AN 328: Interfacing DDR2 SDRAM with Stratix II, Stratix II GX, and Arria GX Devices

© October 2009 AN-328-6.0

Introduction

This application note provides information about interfacing DDR2 SDRAM with Stratix® II, Stratix II GX, and Arria® GX devices. It includes details about supported modes and guidelines for designing with these devices and describes Altera's recommended design flow for implementing a DDR2 SDRAM memory interface on a Stratix II, Stratix II GX, or Arria GX FPGA. Three design example walk-throughs are provided that detail critical aspects of this flow, including:

- Instantiating the PHY to a DDR2 SDRAM component
- Setting the appropriate constraints on the PHY
- Verifying design functionality using simulation
- Timing closure using the TimeQuest timing analyzer in the Quartus® II software.

This application note also includes three design examples. Two of the design examples interface with five DDR2 SDRAM components (amounting to a 72-bit interface) available in the Stratix II GX PCI-Express (PIPE) Development Kit. One design example uses the ALTMEMPHY megafunction and the other design example uses the legacy PHY. The third design example includes a Quartus II project with three DDR2 SDRAM controllers to show you how to create multiple memory controllers with Stratix II devices.

After un-archiving the design example, ensure that you point to the corresponding IP library in your Quartus II installation directory. The IP library is located in the **<quartus_install_directory>\<*version*>\ip\ddr2_high_perf** for the ALTMEMPHY-based controller and in the **<quartus_install_directory>\<*version*>\ip\ddr_ddr2_sdram\lib** for the legacy controller.

DDR2 SDRAM is the second generation of double-data rate (DDR) SDRAM technology, with features such as lower power consumption, higher data bandwidth, enhanced signal quality, and on-die termination schemes. DDR2 SDRAM brings higher memory performance to a broad range of applications, such as PCs, embedded processor systems, image processing, storage, communications, and networking.

Stratix II and Stratix II GX devices support two modes of DDR2 SDRAM interfacing—with and without dedicated DQS phase-shift circuitry. In addition, Stratix II and Stratix II GX device families offer two different data paths or physical interfaces (PHYs)—the legacy integrated static data path and controller (referred to as the legacy controller in this document) and the ALTMEMPHY megafunction. The legacy integrated static data path and controller supports both modes of DDR2 SDRAM interfacing (with and without dedicated DQS phase-shift circuitry). However, you can only use the ALTMEMPHY megafunction with dedicated DQS phase-shift circuitry.

Table 1 lists the maximum clock frequency for DDR2 SDRAM interfaces in Stratix II and Stratix II GX devices with and without dedicated DQS phase-shift circuitry.

Table 1. DDR2 SDRAM Interface Maximum Clock Frequency Support in Stratix II and Stratix II GX Devices *(Note 1)*, *(2)*, *(3)*

Speed Grade	Frequency (MHz)			
	With Dedicated DQS Circuitry			Without Dedicated DQS Circuitry
	ALTMEMPHY			
	Half-Rate Mode	Full-Rate Mode	Legacy PHY	Legacy PHY
-3	333	267	267	200
-4	267	233	267	167
-5	233	200	233	167

Notes to Table 1:

(1) The supported operating frequencies listed here are memory interface maximums for the FPGA device family. Your design's actual achievable performance is based on design- and system-specific factors, as well as static timing analysis of the completed design.

(2) These specifications apply to both commercial and industrial devices.

(3) These specifications are applicable for both interfacing with DDR2 SDRAM modules and discrete devices.

The Arria GX device family supports only ALTMEMPHY-based memory interfaces with dedicated DQS phase-shift circuitry. Memory interfaces with the ALTMEMPHY megafunction can use either half-rate mode in which the system clock for the controller is half the frequency of the memory interface frequency or full-rate mode in which the system clock and the memory interface frequencies are the same.

Table 2 lists the maximum clock frequency for DDR2 SDRAM interfaces in Arria GX devices.

Table 2. DDR2 SDRAM Interface Maximum Clock Frequency Support in Arria GX Devices *(Note 1)*, *(2)*, *(3)*, *(4)*

Speed Grade	Frequency (MHz)	
	Half-Rate	Full-Rate
–6	233	167

Notes to Table 2:

(1) The supported operating frequencies listed here are memory interface maximums for the FPGA device family. Your design's actual achievable performance is based on design- and system-specific factors, as well as static timing analysis of the completed design.

(2) These specifications apply to both commercial and industrial devices.

(3) These specifications are applicable for both interfacing with DDR2 SDRAM modules and discrete devices.

(4) DDR2 SDRAM memory interface support in Arria GX devices is only available with ALTMEMPHY implementation.

DDR2 SDRAM Overview

DDR2 SDRAM is the second generation of the DDR SDRAM memory standard. It is a 4*n* pre-fetch architecture with two data transfers per clock cycle. The memory uses a strobe (DQS) associated with a group of data pins (DQ) for read and write operations. Both the DQ and DQS ports are bi-directional. Address ports are shared for write and read operations.

Although DDR2 SDRAM components can use the optional differential strobes (DQS and DQS#), Stratix II, Stratix II GX, and Arria GX devices do not support this functionality. These devices only use the DQS signal to read from and write to the DDR2 SDRAM component. This is because DQS and DQS# pins are not differential I/O pins in these device families.

DDR2 SDRAM write and read operations support burst lengths of four and eight. Each read and write transaction transfers either four or eight groups of data. The latency between the time the read command is clocked into memory and the time data is presented at the memory pins is called the column address strobe (CAS) latency. DDR2 SDRAM supports CAS latencies of two, three, four, and five. DDR2 SDRAM does not support half-clock latencies, unlike DDR SDRAM.

DDR2 SDRAM components use the SSTL-18 standard and can hold between 256 Mbytes to 4 Gbytes of data. Devices with capacities up to 512 Mbytes are divided into four banks and devices with capacities between one and four Gbytes are divided into eight banks. Only one row per bank can be accessed at one time. The `ACTIVE` command opens a row; the `PRECHARGE` command closes a row.

The Altera DDR2 SDRAM memory controller keeps a row open in every bank of your memory system. If you have a small DDR2 SDRAM component with four banks, the controller keeps track of the four open rows, one per bank. If you have a dual-rank DDR2 SDRAM DIMM made up of devices with eight banks each, the controller tracks 16 open rows.

DDR2 SDRAM uses a delay-locked loop (DLL) inside the device to edge-align the DQ and DQS signals with respect to the CK and CK# signals. DLL is turned on for normal operation and is turned off for debugging purposes. All timing analyses done in this application note assumes that the DLL inside the memory is on.

For more information about DDR2 SDRAM components, refer to the www.jedec.org website.

DDR2 SDRAM components have adjustable data-output drive strength; therefore, for maximum performance Altera recommends you use the highest drive strength of the memory. DDR2 SDRAM components also offer on-die termination and output driver calibration. The on-die termination has an effective resistance of either 50,75, or 150 Ω. IBIS simulation can show the effects of different drive strengths, termination resistors, and capacitive loads on your system.

Stratix II, Stratix II GX, and Arria GX Dedicated DQS Phase-Shift Circuitry

Dedicated phase-shift circuits are available at the top and bottom sides of the Stratix II, Stratix II GX, and Arria GX devices. The circuit mainly consists of a DLL that generates the required phase shift setting for the incoming DQS read signal and DQS delay chains controlled by this DLL setting.

Using dedicated phase shift circuitry allows you to capture data with the DQS sent by the memory device, which in turn results in higher performance than when not using dedicated phase shift circuitry. When using the DQS signals to capture data in the FPGA, you can use the skew between the DQS and DQ signals to analyze read capture timing. When not using the DQS signals to capture data in the FPGA, you must use the skew between the CK/CK# and DQ signals to analyze read capture margin. Skew reduces the read margin. In addition, the skew between the CK/CK# and DQ signals is higher than the skew between the DQS and DQ signals. Therefore, memory interface performance, when not using the DQS signal to capture data, is lower than the performance when using the DQS signal to capture data.

To ensure that DQS mode can perform in challenging environments, system validation has been run to pinpoint the most critical factors affecting system functionality. The factors are the following:

- drive strength
- termination scheme
- addressing scheme
- data pattern effects
- process
- voltage and temperature (PVT) effects
- memory devices

Various responses can be given by system validation, including eye quality, edge rates, voltage sag, ground bounce, and, of course, system functionality, on a pass or fail basis.

For more information about Stratix II DDR2 system validation and results, refer to the *Stratix II DDR2 System Validation Summary* white paper.

Stratix II, Stratix II GX, and Arria GX On Chip Termination (OCT)

The OCT technology is offered on Stratix II, Stratix II GX, and Arria GX devices. Table 3 summarizes the extent of OCT support for DDR2 SDRAM interface.

Table 3. On-Chip Termination Schemes

Termination Scheme	SSTL-18	FPGA Device			
		Stratix II Stratix II GX		Arria GX	
		Column	Row	Column	Row
On-Chip Series Termination without Calibration	Class I	50	50	50	50
	Class II	25	—	25	—
On-Chip Series Termination with Calibration	Class I	50	—	—	—
	Class II	25	—	—	—
On-Chip Parallel Termination with Calibration	Class I and Class II	50*(1)*	—	—	—

Note to Table 3:

(1) Non-dynamic on-chip parallel termination is only supported for input pins.

ALTMEMPHY and Legacy PHY Brief Overview

The legacy PHY uses a resynchronization clock with a static phase shift that is determined before design compilation, while the ALTMEMPHY megafunction implementation features a dynamic resynchronization clock that is calibrated for process (P) variations during initialization and tracks voltage and temperature (VT) variations.

The ALTMEMPHY megafunction is available as a stand-alone PHY for use with third-party memory controllers. The ALTMEMPHY megafunction is also instantiated by Altera's DDR and DDR2 SDRAM High Performance Controller MegaCore® function.

The legacy PHY is embedded in the DDR and DDR2 SDRAM Controller MegaCore function. Unlike the ALTMEMPHY megafunction, the legacy PHY must be extracted manually from the DDR and DDR2 SDRAM controller MegaCore function when using a third-party controller.

Use the ALTMEMPHY megafunction for all new designs to achieve high performance and optimal resynchronization phase shift. All new device families after Stratix II GX support the ALTMEMPHY megafunction and may not support legacy PHY. Use legacy PHY when you need a lower latency interface or when you are not using dedicated phase shift circuitry.

For more information about whether to use the legacy PHY or the ALTMEMPHY megafunction, refer to:

- *TB 091: External Memory Interface Options for Stratix II Devices*
- "Appendix C: Legacy PHY Architecture Description" on page 101
- *DDR and DDR2 SDRAM Controller Compiler User Guide*

DDR2 SDRAM Interfaces in HardCopy II Devices

Designs targeting the Stratix II device, up to 267 MHz, can be migrated to HardCopy® II devices. For the purpose of this application note, any Altera FPGA description referring to a Stratix II device can also apply to a HardCopy II device. HardCopy II device memory interfaces; however, have additional restrictions, such as the usage of the phase-locked loop (PLL) dedicated clock output pins for CK/CK# signals.

Restrictions for HardCopy II device memory interfaces are described in *AN 463: Using the ALTMEMPHY Megafunction with HardCopy II Structured ASICs* if you are using ALTMEMPHY, and in *AN 413: Using Legacy Integrated Static Data Path and Controller Megafunction with HardCopy II Structured ASICs* if you are using the legacy PHY.

Memory Interface Design Flow

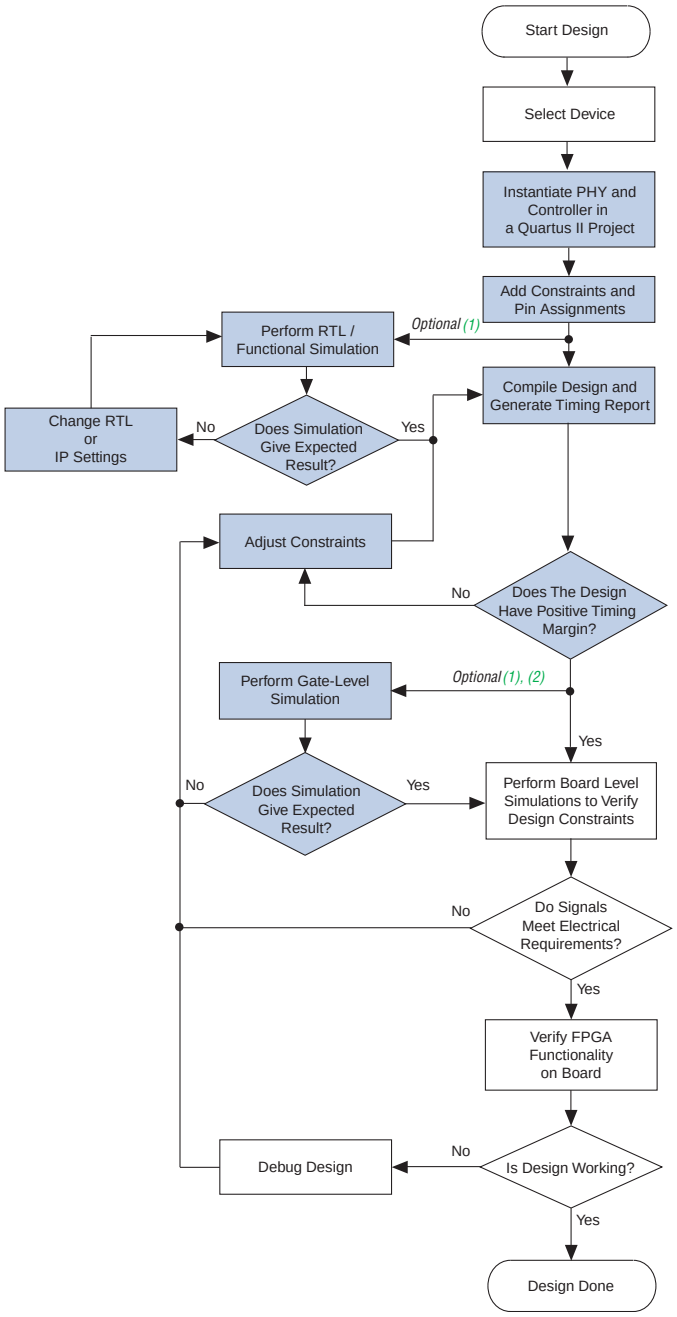
This section describes Altera's recommended flow, shown in Figure 1, for successful memory interface implementation in Stratix II, Stratix II GX, and Arria GX devices. These guidelines provide the best experience with external memory interfaces in these device families.

A detailed description of each step is available in *AN 449: Design Guidelines for Implementing External Memory Interfaces in Stratix II and Stratix II GX Devices*.

The next sections include two design examples that describe the shaded design flow steps in Figure 1.

For more information for the Quartus II version that you are using regarding possible issues with instantiation, simulation, and timing closure, refer to the *Quartus II Software Release Notes*.

Figure 1. Design Flow for Implementing External Memory Interfaces in Stratix II, Stratix II GX, and Arria GX Devices



Notes to Figure 1:

(1) Though optional, Altera recommends that you perform this step to ensure design functionality.

(2) The ALTMEMPHY megafunction does not support gate-level simulation.

Design Checklist

Table 4 contains a design checklist that you can use when implementing DDR2 SDRAM memory interfaces in Stratix II, Stratix II GX, and Arria GX devices.

For specific HardCopy II requirements if you are using the ALTMEMPHY megafunction, refer to *AN 463: Using the ALTMEMPHY Megafunction with HardCopy II Structured ASICs.*

For specific HardCopy II requirements if you are using the legacy PHY for any specific HardCopy II requirements, refer to *AN 413: Using Legacy Integrated Static Data Path and Controller Megafunction with HardCopy II Structured ASICs*.

Table 4. Checklist for Implementing DDR2 SDRAM Memory Interfaces in Stratix II, Stratix II GX, and Arria GX Devices (Part 1 of 4)

Item	Description	Yes or No
Select Device		
1	Have you selected the memory interface frequency of operation and bus width? And, have you selected the FPGA device density and package combination that you will be targeting? Ensure that the target FPGA device supports the desired clock rate and memory bus width. For detailed device resource information, refer to the device handbook chapter on external memory interface support.	
2	Have you selected your PHY implementation? Use the ALTMEMPHY megafunction whenever possible. For more information about selecting between the two PHYs, refer to *TB 091: External Memory Interface Options for Stratix II Devices*.	
Instantiate PHY and Controller		
3	Have you parameterized and instantiated the PHY and controller for your target memory interface? When instantiating multiple controllers, ensure effective sharing of device resources and appropriate constraints by referring to "Multiple Memory Controllers Walkthrough" on page 86 when using the ALTMEMPHY-based interfaces or *AN 392: Implementing Multiple Legacy DDR/DDR2 SDRAM Controller Interfaces* when using the legacy PHY.	
4	If you are using your own controller, have you connected the PHY's local signals to your driver logic and the PHY's memory interface signals to the top-level pins? Ensure that the local interface signals of the PHY are appropriately connected to your own logic. If the PHY 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.	

Table 4. Checklist for Implementing DDR2 SDRAM Memory Interfaces in Stratix II, Stratix II GX, and Arria GX Devices (Part 2 of 4)

Item	Description	Yes or No
Functional Simulation		
5	Have you simulated your design using the register transfer level (RTL) functional model? When using Altera's memory controllers, use the design example, which instantiates the PHY and controller blocks, along with the example driver/testbench module, and the memory device model. When using a custom memory controller, use the PHY functional simulation model in conjunction with your own driver logic/testbench and the memory device model.	
Timing Closure		
6	Have you added constraints to the PHY and the rest of your design? The ALTMEMPHY megafunction is constrained when you use the generated Synopsys design constraint (**.sdc**) file and **.tcl** files. The legacy PHY is constrained with an **.sdc** file generated by the DTW and **.tcl** file generated by the MegaWizard™ Plug-In Manager. Add pin assignment constraints and pin loading constraints to your design. Ensure that generic pin names used in the constraint scripts are modified to match your top-level pin names. Note that the loading-on-memory interface pins are dependent on your board topology (memory components, single DIMM, multiple DIMMs, and single rank DIMM). Add pin location constraints to all memory interface signals. Note that you must place all the address and command pins on the same side of the device for optimal performance. The legacy PHY controller makes the pin location assignment for DQ and DQS. Use board-level simulations to verify the default assignments created by the MegaWizard Plug-In Manager.	
7	Have you compiled your design and verified timing closure using all available models? Run the **Report DDR TimeQuest** task or source the **<***variation_name***>_report_timing.tcl** file to generate a custom timing report for each of your ALTMEMPHY megafunction instances. Repeat this process using all device timing models (slow and fast). Run the **dtw_timing_analysis.tcl** script for legacy PHY.	

Table 4. Checklist for Implementing DDR2 SDRAM Memory Interfaces in Stratix II, Stratix II GX, and Arria GX Devices (Part 3 of 4)

Item	Description	Yes or No
8	If there are timing violations, have you adjusted your constraints to optimize timing? Adjust the PLL clock phase shift settings or appropriate timing and location assignment to optimize margins for the various timing paths in the PHY. The ALTMEMPHY timing analysis script creates a panel that shows the timing margins in the Quartus II software compilation report. The **dtw_timing_analysis.tcl** script creates a clock phase shift recommendation panel in addition to the timing report panel in the Quartus II software compilation report.	
Gate Level Simulation		
9	Have you performed a timing simulation on your design? Gate-level simulation is not applicable to the ALTMEMPHY design.	
Board Level Considerations		
10	Have you selected the termination scheme and drive strength settings for all the memory interface signals on the memory side and FPGA side? Ensure that appropriate termination and drive strength settings are applied on all the memory interface signals and verified using board-level simulations. Stratix II, Stratix II GX, and Arria GX devices support on-chip termination. On the memory side, Altera recommends using the DDR2 SDRAM on-die termination (ODT) feature whenever possible and using external parallel termination on memory input signals that do not support the ODT feature. On the FPGA side, Altera recommends the **Series 25 Ω without Calibration OCT** setting for bi-directional signals (such as DQ and DQS) and the **Series 50 Ω without Calibration OCT** setting for unidirectional output signals to the memory (such as DM and address/command). If there are multiple loads on certain FPGA output pins (for example, when the address bus is driven to multiple memory devices on a DIMM), you may prefer to use the maximum drive strength setting over the series OCT setting. Note that when using the OCT feature on the FPGA, the programmable drive strength feature is unavailable. Use board-level simulations to pick the optimal setting for the best signal integrity.	

Table 4. Checklist for Implementing DDR2 SDRAM Memory Interfaces in Stratix II, Stratix II GX, and Arria GX Devices (Part 4 of 4)

Item	Description	Yes or No
11	Have you performed board-level simulations to ensure electrical and timing margins for your memory interface? Ensure you have a sufficient eye opening using simulations. Be sure to use the latest FPGA and memory IBIS models, board trace characteristics, drive strength, and termination settings in your simulation. You must use any timing uncertainties at the board level that you calculate using such simulations to adjust the input timing constraints to ensure the accuracy of the Quartus II timing margin reports.	
System Verification		
12	Have you verified the functionality of your memory interface in system? You can use the SignalTap® Logic Analyzer to verify the interface signal behavior.	

Design Example Walkthrough for a 333-MHz DDR2 SDRAM Interface Using the ALTMEMPHY Megafunction

This walkthrough describes the steps necessary to create, constrain, and verify operation of a 72-bit wide, 333-MHz/667-Mbps DDR2 SDRAM memory interface targeted for the Stratix II GX PCI Express (PIPE) Development Board. This design example (an already completed version is available for download with this application note) uses the ALTMEMPHY megafunction-based DDR2 SDRAM High Performance Controller MegaCore to interface memory on the Stratix II GX PCI Express (PIPE) Development Board. This example walkthrough uses the MegaWizard Plug-In Manager to instantiate the ALTMEMPHY megafunction. In addition, you can also use SOPC Builder to instantiate the ALTMEMPHY megafunction.

For more information about the ALTMEMPHY instantiation with SOPC Builder, refer to the *DDR and DDR2 SDRAM High-Performance Controller and ALTMEMPHY User Guide*.

If you are using the legacy PHY, go to "Design Example Walkthrough for 267-MHz DDR2 SDRAM Interface Using the Legacy PHY" on page 49.

The ALTMEMPHY megafunction is a memory interface PHY that enables speeds of up to 333 MHz on Stratix II and Stratix II GX devices, 267 MHz on HardCopy II devices, and 200 MHz on Arria GX devices. This PHY uses dedicated DQS phase-shift circuitry for capturing data from memory and a dynamic clock calibration scheme to resynchronize memory read data to the system clock domain. This auto-calibrated solution tracks the changes in voltage and temperature, and simplifies timing closure and clock phase shift selection. The ALTMEMPHY megafunction is used to implement the datapath in the DDR2 SDRAM High Performance Controller MegaCore.

For more information about the PHY and memory controller, refer to the *DDR and DDR2 SDRAM High-Performance Controller and ALTMEMPHY User Guide*.

The Stratix II GX edition of the Altera's PCI Express Development Kit delivers a complete PCI Express (PIPE)-based development platform. This PCI Express (PIPE) solution, interoperable with industry-standard PCI Express (PIPE) platforms, facilitates the development of custom PCI Express (PIPE) applications.

For more information about PCI Express (PIPE) development, refer to the *Stratix II GX PCI Express Development Kit* web page.

This design example walks through the memory interface design flow steps, shown in Figure 1 on page 7.

For more information about the memory interface design flow, refer to *AN 449: Design Guidelines for Implementing External Memory Interfaces in Stratix II and Stratix II GX Devices*.

The design example was created using the Quartus II software version 9.0 and MegaCore IP Library software version 9.0.

While this example focuses on the Stratix II GX device family, the information is also applicable to the ALTMEMPHY-based memory interface designs targeting Stratix II, Arria GX, and HardCopy II devices. However, pay attention to the restrictions for HardCopy II devices described in *AN 463: Using ALTMEMPHY Megafunction with HardCopy II Structured ASICs.*

Step 1: Select Device

This design example uses the EP2SGX90FF1508C3 device that comes with the Stratix II GX PCI Express (PIPE) Development Board. The board is also equipped with five 333-MHz-capable DDR2 SDRAM components—four ×16 devices with part number—MT47H32M16CC-3 and one ×8 device with part number MT47H64M8CB-3.

For more information about all the features of the board, refer to the *Stratix II GX PCI Express Development Board Reference Manual*.

The design example uses the five DDR2 memory devices to create a 72-bit interface running at 333 MHz using the ALTMEMPHY-based DDR2 SDRAM High Performance Controller MegaCore function. This is considered a width-expansion interface because multiple memory devices are used to create one wide interface controlled by a single memory controller.

The maximum number of interfaces you can implement on any given device is limited by resource availability (the number of DQ groups of a desired width, user I/O pins, PLLs, DLLs, clocks, and FPGA core resources). For example, the Stratix II GX EP2SGX90FF1508C3 device supports nine ×8 DQ groups each on the top and bottom sides of the device. The DLL located on the top (or bottom) can be shared among nine DQ groups on that side of the device as long as they are implementing memory interfaces running at the same frequency. The example 72-bit interface uses

all nine of the groups available on the bottom side (I/O banks 7 and 8). You can use the remaining nine DQ groups located on the top side of the device for other memory interfaces. The PHY also uses one enhanced PLL to generate the six clock signals required for each memory interface. The remaining three enhanced PLLs are available for use by other memory interfaces and user logic.

To determine the number DQ groups of each width supported by the FPGA, refer to the *External Memory Interfaces* chapter of the appropriate device handbook (Stratix II, Stratix II GX, or Arria GX).

Stratix II devices offer DQ/DQS pins on all sides of the device; however, the VIO banks support DLL-based higher performance interfaces. For high-performance interfaces, use all DQ groups on one VIO side of the device. When all DQ groups on that side are used up, use the DQ groups on the opposite VIO side. Note that the DQS/DLL signals may not be shared between top and bottom. Therefore, each VIO side needs to have its own DLL instantiation.

For interfaces running at lower speeds, a PLL-based implementation is supported on all Stratix II I/O banks. The PLL-based implementation is interfaced by using DQ groups on one side of the device. When your interface width requirements exceed the number of groups supported on that side, allocate DQ groups on one of the remaining banks. Note that PLLs used for read capture must be placed on the same side as the DQ input pins for best source-synchronous compensation and timing margins.

Expanding your memory interfaces for width or depth with the same memory controller (shared address and command bus) is supported natively by the MegaWizard Plug-In Manager. Creating multiple memory controllers with independent memory transactions (independent address and command buses) requires you to create a unique megafunction variation for each interface. Sharing device resources between multiple memory interfaces may require RTL modifications.

For detailed recommendations, refer to "Multiple Memory Controllers Walkthrough" on page 86.

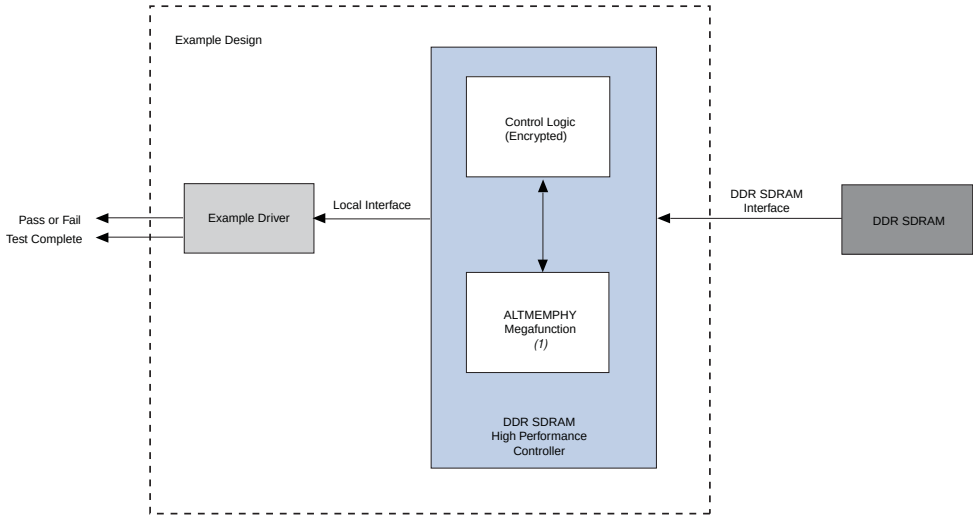
Both the MT47H32M16CC-3 and MT47H64M8CB-3 devices have the same timing specifications and share the same data sheet. If you are using devices with different timing specifications, choose the worst specifications.

Step 2: Instantiate the PHY and Controller in a Quartus II Project

Begin your memory interface design by instantiating the PHY and controller modules. Figure 2 shows a system-level diagram including the top-level design example that the DDR2 SDRAM High Performance Controller MegaCore function creates for you.

The design example is a fully functional design that can be simulated, synthesized, and used in hardware. It contains the PHY, controller, and an example driver. 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.

Figure 2. System-Level Diagram of DDR2 SDRAM Interface



Note to Figure 2:

(1) The PLL and DLL modules are included in the ALTMEMPHY megafunction.

To instantiate the PHY and controller, follow these steps:

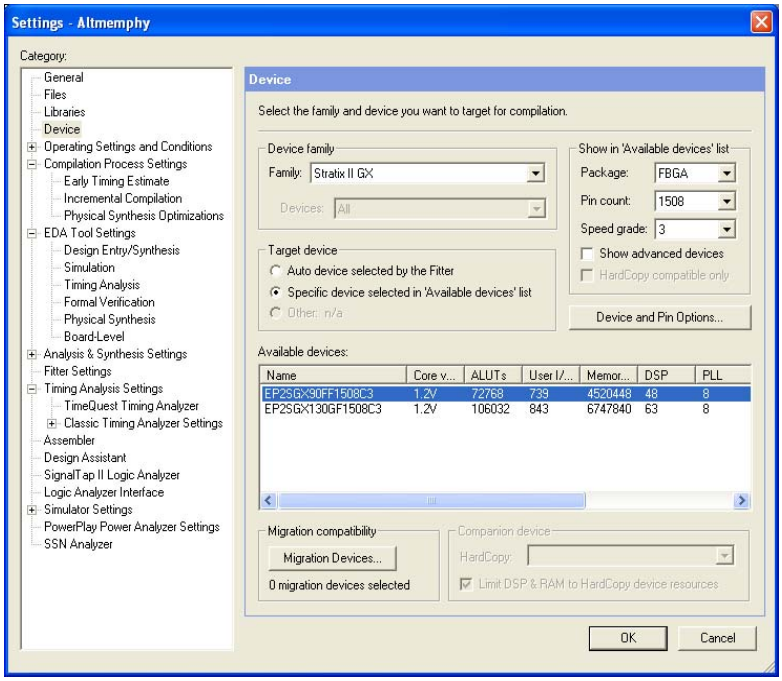
1. There are two options for step 1:
   a. Create a new Quartus II project or open an existing project where you would like to implement the DDR2 SDRAM memory interface. When creating a new project, specify the target FPGA device on the **New Project Wizard: Family and Device Settings [page 3 of 5]**. Set **Stratix II GX** as the **Family** and choose the **EP2SGX90FF1508C3** device from the **Available devices** list.

      or

   b. When using an existing project, set the target FPGA device by going to the Assignments menu and selecting **Device...**. In the dialog box, set **Stratix II GX** as the **Family**, and choose the **EP2SGX90FF1508C3** device from the **Available devices** list, as shown in Figure 3. Device listings displayed are filtered by the fastest speed grade and 1508-pin FPGA package. The Quartus II project name for this design example is **Altmemphy.**

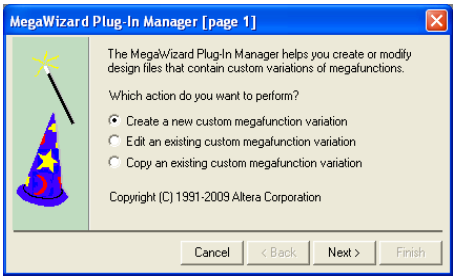
For step-by-step instructions for creating a Quartus II software project, refer to the Tutorial in the *Quartus II Online Help*.

Figure 3. Select the Target FPGA Device in the Quartus II Software



2. In the Quartus II software, on the Tools menu, point to **MegaWizard Plug-In Manager**, select **Create a new custom megafunction variation**, and click **Next** (Figure 4). **MegaWizard Plug-In Manager [page 2a]** displays (Figure 5).

Figure 4. Launch the MegaWizard Plug-In Manager

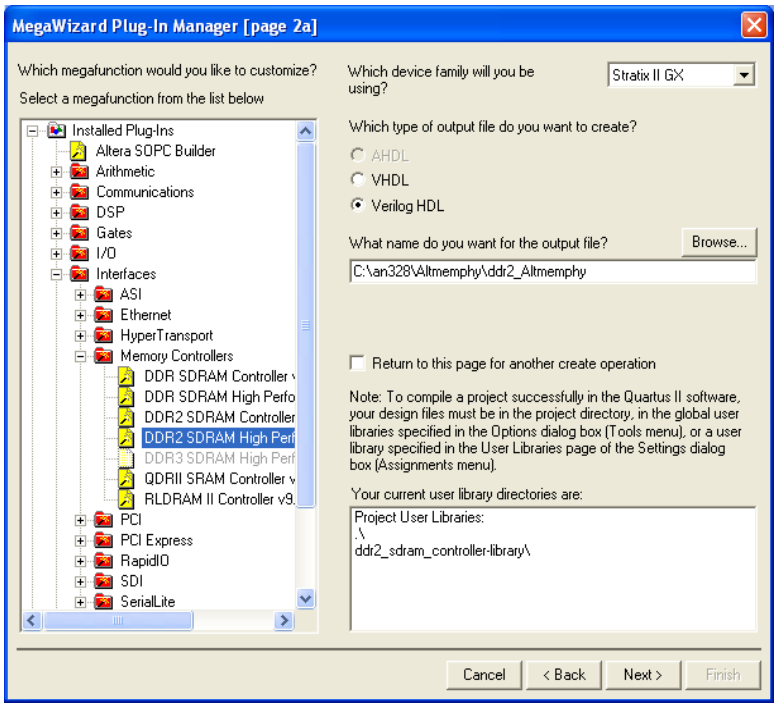


3. Under **Select a megafunction from the list below**, click the "+" icon to expand **Interfaces.** Click the "+" icon to expand **Memory Controllers** and select the **DDR2 SDRAM High Performance Controller v9.0** megafunction, as shown in Figure 5.

If **Memory Controllers** is not listed under the **Interfaces** directory, ensure that you installed the IP Library and specified the installation directory location as a **User Library** if the default location (**c:\altera**<*version*>**\ip**) was not used.

If the controller megafunction is listed but grayed out, ensure that the correct device family is selected in the drop-down menu located in the top right-hand corner of the dialog box **MegaWizard Plug-In Manager [page 2a]**.

Figure 5. Select the MegaCore Function



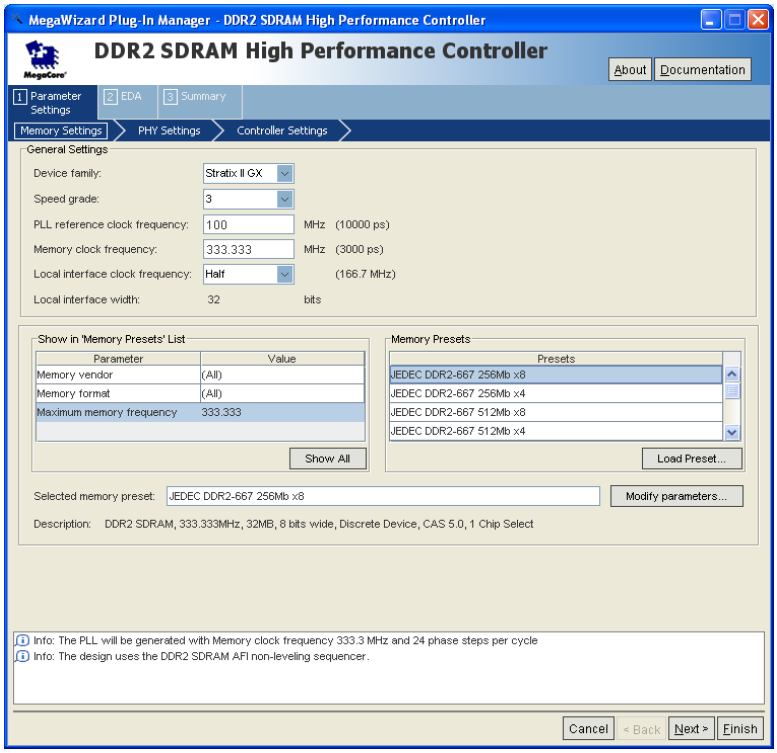
4. Select the type of output files you want the MegaWizard Plug-in Manager to create. For this design example, select **Verilog HDL**.
5. Specify the output file name for this megafunction variation. The MegaWizard Plug-In Manager shows the project path that you specified when creating the project. Append a variation name for the MegaCore function output files **<***project path***>\<***variation name***>**.

   Enter `ddr2_Altmemphy` as the variation name for this design example.

   The variation name must be different from the project name, which is **Altmemphy**.

6. Click **Next.**
   This launches the **Memory Settings** panel, located under the **Parameter Settings** tab in the MegaWizard Plug-in Manager (Figure 6). In the **Memory Settings** panel, you can parameterize the DDR2 SDRAM High Performance Controller MegaCore function.
7. In the **Parameter Settings** tab of the MegaWizard Plug-In Manager, select the **Memory Settings** tab, as shown in Figure 6. In the **Parameter Settings** tab of the MegaWizard Plug-In Manager, there are three categories of parameters that you can set—**Memory Settings**, **PHY Settings**, and **Controller Settings**.

Figure 6. Controller MegaWizard—Memory Settings



8. Verify that the target **Device family** option is set to **Stratix II GX**.
9. Select the **Speed Grade** option for the target device. Because the PCI Express (PIPE) Development Board includes a -3 speed grade FPGA device, select **3** from the drop-down list. The -3 speed grade supports the 333-MHz DDR2 SDRAM memory interface clock rate you intend to implement (based on the specifications listed in Table 1 on page 1 and the *Stratix II GX Device Handbook*).
10. Specify the **PLL reference clock frequency** option as **100 MHz**. The PCI Express (PIPE) Development Board includes a 100-MHz oscillator that feeds the clock input pins on the bottom I/O banks of the Stratix II GX device.
11. Specify the **Memory clock frequency** option as **333.333 MHz**. Be sure to specify three decimal digits for non-integer frequencies to ensure that proper timing constraints and PLL settings are generated. Verify that the period reported in parentheses by the MegaWizard Plug-In Manager is accurate.

    After you specify the PLL reference and memory clock frequencies, the MegaWizard Plug-In Manager determines if valid PLL settings exists to perform the necessary frequency synthesis. When successful, an **Info** message is generated in the bottom pane of the MegaWizard Plug-In Manager (Figure 6).

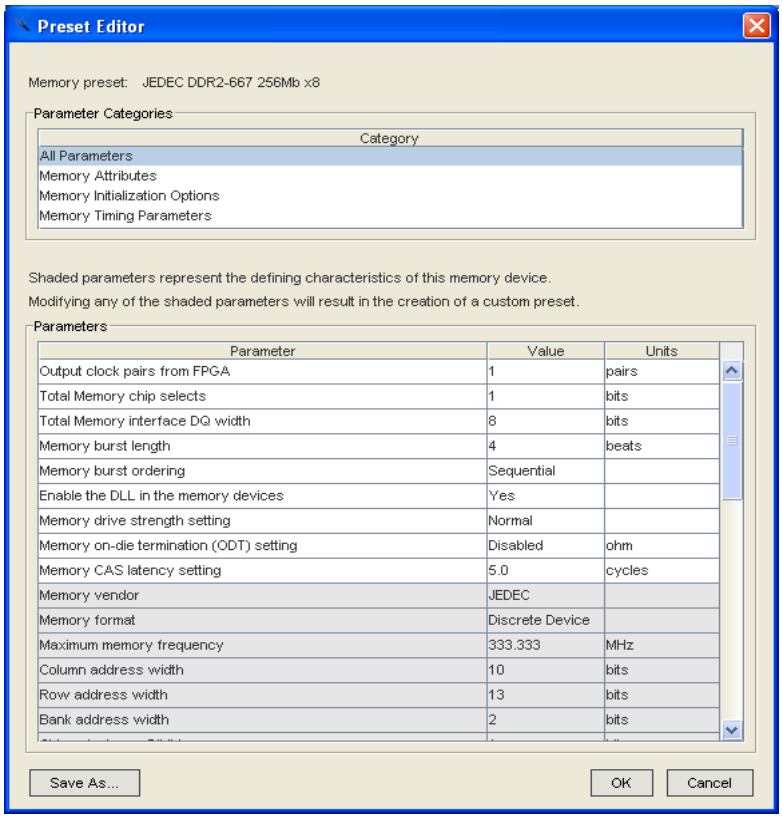
    For example, in this design example, the MegaWizard Plug-In Manager reports "Info: The PLL is generated with Memory clock frequency 333.3 MHz and 24 phase steps per cycle." The "24 phase steps per cycle" refers to the resolution of the PLL phase shift stepping feature available for the calibration block to place the resynchronization clock in the data valid window.

12. Select the **Local interface clock frequency** option. You can choose between **Half** and **Full**, which refers to the relationship between the clock rate at the local side when compared with the memory side. In half-rate mode, the local side operates at half the memory clock frequency. Because full-rate mode does not support the design example's target memory clock rate of 333.333 MHz, select **Half**. The resulting local interface frequency is 166.667 MHz.
13. To specify external memory device specifications, select a **Memory Preset** option from the list of presets that matches your memory. You can use the filters provided in a list to the left of the panel to sort through the list of memory presets.

    Select **333.333** from the **Maximum memory frequency** filter. Only memory presets that meet that clock rate are now displayed. Because a preset for the MT47H32M16CC-3 or MT47H64M8CB-3 device is not available, select a preset that is most similar. Use this as a base preset and modify as necessary.
14. Select the **JEDEC DDR2-667 256-Mb ×8** preset. Select **Modify parameters...,** this launches the **Preset Editor,** as shown in Figure 7.

    The **All Parameters** category is selected by default and the current list of settings are displayed in the **Parameters** section of the panel. The cells with white backgrounds are programmable features supported by the memory. The cells with gray background are defining characteristics of the memory device. Changing any of these settings creates a custom variation of the preset that you can use in current and future designs.

Figure 7. Preset Editor—JEDEC DDR2-667 256Mb ×8



15. Modify the parameters listed in the **Preset Editor** dialog box to match the specifications from the Micron MT47H32M16CC-3 and MT47H64M8CB-3 datasheet.

    a. Select the **Memory Attributes** category and make the modifications listed in Table 5.

Table 5. Memory Attributes Changes for the Example Design

Parameter	Value	Description
Memory vendor	**Micron**	—
Output clock pairs from FPGA	**3**	The PCI Express Development Board shares one pair of CK/CK# clock outputs for every two ×16 DDR2 SDRAM components and drives one pair of CK/CK# clock outputs to the ×8 DDR2 SDRAM component, so the 72-bit interface needs three pairs of CK/CK# outputs.
Total Memory interface DQ width	**72**	To implement a 72-bit DDR2 interface.
Maximum memory frequency for CAS latency 3.0	**200.000 MHz**	—
Maximum memory frequency for CAS latency 4.0	**266.667 MHz**	—

    b. Select the **Memory Initialization Options** category, and verify that the **Memory on-die termination (ODT)** setting is set to **Disabled** because the Stratix II GX PCI Express (PIPE) Board features external termination on all memory interface signals.

    c. Select the **Memory Timing Parameters** category and make the modifications based on the Micron data sheet. See Table 6.

Table 6. Memory Timing Parameters Changes for the Example Design

Parameter	Value
tDS	**345 ps**
tDH	**285 ps**
tIS	**200 ps**
tIH	**275 ps**
tRAS	**40 ns**
tREFI	**7.8 us**
tRFC	**105 ns**

The Stratix II GX device only supports single-ended.

DQS strobes and has typical edge rates of approximately 1 V/ns. The de-rated timing requirements for this edge rate and strobe type are 345 ps setup time and 285 ps hold time, as specified in the Micron data sheet. You must verify these typical edge rates for each system using board-level simulations. Similarly, the derated address and command specifications based on the FPGA output edge rate is 200 ps for setup and 275 ps for hold time.

