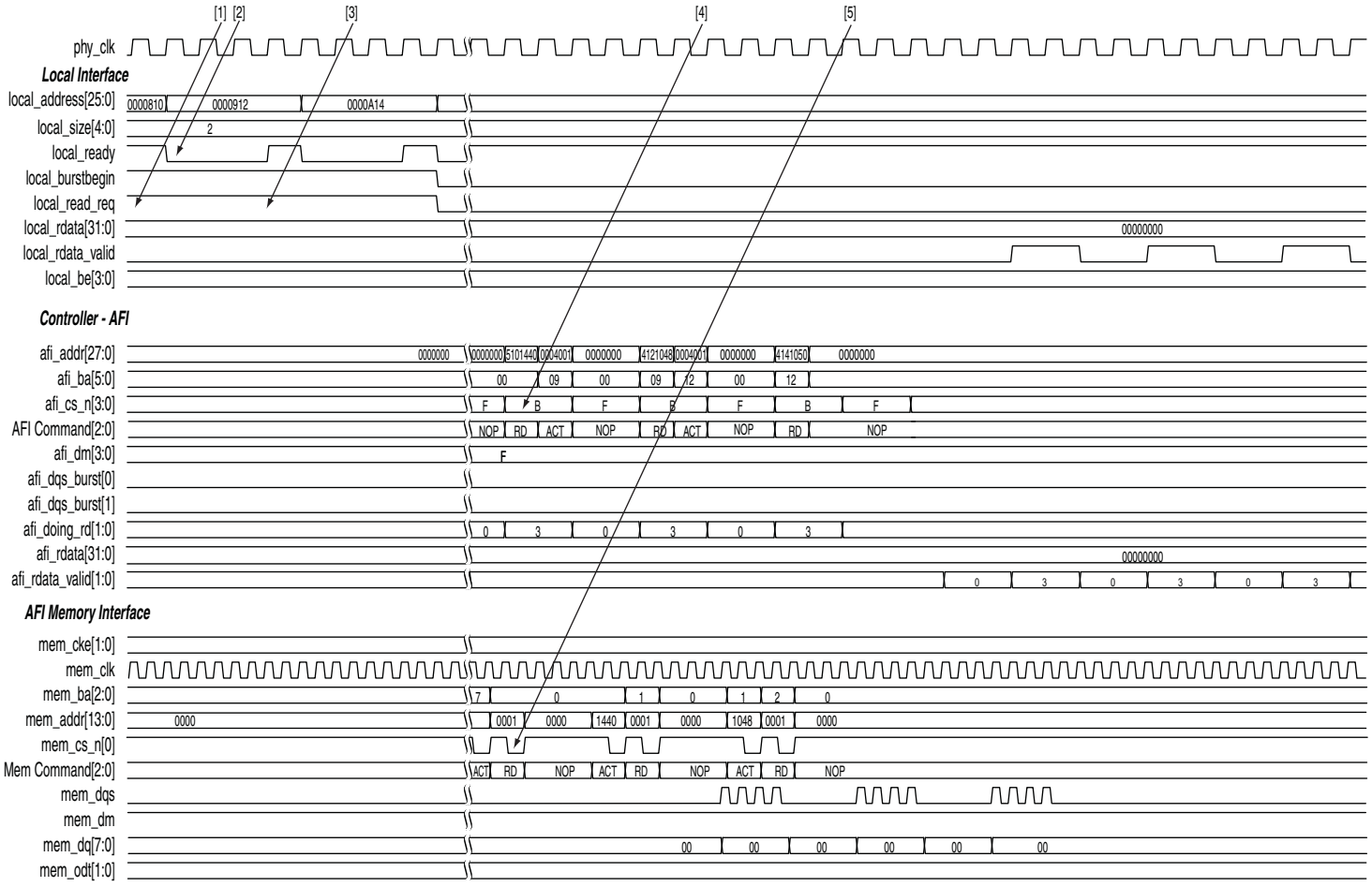
2. The user logic asserts the second `local_write_req` signal with a size of 2 and an address of `0x000003`. The local address `0x000003` is mapped to the following memory address in half-rate mode:

   ```
   mem_row_address = 0x0000

   mem_col_address = 0x000003<<2 = 0x00000C

   mem_bank_address = 0x00
   ```

3. The controller issues the necessary memory command and address signals to the ALTMEMPHY megafunction for it to send to the memory device.
4. The controller asserts the `afi_wdata_valid` signal to indicate to the ALTMEMPHY megafunction that valid write data and write data masks are present on the inputs to the ALTMEMPHY megafunction.
5. The controller asserts the `afi_dqs_burst` signals to control the timing of the DQS signal that the ALTMEMPHY megafunction issues to the memory.
6. The ALTMEMPHY megafunction issues the write command, and sends the write data and write DQS to the memory.
7. The controller generates another write because the first write is to a non-aligned memory address, `0x0004`. The controller performs the second write burst at the memory address of `0x0008`.

# Half-Rate Read With Gaps

**Figure 9–12. Half-Rate Read Operation for HPC II—With Gaps**



The following sequence corresponds with the numbered items in Figure 9–12:

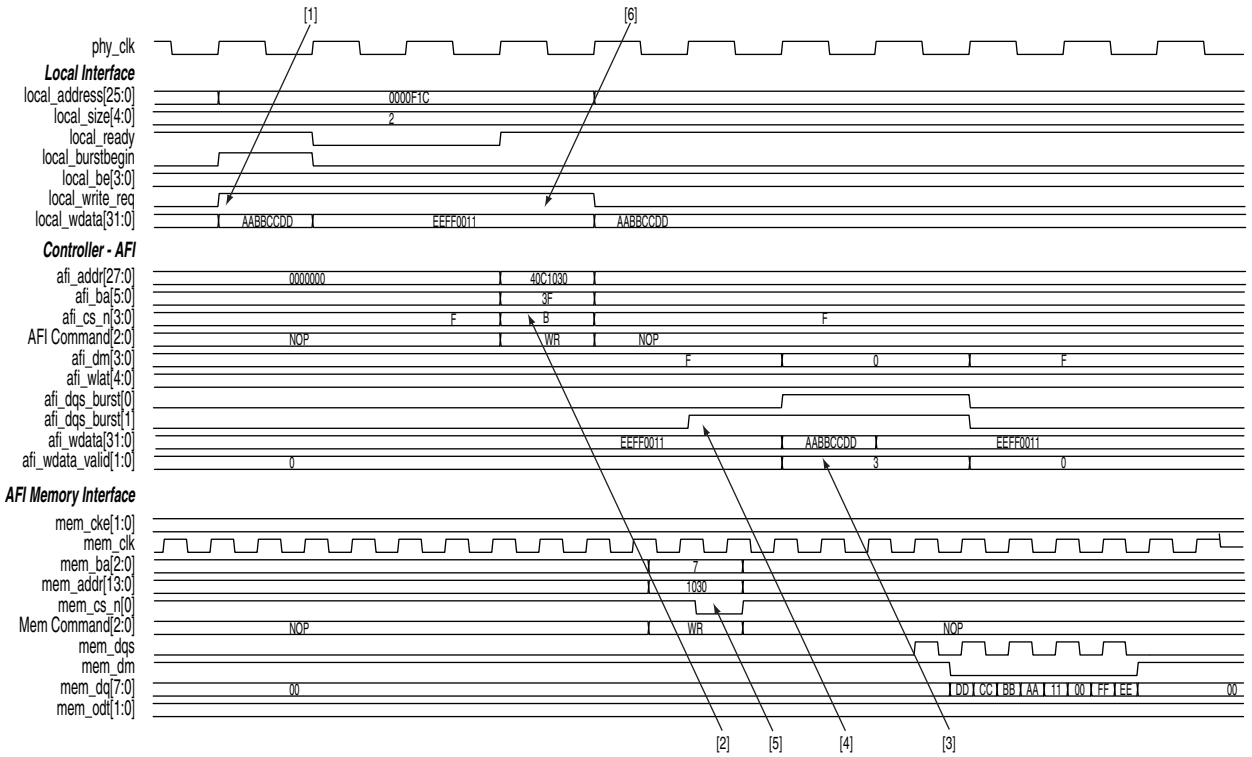
1. The user logic requests the first read by asserting the `local_read_req` signal, and the size and address for this read. In this example, the request is a burst of length of 2 to the local address `0x0000810`. This local address is mapped to the following memory address in half-rate mode:

   ```
   mem_row_address = 0x0001

   mem_col_address = 0x0010<<2 = 0x0040

   mem_bank_address = 0x00
   ```

2. When the command queue is full, the controller deasserts the `local_ready` signal to indicate that the controller has not accepted the command. The user logic must keep the read request, size, and address signal until the `local_ready` signal is asserted again.
3. The user logic asserts a second `local_read_req` signal with a size of 2 and address of `0x0000912`.
4. The controller issues the first read memory command and address signals to the ALTMEMPHY megafunction for it to send to the memory device.
5. The ALTMEMPHY megafunction issues the read command to the memory and captures the read data from the memory.

## Half-Rate Write With Gaps

**Figure 9–13. Half-Rate Write Operation for HPC II—With Gaps**

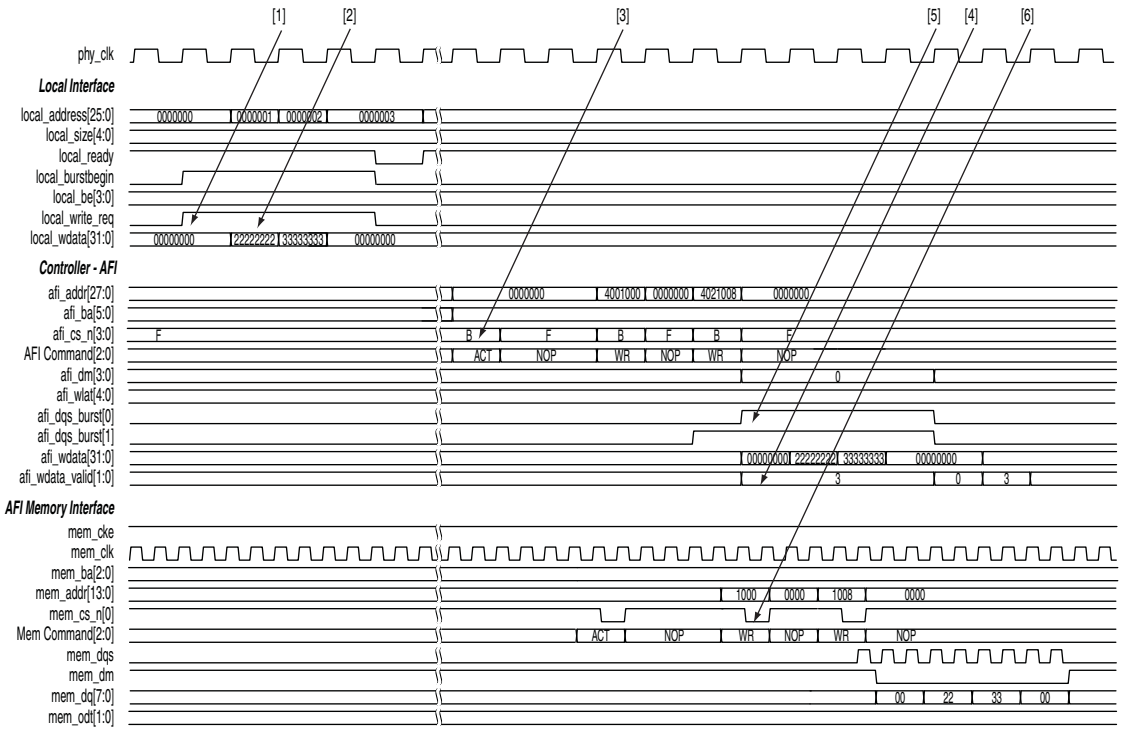


The following sequence corresponds with the numbered items in Figure 9–13:

1. The user logic asserts a `local_write_req` signal with a size of 2 and an address of `0x0000F1C`.
2. The controller issues the necessary memory command and address signals to the ALTMEMPHY megafunction for it to send to the memory device.
3. The controller asserts the `afi_wdata_valid` signal to indicate to the ALTMEMPHY megafunction that valid write data and write data masks are present on the inputs to the ALTMEMPHY megafunction.
4. The controller asserts the `afi_dqs_burst` signals to control the timing of the DQS signal that the ALTMEMPHY megafunction issues to the memory.
5. The ALTMEMPHY megafunction issues the write command, and sends the write data and write DQS to the memory.
6. For transactions with a local size of two, the `local_write_req` and `local_ready` signals must be high for two clock cycles so that all the write data can be transferred to the controller.

## Half-Rate Write Operation (Merging Writes)

**Figure 9–14. Write Operation for HPC II—Merging Writes**



The following sequence corresponds with the numbered items in Figure 9–14:

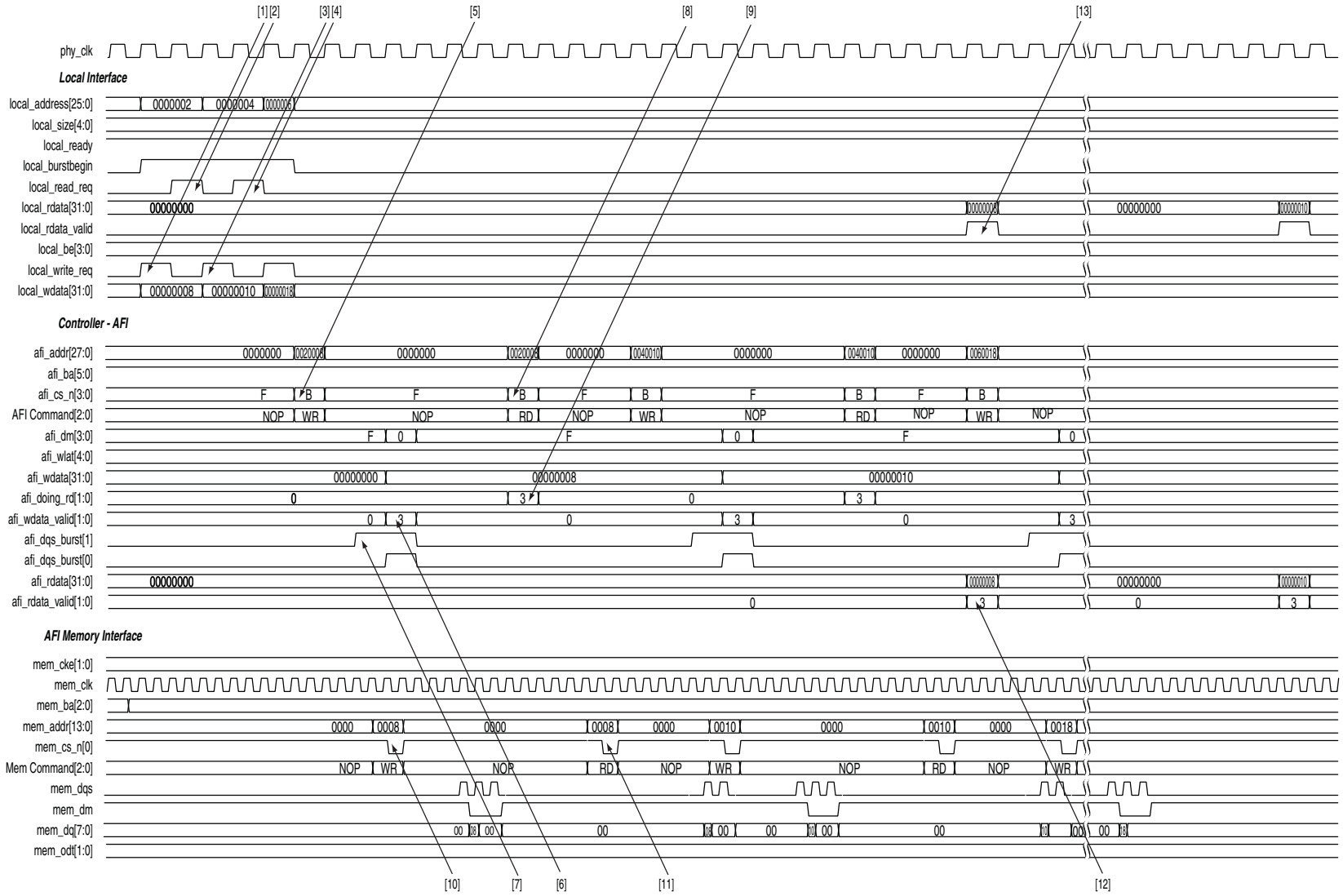
1. The user logic asserts the first `local_write_req` signal with a size of 1 and an address of `0x000000`. The `local_ready` signal is asserted along with the `local_write_req` signal, which indicates that the controller has accepted this request, and the user logic can request another read or write in the following clock cycle. If the `local_ready` signal was not asserted, the user logic must keep the write request, size, and address signals asserted until the `local_ready` signal is registered high. The local address `0x000000` is mapped to the following memory address in half-rate mode:

   ```
   mem_row_address = 0x0000

   mem_col_address = 0x0000<<2 = 0x0000

   mem_bank_address = 0x00
   ```

2. The user logic asserts a second `local_write_req` signal with a size of 1 and address of 1. The `local_ready` signal is asserted along with the `local_write_req` signal, which indicates that the controller has accepted this request. Since the second write request is to a sequential address (same row, same bank, and column increment by 1), this write and the first write can be merged at the memory transaction.
3. The controller issues the necessary memory command and address signals to the ALTMEMPHY megafunction for it to send to the memory device.
4. The controller asserts the `afi_wdata_valid` signal to indicate to the ALTMEMPHY megafunction that valid write data and write data masks are present on the inputs to the ALTMEMPHY megafunction.
5. The controller asserts the `afi_dqs_burst` signals to control the timing of the DQS signal that the ALTMEMPHY megafunction issues to the memory.
6. The ALTMEMPHY megafunction issues the write command, and sends the write data and write DQS to the memory.

# Write-Read-Write-Read Operation

**Figure 9–15. Write-Read Sequential Operation for HPC II**



The following sequence corresponds with the numbered items in Figure 9–15:

1. The user logic requests the first write by asserting the `local_write_req` signal, and the size and address for this write. In this example, the request is a burst length of 1 to a local address `0x000002`. This local address is mapped to the following memory address in half-rate mode:

   ```
   mem_row_address = 0x0000

   mem_col_address = 0x0002<<2 = 0x0008

   mem_bank_address = 0x00
   ```

2. The user logic initiates the first read to the same address as the first write. The request for the read is a burst length of 1. The controller continues to accept commands until the command queue is full. When the command queue is full, the controller deasserts the `local_ready` signal. The starting local address 0x000002 is mapped to the following memory address in half-rate mode:

   ```
   mem_row_address = 0x0000

   mem_col_address = 0x0002<<2 = 0x0008

   mem_bank_address = 0x00
   ```

3. The user logic asserts a second `local_write_req` signal with a size of 1 and address of `0x000004`.
4. The user logic asserts a second `local_read_req` signal with a size of 1 and address of `0x000004`.
5. The controller issues the necessary memory command and address signals to the ALTMEMPHY megafunction for it to send to the memory device.
6. The controller asserts the `afi_wdata_valid` signal to indicate to the ALTMEMPHY megafunction that valid write data and write data masks are present on the inputs to the ALTMEMPHY megafunction.
7. The controller asserts the `afi_dqs_burst` signals to control the timing of the DQS signals that the ALTMEMPHY megafunction issues to the memory.
8. The controller issues the first read memory command and address signals to the ALTMEMPHY megafunction for it to send to the memory device.
9. The controller asserts the `afi_doing_rd` signal to indicate to the ALTMEMPHY megafunction the number of clock cycles of read data it must expect for the first read. The ALTMEMPHY megafunction uses the `afi_doing_rd` signal to enable its capture registers for the expected duration of memory burst.
10. The ALTMEMPHY megafunction issues the write command, and sends the write data and write DQS to the memory.
11. The ALTMEMPHY megafunction issues the first read command to the memory and captures the read data from the memory.
12. The ALTMEMPHY megafunction returns the first data read to the controller after resynchronizing the data to the `phy_clk` domain, by asserting the `afi_rdata_valid` signal when there is valid read data on the `afi_rdata` bus.

13. The controller returns the first read data to the user by asserting the `local_rdata_valid` signal when there is valid read data on the `local_rdata` bus. If the ECC logic is disabled, there is no delay between the `afi_rdata` and the `local_rdata` buses. If there is ECC logic in the controller, there is one or three clock cycles of delay between the `afi_rdata` and `local_rdata` buses.

This chapter provides additional information about the document and Altera.

## Document Revision History

The following table shows the revision history for this document.

| Date | Version | Changes |
|---|---|---|
| December 2010 | 2.1 | Updated for 10.1. |
| July 2010 | 2.0 | ■ Added information for new GUI parameters: **Controller latency**, **Enable reduced bank tracking for area optimization**, and **Number of banks to track**.<br>■ Removed information about IP Advisor. This feature is removed from the DDR3 SDRAM IP support for version 10.0. |
| February 2010 | 1.3 | Corrected typos. |
| February 2010 | 1.2 | ■ Full support for Stratix IV devices.<br>■ Added information for **Register Control Word** parameters.<br>■ Added descriptions for `mem_ac_parity`, `mem_err_out_n`, and `parity_error_n` signals.<br>■ Added timing diagrams for initialization and calibration stages for HPC. |
| November 2009 | 1.1 | Minor corrections. |
| November 2009 | 1.0 | First published. |

## How to Contact Altera

To locate the most up-to-date information about Altera products, refer to the following table.

| Contact *(1)* | Contact Method | Address |
|---|---|---|
| Technical support | Website | www.altera.com/support |
| Technical training | Website | www.altera.com/training |
| | Email | custrain@altera.com |
| Product literature | Website | www.altera.com/literature |
| Non-technical support (General) | Email | nacomp@altera.com |
| (Software Licensing) | Email | authorization@altera.com |

**Note to Table:**

(1) You can also contact your local Altera sales office or sales representative.

# Typographic Conventions

The following table shows the typographic conventions this document uses.

| Visual Cue | Meaning |
|---|---|
| **Bold Type with Initial Capital Letters** | Indicate command names, dialog box titles, dialog box options, and other GUI labels. For example, **Save As** dialog box. For GUI elements, capitalization matches the GUI. |
| **bold type** | Indicates directory names, project names, disk drive names, file names, file name extensions, software utility names, and GUI labels. For example, **\qdesigns** directory, **D:** drive, and **chiptrip.gdf** file. |
| *Italic Type with Initial Capital Letters* | Indicate document titles. For example, *Stratix IV Design Guidelines*. |
| *italic type* | Indicates variables. For example, *n* + 1.<br>Variable names are enclosed in angle brackets (< >). For example, *<file name>* and *<project name>*.**pof** file. |
| Initial Capital Letters | Indicate keyboard keys and menu names. For example, the Delete key and the Options menu. |
| "Subheading Title" | Quotation marks indicate references to sections within a document and titles of Quartus II Help topics. For example, "Typographic Conventions." |
| `Courier type` | Indicates signal, port, register, bit, block, and primitive names. For example, `data1`, `tdi`, and `input`. The suffix `n` denotes an active-low signal. For example, `resetn`.<br>Indicates command line commands and anything that must be typed exactly as it appears. For example, `c:\qdesigns\tutorial\chiptrip.gdf`.<br>Also indicates sections of an actual file, such as a Report File, references to parts of files (for example, the AHDL keyword `SUBDESIGN`), and logic function names (for example, `TRI`). |
| ↵ | An angled arrow instructs you to press the Enter key. |
| 1., 2., 3., and<br>a., b., c., and so on | Numbered steps indicate a list of items when the sequence of the items is important, such as the steps listed in a procedure. |
| ■ ■ ■ | Bullets indicate a list of items when the sequence of the items is not important. |
|  | The hand points to information that requires special attention. |
|  | A question mark directs you to a software help system with related information. |
|  | The feet direct you to another document or website with related information. |
|  | A caution calls attention to a condition or possible situation that can damage or destroy the product or your work. |
|  | A warning calls attention to a condition or possible situation that can cause you injury. |
|  | The envelope links to the Email Subscription Management Center page of the Altera website, where you can sign up to receive update notifications for Altera documents. |

# External Memory Interface Handbook Volume 3

# Section V. DDR2 and DDR3 SDRAM Controller with UniPHY User Guide

101 Innovation Drive
San Jose, CA 95134
www.altera.com

EMI_DDR3UP_UG-1.1

Document last updated for Altera Complete Design Suite version: 10.1
Document publication date: December 2010

Subscribe

# Contents

# 1. About This IP

The Altera® DDR2 and DDR3 SDRAM controllers with UniPHY provide low latency, high-performance, feature-rich controller interfaces to industry-standard DDR2 and DDR3 SDRAM memory. The DDR2 SDRAM controller with UniPHY offers full-rate and half-rate DDR2 interfaces, and the DDR3 SDRAM controller with UniPHY offers a half-rate DDR3 SDRAM interface.

The MegaWizard™ interface generates an example top-level project, consisting of an example driver, and your DDR2 or DDR3 SDRAM controller custom variation. The controller instantiates an instance of the UniPHY datapath.

The example top-level project is a fully-functional design that you can simulate, synthesize, and use in hardware. The example driver is a self-test module that issues read and write commands to the controller and checks the read data to produce the pass, fail, and test-complete signals.

The UniPHY datapath is an interface between a memory controller and memory devices and performs read and write operations to the memory.

For device families not supported by the UniPHY-based designs, use the Altera ALTMEMPHY-based High Performance SDRAM Controller IP core.

If the UniPHY datapath does not match your requirements, you can create your own memory interface datapath using the ALTDLL, ALTDQ_DQS, ALTDQ_DQS2, ALTDQ, or ALTDQS megafunctions, available in the Quartus® II software, but you are then responsible for all aspects of the design including timing analysis and design constraints.

## Release Information

Table 1–1 provides information about this release of the DDR2 and DDR3 SDRAM controllers with UniPHY.

**Table 1–1. Release Information**

| Item | Description |
|---|---|
| Version | 10.1 |
| Release Date | December 2010 |
| Ordering Codes | IP-DDR2/UNI<br>IP-DDR3/UNI |
| Vendor ID | 6AF7 |

Altera verifies that the current version of the Quartus II software compiles the previous version of each MegaCore function. The *MegaCore IP Library Release Notes and Errata* report any exceptions to this verification. Altera does not verify compilation with MegaCore function versions older than one release.

# Device Family Support

IP cores provide the following levels of support for target Altera device families:

- For FPGA device support:
  - Preliminary—verified with preliminary timing models for this device
  - Final—verified with final timing models for this device
- For ASIC devices (HardCopy families)
  - HardCopy companion—verified with preliminary timing models for HardCopy companion device
  - HardCopy compilation—verifed with final timing models for HardCopy device

Table 1–2 shows the level of support offered by the DDR2 and DDR3 SDRAM controllers to each of the Altera device families.

For information about features and supported clock rates for external memory interfaces, refer to the *External Memory Interface System Specifications* section in volume 1 of the *External Memory Interface Handbook* or use the *External Memory Specifiation Estimator*.

**Table 1–2. Device Family Support**

| Device Family | Support |
|---|---|
| Arria II GZ | Preliminary |
| HardCopy® III | Companion |
| HardCopy IV | Companion |
| Stratix® III (For DDR3, only $V_{CC}$ = 1.1V supported) | Final |
| Stratix IV | Final |
| Stratix V | Preliminary |
| Other device families | No support |

# Features

Table 1–3 summarizes key feature support for the DDR2 and DDR3 SDRAM controllers with UniPHY.

**Table 1–3. Key Feature Support for DDR2 and DDR3 SDRAM Controllers with UniPHY**

| Key Feature | DDR2 SDRAM UniPHY | DDR3 SDRAM UniPHY |
|---|---|---|
| High-performance controller (HPC) | — | — |
| High-performance controller II (HPC II) | ✓ | ✓ |
| Half-rate core logic and user interface | ✓ | ✓ |
| Full-rate core logic and user interface | ✓ | — |
| UDIMM and RDIMM in any form factor (1) | ✓ | ✓ |
| Multiple components in a single-rank UDIMM or RDIMM layout | ✓ | ✓ |
| Burst length in HPC II (half-rate) | 8 | — |
| Burst rate in HPC II (full-rate) and HPC full- and half-rate) | 4 | — |
| Burst length of 8 and burst chop of 4 (on the fly) | — | ✓ |
| Reduced controller latency only in HPC II (2) (3) | ✓ | ✓ |
| With leveling | ✓(240 MHz and above) | ✓(240 MHz and above) (4) |
| Without leveling | ✓(below 240 MHz) | — |
| Maximum data width (5) | 144 bits | 144 bits |

**Notes for Table 1–3:**

(1) DIMM form is not supported in Arria II GX and Arria II GZ devices

(2) When you select reduced controller latency, you must use Quatrus II software timing analysis of your complete design to achieve maximum clock rate.

(3) Not available in Arria II GX devices.

(4) The leveling delay on the board between first and last DDR3 SDRAM component laid out as a DIMM must be less than 0.69 $t_{CK}$.

(5) For any interface with data width above 72 bits, you must use Quartus II software timing analysis of your complete design to achieve the maximum clock rate.

# Unsupported Features

Table 1–4 summarizes unsupported features for the DDR2 and DDR3 SDRAM controllers with UniPHY.

**Table 1–4. Unsupported Features for DDR2 and DDR3 SDRAM Controllers with UniPHY (Part 1 of 2)**

| Memory Protocol | Unsupported Feature |
|---|---|
| DDR2 SDRAM | High-performance controller (HPC) |
| | Timing simulation |
| | Arria II GX support |
| | Cyclone III support |
| | Cyclone IV support |

**Table 1–4. Unsupported Features for DDR2 and DDR3 SDRAM Controllers with UniPHY (Part 2 of 2)**

| Memory Protocol | Unsupported Feature |
|---|---|
| DDR3 SDRAM | High-performance controller (HPC) |
| | Timing simulation |
| | Full-rate |
| | Arria II GZ DIMM in any form factor |
| | Stratix III ($V_{cc}$ = 0.9V) |
| | Arria II GX support |
| | Cyclone III support |
| | Cyclone IV support |

# MegaCore Verification

Altera has carried out extensive random, directed tests with functional test coverage using industry-standard models to ensure the functionality of the DDR2 and DDR3 SDRAM controllers with UniPHY. Altera's functional verification of DDR2 and DDR3 controllers with UniPHY use modified Denali models, with certain assertions disabled.

# Resource Utilization

This section lists resource utilization for the DDR2 and DDR3 SDRAM controllers with UniPHY for supported device families.

Table 1–5 shows the typical size of the DDR2 and DDR3 SDRAM controllers with UniPHY in the Quartus II software version 10.1 for Arria II GZ devices.

**Table 1–5. Resource Utilization in Arria II GZ Devices (Part 1 of 2)**

| Protocol | Memory Width (Bits) | Combinational ALUTS | Logic Registers | Mem ALUTs | M9K Blocks | M144K Blocks | Memory (Bits) |
|---|---|---|---|---|---|---|---|
| **Controller** | | | | | | | |
| DDR2 (Half rate) | 8 | 955 | 687 | 32 | 1 | 0 | 8,192 |
| | 16 | 959 | 707 | 64 | 2 | 0 | 16,384 |
| | 64 | 949 | 687 | 32 | 8 | 0 | 72,704 |
| | 72 | 941 | 653 | 0 | 9 | 0 | 82,944 |
| DDR2 (Full rate) | 8 | 963 | 663 | 16 | 1 | 0 | 4,096 |
| | 16 | 964 | 673 | 32 | 1 | 0 | 8,192 |
| | 64 | 957 | 639 | 0 | 5 | 0 | 36,864 |
| | 72 | 953 | 639 | 0 | 5 | 0 | 41,472 |
| DDR3 (Half rate) | 8 | 1,215 | 801 | 32 | 1 | 0 | 8,192 |
| | 16 | 1,219 | 821 | 64 | 2 | 0 | 16,384 |
| | 64 | 1,215 | 801 | 32 | 8 | 0 | 72,704 |
| | 72 | 1,201 | 767 | 0 | 9 | 0 | 82,944 |

**Table 1–5. Resource Utilization in Arria II GZ Devices (Part 2 of 2)**

| Protocol | Memory Width (Bits) | Combinational ALUTS | Logic Registers | Mem ALUTs | M9K Blocks | M144K Blocks | Memory (Bits) |
|---|---|---|---|---|---|---|---|
| **PHY** | | | | | | | |
| DDR2 (Half rate) | 8 | 2,265 | 1,872 | 0 | 24 | 0 | 173,056 |
| | 16 | 2,438 | 2,139 | 0 | 24 | 0 | 173,056 |
| | 64 | 3,360 | 3,741 | 0 | 24 | 0 | 173,056 |
| | 72 | 3,517 | 4,008 | 0 | 24 | 0 | 173,056 |
| DDR2 (Full rate) | 8 | 2,250 | 1,840 | 0 | 24 | 0 | 173,056 |
| | 16 | 2,411 | 2,090 | 0 | 24 | 0 | 173,056 |
| | 64 | 3,311 | 3,590 | 0 | 24 | 0 | 173,056 |
| | 72 | 3,450 | 3,840 | 0 | 24 | 0 | 173,056 |
| DDR3 (Half rate) | 8 | 2,278 | 1,885 | 0 | 24 | 0 | 173,056 |
| | 16 | 2,449 | 2,152 | 0 | 24 | 0 | 173,056 |
| | 64 | 3,367 | 3,754 | 0 | 24 | 0 | 173,056 |
| | 72 | 3,527 | 4,021 | 0 | 24 | 0 | 173,056 |
| **Total** | | | | | | | |
| DDR2 (Half rate) | 8 | 3,220 | 2,559 | 32 | 25 | 0 | 181,248 |
| | 16 | 3,397 | 2,846 | 64 | 26 | 0 | 189,440 |
| | 64 | 4,309 | 4,428 | 32 | 32 | 0 | 245,760 |
| | 72 | 4,458 | 4,661 | 0 | 33 | 0 | 256,000 |
| DDR2 (Full rate) | 8 | 3,213 | 2,503 | 16 | 25 | 0 | 177,152 |
| | 16 | 3,375 | 2,763 | 32 | 25 | 0 | 181,248 |
| | 64 | 4,268 | 4,229 | 0 | 29 | 0 | 209,920 |
| | 72 | 4,403 | 4,479 | 0 | 29 | 0 | 214,528 |
| DDR3 (Half rate) | 8 | 3,493 | 2,686 | 32 | 25 | 0 | 181,248 |
| | 16 | 3,668 | 2,973 | 64 | 26 | 0 | 189,440 |
| | 64 | 4,582 | 4,555 | 32 | 32 | 0 | 245,760 |
| | 72 | 4,728 | 4,788 | 0 | 33 | 0 | 256,000 |

Table 1–6 shows the typical size of the DDR2 and DDR3 SDRAM controllers with UniPHY in the Quartus II software version 10.1 for Stratix III devices.

**Table 1–6. Resource Utilization in Stratix III Devices (Part 1 of 2)**

| Protocol | Memory Width (Bits) | Combinational ALUTS | Logic Registers | Mem ALUTs | M9K Blocks | M144K Blocks | Memory (Bits) |
|---|---|---|---|---|---|---|---|
| **Controller** | | | | | | | |
| DDR2 (Half rate) | 8 | 942 | 656 | 0 | 2 | 0 | 9,216 |
| | 16 | 940 | 656 | 0 | 3 | 0 | 18,432 |
| | 64 | 941 | 656 | 0 | 9 | 0 | 73,728 |
| | 72 | 934 | 656 | 0 | 9 | 0 | 82,944 |

**Table 1–6. Resource Utilization in Stratix III Devices (Part 2 of 2)**

| Protocol | Memory Width (Bits) | Combinational ALUTS | Logic Registers | Mem ALUTs | M9K Blocks | M144K Blocks | Memory (Bits) |
|---|---|---|---|---|---|---|---|
| DDR2 (Full rate) | 8 | 961 | 641 | 0 | 2 | 0 | 4,608 |
| | 16 | 963 | 641 | 0 | 2 | 0 | 9,216 |
| | 64 | 961 | 641 | 0 | 5 | 0 | 36,864 |
| | 72 | 961 | 641 | 0 | 5 | 0 | 41,472 |
| DDR3 (Half rate) | 8 | 1,212 | 768 | 0 | 2 | 0 | 9,216 |
| | 16 | 1,211 | 768 | 0 | 3 | 0 | 18,432 |
| | 64 | 1,208 | 768 | 0 | 9 | 0 | 73,728 |
| | 72 | 1,211 | 768 | 0 | 9 | 0 | 82,944 |
| **PHY** | | | | | | | |
| DDR2 (Half rate) | 8 | 2,270 | 1,870 | 0 | 8 | 1 | 173,056 |
| | 16 | 2,425 | 2,137 | 0 | 8 | 1 | 173,056 |
| | 64 | 3,351 | 3,739 | 0 | 8 | 1 | 173,056 |
| | 72 | 3,515 | 4,006 | 0 | 8 | 1 | 173,056 |
| DDR2 (Full rate) | 8 | 2,245 | 1,840 | 0 | 8 | 1 | 173,056 |
| | 16 | 2,403 | 2,090 | 0 | 8 | 1 | 173,056 |
| | 64 | 3,308 | 3,590 | 0 | 8 | 1 | 173,056 |
| | 72 | 3,448 | 3,841 | 0 | 8 | 1 | 173,056 |
| DDR3 (Half rate) | 8 | 2,281 | 1,885 | 0 | 8 | 1 | 173,056 |
| | 16 | 2,449 | 2,152 | 0 | 8 | 1 | 173,056 |
| | 64 | 3,367 | 3,754 | 0 | 8 | 1 | 173,056 |
| | 72 | 3,521 | 4,021 | 0 | 8 | 1 | 173,056 |
| **Total** | | | | | | | |
| DDR2 (Half rate) | 8 | 3,212 | 2,526 | 0 | 10 | 1 | 182,272 |
| | 16 | 3,365 | 2,793 | 0 | 11 | 1 | 191,488 |
| | 64 | 4,292 | 4,395 | 0 | 17 | 1 | 246,784 |
| | 72 | 4,449 | 4,662 | 0 | 17 | 1 | 256,000 |
| DDR2 (Full rate) | 8 | 3,206 | 2,481 | 0 | 10 | 1 | 177,664 |
| | 16 | 3,366 | 2,731 | 0 | 10 | 1 | 182,272 |
| | 64 | 4,269 | 4,231 | 0 | 13 | 1 | 209,920 |
| | 72 | 4,409 | 4,482 | 0 | 13 | 1 | 214,528 |
| DDR3 (Half rate) | 8 | 3,493 | 2,653 | 0 | 10 | 1 | 182,272 |
| | 16 | 3,660 | 2,920 | 0 | 11 | 1 | 191,488 |
| | 64 | 4,575 | 4,522 | 0 | 17 | 1 | 246,784 |
| | 72 | 4,732 | 4,789 | 0 | 17 | 1 | 256,000 |

Table 1–7 shows the typical size of the DDR2 and DDR3 SDRAM controllers with UniPHY in the Quartus II software version 10.1 for Stratix IV devices.

**Table 1–7. Resource Utilization in Stratix IV Devices (Part 1 of 2)**

| Protocol | Memory Width (Bits) | Combinational ALUTS | Logic Registers | Mem ALUTs | M9K Blocks | M144K Blocks | Memory (Bits) |
|---|---|---|---|---|---|---|---|
| **Controller** | | | | | | | |
| DDR2 (Half rate) | 8 | 951 | 689 | 32 | 1 | 0 | 8,192 |
| | 16 | 952 | 709 | 64 | 2 | 0 | 16,384 |
| | 64 | 947 | 689 | 32 | 8 | 0 | 72,704 |
| | 72 | 934 | 655 | 0 | 9 | 0 | 82,944 |
| DDR2 (Full rate) | 8 | 965 | 664 | 16 | 1 | 0 | 4,096 |
| | 16 | 969 | 674 | 32 | 1 | 0 | 8,192 |
| | 64 | 955 | 640 | 0 | 5 | 0 | 36,864 |
| | 72 | 956 | 640 | 0 | 5 | 0 | 41,472 |
| DDR3 (Half rate) | 8 | 1,206 | 802 | 32 | 1 | 0 | 8,192 |
| | 16 | 1,212 | 822 | 64 | 2 | 0 | 16,384 |
| | 64 | 1,200 | 802 | 32 | 8 | 0 | 72,704 |
| | 72 | 1,195 | 768 | 0 | 9 | 0 | 82,944 |
| **PHY** | | | | | | | |
| DDR2 (Half rate) | 8 | 2,269 | 1,870 | 0 | 8 | 1 | 173,056 |
| | 16 | 2,427 | 2,137 | 0 | 8 | 1 | 173,056 |
| | 64 | 3,352 | 3,739 | 0 | 8 | 1 | 173,056 |
| | 72 | 3,513 | 4,006 | 0 | 8 | 1 | 173,056 |
| DDR2 (Full rate) | 8 | 2,243 | 1,840 | 0 | 8 | 1 | 173,056 |
| | 16 | 2,407 | 2,090 | 0 | 8 | 1 | 173,056 |
| | 64 | 3,307 | 3,590 | 0 | 8 | 1 | 173,056 |
| | 72 | 3,448 | 3,840 | 0 | 8 | 1 | 173,056 |
| DDR3 (Half rate) | 8 | 2,276 | 1,885 | 0 | 8 | 1 | 173,056 |
| | 16 | 2,448 | 2,152 | 0 | 8 | 1 | 173,056 |
| | 64 | 3,382 | 3,754 | 0 | 8 | 1 | 173,056 |
| | 72 | 3,527 | 4,021 | 0 | 8 | 1 | 173,056 |
| **Total** | | | | | | | |
| DDR2 (Half rate) | 8 | 3,220 | 2,559 | 32 | 9 | 1 | 181,248 |
| | 16 | 3,379 | 2,846 | 64 | 10 | 1 | 189,440 |
| | 64 | 4,299 | 4,428 | 32 | 16 | 1 | 245,760 |
| | 72 | 4,447 | 4,661 | 0 | 17 | 1 | 256,000 |
| DDR2 (Full rate) | 8 | 3,208 | 2,504 | 16 | 9 | 1 | 177,152 |
| | 16 | 3,376 | 2,764 | 32 | 9 | 1 | 181,248 |
| | 64 | 4,262 | 4,230 | 0 | 13 | 1 | 209,920 |
| | 72 | 4,404 | 4,480 | 0 | 13 | 1 | 214,528 |

**Table 1–7. Resource Utilization in Stratix IV Devices (Part 2 of 2)**

| Protocol | Memory Width (Bits) | Combinational ALUTS | Logic Registers | Mem ALUTs | M9K Blocks | M144K Blocks | Memory (Bits) |
|---|---|---|---|---|---|---|---|
| DDR3 (Half rate) | 8 | 3,482 | 2,687 | 32 | 9 | 1 | 181,248 |
| | 16 | 3,660 | 2,974 | 64 | 10 | 1 | 189,440 |
| | 64 | 4,582 | 4,556 | 32 | 16 | 1 | 245,760 |
| | 72 | 4,722 | 4,789 | 0 | 17 | 1 | 256,000 |

Table 1–8 shows the typical size of the DDR2 and DDR3 SDRAM controllers with UniPHY in the Quartus II software version 10.1 for Stratix V devices.

**Table 1–8. Resource Utilization in Stratix V Devices (Part 1 of 2)**

| Protocol | Memory Width (Bits) | Combinational LCs | Logic Registers | M20K Blocks | Memory (Bits) |
|---|---|---|---|---|---|
| **Controller** | | | | | |
| DDR2 (Half rate) | 8 | 941 | 667 | 1 | 8,192 |
| | 16 | 948 | 671 | 2 | 16,384 |
| | 64 | 936 | 655 | 8 | 73,728 |
| | 72 | 941 | 671 | 8 | 80,896 |
| DDR2 (Full rate) | 8 | 959 | 651 | 1 | 4,096 |
| | 16 | 958 | 653 | 1 | 8,192 |
| | 64 | 962 | 657 | 4 | 34,816 |
| | 72 | 953 | 641 | 5 | 41,472 |
| DDR3 (Half rate) | 8 | 1,209 | 780 | 1 | 8,192 |
| | 16 | 1,212 | 784 | 2 | 16,384 |
| | 64 | 1,200 | 768 | 8 | 73,728 |
| | 72 | 1,205 | 784 | 8 | 80,896 |
| **PHY** | | | | | |
| DDR2 (Half rate) | 8 | 2,215 | 1,691 | 14 | 173,056 |
| | 16 | 2,322 | 1,783 | 14 | 173,056 |
| | 64 | 2,949 | 2,335 | 14 | 173,056 |
| | 72 | 3,055 | 2,427 | 14 | 173,056 |
| DDR2 (Full rate) | 8 | 2,181 | 1,685 | 14 | 173,056 |
| | 16 | 2,280 | 1,785 | 14 | 173,056 |
| | 64 | 2,804 | 2,385 | 14 | 173,056 |
| | 72 | 2,883 | 2,485 | 14 | 173,056 |
| DDR3 (Half rate) | 8 | 2,226 | 1,706 | 14 | 173,056 |
| | 16 | 2,345 | 1,798 | 14 | 173,056 |
| | 64 | 2,973 | 2,350 | 14 | 173,056 |
| | 72 | 3,076 | 2,442 | 14 | 173,056 |
| **Total** | | | | | |

**Table 1–8. Resource Utilization in Stratix V Devices (Part 2 of 2)**

| Protocol | Memory Width (Bits) | Combinational LCs | Logic Registers | M20K Blocks | Memory (Bits) |
|---|---|---|---|---|---|
| DDR2 (Half rate) | 8 | 3,156 | 2,358 | 15 | 181,248 |
| | 16 | 3,270 | 2,454 | 16 | 189,440 |
| | 64 | 3,885 | 2,990 | 22 | 246,784 |
| | 72 | 3,996 | 3,098 | 22 | 253,952 |
| DDR2 (Full rate) | 8 | 3,140 | 2,336 | 15 | 177,152 |
| | 16 | 3,238 | 2,438 | 15 | 181,248 |
| | 64 | 3,766 | 3,042 | 18 | 207,872 |
| | 72 | 3,836 | 3,126 | 19 | 214,528 |
| DDR3 (Half rate) | 8 | 3,435 | 2,486 | 15 | 181,248 |
| | 16 | 3,557 | 2,582 | 16 | 189,440 |
| | 64 | 4,173 | 3,118 | 22 | 246,784 |
| | 72 | 4,281 | 3,226 | 22 | 253,952 |

# System Requirements

The DDR2 and DDR3 SDRAM controllers with UniPHY are part of the MegaCore IP Library, which Altera distributes with the Quartus II software.

For system requirements and installation instructions, refer to *Altera Software Installation and Licensing*.

# 2. Getting Started

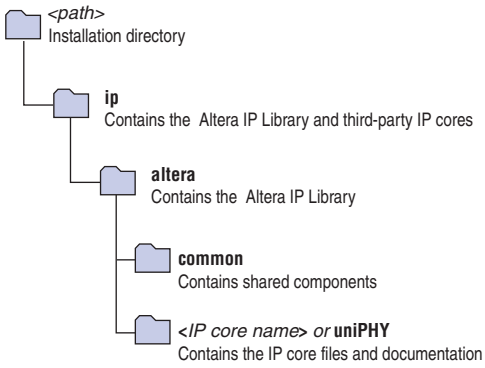
This chapter provides a general overview of the Altera IP core design flow to help you quickly get started with any Altera IP core. The Altera IP Library is installed as part of the Quartus II installation process. You can select and parameterize any Altera IP core from the library. Altera provides an integrated parameter editor that allows you to customize IP cores to support a wide variety of applications. The parameter editor guides you through the setting of parameter values and selection of optional ports. The following sections describe the general design flow and use of Altera IP cores.

## Installation and Licensing

The Altera IP Library is distributed with the Quartus II software and downloadable from the Altera website (www.altera.com).

Figure 2–1 shows the directory structure after you install an Altera IP core, where *<path>* is the installation directory. The default installation directory on Windows is **C:\altera\<***version number***>**; on Linux it is **/opt/altera<***version number***>.**

**Figure 2–1. IP core Directory Structure**




You can evaluate an IP core in simulation and in hardware until you are satisfied with its functionality and performance. Some IP cores require that you purchase a license for the IP core when you want to take your design to production. After you purchase a license for an Altera IP core, you can request a license file from the Altera Licensing page of the Altera website and install the license on your computer. For additional information, refer to *Altera Software Installation and Licensing*.

## Design Flows

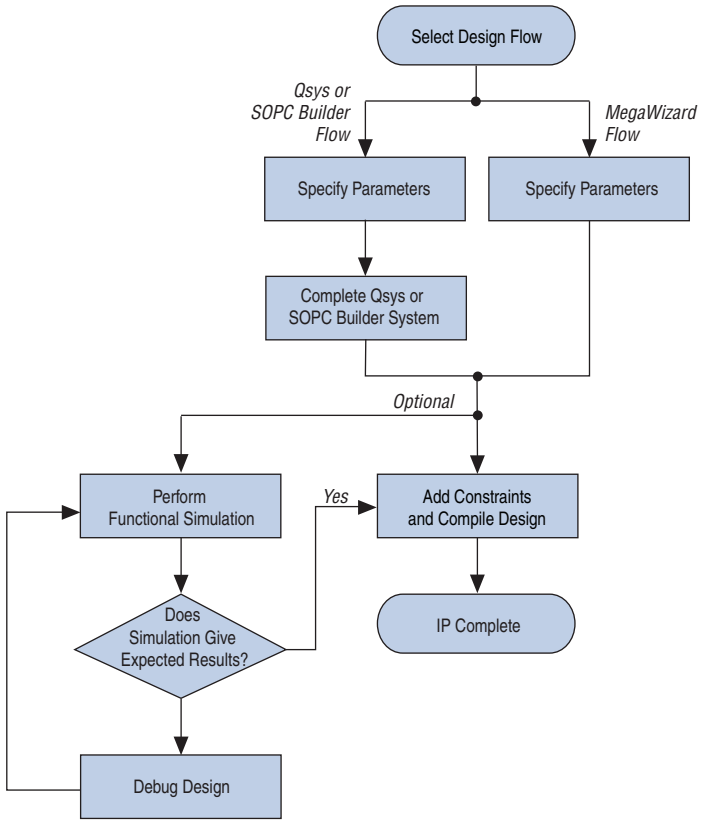
You can use the following flow(s) to parameterize Altera IP cores:

- MegaWizard Plug-In Manager Flow
- SOPC Builder Flow
- Qsys Flow

CAUTION

Altera's Qsys system integration tool is now available as beta for evaluation in the Quartus II software subscription edition version 10.1. Altera does not recommend using the beta release of Qsys in the Quartus II software version 10.1 for designs that are close to completion and are meeting design requirements. Before using Qsys, review the *Quartus II Software Version 10.1 Release Notes* and *AN 632: SOPC Builder to Qsys Migration Guidelines* for known issues and limitations. To submit general feedback or technical support on the beta release of Qsys, submit a service request through **mysupport.altera.com**. Alternatively, to submit general feedback, click **Feedback** on the Quartus II software Help menu.

**Figure 2–2. Design Flows**




The MegaWizard Plug-In Manager flow offers the following advantages:

- Allows you to parameterize an IP core variant and instantiate into an existing design
- For some IP cores, this flow generates a complete example design and testbench.

The SOPC Builder flow offer the following advantages:

- Generates simulation environment
- Allows you to integrate Altera-provided custom components
- Uses Avalon®memory-mapped (Avalon-MM) interfaces

The Qsys flow offers the following additional advantages over SOPC Builder:

- Provides visualization of hierarchical designs
- Allows greater performance through interconnect elements and pipelining
- Provides closer integration with the Quartus II software

# MegaWizard Plug-In Manager Flow

The MegaWizard Plug-In Manager flow allows you to customize your IP core and manually integrate the function into your design.

## Specifying Parameters

To specify IP core parameters with the MegaWizard Plug-In Manager, follow these steps:

1. Create a Quartus II project using the **New Project Wizard** available from the File menu.
2. In the Quartus II software, launch the **MegaWizard Plug-in Manager** from the Tools menu, and follow the prompts in the MegaWizard Plug-In Manager interface to create or edit a custom IP core variation.
3. To select a specific Altera IP core, click the IP core in the **Installed Plug-Ins** list in the MegaWizard Plug-In Manager.
4. Specify the parameters on the **Parameter Settings** pages. For detailed explanations of these parameters, refer to the "*Parameter Settings*" chapter in this document.

   Some IP cores provide preset parameters for specific applications. If you wish to use preset parameters, click the arrow to expand the **Presets** list, select the desired preset, and then click **Apply**. To modify preset settings, in a text editor edit the **<*installation directory*>\ip\altera\uniphy\lib\<*IP core*>.qprs** file.

5. If the IP core provides a simulation model, specify appropriate options in the wizard to generate a simulation model.

   Altera IP supports a variety of simulation models, including simulation-specific IP functional simulation models and encrypted RTL models, and plain text RTL models. These are all cycle-accurate models. The models allow for fast functional simulation of your IP core instance using industry-standard VHDL or Verilog HDL simulators. For some cores, only the plain text RTL model is generated, and you can simulate that model.

   For more information about functional simulation models for Altera IP cores, refer to *Simulating Altera Designs* in volume 3 of the *Quartus II Handbook*.

   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.

6. If the parameter editor includes **EDA** and **Summary** tabs, follow these steps:
   a. Some third-party synthesis tools can use a netlist that contains the structure of an IP core but no detailed logic to optimize timing and performance of the design containing it. To use this feature if your synthesis tool and IP core support it, turn on **Generate netlist**.
   b. On the **Summary** tab, if available, select the files you want to generate. A gray checkmark indicates a file that is automatically generated. All other files are optional.

   ☞ If file selection is supported for your IP core, after you generate the core, a generation report (*<variation name>***.html)** appears in your project directory. This file contains information about the generated files.

7. Click the **Finish** button, the parameter editor generates the top-level HDL code for your IP core, and a simulation directory which includes files for simulation.

   ☞ The **Finish** button may be unavailable until all parameterization errors listed in the messages window are corrected.

8. Click **Yes** if you are prompted to add the Quartus II IP File (**.qip**) to the current Quartus II project. You can also turn on **Automatically add Quartus II IP Files to all projects**.

You can now integrate your custom IP core instance in your design, simulate, and compile. While integrating your IP core instance into your design, you must make appropriate pin assignments. You can create virtual pin to avoid making specific pin assignments for top-level signals while you are simulating and not ready to map the design to hardware.

For some IP cores, the generation process also creates a complete example design in the *<variation_name>***_example_design_fileset/example_project/** directory. This example demonstrates how to instantiate and connect the IP core.

☞ For information about the Quartus II software, including virtual pins and the MegaWizard Plug-In Manager, refer to Quartus II Help.

## Simulate the IP Core

You can simulate your IP core variation with the functional simulation model and the testbench or example design generated with your IP core. The functional simulation model and testbench files are generated in a project subdirectory. This directory may also include scripts to compile and run the testbench.

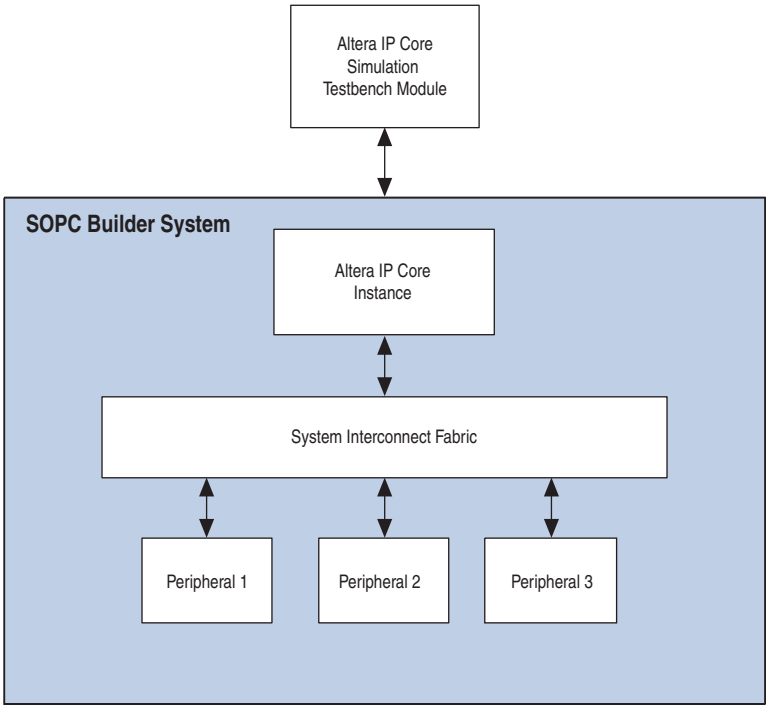
For a complete list of models or libraries required to simulate your IP core, refer to the scripts provided with the testbench.

For more information about simulating Altera IP cores, refer to *Simulating Altera Designs* in volume 3 of the *Quartus II Handbook*.

# SOPC Builder Design Flow

You can use SOPC Builder to build a system that includes your customized IP core. You easily can add other components and quickly create an SOPC Builder system. SOPC Builder automatically generates HDL files that include all of the specified components and interconnections. SOPC Builder defines default connections, which you can modify. The HDL files are ready to be compiled by the Quartus II software to produce output files for programming an Altera device. SOPC Builder generates a simulation testbench module for supported cores that includes basic transactions to validate the HDL files. Figure 2–3 shows a block diagram of an example SOPC Builder system.

**Figure 2–3. SOPC Builder System**




For more information about system interconnect fabric, refer to the *System Interconnect Fabric for Memory-Mapped Interfaces* and *System Interconnect Fabric for Streaming Interfaces* chapters in the *SOPC Builder User Guide* and to the *Avalon Interface Specifications*.

For more information about SOPC Builder and the Quartus II software, refer to the *SOPC Builder Features* and *Building Systems with SOPC Builder* sections in the *SOPC Builder User Guide* and to Quartus II Help.

## Specify Parameters

To specify IP core parameters in the SOPC Builder flow, follow these steps:

1. Create a new Quartus II project using the **New Project Wizard** available from the File menu.
2. On the Tools menu, click **SOPC Builder**.
3. For a new system, specify the system name and language.
4. On the **System Contents** tab, double-click the name of your IP core to add it to your system. The relevant parameter editor appears.
5. Specify the required parameters in the parameter editor. For detailed explanations of these parameters, refer to the *"Parameter Settings"* chapter in this document.

   Some IP cores provide preset parameters for specific applications. If you wish to use preset parameters, click the arrow to expand the **Presets** list, select the desired preset, and then click **Apply**. To modify preset settings, in a text editor edit the <*installation directory*>**\ip\altera\uniphy\lib**<*IP core*>**.qprs** file.

   If your design includes external memory interface IP cores, you must turn on **Generate power of two bus widths** on the **PHY Settings** tab when parameterizing those cores.

6. Click **Finish** to complete the IP core instance and add it to the system.

   The **Finish** button may be unavailable until all parameterization errors listed in the messages window are corrected.

## Complete the SOPC Builder System

To complete the SOPC Builder system, follow these steps:

1. Add and parameterize any additional components. Some IP cores include a complete SOPC Builder system design example.
2. Use the Connection panel on the **System Contents** tab to connect the components.
3. By default, clock names are not displayed. To display clock names in the **Module Name** column and the clocks in the **Clock** column in the **System Contents** tab, click **Filters** to display the **Filters** dialog box. In the **Filter** list, click **All.**
4. If you intend to simulate your SOPC builder system, on the **System Generation** tab, turn on **Simulation** to generate simulation files for your system.
5. Click **Generate** to generate the system. SOPC Builder generates the system and produces the <*system name*>**.qip** file that contains the assignments and information required to process the IP core or system in the Quartus II Compiler.
6. In the Quartus II software, click **Add/Remove Files in Project** and add the **.qip** file to the project.
7. Compile your design in the Quartus II software.

## Simulate the System

During system generation, you can specify whether SOPC Builder 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 and plain-text RTL design files that describe your system in the ModelSim simulation software.

For information about the latest Altera-supported simulation tools, refer to the *Quartus II Software Release Notes*.

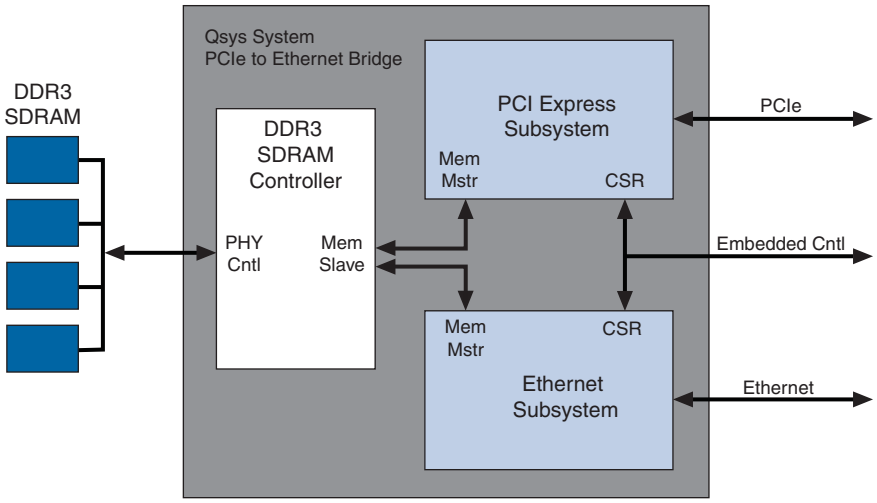
For information about simulating SOPC Builder systems, refer to the *SOPC Builder User Guide* and *AN 351: Simulating Nios II Embedded Processor Designs*.

For general information about simulating Altera IP cores, refer to *Simulating Altera Designs* in volume 3 of the *Quartus II Handbook*.

# Qsys System Integration Tool Design Flow

You can use the Qsys system integration tool to build a system that includes your customized IP core. You easily can add other components and quickly create a Qsys system. Qsys automatically generates HDL files that include all of the specified components and interconnections. In Qsys, you specify the connections you want. The HDL files are ready to be compiled by the Quartus II software to produce output files for programming an Altera device. Qsys generates Verilog HDL simulation models for the IP cores that comprise your system. Figure 2–4 shows a high level block diagram of an example Qsys system.

**Figure 2–4. Example Qsys System**



For more information about the Qsys system interconnect, refer to the *Qsys Interconnect* chapter in volume 1 of the *Quartus II Handbook* and to the *Avalon Interface Specifications*.

For more information about the Qsys tool and the Quartus II software, refer to the *System Design with Qsys* section in volume 1 of the *Quartus II Handbook* and to Quartus II Help.

## Specify Parameters

To specify parameters for your IP core using the Qsys flow, follow these steps:

1. Create a new Quartus II project using the **New Project Wizard** available from the File menu.
2. On the Tools menu, click **Qsys (Beta)**.
3. On the **System Contents** tab, double-click the name of your IP core to add it to your system. The relevant parameter editor appears.
4. Specify the required parameters in all tabs in the Qsys tool. For detailed explanations of these parameters, refer to the *"Parameter Settings"* chapter in this document.

   If your design includes external memory interface IP cores, you must turn on **Generate power of two bus widths** on the **PHY Settings** tab when parameterizing those cores.

   Some IP cores provide preset parameters for specific applications. If you wish to use preset parameters, click the arrow to expand the **Presets** list, select the desired preset, and then click **Apply**. To modify preset settings, in a text editor edit the ***<installation directory>*\ip\altera\uniphy\lib*<IP core>*.qprs** file.

5. Click **Finish** to complete the IP core instance and add it to the system.

   The **Finish** button may be unavailable until all parameterization errors listed in the messages window are corrected.

## Complete the Qsys System

To complete the Qsys system, follow these steps:

1. Add and parameterize any additional components.
2. Connect the components using the Connection panel on the **System Contents** tab.
3. In the **Export As** column, enter the name of any connections that should be a top-level Qsys system port. If the **Export As** column is not present, click the **Project Settings** tab and turn off **Use SOPC Builder port naming**.
4. If you intend to simulate your Qsys system, on the **Generation** tab, turn on one or more options under **Simulation** to generate desired simulation files.
5. If your system is not part of a Quartus II project and you want to generate synthesis RTL files, turn on **Create synthesis RTL files**.

6. Click **Generate** to generate the system. Qsys generates the system and produces the <*system name*>**.qip** file that contains the assignments and information required to process the IP core or system in the Quartus II Compiler.
7. In the Quartus II software, click **Add/Remove Files in Project** and add the **.qip** file to the project.
8. Compile your project in the Quartus II software.

### Simulate the System

During system generation, Qsys generates a functional simulation model—or example design that includes a testbench—which you can use to simulate your system in any Altera-supported simulation tool.

For information about the latest Altera-supported simulation tools, refer to the *Quartus II Software Release Notes*.

For general information about simulating Altera IP cores, refer to *Simulating Altera Designs* in volume 3 of the *Quartus II Handbook*.

For information about simulating Qsys systems, refer to the *System Design with Qsys* section in volume 1 of the *Quartus II Handbook*.

## HardCopy Migration Design Guidelines

If you intend to target your design to a HardCopy® device, ensure you use the following design guidelines:

- On the **General Settings** page of the **DDR2 SDRAM Controller with UniPHY** or **DDR3 SDRAM Controller with UniPHY** MegaWizard, turn on **HardCopy Compatibility Mode**, and then specify whether the **Reconfigurable PLL Location** is **Top_Bottom** or **Left_Right**.

  Altera recommends that you set the **Reconfigurable PLL Location** to the same side as your memory interface.

  When turned on, the **HardCopy Compatibility Mode** option enables run-time reconfiguration for all phase-locked loops (PLLs) and delay-locked loops (DLLs) instantiated in memory interfaces that are configured in PLL and DLL masters, and brings the necessary reconfiguration signals to the top level of the design.

  "Top-Level HardCopy Migration Signals" on page 6–12 lists the top-level signals generated for HardCopy migration.

- Enable run-time reconfiguration mode for all PLLs and DLLs instantiated in interfaces that are configured in PLL and DLL slaves.

For information about PLL megafunctions, refer to the *Phase-Locked Loop (ALTPLL) Megafunction User Guide* and the *Phase-Locked Loops Reconfiguration (ALTPLL_RECONFIG) Megafunctions User Guide*. For information about DLL megafunctions, refer to the *ALTDLL and ALTDQ_DQS Megafunctions User Guide*.

- Ensure that you place all memory interface pins close together. If, for example, address pins are located far away from data pins, closing timing might be difficult.

You can use the example top-level project that is generated when you turn on **HardCopy Migration** as a guide to help you connect the necessary signals in your design.

## Differences in UniPHY IP Generated with HardCopy Migration Support

When you generate a UniPHY memory interface for HardCopy device support, certain features in the IP are enabled that do not exist when you generate the IP core for only the FPGA. This section discusses those additional enabled features.

### ROM Loader for Designs Using Nios II Sequencer

An additional ROM loader is intantiated in the design for UniPHY designs that use the Nios II sequencer. The Nios II sequencer instruction code resides in RAM on either the HardCopy or FPGA device.

When you target only an FPGA device, the RAM is initialized when the device is programmed; however, HardCopy devices are not programmed and therefore the RAM cannot be initialized in this fashion. Instead, the Nios II sequencer instruction code must be stored in an external, non-volatile, ROM that loads the Nios II sequencer RAM through a ROM loader. You must attach the ROM loader to the appropriate pins connected to the external non-volatile ROM.

Table 2–1 summarizes the ports exposed at the top level of the PHY+Controller wrapper to expose the ROM loader utilized by the Nios II-based sequencer within the DDR2 or DDR3 PHY.

**Table 2–1. Top-level Ports that Connect to External ROM for Loading Nios II Code Memory (Part 1 of 2)**

| Port Name | Direction | Description |
|---|---|---|
| hc_rom_config_clock | Input | Write clock for the ROM loader. This clock is the write clock for the Nios II code memory. |
| hc_rom_config_datain | Input | Data input from external ROM. |
| hc_rom_config_rom_data_ready | Input | Asserts to the code memory loader that the word of memory is ready to be loaded. |
| hc_rom_config_init | Input | Signals that the Nios II code memory is being loaded from the external ROM. |
| hc_rom_config_init_busy | Output | Remains asserted throughout initialization and becomes inactive when initialization is complete. soft_reset_n can be issued after hc_rom_config_init_busy is deasserted. |

**Table 2–1. Top-level Ports that Connect to External ROM for Loading Nios II Code Memory (Part 2 of 2)**

| Port Name | Direction | Description |
|---|---|---|
| hc_rom_config_rom_rden | Output | Read-enable signal that connects to the external ROM. |
| hc_rom_config_rom_address | Output | ROM address that connects to the external ROM. |

## PLL/DLL Run-time Reconfiguration

The PLLs and DLLs in the HardCopy design have run-time reconfiguration enabled—provided that they are not in PLL/DLL slave mode.

When the PLLs and DLLs are generated with reconfiguration enabled, there are extra signals that must be connected and driven by user logic. In the example design generated during IP core generation, the PLL/DLL reconfiguration signals are brought to the top level and connected to constants, as shown in Figure 2–5.

For information about PLL megafunctions and reconfiguration, refer to the *Phase-Locked Loop (ALTPLL) Megafunction User Guide* and the *Phase-Locked Loops Reconfiguration (ALTPLL_RECONFIG) Megafunctions User Guide*.

**Figure 2–5. HardCopy UniPHY Example Design**

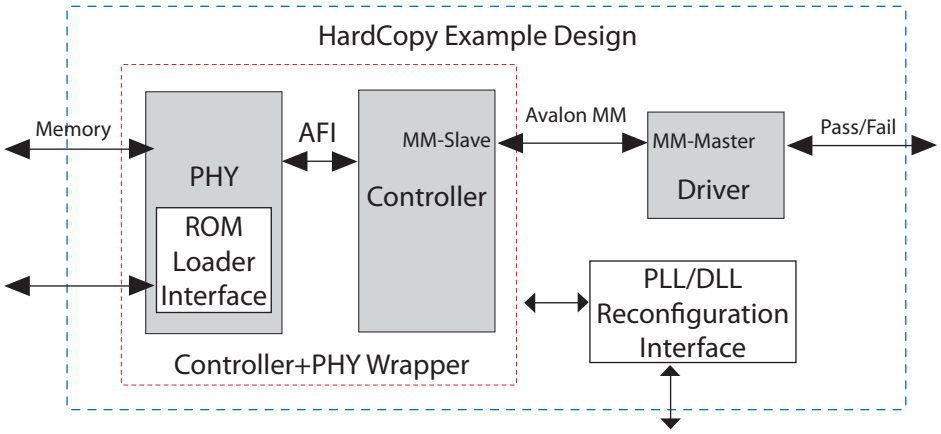



Table 2–2 summarizes the DLL reconfiguration ports exposed at the top level of the Controller+PHY.

**Table 2–2. DLL Reconfiguration Ports Exposed at Top-Level of Controller+PHY Wrapper (Part 1 of**

| Port Name | Direction | Description |
|---|---|---|
| hc_dll_config_dll_offset_ctrl_addnsub | Input | Addition/subtraction control port for the DLL. This port controls if the delay-offset setting on hc_dll_config_dll_offset_ctrl_offset is added or subtracted. |

**Table 2–2. DLL Reconfiguration Ports Exposed at Top-Level of Controller+PHY Wrapper (Part 2 of**

| Port Name | Direction | Description |
|---|---|---|
| hc_dll_config_dll_offset_ctrl_offset | Input | Offset input setting for the PLL. This is a Gray-coded offset that is added or subtracted from the current value of the DLL's delay chain. |
| hc_dll_config_dll_offset_ctrl_offsetctrlout | Output | The registered and Gray-coded value of the current delay-offset setting. |

Table 2–3 summarizes the ports exposed at the top level of the Controller and PHY wrapper to allow PLL reconfiguration.

**Table 2–3. PLL Reconfiguration Ports Exposed at the Top-Level of Controller+PHY Wrapper**

| Port Name | Direction | Description |
|---|---|---|
| hc_pll_config_configupdate | Input | Control signal to enable PLL reconfiguration. (Applies to RLDRAMII and QDRII only, the phase reconfiguration feature for DDR2/3 is included in the CSR port.) |
| hc_pll_config_phasecounterselect | Input | Specifies the counter select for dynamic phase adjustment. (Applies to RLDRAMII and QDR II only.) |
| hc_pll_config_phasestep | Input | Specifies the phase step for dynamic phase shifting. (Applies to RLDRAMII and QDR II only.) |
| hc_pll_config_phaseupdown | Input | Specifies if the phase adjustment should be up or down. (Applies to RLDRAMII and QDR II only.) |
| hc_pll_config_scanclk | Input | PLL reconfiguration scan chain clock. |
| hc_pll_config_scanclkena | Input | Clock enable port of the hc_pll_config_scanclk clock. |
| hc_pll_config_scandata | Input | Serial input data for the PLL reconfiguration scan chain. |
| hc_pll_config_phasedone | Output | When asserted, this signal indicates to core logic that phase adjustment is completed and that the PLL is ready to act on a possible second adjustment pulse. |
| hc_pll_config_scandataout | Output | The data output of the serial scan chain. |
| hc_pll_config_scandone | Output | Asserted when the scan chain write operation is in progress and is deasserted when the write operation is complete. |

To facilitate placement and timing closure and help compensate for PLLs adjacent to I/Os and vertical I/O overhang issues that can occur when targeting HardCopy III and HardCopy IV devices, an additional pipeline stage is added to the write path in the RTL when you turn on **HardCopy Compatibility**. The additional pipeline stage is added in all cases, except when CAS write latency equals 2 (for DDR3) or CAS latency equals 3 (for DDR2), where the additional pipeline stage is not required to meet timing requirements. The additional pipeline stage does not affect the overall latency of the controller.

For information about HardCopy issues such as vertical I/O overhang, PLLs adjacent to I/Os, and timing closure, refer to HardCopy III Device I/O Features in the *HardCopy III Device Handbook, Volume 1*, and HardCopy IV Device I/O Features in the *HardCopy IV Device Handbook, Volume 1*.

# Generated Files

When you complete the IP generation flow, there are generated files created in your project directory. The directory structure created varies somewhat, depending on the tool used to parameterize and generate the IP.

The PLL parameters are statically defined in the <*variation_name*>**_parameters.tcl** at generation time. To ensure timing constraints and timing reports are correct, when you use the GUI to make changes to the PLL component, apply those changes to the PLL parameters in this file.

## MegaWizard Plug-in Manager Flow

The tables in this section list the generated directory structure and key files of interest to users, resulting from the MegaWizard Plug-in Manager flow.

### Synthesis

Table 2–4 lists the generated directory structure and key files created by the synthesis flow with the MegaWizard Plug-in Manager.

**Table 2–4. Generated Directory Structure and Key Files—MWPIM Synthesis Flow**

| Directory | File Name | Description |
|---|---|---|
| <*working_dir*>/ | <*variation_name*>.qip | QIP file which refers to all generated files in the synthesis fileset. |
| <*working_dir*>/ | <*variation_name*>.v (for Verilog), or <*variation_name*>.vhd (for VHDL) | Top-level wrapper for synthesis files. |
| <*working_dir*>/<*variation_name*>/ | <*variation_name*>_<*stamp*>.v (1) | UniPHY top-level wrapper. |
| <*working_dir*>/<*variation_name*>/ | <*variation_name*>_<*stamp*>_*.v (1) | UniPHY Verilog RTL files. |
| <*working_dir*>/<*variation_name*>/ | <*variation_name*>_<*stamp*>_*.sv (1) | UniPHY SystemVerilog RTL files. |
| <*working_dir*>/<*variation_name*>/ | <*variation_name*>_<*stamp*>.sdc (1) | Synopsys constraints file. |
| <*working_dir*>/<*variation_name*>/ | <*variation_name*>_<*stamp*>.ppf (1) | Pin Planner file. |
| <*working_dir*>/<*variation_name*>/ | <*variation_name*>_<*stamp*>_pin_assignments.tcl (1) | Pin constraints script to be run after synthesis. |
| <*working_dir*>/<*variation_name*>/ | <*variation_name*>_<*stamp*>_*.tcl (1) | Other Tcl scripts. |
| <*working_dir*>/<*variation_name*>/ | <*variation_name*>_<*stamp*>_*.hex (1) | Sequencer memory files. |
| <*working_dir*>/<*variation_name*>/ | <*variation_name*>_<*stamp*>_*.mif (1) | Sequencer memory initialization files. |
| <*working_dir*>/<*variation_name*>/ | <*variation_name*>_<*stamp*>_readme.txt (1) | Readme text file. |

**Note to Table 2–4:**

(1) <*stamp*> is a unique identifier determined by the MegaWizard Plug-in Manager at generation time.

## Simulation

Table 2–5 lists the generated directory structure and key files created by the Verilog simulation flow with the MegaWizard Plug-in Manager.

**Table 2–5. Generated Directory Structure and Key Files—MWPIM Simulation Flow (Verilog)**

| Directory | File Name | Description |
|---|---|---|
| <*working_dir*>/<*variation_name*>_sim/ | <*variation_name*>.v (for Verilog), or <*variation_name*>.vho (for VHDL) | UniPHY top-level wrapper. |
| <*working_dir*>/<*variation_name*>_sim/ | <*variation_name*>_*.v | UniPHY Verilog RTL files. |
| <*working_dir*>/<*variation_name*>_sim/ | <*variation_name*>_*.sv | UniPHY SystemVerilog RTL files. |
| <*working_dir*>/<*variation_name*>_sim/ | <*variation_name*>_*.hex | Sequencer memory files. |
| <*working_dir*>/<*variation_name*>_sim/ | <*variation_name*>_*.mif | Sequencer memory initialization files. |
| <*working_dir*>/<*variation_name*>_sim/ | <*variation_name*>_readme.txt | Readme text file. |

Table 2–6 lists the generated directory structure and key files created by the VHDL simulation flow with the MegaWizard Plug-in Manager.

**Table 2–6. Generated Directory Structure and Key Files—MWPIM Simulation Flow (VHDL)**

| Directory | File Name | Description |
|---|---|---|
| <*working_dir*>/<*variation_name*>_sim/ | <*variation_name*>.vho | UniPHY VHDL top-level module. |
| <*working_dir*>/<*variation_name*>_sim/ | <*variation_name*>_*.vhd<br><*variation_name*>_*.vho | UniPHY simulation VHDL files. |
| <*working_dir*>/<*variation_name*>_sim/ | vhdl_files.txt | File list text file. |

## Example Design

Table 2–7 lists the generated directory structure and key files created for the example design with the MegaWizard Plug-in Manager

**Table 2–7. Generated Directory Structure and Key Files—MWPIM Example Design (Part 1 of 2)**

| Directory | File Name‘ | Description |
|---|---|---|
| <*working_dir*>/<*variation_name*>_example_design_fileset/ | <*variation_name*>.qip | QIP which refers to UniPHY RTL in this fileset. This is distinct from ../<*variation_name*>.qip. This file is included automatically in the example project. |
| <*working_dir*>/<*variation_name*>_example_design_fileset/ | <*variation_name*>.v | UniPHY top-level wrapper. |
| <*working_dir*>/<*variation_name*>_example_design_fileset | <*variation_name*>_*.v | UniPHY Verilog RTL files. |

**Table 2–7. Generated Directory Structure and Key Files—MWPIM Example Design (Part 2 of 2)**

| Directory | File Name‘ | Description |
|---|---|---|
| <*working_dir*>/<*variation_name*>_example_design_fileset | <*variation_name*>_*.sv | UniPHY SystemVerilog RTL fies. |
| <*working_dir*>/<*variation_name*>_example_design_fileset/ | <*variation_name*>.sdc | Synopsys constraints file. |
| <*working_dir*>/<*variation_name*>_example_design_fileset/ | <*variation_name*>.ppf | Pin Planner file. |
| <*working_dir*>/<*variation_name*>_example_design_fileset/ | <*variation_name*>_pin_assignments.tcl | Pin constraints script to be run after synthesis. |
| <*working_dir*>/<*variation_name*>_example_design_fileset/ | <*variation_name*>_*.tcl | Other Tcl scripts. |
| <*working_dir*>/<*variation_name*>_example_design_fileset/ | <*variation_name*>_*.hex | Sequencer memory files. |
| <*working_dir*>/<*variation_name*>_example_design_fileset/ | <*variation_name*>_*.mif | Sequencer memory initialization files. |
| <*working_dir*>/<*variation_name*>_example_design_fileset/ | <*variation_name*>_readme.txt | Readme text file. |
| <*working_dir*>/<*variation_name*>_example_design_fileset/example_project/ | <*variation_name*>_example_top.qpf | Example design project file. |
| <*working_dir*>/<*variation_name*>_example_design_fileset/example_project/ | <*variation_name*>_example_top.qsf | Example design project settings file. |
| <*working_dir*>/<*variation_name*>_example_design_fileset/example_project/ | <*variation_name*>_example_top.v | Top-level wrapper including UniPHY, traffic generator, and memory model. |
| <*working_dir*>/<*variation_name*>_example_design_fileset/example_project/ | <*variation_name*>_*.v | Other example design Verilog RTL files. |
| <*working_dir*>/<*variation_name*>_example_design_fileset/example_project/ | <*variation_name*>_*.sv | Other example design SystemVerilog RTL files. |
| <*working_dir*>/<*variation_name*>_example_design_fileset/example_project/ | <*prefix*>_mem_model.sv (1) | Generic memory model. |
| <*working_dir*>/<*variation_name*>_example_design_fileset/rtl_sim/ | <*variation_name*>_example_top_tb.v | Top-level test bench. |

**Note to Table 2–7:**

(1) <*prefix*> varies depending on protocol and type of memory model.

## SOPC Builder Flow

Table 2–8 lists the generated directory structure and key files created by the SOPC Builder flow.

**Table 2–8. Generated Directory Structure and Key Files—SOPC Builder Flow**

| Directory | File Name‘ | Description |
|---|---|---|
| <*working_dir*>/ | <*system_name*>.qip | QIP which refers to all generated files in the SOPC Builder project. |
| <*working_dir*>/ | <*system_name*>.v | SOPC Builder system top-level wrapper. |
| <*working_dir*>/ | <*core_name*>_<*stamp*>.v (1) | UniPHY top-level wrapper. |
| <*working_dir*>/ | <*core_name*>_<*stamp*>_*.v (1) | UniPHY Verilog RTL files. |
| <*working_dir*>/ | <*core_name*>_<*stamp*>_*.sv (1) | UniPHY SystemVerilog RTL files. |
| <*working_dir*>/ | <*core_name*>_<*stamp*>.sdc (1) | Synopsys constraints file. |
| <*working_dir*>/ | <*core_name*>_<*stamp*>.ppf (1) | Pin Planner file. |
| <*working_dir*>/ | <*core_name*>_<*stamp*>_pin_assignments.tcl (1) | Pin constraints script to be run after synthesis. |
| <*working_dir*>/ | <*core_name*>_<*stamp*>_*.tcl (1) | Other Tcl scripts. |
| <*working_dir*>/ | <*core_name*>_<*stamp*>_*.hex (1) | Sequencer memory files. |
| <*working_dir*>/ | <*core_name*>_<*stamp*>_*.mif (1) | Sequencer memory initialization files. |
| <*working_dir*>/ | <*core_name*>_<*stamp*>_readme.txt (1) | Readme text file. |
| <*working_dir*>/ | *Other IP core files.* | Other IP cores. |

**Note to Table 2–8:**

(1) <*stamp*> is a unique identifier determined by SOPC Builder at generation time.

## Qsys Flow

The tables in this section list the generated directory structure and key files of interest to users, resulting from the Qsys flow

### Synthesis

Table 2–9 lists the generated directory structure and key files created by the synthesis flow with Qsys.

**Table 2–9. Generated Directory Structure and Key Files—Qsys Synthesis Flow (Part 1 of 2)**

| Directory | File Name | Description |
|---|---|---|
| <*working_dir*>/<*system_name*>/synthesis/ | <*system_name*>.qip | QIP which refers to all generated files in the Qsys system synthesis fileset. |
| <*working_dir*>/<*system_name*>/synthesis/ | <*system_name*>.v | Qsys system top-level wrapper. |
| <*working_dir*>/<*system_name*>/synthesis/submodules/ | <*core_name*>_<*stamp*>.v (1) | UniPHY top-level wrapper. |
| <*working_dir*>/<*system_name*>/synthesis/submodules/ | <*core_name*>_<*stamp*>_*.v (1) | UniPHY Verilog RTL files. |

**Table 2–9. Generated Directory Structure and Key Files—Qsys Synthesis Flow (Part 2 of 2)**

| Directory | File Name | Description |
|---|---|---|
| <*working_dir*>/<*system_name*>/synthesis/submodules/ | <*core_name*>_<*stamp*>_*.sv (1) | UniPHY SystemVerilog RTL files. |
| <*working_dir*>/<*system_name*>/synthesis/submodules/ | <*core_name*>_<*stamp*>.sdc (1) | Synopsys constraints file. |
| <*working_dir*>/<*system_name*>/synthesis/submodules/ | <*core_name*>_<*stamp*>.ppf (1) | Pin Planner file. |
| <*working_dir*>/<*system_name*>/synthesis/submodules/ | <*core_name*>_<*stamp*>_pin_assignments.tcl (1) | Pin constraints script to be run after synthesis. |
| <*working_dir*>/<*system_name*>/synthesis/submodules/ | <*core_name*>_<*stamp*>_*.tcl (1) | Other Tcl scripts. |
| <*working_dir*>/<*system_name*>/synthesis/submodules/ | <*core_name*>_<*stamp*>_*.hex (1) | Sequencer memory files. |
| <*working_dir*>/<*system_name*>/synthesis/submodules/ | <*core_name*>_<*stamp*>_*.mif (1) | Sequencer memory initialization files. |
| <*working_dir*>/<*system_name*>/synthesis/submodules/ | <*core_name*>_<*stamp*>_readme.txt (1) | Readme text file. |
| <*working_dir*>/<*system_name*>/synthesis/submodules/ | *Other IP core files* | Other IP core files. |

**Note to Table 2–9**

(1) <*stamp*> is a unique identifier created by Qsys during generation.

## Verilog HDL Simulation

Table 2–10 lists the generated directory structure and key files created by the Verilog simulation flow with Qsys.

**Table 2–10. Generated Directory Structure and Key Files—Qsys Verilog HDL Simulation**

| Directory | File Name | Description |
|---|---|---|
| <*working_dir*>/<*system_name*>/sim_verilog/ | <*system_name*>.v | Qsys system top-level wrapper. |
| <*working_dir*>/<*system_name*>/sim_verilog/submodules/ | <*core_name*>_<*stamp*>.v | UniPHY top-level wrapper. |
| <*working_dir*>/<*system_name*>/sim_verilog/submodules/ | <*core_name*>_<*stamp*>_*.v | UniPHY Verilog RTL files. |
| <*working_dir*>/<*system_name*>/sim_verilog/submodules/ | <*core_name*>_<*stamp*>_*.sv | UniPHY SystemVerilog RTL files. |
| <*working_dir*>/<*system_name*>/sim_verilog/submodules/ | <*core_name*>_<*stamp*>_*.hex | Sequencer memory files. |
| <*working_dir*>/<*system_name*>/sim_verilog/submodules/ | <*core_name*>_<*stamp*>_*.mif | Sequencer memory initialization files. |
| <*working_dir*>/<*system_name*>/sim_verilog/submodules/ | <*core_name*>_<*stamp*>_readme.txt (1) | Readme text file. |
| <*working_dir*>/<*system_name*>/sim_verilog/submodules/ | *Other IP core files* | Other IP core files. |

**Note for Table 2–10:**

(1) <*stamp*> is a unique identifier created by Qsys during generation.

## VHDL Simulation

Table 2–11 lists the generated directory structure and key files created by the VHDL simulation flow with Qsys.

**Table 2–11. Generated Directory Structure and Key Files—Qsys VHDL Simulation**

| Directory | File Name | Description |
|---|---|---|
| <*working_dir*>/<*system_name*>/sim_vhdl/ | <*system_name*>.vhd | Qsys system top-level wrapper. |
| <*working_dir*>/<*system_name*>/sim_vhdl/submodules/ | <*core_name*>_<*stamp*>.vho (1) | UniPHY VHDL top-level module. |
| <*working_dir*>/<*system_name*>/sim_vhdl/submodules/ | <*core_name*>_<*stamp*>_*.vhd<br><*core_name*>_<*stamp*>_*.vho (1) | UniPHY VHDL simulation files. |
| <*working_dir*>/<*system_name*>/sim_vhdl/submodules/ | vhdl_files.txt | File list text file. |

**Note to Table 2–11:**

(1) <*stamp*> is a unique identifier created by Qsys during generation.

This chapter describes the parameters you can set in the UniPHY GUI.

# PHY Settings

Use this tab to apply the PHY settings suitable for your design.

## FPGA

**Speed grade** is the speed grade of the targeted FPGA device, which affects the generated timing constraints and timing reporting.

## Clocks

Table 3–10 shows the clock parameters.

**Table 3–1. Clock Parameters**

| Parameter | Description |
|---|---|
| **Memory clock frequency** | The frequency of the clock that drives the memory device. Use up to 4 decimal places of precision. |
| **PLL reference clock frequency** | The frequency of the input clock that feeds the PLL. Use up to 4 decimal places of precision. |
| **Full or half rate on Avalon-MM interface** | The width of data bus on the Avalon-MM interface. **Full** results in a width of 2× the memory data width. **Half** results in a width of 4× the memory data width. |

## Advanced Settings

Table 3–2 shows the advanced settings parameters.

**Table 3–2. Advanced Settings (Part 1 of 2)**

| Parameter | Description |
|---|---|
| **Advanced clock phase control** | Enables access to clock phases. Default value should suffice for most DIMMs and board layouts, but can be modified if necessary to compensate for larger address and command versus clock skews. |
| **Additional address and command clock phase** | Allows you to increase or decrease the amount of phase shift on the address and command clock. The base phase shift center aligns the address and command clock at the memory device, which may not be the optimal setting under all circumstances. Increasing or decreasing the amount of phase shift can improve timing. The default value is 0 degrees. |

**Table 3–2. Advanced Settings (Part 2 of 2)**

| Parameter | Description |
|---|---|
| **Additional CK/CK# phase** | Allows you to increase or decrease the amount of phase shift on the CK/CK# clock. The base phase shift center aligns the address and command clock at the memory device, which may not be the optimal setting under all circumstances. Increasing or decreasing the amount of phase shift can improve timing. Increasing or decreasing the phase shift on CK/CK# also impacts the read, write, and leveling transfers, which increasing or decreasing the phase shift on the address and command clocks does not. |
| **I/O standard** | The I/O standard voltage. |
| **Master for PLL/DLL sharing** | Specifies that the IP core instantiates its own PLL and DLL. All of the PLL clocks and DLL delay values are exported for use by other identical UniPHY-based IP cores that have this option turned off. |
| **Master for OCT control block** | Causes UniPHY datapath to instantiate the required OCT control block. When this parameter is turned off, you must instantiate this block and connect the termination control bus signals to the PHY, or share an OCT control block from another UniPHY instantiation that is in master mode. |
| **HardCopy compatibility** | Enables all required HardCopy compatibility options for the generated IP core. For some parameterizations, a pipeline stage is added to the write datapath to help the more challenging timing closure in HardCopy; the pipeline stage does not affect overall read and write latency. |

## Example Testbench Simulation Options

Table 3–3 shows the simulation options available for the example testbench.

**Table 3–3. Example Testbench Simulation Options**

| Parameter | Description |
|---|---|
| **Autocalibration mode** | Specifies whether you want to improve simulation performance by reducing calibration. There is no change to the generated RTL. The following autocalibration modes are available:<br>■ **Skip calibration**—provides the fastest simulation. It loads the settings calculated from the memory configuration and enters user mode.<br>■ **Quick calibration**—calibrates (without centering) one bit per group before entering user mode.<br>■ **Full calibration**—calibrates the same as in hardware, and includes all phases, delay sweeps, and centering on every data bit. You can use timing annotated memory models. Be aware that full calibration can take hours or days to complete. |
| **Skip memory initialization** | Causes the example testbench to skip the memory initialization sequence. This setting does not change the generated RTL, but can speed up simulation. |

# Memory Parameters

Use this tab to apply the memory parameters from your memory manufacturer's data sheet.

Table 3–4 shows the memory parameters.

**Table 3–4. Memory Parameters**

| Parameter | Description |
|---|---|
| **Memory vendor** | The vendor of the memory device, which is set automatically when you select a configuration from the list of memory presets. |
| **Memory format** | The format of the memory device. |
| **Memory device speed grade** | The maximum frequency at which the memory device can run. |
| **Total interface width** | The total number of DQ pins of the memory device. Limited to 144 bits for DDR2 and DDR3 SDRAM (with or without leveling). |
| **DQ/DQS group size** | The number of DQ bits per DQS group. |
| **Number of chip selects** | The number of chip-selects the IP core uses for the current device configuration. |
| **Number of clocks per chip select** | The width of the clock bus to each chip select of the memory interface. |
| **Number of slots** | Available when DIMM is selected, for DDR2 and DDR3. |
| **Fly-by topology** | Available for discrete devices with total interface width >8. Applies only to DDR3. |
| **DQS# Enable** | Available only for DDR2. |
| **Row address width** | The width of the row address on the memory interface. |
| **Column address width** | The width of the column address on the memory interface. |
| **Bank-address width** | The width of the bank address bus on the memory interface. |
| **Drive DM pins from FPGA** | Specifies whether the DM pins of the memory device are driven by the FPGA. You can turn off this option to avoid overusing FPGA device pins when using ×4 mode memory devices. |
| **Number of DQS groups** | Calculated automatically from the total interface width and the DQ and DQS group size parameters. |

## Memory Initialization Options—DDR2

The following tables describe the memory initialization parameters.

### Mode Register 0

Table 3–5 shows the mode register 0 parameters.

**Table 3–5. Mode Register 0 Parameters (Part 1 of 2)**

| Parameter | Description |
|---|---|
| **READ burst type** | Determines whether the controller performs accesses within a given burst in sequential or interleaved order. |

**Table 3–5. Mode Register 0 Parameters (Part 2 of 2)**

| Parameter | Description |
|---|---|
| **DLL precharge power down** | Determines whether the DLL in the memory device is in slow exit mode or in fast exit mode during precharge power down. |
| **Memory CAS latency setting** | Determines the number of clock cycles between the READ command and the availability of the first bit of output data at the memory device. |

### Mode Register 1

Table 3–6 shows the mode register 1 parameters.

**Table 3–6. Mode Register 1 Parameters**

| Parameter | Description |
|---|---|
| **Output drive strength setting** | Determines the output driver impedance setting at the memory device. |
| **Memory additive CAS latency setting** | Determines the posted CAS additive latency of the memory device. |
| **Memory on-die termination (ODT) setting** | Determines the on-die termination resistance at the memory device. |

### Mode Register 2

**SRT Enable** determines whether selfrefresh temperature (SRT) enable is **1x refresh rate** or **2x refresh rate** (for high-temperature operations).

## Memory Initialization Options—DDR3

The **Mirror addressing: 1 per chip select** parameter is the mirror addressing. For example for four CS, `1101` makes CS #3, #2 and #0 mirrored.

### Mode Register 0

Table 3–7 shows the mode register 0 parameters.

**Table 3–7. Mode Register 0 Parameters**

| Parameter | Description |
|---|---|
| **READ burst type** | Specifies whether accesses within a given burst are in sequential or interleaved order. |
| **DLL precharge power down** | Specifies whether the DLL in the memory device is off or on during precharge power-down. |
| **Memory CAS latency setting** | The number of clock cycles between the read command and the availability of the first bit of output data at the memory device. |

### Mode Register 1

Table 3–8 shows the mode register 1 parameters.

**Table 3–8. Mode Register 1 Parameters**

| Parameter | Description |
|---|---|
| **Output drive strength setting** | The output driver impedance setting at the memory device. |
| **Memory additive CAS latency setting** | The posted CAS additive latency of the memory device. |
| **ODT Rtt nominal value** | The on-die termination resistance at the memory device. |

### Mode Register 2

Table 3–9 shows the mode register 2 parameters.

**Table 3–9. Mode Register 2 Parameters**

| Parameter | Description |
|---|---|
| **Auto selfrefresh method** | Disable or enable auto selfrefresh. |
| **Selfrefresh temperature** | Specifies the selfrefresh temperature as **Normal** or **Extended**. |
| **Memory write CAS latency setting** | The number of clock cycles from the releasing of the internal write to the latching of the first data in, at the memory device. |
| **Dynamic ODT (Rtt_WR) value** | The mode of the dynamic ODT feature of the memory device. |

## Memory Timing

Use this tab to apply the memory timings from your memory manufacturer's data sheet. Table 3–10 shows the memory timing parameters.

**Table 3–10. Memory Timing Parameters (Part 1 of 2)**

| Parameter | Description |
|---|---|
| **tIS (base)** | Address and control setup to CK clock rise. |
| **tIH (base)** | Address and control hold after CK clock rise. |
| **tDS (base)** | Data setup to clock (DQS) rise. |
| **tDH (base)** | Data hold after clock (DQS) rise. |
| **tDQSQ** | DQS, DQS# to DQ skew, per access. |
| **tQHS** (DDR2)<br>**tQH** (DDR3) | DQ output hold time from DQS, DQS# (absolute time value)<br>DQ output hold time from DQS, DQS# (percentage of tCK). |
| **tDQSCK** | DQS output access time from CK/CK#. |
| **tDQSS** | First latching edge of DQS to associated clock edge (percentage of tCK). |
| **tQSH** (DDR3)<br>**tDQSH** (DDR2) | DQS Differential High Pulse Width (percentage of tCK). Specifies the minimum high time of the DQS signal received by the memory. |
| **tDSH** | DQS falling edge hold time from CK (percentage of tCK). |

**Table 3–10. Memory Timing Parameters (Part 2 of 2)**

| Parameter | Description |
|---|---|
| **tDSS** | DQS falling edge to CK setup time (percentage of tCK). |
| **tINIT** | Memory initialization time at power-up. |
| **tMRD** | Load mode register command period. |
| **tRAS** | Active to precharge time. |
| **tRCD** | Active to read or write time. |
| **tRP** | Precharge command period. |
| **tREFI** | Refresh command interval. |
| **tRFC** | Auto-refresh command interval. |
| **tWR** | Write recovery time. |
| **tWTR** | Write to read period. |
| **tFAW** | Four active window time. |
| **tRRD** | RAS to RAS delay time. |
| **tRTP** | Read to precharge time. |

# Board Settings

Use the **Board Settings** tab to model the board-level effects in the timing analysis.

The IP core supports single and multiple chip-select configurations. Altera has determined the effects on the output signalling of these configurations for certain Altera boards, and has stored the effects on the output slew rate and the intersymbol interference (ISI) within the wizard.

These stored values are representative of specific Altera boards. You must change the values to account for the board-level effects for your board. You can use HyperLynx or similar simulators to obtain values that are representative of your board.

The **Board Settings** tab allows you to enter board-related data. In the **Intersymbol Interference** and **Board Skews** sections, you enter information derived during your PCB development process of prelayout (line) and postlayout (board) simulation.

Bus turnaround is not timing analyzed; consequently, the controller dead times are based on assumptions about the user board trace lengths. For timing analysis to be accurate, board trace delays must be limited to 0.6 ns from FPGA to memory and from memory to FPGA.

For more information about how to include your board simulation results in the Quartus II software and how to assign pins using pin planners, refer to *Volume 6: Design Flow Tutorials* of the *External Memory Interface Handbook*. For information about timing deratio n methodology, refer to Chapter 3, Timing Deration Methodology for Multiple Chip Select DDR2 and DDR3 SDRAM Designs in Volume 4 of the *External Memory Interface Handbook*.

## Setup and Hold Derating

The slew rate of the output signals affects the setup and hold times of the memory device.

You can specify the slew rate of the output signals to see their effect on the setup and hold times of both the address and command signals and the DQ signals, or specify the setup and hold times directly. Table 3–11 shows the setup and hold derating parameters.

**Table 3–11. Setup and Hold Derating Parameters**

| Parameter | Description |
|---|---|
| **Derating method** | Derating method. |
| **CK/CK# slew rate (differential)** | CK/CK# slew rate (differential). |
| **Address/Command slew rate** | Address and command slew rate. |
| **DQS/DQS# slew rate (Differential)** | DQS and DQS# slew rate (differential). |
| **DQ slew rate** | DQ slew rate. |
| **tIS** | Address/command setup time to CK. |
| **tIH** | Address/command hold time from CK. |
| **tDS** | Data setup time to DQS. |
| **tDH** | Data hold time from DQS. |

## ISI

ISI is the distortion of a signal in which one symbol interferes with subsequent symbols. Typically, when going from a single chip-select configuration to a multiple chip-select configuration there is an increase in ISI because there are multiple stubs causing reflections. Table 3–12 shows the ISI parameters.

**Table 3–12. ISI Parameters**

| Parameter | Description |
|---|---|
| **Derating method** | Choose between default Altera settings (with specific Altera boards) or manually enter board simulation numbers obtained for your specific board. |
| **Address and command eye reduction (setup)** | The reduction in the eye diagram on the setup side (or left side of the eye) due to ISI on the address and command signals compared to a case when there is no ISI. |
| **Address and command eye reduction (hold)** | The reduction in the eye diagram on the hold side (or right side of the eye) due to ISI on the address and command signals compared to a case when there is no ISI. |
| **DQ eye reduction** | The total reduction in the eye diagram due to ISI on DQ signals compared to a case when there is no ISI. Altera assumes that the ISI reduces the eye width symmetrically on the left and right side of the eye. |
| **Delta DQS arrival time** | The increase in variation on the range of arrival times of DQS compared to a case when there is no ISI. Altera assumes that the ISI causes DQS to further vary symmetrically to the left and to the right. |

## Board Skews

PCB traces can have skews between them that can reduce timing margins. Furthermore, skews between different chip selects can further reduce the timing margin in multiple chip-select topologies. The **Board Skews** section of the parameter editor allows you to enter parameters to compensate for these variations. Very large board trace skews should be specified in your board trace model.

Table 3–13 shows the board skew parameters.

**Table 3–13. Board Skew Parameters**

| Parameter | Description |
|---|---|
| **Minimum delay difference between CK and DQS** | The minimum skew (or largest negative skew) between the CK signal and any DQS signal when arriving at the same DIMM over all DIMMs. This value affects the write leveling margin for DDR3 interfaces with leveling in multirank configurations. |
| **Maximum delay difference between CK and DQS** | The maximum skew (or largest positive skew) between the CK signal and any DQS signal when arriving at the same DIMM over all DIMMs. This value affects the Write Leveling margin for DDR3 interfaces with leveling in multi-rank configurations. |
| **Maximum skew within DQS group** | The largest skew between DQ and DM signals in a DQS group. This value affects the read capture and write margins for DDR2 and DDR3 SDRAM interfaces in all configurations (single or multiple chip-select, DIMM or component). |
| **Maximum skew between DQS groups** | The largest skew between DQS signals in different DQS groups. This value affects the resynchronization margin in memory interfaces wihtout leveling such as DDR2 SDRAM and discrete-device DDR3 SDRAM in both single- or multiple chip-select configurations. |
| **Maximum skew within address and command bus** | The largest skew between the address and command signals. |
| **Average delay difference between address and command and CK** | A value equal to the average of the longest and smallest address and command signal delay values, minus the delay of the CK signal. The value can be positive or negative. Positive values represent address and command signals that are longer than CK signals; negative values represent address and command signals that are shorter than CK signals. The Quartus II software uses this skew to optimize the delay of the address and command signals to have appropriate setup and hold margins for DDR2 and DDR3 SDRAM interfaces. |

# Controller Settings

Use this tab to apply the controller settings suitable for your design.

## Avalon-MM Interface

Table 3–14 shows the Avalon-MM interface parameters.

**Table 3–14. Avalon-MM Interface Parameters**

| Parameter | Description |
|---|---|
| **Generate power-of-2 bus widths** | Rounds down the Avalon-MM side data bus to the nearest power of 2. |
| **Maximum Avalon-MM burst length** | Specifies the maximum burst length on the Avalon-MM bus. Affects the `AVL_SIZE_WIDTH` parameter. |

## Low Power Mode

Table 3–15 shows low-power mode parameters.

**Table 3–15. Low-Power mode Parameters**

| Parameter | Description |
|---|---|
| **Enable self-refresh controls** | Enables the self-refresh signals on the controller top-level design. These controls allow you to control when the memory is placed into self-refresh mode. |
| **Enable auto-power down** | Allows the controller to automatically place the memory into power-down mode after a specified number of idle cycles. Specifies the number of idle cycles after which the controller powers down the memory in the auto-power down cycles parameter. |
| **Auto power-down cycles** | The number of idle controller clock cycles after which the controller automatically powers down the memory. The legal range is from 1 to 65,535 controller clock cycles. |

## Efficiency

Table 3–16 shows the efficiency parameters.

**Table 3–16. Efficiency Parameters (Part 1 of 2)**

| Parameter | Description |
|---|---|
| **Enable user auto-refresh controls** | Enables the user auto-refresh control signals on the controller top level. These controller signals allow you to control when the controller issues memory autorefresh commands. |
| **Enable auto-precharge control** | Enables the autoprecharge control on the controller top level. Asserting the autoprecharge control signal while requesting a read or write burst allows you to specify whether the controller should close (autoprecharge) the currently open page at the end of the read or write burst. |
| **Local-to-memory address mapping** | Allows you to control the mapping between the address bits on the Avalon-MM interface and the chip, row, bank, and column bits on the memory. |

**Table 3–16. Efficiency Parameters (Part 2 of 2)**

| Parameter | Description |
|---|---|
| **Command queue look-ahead depth** | Selects a look-ahead depth value to control how many read or writes requests the look-ahead bank management logic examines. Larger numbers are likely to increase the efficiency of the bank management, but at the cost of higher resource usage. Smaller values may be less efficient, but also use fewer resources. |
| **Reduce controller latency by (excluding PHY)** | Selects the number of controller latency in controller clock cycles for better latency or $f_{MAX}$. This option allows you to reduce the latency in the controller (and overall interface), but does not affect latency through the PHY. Reducing latency may also reduce $f_{MAX}$ performance. Lower latency controllers cannot run as fast as the default frequency. |

## Configuration, Status, and Error Handling

Table 3–17 shows the configuration, status, and error handling parameters.

**Table 3–17. Configuration, Status, and Error Handling Parameters**

| Parameter | Description |
|---|---|
| **Enable Configuration and Status Register Interface** | Enables run-time configuration and status interface for the memory controller. This option adds an additional Avalon-MM slave port to the memory controller top level, which you can use to change or read out the memory timing parameters, memory address sizes, mode register settings and controller status. If Error Detection and Correction Logic is enabled, the same slave port also allows you to control and retrieve the status of this logic. |
| **CSR port host interface** | Specifies the type of connection to the CSR port. The port can be exported, internally connected to a JTAG Avalon Master, or both. |
| **Enable error detection and correction logic** | Enables ECC for single-bit error correction and double-bit error detection. Your memory interface must be a multiple of 40 or 72 bits wide to use ECC. |
| **Enable auto error correction** | Allows the controller to perform auto correction when a single-bit error is detected by the ECC logic. |

## Advanced Features

Table 3–18 shows the advanced features parameters.

**Table 3–18. Advanced Features Parameters**

| Parameter | Description |
|---|---|
| **Enable multi-cast write control** | Enables the multi-cast write control on the controller top level. Multi-cast write is not supported if the ECC logic is enabled or for registered DIMM interfaces. |
| **Enable reduced bank tracking for area optimization** | Reduces the number of bank and timer blocks in the controller. Specifies the number of banks to track in **Number of banks to track**. |
| **Number of banks to track** | The number of banks to track when **Enable reduced bank tracking for area optimization** is turned on. The legal range is from 1 or the value of command queue look-ahead depth (whichever is greater) to 16. |

The wizard generates a Synopsis design constraint (**.sdc**) script. <*variation_name*>**.sdc**, and a pin assignment script, <*variation_name*>**_pin_assignments.tcl**. Both the **.sdc** and the <*variation name*>**_pin_assignments.tcl** support multiple instances. These scripts iterate through all instances of the core and apply the same constraints to all of them.

## Add Pin and DQ Group Assignments

The pin assignment script, <*variation_name*>**_pin_assignments.tcl**, sets up the I/O standards and the input/output termination for the DDR2 or DDR3 SDRAM controller with UniPHY. This script also helps to relate the `DQ` pin groups together for the Quartus II Fitter to place them correctly.

The pin assignment script does not create a clock for the design. You must create a clock for the design and provide pin assignments for the signals of both the example driver and testbench that the MegaCore variation generates.

Run the <*variation_name*>**_pin_assignments.tcl** to add the input and output termination, I/O standards, and DQ group assignments to the example design. To run the pin assignment script, follow these steps:

1. On the Processing menu, point to **Start**, and click **Start Analysis and Synthesis**.
2. On the Tools menu click **Tcl Scripts**.
3. Specify the **pin_assignments.tcl** file and click **Run**.

If the PLL input reference clock pin does not have the same I/O standard as the memory interface I/Os, a no-fit might occur because incompatible I/O standards cannot be placed in the same I/O bank.

## Compile the Design

To compile the design, on the Processing menu, click **Start Compilation**.

After you have compiled the top-level file, you can perform RTL simulation or program your targeted Altera device to verify the top-level file in hardware.

For more information about simulating, refer to the *Simulation* section in volume 4 of the *External Memory Interface Handbook*.

# 5. Functional Description—Controller

This chapter discusses the controller.

## Block Description

Figure 5–1 shows the top-level block diagram of the DDR2 or DDR3 SDRAM Controller with UniPHY

**Figure 5–1. DDR2 or DDR3 SDRAM Controller with UniPHY Block Diagram**

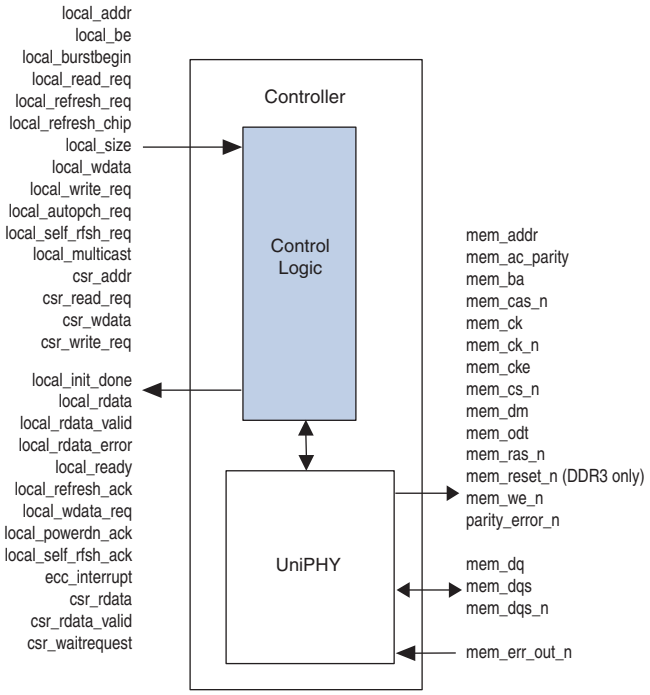
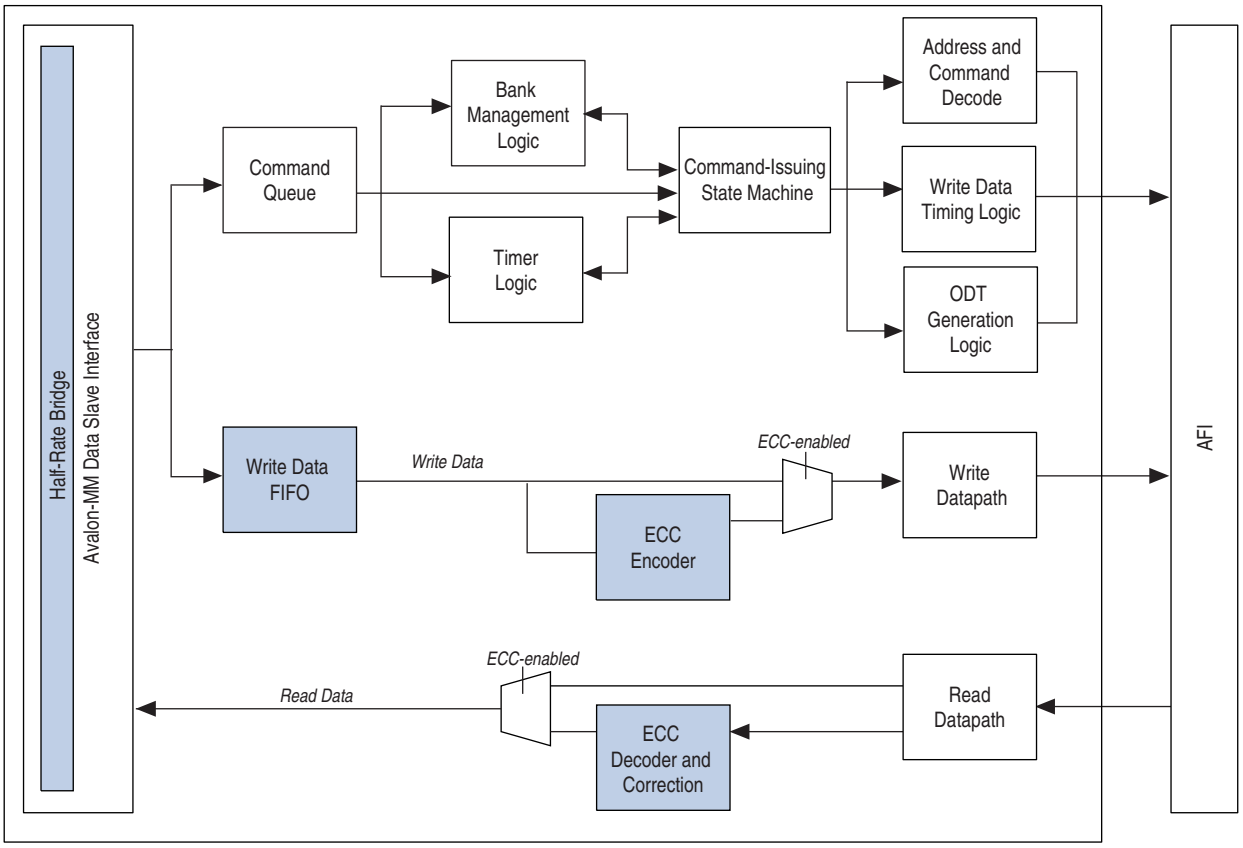


Figure 5–2 shows a block diagram of the DDR2 or DDR3 SDRAM Controller with UniPHY architecture.

**Figure 5–2. DDR2 or DDR3 SDRAM Controller with UniPHY Architecture Block Diagram**




The following sections describe the blocks in Figure 5–2 on page 5–2.

## Avalon-MM Data Slave Interface

The Avalon-MM data slave interface accepts read and write requests from the Avalon-MM master. The width of the data, `local_wdata` and `local_rdata`, is two or four times the width of the external memory, depending on whether you specify a full- or half-rate controller.

The IP core determines the local address width based on the memory chip, row, bank, and column address widths. For example:

- For multiple chip selects:

  width = chip bits + row bits + bank bits + column – *N*

- For single chip select:

  width = row bits + bank bits + column – *N*

Where *N* = 1 for full-rate controller and 2 for half-rate controller.

For every Avalon transaction, the number of read or write requests cannot exceed the maximum local burst count of 64. Altera recommends that you set this maximum burst count to match your system master's supported burst count.

## Write Data FIFO Buffer

The write data FIFO buffer holds the write data and byte-enable from the user logic until the the main state machine requests the data. The `local_ready` signal is deasserted when either the command queue or write data FIFO buffer is full. The write data FIFO buffer is wide enough to store the write data and the byte-enable signals.

## Command Queue

The command queue allows the controller to buffer up to eight consecutive reads or writes. The command queue presents the next 2, 4, 6, or 8 accesses to the internal logic for the look-ahead bank management. The bank management is more efficient if the look-ahead is deeper, but a deeper queue consumes more resources, and may cause maximum frequency degradation.

In addition to storing incoming commands, the command queue also maps the local address to memory address based on the address mapping option selected. By default, the command queue leverages the bank interleaving scheme, where the address increment goes to the next bank instead of the next row to increase chances of page hit.

## Bank Management Logic

The bank management logic keeps track of the current state in each bank across multiple chips. It can keep a row open in every bank in your memory system. When the state machine issues a command, the IP core updates the bank management logic with the latest bank status. The main state machine has look-ahead capability to issue early bank management commands. The controller has an autoprecharge feature to support an open page policy, where the last accessed row in each bank is kept open, and a closed page policy, where a bank is closed after it is used.

## Timer Logic

The timer logic models the internal behavior of each bank in the memory interface and provides status output signals to the state machine. The state machine then decides whether to issue the look-ahead bank management command based on the timer status signals.

## Command-Issuing State Machine

The command-issuing state machine decides what DDR commands to issue based on the inputs from the command queue, bank management logic, and timer logic. The command-issuing state machine operates in two modes: full-rate or half-rate. The full-rate state machine supports 1T address and command, and always issues memory burst length of 4. The half-rate state machine supports 2T address and command, and always issues memory burst length of 8.

A longer memory burst length, in this case 8 beats, increases the command bandwidth by allowing more data cycles for the same amount of command cycles. A longer memory burst length also provides more command cycles that ensures a more effective look-ahead bank management. However, longer memory burst lengths are less efficient if the bursts you issue do not provide enough data.

This state machine accepts any local burst count of 1 to 64. The built-in burst adapter in this state machine maps the local burst count to the most efficient memory burst. The state machine also supports reads and writes that start on non-aligned memory burst boundary addresses. For effective command bus bandwidth, this state machine supports additive latency which issues reads and writes immediately after the ACT command. This state machine accepts additive latency values greater or equal to $t_{RCD}$ – 1, in clock cycle unit ($t_{CK}$).

## Address and Command Decoding Logic

When the main state machine issues a command to the memory, it asserts a set of internal signals. The address and command decoding logic turns these signals into AFI-specific commands and address. This block generates the following signals:

- Clock enable and reset signals: `afi_cke`, `afi_rst_n`
- Command and address signals: `afi_cs_n`, `afi_ba`, `afi_addr`, `afi_ras_n`, `afi_cas_n`, `afi_we_n`

## Write and Read Datapath, and Write Data Timing Logic

The write and read datapath, and the write data timing logic generate the AFI read and write control signals.

When the state machine issues a write command to the memory, the IP core reads the write data for that write burst from the write data FIFO buffer. The relationship between the write command and write data depends on the `afi_wlat` signal. This logic presents the write data FIFO read request signal so that the data arrives on the external memory interface DQ pins at the correct time.

During write, the IP core generates the following AFI signals based on the state machine outputs and the `afi_wlat` signal:

- `afi_dqs_burst`
- `afi_wdata_valid`
- `afi_wdata`
- `afi_dm`

During read, the `afi_doing_read` signal generates the `afi_rdata_valid` signal and controls the PHY postamble circuit.

## ODT Generation Logic

The on-die termination (ODT) generation logic generates the necessary ODT signals for the controller, based on the scheme that Altera recmmends.

### DDR2 SDRAM

Table 5–1 shows which ODT signal is enabled for single-slot single chip-select per DIMM.

There is no ODT for reads.

**Table 5–1. ODT—DDR2 SDRAM Single Slot Single Chip-select Per DIMM (Write)**

| Write On | ODT Enabled |
|---|---|
| mem_cs[0] | mem_odt[0] |

Table 5–2 shows which ODT signal is enabled for single-slot dual chip-select per DIMM.

There is no ODT for reads.

**Table 5–2. ODT—DDR2 SDRAM Single Slot Dual Chip-select Per DIMM (Write)**

| Write On | ODT Enabled |
|---|---|
| mem_cs[0] | mem_odt[0] |
| mem_cs[1] | mem_odt[1] |

Table 5–3 shows which ODT signal is enabled for dual-slot single chip-select per DIMM.

**Table 5–3. ODT—DDR2 SDRAM Dual Slot Single Chip-select Per DIMM (Write)**

| Write On | ODT Enabled |
|---|---|
| mem_cs[0] | mem_odt[1] |
| mem_cs[1] | mem_odt[0] |

Table 5–4 shows which ODT signal is enabled for dual-slot dual chip-select per DIMM.

**Table 5–4. ODT—DDR2 SDRAM Dual Slot Dual Chip-select Per DIMM (Write)**

| Write On | ODT Enabled |
|---|---|
| mem_cs[0] | mem_odt[2] |
| mem_cs[1] | mem_odt[3] |
| mem_cs[2] | mem_odt[0] |
| mem_cs[3] | mem_odt[1] |

## DDR3 SDRAM

Table 5–5 shows which ODT signal is enabled for single-slot single chip-select per DIMM.

There is no ODT for reads.

**Table 5–5. ODT—DDR3 SDRAM Single Slot Single Chip-select Per DIMM (Write)**

| Write On | ODT Enabled |
|---|---|
| mem_cs[0] | mem_odt[0] |

Table 5–6 shows which ODT signal is enabled for single-slot dual chip-select per DIMM.

There is no ODT for reads.

**Table 5–6. ODT—DDR3 SDRAM Single Slot Dual Chip-select Per DIMM (Write)**

| Write On | ODT Enabled |
|---|---|
| mem_cs[0] | mem_odt[0] |
| mem_cs[1] | mem_odt[1] |

Table 5–7 shows which ODT signal is enabled for dual-slot single chip-select per DIMM.

**Table 5–7. ODT—DDR3 SDRAM Dual Slot Single Chip-select Per DIMM (Write)**

| Write On | ODT Enabled |
|---|---|
| mem_cs[0] | mem_odt[0] and mem_odt[1] |
| mem_cs[1] | mem_odt[0] and mem_odt[1] |

Table 5–8 shows which ODT signal is enabled for dual-slot single chip-select per DIMM.

**Table 5–8. ODT—DDR3 SDRAM Dual Slot Single Chip-select Per DIMM (Read)**

| Read On | ODT Enabled |
|---|---|
| mem_cs[0] | mem_odt[1] |
| mem_cs[1] | mem_odt[0] |

Table 5–9 shows which ODT signal is enabled for dual-slot dual chip-select per DIMM.

**Table 5–9. ODT—DDR3 SDRAM Dual Slot Dual Chip-select Per DIMM (Write)**

| Write On | ODT Enabled |
|---|---|
| mem_cs[0] | mem_odt[0] and mem_odt[2] |
| mem_cs[1] | mem_odt[1] and mem_odt[3] |
| mem_cs[2] | mem_odt[0] and mem_odt[2] |
| mem_cs[3] | mem_odt[1] and mem_odt[3] |

Table 5–10 shows which ODT signal is enabled for dual-slot dual chip-select per DIMM.

**Table 5–10. ODT—DDR3 SDRAM Dual Slot Dual Rank Per DIMM (Read)**

| Read On | ODT Enabled |
|---|---|
| mem_cs[0] | mem_odt[2] |
| mem_cs[1] | mem_odt[3] |
| mem_cs[2] | mem_odt[0] |
| mem_cs[3] | mem_odt[1] |

## User-Controlled Side-Band Signals

Side-band signals are Altera-specific signals that do not map directly to a standard interface. The user-controlled side-band signals consists of the following signals.

### User Autoprecharge Commands

The autoprecharge read and autoprecharge write commands allow you to indicate to the memory device that this read or write command is the last access to the currently open row. The memory device automatically closes or autoprecharges the page it is currently accessing so that the next access to the same bank is quicker.

This command is useful for applications that require fast random accesses. You can request an autoprecharge by asserting the `local_autopch` signal during a read or write request.

### User-Refresh Commands

The user-refresh command enables the request to place the memory into refresh. The user-refresh control takes precedence over a read or write request. You can issue up to nine consecutive refresh commands to the memory.

### Multicast Write

The multicast write request signal allows you to ask the controller to send the current write requests to all the chip-selects. The IP core writes the write data to all the ranks in the system. The multicast write feature is useful for $t_{RC}$ mitigation where you can cycle through chips to continuously read data without hitting $t_{RC}$. Multicast write is not available when using RDIMMs, or when you use the ECC feature.

### Low-Power Logic

There are two types of low-power logic: the user-controlled self-refresh logic and automatic power-down with programmable time-out logic.

#### User-Controlled Self-Refresh Logic

When you assert the `local_self_rfsh_req` signal, the controller completes any currently executing reads and writes, and then interrupts the command queue and immediately places the memory into self-refresh mode. When the controller places the memory into self-refresh mode, it responds by asserting the acknowledge signal, `local_self_rfsh_ack`. You can leave the memory in self-refresh mode for as long as you choose.

To bring the memory out of self-refresh mode, you must deassert the request signal, and the controller responds by deasserting the acknowledge signal when the memory is no longer in self-refresh mode.

### Automatic Power-Down with Programmable Time-Out

The controller automatically places the memory in power-down mode to save power if the requested number of idle controller clock cycles is observed in the controller. The **Auto Power Down Cycles** parameter on the **Controller Settings** tab allows you to specify a range between 1 to 65,535 idle controller clock cycles. The counter for the programmable time-out starts when there are no user read or write requests in the command queue. Once the controller places the memory in power-down mode, it responds by asserting the acknowledge signal, `local_powerdown_ack`.

## ECC

The optional ECC logic comprises an encoder and a decoder-corrector, which can detect and correct single-bit errors, and detect double-bit errors. The ECC logic is available in two widths: 64/72 bit and 32/40 bit. The ECC logic has the following features:

- Has Hamming code ECC logic that encodes every 64 or 32 bits of data into 72 or 40 bits of codeword.
- Has a latency increase of one clock for both writes and reads.
- Detects and corrects all single-bit errors.
- Detects all double-bit errors.
- Counts the number of single-bit and double-bit errors.
- Accepts partial writes, which trigger a read-modify-write cycle, for memory devices with DM pins.
- Can inject single-bit and double-bit errors to trigger ECC correction for testing and debugging purposes.
- Generates an interrupt signal when an error occurs.

When a single-bit or double-bit error occurs, the ECC logic triggers the `ecc_interrupt` signal to inform you that an ECC error has occurred. When a single-bit error occurs, the ECC logic reads the error address, and writes back the corrected data. When a double-bit error occurs, the ECC logic does not do any error correction but it asserts the `local_rdata_error` signal to indicate that the data is incorrect. The `local_rdata_error` signal follows the same timing as the `local_rdata_valid` signal.

Enabling autocorrection allows the ECC logic to delay all controller pending activities until the correction completes. You can disable autocorrection and schedule the correction manually when the controller is idle to ensure better system efficiency. To manually correct ECC errors, follow these steps:

1. When an interrupt occurs, read out the `SBE_ERROR` register. When a single-bit error occurs, the `SBE_ERROR` register is equal to one.
2. Read out the `ERR_ADDR` register.

3. Correct the single-bit error by issuing a dummy write to the memory address stored in the `ERR_ADDR` register. A dummy write is a write request with the `local_be` signal zero, that triggers a partial write which is effectively a read-modify-write event. The partial write corrects the data at that address and writes it back.

Table 5–15 through Table 5–18 show detailed information about the register maps.

## Partial Writes

The ECC logic supports partial writes. Along with the address, data, and burst signals, the Avalon-MM interface also supports a signal vector, `local_be`, that is responsible for byte-enable. Every bit of this signal vector represents a byte on the data-bus. Thus, a logic low on any of these bits instructs the controller not to write to that particular byte, resulting in a partial write. The ECC code is calculated on all bytes of the data-bus. If any bytes are changed, the IP core must recalculate the ECC code and write the new code back to the memory.

For partial writes, the ECC logic performs the following steps:

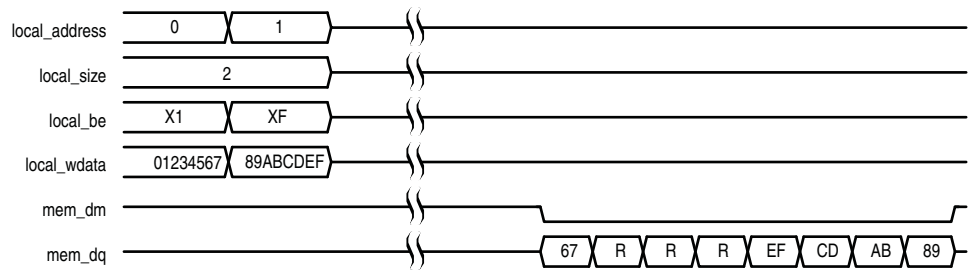
1. The ECC logic sends a read command to the partial write address.
2. Upon receiving a return data from the memory for the particular address, the ECC logic decodes the data, checks for errors, and then merges the corrected or correct dataword with the incoming information.
3. The ECC logic issues a write to write back the updated data and the new ECC code.

The following corner cases can occur:

- A single-bit error during the read phase of the read-modify-write process. In this case, the IP core corrects the single-bit error first, increments the single-bit error counter and then performs a partial write to this corrected decoded data word.
- A double-bit error during the read phase of the read-modify-write process. In this case, the IP core increments the double-bit error counter and issues an interrupt. The IP core writes a new write word to the location of the error. The ECC status register keeps track of the error information.

Figure 5–3 shows the partial write operation for the controller.

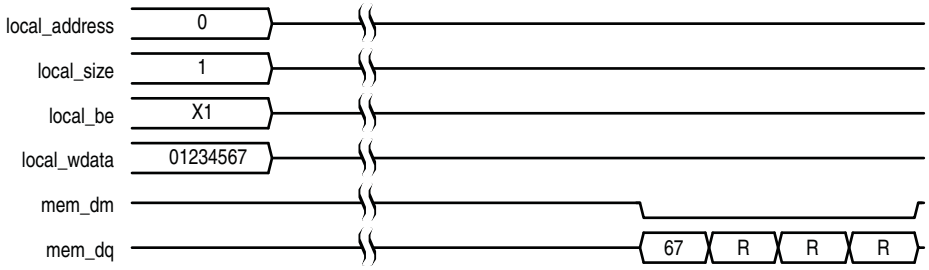
**Figure 5–3. Partial Write for the Controller—Full Rate**




**Note to Figure 5–3:**

(1) R represents the internal read-back memory data during the read-modify-write process.

**Figure 5–4. Partial Write for the Controller—Half Rate**




**Note to Figure 5–4:**

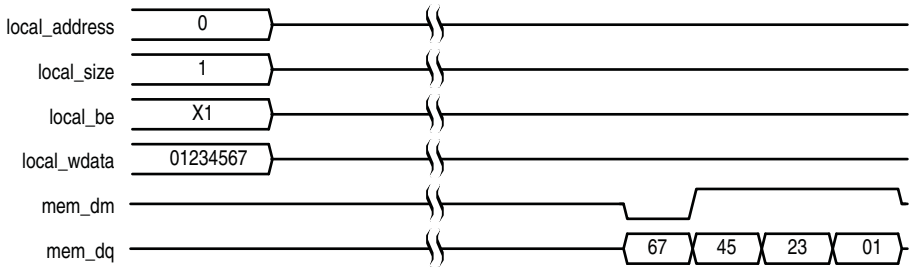
(1) R represents the internal read-back memory data during the read-modify-write process.

## Partial Bursts

DIMMs that do not have the DM pins do not support partial bursts. You must write a minimum of four (half rate) or eight words (full rate) to the memory at the same time.

Figure 5–5 shows the partial burst operation for controller.

**Figure 5–5. Partial Burst for Controller**



# Top-Level Signals Description

Table 5–11 shows the clock and reset signals.

The suffix _n denotes active low signals.

**Table 5–11. Clock and Reset Signals (Part 1 of 2)**

| Name | Direction | Description |
|---|---|---|
| global_reset_n | Input | The asynchronous reset input to the controller. The IP core derives all other reset signals from resynchronized versions of this signal. This signal holds the PHY, including the PLL, in reset while low. |
| pll_ref_clk | Input | The reference clock input to PLL. |
| phy_clk | Output | The system clock that the PHY provides to the user. All user inputs to and outputs from the controller must be synchronous to this clock. |
| reset_phy_clk_n | Output | The reset signal that the PHY provides to the user. The IP core asserts reset_phy_clk_n asynchronously and deasserts synchronously to phy_clk clock domain. |
| aux_full_rate_clk | Output | An alternative clock that the PHY provides to the user. This clock always runs at the same frequency as the external memory interface. In half-rate designs, this clock is twice the frequency of the phy_clk and you can use it whenever you require a 2x clock. In full-rate designs, the same PLL output as the phy_clk signal drives this clock. |
| aux_half_rate_clk | Output | An alternative clock that the PHY provides to the user. This clock always runs at half the frequency as the external memory interface. In full-rate designs, this clock is half the frequency of the phy_clk and you can use it, for example to clock the user side of a half-rate bridge. In half-rate designs, or if the **Enable Half Rate Bridge** option is turned on. The same PLL output that drives the phy_clk signal drives this clock. |
| dll_reference_clk | Output | Reference clock to feed to an externally instantiated DLL. |
| reset_request_n | Output | Reset request output that indicates when the PLL outputs are not locked. Use this signal as a reset request input to any system-level reset controller you may have. This signal is always low when the PLL is trying to lock, and so any reset logic using Altera advises you detect a reset request on a falling edge rather than by level detection. |
| soft_reset_n | Input | Edge detect reset input for SOPC Builder or for control by other system reset logic. Assert to cause a complete reset to the PHY, but not to the PLL that the PHY uses. |
| seriesterminationcontrol | Input (for OCT slave) | Required signal for PHY to provide series termination calibration value. Must be connected to a user-instantiated OCT control block (alt_oct) or another UniPHY instance that is set to OCT master mode. |
| | Output(for OCT master) | Unconnected PHY signal, available for sharing with another PHY. |
| parallelterminationcontrol | Input (for OCT slave) | Required signal for PHY to provide series termination calibration value. Must be connected to a user-instantiated OCT control block (alt_oct) or another UniPHY instance that is set to OCT master mode. |
| | Output (for OCT master) | Unconnected PHY signal, available for sharing with another PHY. |

**Table 5–11. Clock and Reset Signals (Part 2 of 2)**

| Name | Direction | Description |
|---|---|---|
| oct_rdn | Input (for OCT master) | Must connect to calibration resistor tied to GND on the appropriate RDN pin on the device. (Refer to appropriate device handbook.) |
| oct_rup | Input (for OCT master) | Must connect to calibration resistor tied to $V_{ccio}$ on the appropriate RUP pin on the device. (See appropriate device handbook.) |
| dqs_delay_ctrl_import | Input | Allows the use of DLL in another PHY instance in this PHY instance. Connect the export port on the PHY instance with a DLL to the import port on the other PHY instance. |

Table 5–12 on page 5–13 shows the controller local interface signals.

**Table 5–12. Local Interface Signals (Part 1 of 4)**

| Signal Name | Direction | Description |
|---|---|---|
| local_address[] | Input | Memory address at which the burst should start.<br>By default, the IP core maps local address to the bank interleaving scheme. You can change the ordering via the **Local-to-Memory Address Mapping** option in the **Controller Settings** page.<br>The IP core sizes the width of this bus according to the following equations:<br>■ Full rate controllers<br>For one chip select: width = row bits + bank bits + column bits – 1<br>For multiple chip selects: width = chip bits + row bits + bank bits + column bits – 1<br>If the bank address is 2 bits wide, row is 13 bits wide and column is 10 bits wide, the local address is 24 bits wide. To map local_address to bank, row and column address:<br>local_address is 24 bits wide<br>local_address[23:11]= row address[12:0]<br>local_address[10:9] = bank address [1:0]<br>local_address [8:0] = column address[9:1]<br>The IP core ignores the least significant bit (LSB) of the column address (multiples of four) on the memory side, because the local data width is twice that of the memory data bus width.<br>■ Half rate controllers<br>For one chip select: width = row bits + bank bits + column bits – 2<br>For multiple chip selects: width = chip bits + row bits + bank bits + column bits – 2<br>If the bank address is 2 bits wide, row is 13 bits wide and column is 10 bits wide, the local address is 23 bits wide. To map local_address to bank, row and column address:<br>local_address is 23 bits wide<br>local_address[22:10] = row address[12:0]<br>local_address[9:8] = bank address [1:0]<br>local_address [7:0] = column address[9:2]<br>The IP core ignores two LSBs of the column address on the memory side, because the local data width is four times that of the memory data bus width. |
| local_be[] | Input | Byte enable signal, which you use to mask off individual bytes during writes. local_be is active high; mem_dm is active low.<br>To map local_wdata and local_be to mem_dq and mem_dm, consider a full-rate design with 32-bit local_wdata and 16-bit mem_dq.<br>Local_wdata = < 22334455 >< 667788AA >< BBCCDDEE ><br>Local_be = < 1100 >< 0110 >< 1010 ><br>These values map to:<br>Mem_dq = <4455><2233><88AA><6677><DDEE><BBCC><br>Mem_dm = <1 1 ><0 0 ><0 1 ><1 0 ><0 1 ><0 1 > |

**Table 5–12. Local Interface Signals (Part 2 of 4)**

| Signal Name | Direction | Description |
|---|---|---|
| local_burstbegin | Input | The Avalon burst begin strobe, which indicates the beginning of an Avalon burst. Unlike all other Avalon-MM signals, the burst begin signal does not stay asserted if local_ready is deasserted.<br>For write transactions, assert this signal at the beginning of each burst transfer and keep this signal high for one cycle per burst transfer, even if the slave deasserts local_ready. The IP core samples this signal at the rising edge of phy_clk when local_write_req is asserted. After the slave deasserts the local_ready signal, the master keeps all the write request signals asserted until local_ready signal becomes high again.<br>For read transactions, assert this signal for one clock cycle when read request is asserted and local_address from which the data should be read is given to the memory. After the slave deasserts local_ready (waitrequest_n in Avalon interface), the master keeps all the read request signals asserted until local_ready becomes high again. |
| local_read_req | Input | Read request signal. You cannot assert read request and write request signals at the same time. The controller must deassert reset_phy_clk_n before you can assert local_autopch_req. |
| local_refresh_req | Input | User-controlled refresh request. If **Enable User Auto-Refresh Controls** option is turned on, local_refresh_req becomes available and you are responsible for issuing sufficient refresh requests to meet the memory requirements. This option allows complete control over when refreshes are issued to the memory including grouping together multiple refresh commands. Refresh requests take priority over read and write requests, unless the IP core is already processing the requests. |
| local_refresh_chip | Input | Controls which chip to issue the user refresh to. The IP core uses this active high signal with local_refresh_req. This signal is as wide as the memory chip select. This signal asserts a high value to each bit that represents the refresh for the corresponding memory chip.<br>For example: If local_refresh_chip signal is assigned with a value of 4'b0101, the controller refreshes the memory chips 0 and 2, and memory chips 1 and 3 are not refreshed. |
| local_size[] | Input | Controls the number of beats in the requested read or write access to memory, encoded as a binary number. The IP core supports Avalon burst lengths from 1 to 64. The IP core derives the width of this signal based on the burst count that you specify in the **Local Maximum Burst Count** option. With the derived width, you specify a value ranging from 1 to the local maximum burst count specified. |
| local_wdata[] | Input | Write data bus. The width of local_wdata is twice that of the memory data bus for a full-rate controller; four times the memory data bus for a half-rate controller. |
| local_write_req | Input | Write request signal. You cannot assert read request and write request signal at the same time. The controller must deassert reset_phy_clk_n before you can assert local_write_req. |
| local_multicast | Input | In-band multicast write request signal. The IP core uses this active high signal with the local_write_req signal. When this signal is asserted high, the IP core writes data to all the memory chips available. |

**Table 5–12. Local Interface Signals (Part 3 of 4)**

| Signal Name | Direction | Description |
|---|---|---|
| local_autopch_req | Input | User control of autoprecharge. If you turn on **Enable Auto-Precharge Control**, the local_autopch_req signal becomes available and you can request the controller to issue an autoprecharge write or autoprecharge read command. These commands cause the memory to issue a precharge command to the current bank at the appropriate time without an explicit precharge command from the controller. This feature is particularly useful if you know the current read or write is the last one you intend to issue to the currently open row. The next time you need to use that bank, the access could be quicker as the controller does not need to precharge the bank before activating the row you wish to access.<br>If you issue a local burst longer than the memory burst with local_autopch_req asserted, the controller only issues autoprecharge with the last read or write command. |
| local_self_rfsh_req | Input | User control of the self-refresh feature. If you turn on **Enable Self-Refresh Controls**, you can request that the controller place the memory devices into a self-refresh state by asserting this signal. The controller places the memory in the self-refresh state as soon as it can without violating the relevant timing parameters and responds by asserting local_self_rfsh_ack. You can hold the memory in the self-refresh state by keeping this signal asserted. You can release the memory from the self-refresh state at any time by deasserting local_self_rfsh_req and the controller responds by deasserting local__self_rfsh_ack when it has successfully brought the memory out of the self-refresh state. |
| local_init_done | Output | When the memory initialization, training, and calibration are complete, the PHY sequencer asserts ctrl_usr_mode_rdy to the memory controller, which then asserts this signal to indicate that the memory interface is ready for use.<br>The controller still accepts read and write requests before local_init_done is asserted, however it does not issue them to the memory until it is safe to do so.<br>This signal does not indicate that the calibration is successful. |
| local_rdata[] | Output | Read data bus. The width of local_rdata is twice that of the memory data bus for a full rate controller; four times the memory data bus for a half rate controller. |
| local_rdata_error | Output | Asserted if the current read data has an error. This signal is only available if you turn on **Enable Error Detection and Correction Logic**. The controller asserts this signal with the local_rdata_valid signal.<br>If the controller encounters double-bit errors, no correction is made and the controller asserts this signal. |
| local_rdata_valid | Output | Read data valid signal. The local_rdata_valid signal indicates that valid data is present on the read data bus. |
| local_ready | Output | The local_ready signal indicates that the controller is ready to accept request signals. If controller asserts the local_ready signal in the clock cycle that it asserts a read or write request, the controller accepts that request. The controller deasserts the local_ready signal to indicate that it cannot accept any more requests. The controller can buffer eight read or write requests. |

**Table 5–12. Local Interface Signals (Part 4 of 4)**

| Signal Name | Direction | Description |
|---|---|---|
| local_refresh_ack | Output | Refresh request acknowledge, which the controller asserts for one clock cycle every time it issues a refresh. Even if you do not turn on **Enable User Auto-Refresh Controls**, local_refresh_ack still indicates to the local interface that the controller has just issued a refresh command. |
| local_self_rfsh_ack | Output | Self refresh request acknowledge signal. The controller asserts and deasserts this signal in response to the local_self_rfsh_req signal. |
| local_power_down_ack | Output | Auto power-down acknowledge signal. The controller asserts this signal for one clock cycle every time auto power-down is issued. |
| ecc_interrupt | Output | Interrupt signal from the ECC logic. The controller asserts this signal when the ECC feature is turned on, and the controller detects an error. |

Table 5–13 shows the controller interface signals.

**Table 5–13. 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 DDR3 SDRAM and captures read data into the Altera device. |
| mem_dqs_n[] | Bidirectional | Inverted memory data strobe signal, which with the mem_dqs signal improves signal integrity. |
| mem_clk | Bidirectional | Clock for the memory device. |
| mem_clk_n | Bidirectional | Inverted clock for the memory device. |
| mem_addr[] | Output | Memory address bus. |
| mem_ac_parity *(1)* | Output | Address or command parity signal generated by the PHY and sent to the DIMM. DDR3 SDRAM only. |
| mem_ba[] | Output | Memory bank address bus. |
| mem_cas_n | Output | Memory column address strobe signal. |
| mem_cke[] | Output | Memory clock enable signals. |
| mem_cs_n[] | Output | Memory chip select signals. |
| mem_dm[] | Output | Memory data mask signal, which masks individual bytes during writes. |
| mem_odt | Output | Memory on-die termination control signal. |
| mem_ras_n | Output | Memory row address strobe signal. |
| mem_we_n | Output | Memory write enable signal. |
| parity_error_n *(1)* | Output | Active-low signal that is asserted when a parity error occurs and stays asserted until the PHY is reset. DDR3 SDRAM only |
| mem_err_out_n *(1)* | Input | Signal sent from the DIMM to the PHY to indicate that a parity error has occured for a particular cycle. DDR3 SDRAM only. |

**Notes to Table 5–13:**

(1) This signal is for registered DIMMs only.

Table 5–14 shows the CSR interface signals.

**Table 5–14. CSR Interface Signals**

| Signal Name | Direction | Description |
|---|---|---|
| csr_addr[] | Input | Register map address.The width of csr_addr is 16 bits. |
| csr_be[] | Input | Byte-enable signal, which you use to mask off individual bytes during writes. csr_be is active high. |
| csr_wdata[] | Input | Write data bus. The width of csr_wdata is 32 bits. |
| csr_write_req | Input | Write request signal. You cannot assert csr_write_req and csr_read_req signals at the same time. |
| csr_read_req | Input | Read request signal. You cannot assert csr_read_req and csr_write_req signals at the same time. |
| csr_rdata[] | Output | Read data bus. The width of csr_rdata is 32 bits. |
| csr_rdata_valid | Output | Read data valid signal. The csr_rdata_valid signal indicates that valid data is present on the read data bus. |
| csr_waitrequest | Output | The csr_waitrequest signal indicates that the HPC II is busy and not ready to accept request signals. If the csr_waitrequest signal goes high in the clock cycle when a read or write request is asserted, that request is not accepted. If the csr_waitrequest signal goes low, the HPC II is then ready to accept more requests. |

# Register Maps Description

The controller register map gives you control over the memory controller settings. To access the controller resister map, connect the CSR interface signals in Table 5–14 using the Avalon-MM protocol.

Table 5–15 through Table 5–18 show the register maps for the controller.

**Table 5–15. Register Map**

| Address | Contents |
|---|---|
| 0x130 | ECC control register |
| 0x131 | ECC status register |
| 0x132 | ECC error address register |

**Table 5–16. Address 0x130 ECC Control Register**

| Bit | Name | Default | Access | Description |
|---|---|---|---|---|
| 0 | ENABLE_ECC | 1 | RW | When 1, enables the generation and checking of ECC. |
| 1 | ENABLE_AUTO_CORR | — | RW | When 1, enables autocorrection when a single-bit error is detected. |
| 2 | GEN_SBE | 0 | RW | When 1, enables the deliberate insertion of single-bit errors, bit 0, in the data the IP core writes to memory. This bit is only for testing purposes. |
| 3 | GEN_DBE | 0 | RW | When 1, enables the deliberate insertion of double-bit errors, bits 0 and 1, in the data the IP core writes to memory. This bit is only for testing purposes. |
| 4 | ENABLE_INTR | 0 | RW | When 1, enables the interrupt output. |
| 5 | MASK_SBE_INTR | 0 | RW | When 1, masks the single-bit error interrupt. |
| 6 | MASK_DBE_INTR | 0 | RW | When 1, masks the double-bit error interrupt |
| 7 | CLEAR | 0 | RW | When 1, writing to this self-clearing bit clears the interrupt signal, and the error status and error address registers. It will also clear the SBE_ECC and SBE_DCC counters. |
| 9 | Reserved | 0 | — | Reserved for future use. |

**Table 5–17. Address 0x131 ECC Status Register**

| Bit | Name | Default | Access | Description |
|---|---|---|---|---|
| 0 | SBE_ERROR | 1 | RO | Set to 1 when any single-bit errors occur. |
| 1 | DBE_ERROR | 1 | RO | Set to 1 when any double-bit errors occur. |
| 2-7 | Reserved | 0 | — | Reserved for future use. |
| 8-15 | SBE_COUNT | 0 | RO | Reports the number of single-bit errors that have occurred since the status register counters were last cleared. |
| 16-23 | DBE_COUNT | 0 | RO | Reports the number of double-bit errors that have occurred since the status register counters were last cleared. |
| 24-31 | Reserved | 0 | — | Reserved for future use. |

**Table 5–18. Address 0x132 ECC Error Address Register**

| Bit | Name | Default | Access | Description |
|---|---|---|---|---|
| 0-31 | ERR_ADDR | 0 | RO | The address of the most recent ECC error. This address contains concatenation of chip, bank, row, and column addresses. |

This chapter describes the PHY part of the DDR2 and DDR3 SDRAM controllers with UniPHY.

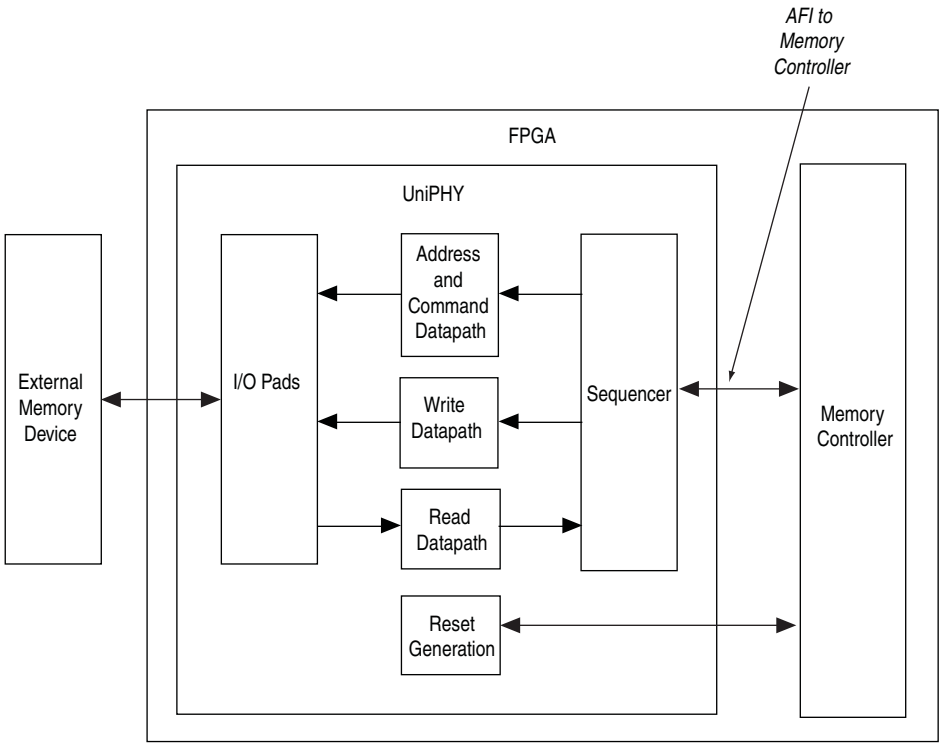
## Block Description

The PHY comprises the following major functional units:

- Reset and Clock Generation
- Address and Command Datapath
- Write Datapath
- Read Datapath
- Sequencer

Figure 6–1 shows the PHY block diagram.

**Figure 6–1. PHY Block Diagram**




## I/O Pads

The I/O pads contain all the I/O instantiations. The bulk of the UniPHY I/O circuitry is encapsulated in the ALTDQ_DQS megafunction for IV-series device families and earlier, and in the ALTDQ_DQS2 megafunction for Stratix V devices.

## Reset and Clock Generation

The clocking operation in the PHY can be classified into two domains: the PHY-memory domain and the PHY-AFI domain. The PHY-memory domain interfaces with the external memory device and is always at full-rate. The PHY-AFI domain interfaces with the memory controller and can be either a full-rate or half-rate clock based on the choice of the controller. Table 6–1 lists the clocks required for half-rate designs.

**Table 6–1. Clocks—Half-Rate Designs**

| Clock | Source | Clock Rate | Phase | Clock Network Type | Description |
|---|---|---|---|---|---|
| pll_afi_clk | PLL: C0 | Half | 0° | Unconstrained | Clock for AFI logic. |
| pll_mem_clk | PLL: C1 | Full | 0° *(1)*<br>-45° *(2)* | Dual-regional *(4)* | Output clock to memory. |
| pll_write_clk | PLL: C2 | Full | 90° *(1)*<br>-135° *(2)*<br>45° *(3)* | Dual-regional *(4)* | Clock for the write data out to memory (data is center aligned with the delayed pll_write_clk). |
| pll_addr_cmd_clk | PLL: C3 | Half | Set in wizard (default 270°) | Dual-regional | Clock for the address and command out to memory (address and command is center aligned with memory clock). |
| pll_afi_half_clk | PLL:C4 | Quarter | 0° | Unconstrained | AFI half clock. Used only when the half-rate bridge option is enabled. |
| pll_avl_clk | PLL:C5 | — | 0° | Dual-regional | NIOS sequencer clock. |
| pll_config_clk | PLL:C6 | — | 0° | Dual-regional | Scan chain clock. |
| DQS | Memory | Full | 90° | Local | The read data strobe. |

**Notes for Table 6–1:**

(1) For memory frequencies >240 MHz
(2) For memory frequencies <=240 MHz
(3) For memory frequencies >=240 MHz, for Stratix V devices only.
(4) For parameterizations with interface width >36, pll_mem_clk and pll_write_clk are assigned to use the global network.

Table 6–2 lists the clocks required for full-rate designs.

**Table 6–2. Clocks—Full-Rate Designs**

| Clock | Source | Clock Rate | Phase | Clock Network Type | Description |
|---|---|---|---|---|---|
| pll_afi_clk | PLL: C0 | Full | 0° | Unconstrained | Clock for AFI logic. |
| pll_mem_clk | PLL: C1 | Full | 90° *(1)*<br>0° *(2)* | Dual-regional *(3)* | Output clock to memory. |
| pll_write_clk | PLL: C2 | Full | 180° *(1)*<br>-90° *(2)* | Dual-regional *(3)* | Clock for write data out to memory (data is center aligned with the delayed pll_write_clk). |
| pll_addr_cmd_clk | PLL: C3 | Full | Set in wizard (default 225°) | Dual-regional | Clock for the address and command out to memory. 180° gives adress and command center aligned with memory clock; 225° produces best overall timing results. |

**Table 6–2. Clocks—Full-Rate Designs**

| Clock | Source | Clock Rate | Phase | Clock Network Type | Description |
|---|---|---|---|---|---|
| pll_afi_half_clk | PLL:C4 | Quarter | 0° | Unconstrained | AFI half clock. Used only when the half-rate bridge option is enabled. |
| pll_avl_clk | PLL:C5 | — | 0° | Dual-regional | NIOS sequencer clock. |
| pll_config_clk | PLL:C6 | — | 0° | Dual-regional | Scan chain clock. |
| DQS | Memory | Full | 90° | Local | The read data strobe. |

**Notes for Table 6–2:**

(1) For memory frequencies >240 MHz

(2) For memory frequencies <=240 MHz

(3) For parameterizations with interface width >36, pll_mem_clk and pll_write_clk are assigned to use the global network.

The UniPHY uses an active-low, asynchronous assert and synchronous de-assert reset scheme. The global reset signal resets the PLL in the PHY and the rest of the system is held in reset until after the PLL is locked. The number of synchronization pipeline stages is set to 4.

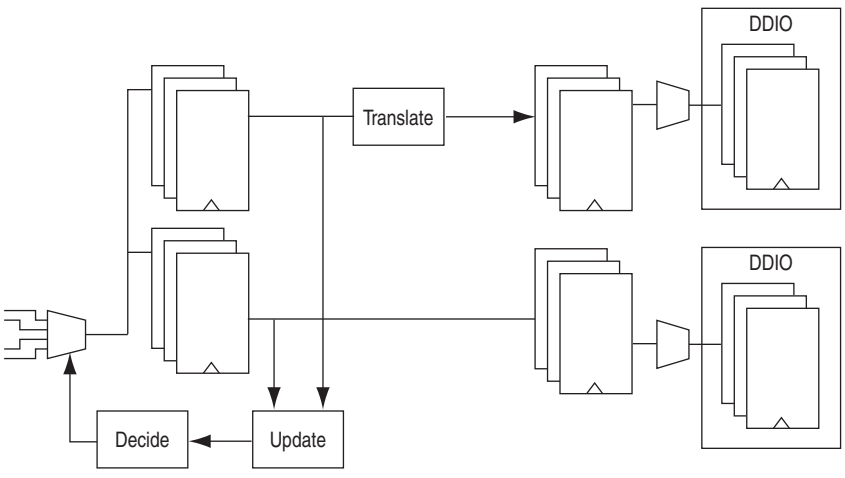
## Address and Command Datapath

The memory controller controls the read and write addresses and commands to meet the memory specifications. The PHY simply passes the address and command received from the memory controller to the memory device. The PHY is indifferent to address or command, the circuitry is the same for both.

The address and command datapath outputs connect to the inputs of the address and command I/Os . An ALTDDIO_OUT megafunction converts the addresses from SDR to DDR. An ALTDDIO_OUT megafunction with an ALTIOBUF megafunction delivers a pair of address and command clock to the memory.

Figure 6–2 shows the address and command datapath.

**Figure 6–2. Address and Command Datapath**



## Write Datapath

The write datapath passes write data from the memory controller to the I/O. The DQ pins are bidirectional and shared between read and write. The write data valid signal from the memory controller generates the output enable signal to control the output buffer. It also generates the dynamic termination control signal, which selects between series (output mode) and parallel (input mode) termination. An ALT_OCT megafunction (instantiated automatically in the top-level file) configures the termination values.

### Leveling Circuitry

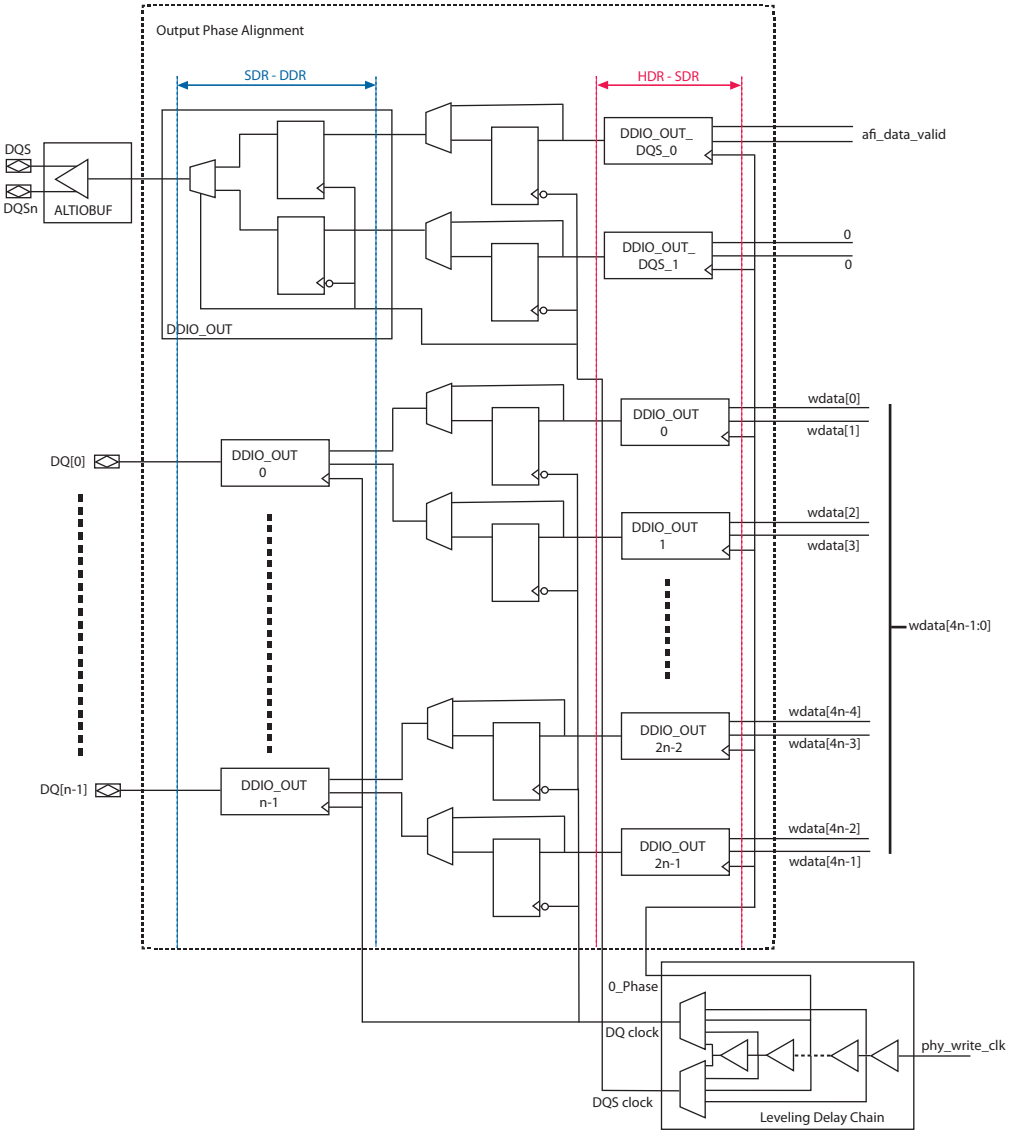
For DDR2, leveling circuitry is invoked automatically for frequencies above 240 MHz; no leveling is used for frequencies below 240 MHz. For DDR3, leveling is always invoked.

For frequencies below 240 MHz, you should use a tree-style layout. For frequencies above 240 MHz, you can choose either a leveled or tree-style layout, as the leveled PHY will calibrate correctly to either implementation. For DDR3 implementations at higher frequencies, a fly-by topology is recommended for best signal integrity.

For details about leveling delay chains, consult the memory interfaces hardware section of the device handbook for your FPGA.

Figure 6–3 shows the write datapath for a leveling interface. The full-rate PLL output clock `phy_write_clk` goes to a leveling delay chain block which generates all other periphery clocks that are needed. The data signals that generate DQ and DQS signals pass to an output phase alignment block. The output phase alignment block feeds an ALTIOBUF buffer which creates a pair of pseudo differential clocks that connect to the memory. In full-rate designs, the HDR-SDR portion of the output phase alignment is bypassed. The <*variation_name*>**_pin_assignments.tcl** script automatically specifies the logic option that associates all DQ pins to the DQS pin. The Quartus II Fitter treats the pins as a DQS/DQ pin group.

**Figure 6–3. Write Datapath for a Leveling Interface**



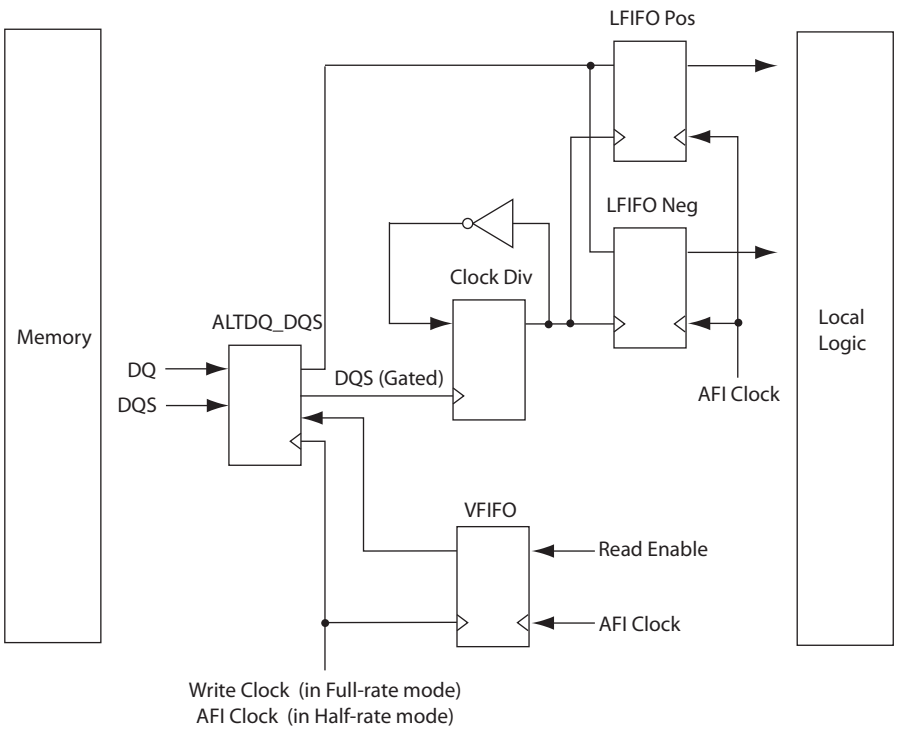

## Read Datapath

The read data is captured in the input mode ALTDQ_DQS in the I/O. The captured data is then forwarded to the read datapath. The read datapath synchronizes read data from the read capture clock domain to the AFI clock domain and converts data from SDR to HDR (half-rate designs only).

In half-rate designs, the write side of the FIFO buffer should be half the width and double the depth of the read side of the FIFO buffer. The read side only reads one entry after the IP core writes the write side into two entries, which effectively converts data from SDR to HDR. In full-rate designs, the size of the FIFO buffer is the same for both write and read as both sides operate at the same rate. For half-rate designs, the FIFO operates at half-rate on both read and write sides, and contains 4 half-rate entries; for full-rate designs, the FIFO operates at full-rate on both read and write sides, and contains 8 full-rate entries.

Figure 6–4 shows the read datapath.

**Figure 6–4. Read Datapath**

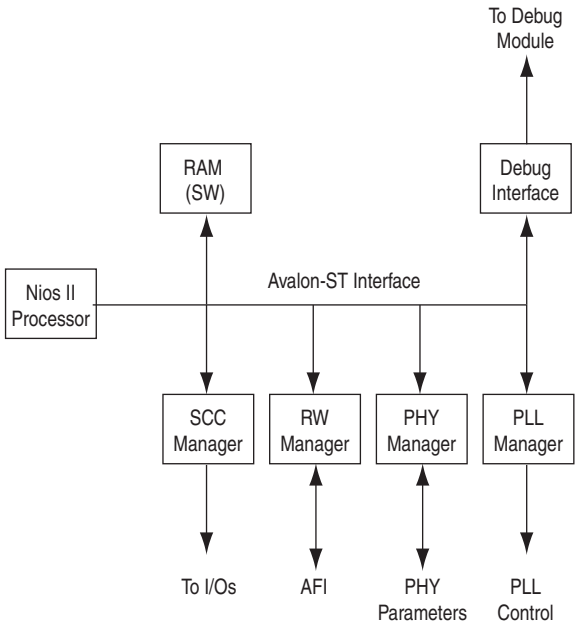



## Sequencer

The sequencer intercepts all the signals from the memory controller and takes control during calibration. The sequencer includes a Nios® II processor that processes the calibration algorithm.

Figure 6–5 shows the sequencer block diagram. The calibration algorithm is compiled into Nios II code that resides in the FPGA RAM blocks and may use a portion of this RAM as scratch space. The debug interface provides a mechanism for interacting with the various managers and for tracking the progress of the calibration algorithm. Although you can use the interface as a debugging tool during the initial development of the sequencer, you may use it to interact with the memory interface and debug problems that arise within the PHY. The managers are specified in RTL and they encapsulate a timing or protocol specific aspect of calibration.

**Figure 6–5. Sequencer Block Diagram**




Calibration configures the PHY and the I/Os that make up the memory interface so that you can reliably transfer data to and from memory. The calibration applies settings to the FIFO buffers within the PHY to minimize latency and ensures that the read valid signal is generated at the appropriate time. The calibration also applies settings to the I/O delay chains and phases so that data transfers to and from memory are centered with respect to the data clock.

## SCC Manager

The scan chain control (SCC) manager allows you to set various delays and phases on the I/Os that make up the memory interface. The latest Altera device families provide dynamic delay chains on input, output, and output enable paths which you can reconfigure at runtime. The SCC manager allows you to access these chains to add delay on incoming and outgoing signals. A master on the Avalon-ST interface may require the maximum allowed delay setting on input and output paths, and may set a particular delay value in this range to apply to the paths.

The SCC manager implements the Avalon-ST interface and the storage mechanism for all input, output, and phase settings. It contains circuitry that configures a DQ- or DQS-configuration block. The Nios II processor may set delay, phases, or register settings. The sequencer scans them serially to the appropriate DQ or DQS configuration block.

## RW Manager

The read write (RW) manager encapsulates the protocol to read and write to the memory device through the AFI. It provides a buffer that stores the data to be sent to and read from memory, and provides the following commands:

- Write configuration—configures the memory for use. Sets up burst lengths, read and write latencies, and other device specific parameters.

- Refresh—initiates a refresh operation at the DRAM. The command does not exist on SRAM devices. The sequencer also provides a register that determines whether the RW manager automatically generates refresh signals.
- Enable or disable multi-purpose register (MPR)—some memory devices provide a special register that contains calibration specific patterns that you can read. This command enables or disables access to this register.
- Activate row—for memory devices that have both rows and columns, this command activates a specific row. Subsequent reads and writes operate on this specific row.
- Precharge—closes a row before you can access a new row.
- Write or read burst—writes or reads a burst length of data.
- Write guaranteed—write with a special mode where the memory holds address and data lines constant. Altera guarantees this type of write to work in the presence of skew, but constrains to write the same data across the entire burst length.
- Write and read back-to-back—performs back-to-back writes or reads to adjacent banks. Most memory devices have strict timing constraints on subsequent accesses to the same bank, thus back-to-back writes and reads have to reference different banks.
- Protocol-specific initialization—which the initialization sequence requires.

## PHY Manager

The PHY manager provides access to the PHY for calibration. In addition, the PHY manager sends information from the calibration algorithm that is relevant to the PHY. Specifically, there are two FIFO buffers inside the PHY: VFIFO and LFIFO. The calibration sets both FIFO buffers parameters during calibration. VFIFO buffer determines when data from the memory is ready to be sampled (`read_valid` prediction), and the LFIFO buffer stores the data sampled from memory. There are settings available on the VFIFO buffer that determine exactly when the read valid signal is generated and there are settings on the LFIFO buffer that determine the read latency. In addition to the ability to control the PHY's FIFO buffers, the PHY manager provides a signal to the PHY to indicate when the memory initialization sequence finishes. Certain signals from the memory are not stable until initialization is complete, thus it is useful to inform the PHY when initialization occurs.

## PLL Manager

The PLL manager provides access to the PHY's PLL. You can set the following two parameters of the PLL:

- The phase of a specific clock output. You can specify one the eight available taps. A phase change does not cause the PLL to lose the clock.
- The frequency at which a single output or the internal VCO run. You can modify the C counter or the M and N counters respectively. Such changes cause the PLL to lose the lock and forces the system to reset, which limits this operation to debugging only.

### Nios II Processor

The Nios II processor manages the calibration algorithm; the NIOS II processor is unavailable once calibration is completed.

The same calibration algorithm supports all device families, with some differences. The following sections describe the calibration algorithm for DDR3 SDRAM on Stratix III devices. Calibration algorithms for other protocols and families are a subset and significant differences are pointed out when necessary. As the algorithm is fully contained in the software of the sequencer (in the C code) enabling and disabling specific steps involves turning flags on and off.

Calibration comprises the following stages:

- Initialize memory
- Calibrate read datapath
- Calibrate write datapath
- Run diagnostics

#### Initialize Memory

The calibration must initialize all memory devices before they can properly operate. As the sequencer takes control of the PHY at startup, the sequencer performs this initialization stage.

#### Calibrate Read Datapath

Calibrating the read datapath comprises the following steps:

- Calibrate DQS enable cycle and phase
- Perform read per-bit deskew to center within data valid window
- Reduce LFIFO latency

#### Calibrate Write Datapath

Calibrating the write datapath involves the following steps:

- Center align DQS with respect to DQ.
- Align DQS with `mem_clk`.

#### Test Diagnostics

The sequencer estimates the read and write margins under noisy conditions, by sweeping input and output DQ and DQS delays to determine the size of the data valid windows on the input and output side. It stores this information in the local memory and you can access it through the debugging interface.

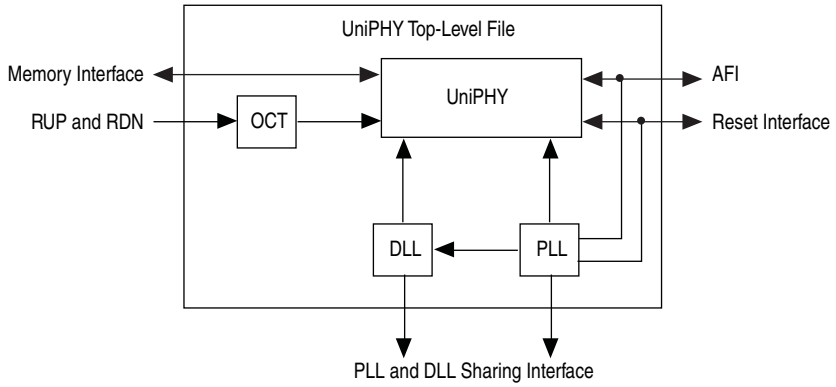
When the diagnostic test completes, the control of the PHY interface passes back to the controller and the sequencer issues a a pass or fail signal.

## Interfaces

Figure 6–6 shows the major blocks of the UniPHY and how it interfaces with the external memory device and the controller.

Instantiating the delay-locked loop (DLL) and the phase-locked loop (PLL) on the same level as the UniPHY eases DLL and PLL sharing.

**Figure 6–6. UniPHY Interfaces with the Controller and the External Memory**




The following interfaces are on the UniPHY top-level file:

- AFI
- Memory interface
- DLL and PLL sharing interface
- OCT interface

## AFI

The UniPHY datapath uses the Altera PHY interface (AFI). The AFI is in a simple connection between the PHY and controller; the AFI is based on a Denali model, with some calibration-related signals not used and some additional Altera-specific sideband signals added.

## The Memory Interface

For more information on the memory interface, refer to "UniPHY Signals" on page 6–13.

## The DLL and PLL Sharing Interface

You can generate the UniPHY memory interface as a master or as a slave, depending on the setting of **Master for PLL/DLL sharing** on the **PHY Settings** tab of the parameter editor. The top-level file of a generated UniPHY variant contains both a PLL and a DLL. The PLL produces a variety of required clock signals derived from the reference clock; the DLL produces a delay codeword. When you instantiate master and slave variants into your HDL code, you must connect the PLL outputs from the master to the clock inputs of the slaves.

The master **.qip** file must appear before the slave **.qip** file in the Quartus II settings **(.qsf)** file.

The UniPHY memory interface requires one PLL and one DLL to produce the clocks and delay codeword. The PLL and DLL can be shared using a master and slave scenario. The top-level file defines the PLL and DLL output signals as inputs and outputs and an additional parameter `PLL_DLL_MASTER` is also defined. If `PLL_DLL_MASTER` is 1, the RTL instantiates the PLL and DLL, which drives the clock and DLL codeword inputs and outputs. If the parameter is 0, the wires previously connected to the output of the PLL and DLL are assigned to the clock and DLL codeword input and outputs. Inputs and outputs are specified based on the setting of the **PLL/DLL sharing** option.

If you generate a slave IP core, you must modify the timing scripts to allow the timing analysis to correctly resolve clock names and analyze the IP core. Otherwise the software issues critical warnings and an incorrect timing report.

To modify the timing script, follow these steps:

1. In a text editor, open the <*IP core name*>**/constraints directory/**<*IP core name*>**_timing.tcl** file.
2. Search for the following 2 lines:
   - `set master_corename "_MASTER_CORE_"`
   - `set master_instname "_MASTER_INST_"`
3. Replace `_MASTER_CORE_` with the core name and `_MASTER_INST_` with instance name of the UniPHY master to which the slave is connected.

   The instance name is the full path to the instance and is in the <*IP core name*>**_all_pins.txt** file that is automatically generated after the <*IP core name*>**_pin_assignments.tcl** script runs.

4. If the slave component is connected to a user-defined PLL rather than a UniPHY master, you must manually enter all clock names.
   - Remove the `master_corename` and `master_instname` variables with the checks performed in the eight lines following them.
   - You can use all clock name assignments as templates. For example set `local_pll_afi_clk "mycomponent|mypll|my_afi_clk"`.

   CAUTION: You must be extremely careful when connecting clock signals to the slave. Connecting to clocks with frequency or phase different than what the core expects may result in hardware failures.

The DLL and PLL Sharing interface is not available with SOPC Builder.

## The OCT Sharing Interface

By default, the UniPHY IP generates the required OCT control block at the top-level RTL file for the PHY. If you want, you can instantiate this block elsewhere in your code and feed the required termination control signals into the IP core by turning off **Master for OCT Control Block** on the **PHY Settings** tab. If you turn off **Master for OCT Control Block**, you must instantiate the OCT control block or use another UniPHY instance as a master, and ensure that the parallel and series termination control bus signals are connected to the PHY.

Figure 6–7 and Figure 6–8, respectively, show the PHY architecture with and without Master for OCT Control Block

**Figure 6–7. PHY Architecture with Master for OCT Control Block**

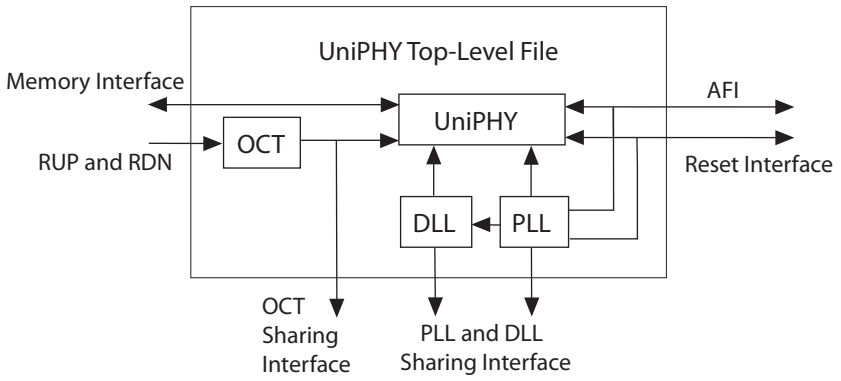
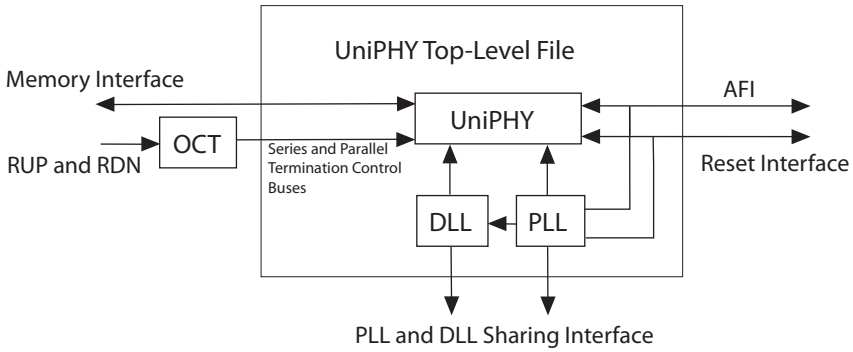



**Figure 6–8. PHY Architecture without Master for OCT Control Block**




☞ The OCT Sharing Interface and OCT slave mode are not available with SOPC Builder.

# UniPHY Signals

This section describes the UniPHY signals. Table 6–3 shows the clock and reset signals.

**Table 6–3. Clock and Reset Signals**

| Name | Direction | Width | Description |
|---|---|---|---|
| pll_ref_clk | input | 1 | PLL reference clock input. |
| global_reset_n | input | 1 | Active low global reset for PLL and all logic in the PHY, which causes a complete reset of the whole system. |
| soft_reset_n | input | 1 | Holding soft_reset_n low holds the PHY in a reset state. However it does not reset the PLL, which keeps running. It also holds the afi_reset_n output low. Mainly for use by SOPC Builder. |
| reset_request_n | output | 1 | When the PLL is locked, reset_request_n is high. When the PLL is out of lock, reset_request_n is low. |

Table 6–4 shows the AFI signals.

**Table 6–4. AFI Signals (Part 1 of 2)**

| Name | Direction | Width | Description |
|---|---|---|---|
| **Clocks and Reset** | | | |
| afi_clk | output | 1 | Half-rate or full-rate clock supplied to controller and system logic. |
| afi_half_clk | output | 1 | Half-rate clock signal. |
| afi_reset_n | output | 1 | Reset output on afi_clk clock domain. For use as asynchronous reset. This signal is asynchronously asserted and synchronously de-asserted. |
| **Address and Command** | | | |
| afi_addr | input | MEM_ADDRESS_WIDTH × AFI_RATIO | Row address. |
| afi_wps_n | output | MEM_CONTROL_WIDTH | Write enable. |
| afi_rps_n | output | MEM_CONTROL_WIDTH | Read enable. |
| **Write Data** | | | |
| afi_dm | input | MEM_DM_WIDTH × AFI_RATIO | Data mask input that generates mem_dm. |
| afi_wdata | input | MEM_DQ_WIDTH × 2 × AFI_RATIO | Write data input that generates mem_dq. |
| afi_wdata_valid | input | MEM_WRITE_DQS_ WIDTH × AFI_RATIO | Write data valid that generates mem_dq and mem_dm output enables. |
| **Read Data** | | | |
| afi_rdata | output | MEM_DQ_WIDTH × 2 × AFI_RATIO | Read data |

**Table 6–4. AFI Signals (Part 2 of 2)**

| Name | Direction | Width | Description |
|---|---|---|---|
| afi_rdata_en | input | MEM_READ_DQS_WIDTH × AFI_RATIO | Doing read input. Indicates that the memory controller is currently performing a read operation. |
| afi_rdata_en_full | input | MEM_READ_DQS_WIDTH × AFI_RATIO | Doing read input for full-rate controllers. Indicates that the memory controller is currently performing a read operation. |
| afi_rdata_valid | output | AFI_RATIO | Read data valid indicating valid read data on afi_rdata, in the byte lanes and alignments that were indicated on afi_rdata_en. |
| **Calibration Control and Status** | | | |
| afi_cal_success | output | 1 | '1' signals that calibration has completed |
| afi_cal_fail | output | 1 | '1' signals that calibration has failed |

Table 6–5 shows the DDR2 and DDR3 SDRAM interface signals.

**Table 6–5. DDR2 and DDR3 SDRAM Interface Signals**

| Name | Direction | Width | Description |
|---|---|---|---|
| mem_address | output | MEM_ADDRESS_WIDTH | Address. |
| mem_bws_n | output | MEM_DM_WIDTH | Data mask. |
| mem_wps_n | output | MEM_CONTROL_WIDTH | Write enable. |
| mem_rps_n | output | MEM_CONTROL_WIDTH | Read enable. |
| mem_doff_n | output | MEM_CONTROL_WIDTH | Connecting this pin to ground turns off the DLL inside the device. |
| mem_k, mem_kn | output | MEM_WRITE_DQS_WIDTH | Write clock(s) to memory, 1 clock per DQS group. |
| mem_cq, mem_cq_n | input | MEM_READ_DQS_WIDTH | Read clock(s) from memory, 1 clock per DQS group |
| mem_d | input | MEM_DQ_WIDTH | Input data bus. |
| mem_q | output | MEM_DQ_WIDTH | Output data bus. |

Table 6–6 shows the top-level signals generated for HardCopy® migration.

**Table 6–6. Top-Level HardCopy Migration Signals (Part 1 of 2)**

| Name | Direction | Description |
|---|---|---|
| **altsyncram Signals** | | |
| hc_rom_config_clock | Input | Write clock for the ROM loader. This clock is for the write clock of the Nios II code memory. |
| hc_rom_config_datain | Input | Data input from external ROM. |
| hc_rom_config_rom_data_ready | Input | Asserts to the code memory loader that the word memory is ready to be loaded. |

**Table 6–6. Top-Level HardCopy Migration Signals (Part 2 of 2)**

| Name | Direction | Description |
|---|---|---|
| hc_rom_config_init | Input | When asserted, it signals that the Nios II code memory is being loaded from the external ROM. |
| hc_rom_config_rom_rden | Output | Read-enable signal that connects to the external ROM. |
| hc_rom_config_rom_address | Output | ROM address that connects to the external ROM. |
| **DLL Reconfiguration Signals** | | |
| hc_dll_config_dll_offset_ctrl_addnsub | Input | Addition and subtraction control port for the DLL. This port controls if the delay-offset setting on hc_dll_config_dll_offset_ctrl_offset is added or subtracted. |
| hc_dll_config_offset_ctrl_offset | Input | Offset input setting for the DLL. This setting is a Gray-coded offset that is added or subtracted from the current value of the DLL's delay chain. |
| hc_dll_config_dll_offset_ctrl_offsetctrlout | Output | The registered and Gray-coded value of the current delay-offset setting. |
| **PLL Reconfiguration Signals** | | |
| hc_pll_config_configupdate | Input | Control signal to enable PLL reconfiguration. |
| hc_pll_config_phasecounterselect | Input | Specifies the counter select for dynamic phase adjustment. |
| hc_pll_config_phasestep | Input | Specfies the phase step for dynamic phase shifting. |
| hc_pll_config_phaseupdown | Input | Specifies whether the phase shift is up or down. |
| hc_pll_config_scanclk | Input | PLL reconfiguration scan chain clock. |
| hc_pll_config_scanclkena | Input | Clock enable port of the hc_pll_config_scanclk clock. |
| hc_pll_config_scandata | Input | Serial input data for the PLL reconfiguration scan chain. |
| hc_pll_config_scandataout | Output | Data output of the serial scanchain. |
| hc_pll_config_scandone | Output | This signal is asserted when the scan chain write operation is in progress. This signal is deasserted when the write operation is complete. |

Table 6–6 shows the top-level signals generated for HardCopy migration.

**Table 6–7. Top-Level HardCopy Migration Signals (Part 1 of 2)**

| Name | Direction | Description |
|---|---|---|
| **altsyncram Signals** | | |
| hc_rom_config_clock | Input | Write clock for the ROM loader. This clock is used as the write clock of the Nios II code memory. |
| hc_rom_config_datain | Input | Data input from external ROM. |
| hc_rom_config_rom_data_ready | Input | Asserts to the code memory loader that the word memory is ready to be loaded. |
| hc_rom_config_init | Input | When asserted, it signals that the Nios II code memory is being loaded from the external ROM. |
| hc_rom_config_rom_rden | Output | Read-enable signal that connects to the external ROM. |
| hc_rom_config_rom_address | Output | ROM address that connects to the external ROM. |
| **DLL Reconfiguration Signals** | | |
| hc_dll_config_dll_offset_ctrl_addnsub | Input | Addition and subtraction control port for the DLL. This port controls if the delay-offset setting on hc_dll_config_dll_offset_ctrl_offset is added or subtracted. |
| hc_dll_config_offset_ctrl_offset | Input | Offset input setting for the DLL. This setting is a Gray-coded offset that is added or subtracted from the current value of the DLL's delay chain. |
| hc_dll_config_dll_offset_ctrl_offsetctrlout | Output | The registered and Gray-coded value of the current delay-offset setting. |
| **PLL Reconfiguration Signals** | | |
| hc_pll_config_configupdate | Input | Control signal to enable PLL reconfiguration. |
| hc_pll_config_phasecounterselect | Input | Specifies the counter select for dynamic phase adjustment. |
| hc_pll_config_phasestep | Input | Specfies the phase step for dynamic phase shifting. |
| hc_pll_config_phaseupdown | Input | Specifies whether the phase shift is up or down. |
| hc_pll_config_scanclk | Input | PLL reconfiguration scan chain clock. |
| hc_pll_config_scanclkena | Input | Clock enable port of the hc_pll_config_scanclk clock. |
| hc_pll_config_scandata | Input | Serial input data for the PLL reconfiguration scan chain. |

**Table 6–7. Top-Level HardCopy Migration Signals (Part 2 of 2)**

| Name | Direction | Description |
|---|---|---|
| hc_pll_config_scandataout | Output | Data output of the serial scanchain. |
| hc_pll_config_scandone | Output | This signal is asserted when the scan chain write operation is in progress. This signal is deasserted when the write operation is complete. |

Table 6–8 shows the parameters that Table 6–4 through Table 6–6 mention.

**Table 6–8. Parameters (Part 1 of 2)**

| Parameter Name | Description |
|---|---|
| AFI_RATIO | AFI_RATIO is 1 in full-rate designs.<br>AFI_RATIO is 2 in half-rate designs. |
| MEM_IF_DQS_WIDTH | The number of DQS pins in the interface. |
| MEM_ADDRESS_WIDTH | The address width of the specified memory device. |
| MEM_BANK_WIDTH | The bank width of the specified memory device. |
| MEM_CHIP_SELECT_WIDTH | The chip select width of the specified memory device. |
| MEM_CONTROL_WIDTH | The control width of the specified memory device. |
| MEM_DM_WIDTH | The DM width of the specified memory device. |
| MEM_DQ_WIDTH | The DQ width of the specified memory device. |
| MEM_READ_DQS_WIDTH | The READ DQS width of the specified memory device. |
| MEM_WRITE_DQS_WIDTH | The WRITE DQS width of the specified memory device. |
| OCT_SERIES_TERM_CONTROL_WIDTH | — |
| OCT_PARALLEL_TERM_CONTROL_WIDTH | — |
| AFI_ADDRESS_WIDTH | The AFI address width, derived from the corresponding memory interface width. |
| AFI_BANK_WIDTH | The AFI bank width, derived from the corresponding memory interface width. |
| AFI_CHIP_SELECT_WIDTH | The AFI chip select width, derived from the corresponding memory interface width. |
| AFI_DATA_MASK_WIDTH | The AFI data mask width. |
| AFI_CONTROL_WIDTH | The AFI control width, derived from the corresponding memory interface width. |
| AFI_DATA_WIDTH | The AFI data width. |
| AFI_DQS_WIDTH | The AFI DQS width. |
| DLL_DELAY_CTRL_WIDTH | The DLL delay output control width. |
| NUM_SUBGROUP_PER_READ_DQS | A read datapath parameter for timing purposes. |
| QVLD_EXTRA_FLOP_STAGES | A read datapath parameter for timing purposes. |
| READ_VALID_TIMEOUT_WIDTH | A read datapath parameter; calibration fails when the timeout counter expires. |
| READ_VALID_FIFO_WRITE_ADDR_WIDTH | A read datapath parameter; the write address width for half-rate clocks. |
| READ_VALID_FIFO_READ_ADDR_WIDTH | A read datapath parameter; the read address width for full-rate clocks. |

**Table 6–8. Parameters (Part 2 of 2)**

| Parameter Name | Description |
|---|---|
| MAX_LATENCY_COUNT_WIDTH | A latency calibration parameter; the maximum latency count width. |
| MAX_READ_LATENCY | A latency calibration parameter; the maximum read latency. |
| READ_FIFO_READ_ADDR_WIDTH | — |
| READ_FIFO_WRITE_ADDR_WIDTH | — |
| MAX_WRITE_LATENCY_COUNT_WIDTH | A write datapath parameter; the maximum write latency count width. |
| INIT_COUNT_WIDTH | An initailization sequence. |
| MRSC_COUNT_WIDTH | A memory-specific initialization parameter. |
| INIT_NOP_COUNT_WIDTH | A memory-specific initialization parameter. |
| MRS_CONFIGURATION | A memory-specific initialization parameter. |
| MRS_BURST_LENGTH | A memory-specific initialization parameter. |
| MRS_ADDRESS_MODE | A memory-specific initialization parameter. |
| MRS_DLL_RESET | A memory-specific initialization parameter. |
| MRS_IMP_MATCHING | A memory-specific initialization parameter. |
| MRS_ODT_EN | A memory-specific initialization parameter. |
| MRS_BURST_LENGTH | A memory-specific initialization parameter. |
| MEM_T_WL | A memory-specific initialization parameter. |
| MEM_T_RL | A memory-specific initialization parameter. |
| SEQ_BURST_COUNT_WIDTH | The burst count width for the sequencer. |
| VCALIB_COUNT_WIDTH | The width of a counter that the sequencer uses. |
| DOUBLE_MEM_DQ_WIDTH | — |
| HALF_AFI_DATA_WIDTH | — |
| CALIB_REG_WIDTH | The width of the calibration status register. |
| NUM_AFI_RESET | The number of AFI resets to generate. |

# PHY-to-Controller Interfaces

This section describes the typical modules that are connected to the UniPHY PHY and the port name prefixes each module uses. This section describes using a custom controller and describes the AFI.

The AFI standardizes and simplifies the interface between controller and PHY for all Altera memory designs, thus allowing you to easily interchange your own controller code with Altera's high-performance controllers. The AFI PHY includes an administration block that configures the memory for calibration and performs necessary accesses to mode registers that configure the memory as required.

For half-rate designs, the address and command signals in the UniPHY are asserted for one mem_clk cycle (1T addressing), such that there are two input bits per address and command pin in half-rate designs. If you require a more conservative 2T addressing, drive both input bits (of the address and command signal) identically in half-rate designs.

Figure 6–9 shows the half-rate write operation.

**Figure 6–9. Half-Rate Write with Word-Aligned Data**

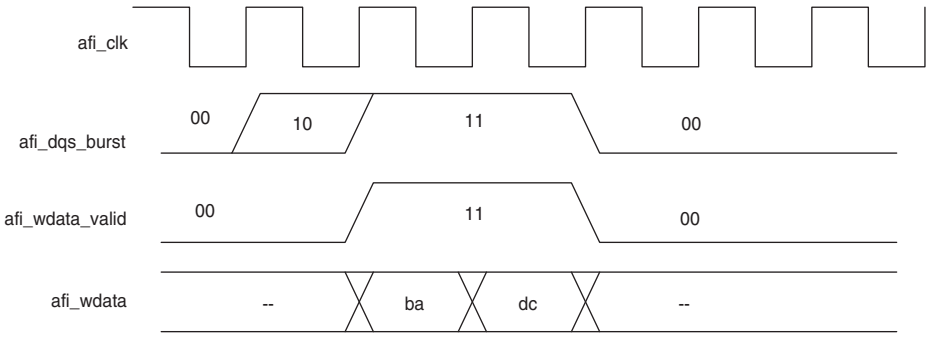
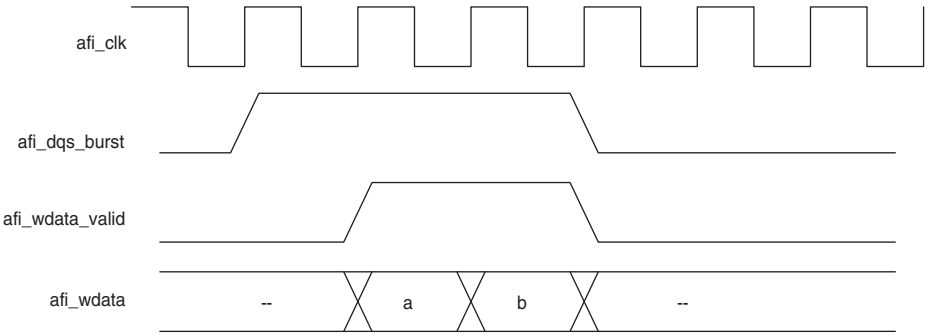



Figure 6–10 shows a full-rate write.

**Figure 6–10. Full-Rate Write**




After calibration is completed, the sequencer sends the write latency in number of clock cycles to the controller.

Figure 6–11 shows full-rate reads; Figure 6–12 shows half-rate reads.

**Figure 6–11. Full-Rate Reads**

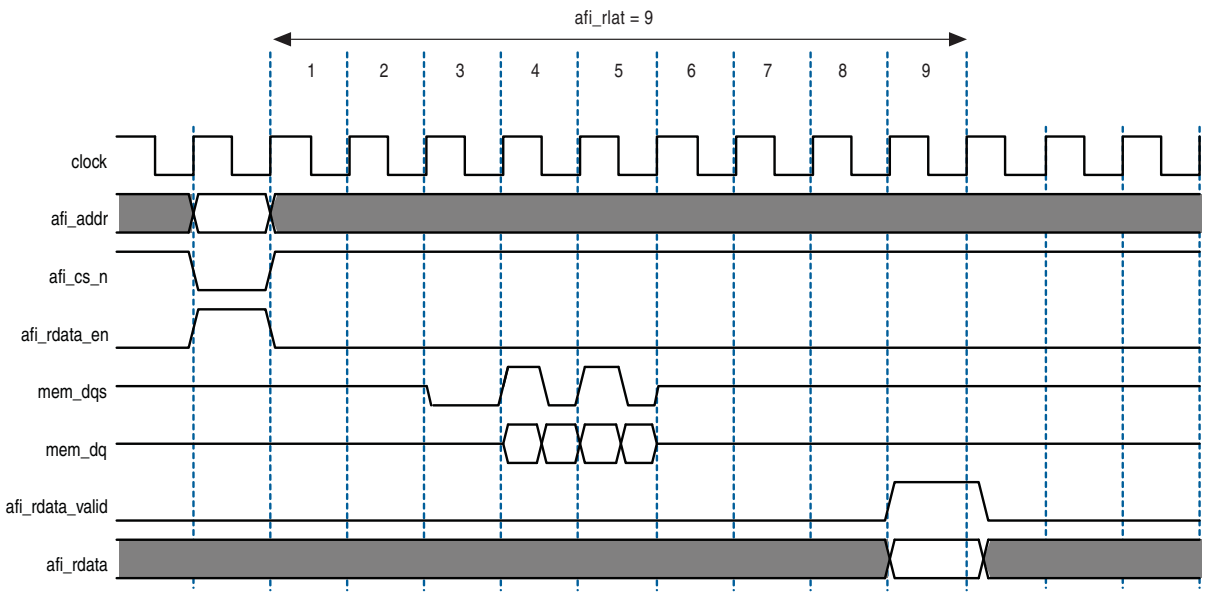



**Figure 6–12. Half-Rate Reads**

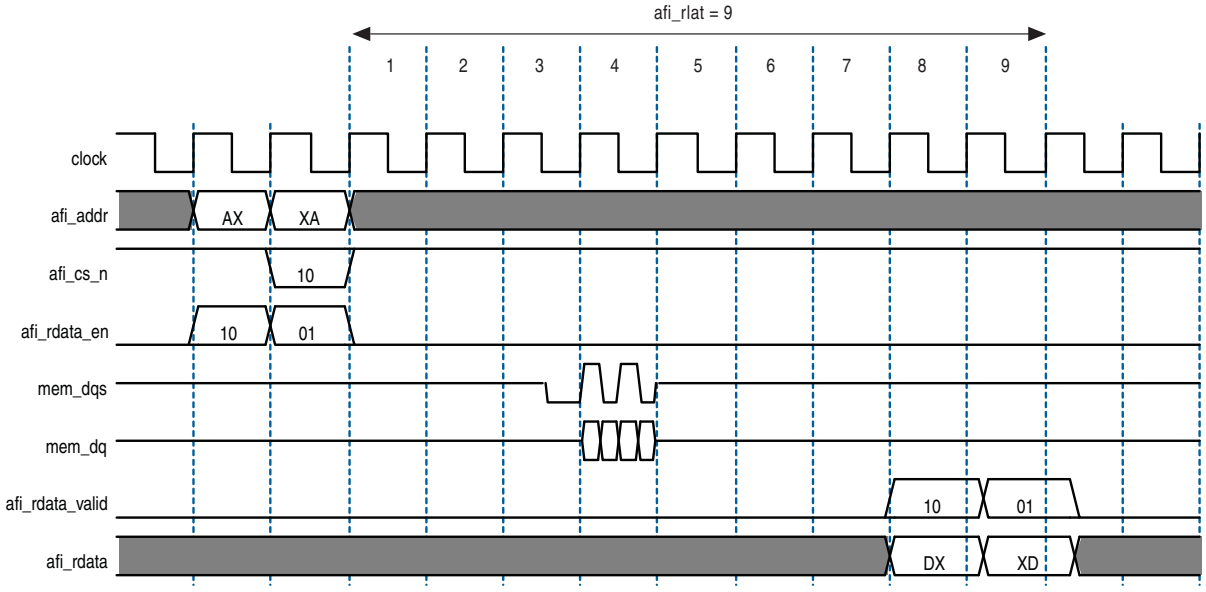



Figure 6–13 and Figure 6–14 show writes and reads, where the IP core writes data to and reads from the same address. In each example, `afi_rdata` and `afi_wdata` are aligned with controller clock (`afi_clk`) cycles. All the data in the bit vector is valid at once.

The AFI has the following conventions:

- With the AFI, high and low signals are combined in one signal, so for a single chip select (`afi_cs_n`) interface, `afi_cs_n[1:0]`, location 0 appears on the memory bus on one `mem_clk` cycle and location 1 on the next `mem_clk` cycle.

  
  This convention is maintained for all signals so for an 8 bit memory interface, the write data (`afi_wdata`) signal is `afi_wdata[31:0]`, where the first data on the DQ pins is `afi_wdata[7:0]`, then `afi_wdata[15:8]`, then `afi_wdata[23:16]`, then `afi_wdata[31:24]`.

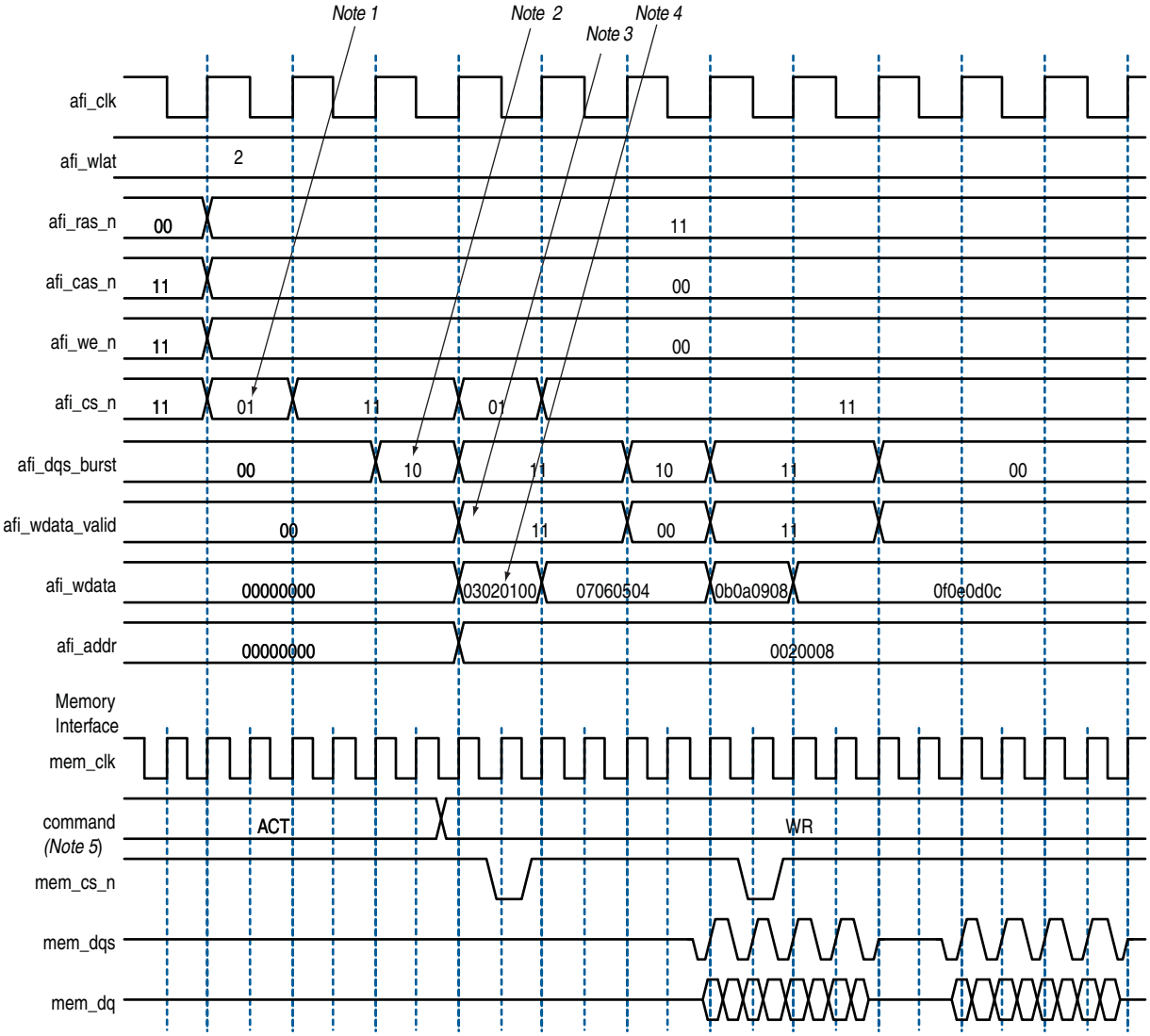
- Spaced reads and writes have the following definitions:
  - Spaced writes—write commands separated by a gap of one controller clock (`afi_clk`) cycle.
  - Spaced reads—read commands separated by a gap of one controller clock (`afi_clk`) cycle.

Figure 6–13 through Figure 6–14 assume the following general points:

- The burst length is four.
- An 8-bit interface with one chip select.

- The data for one controller clock (afi_clk) cycle represents data for two memory clock (mem_clk) cycles (half-rate interface).

**Figure 6–13. Word-Aligned Writes**




**Notes to Figure 6–13:**

(1) To show the even alignment of afi_cs_n, expand the signal (this convention applies for all other signals).

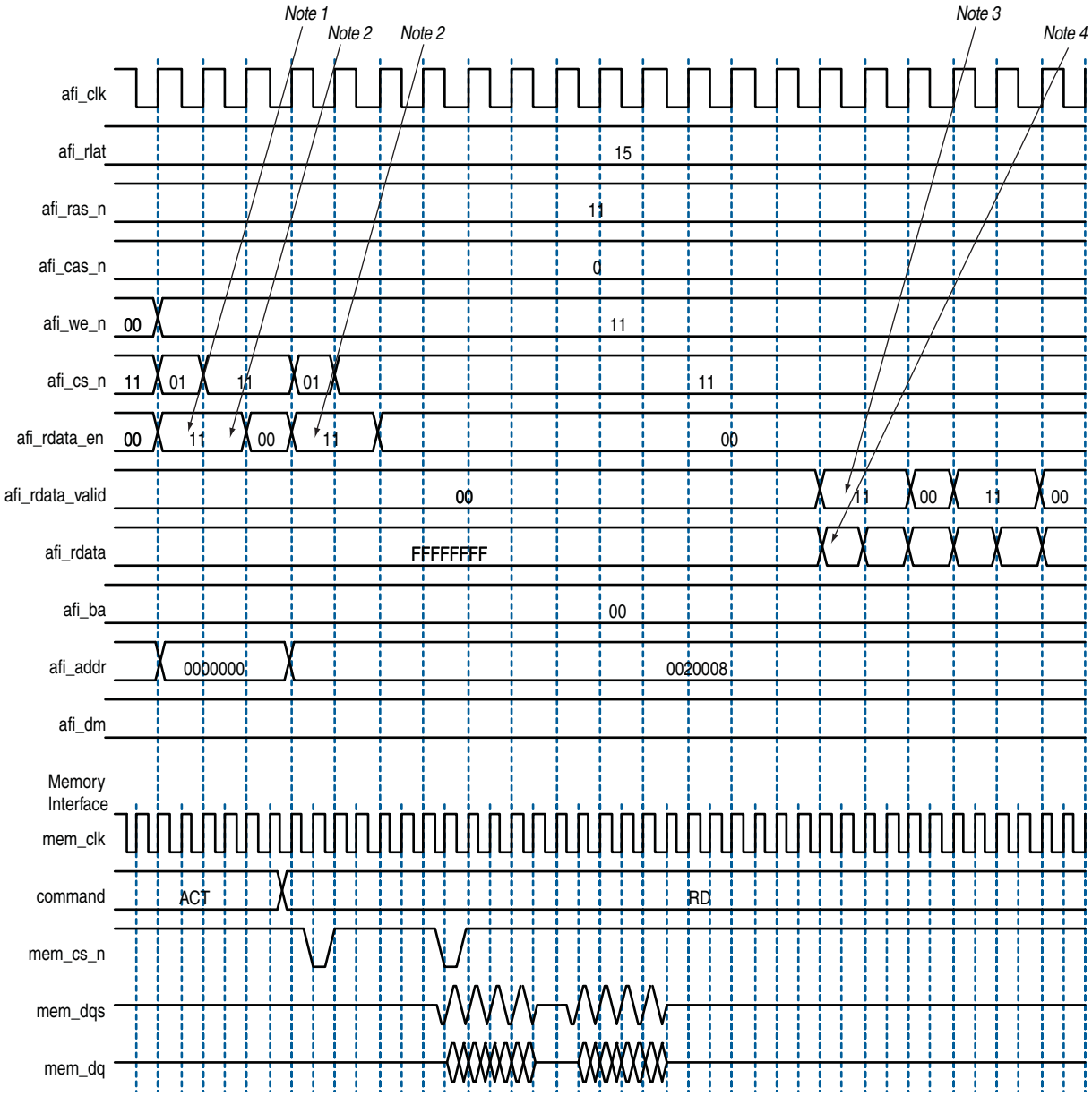
(2) The afi_dqs_burst must go high one memory clock cycle before afi_wdata_valid. Compare with the word-unaligned case.