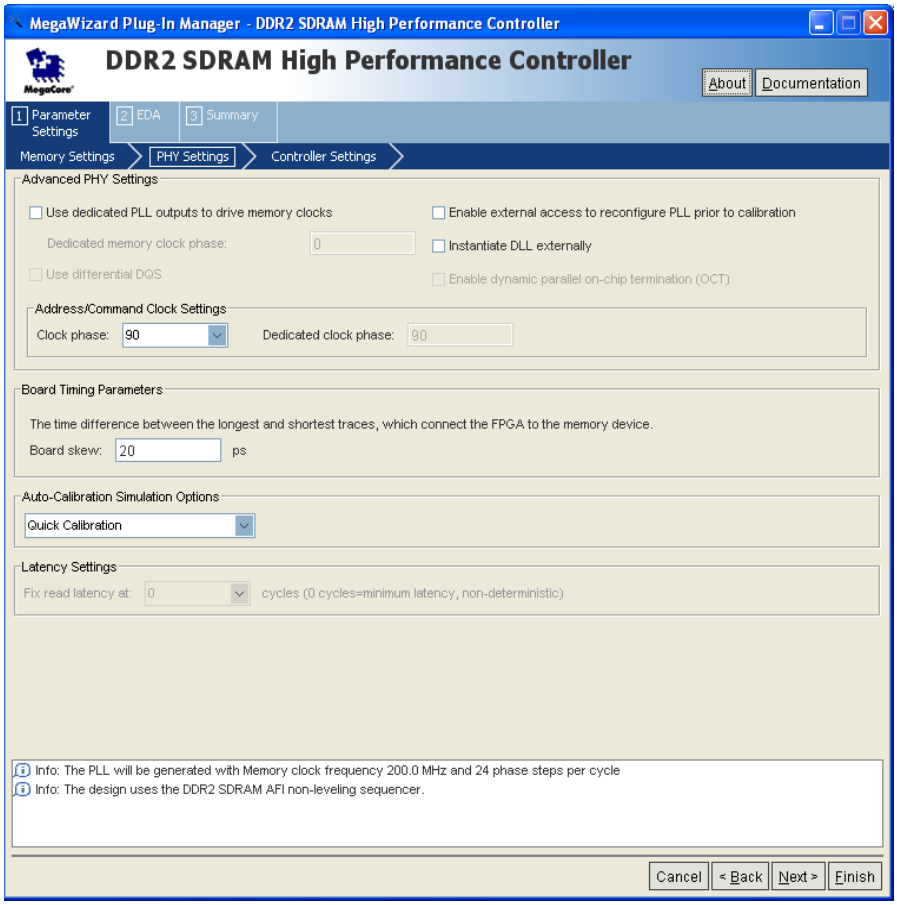
d. Click **Save As**... and save this custom memory preset as **Custom-PCI_SIIGX (JEDEC_DDR2-667_256Mb×8)**.

The presets are stored in the MegaCore **\lib** directory by default (for example, **<***quartus_installation_directory***>\ip\ddr2_high_perf\lib**) and are available for use in future projects. If you save your custom memory preset in a different directory, you can use the **Load Preset...** feature in the **Memory Settings** panel to target that memory device (Figure 6 on page 17).

e. Click **OK** to return to the **Memory Settings** panel.

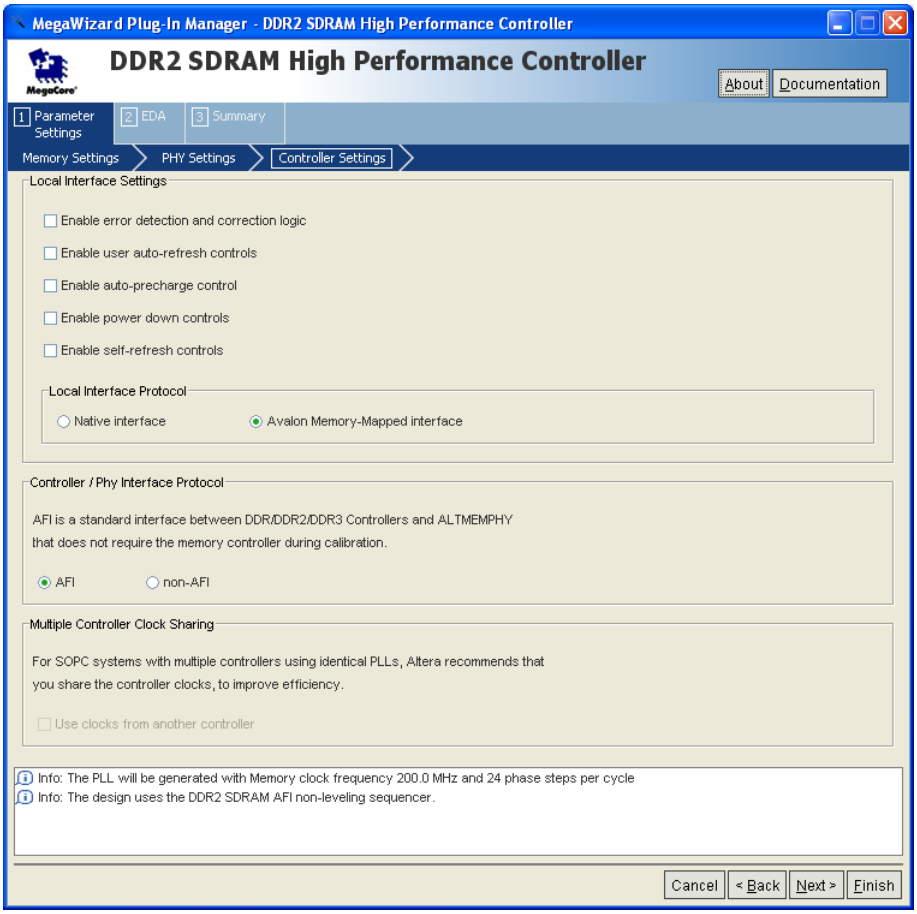
16. You have now specified all the memory settings. Click **Next** to specify the PHY settings (Figure 8 on page 20).

Figure 8. Controller MegaWizard— PHY Settings



17. In the **Advanced PHY Settings** section of the **PHY Settings** panel, you can enable the following advanced features of the PHY.
    a. For designs targeting HardCopy II devices, enable the **Use dedicated PLL outputs to drive memory clocks** and **Enable external access to reconfigure PLL prior to calibration** options. These options are not enabled for the Stratix II GX design example presented in this application note.
    b. For designs sharing DLLs across multiple memory interfaces, enable the **Instantiate DLL externally** option. This option is not enabled for the single-memory interface design example.
    c. Observe the **Clock phase** setting used to generate the address and command signals output to the memory. The choices available for Stratix II, Stratix II GX, and Arria GX devices are **0, 90, 180,** and **270 degrees of offset from the full-rate clock** (`mem_clk_2x`) used to generate the CK/CK# signals. These clock phases are generated using the rising and falling edges of the `mem_clk_2x` and `write_clk_2x` clocks of the ALTMEMPHY megafunction. You can alter the default clock phase of **90** degrees selection to optimize timing margins for the address/command timing path.
18. Specify the **Board skew** (in picoseconds) across all memory interface signals in your design, in the **Board Timing Parameters** region of the **PHY Settings** panel. 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.
19. In the **Auto-Calibration Simulation Options** of the **PHY Settings** panel, the recommended setting is **Quick Calibration**. This setting is to reduce simulation time and only affects functional simulation.
20. You have now specified all the **PHY settings**. Click **Next** to specify the controller settings (Figure 9 on page 22).

Figure 9. Controller MegaWizard—Controller Settings



21. In the **Controller Settings** panel, specify your preference in the **Local Interface Settings**.

The design example uses the default options; therefore, no changes are required on this panel.

a. Select the **Enable error detection and correction logic** option to enable error correction coding (ECC).

b. Turn on **Enable user auto-refresh controls** option to optimize latency by controlling when the controller issues refreshes to the memory.

c. Turn on **Enable auto-precharge control** to allow you access to the same bank quickly. This option is useful for applications that require fast random access.

d. Turn on **Enable power down controls** to allow you to place the external memory device in power-down mode.

e. Turn on **Enable self-refresh controls** to allow you to place the external memory device in self-refresh mode.

f. Select **Local Interface Protocol** for the memory interface. This allows you to easily connect to other Avalon® Memory-Mapped peripherals.

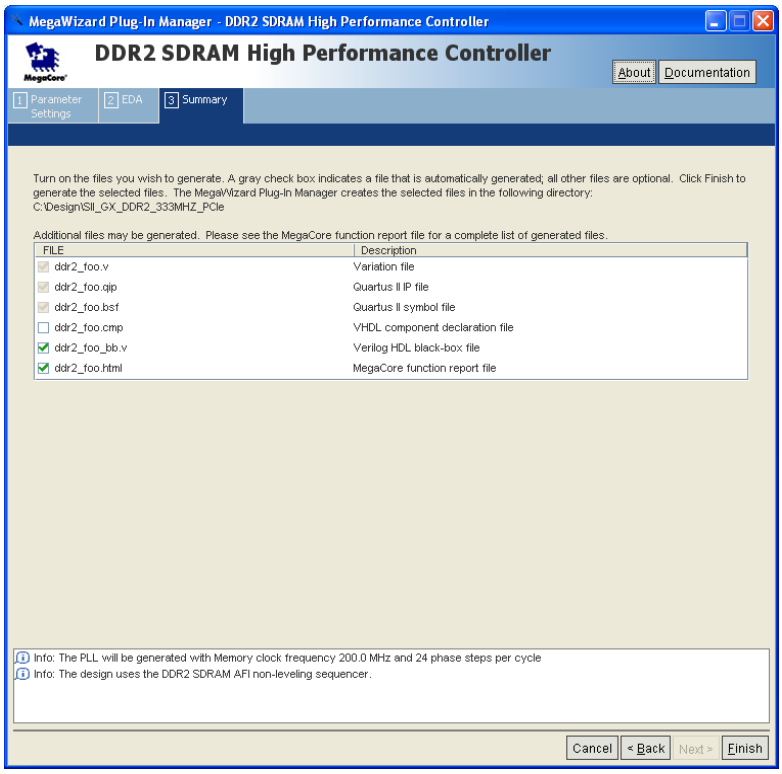
For more information about the Avalon Memory-Mapped interface, refer to the *Avalon Memory-Mapped Interface Specification.*

22. In the **Controller/PHY Interface Protocol**, the recommended setting is **AFI**. This interface between DDR/DDR2/DDR3 Controllers and the ALTMEMPHY megafunction does not require the memory controller during calibration. However, no AFI support is available for Arria GX devices.

For more information about the AFI, refer to the *DDR and DDR2 SDRAM High-Performance Controller and ALTMEMPHY User Guide*.

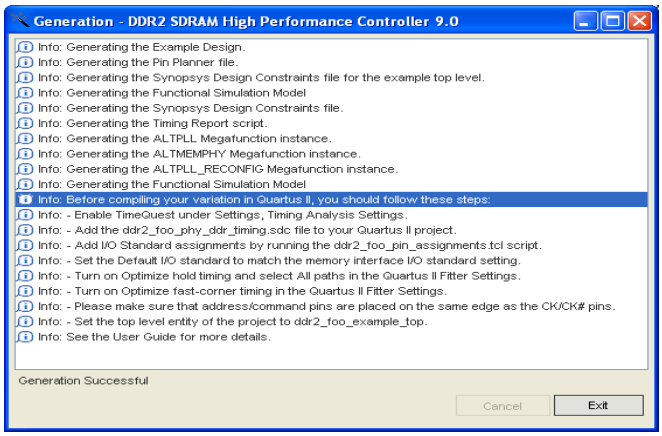
23. You have now specified the **Controller Settings**. Click **Next**.
24. In the **EDA** tab of the MegaWizard Plug-In Manager, enable the **Generate Simulation Model** option. Choose between the **Verilog HDL** or **VHDL** language options and click **Next**.
25. You have now fully parameterized the PHY and memory controller. The **Summary** tab, shown in Figure 10 on page 23, allows you to select optional files that the MegaWizard Plug-in Manager can generate for you. Click **Finish** to generate the megafunction variation.

Figure 10. Controller MegaWizard—Summary Page



The MegaWizard Plug-in Manager generates all the files necessary to constrain, compile, simulate, and timing analyze your memory interface design. Figure 11 shows the messages printed by the MegaWizard Plug-in Manager while generating your megafunction variation. This dialog box also provides recommendations about applying constraints on your design. These messages are described in "Step 3: Add Constraints and Pin Assignments" on page 25.

Figure 11. MegaWizard Messages and Constraint Recommendations



You have now successfully instantiated the PHY and controller in your design.

Table 7 is a partial list of the files generated by the DDR2 SDRAM high performance controller MegaWizard and content descriptions.

Table 7. Files Generated by the Controller MegaWizard

File Name	Description
<*variation_name*>**.v(hd)**	MegaCore variation file.
<*variation_name*>**_example_top.v(hd)**	Example top-level design file that can be simulated, synthesized, and used in hardware. Instantiates the PHY, controller, and a driver.
<*variation_name*>**_example_driver.v(hd)**	Example driver (self-test module) that issues reads and writes to the controller to check functionality.
<*variation_name*>**_phy_ddr_timing.sdc**	Sets timing constraints for the ALTMEMPHY megafunction instance.
<*variation_name*>**_pin_assignments.tcl**	Adds I/O standard settings for all memory interface pins, and adds output enable group assignments to ensure VREF rules are met when the design contains input/bi-directional pins.
<*variation_name*>**_phy_report_timing.tcl**	Generates a detailed timing report for all timing paths in the ALTMEMPHY instance.
<*variation_name*>**_auk_ddr_hp_controller_wrapper.vo or .vho**	Verilog HDL or VHDL functional simulation model of the memory controller.

Table 7. Files Generated by the Controller MegaWizard

File Name	Description
<*variation_name*>**_phy_alt_mem_phy_seq_wrapper.vo or .vho**	Verilog HDL or VHDL functional simulation model of the memory interface PHY.
testbench\<*variation_name*>**_example_top_tb.v (hd)**	Verilog HDL or VHDL testbench for functional simulation with memory controller, PHY, and memory model.

For the full list of the files generated by the DDR2 SDRAM High Performance Controller MegaWizard Plug-In Manager, refer to the *DDR and DDR2 SDRAM High-Performance Controller and ALTMEMPHY User Guide*.

Step 3: Add Constraints and Pin Assignments

All memory interface designs require timing constraints and I/O assignments (including I/O standard, output pin loading, termination, drive strength, and pin location assignments). Most of these constraints are generated for you by the MegaWizard Plug-In Manager. The MegaWizard **Generation** dialog box (Figure 11 on page 24), provides instructions on how to apply these constraints to your design. Apply these constraints to the design before compilation.

Set the Fitter Effort

To set the fitter effort to **Standard Fit**, follow these steps:

1. In the Quartus II software, on the Assignments menu, click **Settings**.
2. On the left side of the **Settings** dialog box, under the **Category** list, click **Fitter Settings**. The **Fitter Settings** panel appears.
3. In the **Fitter Settings** panel, turn on the **Optimize hold timing** and **Optimize multi-corner timing** options. On the pull-down menu next to the **Optimize hold timing** option you have just enabled, select **All paths**.

   For ALTMEMPHY MegaWizard Plug-In Manager designs targeting memory clock frequencies of 267 MHz and above, select the **Standard Fit** option under the **Fitter effort** section of the **Fitter Settings** panel. This option optimizes placement of the resynchronization and postamble registers in such designs.

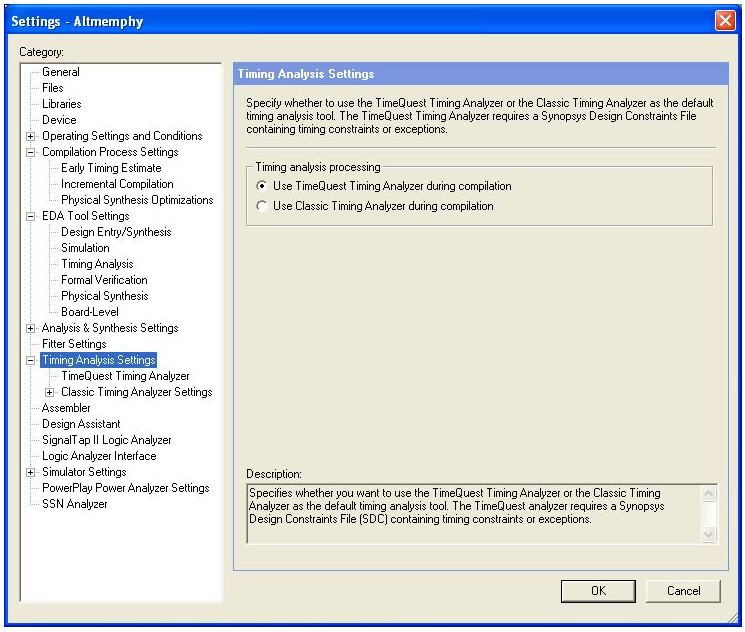
4. Click **OK**.

Add the Timing Constraints

To add timing constraints, follow these steps:

1. On the Assignments menu, click **Settings**. The **Settings** dialog box appears.
2. In the **Settings** dialog box, under the **Category** list, click **Timing Analysis Settings**.
3. In the **Timing Analysis Settings** panel, select **Use TimeQuest Timing Analyzer during compilation,** as shown in Figure 12.

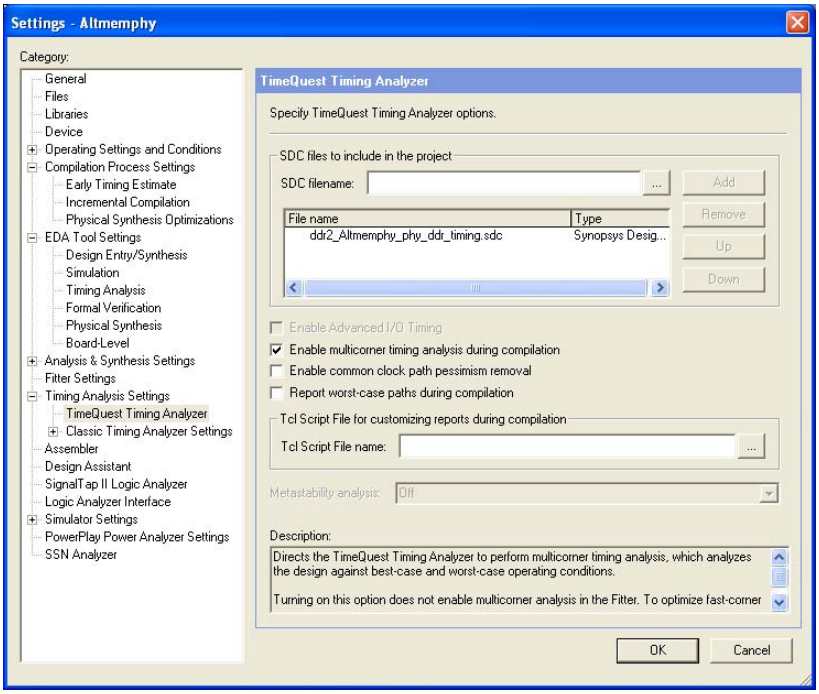
Figure 12. Enable TimeQuest Timing Analyzer



4. Under the list **Category**, click the "**+**" icon next to **Timing Analysis Settings** and select **Time Quest Timing Analyzer**.
5. Add the Synopsys design constraints (**.sdc)** file, <*variation_name*>**_phy_ddr_timing.sdc**, to your project. This file contains all the timing constraints for the PHY. Browse to the **.sdc** file, select it and click **Add**. The file name appears in the **.sdc** file list.
6. Turn on the **Enable multicorner timing analysis during compilation** option. This enables the TimeQuest timing analyzer to perform multicorner timing analysis, which analyzes the design against best-case and worst-case operating conditions.
7. Click **OK** (Figure 13 on page 27).

The **.sdc** timing constraints file is not added to the Quartus II software project automatically. You must manually include it in the project during compilation for successful timing closure and operation.

Figure 13. Add Timing Constraints (SDC)



Set the Top-Level Entity

To set the top-level entity for the example project follow these steps:

1. On the File menu, click **Open**.
2. Browse to **<***variation_name***>_example_top.v(hd)** and click **Open**.
3. On the Project menu, click **Set as Top-Level Entity**.

   If you are not using the design example, but instead are implementing the DDR2 SDRAM interface in your own top-level Quartus II software project, instantiate the **<***variation_name***>.v(hd)** module in your design and continue with the design flow.

Add the I/O Assignments

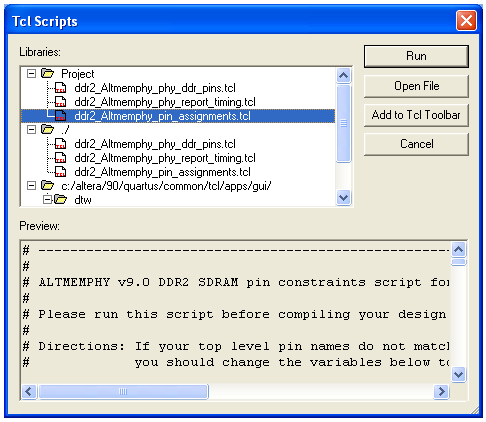
The pin assignment script, <*variation_name*>**_pin_assignments.tcl**, sets up the I/O standards for the DDR2 SDRAM interface. To add the I/O assignment, follow these steps:

1. The **I/O standard** and **output enable group** assignments are specified in the **<***variation_name***>_pin_assignments.tcl** file.

   If your design uses the default top-level pin names for memory interface signals as in this walkthrough design example (names that start with the "`mem_`" prefix, for example, `mem_dq` and `mem_dqs`), select **Tcl Scripts...** from the Tools menu.

2. In the **Tcl Scripts** dialog box, select **<***variation_name***>pin_assignments.tcl.**
3. Click **Run,** as shown in Figure 14.

Figure 14. Add I/O Assignments



4. On the Processing menu, point to **Start** and click **Start Analysis & Synthesis**. This step ensures your design files are in order and creates a list of nodes that can be accessed to create design constraints.
5. If you are not using the default pin names and want to add a unique prefix to memory interface pin names (this is recommended when implementing multiple controllers on the same device), use the **Pin Planner** file generated by the MegaWizard Plug-In Manager to import pin assignments.
   a. In the Quartus II software, on the Assignments menu, click **Pin Planner**. In the **Pin Planner**, edit your top-level design to add a prefix to all DDR2 SDRAM interface signal names. For example, change mem_addr to **core1_mem_addr**.
   b. On the Assignments menu, click **Pins**. Right-click in the page and click **Create/Import Megafunction**. Select **Import an existing custom megafunction** and navigate to <*variation_name*>**.ppf**.
   c. Type the prefix that you added to your top-level DDR2 SDRAM interface signal names into the **Instance name dialog** box and click **OK.**

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 pin locations for the Stratix II development board, import the **altmemphy.qsf** file or manually assign pin locations by using the Pin Planner or Assignment Editor.

To manually assign pin locations, follow these steps:

1. Open Pin Planner or Assignment Editor. On the Assignments menu, click **Pin Planner** or **Assignment Editor.**
2. Assign the pin location according the DDR2 SDRAM Stratix II device pin placements recommended by Altera. Recommendations for pin assignments are as follows:
   a. The DM pins must be placed in their respective DQ group.
   b. To minimize skew, address and control command pins must be placed on any spare I/O pins ideally in the same bank or side of the device as the `mem_clk` pins.
   c. Ensure the `mem_clk` pins are placed on differential I/O pairs for the CK/CK# pin pair.
   d. 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.
   e. For the single-ended clock sources, always use the p pin of the dedicated clock inputs.
   f. Place the `global_reset_n` pin (such as any high fan-out signal) on a dedicated clock pin.
   g. If all the top bank pins are used up, you can use the LVDS pin pairs on the side banks of the device as `mem_clk` pins.


For DDR2 SDRAM Stratix II device pin placements recommended by Altera, refer to "Appendix B: Interface Signal Description" on page 96.


Create pin assignments for the PLL reference clock pin, `clock_source`, and the reset pin, `global_reset_n`. Also, when using this design example, create pin assignments for the `test_complete` and `pnf` (pass not fail) signals.

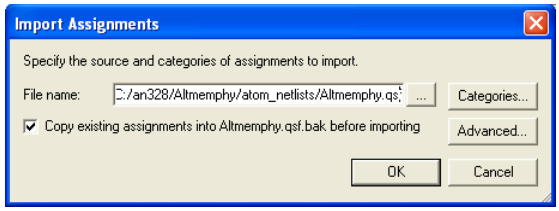

For the list of memory interface pin names and locations for the Stratix II GX PCI Express (PIPE) Development Board, refer to "Appendix A: Stratix II GX PCI-Express Development Board Pin Assignments" on page 92.

To import the pin assignments, follow these steps:

1. In the Quartus II software, on the Assignments menu, click **Import Assignments...**
2. When the **Import Assignments** dialog box appears, click the **...** button to browse for the file (located in the downloadable design example archive shown in Figure 15) **Altmemphy.qsf** and select it.

This file also includes pin assignments for the clock source, test complete, and pass/fail signals used by the top-level design example file, as well as assignments to reserve unused DDR2 memory pins (A13, A14, and BA2) as outputs driving ground.

Figure 15. Downloadable Design Example Archive



Add the Output Pin Load Assignments

Add the **output pin load** assignment as the output pin load value in "Appendix A: Stratix II GX PCI-Express Development Board Pin Assignments" on page 92. The loading for the various output and bi-directional pins on the Stratix II GX board are as follows:

- Two of the CK and CK# clock pairs have two loads = 2 × 2 pF = 4 pF. The third CK/CK# clock output pair that drives the ×8 DDR2 SDRAM components has one load = 2 pF.
- DQ and DQS pins (one load) = 4 pF
- Addr/Cmd pins (five loads) = 5 × 2 pF = 10 pF

Add the Termination or Current Strength Assignments

The default Altera recommendations for termination settings are:

- **Series 25 Ohms without Calibration** for the bi-directional DQ and DQS signals.
- **Series 50 Ohms without Calibration** for all output pins including DM, address, command, and clock signals.

  Altera recommends using the memory's on-die termination (ODT) feature when it is suitable for your system; however, the Stratix II GX PCI board features external termination on the memory side for all memory interface signals. Therefore, this example does not use memory ODT or the FPGA on-chip termination features. Instead, the design example uses the SSTL-18 Class I I/O standard along with the maximum current strength for all signals (including DQ and DQS).

After executing the <*variation_name*>**_pin_assignments.tcl** file, use the **Assignment Editor** to make the following changes:

1. Replace the SSTL-18 Class II I/O standard setting on all DQ and DQS pins (named `mem_dq` and `mem_dqs`) with the SSTL-18 Class I setting.
2. Delete the **Termination** setting of **Series 50- or 25-Ohms without Calibration** on all the DQ, DQS, DM, and CK/CK# pins (named `mem_dq, mem_dqs, mem_dm, mem_clk,` and `mem_clk_n`).
3. Add a **Current Strength** assignment set to **Maximum Current** for all DQ, DQS, DM, and CK/CK# pins.

☞ You must perform board simulations to validate if these recommendations are optimal for your system. For more information, refer to ""Step 7: Perform Board-Level Simulations to Verify the Design Constraints" on page 42.

☞ The drive strength feature is unavailable on the FPGA device when the OCT feature is enabled. If board simulations indicate that the OCT setting is not appropriate for your system, use simulations to determine the optimal drive strength setting and add that constraint to your Quartus II project.

☞ The termination and other I/O settings used in the design example are also listed in "Appendix A: Stratix II GX PCI-Express Development Board Pin Assignments" on page 92.

You have now applied all the required design constraints for this memory interface design.

Step 4: Perform RTL/Functional Simulation (Optional)

You can simulate the memory interface with the MegaWizard Plug-In Manager-generated IP functional simulation model. You must 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 and a memory model.

Use the functional simulation model with any Altera-supported VHDL or Verilog HDL simulator. This walkthrough design example uses the ModelSim® Altera edition software to perform the simulation.

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

Get the Memory Simulation Model

1. Generate the IP functional simulation model.

   You requested Verilog HDL models to be generated during the instantiate PHY and controller design flow step, described in "Step 2: Instantiate the PHY and Controller in a Quartus II Project" on page 14. Figure 11 on page 24 displays the results of your request.

   The MegaWizard Plug-In Manager also generated a Verilog HDL testbench for the design example named **ddr2_Altmemphy_example_top_tb.v**, which is located in the **testbench** directory under the project directory.

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

   The Altera DDR2 SDRAM High-Performance Controller MegaCore automatically creates a generic memory model called <*variation_name*>**_mem_model.v**

   You can use Altera's generic memory model for functional simulation, or use the memory model provided by your memory vendor. If you choose to use Altera's generic model, directly proceed to step 10 in the "Setup the Simulation Options in the Quartus II Software" on page 34.

   For this design example, obtain the **ddr2.v** and **ddr2_parameters.vh** memory model files from the Micron website and save them in the testbench directory.

Prepare the Simulation Model

1. Open the **ddr2.v** memory model file in a text editor and add the following define statements to the top of the file:

```
`define sg3
`define x8
```

   The two define statements prepare the DDR2 SDRAM memory model. The first statement specifies the memory device speed grade as -3. The second statement specifies the memory device width per DQS. This simulation uses the memory in the ×8 mode (instead of the ×16 mode).

2. Save the **ddr2.v** file.

3. Open the **ddr2_parameters.vh** file and search for the `ADDR_BITS` parameter definition. You will find instances of this parameter, one for ×4, ×8, and ×16 bit wide interfaces. Change the **ADDR_BITS** parameter for the ×8 interface to **13**, because the example memory controller only uses 13 address bits.

```
'else 'ifdef x8
parameter ADDR_BITS = 13; // MAX Address Bits
```

Instantiate the Memory Model in the Testbench

1. Open the **ddr2_Altmemphy_example_top_tb.v** testbench in a text editor.

   The testbench instantiates the design example (**ddr2_Altmemphy_example_top**) and a generic DDR2 SDRAM memory (**ddr2_Altmemphy_mem_model**) module, and connects the memory interface signals appropriately. Before running the simulation, you need to edit the testbench to use the downloaded Micron memory model.

2. Locate and delete the following instance of the generic memory model in the testbench. Note that the START and END MEGAWIZARD comments must also be deleted to ensure the MegaWizard Plug-in Manager does not overwrite the changes when the controller megafunction is regenerated.

```
// <<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 and
chipselect requirements

ddr2_Altmemphy_mem_model mem (
.mem_dq (mem_dq),
.mem_dqs (mem_dqs),
.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_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
```

3. Instantiate the first instance of the Micron ddr2 memory model as follows:

```
ddr2 memory_0(
.ck (clk_to_ram),
.ck_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 (),
.rdqs_n (),
.odt (odt_delayed),
);
```

Note the ports of the **ddr2** module are in lower-case. Because Verilog HDL is case sensitive, ensure that the port names use lower case. Also note that the **ddr2** module uses port names that are different from the generic Altera DDR2 model.

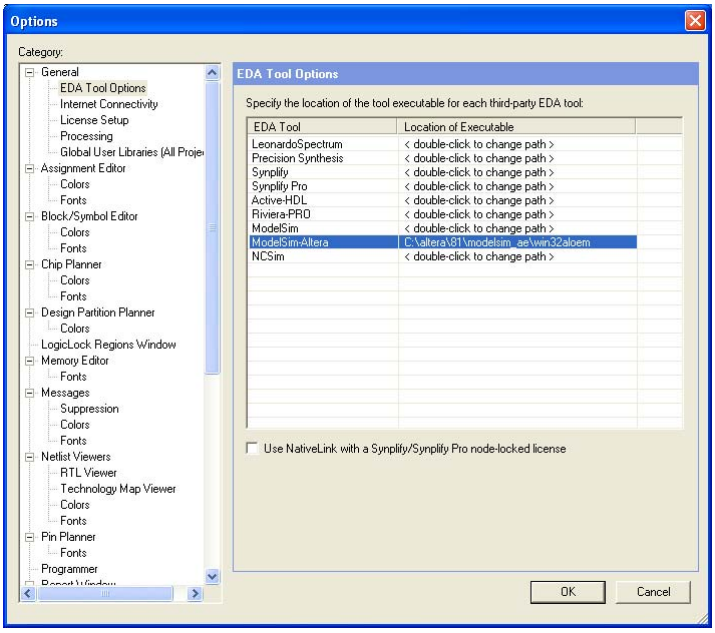
4. Similarly, create the other eight instances of the **ddr2** module. Note that the DQ, DQS, and DM bus indices are different for each instance. For example, the second ×8 memory instance must match the following:

```
ddr2 memory_1 (
.ck (clk_to_ram),
.ck_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 (),
.rdqs_n (),
.odt (odt_delayed),
);
```

Setup the Simulation Options in the Quartus II Software

1. Save the modified testbench file. Check that the absolute path to your third-party simulator executable is set. On the Tools menu, click **Options.**
2. Under the **Category** list, select **EDA Tools Options,** as shown in Figure 16 on page 34. The default path is **C:\<*version*>\modelsim_ae\win32aloem**.
3. Click **OK.**

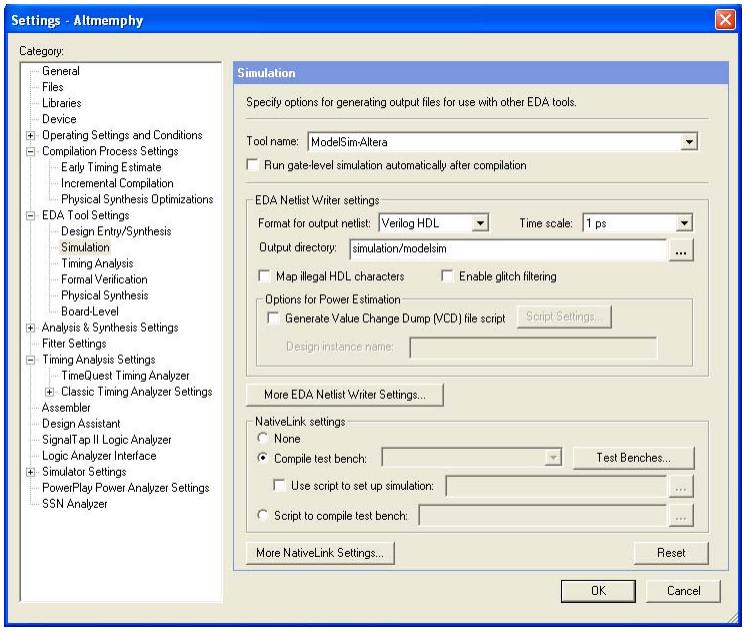
Figure 16. Set Path to ModelSim Altera Edition Software



4. On the Assignments menu, point to **EDA Tool Settings and select Simulation**. The **EDA Tool Settings** panel appears.

5. Under the list **Category** (left-hand side of the panel) click the "+" icon next to **EDA Tool Settings.**
6. Select **Simulation**.
7. From the Tool name menu in the **Simulation** panel, select the **ModelSim-Altera** simulator. Under the **EDA Netlist Writer options** section, on the pull-down menu beside **Format for output netlist**, select **Verilog**. In the **NativeLink settings** section of the **Simulation** panel, select **Compile test bench** and click **Test Benches...,** as shown in Figure 17 on page 35.

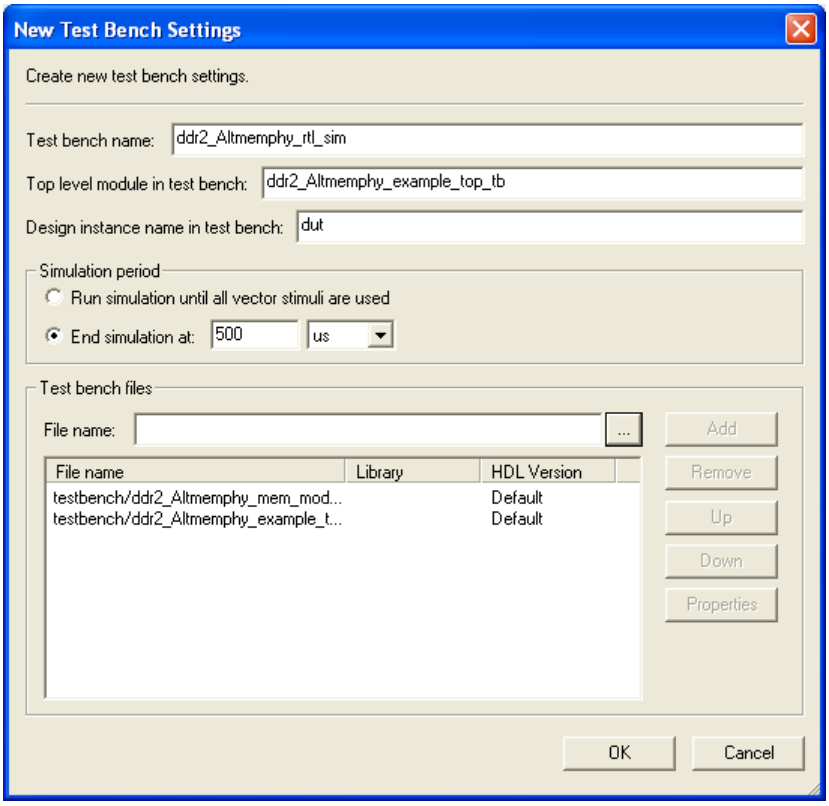
Figure 17. EDA Simulation Settings



8. In the **Test Benches** dialog box, click **New**.
9. In the **New Test Bench Settings** dialog box, enter a name in the **Test bench name** field, as shown in Figure 18. For this design example, enter `ddr2_Altmemphy_rtl_sim`.
10. Enter a name in the **Top level module in testbench** field, for example, `ddr2_Altmemphy_example_top_tb`.
11. Enter the **Design instance name in test bench** as `dut` because that is the instance name used to instantiate `ddr2_Altmemphy_example_top` in the testbench.
12. Select the **End simulation at** option under the **Simulation period** section. In the first field enter `500.` Select **µs** from the unit pull-down menu beside the first field.
13. Add the testbench files.
    a. In the **File name** field, browse to the location of the memory model and testbench and select it.
    b. In this design example, use the Micron memory model file named **ddr2.v**.
    c. If you are simulating with Altera's generic memory model, select the **<*variation_name*>_mem_model.v** file instead of the Micron model.

14. Click **OK.**
15. Click **Add**. Figure 18 on page 36 shows the fully populated **New Test Bench Settings** dialog box.
16. Click **OK** twice to save the EDA simulation settings for your project.

Figure 18. New Test Bench Settings



17. On the Processing menu, point to **Start** and click **Start Analysis & Elaboration**.

Performing Simulation in the ModelSim Software

On the Tools menu, point to the **EDA Simulation Tool** and click **Run EDA RTL Simulation**. This step creates the **\simulation\modelsim** directory under your project directory for use as the ModelSim working directory and creates a ModelSim script file, **Altmemphy_run_msim_rtl_verilog.do**, which compiles all the required design files and libraries and runs the RTL simulation. This **.do** file is regenerated every time you invoke the **Run EDA RTL Simulation** command.

☞ In the ALTMEMPHY megafunction designs targeting DDR2/DDR SDRAM interfaces for Arria GX, HardCopy II, Stratix II, and Stratix II GX devices, you must add the following line in the project **.qsf** file to allow NativeLink simulation to find the correct memory initialization file (**.mif**) for the PLL reconfiguration module:

```
set_global_assignment -name EDA_TEST_BENCH_FILE
ddr2_Altmemphy_phy_alt_mem_phy_pll.mif -section_id
ddr2_Altmemphy_rtl_sim
```

You can also type in the following in the ModelSim console:

```
do Altmemphy_run_msim_rtl_verilog.do
```

Observe the simulation status on the **Transcript** page and the memory interface signals in the **Wave** page. When the functional simulation is successful, the testbench gives a "SIMULATION PASSED" message on the **Transcript** page.

You have now completed the functional simulation of the design example.

Step 5: Compile the Design and Generate the Timing Report

You are now ready to compile your design.

1. On the Processing menu, click **Start Compilation** to compile the design.
2. After compilation is successful, in the Quartus II software, 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, as shown in Figure 19 on page 38. Executing the **Report DDR** task automatically runs the **<***variation_name***>_report_timing.tcl** timing margin report script generated by the MegaWizard Plug-In Manager when we created the megafunction variation.

For more information about the TimeQuest timing analyzer, refer to the *TimeQuest Timing Analyzer* chapter in volume 3 of the *Quartus II Handbook*.

Figure 19. TimeQuest Timing Analyzer

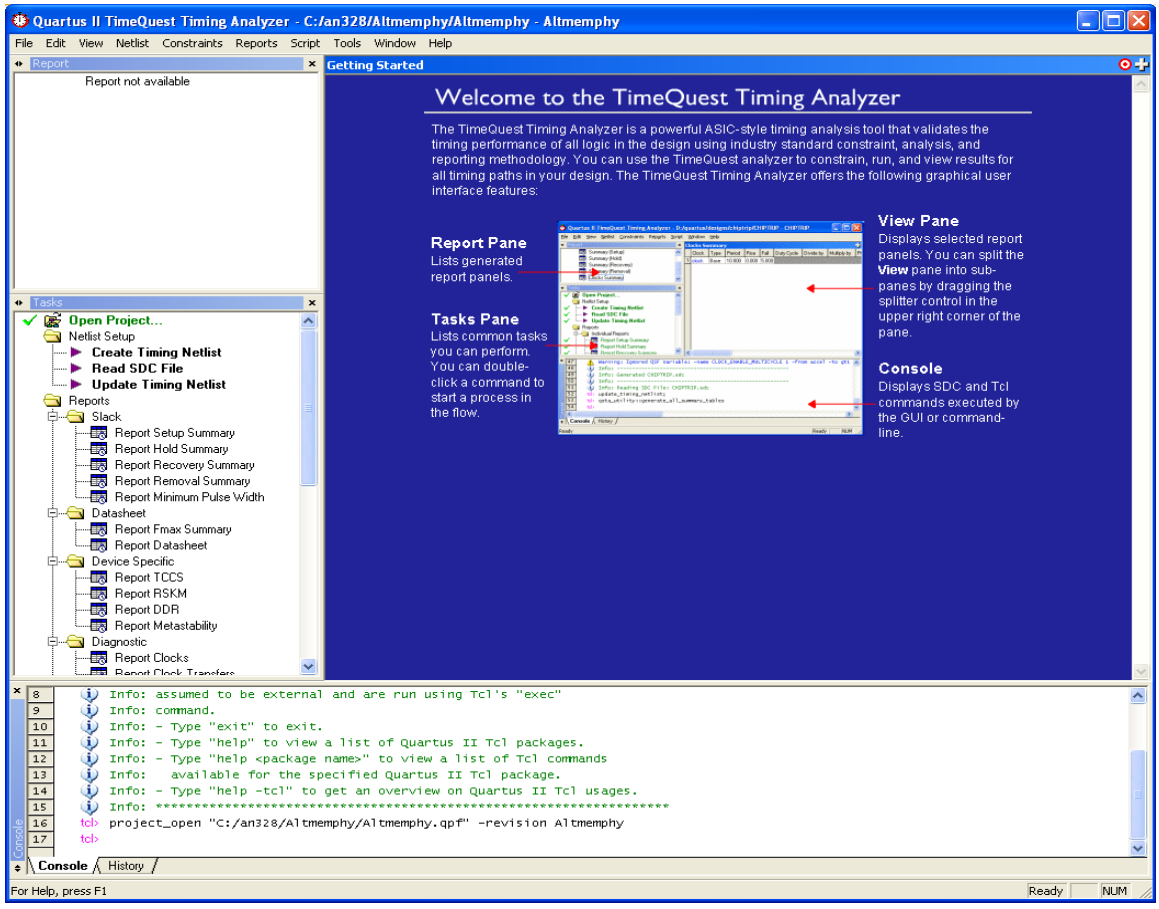
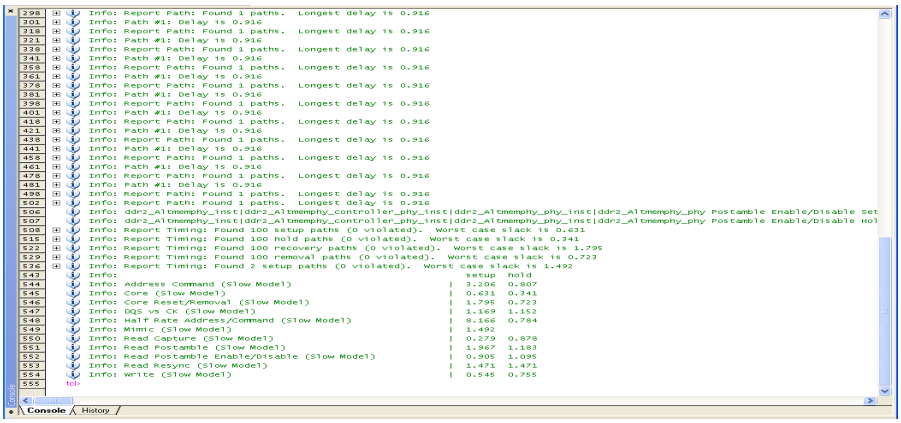


Figure 20 shows the output of the **Report DDR** task. The **Report** pane contains a new folder titled DDR with detailed timing information on the most critical paths and a timing margin summary similar to the one reported on the **TimeQuest Console**.

Figure 20. Report DDR Timing Margin Report from the TimeQuest Console



The report timing script provides timing margin information for the following paths:

- Address and command setup and hold margin
- Core setup and hold margin
- Core reset and removal setup and hold margin
- Strobe-to-clock setup and hold margin (DQS versus CK)
- Half-rate address and command setup and hold margin
- Mimic path setup margin
- Read capture setup and hold margin
- Read postamble setup and hold margin
- Read resynchronization setup and hold margin
- Write setup and hold margin

Note that this timing margin report was generated using the -3 speed grade slow timing model for the Stratix II GX device. In addition to this timing model, you must evaluate timing margins for your design using the fast timing model.

1. In the TimeQuest timing analyzer in the **Task** pane, double click on **Set Operating Conditions**.
2. In the dialog box that appears, select **MIN_fast**. Click **OK**.
3. Regenerate the timing margin report in the **Task** pane by double-clicking **Report DDR**.

Timing margin reports must be verified to be positive using all available device timing models to ensure design functionality.

You have now compiled the design and generated the timing margin report. For this design example, notice that all timing paths have positive margin.

Step 6: Adjust the Constraints

For all timing paths that have negative setup or hold margins, you must adjust the constraints on the design to improve margins for the failing path.