(3) The afi_wdata_valid is asserted two afi_wlat controller clock (afi_clk) cycles after chip select (afi_cs_n) is asserted. The afi_wlat indicates the required write latency in the system. The value is determined during calibration and is dependant upon the relative delays in the address and command path and the write datapath in both the PHY and the external DDR SDRAM subsystem. The controller must drive afi_cs_n and then wait afi_wlat (two in this example) afi_clks before driving afi_wdata_valid.

(4) Observe the ordering of write data (afi_wdata). Compare this to data on the mem_dq signal.

(5) In all waveforms a command record is added that combines the memory pins ras_n, cas_n and we_n into the current command that is issued. This command is registered by the memory when chip select (mem_cs_n) is low. The important commands in the presented waveforms are WR = write, ACT = activate.

**Figure 6–14. Word-Aligned Reads**




**Notes to Figure 6–14:**

(1) For AFI, `afi_rdata_en` is required to be asserted one memory clock cycle before chip select (`afi_cs_n`) is asserted. In the half-rate `afi_clk` domain, this requirement manifests as the controller driving `11` (as opposed to the `01`) on `afi_rdata_en`.

(2) AFI requires that `afi_rdata_en` is driven for the duration of the read. In this example, it is driven to `11` for two half-rate `afi_clks`, which equates to driving to `1`, for the four memory clock cycles of this four-beat burst.

(3) The `afi_rdata_valid` returns 15 (`afi_rlat`) controller clock (`afi_clk`) cycles after `afi_rdata_en` is asserted. Returned is when the `afi_rdata_valid` signal is observed at the output of a register within the controller. A controller can use the `afi_rlat` value to determine when to register to returned data, but this is unnecessary as the `afi_rdata_valid` is provided for the controller to use as an enable when registering read data.

(4) Observe the alignment of returned read data with respect to data on the bus.

# Using a Custom Controller

The UniPHY-based memory interface IP cores are delivered with both the PHY and the memory controller integrated. To replace the Altera high-performance memory controller with a custom memory controller, perform the following steps:

1. Parameterize and generate your variation of the UniPHY-based memory controller IP.

   This step creates a top-level HDL file called <*variation_name*>**.v** (or <*variation_name*>**.vhd**), and a sub-directory called <*variation_name*>.

   The top-level module instantiates the <*variation_name*>_<*stamp*>**_controller_phy** module in the <*variation_name*> subdirectory. The <*variation_name*>_<*stamp*>**_controller_phy** module instantiates the PHY and the controller.

   ☞ <*stamp*> is a unique identifier determined by the MegaWizard Plug-in Manager, SOPC Builder, or Qsys, during generation.

2. Open the <*variation_name*>**/**<*variation_name*>_<*stamp*>**_controller_phy.sv** file.
3. Replace the <*variation_name*>_<*stamp*>**_alt_ddrx_controller** module with your custom controller module.
4. Delete the ports of the Altera high-performance memory controller, and add the top-level ports of your custom controller.
5. Similarily, update the port names in the top-level module in the <*variation_name*>**.v** or <*variation_name*>**.vhd** file.
6. Compile and simulate the design to confirm correct operation.

☞ Regenerating the UniPHY memory interface IP erases all modifications made to the HDL files. The parameters you select in the MegaWizard are stored in the top-level <*variation_name*> module; hence, you must repeat the above steps every time you regenerate the IP variation.

☞ For half-rate controllers, AFI signals are double the bus width of the memory interface. Half rate controllers have double the width of the signal and run at half the speed. Hence, the overall bandwidth is maintained. Such double-width signals are divided into two signals for transmission to the memory interface, with a higher order bits representing the most-significant bit (MSB) and a lower order bits representing the least-significant bit (LSB). The LSB is transmitted first, and is followed by the MSB.

# Using a Vendor-Specific Memory Model

You can replace the Altera-supplied memory model with a vendor-specific memory model. In general, you may find vendor-specific models to be standardized, thorough, and well supported, but sometimes more complex to setup and use.

If you do want to replace the Altera-supplied memory model with a vendor-supplied memory model, observe the following guidelines:

- Ensure that the vendor-supplied memory model that you have is correct for your memory device.
- Disconnect all signals from the default memory model and reconnect them to the vendor-supplied memory model.
- If you intend to run simulation from the Quartus II software, ensure that the **.qip** file points to the vendor-supplied memory model.

# 7. Functional Description—Example Top-Level Project

IP generation creates an example top-level project that shows you how to instantiate and connect the controller.

The example top-level project contains a testbench, which is for use with Verilog HDL only language simulators such as ModelSim-AE Verilog, and shows simple operation of the memory interface.

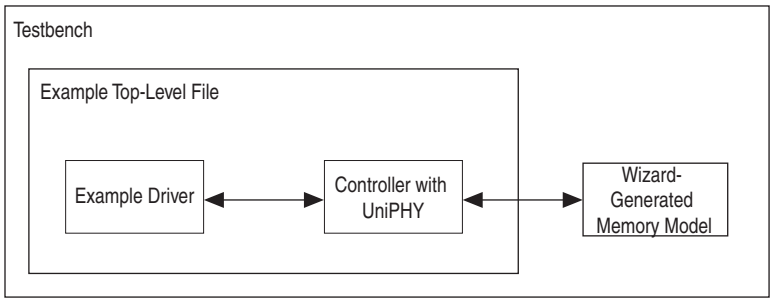
☞ For a VHDL-only simulation, use the VHDL IP functional simulation model.

The testbench contains the following blocks:

- A synthesizable Avalon-MM example driver, which acts as a traffic generator block and implements a pseudo-random pattern of reads and writes to a parameterized number of addresses. The driver also monitors the data read from the memory to ensure it matches the written data and asserts a failure otherwise.
- An instance of the controller, which interfaces between the Avalon-MM interface and the AFI.
- The UniPHY IP, which serves as an interface between the memory controller and external memory device(s) to perform read and write operations to the memory.
- A memory model, which acts as a generic model that adheres to the memory protocol specifications. Memory vendors also provide simulation models for specific memory components that can be downloaded from their websites. This block is available in Verilog HDL only.

Figure 7–1 shows the testbench and the example top-level file.

**Figure 7–1. Testbench and Example Top-Level File**



# Example Driver

The example driver for Avalon-MM memory interfaces generates Avalon-MM traffic on an Avalon-MM master interface. As the read and write traffic is generated, the expected read response is stored internally and compared to the read responses as they arrive. If all reads report their expected response, the pass signal is asserted; however, if any read responds with unexpected data a fail signal is asserted.

Each operation generated by the driver is a single write or block of writes followed by a single read or block of reads to the same addresses, which allows the driver to precisely determine the data that should be expected when the read data is returned by the memory interface. The driver comprises a traffic generation block, the Avalon-MM interface and a read comparison block. The traffic generation block generates addresses and write data, which are then sent out over the Avalon-MM interface. The read comparison block compares the read data received from the Avalon-MM interface to the write data from the traffic generator. If at any time the data received is not the expected data, the read comparison block records the failure, finishes reading all the data, and then signals that there is a failure and the driver enters a fail state. If all patterns have been generated and compared successfully, the driver enters a pass state.

Within the driver, there are the following main states:

- Generation of individual read and writes
- Generation of block read and writes
- The pass state
- The fail state

Within each of the generation states there are the following substates:

- Sequential address generation
- Random address generation
- Mixed sequential and random address generation

For each of the states and substates, the order and number of operations generated for each substate is parameterizable—you can decide how many of each address pattern to generate, or can disable certain patterns entirely if you want. The sequential and random interleave substate takes in additions to the number of operations to generate. An additional parameter specifies the ratio of sequential to random addresses to generate randomly.

## Read and Write Generation

The traffic generator block can perform individual or block read and write generation.

### Individual Read and Write Generation

During the individual read and write generation stage of the driver, the traffic generation block generates individual write followed by individual read Avalon-MM transactions, where the address for the transactions are chosen according to the specific substate. The width of the Avalon-MM interface is a global parameter for the driver, but each substate can have a parameterizable range of burst lengths for each operation.

### Block Read and Write Generation

During the block read and write generation state of the driver, the traffic generator block generates a parameterizable number of write operations followed by the same number of read operations. The specific addresses generated for the blocks are chosen by the specific substates. The burst length of each block operation can be parameterized by a range of acceptable burst lengths.

## Address and Burst Length Generation

The traffic generator block can perform sequential or random addressing.

### Sequential Addressing

The sequential addressing substate defines a traffic pattern where addresses are chosen in sequential order starting from a user definable address. The number of operations in this substate is parameterizable.

### Random Addressing

The random addressing substate defines a traffic pattern where addresses are chosen randomly over a parameterizable range. The number of operations in this substate is parameterizable.

### Sequential and Random Interleaved Addressing

The sequential and random interleaved addressing substate defines a traffic pattern where addresses are chosen to be either sequential or random based on a parameterizable ratio. The acceptable address range is parameterizable as is the number of operations to perform in this substate.

## Example Driver Signals

Table 7–1 lists the signals used by the example driver.

**Table 7–1. Driver Signals (Part 1 of 2)**

| Signal | Width | Signal Type |
|---|---|---|
| clk | | |
| reset_n | | |
| avl_ready | | avl_ready |
| avl_write_req | | avl_write_req |
| avl_read_req | | avl_read_req |
| avl_addr | 24 | avl_addr |
| avl_size | 3 | avl_size |
| avl_wdata | 72 | avl_wdata |
| avl_rdata | 72 | avl_rdata |
| avl_rdata_valid | | avl_rdata_valid |
| pnf_per_bit | | pnf_per_bit |
| pnf_per_bit_ persist | | pnf_per_bit_persist |

**Table 7–1. Driver Signals (Part 2 of 2)**

| Signal | Width | Signal Type |
|---|---|---|
| pass | | pass |
| fail | | fail |
| test_complete | | test_complete |

## Example Driver Add-Ons

Some optional components that can be useful for verifying aspects of the controller and PHY operation are generated in conjunction with certain user-specified options. These add-on components are self-contained, and are not part of the controller or PHY, nor the example driver.

### User Refresh Generator

The user refresh generator sends refresh requests to the memory controller when user refresh is enabled. The memory controller returns an acknowledgement signal and then issues the refresh command to the memory device.

The user refresh generator is created when you turn on **Enable User Refresh** under **Controller Settings** on the **General Settings** tab of the parameter editor. The user refresh generator is instantiated by **example_top_v**. and resides in the example_project subdirectory.

### Refresh Monitor

As its name implies, the refresh monitor monitors refresh commands from the controller and verifies that those commands conform to the necessary refresh timing parameters.

The refresh monitor is created when you turn on **Enable User Refresh** under **Controller Settings** on the **General Settings** tab of the parameter editor. The refresh monitor is instantiated by **example_top_tb.v** and resides in **refresh_monitor.sv** in the **rtl_sim** subdirectory.

### Data Corrupter

The data corrupter intercepts read data in the memory interface bus and introduces errors to that data to test the error detection function in the memory controller. Both the rate of error injection and the number of error bits are configurable (although the per-byte parity protection feature supports only 1 bit error detection).

The data corrupter employs four types of error injection:

- per-bit data corruption in a single memory burst
- per-byte corruption in a single memory burst
- per-bit all-burst corruption
- per-byte all burst corruption

Throughout the four types of error injection tests, the data corrupter exercises a walking-one pattern to confirm correctness.

The data corrupter is created when you turn on **Enable Error Detection Parity** under **Controller Settings** on the **General Settings** tab of the parameter editor. The data corrupter resides in **data_corrupter.sv** in the **rtl_sim** subdirectory.

# 8. Latency

Altera defines read and write latencies in terms of memory clock cycles. There are two types of latencies that exists while designing with memory controllers—read and write latencies, which have the following definitions:

- Read latency—the amount of time it takes for the read data to appear at the local interface after initiating the read request.
- Write latency—the amount of time it takes for the write data to appear at the memory interface after initiating the write request.

For a half-rate controller, the local side frequency is half of the memory interface frequency. For a full-rate controller, the local side frequency is equal to the memory interface frequency.

Table 8–1 shows the DDR2 SDRAM latency in full rate memory clock cycles.

**Table 8–1. DDR2 SDRAM Controller Latency (In Full-Rate Memory Clock Cycles) *(Note 3)***

| Rate | Controller Address and Command *(4)* | PHY Address and Command | Memory Maximum Read | PHY Read Return | Round Trip | Round Trip without Memory |
|---|---|---|---|---|---|---|
| Full | 5 | 1 | 3–7 | 4 | 13–17 | 10 |
| Half | 10 | 3 *(1)*<br>4 *(2)* | 3–7 | 8 | 24–28 *(1)*<br>26–28 *(2)* | 21 *(1)*<br>22 *(2)* |

**Notes to Table 8–1:**

(1) Even write latency.

(2) Odd write latency.

(3) Latency is the number of cycles between the first register of the current stage capturing cmd/data, and the first register in the next stage capturing cmd/data.

(4) Latency shown is best case, for maximum performance specifications. Latency may be higher due to protocol requirements; controller latency may be lower for slower frequencies.

Table 8–2 shows the DDR3 SDRAM latency in full rate memory clock cycles.

**Table 8–2. DDR3 SDRAM Controller Latency (In Full-Rate Memory Clock Cycles) *(Note 5)***

| Rate | Controller Address and Command *(6)* | PHY Address and Command | Memory Maximum Read | PHY Read Return | Round Trip | Round Trip without Memory |
|---|---|---|---|---|---|---|
| Half | 10 | 3 *(1) (2)*<br>4 *(3) (4)* | 5-11 | 7 *(1)*<br>8 *(2)*<br>8 *(3)*<br>7 *(4)* | 26–30 *(1)*<br>26–32 *(2)*<br>28–32 *(3)*<br>26–32 *(4)* | 20 *(1)*<br>21 *(2)*<br>22 *(3)*<br>21 *(4)* |

**Notes to Table 8–2:**

(1) Even write latency and even read latency.

(2) Even write latency and odd read latency.

(3) Odd write latency and even read latency.

(4) Odd write latency and odd read latency.

(5) Latency is the number of cycles between the first register of the current stage capturing cmd/data, and the first register in the next stage capturing cmd/data.

(6) Latency shown is best case, for maximum performance specifications. Latency may be higher due to protocol requirements; controller latency may be lower for slower frequencies.

# Variable Controller Latency

The variable controller latency feature allows you to take advantage of lower latency for variations designed to run at lower frequency. When deciding whether to vary the controller latency from the default value of 0, consider the following points:

- Reduced latency can help achieve a reduction in resource usage and clock cycles in the controller, but might result in lower $f_{MAX}$.
- Increased latency can help achieve greater $f_{MAX}$, but might consume more clock cycles in the controller and result in increased resource usage.

If you select a latency value that is inappropriate for the target frequency, the system displays a warning message in the message area at the bottom of the GUI.

You can change the controller latency by altering the value of the **Reduce Controller Latency** parameter in the **Efficiency** section of the **Controller Settings** tab of the DDR2 or DDR3 SDRAM controller with UniPHY parameter editor.

# 9. Timing Diagrams

The timing diagrams in this chapter are for DDR2 SDRAM with the following parameters:

- `avl_size` is set as 2
- DQ burst length of 8

Figure 9–1 shows DDR2 SDRAM writes.

**Figure 9–1. DDR2 SDRAM Writes**

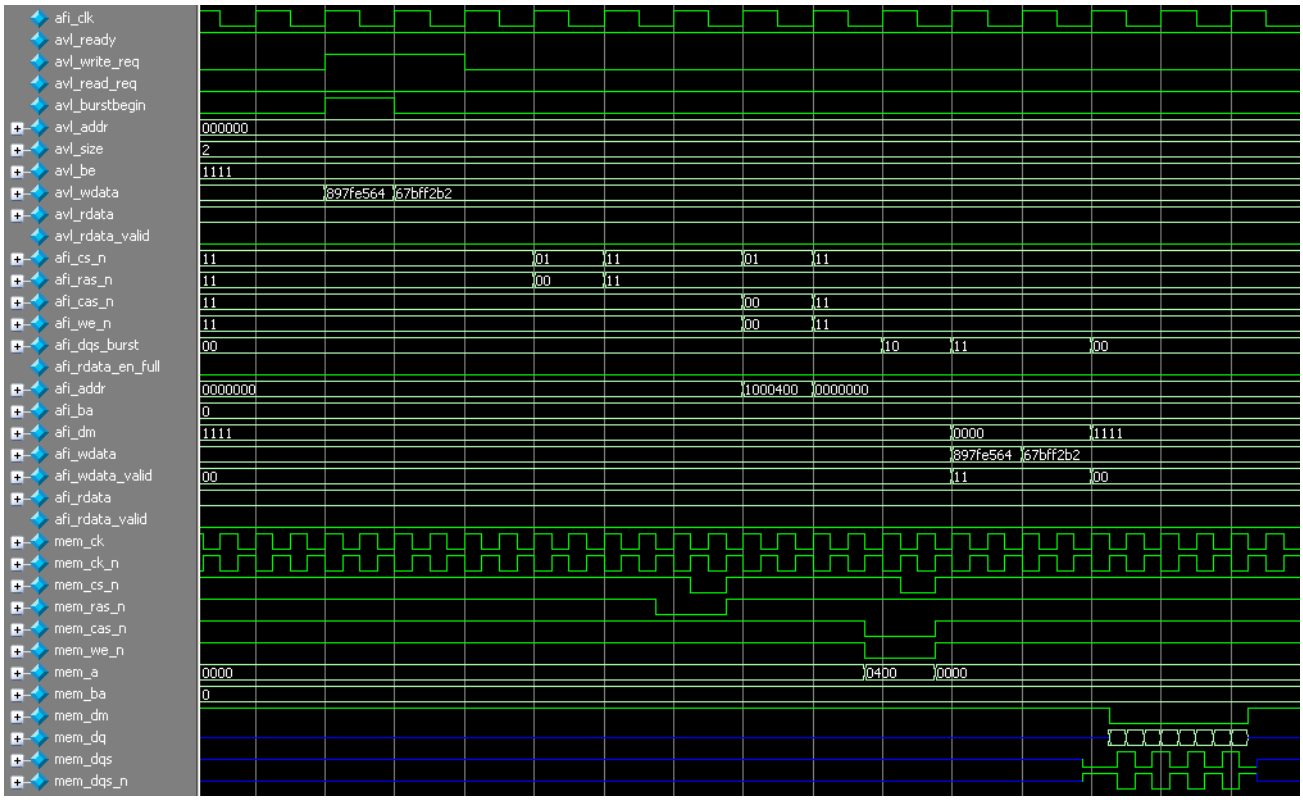
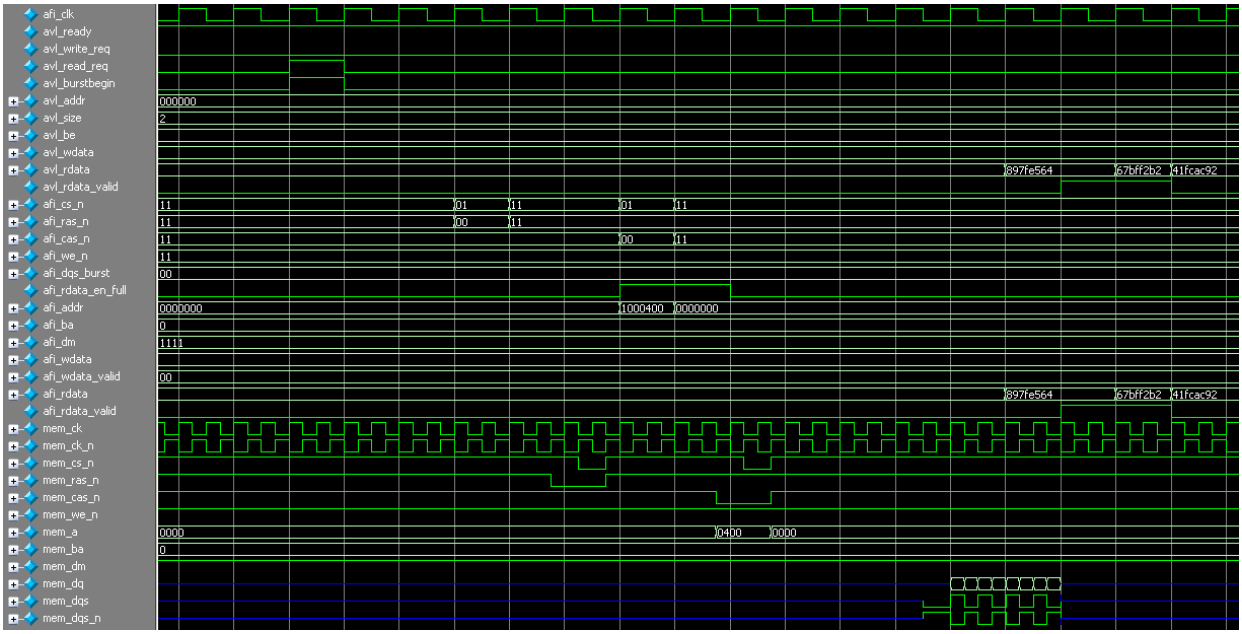


Figure 9–2 shows DDR2 SDRAM reads.

**Figure 9–2. DDR2 SDRAM Reads**



# 10. Upgrading to UniPHY-based Controllers from ALTMEMPHY-based Controllers

This chapter describes upgrading the following ALTMEMPHY-based controller designs to UniPHY-based controllers:

- DDR2 or DDR3 SDRAM High-Performance Controller II with ALTMEMPHY designs
- DDR2 or DDR3 SDRAM High-Performance Controller with ALTMEMPHY designs

Altera does not support upgrading designs that do not use the AFI.

If your design uses non-AFI IP cores, Altera recommends that you start a new design with the UniPHY IP core . In addition, Altera recommends that any new designs targeting Stratix III, Stratix IV, or Stratix V use the UniPHY datapath.

## Upgrading from DDR2 or DDR3 SDRAM High-Performance Controller II with ALTMEMPHY Designs

To upgrade to the DDR2 or DDR3 SDRAM controller with UniPHY IP core from DDR2 or DDR3 SDRAM High-Performance Controller II with ALTMEMPHY designs, follow these steps:

1. Generating Equivalent Design
2. Replacing the ALTMEMPHY Datapath with UniPHY Datapath
3. Resolving Port Name Differences
4. Creating OCT Signals
5. Running Pin Assignments Script
6. Removing Obsolete Files
7. Simulating your Design

The following sections describes these steps.

### Generating Equivalent Design

Create a new DDR2 or DDR3 SDRAM controller with UniPHY IP core, by following the steps in "MegaWizard Plug-In Manager Flow" on page 2–3 and use the following guidelines:

- Specify the same variation name as the ALTMEMPHY variation.
- Specify a directory different than the ALTMEMPHY design directory to prevent files from overwriting each other during generation.

To ease the migration process, ensure the UniPHY-based design you create is as similar as possible to the existing ALTMEMPHY-based design. In particular, you should ensure the following settings are the same in your UniPHY-based design:

- **PHY settings** tab
  - FPGA speed grade
  - PLL reference clock
  - Memory clock frequency
  - There is no need to change the default **Address and command clock phase settings**; however, if you have board skew effects in your ALTMEMPHY design, enter the difference between that clock phase and the default clock phase into the **Address and command clock phase settings**.
- **Memory Parameters** tab—all parameters must match.
- **Memory Timing** tab—all parameters must match.
- **Board settings** tab—all parameters must match.
- **Controller settings** tab—all parameters must match

☞ In ALTMEMPHY-based designs you can turn off dynamic OCT. However, all UnIPHY-based designs use dynamic parallel OCT and you cannot turn it off.

## Replacing the ALTMEMPHY Datapath with UniPHY Datapath

To replace the ALTMEMPHY datapath with the UniPHY datapath, follow these steps:

1. In the Quartus II software, open the Assignment Editor, on the Assignments menu click **Assignment Editor**.
2. Manually, delete all of the assignments related to the external memory interface pins, except for the location assignments if you are preserving the pinout. By default, these pin names start with the `mem` prefix, though in your design they may have a different name.
3. Remove the old ALTMEMPHY **.qip** file from the project:
   - On the Assignments menu click **Settings**.
   - Specify the old **.qip**, and click **Remove**.

Your design now uses the UniPHY datapath.

## Resolving Port Name Differences

Several port names in the ALTMEMPHY datapath are different than in the UniPHY datapath. The different names may cause compilation errors. This section describes the changes you must make in the RTL for the entity that instantiates the memory IP core. Each change applies to a specific port in the ALTMEMPHY datapath. Unconnected ports require no changes.

In some instances, multiple ports in ALTMEMPHY-based designs are mapped to a single port in UniPHY-based designs. If you use both ports in ALTMEMPHY-based designs, assign a temporary signal to the common port and connect it to the original wires. Table 10–1 shows the changes you must make.

**Table 10–1. Changes to ALTMEMPHY Port Names**

| ALTMEMPHY Port | Changes |
|---|---|
| aux_full_rate_clk | Rename to pll_mem_clk. |
| aux_half_rate_clk | Rename to pll_afi_clk. |
| aux_scan_clk | The UniPHY-based design does not generate this signal. You can generate it if you require it. |
| aux_scan_clk_reset_n | The UniPHY-based design does not generate this signal. You can generate it if you require it. |
| dll_reference_clk | Rename to pll_mem_clk. |
| dqs_delay_ctrl_export | This signal is for DLL sharing between ALTMEMPHY instances and is not applicable for UniPHY-based designs. |
| local_address | Rename to avl_addr. |
| local_be | Rename to avl_be. |
| local_burstbegin | Rename to avl_burstbegin. |
| local_rdata | Rename to avl_rdata. |
| local_rdata_valid | Rename to avl_rdata_valid. |
| local_read_req | Rename to avl_read_req. |
| local_ready | Rename to avl_ready. |
| local_size | Rename to avl_size. |
| local_wdata | Rename to avl_wdata. |
| local_write_req | Rename to avl_write_req. |
| mem_addr | Rename to mem_a. |
| mem_clk | Rename to mem_ck. |
| mem_clk_n | Rename to mem_ck_n. |
| mem_dqsn | Rename to mem_dqs_n. |
| oct_ctl_rs_value | Remove from design ("Creating OCT Signals" on page 10–4). |
| oct_ctl_rt_value | Remove from design ("Creating OCT Signals" on page 10–4). |
| phy_clk | Rename to afi_clk. |
| reset_phy_clk_n | Rename to afi_reset_n. |
| local_refresh_ack<br>local_wdata_req | The controller no longer exposes these signals to the top-level design, so comment out these outputs. If you need it, bring the wire out from the high-performance II controller entity in *<project directory>*/**uniphy/rtl/***<variation name>*_**controller_phy.sv**. |

## Creating OCT Signals

In ALTMEMPHY-based designs, the Quartus II Fitter creates the `alt_oct` block outside the IP core and connects it to the `oct_ctl_rs_value` and `oct_ctl_rt_value` signals. In UniPHY-based designs, the OCT block is part of the IP core, so the design no longer requires these two ports. Instead, the UniPHY-based design requires two additional ports, `oct_rup` and `oct_rdn`. You must create these ports in the instantiating entity as input pins and connect to the UniPHY instance. Then route these pins to the top-level design and connect to the OCT $R_{UP}$ and $R_{DOWN}$ resistors on the board.

For information on OCT control block sharing, refer to "The OCT Sharing Interface" on page 6–12.

## Running Pin Assignments Script

Remap your design by running analysis and synthesis. When analysis and synthesis completes, run the pin assignments Tcl script ("Add Pin and DQ Group Assignments" on page 4–1), then verify the new pin assignments in the Assignment Editor.

## Removing Obsolete Files

After you upgrade the design, you may remove the unnecessary ALTMEMPHY design files from your project. To identify these files, examine the original ALTMEMPHY-generated **.qip** file in any text editor.

## Simulating your Design

You must use the UniPHY memory model to simulate your new design. To use the UniPHY memory model, follow these steps:

1. Edit your instantiation of the UniPHY datapath to ensure the `local_init_done`, `afi_cal_success`, and `afi_cal_fail` signals are at the top-level entity so that an instantiating testbench can refer to those signals.
2. Specify that your third-party simulator should use the UniPHY testbench and memory model instead of the ALTMEMPHY memory model:
   a. On the Assignments menu, point to **Settings** and click the **Project Settings** window.
   b. Select the **Simulation** tab, click **Test Benches**, click **Edit**, and replace the ALTMEMPHY testbench files with the following files:
   - **\uniphy**<*variation name*>**\rtl_sim**<*variation name*>**_example_top_tb.v**
   - **\uniphy**<*variation name*>**\example_project\mem_model.sv**
3. Open up the UniPHY testbench file and find the following line in the testbench:

   ```
   force dut.mem_if.controller_phy_inst... = SEQ_CALIB_INIT;
   ```

4. Change the PHY instance name from the default `mem_if` to the instance name in which you instantiated the UniPHY datapath.

5. Update the following port names of the example design in the UniPHY-generated testbench (Table 10–2).

**Table 10–2. Example Design Port Names**

| Example Design Name | New Name |
|---|---|
| `pll_ref_clk` | Rename to `clock_source`. |
| `mem_a` | Rename to `mem_addr`. |
| `mem_ck` | Rename to `mem_clk`. |
| `mem_ck_n` | Rename to `mem_clk_n`. |
| `mem_dqs_n` | Rename to `mem_dqsn`. |
| `pass` | Rename to `pnf`. |

# Upgrading from DDR2 or DDR3 SDRAM High-Performance Controller with ALTMEMPHY Designs

To upgrade from DDR2 or DDR3 SDRAM High-Performance Controller with ALTMEMPHY designs, follow these steps

1. Upgrade your design to use the high-performance controller II.
2. Follow the upgrade flow in "Upgrading from DDR2 or DDR3 SDRAM High-Performance Controller II with ALTMEMPHY Designs" on page 10–1.

This chapter provides additional information about the document and Altera.

## Document Revision History

The following table shows the revision history for this document.

| Date | Version | Changes |
|---|---|---|
| December 2010 | 1.1 | ■ Updated Device Family Support, Features list, and Resource Utilization tables.<br>■ Updated Design Flows, HardCopy Migration Design Guidelines, and Generated Files information.<br>■ Updated Parameter Settings chapter.<br>■ Updated ODT Generation Logic, Reset and Clock Generation, and Write Datapath.<br>■ Added The OCT Sharing Interface, and expanded Using a Custom Controller information.<br>■ Updated Latency data.<br>■ Updated Creating OCT Signals in Upgrading to UniPHY-based Controllers from ALTMEMPHY-based Controllers. |
| July 2010 | 1.0 | First published. |

## How to Contact Altera

To locate the most up-to-date information about Altera products, refer to the following table.

| Contact *(1)* | Contact Method | Address |
|---|---|---|
| Technical support | Website | www.altera.com/support |
| Technical training | Website | www.altera.com/training |
| | Email | custrain@altera.com |
| Product literature | Website | www.altera.com/literature |
| Non-technical support (General) | Email | nacomp@altera.com |
| (Software Licensing) | Email | authorization@altera.com |

**Note to Table:**

(1) You can also contact your local Altera sales office or sales representative.

# Typographic Conventions

The following table shows the typographic conventions this document uses.

| Visual Cue | Meaning |
|---|---|
| **Bold Type with Initial Capital Letters** | Indicate command names, dialog box titles, dialog box options, and other GUI labels. For example, **Save As** dialog box. For GUI elements, capitalization matches the GUI. |
| **bold type** | Indicates directory names, project names, disk drive names, file names, file name extensions, software utility names, and GUI labels. For example, **\qdesigns** directory, **D:** drive, and **chiptrip.gdf** file. |
| *Italic Type with Initial Capital Letters* | Indicate document titles. For example, *Stratix IV Design Guidelines*. |
| *italic type* | Indicates variables. For example, *n* + 1.<br>Variable names are enclosed in angle brackets (< >). For example, *<file name>* and *<project name>***.pof** file. |
| Initial Capital Letters | Indicate keyboard keys and menu names. For example, the Delete key and the Options menu. |
| "Subheading Title" | Quotation marks indicate references to sections within a document and titles of Quartus II Help topics. For example, "Typographic Conventions." |
| `Courier type` | Indicates signal, port, register, bit, block, and primitive names. For example, `data1`, `tdi`, and `input`. The suffix `n` denotes an active-low signal. For example, `resetn`.<br>Indicates command line commands and anything that must be typed exactly as it appears. For example, `c:\qdesigns\tutorial\chiptrip.gdf`.<br>Also indicates sections of an actual file, such as a Report File, references to parts of files (for example, the AHDL keyword `SUBDESIGN`), and logic function names (for example, `TRI`). |
| ↵ | An angled arrow instructs you to press the Enter key. |
| 1., 2., 3., and<br>a., b., c., and so on | Numbered steps indicate a list of items when the sequence of the items is important, such as the steps listed in a procedure. |
| ■ ■ ■ | Bullets indicate a list of items when the sequence of the items is not important. |
|  | The hand points to information that requires special attention. |
|  | A question mark directs you to a software help system with related information. |
|  | The feet direct you to another document or website with related information. |
|  | A caution calls attention to a condition or possible situation that can damage or destroy the product or your work. |
|  | A warning calls attention to a condition or possible situation that can cause you injury. |
|  | The envelope links to the Email Subscription Management Center page of the Altera website, where you can sign up to receive update notifications for Altera documents. |

# External Memory Interface Handbook Volume 3

## Section III. QDR II and QDR II+ SRAM Controller with UniPHY User Guide

101 Innovation Drive
San Jose, CA 95134
www.altera.com

EMI_QDRII_UG-2.1

Document last updated for Altera Complete Design Suite version: 10.1
Document publication date: December 2010

Subscribe

# Contents

## Chapter 9. Timing Diagrams

## Additional Information

# 1. About This IP

The Altera QDR II and QDR II+ SRAM controllers with UniPHY provide simplified interfaces to industry-standard QDR II and QDR II+ SRAM.

The QDR II and QDR II+ SRAM controllers with UniPHY offer full-rate or half-rate QDR II and QDR II+ SRAM interfaces. The UniPHY IP is an interface between a memory controller and memory devices and performs read and write operations to the memory. The UniPHY IP creates the datapath between the memory device and the memory controller and user logic in various Altera devices.

The Quartus II software generates an example top-level project, consisting of an example driver, and your QDR II or QDR II+ SRAM controller custom variation. The controller instantiates an instance of the UniPHY.

The example top-level project is a fully-functional design that you can simulate, synthesize, and use in hardware. The example driver is a self-test module that issues read and write commands to the controller and checks the read data to produce the pass or fail and test-complete signals.

☞ For device families not supported by the UniPHY IP, use the Altera legacy integrated static datapath and controller MegaCore functions.

You can, alternatively, create your own memory interface datapath using the ALTDLL and ALTDQ_DQS megafunctions, available in the Quartus II software, but you then must consider all of the aspects of the design including timing analysis and design constraints.

The UniPHY IP offers the Altera PHY interface (AFI). The AFI results in a simple connection between the PHY and controller.

## Release Information

Table 1–1 provides information about this release of the QDR II and QDR II+ SRAM controllers with UniPHY.

**Table 1–1. Release Information**

| Item | Description |
|---|---|
| Version | 10.1 |
| Release Date | December 2010 |
| Ordering Codes | IP-QDRII/UNI |
| Vendor ID | 6AF7 |

# Device Family Support

IP cores provide the following levels of support for target Altera device families:

- For FPGA device support:
  - Preliminary—verified with preliminary timing models for this device
  - Final—verified with final timing models for this device
- For ASIC devices (HardCopy families)
  - HardCopy companion—verified with preliminary timing models for HardCopy companion device
  - HardCopy compilation—verifed with final timing models for HardCopy device

Table 1–2 shows the level of support offered by the QDR II and QDR II+ SRAM controllers to each of the Altera device families.

For information about supported clock rates for external memory interfaces, refer to the *External Memory Interface System Specifications* section in volume 1 of the *External Memory Interface Handbook*.

**Table 1–2. Device Family Support**

| Device Family | Support |
|---|---|
| Arria® II GX | Final |
| Arria II GZ | Preliminary |
| HardCopy® III | HardCopy companion |
| HardCopy IV | HardCopy companion |
| Stratix® III | Final |
| Stratix IV | Final |
| Stratix V | Preliminary |
| Other device families | No support |

# Features

Table 1–3 summarizes key feature support for the QDR II and QDR II+ SRAM Controllers with UniPHY.

**Table 1–3. Key Feature Support for QDR II and QDR II+ SRAM Controllers with UniPHY (Part 1 of 2)**

| Key Feature | QDR II SDRAM UniPHY | QDR II+ SDRAM UniPHY |
|---|---|---|
| High-performance controller (HPC) | — | — |
| High-performance controller II (HPC II) | ✓ | ✓ |
| Half-rate core logic and user interface | ✓ | ✓ |
| Full-rate core logic and user interface | ✓ | ✓ |
| Burst length (half-rate) | 4 | 4 |
| Burst length (full-rate) | 2 or 4 | 2 or 4 |

**Table 1–3. Key Feature Support for QDR II and QDR II+ SRAM Controllers with UniPHY (Part 2 of 2)**

| Key Feature | QDR II SDRAM UniPHY | QDR II+ SDRAM UniPHY |
|---|---|---|
| Reduced controller latency (1) (2) | ✓ | ✓ |
| Read latency | 1.5 | 2 or 2.5 |
| Maximum data width | 72 bits | 72 bits |
| ODT (in memory device) | — | ✓ |
| x36 emulation mode(3) | ✓ | ✓ |

**Notes for Table 1–3:**

(1) The maximum achievable clock rate when reduced controller latency is selected must be attained through Quatrus II software timing analysis of your complete design.

(2) Not available in Arria II GX devices.

(3) Emulation mode allows emulation of a larger memory-width interface using multiple smaller memory-width interfaces. For example, an x36 QDR II or QDR II+ interface can be emulated using two x18 interfaces.

# Unsupported Features

Table 1–4 summarizes unsupported features for the QDR II and QDR II+ SRAM Controllers with UniPHY.

**Table 1–4. Unsupported Features for the QDR II and QDR II+ SRAM Controllers with UniPHY**

| Memory Protocol | Unsupported Feature |
|---|---|
| QDR II SRAM | Cyclone III devices |
| | Cyclone IV devices |
| | Deterministic latency |
| | ECC |
| | Memory device ODT |
| | Multiple chip select |
| | VHDL support for Arria II GX, Arria II GZ, Stratix III, Stratix IV, and Stratix V devices |
| | x36 emulation mode for Stratix V devices |
| QDR II+ SRAM | Cyclone III devices |
| | Cyclone IV devices |
| | Deterministic latency |
| | ECC |
| | Multiple chip select |
| | VHDL support for Arria II GX, Arria II GZ, Stratix III, Stratix IV, and Stratix V devices |
| | x36 emulation mode for Stratix V devices |

# MegaCore Verification

Altera has carried out extensive random, directed tests with functional test coverage using industry-standard models to ensure the functionality of the QDR II and QDR II+ SRAM controllers with UniPHY.

Altera verifies that the current version of the Quartus II software compiles the previous version of each MegaCore function. The *MegaCore IP Library Release Notes and Errata* report any exceptions to this verification. Altera does not verify compilation with MegaCore function versions older than one release.

# Resource Utilization

This section lists resource utilization for the QDR II and QDR II+ SRAM controllers with UniPHY for supported device families. Resource utilizations are derived with all parameters at their default values.

Table 1–5 shows the typical resource usage of the QDR II and QDR II+ SRAM controllers with UniPHY in the current version of Quartus II software for Arria II GX devices.

**Table 1–5. Resource Utilization in Arria II GX Devices**

| PHY Rate | Memory Width (Bits) | Combinational ALUTS | Logic Registers | Memory (Bits) | M9K Blocks |
|---|---|---|---|---|---|
| Half | 9 | 620 | 701 | 0 | 0 |
| | 18 | 921 | 1122 | 0 | 0 |
| | 36 | 1534 | 1964 | 0 | 0 |
| Full | 9 | 584 | 708 | 0 | 0 |
| | 18 | 850 | 1126 | 0 | 0 |
| | 36 | 1387 | 1962 | 0 | 0 |

Table 1–6 shows the typical resource usage of the QDR II and QDR II+ SRAM controllers with UniPHY in the current version of Quartus II software for Arria II GZ, Stratix III, Stratix IV, and Stratix V devices.

**Table 1–6. Resource Utilization in Arria II GZ, Stratix III, Stratix IV, and Stratix V Devices**

| PHY Rate | Memory Width (Bits) | Combinational ALUTS | Logic Registers | Memory (Bits) | M9K Blocks |
|---|---|---|---|---|---|
| Half | 9 | 602 | 641 | 0 | 0 |
| | 18 | 883 | 1002 | 0 | 0 |
| | 36 | 1457 | 1724 | 0 | 0 |
| Full | 9 | 586 | 708 | 0 | 0 |
| | 18 | 851 | 1126 | 0 | 0 |
| | 36 | 1392 | 1962 | 0 | 0 |

# System Requirements

The QDR II and QDR II+ SRAM controllers with UniPHY are part of the MegaCore IP Library, which is distributed with the Quartus II software.

For system requirements and installation instructions, refer to *Altera Software Installation & Licensing*.

# 2. Getting Started

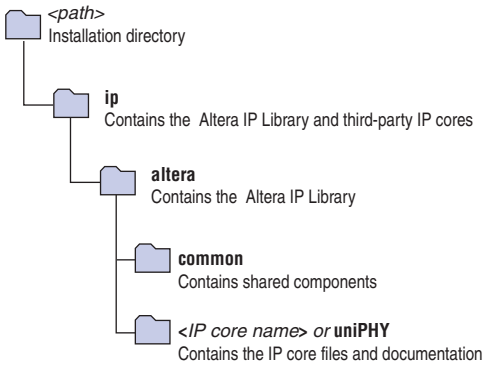
This chapter provides a general overview of the Altera IP core design flow to help you quickly get started with any Altera IP core. The Altera IP Library is installed as part of the Quartus II installation process. You can select and parameterize any Altera IP core from the library. Altera provides an integrated parameter editor that allows you to customize IP cores to support a wide variety of applications. The parameter editor guides you through the setting of parameter values and selection of optional ports. The following sections describe the general design flow and use of Altera IP cores.

## Installation and Licensing

The Altera IP Library is distributed with the Quartus II software and downloadable from the Altera website (www.altera.com).

Figure 2–1 shows the directory structure after you install an Altera IP core, where <*path*> is the installation directory. The default installation directory on Windows is **C:\altera\<***version number***>**; on Linux it is **/opt/altera<***version number***>.**

**Figure 2–1. IP core Directory Structure**




You can evaluate an IP core in simulation and in hardware until you are satisfied with its functionality and performance. Some IP cores require that you purchase a license for the IP core when you want to take your design to production. After you purchase a license for an Altera IP core, you can request a license file from the Altera Licensing page of the Altera website and install the license on your computer. For additional information, refer to *Altera Software Installation and Licensing*.

## Design Flows

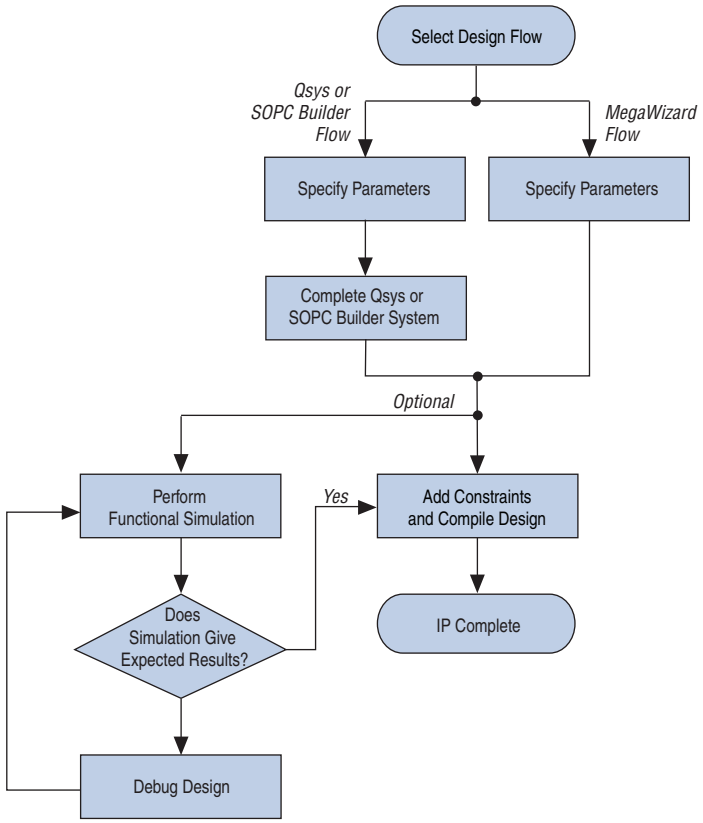
You can use the following flow(s) to parameterize Altera IP cores:

- MegaWizard Plug-In Manager Flow
- SOPC Builder Flow
- Qsys Flow

CAUTION

Altera's Qsys system integration tool is now available as beta for evaluation in the Quartus II software subscription edition version 10.1. Altera does not recommend using the beta release of Qsys in the Quartus II software version 10.1 for designs that are close to completion and are meeting design requirements. Before using Qsys, review the *Quartus II Software Version 10.1 Release Notes* and *AN 632: SOPC Builder to Qsys Migration Guidelines* for known issues and limitations. To submit general feedback or technical support on the beta release of Qsys, submit a service request through **mysupport.altera.com**. Alternatively, to submit general feedback, click **Feedback** on the Quartus II software Help menu.

**Figure 2–2. Design Flows**




The MegaWizard Plug-In Manager flow offers the following advantages:

- Allows you to parameterize an IP core variant and instantiate into an existing design
- For some IP cores, this flow generates a complete example design and testbench.

The SOPC Builder flow offer the following advantages:

- Generates simulation environment
- Allows you to integrate Altera-provided custom components
- Uses Avalon®memory-mapped (Avalon-MM) interfaces

The Qsys flow offers the following additional advantages over SOPC Builder:

- Provides visualization of hierarchical designs
- Allows greater performance through interconnect elements and pipelining
- Provides closer integration with the Quartus II software

# MegaWizard Plug-In Manager Flow

The MegaWizard Plug-In Manager flow allows you to customize your IP core and manually integrate the function into your design.

## Specifying Parameters

To specify IP core parameters with the MegaWizard Plug-In Manager, follow these steps:

1. Create a Quartus II project using the **New Project Wizard** available from the File menu.
2. In the Quartus II software, launch the **MegaWizard Plug-in Manager** from the Tools menu, and follow the prompts in the MegaWizard Plug-In Manager interface to create or edit a custom IP core variation.
3. To select a specific Altera IP core, click the IP core in the **Installed Plug-Ins** list in the MegaWizard Plug-In Manager.
4. Specify the parameters on the **Parameter Settings** pages. For detailed explanations of these parameters, refer to the "*Parameter Settings*" chapter in this document.

   Some IP cores provide preset parameters for specific applications. If you wish to use preset parameters, click the arrow to expand the **Presets** list, select the desired preset, and then click **Apply**. To modify preset settings, in a text editor edit the **<*installation directory*>\ip\altera\uniphy\lib\<*IP core*>.qprs** file.

5. If the IP core provides a simulation model, specify appropriate options in the wizard to generate a simulation model.

   Altera IP supports a variety of simulation models, including simulation-specific IP functional simulation models and encrypted RTL models, and plain text RTL models. These are all cycle-accurate models. The models allow for fast functional simulation of your IP core instance using industry-standard VHDL or Verilog HDL simulators. For some cores, only the plain text RTL model is generated, and you can simulate that model.

   For more information about functional simulation models for Altera IP cores, refer to *Simulating Altera Designs* in volume 3 of the *Quartus II Handbook*.

   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.

6. If the parameter editor includes **EDA** and **Summary** tabs, follow these steps:
   a. Some third-party synthesis tools can use a netlist that contains the structure of an IP core but no detailed logic to optimize timing and performance of the design containing it. To use this feature if your synthesis tool and IP core support it, turn on **Generate netlist**.
   b. On the **Summary** tab, if available, select the files you want to generate. A gray checkmark indicates a file that is automatically generated. All other files are optional.

   ☞ If file selection is supported for your IP core, after you generate the core, a generation report (*<variation name>***.html)** appears in your project directory. This file contains information about the generated files.

7. Click the **Finish** button, the parameter editor generates the top-level HDL code for your IP core, and a simulation directory which includes files for simulation.

   ☞ The **Finish** button may be unavailable until all parameterization errors listed in the messages window are corrected.

8. Click **Yes** if you are prompted to add the Quartus II IP File (**.qip**) to the current Quartus II project. You can also turn on **Automatically add Quartus II IP Files to all projects**.

You can now integrate your custom IP core instance in your design, simulate, and compile. While integrating your IP core instance into your design, you must make appropriate pin assignments. You can create virtual pin to avoid making specific pin assignments for top-level signals while you are simulating and not ready to map the design to hardware.

For some IP cores, the generation process also creates a complete example design in the *<variation_name>***_example_design_fileset/example_project/** directory. This example demonstrates how to instantiate and connect the IP core.

☞ For information about the Quartus II software, including virtual pins and the MegaWizard Plug-In Manager, refer to Quartus II Help.

## Simulate the IP Core

You can simulate your IP core variation with the functional simulation model and the testbench or example design generated with your IP core. The functional simulation model and testbench files are generated in a project subdirectory. This directory may also include scripts to compile and run the testbench.

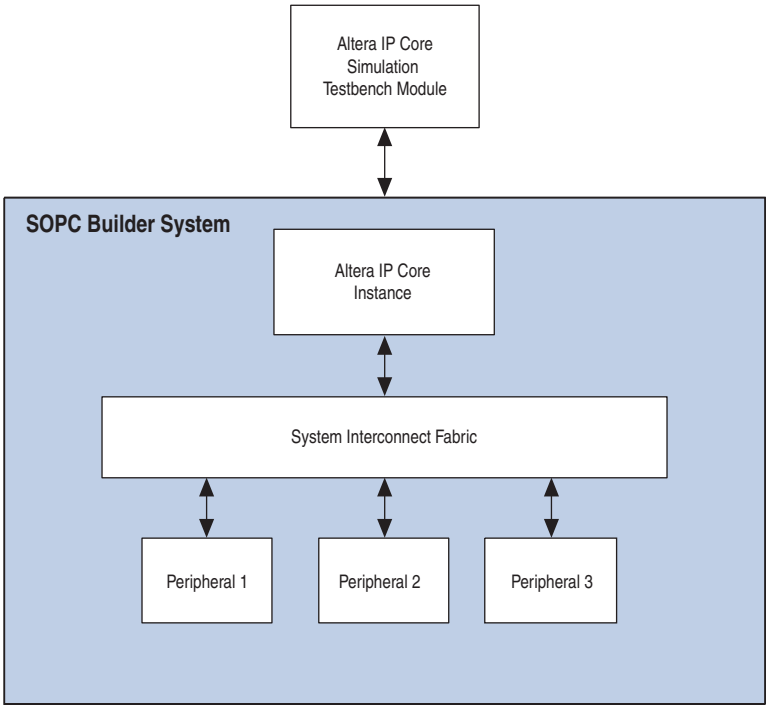
For a complete list of models or libraries required to simulate your IP core, refer to the scripts provided with the testbench.

For more information about simulating Altera IP cores, refer to *Simulating Altera Designs* in volume 3 of the *Quartus II Handbook*.

# SOPC Builder Design Flow

You can use SOPC Builder to build a system that includes your customized IP core. You easily can add other components and quickly create an SOPC Builder system. SOPC Builder automatically generates HDL files that include all of the specified components and interconnections. SOPC Builder defines default connections, which you can modify. The HDL files are ready to be compiled by the Quartus II software to produce output files for programming an Altera device. SOPC Builder generates a simulation testbench module for supported cores that includes basic transactions to validate the HDL files. Figure 2–3 shows a block diagram of an example SOPC Builder system.

**Figure 2–3. SOPC Builder System**




For more information about system interconnect fabric, refer to the *System Interconnect Fabric for Memory-Mapped Interfaces* and *System Interconnect Fabric for Streaming Interfaces* chapters in the *SOPC Builder User Guide* and to the *Avalon Interface Specifications*.

For more information about SOPC Builder and the Quartus II software, refer to the *SOPC Builder Features* and *Building Systems with SOPC Builder* sections in the *SOPC Builder User Guide* and to Quartus II Help.

## Specify Parameters

To specify IP core parameters in the SOPC Builder flow, follow these steps:

1. Create a new Quartus II project using the **New Project Wizard** available from the File menu.
2. On the Tools menu, click **SOPC Builder**.
3. For a new system, specify the system name and language.
4. On the **System Contents** tab, double-click the name of your IP core to add it to your system. The relevant parameter editor appears.
5. Specify the required parameters in the parameter editor. For detailed explanations of these parameters, refer to the *"Parameter Settings"* chapter in this document.

   Some IP cores provide preset parameters for specific applications. If you wish to use preset parameters, click the arrow to expand the **Presets** list, select the desired preset, and then click **Apply**. To modify preset settings, in a text editor edit the <*installation directory*>**\ip\altera\uniphy\lib**<*IP core*>**.qprs** file.

   If your design includes external memory interface IP cores, you must turn on **Generate power of two bus widths** on the **PHY Settings** tab when parameterizing those cores.

6. Click **Finish** to complete the IP core instance and add it to the system.

   The **Finish** button may be unavailable until all parameterization errors listed in the messages window are corrected.

## Complete the SOPC Builder System

To complete the SOPC Builder system, follow these steps:

1. Add and parameterize any additional components. Some IP cores include a complete SOPC Builder system design example.
2. Use the Connection panel on the **System Contents** tab to connect the components.
3. By default, clock names are not displayed. To display clock names in the **Module Name** column and the clocks in the **Clock** column in the **System Contents** tab, click **Filters** to display the **Filters** dialog box. In the **Filter** list, click **All.**
4. If you intend to simulate your SOPC builder system, on the **System Generation** tab, turn on **Simulation** to generate simulation files for your system.
5. Click **Generate** to generate the system. SOPC Builder generates the system and produces the <*system name*>**.qip** file that contains the assignments and information required to process the IP core or system in the Quartus II Compiler.
6. In the Quartus II software, click **Add/Remove Files in Project** and add the **.qip** file to the project.
7. Compile your design in the Quartus II software.

## Simulate the System

During system generation, you can specify whether SOPC Builder 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 and plain-text RTL design files that describe your system in the ModelSim simulation software.

For information about the latest Altera-supported simulation tools, refer to the *Quartus II Software Release Notes*.

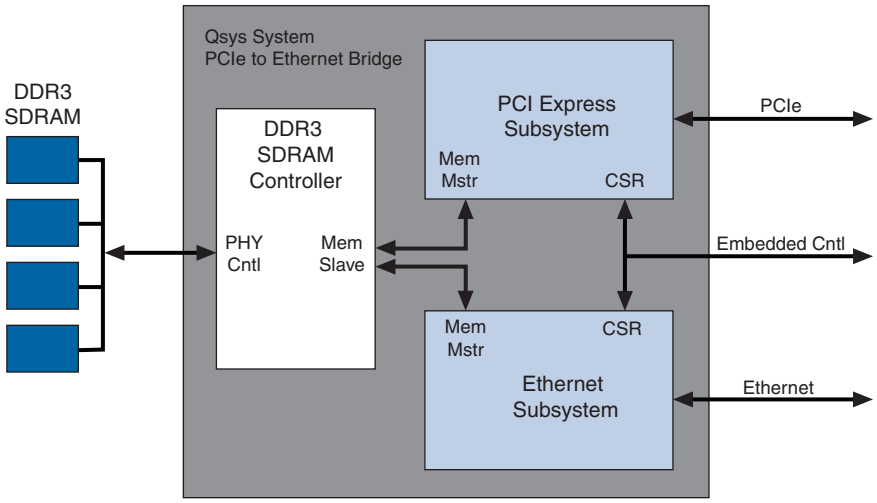
For information about simulating SOPC Builder systems, refer to the *SOPC Builder User Guide* and *AN 351: Simulating Nios II Embedded Processor Designs*.

For general information about simulating Altera IP cores, refer to *Simulating Altera Designs* in volume 3 of the *Quartus II Handbook*.

# Qsys System Integration Tool Design Flow

You can use the Qsys system integration tool to build a system that includes your customized IP core. You easily can add other components and quickly create a Qsys system. Qsys automatically generates HDL files that include all of the specified components and interconnections. In Qsys, you specify the connections you want. The HDL files are ready to be compiled by the Quartus II software to produce output files for programming an Altera device. Qsys generates Verilog HDL simulation models for the IP cores that comprise your system. Figure 2–4 shows a high level block diagram of an example Qsys system.

**Figure 2–4. Example Qsys System**



For more information about the Qsys system interconnect, refer to the *Qsys Interconnect* chapter in volume 1 of the *Quartus II Handbook* and to the *Avalon Interface Specifications*.

For more information about the Qsys tool and the Quartus II software, refer to the *System Design with Qsys* section in volume 1 of the *Quartus II Handbook* and to Quartus II Help.

## Specify Parameters

To specify parameters for your IP core using the Qsys flow, follow these steps:

1. Create a new Quartus II project using the **New Project Wizard** available from the File menu.
2. On the Tools menu, click **Qsys (Beta)**.
3. On the **System Contents** tab, double-click the name of your IP core to add it to your system. The relevant parameter editor appears.
4. Specify the required parameters in all tabs in the Qsys tool. For detailed explanations of these parameters, refer to the *"Parameter Settings"* chapter in this document.

   If your design includes external memory interface IP cores, you must turn on **Generate power of two bus widths** on the **PHY Settings** tab when parameterizing those cores.

   Some IP cores provide preset parameters for specific applications. If you wish to use preset parameters, click the arrow to expand the **Presets** list, select the desired preset, and then click **Apply**. To modify preset settings, in a text editor edit the ***<installation directory>*\ip\altera\uniphy\lib*<IP core>*.qprs** file.

5. Click **Finish** to complete the IP core instance and add it to the system.

   The **Finish** button may be unavailable until all parameterization errors listed in the messages window are corrected.

## Complete the Qsys System

To complete the Qsys system, follow these steps:

1. Add and parameterize any additional components.
2. Connect the components using the Connection panel on the **System Contents** tab.
3. In the **Export As** column, enter the name of any connections that should be a top-level Qsys system port. If the **Export As** column is not present, click the **Project Settings** tab and turn off **Use SOPC Builder port naming**.
4. If you intend to simulate your Qsys system, on the **Generation** tab, turn on one or more options under **Simulation** to generate desired simulation files.
5. If your system is not part of a Quartus II project and you want to generate synthesis RTL files, turn on **Create synthesis RTL files**.

6. Click **Generate** to generate the system. Qsys generates the system and produces the <*system name*>**.qip** file that contains the assignments and information required to process the IP core or system in the Quartus II Compiler.
7. In the Quartus II software, click **Add/Remove Files in Project** and add the **.qip** file to the project.
8. Compile your project in the Quartus II software.

### Simulate the System

During system generation, Qsys generates a functional simulation model—or example design that includes a testbench—which you can use to simulate your system in any Altera-supported simulation tool.

For information about the latest Altera-supported simulation tools, refer to the *Quartus II Software Release Notes*.

For general information about simulating Altera IP cores, refer to *Simulating Altera Designs* in volume 3 of the *Quartus II Handbook*.

For information about simulating Qsys systems, refer to the *System Design with Qsys* section in volume 1 of the *Quartus II Handbook*.

## HardCopy Migration Design Guidelines

If you intend to target your design to a HardCopy® device, ensure you use the following design guidelines:

- On the **General Settings** page of the **DDR2 SDRAM Controller with UniPHY** or **DDR3 SDRAM Controller with UniPHY** MegaWizard, turn on **HardCopy Compatibility Mode**, and then specify whether the **Reconfigurable PLL Location** is **Top_Bottom** or **Left_Right**.

  Altera recommends that you set the **Reconfigurable PLL Location** to the same side as your memory interface.

  When turned on, the **HardCopy Compatibility Mode** option enables run-time reconfiguration for all phase-locked loops (PLLs) and delay-locked loops (DLLs) instantiated in memory interfaces that are configured in PLL and DLL masters, and brings the necessary reconfiguration signals to the top level of the design.

  "Top-Level HardCopy Migration Signals" on page 6–12 lists the top-level signals generated for HardCopy migration.

- Enable run-time reconfiguration mode for all PLLs and DLLs instantiated in interfaces that are configured in PLL and DLL slaves.

For information about PLL megafunctions, refer to the *Phase-Locked Loop (ALTPLL) Megafunction User Guide* and the *Phase-Locked Loops Reconfiguration (ALTPLL_RECONFIG) Megafunctions User Guide*. For information about DLL megafunctions, refer to the *ALTDLL and ALTDQ_DQS Megafunctions User Guide*.

- Ensure that you place all memory interface pins close together. If, for example, address pins are located far away from data pins, closing timing might be difficult.

You can use the example top-level project that is generated when you turn on **HardCopy Migration** as a guide to help you connect the necessary signals in your design.

## Differences in UniPHY IP Generated with HardCopy Migration Support

When you generate a UniPHY memory interface for HardCopy device support, certain features in the IP are enabled that do not exist when you generate the IP core for only the FPGA. This section discusses those additional enabled features.

### ROM Loader for Designs Using Nios II Sequencer

An additional ROM loader is intantiated in the design for UniPHY designs that use the Nios II sequencer. The Nios II sequencer instruction code resides in RAM on either the HardCopy or FPGA device.

When you target only an FPGA device, the RAM is initialized when the device is programmed; however, HardCopy devices are not programmed and therefore the RAM cannot be initialized in this fashion. Instead, the Nios II sequencer instruction code must be stored in an external, non-volatile, ROM that loads the Nios II sequencer RAM through a ROM loader. You must attach the ROM loader to the appropriate pins connected to the external non-volatile ROM.

Table 2–1 summarizes the ports exposed at the top level of the PHY+Controller wrapper to expose the ROM loader utilized by the Nios II-based sequencer within the DDR2 or DDR3 PHY.

**Table 2–1. Top-level Ports that Connect to External ROM for Loading Nios II Code Memory (Part 1 of 2)**

| Port Name | Direction | Description |
|---|---|---|
| `hc_rom_config_clock` | Input | Write clock for the ROM loader. This clock is the write clock for the Nios II code memory. |
| `hc_rom_config_datain` | Input | Data input from external ROM. |
| `hc_rom_config_rom_data_ready` | Input | Asserts to the code memory loader that the word of memory is ready to be loaded. |
| `hc_rom_config_init` | Input | Signals that the Nios II code memory is being loaded from the external ROM. |
| `hc_rom_config_init_busy` | Output | Remains asserted throughout initialization and becomes inactive when initialization is complete. `soft_reset_n` can be issued after `hc_rom_config_init_busy` is deasserted. |

**Table 2–1. Top-level Ports that Connect to External ROM for Loading Nios II Code Memory (Part 2 of 2)**

| Port Name | Direction | Description |
|---|---|---|
| hc_rom_config_rom_rden | Output | Read-enable signal that connects to the external ROM. |
| hc_rom_config_rom_address | Output | ROM address that connects to the external ROM. |

## PLL/DLL Run-time Reconfiguration

The PLLs and DLLs in the HardCopy design have run-time reconfiguration enabled—provided that they are not in PLL/DLL slave mode.

When the PLLs and DLLs are generated with reconfiguration enabled, there are extra signals that must be connected and driven by user logic. In the example design generated during IP core generation, the PLL/DLL reconfiguration signals are brought to the top level and connected to constants, as shown in Figure 2–5.

For information about PLL megafunctions and reconfiguration, refer to the *Phase-Locked Loop (ALTPLL) Megafunction User Guide* and the *Phase-Locked Loops Reconfiguration (ALTPLL_RECONFIG) Megafunctions User Guide*.

**Figure 2–5. HardCopy UniPHY Example Design**

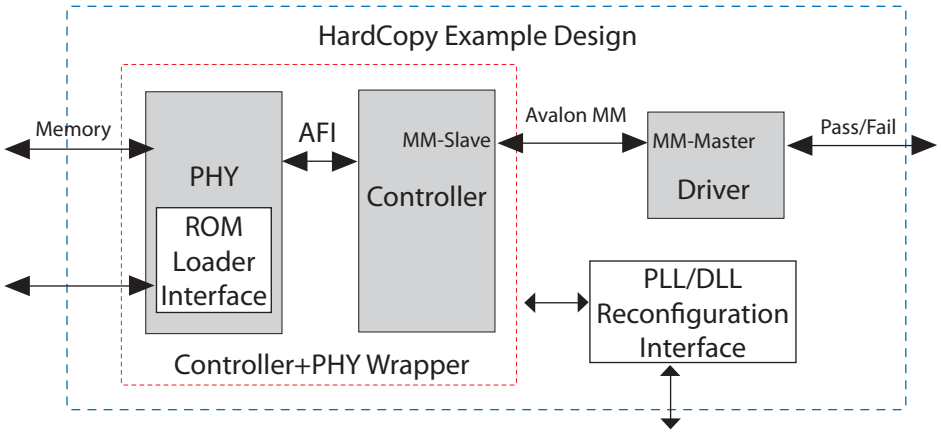



Table 2–2 summarizes the DLL reconfiguration ports exposed at the top level of the Controller+PHY.

**Table 2–2. DLL Reconfiguration Ports Exposed at Top-Level of Controller+PHY Wrapper (Part 1 of**

| Port Name | Direction | Description |
|---|---|---|
| hc_dll_config_dll_offset_ctrl_addnsub | Input | Addition/subtraction control port for the DLL. This port controls if the delay-offset setting on hc_dll_config_dll_offset_ctrl_offset is added or subtracted. |

**Table 2–2. DLL Reconfiguration Ports Exposed at Top-Level of Controller+PHY Wrapper (Part 2 of**

| Port Name | Direction | Description |
|---|---|---|
| hc_dll_config_dll_offset_ctrl_offset | Input | Offset input setting for the PLL. This is a Gray-coded offset that is added or subtracted from the current value of the DLL's delay chain. |
| hc_dll_config_dll_offset_ctrl_offsetctrlout | Output | The registered and Gray-coded value of the current delay-offset setting. |

Table 2–3 summarizes the ports exposed at the top level of the Controller and PHY wrapper to allow PLL reconfiguration.

**Table 2–3. PLL Reconfiguration Ports Exposed at the Top-Level of Controller+PHY Wrapper**

| Port Name | Direction | Description |
|---|---|---|
| hc_pll_config_configupdate | Input | Control signal to enable PLL reconfiguration. (Applies to RLDRAMII and QDRII only, the phase reconfiguration feature for DDR2/3 is included in the CSR port.) |
| hc_pll_config_phasecounterselect | Input | Specifies the counter select for dynamic phase adjustment. (Applies to RLDRAMII and QDR II only.) |
| hc_pll_config_phasestep | Input | Specifies the phase step for dynamic phase shifting. (Applies to RLDRAMII and QDR II only.) |
| hc_pll_config_phaseupdown | Input | Specifies if the phase adjustment should be up or down. (Applies to RLDRAMII and QDR II only.) |
| hc_pll_config_scanclk | Input | PLL reconfiguration scan chain clock. |
| hc_pll_config_scanclkena | Input | Clock enable port of the hc_pll_config_scanclk clock. |
| hc_pll_config_scandata | Input | Serial input data for the PLL reconfiguration scan chain. |
| hc_pll_config_phasedone | Output | When asserted, this signal indicates to core logic that phase adjustment is completed and that the PLL is ready to act on a possible second adjustment pulse. |
| hc_pll_config_scandataout | Output | The data output of the serial scan chain. |
| hc_pll_config_scandone | Output | Asserted when the scan chain write operation is in progress and is deasserted when the write operation is complete. |

To facilitate placement and timing closure and help compensate for PLLs adjacent to I/Os and vertical I/O overhang issues that can occur when targeting HardCopy III and HardCopy IV devices, an additional pipeline stage is added to the write path in the RTL when you turn on **HardCopy Compatibility**. The additional pipeline stage is added in all cases, except when CAS write latency equals 2 (for DDR3) or CAS latency equals 3 (for DDR2), where the additional pipeline stage is not required to meet timing requirements. The additional pipeline stage does not affect the overall latency of the controller.

For information about HardCopy issues such as vertical I/O overhang, PLLs adjacent to I/Os, and timing closure, refer to HardCopy III Device I/O Features in the *HardCopy III Device Handbook, Volume 1*, and HardCopy IV Device I/O Features in the *HardCopy IV Device Handbook, Volume 1*.

# Generated Files

When you complete the IP generation flow, there are generated files created in your project directory. The directory structure created varies somewhat, depending on the tool used to parameterize and generate the IP.

The PLL parameters are statically defined in the <*variation_name*>**_parameters.tcl** at generation time. To ensure timing constraints and timing reports are correct, when you use the GUI to make changes to the PLL component, apply those changes to the PLL parameters in this file.

## MegaWizard Plug-in Manager Flow

The tables in this section list the generated directory structure and key files of interest to users, resulting from the MegaWizard Plug-in Manager flow.

### Synthesis

Table 2–4 lists the generated directory structure and key files created by the synthesis flow with the MegaWizard Plug-in Manager.

**Table 2–4. Generated Directory Structure and Key Files—MegaWizard Plug-In Manager Synthesis Flow**

| Directory | File Name | Description |
|---|---|---|
| <*working_dir*>/ | <*variation_name*>.qip | QIP file which refers to all generated files in the synthesis fileset. |
| <*working_dir*>/ | <*variation_name*>.v (for Verilog), or <*variation_name*>.vhd (for VHDL) | Top-level wrapper for synthesis files. |
| <*working_dir*>/<*variation_name*>/ | <*variation_name*>_<*stamp*>.v (1) | UniPHY top-level wrapper. |
| <*working_dir*>/<*variation_name*>/ | <*variation_name*>_<*stamp*>_*.v (1) | UniPHY Verilog RTL files. |
| <*working_dir*>/<*variation_name*>/ | <*variation_name*>_<*stamp*>_*.sv (1) | UniPHY SystemVerilog RTL files. |
| <*working_dir*>/<*variation_name*>/ | <*variation_name*>_<*stamp*>.sdc (1) | Synopsys constraints file. |
| <*working_dir*>/<*variation_name*>/ | <*variation_name*>_<*stamp*>.ppf (1) | Pin Planner file. |
| <*working_dir*>/<*variation_name*>/ | <*variation_name*>_<*stamp*>_pin_assignments.tcl (1) | Pin constraints script to be run after synthesis. |
| <*working_dir*>/<*variation_name*>/ | <*variation_name*>_<*stamp*>_*.tcl (1) | Other Tcl scripts. |
| <*working_dir*>/<*variation_name*>/ | <*variation_name*>_<*stamp*>_readme.txt (1) | Readme text file. |

**Note to Table 2–4:**

(1) <*stamp*> is a unique identifier determined by the MegaWizard Plug-in Manager at generation time.

## Simulation

Table 2–5 lists the generated directory structure and key files created by the Verilog simulation flow with the MegaWizard Plug-in Manager.

**Table 2–5. Generated Directory Structure and Key Files—MegaWizard Plug-In Manager Simulation Flow (Verilog)**

| Directory | File Name | Description |
|---|---|---|
| <*working_dir*>/<*variation_name*>_sim/ | <*variation_name*>.v (for Verilog), or <*variation_name*>.vho (for VHDL) | UniPHY top-level wrapper. |
| <*working_dir*>/<*variation_name*>_sim/ | <*variation_name*>_*.v | UniPHY Verilog RTL files. |
| <*working_dir*>/<*variation_name*>_sim/ | <*variation_name*>_*.sv | UniPHY SystemVerilog RTL files. |
| <*working_dir*>/<*variation_name*>_sim/ | <*variation_name*>_readme.txt | Readme text file. |

Table 2–6 lists the generated directory structure and key files created by the VHDL simulation flow with the MegaWizard Plug-in Manager.

**Table 2–6. Generated Directory Structure and Key Files—MegaWizard Plug-In Manager Simulation Flow (VHDL)**

| Directory | File Name | Description |
|---|---|---|
| <*working_dir*>/<*variation_name*>_sim/ | <*variation_name*>.vho | UniPHY VHDL top-level module. |
| <*working_dir*>/<*variation_name*>_sim/ | <*variation_name*>_*.vhd<br><*variation_name*>_*.vho | UniPHY simulation VHDL files. |
| <*working_dir*>/<*variation_name*>_sim/ | vhdl_files.txt | File list text file. |

## Example Design

Table 2–7 lists the generated directory structure and key files created for the example design with the MegaWizard Plug-in Manager

**Table 2–7. Generated Directory Structure and Key Files—MegaWizard Plug-In Manager Example Design (Part 1 of 2)**

| Directory | File Name‘ | Description |
|---|---|---|
| <*working_dir*>/<*variation_name*>_example_design_fileset/ | <*variation_name*>.qip | QIP which refers to UniPHY RTL in this fileset. This is distinct from ../<*variation_name*>.qip. This file is included automatically in the example project. |
| <*working_dir*>/<*variation_name*>_example_design_fileset/ | <*variation_name*>.v | UniPHY top-level wrapper. |
| <*working_dir*>/<*variation_name*>_example_design_fileset | <*variation_name*>_*.v | UniPHY Verilog RTL files. |
| <*working_dir*>/<*variation_name*>_example_design_fileset | <*variation_name*>_*.sv | UniPHY SystemVerilog RTL fies. |
| <*working_dir*>/<*variation_name*>_example_design_fileset/ | <*variation_name*>.sdc | Synopsys constraints file. |

**Table 2–7. Generated Directory Structure and Key Files—MegaWizard Plug-In Manager Example Design (Part 2 of 2)**

| Directory | File Name‘ | Description |
|---|---|---|
| <*working_dir*>/<*variation_name*>_example_design_fileset/ | <*variation_name*>.ppf | Pin Planner file. |
| <*working_dir*>/<*variation_name*>_example_design_fileset/ | <*variation_name*>_pin_assignments.tcl | Pin constraints script to be run after synthesis. |
| <*working_dir*>/<*variation_name*>_example_design_fileset/ | <*variation_name*>_*.tcl | Other Tcl scripts. |
| <*working_dir*>/<*variation_name*>_example_design_fileset/ | <*variation_name*>_readme.txt | Readme text file. |
| <*working_dir*>/<*variation_name*>_example_design_fileset/example_project/ | <*variation_name*>_example_top.qpf | Example design project file. |
| <*working_dir*>/<*variation_name*>_example_design_fileset/example_project/ | <*variation_name*>_example_top.qsf | Example design project settings file. |
| <*working_dir*>/<*variation_name*>_example_design_fileset/example_project/ | <*variation_name*>_example_top.v | Top-level wrapper including UniPHY, traffic generator, and memory model. |
| <*working_dir*>/<*variation_name*>_example_design_fileset/example_project/ | <*variation_name*>_*.v | Other example design Verilog RTL files. |
| <*working_dir*>/<*variation_name*>_example_design_fileset/example_project/ | <*variation_name*>_*.sv | Other example design SystemVerilog RTL files. |
| <*working_dir*>/<*variation_name*>_example_design_fileset/example_project/ | <*prefix*>_mem_model.sv (1) | Generic memory model. |
| <*working_dir*>/<*variation_name*>_example_design_fileset/rtl_sim/ | <*variation_name*>_example_top_tb.v | Top-level test bench. |

**Note to Table 2–7:**

(1) <*prefix*> varies depending on protocol and type of memory model.

## SOPC Builder Flow

Table 2–8 lists the generated directory structure and key files created by the SOPC Builder flow.

**Table 2–8. Generated Directory Structure and Key Files—SOPC Builder Flow (Part 1 of 2)**

| Directory | File Name‘ | Description |
|---|---|---|
| <*working_dir*>/ | <*system_name*>.qip | QIP which refers to all generated files in the SOPC Builder project. |
| <*working_dir*>/ | <*system_name*>.v | SOPC Builder system top-level wrapper. |
| <*working_dir*>/ | <*core_name*>_<*stamp*>.v (1) | UniPHY top-level wrapper. |
| <*working_dir*>/ | <*core_name*>_<*stamp*>_*.v (1) | UniPHY Verilog RTL files. |
| <*working_dir*>/ | <*core_name*>_<*stamp*>_*.sv (1) | UniPHY SystemVerilog RTL files. |
| <*working_dir*>/ | <*core_name*>_<*stamp*>.sdc (1) | Synopsys constraints file. |

**Table 2–8. Generated Directory Structure and Key Files—SOPC Builder Flow (Part 2 of 2)**

| Directory | File Name‘ | Description |
|---|---|---|
| <*working_dir*>/ | <*core_name*>_<*stamp*>.ppf (1) | Pin Planner file. |
| <*working_dir*>/ | <*core_name*>_<*stamp*>_pin_assignments.tcl (1) | Pin constraints script to be run after synthesis. |
| <*working_dir*>/ | <*core_name*>_<*stamp*>_*.tcl (1) | Other Tcl scripts. |
| <*working_dir*>/ | <*core_name*>_<*stamp*>_readme.txt (1) | Readme text file. |
| <*working_dir*>/ | *Other IP core files.* | Other IP cores. |

**Note to Table 2–8:**

(1) <*stamp*> is a unique identifier determined by SOPC Builder at generation time.

## Qsys Flow

The tables in this section list the generated directory structure and key files of interest to users, resulting from the Qsys flow

### Synthesis

Table 2–9 lists the generated directory structure and key files created by the synthesis flow with Qsys.

**Table 2–9. Generated Directory Structure and Key Files—Qsys Synthesis Flow (Part 1 of 2)**

| Directory | File Name | Description |
|---|---|---|
| <*working_dir*>/<*system_name*>/synthesis/ | <*system_name*>.qip | QIP which refers to all generated files in the Qsys system synthesis fileset. |
| <*working_dir*>/<*system_name*>/synthesis/ | <*system_name*>.v | Qsys system top-level wrapper. |
| <*working_dir*>/<*system_name*>/synthesis/submodules/ | <*core_name*>_<*stamp*>.v (1) | UniPHY top-level wrapper. |
| <*working_dir*>/<*system_name*>/synthesis/submodules/ | <*core_name*>_<*stamp*>_*.v (1) | UniPHY Verilog RTL files. |
| <*working_dir*>/<*system_name*>/synthesis/submodules/ | <*core_name*>_<*stamp*>_*.sv (1) | UniPHY SystemVerilog RTL files. |
| <*working_dir*>/<*system_name*>/synthesis/submodules/ | <*core_name*>_<*stamp*>.sdc (1) | Synopsys constraints file. |
| <*working_dir*>/<*system_name*>/synthesis/submodules/ | <*core_name*>_<*stamp*>.ppf (1) | Pin Planner file. |
| <*working_dir*>/<*system_name*>/synthesis/submodules/ | <*core_name*>_<*stamp*>_pin_assignments.tcl (1) | Pin constraints script to be run after synthesis. |
| <*working_dir*>/<*system_name*>/synthesis/submodules/ | <*core_name*>_<*stamp*>_*.tcl (1) | Other Tcl scripts. |
| <*working_dir*>/<*system_name*>/synthesis/submodules/ | <*core_name*>_<*stamp*>_readme.txt (1) | Readme text file. |

**Table 2–9. Generated Directory Structure and Key Files—Qsys Synthesis Flow (Part 2 of 2)**

| Directory | File Name | Description |
|---|---|---|
| <*working_dir*>/<*system_name*>/synthesis/submodules/ | *Other IP core files* | Other IP core files. |
| **Note to Table 2–9**<br>(1) <*stamp*> is a unique identifier created by Qsys during generation. | | |

## Verilog Simulation

Table 2–10 lists the generated directory structure and key files created by the Verilog HDL simulation flow with Qsys.

**Table 2–10. Generated Directory Structure and Key Files—Qsys Verilog Simulation**

| Directory | File Name | Description |
|---|---|---|
| <*working_dir*>/<*system_name*>/sim_verilog/ | <*system_name*>.v | Qsys system top-level wrapper. |
| <*working_dir*>/<*system_name*>/sim_verilog/submodules/ | <*core_name*>_<*stamp*>.v | UniPHY top-level wrapper. |
| <*working_dir*>/<*system_name*>/sim_verilog/submodules/ | <*core_name*>_<*stamp*>_*.v | UniPHY Verilog RTL files. |
| <*working_dir*>/<*system_name*>/sim_verilog/submodules/ | <*core_name*>_<*stamp*>_*.sv | UniPHY SystemVerilog RTL files. |
| <*working_dir*>/<*system_name*>/sim_verilog/submodules/ | <*core_name*>_<*stamp*>_readme.txt (1) | Readme text file. |
| <*working_dir*>/<*system_name*>/sim_verilog/submodules/ | *Other IP core files* | Other IP core files. |
| **Note for Table 2–10:**<br>(1) <*stamp*> is a unique identifier created by Qsys during generation. | | |

## VHDL Simulation

Table 2–11 lists the generated directory structure and key files created by the VHDL simulation flow with Qsys.

**Table 2–11. Generated Directory Structure and Key Files—Qsys VHDL Simulation**

| Directory | File Name | Description |
|---|---|---|
| <*working_dir*>/<*system_name*>/sim_vhdl/ | <*system_name*>.vhd | Qsys system top-level wrapper. |
| <*working_dir*>/<*system_name*>/sim_vhdl/submodules/ | <*core_name*>_<*stamp*>.vho (1) | UniPHY VHDL top-level module. |
| <*working_dir*>/<*system_name*>/sim_vhdl/submodules/ | <*core_name*>_<*stamp*>_*.vhd<br><*core_name*>_<*stamp*>_*.vho (1) | UniPHY VHDL simulation files. |
| <*working_dir*>/<*system_name*>/sim_vhdl/submodules/ | vhdl_files.txt | File list text file. |
| **Note to Table 2–11:**<br>(1) <*stamp*> is a unique identifier created by Qsys during generation. | | |

# 3. Parameter Settings

This chapter describes the QDR II and QDR II+ SRAM Controller with UniPHY IP core parameters that you can set in the GUI.

## General Settings

The **General Settings** tab allows you to configure the following parameter settings.

### Clocks

Table 3–1 describes the clock settings.

**Table 3–1. Clock Settings**

| Parameter | Description |
|---|---|
| Memory clock frequency | The frequency of the clock that drives the memory device. |
| PLL reference clock frequency | The frequency of the clock that feeds the PLL. |
| Full or half rate on Avalon-MM interface | Defines the width of the data bus on the Avalon-MM interface. A setting of **Full** results in a width twice the memory data width. A setting of **Half** results in a width of four times the memory data width. |
| Additional address/command clock phase | Increases or decreases the phase shift of the address/command clock. The base phase shift center-aligns the address/command clock at the memory device. In some circumstances, you can improve timing by increasing or decreasing the phase shift. |

### Advanced PHY Settings

Table 3–2 describes the advanced PHY settings.

**Table 3–2. Advanced PHY Settings**

| Parameter | Description |
|---|---|
| Generate power-of-2 bus widths | Rounds down the Avalon-MM side data bus to the nearest power of 2. |
| Maximum Avalon-MM burst length | Specifies the maximum burst length on the Avalon-MM bus. |
| I/O standard | Specifies the I/O standard voltage. |
| Master for PLL/DLL sharing | Causes UniPHY to instantiate its own PLL and DLL. All of the PLL clocks and DLL delay values are exported for use by other identical UniPHY cores that have this option turned on. |

**Table 3–2. Advanced PHY Settings**

| Parameter | Description |
|---|---|
| Master for OCT control block | Causes UniPHY to instantiate the required OCT control block. When this parameter is turned off, you must instantiate this block and connect the termination control bus signals to the PHY, or share an OCT control block from another UniPHY instantiation that is in master mode. |
| Hardcopy compatibility mode | Causes the generated UniPHY memory interface to have all required HardCopy compatibility options enabled. For example, PLLs and DLLs will have their reconfiguration ports exposed. |
| **Example Testbench Simulation options** | |
| Skip memory initialization | Causes the example testbench to skip the memory initialization sequence. This setting does not change the generated RTL, but can speed up simulation. |

## Topology

Table 3–3 describes the topology settings.

**Table 3–3. Topology Settings**

| Parameter | Description |
|---|---|
| Device width | Specifies the number of devices used for width expansion. |
| Device depth | Specifies the number of devices (ranks) used for depth expansion. |

## Controller Settings

Table 3–4 describes the controller settings.

**Table 3–4. Controller Settings**

| Parameter | Description |
|---|---|
| Controller latency | Specifies the number of clock cycles required for a request to pass through an idling controller. |

# Memory Parameters

The **Memory Parameters** tab allows you to configure memory device parameters. You can enter parameters manually from the manufacturer's device data sheet, or you can populate the fields automatically by selecting the required device from the list of presets.

Table 3–5 describes the memory parameters.

**Table 3–5. Memory Parameters (Part 1 of 2)**

| Parameter | Description |
|---|---|
| Address width | The width of the address bus on the memory device. |
| Data width | The width of the data bus on the memory device. |

**Table 3–5. Memory Parameters (Part 2 of 2)**

| Parameter | Description |
|---|---|
| Data-mask width | The width of the data-mask on the memory device, |
| CQ width | The width of the CQ (read strobe) bus on the memory device. |
| K width | The width of the K (write strobe) bus on the memory device. |
| Burst length | The burst length supported by the memory device. |

# Memory Timing

The **Memory Timing** tab allows you to configure memory device timing parameters. You can enter timing parameters manually from the manufacturer's device data sheet, or you can populate the fields automatically by selecting the required device from the list of presets.

Table 3–6 describes the memory timing parameters.

**Table 3–6. Memory Timing Parameters**

| Parameters | Description |
|---|---|
| tWL (cycles) | The write latency. |
| tRL (cycles) | The read latency. |
| tSA | The address and control setup to K clock rise. |
| tHA | The address and control hold after K clock rise. |
| tSD | The data setup to clock (K/K#) rise. |
| tHD | The data hold after clock (K/K#) rise. |
| tCQD | Echo clock high to data valid. |
| tCQDOH | Echo clock high to data invalid. |
| Internal jitter | The QDRII/II+ internal jitter. |
| TCQHCQnH | The CQ clock rise to CQn clock rise (rising edge to rising edge). |
| TKHKnH | The K clock rise to Kn clock rise (rising edge to rising edge). |

# Board Settings

The **Board Settings** tab allows you to enter values derived from board simulation.

## Intersymbol Interference

Intersymbol interference (ISI), occurs when a signal is distorted due to the interference of one symbol with subsequent symbols. Typically, ISI is greater at device depths greater than 1 because there are multiple stubs causing reflections. The advanced I/O timing analysis capabilities of the Quartus II software already includes ISI effects for device depth of 1.

Table 3–7 describes the intersymbol interference settings.

**Table 3–7. Intersymbol Interference Settings**

| Parameter | Description |
|---|---|
| Address/command eye reduction (setup) | The reduction in the eye diagram on the setup side (or left side of the eye) due to ISI on the address/command signals compared to a case where there is no ISI. |
| Address/command eye reduction (hold) | The reduction in the eye diagram on the hold side (or right side of the eye) due to ISI on the address/command signals compared to a case where there is no ISI. |
| D eye reduction | The total reduction in the eye diagram due to ISI on DQ signals compared to a case where there is no ISI. (It is assumed that the ISI reduces the eye width symmetrically on the left and right sides of the eye.) |
| Delta K arrival time | The increase in variation on the range of arrival times of DQS compared to a case when there is no ISI. (It is assumed that the ISI causes DQS to further vary symmetrically to the left and right.) |

## Board Skews

Skews between PCB traces can reduce timing margins.

Table 3–8 describes the board skew settings.

**Table 3–8. Board Skews Settings (Part 1 of 2) (Part 1 of 2)**

| Parameter | Description |
|---|---|
| Maximum delay difference between devices | The maximun delay difference of data signals between devices. For example, in a two-device configuration there is greater propagation delay for data signals going to and returning from the furthest device relative to the nearest device. |
| Maximum skew within write data group (ie, K group) | The maximum skew between D and BWS signals referenced by a common K signal. |
| Maximun slkew within read data group (ie, CQ group) | The maximum skew between Q signals referenced by a common CQ signal. |
| Maximum skew between CQ groups | The maximum skew between CQ signals of different read data groups. |

**Table 3–8. Board Skews Settings (Part 2 of 2) (Part 2 of 2)**

| Parameter | Description |
|---|---|
| Maximun skew within address/command bus | The maximum skew between the address/command signals. |
| Average delay difference between address/command and K | A value equal to the average of the longest and smallest address/command signal delay values, minus the delay of the K signal. The value can be positive or negative. |
| Average delay difference between write data signals and K | A value equal to the average of the longest and smallest write data signal delay values, minus the delay of the K signal. Write data signals include the D and BWS signals. The value can be positive or negative. |
| Average delay difference between read data signals and CQ | A value equal to the average of the longest and smallest read data signal delay values, minus the delay of the CQ signal. The value can be positive or negative. |

The parameter editor generates a Synopsis Design Constraint (**.sdc**) script. <*variation_name*>**.sdc**, and a pin assignment script, <*variation_name*>**_pin_assignments.tcl**. Both the **.sdc** and the <*variation name*>**_pin_assignments.tcl** support multiple instances. These scripts iterate through all instances of the core and apply the same constraints to all of them.

## Add Pin and DQ Group Assignments

The pin assignment script, <*variation_name*>**_pin_assignments.tcl**, sets up the I/O standards and the input/output termination for the QDR II or QDR II+ SRAM controller with UniPHY. This script also helps to relate the `DQ` and `QK` pin groups together for the Fitter to place them correctly in the device.

The pin assignment script does not create a clock for the design. You must create a clock for the design and provide pin assignments for the signals of both the example driver and testbench that the MegaCore variation generates.

Run the <*variation_name*>**_pin_assignments.tcl** to add the input and output termination, I/O standards, and DQ group assignments to the example design. To run the pin assignment script, follow these steps:

1. On Processing menu, point to **Start**, and click **Start Analysis and Synthesis**.
2. On the Tools menu click **Tcl Scripts**.
3. Specify the **pin_assignments.tcl** file and click **Run**.

If the PLL input reference clock pin does not have the same I/O standard as the memory interface I/Os, the design might not fit into the device because incompatible I/O standards cannot be placed in the same I/O bank.

## Board Settings

The **Board Settings** tab allows you to enter board-related data. In the **Intersymbol Interference** and **Board Skews** sections, you enter information derived during your PCB development process of prelayout (line) and postlayout (board) simulation.

Timing analysis does not consider bus turnaround; consequently, the controller dead times are based on assumptions about the user board trace lengths. For timing analysis to be accurate, board trace delays must be limited to 0.6 ns from FPGA to memory and from memory to FPGA.

For more information about how to include your board simulation results in the Quartus II software and how to assign pins using pin planners, refer to *Volume 6: Design Flow Tutorials* of the *External Memory Interface Handbook*.

# Compile the Design

To compile the design, on the Processing menu, click **Start Compilation**.

After you have compiled the top-level file, you can perform RTL simulation or program your targeted Altera device to verify the top-level file in hardware.

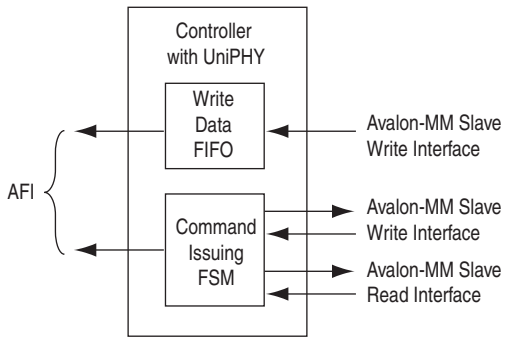
For more information about simulating, refer to the *Simulation* section in volume 4 of the *External Memory Interface Handbook*.

The controller translates memory requests from the Avalon Memory-Mapped (Avalon-MM) interface to AFI, while satisfying timing requirements imposed by the memory configurations. QDR II and QDR II+ SRAM has unidirectional data buses, therefore read and write operations are highly independent of each other and each has its own interface and state machine.

# Block Description

This topic describes the blocks in the IP. Figure 5–1 shows a block diagram of the QDR II and QDR II+ SRAM controller architecture.

**Figure 5–1. QDR II and QDR II+ SRAM Controller Architecture Block Diagram**




## Avalon-MM Slave Read and Write Interfaces

The read and write blocks accept from the Avalon-MM interface read and write requests respectively. Each block has a simple state machine that represents the state of the command and address registers, which stores the command and address when a request arrives.

The read data passes through without the controller registering it, as the PHY takes care of read latency. The write data goes through a pipeline stage to delay for a fixed number of cycles as specified by the write latency. In the full-rate burst length of four controller, the write data is also multiplexed into a burst of 2, which is then multiplexed again in the PHY to become a burst of 4 in DDR.

The user interface to the controller has separate read and write Avalon-MM interfaces because reads and writes are independent of each other in the memory device. The separate channels give efficient use of available bandwidth.

## Command Issuing FSM

The command-issuing full-state machine (FSM) has two states: `INIT` and `INIT_COMPLETE`. In the `INIT_COMPLETE` state, commands are issued immediately as requests arrive using combinational logic and do not require state transitions.

### AFI

In the full-rate burst length of two configuration, the controller can issue both read and write commands in the same clock cycle. In the memory device, both commands are clocked on the positive edge, but the read address is clocked on the positive edge, while the write address is clocked on the negative edge. Care must be taken on how these signals are ordered in the AFI.

For the half-rate burst length of four configuration the controller also issues both read and write commands, but the AFI width is doubled to fill two memory clocks per controller clock. As the controller only issues one write command and one read command per controller clock, the AFI read and write signals corresponding to the other memory cycle are tied to no operation (NOP).

For information on the AFI, refer to "Functional Description—UniPHY" on page 6–1.

## Avalon-MM and Memory Data Width

Table 5–1 shows the data width ratio between the memory interface and the Avalon-MM interface. The half-rate controller does not support burst-of-2 devices because it under-uses the available memory bandwidth. Regardless of full or half-rate decision and the device burst length, the Avalon-MM interface must supply all the data for the entire memory burst in a single clock cycle. Therefore the Avalon-MM data width of the full-rate controller with burst-of-4 devices is four times the memory data width. For width-expanded configurations, the data width is further multiplied by the expansion factor (not shown in table 5-1 and 5-2).

**Table 5–1. Data Width Ratio**

| Memory Burst Length | Half-Rate Designs | Full-Rate Designs |
|---|---|---|
| QDR II 2-word burst | No Support | 2:1 |
| QDR II and II+ 4-word burst | 4:1 | |

## Signal Description

This topic discusses the signals for each interface.

For information on the AFI signals, refer to "UniPHY Signals" on page 6–10.

### Avalon-MM Slave Read Interface

Table 5–2 shows the list of signals of the controller's Avalon-MM slave read interface.

**Table 5–2. Avalon-MM Slave Read Signals (Part 1 of 2)**

| Signal | Width | Direction | Avalon-MM Signal Type | Description |
|---|---|---|---|---|
| avl_r_ready | 1 | Out | waitrequest_n | — |
| avl_r_read_req | 1 | In | read | — |
| avl_r_addr | ≤20 | In | address | — |
| avl_r_rdata_valid | 1 | Out | readdatavalid | — |

**Table 5–2. Avalon-MM Slave Read Signals (Part 2 of 2)**

| Signal | Width | Direction | Avalon-MM Signal Type | Description |
|---|---|---|---|---|
| avl_r_rdata | 16, 18, 36, 72, 144 | Out | readdata | — |
| avl_r_size | log_2(MAX_BURST_SIZE) + 1 | | — | — |

The data width of the Avalon-MM interface is restricted to powers of two when using SOPC Builder or Qsys. Non-power-of-two data widths are supported when using the MegaWizard Plug-In Manager.

## Avalon-MM Slave Write Interface

Table 5–3 shows the list of signals of the controller's Avalon-MM slave write interface.

**Table 5–3. Avalon-MM Slave Write Signals**

| Signal | Width | Direction | Avalon-MM Signal Type | Description |
|---|---|---|---|---|
| avl_w_ready | 1 | Out | waitrequest_n | — |
| avl_w_write_req | 1 | In | write | — |
| avl_w_addr | ≤20 | In | address | — |
| avl_w_wdata | 18, 36, 72, 144 | In | writedata | — |
| avl_w_be | 2,4,8,16 | In | byteenable | — |
| avl_w_size | log_2(MAX_BURST_SIZE) + 1 | | — | — |

# 6. Functional Description—UniPHY

This chapter describes the PHY part of the QDR II and QDR II+ SRAM controllers with UniPHY.

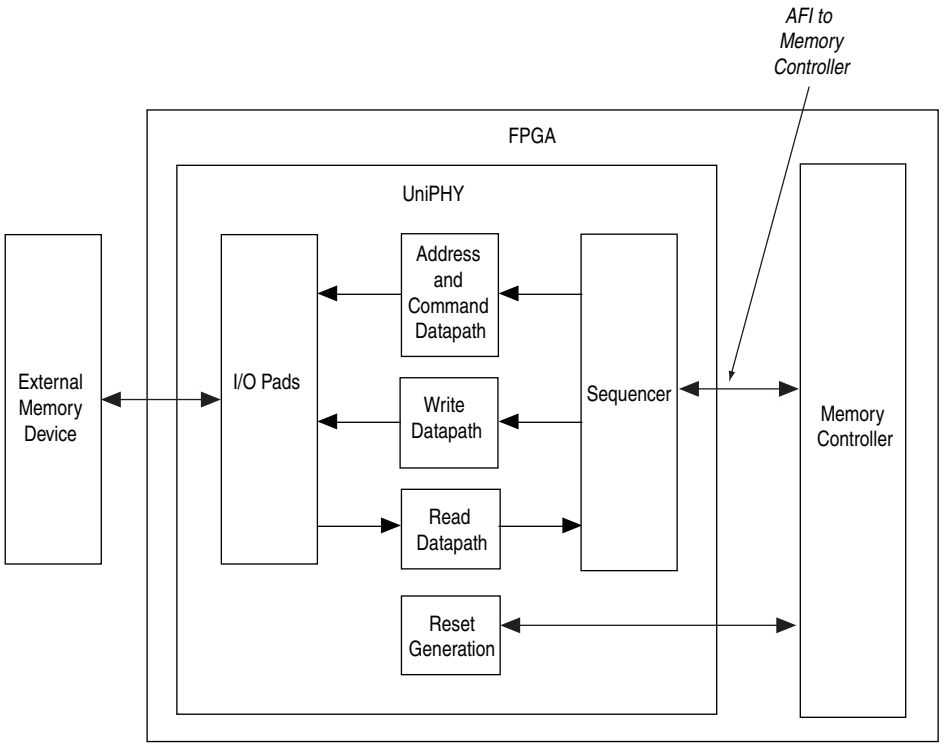
## Block Description

The PHY comprises the following major functional units:

- Reset and Clock Generation
- Address and Command Datapath
- Write Datapath
- Read Datapath
- Sequencer

Figure 6–1 shows the PHY block diagram.

**Figure 6–1. PHY Block Diagram**




## I/O Pads

The I/O pads contain all the I/O instantiations. The bulk of the UniPHY I/O circuitry is encapsulated in the ALTDQ_DQS megafunction (ALTDQ_DQS2 for Stratix V series devices).

## Reset and Clock Generation

The clocking operation in the PHY can be classified into two domains: the PHY-memory domain and the PHY-AFI domain. The PHY-memory domain interfaces with the external memory device and is always at full-rate. The PHY-AFI domain interfaces with the memory controller and can be either a full-rate or half-rate clock based on the choice of the controller. Table 6–1 lists the clocks required for half-rate designs.

**Table 6–1. Clocks—Half-Rate Designs**

| Clock | Source | Clock Rate | Phase | Clock Network Type | Description |
|---|---|---|---|---|---|
| pll_afi_clk | PLL: C0 | Half | 0° | Unconstrained | Clock for AFI logic. |
| pll_mem_clk | PLL: C1 | Full | 0°(1)<br>-45°(2) | Dual-regional(4) | Output clock to memory. |
| pll_write_clk | PLL: C2 | Full | -90°(1)<br>-135°(2)<br>45° (3) | Dual-regional(4) | Clock for write data out to memory (data is center aligned with the delayed pll_write_clk). |
| pll_addr_cmd_clk | PLL: C3 | Half | Set in wizard (default 270°) | Dual-regional | Clock for the address and command out to memory (address and command is center aligned with memory clock). |
| read_capture_clk | Memory | Full | 90° | Local | A continuous running clock from the memory device for capturing read data. |

**Notes for Table 6–1:**

(1) For memory frequencies >240 MHz.
(2) For memory frequencies <=240 MHz.
(3) For memory frequencies >=240 MHz, for Stratix V devices only.
(4) For parameterizations with interface width >36, pll_mem_clk and pll_write_clk are assigned to use the global network.

Table 6–2 lists the clocks required for full-rate designs.

**Table 6–2. Clocks—Full-Rate Designs (Part 1 of 2)**

| Clock | Source | Clock Rate | Phase | Clock Network Type | Description |
|---|---|---|---|---|---|
| pll_afi_clk | PLL: C0 | Full | 0° | Unconstrained | Clock for AFI logic. |
| pll_mem_clk | PLL: C1 | Full | 90°(1)<br>0°(2) | Dual-regional(3) | Output clock to memory. |
| pll_write_clk | PLL: C2 | Full | 180°(1)<br>-90°(2) | Dual-regional(3) | Clock for write data out to memory (data is center aligned with the delayed pll_write_clk). |
| pll_addr_cmd_clk | PLL: C3 | Full | Set in wizard (default 225°) | Dual-regional | Clocks address/command out to memory. 180° gives adress and command center aligned with memory clock; 225° produces best overall timing results. |

**Table 6–2. Clocks—Full-Rate Designs (Part 2 of 2)**

| Clock | Source | Clock Rate | Phase | Clock Network Type | Description |
|---|---|---|---|---|---|
| `read_capture_clk` | Memory | Full | 90° | Local | A continuous running clock from the memory device for capturing read data. |

**Notes for Table 6–2:**

(1) For memory frequencies >240 MHz.

(2) For memory frequencies <=240 MHz.

(3) For parameterizations with interface width >36, pll_mem_clk and pll_write_clk are assigned to use the global network.

The UniPHY uses an active-low, asynchronous assert and synchronous deassert reset scheme. The global reset signal resets the PLL in the PHY and the rest of the system waits in reset until after the PLL is locked. The number of synchronization pipeline stages is 4.

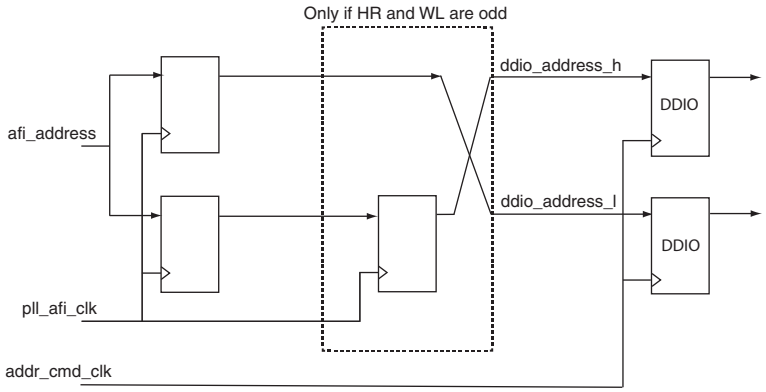
## Address and Command Datapath

The memory controller controls the read and write addresses and commands to meet the memory specifications. The PHY simply passes the address and command received from the memory controller to the memory device. The PHY is also indifferent to address or command; the circuitry is the same for both.

The address and command datapath outputs are connected to the inputs of the address and command I/Os . An ALTDDIO_OUT megafunction converts the addresses from SDR to DDR. An ALTDDIO_OUT megafunction with an ALTIOBUF megafunction delivers a pair of address and command clock to the memory.

Figure 6–2 illustrates the address and command datapath. The controller only requires the registry-and- address-swapping circuitry inside the dotted box when it is operating in half-rate (HR) mode with odd write latency (WL). In full-rate mode, `ddio_address_h` and `ddio_addesss_l` are the same.

**Figure 6–2. Address and Command Datapath**

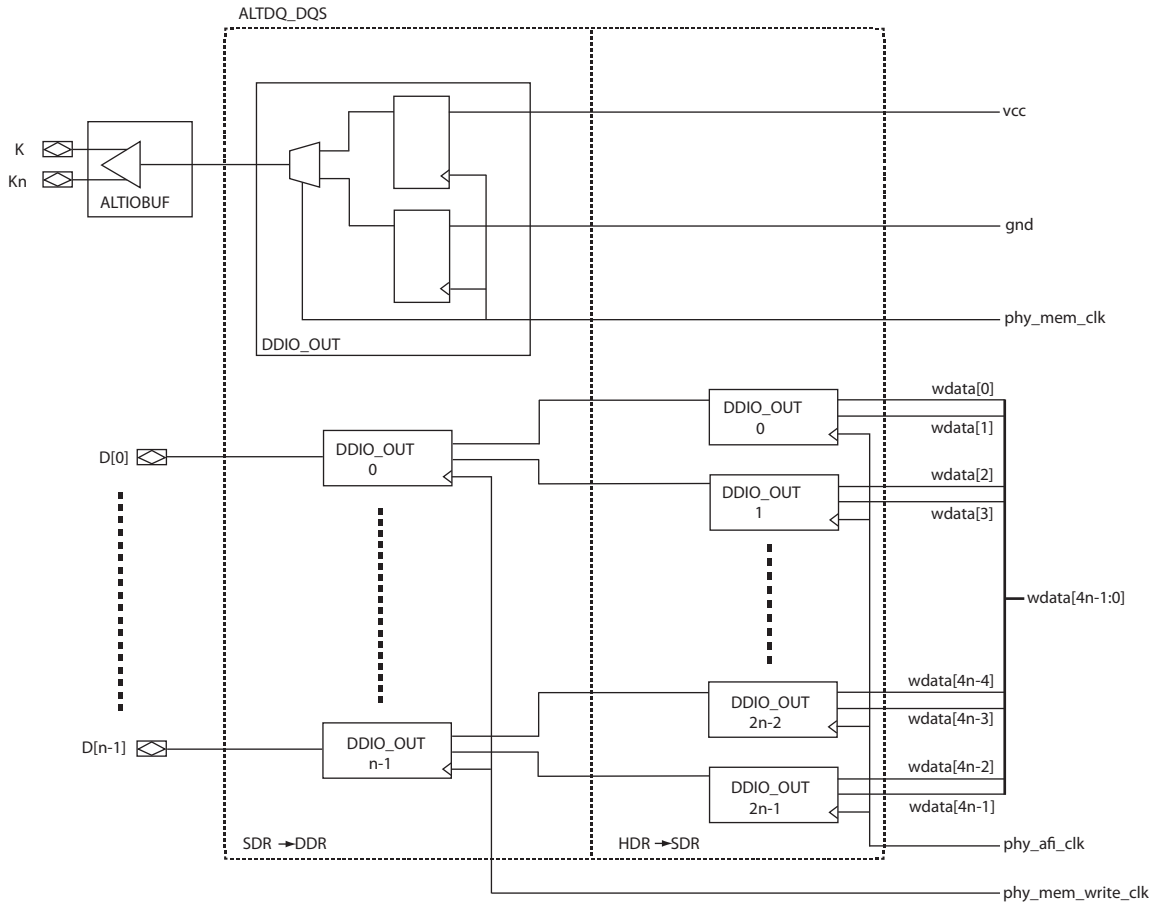


## Write Datapath

The write datapath passes write data from the memory controller to the I/O. The QDR II interface has separate unidirectional read and write pins. The write pins are not controlled by the output-enable signal, and are always driven. Because the pins are unidirectional, there is no need for dynamic termination control.

Figure 6–3 illustrates the write datapath. The full-rate PLL output clock phy_mem_clk is sent to a DDIO_OUT cell. The output of ALTDQ_DQS feeds an ALTIOBUF buffer which creates a pair of pseudodifferential clocks that connects to the memory. In full-rate mode, only the SDR-DDR portion of the ALTDQ_DQS logic is used; in half-rate mode, the HDR-SDR circuitry is also required. The <*variation_name*>**_pin_assignments.tcl** script automatically specifies the logic option that associates all DQ pins to the DQS pin. The Fitter treats the pins as a DQS/DQ pin group.

**Figure 6–3. Write Datapath**



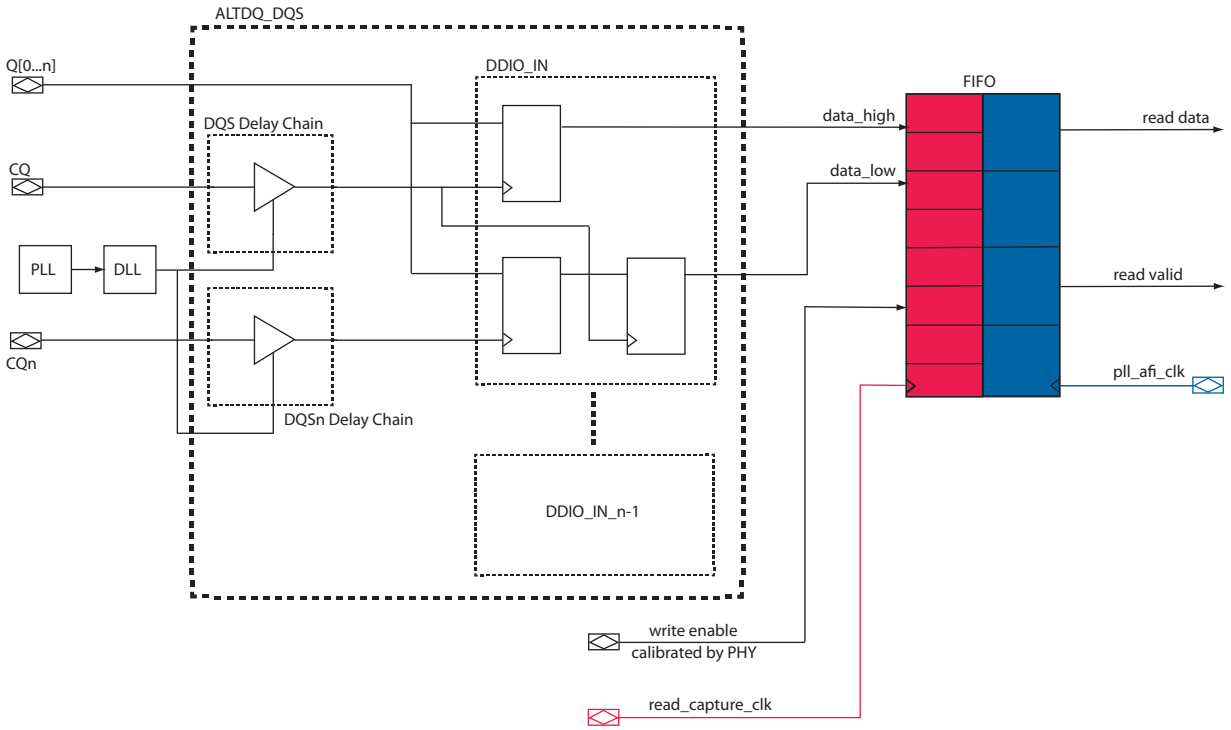

## Read Datapath

The read data is captured in the input mode ALTDQ_DQS in the I/O. The captured data is then forwarded to the read datapath. The read datapath synchronizes read data from the read capture clock domain to the AFI clock domain and converts data from SDR to HDR (half-rate designs only).

In half-rate designs, the write side of the FIFO buffer should be double the size of the read side of the FIFO buffer. The read side only reads one entry after the write side has written into two entries, which effectively converts data from SDR to HDR. In full-rate designs, the size of the FIFO buffer is the same for both write and read as both sides operate at the same rate. For half-rate designs, the FIFO operates at half-rate on both read and write sides, and contains 4 half-rate entries; for full-rate designs, the FIFO operates at full-rate on both read and write sides, and contains 8 full-rate entries.

Figure 6–4 illustrates the read datapath. The DQS and DQSn clocks and the read data (DQ) returned from memory are edge-aligned; the DQS and DQSn delay chains shift the clocks to achieve center alignment.

**Figure 6–4. Read Datapath**




## Sequencer

The sequencer is a state machine that processes the calibration algorithm. The sequencer assumes control of the interface at reset (whether at initial startup or when the IP is reset) and maintains control throughout the calibration process, relinquishing control to the memory controller only after successful calibration. Table 6–3 shows the major states in the sequencer.

**Table 6–3. Sequencer States**

| State | Description |
|---|---|
| RESET | Remain in this state until reset is released. |
| LOAD_INIT | Load any initialization values for simulation purposes. |
| STABLE | Wait until the memory device is stable. The QDR II and QDR II+ specification requires 2,048 cycles of power up wait time. |

**Table 6–3. Sequencer States**

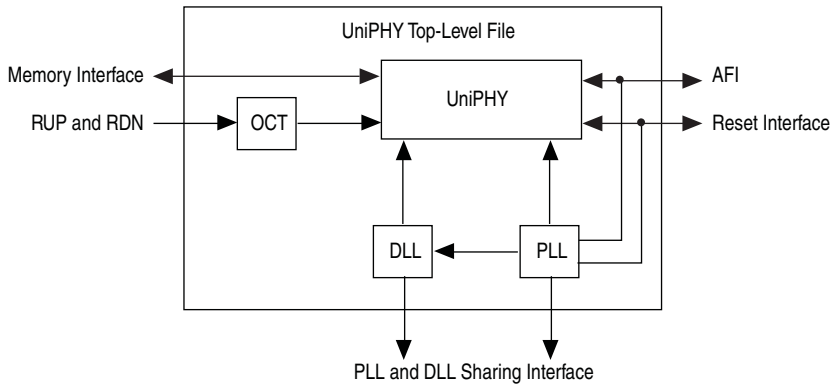
| State | Description |
|---|---|
| WRITE_ZERO | Issue write command to address 0. |
| WAIT_WRITE_ZERO | Write all 0s to address 0. |
| WRITE_ONE | Issue write command to address 1. |
| WAIT_WRITE_ONE | Write all 1s to address 1. |
| **Valid Calibration States** | |
| V_READ_ZERO | Issue read command to address 0 (expected data is all 0s). |
| V_READ_NOP | This state represents the minimum number of cycles required between 2 back-to-back read commands. The number of NOP states depends on the burst length. |
| V_READ_ONE | Issue read command to address 1 (expected data is all 1s). |
| V_WAIT_READ | Wait for read valid signal. |
| V_COMPARE_READ_ZERO_READ_ONE | Parameterizable number of cycles to wait before making the read data comparisons. |
| V_CHECK_READ_FAIL | When a read fails, the write pointer (in the AFI clock domain) of the valid FIFO buffer is incremented. The read pointer of the valid FIFO buffer is in the DQS clock domain. The gap between the read and write pointers is effectively the latency between the time when the PHY receives the read command and the time valid data is returned to the PHY. |
| V_ADD_FULL_RATE | Advance the read valid FIFO buffer write pointer by an extra full rate cycle. |
| V_ADD_HALF_RATE | Advance the read valid FIFO buffer write pointer by an extra half rate cycle. In full-rate designs, equivalent to V_ADD_FULL_RATE. |
| V_READ_FIFO_RESET | Reset the read and write pointers of the read data synchronization FIFO buffer. |
| V_CALIB_DONE | Valid calibration is successful. |
| **Latency Calibration States** | |
| L_READ_ONE | Issue read command to address 1 (expected data is all 1s). |
| L_WAIT_READ | Wait for read valid signal from read datapath. Initial read latency is set to a predefined maximum value. |
| L_COMPARE_READ_ONE | Check returned read data against expected data. If data is correct, go to L_REDUCE_LATENCY; otherwise go to L_ADD_MARGIN. |
| L_REDUCE_LATENCY | Reduce the latency counter by 1. |
| L_READ_FLUSH | Read from address 0 (expected data is all 0s), to flush the contents of the read data resynchronization FIFO buffer. |
| L_WAIT_READ_FLUSH | Wait until the whole FIFO buffer is flushed, then go back to L_READ and try again. |
| L_ADD_MARGIN | Increment latency counter by 3 (1 cycle to get the correct data, 2 more cycles of margin for run time variations). If latency counter value is smaller than predefined ideal condition minimum, then go to CALIB_FAIL. |
| CALIB_DONE | Calibration is successful. |
| CALIB_FAIL | Calibration is not successful. |

# Interfaces

Figure 6–5 shows the major blocks of the UniPHY and how it interfaces with the external memory device and the controller.

☞ Instantiating the DLL and the PLL on the same level as the UniPHY eases DLL and PLL sharing.

**Figure 6–5. UniPHY Interfaces with the Controller and the External Memory**




The following interfaces are on the UniPHY top-level file:

- AFI
- Memory interface
- DLL and PLL sharing interface
- OCT interface

## The Memory Interface

For more information on the memory interface, refer to "UniPHY Signals" on page 6–10.

## The DLL and PLL Sharing Interface

You can generate the UniPHY memory interface as a master or as a slave, depending on the setting of the **Master for PLL/DLL sharing** option on the **General Settings** tab of the parameter editor. The top level of a generated UniPHY variant contains both a PLL and a DLL; the PLL produces a variety of required clock signals derived from the reference clock, and the DLL produces a delay codeword. The top-level file defines the PLL and DLL output signals as outputs for the master, and as inputs for the slave. When you instantiate master and slave variants into your HDL code, you must connect the PLL outputs from the master to the clock inputs of the slaves.

☞ The master **.qip** file must appear before the slave **.qip** file in the Quartus II Settings File (**.qsf**).

The UniPHY memory interface requires one PLL and one DLL to produce the clocks and delay codeword. The PLL and DLL can be shared using a master and slave scenario. The top-level file defines the PLL and DLL output signals as inputs and outputs and an additional parameter `PLL_DLL_MASTER` is also defined. If `PLL_DLL_MASTER` is 1, the RTL instantiates the PLL and DLL, which drives the clock and DLL codeword inputs and outputs. If the parameter is 0, the wires previously connected to the output of the PLL and DLL are assigned to the clock and DLL codeword input and outputs. Inputs and outputs are specified based on the setting of the **PLL/DLL sharing** option.

If you generate a slave IP core, you must modify the timing scripts to allow the timing analysis to correctly resolve clock names and analyze the IP core. Otherwise the software issues critical warnings and an incorrect timing report.

To modify the timing script, follow these steps:

1. In a text editor, open the <*IP core name*>**/constraints directory/**<*IP core name*>**_timing.tcl** file.
2. Search for the following 2 lines:
   - `set master_corename "_MASTER_CORE_"`
   - `set master_instname "_MASTER_INST_"`
3. Replace `_MASTER_CORE_` with the core name and `_MASTER_INST_` with instance name of the UniPHY master to which the slave is connected.

   The instance name is the full path to the instance and is in the <*IP core name*>**_all_pins.txt** file that is automatically generated after the <*IP core name*>**_pin_assignments.tcl** script runs.

4. If the slave component is connected to a user-defined PLL rather than a UniPHY master, you must manually enter all clock names.
   - Remove the `master_corename` and `master_instname` variables with the checks performed in the eight lines following them.
   - You can use all clock name assignments as templates. For example set `local_pll_afi_clk "mycomponent|mypll|my_afi_clk"`.

   CAUTION: You must be extremely careful when connecting clock signals to the slave. Connecting to clocks with frequency or phase different than what the core expects may result in hardware failures.

The DLL and PLL Sharing interface is not available with SOPC Builder.

## The OCT Sharing Interface

By default, the UniPHY IP generates the required OCT control block in the top-level RTL file for the PHY. If you want, you can instantiate this block elsewhere in your code and feed the required termination control signals into the IP core by turning off **Master for OCT Control Block** on the **PHY Settings** tab. If you turn off **Master for OCT Control Block**, you must instantiate the OCT control block, or use another UniPHY instance as a master, and ensure that the parallel and series termination control bus signals are connected to the PHY.

Termination Control Block assignments must be created for all calibrated input-only pins, to designate which OCT control block to use for those pins. If the UniPHY IP is in OCT Control Block master mode, these assignments are included in the *<variation_name>***_pin_assignments.tcl** file which must be run after analysis and synthesis. If the UniPHY IP is not using OCT Control Block master mode you must manually create the required assignments to connect the input-only pins to the relevant OCT control block. For QDRII this is the input clocks and input data pins, all output and bidirectional pins are hard coded between the pin's I/O buffer and the series and parallel termination control signals.

Figure 6–6 and Figure 6–7, respectively, show the PHY architecture with and without Master for OCT Control Block.

**Figure 6–6. PHY Architecture with Master for OCT Control Block**

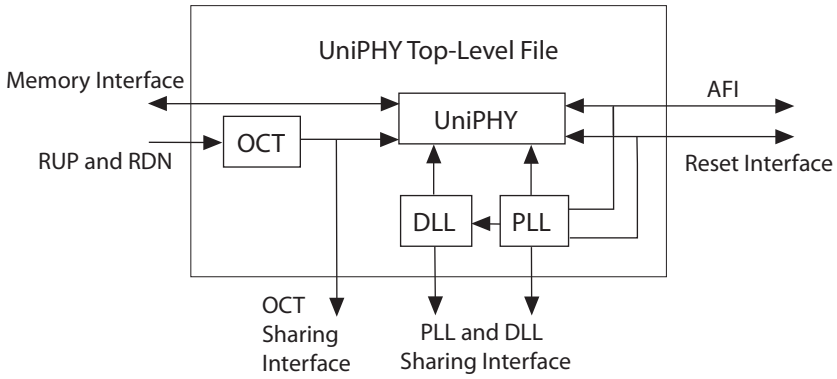
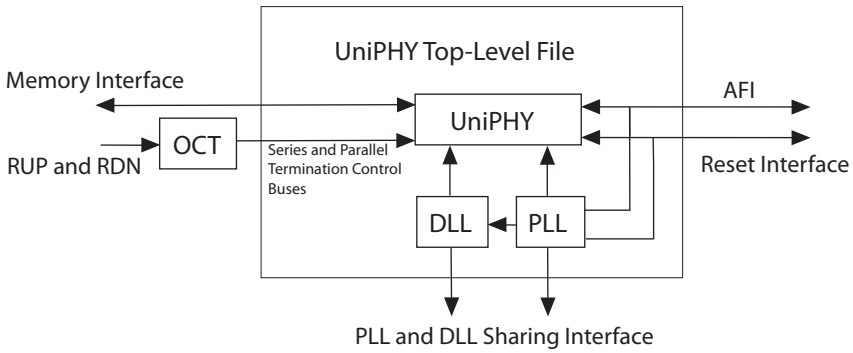



**Figure 6–7. PHY Architecture without Master for OCT Control Block**




☞ The OCT Sharing Interface and OCT slave mode are not available with SOPC Builder.

# UniPHY Signals

This section describes the UniPHY signals. Table 6–4 shows the clock and reset signals.

**Table 6–4. Clock and Reset Signals**

| Name | Direction | Description |
|---|---|---|
| pll_ref_clk | Input | PLL reference clock input. |
| global_reset_n | Input | Active low global reset for PLL and all logic in the PHY, which causes a complete reset of the whole system. |
| soft_reset_n | Input | Holding soft_reset_n low holds the PHY in a reset state. However it does not reset the PLL, which keeps running. It also holds the afi_reset_n output low. Mainly for use by SOPC Builder. |
| reset_request_n | Output | When the PLL is locked, reset_request_n is high. When the PLL is out of lock, reset_request_n is low. |
| seriesterminationcontrol | Input (for OCT slave) | Required signal for PHY to provide series termination calibration value. Must be connected to a user-instantiated OCT control block (alt_oct) or another UniPHY instance that is set to OCT master mode. |
| | Output(for OCT master) | Unconnected PHY signal, available for sharing with another PHY. |
| parallelterminationcontrol | Input (for OCT slave) | Required signal for PHY to provide series termination calibration value. Must be connected to a user-instantiated OCT control block (alt_oct) or another UniPHY instance that is set to OCT master mode. |
| | Output(for OCT master) | Unconnected PHY signal, available for sharing with another PHY. |
| oct_rdn | Input (for OCT master) | Must connect to calibration resistor tied to GND on the appropriate RDN pin on the device. (See appropriate device handbook.) |
| oct_rup | Input (for OCT master) | Must connect to calibration resistor tied to $V_{ccio}$ on the appropriate RUP pin on the device. (See appropriate device handbook.) |

Table 6–5 shows the AFI signals.

**Table 6–5. AFI Signals (Part 1 of 2)**

| Name | Direction | Width | Description |
|---|---|---|---|
| **Clocks and Reset** | | | |
| afi_clk | Output | 1 | Half-rate or full-rate clock supplied to controller and system logic. |
| afi_reset_n | Output | 1 | Reset output on afi_clk clock domain. For use as asynchronous reset. This signal is asynchronously asserted and synchronously deasserted. |

**Table 6–5. AFI Signals (Part 2 of 2)**

| Name | Direction | Width | Description |
|---|---|---|---|
| **Address and Command** | | | |
| afi_addr | Input | MEM_ADDRESS_WIDTH | For full-rate, burst-of-4 configuration. |
| | Input | 2 x MEM_ADDRESS_WIDTH | For full-rate, burst-of-2 configuration. The upper memory address bits contain the write address, while the lower memory address bits contain the read address. |
| | Input | 2 x MEM_ADDRESS_WIDTH | For half-rate, burst-of-4 configuration. The upper memory address bits contain the read address, while the lower memory address bits contain the write address. |
| afi_wps_n | Output | MEM_CONTROL_WIDTH | Write enable. |
| afi_rps_n | Output | MEM_CONTROL_WIDTH | Read enable. |
| **Write Data** | | | |
| afi_dm | Input | MEM_DM_WIDTH × AFI_RATIO | Data mask input that generates mem_dm. |
| afi_wdata | Input | MEM_DQ_WIDTH × 2 × AFI_RATIO | Write data input that generates mem_dq. |
| afi_wdata_valid | Input | MEM_WRITE_DQS_ WIDTH × AFI_RATIO | Write data valid that generates mem_dq and mem_dm output enables. |
| **Read Data** | | | |
| afi_rdata | Output | MEM_DQ_WIDTH × 2 × AFI_RATIO | Read data |
| afi_rdata_en | Input | MEM_READ_DQS_ WIDTH × AFI_RATIO | Doing read input. Indicates that the memory controller is currently performing a read operation. |
| afi_rdata_valid | Output | AFI_RATIO | Read data valid indicating valid read data on afi_rdata, in the byte lanes and alignments that were indicated on afi_rdata_en. |
| **Calibration Control and Status** | | | |
| afi_cal_success | Output | 1 | '1' signals that calibration has completed |
| afi_cal_fail | Output | 1 | '1' signals that calibration has failed |

Table 6–6 shows the QDR II and QDR II+ SRAM interface signals.

**Table 6–6. QDR II and QDR II+ SRAM Interface Signals**

| Name | Direction | Width | Description |
|---|---|---|---|
| mem_address | Output | MEM_ADDRESS_WIDTH | Address. |
| mem_bws_n | Output | MEM_DM_WIDTH | Data mask. |
| mem_wps_n | Output | MEM_CONTROL_WIDTH | Write enable. |
| mem_rps_n | Output | MEM_CONTROL_WIDTH | Read enable. |
| mem_doff_n | Output | MEM_CONTROL_WIDTH | Connecting this pin to ground turns off the DLL inside the device. |

**Table 6–6. QDR II and QDR II+ SRAM Interface Signals**

| Name | Direction | Width | Description |
|---|---|---|---|
| mem_k, mem_kn | Output | MEM_WRITE_DQS_WIDTH | Write clock(s) to memory, 1 clock per DQS group. |
| mem_cq, mem_cq_n | Input | MEM_READ_DQS_WIDTH | Read clock(s) from memory, 1 clock per DQS group |
| mem_d | Input | MEM_DQ_WIDTH | Input data bus. |
| mem_q | Output | MEM_DQ_WIDTH | Output data bus. |

Table 6–7 shows the top-level signals generated for HardCopy migration.

**Table 6–7. Top-Level HardCopy Migration Signals (Part 1 of 2)**

| Name | Direction | Description |
|---|---|---|
| **altsyncram Signals** | | |
| hc_rom_config_clock | Input | Write clock for the ROM loader. This clock is used as the write clock of the NIOS code memory. |
| hc_rom_config_datain | Input | Data input from external ROM. |
| hc_rom_config_rom_data_ready | Input | Asserts to the code memory loader that the word memory is ready to be loaded. |
| hc_rom_config_init | Input | Triggers the ROM loading process. Should be asserted for one hc_rom_config_clock cycle after PLL is locked |
| hc_rom_config_init_busy | Output | When asserted, indicates ROM loading is in progress. The soft_reset_n signal should be de-asserted if the ROM data is not loaded, and also when the ROM is being loaded. The falling edge of hc_rom_config_init_busy indicates the completion of the ROM loading process, at which time, soft_reset_n can be asserted. |
| hc_rom_config_rom_rden | Output | Read-enable signal that connects to the external ROM. |
| hc_rom_config_rom_address | Output | ROM address that connects to the external ROM. |
| **DLL Reconfiguration Signals** | | |
| hc_dll_config_dll_offset_ctrl_addnsub | Input | Addition and subtraction control port for the DLL. This port controls if the delay-offset setting on hc_dll_config_dll_offset_ctrl_offset is added or subtracted. |
| hc_dll_config_offset_ctrl_offset | Input | Offset input setting for the DLL. This setting is a Gray-coded offset that is added or subtracted from the current value of the DLL's delay chain. |
| hc_dll_config_dll_offset_ctrl_offsetctrlout | Output | The registered and Gray-coded value of the current delay-offset setting. |

**Table 6–7. Top-Level HardCopy Migration Signals (Part 2 of 2)**

| Name | Direction | Description |
|---|---|---|
| **PLL Reconfiguration Signals** | | |
| hc_pll_config_configupdate | Input | Control signal to enable PLL reconfiguration. |
| hc_pll_config_phasecounterselect | Input | Specifies the counter select for dynamic phase adjustment. |
| hc_pll_config_phasestep | Input | Specifies the phase step for dynamic phase shifting. |
| hc_pll_config_phaseupdown | Input | Specifies whether the phase shift is up or down. |
| hc_pll_config_scanclk | Input | PLL reconfiguration scan chain clock. |
| hc_pll_config_scanclkena | Input | Clock enable port of the hc_pll_config_scanclk clock. |
| hc_pll_config_scandata | Input | Serial input data for the PLL reconfiguration scan chain. |
| hc_pll_config_scandataout | Output | Data output of the serial scan chain. |
| hc_pll_config_scandone | Output | This signal is asserted when the scan chain write operation is in progress. This signal is deasserted when the write operation is complete. |

Table 6–8 shows the parameters that Table 6–5 through Table 6–7 mention.

**Table 6–8. Parameters (Part 1 of 2)**

| Parameter Name | Description |
|---|---|
| AFI_RATIO | AFI_RATIO is 1 in full-rate designs.<br>AFI_RATIO is 2 for half-rate designs. |
| MEM_IF_DQS_WIDTH | The number of DQS pins in the interface. |
| MEM_ADDRESS_WIDTH | The address width of the specified memory device. |
| MEM_BANK_WIDTH | The bank width of the specified memory device. |
| MEM_CHIP_SELECT_WIDTH | The chip-select width of the specified memory device. |
| MEM_CONTROL_WIDTH | The control width of the specified memory device. |
| MEM_DM_WIDTH | The DM width of the specified memory device. |
| MEM_DQ_WIDTH | The DQ width of the specified memory device. |
| MEM_READ_DQS_WIDTH | The READ DQS width of the specified memory device. |
| MEM_WRITE_DQS_WIDTH | The WRITE DQS width of the specified memory device. |
| OCT_SERIES_TERM_CONTROL_WIDTH | — |
| OCT_PARALLEL_TERM_CONTROL_WIDTH | — |
| AFI_ADDRESS_WIDTH | The AFI address width, derived from the corresponding memory interface width. |
| AFI_BANK_WIDTH | The AFI bank width, derived from the corresponding memory interface width. |
| AFI_CHIP_SELECT_WIDTH | The AFI chip select width, derived from the corresponding memory interface width. |
| AFI_DATA_MASK_WIDTH | The AFI data mask width. |

**Table 6–8. Parameters (Part 2 of 2)**

| Parameter Name | Description |
|---|---|
| AFI_CONTROL_WIDTH | The AFI control width, derived from the corresponding memory interface width. |
| AFI_DATA_WIDTH | The AFI data width. |
| AFI_DQS_WIDTH | The AFI DQS width. |
| DLL_DELAY_CTRL_WIDTH | The DLL delay output control width. |
| NUM_SUBGROUP_PER_READ_DQS | A read datapath parameter for timing purposes. |
| QVLD_EXTRA_FLOP_STAGES | A read datapath parameter for timing purposes. |
| READ_VALID_TIMEOUT_ WIDTH | A read datapath parameter; calibration fails when the timeout counter expires. |
| READ_VALID_FIFO_WRITE_ADDR _WIDTH | A read datapath parameter; the write address width for half-rate clocks. |
| READ_VALID_FIFO_READ_ ADDR_WIDTH | A read datapath parameter; the read address width for full-rate clocks. |
| MAX_LATENCY_COUNT_ WIDTH | A latency calibration parameter; the maximum latency count width. |
| MAX_READ_LATENCY | A latency calibration parameter; the maximum read latency. |
| READ_FIFO_READ_ADDR_ WIDTH | — |
| READ_FIFO_WRITE_ADDR_ WIDTH | — |
| MAX_WRITE_LATENCY_ COUNT_WIDTH | A write datapath parameter; the maximum write latency count width. |
| INIT_COUNT_WIDTH | An initailization sequence. |
| SEQ_BURST_COUNT_WIDTH | The burst count width for the sequencer. |
| VCALIB_COUNT_WIDTH | The width of a counter used by the sequencer. |
| DOUBLE_MEM_DQ_WIDTH | — |
| HALF_AFI_DATA_WIDTH | — |
| CALIB_REG_WIDTH | The width of the calibration status register. |
| NUM_AFI_RESET | The number of AFI resets to generate. |

## AFI Signal Names

The QDR II and QDR II+ SRAM controllers with UniPHY use AFI.

The AFI timing is identical to the DDR3 SDRAM AFI in the Quartus II software version 9.0. However, some signals have been renamed, some added, and others removed from the AFI definition. The AFI includes only signals that are part of the controller-to-PHY interface, clocks, and reset. All signals on the controller-to-PHY interface have the **afi_** prefix to the signal name. Table 6–9 shows the renamed AFI signals and original (Quartus II software version 9.0) names.

**Table 6–9. AFI New Signal Names**

| AFI Name | Old Name |
|---|---|
| afi_clk | ctl_clk |
| afi_reset_n | ctl_reset_n |

**Table 6–9. AFI New Signal Names**

| AFI Name | Old Name |
|---|---|
| afi_addr | ctl_addr |
| afi_ba | ctl_ba |
| afi_cke | ctl_cke |
| afi_cs_n | ctl_cs_n |
| afi_ras_n | ctl_ras_n |
| afi_we_n | ctl_we_n |
| afi_cas_n | ctl_cas_n |
| afi_dqs_burst | ctl_dqs_burst |
| afi_wdata_valid | ctl_wdata_valid |
| afi_wdata | ctl_wdata |
| afi_dm | ctl_dm |
| afi_wlat | ctl_wlat |
| afi_rdata_en | ctl_doing_read |
| afi_rdata | ctl_rdata |
| afi_mem_clk_disable | ctl_mem_clk_disable |
| afi_cal_success | ctl_cal_success |
| afi_cal_fail | ctl_cal_fail |
| afi_cal_req | ctl_cal_req |

# PHY-to-Controller Interfaces

This section describes the typical modules that are connected to the UniPHY PHY and the port name prefixes each module uses. This section describes using a custom controller and describes the AFI.

The AFI standardizes and simplifies the interface between controller and PHY for all Altera memory designs, thus allowing you to easily interchange your own controller code with Altera's high-performance controllers. The AFI PHY includes an administration block that configures the memory for calibration and performs necessary accesses to mode registers that configure the memory as required (these calibration processes are different).

For half-rate designs, the address and command signals in the UniPHY are asserted for one `mem_clk` cycle (1T addressing), such that there are two input bits per address and command pin in half-rate designs. If you require a more conservative 2T addressing, drive both input bits (of the address and command signal) identically in half-rate designs.

Figure 6–8 shows the half-rate write operation.

**Figure 6–8. Half-Rate Write with Word-Aligned Data**

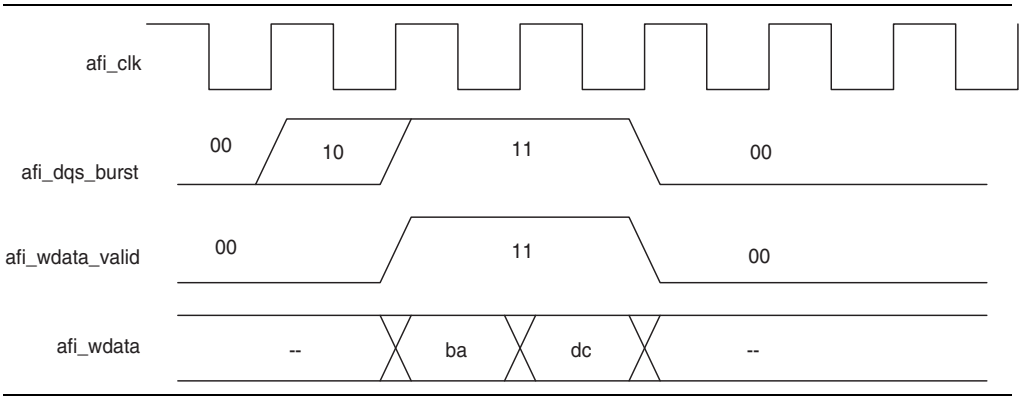



Figure 6–9 shows a full-rate write.

**Figure 6–9. Full-Rate Write**

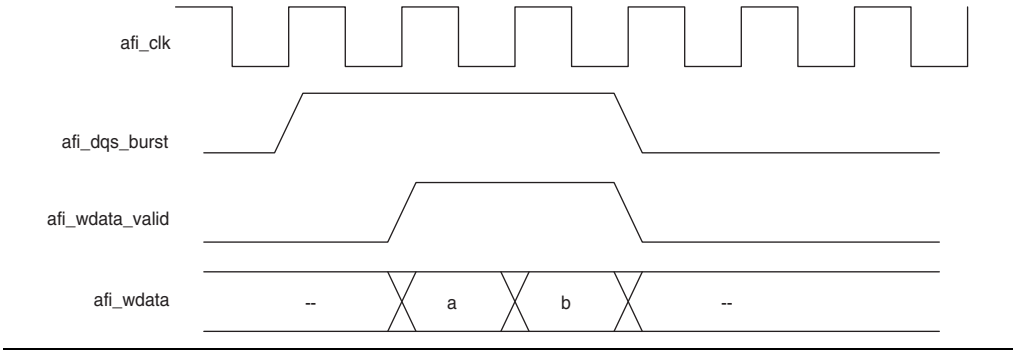


Figure 6–10 shows full-rate reads; Figure 6–11 shows half-rate reads.

**Figure 6–10. Full-Rate Reads**

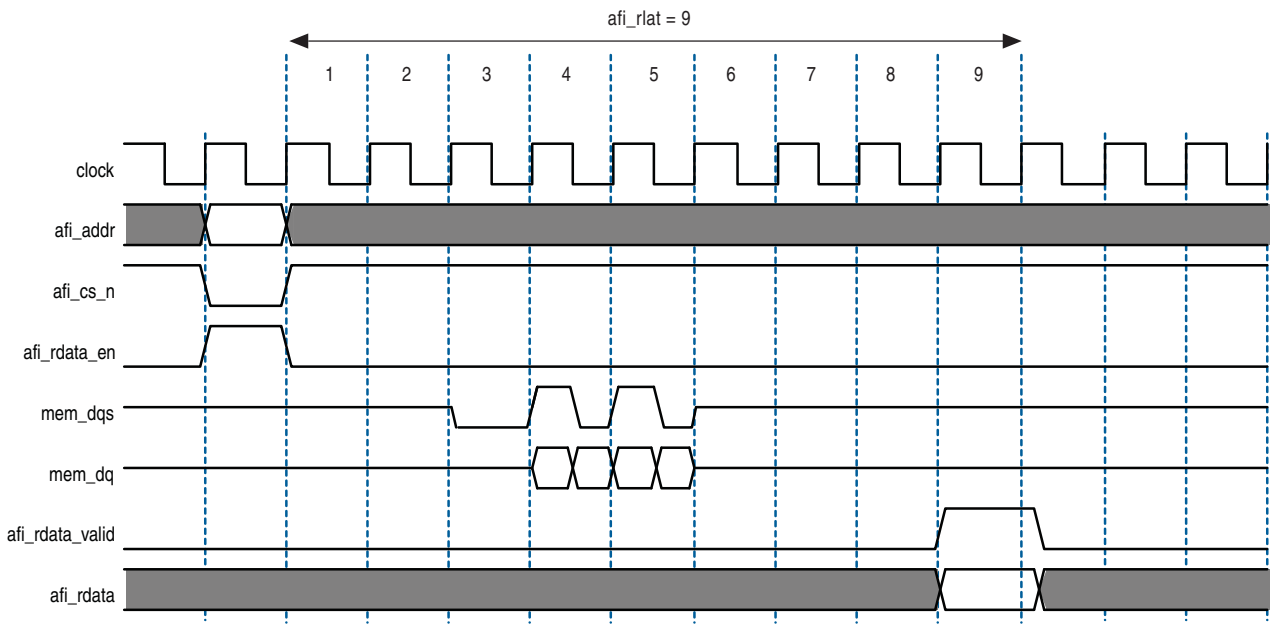



**Figure 6–11. Half-Rate Reads**

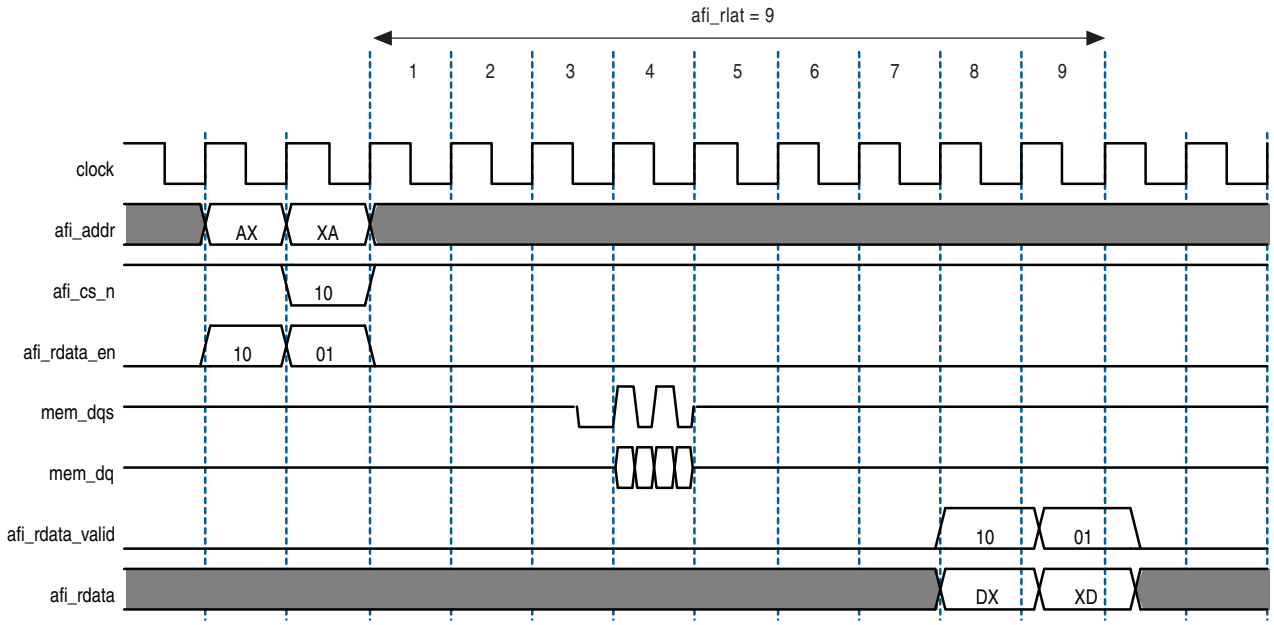


Figure 6–12 and Figure 6–13 show writes and reads, where the data is written to and read from the same address. In each example, afi_rdata and afi_wdata are aligned with controller clock (afi_clk) cycles. All the data in the bit vector is valid at once.

The AFI has the following conventions:

- With the AFI, high and low signals are combined in one signal, so for a single chip-select (afi_cs_n) interface, afi_cs_n[1:0], where location 0 appears on the memory bus on one mem_clk cycle and location 1 on the next mem_clk cycle.

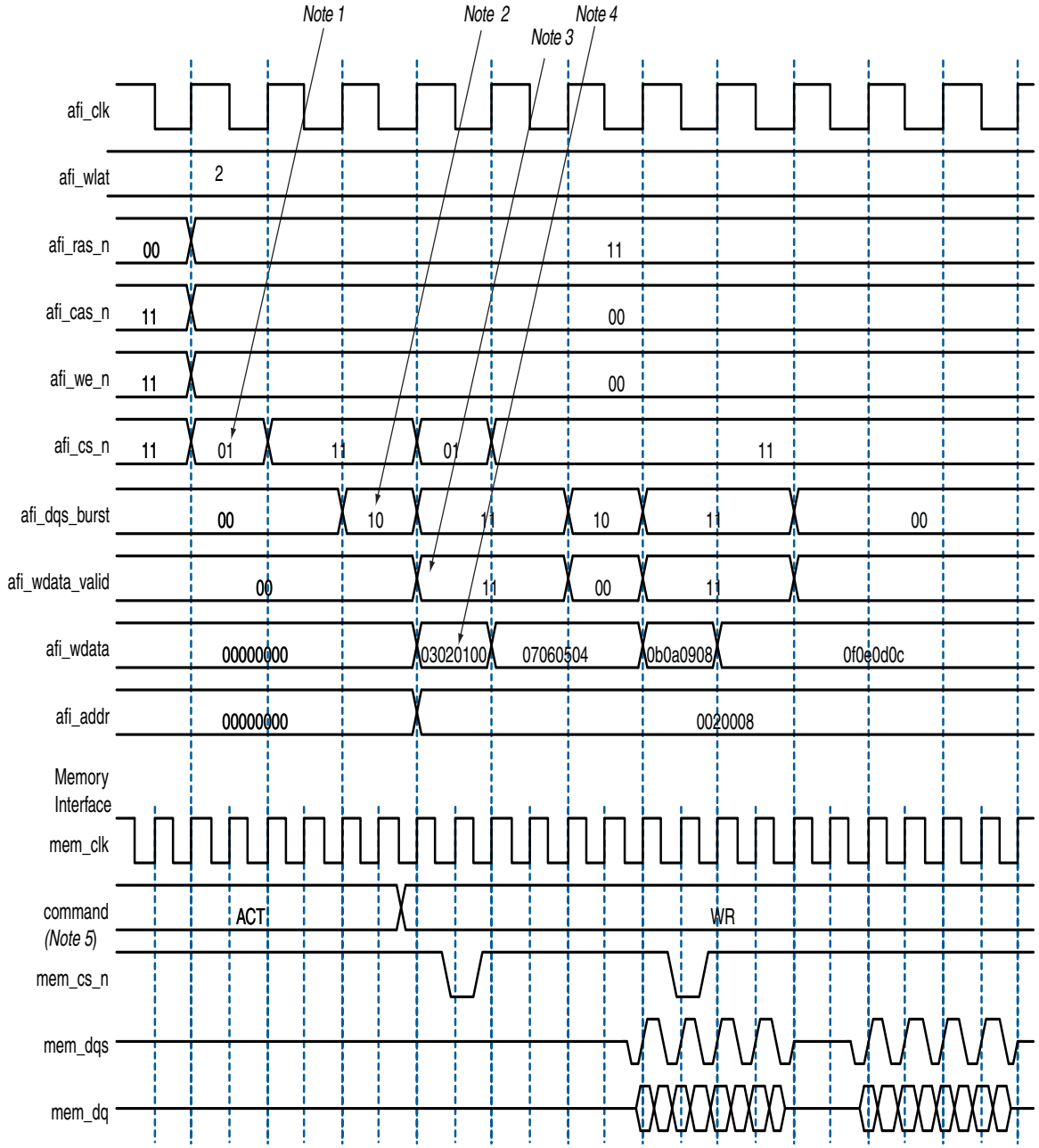
  This convention is maintained for all signals, so for an 8 bit memory interface, the write data (afi_wdata) signal is afi_wdata[31:0], where the first data on the DQ pins is afi_wdata[7:0], then afi_wdata[15:8], then afi_wdata[23:16], then afi_wdata[31:24].

- Spaced reads and writes have the following definitions:
  - Spaced writes—write commands separated by a gap of one controller clock (afi_clk) cycle.
  - Spaced reads—read commands separated by a gap of one controller clock (afi_clk) cycle.

Figure 6–12 through Figure 6–13 assume the following general points:

- The burst length is four.
- An 8-bit interface with one chip-select.
- The data for one controller clock (afi_clk) cycle represents data for two memory clock (mem_clk) cycles (half-rate interface).

**Figure 6–12. Word-Aligned Writes**




**Notes to Figure 6–12:**

(1) To show the even alignment of `afi_cs_n`, expand the signal (this convention applies for all other signals).

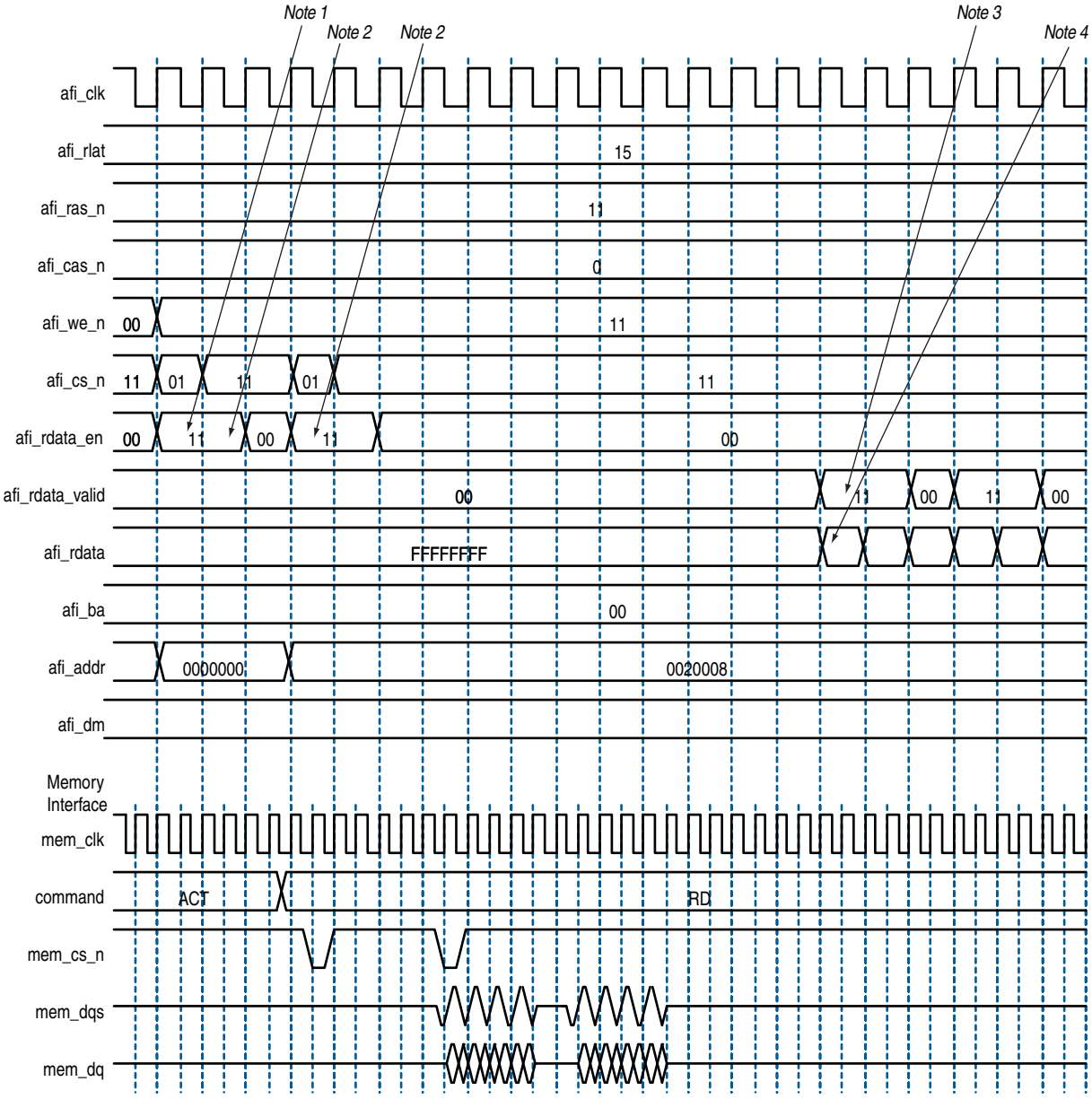
(2) The `afi_dqs_burst` must go high one memory clock cycle before `afi_wdata_valid`. Compare with the word-unaligned case.

(3) The `afi_wdata_valid` is asserted two `afi_wlat` controller clock (`afi_clk`) cycles after chip select (`afi_cs_n`) is asserted. The `afi_wlat` indicates the required write latency in the system. The value is determined during calibration and is dependant upon the relative delays in the address and command path and the write datapath in both the PHY and the external DDR SDRAM subsystem. The controller must drive `afi_cs_n` and then wait `afi_wlat` (two in this example) `afi_clks` before driving `afi_wdata_valid`.

(4) Observe the ordering of write data (`afi_wdata`). Compare this to data on the `mem_dq` signal.

(5) In all waveforms a command record is added that combines the memory pins `ras_n`, `cas_n` and `we_n` into the current command that is issued. This command is registered by the memory when chip select (`mem_cs_n`) is low. The important commands in the presented waveforms are WR = write, ACT = activate.

**Figure 6–13. Word-Aligned Reads**




**Notes to Figure 6–13:**

(1) For AFI, `afi_rdata_en` is required to be asserted one memory clock cycle before chip select (`afi_cs_n`) is asserted. In the half-rate `afi_clk` domain, this requirement manifests as the controller driving `11` (as opposed to the `01`) on `afi_rdata_en`.

(2) AFI requires that `afi_rdata_en` is driven for the duration of the read. In this example, it is driven to `11` for two half-rate `afi_clks`, which equates to driving to `1`, for the four memory clock cycles of this four-beat burst.

(3) The `afi_rdata_valid` returns 15 (`afi_rlat`) controller clock (`afi_clk`) cycles after `afi_rdata_en` is asserted. Returned is when the `afi_rdata_valid` signal is observed at the output of a register within the controller. A controller can use the `afi_rlat` value to determine when to register to returned data, but this is unnecessary as the `afi_rdata_valid` is provided for the controller to use as an enable when registering read data.

(4) Observe the alignment of returned read data relative to data on the bus.

# Using a Custom Controller

The UniPHY-based memory interface IP cores integrate the PHY and the memory controller. To replace the Altera high-performance memory controller with a custom memory controller, perform the following steps:

1. Parameterize and generate your variation of the UniPHY-based memory controller IP as described in "Getting Started" on page 2–1.

   This step creates a top-level HDL file called <*variation_name*>**.v** (or <*variation_name*>**.vhd**), and a sub-directory called <*variation_name*>.

   The top-level module instantiates the <*variation_name*>_<*stamp*>**_controller_phy** module in the <*variation_name*> subdirectory. The <*variation_name*>_<*stamp*>**_controller_phy** module instantiates the PHY and the controller.

    <*stamp*> is a unique identifier determined by the MegaWizard Plug-in Manager, SOPC Builder, or Qsys, during generation.

2. Open the <*variation_name*>**/**<*variation_name*>_<*stamp*>**_controller_phy.sv** file.
3. Replace the <*variation_name*>_<*stamp*>**_alt_qdr_controller** module with your custom controller module.
4. Delete the ports of the Altera high-performance memory controller, and add the top-level ports of your custom controller.
5. Similarily, update the port names in the top-level module in the <*variation_name*>**.v** or <*variation_name*>**.vhd** file.
6. Compile and simulate the design to confirm correct operation.

 Regenerating the UniPHY memory interface IP erases all modifications made to the HDL files. The parameters you select in the parameter editor are stored in the top-level <*variation_name*> module; hence, you must repeat the above steps every time you regenerate the IP variation.

 For half-rate controllers, AFI signals are double the bus width of the memory interface. Half rate controllers have double the width of the signal and run at half the speed. Hence, the overall bandwidth is maintained. Such double-width signals are divided into two signals for transmission to the memory interface, with a higher order bits representing the most-significant bit (MSB) and a lower order bits representing the least-significant bit (LSB). The LSB is transmitted first, and is followed by the MSB.

# Using a Vendor-Specific Memory Model

You can replace the Altera-supplied memory model with a vendor-specific memory model. In general, you may find vendor-specific models to be standardized, thorough, and well supported, but sometimes more complex to setup and use.

If you do want to replace the Altera-supplied memory model with a vendor-supplied memory model, observe the following guidelines:

- Ensure that the vendor-supplied memory model that you have is correct for your memory device.
- Disconnect all signals from the default memory model and reconnect them to the vendor-supplied memory model.
- If you intend to run simulation from the Quartus II software, ensure that the **.qip** file points to the vendor-supplied memory model.

# 7. Functional Description—Example Top-Level Project

IP generation creates an example top-level project that shows you how to instantiate and connect the controller.

The example top-level project contains a testbench, which is for use with Verilog HDL only language simulators such as ModelSim-AE Verilog, and shows simple operation of the memory interface.

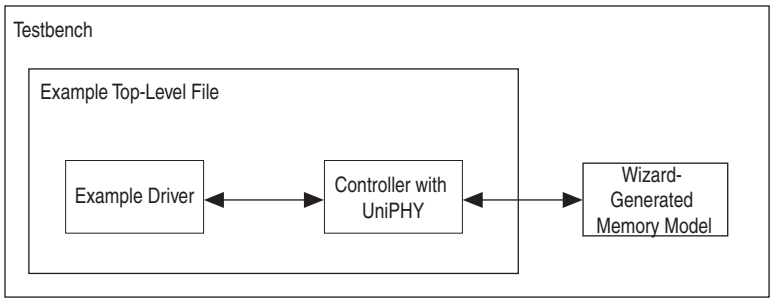
For a VHDL-only simulation, use the VHDL IP functional simulation model.

The testbench contains the following blocks:

- A synthesizable Avalon-MM example driver, which acts as a traffic generator block and implements a pseudo-random pattern of reads and writes to a parameterized number of addresses. The driver also monitors the data read from the memory to ensure it matches the written data and asserts a failure otherwise.
- An instance of the controller, which interfaces between the Avalon-MM interface and the AFI.
- The UniPHY IP, which serves as an interface between the memory controller and external memory device(s) to perform read and write operations to the memory.
- A memory model, which acts as a generic model that adheres to the memory protocol specifications. Memory vendors also provide simulation models for specific memory components that can be downloaded from their websites. This block is available in Verilog HDL only.

Figure 7–1 shows the testbench and the example top-level file.

**Figure 7–1. Testbench and Example Top-Level File**



# Example Driver

The example driver for Avalon-MM memory interfaces generates Avalon-MM traffic on an Avalon-MM master interface. As the read and write traffic is generated, the expected read response is stored internally and compared to the read responses as they arrive. If all reads report their expected response, the pass signal is asserted; however, if any read responds with unexpected data a fail signal is asserted.

Each operation generated by the driver is a single write or block of writes followed by a single read or block of reads to the same addresses, which allows the driver to precisely determine the data that should be expected when the read data is returned by the memory interface. The driver comprises a traffic generation block, the Avalon-MM interface and a read comparison block. The traffic generation block generates addresses and write data, which are then sent out over the Avalon-MM interface. The read comparison block compares the read data received from the Avalon-MM interface to the write data from the traffic generator. If at any time the data received is not the expected data, the read comparison block records the failure, finishes reading all the data, and then signals that there is a failure and the driver enters a fail state. If all patterns have been generated and compared successfully, the driver enters a pass state.

Within the driver, there are the following main states:

- Generation of individual read and writes
- Generation of block read and writes
- The pass state
- The fail state

Within each of the generation states there are the following substates:

- Sequential address generation
- Random address generation
- Mixed sequential and random address generation

For each of the states and substates, the order and number of operations generated for each substate is parameterizable—you can decide how many of each address pattern to generate, or can disable certain patterns entirely if you want. The sequential and random interleave substate takes in additions to the number of operations to generate. An additional parameter specifies the ratio of sequential to random addresses to generate randomly.

## Read and Write Generation

The traffic generator block can perform individual or block read and write generation.

### Individual Read and Write Generation

During the individual read and write generation stage of the driver, the traffic generation block generates individual write followed by individual read Avalon-MM transactions, where the address for the transactions are chosen according to the specific substate. The width of the Avalon-MM interface is a global parameter for the driver, but each substate can have a parameterizable range of burst lengths for each operation.

### Block Read and Write Generation

During the block read and write generation state of the driver, the traffic generator block generates a parameterizable number of write operations followed by the same number of read operations. The specific addresses generated for the blocks are chosen by the specific substates. The burst length of each block operation can be parameterized by a range of acceptable burst lengths.

## Address and Burst Length Generation

The traffic generator block can perform sequential or random addressing.

### Sequential Addressing

The sequential addressing substate defines a traffic pattern where addresses are chosen in sequential order starting from a user definable address. The number of operations in this substate is parameterizable.

### Random Addressing

The random addressing substate defines a traffic pattern where addresses are chosen randomly over a parameterizable range. The number of operations in this substate is parameterizable.

### Sequential and Random Interleaved Addressing

The sequential and random interleaved addressing substate defines a traffic pattern where addresses are chosen to be either sequential or random based on a parameterizable ratio. The acceptable address range is parameterizable as is the number of operations to perform in this substate.

## Example Driver Signals

Table 7–1 lists the signals used by the example driver.

**Table 7–1. Driver Signals (Part 1 of 2)**

| Signal | Width | Signal Type |
|---|---|---|
| clk | | |
| reset_n | | |
| avl_ready | | avl_ready |
| avl_write_req | | avl_write_req |
| avl_read_req | | avl_read_req |
| avl_addr | 24 | avl_addr |
| avl_size | 3 | avl_size |
| avl_wdata | 72 | avl_wdata |
| avl_rdata | 72 | avl_rdata |
| avl_rdata_valid | | avl_rdata_valid |
| pnf_per_bit | | pnf_per_bit |
| pnf_per_bit_ persist | | pnf_per_bit_persist |

**Table 7–1. Driver Signals (Part 2 of 2)**

| Signal | Width | Signal Type |
|---|---|---|
| pass | | pass |
| fail | | fail |
| test_complete | | test_complete |

## Example Driver Add-Ons

Some optional components that can be useful for verifying aspects of the controller and PHY operation are generated in conjunction with certain user-specified options. These add-on components are self-contained, and are not part of the controller or PHY, nor the example driver.

### User Refresh Generator

The user refresh generator sends refresh requests to the memory controller when user refresh is enabled. The memory controller returns an acknowledgement signal and then issues the refresh command to the memory device.

The user refresh generator is created when you turn on **Enable User Refresh** under **Controller Settings** on the **General Settings** tab of the parameter editor. The user refresh generator is instantiated by **example_top_v**. and resides in the example_project subdirectory.

### Refresh Monitor

As its name implies, the refresh monitor monitors refresh commands from the controller and verifies that those commands conform to the necessary refresh timing parameters.

The refresh monitor is created when you turn on **Enable User Refresh** under **Controller Settings** on the **General Settings** tab of the parameter editor. The refresh monitor is instantiated by **example_top_tb.v** and resides in **refresh_monitor.sv** in the **rtl_sim** subdirectory.

### Data Corrupter

The data corrupter intercepts read data in the memory interface bus and introduces errors to that data to test the error detection function in the memory controller. Both the rate of error injection and the number of error bits are configurable (although the per-byte parity protection feature supports only 1 bit error detection).

The data corrupter employs four types of error injection:

- per-bit data corruption in a single memory burst
- per-byte corruption in a single memory burst
- per-bit all-burst corruption
- per-byte all burst corruption

Throughout the four types of error injection tests, the data corrupter exercises a walking-one pattern to confirm correctness.

The data corrupter is created when you turn on **Enable Error Detection Parity** under **Controller Settings** on the **General Settings** tab of the parameter editor. The data corrupter resides in **data_corrupter.sv** in the **rtl_sim** subdirectory.

Altera defines read and write latencies in terms of memory clock cycles. These latencies apply to supported device families (Table 1–2 on page 1–2). There are two types of latencies that exists while designing with memory controllers—read and write latencies, which have the following definitions:

- Read latency—the amount of time it takes for the read data to appear at the local interface after initiating the read request.
- Write latency—the amount of time it takes for the write data to appear at the memory interface after initiating the write request.

For a half-rate controller, the local side frequency is half of the memory interface frequency. For a full-rate controller, the local side frequency is equal to the memory interface frequency.

Table 8–1 shows the latency in full rate memory clock cycles.

**Table 8–1. Latency (In Full-Rate Memory Clock Cycles) *(Note 2)***

| Rate | Controller Address and Command *(3)* | PHY Address and Command | Memory Maximum Read | PHY Read Return | Round Trip | Round Trip without Memory |
|---|---|---|---|---|---|---|
| Full | 1 | 1 | 1.5, 2.0, 2.5 | RL 1.5: 4.5<br>RL 2.0: 4.0<br>RL 2.5: 4.5 *(1)* | RL 1.5: 8<br>RL 2.0: 8<br>RL 2.5: 9 *(1)* | RL 1.5: 6.5<br>RL 2.0: 6.0<br>RL 2.5: 6.5 *(1)* |
| Half | 2 | 2 | 1.5, 2.0, 2.5 | RL 1.5: 6.5<br>RL 2.0: 6.0<br>RL 2.5: 7.5 *(1)* | RL 1.5: 12<br>RL 2.0: 12<br>RL 2.5: 14 *(1)* | RL 1.5: 10.5<br>RL 2.0: 10.0<br>RL 2.5: 11.5 *(1)* |

**Notes to Table 8–1:**

(1) RL = read latency.

(2) Latency is the number of cycles between the first register of the current stage capturing cmd/data, and the first register in the next stage capturing cmd/data.

(3) Latency shown is best case, for maximum performance specifications. Latency may be higher due to protocol requirements; controller latency may be lower for slower frequencies.

# Variable Controller Latency

The variable controller latency feature allows you to take advantage of lower latency for variations designed to run at lower frequency. When deciding whether to vary the controller latency from the default value of 1, be aware of the following considerations:

- Reduced latency can help acheive a reduction in resource usage and clock cycles in the controller, but might result in lower $f_{MAX}$.
- Increased latency can help acheive greater $f_{MAX}$, but might consume more clock cycles in the controller and result in increased resource usage.

If you select a latency value that is inappropriate for the target frequency, the system displays a warning message in the text area at the bottom of the parameter editor.

You can change the controller latency by altering the value of the **Controller Latency** setting in the **Controller Settings** section of the **General Settings** tab of the QDR II and QDR II+ SRAM controller with UniPHY parameter editor.

# 9. Timing Diagrams

The timing diagrams in this chapter are for QDR II SRAM with the following parameters:

- ×18
- Half rate
- Burst length 4
- Latency 2.5

Figure 9–1 shows back-to-back write to addresses 0 and 1.

You can set the `avl_w_size` to `0x2` and hold `avl_w_addr` constant at `0x0` to perform the same back-to-back write.

**Figure 9–1. Back-to-Back Writes**

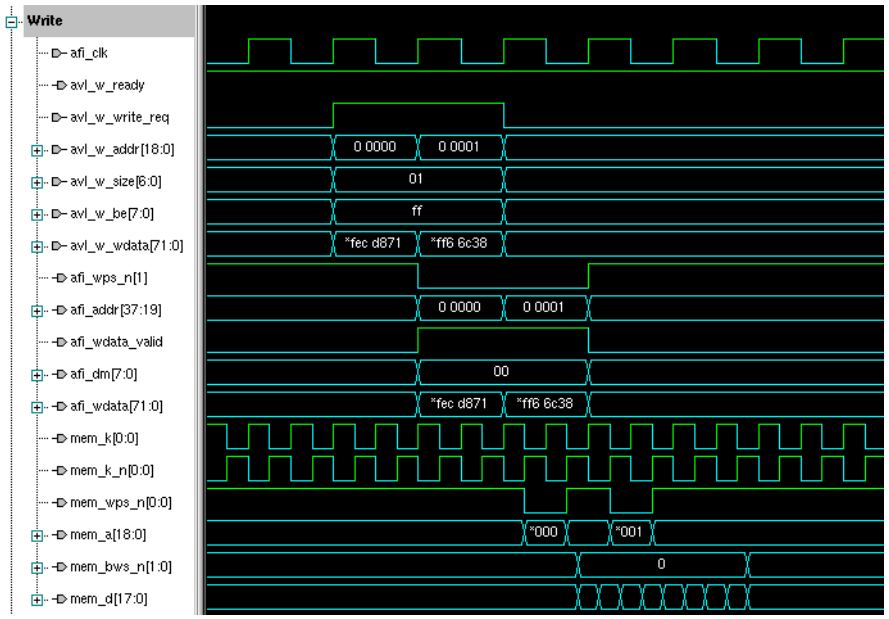
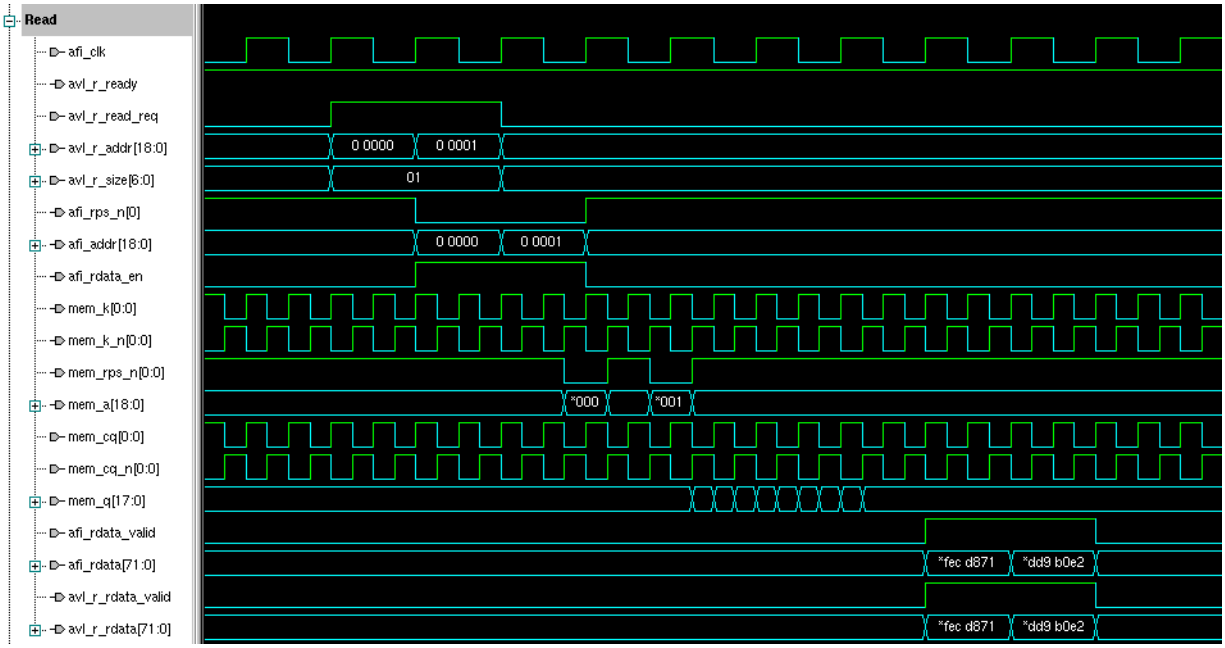



Figure 9–2 shows back-to-back read from addresses 0 and 1.

You can set the `avl_w_size` to `0x2` and hold `avl_w_addr` constant at `0x0` to perform the same back-to-back read.

**Figure 9–2. Back-to-Back Reads**



This chapter provides additional information about the document and Altera.

## Document Revision History

The following table shows the revision history for this document.

| Date | Version | Changes |
|---|---|---|
| December 2010 | 2.1 | ■ Updated Device Family Support, Features list, and Resource Utilization tables<br>■ Updated Design Flows, added new Generated Files information<br>■ Added information to HardCopy Migration Design Guidelines<br>■ Added new Parameter Settings chapter<br>■ Updated Reset and Clock Generation, Write Datapath, and Read Datapath information<br>■ Added The DLL and PLL Sharing Interface and Using a Custom Controller information<br>■ Updated Latency data |
| July 2010 | 2.0 | Updated for the Altera Complete Design Suite version 10.0 release. |
| February 2010 | 1.2 | Corrected typos. |
| January 2010 | 1.1 | Updated features. |
| November 2009 | 1.0 | First published. |

## How to Contact Altera

To locate the most up-to-date information about Altera products, refer to the following table.

| Contact *(1)* | Contact Method | Address |
|---|---|---|
| Technical support | Website | www.altera.com/support |
| Technical training | Website | www.altera.com/training |
| | Email | custrain@altera.com |
| Product literature | Website | www.altera.com/literature |
| Non-technical support (General) | Email | nacomp@altera.com |
| (Software Licensing) | Email | authorization@altera.com |

**Note to Table:**

(1) You can also contact your local Altera sales office or sales representative.

# Typographic Conventions

The following table shows the typographic conventions this document uses.

| Visual Cue | Meaning |
|---|---|
| **Bold Type with Initial Capital Letters** | Indicate command names, dialog box titles, dialog box options, and other GUI labels. For example, **Save As** dialog box. For GUI elements, capitalization matches the GUI. |
| **bold type** | Indicates directory names, project names, disk drive names, file names, file name extensions, software utility names, and GUI labels. For example, **\qdesigns** directory, **D:** drive, and **chiptrip.gdf** file. |
| *Italic Type with Initial Capital Letters* | Indicate document titles. For example, *Stratix IV Design Guidelines*. |
| *italic type* | Indicates variables. For example, *n* + 1.<br>Variable names are enclosed in angle brackets (< >). For example, *<file name>* and *<project name>***.pof** file. |
| Initial Capital Letters | Indicate keyboard keys and menu names. For example, the Delete key and the Options menu. |
| "Subheading Title" | Quotation marks indicate references to sections within a document and titles of Quartus II Help topics. For example, "Typographic Conventions." |
| `Courier type` | Indicates signal, port, register, bit, block, and primitive names. For example, `data1`, `tdi`, and `input`. The suffix `n` denotes an active-low signal. For example, `resetn`.<br>Indicates command line commands and anything that must be typed exactly as it appears. For example, `c:\qdesigns\tutorial\chiptrip.gdf`.<br>Also indicates sections of an actual file, such as a Report File, references to parts of files (for example, the AHDL keyword `SUBDESIGN`), and logic function names (for example, `TRI`). |
| ↵ | An angled arrow instructs you to press the Enter key. |
| 1., 2., 3., and<br>a., b., c., and so on | Numbered steps indicate a list of items when the sequence of the items is important, such as the steps listed in a procedure. |
| ■ ■ ■ | Bullets indicate a list of items when the sequence of the items is not important. |
|  | The hand points to information that requires special attention. |
|  | A question mark directs you to a software help system with related information. |
|  | The feet direct you to another document or website with related information. |
|  | A caution calls attention to a condition or possible situation that can damage or destroy the product or your work. |
|  | A warning calls attention to a condition or possible situation that can cause you injury. |
|  | The envelope links to the Email Subscription Management Center page of the Altera website, where you can sign up to receive update notifications for Altera documents. |

# External Memory Interface Handbook Volume 3

## Section IV. RLDRAM II Controller with UniPHY IP User Guide

101 Innovation Drive
San Jose, CA 95134
www.altera.com

EMI_RLDRAM_II_UG-2.1

Document last updated for Altera Complete Design Suite version: 10.1
Document publication date: December 2010

Subscribe

# Contents

# 1. About This IP

The Altera RLDRAM II controller with UniPHY provides a simplified interface to industry-standard RLDRAM II.

The RLDRAM II controller with UniPHY offers full-rate or half-rate RLDRAM II interfaces. The UniPHY IP is an interface between a memory controller and memory devices and performs read and write operations to the memory. The UniPHY IP creates the datapath between the memory device and the memory controller and user logic in various Altera devices.

The Quartus II software generates an example top-level project, consisting of an example driver, and your RLDRAM II controller custom variation. The controller instantiates an instance of the UniPHY.

The example top-level project is a fully-functional design that you can simulate, synthesize, and use in hardware. The example driver is a self-test module that issues read and write commands to the controller and checks the read data to produce the pass or fail and test-complete signals.

For device families not supported by the UniPHY IP, use the Altera legacy integrated static datapath and controller MegaCore functions.

You can, alternatively, create your own memory interface datapath using the ALTDLL and ALTDQ_DQS megafunctions, available in the Quartus II software, but you must then consider all of the aspects of the design including timing analysis and design constraints.

The UniPHY IP offers the Altera PHY interface (AFI). The AFI results in a simple connection between the PHY and controller.

## Release Information

Table 1–1 provides information about this release of the RLDRAM II controller with UniPHY.

**Table 1–1. Release Information**

| Item | Description |
|---|---|
| Version | 10.1 |
| Release Date | December 2010 |
| Ordering Codes | IP-RLDII/UNI |
| Vendor ID | 6AF7 |

Altera verifies that the current version of the Quartus II software compiles the previous version of each MegaCore function. The *MegaCore IP Library Release Notes and Errata* report any exceptions to this verification. Altera does not verify compilation with MegaCore function versions older than one release.

# Device Family Support

IP cores provide the following levels of support for target Altera device families:

- For FPGA support:
  - Preliminary—verified with preliminary timing models for this device
  - Final—verified with final timing models for this device
- For ASIC devices (HardCopy families)
  - HardCopy companion—verified with preliminary timing models for HardCopy companion device
  - HardCopy compilation—verifed with final timing models for HardCopy device

Table 1–2 shows the level of support offered by the RLDRAM II controller to each of the Altera device families.

For information about supported clock rates for external memory interfaces, refer to the *External Memory Interface System Specifications* section in volume 1 of the *External Memory Interface Handbook*.

**Table 1–2. Device Family Support**

| Device Family | Support |
|---|---|
| Arria II GZ | Preliminary |
| Stratix® III (except $V_{cc}$ = 0.9v) | Final |
| Stratix IV | Final |
| Stratix V | Preliminary |
| HardCopy III | HardCopy companion |
| HardCopy IV | HardCopy companion |
| Other device families | No support |

# Features

Table 1–3 summarizes key feature support for the RLDRAM II Controller with UniPHY.

**Table 1–3. Key Feature Support for RLDRAM II Controller with UniPHY (Part 1 of 2)**

| Key Feature | RLDRAM II UniPHY |
|---|---|
| High-performance controller (HPC) | — |
| High-performance controller II (HPC II) | ✓ |
| Half-rate core logic and user interface | ✓ |
| Full-rate core logic and user interface | ✓ |
| Burst length (half-rate) | 4 or 8 |
| Burst length (full-rate) | 2, 4, or 8 |
| Reduced controller latency (1) (2) | ✓ |
| Read latency | — |

**Table 1–3. Key Feature Support for RLDRAM II Controller with UniPHY (Part 2 of 2)**

| Key Feature | RLDRAM II UniPHY |
|---|---|
| Maximum data width | 72 bits |
| ODT (in memory device) | ✓ |

**Notes for Table 1–3:**

(1) The maximum achievable clock rate when reduced controller latency is selected must be attained through Quatrus II software timing analysis of your complete design.

(2) Not available in Arria II GX devices.

# Unsupported Features

This section summarizes unsupported features for the RLDRAM II Controller with UniPHY:

- Arria II GX
- Cyclone III
- Cyclone IV
- Depth expansion
- Dynamic read latency change
- ECC
- Multicast write
- Multiplexed addressing
- Multiple chip select
- RLDRAM II SIO devices
- Separate I/O
- Stratix III devices with $V_{cc}$=0.9v
- Timing simulation
- VHDL simulation support for Arria II GX, Arria II GZ, Stratix III, Stratix IV, and Stratix V devices

# MegaCore Verification

Altera has carried out extensive random, directed tests with functional test coverage using industry-standard models to ensure the functionality of the RLDRAM II controller with UniPHY.

# Resource Utilization

This section lists resource utilization for the QDR II and QDR II+ SRAM controllers with UniPHY for supported device families. Resource utilizations are derived with all parameters at their default values.

Table 1–4 shows the typical resource usage of the RLDRAM II controller with UniPHY in the Quartus II software version 10.1 for Arria II GZ, Stratix III, Stratix IV, and Stratix V devices.

**Table 1–4. Resource Utilization in Arria II GZ, Stratix III, Stratix IV, and Stratix V Devices *(Note 1)***

| PHY Rate | Memory Width (Bits) | Combinational ALUTS | Logic Registers | Memory (Bits) | M9K Blocks |
|---|---|---|---|---|---|
| Half | 9 | 829 | 763 | 288 | 1 |
| | 18 | 1145 | 1147 | 576 | 2 |
| | 36 | 1713 | 1861 | 1152 | 4 |
| Full | 9 | 892 | 839 | 288 | 1 |
| | 18 | 1182 | 1197 | 576 | 1 |
| | 36 | 1678 | 1874 | 1152 | 2 |

**Note to Table 1–4:**

(1) Half-rate designs use the same amount of memory as full-rate designs, but the data is organized in a different way (half the width, double the depth) and the design may need more M9K resources.

# System Requirements

The RLDRAM II controller with UniPHY is part of the MegaCore IP Library, which is distributed with the Quartus II software.

For system requirements and installation instructions, refer to *Altera Software Installation & Licensing*.

# 2. Getting Started

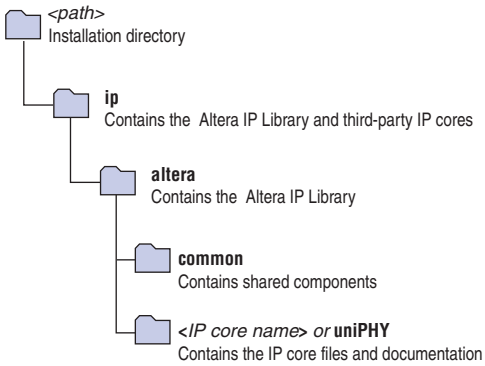
This chapter provides a general overview of the Altera IP core design flow to help you quickly get started with any Altera IP core. The Altera IP Library is installed as part of the Quartus II installation process. You can select and parameterize any Altera IP core from the library. Altera provides an integrated parameter editor that allows you to customize IP cores to support a wide variety of applications. The parameter editor guides you through the setting of parameter values and selection of optional ports. The following sections describe the general design flow and use of Altera IP cores.

## Installation and Licensing

The Altera IP Library is distributed with the Quartus II software and downloadable from the Altera website (www.altera.com).

Figure 2–1 shows the directory structure after you install an Altera IP core, where *<path>* is the installation directory. The default installation directory on Windows is **C:\altera\<***version number***>**; on Linux it is **/opt/altera<***version number***>.**

**Figure 2–1. IP core Directory Structure**




You can evaluate an IP core in simulation and in hardware until you are satisfied with its functionality and performance. Some IP cores require that you purchase a license for the IP core when you want to take your design to production. After you purchase a license for an Altera IP core, you can request a license file from the Altera Licensing page of the Altera website and install the license on your computer. For additional information, refer to *Altera Software Installation and Licensing*.

## Design Flows

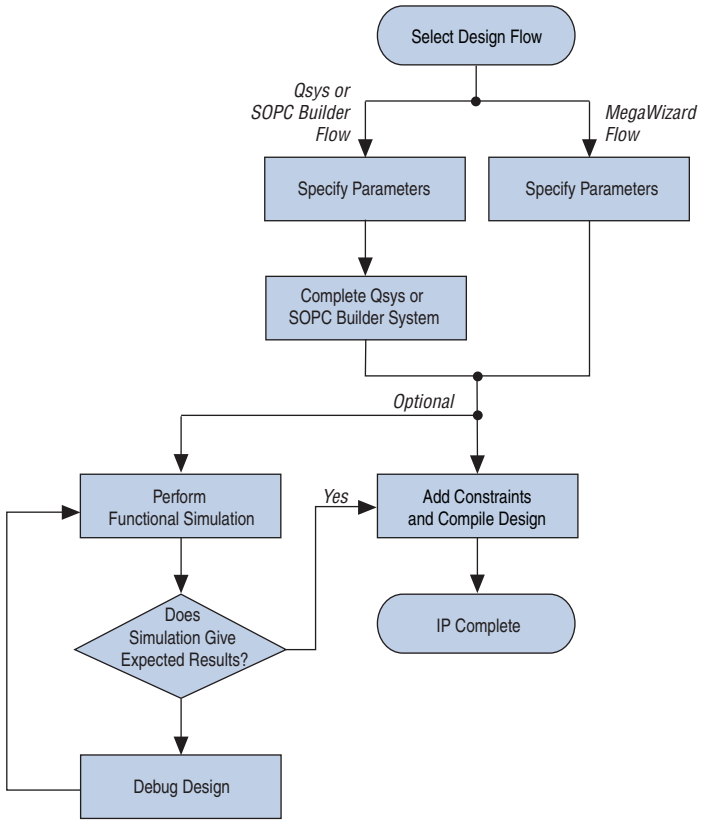
You can use the following flow(s) to parameterize Altera IP cores:

- MegaWizard Plug-In Manager Flow
- SOPC Builder Flow
- Qsys Flow

CAUTION

Altera's Qsys system integration tool is now available as beta for evaluation in the Quartus II software subscription edition version 10.1. Altera does not recommend using the beta release of Qsys in the Quartus II software version 10.1 for designs that are close to completion and are meeting design requirements. Before using Qsys, review the *Quartus II Software Version 10.1 Release Notes* and *AN 632: SOPC Builder to Qsys Migration Guidelines* for known issues and limitations. To submit general feedback or technical support on the beta release of Qsys, submit a service request through **mysupport.altera.com**. Alternatively, to submit general feedback, click **Feedback** on the Quartus II software Help menu.

**Figure 2–2. Design Flows**




The MegaWizard Plug-In Manager flow offers the following advantages:

- Allows you to parameterize an IP core variant and instantiate into an existing design
- For some IP cores, this flow generates a complete example design and testbench.

The SOPC Builder flow offer the following advantages:

- Generates simulation environment
- Allows you to integrate Altera-provided custom components
- Uses Avalon®memory-mapped (Avalon-MM) interfaces

The Qsys flow offers the following additional advantages over SOPC Builder:

- Provides visualization of hierarchical designs
- Allows greater performance through interconnect elements and pipelining
- Provides closer integration with the Quartus II software

# MegaWizard Plug-In Manager Flow

The MegaWizard Plug-In Manager flow allows you to customize your IP core and manually integrate the function into your design.

## Specifying Parameters

To specify IP core parameters with the MegaWizard Plug-In Manager, follow these steps:

1. Create a Quartus II project using the **New Project Wizard** available from the File menu.
2. In the Quartus II software, launch the **MegaWizard Plug-in Manager** from the Tools menu, and follow the prompts in the MegaWizard Plug-In Manager interface to create or edit a custom IP core variation.
3. To select a specific Altera IP core, click the IP core in the **Installed Plug-Ins** list in the MegaWizard Plug-In Manager.
4. Specify the parameters on the **Parameter Settings** pages. For detailed explanations of these parameters, refer to the "*Parameter Settings*" chapter in this document.

   Some IP cores provide preset parameters for specific applications. If you wish to use preset parameters, click the arrow to expand the **Presets** list, select the desired preset, and then click **Apply**. To modify preset settings, in a text editor edit the ***<installation directory>*\ip\altera\uniphy\lib*<IP core>*.qprs** file.

5. If the IP core provides a simulation model, specify appropriate options in the wizard to generate a simulation model.

   Altera IP supports a variety of simulation models, including simulation-specific IP functional simulation models and encrypted RTL models, and plain text RTL models. These are all cycle-accurate models. The models allow for fast functional simulation of your IP core instance using industry-standard VHDL or Verilog HDL simulators. For some cores, only the plain text RTL model is generated, and you can simulate that model.

   For more information about functional simulation models for Altera IP cores, refer to *Simulating Altera Designs* in volume 3 of the *Quartus II Handbook*.

   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.

6. If the parameter editor includes **EDA** and **Summary** tabs, follow these steps:
   a. Some third-party synthesis tools can use a netlist that contains the structure of an IP core but no detailed logic to optimize timing and performance of the design containing it. To use this feature if your synthesis tool and IP core support it, turn on **Generate netlist**.
   b. On the **Summary** tab, if available, select the files you want to generate. A gray checkmark indicates a file that is automatically generated. All other files are optional.

   ☞ If file selection is supported for your IP core, after you generate the core, a generation report (*<variation name>***.html)** appears in your project directory. This file contains information about the generated files.

7. Click the **Finish** button, the parameter editor generates the top-level HDL code for your IP core, and a simulation directory which includes files for simulation.

   ☞ The **Finish** button may be unavailable until all parameterization errors listed in the messages window are corrected.

8. Click **Yes** if you are prompted to add the Quartus II IP File (**.qip**) to the current Quartus II project. You can also turn on **Automatically add Quartus II IP Files to all projects**.

You can now integrate your custom IP core instance in your design, simulate, and compile. While integrating your IP core instance into your design, you must make appropriate pin assignments. You can create virtual pin to avoid making specific pin assignments for top-level signals while you are simulating and not ready to map the design to hardware.

For some IP cores, the generation process also creates a complete example design in the *<variation_name>***_example_design_fileset/example_project/** directory. This example demonstrates how to instantiate and connect the IP core.

☞ For information about the Quartus II software, including virtual pins and the MegaWizard Plug-In Manager, refer to Quartus II Help.

## Simulate the IP Core

You can simulate your IP core variation with the functional simulation model and the testbench or example design generated with your IP core. The functional simulation model and testbench files are generated in a project subdirectory. This directory may also include scripts to compile and run the testbench.

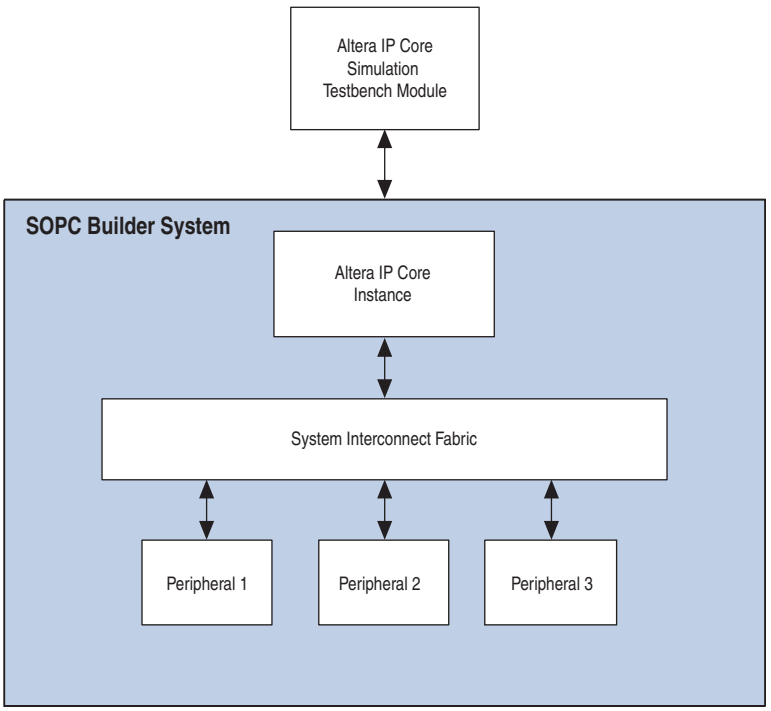
For a complete list of models or libraries required to simulate your IP core, refer to the scripts provided with the testbench.

For more information about simulating Altera IP cores, refer to *Simulating Altera Designs* in volume 3 of the *Quartus II Handbook*.

# SOPC Builder Design Flow

You can use SOPC Builder to build a system that includes your customized IP core. You easily can add other components and quickly create an SOPC Builder system. SOPC Builder automatically generates HDL files that include all of the specified components and interconnections. SOPC Builder defines default connections, which you can modify. The HDL files are ready to be compiled by the Quartus II software to produce output files for programming an Altera device. SOPC Builder generates a simulation testbench module for supported cores that includes basic transactions to validate the HDL files. Figure 2–3 shows a block diagram of an example SOPC Builder system.

**Figure 2–3. SOPC Builder System**




For more information about system interconnect fabric, refer to the *System Interconnect Fabric for Memory-Mapped Interfaces* and *System Interconnect Fabric for Streaming Interfaces* chapters in the *SOPC Builder User Guide* and to the *Avalon Interface Specifications*.

For more information about SOPC Builder and the Quartus II software, refer to the *SOPC Builder Features* and *Building Systems with SOPC Builder* sections in the *SOPC Builder User Guide* and to Quartus II Help.

## Specify Parameters

To specify IP core parameters in the SOPC Builder flow, follow these steps:

1. Create a new Quartus II project using the **New Project Wizard** available from the File menu.
2. On the Tools menu, click **SOPC Builder**.
3. For a new system, specify the system name and language.
4. On the **System Contents** tab, double-click the name of your IP core to add it to your system. The relevant parameter editor appears.
5. Specify the required parameters in the parameter editor. For detailed explanations of these parameters, refer to the *"Parameter Settings"* chapter in this document.

   Some IP cores provide preset parameters for specific applications. If you wish to use preset parameters, click the arrow to expand the **Presets** list, select the desired preset, and then click **Apply**. To modify preset settings, in a text editor edit the <*installation directory*>**\ip\altera\uniphy\lib**<*IP core*>**.qprs** file.

   If your design includes external memory interface IP cores, you must turn on **Generate power of two bus widths** on the **PHY Settings** tab when parameterizing those cores.

6. Click **Finish** to complete the IP core instance and add it to the system.

   The **Finish** button may be unavailable until all parameterization errors listed in the messages window are corrected.

## Complete the SOPC Builder System

To complete the SOPC Builder system, follow these steps:

1. Add and parameterize any additional components. Some IP cores include a complete SOPC Builder system design example.
2. Use the Connection panel on the **System Contents** tab to connect the components.
3. By default, clock names are not displayed. To display clock names in the **Module Name** column and the clocks in the **Clock** column in the **System Contents** tab, click **Filters** to display the **Filters** dialog box. In the **Filter** list, click **All.**
4. If you intend to simulate your SOPC builder system, on the **System Generation** tab, turn on **Simulation** to generate simulation files for your system.
5. Click **Generate** to generate the system. SOPC Builder generates the system and produces the <*system name*>**.qip** file that contains the assignments and information required to process the IP core or system in the Quartus II Compiler.
6. In the Quartus II software, click **Add/Remove Files in Project** and add the **.qip** file to the project.
7. Compile your design in the Quartus II software.

## Simulate the System

During system generation, you can specify whether SOPC Builder 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 and plain-text RTL design files that describe your system in the ModelSim simulation software.

For information about the latest Altera-supported simulation tools, refer to the *Quartus II Software Release Notes*.

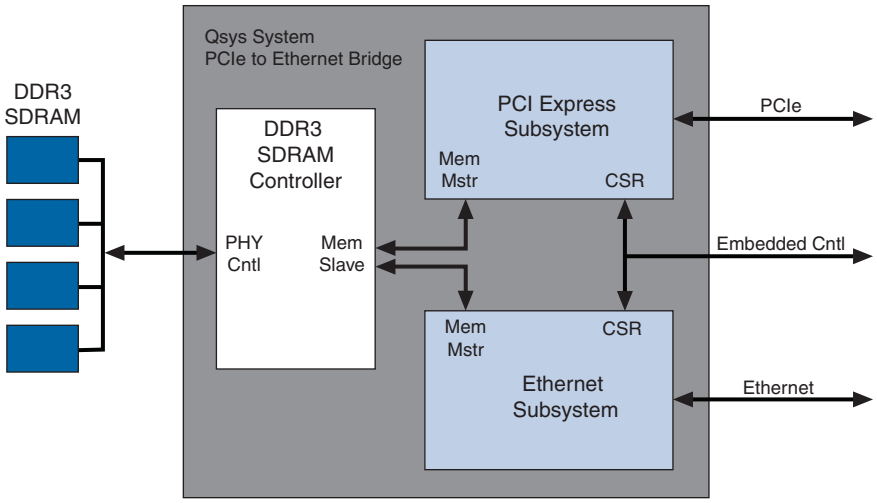
For information about simulating SOPC Builder systems, refer to the *SOPC Builder User Guide* and *AN 351: Simulating Nios II Embedded Processor Designs*.

For general information about simulating Altera IP cores, refer to *Simulating Altera Designs* in volume 3 of the *Quartus II Handbook*.

# Qsys System Integration Tool Design Flow

You can use the Qsys system integration tool to build a system that includes your customized IP core. You easily can add other components and quickly create a Qsys system. Qsys automatically generates HDL files that include all of the specified components and interconnections. In Qsys, you specify the connections you want. The HDL files are ready to be compiled by the Quartus II software to produce output files for programming an Altera device. Qsys generates Verilog HDL simulation models for the IP cores that comprise your system. Figure 2–4 shows a high level block diagram of an example Qsys system.

**Figure 2–4. Example Qsys System**



For more information about the Qsys system interconnect, refer to the *Qsys Interconnect* chapter in volume 1 of the *Quartus II Handbook* and to the *Avalon Interface Specifications*.

For more information about the Qsys tool and the Quartus II software, refer to the *System Design with Qsys* section in volume 1 of the *Quartus II Handbook* and to Quartus II Help.

## Specify Parameters

To specify parameters for your IP core using the Qsys flow, follow these steps:

1. Create a new Quartus II project using the **New Project Wizard** available from the File menu.
2. On the Tools menu, click **Qsys (Beta)**.
3. On the **System Contents** tab, double-click the name of your IP core to add it to your system. The relevant parameter editor appears.
4. Specify the required parameters in all tabs in the Qsys tool. For detailed explanations of these parameters, refer to the *"Parameter Settings"* chapter in this document.

   If your design includes external memory interface IP cores, you must turn on **Generate power of two bus widths** on the **PHY Settings** tab when parameterizing those cores.

   Some IP cores provide preset parameters for specific applications. If you wish to use preset parameters, click the arrow to expand the **Presets** list, select the desired preset, and then click **Apply**. To modify preset settings, in a text editor edit the ***<installation directory>*\ip\altera\uniphy\lib*<IP core>*.qprs** file.

5. Click **Finish** to complete the IP core instance and add it to the system.

   The **Finish** button may be unavailable until all parameterization errors listed in the messages window are corrected.

## Complete the Qsys System

To complete the Qsys system, follow these steps:

1. Add and parameterize any additional components.
2. Connect the components using the Connection panel on the **System Contents** tab.
3. In the **Export As** column, enter the name of any connections that should be a top-level Qsys system port. If the **Export As** column is not present, click the **Project Settings** tab and turn off **Use SOPC Builder port naming**.
4. If you intend to simulate your Qsys system, on the **Generation** tab, turn on one or more options under **Simulation** to generate desired simulation files.
5. If your system is not part of a Quartus II project and you want to generate synthesis RTL files, turn on **Create synthesis RTL files**.

6. Click **Generate** to generate the system. Qsys generates the system and produces the <*system name*>**.qip** file that contains the assignments and information required to process the IP core or system in the Quartus II Compiler.
7. In the Quartus II software, click **Add/Remove Files in Project** and add the **.qip** file to the project.
8. Compile your project in the Quartus II software.

### Simulate the System

During system generation, Qsys generates a functional simulation model—or example design that includes a testbench—which you can use to simulate your system in any Altera-supported simulation tool.

For information about the latest Altera-supported simulation tools, refer to the *Quartus II Software Release Notes*.

For general information about simulating Altera IP cores, refer to *Simulating Altera Designs* in volume 3 of the *Quartus II Handbook*.

For information about simulating Qsys systems, refer to the *System Design with Qsys* section in volume 1 of the *Quartus II Handbook*.

## HardCopy Migration Design Guidelines

If you intend to target your design to a HardCopy® device, ensure you use the following design guidelines:

- On the **General Settings** page of the **DDR2 SDRAM Controller with UniPHY** or **DDR3 SDRAM Controller with UniPHY** MegaWizard, turn on **HardCopy Compatibility Mode**, and then specify whether the **Reconfigurable PLL Location** is **Top_Bottom** or **Left_Right**.

  Altera recommends that you set the **Reconfigurable PLL Location** to the same side as your memory interface.

  When turned on, the **HardCopy Compatibility Mode** option enables run-time reconfiguration for all phase-locked loops (PLLs) and delay-locked loops (DLLs) instantiated in memory interfaces that are configured in PLL and DLL masters, and brings the necessary reconfiguration signals to the top level of the design.

  "Top-Level HardCopy Migration Signals" on page 6–12 lists the top-level signals generated for HardCopy migration.

- Enable run-time reconfiguration mode for all PLLs and DLLs instantiated in interfaces that are configured in PLL and DLL slaves.

For information about PLL megafunctions, refer to the *Phase-Locked Loop (ALTPLL) Megafunction User Guide* and the *Phase-Locked Loops Reconfiguration (ALTPLL_RECONFIG) Megafunctions User Guide*. For information about DLL megafunctions, refer to the *ALTDLL and ALTDQ_DQS Megafunctions User Guide*.

- Ensure that you place all memory interface pins close together. If, for example, address pins are located far away from data pins, closing timing might be difficult.

You can use the example top-level project that is generated when you turn on **HardCopy Migration** as a guide to help you connect the necessary signals in your design.

## Differences in UniPHY IP Generated with HardCopy Migration Support

When you generate a UniPHY memory interface for HardCopy device support, certain features in the IP are enabled that do not exist when you generate the IP core for only the FPGA. This section discusses those additional enabled features.

### ROM Loader for Designs Using Nios II Sequencer

An additional ROM loader is intantiated in the design for UniPHY designs that use the Nios II sequencer. The Nios II sequencer instruction code resides in RAM on either the HardCopy or FPGA device.

When you target only an FPGA device, the RAM is initialized when the device is programmed; however, HardCopy devices are not programmed and therefore the RAM cannot be initialized in this fashion. Instead, the Nios II sequencer instruction code must be stored in an external, non-volatile, ROM that loads the Nios II sequencer RAM through a ROM loader. You must attach the ROM loader to the appropriate pins connected to the external non-volatile ROM.

Table 2–1 summarizes the ports exposed at the top level of the PHY+Controller wrapper to expose the ROM loader utilized by the Nios II-based sequencer within the DDR2 or DDR3 PHY.

**Table 2–1. Top-level Ports that Connect to External ROM for Loading Nios II Code Memory (Part 1 of 2)**

| Port Name | Direction | Description |
|---|---|---|
| hc_rom_config_clock | Input | Write clock for the ROM loader. This clock is the write clock for the Nios II code memory. |
| hc_rom_config_datain | Input | Data input from external ROM. |
| hc_rom_config_rom_data_ready | Input | Asserts to the code memory loader that the word of memory is ready to be loaded. |
| hc_rom_config_init | Input | Signals that the Nios II code memory is being loaded from the external ROM. |
| hc_rom_config_init_busy | Output | Remains asserted throughout initialization and becomes inactive when initialization is complete. soft_reset_n can be issued after hc_rom_config_init_busy is deasserted. |

**Table 2–1. Top-level Ports that Connect to External ROM for Loading Nios II Code Memory (Part 2 of 2)**

| Port Name | Direction | Description |
|---|---|---|
| hc_rom_config_rom_rden | Output | Read-enable signal that connects to the external ROM. |
| hc_rom_config_rom_address | Output | ROM address that connects to the external ROM. |

## PLL/DLL Run-time Reconfiguration

The PLLs and DLLs in the HardCopy design have run-time reconfiguration enabled—provided that they are not in PLL/DLL slave mode.

When the PLLs and DLLs are generated with reconfiguration enabled, there are extra signals that must be connected and driven by user logic. In the example design generated during IP core generation, the PLL/DLL reconfiguration signals are brought to the top level and connected to constants, as shown in Figure 2–5.

For information about PLL megafunctions and reconfiguration, refer to the *Phase-Locked Loop (ALTPLL) Megafunction User Guide* and the *Phase-Locked Loops Reconfiguration (ALTPLL_RECONFIG) Megafunctions User Guide*.

**Figure 2–5. HardCopy UniPHY Example Design**

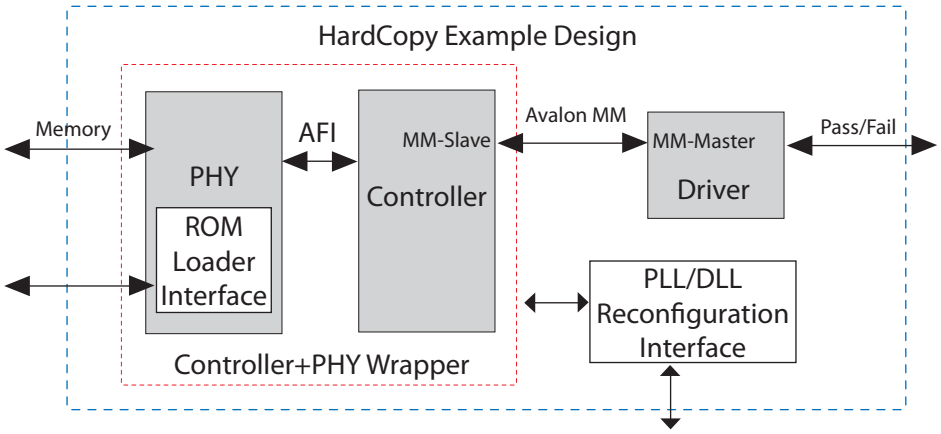



Table 2–2 summarizes the DLL reconfiguration ports exposed at the top level of the Controller+PHY.

**Table 2–2. DLL Reconfiguration Ports Exposed at Top-Level of Controller+PHY Wrapper (Part 1 of**

| Port Name | Direction | Description |
|---|---|---|
| hc_dll_config_dll_offset_ctrl_addnsub | Input | Addition/subtraction control port for the DLL. This port controls if the delay-offset setting on hc_dll_config_dll_offset_ctrl_offset is added or subtracted. |

**Table 2–2. DLL Reconfiguration Ports Exposed at Top-Level of Controller+PHY Wrapper (Part 2 of**

| Port Name | Direction | Description |
|---|---|---|
| hc_dll_config_dll_offset_ctrl_offset | Input | Offset input setting for the PLL. This is a Gray-coded offset that is added or subtracted from the current value of the DLL's delay chain. |
| hc_dll_config_dll_offset_ctrl_offsetctrlout | Output | The registered and Gray-coded value of the current delay-offset setting. |

Table 2–3 summarizes the ports exposed at the top level of the Controller and PHY wrapper to allow PLL reconfiguration.

**Table 2–3. PLL Reconfiguration Ports Exposed at the Top-Level of Controller+PHY Wrapper**

| Port Name | Direction | Description |
|---|---|---|
| hc_pll_config_configupdate | Input | Control signal to enable PLL reconfiguration. (Applies to RLDRAMII and QDRII only, the phase reconfiguration feature for DDR2/3 is included in the CSR port.) |
| hc_pll_config_phasecounterselect | Input | Specifies the counter select for dynamic phase adjustment. (Applies to RLDRAMII and QDR II only.) |
| hc_pll_config_phasestep | Input | Specifies the phase step for dynamic phase shifting. (Applies to RLDRAMII and QDR II only.) |
| hc_pll_config_phaseupdown | Input | Specifies if the phase adjustment should be up or down. (Applies to RLDRAMII and QDR II only.) |
| hc_pll_config_scanclk | Input | PLL reconfiguration scan chain clock. |
| hc_pll_config_scanclkena | Input | Clock enable port of the hc_pll_config_scanclk clock. |
| hc_pll_config_scandata | Input | Serial input data for the PLL reconfiguration scan chain. |
| hc_pll_config_phasedone | Output | When asserted, this signal indicates to core logic that phase adjustment is completed and that the PLL is ready to act on a possible second adjustment pulse. |
| hc_pll_config_scandataout | Output | The data output of the serial scan chain. |
| hc_pll_config_scandone | Output | Asserted when the scan chain write operation is in progress and is deasserted when the write operation is complete. |

To facilitate placement and timing closure and help compensate for PLLs adjacent to I/Os and vertical I/O overhang issues that can occur when targeting HardCopy III and HardCopy IV devices, an additional pipeline stage is added to the write path in the RTL when you turn on **HardCopy Compatibility**. The additional pipeline stage is added in all cases, except when CAS write latency equals 2 (for DDR3) or CAS latency equals 3 (for DDR2), where the additional pipeline stage is not required to meet timing requirements. The additional pipeline stage does not affect the overall latency of the controller.

For information about HardCopy issues such as vertical I/O overhang, PLLs adjacent to I/Os, and timing closure, refer to HardCopy III Device I/O Features in the *HardCopy III Device Handbook, Volume 1*, and HardCopy IV Device I/O Features in the *HardCopy IV Device Handbook, Volume 1*.

# Generated Files

When you complete the IP generation flow, there are generated files created in your project directory. The directory structure created varies somewhat, depending on the tool used to parameterize and generate the IP

The PLL parameters are statically defined in the <*variation_name*>**_parameters.tcl** at generation time. To ensure timing constraints and timing reports are correct, when you use the GUI to make changes to the PLL component, apply those changes to the PLL parameters in this file.

## MegaWizard Plug-in Manager Flow

The tables in this section list the generated directory structure and key files of interest to users, resulting from the MegaWizard Plug-in Manager flow.

### Synthesis

Table 2–4 lists the generated directory structure and key files created by the synthesis flow with the MegaWizard Plug-in Manager.

**Table 2–4. Generated Directory Structure and Key Files—MegaWizard Plug-In Manager Synthesis Flow**

| Directory | File Name | Description |
|---|---|---|
| <*working_dir*>/ | <*variation_name*>.qip | QIP file which refers to all generated files in the synthesis fileset. |
| <*working_dir*>/ | <*variation_name*>.v (for Verilog), or <*variation_name*>.vhd (for VHDL) | Top-level wrapper for synthesis files. |
| <*working_dir*>/<*variation_name*>/ | <*variation_name*>_<*stamp*>.v (1) | UniPHY top-level wrapper. |
| <*working_dir*>/<*variation_name*>/ | <*variation_name*>_<*stamp*>_*.v (1) | UniPHY Verilog RTL files. |
| <*working_dir*>/<*variation_name*>/ | <*variation_name*>_<*stamp*>_*.sv (1) | UniPHY SystemVerilog RTL files. |
| <*working_dir*>/<*variation_name*>/ | <*variation_name*>_<*stamp*>.sdc (1) | Synopsys constraints file. |
| <*working_dir*>/<*variation_name*>/ | <*variation_name*>_<*stamp*>.ppf (1) | Pin Planner file. |
| <*working_dir*>/<*variation_name*>/ | <*variation_name*>_<*stamp*>_pin_assignments.tcl (1) | Pin constraints script to be run after synthesis. |
| <*working_dir*>/<*variation_name*>/ | <*variation_name*>_<*stamp*>_*.tcl (1) | Other Tcl scripts. |
| <*working_dir*>/<*variation_name*>/ | <*variation_name*>_<*stamp*>_readme.txt (1) | Readme text file. |

**Note to Table 2–4:**

(1) <*stamp*> is a unique identifier determined by the MegaWizard Plug-in Manager at generation time.

## Simulation

Table 2–5 lists the generated directory structure and key files created by the Verilog simulation flow with the MegaWizard Plug-in Manager.

**Table 2–5. Generated Directory Structure and Key Files—MegaWizard Plug-In Manager Simulation Flow (Verilog)**

| Directory | File Name | Description |
|---|---|---|
| <*working_dir*>/<*variation_name*>_sim/ | <*variation_name*>.v (for Verilog), or <*variation_name*>.vho (for VHDL) | UniPHY top-level wrapper. |
| <*working_dir*>/<*variation_name*>_sim/ | <*variation_name*>_*.v | UniPHY Verilog RTL files. |
| <*working_dir*>/<*variation_name*>_sim/ | <*variation_name*>_*.sv | UniPHY SystemVerilog RTL files. |
| <*working_dir*>/<*variation_name*>_sim/ | <*variation_name*>_readme.txt | Readme text file. |

Table 2–6 lists the generated directory structure and key files created by the VHDL simulation flow with the MegaWizard Plug-in Manager.

**Table 2–6. Generated Directory Structure and Key Files—MegaWizard Plug-In Manager Simulation Flow (VHDL)**

| Directory | File Name | Description |
|---|---|---|
| <*working_dir*>/<*variation_name*>_sim/ | <*variation_name*>.vho | UniPHY VHDL top-level module. |
| <*working_dir*>/<*variation_name*>_sim/ | <*variation_name*>_*.vhd<br><*variation_name*>_*.vho | UniPHY simulation VHDL files. |
| <*working_dir*>/<*variation_name*>_sim/ | vhdl_files.txt | File list text file. |

## Example Design

Table 2–7 lists the generated directory structure and key files created for the example design with the MegaWizard Plug-in Manager

**Table 2–7. Generated Directory Structure and Key Files—MegaWizard Plug-In Manager Example Design (Part 1 of 2)**

| Directory | File Name‘ | Description |
|---|---|---|
| <*working_dir*>/<*variation_name*>_example_design_fileset/ | <*variation_name*>.qip | QIP which refers to UniPHY RTL in this fileset. This is distinct from ../<*variation_name*>.qip. This file is included automatically in the example project. |
| <*working_dir*>/<*variation_name*>_example_design_fileset/ | <*variation_name*>.v | UniPHY top-level wrapper. |
| <*working_dir*>/<*variation_name*>_example_design_fileset | <*variation_name*>_*.v | UniPHY Verilog RTL files. |
| <*working_dir*>/<*variation_name*>_example_design_fileset | <*variation_name*>_*.sv | UniPHY SystemVerilog RTL fies. |
| <*working_dir*>/<*variation_name*>_example_design_fileset/ | <*variation_name*>.sdc | Synopsys constraints file. |

**Table 2–7. Generated Directory Structure and Key Files—MegaWizard Plug-In Manager Example Design (Part 2 of 2)**

| Directory | File Name‘ | Description |
|---|---|---|
| <*working_dir*>/<*variation_name*>_example_design_fileset/ | <*variation_name*>.ppf | Pin Planner file. |
| <*working_dir*>/<*variation_name*>_example_design_fileset/ | <*variation_name*>_pin_assignments.tcl | Pin constraints script to be run after synthesis. |
| <*working_dir*>/<*variation_name*>_example_design_fileset/ | <*variation_name*>_*.tcl | Other Tcl scripts. |
| <*working_dir*>/<*variation_name*>_example_design_fileset/ | <*variation_name*>_readme.txt | Readme text file. |
| <*working_dir*>/<*variation_name*>_example_design_fileset/example_project/ | <*variation_name*>_example_top.qpf | Example design project file. |
| <*working_dir*>/<*variation_name*>_example_design_fileset/example_project/ | <*variation_name*>_example_top.qsf | Example design project settings file. |
| <*working_dir*>/<*variation_name*>_example_design_fileset/example_project/ | <*variation_name*>_example_top.v | Top-level wrapper including UniPHY, traffic generator, and memory model. |
| <*working_dir*>/<*variation_name*>_example_design_fileset/example_project/ | <*variation_name*>_*.v | Other example design Verilog RTL files. |
| <*working_dir*>/<*variation_name*>_example_design_fileset/example_project/ | <*variation_name*>_*.sv | Other example design SystemVerilog RTL files. |
| <*working_dir*>/<*variation_name*>_example_design_fileset/example_project/ | <*prefix*>_mem_model.sv | Generic memory model. |
| <*working_dir*>/<*variation_name*>_example_design_fileset/rtl_sim/ | <*variation_name*>_example_top_tb.v | Top-level test bench. |

**Note to Table 2–7:**

(1) <*prefix*> varies depending on protocol and type of memory model.

## SOPC Builder Flow

Table 2–8 lists the generated directory structure and key files created by the SOPC Builder flow.

**Table 2–8. Generated Directory Structure and Key Files—SOPC Builder Flow (Part 1 of 2)**

| Directory | File Name‘ | Description |
|---|---|---|
| <*working_dir*>/ | <*system_name*>.qip | QIP which refers to all generated files in the SOPC Builder project. |
| <*working_dir*>/ | <*system_name*>.v | SOPC Builder system top-level wrapper. |
| <*working_dir*>/ | <*core_name*>_<*stamp*>.v (1) | UniPHY top-level wrapper. |
| <*working_dir*>/ | <*core_name*>_<*stamp*>_*.v (1) | UniPHY Verilog RTL files. |
| <*working_dir*>/ | <*core_name*>_<*stamp*>_*.sv (1) | UniPHY SystemVerilog RTL files. |
| <*working_dir*>/ | <*core_name*>_<*stamp*>.sdc (1) | Synopsys constraints file. |

**Table 2–8. Generated Directory Structure and Key Files—SOPC Builder Flow (Part 2 of 2)**

| Directory | File Name‘ | Description |
|---|---|---|
| <*working_dir*>/ | <*core_name*>_<*stamp*>.ppf (1) | Pin Planner file. |
| <*working_dir*>/ | <*core_name*>_<*stamp*>_pin_assignments.tcl (1) | Pin constraints script to be run after synthesis. |
| <*working_dir*>/ | <*core_name*>_<*stamp*>_*.tcl (1) | Other Tcl scripts. |
| <*working_dir*>/ | <*core_name*>_<*stamp*>_readme.txt (1) | Readme text file. |
| <*working_dir*>/ | *Other IP core files*. | Other IP cores. |

**Note to Table 2–8:**

(1) <*stamp*> is a unique identifier determined by SOPC Builder at generation time.

## Qsys Flow

The tables in this section list the generated directory structure and key files of interest to users, resulting from the Qsys flow

### Synthesis

Table 2–9 lists the generated directory structure and key files created by the synthesis flow with Qsys.

**Table 2–9. Generated Directory Structure and Key Files—Qsys Synthesis Flow (Part 1 of 2)**

| Directory | File Name | Description |
|---|---|---|
| <*working_dir*>/<*system_name*>/synthesis/ | <*system_name*>.qip | QIP which refers to all generated files in the Qsys system synthesis fileset. |
| <*working_dir*>/<*system_name*>/synthesis/ | <*system_name*>.v | Qsys system top-level wrapper. |
| <*working_dir*>/<*system_name*>/synthesis/submodules/ | <*core_name*>_<*stamp*>.v (1) | UniPHY top-level wrapper. |
| <*working_dir*>/<*system_name*>/synthesis/submodules/ | <*core_name*>_<*stamp*>_*.v (1) | UniPHY Verilog RTL files. |
| <*working_dir*>/<*system_name*>/synthesis/submodules/ | <*core_name*>_<*stamp*>_*.sv (1) | UniPHY SystemVerilog RTL files. |
| <*working_dir*>/<*system_name*>/synthesis/submodules/ | <*core_name*>_<*stamp*>.sdc (1) | Synopsys constraints file. |
| <*working_dir*>/<*system_name*>/synthesis/submodules/ | <*core_name*>_<*stamp*>.ppf (1) | Pin Planner file. |
| <*working_dir*>/<*system_name*>/synthesis/submodules/ | <*core_name*>_<*stamp*>_pin_assignments.tcl (1) | Pin constraints script to be run after synthesis. |
| <*working_dir*>/<*system_name*>/synthesis/submodules/ | <*core_name*>_<*stamp*>_*.tcl (1) | Other Tcl scripts. |
| <*working_dir*>/<*system_name*>/synthesis/submodules/ | <*core_name*>_<*stamp*>_readme.txt (1) | Readme text file. |

**Table 2–9. Generated Directory Structure and Key Files—Qsys Synthesis Flow (Part 2 of 2)**

| Directory | File Name | Description |
|---|---|---|
| <*working_dir*>/<*system_name*>/ synthesis/submodules/ | *Other IP core files* | Other IP core files. |
| **Note to Table 2–9** (1) <*stamp*> is a unique identifier created by Qsys during generation. | | |

## Verilog Simulation

Table 2–10 lists the generated directory structure and key files created by the Verilog simulation flow with Qsys.

**Table 2–10. Generated Directory Structure and Key Files—Qsys Verilog Simulation**

| Directory | File Name | Description |
|---|---|---|
| <*working_dir*>/<*system_name*>/ sim_verilog/ | <*system_name*>.v | Qsys system top-level wrapper. |
| <*working_dir*>/<*system_name*>/ sim_verilog/submodules/ | <*core_name*>_<*stamp*>.v | UniPHY top-level wrapper. |
| <*working_dir*>/<*system_name*>/ sim_verilog/submodules/ | <*core_name*>_<*stamp*>_*.v | UniPHY Verilog RTL files. |
| <*working_dir*>/<*system_name*>/ sim_verilog/submodules/ | <*core_name*>_<*stamp*>_*.sv | UniPHY SystemVerilog RTL files. |
| <*working_dir*>/<*system_name*>/ sim_verilog/submodules/ | <*core_name*>_<*stamp*>_readme.txt (1) | Readme text file. |
| <*working_dir*>/<*system_name*>/ sim_verilog/submodules/ | *Other IP core files* | Other IP core files. |
| **Note for Table 2–10:** (1) <*stamp*> is a unique identifier created by Qsys during generation. | | |

## VHDL Simulation

Table 2–11 lists the generated directory structure and key files created by the VHDL simulation flow with Qsys.

**Table 2–11. Generated Directory Structure and Key Files—Qsys VHDL Simulation**

| Directory | File Name | Description |
|---|---|---|
| <*working_dir*>/<*system_name*>/ sim_vhdl/ | <*system_name*>.vhd | Qsys system top-level wrapper. |
| <*working_dir*>/<*system_name*>/ sim_vhdl/submodules/ | <*core_name*>_<*stamp*>.vho (1) | UniPHY VHDL top-level module. |
| <*working_dir*>/<*system_name*>/ sim_vhdl/submodules/ | <*core_name*>_<*stamp*>_*.vhd<br><*core_name*>_<*stamp*>_*.vho (1) | UniPHY VHDL simulation files. |
| <*working_dir*>/<*system_name*>/ sim_vhdl/submodules/ | vhdl_files.txt | File list text file. |
| **Note to Table 2–11:** (1) <*stamp*> is a unique identifier created by Qsys during generation. | | |

# 3. Parameter Settings

This chapter describes the RLDRAM II Controller with UniPHY IP core parameters that you can set in the parameter editor.

## General Settings

The **General Settings** tab allows you to configure the following parameter settings.

### Clocks

Table 3–1 describes the clock settings.

**Table 3–1. Clock Settings**

| Parameter | Description |
|---|---|
| Memory clock frequency | The frequency of the clock that drives the memory device. |
| PLL reference clock frequency | The frequency of the clock that feeds the PLL. |
| Full or half rate on Avalon-MM interface | Defines the width of the data bus on the Avalon-MM interface. A setting of **Full** results in a width twice the memory data width. A setting of **Half** results in a width of four times the memory data width. |
| Additional address/command clock phase | Increases or decreases the phase shift of the address/command clock. The base phase shift center-aligns the address/command clock at the memory device. In some circumstances, you can improve timing by increasing or decreasing the phase shift. |

### Advanced PHY Settings

Table 3–2 describes the advanced PHY settings.

**Table 3–2. Advanced PHY Settings**

| Parameter | Description |
|---|---|
| Generate power-of-2 bus widths | Rounds down the Avalon-MM side data bus to the nearest power of 2. |
| Maximum Avalon-MM burst length | Specifies the maximum burst length on the Avalon-MM bus. |
| I/O standard | Specifies the I/O standard voltage. |
| Master for PLL/DLL sharing | Causes UniPHY to instantiate its own PLL and DLL. All of the PLL clocks and DLL delay values are exported for use by other identical UniPHY cores that have this option turned on. |

**Table 3–2. Advanced PHY Settings**

| Parameter | Description |
|---|---|
| Master for OCT control block | Causes UniPHY to instantiate the required OCT control block. When this parameter is turned off, you must instantiate this block and connect the termination control bus signals to the PHY, or share an OCT control block from another UniPHY instantiation that is in master mode. |
| Hardcopy compatibility mode | Causes the generated UniPHY memory interface to have all required HardCopy compatibility options enabled. For example, PLLs and DLLs will have their reconfiguration ports exposed. |
| **Example Testbench Simulation options** | |
| Skip memory initialization | Causes the example testbench to skip the memory initialization sequence. This setting does not change the generated RTL, but can speed up simulation. |

## Topology

Table 3–3 describes the topology settings.

**Table 3–3. Topology Settings**

| Parameter | Description |
|---|---|
| Device width | Specifies the number of devices used for width expansion. |
| Device depth | Specifies the number of devices (ranks) used for depth expansion. |

## Controller Settings

Table 3–4 describes the controller settings.

**Table 3–4. Controller Settings**

| Parameter | Description |
|---|---|
| Controller latency | Specifies the number of clock cycles required for a request to pass through an idling controller. |
| Enable user refresh | Enables user-controlled refresh. Refresh signals will have priority over read/write requests. |
| Enable error detection parity | Enables per-byte parity protection. |

# Memory Parameters

The **Memory Parameters** tab allows you to configure memory device parameters. You can enter parameters manually from the manufacturer's device data sheet, or you can populate the fields automatically by selecting the required device from the list of presets.

Table 3–5 describes the memory parameters.

**Table 3–5. Memory Parameters**

| Parameter | Description |
|---|---|
| Address width | The width of the address bus on the memory device. |
| Data width | The width of the data bus on the memory device. |
| Bank-address width | The width of the bank-address bus on the memory device. |
| Data-mask width | The width of the data-mask on the memory device, |
| QK width | The width of the QK (read strobe) bus on the memory device. |
| DK width | The width of the DK (write strobe) bus on the memory device. |
| Burst length | The burst length supported by the memory device. |
| Memory mode register configuration | Configuration bits that set the memory mode. |

# Memory Timing

The **Memory Timing** tab allows you to configure memory device timing parameters. You can enter timing parameters manually from the manufacturer's device data sheet, or you can populate the fields automatically by selecting the required device from the list of presets.

Table 3–6 describes the memory timing parameters.

**Table 3–6. Memory Timing Parameters (Part 1 of 2)**

| Parameters | Description |
|---|---|
| Maximum memory clock frequency | The maximum frequency at which the memory device can run. |
| Refresh interval | The refresh interval. |
| tCKH (%) | The input clock (K/K#) high expressed as a percentage of the full clock period. |
| tQKH (%) | The read clock (QK/QK#) high expressed as a percentage of tCKH. |
| tAS | Address and control setup to K clock rise. |
| tAH | Address and control hold after K clock rise. |
| tDS | Data setup to clock (K/K#) rise. |
| tDH | Data hold after clock (K/K#) rise. |
| tQKQ_max | QK clock edge to DQ data edge (in same group). |
| tQKQ_min | QK clock edge to DQ data edge (in same group). |

**Table 3–6. Memory Timing Parameters (Part 2 of 2)**

| Parameters | Description |
|---|---|
| tCKDK_max | Clock to input data clock (max). |
| tCKDK_min | Clock to input data clock (min). |

# Board Settings

The **Board Settings** tab allows you to enter values derived from board simulation and from the manufacturer's memory device data sheet.

## Setup and Hold Derating

The slew rate of output signals affects the setup and hold times of the memory device. Enter input slew rate derating parameters from the memory device data sheet to obtain derated setup and hold times.

**Table 3–7. Setup and Hold Derating Parameters (Part 1 of 2)**

| Parameters | Description |
|---|---|
| tAS Vref to CK/CK# Crossing | For a given address/command and CK/CK# slew rate, the memory device data sheet provides a corresponding "tAS Vref to CK/CK# Crossing" value that can be used to determine the derated address/command setup time. |
| tAS VIH MIN to CK/CK# Crossing | For a given address/command and CK/CK# slew rate, the memory device data sheet provides a corresponding "tAS VIH MIN to CK/CK# Crossing" value that can be used to determine the derated address/command setup time. |
| tAH CK/CK# Crossing to Vref | For a given address/command and CK/CK# slew rate, the memory device data sheet provides a corresponding "tAH CK/CK# Crossing to Vref" value that can be used to determine the derated address/command hold time. |
| tAH CK/CK# Crossing to VIH MIN | For a given address/command and CK/CK# slew rate, the memory device data sheet provides a corresponding "tAH CK/CK# Crossing to VIH MIN" value that can be used to determine the derated address/command hold time. |
| tDS Vref to CK/CK# Crossing | For a given data and DK/DK# slew rate, the memory device data sheet provides a corresponding "tDS Vref to CK/CK# Crossing" value that can be used to determine the derated data setup time. |
| tDS VIH MIN to CK/CK# Crossing | For a given data and DK/DK# slew rate, the memory device data sheet provides a corresponding "tDS VIH MIN to CK/CK# Crossing" value that can be used to determine the derated data setup time. |
| tDH CK/CK# Crossing to Vref | For a given data and DK/DK# slew rate, the memory device data sheet provides a corresponding "tDH CK/CK# Crossing to Vref" value that can be used to determine the derated data hold time. |

**Table 3–7. Setup and Hold Derating Parameters (Part 2 of 2)**

| Parameters | Description |
|---|---|
| tDH CK/CK# Crossing to VIH MIN | For a given data and DK/DK# slew rate, the memory device data sheet provides a corresponding "tDH CK/CK# Crossing to VIH MIN" value that can be used to determine the derated data hold time. |
| Derated tAS | The derated address/command setup time is calculated automatically from the "tAS", the "tAS Vref to CK/CK# Crossing", and the "tAS VIH MIN to CK/CK# Crossing" parameters. |
| Derated tAH | The derated address/command hold time is calculated automatically from the "tAH", the "tAH CK/CK# Crossing to Vref", and the "tAH CK/CK# Crossing to VIH MIN" parameters. |
| Derated tDS | The derated data setup time is calculated automatically from the "tDS", the "tDS Vref to CK/CK# Crossing", and the "tDS VIH MIN to CK/CK# Crossing" parameters. |
| Derated tDH | The derated data hold time is calculated automatically from the "tDH", the "tDH CK/CK# Crossing to Vref", and the "tDH CK/CK# Crossing to VIH MIN" parameters. |

## Intersymbol Interference

Intersymbol interference (ISI), occurs when a signal is distorted due to the interference of one symbol with subsequent symbols. Typically, ISI is greater at device depths greater than one because there are multiple stubs causing reflections. The advanced I/O timing analysis capabilities of the Quartus II software already includes ISI effects for device depth of one.

Table 3–8 describes the intersymbol interference settings.

**Table 3–8. Intersymbol Interference Settings**

| Parameter | Description |
|---|---|
| Address/command eye reduction (setup) | The reduction in the eye diagram on the setup side (or left side of the eye) due to ISI on the address/command signals compared to a case where there is no ISI. |
| Address/command eye reduction (hold) | The reduction in the eye diagram on the hold side (or right side of the eye) due to ISI on the address/command signals compared to a case where there is no ISI. |
| DQ eye reduction | The total reduction in the eye diagram due to ISI on DQ signals compared to a case where there is no ISI. (It is assumed that the ISI reduces the eye width symmetrically on the left and right sides of the eye.) |
| Delta K arrival time | The increase in variation on the range of arrival times of DQS compared to a case when there is no ISI. (It is assumed that the ISI causes DQS to further vary symmetrically to the left and right.) |

## Board Skews

Skews between PCB traces can reduce timing margins.

Table 3–9 describes the board skew settings.

**Table 3–9. Board Skews Settings**

| Parameter | Description |
|---|---|
| Minimum delay difference between CK and DK | The minimum delay difference between the CK signal and any DK signal when arriving at the memory device(s). The value is equal to the minimum delay of the CK signal minus the maximum delay of the DK signal. The value can be positive or negative. |
| Maximum delay difference between CK and D | The maximum delay difference between the CK signal and any DK signal when arriving at the memory device(s). The value is equal to the maximum delay of the CK signal minus the minimum delay of the DK signal. The value can be positive or negative. |
| Maximum delay difference between devices | The maximun delay difference of data signals between devices. For example, in a two-device configuration there is greater propagation delay for data signals going to and returning from the furthest device relative to the nearest device. |
| Maximum skew within QK group | The maximum skew between the DQ signals referenced by a common QK signal. |
| Maximum skew between QK groups | The maximum skew between QK signals of different data groups. |
| Maximun skew within address/command bus | The maximum skew between the address/command signals. |
| Average delay difference between address/command and CK | A value equal to the average of the longest and smallest address/command signal delay values, minus the delay of the CK signal. The value can be positive or negative. |
| Average delay difference between write data signals and DK | A value equal to the average of the longest and smallest write data signal delay values, minus the delay of the DK signal. Write data signals include the DQ and DM signals. The value can be positive or negative. |
| Average delay difference between read data signals and QK | A value equal to the average of the longest and smallest read data signal delay values, minus the delay of the QK signal. The value can be positive or negative. |

The Quartus II software generates a Synopsis Design Constraint (**.sdc**) script, <*variation_name*>**.sdc**, and a pin assignment script, <*variation_name*>**_pin_assignments.tcl**. Both the **.sdc** and the <*variation name*>**_pin_assignments.tcl** support multiple instances. These scripts iterate through all instances of the core and apply the same constraints to all of them.

## Add Pin and DQ Group Assignments

The pin assignment script, <*variation_name*>**_pin_assignments.tcl**, sets up the I/O standards and the input/output termination for the RLDRAM II controller with UniPHY. This script also helps to relate the `DQ` and `QK` pin groups together for the Fitter to place them correctly.

The pin assignment script does not create a source clock for the design. You must create a clock for the design and provide pin assignments for the signals of both the example driver and testbench that the MegaCore variation generates.

Run the <*variation_name*>**_pin_assignments.tcl** to add the input and output termination, I/O standards, and DQ group assignments to the example design. To run the pin assignment script, follow these steps:

1. On Processing menu, point to **Start**, and click **Start Analysis and Synthesis**.
2. On the Tools menu click **Tcl Scripts**.
3. Specify the <*variation_name*>**_pin_assignments.tcl** file and click **Run**.

If the PLL input reference clock pin is not the same I/O standard as the memory interface I/Os, the design might not fit into the device because incompatible I/O standards cannot be placed in the same I/O bank.

## Board Settings Tab

The **Board Settings** tab allows you to enter board-related data. In the **Setup and Hold Derating** section, you enter derating parameters from the device data sheet, which the system uses to calculate derated setup and hold values. In the **Intersymbol Interference** and **Board Skews** sections, you enter information derived during your PCB development process of prelayout (line) simulation and finally postlayout (board) simulation.

Timing analysis does not consider bus turnaround; consequently, the controller dead times are based on assumptions about the user board trace lengths. For timing analysis to be accurate, board trace delays must not exceed 0.6 ns from FPGA to memory and from memory to FPGA.

For more information about how to include your board simulation results in the Quartus II software and how to assign pins using pin planners, refer to *Volume 6: Design Flow Tutorials* of the *External Memory Interface Handbook*.

# Compile the Design

To compile the design, on the Processing menu, click **Start Compilation**.

After you have compiled the top-level file, you can perform RTL simulation or program your targeted Altera device to verify the top-level file in hardware.

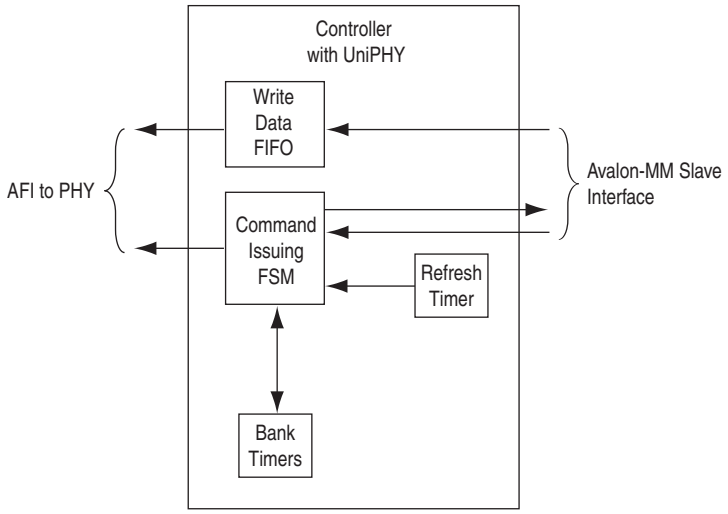
For more information about simulating, refer to the *Simulation* section in volume 4 of the *External Memory Interface Handbook*.

The controller translates memory requests from the Avalon Memory-Mapped (Avalon-MM) interface to AFI, while satisfying timing requirements imposed by the memory configurations.

## Block Description

This topic describes the blocks in the IP. Figure 5–1 shows a block diagram of the RLDRAM II controller architecture.

**Figure 5–1. RLDRAM II Controller Architecture Block Diagram**




### Avalon-MM Slave Interface

This Avalon-MM slave interface accepts read and write requests. A simple state machine represents the state of the command and address registers, which stores the command and address when a request arrives.

The Avalon-MM slave interface decomposes the Avalon-MM address to the memory bank, column, and row addresses. The IP automatically maps the bank address to the LSB of the Avalon address vector.

The Avalon-MM slave interface includes a burst adaptor, which has the following two parts:

- The first part is a read and write request combiner that groups requests to sequential addresses into the native memory burst. Given that the second request arrives within the read and write latency window of the first request, the controller can combine and satisfy both requests with a single memory transaction.

- The second part is the burst divider in the front end of the Avalon-MM interface, which breaks long Avalon bursts into individual requests of sequential addresses, which then pass to the controller state machine.

### Write Data FIFO Buffer

The write data FIFO buffer accepts write data from the Avalon-MM interface. The AFI controls the subsequent consumption of the FIFO buffer write data.

### Command Issuing FSM

The command issuing finite-state machine (FSM) has three states. The controller is in the `INIT` state when the PHY initializes the memory. Upon receiving the `afi_cal_success` signal, the state transitions to `INIT_COMPLETE`. If the calibration fails, `afi_cal_fail` is asserted and the state transitions to `INIT_FAIL`. The PHY receives commands only in the `INIT_COMPLETE` state.

When a refresh request arrives at the state machine at the same time as a read or write request, the refresh request takes precedence. The read or write request waits until there are no more refresh requests, and is issued immediately if timing requirements are met.

### Refresh Timer

With automatic refresh, the refresh timer periodically issues refresh requests to the command issuing FSM. The refresh interval can be set at generation.

### Timer Module

The timer module contains one DQ timer and eight bank timers (one per bank). The DQ timer tracks how often read and write requests can be issued, to avoid bus contention. The bank timers track the cycle time ($t_{RC}$).

The 8-bit wide output bus of the bank timer indicates to the command issuing FSM whether each bank can be issued a read, write, or refresh command.

### AFI

For information on the AFI, refer to "Functional Description—UniPHY" on page 6–1.

## User-Controlled Features

The following features are available on the **General Settings** tab of the parameter editor. These features are disabled by default.

### Error Detection Parity

The error detection parity protection feature creates a simple parity encoder block which processes all read and write data. The error detection feature asserts an error signal if it detects any corrupted data during the read process. For every 8 bits of write data, a parity bit is generated and concatenated to the data before it is written to the memory. During the subsequent read operation, the parity bit is checked against the data bits to ensure data integrity.

Enabling the error detection parity protection feature reduces the local data width by one. For example, a nine-bit memory interface will present eight bits of data to the controller interface.

You can enable error detection parity protection in the **Controller Settings** section of the **General Settings** tab of the parameter editor.

### User-Controlled Refresh

The user-controlled refresh feature allows you to take control of the refresh process that the controller normally performs automatically. You can control when refresh requests occur, and, if there are multiple memory devices, you control which bank receives the refresh signal. When you enable this feature, you disable auto-refresh, and assume responsibility for maintaining the necessary average periodic refresh rate.

You can enable user-controlled refresh in the **Controller Settings** section of the **General Settings** tab of the parameter editor.

## Avalon-MM and Memory Data Width

Table 5–1 shows the data width ratio between the memory interface and the Avalon-MM interface. The half-rate controller does not support burst-of-2 devices because it under-uses the available memory bandwidth.

**Table 5–1. Data Width Ratio**

| Memory Burst Length | Half-Rate Designs | Full-Rate Designs |
|---|---|---|
| 2-word | No Support | 2:1 |
| 4-word | 4:1 | |
| 8-word | | |

## Signal Description

This topic discusses the signals for each interface.

For information on the AFI signals, refer to "UniPHY Signals" on page 6–10.

## Avalon-MM Slave Interface

Table 5–2 shows the list of signals of the controller's Avalon-MM slave interface.

**Table 5–2. Avalon-MM Slave Signals**

| Signal | Width | Direction | Avalon-MM Signal Type | Description |
|---|---|---|---|---|
| avl_size | 1 to 11 | In | burstcount | — |
| avl_ready | 1 | Out | waitrequest_n | — |
| avl_read_req | 1 | In | read | — |
| avl_write_req | 1 | In | write | — |
| avl_addr | ≤25 | In | address | — |
| avl_rdata_valid | 1 | Out | readdatavalid | — |
| avl_rdata | 18, 36, 72, 144 | Out | readdata | — |
| avl_wdata | 18, 36, 72, 144 | In | writedata | — |

The data width of the Avalon-MM interface is restricted to powers of two when using SOPC Builder or Qsys. Non-power-of-two data widths are supported when using the MegaWizard Plug-In Manager.

This chapter describes the PHY part of the RLDRAM II controller with UniPHY.

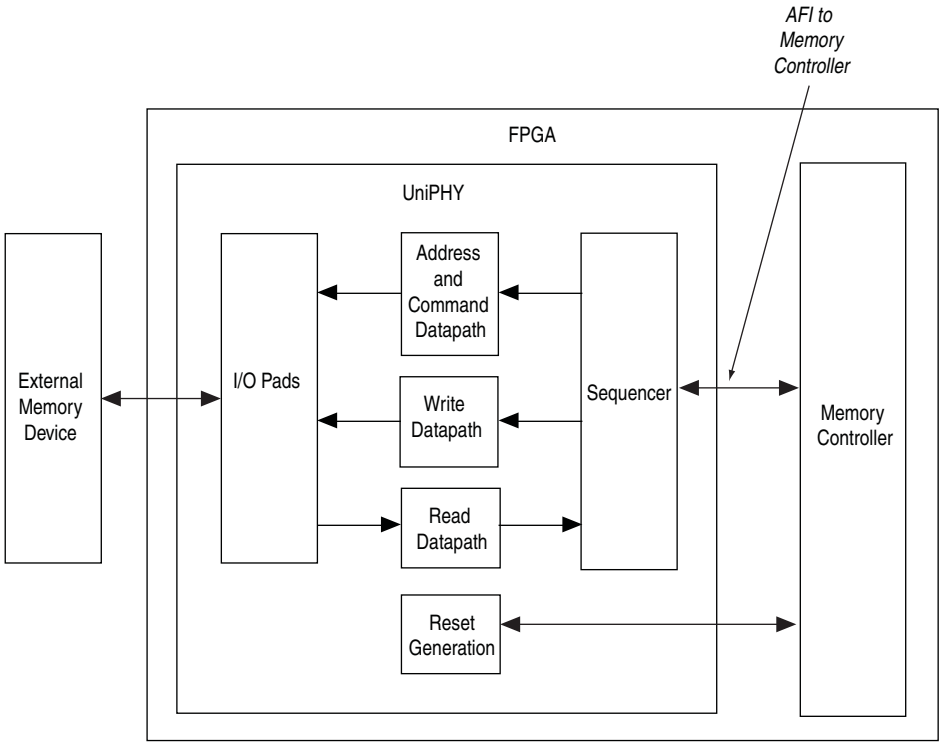
## Block Description

The PHY comprises the following major functional units:

- I/O Pads
- Reset and Clock Generation
- Address and Command Datapath
- Write Datapath
- Read Datapath
- Sequencer

Figure 6–1 shows the PHY block diagram.

**Figure 6–1. PHY Block Diagram**




## I/O Pads

The I/O pads contain all the I/O instantiations. The bulk of the UniPHY I/O circuitry is encapsulated in the ALTDQ_DQS megafunction (ALTDQ_DQS2 for Stratix V series devices).

## Reset and Clock Generation

The clocking operation in the PHY can be classified into two domains: the PHY-memory domain and the PHY-AFI domain. The PHY-memory domain interfaces with the external memory device and is always at full-rate. The PHY-AFI domain interfaces with the memory controller and can be either a full-rate or half-rate clock based on the choice of the controller. Table 6–1 lists the clocks required for half-rate designs.

**Table 6–1. Clocks—Half-Rate Designs**

| Clock | Source | Clock Rate | Phase | Clock Network Type | Description |
|---|---|---|---|---|---|
| pll_afi_clk | PLL: C0 | Half | 0° | Unconstrained | Clock for AFI logic. |
| pll_mem_clk | PLL: C1 | Full | 0°(1)<br>-45°(2) | Dual-regional(4) | Output clock to memory. |
| pll_write_clk | PLL: C2 | Full | -90°(1)<br>-135°(2)<br>45° (3) | Dual-regional(4) | Clock for write data out to memory (data is center aligned with the delayed pll_write_clk). |
| pll_addr_cmd_clk | PLL: C3 | Half | Set in wizard (default 270°) | Dual-regional | Clock for the address and command out to memory (address and command is center aligned with memory clock). |
| DQS | Memory | Full | 90° | Local | A continuous running clock from the memory device for capturing read data. |

**Notes for Table 6–1:**

(1) For memory frequencies >240 MHz.
(2) For memory frequencies <=240 MHz.
(3) For memory frequencies >=240 MHz, for Stratix V devices only.
(4) For parameterizations with interface width >36, pll_mem_clk and pll_write_clk are assigned to use the global network.

Table 6–2 lists the clocks required for full-rate designs.

**Table 6–2. Clocks—Full-Rate Designs (Part 1 of 2)**

| Clock | Source | Clock Rate | Phase | Clock Network Type | Description |
|---|---|---|---|---|---|
| pll_afi_clk | PLL: C0 | Full | 0° | Unconstrained | Clock for AFI logic. |
| pll_mem_clk | PLL: C1 | Full | 90°(1)<br>0°(2) | Dual-regional(3) | Output clock to memory. |
| pll_write_clk | PLL: C2 | Full | 180°(1)<br>-90°(2) | Dual-regional(3) | Clock for write data out to memory (data is center aligned with the delayed pll_write_clk). |
| pll_addr_cmd_clk | PLL: C3 | Full | Set in wizard (default 225°) | Dual-regional | Clocks address/command out to memory. 180° gives adress and command center aligned with memory clock; 225° produces best overall timing results. |

**Table 6–2. Clocks—Full-Rate Designs (Part 2 of 2)**

| Clock | Source | Clock Rate | Phase | Clock Network Type | Description |
|---|---|---|---|---|---|
| DQS | Memory | Full | 90° | Local | A continuous running clock from the memory device for capturing read data. |

**Notes for Table 6–2:**

(1) For memory frequencies >240 MHz.

(2) For memory frequencies <=240 MHz.

(3) For parameterizations with interface width >36, pll_mem_clk and pll_write_clk are assigned to use the global network.

The UniPHY uses an active-low, asynchronous assert and synchronous de-assert reset scheme. The global reset signal resets the PLL in the PHY and the rest of the system waits in reset until after the PLL becomes locked. The number of synchronization pipeline stages is 4.

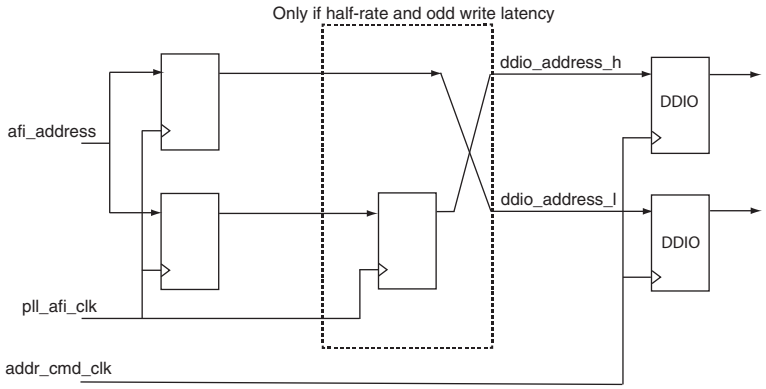
## Address and Command Datapath

The memory controller controls the read and write addresses and commands to meet the memory specifications. The PHY simply passes the address and command received from the memory controller to the memory device. The PHY circuitry is the same for both address and command.

The address and command datapath outputs connect to the inputs of the address and command I/Os. An ALTDDIO_OUT megafunction converts the addresses from SDR to DDR. An ALTDDIO_OUT megafunction with an ALTIOBUF megafunction delivers a pair of address and command clock to the memory.

Figure 6–2 illustrates the address and command datapath. The controller requires the registry-and- address-swapping circuitry inside the dotted box only when it is operating in half-rate mode with odd write latency. In full-rate mode, `ddio_address_h` and `ddio_addesss_l` are the same.

**Figure 6–2. Address and Command Datapath**

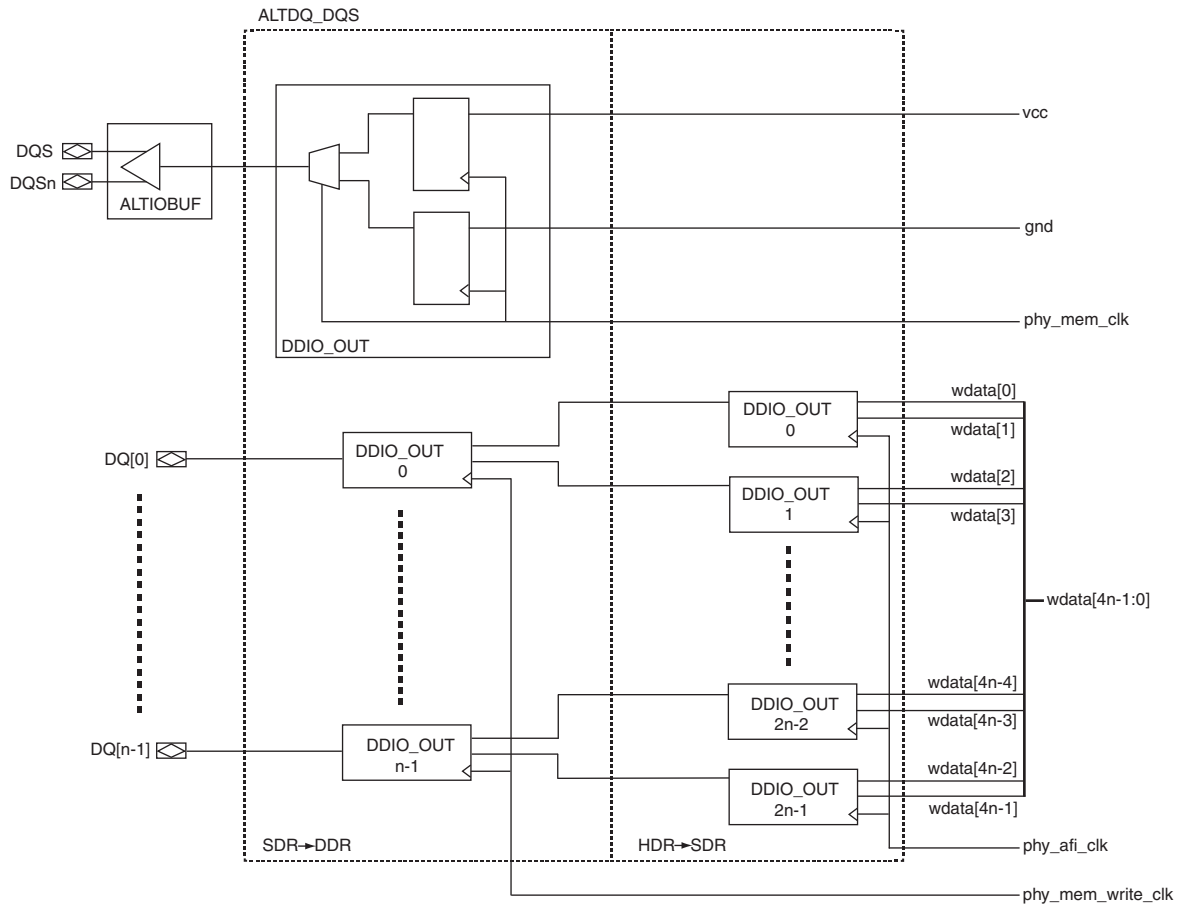


## Write Datapath

The write datapath passes write data from the memory controller to the I/O. The DQ pins are bidirectional and shared between read and write. The write data valid signal from the memory controller generates the output enable signal to control the output buffer. It also generates the dynamic termination control signal, which selects between series (output mode) and parallel (input mode) termination. An ALT_OCT megafunction (instantiated in the top-level file) configures the termination values.

Figure 6–3 illustrates the write datapath. The full-rate PLL output clock phy_mem_clk is sent to a DDIO_OUT cell. The output of ALTDQ_DQS feeds an ALTIOBUF buffer which creates a pair of pseudodifferential clocks that connects to the memory. In full-rate mode, only the SDR-DDR portion of the ALTDQ_DQS logic is used; in half-rate mode, the HDR-SDR circuitry is also required. The <*variation_name*>**_pin_assignments.tcl** script automatically specifies the logic option that associates all DQ pins to the DQS pin. The Fitter treats the pins as a DQS/DQ pin group.

**Figure 6–3. Write Datapath**



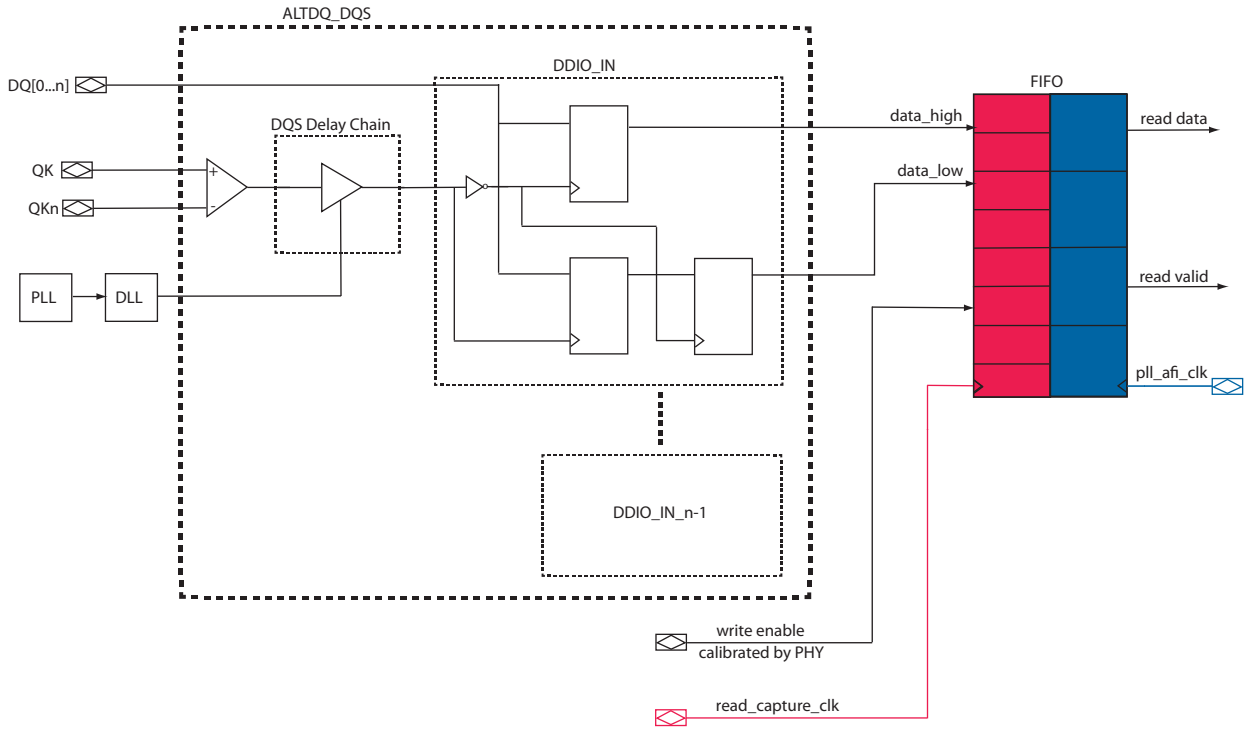
## Read Datapath

The read data is captured in the input mode ALTDQ_DQS in the I/O. The captured data passes to the read datapath. The read datapath synchronizes read data from the read capture clock domain to the AFI clock domain and converts data from SDR to HDR (half-rate designs only).

In half-rate designs, the write side of the FIFO buffer should be double the size of the read side of the FIFO buffer. The read side only reads one entry after the write side has written into two entries, which effectively converts data from SDR to HDR. In full-rate designs, the size of the FIFO buffer is the same for both write and read as both sides operate at the same rate. For half-rate designs, the FIFO operates at half-rate on both read and write sides, and contains 4 half-rate entries; for full-rate designs, the FIFO operates at full-rate on both read and write sides, and contains 8 full-rate entries.

Figure 6–4 illustrates the read datapath. The DQS and DQSn clocks and the read data (DQ) returned from memory are edge-aligned; the DQS and DQSn delay chains shift the clocks to achieve center alignment.

**Figure 6–4. Read Datapath**



## Sequencer

The sequencer is a state machine that processes the calibration algorithm. The sequencer assumes control of the interface at reset (whether at initial startup or when the IP is reset) and maintains control throughout the calibration process, relinquishing control to the memory controller only after successful calibration. Table 6–3 shows the major states in the sequencer.

**Table 6–3. Sequencer States (Part 1 of 2)**

| State | Description |
|---|---|
| RESET | Remain in this state until reset is released. |
| LOAD_INIT | Load any initialization values for simulation purposes. |
| STABLE | Wait until the memory device is stable. |
| WRITE_ZERO | Issue write command to address 0. |
| WAIT_WRITE_ZERO | Write all 0s to address 0. |
| WRITE_ONE | Issue write command to address 1. |
| WAIT_WRITE_ONE | Write all 1s to address 1. |
| **Valid Calibration States** | |
| V_READ_ZERO | Issue read command to address 0 (expected data is all 0s). |
| V_READ_NOP | This state represents the minimum number of cycles required between 2 back-to-back read commands. The number of NOP states depends on the burst length. |
| V_READ_ONE | Issue read command to address 1 (expected data is all 1s). |
| V_WAIT_READ | Wait for read valid signal. |
| V_COMPARE_READ_ZERO_READ_ONE | Parameterizable number of cycles to wait before making the read data comparisons. |
| V_CHECK_READ_FAIL | When a read fails, the write pointer (in the AFI clock domain) of the valid FIFO buffer is incremented. The read pointer of the valid FIFO buffer is in the DQS clock domain. The gap between the read and write pointers is effectively the latency between the time when the PHY receives the read command and the time valid data is returned to the PHY. |
| V_ADD_FULL_RATE | Advance the read valid FIFO buffer write pointer by an extra full rate cycle. |
| V_ADD_HALF_RATE | Advance the read valid FIFO buffer write pointer by an extra half rate cycle. In full-rate designs, equivalent to V_ADD_FULL_RATE. |
| V_READ_FIFO_RESET | Reset the read and write pointers of the read data synchronization FIFO buffer. |
| V_CALIB_DONE | Valid calibration is successful. |
| **Latency Calibration States** | |
| L_READ_ONE | Issue read command to address 1 (expected data is all 1s). |
| L_WAIT_READ | Wait for read valid signal from read datapath. Initial read latency is set to a predefined maximum value. |
| L_COMPARE_READ_ONE | Check returned read data against expected data. If data is correct, go to L_REDUCE_LATENCY; otherwise go to L_ADD_MARGIN. |
| L_REDUCE_LATENCY | Reduce the latency counter by 1. |
| L_READ_FLUSH | Read from address 0 (expected data is all 0s), to flush the contents of the read data resynchronization FIFO buffer. |
| L_WAIT_READ_FLUSH | Wait until the whole FIFO buffer is flushed, then go back to L_READ and try again. |

**Table 6–3. Sequencer States (Part 2 of 2)**

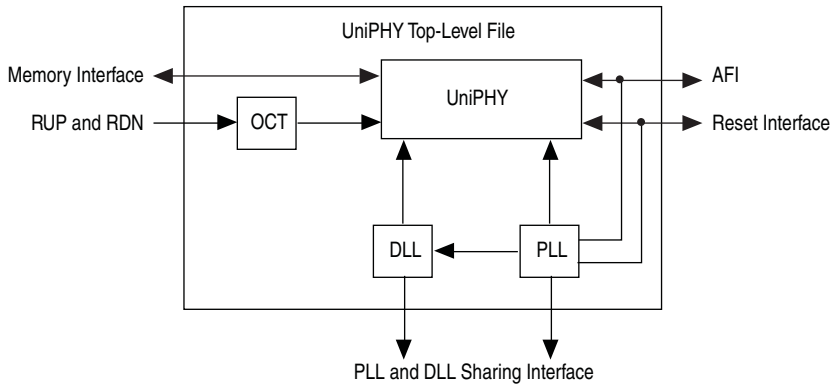
| State | Description |
|---|---|
| L_ADD_MARGIN | Increment latency counter by 3 (1 cycle to get the correct data, 2 more cycles of margin for run time variations). If latency counter value is smaller than predefined ideal condition minimum, then go to CALIB_FAIL. |
| CALIB_DONE | Calibration is successful. |
| CALIB_FAIL | Calibration is not successful. |

# Interfaces

Figure 6–5 shows the major blocks of the UniPHY and how it interfaces with the external memory device and the controller.

Instantiating the DLL and PLL on the same level as the UniPHY eases DLL and PLL sharing.

**Figure 6–5. UniPHY Interfaces with the Controller and the External Memory**




The following interfaces are on the UniPHY top-level file:

- AFI
- Memory interface
- DLL and PLL sharing interface
- OCT interface

## The Memory Interface

For more information on the memory interface, refer to "UniPHY Signals" on page 6–10.

## The DLL and PLL Sharing Interface

You can generate the UniPHY memory interface as a master or as a slave, depending on the setting of the **Master for PLL/DLL sharing** option on the **General Settings** tab of the parameter editor. The top level of a generated UniPHY variant contains both a PLL and a DLL; the PLL produces a variety of required clock signals derived from the reference clock, and the DLL produces a delay codeword. The top-level defines the PLL and DLL output signals as outputs for the master and as inputs for the slave. When you instantiate master and slave variants into your HDL code, you must connect the PLL outputs from the master to the clock inputs of the slaves.

☞ The master **.qip** file must appear before the slave **.qip** file in the Quartus II Settings File (**.qsf**).

The UniPHY memory interface requires one PLL and one DLL to produce the clocks and delay codeword. The PLL and DLL can be shared using a master and slave scenario. The top-level file defines the PLL and DLL output signals as inputs and outputs and an additional parameter `PLL_DLL_MASTER` is also defined. If `PLL_DLL_MASTER` is 1, the RTL instantiates the PLL and DLL, which drives the clock and DLL codeword inputs and outputs. If the parameter is 0, the wires previously connected to the output of the PLL and DLL are assigned to the clock and DLL codeword input and outputs. Inputs and outputs are specified based on the setting of the **PLL/DLL sharing** option.

☞ If you generate a slave IP core, you must modify the timing scripts to allow the timing analysis to correctly resolve clock names and analyze the IP core. Otherwise the software issues critical warnings and an incorrect timing report.

To modify the timing script, follow these steps:

1. In a text editor, open the <*IP core name*>**/constraints directory/**<*IP core name*>**_timing.tcl** file.
2. Search for the following 2 lines:
   - `set master_corename "_MASTER_CORE_"`
   - `set master_instname "_MASTER_INST_"`
3. Replace `_MASTER_CORE_` with the core name and `_MASTER_INST_` with instance name of the UniPHY master to which the slave is connected.

   ☞ The instance name is the full path to the instance and is in the <*IP core name*>**_all_pins.txt** file that is automatically generated after the <*IP core name*>**_pin_assignments.tcl** script runs.

4. If the slave component is connected to a user-defined PLL rather than a UniPHY master, you must manually enter all clock names.
   - Remove the `master_corename` and `master_instname` variables with the checks performed in the eight lines following them.
   - You can use all clock name assignments as templates. For example set `local_pll_afi_clk "mycomponent|mypll|my_afi_clk"`.


You must be extremely careful when connecting clock signals to the slave. Connecting to clocks with frequency or phase different than what the core expects may result in hardware failures.

## The OCT Sharing Interface

By default, the UniPHY IP generates the required OCT control block in the top-level RTL file for the PHY. If you want, you can instantiate this block elsewhere in your code and feed the required termination control signals into the IP core by turning off **Master for OCT Control Block** on the **PHY Settings** tab. If you turn off **Master for OCT Control Block**, you must instantiate the OCT control block, or use another UniPHY instance as a master, and ensure that the parallel and series termination control bus signals are connected to the PHY.

You must create termination Control Block assignments for all calibrated input-only pins, to designate which OCT control block to use for those pins. If the UniPHY IP is in OCT Control Block master mode, these assignments are included in the *<variation_name>*_**pin_assignments.tcl** file which must be run after analysis and synthesis. If the UniPHY IP is not using OCT Control Block master mode the user must manually create the required assignments to connect the input-only pins to the relevant OCT control block. For RLDRAM this is just the input clock, all output and bidirectional pins are hard coded between the pin's I/O buffer and the series and parallel termination control signals.

Figure 6–6 and Figure 6–7, respectively, show the PHY architecture with and without Master for OCT Control Block.

**Figure 6–6. PHY Architecture with Master for OCT Control Block**

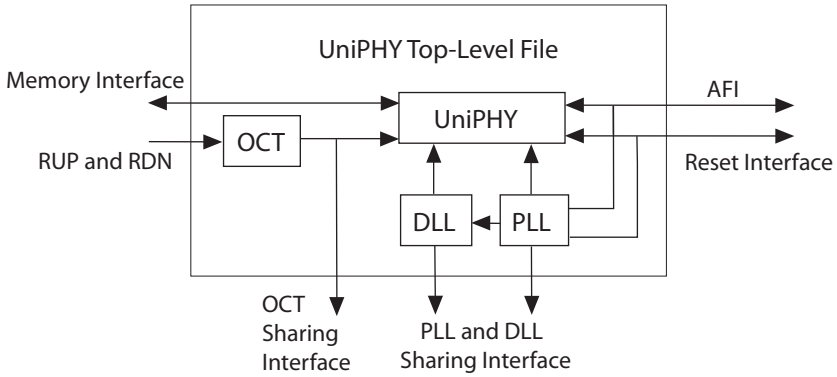
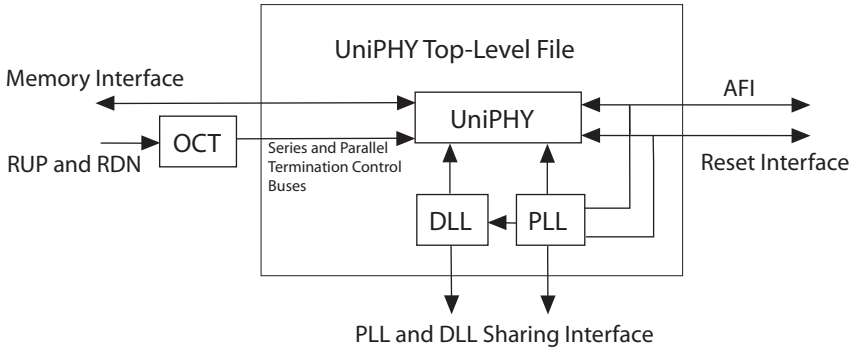


**Figure 6–7. PHY Architecture without Master for OCT Control Block**




# UniPHY Signals

This section describes the UniPHY signals. Table 6–4 shows the clock and reset signals.

**Table 6–4. Clock and Reset Signals**

| Name | Direction | Description |
|---|---|---|
| pll_ref_clk | Input | PLL reference clock input. |
| global_reset_n | Input | Active low global reset for PLL and all logic in the PHY, which causes a complete reset of the whole system. |
| soft_reset_n | Input | Holding soft_reset_n low holds the PHY in a reset state. However it does not reset the PLL, which keeps running. It also holds the afi_reset_n output low. Mainly for use by SOPC Builder. |
| reset_request_n | Output | When the PLL is locked, reset_request_n is high. When the PLL is out of lock, reset_request_n is low. |
| seriesterminationcontrol | Input (for OCT slave) | Required signal for PHY to provide series termination calibration value. Must be connected to a user-instantiated OCT control block (alt_oct) or another UniPHY instance that is set to OCT master mode. |
| | Output(for OCT master) | Unconnected PHY signal, available for sharing with another PHY. |
| parallelterminationcontrol | Input (for OCT slave) | Required signal for PHY to provide series termination calibration value. Must be connected to a user-instantiated OCT control block (alt_oct) or another UniPHY instance that is set to OCT master mode. |
| | Output (for OCT master) | Unconnected PHY signal, available for sharing with another PHY. |
| oct_rdn | Input (for OCT master) | Must connect to calibration resistor tied to GND on the appropriate RDN pin on the device. (See appropriate device handbook.) |
| oct_rup | Input (for OCT master) | Must connect to calibration resistor tied to $V_{ccio}$ on the appropriate RUP pin on the device. (See appropriate device handbook.) |

Table 6–5 shows the AFI signals.

**Table 6–5. AFI Signals**

| Name | Direction | Width | Description |
|---|---|---|---|
| **Clocks and Reset** | | | |
| afi_clk | Output | 1 | Half-rate or full-rate clock supplied to controller and system logic. |
| afi_reset_n | Output | 1 | Reset output on afi_clk clock domain. For use as asynchronous reset. This signal is asynchronously asserted and synchronously de-asserted. |
| **Address and Command** | | | |
| afi_addr | Input | MEM_ADDRESS_WIDTH × AFI_RATIO | Row address. |
| afi_ba | Input | MEM_BANK_WIDTH × AFI_RATIO | Bank address. |
| afi_cas_n | Input | MEM_CONTROL_WIDTH × AFI_RATIO | Column address strobe (CAS). |
| afi_cs_n | Input | MEM_CHIP_SELECT_ WIDTH × AFI_RATIO | Chip select. |
| afi_ras_n | Input | MEM_CONTROL_WIDTH × AFI_RATIO | Row address strobe (RAS). |
| afi_we_n | Input | MEM_CONTROL_WIDTH × AFI_RATIO | Write enable. |
| **Write Data** | | | |
| afi_dm | Input | MEM_DM_WIDTH × AFI_RATIO | Data mask input that generates mem_dm. |
| afi_wdata | Input | MEM_DQ_WIDTH × 2 × AFI_RATIO | Write data input that generates mem_dq. |
| afi_wdata_valid | Input | MEM_WRITE_DQS_ WIDTH × AFI_RATIO | Write data valid that generates mem_dq and mem_dm output enables. |
| **Read Data** | | | |
| afi_rdata | Output | MEM_DQ_WIDTH × 2 × AFI_RATIO | Read data |
| afi_rdata_en | Input | MEM_READ_DQS_ WIDTH × AFI_RATIO | Doing read input. Indicates that the memory controller is currently performing a read operation. |
| afi_rdata_valid | Output | AFI_RATIO | Read data valid indicating valid read data on afi_rdata, in the byte lanes and alignments that were indicated on afi_rdata_en. |
| **Calibration Control and Status** | | | |
| afi_cal_success | Output | 1 | '1' signals that calibration has completed |
| afi_cal_fail | Output | 1 | '1' signals that calibration has failed |

Table 6–6 shows the sideband signals.

**Table 6–6. Sideband Signals**

| Signal name | Direction | Width | Description |
|---|---|---|---|
| oct_ctl_rs_value | Input | OCT_SERIES_TERM_CONTROL_WIDTH | OCT Rs value port for use with ALTOCT megafunction. |
| oct_ctl_rt_value | Input | OCT_PARALLEL_TERM_CONTROL_WIDTH | OCT Rt value port for use with ALTOCT megafunction. |

Table 6–7 shows the RLDRAM II interface signals.

**Table 6–7. RLDRAM II Interface Signals**

| Name | Direction | Width | Description |
|---|---|---|---|
| mem_a | Output | MEM_ADDRESS_WIDTH | Address. |
| mem_ba | Output | MEM_CONTROL_WIDTH | Bank address. |
| mem_ck, mem_ck_n | Output | 1 | Address and command clock to memory. |
| mem_dk, mem_dkn | Output | MEM_WRITE_DQS_ WIDTH | Write clock(s) to memory, 1 clock per DQS group. |
| mem_dm | Output | MEM_DM_WIDTH | Data mask. |
| mem_dq | Bidirectional | MEM_DQ_WIDTH | Input and output data bus. |
| mem_ref_n | Output | MEM_CONTROL_WIDTH | Connecting this pin to ground turns off the DLL inside the device. |
| mem_qk, mem_qk_n | Input | MEM_READ_DQS_ WIDTH | Read clock(s) from memory, 1 clock per DQS group |
| mem_we_n | Output | MEM_CONTROL_WIDTH | Write enable. |

Table 6–8 shows the top-level signals generated for HardCopy migration.

**Table 6–8. Top-Level HardCopy Migration Signals (Part 1 of 2)**

| Name | Direction | Description |
|---|---|---|
| **altsyncram Signals** | | |
| hc_rom_config_clock | Input | Write clock for the ROM loader. This clock is used as the write clock of the Nios II code memory. |
| hc_rom_config_datain | Input | Data input from external ROM. |
| hc_rom_config_rom_data_ready | Input | Asserts to the code memory loader that the word memory is ready to be loaded. |
| hc_rom_config_init | Input | Triggers the ROM loading process. Should be asserted for one hc_rom_config_clock cycle after PLL is locked. |

**Table 6–8. Top-Level HardCopy Migration Signals (Part 2 of 2)**

| Name | Direction | Description |
|---|---|---|
| hc_rom_config_init_busy | Output | When asserted, indicates ROM loading is in progress. The soft_reset_n signal should be de-asserted if the ROM data is not loaded, and also when the ROM is being loaded. The falling edge of hc_rom_config_init_busy indicates the completion of the ROM loading process, at which time, soft_reset_n can be asserted. |
| hc_rom_config_rom_rden | Output | Read-enable signal that connects to the external ROM. |
| hc_rom_config_rom_address | Output | ROM address that connects to the external ROM. |
| **DLL Reconfiguration Signals** | | |
| hc_dll_config_dll_offset_ctrl_addnsub | Input | Addition and subtraction control port for the DLL. This port controls if the delay-offset setting on hc_dll_config_dll_offset_ctrl_offset is added or subtracted. |
| hc_dll_config_offset_ctrl_offset | Input | Offset input setting for the DLL. This setting is a Gray-coded offset that is added or subtracted from the current value of the DLL's delay chain. |
| hc_dll_config_dll_offset_ctrl_offsetctrlout | Output | The registered and Gray-coded value of the current delay-offset setting. |
| **PLL Reconfiguration Signals** | | |
| hc_pll_config_configupdate | Input | Control signal to enable PLL reconfiguration. |
| hc_pll_config_phasecounterselect | Input | Specifies the counter select for dynamic phase adjustment. |
| hc_pll_config_phasestep | Input | Specifies the phase step for dynamic phase shifting. |
| hc_pll_config_phaseupdown | Input | Specifies whether the phase shift is up or down. |
| hc_pll_config_scanclk | Input | PLL reconfiguration scan chain clock. |
| hc_pll_config_scanclkena | Input | Clock enable port of the hc_pll_config_scanclk clock. |
| hc_pll_config_scandata | Input | Serial input data for the PLL reconfiguration scan chain. |
| hc_pll_config_scandataout | Output | Data output of the serial scan chain. |
| hc_pll_config_scandone | Output | This signal is asserted when the scan chain write operation is in progress. This signal is deasserted when the write operation is complete. |

Table 6–9 shows the parameters that Table 6–5 through Table 6–7 mention.

**Table 6–9. Parameters (Part 1 of 2)**

| Parameter Name | Description |
|---|---|
| AFI_RATIO | AFI_RATIO is 1 in full-rate designs.<br>AFI_RATIO is 2 for half-rate designs. |
| MEM_IF_DQS_WIDTH | The number of DQS pins in the interface. |
| MEM_ADDRESS_WIDTH | The address width of the specified memory device. |
| MEM_BANK_WIDTH | The bank width of the specified memory device. |
| MEM_CHIP_SELECT_WIDTH | The chip select width of the specified memory device. |
| MEM_CONTROL_WIDTH | The control width of the specified memory device. |
| MEM_DM_WIDTH | The DM width of the specified memory device. |
| MEM_DQ_WIDTH | The DQ width of the specified memory device. |
| MEM_READ_DQS_WIDTH | The READ DQS width of the specified memory device. |
| MEM_WRITE_DQS_WIDTH | The WRITE DQS width of the specified memory device. |
| OCT_SERIES_TERM_CONTROL_WIDTH | — |
| OCT_PARALLEL_TERM_CONTROL_WIDTH | — |
| AFI_ADDRESS_WIDTH | The AFI address width, derived from the corresponding memory interface width. |
| AFI_BANK_WIDTH | The AFI bank width, derived from the corresponding memory interface width. |
| AFI_CHIP_SELECT_WIDTH | The AFI chip-select width, derived from the corresponding memory interface width. |
| AFI_DATA_MASK_WIDTH | The AFI data mask width. |
| AFI_CONTROL_WIDTH | The AFI control width, derived from the corresponding memory interface width. |
| AFI_DATA_WIDTH | The AFI data width. |
| AFI_DQS_WIDTH | The AFI DQS width. |
| DLL_DELAY_CTRL_WIDTH | The DLL delay output control width. |
| NUM_SUBGROUP_PER_READ_DQS | A read datapath parameter for timing purposes. |
| QVLD_EXTRA_FLOP_STAGES | A read datapath parameter for timing purposes. |
| READ_VALID_TIMEOUT_WIDTH | A read datapath parameter; calibration fails when the timeout counter expires. |
| READ_VALID_FIFO_WRITE_ADDR_WIDTH | A read datapath parameter; the write address width for half-rate clocks. |
| READ_VALID_FIFO_READ_ADDR_WIDTH | A read datapath parameter; the read address width for full-rate clocks. |
| MAX_LATENCY_COUNT_WIDTH | A latency calibration parameter; the maximum latency count width. |
| MAX_READ_LATENCY | A latency calibration parameter; the maximum read latency. |
| READ_FIFO_READ_ADDR_WIDTH | — |
| READ_FIFO_WRITE_ADDR_WIDTH | — |
| MAX_WRITE_LATENCY_COUNT_WIDTH | A write datapath parameter; the maximum write latency count width. |
| INIT_COUNT_WIDTH | An initailization sequence. |
| MRSC_COUNT_WIDTH | An RLDRAM II-specific initialization parameter. |
| INIT_NOP_COUNT_WIDTH | An RLDRAM II-specific initialization parameter. |

**Table 6–9. Parameters (Part 2 of 2)**

| Parameter Name | Description |
|---|---|
| MRS_CONFIGURATION | An RLDRAM II-specific initialization parameter. |
| MRS_BURST_LENGTH | An RLDRAM II-specific initialization parameter. |
| MRS_ADDRESS_MODE | An RLDRAM II-specific initialization parameter. |
| MRS_DLL_RESET | An RLDRAM II-specific initialization parameter. |
| MRS_IMP_MATCHING | An RLDRAM II-specific initialization parameter. |
| MRS_ODT_EN | An RLDRAM II-specific initialization parameter. |
| MRS_BURST_LENGTH | An RLDRAM II-specific initialization parameter. |
| MEM_T_WL | An RLDRAM II-specific initialization parameter. |
| MEM_T_RL | An RLDRAM II-specific initialization parameter. |
| SEQ_BURST_COUNT_WIDTH | The burst count width for the sequencer. |
| VCALIB_COUNT_WIDTH | The width of a counter used by the sequencer. |
| DOUBLE_MEM_DQ_WIDTH | — |
| HALF_AFI_DATA_WIDTH | — |
| CALIB_REG_WIDTH | The width of the calibration status register. |
| NUM_AFI_RESET | The number of AFI resets to generate. |

## AFI Signal Names

The RLDRAM II controller with UniPHY uses AFI.

The AFI timing is identical to the DDR3 SDRAM AFI in the Quartus II software version 9.0. However, some signals have been renamed, some added, and others removed from the AFI definition. The AFI includes only signals that are part of the controller-to-PHY interface, clocks, and reset. All signals on the controller-to-PHY interface have the **afi**_ prefix to the signal name. Table 6–10 shows the renamed AFI signals and original (Quartus II software version 9.0) names.

**Table 6–10. AFI New Signal Names**

| AFI Name | Old Name |
|---|---|
| afi_clk | ctl_clk |
| afi_reset_n | ctl_reset_n |
| afi_addr | ctl_addr |
| afi_ba | ctl_ba |
| afi_cke | ctl_cke |
| afi_cs_n | ctl_cs_n |
| afi_ras_n | ctl_ras_n |
| afi_we_n | ctl_we_n |
| afi_cas_n | ctl_cas_n |
| afi_dqs_burst | ctl_dqs_burst |
| afi_wdata_valid | ctl_wdata_valid |
| afi_wdata | ctl_wdata |
| afi_dm | ctl_dm |

**Table 6–10. AFI New Signal Names**

| AFI Name | Old Name |
|---|---|
| afi_wlat | ctl_wlat |
| afi_rdata_en | ctl_doing_read |
| afi_rdata | ctl_rdata |
| afi_mem_clk_disable | ctl_mem_clk_disable |
| afi_cal_success | ctl_cal_success |
| afi_cal_fail | ctl_cal_fail |
| afi_cal_req | ctl_cal_req |

# PHY-to-Controller Interfaces

This section describes the typical modules that are connected to the UniPHY PHY and the port name prefixes each module uses. This section describes using a custom controller and describes the AFI.

The AFI standardizes and simplifies the interface between controller and PHY for all Altera memory designs, thus allowing you to easily interchange your own controller code with Altera's high-performance controllers. The AFI PHY includes an administration block that configures the memory for calibration and performs necessary accesses to mode registers that configure the memory as required (these calibration processes are different).

For half-rate designs, the address and command signals in the UniPHY are asserted for one mem_clk cycle (1T addressing), such that there are two input bits per address and command pin in half-rate designs. If you require a more conservative 2T addressing, drive both input bits (of the address and command signal) identically in half-rate designs.

Figure 6–8 shows the half-rate write operation.

**Figure 6–8. Half-Rate Write with Word-Aligned Data**

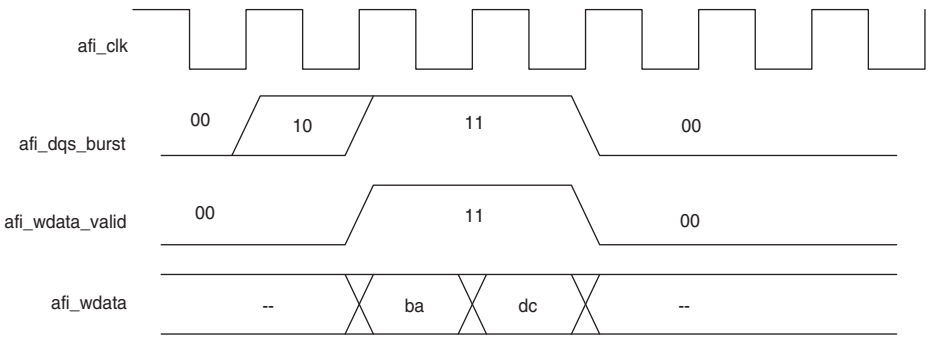
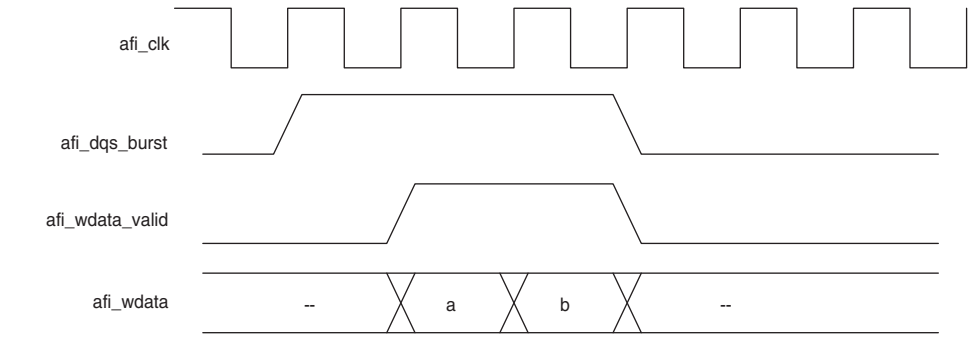


Figure 6–9 shows a full-rate write.

**Figure 6–9. Full-Rate Write**




After calibration completes, the sequencer sends the write latency in number of clock cycles to the controller.

Figure 6–10 shows full-rate reads; Figure 6–11 shows half-rate reads.

**Figure 6–10. Full-Rate Reads**

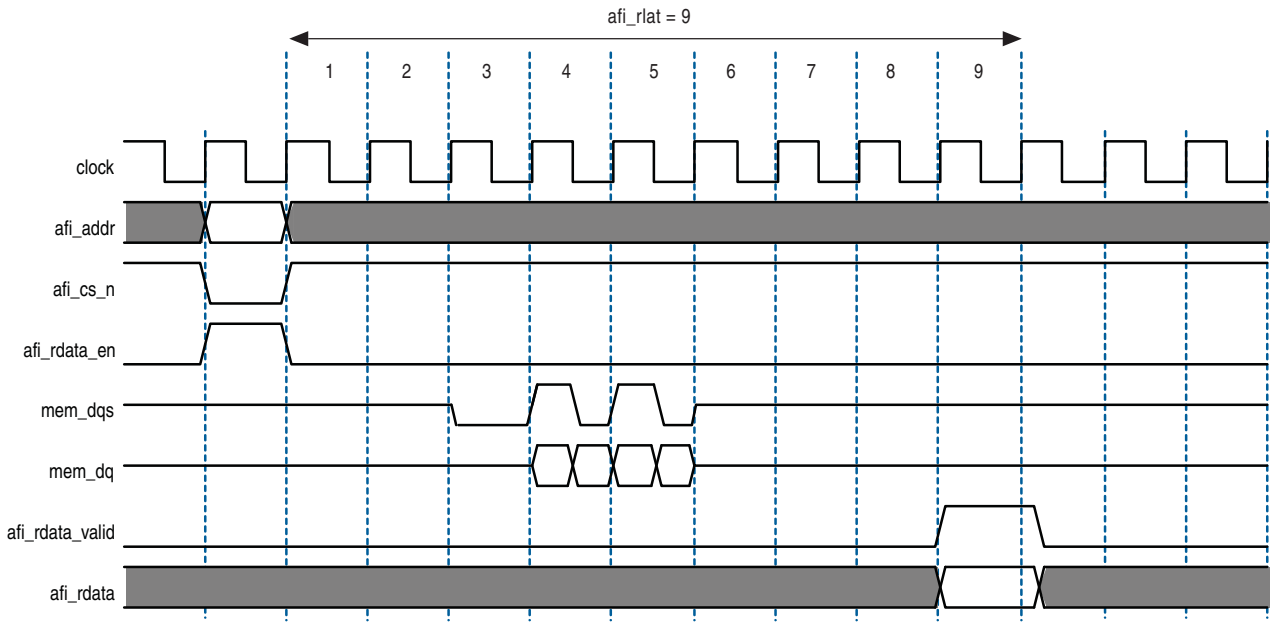


**Figure 6–11. Half-Rate Reads**

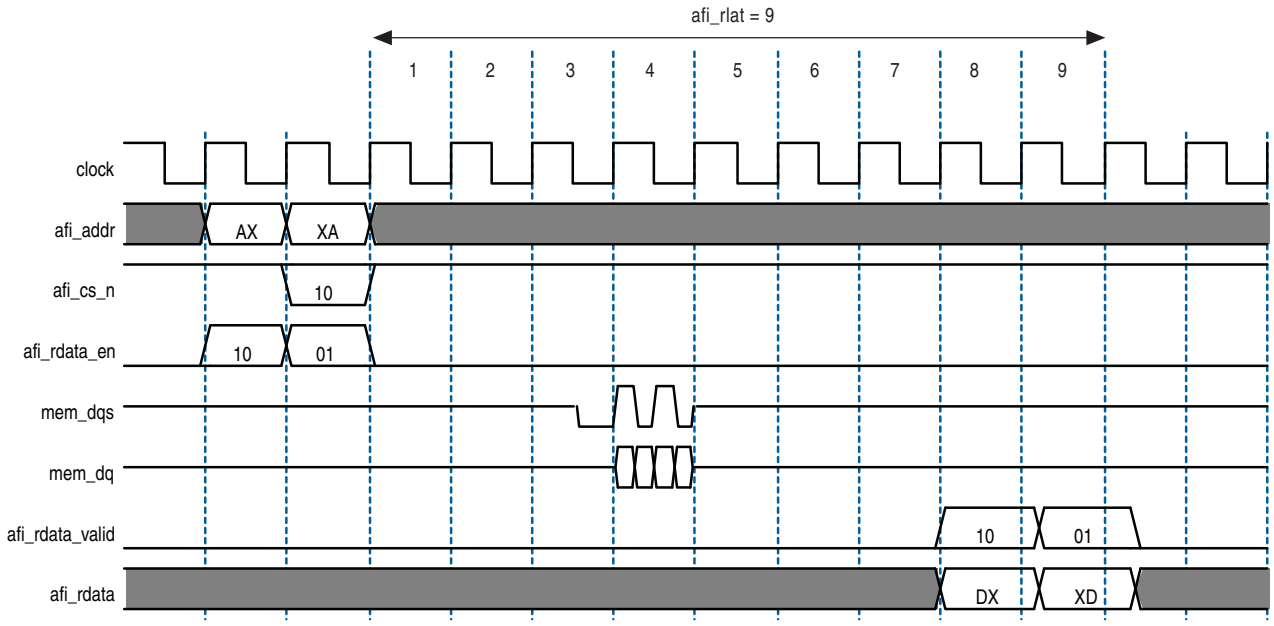



Figure 6–12 and Figure 6–13 show writes and reads, where the data is written to and read from the same address. In each example, `afi_rdata` and `afi_wdata` align with controller clock (`afi_clk`) cycles. All the data in the bit vector is valid at once.

The AFI has the following conventions:

- With the AFI, high and low signals are combined in one signal, so for a single chip-select (`afi_cs_n`) interface, `afi_cs_n[1:0]`, where location 0 appears on the memory bus on one `mem_clk` cycle and location 1 on the next `mem_clk` cycle.

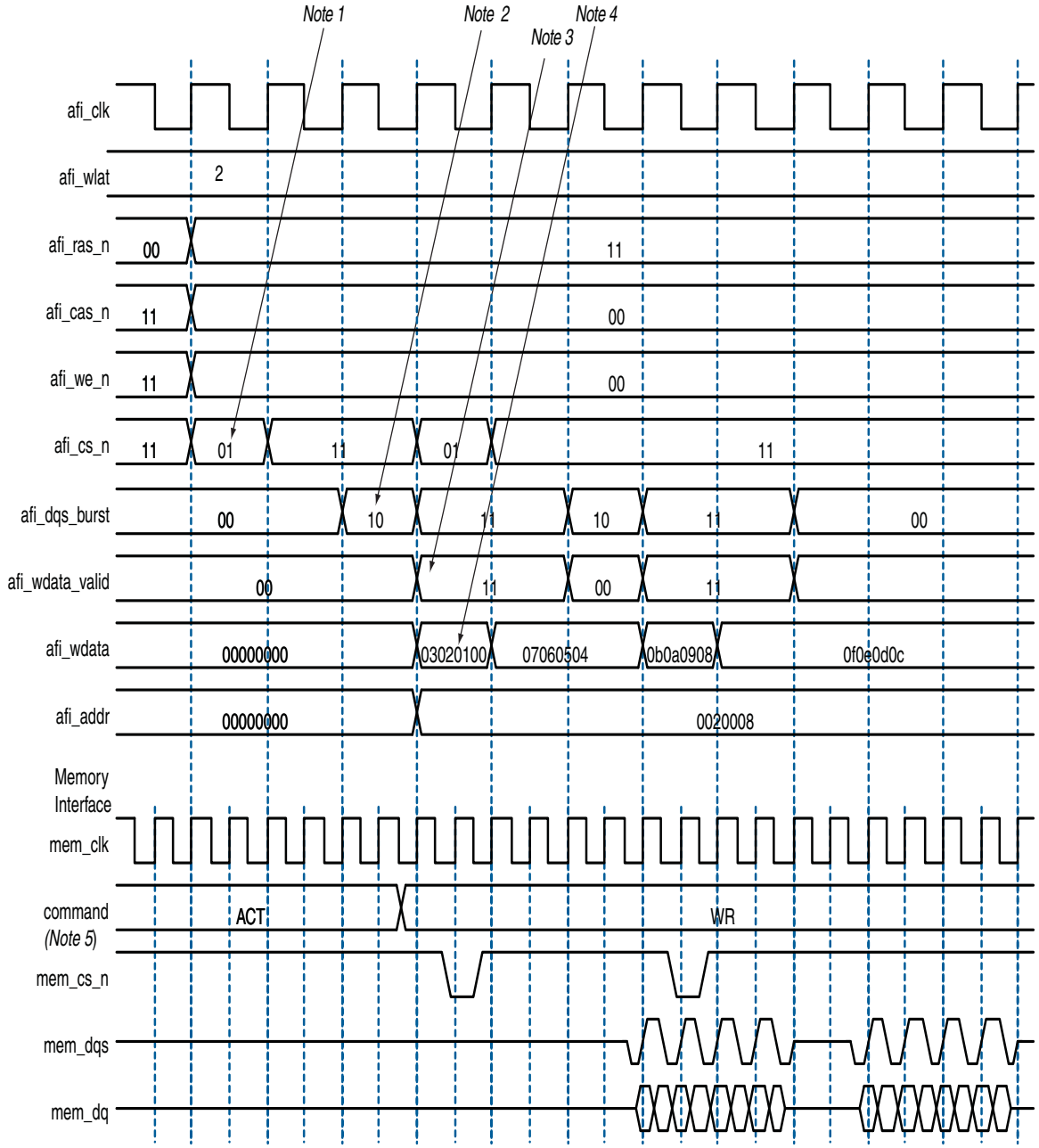
  This convention applies for all signals, so for an 9-bit memory interface, the write data (`afi_wdata`) signal is `afi_wdata[31:0]`, where the first data on the DQ pins is `afi_wdata[7:0]`, then `afi_wdata[15:8]`, then `afi_wdata[23:16]`, then `afi_wdata[31:24]`.

- Spaced reads and writes have the following definitions:
  - Spaced writes—write commands separated by a gap of one controller clock (`afi_clk`) cycle.
  - Spaced reads—read commands separated by a gap of one controller clock (`afi_clk`) cycle.

Figure 6–12 through Figure 6–13 assume the following general points:

- The burst length is four.
- An 9-bit interface with one chip-select.
- The data for one controller clock (`afi_clk`) cycle represents data for two memory clock (`mem_clk`) cycles (half-rate interface).

**Figure 6–12. Word-Aligned Writes**




**Notes to Figure 6–12:**

(1) To show the even alignment of `afi_cs_n`, expand the signal (this convention applies for all other signals).

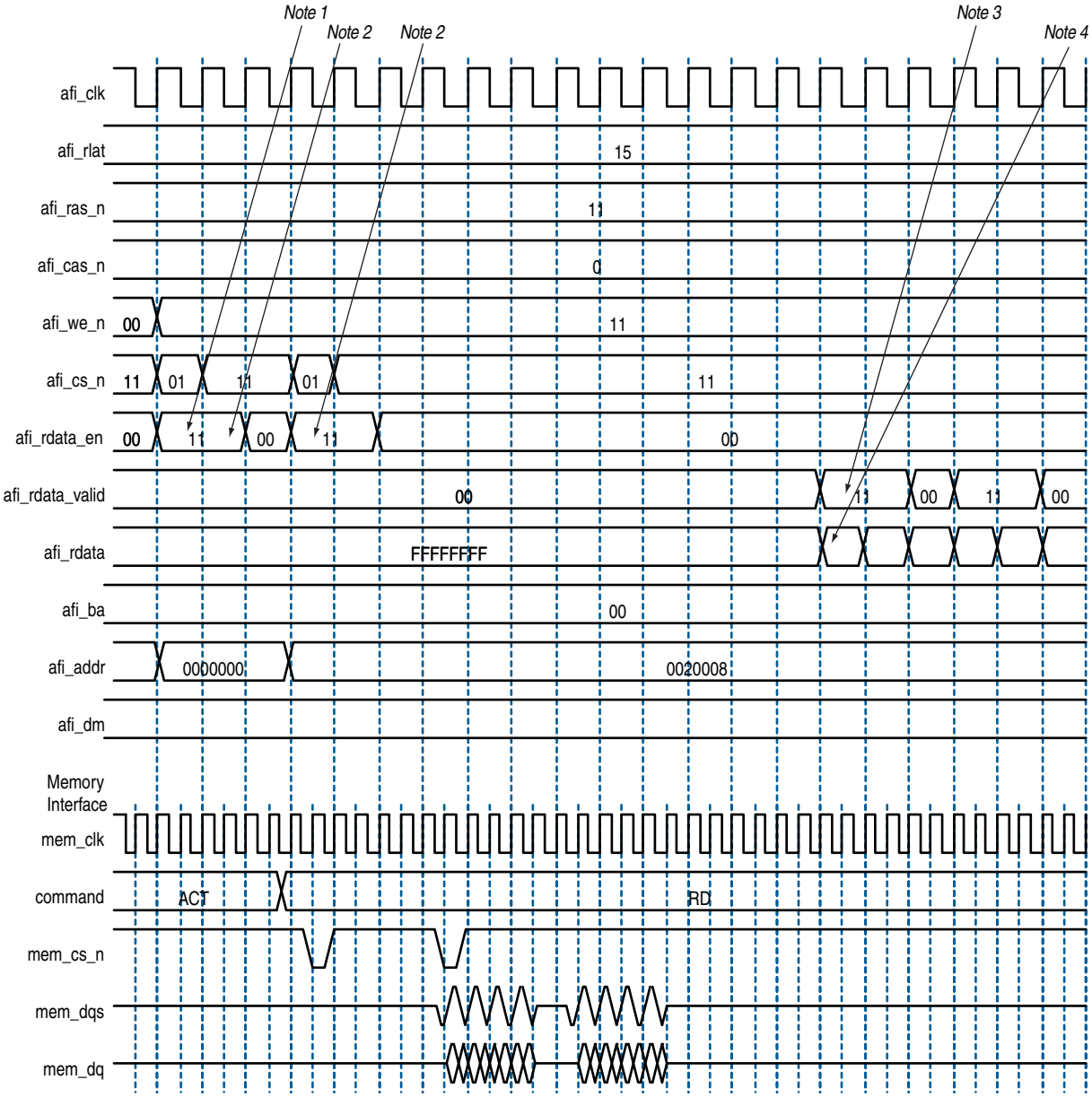
(2) The `afi_dqs_burst` must go high one memory clock cycle before `afi_wdata_valid`. Compare with the word-unaligned case.

(3) The `afi_wdata_valid` is asserted two `afi_wlat` controller clock (`afi_clk`) cycles after chip select (`afi_cs_n`) is asserted. The `afi_wlat` indicates the required write latency in the system. The value is determined during calibration and is dependant upon the relative delays in the address and command path and the write datapath in both the PHY and the external DDR SDRAM subsystem. The controller must drive `afi_cs_n` and then wait `afi_wlat` (two in this example) `afi_clks` before driving `afi_wdata_valid`.

(4) Observe the ordering of write data (`afi_wdata`). Compare this to data on the `mem_dq` signal.

(5) In all waveforms a command record is added that combines the memory pins `ras_n`, `cas_n` and `we_n` into the current command that is issued. This command is registered by the memory when chip select (`mem_cs_n`) is low. The important commands in the presented waveforms are WR = write, ACT = activate.

**Figure 6–13. Word-Aligned Reads**




**Notes to Figure 6–13:**

(1) For AFI, `afi_rdata_en` is required to be asserted one memory clock cycle before chip select (`afi_cs_n`) is asserted. In the half-rate `afi_clk` domain, this requirement manifests as the controller driving `11` (as opposed to the `01`) on `afi_rdata_en`.

(2) AFI requires that `afi_rdata_en` is driven for the duration of the read. In this example, it is driven to `11` for two half-rate `afi_clks`, which equates to driving to `1`, for the four memory clock cycles of this four-beat burst.

(3) The `afi_rdata_valid` returns 15 (`afi_rlat`) controller clock (`afi_clk`) cycles after `afi_rdata_en` is asserted. Returned is when the `afi_rdata_valid` signal is observed at the output of a register within the controller. A controller can use the `afi_rlat` value to determine when to register to returned data, but this is unnecessary as the `afi_rdata_valid` is provided for the controller to use as an enable when registering read data.

(4) Observe the alignment of returned read data relative to data on the bus.

# Using a Custom Controller

The UniPHY-based memory interface IP cores integrate the PHY and the memory controller. To replace the Altera high-performance memory controller with a custom memory controller, perform the following steps:

1. Parameterize and generate your variation of the UniPHY-based memory controller IP as described in "Getting Started" on page 2–1.

   This step creates a top-level HDL file called <*variation_name*>**.v** (or <*variation_name*>**.vhd**), and a sub-directory called <*variation_name*>.

   The top-level module instantiates the <*variation_name*>_<*stamp*>**_controller_phy** module in the <*variation_name*> subdirectory. The <*variation_name*>_<*stamp*>**_controller_phy** module instantiates the PHY and the controller.

    <*stamp*> is a unique identifier determined by the MegaWizard Plug-in Manager, SOPC Builder, or Qsys, during generation.

2. Open the <*variation_name*>**/**<*variation_name*>_<*stamp*>**_controller_phy.sv** file.
3. Replace the <*variation_name*>_<*stamp*>**_alt_rld_controller** module with your custom controller module.
4. Delete the ports of the Altera high-performance memory controller, and add the top-level ports of your custom controller.
5. Similarily, update the port names in the top-level module in the <*variation_name*>**.v** or <*variation_name*>**.vhd** file.
6. Compile and simulate the design to confirm correct operation.

 Regenerating the UniPHY memory interface IP erases all modifications made to the HDL files. The parameters you select in the parameter editor are stored in the top-level <*variation_name*> module; hence, you must repeat the above steps every time you regenerate the IP variation.

 For half-rate controllers, AFI signals are double the bus width of the memory interface. Half rate controllers have double the width of the signal and run at half the speed. Hence, the overall bandwidth is maintained. Such double-width signals are divided into two signals for transmission to the memory interface, with a higher order bits representing the most-significant bit (MSB) and a lower order bits representing the least-significant bit (LSB). The LSB is transmitted first, and is followed by the MSB.

# Using a Vendor-Specific Memory Model

You can replace the Altera-supplied memory model with a vendor-specific memory model. In general, you may find vendor-specific models to be standardized, thorough, and well supported, but sometimes more complex to setup and use.

If you do want to replace the Altera-supplied memory model with a vendor-supplied memory model, observe the following guidelines:

- Ensure that the vendor-supplied memory model that you have is correct for your memory devices.
- Disconnect all signals from the default memory model and reconnect them to the vendor-supplied memory model.
- If you intend to run simulation from the Quartus II software, ensure that the **.qip** file points to the vendor-supplied memory model.

# 7. Functional Description—Example Top-Level Project

IP generation creates an example top-level project that shows you how to instantiate and connect the controller.

The example top-level project contains a testbench, which is for use with Verilog HDL only language simulators such as ModelSim-AE Verilog, and shows simple operation of the memory interface.

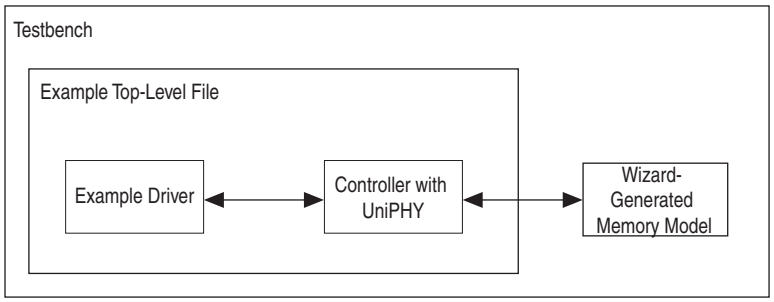
☞ For a VHDL-only simulation, use the VHDL IP functional simulation model.

The testbench contains the following blocks:

- A synthesizable Avalon-MM example driver, which acts as a traffic generator block and implements a pseudo-random pattern of reads and writes to a parameterized number of addresses. The driver also monitors the data read from the memory to ensure it matches the written data and asserts a failure otherwise.
- An instance of the controller, which interfaces between the Avalon-MM interface and the AFI.
- The UniPHY IP, which serves as an interface between the memory controller and external memory device(s) to perform read and write operations to the memory.
- A memory model, which acts as a generic model that adheres to the memory protocol specifications. Memory vendors also provide simulation models for specific memory components that can be downloaded from their websites. This block is available in Verilog HDL only.

Figure 7–1 shows the testbench and the example top-level file.

**Figure 7–1. Testbench and Example Top-Level File**



# Example Driver

The example driver for Avalon-MM memory interfaces generates Avalon-MM traffic on an Avalon-MM master interface. As the read and write traffic is generated, the expected read response is stored internally and compared to the read responses as they arrive. If all reads report their expected response, the pass signal is asserted; however, if any read responds with unexpected data a fail signal is asserted.

Each operation generated by the driver is a single write or block of writes followed by a single read or block of reads to the same addresses, which allows the driver to precisely determine the data that should be expected when the read data is returned by the memory interface. The driver comprises a traffic generation block, the Avalon-MM interface and a read comparison block. The traffic generation block generates addresses and write data, which are then sent out over the Avalon-MM interface. The read comparison block compares the read data received from the Avalon-MM interface to the write data from the traffic generator. If at any time the data received is not the expected data, the read comparison block records the failure, finishes reading all the data, and then signals that there is a failure and the driver enters a fail state. If all patterns have been generated and compared successfully, the driver enters a pass state.

Within the driver, there are the following main states:

- Generation of individual read and writes
- Generation of block read and writes
- The pass state
- The fail state

Within each of the generation states there are the following substates:

- Sequential address generation
- Random address generation
- Mixed sequential and random address generation

For each of the states and substates, the order and number of operations generated for each substate is parameterizable—you can decide how many of each address pattern to generate, or can disable certain patterns entirely if you want. The sequential and random interleave substate takes in additions to the number of operations to generate. An additional parameter specifies the ratio of sequential to random addresses to generate randomly.

## Read and Write Generation

The traffic generator block can perform individual or block read and write generation.

### Individual Read and Write Generation

During the individual read and write generation stage of the driver, the traffic generation block generates individual write followed by individual read Avalon-MM transactions, where the address for the transactions are chosen according to the specific substate. The width of the Avalon-MM interface is a global parameter for the driver, but each substate can have a parameterizable range of burst lengths for each operation.

### Block Read and Write Generation

During the block read and write generation state of the driver, the traffic generator block generates a parameterizable number of write operations followed by the same number of read operations. The specific addresses generated for the blocks are chosen by the specific substates. The burst length of each block operation can be parameterized by a range of acceptable burst lengths.

## Address and Burst Length Generation

The traffic generator block can perform sequential or random addressing.

### Sequential Addressing

The sequential addressing substate defines a traffic pattern where addresses are chosen in sequential order starting from a user definable address. The number of operations in this substate is parameterizable.

### Random Addressing

The random addressing substate defines a traffic pattern where addresses are chosen randomly over a parameterizable range. The number of operations in this substate is parameterizable.

### Sequential and Random Interleaved Addressing

The sequential and random interleaved addressing substate defines a traffic pattern where addresses are chosen to be either sequential or random based on a parameterizable ratio. The acceptable address range is parameterizable as is the number of operations to perform in this substate.

## Example Driver Signals

Table 7–1 lists the signals used by the example driver.

**Table 7–1. Driver Signals (Part 1 of 2)**

| Signal | Width | Signal Type |
|---|---|---|
| clk | | |
| reset_n | | |
| avl_ready | | avl_ready |
| avl_write_req | | avl_write_req |
| avl_read_req | | avl_read_req |
| avl_addr | 24 | avl_addr |
| avl_size | 3 | avl_size |
| avl_wdata | 72 | avl_wdata |
| avl_rdata | 72 | avl_rdata |
| avl_rdata_valid | | avl_rdata_valid |
| pnf_per_bit | | pnf_per_bit |
| pnf_per_bit_ persist | | pnf_per_bit_persist |

**Table 7–1. Driver Signals (Part 2 of 2)**

| Signal | Width | Signal Type |
|---|---|---|
| pass | | pass |
| fail | | fail |
| test_complete | | test_complete |

## Example Driver Add-Ons

Some optional components that can be useful for verifying aspects of the controller and PHY operation are generated in conjunction with certain user-specified options. These add-on components are self-contained, and are not part of the controller or PHY, nor the example driver.

### User Refresh Generator

The user refresh generator sends refresh requests to the memory controller when user refresh is enabled. The memory controller returns an acknowledgement signal and then issues the refresh command to the memory device.

The user refresh generator is created when you turn on **Enable User Refresh** under **Controller Settings** on the **General Settings** tab of the parameter editor. The user refresh generator is instantiated by **example_top_v**. and resides in the example_project subdirectory.

### Refresh Monitor

As its name implies, the refresh monitor monitors refresh commands from the controller and verifies that those commands conform to the necessary refresh timing parameters.

The refresh monitor is created when you turn on **Enable User Refresh** under **Controller Settings** on the **General Settings** tab of the parameter editor. The refresh monitor is instantiated by **example_top_tb.v** and resides in **refresh_monitor.sv** in the **rtl_sim** subdirectory.

### Data Corrupter

The data corrupter intercepts read data in the memory interface bus and introduces errors to that data to test the error detection function in the memory controller. Both the rate of error injection and the number of error bits are configurable (although the per-byte parity protection feature supports only 1 bit error detection).

The data corrupter employs four types of error injection:

- per-bit data corruption in a single memory burst
- per-byte corruption in a single memory burst
- per-bit all-burst corruption
- per-byte all burst corruption

Throughout the four types of error injection tests, the data corrupter exercises a walking-one pattern to confirm correctness.

The data corrupter is created when you turn on **Enable Error Detection Parity** under **Controller Settings** on the **General Settings** tab of the parameter editor. The data corrupter resides in **data_corrupter.sv** in the **rtl_sim** subdirectory.

# 8. Latency

Altera defines read and write latencies in terms of memory clock cycles. These latencies apply to supported device families (Table 1–2 on page 1–2).

There are two types of latencies that exists while designing with memory controllers—read and write latencies, which have the following definitions:

- Read latency—the amount of time it takes for the read data to appear at the local interface after initiating the read request.
- Write latency—the amount of time it takes for the write data to appear at the memory interface after initiating the write request.

For a half-rate controller, the local side frequency is half of the memory interface frequency. For a full-rate controller, the local side frequency is equal to the memory interface frequency.

Table 8–1 shows the latency in full rate memory clock cycles.

**Table 8–1. Latency (In Full-Rate Memory Clock Cycles) *(Note 3)***

| Rate | Controller Address and Command *(4)* | PHY Address and Command | Memory Maximum Read | PHY Read Return | Round Trip | Round Trip without Memory |
|---|---|---|---|---|---|---|
| Full | 2 | 1 | 3–8 | 4 | 10–15 | 7 |
| Half | 4 | 1 *(2)*<br>2 *(1)* | 3–8 | 7 | 16–20 *(2)*<br>16–18 *(1)* | 12 *(2)*<br>13 *(1)* |

**Notes to Table 8–1:**

(1) Even write latency.

(2) Odd write latency.

(3) Latency is the number of cycles between the first register of the current stage capturing cmd/data, and the first register in the next stage capturing cmd/data.

(4) Latency shown is best case, for maximum performance specifications. Latency may be higher due to protocol requirements; controller latency may be lower for slower frequencies.

## Variable Controller Latency

The variable controller latency feature allows you to take advantage of lower latency for variations designed to run at lower frequency. When deciding whether to vary the controller latency from the default value of 2, be aware of the following considerations:

- Reduced latency can help achieve a reduction in resource usage and clock cycles in the controller, but might result in lower $f_{MAX}$.
- Increased latency can help achieve greater $f_{MAX}$, but may consume more clock cycles in the controller and result in increased resource usage.

If you select a latency value that is inappropriate for the target frequency, the system displays a warning message in the text area at the bottom of the parameter editor.

You can change the controller latency by altering the value of the **Controller Latency** setting in the **Controller Settings** section of the **General Settings** tab of the RLDRAM II Controller with UniPHY parameter editor.

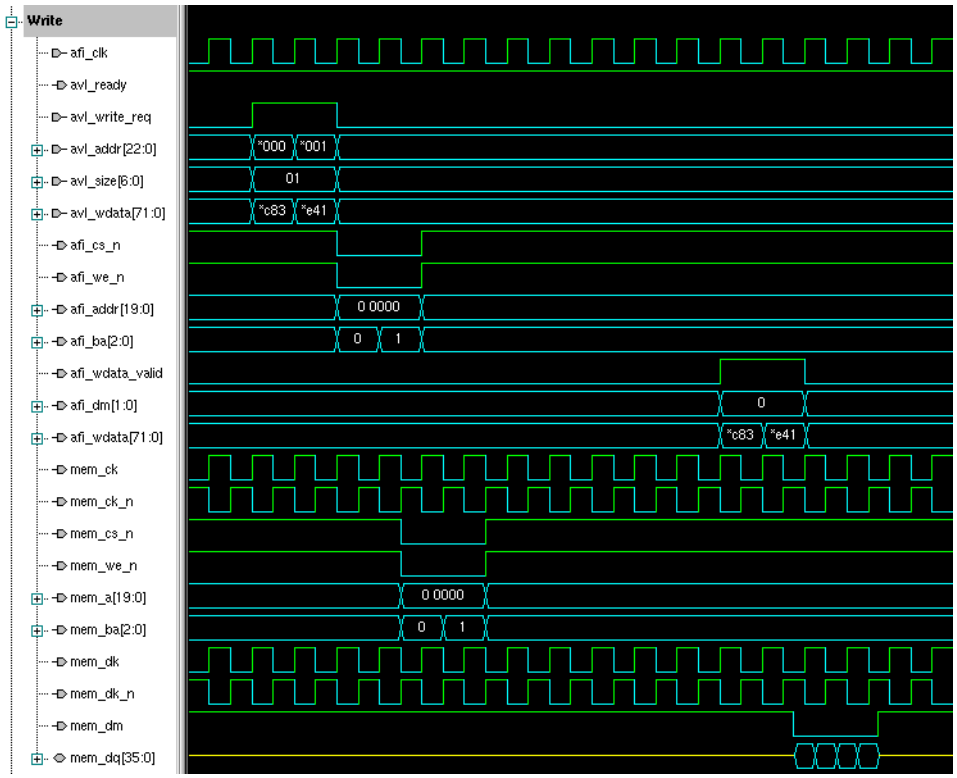
# 9. Timing Diagrams

This chapter details the following timing diagrams for a RLDRAM II controller with the following parameters:

- ×36
- Full rate
- Burst length 2

Figure 9–1 shows back-to-back write to addresses 0 and 1.

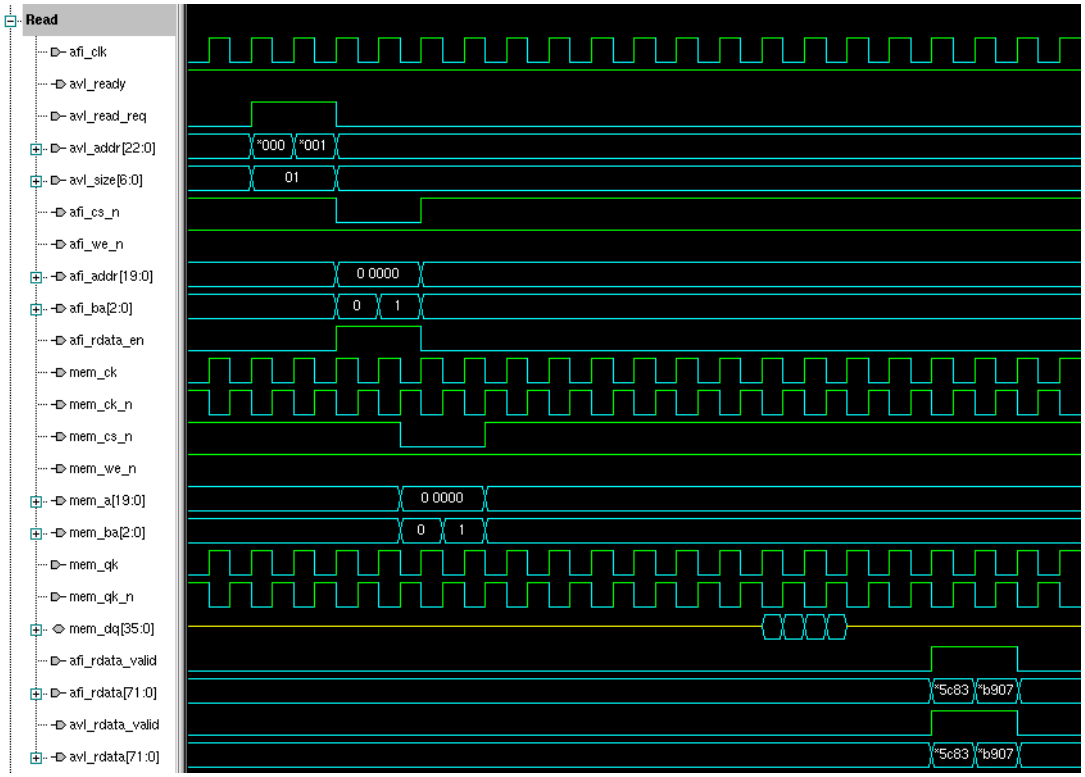
**Figure 9–1. Back-to-Back Writes**




You can set the `avl_size` to `0x2` and hold `avl_addr` constant at `0x0` to perform the same back-to-back write.

Figure 9–2 shows back-to-back read from addresses 0 and 1.

**Figure 9–2. Back-to-Back Reads**




You can set the `avl_size` to `0x2` and hold `avl_addr` constant at `0x0` to perform the same back-to-back read.

Figure 9–3 shows refresh to bank 0.

**Figure 9–3. Refresh to Bank 0**

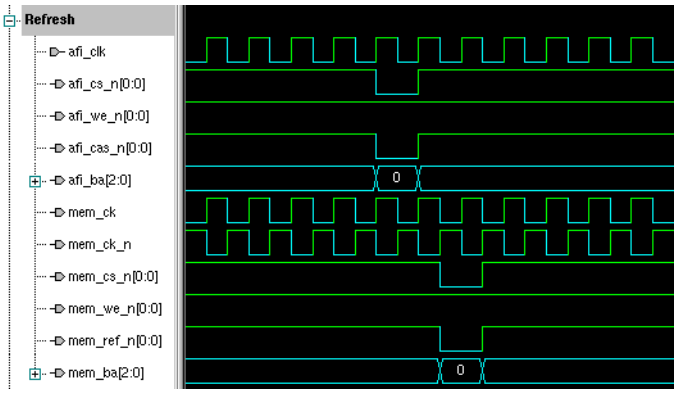


Figure 9–4 shows read-to-write signals.

**Figure 9–4. Read-to-Write Signals**

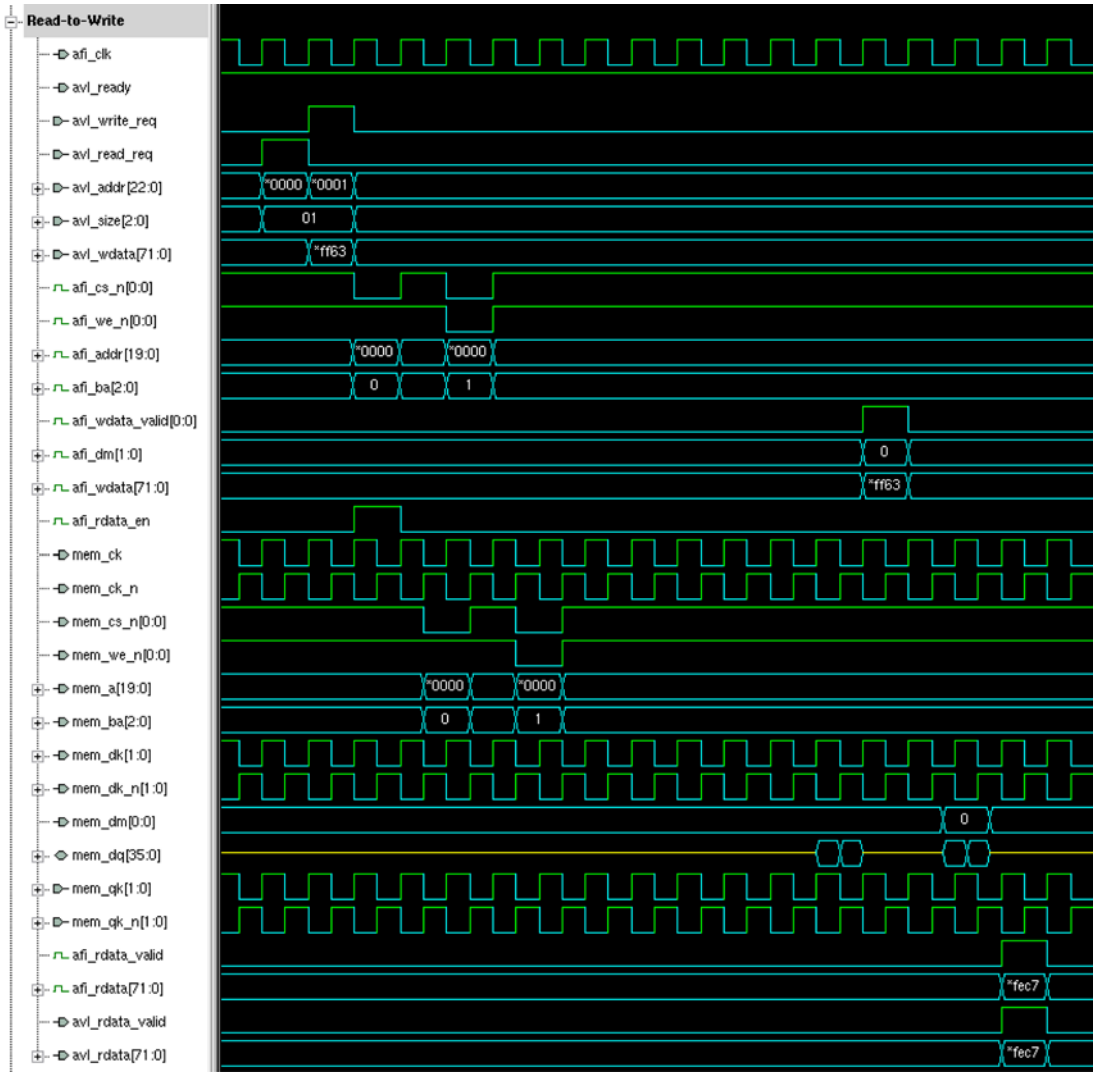
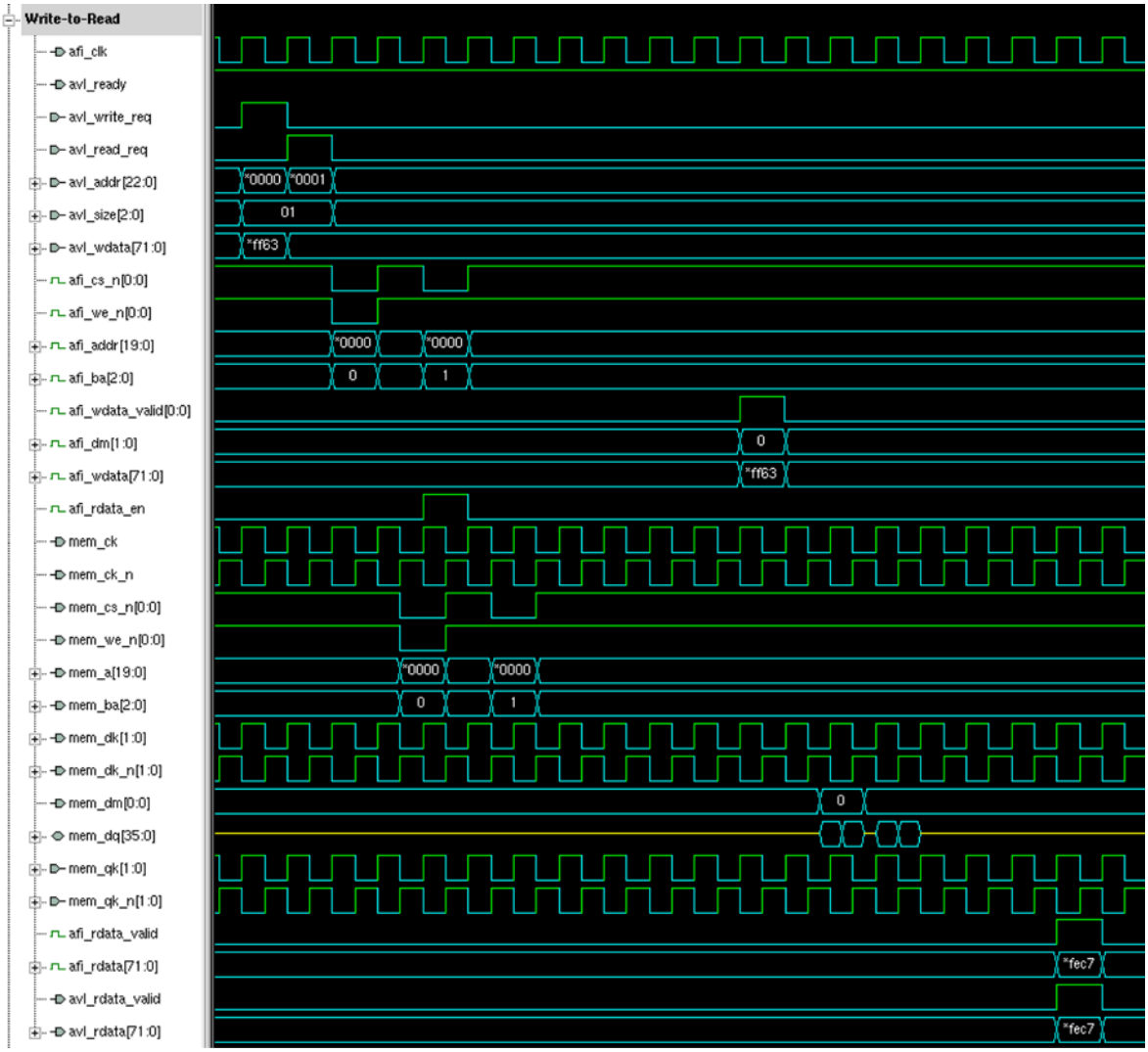


Figure 9–5 shows write-to-read signals.

**Figure 9–5. Write-to-Read Signals**



This chapter provides additional information about the document and Altera.

## Document Revision History

The following table shows the revision history for this document.

| Date | Version | Changes |
|---|---|---|
| December 2010 | 2.1 | ■ Updated Device Family Support, Features list, and Resource Utilization tables.<br>■ Updated Design Flows, added new Generated Files information.<br>■ Added information to HardCopy Migration Design Guidelines.<br>■ Added new Parameter Settings chapter.<br>■ Updated Reset and Clock Generation, and Read Datapath information.<br>■ Added The OCT Sharing Interface.<br>■ Updated Latency information. |
| July 2010 | 2.0 | Updated for the Altera Complete Design Suite version 10.0 release. |
| February 2010 | 1.2 | Corrected typos. |
| January 2010 | 1.1 | Updated features. |
| November 2009 | 1.0 | First published. |

## How to Contact Altera

To locate the most up-to-date information about Altera products, refer to the following table.

| Contact *(1)* | Contact Method | Address |
|---|---|---|
| Technical support | Website | www.altera.com/support |
| Technical training | Website | www.altera.com/training |
| | Email | custrain@altera.com |
| Product literature | Website | www.altera.com/literature |
| Non-technical support (General) | Email | nacomp@altera.com |
| (Software Licensing) | Email | authorization@altera.com |

**Note to Table:**

(1) You can also contact your local Altera sales office or sales representative.

# Typographic Conventions

The following table shows the typographic conventions this document uses.

| Visual Cue | Meaning |
|---|---|
| **Bold Type with Initial Capital Letters** | Indicate command names, dialog box titles, dialog box options, and other GUI labels. For example, **Save As** dialog box. For GUI elements, capitalization matches the GUI. |
| **bold type** | Indicates directory names, project names, disk drive names, file names, file name extensions, software utility names, and GUI labels. For example, **\qdesigns** directory, **D:** drive, and **chiptrip.gdf** file. |
| *Italic Type with Initial Capital Letters* | Indicate document titles. For example, *Stratix IV Design Guidelines*. |
| *italic type* | Indicates variables. For example, *n* + 1.<br>Variable names are enclosed in angle brackets (< >). For example, *<file name>* and *<project name>***.pof** file. |
| Initial Capital Letters | Indicate keyboard keys and menu names. For example, the Delete key and the Options menu. |
| "Subheading Title" | Quotation marks indicate references to sections within a document and titles of Quartus II Help topics. For example, "Typographic Conventions." |
| `Courier type` | Indicates signal, port, register, bit, block, and primitive names. For example, `data1`, `tdi`, and `input`. The suffix `n` denotes an active-low signal. For example, `resetn`.<br>Indicates command line commands and anything that must be typed exactly as it appears. For example, `c:\qdesigns\tutorial\chiptrip.gdf`.<br>Also indicates sections of an actual file, such as a Report File, references to parts of files (for example, the AHDL keyword `SUBDESIGN`), and logic function names (for example, `TRI`). |
| ↵ | An angled arrow instructs you to press the Enter key. |
| 1., 2., 3., and<br>a., b., c., and so on | Numbered steps indicate a list of items when the sequence of the items is important, such as the steps listed in a procedure. |
| ■ ■ ■ | Bullets indicate a list of items when the sequence of the items is not important. |
|  | The hand points to information that requires special attention. |
|  | A question mark directs you to a software help system with related information. |
|  | The feet direct you to another document or website with related information. |
|  | A caution calls attention to a condition or possible situation that can damage or destroy the product or your work. |
|  | A warning calls attention to a condition or possible situation that can cause you injury. |
|  | The envelope links to the Email Subscription Management Center page of the Altera website, where you can sign up to receive update notifications for Altera documents. |

# External Memory Interface Handbook

# Volume 4: Simulation, Timing Analysis, and Debugging

101 Innovation Drive
San Jose, CA 95134
www.altera.com

EMI_DEBUG-2.1

Document last updated for Altera Complete Design Suite version: 10.1
Document publication date: December 2010

Subscribe

# External Memory Interface Handbook Volume 4

## Section I. Simulation

101 Innovation Drive
San Jose, CA 95134
www.altera.com

EMI_DEBUG_VERIFY-2.1

Document last updated for Altera Complete Design Suite version: 10.1
Document publication date: December 2010

Subscribe

# Contents

# 1. Simulation Walkthrough

For high-performance memory controllers, you can simulate the example top-level file with the MegaWizard-generated IP functional simulation models. The MegaWizard™ Plug-In generates a VHDL or Verilog HDL testbench for the example top-level file, which is in the **\testbench** directory of your project directory.

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.

The ALTMEMPHY megafunction cannot be simulated alone. To simulate the ALTMEMPHY megafunction, you must use all of the following blocks:

- Memory controller
- Example driver (to initiate read and write transactions)
- Testbench and a suitable vendor memory model

Simulating the whole memory interface is a good way to determine the latency of your system. However, the latency found in simulation may be different than the latency found on the board because functional simulation does not take into account board trace delays and different process, voltage, and temperature scenarios. For a given design on a given board, the latency found may differ by one clock cycle (for full-rate designs) or two clock cycles (for half-rate designs) upon resetting the board. Different boards can also show different latencies even with the same design.

☞ The ALTMEMPHY megafunction only supports functional simulation; it does not support gate-level simulation, for the following reasons:

- The ALTMEMPHY is a calibrated interface. Therefore, gate-level simulation time can be very slow and take up to several hours to complete.
- Gate-level timing annotations, and the phase sweeping that the calibration uses, determine setup and hold violations. Because of the effect of X (unknown value) propagation within the atom simulation models, this action causes gate-level simulation failures in some cases.
- Memory interface timing methodology does not match the timing annotation that gate-level simulations use. Therefore the gate-level simulation does not accurately match real device behavior.

☞ Altera recommends that you validate the functional operation of your design via RTL simulation, and the timing of your design using TimeQuest Timing Analysis.

# Before Simulating

In general, you need the following files to simulate:

- Library files from the <*Quartus II install path*>\quartus\eda\sim_lib\ directory:
  - 220model
  - altera_primitives
  - altera_mf
  - sgate
  - **arriaii_atoms, stratixiv_atoms, stratixiii_atoms, cycloneiii_atoms, stratixii_atoms, stratixiigx_atoms** (device dependent)

  If you are targeting Stratix IV devices, you need both the Stratix IV and Stratix III files (**stratixiv_atoms** and **stratixiii_atoms**) to simulate, unless you are using NativeLink.

- Sequencer wrapper file (in **.vo** or **.vho** format)
- PLL file (for example <*variation_name*>**_alt_mem_phy_pll.v** or **.vhd**)
- ALTMEMPHY modules (in the <*variation_name*>**_alt_mem_phy.v**)
- Top-level file
- User logic, or a driver for the PHY
- Testbench
- Vendor memory model

# Preparing the Vendor Memory Model

If you are using a vendor memory model, instead of the MegaWizard-generated functional simulation model, you need to make some modifications to the vendor memory model and the testbench files by following these steps:

1. Make sure the IP functional simulation model is generated by turning on **Generate Simulation Model** during the instantiate PHY and controller design flow step.
2. Obtain and copy the vendor memory model to the **\testbench** directory. For example, obtain the **ddr2.v** and **ddr2_parameters.vh** simulation model files from the Micron website and save them in the testbench directory.

   The auto-generated generic SDRAM model may be used as a placeholder for a specific vendor memory model.

   Some vendor DIMM memory models do not use data mask (DM) pin operation, which can cause calibration failures. In these cases, use the vendor's component simulation models directly.

3. Open the vendor memory model file in a text editor and specify the speed grade and device width at the top of the file. For example, you can add the following statements for a DDR2 SDRAM model file:

   ```
   'define sg25

   'define x8
   ```

   The first statement specifies the memory device speed grade as –25 (for 400 MHz operation). The second statement specifies the memory device width per DQS.

4. Check that the following statement is included in the vendor memory model file. If not, include it at the top of the file. This example is for a DDR2 SDRAM model file:

   ```
   `include "ddr2_parameters.vh"
   ```

5. Save the vendor memory model file.

6. Open the testbench in a text editor, delete the whole section between the `START MEGAWIZARD INSERT MEMORY_ARRAY` and `END MEGAWIZARD INSERT MEMORY_ARRAY` comments, instantiate the downloaded memory model, and connect its signals to the rest of the design.

7. Delete the `START MEGAWIZARD INSERT MEMORY_ARRAY` and `END MEGAWIZARD INSERT MEMORY_ARRAY` lines so that the wizard does not overwrite your changes if you use the wizard to regenerate the design.

8. Ensure that ports names and capitalization in the memory model match the portnames and capitalization in the testbench.

   
   The vendor memory model may use different pin names and capitalization than the MegaWizard-generated functional model.

9. Save the testbench file.

The original instantiation may be similar to the following code:

```
// << START MEGAWIZARD INSERT MEMORY_ARRAY
    // This will need updating to match the memory models you are using.
// Instantiate a generated DDR memory model to match the datawidth &
chipselect requirements
ddr2_mem_model mem (
        .mem_dq      (mem_dq),
        .mem_dqs     (mem_dqs),
        .mem_dqs_n   (mem_dqs_n),
        .mem_addr    (a_delayed),
        .mem_ba      (ba_delayed),
        .mem_clk     (clk_to_ram),
        .mem_clk_n   (clk_to_ram_n),
        .mem_cke     (cke_delayed),
        .mem_cs_n    (cs_n_delayed),
        .mem_ras_n   (ras_n_delayed),
        .mem_cas_n   (cas_n_delayed),
```

```
        .mem_we_n    (we_n_delayed),
        .mem_dm      (dm_delayed),
        .mem_odt     (odt_delayed)
    );
// << END MEGAWIZARD INSERT MEMORY_ARRAY
```

Replace with the following code:

```
// << START MEGAWIZARD INSERT MEMORY_ARRAY
    // This will need updating to match the memory models you are using.
// Instantiate a generated DDR memory model to match the datawidth &
chipselect requirements
ddr2 memory_0 (
    .clk     (clk_to_ram),
    .clk_n   (clk_to_ram_n),
        .cke     (cke_delayed),
       .cs_n    (cs_n_delayed),
        .ras_n   (ras_n_delayed),
        .cas_n   (cas_n_delayed),
        .we_n    (we_n_delayed),
        .dm_rdqs      (dm_delayed[0]),
    .ba      (ba_delayed),
    .addr    (a_delayed),
    .dq      (mem_dq[7:0]),
        .dqs     (mem_dqs[0]),
        .dqs_n   (mem_dqs_n[0]),
    .rdqs_n (),
        .odt     (odt_delayed)
    );
    // << END MEGAWIZARD INSERT MEMORY_ARRAY
```

If you are interfacing with a DIMM or multiple memory components, you need to instantiate all the memory components in the testbench file.

# Simulating Using NativeLink

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 top-level entity to the example project.
   a. On the File menu, click **Open**.
   b. Browse to <*variation name*>**_example_top** and click **Open**.
   c. On the Project menu, click **Set as Top-Level Entity**.

3. Ensure that the Quartus II **EDA Tool Options** are configured correctly for your simulation environment.
   a. On the Tools menu, click **Options**.
   b. In the **Category** list, click **EDA Tool Options** and verify the locations of the executable files.
4. Set up the Quartus II NativeLink.
   a. On the Assignments menu, click **Settings**. In the **Category** list, expand **EDA Tool Settings** and click **Simulation**.
   b. From the **Tool name** list, click on your preferred simulator.
   c. In **NativeLink settings**, select **Compile test bench** and click **Test Benches**.
   d. Click **New** at the **Test Benches** page to create a testbench.
5. On the **New Test Bench Settings** dialog box:
   a. Type a name for the **Test bench name**, for example <*variation name*>`_example_top_tb`.
   b. In **Top level module in test bench**, type the name of the automatically generated testbench, <*variation name*>`_example_top_tb`.
   c. In **Design instance in test bench**, type the name of the top-level instance, `dut`.
   d. Under **Simulation period**, set **End simulation at** to `600 µs`.
   e. Add the testbench files and automatically-generated memory model files. In the **File name** field, browse to the location of the memory model and the testbench, click **Open** and then click **Add**. The testbench is <*variation name*>**_example_top_tb.v**; memory model is <*variation name*>**_mem_model.v**.
   f. Select the files and click **OK**.
6. On the Processing menu, point to **Start** and click **Start Analysis & Elaboration** to start analysis.
7. On the Tools menu, point to **Run EDA Simulation Tool** and click **EDA RTL Simulation**.

If your Quartus II project appears to be configured correctly but the example testbench still fails, check the known issues on the *Knowledge Database* page before filing a service request.

For a complete MegaWizard Plug-In Manager system design example containing the DDR and DDR2 SDRAM Controller with ALTMEMPHY IP, refer to *Volume 6: Design Flow Tutorials* of the *External Memory Interface Handbook*.

# IP Functional Simulations

This topic discusses VHDL and Verilog HDL simulations with IP functional simulation models.

## VHDL

For VHDL simulations with IP functional simulation models, perform the following steps:

1. Create a directory in the <*project directory*>**\testbench** directory.
2. Launch your simulation tool from this directory and create the following libraries:
   - altera_mf
   - lpm
   - sgate
   - <*device name*>
   - altera
   - ALTGXB
   - <*device name*>_hssi
   - auk_ddr_hp_user_lib
3. Compile the files into the appropriate library (Table 1–1). The files are in VHDL93 format.

**Table 1–1. Files to Compile—VHDL IP Functional Simulation Models (Part 1 of 2)**

| Library | File Name |
|---|---|
| altera_mf | <*QUARTUS ROOTDIR*>**\eda\sim_lib\altera_mf_components.vhd**<br><*QUARTUS ROOTDIR*>**\eda\sim_lib\altera_mf.vhd** |
| lpm | **\eda\sim_lib\220pack.vhd**<br>**\eda\sim_lib\220model.vhd** |
| sgate | **eda\sim_lib\sgate_pack.vhd**<br>**eda\sim_lib\sgate.vhd** |
| <*device name*> | **eda\sim_lib**<*device name*>**_atoms.vhd**<br>**eda\sim_lib**<*device name*>**_ components.vhd**<br>**eda\sim_lib**<*device name*>**_hssi_atoms.vhd** *(1)* |
| altera | **eda\sim_lib\altera_primitives_components.vhd**<br>**eda\sim_lib\altera_syn_attributes.vhd**<br>**eda\sim_lib\altera_primitives.vhd** |
| ALTGXB *(1)* | <*device name*>**_mf.vhd**<br><*device name*>**_mf_components.vhd** |
| <*device name*>_hssi *(1)* | <*device name*>**_hssi_components.vhd**<br><*device name*>**_hssi_atoms.vhd** |

**Table 1–1. Files to Compile—VHDL IP Functional Simulation Models (Part 2 of 2)**

| Library | File Name |
|---|---|
| auk_ddr_hp_user_lib | *<QUARTUS ROOTDIR>***libraries\vhdl\altera\altera_europa_support_lib.vhd** |
| | *<project directory>**<variation name>*_**phy_alt_mem_phy_seq_wrapper.vho** |
| | *<project directory>**<variation name>*_**auk_ddr_hp_controller_wrapper.vho** |
| | *<project directory>**<variation name>*_**phy.vho** |
| | *<project directory>**<variation name>*.**vhd** |
| | *<project directory>**<variation name>*_**example_top.vhd** |
| | *<project directory>**<variation name>*_**controller_phy.vhd** |
| | *<project directory>**<variation name>*_**phy_alt_mem_phy_pll.vhd** |
| | *<project directory>**<variation name>*_**phy_alt_mem_phy_seq.vhd** |
| | *<project directory>**<variation name>*_**example_driver.vhd** |
| | *<project directory>**<variation name>*_**ex_lfsr8.vhd** |
| | **testbench***<variation name>*_**example_top_tb.vhd** |
| | **testbench***<variation name>*_**mem_model.vhd** |

**Note for Table 1–1:**

(1) Applicable only for Arria® II GX and Stratix® IV devices.

> If you are targeting a Stratix IV device, you need both the Stratix IV and Stratix III files (**stratixiv_atoms** and **stratixiii_atoms**) to simulate in your simulator, unless you are using NativeLink.

4. Load the testbench in your simulator with the timestep set to **picoseconds**.
5. Compile the testbench file.

## Verilog HDL

For Verilog HDL simulations with IP functional simulation models, follow these steps:

1. Create a directory in the *<project directory>***testbench** directory.
2. Launch your simulation tool from this directory and create the following libraries:
   - altera_mf_ver
   - lpm_ver
   - sgate_ver
   - *<device name>*_ver
   - altera_ver
   - ALTGXB_ver
   - *<device name>*_hssi_ver
   - auk_ddr_hp_user_lib

3. Compile the files into the appropriate library as shown in Table 1–2 on page 1–8.

**Table 1–2. Files to Compile—Verilog HDL IP Functional Simulation Models**

| Library | File Name |
|---|---|
| altera_mf_ver | *<QUARTUS ROOTDIR>*\eda\sim_lib\altera_mf.v |
| lpm_ver | \eda\sim_lib\220model.v |
| sgate_ver | eda\sim_lib\sgate.v |
| *<device name>*_ver | eda\sim_lib*<device name>*_atoms.v<br>eda\sim_lib*<device name>*_hssi_atoms.v *(1)* |
| altera_ver | eda\sim_lib\altera_primitives.v |
| ALTGXB_ver *(1)* | *<device name>*_mf.v |
| *<device name>*_hssi_ver *(1)* | *<device name>*_hssi_atoms.v |
| auk_ddr_hp_user_lib | *<QUARTUS ROOTDIR>*\<br>libraries\vhdl\altera\altera_europa_support_lib.v |
| | alt_mem_phy_defines.v |
| | *<project directory>**<variation name>*_phy_alt_mem_phy_seq_wrapper.vo |
| | *<project directory>**<variation name>*_auk_ddr_hp_controller_wrapper.vo |
| | *<project directory>**<variation name>*.v |
| | *<project directory>**<variation name>*_example_top.v |
| | *<project directory>**<variation name>*_phy.v |
| | *<project directory>**<variation name>*_controller_phy.v |
| | *<project directory>**<variation name>*_phy_alt_mem_phy_pll.v |
| | *<project directory>**<variation name>*_phy_alt_mem_phy.v |
| | *<project directory>**<variation name>*_example_driver.v |
| | *<project directory>**<variation name>*_ex_lfsr8.v |
| | testbench*<variation name>*_example_top_tb.v |
| | testbench*<variation name>*_mem_model.v |

**Notes for Table 1–2:**

(1) Applicable only for Arria II GX and Stratix IV devices.

If you are targeting a Stratix IV device, you need both the Stratix IV and Stratix III files (**stratixiv_atoms** and **stratixiii_atoms**) to simulate in your simulator, unless you are using NativeLink

4. Configure your simulator to use transport delays, a timestep of **picoseconds**, and to include all the libraries in Table 1–2.
5. Compile the testbench file.

# Simulation Tips and Issues

This topic discusses simulation tips and issues.

## Tips

The ALTMEMPHY datapath is in Verilog HDL; the sequencer is in VHDL. For ALTMEMPHY designs with the Altera PHY interface (AFI), to allow the Verilog HDL simulator to simulate the design after modifying the VHDL sequencer, follow these steps:

1. On the View menu, point to **Utility Windows,** and click **TCL console**.
2. Enter the following command in the console:

```
quartus_map --read_settings_file=on --write_settings_file=off
--source=<variation_name>_phy_alt_mem_phy_seq.vhd
--source=<variation_name>_phy_alt_mem_phy_seq_wrapper.v --simgen
--simgen_parameter=CBX_HDL_LANGUAGE=verilog
<variation_name>_phy_alt_mem_phy_seq_wrapper -c
<variation_name>_phy_alt_mem_phy_seq_wrapper
```

The Quartus II software regenerates *<variation_name>***_phy_alt_mem_phy_seq_wrapper.vo** and uses this file when the simulation runs.

## DDR3 SDRAM (without Leveling) Warnings and Errors

You may see the following warning and error messages with skip calibration and quick calibration simulation:

- `WARNING: 200 us is required before RST_N goes inactive`
- `WARNING: 500 us is required after RST_N goes inactive before CKE goes active`

If these warning messages appear, change the values of the two parameters (`tinit_tck` and `tinit_rst`) in the following files to match the parameters in *<variation_name>***_phy_alt_mem_phy_seq_wrapper.v**:

- *<variation_name>***_phy_alt_mem_phy_seq_wrapper.vo** or
- *<variation_name>***_phy_alt_mem_phy_seq_wrapper.vho** files

You may see the following warning and error messages with full calibration simulation during write leveling, which you can ignore:

- `Warning: tWLS violation on DQS bit 0 positive edge. Indeterminate CK capture is possible`
- `Warning: tWLH violation on DQS bit 0 positive edge. Indeterminate CK capture is possible.`
- `ERROR: tDQSH violation on DQS bit 0`

You may see the following warning messages at time 0 (before reset) of simulation, which you can ignore:

- `Warning: There is an 'U'|'X'|'W'|'Z'|'-' in an arithmetic operand, the result will be 'X'(es).`

- `Warning: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0`

You may see the following warning and error messages during reset, which you can ignore:

- `Error : clock switches from 0/1 to X (Unknown value) on DLL instance`
- `Warning : Duty Cycle violation DLL instance Warning: Input clock duty cycle violation.`

# External Memory Interface Handbook Volume 4

## Section II. Timing Analysis

101 Innovation Drive
San Jose, CA 95134
www.altera.com

EMI_DEBUG_TIMING-2.1

Document last updated for Altera Complete Design Suite version: 10.1
Document publication date: December 2010

Subscribe

# Contents

## Chapter 1. Timing Analysis Methodology

## Chapter 2. Timing Closure



## Chapter 3. Timing Deration Methodology for Multiple Chip Select DDR2 and DDR3 SDRAM Designs

## Additional Information

# 1. Timing Analysis Methodology

Ensuring that your external memory interface meets the various timing requirements of today's high-speed memory devices can be a challenge. Altera addresses this challenge by offering external memory physical layer (PHY) interface IPs—ALTMEMPHY and UniPHY, which employ a combination of source-synchronous and self-calibrating circuits to maximize system timing margins. This PHY interface is a plug-and-play solution that the Quartus® II TimeQuest Timing Analyzer timing constrains and analyzes. The ALTMEMPHY and UniPHY IP, and the numerous device features offered by Arria® II, Cyclone® III, Cyclone IV, Stratix® III, Stratix IV, and Stratix V FPGAs, greatly simplifies the implementation of an external memory interface. All the information presented in this document for Stratix III and Stratix IV devices is applicable to HardCopy® III and HardCopy IV devices, respectively.

This chapter details the various timing paths that determine overall external memory interface performance, and describes the timing constraints and assumptions that the PHY IP uses to analyze these paths.

This chapter focuses on timing constraints for external memory interfaces based on the ALTMEMPHY and UniPHY IP. For information about timing constraints and analysis of external memory interfaces and other source-synchronous interfaces based on the ALTDQ_DQS megafunction, refer to *AN 433: Constraining and Analyzing Source Synchronous Interfaces*. and the *Quartus II TimeQuest Timing Analyzer* chapter in volume 3 of the *Quartus II Handbook*.

External memory interface timing analysis is supported only by the TimeQuest Timing Analyzer, for the following reasons:

- The wizard-generated timing constraint scripts only support the TimeQuest analyzer.
- The Classic Timing Analyzer does not offer analysis of source-synchronous outputs. For example, write data, address, and command outputs.
- The Classic Timing Analyzer does not support detailed rise and fall delay analysis.

The performance of an FPGA interface to an external memory device is dependent on the following items:

- Read datapath timing
- Write datapath timing
- Address and command path timing
- Clock to strobe timing ($t_{DQSS}$ in DDR and DDR2 SDRAM, and $t_{KHK\#H}$ in QDR II and QDRII+ SRAM)
- Read resynchronization path timing (applicable for DDR, DDR2, and DDR3 SDRAM in Arria II, Stratix III, Stratix IV, and Stratix V devices)
- Read postamble path timing (applicable for DDR, DDR2, and DDR3 SDRAM in Arria II, Stratix III, Stratix IV, and Stratix V devices)

- Write leveling path timing (applicable for DDR3 SDRAM with ALTMEMPHY DIMMs and DDR3 SDRAM with UniPHY in Stratix III, Stratix IV, and Stratix V devices)
- PHY timing paths between I/O element and core registers
- PHY and controller internal timing paths (core $f_{MAX}$ and reset recovery/removal)
- I/O toggle rate
- Output clock specifications

☞ External memory interface performance depends on various timing components, and overall system level performance is limited by performance of the slowest link (that is, the path with the smallest timing margins).

# Memory Interface Timing Components

There are several categories of memory interface timing components, including source-synchronous timing paths, calibrated timing paths, internal FPGA timing paths, and other FPGA timing parameters.

Understanding the nature of timing paths enables you to use an appropriate timing analysis methodology and constraints. The following section examines these aspects of memory interface timing paths.

## Source-Synchronous Paths

These are timing paths where clock and data signals pass from the transmitting device to the receiving device.

An example of such a path is the FPGA-to-memory write datapath. The FPGA device transmits DQ output data signals to the memory along with a center-aligned DQS output strobe signal. The memory device uses the DQS signal to clock the data on the DQ pins into its internal registers.

☞ For brevity, the remainder of this chapter refers to data signals and strobe and clock signals as DQ signals and DQS signals, respectively. While the terminology is formally correct only for DDR-type interfaces and does not match QDR II, QDR II+ and RLDRAM II pin names, the behavior is similar enough that most timing properties and concepts apply to both. The clock that captures address and command signals is always referred to as CK/CK# too.

## Calibrated Paths

These are timing paths where the clock used to capture data is dynamically positioned within the data valid window (DVW) to maximize timing margin.

For Arria II, Stratix III, and Stratix IV FPGAs interfacing with a DDR2 SDRAM controller with ALTMEMPHY IP, the resynchronization of read data from the DQS-based capture registers to the FPGA system clock domain is implemented using a self-calibrating circuit. On initialization, the sequencer block analyzes all path delays between the read capture and resynchronization registers to set up the resynchronization clock phase for optimal timing margin. Similarly for UniPHY-based controllers, the sequencer block analyzes all path delays between the read capture

registers and the read FIFO buffer to set up the FIFO write clock phase for optimal timing margin. The read postamble calibration process is implemented in a similar manner to the read resynchonization calibration. In addition, the sequencer block calibrates a read data valid signal to the delay between a controller issuing a read command and read data returning to controller.

For Stratix III, Stratix IV, and Stratix V FPGAs interfacing with a DDR3 SDRAM DIMM, the write-leveling chains and programmable output delay chain are calibrated to align the DQS edge with the CK edge at each memory device in the DIMM. Deskew circuitry is automatically enabled for ALTMEMPHY-based interfaces higher than 400 MHz.

In Cyclone III and Cyclone IV FPGAs, the ALTMEMPHY IP performs the initial data capture from the memory device using a self-calibrating circuit. The DQS strobes from the memory are not used for capture; instead, a dynamic PLL clock signal is used to capture DQ data signals into core LE registers.

## Internal FPGA Timing Paths

Other timing paths that have an impact on memory interface timing include FPGA internal $f_{MAX}$ paths for PHY and controller logic. This timing analysis is common to all FPGA designs. With appropriate timing constraints on the design (such as clock settings), the TimeQuest Timing Analyzer reports the corresponding timing margins.

For more information about the TimeQuest Timing Analyzer, refer to the *Quartus II TimeQuest Timing Analyzer* chapter in volume 3 of the *Quartus II Handbook*.

## Other FPGA Timing Parameters

Some FPGA data sheet parameters, such as I/O toggle rate and output clock specifications, can limit memory interface performance.

I/O toggle rates vary based on speed grade, loading, and I/O bank location—top/bottom versus left/right. This toggle rate is also a function of the termination used (OCT or external termination) and other settings such as drive strength and slew rate.

Ensure you check the I/O performance in the overall system performance calculation. Altera recommends that you perform signal integrity analysis for the specified drive strength and output pin load combination.

For information about signal integrity, refer to the *Board Layout Guidelines* section in volume 2 of the *External Memory Interface Handbook* and *AN 476: Impact of I/O Settings on Signal Integrity in Stratix III Devices*.

Output clock specifications include clock period jitter, half-period jitter, cycle-to-cycle jitter, and skew between FPGA clock outputs. You can obtain these specifications from the FPGA data sheet and must meet memory device requirements. You can use these specifications to determine the overall data valid window for signals transmitted between the memory and FPGA device.

## Timing Paths—ALTMEMPHY

Table 1–1 lists the timing paths for the ALTMEMPHY megafunction that are analyzed by the **Report DDR** in the TimeQuest Timing Analyzer.

**Table 1–1. Timing Paths—ALTMEMPHY (Part 1 of 5)** ***(Note 1)***

| Timing Path | Applicable ALTMEMPHY Variation(s) | Applicable Clock *(2)* | Description |
|---|---|---|---|
| Address and command | All ALTMEMPHY variations | mem_clk<br>(<*variation_name*>_ck_n_mem_clk_n[i]_ac_rise<br><*variation_name*>_ck_p_mem_clk[i]_ac_rise<br><*variation_name*>_ck_n_mem_clk_n[i]_ac_fall<br><*variation_name*>_ck_p_mem_clk[i]_ac_fall) | Setup and hold margin for all address and command pins or for mem_cke, mem_cs_n, and mem_odt pins |
| PHY | All ALTMEMPHY variations | All clocks in the PHY that drive ALTMEMPHY registers | Internal timing of the ALTMEMPHY megafunction. |
| PHY reset | All ALTMEMPHY variations | All clocks in the PHY that drive ALTMEMPHY registers | Internal timing of the asynchronous reset signals to the ALTMEMPHY megafunction. |
| DQS vs. CK | DDR2/DDR SDRAM | <*variation_name*>_ck_n_mem_clk_n[i]_tDQSS<br><*variation_name*>_ck_p_mem_clk[i]_tDQSS<br><*variation_name*>_ck_n_mem_clk_n[i]_tDSS<br><*variation_name*>_ck_p_mem_clk[i]_tDSS | Skew requirement for the arrival time of the DQS strobe at the memory with respect to the arrival time of CK/CK# at the memory. |

**Table 1–1. Timing Paths—ALTMEMPHY (Part 2 of 5)** ***(Note 1)***

| Timing Path | Applicable ALTMEMPHY Variation(s) | Applicable Clock *(2)* | Description |
|---|---|---|---|
| Half-rate address and command | DDR3/DDR2/ DDR SDRAM in half-rate mode | mem_clk (<*variation_name*>_ck_n_mem_clk_n[i]_ac_rise <*variation_name*>_ck_p_mem_clk[i]_ac_rise <*variation_name*>_ck_n_mem_clk_n[i]_ac_fall <*variation_name*>_ck_p_mem_clk[i]_ac_fall) | Setup and hold margin for the address and command pins (except for mem_cs_n, mem_cke, and mem_odt pins) with respect to the mem_clk clock at the memory when the PHY is in half-rate mode. |
| Mimic | DDR2/DDR SDRAM variation in Arria GX, Arria II GX, Cyclone III, Cyclone IV, HardCopy II, Stratix II, and Stratix II GX Devices | <*variation_name*>_ck_n_mem_clk[0] | The setup margin for the voltage and temperature tracking mechanism. |

**Table 1–1. Timing Paths—ALTMEMPHY (Part 3 of 5)** ***(Note 1)***

| Timing Path | Applicable ALTMEMPHY Variation(s) | Applicable Clock *(2)* | Description |
|---|---|---|---|
| Read capture | All ALTMEMPHY variations | dqs (<*variation_name*>_ddr_dqsin_mem_dqs) | Setup and hold margin for the DQ pins with respect to DQS strobe at the FPGA capture registers.<br>In Arria II, Cyclone IV, Stratix IV, and Stratix V devices, the margin is reported based on a combination of the TimeQuest Timing Analyzer and further steps to account for calibration that occurs at runtime. First, the TimeQuest Timing Analyzer returns the base setup and hold slacks, and then further processing adjusts the slacks to account for effects which cannot be modeled in TimeQuest. Refer to the <*variation_name*>_**report_timing_core.tcl** file, for more information about the timing analysis of calibrated memory interfaces.<br>In Stratix III devices, the margin is reported using an equation based on the applicable sampling window (SW) value for the configuration. Refer to the <*variation_name*>**_report_timing.tcl** file, for the equation or refer to the *DC and Switching Characteristics* chapter in the relevant device handbook.<br>In Cyclone III devices, it is a calibrated path. The resulting setup and hold margin, in these three device families, is always balanced as it is calculated as the total margin divided by two. Your actual setup and hold margin on the board can differ. |
| Read Postamble | DDR2/DDR SDRAM variation in Arria GX, Stratix II, and Stratix II GX Devices | Postamble clock (<*variation_name*>_ddr_postamble) | Setup and hold time margin for the postamble path that is calibrated with the resynchronization clock phase.<br>The setup and hold margin is always balanced as it is calculated as the total margin divided by two. Your actual setup and hold margin on the board can differ. |
| Read Postamble Enable | DDR2/DDR SDRAM variation in Arria GX, Stratix II, and Stratix II GX Devices | DQS clocks (<*variation_name*>_ddr_dqsin_*) | The setup and hold margin for the postamble logic that enables and disables the DQS signal going to the DQ registers. |

**Table 1–1. Timing Paths—ALTMEMPHY (Part 4 of 5)** ***(Note 1)***

| Timing Path | Applicable ALTMEMPHY Variation(s) | Applicable Clock *(2)* | Description |
|---|---|---|---|
| Read Resynchronization | DDR3/DDR2/ DDR SDRAM | Resynchronization clock (<*variation_name*>_ddr_resync) | Setup and hold margin for the DQ data with respect to resynchronization and the postamble clock at the resynchronization and the postamble registers.<br>In Arria II GX, Arria II GZ, Stratix IV, and Stratix V devices, the margin is reported using an equation based on the applicable SW value for the configuration. Refer to the <*variation_name*> **_report_timing.tcl** file, for the equation.<br>In Stratix III devices, the margin is reported via an equation based on the applicable SW value for the configuration. Refer to the <*variation_name*> **_report_timing.tcl** file, for the equation.<br>The setup and hold margin is always balanced as it is calculated as the total margin divided by two. Your actual setup and hold margin on the board can differ. |
| Write datapath | All ALTMEMPHY variations | DQS (<*variation_name*>_ddr_dqsout_ mem_dqs) | Setup and hold margin for the DQ pins with respect to DQS strobe at the memory.<br>In Arria II, Cyclone IV, Stratix IV, and Stratix V devices, the margin is reported based on a combination of the TimeQuest Timing Analyzer and further steps to account for calibration that occurs at runtime. First the TimeQuest Timing Analyzer returns the base setup and hold slacks, and then further processing adjusts the slacks to account for effects which cannot be modeled in TimeQuest. Refer to the <*variation_name*> **_report_timing_core.tcl** file, for more information regarding the timing analysis of calibrated memory interfaces.<br>In Cyclone III and Stratix III devices, the margin is reported using an equation based on the applicable channel-to-channel skew (TCCS) value for the configuration.<br>The setup and hold margin is nearly balanced. Your actual setup and hold margin on the board can differ. |
| Write leveling $t_{DQSS}$ | DDR3 SDRAM (except Arria II GX devices) | CK/CK# clocks (<*variation_name*>_ck_n_ <*CK# pin name*>_tDQSS) or (<*variation_name*>_ck_p_ <CK pin name>_tDQSS) | Skew margin for the arrival time of the DQS strobe at the memory with respect to the arrival time of CK/CK# at the memory.<br>This path is a calibrated path, such that the margin is reported via an equation. The setup and hold margin is nearly balanced. Your actual setup and hold margin on the board can differ. |

**Table 1–1. Timing Paths—ALTMEMPHY (Part 5 of 5)** ***(Note 1)***

| Timing Path | Applicable ALTMEMPHY Variation(s) | Applicable Clock *(2)* | Description |
|---|---|---|---|
| Write leveling $t_{DSS}/t_{DSH}$ | DDR3 SDRAM (except Arria II GX devices) | CK/CK# clocks<br>*(<variation_name>*`_ck_n_`<br>*<CK# pin name>*`_tDQSS`)<br>or<br>*(<variation_name>*`_ck_p_`<br>*<CK pin name>*`_tDQSS`) | Setup and hold margin for the DQS falling edge with respect to the CK clock at the memory.<br>A calibrated path, such that the margin is reported via an equation. The setup and hold margin is nearly balanced. Your actual setup and hold margin on the board can differ. |

**Note to Table 1–1:**

(1) You cannot see the details of the timing nodes for any calibrated path (such as the read postamble and read resynchronization paths) in Stratix III and Stratix IV devices. You also cannot see the details of the timing nodes for read and write paths in Cyclone III and Stratix III devices as these paths are calculated using the sampling window and channel-to-channel skew published in the *DC and Switching Characteristics* chapter in the device family handbook.

(2) `[i]` refers to the clock number—1, 2, or 3.

## Timing Paths—DDR2 and DDR3 SDRAM with UniPHY

Table 1–3 lists the timing paths for DDR2 and DDR3 controllers with UniPHY that are analyzed by the **Report DDR** in the TimeQuest Timing Analyzer.

**Table 1–2. Timing Paths—DDR2 and DDR3 SDRAM Controllers with UniPHY** ***(Note 1)***

| Timing Path | Applicable Clock *(2)* | Description |
|---|---|---|
| Address and command | <*variation_name*>_pll_addr_cmd_clk | Setup and hold margin for all address and command pins. |
| Core | <*variation_name*>_pll_afi_clk | Internal timing of the UniPHY IP. |
| Core recovery/removal | All clocks in the PHY that drive UniPHY registers | Internal timing of the asynchronous reset signals to the UniPHY IP. |
| Read capture | <*dqs_pin_name*>_IN | Setup and hold margin for the DQ pins with respect to DQS strobe at the FPGA capture registers.<br>In Arria II GZ, Stratix IV, and Stratix V devices, the TimeQuest Timing Analyzer reports the margin directly. In Stratix III devices, the margin is reported via an equation based on the applicable sampling window (SW) value for the configuration. Refer to the <*variation_name*>_**report_timing.tcl** file, for the equation or refer to the *DC and Switching Characteristics* chapter in the relevant device handbook. |
| Write datapath | <*dqs_pin_name*>_OUT | Setup and hold margin for the DQ pins with respect to DQS strobe at the memory.<br>In Arria II GZ, Stratix IV, and Stratix V devices, the TimeQuest Timing Analyzer reports the margin directly. In Stratix III devices, the margin is reported via an equation based on the applicable sampling window (SW) value for the configuration.<br>Refer to the <*variation_name*>_**report_timing.tcl** file, for the equation or refer to the *DC and Switching Characteristics* chapter in the relevant device handbook. |
| Sequencer datapath | <*variation_name*>_pll_addr_cmd_clk | Internal timing of the sequencer (Nios and Avalon interfaces). |
| Read enable | <*variation_name*>_pll_addr_cmd_clk | The setup and hold margin for the postamble logic that enables and disables the DQS signal going to the DQ registers. |

## Timing Paths—RLDRAM II, and QDR II and QDR II+ SRAM with UniPHY

Table 1–3 lists the timing paths for RLDRAM II, and QDR II and QDR II+ controllers with UniPHY that are analyzed by the **Report DDR** in the TimeQuest Timing Analyzer.

**Table 1–3. Timing Paths—RLDRAM II, and QDR II and QDR II+ SRAM Controllers with UniPHY** ***(Note 1)***

| Timing Path | Applicable Clock *(2)* | Description |
|---|---|---|
| Address and command | K/Kn for QDR II and QDR II+ SRAM<br>CK/CKn for RLDRAM II | Setup and hold margin for all address and command pins. |
| Core | All clocks in the core (controller + PHY) that drive UniPHY registers | Internal timing of the UniPHY IP. |
| Core recovery/removal | All clocks in the PHY that drive UniPHY registers | Internal timing of the asynchronous reset signals to the UniPHY IP. |
| Read capture | CQ/CQn for QDR II and QDR II+ SRAM<br>QK/QKn for RLDRAM II | Setup and hold margin for the DQ pins with respect to DQS strobe at the FPGA capture registers.<br>In Arria II, Stratix IV, and Stratix V devices, the TimeQuest Timing Analyzer reports the margin directly. In Stratix III devices, the margin is reported via an equation based on the applicable sampling window (SW) value for the configuration. Refer to the <*variation_name*> **_report_timing.tcl** file, for the equation or refer to the *DC and Switching Characteristics* chapter in the relevant device handbook. |
| Write datapath | K/Kn for QDR II and QDR II+ SRAM<br>DK/DKn for RLDRAM II | Setup and hold margin for the DQ pins with respect to DQS strobe at the memory.<br>In Arria II, Stratix IV, and Stratix V devices, the TimeQuest Timing Analyzer reports the margin directly. In Stratix III devices, the margin is reported via an equation based on the applicable transmitter channel-to-channel skew (TCCS) value for the configuration.<br>Refer to the <*variation_name*> **_report_timing.tcl** file, for the equation or refer to the *DC and Switching Characteristics* chapter in the relevant device handbook. |

# FPGA Timing Paths

This topic describes the FPGA timing paths, the timing constraints examples, and the timing assumptions that the constraint scripts use.

In Arria II, Stratix IV, and Stratix V devices, the margin for DDR2 and DDR3 SDRAM interfaces is reported based on a combination of the TimeQuest Timing Analyzer and further steps to account for calibration that occurs at runtime. First the TimeQuest analyzer returns the base setup and hold slacks, and then further processing adjusts the slacks to account for effects which cannot be modeled in TimeQuest. For DDR SDRAM, QDR II, QDR II+ SRAM, and RLDRAM II interfaces, the margin is entirely from the TimeQuest analyzer.

## Arria II Device PHY Timing Paths

Table 1–4 categorizes all Arria II devices external memory interface timing paths.

**Table 1–4. Arria II Devices External Memory Interface Timing Paths *(Note 1)***

| Timing Path | Circuit Category | Source | Destination |
|---|---|---|---|
| Read Data *(2)* | Source-Synchronous | Memory DQ, DQS Pins | DQ Capture Registers in IOE |
| Write Data *(2)* | Source-Synchronous | FPGA DQ, DQS Pins | Memory DQ, DM, and DQS Pins |
| Address and command *(2)* | Source-Synchronous | FPGA CK/CK# and Addr/Cmd Pins | Memory Input Pins |
| Clock-to-Strobe *(2)* | Source-Synchronous | FPGA CK/CK# and DQS Output Pins | Memory Input Pins |
| Read Resynchronization *(2)*, *(3)* | Calibrated | IOE Capture Registers | IOE Resynchronization Registers |
| Read Resynchronization *(2)*, *(7)* | Calibrated | IOE Capture Registers | Read FIFO in FPGA Core |
| Read Postamble *(6)* | Calibrated | IOE Postamble-Control Alignment Registers | IOE Postamble Registers |
| PHY IOE-Core Paths *(2)*, *(3)* | Source-Synchronous | IOE Resynchronization Registers | FIFO in FPGA Core |
| PHY and Controller Internal Paths *(2)* | Internal Clock $f_{MAX}$ | Core Registers | Core Registers |
| I/O Toggle Rate *(4)* | I/O | FPGA Output Pin | Memory Input Pins |
| Output Clock Specifications (Jitter, DCD) *(5)* | I/O | FPGA Output Pin | Memory Input Pins |

**Notes to Table 1–4:**

(1) Timing paths applicable for an interface between Arria II devices and SDRAM component.
(2) Timing margins for this path are reported by the TimeQuest Timing Analyzer Report DDR function.
(3) Only for ALTMEMPHY megafunctions.
(4) Altera recommends that you perform signal integrity simulations to verify I/O toggle rate.
(5) For output clock specifications, refer to the *Arria II Device Data Sheet* chapter of the *Arria II Handbook*.
(6) Only for DDR, DDR2, and DDR3 SDRAM controllers with ALTMEMPHY using the High-Performance Controller II (HPC II) architecture.
(7) Only for UniPHY IP..

Figure 1–1 shows the Arria II GX devices input datapath registers and circuit types.

UniPHY IP interfaces bypass the synchronization registers.

**Figure 1–1. Arria II GX Devices Input Data Path Registers and Circuit Types in SDRAM Interface**

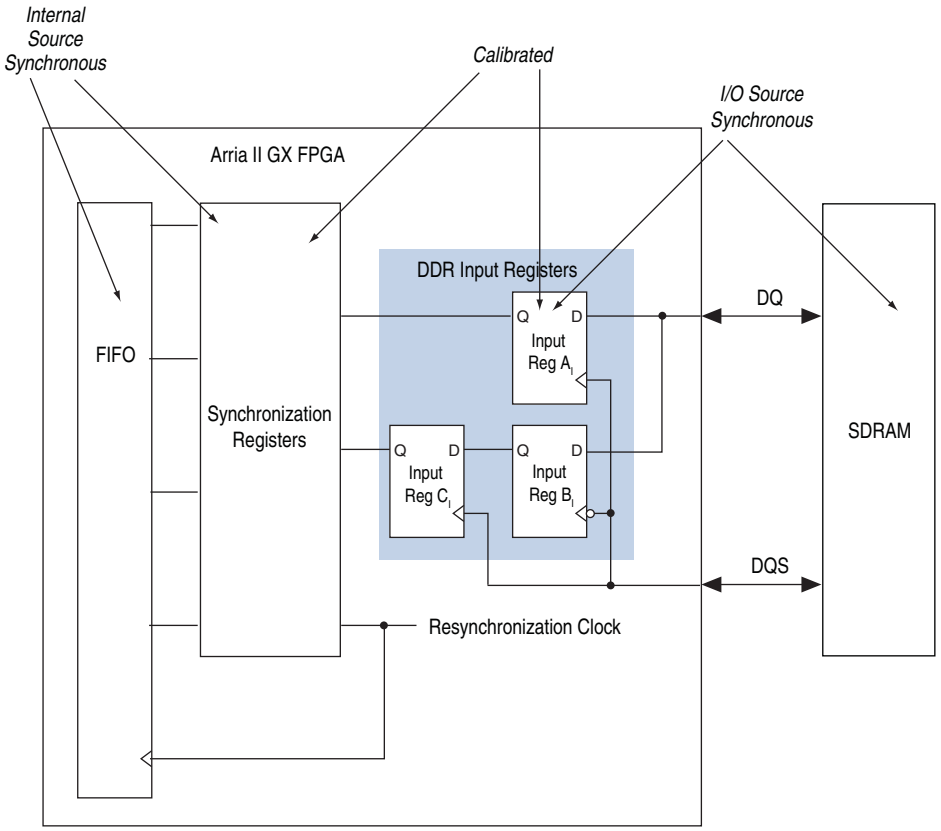
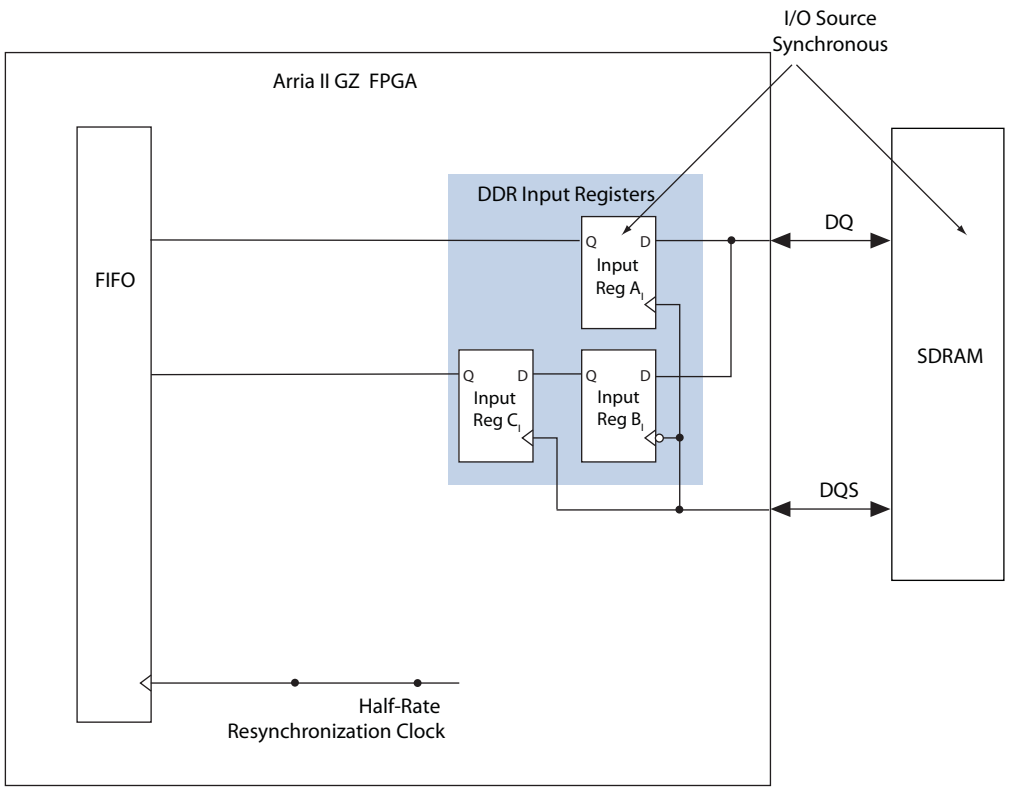


Figure 1–2 shows the Arria II GZ devices input datapath registers and circuit types.

**Figure 1–2. Arria II GZ Devices Input Data Path Registers and Circuit Types in SDRAM Interface**




## Stratix IV PHY Timing Paths

A closer look at all the register transfers occurring in the Stratix IV input datapath reveals many source-synchronous and calibrated circuits. Figure 1–3 shows a block diagram of this input path with some of these paths identified. The output datapath contains a similar set of circuits.

☞ UniPHY IP interfaces bypass the alignment and synchronization registers.

**Figure 1–3. Stratix IV Input Path Registers and Circuit Types in SDRAM Interface**

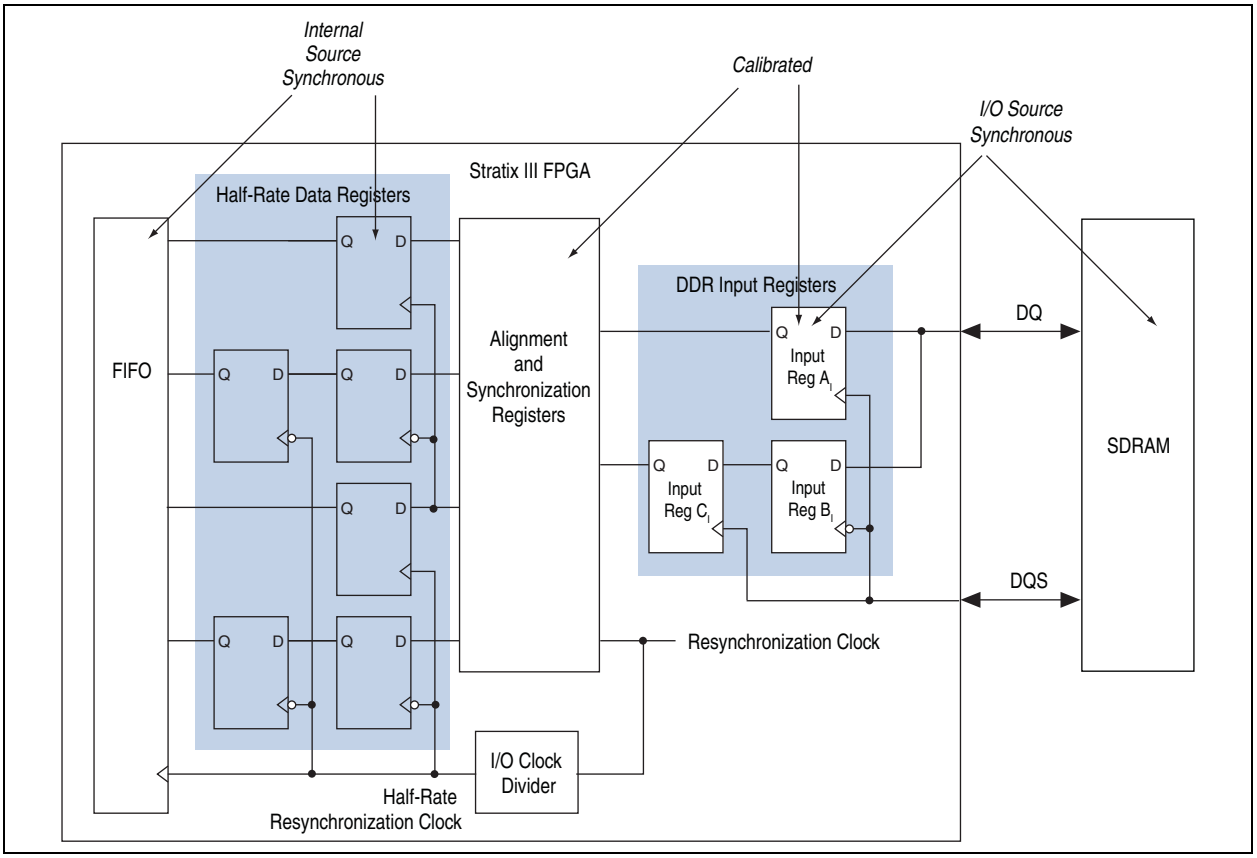



Table 1–5 categorizes all Stratix IV external memory interface timing paths.

**Table 1–5. Stratix IV External Memory Interface Timing Paths *(Note 1)* (Part 1 of 2)**

| Timing Path | Circuit Category | Source | Destination |
|---|---|---|---|
| Read Data *(2)* | Source-Synchronous | Memory DQ, DQS Pins | DQ Capture Registers in IOE |
| Write Data *(2)* | Source-Synchronous | FPGA DQ, DQS Pins | Memory DQ, DM, and DQS Pins |
| Address and command *(2)* | Source-Synchronous | FPGA CK/CK# and Addr/Cmd Pins | Memory Input Pins |
| Clock-to-Strobe *(2)* | Source-Synchronous | FPGA CK/CK# and DQS Output Pins | Memory Input Pins |
| Read Resynchronization *(2)*, *(3)* | Calibrated | IOE Capture Registers | IOE Alignment and Resynchronization Registers |
| Read Postamble *(3)* | Calibrated | IOE Postamble Alignment Registers | IOE Postamble Registers |
| PHY IOE-Core Paths *(2)*, *(3)* | Source-Synchronous | IOE Half Data Rate Registers and Half-Rate Resynchronization Clock | FIFO in FPGA Core |
| PHY & Controller Internal Paths *(2)* | Internal Clock $f_{MAX}$ | Core registers | Core registers |

**Table 1–5. Stratix IV External Memory Interface Timing Paths *(Note 1)* (Part 2 of 2)**

| Timing Path | Circuit Category | Source | Destination |
|---|---|---|---|
| I/O Toggle Rate *(4)* | I/O – Data sheet | FPGA Output Pin | Memory Input Pins |
| Output Clock Specifications (Jitter, DCD) *(5)* | I/O – Data sheet | FPGA Output Pin | Memory Input Pins |

**Notes to Table 1–5:**

(1) Table 1–5 lists the timing paths applicable for an interface between Stratix IV devices and half-rate SDRAM components.
(2) Timing margins for this path are reported by the TimeQuest Timing Analyzer Report DDR function.
(3) Only for ALTMEMPHY megafunctions.
(4) Altera recommends that you perform signal integrity simulations to verify I/O toggle rate.
(5) For output clock specifications, refer to the *DC and Switching Characteristics* chapter of the *Stratix IV Device Handbook*..

## Stratix V Timing paths

Figure 1–4 shows a block diagram of the Stratix V input data path.

**Figure 1–4. Stratix V Input Data Path**

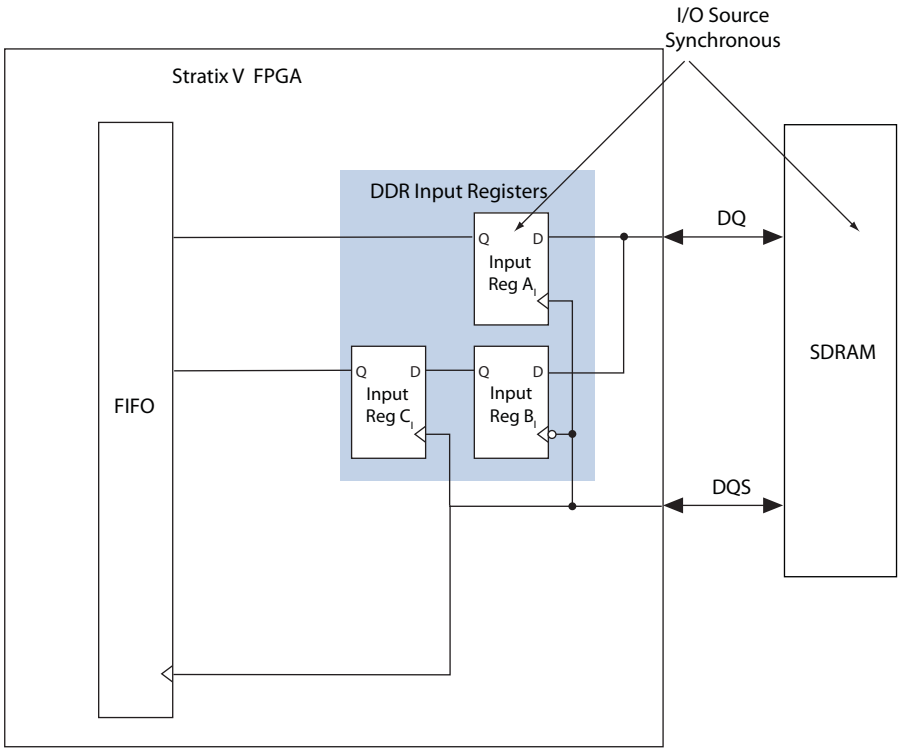



Table 1–6 categorizes all Stratix V devices external memory interface timing paths.

**Table 1–6. Stratix V External Memory Interface Timing Paths *(Note 1)* *(Part 1 of 2)***

| Timing Path | Circuit Category | Source | Destination |
|---|---|---|---|
| Read Data *(2)* | Source-Synchronous | Memory DQ, DQS Pins | DQ Capture Registers in IOE |
| Write Data *(2)* | Source-Synchronous | FPGA DQ, DM, DQS Pins | Memory DQ, DM, and DQS Pins |

**Table 1–6. Stratix V External Memory Interface Timing Paths *(Note 1)* *(Part 2 of 2)***

| Timing Path | Circuit Category | Source | Destination |
|---|---|---|---|
| Address and command *(2)* | Source-Synchronous | FPGA CK/CK# and Addr/Cmd Pins | Memory Input Pins |
| Clock-to-Strobe *(2)* | Source-Synchronous | FPGA CK/CK# and DQS Output Pins | Memory Input Pins |
| Read Resynchronization *(2)* | Source-Synchronous | IOE Capture Registers | Read FIFO in FPGA core |
| PHY & Controller Internal Paths *(2)* | Internal Clock $f_{MAX}$ | Core Registers | Core Registers |
| i/O Toggle Rate *(3)* | I/O – Data sheet | FPGA Output Pin | Memory Input Pins |
| Output Clock Specifications (Jitter, DCD) *(4)* | I/O – Data sheet | FPGA Output Pin | Memory Input Pins |

**Notes to Table 1–6:**

(1) This table lists the timing paths applicable for an interface between Stratix V devices and half-rate SDRAM components.
(2) Timing margins for this path are reported by the TimeQuest Timing Analyzer Report DDR function.
(3) Altera recommends that you perform signal integrity simulations to verify I/O toggle rate.
(4) For output clock specifications, refer to the *DC and Switching Characteristics* chapter of the Stratix V Device Handbook.

## Cyclone III and Cyclone IV PHY Timing Paths

Table 1–7 categorizes the various timing paths in a Cyclone III and Cyclone IV memory interface. Cyclone III and Cyclone IV devices use a calibrated PLL output clock for data capture and ignore the DQS strobe from the memory. Therefore, read resynchronization and postamble timing paths do not apply to Cyclone III and Cyclone IV designs. The read capture is implemented in LE registers specially placed next to the data pin with fixed routing, and data is transferred from the capture clock domain to the system clock domain using a FIFO block. Figure 1–5 shows the Cyclone III and Cyclone IV input datapath registers and circuit types.

**Table 1–7. Cyclone III and Cyclone IV SDRAM External Memory Interface Timing Paths *(Note 1)* (Part 1 of 2)**

| Timing Path | Circuit Category | Source | Destination |
|---|---|---|---|
| Read Data *(2)* | Calibrated | Memory DQ, DQS Pins | FPGA DQ Capture Registers in LEs |
| Write Data *(2)* | Source-Synchronous | FPGA DQ, DQS Pins | Memory DQ, DM, and DQS Pins |
| Address and command *(2)* | Source-Synchronous | FPGA CK/CK# and Addr/Cmd Pins | Memory Input Pins |
| Clock-to-Strobe *(2)* | Source-Synchronous | FPGA CK/CK# and DQS Output Pins | Memory Input Pins |
| PHY Internal Timing *(2)* | Internal Clock $f_{MAX}$ | LE Half Data Rate Registers | FIFO in FPGA Core |
| I/O Toggle Rate *(3)* | I/O – Data sheet *I/O Timing* section | FPGA Output Pin | Memory Input Pins |

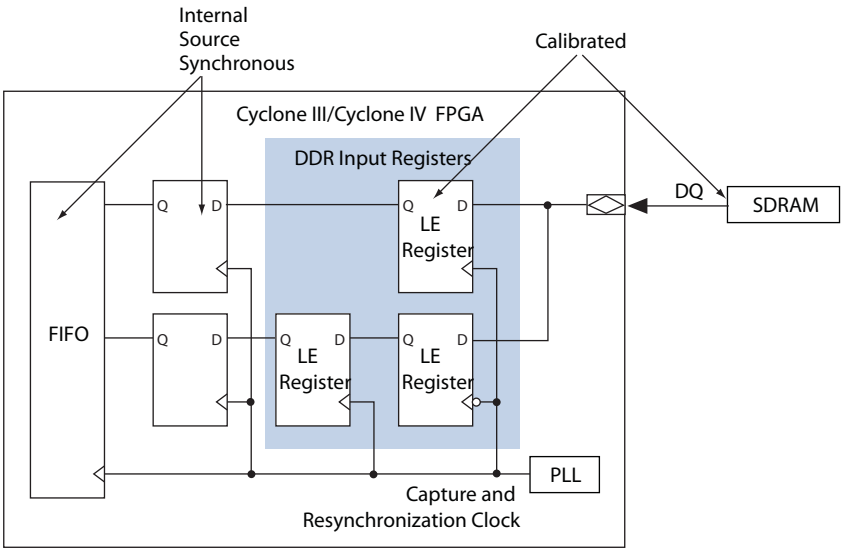
**Table 1–7. Cyclone III and Cyclone IV SDRAM External Memory Interface Timing Paths *(Note 1)* (Part 2 of 2)**

| Timing Path | Circuit Category | Source | Destination |
|---|---|---|---|
| Output Clock Specifications (Jitter, DCD) *(4)* | I/O – Data sheet *Switching Characteristics* section | FPGA Output Pin | Memory Input Pins |

**Notes to Table 1–7:**

(1) Table 1–7 lists the timing paths applicable for an interface between Cyclone III and Cyclone IV devices and SDRAM.
(2) Timing margins for this path are reported by the TimeQuest Timing Analyzer Report DDR function.
(3) Altera recommends that you perform signal integrity simulations to verify I/O toggle rate.
(4) For output clock specifications, refer to the *DC and Switching Characteristics* chapter of the *Cyclone III Device Handbook* and of the *Cyclone IV Device Handbook*

**Figure 1–5. Cyclone III or Cyclone IV Input Data Path Registers and Circuit Types in SDRAM Interface**




# Timing Margin Components

This section details the timing margins, such as the read data and write data timing paths, which the TimeQuest Timing Analyzer calaulates for Cyclone III and Stratix III designs. Timing paths internal to the FPGA are either guaranteed by design and tested on silicon, or analyzed by the TimeQuest Timing Analyzer using corresponding timing constraints.

For design guidelines about implementing and analyzing your external memory interface using the PHY in Cyclone III, Stratix III, and Stratix IV devices, refer to *Volume 6: Design Flow Tutorials* of the *External Memory Interface Handbook*.

Timing margins for chip-to-chip data transfers can be defined as:

Margin = bit period – transmitter uncertainties – receiver requirements

where:

- Sum of all transmitter uncertainties = transmitter channel-to-channel skew (TCCS).

  The timing difference between the fastest and slowest output edges on data signals, including $t_{CO}$ variation, clock skew, and jitter. The clock is included in the TCCS measurement and serves as the time reference.

- Sum of all receiver requirements = receiver sampling window (SW) requirement.

  The period of time during which the data must be valid to capture it correctly. The setup and hold times determine the ideal strobe position within the sampling window.

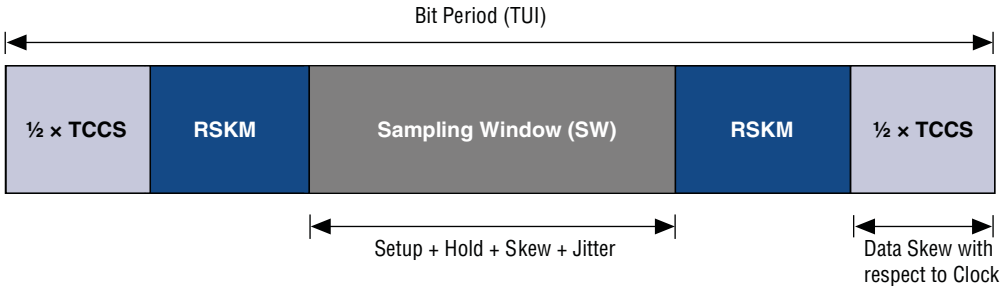
- Receiver skew margin (RSKM) = margin or slack at the receiver capture register.

In Arria II, Cyclone IV, and Stratix IV devices, the margin is reported based on a combination of the TimeQuest Timing Analyzer calculation results and further processing steps that account for the calibration that occurs at runtime. First, the TimeQuest analyzer returns the base setup and hold slacks, and further processing steps adjust the slacks to account for effects which the TimeQuest analyzer cannot model.

Refer to the *DC and Switching Characteristics* chapter of the *Cyclone III Device Handbook* or *Stratix III Device Handbook* for TCCS and SW specifications.

Figure 1–6 relates this terminology to a timing budget diagram.

**Figure 1–6. Sample Timing Budget Diagram**




The timing budget regions marked "½ × TCCS" represent the latest data valid time and earliest data invalid times for the data transmitter. The region marked sampling window is the time required by the receiver during which data must stay stable. This sampling window comprises the following:

- Internal register setup and hold requirements
- Skew on the data and clock nets within the receiver device
- Jitter and uncertainty on the internal capture clock

The sampling window is not the capture margin or slack, but instead the requirement from the receiver. The margin available is denoted as RSKM.

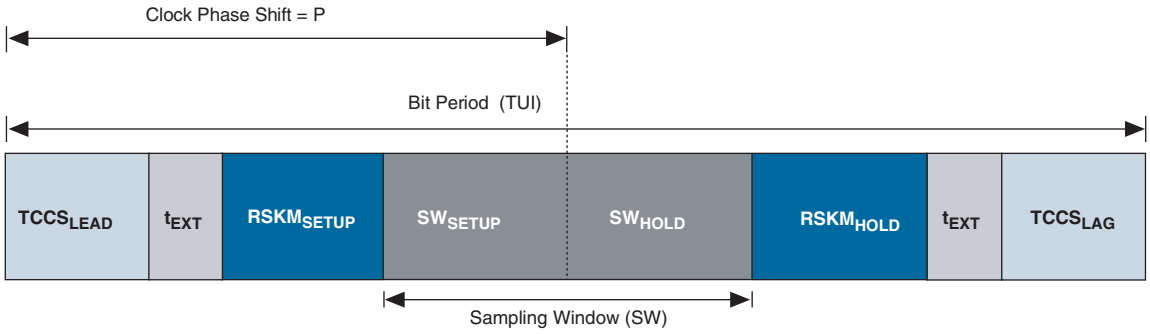
The simple example illustrated in Figure 1–6 does not consider any board level uncertainties, assumes a center-aligned capture clock at the middle of the receiver sampling window region, and assumes an evenly distributed TCCS with respect to the transmitter clock pin. In this example, the left end of the bit period corresponds to time t = 0, and the right end of the bit period corresponds to time t = TUI (where TUI stands for time unit interval). Therefore, the center-aligned capture clock at the receiver is best placed at time t = TUI/2.

Therefore:

the total margin = 2 × RSKM = TUI – TCCS – SW.

Consider the case where the clock is not center-aligned within the bit period (clock phase shift = P), and the transmitter uncertainties are unbalanced ($TCCS_{LEAD} \neq$ and $TCCS_{LAG}$). $TCCS_{LEAD}$ is defined as the skew between the clock signal and latest data valid signal. $TCCS_{LAG}$ is defined as the skew between the clock signal and earliest data invalid signal. Also, the board level skew across data and clock traces are specified as $t_{EXT}$. For this condition, you should compute independent setup and hold margins at the receiver ($RSKM_{SETUP}$ and $RSKM_{HOLD}$). In this example, the sampling window requirement is split into a setup side requirement ($SW_{SETUP}$) and hold side ($SW_{HOLD}$) requirement. Figure 1–7 illustrates the timing budget for this condition. A timing budget similar to that shown in Figure 1–7 is used for Cyclone III and Stratix III FPGA read and write data timing paths.

**Figure 1–7. Sample Timing Budget with Unbalanced (TCCS and SW) Timing Parameters**




Therefore:

Setup margin = $RSKM_{SETUP} = P - TCCS_{LEAD} - SW_{SETUP} - t_{EXT}$

Hold margin = $RSKM_{HOLD} = (TUI - P) - TCCS_{LAG} - SW_{HOLD} - t_{EXT}$

The timing budget illustrated in Figure 1–6 with balanced timing parameters applies for calibrated paths where the clock is dynamically center-aligned within the data valid window. The timing budget illustrated in Figure 1–7 with unbalanced timing parameters applies for circuits that employ a static phase shift using a DLL or PLL to place the clock within the data valid window.

## Read Data Timing

Memory devices provide edge-aligned DQ and DQS outputs to the FPGA during read operations. Stratix III FPGAs center-aligns the DQS strobe using static DLL-based delays, and the Cyclone III FPGAs use a calibrated PLL clock output to capture the read data in LE registers without using DQS. While Stratix III devices use a source synchronous circuit for data capture and Cyclone III devices use a calibrated circuit, the timing analysis methodology is quite similar, as shown in the following section.

When applying this methodology to read data timing, the memory device is the transmitter and the FPGA device is the receiver.

The transmitter channel-to-channel skew on outputs from the memory device is available from the corresponding device data sheet. Let us examine the TCCS parameters for a DDR2 SDRAM component.

For DQS-based capture:

- The time between DQS strobe and latest data valid is defined as $t_{DQSQ}$
- The time between earliest data invalid and next strobe is defined as $t_{QHS}$
- Based on earlier definitions, $TCCS_{LEAD} = t_{DQSQ}$ and $T_{CCSLAG} = t_{QHS}$

The sampling window at the receiver, the FPGA, includes several timing parameters:

- Capture register micro setup and micro hold time requirements
- DQS clock uncertainties because of DLL phase shift error and phase jitter
- Clock skew across the DQS bus feeding DQ capture registers
- Data skew on DQ paths from pin to input register including package skew

For TCCS and SW specifications, refer to the *DC and Switching Characteristics* chapter of the *Cyclone III Device Handbook* or the *Stratix III Device Handbook*.

Figure 1–8 shows the timing budget for a read data timing path.

**Figure 1–8. Timing Budget for Read Data Timing Path**

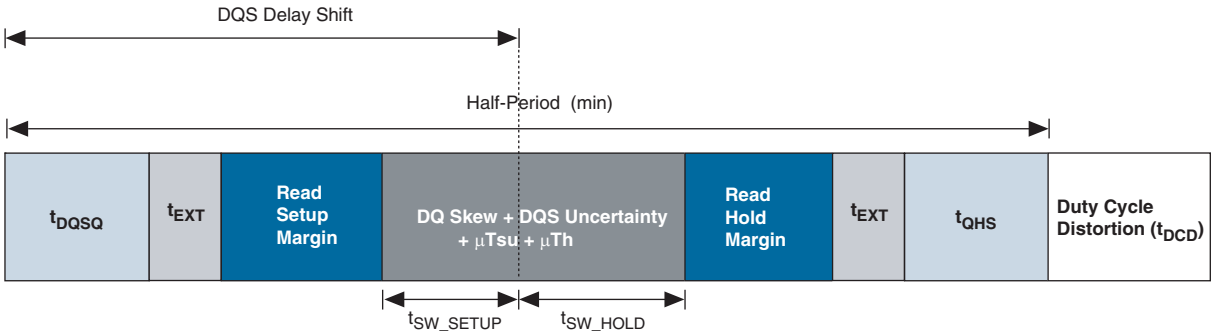


Table 1–8 details a read data timing analysis for a Stratix III –2 speed-grade device interfacing with a 400-MHz DDR2 SDRAM component.

**Table 1–8. Read Data Timing Analysis for Stratix III Device with a 400-MHz DDR2 SDRAM *(Note 1)***

| Parameter | Specifications | Value (ps) | Description |
|---|---|---|---|
| Memory Specifications *(1)* | $t_{HP}$ | 1250 | Average half period as specified by the memory data sheet, $t_{HP}$ = 1/2 * $t_{CK}$ |
| | $t_{DCD}$ | 50 | Duty cycle distortion = 2% × $t_{CK}$ = 0.02 × 2500 ps |
| | $t_{DQSQ}$ | 200 | Skew between DQS and DQ from memory |
| | $t_{QHS}$ | 300 | Data hold skew factor as specified by memory |
| FPGA Specifications | $t_{SW\_SETUP}$ | 181 | FPGA sampling window specifications for a given configuration (DLL mode, width, location, and so on.) |
| | $t_{SW\_HOLD}$ | 306 | |
| Board Specifications | $t_{EXT}$ | 20 | Maximum board trace variation allowed between any two signal traces (user specified parameter) |
| Timing Calculations | $t_{DVW}$ | 710 | $t_{HP} - t_{DCD} - t_{DQSQ} - t_{QHS} - 2 \times t_{EXT}$ |
| | $t_{DQS\_PHASE\_DELAY}$ | 500 | Ideal phase shift delay on DQS capture strobe = (DLL phase resolution × number of delay stages × $t_{CK}$) / 360° = (36° × 2 stages × 2500 ps)/360° = 500 ps |
| Results | Setup margin | 99 | $RSKM_{SETUP} = t_{DQSQ\_PHASE\_DELAY} - t_{DQSQ} - t_{SW\_SETUP} - t_{EXT}$ |
| | Hold margin | 74 | $RSKM_{HOLD} = t_{HP} - t_{DCD} - t_{DQS\_PHASE\_DELAY} - t_{QHS} - t_{SW\_HOLD} - t_{EXT}$ |

**Notes to Table 1–8:**

(1) This sample calculation uses memory timing parameters from a 72-bit wide 256-MB micron MT9HTF3272AY-80E 400-MHz DDR2 SDRAM DIMM.

Table 1–9 details a read data timing analysis for a DDR2 SDRAM component at 200 MHz using the SSTL-18 Class I I/O standard and termination. A 267-MHz DDR2 SDRAM component is required to ensure positive timing margins for the 200-MHz memory interface clock frequency for the 200 MHz operation.

**Table 1–9. Read Data Timing Analysis for a 200-MHz DDR2 SDRAM on a Cyclone III Device *(Note 1)***

| Parameter | Specifications | Value (ps) | Description |
|---|---|---|---|
| Memory Specifications *(1)* | $t_{HP}$ | 2500 | Average half period as specified by the memory data sheet |
| | $t_{DCD\_TOTAL}$ | 250 | Duty cycle distortion = 2% × tCK = 0.02 × 5000 ps |
| | $t_{AC}$ | ± 500 | Data (DQ) output access time for a 267-MHz DDR2 SDRAM component |
| FPGA Specifications | $t_{SW\_SETUP}$ | 580 | FPGA sampling window specification for a given configuration (interface width, location, and so on). |
| | $t_{SW\_HOLD}$ | 550 | |
| Board Specifications *(1)* | $t_{EXT}$ | 20 | Maximum board trace variation allowed between any two signal traces (user specified parameter) |
| Timing Calculations | $t_{DVW}$ | 1230 | $t_{HP}$ - $t_{DCD}$ - 2 × $t_{AC}$ – 2 × $t_{EXT}$ |
| Results | Total margin | 100 | $t_{DVW}$ - $t_{SW\_SETUP}$ - $t_{SW\_HOLD}$ |

**Notes to Table 1–9:**

(1) For this sample calculation, total duty cycle distortion and board skew are split over both setup and hold margin. For more information on Cyclone III –6 speed-grade device read capture and timing analysis, refer to "Cyclone III and Cyclone IV PHY Timing Paths" on page 1–16.

## Write Data Timing

During write operations, the FPGA generates a DQS strobe and a center-aligned DQ data bus using multiple PLL-driven clock outputs. The memory device receives these signals and captures them internally. The Stratix III family contains dedicated DDIO (double data rate I/O) blocks inside the IOEs.

For write operations, the FPGA device is the transmitter and the memory device is the receiver. The memory device's data sheet specifies data setup and data hold time requirements based on the input slew rate on the DQ/DQS pins. These requirements make up the memory sampling window, and include all timing uncertainties internal to the memory.

Output skew across the DQ and DQS output pins on the FPGA make up the TCCS specification. TCCS includes contributions from numerous internal FPGA circuits, including:

- Location of the DQ and DQS output pins
- Width of the DQ group
- PLL clock uncertainties, including phase jitter between different output taps used to center-align DQS with respect to DQ
- Clock skew across the DQ output pins, and between DQ and DQS output pins
- Package skew on DQ and DQS output pins

Refer to the *DC and Switching Characteristics* chapter of the the *Cyclone III Device Handbook* or the *Stratix III Device Handbook* for TCCS and SW specifications.

Figure 1–9 illustrates the timing budget for a write data timing path.

**Figure 1–9. Timing Budget for Write Data Timing Path**

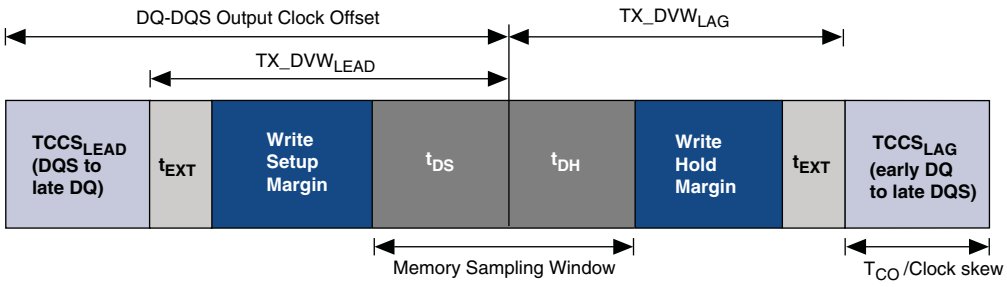


Table 1–10 details a write data timing analysis for a Stratix III –2 speed-grade device interfacing with a DDR2 SDRAM component at 400 MHz. This timing analysis assumes the use of a differential DQS strobe with 2.0-V/ns edge rates on DQS, and 1.0-V/ns edge rate on DQ output pins. Consult your memory device's data sheet for derated setup and hold requirements based on the DQ/DQS output edge rates from your FPGA.

**Table 1–10. Write Data Timing Analysis for 400-MHz DDR2 SDRAM Stratix III Device *(Note 1)***

| Parameter | Specifications | Value (ps) | Description |
|---|---|---|---|
| Memory Specifications *(1)* | $t_{HP}$ | 1250 | Average half period as specified by the memory data sheet |
| | $t_{DSA}$ | 250 | Memory setup requirement (derated for DQ/DQS edge rates and $V_{REF}$ reference voltage) |
| | $t_{DHA}$ | 250 | Memory hold requirement (derated for DQ/DQS edge rates and $V_{REF}$ reference voltage) |
| FPGA Specifications | $TCCS_{LEAD}$ | 229 | FPGA transmitter channel-to-channel skew for a given configuration (PLL setting, location, and width). |
| | $TCCS_{LAG}$ | 246 | |
| Board Specifications | $t_{EXT}$ | 20 | Maximum board trace variation allowed between any two signal traces (user specified parameter) |
| Timing Calculations | $t_{OUTPUT\_CLOCK\_OFFSET}$ | 625 | Output clock phase offset between DQ & DQS output clocks = 90°. $t_{OUTPUT\_CLOCK\_OFFSET}$ = (output clock phase DQ and DQS offset x $t_{CK}$)/360° = (90° x 2500)/360° = 625 |
| | $TX\_DVW_{LEAD}$ | 396 | Transmitter data valid window = $t_{OUTPUT\_CLOCK\_OFFSET} - TCCS_{LEAD}$ |
| | $TX\_DVW_{LAG}$ | 379 | Transmitter data valid window = tHP - $t_{OUTPUT\_CLOCK\_OFFSET} - TCCS_{LAG}$ |
| Results | Setup margin | 126 | $TX\_DVW_{LEAD} - t_{EXT} - t_{DSA}$ |
| | Hold margin | 109 | $TX\_DVW_{LAG} - t_{EXT} - t_{DHA}$ |

**Notes to Table 1–10:**

(1) This sample calculation uses memory timing parameters from a 72-bit wide 256-MB micron MT9HTF3272AY-80E 400-MHz DDR2 SDRAM DIMM

Table 1–11 details a write timing analysis for a Cyclone III –6 speed-grade device interfacing with a DDR2 SDRAM component at 200 MHz. A 267-MHz DDR2 SDRAM component is used for this analysis.

**Table 1–11. Write Data Timing Analysis for a 200-MHz DDR2 SDRAM Interface on a Cyclone III Device *(Note 1)* (Part 1 of 2)**

| Parameter | Specifications | Value (ps) | Description |
|---|---|---|---|
| Memory Specifications | $t_{HP}$ | 2500 | Average half period as specified by the memory data sheet |
| | $t_{DCD\_TOTAL}$ | 250 | Total duty cycle distortion = 5% × $t_{CK}$ = 0.05 x 5000 |
| | $t_{DS\ (derated)}$ | 395 | Memory setup requirement from a 267-MHz DDR2 SDRAM component (derated for single-ended DQS and 1 V/ns slew rate) |
| | $t_{DH\ (derated)}$ | 335 | Memory hold from DDR2 267-MHz component (derated for single-ended DQS and 1 V/ns slew rate) |
| FPGA Specifications | $TCCS_{LEAD}$ | 790 | FPGA TCCS for a given configuration (PLL setting, location, width) |
| | $TCCS_{LAG}$ | 380 | |

**Table 1–11. Write Data Timing Analysis for a 200-MHz DDR2 SDRAM Interface on a Cyclone III Device *(Note 1)* (Part 2 of 2)**

| Parameter | Specifications | Value (ps) | Description |
|---|---|---|---|
| Board Specifications | $t_{EXT}$ | 20 | Maximum board trace variation allowed between any two signal traces (user specified parameter) |
| Timing Calculations | $TX\_DVW_{LEAD}$ | 460 | Transmitter data valid window = $t_{OUTPUT\_CLOCK\_OFFSET} - TCCS_{LEAD}$ |
| | $TX\_DVW_{LAG}$ | 870 | Transmitter data valid window = $t_{HP}$ - $t_{OUTPUT\_CLOCK\_OFFSET} - TCCS_{LAG}$ |
| | $t_{OUTPUT\_CLOCK\_OFFSET}$ | 1250 | Output clock phase offset between DQ/DQS output clocks = 90°<br>$t_{OUTPUT\_CLOCK\_OFFSET}$ = (output clock phase DQ & DQS offset x $t_{CK}$)/360° = (90° x 5000)/360° = 1250 |
| Results | Setup margin | 45 | $TX\_DVW_{LEAD} - t_{EXT} - t_{DS}$ |
| | Hold margin | 265 | $TX\_DVW_{LAG} - t_{EXT} - t_{DH} - t_{DCD\_TOTAL}$ |

**Note to Table 1–11:**

(1) For more information on Cyclone III –6 speed-grade device read capture and timing analysis, refer to "Read Data Timing" on page 1–20.

# Dynamic Deskew and DQ-DQS Offset Timing

This section briefly describes the dynamic deskew and DQ-DQS offset timing calculations applied for Stratix III devices. The DQ-DQS offset scheme is applicable for a Stratix III device and ALTMEMPHY cores interfacing with DDR, DDR2 and DDR3 SDRAM components for frequencies of 400 MHz and below. For frequencies above 400 MHz, a dynamic deskew scheme improves the timing margins for a Stratix III device interfacing with a DDR3 SDRAM component. The UniPHY IP implements dynamic deskew for DDR2 and DDR3 interfaces at all frequencies.

This section does not apply to Cyclone III or Cyclone IV devices.

## DQ-DQS Offset

At frequencies of 400 MHz and below, the ALTMEMPHY sequencer uses the delay chains in the IOE to shift the offset between DQ and DQS such that data setup and hold requirement are met at the memory device. The actual offsets employed are FPGA family, speed grade, and frequency dependent. The DQ-DQS offset applies only to the write data timing path.

The `mem_clk` and `mem_clk_n` pins may have a D6 Delay assignment set to 2 DDR3 SDRAM with leveling interfaces depending on the configuration of the DQ clocks generated by the write levelling delay chain.

Table 1–12 details a write timing analysis for a Stratix III –2 speed-grade device interfacing with a 400-MHz DDR3 SDRAM component.

**Table 1–12. Write Data Timing Analysis for a 400-MHz DDR3 SDRAM Stratix III Device *(Note 1)* (Part 1 of 2)**

| Parameter | Specifications | Value (ps) | Description |
|---|---|---|---|
| Memory Specifications | $t_{HP}$ | 1250 | Average half period as specified by the memory data sheet |
| | $t_{DS}$ | 250 | Memory setup requirement |
| | $t_{DH}$ | 250 | Memory hold requirement |

**Table 1–12. Write Data Timing Analysis for a 400-MHz DDR3 SDRAM Stratix III Device *(Note 1)* (Part 2 of 2)**

| Parameter | Specifications | Value (ps) | Description |
|---|---|---|---|
| FPGA Specifications | $TCCS_{LEAD}$ | 293 | FPGA TCCS for C2 speed grade, 8-tap DLL above 375 MHz |
| | $TCCS_{LAG}$ | 284 | |
| Board Specifications | $t_{EXT}$ | 20 | Maximum board trace variation |
| Timing Calculations | $t_{OUTPUT\_CLOCK\_OFFSET}$ | 625 | Output clock phase offset between DQ and DQS (90°)<br>$t_{OUTPUT\_CLOCK\_OFFSET}$ = (output clock phase DQ & DQS offset x $t_{CK}$)/360° = (90° x 2500)/360° = 625 |
| | $TX\_DVW_{LEAD}$ | 332 | Transmitter data valid window = $t_{OUTPUT\_CLOCK\_OFFSET}$ - $TCCS_{LEAD}$ |
| | $TX\_DVW_{LAG}$ | 341 | Transmitter data valid window = $t_{HP}$ - $t_{OUTPUT\_CLOCK\_OFFSET}$ - $TCCS_{LAG}$ |
| Results | Setup margin | 62 | Margin = $TX\_DVW_{LEAD}$ - $t_{EXT}$ - $t_{DS}$ |
| | Hold margin | 71 | Margin = $TX\_DVW_{LAG}$ - $t_{EXT}$ - $t_{DH}$ |

**Note to Table 1–12:**

(1) At a frequency of 400 MHz, the DLL is configured for 8-tap mode, resulting in a 90° output clock phase shift offset between the DQ and DQS pins.

Table 1–13 details a write timing analysis for a Stratix III –3 speed-grade device interfacing with a 333-MHz DDR3 SDRAM component.