Generally, there are two methods to improve timing margins:

- Adjust the PLL phase shift selection for I/O timing paths (write, address and command, DQS versus CK)
- Optimize the register placement for internal timing paths (core, resync, postamble, mimic)

To avoid the core noise affect on the output jitter of the memory clock CK/CK#, follow these recommendations:

- Cascaded PLLs are not recommended for ALTMEMPHY-based designs. You can only cascade PLLs between adjacent PLLs on the same side of the device by using the adjacent PLL (direct connection) method.
- Any PLL on any side of a Stratix II device can support a single ALTMEMPHY interface. Ideally, you must pick a PLL and a PLL input clock pin located on the same side of the device as the memory interface pins.

Example 1: Postamble

You can improve the postamble timing path setup margin by placing the postamble enable register (`postamble_en_pos_2x[n]`) closer to the `DQS[n]` pin that it is driving.

For example, `DQS[5]` is driven by the following postamble enable register:

```
ddr2_Altmemphy:ddr2_Altmemphy_inst|ddr2_Altmemphy_controller_ph
y:ddr2_Altmemphy_controller_phy_inst|ddr2_Altmemphy_phy:alt_mem
_phy_inst|ddr2_Altmemphy_phy_alt_mem_phy_sii:ddr2_Altmemphy_phy
_alt_mem_phy_sii_inst|ddr2_Altmemphy_phy_alt_mem_phy_postamble_
sii:poa|postamble_en_pos_2x[5]
```

Because `DQS[5]` is assigned to pin location AU23, assign the postamble enable register to the LAB nearest to that I/O pin, `LAB_x33_Y1`. This improves the setup margin for that path.

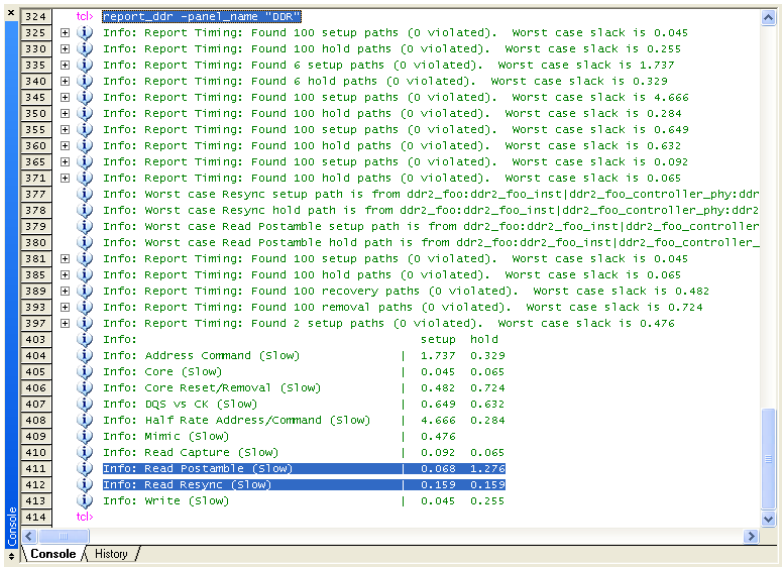
Manually locating each of the nine DQS pins in the design example and creating postamble register location constraints to the nearest logic array block (LAB) is a tedious process. The **relative_constraint.tcl** utility is a handy tool that generates LAB location assignments for a bus of registers relative to the location of related pins. Detailed descriptions of this script's options and usage are available in "Appendix E: The relative_constraint.tcl Script" on page 134.

For this design example, the following three commands place the nine postamble registers (named "`*postamble_en_pos_2x[*]`") near the corresponding DQS pin, and the 72 high and 72 low resynchronization registers (named "`rdata_p_ams[*]`" and "`rdata_n_ams[*]`") near the corresponding DQ pin.

```
- quartus_sh -t relative_constraint.tcl -project Altmemphy -pin_name
"*mem_dqs[*]" -reg_name "*postamble_en_pos_2x[*]" -row_offset 1 -apply
- quartus_sh -t relative_constraint.tcl -project Altmemphy -pin_name
"*mem_dq[*]" -reg_name "*rdata_p_ams[*]" -row_offset 1 -apply
- quartus_sh -t relative_constraint.tcl -project Altmemphy -pin_name
"*mem_dq[*]" -reg_name "*rdata_n_ams[*]" -row_offset 1 -apply
```

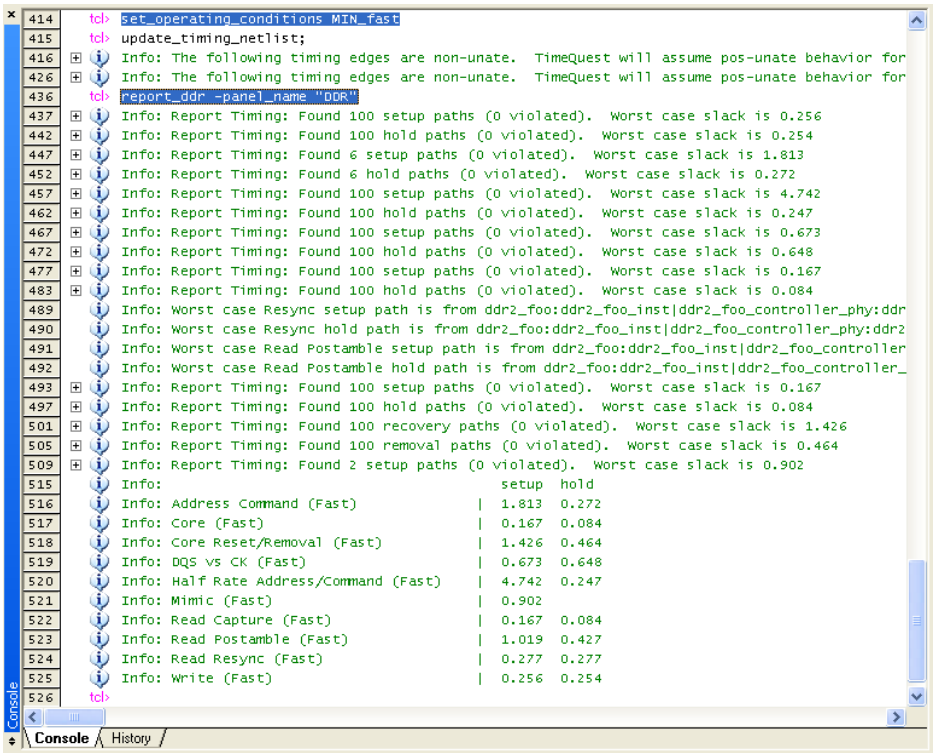
Recompile the design and regenerate the timing margin report with these new location assignments. Figure 21 on page 41 shows the timing margin report with the postamble and resynchronization timing optimizations.

Figure 21. Timing Report Example 1—Slow Timing Model



Now that the slow timing model margins are all positive, verify timing margins using the fast timing model. Figure 22 shows the design example timing margins using the minimum timing model.

Figure 22. Timing Report Example 2—Fast Timing Model



Example 2: Address and Command

Suppose that the address and command timing path had a negative hold margin. You can adjust the PLL clock phase shift setting for the clock generating the address and command signals to improve the hold margin (by shifting the clock such that data is launched earlier). This clock setting is adjusted by changing the **clock phase** setting for the address and command clock in the **PHY Settings** page of the DDR2 SDRAM High Performance Controller MegaWizard Plug-In Manager, as shown in Figure 8 on page 20.

Step 7: Perform Board-Level Simulations to Verify the Design Constraints

For this design example, the board design constraints are already determined. However, for your actual board, determine the optimal termination scheme, termination implementation (OCT versus external resistors), drive strength settings, and system loading.

Evaluate trade-offs posed by various board design choices using simulations. Different factors contribute to signal integrity and affect the overall timing margin for the memory and FPGA. These include the termination scheme used, slew rate, and drive strength settings on the FPGA, and the loading seen by the driver. You must run board-level simulations to evaluate the trade-offs among the different types of termination schemes, the effects of output drive strengths, and loading, so that you can navigate through the various design choices and choose optimal settings for your design.

Table 8 lists the Altera-recommended board design constraints. However, ensure that the constraints below satisfy your application's needs. For example, the Stratix II GX PCI Express (PIPE) Development does not use ODT. Additionally, the board form factor only allows them to use single parallel termination for the bi-directional signals.

Table 8. Termination Recommendations (Part 1 of 2) *(Note 1)*

Form Factor	Signal Type	SSTL 18 I/O Standard *(2)*, *(3)*, *(4)*, *(5)*, *(6)*	FPGA End Discrete Termination	Memory End Termination 1 Rank/DIMM	Memory I/O Standard
DDR2	DQ/DQS	Class I R50 NO CAL	50 Ω Parallel to V_{tt} discrete	ODT75 *(8)*	FULL *(9)*
	DM	Class I R50 NO CAL	N/A	56 Ω Parallel to V_{tt} discrete	N/A
	Add/Cmd	Class I MAX	N/A		N/A
	Clock/Clock#	Class I MAX	N/A	N/A = on DIMM	N/A

Table 8. Termination Recommendations (Part 2 of 2) *(Note 1)*

Form Factor	Signal Type	SSTL 18 I/O Standard *(2)*, *(3)*, *(4)*, *(5)*, *(6)*	FPGA End Discrete Termination	Memory End Termination 1 Rank/DIMM	Memory I/O Standard
DDR2 Discrete	DQ/DQS	Class II R25 NO CAL	50 Ω Parallel to V_{tt} discrete	ODT75 *(8)*	HALF *(7)*
	DM	Class I R50 NO CAL	N/A	ODT75 *(8)*	N/A
	Add/Cmd	Class I R50 NO CAL	N/A	56 Ω Parallel to V_{tt} discrete	N/A
	Clock/Clock#	Class I R50 NO CAL	N/A	×1 = 100 Ω differential *(10)*	N/A
				×2 = 200 Ω differential *(11)*	

Notes to Table 8:

(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) HALF is reduced drive strength.
(8) ODT75 verses ODT50 on the memory has the effect of opening the eye more, with a limited increase in overshoot, undershoot, or both.
(9) FULL is full drive strength.
(10) ×1 is a single device load.
(11) ×2 is two device loads.

To determine the correct board constraints, run 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 electronic design automation (EDA) simulation tools available to perform board-level simulations. Perform the simulations on the data, clock, control, command, and address signals. If the memory interface does not have good signal integrity, adjust the settings, for example, the drive strength setting, termination scheme, or termination values, to improve the signal integrity. Realize that changing these settings affects timing. It may be necessary to go back to the timing closure step if these settings change.

Step 8: Verify the Functionality on the Hardware

Perform system-level verification to correlate your system performance against your design targets. Use Altera's SignalTap II logic analyzer to help in this effort. At this point, you are now ready to verify functionality on the hardware and implement memory interfaces such as this in your current and future designs.

For detailed information about using SignalTap II, refer to the *Design Debugging Using the SignalTap II Embedded Logic Analyzer* chapter in volume 3 of the *Quartus II Handbook*.

Design Example Walkthrough for a 267-MHz DDR2 SDRAM Interface Using the Legacy PHY

This walkthrough describes the steps necessary to create, constrain, and verify the operation of a 267-MHz DDR2 SDRAM memory interface on a Stratix II GX device. This design example uses the DDR2 SDRAM Controller MegaCore (with the legacy-integrated static data path and controller) to target the Stratix II GX PCI Express (PIPE) Development Board. The design example created with this walkthrough is available for download along with this application note. Using the walkthrough, you will create a 72-bit wide, 267-MHz/533-Mbps DDR2 SDRAM memory interface targeted for the Stratix II GX PCI Express (PIPE) Development Board. This walkthrough design example functions as a step-by-step guide to reproducing the design example.

If you are using ALTMEMPHY, go to "Design Example Walkthrough for a 333-MHz DDR2 SDRAM Interface Using the ALTMEMPHY Megafunction" on page 11. All memory interfaces using Arria GX devices must use ALTMEMPHY.

Legacy PHY is a memory interface PHY that enables speeds of up to 267 MHz on Stratix II and Stratix II GX devices. The legacy PHY uses dedicated DQS phase-shift circuitry for capturing data from memory and a PLL phase shift determined during compilation to resynchronize memory read data to the system clock domain.

For more information about the legacy PHY architecture and memory controller, refer to "Appendix C: Legacy PHY Architecture Description" on page 102 and the *DDR and DDR2 SDRAM High-Performance Controller and ALTMEMPHY User Guide*, respectively.

The Stratix II GX edition of Altera's PCI Express (PIPE) Development Board delivers a complete PCI Express-based (PIPE) development platform. This PCI Express (PIPE) solution, interoperable with industry-standard PCI Express (PIPE) platforms, facilitates the development of custom PCI Express (PIPE) applications.

For more information about the PCI Express Development Board, refer to the *PCI Express Development Kit, Stratix II GX Edition*.

This design example walks through the memory interface design flow steps shown in Figure 1 on page 7.

For more details about this design flow, see *AN 449: Design Guidelines for Implementing External Memory Interfaces in Stratix II and Stratix II GX Devices*.

The design example was created using the Quartus II software version 9.0 and MegaCore IP Library software version 9.0.

This example focuses on the Stratix II GX device family. However, the information is also applicable to the memory interface designs using legacy PHY that target Stratix II and HardCopy II devices. Pay attention to the restrictions for HardCopy II devices, described in *AN 413: Using Legacy Integrated Static Data Path and Controller Megafunction with HardCopy II Structured ASICs.*

Step 1: Select the Device

This example uses the EP2SGX90FF1508C3 device that comes with the Stratix II GX PCI Express (PIPE) Development Board. The board is also equipped with five 333-MHz capable DDR2 SDRAM components—four ×16 devices with part number MT47H32M16CC-3 and one ×8 device with part number MT47H64M8CB-3.

For more information about all the available features of the board, refer to the *Stratix II GX PCI Express Development Board Reference Manual*.

The design example uses five memory devices to create a 72-bit interface running at 267 MHz using the 2-PLL DQS mode, described in "Appendix C: Legacy PHY Architecture Description" on page 102, with the DDR2 SDRAM Controller Compiler MegaCore function. This is considered a width-expansion interface because multiple memory devices are used to create one wide interface using a single memory controller.

The maximum number of interfaces you can implement on any given device is limited by resource availability (number of DQ groups of desired width, user I/O pins, PLLs, DLLs, clocks, and FPGA core resources). Expanding your memory interfaces for width or depth with the same memory controller is supported by the MegaWizard Plug-In Manager. Creating multiple memory controllers with independent memory transactions may require RTL modifications depending on whether you must share any device resources.

To determine the number of DQ groups of each width that are supported by the FPGA, refer to the External Memory Interfaces chapter of the *Stratix II Device Handbook* or the *Stratix II GX Handbook*.

Expanding your memory interfaces for width or depth with the same memory controller (shared address/command bus) is supported natively by the MegaWizard Plug-In Manager. Creating multiple memory controllers with independent memory transactions (independent address/command buses) requires you to create a unique megafunction variation for each interface. Sharing device resources between multiple memory interfaces may require RTL modifications.

For more information about sharing device resources between multiple memory interfaces, refer to *AN 392: Implementing Multiple Legacy DDR/DDR2 SDRAM Controller Interfaces*.

Both MT47H32M16CC-3 and MT47H64M8CB-3 devices have the same timing specifications because they share the same data sheet. If you are using devices with different data sheets, choose the worst-case specifications.

Step 2: Instantiate the PHY and Controller in a Quartus II Project

Stratix II and Stratix II GX devices have a few legacy PHY implementations that you can choose from, listed in Table 9.

Table 9. Available DDR2 SDRAM Legacy PHY Implementations for Stratix II and Stratix II GX Devices

Implementation Variations	When to Use
Non-DQS	■ When using the side I/O banks ■ When interfacing with more than 72-bit data per Stratix II device side ■ When not using the DLL for read capture
One PLL	For interfaces that cannot use ALTMEMPHY megafunction (for example, when you need lower latency interfaces than offered by the ALTMEMPHY megafunction) and are running at or below 200 MHz. *(1)*
Two PLLs	When you cannot use the ALTMEMPHY megafunction (for example, when you need lower latency interfaces than offered by the ALTMEMPHY megafunction) and cannot achieve the required performance with the legacy PHY using one PLL. This implementation is limited to a maximum of up to 267 MHz. *(1)*

Note to Table 9:

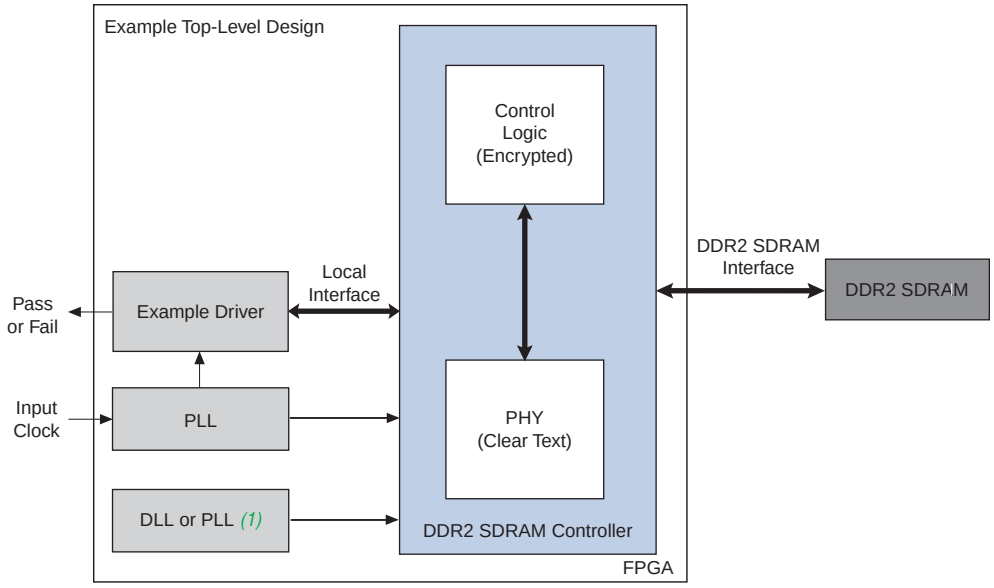
(1) For legacy PHY, always perform timing analysis using the DDR Timing Wizard (DTW) with the targeted device to ensure you can run at your desired frequency.

Maximum performance for the legacy PHY implementations depend on:

- FPGA density
- Memory speed grade
- Board trace length skew
- PHY variation (non-DQS, 1-PLL, 2-PLL)

Figure 23 shows a system-level diagram for the legacy controllers. An example top-level design is created when you generate a DDR2 SDRAM Controller MegaCore function. This top-level design includes an example driver in addition to the memory controller, PLL, and DLL modules. The memory controller itself consists of a clear-text PHY module and an encrypted control logic module. When using your own memory controller, you must manually extract the PHY from the encrypted controller, keeping the PLL and DLL connections to the PHY.

Figure 23. DDR2 SDRAM Legacy Controller System Level Diagram



Note to Figure 23:

(1) When using dedicated DQS phase-shift circuitry, the DLL center-aligns the DQS strobe to the DQ data bus during read operations. The DLL input reference clock can come from either PLL5 or `CLK[15..12]p` for the DQS phase-shift circuitry on the top I/O banks and PLL6 or `CLK[7..4]p` for the DQS phase-shift circuitry on the bottom I/O banks. When not using the dedicated DQS phase-shift circuitry, a PLL implements this phase shift.

The legacy PHY requires a project, targeted to a specific device, to be open before you invoke the DDR2 SDRAM MegaWizard Plug-In Manager. Create a project in the Quartus II design software or open a project if you already have one created.

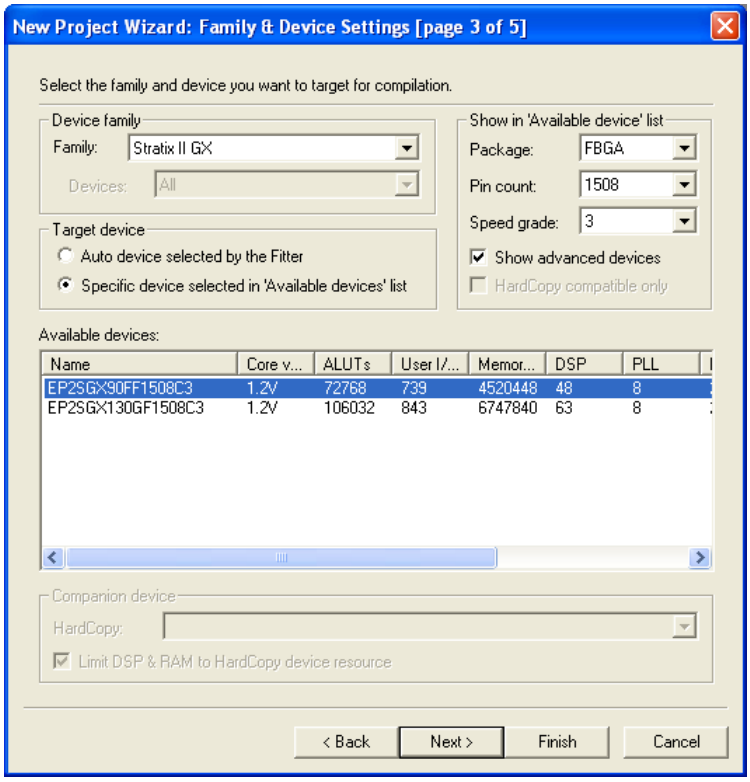
You can either create a new Quartus II project or open an existing project where you would like to implement the DDR2 SDRAM memory interface. When creating a new project, specify the target FPGA device on page 3 of the **New Project Wizard: Family and Device Settings**. Set **Stratix II GX** as the **Family** and choose the **EP2SGX90FF1508C3 device** from the **Available devices** list.

When using an existing project, set the target FPGA device by opening the Assignments menu and selecting **Device...** Set **Stratix II GX** as the **Family** and choose the **EP2SGX90FF1508C3 device** from the **Available devices** list, as shown in Table 3 on page 5. Device listings displayed are filtered by the fastest speed grade and the 1508-pin FPGA package. The Quartus II project name for this design example is **Legacy_PHY**.

Refer to the tutorial in the Quartus II Help menu for step-by-step instructions for creating a Quartus II software project.

Figure 24 on page 48 shows a screenshot with the **filter** options on the top right side of the page to display the fastest speed grade devices available in a 1508-pin FBGA package. For this example, the project name is **Legacy_PHY.qpf.**

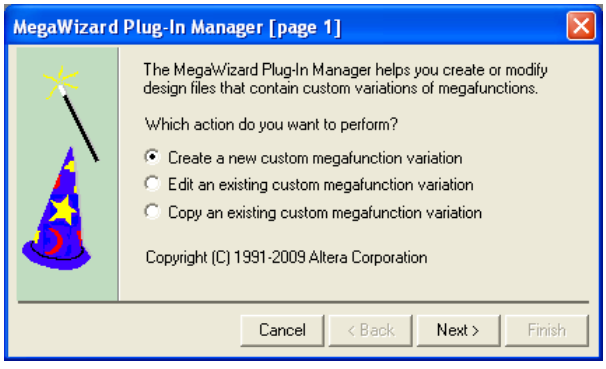
Figure 24. Select the Target FPGA Device in Quartus II Software



Because the target frequency is 267 MHz, the design example uses the 2-PLL implementation of the legacy controller. Create this interface using the following steps:

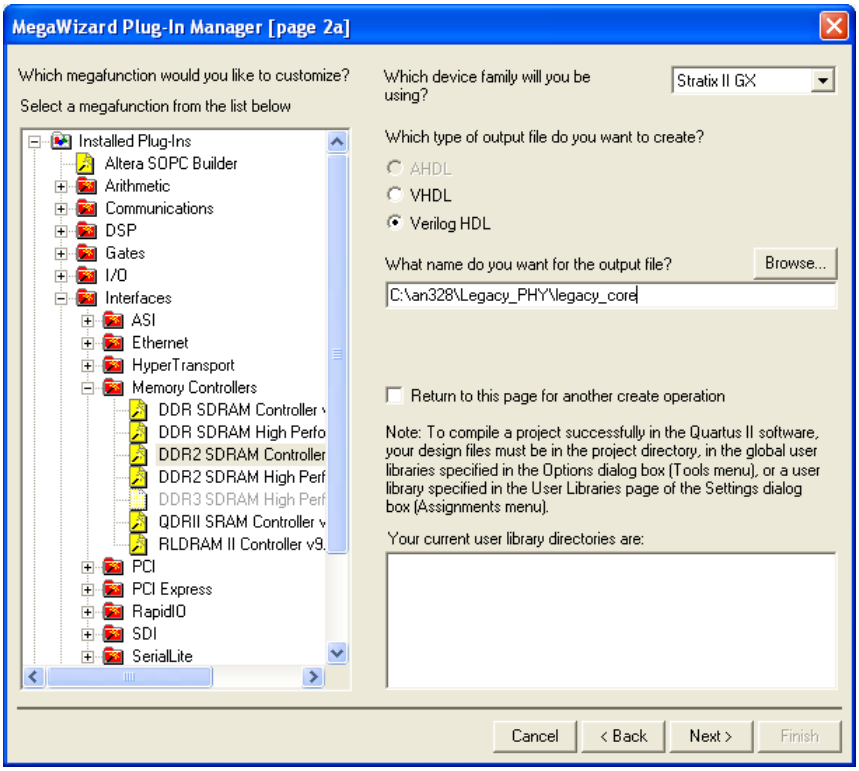
1. Open the Tools menu in the Quartus II software and select **MegaWizard Plug-In Manager**.
2. In the **MegaWizard Plug-In Manager [page 1]** dialog box, select **Create a new custom megafunction variation** (Figure 25) and click **Next**.

Figure 25. Launching the MegaWizard Plug-In Manager



3. On the **MegaWizard Plug-In Manager [page 2a]** dialog box, under the **Select a megafunction from the list below** list, click the "+" icon next to **Interfaces**.
   a. Click the "+" icon next to **Memory Controllers** and select the **DDR2 SDRAM Controller v9.0 megafunction**.
   b. Ensure the pull-down menu in the upper right-hand corner of the dialog box is set to **Stratix II GX**, as the project is targeted to the EP2SGX90FF1508C3.
   c. Select **Verilog HDL** as the output file type and enter an instance name of your choice (for example, **legacy_core**).
   d. Click **Next** (Figure 26).

Figure 26. Creating a Megafunction Variation



4. In the next dialog box (Figure 27), select **Parameterize** to create the 267-MHz memory interface. Click **Next**.

Figure 27. DDR2 SDRAM Controller Window

5. The **Parameterize** dialog box appears, showing seven tabs representing different groups of settings for the memory interface. First, choose a memory device under the **Presets** list and a clock speed for the interface.

   Because MT47H32M16CC-3 is not in the **Presets** list, choose **MT47H64M16BT-37E**. You can use this as a base before modifying the numbers to match the actual memory device specifications. This device has the closest timing specifications to the devices on the board because it is the fastest Micron device listed in the **Presets** list.

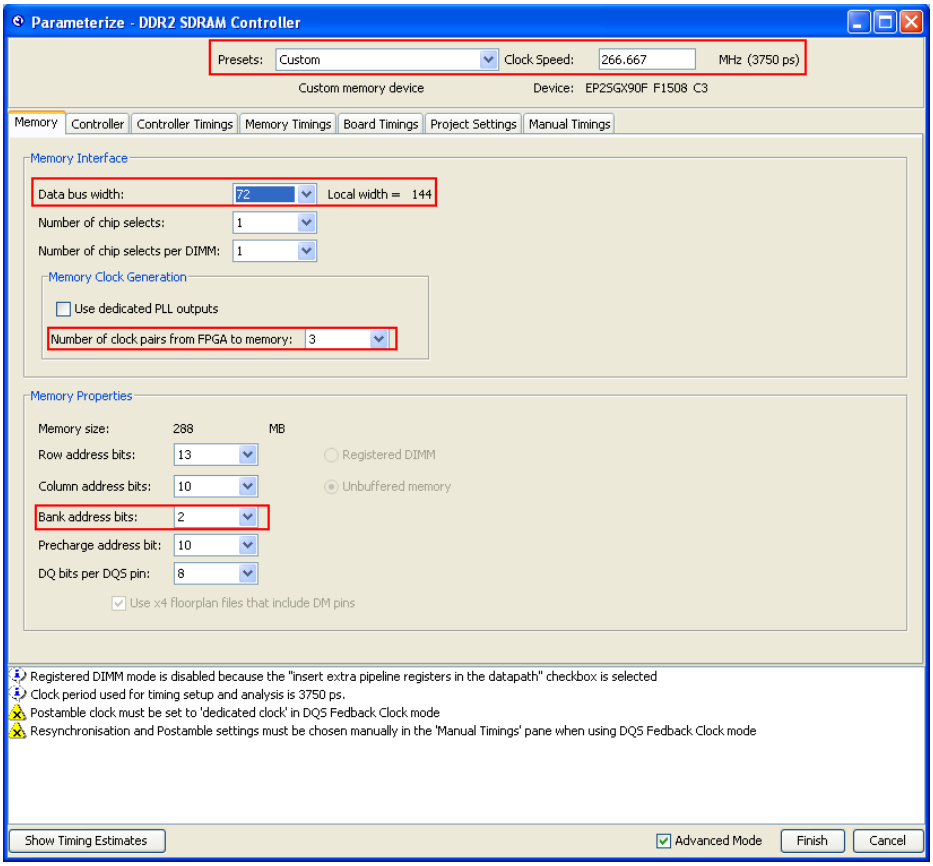
6. Set **Clock Speed** to **266.667 MHz**. This is so that the PLL can get the accurate clock period for the interface. The PLL may not lock if the clock period is not exact.

   Figure 28 through Figure 34 show screens from the DDR2 SDRAM MegaWizard Plug-In Manager with all the changes that were made from the base settings highlighted in red boxes.

☞ The parameters that you need to change vary with the memory device and the base settings that you chose in the MegaWizard Plug-in Manager. You must check that each parameter matches the memory device data sheet values.

7. In the **Memory** page, shown in Figure 28, do the following:
   a. Set **Data bus width** to **72** to reflect the actual interface width.
   b. Set **Number of clock pairs from FPGA to memory** to **3**, because there are three pairs of clocks connected to the five memory devices. Two of the clock-pair signals go to two ×16 memory devices each, while one clock-pair signal goes to the ×8 memory device.
   c. Set **Bank address bits** to **2**, as the MT47H32M16CC-3 and MT47H64M8CB-3 devices only have four banks (instead of eight banks as in the base MT47H64M16BT-37E device). The **Presets** dialog box automatically changes from **MT47H64M16BT-37E** to **Custom**. This occurs any time you change any **Memory Properties** item.

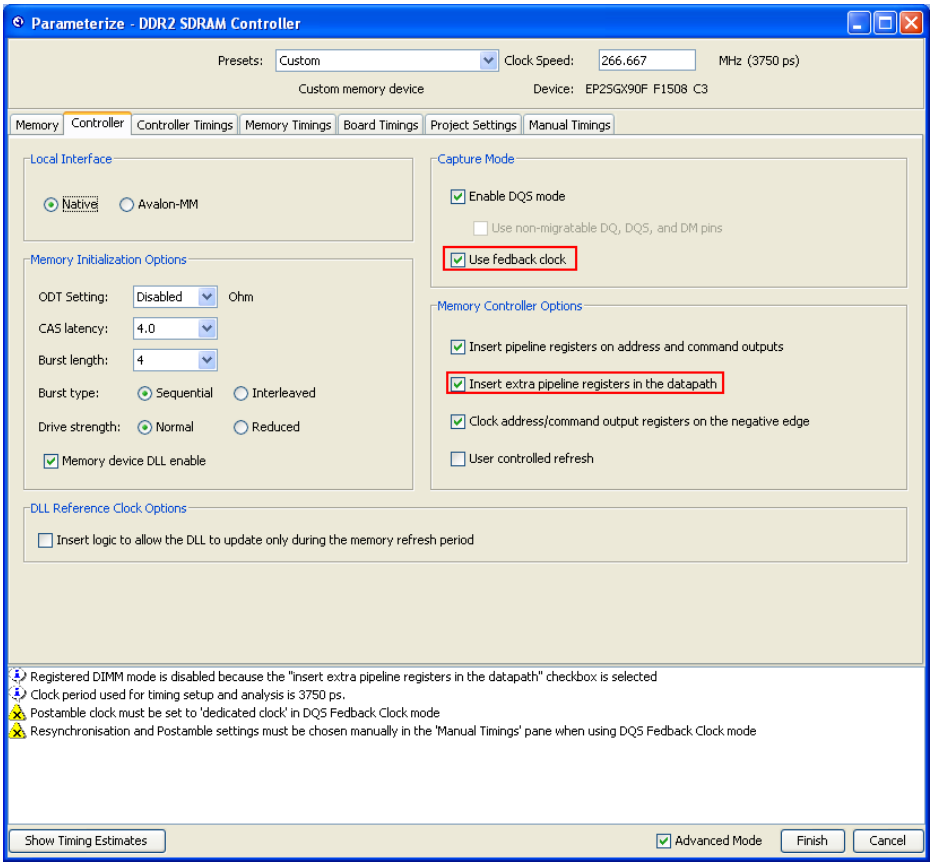
Figure 28. Memory Settings



The message window on the bottom of the screen gives you a warning to use fedback clock mode for interfaces greater than 200 MHz. You must always heed the warnings and follow the recommendations shown here.

8. In the **Controller** page, enable the **Use fedback clock** option because the design is running above 200 MHz and turn on the **Insert extra pipeline registers in the datapath** option to allow a dedicated PLL output to be connected to generate the address and command signals. Refer to Figure 29 to view the results of these changes.

Figure 29. Controller Settings



Turning on the **Insert extra pipeline registers in the datapath** option propagates the address and command clock to the top-level design. This makes it easier to connect a different PLL output (other than the default negative edge of the system clock) if the timing results after compilation show that you need to shift this clock. However, there will be an additional clock cycle of latency, because a second pipeline register is inserted between the memory controller and the address and command outputs. This means that you cannot use this option if you use a registered DIMM, as address and command signals are registered on-board in registered DIMMs. Only use this option if you also turn on the **Insert pipeline registers on address and command outputs** option.

The previous warning, shown in Figure 28 on page 51, disappears after choosing the **Use fedback clock** option. However, two new warnings show up at the bottom of the MegaWizard Plug-In Manager. Modify the **Manual Timings** page to heed these warnings.

You must use the memory's on-die termination (ODT) option when it is suitable for your system. However, the Stratix II PCI-Express (PIPE) Development Board uses an external resistor for termination on the memory side for all memory interface signals, so this design example does not use ODT.

9. In the **Controller Timings** and **Memory Timings** pages, modify the numbers based on the MT47H32M16CC-3 or MT47H64M8CB-3 data sheet, as shown in Figure 30 and Figure 31.

Figure 30. Controller Timings

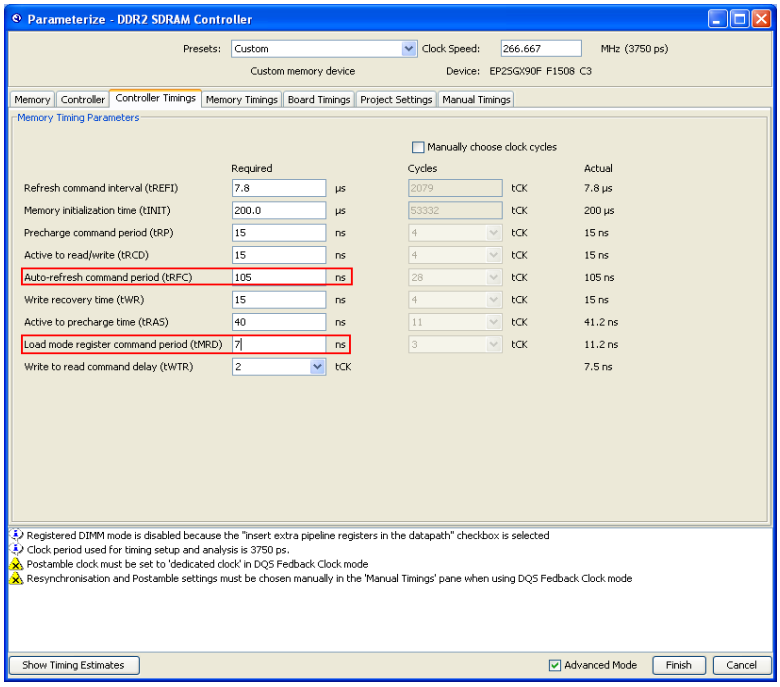
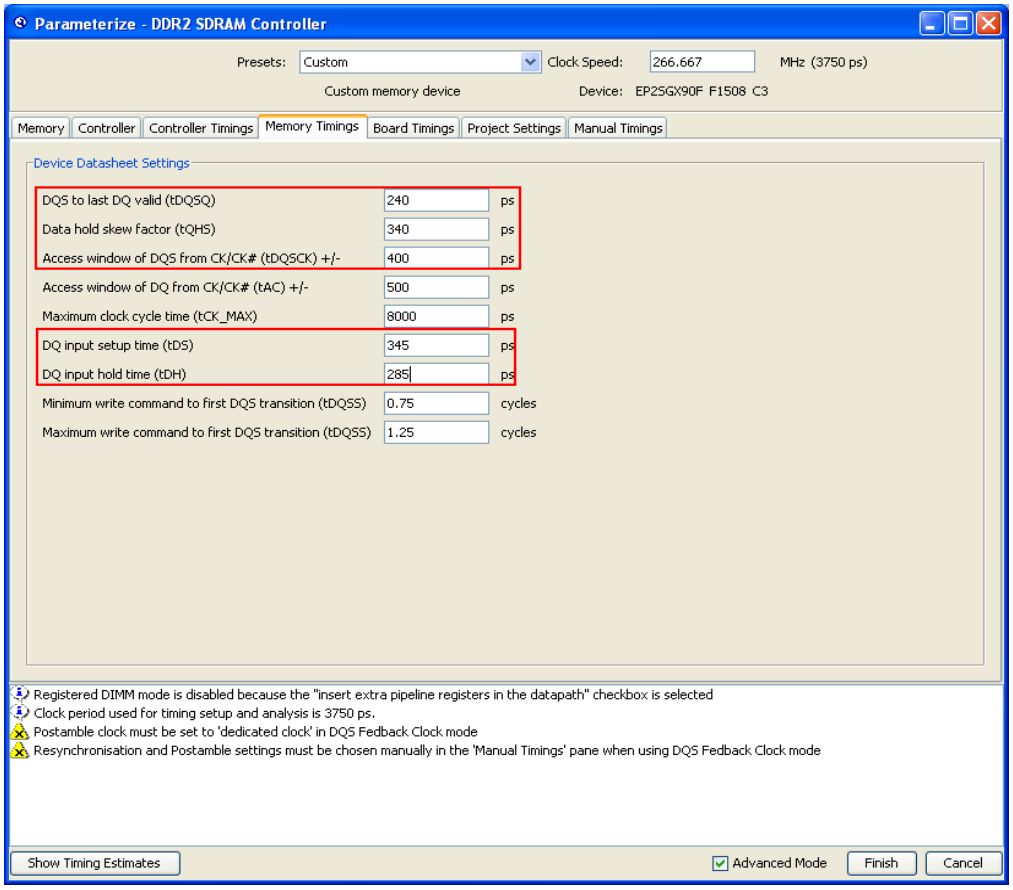


Figure 31. Memory Timings

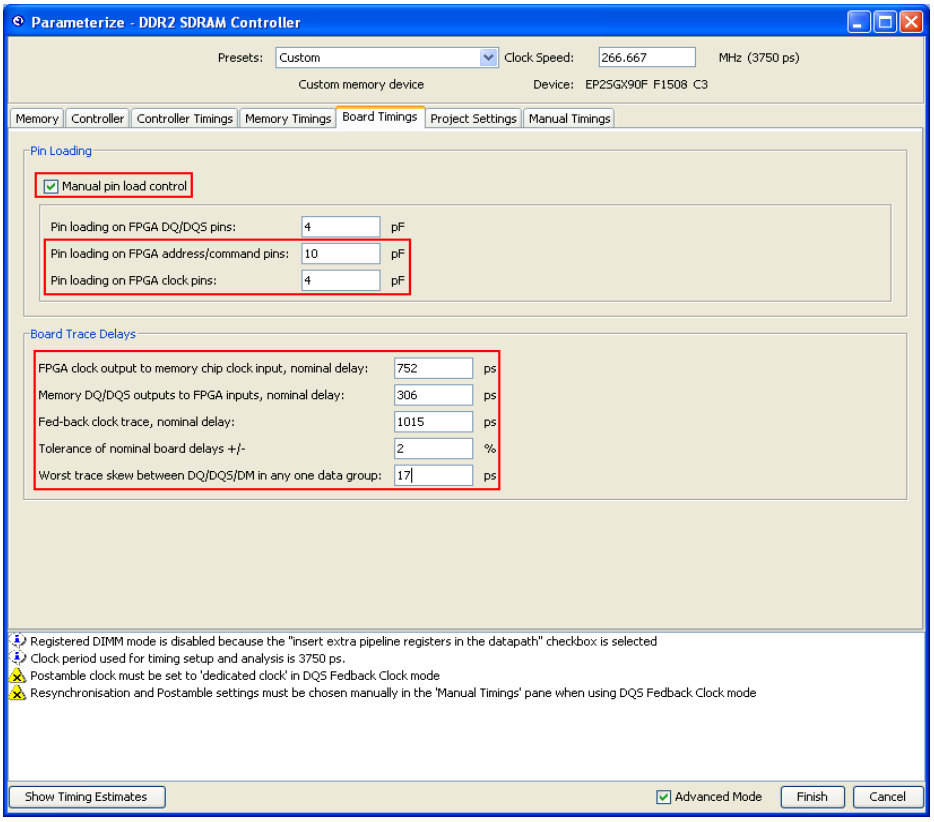


10. In the **Board Timing** page, (Figure 32) do the following:
    a. Turn on the **Manual pin load control** box.
    b. Modify pin loading with the pin capacitance specification from the memory data sheet, as seen by the FPGA output pins.

       The address and command pins are connected to five devices, so multiply the capacitance listed by five.

       Multiply the capacitance on the clock pins by two, as the clock pins going to the ×16 devices are double-loaded. You can change the capacitance for the ×8 device in "Step 3: Add the Constraints" on page 61 as that clock pin-pair only goes to one memory device.
    c. Modify the board timing information per the Stratix II GX PCI Express (PIPE) Development Board specification, shown in Figure 32.

Figure 32. Board Timings



11. In the **Project Settings** page, turn off the **Automatically verify datapath-specific timing in the Quartus II project** option, shown in Figure 33 on page 56. This example uses the DDR Timing Wizard (DTW) to close timing.
12. In the **Project Settings** page, set **Prefix all pins on the device with** to **mem_**.

Figure 33. Project Settings

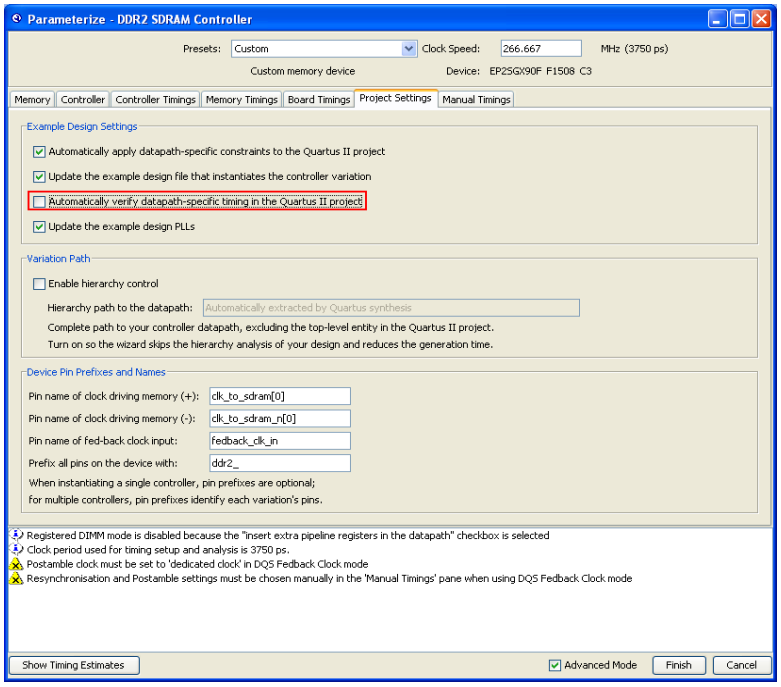
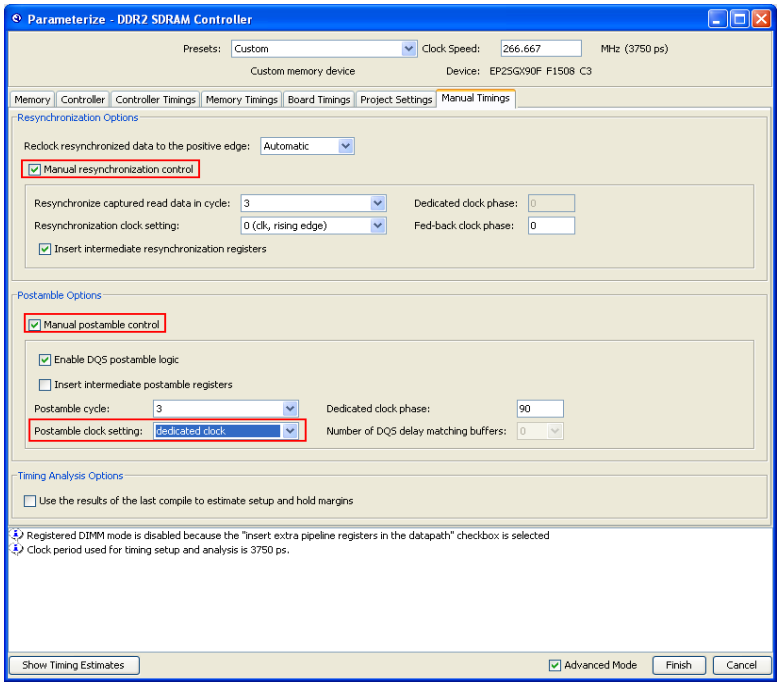


Figure 34 shows the **Manual Timings** page, which displays resynchronization and postamble controls that you can change manually. The MegaWizard Plug-in Manager calculates and implements the resynchronization and postamble clock phase shifts based on the information entered in previous pages for the 1-PLL implementation. However, you must turn on the **Manual resynchronization control** and **Manual postamble control** check boxes for the 2-PLL mode implementation.