**Table 1–13. Write Data Timing Analysis for a 333-MHz DDR3 SDRAM Stratix III Device *(Note 1)***

| Parameter | Specifications | Value (ps) | Description |
|---|---|---|---|
| Memory Specifications | $t_{HP}$ | 1500 | Average half period as specified by the memory data sheet |
| | $t_{DS}$ | 250 | Memory setup requirement |
| | $t_{DH}$ | 250 | Memory hold requirement |
| FPGA Specifications | $TCCS_{LEAD}$ | 217 | FPGA TCCS for C3 speed grade, 10-tap DLL |
| | $TCCS_{LAG}$ | 496 | |
| Board Specifications | $t_{EXT}$ | 20 | Maximum board trace variation |
| Timing Calculations | $t_{OUTPUT\_CLOCK\_OFFSET}$ | 600 | Output clock phase offset between DQ and DQS (72°)<br>$t_{OUTPUT\_CLOCK\_OFFSET}$ = (output clock phase DQ & DQS offset x $t_{CK}$)/360° = (72° x 3000)/360° =600 |
| | $TX\_DVW_{LEAD}$ | 383 | Transmitter data valid window = $t_{OUTPUT\_CLOCK\_OFFSET}$ - $TCCS_{LEAD}$ |
| | $TX\_DVW_{LAG}$ | 404 | Transmitter data valid window = $t_{HP}$ - $t_{OUTPUT\_CLOCK\_OFFSET}$ - $TCCS_{LAG}$ |
| Results | Setup margin | 113 | Margin = $TX\_DVW_{LEAD}$ - $t_{EXT}$ - $t_{DS}$ |
| | Hold margin | 134 | Margin = $TX\_DVW_{LAG}$ - $t_{EXT}$ - $t_{DH}$ |

**Note to Table 1–13:**

(1) At a frequency of 333 MHz, the DLL operates in 10-tap mode, resulting in a 72° output clock phase shift offset between the DQ and DQS pins.

## Dynamic Deskew

For Stratix III devices interfacing with a DDR3 SDRAM component at frequencies above 400 MHz, the timing margins are too small to be fixed by the static DQ-DQS offset scheme. A dynamic scheme improves setup and hold margin at the memory device with the DLL set to 8-tap mode. The DDR3 SDRAM sequencer uses the configurable delay elements and delay-chains in the IOE to make an observation of the write window as seen in the DDR3 SDRAM components. This information is then used to configure the delays for each individual DQ and DM pin, to meet the memory device setup and hold requirements and reduce skew between DQ and DM pins in a DQS group.

For more information on deskew feature, refer to *Calibration Techniques for High-Bandwidth Source-Synchronous Interfaces.*

## DDR3 Deskew Effects on Timing Analysis Methodology

This section briefly describes the DDR3 SDRAM deskew effects on the timing analysis methodology for external memory interfaces. The TCCS and SW specifications are updated for a Stratix III –2 speed-grade device interfacing with a DDR3 SDRAM component at a frequency range from 401 MHz to 533 MHz.

For more information on the TCCS and SW specifications, refer to the *DC and Switching Characteristics* chapter of the *Stratix III Device Handbook*.

With deskew circuitry and 8-tap phase offset, the board skew specifications are handled differently. When the board skew does not exceed the lead or lag thresholds, it does not affect timing performance. In contrast, the board skew can degrade timing margins if it exceeds the lead and the lag thresholds. Table 1–14 and Table 1–15 detail the impacts by illustrating the timing analysis of different board skew settings for a Stratix III –2 speed-grade device interfacing with a 533-MHz DDR3 SDRAM component.

**Table 1–14. Write Data Timing Analysis for a 533-MHz DDR3 SDRAM Stratix III Device (Part 1 of 2)**

| Parameter | Specifications | Description | Value Based on 25ps Board Trace Skew (ps) |
|---|---|---|---|
| Memory Specifications *(1)* | $t_{HP}$ | Average half period as specified by the memory data sheet | 938 |
| | $t_{DS}$ | Memory setup requirement | 200 |
| | $t_{DH}$ | Memory hold requirement | 200 |
| FPGA Specifications | $TCCS_{LEAD}$ *(2)* | FPGA transmitter channel-channel skew for dynamic deskew | 253 |
| | $TCCS_{LAG}$ *(2)* | | 262 |

**Table 1–14. Write Data Timing Analysis for a 533-MHz DDR3 SDRAM Stratix III Device (Part 2 of 2)**

| Parameter | Specifications | Description | Value Based on 25ps Board Trace Skew (ps) |
|---|---|---|---|
| Timing Calculations | $t_{OUTPUT\_CLOCK\_OFFSET}$ | Output clock phase offset between DQ and DQS (90°) | 469 |
| | $t_{EXT\_DELTA}$ | Amount (if any) by which board trace skew exceeds 20 ps FPGA skew threshold | 5 |
| | $TX\_DVW_{LEAD}$ | Transmitter data valid window = $t_{OUTPUT\_CLOCK\_OFFSET}$ - $TCCS_{LEAD}$ | 216 |
| | $TX\_DVW_{LAG}$ | Transmitter data valid window = $t_{HP}$ - $t_{OUTPUT\_CLOCK\_OFFSET}$ - $TCCS_{LAG}$ | 207 |
| Results | Setup margin | Margin = $TX\_DVW_{LEAD}$ - $t_{EXT\_DELTA}$ - $t_{DS}$ | 11 |
| | Hold margin | Margin = $TX\_DVW_{LAG}$ - $t_{EXT\_DELTA}$ - $t_{DH}$ | 2 |

**Notes to Table 1–14:**

(1) This sample calculation uses memory timing parameters from a 72-bit wide 1152-MB micron MT9JSF12872AY- 1G1BZES 533-MHz DDR3 SDRAM DIMM.

(2) The TCCS specifications used in this sample calculation are obtained from the Stratix III data sheet for a –2 speed-grade device and the SSTL-18 I/O standard.

**Table 1–15. Read Data Timing Analysis for a 533-MHz DDR3 SDRAM Stratix III Device**

| Parameter | Specifications | Description | Value Based on 25ps Board Trace Skew (ps) |
|---|---|---|---|
| Memory Specifications *(1)* | $t_{DQSQ}$ | Skew between DQS and DQ from memory | 150 |
| | $t_{QH}$ | DQ output hold time = 0.38*$t_{CK}$ | 712 |
| FPGA Specifications | $t_{SW\_SETUP}$ *(2)* | FPGA sampling window specifications for dynamic read deskew | 295 |
| | $t_{SW\_HOLD}$ *(2)* | | 171 |
| Timing Calculations | $t_{DQS\_PS}$ | DQS phase shift setting (nominal) | 469 |
| | $t_{EXT\_DELTA}$ | Amount (if any) by which board trace skew exceeds 20 ps FPGA skew threshold | 5 |
| Results | Setup margin | Margin = $t_{DQS\_PS}$ - $t_{DQSQ}$ - $t_{SW\_SETUP}$ - $t_{EXT\_DELTA}$ | 19 |
| | Hold margin | Margin = $t_{QH}$ - $t_{DQS\_PS}$ - $t_{SW\_HOLD}$ - $t_{EXT\_DELTA}$ | 67 |

**Notes to Table 1–15:**

(1) This sample calculation uses memory timing parameters from a 72-bit wide 1152-MB micron MT9JSF12872AY- 1G1BZES 533-MHz DDR3 SDRAM DIMM.

(2) The sampling window specifications used in this sample calculation are obtained from the Stratix III data sheet for a –2 speed-grade device and the SSTL-18 I/O standard.

# Timing Constraint and Report Files

The timing contraints differ for the ALTMEMPHY megafunction and the UniPHY IP.

## ALTMEMPHY Megafunction

To ensure a successful external memory interface operation, the ALTMEMPHY MegaWizard Plug-In Manager generates the following files for timing constraints and reporting scripts:

- <*variation_name*>**_ddr_timing.sdc**
- <*variation_name*>**_ddr_timing.tcl**
- <*variation_name*>**_report_timing.tcl**
- <*variation_name*>**_report_timing_core.tcl**
- <*variation_name*>**_ddr_pins.tcl**

### <*variation_name*>_ddr_timing.sdc

The Synopsys design constraint (**.sdc**) file has the name <*controller_variation_name*>**_phy_ddr_timing.sdc** when you instantiate the ALTMEMPHY megafunction in the Altera memory controller, and has the name <*phy_variation_name*>**_ddr_timing.sdc** when you instantiate the ALTMEMPHY megafunction as a stand-alone design. You must add this **.sdc** to your Quartus II project to allow the Quartus II Fitter to use the timing driven compilation to optimize the timing margins. To add your **.sdc**, perform the following steps:

1. On the Assignments menu, click **Settings**.
2. In the **Settings** dialog box, under **Timing Analysis Settings**, select **TimeQuest Timing Analyzer**.
3. Add the **.sdc**.

To analyze the timing margins for all ALTMEMPHY megafunction timing paths, execute the Report DDR function in the TimeQuest Timing Analyzer; refer to the "Timing Analysis Description" on page 1–31. No timing constraints are necessary (or specified in the **.sdc**) for Arria II GX, Arria II GZ, Stratix III, Stratix IV, and Stratix V devices read capture and write datapaths, because all DQ and DQS pins are pre-defined. The capture and output registers are built into the IOE, and the signals are using dedicated routing connections. Timing constraints have no impact on the read and write timing margins. However, the timing margins for these paths are analyzed using FPGA data sheet specifications and the user-specified memory data sheet parameters.

The ALTMEMPHY megafunction uses the following **.sdc** constraints for internal FPGA timing paths, address and command paths, and clock-to-strobe timing paths:

- Creating clocks on PLL inputs
- Creating generated clocks using `derive_pll_clocks`, which includes all full-rate and half-rate PLL outputs, PLL reconfiguration clock, and I/O scan clocks
- Calling `derive_clock_uncertainty`
- Cutting timing paths for DDR I/O, calibrated paths, and most reset paths

- Setting output delays on address and command outputs (versus CK/CK# outputs)
- Setting `2T` or two clock-period multicycle setup for all half-rate address and command outputs, except nCS and on-die termination (ODT) (versus CK/CK# outputs)
- Setting output delays on DQS strobe outputs (versus CK/CK# outputs for DDR2 and DDR SDRAM)

The high-performance controller MegaWizard Plug-In Manager generates an extra *<variation_name>*_**example_top.sdc** file for the example driver design. This file contains the timing constraints for the non-DDR specific parts of the project.

### *<variation_name>*_ddr_timing.tcl

This script includes the memory interface and FPGA device timing parameters for your variation. It is included within *<variation_name>*_**report_timing.tcl** and *<variation_name>*_**ddr_timing.sdc** and runs automatically during compilation. This script is run for every instance of the same variation.

### *<variation_name>*_report_timing.tcl

This script reports the timing slacks for your variation. It runs automatically during compilation. You can also run this script with the Report DDR task in the TimeQuest Timing Analyzer window. This script is run for every instance of the same variation.

### *<variation_name>*_report_timing_core.tcl

This script contains high-level procedures that *<variation_name>*_**report_timing.tcl** uses to compute the timing slacks for your variation. It runs automatically during compilation.

### *<variation_name>*_ddr_pins.tcl

This script includes all the functions and procedures required by the *<variation_name>*_**report_timing.tcl** and *<variation_name>*_**ddr_timing.sdc** scripts. It is a library of useful functions to include at the top of an **.sdc** file. It finds all the variation instances in the design and the associated clock, register, and pin names of each instances. The results are saved in the same directory as the **.sdc** and *<variation_name>*_**report_timing.tcl** as *<variation_name>*_**autodetectedpins.tcl**.

Because this **.tcl** file traverses the design for the project pin names, you do not need to keep the same port names on the top level of the design.

## UniPHY IP

To ensure a successful external memory interface operation, the wizards for the controllers with UniPHY generate the following files for timing constraints and reporting scripts:

- *<variation_name>*.**sdc**
- *<variation_name>*_**timing.tcl**
- *<variation_name>*_**report_timing.tcl**

- <*variation_name*>**_report_timing_core.tcl**
- <*variation_name*>**_pin_map.tcl**
- <*variation_name*>**_parameters.tcl**

### <*variation_name*>.sdc

The **.sdc** file <*variation_name*>**.sdc** is listed in the wizard-generated Quartus II IP (**.qip)** file. Including this file in the project allows the Quartus II Synthesis and Fitter to use the timing driven compilation to optimize the timing margins.

To analyze the timing margins for all UniPHY timing paths, execute the Report DDR function in the TimeQuest Timing Analyzer.

The UniPHY IP uses the **.sdc** file to constrain internal FPGA timing paths, address and command paths, and clock-to-strobe timing paths, and more specifically:

- Creating clocks on PLL inputs
- Creating generated clocks
- Calling `derive_clock_uncertainty`
- Cutting timing paths for specific reset paths
- Setting input and output delays on DQ inputs and outputs
- Setting output delays on address and command outputs (versus CK/CK# outputs)

### <*variation_name*>_timing.tcl

This script includes the memory interface and FPGA device timing parameters for your variation. It is included within <*variation_name*>**_report_timing.tcl** and <*variation_name*>**.sdc**. Do not change this file.

### <*variation_name*>_report_timing.tcl

This script reports the timing slack for your variation. It runs automatically during compilation (during static timing analysis). You can also run this script with the Report DDR task in the TimeQuest Timing Analyzer. This script is run for every instance of the same variation.

### <*variation_name*>_report_timing_core.tcl

This script contains high-level procedures that the <*variation_name*>**_report_timing_core.tcl** script uses to compute the timing slack for your variation. This script runs automatically during compilation.

### <*variation_name*>_pin_map.tcl

This script is a library of functions and procedures that the <*variation_name*>**_report_timing.tcl** and <*variation_name*>**.sdc** scripts use. The <*variation_name*>**_pin_assignments.tcl** script, which is not relevant to timing constraints, also uses this library.

### *<variation_name>*_parameters.tcl

This script defines some of the parameters that describe the geometry of the core and the PLL configuration. Do not change this file, except when you modify the PLL through the MegaWizard Plug-In Manager. In this case, the changes to the PLL parameters do not automatically propagate to this file and you must manually apply those changes in this file.

# Timing Analysis Description

The following sections describe the timing analysis using the respective FPGA data sheet specifications and the user-specified memory data sheet parameters.

The timing slacks obtained from the TimeQuest Timing Analyzer include all the effects included with the Quartus II timing model such as die-to-die and within-die variations, aging, systematic skew, and operating condition variations. The TimeQuest analyzer reports do not include the effects of runtime calibration that occurs on some of the memory interface PHYs.

To account for the effects of calibration, the ALTMEMPHY and UniPHY IP include additional scripts that are part of the <*phy_variation_name*>**_report_timing.tcl** and <*phy_variation_name*>_ **report_timing_core.tcl** files that determine the timing margin after calibration. These scripts use the setup and hold slacks of individual pins to emulate what is occurring during calibration to obtain timing margins that are representative of calibrated PHYs. The effects considered as part of the calibrated timing analysis include improvements in margin because of calibration, and quantization error and calibration uncertainty because of voltage and temperature changes after calibration.

## Address and Command

Address and command signals are single data rate signals latched by the memory device using the FPGA output clock. Some of the address and command signals are half-rate data signals, while others, such as the chip select, are full-rate signals. The TimeQuest Timing Analyzer analyzes the address and command timing paths using the `set_output_delay (max and min)` constraints.

### Arria II GX, Cyclone III, Cyclone IV, Stratix III, and Stratix IV Devices

For a 533-MHz DDR3 SDRAM component with Stratix IV memory interface, Example 1–1 and Example 1–2 ensure optimum address and command timing margin. These equations also apply for Arria II GX, Cyclone III, Cyclone IV, and Stratix III devices.

**Example 1–1. Optimum Address and Command Timing**

```
set_output_delay -add_delay -clock $ckclock -max [round_3dp [expr {$off*$t(period) + $t(IS) +
$t(board_skew) + $fpga_tCK_ADDR_CTRL_SETUP_ERROR + $::ISI(addresscmd_setup) +
$t(additional_addresscmd_tpd)}]] [concat $pins(addrcmd) $pins(addrcmd_2t)]
```

**Example 1–2. Optimum Address and Command Timing**

```
set_output_delay -add_delay -clock $ckclock -min [round_3dp [expr {$off*$t(period) -
$t(IH) - $t(board_skew) - $fpga_tCK_ADDR_CTRL_HOLD_ERROR - $::ISI(addresscmd_hold) +
$t(additional_addresscmd_tpd)}]] [concat $pins(addrcmd) $pins(addrcmd_2t)]
```

Table 1–16 describes the terms for Arria II GX, Cyclone III, Cyclone IV, Stratix III, and Stratix IV address and command timing constraints in <*phy_variation_name*>_**ddr_timing.sdc**:

**Table 1–16. Description of Address and Command Timing Constraints**

| Term | Description |
|---|---|
| $off*$t(period) | For a Stratix III or Stratix IV device interfacing with a 533-MHz DDR3 SDRAM component, this offset is set to zero as the address and command interface is driven from a dedicated clock phase. |
| $t(IS) | The setup time for address and command pins at the memory side. |
| $t(IH) | The hold time for the address and command pins at the memory side. |
| $t(board_skew) | The worst case difference in propagation delay between the clock pins and any address and command pins at the memory side. |
| $ISI(addresscmd_setup) | The intersymbol interference that reduces the address and command eye opening on the left side of the window. For Arria II GX, Cyclone IV, Stratix III and Stratix IV devices. |
| $ISI(addresscmd_hold) | The intersymbol interference that reduces the address and command eye opening on the right side of the window. For Arria II GX, Cyclone IV, Stratix III and Stratix IV devices. |
| $fpga_tCK_ADDR_CTRL_SETUP_ERROR | An uncertainty factor for HardCopy II designs. This factor is unnecessary for the Stratix III or Stratix IV family devices, and is set to zero. |
| $t(additional_addresscmd_tpd) | An extra offset parameter that can be set by modifying the assignment at the top of the **.sdc** file, allowing you to compensate for a longer address and command bus. If all the address and command traces are 0.500 ns longer than the clock traces, set this value to 0.500. This parameter is negative if the clock traces are longer than the address and command traces. |

## Read Capture

Read capture timing analysis indicates the amount of slack on the DDR DQ signals that are latched by the FPGA using the DQS strobe output of the memory device. The read capture timing paths are analyzed by a combination of the TimeQuest Timing Analyzer using the `set_input_delay (max and min)`, `set_max_delay`, and `set_min_delay` constraints, and further steps to account for calibration that occurs at runtime. The ALTMEMPHY and UniPHY IP include timing constraints in the <*phy_variation_name*>**_ddr_timing.sdc** file (ALTMEMPHY ) or <*phy_variation_name*>**.sdc** file (UniPHY), and further slack analysis in <*phy_variation_name*>**_report_timing.tcl** and <*phy_variation_name*>**_report_timing_core.tcl** files.

## Arria II GX, Stratix III, and Stratix IV Devices

For a 533-MHz DDR3 SDRAM component with Stratix IV memory interface, Example 1–3 and Example 1–4 ensure optimum read capture timing.

**Example 1–3. Read Capture Timing**

```
set input_max_delay   [round_3dp [expr -$::t(DQS_PSERR_min) -
     $DQSpathjitter*$DQSpathjitter_setup_prop +
     $::t(min_additional_dqs_variation) - $fpga_tREAD_CAPTURE_SETUP_ERROR]]
set input_max_delay2  [round_3dp [expr $::t(DQSQ) + $::t(board_skew) +
     $::SSN(rel_pushout_i)]]

set_max_delay -from [lindex $dqsgroup 2] -to * $input_max_delay
set_input_delay -add_delay -clock $dqs_in_clockname -max $input_max_delay2
     [lindex $dqsgroup 2]
```

**Example 1–4. Read Capture Timing**

```
set input_min_delay   [round_3dp [expr  $::t(DQS_PSERR_max) +
    $DQSpathjitter*(1.0- $DQSpathjitter_setup_prop) +
    $::t(max_additional_dqs_variation) +   $fpga_tREAD_CAPTURE_HOLD_ERROR -
    $::t(QH) + $tJITper]]
set input_min_delay2  [round_3dp [expr - $::t(board_skew) -
    $::SSN(rel_pullin_i)]]

set_min_delay -from [lindex $dqsgroup 2] -to * $input_min_delay
set_input_delay -add_delay -clock $dqs_in_clockname -min $input_min_delay2
     [lindex $dqsgroup 2]
```

Table 1–17 describes the terms for Arria II GX, Stratix III, and Stratix IV read capture timing equations in *<phy_variation_name>*_**ddr_timing.sdc**:

**Table 1–17. Description of Arria II GX and Stratix IV Read Capture Timing Equation (Part 1 of 2)**

| Term | Description |
|---|---|
| $t(DQS_PSERR) | The phase shift error in the DQS delay chain from what is specified. |
| $t(DQSQ) | Memory uncertainty: DQS to DQ skew, per group. |
| $t(QH) (for DDR3 SDRAM only) | The memory output timing. |
| $period * 0.5 - tQHS (for DDR/DDR2 SDRAM) | |
| $t(board_skew) | The worst case difference in propagation delay between the clock pins and any address and command pins at the memory side. |
| $DQSpathjitter | The peak-to-peak jitter on the read capture paths that reduces the available timing margin. |
| $DQSpathjitter_setup_prop | The proportion of the jitter on the read capture paths that affects the setup margin. |
| $fpga_tREAD_CAPTURE_SETUP_ERROR | |
| $fpga_tREAD_CAPTURE_HOLD_ERROR | Uncertainty factors for HardCopy designs. These factors are unnecessary for the Stratix IV or Arria II GX devices, and are set to zero. |

**Table 1–17. Description of Arria II GX and Stratix IV Read Capture Timing Equation (Part 2 of 2)**

| Term | Description |
|---|---|
| $SSN(rel_pullin_i) | The timing pull-in because of simultaneous switching noise (SSN) on input pins caused by multiple DQ/DQS pins switching at the same time. |
| $SSN(rel_pushout_i) | The timing push-out because of SSN on input pins caused by multiple DQ/DQS pins switching at the same time. |

## Cyclone III Devices

Cyclone III devices read data is captured using a PLL phase that is calibrated and tracked with the sequencer. The ALTMEMPHY megafunction automatically calibrates the phase of a dedicated PLL read clock to center it in the read data valid window presented to all DQ capture registers. As DQS read strobes are ignored, read postamble path is not available for Cyclone III devices. The phase of the read clock is adjusted to account for voltage and temperature variations seen in the mimic path. Also, read resynchronization path does not apply as the ALTMEMPHY clock domain is established using a dual-clock FIFO buffer.

For a DDR2 SDRAM component with Cyclone III memory interface, Example 1–5 ensures optimum read capture timing margin:

**Example 1–5. Optimum Read Capture Timing Margin**

```
set su   [round_3dp [expr {0.25*$period - 0.5*$tDCD_total - $tAC - [lindex $tsw 0]/1000.0 -
0.5 * $board_skew}]]

set hold [round_3dp [expr {0.25*$period - 0.5*$tDCD_total - $tAC - [lindex $tsw 1]/1000.0 - 0.5 *
$board_skew}]]
```

Table 1–18 describes the terms for a Cyclone III device read capture timing equation in *<phy_variation_name>*_**report_timing.tcl**:

**Table 1–18. Description of Cyclone III Device Read Capture Timing Equation**

| Term | Description |
|---|---|
| 0.25*$period | Minimum setup/hold margin. |
| $tDCD_total | Total duty cycle distortion is split over both setup and hold margin. |
| $tAC | Data (DQ) output access time for a DDR2 SDRAM component. |
| $board_skew | The worst case difference in propagation delay between all DQ pins. It is split over both setup and hold margin. |
| [lindex $tsw 0] | Read sampling window ($t_{SW\_SETUP}$). |
| [lindex $tsw 1] | Read sampling window ($t_{SW\_HOLD}$). |

## Cyclone IV Devices

Cyclone IV devices read data is captured using a PLL phase that is calibrated and tracked with the sequencer.

Example 1–6 shows the equations from the <*phy_variation_name*>**_report_timing_core.tcl** that ensures optimum read capture timing margin.

**Example 1–6. Optimum Read Capture Timing Margin**

```
# Ideal setup and hold slacks is half of the minimum high time
    set setup_slack [expr 0.5*$t(HP)]
    set hold_slack  [expr 0.5*$t(HP)]

    # Remove the variation in the DQ signals coming back from the memory in reference to the clock
    set tAC $t(AC)
    set setup_slack [expr $setup_slack - $tAC]
    set hold_slack  [expr $hold_slack  - $tAC]

    # Remove the variation in the clock and DQ signals inside the FPGA
    set setup_slack [expr $setup_slack - $minmax_DQ_variation/2.0 - $minmax_clock_variation/2.0]
    set hold_slack  [expr $hold_slack  - $minmax_DQ_variation/2.0 - $minmax_clock_variation/2.0]

    # Remove the uncertaininty in the path tracking
    set pll_step_size [expr $::period / $::pll_steps]
    set phy_uncertainity_steps [get_micro_node_delay -micro
MIMIC_PATH_UNCERTAINTY -parameters [list IO VPAD]]
    set phy_uncertainty [expr $pll_step_size * $phy_uncertainity_steps]
    set setup_slack [expr $setup_slack - $phy_uncertainty]
    set hold_slack  [expr $hold_slack  - $phy_uncertainty]

    # Remove the worst-case quantization uncertainty
    set setup_slack [expr $setup_slack - $pll_step_size/2.0]
    set hold_slack  [expr $hold_slack - $pll_step_size/2.0]

    # Remove SSN effects
    set setup_slack [expr $setup_slack - $::SSN(pushout_i)]
    set hold_slack  [expr $hold_slack  - $::SSN(pullin_i)]

    # Remove the board skew
    set setup_slack [expr $setup_slack - $::board(intra_DQS_group_skew)/2]
    set hold_slack  [expr $hold_slack  - $::board(intra_DQS_group_skew)/2]
```

## Write

Write timing analysis indicates the amount of slack on the DDR DQ signals that are latched by the memory device using the DQS strobe output from the FPGA device. The write timing paths are analyzed by a combination of the TimeQuest Timing Analyzer using the `set_output_delay (max and min)` and further steps to account for calibration that occurs at runtime. The ALTMEMPHY and UniPHY IP include timing constraints in the <*phy_variation_name*>**_ddr_timing.sdc** file (ALTMEMPHY ) or <*phy_variation_name*>**.sdc** file (UniPHY), and further slack analysis in <*phy_variation_name*>**_report_timing.tcl** and <*phy_variation_name*>**_report_timing_core.tcl** files.

## Arria II GX, Cyclone IV, Stratix III, Stratix IV, and Stratix V Devices

For a 533-MHz DDR3 SDRAM component with Stratix IV memory interface, Example 1–7 and Example 1–8 ensure optimum write timing margin:

**Example 1–7. Optimum Write Timing Margin**

```
set output_max_delay  [round_3dp [expr   $::t(board_skew) + $::t(DS) +
$outputDQSpathjitter*$outputDQSpathjitter_setup_prop + $WR_DQS_DQ_SETUP_ERROR + $::ISI(DQ)/2 +
$::ISI(DQS)/2 + $::SSN(rel_pushout_o)]]
```

**Example 1–8. Optimum Write Timing Margin**

```
set output_min_delay  [round_3dp [expr  -$::t(board_skew) - $::t(DH) -
$outputDQSpathjitter*(1.0-$outputDQSpathjitter_setup_prop) - $WR_DQS_DQ_HOLD_ERROR  - $::ISI(DQ)/2 -
$::ISI(DQS)/2 - $::SSN(rel_pullin_o)]]
```

Table 1–19 describes the terms for an Arria II GX, Cyclone IV, Stratix III, or a Stratix IV timing equation in <*phy_variation_name*>**_ddr_timing.sdc**:

**Table 1–19. Description of an Arria II GX or Stratix IV Write Timing Equation**

| Term | Description |
|---|---|
| $t(DQS_PSERR) | The phase shift error in the DQS delay chain from what is specified. |
| $tDS/$tDH | Memory uncertainty for setup and hold requirement. |
| $t(board_skew) | The worst case difference in propagation delay between the clock pins and any address and command pins at the memory side. |
| $t(WL_DCD) | The duty-cycle-distortion on DQS caused by going through the write leveling delay chains. |
| $t(WL_JITTER) | Jitter caused by DQ and DQS going through the write leveling delay chains. |
| $t(WL_PSE) | The phase shift error in the write leveling delay chains from what is specified. |
| $output_DQSpathjitter | The peak-to-peak jitter on the write paths that reduces the available timing margin. |
| $output_DQSpathjitter_setup_prop | The proportion of the jitter on the write paths that affects the setup margin. |
| $WR_DQS_DQ_SETUP_ERROR | Uncertainty factors for HardCopy designs. These factors are unnecessary for the Stratix IV or Arria II GX family devices, and are set to zero. |
| $WR_DQS_DQ_HOLD_ERROR | |
| $SSN(rel_pullin_o) | The timing pull-in because of SSN on output pins caused by multiple DQ/DQS pins switching at the same time. |
| $SSN(rel_pushout_o) | The timing push-out because of SSN on output pins caused by multiple DQ/DQS pins switching at the same time. |
| $ISI(DQ) | The intersymbol interference that reduces the total write eye opening. |
| $ISI(DQS) | The intersymbol interference that increases the variation in DQS arrival times. |

## Cyclone III Devices

For a DDR2 SDRAM component with Cyclone III memory interface, Example 1–9 and Example 1–10 ensure optimum write timing margin:

**Example 1–9. Optimum Write Timing Margin**

```
set su  [round_3dp [expr {$period*$dq2dqs_output_phase_offset/360.0 - $write_board_skew - [lindex $tccs
0]/1000.0 - $tDS}]]
```

**Example 1–10. Optimum Write Timing Margin**

```
set hold [round_3dp [expr {$period*(0.5 -
$dq2dqs_output_phase_offset/360.0) - $tDCD_total -
$write_board_skew - [lindex $tccs 1]/1000.0 - $tDH}]]
```

Table 1–20 describes the terms for a Cyclone III device write timing equation in *<phy_variation_name>*_**report_timing.tcl**:

**Table 1–20. Description of Cyclone III Device Write Timing Equation**

| Term | Description |
|---|---|
| $period * $dq2dqs_output_phase_offset / 360 | The DQ write clock and DQS write clock are offset by 90° , assuming that this offset is the optimum timing margin. |
| $$t_{DCD\_total}$$ | Total duty cycle distortion. |
| $board_skew | The worst case difference in propagation delay between the DQS strobe and any DQ pin in the same DQ/DQS group. |
| $tDS/$tDH | Memory uncertainty for setup/hold requirement. |
| [lindex $tccs 0] | Write channel to channel skew ($$TCCS_{LEAD}$$). |
| [lindex $tccs 1] | Write channel to channel skew ($$TCCS_{LAG}$$). |

## Stratix III Devices

For a DDR3 SDRAM component with Stratix III memory interface, Example 1–11 and Example 1–12 ensure optimum write timing margin:

**Example 1–11. Optimum Write Timing Margin**

```
set su [round_3dp [expr {$period*$dq2dqs_output_phase_offset/360.0 - $write_board_skew - [lindex $tccs
0]/1000.0 - $tDS}]]
```

**Example 1–12. Optimum Write Timing Margin**

```
set hold[round_3dp [expr {$period*(0.5 - $dq2dqs_output_phase_offset/360.0) - $write_board_skew -
[lindex $tccs 1]/1000.0 - $tDH}]]
```

Table 1–21 describes the terms for a Stratix III timing equation in <*phy_variation_name*>**_report_timing.tcl**:

**Table 1–21. Description of Stratix III Write Timing Equation**

| Term | Description |
|---|---|
| $period * $dq2dqs_output_phase_offset / 360 | The DQ write clock and DQS write clock are offset by 90°, assuming this offset provides the optimum timing margin. |
| $write_board_skew | The worst case difference in propagation delay between the DQS strobe and any DQ pin in the same DQ/DQS group during write operation. |
| $tDS/$tDH | Memory uncertainty for setup/hold requirement. |
| [lindex $tccs 0] | Write channel to channel skew ($TCCS_{LEAD}$). |
| [lindex $tccs 1] | Write channel to channel skew ($TCCS_{LAG}$). |

For a Stratix III memory interface with QDRII or QDRII+ SRAM, Example 1–13 and Example 1–14 ensure optimum write timing margin.

**Example 1–13. QDR II and QDR II+ Write Margin—Setup**

```
set su   [round_3dp [expr {$tCYC*$dq2dqs_output_phase_offset/360.0 - $board_skew - [lindex $tccs
0]/1000.0 - $tSD}]]
```

**Example 1–14. QDR II and QDR II+ Write Margin—Hold**

```
set hold [round_3dp [expr {$tCYC*(0.5 - $dq2dqs_output_phase_offset/360.0) - $board_skew - [lindex $tccs
1]/1000.0 - $tHD}]]
```

Table 1–22 describes the terms for a Stratix III write timing equation for QDR II and QDR II+ SRAM.

**Table 1–22. Description of Stratix III Write Capture Timing Equation for QDR II and QDR II+ SRAM**

| Term | Description |
|---|---|
| $tCYC | Clock period. |
| $dq2dqs_phase_shift/360.0 | Write phase shift as a percentage of the clock period. |
| $tSD | Memory data setup requirement. |
| $tHD | Memory data hold requirement. |
| $board_skew | The worst case difference in propagation delay between the DQS strobe and any DQ pin in the same group. |
| [lindex $tccs 0] | Write channel to channel skew (TCCSLEAD). |
| [lindex $tccs 1] | Write channel to channel skew (TCCSLAG). |

## Read Resynchronization

In the DDR3, DDR2, and DDR SDRAM interfaces with Arria II GX and Stratix IV FPGAs, the resynchronization timing analysis concerns transferring read data that is captured with a DQS strobe to a clock domain under the control of the ALTMEMPHY. After calibration by a sequencer, a dedicated PLL phase tracks any movements in the data valid window of the captured data. The exact length of the DQS and CK traces

does not affect the timing analysis. The calibration process centers the resynchronization clock phase in the middle of the captured data valid window to maximize the resynchronization setup and hold the margin, and removes any static offset from other timing paths. With the static offset removed, any remaining uncertainties are voltage and temperature variation, jitter and skew.

☞ In a UniPHY interface, a FIFO buffer synchronizes the data transfer from the data capture to the core. The calibration process sets the depth of the FIFO buffer and no dedicated synchronization clock is required.

## Arria II GX and Stratix IV Devices

Table 1–23 describes the terms for a Arria II GX or Stratix IV resynchronization timing equation in <*phy_variation_name*>**_report_timing.tcl**. The timing analysis consists of subtracting the uncertainties from the nominal data valid window of one full-rate clock cycle.

**Table 1–23. Description of Arria II GX and Stratix IV Resynchronization Timing Equation**

| Term | Description |
|---|---|
| $t_{DQSCK}$ | The DQS strobe varies from the CK clocks driven into the memory because of changes in output timing of the CK clocks, and the input timing of DQS IOE are assumed to be compensated by the mimic path. |
| $t_{JITper}$ | Peak-to-peak period jitter when the DLL in lock mode. |
| resync_clock_jitter | Peak-to-peak jitter generated by the PLL and the global or regional clock network used by the resynchronization clock. |
| DQS_path_period_jitter | Peak-to-peak jitter generated on the whole DQS path relative to the DQ path. |
| DQS_delay_chain_period_jitter | Peak-to-peak jitter generated by the DQS delay chain. |
| phy_uncertainty | The accuracy with which the calibration algorithm in the sequencer can predict and track the center of the data valid window. |
| DQS_clock_skew | Maximum arrival skew of the DQS clock network. |
| resync_clock_skew | Maximum arrival skew of the global or regional clock network used by the resynchronization clock. Only for ALTMEMPHY IP. |
| quantization_error | During DDR2 or DDR3 calibration, the ALTMEMPHY megafunction automatically selects the best resynchronization clock phase from the PLL and PVT-compensated read leveling delay chain. Any error from perfectly centering the read data must be considered. |
| t7_vt_variation | Uncertainty introduced due to VT variations in the delay chain used for deskew. |
| SSN(pushout_o), $SSN(pullin_o), SSN(pushout_i), $SSN(pullin_o), | SSN causing pullin or pushout on both the clock sent to the memory and the input back to the FPGA. |
| resync_micro_tsu | Resynchronization register micro setup time requirements. |
| resync_micro_th | Resynchronization register micro hold time requirements. |

**Table 1–23. Description of Arria II GX and Stratix IV Resynchronization Timing Equation**

| Term | Description |
|---|---|
| board (inter_DQS_group_skew) | Skew between different DQS groups, for non-leveled interfaces. |
| board (tpd_inter_DIMM) | Multi rank board skew effect. |

## Stratix III Devices

For a DDR3 SDRAM component with Stratix III memory interface, Example 1–15 ensures optimum read capture timing margin.

**Example 1–15. Optimum Read Capture Timing Margin**

```
set resync_window [expr $::period - 2 * $tDQSCK - 2 * $::tJITper/2 - 2 *
$DQS_clock_period_jitter - $DQS_clock_skew  - $phy_uncertainty - $resync_clock_skew - 2 *
$resync_clock_jitter/2 - 2 * $read_leveling_delay_VT_variation - $resync_micro_tsu -
$resync_micro_th]

set su [round_3dp [expr $resync_window * 0.5]]

set hold [round_3dp [expr $resync_window * 0.5]]
```

The timing analysis consists of subtracting the following uncertainties from the nominal data valid window of one full-rate clock cycle.

Table 1–24 describes the terms for a Stratix III resynchronization timing equation in <*phy_variation_name*>**_report_timing_core.tcl**:

**Table 1–24. Description of Stratix III Resynchronization Timing Equation**

| Term | Description |
|---|---|
| $t_{DQSCK}$ | The DQS strobe varies from the CK clocks driven into the memory because of changes in output timing of the CK clocks, and the input timing of DQS IOE are assumed to be compensated by the mimic path. |
| $t_{JITper}$ | Single period jitter when the DLL in lock mode. |
| DQS_clock_period_jitter | Jitter generated by the DQS delay chain and clock network. |
| phy_uncertainty | The accuracy with which the calibration algorithm in the sequencer can predict and track the center of the data valid window. |
| DQS_clock_skew | Maximum arrival skew of the DQS clock network. |
| resync_clock_skew | Maximum arrival skew of the global or regional clock network used by the resynchronization clock. Only for ALTMEMPHY IP. |
| resync_clock_jitter | Jitter generated by the PLL and the global or regional clock network used by the resynchronization clock. Only for ALTMEMPHY IP. |

**Table 1–24. Description of Stratix III Resynchronization Timing Equation**

| Term | Description |
|---|---|
| read_leveling_delay_VT_variation | During DDR3 calibration, the ALTMEMPHY megafunction automatically selects the best resynchronization clock phase from the PVT-compensated read leveling delay chain and fine tunes the phase selection with an uncompensated DQS delay chain to center the read data at the resynchronization registers. Any variations that may occur in the uncompensated delay chain must be considered. |
| resync_micro_tsu | Resynchronization register micro setup time requirements. |
| resync_micro_th | Resynchronization register micro hold time requirements. |

## Mimic Path

The mimic path mimics the FPGA portion of the elements of the round-trip delay, which enables the calibration sequencer to track delay variations because of voltage and temperature changes during the memory read and write transactions without interrupting the operation of the ALTMEMPHY megafunction.

As the timing path register is integrated in the IOE, there is no timing constraint required for the Arria II GX and Stratix IV device families.

For Cyclone III and Cyclone IV devices, the mimic register is a register in the core and it is placed closer to the IOE by the fitter.

The UniPHY IP does not use any mimic path.

## DQS versus CK—Cyclone III and Cyclone IV Devices

The DQS versus CK timing path indicates the skew requirement for the arrival time of the DQS strobe at the memory with respect to the arrival time of CK/CK# at the memory. Cyclone III and Cyclone IV devices require the DQS strobes and CK clocks to arrive edge aligned.

There are two timing constraints for DQS versus CK timing path to account for duty cycle distortion. The DQS/DQS# rising edge to CK/CK# rising edge ($t_{DQSS}$) requires the rising edge of DQS to align with the rising edge of CK to within 25% of a clock cycle, while the DQS/DQS# falling edge setup/hold time from CK/CK# rising edge ($t_{DSS}/t_{DSH}$) requires the falling edge of DQS to be more than 20% of a clock cycle away from the rising edge of CK.

The TimeQuest Timing Analyzer analyzes the DQS vs CK timing paths using the `set_output_delay (max and min)` constraints as shown below. For a DDR2 SDRAM component with Cyclone III or Cyclone IV memory interface, Example 1–16 ensures optimum DQS versus CK timing margin:

**Example 1–16. Optimum DQS versus CK Timing Margins**

```
set_output_delay -add_delay -clock $ckclock -max [round_3dp [expr {($off_tDQSS+1-$t(DQSS)) *
$t(period) + $t(board_skew) + fpga_tDQSS_SETUP_ERROR}]] $dqspin

set_output_delay -add_delay -clock $ckclock -min [round_3dp [expr {($off_tDQSS+$t(DQSS)) *
$t(period) - $t(board_skew) - $fpga_tDQSS_HOLD_ERROR}]] $dqspin

set_output_delay -add_delay -clock $ckclock -max [round_3dp [expr {($off_tDSS+$t(DSS)) *
$t(period) + $t(board_skew) + $fpga_tDSSH_SETUP_ERROR}]] $dqspin

set_output_delay -add_delay -clock $ckclock -min [round_3dp [expr {($off_tDSS-$t(DSH)) * $t(period) -
$t(board_skew) - $fpga_tDSSH_HOLD_ERROR}]] $dqspin
```

Table 1–25 describes the terms for a Cyclone III or Cyclone IV device DQS versus CK timing constraint <*phy_variation_name*>**_phy_ddr_timing.sdc**:

**Table 1–25. Description of Cyclone III Device DQS versus CK Timing Constraint**

| Term | Description |
|---|---|
| $off_tDQSS * $t(period) | Clock cycle adjustment for dedicated clock mode. For a Cyclone III device interfacing with a 200-MHz DDR2 SDRAM component, this offset is set to zero as DDIO structures is used for `mem_clk` pins. |
| $t(DQSS) * $t(period) | DQS, DQS# rising edge to CK, CK# rising edge period. |
| $t(DSS) * $t(period) | The memory uncertainty. |
| $t(board_skew) | The worst case difference in propagation delay between the DQS strobe and the CK/CK# pins clocking the memory device. |
| $t(DCD_total) | Total duty cycle distortion. |
| $t(DSS) | The DQS/DQS# falling edge setup time from CK/CK# rising edge. |
| $t(DSH) | The DQS/DQS# falling edge hold time from CK/CK# rising edge. |
| $fpga_tDQSS_SETUP_ERROR | Board uncertainties. |
| $fpga_tDSSH_SETUP_ERROR | Board uncertainties. |

## Write Leveling $t_{DQSS}$

In DDR2 SDRAM (with UniPHY) and DDR3 SDRAM (with ALTMEMPHY and UniPHY) interfaces, write leveling $t_{DQSS}$ timing is a calibrated path that details skew margin for the arrival time of the DQS strobe with respect to the arrival time of CK/CK# at the memory side. For proper write leveling configuration, DLL delay chain must be equal to 8.

## Stratix IV GX Devices

For a DDR3 SDRAM component with Stratix IV memory interface, Example 1–17 ensures optimum write leveling $t_{DQSS}$ timing margin:

**Example 1–17. Optimum Write Leveling Timing Margin**

```
set tdqss_setup [expr $tDQSS - $tWLH - $vt_variation -
    $tJITper - $WL_jitter/2 - $SSN(pushout_o) -
    $SSN(pullin_o)]
set tdqss_hold  [expr $tDQSS - $tWLS - $vt_variation -
    $tJITper - $WL_jitter/2 - $SSN(pushout_o) -
    $SSN(pullin_o) - $quantization_error]
```

Table 1–26 describes the terms for a Stratix IV write leveling $t_{DQSS}$ timing equation in *<phy_variation_name>*_ **report_timing.tcl**:

**Table 1–26. Description of a Stratix IV Write Leveling $t_{DQSS}$ Timing Equation**

| Term | Description |
|---|---|
| $t_{WLS}$ | Write leveling setup time from rising CK/CK# crossing point to rising DQS/DQS# crossing point. |
| $t_{WLH}$ | Write leveling hold time from rising DQS/DQS# crossing point to rising CK/CK# crossing point. |
| $vt_variation | During DDR3 SDRAM calibration, the ALTMEMPHY megafunction automatically selects the best write clock phase from the PVT-compensated write leveling delay chain and fine tunes the phase selection with an uncompensated DQS delay chain to edge the read data at the registers. Any variations that may occur in the uncompensated delay chain must be considered. |
| $t_{DQSS}$ | DQS/DQS# rising edge to CK/CK# rising edge. |
| $quantization_error | During DDR3 SDRAM calibration, the ALTMEMPHY megafunction increases the delay on DQS or DQS# until the device is levelled. This procedure may lead to an overshoot of the optimal delay setting, and this error must be considered. |
| $SSN(pushout_o), $SSN(pullin_o), | SSN causing pullin and pushout on both the clock and DQS signal can lead to write leveling degradation. |
| $tJITper | Peak-to-peak period jitter when the DLL in lock mode, affects both clock and DQS. |
| $WL_jitter | Extra peak-to-peak jitter on DQS for going through the write-leveling delay chains. |

## Stratix III Devices

For a DDR3 SDRAM component with Stratix III memory interface, Example 1–18 ensures optimum write leveling $t_{DQSS}$ timing margin:

**Example 1–18. Optimum Write Leveling Timing Margin**

```
set min_write_leveling_inaccuracy [expr $tWLS + $write_leveling_delay_VT_variation]

set max_write_leveling_inaccuracy [expr $tWLH + $write_leveling_delay_VT_variation +
$fpga_leveling_step]

set su [round_3dp [expr $tDQSS - $min_write_leveling_inaccuracy]]

set hold [round_3dp [expr $tDQSS - $max_write_leveling_inaccuracy]]
```

Table 1–26 describes the terms for a Stratix III write leveling $t_{DQSS}$ timing equation in <*phy_variation_name*>_ **report_timing.tcl**:

**Table 1–27. Description of Stratix III Write Leveling $t_{DQSS}$ Timing Equation**

| Term | Description |
|---|---|
| $t_{WLS}$ | Write leveling setup time from rising CK/CK# crossing point to rising DQS/DQS# crossing point. |
| $t_{WLH}$ | Write leveling hold time from rising DQS/DQS# crossing point to rising CK/CK# crossing point. |
| $write_leveling_delay_VT_variation | During DDR3 SDRAM calibration, the ALTMEMPHY megafunction automatically selects the best write clock phase from the PVT-compensated write leveling delay chain and fine tunes the phase selection with an uncompensated DQS delay chain to edge the read data at the registers. Any variations that may occur in the uncompensated delay chain must be considered. |
| $t_{DQSS}$ | DQS/DQS# rising edge to CK/CK# rising edge. |
| $min_write_leveling_inaccuracy/ $max_write_leveling_inaccuracy | Margin of write leveling accuracy. |

## Write Leveling $t_{DSH}/t_{DSS}$

In DDR2 SDRAM (with UniPHY) and DDR3 SDRAM (with ALTMEMPHY and UniPHY) interfaces, write leveling $t_{DSH}/t_{DSS}$ timing details the setup and hold margin for the DQS falling edge with respect to the CK clock at the memory.

### Stratix IV GX Devices

For a DDR3 SDRAM component with Stratix IV memory interface, Example 1–19 ensures optimum write leveling $t_{DSH}/t_{DSS}$ timing margin:

**Example 1–19. Optimum Write Leveling Timing Margin**

```
set tdss [expr 0.5*$period - $tWLS - $tDSS - $vt_variation -
    $tJITper - $WL_jitter/2 - $t(WL_DCJ) - $t(WL_DCD) -
    $SSN(pushout_o) - $SSN(pullin_o) - $quantization_error]
set tdsh  [expr 0.5*$period - $tWLH - $tDSH - $vt_variation -
    $tJITper - $WL_jitter/2 - $t(WL_DCJ) - $t(WL_DCD) - $SSN(pushout_o) - $SSN(pullin_o)]
```

Table 1–28 describes the terms for a Stratix IV write leveling $t_{DSH}/t_{DSS}$ timing equation in <*phy_variation_name*>_ **report_timing.tcl**.

**Table 1–28. Description of a Stratix IV Write Leveling $t_{DSH}/t_{DSS}$ Timing Equation**

| Term | Description |
|---|---|
| $tWLS | Write leveling setup time from rising CK/CK# crossing point to rising DQS/DQS# crossing point. |
| $tWLH | Write leveling hold time from rising DQS/DQS# crossing point to rising CK/CK# crossing point. |

**Table 1–28. Description of a Stratix IV Write Leveling $t_{DSH}/t_{DSS}$ Timing Equation**

| Term | Description |
|---|---|
| $vt_variation | During DDR3 SDRAM calibration, the ALTMEMPHY megafunction automatically selects the best write clock phase from the PVT-compensated write leveling delay chain and fine tunes the phase selection with an uncompensated DQS delay chain to edge the read data at the registers. Any variations that may occur in the uncompensated delay chain must be considered. |
| $tDSS | The DQS/DQS# falling edge setup time from CK/CK# rising edge. |
| $tDSH | The DQS/DQS# falling edge hold time from CK/CK# rising edge. |
| $quantization_error | During DDR3 SDRAM calibration, the ALTMEMPHY megafunction increases the delay on DQS or DQS# until the device is levelled. This procedure may lead to an overshoot of the optimal delay setting, and this error must be considered. |
| $SSN(pushout_o), $SSN(pullin_o), | SSN causing pullin or pushout on both the clock and DQS signal can lead to write leveling degradation. |
| $tJITper | Peak-to-peak period jitter when the DLL in lock mode, affects both clock and DQS. |
| $WL_jitter | Extra peak-to-peak jitter on DQS for going through the write-leveling delay chains. |
| $t(WL_DCJ) | Duty-cycle jitter on DQS for going through the write-leveling delay chains. |
| $t(WL_DCD) | Duty-cycle-distortion on DQS for going through the write-leveling delay chains. |

## Stratix III Devices

For a DDR3 SDRAM component with Stratix III memory interface, Example 1–20 ensures optimum write leveling $t_{DSH}/t_{DSS}$ timing margin:

**Example 1–20. Optimum Write Leveling Timing Margin**

```
set su [round_3dp [expr $min_DQS_low - $max_write_leveling_inaccuracy -
$CK_period_jitter - $tDSS]]
set hold [round_3dp [expr $min_DQS_high - $min_write_leveling_inaccuracy - $tDSH]]
```

Table 1–29 describes the terms for a Stratix III or Stratix IV write leveling $t_{DSH}/t_{DSS}$ timing equation in *<phy_variation_name>*_ **report_timing.tcl**:

**Table 1–29. Description of Stratix III and Stratix IV Write Leveling $t_{DSH}/t_{DSS}$ Timing Equation**

| Term | Description |
|---|---|
| $min_DQS_high | Minimum specification on DQS high. |
| $min_DQS_low | Minimum specification on DQS low for memory. |
| $CK_period_jitter | The largest deviation of any clock period signal from the average clock period across any consecutive 200 cycle window ($t_{CK}$(avg)) for memory. |
| $tDSS | The DQS/DQS# falling edge setup time from CK/CK# rising edge. |
| $tDSH | The DQS/DQS# falling edge hold time from CK/CK# rising edge. |
| $min_write_leveling_inaccuracy/ $max_write_leveling_inaccuracy | Margin of write leveling accuracy. |

# Timing Margin Report

The **Report DDR** task in the TimeQuest Timing Analyzer generates custom timing margin reports for all ALTMEMPHY and UniPHY instances in your design. The TimeQuest Timing Analyzer generates this custom report by sourcing the wizard-generated <*variation*>**_report_timing.tcl** script.

This <*variation*>**_report_timing.tcl** script reports the timing slacks on specific paths of the DDR SDRAM design for example, such as:

- Read capture
- Read resynchronization
- Mimic, address and command
- Core
- Core reset and removal
- Half-rate address and command
- DQS versus CK
- Write
- Write leveling ($t_{DQSS}$)
- Write leveling ($t_{DSS}/t_{DSH}$)

The <*variation_name*>**_report_timing.tcl** script checks the design rules and assumptions as listed in "Timing Model Assumptions and Design Rules" on page 1–48. If you do not adhere to these assumptions and rules, you receive critical warnings when the TimeQuest Timing Analyzer runs during compilation or when you run the **Report DDR** task.

ALTMEMPHY- and UniPHY-based memory interface designs do not support the use of report data sheet timing specifications for analyzing margins on DDR I/O timing paths (read and write datapaths). Instead, you must use **Report DDR** to perform I/O timing analysis with the TCCS and SW timing specifications described in the "Timing Margin Components" on page 1–17. Report data sheet results are based on the micro-timing model of the FPGA, and do not use the memory interface TCCS or SW specifications.

To generate a timing margin report, follow these steps:

1. Compile your design in the Quartus II software.
2. Launch the TimeQuest Timing Analyzer.
3. Double-click **Report DDR** from the **Tasks** pane. This action automatically executes the **Create Timing Netlist**, **Read SDC File**, and **Update Timing Netlist** tasks for your project.


The **.sdc** file may not be applied correctly if the variation top-level file is the top-level file of the project. You must have the top-level file of the project instantiate the variation top-level file.

The **Report DDR** feature creates a new DDR folder in the TimeQuest Timing Analyzer **Report** pane. Expanding the DDR folder reveals the detailed timing information for each PHY timing path, in addition to an overall timing margin summary for the ALTMEMPHY or UniPHY instance, as shown in Figure 1–10 and Figure 1–11.

**Figure 1–10. DDR Timing Report in the TimeQuest Window Report Pane**

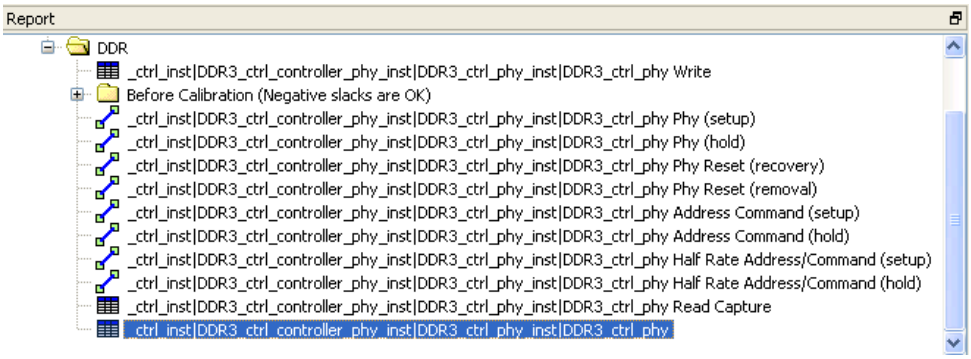



**Figure 1–11. Timing Margin Summary Window Generated by the Report DDR Task**

DDR3_ctrl_inst|DDR3_ctrl_controller_phy_inst|DDR3_ctrl_phy_inst|DDR3_ctrl_phy

| | Path | Operating Condition | Setup Slack | Hold Slack |
|---|---|---|---|---|
| 1 | Address Command (Slow 900mV 85C Model) | Slow 900mV 85C Model | 0.323 | 0.283 |
| 2 | Half Rate Address/Command (Slow 900mV 85C Model) | Slow 900mV 85C Model | 1.971 | 0.340 |
| 3 | Phy (Slow 900mV 85C Model) | Slow 900mV 85C Model | 0.067 | 0.069 |
| 4 | Phy Reset (Slow 900mV 85C Model) | Slow 900mV 85C Model | 0.685 | 0.360 |
| 5 | Read Capture (Slow 900mV 85C Model) | Slow 900mV 85C Model | 0.187 | 0.147 |
| 6 | Read Resync (All Conditions) | All Conditions | 0.191 | 0.218 |
| 7 | Write (Slow 900mV 85C Model) | Slow 900mV 85C Model | 0.167 | 0.167 |
| 8 | Write Leveling tDQSS (All Conditions) | All Conditions | 0.344 | 0.318 |
| 9 | Write Leveling tDSS/tDSH (All Conditions) | All Conditions | 0.343 | 0.368 |

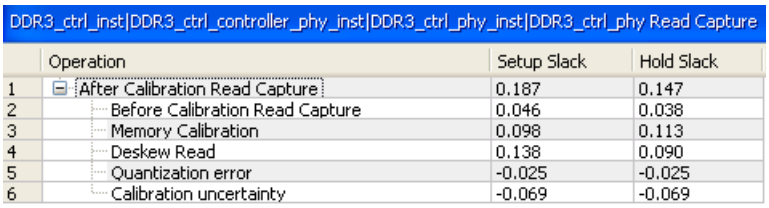
Figure 1–12 shows the timing margin report from the TimeQuest Timing Analyzer Console window. This is a command-line version of the report in Figure 1–11.

**Figure 1–12. Timing Margin Summary Information in TimeQuest Console Window**

```
tcl> report_ddr -panel_name "DDR"
Info: Report Timing: Found 100 setup paths (0 violated).  Worst case slack is 0.067
Info: Report Timing: Found 100 hold paths (0 violated).  Worst case slack is 0.069
Info: Report Timing: Found 100 recovery paths (0 violated).  Worst case slack is 0.685
Info: Report Timing: Found 100 removal paths (0 violated).  Worst case slack is 0.360
Info: Report Timing: Found 12 setup paths (0 violated).  Worst case slack is 0.323
Info: Report Timing: Found 12 hold paths (0 violated).  Worst case slack is 0.283
Info: Report Timing: Found 80 setup paths (0 violated).  Worst case slack is 1.971
Info: Report Timing: Found 80 hold paths (0 violated).  Worst case slack is 0.340
Info:                                                        setup  hold
Info: Address Command (Slow 900mV 85C Model)                | 0.323  0.283
Info: Half Rate Address/Command (Slow 900mV 85C Model)      | 1.971  0.340
Info: Phy (Slow 900mV 85C Model)                            | 0.067  0.069
Info: Phy Reset (Slow 900mV 85C Model)                      | 0.685  0.360
Info: Read Capture (Slow 900mV 85C Model)                   | 0.187  0.147
Info: Read Resync (All Conditions)                          | 0.191  0.218
Info: Write (Slow 900mV 85C Model)                          | 0.167  0.167
Info: Write Leveling tDQSS (All Conditions)                 | 0.344  0.318
Info: Write Leveling tDSS/tDSH (All Conditions)             | 0.343  0.368
tcl>
```

Figure 1–13 and Figure 1–14 show the read capture and write margin summary window generated by the Report DDR Task for a DDR3 core. It first shows the timing results calculated using the FPGA timing model. The *<variation_name>*_**report_timing_core.tcl** then adjusts these numbers to account for effects that are not modeled by either the timing model or by TimeQuest Timing Analyzer.

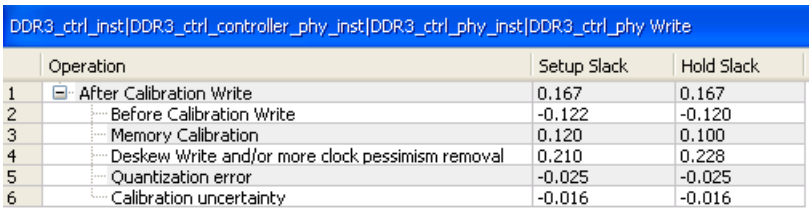
**Figure 1–13. Read Capture Margin Summary Window**



DDR3_ctrl_inst|DDR3_ctrl_controller_phy_inst|DDR3_ctrl_phy_inst|DDR3_ctrl_phy Read Capture

| | Operation | Setup Slack | Hold Slack |
|---|---|---|---|
| 1 | After Calibration Read Capture | 0.187 | 0.147 |
| 2 | Before Calibration Read Capture | 0.046 | 0.038 |
| 3 | Memory Calibration | 0.098 | 0.113 |
| 4 | Deskew Read | 0.138 | 0.090 |
| 5 | Quantization error | -0.025 | -0.025 |
| 6 | Calibration uncertainty | -0.069 | -0.069 |

**Figure 1–14. Write Capture Margin Summary Window**



DDR3_ctrl_inst|DDR3_ctrl_controller_phy_inst|DDR3_ctrl_phy_inst|DDR3_ctrl_phy Write

| | Operation | Setup Slack | Hold Slack |
|---|---|---|---|
| 1 | After Calibration Write | 0.167 | 0.167 |
| 2 | Before Calibration Write | -0.122 | -0.120 |
| 3 | Memory Calibration | 0.120 | 0.100 |
| 4 | Deskew Write and/or more clock pessimism removal | 0.210 | 0.228 |
| 5 | Quantization error | -0.025 | -0.025 |
| 6 | Calibration uncertainty | -0.016 | -0.016 |

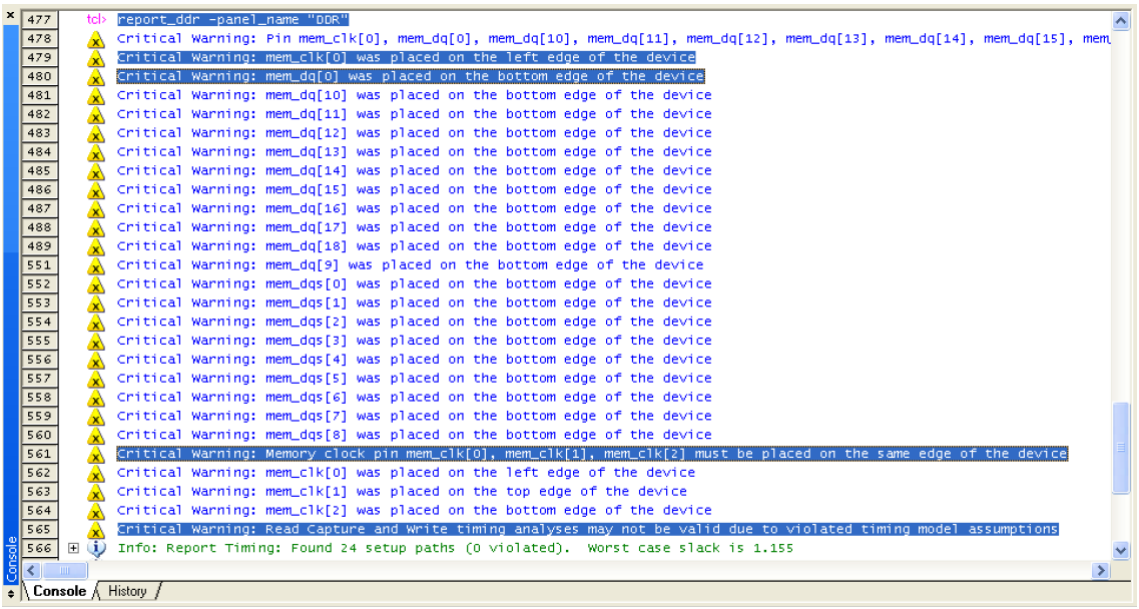
# Timing Model Assumptions and Design Rules

External memory interfaces using Altera IP are optimized for highest performance, and use a high-performance timing model to analyze calibrated and source-synchronous, double-data rate I/O timing paths. This timing model applies to designs that adhere to a set of predefined assumptions. These timing model assumptions include memory interface pin-placement requirements, PLL and clock network usage, I/O assignments (including I/O standard, termination, and slew rate), and many others.

For example, the read and write datapath timing analysis is based on the FPGA pin-level $t_{TCCS}$ and $t_{SW}$ specifications, respectively. While calculating the read and write timing margins, the Quartus II software analyzes the design to ensure that all read and write timing model assumptions are valid for your variation instance.

When the Report DDR task or **report_timing.tcl** script is executed, the timing analysis assumptions checker is invoked with specific variation configuration information. If a particular design rule is not met, the Quartus II software reports the failing assumption as a Critical Warning message. Figure 1–15 shows a sample set of messages generated when the memory interface DQ, DQS, and CK/CK# pins are not placed in the same edge of the device.

**Figure 1–15. Read and Write Timing Analysis Assumption Verification**




## Memory Clock Output Assumptions

To verify the quality of the FPGA clock output to the memory device (CK/CK# or K/K#), which affects FPGA performance and quality of the read clock/strobe used to read data from the memory device, the following assumptions are necessary:

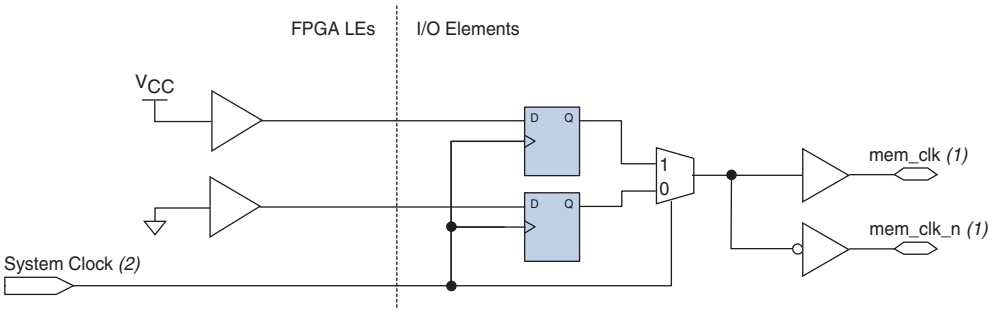
- The slew rate setting must be **Fast** or an on-chip termination (OCT) setting must be used.
- The output delay chains must all be **0** (the default value applied by the Quartus II software). These delay chains include the Cyclone III output register to pin delay chain and the Stratix III D5 and D6 output delay chains.
- The output open-drain parameter on the memory clock pin `IO_OBUF` atom must be **Off**. The **Output Open Drain** logic option must not be enabled.
- The weak pull-up on the CK and CK# pads must be **Off**. The **Weak Pull-Up Resistor** logic option must not be enabled.
- The bus hold on the CK and CK# pads must be **Off**. The **Enable Bus-Hold Circuitry** logic option must not be enabled.
- All CK and CK# pins must be declared as output-only pins or bi-directional pins with the output enable set to $V_{CC}$.

## Arria II, Stratix IV, and Stratix V Devices

For Arria II, Stratix IV, and Stratix V devices the following additional memory clock assumptions are necessary (however, Arria II GZ and Stratix V devices support only the UnIPHY timing model assumptions):

- All memory clock output pins must be placed on DIFFOUT pin pairs.
  - In all DDR3 designs and DDR2 using differential DQS/DQS# strobes, (mem_clk[0] and mem_clk_n[0]) must be placed on DIFFIO_RX p- and n- pins for voltage and temperature tracking using the mimic path. The remaining memory output clock pins (mem_clk[i] and mem_clk_n[i], where i = 1 or greater), can be placed on the DIFFOUT p- and n- pins.
  - For all other cases (including when the CK pin is a mimic pin, but the design does not use true differential DQS), the memory clock output pins must be placed on DIFFOUT p- and n- pins.
- DDIO output registers must feed the CK output pin.
- The CK# output pin must be fed by a signal splitter from its associated CK pin.
- All memory clock pins must be placed on the same edge of the device (top, bottom, left, or right).
- For DDR3 with ALTMEMPHY interfaces, the CK pins must be placed on FPGA output pins marked DQ, DQS, or DQSn.
- For DDR3 interfaces, the CK pin must be fed by an OUTPUT_PHASE_ALIGNMENT WYSIWYG with a 0° phase shift.
- For DDR3 interfaces, the PLL clock driving CK pins must be the same as the clock driving the DQS pins.
- The T4 (DDIO_MUX) delay chains must be set to **2**.
- The memory output clock signals must be generated with the DDIO configuration shown in Figure 1–16, with a signal splitter to generate the n- pin pair and a regional clock network-to-clock to output DDIO block.

**Figure 1–16. DIDO Configuration with Signal Splitter**




**Notes to Figure 1–16:**

(1) The mem_clk[0] and mem_clk_n[0] pins for DDR3, DDR2, and DDR SDRAM interfaces use the I/O input buffer for feedback, therefore bidirectional I/O buffers are used for these pins. For memory interfaces using a differential DQS input, the input feedback buffer is configured as differential input; for memory interfaces using a single-ended DQS input, the input buffer is configured as a single-ended input. Using a single-ended input feedback buffer requires that the I/O standard's VREF voltage is provided to that I/O bank's VREF pins.

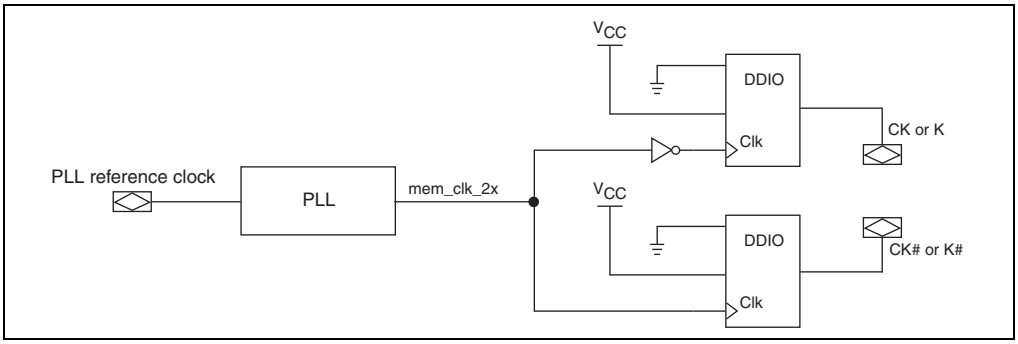
(2) Regional QCLK (quadrant) networks are required for memory output clock generation to minimize jitter.

## Cyclone III Devices

For Cyclone III devices the following additional memory clock assumptions are necessary:

- The memory clock output pins must be fed by DDIO output registers and placed on DIFFIO `p-` and `n-` pin pairs.
- The memory output clock signals must be generated using the DDIO configuration shown in Figure 1–17. In this configuration, the high register connects to $V_{CC}$ and the low register connects to GND.

**Figure 1–17. DDIO Configuration**



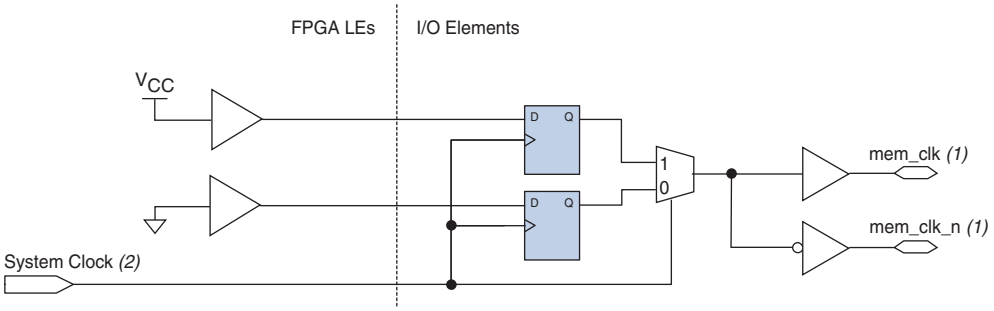

- CK and CK# pins must be fed by a DDIO_OUT WYSIWYG with `datainhi` connected to GND and `datainlo` connected to $V_{CC}$.
- CK or K pins must be fed by a DDIO_OUT with its clock input from the PLL inverted.
- CK# or K# pins must be fed by a DDIO_OUT with its clock input from the PLL uninverted.
- The I/O standard and current strength settings on the memory clock output pins must be as follows:
  - SSTL-2 Class I and 12 mA, or SSTL-2 Class II and 16 mA for DDR SDRAM interfaces
  - SSTL-18 Class I and 12 mA, or SSTL-18 Class II and 16 mA for DDR2 SDRAM interfaces

## Stratix III Devices

For Stratix III devices the following additional memory clock assumptions are necessary:

- All memory clock output pins must be placed on DIFFOUT pin pairs on the same edge of the device.
  - In all DDR2 and DDR3 SDRAM designs using differential DQS/DQS# strobes, (mem_clk[0] and mem_clk_n[0]) must be placed on DIFFIO_RX p- and n- pins for voltage and temperature tracking using the mimic path. The remaining memory output clock pins (mem_clk[i] and mem_clk_n[i], where i = 1 or greater), can be placed on the DIFFOUT p- and n- pins.
  - For DDR SDRAM designs where the CK pin is a mimic pin, but the design does not use true differential DQS, the memory clock output pins must be placed on DIFFOUT p- and n- pins.
- For DDR3 SDRAM interfaces:
  - The CK pins must be placed on FPGA output pins marked DQ, DQS, or DQSn.
  - The CK pin must be fed by an OUTPUT_PHASE_ALIGNMENT WYSIWYG with a 0° phase shift for ALTMEMPHY-based interfaces.
  - The PLL clock driving CK pins must be the same as the clock driving the DQS pins.
  - The T4 (DDIO_MUX) delay chains setting for the memory clock pins must be the same as the settings for the DQS pins.
- For non-DDR3 interfaces, the T4 (DDIO_MUX) delay chains setting for the memory clock pins must be greater than 0.
- The programmable rise and fall delay chain settings for all memory clock pins must be set to 0.
- The memory output clock signals must be generated with the DDIO configuration shown in Figure 1–18, with a signal splitter to generate the n- pin pair and a regional clock network-to-clock to output DDIO block.

**Figure 1–18. DIDO Configuration with Signal Splitter**




**Notes to Figure 1–18:**

(1) The mem_clk[0] and mem_clk_n[0] pins for DDR3, DDR2, and DDR SDRAM interfaces use the I/O input buffer for feedback, therefore bidirectional I/O buffers are used for these pins. For memory interfaces using a differential DQS input, the input feedback buffer is configured as differential input; for memory interfaces using a single-ended DQS input, the input buffer is configured as a single-ended input. Using a single-ended input feedback buffer requires that the I/O standard's VREF voltage is provided to that I/O bank's VREF pins.

(2) Regional QCLK (quadrant) networks are required for memory output clock generation to minimize jitter.

## Write Data Assumptions

To verify that the ALTMEMPHY-based memory interface can use the FPGA TCCS output timing specifications, the following assumptions are necessary:

- For QDRII, QDRII+, and RLDRAM II SIO memory interfaces, the write clock output pins (such as K/K# or DK/DK#) must be placed in `DQS/DQSn` pin pairs.
- The PLL clock used to generate the write-clock signals and the PLL clock used to generate the write-data signals must come from the same PLL.
- The slew rate for all write clocks and write data pins must be set to **Fast** or OCT must be used.
- When auto deskew is not enabled (or not supported by the ALTMEMPHY configuration), the output delay chains and output enable delay chains must all be set to the default values applied by Quartus II. These delay chains include the Cyclone III output register and output enable register-to-pin delay chains, and the Stratix III D5 and D6 delay chains.
- The output open drain for all write clocks and write data pins' `IO_OBUF` atom must be set to **Off**. The **Output Open Drain** logic option must not be enabled.
- The weak pull-up for all write clocks and write data pins must be set to **Off**. The **Weak Pull-Up Resistor** logic option must not be enabled.
- The Bus Hold for all write clocks and write data pins must be set to **Off**. The **Enable Bus-Hold Circuitry** logic option must not be enabled.

### Arria II, Stratix IV, and Stratix V Devices

For Arria II, Stratix IV, and Stratix V devices the following additional write data assumptions are necessary (however, Arria II GZ and Stratix V devices support only the UnIPHY timing model assumptions):

- Differential write clock signals (DQS/DQSn) must be generated using the signal splitter.
- The write data pins (DQ/DM/BWS#) must be placed in related `DQ` pins associated with the chosen `DQS` pin. The only exception to this rule is for QDRII and QDRII+ ×36 interfaces emulated using two ×18 DQ groups. For such interfaces, all of the write data pins must be placed on the same edge of the device (left, right, top, or bottom). Also, the write clock K/K# pin pair should be placed on one of the `DQS/DQSn` pin pairs on the same edge.
- All write clock pins must have similar circuit structure.
  - For DDR3 interfaces, all `DQS/DQS#` write strobes must be fed by DDIO output registers clocked by the write-leveling delay chain in the OUTPUT_PHASE_ALIGNMENT block.
  - For non-DDR3 memory interfaces, all write clock pins must be fed by DDIO output registers clocked by a global or regional clock network.

- All write data pins must have similar circuit structure.
  - For DDR3 interfaces (ALTMEMPHY) or leveling interfaces (UniPHY), all write data pins must be fed by either DDIO output registers clocked by the OUTPUT_PHASE_ALIGNMENT block, $V_{CC}$, or GND.
  - For non-DDR3 memory interfaces (ALTMEMPHY) or non-leveling interfaces (UniPHY), all write data pins must be fed by either DDIO output registers clocked by a global or regional clock network, $V_{CC}$, or GND.
- The write clock output must be 72,° 90°, or 108° more than the write data output.
  - For DDR3 interfaces, the write-leveling delay chain in the OUTPUT_PHASE_ALIGNMENT block must implement a phase shift of 72°, 90°, or 108° to center-align write clock with write data.
  - For non-DDR3 memory interfaces, the phase shift of the PLL clock used to clock the write clocks must be 72 to 108° more than the PLL clock used to clock the write data clocks to generated center-aligned clock and data.
- The T4 (DDIO_MUX) delay chains must all be set to **3**. When differential DQS (using splitter) is used, T4 must be set to **2**.

Table 1–30 lists I/O standards supported for the write clock and write data signals for each memory type and pin location.

**Table 1–30. I/O standards**

| Memory Type | Placement | Legal I/O Standards for DQS | Legal I/O Standards for DQ |
|---|---|---|---|
| DDR3 SDRAM | Row I/O | Differential 1.5-V SSTL Class I | 1.5-V SSTL Class I |
| DDR3 SDRAM | Column I/O | Differential 1.5-V SSTL Class I<br>Differential 1.5-V SSTL Class II | 1.5-V SSTL Class I<br>1.5-V SSTL Class II |
| DDR2 SDRAM | Any | SSTL-18 Class I<br>SSTL-18 Class II<br>Differential 1.8V SSTL Class I<br>Differential 1.8V SSTL Class II | SSTL-18 Class I<br>SSTL-18 Class II |
| DDR SDRAM | Any | SSTL-2 Class I<br>SSTL-2 Class II | SSTL-2 Class I<br>SSTL-2 Class II |
| QDRII and QDR II + SRAM | Any | HSTL-1.5 Class I<br>HSTL-1.8 Class I | HSTL-1.5 Class I<br>HSTL-1.8 Class I |
| RLDRAM II | Any | HSTL-1.5 Class I<br>HSTL-1.8 Class I | HSTL-1.5 Class I<br>HSTL-1.8 Class I |

## Cyclone III Devices

For Cyclone III devices the following additional write data assumptions are necessary:

- Write data pins (including the DM pins) must be placed on DQ pins related to the selected DQS pins.
- All write clock pins (DQS/DQS#) must be fed by DDIO output registers.
- All write data pins must be fed by DDIO output registers, $V_{CC}$, or GND.

- The phase shift of the PLL clock used to generate the write clocks must be 72° to 108° more than the PLL clock used to generate the write data (nominally 90° offset).
- The I/O standard and current strength settings on the write data- and clock-output pins must be as follows:
  - SSTL-2 Class I and 12 mA, or SSTL-2 Class II and 16 mA for DDR SDRAM interfaces
  - SSTL-18 Class I and 8/12 mA, or SSTL-18 Class II and 16 mA for DDR2 SDRAM interfaces

## Stratix III Devices

For Stratix III devices the following additional write data assumptions are necessary:

- Differential write clock signals (DQS/DQSn) must be generated using the signal splitter.
- The write data pins (including the DM pins) must be placed in related `DQ` pins associated with the chosen `DQS` pin. The only exception to this rule is for QDRII and QDRII+ ×36 interfaces emulated using two ×18 DQ groups. For such interfaces, all of the write data pins must be placed on the same edge of the device (left, right, top, or bottom). Also, the write clock K/K# pin pair should be placed on one of the `DQS/DQSn` pin pairs on the same edge.
- All write clock pins must have similar circuit structure.
  - For DDR2 SDRAM interfaces (only UniPHY) and DDR3 SDRAM with leveling interfaces (ALTMEMPHY and UniPHY), all `DQS/DQS#` write strobes must be fed by DDIO output registers clocked by the write-leveling delay chain in the OUTPUT_PHASE_ALIGNMENT block.
  - For DDR and DDR2 SDRAM interfaces (only ALTMEMPHY or non-leveling UniPHY), all write clock pins must be fed by DDIO output registers clocked by a global or regional clock network.
- All write data pins must have similar circuit structure.
  - For DDR3 SDRAM interfaces, all write data pins must be fed by either DDIO output registers clocked by the OUTPUT_PHASE_ALIGNMENT block, $V_{CC}$, or GND.
  - For DDR and DDR2 SDRAM interfaces, all write data pins must be fed by either DDIO output registers clocked by a global or regional clock network, $V_{CC}$, or GND.
- The write clock output must be 72,° 90°, or 108° more than the write data output.
  - For DDR2 SDRAM (only UniPHY) and DDR3 SDRAM (ALTMEMPHY and UniPHY) with leveling interfaces, the write-leveling delay chain in the OUTPUT_PHASE_ALIGNMENT block must implement a phase shift of 72°, 90°, or 108° to center-align write clock with write data.
  - For DDR and DDR2 SDRAM interfaces (only ALTMEMPHY or non-leveling UniPHY), the phase shift of the PLL clock used to clock the write clocks must be 72 to 108° more than the PLL clock used to clock the write data clocks to generated center-aligned clock and data.

- The T4 (DDIO_MUX) delay chains must all be set to **3**. When differential DQS (using splitter) is used, T4 must be set to **2**.
- The programmable rise and fall delay chain settings for all memory clock pins must be set to **0**.

Table 1–31 lists I/O standards supported for the write clock and write data signals for each memory type and pin location.

**Table 1–31. I/O standards**

| Memory Type | Placement | Legal I/O Standards for DQS | Legal I/O Standards for DQ |
|---|---|---|---|
| DDR3 SDRAM | Row I/O | Differential 1.5-V SSTL Class I | 1.5-V SSTL Class I |
| DDR3 SDRAM | Column I/O | Differential 1.5-V SSTL Class I<br>Differential 1.5-V SSTL Class II | 1.5-V SSTL Class I<br>1.5-V SSTL Class II |
| DDR2 SDRAM | Any | SSTL-18 Class I<br>SSTL-18 Class II<br>Differential 1.8V SSTL Class I<br>Differential 1.8V SSTL Class II | SSTL-18 Class I<br>SSTL-18 Class II |
| DDR SDRAM | Any | SSTL-2 Class I<br>SSTL-2 Class II | SSTL-2 Class I<br>SSTL-2 Class II |
| QDRII and QDR II + SRAM | Any | HSTL-1.5 Class I<br>HSTL-1.8 Class I | HSTL-1.5 Class I<br>HSTL-1.8 Class I |
| RLDRAM II | Any | HSTL-1.5 Class I<br>HSTL-1.8 Class I | HSTL-1.5 Class I<br>HSTL-1.8 Class I |

## Read Data Assumptions

To verify that the external memory interface can use the FPGA Sampling Window (SW) input timing specifications, the following assumptions are necessary:

- The read clocks input pins must be placed on DQS pins. DQS/DQS# inputs must be placed on differential DQS/DQSn pins on the FPGA.
- Read data pins (DQ) must be placed on the DQ pins related to the selected DQS pins.
- For QDR II and QDR II+ SRAM interfaces, the complementary read clocks must have a single-ended I/O standard setting of HSTL-18 Class I or HSTL-15 Class I.
- For RLDRAM II interfaces, the differential read clocks must have a single ended I/O standard setting of HSTL 18 Class I or HSTL 15 Class I.

### Arria II, Stratix IV, and Stratix V Devices

For Arria II, Stratix IV, and Stratix V devices the following additional read data and mimic pin assumptions are necessary (however, Arria II GZ and Stratix V devices support only the UnIPHY timing model assumptions):

- For DDR3, DDR2, and DDR SDRAM interfaces, the read clock pin can only drive a DQS bus clocking a ×4 or ×9 DQ group.

- For QDRII, QDRII+, and RLDRAM II memory interfaces, the read clock pin can only drive a DQS bus clocking a ×9, ×18, or ×36 DQ group.
- For non-wraparound interfaces, the mimic pin (only ALTMEMPHY), all read clock, and all read data pins must be placed on the same edge of the device (top, bottom, left, or right). For wraparound interfaces, these pins can be placed on adjacent row I/O and column I/O edges and operate at reduced frequencies. Wraparound interfaces are not supported in Stratix IV devices (all interface pins must be placed on the same edge).
- All read data pins and the mimic pin (only ALTMEMPHY) must feed DDIO_IN registers.
- DQS phase-shift setting must be either 72° or 90° (supports only one phase shift for each operating band and memory standard).
- All read clock pins must have the `dqs_ctrl_latches_enable` parameter of its DQS_DELAY_CHAIN WYSIWYG set to false.

## Cyclone III Devices

For Cyclone III devices the following additional read data and mimic pin assumptions are necessary:

- The I/O standard setting on read data and clock input pins must be as follows:
  - SSTL-2 Class I and Class II for DDR SDRAM interface
  - SSTL-18 Class I and Class II for DDR2 SDRAM interfaces
- The read data and mimic input registers (flip-flops fed by the read data pin's input buffers) must be placed in the LAB adjacent to the read data pin. A read data pin can have 0 input registers.
- Specific routing lines from the IOE to core read data/mimic registers must be used. The Quartus II Fitter ensures proper routing unless user-defined placement constraints or LogicLock™ assignments force non-optimal routing. User assignments that prevent input registers from being placed in the LAB adjacent to the IOE must be removed.
- The read data and mimic input pin input pad to core/register delay chain must be set to **0**.
- If all read data pins are on row I/Os or column I/Os, the mimic pin must be placed in the same type of I/O (row I/O for read-data row I/Os, column I/O for read-data column I/Os). For wraparound cases, the mimic pin can be placed anywhere.

## Stratix III Devices

For Stratix III devices the following additional read data and mimic pin assumptions are necessary:

- For DDR3, DDR2, and DDR SDRAM interfaces, the read clock pin can only drive a DQS bus clocking a ×4 or ×9 DQ group.
- For QDR II, QDR II+ SRAM, and RLDRAM II interfaces, the read clock pin can only drive a DQS bus clocking a ×9, ×18, or ×36 DQ group.

- For non-wraparound DDR, DDR2, and DDR3 interfaces, the mimic pin (only ALTMEMPHY), all read clock, and all read data pins must be placed on the same edge of the device (top, bottom, left, or right). For wraparound interfaces, these pins can be placed on adjacent row I/O and column I/O edges and operate at reduced frequencies.
- All read data pins and the mimic pin (only ALTMEMPHY) must feed DDIO_IN registers and their input delay chains D1, D2, and D3 set to the Quartus II default.
- DQS phase-shift setting must be either 72° or 90° (supports only one phase shift for each operating band and memory standard).
- All read clock pins must have the `dqs_ctrl_latches_enable` parameter of its DQS_DELAY_CHAIN WYSIWYG set to false.
- The read clocks pins must have their D4 delay chain set to the Quartus II default value of **0**.
- The read data pins must have their T8 delay chain set to the Quartus II default value of **0**.
- When differential DQS strobes are used (DDR3 and DDR2 SDRAM), the mimic pin must feed a true differential input buffer. Placing the memory clock pin on a `DIFFIO_RX` pin pair allows the mimic path to track timing variations on the DQS input path.
- When single ended DQS strobes are used, the mimic pin must feed a single ended input buffer.

## Mimic Path Assumptions

To verify that the ALTMEMPHY-based DDR, DDR2, or DDR3 SDRAM interface's mimic path is configured correctly, the mimic path input must be placed on the `mem_clk[0]` pin.

### Arria II GX, Stratix III and Stratix IV Devices

To verify that the voltage and temperature tracking mimic path in ALTMEMPHY designs is set up correctly, the following assumptions are necessary:

- When differential DQS strobes are used (DDR3 and DDR2), the mimic pin must feed a true-differential input buffer. Placing the memory clock pin on a `DIFFIO_RX` pin pair allows the mimic path to track timing variations on the DQS input path.
- When single-ended DQS strobes are used, the mimic pin must feed a single-ended input buffer.

## DLL Assumptions

The following DLL assumptions are necessary:

These assumptions do not apply to Cyclone III devices.

- The DLL must directly feed its `delayctrlout[]` outputs to all DQS pins without intervening logic or inversions.

- The DLL must be in a valid frequency band of operation as defined in the corresponding device data sheet.
- The DLL must have jitter reduction mode and dual-phase comparators enabled.

## PLL and Clock Network Assumptions

The PLL and clock network assumptions vary for each device family.

### Arria II, Stratix III, Stratix IV, and Stratix V Devices

- The PLL that generates the memory output clock signals and write data and clock signals must be set to **No compensation** mode in Stratix IV and Stratix V devices to minimize output clock jitter.
- The reference input clock signal to the PLL must be driven by the dedicated clock input pin located adjacent to the PLL, or from the clock output signal from the adjacent PLL. To minimize output clock jitter, the reference input clock pin to the ALTMEMPHY PLL must not be routed through the core using global or regional clock networks. If the reference clock cascades from another PLL, that upstream PLL must be in **No compensation** mode and **Low bandwidth** mode.
- For DDR3 and DDR2 SDRAM interfaces, use only regional or dual regional clock networks to route PLL outputs that generate the write data, write clock, and memory output clock signals. This requirement ensures that the memory output clocks (CK/CK#) meet the memory device input clock jitter specifications, and that output timing variations or skews are minimized.
- For other memory types, the same clock tree type (global, regional, or dual regional) is recommended for PLL clocks generating the write clock, write data, and memory clock signals to minimize timing variations or skew between these outputs.

### Cyclone III Devices

To verify that the memory interface's PLL is configured correctly, the following assumptions are necessary:

- The PLL that generates the memory output clock signals and write data/clock signals must be set to normal compensation mode in Cyclone III devices.
- PLL cascading is not supported.
- The reference input clock signal to the PLL must be driven by the dedicated clock input pin located adjacent to the PLL. The reference input clock pin must not be routed through the core using global or regional clock networks to minimize output clock jitter.

# 2. Timing Closure

This chapter describes common issues and how to optimize timing.

## Common Issues

This topic describes potential timing closure issues that can occur when using the ALTMEMPHY or UniPHY IP. For possible timing closure issues with ALTMEMPHY or UniPHY variations, refer to the *Quartus II Software Release Notes* for the software version that you are using. You can solve some timing issues by moving registers or changing the project fitting setting to **Standard** (from **Auto**).

The *Quartus II Software Release Notes* list common timing issues that can be encountered in a particular version of the Quartus II software.

### Missing Timing Margin Report

The ALTMEMPHY and UniPHY timing margin reports may not be generated during compilation if the **.sdc** does not appear in the Quartus II project settings.

Timing margin reports are not generated if you specify the ALTMEMPHY or UniPHY variation as the top-level project entity. Instantiate the ALTMEMPHY or UniPHY variation as a lower level module in your user design or memory controller.

### Incomplete Timing Margin Report

The timing report may not include margin information for certain timing paths if certain memory interface pins are optimized away during synthesis. Verify that all memory interface pins appear in the <*variation*>**_autodetectedpins.tcl** (ALTMEMPHY) or <*variation*>**_all_pins.txt** (UniPHY) file generated during compilation, and ensure that they connect to the I/O pins of the top-level FPGA design.

### Read Capture Timing

In Stratix III and Stratix IV devices, read capture timing may fail if the DQS phase shift selected is not optimal or if the board skew specified is large.

- You can adjust the effective DQS phase shift implemented by the DLL to balance setup and hold margins on the read timing path. The DQS phase shift can be adjusted in coarse PVT-compensated steps of 22.5°, 30°, 36°, or 45° by changing the number of delay buffers (range 1 to 4), or in fine steps using the DQS phase offset feature that supports uncompensated delay addition and subtraction in approximately 12 ps steps.

- To adjust the coarse phase shift selection, determine the supported DLL modes for your chosen memory interface frequency by referencing the DLL and DQS Logic Block Specifications tables in the *Switching Characteristics* section of the device data sheet. For example, a 400 MHz DDR2 interface on a -2 speed grade device can use DLL mode 5 (resolution 36°, range 290 – 450 MHz) to implement a 72° phase shift, or DLL mode 6 (resolution 45°, range 360–560 MHz) to implement a 90° phase shift.
- You can use the **phase_shift.tcl** script and README files available for download adjacent to this document to adjust this DQS phase shift. This script only supports phase shift adjustments for ALTMEMPHY-based Stratix III and Stratix IV designs.

  To download the phase shift Tcl script, **s3_s4_phase_shift.zip**, refer to *External Memory Interfaces Design Examples*.

In Cyclone III devices, the read capture is implemented using a calibrated clock, and therefore no clock phase-shift adjustment is possible. Additionally, the capture registers are routed to specific LE registers in the logic array blocks (LABs) adjacent to the IOE using predefined routing. Therefore, no timing optimization is possible for this path. Ensure that you select the correct memory device speed grade for the FPGA speed grade and interface frequency.

Ensure that you specify the appropriate board-skew parameter when you parameterize the controllers in the parameter editor. The default board trace length mismatch used is 20 ps.

## Write Timing

Negative timing margins may be reported for write timing paths if the PLL phase shift used to generate the write data signals is not optimal. Adjust the PLL phase shift selection on the write clock PLL output using the PLL MegaWizard Plug-In Manager.

Regenerating the ALTMEMPHY- or UniPHY-based controller overwrites changes made using the PLL MegaWizard Plug-In Manager.

## Address and Command Timing

You can optimize the timing margins on the address and command timing path by changing the PLL phase shift used to generate these signals. Modify the **Dedicated Clock Phase** parameter in the **PHY Settings** page of the ALTMEMPHY parameter editor. In the DDR2 or DDR3 SDRAM Controllers with UniPHY IP cores, modify the **Additional CK/CK# phase** and **Additional Address/Command clock phase** parameters.

Use the Pin Planner feature in the Quartus II software to accurately specify the board trace model information for all your address and command and memory clock output pins. The far-end load value and board trace delay differences between address and command and memory clock pins can result in timing failures if they are not accounted for during timing analysis.

The Quartus II Fitter may not optimally set output delay chains on the address and command pins. To ensure that any PLL phase-shift adjustments are not negated by delay chain adjustments, create logic assignments using the Assignment Editor to set all address and command output pin D5 delay chains to 0.

For HardCopy III, HardCopy IV, Stratix III, and Stratix IV devices, some corner cases of device family and memory frequency may require an increase to the address and command clock phase to meet core timing. You can identify this situation, if the DDR timing report shows a `PHY setup` violation with the `phy_clk` launch clock, and the address and command latch clock—clock 0 (half-rate `phy_clk`) or 2 (full-rate `phy_clk`), and 6, respectively.

If you see this timing violation, you may fix it by advancing the address and command clock phase as required. For example, a 200-ps violation for a 400-MHz interface represents 8% of the clock period, or 28.8° . Therefore, advance the address and command phase from 270° to 300° . However, this action reduces the setup and hold margin at the DDR device.

## PHY Reset Recovery and Removal

A common cause for reset timing violations in ALTMEMPHY or UniPHY designs is the selection of a global or regional clock network for a reset signal. The ALTMEMPHY or UniPHY IP does not require any dedicated clock networks for reset signals. Only ALTMEMPHY or UniPHY PLL outputs require clock networks, and any other PHY signal using clock networks may result in timing violations.

You can correct such timing violations by:

- Setting the Global Signal logic assignment to **Off** for the problem path (using the Assignment Editor), or
- Adjusting the logic placement (using the Assignment Editor or Chip Planner)

## Clock-to-Strobe (for DDR and DDR2 SDRAM Only)

Memory output clock signals and DQS strobes are generated using the same PLL output clock. Therefore, no timing optimization is possible for this path and positive timing margins are expected for interfaces running at or below the FPGA data sheet specifications.

For DDR3 interfaces, the timing margin for this path is reported as **Write Leveling**.

## Read Resynchronization, Postamble, and Write Leveling Timing (for SDRAM Only)

These timing paths apply only to Arria II GX, Stratix III, and Stratix IV devices, and are implemented using calibrated clock signals driving dedicated IOE registers. Therefore, no timing optimization is possible for these paths, and positive timing margin is expected for interfaces running at or below the FPGA data sheet specifications.

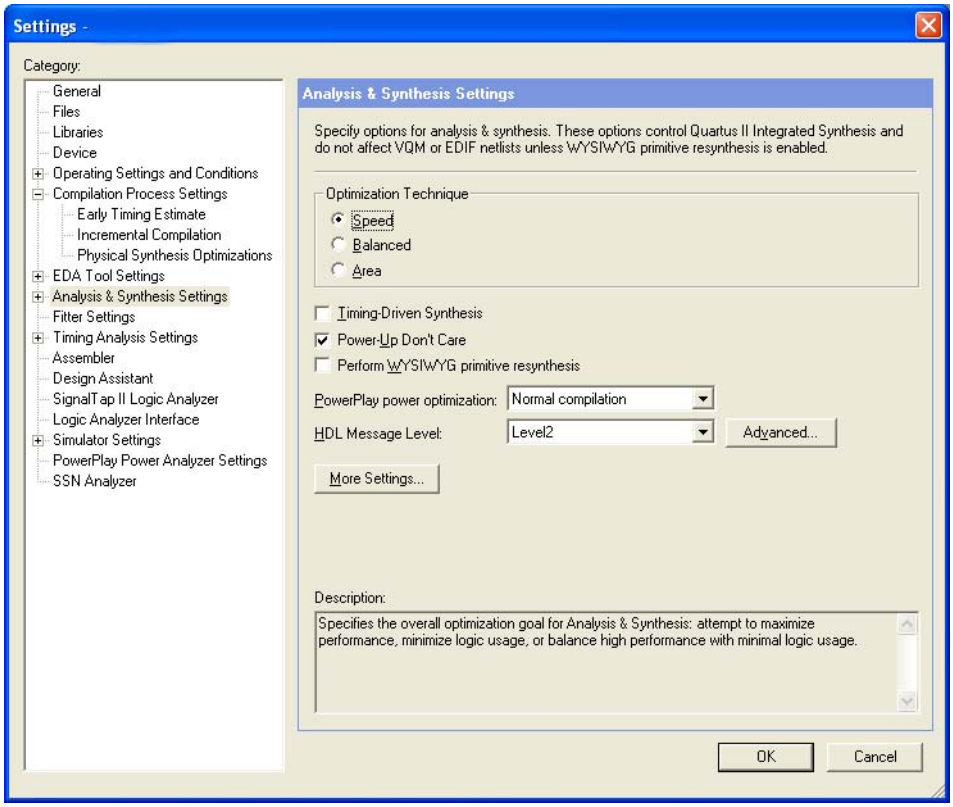
Ensure that you specify the correct memory device timing parameters ($t_{DQSCK}$, $t_{DSS}$, $t_{DSH}$) and board skew ($t_{EXT}$) in the ALTMEMPHY, DDR, DDR2, and DDR3 SDRAM Controllers with ALTMEMPHY, or DDR2 and DDR3 SDRAM Controllers with UniPHY parameter editor.

# Optimizing Timing

For full-rate designs you may need to use some of the Quartus II advanced features, to meet core timing, by following these steps:

1. On the Assignments menu click **Settings**. In the **Category** list, click **Analysis & Synthesis Settings**. For **Optimization Technique** select **Speed** (see Figure 2–1).

**Figure 2–1. Optimization Technique**



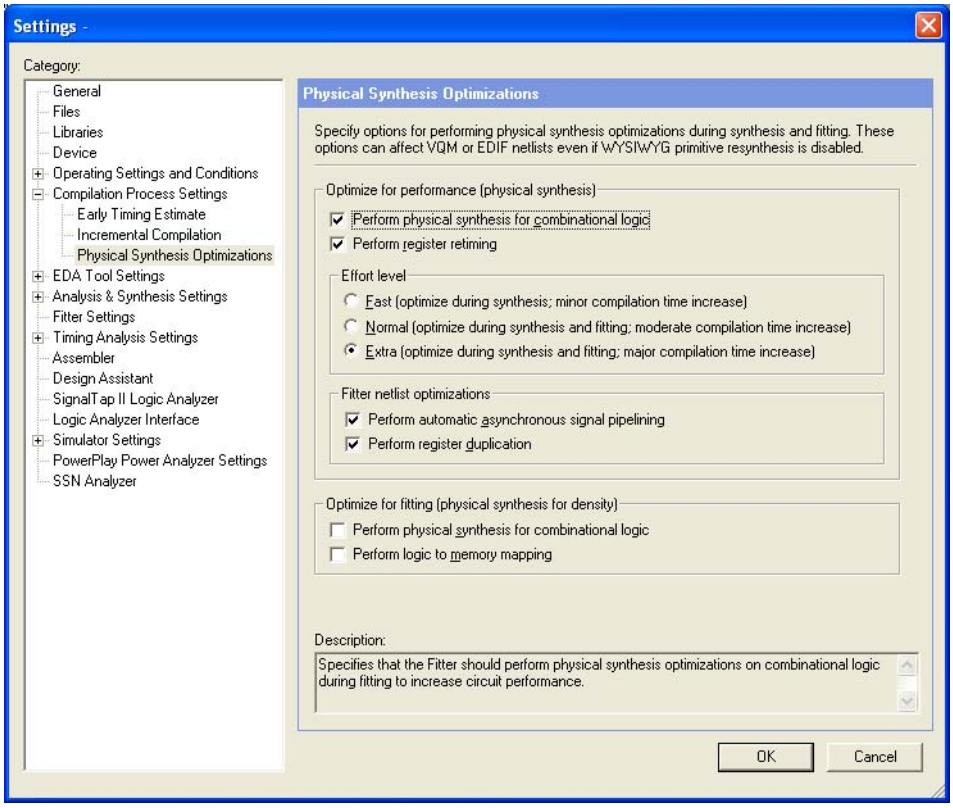

2. In the **Category** list, click **Physical Synthesis Optimizations**. Specify the following options:
   - Turn on **Perform physical synthesis for combinational logic**.

   For more information on physical synthesis, refer to the *Netlist and Optimizations and Physical Synthesis* chapter in the *Quartus II Software Handbook*.

   - Turn on **Perform register retiming**
   - Turn on **Perform automatic asynchronous signal pipelining**
   - Turn on **Perform register duplication**

You can initially select **Normal** for **Effort level** , then if the core timing is still not met, select **Extra** (see Figure 2–2).

**Figure 2–2. Physical Synthesis Optimizations**



# 3. Timing Deration Methodology for Multiple Chip Select DDR2 and DDR3 SDRAM Designs

This chapter explains two timing deration methodologies for multiple chip-select DDR2 and DDR3 SDRAM designs. For Arria II GX, Arria II GZ, Stratix IV, and Stratix V devices, the ALTMEMPHY, and ALTMEMPHY- and UniPHY-based controller parameter editors have an option to select multiple chip-select deration.

To perform multiple chip-select timing deration for other Altera devices (for example Cyclone III and Stratix III devices), Altera provides an Excel-based calculator available from the Altera web site.

Timing deration in this chapter applies to to either discrete components or DIMMs.

You can derate DDR SDRAM multiple chip select designs by using the DDR2 SDRAM section of the Excel-based calculator, but Altera does not guarantee the results.

This chapter assumes you know how to obtain data on PCB simulations for timing deration from HyperLynx or any other PCB simulator.

## Background

A DIMM contains one or several RAM chips on a small PCB with pins that connect it to another system such as a motherboard or router.

Nonregistered (unbuffered) DIMMs (or UDIMMs) connect address and control buses directly from the module interface to the DRAM on the module.

Registered DIMMs (RDIMMs) improve signal integrity (and hence potentially clock rates and overall memory size) by electrically buffering the signals with a register, at a cost of an extra clock of increased latency. In addition, most RDIMMs come with error correction coding (ECC) as standard.

Multiple chip select configurations allow for one set of data pins (and address pins for UDIMMs) to be connected to two or more memory ranks. Multiple chip select configurations have a number of effects on the timing analysis including the intersymbol interference (ISI) effects, board effects, and calibration effects.

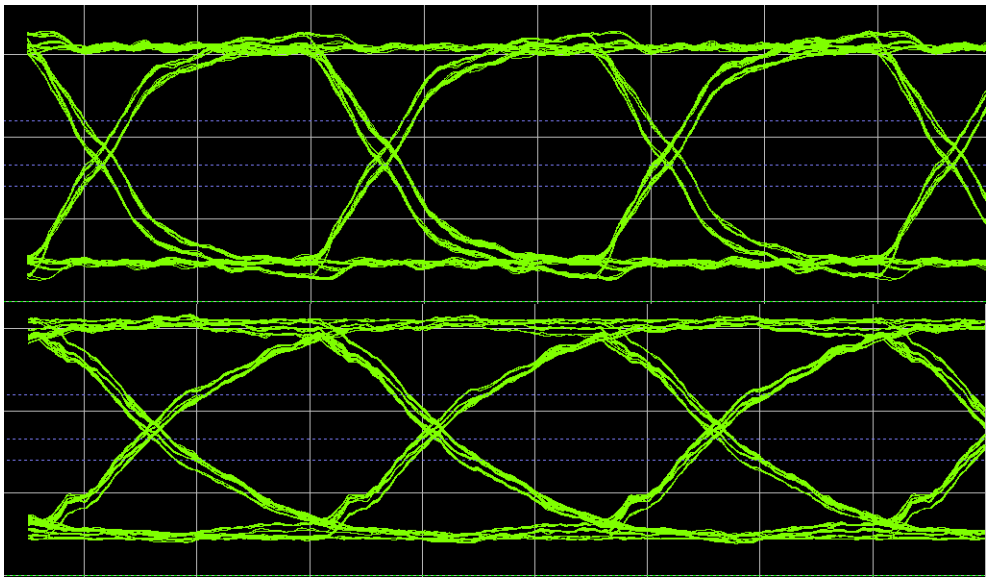
### ISI Effects

With multiple chip selects and possible slots loading the far end of the pin, there may be ISI effects on a signal causing the eye openings for DQ, DQS, and address and command signals to be smaller than for single-rank designs (Figure 3–1). Figure 3–1 shows the eye shrinkage for DQ signal of a single rank system (top) and multiple chip select system (bottom). The ISI eye reductions reduce the timing window available for both the write path and the address and command path analysis. You must specify them as output delay constraints in the Synopsis design constraints file (**.sdc**) file.

Board trace models in the Quartus II software can account for the effects for a single rank, but cannot analyze effects caused by multiple chip selects.

Extra loading from the additional ranks causes the slew rate of signals from the FPGA to be reduced. This reduction in slew rate affects some of the memory parameters including data, address, command and control setup and hold times (tDS, tDH, tIS, and tIH).

**Figure 3–1. Eye Shrinkage for DQ Signal**



## Calibration Effects

In addition to the SI effects, multiple chip select topologies change the way that the FPGA calibrates to the memories. In single-rank situations with leveling, the calibration algorithms set delay chains in the FPGA such that specific DQ and DQS pin delays from the memory are equalized (only for ALTMEMPHY-based designs at 401 MHz and above) so that the write-leveling and resynchronization timing requirements are met. In single rank without leveling situations, the calibration algorithm centers the resynchronization or capture phase such that it is optimum for the single rank. When there are two or more ranks in a system, the calibration algorithms must calibrate to the average point of the ranks.

## Board Effects

Unequal length PCB traces result in delays reducing timing margins. Furthermore, skews between different memory ranks can further reduce the timing margins in multiple chip select topologies. Board skews can also affect the extent to which the FPGA can calibrate to the different ranks. If the skew between various signals for different ranks is large enough, the timing margin on the fully calibrated paths such as write leveling and resynchronization changes.

To account for all these board effects for Arria II GX, Arria II GZ, Stratix IV, and Stratix V devices, refer to the **Board Settings** page in the ALTMEMPHY- or UniPHY-based controller parameter editors.

☞ To perform multiple chip select timing deration for other Altera devices (for example Cyclone III and Stratix III devices), use the Excel-based calculator available from the Altera web site.

# Implementing Multiple Chip Selects in the Designs

In a multiple chip select system, each individual rank has its own chip select signal. Consequently, you must change the **Total Memory chip selects**, **Number of chip select** (for discrete components) or **Number of chip select per slot** (DIMMs) in the **Preset Editor** of the ALTMEMPHY- or UniPHY-based parameter editors.

☞ In the **Preset Editor**, you must leave the baseline non-derated tDS, tDH, tIS, tIH values, because the settings on the **Board Settings** page account for multiple chip select slew rate deration.

## Timing Deration using the Board Settings

When you target Arria II GX, Arria II GZ, Stratix IV, or Stratix V devices, the ALTMEMPHY- or UniPHY-based parameter editors include the **Board Settings** page, to automatically account for the timing deration caused by the multiple chip selects in your design.

If you are targeting Cyclone III or Stratix III devices you see the following warning:

```
"Warning: Calibration performed on all chip selects, timing analysis
only performed on first chip select. Manual timing derating is
required"
```

☞ You must perform perform manual timing deration using the Excel-based calculator.

The **Board Settings** page allows you to enter the parameters related to the board design including skews, signal integrity, and slew rates. The **Board Settings** page also includes the board skew setting parameter, **Addr/Command to CK skew**, (previously on the **PHY Settings** tab).

### Slew Rates

You can obtain the slew rates in one of the following ways:

- Altera performs PCB simulations on internal Altera boards to compute the output slew rate and ISI effects of various multiple chip select configurations. These simulation numbers are prepopulated in the **Slew Rates** based on the number of ranks selected. The address and command setup and hold times (tDS, tDH, tIS, tIH) are then computed from the slew rate values and the baseline nonderated tDS, tDH, tIS, tIH numbers entered in the **Preset Editor**. The wizard shows the computed values in **Slew Rates**. If you do not have access to a simulator to obtain accurate slew rates for your specific system, Altera recommends that you use these prepopulated numbers for an approximate estimate of your actual board parameters.

- Alternatively, you can update these default values, if dedicated board simulation results are available for the slew rates. Custom slew rates cause the tDS, tDH, tIS, tIH values to be updated. Altera recommends performing board level simulations to calculate the slew rate numbers that accounts for accurate board-level effects for your board.
- You can modify the auto-calculated tDS, tDH, tIS, tIH values with more accurate dedicated results direct from the vendor data sheets, if available.

## Intersymbol Interference

ISI parameters are similarly autopopulated based on the number of ranks you select with Altera's PCB simulations. You can update these autopopulated typical values, if more accurate dedicated simulation results are available.

Altera recommends performing board-level simulations to calculate the slew rate and ISI deltas that account for accurate board level effects for your board. You can use HyperLynx or similar simulators to obtain these simulation numbers. The default values have been computed using HyperLynx simulations for Altera boards with multiple DDR2 and DDR3 SDRAM slots.

For DQ and DQS ISI there is one textbox for the total ISI, which assumes symmetric setup and hold. For address and command, there are two textboxes: one for ISI on the leading edge, and one for the lagging edge, to allow for asymmetric ISI.

The wizard writes these parameters for the slew rates and the ISI into the **.sdc** file and they are used during timing analysis.

## Board Skews

Table 3–1 describes the types of board skew.

**Table 3–1. Board Skews (Part 1 of 2)**

| Board Skew | | Description |
|---|---|---|
| Altmemphy | UniPHY | |
| Minimum CK/DQS skew to DIMM | — | The largest negative skew that exists between the CK signal and any DQS signal when arriving at any rank. This value affects the write leveling margin for DDR3 SDRAM DIMM interfaces in multiple chip select configurations only. |
| Maximum CK/DQS skew to DIMM | — | The maximum skew (or largest positive skew) between the CK signal and any DQS signal when arriving at any rank. This value affects the write leveling margin for DDR3 SDRAM DIMM interfaces in multiple chip select configurations. |
| Maximum skew between DIMMs | Maximum delay difference between DIMMs/devices | The largest skew or propagation delay between ranks (especially for different ranks in different slots). This value affects the resynchronization margin for DDR2 and DDR3 SDRAM interfaces in multiple chip select configurations. |
| Maximum skew within DQS group | Maximum skew within DQS group | The largest skew between DQ pins in a DQS group. This value affects the read capture and write margins for DDR2 and DDR3 SDRAM interfaces. |

**Table 3–1. Board Skews (Part 2 of 2)**

| Board Skew | | Description |
|---|---|---|
| **Altmemphy** | **UniPHY** | |
| Maximum skew between DQS groups | Maximum skew between DQS groups | The largest skew between DQS signals in different DQS groups. This value affects the resynchronization margin in non-leveled memory interfaces such as DDR2 and DDR3 SDRAM. |
| Address and command to CK skew | Maximum delay difference between Address/Command and CK | The skew (or propagation delay) between the CK signal and the address and command signals. Positive values represent address and command signals that are longer than CK signals; negative values represent address and command signals that are shorter than CK signals. The Quartus II software uses this skew to optimize the delay of the address and command signals to have appropriate setup and hold margins for DDR2 and DDR3 SDRAM interfaces. |
| — | Maximum skew within Address/Command bus | The largest skew between the Address/Command signals. |

## Timing Deration Using the Excel-Based Calculator

To perform multiple chip select timing deration for other Altera devices (for example Stratix III and Cyclone III devices), use the Excel-based calculator, which is available from the Altera web site.

The Excel-based calculator requires data like the slew rate, eye reduction, and the board skews of your multiple chip select system as inputs and outputs the final result based on built-in formula.

The calculator requires the Quartus II timing results (report DDR section) from the single rank version of your system. Two simulations are also required for the slew rate and ISI information required by the calculator: a baseline single rank version of your system and your multiple chip select system. The calculator uses the timing deltas of these simulation results for the signals of interest (DQ, DQS, CK/CK#, address and command, and CS). You must enter board skews for your specific board. The calculator outputs the final derated timing margins, which provides a full analysis of your multiple chip select system's performance.

The main assumption for this flow is that you have board trace models available for the single rank version of your system. If you have these board trace models, the Quartus II software automatically derates the effects for the single rank case correctly. Hence the Excel-based calculator provides the deration of the supported single-rank timing analysis, assuming that the single rank version has provided an accurate baseline.

You must ensure that the single rank board trace models are included in your Quartus II project so that the baseline timing analysis is accurate. If you do not have board trace models, follow the process described at the end of this section.

## Before You Use the Excel-based Calculator for Timing Deration

Ensure you have the following items before you use the Excel-based calculator for timing deration:

1. A Quartus II project with your instantiated memory IP. Always use the latest version of the Quartus II software, for the most accurate timing models.
2. The board trace models for the single rank version of your system.

If you do not have board trace models, refer to "Using the Excel-based Calculator for Timing Deration (Without Board Trace Models)" on page 3–9.

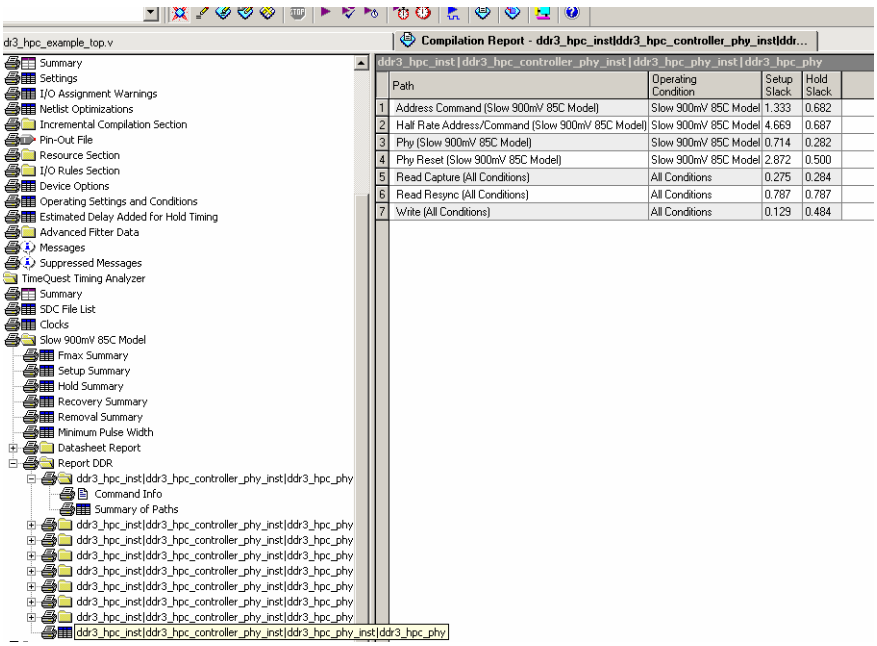
## Using the Excel-Based Calculator

To obtain derated timing margins for multiple chip select designs using the Excel-based calculator, follow these steps:

1. Create a memory interface design in the Quartus II software.
2. Ensure board trace models are specified for your single rank system. Extract Quartus II reported timing margins into the Excel-based calculator.
3. Use the slow 85C model timing results (Figure 3–2).

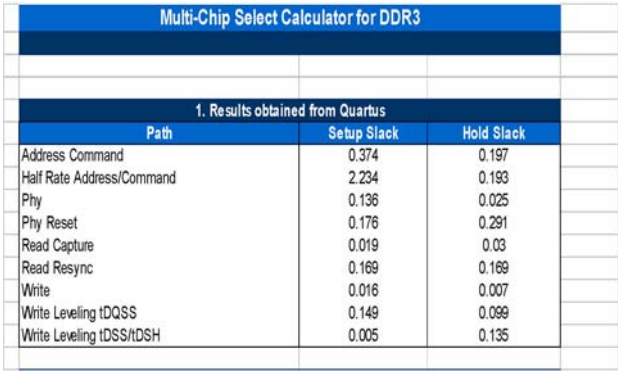
Use the worst-case values from all corners, which means that some values may come from different corners. For example, a setup value may come from the slow 85C model, while the hold value for the same metric may come from the fast 0C model. In most cases, using the slow 85C model should be accurate enough.

**Figure 3–2. Quartus II Report DDR Section Timing Results for the Slow 85C Model**



4. Enter the Report DDR results from Quartus II timing analysis into the Excel-based calculator (Figure 3–3).

**Figure 3–3. Calculator**




5. Perform PCB SI simulations to get the following values:
   - Single rank eye width and topology eye width for data, strobe, and address and command signals.
   - Multiple chip select topology slew rates for clock, address and command, and data and strobe signals.

   Table 3–2 suggests the data rates and recommended stimulus patterns for various signals and memory interface types.

   Use a simulation tool (for example, HyperLynx), if you have access to the relevant input files that the tool requires, or use prelayout line simulations if the more accurate layout information is not available.

**Table 3–2. Data Rates and Stimulus Patterns**

| Memory Interface | CLK and DQS Toggling Pattern (MHz) | DQ PRBS Pattern (MHz) | Address and Command PRBS Pattern (MHz) |
|---|---|---|---|
| DDR2 SDRAM (with a half-rate controller) | 400 | 400 | 100 |
| DDR2 SDRAM (with a full-rate controller) | 300 | 300 | 150 |
| DDR3 SDRAM (with a half-rate controller) | 533 | 533 | 133 |

6. Calculate the deltas to enter into the Excel-based calculator. For example, if DQ for the single rank case is 853.682 ps and DQ for the dual rank case is 805.137 ps, enter 48 ps into the calculator (Figure 3–4).

For signals with multiple loads, look at the measurements at all the target locations and pick the worst case eye width in all cases. For example, for the address bus, if A7 is the worst case eye width from pins A0 to A14, use that measurement for the address signal.

**Figure 3–4. ISI and Slew Rate Values**

| Intersymbol Interference | |
|---|---|
| **Path** | **Time (in ns)** |
| Address Command eye reduction on the setup | 0.013 |
| Address Command eye reduction on the hold | 0.013 |
| DQ eye reduction | 0.048 |
| Variation in DQS arrival time | 0.003 |

| Slew Rates Deration | |
|---|---|
| **Path** | **V/ns** |
| DQ | 1 |
| DQS (differential) | 2 |
| Address/Command | 1 |
| CK (differential) | 2 |
| extra tDS | 0 |
| extra tDH | 0 |
| extra tIS | 0 |
| extra tIH | 0 |

7. Enter the topology slew rates into the slew rate deration section. The calculator calculates the extra tDS, tDH, tIS, tIH values.
8. Obtain the board skew numbers for your PCB from either your board simulation or from your PCB vendor and enter them into the calculator (Figure 3–5).

**Figure 3–5. Board Skew Values**

| Board Skews | |
|---|---|
| **Path** | **Time (in ns)** |
| Maximum skew between DIMMs | 0.05 |
| Minimum CK/DQS to one DIMM | 0.01 |
| Maximum CK/DQS to one DIMM | 0.03 |

The Excel-based calculator then outputs the final derated numbers for your multiple chip select design.

**Figure 3–6. Derated Setup and Hold Values**

| Final Multi Chip Select Results | | |
|---|---|---|
| **Path** | **Setup Slack** | **Hold Slack** |
| Address Command | 0.361 | 0.184 |
| Half Rate Address/Command | 2.228 | 0.187 |
| Phy | 0.136 | 0.025 |
| Phy Reset | 0.176 | 0.291 |
| Read Capture | 0.019 | 0.030 |
| Read Resync | 0.119 | 0.119 |
| Write | -0.010 | -0.019 |
| Write Leveling tDQSS | 0.126 | 0.076 |
| Write Leveling tDSS/tDSH | -0.018 | 0.112 |

These values are an accurate calculation of your multiple chip select design timing, assuming the simulation data you provided is correct. In this example, there is negative slack on some of the paths, so this design does not pass timing. You have the following four options available:

- Try to optimize margins and see if it improves timing (for example modify address and command phase setting)
- Lower the frequency of your design
- Lower the loading (change the topology of your interface to lower the loading and skew)
- Use a faster DIMM

## Using the Excel-based Calculator for Timing Deration (Without Board Trace Models)

If board trace models are not available for any of the signals of the single rank system, follow these steps:

1. Create a new Quartus II Project with the Stratix III or Cyclone III device that you are targeting and instantiate a High-Performance SDRAM Controller for your memory interface.
2. Do not enter the board trace models (assumes a 0-pf load) and compile the Quartus II design.
3. Enter the Report DDR setup and hold slack numbers into the Excel-based calculator.
4. Perform a prelayout line simulation of a 0-pf load simulation and obtain eye width and slew rate numbers. Perform multiple chip select simulations of your topology and use the Excel-based calculator.

# External Memory Interface Handbook Volume 4

## Section III. Debugging

101 Innovation Drive
San Jose, CA 95134
www.altera.com

EMI_DEBUG_HW-2.1

Document last updated for Altera Complete Design Suite version: 10.1
Document publication date: December 2010

Subscribe

# Contents

## Chapter 3. ALTMEMPHY Calibration Stages

# 1. Verifying Functionality using the SignalTap II Logic Analyzer

The SignalTap® II logic analyzer shows read and write activity in the system.

For more information on using the SignalTap II logic analyzer, refer to *Design Debugging Using the SignalTap II Embedded Logic Analyzer* chapter in volume 3 of the *Quartus II Software Handbook*

To add the SignalTap II logic analyzer, follow these steps:

1. On the Tools menu click **SignalTap II Logic Analyzer**.
2. In the **Signal Configuration** window next to the **Clock** box, click **...** (Browse Node Finder).
3. Type the memory interface system clock (typically `*phy_clk`) in the **Named** box, for **Filter** select **SignalTap II: pre-synthesis** and click **List**.
4. Select the memory interface system clock (**<***variation name***>_example_top|<***variation name***>:<***variation name***>_inst|<***variation name***>_controller_phy:<***variation name***>_controller_phy_inst|phy_clk|phy_clk)** in **Nodes Found** and click **>** to add the signal to **Selected Nodes**.
5. Click **OK**.
6. Under Signal Configuration, specify the following settings:
   - For **Sample depth**, select **512**
   - For **RAM type**, select **Auto**
   - For **Trigger flow control**, select **Sequential**
   - For **Trigger position**, select **Center trigger position**
   - For **Trigger conditions**, select **1**
7. On the Edit menu, click **Add Nodes**.
8. Search for specific nodes by typing `*local*` in the **Named** box, for **Filter** select **SignalTap II: pre-synthesis** and click **List**.

9. Select the following nodes in **Nodes Found** and click **>** to add to **Selected Nodes**:
   - **local_address**
   - **local_rdata**
   - **local_rdata_valid**
   - **local_read_req**
   - **local_ready**
   - **local_wdata**
   - **local_wdata_req**
   - **local_write_req**
   - **pnf**
   - **pnf_per_byte**
   - **test_complete** (trigger)
   - **ctl_cal_success**
   - **ctl_cal_fail**
   - **ctl_wlat**
   - **ctl_rlat**

   ☞ Do not add any memory interface signals to the SignalTap II logic analyzer. The load on these signals increases and adversely affects the timing analysis.

10. Click **OK**.
11. To reduce the SignalTap II logic size, turn off **Trigger Enable** on the following bus signals:
    - `local_address`
    - `local_rdata`
    - `local_wdata`
    - `pnf_per_byte`
    - `ctl_wlat`
    - `ctl_rlat`
12. Right-click **Trigger Conditions** for the `test_complete` signal and select **Rising Edge**.
13. On the File menu, click **Save**, to save the SignalTap II **.stp** file to your project.

    ☞ If you see the message **Do you want to enable SignalTap II file "stp1.stp" for the current project**, click **Yes**.

14. Once you add signals to the SignalTap II logic analyzer, recompile your design, on the Processing menu, click **Start Compilation**.

15. When the design compiles, ensure that TimeQuest timing analysis passes successfully. In addition to this FPGA timing analysis, check your PCB or system SDRAM timing. To run timing analysis, run the ***_phy_report_timing.tcl** script.
    a. On the Tools menu, click **Tcl Scripts**.
    b. Select <*variation name*>**_phy_report_timing.tcl** and click **Run**.
16. Connect the development board to your computer.
17. On the Tools menu, click **SignalTap II Logic Analyzer**.
18. Add the correct <*your project name*>**.sof** file to the SOF Manager:
    a. Click **...** to open the **Select Program Files** dialog box.
    b. Select <*your project name*>**.sof**.
    c. Click **Open**.
    d. To download the file, click the **Program Device** button.
19. When the example design including SignalTap II successfully downloads to your development board, click **Run Analysis** to run once, or click **Autorun Analysis** to run continuously.

# 2. Debugging Hardware

This chapter describes the process of debugging hardware and the tools to debug any external memory interface. The concepts contained discussed can be applied to any IP but focus on the debug of issues using the Altera DDR, DDR2, DDR3, QDRII, QDRII+, and RLDRAM II IP.

Use this document with the respective IP user guide and other sections in the *External Memory Interface Handbook*.

Increases in external memory interface frequency results in the following issues that increase the challenges of debugging interfaces:

- More complex memory protocols
- Increased features and functionality
- More critical timing
- Increased complexity of calibration algorithm

Before the in-depth debugging of any issue, gather and confirm all information regarding the issue.

## Debugging Checklist

The following checklist is a good starting point when debugging an external memory interface. This chapter discusses all of the items in the checklist.

| Check | Item |
|---|---|
| ☐ | Try a different fit. |
| ☐ | Check IP parameters at the operating frequency ($t_{MRD}$, $t_{WTR}$ for example). |
| ☐ | Ensure you have constrained your design with proper timing deration and have closed timing. |
| ☐ | Simulate the design. If it fails in simulation, it will fail in hardware. |
| ☐ | Analyze timing. |
| ☐ | Place and assign $R_{UP}$ and $R_{DN}$ (OCT). |
| ☐ | Measure the power distribution network (PDN). |
| ☐ | Measure signal integrity. |
| ☐ | Measure setup and hold timing. |
| ☐ | Measure FPGA voltages. |
| ☐ | Vary voltages. |
| ☐ | Heat and cool the PCB. |
| ☐ | Operate at a lower or higher frequency. |

| Check | Item |
|---|---|
| ☐ | Check board timing and trace Information. |
| ☐ | Check LVDS and clock sources, I/O voltages and termination. |
| ☐ | Check PLL clock source, specification, and jitter. |
| ☐ | Ensure the correct number of PLL phase steps take place during calibration. If the number stated in the IP does not match the number, you may have manually altered the PLL. |
| ☐ | Retarget to a smaller interface width or a single bank. |

# Quartus II Resource and Planning Issue

Debug issues may not be directly related to interface operation. Issues can also arise at the Quartus II Fitter stage, or complex designs may have timing analysis issues.

## Quartus II Resource and Planning Issue Characteristics

Typically, single stand-alone interfaces should not present any Quartus II Fitter or timing issues. You may find that fitter, timing, and hardware operation can sometimes become a challenge, as multiple interfaces are combined into a single project, or as the device utilization increases. In such cases, the interface configuration is not the issue, the placement and total device resource requirements create problems.

## Resource Issue Evaluation

External memory interfaces typically require the following resource types, which you must consider when trying to manually place, or perhaps use additional constraints to force the placement or location of external memory interface IP:

- Dedicated IOE DQS group resources and pins
- Dedicated DLL resources
- Specific PLL resources
- Specific global, regional, and dual-regional clock net resources

### Dedicated IOE DQS Group Resources and Pins

Fitter issues can occur with even a single interface, if you do not size the interface to fit within the specified constraints and requirements. A typical requirement includes containing assignments for the interface within a single bank or possibly side of the chosen device.

Such a constraint requires that the chosen device meets the following conditions:

- Sufficient DQS groups and sizes to support the required number of common I/O (CIO) or separate I/O (SIO) data groups.
- Sufficient remaining pins to support the required number of address, command, and control pins.

Failure to evaluate these fundamental requirements can result in suboptimal interface design, if the chosen device cannot be modified. The resulting wraparound interfaces or suboptimal pseudo read and write data groups artificially limit the maximum operating frequency.

Multiple blocks of IP further complicate the issue, if other IP has either no specified location constraints or incompatible location constraints.

The Quartus II fitter may first place other components in a location required by your memory IP, then error at a later stage because of an I/O assignment conflict between the unconstrained IP and the constrained memory IP.

Your design may require that one instance of IP is placed anywhere on one side of the device, and that another instance of IP is placed at a specific location on the same side.

While the two individual instances may compile in isolation, and the physical number of pins may appear sufficient for both instances, issues can occur if the instance without placement constraints is placed before the instance with placement constraints.

In such circumstances ,Altera recommends manually placing each individual pin, or at least try using more granular placement constraints.

For more information about the pin number and DQS group capabilities of your chosen device, refer to device data sheets or the Quartus II pin planner.

## Dedicated DLL Resources

Altera devices typically use DLLs to enhance data capture at the FPGA.

While multiple external memory interfaces can usually share DLL resources, fitter issues can occur when there is insufficient planning before HDL coding. If DLL sharing is required, Altera gives the following recommendations for each instance of the IP that shares the DLL resources:

- Must have compatible DLL requirements—same frequency and mode.
- Exports its autogenerated DLL instance out of its own dedicated PHY hierarchy and into the top-level design file. This procedure allows easy comparison of the generated DLL's mode, and allows you to explicitly show the required DLL sharing between two IP blocks in the HDL

The Quartus II fitter does not dynamically merge DLL instances.

## Specific PLL Resources

When only a single interface resides on one side or one quadrant of a device, PLL resources are typically not an issue. However if multiple interfaces or IP are required on a single side or quadrant, consider the specific PLL used by each IP, and the sharing of any PLL resources.

The Quartus II software automerges PLL resources, but not for any dynamically controlled PLL components. Use the following PLL resource rules:

- Ensure that the PLL located in the same bank or side of the device is available for your memory controller.

- If multiple PLLs are required for multiple controllers that cannot be shared, ensure that enough PLL resources are available within each quadrant to support your interface number requirements.
- Try to limit multiple interfaces to a single quadrant. For example, if two complete same size interfaces can fit on a single side of the device, constrain one interface entirely in one bank of that side, and the other controller in the other bank.

For more information about using multiple PHYs or controllers, refer to *Volume 6: Design Flow Tutorials* of the *External Memory Interface Handbook*.

### Specific Global, Regional and Dual-Regional Clock Net Resources

Memory PHYs typically have specific clock resource requirements for each PLL clock output. For example because of characterization data, the PHY may require that the `phy_clk` is routed on a global clock net. The remaining clocks may all be routed on a global or a regional clock net. However, they must all be routed on the same type. Otherwise, the operating frequency of the interface is lowered, because of the increased uncertainty between two different types of clock nets. The design may still fit, but not meet timing.

## Planning Issue Evaluation

Plan the total number of DQS groups and total number of other pins required in your shared area. Use the Pin Planner to assist with this activity.

Decide which PLLs or clock networks can be shared between IP blocks, then ensure that sufficient resources are available. For example, if an IP core requires a regional clock network, a PLL located on the opposite side of the device cannot be used.

Calculate the number of total clock networks and types required when trying to combine multiple instances of IP.

You must understand the number of quadrants that the IP uses and if this number can be reduced. For example, an interface may be autoplaced across an entire side of the device, but may actually be constrained to fit in a single bank.

Optimizing the physical placement ensures that when possible, regional clock networks are used as opposed to dual-regional clock networks, hence clock net resources are maintained and routing is simplified.

As device utilization increases, the Quartus II software may have difficulty placing the core. To optimize design utilization, follow these steps:

- Review any fitter warning messages in multiple IP designs to ensure that clock networks or PLL modes are not modified to achieve the desired fit.
- Use the Quartus II Fitter resource section to compare the types of resources used in a successful standalone IP implementation to those used in an unreliable multiple IP implementation.
- Use this information to better constrain the project to achieve the same results as the standalone project.
- Use the Chip Planner (Floorplan and Chip Editor) to compare the placement of the working stand-alone design to the multiple IP project. Then use LogicLock or Design Partitions to better guide the Quartus II software to the required results.

- When creating LogicLock regions, ensure that they encompass all required resources. For example, if constraining the read and write datapath hierarchy, ensure that your LogicLock region includes the IOE blocks used for your datapath pin out.

# Performance, Efficiency, and Bottleneck Issues

This topic describes the performance ($f_{MAX}$), efficiency (latency and transaction efficiency) and bottleneck (the limiting factor) issues that you can encounter in any design.

## Performance Issues

There are a large number of interface combinations and configurations possible in an Altera design, therefore it is impractical for Altera to explicitly state the achievable $f_{MAX}$ for every combination. Altera seeks to provide guidance on typical performance, but this data is subject to memory component timing characteristics, interface widths, depths directly affecting timing deration requirements, and the achieved skew and timing numbers for a specific PCB.

FPGA timing issues should generally not be affected by interface loading or layout characteristics. In general, the Altera performance figures for any given device family and speed-grade combination should usually be achievable.

To resolve FPGA (PHY and PHY reset) timing issues, refer to the *Timing Analysis* section in volume 4 of the *External Memory Interface Handbook*.

Achievable interface timing (address and command, half-rate address and command, read and write capture) is directly affected by any layout issues (skew), loading issues (deration), signal integrity issues (crosstalk timing deration), and component speed grades (memory timing size and tolerance). Altera performance figures are typically stated for the default (single rank, unbuffered DIMM) case. Altera provides additional expected performance data where possible, but the $f_{MAX}$ is not achievable in all configurations. Altera recommends that you optimize the following items whenever interface timing issues occur:

- Improve PCB layout tolerances
- Use a faster speed grade of memory component
- Ensure that the interface is fully and correctly terminated
- Reduce the loading (reduce the deration factor)

## Bottleneck and Efficiency Issues

Depending on the transaction types, efficiency issues can exist where the achieved data rate is lower than expected. Ideally, these issues should be assessed and resolved during the simulation stage because they are sometimes impossible to solve later without rearchitecting the product.

Any interface has a maximum theoretical data rate derived from the clock frequency, however, in practise this theoretical data rate can never be achieved continuously due to protocol overhead and bus turnaround times.

Simulate your desired configuration to ensure that you have specified a suitable external memory family and that your chosen controller configuration can achieve your required bandwidth.

Efficiency can be assessed in several different ways, and the primary requirement is an achievable continuous data rate. The local interface signals combined with the memory interface signals and a command decode trace should provide adequate visibility of the operation of the IP to understand whether your required data rate is sufficient and the cause of the efficiency issue.

To show if under ideal conditions the required data rate is possible in the chosen technology, follow these steps:

1. Use the memory vendors own testbench and your own transaction engine.
2. Use either your own driver, or modify the provided example driver, to replicate the transaction types typical of your system.
3. Simulate this performance using your chosen memory controller and decide if the achieved performance is still acceptable.

Observe the following points that may cause efficiency or bottleneck issues at this stage:

- Identify the memory controller rate (full, half, or quarter) and commands, which may take two or four times longer than necessary
- Determine whether the memory controller is starved for data by observing the appropriate request signals.
- Determine whether the memory controller processor transactions at a rate sufficient to meet throughput requirements by observing appropriate signals, including the local ready signal.

Altera has several versions and types of memory controller, and where possible you can evaluate different configurations based on the results of the first tests.

Consider using either a faster interface, or a different memory type to better align your data rate requirements to the IP available directly from Altera.

Altera also provides stand-alone PHY configurations so that you may develop custom controllers or use third-party controllers designed specifically for your requirements.

# Functional Issues

This topic discusses functional issues.

## Functional Issue Characteristics

Functional issues occur at all frequencies (using the same conditions) and are not altered by speed grade, temperature, or PCB changes.

## Functional Issue Evaluation

Functional issues should be evaluated using functional simulation.

The Altera IP includes the option to autogenerate a testbench specific to your IP configuration, which provides an easy route to functional verification.

The following issues should be considered when trying to debug functional issues in a simulation environment.

### Correct Combination of the Quartus II Software and ModelSim-Altera Device Models

When running any simulations, ensure that you are using the correct combination of the Quartus II software and device models. Altera only tests each release of software and IP with the aligned release of device models. Failure to use the correct RTL and model combination may result in unstable simulation environments.

The ModelSim-Altera edition of the ModelSim simulator comes precompiled with the Altera device family libraries included. You must always install the correct release of ModelSim-Altera to align with your Quartus II software and IP release.

If you are using a full version of ModelSim-SE or PE, or any other supported simulation environment, ensure that you are compiling the current Quartus II supplied libraries. These libraries are located in the <*Quartus II install path*>**/quartus/eda/sim_lib/** directory.

### Altera IP Memory Model

Altera memory IP autogenerates a generic simplified memory model that works in all cases. This simple read and write model is not designed or intended to verify all entered IP parameters or transaction requirements.

The Altera-generated memory model may be suitable to evaluate some limited functional issues, but it does not provide comprehensive functional simulation.

### Vendor Memory Model

Contact the memory vendor directly as many additional models are available from the vendors support system.

When using memory vendor models, ensure that the model is correctly defined for the following characteristics:

- Speed grade
- Organization
- Memory allocation
- Maximum memory usage
- Number of ranks on a DIMM
- Buffering on the DIMM
- ECC

Refer to the **readme.txt** file supplied with the memory vendor model, for more information about how to define this information for your configuration.

During simulation vendor models output a wealth of information regarding any device violations that may occur because of incorrectly parameterized IP.

Refer to Transcript Window Messages, for more information.

## Out of PC Memory Issues

If you are running the ModelSim-Altera simulator, the limitation on memory size, may mean that you have insufficient memory to run your simulation. Or, if you are using a 32-bit operating system, your PC may have insufficient memory.

Typical simulation tool errors include: "Iteration limit reached" or "Error out of memory".

When using either the Altera generic memory model, or a vendor specific model quite large memory depths can be required if you do not specify your simulation carefully.

For example, if you simulate an entire 4-GB DIMM interface, the hardware platform that performs that simulation requires at least this amount of memory just for the simulation contents storage.

Refer to Memory Allocation and Max Memory Usage in the vendor's **readme.txt** files for more information.

## Transcript Window Messages

When debugging a functional issue in simulation, vendor models typically provide a much more detailed checks and feedback regarding the interface and their operational requirements than the Altera generic model.

In general, you should use a vendor-supplied model whenever one is available. Consider using second-source vendor models in preference to the Altera generic model.

Many issues can be traced to incorrectly configured IP for the specified memory components. Component data sheets usually contain settings information for several different speed grades of memory. Be aware data sheet specify parameters in fixed units of time, frequencies, or clock cycles.

The Altera generic memory model always matches the parameters specified in the IP, as it is generated using the same engine. Because vendor models are independent of the IP generation process, they offer a more robust IP parameterization check.

During simulation, review the transcript window messages and do not rely on the Simulation Passed message at the end of simulation. This message only indicates that the example driver successfully wrote and then read the correct data for a single test cycle.

Even if the interface functionally passes in simulation, the vendor model may report operational violations in the transcript window. These reported violations often specifically explain why an interface appears to pass in simulation, but fails in hardware.

Vendor models typically perform checks to ensure that the following types of parameters are correct:

- Burst length
- Burst order
- tMRD
- tMOD
- tRFC

- tREFPDEN
- tRP
- tRAS
- tRC
- tACTPDEN
- tWR
- tWRPDEN
- tRTP
- tRDPDEN
- tINIT
- tXPDLL
- tCKE
- tRRD
- tCCD
- tWTR
- tXPR
- PRECHARGE
- CAS length
- Drive strength
- AL
- tDQS
- CAS_WL
- Refresh
- Initialization
- tIH
- tIS
- tDH
- tDS

If a vendor model can verify all these parameters are compatible with your chosen component values and transactions, it provides a specific insight into hardware inteface failures.

## Simulation

Passing simulation means that the interface calibrates and successfully completes a single test complete cycle without asserting pass not fail (`pnf`). It does not take into account any warning messages that you may receive during simulation. If you are debugging an interface issue, review and, if necessary, correct any warning messages from the transcript window before continuing.

### Modifying the Example Driver to Replicate the Failure

Often during debugging, you may discover that the example driver design works successfully, but that your custom logic is observing data errors. This information indicates that the issue is either:

- Related to the way that the local interface transactions are occurring. Altera recommends you probe and compare using the SignalTap II analyzer.
- Related to the types or format of transactions on the external memory interface. Altera recommends you modify the example design to replicate the issue.

Typical issues on the local interface side include:

- Incorrect local address to memory address translation causing the word order to be different than expected. Refer to *Burst Definition* in your memory vendor data sheet.
- Incorrect timing on the local interface. When your design requests a transaction, the local side must be ready to service that transaction as soon as it is accepted without any pause.


For more information, refer to the *Avalon Interface Specification*.

The default example driver only performs a limited set of transaction types, consequently potential bus contention or preamble and postamble issues can often be masked in its default operation. For successful debugging, isolate the custom logic transaction types that are causing the read and write failures and modify the example driver to demonstrate the same issue. Then, you can try to replicate the failure in RTL simulation with the modified driver.

An issue that you can replicate in RTL simulation indicates a potential bug in the IP. You should recheck the IP parameters. An issue that you can not replicate in RTL simulation indicates a timing issue on the PCB. You can try to replicate the issue on an Altera development platform to rule out a board issue.


Ensure that all PCB timing, loading, skew, and deration information is correctly defined in the Quartus II software, as the timing report is inaccurate if this initial data is not correct.

Functional simulation allows you to identify any issues with the configuration of either the Altera memory controller and or PHY. You can then easily check the operation against both the memory vendor data sheet and the respective Jedec specification. After you resolve functional issues, you can start testing hardware.


For more information about simulating, refer to the *Simulation* section in volume 4 of the *External Memory Interface Handbook*.

## Timing Issues

The Altera PHY and controller combinations autogenerate timing constraint files to ensure that the PHY and external interface are fully constrained and that timing is analyzed during compilation. However, timing issues can still occur. This topic discusses how to identify and resolve any timing issues that you may encounter.

## Timing Issue Characteristics

Timing issues typically fall into two distinct categories:

- FPGA core timing reported issues
- External memory interface timing issues in a specific mode of operation or on a specific PCB

TimeQuest reports timing issues in two categories: core to core and core to IOE transfers. These timing issues include the PHY and PHY reset sections in the TimeQuest Report DDR subsection of timing analysis. External memory interface timing issues are specifically reported in the TimeQuest Report DDR subsection, excluding the PHY and PHY reset. The Report DDR PHY and PHY reset sections only include the PHY, and specifically exclude the controller, core, PHY-to-controller and local interface. Quartus II timing issues should always be evaluated and corrected before proceeding to any hardware testing.

PCB timing issues are usually Quartus II timing issues, which are not reported in the Quartus II software, if incorrect or insufficient PCB topology and layout information is not supplied. PCB timing issues are typically characterized by calibration failure, or failures during user mode when the hardware is heated or cooled. Further PCB timing issues are typically hidden if the interface frequency is lowered.

## Timing Issue Evaluation

Try to fix and identify timing issues in the Quartus II software. Consider the following issues when resolving timing issues.

### FPGA Timing Issues

In general, you should not have any timing issues with Altera-provided IP unless you running the IP outside of Altera's published performance range or are using a version of the Quartus II software with preliminary timing model support for new devices. However, timing issues can occur in the following circumstances:

- The **.sdc** files are incorrectly added to the Quartus II project
- Quartus II analysis and synthesis settings are not correct
- Quartus II Fitter settings are not correct

For all of these issues, refer to the correct user guide for more information about recommended settings and follow these steps:.

1. Ensure that the IP generated **.sdc** files are listed in the Quartus II TimeQuest Timing Analyzer files to include in the project window.
2. Ensure that **Analysis and Synthesis Settings** are set to **Optimization Technique Speed**.
3. Ensure that **Fitter Settings** are set to **Fitter Effort Standard Fit**.
4. Use **TimeQuest Report Ignored Constraints**, to ensure that **.sdc** files are successfully applied.
5. Use **TimeQuest Report Unconstrained Paths**, to ensure that all critical paths are correctly constrained.

More complex timing issues can occur if any of the following conditions are true:

- The design includes multiple PHY or core projects
- Devices where the resources are heavily used
- The desing includes wide, distributed, maximum performance interfaces in large die sizes

Any of these conditions can lead to suboptimal placement results when the PHY or controller are distributed around the FPGA. To evaluate such issues, simplify the design to just the autogenerated example top-level file and determine if the core meets timing and you see a working interface. Failure implies that a more fundamental timing issue exists. If the standalone design passes core timing, evaluate how this placement and fit is different than your complete design.

Use LogicLock regions, or design partitions to better define the placement of your memory controllers. When you have your interface standalone placement, repeat for additional interfaces, combine, and finally add the rest of your design.

Additionally, use fitter seeds and increase the placement and router effort multiplier.

## External Memory Interface Timing Issues

External memory interface timing issues are not directly related to the FPGA timing but are actually derived from the FPGA input and output characteristics, PCB timing, and the memory component characteristics.

The FPGA input and output characteristics tend to be a predominately fixed value, as the IOE structure of the devices is fixed. Optimal PLL characteristics and clock routing characteristics do have an effect. Assuming the IP is correctly constrained with the autogenerated assignments, and you follow implementation rules, the design should reach the stated performance figures.

The memory component characteristics are fixed for any given component or DIMM. However, consider using faster components or DIMMs in marginal cases when PCB skew may be suboptimal, or your design includes multiple ranks when deration may be causing read capture or write timing challenges. Using faster memory components typically reduces the memory data output skew and uncertainty easing read capture, and lowering the memory's input setup and hold requirement, which eases write timing.

Increased PCB skew reduces margins on address, command, read capture and write timing. If you are narrowly failing timing on these paths, consider reducing the board skew (if possible), or using faster memory. Address and command timing typically requires you to manually balance the reported setup and hold values with the dedicated address and command phase in the IP.

Refer to the respective IP user guide for more information.

Multiple-slot multiple-rank UDIMM interfaces can place considerable loading on the FPGA driver. Typically a quad rank interface can have thirty-six loads. In multiple-rank configurations, Altera's stated maximum data rates are not likely to be achievable because of loading deration. Consider using different topologies, for example registered DIMMs, so that the loading is reduced.

Deration because of increased loading, or suboptimal layout may result in a lower than desired operating frequency meeting timing. You should close timing in the Quartus II software using your expected loading and layout rules before committing to PCB fabrication.

Ensure that any design with an Altera PHY is correctly constrained and meets timing in the Quartus II software. You must address any constraint or timing failures before testing hardware.

For more information about constraints, refer to *Volume 3: Implementing Altera Memory Interface IP* of the *External Memory Interface Handbook*.

# Hardware Debugging in the Laboratory

Before starting to debug, confirm the design followed the Altera recommended design flow.

Refer to the *Recommended Design Flow* section in volume 1 of the *External Memory Interface Handbook*.

Always keep a record of tests, to avoid repeating the same tests later. To start debugging the design, perform the following initial steps.

## Create a Simplified Design that Demonstrates the Same Issue

To help debugging create a simple design that replicates the issue. A simple design compiles faster and is much easier to understand. Altera's external memory interface IP generates an example top-level file that is ideal for debugging. The example top-level file uses all the same parameters, pin-outs, and so on.

## Measure Power Distribution Network

Ensure you take measurements of the various power supplies on their hardware development platform over a suitable time base and with a suitable trigger using an appropriate probe and grounding scheme. In addition, take the measurements directly on the pins or vias of the devices in question, and with the hardware operational.

## Measure Signal Integrity and Setup and Hold Margin

Measure the signals on their PCB to ensure that everything looks correct. This information can be vital. When measuring any signal, consider the edge rate of the signal, not just its frequency. Modern FPGA devices have very fast edge rates, therefore you must use a suitable oscilloscope, probe, and grounding scheme when you measure the signals.

You can take measurements to capture the setup and hold time of key signal classes with respect to their clock or strobe. Ensure that the measured setup and hold margin is at least better than that reported in the Quartus II software. A worse margin indicates that a timing discrepancy exists somewhere in the project. However, this timing issue may not be the cause of your problem.

## Vary Voltage

Try and vary the voltage of your system, if you suspect a marginality issue. Increasing the voltage typically causes devices to operate faster and also usually provides increased noise margin.

## Use Freezer Spray and Heat Gun

If you have an intermittent marginal issue, cool or heat the interface to try and stress the issue. Cooling down ICs causes them to run faster, which makes timing easier. Conversely heating up ICs causes them to run slower, which makes timing more difficult.

If cooling or heating fixes the issue, you are probably looking at a timing issue.

## Operate at a Lower Speed

Test the interface at a lower speed. If the interface works, the interface is correctly pinned out and functional. However, if the interface fails at a lower speed, determine if the test is valid. Many high-speed memory components have a minimal operating frequency, or require subtly different configurations when operating at a lower speeds.

For example, DDR, DDR2, or DDR3 SDRAM typically requires modification to the following parameters if you want operate the interface a lower speeds:

- $t_{MRD}$
- $t_{WTR}$
- CAS latency and CAS write latency

## Find Out if the Issue Exists in Previous Versions of Software

Hardware that works before an update to either the Quartus II software or the memory IP indicates that the development platform is not the issue. However, the previous generation IP may be less susceptible to a PCB issue, masking the issue.

## Find out if the Issue Exists in the Current Version of Software

Designs are often tested using previous generations of Altera software or IP. Projects do not always get upgraded for the following reasons:

- Multiple engineers are on the same project. To ensure compatibility, a common release of Altera software is used by all engineers for the duration of the product development. The design may be several releases behind the current Quartus II software version.
- Many companies delay before adopting a new release of software so that they can first monitor Internet forums to get a feel for how successful other users say the software is.
- Many companies never use the latest version of any software, preferring to wait until the first service pack is released that fixes the primary issues.

- Some users may only have a license for the older version of the software and can only use that version until their company makes the financial decision to upgrade.
- The local interface specification from Altera IP to the customer's logic sometimes changes from software release to software release. If you have already spent resources designing interface logic, you may be reluctant to repeat this exercise. If a block of code is already signed off, you may be reluctant to modify it to upgrade to newer IP from Altera..

In all of these scenarios, you must determine if the issue still exists in the latest version of the Altera software. Bugs are fixed and enhancements are added to the Altera IP every release. Depending on the nature of the bug or enhancement, it may not always be clearly documented in the release notes.

Finally, if the latest version of the software resolves the issue, it may be easier to debug the version of software that you are using.

### Try A Different PCB

If you are using the same Altera IP on a number of different hardware platforms; find out if the issue occurs on all of these hardware platforms, or just one. Multiple instances of the same PCB, or multiple instances of the same interface, on physically different hardware platforms may exhibit different behavior. You can determine if the configuration is fundamentally not working, or if some form of marginality is involved in the issue.

Issues are often reported on the alpha build of a development platform. These are produced in very limited numbers and often have received limited BBT (bare board testing), or FT (functional testing). Hence, these early boards are often more unreliable than production quality PCBs.

Additionally, if the IP is from a previous project to help save development resources, find out if this specific IP configuration works on a previous platform.

### Try Other Configurations

Designs are typically quite large, using multiple blocks of IP in many different combinations. Find out if any other configurations work correctly on the development platform. The full project may have multiple external memory controllers in the same device, or may have configurations where only half the memory width or frequency is required. Find out what does and does not work to help the debugging of the issue.

## Catagorizing Hardware Issues

The following topic catagorizises issues. Identifying which category or groups of category an issue may be classified within allows you to focus on the cause of the issue.

# Signal Integrity Issues

Many design issues, even ones that you find at the protocol layer, can often be traced back to signal integrity issues. Hence, you must check circuit board construction, power systems, command, and data signaling to determine if they meet specifications. If infrequent, random errors exist in the memory subsystem, product reliability suffers. As such, electrical verification is vital. Check the bare circuit board or PCB design file. Circuit board errors can cause poor signal integrity, signal loss, signal timing skew, and trace impedance mismatches. Differential traces with unbalanced lengths or signals that are routed too closely together can cause crosstalk.

## Characteristics

Signal integrity issues often appear when the performance of the hardware design is marginal. The design may not always initialize and calibrate correctly, or may exhibit occasional bit errors in user mode. Severe signal integrity issues can result in total failure of an interface at certain data rates, and sporadic component failure because of electrical stress. PCB component variance and signal integrity issues often show up as failures on one PCB, but not on another identical board. Timing issues can have a similar characteristic. Multiple calibration windows or significant differences in the calibration results from one calibration to another can also indicate signal integrity issues.

## Evaluating Signal Integrity Issues

Signal integrity issues can only really be evaluated in two ways, direct measurement using suitable test equipment like an oscilloscope and probe, or simulation using a tool like HyperLynx or Allegro PCB SI. Signals should be compared against the respective electrical specification. You should look for overshoot and undershoot, non-monotonicity, eye height and width, and crosstalk.

### Skew

Ensure that all clocked signals, commands, addresses, and control signals arrive at the memory inputs at the same time. Trace length variations cause data valid window variations between the signals reducing margin. For example, DDR2-800 at 400 MHz has a data valid window that is smaller than 1,250 ps. Trace length skew or crosstalk can reduce this data valid window further, making it difficult to design a reliably operating memory interface. Ensure that the skew figure previously entered into the Altera IP matches that actually achieved on the PCB, otherwise Quartus II timing analysis of the interface is accurate.

### Crosstalk

Crosstalk is another issue that is best evaluated early in the memory design phase. Check the clock-to-data strobes, as these are bi-directional. Measure the crosstalk at both ends of the line. Check the data strobes to clock, as the clocks are unidirectional, these only need checking at the memory end of the line.

### Power System

Some memory interfaces tend to draw current in spikes from their power delivery system as SDRAMs are based on capacitive memory cells. Rows are read and refreshed one at a time, which causes dynamic currents that can stress any power distribution network (PDN). The various power rails should be checked either at or as close as possible to the SDRAM pins power pins. Ideally, a real-time oscilloscope set to fast glitch triggering should be used for this activity.

### Clock Signals

The clock signal quality is important for any external memory system. Measurements include frequency, digital core design (DCD), high width, low width, amplitude, jitter, rise, and fall times.

### Read Data Valid Window and Eye Diagram

The memory generates the read signals. Take measurements at the FPGA end of the line. To ease read diagram capture, modify the example driver to mask writes or modify the PHY to include a signal that you can trigger on when performing reads.

### Write Data Valid Window and Eye Diagram

The FPGA generates the write signals. Take measurements at the memory device end of the line. To ease write diagram capture, modify the example driver to mask reads or modify the PHY export a signal that is asserted when performing writes.

### OCT and ODT Usage

Modern external memory interface designs typically use OCT for the FPGA end of the line, and ODT for the memory component end of the line. If either the OCT or ODT are incorrectly configured or enabled, signal integrity issues exist. If the design is using OCT, $R_{UP}$ or $R_{DN}$ pins must be placed correctly for the OCT to work. If you do not place these pins, the Quartus II software allocates them automatically with the following warning:

```
Warning: No exact pin location assignment(s) for 2 pins of 110 total pins

Info: Pin termination_blk0~_rup_pad not assigned to an exact location
on the device

Info: Pin termination_blk0~_rdn_pad not assigned to an exact location
on the device
```

If you see these warnings, the $R_{UP}$ and $R_{DN}$ pins may have been allocated to a pin that does not have the required external resistor present on the board. This allocation renders the OCT circuit faulty, resulting in unreliable UniPHY and ALTMEMPHY calibration and or interface behavior. The pins with the required external resistor must be specified in the Quartus II software.

For the FPGA, ensure that follow these actions:

- Specify the $R_{UP}$ and $R_{DN}$ pins in either the projects HDL port list, or in the assignment editor (`termination_blk0~_rup_pad/ termination_blk0~_rdn_pad`).
- Connect the $R_{UP}$ and $R_{DN}$ pins to the correct resistors and pull-up and pull-down voltage in the schematic or PCB.
- Contain the $R_{UP}$ and $R_{DN}$ pins within a bank of the device that is operating at the same VCCIO voltage as the interface that is terminated.

- Check that only the expected number of $R_{UP}$ and $R_{DN}$ pins exists in the project pin-out file. Look for `Info: Created on-chip termination` messages at the fitter stage for any calibration blocks not expected in your design.
- Review the Fitter Pin-Out file for $R_{UP}$ and $R_{DN}$ pins to ensure that they are on the correct pins, and that only the correct number of calibration blocks exists in your design.
- Check in the fitter report that the input, output, and bidirectional signals with calibrated OCT all have the termination control block applicable to the associated $R_{UP}$ and $R_{DN}$ pins.

For the memory components, ensure that you follow these actions:

- Connect the required resistor to the correct pin on each and every component, and ensure that it is pulled to the correct voltage.
- Place the required resistor close to the memory component.
- Correctly configure the IP to enable the desired termination at initialization time.
- Check that the speed grade of memory component supports the selected ODT setting.
- Check that the second source part that may have been fitted to the PCB, supports the same ODT settings as the original

## Hardware and Calibration Issues

When you resolve functional, timing, and signal integrity issues, assess the operation of the PHY and its interface calibration.

### Hardware and Calibration Issue Characteristics

Hardware and calibration issues have the following definitions:

- Calibration issues result in calibration failing, which typically results in the design asserting the `ctl_cal_fail` signal.
- Hardware issues result in read and write failures, which typically results in the design asserting the pass not fail (`pnf`) signal

☞ Ensure that functional, timing, and signal integrity issues are not the direct cause of your hardware issue, as functional, timing or signal integrity issues are usually the cause of any hardware issue.

### Evaluating Hardware and Calibration Issues

Use the following methods to evaluate hardware and calibration issues:

- Evaluate hardware issues using the SignalTap II logic analyzer to monitor the local side read and write interface with the pass or fail or error signals as triggers
- Evaluate calibration issues using the SignalTap II logic analyzer to monitor the various calibration, configuration with the pass or fail or error signals as triggers, but also use the debug toolkit and system consoles when available

Refer to Chapter 4, Debug Toolkit for DDR2 and DDR3 SDRAM Controllers with ALTMEMPHY IP.

Refer to the respective user guide for information on which signals you should use during debugging. Consider adding core noise to your design to aggravate margin timing and signal integrity issues. Steadily increase the stress on the interface in the following order:

1. Increase the interface utilization by modifying the example driver to focus on the types of transactions that exhibit the issue.
2. Increase the SNN or aggressiveness of the data pattern by modifying the example driver to output in synchronization PRBS data patterns, or hammer patterns.
3. Increase the stress on the PDN by adding more and more core noise to your system. Try sweeping the fundamental frequency of the core noise to help identify resonances in your power system.

Steadily increasing the stress on the external memory interface is an ideal way to assess and understand the cause of any previously intermittent failures that you may observe in your system. Using the SignalTap II probe tool can provide insights into the source or cause of operational failure in the system.

Additionally, steadily increasing stress on the external memory interface allows you to assess and understand the impact that such factors have on the amount of timing margin and resynchronization window. Take measurements with and without the additional stress factor to allow evaluation of the overall effect.

### Write Timing Margin

Determine the write timing margin by phase sweeping the write clock from the PLL. Use sources and probes to dynamically control the PLL phase offset control, to increase and decrease the write clock phase adjustment so that the write window size may be ascertained.

Remember that when sweeping PLL clock phases, the following two factors may cause operational failure:

- The available write margin.
- The PLL phase in a multi-clock system.

The following code achieves this adjustment. You should use sources and probes to modify the respective output of the PLL. Ensure that the example driver is writing and reading from the memory while observing the `pnf_per_byte` signals to see when write failures occur:

```
//////////////////
wire [7:0] Probe_sig;
wire [5:0] Source_sig;
PhaseCount PhaseCounter (
 .resetn (1'b1),
 .clock (pll_ref_clk),
 .step (Source_sig[5]),
 .updown (Source_sig[4]),
 .offset (Probe_sig)
 );
CheckoutPandS freq_PandS (
```

```
 .probe (Probe_sig),
 .source (Source_sig)
 );
ddr2_dimm_phy_alt_mem_phy_pll_siii pll (
 .inclk0 (pll_ref_clk),
 .areset (pll_reset),
 .c0 (phy_clk_1x), // hR
.c1 (mem_clk_2x), // FR
.c2 (aux_clk), // FR
 .c3 (write_clk_2x), // FR
 .c4 (resync_clk_2x), // FR
 .c5 (measure_clk_1x), // hR
 .c6 (ac_clk_1x), // hR
 .phasecounterselect (Source_sig[3:0]),
 .phasestep (Source_sig[5]),
 .phaseupdown (Source_sig[4]),
 .scanclk (scan_clk),
 .locked (pll_locked_src),
 .phasedone (pll_phase_done)
 );
```

### Read Timing Margin

Similarly, assess the read timing margin by using sources and probes to manually control the DLL phase offset feature. Open the autogenerated DLL using ALT_DLL and add the additionally required offset control ports. This action allows control and observation of the following signals:

```
dll_delayctrlout[5:0], // Phase output control from DLL to DQS pins
(Gray Coded)

dll_offset_ctrl_a_addnsub, // Input add or subtract the phase offset
value

dll_offset_ctrl_a_offset[5:0], // User Input controlled DLL offset
value (Gray Coded)

dll_aload, // User Input DLL load command

dll_dqsupdate, // DLL Output update required signal.
```

In examples where the applied offset applied results in the maximum or minimum `dll_delayctrlout[5:0]` setting without reaching the end of the read capture window, regenerate the DLL in the next available phase setting, so that the full capture window is assessed.

Modify the example driver to constantly perform reads (mask writes). Observe the `pnf_per_byte` signals while the DLL capture phase is manually modified to see when failures begin, which indicates the edge of the window.

A resynchronization timing failure can indicate failure at that capture phase, and not a capture failure. You should recalibrate the PHY with the calculated phase offset to ensure that you are using the true read-capture margin.

#### Address and Command Timing Margin

You set the address and command clock phase directly in the IP. Assuming you enter the correct board trace model information into the Quartus II software, the timing analysis should be correct. However, if you want to evaluate the address and command timing margin, use the same process as in "Write Timing Margin", only phase step the address and command PLL output (c6 `ac_clk_1x`). You can achieve this effect using the debug toolkit or system console.

Refer to Chapter 4, Debug Toolkit for DDR2 and DDR3 SDRAM Controllers with ALTMEMPHY IP.

#### Resynchronization Timing Margin

Observe the size and margins, available for resynchronization using the debug toolkit or system console.

Refer to Chapter 4, Debug Toolkit for DDR2 and DDR3 SDRAM Controllers with ALTMEMPHY IP.

Additionally for PHY configurations that use a dedicated PLL clock phase (as opposed to a resynchronization FIFO buffer), use the same process as described in "Write Timing Margin", to dynamically sweep resynchronization margin (c4 `resynch_clk_2x`).

#### Postamble Timing Issues and Margin

The postamble timing is set by the PHY during calibration. You can diagnose postamble issues by viewing the `pnf_per_byte` signal from the example driver. Postamble timing issues mean only read data is corrupted during the last beat of any read request.

## Intermittent Issues

Intermittent issues are typically the hardest type of issue to debug—they appear randomly and are hard to replicate.

### Intermittent Issue Evaluation

Errors that occur during run-time indicate a data related issue, which you can identify by the following actions:

- Add the SignalTap II logic analyzer and trigger on the post-trigger `pnf`
- Use a stress pattern of data or transactions, to increase the probability of the issue
- Heat up or cool down the system
- Run the system at a slightly faster frequency

If adding the SignalTap II logic analyzer or modifying the project causes the issue to go away, the issue is likely to be placement or timing related.

Errors that occur at start-up indicate that the issue is related to calibration, which you can identify by the following actions:

- Modify the design to continually calibrate and reset in a loop until the error is observed

- Where possible, evaluate the calibration margin either from the the debug toolkit or system console.

  Refer to Chapter 4, Debug Toolkit for DDR2 and DDR3 SDRAM Controllers with ALTMEMPHY IP

- Capture the calibration error stage or error code, and use this information with whatever specifically occurs at that stage of calibration to assist with your debug of the issue.

# Monitoring Signals with the SignalTap II Logic Analyzer

The following sections detail the memory controller signals you should consider analyzing for different memory interfaces. The list is not exhaustive, but is a starting point.

For a description of each signal, refer to *Volume 3: Implementing Altera Memory Interface IP* of the *External Memory Interface Handbook*.

## DDR, DDR2, and DDR3 ALTMEMPHY Designs

Monitor the following signals for DDR, DDR2, and DDR3 SDRAM ALTMEMPHY designs:

- `Local_* -example_driver` (all the local interface signals)
- `Pnf -example_driver`
- `Pnf_per_byte -example_driver`
- `Test_complete -example_driver`
- `Test_status -example_driver`
- `Ctl_cal_req -phy_inst`
- `Ctl_init_fail -phy_inst`
- `Ctl_init_success -phy_inst`
- `Ctl_cal_fail -phy_inst`
- `Ctl_cal_success -phy_inst`
- `Cal_codvw_phase * -phy_inst`
- `Cal_codvw_size * -phy_inst`
- `Codvw_trk_shift * -phy_inst`
- `Ctl_rlat * -phy_inst`
- `Ctl_wlat * -phy_inst`
- `Locked -altpll_component`
- `Phasecounterselect * -altpll_component`
- `Phaseupdown -altpll_component`
- `Phasestep -altpll_component`

- Phase_done -altpll_component
- Flag_done_timeout -seq_inst:ctrl
- Flag_ack_timeout -seq_inst:ctrl
- Proc_ctrl.command_err -seq_inst:ctrl
- Proc_ctrl.command_result * -seq_inst:ctrl
- dgrb_ctrl.command_err -seq_inst:ctrl
- dgrb_ctrl.command_result * -seq_inst:ctrl
- dgwb_ctrl.command_err -seq_inst:ctrl
- dgwb_ctrl.command_result * -seq_inst:ctrl
- admin_ctrl.command_err -seq_inst:ctrl
- admin_ctrl.command_result * -seq_inst:ctrl
- setup_ctrl.command_err -seq_inst:ctrl
- setup_ctrl.command_result * -seq_inst:ctrl
- state.s_phy_initialise -seq_inst:ctrl
- state.s_init_dram -seq_inst:ctrl
- state.s_write_ihi -seq_inst:ctrl
- state.s_cal -seq_inst:ctrl
- state.s_write_btp -seq_inst:ctrl
- state.s_write_mtp -seq_inst:ctrl
- state.s_read_mtp -seq_inst:ctrl
- state.s_rrp_reset -seq_inst:ctrl
- state.s_rrp_sweep -seq_inst:ctrl
- state.s_rrp_seek -seq_inst:ctrl
- state.s_rdv -seq_inst:ctrl
- state.s_poa -seq_inst:ctrl
- state.s_was -seq_inst:ctrl
- state.s_adv_rd_lat -seq_inst:ctrl
- state.s_adv_wr_lat -seq_inst:ctrl
- state.s_prep_customer_mr_setup -seq_inst:ctrl
- state.s_tracking_setup -seq_inst:ctrl
- state.s_tracking -seq_inst:ctrl
- state.s_reset -seq_inst:ctrl
- state.s_non_operational -seq_inst:ctrl
- state.s_operational -seq_inst:ctrl
- dqs_delay_ctrl_export * -phy_inst

- * = Disable Trigger Enable

## UniPHY Designs

Monitor the following signals for UniPHY designs:

- avl_addr
- avl_rdata
- avl_rdata_valid
- avl_read_req
- avl_ready
- avl_wdata
- avl_write_req
- fail
- pass
- afi_cal_fail
- afi_cal_success
- test_complete
- be_reg (QDRII only)
- pnf_per_bit
- rdata_reg
- rdata_valid_reg
- data_out
- data_in
- written_data_fifo|data_out
- usequencer|state*
- usequencer|phy_seq_rdata_valid
- usequencer|phy_seq_read_fifo_q
- usequencer|phy_read_increment_vfifo*
- usequencer|phy_read_latency_counter
- uread_datapath|afi_rdata_en
- uread_datapath|afi_rdata_valid
- uread_datapath|ddio_phy_dq
- qvld_wr_address*
- qvld_rd_address*

# 3. ALTMEMPHY Calibration Stages

In all configurations, the noncalibrated address, command and control interfaces must be correctly constrained and meet timing.

If calibration fails at a specific stage, use this chapter to understand what functionally happens at that stage, to assist with the debug.

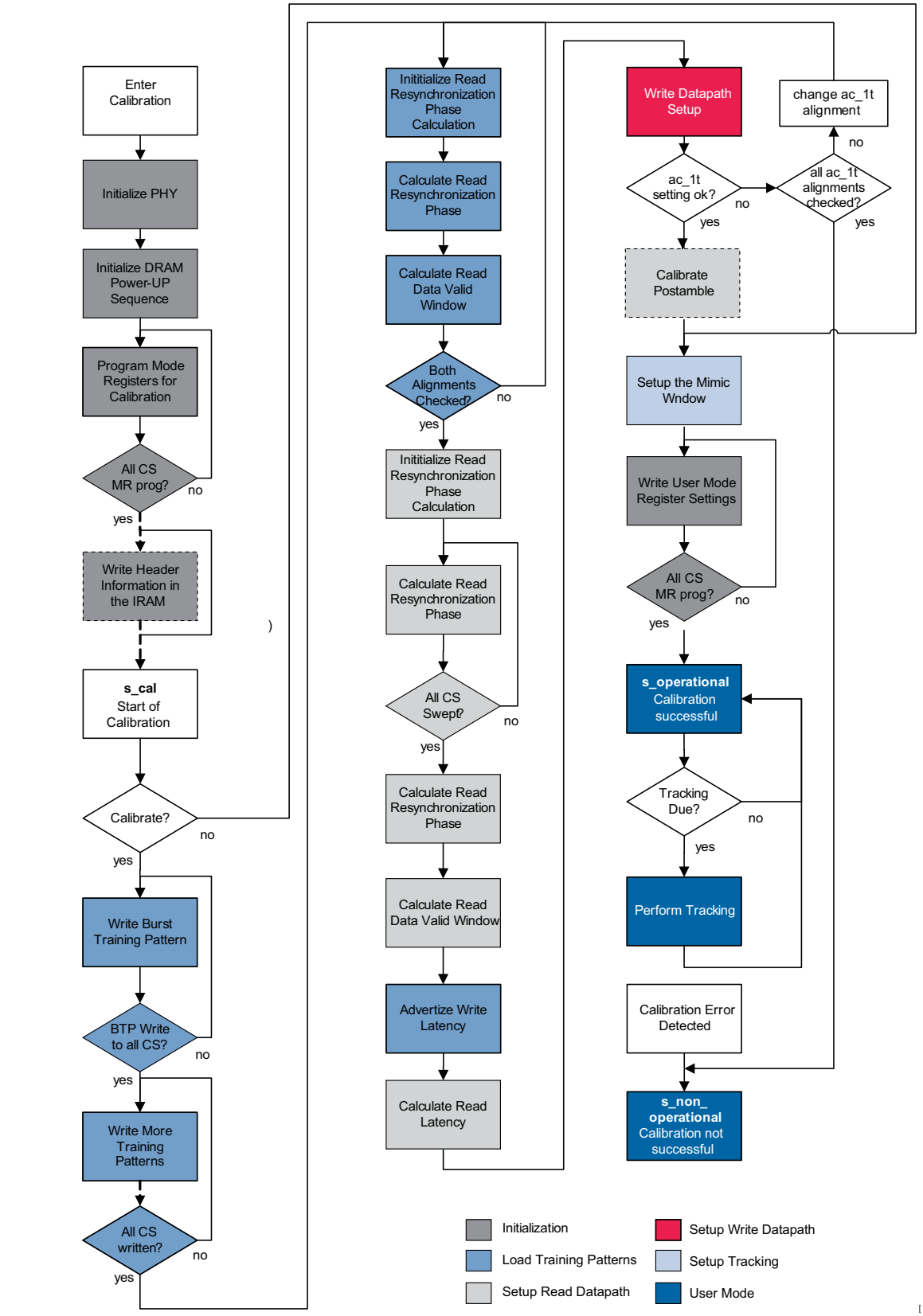
## Without Leveling

The ALTMEMPHY without leveling PHY performs the following calibration stages:

1. Enter Calibration (s_reset)
2. Initialize PHY (s_phy_initialize)
3. Initialize DRAM
4. Write Header Information in the internal RAM (s_write_ihi)
5. Load Training Patterns
6. Test More Pattern Writes
7. Calibrate Read Resynchronization Phase
8. Advertize Write Latency (s_was)
9. Calculate Read Latency (s_adv_rlat)
10. Output Write Latency (s_adv_wlat)
11. Calibrate Postamble (s_poa)
12. Set Up Address and Command Clock Cycle
13. Write User Mode Register Settings (s_prep_customer_mr_setup)
14. Voltage and Temperature Tracking

This chapter discusses these stages. Figure 3–1 on page 3–2 shows a flow chart of the calibration stages for the PHY without leveling.

**Figure 3–1. Calibration Stages—Without Leveling**



## Enter Calibration (s_reset)

Calibration starts when the PHY reset signal is de-asserted and AFI signal `ctl_cal_req` is low.

## Initialize PHY (s_phy_initialize)

This stage holds off calibration until the DLL has locked and (if debug toolkit is enabled) internal RAM contents are all reset to zero.

## Initialize DRAM

Initializing the DRAM has the following two stages:

- Initialize DRAM Power Up Sequence (s_int_dram)
- Program Mode Registers for Calibration (s_prog_mr)

### Initialize DRAM Power Up Sequence (s_int_dram)

This stage brings the SDRAM out of a reset state (from any previous state) through the initialization sequence specified in the JEDEC specification for each device type, up to but not including mode register set commands. At the end of this stage the SDRAM is ready to receive mode register load commands, which must occur (on each rank) before refreshes can occur.

This calibration stage is applied to all chip selects in parallel.

### Program Mode Registers for Calibration (s_prog_mr)

Mode register set commands are issued on a per chip select basis, which allows you great flexibility to issue different mode register settings to different chip selects. When all chip selects have mode registers programmed (the initialization of that chip select is complete), refreshes are enabled.

The following overrides are applied to user settings:

- For DDR and DDR2 SDRAM:
  - DLL enable
  - Burst length 4
  - OCD calibration (DDR2 only)
- For DDR3 SDRAM:
  - DLL enable
  - Output buffer enable
  - Disable write leveling
  - Runtime burst length select
  - Test mode disabled
  - DLL reset

For DDR3 SDRAM this stage also includes a `ZQ-cal` long operation (refer to the JEDEC specification).

## Write Header Information in the internal RAM (s_write_ihi)

In this stage, the internal header information in the first eight locations in the internal RAM is loaded with the parameterization of the PHY. The debug toolkit uses this information to provide the current PHY parameterization and IP version numbers.

This stage is only executed when the debug toolkit is enabled.

Refer to Chapter 4, Debug Toolkit for DDR2 and DDR3 SDRAM Controllers with ALTMEMPHY IP

## Load Training Patterns

In this stage, training patterns are written to the memory to be read in later calibration stages. Because of the matched trace lengths to DDR SDRAM components, after memory initialization, you can assume write capture works.

Training pattern writes are divided into the following two stages:

- Write Block Training Pattern (s_write_btp)
- Write More Training Patterns (s_write_mtp)

A further training pattern is written in the calibration of the write datapath, refer to Advertize Write Latency (s_was).

### Write Block Training Pattern (s_write_btp)

This stage is used in read data valid alignment (`s_rdv`), advertise read latency (`s_adv_rd_lat`) and postamble calibration (`s_poa`). For these calibration stages a pattern of all 1s and all 0s is sufficient to setup the PHY.

Writing of these two patterns is trivial and requires all DDIO outputs (high and low phases (bits)) to be held at either 1 or 0, for all 1 and all 0 patterns, respectively. To write these patterns, the DDIO outputs are held low (or high) and DQS toggled for a predetermined length of time and a single write command is issued. A full range of memory write latencies are tested.

To support DDR3 SDRAM discrete components (burst of eight reads) eight memory locations are loaded with 1s and eight with 0s.

The following memory locations contain the following patterns:

- Locations [7.:0], all 0s
- Locations [15:8], all 1s

Patterns of all 1s and all 0s are also needed for calibrating the read resynchronization phase.

### Write More Training Patterns (s_write_mtp)

This stage is used in calculating the read resynchronization phase (`s_rrp_sweep`).

The pattern is `0x30F5` and comprises separately written patterns. This pattern is required to match the characterization behavior for nonDQS capture based schemes (for example, Cyclone III devices). All device families use the following pattern:

- All 0: `'b0000` to DDIO high and low bits held at 0
- All 1: `'b1111` to DDIO high and low bits held at 1
- Toggle: `'b0101` to DDIO high bits held at 0 and DDIO low bits held at 1
- Mixed: `'b0011` to DDIO high and low bits have to toggle

While you can ensure that all zeros, all ones, or toggle are written into a burst of memory locations (output DDIO bits are held at constant values), it is challenging to ensure that a pattern of `0011` is written into memory. The challenge occurs because the write latency is unknown at this time. For example, if this pattern is repeated on DQ pins (`0011 0011 0011`) and a single write command issued (as for the other patterns), it is not known whether the memory location contains the pattern `0011` or `1100`.

ALTMEMPHY provides a methodology to robustly write these patterns (Test More Pattern Writes). For this section two locations (X and Y) are populated with write data, one contains the pattern `0011`; the other contains `1100`.

The memory locations contain the following patterns:

- `x30` alignment 0 to location (X): `23..16`
- `x30` alignment 1 to location (Y): `31..24`
- `xF5` to location: `39..32`

These patterns are written in bursts of four beats, so in the pattern `xF5`, `F` is written separately to 5. Patterns `F` and `0` are written as a part of the writing of the block training pattern (Write Block Training Pattern (s_write_btp)).

## Test More Pattern Writes

This stage comprises a number of calibration stages of the PHY, but the stage is described as one entity. This stage comprises:

- Initialize read resynchronization phase calculation (`s_rrp_reset`)
- Calculate read resynchronization phase (`s_rrp_sweep`)
- Calculate read data valid window (`s_read_mtp`)

The algorithm ensures the pattern `0011` is written to a known location in memory. The following assumptions and PHY settings apply to this stage:

- Assumptions:
  - Burst length of 4 writes
  - Write capture in the memory device works. Data can be safely written to memory. No write leveling is required. (Refer to JEDEC specification for more information on DDR3 SDRAM).
  - Writes are aligned on a clock cycle basis. You know which beats of DQ data to write on the low and high phases of DQS (ultimately DQ and DQS board delays are well matched).

- Settings:
  - Address and command 1T setting. You can add additional latency to the address and command path (specified as a maximum of one memory clock cycles (t)). This setting aligns write data to address and command signals relative to the controller clock domain, where address and command signals are issued in a given alignment. This setting is not required (`0t`) for a full-rate PHY.
  - Read data 1T alignment. Additional delay of captured read data to align with read commands in the half rate controller cock domain. The interpretation of 1T is the same as for address and command, but applied to read data. This setting does not apply to a full-rate PHY.
  - Read resynchronization phase setting. This setting is the primary task of the training pattern to correctly set the resynchronization phase in the middle of data valid window for the read data (on DQ pins) to be captured.

From this algorithm, to determine PHY settings B and C, given that A is not set, follow these steps:

1. Try to write the pattern and indistinguishable variations of it to different memory locations. For example, write to different locations with the following two patterns on the DQ bus (timed to a local controller rate clock), refer to Write More Training Patterns (s_write_mtp):
   a. `0011 0011 0011` (try to write `0011`) in location X
   b. `1100 1100 1100` (try to write `1100`) in location Y
2. Perform two single-pin DQ pin and single-chip select read resynchronization phase calibrations using location X and location Y, as part of the larger training pattern (`0x30F5`). You do not know at this time which locations X and Y contain the pattern `0011`. This stage iterates through the following stages:
   - Initialize read resynchronization phase calculation (`s_rrp_reset`)
   - Calculate read resynchronization phase (`s_rrp_sweep`)
   - Calculate read data valid window (`s_read_mtp`)

   Calculate read data valid window (`s_read_mtp`) is a special case of calibrate read resynchronization phase (`s_rrp_sweep`), refer to Calibrate Read Resynchronization Phase (s_rrp_sweep). This stage reports the size of the returned window without setting up the PLL phase or producing an error if no window is observed.
3. The single pin read resynchronization calibration (using pattern X or Y), which results in the largest data valid window, contains the optimal pattern. The read resynchronization calibration with the largest window indicates the location (X or Y) that contains the correct alignment (`0011`). Read resynchronization phase calibration uses this alignment (X or Y) , to perform the full resynchronization phase calibration across all pins and chip selects.

During calibration of the read resynchronization phase, the DQ pin is captured using a free running clock, phase shifted through a given number of steps. You should try to match a training pattern against read data, for each phase shift, and a window of valid phases is composed.

In waveforms, you observe two resynchronization phase sweeps, over single pins, before the larger phase sweep. Except in fast simulation mode, where the duration of all three sweeps is equal. For half-rate interfaces, you may observe a total of six phase sweeps, where the entire calibration is repeated when the address and command 1T setting is toggled.

## Calibrate Read Resynchronization Phase

This stage encompasses the following calibration stages:

- Initialize Read Resynchronisation Phase Calibration (s_rrp_reset)
- Calibrate Read Resynchronization Phase (s_rrp_sweep)
- Calculate Read Resynchronization Phase (s_rrp_seek)

This stage adjusts the phase of the resynchronization (or capture) clock to determine the optimal phase that gives the greatest margin. For DQS based capture schemes the resynchronization clock captures the outputs of DQS capture registers (DQS is the capture clock). In a non-DQS capture based scheme, the capture clock captures the input DQ pin data (the DQS signal is unused).

For all half-rate PHY interfaces, a 720° resynchronization or capture clock phase sweep is performed. For a half-rate PHY this sweep is effectively 360° of the half-rate clock, because resynchronization or capture clock is at the memory clock rate. A 720° sweep is required, so that read data can be presented to a controller aligned to one half-rate controller clock cycle.

For full-rate DQS based capture, because the DQ pins are captured using the DQS signal, in a 360° phase sweep, all resynchronization clock phases may pass. In this case the correct resynchronization phase to set cannot be determined. The correct phase is the one in the center of a valid window, where returned read data is correct. Thus a 720° phase sweep is performed. From 360 to 720°, a clock cycle of latency may be added to a 0 to 360° sweep. The returned read data is compared to a training pattern pseudo half-rate (at half the clock rate of the sequencer), so data can only be valid for 360° of the sweep. This method introduces edges to the data valid window such that a correct phase can be chosen.

For non-DQS capture in general, up to half (180°) of the swept capture clock phases can result in correct capture data, because the DDR to SDR conversion is performed by the capture clock, and thus high and low phases of DQ are captured in the incorrect alignment for half of the capture clock phases. Therefore, only 360° of capture clock need be swept for full-rate non-DQS capture based PHYs.

The pseudo half-rate case potentially adds one clock cycle of latency into the read datapath because of the 720° sweep. In ALTMEMPHY this occurrence is detected and the clock cycle of latency removed in the calculate read resynchronization phase (`s_rrp_seek`).

### Initialize Read Resynchronisation Phase Calibration (s_rrp_reset)

This stage returns the PLL to a nominal zero phase shift.

### Calibrate Read Resynchronization Phase (s_rrp_sweep)

This stage performs a sweep through a parameterised 360° or 720° of resynchronization clock phase. These results are optionally stored in the internal RAM.

The command has the following attributes:

- `Single_pin` to indicate just to sweep DQ pin 0 as for the use in testing the write more training patterns stage.
- `mtp_alignment` to say which location (X/Y) to use from the write more training patterns stage.

### Calculate Read Resynchronization Phase (s_rrp_seek)

This stage calculates the size and center (in phase steps) of the largest data valid window found during the calibrate read resynchronization phase and sets PLL phase to the center phase.

Calculate read data valid window (`s_read_mtp`) is a special case of this stage which reports no errors for an invalid window (a failure is expected in one case) and does not setup the PLL.

### Calculate Read Data Valid Window (s_rdv)

This stage sets the latency on a delayed version of the `doing_rd` signal, so that it is aligned with the `rdata_valid` signal for the read data it is incident with the read command for.

This stage has the following process:

1. Reads a continuous stream of 1s followed by one read of zeros. The sequencer only asserts `doing_rd` when read command for zeros is issued.
2. Checks for alignment of read data valid signal (delayed version of `doing_rd`) to the zeros (`rdata` = 0, `rdata_valid` = 1).
3. If not aligned, reduces latency between `doing_rd` and `rdata_valid` signal and loop.

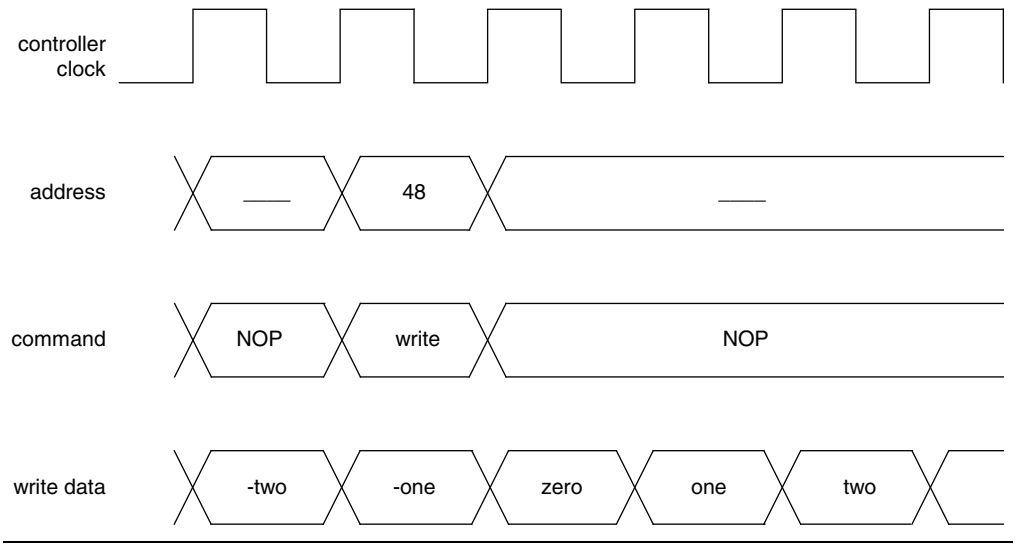
Read data valid latency is reset to a high value (before calibration) and then reduced until it matches the correct alignment.

## Advertize Write Latency (s_was)

This stage writes a suitable pattern to the DRAM to calculate the write latency.

A write command is issued to a memory address 48 (Figure 3–2) while driving a count of frequency controller clock rate to the DQ pins (Figure 3–2), using the write datapath observed by the controller. For half-rate interfaces, each four beats of write data on DQ are identical. For full-rate interfaces, each two beats of write data are identical. In general, the write latency is written into addresses A... A + ($n$ – 1), where $n$ is two times the ratio of controller to memory clock rate and each $n$ bits of write data must be identical. The pattern is written into memory locations 48 to 55.

**Figure 3–2. Description of Write Pattern**




## Calculate Read Latency (s_adv_rlat)

In addition to read data valid window calculation (`s_rdv`), the advertised read latency is calculated in this stage in the following way:

- Issues a read command (with `doing_rd`) and starts a counter at PHY clock rate
- When `rdata_valid` returns, outputs the value of the counter to `ctl_rlat` signal.

This signal is redundant, because a controller can use the `rdata_valid` signal to determine when valid read data is returned.

☞ The read data from the DRAM is not important here. The count is performed between the issue of `doing_rd` and `rdata_valid` returning.

## Output Write Latency (s_adv_wlat)

To calibrate a PHY write datapath to a minimum latency, a robust process is required to determine the write latency (WL) between a memory controller write command and the associated write data. Factors that can contribute to WL are: the memory CAS latency, arbitrary additive delays in the PHY, and board trace lengths. The presented approach extends from calibrating a PHY, where the controller operates at the memory clock frequency, to controller operation at half or a quarter of the memory clock frequency.

After a predetermined maximum read latency clock cycles have passed, the contents of the chosen memory address (0x48 in Figure 3–2) are read to recover the write latency. The first *n* beats of read data contain the write latency.

While this method recovers the write latency it can determine the address and command 1T setting in half rate mode.

The returned read data, in the controller clock domain, should be equal across the first four read data beats, as aligned to the controller clock domain. For this check to work an alternate read location must be read immediately before and after that containing the write latency.

If read data is not correctly aligned then the address and command 1T setting is toggled and calibration is rerun from write training patterns stage.

## Calibrate Postamble (s_poa)

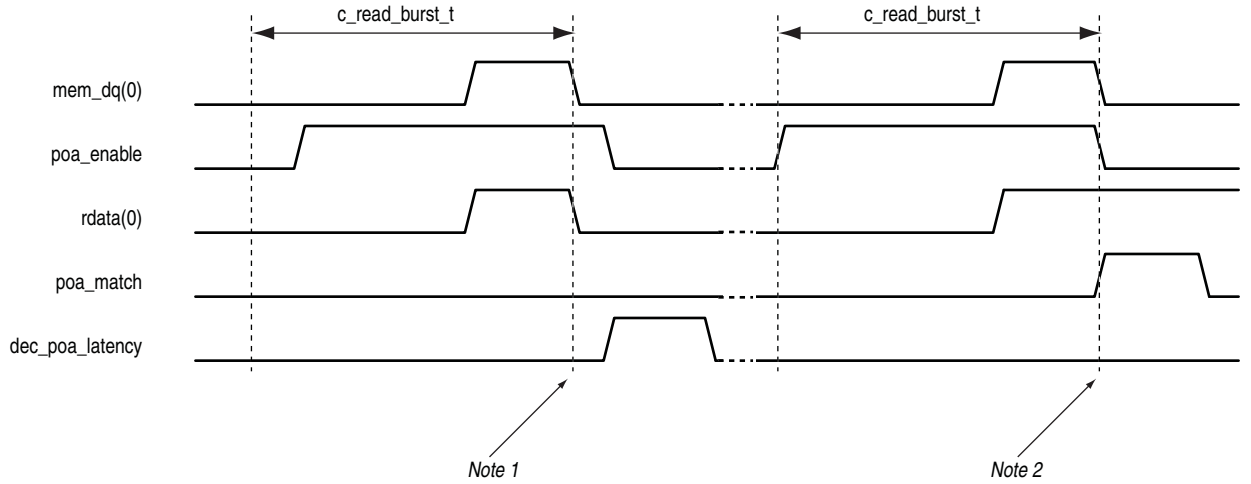
This stage is only implemented for DQS capture based PHYs (not used for Cyclone III devices).

For this stage, the PHY reads the pattern 0x30 from memory, so that the de-assertion of the postamble protection signal (poa_enable) can be aligned to the 1's in this pattern.

This stage sets the correct clock cycle for the postamble path. The aim of the postamble path is to eliminate false DQ data capture because of postamble glitches on the DQS signal, through an override on DQS. This stage ensures the correct clock cycle timing of the postamble enable (override) signal.

The delay on the postamble enable signal starts off too large. It is then iteratively reduced until postamble enable de-asserts in the clock cycle before the last falling edge on DQS. Figure 3–3 shows the calibration timing diagram.

**Figure 3–3. Calibration Timing**




**Note to Figure 3–3:**

(1) The poa_enable signal is late, and the zeros on mem_dq after here are captured.

(2) The poa_enable signal is aligned. Zeros following here are not captured and rdata remains at 1.

## Set Up Address and Command Clock Cycle

For half-rate interfaces, this stage also optionally adds an additional memory clock cycle of delay from the address and command path. This stage aligns write data to memory commands given in the controller clock domain. You see this stage in the waveform as a rerun of calibration (from the writing of training patterns) to calibrate to the new setting.

This stage is detected in the advertise write latency stage (`s_adv_wlat`)

## Write User Mode Register Settings (s_prep_customer_mr_setup)

User mode register setting are applied on a per chip select basis without the overrides in the program mode registers for calibration (s_prog_mr) stage.

## Voltage and Temperature Tracking

Voltage and temperature tracking is a background process that tracks the voltage and temperature variations to maintain the relationship between the resynchronization or capture clock and the data valid window that were achieved at calibration. When the data calibration phase completes, the sequencer issues the mimic calibration sequence every 128 ms (in user mode).

### Setup the Mimic Window (s_tracking_setup)

During initial calibration, the mimic path is sampled using the measure clock. The `measure_clk` signal has a `_1x` or `_2x` suffix, depending whether the ALTMEMPHY is a full-rate or half-rate design. The sampled value is then stored by the sequencer. After a sample value is stored, the sequencer uses the PLL reconfiguration logic to change the phase of the measure clock by one voltage-controlled oxcillator (VCO) phase tap. The sequencer then stores the sampled value for the new mimic path clock phase. This sequence continues until all mimic path clock phase steps are swept. After the sequencer stores all the mimic path sample values, it calculates the phase which corresponds to the center of the high period of the mimic path waveform. This reference mimic path sampling phase is used during the voltage and temperature tracking phase.

### Perform Tracking (s_tracking)

In user mode, the sequencer periodically performs a tracking operation. At the end of the tracking calibration, the sequencer compares the most recent optimum tracking phase against the reference sampling phase. If the sampling phases do not match, the mimic path delays have changed because of voltage and temperature variations. When the sequencer detects that the mimic path reference and most recent sampling phases do not match, the sequencer uses the PLL reconfiguration logic to change the phase of the resynchronization clock by the VCO taps in the same direction. This procedure allows the tracking process to maintain the near-optimum capture clock phase setup during data tracking calibration as voltage and temperature vary over time. The relationship between the resynchronization or capture clock and the data valid window is maintained by measuring the mimic path variations because of the voltage and temperature variations and applying the same variation to the resynchronization clock.

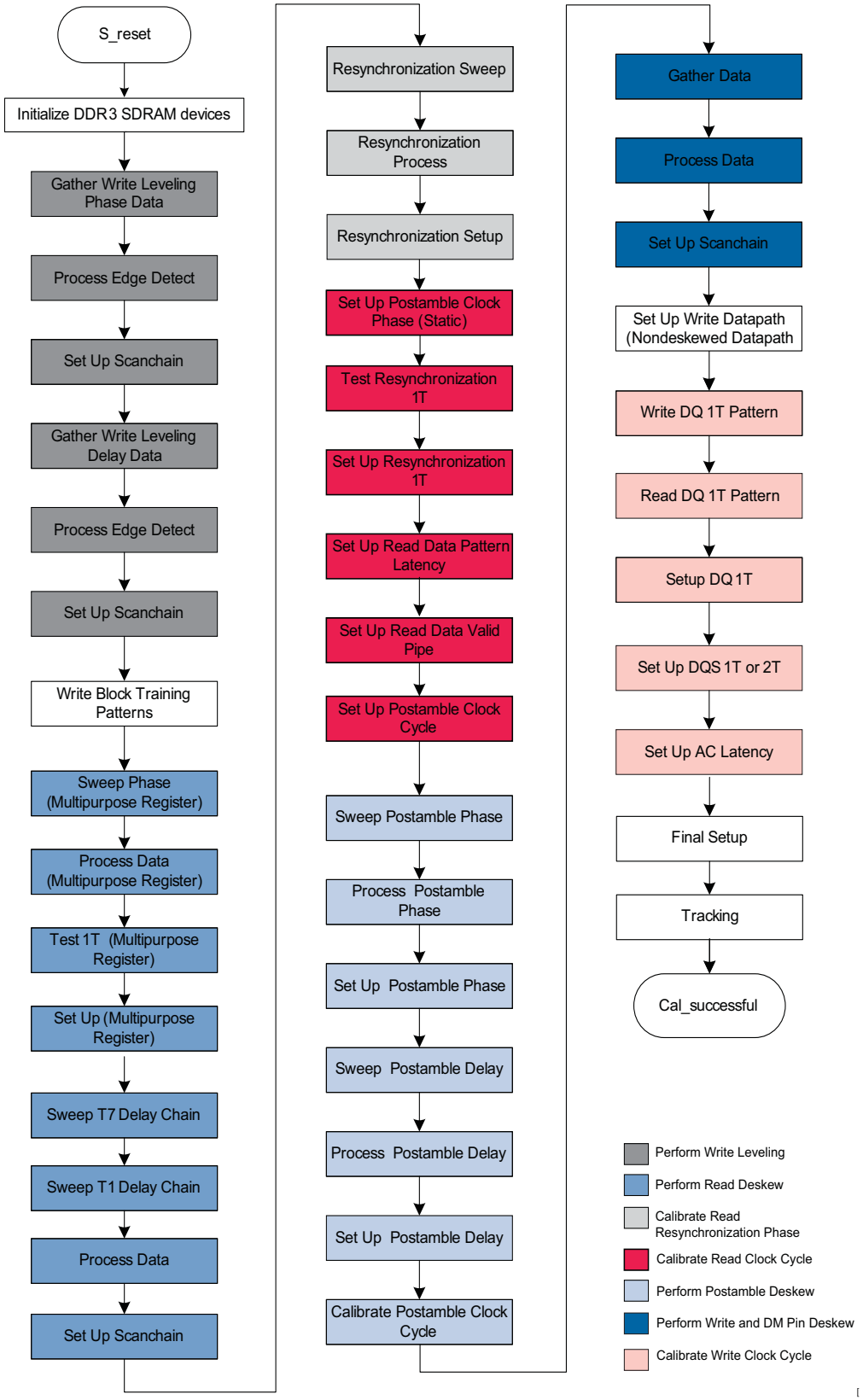
# With Leveling Calibration Stages

The ALTMEMPHY with leveling performs the following calibration stages:

1. Initialize
2. Perform Write Leveling
3. Write Block Training Patterns to Memory
4. Perform Read Deskew
5. Calibrate Read Resynchronization Phase
6. Calibrate Read Clock Cycle
7. Perform Postamble Deskew
8. Perform Write and DM Pin Deskew
9. Set Up Write Datapath (for Nondeskewed Datapath)
10. Calibrate Write Clock Cycle
11. Final Setup
12. Tracking
13. Idle (User Mode)

Figure 3–4 on page 3–13 shows a flow chart of the calibration stages for the PHY with leveling.

**Figure 3–4. Calibration Stages—With Leveling**



## Initialize

Before DDR3 SDRAM calibration begins, the ALTMEMPHY sequencer is held in reset until the PLL locks. When the PLL locks, the sequencer starts the DDR3 SDRAM calibration sequence. The initialization stage performs the following operations:

- Initialize all internal ALTMEMPHY memories
- Initialize all DDR3 SDRAM interface I/O scanchains, including the individual DQ and DQS group I/O scanchains

  ☞ When initialization completes, all the following interface related settings are incorrect:

  - Phase and delay settings for the generated clocks
  - DQ, DQS, resynchronization and postamble clocks (`MEM_CLK`, `DQS_CLK`, `RESYNC_CLK` and `POSTAMBLE_CLK`)
  - Read and write 1T settings per DQS group
- Set DDR3 SDRAM ODT settings and issue `ZQCAL` calibration command
- Set appropriate mode register settings for MR0, as per ALTMEMPHY GUI settings except:
  - Burst length configured to burst chop 4 or 8 on-the-fly
  - DLL reset
- Set appropriate mode register settings for MR0, as per ALTMEMPHY GUI settings except:
  - Enable DLL
  - Enable output buffer
  - Disable write leveling
- Set appropriate mode register settings for MR2 as GUI settings
- Set appropriate mode register settings for MR3 to all 0s

☞ These mode register settings are specific to the DDR3 SDRAM calibration sequence and do not match the user mode register settings. These settings are setup in the mode register configuration stage, before entering user mode.

## Perform Write Leveling

At ALTMEMPHY reset or FPGA power-up time the ALTMEMPHY sequencer initializes all DDR3 SDRAM I/O write and read clocks with a phase setting of 0° and delays of 0 ps across all DQS groups in the DDR3 SDRAM interface. The relationship of DQS and the memory clock at each DDR3 SDRAM component is unknown and violates the DDR3 SDRAM $t_{DQSS}$ timing parameter.

Write leveling determines and configures the required DQS phase and delay for each DQS signal, so that the DDR3 SDRAM $t_{DQSS}$ timing parameter functions correctly across PVT. Write leveling consists of the following stages:

- Gather data using the DDR3 SDRAM with write leveling

- Process the data
- Set up the appropriate I/O scanchain functions (phase and delay) based on the output of the processing stage

Write leveling is not confined to DDR3 SDRAM UDIMM interfaces. Write leveling and the DDR3 SDRAM calibration sequencer covers DDR3 SDRAM components where the memory clock, address and command signals are routed as per a DDR3 UDIMM in a fly-by topology.

This section provides a description of the following stages in write leveling:

- Gather write leveling phase data
- Process edge detect
- Set up scanchain
- Gather write leveling delay data
- Process edge detect
- Set up scanchain

During write leveling the DDR3 ALTMEMPHY sequencer determines the following optimal settings:

- Phase of the clock (`DQS_CLK`) to generate DQS
- Above 360 MHz, the DLL operates in 8-tap mode, providing 45° resolution
- At 360 MHz, the DLL operates in 10-tap mode, providing 36° resolution
- Delay applied to the DQS output pins

The write leveling calibration stage uses the write leveling of the DDR3 SDRAM components where the DQS signal samples the memory clock. The sampled value is then sent to the FPGA on the prime DQ pin within a DQS group. Write leveling is performed on all DQS groups in the DDR3 SDRAM interface.

### Gather Write Leveling Phase Data

During the write leveling phase data gathering stage, all phases of the write leveling delay chain generated DQS clock (`DQS_CLK`) are swept through 360° using the leveling multiplex controlled by the DQS group I/O scanchains. For each phase tap tested, the sequencer toggles DQS from 0 to 1. The DQS is sampled by the memory clock and output on the prime DQ pin. The fedback value sampled from the prime DQ pin is stored in the main sequencer data storage RAM (internal RAM).

The prime DQ pin can be any pin or multiple pins within the DQS group.

When all `DQS_CLK` clock phases are swept through 360°, the internal RAM contains a window of the fedback and sampled DDR3 SDRAM clock values. The internal RAM now contains a mixture of 0s and 1s. The sampled values are stored per DQS group. The values stored in the internal RAM are a coarse result, because of the resolution of the write leveling delay chains compared with the delay chain resolution.

On starting write leveling, the prime DQ pin number is unknown. Therefore all DQ pins have to be considered prime, the fedback sampled result can be output on any DQ pin in a DQS group. The sequencer uses the fact that the non-prime DQ pins either drive the same sampled result or drive zero as sampling feedback is not supported on these pins. Therefore, when writing the sampled result into the internal RAM the results for each DQ within a DQS group are ORed together.

### Process Edge Detect

During this stage the write leveling phase sweep results are read out of the internal RAM and are processed. For each DQS group the last phase tap value which returns 0 is determined. The next phase tap value should return a 1 value. As the write leveling calibration sequence is looking for a 0 to 1 transition per DQS group, the last phase returning a 0 needs to be determined to allow a fine grained delay to be applied during the subsequent calibration stages. The results of the processing stage are stored in the internal RAM.

### Set Up Scanchain

During the scanchain setup stage each DQS_CLK clock phase per DQS group is set to the final phase, where 0 is returned as determined during the edge detect processing stage. The ALTMEMPHY sequencer writes to the DQS group scanchain, to setup the DQS_CLK clock phase. During this stage, the DQS phase has been coarsely setup to a phase tap resolution of either 45 or 35°. The next calibration stages describe the fine delay setup using the fine resolution delay chains.

### Gather Write Leveling Delay Data

Now the fine 50 ps delay setting needs to be calibrated. As for the phase gathering stages, the delay chain is swept for each DQS pin in the DDR3 SDRAM interface. Again in a similar fashion to the phase gathering stages, the value returned on the prime DQ pin is stored in the sequencer internal RAM. The 50 ps delay sweep provides a more accurate DQS and memory clock relationship when write leveling calibration completes. For each phase tap tested, the sequencer toggles DQS from 0 to 1. The ALTMEMPHY sequencer internal RAM stores the sampled result.

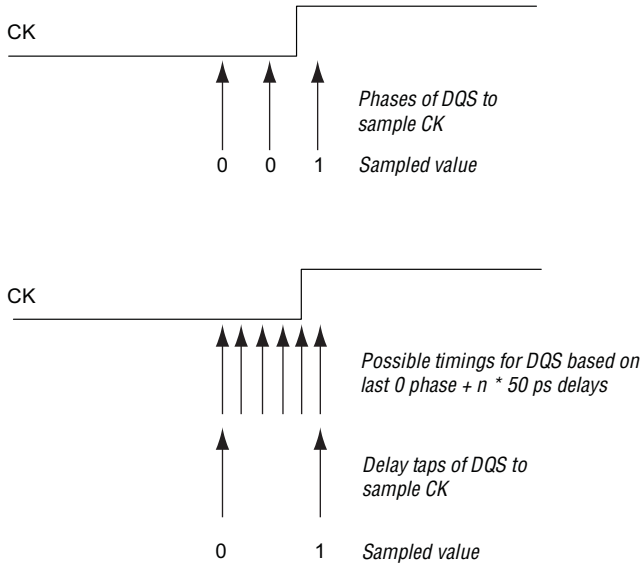
### Process Edge Detect

During this stage, the write leveling delay sweep results are read out of the internal RAM and are processed. For each DQS group the first delay tap value that returns 1 is determined. This delay for the configured phase tap per DQS group satisfies the write leveling calibration requirement where the DDR3 SDRAM $t_{DQSS}$ is met when a 0 to 1 transition is detected on the fedback and sampled signal.

### Set Up Scanchain

During the scanchain setup stage each DQS_CLK clock delay per DQS group is set to the first delay where 1 is returned as determined during the edge detect processing stage. Figure 3–5 shows how to determine phase tap to use for delay calibration and that you should pick RHS most phase that returns 0 when next phase returns 1.

**Figure 3–5. Determine Phase Tap**




### Multiple Chip Selects

When configured for multiple chip selects write leveling is performed once per chip select. The write leveling phase and delay results per chip select are written into separate internal RAM locations. When the write leveling gathering results are obtained for each chip select, the processing determines an average phase and delay setting for all chip selects.

## Write Block Training Patterns to Memory

To perform DDR3 SDRAM interface calibration, the ALTMEMPHY sequencer reads back data patterns from the DDR3 SDRAM. At ALTMEMPHY reset or FPGA power-up time, the ALTMEMPHY has not established the following relationships:

- Write latency, address and command to write data
- DQS, address and command required
- DQ and DQS required

You cannot write toggling write DQ data into the DDR3 SDRAM to form a calibration training pattern. You can only write a DC pattern into the DDR3 SDRAM, as no data is toggling when DQS is toggling. To perform a block training pattern write, all DQs are held at a DC level for a period of time before the write DQS and are additionally held for a period of time after the write DQS, to avoid any setup or hold write memory write failures with respect to the toggling DQS signal.

The block training patterns that write into the DDR3 SDRAM are:

- 11111111
  - Eight consecutive ones, the ones pattern
  - Write to address address 48
  - Write as BL8
- 00000000
  - 8 consecutive zeroes, the ZEROS pattern
  - Write to address 16
  - Write as BL8
- 11110000
  - Four consecutive ones, four consecutive zeroes, mixed pattern
  - Written to address 24
  - Write as BL8 and BC4

The calibration writes the ones and zeroes patterns as a continuous burst length 8 operation (BL8). The calibration writes the mixed pattern as a BL8 (zeros) and then overwrites with a burst chop 4 (BC4) operation (ones pattern). The following calibration stages use the block training patterns:

- Perform read deskew
- Calibrate read resynchronization phase
- Calibrate read clock cycle
- Perform postamble deskew

## Perform Read Deskew

Dynamic read deskew is only active on Stratix III and Stratix IV designs above 400 MHz.

Dynamic read deskew is a calibration technique where the size and location of the read data valid window for each DQ pin is individually assessed. If necessary, the window position is shifted to guarantee that a you see always a valid window across all possible operating conditions (voltage and temperature).

The following two mechanisms shift the position of the valid window:

- The T7 (D4) delay-chain in the DQS logic-block shifts an entire DQS group in one direction by increasing the delay on the DQS input path used for capture.
- The DQ T1 (D1, 2, and 3) delay-chains individually configure for each DQ pin. DQ T1 shifts the individual DQ pins in the opposite direction to the DQS logic-block T7 shift by increasing the input delay on the DQ pin.

When the calibration measures the size and position of the each DQ window, it applies an algorithm to pull the capture point more towards the center, if required by adjusting the DQS logic-block T7 and DQ T1 delay-chain settings appropriately.

Read deskew is a two-pass flow. In the first pass, the calibration uses the DDR3 in-built multipurpose register pattern (which may only return data on a single DQ pin per DDR3 SDRAM component). In the second pass, the calibration reads one of the block training patterns (`11110000`) from memory (which returns data on all DQ pins).

The first pass deskews and configures the interface sufficiently that the calibration can read the more aggressive block training pattern.

### Gather Data

During the read deskew data gathering stage, the calibration sweeps all DQS logic-block T7 delay-chain values, while it holds all DQ T1 delay-chains at zero. Then the calibration sweeps all DQ T1 delay-chain values, while it holds all DQS T7 delay-chains at zero.

For each configured delay-chain settings, the calibration reads the appropriate pattern (multipurpose register or block training pattern) from external DRAM, with four-times over-sampling to mitigate the effects of jitter and capture the minimum size window. If the calibration successfully reads the pattern, the corresponding result bit in the internal RAM is set to 1, otherwise it is cleared. Different areas of the internal RAM store the multipurpose register and block training pattern results, the latter does not overwrite the former.

The calibration implements oversampling in the control-plane rather than the data-plane, a single read is performed at each of the configured T7/T1 delay-chain combinations and this entire process is then repeated four times (rather than performing four reads at each T7/T1 combination). Oversampling logically `AND`s the DQ valid windows across the four reads.

### Process Data

This stage reads the data gathering results from the internal RAM (the representation of the size of the DQ valid windows) and analyses the results to generate a deskew configuration (DQS logic-block T7 and DQ T1) that is then also written to the internal RAM.

The algorithm determines the center position for each DQ window, applies a clamp limit to this value (only allows the configuration to shift up to a certain amount from the nominal case) and stores the result, in terms of DQS logic-block T7 and DQ T1 settings, back to the internal RAM.

### Set Up Scanchain

This stage reads the deskew configuration results from the internal RAM and applies these to the DQS logic-block T7 and DQ T1 delay-chains for all DQ pins in all DQS groups.

### Calibrate Resynchronization (Multipurpose Register Pass only)

This stage (sweep phase, process data, set up, and 1T test or setup) configures the interface to successfully read the more aggressive block training pattern.

## Calibrate Read Resynchronization Phase

The read resynchronization phase calibration has the following stages:

1. Resynchronization phase sweep. Collects pass and fail information for PLL phase settings over the range of 0 to 720° in increments of PLL phase steps (determined by the PLL megafunction at PHY generation).
2. Resynchronization process. Processes the results from the resynchronization phase sweep to determine the center of the passing window in PLL steps. Then converts to DLL taps by which to delay the resynchronization clock.
3. Resynchronization setup. The scanchain sets the delay (in DLL taps) in the I/O on the resynchronization clock.

### Training Pattern

For resynchronization training to work, you need a pattern for which you can see a correct pattern in the half-rate clock domain. You can write before any setup of the write path (including deskew), therefore you are limited to the block training patterns. However, only having patterns of 8 × zeros, 8 x ones, or 4 zeros and 4 ones limits what you can do. Therefore, you must use the `11110000` pattern, which is stored in address `24`.

Although DDR3 SDRAM contains a hard-coded training pattern accessible through the multipurpose register, the pattern read back is always a toggling bit. This pattern is not suitable as a training pattern for resynchronization. If capture is successful, the bits representing the rising-edge captured data and the falling-edge captured data do not change. Therefore there are no transitions at the resynchronization registers and cannot use this pattern to calibrate.

### Assumptions

Use the following assumptions:

- Working capture. At this point (after read deskew) assume that read capture is working. Because of the nature of the system, you cannot differentiate between capture or resynchronization failures, so assume that any failure is resynchronization. Failing read resynchronization may be because of read capture.
- Unknown read alignment. When you read a training pattern from memory, you do not know the latency of the entire read path, as a result you cannot predict in what cycle the read data returns. Also, because the timing for the resynchronization registers is so good, all phases in 360° may pass. Therefore, check for data transitions in the half-rate clock domain. If you define one data alignment as correct; another alignment of data is incorrect. You can use this fact to measure a resynchronization window, so you need to sweep 720° of PLL phases.

### Resynchronization Sweep

The resynchronization sweep determines which resynchronization phases work and do not work. Because of the large step size of the resynchronization phases available from the PLL or DLL compensated delay chains, you may see that all phases work, as the setup and hold times and the uncertainties in the hard paths are so small in relation to phase step (PLL or DLL).

As the size of a PLL phase step is smaller than the size of the DLL compensated phase delay, measuring the resynchronization window in terms of PLL phase steps gives a more accurate measurement (two to three times the number of steps).

Because of the all phases valid problem and well matched paths (as they are hard), you need to compare over 720° instead of 360°. Additionally, you define one alignment of read data as correct and the other as incorrect, so you see a 360° passing window out of 720°.

At this stage of calibration, you can only write the block training patterns, so you are limited in the patterns you can read back. The periodicity of the patterns you can read back means that the data only changes every two memory clock cycles. This situation is not ideal for the resynchronization window measurement, as bimodal jitter in the I/O may affect the measured margins.

As at this stage of calibration you have done nothing to determine the relevant delays for each group (address and command + memory = FPGA). It is more than likely that the read data comes back at different times, equating to a different phase and clock cycle for each group.

Use the following process:

- Activate the row and bank containing the mixed frequency block training pattern (`11110000`). Then read this address continuously.
- Bypass postamble circuitry and permanently enable.
- Start phase sweeping loop.
- Allow time to guarantee that read data has propagated from capture through resynchronization and into the core.
- Sample the read data and store the result (in internal RAM).
- Phase shift the PLL 1 step.
- If you have not done enough phase shifts (2 × `PLL_PHASES_PER_TCK`) go to loop start.
- Finish. Precharge the open row. Return postamble enable to normal.

One pin per DQS group is tested. The sweep is looped over all chip selects and the results collected with an `AND` operation, so that the final result is the worst case window across all chip selects.

## Resynchronization Process

To maximize the setup and hold margin at the resynchronization registers, you employ a center of data valid window algorithm. In case there is a fringing effect on the edge of the window, you employ a center of largest data valid window approach.

The algorithm returns the center of the largest group of passing phases, which is implemented as a shift-register based bit-serial implementation for size and speed, and is designed to indicate errors on:

- No window (all fail)
- Continuous window (all pass)
- Multiple equally sized windows (2 or more passing regions with the same size)

This algorithm returns a value ranged between 0 and number of PLL steps (between 0 and 360°).

The result of the center of largest window operation above is scaled to the nearest DLL based value.

This stage is performed on a per DQS group basis.

## Resynchronization Setup

The per-group (byte lane) scaled resynchronization value is programmed into the scanchains. However, as the processing function returns a value mod 360°, you have chosen the correct phase, but it may not be the correct clock cycle. As a result the clock cycle calibration is still required to ensure the correct work alignment for a half-rate PHY.

Additionally, to match the latency of each DQS group to compensate for leveling and use the read data valid flag, perform the read clock cycle calibration stage.

## Data Storage

The resynchronization phase selection result header has the following format::

```
1 word : header
N words : one nibble per DQS group defining the resynchronization phase.
```

The resynchronization phase selection working dataset has the following format:

```
1 Word : header (ID code 17 in location [31:24])
N Words : 2 x PLL phase steps/32 words per DQS group
One bit per phase tested moving extending over the words.
(packed from LSB)
```

Table 3–1 shows the internal RAM data format.

**Table 3–1. Internal RAM Data Format**

| Address | | Bit | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | 31 | 30 | 29 | ... | 3 | 2 | 1 | 0 |
| A | Header | | | | | | | | |
| A + 1 | Group 0 | Phase 31 | Phase 30 | Phase 29 | ... | Phase 3 | Phase 2 | Phase 1 | Phase 0 |
| ... | Group ... | Phase 31 | Phase 30 | Phase 29 | ... | Phase 3 | Phase 2 | Phase 1 | Phase 0 |
| A + 7 | Group 7 | Phase 31 | Phase 30 | Phase 29 | ... | Phase 3 | Phase 2 | Phase 1 | Phase 0 |
| A + 8 | Group 0 | 0 | 0 | 0 | ... | 0 | 0 | Phase 33 | Phase 32 |
| ... | Group ... | 0 | 0 | 0 | ... | 0 | 0 | Phase 33 | Phase 32 |
| A + 14 | Group 7 | 0 | 0 | 0 | ... | 0 | 0 | Phase 33 | Phase 32 |

### Multiple Chip Selects

For multiple chip selects, the read resynchronization phase sweep process is iterated once per chip select. The calibration writes the results for sweep 1 (for chip select 0) into the internal RAM as normal. Subsequent sweeps (for chip selects 1,2...) write into the internal RAM as a logical AND with the current internal RAM value. After a sweep of all windows, the stored internal RAM window is the intersection of windows for all chip selects (the worst case window). Processing and setup continues as for the single chip select case.

## Calibrate Read Clock Cycle

This section describes the read datapath setup sequence for clock cycle calibration.

### Set Up Postamble Clock Phase (Static)

The postamble clock phase leads the resynchronization clock phase by one read leveling delay chain tap, which is loaded into the I/O scanchains. This phase is a rough estimate for the required postamble clock phase, which is sufficient for clock cycle postamble calibration (postamble clock cycle calibration is done as the final part of read clock cycle calibration).

### Test and Set Up Resynchronization 1T

This stage determines the read data alignment on a DQS group basis. As the phase of the half-rate resync_clk (DIV2 clock) per DQS group is unknown at the start of calibration, there can be a 1T memory clock difference in the alignment of the captured and resynchronized read data. If there is a 1T alignment problem for a given DQS group, each BL4 read is distributed across 2 DIV2 clock cycles. The requirement is that each BL4 read is aligned to a single DIV2 clock cycle.

The sequencer continuously reads the mixed block training pattern (11110000) from the DDR3 SDRAM as 2 contiguous BL4 reads. The captured read data is checked that each BL4 read is aligned to a single DIV2 clock cycle. The pass or fail result is stored in the sequencer internal RAM.

Next, the sequencer reads the resynchronization 1T test result from the internal RAM, to determine which DQS groups require the 1T input register to be dynamically switched in to align all captured read data. The read 1T register is controlled using the I/O scanchains.

### Set Up Read Data Pattern Latency

Now that the captured read data is correctly aligned to the DIV2 clock on a DQS group basis, set up the read datapath so that all read data for a given read command appears at the ALTMEMPHY local interface in the same PHY_CLK clock cycle. The resynchronization 1T scanchain setup calibration stage may force the read data from the different DQS groups to be returned in different PHY_CLK clock cycles.

During this calibration stage the sequencer performs the following operations:

- Continuously read the 1s block training pattern
- Read the mixed block training pattern once
- Continuously read the 1s block training pattern

When the read data is returned the sequencer looks for the DQS group returning a PHY_CLK clock cycle of zeros with the highest latency from all DQS groups. The sequencer increases the latency through the read RAMs by 1 PHY_CLK clock cycle for all groups that do not have the same latency as the highest latency DQS group. This stage is repeated until all DQS groups return read zeros in the same PHY_CLK clock cycle

### Set Up Read Data Valid Pipe

This stage ensures the rdata_valid flag is asserted coincident with respect to the aligned read data at the ALTMEMPHY local interface. This stage performs the following operations:

- Continuously read the 1s block training pattern
- Read the mixed block training pattern once
- Continuously read the 1s block training pattern

When the aligned read data contains 0s, the rdata_valid flag is checked and adjusted until aligned with the read data. Rdata_valid is generated using a dual-port RAM where the input is the doing_rd flag. The sequencer decrements the latency through the dual-port RAM until it matches the latency of the read data.

### Set Up Postamble Clock Cycle (poa_cc_setup)

When the read data is aligned to a single DIV2 clock cycle, the postamble clock cycle relationship needs to be calibrated. If the postamble clock cycle relationship is not setup correctly, the read data is not captured correctly. This stage performs the following operations:

- Continuously read the 1s block training pattern
- Read the mixed block training pattern once
- Continuously read the 1s block training pattern

The postamble control signal gates the DQS signal (to avoid a postamble glitch from resulting in incorrect data being captured). It is generated (per DQS group) by passing the doing_rd signal through a RAM (per DQS group). The latency through this RAM is decremented until a postamble failure associated with deasserting the postamble control signal too soon is detected (read data contains all 0s for two consecutive PHY clock cycles, indicating that the 1s from the mixed pattern are not captured).

When this postamble failure condition is reached for all DQS groups, the latency through each RAM for all the DQS groups is incremented by 1, so that the interface is left in a working state.

## Perform Postamble Deskew

The postamble clock cycle calibration (part of the read clock cycle calibration), already sets up the postamble enable signal, to within a clock cycle of accuracy. Postamble deskew improves the accuracy of the postamble enable signal by adjusting the postamble clock phase and T11 delay.

### Sweep Postamble Phase

The first stage of postamble deskew is the phase sweep. Each phase of the postamble clock phase is tested (using the same data pattern and checking as for postamble clock cycle setup). The pass or fail result for each phase is written into the internal RAM (on a per DQS group basis).

### Process Postamble Phase

The phase sweep results are processed to find the passing to failing transition. The failing phase value at this transition point is stored in the internal RAM (on a per DQS group basis).

### Set Up Postamble Phase

The phase result from the previous stage is read out of the internal RAM and the scanchains are programmed (on a per DQS group basis). At this point, the sequencer is setup such that each DQS group should be failing because of postamble, which is intentional, ready for the subsequent delay sweep.

### Sweep Postamble Delay

Each T11 delay tap is tested (using the same data pattern and checking as for postamble clock cycle setup). The calibration writes the pass or fail result for each delay into the internal RAM (on a per DQS group basis).

### Process Postamble Delay

The calibration processes the delay sweep results to find the first passing delay value (on a per DQS group basis). If the calibration finds a passing delay, it writes this result into the internal RAM. If it finds no passing delays, it sets the delay result to 0 and increments the phase result 1. Finally, 180° is added to the phase result to setup the postamble enable signal optimally.

### Set Up Postamble Delay

The calibration reads the phase and delay results out of the internal RAM and the programs the scanchains accordingly.

### Calibrate Postamble Clock Cycle

The final stage of postamble deskew is to repeat the clock cycle calibration, because the phase adjustments performed earlier may cause a clock cycle boundary crossing.

## Perform Write and DM Pin Deskew

Dynamic write deskew is only active on Stratix III and Stratix IV designs above 400 MHz.

Dynamic write deskew is a calibration technique where the size and location of the write data valid window for each DQ and DM pin is individually assessed. If necessary, the window position is shifted to guarantee that a you see always a valid window across all possible operating conditions (voltage and temperature).

The following two mechanisms are employed to shift the position of the window:

- The DQ/DM T9 (D5) delay-chains, which can be individually configured for each DQ or DM pin, can shift the pin in either direction (with respect to DQS) by increasing or decreasing the delay on the DQ or DM output pin. This actions provides fine grain deskew control over a limited range.
- Only during the data gathering stage, the DQ write-leveling phase (part of the DQS logic-block) coarsely shifts all DQ and DM pins within a DQS group in either direction in steps sizes of 45°. Measurements are made in three phases; the nominal 90° DQ, DM, and DQS phase and the two adjacent phases (45° and 135°).

For each DQ and DM pin, the valid window information from all T9 (D5) delay-chain values from all three phases is combined to give an extended view of the windows. An algorithm is applied to pull the windows more towards the center if required.

Eight data patterns are used for write deskew, the calibration writes these to different areas of the external DDR3 SDRAM.

This section describes the differences in write deskew calibration for DQ and DM pins. The DM specific stages are not invoked if no DM pins are present in the PHY.

## Gather Data

The calibration uses the following eight data patterns during write deskew:

- `10101010`
- `01010101`
- `00100010`
- `01000100`
- `11011101`
- `10111011`
- `11001100`
- `11110000`

Data gathering writes all of the patterns to external DDR3 SDRAM for all swept phase and delay-chain combinations. There are distinct calibration stages for DQ and DM writes.

The procedure for DQ is:

- Iterate through all eight write data patterns.
- For each data pattern, iterate through all three DQ and DQS phase settings.
- For each phase setting, iterate through all T9 (D5) delay-chain settings.
- For each delay setting, write the selected data pattern to external DRAM. Each data-pattern, phase, or delay combination uses a different address in memory, to perform discrimination when the success of the writes is later checked by reading.

No oversampling is performed for write deskew other than that inherent in using multiple data patterns and hence have multiple writes.

The procedure for DM is slightly different. As the function of DM pins is to mask or unmask writes, the relevant portion of external DRAM is first cleared using safe writes (writes with DQ and DM values held static).

However, the procedure is the same as for DQ with the following exceptions:

- The calibration holds all DQ pins at 1, so that the success of a write can be detected. A successful write changes the memory contents from the previously written 0, to a 1, whereas an unsuccessful write the previously written 0 persists.
- The eight data patterns are applied to the DM pins.

The next part of data gathering is to read back the results of the writes and thus build up a view of the DQ and DM write data valid windows. Again, there are distinct calibration stages for DQ and DM reads.

The following process is the same for DQ and DM:

- Perform memory reads for all of the addresses that correspond to all of the pattern, phase, or delay combinations used during the preceding data gathering write stages.
- For each read, determine whether or not the corresponding write succeeded and update the write data gathering internal RAM result in a similar fashion as to read deskew.
- The results for all data-patterns for each DQ and DM pin are logically ANDed in a similar way to how read deskew over-sampling is handled.

### Process Data

This stage reads the data gathering results from the internal RAM (the representation of the size of the DQ and DM valid windows) and analyzes the results to generate a deskew configuration that the calibration then also writes to a different area of the internal RAM.

The algorithm determines the center position for each DQ and DM window, applies a clamp limit to value (only allows the configuration to shift up to a certain amount from the nominal case) and stores the result, in terms of DQ and DQS phase and DQ and DM T9 (D5) settings, back to the internal RAM.

The calibration sweeps three different phases during the data gathering stage. The processing stage combines the measured valid windows from all three phases to yield a wider view of the valid window. However the calibration always chooses the deskew result to be T9 (D5) tap within the nominal 90° phase.

### Set Up Scanchain

This stage is similar to read deskew and other setup stages. The DQS and DQ phase and DQ and DM T9 (D5) delay-chain results are read from internal RAM and set up on the appropriate scanchains.

## Set Up Write Datapath (for Nondeskewed Datapath)

To setup the DQ (and DM) versus DQS relationship, the sequencer writes to the appropriate scanchains to setup the following offsets:

- DQ phase = DQS phase – phase offset
  - 300 to 360 MHz phase offset is 72° (2 taps)
  - 360 to 400 MHz phase offset is 90° (2 taps)
  - >400 MHz. This stage is not run (write deskew is run)
- DQ T9 = DQS T9 + DQS T10 (D6) +/- static offset

T9 is given a positive or negative shift from the nominal value (DQS T10 +/- static offset), to balance the effect of the DQ and DQS phase shift, and for 10 tap DLL mode (below 360 MHz), to account for the phase shift of 72° rather than 90°.

DM phase and T9 is also set in the same way as DQ.

This static setup is for 300- to 400-MHz operation. Above 400 MHz, write deskew is run.

## Calibrate Write Clock Cycle

This stage comprises write DQ 1T pattern, read DQ 1T pattern, setup DQ 1T pattern and finds the correct write data, DQ, 1T setting for each DQS group to help correct writes to memory. This stage ensures correct alignment of write DQ data against the memory clock cycle.

### Write DQ 1T Pattern

The calibration writes a write data pattern, similar to the block training pattern, of `1111000011110000` to the DDR3 SDRAM. At this stage, the calibration uses the same technique of writing a DC value (four DQS edges for each DC value), which has a large setup and hold time, as the DQ and address and command clock cycle relationship is unknown.

### Read DQ 1T Pattern

The calibration reads the data from the same address and the read 1T alignment is checked. The calibration writes the pass or fail result into the internal RAM. Read 1T alignment issues are detected when the expected read data is distributed across 2 `DIV2` clock cycles.

### Setup DQ 1T

If the internal RAM result indicates that the read 1T alignment is incorrect, the write datapath needs a write 1T register setting to be toggled on the failing DQS group. As the read datapath has already been configured any alignment issues can only occur if the write 1T settings are incorrect.

The write 1T registers are controlled using the DQS group scanchains.

### Setup DQS 1T or 2T

This stage configures the DQS to be clock cycle correct in relation to the DQ and address and command path. This stage is a simple evaluation based on the phase of DQ, DQS, number of phases and the DQ 1T register setting. This stage determines the setting of the DQS 1T, and 2T register on a per group basis.

### Set Up AC Latency

This stage performs the following operations:

- Setups the write command and write DQ data relationship
- Calculates the number of PHY_CLK cycles between the DDR3 SDRAM controller providing the write command to the ALTMEMPHY and the write data to the ALTMEMPHY for a given write operation. This cycle is the write latency, CTL_WLAT.

The calibration writes an incrementing pattern, 0000111122223333444455556666.... pattern to the DDR3 SDRAM. Additionally, in the same clock cycle the 0000 is sent, a single write command is launched onto the address and command bus.

The value read back represents the write latency.

When the sequencer reads back from the location written to the single write command, the result per DQS group is stored. If on reading back all the data the sequencer does not see four symbols that are the same 0000 or 1111, you can assume that there is an error in the DQ_1t setup. This error results in calibration stopping.

If the address and command latencies are the same for all DQS groups, this calibration stage completes successfully.

If the address and command latencies are off by one memory clock cycle for one or more DQS groups, the address and command 1T setting is adjusted in the set AC_1T stage. You must then go back and perform the read path clock cycle setup again, and then repeat the write datapath address and command setup process again.

If the address and command latencies are off by one PHY clock cycle for one or more DQS groups, and it is the second time through this calibration stage, calibration fails and the state machine transitions to the cal_failed state. User mode is not entered, as the calibration finds no valid configuration.

The calibration does not support a difference of more than one memory clock cycle between groups.

## Final Setup

During this stage, the ALTMEMPHY sequencer programs the DDR3 SDRAM mode registers to the GUI values and tells the memory controller that the calibration sequence is complete. At this point the memory controller can start sending transactions to the attached DDR3 SDRAM components.

## Tracking

Tracking maintains equal setup and hold times at the resynchronization registers in the I/O with respect to to the resynchronization clock, RESYNC_CLK. The optimal phase of RESYNC_CLK is calibrated during the resynchronization calibration stage. As the voltage and temperature conditions of the FPGA device change, the delays on the paths associated with the capture and resynchronization stages varies. The phase of

RESYNC_CLK is no longer optimal, resulting in asymmetric setup and hold times at the I/O resynchronization registers. If the voltage and temperature variation is large enough from the initial voltage and temperature conditions at calibration time, the RESYNC_CLK clock phase may drift outside of the resynchronization window. This situation results in invalid data resynchronized to the RESYNC_CLK clock domain.

To preserve the calibrated setup and hold times the tracking process monitors a signal in the mimic path to determine how a reference point shifts as the voltage and temperature conditions change. As voltage and temperature conditions change, the ALTMEMPHY sequencer detects and changes the phase of RESYNC_CLK accordingly.

Figure 3–6 shows the mimic path block.

**Figure 3–6. Mimic Path Block**

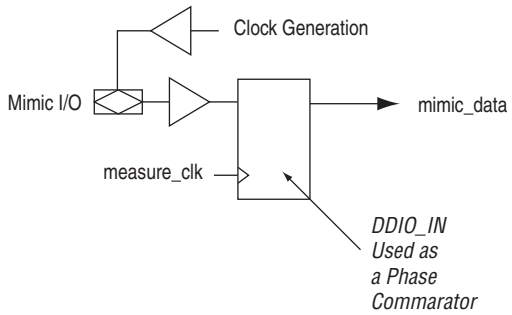



Figure 3–6 shows the following points:

- The mimic path is formed by feeding back the memory clock, MEM_CLK inside the I/O cell
- The fedback MEM_CLK is input to a DDIO_IN capture register
- Unlike read capture the DDIO_IN is clocked by variable phase MEASURE_CLK from the PLL
- The mimic path approximates the voltage and temperature variation on the FPGA memory clock and DQS input path

Tracking is performed at the following times:

- At the end of the DDR3 SDRAM calibration sequence, before entering user mode
- Periodically during user mode

At the end of calibration the ALTMEMPHY sequencer requests a mimic operation where the mimic path logic takes successive samples of the fedback MEM_CLK for a given MEASURE_CLK phase. When the sampling process completes, the mimic path logic returns the sampled value to the ALTMEMPHY sequencer, where the sample value is stored in a register. The ALTMEMPHY sequencer then increments the phase of MEASURE_CLK using the dedicated phase shifting interface on the PLL and requests another mimic operation. This sequence continues until all PLL phases in a clock cycle are swept and the sampled value has been returned.

During this sampling process, the ALTMEMPHY sequencer looks to determine future tracking operations. When determined, the PLL phase tap reference point value is stored in the internal RAM. In user mode, tracking is performed periodically. The output again is the phase step that corresponds to the reference point of the fedback MEM_CLK. The sequencer compares the previous and current phase tap value stored in the internal RAM and updates the RESYNC_CLK phase accordingly if there is a difference between the two values. For example, if the previous tracking operation returns a PLL phase tap value of 2 and the current tracking operation returns a PLL phase tap value of 3 the sequencer increments the phase of RESYNC_CLK by +1 phase step.

## Idle (User Mode)

When calibration is complete, you can enter user mode.

# 4. Debug Toolkit for DDR2 and DDR3 SDRAM Controllers with ALTMEMPHY IP

This chapter describes a debug toolkit for ALTMEMPHY-based high performance controllers. The debug toolkit uses a JTAG connection to a Windows PC. The debug toolkit supports the following Altera AFI-based IP:

- ALTMEMPHY megafunction
- DDR2 and DDR3 SDRAM High-Performance Controller and High Performance Controller II—with leveling and without leveling interfaces

The debug toolkit does not support the QDR II and II+ SRAM, RLDRAM II with UniPHY controllers.

The debug toolkit provides detailed information regarding the calibration process. The debug toolkit and the SignalTap II logic analyzer can be run at the same time. However using **Autorun Analysis** in the SignalTap II logic analyzer slows down the JTAG communication with the debug toolkit.

This chapter provides the following information:

- "Debug Toolkit Overview"
- "Install the Debug Toolkit"
- "Modify the Example Top-Level File to use the Debug Toolkit"
- "Use the Debug Toolkit"
- "Interpret the Results"
- "Understand the Checksum and Failure Code"

The debug toolkit provides information on the failures and calibration results that assist and direct the hardware debug process. The debug toolkit does not fix a failing design. Before you use the debug toolkit, refer to Chapter 2, Debugging Hardware of the *External Memory Interfaces Handbook*.

## Debug Toolkit Overview

The debug TOOLKIT provides the following information:

- Lists the various calibration stages and indicates whether each stage was successful or not.
- States an error code specific to the exact type of calibration failure.
- Provides possible causes for calibration failures.

- Provides graphics to visualize the following parameters:
  - Resync clock phase setup (leveling = per DQ group, without leveling = per pin)
  - Read deskew multipurpose registers (MPR) (prime DQ pins only)
  - Read deskew block training pattern (per pin)
  - Write deskew (per pin)
  - Write leveling report

You can export a **.doom** file from the debug toolkit. This file provides a record of your calibration results and the ALTMEMPHY IOE configuration specific to your system, allowing you to refer to this data offline later.

## Install the Debug Toolkit

To install the debug toolkit, follow these steps:

1. Download the debug toolkit, **debug-toolkit.zip,** file from Altera website.
2. Unzip the **debug-toolkit.zip** file.
3. To start the debug toolkit, navigate to the directory where you unzipped the **.zip** file and run **debug-toolkit.exe**.

To install the debug toolkit on a Quartus II production programming PC, follow these steps:

1. On a PC running the Windows OS, copy the **debug-toolkit.zip** file to your project directory or a common programming directory that you also use to program your test platform using a USB-Blaster™ download cable.
2. Unzip the **debug-toolkit.zip** file to either your project folder or a common programming folder.

## Modify the Example Top-Level File to use the Debug Toolkit

Before you use the debug toolkit, you must modify your design's example-top-level file, by following these steps:

- Verify the Design
- Regenerate the IP
- Instantiate the JTAG Avalon-MM port in to the Example-Top Level Project
- Add Additional Signals
- Add alt_jtagavalon.v to your Quartus II Project Settings Files List
- Recompile your Quartus II Test Design
- Program Hardware with Debug Enabled .sof

Your design must follow the recommended design flow, refer to the *Recommended Design Flow* section in volume 1 of the *External Memory Interface Handbook.*

## Verify the Design

Ensure your design meets the following conditions:

- The parameters entered into the IP are correct for the memory and data rate.
- The design passes functional simulation.
- The Quartus II project has the correct board trace models specified for the PCB you are using.
- For Stratix III and Cyclone III devices, ensure that the **set t(additional_addresscmd_tpd)** parameter is correctly specified in your **.sdc** file.
- For Stratix IV and Arria II GX devices, ensure that the **Address and Command to CK skew** parameter is correctly specified in the **Board Settings** tab of the IP wizard.
- The address and command clock phase is correct, to ensure optimum balanced setup and hold times.
- The Quartus II design successfully closes timing.
- The Quartus II project has the correct pin location assignments for the PCB that you are using.
- The autogenerated IP assignments are correctly applied to the example top-level file.
- The **.sdc** constraint files are correctly applied to the example top-level file.
- The Quartus II settings are correctly applied.
- The $R_{UP}/R_{DN}$ pin locations are correctly specified in the example top-level file if required.
- The SignalTap II logic analyzer is added to the example top-level file.

Before you use the debug toolkit, follow these steps:

1. Edit the example top-level file to enable debugging:
   a. Open the <*variation name*>**.v** or **.vhd** and find the `export_debug_port` private value.

   
   Do not edit this value in the file <*variation name*>**_phy.v** or **.vhd** file.

   The value is at the bottom of the file:

```
// ============================================================
// DDR3 High Performance Controller Wizard Data
// ==============================
// DO NOT EDIT FOLLOWING DATA
// @Altera, IP Toolbench@
...
...
```

```
// Retrieval info: <PRIVATE name = "export_debug_port" value="false"
type="STRING" enable="1" />
```

b. Edit the `export_debug_port` private value to `true`:

```
// Retrieval info: <PRIVATE name = "export_debug_port" value="true"
type="STRING" enable="1" />
```

## Regenerate the IP

To regenerate the IP, follow these steps:

1. Open the MegaWizard Plug In Manager, and select **Edit an existing custom megafunction variation**.
2. Select your modified high-performance controller.
3. Click **Next** to open the IP.
4. Click **Finish** to regenerate the IP.

You now have a version of the design with debug enabled. Seven new ports with the prefix `dbg_*` are added to the controller instance through the design hierarchy up to the <*variation name*>**_example_top.v** or **.vhd**.

## Instantiate the JTAG Avalon-MM port in to the Example-Top Level Project

To instantiate the JTAG Avalon-MM port, follow these steps:

1. Declare the following wires in <*variation name*>**_example_top.v** or **.vhd**.

```
wire [12: 0] av_address;
wire av_write_n;
wire [31: 0] av_writedata;
wire av_read_n;
wire av_waitrequest;
wire [31: 0] av_readdata;
```

2. Add the following instances in <*variation name*>**_example_top.v** or **.vhd**.

```
// inst jtag avalon:
alt_jtagavalon alt_jtagavalon(
.clk (phy_clk),
 .rst_n (reset_phy_clk_n),
 .av_address (av_address),
 .av_write_n (av_write_n),
 .av_writedata (av_writedata),
 .av_read_n (av_read_n),
 .av_readdata (av_readdata),
 .av_waitrequest (av_waitrequest)
);
defparam alt_jtagavalon.SLD_NODE_INFO = 203976192;
defparam alt_jtagavalon.ADDR_WIDTH = 13;
```

```
defparam alt_jtagavalon.DATA_WIDTH = 32;
defparam alt_jtagavalon.MODE_WIDTH = 3;
```

3. Update the following port connections in the DDR2 or DDR3 SDRAM instance in <*variation name*>**_example_top.v** or **.vhd**:

   a. Locate the PHY or controller instance in the top-level file and locate the following debug port connections:

```
//<< START MEGAWIZARD INSERT WRAPPER_NAME
<variation_name> <variation_name>_inst
(
.dbg_addr (13'b0),
.dbg_cs (1'b0),
.dbg_rd (1'b0),
.dbg_rd_data (dbg_rd_data_sig),
.dbg_waitrequest (dbg_waitrequest_sig),
.dbg_wr (1'b0),
.dbg_wr_data (32'b0),
```

   b. Change the following debug port connections to:

```
//<< START MEGAWIZARD INSERT WRAPPER_NAME
<variation_name> <variation_name>_inst
(
.dbg_addr (av_address),
.dbg_cs (1'b1),
.dbg_rd (~av_read_n),
.dbg_rd_data (av_readdata),
.dbg_waitrequest (av_waitrequest),
.dbg_wr (~av_write_n),
.dbg_wr_data (av_writedata),
```

The debug toolkit is added to your example top-level file.

## Add Additional Signals

In addition to the standard SignalTap II signals, you can add the following signals during debug to understand the following situations:

- Where calibration failed:
  - `*ctl_init_fail -phy_inst`
  - `*ctl_init_success -phy_inst`
  - `ctl_cal_fail -phy_inst`
  - `ctl_cal_success -phy_inst`

- How much resynchronization margin is available:
  - *cal_codvw_phase *DT-phy_inst
  - *cal_codvw_size *DT-phy_inst
  - *codvw_trk_shift *DT-phy_inst
- What the read and write latency is calibrated as:
  - ctl_rlat *DT-phy_inst
  - ctl_wlat *DT-phy_inst
- If the PLL is locked and phase stepping as expected:
  - Locked -altpll_component
  - Phasecounterselect *DT-altpll_component
  - Phaseupdown -altpll_component
  - Phasestep -altpll_component
  - phasedone -altpll_component
  - dqs_delay_ctrl_export *DT-phy_inst

For signals marked with *DT, disable trigger enable in the SignalTap II logic analyzer to reduce memory requirement.

Table 4–1 shows sequencer signals that you can also probe using the SignalTap II logic analyzer, to help you understand where calibration failure is occurring. The signals are in the <*vaiation_name*>**_alt_mem_phy_seq.vhd** file.

All signals are active high.

**Table 4–1. Sequencer Signals (Part 1 of 2)**

| Port Name | Description |
|---|---|
| Flag_done_timeout | Calibration stage timeout failure, memory did not respond. |
| Flag_ack_timeout | Sequencer failed to respond. |
| state.s_phy_initialise | PHY initialization Stage: wait for DLL lock and init_done. |
| state.s_init_dram | DRAM initialization stage: reset sequence. |
| State.s_prog_cal_mrs | DRAM initialization stage: programming mode registers (once per chip select). |
| state.s_write_ihi | Write internal RAM header initialization. |
| state.s_cal | Calibration required stage. |
| state.s_write_btp | Write block training pattern stage: 00001111. |
| state.s_write_mtp | Write memory training patterns: 00110101. |
| state.s_rrp_reset | Read resynchronization phase reset: PLL initial condition. |
| state.s_rrp_sweep | Read resynchronization phase sweep: sweep PLL phases per chip select. |
| state.s_read_mtp | Read memory training patterns to find correct alignment. |
| State.s_rrp_seek | Read resynchronization phase setup stage: set PLL to center of valid window. |

**Table 4–1. Sequencer Signals (Part 2 of 2)**

| Port Name | Description |
|---|---|
| state.s_rdv | Read data valid stage. |
| state.s_poa | Postamble calibration stage. |
| state.s_was | Write datapath setup: write data to DRAM so that latency can be determined. |
| state.s_adv_rd_lat | Advertise read latency stage. |
| state.s_adv_wr_lat | Advertise write latency stage. |
| state.s_tracking_setup | Tracking setup stage (first pass to setup mimic window). |
| state.s_prep_customer_mr_setup | Set custom mode register settings (admin). |
| state.s_tracking | Tracking stage (mimic path tracking in user mode). |
| state.s_operational | Calibration success: user mode. |
| state.s_non_operational | Calibration failed or tracking failed in user mode. |
| state.s_reset | Reset stage. |
| | |
| dgrb_ctrl.command_err | Error in the data gather read bias block. |
| dgrb_ctrl.command_result[7..0] | Data gather read block (DGRB) error code. |
| dgwb_ctrl.command_err | Error in the data gather write bias block. |
| dgwb_ctrl.command_result[7..0] | Data gather write block (DGWB) error bode. |
| admin_ctrl.command_err | Error in the admin (DRAM initialization and control) block. |
| admin_ctrl.command_result[7..0] | Admin block error code. |
| | |
| **With Leveling ALTMEMPHY only** | |
| Proc_ctrl.command_err | Error in the processing block. |
| Proc_ctrl.command_result[7..0] | Processing block error code. |
| setup_ctrl.command_err | Error in the setup (IOE and scanchain control) block. |
| setup_ctrl.command_result[7..0] | Setup block error code. |

## Add alt_jtagavalon.v to your Quartus II Project Settings Files List

Before you compile your design, you must add the **alt_jtagavalon.v** file to your projects file list. This **alt_jtagavalon.v** file is included with the debug toolkit.

## Recompile your Quartus II Test Design

You must compile your modified design to generate a new **.sof** for testing that includes the debug toolkit code. Altera recommends you ensure that this modified design continues to pass timing analysis. Any timing failures should be assessed and corrected before using the debug toolkit.

### Program Hardware with Debug Enabled .sof

To program hardware with the debug enabled **.sof**, program the device using the SignalTap II logic analyzer. Then click **Run Analysis** to run once. Typically, the SignalTap II logic analyzer is initially configured to trigger on the signal `test_complete`, which is fine for working designs.

For designs that are failing calibration, Altera recommends modifying the trigger based on the observed results. Combine this SignalTap II trigger fault isolation activity with use of the debug toolkit. For example:

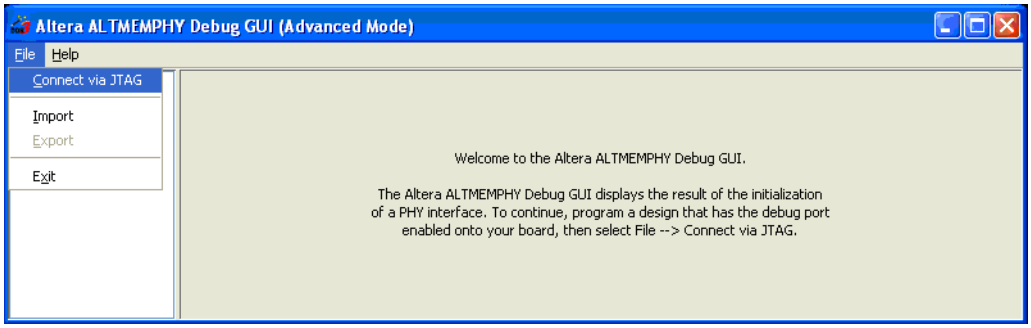
1. Initially trigger on `test_complete`, if the interface works first time.
2. Trigger on `cal_fail`, if the PHY is failing calibration.
3. Trigger on the same `state.s_*` or error code that is reported as the calibration failure point in the debug toolkit.
4. Trigger on `init_fail`, if the memory is failing to initialize.
5. Trigger on `pll_locked`, if the PLL is operating incorrectly.

## Use the Debug Toolkit

To use the debug toolkit, follow these steps:

1. Double-click **debug-toolkit.exe**.
2. On the File menu, click **Connect via JTAG** (Figure 4–1).

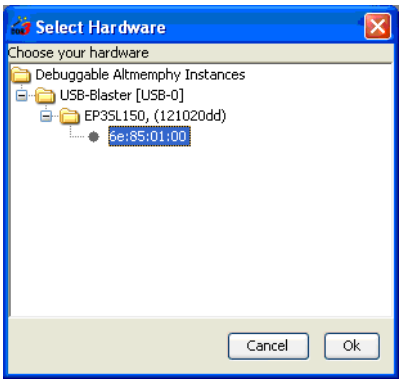
**Figure 4–1. Connect to JTAG**




3. Navigate down the hierarchy and click on the Avalon-MM JTAG node (Figure 4–2).

If you encounter connection problems, Altera recommends that you have only a single USB-Blaster™ download cable programming adaptor connected to your PC.

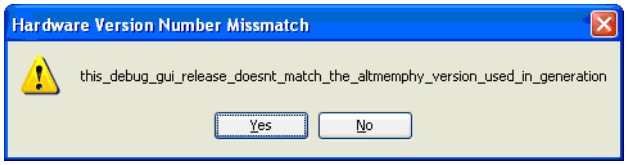
**Figure 4–2. Select Hardware**




4. If you receive a prompt stating the following message, verify you have the latest debug toolkit, and click **Yes** (Figure 4–3).

```
Hardware Version Number Mismatch -
this_debug_GUI_release_doesnt_match_the_altmemphy_version_used_in_g
eneration
```

**Figure 4–3. Hardware Mismatch**




# Interpret the Results

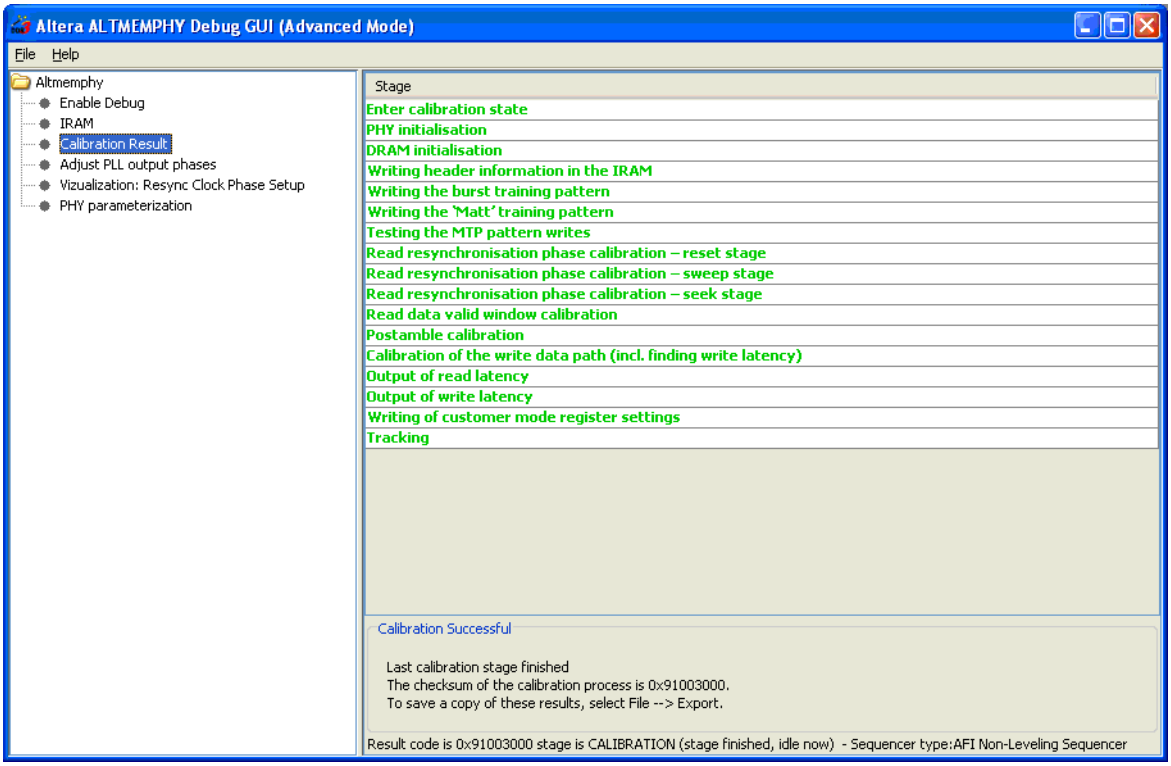
This topic discusses:

- Calibration Successful—Without Leveling
- Calibration Fails—Without Leveling
- Calibration Successful—With Leveling
- Calibration Fails—Without Leveling

## Calibration Successful—Without Leveling

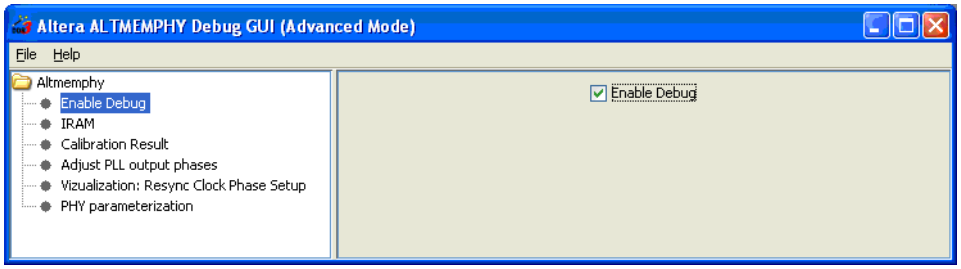
If calibration is successful, you see the following screen (Figure 4–4).

**Figure 4–4. Calibration Successful—Without Leveling**




For optimum operation of the debug toolkit, ensure that you turn on **Enable Debug** (Figure 4–5).

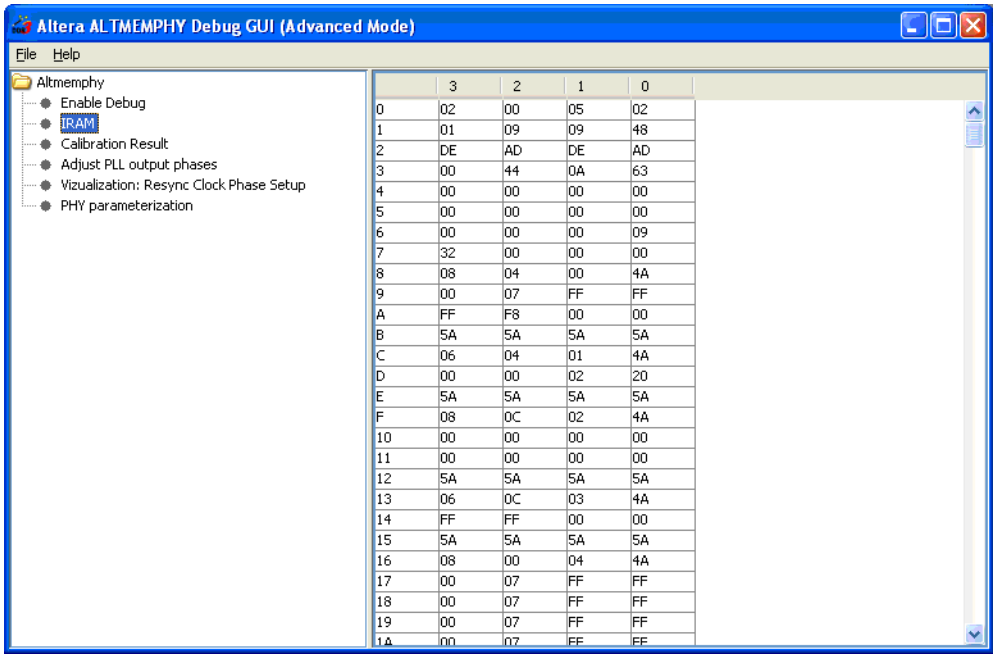
**Figure 4–5. Enable Debug**




Click **internal RAM** to display the calibration memory results (Figure 4–6).

This setting is not typically used.

**Figure 4–6. Calibration Memory Results—Without Leveling**

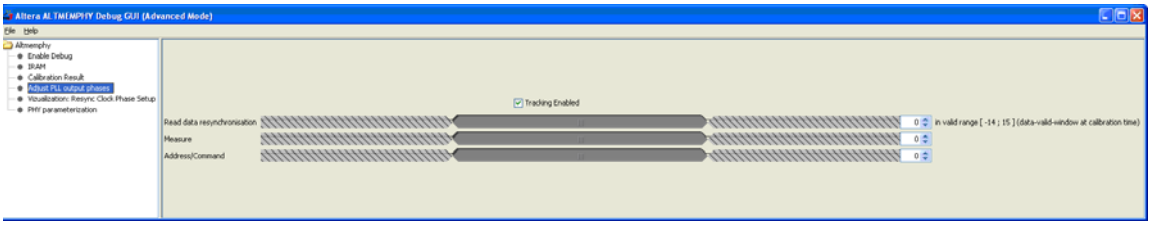



| | 3 | 2 | 1 | 0 |
|---|---|---|---|---|
| 0 | 02 | 00 | 05 | 02 |
| 1 | 01 | 09 | 09 | 48 |
| 2 | DE | AD | DE | AD |
| 3 | 00 | 44 | 0A | 63 |
| 4 | 00 | 00 | 00 | 00 |
| 5 | 00 | 00 | 00 | 00 |
| 6 | 00 | 00 | 00 | 09 |
| 7 | 32 | 00 | 00 | 00 |
| 8 | 08 | 04 | 00 | 4A |
| 9 | 00 | 07 | FF | FF |
| A | FF | F8 | 00 | 00 |
| B | 5A | 5A | 5A | 5A |
| C | 06 | 04 | 01 | 4A |
| D | 00 | 00 | 02 | 20 |
| E | 5A | 5A | 5A | 5A |
| F | 08 | 0C | 02 | 4A |
| 10 | 00 | 00 | 00 | 00 |
| 11 | 00 | 00 | 00 | 00 |
| 12 | 5A | 5A | 5A | 5A |
| 13 | 06 | 0C | 03 | 4A |
| 14 | FF | FF | 00 | 00 |
| 15 | 5A | 5A | 5A | 5A |
| 16 | 08 | 00 | 04 | 4A |
| 17 | 00 | 07 | FF | FF |
| 18 | 00 | 07 | FF | FF |
| 19 | 00 | 07 | FF | FF |
| 1A | 00 | 07 | FF | FF |

The debug toolkit can dynamically alter the PLL clock phases (Figure 4–7).

This setting is not typically used.

**Figure 4–7. Altering PLL Clock Phases**




The debug toolkit states the number of resynchronization clock phase steps that are valid at calibration time. For example, the resynchronization window size in PLL phase steps at calibration in Figure 4–7 is 26 PLL phase steps wide.

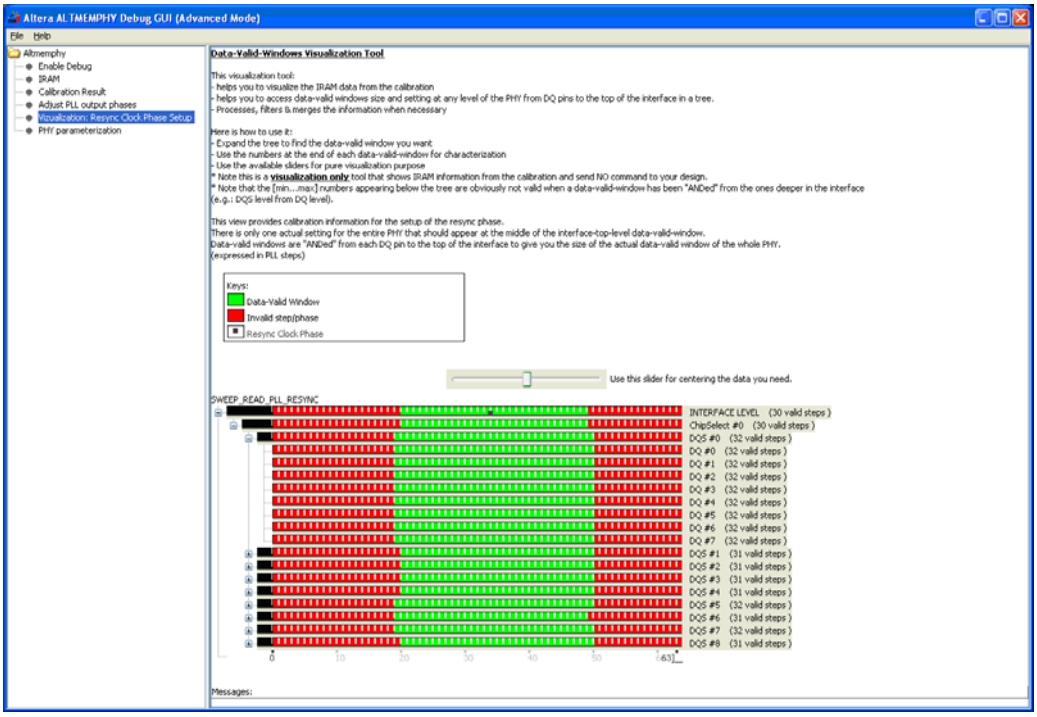
The address and command phase sweep is limited to either address and command pin margins or address and command core-to-I/O transfer margins.

In Figure 4–7, moving the slider to the left increases the value, while moving the slider to the right decreases the value.

Click **Visualization: resync clock phase setup** (Figure 4–8) to show the PHY resynchronization pass and fail results in an expandable tree structure:

- For the whole interface including the chosen phase (black dot)
- On a DQS group basis
- On a per DQ pin basis

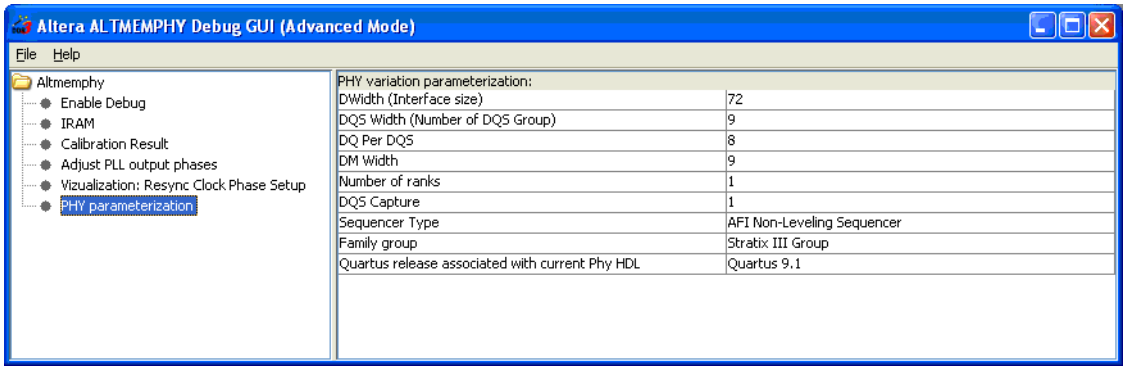
**Figure 4–8. Visualization**




The debug toolkit additionally states the number of passing phase steps that it finds during calibration. Thus to calculate the resynchronization margin in this design, the resynchronization clock (C6) from the PLL has a phase-shift step resolution of 78.12 ps or 5.62 degrees. So 30 valid steps means that the window size = 2.343 ns or 168.6 degrees.

Click **PHY parameterization** (Figure 4–9), to show the exact calibration configuration of the generated IP.

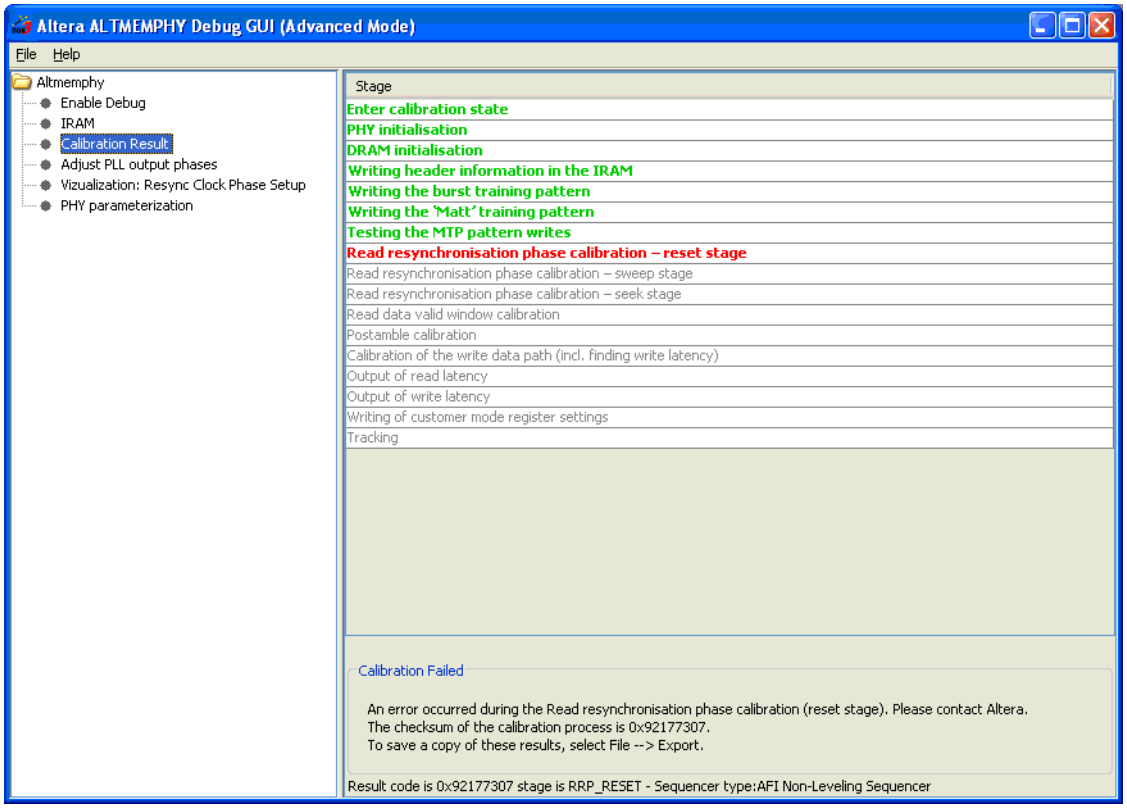
**Figure 4–9. PHY Parameterization**




## Calibration Fails—Without Leveling

If calibration fails, you see the following screen (Figure 4–10).

**Figure 4–10. Calibration Fails—Without Leveling**



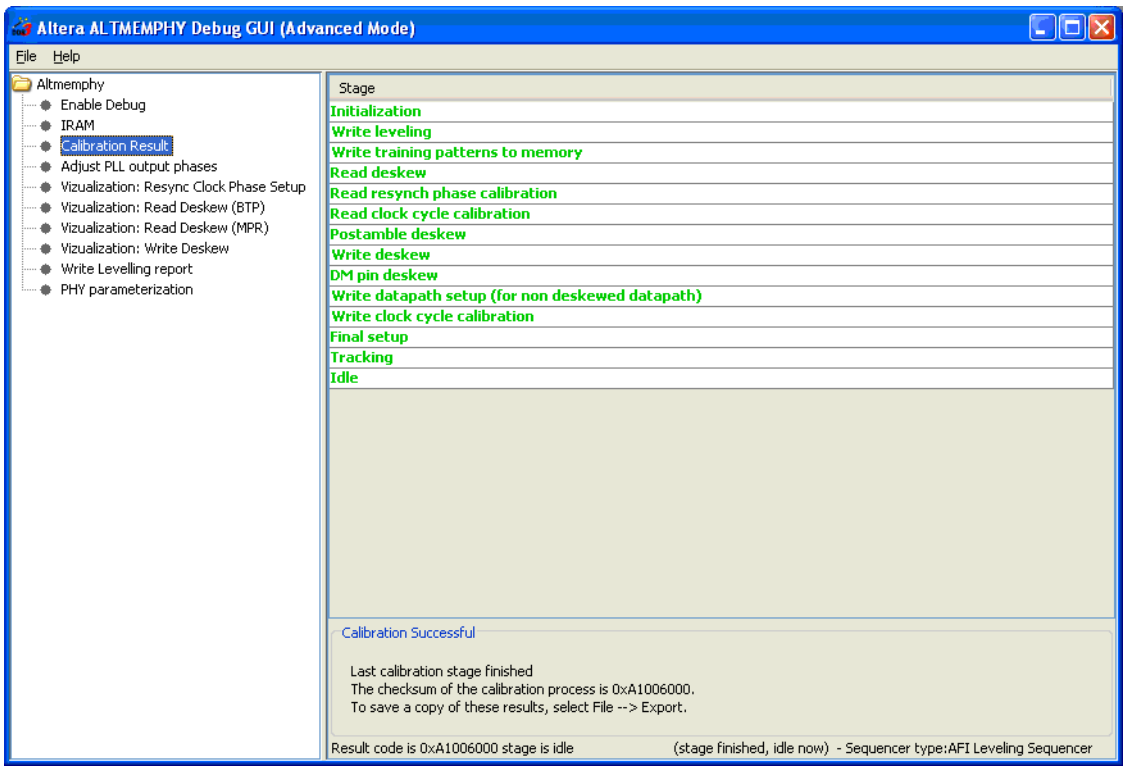

The stage at which calibration fails is highlighted in red; the stages that have successfully passed are in green. When possible, the debug toolkit also provides a possible cause for the failure code.

This failure code is: `0x92177307`, for more information on failure codes, refer to "Understand the Checksum and Failure Code" on page 4–23.

## Calibration Successful—With Leveling

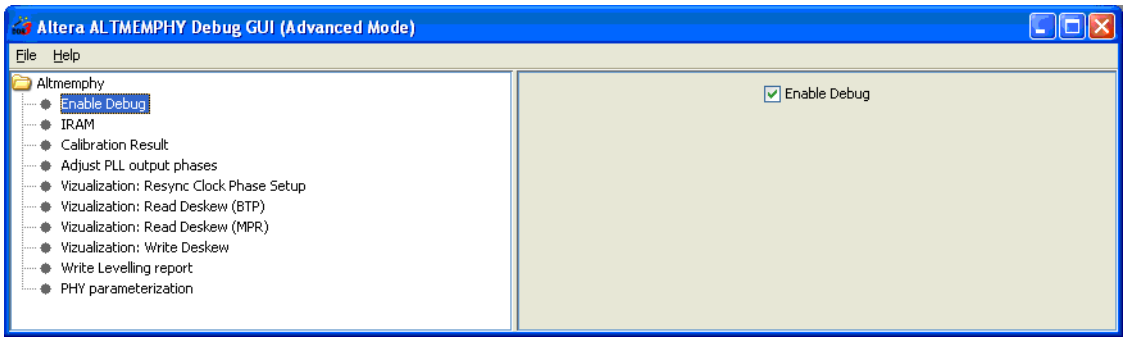
If calibration is successful, you see the following window (Figure 4–11).

**Figure 4–11. Successful Calibration—With Leveling**




For optimum operation of the debug toolkit, ensure that you turn on **Enable Debug** (Figure 4–12).

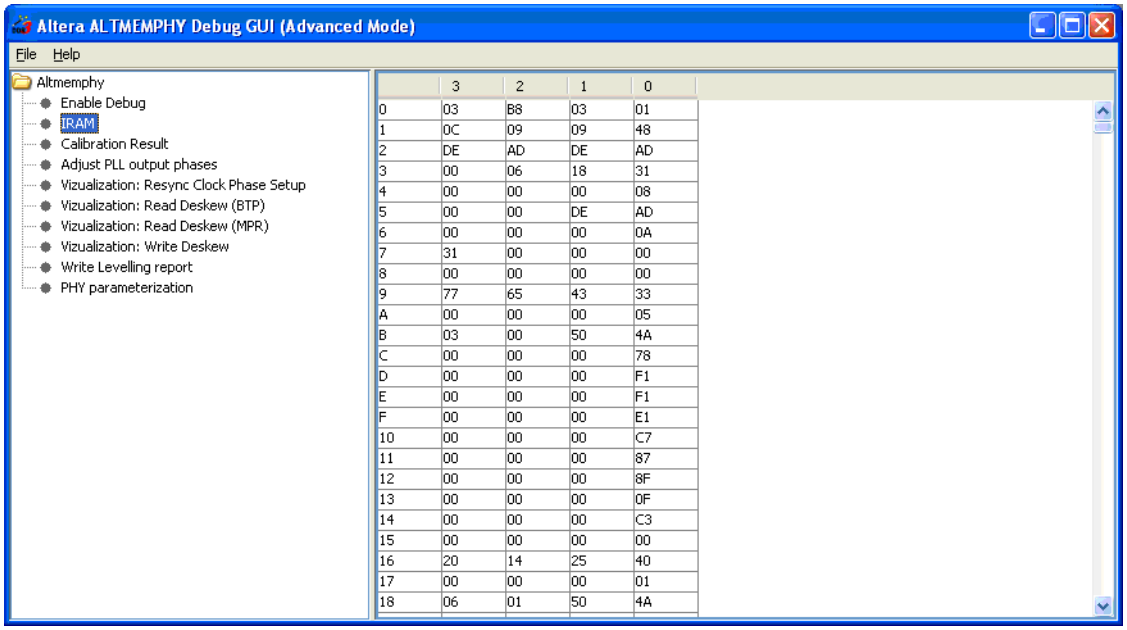
**Figure 4–12. Enable Debug**




Click **IRAM** to display the calibration memory results (Figure 4–13).

This setting is not typically used.

**Figure 4–13. Calibration Memory Results—With Leveling**

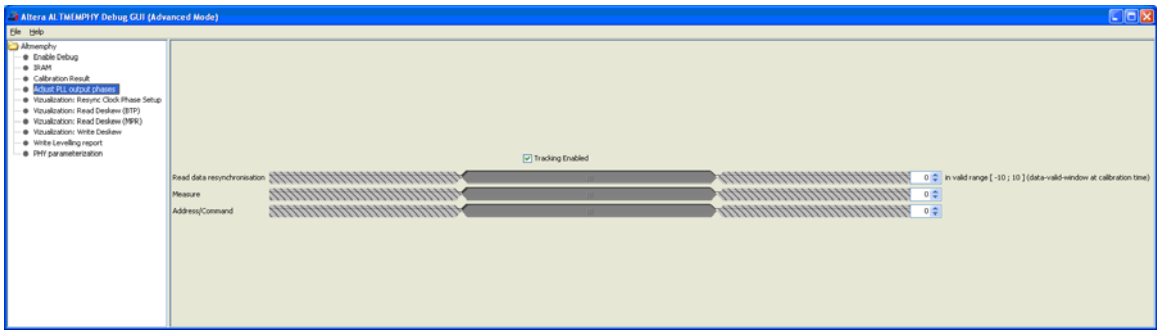



| | 3 | 2 | 1 | 0 |
|---|---|---|---|---|
| 0 | 03 | B8 | 03 | 01 |
| 1 | 0C | 09 | 09 | 48 |
| 2 | DE | AD | DE | AD |
| 3 | 00 | 06 | 18 | 31 |
| 4 | 00 | 00 | 00 | 08 |
| 5 | 00 | 00 | DE | AD |
| 6 | 00 | 00 | 00 | 0A |
| 7 | 31 | 00 | 00 | 00 |
| 8 | 00 | 00 | 00 | 00 |
| 9 | 77 | 65 | 43 | 33 |
| A | 00 | 00 | 00 | 05 |
| B | 03 | 00 | 50 | 4A |
| C | 00 | 00 | 00 | 78 |
| D | 00 | 00 | 00 | F1 |
| E | 00 | 00 | 00 | F1 |
| F | 00 | 00 | 00 | E1 |
| 10 | 00 | 00 | 00 | C7 |
| 11 | 00 | 00 | 00 | 87 |
| 12 | 00 | 00 | 00 | 8F |
| 13 | 00 | 00 | 00 | 0F |
| 14 | 00 | 00 | 00 | C3 |
| 15 | 00 | 00 | 00 | 00 |
| 16 | 20 | 14 | 25 | 40 |
| 17 | 00 | 00 | 00 | 01 |
| 18 | 06 | 01 | 50 | 4A |

The debug toolkit can dynamically alter the PLL clock phases (Figure 4–14).

This setting is not typically used.

**Figure 4–14. Altering PLL Clock Phases**




The debug toolkit states the number of resynchronization clock phase steps that are valid at calibration time. For example, the resynchronization window size in PLL phase steps at calibration in Figure 4–14 is 20 PLL phase steps wide.

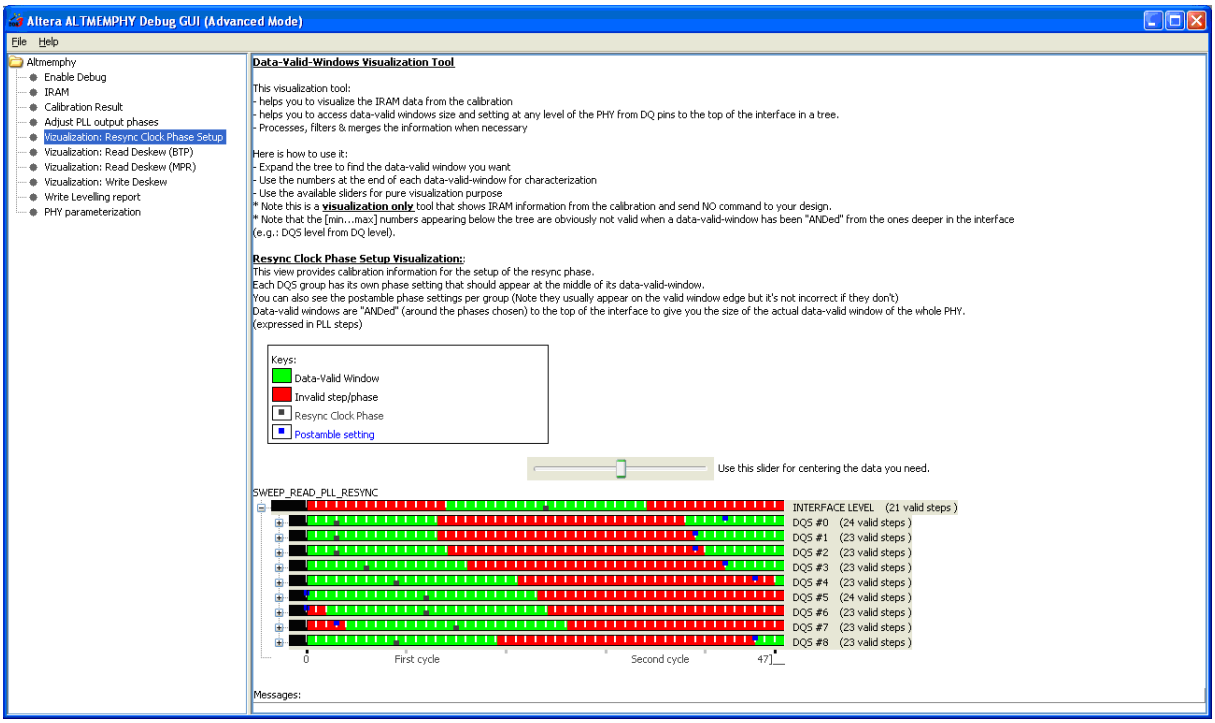
Click **Visualization: resync clock phase setup** (Figure 4–15), to show the PHY resynchronization pass and fail results in an expandable tree structure:

- For the whole interface including the chosen phase (black dot)
- On a DQS group basis

This feature is similar to the non-AFI PHY enable debug in system memory contents editor feature that allowed you to view the calibration results.

The interface level resynchronization calibration results may be 360 degrees offset when compared to the per pin results.

**Figure 4–15. Visualization: Resynch Clock Phase Setup**



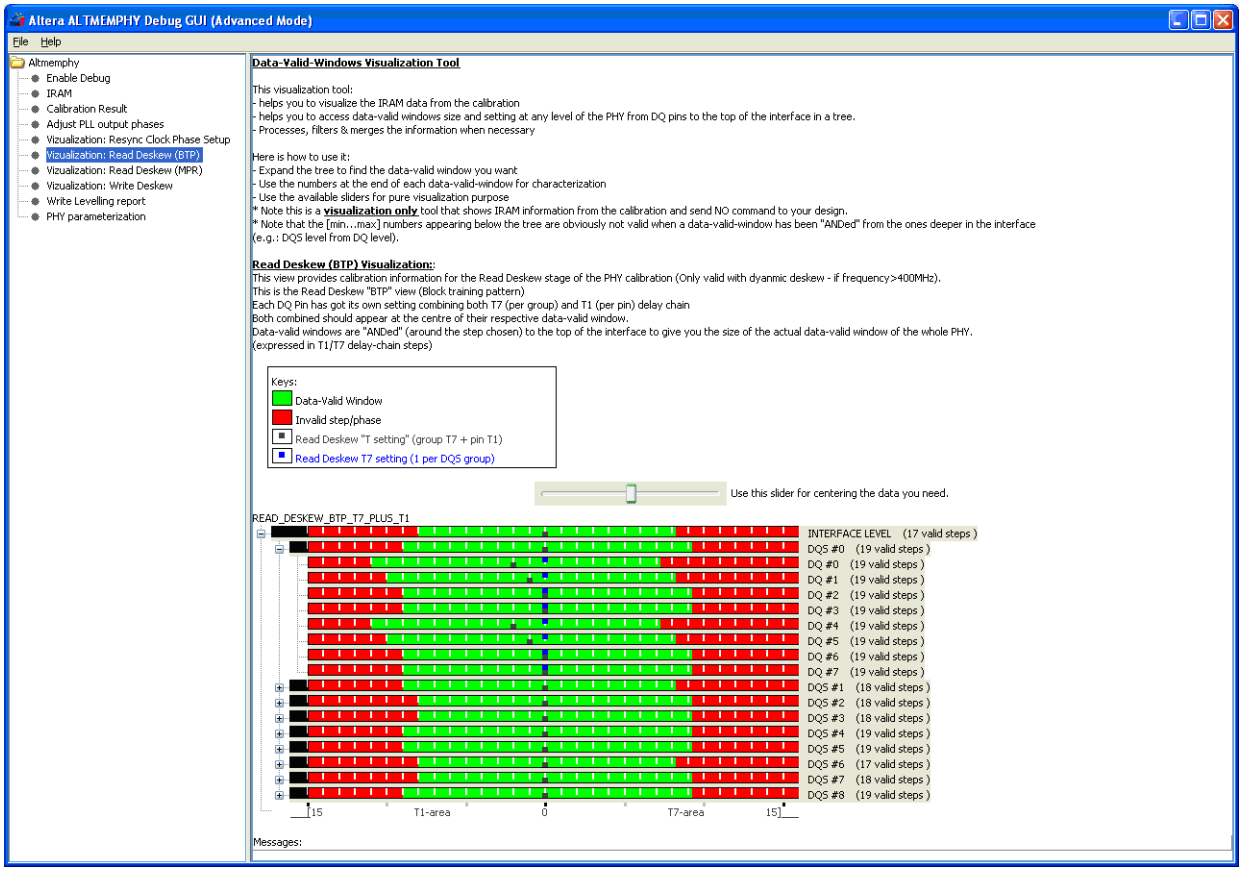

The debug toolkit additionally states the number of passing phase steps that it finds during calibration. Thus to calculate the resynchronization margin in this design the resynchronization clock (C6) from the PLL has a phase-shift step resolution of 78.12 ps or 7.50 degrees, so 21 valid steps means that the window size is 1.64 ns or 157.5 degrees..

Click **Visualization: Read Deskew (BTP)** (Figure 4–16), which shows the PHY read deskew block training pattern pass and fail results in an expandable tree structure:

- For the whole interface including the chosen phase (black dot)
- On a DQS group basis

- On Per DQ pin basis

**Figure 4–16. Visualization Read Deskew BTP**



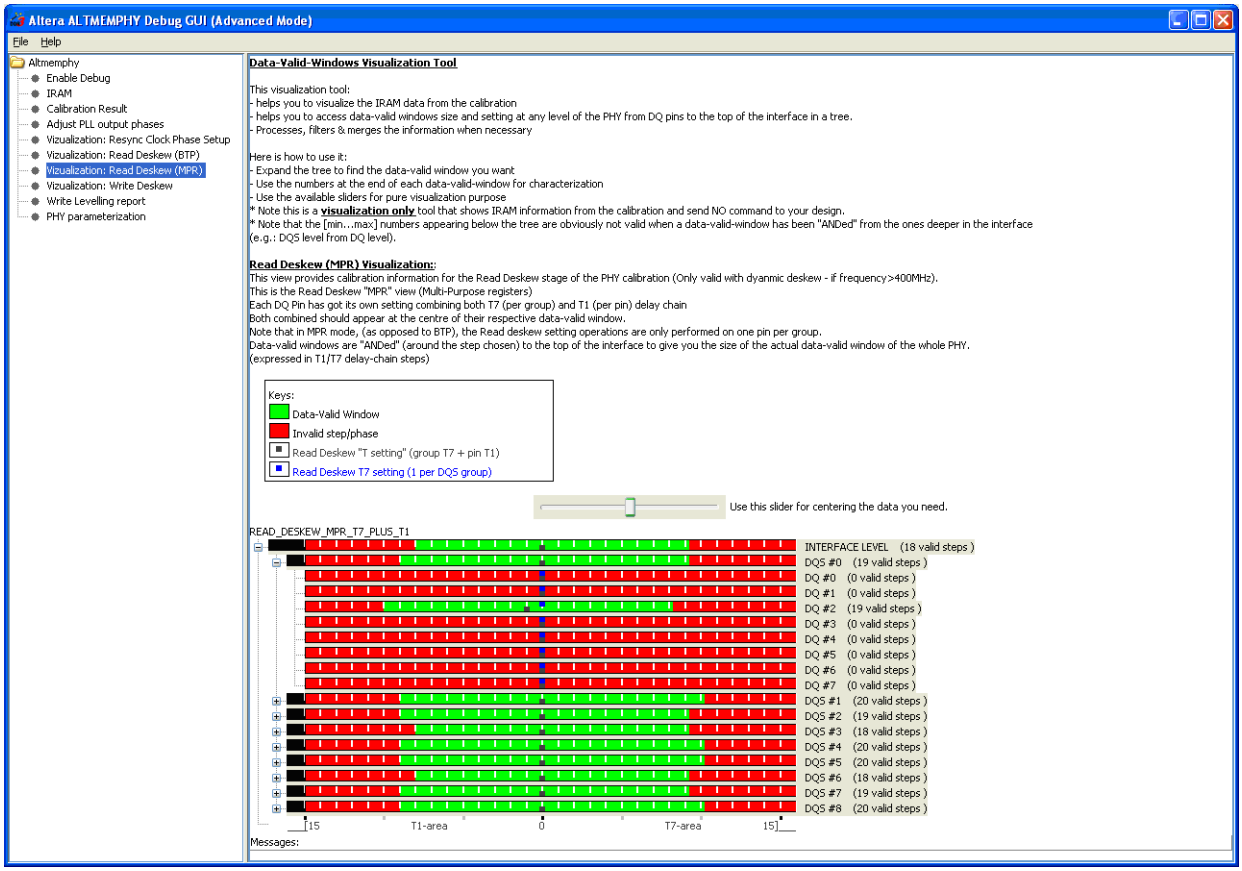

The debug toolkit additionally states the number of passing phase steps that it finds during calibration. Thus to calculate the read deskew margin in this design with a Stratix III device, the delay resolution is 50 ps, hence the margin is 17 × 50 = 850 ps or +/-425 ps.

Click **Visualization: Read Deskew (MPR)** (Figure 4–17), to show the PHY read deskew multi-purpose registers pass and fail results in an expandable tree structure:

- For the whole interface including the chosen delay (black dot)
- On a DQS group basis

- On Per DQ pin basis

**Figure 4–17. Visualization Read Deskew MPR**




This first phase of read deskew uses the prime DQ pin within each DQ group only.

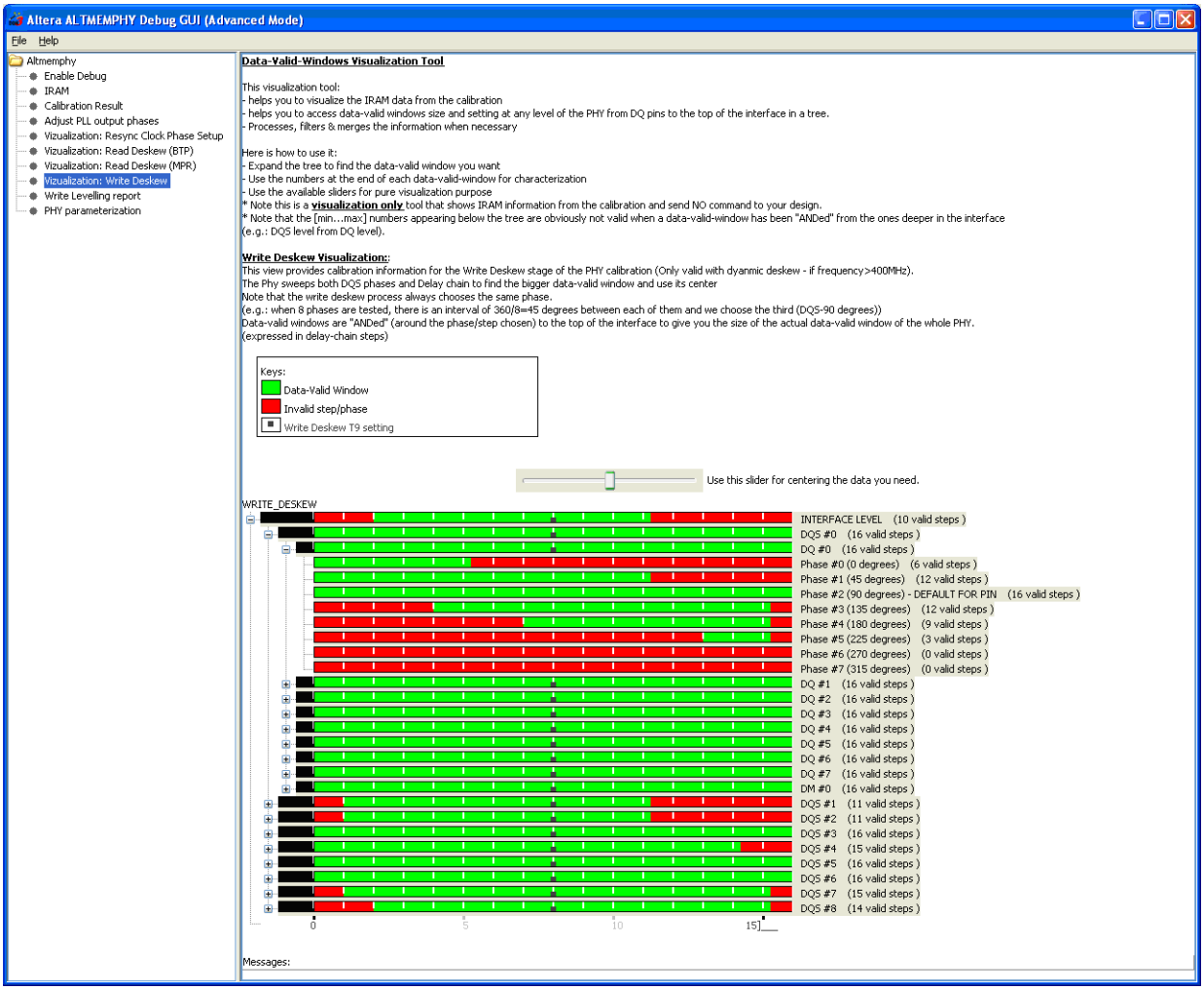
The debug toolkit additionally states the number of passing phase steps that it finds during calibration. Thus, to calculate the read deskew (MPR) margin, the delay resolution is 50 ps, hence the margin is 18 × 50 = 900 ps or +/–450ps.

Click **Visualization: Write Deskew Pan** (Figure 4–18), to show the PHY write deskew pass and fail results in an expandable tree structure:

- For the whole interface including the chosen delay (black dot)
- On a DQS group basis

- On Per DQ pin basis

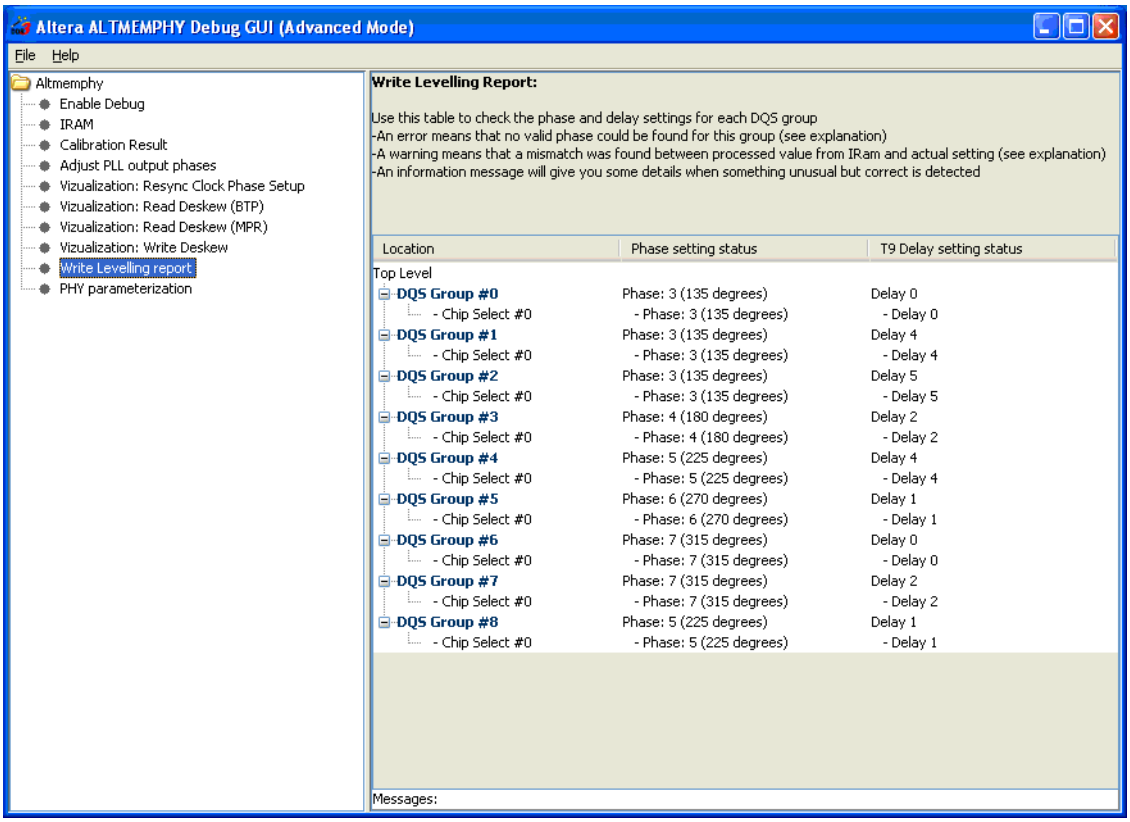
**Figure 4–18. Visualization Write Deskew Pan**




The debug toolkit additionally states the number of passing phase steps that it finds during calibration. Thus to calculate the write deskew margin, the delay resolution is 50ps, hence the margin is = 10 × 50 = 500 ps or +/–250ps

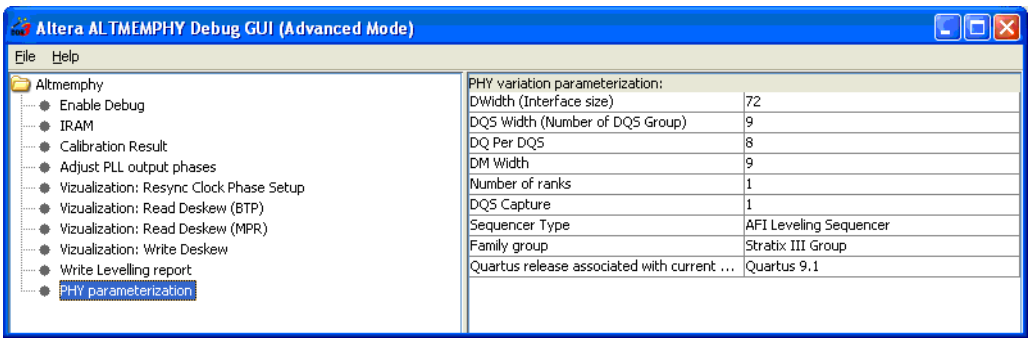
Click **Write Leveling Report** (Figure 4–19), to show the results of the write leveling. This first calibration test may often be the stage that fails in an unreliable PHY. Write leveling is performed on a per rank basis, so results are for each chip select signal in the design.

**Figure 4–19. Write Leveling Report**




Click **PHY parameterization** (Figure 4–20), to show the exact calibration configuration of the generated IP.

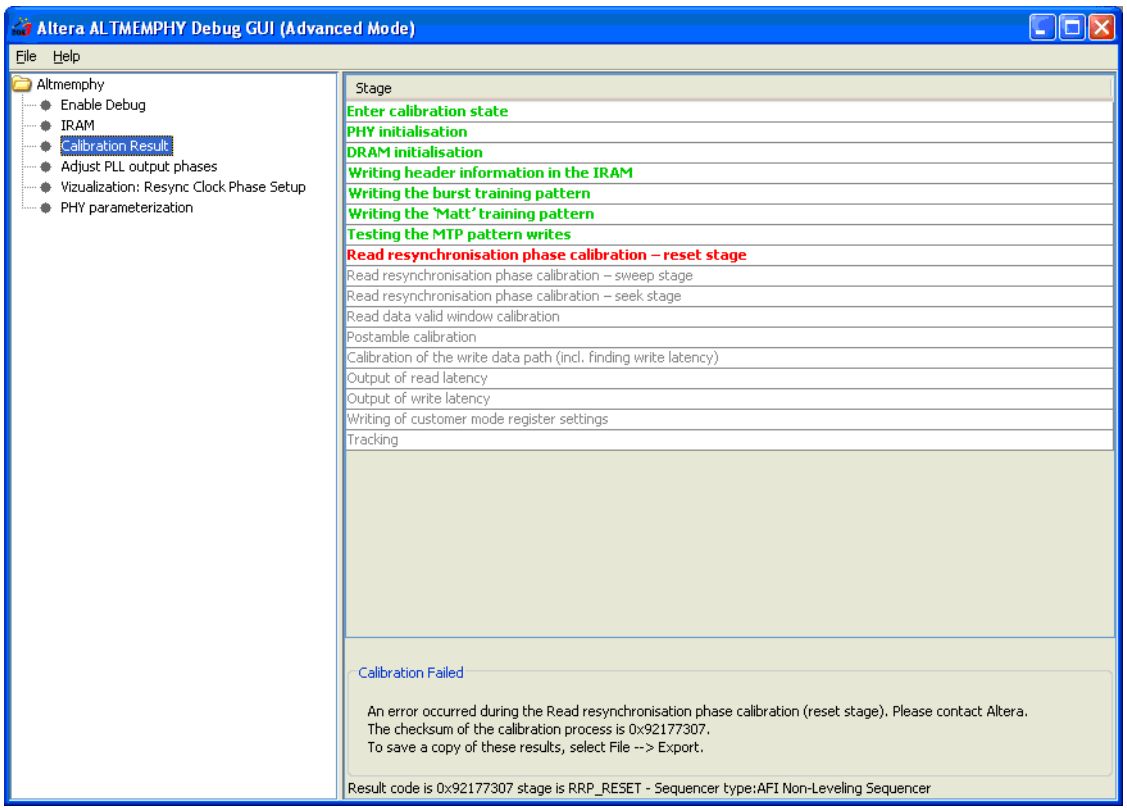
**Figure 4–20. PHY Paremeterization**



## Calibration Fails—With Leveling

If calibration fails, you see the following screen (Figure 4–21).

**Figure 4–21. Calibration Fails—With Leveling**




The stage at which calibration fails is highlighted in red; the stages that have successfully passed are in green. When possible the debug toolkit also provides a possible cause for the failure code.

This failure code is: `0x92177307`, for more information on failure codes, refer to "Understand the Checksum and Failure Code" on page 4–23.

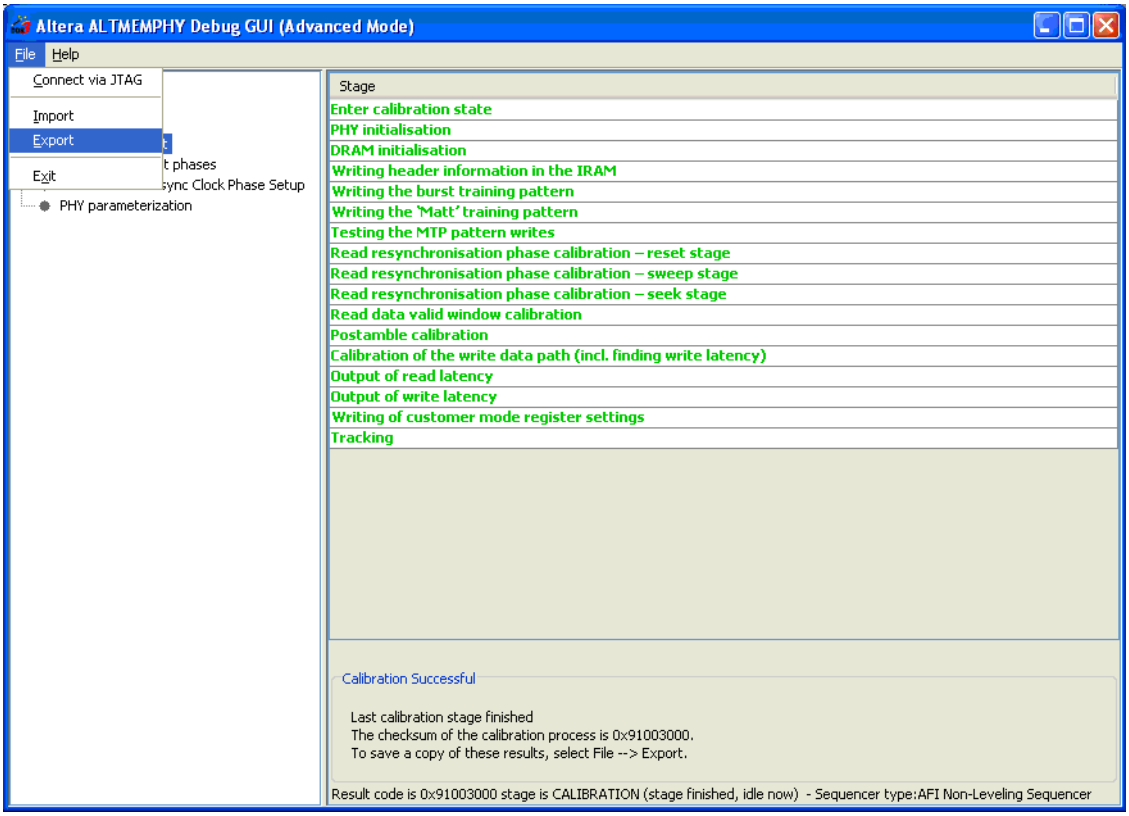
In this example, the debug toolkit suggests `One or more DDR3 Chips are being held in Reset`. As this failure is reported at the write levelling stage, look at the write leveling report to see that DQS group 8 of the interface is not responding.

## Save the Calibration Results

Often the calibration failure stage, the reported suggested failure cause, or the combined debug toolkit result and waveforms viewed in the SignalTap II logic analyzer provide enough detail to resolve the failure directly. However, you may wish to save your calibration results, so that you can refer to them later.

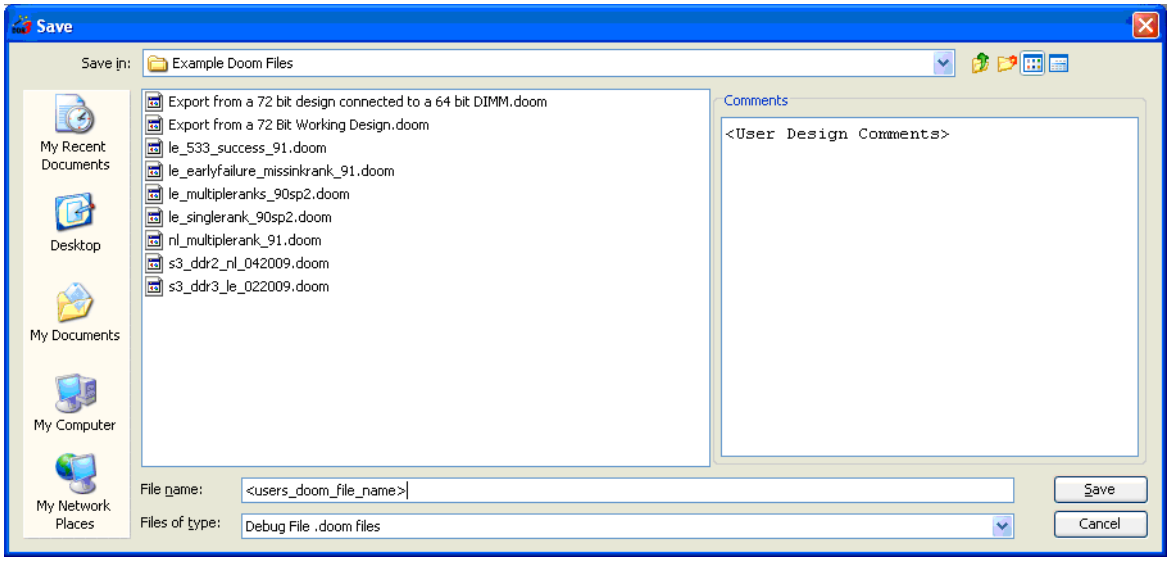
With your calibration process results still displayed on the File menu, click **Export** (Figure 4–22).

**Figure 4–22. Export**



In the **Save** dialog box (Figure 4–23), specify a file name and any comments to help with the identification and understanding of the controller configuration that you have evaluated, and details on what you may have tested.

**Figure 4–23. Save**




You can save this **.doom** file with a Quartus II archive (**.qar)** file of the test design, and a copy of the captured SignalTap II waveform files, as a single design archive. This archive provides a record of your calibration results and the ALTMEMPHY IOE configuration specific to your system, so you can refer to this data at a later date.

# Understand the Checksum and Failure Code

The debug toolkit checksum provides a direct correlation to the exact stage that calibration failed and the error code for that failure.

For example, the hexadecimal code in the format `0xAABBCCDD` represents the full 32-bit contents of the calibration status register.

During debug the code or the calibration stage and the subcode or the error code have the following definitions:

- The code and calibration stage is the first byte (`DD`) or `[7..0]`
- The subcode or the error code is the third byte (`BB`) or `[23..16]`

Take the hexadecimal value of each code and convert that to decimal. When you have these two numbers in decimal format, you can open the **failuremessages.csv** (levelling) or **failuremessages_nl.csv** (without-leveling) spreadsheet file and look up the likely causes of your calibration failure.

☞ In a passing interface, these two numbers are zero.

For the ALTMEMPHY with leveling, Table 4–2 shows the codes that correspond to the indicated calibration stages.

**Table 4–2. Leveling Calibration Stage**

| Stage | Codes |
|---|---|
| Initialization | 1 to 2 |
| Write leveling | 3 to 8 |
| Write training patterns to memory | 9 |
| Read deskew | 10 to 22 |
| Read resynch phase calibration | 23 to 25 |
| Read clock cycle calibration | 27 to 31 |
| Postamble deskew | 32 to 38 |
| Write deskew | 39 to 40 then 43 to 46 |
| DM pin deskew | 41 and 42 |
| Write datapath setup (for non deskewed datapath) | 47 |
| Write clock cycle calibration | 48 to 56 |
| Final setup | 57 to 59 |
| Tracking | 66 |
| Idle | 0 |

For the ALTMEMPHY without-leveling, Table 4–3 shows the codes that correspond to the indicated calibration stages.

**Table 4–3. Without-Leveling Calibration Stage**

| Stage | Code |
|---|---|
| Enter calibration state | 0 |
| PHY initialization | 1 |
| DRAM initialization | 2 |
| Writing header information in the internal RAM | 3 |
| Writing the burst training pattern | 4 |
| Writing more training patterns | 5 |
| Testing more training pattern writes | 6 |
| Read resynchronization phase calibration—reset stage | 7 |
| Read resynchronization phase calibration—sweep stage | 8 |
| Read resynchronization phase calibration—seek stage | 9 |
| Read data valid window calibration | 10 |
| Postamble calibration | 11 |
| Calibration of the write datapath (including finding write latency) | 12 |
| Output of read latency | 13 |
| Output of write latency | 14 |
| Writing of customer mode register settings | 15 |
| Tracking | 16 |

You can find the same information from the SignalTap II logic analyzer if the `*_ctrl.command_err`, `*_ctrl.command_result *` and `state.s_*` signals are added. The command error and state signals identity within which calibration stage the interface fails The corresponding command result then includes the same information as the suberror code.

For more information on the stages of calibration, refer to "ALTMEMPHY Calibration Stages" on page 3–1.

# 5. DDR2 and DDR3 SDRAM Controllers with UniPHY Calibration Stages

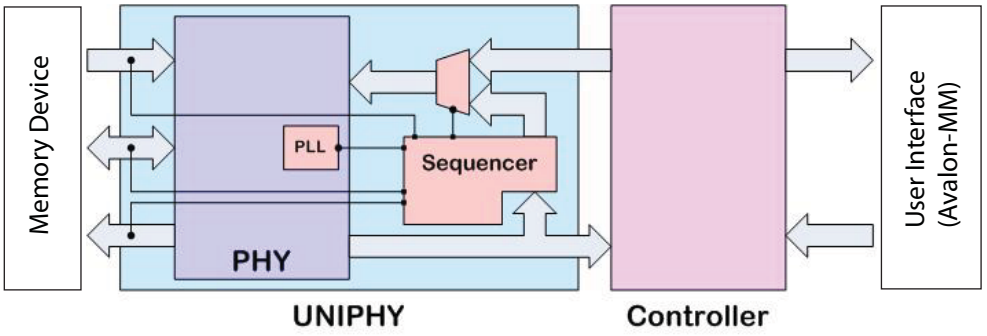
This chapter describes the calibration stages performed by the DDR2 and DDR3 SDRAM Controllers with UniPHY. This information is useful in debugging calibration failures. The chapter includes an overview of calibration, explanation of the calibration stages, and a list of generated calibration signals. The information in this chapter applies only to the Nios® II-based sequencer used in the DDR2 and DDR3 SDRAM Controllers with UniPHY versions 10.0 and 10.1. The information in this chapter applies to the Stratix III and Stratix IV device families.

## Overview

Calibration configures the memory interface (PHY and I/Os) so that data can pass reliably to and from memory. The sequencer illustrated in Figure 5–1 calibrates the PHY and the I/Os. To correctly transmit data between a memory device and the FPGA at high speed, the data must be center aligned with respect to the data clock.

Calibration also determines the delay settings needed to center-align the various data signals with respect to their clocks. I/O delay chains implement the required delays in accordance with the computed alignments. The Nios II-based sequencer performs two major tasks: FIFO buffer calibration and I/O calibration. FIFO buffer calibration adjusts FIFO lengths and I/O calibration adjusts any delay chain and phase settings to center-align data signals with respect to clock signals for both reads and writes. When the calibration process is completed, the sequencer shuts off and passes control to the memory controller.

**Figure 5–1. Sequencer in Memory Interface Logic**




## Calibration Stages

The calibration process begins when the PHY reset signal is de-asserted and the PLL and DLL are locked. The following are the stages of calibration:

1. Read calibration part one—DQS enable calibration and DQ/DQS centering
2. Write calibration part one—Leveling
3. Read calibration part two—Read latency minimization

4. Write calibration part two—DQ/DQS centering
5. Diagnostic test

For multirank calibration, every read and write command is transmitted to each rank in sequence. Each read and write test is successful only if all ranks pass the test. The sequencer calibrates to the intersection of all ranks.

## Assumptions

The calibration process assumes the following conditions; if either of these conditions is not true, calibration will likely fail in its early stages:

- The address and command paths must be functional; calibration does not tune the address and command paths. (The address and command paths are fully timing analyzed by the Quartus II software, and the slack report will be accurate, assuming the correct board timing parameters.)
- At least one bit per group must work before running per-bit-deskew calibration. (This assumption requires that DQ-to-DQS skews be within the recommended 20 ps.)

## Memory Initialization

The SDRAM is powered up according to JEDEC initialization specifications. All ranks power up simultaneously. Once powered, the SDRAM device is ready to receive mode register load commands. This part of initialization occurs separately for each rank. The sequencer issues mode register set commands on a per-chip-select basis and the memory is initialized to the user-specified settings.

## Stage 1: Read Calibration Part One—DQS Enable Calibration and DQ/DQS Centering

Read calibration occurs in two parts. Part one is DQS enable calibration with DQ/DQS centering, which happens during stage 1 of the overall calibration process; part two is read latency minimization, which happens during stage 3 of the overall calibration process.

The objectives of DQS enable calibration and DQ/DQS centering are as follows:

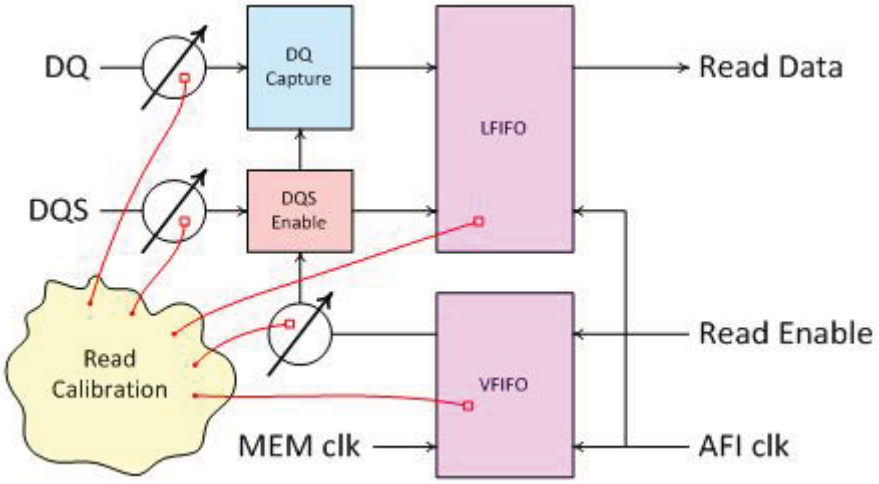
- To calculate when the read data is received after a read command is issued to setup the Data Valid Prediction FIFO (VFIFO) cycle
- To align the input data (DQ) with respect to the clock (DQS) to maximize the read margins

DQS enable calibration and DQ/DQS centering consists of the following actions:

- Guaranteed Write
- DQS Enable Phase Calibration
- DQ/DQS Centering

Figure 5–2 illustrates the components in the read data path that are calibrated in this stage. (The round knobs in the figure represent configurable hardware over which the sequencer has control.)

**Figure 5–2. Read Data Path Calibration Model**

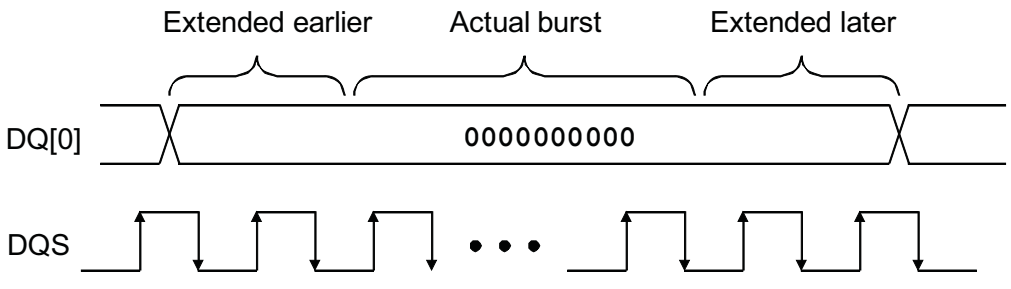



## Guaranteed Write

Since initially no communication can be reliably performed with the memory device, the sequencer uses a guaranteed write mechanism to write data into the memory device.

The guaranteed write is a write command issued with all data pins, all address and bank pins, and all command pins (except chip select) held constant. The sequencer begins toggling DQS well before the expected latch time at memory and continues to toggle DQS well after the expected latch time at memory. DQ-to-DQS relationship is not a factor at this stage because DQ is held constant. Figure 5–3 illustrates a guaranteed write of zeros.

**Figure 5–3. Guaranteed Write of Zeros**




For DQ[0], the guaranteed write performs the following operations:

1. Writes a full burst of zeros to bank 0, column 0
2. Writes a full burst of zeros to bank 0, column 1
3. Writes a full burst of ones to bank 1, column 0

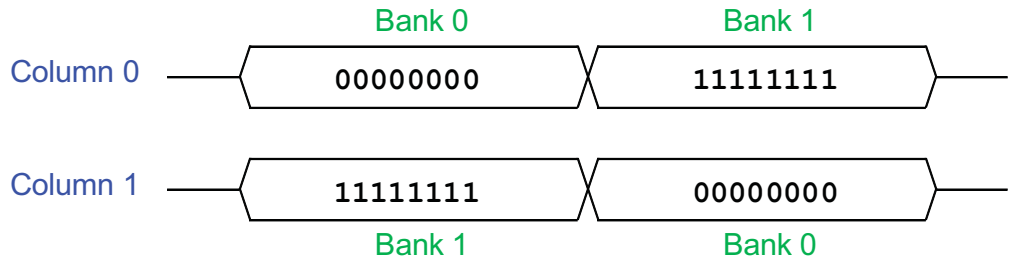
4. Writes a full burst of ones to bank 1, column 1

The guaranteed write is followed by back-to-back read operations at alternating banks, effectively producing a stream of zeros followed by a stream of ones, or vice versa. The sequencer uses the zero-to-one and one-to-zero transitions in between the two bursts to identify a correct read operation, as shown in Figure 5–4.

Although the approach described above for pin DQ[0] would work by writing the same pattern to all DQ pins, it is more effective and robust to write (and read) alternating ones and zeros to alternating DQ bits. The value of the DQ bit is still constant across the burst, and the back-to-back read mechanism works exactly as described above, except that odd DQ bits have ones instead of zeros, or vice versa.

The guaranteed write does not ensure a correct DQS-to-memory clock alignment at the memory device—DQS-to-memory clock alignment is performed later, in stage 2 of the calibration process. However, the process of guaranteed write followed by read calibration is repeated several times for different DQS-to-memory clock alignments, to ensure at least one correct alignment is found.

**Figure 5–4. Back to Back Reads on pin DQ[0]**



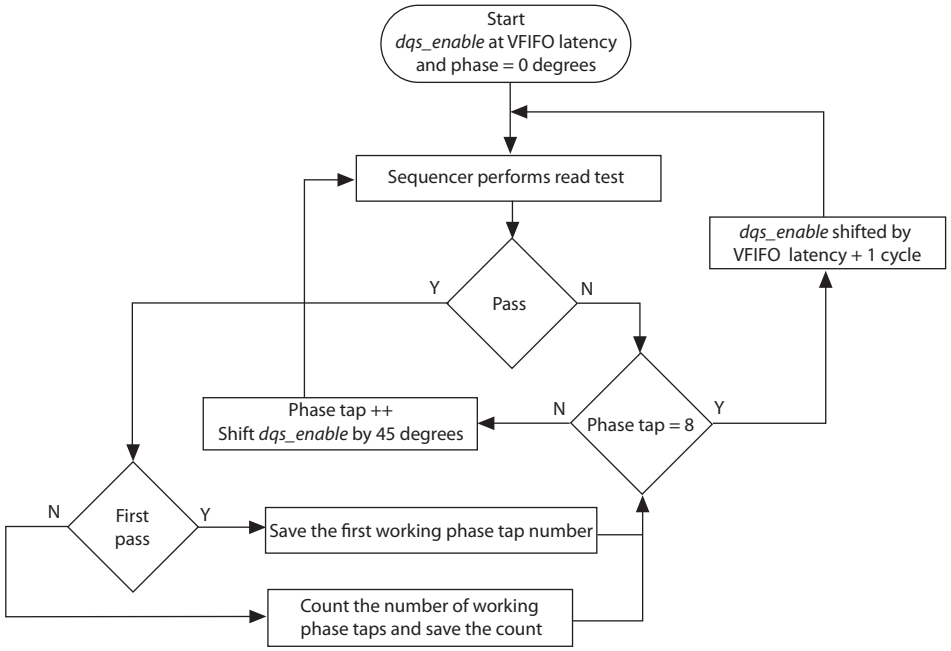

## DQS Enable Phase Calibration

DQS enable phase calibration ensures reliable capture of the DQ signal without glitches on the DQS line. At this point LFIFO is set to its maximum value to guarantee a reliable read from read capture registers to the core. Read latency is minimized later.

The VFIFO generates the DQS enable signal by shifting the controller-generated read data enable signal, `rdata_en`, by a number of full-rate clock cycles. The sequencer determines DQS enable signal phases by sweeping the DQS enable signal with coarse delays of 360° (VFIFO latency) and fine delays of 45° (phase). Fine delays are VT-compensated.

For each VFIFO latency, the sequencer sweeps through all phases until a read test fails, or until the sweep completes all available phases. For each VFIFO latency and phase setting, the sequencer issues back-to-back reads from column 0 of bank 0 and bank 1, and column 1 of bank 0 and bank 1, as shown in Figure 5–4. Two full bursts are read and compared with the reference data for each phase and delay setting. Figure 5–5 shows the steps taken in DQS enable calibration.

**Figure 5–5. DQS Enable Phase Calibration Steps**



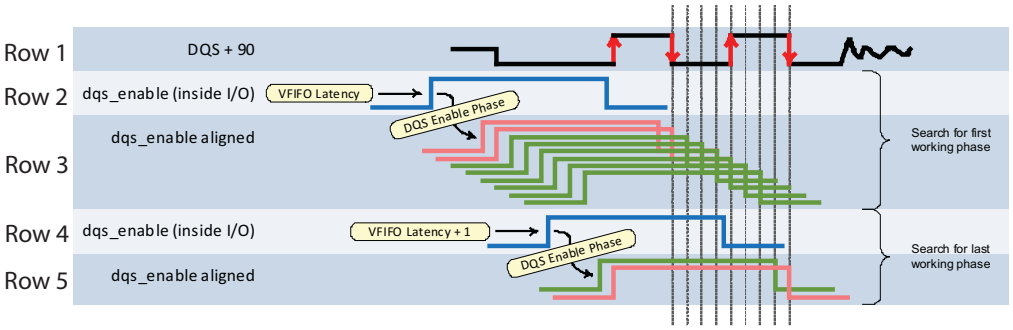

Once the sequencer identifies a range of working phases, it center-aligns the falling edge of the DQS enable signal within a valid phase range with respect to the DQS signal. At this point, per-bit deskew has not yet been performed, therefore not all bits are expected to pass the read test; however, for read calibration to succeed, at least one bit per group must pass the read test.

Figure 5–6 shows the DQS enable signal and phase relationships. The goal of VFIFO latency and phase sweeping is to find settings that satisfy the following conditions:

- The DQS enable signal rises before the first rising edge of DQS.
- The DQS enable signal is at one after the second-last falling edge of DQS.
- The DQS enable signal falls before the last falling edge of DQS.

The ideal position for the falling edge of the DQS enable signal is centered between the second-last and last falling edges of DQS.

**Figure 5–6. DQS Enable Signal and Phase Relationships**




The following points describe each row of Figure 5–6:

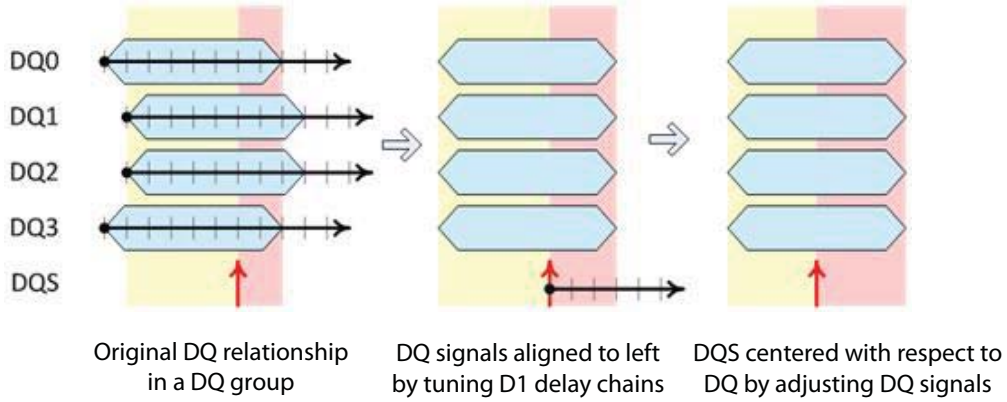
- Row 1 shows the DQS signal shifted by 90° to center-align it to the DQ data.
- Row 2 shows the raw DQS enable signal from the VFIFO.
- Row 3 shows the effect of sweeping DQS enable phases. The first two phase settings (shown in red) fail to properly gate the DQS signal because the enable signal turns off before the second-last falling edge of DQS. The next six phase settings (shown in green) gate the DQS signal successfully, with the DQS signal covering DQS from the first rising edge to the second-last falling edge.
- Row 4 shows the raw DQS enable signal from the VFIFO, increased by one clock cycle relative to Row 2.
- Row 5 shows the effect of sweeping DQS enable phases, beginning from the initial DQS enable of Row 4. The first phase setting (shown in green) successfully gates DQS, with the signal covering DQS from the first rising edge to the second-last falling edge. The second signal (shown in red), does not gate DQS successfully because the enable signal extends past the last falling edge of DQS. Any further phase adjustment would show the same failure.

## Centering DQ/DQS

The centering DQ/DQS stage attempts to align DQ and DQS signals on reads within a group. Since no per-bit deskew is performed at this stage, it is expected that at least one bit works correctly. In reality, each DQ signal within a DQS group might be skewed and consequently arrive at the FPGA at a different time. At this point, the sequencer sweeps each DQ signal in a DQ group to align them, by adjusting DQ input delay chains (D1); D1 has 15 taps, with a delay of 50 ps each.

Figure 5–7 illustrates a four DQ/DQS group per-bit-deskew and centering.

**Figure 5–7. Per-bit Deskew**




To align and center DQ and DQS, the sequencer finds the right edge of DQ signals with respect to DQS by sweeping DQ signals within a DQ group to the right until a failure occurs. In Figure 5–7, DQ0 and DQ3 fail after six taps to the right; DQ1 and DQ2 fail after 5 taps to the right. To align the DQ signals, DQ0 and DQ3 are shifted to the right by 1 tap.

To find the center of DVW, the DQS signal is shifted to the right until a failure occurs. In Figure 5–7, a failure occurs after 3 taps, meaning that there are 5 taps to the right edge and 3 taps to the left edge. To center-align DQ and DQS, the sequencer shifts the aligned DQ signal by 1 more tap to the right.

☞ The sequencer does not adjust DQS directly; instead, the sequencer center-aligns DQS with respect to DQ by delaying the DQ signals.

## Stage 2: Write Calibration Part One—Leveling

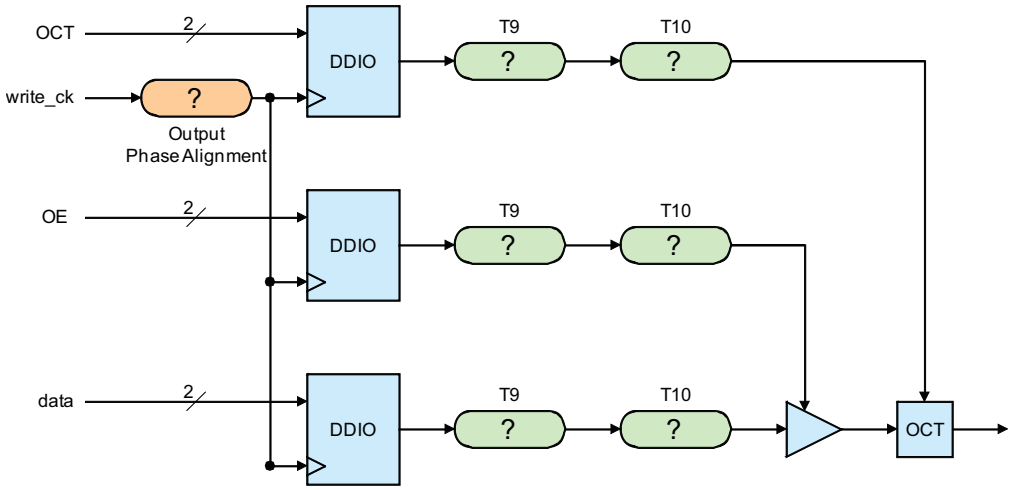
The objectives of the write leveling stage are to align DQS to the memory clock at each memory device, and to compensate for address, command, and memory clock skew at each memory device.

Write leveling is required for fly-by topologies where address, command, and clock signals are routed sequentially to all individual memory devices. The implication of such routing is that address, command, and clock signals arrive at each memory component at different times. Memory clock signals and DQ/DM and DQS signals have specific relationships mandated by the memory device. The PHY must ensure that these relationships are met by skewing DQ/DM and DQS signals. The relationships between DQ/DM and DQS and memory clock signals must meet the tDQSS, tDSS, and tDSH timing constraints.

The sequencer calibrates the write data path using a variety of random burst patterns to compensate for the jitter on the output data path. Simple write patterns are insufficient to ensure a reliable write operation because they might cause imprecise DQS-to-CK alignments, depending on the actual capture circuitry on a memory device. The write patterns in the write leveling stage have a burst length of 8, and are generated by a linear feedback shift register in the form of a pseudo-random binary sequence.

The write data path architecture is the same for DQ, DM, and DQS pins. Figure 5–8 illustrates the write data path for a DQ signal. The phase coming out of the Output Phase Alignment block can be set to different values to center-align DQS with respect to DQ, and it is the same for data, OE, and OCT of a given output.

**Figure 5–8. Write Data Path**




In write leveling, the sequencer performs write operations with different delay and phase settings, followed by a read. The sequencer can implement any phase shift between 0° and 720°. The sequencer uses the Output Phase Alignment for coarse delays and D5 and D6 for fine delays; D5 has 15 taps of 50 ps each, and D6 has 7 taps of 50 ps each. The DQS signal phase is held at +90° with respect to DQ signal phase.

Coarse delays are called *phases*, and fine delays are called *delays*; phases are PVT compensated, delays are not.

The sequencer writes and reads back several burst-length-8 patterns. Since per-bit deskew has not yet been performed, not all bits are expected to pass the write test; however, for write calibration to succeed, at least one bit per group must pass the write test. The test begins by shifting the DQ/DQS phase until the first write operation completes successfully. The DQ/DQS signals are then delayed to the left by D5 and D6 to find the left edge for that working phase. Then DQ/DQS phase continues the shift to find the last working phase. For the last working phase, DQ/DQS is delayed in 50 ps steps to find the right edge of the last working phase.

The sequencer sweeps through all possible phase and delay settings for each DQ group where the data read back is correct, to define a window within which the PHY can reliably perform write operations. The sequencer picks the closest value to the center of that window as the phase/delay setting for the write data path.

## Stage 3: Read Calibration Part Two—Read Latency Minimization

At this stage of calibration the sequencer adjusts LFIFO latency to determine the minimum read latency that guarantees correct reads.

### Read Latency Tuning

In general, DQ signals from different DQ groups may arrive at the FPGA in a staggered fashion. In a DIMM or multiple memory device system, the DQ/DQS signals from the first memory device arrive sooner, while the DQ/DQS signals from the last memory device arrive the latest at the FPGA.

LFIFO transfers data from the capture registers in IOE to the core and aligns read data to the AFI clock. Up to this point in the calibration process, the read latency has been a maximum value set initially by LFIFO; now, the sequencer progressively lowers the read latency until the data can no longer be transferred reliably. The sequencer then increases the latency by one cycle to return to a working value and adds an additional cycle of margin to assure reliable reads.

## Stage 4: Write Calibration Part Two—DQ/DQS Centering

The process of DQ/DQS centering in write calibration is similar to that performed in read calibration, except that write calibration is performed on the output path, using D5 and D6 delay chains.

## Stage 5: Diagnostic Test

The diagnostic test is the final stage of calibration before the sequencer passes control to the memory controller. At this stage, the sequencer writes and reads more complex patterns to and from the memory device. The goals of this stage are to verify that calibration is successful and robust, and to estimate the read and write margins under more noisy conditions.

The sequencer writes and reads a sequential 8-bit burst pattern from 00000001 to 11111111 on columns 0 and 1 of bank 0 and column 0 and 1 of bank 1.

## Calibration Signals

Table 5–1 lists signals produced by the calibration process.

**Table 5–1. Calibration Signals (Part 1 of 2)**

| Signal | Description |
|---|---|
| afi_cal_fail | Asserts high if calibration fails. |
| afi_cal_success | Asserts high if calibration is successful. |

**Table 5–1. Calibration Signals (Part 2 of 2)**

| Signal | Description |
|---|---|
| afi_cal_debug_info | A 32-bit signal containing additional data relating to the success or failure of calibration. Only 16 bits are used.<br>■ In cases of calibration success:<br>■ DEBUG_INFO[7:0] is a binary representation of the read margin and DVW width<br>■ DEBUG_INFO[15:8] is a binary representation of the write margin and DVW width<br>■ In cases of calibration failure:(1)<br>■ DEBUG_INFO[7:0] is a binary representation of the stage at which failure occurred<br>■ DEBUG_INFO[15:8] is a binary representation of the group that was being calibrated at the time of failure |

**Notes for Table 5–1:**

(1) In cases of calibration failure, DEBUG_INFO values are valid only if failure occurred during the Read Calibration, Write Leveling, or Write Per-bit deskew and Centering stages of calibration. DEBUG_INFO values are not valid if failure occurred during Read Latency Tuning.

This chapter provides additional information about the document and Altera.

## Revision History

The following table shows the revision history for this section.

| Date | Version | Changes Made |
|---|---|---|
| December 2010 | 2.1 | Added Arria II GZ and Stratix V information. |
| July 2010 | 2.0 | Added information about UniPHY-based IP and controllers. |
| January 2010 | 1.2 | Corrected minor typos. |
| December 2009 | 1.1 | Added *Timing Deration* chapter. |
| November 2009 | 1.0 | First published. |

## How to Contact Altera

To locate the most up-to-date information about Altera products, refer to the following table.

| Contact *(1)* | Contact Method | Address |
|---|---|---|
| Technical support | Website | www.altera.com/support |
| Technical training | Website | www.altera.com/training |
| | Email | custrain@altera.com |
| Product literature | Website | www.altera.com/literature |
| Non-technical support (General) | Email | nacomp@altera.com |
| (Software Licensing) | Email | authorization@altera.com |

**Note to Table:**

(1) You can also contact your local Altera sales office or sales representative.

## Typographic Conventions

The following table shows the typographic conventions this document uses.

| Visual Cue | Meaning |
|---|---|
| **Bold Type with Initial Capital Letters** | Indicate command names, dialog box titles, dialog box options, and other GUI labels. For example, **Save As** dialog box. For GUI elements, capitalization matches the GUI. |
| **bold type** | Indicates directory names, project names, disk drive names, file names, file name extensions, software utility names, and GUI labels. For example, **\qdesigns** directory, **D:** drive, and **chiptrip.gdf** file. |
| *Italic Type with Initial Capital Letters* | Indicate document titles. For example, *Stratix IV Design Guidelines*. |

# 6. DDR2 and DDR3 SDRAM Controllers with UniPHY EMIF Toolkit

This chapter describes the UniPHY External Memory Interface Toolkit. It explains how to enable, launch, and run the toolkit, and provides a guide for interpreting results and troubleshooting. This toolkit is a GUI and Tcl-based interface that runs on your PC and enables you to debug your external memory interface design on the circuit board, retrieve calibration status, and perform margining activities.

This toolkit supports only the DDR2 and DDR3 SDRAM Controllers with UniPHY.

## Feature Description

The External Memory Interface Toolkit consists of the following parts:

- DDR2 and DDR3 SDRAM Controllers with UniPHY
- Avalon Memory-Mapped (Avalon-MM) slave interface
- JTAG Avalon master

The EMIF toolkit allows you to display information about your external memory interface and generate calibration and margining reports. The toolkit can aid in diagnosing the type of failure that may be occurring in your external memory interface, and help identify areas of reduced margin that might be potential failure points.

## Using the External Memory Interface Toolkit

Using the external memory interface toolkit to analyze your external memory interface involves the following steps:

1. (Optional) Generating your IP core with the CSR port enabled and with the CSR communication interface type properly set.
2. Launching the debug toolkit.
3. Specifying project settings.
4. Using the toolkit to view information about your interface.
5. Interpreting results and troubleshooting your interface.

The following sections discuss each of the above steps in detail.

## Enabling Communication with the Controller via the CSR Port

Optionally, you can enable communication between the EMIF toolkit and the memory controller through the Configuration and Status Register (CSR) port on the controller.

You do not have to enable the CSR port in order to use the toolkit's report-generation functions; however, enabling the CSR port provides the following additional capabilities:

- allows the toolkit to verify memory device operation by determining whether the controller receives DQS edges from the memory device
- allows the toolkit to issue soft resets to the memory interface
- allows the toolkit to monitor PLL locked status

Before the toolkit can communicate with the controller through the CSR port, you must enable the CSR port from the Controller Settings tab of your memory interface parameter editor, as described in the following steps:

1. Open the DDR2 or DDR3 SDRAM Controller with UniPHY parameter editor from the MegaWizard Plug-in Manager, SOPC Builder, or Qsys.
2. Under **Advanced Controller Features** on the **Controller Settings** tab, specify the following options:
   a. Turn on **Enable Configuration and Status Register Interface**.
   b. Set **CSR port host interface** to INTERNAL_JTAG.
3. Regenerate the IP core.
4. Compile your design in the Quartus II software.
5. Program the Altera FPGA device using the programming file from your project.

Figure 6–1 illustrates the external memory interface components with the optional CSR port and JTAG Avalon master configured.

**Figure 6–1. External Memory Interface with CSR Port Configured**

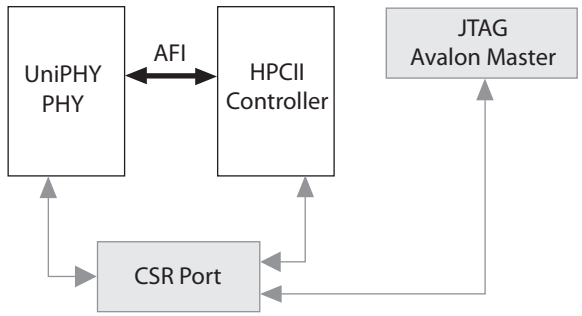


Table 6–1 shows the UniPHY and High Performance Controller II CSR address map.

**Table 6–1. CSR Address Map**

| Address | Bit | Name | Default | Access | Description |
|---|---|---|---|---|---|
| 0x001 | 15:0 | Reserved | 0 | — | Reserved for future use. |
| | 31:16 | | | | |
| 0x002 | 15:0 | | | | |
| | 31:16 | | | | |
| 0x004 | 0 | soft_reset | — | WO | Initiates a soft reset of the memory interface. This bit is automatically deasserted when reset is completed. |
| | 23:1 | Reserved | 0 | — | Reserved for future use. |
| | 24 | afi_cal_success | — | RO | Reports the value of the UniPHY `afi_cal_success` output. Writing to this bit has no effect. |
| | 25 | afi_cal_fail | — | RO | Reports the value of the UniPHY `afi_cal_fail` output. Writing to this bit has no effect. |
| | 31:26 | Reserved | 0 | — | Reserved for future use. |
| 0x005 | 7:0 | fom_in | — | RO | The figure of merit *(1)* for read as calculated by the sequencer. Only applicable if `afi_cal_success` is 1. |
| | 15:8 | Reserved | 0 | — | Reserved for future use. |
| | 23:16 | fom_out | — | RO | The figure of merit *(1)* for write as calculated by the sequencer. Only applicable if `afi_cal_success` is 1. |
| | 31:24 | Reserved | 0 | — | Reserved for future use. |
| 0x006 | 7:0 | init_failing_stage | — | RO | Initial failing error stage of calibration. Only applicable if `afi_cal_fail` is 1. |
| | 15:8 | Reserved | 0 | — | Reserved for future use |
| | 23:16 | init_failing_group | — | RO | Initial failing error group of calibration. Only applicable if `afi_cal_fail` is 1. |
| | 31:24 | Reserved | 0 | — | Reserved for future use |
| 0x007 | 31:0 | dqs_detect | — | RO | Indicates whether DQS edges have been identified for each group. Each bit corresponds to 1 DQS group. |

**Note to Table 6–1:**

(1) The figure of merit (FOM) is a measure of the health of the read (or write) interface; it is calculated as the sum over all groups of the minimum margin on DQ plus the margin on DQS, divided by 2.

## Launching the External Memory Interface Debug Toolkit

To launch the debug toolkit from the Quartus II software, perform the following steps:

1. Program the Altera FPGA device using the programming file from your project.
2. To open the External Memory Interface Toolkit in the System Console, click **External Memory Interface Toolkit** on the **Tools** menu in the Quartus II software.
3. On the File menu in the System Console window, select **Load Design** to load your Quartus II Project File (**.qpf**) into the EMIF toolkit. When your project is loaded, your design folder appears under **design_instances** in the System Explorer tree.

4. In the System Explorer tree, right-click the **<*instance_name*>.jdi** file in your design folder, and click **Link design instance to device** to link the design instance to the target device.
5. Under **Hardware Setup**, click **Apply Linked Design** to load the project. Your interface connection now appears under **New Memory Interface Connection**.
6. Under **Hardware Setup** on the **External Memory Interface Toolkit** tab, select your hardware and device. The **Hardware** list shows all the detected connections between your board and the PC; the **Device** list shows all the devices on your board, detected by the selected connection
7. If you have more than one interface, select the interface you want from the **Memory Interface** list, under **New Memory Interface Connection**.

   Once you have selected your interface, the **New Memory Interface Connection** group box displays the UniPHY instance name, and—if you have set up communication with the CSR port—the CSR instance name and PLL status.
8. Under **New Memory Interface Connection**, click **Establish Connection** to create a tab for your external memory interface.

   You can optionally repeat this step to create tabs for any other external memory interfaces in your design.

## Specifying Project Settings

Before rerunning calibration or generating reports, you must establish project settings to allow for correct calibration and margining results. To establish the project settings, perform the following steps:

1. On the **Project Settings** tab, ensure the **Settings Type** field is set to **Device Settings**.
2. Under **Project FPGA Settings**, select values for the **FPGA Family** and **FPGA Speedgrade**.
3. Under **Project PLL Settings**, type your memory interface frequency.
4. Click **Update Project Settings** to save the information for your project in memory.

## Viewing Information About Your External Memory Interface

The following steps explain how to use the External Memory Interface Toolkit to view information about your interface.

1. On the **Memory Interface Status and Control** tab, click **Generate Calibration Report** to generate detailed calibration information for your interface.

2. To view the calibration report, on the **Reports** tab, under **Report Type**, select **Calibration report per DQ group** or **Calibration report**.

   The Calibration report per DQ group lists read and write data valid windows and calibration status for each group and rank.

   The Calibration report is comprehensive, and includes the following information:

   - Group and rank mask status
   - Calibration status
   - Margins observed during calibration, per DQ group
   - DQ and DQS I/O settings

3. To view read and write margining and data valid window information, click **Generate margining report**.
4. Under **Report Type**, select **Margining Report** or **DVW Report**.

   The Margining report lists each DQ pin and DQS group margin observed following calibration. Any reduced margins on DQ pins can be indicative of possible problems with the PCB layout of the memory interface.

   The DVW report is a graphical representation of the Margining report, and can help you visualize the range of the data valid window per DQ pin. The black line in the report represents the point within the window to which the DQ pin is calibrated.

5. Click **Save Report** to save all generated reports in HTML format on your computer.

    You cannot save the DVW report.

6. On the **Summary** tab, select **Connection Summary** to review the interface and its connection properties.
7. Select **Result Summary** to display read and write latency values, as well as the DQS Captured Status. (If your design does not contain a CSR port, or you have not enabled the CSR port for use, the DQS Captured Status is not available.)

    For information about using the CSR port, refer to "Enabling Communication with the Controller via the CSR Port" on page 6–2.

## Interpreting Results and Troubleshooting

This section provides information on how to interpret the results returned by the toolkit.

### Calibration Successful

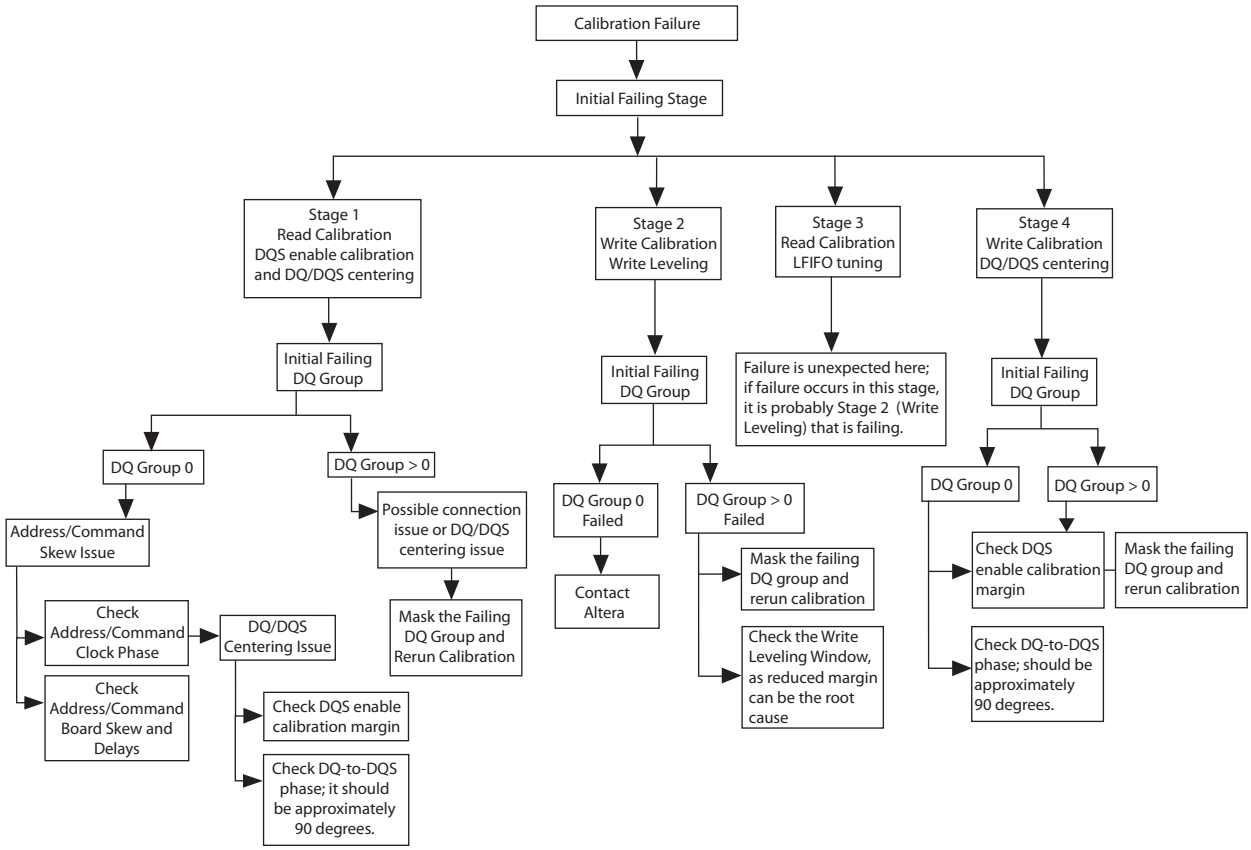
When both calibration and post-calibration tests complete successfully, you can use the debug toolkit to retrieve information about your memory interface and to identify areas of reduced margin. DQ pins with small margins tend to be the pins most susceptible to the effects of data corruption, board delays, and noise.

## Calibration Failed

In the event of calibration failure, you should check the DQS Capture Status in the Summary results to verify that the memory interface is functioning. If DQS edges are not detected, it is likely that the memory interface is not functioning. If DQS edges are detected, refer to Figure 6–2 to assist in troubleshooting your design.

Figure 6–2 shows a flowchart of debugging tips to assist in resolving calibration failure.

**Figure 6–2. Debugging Tips**




For detailed information about each calibration stage, refer to Chapter 5, "DDR2 and DDR3 SDRAM Controllers with UniPHY Calibration Stages".

This chapter provides additional information about the document and Altera.

## Document Revision History

The following table shows the revision history for this document.

DDR2 and DDR3 SDRAM Controllers with UniPHY Calibration Stages

| Date | Version | Changes |
|---|---|---|
| December 2010 | 2.1 | ■ Added new chapter: "DDR2 and DDR3 SDRAM Controllers with UniPHY Calibration Stages"<br>■ Added new chapter: "DDR2 and DDR3 SDRAM Controllers with UniPHY EMIF Toolkit" |
| July 2010 | 2.0 | Updated for 10.0 release. |
| January 2010 | 1.2 | Corrected minor typos. |
| December 2009 | 1.1 | Added *Debug Toolkit for DDR2 and DDR3 SDRAM High-Performance Controllers* chapter and *ALTMEMPHY Calibration Stages* chapter. |
| November 2009 | 1.0 | First published. |

## How to Contact Altera

To locate the most up-to-date information about Altera products, refer to the following table.

| Contact *(1)* | Contact Method | Address |
|---|---|---|
| Technical support | Website | www.altera.com/support |
| Technical training | Website | www.altera.com/training |
| | Email | custrain@altera.com |
| Product literature | Website | www.altera.com/literature |
| Non-technical support (General) | Email | nacomp@altera.com |
| (Software Licensing) | Email | authorization@altera.com |

**Note to Table:**

(1) You can also contact your local Altera sales office or sales representative.

# Typographic Conventions

The following table shows the typographic conventions this document uses.

| Visual Cue | Meaning |
|---|---|
| **Bold Type with Initial Capital Letters** | Indicate command names, dialog box titles, dialog box options, and other GUI labels. For example, **Save As** dialog box. For GUI elements, capitalization matches the GUI. |
| **bold type** | Indicates directory names, project names, disk drive names, file names, file name extensions, software utility names, and GUI labels. For example, **\qdesigns** directory, **D:** drive, and **chiptrip.gdf** file. |
| *Italic Type with Initial Capital Letters* | Indicate document titles. For example, *Stratix IV Design Guidelines*. |
| *italic type* | Indicates variables. For example, *n* + 1.<br>Variable names are enclosed in angle brackets (< >). For example, *<file name>* and *<project name>***.pof** file. |
| Initial Capital Letters | Indicate keyboard keys and menu names. For example, the Delete key and the Options menu. |
| "Subheading Title" | Quotation marks indicate references to sections within a document and titles of Quartus II Help topics. For example, "Typographic Conventions." |
| `Courier type` | Indicates signal, port, register, bit, block, and primitive names. For example, `data1`, `tdi`, and `input`. The suffix n denotes an active-low signal. For example, `resetn`.<br>Indicates command line commands and anything that must be typed exactly as it appears. For example, `c:\qdesigns\tutorial\chiptrip.gdf`.<br>Also indicates sections of an actual file, such as a Report File, references to parts of files (for example, the AHDL keyword `SUBDESIGN`), and logic function names (for example, `TRI`). |
| ↵ | An angled arrow instructs you to press the Enter key. |
| 1., 2., 3., and<br>a., b., c., and so on | Numbered steps indicate a list of items when the sequence of the items is important, such as the steps listed in a procedure. |
| ■ ■ ■ | Bullets indicate a list of items when the sequence of the items is not important. |
|  | The hand points to information that requires special attention. |
|  | A question mark directs you to a software help system with related information. |
|  | The feet direct you to another document or website with related information. |
|  | A caution calls attention to a condition or possible situation that can damage or destroy the product or your work. |
|  | A warning calls attention to a condition or possible situation that can cause you injury. |
|  | The envelope links to the Email Subscription Management Center page of the Altera website, where you can sign up to receive update notifications for Altera documents. |

| Visual Cue | Meaning |
|---|---|
| *italic type* | Indicates variables. For example, *n* + 1.<br>Variable names are enclosed in angle brackets (< >). For example, *<file name>* and *<project name>*.**pof** file. |
| Initial Capital Letters | Indicate keyboard keys and menu names. For example, the Delete key and the Options menu. |
| "Subheading Title" | Quotation marks indicate references to sections within a document and titles of Quartus II Help topics. For example, "Typographic Conventions." |
| `Courier type` | Indicates signal, port, register, bit, block, and primitive names. For example, `data1`, `tdi`, and `input`. The suffix `n` denotes an active-low signal. For example, `resetn`.<br>Indicates command line commands and anything that must be typed exactly as it appears. For example, `c:\qdesigns\tutorial\chiptrip.gdf`.<br>Also indicates sections of an actual file, such as a Report File, references to parts of files (for example, the AHDL keyword `SUBDESIGN`), and logic function names (for example, `TRI`). |
| ↵ | An angled arrow instructs you to press the Enter key. |
| 1., 2., 3., and<br>a., b., c., and so on | Numbered steps indicate a list of items when the sequence of the items is important, such as the steps listed in a procedure. |
| ■ ■ ■ | Bullets indicate a list of items when the sequence of the items is not important. |
|  | The hand points to information that requires special attention. |
|  | A question mark directs you to a software help system with related information. |
|  | The feet direct you to another document or website with related information. |
|  | A caution calls attention to a condition or possible situation that can damage or destroy the product or your work. |
|  | A warning calls attention to a condition or possible situation that can cause you injury. |
|  | The envelope links to the Email Subscription Management Center page of the Altera website, where you can sign up to receive update notifications for Altera documents. |

# External Memory Interface Handbook Volume 5

## Section I. Implementing Custom Memory Interface PHY

101 Innovation Drive
San Jose, CA 95134
www.altera.com

EMI_CUSTOM-1.1

Subscribe

# Contents

## Additional Information

# 1. Creating a Custom PHY

This chapter describes the FPGA design flow to implement a memory interface datapath (PHY) using Stratix® III and Stratix IV devices and the ALTDLL and ALTDQ_DQS megafunctions. You can use this method as an alternative to using the available external memory interface IP. You can then connect this PHY with a custom memory controller to interface with other memory components.

Altera recommends using the available external memory IPs instead of using the ALTDLL and ALTDQ_DQS megafunctions whenever possible.

The ALTDLL megafunction implements the dedicated DQS circuitry, while the ALTDQ_DQS megafunction implements the read and write PHY required for the interface. For more information about these megafunctions, refer to the *ALTDLL and ALTDQ_DQS Megafunctions User Guide*.

You must constrain the timing of the PHY when using ALTDLL and ALTDQ_DQS megafunctions. This chapter does not cover details of the timing constraints; however, it includes information about timing analysis and board-level constraints that demonstrate and validate the interface.

The design flow steps in this chapter focus on the "Instantiate PHY and Controller" step of the *Recommended Design Flow* chapter of the *External Memory Handbook*. You can adapt the steps for a DDR2 SDRAM interface to create other types of memory interfaces.

After selecting the appropriate device and memory type, create a project in the Quartus II software that targets the device and memory type. When instantiating the datapath for DDR and DDR2 SDRAM interfaces in Stratix III and Stratix IV devices, Altera recommends using the ALTDLL and ALTDQ_DQS megafunctions for the datapath of custom external memory interfaces that are not supported by the ALTMEMPHY or UniPHY IP.

You use the following megafunctions to create a complete memory interface PHY:

- ALTDLL megafunction
- ALTDQ_DQS megafunction
- ALTPLL megafunction
- ALTIOBUF megafunction
- ALTOCT megafunction

The following sections describe the work flow when you instantiate PHY via ALTDLL and ALTDQ_DQS megafunctions and your custom-designed controller in a Quartus II project:

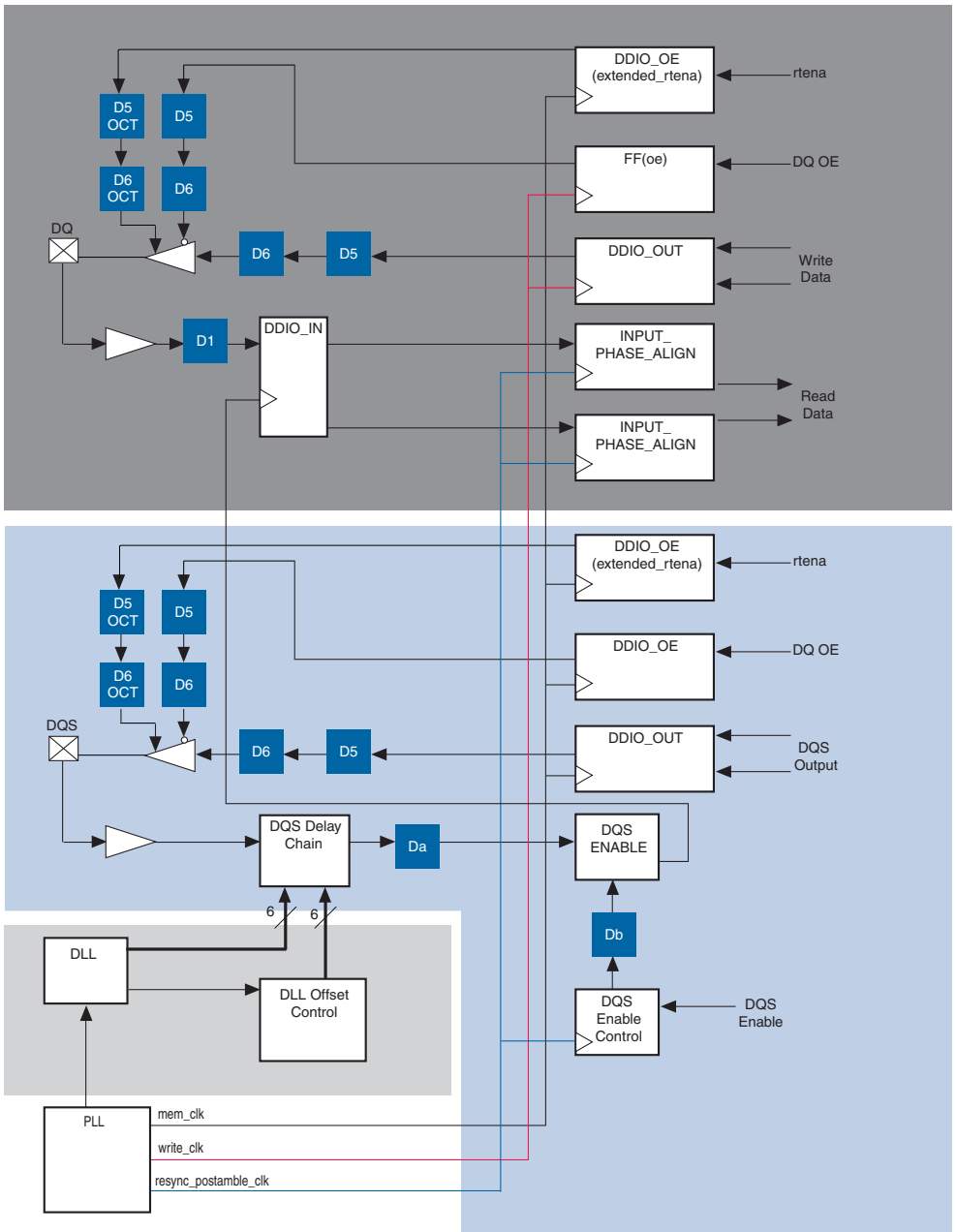
# Instantiate the ALTPLL Megafunction

The ALTPLL instance provides the necessary clocking scheme for the custom PHY. Clocking schemes for the custom external memory interface are important. You must understand what clocks are required for a full-rate or half-rate interface. Both interfaces require different clocking schemes.

## Clocking Scheme Requirement for Full-Rate Interface

Figure 1–1 on page 1–2 shows ta sample full-rate interface.

**Figure 1–1. Example Full-Rate Interface**



The PLL instance should generate the following five clock outputs for the full-rate interface (refer to Figure 1–1):

- DLL Input clock
- Resynchronize postamble clock (`resync_postamble_clk`)
- Write Clock (`write_clk`)
- Memory clock (`mem_clk`)
- Address/command clock

The following sections descibe each of the five clock outputs for the full-rate interface.

## DLL Input Clock

This clock toggles the DLL block. The DLL clock is equivalent to the full rate of the interface and has a 0° phase shift. You must connect the DLL block directly to a single dedicated PLL clock output.

## Resynchronize Postamble Clock (resync_postamble_clk)

The `resync_postamble_clk` clock is an optional clock.

You can use this clock for two purposes:

- Clock the DQS enable control block. This block is used to disable the DQS input strobe when the strobe goes to Hi-Z (after a DDR read postamble). This is part of the DQS input path.
- Clock the INPUT_PHASE_ALIGN block. This block phase shifts the input signal and is primarily used to match the arrival delay of the DQS to the latest arrival delay of a DQS from the DDR or DDR2 DIMM. It is also used to resynchronize the data read from the DDIO_IN block. This is part of the DQ input path.

The `resync_postamble_clk` clock should be the full rate of the interface and have dynamic phase. You can set the dynamic phase capability for the PLL clock outputs with the ALTPLL megafunction. The correct `resync_postamble_clk` clock phase is crucial to correct data transfer. The ALTDQ_DQS megafunction cannot determine the phase that the `resync_postamble_clk` clock should have. Therefore, you must solve this with round-trip delay analysis or creating a custom data training circuitry to write and read back a training pattern to and from the memory device, and then dynamically adjust the resynchronization clock phase of the PLLs to find a good working phase.

## Write Clock (write_clk)

The `write_clk` clock clocks the DDIO_OUT block that is part of the DQ output path. The clock is used for writing data to the DQ pins and also to clock the FF (oe) block (refer to Figure 1–1 on page 1–2), which is part of the DQ OE path. The FF (oe) block acts as the output enable for the DQ output path.

The `write_clk` clock should be the full rate of the interface and have a –90° phase shift. This ensures the write DQ data is center-aligned with respect to the DQS write strobe.

### Memory Clock (mem_clk)

The mem_clk clocks the following blocks:

- DDIO_OUT block that is part of the DQS output path and
- DDIO_OUT blocks that generate the CK and CK# signals.
- DDIO_OE block that is part of the DQS OE path
- DDIO_OE (extended_rtena) block that is part of the dynamic termination for the DQS pin
- DDIO_OE (extended_rtena) block that is part of the dynamic termination for the DQ pin

The mem_clk should be the full rate of the interface and typically have 0° phase shift.

### Address/Command Clock

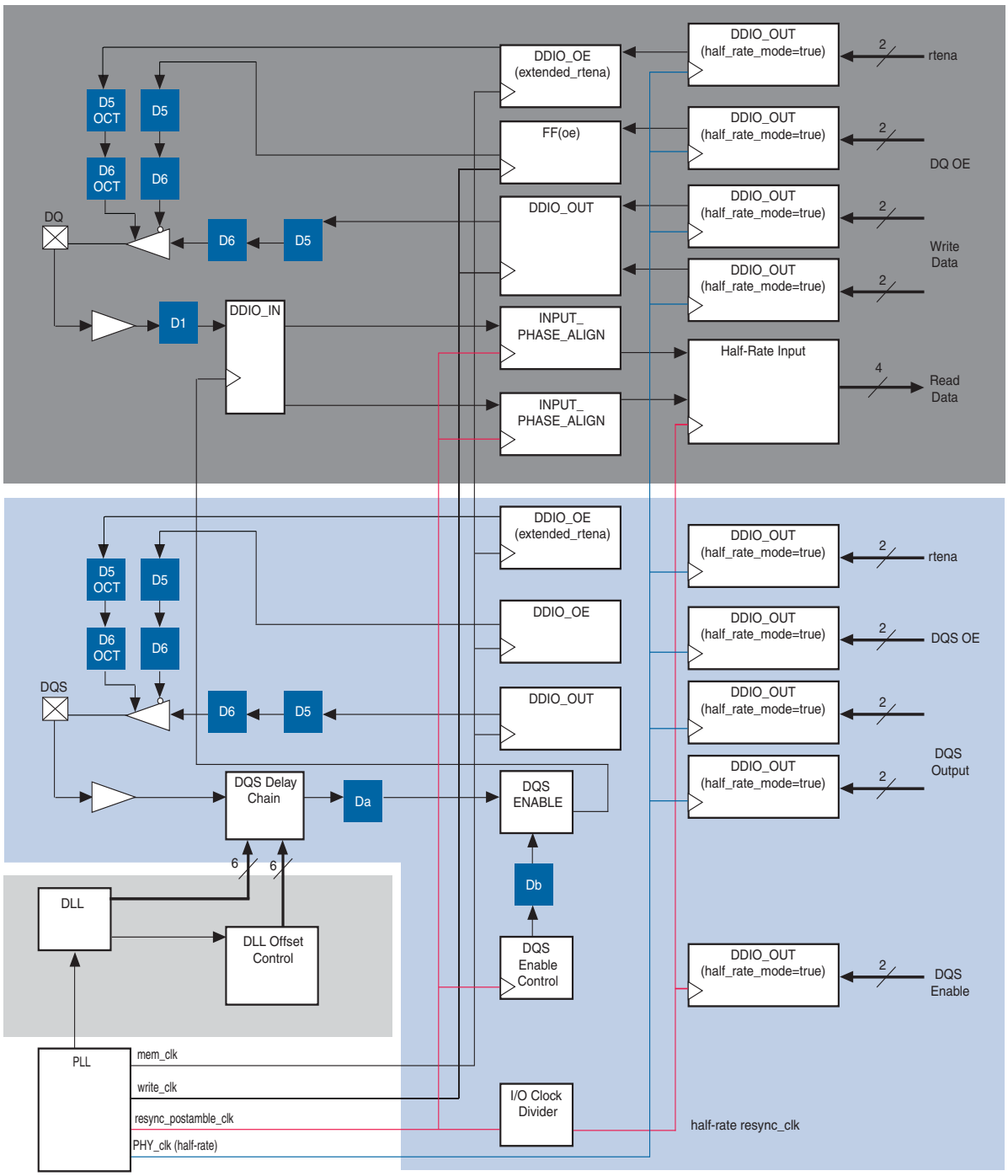
The address/command clock is used to clock the DDIO_OUT blocks that are used for the address and command pins. Although not shown in Figure 1–1 on page 1–2, it is discussed further in "Create Controller Logic for Address and Command Pins" on page 1–21.

The address/command clock should be the full rate of the interface and have a 180° phase shift. The 180° value is dependent on the address/command/control signals being perfectly matched to the clock signals in time delay on the PCB. Typically, this is not the case, but it is ideal.

## Clocking Scheme Requirement for Half-Rate Interface

Figure 1–2 shows a sample half-rate interface.

**Figure 1–2. Example Half-Rate Interface**




The PLL instance for a half-rate interface generates the following six clock outputs (refer to Figure 1–2):

- DLL input clock

- Resynchronize postamble clock (resync_postamble_clk)
- Write clock (write_clk)
- Memory clock (mem_clk)
- PHY Clock (PHY_clk)
- Address/command clock

The following sections describe the details of the six clock outputs for the half-rate interface.

### DLL Input Clock

The DLL clock is used to toggle the DLL block. The DLL clock should be equivalent to the full rate of the interface and have a 0° phase shift. You must connect the DLL block directly to a single dedicated PLL clock output.

### Resynchronize Postamble Clock (resync_postamble_clk)

The resync_postamble_clk clock is an optional clock. You can use this clock for three purposes:

- Clock the DQS enable control block. The DQS enable control block is used to disable the DQS input strobe after the strobe goes to Z (after a DDR read postamble). This is part of the DQS input path.
- Clock the INPUT_PHASE_ALIGN block. This block phase shifts the input signal and is primarily used to match the arrival delay of the DQS to the latest arrival delay of a DQS from the DDR or DDR2 DIMM. It is also used to resynchronize the data read from the DDIO_IN block.
- Feed the I/O clock divider block that is a part of the DQS input path. The I/O clock divider divides the clock by two (half-rate resync_clk) and uses it to clock the DDIO_OUT (half_rate_mode=TRUE) block that is part of the DQS input path. The DQS input path controls the DQS enable signal and also clocks the half-rate input block that is part of the DQ input path.

The resync_postamble_clk clock should be the full rate of the interface and have dynamic phase. You can set the dynamic phase capability for the PLL clock outputs with the ALTPLL megafunction. The correct resync_postamble_clk clock phase is crucial to correct data transfer. The ALTDQ_DQS megafunction cannot determine the phase that the resync_postamble_clk should have. Therefore, you must solve this with RTD analysis or creating a custom data training circuitry to write and read back a training pattern to and from the memory device and then dynamically adjust the resynchronization clock phase of the PLLs to find a good working phase.

### Write Clock (write_clk)

The write_clk clocks the DDIO_OUT block that is a part of the DQ output path. The DDIO_OUT block is used for writing data to the DQ pins and to clock the FF (oe) block that is a part of the DQ OE path. The FF (oe) block (refer to Figure 1–2) acts as the output enable for the DQ output path.

The write_clk should be the full rate of the interface and have about a –90° phase shift. This requirement ensures the write DQ data is center-aligned with respect to the DQS write strobe.

### Memory Clock (mem_clk)

The `mem_clk` clocks the following blocks:

- DDIO_OUT block that is part of the DQS output path

  The memory clock is also used to clock the DDIO_OUT blocks that generate the `CK` and `CK#` signals.
- DDIO_OE block that is part of the DQS OE path
- DDIO_OE (`extended_rtena`) block that is part of the dynamic termination for the DQS pin
- DDIO_OE (`extended_rtena`) block that is part of the dynamic termination for the DQ pin

The memory clock should be the full rate of the interface and have about a 0° phase shift.

### PHY Clock (PHY_clk)

The `PHY_clk` clocks the following blocks:

- Two DDIO_OUT (`half_rate_mode=TRUE`) blocks that are part of the DQS output path
- DDIO_OUT (`half_rate_mode=TRUE`) block that is part of the DQS OE path
- DDIO_OUT (`half_rate_mode=TRUE`) block that is part of the dynamic termination for the DQS pin
- Two DDIO_OUT (`half_rate_mode=TRUE`) blocks that are part of the DQ output path
- DDIO_OUT (`half_rate_mode=TRUE`) block that is part of the DQ OE path
- DDIO_OUT (`half_rate_mode=TRUE`) block that is part of the dynamic termination for the DQ pin

The `PHY_CLK` should be the half rate of the interface and have a 0° phase shift.

### Address and Command Clock

The address and command clock clocks the DDIO_OUT blocks that are used for the address and commend pins. Although not shown in Figure 1–2 on page 1–5, it is discussed further in "Create Controller Logic for Address and Command Pins" on page 1–21.

The address and command clock should be the full rate of the interface and have a phase shift between 180° and 359°. The value is dependent on the address/command/control signals being perfectly matched to the clock signals in time delay on the PCB. Typically, this is not the case, but it is ideal.

For more information about using PLLs or other blocks, refer to the *ALTPLL Megafunction User Guide.*

After you decide on your clocking scheme as discussed in this section, you can proceed to the next step.

# Instantiate ALTDLL Megafunctions

Using the MegaWizard Plus-In Manager, instatiate an ALTDLL megafunction that corresponds with the memory interface frequency. You must set the input frequency, delay chain length, and delay buffer mode for the DLL in the MegaWizard interface. Specify a DLL frequency mode that corresponds with the middle frequency range of the DLL frequency mode. This DLL frequency mode determines the delay buffer mode and delay chain length of interface. You can also specify a jitter reduction for the DLL offset control blocks.

Refer to the "DLL and DQS Logic Block Specifications" section of the the *Stratix III Device Datasheet* or the *Stratix IV Device Datasheet*.

For more information about the settings in this section, refer to the “MegaWizard Plug-In Manager Page Descriptions for the ALTDLL Megafunction” section of the *ALTDLL and ALTDQ_DQS Megafunctions User Guide*.

# Instantiate ALTDQ_DQS Megafunctions

After configuring the DLL settings, you must set the dedicated circuitry settings for external memory interfaces with the ALTDQ_DQS megafunction.

A typical custom external memory interface consists of the following:

- Zero or one DQS I/O pin (read or write strobe/clock) with an optional DQSn I/O pin (differential strobe/clock), or a CQ and CQn I/O pair (complementary clock)
- One or more DQ I/O pins (read or write data)
- One DM/D pin (output-only data mask, write data, or both)

The ALTDQ_DQS megafunction allows you to instantiate a group of DQ pins (ranging from ×4 to ×36) along with their corresponding DQS block, or DQSn I/O (optional) block, or both (optional). The ALTDQ_DQS megafunction enables you to generate the necessary DQ/DQS circuitry to use with external memory interfaces. You can configure the DQS and DQSn I/O as input-only, output-only, or bidirectional pins. All `bidir_dq` I/O pins are identically configured. Similarly, all `output_dq` (output-only DQ/DM I/O pins) and `input_dq` I/O pins (input-only DQ pins) are identically configured. Hence, if delay chains or half-data rate blocks are used in a path, the configuration applies to all paths of the same type. The paths can be all-input paths, all-output paths, or bidirectional paths in the ALTDQ_DQS variation. The ALTDQ_DQS megafunction generates only one DQS I/O pin per instantiation. If you need a 72-bit data interface composed of ×9 data groups, you must generate a ×9 data group with the megafunction and then instantiate it eight times.

The following steps describe the general flow when using the ALTDQ_DQS megafunction:

1. Configure the general settings for the ALTDQ_DQS instances, which includes the number of input, output, and bidirectional DQ, DQS delay chain stages, DQS input frequency, use half-rate components, and use dynamic OCT path.
2. Configure the DQS input path, which includes using input delay chain (D1), using DQS delay chain, enabling DQS busout delay chain, enabling DQS enable block, enabling DQS enable control block, and enabling DQS enable block delay chain.

3. If you enable the DQS delay chain block to true, the **DQS Delay Chain Settings** dialog box appears. You must configure the DQS delay chain settings.
4. If you enable the DQS enable control to true, the **DQS Enable Control Settings** dialog box appears. You must configure the DQS enable control settings.
5. Configure the DQS output/OE path, which includes enabling DQS output delay chain 1 (D5), enabling DQS output delay chain 2 (D6), configuring DQS output registers, enabling DQS OE delay chain 1 (D5), enabling DQS OE delay chain 2 (D6), and configuring the DQS OE registers.
6. Configure the DQ input path, which includes configuring the DQ input registers, selecting the clock source for the DQ input register, enabling the DQ input phase alignment block, use the DQ half-rate `dataoutbypass` port, and enabling the DQ input delay chain (D1).
7. If you enable the DQ input phase alignment block to true, the **DQ Input Phase Alignment Block Settings** dialog box appears. You must configure the DQ input phase alignment block settings.
8. Configure the DQ output/OE path, which includes enabling DQ output delay chain 1 (D5), enabling DQ output delay chain 2 (D6), configuring the DQ output registers, enabling DQ OE delay chain 1 (D5), enabling DQ OE delay chain 2 (D6), and configuring the DQ OE registers.
9. Configure the half-rate components, which includes configuring the IO clock divider block, source clock, creating the `io_clock_divider_masterin` input port for chaining multiple IO_CLOCK_DIVIDER blocks, enabling the `io_clock_divider_clkout` output port for clocking core registers, enabling the `io_clock_divider_slaveout` output port for chaining multiple IO_CLOCK_DIVIDER blocks, and inverting the phase through this block.
10. Configure the Dynamic OCT components, which includes enabling OCT delay chain 1(D5 OCT), enabling OCT delay chain 2(D6 OCT), and configuring OCT register mode.
11. Configure the DQS/DQSn IO,which includes configuring the DQS/DQSn pair as a differential pair or complementary pair.
12. Configure the reset ports for the ALTDQ_DQS instance. The configuration enables the asynchronous and synchronous reset ports to all the registers in the DQS and DQ datapath.

Table 1–1 shows two examples of the ALTDQ_DQS settings.

**Table 1–1. ALTDQ_DQS Interface Example Settings (Part 1 of 2)**

| Settings | Full-Rate Interface | Half-Rate Interface |
|---|---|---|
| Device | Stratix III, speed grade C3 | Stratix III, speed grade C3 |
| DQS/DQSn | 1 bidirectional differential DQS/DQS | 1 bidirectional differential DQS/DQS |
| DQ Pins | 8 bidirectional DQ pins | 8 bidirectional DQ pins |
| DM Pins | 1 output DM pin | 1 output DM pin |
| DQS Frequency | 155 MHz | 450 MHz |
| Using dynamic termination path | Yes | Yes |
| Using half-rate blocks | No | Yes |

**Table 1–1. ALTDQ_DQS Interface Example Settings (Part 2 of 2)**

| Settings | Full-Rate Interface | Half-Rate Interface |
|---|---|---|
| DQS delay chain phase shift | 90° | 90° |
| delay_chain_length for DLL | 12 | 8 |
| DLL has offset control blocks enabled | Yes | Yes |
| Data rate of interface | 310 Mbps | 900 Mbps |

## MegaWizard Plug-In Manager Options for ALTDQ_DQS Megafunction for Full-Rate and Half-Rate Settings

This section provides descriptions of the options available on the individual pages of the ALTDQ_DQS MegaWizard™ Plug-In Manager for both full-rate and half-rate interfaces, as shown in Figure 1–1 on page 1–2 and Figure 1–2 on page 1–5.

On page 3 of the ALTDQ_DQS MegaWizard Plug-In Manager, you can select options on the **Parameter Settings** page as shown in Table 1–2.

**Table 1–2. Parameter Settings (Part 1 of 2)** ***(Note 1)***

| Option *(2)* | Full-Rate Interface | Half-Rate Interface |
|---|---|---|
| Number of bidirectional DQ | **8**. There are 8 DQ pins for the interface. | **8**. There are 8 DQ pins for the interface. |
| Number of input DQ | **0**. This option is not used for the interface. | **0**. This option is not used for the interface. |
| Number of output DQ | **1**. This is used for the 1 DM output pin. | **1**. This is used for the 1 DM output pin. |
| Number of stages in dqs_delay_chain | **3**. Center the DQS strobe signal with the DQ data (DQS signal must be 90° phase shifted). This setting depends on the intended phase shifted (90°) and the DLL delay chain length (12). | **2**. Center the DQS strobe signal with the DQ data (DQS signal must be 90° phase shifted). This setting depends on the intended phase shifted (90°) and the DLL delay chain length (8). |
| DQS Input Frequency | Enter **155 MHz**. This value must match the input frequency for the DLL instance (via ALTDLL). | Enter **450 MHz**. This value must match the input frequency for the DLL instance (via ALTDLL). |
| Use half rate components | Turn off this option. This option is not applicable for the full-rate interface. | Turn on this option.<br>Turning on this option enables the following 11 half-rate components. |

**Table 1–2. Parameter Settings (Part 2 of 2) *(Note 1)***

| Option *(2)* | Full-Rate Interface | Half-Rate Interface |
|---|---|---|
| Use dynamic OCT path *(3)* | Turn on this option. Turning on this option enables the following two dynamic OCT components. | Turn on this option. Turning on this option enables the following four dynamic OCT components. |

**Notes to Table 1–2:**

(1) For more information about these options, refer to the "ALTDQ_DQS High-Level Configuration Settings" and "MegaWizard Plug-In Manager Page Descriptions for the ALTDQ_DQS Megafunction" sections in the *ALTDLL and ALTDQ_DQS Megafunctions User Guide*.

(2) You can use differential or complementary mode instead of single-ended mode for the DQS IO.

(3) With dynamic OCT, you can enable the parallel termination ($R_t$) during reads from the DDR and DDR2 external memory, and disable the $R_t$ during writes from the DDR and DDR2 external memory. This significantly reduces power consumption for external memory interfaces.

Table 1–3 shows the settings needed to achieve particular phase shifts for the DQS delay chain block mentioned in Table 1–2:

Setting the DQS delay chain phase is one of the most important parts of the ALTDQ_DQS megafunction. It determines the necessary phase shift in the DQS delay chain that clocks the DDIO_IN block in the DQ input path. This requires a phase shift of 90° to center align the DQS strobe with the DQ data. Consider the following scenarios:

- Scenario One

  If you are using a Stratix III device with C3 speed grade, refer to the *Stratix III Device Handbook* for the appropriate settings.

  Because the DQS input frequency for the first scenario is 155 MHz, the device speed grade is C3, the DLL delay chain length is 12, delay buffer mode is low, and the intended phase shift is 90°. For these parameters, set the **Stages of the DQS Delay Chain** option to 3.

- Scenario Two

  For a particular DLL delay chain length, you might not get your intended phase setting.

  For example, set the DLL delay chain length to 10 and the DQS input frequency to 190 MHz. Your intended phase setting is 90°, but the closest available is a 72° phase shift, which is DQS delay stages setting of 2. Instantiate the DLL offset control blocks (ALTDLL) to introduce the necessary offset to the DQS delay chain to achieve the intended 90° phase shift.

Page 4 of the ALTDQ_DQS MegaWizard Plug-In Manager is the **DQS IN Advanced Options** page. Table 1–3 shows how to configure the DQS input path of the ALTDQ_DQS instance.

**Table 1–3. DQS IN Advanced Options Page (Part 1 of 3)**

| Option *(1)* | Full-Rate Interface | Half-Rate Interface |
|---|---|---|
| Enable DQS Input Path *(2)* | Turn on this option. Turning on this option gives you the access to the five blocks in the DQS input path as shown in Figure 1–1 on page 1–2 | Turn on this option. Turning on this option gives you the access to the six blocks in the DQS input path as shown in Figure 1–2 on page 1–5. |
| Delay chain usage: (Enable dynamic delay chain) | Turn off this option. This option is not applicable for this interface. | Turn off this option. This option is not applicable for this interface. |

**Table 1–3. DQS IN Advanced Options Page (Part 2 of 3)**

| Option *(1)* | Full-Rate Interface | Half-Rate Interface |
|---|---|---|
| Delay chain usage: (Enable dqs_delay_chain) | Turn on this option. Turning on this option enables the DQS delay chain block. | Turn off this option. Turning off this option enables the DQS delay chain block. |
| Advanced Delay Chain Options (DQS Delay Chain Phase Setting Options) | Turn off the **Set dynamically using configuration registers** option. | Turn off the **Set dynamically using configuration registers** option. |
| Advanced Delay Chain Options (DQS Delay Chain Phase Setting Options) | Select the **DLL** option. This allows DLL to control the delay in the DQS delay chain block. | Select the **DLL** option. This allows DLL to control the delay in the DQS delay chain block. |
| Advanced Delay Chain Options (DQS Delay Buffer Mode) | Select the **Low** option. This must be set to low because this must match the delay buffer mode of the DLL which is used for full-rate interface. | Select the **High** option. This must be set to high because this must match the delay buffer mode of the DLL which is used for half rate interface. |
| Advanced Delay Chain Options (DQS Phase Shift) *(3)* | Enter **9,000** for 90°. The phase shift for the interface. | Enter **9,000** for 90°. The phase shift for the interface. |
| Advanced Delay Chain Options (Enable DQS offset control) | Turn on this option. This allows the offset settings (6-bit output) from the DLL offset control block to control the DQS delay chain block in. | Turn on this option. This allows the offset settings (6- bit output) from the DLL offset control block to control the DQS delay chain block. |
| Advanced Delay Chain Options: (Enable DQS delay chain latches) | Turn off this option. This option is not applicable for this interface. | Turn off this option. This option is not applicable for this interface. |
| Enable DQS busout delay chain *(2)* | Turn on this option. Turning on this option enables the Da block. | Turn on this option. Turning on this option enables the Da block**.** |
| Enable DQS enable block *(4)* | Turn on this option. This is used to enable or disable the DQS signal. | Turn on this option. This is used to enable or disable the DQS signal. |
| Enable DQS enable control block | Turn on this option. This is used to control the DQS enable block. | Turn on this option. This is to control the DQS enable block. |
| Advanced Enable Control Options (DQS Enable Control Phase Setting) | Select the **Set statically to '0'** option. The delay here is set to the static value of zero. | Select the **Set statically to '0'** option. The delay here is set to the static value of zero. |
| Advanced Enable Control Options (DQS Enable Control Invert Phase) | Select the **Never** option. | Select the **Never** option. |

**Table 1–3. DQS IN Advanced Options Page (Part 3 of 3)**

| Option *(1)* | Full-Rate Interface | Half-Rate Interface |
|---|---|---|
| Enable DQS enable block delay chain *(2)* | Turn on this option. | Turn on this option. |

**Notes to Table 1–3:**

(1) For more information about these options, refer to the "Configuring the DQS Input Path" and "MegaWizard Plug-In Manager Page Descriptions for the ALTDQ_DQS Megafunction" sections in the *ALTDLL and ALTDQ_DQS Megafunctions User Guide*.

(2) Da represents the DQSBUSOUT_DELAY_CHAIN block and Db represents the DQSENABLE_DELAY_CHAIN block.

(3) This setting is meant for timing analysis. This informs the TimeQuest timing analyzer to analyze the delay through the DQS delay chain block as the corresponding phase shift.

(4) The DQS enable represents the AND-gate control on the DQS input used to enable the DQS input strobe after the strobe goes to Z (after a DDR read postamble).

Page 5 of the ALTDQ_DQS MegaWizard Plug-In Manager is the **DQS OUT/OE Advanced Options** page. Table 1–4 shows how to configure the DQS OUTPUT and DQS OE path of the ALTDQ_DQS instance.

**Table 1–4. DQS OUT/OE Advanced Options Page**

| Option *(1)* | Full-Rate Interface | Half-Rate Interface |
|---|---|---|
| Enable DQS output path *(2)* | Turn on this option. Turning on this option gives you the access to the three blocks in the DQS output path. | Turn on this option. Turning on this option gives you the access to the three blocks in the DQS output path. |
| DQS Output Path Options (Enable DQS output delay chain1) *(2)* | Turn on this option. Turning on this option enables the D5 block. | Turn on this option. Turning on this option enables the D5 block. |
| DQS Output Path Options: (Enable DQS output delay chain2) *(2)* | Turn on this option. Turning on this option enables the D6 block. | Turn on this option. Turning on this option enables the D6 block. |
| DQS Output Path Options: (DQS output register mode) | Select the **DDIO** option. This sets the DDIO_OUT block that is part of the DQS output path in to a DDIO_OUT. | Select the **DDIO** option. This sets the DDIO_OUT block that is part of the DQS output path in to a DDIO_OUT. |
| DQS Output Enable Options (Enable DQS output enable) *(2)* | Turn on this option. Turning on this option gives you the access to the three blocks in the DQS OE path. | Turn on this option. Turning on this option gives you the access to the three blocks in the DQS OE path. |
| DQS Output Enable Options (Enable DQS output enable delay chain1) *(2)* | Turn on this option. Turning on this option enables the D5 block. This is part of the DQS OE path. | Turn on this option. Turning on this option enables the D5 block. This is part of the DQS OE Path. |
| DQS Output Enable Options (Enable DQS output enable delay chain2) *(2)* | Turn on this option. Turning on this option enables the D6 block. This is part of the DQS OE path. | Turn on this option. Turning on this option enables the D6 block. This is part of the DQS OE path. |
| DQS Output Enable Options (DQS output enable register mode) | Select the **DDIO** option. This sets the DDIO_OUT block this is part of the DQS OE path to a DDIO_OUT. | Select the **DDIO** option. This sets the DDIO_OUT block that is part of the DQS OE path to a DDIO_OUT. |

**Notes to Table 1–4:**

(1) For more information about these options, refer to the "Configuration the DQS Output Path, Configuration the DQS OE Path" and "MegaWizard Plug-In Manager Page Description for the ALTDQ_DQS Megafunction" sections in the ALTDLL and ALTDQ_DQS Megafunction User Guide.

(2) D5 block is dynamic output delay chain1 and D6 block is dynamic output delay chain 2. These delay chains are configured dynamically during FPGA run-time to deskew the DQS pins, providing improved timing margins. For more information about delay chains, refer to the "Delay Chains For External Memory Interfaces" section in the ALTDLL and ALTDQ_DQS Megafunctions User Guide.

Page 6 of the ALTDQ_DQS MegaWizard Plug-In Manager is the **DQ IN Advanced Options** page. You can select options based on Table 1–5. This page configures the DQ input path of the ALTDQ_DQS instance.

**Table 1–5. DQ IN Advanced Options Page (Part 1 of 2)**

| Option *(1)* | Full-Rate Interface | Half-Rate Interface |
|---|---|---|
| Enable DQS Input Path *(2)*, *(4)*, *(5)* | Turn on this option. Turning on this option gives you the access to the four blocks in the DQ input path. | Turn on this option. Turning on this option gives you the access to the five blocks in the DQ input path . |
| DQ Input Register Options (DQ input register mode) | Select the **DDIO** option. This sets the DDIO_IN block that is part of the DQ input path. | Select the **DDIO** option. This sets DDIO_IN block that is part of the DQ input path. |
| DQ Input Register Options (DQ input register clock source) *(3)* | Select the **'dqs_bus_out' port** option and turn off the **'Connect DDIO clkn to DQS_BUS from complementary DQSn'** option. This is used for the DDIO_IN block. | Select the **'dqs_bus_out' port** option and turn off the **'Connect DDIO clkn to DQS_BUS from complementary DQSn'**option. This is used for the DDIO_IN. |
| DQ Input Register Options Use DQ input phase alignment *(4)* | Turn on this option. Turning on this option enables two INPUT_PHASE_ALIGN blocks that are part of the DQ input path. This is part of the DQ input path. | Turn on this option. Turning on this option enables two INPUT_PHASE_ALIGN blocks that are part of the DQ input path. This is part of the DQ Input Path. |
| Advanced DQ IPA options (DQ Input Phase Alignment Phase Setting) | Select the **Set statically to '0'** option. The delay is set to the static value of zero. This is used for the two INPUT_PHASE_ALIGN blocks that are part of the DQ input path. | Select the **Set statically to '0'** option. The delay is set to the static value of zero. This is meant for the two INPUT_PHASE_ALIGN blocks that are part of the DQ input path. |
| Advanced DQ IPA options (Add DQ Input Phase Alignment Input Cycle Delay) | Select the **Never** option. This option is only used for the two INPUT_PHASE_ALIGN blocks that are part of the DQ input path. | Select the **Never** option. This option is only used for the two INPUT_PHASE_ALIGN blocks that are part of the DQ input path. |
| Advanced DQ IPA options (Invert DQ Input Phase Alignment Phase) | Select the **Never** option. This option is only used for the two INPUT_PHASE_ALIGN blocks that are part of the DQ input path. | Select the **Never** option. This option is only used for the two INPUT_PHASE_ALIGN blocks that are part of the DQ input path. |
| Advanced DQ IPA options (Register DQ input phase alignment bypass output) | Turn off this option. This option is only used for two INPUT_PHASE_ALIGN blocks that are part of the DQ input path. | Turn off this option. This option is only used for two INPUT_PHASE_ALIGN blocks that are part of the DQ input path. |
| Advanced DQ IPA options (Register DQ input phase alignment add phase transfer) | Turn off this option. This option is only used for the two INPUT_PHASE_ALIGN blocks that are part of the DQ input path. | Turn off this option. This option is only used for the two INPUT_PHASE_ALIGN blocks that are part of the DQ input path. |
| Use DQ half rate 'dataoutbypass' port *(5)* | Turn off this option. This option is only used for the half-rate input block that is part of the DQ input path. | Turn off this option. This option is only used for the half-rate input block that is part of the DQ input path. |

**Table 1–5. DQ IN Advanced Options Page (Part 2 of 2)**

| Option *(1)* | Full-Rate Interface | Half-Rate Interface |
|---|---|---|
| Use DQ input delay chain *(2)* | Turn on this option. Turning on this option enables the D1 block. This is part of the DQ input path. | Turn on this option. Turning on this option enables the D1 block. This is part of the DQ input path. |

**Notes to Table 1–5:**

(1) For more information about these options, refer to the "Configuring the DQ Input Path" and "MegaWizard Plug-In Manager Page Descriptions for the ALTDQ_DQS Megafunction" sections in the *ALTDLL and ALTDQ_DQ Megafunctions User Guide*.

(2) D1 block is dynamic input delay chain 1.This delay chain can be dynamically configured during FPGA run-time to deskew the DQ pins, hence providing improved timing margins. For more information about delay chains, refer to "Delay Chains For External Memory Interfaces" section in the *ALTDLL and ALTDQ_DQS Megafunctions User Guide*.

(3) You have a choice whether to clock the DQ input registers from the DQS input path (via `dqs_bus_out` port) or from the FPGA core (via `dq_input_reg_clk`).

(4) The INPUT_PHASE_ALIGN blocks represent the circuitry required to phase shift the input signal. This is primarily used to match the arrival delay of the DQS to the latest arrival delay of a DQS from the DIMM. The phase settings for the INPUT_PHASE_ALIGN blocks must match the I/O clock divider block.

(5) The half-rate input block represents the circuitry required to transfer the input signal to a half-rate clock. One of the outputs can also be switched to a direct input from the input buffer. This block is use only in half-rate interfaces.

Page 7 of the ALTDQ_DQS MegaWizard Plug-In Manager is the **DQ OUT/OE Advanced Options** page. You can select options based on Table 1–6. This configures the DQS OUTPUT and DQS OE path of the ALTDQ_DQS instance.

**Table 1–6. DQ OUT/OE Advanced Option Page**

| Option *(1)* | Full-Rate Interface | Half-Rate Interface |
|---|---|---|
| DQ Output Path Options (Enable DQ output delay chain1) *(2)* | Turn on this option. Turning on this option enables the D5 block. This is part of the DQ output path. | Turn on this option. Turning on this option enables the D5 block. This is part of the DQ output path. |
| DQ Output Path Options (Enable DQ output delay chain2) *(2)* | Turn on this option. Turning on this option enables the D6 block. This is part of the DQ output path. | Turn on this option. Turning on this option enables the D6 block . This is part of the DQ output path. |
| DQ Output Path Options (DQS output register mode). | Select the **DDIO** option. This sets the DDIO_OUT block that is part of the DQ output path. | Select the **DDIO** option. This sets DDIO_OUT block that is part of the DQ output path. |
| DQ Output Enable Options (Enable DQ output enable) | Turn on this option. Turning on this option gives you the access to the three blocks in the DQ OE path. | Turn on this option. Turning on this option gives you the access to the three blocks in the DQ OE path. |
| DQ Output Enable Options (Enable DQS output enable delay chain1) *(2)* | Turn on this option. Turning on this option enables D5 block. This is part of the DQ OE path. | Turn on this option. Turning on this option enables D5 block. This is part of the DQ OE path. |
| DQ Output Enable Options (Enable DQ output enable delay chain2) *(2)* | Turn on this option. Turning on this option enables the D6 block in. This is part of the DQ OE path. | Turn on this option. Turning on this option enables the D6 block. This is part of the DQ OE path. |
| DQ Output Enable Options (DQ output enable register mode). | Select the **FF** option. This sets the FF (oe) block that is part of the DQ OE path. | Select the **FF** option. This sets FF (oe) block that is part of the DQ OE path. |

**Notes to Table 1–6:**

(1) For more information about these options, refer to the "Configuring the DQS Output Path", "Configuring the DQ OE Path", and "MegaWizard Plug-In Manager Page Descriptions for the ALTDQ_DQS Megafunction" sections in the *ALTDLL and ALTDQ_DQS Megafunctions User Guide*.

(2) D5 block is a dynamic output delay chain 1 and D6 block is dynamic output delay chain 2. These delay chains can be dynamically configured during FPGA run-time to deskew the DQS pins, hence providing improved timing margins. For more information about delay chains, refer to the "Delay Chains For External Memory Interfaces" section in the *ALTDLL and ALTDQ_DQS Megafunctions User Guide*.

Page 8 of the ALTDQ_DQS MegaWizard Plug-In Manager is the **Half-Rate Advanced Options** page. You can select options based on Table 1–7. This configures the DQ input path of the ALTDQ_DQS instance.

**Table 1–7. Half-Rate Advanced Options Page**

| Option *(1)* | Full-Rate Interface | Half-Rate Interface |
|---|---|---|
| IO Clock Divider Source *(2)*, *(3)* | This option is not applicable for full-rate interface. | Select the **'dqs_bus_out' port** option. This uses the clock source from the DQS input path, divides it by half, and then clocks the half-rate blocks, which are:<br>■ DDIO_OUT (`half_rate_mode=TRUE`) block that is part of the DQS input path, which controls the DQS enable signal<br>■ Half-rate input block that is part of the DQ input path. This is used for the I/O clock divider block. |
| Create 'io_clock_divider_masterin' input port *(2)*, *(4)* | No. Not applicable for full-rate interface. | Turn off this option. For this half-rate interface, there is only one ALTDQ_DQS instance. Therefore, I/O clock divider blocks are automatically chained in the instance depending on the amount of DQ pins. This is used for the I/O clock divider block . |
| Create 'io_clock_divider_clkout' output port *(2)*, *(5)* | | Turn on this option. This output should clock the core registers that are getting the half-rate DQ data from the half-rate registers. |
| Create 'io_clock_divider_slaveout' output port *(2)* | | Turn off this option. For this half-rate interface, there is only one ALTDQ_DQS instance. Therefore, I/O clock divider blocks are automatically chained in the instance depending on the amount of DQ pins. This is used for the I/O clock divider block. |
| IO Clock Divider Invert Phase *(2)* | | Select the **Never** option. This is used for the IO_Clock_Divider block. |

**Notes to Table 1–7:**

(1) For more information about these options, refer to the "Configuring the DQS Input Path", "I/O Clock Divider Primitive", and "MegaWizard Plug-In Manager Page Descriptions for the ALTDQ_DQS Megafunction" sections in the ALTDLL and ALTDQ_DQS Megafunctions User Guide.

(2) The I/O_Clock_Divider block represents a div-by-2 clock divider for transferring data to the core at one half the speed of the I/O input and output clock. Each divider can feed up to six pins (a x4 DQS group) in the device. To feed wider DQS groups, multiple clock dividers must be chained together by feeding the `slaveout` output of one divider to the `masterin` input of the neighboring pins' divider. The phase settings for the INPUT_PHASE_ALIGN blocks must match the I/O clock divider block.

(3) This is used to clock the IO_CLOCK_DIVIDER sub-block, which is used during ahalf-rate operation.

(4) If enabled, then the `masterin` input is used to synchronize this divider with another IO_CLOCK_DIVIDER sub-block. If disabled, then this divider operates independently (this mode is meant for the master divider for a group of dividers).

(5) Enables the clock output signal divided by 2. It can be connected to the clock input of a half-rate input block or it can feed the FPGA core.

Page 9 of the ALTDQ_DQS MegaWizard Plug-In Manager is the **OCT Advanced Options** page. You can select options based on Table 1–8. This configures the OCT registers and delay chain settings of the ALTDQ_DQS instance.

**Table 1–8. OCT Advanced Options Page**

| Option *(1)* | Full-Rate Interface | Half-Rate Interface |
|---|---|---|
| Dynamic OCT Options (Enable OCT delay chain1) *(2)* | Turn on this option. Turning on this option enables two D5 OCT blocks. This is part of the DQ and DQS dynamic termination path. | Turn on this option. Turning on this option enables two D5 OCT blocks. This is part of the DQ OE Path. This is part of the DQ and DQS dynamic termination path. |
| Dynamic OCT Options (Enable OCT delay chain2) *(2)* | Turn on this option. Turning on this option enables two D6 OCT blocks. This is part of the DQ and DQS dynamic termination path. | Turn on this option. Turning on this option enables two D6 OCT blocks. This is part of the DQ OE Path. This is part of the DQ and DQS dynamic termination path. |
| DQ Output Path Options (OCT register mode) | Select the **DDIO** option. This sets two blocks:<br>■ DDIO_OE (`extended_rtena`) block that is part of the dynamic termination for the DQS pin<br>■ DDIO_OE (`extended_rtena`) block that is part of the dynamic termination for the DQ pin. | Select the **DDIO** option. This sets four blocks:<br>■ DDIO_OE (`extended_rtena`) block that is part of the dynamic termination for the DQS pin<br>■ DDIO_OE (`extended_rtena`) block that is part of the dynamic termination for the DQ pin<br>■ DDIO_OUT (`half_rate_mode=TRUE`) block that is part of the dynamic termination for the DQS pin<br>■ DDIO_OUT (`half_rate_mode=TRUE`) block that is part of the dynamic termination for the DQ pin. |

**Notes to Table 1–8:**

(1) For more information about these options, refer to the "Configuring the DQ/DQS OCT Path" and "MegaWizard Plug-In Manager Page Descriptions for the ALTDQ_DQS Megafunction" sections in the *ALTDLL and ALTDQ_DQS Megafunctions User Guide*.

(2) The D5 OCT block is dynamic output delay chain 1 and the D6 OCT block is dynamic output delay chain 2. These delay chains are used together with the D5 block and D6 block. With Dynamic OCT, you can enable $R_t$ during reads from the DDR and DDR2 external memory and disable $R_t$ during writes from the DDR and DDR2 external memory. This has a benefit of significantly reducing power consumption for external memory interfaces. Hence the necessary usage of the D5 OCT block and D6 OCT block to synchronize during reads and writes from both DQ and DQS pins, and improve overall timing margins. For more information about delay chains, refer to the "Delay Chains For External Memory Interfaces" section in the *ALTDLL and ALTDQ_DQS Megafunctions User Guide*.

Page 10 of the ALTDQ_DQS MegaWizard Plug-In Manager is the **DQS/DQSn IO Advanced Options** page. You can select options based on Table 1–9.

**Table 1–9. DQS/DQSn IO Advanced Options Page**

| Options *(1)* | Full-Rate Interface | Half-Rate Interface |
|---|---|---|
| Use DQSn I/O | Turn on this option. | Turn on this option. |
| DQS and DQSn IO Configuration mode *(2)* | Select **Differential Pair**. This is required for this particular scenario. | Select **Differential Pair**. This is required for this particular scenario |

**Notes to Table 1–9:**

(1) For more information about these options, refer to the "Configuring the DQSn I/O Pin Path" and "MegaWizard Plug-In Manager Page Descriptions for the ALTDQ_DQS Megafunction" sections in the *ALTDLL and ALTDQ_DQS Megafunctions User Guide*.

(2) For the "Differential pair" configuration, the DQSn I/O is configured in a differential pair along with the DQS IO. This means that the OE and OCT paths are configured for the DQSn I/O, whereas the input and output paths are shared with the DQS IO.

Page 11 of the ALTDQ_DQS MegaWizard Plug-In Manager is the **Reset Port Advanced Options** page. You can select options based on Table 1–10.

**Table 1–10. Reset Ports Advanced Options Page**

| Option | Full-Rate Interface | Half-Rate Interface |
|---|---|---|
| Create 'dqs_areset' input port | Turn on this option. Turning on this option enables asynchronous reset port for all registers in the DQS datapath. | Turn on this option. Turning on this option enables asynchronous reset port for all registers in the DQS datapath. |
| Create 'dqs_sreset' input port | Turn on this option. Turning on this option enables synchronous reset port for all registers in the DQS datapath. | Turn on this option. Turning on this option enables synchronous reset port for all registers in the DQS datapath. |
| Create 'input_dq_areset' input port | Turn off this option. This option is not applicable in this scenario because the DQ datapath is bidirectional. | Turn off this option. This option is not applicable in this scenario because the DQ datapath is bidirectional. |
| Create 'input_dq_sreset' input port | Turn off this option. This option is not applicable in this scenario because the DQ datapath is bidirectional. | Turn off this option. This option is not applicable in this scenario because the DQ datapath is bidirectional. |
| Create 'output_dq_areset' input port | Turn on this option. Turning on this option enables asynchronous reset port for all registers in the DQ datapath. This is meant for the DM pin. | Turn on this option. Turning on this option enables asynchronous reset port for all registers in the DQ datapath. This is meant for the DM pin. |
| Create 'output_dq_sreset' input port | Turn on this option. Turning on this option enables synchronous reset port for all registers in the DQ datapath. This is meant for the DM pin. | Turn on this option. Turning on this option enables synchronous reset port for all registers in the DQ datapath. This is meant for the DM pin. |
| Create 'bidir_dq_areset' input port | Turn on this option. Turning on this option enables asynchronous reset port for all registers in the DQ datapath. | Turn on this option. Turning on this option enables asynchronous reset port for all registers in the DQ datapath. |
| Create 'bidir_dq_sreset' input port | Turn on this option. Turning on this option enables synchronous reset port for all registers in the DQ datapath. | Turn on this option. Turning on this option enables synchronous reset port for all registers in the DQ datapath. |

This completes the settings for both full-rate and half-rate scenarios for the ALTDQ_DQS instance.

# Instaniate the ALTOCT Megafunction

Use of the OCT scheme in Stratix III and Stratix IV devices eliminates the need for external series or parallel termination resistors and simplifies the design of a PCB. Stratix III and Stratix IV devices support calibrated on-chip series, parallel, and dynamic termination in all I/O banks for single-ended I/O standards. OCT calibration allows you to establish an optimal termination value that compensates for impedance change due to temperature and voltage fluctuation. You can calibrate Stratix III and Stratix IV devices with user-controlled signals during device operation, or by default during device configuration. For calibrated termination, use the ALTOCT megafunction to utilize the dedicated calibration blocks for termination available in the FPGA together with the dedicated OCT in the I/O buffers with the ALTIOBUF megafunction.

For custom external memory interfaces, there are significant advantages to using both dynamic OCT and calibrated termination:

- OCT calibration allows you to establish an optimal termination value that compensates for impedance change due to temperature and voltage fluctuation.
- Simplifies the design of a PCB.
- Significant power consumption reduction for external memory interfaces. With dynamic OCT you can enable $R_t$ during reads from the DDR and DDR2 external memory and disable $R_t$ during writes from the DDR and DDR2 external memory and when the interface is idle.

You can use the ALTDLL and ALTDQ_DQS custom external memory interfaces to achieve these advantages.

For more information about using the dynamic calibration blocks for termination, refer to *Dynamic Calibrated On-Chip Termination (ALTOCT) Megafunction User Guide*.

For more information about using the dynamic OCT, refer to *I/O Buffer (ALTIOBUF) Megafunction User Guide*.

For more information about implementing calibrated dynamic OCT, refer to *AN 465: Implementing OCT Calibration in Stratix III Devices*.

## Create Controller Logic for the Read and Write Datapaths

Consider the following details when designing the controller for these signals:

- General (applies to both full-rate and half-rate interfaces):
  - The DQ signals are edge-aligned with the DQS signal during a read from the memory and are center-aligned with the DQS signal during a write to the memory.
  - The memory controller shifts the DQ signals by –90° during a write operation to center-align the DQ and DQS signals. Center-align the PLL to the DQS signal with respect to the DQ signals during writes.
  - The memory controller delays the DQS signal during a read, to ensure that the DQ and DQS signals are center-align at the capture register. Use dedicated DQS phase-shift circuitry (ALTDLL and ALTDQ_DQS) to shift the incoming DQS signal during reads.
  - The DQS signal is generated on the positive edge of the system clock to meet the $t_{DQSS}$ requirement. The setup ($t_{DS}$) and hold times ($t_{DH}$) of the memory device for the write DQ and DM pins are relative to the edges of DQS write signals and not the CK or CK# clock.
  - DQ and DM signals use a clock shifted –90° from the system clock to ensure that the DQS edges are centered on the DQ or DM signals when they arrive at the DDR2 SDRAM.
  - DDR2 SDRAM uses the DM pins during a write operation. Driving the DM pins low shows that the write is valid. The memory masks the DQ signals if the DM pins are driven high. The timing requirements of the DM signal at the DDR2 SDRAM input are identical to those for DQ data. The Stratix III DDR registers, clocked by the –90° shifted clock, create the DM signals. While you can use any of the I/O pins in the same bank as the associated DQS and DQ pins to generate the DM signal, Altera recommends using the spare DQ pin in the same DQS group as the respective data to minimize skew.
  - Some DDR2 SDRAM DIMMs support error correction coding (ECC) to detect and automatically correct errors in data transmission. The 72-bit DDR2 SDRAM modules contain eight ECC pins in addition to 64-data pins. Connect the eight DDR2 SDRAM device ECC pins to a single DQS or DQ group.
  - The DQS, DQ, and DM board trace lengths must be tightly matched (20 ps).
  - The DQS is bidirectional. The DQSn pins in DDR2 SDRAM devices are optional but recommended for DDR2 SDRAM designs operating at more than 333 MHz.
  - The DQ pins are also bidirectional. Regardless of interface width, DDR SDRAM always operates in ×8 mode DQS groups. DDR2 SDRAM interfaces can operate in either ×4 or ×8 mode DQS groups, which is dependant on your chosen memory device or DIMM, and is also not related to the actual interface width. The ×4 and ×8 configurations use one pair of bidirectional data strobe signals, DQS and DQSn, to capture input data.
  - Two pairs of data strobes, UDQS and UDQS# (upper byte) and LDQS and LDQS# (lower byte), are required by the ×16 configuration devices. A group of DQ pins must remain associated with its respective DQS and DQSn pins.

- Full-rate interface (refer to Figure 1–1 on page 1–2):
  - DQS, DQSn, DM, and DQ signals from FPGA core must be connected to the ALTDQ_DQS instance. The ALTDQ_DQS instance interfaces to the external memory (DDR and DDR2) for these pins.
  - Read data from the DQ input path should go through a dual-clock FIFO to transfer the data from the INPUT_PHASE_ALIGN blocks (`resync_postamble_clk` clock domain) to the FPGA core clock domain.
  - Controller has to ensure that the write data from core and DQS output path strobe are aligned to ensure that when writing data to external memory, the DQS strobe is center-aligned with the output DQ data to the external memory.
- Half-rate interface (refer to Figure 1–2 on page 1–5):
  - DQS, DQSn, DM, and DQ signals from FPGA core must be connected to the ALTDQ_DQS instance. The ALTDQ_DQS instance interfaces to the external memory (DDR and DDR2) for these pins.
  - Read data from the DQ input path should go through a dual-clock FIFO to transfer the data from the half-rate input block (half-rate `resync_clk` clock domain) to the FPGA core clock domain.
  - Controller has to make sure the write data from core and DQS output path strobe must be aligned to ensure that when writing data to external memory, the DQS strobe is center-aligned with the output DQ data to the external memory.

For more information about the timing requirements for the DQS and DQSn strobe signal, DM signal, and DQ data during reading and writing operations, refer to the respective external memory device datasheet.

## Create Controller Logic for Address and Command Pins

Because this is a custom external memory interface for DDR and DDR2 interface, you must design your own custom controller to control the address and command pins from the FPGA core.

Consider the following details when designing the controller for the `CK` and `CK#` pins:

- DDR2 SDRAM components use `CK` and `CK#` signals to clock the address and command signals into the memory. Furthermore, the memory uses these clock signals to generate the DQS signal during a read through the DLL inside the memory. The DDR2 SDRAM data sheet specifies the following timings:
  - $t_{DQSCK}$ is the skew between the `CK` or `CK#` signals and the DDR3 SDRAM-generated DQS signal
  - $t_{DSH}$ is the DQS falling edge from `CK` rising edge hold time
  - $t_{DSS}$ is the DQS falling edge from `CK` rising edge setup time
  - $t_{DQSS}$ is the positive DQS latching edge to `CK` rising edge
  - $t_{DQSCK}$ is the DQS output access time from `CK`

- The DDR2 SDRAM has a write requirement ($t_{DQSS}$) that states the positive edge of the DQS signal on writes must be ± 25% (± 90°) of the positive edge of the DDR2 SDRAM clock input. Therefore, the memory controller should generate the CK and CK# signals using the DDR registers in the IOE to match with the DQS signal and reduce any variations across process, voltage, and temperature. The positive edge of the DDR2 SDRAM clock, CK, is aligned with the DQS write to satisfy $t_{DQSS}$.

Consider the following details when designing the controller for the A (address), BA (bank address), CKE, CS#, RAS#, CAS#, WE#, and ODT pins:

1. Address and command signals in DDR2 SDRAM components are clocked into the memory device with the CK or CK# signal. These pins operate at a single data rate (SDR) using only one clock edge. The number of address pins depends on the DDR2 SDRAM device capacity. The address pins are multiplexed, so two clock cycles are required to send the row, column, and bank address. The CS, RAS, CAS, WE, CKE, and ODT pins are DDR2 SDRAM command and control pins. The DDR2 SDRAM address and command inputs do not have a symmetrical setup and hold time requirement with respect to the DDR2 SDRAM clocks, CK, and CK#.
2. In Stratix III and Stratix IV devices, the address and command clock should be one of the PLL dedicated clock outputs whose phase can be adjusted to meet the setup and hold requirements of the memory clock. The address and command clock is also typically half rate, although a full rate implementation can also be created. The command and address pins use the DDIO output circuitry to launch commands from either the rising or falling edges of the clock.
3. The chip selects (CS_N), clock enable (CKE), and ODT pins are only enabled for one memory clock cycle and can be launched from either the rising or falling edge of the address and command clock signal. The address and other command pins are enabled for two memory clock cycles and can also be launched from either the rising or falling edge of the address and command clock signal.

For more information about the timing requirements for the address and command pins during reading and writing operations, refer to the respective external memory device datasheet.

## Create Controller Logic for the OCT Calibration Block

If calibrated series, parallel, or dynamic termination is used for the I/O in your design, your design requires a calibration block. This block requires a pair of RUP and RDN pins located in a bank that shares the same $V_{CCIO}$ voltage as your memory interface. This calibration block is not required to be in the same bank or side of the device as the IOEs it is serving. To use these capabilities in the FPGA, you must use the ALTOCT megafunction.

Consider the following details when designing the memory controller to be used with the ALTOCT megafunction:

- The RUP and RDN input ports of the ALTOCT instance must be connected to the respective RUP and RDN pins in the FPGA. Note that the RUP and RDN pins are dual-purpose pins and there are multiple pairs of RUP and RDN pins for each bank.

- The memory controller controls the `calibration_request`, `s2pload`, and `calibration_wait` input ports of this ALTOCT instance, to ensure that the termination is calibrated based on the PVT (process, voltage and temperature) variation requirements of the custom external memory interface.
- The `calibration_busy` and `calibration_shift_busy` output ports of this ALTOCT instance must be used by the memory controller to determine when the calibration process of the termination is complete. This is important, because then only normal operations like reading to and writing from and to external memory can be performed. During termination calibration, the pins interfacing to the external memory must not be used for any operations.
- The `seriesterminationcontrol` and `parallelterminationcontrol` output ports of this ALTOCT instance must be connected to the `ser_term_ctrl` and `par_term_ctrl` input ports of the ALTIOBUF instance, which interfaces with the external memory pins. This is to transfer the calibrated termination settings to the I/O buffers.

For more information about using these ports in the ALTOCT megafunction, refer to *Dynamic Calibrated On-Chip Termination (ALTOCT) Megafunction User Guide*.