13. In the **Postamble clock setting** pull-down list, select **dedicated clock**, which is another 2-PLL mode requirement.

These changes are necessary for correct RTL connection between the resynchronization and postamble paths and their respective clocks or else the design may not function properly.

Figure 34. Manual Timings



14. Click **Finish**.
15. In the **DDR2 SDRAM Controller MegaWizard** page, click **Constraints** (Figure 27 on page 50). Fix the location of the DQS and DQ pins per board specifications. Figure 35 shows how the DQS groups are laid out on the board. The layout of the constraints window resembles Stratix II or Stratix II GX DQS and DQ grouping layout on the top and bottom of the device.
16. Click **OK**.

Figure 35. Memory Interface Location for the Stratix II GX PCI Express Development Board

Numbers 0 through 8 that you select from the pull-down menu represent the `DQS[8..0]` pins in the design. You do not need to have the numbers in order from left to right, or right to left, as long as the user logic after the controller knows how to parse the data coming in. The DQS pins selected here are to match the Stratix II GX PCI Express (PIPE) Development Board connection with the DDR2 SDRAM components.

You cannot select DQS pins from both the top and bottom side of the device. Each memory interface must reside in one side of the device.

CAUTION

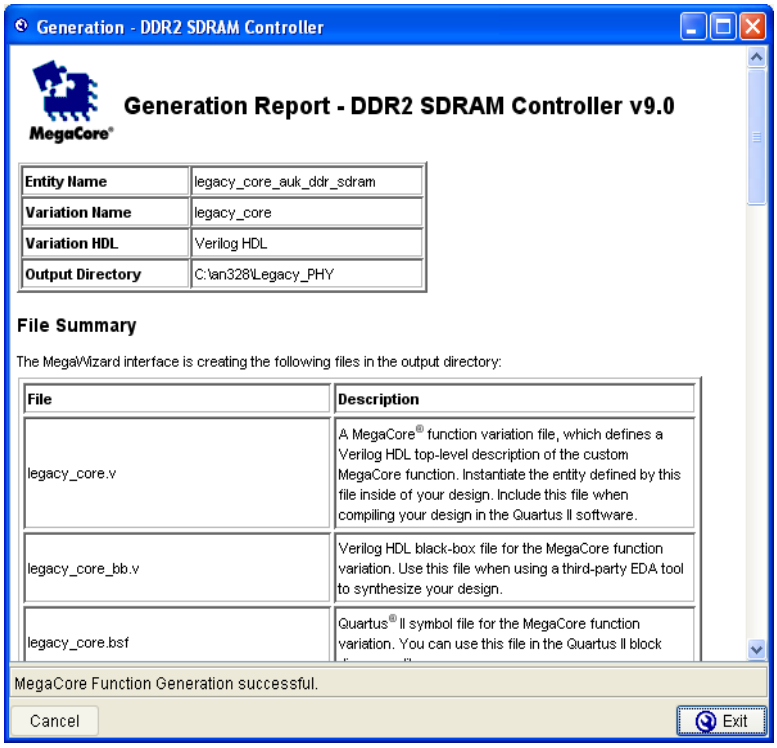
The DQ pins associated with a DQS pin are fixed in the DDR2 SDRAM MegaWizard Plug-in Manager database. If you need to swap the locations for any of the DQ pins, use the **Assignment Editor** or the Pin Planner. In addition, remember to turn off the **Automatically apply datapath-specific constraints to the Quartus II project** option in the **Project Settings** panel of the DDR2 SDRAM MegaWizard Plug-in Manager if you need to regenerate the controller at a later time.

17. In the **DDR2 SDRAM controller MegaWizard** page, click **Set Up Simulation**. Turn on the **Generate Simulation Model** check box and choose either **Verilog HDL** or **VHDL** to generate a **.vo (legacy_core.vo)** file or a **.vho (legacy_core.vho)** file used to simulate the design. Click **OK** when you are done.

You can also turn on the **Generate netlist** check box in this page to generate a netlist for a third-party synthesis tool for resource usage and timing report estimation.

18. In the **DDR2 SDRAM controller MegaWizard** page, click **Generate**. (Figure 27 on page 50) to generate all the files needed for this memory interface. This action generates a summary of the interface when the generation is successful, as shown in Figure 36.
19. To close the **DDR2 SDRAM controller MegaWizard** page, click **Exit**.

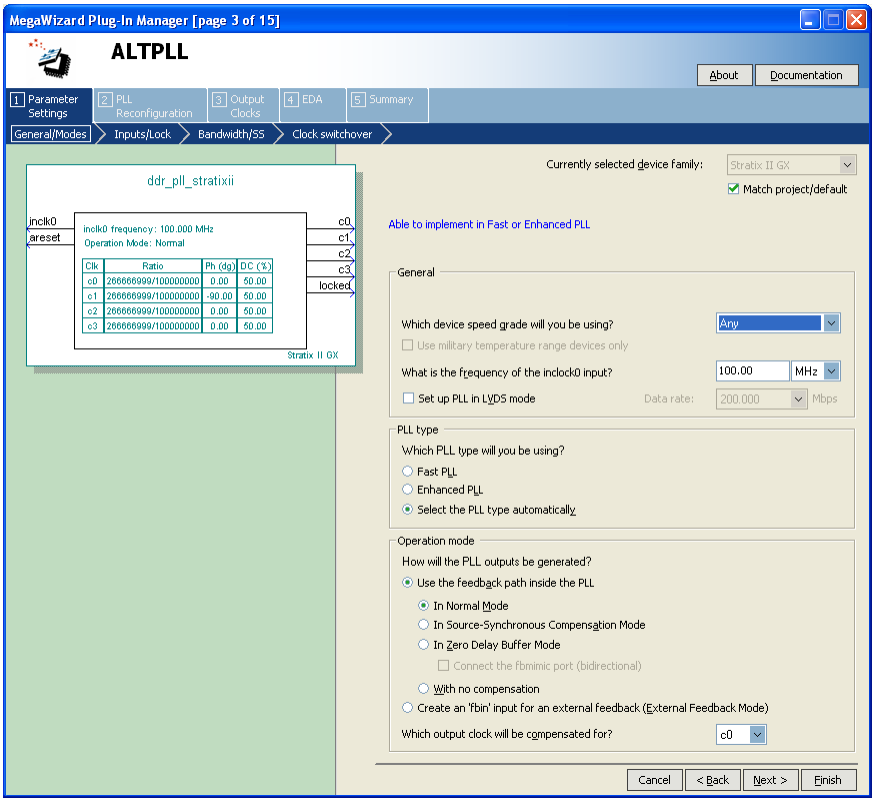
Figure 36. DDR2 SDRAM MegaWizard Generation Report



The DDR2 SDRAM MegaWizard Plug-In Manager also creates an example driver design with the same name as your project name (**Legacy_PHY.v** in the design example) such that you can compile and simulate the design to verify functionality before integrating it with the rest of your design.

20. Change the example driver to meet the requirements of your application.
21. Change the PLL input frequency if it is not of the same frequency of the interface. The Stratix II GX PCI Express (PIPE) Development Board is equipped with a 100-MHz oscillator for the memory interface, so you must change the PLL input frequency by invoking the MegaWizard Plug-In Manager to edit the **ddr_pll_stratixii** module, as shown in Figure 37.
22. Click **Finish**.

Figure 37. Changing the PLL Input Clock Frequency



23. Because the fedback PLL inputs are generated by the system PLL, you must add some user logic that will toggle the `areset` pin of the fedback PLL after the system PLL is locked. This is to ensure that the fedback PLL gets a clean input clock signal when it is locked. However, the design example does not have this extra logic.

For more information about instantiating a memory controller using the legacy PHY, refer to the *DDR and DDR2 SDRAM High-Performance Controller and ALTMEMPHY User Guide*.

Step 3: Add the Constraints

The IP Tool bench also creates a **.tcl** script called **auto_add_ddr_constraints.tcl** that constrains resynchronization register locations, DQS and DQ pin locations, I/O standards, output loads, and output enable groups. After sourcing the **.tcl** script, you also need to run DTW to time-constrain the design. You can choose either the Classic timing analyzer or TimeQuest timing analyzer when using DTW. However, DTW constraints using the TimeQuest timing analyzer give more accurate compilation results as the constraints apply for both timing models instead of just one corner (slow or fast timing model). This section describes how to add these constraints step-by-step.

The MegaWizard Plug-In Manager creates a **verify_timing_for_<*variation_name*>.tcl** file to report memory interface timing. However, Altera recommends you use DTW because it is more accurate and more flexible to use. The **report_timing.tcl** file makes certain assumptions that may not pertain to your design and does not support the TimeQuest timing analyzer.

For more information about using DTW and its companion timing analysis script, refer to the *DDR Timing Wizard (DTW) Megafunction User Guide*.

You must add the appropriate pin location and pin loading assignments, termination, and drive strength to the memory interface signals in your design, in addition to timing and device constraints for the rest of your design.

To determine which drive strength and termination to use, refer to *"Stratix II, Stratix II GX, and Arria GX On Chip Termination (OCT)" on page 5.* For more information about memory interface signals, go to "Appendix D: Interface Timing Analysis" on page 108.

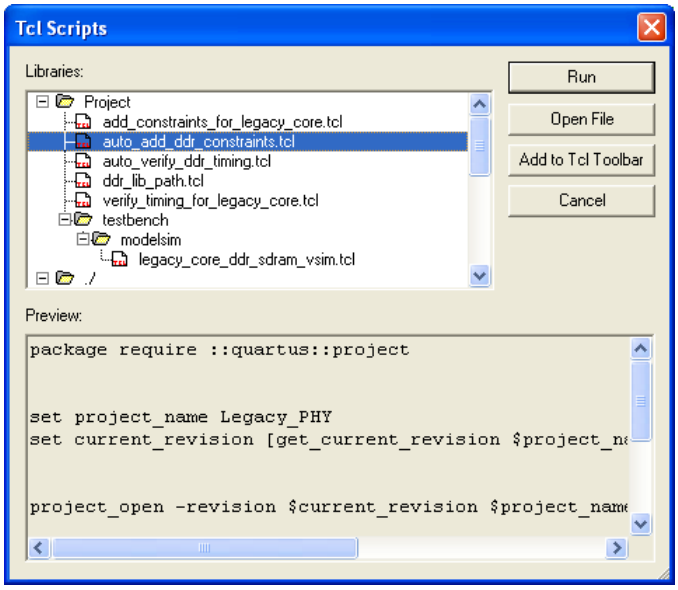
To have a reliable design running at the desired performance, you must constrain the design properly. The following steps guide you through all the constraints needed for the memory interface:

1. Open the **Legacy_PHY.v** file to ensure that the Quartus II software is pointing to the project directory.

If you previously opened a file from a different directory other than the project directory, the **auto_add_ddr_constraints.tcl** script might fail as the Quartus II software is not correctly pointing to the right directory. Opening the design top-level file ensures that the Quartus II software is pointing to the correct directory.

2. Source the **auto_add_ddr_constraints.tcl** script generated by the DDR2 SDRAM controller MegaWizard Plug-In Manager. To locate the file, open the Tools menu and select **Tcl Scripts**.
3. In the **Tcl Scripts** dialog box, highlight the **auto_add_ddr_constraints.tcl** file, as shown in Figure 38.
4. Click **Run** to add constraints to the design. The script automatically performs analysis and elaboration before adding constraints to the design.

Figure 38. Adding the IP Tool Bench Constraint



CAUTION

The **auto_add_ddr_constraints.tcl** file contains the default I/O assignments including I/O standard, pin location, and output loading settings. If your design is using non-default settings, source this **.tcl** file before applying your custom settings because this **.tcl** file overwrites pre-existing assignments in your Quartus II project. In addition, if you make a change to the settings set by this **.tcl** file, in the **Project Settings** page of the DDR2 SDRAM MegaWizard Plug-In Manager, turn off the **Automatically apply datapath specific constraints to the Quartus II project** option the next time you regenerate the controller.

The **auto_add_ddr_constraints.tcl** script file is the only **.tcl** file that you need from the DDR2 SDRAM controller MegaWizard Plug-In Manager. However, the MegaWizard Plug-in Manager creates the following **.tcl** files (in addition to **auto_add_ddr_constratints.tcl**) per variation:

- **add_constraints_for_legacy_core.tcl**—The **auto_add_ddr_constraints.tcl** calls this file which has the actual constraints for the controller.
- **auto_verify_ddr_timing.tcl**—This script calls **verify_timing_for_<*variation_name*>.tcl.**
- **ddr_lib_path.tcl**—This script contains the controller library path in the Quartus II installation directory.
- **remove_add_constraints_for_<*variation_name*>.tcl**—Run this script if you want to start the design with new assignments.
- **verify_timing_for_<*variation_name*>.tcl**—This is the MegaWizard-generated script for the memory interface timing report. Use DTW and its companion script **dtw_timing_analysis.tcl** instead.

If you get an error message similar to the one shown below, on the Processing Menu, point to **Start** and select **Start Analysis & Elaboration** to perform analysis and elaboration before sourcing the **.tcl** script.

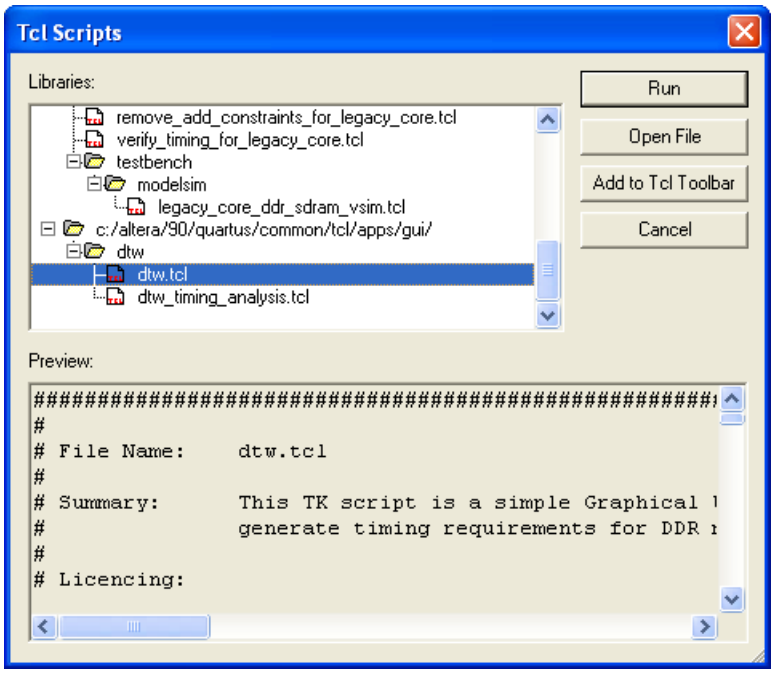
```
Error: Cannot run Tcl Script File
"C:\AN328\Legacy_PHY\auto_add_ddr_constraints.tcl"

Error: ERROR: Project does not exist or has illegal name
characters: Legacy_PHY. Specify a legal project name.
```

The **auto_add_ddr_constraints.tcl** script includes placement constraints for the resynchronization registers. It also contains the I/O standard, output loading, output enable grouping, and pin location assignments for the DQS and DQ I/O pins.

Run DTW to time-constrain the rest of the interface by following these steps:

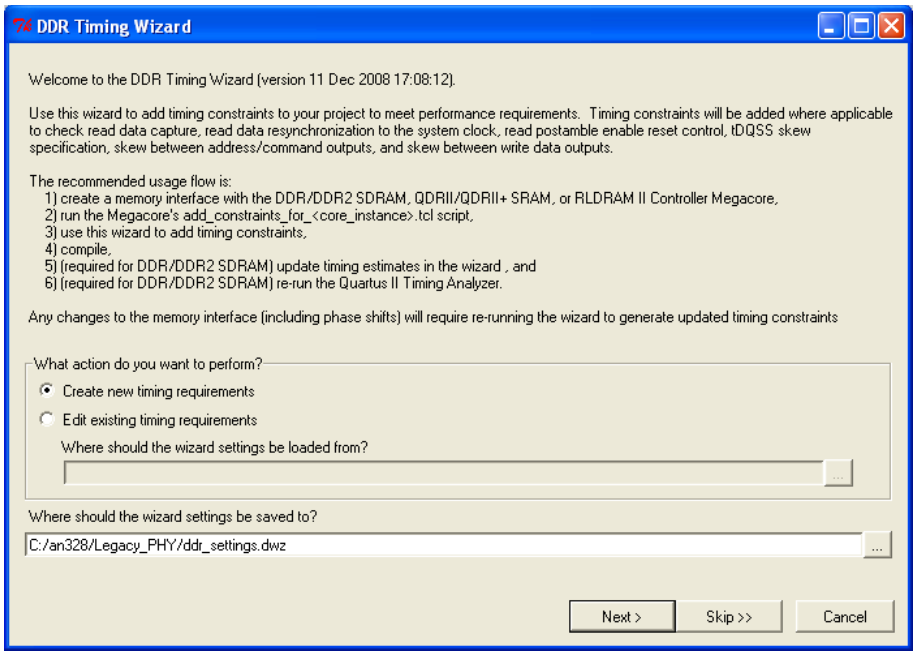
1. On the Tools menu, select **Tcl Scripts.** In the **Tcl Scripts** dialog box, select **DTW**, as shown in Figure 39.

Figure 39. Invoking DTW



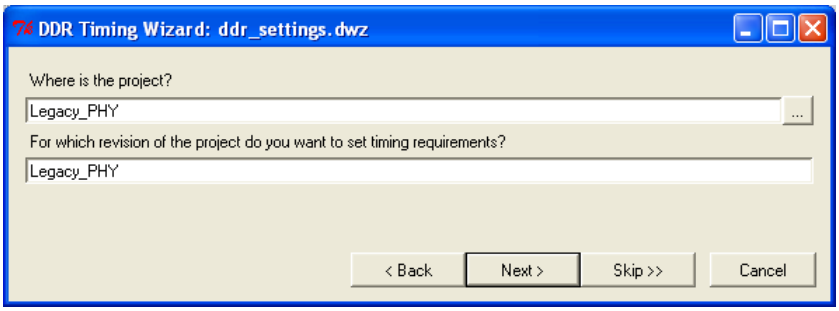
2. Pick a file name with a **.dwz** extension to store the settings and click **Next**. The default name for the file is **ddr_settings.dwz**, as shown in Figure 40.

Figure 40. Creating a New DTW File



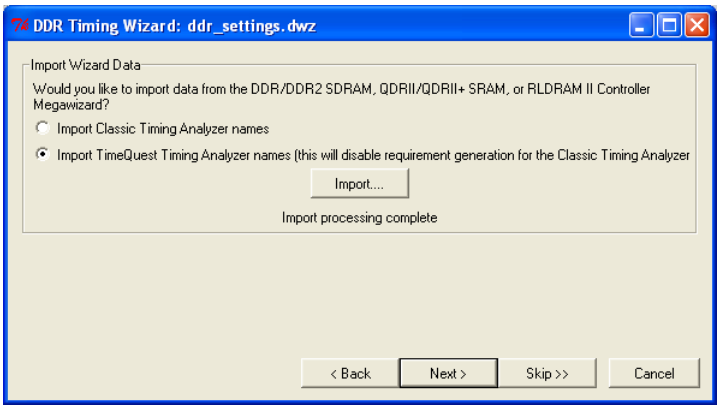
3. Ensure that the project and project revision names are correct on the next page, as shown in Figure 41. Click **Next.**

Figure 41. Selecting Project and Project Revision Names



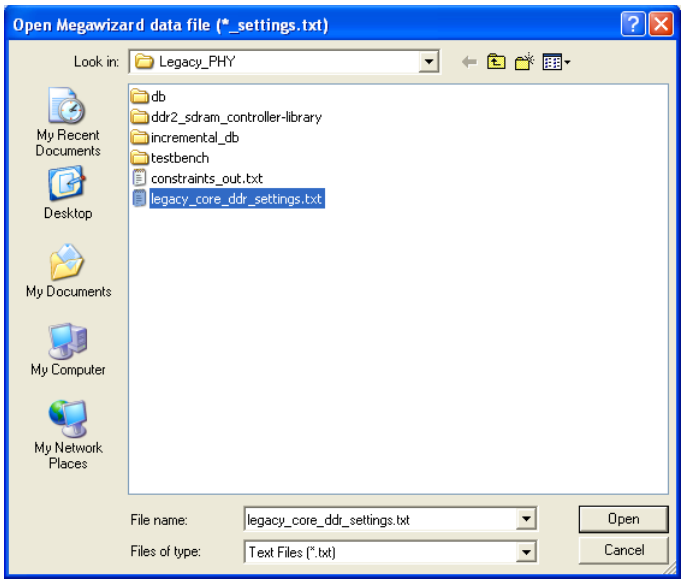
4. Select whether you are going to import the MegaCore function settings using the Classic timing analyzer or TimeQuest timing analyzer names, as shown in Figure 42. This design example uses the TimeQuest timing analyzer, which is the recommended timing analyzer, as the DTW-generated **.sdc** constraints apply for both fast and slow timing models.

Figure 42. Importing Settings from the IP Tool Bench



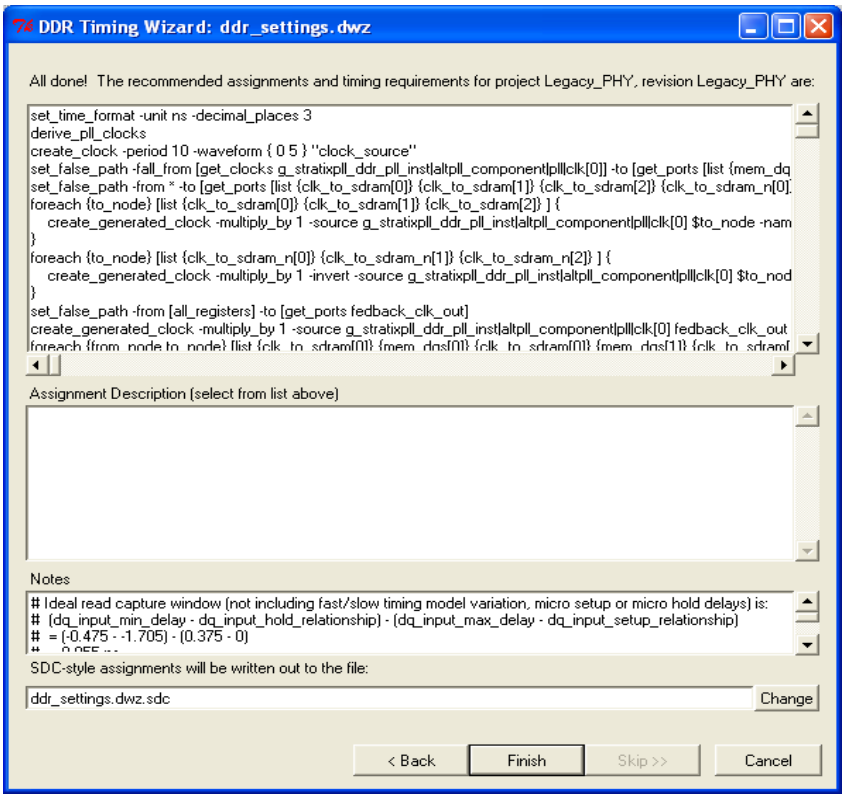
5. Click **Import** and select **legacy_core_ddr_setting.txt**, which contains all the information that was entered in the DDR2 SDRAM controller MegaWizard Plug-In Manager.
6. Click **Open** to import the settings into DTW (Figure 43).

Figure 43. Importing the DDR Settings to the DTW



7. After the import process is complete, click **Next** through each subsequent page, while confirming that all the information in the DTW is correct.
8. The last page of the DTW is shown in Figure 44. Click **Finish**.

Figure 44. Last Page of the DTW



9. In the **Assignment Editor**, add other assignments to the design.

For more information about adding assignments to your design, and the DTW in general, refer to the *DDR Timing Wizard (DTW) User Guide*.

CAUTION

You must disable the MegaWizard Plug-in Manager from creating the constraints file if you make any changes to it, then regenerate the memory controller, otherwise these changes will get overwritten in the next compile.

10. Make these changes to the constraints file:
    a. Modify the I/O standard settings for the DQS and DQ pins from SSTL-18 Class II (this was set by the **auto_add_ddr_constraints.tcl** script) to SSTL-18 Class I, as the board uses Class I termination.
    b. Change the **Output Pin Load** for the `fedback_clk_out` pin to **6**. This is necessary because this signal goes to the `CLK6p` pin of the FPGA which has 6 pF loading, instead of going to the memory clock pin which has 2 pF loading. The constraints show 4 pF as set in the MegaWizard Plug-in Manager.
    c. Similarly, change the **Output Pin Load** for the `clk_to_sdram[2]` and `clk_to_sdram_n[2]` pins to **2**. These pins are only connected to the single ×8 memory device instead of two, such as the `clk_to_sdram[1:0]` and `clk_to_sdram_n[1:0]` pins, which go to two ×16 memory devices each, as indicated during the PHY and controller instantiation step.
    d. Change the I/O standard for the `clock_source` pin to **LVDS** to match the settings on the board.
    e. Set pin location assignments for the `clock_source`, CK/CK#, fedback clock input, feedback clock output, address, and command pins. For the pin assignments for the Stratix II GX PCI Express Development Board, refer to "Appendix A: Stratix II GX PCI-Express Development Board Pin Assignments" on page 92.

☞ You can set unused pins as inputs tri-stated with a weak pull-up resistor to ensure that those pins are not floating on the board. Set this assignment from the Assignment menu on the **Device and Pin Options** page in the **Device** window under **Settings**.

    f. Change the DQ pin assignments to match the board. The DQ pin assignments generated by the MegaWizard Plug-In Manager does not match the pin assignment on the board, so you need to fix this manually. For the actual pin assignments for the Stratix II GX PCI Express (PIPE) Development Board, refer to "Appendix A: Stratix II GX PCI-Express Development Board Pin Assignments" on page 92.

    ☞ Changing the DQ pin locations mean that the resynchronization registers' location assignments may not be valid any longer. You can fix this by running the **relative_constraint.tcl** script that is available with the design example, but to use it, you must compile the design first. For more information, refer to "Step 6: Adjust the Constraints" on page 76.

g. Add reserve pin assignments for the `ddr2_a[13]`, `ddr2_a[14]`, and `ddr2_ba[2]` pins to the **As output driving ground** option. These pins are bonded on the board but are not used in the design.

h. Add I/O standard assignments for the `pnf`, `reset_n`, `test_complete`, `ddr2_a[13]`, `ddr2_a[14]`, and `ddr2_ba[2]` pins to **SSTL-18 Class I**. You can also set the default voltage for the project to **1.8 V** by opening the Assignment menu, selecting the **Settings** page (**Device and Pin Options** in the **Device** folder) and making your changes there.

i. Set the termination and drive strength for the memory interface pins. The **auto_add_ddr_constraints.tcl** assigns all uni-directional pins to use the **Series 50 Ω without Calibration** and the bi-directional pins to use the **Series 50 Ω without Calibration termination** options. However, you can only use **Series 50 Ω without Calibration** for the memory interface pins (including the DQS and DQ pins) with this board.

   For the design example, you must disable the **Series 25 Ω without Calibration termination** settings for the DQS and DQ pins because these pins are using Class I termination on the board. Furthermore, you cannot set these pins to have the **Series 50 Ω without Calibration** termination setting as the resultant drive strength from this setting is not enough for 267-MHz operation.

j. In the **Assignment Editor**, set the **Delay from Output Register to Output Pin** option for the `clk_to_sdram*` outputs and fedback clock outputs to **0**. This is to disallow the Quartus II software from adding output delay chains to the pins.

k. Save the Assignment Editor file.

11. On the Assignments menu, click **Settings** and complete the following:

a. Under **Fitter settings**:

- Under the **Optimize hold timing** option, choose **All Paths** because you are using the TimeQuest timing analyzer. Turn off the **Optimize hold timing** option if you are using the Classic timing analyzer because having this option turned on sometimes yields incorrect phase shift results when using **dtw_timing_analysis.tcl**.
- Ensure that the **Optimize fast corner timing** option is unchecked, whether you are using the TimeQuest or Classic timing analyzer. The design example uses **.sdc** constraints that are already optimized for both ends of the timing model.

b. Under the **Timing Analysis Settings** section, choose **Use TimeQuest Timing Analyzer** during compilation.

c. Under **Timing Analysis Settings** in the TimeQuest timing analyzer sub-option, add the DTW-generated **.sdc** file **ddr_settings.dwz.sdc**.

d. Click **OK**.

When your design is properly constrained, you are ready to compile the design.

Step 4: Perform the RTL/Functional Simulation (Optional)

For performing a functional simulation of your memory interface, use the functional models generated by the MegaWizard Plug-In Manager. You must use this model in conjunction with your own driver or the MegaWizard testbench that issues read and write operations, and a memory model.

The Verilog HDL or VHDL simulation model of the PHY is found in your project directory. For the design example, the file is named **Legacy_PHY.vo**. During controller generation, the DDR2 SDRAM controller also creates a subdirectory called testbench, which contains the testbench file (**Legacy_PHY_tb.v**) and a folder called **ModelSim**. The **ModelSim** folder contains a **.tcl** file (**legacy_core_ddr_sdram_vsim.tcl**) and a **wave.do** file to run simulations in the ModelSim Altera software.

The design example here is unique in that it uses two different types of DQS modes. The testbench assumes that nine ×8 memory devices are used as if you are interfacing with a 72-bit DIMM for all designs. Some modifications described below are needed to simulate the design example properly.

Get the Memory Simulation Model

1. Download the simulation model of the memory type that you selected from the memory vendor's website for the project into the <*project_directory*>**\testbench** directory. The model name for the MT47H32M16CC-3 and MT47H64M8CB-3 DDR2 SDRAM components is called **ddr2.v**.
2. Copy the parameter file **ddr2_paremeters.vh** into the <*project_directory*>**\testbench\modelsim** directory.

Prepare the Simulation Model

1. Open the **ddr2.v** file in a text editor and set the following define statements at the top of the file:

```
`define sg3
`define ×8
```

   The first line defines the memory speed grade; the second line defines the memory device width. Even though four of the five devices used in the design are ×16 DDR2 SDRAM components, the steps below allude that nine ×8 DDR2 SDRAM components are used.

The design example downloadable with this application note offers both workarounds for the testbench file. The **Legacy_PHY_tb.v** testbench file in the design example uses nine ×8 DDR2 SDRAM models. You cannot perform the simulation with both ×16 and ×8 DDR2 SDRAM models. In addition, because 72 is not divisible by 16, you cannot use the ×16 DDR2 SDRAM models because the width interface does not match. If you use two ×8 DDR2 SDRAM components, you may use the ×16 DDR2 SDRAM models.

2. Save the **ddr2.v** file.

Instantiate the Memory Model in the Testbench

1. From the **\testbench** directory, open **Legacy_PHY_tb.v** in a text editor.
2. Locate the line `generic_ddr2_sdram_rtl #`, which is the first instantiation of the memory device, as shown below:

```
generic_ddr2_sdram_rtl # (

        .BANKBITS (2),
        .ROWBITS  (13),
        .COLBITS  (10),
        .DATABITS (8)

    ) memory_0_0 (
      .DQ (mem_dq[ 8* (0+1) - 1 :  8 * 0]),
      .DQS (mem_dqs[0]),
      .ADDR (a_delayed[13-1: 0]),
      .BA (ba_delayed),
      .CLK (clk_to_ram),
      .CLK_N (clk_to_ram_n),
      .CKE (cke_delayed[0]),
      .CS_N (cs_n_delayed[0]),
      .RAS_N (ras_n_delayed),
      .CAS_N (cas_n_delayed),
      .WE_N (we_n_delayed),
      .DM_RDQS (dm_delayed[0]),
      .ODT (odt_delayed),
      .RDQS_N (),
      .DQS_N ()
   );
```

3. The testbench assumes that the memory model used is named **generic_ddr2_sdram_rtl.v**. For this design, replace **generic_ddr2_sdram_rtl** with the `ddr2` for the nine memory device instantiations to match the memory model prepared in step 1.
4. Ensure that the port names in the memory model match the memory device port names in the testbench.

In Verilog HDL, the names are case-sensitive, which is not the case in VHDL. The **ddr2.v** file uses lower-case signal names—change the signal names in the module instantiations in the testbench file that have upper-case names. For more information, refer to "Modify the System PLL File" on page 71.

The ddr2 module in the design example uses `ck` and `ck_n` ports instead of `clk` and `clk_n` ports. Change the signal names to match the port names in the ddr2 module.

If you are using a ×16 DDR2 SDRAM model for your actual design, change the DQ, DQS, and DM indices in the module instantiation as shown in this example:

```
ddr2 memory_0_0 (
        .dq      (mem_dq[mem_dq[8*(0+1)- 1: 8*0]]),
        .dqs     (mem_dqs[0]),
        .addr    (a_delayed[13-1: 0]),
        .ba      (ba_delayed),
        .ck      (clk_to_ram),
        .ck_n    (clk_to_ram_n),
        .cke     (cke_delayed[0]),
        .cs_n    (cs_n_delayed[0]),
        .ras_n   (ras_n_delayed),
        .cas_n   (cas_n_delayed),
        .we_n    (we_n_delayed),
        .dm_rdqs (dm_delayed[0]),
        .odt     (odt_delayed),
        .rdqs_n  (),
        .dqs_n   ()
    );
```

5. Look for the following line and change the period of the clock to **10000**:

```
parameter REF_CLOCK_TICK_IN_PS  = 3750;
//edit if you change your PLL reference clock frequency
```

6. Save the testbench.

Modify the System PLL File

1. Open the **ddr_pll_stratixii.v** file located in the project directory.
2. Comment out output port c3 in the port list and the port declaration portion of the file. The changes are shown in this example:

```
module ddr_pll_stratixii (
    inclk0,
    c0,
    c1,
    c2,
    //c3
    );

    input  inclk0;
    output  c0;
    output  c1;
    output  c2;
    //output  c3;
```

This port is created by the DDR2 SDRAM controller MegaWizard Plug-in Manager for 1-PLL mode operation but is not used in this design example so it does not affect design compilation. You must comment out the code or the ModelSim software will give you a missing port error.

Use this PLL output for the address and command clock in "Step 6: Adjust the Constraints" on page 76.

3. Save the file.

Update the wave.do File

1. Open the **wave.do** file in the <*project_directory*>**\testbench\modelsim** directory.
2. Change the following code:

```
add wave -noupdate -format Logic -radix hexadecimal
/${testbench_name}/dut/resynch_clk
```

to

```
add wave -noupdate -format Logic -radix hexadecimal
/${testbench_name}/dut/fedback_resynch_clk
```

The default **wave.do** file wrongly names the resynchronization clock. This is a generic file that is also used for 1-PLL mode simulation.

When using 1-PLL mode, the default `resynch_clk` signal is called from the device under the test (DUT) module, when it is supposed to be called one level below that. In this implementation, change the line to:

```
add wave -noupdate -format Logic -radix
hexadecimal /${testbench_name}/dut/legacy_core_ddr_sdram/
resynch_clk
```

3. Save the **wave.do** file.

Perform the Simulation in the ModelSim Software

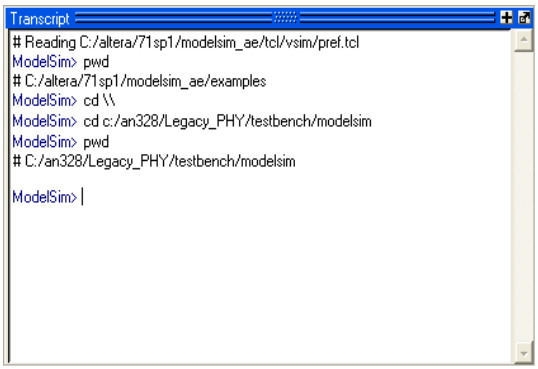
To use the NativeLink feature, follow the instructions in the *Using the NativeLink Feature with ModelSim* section volume 3 of the *Quartus II Handbook*.

1. Ensure that ModelSim is installed on your system and invoke the ModelSim simulator.

You can use either the ModelSim-Altera edition or the full version of the ModelSim simulator.

2. Change the directory in the ModelSim Transcript window to <*project_directory*>`/testbench/modelsim`, as shown in Figure 45.

Figure 45. Changing the Directory in ModelSim



3. Set the memory model used for this simulation by entering the following:

```
set memory_model ddr2
```

4. On the Tools menu, click **Execute Macro** and select **legacy_core_ddr_sdram_vsim.tcl**.

You can also type:

```
do legacy_core_ddr _sdram_vsim.tcl
```

The simulator runs until the time elapsed is ~204 µs with the ModelSim Transcript window showing that simulation passed (Figure 46). The first 200 µs of the simulation initializes the memory.

Figure 46. End of Simulation in ModelSim Transcript Window

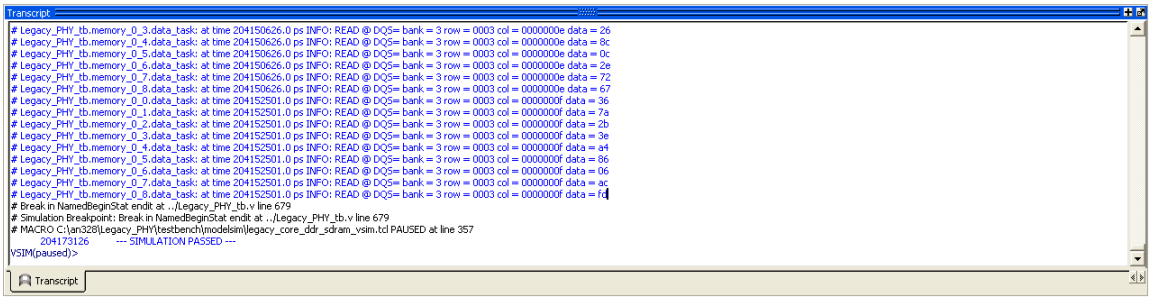
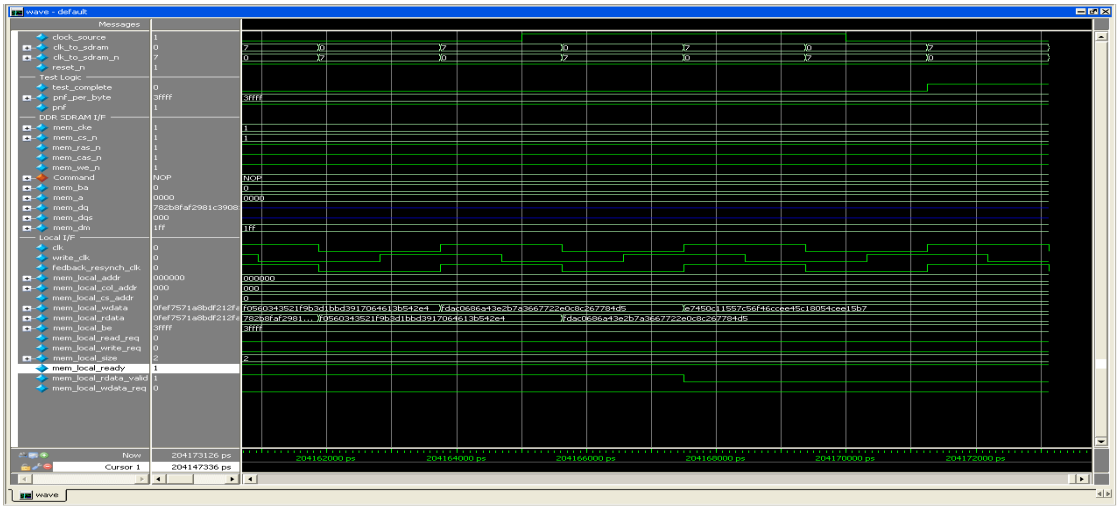


Figure 47 shows the last read and write transactions that occurred near the end of the simulation.

Figure 47. Read and Write Transactions in ModelSim



Step 5: Compile the Design and Generate the Timing Report

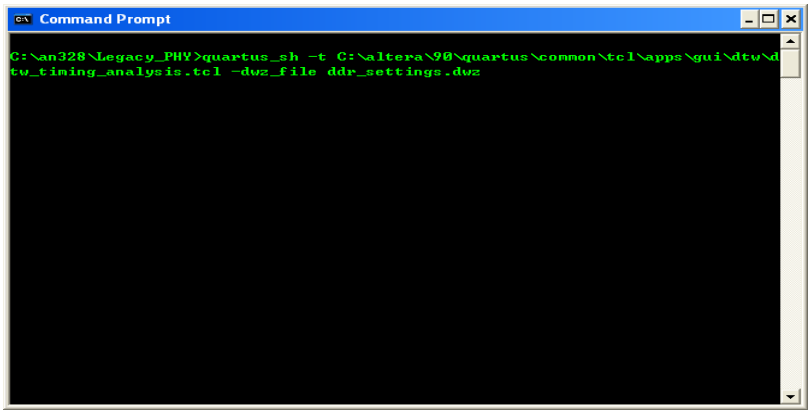
Compile the fully constrained design. After successfully compiling your design in the Quartus II software, run the **dtw_timing_analysis.tcl** script from the command prompt to analyze interface timing (Figure 48). This script is available in the **<***quartus_installation_directory***>\quartus\common\tcl\apps\gui\dtw** folder. You can either copy the script into your project directory or access the script from its default location. You must however, run the script from your project directory in the command prompt.

The command to call the **dtw_timing_analysis.tcl** script from the default Quartus II 9.0 installation directory is as follows (provided that you are using **ddr_settings.dwz** file for your design):

```
quartus_sh -t
c:\altera\90\quartus\common\tcl\apps\gui\dtw\
dtw_timing_analysis.tcl -dwz_file ddr_settings.dwz
```

The script must know which **.dwz** settings to analyze, so you must specify them manually using the command line. There are other optional arguments for the script, which are documented in the *DDR Timing Wizard (DTW) User Guide*.

Figure 48. Sourcing the dtw_timing_analysis.tcl Script



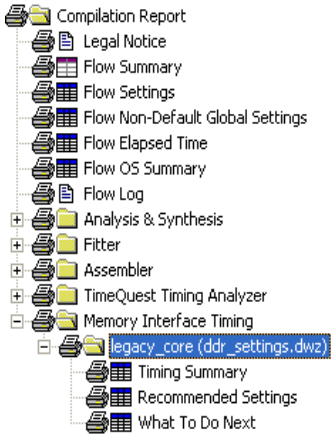
CAUTION

You cannot run the **dtw_timing_analysis.tcl** script from the **Tcl Scripts** option under the Tools menu.

After the **dtw_timing_analysis.tcl** script has finished running, close the compilation report and reopen it to display the script results.

The **dtw_timing_analysis.tcl** script results are added at the bottom of the compilation report in the **Memory Interface Timing** folder (Figure 49). This folder has a subfolder named **legacy_core (ddr_settings.dwz)**, where **legacy_core** is the name of the MegaCore controller and **ddr_settings.dwz** is the name of the **.dwz** file used for the analysis.

Figure 49. The dtw_timing_analysis.tcl Script Results



There are three panels under the subfolders named **Timing Summary, Recommended Settings,** and **What To Do Next**, shown in Figure 50 through Figure 52.

Figure 50. Design Example Timing Summary

Timing Summary

	Clock	Current Margin (ns)	Ideal Margin (ns)	Slow Setup (ns)	Slow Hold (ns)	Fast Setup (ns)	Fast Hold (ns)	PLL Name
1	Read capture	0.276	0.289	0.302	0.276	0.377	0.295	
2	Fed-back clock	-3.388	0.100	-3.388	4.336	-2.212	3.588	g_stratixpll_ddr_fedback_pll_inst|altpll_component|pll|clk[0]
3	Resynchronization clock	0.064	0.926	0.064	2.728	1.199	1.788	g_stratixpll_ddr_pll_inst|altpll_component|pll|clk[0]
4	Postamble clock	0.223	1.168	0.727	2.112	0.223	2.777	g_stratixpll_ddr_fedback_pll_inst|altpll_component|pll|clk[1]
5	Recovery/Removal	0.394	0.619	0.845	0.585	1.205	0.394	
6	tDQSS	0.365	0.689	1.012	0.389	1.288	0.365	g_stratixpll_ddr_pll_inst|altpll_component|pll|clk[0]
7	Write Capture	0.150	0.256	0.150	0.361	0.360	0.361	g_stratixpll_ddr_pll_inst|altpll_component|pll|clk[1]
8	Address/Command	0.712	0.946	0.712	1.335	0.910	1.179	g_stratixpll_ddr_pll_inst|altpll_component|pll|clk[0]

As shown in the Timing Summary in Figure 50, the slow setup margin for the fedback clock timing shows a violation of over one clock period. This indicates that the clock cycle selection for the resynchronization path is incorrect, as shown in the recommended settings in Figure 51.