For more information about implementing calibrated dynamic OCT in your designs, refer to *AN 465: Implementing OCT Calibration in Stratix III Devices*.

## Instantiate ALTIOBUF Megafunctions

All of the interface pins must be connected to I/O buffers via the ALTIOBUF megafunction. You must use this megafunction to enable differential capabilities for the I/O buffer which is connected to a differential DQS pin or to enable dynamic OCT capabilities for the respective interface pins.

Consider the following details when designing the memory controller to be used with the ALTIOBUF megafunction:

1. The `ser_term_ctrl` and `par_term_ctrl` input ports of this ALTIOBUF instance must be connected to the respective `seriesterminationcontrol` and `parallelterminationcontrol` output ports of the ALTOCT instance, which interfaces with the external memory pins. This is to transfer the calibrated termination settings to the I/O buffers.
2. The `dyn_term_ctrl` input ports of this ALTIOBUF instance must be connected. This port enables $R_t$ during reads from the DDR and DDR2 external memory, and disables $R_t$ during writes from the DDR and DDR2 external memory. This port must be connected to the output of the dynamic OCT DQ/DQS path in the ALTDQ_DQS instance. The input of this path in the ALTDQ_DQS instance should be connected to the memory controller, where it can determine when a read or write operation is done.

For more information about using these ports, refer to *I/O Buffer Megafunction (ALTIOBUF) User Guide*.

For more information about implementing calibrated dynamic OCT, refer to *AN 465: Implementing OCT Calibration in Stratix III Devices*.

# Connect all Instances Used in Custom External Memory Interface

When all instances of ALTDLL, ALTPLL, ALTDQ_DQS, ALTIOBUF, ALTOCT, and the custom memory controller are completed, you must connect them in the Quartus II software.

# Add Constraints

The next step in the design flow is to add the timing, location, and physical constraints related to the external memory interface. These constraints include:

- Timing constraints
- Pin locations, DQ group assignments and I/O standards
- Pin loading, termination, and drive strength assignments

Unlike the ALTMEMPHY solution that automatically generates TCL scripts for timing constraints, I/O standard settings, and DQ group assignments, the ALTDLL and ALTDQ_DQS solution requires you to manually create these constraints via the Assignment Editor in the Quartus II software, and then source these assignments to a TCL script for reusability.

## Timing Constraints

The ALTDLL and ALTDQ_DQS external memory solution only supports timing analysis using the TimeQuest timing analyzer with Synopsys Design Constraints (**.sdc**) assignments. These constraints are derived from the DDR2 and DDR SDRAM data sheet and tolerances from the board layout. The ALTDLL and ALTDQ_DQS external memory solution uses TimeQuest timing constraints and the timing driven fitter to achieve timing closure.

Consider the following details when performing timing analysis external on the memory interface:

- Read timing margins for DLL-based implementation
- Write data timing analysis
- Round-trip delay calculation
- DQS postamble timing

Because external memory interfaces are essentially source-synchronous interfaces, these are timing margins that you must verify:

- Address and command setup, and hold margin
- Half-rate address and command setup, and hold margin
- FPGA Core setup and hold margin
- FPGA Core reset and removal setup and hold margin
- Write setup and hold margin
- Read capture setup and hold margin

These are explained in detail in *AN 433: Constraining and Analyzing Source-Synchronous Interfaces*.

For more information about creating timing constraints in SDC format for the TimeQuest timing analyzer, refer to the *TimeQuest Timing Analyzer* chapter in volume 3 of the *Quartus II Handbook*.

### Pin Locations, DQ Group Assignments, and I/O Standard

Pin locations assignments and DQ group assignments depend on your external memory interface pins. This is specific to FPGA device used. These assignments must be done in the Assignment Editor.

For the proper assignment values for pin location assignments and DQ group assignments, refer to the *External Memory Interfaces in Stratix III Devices* chapter in volume 1 of the *Stratix III Device Handbook* and the *External Memory Interfaces in Stratix IV Devices* chapter in volume 1 of the *Stratix IV Device Handbook*.

I/O standard assignments depend on your external memory that interfaced to the FPGA. This is specific to the external memory used. For example, DDR interfaces use SSTL I/O Standards. These assignments must be done in the Assignment Editor.

For the proper assignment value for I/O standard assignments, refer to the *Stratix III Device I/O Features* chapter in volume 1 of the *Stratix III Device Handbook* and the *I/O Features in Stratix IV Devices* chapter in volume 1 of the *Stratix IV Device Handbook*.

### Pin Loading, Termination, and Drive Strength Assignments

These assignments depend on your external memory that interfaced to the FPGA. You must execute these assignments through the Assignment Editor.

For the proper assignment values, refer to the *Board Design Guidelines* section in Volume 2 of the *Eternal Memory Interface Handbook*.

## Plan Resources

This section describes planning resources.

Table 1–11 shows the pin placements that Altera recommends.

**Table 1–11. Stratix III DDR and DDR2 SDRAM Pin Placement Recommendations**

| Signal | Pin or FPGA | Pin on Memory Devices |
|---|---|---|
| Data (`mem_dq`) | DQ | DQ |
| Data mask (`mem_dm`) | DQ *(1)* | DM |
| Data strobe (`mem_dqs`) | DQS or DQS | DQS or DQS# |
| Memory clock (`mem_clk`) | DQ, or DQ, or DQSn | CK or CK# |
| Address | Any user I/O *(2)* | A or BA |
| Command | Any user I/O *(2)* | CS#, RAS#, CAS#, WE#, CKE, or ODT |

**Notes to Table 1–11:**

(1) The DM pins must be in the DQ group.

(2) Ensure that address and command pins are placed on the same side of the device as the memory clock pins. If OCT is used, ensure that the RUP and RDN pins are assigned correctly.

The ALTDLL and ALTDQ_DQS external memory solution do not generate pin assignments for non-memory signals such as clock sources or pin location assignments for your design. Launch the Pin Planner to make these assignments to your design.

## Advanced IO Timing

As part of I/O planning, especially with high-speed designs, you must take board-level signal integrity and timing into account. When adding an FPGA device with high-speed interfaces to a board design, the quality of the signal at the far end of the board route, and the propagation delay in getting there, are vital for proper system operation.

## Perform RTL or Functional Simulation

After instantiating the SDRAM high-performance controller, it generates a design example and driver for testing the memory interface.

When using the ALTDLL and ALTDQ_DQS external memory solution, unlike the ALTMEMPHY solution, it does not use the SDRAM high-performance controller to generate an example design and driver for testing the memory interface. If you use this solution, you must create your own driver circuitry to test the custom-made controller logic with the datapath created via the ALTDLL, ALTDQ_DQS, ALTOCT, ALTPLL, and ALTIOBUF megafunctions. This implies that you have to manually create your own testbench to verify the custom external memory interface.

To perform functional simulation with the ALTDLL and ALTDQ_DQS external memory solution, you must connect the simulation model of the driver, the simulation model of your design (which includes customized controller logic and ALTDLL and ALTDQ_DQS datapath), and the simulation model of their external memory in their functional simulation environment.

## Compile Design and Verify Timing

After constraining your design, compile your design in the Quartus II software. Because the ALTDLL and ALTDQ_DQS external memory solution does not automatically generate timing reports to verify whether timing has been met, you must use SDC commands to manually compile the design.

After compiling your design in the Quartus II software, run the verifying timing script to produce the timing report for different paths, such as write data, read data, address and command, and core (entire interface) timing paths in your design.

The custom verifying timing script should report about margins on the following paths:

- Address and command setup and hold margin
- Half-rate address and command setup and hold margin
- Core setup and hold margin
- Core reset and removal setup and hold margin
- Write setup and hold margin

- Read capture setup and hold margin

For more information about timing analysis and reporting using the ALTDLL and ALTDQ_DQS external memory solution, refer to *AN 438: Constraining and Analyzing Timing for External Memory Interfaces*.

# Adjust Constraints

The timing report of your designs show the worst case setup and hold margin for the different paths in your design. If the setup and hold margin are unbalanced, achieve a balanced setup and hold margin by adjusting the phase setting of the clocks that clock these paths.

For example, for the address and command margin, the address and command outputs are clocked by an address and command clock that can be different with respect to the system clock, which is 0º. The system clock times the clock outputs going to the memory. If the report timing script indicates that using the default phase setting for the address and command clock results in more hold time than setup time, adjust the address and command clock to be less negative than the default phase setting with respect to the system clock to ensure that there is less hold margin. Similarly, adjust the address and command clock to be more negative than the default phase setting with respect to the system clock if there is more setup margin.

# 2. Implementing a Custom DDR2 SDRAM Interface

This chapter describes how to implement an 8-bit wide, 150-MHz, 300-Mbps DDR2 SDRAM full-rate interface using the ALTDLL and ALTDQ_DQS megafunctions.

This design example also provides some recommended settings, such as termination scheme and drive strength settings, to simplify your design. Although the design example is specifically for the DDR2 SDRAM interface, the design flow for a DDR SDRAM interface is the same.

At the end of this chapter, you create an example design similar to the one in www.altera.com/later/later, that targets the Stratix III FPGA Development Kit, which includes a component module (MT47H32M8BP-3:B). You can adapt this design to target any other board.

## Software Requirements

This chapter assumes that you have experience with the Quartus II software. In addition, ensure that you have the Quartus II software version 9.0 or later installed.

## Create a Quartus II Project and Specify a Target Device

Use the New Project wizard (**File > New Project Wizard**) to create a project in the Quartus II software that targets the EP3SL150F1152-C2ES device. This example design targets the EP3SL150F1152-C2 device targeting an 8-bit wide DDR2 SDRAM interface at 150 MHz. The design uses an 8-bit wide 256-MB Micron MT47H32M8BP-3:B 333-MHz DDR2 SDRAM component.

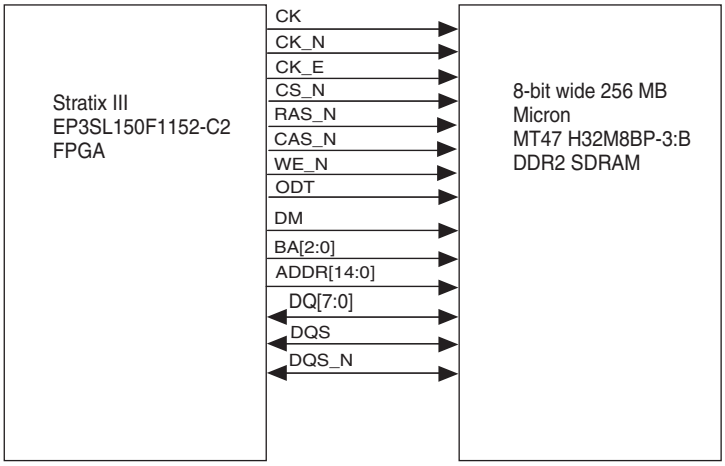
For detailed step-by-step instructions on how to create a Quartus II project, refer to the Tutorial in the Quartus II software. On the Help menu in the Quartus II window, click **Tutorial**.

## Instantiate the PHY and Controller

This section describes the steps required to instantiate a custom PHY, implemented using the ALTDLL and ATLTDQ_DQS megafunctions, as well as a custom designed controller. This design example is in Verilog HDL and consists of one top-level module or instance that connects 19 sub-modules or instances for a complete customized memory controller for DDR2. All these modules or instances are defined in the following sections.

Figure 2–1 shows the example interface between the Stratix III EP3SL150F1152-C2 device and an 8-bit wide 256-MB Micron MT47H32M8BP-3:B 333-MHz DDR2 SDRAM component.

**Figure 2–1. Interface Between a Stratix III Device and a DDR2 SDRAM Component**




The CK, CK_N, CK_E, CS_N, RAS_N, CAS_N, WE_N, ODT, DM, BA[2:0], and ADDR[14:0] signals are output-only signals. DQ[7:0], DQS and DQS_N are bidirectional signals.

## Instantiate ALTPLL Megafunctions

This design example uses the ALTPLL megafunction to implement two PLLs that clock the entire full-rate interface. Use the MegaWizard Plug-In Manager (Tools menu) to instantiate two variations of the ALTPLL megafunction with the following parameters:

- pll_1:
  - in_clock = 50 MHz
  - mode = no compensation
  - c0 = 150 MHz, 0° phase shift. Used to clock the dll_1 instance. This PLL is dedicated to clock only the DLL
- pll_2:
  - in_clock = 50 MHz
  - mode = normal
- c0 = 150 MHz, 0° phase shift. This is the mem_clk. It is used to clock the registers in the altdq_dqs_1 instance (the DQS output registers, the DQS OE registers, the DQS dynamic OCT registers, and DQ dynamic OCT registers), in the mem_clock_generate instance (the registers for generating CK and CK_N signals), the control_init_ddr instance (state machine for initialization of the DDR2), the control_write_ddr instance (state machine for writing data to the DDR2), the control_read_ddr instance (state machine for reading data from the DDR2), and

the control_driver_ddr instance (state machine for driving the control_init_ddr instance, control_write_ddr instance, and control_read_ddr instance). It is also used to clock the registers in the addr_cmd_generate and addr_cmd_generate_oe instances which are used to generate the ADDR[14:0], BA[2:0], CK_E, CS_N, RAS_N, CAS_N, WE_N, and ODT signals.

- c1 = 150 MHz, –90° phase shift. This is the write_clk. It is used to clock the registers in the altdq_dqs_1 instance (the DQ output registers and the DQ OE registers).

## Instantiate the ALTDLL Megafunction

After deciding on the clocking scheme, you use the ALTDLL megafunction to instantiate a DLL of the correct operational frequency (DQS signal frequency) of the custom external memory. This frequency setting depends on the bandwidth you want.

Use the MegaWizard Plug-In Manager to instantiate a variations of the ALTPLL megafunction with the following parameters:

- dll_1:
  - turn on jitter reduction = No
  - delay chain length = 12
  - delay buffer mode = low
  - DQS input frequency = 150 MHz
  - Instantiate DLL offset control A = No
  - Instantiate DLL offset control B = No
  - create ‘dll_aload’ port = No
  - create ‘dll_dqsupdate’ port = Yes

## Instantiate ALTDQ_DQS Megafunction

Next, you initialize the settings for the dedicated circuitry for external memory interfaces using the ALTDQ_DQS megafunction.

This custom DDR2 external memory interface consists of the following:

- One pair of differential DQS/DQSn I/O pin (read or write strobe/clock)
- Eight DQ I/O pins (read or write data)
- One DM pin (output-only data mask)

The following section describes how to instantiate a variation of the ALTDQ_DQS megafunction with the correct parameters.

### Specify altdq_dqs_1 Parameters

This section provides descriptions of the options that must be set for the altdq_dqs_1 instance on the individual pages of the ALTDQ_DQS MegaWizard Plug-In Manager for full-rate mode.

Page 3 of the ALTDQ_DQS MegaWizard Plug-In Manager is the **Parameter Settings** page. To configure the general settings for the ALTDQ_DQS instance, specify the options shown in Table 2–1.

**Table 2–1. Parameter Settings**

| Option | Full-Rate Interface |
|---|---|
| Number of bidirectional DQ | **8**. There are 8 bidirectional DQ pins |
| Number of input DQ | **0**. Not used. |
| Number of output DQ | **1**. There is 1 output only DM pin. |
| Number of stages in dqs_delay_chain | **3**. This enables the DQS strobe signal to be centered with the DQ data (DQS signal must be +90° phase shifted) |
| DQS input frequency | Enter **150 MHz** |
| Use half-rate components | Turn off this option. |
| Use dynamic OCT path | Turn off this option. |

Page 4 of the ALTDQ_DQS MegaWizard Plug-In Manager is the **DQS IN Advanced Options** page. To configure the DQS input path of the ALTDQ_DQS instance, set the options shown in Table 2–2.

**Table 2–2. DQS IN Advanced Options Page**

| Option | Full-Rate Interface |
|---|---|
| Enable DQS Input Path | Turn on this option. |
| Delay chain usage: | Select the **Enable dqs_delay_chain** option. |
| Advanced Delay Chain Options (DQS Delay Chain Phase Setting Options) | Turn off the **Select dynamically using configuration registers** option. |
| Advanced Delay Chain Options (DQS Delay Chain Phase Setting Options) | Select the **DLL** option. |
| Advanced Delay Chain Options (DQS Delay Buffer Mode) | Select the **Low** option. |
| Advanced Delay Chain Options (DQS Phase Shift) | Enter **9,000** |
| Advanced Delay Chain Options (Enable DQS offset control) | Turn off this option. |
| Advanced Delay Chain Options Enable DQS delay chain latches | Turn on this option. |
| Enable DQS busout delay chain | Turn on this option. |
| Enable DQS enable block | Turn on this option. |
| Enable DQS enable control block | Turn on this option. |
| Advanced Enable Control Options (DQS Enable Control Phase Setting) | Select the **Set statically to '0'** option. |
| Advanced enable control options: (DQS Enable Control Invert Phase) | Select the **Never** option. |
| Enable DQS enable block delay chain | Turn on this option. |

Page 5 of the ALTDQ_DQS MegaWizard Plug-In Manager is the **DQS OUT/OE Advanced Options** page. To configure the DQS OUTPUT and DQS OE path of the ALTDQ_DQS instance, select the options shown in Table 2–3.

**Table 2–3. DQS OUT/OE Advanced Options Page**

| Option | Full-Rate Interface |
|---|---|
| Enable DQS output path | Turn on this option. |
| DQS Output Path Options (Enable DQS output delay chain1) | Turn on this option. |
| DQS Output Path Options (Enable DQS output delay chain2) | Turn on this option. |
| DQS Output Path Options (DQS output register mode) | Select the **DDIO** option. |
| DQS Output Enable Options (Enable DQS output enable) | Turn on this option. |
| DQS Output Enable Options (Enable DQS output enable delay chain1) | Turn on this option. |
| DQS Output Enable Options (Enable DQS output enable delay chain2) | Turn on this option. |
| DQS Output Enable Options (DQS output enable register mode) | Select the **FF** option. |

Page 6 of the ALTDQ_DQS MegaWizard Plug-In Manager is the **DQ IN Advanced Options** page. To configure the DQ input path of the ALTDQ_DQS instance, select the options shown in Table 2–4.

**Table 2–4. DQ IN Advanced Options Page**

| Option | Full-Rate interface |
|---|---|
| DQ Input Register Options (DQ input register mode) | Select DDIO option. |
| DQ Input Register Options (DQ input register clock source) | Select the **'dqs_bus_out' port** option and disable the **'Connect DDIO clkn to DQS_BUS from complementary DQSn'** option. |
| DQ Input Register Options (Use DQ input phase alignment) | Turn on this option. |
| Advanced DQ IPA options:<br>(DQ Input Phase Alignment Phase Setting) | Select the **Set statically to '0'** option. |
| Advanced DQ IPA options:<br>(Add DQ Input Phase Alignment Input Cycle Delay) | Select the **Never** option. |
| Advanced DQ IPA options:<br>(Invert DQ Input Phase Alignment Phase) | Select the **Never** option. |
| Advanced DQ IPA options:<br>(Register DQ input phase alignment bypass output) | Turn on this option. |
| Advanced DQ IPA options:<br>(Register DQ input phase alignment add phase transfer) | Turn off this option. |
| DQ Input Register Options (Use DQ half rate 'dataoutbypass' port) | Turn off this option. |
| Use DQ input delay chain | Turn on this option. |

Page 7 of the ALTDQ_DQS MegaWizard Plug-In Manager is the **DQ OUT/OE Advanced Options** page. To configure the DQ OUTPUT and DQ OE path of the ALTDQ_DQS instance, select the options shown in Table 2–5.

**Table 2–5. DQ OUT/OE Advanced Options Page**

| Option | Full-Rate Interface |
|---|---|
| DQ Output Path Options (Enable DQ output delay chain1) | Turn on this option. |
| DQ Output Path Options (Enable DQ output delay chain2) | Turn on this option. |
| DQ Output Path Options (DQ output register mode) | Select the **DDIO** option. |
| DQ Output Enable Options (Enable DQ output enable) | Turn on this option. |
| DQ Output Enable Options (Enable DQ output enable delay chain1) | Turn on this option. |
| DQ Output Enable Options (Enable DQ output enable delay chain2) | Turn on this option. |
| DQ Output Enable Options (DQ output enable register mode) | Select the **FF** option. |

Page 8 of the ALTDQ_DQS MegaWizard Plug-In Manager is the **Half-rate Advanced Options** page. To configure the half-rate settings of the ALTDQ_DQS instance, select the options shown in Table 2–6.

**Table 2–6. Half-Rate Advanced Options Page**

| Option | Full-Rate Interface |
|---|---|
| IO Clock Divider Source | Turn off this option. This option is not applicable for full-rate interface. |
| Create 'io_clock_divider_masterin' input port | Turn off this option. This option is not applicable for full-rate interface. |
| Create 'io_clock_divider_clkout' output port | Turn off this option. This option is not applicable for full-rate interface. |
| Create 'io_clock_divider_slaveout' output port | Turn off this option. This option is not applicable for full-rate interface. |
| IO Clock Divider Invert Phase | Select the **Never** option. Not applicable for full-rate interface |

Page 9 of the ALTDQ_DQS MegaWizard Plug-In Manager is the **OCT Path Advanced Options** page. To configure the OCT registers and delay chain settings of the ALTDQ_DQS instance, select the options shown in Table 2–7.

**Table 2–7. OCT Path Advanced Options Page**

| Option | Full-Rate Interface |
|---|---|
| Dynamic OCT Options (Enable OCT delay chain 1) | Turn off this option. This option is not applicable for full-rate interface. |
| Dynamic OCT Options (Enable OCT delay chain 2) | Turn off this option. This option is not applicable for full-rate interface. |
| DQ Output Path Options (OCT register mode) | Turn off this option. This option is not applicable for full-rate interface. |

Page 10 of the ALTDQ_DQS MegaWizard Plug-In Manager is the **DQS/DQSn IO Advanced Options** page. Select options shown in Table 2–8.

**Table 2–8. DQS/DQSn IO Advanced Options Page**

| Option | Full-Rate Interface |
|---|---|
| Use DQSn I/O | Turn on this option. |
| DQS and DQSn IO Configuration mode | Select the **Differential Pair** option. This is required for this particular scenario. |

Page 11 of the ALTDQ_DQS MegaWizard Plug-In Manager is the **Reset Ports Advanced Options** page. Select options shown in Table 2–9.

**Table 2–9. Reset Ports Advanced Options Page**

| Option | Full-Rate Interface |
|---|---|
| Create 'dqs_areset' input port | Turn on this option. |
| Create 'dqs_sreset' input port | Turn off this option. |
| Create 'input_dq_areset' input port | Turn off this option. |
| Create 'input_dq_sreset' input port | Turn off this option. |
| Create 'output_dq_areset' input port | Turn on this option. |
| Create 'output_dq_sreset' input port | Turn off this option. |
| Create 'bidir_dq_areset' input port | Turn on this option. |
| Create 'bidir_dq_sreset' input port | Turn off this option. |

This completes the parameter settings for the `altdq_dqs_1` instance for this design example.

## Design Customized Memory Controller Datapath Logic

Because this is a custom external memory interface for DDR2 interface, you must design the necessary datapath to control the `CK`, `CK_N`, `ADDR[14:0]`, `BA[2:0]`, `CK_E`, `CS_N`, `RAS_N`, `CAS_N`, `WE_N`, and `ODT` signals. Altera provides a design example that you can use to create your own logic.

In the Altera-provided design example, the datapath of the `CK` and `CK_N` signals are controlled by the `mem_clock_generate` instance. This instance consists of two DDIO_OUT blocks. For the `CK` signal, the inputs of the DDIO_OUT block are each tied to $V_{CC}$ and GND. For the `CK_N` signal, the inputs of the DDIO_OUT block are each tied to GND and $V_{CC}$ to reflect the inverse of the `CK` signal.

The `addr_cmd_generate` instance controls the datapath of the `ADDR[14:0]`, `BA[2:0]`, `CK_E`, `CS_N`, `RAS_N`, `CAS_N`, `WE_N`, and `ODT` signals. This `addr_cmd_generate` instance consists of 24 DDIO_OUT blocks to individually represent the 24 signals. For these signals, there are 24 DFF blocks to control their respective OE (output enable) signals. The OE signals are controlled by the `addr_cmd_generate_oe` instance.The inputs of these 48 blocks are fed accordingly with data, depending on the three state machines that act as the control path of the customized memory controller. The three state machines are as follows:

- `control_init_ddr` instance

- control_write_ddr instance
- control_driver_ddr instance

## Multiplxer Instances

There are two multiplexer instances: cmd_addr_mux_1 and dqs_dqsn_dq_dm_mux_1.

### cmd_addr_mux_1

The cmd_addr_mux_1 instance is a 144-bit to 72-bit multiplexer with a 1-bit select signal. The cmd_addr_mux_1 instance multiplexes the CK_E, CK_N, RAS_N, CAS_N, WE_N, BA_OE, BA_DATA, ADDR_OE, ADDR_DATA and ODT signals from the following two control path state machines of the customized memory controller:

- control_init_ddr instance
- control_write_ddr instance

The output of the multiplexer is sent to the data path of the command and address instances, addr_cmd_generate and addr_cmd_generate_oe. For this multiplexer, the control_driver_ddr instance controls the 1-bit select.

### dqs_dqsn_dq_dmr_mux_1

The dqs_dqsn_dq_dmr_mux_1 instance is a 66-bit to 33-bit multiplexer with a 1-bit select signal. The dqs_dqsn_dq_dmr_mux_1 instance multiplexes the dm_oe, dm_data, dq_oe, dq_data, dqs_oe, dqs_data, dqsn_oe, and dqsn_data signals from the following two control path state machines of the customized memory controller:

- control_init_ddr instance
- control_write_ddr instance

The output of the multiplexer is sent to the data path of the DQS, DQSN, DQ, and DM (ALTDQ_DQS instance), which is the altdq_dqs_1 instance. The 1-bit select for this multiplexer is controlled by the control_driver_ddr instance.

This completes the steps for the customized memory controller datapath logic to generate the CK,CK_N, ADDR[14:0], BA[2:0], CK_E, CS_N, RAS_N, CAS_N, WE_N, and ODT signals.

## Design Customized Memory Controller Control Path Logic

Because this is a custom external memory interface for the DDR and DDR2 interface, you must design control path logic to control the DQS, DQS_N, DQ, DM, A (address), BA (bank address), CK, CK#, CKE, CS#, RAS#, CAS#, WE#, and ODT signals from the FPGA core.

The design example contains three state machines to control these signals to enable proper operation of the external memory interface. These state machines are as follows:

- control_init_ddr instance

  The control_init_ddr instance initializes the DDR2 component for the proper interface operation following the timing requirements as specified in the specific Micron DDR2 datasheet. Because the ALTMEMPHY megafunction does not support a burst length of 8, the customized memory controller in this design example initialized the DDR2 for this mode of operation.

- control_write_ddr instance

  The control_write_ddr instance writes a set of 8 data to the memory array in the DDR2 component following the timing requirements as specified in the specific Micron DDR2 datasheet.

- control_driver_ddr instance

  The control_driver_ddr instance coordinates the two state machines (control_init_ddr and control_write_ddr instances) following the timing requirements as specified in the specific Micron DDR2 datasheet to enable proper operation of this design example. This includes sequentially enabling the following instances:

  - The control_init_ddr instance state machine to initialize the DDR2 component
  - The control_write_ddr instance state machine to write the data to the memory array in the DDR2 component

  The state machine also controls the select signals of the two multiplexers (cmd_addr_mux_1 and dqs_dqsn_dq_dm_mux_1) depending on which of the two control path state machine is currently active.

## Instantiate ALTIOBUF Megafunctions for Pins

For an external memory interface, the DQ, DQS, DQSn, DM, ADDR[14:0], BA[2:0], CK, CK_N, CK_E, CS_N, RAS_N, CAS_N, WE_N, and ODT pins must be connected to I/O buffers via the ALTIOBUF megafunction. There are 14 interface signals for this design example that need I/O buffers to be interfaced with the FPGA pins. These I/O buffers are contained in the dqs_io_buffer, ck_io_buffer, dq_io_buffer, dm_io_buffer, addr_io_buffer, ba_io_buffer, and cmd_io_buffer instances. There are 35 I/O buffers used in this instance.

Table 2–10 shows the requirements that must be set for these signals in the I/O buffer instances.

**Table 2–10. ddr2_io_buffer Instance Signal Requirements (Part 1 of 2)**

| Signal | Instance | Requirement |
|---|---|---|
| DQS and DQS_N | dqs_io_buffer | 1-bit bidirectional I/O buffer with differential capabilities enabled |
| CK and CK_N | ck_io_buffer instance | 2-bit output I/O buffer with differential capabilities enabled |
| DQ[7:0] | dq_io_buffer | 8-bit bidirectional I/O bufferI/O buffer |
| DM | dm_io_buffer | 1-bit output I/O buffer |
| ADDR[14:0] | addr_io_buffer | 15-bit output I/O buffer |
| BA[2:0] | ba_io_buffer | 3-bit output I/O buffer |

**Table 2–10. ddr2_io_buffer Instance Signal Requirements (Part 2 of 2)**

| Signal | Instance | Requirement |
|---|---|---|
| CK_E | cmd_io_buffer | 1-bit output I/O buffer |
| CS_N | cmd_io_buffer | 1-bit output I/O buffer |
| RAS_N | cmd_io_buffer | 1-bit output I/O buffer |
| CAS_N | cmd_io_buffer | 1-bit output I/O buffer |
| WE_N | cmd_io_buffer | 1-bit output I/O buffer |
| ODT | cmd_io_buffer | 1-bit output I/O buffer |

### Connect All the Instances That are Used in the Custom External Memory Interface

This connection is highlighted with the top-level instance of this design example that top_custom_ddr2_controller instances. It connects all instances discussed in the previous section.

## Add Constraints to Design Example

After instantiating the necessary instances to create a customized DDR2 memory controller from the "Instantiate PHY (via ALTDLL and ALTDQ_DQS) and (Custom-Designed) Controller in a Quartus II Project" stage, you must generate the constraints files for the design example. Apply these constraints to the design before compilation.

☞ You must manually specify constraints because this design example does not use the ALTMEMPHY megafunction or the DDR2 SDRAM high-performance controller.

### Add Timing Constraints

When you instantiate the customized DDR2 memory controller design, it does not automatically generate a timing constraint file (SDC file). You must manually create your own SDC file to constrain the timing on this design. This design example comes with a timing constraints file, **top_custom_ddr2_controller_phy_ddr_timing.sdc**.

The timing constraint file constrains the clocks on the customized DDR2 memory controller design.

To add timing constraints, perform the following steps:

1. On the Assignments menu, click **Settings**.
2. In the **Category** list, expand **Timing Analysis Settings** and select **TimeQuest Timing Analyzer**.
3. Select the **top_custom_ddr2_controller_phy_ddr_timing.sdc** file and click **Add**.
4. Click **OK**.

### Set Top-Level Entity

Before compiling the design, set the top-level entity of the project. The design example top-level file is **top_custom_ddr2_controller.v**, which connects 13 other sub-modules or instances for a complete customized memory controller for DDR2.

To set the top-level file, perform the following steps:

1. Open the top-level entity file, **top_custom_ddr2_controller.v**.
2. On the Project menu, click **Set as Top-Level Entity**.

## Set Optimization Technique

To ensure the remaining unconstrained paths are routed with the highest speed and efficiency, set the optimization technique to **Speed**. To set the optimization technique, perform the following steps:

1. On the Assignments menu, click **Settings**.
2. Select **Analysis & Synthesis Settings**.
3. Under **Optimization Technique,** select **Speed**.
4. Click **OK**.

## Set Fitter Effort

To set the Fitter effort to Standard Fit, perform the following steps:

1. On the Assignments menu, click **Settings**.
2. Expand **Fitter Settings**.
3. Turn on **Optimize Hold Timing** and select **All Paths**.
4. Turn on **Optimize Fast Corner Timing**.
5. Under **Fitter Effort,** select **Standard Fit**.
6. Click **OK**.

## Enter Pin Location Assignments

To enter the pin location assignments, perform the following steps:

1. On the Processing menu, point to **Start**, and click **Start Analysis and Synthesis**.
2. Assign all your pins, so the Quartus II software fits your design correctly and gives correct timing analysis. To assign pin locations for the Stratix III development kit, run the **top_custom_ddr2_controller_PinLocations.tcl** file, which is provided with the design example or manually assign pin locations with the Pin Planner using the Stratix III FPGA Development Kit at the Altera website.

If you are at the design exploration phase of your design cycle and do not have any PCB defined pin locations, you should still manually define an initial set of pin constraints, which can become more specific during your development process.

To manually assign pin locations, perform the following steps:

1. Open Pin Planner. On the Assignments menu, click **Pin Planner**.

2. Assign DQ and DQS pins.
   a. To select the device DQS pin groups that the design uses, assign each `DQS` pin in your design to the required DQS pin in the Pin Planner. The Quartus II Fitter then automatically places the respective DQ signals onto suitable DQ pins within each group. To see the DQS groups in Pin Planner, right click, select **Show DQ/DQS Pins**, and click **In ×8/×9 Mode**. Pin Planner shows each DQS group in a different color and with a different legend: S = DQS pin, Sbar = DQSn pin, and Q = DQ pin.

   
   Most DDR2 SDRAM devices operate in ×8/×9 mode. However, some DDR2 SDRAM devices operate in ×4 mode. Refer to your specific memory device datasheet.

   b. Select the DQ mode to match the DQ group width (number of DQ pins/number of DQS pins) of your memory device. DQ mode is not related to the memory interface width.

   
   The DQ group order and DQ pin order in each group is not important. However, you must place DQ pins in the same group as their respective strobe pin.

3. Place DM pins in their respective DQ group.
4. Place address and control command pins on any spare I/O pins ideally in the same bank or side of the device as the `CK` and `CK_N` pins.
5. Ensure you place `CK` and `CK_N` pins on differential I/O pairs for the CK/CK# pin pair. To identify differential I/O pairs, right-click in Pin Planner and select **Show Differential Pin Pair Connections**. Pin pairs show a red line between each pin pair.

   
   You must place `CK` and `CK_N` on a `DIFFIO_RX` pin pair, if your design uses differential DQS signaling.

6. Place the `clock_source` pin on a dedicated PLL clock input pin with a direct connection to this design example's PLL and DLL pair—usually on the same side of the device as your memory interface. This recommendation reduces PLL jitter, saves a global clock resource, and eases timing and fitter effort.
7. Place the `global_reset` pin (like any high fan-out signal) on a dedicated clock pin.


For more information about how to use the Quartus II Pin Planner, refer to the *I/O Management* chapter in volume 2 of the *Quartus II Handbook*.

## Board Trace Delay Models

For accurate I/O timing analysis, you must specify the board trace and loading information. This information should be derived and refined during your PCB development process of pre-layout (line) simulation and finally post-layout (board) simulation.

# Perform RTL or Functional Simulation (Optional)

This section describes RTL and functional simulation. To set up simulation option, perform the following steps:

1. Unzip the **top_custom_ddr2_controller_msim.zip** file to any directory on your PC. Obtain and copy the vendors memory Verilog HDL simulation model to the directory where the previous zip file was uncompressed. This can be retrieved from the Micron website. Get the **ddr2.v**, **ddr2_mcp.v**, and **ddr2_parameters.vh** memory model files from the Micron website and save them in the directory of this design example.
2. Open the memory model file (**ddr2.v**) in a text editor and add the following `define` statements to the top of the file:

   ```
   'define sg3
   'define x8
   'define MAX_MEM
   ```

   The three `define` statements prepare the DDR2 SDRAM interface model.

   The first statement specifies the memory device speed grade as –3.

   The second statement specifies the memory device width per DQS.

   The third statement says to allocate memory for every address supported by the DDR2 model.
3. Open the testbench (**tb_top_custom_ddr2_controller.v**) from the directory in a text editor, instantiate the downloaded memory model, and connect its signals to the rest of the design.
4. Start the ModelSim® software.
   a. On the File menu, click **Change Directory**.
   b. Select the folder in which you unzipped the files.
   c. On the Tools menu, click **Execute Macro**.
   d. Select the **top_custom_ddr2_controller_msim.do** file and click **Open**. This is a script file for the ModelSim-Altera software to automate the necessary settings for the simulation.
   e. Verify the results shown in the Wave window.

# Compile Design and Verify Timing for Design Example

To compile the design, on the Processing menu, click **Start Compilation**.

After successfully compiling the design, run the TimeQuest timing analyzer to verify the timing based on the SDC file.For more information about the TimeQuest Timing Analyzer window, refer to the *Quartus II TimeQuest Timing Analyzer* chapter in volume 3 of the *Quartus II Handbook*.

For more information about timing analysis, refer to the *Timing Analysis* section of the *External Memory Interfaces Handbook*.

# Adjust Constraints for this Design Example

If the timing margin report shows negative hold time on the address and command datapath, adjusting the clock that is regulating the address and command output registers can improve the hold margin on the address and command datapath.

# Verifying Design on a Board

You have the option to verify the customized DDR2 memory controller design example. You can implement the necessary debug logic to observe the read and write activity of the external memory interface during FPGA run-time.

For more information about using the SignalTap II logic analyzer, refer to *Design Debugging Using the SignalTap II Embedded Logic Analyzer* chapter in volume 3 of the *Quartus II Handbook*

This chapter provides additional information about the document and Altera.

## Document Revision History

The following table shows the revision history for this document.

| Date | Version | Changes |
|---|---|---|
| December 2010 | 1.1 | Maintenance release. |
| April 2010 | 1.0 | Initial release. |

## How to Contact Altera

To locate the most up-to-date information about Altera products, refer to the following table.

| Contact *(1)* | Contact Method | Address |
|---|---|---|
| Technical support | Website | www.altera.com/support |
| Technical training | Website | www.altera.com/training |
| | Email | custrain@altera.com |
| Product literature | Website | www.altera.com/literature |
| Non-technical support (General) | Email | nacomp@altera.com |
| (Software Licensing) | Email | authorization@altera.com |

**Note to Table:**

(1) You can also contact your local Altera sales office or sales representative.

## Typographic Conventions

The following table shows the typographic conventions this document uses.

| Visual Cue | Meaning |
|---|---|
| **Bold Type with Initial Capital Letters** | Indicate command names, dialog box titles, dialog box options, and other GUI labels. For example, **Save As** dialog box. For GUI elements, capitalization matches the GUI. |
| **bold type** | Indicates directory names, project names, disk drive names, file names, file name extensions, software utility names, and GUI labels. For example, **\qdesigns** directory, **D:** drive, and **chiptrip.gdf** file. |
| *Italic Type with Initial Capital Letters* | Indicate document titles. For example, *Stratix IV Design Guidelines*. |
| *italic type* | Indicates variables. For example, *n* + 1.<br>Variable names are enclosed in angle brackets (< >). For example, *<file name>* and *<project name>*.**pof** file. |

| Visual Cue | Meaning |
|---|---|
| Initial Capital Letters | Indicate keyboard keys and menu names. For example, the Delete key and the Options menu. |
| "Subheading Title" | Quotation marks indicate references to sections within a document and titles of Quartus II Help topics. For example, "Typographic Conventions." |
| `Courier type` | Indicates signal, port, register, bit, block, and primitive names. For example, `data1`, `tdi`, and `input`. The suffix `n` denotes an active-low signal. For example, `resetn`.<br>Indicates command line commands and anything that must be typed exactly as it appears. For example, `c:\qdesigns\tutorial\chiptrip.gdf`.<br>Also indicates sections of an actual file, such as a Report File, references to parts of files (for example, the AHDL keyword `SUBDESIGN`), and logic function names (for example, `TRI`). |
| ↵ | An angled arrow instructs you to press the Enter key. |
| 1., 2., 3., and<br>a., b., c., and so on | Numbered steps indicate a list of items when the sequence of the items is important, such as the steps listed in a procedure. |
| ■ ■ ■ | Bullets indicate a list of items when the sequence of the items is not important. |
|  | The hand points to information that requires special attention. |
|  | A question mark directs you to a software help system with related information. |
|  | The feet direct you to another document or website with related information. |
|  | A caution calls attention to a condition or possible situation that can damage or destroy the product or your work. |
|  | A warning calls attention to a condition or possible situation that can cause you injury. |
|  | The envelope links to the Email Subscription Management Center page of the Altera website, where you can sign up to receive update notifications for Altera documents. |

# External Memory Interface Handbook

## Volume 6: Design Tutorials

101 Innovation Drive
San Jose, CA 95134
www.altera.com

EMI_TUT-2.1

Document last updated for Altera Complete Design Suite version: 10.1
Document publication date: December 2010

# External Memory Interface Handbook Volume 6

## Section I. ALTMEMPHY Design Tutorials

101 Innovation Drive
San Jose, CA 95134
www.altera.com

EMI_TUT_DDR-2.1

Document last updated for Altera Complete Design Suite version: 10.1
Document publication date: December 2010

Subscribe

# Contents

## Chapter 5. Using DDR3 SDRAM Devices in Stratix III and Stratix IV Devices

## Chapter 6. Using High-Performance DDR, DDR2, and DDR3 SDRAM with SOPC Builder



## Chapter 7. Implementing Multiple ALTMEMPHY-Based Controllers

# 1. Using High-Performance Controller II with Native Interface Design

This tutorial shows how to use your existing Native interface design with the high-performance controller II (HPC II) architecture. The HPC II architecture only supports the Avalon® Memory-Mapped (Avalon-MM) interface, and requires an adaptor to work with designs using the Native interface.

The HPC II architecture offers significantly more efficient memory access with better flexibility. As HPC II only supports the Avalon-MM interface, all Native interface designs migrating to HPC II architecture must adapt to the Avalon-MM interface. The only significant difference between the two interfaces is the write data timing. Using Avalon-MM interface, the user logic presents a write request, address, burst count, and the first beat of write data at the same time. The subsequent write data beats are placed into the FIFO buffer until they are needed. In the Native interface, the user logic presents a write request, address, and burst count. The controller then requests the correct number of write data beats by asserting the write data request signal.

This tutorial shows how you can compile and simulate using the native_to_avalon_adaptor design example.

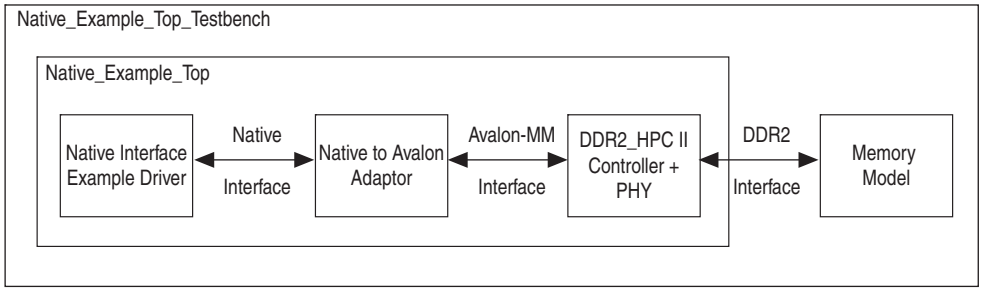
The design example implements a 200-MHz, 8-bit DDR2 SDRAM memory interface with a full-rate DDR2 SDRAM HPC II.

To download the design example, **native_to_avalon_adaptor.zip**, go to the External Memory Interface Design Examples page.

## Functional Description

Figure 1–1 shows the different components of the design example and how they are connected.

**Figure 1–1. Design Example Components**




The native_to_avalon_adaptor module in the design example enables designs with Native interface to work with the Avalon-MM interface of HPC II. The module registers all its outputs except these signals: `ntv_ready`, `ntv_rdata`, `ntv_rdata_valid`, and `ntv_wdata_req`.

The native_to_avalon_adaptor module indicates it is ready to accept commands on the Native interface by asserting the `ntv_ready` signal. The `ntv_ready` signal is not registered, so the signal is asserted at the same cycle as the `avl_ready` signal.

The Avalon read command is the same as the Native read command; so the Native interface signals are simply registered, and connected to the Avalon interface signals. The Avalon read data and valid signals are passed to the Native interface directly without being registered.

The write command and data adaptation is slightly more complicated. The native_to_avalon_adaptor module generates data requests to the Native interface master, when it sees a valid write command request by the master. The `ntv_wdata_req` signal path is combinatorial, so the native_to_avalon_adaptor module is able to generate the data requests on the same cycle as the command requests. A state machine tracks the outstanding write data requested by the native_to_avalon_adaptor module on the Native interface.

The module has a maximum count of write data cycles that can be tracked, and deasserts the `ntv_ready` signal when the maximum count is reached. The module only handles Native interface commands with a burst of 1 or 2.

## Performance

The adaptor has a command latency of one cycle for both read and write commands. Figure 1–2 on page 1–3 and Figure 1–3 on page 1–4 show how a read or write command is presented on the Avalon-MM interface on the following cycle after the command is presented on the Native interface.

**Figure 1–2. Read Commands**

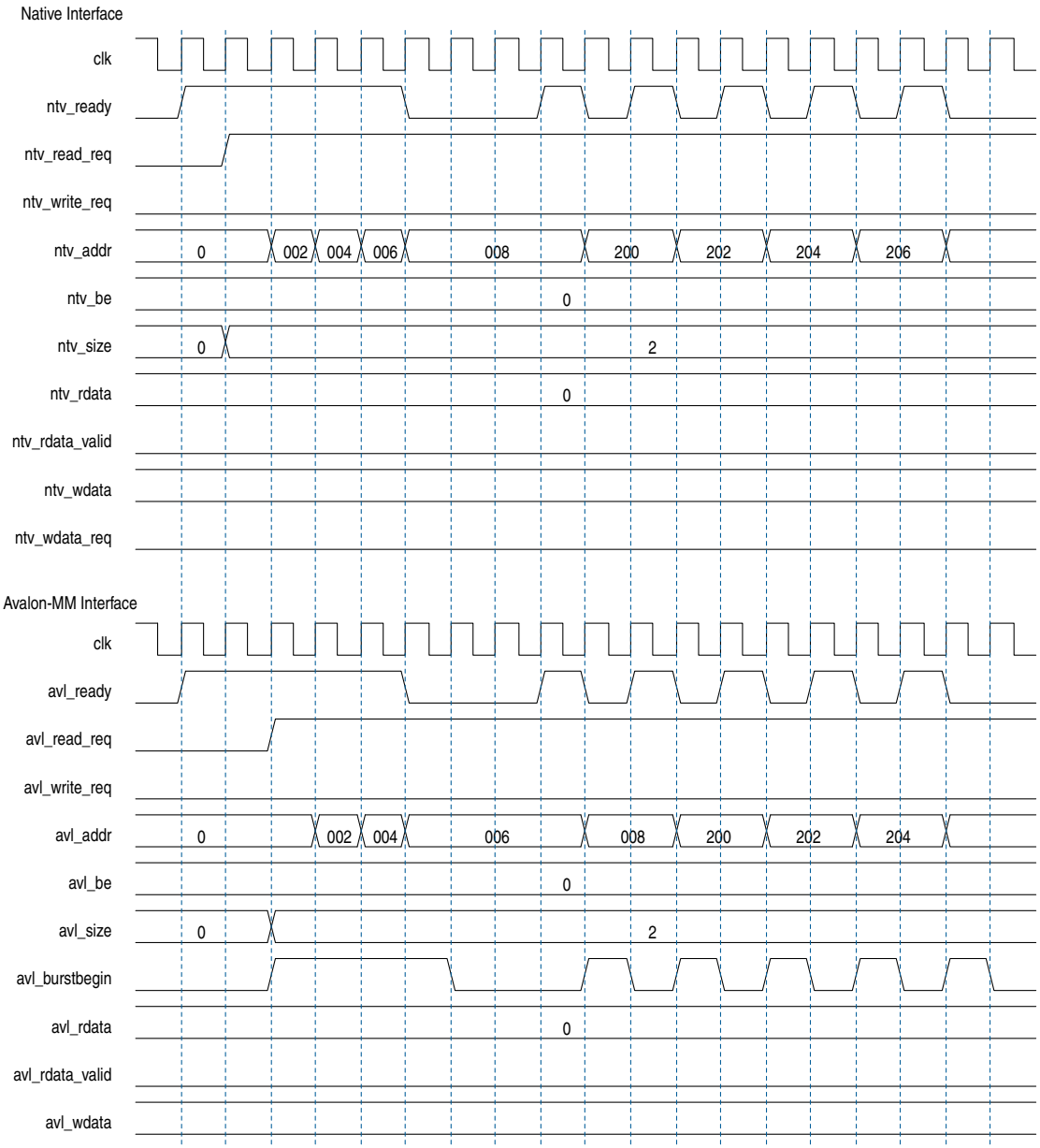
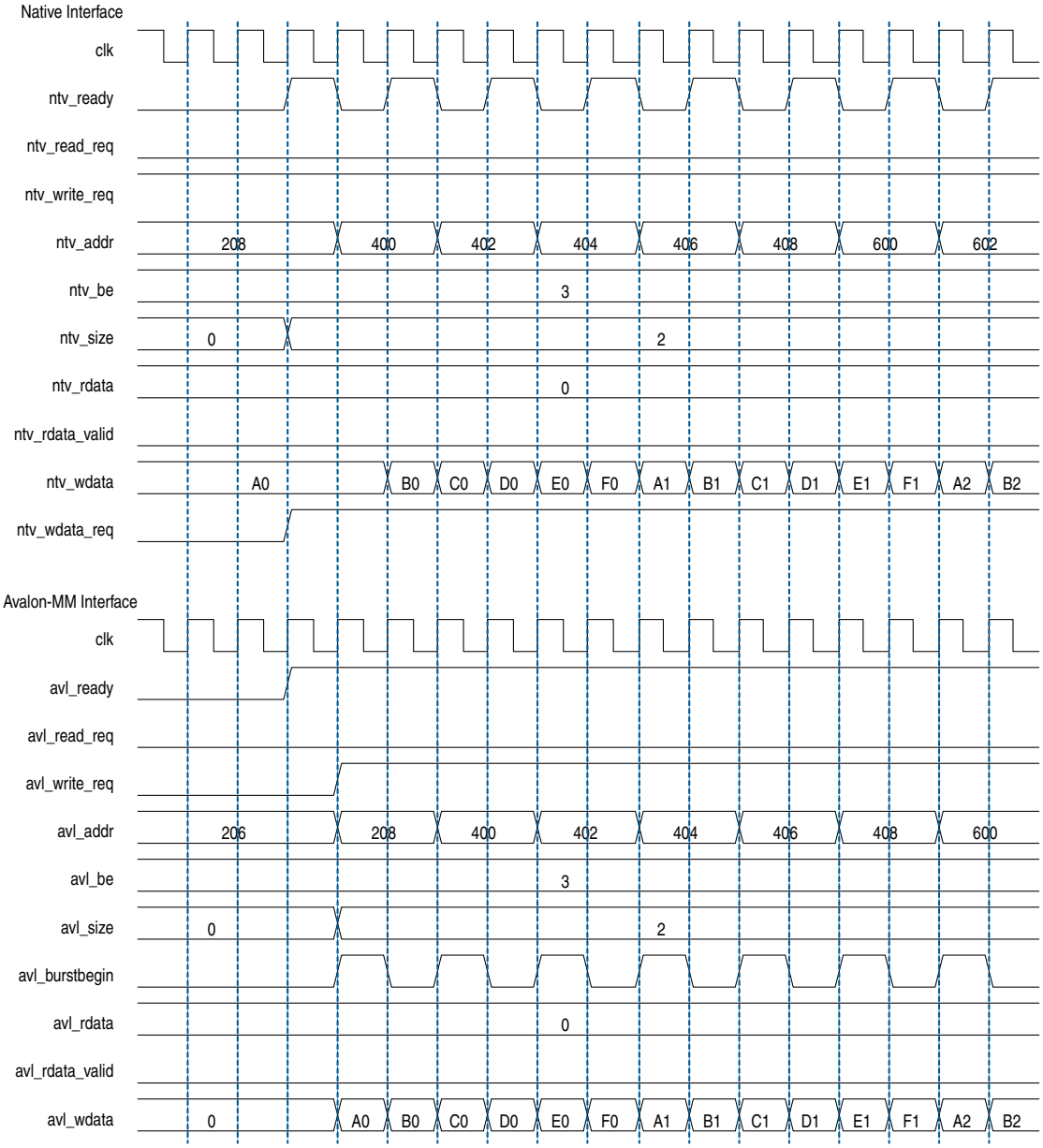
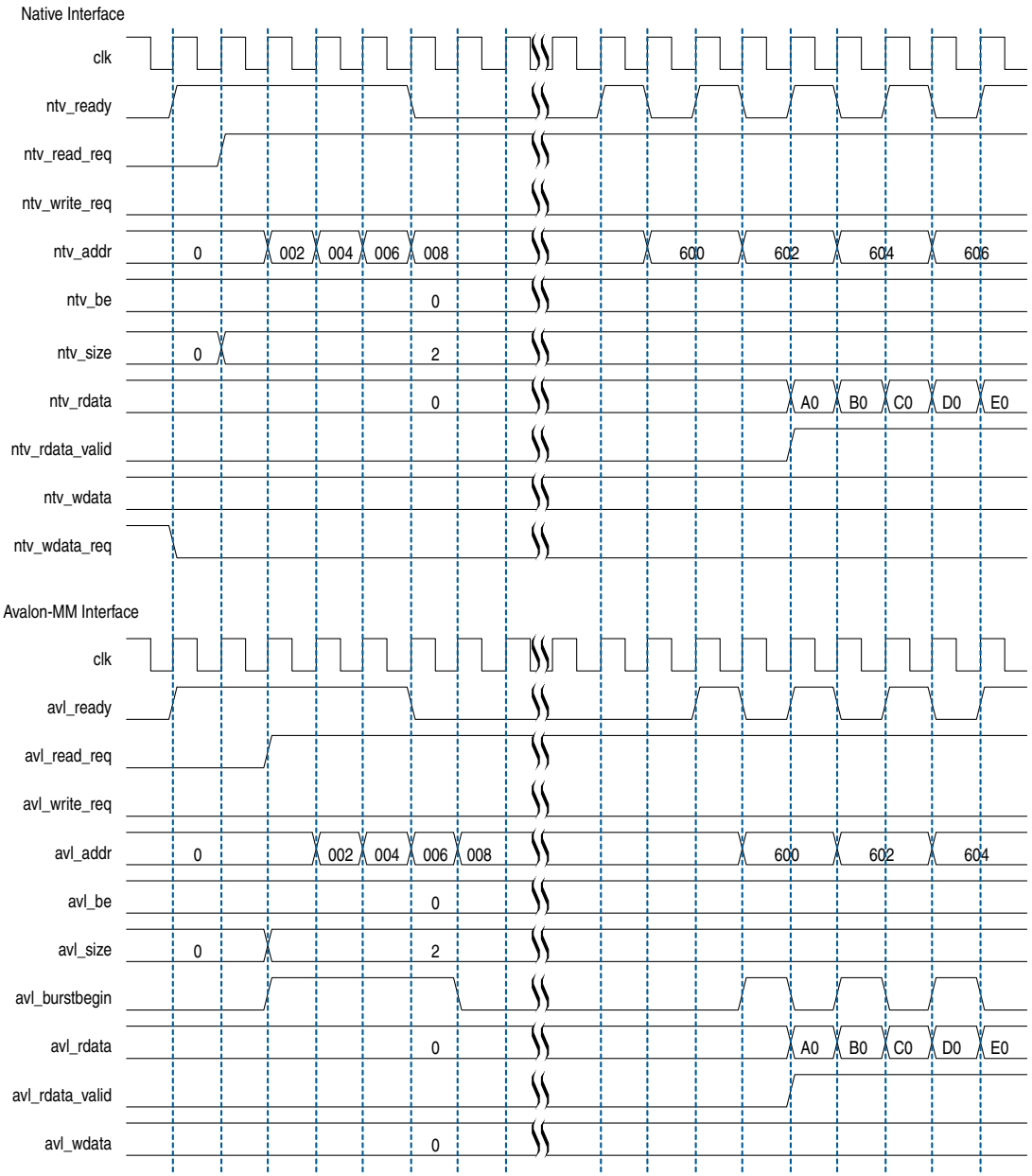


**Figure 1–3. Write Commands**



There are no latencies on the ntv_rdata and ntv_rdatavalid datapaths. Figure 1–4 shows that the data from a read command is presented on the Native interface, on the same cycle the data is presented on the Avalon-MM interface.

**Figure 1–4. Read Data Returned**




The adaptor asserts the ntv_wdata_req signal on the same cycle where the ntv_write_req signal is asserted. The Native interface master then presents the ntv_wdata signal on the subsequent cycle.

The adaptor uses a counter to keep track of outstanding write data beats that it needs to request on the Native interface. If the counter reaches a maximum value, the adaptor deasserts the ntv_ready signal to stop further commands on the Native interface. To increase the number of outstanding write data beats, increase the NTV_SIZE_COUNTER_WIDTH parameter to a sufficient value.

## Parameters

Table 1–1 shows the parameters for the native_to_avalon_adaptor module.

**Table 1–1. Module Parameters**

| Parameter | Default Setting | Description |
|---|---|---|
| AVL_ADDR_WIDTH | 24 | Set to match the HPCII local_address width. |
| AVL_BE_WIDTH | 2 | Set to match the HPCII local_be width. |
| AVL_LSIZE_WIDTH | 2 | Set to match the HPCII local_size width. |
| AVL_DATA_WIDTH | 16 | Set to match the HPCII local_wdata and local_rdata width. |
| NTV_SIZE_COUNTER-WIDTH | 8 | Controls the width of the counter that tracks outstanding write data beats on the Native interface. |
| NTV_STATE_WIDTH | 3 | Fixed value. |

## Signals

Table 1–2 through Table 1–4 show some of the signals for the native_to_avalon_adaptor design example.

**Table 1–2. Clock and Reset Signals**

| Port | Direction | Description |
|---|---|---|
| clk | Input | Module clock |
| reset_n | Input | Module reset |

**Table 1–3. Native Interface Signals**

| Port | Direction |
|---|---|
| ntv_read_req | Input |
| ntv_write_req | Input |
| ntv_addr | Input |
| ntv_be | Input |
| ntv_size | Input |
| ntv_wdata | Input |
| ntv_ready | Output |
| ntv_rdata | Output |
| ntv_rdata_valid | Output |
| ntv_wdata_req | Output |

**Table 1–4. Avalon-MM Interface Signals**

| Port | Direction | Direction |
|---|---|---|
| avl_ready | Input | local_ready |
| avl_rdata | Input | local_rdata |
| avl_rdata-valid | Input | local_rdata_valid |
| avl_read_req | Output | local_read_req |
| avl_write_req | Output | local_write_req |
| avl_addr | Output | local_address |
| avl_be | Output | local_be |
| avl_size | Output | local_size |
| avl_wdata | Output | local_wdata |
| avl_burstbegin | Output | local_burstbegin |

# Getting Started

This section discusses the requirements and related procedures to compile and simulate the design example. This section contains the following topics:

- Software Requirements
- Directory Structure

- Compile the Design
- Simulate the Design

## Software Requirements

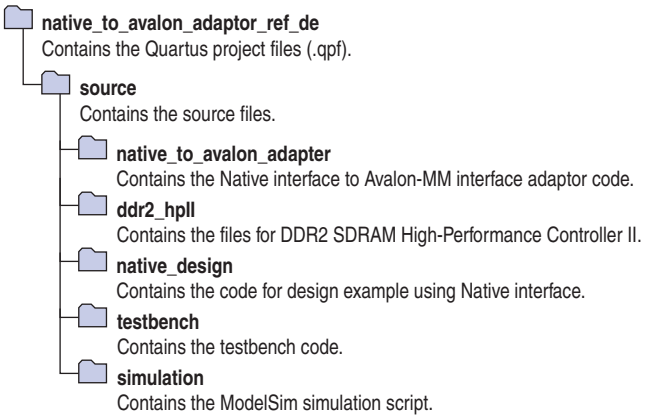
This design example requires the following software:

- Quartus® II software, version 9.1 or later
- ModelSim®-Altera® software, version 6.5b or later
- DDR2 Controller with ALTMEMPHY IP version 9.1

## Directory Structure

Figure 1–5 shows the directory structure of the design example.

**Figure 1–5. Directory Structure**




## Compile the Design

To compile the design example, perform the following steps:

1. Launch the Quartus II software.
2. On the File menu, click **Open Project**, navigate to *<your project path>***/native_to_avalon_adaptor_ref_de/native_to_avalon_adaptor_ref_de.qpf**, and click **Open**.
3. On the Processing menu, click **Start Compilation**.

## Simulate the Design

To set up the simulation environment, and start the simulation, perform the following steps:

1. Launch **ModelSim-Altera**.
2. On the File menu, click **Change Directory**.
3. Select *<your project path>***/native_to_avalon_adaptor_ref_de/source/simulation/modelsim** and click **OK**.

4. On the Tools menu, click **Tcl**, then click **Execute Macro.** Select **native_example_top_run_msim_rtl_verilog.tcl**, and click **Open** to start the simulation.

Alternatively, you can run the script in the ModelSim transcript window by typing

```
source ./native_example_top_run_msim_rtl_verilog.tcl,
```

or in a terminal console by typing

```
vsim -do ./native_example_top_run_msim_rtl_verilog.tcl.
```

# 2. Using DDR, DDR2, and DDR3 SDRAM Devices in Arria II GX Devices

This tutorial describes how to use the design flow to design a 64-bit wide, 267-MHz, 533-Mbps DDR2 SDRAM interface, and a 16-bit wide, 300-MHz, 600-Mbps DDR3 SDRAM interface. The design examples provide some recommended settings, including termination scheme and drive strength settings, to simplify the design. You can also use the DDR2 SDRAM design example to design a DDR SDRAM interface.

The design examples target the Arria® II GX FPGA development kit, which includes a 64-bit wide 1-GB Micron MT8HTF12864HDY-800G1 400-MHz DDR2 SDRAM SODIMM, and a 16-bit wide 1-GB Micron MT41J64M16LA-15E DDR3 SDRAM component.

To download the design examples, **emi_ddr2_aii.zip** and **emi_ddr3_aii.zip**, go to the External Memory Interface Design Examples page.For more information about the design flow, refer to the *Recommended Design Flow* section in volume 1 of the *External Memory Interface Handbook*.

## System Requirements

This tutorial requires the following hardware and software:

- DDR2 SDRAM interface:
  - Quartus II software version 9.1
  - DDR2 SDRAM Controller with ALTMEMPHY IP version 9.1
  - Arria II GX FPGA Development Kit with the EP2AGX125EF35C5 device
- DDR3 SDRAM interface:
  - Quartus II software version 9.1
  - DDR3 SDRAM Controller with ALTMEMPHY IP version 9.1
  - Arria II GX FPGA Development Kit with the EP2AGX260FF35C4 device

## Create a Quartus II Project

Create a project in the Quartus II software that targets the respective device; EP2AGX260FF35C5 for the DDR2 SDRAM interface or EP2AGX260FF35C4 for the DDR3 SDRAM interface.

For step-by-step instructions on how to create a Quartus II project, refer to the Quartus II software tutorial by clicking the Help menu in the Quartus II window and selecting **PDF Tutorials**.

## Instantiate and Parameterize a Controller

After creating a Quartus II project, you need to instantiate a controller and set its parameters.

## Instantiate a Controller

To instantiate a controller, perform the following steps:

- DDR2 SDRAM controller
  a. Copy the memory parameters file, **ArriaIIGX_DDR2_Kit(MT8HTF12864HDY-800G1).xml**, to your <*installation directory*>**\91\ip\ddr2_high_perf\lib** directory.
  b. Launch the MegaWizard™ Plug-in Manager.
  c. In the MegaWizard Plug-In Manager, expand **External Memory** in the **Interfaces** folder and select the **DDR2 SDRAM High Performance Controller**.
  d. Enter `ddr2_sodimm` for the name of the DDR2 SDRAM high-performance controller.
- DDR3 SDRAM controller
  a. Copy the memory parameters file, **ArriaIIGX_DDR3_Kit(MT41J64M16LA-15E).xml**, to your <*installation directory*>**\91\ip\ddr3_high_perf\lib** directory.
  b. Launch the MegaWizard Plug-in Manager.
  c. In the MegaWizard Plug-In Manager, expand **External Memory** in the **Interfaces** folder and select the **DDR3 SDRAM High Performance Controller.**
  d. Enter `ddr3` for the name of the DDR3 SDRAM high-performance controller.

Both the DDR2 and DDR3 design examples instantiate the ALTMEMPHY megafunction automatically.

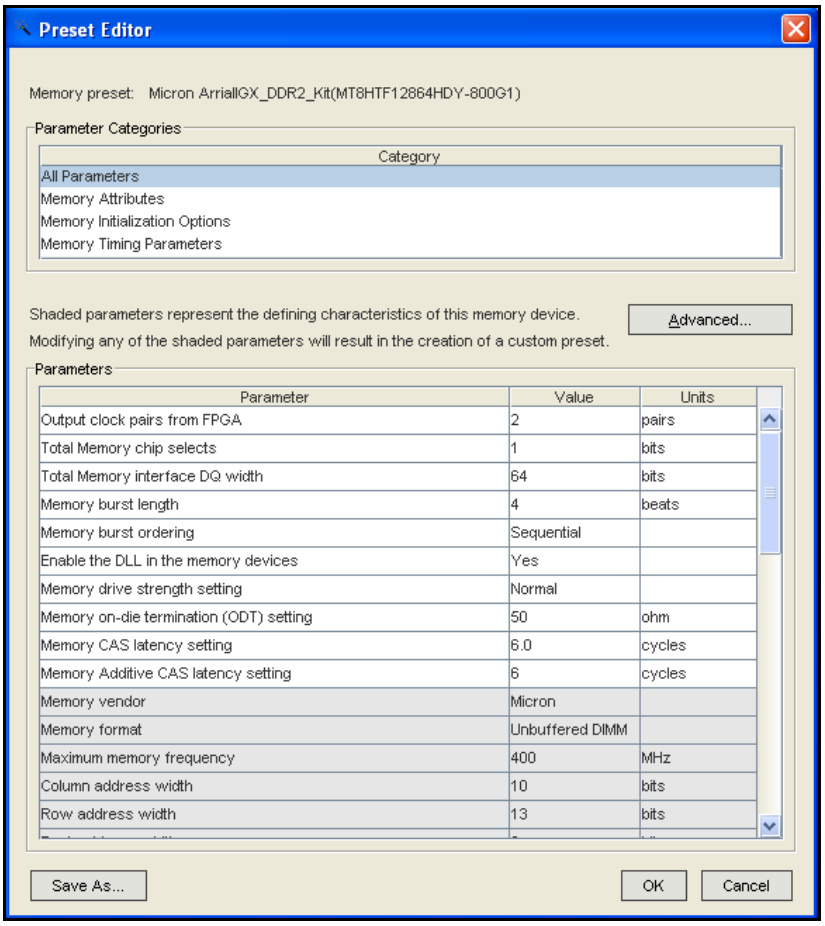
## Parameterize DDR2 SDRAM

To parameterize the DDR2 high-performance controller to interface with a 267-MHz 64-bit wide DDR2 SDRAM interface, perform the following steps:

1. In the **Memory Setting** tab, set **Speed grade** to 5.
2. For **PLL reference clock frequency**, enter **100** MHz. **The input clock source,** `clock_source`, supplies the PLL **reference clock frequency.**
3. For **Memory clock frequency**, enter **267** MHz. This value is the maximum frequency supported for the DDR2 SDRAM interface on Arria II GX devices.

4. For **Memory Presets**, select **Micron ArriaIIGX_DDR2_Kit(MT8HTF12864HDY-800G1)**, which gives a 64-bit wide 1-GB 400-MHz DDR2 SODIMM.To create or modify a memory preset, click **Modify parameters**. In the **Preset Editor** dialog box, modify the memory presets. Refer to Figure 2–1 on page 2–3.

**Figure 2–1. Modify the Memory Presets to Create a Custom Memory**




5. Turn on the **Enable Memory Chip Calibration in Timing Analysis** option in the **Advanced** page. This option is required for Arria II GX devices, which need post-processing script to remove timing model pessimism.
6. In the **PHY Settings** tab, under **Advanced PHY Settings**, turn on the U**se differential DQS** option to enhance signal to noise ratio. Turn on this option if noise margin is a concern.

7. By default, **Clock Phase** is set to **90** for **Address/Command Clock Settings**. The **Auto-Calibration Simulation Options** parameter is for RTL simulation only and is not applicable for gate-level simulation. ALTMEMPHY does not support gate-level timing simulation.In the **Board Settings** tab, enter the following values for the parameters under **Slew Rates** and **Board Skews**:
   - **CK/CK# slew rate (Differential)** = **2.475 V/ns**
   - **Addr/Command slew rate** = **0.859 V/ns**
   - **DQS/DQS# slew rate (Differential)** = **1.386 V/ns**
   - **DQ slew rate** = **2.665 V/ns**

   
   These slew rates are obtained from a simulation using the default I/O standard and drive options.

   - Max skew within DQS group = 0.0339 ns
   - Max skew between DQS groups = 0.0824 ns
   - Addr/Command to CK skew = 0.0356 ns

   The Intersymbol Interference parameters are not applicable for single rank configurations. Set all these parameters to **0 ns**.
8. In the **Controller Settings** tab, for **Controller Architecture**, select **High Performance Controller II** for higher efficiency and advanced features.
9. Under **Efficiency**, select the specified values for the following options:
   a. For **Command Queue Look-Ahead Depth**, select **6**.
   b. For **Local-to-Memory Address Mapping**, select **CHIP-ROW-BANK-COL**.
   c. For **Local Maximum Burst Count**, select **8**.
10. Click **Next**.
11. Turn on the **Generate simulation model** option to generate the simulation model for the RTL simulation.
12. Click **Finish** to generate your MegaCore function variation. The MegaWizard Plug-In Manager generates all the files necessary for your DDR2 SDRAM controller and a top-level design, which you use to test or verify the board operation.


For more information about the DDR/DDR2 SDRAM high-performance controller, refer to the *DDR and DDR2 SDRAM Controller with ALTMEMPHY IP User Guide* in volume 3 of the *External Memory Interface Handbook*.

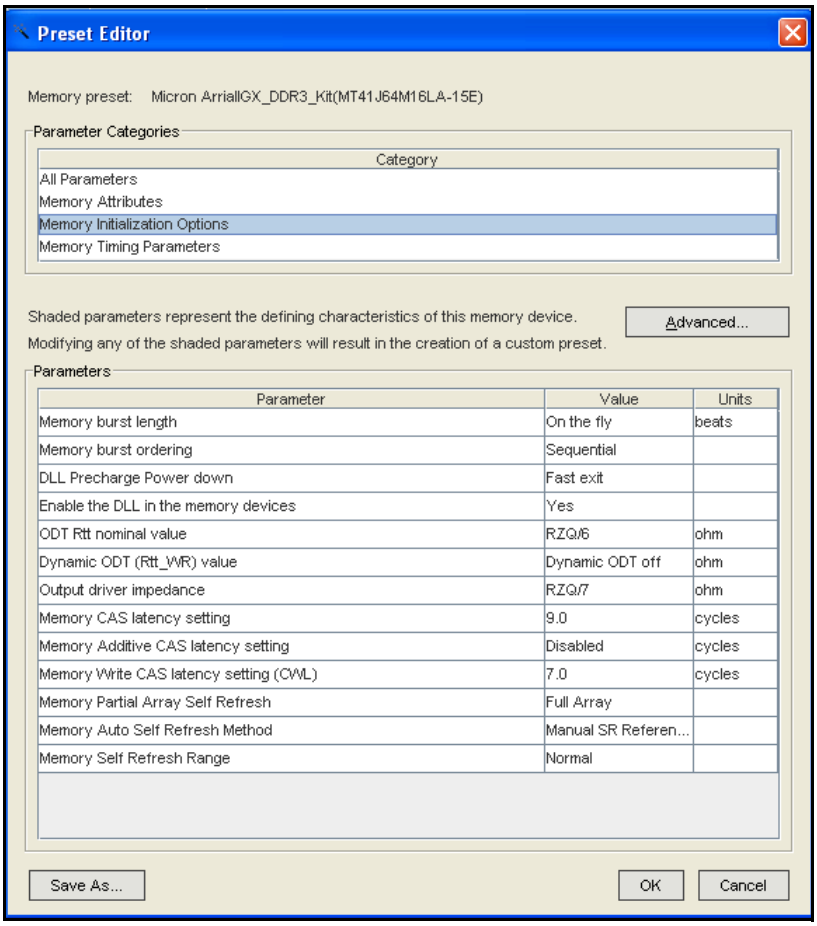
## Parameterize DDR3 SDRAM

To parameterize the DDR3 high-performance controller to interface with a 300-MHz, 16-bit wide, DDR3 SDRAM interface, perform the following steps:

1. In the **Memory Setting** tab, set **Speed grade** to 4.
2. For **PLL reference clock frequency**, enter **100** MHz, to match the on-board oscillator**.**

3. For **Memory clock frequency**, enter **300** MHz. This is the maximum frequency supported for DDR3 SDRAM interface on Arria II GX devices.
4. For **Memory Presets**, select **Micron ArriaIIGX_DDR3_Kit(MT41J64M16LA-15E)**, which gives a 16-bit wide 1-GB 667-MHz DDR3 component. To create or modify a memory preset, click **Modify parameters**. In the **Preset Editor** dialog box, modify the memory presets (Figure 2–2 on page 2–5).

**Figure 2–2. Modify the Memory Presets to Create a Custom Memory**




5. In **Advanced** page, turn on the **Enable Memory Chip Calibration in Timing Analysis** option. This option is required for Arria II GX devices, which need post-processing script to remove timing model pessimism.
6. Under **Address/Command Clock Settings**, use the default value of **Clock Phase,** which is **90**.
7. In the **Board Settings** tab, set the following **Slew Rates** and **Board Skews** parameters to the specified values:
   - **CK/CK# slew rate (Differential) = 4 V/ns**
   - **Addr/Command slew rate = 2 V/ns**
   - **DQS/DQS# slew rate (Differential) = 1.8 V/ns**
   - **DQ slew rate = 1.5 V/ns**

These slew rates are obtained from simulation using the default I/O standard and drive options.

- **Max skew within DQS group = 0.020 ns**
- **Max skew between DQS groups = 0.020 ns**
- **Addr/Command to CK skew = -0.045 ns**

The **Intersymbol Interference** parameters are not applicable for single rank configurations. Set all these parameters to **0 ns**.

8. In the **Controller Settings** table, for **Controller Architecture**, select **High Performance Controller II** for higher efficiency and advanced features.Under **Efficiency**, select the specified values for the following options:
   a. For **Command Queue Look-Ahead Depth**, select **6**.
   b. For **Local-to-Memory Address Mapping**, select **CHIP-ROW-BANK-COL**.
   c. For **Local Maximum Burst Count**, select **4**.
9. Click **Next**.
10. Turn on the **Generate simulation model** option to generate the simulation model for the RTL simulation.
11. Click **Finish** to generate your MegaCore function variation. The MegaWizard Plug-In Manager generates all the files necessary for your DDR3 SDRAM controller and generates an example top-level design, which you use to test or verify board operation.

For more information about the DDR3 SDRAM high-performance controller, refer to the *DDR3 SDRAM Controller with ALTMEMPHY IP User Guide* in volume 3 of the *External Memory Interface Handbook*.

# Add Constraints

After instantiating the DDR2 or DDR3 SDRAM high-performance controller, the ALTMEMPHY megafunction generates the constraint files for the design example. You must apply these constraints to the design before compilation.

## Set the Top-Level Entity

The top-level entity of the project must be set to the correct entity. The naming convention for an ALTMEMPHY megafunction entity is <*variation_name*>**_phy.v** or **vhd**; an SDRAM high-performance controller entity is <*variation_name*>**.v** or **vhd**.

To set the top-level file, perform the following steps:

1. Open the entity file, **ddr2_sodimm_example_top.v** or **vhd** for the DDR2 design example or **ddr3_example_top.v** or **vhd** for the DDR3 design example.
2. On the Project menu, click **Set as Top-Level Entity**.

## Set the Optimization Technique

To ensure the remaining unconstrained paths are routed with the highest speed and efficiency, perform the following steps to set the optimization technique:

1. On the Assignments menu, click **Settings**.
2. Select **Analysis & Synthesis Settings**.
3. Select **Speed** under **Optimization Technique**. Click **OK**.

## Set the Fitter Effort

To set the fitter effort, perform the following steps:

1. On the Assignments menu, click **Settings**.
2. Select **Fitter Settings**.
3. Turn on **Optimize hold timing** and select **All Paths**.
4. Turn on **Optimize multi-corner timing**.
5. Under **Fitter effort**, select **Standard Fit (highest effort)**.
6. Click **OK**.

## Add Timing Constraints

When you instantiate an SDRAM high-performance controller, it generates the timing constraints file <*variation_name*>**_phy_ddr_timing.sdc**, or <*variation_name*>**_phy_ddr3_timing.sdc**. The timing constraint file constrains the clock and input and output delay on the SDRAM high-performance controller.

To add the timing constraints, perform the following steps:

1. On the Assignments menu, click **Settings**.
2. In the **Category** list, expand **Timing Analysis Settings** and select **TimeQuest Timing Analyzer**.
3. Select the <*variation_name*>**_phy_ddr_timing.sdc** file for DDR2, or <*variation_name*>**_phy_ddr3_timing.sdc** file for DDR3 and click **Add**.
4. Click **OK**.

## Add Pin and DQ Group Assignments

The pin assignment script, <*variation_name*>**_pin_assignments.tcl**, sets up the I/O standards for the memory interface.

This script does not create a clock for the design. You must create a clock for the design and provide pin assignments for the signals of both the example driver and testbench that the MegaCore variation generates.

To add the pin and I/O standards to the design example, perform the following steps:

1) On the Tools menu, click **Tcl scripts**.

2) Under **Libraries**, select <*variation_name*>**_pin_assignments.tcl**

3) Click **Run**.

## Enter Pin Location Assignments

To enter the pin location assignments using the Pin Planner, perform the following steps:

1. Run analysis and synthesis. On the Processing menu, point to **Start** and click **Start Analysis & Synthesis**.
2. Assign all of your pins, so that the Quartus II software fits your design correctly and gives correct timing analysis. Manually assign the pin locations by using the Pin Planner or Assignment Editor. To assign the pin locations for the Arria II GX Development Kit, run the Altera-provided **ArriaIIGX_DDR2_PinLocations.tcl** script for the DDR2 SDRAM interface design or the **ArriaIIGX_DDR3_PinLocations.tcl** script for the DDR3 SDRAM interface design.

   If you are at the design exploration phase of your design cycle and do not have any PCB defined pin locations, you must still manually define an initial set of pin constraints, which can become more specific during your development process.

To manually assign pin locations using Pin Planner, perform the following steps:

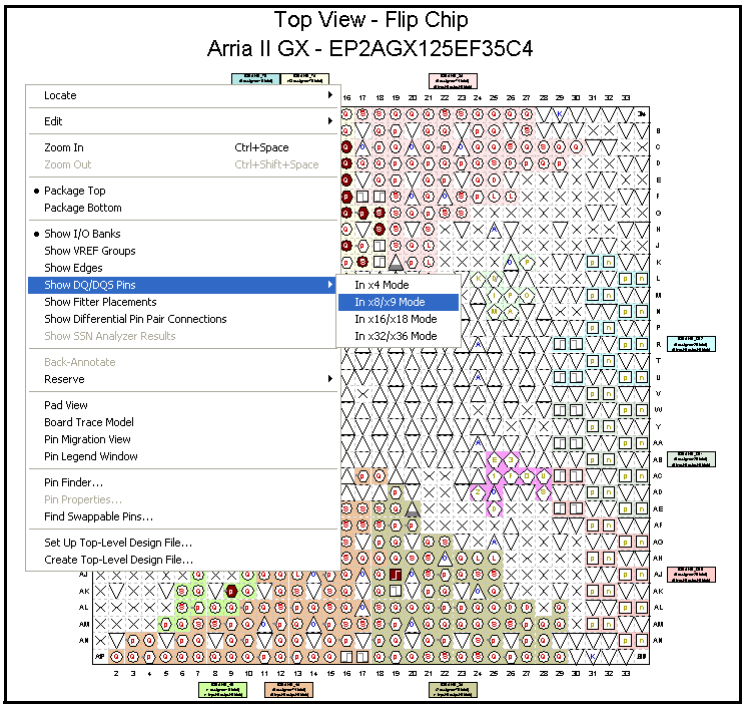
1. On the Assignments menu, click **Pin Planner**.
2. Assign DQ and DQS pins.
   a. To select the device DQS pin groups that the design uses, assign each DQS pin in your design to the required DQS pin in the Pin Planner. The Quartus II Fitter then automatically places the respective DQ signals onto suitable DQ pins within each group. To see DQS groups in Pin Planner, right click, select **Show DQ/DQS Pins**, and click **In ×8/×9 Mode**. Pin Planner shows each DQS group in a different color and with a different legend: S = DQS pin, Sbar = DQSn pin and Q = DQ pin (refer to Figure 2–3).

      Most memory devices operate in ×8/×9 mode; however, as some memory devices operate in ×4 mode, refer to your specific memory device datasheet.

   b. Select the DQ mode to match the DQ group width (number of DQ pins/number of DQS pins) of your memory device. DQ mode is not related to the memory interface width.

DQ group order and DQ pin order within each group is not important. However, you must place the DQ pins in the same group as their respective strobe pin.

**Figure 2–3. Quartus II Pin Planner, Show DQ/DQS Pins in ×8/×9 Mode**




3. Place the DM pins within their respective DQ group.
4. Place address and control command pins on any spare I/O pins, ideally in the same bank or side of the device as the mem_clk pins.
5. Ensure that you place mem_clk pins on differential I/O pairs for the CK/CK# pin pair. To identify the differential I/O pairs, right-click in the Pin Planner and select **Show Differential Pin Pair Connections**. The pin pairs show a red line between each pin pair.
6. Ensure that the mem_clk pins use any regular adjacent I/O pins; ideally the differential I/O pairs for the CK/CK# pin pair. To identify the differential I/O pairs, right-click in Pin Planner and select **Show Differential Pin Pair Connections**. Pin pairs show a red line between each pin pair.
7. Place the clock_source pin on a dedicated PLL clock input pin with a direct connection to the SDRAM controller PLL and DLL pair—usually on the same side of the device as your memory interface. This recommendation reduces PLL jitter, saves a global clock resource, and eases timing and fitter effort.
8. Place the global_reset_n pin (as any high fan-out signal) on a dedicated clock pin.
9. Ensure that you place the $R_{UP}$ and $R_{DN}$ pins, termination_blk0~_rdn_pad and termination_blk0~_rup_pad, at locations within the same $V_{CCIO}$ voltage bank.

For more information about how to use the Quartus II Pin Planner, refer to the *I/O Management* chapter in volume 2 of the *Quartus II Handbook*.

## Assign I/O Standards

To assign the I/O standards, perform the following steps:

1. On the Assignments menu, click **Assignment Editor**.
2. Specify **LVDS** as the I/O standard for `clock_source`.
3. Specify **1.8** V for DDR2 SDRAM, or **1.5** V for DDR3 SDRAM as the I/O standard for `global_reset_n` and `mem_odt[0]`.

## Assign Virtual Pins

The example top-level design, which is auto-generated by the high-performance controller, includes an example driver to stimulate the interface. This example driver is not part of the SDRAM high-performance controller IP, but allows easy testing of the IP.

The example driver outputs several test signals to indicate its operation and the status of the stimulated memory interface. These signals are `pnf`, `pnf_per_byte`, `test_complete`, and `test_status`. These signals are not part of the memory interface, but are for testing. You must either take these signals to a debug header or set the signals to virtual pins using the Quartus II Assignment Editor. When using the example driver for testing, do not remove these signals from the top-level signal list; otherwise, the Quartus II software optimizes the driver away and the example driver fails.

To assign virtual pin assignments for the Arria II GX development board, run the Altera-provided **ArriaIIGX_DDR2_exdriver_vpin.tcl** file for the DDR2 SDRAM design example or **ArriaIIGX_DDR3_exdriver_vpin.tcl** file for DDR3 SDRAM design example, or manually assign the virtual pin assignments using the Assignment Editor.

## Enter Board Trace Delay Models

For accurate I/O timing analysis, the Quartus II software must be aware of the board trace and loading information. This information must be derived and refined during your PCB development process of pre-layout (line) simulation through post-layout (board) simulation.

For external memory interfaces that use memory modules (DIMMs), the board trace and loading information must also include the trace and loading information of the module, which you can obtain from your memory vendor.

To enter board trace information, perform the following steps:

1. In the Pin Planner, select the pin or group of pins that you want to enter the information for.

2. Right-click and select **Board Trace Model, as shown in** Figure 2–4.

**Figure 2–4. Board Trace Model**

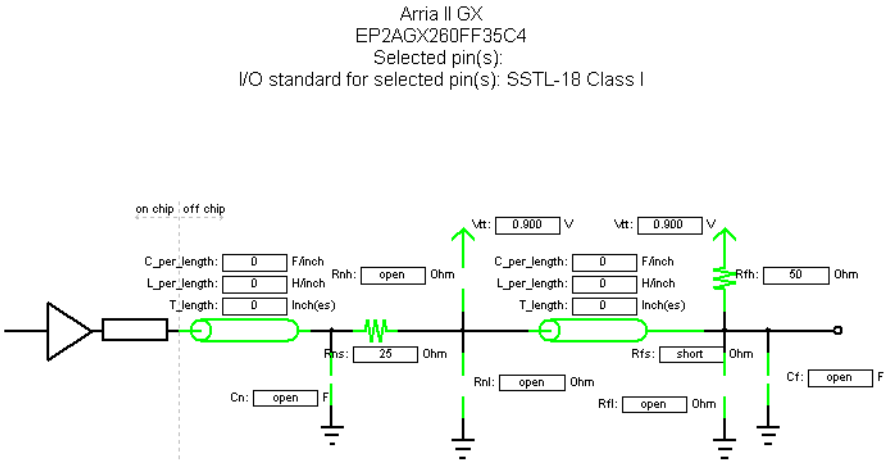



Table 2–1 shows the board trace model parameters for the Arria II GX development board.

**Table 2–1. Arria II GX Development Board Trace Model Summary for DDR2 and DDR3 (Part 1 of 2)**

| Net | Near (FPGA End of Line) | | | | | | Far (Memory End of Line) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Length (Inch) | L_per_length (nH/In) | C_per_length (pF/In) | Cn (pF) | Rns | Rnh | Length (Inch) | L_per_length (nH/In) | C_per_length (pF/In) | Cf (pF) | Rfh/R (pF) |
| **DDR2 SODIMM** | | | | | | | | | | | |
| Addr *(1)* | 1.743 | 8 | 4.04 | — | 3 | 56 | 2.230 | 10.6 | 3.06 | 16 | — |
| CLK0 | 2.614 | 8.66 | 3.74 | — | — | — | 1.017 | 10.6 | 3.06 | 8 | 100 |
| CLK1 | 1.616 | 8.66 | 3.74 | — | — | — | 1.020 | 10.6 | 3.06 | 8 | 100 |
| CKE/CS# | 1.599 | 8 | 4.04 | — | 3 | 56 | 2.155 | 10.6 | 3.06 | 2 | |
| DQS0 | 1.866 | 8 | 4.04 | — | 3 for dq, 22 for dqs | 56 | 0.637 | 10.6 | 3.06 | 4 | — |
| DQS1 | 1.757 | 8 | 4.04 | — | 3 for dq, 22 for dqs | 56 | 0.661 | 10.6 | 3.06 | 4 | — |
| DQS2 | 1.437 | 8 | 4.04 | — | 3 for dq, 22 for dqs | 56 | 0.741 | 10.6 | 3.06 | 4 | — |
| DQS3 | 1.457 | 8 | 4.04 | — | 3 for dq, 22 for dqs | 56 | 0.761 | 10.6 | 3.06 | 4 | — |
| DQS4 | 1.450 | 8 | 4.04 | — | 3 for dq, 22 for dqs | 56 | 0.749 | 10.6 | 3.06 | 4 | — |

**Table 2–1. Arria II GX Development Board Trace Model Summary for DDR2 and DDR3 (Part 2 of 2)**

| Net | Near (FPGA End of Line) | | | | | | Far (Memory End of Line) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Length (Inch) | L_per_length (nH/In) | C_per_length (pF/In) | Cn (pF) | Rns | Rnh | Length (Inch) | L_per_length (nH/In) | C_per_length (pF/In) | Cf (pF) | Rfh/R (pF) |
| DQS5 | 1.548 | 8 | 4.04 | — | 3 for dq, 22 for dqs | 56 | 0.667 | 10.6 | 3.06 | 4 | — |
| DQS6 | 1.677 | 8 | 4.04 | — | 3 for dq, 22 for dqs | 56 | 0.665 | 10.6 | 3.06 | 4 | — |
| DQS7 | 1.831 | 8 | 4.04 | — | 3 for dq, 22 for dqs | 56 | 0.666 | 10.6 | 3.06 | 4 | — |
| **DDR3** *(2)* | | | | | | | | | | | |
| Addr *(1)* | 1.879 | 8.66 | 3.74 | — | — | — | — | — | — | 1.3 | 56 |
| CLK | 1.359 | 8.66 | 3.74 | — | — | — | — | — | — | 1.4 | 100 |
| CKE/CS# | 1.775 | 8.66 | 3.74 | — | — | — | — | — | — | 1.3 | 56 |
| DQS0 | 1.278 | 8.66 | 3.74 | — | — | — | — | — | — | 2.5 | 56 |
| DQS1 | 1.249 | 8.66 | 3.74 | — | — | — | — | — | — | 2.5 | 56 |

**Note to Table 2–1:**

(1) Addr = Addr, ba, we#, ras#, odt, and cas.

(2) The near (FPGA end of line) board trace model summary is not applicable for memory components configuration.

Altera recommends that you use the **Board Trace Model** assignment for all memory interface signals. To apply the board trace model assignments for the Arria II GX FPGA development kit, run the Altera-provided **ArriaIIGX_DDR2_BTModels.tcl** script for the DDR2 SDRAM interface design, or the **ArriaIIGX_DDR3_BTModels.tcl** script for the DDR3 SDRAM interface design, or manually assign virtual pin assignments using the Quartus II Pin Planner.

# Perform RTL or Functional Simulation (Optional)

After instantiating the DDR2 or DDR3 SDRAM high-performance controller, the Quartus II software generates a design example that includes driver, test bench, and memory model that allows you to perform functional simulation on your design.

The Verilog HDL or VHDL simulation model of the PHY, *<variation_name>*_**alt_mem_phy_sequencer_wrapper.vo/.vho** file, is located in your project directory.

To run the simulation with NativeLink, perform the following steps:

1. Set the absolute path to your third-party simulator executable file, by performing the following steps:
   a. On the Assignments menu, point to **EDA Tool Settings**.
   b. In the **Category** list, expand **EDA Tool Settings** and click **Simulation**.
   c. Under **Tool name**, select **ModelSim**.
   d. Under **NativeLink settings**, select **Compile test bench** and click **Test Benches**.
   e. Click **New**.
   f. Type the name of your testbench top-level module and simulation period.
   g. Click **OK**.
2. To elaborate your design, on the Processing menu, point to **Start** and click **Start Analysis & Elaboration**.
3. On the Tools menu, point to the **Run EDA Simulation Tool** and click **EDA RTL Simulation**. This step creates the **\simulation** directory in your project directory and a script that compiles all necessary files and runs the simulation.

# Compile Design and Verify Timing

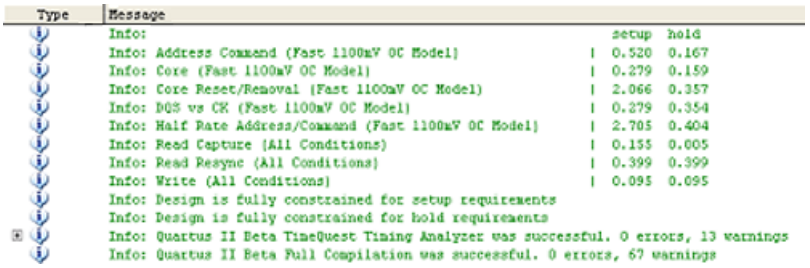
To compile the design, on the Processing menu, click **Start Compilation**. After successfully compiling the design, the Quartus II software automatically runs the verify timing script, which produces a timing report for the design together with the compilation report.

The report timing script provides information about the following margins and paths:

- Address and command setup and hold margin
- Core setup and hold margin
- Core reset, removal setup, and hold margin
- Read capture setup and hold margin
- Write setup and hold margin
- Read resync setup and hold margin
- DQS vs CK setup and hold margin

Figure 2–5 shows the timing margin report for a DDR2 SDRAM interface design in the message window in the Quartus II software.

**Figure 2–5. Timing Margin Report in the Quartus II Software**

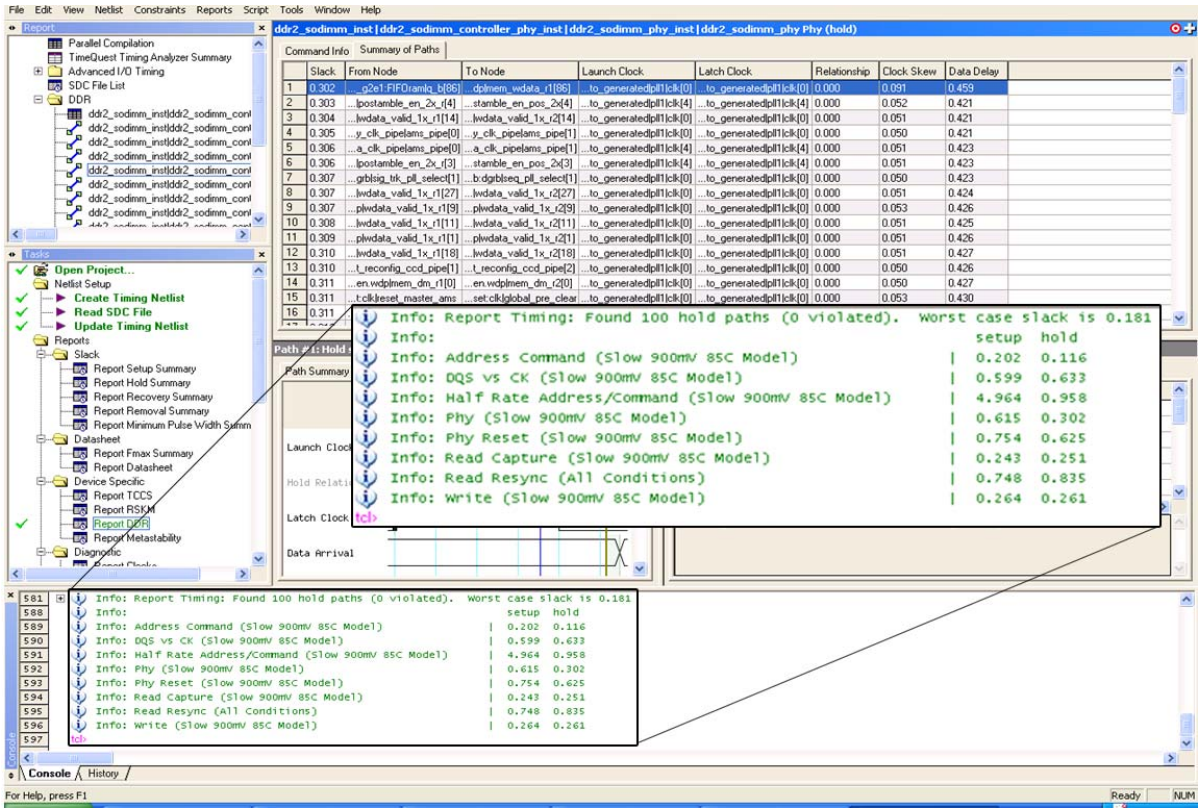


You can also obtain the timing report by running the report timing script (**ddr2_sodimm_phy_report_timing.tcl** or **ddr3_phy_report_timing.tcl)** in the TimeQuest timing analyzer. To obtain the timing report in the TimeQuest Timing Analyzer window, perform the following steps:

1. In the Quartus II software, on the Tools menu, click **TimeQuest Timing Analyzer**.
2. On the **Tasks** pane, double-click **Report DDR** to run **Update Timing Netlist**, **Create Timing Netlist**, and **Read SDC File**. This command subsequently executes the report timing script to generate the timing margin report.

Figure 2–6 shows the timing margin report in the TimeQuest Timing Analyzer window after the report timing script runs. The results are the same as the Quartus II software results.

**Figure 2–6. Timing Margin Report in the TimeQuest Timing Analyzer**






You must verify the timing on every corner of the timing model. You must run the timing report script with all available timing models—slow 0°C, slow 85°C, and fast 0°C—to ensure positive margins across process, voltage, and temperature. To analyze timing on a different corner, first double-click **Set Operating Conditions** in the left pane. Select a new timing corner, then go to the Script menu and rerun the <*variation name*>**_report_timing.tcl** script.

For designs that target Arria II GX devices, you must run a post-processing script to remove timing model pessimism on the write and read capture path margins. For example, refer to Figure 2–7 and Figure 2–8.

**Figure 2–7. Write Margin Summary**

| | Operation | Setup Slack | Hold Slack |
|---|---|---|---|
| 1 | ⊟ Standard Write | 0.249 | 0.246 |
| 2 | Spatial correlation pessimism removal | 0.015 | 0.015 |
| 3 | Write | 0.264 | 0.261 |

**Figure 2–8. Read Capture Path Margin Summary**

| | Operation | Setup Slack | Hold Slack |
|---|---|---|---|
| 1 | ⊟ Standard Read Capture | 0.172 | 0.181 |
| 2 | Spatial correlation pessimism removal | 0.071 | 0.070 |
| 3 | Read Capture | 0.243 | 0.251 |

The standard write and read capture margins are initially calculated using the FPGA timing model and adjusted to account for the effects not modeled by either the timing model or TimeQuest timing analyzer. These effects include memory calibration, deskew topologies, quantization error and calibration uncertainties.

The final write and read capture margins are summarized in the Report DDR margin summary. Figure 2–9 shows an example of a Report DDR margin summary.

**Figure 2–9. Report DDR Margin Summary**

| | Path | Operating Condition | Setup Slack | Hold Slack |
|---|---|---|---|---|
| 1 | Address Command (Slow 900mV 85C Model) | Slow 900mV 85C Model | 0.202 | 0.116 |
| 2 | DQS vs CK (Slow 900mV 85C Model) | Slow 900mV 85C Model | 0.599 | 0.633 |
| 3 | Half Rate Address/Command (Slow 900mV 85C Model) | Slow 900mV 85C Model | 4.964 | 0.958 |
| 4 | Phy (Slow 900mV 85C Model) | Slow 900mV 85C Model | 0.615 | 0.302 |
| 5 | Phy Reset (Slow 900mV 85C Model) | Slow 900mV 85C Model | 0.754 | 0.625 |
| 6 | Read Capture (Slow 900mV 85C Model) | Slow 900mV 85C Model | 0.243 | 0.251 |
| 7 | Read Resync (All Conditions) | All Conditions | 0.748 | 0.835 |
| 8 | Write (Slow 900mV 85C Model) | Slow 900mV 85C Model | 0.264 | 0.261 |

For more information about the TimeQuest Timing Analyzer window, refer to *The Quartus II TimeQuest Timing Analyzer* chapter in volume 7 of the *Quartus II Handbook*. For information about timing analysis, refer to the *Timing Analysis* section in volume 4 of the *External Memory Interface Handbook*.

# Verify Design on a Board

The SignalTap® II Embedded Logic Analyzer shows read and write activity in the system.

For more information about using the SignalTap II Embedded Logic Analyzer, refer to the *Design Debugging Using the SignalTap II Embedded Logic Analyzer* chapter in the *Quartus II Handbook*, *AN 323: Using SignalTap II Embedded Logic Analyzers in SOPC Builder Systems* and *AN 446: Debugging Nios II Systems with the SignalTap II Embedded Logic Analyzer*.

To add the SignalTap II Embedded Logic Analyzer, perform the following steps:

1. On the Tools menu, click **SignalTap II Logic Analyzer**.
2. Under **Signal Configuration**, next to the **Clock** box click **...** (**Browse Node Finder**).
3. In the **Named** box, type `*phy_clk`.
4. For **Filter**, select **SignalTap II: pre-synthesis** and click **List**.
5. In **Nodes Found**, for a DDR2 SDRAM design, select **ddr2_sodimm_example_top | ddr2_sodimm:ddr2_sodimm_inst | ddr2_sodimm_controller_phy:ddr2_sodimm_controller_phy_inst | phy_clk | phy_clk**, or for a DDR3 SDRAM design, select **ddr3_example_top | ddr3:ddr3_inst | ddr3_controller_phy:ddr3_controller_phy_inst | phy_clk | phy_clk**, and click **>** to add the signal to **Selected Nodes**.
6. Click **OK**.
7. Under **Signal Configuration**, specify the following settings:
   - For **Sample depth**, select **512**
   - For **RAM type**, select **Auto**
   - For **Trigger flow control**, select **Sequential**
   - For **Trigger position**, select **Center trigger position**
   - For **Trigger conditions**, select **1**
8. On the Edit menu, click **Add Nodes**.
9. In the **Named** box, search for specific nodes by typing `*local*`.
10. For **Filter**, select **SignalTap II: pre-synthesis** and click **List**.

11. Select the following nodes in **Nodes Found** and click **>** to add to **Selected Nodes**:
    - **local_address**
    - **local_rdata**
    - **local_rdata_valid**
    - **local_read_req**
    - **local_ready**
    - **local_wdata**
    - **local_wdata_req**
    - **local_write_req**
    - **pnf**
    - **pnf_per_byte**
    - **test_complete** (trigger)

    Do not add any DDR2 SDRAM or DDR3 SDRAM interface signals to the SignalTap II Embedded Logic Analyzer. The load on these signals increases and adversely affects the timing analysis.

12. Click **OK**.
13. To reduce the SignalTap II logic size, turn off **Trigger Enable** on the following bus signals:
    - **local_address**
    - **local_rdata**
    - **local_wdata**
    - **pnf_per_byte**
14. Right-click **Trigger Conditions** for the `test_complete` signal and select **Rising Edge**.
15. On the File menu, click **Save**.

    If you see the message **Do you want to enable SignalTap II file "stp1.stp" for the current project**, click **Yes**.

## Compile the Project

After you add signals to the SignalTap II Embedded Logic Analyzer, to recompile your design, on the Processing menu, click **Start Compilation**.

## Verify Timing

After the design compiles, ensure that TimeQuest timing analysis passes successfully. In addition to this FPGA timing analysis, check your PCB or system SDRAM timing. To perform timing analysis, run the *<variation name>*_**phy_report_timing.tcl** script.

1. On the Tools menu, click **Tcl Scripts**.

2. Select <*variation name*>**_phy_report_timing.tcl**.
3. Click **Run**.

## Download the Object File

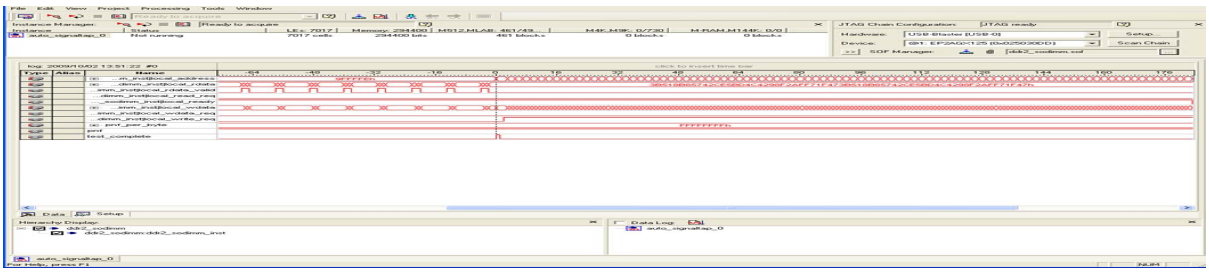
To download the object file to the device, perform the following steps:

1. Connect the development board to your computer.
2. On the Tools menu, click **SignalTap II Logic Analyzer**. The SignalTap II dialog box appears.
3. Click **...** to open the **Select Program Files** dialog box.
4. Select <*your project name*>**.sof**.
5. Click **Open**.
6. To download the file, click the **Program Device** button.

## Test the Design Example in Hardware

When the design example including SignalTap II Embedded Logic Analyzer successfully downloads to your development board, click **Run Analysis** to run once, or click **Autorun Analysis** to run continuously. Figure 2–10 shows the design analysis.

**Figure 2–10. SignalTap II Example DDR2 SDRAM Design Analysis**



# 3. Using DDR and DDR2 SDRAM Devices in Cyclone III and Cyclone IV Devices

This tutorial shows how to use the design flow described in the preceding sections to design a 32-bit wide 167-MHz DDR2 SDRAM memory interface targeted for the Cyclone® III FPGA development kit. The design example also provides some recommended settings, including termination scheme and drive strength settings, to simplify your design flow. Although the design example is specifically for the DDR2 SDRAM memory interface and Cyclone III devices, the design flow is identical to that for a DDR SDRAM memory interface or using Cyclone IV devices.

This design example targets a memory interface frequency of 167 MHz because the targeted development kit uses an EP3C120F780 device. This device is available in –7 and –8 speed grades only. You can achieve a higher clock rate, up to 200 MHz, for DDR2 SDRAM if you select a –6 speed grade device from the Cyclone III family. This design example uses the DDR2 SDRAM Controller with ALTMEMPHY IP and is compiled in the Quartus II software version 9.0.

To download the design example, **emi_ddr2_ciii.zip**, go to the External Memory Interface Design Examples page.

## Select the Device

Cyclone III and Cyclone IV devices support various data widths for DDR2 and DDR SDRAM memory interfaces. This walkthrough uses the Cyclone III EP3C120F780C7 device with 32-bit wide DDR2 SDRAM interface up to 167 MHz on the bottom I/O banks. The 32-bit wide interface uses four ×8 groups. For the DDR2 SDRAM memory device, select Micron's 512-MB MT47H32M16CC-3 333-MHz DDR2 SDRAM device, because it is the device used on the development board.

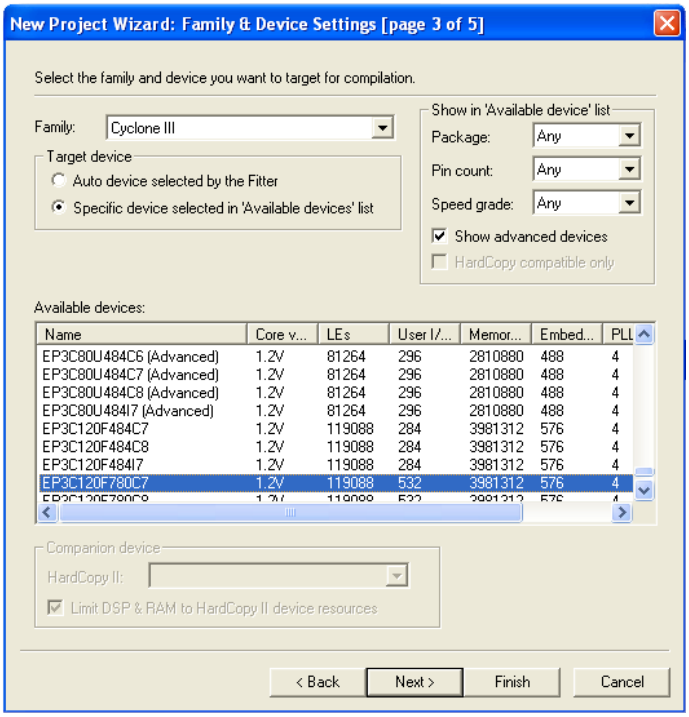
Top/bottom DQ groups provide the fastest performance. The Quartus II software automatically uses the top/bottom DQ groups if they are available. Combining top/bottom DQ groups with left/right DQ groups for a single interface is not recommended and may result in a degraded performance.‘

For more information about the DQ/DQS bus groups for different densities, packages, and sides of the Cyclone III or Cyclone IV device, refer to the *External Memory Interfaces in the Cyclone III Device Family* chapter in volume 1 of the *Cyclone III Device Handbook*, or *External Memory Interfaces in the Cyclone IV Devices* chapter in volume 1 of the *Cyclone IV Device Handbook*.

# Instantiate PHY and Controller in a Quartus II Project

Create a project in the Quartus II software targeting the EP3C120F780C7 device. Figure 3–1 shows how to target this device in the **New Project Wizard**.

**Figure 3–1. Creating a Quartus II Project Targeting the EP3C120F780C7 Device**



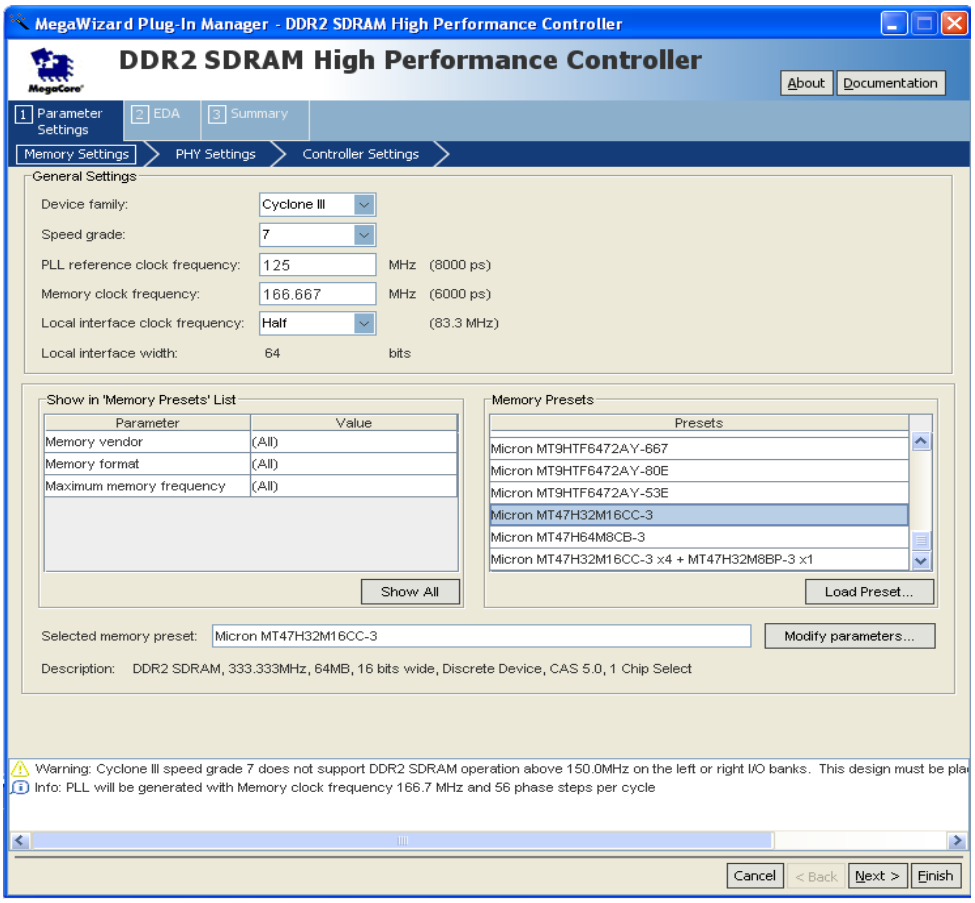

For detailed step-by-step instructions about how to create a Quartus II project, refer to the Quartus II software tutorial by clicking the Help menu in the Quartus II window and selecting **PDF Tutorials**.

After creating a Quartus II project, instantiate the DDR2 SDRAM Controller with ALTMEMPHY IP. This example uses the DDR2 SDRAM high-performance controller, which instantiates the ALTMEMPHY megafunction automatically. Select the **DDR2 SDRAM Controller with ALTMEMPHY IP** in the **Interfaces** section of the MegaWizard™ Plug-In Manager. Name the controller `DDR2`.

The subsequent files mentioned in this document that are generated by the MegaWizard Plug-In Manager and also other project files will have **DDR2** as the prefix for the file names.

The rest of this subsection specifies the memory settings. Select the Cyclone III device with –7 speed grade. Then set the PLL reference clock frequency to 125 MHz and the memory clock frequency to 166.667 MHz. The 125 MHz PLL reference clock is provided by the on-board oscillator. Select the Micron **MT47H32M16CC-3** 333-MHz device. This 512-MB DDR2 device has a 16-bit data width. Figure 3–2 shows the memory settings panel after you make the selections.

**Figure 3–2. Configuring the DDR2 SDRAM Controller with ALTMEMPHY IP for the DDR2 Memory Interface**



The DDR2 SDRAM controller MegaWizard interface uses the default parameters for the memory when you instantiate the controller. The default parameters are based on the memory datasheets. You can customize your memory presets by modifying the parameters. Click the **Modify parameters** button to modify the memory attributes, memory initialization options, or memory timing parameters in the Preset Editor dialog box. In this design example, modify the memory interface DQ width to 32 to match the targeted memory width on the development kit. Figure 3–3 shows the Preset Editor dialog box after you perform the customization.

**Figure 3–3. Editing the Memory Presets to Create a Custom Memory**

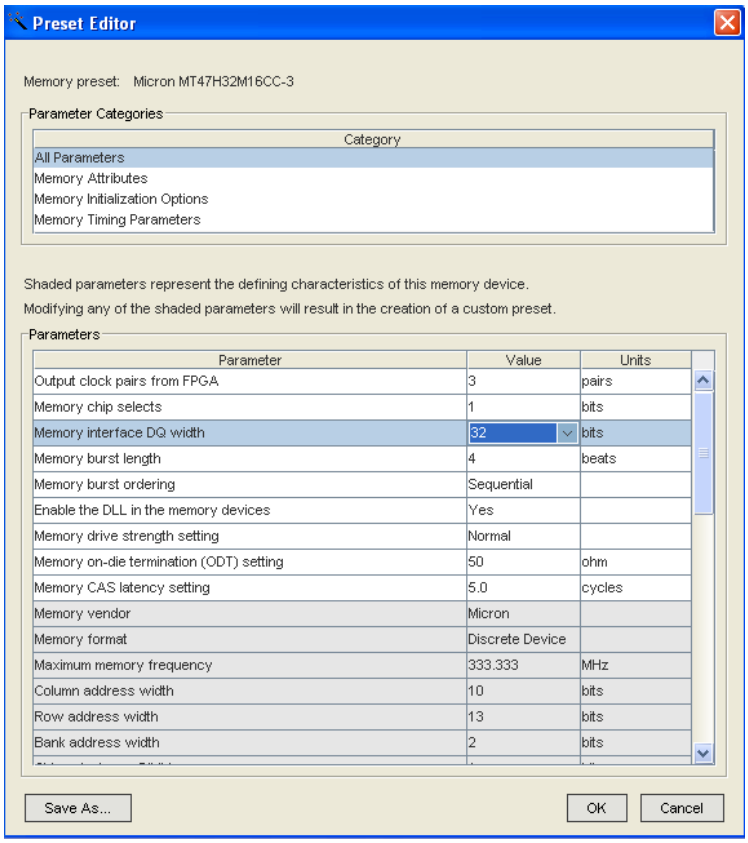
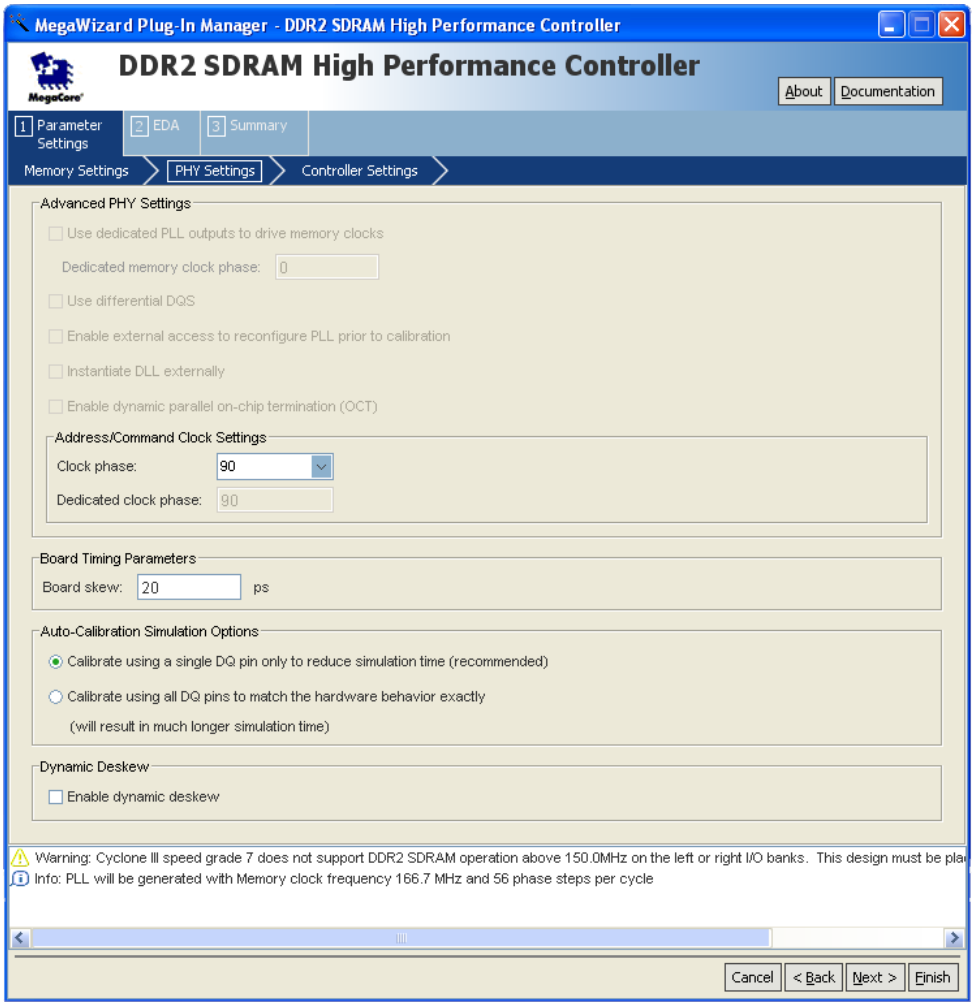



Figure 3–4 shows the PHY Settings tab. In the PHY Settings tab, under **Board Timing Parameters**, you must parameterize the **Board Skew** settings. The specified skew is across all memory interface signal types including data, strobe, clock, address and command, and is used to generate the PHY timing constraints for all paths. The default value is set to 20 ps. You must update this number based on your board specification, because this number is used to calculate the overall system timing margin.

The DDR2 SDRAM device uses the `CK` and `CK#` signals to clock the command and address signals into the memory. The controller names the `CK` and `CK#` signals `mem_clk` and `mem_clk_n`, respectively. The skew between the `CK` or `CK#` and the DDR2 SDRAM-generated `DQS` signal is the value of $t_{DQSCK}$ in the DDR2 SDRAM data sheet.

The DDR2 SDRAM has a write requirement ($t_{DQSS}$) that the positive edge of the DQS signal on writes be within ± 25% (± 90°) of the positive edge of the DDR2 SDRAM clock input. $t_{DQSS}$ is defined as the time between the DQS latching edge to its associated clock edge. The controller generates the mem_clk and mem_clk_n signals using the DDR registers in the input/output element (IOE) to match the DQS signal and reduce any variations across process, voltage, and temperature. The positive edge of the DDR2 SDRAM clock, mem_clk, is aligned with the DQS write to satisfy $t_{DQSS}$.

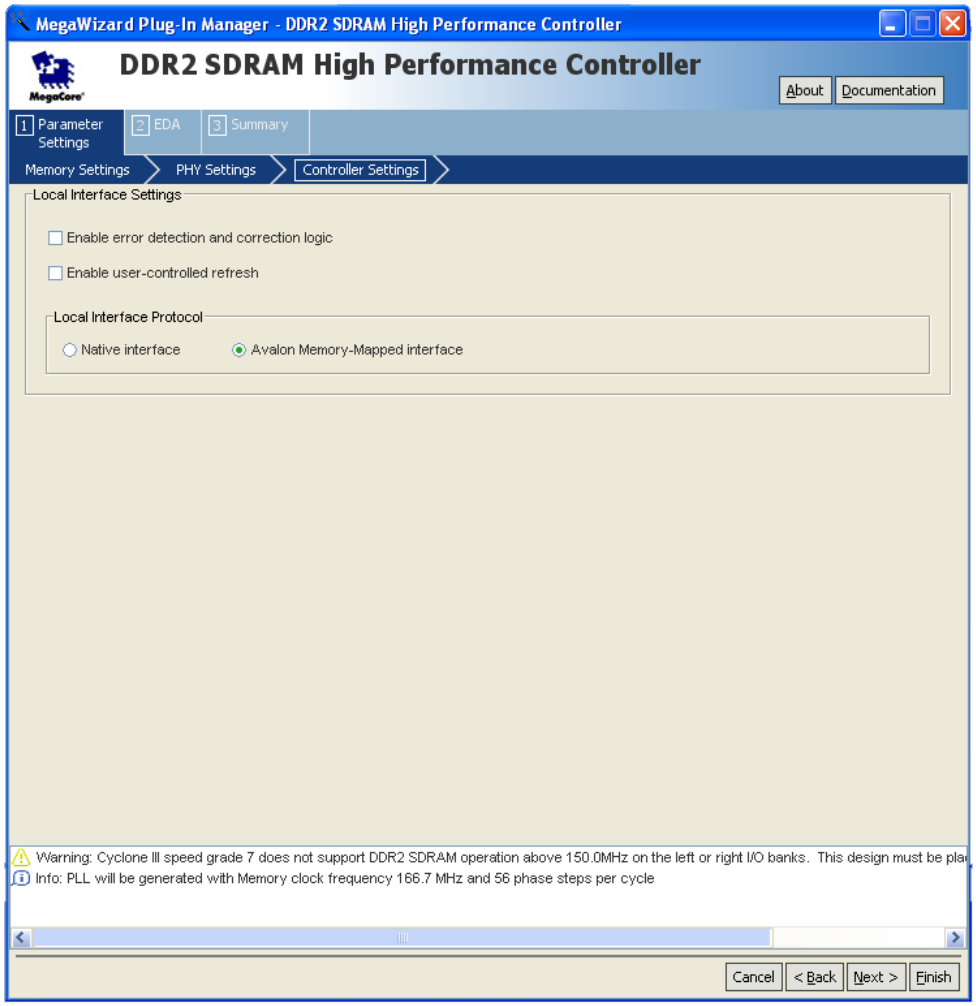
**Figure 3–4. DDR2 SDRAM Controller with ALTMEMPHY IP PHY Settings**




The settings in the **Auto-Calibration Simulation Options** section are for RTL simulation only and are not applicable for gate-level simulation.

Figure 3–5 shows the **Controller Settings** panel.

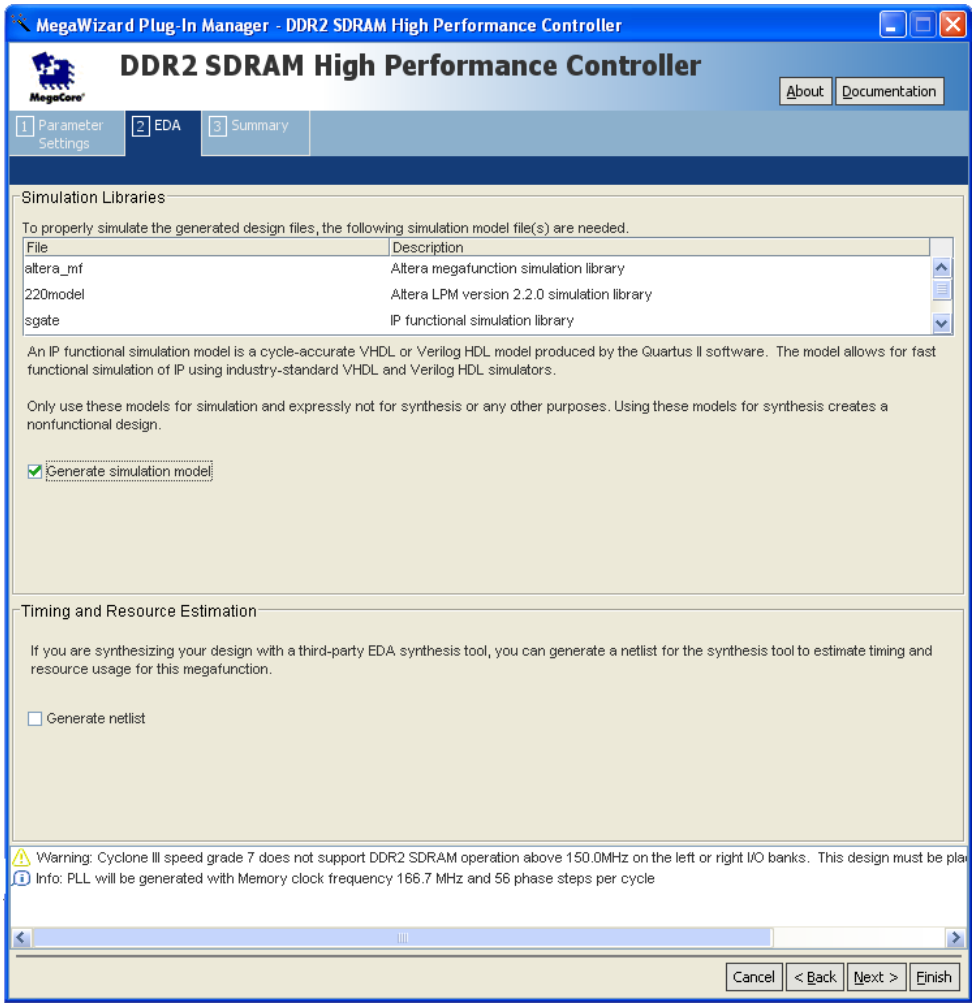
**Figure 3–5. DDR2 SDRAM Controller with ALTMEMPHY IP Settings**




Choose your local interface setting in the **Controller Settings** panel. Turn on the **Enable error detection and correction logic** option if you want to use error code correction (ECC). If you have your own refresh requirements, turn on **Enable user-controlled refresh**. Next, select the **Local Interface Protocol** for the memory interface. The default interface is the **Avalon Memory-Mapped Interface**. This interface allows you to easily connect to other Avalon-MM peripherals.

Figure 3–6 on page 3–7 shows the **EDA** panel. The MegaWizard can generate the simulation model for simulating the memory controller in either Verilog HDL or VHDL.

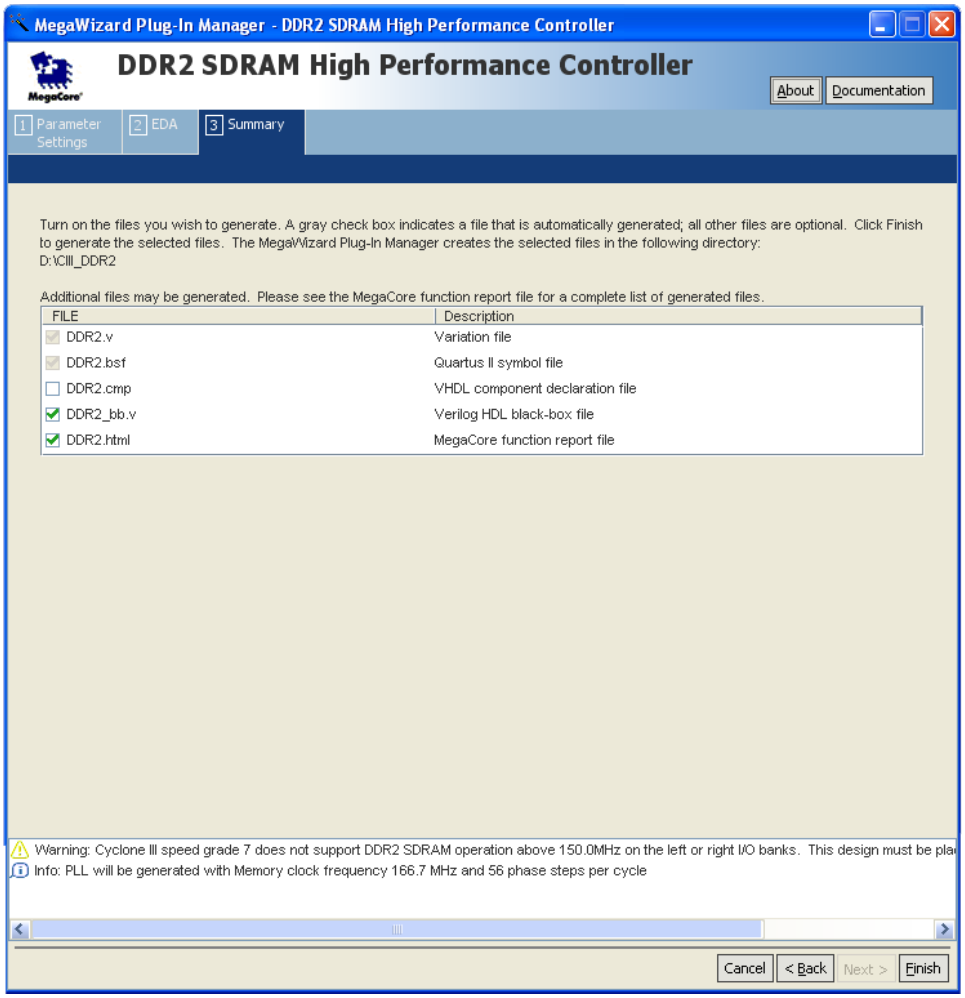
**Figure 3–6. DDR2 SDRAM Controller with ALTMEMPHY IP EDA**




In the **Timing and Resource Estimation**, you can choose to generate a netlist if you are synthesizing your design with a third-party EDA synthesis tool.

Figure 3–7 shows the **Summary** panel.

**Figure 3–7. Summary**



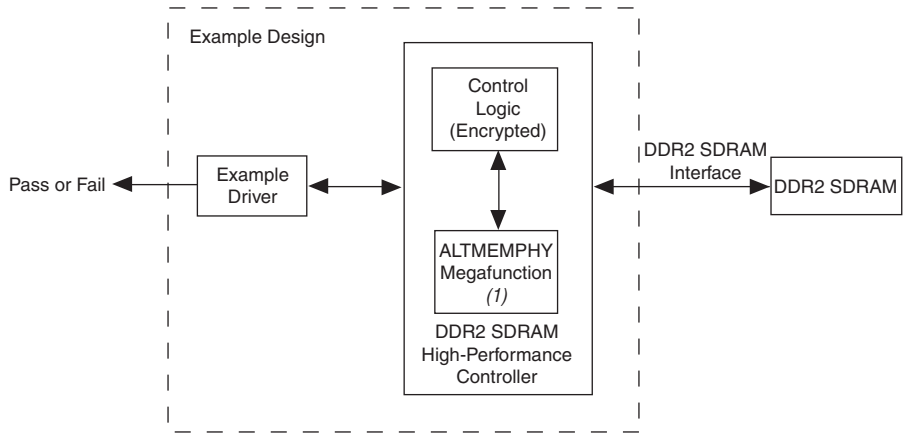

For detailed step-by-step instructions about configuring the DDR2 SDRAM Controller with ALTMEMPHY IP, refer to the *DDR and DDR2 SDRAM Controller with ALTMEMPHY IP User Guide* section in volume 3 of the *External Memory Interface Handbook*.

Click **Finish** to generate the controller.

# Perform RTL or Functional Simulation (Optional)

After instantiating the DDR2 SDRAM Controller with ALTMEMPHY IP, the MegaWizard Plug-In Manager generates a design example and driver for testing the memory interface. Figure 3–8 shows a system-level diagram of the design example that the DDR2 SDRAM Controller with ALTMEMPHY IP MegaWizard creates for you.

**Figure 3–8. DDR2 SDRAM Controller with ALTMEMPHY IP System-Level Diagram**




**Note to Figure 3–8:**

(1) The ALTMEMPHY megafunction automatically generates the PLL. The PLL is part of the ALTMEMPHY megafunction.

For more information about the different files generated with the DDR2 SDRAM Controller with ALTMEMPHY IP, refer to the *DDR and DDR2 SDRAM Controller with ALTMEMPHY IP User Guide* section in volume 3 of the *External Memory Interface Handbook*.

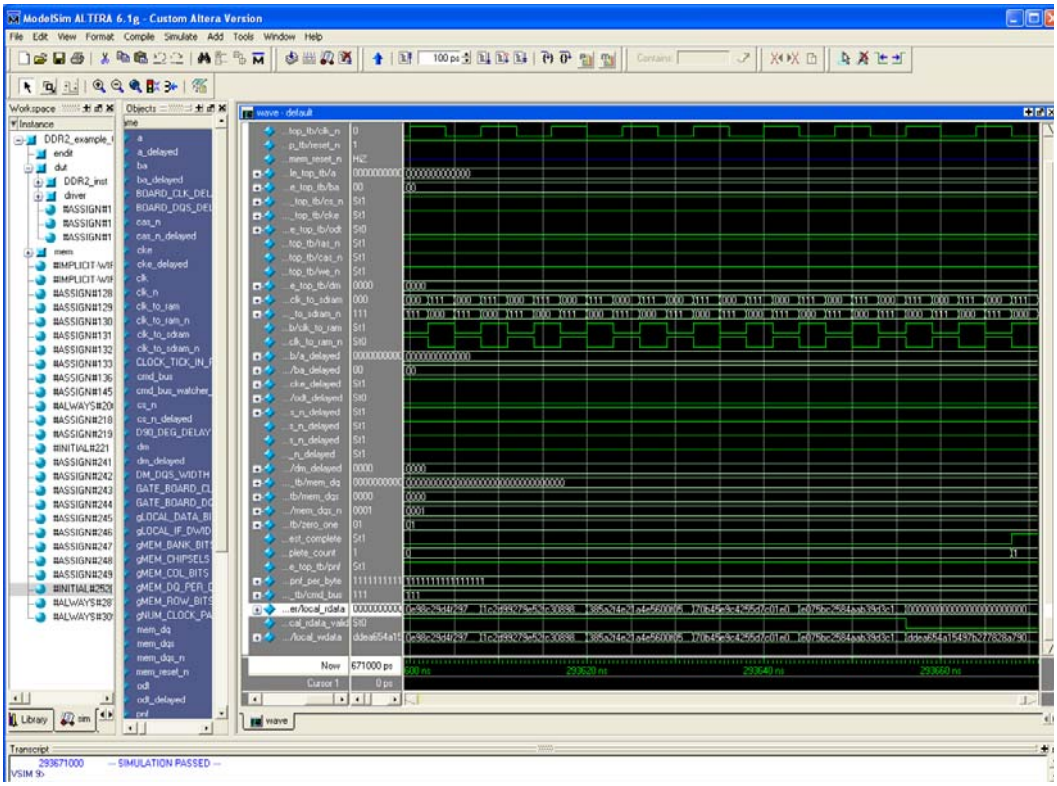
You can simulate the memory interface with the MegaWizard Plug-In Manager-generated IP functional simulation model. You should use this model in conjunction with your own driver or the testbench generated by the MegaWizard Plug-In Manager that issues read and write operations. The memory model file is also automatically generated by the MegaWizard Plug-In Manager in the testbench.

Use the functional simulation model with any Altera-supported VHDL or Verilog HDL simulator. This walkthrough uses the Quartus II NativeLink feature to run the Altera-ModelSim software to perform the simulation.

For more information about how to set up the simulation in Quartus II software using NativeLink, refer to the *Simulation Walkthrough* chapter in volume 4 of the *External Memory Interface Handbook*.

Figure 3–9 shows an Altera-ModelSim RTL simulation.

**Figure 3–9. Altera-ModelSim RTL/Functional Simulation**




# Add Constraints

When you create your memory controller, the MegaWizard Plug-In Manager generates the following constraint files for timing constraint and pin assignment.

- *DDR2_phy_ddr_timing.sdc*
- *DDR2_pin_assignments.tcl*

The DDR2_phy_ddr_timing.sdc file is used to constrain the clock and input/output delays in the ALTMEMPHY megafunction. Enable the TimeQuest timing analyzer before compiling your design. To enable the TimeQuest timing analyzer, perform the following steps:

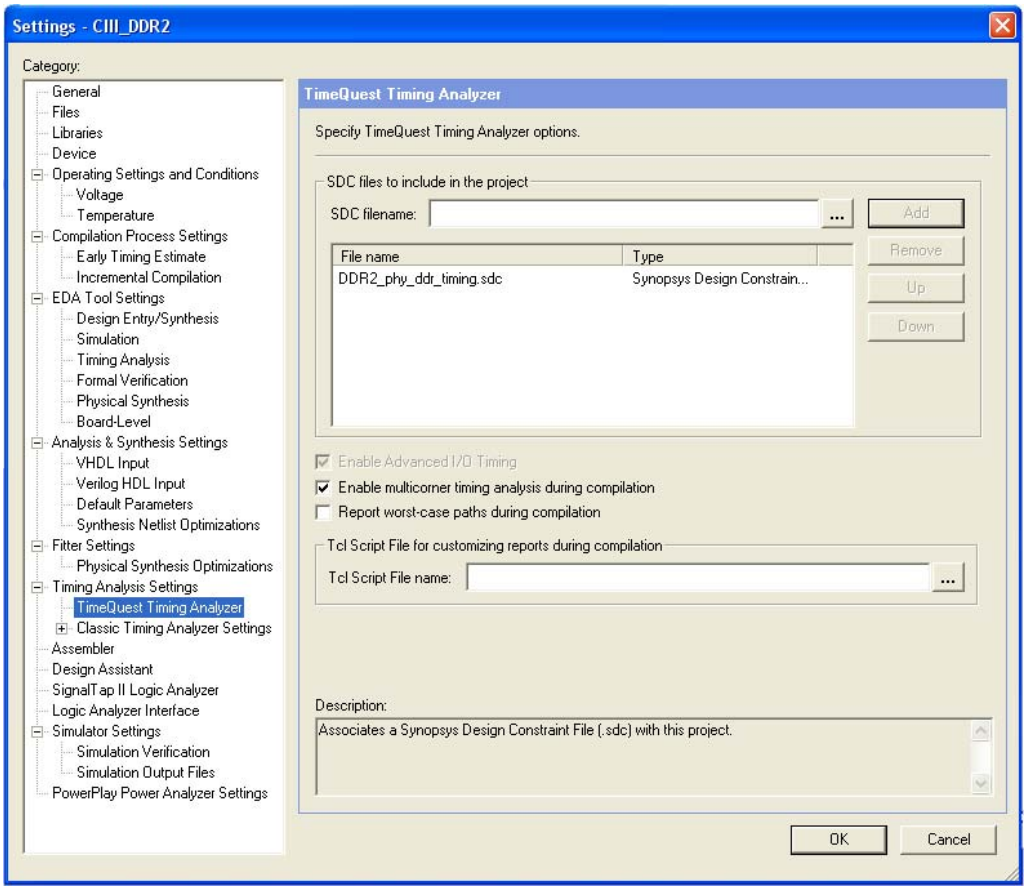
1. On the Assignments menu, click **Settings**. The **Settings** dialog box appears.
2. From the **Category** list, click **Timing Analysis Settings** and select **Use TimeQuest Timing Analyzer during compilation**.
3. Click **OK**.

Next, to add the timing constraints, perform the following steps:

1. On the **Settings** dialog box, click **Timing Analysis Settings** and select **TimeQuest Timing Analyzer**. The **TimeQuest Timing Analyzer** page appears.
2. Specify the SDC file and click **Add** (Figure 3–10).

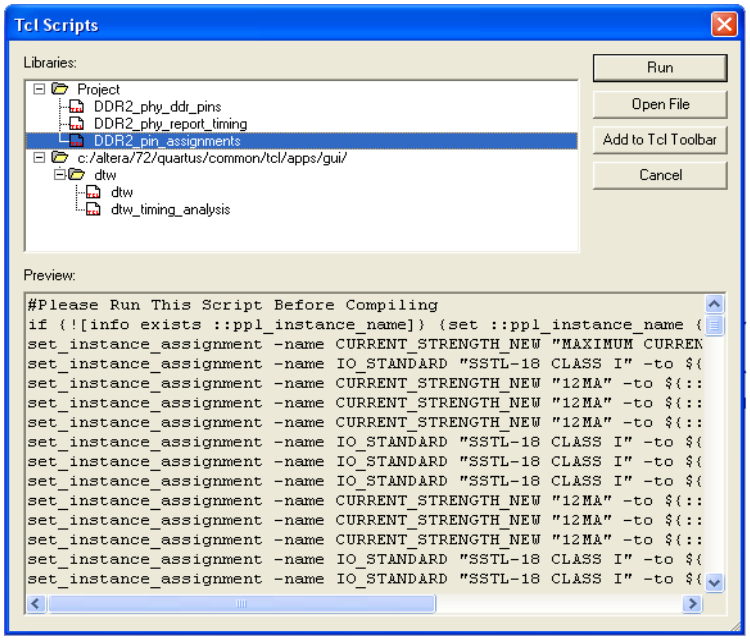
3. Click **OK**.

**Figure 3–10. Specifying the Timing Constraint SDC File for the Design Example**



The **DDR2_pin_assignments.tcl** file specifies the I/O standard assignment for the memory interface pins. To run the Tcl file, on the **Tools** menu, click **Tcl Scripts**. Select the Tcl file and click **Run** (Figure 3–11 on page 3–12). Running the script adds the information into your Quartus II project. Alternatively, you can also use the Tcl Console to run the Tcl file.

**Figure 3–11. Running Pin Assignment Constraint Script in the Tcl Script Panel**




After running the **DDR2_pin_assignments.tcl** file, you can assign the I/O locations based on your board design in the **Assignment Editor** or **Pin Planner**. In this design example, assign the I/O locations based on the Cyclone III FPGA development kit board. The 32-bit interface is located at the bottom I/O banks of the device.

# Compile Design and Verify Timing Closure

Before you compile the design, set the top level entity of the project to the design example created by the high-performance controller in the previous section, by performing the following steps:

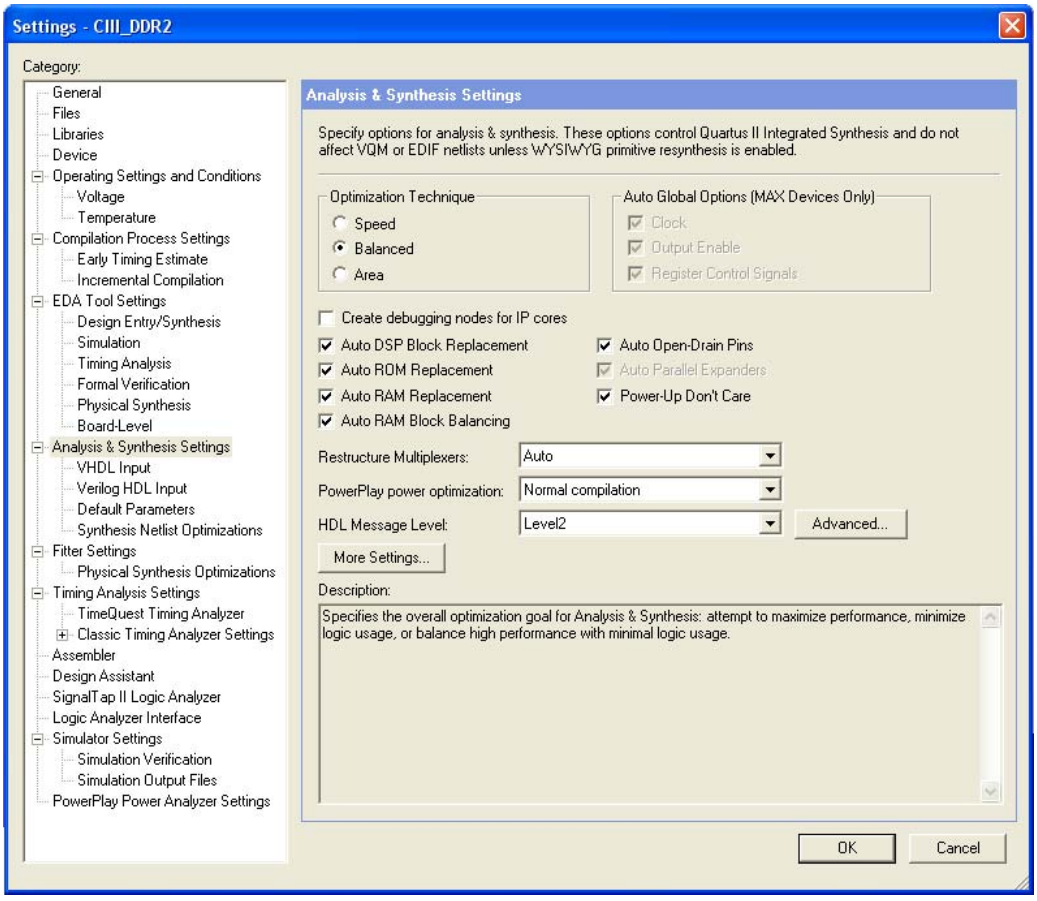
1. On the File menu, click **Open**.
2. Browse to *<variation name>*_**example_top** and click **Open**.
3. On the Project menu, click **Set as Top-Level Entity**.

After you specify that the design example is the top level entity, set the Quartus II software to ensure the remaining unconstrained paths are routed with the highest speed and efficiency. To do so, perform the following steps:

1. On the Assignments menu, click **Settings**. The **Settings** dialog box appears.
2. In the **Category** list, click **Analysis & Synthesis Settings**.

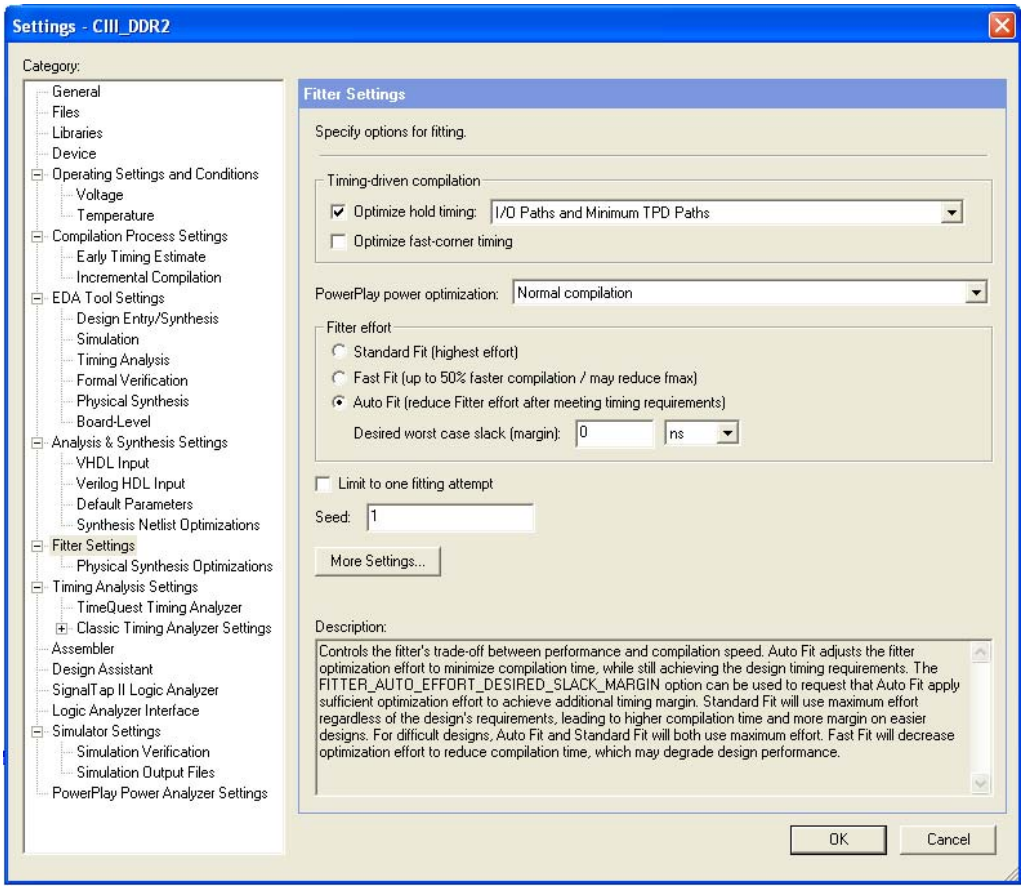
3. Define the **Optimization Technique** setting. The default setting is **Balanced** (Figure 3–12 on page 3–13).

**Figure 3–12. Default Setting for Optimization Technique**



4. In the **Category** list, click **Fitter Settings**, and set the **Fitter effort**. The default setting is **Auto Fit** (Figure 3–13 on page 3–14).

**Figure 3–13. Default Setting for Fitter Effort**




To compile your design, perform the following steps:

1. On the Processing menu, click **Start Compilation**.
2. After compilation is successful, on the Tools menu, select **TimeQuest Timing Analyzer**.
3. Generate the timing margin report for your memory interface design by executing the **Report DDR** function from the **Tasks** pane of the TimeQuest Timing Analyzer window (Figure 3–14).

Executing the **Report DDR** task automatically runs the *<variation_name>*_**phy_report_timing.tcl** timing margin report script generated by the MegaWizard Plug-In Manager when the megafunction variation was created.

For more information about the TimeQuest timing analyzer, refer to the *Quartus II TimeQuest Timing Analyzer* chapter in volume 3 of the *Quartus II Handbook*.

**Figure 3–14. TimeQuest Timing Analyzer Window**

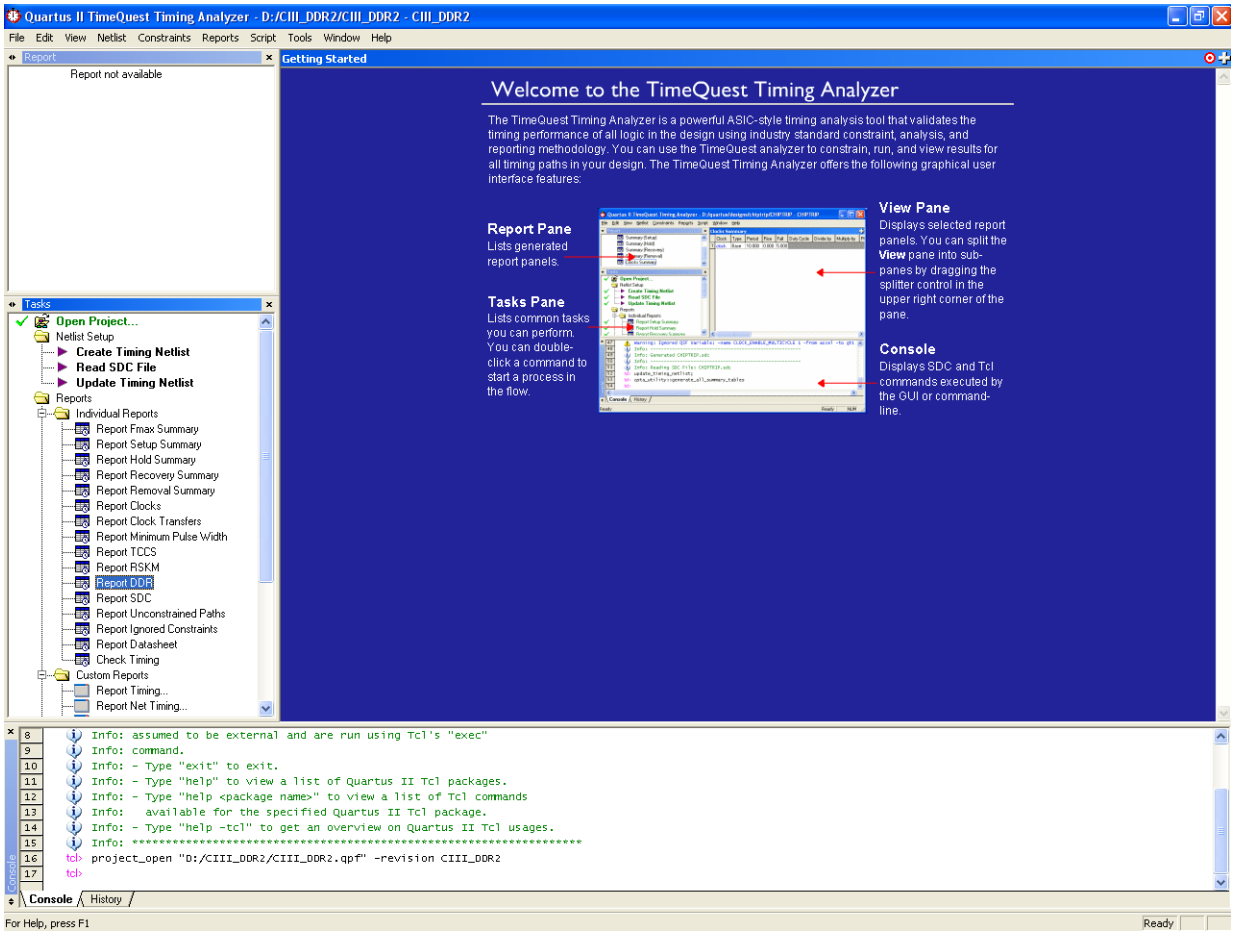
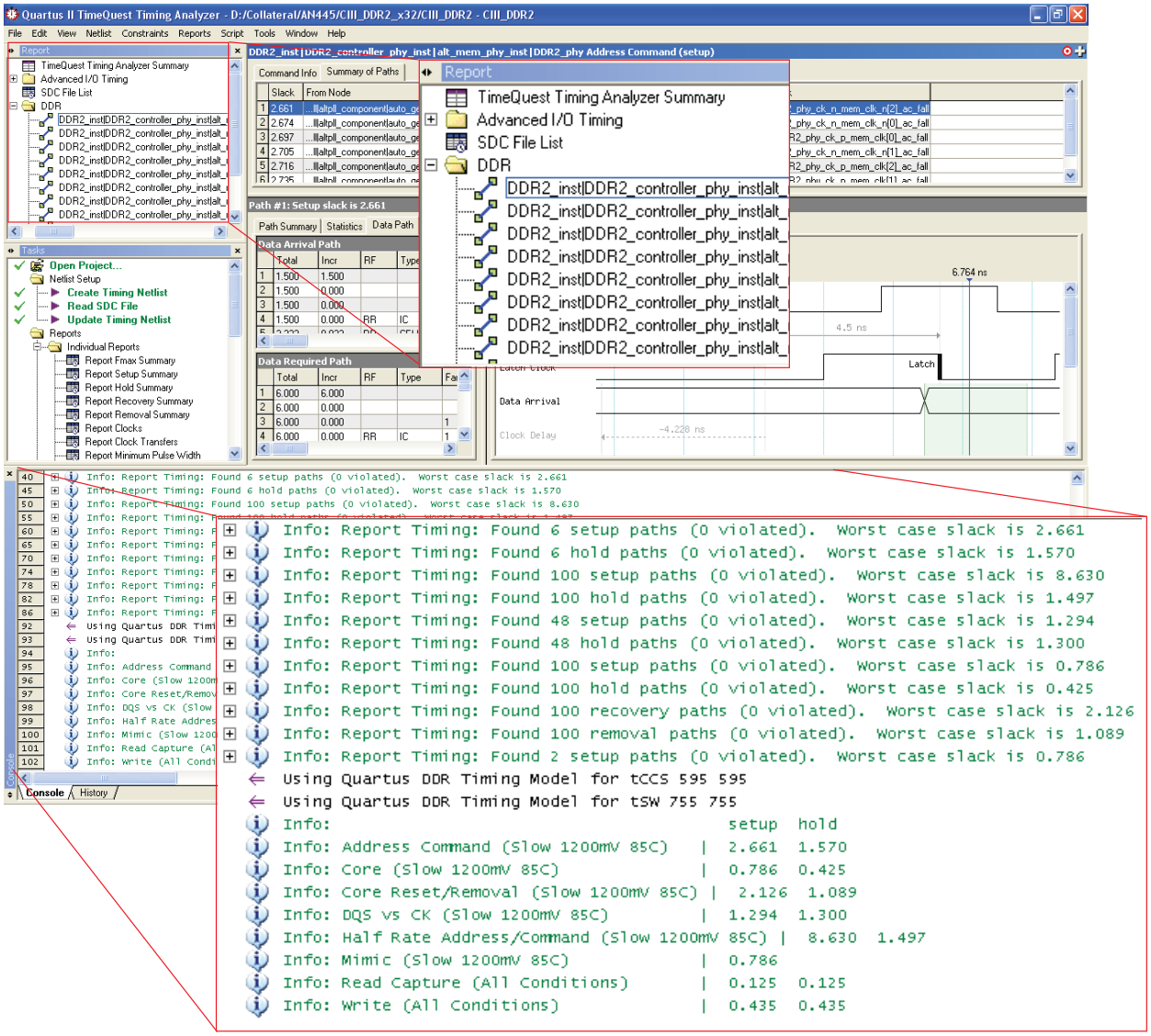


Figure 3–15 shows the output of the Report DDR task. The Report pane contains a new folder titled **DDR** with detailed timing information about the most critical paths, and a timing margin summary similar to the summary on the TimeQuest Console.

**Figure 3–15. DDR2 Report Panel and Timing Margin in TimeQuest Timing Analyzer**




The report timing script provides information about the following margins and paths:

- Address/command setup and hold margin
- Half-rate address/command setup and hold margin
- Core setup and hold margin
- Core reset/removal setup and hold margin
- DQS vs CK setup and hold margin
- Mimic path

- Write setup and hold margin
- Read capture setup and hold margin

For more information about the timing analysis, refer to the *Timing Analysis* section in volume 4 of the *External Memory Interface Handbook*.

# Adjust Constraints

Even if the MegaWizard Plug-In Manager generates the controller with the correct settings and you do not see any timing violation, you can manually adjust some of the constraints to improve the timing for your system. The timing margin report shows the address and command datapath with a 2.668 ns setup and 1.601 ns hold margin. Adjusting the clock that is regulating the address and command output registers can balance the setup and hold time better by decreasing the setup margin and increasing

the hold margin on the address and command datapath. To determine the clock that is clocking the address and command registers, in TimeQuest timing analyzer, on the Report panel, click on the address/command report, and select the path for the setup or hold time for the address and command datapath as Path Summary (Figure 3–16) or in Waveform View (Figure 3–17).

**Figure 3–16. Path Summary for the Address and Command Datapath**

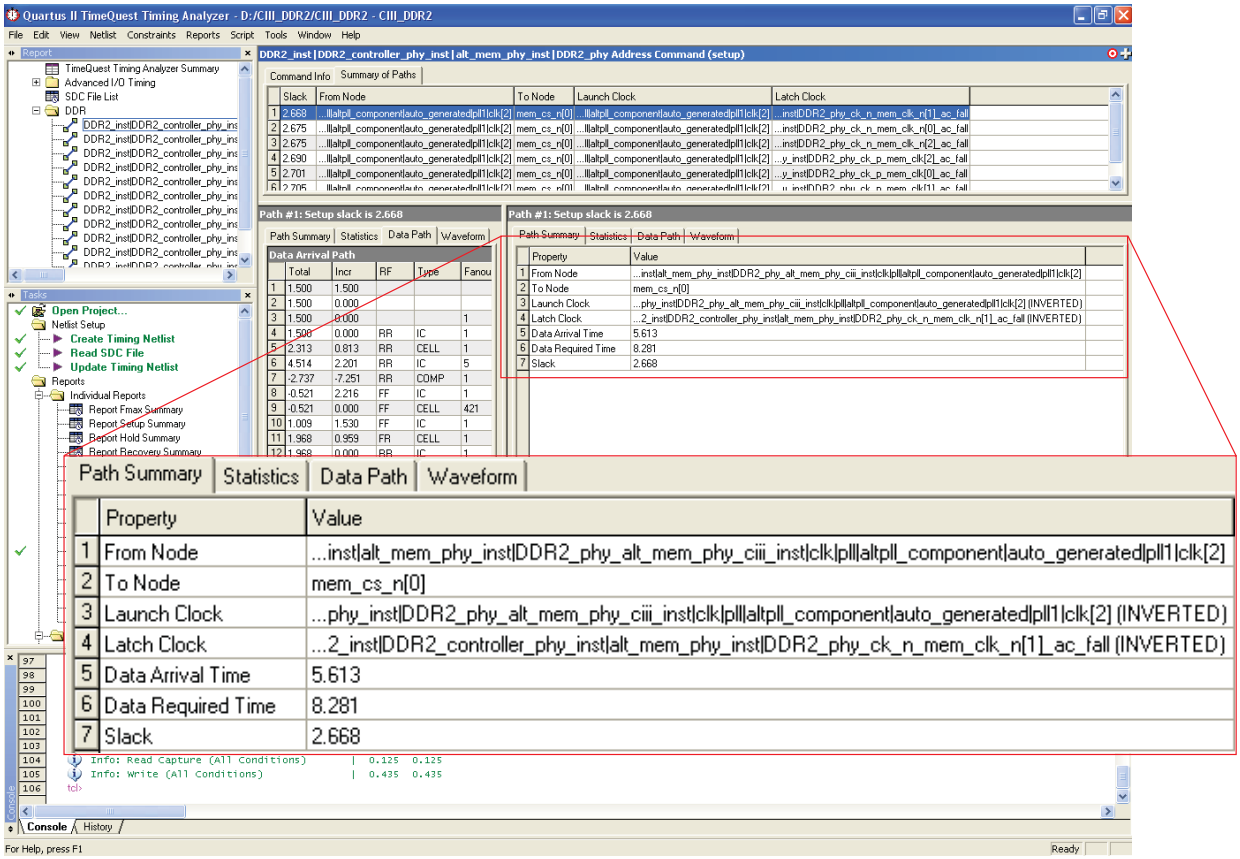
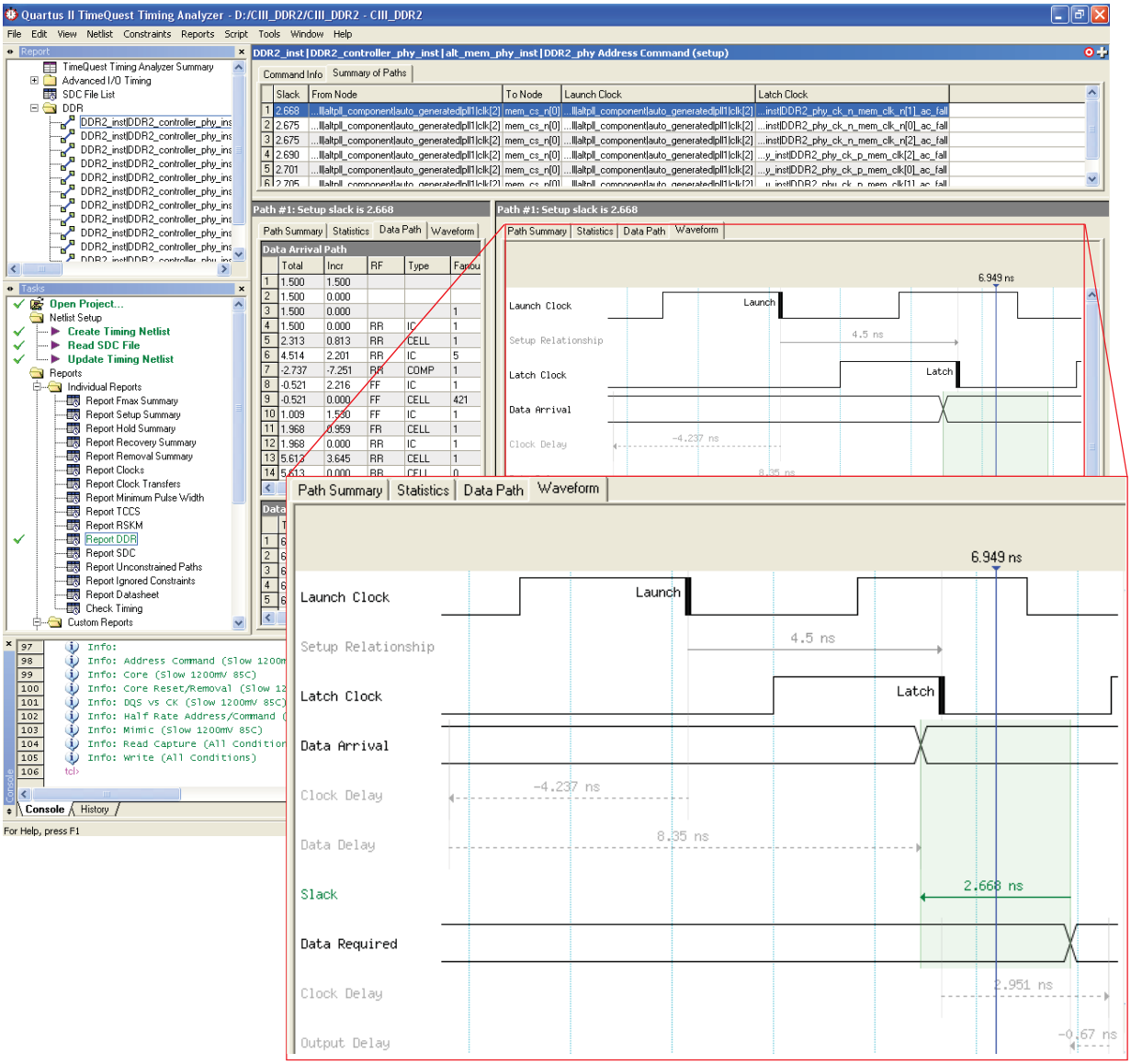


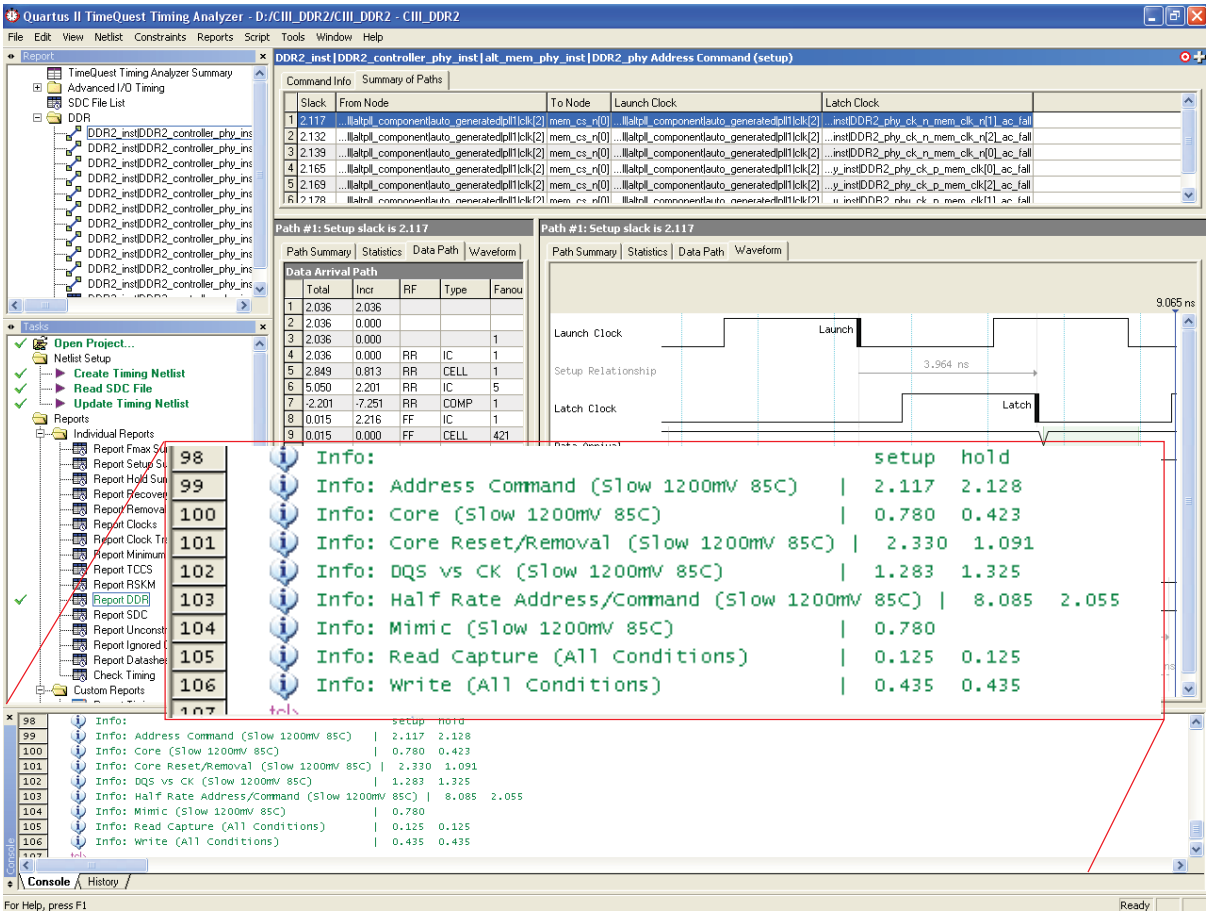
**Figure 3–17. Waveform View for the Address and Command Datapath**




The report indicates that `clk2` of the PLL is clocking the address and command registers. Edit PLL megafunction to change the phase setting of clk2. For this design, the initial phase setting of `clk2` is –90° with reference to the system clock. By adjusting `clk2` to later than –90° (specifying that the address and command launch later), the setup margin is decreased and the hold margin is increased.

Modify the clk2 phase setting to –55°, and recompile the design for the new PLL setting to take effect. Run the report timing script again. Figure 3–18 shows the timing margin reported in the Quartus II software after adjusting the phase setting of clk2. The address and command datapath now has a more balanced 2.117 ns setup and 2.128 ns hold margin.

**Figure 3–18. Timing Margin Reported After Adjusting clk2**




# Determine Board Design Constraints

Both Cyclone III and Cyclone IV FPGA support the series on-chip termination (OCT) to improve signal integrity and simplify the board design. The Cyclone III and Cyclone IV devices support OCT with or without calibration. In addition, you can choose the resistance value 25 Ω or 50 Ω depending on the I/O standards you use for the memory interface and the termination scheme.

The DDR2 SDRAM supports dynamic parallel on-die termination (ODT). This feature allows you to turn on ODT when the FPGA is writing to the DDR2 SDRAM memory and turn off ODT when the FPGA is reading from the DDR2 SDRAM memory. The ODT features are available in settings of 150 Ω, 75 Ω, and 50 Ω. The 50-Ω setting is only available in DDR2 SDRAM with operating frequencies greater than 267 MHz.

Refer to the respective memory data sheet for additional information about the available settings for the ODT and the output driver impedance features, and the timing requirements for driving the ODT pin in DDR2 SDRAM.

For this design walkthrough, which targets the Cyclone III FPGA development kit, drive strength setting is used together with Class I termination. Using Class I termination allows a higher maximum DDR2 SDRAM clock frequency and reduces the number of external resistors required on the board. You can adjust the current strength of the output pin of the Cyclone III and Cyclone IV devices according to your board setup. If the current strength is too high, you might see excessive overshoot and undershoot of your signal. Use the oscilloscope to check the signal overshoot and undershoot.

Figure 3–19 shows the setup for write operation to the DDR2 SDRAM memory with Class I termination together with the drive strength setting of the Cyclone III or Cyclone IV device. In this setup, the driver's (FPGA) output impedance matches that of the transmission line, resulting in optimal signal transmission to the DDR2 SDRAM memory. On the receiver (DDR2 SDRAM memory) side, the receiving pin is properly terminated with matching impedance to the transmission line, through the external pull-up resistor to $V_{TT}$, to eliminate any ringing or reflection.

**Figure 3–19. Write Operation to DDR2 SDRAM Memory with Class I Termination**

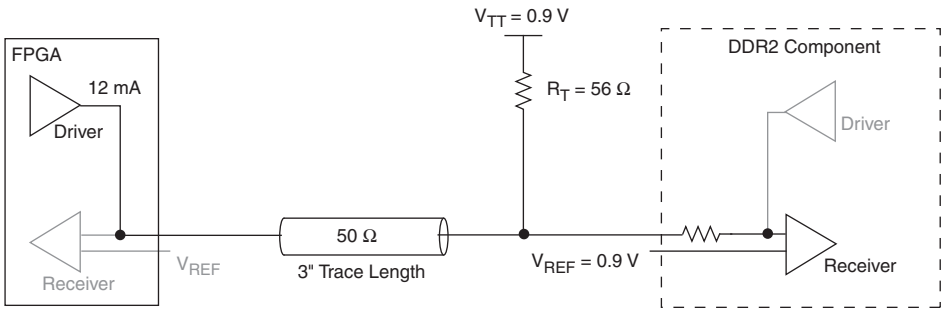
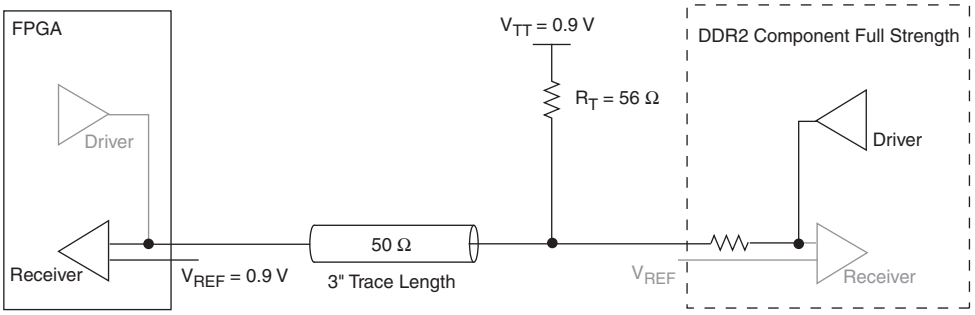



Figure 3–20 shows the setup for read operation from the DDR2 SDRAM memory.

**Figure 3–20. Read Operation from DDR2 SDRAM Memory with Class I Termination**




If you choose not to use the drive strength setting, you can use the OCT feature of the Cyclone III or Cyclone IV device instead.

Finally, the loading seen by the FPGA during writes to the memory is different in a system using dual-inline memory modules (DIMMs) and a system using components. The additional loading from the DIMM connector can reduce the edge rates of the signals arriving at the memory, thus affecting available timing margin.

For more information about Cyclone III and Cyclone IV OCT, refer to the *I/O Features in the Cyclone III Device Family* chapter in volume 1 of the *Cyclone III Device Handbook*, and the *I/O Features in Cyclone IV Devices* chapter in volume 1 of the *Cyclone IV Device Handbook*.

For detailed information about different effects on signal integrity design, refer to *Board Layout Guidelines* section in volume 2 of the *External Memory Interface Handbook*.

# 4. Using DDR and DDR2 SDRAM Devices in Stratix III and Stratix IV Devices

This tutorial describes how to use the design flow to design a 72-bit wide, 400-MHz, 800-Mbps DDR2 SDRAM interface with Stratix® III devices. This design example also provides some recommended settings, including termination scheme and drive strength setting, to simplify the design. Although the design example is specifically for the DDR2 SDRAM interface, the design flow for a DDR SDRAM interface is the same.

The design example targets the Stratix III FPGA development kit, which includes a 72-bit wide 1-GB Micron MT9HTF12872AY-800E 400-MHz DDR2 SDRAM DIMM.

To download the design example, **emi_ddr2_siii.zip**, go to the External Memory Interface Design Examples page. You can also use this design example if you are targeting a HardCopy® device for your design. For ALTMEMPHY megafunction supported devices, refer to mmmm section in volume 3 of the *External Memory Interface Handbook*.

For more information about design flow, refer to the *Recommended Design Flow* section in volume 1 of the *External Memory Interface Handbook*.

## Software Requirements

This tutorial assumes that you have experience with the Quartus II software. This tutorial requires the following hardware and software:

- Quartus II software version 9.0
- DDR and DDR2 SDRAM Controllers with ALTMEMPHY IP version 9.0
- Stratix III FPGA development kit

## Create a Quartus II Project

Create a project in the Quartus II software that targets the EP3SL150F1152-C2 device.

For detailed step-by-step instructions about how to create a Quartus II project, refer to the Quartus II software tutorial by clicking the Help menu in the Quartus II window and selecting **PDF Tutorials**.

## Instantiate and Parameterize a Controller

After creating a Quartus II project, you need to instantiate a controller and its parameters.

### Instantiate a Controller

To instantiate a controller, perform the following steps:

1. Start the MegaWizard™ Plug-In Manager.

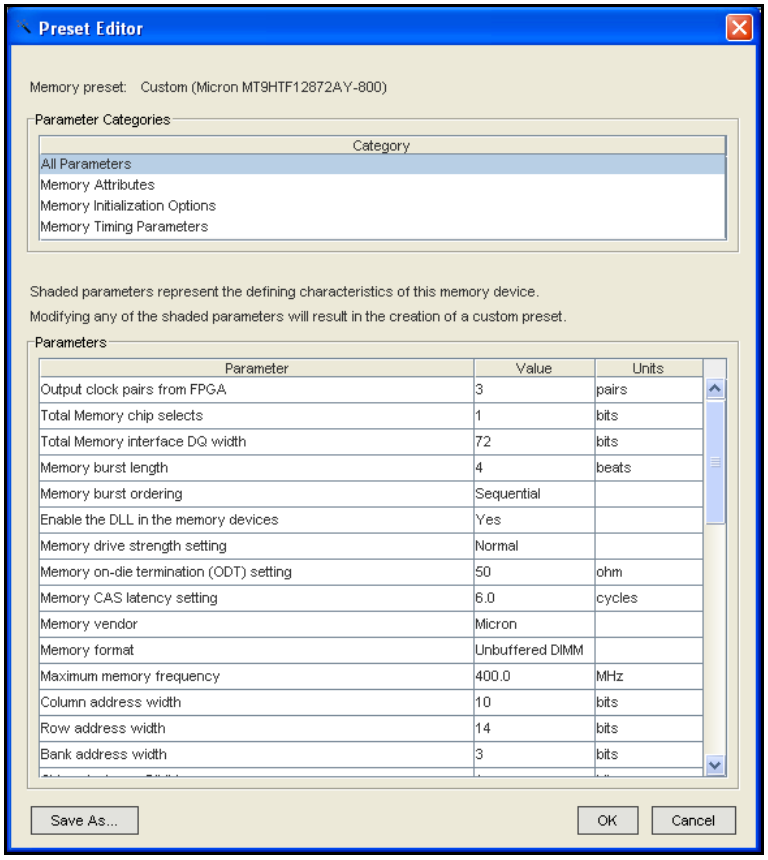
2. In the MegaWizard Plug-In Manager, expand **External Memory** in the **Interfaces** folder and select the **DDR2 SDRAM Controller with ALTMEMPHY**.
3. Type `ddr2_dimm` for the name of the DDR2 SDRAM high-performance controller.

## Parameterize DDR2 SDRAM with Stratix III

To parameterize the DDR2 SDRAM high-performance controller to interface with a 400-MHz, 72-bit wide DDR2 SDRAM interface.

1. In the **Memory Setting** tab, set **Speed grade** to **2**.
2. For **PLL reference clock frequency**, type **50** MHz (to match the on-board oscillator).
3. For **Memory clock frequency**, type **400** MHz.
4. For the **Memory Presets**, select **Micron MT9HTF12872AY-800**, which gives a 72-bit wide 1-GB 400-MHz DDR2 DIMM.
5. To create or modify a memory preset, click **Modify parameters**. In the **Preset Editor** dialog box, you can modify the memory presets, see Figure 4–1.

**Figure 4–1. Modify the Memory Presets to Create a Custom Memory**




The $t_{IS}$, $t_{IH}$, $t_{DS}$, and $t_{DH}$ parameters typically require slew rate derating.

For more information on slew rate derating and how to perform slew rate derating calculations, refer to the memory vendor datasheet.

Simulation and measurement show the following slew rate for the clock, address and command, DQ, and DQS pins on the Stratix III Development Board when using the default I/O standard and drive options:

- Address and command = 0.5 V/ns
- CLK and CLK# = 1.5 V/ns (differential)
- DQ = 1.5 V/ns
- DQS and DQSn = 2.8 V/ns (differential)

Hence, the correct $t_{IS}$, $t_{IH}$, $t_{DS}$, and $t_{DH}$ values for this design are:

- $t_{IS} = t_{ISb} + \Delta t_{IS} + (V_{IHAC} - V_{REF})$/address and command rising slew rate
  $= 175 + (-30) + 400 = 545$ ps
- $t_{IH} = t_{IHb} + \Delta t_{IH} + (V_{REF} - V_{ILDC})$/address and command rising slew rate
  $= 250 + (-95) + 250 = 405$ ps
- $t_{DS} = t_{DSa} + \Delta t_{DS} + (V_{IHAC} - V_{REF})$/DQ rising slew rate
  $= 50 + 67 + 133 = 250$ps
- $t_{DH} = t_{DHa} + \Delta t_{DH} + (V_{REF} - V_{ILDC})$/DQ rising slew rate
  $= 125 + 42 + 83 = 250$ ps

You should always simulate or measure your own design and topology to ensure accurate timing information and analysis.

For information about how to derate these numbers, refer to *Derate Memory Setup and Hold Timing* in the *DDR and DDR2 SDRAM Controller with ALTMEMPHY IP User Guide* section in volume 3 of the *External Memory Interface Handbook*.

The DDR2 SDRAM has a write requirement ($t_{DQSS}$) that states the positive edge of the DQS signal on writes must be within ± 25% (± 90°) of the positive edge of the DDR2 SDRAM clock input. To achieve this skew requirement, ALTMEMPHY-based designs always use DDR IOE registers to generate the CK and CK# signals.

6. In the **PHY Settings** tab, under **Advanced PHY Settings** turn on the **Use differential DQS** option to enhance signal to noise ratio. Turn on this option where noise margin is a concern. Differential DQS is recommended for DDR2 SDRAM interfaces operating at above 266 MHz and enhances signal to noise ratio. The Stratix III development board is routed for differential DQS.
7. Turn on the **Enable dynamic parallel on-chip termination (OCT)** option for this example. The Stratix III development board does not include discrete external termination on the DQ, DQS, or DM pins, as it was designed to use OCT.
8. Under **Address/Command Clock Settings**, for **Dedicated clock phase** type **240**. Timing analysis shows that 240° is optimal for the Stratix III development board.

9. Under **Board Timing Parameters**, for **Board skew** type **20 ps**. This timing parameter is the board trace variation between the DQ and DQS pins. If your board can perform better or worse than this value, update it accordingly. The wizard uses this number to calculate the overall system timing margin. For this design example, type the value of 20 ps as the board skew tolerance target is 20 ps.

The **Auto-Calibration Simulation Options** parameter is for RTL simulation only and is not applicable for gate-level simulation.

10. Click **Next**.
11. Click **Next**.
12. Turn the **Generate simulation model** option.
13. Click **Finish** to generate your MegaCore function variation. The MegaWizard Plug-In generates all the files necessary for your DDR2 SDRAM controller, and generates an example top-level design, which you may use to test or verify board operation.

For detailed step-by-step instructions for parameterizing the DDR2 SDRAM Controller with ALTMEMPHY IP, refer to *Volume 3: Implementing Altera Memory Interface IP* of the *External Memory Interface Handbook*.

# Perform RTL or Functional Simulation (Optional)

This section describes RTL and functional simulation.

## Set Up Simulation Options

To set up simulation option, perform the following steps:

1. Obtain and copy the vendors memory model to a suitable location. For example, obtain the **ddr2.v** and **ddr2_parameters.vh** memory model files from the Micron website and save them in the testbench directory.

   Some vendor DIMM models do not use DM pin operation, which can cause calibration failures. In these cases, use the vendors component models directly.

2. Open the memory model file in a text editor and add the following define statements to the top of the file:

   ```
   'define sg25
   'define x8
   ```

   The two define statements prepare the DDR2 SDRAM interface model.

   The first statement specifies the memory device speed grade as –25. The second statement specifies the memory device width per DQS.

3. Make sure the following statement is included in the model file:

   ```
   `include "ddr2_parameters.vh"
   ```

4. Open the testbench in a text editor, instantiate the downloaded memory model, and connect its signals to the rest of the design.

5. Delete the whole section between the START and END MEGAWIZARD comments.

```
// << START MEGAWIZARD INSERT MEMORY_ARRAY

    // This will need updating to match the memory models you are
using.


   // Instantiate a generated DDR memory model to match the datawidth
& chipselect requirements


    ddr2_dimm_mem_model mem (
        .mem_dq       (mem_dq),
        .mem_dqs      (mem_dqs),
        .mem_dqs_n    (mem_dqs_n),
        .mem_addr     (a_delayed),
        .mem_ba       (ba_delayed),
        .mem_clk      (clk_to_ram),
        .mem_clk_n    (clk_to_ram_n),
        .mem_cke      (cke_delayed),
        .mem_cs_n     (cs_n_delayed),
        .mem_ras_n    (ras_n_delayed),
        .mem_cas_n    (cas_n_delayed),
        .mem_we_n     (we_n_delayed),
        .mem_dm       (dm_delayed),
        .mem_odt      (odt_delayed)
    );


    // << END MEGAWIZARD INSERT MEMORY_ARRAY
```

6. Instantiate the first instance of the DDR2 SDRAM model by editing the following section in the testbench file:

```
ddr2 memory_0 (
    .clk      (clk_to_ram),
    .clk_n    (clk_to_ram_n),
    .cke      (cke_delayed),
    .cs_n     (cs_n_delayed),
    .ras_n    (ras_n_delayed),
    .cas_n    (cas_n_delayed),
    .we_n     (we_n_delayed),
    .dm_rdqs       (dm_delayed[0]),
    .ba       (ba_delayed),
    .addr     (a_delayed),
    .dq       (mem_dq[7:0]),
    .dqs      (mem_dqs[0]),
```

```
    .dqs_n    (mem_dqs_n[0]),
    .rdqs_n (),
    .odt      (odt_delayed)
    );
```


Make sure that port names in the memory model match with those in the testbench. Also the names and case of the names should be the same.

7. Similarly, create the other eight instances of the DDR2 module by changing the DQ, DQS, DQS_N, and DM bus indices, and the instance name `memory_0`. The following code shows the second ×8 memory instance of the DDR2 module:

```
ddr2 memory_1 (
    .clk      (clk_to_ram),
    .clk_n    (clk_to_ram_n),
    .cke      (cke_delayed),
    .cs_n     (cs_n_delayed),
    .ras_n    (ras_n_delayed),
    .cas_n    (cas_n_delayed),
    .we_n     (we_n_delayed),
    .dm_rdqs       (dm_delayed[1]),
    .ba       (ba_delayed),
    .addr     (a_delayed),
    .dq       (mem_dq[15:8]),
    .dqs      (mem_dqs[1]),
    .dqs_n    (mem_dqs_n[1]),
    .rdqs_n (),
    .odt      (odt_delayed)
    );
```

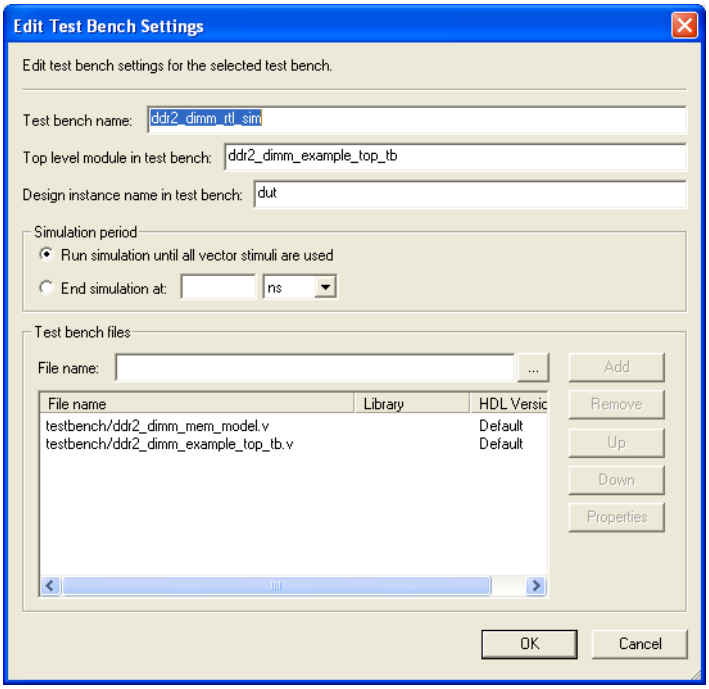
8. To launch the ModelSim simulator from the Quartus II software, set the path to the ModelSim simulator in the Quartus II software:
   a. On the Tools menu click **Options**.
   b. Click **EDA Tools Options** in the **Category** list.
   c. Set the path to the simulator.

## Run Simulation with NativeLink

To run the simulation with NativeLink, perform the following steps:

1. Set the absolute path to your third-party simulator executable file, by performing the following steps:
   a. On the Assignments menu, click **EDA Tool Settings**.
   b. In the **Category** list, expand **EDA Tool Settings** and click **Simulation**.
   c. Under **Tool Name**, select **ModelSim-Altera**.
   d. Under **NativeLink settings**, select **Compile test bench** and click **Test Benches**.
   e. Click **New**.
2. In the **Edit Test Bench Settings** dialog box, perform the following steps:
   a. Type the testbench name, testbench top-level module, design instance name, and simulation period.
   b. In the **Test bench files** field, include the testbench file and the memory model file (Figure 4–2).
   c. Click **OK**.

**Figure 4–2. Testbench Files**




3. To elaborate your design, on the Processing menu, point to **Start** and click **Start Analysis & Elaboration**.
4. On the Tools menu, point to the **Run EDA Simulation Tool** and click **EDA RTL Simulation**. This step creates the **\simulation** directory in your project directory and a script that compiles all necessary files and runs the simulation.

For example timing diagrams, refer to *Volume 3: Implementing Altera Memory Interface IP* of the *External Memory Interface Handbook*.

# Add Constraints

After instantiating the DDR2 SDRAM high-performance controller, the ALTMEMPHY megafunction generates the constraints files for the design example. Apply these constraints to the design before compilation.

## Add Timing Constraints

When you instantiate an SDRAM high-performance controller, it generates a timing constraints file, <*variation_name*>**_phy_ddr_timing.sdc**. The timing constraint file constrains the clock and input and output delay on the SDRAM high-performance controller.

To add timing constraints, perform the following steps:

1. On the Assignments menu click **Settings**.
2. In the **Category** list, expand **Timing Analysis Settings**, and select **TimeQuest Timing Analyzer**.
3. Select the <*variation_name*>**_phy_ddr_timing.sdc** file and click **Add**.
4. Click **OK**.

## Add Pin and DQ Group Assignments

The pin assignment script, <*variation_name*>**_pin_assignments.tcl**, sets up the I/O standards for the DDR2 SDRAM interface. It also launches the DQ group assignment script, <*variation_name*>**_phy_assign_dq_groups.tcl**, which relates the `DQ` and `DQS` pin groups together for the fitter to place them correctly in the Quartus II software.

This script does not create a clock for the design. You need to create a clock for the design and provide pin assignments for the signals of both the example driver and testbench that the MegaCore variation generates.

Run the <*variation_name*>**_pin_assignments.tcl** to add the pin, I/O standards, and DQ group assignments to the design example.

## Set Top-Level Entity

Before compiling the design, set the top-level entity of the project to the correct entity. The ALTMEMPHY megafunction entity is <*variation_name*>**_phy.v** or **vhd**; the SDRAM high-performance controller entity is <*variation_name*>**.v** or **vhd**.

The example top-level design, which instantiates the SDRAM high-performance controller and an example driver, is <*variation_name*>**_example_top.v** or **vhd**.

To set the top-level file, perform the following steps:

1. Open the top-level entity file, <*variation_name*>**_example_top.v** or **vhd**.
2. On the Project menu click **Set as Top-Level Entity**.

## Set Optimization Technique

To ensure the remaining unconstrained paths are routed with the highest speed and efficiency, perform the following steps to set the optimization technique:

1. On the Assignments menu, click **Settings**.
2. Select **Analysis & Synthesis Settings**.
3. Select **Speed** under **Optimization Technique**. Click **OK**.

## Set Fitter Effort

To set the fitter effort, perform the following steps:

1. On the Assignments menu, click **Settings**.
2. Expand **Fitter Settings**.
3. Turn on **Optimize hold timing** and select **All Paths**.
4. Turn on **Optimize multi-corner timing**.
5. Select **Standard Fit (highest effort)** under **Fitter effort**.
6. Click **OK**.

## Enter Pin Location Assignments

To enter the pin location assignments, perform the following steps:

1. Run analysis and synthesis. On the Processing menu, point to **Start** and click **Start Analysis & Synthesis**.
2. Assign all of your pins, so the Quartus II software fits your design correctly and gives correct timing analysis. To assign pin locations for the Stratix III development board, run the Altera-provided **S3_Host_DDR2_PinLocations.tcl** file or manually assign pin locations by using the Pin Planner.

   The SDRAM high-performance controller auto-generated scripts do not make any pin location assignments.

   If you are at the design exploration phase of your design cycle and do not have any PCB defined pin locations, you should still manually define an initial set of pin constraints, which can become more specific during your development process.

To manually assign pin locations using the Pin Planner, perform the following steps:

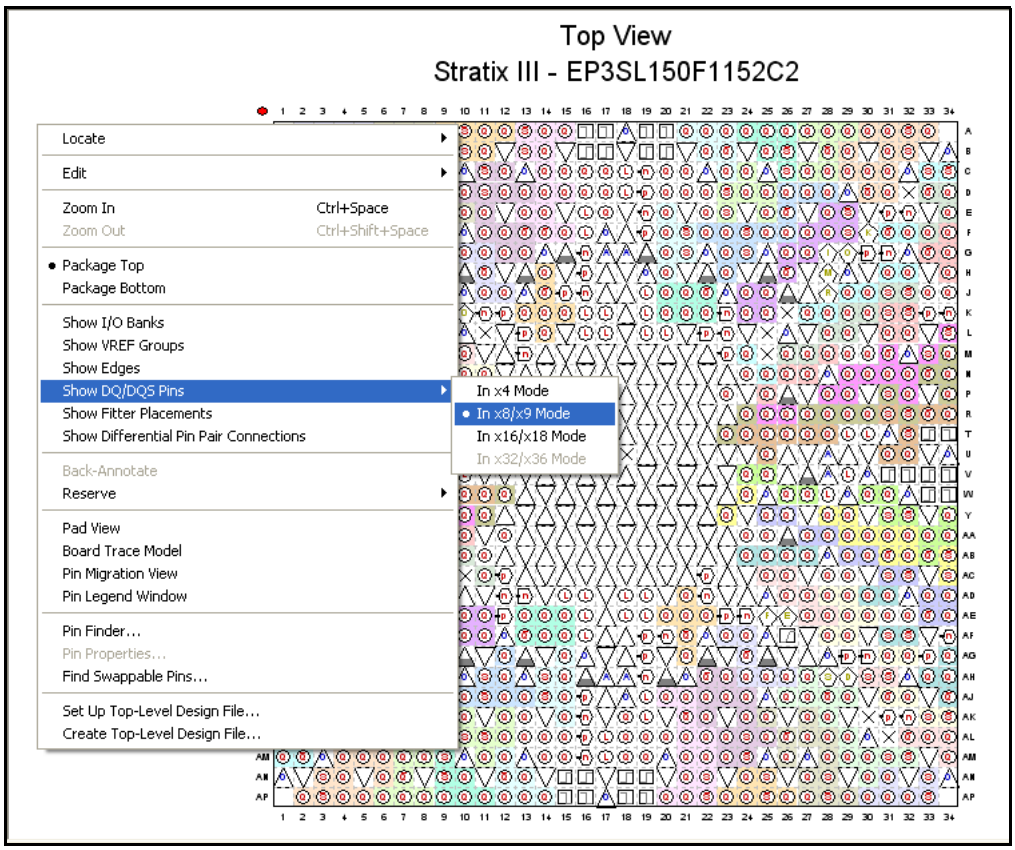
1. On the Assignments menu, click **Pin Planner**.
2. Assign `DQ` and `DQS` pins.
   a. To select the device DQS pin groups that the design uses, assign each `DQS` pin in your design to the required `DQS` pin in the Pin Planner. The Quartus II Fitter then automatically places the respective `DQ` signals onto suitable `DQ` pins within each group. To see DQS groups in the Pin Planner, right-click, select **Show DQ/DQS Pins**, and click **In x8/x9 Mode**. The Pin Planner shows each DQS group in a different color and with a different legend: S = `DQS` pin, Sbar = `DQSn` pin and Q = `DQ` pin, refer to Figure 4–3 on page 4–10.

Most DDR2 SDRAM devices operate in ×8/×9 mode, however as some DDR2 SDRAM devices operate in ×4 mode, refer to your specific memory device datasheet.

b. Select the DQ mode to match the DQ group width (number of DQ pins/number of DQS pins) of your memory device. DQ mode is not related to the memory interface width.

DQ group order and DQ pin order within each group is not important. However, you must place the DQ pins in the same group as their respective strobe pin.

**Figure 4–3. Quartus II Pin Planner, Show DQ/DQS Pins, In x8/x9 Mode**




3. Place the DM pins within their respective DQ group.
4. Place address and control command pins on any spare I/O pins ideally within the same bank or side of the device as the mem_clk pins.
5. Ensure that you place mem_clk pins on differential I/O pairs for the CK/CK# pin pair. To identify differential I/O pairs, right-click in the Pin Planner and select **Show Differential Pin Pair Connections**. The pin pairs show a red line between each pin pair.

You must place mem_clk[0] and mem_clk_n[0] on a DIFFIO_RX pin pair, if your design uses differential DQS signaling.

6. Ensure `mem_clk` pins use any regular adjacent I/O pins—ideally differential I/O pairs for the `CK/CK#` pin pair. To identify differential I/O pairs, right-click in the Pin Planner and select **Show Differential Pin Pair Connections**. The pin pairs show a red line between each pin pair.
7. Place the `clock_source` pin on a dedicated PLL clock input pin with a direct connection to the SDRAM controller PLL and DLL pair—usually on the same side of the device as your memory interface. This recommendation reduces PLL jitter, saves a global clock resource, and eases timing and fitter effort.
8. Place the `global_reset_n` pin (like any high fan-out signal) on a dedicated clock pin.

For more information on how to use the Quartus II Pin Planner, refer to the *I/O Management* chapter in volume 2 of the *Quartus II Handbook*.

## Assign Virtual Pins

The example top-level design, which is auto-generated by the high- performance controller, includes an example driver to stimulate the interface. This example driver is not part of the SDRAM high-performance controller IP, but allows easy testing of the IP.

The example driver outputs several test signals to indicate its operation and the status of the stimulated memory interface. These signals are `pnf`, `pnf_per_byte`, and `test_complete`. These signals are not part of the memory interface, but are to facilitate testing. You should connect these signals to either a debug bus or assign the signals to virtual pins using the Quartus II Assignment Editor. When using the example driver for testing, do not remove these signals from the top-level signal list. If you remove these signals from the top-level module, the Quartus II software optimizes the driver away and the example driver fails.

To assign virtual pin assignments for the Stratix III development board, run the Altera-provided **s3_Host_ddr2_exdriver_vpin.tcl** file or manually assign virtual pin assignments using the Assignment Editor.

## Advanced I/O Timing

ALTMEMPHY-based designs assume that the memory address and command signals are matched length to the memory clock signals. Typically, this length match is not true for DIMM-based designs. You should verify the difference in your design. For the Stratix III development board fitted with the MT9HTF12872AY DIMM, the address and command signals remain asserted 750 ps longer than the clock signals.

To amend the TimeQuest **.sdc** file, <*variation name*>**_phy_ddr_timing.sdc**, to include this difference, perform the following steps:

1. Open the **ddr2_dimm_phy_ddr_timing.sdc** file in a text editor and find the following line (usually line 31):

```
set t(additional_addresscmd_tpd) 0.000
```

2. Change the line to the following text:

```
set t(additional_addresscmd_tpd) 0.750
```

3. Save the file.

If the DDR2 SDRAM controller **.sdc** file is regenerated, this change is lost and you must re-edit the file.

## Enter Board Trace Delay Models

For accurate I/O timing analysis, the Quartus II project must include the board trace and loading information. This information should be derived and refined during your PCB development process of prelayout (line) simulation and final post-layout (board) simulation. For external memory interfaces that use memory modules (DIMMs), this information should include the trace and loading information of the module in addition to the main and host platform. You can obtain the information from your memory vendor.

To enter board trace information, perform the following steps:

1. In the Pin Planner, select the pin or group of pins that you want to enter the information for.
2. Right-click and select **Board Trace Model**.

Figure 4–4 through 4–14 show a typical board trace model for an address, memory clock, DQ, and DQS pin on the Stratix III development board including the data for the MT9HTF12872AY-800E memory module that is included with the kit.

**Figure 4–4. Stratix III Development Board Address Signal Board Trace Model**

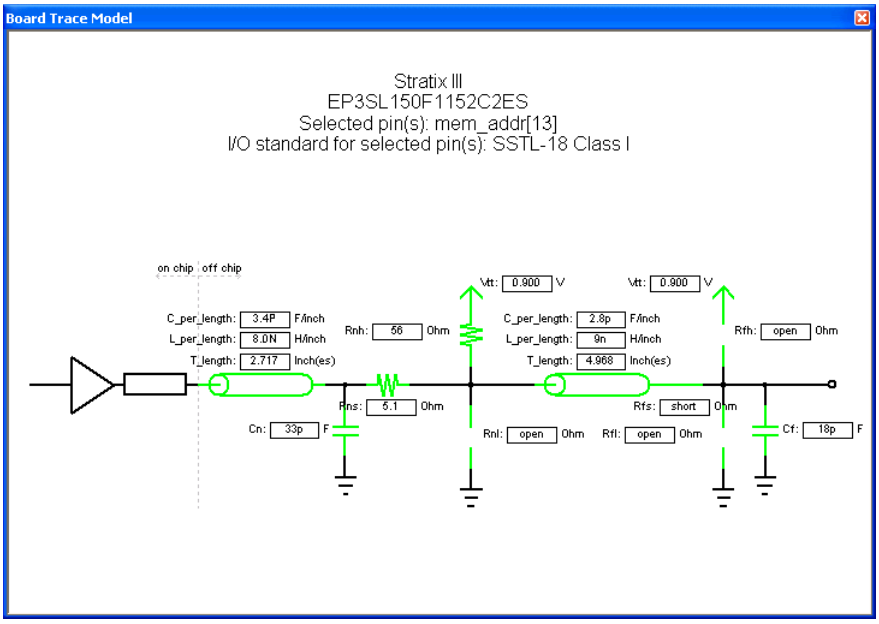



**Figure 4–5. Stratix III Development Board Memory Clock Signal Board Trace Model**

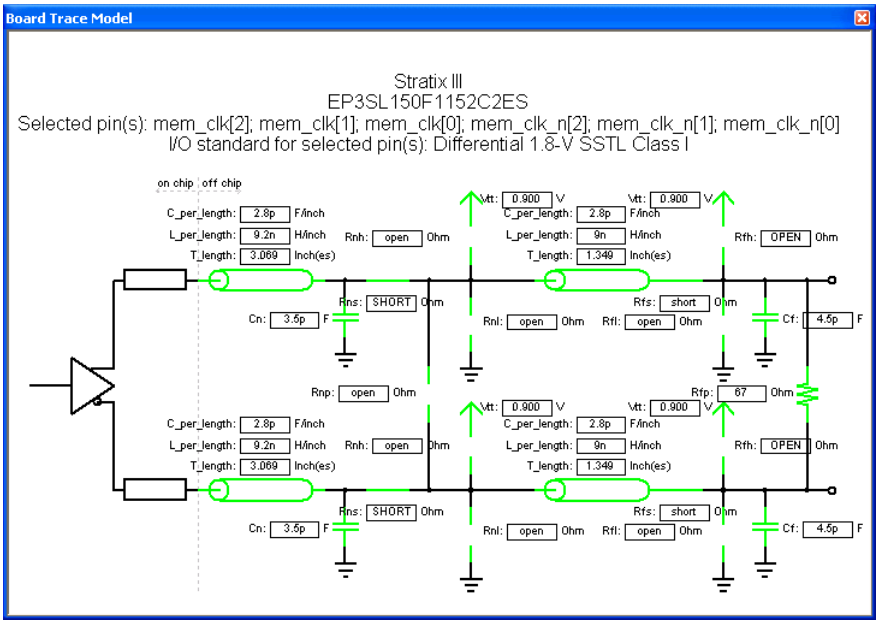


**Figure 4–6. Stratix III Development Board DQ Signal Board Trace Model**

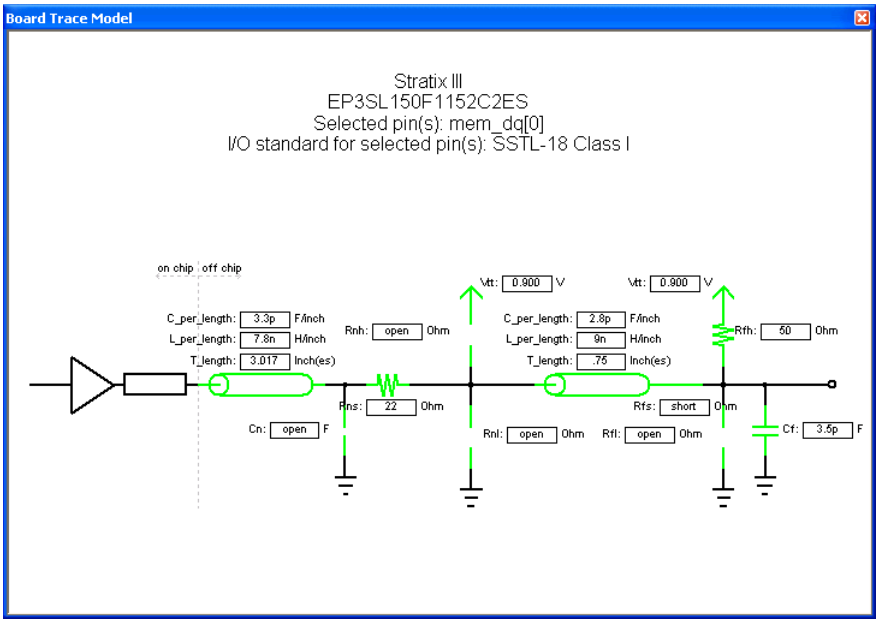



**Figure 4–7. Stratix III Development Board DQS Signal Board Trace Model**

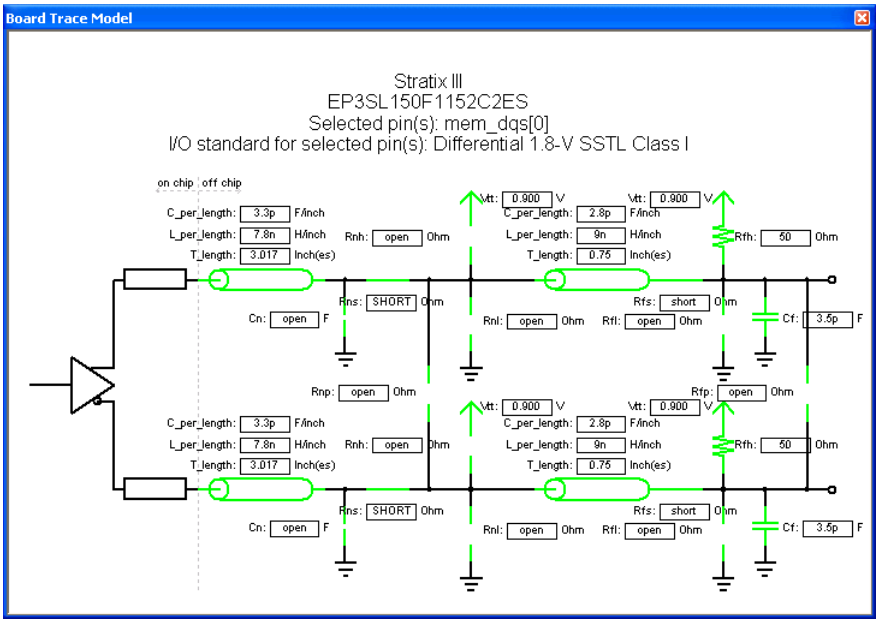


Table 4–1 shows the board trace model parameters for the Stratix III development board.

**Table 4–1. Stratix III Development Board Trace Model Summary**

| Net | Near (FPGA End of Line) | | | | | | Far (Memory End of Line) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Length | C_per_length (pF/In) | L_per_length (nH/In) | Cn (pF) | Rns | Rnh | Length | C_per_length (pF/In) | L_per_length (nH/In) | Cf (pF) | Rfh/Rfp |
| Addr *(1)* | 2.717 | 3.3 | 7.8 | 33 | 5.1 | 56 | 4.968 | 2.8 | 9 | 18 | — |
| CLK | 3.069 | 2.8 | 9.2 | 3.5 | — | — | 1.349 | 2.8 | 9 | 4.5 | 67 |
| CKE/CS# | 2.717 | 3.3 | 7.8 | 33 | — | 56 | 3.936 | 2.8 | 9 | 42 | — |
| ODT | 2.717 | 3.3 | 7.8 | 33 | — | 56 | 3.936 | 2.8 | 9 | 39.75 | — |
| DQS0 | 3.017 | 3.4 | 7.8 | — | 22 | — | 0.750 | 2.8 | 9 | 3.5 | 50 |
| DQS1 | 3.005 | 3.4 | 8.1 | — | 22 | — | 0.760 | 2.8 | 9 | 3.5 | 50 |
| DQS2 | 2.851 | 3.4 | 7.8 | — | 22 | — | 0.760 | 2.8 | 9 | 3.5 | 50 |
| DQS3 | 2.653 | 3.4 | 8.1 | — | 22 | — | 0.760 | 2.8 | 9 | 3.5 | 50 |
| DQS4 | 2.686 | 3.4 | 7.8 | — | 22 | — | 0.760 | 2.8 | 9 | 3.5 | 50 |
| DQS5 | 2.701 | 3.4 | 7.8 | — | 22 | — | 0.760 | 2.8 | 9 | 3.5 | 50 |
| DQS6 | 2.750 | 3.4 | 7.8 | — | 22 | — | 0.760 | 2.8 | 9 | 3.5 | 50 |
| DQS7 | 2.923 | 3.4 | 8.1 | — | 22 | — | 0.750 | 2.8 | 9 | 3.5 | 50 |
| DQS8 | 2.536 | 3.4 | 7.8 | — | 22 | — | 0.940 | 2.8 | 9 | 3.5 | 50 |

**Note to Table 4–1:**

(1) Addr = A, BA, WE#, RAS#, ODT, and CAS#.

Altera recommends you use the **Board Trace Model** assignment on all DDR and DDR2 SDRAM interface signals. To apply board trace model assignments for the Stratix III development board, run the Altera-provided **S3_Host_DDR2_BTModels.tcl** file or manually assign virtual pin assignments using the Quartus II Pin Planner.

The Stratix III development board has the following compensation capacitors fitted to its DDR2 SDRAM address and command, and CLK and CLK# signals:

- Address and command = 33 pF compensation capacitors
- CLK and CLK# = 7 pF (differential) compensation capacitors.

These capacitors are typically fitted to designs that use asymmetric DIMM designs. You should simulate your design to see if compensation capacitors are required. Stratix III devices have various programmable drive strength and OCT I/O options, so compensation capacitors should not usually be required. Fitting compensation capacitors reduces the edge rate of your signals, so you should observe memory vendor derating guidelines.

For more information on compensation capacitors, refer to *Micron Technical Note TN_47_01*.

# Compile Design and Verify Timing

To compile the design, on the Processing menu, click **Start Compilation**.

Figure 4–8 shows the timing margin report in the message window in the Quartus II software.

**Figure 4–8. Timing Margin Report in the Quartus II Software**

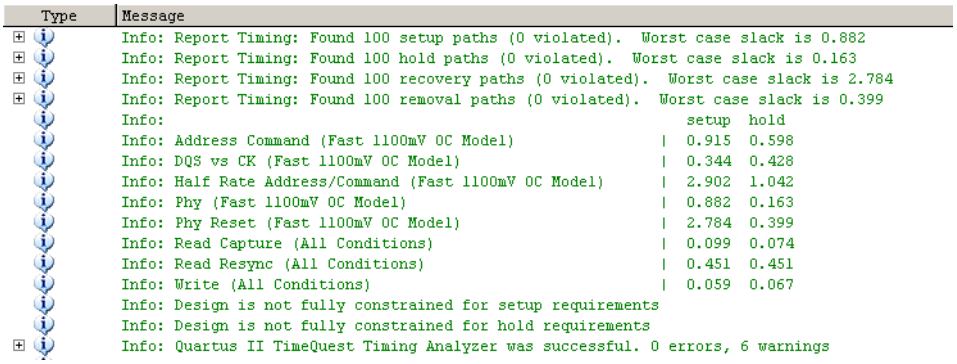
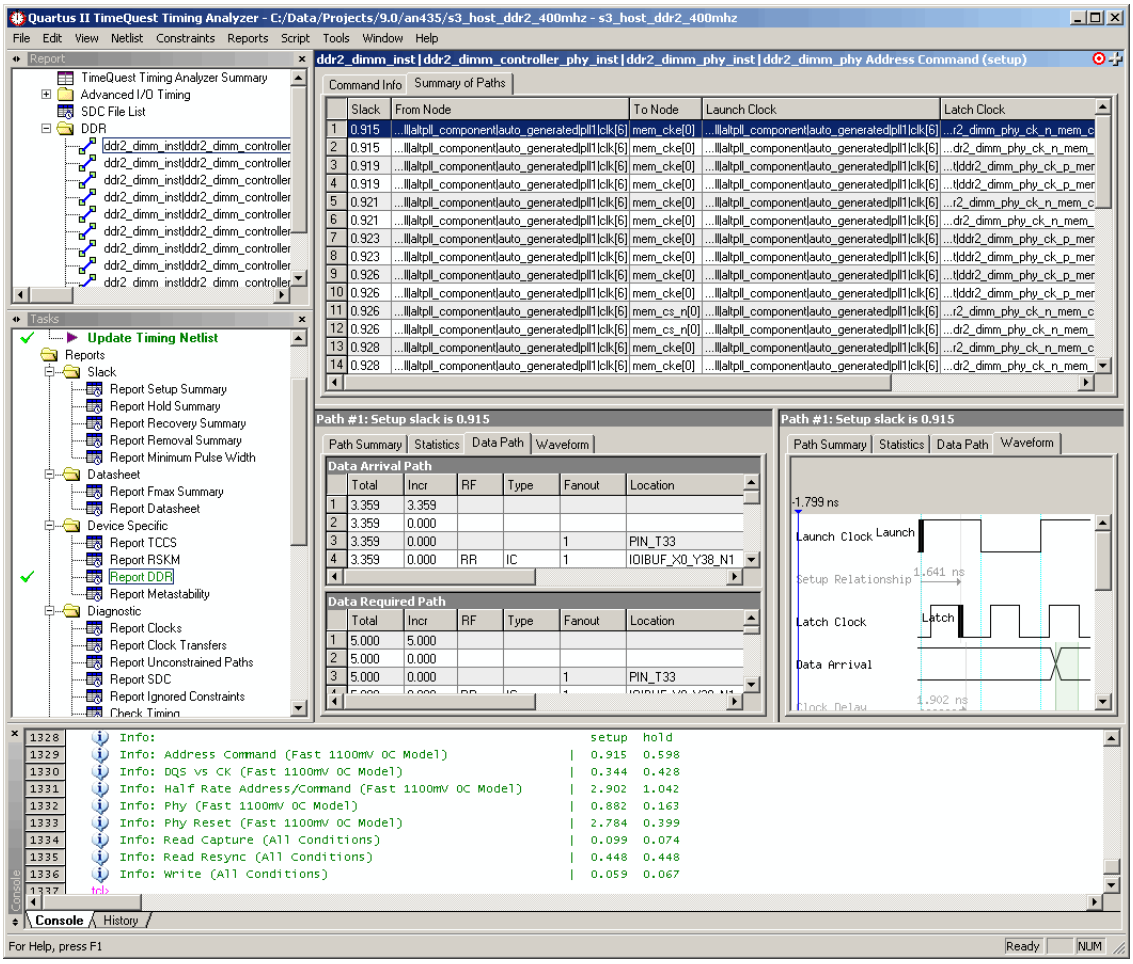


Figure 4–9 on page 4–17 shows the timing margin report in the TimeQuest Timing Analyzer window after running the Report DDR Task. The results are the same as the Quartus II software results.

**Figure 4–9. Timing Margin Report in TimeQuest Timing Analyzer**




For more information about the TimeQuest timing analyzer, refer to *The Quartus II TimeQuest Timing Analyzer* chapter in volume III of the Quartus II Handbook.

For information, refer to the *Timing Analysis* section in volume 4 of the *External Memory Interface Handbook*.

# Adjust Constraints

As the design meets timing, there is no need to adjust any constraints.

For information on timing closure, refer to the *Timing Analysis* section in volume 4 of the *External Memory Interface Handbook*.

# Determine Board Design Constraints and Perform Board-Level Simulations

Stratix III devices support both series and parallel OCT resistors to improve signal integrity. Another benefit of the Stratix III OCT resistors is eliminating the need for external termination resistors on the FPGA side. This feature simplifies board design and reduces overall board cost. You can dynamically switch between the series and parallel OCT resistor depending on whether the Stratix III devices are performing a write or a read operation. The OCT features offer user-mode calibration to compensate for any variation in voltage and temperature during normal operation to ensure that the OCT values remain constant. The parallel and series OCT features of the Stratix III devices are available in either a 25-Ω or 50-Ω setting.

Refer to the *Stratix III Device I/O Features* chapter of the *Stratix III Device Handbook*, for more information about the OCT features.

Refer to the respective memory data sheet, for more information about the available settings of the ODT and the output driver impedance features, and the timing requirements for driving the ODT pin in DDR2 SDRAM.

Figure 4–10 illustrates the write operation to the DDR2 SDRAM with the ODT feature turned on and using the 50-Ω series OCT feature of the Stratix III FPGA device. In this setup, the transmitter (FPGA) is properly terminated with matching impedance to the transmission line, thus eliminating any ringing or reflection. The receiver (DDR2 SDRAM) is also properly terminated when the dynamic ODT setting is at 75 Ω

**Figure 4–10. Write Operation Using Parallel ODT and 50-Ω Series OCT of the Stratix III FPGA Device**

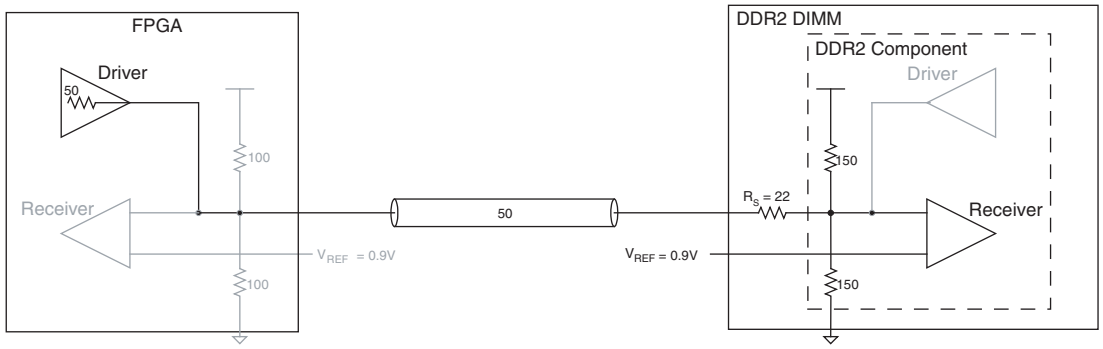
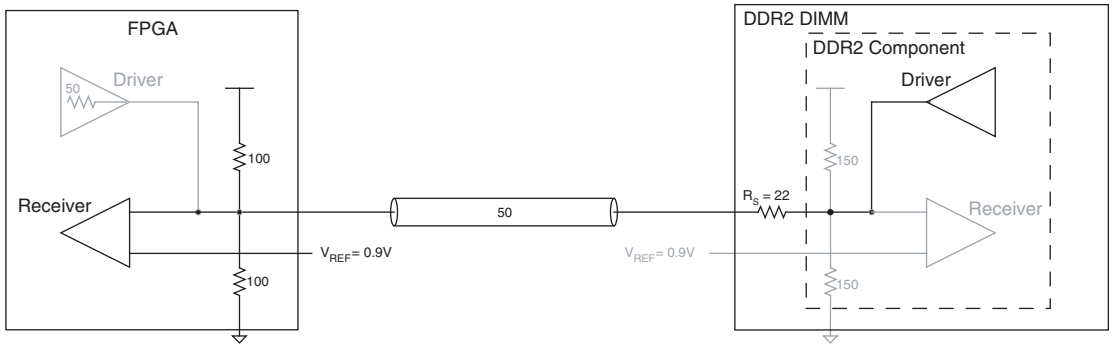


Figure 4–11 illustrates the read operation from the DDR2 SDRAM using the parallel OCT feature of the Stratix III device. In this setup, the driver's (DDR2 SDRAM) output impedance is not larger than 21 Ω. This impedance is in keeping with SSTL-18 JEDEC specification JESD79-2. Combined with an on dual-inline memory modules (DIMM) series resistor, the impedance matches that of the transmission line resulting in optimal signal transmission to the receiver (FPGA). On the receiver (FPGA) side, it is properly terminated with 50-Ω, which matches the impedance of the transmission line, thus eliminating any ringing or reflection.

**Figure 4–11. Read Operation From DDR2 SDRAM Using the Parallel OCT Feature of the Stratix III FPGA Device**




Finally, the loading seen by the FPGA during writes to the memory is different between a system using DIMMs versus a system using components. The additional loading from the DIMM connector can reduce the edge rates of the signals arriving at the memory thus affecting available timing margin.

For more information about Stratix III devices signal integrity, refer to the Stratix III Device Signal and Power Integrity page.

## Adjust Termination and Drive Strength

Due to the loading of the line, the Quartus II software may report that the default or chosen drive strength cannot drive the line to the specified toggle rate or minimum pulse width. If you encounter this error, use the stronger drive strength I/O standard. Ensure that you re-simulate your design with the new drive strength to ensure that signal quality is still acceptable.

## Verify Design on a Board

The SignalTap II Embedded Logic Analyzer shows read and write activity in the system.

For more information on using the SignalTap II Embedded Logic Analyzer, refer to the *Design Debugging Using the SignalTap II Embedded Logic Analyzer* chapter in the *Quartus II Handbook*, *AN 323: Using SignalTap II Embedded Logic Analyzers in SOPC Builder Systems* and *AN 446: Debugging Nios II Systems with the SignalTap II Logic Analyzer*.

To add the SignalTap II Embedded Logic Analyzer, perform the following steps:

For this design, Altera provides a Tcl file, **S3_Host_DDR2_SignalTap.tcl**, to automate the following steps. The **.stp** file is included with the design example file.

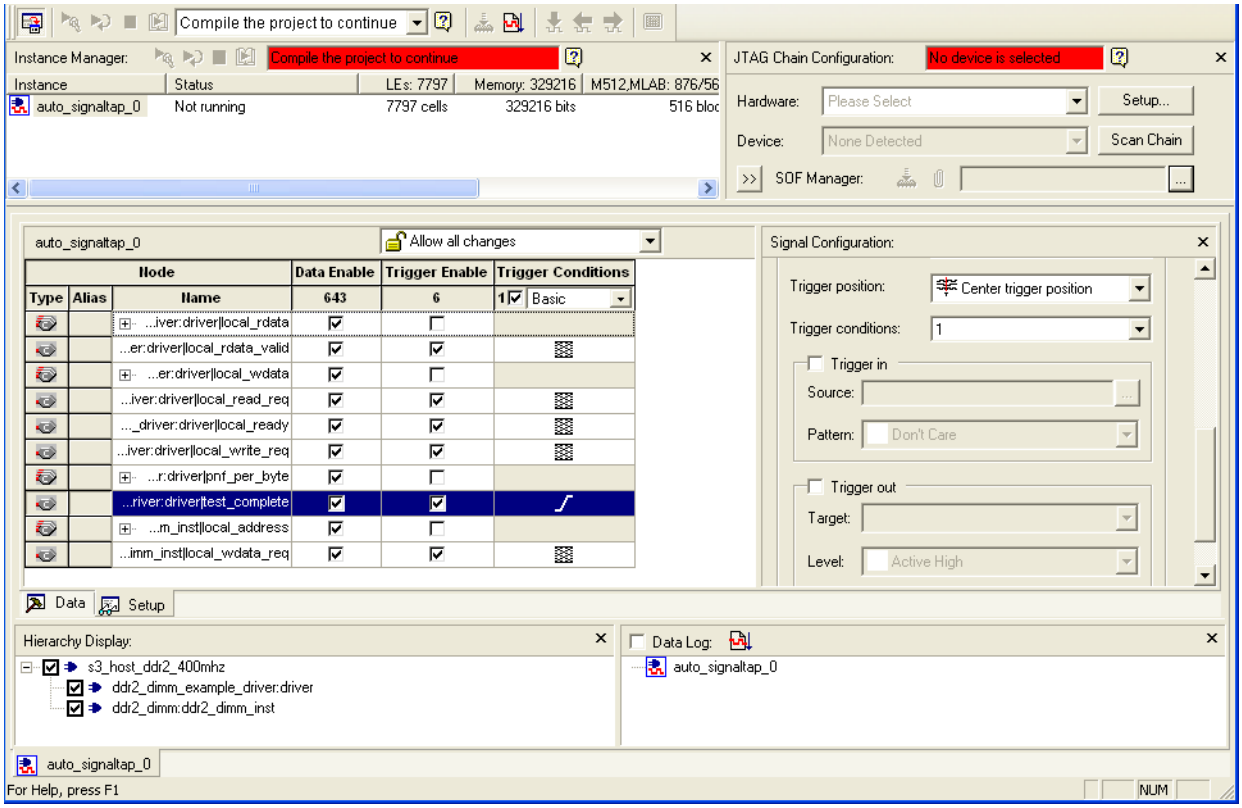
1. On the Tools menu, click **SignalTap II Logic Analyzer**.
2. In the **Signal Configuration** window next to the **Clock** box, click **…** (**Browse Node Finder**).
3. Type `*phy_clk` in the **Named** box, for **Filter** select **SignalTap II: pre-synthesis** and click **List**.
4. Select **ddr2_dimm_example_top | ddr2_dimm:ddr2_dimm_inst | ddr2_dimm_controller_phy:ddr2_dimm_controller_phy_inst | phy_clk | phy_clk** in **Nodes Found** and click **>** to add the signal to **Selected Nodes**.
5. Click **OK**.
6. Under **Signal Configuration**, specify the following settings:
   - For **Sample depth**, select **512**
   - For **RAM type**, select **Auto**
   - For **Trigger flow control**, select **Sequential**
   - For **Trigger position**, select **Center trigger position**
   - For **Trigger conditions**, select **1**
7. On the Edit menu, click **Add Nodes**.
8. Search for specific nodes by typing `*local*` in the **Named** box, for **Filter** select **SignalTap II: pre-synthesis** and click **List**.
9. Select the following nodes in **Nodes Found** and click **>** to add to **Selected Nodes**:
   - **local_address**
   - **local_rdata**
   - **local_rdata_valid**
   - **local_read_req**
   - **local_ready**
   - **local_wdata**
   - **local_wdata_req**
   - **local_write_req**
   - **pnf**
   - **pnf_per_byte**
   - **test_complete** (trigger)

   Do not add any DDR SDRAM interface signals to the SignalTap II Embedded Logic Analyzer. The load on these signals increases and adversely affects the timing analysis.
10. Click **OK**.

11. To reduce the SignalTap II logic size, turn off **Trigger Enable** on the following bus signals:
    - **local_address**
    - **local_rdata**
    - **local_wdata**
    - **pnf_per_byte**
12. Right-click **Trigger Conditions** for the `test_complete` signal and select **Rising Edge**.

    Figure 4–12 shows the completed SignalTap II Embedded Logic Analyzer.

**Figure 4–12. SignalTap II Embedded Logic Analyzer**




13. On the File menu, click **Save**, to save the SignalTap II **.stp** file to your project.

    If you see the message **Do you want to enable SignalTap II file "stp1.stp" for the current project**, click **Yes**.

## Compile the Project

Once you add signals to the SignalTap II Embedded Logic Analyzer, recompile your design, on the Processing menu, click **Start Compilation**.

## Verify Timing

Once the design compiles, ensure that TimeQuest timing analysis passes successfully. In addition to this FPGA timing analysis, check your PCB or system SDRAM timing. To run timing analysis, run the <*variation name*>**_phy_report_timing.tcl** script.

1. On the Tools menu, click **Tcl Scripts**.
2. Select <*variation name*>**_phy_report_timing.tcl** and click **Run**.

## Download the Object File

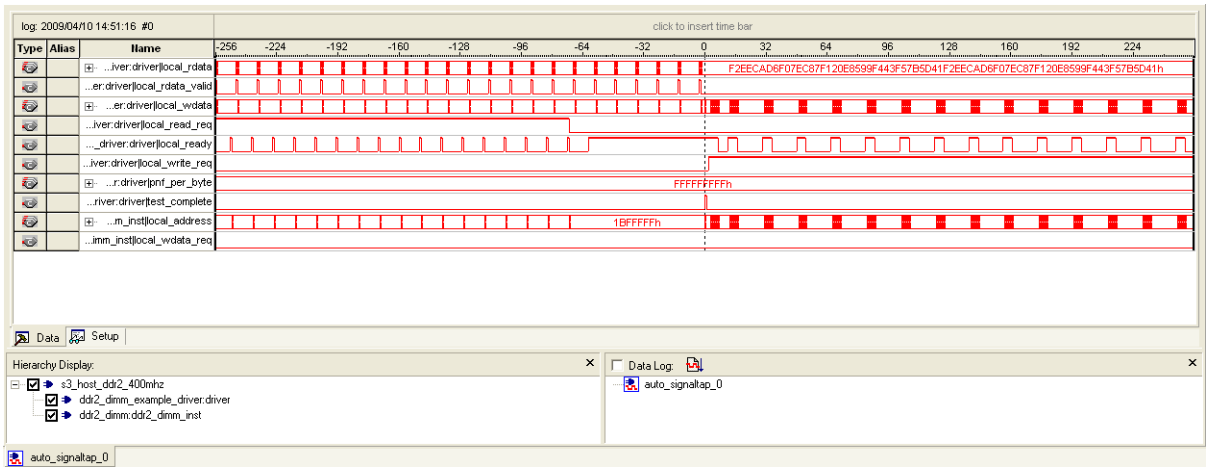
To download the object file to the device, perform the following steps:

1. Connect the development board to your computer.
2. On the Tools menu, click **SignalTap II Logic Analyzer**. The SignalTap II dialog box appears.
3. Click **...** to open the **Select Program Files** dialog box.
4. Select <*your project name*>**.sof**.
5. Click **Open**.
6. To download the file, click the **Program Device** button.

## Test the Design Example in Hardware

When the design example including SignalTap II successfully downloads to your development board, click **Run Analysis** to run once, or click **Autorun Analysis** to run continuously. Figure 4–13 shows the design analysis.

**Figure 4–13. SignalTap II Example DDR2 SDRAM Design Analysis**



# 5. Using DDR3 SDRAM Devices in Stratix III and Stratix IV Devices

This tutorial describes how to use the design flow to implement a 64-bit wide, 533-MHz, 1,066-Mbps DDR3 SDRAM interface with Stratix III devices, and a 72-bit wide, 400-MHz, 800-Mbps DDR3 SDRAM interface with Stratix IV devices. The design examples also provide some recommended settings to simplify the design.

The design examples target the Stratix III memory demonstration kit, which includes a 72-bit wide 1-GB Micron MT9JSF12872AY-1G1BZES 533-MHz DDR3 SDRAM DIMM, and the Stratix IV GX FPGA development kit, which includes four 16-bit wide 1-GB Micron MT41J64M16LA-187E 533-MHz DDR3 SDRAM devices.

To download the design examples, **emi_ddr3_siii.zip** and **emi_ddr3_siv.zip**, go to the External Memory Interface Design Examples page. You can also use this design example if you are targeting a HardCopy device for your design. For ALTMEMPHY megafunction supported devices, refer to mmmm section in volume 3 of the *External Memory Interface Handbook*.

The Stratix III memory demonstration kit is not available for purchase. The early versions of the Stratix III memory demonstration kit include a MT16JTF25664AY-1G1D1 DDR3 SDRAM DIMM, which is a dual-rank DIMM and is not supported by ALTMEMPHY-based solutions. Make sure that the MT9JSF12872AY-1G1BZES DDR3 SDRAM DIMM is fitted.

For more information about the design flow, refer to the *Recommended Design Flow* section in volume 1 of the *External Memory Interface Handbook*.

## System Requirements

This tutorial requires the following hardware and software for the DDR3 SDRAM interface with a Stratix III device:

- Quartus II software version 9.1
- DDR3 SDRAM Controller with ALTMEMPHY IP version 9.1
- Stratix III memory demonstration kit, with Micron MT9JSF12872AY-1G1BZES device

This tutorial requires the following hardware and software for the DDR3 SDRAM interface with a Stratix IV device:

- Quartus II software version 9.1
- DDR3 SDRAM Controller with ALTMEMPHY IP version 9.1
- Stratix IV GX FPGA development kit, with Micron MT41J64M16LA-187E device

## Create a Quartus II Project

Create a project in the Quartus II software that targets the appropriate device; the EP3SL150F1152-C2ES device for the Stratix III device family or the EP4SGX230KF40C3ES device for the Stratix IV device family.

For step-by-step instructions to create a Quartus II project, refer to the Quartus II software tutorial by clicking the Help menu in the Quartus II window and selecting **PDF Tutorials**.

# Instantiate and Parameterize a Controller

After you create a Quartus II project, you must instantiate a controller and its parameters.

## Instantiate a Controller

To instantiate a controller, perform the following steps:

- DDR3 SDRAM controller with a Stratix III device:
  a. Copy the memory parameters files, **S3MB1_Derated (Micron MT9JSF12872AY-1G1BZES).xml**, to your <*installation directory*>**\91\ip\ddr3_high_perf\lib** directory.
  b. Start the MegaWizard™ Plug-In Manager.
  c. In the MegaWizard Plug-In Manager, expand **External Memory** in the **Interfaces** folder and select the **DDR3 SDRAM High Performance Controller**.
  d. Type `ddr3_dimm` for the name of the DDR3 SDRAM high-performance controller.
- DDR3 SDRAM controller with a Stratix IV device:
  a. Copy the memory parameters files, **SD_PCIe_DDR3_Kit (4xMicron MT41J64M16LA-187E).xml**, to your <*installation directory*>**\91\ip\ddr3_high_perf\lib** directory.
  b. Start the MegaWizard Plug-In Manager.
  c. In the MegaWizard Plug-In Manager, expand **External Memory** in the **Interfaces** folder and select the **DDR3 SDRAM High Performance Controller**.
  d. Type `ddr3_bot_x64` for the name of the DDR3 SDRAM high-performance controller.

Both design examples instantiate the ALTMEMPHY megafunction automatically.

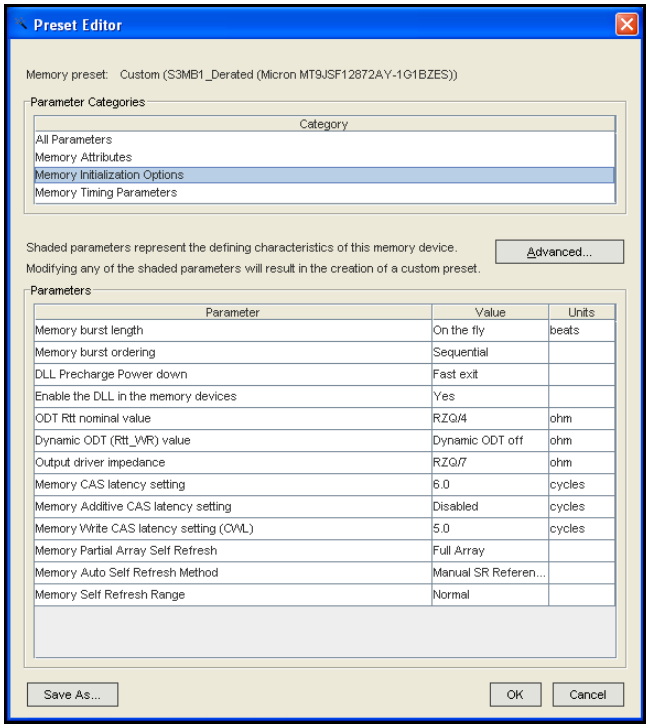
## Parameterize DDR3 SDRAM with Stratix III

To parameterize the DDR3 SDRAM high-performance controller to interface with a 533-MHz, 72-bit wide DDR3 SDRAM interface, perform the following steps:

1. In the **Memory Setting** tab, set **Speed grade** to **2**.
2. For **PLL reference clock frequency**, type **100** MHz (to match the on-board oscillator).
3. For **Memory clock frequency**, type **533** MHz (the maximum frequency supported for DDR3 SDRAM interfaces on Stratix III devices).
4. For **Memory Presets**, select **S3MB1_Derated (Micron MT9JSF12872AY-1G1BZES)**, which gives a 72-bit wide, 1,152-Mbps 533-MHz DDR3 unbuffered DIMM.

5. To create or modify a memory preset, click **Modify parameters**. In the **Preset Editor** dialog box, you can modify the memory presets. Refer to Figure 5–1.

**Figure 5–1. Modify the Memory Presets to Create a Custom Memory**




The $t_{AC}$ and $t_{QHS}$ parameters are often not defined by memory vendors, as these values are only used in static RTD calculations and non-DQS capture mode. The wizard does not require these parameters, so use the default values.

The $t_{IS}$, $t_{IH}$, $t_{DS}$, and $t_{DH}$ parameters typically require slew rate derating.

Simulation and measurement show the following slew rate for the clock, address and command, and DQ and DQS pins on the Stratix III memory demonstration board when using the default I/O standard and drive options:

- Address and command = 1.5 V/ns
- CLK and CLK# = 3 V/ns (differential)
- DQ = 2 V/ns
- DQS = 3 V/ns (differential)

Hence, the correct $t_{IS}$, $t_{IH}$, $t_{DS}$, and $t_{DH}$ values for this design are:

- $t_{IS} = t_{ISb} + \Delta t_{IS} + (V_{IHAC} - V_{REF})$/address and command rising slew rate
  $=125 + 59 + 175/1.5 = 301$ ps
- $t_{IH} = t_{IHb} + \Delta t_{IH} + (V_{REF} - V_{ILDC})$/address and command rising slew rate
  $= 200 + 34 + 100/1.5 = 301$ ps
- $t_{DS} = t_{DSa} + \Delta t_{DS} + (V_{IHAC} - V_{REF})$/DQ rising slew rate
  $= 25 + 88 + 175/2 = 201$ ps
- $t_{DH} = t_{DHa} + \Delta t_{DH} + (V_{REF} - V_{ILDC})$/DQ rising slew rate

= 100 + 50 + 100/2 = 200 ps

For more information about how to derate these numbers, refer to *Derate Memory Setup and Hold Timing* in the *DDR3 SDRAM Controller with ALTMEMPHY IP User Guide* in volume 3 of the *External Memory Interface Handbook*.

The DDR3 SDRAM has a write requirement ($t_{DQSS}$) that states the positive edge of the DQS signal on writes must be within ±25% (±90°) of the positive edge of the DDR3 SDRAM clock input. To achieve this skew requirement, the ALTMEMPHY-based designs always use the DDR IOE registers to generate the CK and CK# signals.

6. To set the ODT settings for the DDR3 SDRAM interface on your board, in the **Preset Editor** dialog box, select **Memory Initialization Options**.
7. In the **Memory Initialization Options** dialog box, perform the following steps:
   a. For **Output driver impedance**, select **RZQ/7** (which is 34).
   b. For **Dynamic ODT (Rtt_WR) value**, select **Dynamic ODT off**.
   c. For **ODT Rtt nominal value**, select **RZQ/4** (which is 60).
   d. Click **OK** to apply the settings and exit the dialog box.
8. In the **PHY Settings** tab, under **Advanced PHY Settings** turn on the **Enable dynamic parallel on-chip termination (OCT)** option for this example.
9. Under **Address/Command Clock Setting**s, for **Dedicated clock phase** type **240**.
10. Under **Board Timing Parameters**, for **Board skew** type **20 ps**. This timing parameter is the board trace variation between the CK, CK#, CAC, DQ, DQS, and DQS# pins. If your board can perform better or worse than this value, update it accordingly.

The **Auto-Calibration Simulation Options** parameter is for RTL simulation only and is not applicable for gate-level simulation.

11. In the **Controller Settings** tab, for **Controller Architecture** select **High Performance Controller II** for higher efficiency and advanced features.
12. Under **Efficiency**, select the specified values for the following options:
    a. For **Command Queue Look-Ahead Depth**, select **6**.
    b. For **Local-to-Memory Address Mapping**, select **CHIP-ROW-BANK-COL**.
    c. For **Local Maximum Burst Count**, select **4**.
13. Click **Next**.
14. Turn on the **Generate simulation model** option to generate the simulation model for the RTL simulation.
15. Click **Finish** to generate your MegaCore function variation. The MegaWizard Plug-In Manager generates all the files necessary for your DDR3 SDRAM controller, and generates an example top-level design, which you use to test or verify board operation.

## Parameterize DDR3 SDRAM with Stratix IV

To parameterize the DDR3 SDRAM high-performance controller to interface with a 400-MHz, 64-bit wide DDR3 SDRAM interface, perform the following steps:

1. In the **Memory Setting** tab, set **Speed grade** to 3.
2. For **PLL reference clock frequency**, type **100** MHz to match the on-board oscillator.
3. For **Memory clock frequency**, type **400** MHz, the maximum frequency supported for DDR3 SDRAM interfaces on Stratix IV devices.
4. For **Memory Presets**, select **SD_PCIe_DDR3_Kit (4xMicron MT41J64M16LA-187E)**, which is a 64-bit wide 512-MB 400-MHz DDR3 unbuffered DIMM.

   The memory format in the **SD_PCIe_DDR3_Kit (4xMicron MT41J64M16LA-187E).xml** file is changed from discrete device to unbuffered DIMM to enable leveling functionality.

5. To create or modify a memory preset, click **Modify parameters**. In the **Preset Editor** dialog box, you can modify the memory presets.

   The DDR3 SDRAM has a write requirement ($t_{DQSS}$) that states the positive edge of the DQS signal on writes must be within ±25% (±90°) of the positive edge of the DDR3 SDRAM clock input. To achieve this skew requirement, the ALTMEMPHY-based designs always use the DDR IOE registers to generate the CK and CK# signals.
6. Turn on the **Enable Memory Chip Calibration in Timing Analysis** option on the **Advanced** page. This option is required for Stratix IV devices, which need post-processing script to remove timing model pessimism.
7. To specify the ODT settings for the DDR3 SDRAM interface on your board, in the **Preset Editor** dialog box, select **Memory Initialization Options**.
8. In the **Memory Initialization Options** dialog box, perform the following steps:
   a. For **Output driver impedance**, select **RZQ/7** (which is 34).
   b. For **Dynamic ODT (Rtt_WR) value**, select **RZQ/4** (which is 60).
   c. For **ODT Rtt nominal value**, select **RZQ/4** (which is 60).
   d. Click **OK** to apply the settings and exit the dialog box.
9. In the **PHY Settings** tab, turn on the **Enable dynamic parallel on-chip termination (OCT)** option for this example. The Stratix IV GX FPGA development kit does not include discrete external termination on the DQ, DQS, DQS#, or DM pins as the board was designed to use OCT.
10. Under **Address/Command Clock Setting**s, for **Dedicated clock phase** type **240**.

    The **Auto-Calibration Simulation Options** parameter is for RTL simulation only and is not applicable for gate-level simulation.

11. In the **Board Settings** tab, set the following **Slew Rates** and **Board Skews** parameters to the specified values:
    - **CK/CK# slew rate (Differential)** = **4 V/ns**
    - **Addr/Command slew rate** = **1.5 V/ns**
    - **DQS/DQS# slew rate (Differential)** = **3 V/ns**
    - **DQ slew rate** = **1.5 V/ns**

    ☞ These slew rates are obtained from simulation using the default I/O standard and drive options.

    - **Max skew within DQS group** = **0.015 ns**
    - **Max skew between DQS groups** = **0.128 ns**
    - **Addr/Command to CK skew** = **- 0.05 ns**

    The Intersymbol Interference (ISI) parameters are not applicable for single rank configurations. Set these parameters to **0 ns**.
12. In the **Controller Settings** tab, for **Controller Architecture** select **High Performance Controller II** for higher efficiency and advanced features.
13. Under **Efficiency**, select the specified values for the following options:
    a. For **Command Queue Look-Ahead Depth**, select **6**.
    b. For **Local-to-Memory Address Mapping**, select **CHIP-ROW-BANK-COL**.
    c. For **Local Maximum Burst Count**, select **4**.
14. Click **Next**.
15. Turn on the **Generate simulation model** option to generate the simulation model for the RTL simulation.
16. Click **Finish** to generate your MegaCore function variation. The MegaWizard Plug-In Manager generates all the files necessary for your DDR3 SDRAM controller, and an example top-level design, which you use to test or verify board operation.

For more information about the DDR3 SDRAM high-performance controller, refer to the *DDR3 SDRAM Controller with ALTMEMPHY IP User Guide* in volume 3 of the *External Memory Interface Handbook*.

# Add Constraints

After instantiating the DDR3 SDRAM high-performance controller, the ALTMEMPHY megafunction generates the constraints files for the design example. You must apply these constraints to the design before compilation.

## Set Top-Level Entity

The top-level entity of the project must be set to the correct entity. The naming convention for an ALTMEMPHY megafunction entity is <*variation_name*>**_phy.v** or **vhd**; an SDRAM high-performance controller entity is <*variation_name*>**.v** or **vhd**.

To set the top-level file, perform the following steps:

1. Open the entity file, <*variation_name*>**_example_top.v** or **vhd**.
2. On the Project menu, click **Set as Top-Level Entity**.

## Set Optimization Technique

To ensure the remaining unconstrained paths are routed with the highest speed and efficiency, perform the following steps to set the optimization technique:

1. On the Assignments menu, click **Settings**.
2. Select **Analysis & Synthesis Settings**.
3. Select **Speed** under **Optimization Technique**. Click **OK**.

## Set Fitter Effort

To set the fitter effort, perform the following steps:

1. On the Assignments menu, click **Settings**.
2. Select **Fitter Settings**.
3. Turn on **Optimize hold timing** and select **All Paths**.
4. Turn on **Optimize multi-corner timing**.
5. Select **Standard Fit (highest effort)** under **Fitter effort**.
6. Click **OK**.

## Add Timing Constraints

When you instantiate an SDRAM high-performance controller, it generates a timing constraints file, <*variation_name*>**_phy_ddr_timing.sdc**. The timing constraint file constrains the clock, input, and output delay on the SDRAM high-performance controller.

To add timing constraints, perform the following steps:

1. On the Assignments menu click **Settings**.
2. In the **Category** list, expand **Timing Analysis Settings**, and select **TimeQuest Timing Analyzer**.
3. Select the <*variation_name*>**_phy_ddr_timing.sdc** file and click **Add**.
4. Click **OK**.

## Add Pin and DQ Group Assignments

The pin assignment script, <*variation_name*>**_pin_assignments.tcl**, sets up the I/O standards for the DDR3 SDRAM interface. This script also launches the DQ group assignment script, <*variation_name*>**_phy_assign_dq_groups.tcl**, which relates the `DQ` and `DQS` pin groups together for the fitter to place them correctly in the Quartus II software.

This script does not create a clock for the design. You must create a clock for the design and provide pin assignments for the signals of both the example driver and testbench that the MegaCore variation generates.

To add the pin and I/O standards to the design example, perform the following steps:

1. On the Tools menu, click **Tcl scripts**.
2. Under **Libraries**, select <*variation_name*>**_pin_assignments.tcl**.
3. Click **Run**.

## Enter Pin Location Assignments

To enter the pin location assignments using the Pin Planner, perform the following steps:

1. Run analysis and synthesis. On the Processing menu, point to **Start** and click **Start Analysis & Synthesis**.
2. Assign all of your pins, so the Quartus II software fits your design correctly and performs correct timing analysis. To assign pin locations for the Stratix III memory demonstration kit, run the Altera-provided **S3_MB1_DDR3_PinLocations.tcl** file, and for the Stratix IV GX FPGA development kit, run the Altera-provided **SIV_DDR3×64_PinLocations.tcl** file, or manually assign the pin locations by using the Pin Planner.

   The SDRAM high-performance controller auto-generated scripts do not make any pin location assignments.

   If you are at the design exploration phase of your design cycle and do not have any PCB defined pin locations, you should manually define an initial set of pin constraints, which can become more specific during your development process.

To manually assign pin locations using the Pin Planner, perform the following steps:

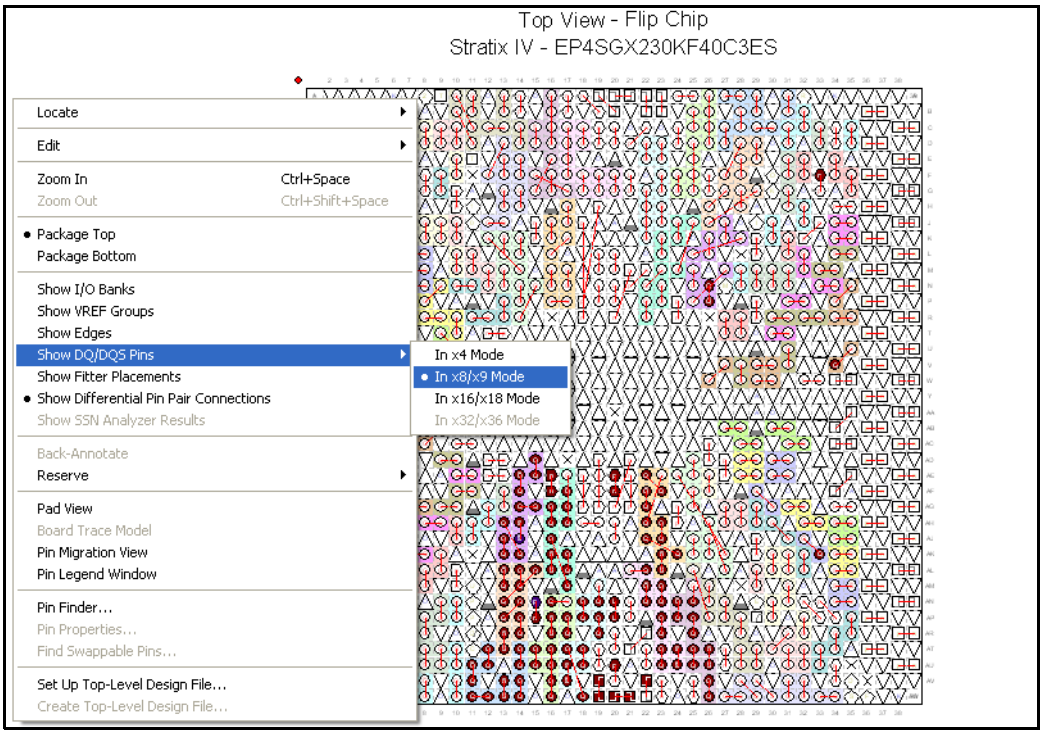
1. On the Assignments menu, click **Pin Planner**.
2. Assign `DQ` and `DQS` pins.
   a. To select the device DQS pin groups that the design uses, assign each `DQS` pin in your design to the required `DQS` pin in the Pin Planner. The Quartus II Fitter automatically assigns the respective `DQ` signals to suitable `DQ` pins within each group. To view the DQS groups in the Pin Planner, right-click, select **Show DQ/DQS Pins**, and click **In ×8/×9 Mode**. The Pin Planner shows each DQS group in a different color and with a different legend: S = `DQS` pin, Sbar = `DQSn` pin and Q = `DQ` pin, as shown in Figure 5–2.

   Most DDR3 SDRAM devices operate in ×8/×9 mode. However, some DDR3 SDRAM devices operate in ×4 mode. Refer to your specific memory device datasheet.

   b. Select the DQ mode to match the DQ group width (number of `DQ` pins/number of `DQS` pins) of your memory device. DQ mode is not related to the memory interface width.

DQ group order and DQ pin order within each group is not important. However, you must place the DQ pins in the same group as their respective strobe pin.

**Figure 5–2. Quartus II Pin Planner, Show DQ/DQS Pins, In x8/x9 Mode for Stratix IV**




3. Place the DM pins within their respective DQ group.
4. Place the address and control command pins on any spare I/O pins; ideally within the same bank or side of the device as the mem_clk pins.
5. Ensure that you place the mem_clk pins on differential I/O pairs for the CK/CK# pin pair. To identify differential I/O pairs, right-click in the Pin Planner and select **Show Differential Pin Pair Connections**. The pin pairs show a red line between each pin pair.

   You must place the mem_clk[0] and mem_clk_n[0] pins on an unused DQ or DQS pin pairs with DIFFIO_RX capability.

6. Place the clock_source pin on a dedicated PLL clock input pin with a direct connection to the SDRAM controller PLL and DLL pair—usually on the same side of the device as your memory interface. This recommendation reduces PLL jitter, saves a global clock resource, and eases timing and fitter effort.
7. Place the global_reset_n pin (like any high fan-out signal) on a dedicated clock pin.
8. Ensure that you place the $R_{UP}$ and $R_{DN}$ pins, termination_blk0~_rdn_pad and termination_blk0~_rup_pad, at locations within the same $V_{CCIO}$ voltage bank.

For more information about the Quartus II Pin Planner, refer to the *I/O Management* chapter in volume 2 of the *Quartus II Handbook*.

## Assign I/O Standards

To assign the I/O standards, perform the following steps:

1. On the Assignments menu, click **Assignment Editor**.
2. Specify **LVDS** as the I/O standard for `clock_source`.
3. Specify **2.5** V as the I/O standard for `global_reset_n`.

## Assign Virtual Pins

The example top-level design, which is auto-generated by the high- performance controller, includes an example driver to simulate the interface. This example driver is not part of the SDRAM high-performance controller IP, but allows easy testing of the IP.

The example driver outputs several test signals to indicate its operation and the status of the simulated memory interface. These test signals are `pnf`, `pnf_per_byte`, and `test_complete`. The test signals are not part of the memory interface, but are to facilitate testing. You must connect these signals to either a debug bus or assign the signals to virtual pins using the Quartus II Assignment Editor. When using the example driver for testing, do not remove the test signals from the top-level signal list. If you remove the test signals from the top-level module, the Quartus II software optimizes the driver away, and the example driver fails.

To assign virtual pin assignments for the Stratix III memory demonstration board, run the Altera-provided **s3_MB1_ddr3_exdriver_vpin.tcl** file, and for the Stratix IV GX FPGA development kit, run the Altera-provided **SIV_DDR3×64_exdriver_vpin.tcl** file, or manually assign the virtual pin assignments using the Assignment Editor.

The memory interface pins (`DQ`, `DQS`, `DM`, `CK`, `CK#`, address and command) cannot be assigned as virtual pins.

## Advanced I/O Timing

The ALTMEMPHY-based Stratix III designs assume that the memory address and command signals are length-matched to the memory clock signals. Typically, this length match is not true for DIMM-based designs. You must verify the difference in your design. To edit the TimeQuest **.sdc** file, <*variation name*>**_phy_ddr_timing.sdc**, to include this difference, perform the following steps:

1. Open the **ddr3_dimm_phy_ddr_timing.sdc** file in a text editor and find the following command line (usually line 31):

```
set t(additional_addresscmd_tpd) 0.000
```

   and change to the following command line:

```
set t(additional_addresscmd_tpd) -0.300
```

2. Save the file.

☞ If the DDR3 SDRAM controller **.sdc** file is regenerated, this change is lost and you must edit the file again.

## Enter Board Trace Delay Models

For accurate I/O timing analysis, the Quartus II software must include the board trace and loading information. This information should be derived and refined during your PCB development process of prelayout (line) simulation and final post-layout (board) simulation.

☞ For external memory interfaces that use memory modules (DIMMs), the board trace and loading information must also include the trace and loading information of the module. You can obtain these information from your memory vendor.

To enter board trace information, perform the following steps:

1. In the Pin Planner, select the pin or group of pins that you want to enter the information for.
2. Right-click and select **Board Trace Model**. Figure 5–3 shows the board trace model.

**Figure 5–3. Board Trace Model for Stratix IV**

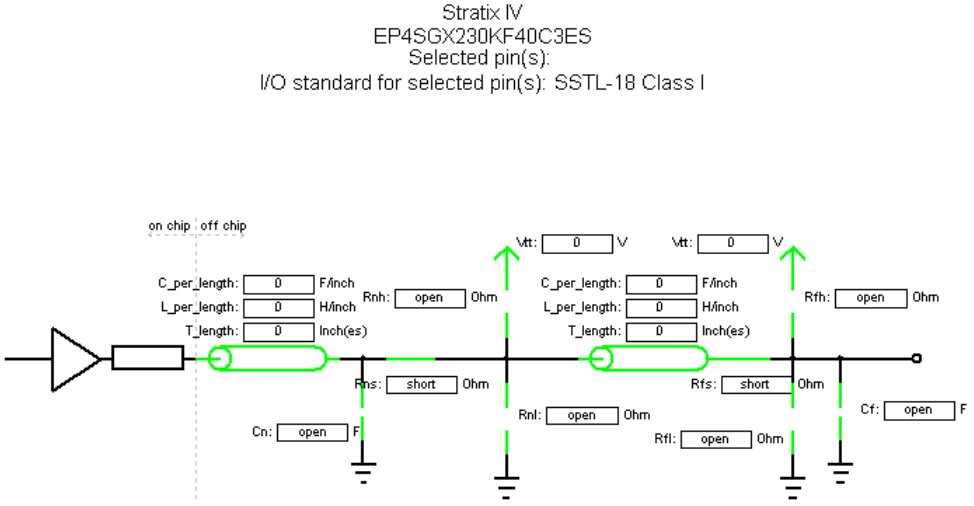


Table 5–1 shows the board trace model parameters for the Stratix III and Stratix IV GX development boards.

**Table 5–1. Stratix III and Stratix IV Development Board Trace Model Summary**

| Net | Near (FPGA End of Line) | | | | | | Far (Memory End of Line) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Length (Inch) | C_per_ length (pF/In) | L_per_ length (nH/In) | Cn (pF) | Rns | Rnh | Length (Inch) | C_per_ length (pF/In) | L_per_ length (nH/In) | Cf (pF) | Rfh/Rfp |
| Stratix III | | | | | | | | | | | |
| Addr *(1)* | 2.904 | 3.5 | 8.3 | — | — | — | 8.488 | 3.75 | 8.9 | 13.5 | 39 |
| CLK | 3.069 | 3.1 | 9.3 | 4.6 *(2)* | — | — | 8.488 | 3.75 | 8.9 | 7.2 | 36 |
| CKE/CS# | 2.937 | 3.5 | 8.3 | — | — | — | 8.480 | 3.75 | 8.9 | 13.5 | 39 |
| ODT | 2.853 | 3.5 | 8.3 | — | — | — | 8.480 | 3.75 | 8.9 | 13.5 | 39 |
| DQS0 | 2.905 | 3.5 | 8.3 | — | 15 | — | 0.661 | 3.0 | 10.7 | 3.0 | 60 |
| DQS1 | 2.973 | 3.5 | 8.3 | — | 15 | — | 0.780 | 3.0 | 10.7 | 3.0 | 60 |
| DQS2 | 2.893 | 3.5 | 8.3 | — | 15 | — | 0.913 | 3.0 | 10.7 | 3.0 | 60 |
| DQS3 | 2.778 | 3.5 | 8.3 | — | 15 | — | 1.106 | 3.0 | 10.7 | 3.0 | 60 |
| DQS4 | 2.877 | 3.5 | 8.3 | — | 15 | — | 1.051 | 3.0 | 10.7 | 3.0 | 60 |
| DQS5 | 2.936 | 3.5 | 8.3 | — | 15 | — | 0.870 | 3.0 | 10.7 | 3.0 | 60 |
| DQS6 | 3.072 | 3.5 | 8.3 | — | 15 | — | 0.728 | 3.0 | 10.7 | 3.0 | 60 |
| DQS7 | 3.080 | 3.5 | 8.3 | — | 15 | — | 0.665 | 3.0 | 10.7 | 3.0 | 60 |
| DQS8 | 2.708 | 3.5 | 8.3 | — | 15 | — | 0.661 | 3.0 | 10.7 | 3.0 | 60 |
| Stratix IV | | | | | | | | | | | |
| Addr *(1)* | — | — | — | — | — | — | 3.568 | 4.2 | 7.6 | 5.2 | 56 |
| CLK | — | — | — | — | — | — | 3.73 | 4.1 | 7.8 | 5.2 | 100 |
| CKE/CS# | — | — | — | — | — | — | 3.568 | 4.2 | 7.6 | 5.2 | 56 |
| ODT | — | — | — | — | — | — | 3.51 | 4.2 | 7.6 | 5.2 | 56 |
| DQS | — | — | — | — | — | — | 2.3 | 4.1 | 7.8 | 2.5 | 60 |

**Notes to Table 5–1:**

(1) Addr = Addr, ba, we#, ras#, odt, and cas.

(2) Cn value of 4.6 pF comprises 7 pF on memory demonstration board plus 2.2 pF on the DIMM, which is 9.2 pF differential or 4.6 pF SE.

Altera recommends you to use the **Board Trace Model** assignment for all DDR3 SDRAM interface signals. To apply the board trace model assignments for the Stratix III memory demonstration board, run the Altera-provided **S3_MB1_DDR3_BTModels.tcl** script, and for the Stratix IV GX FPGA development kit, run the Altera-provided **SIV_DDR3×64_BoardTraceModels.tcl** script, or manually assign the virtual pin assignments using the Quartus II Pin Planner.

The Stratix III demonstration board has the 7 pF (differential) compensation capacitors fitted to its DDR3 SDRAM CLK and CLK# signals. These capacitors are typically fitted to designs that use asymmetric DIMM designs. Simulate your design to check if the compensation capacitors are required. Stratix III devices have various programmable drive strength and OCT I/O options, so compensation capacitors are not usually required. Because fitting compensation capacitors reduces the edge rate of your signals, you should observe the memory vendor derating guidelines.

For more information on compensation capacitors, refer to *Micron Technical Note TN_47_01*.

# Perform RTL or Functional Simulation (Optional)

After instantiating the DDR3 SDRAM high-performance controller, the MegaWizard Plug-In Manager generates a design example that includes a driver, a test bench, and a memory model that allows you to perform functional simulation on your design.

The Verilog HDL or VHDL simulation model of the PHY, <*variation_name*>**_alt_mem_phy_sequencer_wrapper.vo/.vho** file, is located in your project directory.

To run the simulation with NativeLink, perform the following steps:

1. Set the absolute path to your third-party simulator executable file by performing the following steps:
   a. On the Assignments menu, click **EDA Tool Settings**.
   b. In the **Category** list, expand **EDA Tool Settings** and click **Simulation**.
   c. Under **Tool name**, select **ModelSim-Altera**.
   d. Under **NativeLink settings**, select **Compile test bench** and click **Test Benches**.
   e. Click **New**.
   f. Type the name of your testbench top-level module and simulation period.
   g. Click **OK**.
2. To elaborate your design, on the Processing menu, point to **Start** and click **Start Analysis & Elaboration**.
3. On the Tools menu, point to the **Run EDA Simulation Tool** and click **EDA RTL Simulation**. This step creates the **\simulation** directory in your project directory and a script that compiles all necessary files and runs the simulation.

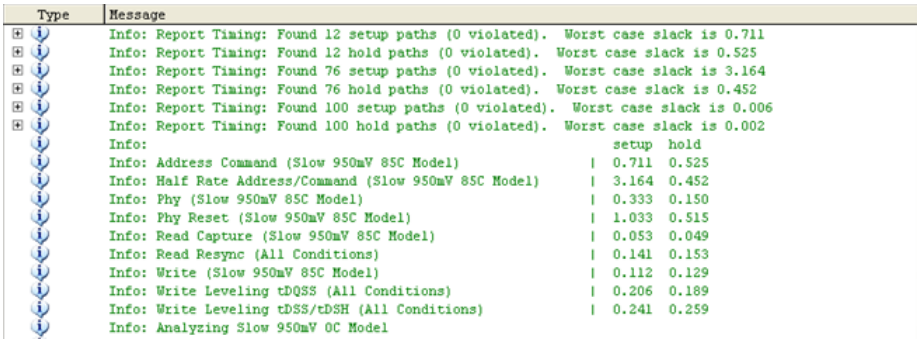
For example timing diagrams, refer to the *DDR3 SDRAM Controller with ALTMEMPHY IP User Guide* in volume 3 of the *External Memory Interface Handbook*.

# Compile Design and Verify Timing

To compile the design, on the Processing menu, click **Start Compilation**. After successfully compiling the design, the Quartus II software automatically runs the verify timing script, <*variation_name*>**_timing.tcl**, which produces a timing report for the design together with the compilation report.

Figure 5–4 shows the timing margin report in the message window in the Quartus II software.

**Figure 5–4. Timing Margin Report in the Quartus II Software**

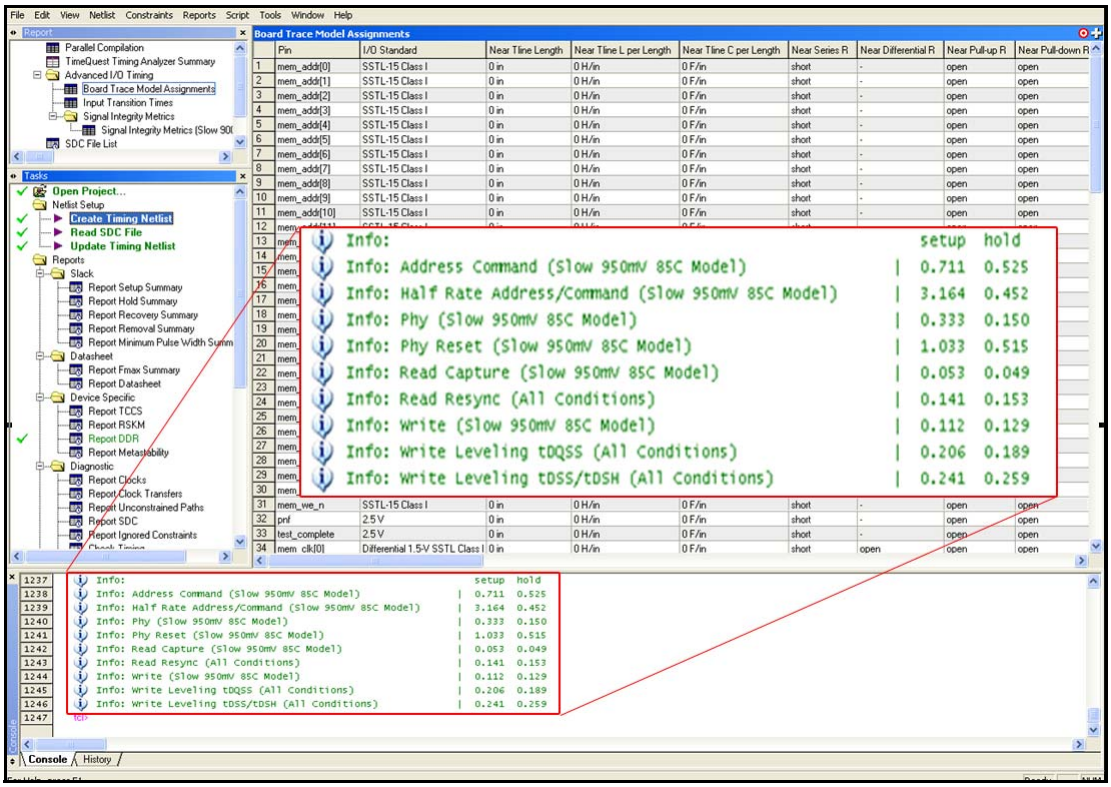



You can also obtain the timing report by running the report timing script, <*variation_name*>**_timing.tcl**, in the TimeQuest Timing Analyzer window. To obtain the timing report in the TimeQuest Timing Analyzer window, perform the following steps:

1. On the Tools menu, click **TimeQuest Timing Analyzer**.
2. On the **Tasks** pane, double-click on **Report DDR** to run **Update Timing Netlist**, **Create Timing Netlist,** and **Read SDC File**. This command subsequently executes the report timing script to generate the timing margin report.

Figure 5–5 shows the timing margin report in the TimeQuest Timing Analyzer window after running the report timing script. The results are the same as those obtained from the Quartus II software directly.

**Figure 5–5. Timing Margin Report in TimeQuest Timing Analyzer**






You must verify the timing at every corner of the timing model. You must run the timing report script with all available timing models—slow 0°C, slow 85°C, and fast 0°C—to ensure positive margins across the process, voltage, and temperature variations. To analyze timing on a different corner, double-click **Set Operating Conditions** in the left pane. Select a new timing corner, then go to the Script menu and rerun the <*variation_name*>**_report_timing.tcl** script.

The Stratix IV devices need post-processing script to remove timing model pessimism on the write and read capture path margins, refer to Figure 5–6 and Figure 5–7.

**Figure 5–6. Write Margin Summary**

| | Operation | Setup Slack | Hold Slack |
|---|---|---|---|
| 1 | ⊟ Standard Write | -0.044 | -0.025 |
| 2 | Spatial correlation pessimism removal | 0.065 | 0.065 |
| 3 | More clock pessimism removal | 0.096 | 0.094 |
| 4 | Write | 0.117 | 0.134 |

**Figure 5–7. Read Capture Path Margin Summary**

| | Operation | Setup Slack | Hold Slack |
|---|---|---|---|
| 1 | ⊟ Standard Read Capture | 0.011 | 0.007 |
| 2 | Spatial correlation pessimism removal | 0.047 | 0.047 |
| 3 | Read Capture | 0.058 | 0.054 |

The standard write and read capture margins are initially calculated using the FPGA timing model and adjusted to account for the effects not modeled by either the timing model or TimeQuest timing analyzer. These effects include memory calibration, deskew topologies, quantization error and calibration uncertainties.

The final write and read capture margins are summarized in the Report DDR margin summary, refer to Figure 5–8.

**Figure 5–8. Report DDR Margin Summary**

| | Path | Operating Condition | Setup Slack | Hold Slack |
|---|---|---|---|---|
| 1 | Address Command (Slow 950mV 85C Model) | Slow 950mV 85C Model | 0.717 | 0.528 |
| 2 | Half Rate Address/Command (Slow 950mV 85C Model) | Slow 950mV 85C Model | 3.170 | 0.444 |
| 3 | Phy (Slow 950mV 85C Model) | Slow 950mV 85C Model | 0.333 | 0.121 |
| 4 | Phy Reset (Slow 950mV 85C Model) | Slow 950mV 85C Model | 1.042 | 0.519 |
| 5 | Read Capture (Slow 950mV 85C Model) | Slow 950mV 85C Model | 0.058 | 0.054 |
| 6 | Read Resync (All Conditions) | All Conditions | 0.142 | 0.154 |
| 7 | Write (Slow 950mV 85C Model) | Slow 950mV 85C Model | 0.117 | 0.134 |
| 8 | Write Leveling tDQSS (All Conditions) | All Conditions | 0.206 | 0.189 |
| 9 | Write Leveling tDSS/tDSH (All Conditions) | All Conditions | 0.241 | 0.259 |

For more information about the TimeQuest timing analyzer, refer to *The Quartus II TimeQuest Timing Analyzer* chapter in volume 7 of the *Quartus II Handbook*. For information about timing analysis, refer to the *Timing Analysis* section in volume 4 of the *External Memory Interface Handbook*.

# Determine Board Design Constraints and Perform Board-Level Simulations

The Stratix III and Stratix IV devices support both series and parallel OCT resistors to improve signal integrity. The Stratix III and Stratix IV OCT resistors also eliminate the need for external termination resistors on the FPGA. This feature simplifies board design and reduces overall board cost. The series ODT features are available in settings of 34 Ω and 40 Ω. Although 40 Ω is not supported by all vendors.

All DDR3 SDRAM interfaces use the following two classes of signal type:

- Unidirectional class I terminated signals, which include clocks, and address and command signals.
- Bidirectional class II terminated signals, which include `DQS`, `DQ`, and `DM` signals.

For bidirectional signals, Class II termination is recommended to ensure proper termination for the following operation:

- Reads from the memory component, termination at the Stratix III or Stratix IV GX FPGA side.
- Writes to the memory component, termination at the memory side.

The Stratix III and Stratix IV devices include on-chip series and parallel termination. So generally, discrete termination at the FPGA end of the line is not required.

The DDR3 SDRAM devices support dynamic parallel ODT at the memory end of the line. So typically, discrete termination is not required. For these reasons, Class I termination are used for bidirectional signals in this tutorial.

To understand better about the different effects on signal integrity design, refer to the *Board Layout Guidelines* section in volume 2 of the *External Memory Interface Handbook*. For more information about the signal integrity of the Stratix III and Stratix IV devices, refer to Stratix III Device Signal and Power Integrity and Stratix IV FPGA Signal and Power Integrity pages.

# Verify Design on a Board

The SignalTap II Embedded Logic Analyzer shows read and write activity in the system.

For more information about using the SignalTap II Embedded Logic Analyzer, refer to the For more information about using the SignalTap II Embedded Logic Analyzer, refer to the *Design Debugging Using the SignalTap II Embedded Logic Analyzer* chapter in the *Quartus II Handbook*, *AN 323: Using SignalTap II Embedded Logic Analyzers in SOPC Builder Systems* and *AN 446: Debugging Nios II Systems with the SignalTap II Logic Analyzer*.

To add the SignalTap II Embedded Logic Analyzer, perform the following steps:

1. On the Tools menu, click **SignalTap II Logic Analyzer**.
2. Under **Signal Configuration**, next to the **Clock** box click **…** (**Browse Node Finder**).

3. Type `*phy_clk` in the **Named** box, for **Filter** select **SignalTap II: pre-synthesis** and click **List**.
4. Select **ddr3_dimm | ddr3_dimm_inst | phy_clk** for a Stratix III design, or **ddr3_bot_x64_example_top | ddr3_bot_x64:ddr3_bot_x64 | phy_clk** for a Stratix IV design, in **Nodes Found** and click **>** to add the signal to **Selected Nodes**.
5. Click **OK**.
6. Under **Signal Configuration**, specify the following settings:
   - For **Sample depth**, select **512**
   - For **RAM type**, select **Auto**
   - For **Trigger flow control**, select **Sequential**
   - For **Trigger position**, select **Center trigger position**
   - For **Trigger conditions**, select **1**
7. On the Edit menu, click **Add Nodes**.
8. Search for specific nodes by typing `*local*` in the **Named** box, for **Filter** select **SignalTap II: pre-synthesis** and click **List**.
9. Select the following nodes in **Nodes Found** and click **>** to add to **Selected Nodes**:
   - **local_address**
   - **local_rdata**
   - **local_rdata_valid**
   - **local_read_req**
   - **local_ready**
   - **local_wdata**
   - **local_wdata_req**
   - **local_write_req**
   - **pnf**
   - **pnf_per_byte**
   - **test_complete** (trigger)
   - **ctl_cal_success**
   - **ctl_cal_fail**
   - **ctl_wlat**
   - **ctl_rlat**

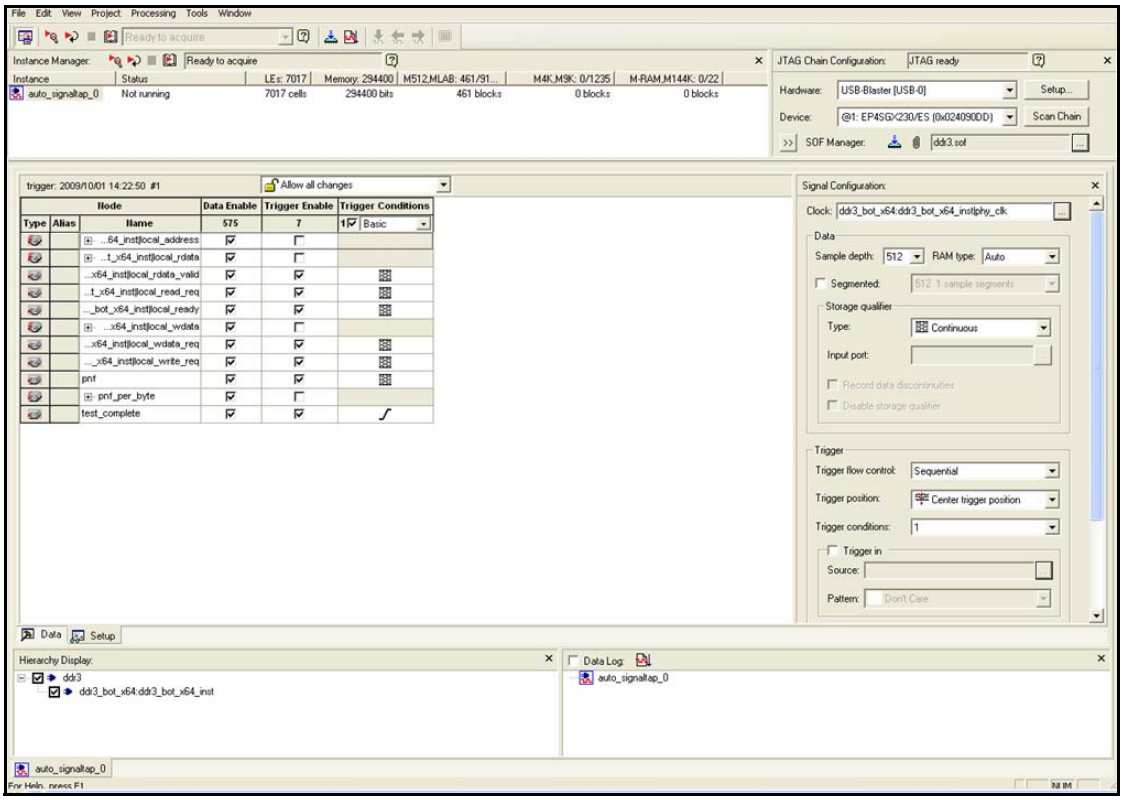
   ☞ Do not add any DDR3 SDRAM interface signals to the SignalTap II Embedded Logic Analyzer. The load on these signals increases and adversely affects the timing analysis.

10. Click **OK**.

11. To reduce the SignalTap II logic size, turn off **Trigger Enable** on the following bus signals:
    - **local_address**
    - **local_rdata**
    - **local_wdata**
    - **pnf_per_byte**
    - **ctl_wlat**
    - **ctl_rlat**
12. Right-click **Trigger Conditions** for the `test_complete` signal and select **Rising Edge**.

    Figure 5–9 shows the completed SignalTap II Embedded Logic Analyzer.

**Figure 5–9. SignalTap II Embedded Logic Analyzer**




13. On the File menu, click **Save**, to save the SignalTap II **.stp** file to your project.

    If you see the message **Do you want to enable SignalTap II file "stp1.stp" for the current project**, click **Yes**.

## Compile the Project

After you add signals to the SignalTap II Embedded Logic Analyzer, to recompile your design, on the Processing menu, click **Start Compilation**.

## Verify Timing

After the design compiles, ensure that TimeQuest timing analysis passes successfully. In addition to this FPGA timing analysis, check your PCB or system SDRAM timing. To run timing analysis, perform the following steps to run the <*variation name*>**_phy_report_timing.tcl** script:

1. On the Tools menu, click **Tcl Scripts**.
2. Select <*variation name*>**_phy_report_timing.tcl**. \
3. Click **Run**.

## Download the Object File

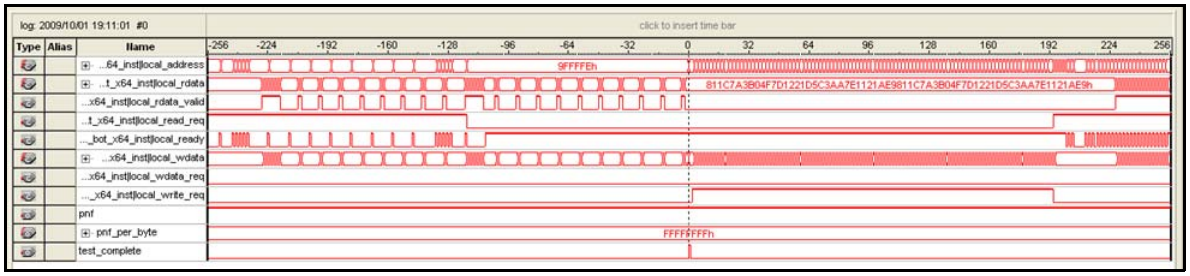
To download the object file to the device, perform the following steps:

1. Connect the development board to your computer.
2. On the Tools menu, click **SignalTap II Logic Analyzer**. The SignalTap II dialog box appears.
3. Click **...** to open the **Select Program Files** dialog box.
4. Select <*your project name*>**.sof**.
5. Click **Open**.
6. To download the file, click the **Program Device** button.

## Test the Design Example in Hardware

When the design example including SignalTap II successfully downloads to your development board, click **Run Analysis** to run once, or click **Autorun Analysis** to run continuously. Figure 5–10 shows the design analysis.

**Figure 5–10. SignalTap II Example DDR3 SDRAM Design Analysis**



# 6. Using High-Performance DDR, DDR2, and DDR3 SDRAM with SOPC Builder

The Altera DDR, DDR2, and DDR3 SDRAM Controllers with ALTMEMPHY IP version 9.1 support SOPC Builder, enabling you to instantiate a DDR, DDR2, or DDR3 SDRAM Controller with ALTMEMPHY IP in an SOPC Builder system.

This walkthrough discusses the following topics:

- Consider SOPC Builder system interconnect fabric and performance implications:
  - High-performance controller (HPC) or high-performance controller II (HPC II)
  - Full- or half-rate SDRAM high-performance controller
  - System and component clock selection, and half-rate bridges
  - Burst reads and writes
  - Latency and how to optimize read or write addressing
- Implement a DDR2 SDRAM high-performance controller in SOPC Builder
- Incorporate a Nios® II Processor and other peripherals
- Compile the design and generate the programming file
- Download the design to a development board and run sample code on your design to test read and write transactions

The design in this walkthrough ensures an optimum SOPC Builder Avalon-MM architecture and demonstrates how a simple Nios II C program, when combined with a SignalTap II Embedded Logic Analyzer, can verify both hardware and software operation.

For more information on functional simulation of an SOPC Builder system, refer to *AN 351: Simulating Nios II Embedded Processor Designs*. For more information about the DDR, DDR2, and DDR3 SDRAM high-performance controllers, refer to the *DDR and DDR2 SDRAM Controller with ALTMEMPHY IP User Guide* section and the *DDR3 SDRAM Controller with ALTMEMPHY IP User Guide* section in volume 3 of the *External Memory Interface Handbook*.

## SOPC Builder System Considerations

Always consider the following caveats and limitations when integrating a DDR, DDR2, or DDR3 SDRAM high-performance controller in SOPC Builder:

- High-Performance Controller (HPC) or High-Performance Controller II (HPC II)
- Full- or Half-Rate SDRAM High-Performance Controller
- Clock Selection and Clock Crossing Bridges
- Burst Reads and Writes
- Multimasters
- Direct Memory Access (DMA) Controller

- Read and Write Addressing and Latency

## High-Performance Controller (HPC) or High-Performance Controller II (HPC II)

Select either HPC or HPC II based on your system design. HPC II is an enhanced version of HPC with higher efficiency, and provides the following features:

- Supports higher efficiency look-ahead bank management with in-order read and write, out of order management.
- Supports bank interleaving with additive latency and auto precharge to provide higher efficiency.
- Provides optional run-time programmability to configure the behavior of the controller such as memory timing, size and mode register settings, allowing you to reconfigure and read the controller parameters in user mode, and reduce compilation time.
- Has half-bridge integrated in the controller to reduce the memory access latency.
- Supports multi-cast write to mitigate $t_{RC}$.
- Has integrated flexible built-in burst adapter to automatically split or merge user burst request to match memory's native burst length, allowing you to set a maximum of 64 beats at the local side.
- Supports integrated error correction coding (ECC), which supports 40-bit and 72-bit interfaces with sub-word writes, and optional automatic write-back on error.
- Supports Avalon-MM interface.

For more information about HPC II, the *DDR and DDR2 SDRAM Controller with ALTMEMPHY IP User Guide* section and the *DDR3 SDRAM Controller with ALTMEMPHY IP User Guide* section in volume 3 of the *External Memory Interface Handbook*.

Altera recommends that you use the HPC II, which provides more flexibility and higher efficiency.

## Full- or Half-Rate SDRAM High-Performance Controller

Full- or half-rate SDRAM high-performance controllers have the following definitions:

- Full-rate controllers present data to the local interface at twice the width of the actual SDRAM interface at the full SDRAM clock rate.
- Half-rate controllers present data to the local interface at four times the width of the actual SDRAM interface at half the SDRAM clock rate.

Implementing the SDRAM high-performance controllers in half-rate mode gives the highest possible SDRAM clock frequency while allowing the more complex core logic to operate at half this frequency. You can simplify the complexity of your design by permitting your Nios II processor to run at the slower, half-rate memory speed while still achieving the required SDRAM bandwidth per I/O pin.

However, where possible it is generally more optimal to configure the controller in full-rate mode with the core operating at the same clock frequency as your SOPC Builder system.

## Full-Rate Versus Half-Rate Command Operation

Commands can be slower using a half-rate controller. For example, a DDR SDRAM device can have a number of banks open at once. Each bank has a currently selected row. Changing the column within the selected row of an open bank requires no additional bank management commands to be issued. Changing the row in an active bank, or changing the bank both incur a protocol penalty that requires the precharge (PCH) command closes the active row or bank, and the active (ACT) command then opens or activates the new row or bank combination.

The duration of this penalty is a function of the controller clock frequency, the memory clock frequency, and the memory device characteristics. Calculating the impact of a change of memory and controller configuration on a given system is not a trivial task, as it depends on the nature of the accesses that are performed.

In this example each command takes a single clock cycle in a full-rate controller, but two clock cycles in a half-rate controller. The bank is not available for the subsequent ACT command until ($t_{RP}$) after the PCH. So the issuing of commands can be slower using a half-rate controller, even if the respective memory timing parameters remain the same.

## Time-Specified Memory Parameters

For a half-rate SDRAM high-performance controller, the control circuitry is clocked at half rate and so control operations are slower than in full-rate mode. However, the memory's clock frequency and physical properties are not affected.

When you use half-rate mode, any time-specified memory parameters in the controller are modified.

For example, if:

$t_{RCD}$min = 20 ns

For a 133-MHz controller:

$t_{CK}$ = 7.5 ns

20/7.5 = 2.666 rounded up to 3 clock cycles (22.5ns).

For a half-rate 66-MHz controller:

$t_{CK}$ = 15 ns

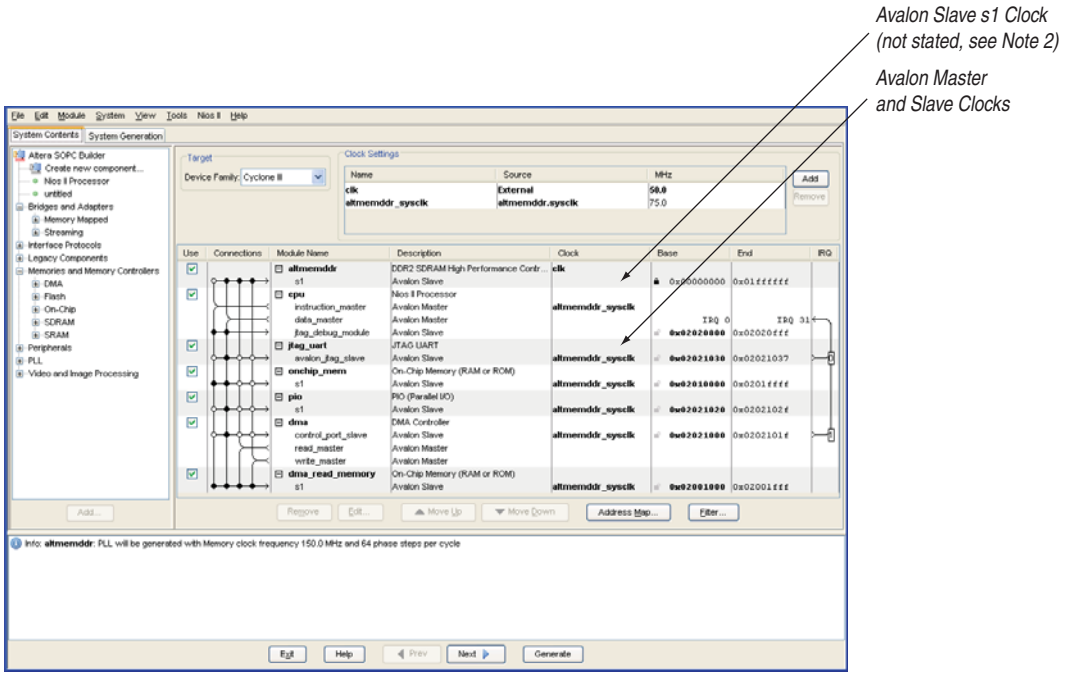
20/15 = 1.33 rounded up to 2 clock cycles (30ns).

Thus bank and row changes are slower in half-rate mode, but are not twice as slow. The easiest way to measure this effect for your chosen memory device and interface clock speed is to simulate both half-rate and full-rate designs and record the increased latency when switching rows. Typically a full-rate controller is around 14% more efficient when switching rows within a bank. As a half-rate controller has less read latency and if the masters in your system do not cause bank switching to occur often, the half-rate controller still gives higher performance. In SOPC Builder, you can alter arbitration shares to prevent masters from switching memory banks, thus creating a more optimal system.

## Clock Selection and Clock Crossing Bridges

Ideally every component in the SOPC Builder system should be clocked using the same clock, to prevent SOPC Builder automatically adding clock domain crossing logic, which adds latency. The SDRAM high-performance controllers already include a PLL, so you should set the SOPC Builder system clock to `altmemddr.sysclk`, which is the clock the controller and local interface logic use. SOPC Builder only shows the input clock to the controller PLL, not the interface clock, refer to Figure 6–1 on page 6–4.

**Figure 6–1. SOPC Builder Avalon-MM Slave and Master Clock Selection *(Note 1)* and *(2)***




**Notes to Figure 6–1:**

(1) The `clk` clock is the input clock to the controller PLL.

(2) The `s1` clock is not stated, but is on the `altmemddr.sysclk` clock domain, not on the `clk` domain.

If a different Avalon-MM clock is specified for connected components and a clock crossing bridge is not used, SOPC Builder automatically adds clock crossing adapters between any data masters of your SOPC Builder system and the SDRAM high-performance controller slave interface. Clock crossing adapters provide robust and safe transactions between different clock domains, however they increase latency and limit total bandwidth.

To prevent SOPC Builder from auto-inserting clock adapters:

- If the connected SOPC Builder components and SDRAM high-performance controller operate at the same frequency, use `altmemddr.sysclk` as the clock for the rest of your system. No clock domain crossing adapters are added by SOPC Builder.

- If the connected SOPC Builder components and SDRAM high-performance controller operate at different frequencies, manually insert an Avalon-MM clock crossing bridge between the SDRAM high-performance controller and the other SOPC Builder components.
- If the connected SOPC Builder components and SDRAM high-performance controller operate at different frequencies but are in the same clock phase, manually insert an Avalon-MM DDR memory half-rate bridge between the SDRAM high-performance controller and the other SOPC Builder components. Alternatively, turn on the **Enable Half Rate Bridge** option on the **Memory Settings** tab to use the embedded half-rate bridge in HPC II.

To use the Avalon-MM clock crossing bridge or the Avalon-MM DDR memory half-rate bridge, set the slave clock to your SOPC Builder system clock, and the master clock to `altmemddr.sysclk`.

For more information about the Avalon-MM clock crossing bridge and Avalon-MM DDR memory half-rate bridge, refer to the *Avalon Memory-Mapped Bridges* chapter in volume 4 of the *Quartus II Handbook*.

The Avalon-MM clock crossing bridge also works when you are using two different clocks at the same frequency but with different phases. However, the Avalon-MM DDR memory half-rate bridge must meet the following requirements:

- The clock frequency of the memory-side master must be in the same phase and twice of the processor-side slave frequency.
- The memory-side master is half as wide as the processor-side slave.

Use either the Avalon-MM clock crossing bridge or the Avalon-MM DDR memory half-rate bridge to allow a full-rate controller to connect to a half-rate SOPC builder design. As the memory controller still operates at full rate, twice the Nios II frequency, the memory interface commands operate at the faster rate.

The Avalon-MM DDR memory half-rate bridge has the same functionality as the Avalon-MM clock crossing bridge but provides lower latency. Altera recommends you use the Avalon-MM DDR memory half-rate bridge for Nios II Processors that require low latency access to high-speed memory.

Figure 6–2 shows a block diagram of how to use an Avalon-MM clock crossing bridge or an Avalon-MM DDR memory half-rate bridge.

**Figure 6–2. Clock Crossing Bridge**

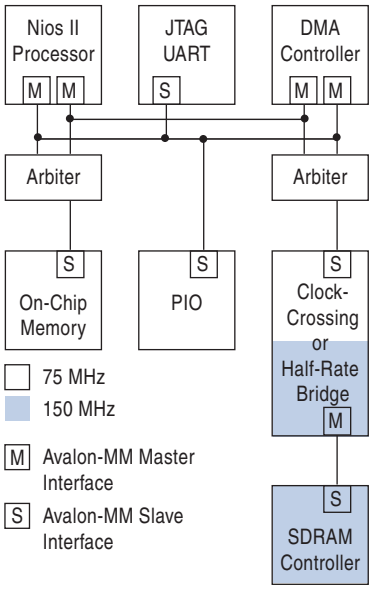
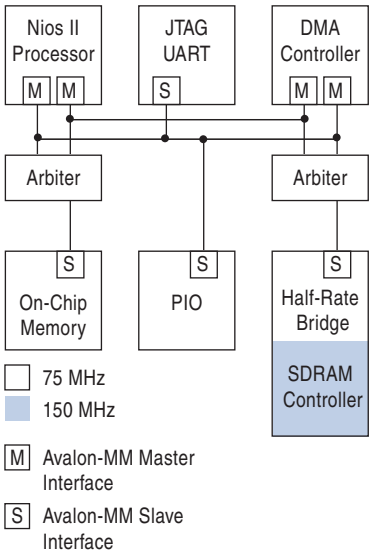



Figure 6–3 shows a block diagram of how a half-rate bridge is embedded in HPC II.

**Figure 6–3. Half-Rate Bridge in DDR SDRAM High-Performance Controller II**




## Burst Reads and Writes

Burst reads and writes differ in the following ways:

- In half-rate designs, the Avalon-MM slave interface is four times the width of the SDRAM device. Hence four transactions are performed on the SDRAM for every single Avalon-MM transaction. Avalon-MM burst requests on the local side serve no purpose as the memory interface is already using the maximum supported memory burst size for every single Avalon-MM transaction.
- In full-rate designs, you can use Avalon-MM bursts of one or two with the SDRAM high-performance controller, as each Avalon-MM transaction results in only two SDRAM transactions. So two Avalon-MM burst transactions may be combined to support the four supported on the SDRAM high-performance controller.

When a burst capable master supports larger burst lengths than the slave, SOPC Builder automatically places a burst length adapter into the path.

To obtain performance advantages, ensure that the data widths of master and slave pairs are matched in SOPC Builder. Whenever a master port is connected to a slave port of a different width, SOPC Builder automatically inserts adapter logic to convert between the different data widths.

Ideally, the data size of an Avalon-MM interface is 32 bits for a Nios II processor:

- For a full-rate SDRAM high-performance controller, the double-data rate stage doubles the data width, thus a 16-bit external memory width is best.
- For a half-rate SDRAM high-performance controller, the double-data rate and half-rate stages both double the data width, thus an 8-bit external memory width is best.

Ideally, match the following data cache line sizes to the memory controller burst length:

- 4-byte line = bursts of 1 (32-bit word)
- 16-byte line = bursts of 4
- 32-byte line = bursts of 8

You can set the data bus arbitration priority to avoid using the burst signal.

Table 6–1 and Table 6–2 show the burst length support for each type of DDR SDRAM HPC and HPC II.

**Table 6–1. Burst Length Support for DDR SDRAM HPC**

| Memory Type | Full-Rate Mode | Half-Rate Mode |
|---|---|---|
| DDR3 | — | 8 |
| DDR2 | 4 | 4 |
| DDR | 2 or 4 | 4 |

**Table 6–2. Burst Length Support for DDR SDRAM HPC II**

| Memory Type | Full-Rate Mode | Half-Rate Mode |
|---|---|---|
| DDR3 | — | 8 |
| DDR2 | 4 | 8 |
| DDR | 4 | 8 |

HPC II consists of a built-in burst adapter that automatically splits or merges bust request to match the memory's native burst length. This built-in adapter allows the Nios II processor to request any burst length and maps the burst length to the most efficient memory burst. HPC II accepts up to 64 burst count.

## Multimasters

To achieve best throughput and latency of the SDRAM, you should connect the controller to the smallest number of masters and share those masters with the smallest number of slaves. Fewer connections reduce the complexity of the SOPC Builder auto-inserted data multiplexers and increase the $f_{MAX}$ of the Avalon-MM interface.

If the Avalon-MM interface $f_{MAX}$ becomes the limiting factor, insert pipeline bridges to increase $f_{MAX}$ the (at the expense of latency).

Master and slave pairs are locked for the whole duration of a burst—no other master is granted access to the slave target of a burst until the burst is completed. You should isolate critical memory connections from the rest of the system.

One master may lock an SDRAM high-performance controller until a write burst is completed, which may take several cycles of time, during which the SDRAM high-performance controller cannot accept any other request. The write burst command is complete when all burst data is passed to the SDRAM high-performance controller. For a read burst, the controller quickly transfers back the whole burst at full bandwidth and becomes immediately available for new requests.

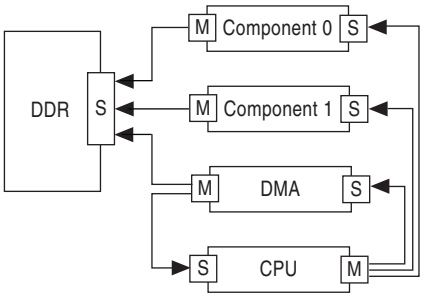
## Direct Memory Access (DMA) Controller

The DMA controller enables the memory data transfer directly without being preformed through the processor to achieve better performance and parallelism between operations. By enabling the direct transfer, the processor can perform other tasks in parallel.

For more information, refer to the *DMA Controller Core* chapter in volume 5 of the *Quartus II Handbook*.

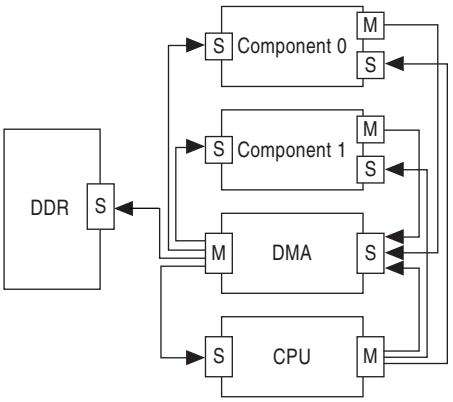
The DMA controller can perform bulk data transfers, reading data from a source address range and writing data to a different address range. The DMA controller boosts the memory band-width and alleviates the multimaster bottleneck. With the use of a DMA controller, you can reduce the number of masters that access the memory. Figure 6–4 shows an example where there are multiple masters access to the memory. This increases the complexity of the connection and latency to the system.

**Figure 6–4. Multiple Masters Accessing a Single Memory**




You can use the DMA controller to reduce the bottleneck for the multimaster access to the memory. The DMA controller acts as the only master that accesses the memory but as a slave for other components that wants access to the memory. DMA controller uses interrupt request to notify the masters of other components that the transfers are completed without wasting time with polling. Figure 6–5 shows an example of how DMA controller is used to enhance the memory bandwidth for multimaster.

**Figure 6–5. A Single Master Accessing Memory with DMA Controller**




## Read and Write Addressing and Latency

Systems with deterministic access patterns can minimize the number of bank and row changes. For example, by suitably arranging the memory map. In systems with more random access patterns (often typical in embedded SOPC Builder-type systems), minimizing bank and row changes is more difficult and the increased latency (by constantly changing the row in an active bank, or changing the bank) has a greater effect. Non-optimal cache implementations can waste cycles with half-rate controllers.

You should always consider the following actions:

- Match controller Avalon-MM interface width to Avalon-MM master interface width.
- Minimize non-sequential addressing to reduce row addressing time.
- Match the Nios II cache line size to memory burst length.
- Set arbitration priorities correctly.
- Insert bridges to increase $f_{MAX}$ at the expense of latency.
- Minimize multimaster designs where arbitration stalls may take place.

In multi-mastering designs, the memory is not available to all masters concurrently.

Setting the arbitration priorities correctly prevents unnecessary memory accesses. For example, always set the Nios II instruction master arbitration priority to eight, because it always tries to access eight sequential addresses. Set the data master arbitration priorities depending on the cache line size; do not leave the arbitration priority value as default.

# DDR2 SDRAM Controller with ALTMEMPHY IP with SOPC Builder Walkthrough

This walkthrough shows how to use the SOPC Builder design flow to design a 8-bit wide, 150-MHz, 300-Mbps DDR2 SDRAM interface. The design example also provides some recommended settings to simplify the design. Although the design example is specifically for the DDR2 SDRAM interface, the design flow for DDR and DDR3 SDRAM interfaces are the same.

The design example targets the Cyclone III FPGA development kit with four MT47H32M16CC-3 components and an 8-bit wide 512-MB Micron MT47H32M16CC-3 333-MHz DDR2 SDRAM component.

## System Requirement

This application note assumes that you are familiar with the Quartus II software and SOPC Builder. This tutorial requires the following hardware and software:

- Quartus II software version 9.1
- ModelSim-SE 6.5b or higher
- DDR2 SDRAM Controller with ALTMEMPHY IP version 9.1
- Nios II Embedded Design Suite (EDS) version 9.1

The design is targeted to the Cyclone III FPGA development kit. You can target other supported device families or development kits.

For more information on the Cyclone III FPGA development kit, refer to www.altera.com/products/devkits/altera/kit-cyc3.html.

# Create Your Example Project

This section shows you how to create a new Quartus II project and an SOPC Builder system.

## Create a New Quartus II Project

In the Quartus II software, create a new project with the **New Project Wizard**, ensure that the device type is set to Cyclone III, EP3C120F780C7.

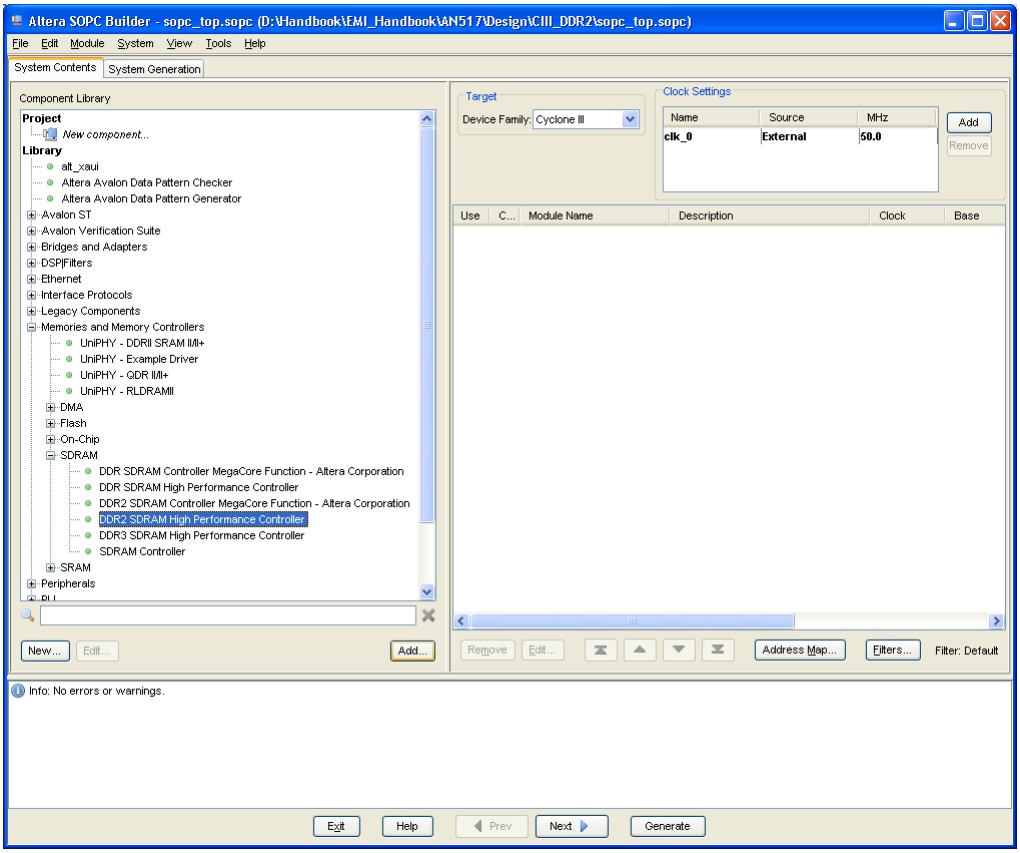
Ensure that your project path does not include any spaces or extended characters.

## Create the SOPC Builder System

To create an SOPC Builder system, perform the following steps:

1. On the Tools menu, click **SOPC Builder**.
2. In the **Create New System** dialog box, type `sopc_top` for the **System Name**. In the **Target HDL**, select **Verilog**, then click **OK**.
3. In the **System Contents** tab, under **Component Library**, expand **Memories and Memory Controllers**. Expand **SDRAM**. Select **DDR2 SDRAM High Performance Controller** and click **Add**. The DDR2 SDRAM Controller with ALTMEMPHY IP wizard opens, refer to Figure 6–6 on page 6–11.

**Figure 6–6. System Content**



4. In the **DDR2 High Performance Controller** wizard interface, select **7** for the **Speed grade** to match the chosen device.
5. For the **PLL reference clock frequency**, type **50** MHz, to match signal `CLKIN50`.
6. For the **Memory clock frequency**, type **150** MHz.

☞ 150 MHz is the maximum supported frequency for DDR2, SSTL-18 class I, in top and bottom I/O, in a C7 speed grade Cyclone III device.

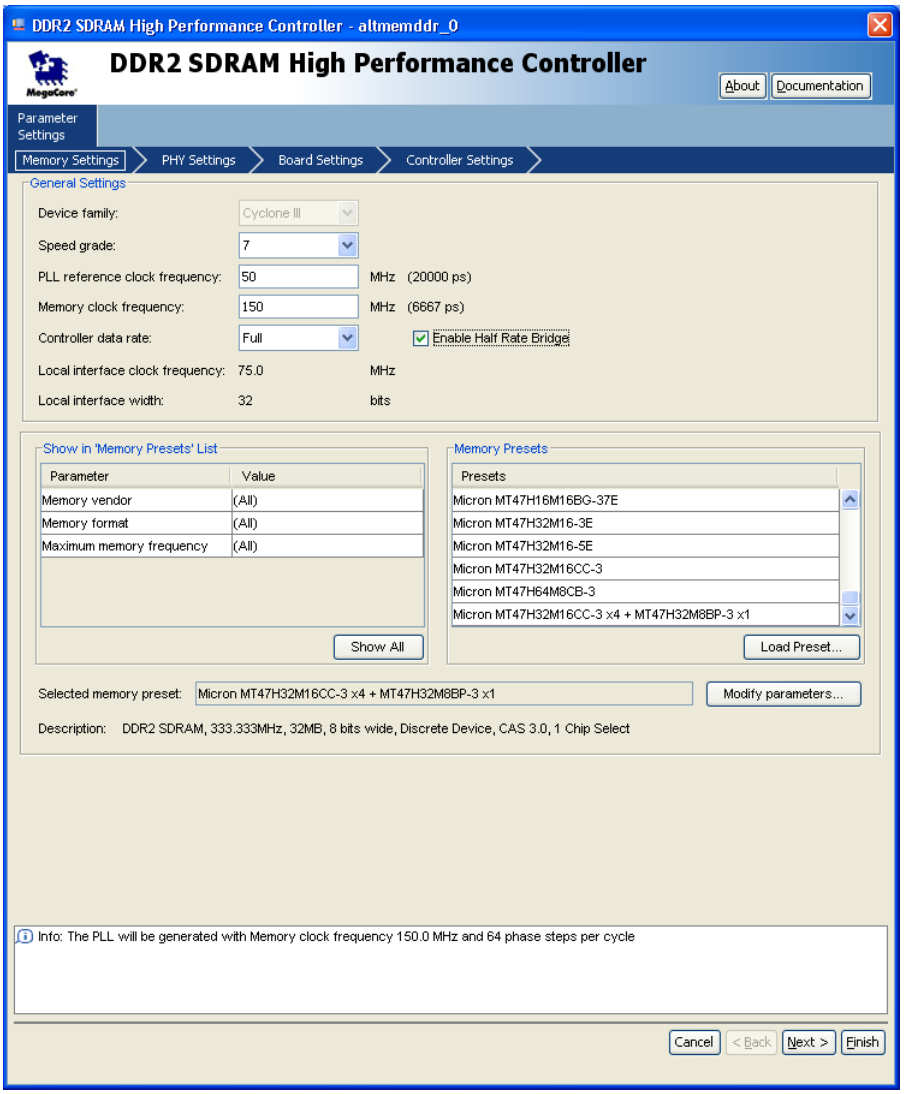
7. For the **Controller data rate**, select **Full**. Full-rate frequency is selected because the half-rate bridge is used in HPC II to reduce latency by driving the PHY and controller to work faster.

   ☞ The **Enable Half Rate Bridge** option is only available with the full-rate memory controllers.

8. For the **Memory vendor**, select **Micron**.
9. For the **Memory format**, select **Discrete Device**.
10. For the **Memory Presets**, select **Micron MT47H32M16CC-3 x4 + MT47H32M8BP-3 x1**, refer to Figure 6–7 on page 6–13.

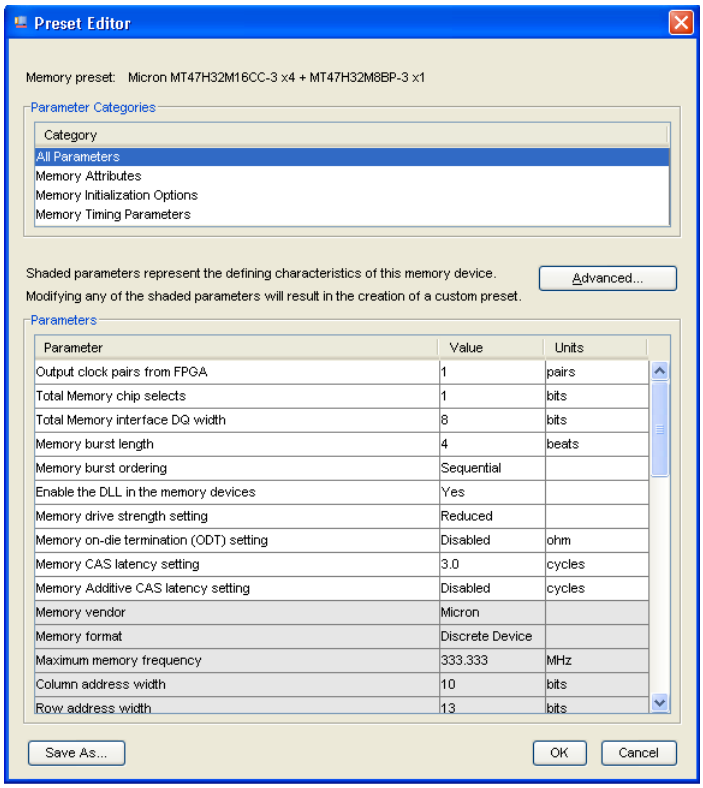
This memory preset has been included by Altera in the DDR2 SDRAM Controller with ALTMEMPHY IP as it matches the exact configuration that both the Cyclone III development board and the Stratix II GX PCIe development kit use.

**Figure 6–7. Memory Settings**



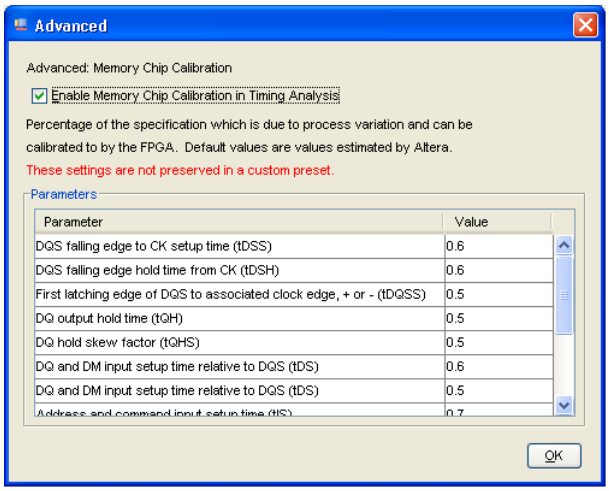
11. Click **Modify parameters** and ensure that you change the following parameters:
    - **Output clock pairs from FPGA**, select **1 pair**
    - **Total Memory interface DQ width**, select **8 bits**, to give in an Avalon-MM width of 32 bits, which is ideal to connect to a Nios II processor
    - **Memory drive strength setting**, select **Reduced** (DQ are low load point-to-point connections)
    - **Memory on-die termination (ODT) setting**, select **Disabled** (discrete class I termination is already fitted to the development board)
    - **Memory CAS latency setting,** select **3.0 cycles** (this DDR2 SDRAM supports CAS = 3 for frequencies of 200 MHz or lower)
12. Click **OK**, then click **Finish** in the DDR2 SDRAM Controller with ALTMEMPHY IP interface.

**Figure 6–8. Preset Editor**



If you are using a device that requires timing model pessimism removal, ensure that you turn on the **Enable Memory Chip Calibration in Timing Analysis** option on the **Advanced** page. This option is not supported by Cyclone III devices.

**Figure 6–9. Advanced Memory Settings**



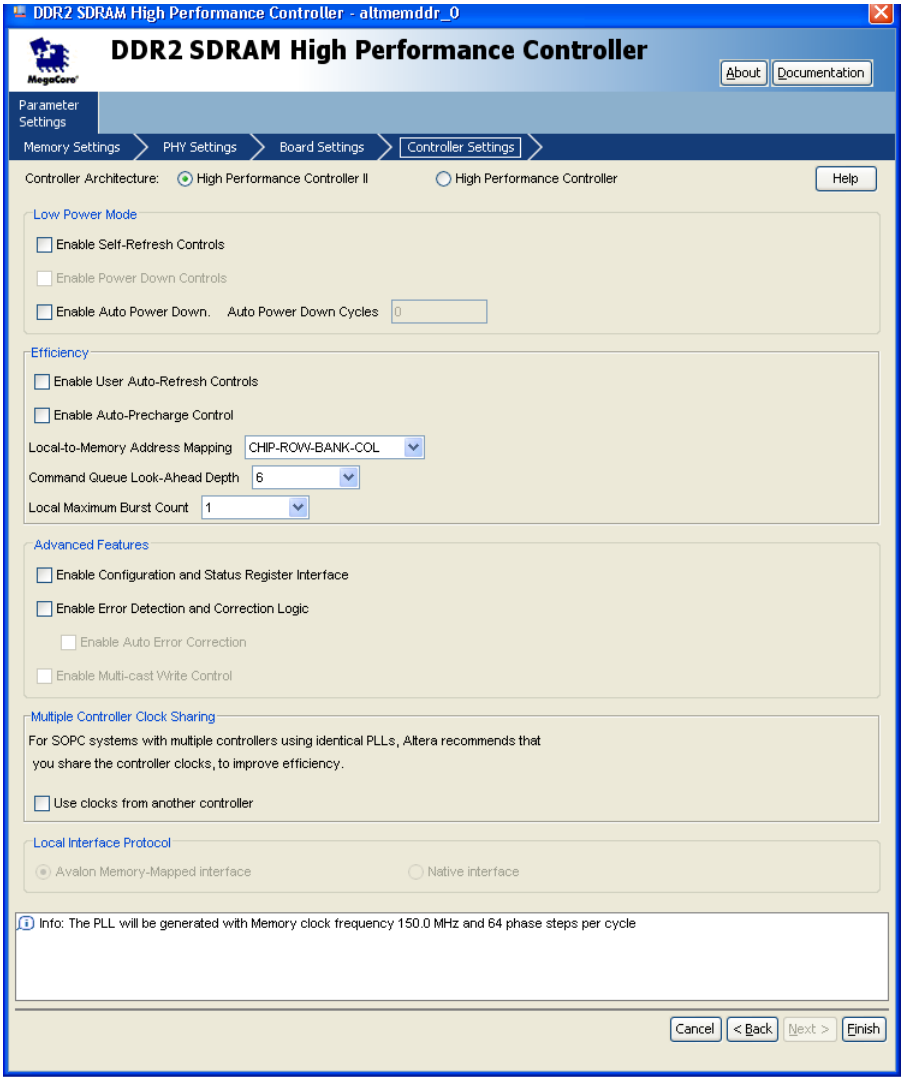

Because Cyclone III devices do not support flexible timing methodology, ignore the board settings. If you are using devices that support flexible timing methodology, ensure that you fill in the **Board Settings** parameters based on your board simulation results to achieve accurate timing analysis.

For more information about timing model pessimism removal, refer to the *Timing Analysis* section in volume 4 of the *External Memory Interface Handbook*.

13. In the **Controller Settings** tab, select **High-Performance Controller II**.
14. For **Command Queue Look-Ahead Depth**, select **6**.
15. For **Local-to-Memory Address Mapping**, select **CHIP-ROW-BANK-COL**.
16. For **Local Maximum Burst Count**, select **1**, refer to Figure 6–10.

Set the maximum burst count based on the burst count of the core logic system.

**Figure 6–10. DDR2 SDRAM HPC II Settings**




If you are using multiple controllers, turn on the **Use clocks from another controller** option under **Multiple Controller Clock Sharing** to save clock resources, and improve system efficiency and latency. This option allows the controllers to share the static PHY clocks between multiple controllers that run on the same frequency and share the same PLL reference clock.

For more information about how to share the clock for multiple memory interfaces, refer to the *Implementing Multiple Memory Interfaces* section in volume 6 of the *External Memory Interface Handbook*.

17. Click the **Memory Settings** tab, and turn on the **Enable Half Rate Bridge** option.
18. Click **Finish**.

## Add SOPC Builder Components

To add components from the **System Contents** tab, perform the following steps:

1. Under **Component Library**, expand **Memories and Memory Controllers** and expand **On-Chip**. Select **On-Chip Memory (RAM or ROM)**, and click **Add**.
   a. For **Total memory size** type **64 KBs**.
   b. Click **Finish**.
2. Select **On-Chip Memory** again, click **Add**.
   a. Set **Total memory size** to **4096 Bytes.**
   b. Click **Finish.**
   c. Right click on this second on-chip memory, click **Rename** and type `dma_read_memory` and press Enter.
3. On the System menu, click **Auto-Assign Base Addresses**.
4. Under **Component Library**, expand **Processors**, select **Nios II Processor** and click **Add**.
   a. Select **Nios II/s**.
   b. For **Reset Vector Memory** and **Exception Vector Memory**, select **onchip_memory**. If the local on-chip memory holds the Nios II instruction code, less arbitration is required to the SDRAM interface resulting in a more optimal Avalon-MM structure.
   c. Change **Reset Vector Offset** to **0x20** and **Exception Vector Offset** to **0x40**.

   
   If you select SDRAM as reset and exception vector, the controller performs memory interface calibration every time it is reset and in doing so writes to addresses `0x00` to `0x1f`. If you want your memory contents to remain intact through a system reset, you should avoid using the memory addresses below `0x20`. This step is not necessary, if you reload your SDRAM contents from flash every time you reset.

   d. Click **Finish**.
5. Expand **Interface Protocols** and expand **Serial**, select **JTAG UART** and click **Add**.
   a. In the **Configuration** tab, under **Write FIFO** and **Read FIFO**, for the **Buffer depth (bytes)** select **64** and for **IRQ threshold** type **8**.
   b. In the **Simulation** tab, select **Create Modelsim alias to open an interactive stimulus/response window**, to open the interactive display window during simulation.
   c. Click **Finish**.
6. Expand **Peripherals** and expand **Microcontroller Peripherals**, select **PIO (Parallel I/O)**, click **Add**.
   a. For the **Width** type **8 bits**.
   b. For **Direction** select **Output ports only**.
   c. Click **Finish**.

7. Expand **Memories and Memory Controllers** and expand **DMA**, select **DMA Controller** and click **Add**.
   a. In the **DMA Parameters** tab, turn on **Enable burst transfers**, and for the **Maximum burst size** select **512 words**.
   b. In the **Advanced** tab, turn off **byte**, **halfword**, **doubleword** and **quadword**.
   c. Click **Finish.**

   ☞ Do not to add a PLL component to your SOPC Builder design as the DDR2 SDRAM Controller with ALTMEMPHY IP includes one.

8. Set the bus links, refer to Figure 6–11 on page 6–19.
   a. Connect the DMA **read_master** and **write_master** to both the **altmemddr** and the **dma_read_memory**.

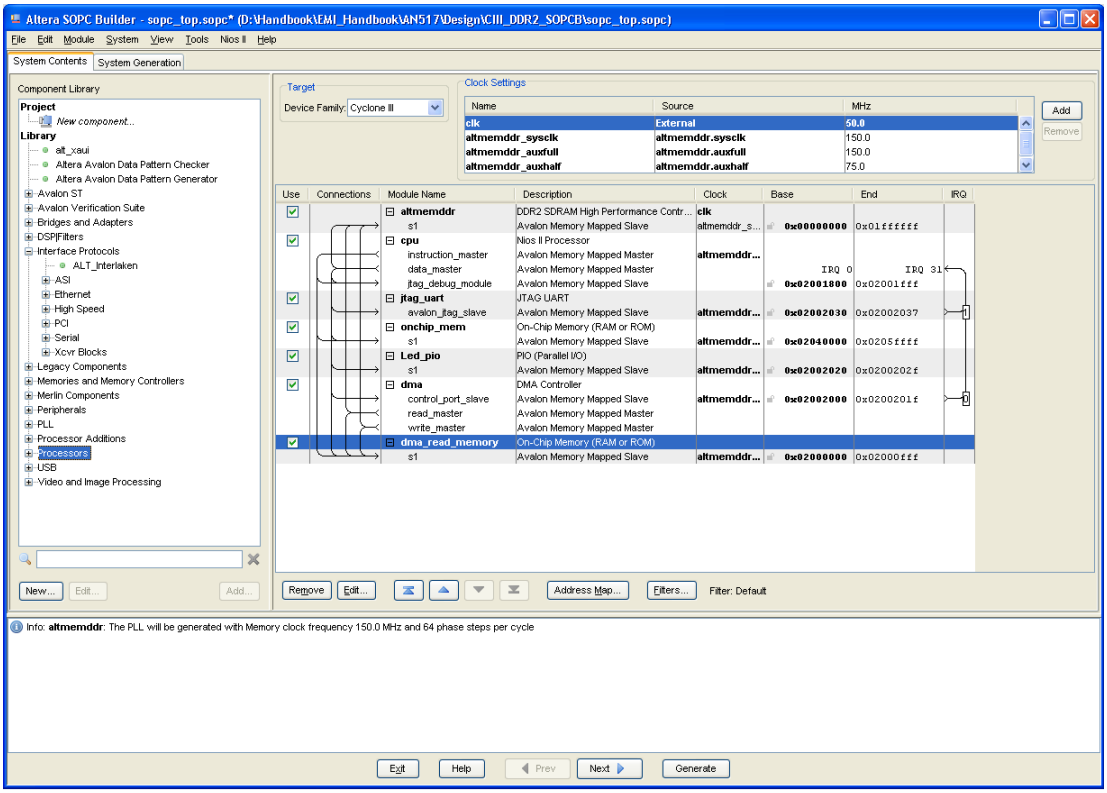
   ☞ If there are warnings about overlapping addresses, on the System menu click **Auto-Assign Base Addresses**.

   ☞ If there are warnings about overlapping IRQ, on the System menu click **Auto-Assign IRQs**.

   b. Ensure that **altmemddr** is clocked on the external clock **clk** and that the frequency matches the external oscillator (50 MHz for the Cyclone III development board).
   c. Ensure that all other modules are clocked on the **altmemddr_sysclk**, to avoid any unnecessary clock-domain crossing logic.

For more information on SOPC Builder system interconnect fabric, refer to the *System Interconnect Fabric for Memory-Mapped Interfaces* chapter in the *Quartus II Handbook*.

**Figure 6–11. SOPC Builder Final System Connection**




## Generate the SOPC Builder System

To generate the system, perform the following steps:

1. In SOPC Builder, click **Next**.
2. Turn on the **Simulation. Create simulator project files** option.
3. In SOPC Builder, click **Generate**. Click **Save**. When the generation is completed, the following message appears:

   ```
   SUCCESS: SYSTEM GENERATION COMPLETED.
   ```

4. In SOPC Builder, click **Exit**.

For more information on SOPC simulation and testbench options, refer to the *SOPC Builder User Guide* and *AN 351: Simulating Nios II Systems*.

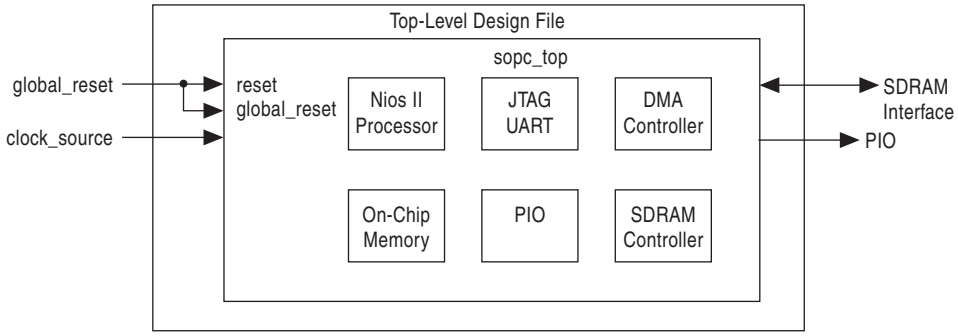
## Create the Top-Level Design File

Conceptually, you can consider the SOPC Builder system as component in your design. It can be the only component; or one of many components. Hence, when your SOPC Builder system is complete, you must add it to your top-level design.

The top-level design can be in your preferred HDL language, or simply a **.bdf** schematic design.

In this walkthrough, the top-level design is a simple wrapper file around the SOPC Builder system with no additional components. The top-level design file just defines the pin naming convention and port connections. Figure 6–12 shows the SOPC Builder top-level block diagram.

**Figure 6–12. SOPC Builder Top-Level Block Diagram**



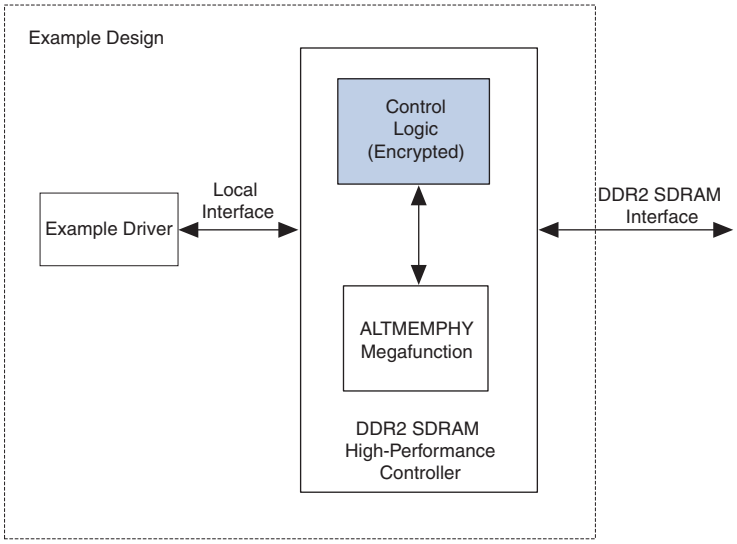

The SDRAM high-performance controller must make assignments to the top-level memory interface pins. These assignments are applied with the auto-generated script and constraint files. To ensure the constraints work, you must ensure the pin names and pin group assignments match, otherwise you get a no fit when you compile your design. The pin names default to `mem_*`. You can see the expected default pin names in the generated **altmemddr_example_top** file. You may optionally apply a prefix to the default pin naming convention.

All the SDRAM high-performance controllers generate the standalone design example **altmemddr_example_top**. This file only includes the controller and an example driver, refer to Figure 6–13. This file does not instantiate your SOPC Builder system but you can use the file in one of the following ways:

- Identify the default memory controller pin names

- Use as a starting point for the rest of the design

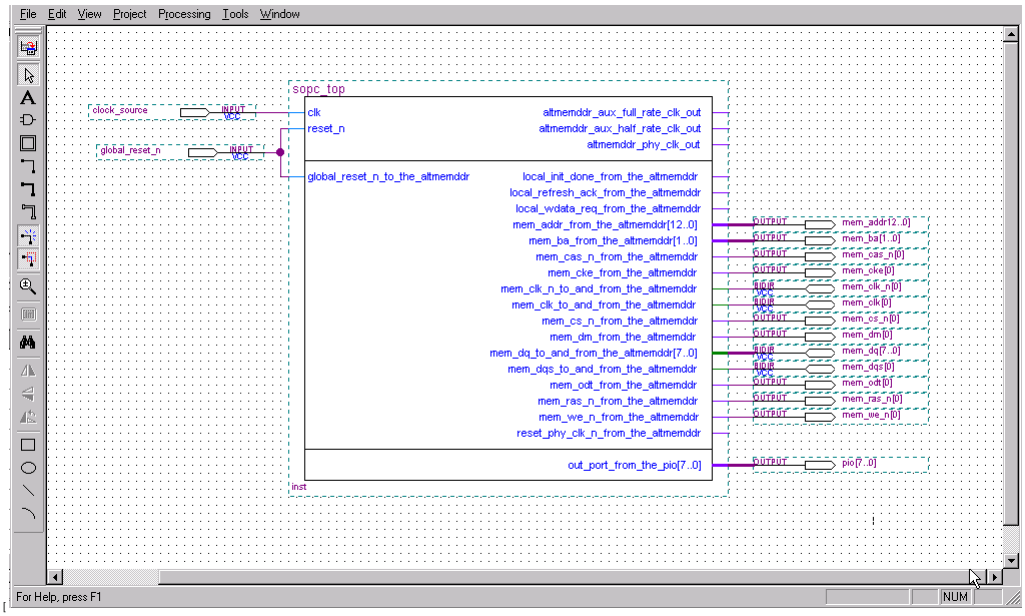
**Figure 6–13. Design Example**




You can create a HDL top-level design using the **altmemddr_example_top** as a template. You may edit the pin names (if required) in the **altmemddr_example_top** file only with the addition of a prefix. You must replace the example driver and the DDR2 SDRAM high-performance controller and instantiate your SOPC Builder-generated system.

As a reference, Altera provides an example**.bdf** top-level design with the correct pin names, refer to Figure 6–14. The `mem_clk[0]` and `mem_clk_n[0]` pins are bidirectional because of the mimic path in the ALTMEMPHY megafunction.

**Figure 6–14. Quartus II Top-Level Project**



To create a top-level design for your SOPC Builder system using a Quartus II **.bdf** schematic, perform the following steps:

1. In the Quartus II software, on the File menu click **New**.
2. Select **Block Diagram/Schematic File** and click **OK**. A blank **.bdf**, **Block1.bdf**, opens.
3. On the File menu click **Save As**. In the **Save As** dialog window, click **Save**.

   The Quartus II software automatically sets the **.bdf** file name to your project name.

4. Right-click in the blank **.bdf**, point to **Insert** and click **Symbol** to open the **Symbol** dialog box.
5. Expand **Project**, under **Libraries** select **sopc_top**, click **OK**.
6. Position the SOPC Builder system component outline in the *<project>***.bdf** and left-click.
7. Right-click on the SOPC Builder system component and click **Generate Pins for Symbol Ports**, to automatically add pins and nets to the schematic symbol.

   The SOPC Builder system includes the following signals, which are not required for this design example and can be disconnected:

   - `altmemddr_phy_clk_out`
   - `local_init_done_from_the_altmemddr`
   - `local_refresh_ack_from_the_altmemddr`
   - `local_wdata_req_from_the_altmemddr`
   - `reset_phy_clk_n_from_the_altmemddr`
   - `altmemddr_aux_full_rate_clk_out`
   - `altmemddr_aux_half_rate_clk_out`

   The following single vector signals must have a `[0]` vectored pin name, otherwise the simulation may fail:

   - `mem_dm`
   - `mem_dqs`
   - `mem_clk`
   - `mem_cke`
   - `mem_cs_n`
   - `mem_odt`

The `altmemddr_aux_full_rate_clk_out` and `altmemddr_aux_half_rate_clk_out` clock signals instantiate the full-rate (`altmemddr_auxfull`) and half-rate (`altmemddr_auxfhalf`) clocks that are exported from the DDR2 SDRAM high-performance controller to the top level of SOPC Builder. These two clocks are used to clock the half-rate bridge if the half-rate bridge is instantiated externally. These clocks are also used to clock other components within the system.

8. The SOPC Builder system has two reset inputs, `reset_n` and `global_reset_n_to_the_altmemddr`. Connect both these signals to a single pin, `global_reset_n`.
9. Ensure that you rename the clock signal as `clock_source`.

For more information on the signals, refer to the *DDR and DDR2 SDRAM Controller with ALTMEMPHY IP User Guide* section and the *DDR3 SDRAM Controller with ALTMEMPHY IP User Guide* section in volume 3 of the *External Memory Interface Handbook*.

10. Rename `out_port_from_the_Led_pio[7...0]` to `pio[7...0]`.
11. On the File menu, click **Save**, to save your changes.
12. On the Project menu, click **Set as Top-Level Entity**.

## Add Constraints

After generating the DDR2 SDRAM Controller with ALTMEMPHY IP, the ALTMEMPHY megafunction generates the constraint files for the design. You need to apply these constraints to the design before compilation.

### Device Settings

Set the unused pin and device voltage correctly before adding the constraint and pin assignment. Perform the following steps to set the device voltage and unused pin:

1. In the **Category** list, select **Device** and click **Device and Pin Options**.
2. Click the **Unused Pins** tab, and for **Reserve all unused pins** select **As input tri-stated with weak pull-up resistor**.
3. Click the **Voltage** tab, and for **Default I/O standard** select the same VCCIO voltage as the chosen SDRAM interface; for DDR2 SDRAM **select 1.8 V**.
4. Click **OK**.

### Add Timing Constraints

When you instantiate an SDRAM high-performance controller, it automatically generates a timing constraints file, **altmemddr_phy_ddr_timing.sdc**. The timing constraint file constrains the clock, and the input and output delay on the SDRAM high-performance controller.

To add timing constraints, perform the following steps:

1. On the Assignments menu, click **Settings**.

2. In the **Category** list, expand **Timing Analysis Settings**, and select **TimeQuest Timing Analyzer**.
3. Select the **altmemddr_phy_ddr_timing.sdc**, **altera_avalon_half_rate_bridge_constraints.sdc cpu.sdc**, and **cycloneIII_3c120_generic.sdc** files and click **Add**.

   Add `derive_pll_clocks` command in the **altera_avalon_half_rate_bridge_constraints.sdc** file before you add this file.

4. Click **OK**.

## Set Optimization Technique

To ensure that the remaining unconstrained paths are routed with the highest speed and efficiency, set the optimization technique. To set the optimization technique, perform the following steps:

1. On the Assignments menu, click **Settings**.
2. Select **Analysis & Synthesis Settings**.
3. Select **Speed** under **Optimization Technique**.
4. Click **OK**.

## Set Fitter Effort

To set the Fitter effort, perform the following steps:

1. On the Assignments menu click **Settings**.
2. In the **Category** list, expand **Fitter Settings**.
3. Turn on **Optimize Hold Timing** and select **All Paths**.
4. Turn on **Optimize Multi-Corner Timing**.
5. Select **Standard Fit** under **Fitter Effort**.
6. Click **OK**.

## Add Pin, DQ Group, and IO Standard Assignments

The pin assignment script, **altmemddr_pin_assignments.tcl**, sets up the I/O standards for the DDR2 SDRAM interface. It also does the DQ group assignment for the Fitter to place them correctly. However, the pin assignment does not create a clock for the design. You need to create the clock for the design.

To run the **altmemddr_pin_assignments.tcl** script to add the pin, I/O standards, and DQ group assignments to the design example, perform the following steps:

1. On the Tools menu, click **Tcl Scripts**.
2. Select **altmemddr_pin_assignments.tcl**.
3. Click **Run**.

If you require pin name changes (prefix changes only), then in the Quartus II Pin Planner, perform the following steps:

1. On the Assignment menu, click **Pin Planner**.
2. Right-click in any area under **Node Name** and select **Create/Import Megafunction**, refer to Figure 6–15.
3. Select **Import an existing custom megafunction** and select the **<variation name>.ppf** file.
4. In the **Instance name** field, type the prefix that you want to use, refer to Figure 6–16 on page 6–25.

**Figure 6–15. Create/Import Megafunction in Pin Planner**

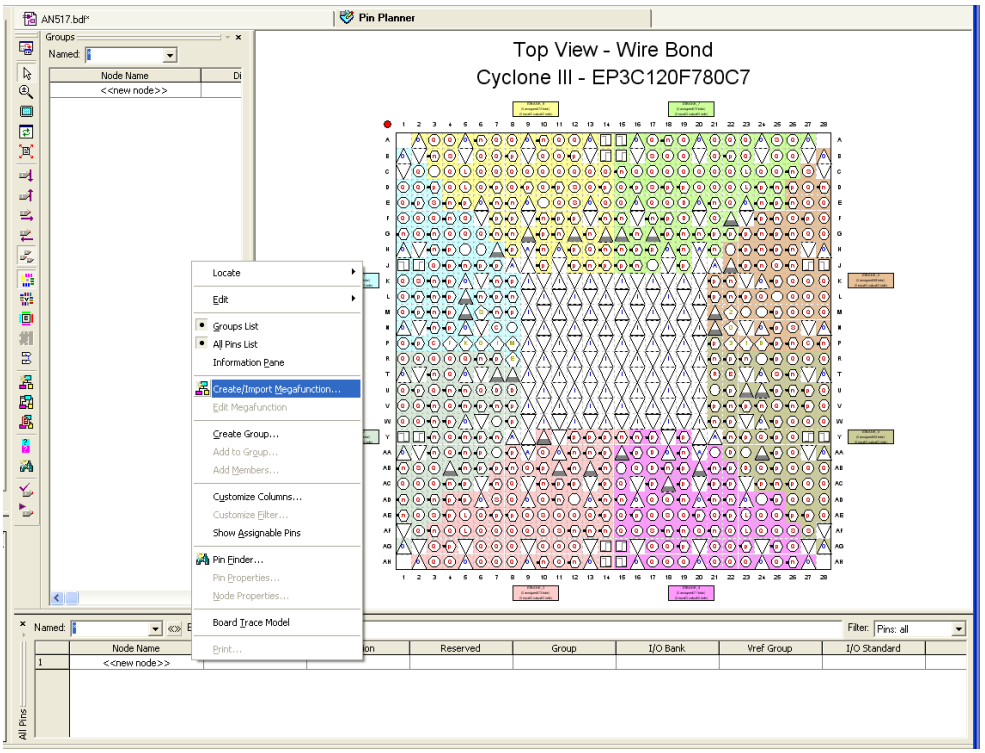
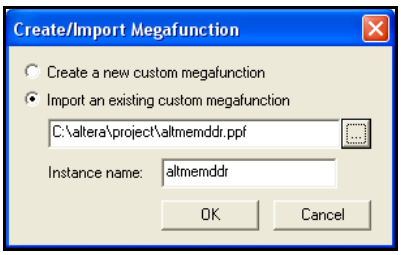



**Figure 6–16. Adding Prefix**




Alternatively, you may edit the **altmemddr_pin_assignments.tcl** file to change the prefix, and assign the pins without the **_from_to_** pin names, by following these steps:

1. Open the **altmemddr_pin_assignments.tcl** file.
2. To create assignments that have **_from_to_** pin names, change to the following code:

```
switch $sopc_mode {
  YES {
    set output_suffix _from_the_${instance_name}
    set bidir_suffix _to_and_from_the_${instance_name}
    set input_suffix _to_the_${instance_name}
  }
  default {
    set output_suffix ""
    set bidir_suffix ""
    set input_suffix ""
  }
}

to

switch $sopc_mode {
  NO {
    set output_suffix _from_the_${instance_name}
    set bidir_suffix _to_and_from_the_${instance_name}
    set input_suffix _to_the_${instance_name}
  }
  default {
    set output_suffix ""
    set bidir_suffix ""
    set input_suffix ""
  }
}
```

3. Type your preferred prefix in the `pin_prefix` variable. For example, to add prefix **mem_**, change following code:

```
if {![info exists set_prefix]}{set pin_prefix "my mem_"}
```

4. Save the **altmemddr_pin_assignments.tcl** file.
5. Run the **.tcl** script.

For more information about the DDR, DDR2, and DDR3 SDRAM high-performance controllers pin naming, refer to the *DDR and DDR2 SDRAM Controller with ALTMEMPHY IP User Guide* and the *DDR3 SDRAM Controller with ALTMEMPHY IP User Guide* in volume 3 of the *External Memory Interface Handbook*.

## Pin Location Assignments

To assign pin locations, perform the following steps:

1. On the Processing menu, point to **Start** and click **Start & Synthesis**.

2. Assign all of your pins, so the Quartus II software fits your design correctly and gives correct timing analysis. To assign pin locations for the Cyclone III development board, run the Altera-provided **C3_Dev_DDR2_Sopc_Pin_Location.tcl** file or manually assign pin locations by using either the Pin Planner or Assignment Editor.

   The SDRAM high-performance controller auto-generated scripts do not make any pin location assignments.

   If you are at the design exploration phase of your design cycle and do not have any PCB defined pin locations, you should still manually define an initial set of pin constraints, which can become more specific during your development process.

To manually assign pin locations in the Pin Planner, perform the following steps:

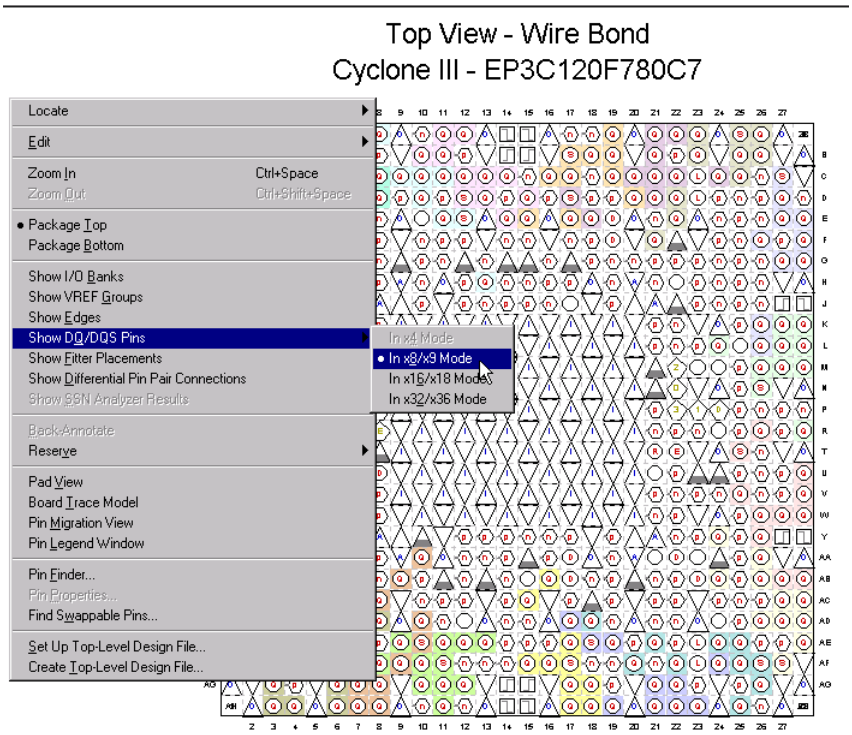
1. On the Assignments menu, click **Pin Planner**.
2. Assign DQ and DQS pins.
   a. To select the device DQS pin groups that the design uses, assign each DQS pin in your design to the required DQS pin in the Pin Planner. The Quartus II Fitter then automatically places the respective DQ signals onto suitable DQ pins within each group. To see DQS groups in the Pin Planner, right click, select **Show DQ/DQS Pins**, and click **In x8/x9 Mode**. The Pin Planner shows each DQS group in a different color and with a different legend: S = DQS pin, Sbar = DQSn pin and Q = DQ pin, refer to Figure 6–17 on page 6–28.

      Most DDR2 SDRAM devices operate in ×8/×9 mode, however as some DDR2 SDRAM devices operate in ×4 mode, refer to your specific memory device datasheet.

   b. Select the DQ mode to match the DQ group width (number of DQ pins/number of DQS pins) of your memory device. DQ mode is not related to the memory interface width.

DQ group order and DQ pin order within each group is not important. However, you must place DQ pins in the same group as their respective strobe pin.

**Figure 6–17. Quartus II Pin Planner, Show DQ/DQS Pins, In x8/x9 Mode**




3. Place the DM pins within their respective DQ group.
4. Place the address and control/command pins on any spare I/O pins ideally within the same bank or side of the device as the mem_clk pins.
5. Ensure the mem_clk pins use any regular adjacent I/O pins—ideally differential I/O pairs for the CK/CK# pin pair. To identify differential I/O pairs, right-click in Pin Planner and select **Show Differential Pin Pair Connections** (**DIFFIO** in Pin Planner). Pin pairs show a red line between each pin pair.

For Cyclone III and Cyclone IV devices, the mem_clk[0] pin cannot be located in the same row and column pad group as any of the interfacing DQ pins.

The DDR2 SDRAM in Stratix III and Stratix IV devices with differential DQS mode require the mem_clk[0]/mem_clk_n[0] pin on a DIFFIO_RX capable pin pair. The DDR3 SDRAM interfaces in Stratix III and Stratix IV devices require the mem_clk[0]/mem_clk_n[0] pin in any DQ or DQS pin pair with DIFFIO_RX capability.

For more information about memory clock pin placement and external memory pin selection, refer to the *Device and Pin Planning* section in volume 2 of the *External Memory Interface Handbook*.

6. Place the `clock_source` pin on a dedicated PLL clock input pin with a direct connection to the SDRAM controller PLL/DLL pair—usually on the same side of the device as your memory interface. This recommendation reduces PLL jitter, saves a global clock resource, and eases timing and fitter effort.
7. Place the `global_reset_n` pin (like any high fan-out signal) on a dedicated clock pin.

For more information on how to use the Quartus II Pin Planner, refer to the *I/O Management* chapter in volume 2 of the *Quartus II Handbook*. For more information on Cyclone III external memory pin selection, refer to the *External Memory Interface in Cyclone III Devices* chapter in the *Cyclone III Device Handbook*.

## Enter Board Trace Delay Model

For accurate I/O timing analysis, the Quartus II must be aware of the board trace and loading information. This information should be derived and refined during your PCB development process of prelayout (line) simulation and finally post-layout (board) simulation. For the external memory interfaces that use memory modules (DIMMs), the board trace and loading information should include the trace and loading information of the module in addition to the main and host platform, which you can obtain from your memory vendor.

To include the board trace information, perform the following steps:

1. In the Pin Planner, select the pin or group of pins that you want to enter the information for.
2. Right-click and select **Board Trace Model**.

Figure 6–18 on page 6–30 shows the board trace model. Enter the values such as the trace length, capacitance/length, inductance/length, pull-up, pull-down and others based on your board.

**Figure 6–18. Board Trace Model**

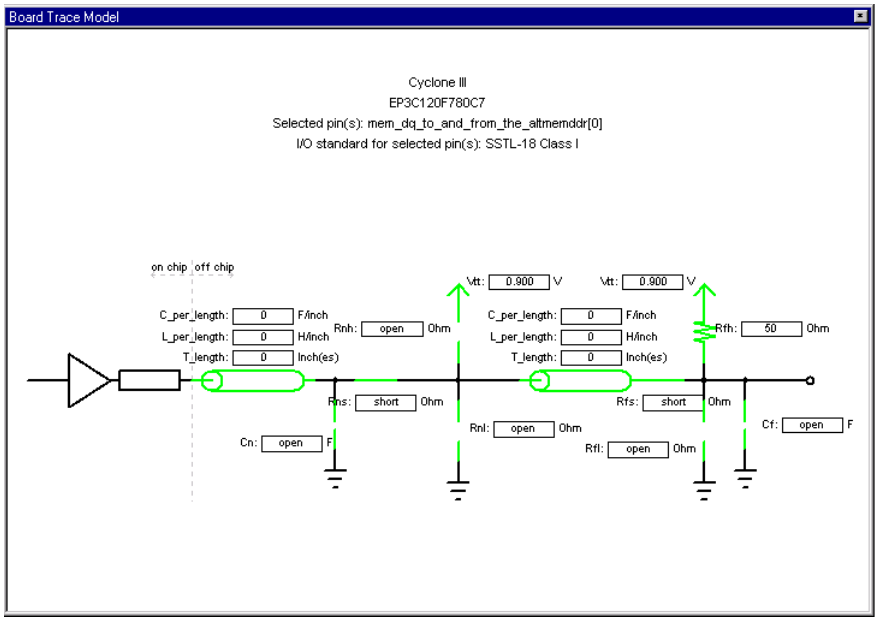



Table 6–3 shows the board trace model parameters for the Cyclone III development board.

**Table 6–3. Cyclone III Development Board Trace Model Summary**

| Net | Near (FPGA End of Line) | | | | | | Far (Memory End of Line) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Length | C_Per_ Length | L_Per_ Length | Cn | Rns | Rnh | Length | C_Per_ Length | L_Per_ Lenght | Cf | Rfh/Rfp |
| Address *(1)* | 1.764 | 3.8p | 8.0n | — | — | 56 | 0.456 | 3.6p | 8.0n | 4.0p | — |
| CLK | 1.872 | 3.4p | 8.4n | — | — | — | 0.566 | 3.4p | 8.4n | 4.0p | — |
| CKE/CSn | 1.862 | 3.7p | 8.1n | — | — | 56 | 0.418 | 3.6p | 8.0n | 4.0p | — |
| ODT | 1.527 | 3.8p | 8.0n | — | — | 56 | 0.449 | 3.8p | 8.0n | 4.0p | — |
| DQS *(2)* | 1.165 | 3.6p | 8.0n | — | — | 56 | 0 | 0p | 0n | 3.5p | — |

**Note to Table 6–3:**

(1) Address = `addr`, `ba`, `we#`, `ras#`, and `cas`.

(2) `DQS` = to `DM`, `DQ`, and `DQS` pins.

Altera recommends you use the **Board Trace Model** assignment for all DDR, DDR2 and DDR3 SDRAM interface signals. To apply board trace model assignments for the Cyclone III development board, run the Altera-provided **C3_Dev_DDR2_BTModels.tcl** file or manually assign virtual pin assignments using the Quartus II Pin Planner.

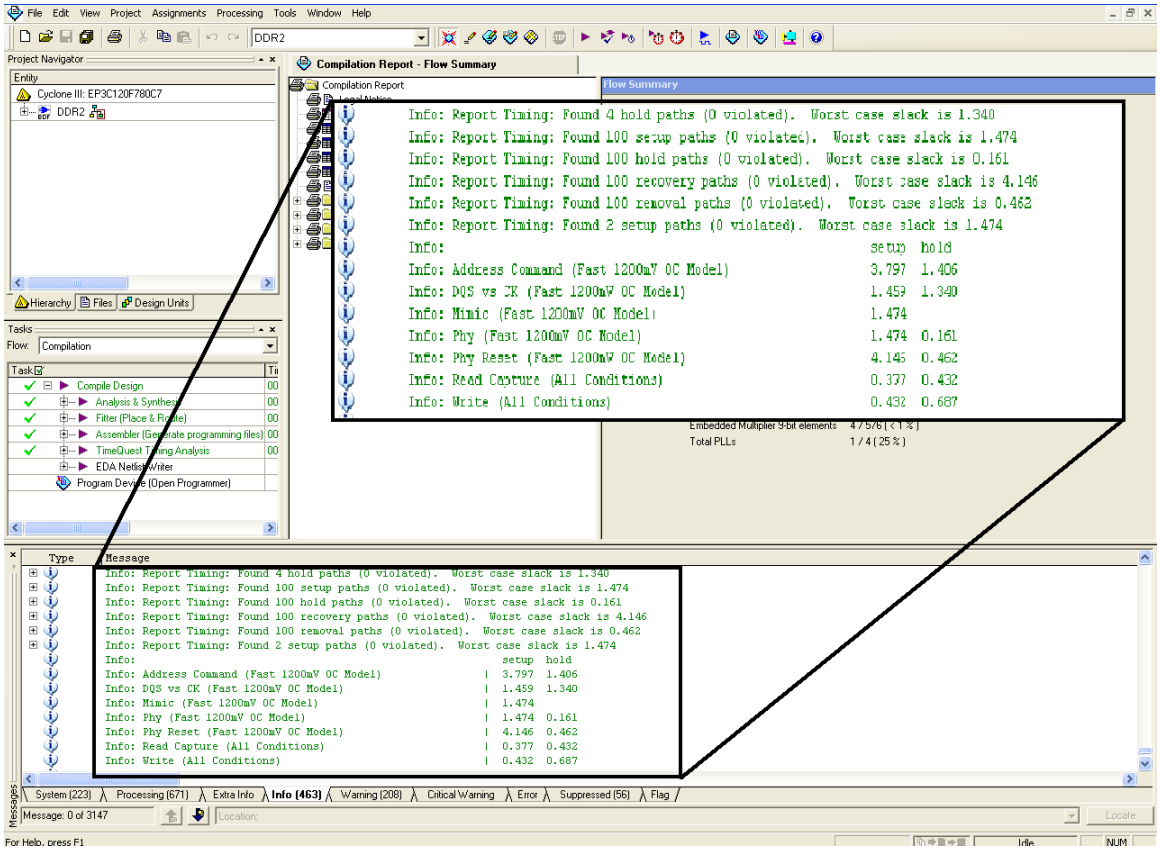
## Compile the Design

To compile the design, on the Processing menu, click **Start Compilation**.

After successfully compiling the design, the Quartus II software automatically runs the **altmemddr_phy_report_timing.tcl** file, which produces a timing report for the design together with the compilation report.

Figure 6–19 shows the timing margin report in the message window in the Quartus II software.

**Figure 6–19. Timing Margin Report in the Quartus II Software**



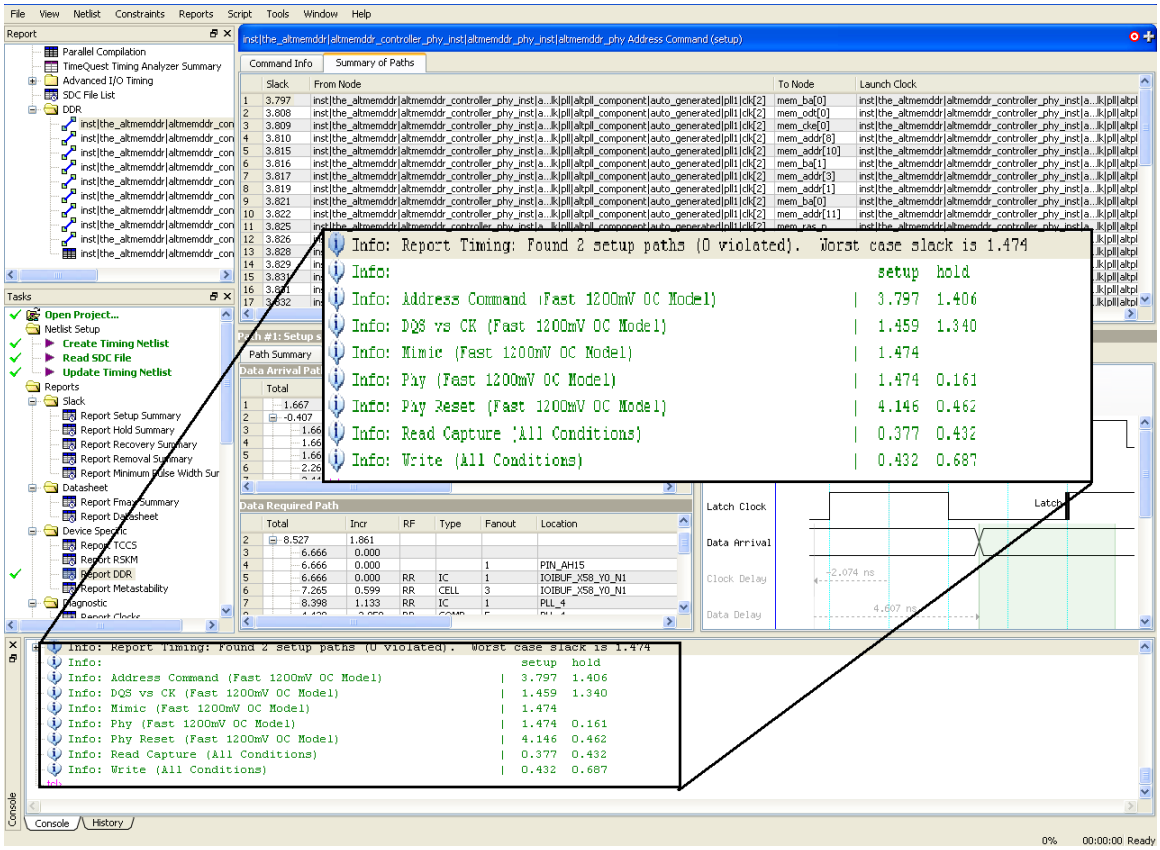

You may also run the report timing script in the TimeQuest Timing Analyzer window, To run the timing script, perform the following steps:

1. On the Tools menu, click **TimeQuest Timing Analyzer**.
2. Double-click **Update Timing Netlist** in the **Tasks** pane, which automatically runs **Create Timing Netlist** and **Read SDC File**. After a task is executed, it turns green.
3. After completing the tasks, run the report timing script by going to the Script menu and clicking **Run Tcl Script**.

Alternatively, you can directly double-click on **Report DDR** in the **Tasks** pane to get the timing report.

Figure 6–20 shows the timing margin report in the TimeQuest Timing Analyzer window after running the report timing script. The results are the same as the Quartus II software results.

**Figure 6–20. Timing Margin Report in the TimeQuest Timing Analyzer**




## Incorporate the Nios II IDE

You can now add test code to the project's Nios II processor and use this program to run some simple tests.

When doing memory tests, you must read and write from the external DDR SDRAM and not from cached memory. The following three methods avoid using Nios II cached memory:

- Use a Nios II processor that does not have cache memory, like the Nios II/s used in this example project.
- Use the `alt_remap_uncached` function.

### Launch the Nios II IDE

To launch the Nios II IDE, perform the following steps:

1. One the Windows Start menu, point to **All Programs**, **Altera**, **Nios II EDS** <*version*>, **Legacy Nios II Tools**, and then click **Nios II** <*version*> **IDE**.

   If it is the first time you have run the IDE, click **Workbench**.

2. On the File menu, click **Switch Workspace** and select your project directory.
3. On the File menu, point to **New** and click **Project**.
4. Expand **Altera Nios II** select **Nios II C/C++ Application** and click **Next**.
5. Select **Blank Project** under Select Project Templates and click **Next**.
6. Ensure that SOPC Builder System **.ptf** is *<your project path>/<your sopc top>*.**ptf**.
7. Click **Next** and click **Finish**.
8. In Windows Explorer, drag Altera-supplied **DDR_TEST.c** to the **blank_project_0** directory.

To see the **DDR_TEST.c** example test program, refer to "Example DDR_TEST.c Test Program File" on page 6–45.

This program consists of a simple loop that takes commands from the JTAG UART and executes them. Table 6–4 shows the commands that are sent to the Nios II C code.

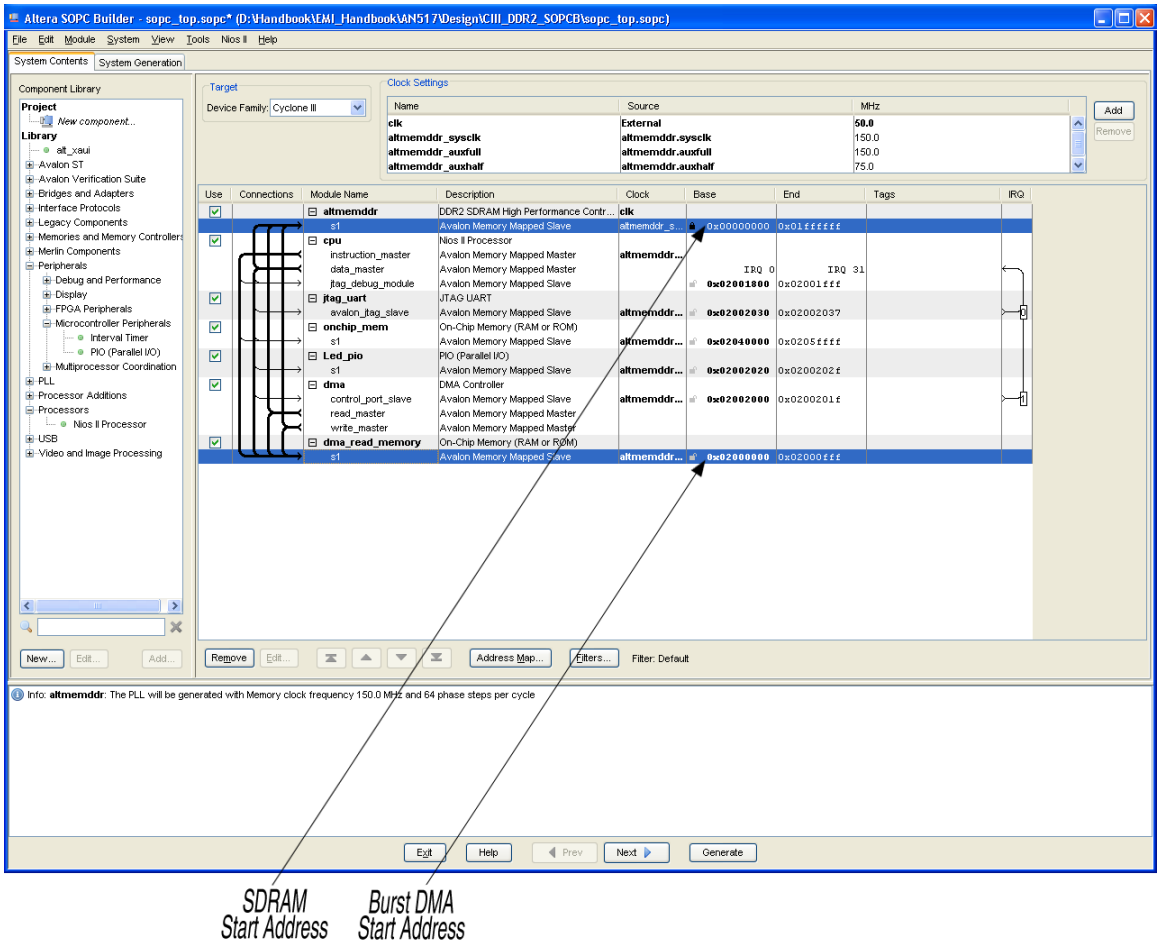
The commands must be in the following format and the switches A, B, C, and D must be entered in upper case. Both address and data strings must always be 8 characters long.

**Table 6–4. Commands**

| Command | Format | Description | Example |
|---|---|---|---|
| Switch A | Switch \| Data | Controls the LEDs. | `A000000FF`. The last two bytes control all eight LEDs. The LEDs are active low on the board. |
| Switch B | Switch \| Address \| Data | Performs a single write to an address offset location in memory. | Enter `B`, followed by `00000001`, then `00000001`, writes `00000001` at SDRAM memory address location `00000001`. |
| Switch C | Switch \| Address | Performs a single read to an address offset location in memory. | Enter `C`, followed by `00000001`, reads the contents of the memory at SDRAM address offset location `000000010`. |
| Switch D | Switch \| From Address \| To Address | Performs an incremental write to the first address location, followed by a DMA burst transfer from the first address location to the second address location. The burst is a fixed length of 512 words or 2048 bytes. | `D0200000000000000` loads the DMA read memory with the incrementing pattern, then DMA burst transfers it to the SDRAM high-performance controller. |

You can read the SOPC Builder component address locations directly from the SOPC Builder Window, refer to Figure 6–21.

**Figure 6–21. SOPC Builder Component Address**




## Set Up the Project Settings

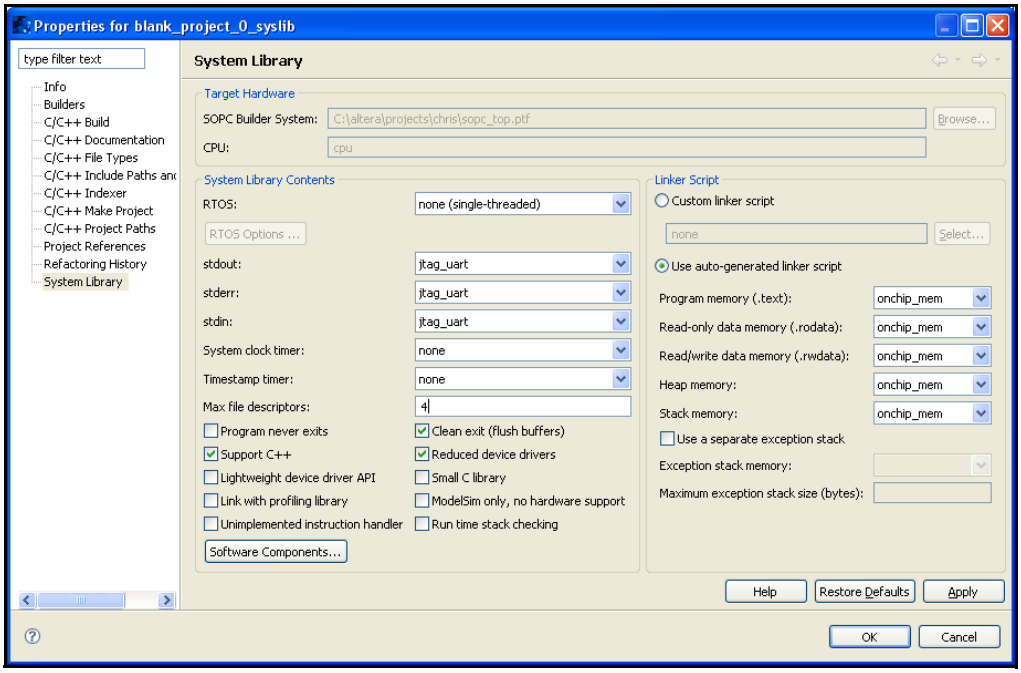
To set up the Nios II project settings, perform the following steps:

1. In the **Nios II C/C++ Projects** tab, right-click **blank_project_0** and click **System Library Properties.**
2. Click **System Library** in the list on the left side of the **Properties** dialog box.
3. To optimize the footprint size of the library project, under **System Library Contents**, turn on **Reduced device drivers**.
4. Under **Linker Script**, for all the memory options select **onchip_mem**. The on-chip memory is the target memory space for the executable file.

5. To reduce the memory size allocated for the system library, for **Max file descriptors** type 4.

   Figure 6–22 shows the **Properties** dialog box.

**Figure 6–22. Nios II Project Settings**




6. Click **OK**.

## Perform RTL or Functional Simulation (Optional) with Nios II

Perform the following steps to run RTL or functional simulation using the Nios II processor:

1. From the Nios II IDE, on the Run menu click **RUN**.
2. Under Nios II ModelSim, select **blank_project_0**.
3. Specify your ModelSim installation path in **ModelSim Path**.
4. Click **Run**. ModelSim is launched.
5. Type `s` in the ModelSim Transcript window to run the **s** macro to load the design.
6. Type `w` in the ModelSim Transcript window to run the **w** macro to display the waveform.
7. Type `jtag_uart_drive` to launch the interactive terminal.
8. Type `Run -all` to run the design.

9. In the interactive terminal, type your desired operation, data, and address, refer to Figure 6–23.

**Figure 6–23. The JTAG UART Interactive Terminal**

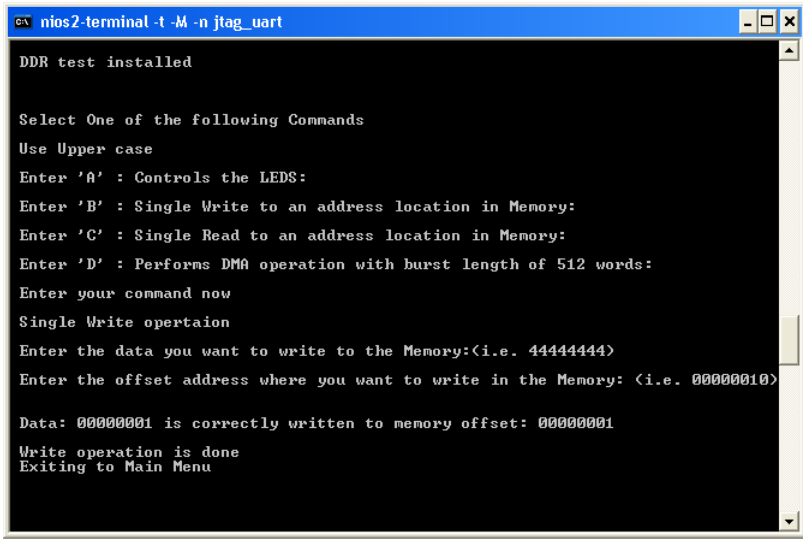
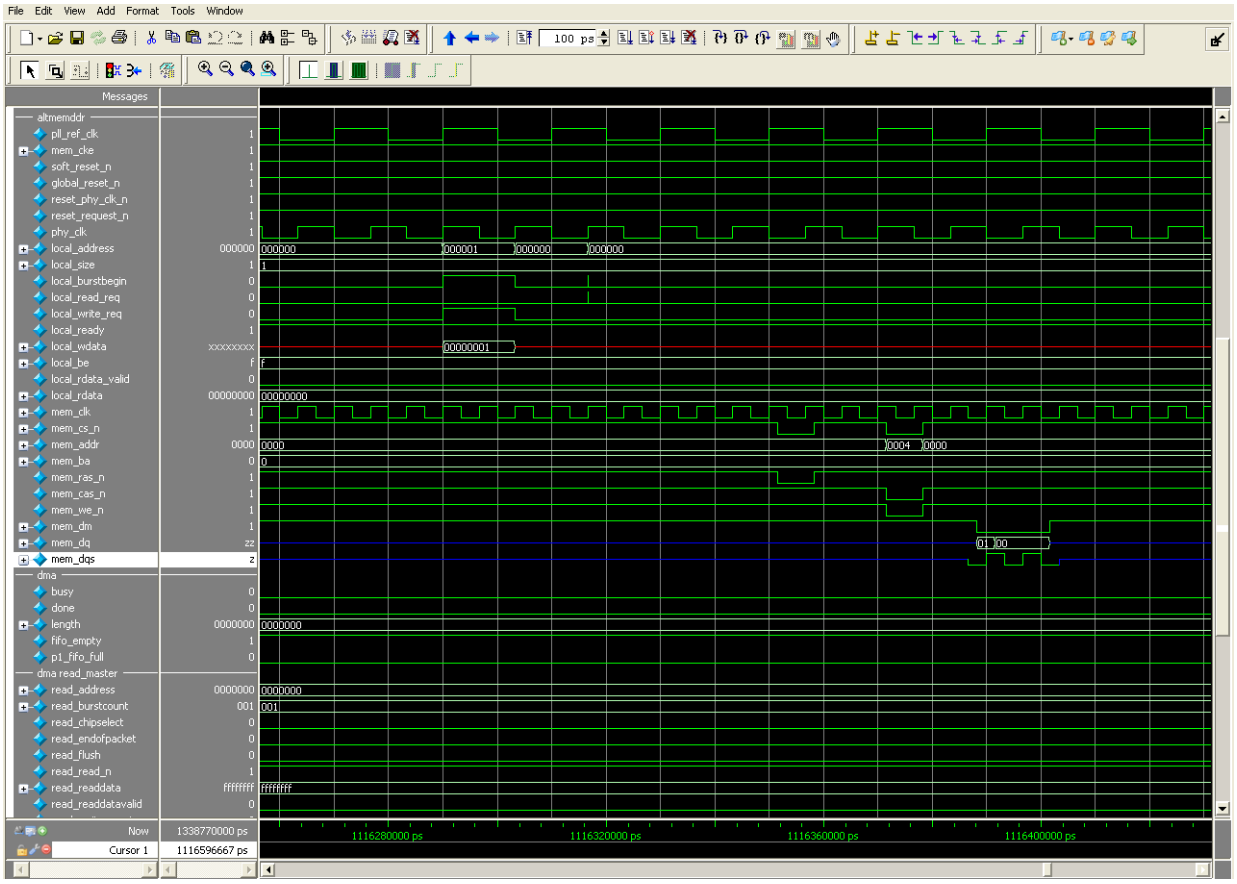


Figure 6–24 shows a single write operation where data `00000001` is written to address `00000001`.

**Figure 6–24. Waveform for Single Write Operation**




## Verify Design on a Development Platform

The SignalTap II Embedded Logic Analyzer shows read and write activity in the system.

For more information about using the SignalTap II Embedded Logic Analyzer, refer to the *Design Debugging Using the SignalTap II Embedded Logic Analyzer* chapter in the *Quartus II Handbook*, *AN 323: Using SignalTap II Embedded Logic Analyzers in SOPC Builder Systems* and *AN 446: Debugging Nios II Systems with the SignalTap II Logic Analyzer*.

To add the SignalTap II Embedded Logic Analyzer, perform the following steps:

1. On the Tools menu, click **SignalTap II Logic Analyzer**.
2. In the **Signal Configuration** window next to the **Clock** box, click **...** (Browse Node Finder).
3. Type `*phy_clk` in the **Named** box, for **Filter** select **SignalTap II: pre-synthesis** and click **List**.

4. Select **sopc_top:inst|altmemddr:the_altmemddr|phy_clk** in **Nodes Found** and click **>** to add the signal to **Selected Nodes**.
5. Click **OK**.
6. Under Signal Configuration, specify the following settings:
   - For **Sample depth**, select **512**
   - For **RAM type**, select **Auto**
   - For **Trigger flow control**, select **Sequential**
   - For **Trigger position**, select **Center trigger position**
   - For **Trigger conditions**, select **1**
7. On the Edit menu, click **Add Nodes**.
8. Search for specific nodes by typing `*local*` in the **Named** box, for **Filter** select **SignalTap II: pre-synthesis** and click **List**.
9. In **Nodes Found**, select the following nodes and click **>** to add to **Selected Nodes**:
   - **local_address**
   - **local_rdata**
   - **local_rdata_valid** (alternative trigger to compare read/write data)
   - **local_read_req**
   - **local_ready**
   - **local_wdata**
   - **local_write_req** (trigger)

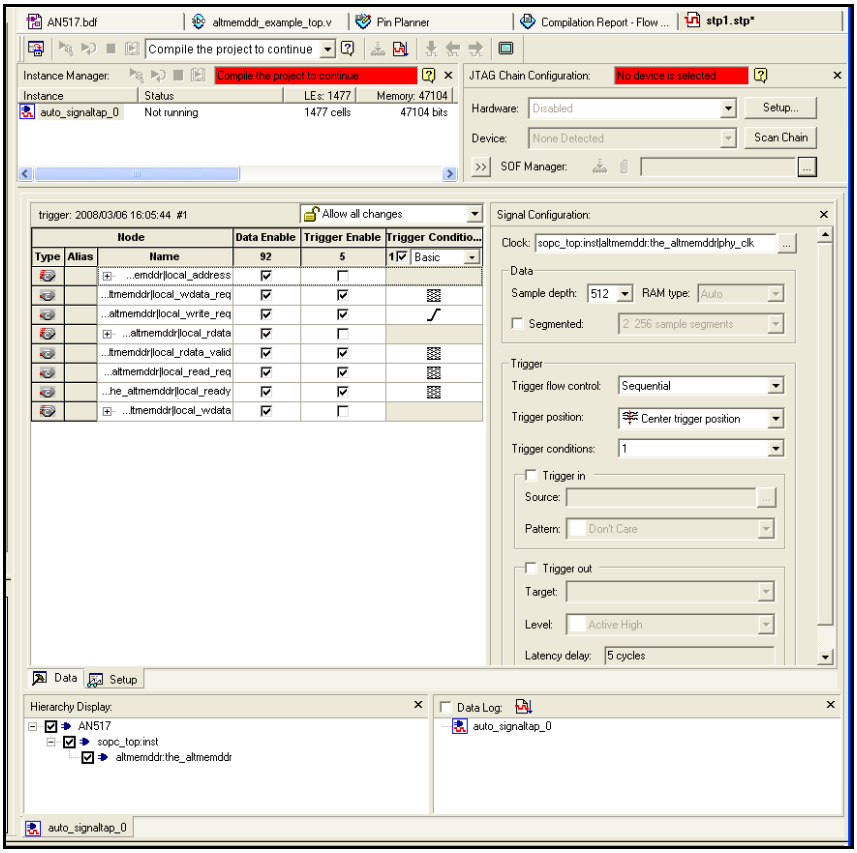
   
   Do not add any DDR SDRAM interface signals to the SignalTap II Embedded Logic Analyzer. The load on these signals increases and adversely affects the timing analysis.

10. Click **OK**.
11. To reduce the SignalTap II logic size, turn off **Trigger Enable** on the following bus signals:
    - **local_address**
    - **local_rdata**
    - **local_wdata**

12. Right-click **Trigger Conditions** for the `local_write_req` signal and select **Rising Edge**.

    Figure 6–25 shows the completed SignalTap II Embedded Logic Analyzer.

**Figure 6–25. SignalTap II Embedded Logic Analyzer**




13. On the File menu, click **Save**, to save the SignalTap II **.stp** file to your project.

    If you get the message **Do you want to enable SignalTap II file "stp1.stp" for the current project**, click **Yes**.

## Compile the Project

Once you add signals to the SignalTap II Embedded Logic Analyzer, recompile your design, on the Processing menu, click **Start Compilation**.

## Verify Timing

Once the design compiles, ensure that the TimeQuest timing analysis passes successfully. In addition to this FPGA timing analysis, check your PCB or system SDRAM timing.

To run timing analysis, run the <*variation name* >**_phy_report_timing.tcl** script by performing the following steps:

1. On the Tools menu, click **Tcl Scripts**.

2. Select <*variation name* >**_phy_report_timing.tcl** and click **Run**.

## Connect the Development Board

Connect the Cyclone III development board to your computer.

For more information on the Cyclone III development board, refer to the *Cyclone III Development Kit User Guide*.

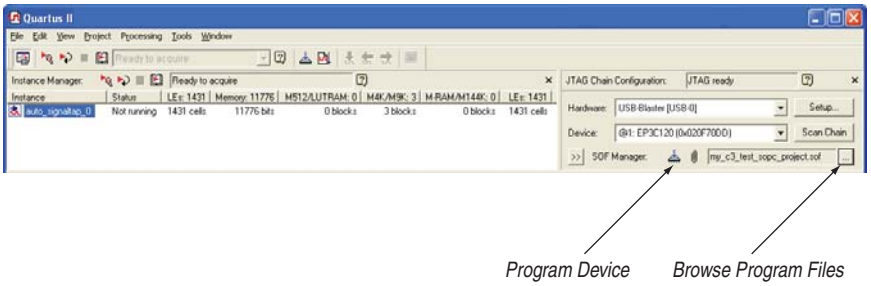
## Download the Object File

On the Tools menu, click **SignalTap II Logic Analyzer**. The SignalTap II dialog box appears.

The SRAM Object File (SOF) Manager should contain the <*your project name*>**.sof** file. To add the correct file to the SOF Manager, perform the following steps:

1. Click **...** to open the **Select Program Files** dialog box, refer to Figure 6–25.
2. Select <*your project name*>**.sof**.
3. Click **Open**.
4. To download the file, click the **Program Device** button.

**Figure 6–26. Install the SRAM Object File in the SignalTap II Dialog Box**




## Verify Design with Nios II

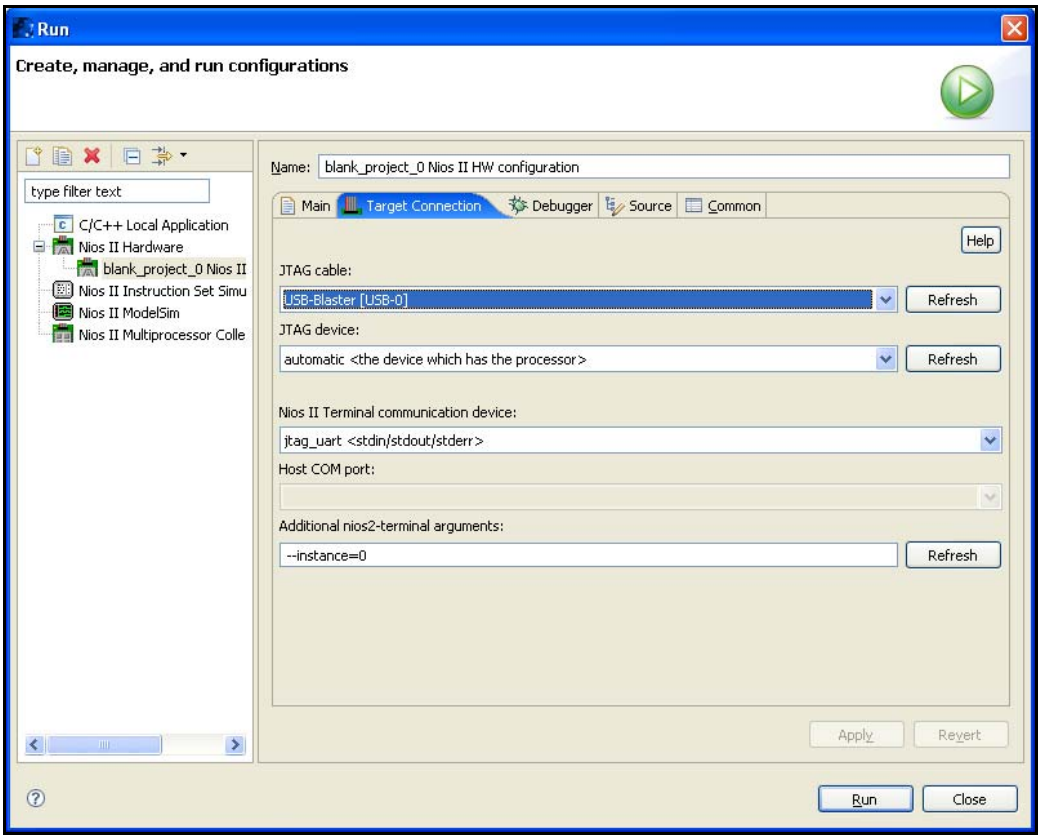
Right-click on **blank_project_0**, point to **Run As**, and click **Nios II Hardware** for the the Nios II C/C++ IDE to compile the example test program.

If you have more than one JTAG download cable connected to your computer you may see an JTAG error. If you receive a JTAG error, perform the following steps:

1. From the Nios II IDE, on the Run menu click **RUN**. Click the **Target Connection** tab and correct the **JTAG cable connection**, refer to Figure 6–27.

2. Right click on **blank_project_0**, point to **Run As**, and click **Nios II Hardware**.

**Figure 6–27. JTAG Download Target Connection Settings**




## Test the System

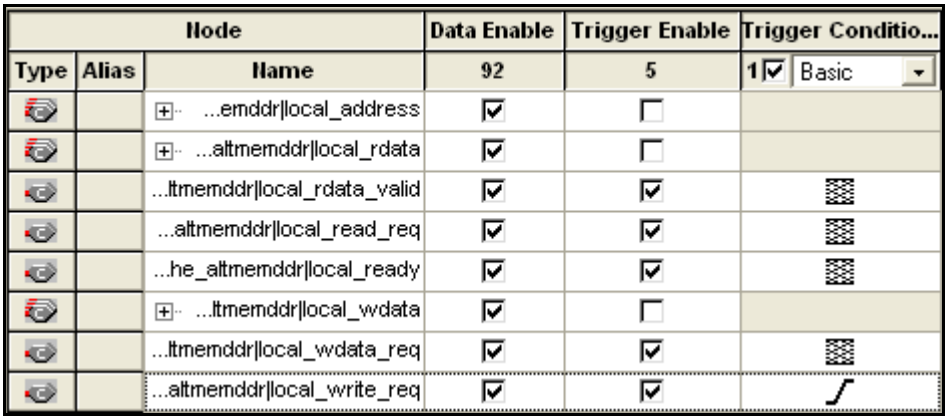
Perform the following tests to verify your system is operating correctly.

### Set SignalTap II Trigger to local_write_req

Use the SignalTap II Embedded Logic Analyzer to capture write activity. To show write activity, perform the following steps:

1. In the **Setup** tab, select to trigger on a rising edge of `local_write_req`, refer to Figure 6–28.

**Figure 6–28. Set Trigger for local_write_req Signal**

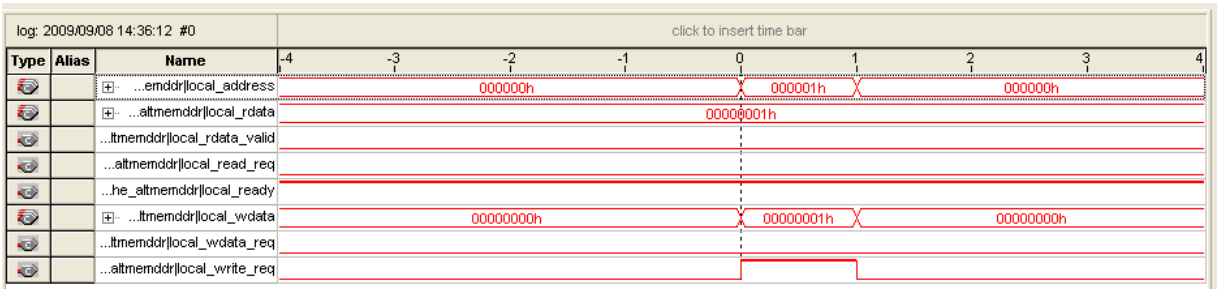



2. Click **Run Once Analysis** to capture the write request.
3. Type the following command in the Nios II command console:

```
B0000000010000001
```

4. Return to the SignalTap II Embedded Logic Analyzer window and check that at time 0 (refer to Figure 6–29):
   - `local_write_req` signal goes high for a single cycle
   - `local_wdata` shows `00000001h`
   - `local_address` shows `000001h`

**Figure 6–29. Waveform for Write Request**

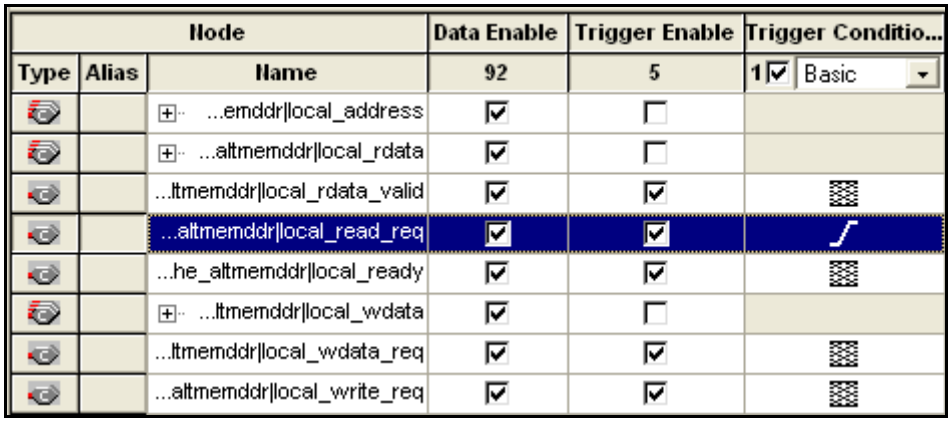



### Set SignalTap II Trigger to local_read_req

Use the SignalTap II Embedded Logic Analyzer to capture read activity. To show read activity, perform the following steps:

1. In the **Setup** tab, select to trigger on a rising edge of `local_read_req`, refer to Figure 6–30.

**Figure 6–30. Set Trigger for local_read_req Signal**




2. Click **Run Once Analysis** to capture the read request
3. Type the following command in the Nios II command console:

```
C00000001
```

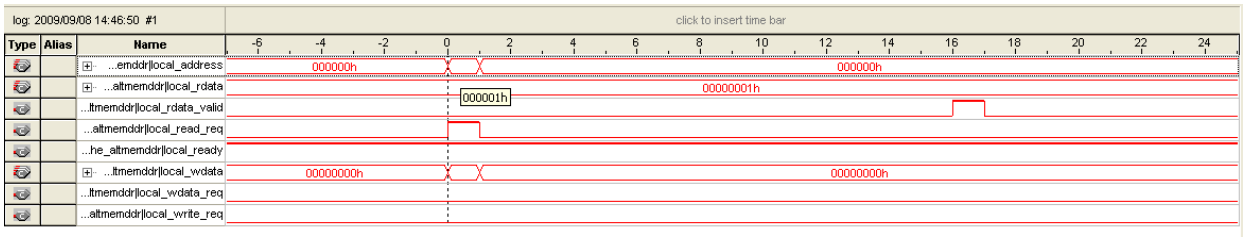
4. Return to the SignalTap II Embedded Logic Analyzer window and check that at time 0 (refer to Figure 6–22):

- `local_read_req` signal goes high
- `local_address` shows `000001h`

Several clock cycles later, depending on the system read latency:

- `local_rdata_valid` goes high for one cycle
- `local_rdata` shows `00000001h`

**Figure 6–31. Waveform for Read Request**

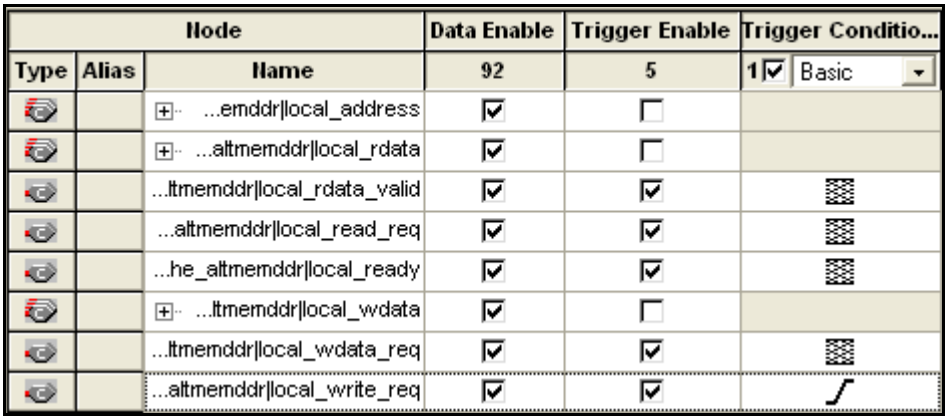



### Test Burst Write Operation

Use the SignalTap II Embedded Logic Analyzer to capture write activity. To show write activity, perform the following steps:

1. In the **Setup** tab, select to trigger on a rising edge of `local_write_req`, refer to Figure 6–32.

**Figure 6–32. Set Trigger for local_write_req Signal**



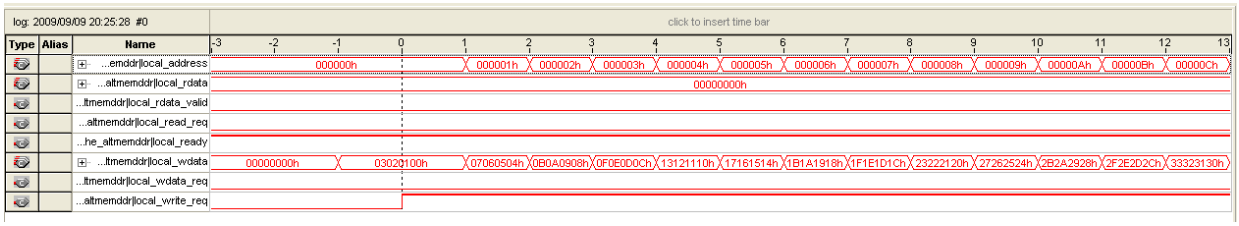

2. Click **Run Once Analysis** to capture the write request.
3. Type the following command in the Nios II command console:

   `D0200000000000000`

4. Return to the SignalTap II Embedded Logic Analyzer window and check that at time 0 (refer to Figure 6–22):
   - `local_write_req` signal goes high for multiple cycles
   - `local_wdata` shows `03020100h` followed by `07060504h` and so on
   - `local_address` shows `000000h` followed by `000002h` and so on

   The write data is first to last, most significant byte (MSB) to the least significant byte (LSB) count format. For example, a count of `00,01,02,03` = `03020100h`.

**Figure 6–33. Waveform for Burst Write Request**




### Test Burst Read Operation

Performing a burst read operation is similar to performing a burst write operation (refer to "Test Burst Write Operation" on page 6–43), but with the memory locations swapped. To perform a burst read operation, perform the following steps:

1. In the **Setup** tab, set to trigger on a rising edge of `local_read_req`.
2. Click **Run Once Analysis** to capture the read request:

3. Type the following command in the Nios II command console:

```
D0000000002000000
```

# Example DDR_TEST.c Test Program File

```
#include<stdio.h>
#include"sys/alt_dma.h"
#include "sys/alt_cache.h"
#include "system.h"
#include "altera_avalon_dma_regs.h"
#define length 512
int to_hex(char* pkt)
{
    unsigned int value[8];
    unsigned int value1=0;
    unsigned int q;
    for (q=0;q<=7;q++)
       {
            value[q]=(pkt[q]>0x39)?(pkt[q]-0x37):(pkt[q]-0x30);

        if (q==0)
            {
            value1=(value1+value[q]);
            }
        else
            {
            value1=((value1<<4)+value[q]);
             }

     }
        return value1;
}

/*****************************************************************
*  Function: Main_menu
*
*  Purpose: Prints the main menu commands
*
*****************************************************************/
static void Main_menu(void)
{
  printf("\n\n");
  printf(" \n Select One of the following Commands \n");
  printf( "\n Use Upper case \n");
  printf(" \n Enter 'A' : Controls the LEDS:\n");
  printf(" \n Enter 'B' : Single Write to an address location in
Memory:\n");
  printf(" \n Enter 'C' : Single Read to an address location in
Memory:\n");
  printf(" \n Enter 'D' : Performs DMA operation with burst length of
512 words:\n");
  printf( "\n Enter your command now \n");

}
 /*****************************************************************
*  Function: LED_Control
*
*  Purpose: Controls the LEDs..
*
*****************************************************************/
void LED_Control(void)
{
```

```
  unsigned int led_value;
  unsigned char led[8];
  printf(" \n LED Test operation \n");
  printf( "\n Enter the value in Hex you want to write to the LEDs:(i.e.
000000FF)\n");
  printf(" \n The last two bytes control all eight LEDs. \n");
  gets(led);
  led_value=to_hex(&led[0]);
  printf(" \n Value to be displayed on LEDs is: %08x \n",led_value);
  IOWR_32DIRECT(LED_PIO_BASE,0, led_value);

}

 /*****************************************************************
*  Function: Single_Write
*
*  Purpose: Performs a single write to an address location in memory.
*
*****************************************************************/
void Single_Write(void)
{
  unsigned char write_offset[8];
  unsigned char data[8];
  unsigned int DDR_write_OFFSET_ADDRESS;
  unsigned int write_data;
  printf(" \n Single Write operation \n");
  printf( "\n Enter the data you want to write to the Memory:(i.e.
44444444) \n");
  gets(data);
  write_data=to_hex(&data[0]);
  printf( "\n Enter the offset address where you want to write in the
Memory: (i.e. 00000010)\n");
  gets(write_offset);
  DDR_write_OFFSET_ADDRESS = to_hex(&write_offset[0]);
  if
((DDR_write_OFFSET_ADDRESS<0)||(DDR_write_OFFSET_ADDRESS>=(ALTMEMDDR_S
PAN/4)))
  {
    printf(" \n Invalid Offset \n");
    printf( "\n You have entered wrong offset address : \n");
    return;
  }
  IOWR(ALTMEMDDR_BASE,DDR_write_OFFSET_ADDRESS,write_data);
  if (IORD(ALTMEMDDR_BASE,DDR_write_OFFSET_ADDRESS)==write_data)
  {
    printf("\n Data: %08x is correctly written to memory offset: %08x
\n", write_data,DDR_write_OFFSET_ADDRESS);
    printf("\n Write operation is done \n");
  }
  else
  {
     printf("\n Write operation is Failed \n");
  }
}
 /*****************************************************************
*  Function: Single_Read
*
*  Purpose: Performs a single read to an address location in memory
*
*****************************************************************/
void Single_Read(void)
{
  unsigned char read_offset[8];
  unsigned int DDR_read_OFFSET_ADDRESS;
```

```
  unsigned int read_data;
  printf(" \n Single Read operation \n");
  printf( "\n Enter the offset address from where you want to read in
the Memory:(i.e. 00000010) \n");
  gets(read_offset);
  DDR_read_OFFSET_ADDRESS =  to_hex(&(read_offset[0]));
  if ((DDR_read_OFFSET_ADDRESS<0) ||
(DDR_read_OFFSET_ADDRESS>=(ALTMEMDDR_SPAN/4)))
  {
    printf(" \n Invalid Offset \n");
  }
  else
  {
    read_data=IORD(ALTMEMDDR_BASE,DDR_read_OFFSET_ADDRESS);
    printf("Read %08x from address %08x
\n",read_data,(ALTMEMDDR_BASE+DDR_read_OFFSET_ADDRESS));
  }
}
/*****************************************************************
*  Function: Verify Operation
*
* Purpose: Compares the memory contents of the read data master and write
data master
*
*****************************************************************/
void Call_verify(unsigned char* source, unsigned char* destination)
{
    if (memcmp(source,destination,(length*4))==0)
    {
     printf("\n DMA operation successful \n");

    }
    else
    {
     printf("\n DMA operation failed \n");
     printf( "\n Please check that DMA is correctly setup \n ");
     }
}
/*****************************************************************
*  Function: DMA_Operation
*
* Purpose: Performs an incremental write to the first address
location, followed by a DMA burst transfer from the first address
location to the second address location.
The burst is a fixed length of 512 words or 2048 bytes.
*
*****************************************************************/

void DMA_operation(void)
{
  unsigned int read_address;
  unsigned char* read_DMA_address;
  unsigned char *write_DMA_address;
  unsigned int write_address;
  unsigned char verify[8];
  unsigned char read[8];
  unsigned char write[8];
  unsigned char status_reg;
  unsigned int i;
  printf(" \n DMA setup operation \n");
  printf( "\n Enter DMA read master address:(i.e. 00000010) \n ");
  gets(read);
  read_address =to_hex(&read[0]);
  read_DMA_address=read_address;
  printf( " \n DMA read master address is %08x \n",read_DMA_address);
```

```
  printf( "\n Enter DMA write master address:(i.e. 00000010) \n ");
  gets(write);
  write_address =to_hex(&write[0]);
  write_DMA_address=write_address;
  printf( " \n DMA write master address is %08x \n",write_DMA_address);
  printf("\n Using %d bytes to verify the DMA operation \n",(length*4));
  printf( "\n Initializing Read memory with incremental data starting
from 00 \n");
  for (i=0;i<=(length*4);i++)
  {
   *read_DMA_address=i;
   read_DMA_address++;
  }
  read_DMA_address=read_address;
  printf( "\n Writing to the DMA control registers \n");
  IOWR_ALTERA_AVALON_DMA_CONTROL(DMA_BASE,0x00000084);
//DMA go bit is set to zero.
  IOWR_ALTERA_AVALON_DMA_STATUS(DMA_BASE, 0x00000000);
//clearing  done bit
  IOWR_ALTERA_AVALON_DMA_RADDRESS(DMA_BASE,read_DMA_address);
  IOWR_ALTERA_AVALON_DMA_WADDRESS(DMA_BASE,write_DMA_address);
  IOWR_ALTERA_AVALON_DMA_LENGTH(DMA_BASE,(length*4));
  printf( "\n Starting DMA engine \n");
  IOWR_ALTERA_AVALON_DMA_CONTROL(DMA_BASE, 0x0000008C);
  printf(" \n DMA operation is in progress \n ");
  status_reg= (IORD_ALTERA_AVALON_DMA_STATUS(DMA_BASE)&
ALTERA_AVALON_DMA_STATUS_DONE_MSK) ;
  if (status_reg ==1)
  {
  printf("\n DMA operation is Done \n ");
  printf("\n Do you want to verify the DMA operation \n");|
  printf("\n Enter Y for yes, N for No \n ");
  gets(verify);
  switch( verify[0])
  {
   case 'N':
       break;
   case 'Y':
       Call_verify(read_DMA_address, write_DMA_address);
       break;
   default :
      printf(" Error: unexpected command\n");
      break;

  }

  }

  else

  {
   printf(" \n DMA operation is in progress \n ");

  }

}

/*****************************************************************

*  Function: Main

*

* Purpose: Output the main menu and selects the app. function.

*

*****************************************************************/
```

```
int main()
{
  char packet[1];
  printf("\n DDR test installed \n");
  while (1)
  {
   Main_menu();
   gets (packet);
   switch(packet[0])
   {
     case 'A' :

       LED_Control();
       printf(" Exiting to Main Menu \n");
       break;

     case 'B' :

       Single_Write();
       printf(" Exiting to Main Menu \n");
       break;

     case 'C':

       Single_Read();
       printf(" Exiting to Main Menu \n");
       break;

     case 'D':

       DMA_operation();
       printf(" Exiting to Main Menu \n");
       break;

     default :
       printf(" Error: unexpected command. Switch was -%C\n",
packet[0]);
       break;

   }               //switch loop closure

  }               // while loop closure

  return 0 ;

}
```

# 7. Implementing Multiple ALTMEMPHY-Based Controllers

Many systems and applications use external memory interfaces as data storage or buffer mechanisms. As system applications require increasing bandwidth, become more complex, and devices get more expensive, designers prefer to have multiple memory interfaces in a single device to save cost and space on the board, along with removing partitioning complexity. To implement multiple memory interfaces on the same device that are efficient and optimized for the device architecture, designers must pay attention to the device and the physical interface (PHY) features.

This chapter describes the steps to implement multiple memory interfaces where there are independent memory transactions, but the interfaces are operating at the same frequency. Such implementations require multiple instantiations of the PHY, which in turn may necessitate device resource sharing, such as the delay-locked loops (DLLs) and clock networks, and may require extra steps to create the interfaces in a Quartus II project. Multiple memory interfaces may also involve different types of memory standards, for example, if you have DDR2 SDRAM and RLDRAM II interfaces in a single Stratix IV device. Width and depth expansion of a memory interface are not covered in this document as they are natively supported by the PHY. The memory interfaces described in this chapter use the ALTMEMPHY megafunction.

While you may not be able to share one PLL between multiple ALTMEMPHY-based controllers, you may be able to share the static clocks to reduce the number of clock networks used in the design.
You cannot merge dynamic clocks of the ALTMEMPHY megafunction or high-performance controllers. The Quartus II software may not give a warning, but this merging does not work in the hardware.

This chapter assumes that you are familiar with the ALTMEMPHY megafunction and Altera FPGA resources. There is no design example associated with this tutorial.

## Before Creating a Design in the Quartus II Software

When creating multiple memory interfaces to be fitted in a single device, first ensure that there are enough device resources for the different interfaces. These device resources include the number of pins, DLLs, PLLs, and clock networks, where applicable. The DLLs, PLLs, and clock network resources from the clock and reset management block can be shared between multiple memory interfaces, but there are sharing limitations that you need to pay close attention to.



For more information about selecting a device, refer to the *Device and Pin Planning* section in volume 2 of the *External Memory Interface Handbook*.

The following different scenarios exist for the resources sharing DLL and PLL static clocks:

- Multiple controllers with no sharing. You need to check the number of PLLs and DLLs that available in the targeted device and the number of PLLs and DLLs that needed in your design. If you have enough resources for the controllers, you do not need to share the DLL and PLL static clocks.
- Multiple controllers sharing DLL only. If you have enough resources on PLL clocks output and only need to share DLL, ensure that the controllers are running at the same memory clock frequency.
- Multiple controllers sharing PLL static clocks only. Ensure that all the controllers sharing the PLL static clocks are located in the same half of the device, as ALTMEMPHY requires the use of regional and dual-regional clocks, instead of global clocks. Each controller can have their own DLL as the DLL can only access the adjacent sides from its location.
- Multiple controllers sharing DLL and PLL static clocks. The PLL static clocks and DLL can be shared when the controllers are on the same side or adjacent side of the device and provided they are running at the same memory-clock frequency.

After understanding the device resource limitations, determine the resources to share for your multiple controller design. You can use the resources that you determined as a framework for your Quartus II design constraints. If you also have a PCI interface in your design, you need to leave at least one bank (either on the top or the bottom of the device) for the PCI interface. For this particular example interface, you may need to share a DLL.

# Creating PHY and Controller in a Quartus II Project

You can instantiate multiple controllers using either one of the following flows:

- "SOPC Builder Flow"
- "MegaWizard Plug-In Manager Flow"

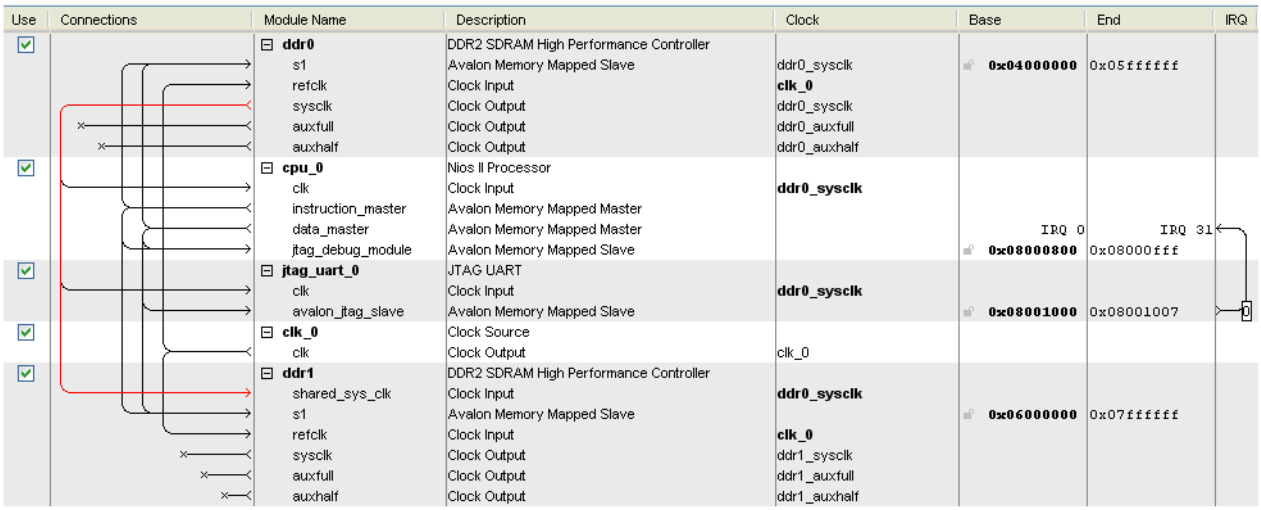
The SOPC Builder flow provides useful validation and automatically generates all the wiring in SOPC Builder system and assignment. For the MegaWizard™ Plug-In Manager flow, you must manually set the assignment for the PLL static clock sharing but this flow gives you more flexibility.

## SOPC Builder Flow

The SOPC Builder in version 8.1 and onwards of the Quartus II software allows you to add multiple controllers directly to a new or existing SOPC Builder system. The SOPC Builder supports sharing static PHY clocks between multiple controllers in the SOPC Builder that are running on the same frequency and sharing the same PLL reference clock. To share the static clocks, you must turn on the **Clock source from another controller** option in the **Controller Settings** page of the DDR, DDR2, or DDR3 SDRAM High Performance Controller wizard. This option must be turned on for the

slave controllers. Turning on this option adds a new clock input connection point to the slave controller named `shared_sys_clk`. You must connect the `sys_clk` signal from the master controller to the `shared_sys_clk` signal of the slave controller (Figure 7–1). The **Force Merging of PLL Clock Fanouts** assignment is automatically assigned in the Quartus II software.

**Figure 7–1. Connection Between Master and Slave Controller**




After the system generation, you must add the **.sdc** file and run the pin assignment Tcl scripts as usual.

When an SOPC Builder multiple controllers system does not share the `sys_clk`, SOPC Builder inadvertently inserts clock-crossing adaptors between the controller and the Nios II processor. In this situation, the system cannot achieve the maximum performance for both controllers.

SOPC Builder checks all the conditions listed in the regulation section and prevents you from generating the system when they are not met.

For more information on DDR, DDR2, and DDR3 SDRAM High Performance Controllers, refer to the *DDR and DDR2 SDRAM Controller with ALTMEMPHY IP User Guide* section and the *DDR3 SDRAM Controller with ALTMEMPHY IP User Guide* section in volume 3 of the *External Memory Interface Handbook*.

For more information on simulating SOPC Builder systems, refer to *AN 351: Simulating Nios II Systems*.

## MegaWizard Plug-In Manager Flow

If you are creating multiple instances of the same controller with the same parameters (frequency of operation, data width, burst mode, etc.), you only need to generate the controller once. You can then instantiate this controller variation multiple times in your top-level design file. If you are using controllers with different parameters, or you plan on modifying the RTL generated by the MegaWizard Plug-In Manager (as required for resource sharing in some cases), then you need to generate each variation of the memory controller individually.

If the controllers are generated individually Altera recommends you generate each controller's files in a separate directory. However, you may get compilation errors if you add all the **.qip** files to the project. You may remove the following line in all the **.qip** files except for one to remove the compilation error:

```
set_global_assignment -name VHDL_FILE [file join $::quartus(qip_path)
"auk_ddr3_hp_controller.vhd"]
```

The high-performance memory controller is generated with a top-level design called <*variation_name*>**_example_top.v** or **.vhd**, where *variation_name* is the name of the memory controller that you entered in the first page of the MegaWizard Plug-In Manager. This example top-level file instantiates the memory controller (<*variation_name*>**.v** or **.vhd**) and an example driver, which performs a comparison test after reading back data that was written to the memory. You can also generate simulation files for each high-performance controller or ALTMEMPHY megafunction.

When creating a multiple memory interface design, you have the option to combine certain aspects of the design flow. For example:

- You can use one of the top-level designs to instantiate all the memory controllers in the design, or create a new top-level design.
- You can either modify one of the example drivers to test all the memory interfaces in the design, or instantiate an example driver with each memory controller.
- You can simulate each memory interface separately, or create a testbench which combines all the memory interfaces in the design.

## Sharing DLLs

If the controllers are sharing a DLL, ensure that the **Instantiate DLL externally** option is turned on in the **PHY Settings** page of the DDR3/DDR2/DDR SDRAM High-Performance Controller or the ALTMEMPHY MegaWizard Plug-In Manager. This option must be turned on for every interface that is sharing a DLL.

When you turn on **Instantiate DLL externally**, the MegaWizard Plug-In Manager generates a file called <*variation_name*>**_phy_alt_mem_phy_dll_**<*device_family*>**.v/.vhd**, where <*device_family*> is:

- **siii** when targeting HardCopy® III, HardCopy IV, Stratix III, or Stratix IV devices
- **aii** when targeting Arria II GX devices

The example top-level file then instantiates the DLL in addition to instantiating the memory controller and the example driver. You can then instantiate the other memory controllers for the design and either modify the example driver to create the test pattern for all controllers or create another design example for each controller instantiated.

If you do not need to share any PLL static clocks, you can continue to "Adding Constraints to the Design" on page 7–9.

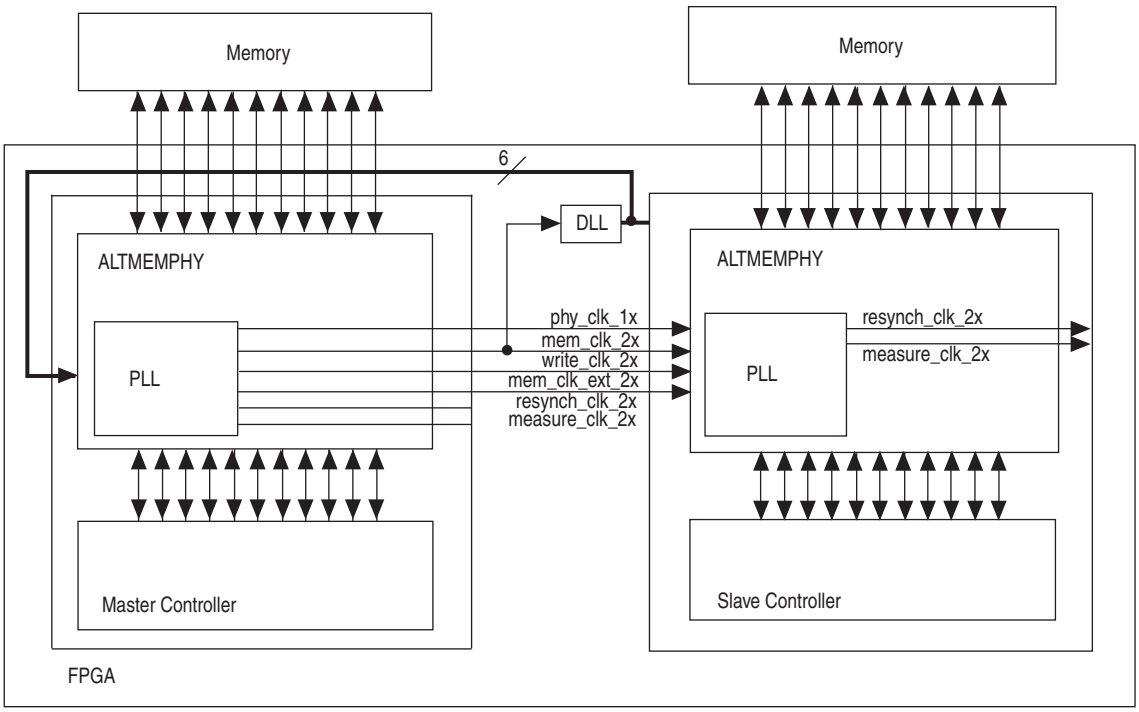
## Sharing PLL Clock Outputs or Clock Networks

The ALTMEMPHY sources a system clock, for use in your design, to clock your ALTMEMPHY interface. This same clock is used by Altera High-Performance controllers, SOPC Builder systems, and the Avalon-MM interface connected to the memory controller. If you use more than one similarly configured ALTMEMPHY in a design, you can share these system clocks, which allows the ALTMEMPHY interfaces to operate synchronously to each other. PLL static clock sharing has the following benefits:

- Although a PLL instance is used for each ALTMEMPHY instance, the static clocks are shared, which uses fewer clocking resources
- The multiple ALTMEMPHYs operate on the same system clock so can be directly connected to logic on the same clock (for example, Avalon-MM interface) without any clock domain crossing logic required, which causes an increase in logic usage and read latency

PLL static clock sharing reduces the required number of clocks by up to four clock networks, which are used in the memory interface logic to clock the controller and generate DQ, DQS, CK/CK#, and address and command signals. Figure 7–2 shows a diagram of the connection between the DLL and the PLL for this scheme.