Remember also that the DQS/DQ pin assignments were changed, resulting in the resynchronization register locations not being optimally placed. These location assignments are fixed in "Step 6: Adjust the Constraints" on page 76.

Figure 51. Design Example Recommended Settings

Recommended Settings

	Clock	Current Clock Cycle	New Clock Cycle	Current Phase	New Phase	PLL Name
1	Fed-back clock	2	2	0	335	g_stratixpll_ddr_fedback_pll_inst|altpll_component|pll|clk[0]
2	Resynchronization clock	3	4	0	58	g_stratixpll_ddr_pll_inst|altpll_component|pll|clk[0]
3	System postamble clock	3	3	-180	0	g_stratixpll_ddr_pll_inst|altpll_component|pll|clk[0]
4	Postamble clock	2	2	90	112	g_stratixpll_ddr_fedback_pll_inst|altpll_component|pll|clk[1]
5	CK/CK#	N/A	N/A	0	-31	g_stratixpll_ddr_pll_inst|altpll_component|pll|clk[0]
6	Write Capture	N/A	N/A	-90	-80	g_stratixpll_ddr_pll_inst|altpll_component|pll|clk[1]
7	Address/Command	N/A	N/A	0	-22	g_stratixpll_ddr_pll_inst|altpll_component|pll|clk[0]

The **What To Do Next** panel in Figure 52 provides high-level suggestions about what you must do to balance your timing margins.

Figure 52. What To Do Next

What To Do Next

	What To Do Next
1	⊟ You should change the Optimize Hold Timing assignment because clock phase calculations may be incorrect.
2	Turn off Optimize Hold Timing while you close timing on this memory interface.
3	Make this change, recompile your design, and rerun this script.
4	Adjust the clock cycles as recommended.
5	⊟ Choose one of the following options:
6	a) Rerun this script and add the -auto_adjust_cycles option.
7	b) Open DTW, update the clock cycles manually, then rerun this script with the same options.
8	These options do not change clock cycle settings in the IP Toolbench
9	⊟ If necessary, update clock cycle settings for these clocks in the IP Toolbench:
10	Resynchronization clock and postamble clock

For more information about how to change the phase shifts of the postamble, CK/CK#, and address and command clocks, refer to the *DDR Timing Wizard (DTW) User Guide.*

You must not adjust the phase shift of the system clock or the write clock, which defaults to the `pllclk[0]` and `pllclk[1]` signals. Instead, if another path uses either of these clocks and requires phase-shift adjustment, change the connection and use dedicated PLL outputs.

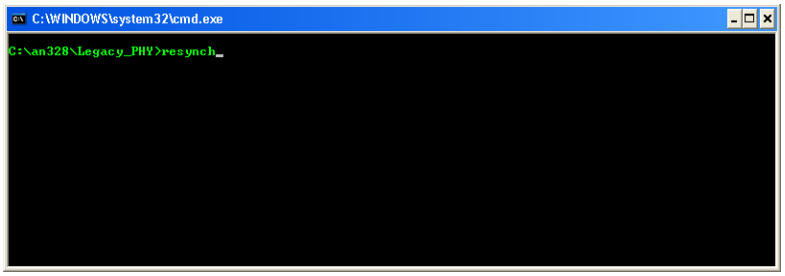
Because the design does not meet timing, you must adjust the design constraints.

Step 6: Adjust the Constraints

Before using the **dtw_timing_analysis.tcl** recommendations, fix the resynchronization registers' location assignments to ensure that these registers are optimally placed.

1. Run the **resynch.bat** batch file in the command editor pointing to your project directory (Figure 53) to fix the resynchronization registers' location assignments.

Figure 53. Running a Batch File to Fix the Resynchronization Registers' Location Assignments



The **resynch.bat** batch file runs **relative_constraint.tcl** in the background. The batch file groups the two groups of resynchronization registers with each DQ pin and places the registers one row above the pins. The following is an example of the code in the batch file for DQS/DQ group 0:

```
quartus_sh -t relative_constraint.tcl -project Legacy_PHY -
pin_name *ddr2_dq[* -reg_name "*0:*|resynched_data[*]" -
show_regs -reg_range 7:0 -pin_range 7:0 -row_offset 1 -apply
```

```
quartus_sh -t relative_constraint.tcl -project Legacy_PHY -
pin_name *ddr2_dq[* -reg_name "*0:*|resynched_data[*]" -
show_regs -reg_range 15:8 -pin_range 7:0 -row_offset 1 -apply
```

For more information about **relative_constraint.tcl**, refer to "Appendix E: The relative_constraint.tcl Script" on page 134.

2. After running the batch file, compile the design and rerun the **dtw_timing_analysis.tcl** script. Then, follow the recommended phase shift for the resynchronization, postamble, and write clocks from the **dtw_timing_analysis.tcl** script and recompile the design to close timing.

   Figure 54 and Figure 55 show the new timing analysis results and recommendation after the resynchronization registers' location assignments are fixed.

Figure 54. Timing Summary with Optimized Resynchronization Registers' Locations

Timing Summary

	Clock	Current Margin (ns)	Ideal Margin (ns)	Slow Setup (ns)	Slow Hold (ns)	Fast Setup (ns)	Fast Hold (ns)	PLL Name
1	Read capture	0.276	0.289	0.302	0.276	0.377	0.295	
2	Fed-back clock	-3.398	0.095	-3.398	4.338	-2.208	3.587	g_stratixpll_ddr_fedback_pll_inst|altpll_component|pll|clk[0]
3	Resynchronization clock	0.064	0.926	0.064	2.727	1.201	1.788	g_stratixpll_ddr_pll_inst|altpll_component|pll|clk[0]
4	Postamble clock	0.242	1.177	0.752	2.111	0.242	2.777	g_stratixpll_ddr_fedback_pll_inst|altpll_component|pll|clk[1]
5	Recovery/Removal	0.395	0.590	0.785	0.582	1.174	0.395	
6	tDQSS	0.365	0.689	1.012	0.389	1.288	0.365	g_stratixpll_ddr_pll_inst|altpll_component|pll|clk[0]
7	Write Capture	0.150	0.256	0.150	0.361	0.360	0.361	g_stratixpll_ddr_pll_inst|altpll_component|pll|clk[1]
8	Address/Command	0.712	0.946	0.712	1.335	0.910	1.179	g_stratixpll_ddr_pll_inst|altpll_component|pll|clk[0]

Figure 55. Recommended Settings with Optimized Resynchronization Registers Locations

Recommended Settings

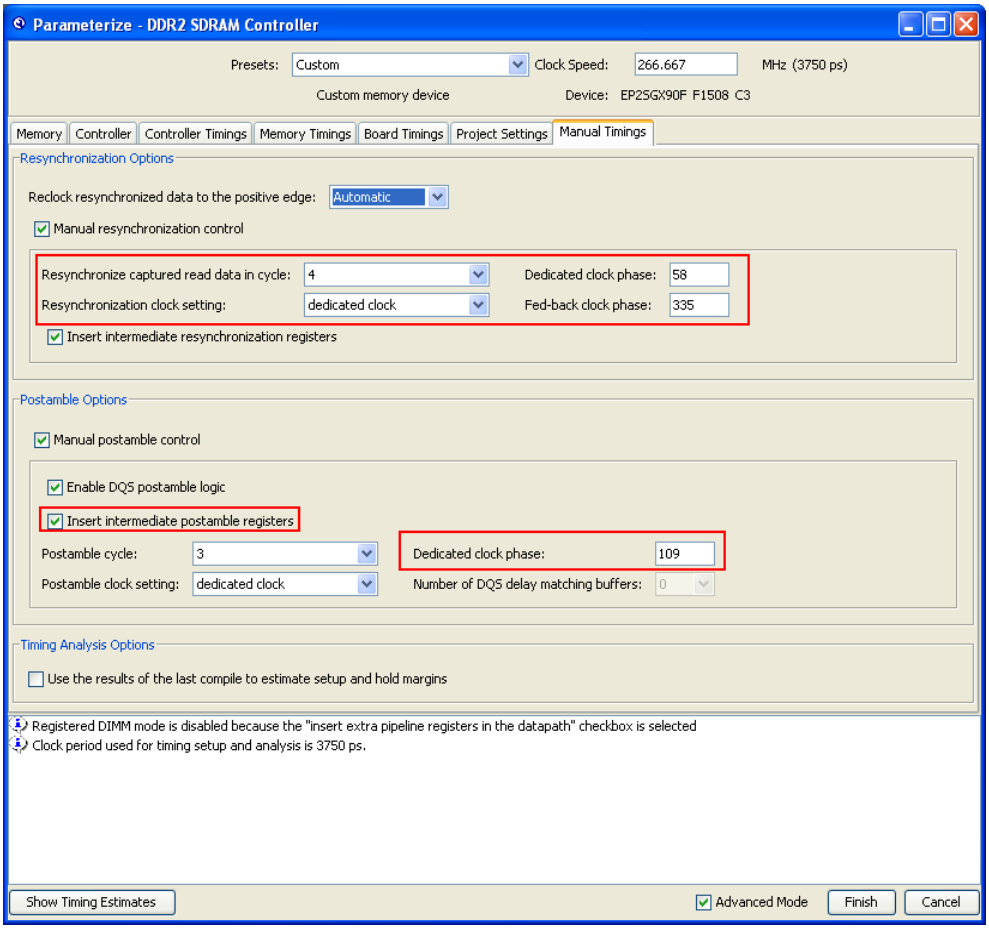
	Clock	Current Clock Cycle	New Clock Cycle	Current Phase	New Phase	PLL Name
1	Fed-back clock	2	2	0	335	g_stratixpll_ddr_fedback_pll_inst|altpll_component|pll|clk[0]
2	Resynchronization clock	3	4	0	58	g_stratixpll_ddr_pll_inst|altpll_component|pll|clk[0]
3	System postamble clock	3	3	-180	0	g_stratixpll_ddr_pll_inst|altpll_component|pll|clk[0]
4	Postamble clock	2	2	90	109	g_stratixpll_ddr_fedback_pll_inst|altpll_component|pll|clk[1]
5	CK/CK#	N/A	N/A	0	-31	g_stratixpll_ddr_pll_inst|altpll_component|pll|clk[0]
6	Write Capture	N/A	N/A	-90	-80	g_stratixpll_ddr_pll_inst|altpll_component|pll|clk[1]
7	Address/Command	N/A	N/A	0	-22	g_stratixpll_ddr_pll_inst|altpll_component|pll|clk[0]

For more information about how to close timing with DTW, refer to the "Timing Closure Process" section of the *DDR Timing Wizard (DTW) User Guide*.

3. On the **Manual Timing** page, change both the clock cycle and phase shift of the fedback resynchronization clock. Perform the changes in the DDR2 SDRAM controller MegaWizard Plug-In Manager.

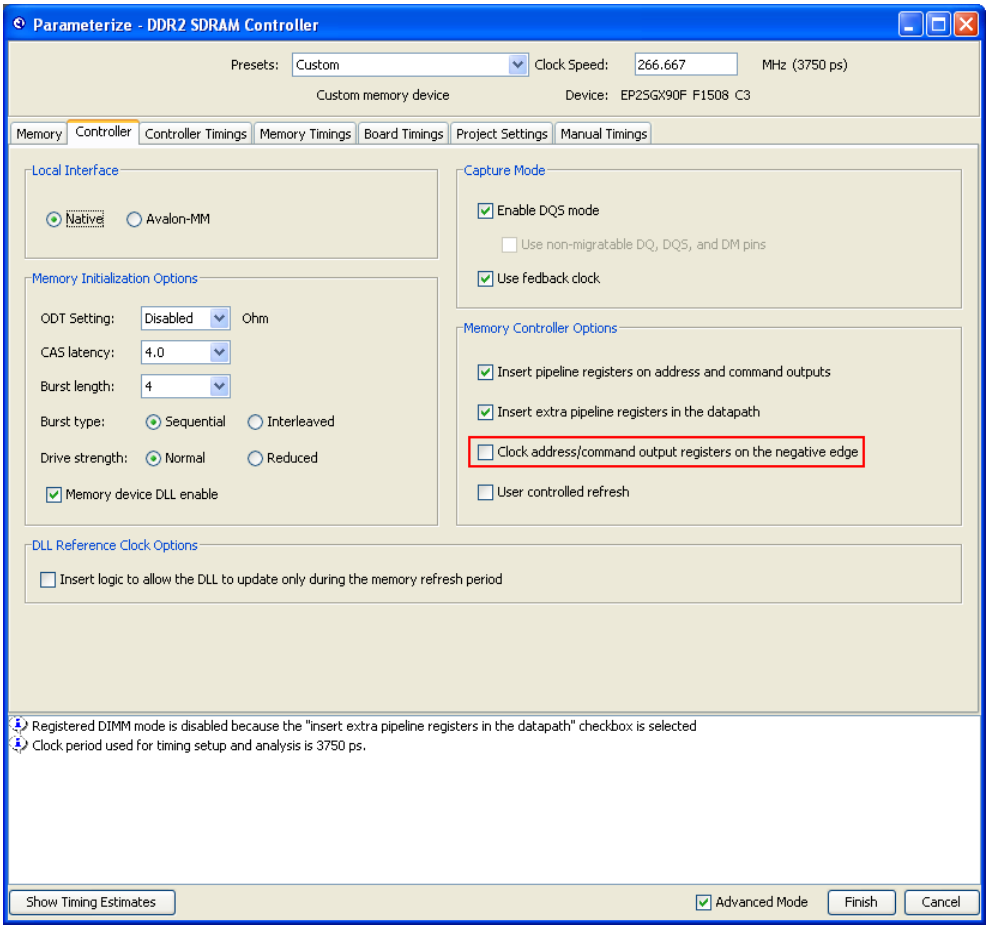
Changing the phase shift for the CK/CK# signals affects the read data path timing, rendering the timing analysis results shown in Figure 56 invalid.

Figure 56. Changing Clock Phase Shift Settings in the DDR2 SDRAM MegaWizard Plug-In Manager



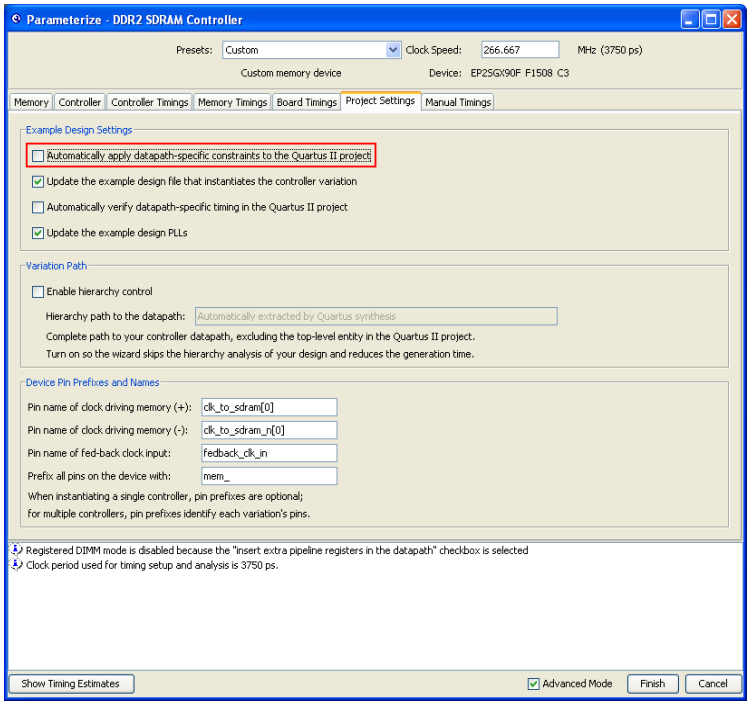
4. On the **Controller** page, turn off the **Clock address and command output registers on the negative edge** option. The address and command clock also needs a dedicated PLL output, based on the **dtw_timing_analysis.tcl** script phase shift recommendation, shown in Figure 51 on page 75. You can add 180° phase shift manually in the ALTPLL MegaWizard Plug-In Manager for easier tracking with this option turned off. This change is shown in Figure 57.

Figure 57. Changing the Address and Command Clock Edge



5. In the **Project Settings** page, turn off the **Automatically apply datapath-specific constraints to the Quartus II project** option, as shown in Figure 58. This prevents the DDR2 SDRAM MegaWizard Plug-In Manager from assigning the pin location and I/O standard assignments that have been changed before the previous compilation.

Figure 58. Disabling the auto_add_ddr_constraints.tcl Script



Disable the simulation netlist generation or any changes you made for functional simulation will be overwritten.

6. Click **Finish** and regenerate the controller to apply these changes.
7. Click **OK** when the MegaWizard Plug-In Manager warns you about overwriting existing files. To use a dedicated PLL output for the address and command clock, enable the c3 output of the system PLL and connect this output to `addrcmd_clk` in the `legacy_core` instantiation. The system PLL instantiation code in the **Legacy_PHY.v** file looks similar to the following example:

```
ddr_pll_stratixii g_stratixpll_ddr_pll_inst
    (
        .areset (pll_reset),
        .c0 (clk),
        .c1 (write_clk),
        .c2 (dedicated_resynch_or_capture_clk),
        .c3 (dedicated_addrcmd_clk),
        .inclk0 (clock_source),
        .locked (pll_locked
    );
```

The MegaWizard Plug-In Manager uses an input PLL clock that is the same frequency as the memory interface. You must change the input clock frequency in the ALTPLL MegaWizard Plug-In Manager for the `ddr_pll_stratixii` module. Remember to change the input clock frequency in this module to **100 MHz** for the design example. The MegaWizard Plug-In Manager overwrites any changes that you made. To avoid this, on the **Project Settings** page, you can disable the **Update the example design PLLs** option, as shown in Figure 58 on page 80. However, this means that you must manually update any changes in the PLL phase shifts.

In the `legacy_core` instantiation in the **Legacy_PHY.v** file, change:

```
.addrcmd_clk (clk),
```

to:

```
.addrcmd_clk (dedicated_addrcmd_clk),
```

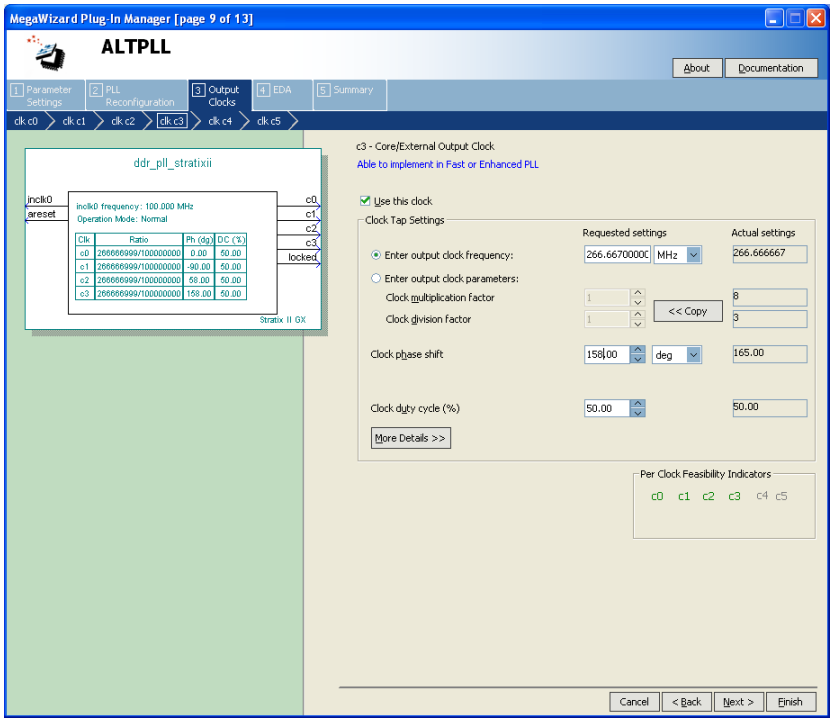
The `addrcmd_clk` signal is only available when the **Insert extra pipeline registers in the datapath** option is turned on in the **Controller** page of the DDR2 SDRAM **Controller Parameterization** window.

CAUTION

Because the design example was modified, on the **Project Setting** page shown in Figure 62 on page 89, make sure you turn off the **Update the example design file that instantiates the controller variation** option the next time you invoke the DDR2 SDRAM controller MegaWizard Plug-In Manager.

You must configure the phase shift for the output using the ALTPLL MegaWizard Plug-In Manager, as shown in Figure 59. The **dtw_timing_analysis.tcl** script recommends a phase shift of 22° for the address and command clock. However, in the previous compilation, the negative edge of the system clock was used for the address and command clock, which translates to 180° phase shift. Therefore, the address and command clock must have a 158° (180° – 22°) phase shift after the **Clock address and command output registers on the negative edge** option is turned off.

Figure 59. Address and Command Clock Phase Shift Setting



After changing the postamble and address/command clocks (leaving the CK/CK# clocks alone) as recommended by the **dtw_timing_analysis.tcl** script, perform Analysis and Elaboration to update the PLL settings and connections in the design netlist before running DTW again.

Run DTW and re-import the settings, then ensure that the updates you made to the design are reflected in the DTW pages.

You can also run the **dtw_timing_analysis.tcl** script with the `-after_iptb import_and_compile` switch to resynthesize the design, update DTW, recompile the design, and re-analyze the timing without having to perform each flow manually.

Compile the design and re-run the **dtw_timing_analysis.tcl** script in the command prompt after compilation is done. Figure 60 shows the timing results with all memory interface timings met. You can further fine-tune the phase shifts according the script result recommendation.

Figure 60. The Proper Timing Results After Compilation

Timing Summary

	Clock	Current Margin (ns)	Ideal Margin (ns)	Slow Setup (ns)	Slow Hold (ns)	Fast Setup (ns)	Fast Hold (ns)	PLL Name
1	Read capture	0.203	0.273	0.343	0.203	0.400	0.240	
2	Fed-back clock	0.029	0.077	0.029	0.934	1.228	0.126	g_stratixpll_ddr_fedback_pll_inst|altpll_component|pllclk[0]
3	Resynchronization clock	0.575	0.768	0.575	1.882	1.862	0.961	g_stratixpll_ddr_pll_inst|altpll_component|pllclk[2]
4	Postamble clock	0.371	1.003	0.691	1.635	0.371	2.384	g_stratixpll_ddr_fedback_pll_inst|altpll_component|pllclk[1]
5	Recovery/Removal	0.356	0.571	0.356	0.966	0.746	0.786	
6	tDQSS	0.395	0.700	1.005	0.395	1.239	0.411	g_stratixpll_ddr_pll_inst|altpll_component|pllclk[0]
7	Write Capture	0.149	0.254	0.149	0.359	0.360	0.358	g_stratixpll_ddr_pll_inst|altpll_component|pllclk[1]
8	Address/Command	0.903	1.121	1.339	0.907	1.350	0.903	g_stratixpll_ddr_pll_inst|altpll_component|pllclk[3]

Some core timing paths are not met. You must close the core timing by moving registers closer together or by creating one or more LogicLock™ regions.

For complete information about closing timing on this design example, refer to the *DDR Timing Wizard (DTW) User Guide*.

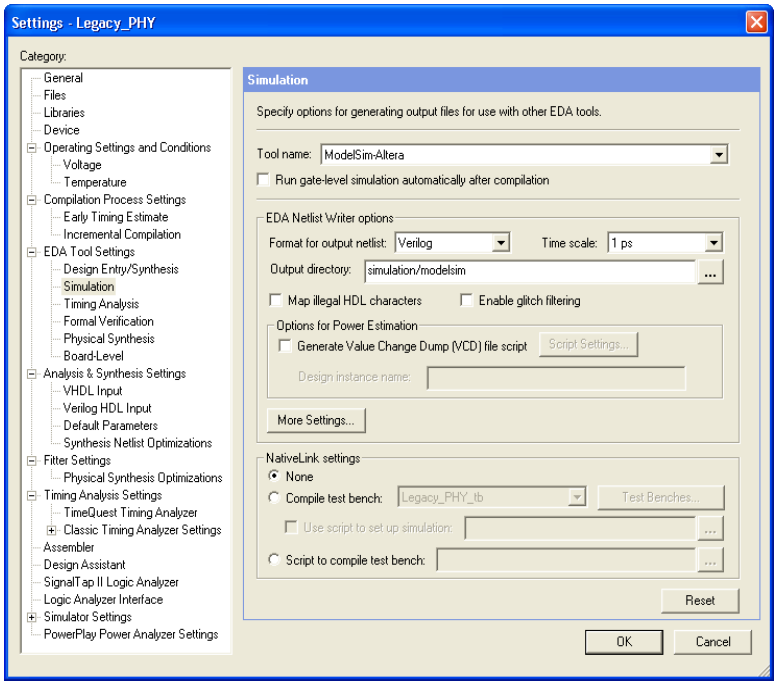
Step 7: Perform Gate-Level Simulation (Optional)

To perform gate-level simulation, follow these steps:

1. On the Assignment menu, click **EDA Tool Settings**. The **EDA Tool Settings** page appears.
2. In the **EDA Tool Settings** page, double-click on **Simulation**.
3. Set **Tool name** to **ModelSim-Altera**.
4. Choose **Verilog** for the output netlist. Figure 61 shows the changes to the **Simulation** page.

   You don't need to change the output directory because the MegaWizard-generated simulation script looks for the **.vo** and **.sdo** files in the default *<project_directory>***simulation****modelsim** directory.

Figure 61. Simulation Page



5. Click **OK**.
6. To generate the simulation netlist, open the Processing menu, point to **Start** and choose the **Start EDA Netlist Writer** option.
7. Follow the steps described in "Step 4: Perform the RTL/Functional Simulation (Optional)" on page 69. Before you execute the **.tcl** file, set the simulation mode to gate level by typing:

```
set use_gate_model 1
```

If you performed a functional simulation, you only need to set the gate model usage and execute the **.tcl** file.

To use the NativeLink feature, follow the instructions in the *Using the NativeLink Feature with ModelSim* section of the *Quartus II Handbook*, volume 3.

Step 8: Perform Board-Level Simulations to Verify the Design Constraints

For this design example, the board design constraints are already determined. However, for your actual board, determine the optimal termination scheme, termination implementation (OCT versus external resistors), drive strength settings, and system loading.

Table 10 shows the Altera-recommended board design constraints. However, ensure that the constraints in Table 10 satisfy your application's needs. The Stratix II GX PCI Express (PIPE) Development; for example, does not use ODT. Additionally, the board form factor only allows you to use single parallel termination for the bi-directional signals.

Table 10. Termination Recommendations (Part 1 of 2) *(Note 1)*

Form Factor	Signal Type	SSTL 18 IO Standard *(2)*, *(3)*, *(4)*, *(5)*, *(6)*	FPGA End Discrete Termination	Memory End Termination 1 Rank/DIMM	Memory IO Standard
DDR2	DQ/DQS	Class I R50 NO CAL	50 Ω Parallel to V_{tt} discrete	ODT75 *(8)*	FULL *(9)*
	DM	Class I R50 NO CAL	N/A	56 Ω Parallel to V_{tt} discrete	N/A
	Add/Cmd	Class I MAX	N/A		N/A
	Clock/Clock#	Class I MAX	N/A	N/A = on DIMM	N/A

Table 10. Termination Recommendations (Part 2 of 2) *(Note 1)*

Form Factor	Signal Type	SSTL 18 IO Standard *(2)*, *(3)*, *(4)*, *(5)*, *(6)*	FPGA End Discrete Termination	Memory End Termination 1 Rank/DIMM	Memory IO Standard
DDR2 Discrete	DQ/DQS	Class II R25 NO CAL	50 Ω Parallel to V_{tt} discrete	ODT75 *(8)*	HALF *(7)*
	DM	Class I R50 NO CAL	N/A	ODT75 *(8)*	N/A
	Add/Cmd	Class I R50 NO CAL	N/A	56 Ω Parallel to V_{tt} discrete	N/A
	Clock/Clock#	Class I R50 NO CAL	N/A	×1 = 100 Ω differential *(10)*	N/A
				×2 = 200 Ω differential *(11)*	

Notes to Table 10:

(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) HALF is reduced drive strength.
(8) ODT75 verses ODT50 on the memory has the effect of opening the eye more, with a limited increase in overshoot, undershoot, or both.
(9) FULL is full drive strength.
(10) ×1 is a single device load.
(11) ×2 is two device loads.

Evaluate trade-offs posed by various board design choices using simulations. Different factors contribute to signal integrity and affect the overall timing margin for the memory and FPGA. These include the termination scheme used, slew rate, and drive strength settings on the FPGA, and the loading seen by the driver. Evaluate the trade-offs between the different types of termination schemes, the effects of output drive strengths, and loading, so that you can navigate through the various design choices and choose optimal settings for your design.

To determine the correct board constraints, run 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 electronic design automation (EDA) simulation tools available to perform board-level simulations. You must perform the simulations on the data, clock, control, command, and address signals. If the memory interface does not have good signal integrity, adjust the settings, such as the drive strength setting, termination scheme, or termination values, to improve the signal integrity. (Changing these settings affects the timing.) It may be necessary to go back to the timing closure step if these change.

Step 9: Verify the FPGA Functionality

To verify the functionality of the design example, download the design to the Stratix II GX PCI-Express Development Board. The design example contains a `pnf` (pass not fail) signal that indicates whether the memory interface is functioning correctly. You can also use the SignalTap Embedded Logic Analyzer for board testing to verify the interface signals. You can now download the design into the board to verify functionality or expand the design for your application.

When considering implementing multi-DIMM systems, refer to *AN 380: Test DDR or DDR2 SDRAM Interfaces on Hardware Using the Example Driver.*

Multiple Memory Controllers Walkthrough

This walkthrough describes the steps to implement multiple memory interfaces in which 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), phase-locked loops (PLLs), and clock networks, and may require extra steps to create the interfaces in a Quartus II project.

A design example (**top.qar**) demonstrating multiple memory interfaces in a Stratix II device is downloadable along with this application note.

Getting to Know Your Altera Device

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 and clock networks, where applicable. The DLLs and clock network resources from the clock and reset management block are shared between multiple memory interfaces, but there are sharing limitations that you must pay close attention to.

Perform the following before actually creating the multiple memory interfaces in Quartus II:

- Compare the requirement for the multiple interface design with the resources available on the device.
- Determine which resources limit multiple interfaces.
- Decide which resources are shared in order to fit more memory interfaces in the device.

The following sections discuss the device resources to be considered for your multiple memory interface design:

- "I/O Banks" on page 87
- "I/O Pins" on page 87
- "I/O DLLs" on page 87
- "I/O PLLs and Clock Network Resources" on page 87
- "Other Resources" on page 88

After you have compared the required and available resources, you can start defining resources to be shared. This in turn determines the topology of your system. If no resources are to be shared, you can create the memory controllers as two different modules in the device. If you are sharing resources, the following sections offer guidance for sharing the DLL or PLL clock outputs, or both.

I/O Banks

You must determine the I/O bank placement of your memory interface. Arria GX, Stratix II, and Stratix II GX devices only support ALTMEMPHY-based memory interfaces on the top and bottom I/O banks (I/O banks 3, 4, 7, and 8). You can have multiple memory interfaces in a Stratix II I/O bank, provided that you are able to share the DLL and you have enough pins and clock networks for those interfaces. If you do not have enough clock networks, you must be able to share some of the clocks between the interfaces.

After deciding which I/O banks to use, ensure that you have enough I/O pins in the chosen I/O banks for your memory interfaces.

I/O Pins

DQS and DQ pins are listed in the device pin-outs and fixed at specific locations in the device. These locations are optimized in routing to minimize skew and maximize margin. Altera recommends you to always check the pin table and the external memory interfaces chapters for the number of DQS/DQ groups supported in a particular device.

You cannot share any I/O pins for multiple memory interfaces described in this document.

To check if the memory interfaces fit in the FPGA, compare the number of pins available in the FPGA with the number of pins required for the memory interfaces.

I/O DLLs

A single DLL can support any number of interfaces (limited by pins, PLLs, clock networks, and logic element resources) running at the same frequency and using the same DLL frequency mode.

For Arria GX, HardCopy II, Stratix II, and Stratix II GX devices, there is one DLL on the top and one DLL on the bottom of each device. Each DLL can only shift the DQS pins located on the same side as the DLL. Since there are only two DLLs, the Stratix II device can have a maximum of two memory interfaces with unique frequencies.

I/O PLLs and Clock Network Resources

The ALTMEMPHY megafunction uses a PLL that is instantiated in the ALTMEMPHY MegaWizard® Plug-In Manager. The PLL is responsible for generating the various clocks needed in the memory interface data path and controller.

You must decide whether you need to share clock networks, PLL clock outputs, or PLLs for your multiple memory interfaces.

Other Resources

You must pay attention to other device resources that are used in an external memory interface, for example the TriMatrix™ embedded memory blocks. If your design requires many TriMatrix embedded memory blocks for other modules in the design, ensure that there are enough for the memory interfaces as well.

Stratix II Design Example

Consider a Stratix II EP2S90F1508C3 device with the following memory requirements:

- One controller interfacing with a ×72 DDR2 SDRAM DIMM device running at 267 MHz
- Two controllers interfacing with ×8 DDR2 SDRAM components running at 267 MHz

All three controllers use the Altera DDR2 SDRAM High-Performance Controllers in half-rate mode.

Half-rate mode means that the clock frequency of the memory controller is half the frequency of the actual memory device's clock rate. In half-rate mode, ALTMEMPHY multiplexes and demultiplexes data between the memory controller and the memory devices to transfer data between the two frequencies.

Comparing I/O Pin Number Requirements

A DDR2 SDRAM DIMM interface requires about 124 pins, while a ×8 DDR2 SDRAM interface requires around 38 pins. As noted in the *Stratix II Device Handbook*, Stratix II has a maximum of 9 groups of ×8 DQS/DQ groups, such that the DIMM interface needs to take one side of the FPGA. Furthermore, the Quartus II Pin Planner indicates that there are 94 pins in I/O bank 3, and 102 pins in I/O bank 4, so the whole DIMM interface can fit in the top bank.

I/O banks 7 and 8 have 100 and 95 user I/O pins, respectively, so the two ×8 DDR2 SDRAM interfaces can share one I/O bank. However, for this exercise, the ×8 DDR2 SDRAM interfaces are placed in two different I/O banks so that the separation between the two interfaces are clearer.

To summarize, the DIMM interface is going to be located on the top of the device (I/O banks 3 and 4), while one ×8 DDR2 SDRAM controller is located on I/O bank 7, and the other on I/O bank 8.

Deciding DLL Requirements

Because there are two interfaces on the bottom side of the design, these interfaces must share a single DLL. This example uses two DLLs.

Creating PHY and Controller for the Design Example

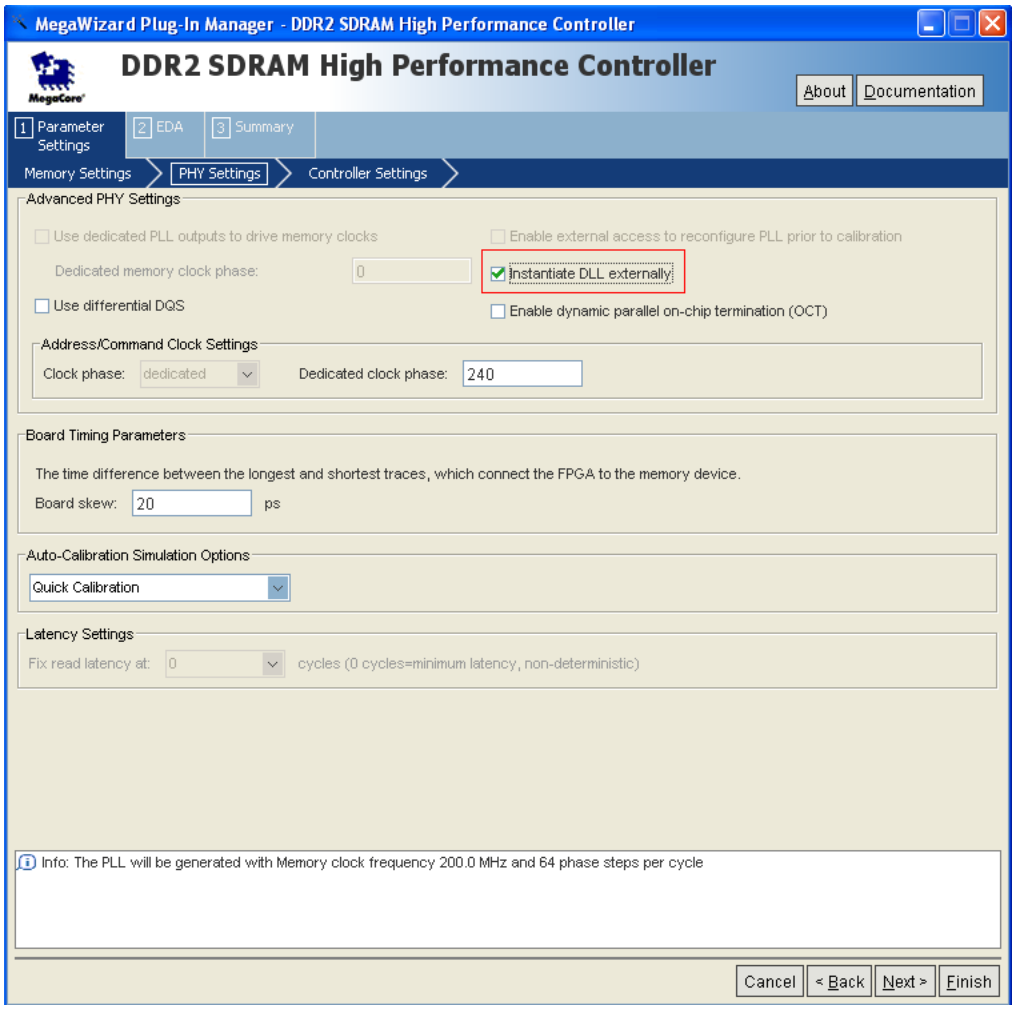
You must create the DIMM interface as a single interface without any special handling (called **top.v** in the design example) because it is not sharing any resources. The two memory interfaces at the bottom of the device are exactly the same. They both receive their static clocks from the **top.v** module. In this design example, because they are identical, you can create one variation and instantiate the same variation twice in the top-level file.

For easy organization of the files, Altera recommends that you create each controller as subfolders of the top project.

Sharing the DLLs Between the Two Bottom Controllers

The DLL is shared between the two bottom controllers in the example design. 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, as shown in Figure 62 on page 89. This option must be checked for every interface that is sharing a DLL.

Figure 62. Allowing the DLL to be Instantiated Externally for Controllers Sharing DLLs



When the Instantiate DLL externally option is checked, the MegaWizard Plug-In Manager generates a file called <*variation_name*>**_phy_alt_mem_phy_dll_sii.v/.vhd.**

The example top design 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 that is instantiated.

Sharing the PLL Clock Outputs

You can share the static clock networks in multiple memory controllers by setting the **Force Merging of PLL Clock Fanouts** assignment to **On**. You need to manually add the two PLL merging assignments using the Assignment Editor or directly update the **.qsf** file.

Figure 63 shows the snapshot of the assignment for Force Merge PLLs in **.qsf** file.

Figure 63. Assignments for Force Merge PLLs

```
840  set_instance_assignment -name FORCE_MERGE_PLL_FANOUTS ON -from
841  "*|bottom_phy_alt_mem_phy_clk_reset:clk|bottom_phy_alt_mem_phy_pll:*:altpll_component|*clk0" -to
842  "*|top_phy_alt_mem_phy_clk_reset:clk|top_phy_alt_mem_phy_pll:*:altpll_component|*clk0"
843  set_instance_assignment -name FORCE_MERGE_PLL_FANOUTS ON -from
844  "*|bottom_phy_alt_mem_phy_clk_reset:clk|bottom_phy_alt_mem_phy_pll:*:altpll_component|*clk2" -to
845  "*|top_phy_alt_mem_phy_clk_reset:clk|top_phy_alt_mem_phy_pll:*:altpll_component|*clk2"
```

Adding Constraints to the Design Example

To add constraints to the design example, you must enable TimeQuest timing analyzer and add all the **.sdc** files for the three controllers. Next, perform the following steps:

1. In the Quartus II software, open the **Pin Planner**.
2. Right-click **Create/Import Megafunction**.
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. The design example uses prefixes `top`, `bot1`, and `bot2`.

Each controller comes with its own <*variation_name*>**_pin_assignments.tcl** file. The pin assignments contained in this script use default pin names that must be changed so that they are unique for each controller instance.

You can also use the Pin Planner to assign actual locations to the pins. 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 the Assignment menu and click **Setting**.

Compiling the Design Example

Add the two **.sdc** files from the two controller variations to the design. You must also add another **.sdc** file to the list of TimeQuest timing analyzer **.sdc** files to describe the frequency of the input PLL reference clock pins of all the controllers in the design. The design example uses a file called **top.sdc** which has the following lines:

```
create_clock -period 10 top_pll_ref_clk
create_clock -period 10 bot1_pll_ref_clk
create_clock -period 10 bot2_pll_ref_clk
```

If you place the controllers in a different folder than the top-level project, you must add the directories of your controllers to the project library.

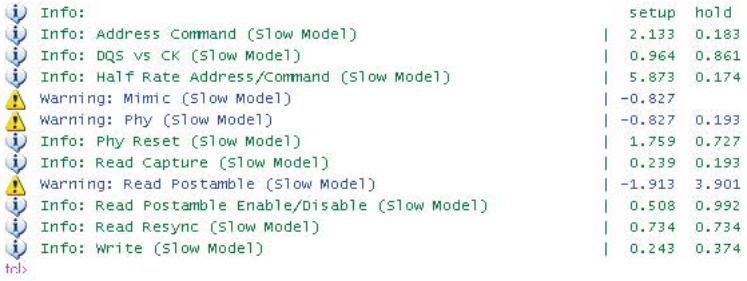
Compiling the Design Example to Verify Timing

You must compile the design and select the **TimeQuest Timing Analyzer** from the Tools menu. Double-click on the **Report DDR** macro to check if the design meets all timing requirements. If not, you must fix the timing by either adding or removing delay chains, or by moving the registers closer using placement constraints.

To ensure faster timing closure, you can close timing on each controller and place the controller logic in a LogicLock™ block before creating the multiple interface top level design.

Figure 64 shows the initial timing analysis results for the three controllers in the design.

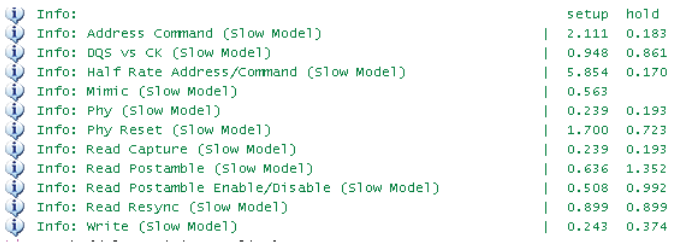
Figure 64. Initial Timing Analysis Result for the Design Example



If you look into the failing path, you can see that the failing path is on the mimic register. You must move this path closer to the IOE that is feeding the register in **Chip Planner** and recompile the design.

Figure 65 shows the final timing analysis result for the design.

Figure 65. Final Timing Analysis Result for the Design Example



After your design meets timing, you can continue your design flow by either performing a gate-level simulation or determining the board design constraints.

Conclusion

The advanced clocking features available in Stratix II, Stratix II GX, and Arria GX devices allow for a high performance, versatile interface to DDR2 SDRAM. For applications requiring the greater memory bandwidth offered by DDR2 SDRAM, Altera offers device families with complete, proven memory solutions for these devices.

Arria GX devices only support ALTMEMPHY implementations, but when targeting Stratix II or Stratix II GX devices, you have the choice of either using ALTMEMPHY for high-performance memory interfaces up to 333 MHz or using the legacy PHY to interface with more than 72-bit wide memory per Stratix II or Stratix II GX device side. Altera recommends using the ALTMEMPHY implementation to get the optimal margin. Use the legacy PHY only if you need a non-DQS implementation or lower latency application.

Appendix A: Stratix II GX PCI-Express Development Board Pin Assignments

Table 11 shows pin assignments used in Stratix II GX PCI Express (PIPE) Development Board. The pin names shown here are based on the default pin names when using the ALTMEMPHY-based controllers. There are some pin names in the legacy PHY different from pin names in ALTMEMPHY such as the `reset_n(global_reset_n)`, `clk_to_sdram[0:2](mem_clk[0:2])`, `clk_to_sdram_n[0:2](mem_clk_n[0:2])`, and `mem_a[0:14](mem_addr[0:14])`.