**Figure 7–2. Two Controllers Static Clocks with Separate Resynchronization Clocks from Different PLLs**




The maximum number of interfaces that can be shared in this scheme is limited by the number of PLLs and the number of pins available in the device. ALTMEMPHY-based memory interfaces may share static clocks between multiple controllers when the following rules are observed:

- All ALTMEMPHYs are full-rate or all half-rate designs
- All ALTMEMPHYs have the same PLL input and output frequencies
- The ALTMEMPHY memory types are compatible. DDR and DDR2 SDRAM can be mixed in the same system; DDR3 SDRAM controllers may be compatible with DDR or DDR2 SDRAM if the static clock phases are identical.

  DDR3 with and without levelling implementations have different static clock phases. Check for the ALTMEMPHY static clock phase compatibility in the clocking tables of the appropriate device and memory interface standard.

- The `ref_clk` input of each ALTMEMPHY PLL must be connected to a clock signal derived from a single clock source. You can use a clock fanout buffer to create separate PLL reference clocks from a single clock source.

- All of the memory controller's circuitry is located in the same two device quadrants
- All of the clocks can be routed as required by their **Global Network Clock Type**. To see the global clocks, look in the Fitter Report > Resource selection > Global and other fast signals.
- The general routing rules defined in the relevant memory standard layout section are met.

Static clocks which can be shared are divided into two categories:

- Static clocks with a fixed phase in the IP (all static clocks except the address and command clock `ac_clk_1x`)
- Static clocks where the phase is set in the IP (the address and command clock `ac_clk_1x`)

Static clocks with a fixed phase in the IP do not require each controller's interface PCB traces to have the same delays. Static clocks with the phase set in the IP have several options where they may be shared between multiple controllers:

- The address and command clock traces all have the same respective delays from the FPGA to each device
- The TPD delay between the address and command signals and the memory clock of each of the interfaces is the same to all devices
- The TPD delay between the address and command signals and the memory clock of each of the interfaces is similar and the optimal address and command phase for each separate interface is within one to two PLL phase steps. Most designs should be able to use a single `ac_clk_1x` clock shared between all controllers and accept less margin.

For multiple controllers, where not all of the memory interface pins are on the same device edge, evaluating whether you can share static clocks is a complex process and there are many possible combinations of pin outs. In this case, you must evaluate the skews between signals across all controllers and factor this into the rules above for sharing clocks.

When static clocks are to be shared select which controller is to be the master. The slave controllers are made to share the master's static clocks by using the **Force Merging of PLL Clock Fanouts** assignments. You need to manually add the two PLL merging assignments using the Assignment Editor or directly update the **.qsf** file.

The **Force Merging of PLL Clock Fanouts** option requires both the master and slave PLL to have the same input reference clock in the RTL. This option limits the placement of the controllers to be only on the same side where one input clock can provide the reference clock for both center PLLs. If your multiple memory controllers' PLLs cannot share the same input reference clock pin, you need to manually connect the clocks together via RTL, instead of using this assignment.

If you are not sharing the address and command clocks, generate the slave controller address and command clocks from the master controller's PLL spare outputs. Create these in the PLL wizard. Modify the PHY source code to bring these clocks out to the top-level file and then modify the slave controllers to connect in the new address and command clock.

Table 7–1 shows a list of PLL merging assignments.

**Table 7–1. PLL Merging Assignments *(Note 1)* (Part 1 of 2)**

| Device | Rate | Assignments |
|---|---|---|
| Arria II GX | Half | set_instance_assignment -name FORCE_MERGE_PLL_FANOUTS ON -from *\|<SLAVE>_phy_alt_mem_phy_clk_reset:clk\|<SLAVE>_phy_alt_mem_phy_pll:*:altpll_component\|*:auto_generated\|clk[0] -to *\|<MASTER>_phy_alt_mem_phy_clk_reset:clk\|<MASTER>_phy_alt_mem_phy_pll:*:altpll_component\|*:auto_generated\|clk[0]<br><br>set_instance_assignment -name FORCE_MERGE_PLL_FANOUTS ON -from *\|<SLAVE>_phy_alt_mem_phy_clk_reset:clk\|<SLAVE>_phy_alt_mem_phy_pll:*:altpll_component\|*:auto_generated\|clk[3] -to *\|<MASTER>_phy_alt_mem_phy_clk_reset:clk\|<MASTER>_phy_alt_mem_phy_pll:*:altpll_component\|*:auto_generated\|clk[3] |
| | Full | set_instance_assignment -name FORCE_MERGE_PLL_FANOUTS ON -from *\|<SLAVE>_phy_alt_mem_phy_clk_reset:clk\|<SLAVE>_phy_alt_mem_phy_pll:*:altpll_component\|*:auto_generated\|clk[1] -to *\|<MASTER>_phy_alt_mem_phy_clk_reset:clk\|<MASTER>_phy_alt_mem_phy_pll:*:altpll_component\|*:auto_generated\|clk[1]<br><br>set_instance_assignment -name FORCE_MERGE_PLL_FANOUTS ON -from *\|<SLAVE>_phy_alt_mem_phy_clk_reset:clk\|<SLAVE>_phy_alt_mem_phy_pll:*:altpll_component\|*:auto_generated\|clk[3] -to *\|<MASTER>_phy_alt_mem_phy_clk_reset:clk\|<MASTER>_phy_alt_mem_phy_pll:*:altpll_component\|*:auto_generated\|clk[3] |
| Cyclone III | Half | set_instance_assignment -name FORCE_MERGE_PLL_FANOUTS ON -from *\|<SLAVE>_phy_alt_mem_phy_clk_reset:clk\|<SLAVE>_phy_alt_mem_phy_pll:*:altpll_component\|*:auto_generated\|clk[0] -to *\|<MASTER>_phy_alt_mem_phy_clk_reset:clk\|<MASTER>_phy_alt_mem_phy_pll:*:altpll_component\|*:auto_generated\|clk[0]<br><br>set_instance_assignment -name FORCE_MERGE_PLL_FANOUTS ON -from *\|<SLAVE>_phy_alt_mem_phy_clk_reset:clk\|<SLAVE>_phy_alt_mem_phy_pll:*:altpll_component\|*:auto_generated\|clk[2] -to *\|<MASTER>_phy_alt_mem_phy_clk_reset:clk\|<MASTER>_phy_alt_mem_phy_pll:*:altpll_component\|*:auto_generated\|clk[2] |
| | Full | set_instance_assignment -name FORCE_MERGE_PLL_FANOUTS ON -from *\|<SLAVE>_phy_alt_mem_phy_clk_reset:clk\|<SLAVE>_phy_alt_mem_phy_pll:*:altpll_component\|*:auto_generated\|clk[1] -to *\|<MASTER>_phy_alt_mem_phy_clk_reset:clk\|<MASTER>_phy_alt_mem_phy_pll:*:altpll_component\|*:auto_generated\|clk[1]<br><br>set_instance_assignment -name FORCE_MERGE_PLL_FANOUTS ON -from *\|<SLAVE>_phy_alt_mem_phy_clk_reset:clk\|<SLAVE>_phy_alt_mem_phy_pll:*:altpll_component\|*:auto_generated\|clk[2] -to *\|<MASTER>_phy_alt_mem_phy_clk_reset:clk\|<MASTER>_phy_alt_mem_phy_pll:*:altpll_component\|*:auto_generated\|clk[2] |

**Table 7–1. PLL Merging Assignments *(Note 1)* (Part 2 of 2)**

| Device | Rate | Assignments |
|---|---|---|
| Stratix III and Stratix IV | Half and Full | `set_instance_assignment -name FORCE_MERGE_PLL_FANOUTS ON -from *\|<SLAVE>_phy_alt_mem_phy_clk_reset:clk\|write_clk_2x -to *\|<MASTER>_phy_alt_mem_phy_clk_reset:clk\|write_clk_2x`<br><br>`set_instance_assignment -name FORCE_MERGE_PLL_FANOUTS ON -from *\|<SLAVE>_phy_alt_mem_phy_clk_reset:clk\|phy_clk_1x -to *\|<MASTER>_phy_alt_mem_phy_clk_reset:clk\|phy_clk_1x` |

**Notes to Table 7–1:**

(1) In these assignments, if you are using the ALTMEMPHY megafunction, you must replace `<SLAVE>_phy` and `<MASTER>_phy` by the variation names of your two variations. If you are using the high performance controller, you must replace `<SLAVE>` and `<MASTER>` by the variation names of your two variations.

To check that the Quartus II software has applied the assignments, read the Compilation Report - PLL Usage (available only if the fitter step is successful), which shows the two clocks merged. This report allows you to compare the PLL usage before PLL merging and after PLL merging.

If the report does not show the clocks merged as expected you should check the `FORCE_MERGE_PLL_FANOUTS` assignment carefully for incorrect clock names. You can also open your <*projectname*>**_fit.rpt** file and look for `Merged PLL`.

# Adding Constraints to the Design

The MegaWizard Plug-In Manager generates an **.sdc** file for timing constraints and **.tcl** scripts for I/O standard and DQS/DQ grouping constraints of each controller. TimeQuest™ is the default timing analyzer for the Arria GX, Stratix III, and Cyclone III device families. To optimize timing with the Quartus II compiler and to use the timing report script created by the MegaWizard Plug-In Manager manually, you must enable the **TimeQuest Timing Analyzer**.

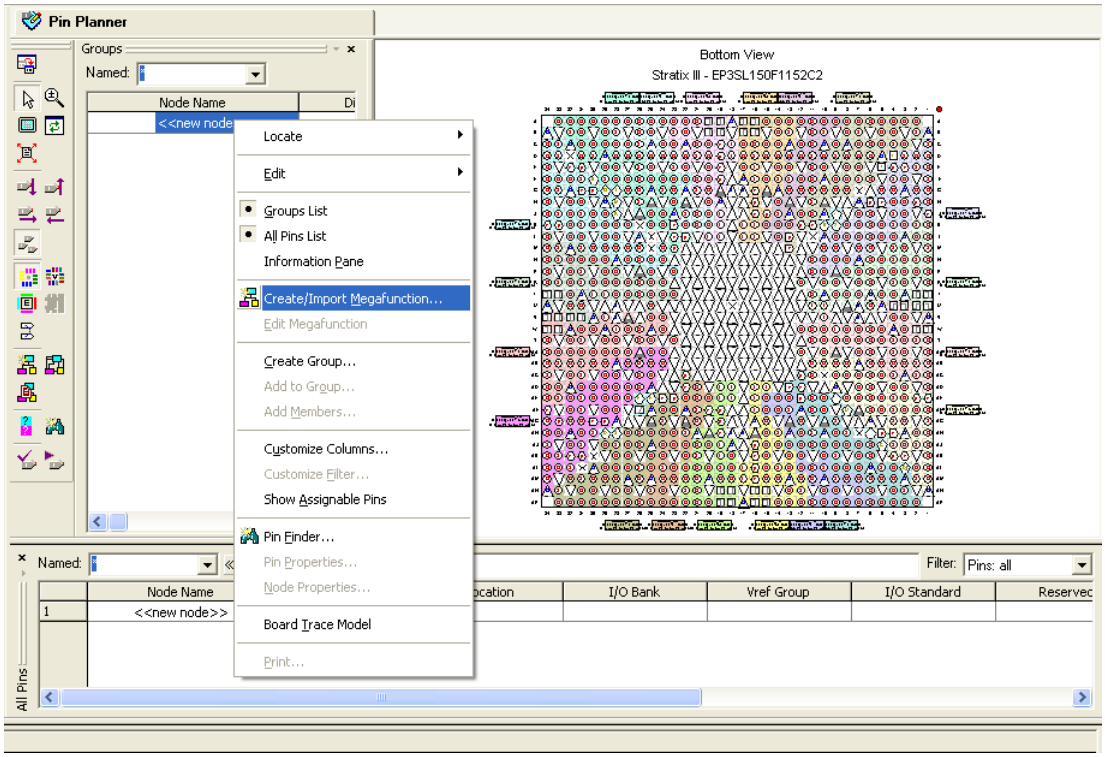
You must then add the **.sdc** file for each controller. If you are using two or more different memory controller variations, then you need to add the .sdc files generated for each variation. If, however, your design consists of multiple instantiations of the same controller, you only need to add the .sdc file once without modification. The Quartus II software is able to apply the .sdc file for all the interfaces using the same variation.

The High-Performance Memory Controller MegaWizard Plug-In Manager also generates two .tcl scripts per variation to set the I/O standard, output enable grouping, termination, and current strength for the memory interface pins. These .tcl scripts use the default MegaWizard Plug-In Manager names; for example, `mem_dq[71..0]`, `local_ready`, and `mem_ras_n`. Every variation of the high-performance memory controller uses this naming convention. However, if there are multiple memory controllers in the design, you must change the names of the interface pins in the top-level file to differentiate each controller. You can add prefixes to the controller pin names in the Quartus II Pin Planner by following these steps:

1. Open the Assignment menu and click on Pin Planner.

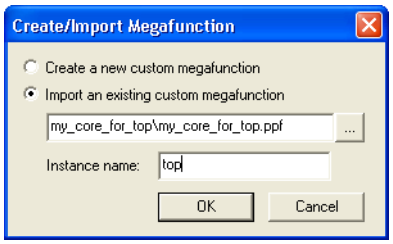
2. Right-click in any area under Node Name, and select Create/Import Megafunction (Figure 7–3).

**Figure 7–3. Create/Import Megafunction Option in the Pin Planner**



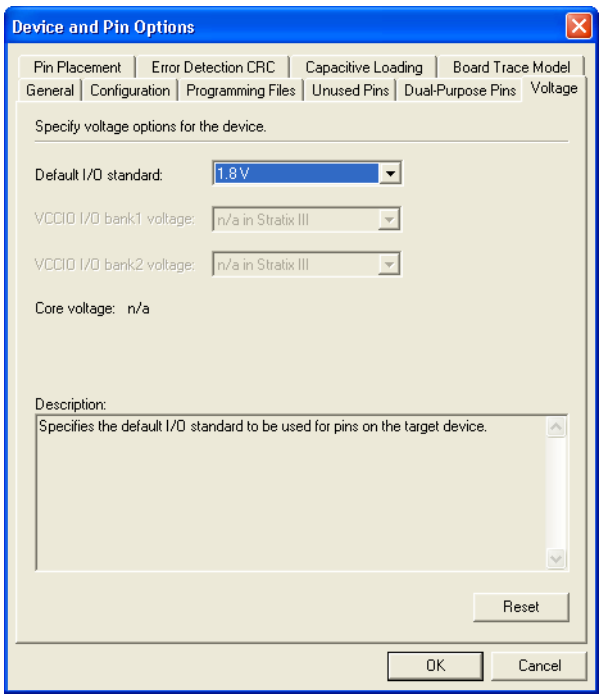

3. Turn on the Import an existing custom megafunction radio button and choose the *<variation_name>***.ppf** file.
4. Type the prefix that you want to use under the Instance name (Figure 7–4).

**Figure 7–4. Adding Prefix to the Memory Interface Pin Names**



You may also want to set the default I/O standard for your design in the **Voltage** section of the **Device and Pin** Options, by navigating to **Assignment and clicking Setting** (Figure 7–5).

**Figure 7–5. Setting a Default Voltage for Your Design**




Be aware of the number of available pins per I/O bank to ensure that the Quartus II software can successfully compile the design.

# Compiling the Design to Generate a Timing Report

During design compilation, the Quartus II software analyzes the timing margins for the memory controllers across the three timing model corners. As an alternative, you can also open the Tools menu and click **TimeQuest Timing Analyzer**. Double click on **Report DDR** to get the timing report for all ALTMEMPHY-based memory controllers in the design for the timing model that you select in the TimeQuest timing analyzer.

For more information about analyzing memory interface timing in Stratix III and Cyclone III devices, refer to the *Timing Analysis* section in volume 4 of the *External Memory Interface Handbook*.

# Timing Closure

You may encounter some of the following timing failures:

- If read capture setup time is negative, you need to decrease the delay chain used in the DQ pins by setting the **Input Delay from Pin to Input Register** to 0 or a lower number than the current setting in the **Assignment Editor**. However, if hold time is negative, you need to do the opposite; increment the Input Delay from Pin to Input Register setting to a higher number than the current setting in the Assignment Editor.
- If you are experiencing any write and address/command timing failures, you may be able to resolve them by setting:

```
set_instance_assignment -name CLOCK_TO_OUTPUT_DELAY 0 -to
<address/command/CK/CK# pin>
```

- If the resynchronization path is not meeting timing, you must move the resynchronization registers closer to the IOE. Report DDR shows information messages with the names of the source and destination registers of the worst case setup path. You must copy the name of the destination register from the message and move it as close as possible to the source register using the Assignment Editor. Similarly, if the postamble path is not meeting recovery timing, move the postamble registers closer to the IOE.
- Setup failures for transfers from the measure clock (`clk5`) to the PHY clock (`clk0` for half-rate interfaces or `clk1` for full-rate interfaces) should be ignored. Due to the dynamic nature of the measure clock these should be treated as asynchronous transfers.

For more information about other issues that may cause timing failures in your design, refer to the latest version of the *Quartus II Release Notes*.

# External Memory Interface Handbook Volume 6

## Section II. UniPHY Design Tutorials

101 Innovation Drive
San Jose, CA 95134
www.altera.com

EMI_TUT_QDR-2.1

Document last updated for Altera Complete Design Suite version: 10.1
Document publication date: December 2010

Subscribe

# Contents

# 1. Using QDR II and QDR II+ SRAM Controller with UniPHY in Arria II GX, Stratix III and Stratix IV Devices

This tutorial describes how to use the design flow to implement an 18-bit wide, 400-MHz, 1600-Mbps QDR II+ SRAM controller with UniPHY interface using Stratix® III devices, and an 18-bit wide, 400-MHz, 1400-Mbps QDR II+ SRAM controller with UniPHY interface using Stratix IV devices. This tutorial also provides some recommended settings to simplify the design.

The design flow is also applicable for a QDR II SRAM controller with UniPHY interface.

The design examples target the Stratix III FPGA development kit, which includes an 18-bit wide, 36-Mb, 4-word burst, Cypress CY7C1263V18 400-MHz QDR II+ SRAM memory component, and the Stratix IV GX FPGA development kit, which includes an 18-bit wide, 72-Mb, 4-word burst, Cypress CY7C1563V18 400-MHz QDR II+ SRAM memory component.

You can also use these design examples if you are targeting an Arria® II GX, HardCopy® III, or HardCopy IV device for your design.

To download the design examples, **emi_qdrii_plus_siii.zip** and **emi_qdrii_plus_siv.zip**, go to the External Memory Interface Design Examples page.

For more information about the design flow, refer to the *Recommended Design Flow* section in volume 1 of the *External Memory Interface Handbook*. For more information about the QDR II and QDR II+ SRAM controller with UniPHY, refer to the *QDR II and QDR II+ SRAM Controller with UniPHY* section in volume 3 of the *External Memory Interface Handbook*.

## System Requirements

This tutorial assumes that you have experience with the Quartus® II software. This tutorial requires the following software:

- Quartus II software version 10.0 or later.
- QDR II and QDR II+ SRAM Controllers with UniPHY version 10.0.
- ModelSim®-Altera® version 6.5b or later.

## Create a Quartus II Project

Create a project in the Quartus II software that targets the respective device: a EP3SL150F1152C2 device for the Stratix III device family, or a EP4SGX230KF40C2 device for the Stratix IV GX device family.

For detailed step-by-step instructions to create a Quartus II project, refer to the Quartus II software tutorial by clicking the Help menu in the Quartus II window and selecting **PDF Tutorials**.

# Instantiate and Parameterize a Controller

After creating a Quartus II project, instantiate the QDR II and QDR II+ SRAM controller with UniPHY and its parameters.

## Instantiate a Controller

To instantiate the controller with UniPHY, perform the following steps:

1. Start the MegaWizard™ Plug-In Manager.
2. In the MegaWizard Plug-In Manager, expand **External Memory** in the **Interfaces** folder and select **QDR II and QDR II+ SRAM Controller with UniPHY v10.0**.
3. Type `qdrii_uniphy` for the name of the QDR II and QDR II+ SRAM controller with UniPHY.

## Parameterize QDR II+ SRAM Controller with UniPHY Interface

To parameterize the QDR II and QDR II+ SRAM controller with UniPHY for a 400-MHz, 18-bit wide QDR II+ SRAM interface with Stratix III or Stratix IV devices, perform the following steps:

1. In the **Presets** list, select a memory preset from the list of presets that matches your memory based on your memory device specifications. If your memory device is not available in preset list, you have to modify the parameters based on your memory datasheet. Select CY7C1263V18-400 from the preset list for Stratix III design, or select CY7C1563KV18-400 (which meets the requirements for CY7C1563V18-400) from the preset list for Stratix IV design, and click **Apply**.

   CAUTION Selecting a new memory preset overwrites your existing settings. You need to enter the settings again.

2. In the **General Settings** tab, under **Clocks**, for **Memory clock frequency**, type **400** MHz as the system frequency.
3. For **PLL reference clock frequency**, type **125** MHz for Stratix III design, or **50** MHz for Stratix IV design.
4. For **Full or half rate on Avalon-MM interface**, select **Half**.
5. For **Additional Address/Command clock phase**, select **0**.
6. Under **Advanced Settings**, for **Maximum Avalon-MM burst length**, specify the burst length based on your system. For example, if your system only generates up to 64 beats, then select **64**.

   If you are using the SOPC Builder system, turn on **Generate power-of-2 bus widths**.

7. For **I/O standard**, select **1.8-V HSTL** for Stratix III design, or **1.5-V HSTL** for Stratix IV design.
8. Turn on **Master for PLL/DLL sharing** if you want the UniPHY to instantiate its own PLL. If the **Master for PLL/DLL sharing** option is disabled, the PLL clock shares with other identical UniPHY core.

If you want to migrate your design to a HardCopy device, turn on **HardCopy Compatibility Mode**, and select the appropriate **Reconfigurable PLL Location**.

9. Under **Example Testbench Simulation Options**, turn on **Skip memory initialization**. Enabling this option allows you to skip the memory initialization sequence during simulation.
10. Under **Controller Settings**, select **1** for **Controller Latency**, as the maximum clock rates are selected for these design examples.
11. In the **Board Settings** tab, set the **Board Skews** parameters to the specified values in Table 1–1.

**Table 1–1. Board Skews Parameters for EP3SL150F1152C2 and EP4SGX230KF40C2 Devices**

| Board Skews Parameter | Stratix III Value (ps) | Stratix IV Value (ps) |
|---|---|---|
| Maximum delay difference between devices | 0 | **0** |
| Maximum skew within K group | 20 | **10** |
| Maximum skew between K groups | 0 | **0** |
| Average delay difference between Address/Command and K | 1 | **5** |
| Average delay difference between Data and K | 3 | **17** |
| Average delay difference between Data and CQ | 9 | **3** |
| Maximum skew within Address/Command bus | 81 | **27** |

The Intersymbol Interference (ISI) parameters are not applicable for single chip-select configurations. Set these parameters to **0 ns**.

For more accurate timing analysis, assign the board trace delay model for every single pin. The board skew values are used to calculate the overall system timing margin. Change these values based on the performance of your board if you are not using the board trace delay model, or if you cannot accurately assign the board trace model for each pin. Refer to "Enter Board Trace Delay Models" on page 1–6 for more information about board trace delay models.

For more information about layout guidelines for the QDR II and QDR II+ SRAM components, refer to the *QDR II Interface Termination and Layout Guidelines* chapter in volume 2 of the *External Memory Interface Handbook*.

12. Click **Finish** to generate your MegaCore function variation. The MegaWizard Plug-In Manager generates all the files necessary for your QDR II and QDR II+ SRAM controller with UniPHY, and an example top-level design, which you may use to test or verify board operation.

# Add Constraints

When you instantiate a QDR II and QDR II+ SRAM controller with UniPHY, the Quartus II software generates the constraints files for the design example. Apply these constraints to the design before compilation.

## Add Example Project

Add the example project files to your project by performing the following steps:

1. On the Project menu, click **Add/Remove Files in Project**.
2. Browse to the <*variation_name*>**\example_project** directory.
3. Select all the files with the <*variation_name*> prefix, except for the <*variation_name*>**_mem_model.sv** file, and click **Open**.
4. Click **OK**.

## Set Top-Level Entity

The top-level entity of the project must be set to the correct entity. To set the top-level file, perform the following steps:

1. On the File menu, click **Open**.
2. Browse to the <*variation_name*>**_example_top.v** or **.vhd** file and click **Open**.
3. On the Project menu, click **Set as Top-Level Entity**.

## Add Pin and DQ Group Assignments

The pin assignment script, <*variation_name*>**_pin_assignments.tcl**, sets up the I/O standards for the QDR II+ SRAM with UniPHY interface. This script does not include the assignment for the design's PLL input clock. You must create a PLL input clock for the design and provide pin assignments for the signals of both the example driver and testbench that the MegaCore variation generates.

Run the <*variation_name*>**_pin_assignments.tcl** script to add the pin, I/O standards, and DQ group assignments to the design example. To run the pin assignment scripts, perform the following steps:

1. On the Processing menu, point to **Start** and click **Start Analysis & Synthesis**.
2. In the Tools menu, click **Tcl Scripts**.
3. Locate the <*variation_name*>**_pin_assignments.tcl** file.
4. Click **Run**.

The PLL input clock I/O does not need to have the same I/O standard as the memory interface I/Os. However, you may see a no-fit error as the bank in which the PLL input clock I/O gets placed becomes unusable for placement of the memory interface I/Os because of the incompatible $V_{CCIO}$ levels. Altera recommends that you assign the same I/O standard to the PLL input clock I/O.

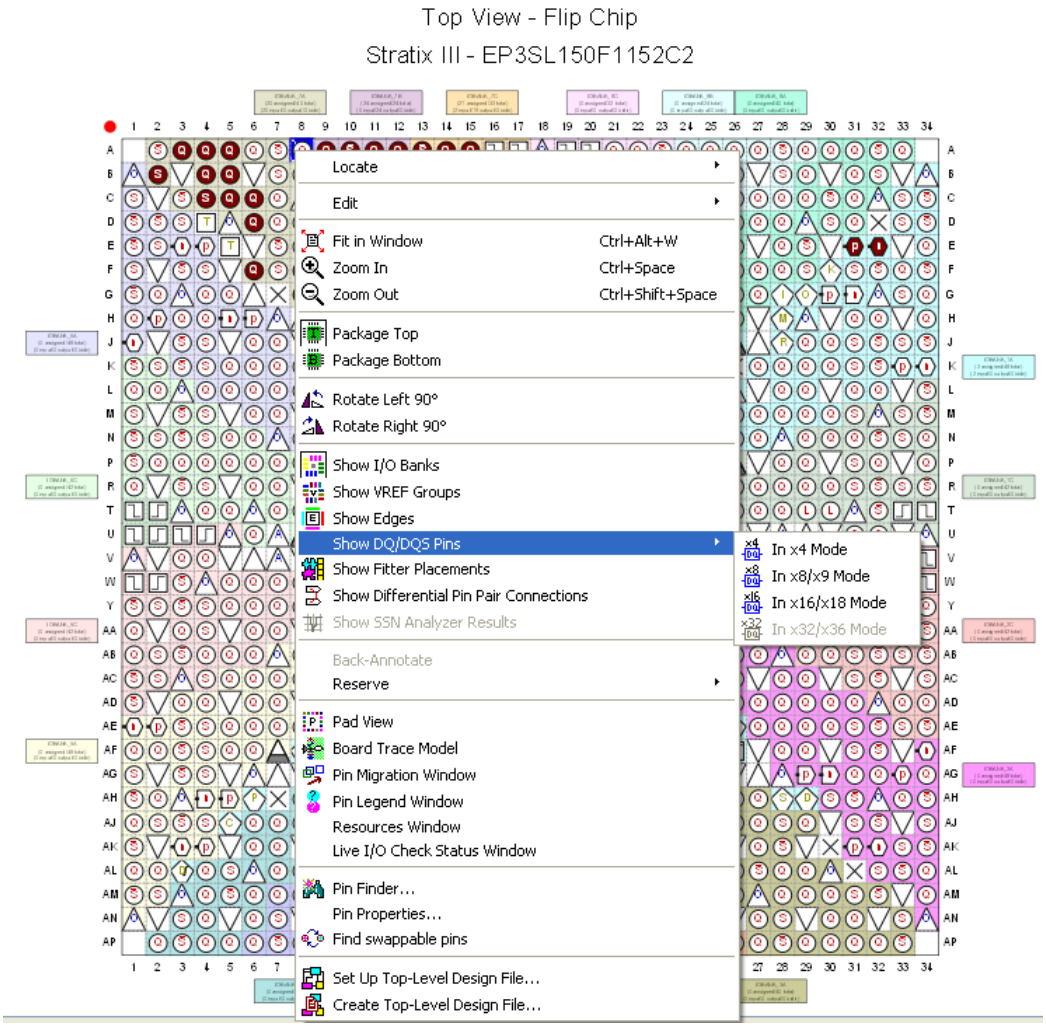
## Enter Pin Location Assignments

To enter the pin location assignments, assign all of your pins, so the Quartus II software fits your design correctly and gives correct timing analysis. To assign the pin locations, run the Altera-provided **SIII_qdrii_pin_location.tcl** file for the Stratix III FPGA development kit, and **SIV_GX_qdrii_pin_location.tcl** file for the Stratix IV GX FPGA development kit.

Alternatively, to manually assign the pin locations in the Pin Planner, perform the following steps:

1. On the Assignments menu, click **Pin Planner**.
2. To view the DQS groups in the Pin Planner, right click, select **Show DQ/DQS Pins**, and click **In ×16/×18 Mode**. The Pin Planner shows each DQS group in a different color and with a different legend: S = `DQS` pin, Sbar = `DQSn` pin and Q = `DQ` pin, as shown in Figure 1–1.

**Figure 1–1. Quartus II Pin Planner, Show DQ/DQS Pins, In ×16/×18 Mode**




3. To identify the differential I/O pairs in the Pin Planner, right-click and select **Show Differential Pin Pair Connections**. The pin pairs show a red line between each pin pair.

For more information about pin location assignments, refer to the *Device and Pin Planning* section in volume 2 of the *External Memory Interface Handbook*.

## Assign Virtual Pins

The example top-level design, which is auto-generated by the QDR II and QDR II+ SRAM controller with UniPHY, includes an example driver to simulate the interface. This example driver is not part of the QDR II and QDR II+ SRAM controller with UniPHY IP, but allows easy testing of the IP.

The example driver outputs several test signals to indicate its operation and the status of the simulated memory interface. These test signals are `afi_cal_success`, `afi_cal_fail`, `pass`, `fail`, and `test_complete`. The test signals are not part of the memory interface, but are to facilitate testing. You must connect these signals to a debug bus or assign the signals to virtual pins using the Quartus II Assignment Editor. When using the example driver for testing, do not remove the test signals from the top-level signal list. If you remove the test signals from the top-level module, the Quartus II software optimizes the driver away, and the example driver fails.

To assign the virtual pins for the Stratix III FPGA development kit, run the Altera-provided **SIII_qdrii_exdriver_vpin.tcl** file, and for the Stratix IV GX FPGA development kit, run the Altera-provided **SIV_GX_qdrii_exdriver_vpin.tcl** file, or manually assign the virtual pins using the Assignment Editor.

## Enter Board Trace Delay Models

For accurate I/O timing analysis, the Quartus II software must be aware of the board trace and loading information. You should derive and refine board trace and loading information during your PCB development process of prelayout (line) and post-layout (board) simulation. For external memory interfaces that use memory modules (DIMMs), this information should include the trace and loading information of the module in addition to the main and host platform, which you can obtain from your memory vendor.

To enter the board trace information, perform the following steps:

1. In the Pin Planner, select the pin or group of pins for which you want to enter board trace parameters.
2. Right-click and select **Board Trace Model**.

Altera recommends that you use the board trace model assignment on all QDR II and QDR II+ SRAM interface signals. To apply the board trace model assignments for the Stratix III FPGA development kit, run the Altera-provided **SIII_qdrii_board_trace_models.tcl** file, and for the Stratix IV GX FPGA development kit, run the Altera-provided **SIV_GX_qdrii_board_trace_models.tcl** file. You can also manually assign the board trace model values using the Pin Planner.

Table 1–2 shows the board trace model parameters for the Stratix III FPGA and Stratix IV GX FPGA development kits.

**Table 1–2. QDR II+ SRAM Board Trace Model Summary for Stratix III and Stratix IV GX FPGA Development Kits**

| Net | Near (FPGA End of Line) | | | | | | Far (Memory End of Line) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Length (in) | C_per_ length (pF/in) | L_per_ length (nH/in) | Cn (pF) | Rns (W) | Rnh (W) | Length (in) | C_per_ length (pF/in) | L_per_ length (nH/in) | Cf (pF) | Rfh (W) |
| **Stratix III** | | | | | | | | | | | |
| Add_Cntl | 2.947 | 3.36 | 7.77 | — | — | — | — | — | — | 5 | 56 |
| bws_n | 2.572 | 3.35 | 7.90 | — | — | — | — | — | — | 5 | 56 |
| K | 2.722 | 3.38 | 8.13 | — | — | — | — | — | — | 4 | 56 |
| D Group | 3.035 | 3.36 | 7.77 | — | — | — | — | — | — | 5 | 56 |
| **Stratix IV GX** | | | | | | | | | | | |
| Add_Cntl | 2.424 | 4.25 | 7.61 | — | — | — | — | — | — | 5 | — |
| bws_n | 2.309 | 4.25 | 7.61 | — | — | — | — | — | — | 5 | — |
| K | 2.282 | 4.11 | 7.84 | — | — | — | — | — | — | 6 | — |
| doffn | 1.578 | 4.25 | 7.61 | — | — | 10 k | — | — | — | 5 | — |
| D Group | 2.406 | 4.25 | 7.61 | — | — | — | — | — | — | 5 | — |

# Perform RTL Simulation (Optional)

This section describes RTL (functional) simulation.

To perform the functional simulation of your memory interface, use the functional model of the UniPHY generated by the MegaWizard Plug-In Manager. You must use this model along with your own driver or testbench that issues the read and write operations, and a memory model.

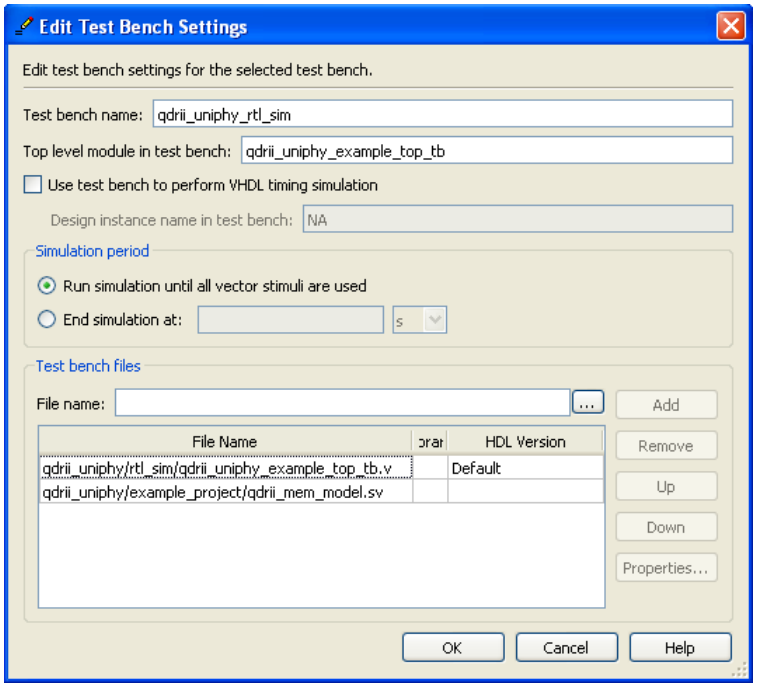
This design example includes the driver, testbench, and the memory model that allows you to perform functional simulation on the design. Use the ModelSim-Altera simulator to perform the functional simulation. This design example also includes a script file that directs the simulator to perform the necessary compilation and simulation.

To run the RTL simulation with NativeLink, perform the following steps:

1. Set the absolute path to your third-party simulator executable file, by performing the following steps:
   a. On the Tools menu, click **Options**.
   b. In the **Category** list, select EDA Tools Options and set the default path for **ModelSim-Altera** to **C:\<version>\modelsim_ae\win32aloem**.
   c. Click **OK**.
2. On the Assignments menu, click **EDA Tool Settings**.
3. In the **Category** list, expand **EDA Tool Settings** and click **Simulation**.
4. Under **Tool name**, select **ModelSim-Altera**.

5. Under **NativeLink settings**, select **Compile test bench** and click **Test Benches**.
6. Click **New**.
7. In the **Edit Test Bench Settings** dialog box, perform the following steps:
   a. Type the testbench name, `qdrii_uniphy_rtl_sim`.
   b. Type the top-level module file name, `qdrii_uniphy_example_top_tb`.
   c. Under **Simulation period**, select **Run simulation until all vector stimuli are used**.
   d. In the **Test bench files** field, include the testbench file, <*variation name*>**_example_top_tb.v**, and the memory model file, <*variation name*>**_mem_model.sv** (Figure 1–2).
   e. Click **OK**.

**Figure 1–2. Testbench Settings**




8. To elaborate your design, on the Processing menu, point to **Start** and click **Start Analysis & Elaboration**.
9. On the Tools menu, point to the **Run EDA Simulation Tool** and click **EDA RTL Simulation**. This step creates the **\simulation** directory in your project directory and a script that compiles all necessary files and runs the simulation.

# Compile Design and Verify Timing

To compile the design, on the Processing menu, click **Start Compilation**.

After successfully compiling the design, the Quartus II software automatically runs the verify timing script, <*variation_name*>**_report_timing.tcl**, which produces a timing report for the design together with the compilation report.

Figure 1–3 shows the timing margin report in the message window in the Quartus II software.

**Figure 1–3. Timing Margin Report in the Quartus II Software**

```
Info: Core: qdrii_uniphy - Instance: mem_if
Info:                                                          setup  hold
Info: Address Command (Fast 1100mV 0C Model)                 |  0.491  0.584
Info: Core (Fast 1100mV 0C Model)                            |  0.513  0.091
Info: Core Recovery/Removal (Fast 1100mV 0C Model)           |   1.06  0.298
Info: Read Capture CQ (All Conditions)                       |  0.019  0.002
Info: Read Capture CQn (All Conditions)                      |  0.019  0.002
Info: Read Resync (Fast 1100mV 0C Model)                     |  6.279 65.327
Info: Write (All Conditions)                                 |  0.066  0.059
Info: Design is not fully constrained for setup requirements
Info: Design is not fully constrained for hold requirements
```

The report timing script performs the following tasks:

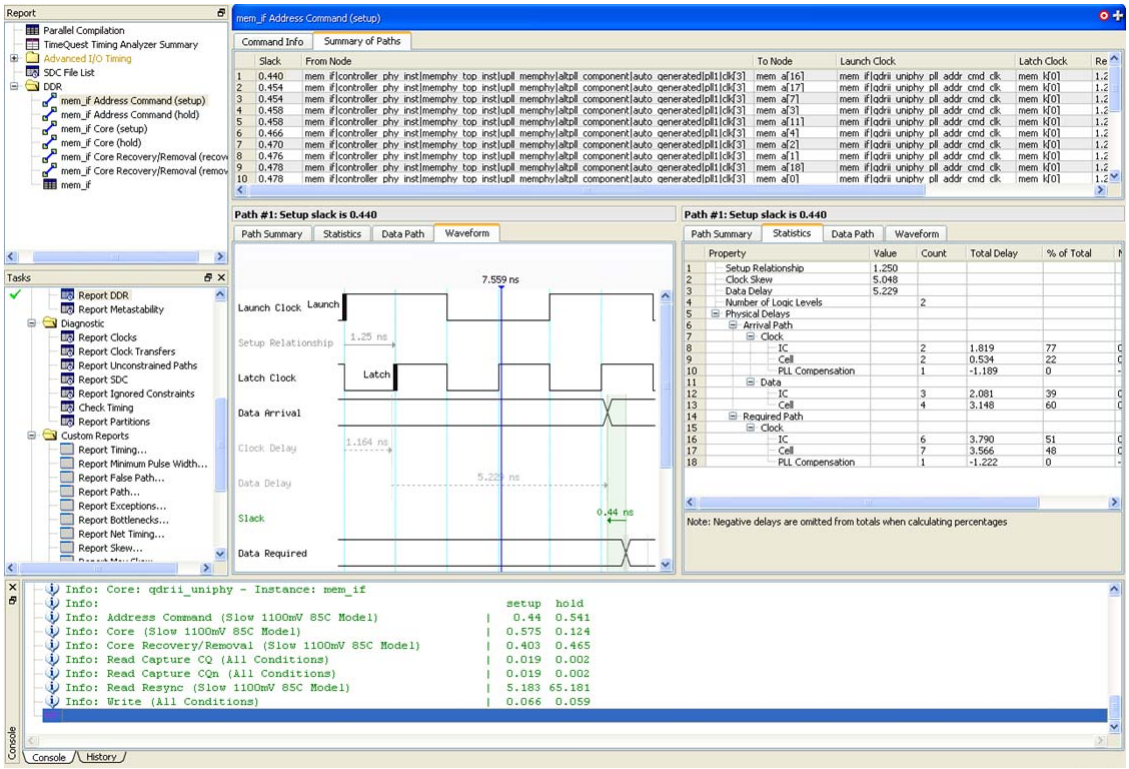
1. Creates a timing netlist.
2. Reads the <*variation_name*>**.sdc** file.
3. Updates the timing netlist.

You can also obtain the timing report by running the report timing script, <*variation_name*>**_timing.tcl**, in the TimeQuest timing analyzer. To obtain the timing report in the TimeQuest Timing Analyzer window, perform the following steps:

1. On the Tools menu, click **TimeQuest Timing Analyzer**.
2. On the **Tasks** pane, double-click **Report DDR** to run **Update Timing Netlist**, **Create Timing Netlist**, and **Read SDC File**. This command subsequently executes the report timing script to generate the timing margin report.

Figure 1–4 shows the timing margin report in the TimeQuest Timing Analyzer window after running the report timing script. The results are the same as the Quartus II software results.

**Figure 1–4. Timing Margin Report in the TimeQuest Timing Analyzer**




For more information about the TimeQuest timing analyzer, refer to *The Quartus II TimeQuest Timing Analyzer* chapter in volume 3 of the *Quartus II Handbook*. For information timing analysis, refer to the *Timing Analysis* section in volume 4 of the *External Memory Interface Handbook*.

# Verify FPGA Functionality

The SignalTap® II Embedded Logic Analyzer shows read and write activity in the system.

For more information about using the SignalTap II Embedded Logic Analyzer, refer to the *Design Debugging Using the SignalTap II Embedded Logic Analyzer* chapter in the *Quartus II Handbook*, *AN 323: Using SignalTap II Embedded Logic Analyzers in SOPC Builder Systems* and *AN 446: Debugging Nios II Systems with the SignalTap II Embedded Logic Analyzer*.

To add the SignalTap II Embedded Logic Analyzer to your Quartus II project, perform the following steps:

1. On the Tools menu, click **SignalTap II Logic Analyzer**.
2. Under **Signal Configuration** next to the **Clock** box, click **...** (Browse Node Finder).

3. Turn on **Include subentities** to include all sub-entities in the hierarchy.
4. Type `*afi_clk` in the **Named** box, for **Filter** select **SignalTap II: pre-synthesis** and click **List**.
5. Select **<variation_name>:mem_if|afi_clk** in **Nodes Found** and click **>** to add the signal to **Selected Nodes**.
6. Click **OK**.
7. Under **Signal Configuration**, specify the following settings:
   - For **Sample depth**, select **512**.
   - For **RAM type**, select **Auto**.
   - For **Trigger flow control**, select **Sequential**.
   - For **Trigger position**, select **Center trigger position**.
   - For **Trigger conditions**, select **1**.
8. On the Edit menu, click **Add Nodes**.
9. In the **Named** box, search for specific nodes by typing *, for **Filter** select **SignalTap II: pre-synthesis** and click **List**.
10. In **Nodes Found**, select the following nodes and click **>** to add to **Selected Nodes**:
    - **avl_addr**
    - **avl_read_req**
    - **avl_ready**
    - **avl_write_req**
    - **fail**
    - **pass**
    - afi_cal_fail
    - afi_cal_success
    - **test_complete**
    - **be_reg**
    - pnf_per_bit
    - rdata_reg
    - rdata_valid_reg
    - data_out

    
    Do not add any QDR II+ SRAM interface signals to the SignalTap II Embedded Logic Analyzer. The load on these signals increases and adversely affects the timing analysis.

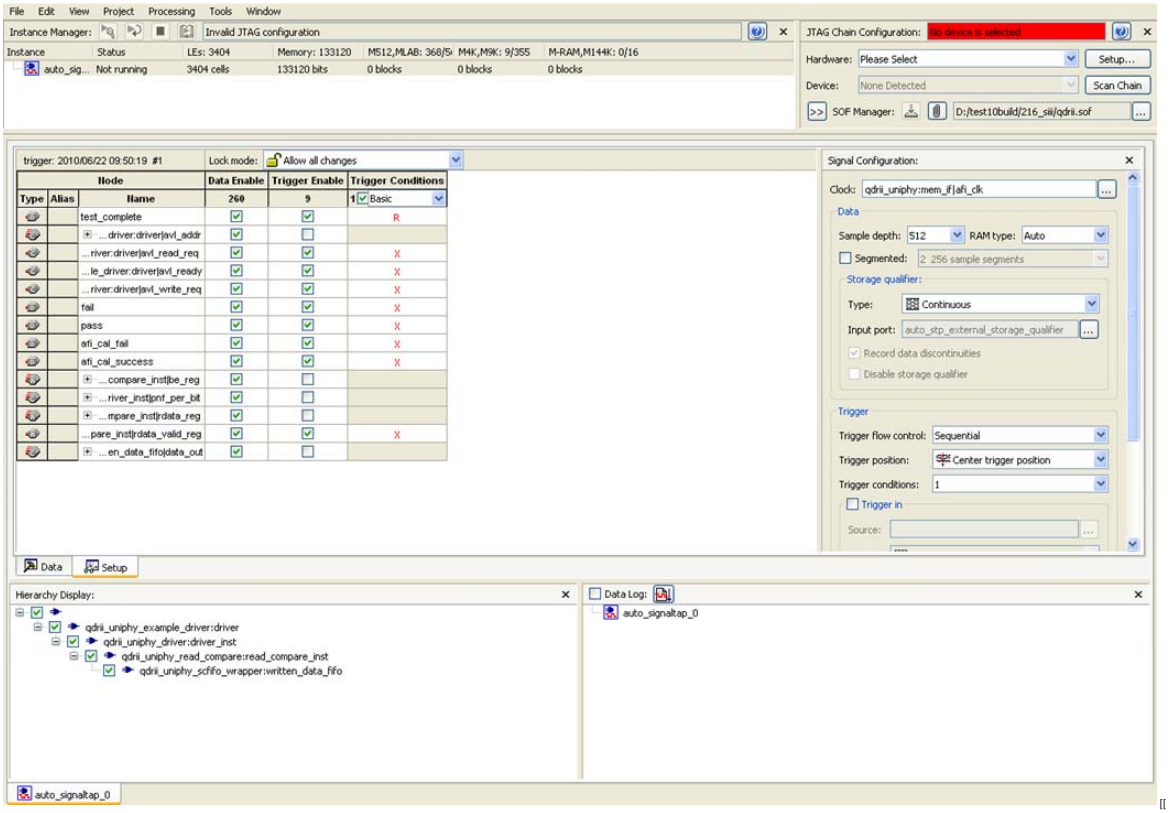
11. Click **OK**.

12. To reduce the SignalTap II logic size, turn off **Trigger Enable** on the following bus signals:
    - **avl_addr**
    - **data_out**
    - **pnf_per_bit**
    - **rdata_reg**
    - **be_reg**
13. Right-click **Trigger Conditions** for the `test_complete` signal and select **Rising Edge**.
14. On the File menu, click **Save** to save the SignalTap **.stp** file to your project.

If you see the message **Do you want to enable SignalTap II file "stp1.stp" for the current project?**, click **Yes**.

Figure 1–5 shows the completed SignalTap II Embedded Logic Analyzer after you perform all the steps.

**Figure 1–5. SignalTap II Embedded Logic Analyzer**




## Compile the Project

After you add signals to the SignalTap II Embedded Logic Analyzer, on the Processing menu, click **Start Compilation** to recompile your design.

## Verify Timing

After the design recompiles, ensure that the TimeQuest timing analysis passes successfully. In addition to this FPGA timing analysis, check your PCB or system SDRAM timing. To perform timing analysis, run the *<variation name>*_**report_timing.tcl** script by performing the following steps:

1. On the Tools menu, click **Tcl Scripts**.
2. Select *<variation name>*_**report_timing.tcl**.
3. Click **Run**.

## Download the Object File

To download the object file to the device, perform the following steps:

1. Connect the development board to your computer.
2. On the Tools menu, click **SignalTap II Logic Analyzer**. The SignalTap II dialog box appears.
3. Click **...** to open the **Select Program Files** dialog box.
4. Select *<your project name>***.sof**.
5. Click **Open**.
6. To download the file to the device, click the **Program Device** button.

## Test the Design Example in Hardware

When the design example including SignalTap II Embedded Logic Analyzer successfully downloads to your development board, click **Run Analysis** to run the analysis once, or click **Autorun Analysis** to run the analysis continuously.

The design analysis in Figure 1–6 and Figure 1–7 show that the `pnf_per_bit` and `pass` signals are high, indicating that the test passes in hardware.

**Figure 1–6. SignalTap II Example QDR II and QDR II+ SRAM Design Analysis Using Stratix III Devices**

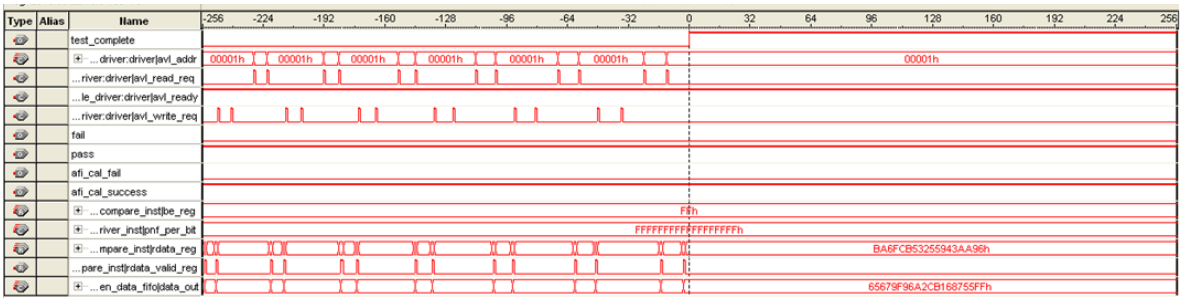
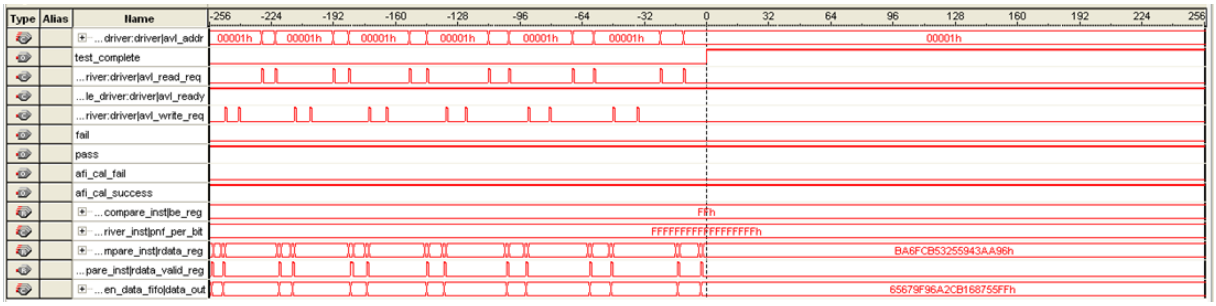
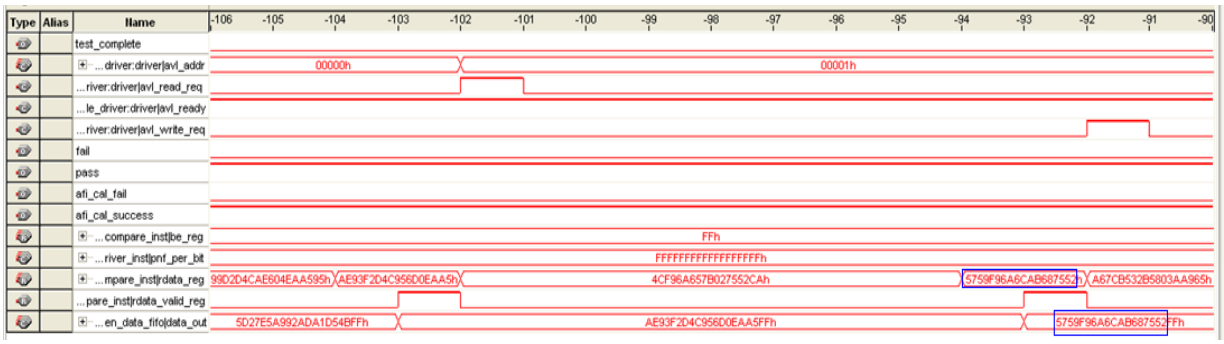



**Figure 1–7. SignalTap II Example QDR II and QDR II+ SRAM Design Analysis Using Stratix IV Devices**




The design analysis in Figure 1–8 shows the data comparison between `rdata_reg[71:0]` and `data_out[71:0]` when the `rdata_valid_reg` and `data_out[79:72]` signals are high.

**Figure 1–8. SignalTap II Example QDR II and QDR II+ SRAM Data Comparison**




The `data_out[79:72]` signal that acts as byte-enable signal, selects certain ×9 groups to compare the data. For this example, the `data_out[79:72]` signal is `8'b11111111`, which indicates that all the ×9 groups in the `data_out[71:0]` signal are selected for data comparison.

# 2. Using RLDRAM II Controller with UniPHY in Stratix III and Stratix IV Devices

This tutorial describes how to use the design flow to implement a 36-bit wide, 400-MHz, 800-Mbps RLDRAM II controller with UniPHY interface. This tutorial also provides some recommended settings, including termination schemes and drive strength settings, to simplify the design.

The design examples target Stratix III memory demonstration board and Stratix IV FPGA development board, which include a 36-bit wide 576-Mb Micron MT49H16M36-18 533-MHz component.

The Stratix III demonstration board is for internal use only.

To download the design examples, **emi_rldramii_siii.zip** and **emi_rldramii_siv.zip**, go to the External Memory Interface Design Examples page. You can also use these design examples if you are targeting a HardCopy device for your design. For UniPHY supported devices, refer to mmmm section in volume 3 of the *External Memory Interface Handbook*.

For more information about the design flow, refer to the *Recommended Design Flow* section in volume 1 of the *External Memory Interface Handbook*. For more information about the RLDRAM II SRAM controller with UniPHY, refer to the *RLDRAM II Controller with UniPHY* section in volume 3 of the *External Memory Interface Handbook*.

## System Requirements

This tutorial requires the following hardware and software for the RLDRAM II controller with UniPHY using a Stratix III device:

- Quartus II software version 10.0
- RLDRAM II Controller with UniPHY version 10.0
- ModelSim-Altera version 6.5b or later
- Stratix III FPGA memory demonstration board with EP3SL150F1152C2 device

This tutorial requires the following hardware and software for the RLDRAM II controller with UniPHY using a Stratix IV device:

- Quartus II software version 10.0
- RLDRAM II Controller with UniPHY MegaCore version 10.0
- ModelSim-Altera version 6.5b or later
- Stratix IV FPGA development board with EP4SE530H35C2 device

If you use a Quartus II software version later than 10.0, you must regenerate the MegaCore function in the design example.

# Create a Quartus II Project

Create a project in the Quartus II software that targets the respective device; EP3SL150F1152C2 device for the Stratix III device family, or EP4SE530H35C2 device for the Stratix IV device family.

For detailed step-by-step instructions to create a Quartus II project, refer to the Quartus II software tutorial by clicking the Help menu in the Quartus II window and selecting **PDF Tutorials**.

# Instantiate and Parameterize a Controller

After creating a Quartus II project, instantiate the RLDRAM II controller with UniPHY and its parameters.

## Instantiate a Controller

To instantiate the controller with UniPHY, perform the following steps:

1. Start the MegaWizard Plug-In Manager.
2. In the MegaWizard Plug-In Manager, expand **External Memory** in the **Interfaces** folder and select **RLDRAM II Controller with UniPHY v10.0**.
3. Type `rldram` for the name of the RLDRAM II controller with UniPHY.

## Parameterize a Controller

To parameterize the RLDRAM II controller with UniPHY to interface with a 400-MHz, 36-bit wide RLDRAM II interface, perform the following steps:

1. In the **Presets** list, select **MT49H16M36-18** and click **Apply**. The memory parameters are set automatically based on the memory preset settings you choose. You may change the parameters to suit your design requirements.

   CAUTION Selecting a new memory preset overwrites your existing settings. You need to enter the settings again.

2. In the **General Settings** tab, under **Clocks**, for **Memory clock frequency**, type **400** MHz as the system frequency.
3. For **PLL reference clock frequency**, type **100** MHz for Stratix III design, or **50** MHz for Stratix IV design to match the on-board oscillator.
4. For **Full or half rate on Avalon-MM interface**, select **Half**. This option allows you to choose between the full-rate and half-rate controller, and define the bus data width between the controller and the PHY.

   For more information about selecting half-rate or full-rate controller, refer to the *Device and Pin Planning* section in volume 2 of the *External Memory Interface Handbook*.

5. For **Additional Address/Command clock phase**, select **0**.

6. Under **Advanced Settings**, for **Maximum Avalon-MM burst length**, specify the burst length based on the burst count sent from the core fabric. For both design examples, select **128**.

   If you are using the SOPC Builder system, turn on **Generate power-of-2 bus widths**.

7. For **I/O standard**, select **1.8-V HSTL**.
8. Turn on **Master for PLL/DLL sharing** if you want the UniPHY to instantiate its own PLL. If the **Master for PLL/DLL sharing** option is disabled, the PLL clock shares with other identical UniPHY core.

   If you want to migrate your design to a HardCopy device, turn on **HardCopy Compatibility Mode**, and select the appropriate **Reconfigurable PLL Location**.

9. Under **Example Testbench Simulation Options**, turn on **Skip memory initialization**. Enabling this option allows you to skip the memory initialization sequence during simulation so that you can reduce the simulation time.
10. Under **Controller Settings**, select **2** for **Controller Latency**.

    Select the controller latency based on your targeted frequency. For more information, refer to the Latency chapter in the *RLDRAM II Controller with UniPHY IP User Guide* section in the *External Memory Interface Handbook.*

11. In the **Memory Parameters** tab, for **Memory Mode Register Configuration**, select **tRC=6, tRL=6, tWL=7, f=400–175MHz**.
12. In the **Board Settings** tab, set the **Setup and Hold Derating** parameters to the specified values in Table 2–1.

**Table 2–1. Setup and Hold Derating Parameters for Stratix III and Strativ IV Devices**

| Setup and Hold Derating Parameters | Stratix III (ps) | Stratix IV (ps) |
|---|---|---|
| tAS Vref to CK/CK# Crossing | 25 | 0 |
| tAS VIH MIN to CK/CK# Crossing | -100 | -100 |
| tAH CK/CK# Crossing to Vref | 13 | 0 |
| tAH CK/CK# Crossing to VIH MIN | -50 | -50 |
| tDS Vref to CK/CK# Crossing | 11 | 5 |
| tDS VIH MIN to CK/CK# Crossing | -100 | -100 |
| tDH CK/CK# Crossing to Vref | 6 | 3 |
| tDH CK/CK# Crossing to VIH MIN | -50 | -50 |

The Intersymbol Interference (ISI) parameters are not applicable for single chip-select configurations. Set these parameters to **0 ps**.

For more accurate timing analysis, assign the board trace delay model for every single pin. The board skew values are used to calculate the overall system timing margin. Change these values based on the performance of your board if you are not using the board trace delay model, or if you cannot accurately assign the board trace model for each pin. Refer to "Enter Board Trace Delay Models" on page 2–8 for more information about board trace delay model.

13. Set the **Board Skews** parameters to the specified values in Table 2–2.

**Table 2–2. Board Skews Parameters for Stratix III and Strativ IV Devices**

| Board Skews Parameters | Stratix III (ps) | Stratix IV (ps) |
|---|---|---|
| Minimum delay difference between CK and DK | -55 | 15 |
| Maximum delay difference between CK and DK | 0 | 26 |
| Maximum delay difference between devices | 0 | 0 |
| Maximum skew within DK group | 45 | 5 |
| Maximum skew between DK groups | 32 | 7 |
| Average delay difference between Address/Command and CK | 87 | 12 |
| Average delay difference between Data and DK | 61 | 124 |
| Average delay difference between Data and QK | -32 | 42 |
| Maximum skew within Address/Command bus | 93 | 13 |

14. Click **Finish** to generate your MegaCore function variation. The MegaWizard Plug-In generates all the files necessary for your RLDRAM II controller with UniPHY, and an example top-level design, which you may use to test or verify the board operation.
15. If you generate the MegaCore function instance in a Quartus II project, you are prompted to add the **.qip** files to the current Quartus II project. When prompted to add the **.qip** files to your project, click **Yes**. The NativeLink requires the **.qip** files to include libraries for simulation.

    The **.qip** file is generated by the MegaWizard interface, and contains information about the generated IP core. In most cases, the **.qip** file contains all of the necessary assignments and information required to process the MegaCore function or system in the Quartus II compiler. The MegaWizard interface generates a single **.qip** file for each MegaCore function.

16. Click **Exit** to close the MegaWizard Plug-In Manager after you have reviewed the generation report.

For detailed step-by-step instructions for parameterizing the RLDRAM II controller with UniPHY, refer to *Volume 3: Implementing Altera Memory Interface IP* of the *External Memory Interface Handbook*.

# Add Constraints

After instantiating the RLDRAM II controller with UniPHY, the Quartus II software generates the constraints files for the design example. Apply these constraints to the design before compilation.

## Add Example Project

Add the example project files to your project, by performing the following steps:

1. On the Project menu, click **Add/Remove Files in Project**.
2. Browse to the <*variation_name*>**\example_project** directory.
3. Select all the files with the <*variation_name*> prefix, except for the <*variation_name*>**mem_model.sv** file, and click **Open**.
4. Click **OK**.

## Set Top-Level Entity

The top-level entity of the project must be set to the correct entity. To set the top-level entity, perform the following steps:

1. On the File menu, click **Open**.
2. Browse to the <*variation_name*>**_example_top.v** or **.vhd** file and click **Open**.
3. On the Project menu, click **Set as Top-Level Entity**.

Refer to the <*variation_name*>**_example_top.v** or **.vhd** file to instantiate the memory controller in your own design.

## Add Pin and DQ Group Assignments

The pin assignment script, <*variation_name*>**_pin_assignments.tcl**, sets up the I/O standards and the input/output termination for the RLDRAM II with UniPHY interface. This script also helps to relate the `DQ` and `QK` pin groups together for the Fitter to place them correctly in the Quartus II software.

This script does not create a clock for the design. You need to create a clock for the design and provide pin assignments for the signals of both the example driver and testbench that the MegaCore variation generates.

Run the <*variation_name*>**_pin_assignments.tcl** script to add the input/output termination, I/O standards, and DQ group assignments to the design example. To run the pin assignment script, perform the following steps:

1. On the Processing menu, point to **Start** and click **Start Analysis & Synthesis**.
2. In the Tools menu, click Tcl Scripts.
3. Locate the <*variation_name*>**_pin_assignments.tcl** file.
4. Click **Run**.

The PLL input clock I/O does not need to have the same I/O standard as the memory interface I/Os. However, you may see a no-fit error as the bank in which the PLL input clock I/O gets placed becomes unusable for placement of the memory interface I/Os because of the incompatible $V_{CCIO}$ levels. Altera recommends that you assign the same I/O standard to the PLL input clock I/O.

## Enter Pin Location Assignments

To enter the pin location assignments, assign all of your pins, so the Quartus II software fits your design correctly and performs correct timing analysis. To assign the pin locations for the Stratix III demonstration board, run the Altera-provided **S3_Memory_RLDRAM_PinLocations.tcl** file, and for the Stratix IV GX FPGA development board, run the Altera-provided **S4_Host_RLDRAM_PinLocations.tcl** file.

If you are at the design exploration phase of your design cycle and do not have any PCB defined pin locations, you must still manually define an initial set of pin constraints, which can become more specific during your development process.

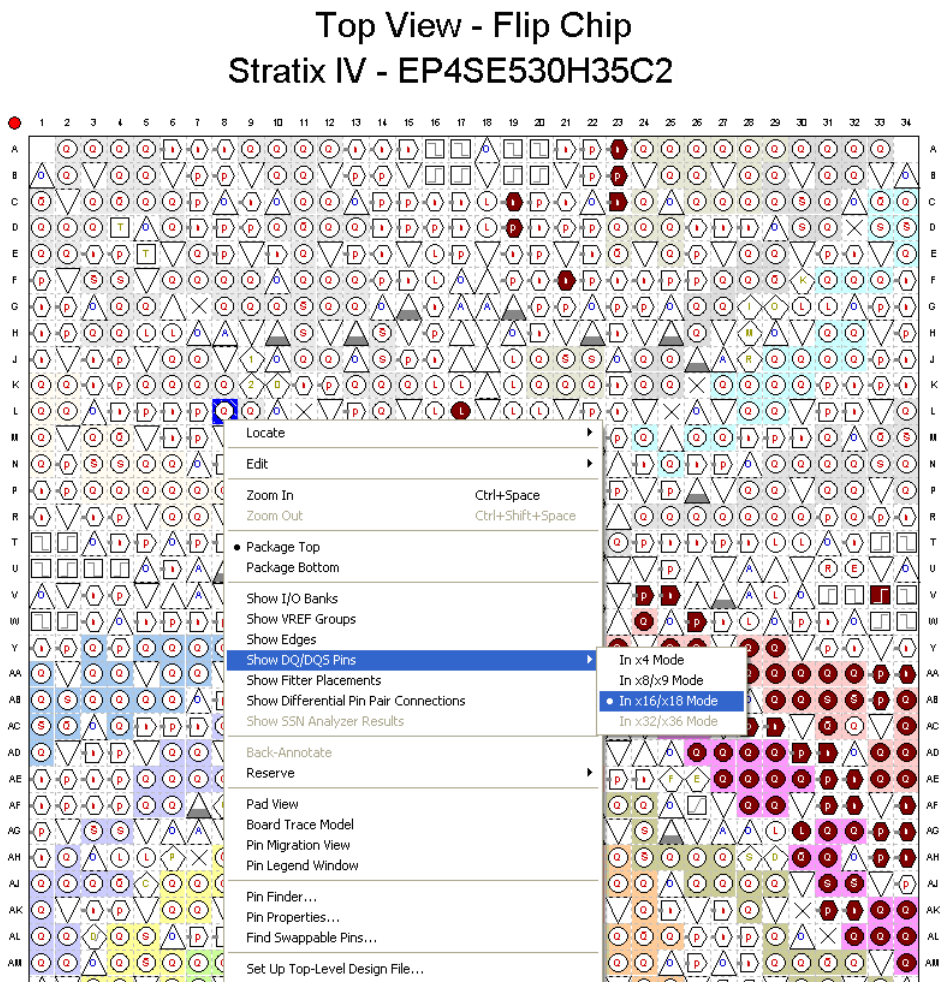
Alternatively, to manually assign the pin locations, perform the following steps:

1. Open the Pin Planner. On the Assignments menu, click **Pin Planner**.
2. To assign DQ, DK, DKn, QK, and QKn pins, follow these steps:
   a. Assign each QK and QKn pin in your design to the required DQS and DQSn pin in the Pin Planner. The Quartus II Fitter then automatically places the respective DQ signals onto suitable DQ pins within each group. To view the DQS groups in Pin Planner, right click, select **Show DQ/DQS Pins**, and click **In ×16/×18 Mode**. The Pin Planner shows each DQS group in a different color and with a different legend: S = DQS pin, Sbar = DQSn pin and Q = DQ pin, as shown in Figure 2–1.

In the ×36 CIO RLDRAM II interface, two ×16/×18 DQS groups are used. Select the DQ mode to match the DQ group width (number of DQ pins/number of DQS pins) of your memory device. DQ mode is not related to the memory interface width. DQ group order and DQ pin order within each group is not important. However, you must place DQ pins in the same group as their respective strobe pin.

b. Assign the DK and DKn pins to unused DQ pins in the DQS group, where the QK pin is assigned, or in adjacent DQS group, or in the same bank as the address and command pins.

**Figure 2–1. Quartus II Pin Planner, Show DQ/DQS Pins, In ×16/×18 Mode**




3. Assign the DM pin to an unused DQ pin in the DQS group where the QK pin is assigned.
4. Assign the address and control command pins to any spare I/O pins, ideally within the same bank or side of the device as the mem_clk, DQ, QK, and DM pins.

5. Ensure you assign the mem_clk pins on differential I/O pairs to the CK/CK# pin pair. To identify differential I/O pairs, right-click in the Pin Planner and select **Show Differential Pin Pair Connections**. The Pin Planner shows a red line between each pin pair. You must assign the mem_clk pins on the same side of the device as the address and command pins.
6. Assign the oct_rup and oct_rdn pins to the pull-up and pull-down resistance pins on the board.
7. Assign the pll_ref_clk pin to a dedicated PLL clock input pin with a direct connection to the UniPHY PLL and DLL pair—usually on the same side of the device as your memory interface. This recommendation reduces PLL jitter, saves a global clock resource, and eases timing and Fitter effort.
8. Assign the global_reset_n pin, like any high fan-out signal, to a dedicated clock pin.

For more information about how to assign the pin for the RLDRAM II CIO, refer to the *Device and Pin Planning* section in volume 2 of the *External Memory Interface Handbook*. For more information about how to use the Quartus II Pin Planner, refer to the *I/O Management* chapter in volume 2 of the *Quartus II Handbook*.

## Assign Virtual Pins

For debugging purposes, the reset_request_n, afi_cal_success, and afi_cal_fail signals are routed from the internal PHY to the top level of the design so that the signals can be monitored externally. When the PLL is out of lock, the reset_request_n signal goes high and when the PLL is locked, the reset_request_n signal goes low. The afi_cal_success and afi_cal_fail signals are used to check the calibration status of the sequencer. If the sequencer calibrates successfully, the afi_cal_success signal goes high and the afi_cal_fail signal goes low.

The example top-level design, which is auto-generated by the RLDRAM II controller with UniPHY, includes an example driver to stimulate the interface. This example driver is not part of the RLDRAM II controller with UniPHY IP, but allows easy testing of the IP.

The example driver outputs several test signals to indicate its operation and the status of the stimulated memory interface. These test signals are pass, fail, and test_complete. The test signals are not part of the memory interface, but are to facilitate testing. You must either connect these signals to a debug bus or assign the signals to virtual pins using the Quartus II Assignment Editor. When using the example driver for testing, do not remove the test signals from the top-level signal list. If you remove the test signals from the top-level module, the Quartus II software optimizes the driver away, and the example driver fails.

Use the Assignment Editor to assign these virtual pins.

## Enter Board Trace Delay Models

For accurate I/O timing analysis, the Quartus II software must be aware of the board trace and loading information. You should derive and refine board trace and loading information during your PCB development process of prelayout (line) and post-layout (board) simulation.

To enter board trace information, perform the following steps:

1. In the Pin Planner, select the pin or group of pins for which you want to enter board trace parameters.
2. Right-click and select **Board Trace Model**.
3. Enter the board values based on the trace length, capacitance length, inductance length, pull-up, pull-down and so on based on your board.

Figure 2–2 shows a typical board trace model.

**Figure 2–2. Stratix IV Development Board Address Signal Board Trace Model**

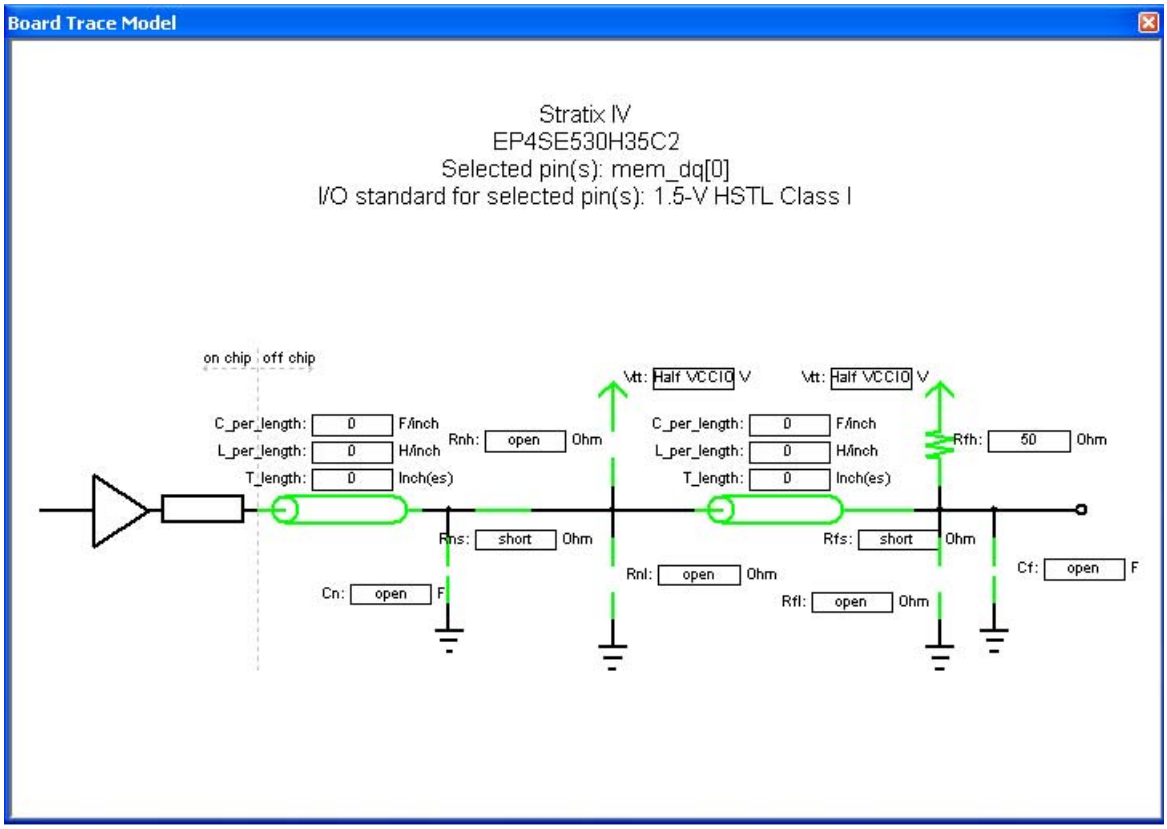



Table 2–3 shows the board trace model parameters for the Stratix III demonstration board and Stratix IV GX FPGA development board.

**Table 2–3. Stratix III and Stratix IV Board Trace Model Summary (Part 1 of 2)**

| Net | Near (FPGA End of Line) | | | | | | Far (Memory End of Line) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Length (in) | C_per_ length (pF/in) | L_per_ length (H/In) | Cn (pF) | Rns (W) | Rnh (W) | Length (in) | C_per_ length (pF/in) | L_per_ length (nF/in) | Cf (pF) | Rfh/Rfp (W) |
| **Stratix III** | | | | | | | | | | | |
| Addr *(1)* | 3.123 | 3.738 | 8.257 | — | — | — | — | — | — | 4.0 | 56 |
| CLK | 1.907 | 3.33 | 9.266 | — | — | — | — | — | — | 2.5 | — |
| DQ/DM | 1.801 | 3.738 | 8.257 | — | — | — | — | — | — | 4.25 | — |

**Table 2–3. Stratix III and Stratix IV Board Trace Model Summary (Part 2 of 2)**

| Net | Near (FPGA End of Line) | | | | | | Far (Memory End of Line) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Length (in) | C_per_ length (pF/in) | L_per_ length (H/In) | Cn (pF) | Rns (W) | Rnh (W) | Length (in) | C_per_ length (pF/in) | L_per_ length (nF/in) | Cf (pF) | Rfh/Rfp (W) |
| DK0 | 1.723 | 3.33 | 9.266 | — | — | — | — | — | — | 2.5 | 56 |
| DK1 | 1.795 | 3.33 | 9.266 | — | — | — | — | — | — | 2.5 | 56 |
| **Stratix IV** | | | | | | | | | | | |
| Addr *(1)* | 2.231 | 3.2 | 8.1 | — | — | — | — | 4.5 | — | 2.0 | 56 |
| CLK | 2.107 | 2.9 | 9.1 | — | — | — | — | 4.5 | — | 2.5 | 56 |
| DQ/DM | 1.868 | 2.0 | 8.1 | — | — | — | — | 4.5 | — | 4.3 | 125 |
| DK0 | 1.895 | 2.9 | 9.1 | — | — | — | — | 2.3 | — | 2.5 | 56 |
| DK1 | 1.897 | 2.9 | 9.1 | — | — | — | — | 2.3 | — | 2.5 | 56 |

**Note to Table 2–3:**

(1) Addr = Addr, ba, cs_n, ref_n, and we_n.

# Perform RTL or Functional Simulation (Optional)

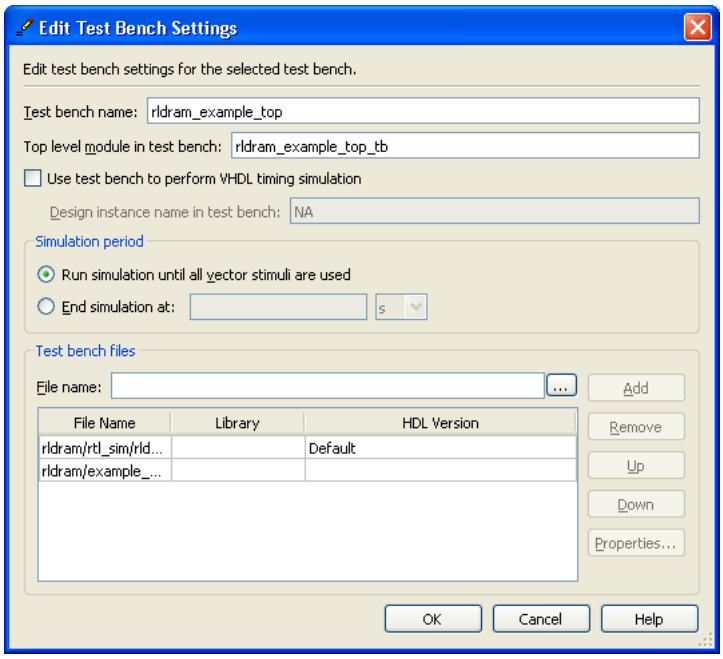
This section describes RTL simulation. The RLDRAM II controller with UniPHY automatically generates the RTL simulation memory model files, *<variation name>*_**mem_model.sv**, and testbench top-level files, *<variation name>*_**example_top_tb.v**, located in the **rtl_sim** folders. You can use these files to verify your designs.

To run the RTL simulation with NativeLink, perform the following steps:

1. Set the absolute path to your third-party simulator executable, by performing the following steps:
   a. On the Tools menu, click **Options**.
   b. In the **Category** list, select EDA Tools Options and set the default path for **ModelSim-Altera** to **C:\<version>\modelsim_ae\win32aloem**.
   c. Click **OK**.
2. On the Assignments menu, click **EDA Tool Settings**.
3. In the **Category** list, expand **EDA Tool Settings** and click **Simulation**.
4. Under **Tool name**, select **ModelSim**.
5. Under **NativeLink settings**, select **Compile test bench** and click **Test Benches**.
6. Click **New**.

7. In the **Edit Test Bench Settings** dialog box, perform the following steps:
   a. Type the testbench name, testbench top-level module, design instance name, and simulation period.
   b. In the **Test bench files** field, include the testbench file, <*variation name*>**_example_top_tb.v**, and the memory model file, <*variation name*>**_mem_model.sv**. (Figure 2–3).
   c. Click **OK**.

**Figure 2–3. Testbench Settings**




8. To elaborate your design, on the Processing menu, point to **Start** and click **Start Analysis & Elaboration**.
9. On the Tools menu, point to the **Run EDA Simulation Tool** and click **EDA RTL Simulation**. This step creates the **\simulation** directory in your project directory and a script that compiles all necessary files and runs the simulation.

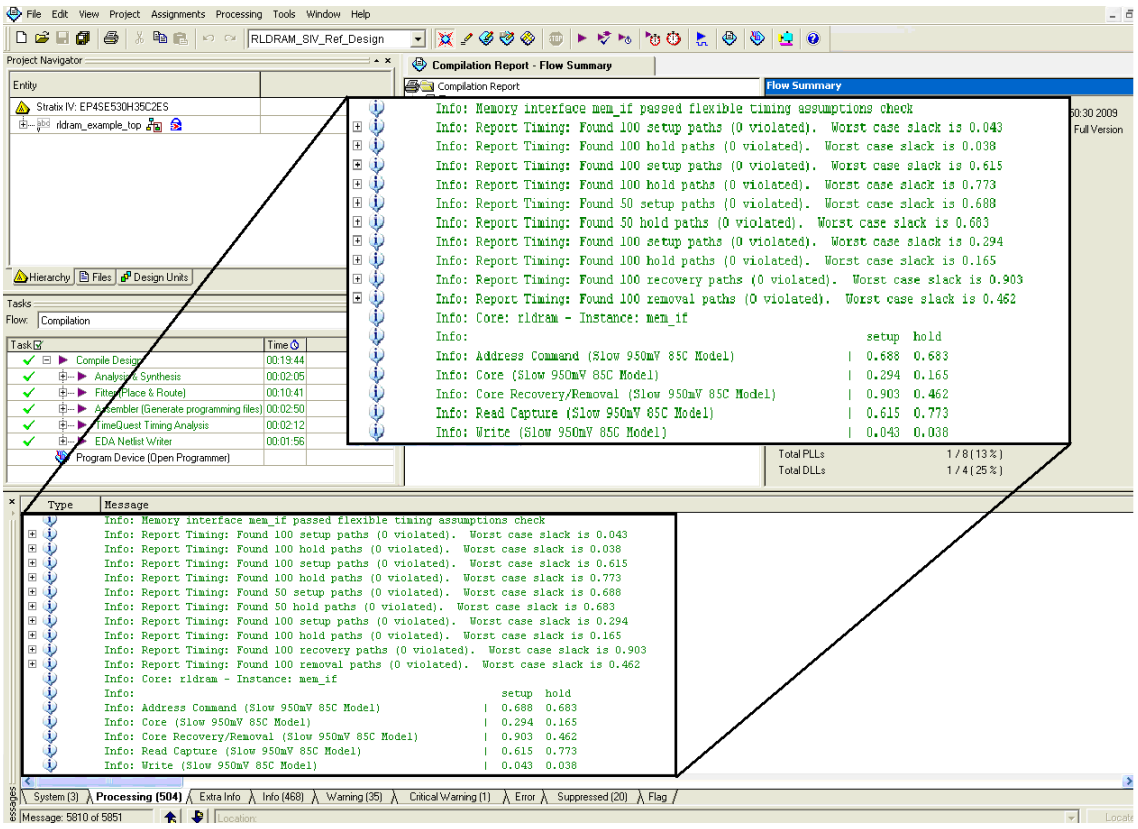
# Compile Design and Verify Timing

To compile the design, on the Processing menu, click **Start Compilation**.

After successfully compiling the design, the Quartus II software automatically runs the verify timing script, <*variation_name*>**_report_timing.tcl**, which produces a timing report for the design together with the compilation report.

Figure 2–4 shows the timing margin report in the message window in the Quartus II software.

**Figure 2–4. Timing Margin Report in the Quartus II Software**




The report timing script performs the following tasks:

1. Creates a timing netlist.
2. Reads the **.sdc** file.
3. Updates the timing netlist.

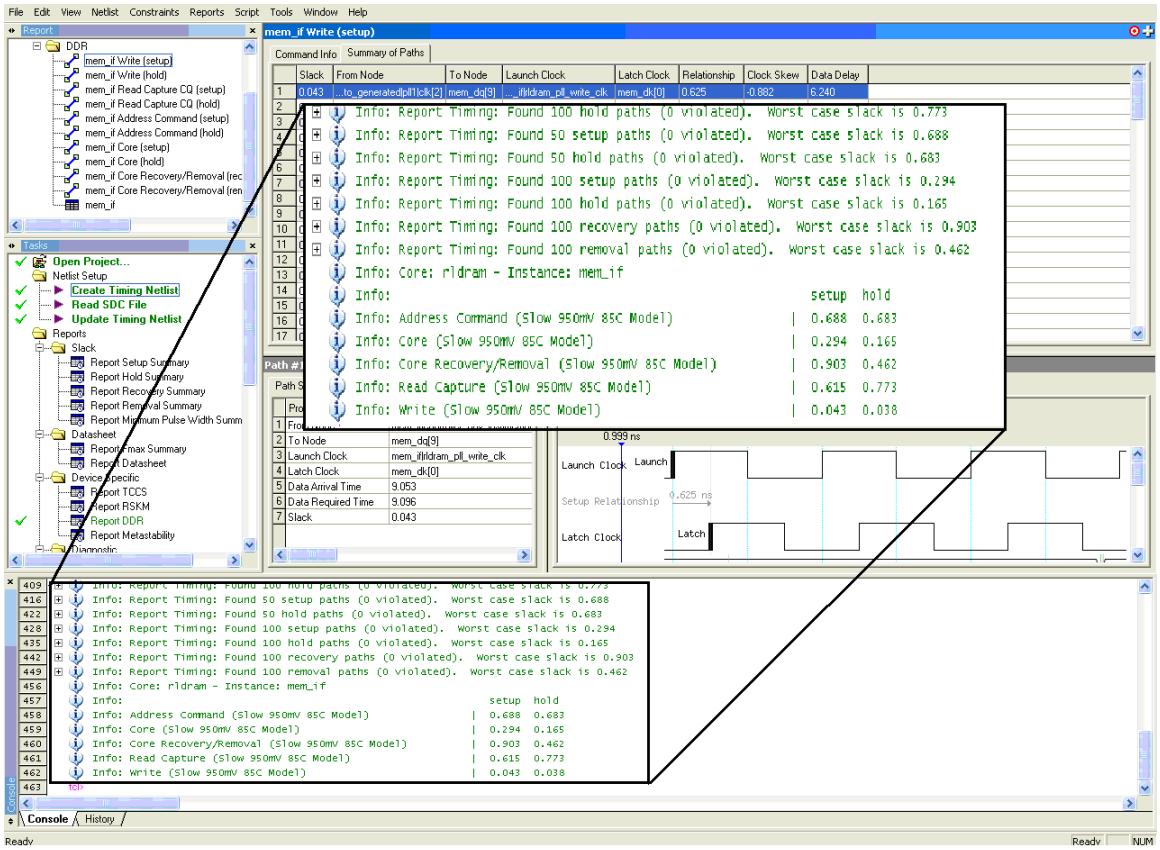
You can also obtain the timing report by running the report timing script in the TimeQuest timing analyzer. To obtain the timing report in the TimeQuest Timing Analyzer window, perform the following steps:

1. In the Quartus II software, on the Tools menu, click **TimeQuest Timing Analyzer**.
2. On the **Tasks** pane, double-click **Update Timing Netlist**, which automatically runs **Create Timing Netlist** and **Read SDC**. After a task is executed, it turns green.
3. After completing the tasks, run the report timing script by going to the Script menu and clicking **Run Tcl Script**.

Alternatively, you can directly double-click on **Report DDR** on the **Tasks** pane to get the timing report.

Figure 2–5 shows the timing margin report in the TimeQuest Timing Analyzer window after running the report timing script. The results are the same as the Quartus II software results.

**Figure 2–5. Timing Margin Report in TimeQuest Timing Analyzer**




For more information about the TimeQuest timing analyzer, refer to *The Quartus II TimeQuest Timing Analyzer* chapter in volume 3 of the *Quartus II Handbook*.

For information timing analysis, refer to the *Timing Analysis* section in volume4 of the *External Memory Interface Handbook*.

# Verify Design on a Board

The SignalTap II Embedded Logic Analyzer shows read and write activity in the system.

For more information about using the SignalTap II Embedded Logic Analyzer, refer to the *Design Debugging Using the SignalTap II Embedded Logic Analyzer* chapter in the *Quartus II Handbook*, *AN 323: Using SignalTap II Embedded Logic Analyzers in SOPC Builder Systems* and *AN 446: Debugging Nios II Systems with the SignalTap II Embedded Logic Analyzer*.

To add the SignalTap II Embedded Logic Analyzer, perform the following steps:

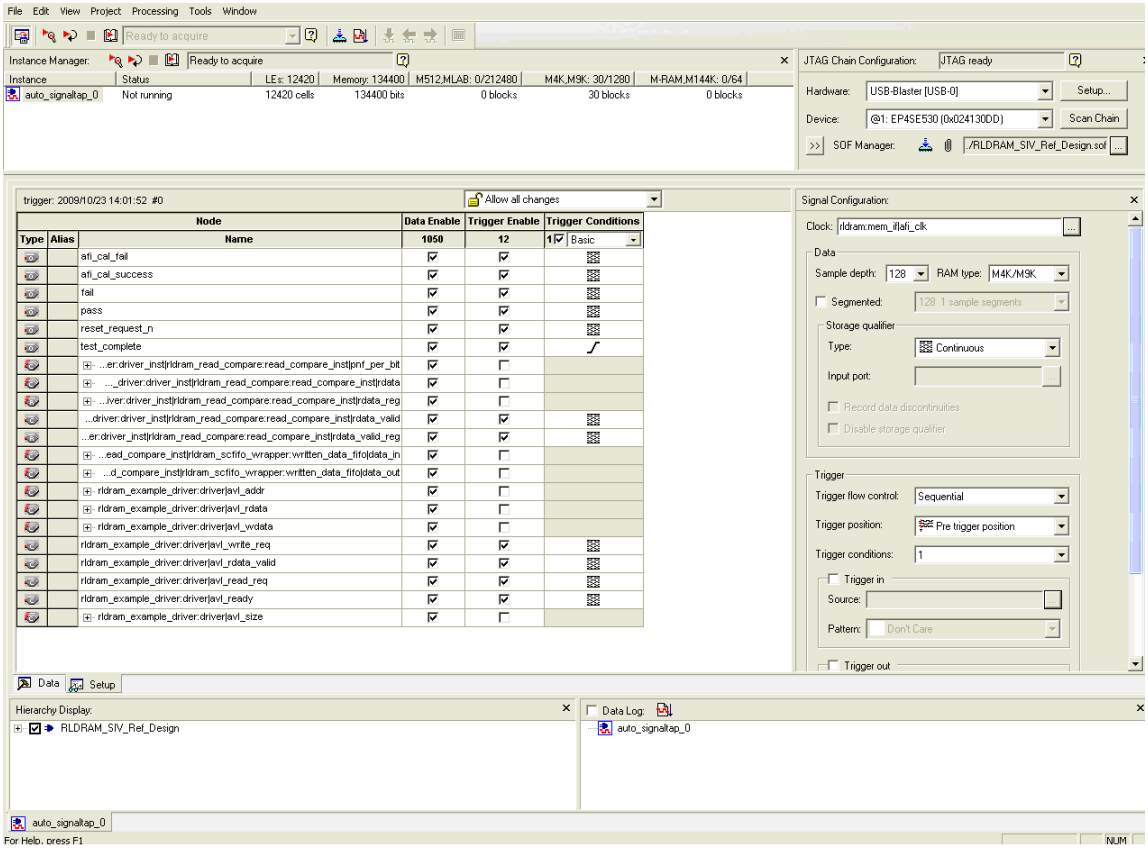
1. On the Tools menu, click **SignalTap II Logic Analyzer**.
2. Under **Signal Configuration** next to the **Clock** box, click **…** (Browse Node Finder).
3. Type `*afi_clk` in the **Named** box, for **Filter** select **SignalTap II: pre-synthesis** and click **List**.
4. Select **<variation_name>:mem_if|afi_clk** in **Nodes Found** and click **>** to add the signal to **Selected Nodes**.
5. Click **OK**.
6. Under **Signal Configuration**, specify the following settings:
   - For **Sample depth**, select **128**
   - For **RAM type**, select **Auto**
   - For **Trigger flow control**, select **Sequential**
   - For **Trigger position**, select **Center trigger position**
   - For **Trigger conditions**, select **1**
7. On the Edit menu, click **Add Nodes**.
8. Search for specific nodes by typing `*avl*` in the **Named** box, for **Filter** select **SignalTap II: pre-synthesis** and click **List**.
9. In **Nodes Found**, select the following nodes and click **>** to add to **Selected Nodes**:
   - **avl_addr**
   - **avl_rdata**
   - **avl_rdata_valid**
   - **avl_read_req**
   - **avl_ready**
   - **avl_wdata**
   - **avl_write_req**
   - **fail**
   - **pass**
   - afi_cal_fail
   - afi_cal_success
   - **test_complete** (trigger)
   - **pnf_per_bit**
   - **rdata_valid**
   - **rdata_valid_reg**
   - **data_out**
   - **data_in**

Do not add any RLDRAM II interface signals to the SignalTap II Embedded Logic Analyzer. The load on these signals increases and adversely affects the timing analysis.

10. Click **OK**.
11. To reduce the SignalTap II logic size, turn off **Trigger Enable** on the following bus signals:
    - **avl_addr**
    - **avl_rdata**
    - **avl_wdata**
    - **data_out**
    - **data_in**
12. Right-click **Trigger Conditions** for the `test_complete` signal and select **Rising Edge**.

    Figure 2–6 shows the completed SignalTap II Embedded Logic Analyzer.

**Figure 2–6. SignalTap II Embedded Logic Analyzer**




13. On the File menu, click **Save**, to save the SignalTap II **.stp** file to your project.

If you see the message **Do you want to enable SignalTap II file "stp1.stp" for the current project**?, click **Yes**.

## Compile the Project

After you add the signals to the SignalTap II Embedded Logic Analyzer, on the Processing menu, click **Start Compilation** to recompile your design.

## Verify Timing

After you have recompiled the design, ensure that the TimeQuest timing analysis passes successfully. In addition to this FPGA timing analysis, check your PCB or system SDRAM timing. To run timing analysis, follow these steps:

1. On the Tools menu, click **Tcl Scripts**.
2. Select <*variation name*>**_report_timing.tcl** and click **Run**.

## Connect the Development Board

Connect the development board to your computer.

## Download the Object File

On the Tools menu, click **SignalTap II Logic Analyzer**. The SignalTap II dialog box appears.

The SOF Manager should contain the **<***your project name***>.sof** file. If not, to add the correct file to the SOF Manager, perform the following steps:

1. Click **...** to open the **Select Program Files** dialog box.
2. Select <*your project name*>**.sof**.
3. Click **Open**.

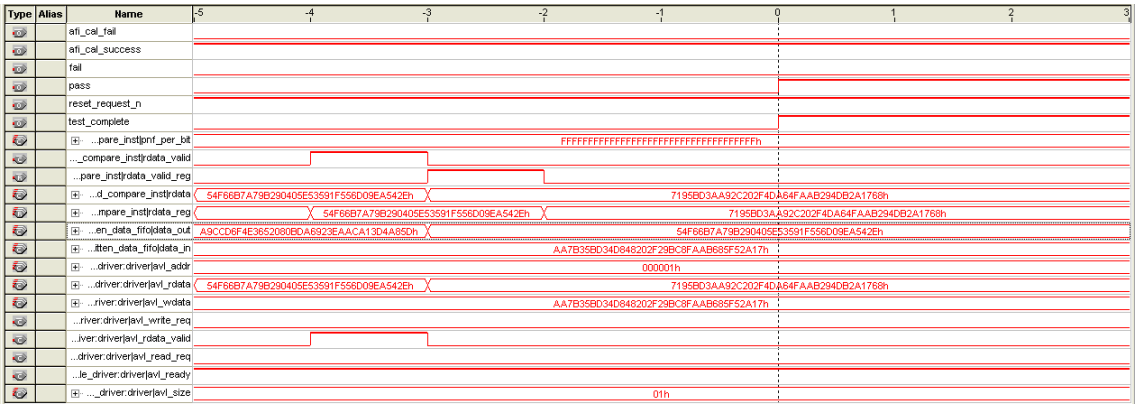
To download the file, click the **Program Device** button.

## Test the Design Example in Hardware

When the design example including SignalTap II successfully downloads to your development board, click **Run Analysis** to run the analysis once, or click **Autorun Analysis** to run the analysis continuously.

Figure 2–7 on page 2–17 shows the design analysis. The example driver performs self verification by comparing the written out data, `data_out` with the read back data, `rdata_reg` when the `rdata_valid_reg` signal goes high.

**Figure 2–7. SignalTap II Example RLDRAM II Design Analysis**



# 3. Using DDR2 and DDR3 SDRAM Controller with UniPHY in Stratix III and Stratix IV Devices

This tutorial describes how to use the design flow to implement a a 72-bit wide, 400-MHz DDR2 SDRAM controller and a 64-bit wide, 533-MHz, 1066-Mbps DDR3 SDRAM controller with UniPHY interface. This tutorial also provides some recommended settings to simplify the design.

The design examples target the Stratix III FPGA development board and Stratix IV GX FPGA development board. The Stratix III FPGA development kit includes a 72-bit wide 1-Gb Micron MT9HTF12872AY-800E 400-MHz DDR2 SDRAM DIMM, and the Stratix IV GX FPGA development board includes four 16-bit wide, 1-Gb Micron MT41J64M16LA-15E 667-MHz DDR3 SDRAM components.

To download the design examples, **emi_uniphy_ddr2_siii.zip** and **emi_uniphy_ddr3_siv.zip**, go to the External Memory Interface Design Examples page.

The design flow is also applicable for Stratix V devices.

For more information about the design flow, refer to the *Recommended Design Flow* section in volume 1 of the *External Memory Interface Handbook*. For more information about the DDR2 and DDR3 SDRAM controllers with UniPHY, refer to the *DDR2 and DDR3 SDRAM Controller with UniPHY User Guide* section in volume 3 of the *External Memory Interface Handbook*.

## System Requirements

This tutorial requires the following hardware and software:

- DDR2 SDRAM controller with UniPHY with a Stratix III device
  - Quartus II software version 10.1
  - DDR2 SDRAM Controller with UniPHY version 10.1
  - ModelSim-Altera version 6.6c or later
  - Stratix III FPGA development kit with EP3SL150F1152-C2 device
- DDR3 SDRAM controller with UniPHY with a Stratix IV device
  - Quartus II software version 10.0
  - DDR3 SDRAM Controller with UniPHY version 10.0
  - ModelSim-Altera version 6.5e or later
  - Stratix IV FPGA development board with EP4SGX230KF40C2 device

    If you use a Quartus II software version later than the required version, you must regenerate the MegaCore function in the design example.

# Create a Quartus II Project

Create a project in the Quartus II software that targets the appropriate device: the EP3SL150F1152-C2 device for DDR2 SDRAM controller with UniPHY or the EP4SGX230KF40C2 device for DDR3 SDRAM controller with UniPHY.

For detailed step-by-step instructions to create a Quartus II project, refer to the Quartus II software tutorial by clicking the Help menu in the Quartus II window and selecting **PDF Tutorials**.

# Instantiate and Parameterize a Controller

After creating a Quartus II project, instantiate the preferred controller and its parameters.

## Instantiate a Controller

To instantiate the DDR2 or DDR3 SDRAM controller with UniPHY, perform the following steps:

- DDR2 SDRAM controller with UniPHY
  a. Start the MegaWizard Plug-In Manager.
  b. In the MegaWizard Plug-In Manager, expand **External Memory** in the **Interfaces** folder and select **DDR2 SDRAM Controller with UniPHY v10.1**.
  c. Type `ddr2_uniphy` for the name of the DDR2 SDRAM controller with UniPHY.
- DDR3 SDRAM controller with UniPHY
  a. Start the MegaWizard Plug-In Manager.
  b. In the MegaWizard Plug-In Manager, expand **External Memory** in the **Interfaces** folder and select **DDR3 SDRAM Controller with UniPHY v10.0**.
  c. Type `ddr3_uniphy` for the name of the DDR3 SDRAM controller with UniPHY

## Parameterize DDR2 SDRAM Controller with UniPHY using a Stratix III Device

To parameterize the DDR2 SDRAM controller with UniPHY to interface with a 400-MHz, 72-bit wide DDR2 SDRAM interface, perform the following steps:

1. In the **Presets** list, select **MICRON MT9HTF12872AY-800** and click **Apply**. The memory parameters are set automatically based on the memory preset settings you choose. You may change the parameters to suit your design requirements.

   CAUTION Selecting a new memory preset overwrites your existing settings. You need to enter the settings again.

2. In the **PHY Settings** tab, under **FPGA**, for **Speed Grade**, select **2**.
3. Under **Clocks**, for **Memory clock frequency**, type **400** MHz as the system frequency.
4. For **PLL reference clock frequency**, type **50** MHz to match the on-board oscillator.

5. For **Full- or half-rate Avalon-MM interface**, select **Half**.
6. Under **Advanced PHY Settings**, turn on **Advanced clock phase control** and key in the values for **Additional address and command clock phase** and **Additional CK/CK# phase** if you want to increase or decrease the amount of phase shift on the clocks. However, Altera recommends that you to retain the default values.
7. Turn on **HardCopy compatibility** if you want to migrate your design to HardCopy.
8. Under **Example Testbench Simulation Options**, for **Auto-calibration mode**, select **Skip calibration**. Selecting this option allows you to skip memory calibration during simulation so that you can reduce the simulation time.
9. In the **Memory Parameters** tab, for **Memory format**, select **Unbuffered DIMM**.
10. For **Memory device speed grade**, select **400** MHz.
11. For **Total interface width**, type `72`.
12. For **Number of slots** and **Number of chip select per slot**, select **1**.
13. For **Number of clocks per chip select**, select 3.
14. For **Row address width** and **Column address width**, type `10`.
15. For **Bank address width**, type `3`.
16. Under **Memory Initialization Options**, for **Memory CAS latency setting**, select **6**.
17. For **Output drive strength setting**, select **Full**.
18. For **Memory on-die termination (ODT) setting**, select **75**.
19. In the **Board Settings** tab under **Board Skews**, set the **Board Skews** parameters to the specified values in Table 3–1.

**Table 3–1. Board Skew Parameters for Stratix III Devices**

| Slew Rate Parameter | Skew (ns) |
|---|---|
| Minimum delay difference between CK and DQS | 0.098 |
| Maximum delay difference between CK and DQS | 0.151 |
| Maximum skew within DQS group | 0.000 |
| Maximum skew between DQS groups | 0.053 |
| Average delay difference between DQ and DQS | 0.000 |
| Maximum skew within address and command bus | 0.155 |
| Average delay difference between address and command and CK | 0.490 |

For designs using Stratix III devices, you cannot specify the setup and hold derating settings . By default, the slew rates of the single-ended signals are 1 V/ns and the slew rates of the differential signals are 2 V/ns.
For designs using other supported devices, always simulate or measure the design and topology to ensure accurate timing information and analysis.

20. In the **Controller Settings** tab, under **Avalon Interface**, for **Maximum Avalon-MM burst length**, specify the burst length based on the burst count that the core fabric sends. For example, if your system generates up to 64 beats, then select **64**.

If you are using the SOPC Builder system, turn on **Generate power-of-2 bus widths**.

21. Click **Finish** to generate your MegaCore function variation. The MegaWizard Plug-In Manager generates all the files necessary for your DDR2 SDRAM controller with UniPHY, and an example top-level design, which you may use to test or verify the board operation.
22. Click **Exit** to close the MegaWizard Plug-In Manager after you have reviewed the generation report.

## Parameterize DDR3 SDRAM Controller with UniPHY using a Stratix IV Device

To parameterize the DDR3 SDRAM controller with UniPHY to interface with a 533-MHz, 64-bit wide DDR3 SDRAM interface, perform the following steps:

1. In the **Presets** list, select **MT41J64M16LA-15E** and click **Apply**. The memory parameters are set automatically based on the memory preset settings you choose. You may change the parameters to suit your design requirements.

Selecting a new memory preset overwrites your existing settings. You need to enter the settings again.

2. In the **PHY Settings** tab, under **FPGA**, for **Speed Grade**, select **2**.
3. Under **Clocks**, for **Memory clock frequency**, type **533** MHz as the system frequency.
4. For **PLL reference clock frequency**, type **100** MHz to match the on-board oscillator.
5. Under **Advanced Settings**, turn on **Advanced Clock Phase Control** and key in the values for **Additional Address/Command clock phase** and **Additional CK/CK# phase** if you want to increase or decrease the amount of phase shift on the clocks. However, Altera recommends that you to retain the default values.
6. Turn on **Master for PLL/DLL sharing** if you want the UniPHY to instantiate its own PLL. If the **Master for PLL/DLL sharing** option is disabled, the PLL clock shares with other identical UniPHY core.
7. Turn off **HardCopy Compatibility Mode**. This design cannot be migrated to HardCopy.
8. Under **Example Testbench Simulation Options**, for **Auto-calibration mode**, select **Quick initialization and skip calibration**. Selecting this option allows you to skip memory calibration during simulation so that you can reduce the simulation time.
9. In the **Memory Parameters** tab, for **Total Interface Width**, type `64`.
10. Turn on **Fly-by topology**.
11. Under **Memory Initialization Options**, for **Memory CAS latency setting**, select **8**.
12. For **Output drive strength setting**, select **RZQ/7**.
13. For **ODT Rtt nominal value**, select **RZQ/4**.

14. For **Memory Write CAS latency setting**, select **6**.
15. For **Dynamic ODT (Rtt_WR) value**, select **RZQ/4**.
16. In the **Board Settings** tab under **Setup and Hold Derating**, for **Derating Method**, select **Specify slew rates to calculate setup and hold times**.
17. Set the slew rate parameters to the specified values in Table 3–2.

**Table 3–2. Slew Rate Parameters for Stratix IV Devices**

| Slew Rate Parameter | Slew Rate (V/ns) |
|---|---|
| CK/CK# slew rate (Differential) | 4 |
| Address/Command slew rate | 1.5 |
| DQS/DQS# slew rate (Differential) | 3 |
| DQ slew rate | 1.5 |

If your design is targeting a Stratix III device, you have to manually derate $t_{DS}$, $t_{DH}$, $t_{IS}$, and $t_{IH}$ parameters in **Memory Timing** tab. For more information about how to derate memory setup and hold timing , refer to the *DDR3 SDRAM Controller with UniPHY User Guide* in volume 3 of the *External Memory Interface Handbook*.

The Intersymbol Interference (ISI) parameters are not applicable for single chip-select configurations or designs that target Stratix III devices. Set these parameters to **0 ps**.

18. Set the **Board Skews** parameters to the specified values in Table 3–3.

**Table 3–3. Board Skews Parameters for Stratix IV Devices**

| Board Skews Parameter | Skew (ns) |
|---|---|
| Maximum skew within DQS group | 0.015 |
| Maximum skew between DQS groups | 0.128 |
| Maximum skew within Address/Command bus | 0.088 |
| Maximum delay difference between Address/Command and CK | -0.05 |

19. In the **Controller Settings** tab, under **Avalon Interface**, for **Maximum Avalon-MM burst length**, specify the burst length based on the burst count sent from the core fabric. For example, if your system generates up to 64 beats, then select **64**.

If you are using the SOPC Builder system, turn on **Generate power-of-2 bus widths**.

20. Click **Finish** to generate your MegaCore function variation. The MegaWizard Plug-In Manager generates all the files necessary for your DDR3 SDRAM controller with UniPHY, and an example top-level design, which you may use to test or verify the board operation.
21. Click **Exit** to close the MegaWizard Plug-In Manager after you have reviewed the generation report.

# Add Constraints

After instantiating the DDR2 or DDR3 SDRAM controller with UniPHY, the Quartus II software generates the constraints files for the design example. Apply these constraints to the design before compilation.

## Add Example Project

Add the example project files to your project, by performing the following steps:

1. On the Project menu, click **Add/Remove Files in Project**.
2. Browse to the <*variation_name*>**_example_design_fileset\example_project** (Stratix III) or <*variation_name*>**\example_project** (Stratix IV) directory.
3. Select all the files with the <*variation_name*> prefix, except for the <*variation_name*>**mem_model.sv** file, and click **Open**.
4. Browse to the <*variation_name*>**_example_design_fileset\example_project** (Stratix III) or <*variation_name*>**\example_project** directory (Stratix IV) and add the <*variation_name*>**.qip** file.
5. Click **OK**.

## Set Top-Level Entity

The top-level entity of the project must be set to the correct entity. To set the top-level entity, perform the following steps:

1. On the File menu, click **Open**.
2. Browse to the <*variation_name*>**_example_top.v** or **.vhd** file and click **Open**.
3. On the Project menu, click **Set as Top-Level Entity**.


Refer to the <*variation_name*>**_example_top.v** or **.vhd** file to instantiate the memory controller in your own design.

## Add Pin and DQ Group Assignments

The pin assignment script, <*variation_name*>**_pin_assignments.tcl**, sets up the I/O standards and the input/output termination for the controller. This script does not include assignments for the design's PLL input clock.


You must create a PLL clock for the design, and provide pin assignments for the signals of both the example driver and testbench that the MegaCore variation generates.

Run the <*variation_name*>**_pin_assignments.tcl** script to add the pins, I/O standards, and DQ group assignments to the design example. To run the pin assignment script, perform the following steps:

1. On the Processing menu, point to **Start** and click **Start Analysis & Synthesis**.
2. In the Tools menu, click **Tcl Scripts**.
3. Locate the <*variation_name*>**_pin_assignments.tcl** file.

4. Click **Run**.

The PLL input clock I/O does not need to have the same I/O standard as the memory interface I/Os. However, you may see a no-fit error as the bank in which the PLL input clock I/O gets placed becomes unusable for placement of the memory interface I/Os because of the incompatible $V_{CCIO}$ levels. Altera recommends that you assign the same I/O standard to the PLL input clock I/O.

## Enter Pin Location Assignments

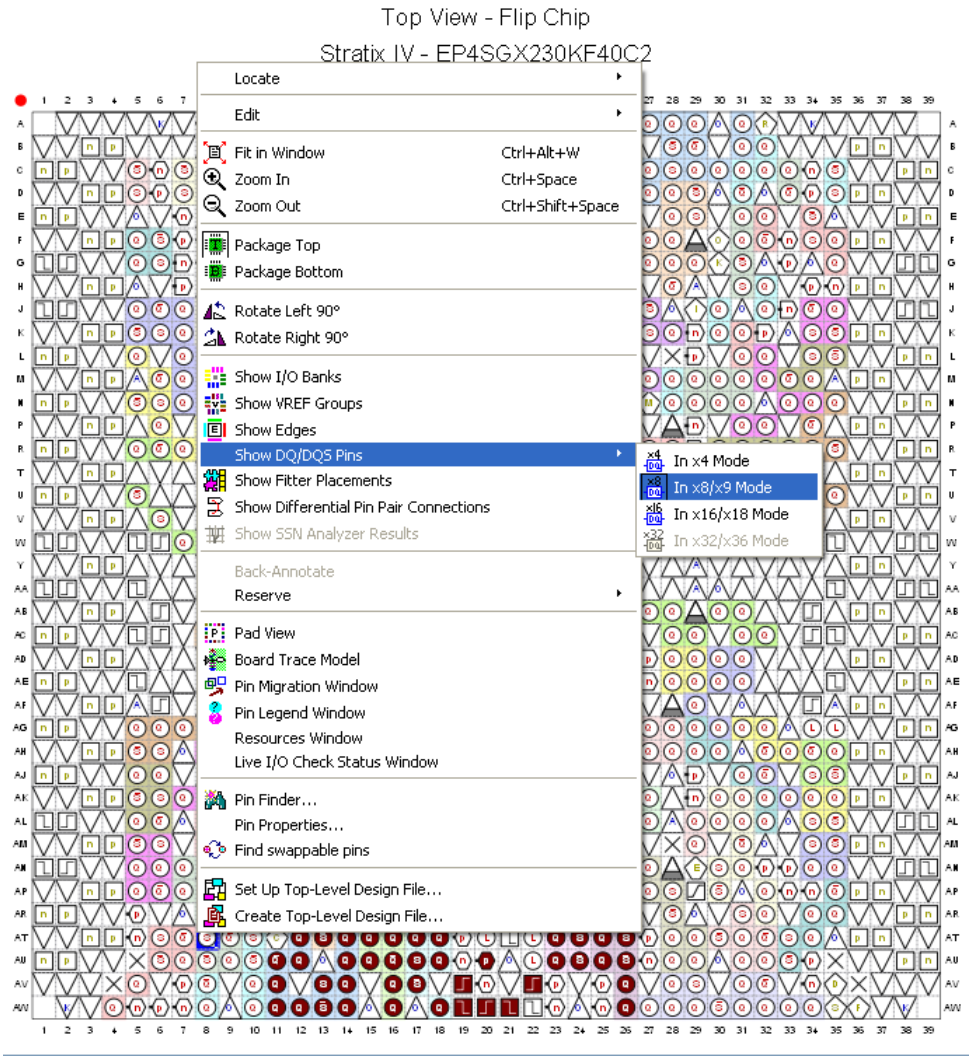
Assign all of your pins so that the Quartus II software fits your design correctly and performs correct timing analysis. To assign the pin locations for the Stratix IV GX FPGA development board, run the Altera-provided **siv_gx_ddr3_pin_location.tcl** file.

Alternatively, to manually assign the pin locations in the Pin Planner, perform the following steps:

1. On the Assignments menu, click **Pin Planner**.
2. To view the DQS groups in Pin Planner, right-click, select **Show DQ/DQS Pins**, and click **In ×8/×9 Mode**. The Pin Planner shows each DQS group in a different color and with a different legend: S = `DQS` pin, Sbar = `DQSn` pin and Q = `DQ` pin, as shown in Figure 3–1.
3. To identify differential I/O pairs, right-click in Pin Planner and select **Show Differential Pin Pair Connections**. The Pin Planner shows a red line between each pin pair.

To assign pin locations for the Stratix III development board, you can also run the Altera-provided **S3_Host_DDR2_PinLocation.tcl** file.

**Figure 3–1. Quartus II Pin Planner, Show DQ/DQS Pins, In ×8/×9 Mode**




For more information about how to assign pin locations, refer to the *Device and Pin Planning* section in volume 2 of the *External Memory Interface Handbook*. For more information about how to use the Quartus II Pin Planner, refer to the *I/O Management* chapter in volume 2 of the *Quartus II Handbook*.

## Assign Virtual Pins

The example top-level design, which is auto-generated by the controller, includes an example driver to simulate the interface. This example driver is not part of the controller, but allows easy testing of the IP.

The example driver outputs several test signals to indicate its operation and the status of the simulated memory interface. These test signals are `afi_cal_success`, `afi_cal_fail`, `pass`, `fail`, and `test_complete`. These signals are not part of the memory interface, but are to facilitate testing. You must either connect these signals to a debug bus or assign the signals to virtual pins using the Quartus II Assignment Editor. When using the example driver for testing, do not remove the test signals from the top-level signal list. If you remove the test signals from the top-level module, the Quartus II software optimizes the driver away, and the example driver fails.

To assign the virtual pins for DDR2 SDRAM controller with UniPHY using the Stratix III development board, run the Altera-provided **s3_Host_ddr2_exdriver_vpin.tcl** file, and for DDR3 SDRAM controller with UniPHY using the Stratix IV GX development board, run the Altera-provided **siv_gx_ddr3_exdriver_vpin.tcl** file, or manually assign the virtual pins using the Assignment Editor.

## Enter Board Trace Delay Models

For accurate I/O timing analysis, the Quartus II software must be aware of the board trace and loading information. You should derive and refine board trace and loading information during your PCB development process of prelayout (line) and post-layout (board) simulation. For external memory interfaces that use memory modules such as DIMMs, this information must also include the trace and loading information in addition to the main and host platform, which you can obtain from your memory vendor.

Altera recommends that you use the board trace model assignment on all DDR2 or DDR3 SDRAM interface signals. To apply the board trace model assignments, run the Altera-provided **s3_ Host_DDR2_BTModels.tcl** file (DDR2 SDRAM) or **siv_gx_ddr3_btmodels.tcl** file (DDR3 SDRAM). To manually assign the board trace model values, use the Pin Planner.

To enter board trace information, perform the following steps:

1. In the Pin Planner, select the pin or group of pins for which you want to enter board trace parameters.
2. Right-click and select **Board Trace Model**.
3. Enter the board values based on the trace length, capacitance per length, inductance per length, pull-up resistance, pull-down resistance, and so on based on your board.

Table 3–4 shows the board trace model parameters for the Stratix III FPGA board.

**Table 3–4. Stratix III FPGA Board Trace Model Summary (Part 1 of 2)**

| Net | Near (FPGA End of Line) | | | | | | Far (Memory End of Line) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Length (Inch) | C_per_ length (pF/In) | L_per_ length (H/In) | Cn (pF) | Rns (W) | Rnh (W) | Length (Inch) | C_per_ length (pF/In) | L_per_ length (nL/In) | Cf (pF) | Rfh/Rfp (W) |
| `Addr` *(1)* | 2.717 | 3.3 | 7.8 | 33 | 5.1 | 56 | 4.968 | 2.8 | 9 | 14 | — |
| `CLK` | 3.069 | 2.8 | 9.2 | 3.5 | — | — | 1.349 | 2.8 | 9 | 4.5 | 67 |

**Table 3–4. Stratix III FPGA Board Trace Model Summary (Part 2 of 2)**

| Net | Near (FPGA End of Line) | | | | | | Far (Memory End of Line) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Length (Inch) | C_per_ length (pF/In) | L_per_ length (H/In) | Cn (pF) | Rns (W) | Rnh (W) | Length (Inch) | C_per_ length (pF/In) | L_per_ length (nL/In) | Cf (pF) | Rfh/Rfp (W) |
| CKE | 2.717 | 2.8 | 7.8 | 33 | — | 56 | 4.968 | 2.8 | 9 | 42 | — |
| CS# | 2.717 | 2.8 | 7.8 | 33 | — | 56 | 3.936 | 2.8 | 9 | 42 | — |
| ODT | 2.717 | 2.8 | 7.8 | 33 | — | 56 | 4.968 | 2.8 | 9 | 39.75 | — |
| DQS0 | 3.017 | 3.4 | 7.8 | — | 22 | — | 0.750 | 2.8 | 9 | 3.5 | 50 |
| DQS1 | 3.005 | 3.4 | 8.1 | — | 22 | — | 0.760 | 2.8 | 9 | 3.5 | 50 |
| DQS2 | 2.851 | 3.4 | 7.8 | — | 22 | — | 0.760 | 2.8 | 9 | 3.5 | 50 |
| DQS3 | 2.653 | 3.4 | 8.1 | — | 22 | — | 0.760 | 2.8 | 9 | 3.5 | 50 |
| DQS4 | 2.686 | 3.4 | 7.8 | — | 22 | — | 0.760 | 2.8 | 9 | 3.5 | 50 |
| DQS5 | 2.701 | 3.4 | 7.8 | — | 22 | — | 0.760 | 2.8 | 9 | 3.5 | 50 |
| DQS6 | 2.75 | 3.4 | 7.8 | — | 22 | — | 0.760 | 2.8 | 9 | 3.5 | 50 |
| DQS7 | 2.923 | 3.4 | 8.1 | — | 22 | — | 0.750 | 2.8 | 9 | 3.5 | 50 |
| DQS8 | 2.536 | 3.4 | 7.8 | — | 22 | — | 0.940 | 2.8 | 9 | 3.5 | 50 |

**Note to Table 3–5:**

(1) Addr = Addr, ba, cs#, we#, ras#, odt, and cas#.

The following compensation capacitors are fitted to the DDR2 SDRAM address and command, and CLK and CLK# signals on the Stratix III development board.

- 33 pF compensation capacitors for address and command signals
- 7 pF (differential) compensation capacitors for CLK and CLK# signals

You fit these capacitors to designs that use asymmetric DIMMs. Simulate your design first to check if it requires compensation capacitors. Observe the memory vendor derating guidelines, because fitting compensation capacitors reduce the edge rate of your signals.

In typical cases your designs may not need compensation capacitors because Stratix III devices have various programmable drive strength and OCT options.

For more information about compensation capacitors, refer to *Micron Technical Note TN_47_01*.

Table 3–5 shows the board trace model parameters for the Stratix IV GX FPGA board.

**Table 3–5. Stratix IV GX FPGA Board Trace Model Summary (Part 1 of 2)**

| Net | Near (FPGA End of Line) | | | | | | Far (Memory End of Line) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Length (Inch) | C_per_ length (pF/In) | L_per_ length (H/In) | Cn (pF) | Rns (W) | Rnh (W) | Length (Inch) | C_per_ length (pF/In) | L_per_ length (nL/In) | Cf (pF) | Rfh/Rfp (W) |
| Addr *(1)* | — | — | — | — | — | — | 4.177 | 4.25 | 7.61 | 5.2 | 56 |
| CLK | — | — | — | — | — | — | 3.865 | 4.11 | 7.84 | 5.6 | 100 |
| CKE | — | — | — | — | — | — | 4.246 | 4.25 | 7.61 | 5.2 | 56 |

**Table 3–5. Stratix IV GX FPGA Board Trace Model Summary (Part 2 of 2)**

| Net | Near (FPGA End of Line) | | | | | | Far (Memory End of Line) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Length (Inch) | C_per_length (pF/In) | L_per_length (H/In) | Cn (pF) | Rns (W) | Rnh (W) | Length (Inch) | C_per_length (pF/In) | L_per_length (nL/In) | Cf (pF) | Rfh/Rfp (W) |
| DQ/DM | — | — | — | — | — | — | 2.317 | 4.25 | 7.61 | 2.5 | 60 |
| DQS | — | — | — | — | — | — | 2.299 | 4.11 | 7.84 | 2.5 | 60 |

**Note to Table 3–5:**

(1) Addr = Addr, ba, cs#, we#, ras#, odt, and cas#.

# Perform RTL or Functional Simulation (Optional)

This section describes RTL (functional) simulations. To perform the functional simulation for your memory interface, use the functional model of the UniPHY generated by the MegaWizard Plug-In Manager. You must use this model together with your own driver or testbench that issues the read and write operations, and a memory model.

You cannot run RTL simulation for designs targeting Stratix V devices.

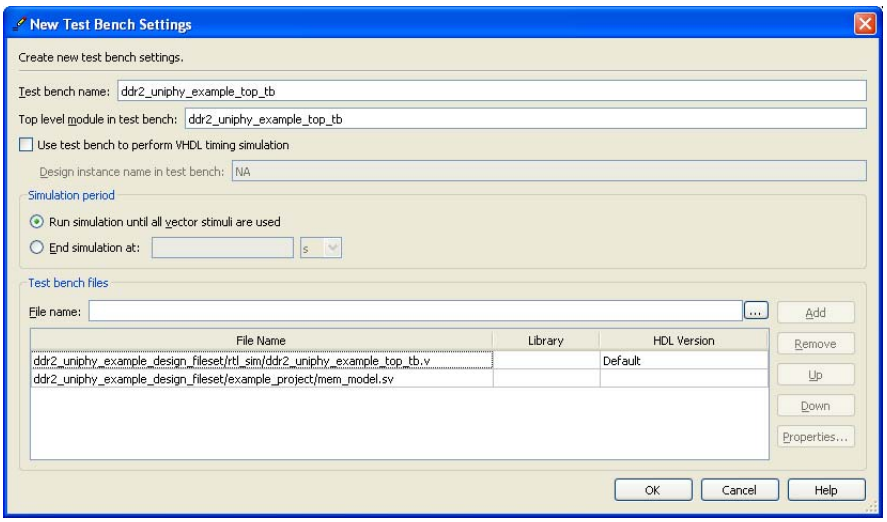
This design example includes the driver, testbench, and the memory model that allows you to perform functional simulation on the design. Use the ModelSim-Altera simulator to perform the functional simulation. This design example also includes a script file that directs the simulator to perform the necessary compilation and simulation.

To run the RTL simulation with NativeLink, perform the following steps:

1. Set the absolute path to your third-party simulator executable, by performing the following steps:
   a. On the Tools menu, click **Options**.
   b. In the **Category** list, select EDA Tools Options and set the default path for **ModelSim-Altera** to **C:\<version>\modelsim_ae\win32aloem**.
   c. Click **OK**.
2. On the Assignments menu, click **Settings**.
3. In the **Category** list, expand **EDA Tool Settings** and click **Simulation**.
4. Under **Tool name**, select **ModelSim-Altera**.
5. Under **NativeLink settings**, select **Compile test bench** and click **Test Benches**.
6. Click **New**.

7. In the **New Test Bench Settings** dialog box, perform the following steps:
   a. Type the **Test bench name** and **Top level module in test bench**.
   b. For **Top level module in test bench**, type `ddr2_uniphy_example_top_tb` for DDR2 SDRAM or `ddr3_example_top_tb` for DDR3 SDRAM.
   c. Under **Simulation period**, select **Run simulation until all vector stimuli are used**.
   d. In the **Test bench files** field, include the testbench file, *<variationname>*_**example_design_fileset/rtl_sim/<variation name>_example_top_tb.v**, and the memory model file, *<variationname>*_**example_design_fileset/example_project/mem_model.sv**. Figure 3–2 shows the testbench settings window for the DDR2 SDRAM design.
   e. Click **OK**.

**Figure 3–2. Testbench Settings**



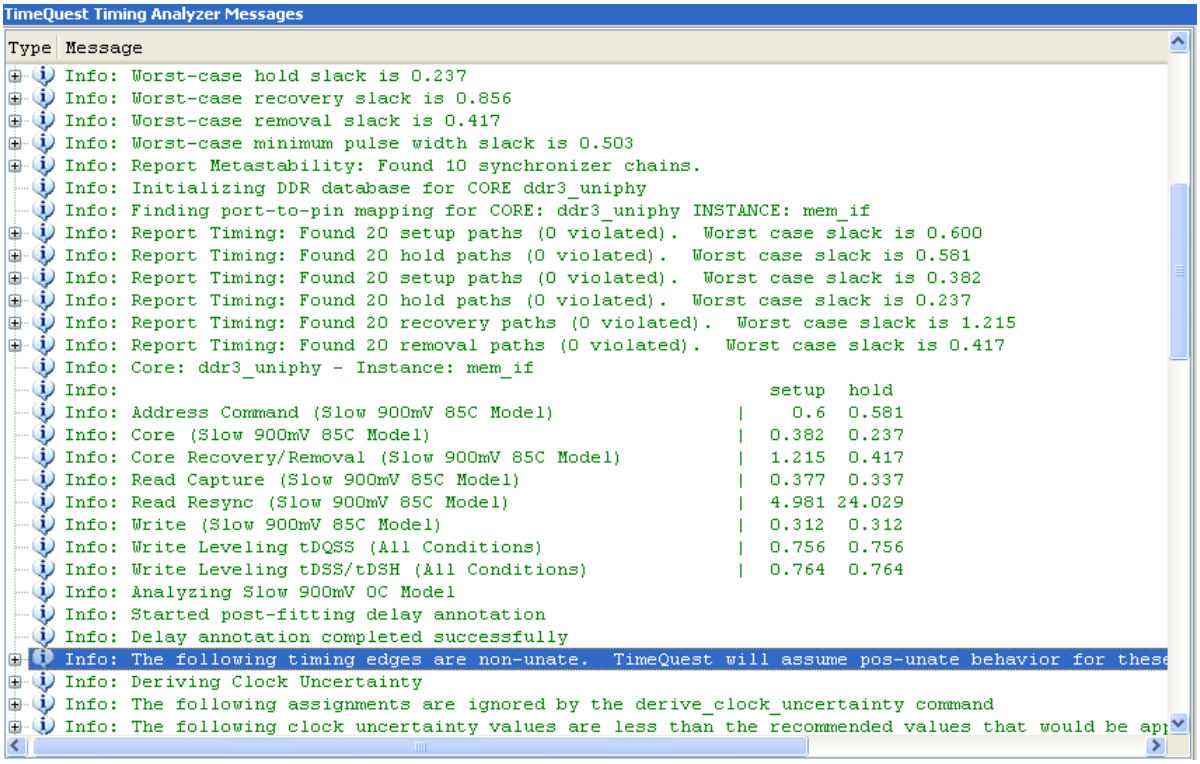

8. To elaborate your design, on the Processing menu, point to **Start** and click **Start Analysis & Elaboration**.
9. On the Tools menu, point to the **Run EDA Simulation Tool** and click **EDA RTL Simulation**. This step creates the **\simulation** directory in your project directory and a script that compiles all necessary files and runs the simulation.

# Compile Design and Verify Timing

To compile the design, on the Processing menu, click **Start Compilation**. After successfully compiling the design, the Quartus II software automatically runs the verify timing script, *<variation_name>*_**report_timing.tcl**, which produces a timing report for the design together with the compilation report.

Figure 3–3 shows the timing margin report in the message window in the Quartus II software.

**Figure 3–3. Timing Margin Report in the Quartus II Software**




The report timing script performs the following tasks:

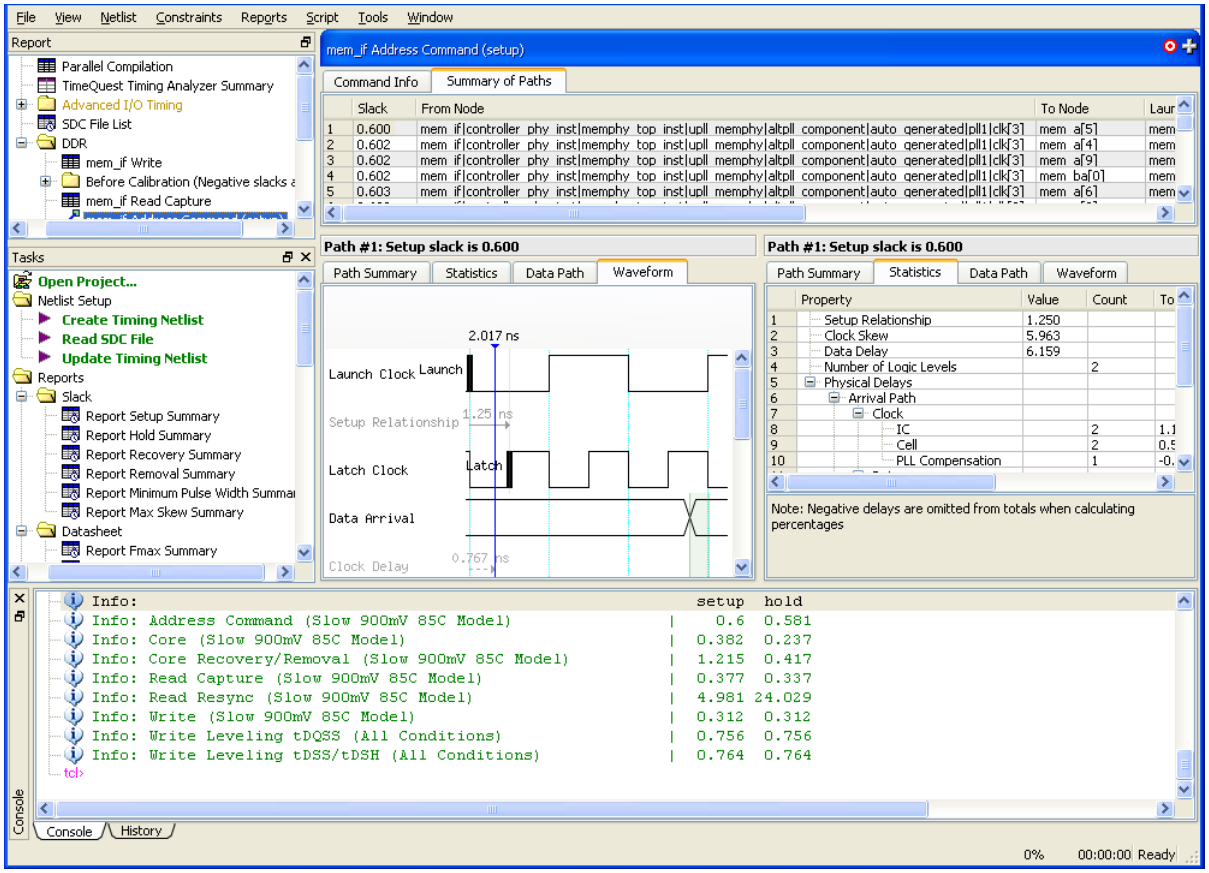
1. Creates a timing netlist.
2. Reads the **.sdc** file.
3. Updates the timing netlist.

You can also obtain the timing report by reading the **.sdc** files and running the report timing script, *<variation_name>*_**report_timing.tcl**, in the TimeQuest timing analyzer. To obtain the timing report in the TimeQuest Timing Analyzer window, perform the following steps:

1. In the Quartus II software, on the Tools menu, click **TimeQuest Timing Analyzer**.
2. On the **Tasks** pane, double-click **Report DDR** which automatically runs **Update Timing Netlist**, **Create Timing Netlist**, and **Read SDC**. After a task is executed, it turns green. After the tasks are completed, the timing margin report is generated.

Figure 3–4 shows the timing margin report in the TimeQuest Timing Analyzer window after running the report timing script. The results are the same as the Quartus II software results.

**Figure 3–4. Timing Margin Report in TimeQuest Timing Analyzer**




For more information about the TimeQuest timing analyzer, refer to *The Quartus II TimeQuest Timing Analyzer* chapter in volume 3 of the *Quartus II Handbook*. For information timing analysis, refer to the *Timing Analysis* section in volume 4 of the *External Memory Interface Handbook*.

# Verify Design on a Board

The SignalTap II Embedded Logic Analyzer shows read and write activity in the system.

For more information about using the SignalTap II Embedded Logic Analyzer, refer to the *Design Debugging Using the SignalTap II Embedded Logic Analyzer* chapter in the *Quartus II Handbook*, *AN 323: Using SignalTap II Embedded Logic Analyzers in SOPC Builder Systems* and *AN 446: Debugging Nios II Systems with the SignalTap II Embedded Logic Analyzer*.

To add the SignalTap II Embedded Logic Analyzer, perform the following steps:

1. On the Tools menu, click **SignalTap II Logic Analyzer**.
2. Under **Signal Configuration** next to the **Clock** box, click **…** (Browse Node Finder).
3. Type `*phy_clk` for the DDR2 SDRAM design or `*afi_clk` for the DDR3 SDRAM design in the **Named** box, for **Filter** select **SignalTap II: pre-synthesis** and click **List**.
4. Select **<variation_name>:mem_if|phy_clk** for the DDR2 SDRAM design or **<variation_name>:mem_if|afi_clk** for the DDR3 SDRAM design in **Nodes Found** and click **>** to add the signal to **Selected Nodes**.
5. Click **OK**.
6. Under **Signal Configuration**, specify the following settings:
   - For **Sample depth**, select **512**.
   - For **RAM type**, select **Auto**.
   - For **Trigger flow control**, select **Sequential**.
   - For **Trigger position**, select **Center trigger position**.
   - For **Trigger conditions**, select **1**.
7. On the Edit menu, click **Add Nodes**.
8. Search for specific nodes by selecting **SignalTap II: pre-synthesis** for **Filter**, and click **List**.
9. In **Nodes Found**, select the following nodes listed in Table 3–6 and click **>** to add to **Selected Nodes**.

**Table 3–6. Nodes**

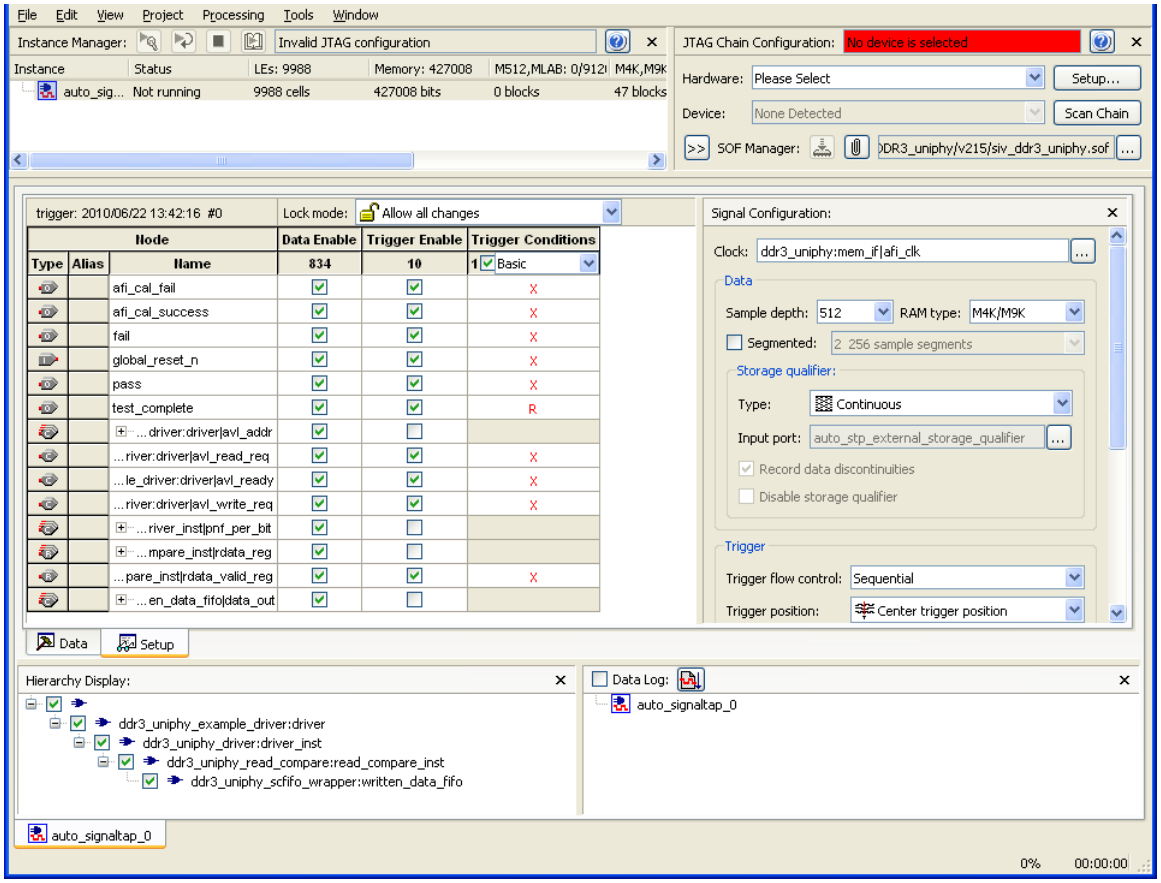
| DDR2 SDRAM | DDR3 SDRAM |
|---|---|
| ■ pass<br>■ fail<br>■ afi_cal_fail<br>■ afi_cal_success<br>■ test_complete<br>■ compare*rdata_reg<br>■ compare*rdata_valid_reg<br>■ compare*scfifo_data_out<br>■ compare*pnf_per_bit | ■ pass<br>■ fail<br>■ afi_cal_fail<br>■ afi_cal_success<br>■ test_complete<br>■ avl_addr<br>■ avl_read_req<br>■ avl_ready<br>■ avl_write_req<br>■ pnf_per_bit<br>■ rdata_reg<br>■ rdata_valid_reg<br>■ written_data_fifo\|data_out |

Do not add any DDR2 or DDR3 SDRAM interface signals to the SignalTap II Embedded Logic Analyzer. The load on these signals increases and adversely affects the timing analysis.

10. Click **OK**.
11. To reduce the SignalTap II logic size, turn off **Trigger Enable** on the following bus signals:
    - **avl_addr** (DDR3 SDRAM design only)
    - **written_data_fifo|data_out**
    - **pnf_per_bit**
    - **rdata_reg**
12. Right-click **Trigger Conditions** for the `test_complete` signal and select **Rising Edge**.

    Figure 3–5 shows the completed SignalTap II Embedded Logic Analyzer for a DDR3 design.

**Figure 3–5. SignalTap II Embedded Logic Analyzer for DDR3 Design**




13. On the File menu, click **Save**, to save the SignalTap II **.stp** file to your project.

If you see the message **Do you want to enable SignalTap II file "stp1.stp" for the current project**?, click **Yes**.

## Compile the Project

After you add the signals to the SignalTap II Embedded Logic Analyzer, on the Processing menu, click **Start Compilation** to recompile your design.

## Verify Timing

After you have recompiled the design, ensure that the TimeQuest timing analysis passes successfully. In addition to this FPGA timing analysis, check your PCB or system SDRAM timing. To run timing analysis, follow these steps:

1. On the Tools menu, click **Tcl Scripts**.
2. Select <*variation name*>**_report_timing.tcl** and click **Run**.

## Download the Object File

To download the object file to the device, perform the following steps:

1. Connect the development board to your computer.
2. On the Tools menu, click **SignalTap II Logic Analyzer**. The SignalTap II dialog box appears.
3. Click **...** to open the **Select Program Files** dialog box.
4. Select <*your project name*>**.sof**.
5. Click **Open**.
6. To download the file, click the **Program Device** button.

## Test the Design Example in Hardware

When the design example including SignalTap II successfully downloads to your development board, click **Run Analysis** to run the analysis once, or click **Autorun Analysis** to run the analysis continuously.

Figure 3–6 and Figure 3–7 show the design analysis. The `pnf_per_bit` and `pass` signals go high, indicating that the test passes in hardware.

**Figure 3–6. SignalTap II Example DDR2 SDRAM Controller with UniPHY Design Analysis**

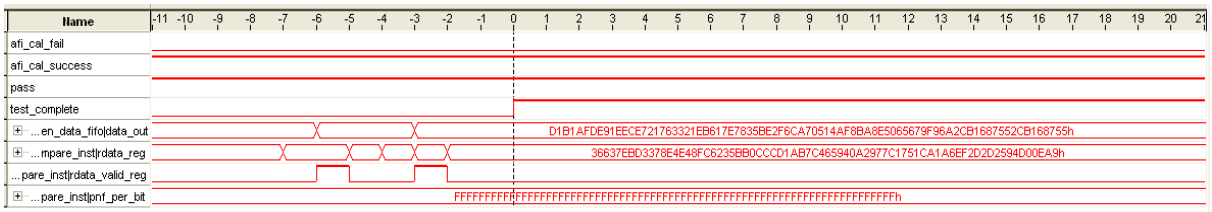
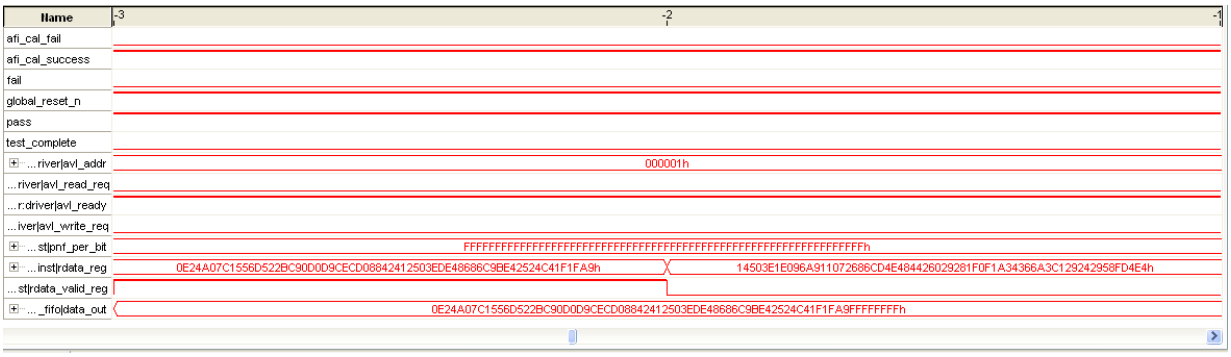



**Figure 3–7. SignalTap II Example DDR3 SDRAM Controller with UniPHY Design Analysis**



# 4. Implementing Multiple Memory Interfaces Using UniPHY

This tutorial describes how to use the design flow to implement a design with multiple memory interfaces, using an 18-bit wide, 400-MHz, 1600-Mbps QDR II+ SRAM controller with UniPHY interface and a 36-bit wide, 400-MHz, 800-Mbps RLDRAM II controller with UniPHY interface. This tutorial also provides some recommended settings to simplify the design.


This design is only for evaluation purposes. You cannot create multiple memory interfaces using different memory standards.

In this tutorial, you implement two types of different memory standards, QDR II+ SRAM and RLDRAM II, operating at the same frequency of 400 MHz and using the UniPHY IP, in a single Stratix IV device. This implementation requires device resource sharing, such as delay-locked loops (DLLs), phase-locked loops (PLLs), and on-chip termination (OCT) control blocks, and may require extra steps to create the interfaces in a Quartus II project.

The design example in this tutorial uses the Stratix IV E FPGA development kit, which includes a 18-bit wide, 72-Mb, 4-word burst, Cypress CY7C15632V18 400-MHz QDR II+ SDRAM memory component, and a 36-bit wide 576-Mb Micron MT49H16M36-18 533-MHz component.

To download the design example, **emi_multiple_rldramii_qdrii_plus_siv.zip**, go to the External Memory Interface Design Examples page. You can also use these design examples if you are targeting a HardCopy device for your design.

For more information about the design flow, refer to the *Recommended Design Flow* section in volume 1 of the *External Memory Interface Handbook*. For more information about the QDR II+ SRAM and RLDRAM II controller with UniPHY, refer to the *QDR II and QDR II+ SRAM Controller with UniPHY* and *RLDRAM II Controller with UniPHY* sections in volume 3 of the *External Memory Interface Handbook*.

## Before Creating a Design in the Quartus II Software

When creating multiple memory interfaces to be fitted in a single device, first ensure that there are enough device resources for the different interfaces. These device resources include the number of pins, DLLs, PLLs, OCT control blocks, and clock networks, where applicable. You can share the DLLs, PLLs, OCT control blocks, and clock network resources between multiple memory interfaces, but there are some sharing limitations.

For more information about device resources, refer to the *Device and Pin Planning* section in volume 2 of the *External Memory Interface Handbook*.

The following different scenarios exist for the resource sharing DLL and PLL static clocks:

- Multiple controllers with no sharing. Check the number of PLLs and DLLs that are available in the targeted device, and the number of PLLs and DLLs needed in your design. If you have enough resources for the controllers, you do not need to share the DLL and PLL static clocks.
- Multiple controllers sharing PLLs and DLLs. The UniPHY supports the master-and-slave approach, with a PLL and DLL instantiated in a master controller and the PLL clocks and DLL shared with other identical UniPHY slave controllers. You require extra steps to explicitly merge the PLLs and DLLs in the top-level design file. The PLL and DLL signals from the master controller must be connected to their equivalent signals in the slave controllers.


To share the PLLs and DLLs, ensure that the controllers are on the same or adjacent side of the device and the PLL and DLL are running at the same memory-clock frequency.

If you use FPGA OCT calibrated series, parallel, or dynamic termination for any I/Os in an external memory interface design, you require a calibration block for the OCT circuitry. This calibration block requires a pair of $R_{UP}$ and $R_{DN}$ pins, or an $R_{ZQ}$ pin that you must place within any bank with the same I/O voltage as the memory interface signals. Route these pins to the design top-level and connect to the $R_{UP}$ and $R_{DN}$, or the $R_{ZQ}$ resistors on the board.


You must prioritize the placement of the $R_{UP}$ and $R_{DN}$ pin pairs, or the $R_{ZQ}$ pin in your design, because these pins may use up to two DQS or DQ pins from a group.

After understanding the device resource limitations, determine the resources to share for your multiple controller design. You can use the resources that you determined as a framework for your Quartus II design constraints.

## Creating a PHY and Controller in a Quartus II Project

If you are creating multiple instances of the same controller with the same parameters (frequency of operation, data width, burst mode, etc.) and without sharing resources, you only need to generate the controller once. You can then instantiate this controller variation multiple times in your top-level design file. If you are using the controllers with different parameters, or different controllers, then you need to generate each variation of the memory controller individually. Extra steps are required to modify the RTL generated by the MegaWizard Plug-In Manager for resource sharing.

The high-performance memory controller with UniPHY is generated with a top-level design file called <*variation_name*>**_example_top.v** or **.vhd**, where *variation_name* is the name of the memory controller that you typed in the first page of the MegaWizard Plug-In Manager. This example top-level file instantiates the memory controller (<*variation_name*>**.v** or **.vhd**) and an example driver, which performs a comparison test after reading back data that was written to the memory.

When creating a multiple memory interface design, you have the option to combine certain aspects of the design flow illustrated in the following examples:

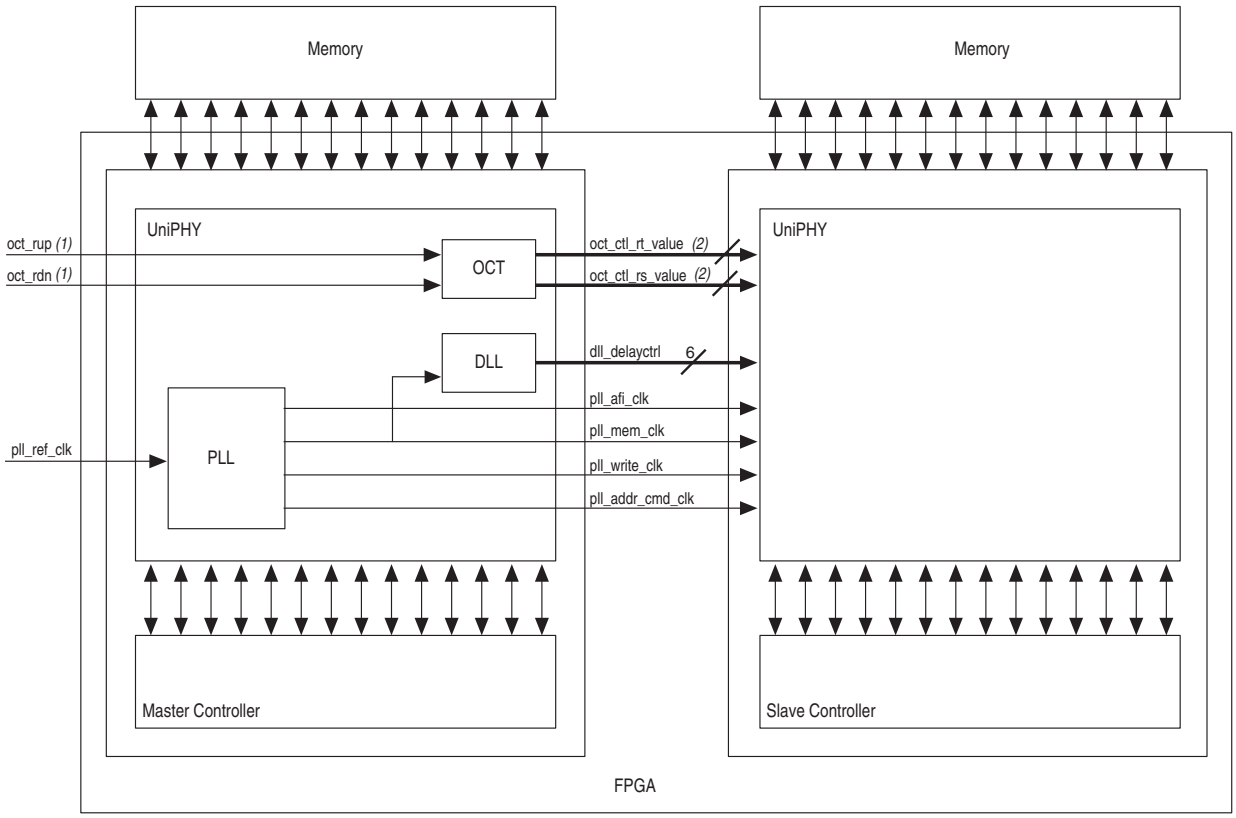
- Use one of the top-level designs to instantiate all the memory controllers in the design, or create a new top-level design.

- Either modify one of the example drivers to test all the memory interfaces in the design, or instantiate an example driver with each memory controller.
- Simulate each memory interface separately, or create a testbench which combines all the memory interfaces in the design.

In this tutorial, you create a new top-level design schematic with an example driver instantiated in both RLDRAM II and QDR II+ SRAM controllers.

Figure 4–1 shows how the DLL and PLL clocks and the OCT control blocks are connected for the clock sharing scheme.

**Figure 4–1. A Master-Slave Scenario Sharing DLL and PLL Clocks and OCT Control Block**




**Note to Figure 4–1:**

(1) For Stratix V designs, the input to the OCT control is only an $R_{ZQ}$ pin, `oct_rzqin`.

(2) For Arria II GX designs, the output termination control bus name is `oct_ctl_value`, which connects to the ALT_OCT megafunction's output termination control.

## Sharing PLLs and DLLs

The controllers with UniPHY memory interface require one PLL and one DLL to produce the clocks and delay codeword. Altera recommends that you use the master-and-slave approach to share the PLL and DLL when there are not enough resources. The PLL and DLL output signals are inputs or outputs in the top-level file, and an additional parameter, `PLL_DLL_MASTER` determines the direction of these output signals.

If you turn on **Master for PLL/DLL sharing** on the DDR2, DDR3 SDRAM, RLDRAM II, or QDR II and QDR II+ SRAM controllers with UniPHY parameter interface, the `PLL_DLL_MASTER` parameter is 1 and the RTL instantiates the PLL and DLL which drive the clock and DLL codeword inputs and outputs. If you turn off **Master for PLL/DLL sharing**, the `PLL_DLL_MASTER` parameter is 0, and the wires previously connected to the outputs of the PLL and DLL are now assigned to the clock and DLL codeword inputs and outputs. During synthesis, the direction is automatically resolved to either input or output.

By sharing the PLL output clocks, `pll_afi_clk`, `pll_mem_clk`, `pll_write_clk`, and `pll_add_cmd_clk`, you can reduce the required number of clocks up to four clock networks.

The number of interfaces that you can share in this clock sharing scheme is limited by the number of PLLs and pins available in the device. UniPHY-based memory interfaces may share the output clocks between multiple controllers in the following two conditions:

- All UniPHY-based memory interfaces are either full-rate or half-rate designs.
- All UniPHY-based memory interfaces have the same PLL input and output frequencies.
- The `ref_clk` input of each PLL are connected to a clock signal derived from a signal clock source. You can use a clock fanout buffer to create separate PLL reference clocks from a single clock source.
- The memory controllers' circuitry is located in the same two device quadrants.
- All the clocks are routed as required by their global network clock type.
- The general routing rules defined in the relevant memory standard layout section are met.

## Sharing OCT Control Block

The controllers with UniPHY memory interface require an OCT control block to calibrate the I/O buffer on chip impedances off external resistors connected to the $R_{UP}$ and $R_{DN}$, or $R_{ZQ}$ pins on the FPGA. These resistance inputs connect directly to the OCT control block, which outputs 14- or 16-bit termination control buses.

If you turn on **Master for OCT control block** in the DDR2, DDR3 SDRAM, RLDRAM II, or QDR II and QDR II+ SRAM controllers with UniPHY parameter interface, the RTL instantiates the OCT control block in the UniPHY IP.

If you turn off **Master for OCT control block**, the UniPHY IP does not include the OCT control block and the RTL instantiates the block in the example top level. Then, you must explicitly connect the output termination control buses of the OCT control block to the PHY for proper operation.

# System Requirements

This tutorial implements RLDRAM II and QDR II+ SRAM interfaces on the Stratix IV E development board, with the RLDRAM II as the master controller instantiating its own PLL to be shared with the QDR II+ SRAM slave controller. This tutorial requires the following software and hardware:

- Quartus II software version 10.1
- RLDRAM II Controller with UniPHY version 10.1
- QDR II and QDR II+ SRAM Controllers with UniPHY version 10.1
- ModelSim-Altera version 6.5e or later
- Stratix IV E FPGA development board

# Create a Quartus II Project

Create a project in the Quartus II software that targets a EP4SE530H35C2 device.

For detailed step-by-step instructions to create a Quartus II project, refer to the Quartus II software tutorial by clicking the Help menu in the Quartus II window and selecting **PDF Tutorials**.

# Instantiate and Parameterize the Controllers

After creating a Quartus II project, instantiate and parameterize the RLDRAM II controller with UniPHY and QDR II+ SRAM controller with UniPHY.

## Instantiate the RLDRAM II Master Controller

To instantiate the RLDRAM II controller with UniPHY, perform the following steps:

1. Start the MegaWizard Plug-In Manager.
2. In the MegaWizard Plug-In Manager, expand **External Memory** in the **Interfaces** folder and select **RLDRAM II Controller with UniPHY v10.1**.
3. Type `rldramii` for the name of the RLDRAM II controller with UniPHY.

## Parameterize the RLDRAM II Master Controller

To parameterize the RLDRAM II controller with UniPHY to interface with a 400-MHz, 36-bit wide RLDRAM II interface, perform the following steps:

1. In the **Presets** list, select **MT49H16M36-18** and click **Apply**. The memory parameters are set automatically based on the memory preset settings you choose. You may change the parameters to suit your design requirements.

   CAUTION Selecting a new memory preset overwrites your existing settings. You need to enter the settings again.

2. In the **General Settings** tab, under **Clocks**, for **Memory clock frequency**, type **400** MHz as the system frequency.

3. For **PLL reference clock frequency**, type **125** MHz to match the on-board oscillator.
4. For **Full or half rate on Avalon-MM interface**, select **Half**. This option allows you to choose between the full-rate and half-rate controller, and define the bus data width between the controller and the PHY.

   For more information about selecting half-rate or full-rate controller, refer to the *Device and Pin Planning* section in volume 2 of the *External Memory Interface Handbook*.

5. For **Additional Address/Command clock phase**, select **0**.
6. Under **Advanced Settings**, for **Maximum Avalon-MM burst length**, specify the burst length based on the burst count sent from the core fabric. For this design example, select **128**.
7. For **I/O standard**, select **1.5-V HSTL**.
8. Turn on **Master for PLL/DLL sharing** for the UniPHY to instantiate its own PLL. Turn off this option if you want to parameterize the slave controller for the UniPHY.

   If you want to migrate your design to a HardCopy device, turn on **HardCopy Compatibility Mode**, and select the appropriate **Reconfigurable PLL Location**.

9. Turn on **Master for OCT control block** for the UniPHY to instantiate the required OCT control block. Turn off this option if you want to instantiate the OCT block at the top level and connect the series and parallel termination control bus signals to the PHY.
10. Under **Example Testbench Simulation Options**, turn on **Skip memory initialization**. Enabling this option allows you to skip the memory initialization sequence during simulation so that you can reduce the simulation time.
11. Under **Controller Settings**, select **2** for **Controller Latency**.

    Select the controller latency based on your targeted frequency. For more information, refer to the Latency chapter in the *RLDRAM II Controller with UniPHY IP User Guide* section in the *External Memory Interface Handbook*.

12. In the **Memory Parameters** tab, for **Memory Mode Register Configuration**, select **tRC=6, tRL=6, tWL=7, f=400–175MHz**.
13. In the **Board Settings** tab, set the **Setup and Hold Derating** parameters to the specified values in Table 4–1.

**Table 4–1. Setup and Hold Derating Parameters for MT49H16M36-18 Device**

| Setup and Hold Derating Parameter | Value (ps) |
|---|---|
| tAS VREF to CK/CK# Crossing | 0 |
| tAS VIH MIN to CK/CK# Crossing | -100 |
| tAH CK/CK# Crossing to Vref | 0 |

**Table 4–1. Setup and Hold Derating Parameters for MT49H16M36-18 Device**

| Setup and Hold Derating Parameter | Value (ps) |
|---|---|
| tAH CK/CK# Crossing to VIH MIN | -50 |
| tDS Vref to CK/CK# Crossing | 0 |
| tDS VIH MIN to CK/CK# Crossing | -100 |
| tDH CK/CK# Crossing to Vref | 0 |
| tDH CK/CK# Crossing to VIH MIN | -50 |

The Intersymbol Interference (ISI) parameters are not applicable for single chip-select configurations. Set these parameters to **0 ps**.

For more accurate timing analysis, assign the board trace delay model for every single pin. The board skew values are used to calculate the overall system timing margin. Change these values based on the performance of your board if you are not using the board trace delay model, or if you cannot accurately assign the board trace model for each pin. Refer to "Enter Board Trace Delay Models" on page 4–14 for more information about board trace delay model.

14. Set the **Board Skews** parameters to the specified values in Table 4–2.

**Table 4–2. Board Skews Parameters for EP4SE530H35C2 Device**

| Board Skews Parameter | Value (ps) |
|---|---|
| Minimum delay difference between CK and DK | 35 |
| Maximum delay difference between CK and DK | 35 |
| Maximum delay difference between devices | 0 |
| Maximum skew within QK group | 15 |
| Maximum skew between QK groups | 1 |
| Maximum skew within Address/Command bus | 73 |
| Average delay difference between Address/Command and CK | 23 |
| Average delay difference between write data signals and DK | 5 |
| Average delay difference between read data signals and QK | 5 |

15. Click **Finish** to generate your MegaCore function variation. The MegaWizard Plug-In Manager generates all the files necessary for your RLDRAM II controller with UniPHY, and an example top-level design, which you may use to test or verify the board operation.
16. Click **Exit** to close the MegaWizard Plug-In Manager after you have reviewed the generation report.

For detailed step-by-step instructions for parameterizing the RLDRAM II controller with UniPHY, refer to *Volume 3: Implementing Altera Memory Interface IP* of the *External Memory Interface Handbook*.

## Instantiate the QDR II and QDR II+ SRAM Slave Controller

To instantiate the QDR II and QDR II+ SRAM controller with UniPHY, perform the following steps:

1. Start the MegaWizard Plug-In Manager.
2. In the MegaWizard Plug-In Manager, expand **External Memory** in the **Interfaces** folder and select **QDR II and QDR II+ SRAM Controller with UniPHY v10.1**.
3. Type `qdrii_plus` for the name of the QDR II and QDR II+ SRAM controller with UniPHY.

## Parameterize the QDR II and QDR II+ SRAM Slave Controller

To parameterize the QDR II+ SRAM controller with UniPHY to interface with a 400-MHz, 18-bit wide QDR II+ interface, perform the following steps:

1. In the **Presets** list, select a memory preset that matches your memory based on the memory device specifications. Select **CY7C1563KV18-400** (which meets the requirements for CY7C1563V18-400) and click **Apply**. The memory parameters are set automatically based on the memory preset settings you choose. You may change the parameters to suit your design requirements.

   CAUTION Selecting a new memory preset overwrites your existing settings. You need to enter the settings again.

2. In the **General Settings** tab, under **Clocks**, for **Memory clock frequency**, type **400** MHz as the system frequency.
3. For **PLL reference clock frequency**, type **125** MHz to match the on-board oscillator.
4. For **Full or half rate on Avalon-MM interface**, select **Half**. This option allows you to choose between the full-rate and half-rate controller, and define the bus data width between the controller and the PHY.

   For more information about selecting half-rate or full-rate controller, refer to the *Device and Pin Planning* section in volume 2 of the *External Memory Interface Handbook*.

5. For **Additional Address/Command clock phase**, select **0**.
6. Under **Advanced PHY Settings**, for **Maximum Avalon-MM burst length**, specify the burst length based on your system. For example, if your system only generates up to 64 beats, select **64**.
7. For **I/O standard**, select **1.5-V HSTL**.
8. Turn off **Master for PLL/DLL sharing** so that the PLL clock shares the clock with the RLDRAM II controller with UniPHY.

   If you want to migrate your design to a HardCopy device, turn on **HardCopy Compatibility Mode**, and select the appropriate **Reconfigurable PLL Location**.

9. Turn off **Master for OCT control block** to share the OCT control block from the RLDRAM II controller with UniPHY .
10. Under **Example Testbench Simulation Options**, turn on **Skip memory initialization**. Enabling this option allows you to skip the memory initialization sequence during simulation so that you can reduce the simulation time.
11. Under **Controller Settings**, select **1** for **Controller Latency**.
12. In the **Board Settings** tab, set the **Board Skews** parameters to the specified values in Table 4–2.

**Table 4–3. Board Skews Parameters for EP4SE530H35C2 Device**

| Board Skews Parameter | Value (ps) |
|---|---|
| Maximum delay difference between devices | 0 |
| Maximum skew within write data group (i.e. K group) | 8 |
| Maximum skew within read data group (i.e. CQ group) | 8 |
| Maximum skew between CQ groups | 0 |
| Maximum skew within Address/Command bus | 92 |
| Average delay difference between Address/Command and K | 39 |
| Average delay difference between write data signals and K | 35 |
| Average delay difference between read data signals and CQ | 1 |

The Intersymbol Interference (ISI) parameters are not applicable for single chip-select configurations. Set these parameters to **0 ps**.

For more accurate timing analysis, assign the board trace delay model for every single pin. The board skew values are used to calculate the overall system timing margin. Change these values based on the performance of your board if you are not using the board trace delay model, or if you cannot accurately assign the board trace model for each pin. Refer to "Enter Board Trace Delay Models" on page 4–14 for more information about board trace delay model.

13. Click **Finish** to generate your MegaCore function variation. The MegaWizard Plug-In Manager generates all the files necessary for your QDR II+ SRAM controller with UniPHY, and an example top-level design, which you may use to test or verify the board operation.
14. Click **Exit** to close the MegaWizard Plug-In Manager after you have reviewed the generation report.

For detailed step-by-step instructions for parameterizing the QDR II+ SRAM controller with UniPHY, refer to *Volume 3: Implementing Altera Memory Interface IP* of the *External Memory Interface Handbook*.

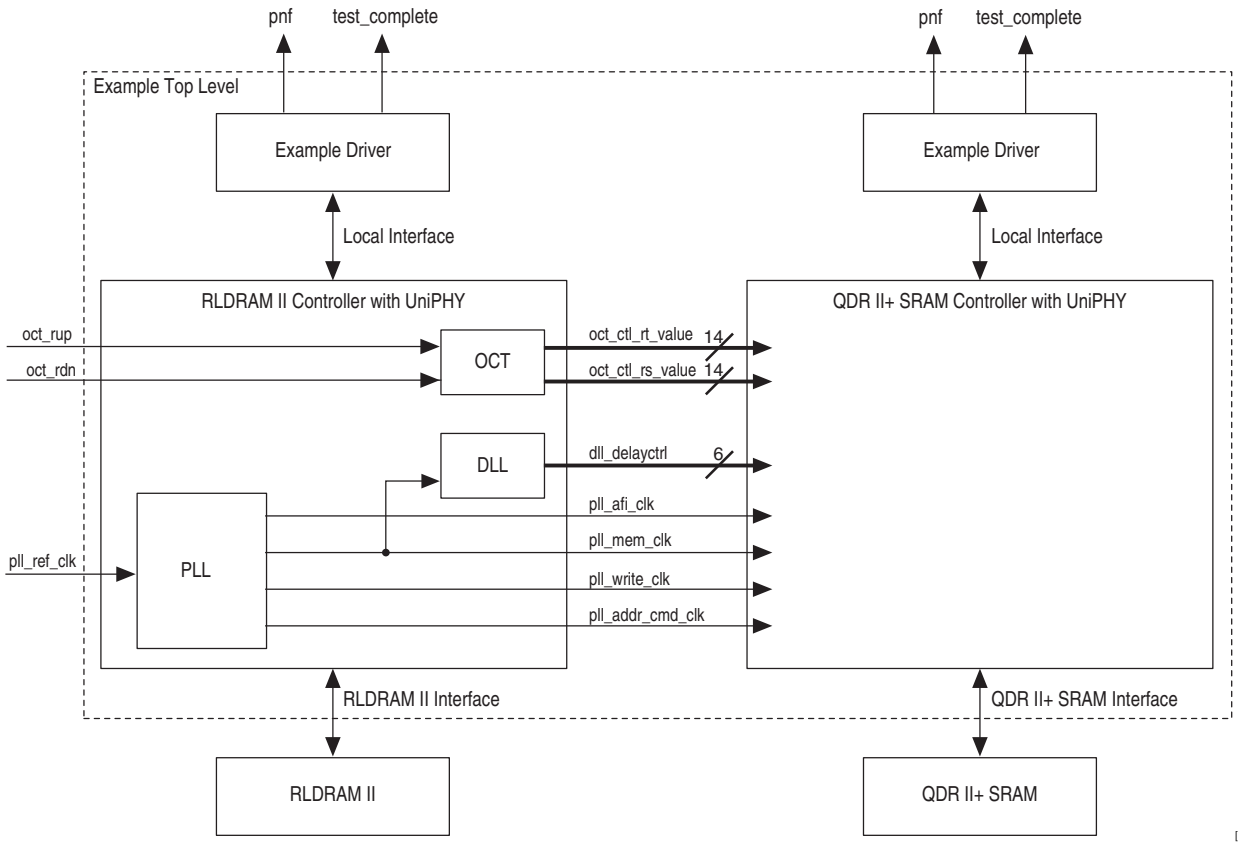
# Add Constraints

After instantiating the RLDRAM II and QDR II+ SRAM controllers with UniPHY, the Quartus II software generates the constraints files for the design example. Apply these constraints to the design before compilation.

## Create Top-Level Project

To create a top-level design for multiple memory interfaces, use the top-level design with master controller instance and instantiate other slave memory interfaces, and add them to this top-level design. To demonstrate the multiple memory interfaces, instantiate an example driver with each memory controller. In this design example, the top-level design is created by instantiating the RLDRAM II example top-level design and adding the instantiation of QDR II+ SRAM interface to the top-level design. Figure 4–2 shows the top-level diagram for the multiple memory controllers.

**Figure 4–2. Multiple Memory-Controller Top-Level Diagram**




For detailed information about how you can create a top-level design for multiple memory interfaces, refer to the Altera-provided **siv_e_rldramii_qdrii_plus.v** file.

Alternatively, you can create your own new top-level design to instantiate all the memory controllers in the design.

## Add PLL Sharing Constraint

Enable the QDR II+ SRAM slave controllers to share the RLDRAM II master's PLL output clocks by performing the following steps.

1. On the File menu, click **Open**.
2. Browse to **qdrii_plus_example_design_fileset\qdrii_plus.sdc** and click **Open**.
3. Locate the following PLL clock assignments:

```
set master_corename "_MASTER_CORE_"
set master_instname "_MASTER_INST_"
```

   and change the master instance and core names to match the names you specify in your design, for example:

```
set master_corename “rldramii”
set master_instname “master_mem_if”
```

4. Locate the following clock assignment:

```
set pll_k_clock $pins(pll_k_clock)
```

   and change to:

```
set pll_k_clock
master_mem_if|controller_phy_inst|memphy_top_inst|upll_memphy|altpl
l_component|auto_generated|wire_pll1_clk[1]~clkctrl|outclk
```

5. Locate the following clock assignment:

```
set local_pll_mem_clk
"${::master_instname}|${::master_corename}_pll_mem_clk"
```

   and change to:

```
set local_pll_mem_clk
"${::master_instname}|${::master_corename}_clkbuf_ckdk_common_clk"
```

6. Click **Save**.

Apply the PLL sharing constraints to all the slave controllers if you have more than one slave controller in your design.

## Device Settings

Set the unused pin and device voltage correctly before adding the constraint and pin assignment. Perform the following steps to set the device voltage and unused pins:

1. In the **Assignments** list, select **Device** and click **Device and Pin Options**.
2. Click the **Unused Pins** tab, and for **Reserve all unused pins** select **As input tri-stated with weak pull-up**.
3. Click the **Voltage** tab, and for **Default I/O standard** select **1.5 V.**
4. Click **OK**.

## Add Example Project

Add the example project files to your project, by performing the following steps:

1. On the Project menu, click **Add/Remove Files in Project**.

2. Browse to the <*variation_name*>**_design_fileset** directory.
3. Select <*variation_name*>**.qip**, and click **Open**.

The parameter interface generates the **.qip** file, which contains information about the generated IP core. In most cases, the **.qip** file contains all of the necessary assignments and information required to process the MegaCore function or system in the Quartus II compiler. The parameter interface generates a single **.qip** file for each MegaCore function.

4. Select all the files with the <*variation_name*> prefix, except for the <*variation_name*>**_mem_model.sv** file, and click **Open**.
5. Click **OK**.

## Set Top-Level Entity

The top-level entity of the project must be set to the correct entity. To set the top-level file, perform the following steps:

1. On the File menu, click **Open**.
2. Browse to the <*top_level_name*>**.v** or **.vhd** file and click **Open**.
3. On the Project menu, click **Set as Top-Level Entity**.

## Add Pin and DQ Group Assignments

The pin assignment scripts, **rldramii_pin_assignments.tcl** and **qdrii_plus_pin_assignments.tcl**, set up the I/O standards for the RLDRAM II and QDR II+ SRAM with UniPHY interface. These scripts do not include assignments for the design's PLL input clock. You must create a PLL clock for the design, and provide pin assignments for the signals of both the example driver and testbench that the MegaCore variations generate.

Run the **rldramii_pin_assignments.tcl** and **qdrii_plus_pin_assignments.tcl** scripts to add the pins, I/O standards, and DQ group assignments to the design example. To run the pin assignment scripts, perform the following steps:

1. On the Processing menu, point to **Start** and then click **Start Analysis & Synthesis**.
2. In the Tools menu, click **Tcl Scripts**.
3. Locate the **rldramii_pin_assignments.tcl** file.
4. Click **Run**.
5. Repeat steps 1 to 4 to locate and run the **qdrii_plus_pin_assignments.tcl** script.

The PLL input clock I/O does not need to have the same I/O standard as the memory interface I/Os. However, you may see a no-fit error as the bank in which the PLL input clock I/O gets placed becomes unusable for placement of the memory interface I/Os because of the incompatible $V_{CCIO}$ levels. Altera recommends that you assign the same I/O standard to the PLL input clock I/O.

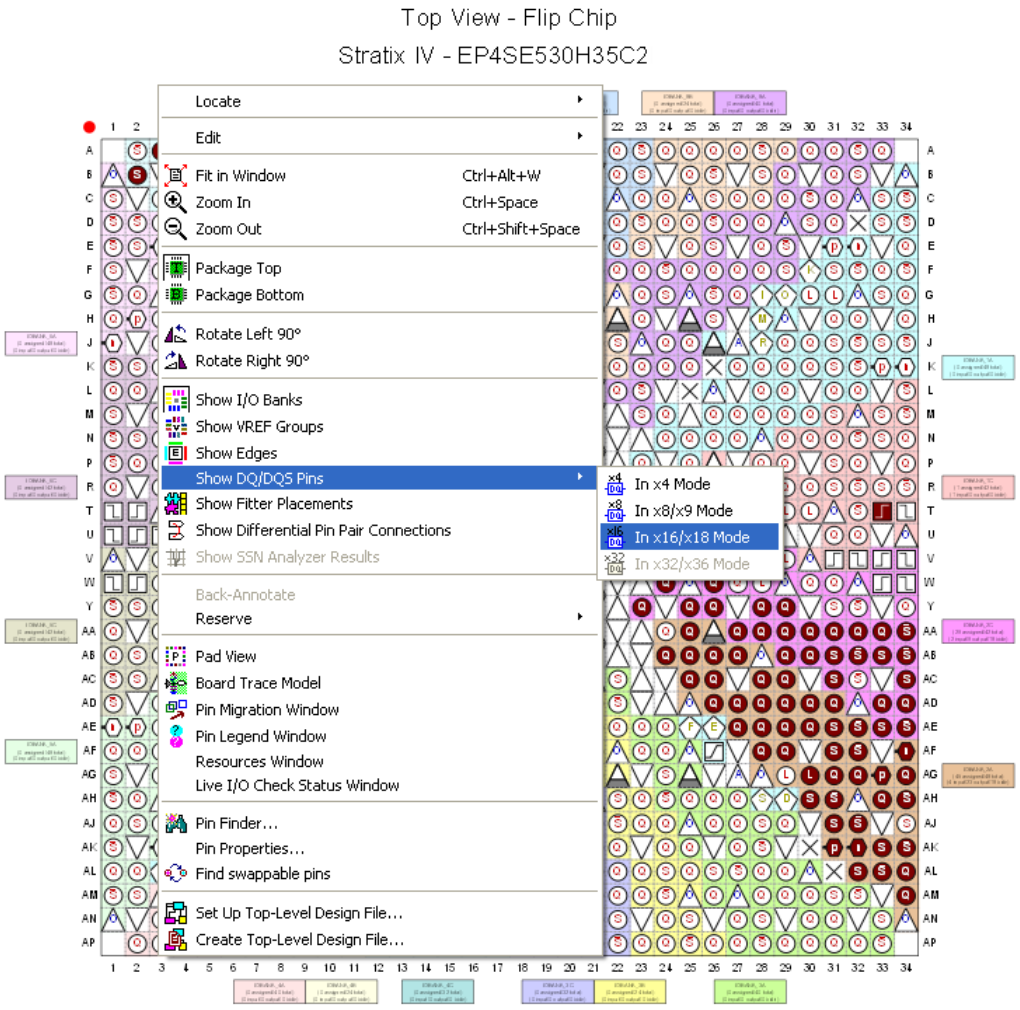
## Enter Pin Location Assignments

To enter the pin location assignments, assign all of your pins, so that the Quartus II software fits your design correctly and performs correct timing analysis. To assign the pin locations, run the Altera-provided **siv_e_rldramii_qdrii_plus_pin_location.tcl** file for the Stratix IV E FPGA development board.

Alternatively, to manually assign the pin locations in the Pin Planner, perform the following steps:

1. On the Assignments menu, click **Pin Planner**.
2. To view the DQS groups in Pin Planner, right-click and select **Show DQ/DQS Pins**, and click **In ×16/×18 Mode**. The Pin Planner shows each DQS group in a different color and with a different legend: S = `DQS` pin, Sbar = `DQSn` pin and Q = `DQ` pin, as shown in Figure 4–3.
3. To identify differential I/O pairs, right-click in Pin Planner and select **Show Differential Pin Pair Connections**. The Pin Planner shows a red line between each pin pair.

**Figure 4–3. Quartus II Pin Planner, Show DQ/DQS Pins, In ×16/×18 Mode**



For more information about pin location assignments, refer to the *Device and Pin Planning* section in volume 2 of the *External Memory Interface Handbook*. For more information about how to use the Quartus II Pin Planner, refer to the *I/O Management* chapter in volume 2 of the *Quartus II Handbook*.

## Assign Virtual Pins

The example top-level design, which is auto-generated by the RLDRAM II or QDR II and QDR II+ SRAM controller with UniPHY, includes an example driver to simulate the interface. This example driver is not part of the RLDRAM II or QDR II and QDR II+ SRAM controller with UniPHY IP, but allows easy testing of the IP.

The example driver outputs several test signals to indicate its operation and the status of the simulated memory interface. These test signals are `reset_request_n`, `afi_cal_fail`, `afi_cal_success`, `fail`, `test_complete`, and `pass`. The test signals are not part of the memory interface, but are to facilitate testing. You must either connect these signals to a debug bus or assign the signals to virtual pins using the Quartus II Assignment Editor. When using the example driver for testing, do not remove the test signals from the top-level signal list. If you remove the test signals from the top-level module, the Quartus II software optimizes the driver away, and the example driver fails.

To assign virtual pin assignments for the Stratix IV E FPGA development board, run the Altera-provided **siv_e_ rldramii_qdrii_plus_exdriver_vpin.tcl** file, or manually assign the virtual pin assignments using the Assignment Editor.

## Enter Board Trace Delay Models

For accurate I/O timing analysis, the Quartus II software must be aware of the board trace and loading information. You should derive and refine board trace and loading information during your PCB development process of prelayout (line) and post-layout (board) simulation.

To enter board trace information, perform the following steps:

1. In Pin Planner, select the pin or group of pins for which you want to enter board trace parameters.
2. Right-click and select **Board Trace Model**.
3. Enter the board trace model based on the trace length, capacitance per length, inductance per length, pull-up resistance, pull-down resistance and so on based on your board.

   To apply the board trace model assignments for the Stratix IV E FPGA development board, run the Altera-provided **SIV_E_BTModels.tcl** file.

Table 4–4 shows the board trace model parameters for the Stratix IV E FPGA development board.

**Table 4–4. Stratix IV E FPGA Board Trace Model Summary**

| Net | Near (FPGA End of Line) | | | | | | Far (Memory End of Line) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Length (in) | C_per_ length (pF/In) | L_per_ length (H/In) | Cn (pF) | Rns (W) | Rnh (W) | Length (in) | C_per_ length (pF/In) | L_per_ length (nF/In) | Cf (pF) | Rfh/Rfp (W) |
| **RLDRAM II** | | | | | | | | | | | |
| Addr *(1)* | 2.231 | 3.19 | 8.13 | — | — | — | — | — | — | 2.0 | 56 |
| CLK | 2.107 | 2.86 | 9.14 | — | — | — | — | — | — | 2.0 | 56 |
| DQ/DM | 1.868 | 3.19 | 8.13 | — | — | — | — | — | — | 2.0 | 50 |
| DK0 | 1.895 | 2.86 | 9.14 | — | — | — | — | — | — | 2.0 | 56 |
| DK1 | 1.897 | 2.86 | 9.14 | — | — | — | — | — | — | 2.0 | 56 |
| **QDR II+ SRAM** | | | | | | | | | | | |
| Addr *(2)* | 3.046 | 3.19 | 8.13 | — | — | — | — | — | — | 2.0 | 56 |
| K | 2.811 | 3.19 | 8.13 | — | — | — | — | — | — | 2.0 | 50 |
| D Group | 3.027 | 3.19 | 8.13 | — | — | — | — | — | — | 2.0 | 56 |
| doffn | 2.341 | 3.19 | 8.13 | — | — | — | — | — | — | 2.0 | 56 |
| bws_n | 3.055 | 3.19 | 8.13 | — | — | — | — | — | — | 2.0 | 56 |

**Notes to Table 4–4:**

(1) Addr = Addr, ba, cs_n, ref_n, and we_n.

(2) Addr = Addr, rps_n, and wps_n.

# Perform RTL Simulation (Optional)

This section describes RTL simulation. Both the RLDRAM II and QDR II+ SRAM controllers with UniPHY automatically generate the RTL simulation memory model files, *<variation name>*_**mem_model.sv**, and testbench top-level files, *<variation name>*_**example_top_tb.v**, located in the **rtl_sim** folders. You can use these files to verify your design.

When creating a multiple-memory interface design, you must modify the testbench to perform functional simulation. You can simulate each interface separately, or create a testbench that combines all the memory interfaces in the design.

To modify the testbench for the RLDRAM II and QDR II+ SRAM memory interfaces, perform the following steps:

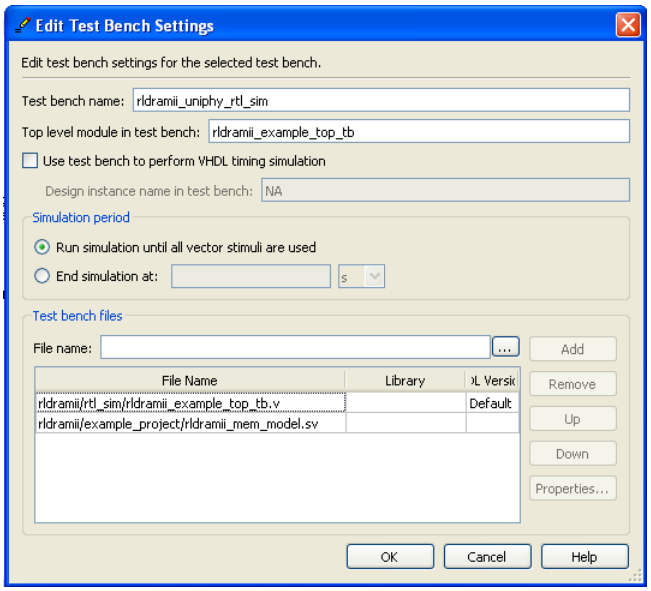
1. On the File menu, click **Open**.
2. Browse to *<variation_name>*_**example_design_fileset\rtl_sim** directory, and select *<variation_name>*_**example_top_tb.v**.
3. Change the module name for the **dut** instance to the top-level name, `siv_e_rldramii_qdrii_plus`.

Ensure that the ports names of this module are matched to those in your top-level file, and match those in your design.

To run the RTL simulation with NativeLink, perform the following steps:

1. Set the absolute path to your third-party simulator executable, by performing the following steps:
   a. On the Tools menu, click **Options**.
   b. In the **Category** list, select EDA Tools Options and set the default path for **ModelSim-Altera** to **C:\<version>\modelsim_ae\win32aloem**.
   c. Click **OK**.
2. On the Assignments menu, click **Settings**.
3. In the **Category** list, expand **EDA Tool Settings** and click **Simulation**.
4. Under **Tool name**, select **ModelSim-Altera**.
5. Under **NativeLink settings**, select **Compile test bench** and click **Test Benches**.
6. Click **New**.
7. In the **Edit Test Bench Settings** dialog box, perform the following steps:
   a. For **Test bench name**, type `rldramii_uniphy_rtl_sim` for RLDRAM II or `qdrii_plus_rtl_sim` for QDR II+ SRAM.
   b. For **Top level module in test bench**, type `rldramii_example_top_tb` for RLDRAM II or `qdrii_plus_example_top_tb` for QDR II+ SRAM.
   c. Under **Simulation period**, select **Run simulation until all vector stimuli are used**.
   d. In the **Test bench files** field, include the testbench file, *<variation name>*_**example_top_tb.v**, and the memory model file, *<variation name>*_**mem_model.sv**. (Figure 4–4).
   e. Click **OK**.

**Figure 4–4. Testbench Settings**



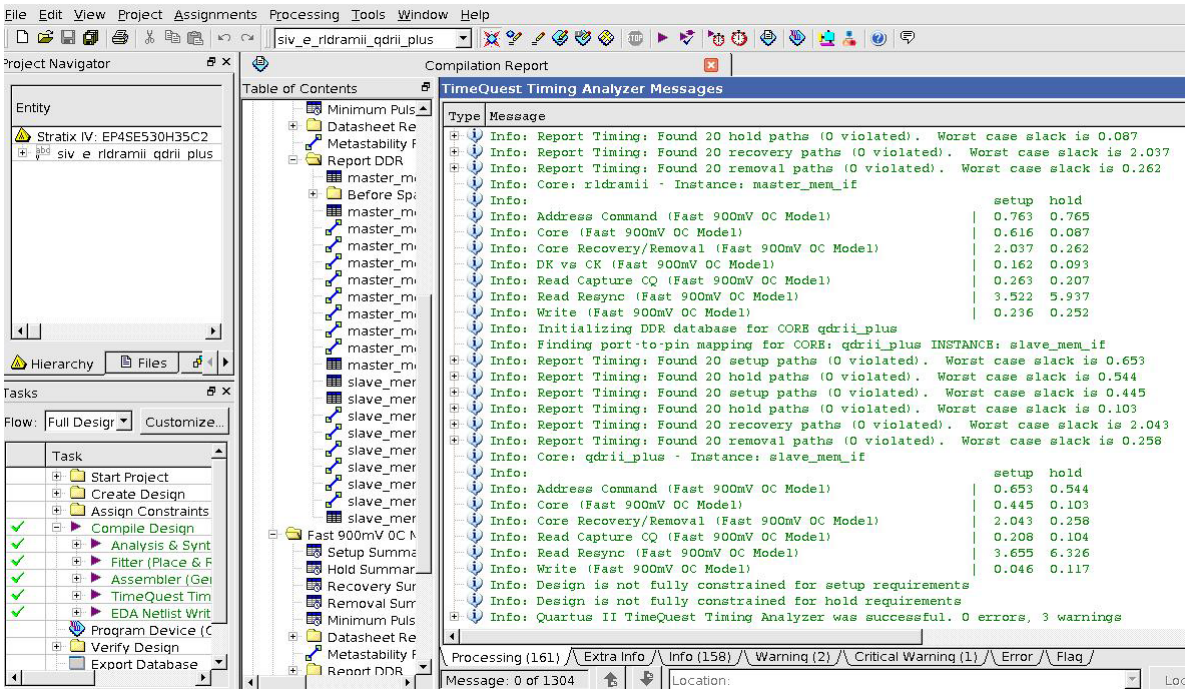
8. To elaborate your design, on the Processing menu, point to **Start** and click **Start Analysis & Elaboration**.
9. On the Tools menu, point to the **Run EDA Simulation Tool** and click **EDA RTL Simulation**. This step creates the **\simulation** directory in your project directory and a script that compiles all necessary files and runs the simulation.

# Compile Design and Verify Timing

To compile the design, on the Processing menu, click **Start Compilation**. After successfully compiling the design, the Quartus II software automatically runs the verified timing script of the RLDRAM II and QDR II+ SRAM memories, *<variation_name>*_**report_timing.tcl**, which produces a timing margin report for the design together with the compilation report.

Figure 4–5 shows the timing margin report in the message window in the Quartus II software.

**Figure 4–5. Timing Margin Report in the Quartus II Software**




The report timing script performs the following tasks:

1. Creates a timing netlist.
2. Reads the **.sdc** file.
3. Updates the timing netlist.

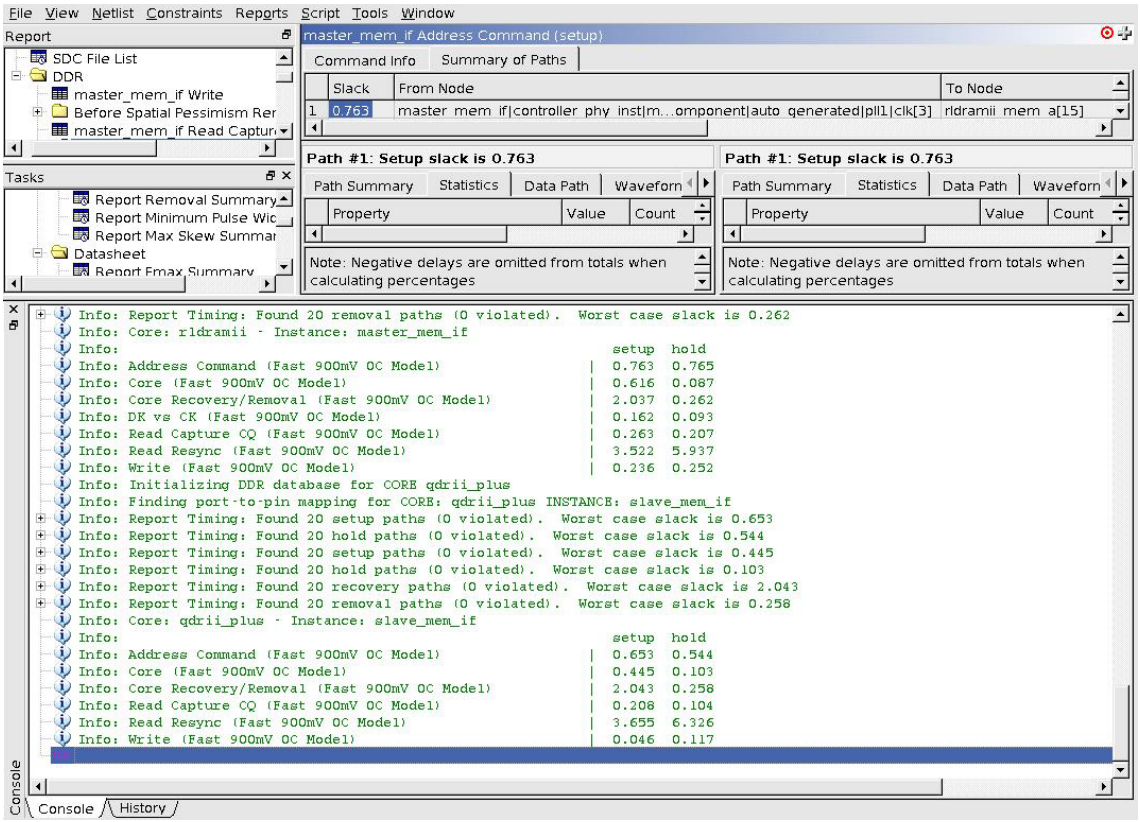
You can also obtain the timing report by running the report timing script in the TimeQuest timing analyzer. To obtain the timing report in the TimeQuest Timing Analyzer window, perform the following steps:

1. In the Quartus II software, on the Tools menu, click **TimeQuest Timing Analyzer**.

2. On the **Tasks** pane, double-click **Report DDR** which automatically runs **Update Timing Netlist**, **Create Timing Netlist**, and **Read SDC**. After a task is executed, it turns green. After the tasks are completed, the timing margin report is generated.

Figure 4–6 shows the timing margin report in the TimeQuest Timing Analyzer window after running the report timing script. The results are the same as the Quartus II software results.

**Figure 4–6. Timing Margin Report in TimeQuest Timing Analyzer**




For more information about the TimeQuest timing analyzer, refer to *The Quartus II TimeQuest Timing Analyzer* chapter in volume 3 of the *Quartus II Handbook*.
For information timing analysis, refer to the *Timing Analysis* section in volume 4 of the *External Memory Interface Handbook*.

# Verify Design on a Board

The SignalTap II Embedded Logic Analyzer shows read and write activity in the system. To verify the multiple memory interfaces design on Stratix IV E FPGA development board, run the Signal Tap Embedded Logic Analyzer to verify each memory interface separately, or create a Signal Tap file, which combines all the memory interfaces nodes in the design.

For more information about using the SignalTap II Embedded Logic Analyzer, refer to the *Design Debugging Using the SignalTap II Embedded Logic Analyzer* chapter in the *Quartus II Handbook*, *AN 323: Using SignalTap II Embedded Logic Analyzers in SOPC Builder Systems* and *AN 446: Debugging Nios II Systems with the SignalTap II Embedded Logic Analyzer*.

To add the SignalTap II Embedded Logic Analyzer to your Quartus II project, open the Altera-provided **siv_e_rldamii_qdrii_plus.stp**, or perform the following steps:

1. On the Tools menu, click **SignalTap II Logic Analyzer**.
2. Under **Signal Configuration** next to the **Clock** box, click **…** (Browse Node Finder).
3. Turn on **Include subentities** to include all sub-entities in the hierarchy.
4. Type `*afi_clk` in the **Named** box, for **Filter** select **SignalTap II: pre-synthesis** and click **List**.
5. Select **<top_level>|rldramii:master_mem_afi|afi_clk** in **Nodes Found** and click **>** to add the signal to **Selected Nodes**.
6. Click **OK**.
7. Under **Signal Configuration**, specify the following settings:
   - For **Sample depth**, select **512**.
   - For **RAM type**, select **Auto**.
   - For **Trigger flow control**, select **Sequential**.
   - For **Trigger position**, select **Center trigger position**.
   - For **Trigger conditions**, select **1**.
8. On the Edit menu, click **Add Nodes**.
9. Search for specific nodes by typing in the **Named** box, for **Filter** select **SignalTap II: pre-synthesis** and click **List**.
10. In **Nodes Found**, select the following nodes and click **>** to add to **Selected Nodes**:

- RLDRAM II memory interface instances
  - **avl_addr**
  - **avl_rdata**
  - **avl_rdata_valid**
  - **avl_read_req**
  - **avl_ready**
  - **avl_wdata**
  - **avl_write_req**
  - **rldramii_fail**
  - **rldramii_pass**
  - **rldramii_afi_cal_fail**
  - **rldramii_afi_cal_success**

- **rldramii_test_complete**
- **pnf_per_bit**
- **rdata_valid**
- **rdata_valid_reg**
- **data_out**
- **data_in**

- QDR II+ SRAM memory interface instances
  - **avl_addr**
  - **avl_read_req**
  - **avl_ready**
  - **avl_write_req**
  - **qdrii_plus_fail**
  - **qdrii_plus_pass**
  - **qdrii_plus_afi_cal_fail**
  - **qdrii_plus_afi_cal_success**
  - **qdrii_plus_test_complete**
  - **be_reg**
  - **pnf_per_bit**
  - **rdata_reg**
  - **rdata_valid_reg**
  - **data_out**

Do not add any RLDRAM II or QDR II+ SRAM interface signals to the SignalTap II Embedded Logic Analyzer. The load on these signals increases and adversely affects the timing analysis.

11. Click **OK**.
12. To reduce the SignalTap II logic size, turn off **Trigger Enable** on the following bus signals:

- RLDRAM II memory interface instances
  - **avl_addr**
  - **avl_rdata**
  - **avl_wdata**
  - **data_out**
  - **data_in**

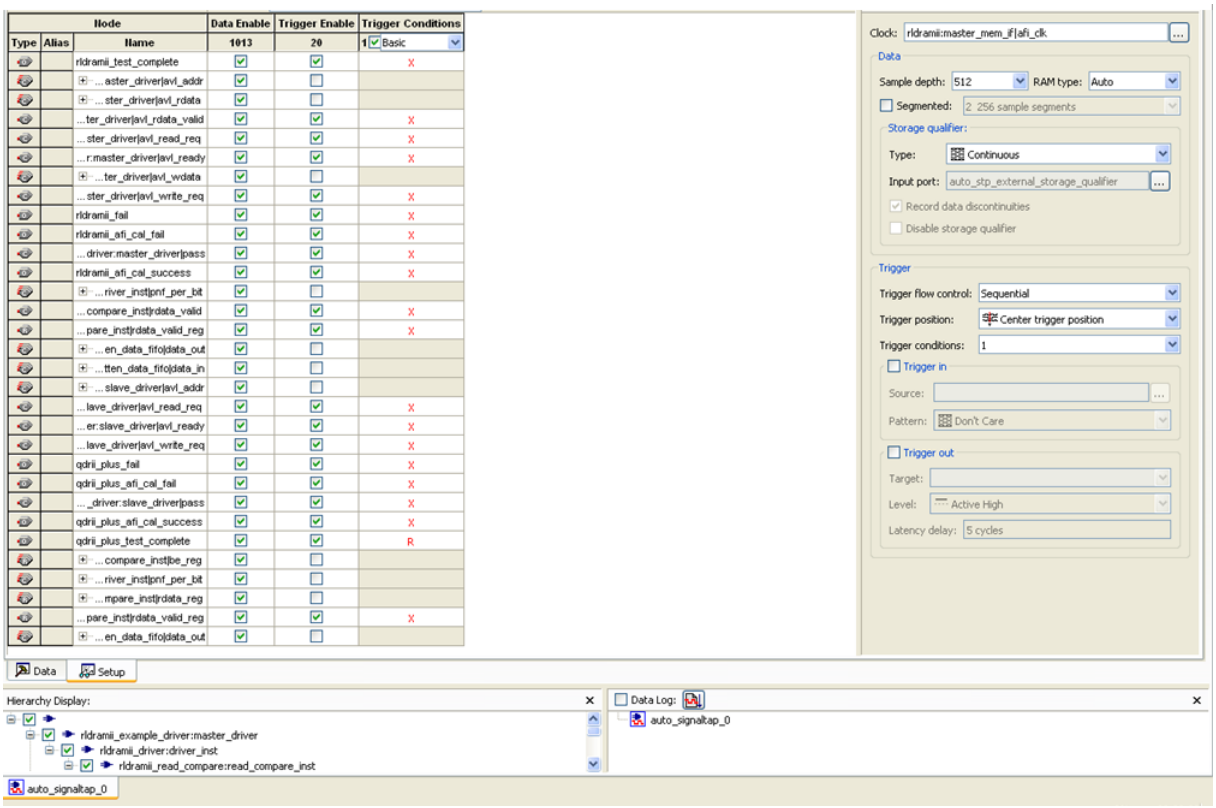
- QDR II+ SRAM memory interface instances
  - **avl_addr**
  - **data_out**
  - **pnf_per_bit**
  - **rdata_reg**
  - **be_reg**

13. Right-click **Trigger Conditions** for the `rldramii_test_complete` signal and select **Rising Edge** to verify the RLDRAM II memory interfaces functionality.

To verify the QDR II+ memory interfaces functionality, select **Don't Care** for trigger conditions for the the `rldramii_test_complete` signal, and right-click **Trigger Conditions** for the the `qdrii_plus_test_complete` signal, and select **Rising Edge**.

Figure 4–7 shows the completed SignalTap II Embedded Logic Analyzer.

**Figure 4–7. SignalTap II Embedded Logic Analyzer**




14. On the File menu, click **Save**, to save the SignalTap II **.stp** file to your project.

If you see the message **Do you want to enable SignalTap II file "stp1.stp" for the current project**?, click **Yes**.

## Compile the Project

After you add the signals to the SignalTap II Embedded Logic Analyzer, on the Processing menu, click **Start Compilation** to recompile your design.

## Verify Timing

After you have recompiled the design, ensure that the TimeQuest timing analysis passes successfully. In addition to this FPGA timing analysis, check your PCB or system SDRAM timing. To run timing analysis, run the <*variation name*>**_report_timing.tcl** script.

1. On the Tools menu, click **Tcl Scripts**.
2. Select <*variation name*>**_report_timing.tcl**.
3. Click **Run**.

## Download the Object File

To download the object file to the device, perform the following steps:

1. Connect the development board to your computer.
2. On the Tools menu, click **SignalTap II Logic Analyzer**. The SignalTap II dialog box appears.
3. Click **...** to open the **Select Program Files** dialog box.
4. Select <*your project name*>**.sof**.
5. Click **Open**.
6. To download the file, click the **Program Device** button.

## Test the Design Example in Hardware

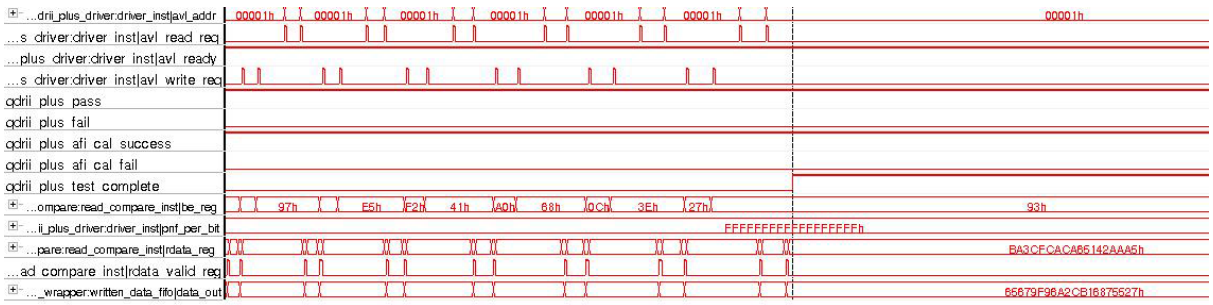
When the design example including SignalTap II successfully downloads to your development board, click **Run Analysis** to run once, or click **Autorun Analysis** to run continuously.

Figure 4–8 shows the RLDRAM II design analysis. The example driver performs self verification by comparing the written out data, `data_out` with the read back data, `rdata_reg` when the `rdata_valid_reg` signal goes high.

**Figure 4–8. SignalTap II Example RLDRAM II Design Analysis**


Figure 4–9 shows the QDR II+ SRAM design analysis.

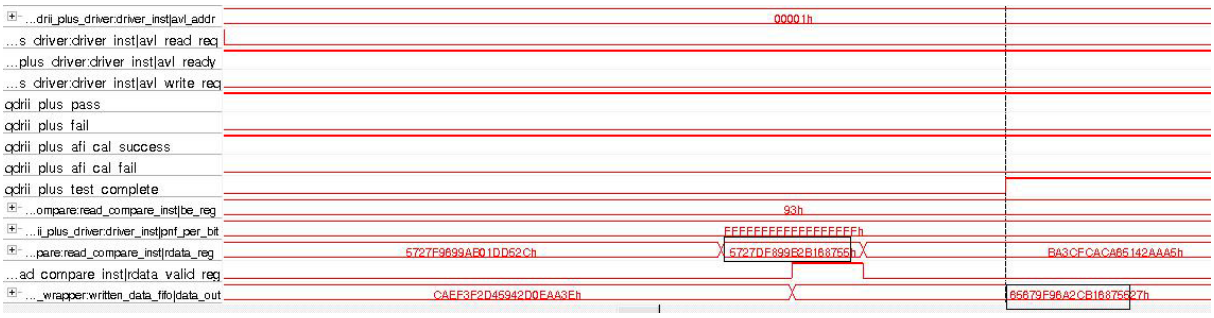
**Figure 4–9. SignalTap II Example QDR II+ SRAM Design Analysis**




The `pnf_per_bit` and `pass` signals are high, indicating the test passes in hardware.

Figure 4–10 shows the data comparison between `rdata_reg[71:0]` and `data_out[71:0]` when the `rdata_valid_reg` and `data_out[79:72]` signals go high.

**Figure 4–10. SignalTap II Example QDR II and QDR II+ SRAM Design Analysis**




The `data_out[79:72]` signal, which acts as a byte-enable signal, selects certain ×9 groups to compare the data.

This chapter provides additional information about the document and Altera.

## Document Revision History

The following table shows the revision history for this section.

| Date | Version | Changes |
|---|---|---|
| December 2010 | 2.1 | ■ Added a new chapter: DDR3 SDRAM Controller with UniPHY using Qsys.<br>■ Updated information about Stratix IV devices and OCT. |
| July 2010 | 2.0 | Updated for 10.0 release. |
| February 2010 | 1.1 | Updated for 9.1 SP1 release. |
| November 2009 | 1.0 | First published. |

## How to Contact Altera

To locate the most up-to-date information about Altera products, refer to the following table.

| Contact *(1)* | Contact Method | Address |
|---|---|---|
| Technical support | Website | www.altera.com/support |
| Technical training | Website | www.altera.com/training |
| | Email | custrain@altera.com |
| Product literature | Website | www.altera.com/literature |
| Non-technical support (General) | Email | nacomp@altera.com |
| (Software Licensing) | Email | authorization@altera.com |

**Note to Table:**

(1) You can also contact your local Altera sales office or sales representative.

## Typographic Conventions

The following table shows the typographic conventions this document uses.

| Visual Cue | Meaning |
|---|---|
| **Bold Type with Initial Capital Letters** | Indicate command names, dialog box titles, dialog box options, and other GUI labels. For example, **Save As** dialog box. For GUI elements, capitalization matches the GUI. |
| **bold type** | Indicates directory names, project names, disk drive names, file names, file name extensions, software utility names, and GUI labels. For example, **\qdesigns** directory, **D:** drive, and **chiptrip.gdf** file. |
| *Italic Type with Initial Capital Letters* | Indicate document titles. For example, *Stratix IV Design Guidelines*. |

| Visual Cue | Meaning |
|---|---|
| *italic type* | Indicates variables. For example, *n* + 1.<br>Variable names are enclosed in angle brackets (< >). For example, *<file name>* and *<project name>*.**pof** file. |
| Initial Capital Letters | Indicate keyboard keys and menu names. For example, the Delete key and the Options menu. |
| "Subheading Title" | Quotation marks indicate references to sections within a document and titles of Quartus II Help topics. For example, "Typographic Conventions." |
| `Courier type` | Indicates signal, port, register, bit, block, and primitive names. For example, `data1`, `tdi`, and `input`. The suffix `n` denotes an active-low signal. For example, `resetn`.<br>Indicates command line commands and anything that must be typed exactly as it appears. For example, `c:\qdesigns\tutorial\chiptrip.gdf`.<br>Also indicates sections of an actual file, such as a Report File, references to parts of files (for example, the AHDL keyword `SUBDESIGN`), and logic function names (for example, `TRI`). |
| ↵ | An angled arrow instructs you to press the Enter key. |
| 1., 2., 3., and<br>a., b., c., and so on | Numbered steps indicate a list of items when the sequence of the items is important, such as the steps listed in a procedure. |
| ■ ■ ■ | Bullets indicate a list of items when the sequence of the items is not important. |
|  | The hand points to information that requires special attention. |
|  | A question mark directs you to a software help system with related information. |
|  | The feet direct you to another document or website with related information. |
|  | A caution calls attention to a condition or possible situation that can damage or destroy the product or your work. |
|  | A warning calls attention to a condition or possible situation that can cause you injury. |
|  | The envelope links to the Email Subscription Management Center page of the Altera website, where you can sign up to receive update notifications for Altera documents. |