Table 11. PCI Express (PIPE) Development Board, Stratix II GX Edition I/O and Pin Assignments (Part 1 of 4) *(Note 1)*, *(2)*

Pin Name	I/O Standard	Pin Location	Output Pin Load (pF)	Termination	Current Strength
mem_addr[0]	SSTL-18 Class I	AP16	10	N/A	Maximum Current
mem_addr[1]	SSTL-18 Class I	AH28	10	N/A	Maximum Current
mem_addr[10]	SSTL-18 Class I	AT30	10	N/A	Maximum Current
mem_addr[11]	SSTL-18 Class I	AN21	10	N/A	Maximum Current
mem_addr[12]	SSTL-18 Class I	AP28	10	N/A	Maximum Current
mem_addr[13] *(3)*	SSTL-18 Class I	AL28	10	N/A	Maximum Current
mem_addr[14] *(3)*	SSTL-18 Class I	AP19	10	N/A	Maximum Current
mem_addr[2]	SSTL-18 Class I	AP26	10	N/A	Maximum Current
mem_addr[3]	SSTL-18 Class I	AP29	10	N/A	Maximum Current
mem_addr[4]	SSTL-18 Class I	AL15	10	N/A	Maximum Current
mem_addr[5]	SSTL-18 Class I	AK27	10	N/A	Maximum Current
mem_addr[6]	SSTL-18 Class I	AK25	10	N/A	Maximum Current
mem_addr[7]	SSTL-18 Class I	AU29	10	N/A	Maximum Current
mem_addr[8]	SSTL-18 Class I	AH15	10	N/A	Maximum Current
mem_addr[9]	SSTL-18 Class I	AH25	10	N/A	Maximum Current
mem_ba[0]	SSTL-18 Class I	AN28	10	N/A	Maximum Current

Table 11. PCI Express (PIPE) Development Board, Stratix II GX Edition I/O and Pin Assignments (Part 2 of 4) *(Note 1)*, *(2)*

Pin Name	I/O Standard	Pin Location	Output Pin Load (pF)	Termination	Current Strength
mem_ba[1]	SSTL-18 Class I	AG24	10	N/A	Maximum Current
mem_ba[2] *(3)*	SSTL-18 Class I	AH27	10	N/A	Maximum Current
mem_cas_n	SSTL-18 Class I	AG23	10	N/A	Maximum Current
mem_cke[0]	SSTL-18 Class I	AF18	10	N/A	Maximum Current
mem_clk[0]	SSTL-18 Class I	AW19	4	N/A	Maximum Current
mem_clk[1]	SSTL-18 Class I	AU20	4	N/A	Maximum Current
mem_clk[2]	SSTL-18 Class I	AP20	2	N/A	Maximum Current
mem_clk_n[0]	SSTL-18 Class I	AV19	4	N/A	Maximum Current
mem_clk_n[1]	SSTL-18 Class I	AT20	4	N/A	Maximum Current
mem_clk_n[2]	SSTL-18 Class I	AN20	2	N/A	Maximum Current
mem_cs_n[0]	SSTL-18 Class I	AJ25	10	N/A	Maximum Current
mem_dm[0]	SSTL-18 Class I	AT11	4	N/A	Maximum Current
mem_dm[1]	SSTL-18 Class I	AP12	4	N/A	Maximum Current
mem_dm[2]	SSTL-18 Class I	AU15	4	N/A	Maximum Current
mem_dm[3]	SSTL-18 Class I	AT17	4	N/A	Maximum Current
mem_dm[4]	SSTL-18 Class I	AP18	4	N/A	Maximum Current
mem_dm[5]	SSTL-18 Class I	AU24	4	N/A	Maximum Current
mem_dm[6]	SSTL-18 Class I	AV27	4	N/A	Maximum Current
mem_dm[7]	SSTL-18 Class I	AV30	4	N/A	Maximum Current
mem_dm[8]	SSTL-18 Class I	AW36	4	N/A	Maximum Current
mem_dq[0]	SSTL-18 Class I	AU9	4	N/A	Maximum Current
mem_dq[1]	SSTL-18 Class I	AN10	4	N/A	Maximum Current
mem_dq[10]	SSTL-18 Class I	AR12	4	N/A	Maximum Current
mem_dq[11]	SSTL-18 Class I	AW12	4	N/A	Maximum Current
mem_dq[12]	SSTL-18 Class I	AN13	4	N/A	Maximum Current
mem_dq[13]	SSTL-18 Class I	AT13	4	N/A	Maximum Current
mem_dq[14]	SSTL-18 Class I	AN12	4	N/A	Maximum Current
mem_dq[15]	SSTL-18 Class I	AU13	4	N/A	Maximum Current
mem_dq[16]	SSTL-18 Class I	AW13	4	N/A	Maximum Current
mem_dq[17]	SSTL-18 Class I	AN14	4	N/A	Maximum Current
mem_dq[18]	SSTL-18 Class I	AV13	4	N/A	Maximum Current
mem_dq[19]	SSTL-18 Class I	AP14	4	N/A	Maximum Current
mem_dq[2]	SSTL-18 Class I	AP10	4	N/A	Maximum Current
mem_dq[20]	SSTL-18 Class I	AT15	4	N/A	Maximum Current
mem_dq[21]	SSTL-18 Class I	AR15	4	N/A	Maximum Current
mem_dq[22]	SSTL-18 Class I	AW14	4	N/A	Maximum Current
mem_dq[23]	SSTL-18 Class I	AW15	4	N/A	Maximum Current
mem_dq[24]	SSTL-18 Class I	AN16	4	N/A	Maximum Current

Table 11. PCI Express (PIPE) Development Board, Stratix II GX Edition I/O and Pin Assignments (Part 3 of 4) *(Note 1)*, *(2)*

Pin Name	I/O Standard	Pin Location	Output Pin Load (pF)	Termination	Current Strength
mem_dq[25]	SSTL-18 Class I	AN15	4	N/A	Maximum Current
mem_dq[26]	SSTL-18 Class I	AU16	4	N/A	Maximum Current
mem_dq[27]	SSTL-18 Class I	AT16	4	N/A	Maximum Current
mem_dq[28]	SSTL-18 Class I	AN17	4	N/A	Maximum Current
mem_dq[29]	SSTL-18 Class I	AW16	4	N/A	Maximum Current
mem_dq[3]	SSTL-18 Class I	AW9	4	N/A	Maximum Current
mem_dq[30]	SSTL-18 Class I	AV16	4	N/A	Maximum Current
mem_dq[31]	SSTL-18 Class I	AP17	4	N/A	Maximum Current
mem_dq[32]	SSTL-18 Class I	AW18	4	N/A	Maximum Current
mem_dq[33]	SSTL-18 Class I	AT18	4	N/A	Maximum Current
mem_dq[34]	SSTL-18 Class I	AW17	4	N/A	Maximum Current
mem_dq[35]	SSTL-18 Class I	AR18	4	N/A	Maximum Current
mem_dq[36]	SSTL-18 Class I	AN18	4	N/A	Maximum Current
mem_dq[37]	SSTL-18 Class I	AT19	4	N/A	Maximum Current
mem_dq[38]	SSTL-18 Class I	AU19	4	N/A	Maximum Current
mem_dq[39]	SSTL-18 Class I	AN19	4	N/A	Maximum Current
mem_dq[4]	SSTL-18 Class I	AV10	4	N/A	Maximum Current
mem_dq[40]	SSTL-18 Class I	AP23	4	N/A	Maximum Current
mem_dq[41]	SSTL-18 Class I	AW23	4	N/A	Maximum Current
mem_dq[42]	SSTL-18 Class I	AW24	4	N/A	Maximum Current
mem_dq[43]	SSTL-18 Class I	AV24	4	N/A	Maximum Current
mem_dq[44]	SSTL-18 Class I	AT24	4	N/A	Maximum Current
mem_dq[45]	SSTL-18 Class I	AP24	4	N/A	Maximum Current
mem_dq[46]	SSTL-18 Class I	AW25	4	N/A	Maximum Current
mem_dq[47]	SSTL-18 Class I	AV25	4	N/A	Maximum Current
mem_dq[48]	SSTL-18 Class I	AP25	4	N/A	Maximum Current
mem_dq[49]	SSTL-18 Class I	AR25	4	N/A	Maximum Current
mem_dq[5]	SSTL-18 Class I	AU10	4	N/A	Maximum Current
mem_dq[50]	SSTL-18 Class I	AU26	4	N/A	Maximum Current
mem_dq[51]	SSTL-18 Class I	AW26	4	N/A	Maximum Current
mem_dq[52]	SSTL-18 Class I	AU27	4	N/A	Maximum Current
mem_dq[53]	SSTL-18 Class I	AW27	4	N/A	Maximum Current
mem_dq[54]	SSTL-18 Class I	AW28	4	N/A	Maximum Current
mem_dq[55]	SSTL-18 Class I	AT27	4	N/A	Maximum Current
mem_dq[56]	SSTL-18 Class I	AT28	4	N/A	Maximum Current
mem_dq[57]	SSTL-18 Class I	AW29	4	N/A	Maximum Current
mem_dq[58]	SSTL-18 Class I	AR28	4	N/A	Maximum Current
mem_dq[59]	SSTL-18 Class I	AT29	4	N/A	Maximum Current

Table 11. PCI Express (PIPE) Development Board, Stratix II GX Edition I/O and Pin Assignments (Part 4 of 4) *(Note 1)*, *(2)*

Pin Name	I/O Standard	Pin Location	Output Pin Load (pF)	Termination	Current Strength
mem_dq[6]	SSTL-18 Class I	AN11	4	N/A	Maximum Current
mem_dq[60]	SSTL-18 Class I	AU30	4	N/A	Maximum Current
mem_dq[61]	SSTL-18 Class I	AW30	4	N/A	Maximum Current
mem_dq[62]	SSTL-18 Class I	AW31	4	N/A	Maximum Current
mem_dq[63]	SSTL-18 Class I	AU31	4	N/A	Maximum Current
mem_dq[64]	SSTL-18 Class I	AW32	4	N/A	Maximum Current
mem_dq[65]	SSTL-18 Class I	AU32	4	N/A	Maximum Current
mem_dq[66]	SSTL-18 Class I	AU33	4	N/A	Maximum Current
mem_dq[67]	SSTL-18 Class I	AW34	4	N/A	Maximum Current
mem_dq[68]	SSTL-18 Class I	AW35	4	N/A	Maximum Current
mem_dq[69]	SSTL-18 Class I	AV34	4	N/A	Maximum Current
mem_dq[7]	SSTL-18 Class I	AW10	4	N/A	Maximum Current
mem_dq[70]	SSTL-18 Class I	AV37	4	N/A	Maximum Current
mem_dq[71]	SSTL-18 Class I	AW37	4	N/A	Maximum Current
mem_dq[8]	SSTL-18 Class I	AT12	4	N/A	Maximum Current
mem_dq[9]	SSTL-18 Class I	AW11	4	N/A	Maximum Current
mem_dqs[0]	SSTL-18 Class I	AT9	4	N/A	Maximum Current
mem_dqs[1]	SSTL-18 Class I	AU12	4	N/A	Maximum Current
mem_dqs[2]	SSTL-18 Class I	AT14	4	N/A	Maximum Current
mem_dqs[3]	SSTL-18 Class I	AP15	4	N/A	Maximum Current
mem_dqs[4]	SSTL-18 Class I	AV18	4	N/A	Maximum Current
mem_dqs[5]	SSTL-18 Class I	AU23	4	N/A	Maximum Current
mem_dqs[6]	SSTL-18 Class I	AT25	4	N/A	Maximum Current
mem_dqs[7]	SSTL-18 Class I	AU28	4	N/A	Maximum Current
mem_dqs[8]	SSTL-18 Class I	AW33	4	N/A	Maximum Current
mem_odt[0]	SSTL-18 Class I	AN25	10	N/A	Maximum Current
mem_ras_n	SSTL-18 Class I	AJ27	10	N/A	Maximum Current
mem_we_n	SSTL-18 Class I	AL13	10	N/A	Maximum Current
clock_source	LVDS	AW22	—	—	—
global_reset_n	1.8 V	AM22	—	—	—
pnf	1.8 V	AN31	—	—	—
test_complete	1.8 V	AT31	—	—	—

Notes to Table 11

(1) The pin names correspond to the pin names used in the ALTMEMPHY design example. The legacy PHY design example uses a `ddr2_` prefix instead of a `mem_` prefix.

(2) Address and command pins have more loads because they are connected to more than one device; therefore, they need the available maximum current from the FPGA.

(3) These pins are connected in the board but are not used in the design. Reserve these pins To be used as the output driving ground in the Assignment Editor.

Appendix B: Interface Signal Description

The following section describes the interface signals between the FPGA and the DDR2 SDRAM components, how you need to configure the FPGA pins to meet the DDR2 SDRAM electrical and timing requirements, and lists the number of DQS and DQ pins available in the FPGA.

Interface Signals

Table 12 lists a summary of the DDR2 SDRAM interface pins and how to connect them to a Stratix II, Stratix II GX, or Arria GX device.

Table 12. DDR2 SDRAM Interface Pins

Pins	Description	Stratix II, Stratix II GX, or Arria GX Pin Utilization
DQ	Bi-directional read and write data bus	DQ
DQS	Bi-directional read and write data strobe	DQS
DQS# *(1)*	Optional bi-directional differential write data strobe	N/A
CK, CK#	System clock	DQ, DQS, or DQSn *(3)*, *(4)*
`fedback_clk_out` *(2)*	Copy of CK output clock signal fedback to read the PLL for resynchronization in DLL-based implementation with two PLLs or for read capture in PLL-based implementation.	User I/O output pin feeding PLL clock input pin
`fedback_clk_in` *(2)*	Loopback signal from the `fedback_clk_out` signal	PLL clock input pin
DM	Optional write data mask, edge-aligned to DQ during write	DQ *(5)*
All other	Address, command, and so on	User I/O pin

Notes to Table 12:

(1) The DQS# signal in DDR2 SDRAM components is optional. Stratix II, Stratix II GX, and Arria GX devices do not use the DQS# pins when interfacing with DDR2 SDRAM, as they do not support differential DQS signaling.

(2) This signal is not applicable for the PLL–based implementation with one PLL or for the ALTMEMPHY implementation.

(3) Any unused DQ or DQS pins with DIFFIO_RX capability for `mem_clk[0]` and `mem_clk_n[0]`.

(4) Any unused DQ or DQS pins with DIFFOUT capability for `mem_clk[n:1]` and `mem_clk_n[n:1]`. Where *n* is greater than or equal to 1.

(5) The DM pins must be in the DQ group.

The following section describes the clock, strobe, data, address, and command signals on a DDR2 SDRAM component.

Clock Signals

The DDR2 SDRAM component uses 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 of the DLL inside the memory. The skew between the CK or CK# signals and the DDR2 SDRAM-generated DQS signal is specified as t_{DQSCK} in the DDR2 SDRAM data sheet.

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. Therefore, you must 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}.

HardCopy II structured ASICs require the use of the dedicated PLL clock outputs for CK and CK# signals. For more information about interfacing with external memory devices using HardCopy II structured ASICs, refer to *AN 463: Using ALTMEMPHY Megafunction with HardCopy II Structured ASICs* or *AN 413: Using Legacy Integrated Static Data Path and Controller Megafunction with HardCopy II Structured ASICs*.

To improve resynchronization timing for DDR2 SDRAM interfaces running at or above 200 MHz, you can route a copy of the CK signal (called `fedback_clk_out`) from the memory pin back to the Stratix II or Stratix II GX devices (called `fedback_clk_in`). Arria GX devices do not support this implementation. For more information, refer to "Round-Trip Delay Calculation" on page 121.

Strobes, Data, DM, and Optional ECC Signals

The DQS is bi-directional. The DQS# pins in DDR2 SDRAM components and the DQS# pins in Stratix II devices are not used in DDR2 SDRAM interfaces. Connect the memory's DQS pins to the Stratix II, Stratix II GX, or Arria GX DQS pins. The DQ pins are also bi-directional. A group of DQ pins is associated with one DQS pin.

In ×8 and ×16 DDR2 SDRAM components, one DQS pin is always associated with eight DQ pins (×8/×9 mode in the Stratix II device).

Refer to Table 13 through Table 17 for the number of DQS and DQ groups supported in Stratix II, Stratix II GX, and Arria GX devices. Stratix II and Stratix II GX devices support both DLL- and PLL- based implementations, while Arria GX devices only support DLL based implementations with ALTMEMPHY. Each DQS pin can drive up to 4, 9, 18, or 36 DQ pins.

Table 13. DQS and DQ Bus Mode Support in Stratix II Devices for DLL-Based Implementations (Part 1 of 2)

Package	Number of ×4 Groups	Number of ×8/×9 Groups
484-pin FineLine BGA	8	4
672-pin FineLine BGA	18	8
484-pin FineLine BGA	8	4
672-pin FineLine BGA	18	8
484-pin FineLine BGA	8	4
672-pin FineLine BGA	18	8
1,020-pin FineLine BGA	36	18
484-pin Hybrid FineLine BGA	8	4
780-pin FineLine BGA	18	8
1,020-pin FineLine BGA	36	18

Table 13. DQS and DQ Bus Mode Support in Stratix II Devices for DLL-Based Implementations (Part 2 of 2)

Package	Number of ×4 Groups	Number of ×8/×9 Groups
1,508-pin FineLine BGA	36	18
780-pin FineLine BGA	18	8
1,020-pin FineLine BGA	36	18
1,508-pin FineLine BGA	36	18
1,020-pin FineLine BGA	36	18
1,508-pin FineLine BGA	36	18

Table 14. Stratix II DQS and DQ Bus Mode Support for PLL-Based Implementaitons *(Note 1)*

Device	Package	Number of ×4 Groups	Number of ×8/×9 Groups
EP2S15	484-pin FineLine BGA	13	7
	672-pin FineLine BGA	24	9
EP2S30	484-pin FineLine BGA	13	7
	672-pin FineLine BGA	36	15
EP2S60	484-pin FineLine BGA	13	7
	672-pin FineLine BGA	36	15
	1,020-pin FineLine BGA	51	26
EP2S90	780-pin FineLine BGA	40	24
	1,020-pin FineLine BGA	51	25
	1,508-pin FineLine BGA	51	25
EP2S130	780-pin FineLine BGA	40	24
	1,020-pin FineLine BGA	51	25
	1,508-pin FineLine BGA	51	25
EP2S180	1,020-pin FineLine BGA	51	25
	1,508-pin FineLine BGA	51	25

Note to Table 14:

(1) Check the pin table for each DQS and DQ group in the different modes.

Table 15. Stratix II GX DQS and DQ Bus Mode Support for DLL-Based Implementations *(Note 1)*

Package	Number of ×4 Groups	Number of ×8/×9 Groups
780-pin FineLine BGA	18	8
780-pin FineLine BGA	18	8
1,152-pin FineLine BGA	36	18
1,152-pin FineLine BGA	36	18
1,508-pin FineLine BGA	36	18
1,508-pin FineLine BGA	36	18

Note to Table 15:

(1) Check the pin table for each DQS and DQ group in the different modes.

Table 16. Stratix II GX DQS and DQ Bus Mode Support for PLL-Based Implementations *(Note 1)*

Package	Number of ×4 Groups	Number of ×8/×9 Groups
780-pin FineLine BGA	18	8
780-pin FineLine BGA	18	8
1,152-pin FineLine BGA	25	13
1,152-pin FineLine BGA	25	13
1,508-pin FineLine BGA	25	12
1,508-pin FineLine BGA	25	12

Note to Table 16:

(1) Check the pin table for each DQS and DQ group in the different modes.

Table 17. Arria GX DQS and DQ Bus Mode Support for DLL-Based Implementations *(Note 1)*, *(2)*

Package	Number of ×4 Groups	Number of ×8/×9 Groups
484-pin FineLine BGA	2	0
780-pin FineLine BGA	18	8
1,152-pin FineLine BGA	36	18

Notes to Table 17:

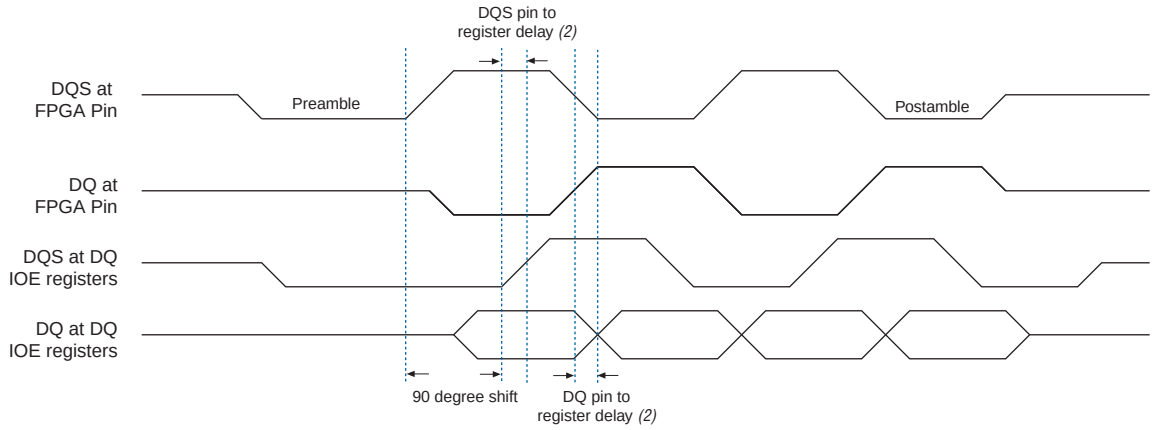
(1) Check the pin table for each DQS and DQ group in the different modes.

(2) Arria GX devices only support DLL-based implementation using the ALTMEMPHY MegaWizard Plug-In Manager.

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; the memory controller delays the DQS signal during a read so that the DQ and DQS signals are center aligned at the capture register. Stratix II devices use a phase-locked loop (PLL) to center-align the DQS signal with respect to the DQ signals during writes and use dedicated DQS phase-shift circuitry or another PLL to shift the incoming DQS signal during reads.

Figure 66 shows an example where the DQS signal is shifted by 90° for a read from the DDR2 SDRAM.

Figure 66. DQ and DQS Relationship During a DDR2 SDRAM Read in Burst-of-Four Mode *(Note 1)*



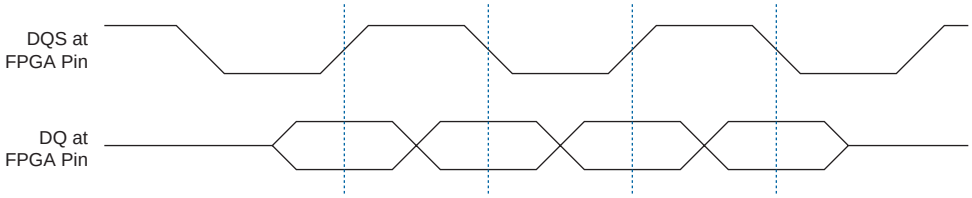
Notes to Figure 66:

(1) This is an example of a 90° shift. Base your timing analysis on the shift value read for your system. The shift may not be 90°.

(2) The delay from the DQS pin to the capture register and DQ pin to the capture register to minimize additional skew between these signals at the IOE registers.

Figure 67 shows an example of the relationship between the data and data strobe during a burst-of-four write.

Figure 67. DQ and DQS Relationship During a DDR2 SDRAM Write in Burst-of-Four Mode



The memory device's setup (t_{ds}) and hold times (tdh) for the write DQ and DM pins are relative to the edges of the DQS write signals and not the CK or CK# clock. These specifications are not necessarily symmetrical in DDR2 SDRAM, unlike in DDR SDRAM components.

The DQS signal is generated on the positive edge of the system clock to meet the t_{DQSS} requirement. DQ and data mask (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.

The 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. 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.

The DM signal's timing requirements at the DDR2 SDRAM input are identical to those for DQ data. The Stratix II DDR registers, clocked by the –90° shifted clock, create the DM signals.

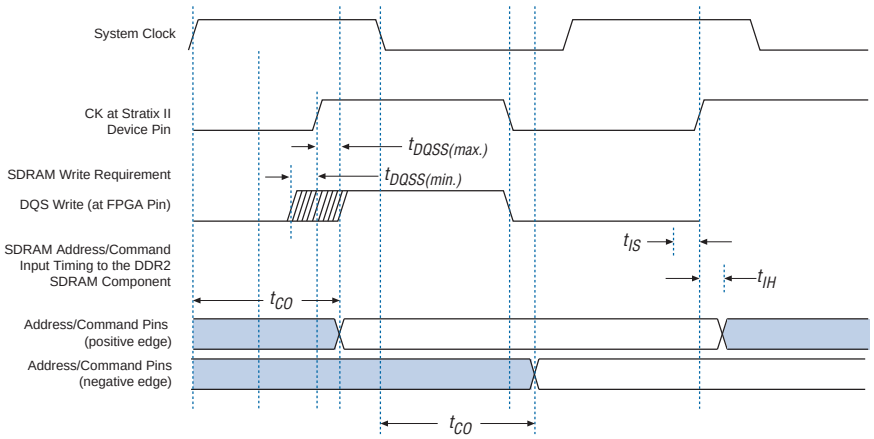
Some DDR2 SDRAM components support error correction coding (ECC) to detect and automatically correct an error in data transmission. The 72-bit DDR2 SDRAM modules contain eight ECC pins in addition to the 64 data pins. Connect the DDR2 SDRAM component ECC pins to a Stratix II device DQS/DQ group. You must create a 72-bit DDR2 SDRAM memory controller and add logic to decode and encode the ECC bits on the local interface of the controller.

Address and Command Signals

Address and command signals in DDR2 SDRAM components are clocked into the memory 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 DDR2 SDRAM component 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 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# (Figure 68).

Figure 68. Address and Command Timing *(Note 1), (2)*



Notes to Figure 68:

(1) The address and command timing shown in Figure 67 is applicable for both reads and writes.

(2) If the board trace lengths for the DQS, CK, address, and command pins are the same, the signal relationships at the Stratix II, Stratix II GX, or Arria GX device pins are maintained at the DDR2 SDRAM pins.

You must perform a separate timing analysis for addresses and commands to decide how to generate these signals (using the positive or negative edge of the system clock or a phase-shifted version of the system clock). This clock edge selection is made based on timing analysis with accurate pin loading information. The timing analysis methodology for address and command signals is very similar to the write timing paths described in "Write Data Timing Analysis" on page 118. You can use any of the I/O pins for the commands and addresses.

The address and command pins at the Stratix II device nominally change at the same time as the DQS write signal because they are both generated from the system clock.

Appendix C: Legacy PHY Architecture Description

There are two data paths or PHY available in Stratix II and Stratix II GX devices; the legacy PHY and the ALTMEMPHY megafunction. For highest performance, use the ALTMEMPHY megafunction, which is the only data path supported in Arria GX devices.

For more information about this implementation, refer to the *DDR and DDR2 SDRAM High-Performance Controller and ALTMEMPHY User Guide*.

If you are not sure which PHY to use, refer to *Technical Brief 091: External Memory Interface Options for Stratix II Devices*.

In addition, the legacy PHY offers two read-side implementations:

- DLL-based read implementation—This method uses the DLL to phase shift the DQS strobe and center-align the read data DQ with respect to the DQS at the IOE registers. This implementation is limited to the top and bottom banks of the Stratix II and Stratix II GX devices with a maximum clock rate of 267 MHz. This implementation is also offered with two modes:
  - One-PLL implementation with performance up to 200 MHz. This implementation uses one DLL and one PLL, whereby the PLL generates all the necessary clocks needed for the interface.
  - Fedback-clock mode with performance up to 267 MHz. This implementation uses one DLL and two PLLs. One PLL is used to generate the system and the write clocks, while the other PLL is used to generate resynchronization and postamble clocks.
- PLL-based read implementation—This method uses a PLL to generate the phase-shifted clock to capture the read data DQ (instead of using the DQS strobe from the memory devices). You can use this implementation in any I/O banks of the Stratix II and Stratix II GX devices, but the performance is limited to 200 MHz.

The DLL-based mode implementation always gives a higher performance than the PLL-based implementation. This is because the PLL-based implementation ignores the DQS strobes during reads. Timing analysis, when not using the DQS strobes during reads, uses the t_{AC} specifications (which is ± 600 ps for a 200-MHz DDR2 SDRAM component) for skew between data signals. Timing analysis using DQS strobes uses the t_{DQSQ} and t_{QHS} specifications, which is 350 ps and 450 ps, respectively, for a 200-MHZ DDR2 SDRAM component. By using the DQS strobes during read, you save 400 ps of timing uncertainties at 200 MHz.

When using PLL-based implementation, the top and bottom I/O banks may also give better performance. This is because the side I/O pins support LVDS, which results in the buffer having higher capacitance and lower I/O performance.

Write-side implementation includes a PLL that outputs two clocks that generate the write data and center-aligned write clock using the DDR registers in the IOE. This implementation results in matched propagation delays for clock and data signals from the FPGA to the DDR2 SDRAM, minimizing skew.

Legacy Data Path Architecture Using Dedicated DQS Phase-Shift Circuitry

The DDR2 SDRAM interface legacy PHY implementation using dedicated DQS phase-shift circuitry uses the following:

- A write-side PLL to generate CK and CK# system clocks and clock-out address, command, strobe, and data signals.
- A read-side DLL-based phase circuitry to capture read data from the memory using strobe signals, DQS.
- An optional PLL for the fedback clock mode to generate resynchronization and postamble clocks.

This implementation is also called the DQS mode or DLL-based implementation available with the DDR and DDR2 SDRAM Controller MegaCore function. The DDR2 SDRAM Controller initializes the memory devices, manages SDRAM banks, and keeps devices refreshed at the appropriate intervals. The MegaCore function translates read and write requests from the local interface into all the necessary SDRAM command signals. The controller also contains encrypted control logic as well as a clear-text data path that you can use in your design without a license. Download this MegaCore function whether you plan to use the Altera DDR2 SDRAM controller or not, to get the clear-text data path, clear-text DQS postamble logic, and placement constraints. The MegaCore function is accessible through the DDR2 SDRAM Controller MegaWizard Plug-In Manager. When you parameterize your custom DDR2 SDRAM interface, the DDR2 SDRAM Controller MegaWizard Plug-In Manager automatically decides the best phase-shift and FPGA settings to give you the best margin for your DDR2 SDRAM interface. It then generates an example instance that instantiates a PLL, an example driver, and your DDR2 SDRAM controller custom variation, as shown in Figure 29 on page 52.

For more information, refer to Altera's memory controllers page and download the *DDR2 SDRAM Controller MegaCore MegaFunction*.

The 1-PLL mode supports up to 200-MHz memory interface, while the fedback-clock mode supports up to 267-MHz memory interface. However, due to complex timing analysis with two PLLs, Altera recommends the ALTMEMPHY Megafunction for interfaces above 200 MHz.

This implementation is only supported on Stratix II and Stratix II GX top and bottom I/O banks because the DLLs and dedicated DQS phase-shift circuitry are available on the top and bottom side only. Multiple controllers sharing the DLL must have the same frequency, but can have different implementations (that is, 1-PLL or 2-PLL implementation). You can implement multiple controllers with up to two different frequencies per device as each DLL can run at different frequencies.

Figure 69 shows a summary of how Stratix II and Stratix II GX devices generate the DQ, DQS, CK, and CK# signals. The write PLL generates the system clock and the –90× shifted clock (write clock). If the write PLL input clock and the DDR2 SDRAM frequencies are different, you must provide the input reference clock to the DQS phase-shift circuitry either from another input clock pin or from PLL 5 or PLL 6.

For more information about the input clock, refer to the *External Memory Interfaces* chapter in volume 2 of the Stratix II Device Handbook.

The system clock and write clock are the same frequency as the DQS frequency. The write clock is shifted -90° from the system clock.

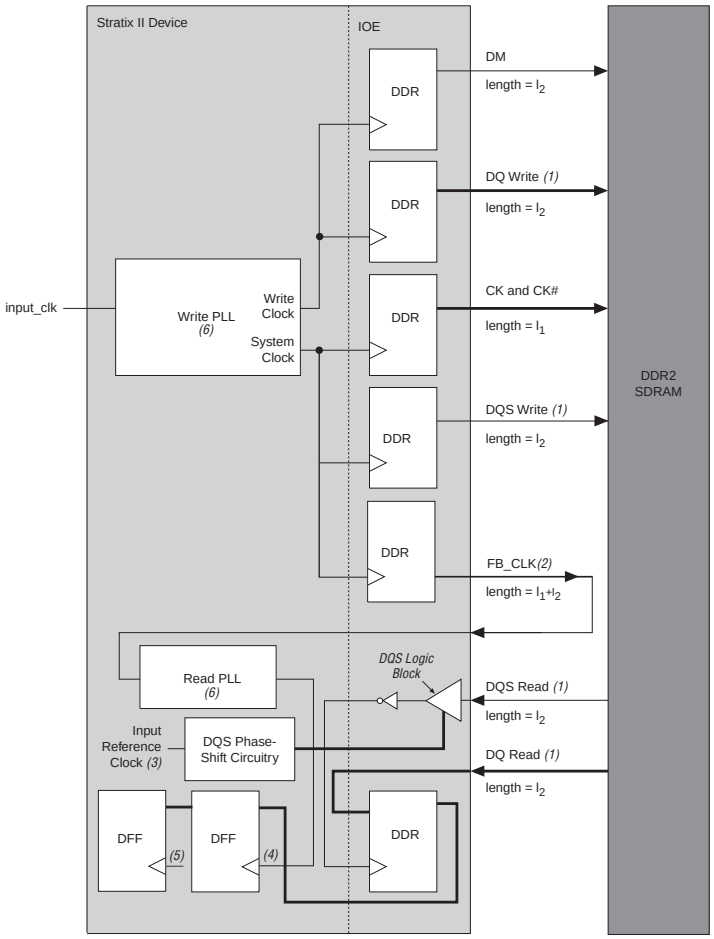
The `areset` signal of the read PLL must be toggled after power up and the system PLL is locked.

Reset the PLL if the input clock to the PLL is not stable during power up, after a clock switch over, or if the PLL loses lock after power up.

The example in Figure 69 uses a feedback clock and a second (read) PLL to ease resynchronization. The board trace length for the feedback clock must match closely with the board trace lengths for the CK, DQ, and DQS signals to compensate for off-chip voltage and temperature variation.

In Figure 69, the `fedback_clk_out` pin mirrors the DDR2 SDRAM CK pin with a board trace length of l_1 and is routed back to the Stratix II device with a board trace length of l_2. The controller generates the `FB_CLK` output using the same circuit as the CK output and constrains this pin to an adjacent location on the same I/O bank.

Figure 69. Stratix II Device and DDR2 SDRAM Interface Data Path for DLL-Based Read



Notes to Figure 69:

(1) The DQ and DQS signals are bi-directional. One DQS signal is associated with a group of DQ signals.

(2) The feedback clock, FB_CLK, is to help ease resynchronization for interface speeds > 200 MHz.

(3) The input reference clock can either be from input_clk, another clock pin, or a PLL 5 or PLL 6 output.

(4) The clock to the resychronization register is either from the system clock, write clock, and extra clock output from the write PLL, or from the read PLL.

(5) The clock to this register is either from the system clock or another clock output of the write PLL. If another clock output of the write PLL is needed, another register is needed to transfer the data back to the system clock domain.

(6) The read and write PLLs are configured in normal mode.

(7) The average values of I_1 and I_2 must be used for FB_CLK trace length.

The Stratix II device also has another legacy DLL-based implementation where only one PLL is used. This PLL generates the resynchronization and post amble clocks, in addition to the system and write clocks. This implementation differs from Figure 69 in that the fedback_clk trace and the read PLL module does not exist. In general, this implementation is limited to 200 MHz; however, this depends on the FPGA density and memory device speed grade used. Always perform timing analysis in the Quartus II software for your design to ensure that performance can be met.

Legacy Data Path Architecture Without Using Dedicated DQS Phase-Shift Circuitry

When using the PLL-based read implementation, Stratix II devices can support up to 200-MHz DDR2 SDRAM where each interface uses two PLLs for best performance (Figure 69 on page 105).

This implementation is useful when interfacing with more than 72-bit wide memory interfaces per Stratix II device side or when interfacing with more than 144-bit data bus.

The DDR2 SDRAM interface implementation without using dedicated DQS phase-shift circuitry uses the following:

- A write-side PLL to generate CK and CK# system clocks and clock out address, command, strobe, and data signals.
- A read-side PLL-based phase-shift to register read data from the memory using a feedback clock, FB_CLK.

This implementation is also called PLL-based read implementation (or non-DQS mode).

For the recommended DQS and DQ pins in this mode, refer to the Stratix II device pin-out table. The board trace lengths for the DQ, DQS, and DM pins must be tightly matched.

In this implementation, the write PLL generates the system clock, the -90° shifted clock, and the feedback clock (FB_CLK). The feedback clock is routed outside the FPGA and back into the FPGA. This board trace length should equal the clock trace length from the FPGA to the memory plus the DQ trace length from the memory to the FPGA. In Figure 70, assuming that the clock trace length equals l_1 and the DQ trace length equals l_2, FB_CLK must have a trace length of $l_3 = (l_1 + l_2)$. Use an average value of l_1 and l_2 when calculating l_3.

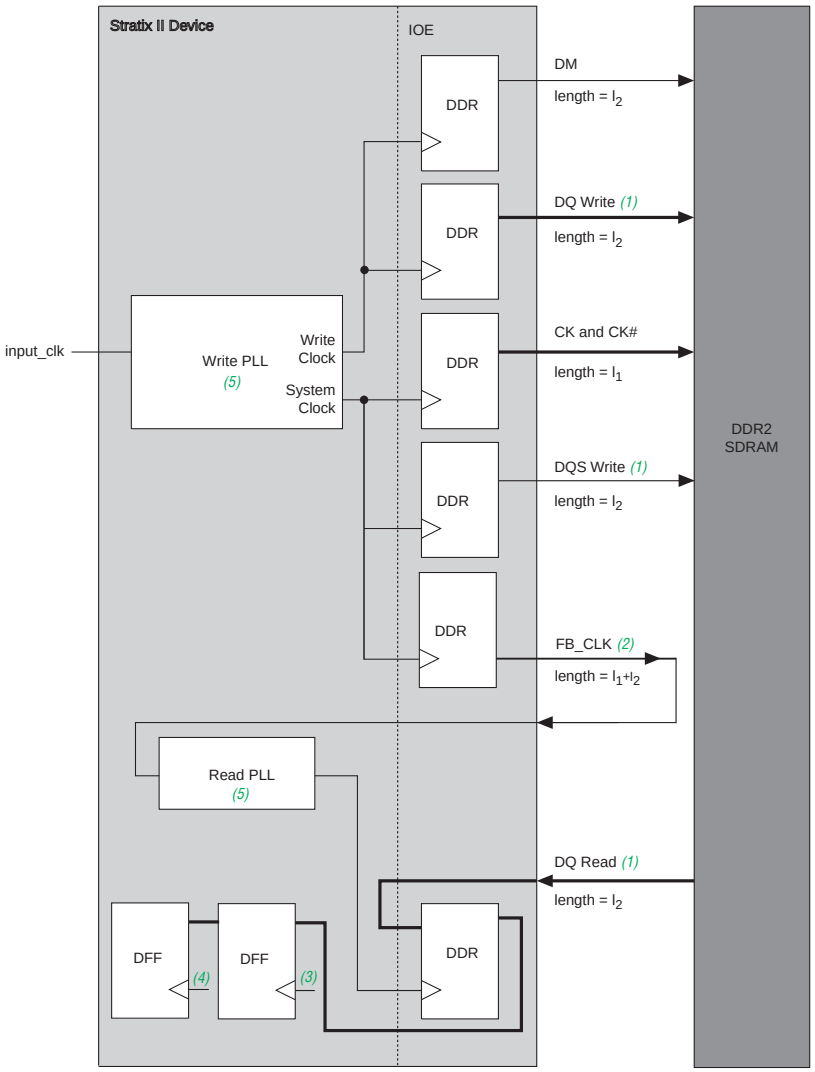
The read PLL uses the feedback clock as the input clock and generates the clock needed to capture the DQ during reads. The Stratix II device ignores the DQS signal in this scheme. The read PLL is in source-synchronous mode, such that the clock propagation delay to the IOE register is matched to the data propagation. Because the trace length of the feedback clock is the same as the CK or CK# and DQS trace, FB_CLK coming into the FPGA resembles the DQS signal with a small amount of skew. The read PLL can then be shifted to compensate for the skew and to implement the 90° PLL phase shift required to capture the DQ signals during reads.

The Quartus II software associates a particular PLL with a particular I/O bank for source-synchronous operation, so you must ensure that the read PLL is on the same side of the device as the data pins. The clock delay to the worst case I/O registers in this I/O bank are fully compensated and result in closely matched data delays and clock delays from the pin to the I/O registers across PVT. When using I/O registers in the non-compensated I/O banks, clock delays and data delays are less closely matched. For best clock and data delay matching, use a fast PLL for implementing the interface on side I/O banks and use an enhanced PLL for implementing the interface on the top or bottom I/O banks. You must also set the input pin delay-to-register option to **0** in the Quartus II software.

The DQS signal is ignored during read, so the DQS-DQ relationship from the DDR SDRAM component no longer applies. The skew and timing variations on the interface determine the maximum data rate you can achieve with this method.

Figure 70 shows a summary of how Stratix II devices generate the DQ, DQS, CK, and CK# signals. The write PLL generates the system and write clocks. The read clock (FB_CLK) from the DDR2 SDRAM component goes to a PLL input pin that generates the proper phase shift to capture read data.

Figure 70. Stratix II and DDR2 SDRAM Interface Data Path for PLL-Based Read



Notes to Figure 70:

(1) The DQ and DQS signals are bi-directional. One DQS signal is associated with a group of DQ signals.

(2) The feedback clock, FB_CLK, is to help ease resynchronization.

(3) The clock to the resynchronization register is either from the system clock, write clock, or an extra clock output from the write PLL.

(4) The clock to this register is either from the system clock or another clock output of the write PLL. If another clock output of the write PLL is needed, another register is needed to transfer the data back to the system clock domain.

(5) The read PLL is configured in the source synchronous mode, while the write PLL is configured in the normal mode.

Appendix D: Interface Timing Analysis

When designing an external memory interface for your FPGA, you must analyze timing margins for several paths. All memory interfaces require analysis of the write and read capture timing paths. Additionally, some interfaces might require analysis of the resynchronization timing paths and other memory-specific paths (such as postamble timing).

This application note describes Altera's recommended timing methodology using write and read capture timing paths as examples. You must use this methodology for analyzing timing for all applicable timing paths (including address and command, resynchronization, and postamble). To ensure successful operation, the Altera DDR2 SDRAM controller MegaCore performs timing analysis on the read capture, resynchronization, and postamble paths. While these analyses account for all FPGA related timing effects, you must design in adequate margin to account for board-level effects, such as crosstalk, inter-symbol interference, and other noise effects. These effects and their impact on timing margins are best analyzed by performing board-level simulations. You can use simulation results to adjust the board timing requirement parameter, t_{EXT}, for use in margin calculations. Your overall system performance is determined by the slowest of all the timing paths.

This section analyzes the write and read capture timing margins for a DDR2 SDRAM interface with a Stratix II device. The timing analysis methodology is illustrated using the EP2S60F1020C3 FPGA interfacing with a Micron MT9HTF3272AY-53E DIMM. However, you must use the same methodology to analyze timing for your preferred FPGA and memory device.

Methodology Overview

Timing paths are analyzed by considering the data and clock arrival times at the destination register. In Figure 71 and Figure 72, the setup margin is defined as the time between the "earliest clock arrival time" and the "latest valid data arrival time" at the register ports. Similarly, hold margin is defined as the time between the "earliest invalid data arrival time" and the "latest clock arrival time" at the register ports. These arrival times are calculated based on propagation delay information with respect to a common reference point (such as a DQS edge or system clock edge).

Figure 71. Simplified Block Diagram for Timing Analysis

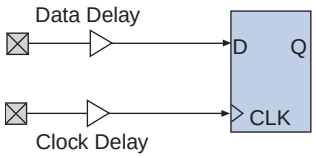
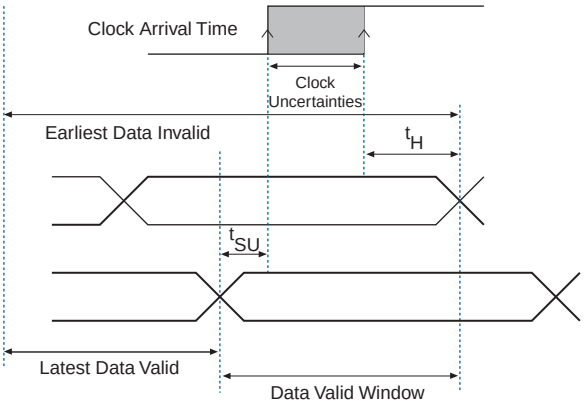


Figure 72. Data Valid Window Timing Waveform



FPGA Timing Information

Because you expect your design to work under all conditions, timing margins must be evaluated at all process, voltage, and temperature (PVT) conditions. To facilitate this, Altera provides two device timing models in the Quartus II software—the slow corner model and the fast corner model.

- Slow Corner Model—provides timing delays between two nodes in the FPGA with slow silicon, high temperature, and low voltage. In other words, the model provides the slowest possible delay for that timing path on any device for that particular speed grade.
- Fast Corner Model—provides timing delays between two nodes in the FPGA with fast silicon, low temperature, and high voltage. In other words, the model provides the fastest possible delay for that timing path on any device.

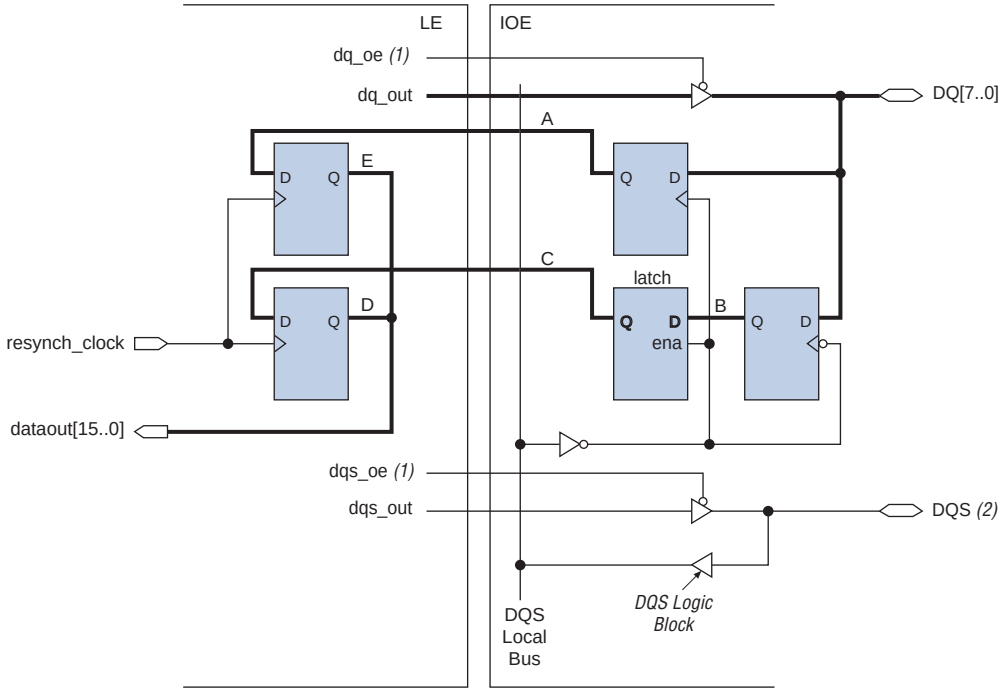
Note that while almost all FPGA timing delays and uncertainties are modeled in the Quartus II software, a limited number of uncertainties that cannot be modeled are published in the FPGA handbook for use in margin calculations. These data sheet specifications are based on device characterization and account for device-level effects such as on-chip variation, rise and fall mismatch, and noise. Some examples include clock jitter on PLL and DLL outputs. These timing uncertainties or adder terms, when used in conjunction with the Quartus II software reported timing data, provide the most accurate device timing information. The following read and write timing analyses sections describes the use of these timing adder terms.

Read Timing Margins for a DLL-Based Implementation

During read operations, the DDR2 SDRAM provides a clock strobe (DQS) that is edge-aligned with the data bus (DQ). The memory controller (in the FPGA) is required to shift the clock edge to the center of the data valid window and capture the DQ input data. Figure 66 on page 100 shows the timing relationship between the DQS and DQ signals during a read operation.

Figure 73 shows the Stratix II device read data path for ×8 mode. The DQS signal goes to the DQS logic block and is shifted by 90°. The DQS local bus then inverts the shifted DQS signal before it clocks the DQ at the input registers. The DQ input register outputs then go to the resynchronization register in the logic array. The `resynch_clock` signal clocks the resynchronization register. The `resynch_clock` signal can come from the system clock, the write clock, the write PLL clock, or the second (read) PLL (if you use the feedback clock scheme).

Figure 73. DDR2 SDRAM Read Data Path in Stratix II Devices



Notes to Figure 73:

(1) The output enable registers are not shown here, but `dqs_oe` and `dq_oe` are active low in silicon. However, the Quartus II software implements it as active high and adds the inverter automatically during compilation.

(2) Figure 75 does not show the DQS postamble circuitry.

For more information about the DQ and DQS paths, refer to the *External Memory Interfaces* chapter in volume 2 of the *Stratix II Device Handbook*.

Memory Timing Parameters

You start the read timing analysis by obtaining the timing relationship between the DQ and DQS outputs from the DDR2 SDRAM memory device. Because this example analyzes timing for 267 MHz clock speeds or 533 Mbps data rates, the half clock period is 1688 ps after accounting for duty cycle distortion on the DQS strobe. This is specified as t_{HP} in the memory data sheet and is 45% of the 3750 ps clock period. Aside from t_{HP}, the memory also specifies t_{DQSQ} and t_{QHS}. The former specifies the maximum time from a DQS edge to the last DQ bit valid and the latter specifies the data hold skew factor.

With these memory timing parameters, the data valid window at the memory can be calculated as $t_{HP} - t_{QHS} - t_{DQSQ} = 988$ ps. Assuming the board trace length variations between all DQ and DQS traces are not more than ± 20 ps, the data valid window present at the FPGA input pins is 948 ps.

FPGA Timing Parameters

FPGA timing parameters are obtained from two sources— the Quartus II software timing analyzer and the Stratix II data sheet. The Quartus II software provides all clock and data propagation delays, and the data sheet specifies all clock uncertainties and skew adder terms.

Stratix II devices feature dedicated DQS phase-shift circuitry in the top and bottom I/O banks of the device, which center-aligns the DQS edge with respect to the DQ input signals. This phase shift circuitry has a coarse and fine delay resolution. The coarse delay feature is self-compensating over PVT and has a resolution of 22.5°, 30°, or 36° of the reference clock frequency (based on the DLL mode of operation).

For more information about DLL operation, refer to the *External Memory Interfaces* chapter in volume 2 of the *Stratix II Device Handbook*.

The target memory speed is 267 MHz, so you can select between DLL modes 2 (high) and 3 (very high). DLL mode 2 provides a coarse phase resolution of 30°; mode 3 provides a 36° resolution. You can fine-tune this phase shift further with a DLL offset implemented using uncompensated delay chains.

This analysis assumes a 90° phase shift on the DQS strobe (DLL mode 2), knowing that the phase shift (and DLL mode) can always be adjusted at the end of this timing analysis for balanced setup and hold margins on the read capture register.

The DQS phase-shift circuitry uses a DLL to provide the self-compensating coarse delay shift. You must account for any jitter and phase-shift error on the DQS signal. The data sheet specifies the $t_{DQS\_PHASE\_JITTER}$ (± 45 ps) and t_{DQS_PSERR} (± 38 ps) timing parameters for DLL mode 2.

After encountering the phase-shift circuitry, the DQS signal travels on a dedicated local clock bus to the DQ capture registers. The fanout of this local clock bus could range from ×4 to ×36. While the Quartus II software provides clock propagation delays to each of these DQ register clock, un-modeled uncertainties are accounted for with the $t_{DQS\_SKEW\_ADDER}$ skew adder term listed in the data sheet. For the ×8 mode used in this example, the skew adder is ± 35 ps.

To obtain the Quartus II software timing data for the target device, you must instantiate and compile the DDR2 SDRAM Controller MegaCore. If you are using your own controller logic, you must instantiate the clear-text DDR2 SDRAM data path instead to obtain timing delays. For the read interface, the MegaCore function extracts and reports timing delays associated with each DQ and DQS pin in the **<*core_instance_name*>_extraction_data.txt** file located in your project directory. Using this data file and the **extract.tcl** utility, minimum and maximum propagation delays on the clock and data path are extracted and presented in Table 18. This timing extraction is performed twice, once with each device model (fast corner and slow corner). Observe that the difference between the minimum and maximum delays is minimal because of the matched routing paths in the die and package.

Table 18. FPGA Timing Delays *(Note 1)*

	Fast Corner (ns)	Slow Corner (ns) (-3 Speed Grade)
Data delay (minimum)	1.113	1.698
Data delay (maximum)	1.173	1.758
Clock delay (minimum)	2.031	2.612
Clock delay (maximum)	2.051	2.632
Micro setup *(2)*	0.068	0.122
Micro hold *(2)*	0.037	0.072

Notes to Table 18:

(1) These delays are reported in the **<*core_instance_name*>_extraction_data.txt** file located in your project directory. Data delay is the propagation delay from each DQ pin to the input DDR register and is reported as `dq_2_ddio`. Clock delay is the propagation delay to the DDR input registers from the corresponding DQS pin and is calculated as `dqspin_2_dqsclk` + `dqsclk_2_ddio_resync`.

(2) The micro setup and micro hold times are specified in the *DC Switching Characteristics* chapter in volume 1 of the *Stratix II Device Handbook*.

Setup and Hold Margins Calculations

After obtaining all relevant timing information from the memory, FPGA, and board, you can calculate the setup and hold margins at the DQ capture register during read operations.

With extracted timing information from the FPGA slow corner model:

- Earliest clock arrival time = minimum clock delay within
- (t_{EARLY_CLOCK})FPGA – DQS uncertainties = clock delay (minimum) – $t_{DQS\_PHASE\_JITTER}$ – t_{DQS_PSERR} – $t_{DQSQINT}$ = 2612 – 45 – 37.5 – 35= 2495 ps
- Latest data valid time = memory DQS-to-DQ valid + maximum ($t_{LATE\_DATA\_VALID}$)data delay in FPGA= t_{DQSQ} + data delay (maximum) = 300 + 1758 = 2058 ps
- Setup margin = earliest clock arrival – latest data valid – micro setup – board uncertainty = t_{EARLY_CLOCK} – $t_{LATE\_DATA\_VALID}$ – μt_{SU} – t_{EXT} = 2495 – 2058 – 122 – 20 = 295 ps
- Therefore, the setup margin with the slow corner timing model is 230 ps. Repeating these calculations with the fast corner timing model, the setup margin is calculated to be 353 ps.
- Latest clock arrival time = maximum clock delay within (t_{LATE_CLOCK})FPGA + DQS uncertainties = clock delay (maximum)+ $t_{DQS\_PHASE\_JITTER}$+ t_{DQS_PSERR}+$t_{DQSQINT}$ = 2632 + 45 + 37.5 + 35 = 2750 ps

- Earliest data invalid time = memory DQS-to-DQ invalid + ($t_{EARLY\_DATA\_INVALID}$)minimum data delay in FPGA = ($t_{HP} - t_{QHS}$) + data delay (minimum) = (1688 – 400) + 1698 = 2986 ps
- Hold margin = latest clock arrival time – earliest data invalid time – micro hold – board uncertainty = $t_{EARLY\_DATA\_INVALID} - t_{LATE\_CLOCK} - \mu t_H - t_{EXT}$ = 2986 – 2750 – 72 – 20 = 144 ps

Therefore, the hold margin with the slow corner timing model is 144 ps. Repeating these calculations with the fast corner model, the hold margin is calculated to be 175 ps. The setup and hold margins can be balanced by using a 72° phase shift in DLL mode 3, or a 60° phase shift in DLL mode 2 along with a positive delay offset. Table 19 lists timing analysis for DLL-based read operations from a DDR2 SDRAM memory.

Table 19. Read Timing Analysis Example for 267-MHz DDR2 SDRAM Interface in EP2S60F1020C3 Using the Dedicated DQS Phase-Shift Circuitry (Part 1 of 2)

Parameter	Specifications	Fast Corner Model	Slow Corner Model	Description
Memory specifications *(1)*	t_{HP}	1.688	1.688	Half period as specified by the memory data sheet (including memory clock duty cycle distortion)
	t_{DQSQ}	0.300	0.300	Skew between DQS and DQ from the memory
	t_{QHS}	0.400	0.400	Data hold skew factor as specified by the memory data sheet
FPGA specifications *(2)*, *(3)*, *(4)*	$t_{DQS\_PHASE\_JITTER}$	0.045	0.045	Phase jitter on DQS output delayed by DLL (three delay stages = ± 3 × 15)
	t_{DQS_PSERR}	0.038	0.038	Phase-shift error on DQS output delayed by DLL (three delay stages)
	$t_{DQS\_SKEW\_ADDER}$	0.035	0.035	Clock Delay Skew Adder for x8
	Minimum clock delay (input)	2.031	2.612	Minimum DQS pin to IOE register delay from the Quartus II software (with 90° DLL-based phase shift)
	Maximum clock delay (input)	2.051	2.632	Maximum DQS pin to IOE register delay from the Quartus II software (with 90° DLL-based phase shift)
	Minimum data delay (input)	1.113	1.698	Minimum DQ pin to IOE register delay from the Quartus II software
	Maximum data delay (input)	1.173	1.758	Maximum DQ pin to IOE register delay from the Quartus II software
	μt_{SU}	0.068	0.122	Intrinsic setup time of the IOE register
	μt_H	0.037	0.072	Intrinsic hold time of the IOE register
Board specifications	t_{EXT}	0.020	0.020	Board trace variations on the DQ and DQS lines

Table 19. Read Timing Analysis Example for 267-MHz DDR2 SDRAM Interface in EP2S60F1020C3 Using the Dedicated DQS Phase-Shift Circuitry (Part 2 of 2)

Parameter	Specifications	Fast Corner Model	Slow Corner Model	Description
Timing calculations	t_{EARLY_CLOCK}	1.914	2.495	Earliest possible clock edge after DQS phase-shift circuitry and uncertainties (minimum clock delay – t_{DQS_JITTER} – t_{DQS_PSERR} – $t_{DQS\_SKEW\_ADDER}$)
	t_{LATE_CLOCK}	2.169	2.750	Latest possible clock edge after DQS phase-shift circuitry and uncertainties (maximum clock delay + t_{DQS_JITTER} + t_{DQS_PSERR} + $t_{DQS\_SKEW\_ADDER}$)
	$t_{EARLY\_DATA\_INVALID}$	2.401	2.986	Time for earliest data to become invalid for sampling at FPGA flop (t_{HP} – t_{QHS} + minimum data delay)
	$t_{LATE\_DATA\_VALID}$	1.473	2.058	Time for latest data to become valid for sampling at FPGA flop (t_{DQSQ} + maximum data delay)
Results	Read setup timing margin *(3)*	0.353	0.295	t_{EARLY_CLOCK} – $t_{LATE\_DATA\_VALID}$ – μt_{SU} – t_{EXT}
	Read hold timing margin *(3)*	0.175	0.144	$t_{EARLY\_DATA\_INVALID}$ – t_{LATE_CLOCK} – μt_{H} – t_{EXT}
	Total margin	0.528	0.439	Setup margin + hold margin

Notes to Table 19:

(1) The memory specifications are based on the Micron MT9HTF3272AY-53E DIMM data sheet.

(2) This analysis is performed with FPGA timing parameters for the EP2S60F1020C3 device. You must use this template to analyze timing for your preferred Stratix II density-package combination. For FPGA specifications, see the External Memory Interfaces section in the *DC Switching Characteristics* chapter in volume 1 of the *Stratix II Device Handbook*.

(3) These numbers are from the Quartus II software, version 5.1 using the DDR2 SDRAM Controller MegaCore 3.3.0.

(4) Package trace skews are modeled by the Quartus II software.

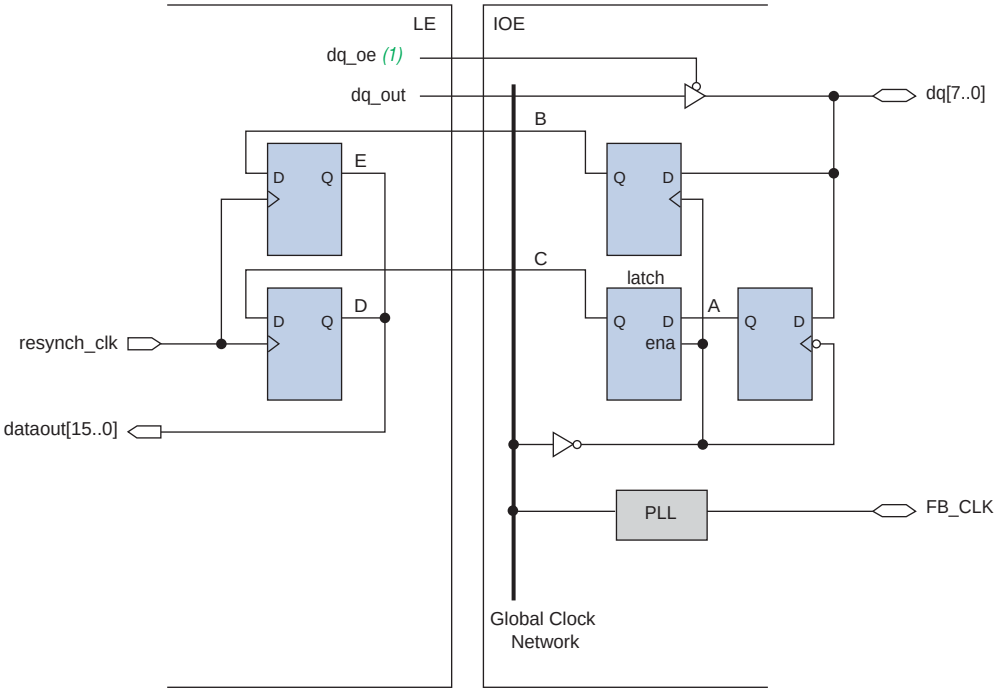
Read Timing Margins for PLL-Based Implementation

Timing margin analysis for a PLL-based implementation is very similar to the previously described DLL-based implementation. The only differences are the capture clock used and related clock uncertainties. In this mode, a copy of the CK clock signal is fed back to a PLL inside the FPGA.

In this example, you analyze margins for a DDR2-400 memory device for a 200 MHz read operation using a PLL.

Figure 74 shows the PLL-based read data path in a Stratix II device. The FB_CLK signal goes to the PLL and the PLL generates a phase-shifted read clock to capture the read data. The outputs of the DQ registers then go to the LE resynchronization registers. You may need multiple resynchronization registers before data is synchronized with the system clock.

Figure 74. PLL-Based Read Data Path in a Stratix II Device



Note to Figure 74:

(1) The dq_oe signals are active-low in silicon. However, the Quartus II software adds the inverter automatically during compilation.

Memory Timing Parameters

The timing relationship of data (DQ) with respect to the CK clock is governed by the t_{AC} parameter. For the DDR2-400 memory device under consideration, this timing parameter is ± 600 ps. This memory parameter replaces the t_{DQSQ} and t_{QHS} parameters used in the DLL-based implementation.

FPGA Timing Parameters

When the CK clock is fed back into the PLL for read capture, uncertainties introduced on this clock include jitter, phase-shift error, and compensation error. While jitter and phase-shift error parameters have been defined before, the compensation error is a measure of the PLL's ability to regenerate a clock output that tracks the input reference. For the source-synchronous mode of the PLL, this parameter is typically $t_{PLL\_COMP\_ERROR}$ = ± 100 ps.

For more information about PLL specifications, refer to the *PLLs in Stratix II Devices* chapter in volume 2 of the *Stratix II Handbook*.

PLL-based read implementation uses a single global clock network to distribute the phase shifted clock signal to DQ capture registers in the IOE. Differences in clock arrival times to these registers (clock skew) are modeled in the Quartus II software and is reflected in the minimum or maximum propagation delays for the clock. Additionally, the Quartus II software models the package trace delays for every pin in the device. Therefore, this analysis does not account for such skews separately in the timing margin analysis. The extracted minimum or maximum clock and data delays account for these uncertainties.

To obtain the Quartus II software timing data for the target device, instantiate and compile the DDR2 SDRAM controller MegaCore function. If you are using your own controller logic, instantiate the clear text DDR2 data path instead to obtain timing delays. For the read interface, the Quartus II software reports individual setup and hold times for each DQ pin. In the timing report, select the **List Paths** option to get the data and clock propagation delays for that DQ pin. Select the worst-case setup and hold DQ registers to extract the minimum and maximum propagation delays.

For example, the list paths example shown in "List Paths Example" indicates the setup time for `DQ[9]`. This path shows propagation delays of 0.964 ns on the DQ pin to register path and 2.148 ns (-0.750 + 2.898) on the `FB_CLK` clock pin to the register path. Note that the clock path includes the PLL phase-shift delay of 75° or 1.042 ns. Using this approach, minimum and maximum propagation delays on the clock and data path are extracted (refer to Table 20). This timing extraction is performed twice, once with each device model (fast model and slow model).

Example 1. List Paths Example

```
Info: tsu for register
"ddr2_nondqs:ddr2_nondqs_ddr_sdram|...|\g_dq_io:1:dq_io~ddio_out_reg"
(data pin = "ddr2_dq[9]",
clock pin = "fedback_clk_in") is -1.062 ns
Info: + Longest pin to register delay is 0.964 ns
Info: + Micro setup delay of destination is 0.122 ns
Info: - Offset between input clock "fedback_clk_in" and output clock
"ddr_pll_fb_stratixii:g_stratixpll_ddr_fedback_pll_inst|altpll:altpll_
component|_clk0" is -0.750 ns
Info: - Shortest clock path from clock
"ddr_pll_fb_stratixii:g_stratixpll_ddr_fedback_pll_inst|altpll:altpll_
component|_clk0"
to destination register is 2.898 ns
```

Table 20 shows the read capture margins for a PLL-based implementation of 200 MHz. The IP Toolbench utility performs a similar timing margin analysis for the read timing path of your DDR2 SDRAM Controller MegaCore instance.

Table 20. Read Timing Analysis Example for 200-MHz DDR2 SDRAM Interface in EP2S60F1020C3 (PLL Based Read) (Part 1 of 2)

Parameter	Specification	200-MHz Fast Model	200-MHz Slow Model	Description
Memory specifications *(1)*	t_{HP}	2.250	2.250	Half period as specified by the memory data sheet
	t_{AC}	0.600	0.600	Data (DQ) output access time from memory clock (CK) for DDR2 400 Mbps device

Table 20. Read Timing Analysis Example for 200-MHz DDR2 SDRAM Interface in EP2S60F1020C3 (PLL Based Read) (Part 2 of 2)

Parameter	Specification	200-MHz Fast Model	200-MHz Slow Model	Description
FPGA specifications	PLL phase shift *(3)*	1.042	1.042	PLL phase shift to capture data (this is using 75°)
	t_{PLL_JITTER} *(2)*	0.125	0.125	Stratix II PLL jitter
	$t_{PLL\_COMP\_ERROR}$	0.100	0.100	PLL compensation error (high bandwidth)
	t_{PLL_PSERR} *(2)*	0.015	0.015	PLL phase-shift error
	t_{CO_SKEW} (CK and FB_CLK)	0.025	0.025	Variations of the CK and feedback clock signals clock-to-out times from the Quartus II software *(5)*
	Minimum clock delay *(4)*, *(6)*	0.840	1.063	Minimum feedback clock pin to IOE register delay from the Quartus II software
	Maximum clock delay *(4)*, *(6)*	0.861	1.106	Maximum feedback clock pin to the IOE register delay from the Quartus II software
	Minimum data delay *(4)*, *(6)*	0.611	0.964	Minimum DQ pin to the IOE register delay from the Quartus II software
	Maximum data delay *(4)*, *(6)*	0.705	1.060	Maximum DQ pin to the IOE register delay from the Quartus II software
	μt_{SU}	0.068	0.122	Intrinsic setup time of the IOE register
	μt_{H}	0.037	0.072	Intrinsic hold time of the IOE register
Board specifications	t_{EXT_SKEW}	0.020	0.020	Board trace variations on the DQ and DQS lines
Timing calculations	t_{EARLY_CLOCK}	1.642	1.865	Earliest possible clock edge after DQS phase-shift circuitry and uncertainties (minimum clock delay + PLL phase shift – t_{PLL_PSERR} – t_{PLL_JITTER} – $t_{PLL\_COMP\_ERROR}$)
	t_{LATE_CLOCK}	2.143	2.388	Latest possible clock edge after DQS phase-shift circuitry and uncertainties (maximum clock delay + PLL phase shift + t_{PLL_PSERR} + t_{PLL_JITTER} + $t_{PLL\_COMP\_ERROR}$)
	$t_{EARLY\_DATA\_INVALID}$	2.236	2.589	Time for earliest data to become invalid for sampling at FPGA flop (t_{HP} – t_{AC} + minimum data delay – t_{CO_SKEW})
	$t_{LATE\_DATA\_VALID}$	1.330	1.685	Time for latest data to become valid for sampling at FPGA flop (t_{AC} + maximum data delay + t_{CO_SKEW})
Results	Read setup timing margin	0.224	0.038	t_{EARLY_CLOCK} – $t_{LATE\_DATA\_VALID}$ –μt_{SU} – t_{EXT}
	Read hold timing margin	0.036	0.109	$t_{EARLY\_DATA\_INVALID}$ – t_{LATE_CLOCK} –μt_{H} – t_{EXT}
	Total margin	0.260	0.147	Setup + hold-time margin

Notes to Table 20:

(1) Specifications are based on the 200 MHz Micron MT9HTF3272AG-40E data sheet.

(2) For FPGA PLL specifications, refer to the PLL Timing Specifications section of the *DC Switching Characteristics* chapter in volume 1 of the Stratix II Device Handbook.

(3) PLL phase shift is adjustable if you must balance the setup and hold time margin.

(4) Package trace length skew is modeled in the Quartus II software version 5.0 and higher. No additional adder is required.

(5) This number represents the difference between the Quartus II software reported t_{CO} for your CK and feedback clocks. The value used in your analysis can be minimized by choosing pins with closely matched t_{CO}, such as adjacent pins.

(6) These numbers are from the Quartus II software, version 5.1 using the DDR2 SDRAM Controller MegaCore 3.3.0.

Write Data Timing Analysis

Whether you are using the DQS phase-shift circuitry or the PLL to capture the data during a read operation from the DDR2 SDRAM component, there is only one implementation for the write operation. Timing margin analysis for write data and address and command signals are very similar. This section analyzes timing for the write data signals. You must use the same approach to repeat this for the address and command signals.

For write operations, the DDR2 SDRAM memory requires the DQS to be center-aligned with the DQ. This is implemented in the Stratix II device using the PLL phase shift feature. Two output clocks are created from the PLL, with a relative 90° phase offset. The leading (–90°) clock edge clocks out the DQ write data output pins to the memory, while the lagging (0°) clock edge generates the DQS clock strobe and CK and CK# memory output clocks. Figure 66 on page 100 shows the timing relationship between the DQS and DQ inputs required by the memory during a read operation.

Figure 67 on page 100 shows the DQ, DQS timing relationship. The write side uses a PLL to generate the clocks listed in Table 21.

Table 21. PLL Clock Outputs

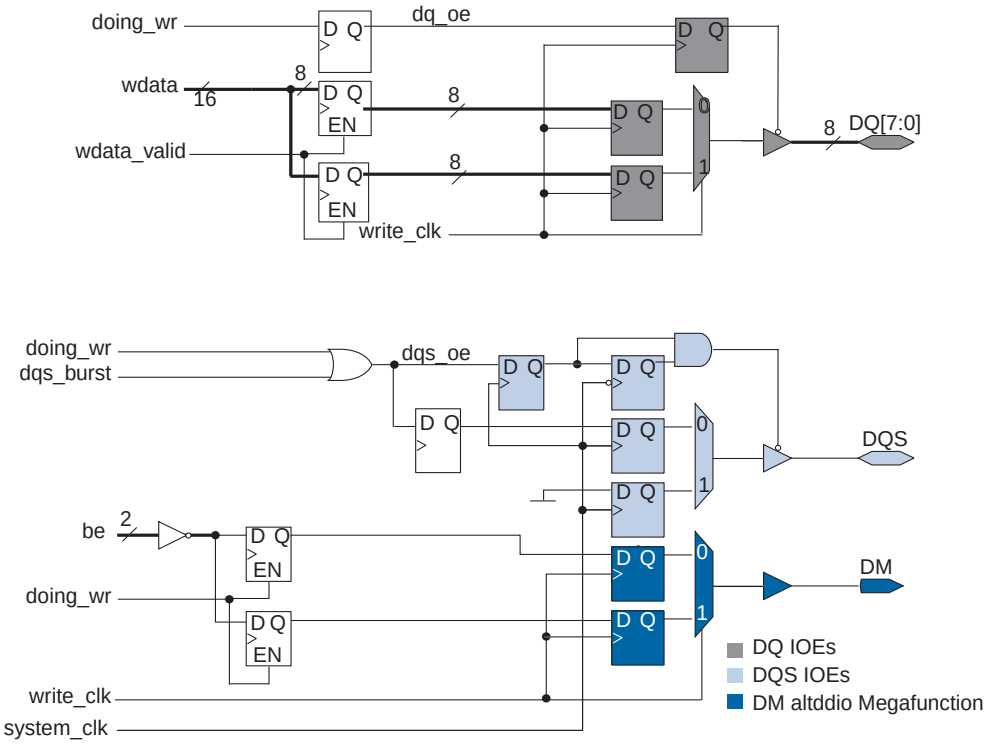
Clock	Description
System clock	Used for the memory controller and to generate the DQS write, CK, and CK# signals.
Write clock (–90° shifted from system clock)	Used in the data path to generate the DQ write signals.
Feedback clock	Optional clock used if you are not using the DQS phase-shift circuitry when reading from the DDR2 SDRAM component or if you are using the feedback clock scheme for resynchronization.
Resynchronization clock	Optional clock only used if you are using the DQS phase-shift circuitry and need a different clock phase shift than available for resynchronization.

Memory Timing Parameters

When writing to a specific memory, the FPGA must ensure that setup and hold times are met. Obtain these specifications (t_{DS} and t_{DH}) from the data sheet (for the 267 MHz DDR2 SDRAM example, 350 ps and 350 ps, respectively). Additionally, the FPGA must provide a memory clock (CK and CK#) that meets the clock high and low time specifications. And finally, the skew between the DQS output strobe and CK output clock cannot exceed limits set by the memory. While the last parameter does not directly affect timing margins, it must be met for successful memory operation.

Figure 75 shows the DDR2 SDRAM write data path in Stratix II devices.

Figure 75. DDR2 SDRAM Write Data Path for Stratix II Devices *(Note 1), (2)*



Notes to Figure 75:

(1) This figure shows the logic for one DQ output only. A complete byte group consists of four or eight instances of the DQ logic with the DQS and DM logic.

(2) All clocks are `system_clk` unless marked otherwise.

FPGA Timing Parameters

The timing paths in the FPGA for the DQ and DQS outputs to memory are matched by data path design. Dedicated clock networks drive double-data rate I/O structures to generate DQ and DQS. This results in minimal skew between these outputs. These skew parameters include phase-shift error, clock skew, and package skew.

The two clock networks used are driven by the same PLL, but with a 90° relative phase shift. The 0° clock generates DQS, while a – 90° clock generates DQ. Typical PLL uncertainties, such as jitter and compensation error, affect both clock networks equally. Therefore, these timing parameters do not affect write timing margins. As the clock generating DQ is phase shifted, the PLL phase-shift uncertainty (t_{PLL_PSERR} = ± 15 ps, listed in the *DC Switching Characteristics* chapter in volume 1 of the *Stratix II Device Handbook*) affects DQ arrival times at the memory pins.

The Quartus II software models intra-clock skew; that is, skew between nodes driven by the same dedicated clock network. However, skew between two such clock networks is not modeled and specified in the data sheet as an adder term. You must add this skew component to the propagation delays extracted from the Quartus II software.

For a 72-bit DDR2 SDRAM interface that spans two I/O banks in the top or bottom of the device, the clock skew adder between two clock networks is specified as ± 50 ps ($t_{CLOCK\_SKEW\_ADDER}$). This uncertainty is used when calculating DQS arrival times at the memory pins.

The final skew component is package skew. As noted earlier, the Quartus II software models package trace delay for each pin on the device. Extracted propagation delays reflect any skew between output signals to the memory.

Table 22 shows the timing analysis for write operations to a DDR2 SDRAM memory device.

Table 22. Write Timing Analysis Example for 267-MHz DDR2 SDRAM Interface in EP2S60F1020C3 (Part 1 of 2) *(Note 1)*

Parameter	Specification	Fast Corner Model	Slow Corner Model	Description
Memory specifications *(2)*	t_{DS}	0.395	0.395	Memory data setup requirement
	t_{DH}	0.335	0.335	Memory data hold requirement
FPGA specifications *(1)*	t_{HP}	1.487	1.487	Ideal half-period time minus 5% duty cycle distortion and half-period jitter
	t_{PLL_JITTER}	0.000	0.000	Does not affect margin as the same PLL generates both write clocks (0° and –90°)
	t_{PLL_PSERR}	0.015	0.015	PLL phase-shift error (on –90° clock output)
	$t_{CLOCK\_SKEW\_ADDER}$	0.050	0.050	Clock skew between two dedicated clock networks feeding IO banks on the same side of the FPGA
	Minimum clock delay (output) *(3)*, *(4)*	0.925	1.864	Minimum DQS t_{CO} from the Quartus II software (0° PLL output clock)
	Maximum clock delay (output) *(3)*, *(4)*	0.960	1.909	Maximum DQS t_{CO} from the Quartus II software (0° PLL output clock)
	Minimum data delay (output) *(3)*, *(4)*	-0.021	0.918	Minimum DQ t_{CO} from the Quartus II software (–90° PLL output clock)
	Maximum data delay (output) *(3)*, *(4)*	0.117	1.226	Maximum DQ t_{CO} from the Quartus II software (–90° PLL output clock)
Board specifications	t_{EXT}	0.020	0.020	Board trace variations on the DQ and DQS lines
Timing calculations	t_{EARLY_CLOCK}	0.875	1.814	Earliest possible clock edge seen by the memory device (minimum clock delay – t_{PLL_JITTER} – $t_{CLOCK\_SKEW\_ADDER}$)
	t_{LATE_CLOCK}	1.010	1.959	Latest possible clock edge seen by the memory device (maximum clock delay + t_{PLL_JITTER} + $t_{CLOCK\_SKEW\_ADDER}$)
	$t_{EARLY\_DATA\_INVALID}$	1.452	2.391	Time for earliest data to become invalid for sampling at the memory input pins (t_{HP} + minimum data delay – t_{PLL_PSERR})
	$t_{LATE\_DATA\_VALID}$	0.132	1.241	Time for latest data to become valid for sampling at the memory input pins (maximum data delay + t_{PLL_PSERR})

Table 22. Write Timing Analysis Example for 267-MHz DDR2 SDRAM Interface in EP2S60F1020C3 (Part 2 of 2) *(Note 1)*

Parameter	Specification	Fast Corner Model	Slow Corner Model	Description
Results	Write setup timing margin *(3)*	0.328	0.158	$t_{EARLY\_CLOCK} - t_{LATE\_DATA\_VALID} - t_{DS} - t_{EXT}$
	Write hold timing margin *(3)*	0.087	0.077	$t_{EARLY\_DATA\_INVALID} - t_{LATE\_CLOCK} - t_{DH} - t_{EXT}$
	Total margin	0.415	0.235	Setup margin + hold margin

Notes to Table 22:

(1) This analysis is performed with FPGA timing parameters for an EP2S60F1020C3 device. You must use this template to analyze timing for your preferred Stratix II density-package combination. Refer to the "PLL Timing Specifications" and "Clock Network Skew Adders" sections of the *DC Switching Characteristics* chapter volume 1 of the *Stratix II Device Handbook*, for FPGA specifications.

(2) The memory numbers used here come from the Micron MT47H64M8-37E component data sheet using DQ and DQS edge rates of 1 V/ns. To determine the edge rate for your design, you must perform a board level simulation as described in *Application Note 408: Stratix II FPGA DDR2 Memory Interface Termination, Drive Strength and Loading Design Guidelines.*

(3) These numbers are from the Quartus II software, version 5.1 using the DDR2 SDRAM Controller MegaCore 3.3.0.

(4) Package trace skews are modeled by the Quartus II software.

Round-Trip Delay Calculation

Resynchronization involves transferring memory read data from the DQS clock domain to the FPGA system clock domain. Analyzing timing for the resynchronization path requires round-trip delay (RTD) calculation for clock and data signals across PVT. When calculating RTD using the fast and slow timing models, a safe resynchronization window might not exist at higher speeds when using the single-stage resynchronization implementation. Timing analysis for typical Stratix II DDR2 interfaces has shown that single-stage resynchronization performance (without a fedback clock) is limited to approximately 200 MHz. A feedback clock implementation increases this performance limit using delay matching and multistage resynchronization.

A typical RTD path includes delay components such as t_{CO} of the FPGA CK output pin, board trace delay for CK clock, memory propagation delay from CK to DQS, DQS phase-shift delay inside the FPGA, and DQ propagation delay from capture register to resynchronization register. Each of these delay components can vary significantly across PVT and result in a data valid window that is severely diminished or non-existent. A feedback clock architecture uses two register stages between the memory clock domain and the system clock to split uncertainties. Furthermore, the clock and data delay paths to the first stage register have matched delays through the FPGA clock output pin, board trace, and FPGA input pin to register. The uncertainties from the first resynchronization register stage are moved to the second stage register. In this mode, the feedback clock output uses the same I/O standard as the FPGA CK clock output. And a PLL-compensated clock network eliminates delay variation across PVT. Therefore, this 2-PLL fedback clock implementation is recommended for higher speeds. The DDR2 SDRAM MegaCore IP Toolbench provides timing margin analysis for resynchronization paths for both implementations. Review the timing analysis output from the Quartus II software or perform a similar paper analysis to select the best resynchronization implementation for your design.

Round-Trip Delay with the Optional Feedback Clock

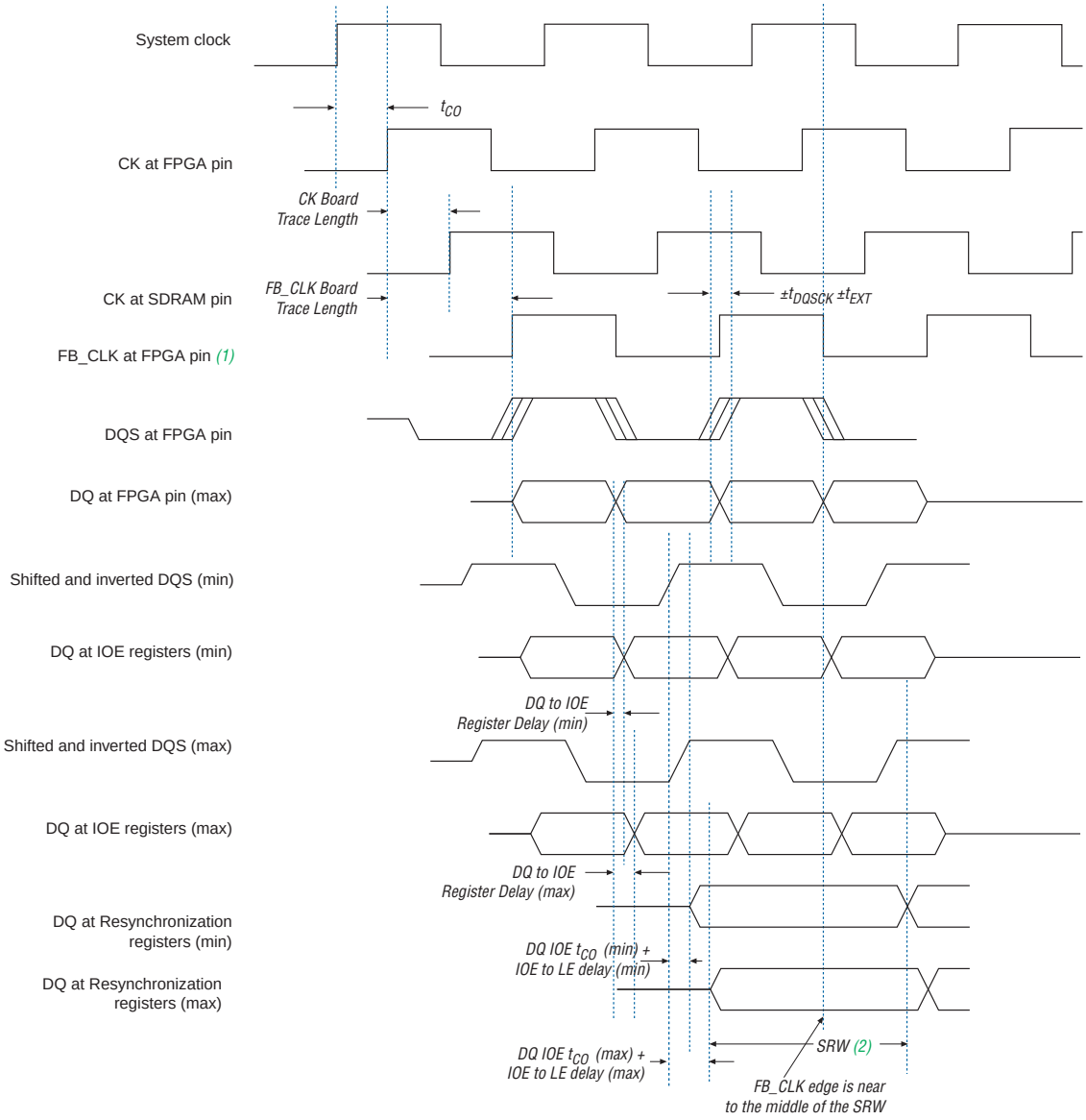
The feedback clock and the read PLL shown in Figure 67 on page 100 improves resynchronization. This feedback clock follows the FB_CLK signal to memory and routed back to the Stratix II FPGA to feed a second PLL, called the read PLL. This read PLL must be in normal mode so that the output is in phase with the input to the PLL (if there is no phase-shifting). The input to the PLL is then skewed by $\pm t_{DQSQ}$ from the DDR2 SDRAM plus any board trace skew between DQS, CK, and the FB_CLK traces. The PLL can then compensate for the delay between the clock input pin to the LE register and synchronize the data from the DQS clock domain to the feedback clock domain.

The feedback clock lags the system clock by the board trace lengths for the CK and DQS signals ($l_1 + l_2$) delay. You can calculate whether outputs of the registers clocked by the feedback clock need another resynchronization stage before getting to the system clock domain. In this calculation, you need to have the maximum and minimum values for the following delays:

- Clock-to-out delays for the CK signal from the Stratix II device
- CK board trace lengths
- DQS board trace lengths
- Register-to-register delays between the registers in the feedback clock domain and the registers in the system clock domain

Figure 76 shows an example of a timing waveform when using the optional feedback clock for resynchronization.

Figure 76. Round-Trip Delay Example



Notes to Figure 76:

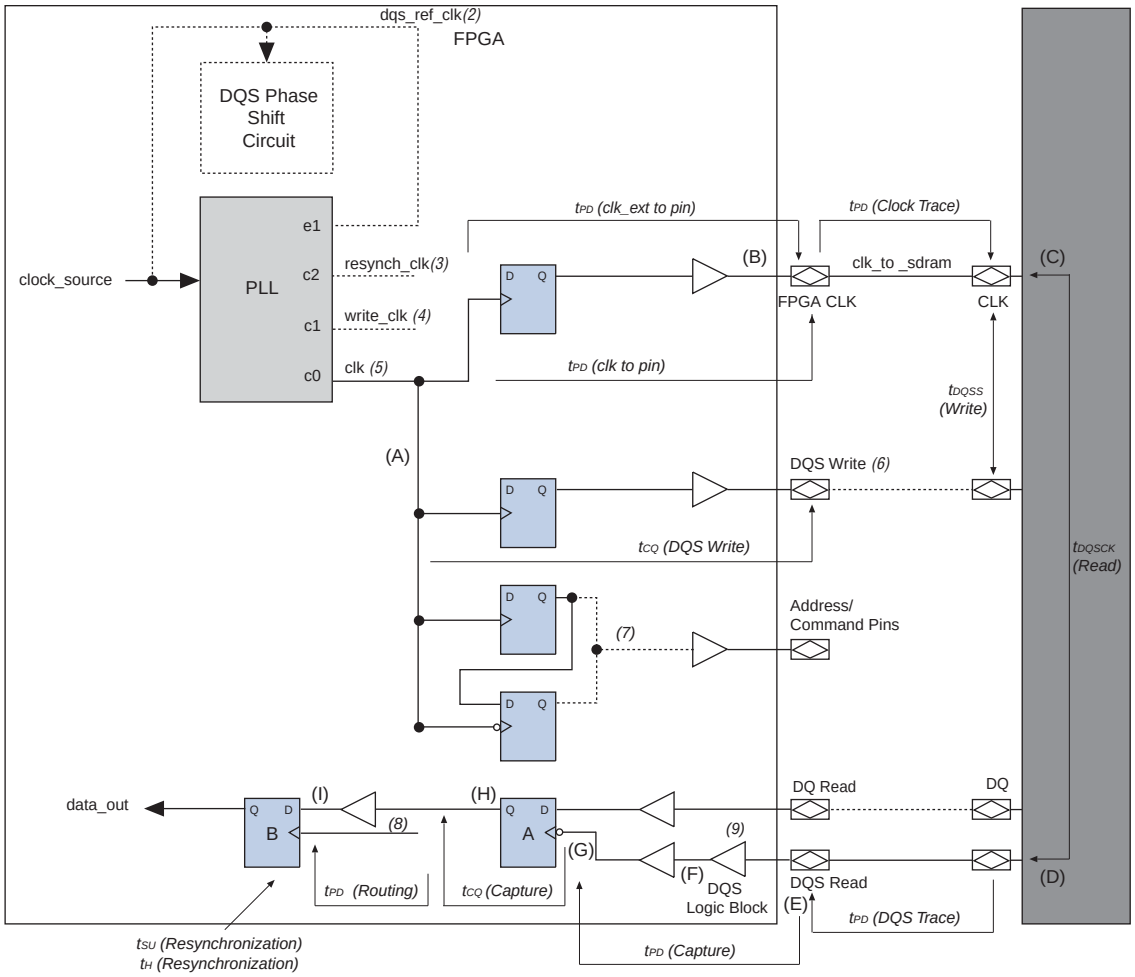
(1) The FB_CLK input signal goes to a PLL in which the output can be shifted in such a way that it is centered in the safe resynchronization window (SRW).

(2) SRW.

Round Trip Delay Calculation Without the Optional Feedback Clock

Figure 77 shows the timing analysis and the round trip delay when the feedback clock is not used. The round trip delay is the delay from the FPGA clock to the DDR2 SDRAM and back to the FPGA (input to register B). This analysis is required to reliably transfer data from register A (in the IOE) to register B (in the LE).

Figure 77. Round Trip Delay Without a Feedback Clock



Notes to Figure 77:

(1) The nodes for the round trip delay (RTD) analysis are marked with letters (A) through (I).

(2) The `dqs_ref_clk` input for Stratix II devices can be either from an output of the PLL or directly from an input clock pin.

(3) The `resynch_clk` is optional based on the round trip delay of the system.

(4) The `write_clk` signal is shown for completeness, but it is not needed in the timing analysis for round-trip delay or address/command timing.

(5) `clk` is the system clock.

(6) The DQS signal is bidirectional. DQS write and DQS read are shown as two separate pins for this timing analysis.

(7) You can clock the address/command register with either a rising edge or falling edge of the `clk` signal.

(8) Register B's clock input can be `clk`, `write_clk`, or `resynch_clk`. The `clk` and `write_clk` signals can also be inverted at Register B if needed.

(9) The DQS phase-shift reference circuit controls the 90° phase shift dynamically. The control path is not shown and its operation is user transparent.

Register A in Figure 77 represents the DDR2 SDRAM capture logic. The Q output from register A represents the point at which the read data has been converted from DDR2 SDRAM to single data rate (SDR). At the output of register A, the data is already at single data rate, but is still in the DQS clock domain. DQ_H (DQ data during DQS high) is sampled on the positive edge of the 90° phase-shifted DQS pulse, but re-sampled on the negative edge of the 90° phase-shifted DQS pulse, to align it with DQ_L (DQ data during DQS low).

After sampled by the negative edge of the 90° phase-shifted DQS pulse, DQ_L and DQ_H are available for resynchronization.

To sample the Q output of register A into register B, you need the time relationship between register B's clock input and the D input, which depends on the phase relationship between the DQS and `clk` signals and involves the following steps:

1. Calculate the systems' round trip delay.
2. Select a resynchronization phase of the system clock or other available clock that reliably samples the Q output of register A, based on the calculated safe resynchronization window (SRW).
3. Apply the correct clock edge for your resynchronization logic in your memory controller.

You can use the `clk`, `write_clk`, or `resynch_clk` signals as the register B clock input. You can also invert `clk` and `write_clk` if needed. To determine the timing of data at the D input of register B relative to `clk`, consider the following timing-path dependencies:

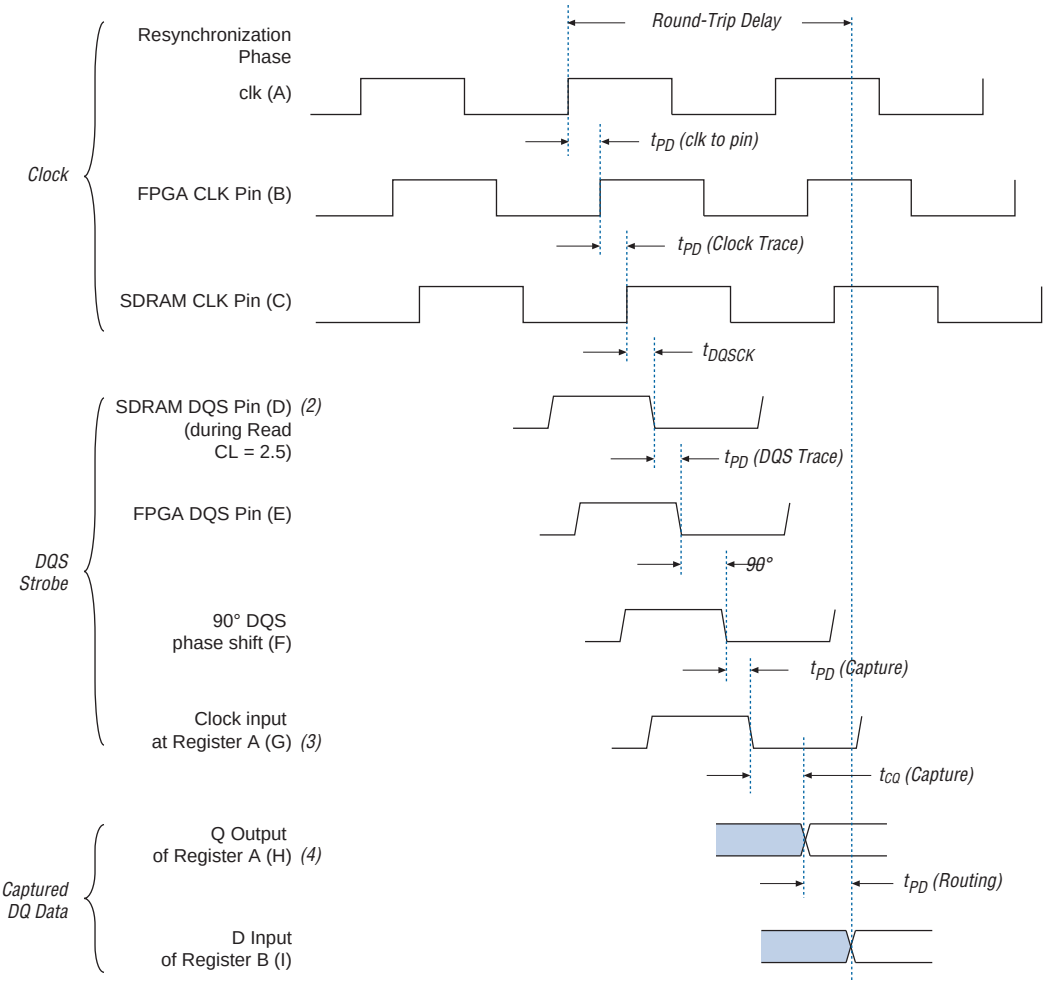
- The DDR2 SDRAM clock input arrives (a delayed version of `clk`).
- DQS strobe from the DDR2 SDRAM arrives at the clock input of register A.
- Data arrives at the Q output of register A.
- Data arrives at the D input of register B.

There are three main parts to this path:

- Clock delays—between the FPGA global clock net and the DDR2 SDRAM clock input.
- DQS strobe delays—between the DDR2 SDRAM clock input and dqs's arrival at the FPGA capture registers.
- Read data delays—between the output of register A and the input of register B.

Figure 78 shows the timing relationship of the signals for the delays between points (A) to (I) for a CAS latency of 2.5 clock cycles.

Figure 78. RTD Calculation without a Feedback Clock



Notes to Figure 78:

(1) The letters in parenthesis refer to the letters in Figure 77.

(2) The DQS strobe edge can be anywhere within $\pm t_{DQSCK}$ of the DDR2 SDRAM clock pin edge. For calculating the maximum round trip delay the diagram assumes the strobe occurs t_{DQSCK} after the clock. For calculating the minimum it has to be assumed it occurs t_{DQSCK} before.

(3) The delays in the DQS path from FPGA pin to capture register are matched to those for the DQ path with the exception of the DQS delay chain.

(4) Although data is initially sampled at a capture register on the positive edge of DQS, it is only on the negative edge that both DQ_H and DQ_L are available at the Q outputs of the DDR2 SDRAM capture logic. At this point they are then single data rate.

To determine the point at which the data can be reliably resynchronized, calculate the minimum and maximum round trip delay. You can then determine what resynchronization logic to use for your system. Remember to take into account PVT variations.

Delay (A) to (B) is the clock-to-out time to generate the clock signals to the DDR2 SDRAM component.

Delay (B) to (C) is the trace delay for the clock. If there are multiple DIMMs or devices in the system, the one furthest away from the FPGA must be used for the maximum calculation; the closest for the minimum.

Delay (C) to (D) is the relationship between the clock and the DQS strobe timing during reads. This is t_{DQSCK} in DDR2 SDRAM specifications, nominally 0, but typically varies by ±0.75 ns depending on the DDR2 SDRAM component-speed grade. The DQS output strobe is only guaranteed to be within $\pm t_{DQSCK}$ of the clock input. So use t_{DQSCK} (maximum), typically +0.75 ns, for calculating the maximum round trip delay; t_{DQSCK} (minimum), typically –0.75 ns, for calculating the minimum delay.

Delay (D) to (E) is the trace delay for DQS, which typically matches the trace delay for the DQ signals in the same byte group. To calculate the maximum RTD, use the byte group with the longest trace lengths; for the minimum use the shortest. Similarly, if there are multiple DIMMs or devices in the system, use the one furthest from the FPGA for the maximum calculation and the one closest to the FPGA for the minimum. Trace lengths between different byte groups do not have to be tightly matched, but a difference between the longest and shortest decreases the safe resynchronization window. This increase reduces the window size in which the data can be reliably resynchronized.

PLL jitter and clock duty cycle also affect the RTD. Add each of these delays to the maximum value and subtract from the minimum. PLL jitter and clock duty cycle are not shown in Figure 77 on page 124, but are included in Table 23 on page 130 which shows example RTD calculations.

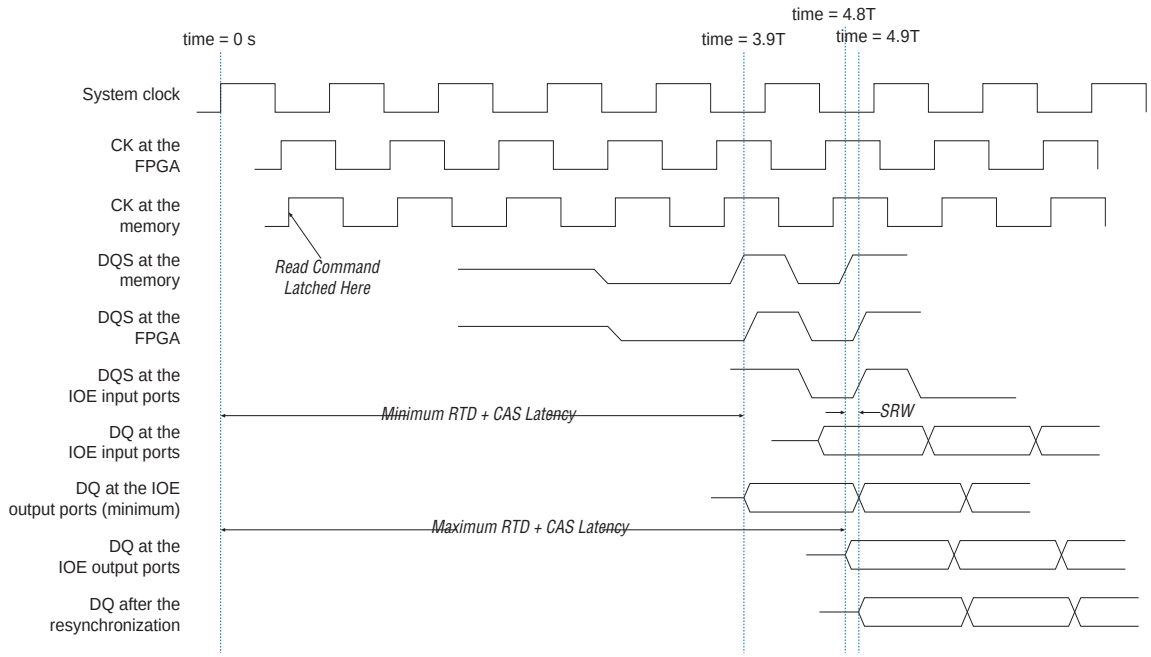
Resynchronization Selections

When the DQS signal arrives at the Stratix II device, the dedicated phase-shift circuitry shifts the signal by 90° to capture the DQ signals. The DQ signals are then ready to be synchronized with the system clock. The round trip delay numbers vary depending on the board delay and the device internal delay. Complete a timing analysis to decide whether to use the falling edge or the rising edge of the system clock of the write clock for the synchronization registers. After calculating the maximum and minimum RTD, determine the equivalent number of system clock cycles at your operating frequency to find the point at which the data becomes valid relative to `clk`. The example maximum delay in Table 23 on page 130 represents 1.8 cycles at 200 MHz; the minimum represents 0.9 cycles. If the CAS latency is included, which is equal to three in Figure 73 on page 110, the example represents a minimum delay of 3.9 cycles and a maximum delay of 4.8 cycles.

The overlap of the minimum and maximum data valid windows defines the data valid window, which comprises the safe resynchronization window and t_{SU} and t_H of register B.

Figure 79 shows an example timing waveform of the RTD analysis in Table 23 on page 130.

Figure 79. RTD Diagram without a Feedback Clock Example 1 *(Note 1)*



Note to Figure 79:

(1) T refers to the clock period of the system, which is 5 ns in this example.

The RTD helps you determine the safe resynchronization window (SRW) and how you need to resynchronize the data. Because the shifted DQS signal can go into the LEs, you can use the shifted DQS signal for resynchronization in the LE. This may mean that the DQS clock-domain to system clock-domain transfer happens between the two LE registers. The timing analysis in this section assumes that the clock domain transfer happens from the IOE register to the LE register.

Assume that time 0 is the time of the clock rising edge. After the DDR2 SDRAM component receives this rising edge from the Stratix II device, the read command is clocked into the SDRAM upon receiving this positive edge. You can calculate the SRW valid time as follows:

Equation 1.

$$\text{Minimum SRW valid time} = \text{maximum RTD} + \text{CAS latency} \times \text{clock period} + \mu t_{SU}$$
$$\text{Maximum SRW valid time} = \text{minimum RTD} + (\text{CAS latency} + 1) \times \text{clock period} - \mu t_{H}$$

From the example in Table 23 on page 130, the minimum SRW valid time is 4.8 cycles and the maximum SRW valid time is 4.9 cycles (ignoring μt_{SU} and μt_{H}).

The size of the SRW in the example is then 0.1 cycles, as calculated in Equation 2:

Equation 2.

$$\text{SRW size} = \text{maximum SRW valid time} - \text{minimum SRW valid time}$$

The SRW size must be large enough to accommodate the worst-case clock skew between two PLL output clocks (150 ps).

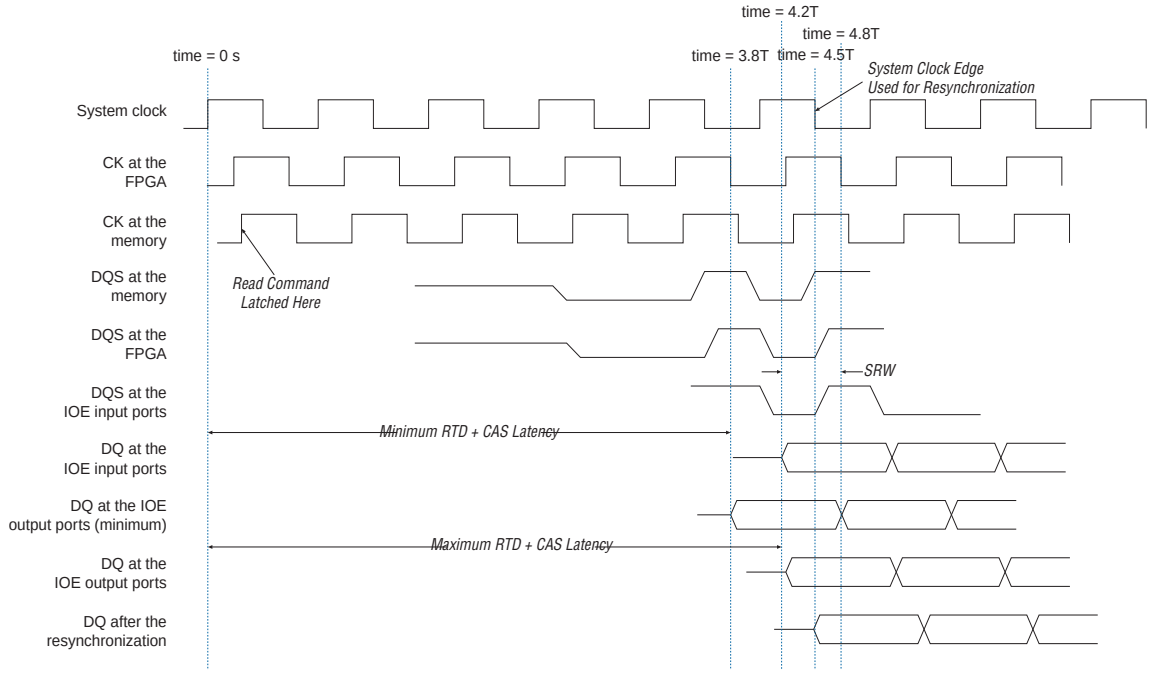
Next, determine how many half clock cycles elapse from time 0 to the minimum SRW valid time (`numcycle`) by calculating the ceiling function of the minimum SRW valid time divided by half a clock cycle. To find out whether the SRW falls in a clock edge, multiply `numcycle` by half a clock cycle. If the result is less than the maximum SRW valid time, then a system clock edge falls in the SRW. Otherwise, you need an extra PLL output for your resynchronization clock.

The example in Table 23 on page 130 shows that `numcycle` is equal to 10 and that the SRW does not fall in a system clock edge.

If you do not need a resynchronization clock and `numcycle` is an even number, the active system clock edge for resynchronization is the positive edge. If `numcycle` is odd, the resynchronization system clock edge is the negative edge, and you must determine the resynchronization phase selection.

Figure 80 shows an example in which the SRW is in a system clock edge. In this example, `numcycle` is equal to 9 (time = 4.5T) and the negative edge of the system clock is used for the resynchronization clock.

Figure 80. RTD Diagram without a Feedback Clock Example 2



Note to Figure 80:

(1) T refers to the clock period of the system, which is 5 ns in this example.

If there is no clock edge available in the safe resynchronization window, you need an extra resynchronization clock, you can shift the system clock from either edge. If `numcycle` is even in this case, the closest system clock edge to the SRW is negative and if `numcycle` is odd, the closest clock edge is positive.

You can calculate the needed phase shift for the resynchronization clock by using the Equation 3 and Equation 4.

Equation 3.

$$\text{Minimum phase shift} = \text{minimum SRW valid time} + \text{PLL clock skew (150 ps)} - (\text{numcycle} - 1) \times t_{CK}/2$$

Equation 4.

$$\text{Maximum phase shift} = \text{minimum SRW valid time} - \text{PLL clock skew (150 ps)} - (\text{numcycle} - 1) \times t_{CK}/2$$

The phase-shift calculation example in Table 23 lists that the minimum phase shift is 1.61 ns and the maximum phase shift is 1.52 ps. This is because the SRW is less than 300 ps. You can still choose median (1.565 ns) for the phase shift of the resynchronization clock.

You then need to convert the results to the equivalent degree phase shifts. If the closest clock edge to the SRW is negative, add or subtract 180° after the conversion to shift the clock from the positive edge. For example, in Table 23, the phase-shift range is between 1.52 to 1.61 ns based on the negative edge clock. The median of this number is 1.565 ns, which equates to ~113° (from the 200-MHz clock). If you want this clock to be shifted from the positive edge of system clock, you can either use 293° (113° + 180°) or –67° (113° – 180°).

Table 23. Example RTD Calculation *(Note 1)*

Delay	Numbers in Figure 77 and Figure 78	Example Minimum Values (ns)	Example Maximum Values (ns)	Comments
t_{PD} (clk to pin)	(A) to (B)	2.00	3.00	Equal to t_{CQ} (DQS write)
t_{PD} (clock trace)	(B) to (C)	0.33	0.50	2 to 3 inches at 166 ps per inch *(2)*
t_{DQSCK}	(C) to (D)	–0.60	+ 0.60	Refer to DDR2 SDRAM specifications
t_{PD} (DQS trace)	(D) to (E)	0.33	0.50	2 to 3 inches at 166 ps per inch *(2)*
90° phase-shift	(E) to (F)	1.133	1.333	Includes Stratix II DLL jitter and phase-shift error
t_{PD} (capture)	(F) to (G)	0.50	1.00	
t_{CQ} (capture)	(G) to (H)	0	0.16	
t_{PD} (routing)	(H) to (I)	1.00	1.50	
PLL jitter	-	–0.10	+ 0.10	PLL jitter specification
Clock duty cycle	-	–0.25	+ 0.25	45-55% duty at 200 MHz
Round-trip total	(A) to (I)	4.343	8.943	

Notes to Table 23:

(1) Numbers quoted here are not taken from a specific system or a specific device.

(2) To know the exact delay for your system, perform a time domain reflectrometry (TDR) analysis on your system.

DQS Postamble

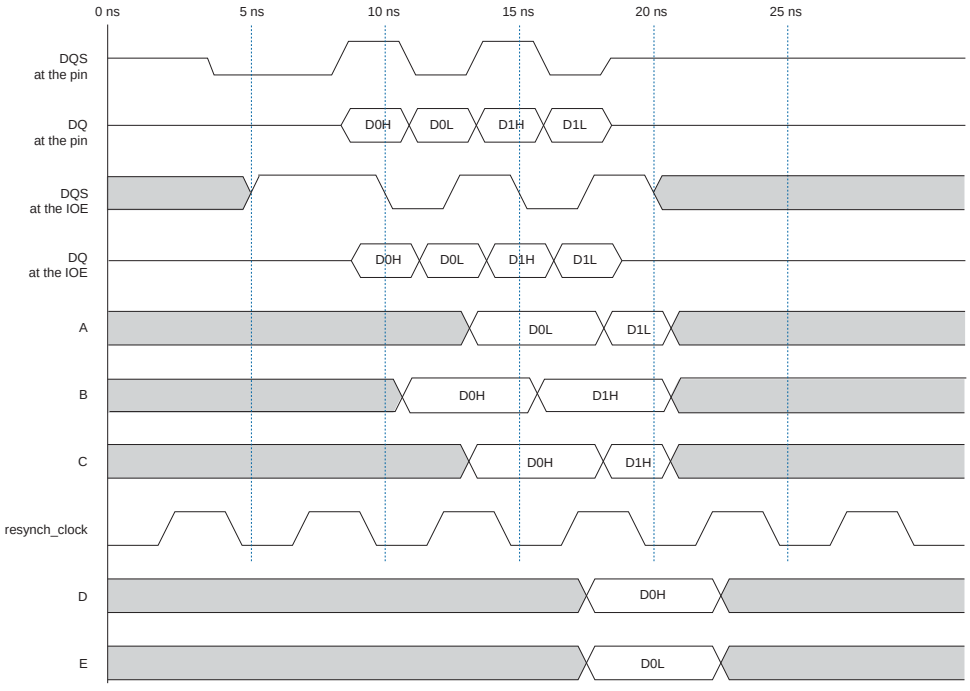
The DQS postamble feature only applies to the DLL-based read implementation. The DQS strobe is not used in the PLL-based implementation, so you do not have to consider DQS postamble.

The DDR2 SDRAM DQ and DQS pins use the SSTL-18 I/O standard. When neither the Stratix II nor DDR2 SDRAM component drive the DQ and DQS pins, the signals go to a high-impedance state. A pull-up resistor terminates both DQ and DQS to V_{TT} (0.9 V), so the effective voltage on the high-impedance line is 0.9 V. According to the JEDEC JESD 8-15A specification for the SSTL-18 I/O standard, this is an indeterminate logic level and the input buffer can interpret this as either a logic high or logic low. If there is any noise on the DQS line, the input buffer may interpret that noise as actual strobe edges. Therefore, when the DQS signal goes to tri-state after a read postamble, you must disable the clock to the input registers so erroneous data does not get latched in and all the data from the memory is resynchronized properly.

Figure 81 shows a read operation example in which the DQS postamble could be a problem. This figure shows definitions of waveforms A, B, C, D, and E.

- Waveform A shows the output of the active high IOE input register.
- Waveform B shows the active low register output of the Stratix II IOE input register.
- The active low register output goes into the latch, whose output is shown in waveform C.
- Waveforms D and E show the output signals after the resynchronization registers.

Figure 81. Read Example with a DQS Postamble Issue



The first falling edge of the DQS at the IOE register occurs at 10 ns. At this point, data D0H is clocked in by the active low register (waveform B). At 12.5 ns, data D0L is sampled in by the active high register (waveform A) and data D0H passes through the latch (waveform C). In this example, the positive edge of the `resynch_clock` occurs at 16.5 ns, in which both D0H and D0L are sampled by the logic element's (LE's) resynchronization registers. Similarly, data D1H is clocked in by the active low register at 15 ns, while data D1L is clocked in by the active high register and data D1H passes through the latch at 17.5 ns.

At 20 ns, assume that noise on the DQS line causes a valid clock edge at the IOE registers, such that it changes the value of waveforms A, B, and C. The next rising edge of the `resynch_clock` signal does not occur until 21.5 ns, but data D1L and D1H are not valid anymore at the output of the latch and the active-high input register, so the resynchronization registers do not sample D1L and D1H and may sample the wrong data instead.

Stratix II devices have circuitry to prevent a false edge trigger at the end of the DQS postamble. Each Stratix II DQS logic block is connected to a postamble circuitry that consists of AND, NAND, and NOT gates (Figure 82). When you enable the `gated_dqs` control (in the Quartus II software), you can AND the DQS signal with the output of the input register located inside the DQS IOE. The shifted DQS clock must clock the input register. Connect the register's SCLR input to V_{CC}. The controller must include extra logic to tell the reset signal to release the preset signal on the falling DQS edge at the start of the postamble. This disables any glitches that happen following the postamble.

Figure 82. Stratix II DQS Postamble Circuitry Connection

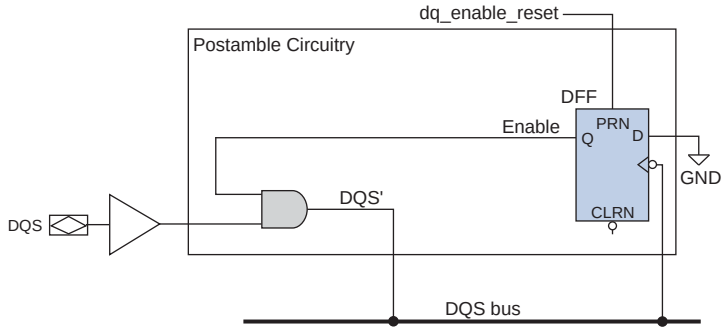


Figure 83 shows the timing waveform for Figure 82.

Figure 83. Stratix II DQS Postamble Circuitry Control Timing Waveform

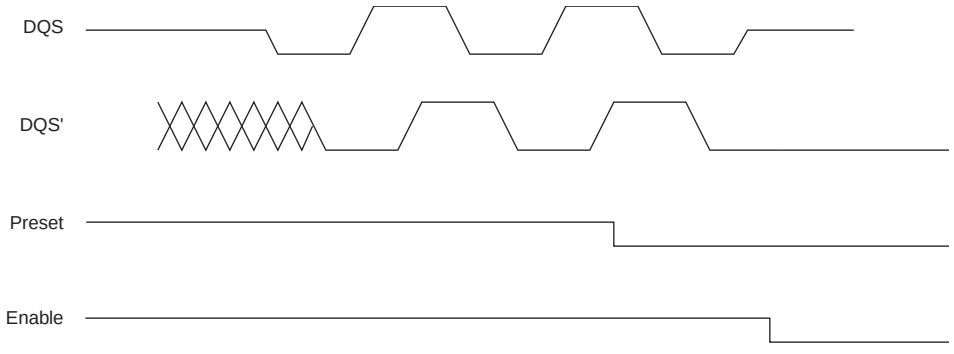
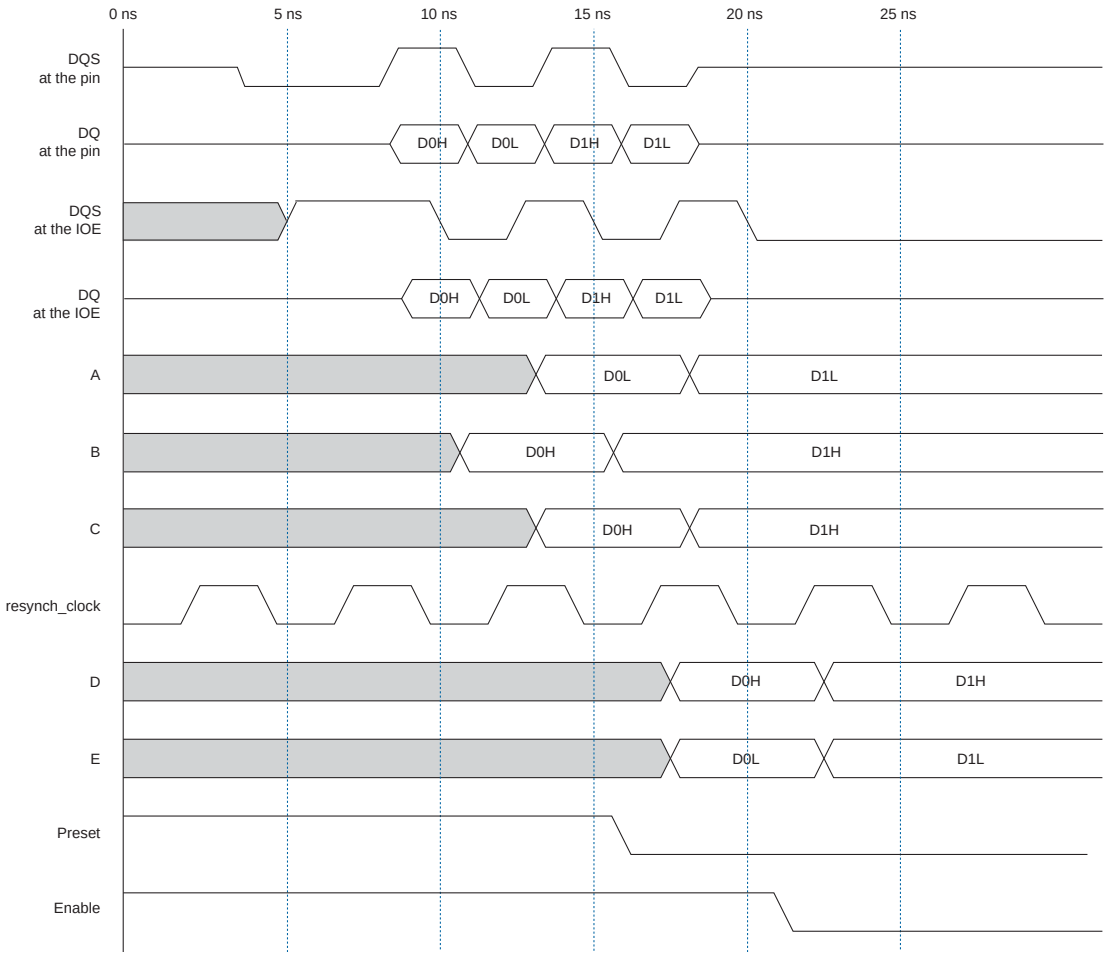


Figure 84 shows the read timing waveform when you use the Stratix II DQS postamble circuitry.

Figure 84. Stratix II DQS Postamble Circuitry Read Timing Waveform



Appendix E: The relative_constraint.tcl Script

The **relative_constraint.tcl** script is used for making location constraints to registers in a DDR interface.

In some situations, the Quartus II software does not automatically place registers in the LAB adjacent to the I/O pin. This script creates LAB location constraints using offsets that you specify relative to the pin locations.

Table 24 lists the arguments that are available with the script.

Table 24. The relative_constraint.tcl Arguments

Arguments	Value	Default value	Required
`-project value`	Name of project	<>	Yes
`-revision value`	Name of revision	<>	No
`-pin_name value`	Name of items that are fixed	<>	Yes
`-reg_name value`	Name of floating items to position	<>	Yes
`-row_offset value`	Row offset relative to anchor location as an integer	0	Yes
`-column_offset value`	Column offset relative to anchor location as an integer	0	Yes
`-apply` *(1)*	Apply the actual constraints	N/A	Yes
`-show_regs`	Print out the matching register names	N/A	No
`-show_pins`	Print out the matching pin names	N/A	No
`-pin_range value`	Pin bus slice to process, for example 71:0	<>	No
`-reg_range value`	Register bus slice to process, for example 71:0	<>	No
`-bidir` *(2)*	Check bi-directional pins	N/A	No
`-input` *(2)*	Check input pins	N/A	No
`-output` *(2)*	Check output pins	N/A	No
`-help`	Print this message	N/A	No
`-?`	Print this message	N/A	No

Notes to Table 24:

(1) This is a required argument when you actually make the location assignment.

(2) If you do not specify `-bidir`, `-input`, or `-output`, the default is `–bidir`.

Usage

The **relative_constraint.tcl** script only works for pins and registers that are parts of a bus. It also requires that the number of pins match the numbers of registers. If not, you can bound it with the **–pin_range** or **–reg_range** argument.

The script issues the following warnings if the numbers of pins and registers do not match:

```
Warning: Unequal number of pins and registers. Ensure your
patterns are correct.

Warning: Use the -show_regs and/or -show_pins options to show
matching registers and pins.
```

To use the script follow these steps:

1. Copy the script to your project directory.
2. Run the script with the following options to locate the specific pin and registers in your design:

   ```
   quartus_sh -t relative_constraint.tcl -project <project name> -
   pin_name <wildcard to match IO pins> -reg_name <wildcard to match
   register names> <-bidir/-output/-input>
   ```

   This finds all the registers and pins matching those names.

   Figure 85 shows an example result of finding the address pins and registers in a DDR2 SDRAM interface. The current pin locations, the would-be LAB locations for the registers, as well as the pin and register match-up are displayed in the result. Notice that the Y-coordinates of the LAB locations are Y0. This is because the default **–row_offset** and **–column_offset** values are 0 and because the address registers are located at the bottom of the device, the Y-coordinate is 0.

Figure 85. Finding Address Pins and Registers in a DDR2 SDRAM Interface

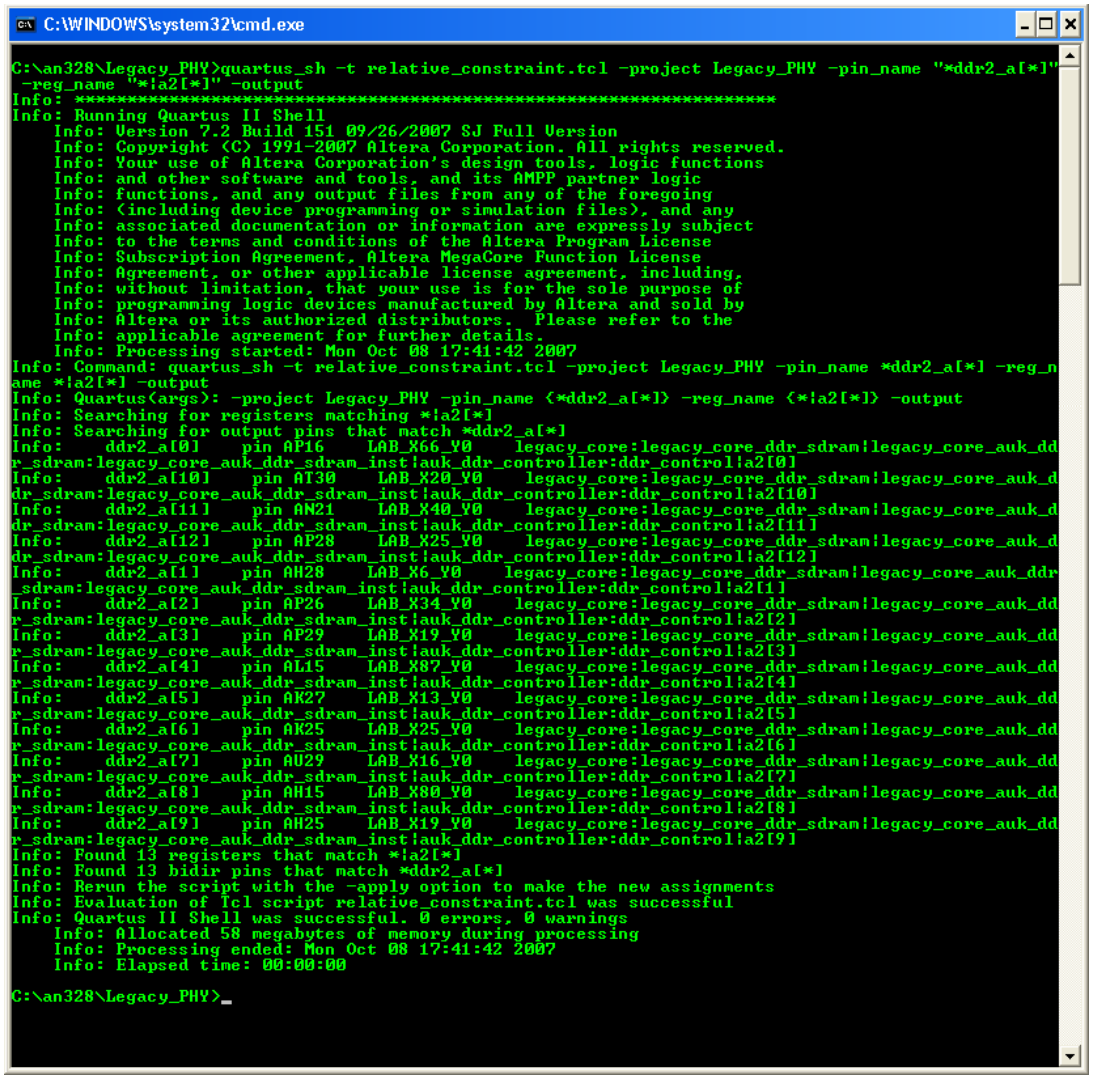
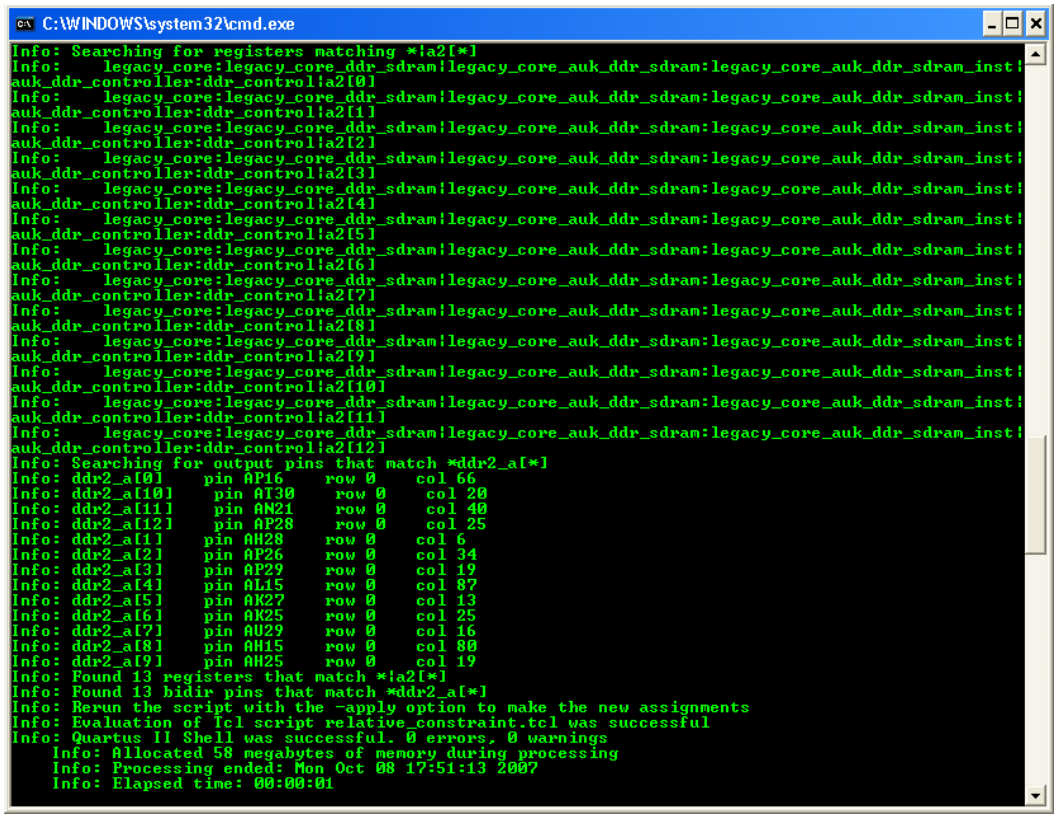


Figure 86 shows the result of the script when you use the **–show_regs** and the **–show_pins** arguments.

Figure 86. Output of the -show_regs and -show_pins Options on relative_constraint.tcl



3. You can bypass step 2 if you already know where you want to place the registers.

   ```
   quartus_sh -t relative_constraint.tcl -project <project name> -
   pin_name <wildcard to match IO pins> -reg_name <wildcard to match
   register names> <-bidir/-output/-input> -row_offset <+/- number of
   rows relative to the pin to make the LAB assignment> -column_offset
   <number of columns relative to the pin to make the LAB assignment>
   ```

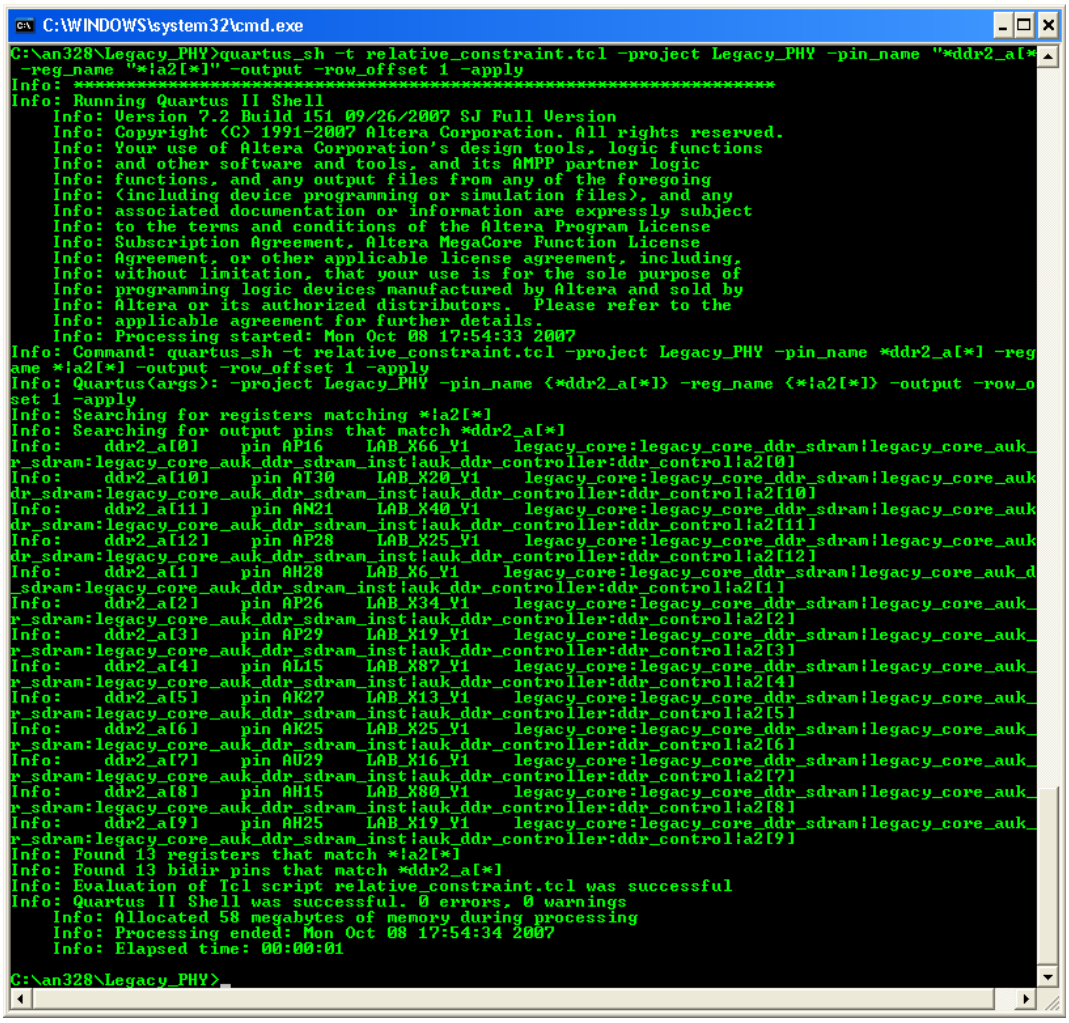
   The script prints out information about the assignments it makes so you can verify the assignments.

4. To apply the assignment, repeat the command in step 3 with the **–apply** argument appended. In the address pins and registers example, the command:

   ```
   quartus_sh -t relative_constraint.tcl -project Legacy_PHY -pin_name
   "*ddr2_a[*]” -reg_name "*|a2[*]" -output -row_offset 1 -apply
   ```

The previous commands make location assignments to specified registers one LAB above the pin locations at the same column location. The result is shown in Figure 87.

Figure 87. Applying Location Assignment to the Address Registers



The script does not add the location of the assignments to your project until you use the **-apply** option so you do not have to worry about getting everything right the first time.

If you have a bus on the top side of the chip, specify a row offset of -1 to add LAB location constraints for the specified registers one row below the pins. Similarly, if you have a bus on the right side of the chip, specify a column offset of -1 to add LAB location constraints for the specified registers one column below the pins.

You have the option to create a batch file to execute the **relative_constraint.tcl** script multiple times to specify location constraints for different buses of pins and registers in your design.

Referenced Documents

This application note references the following documents:

- *AN 380: Test DDR or DDR2 SDRAM Interfaces on Hardware Using the Example Driver*
- *AN 392: Implementing Multiple Legacy DDR/DDR2 SDRAM Controller Interfaces*
- *AN 408: DDR2 Memory Interface Termination, Drive Strength and Loading Design Guidelines*
- *AN 413: Using Legacy Integrated Static Data Path and Controller Megafunction with HardCopy II Structured ASICs*
- *AN 449: Design Guidelines for Implementing External Memory Interfaces in Stratix II and Stratix II GX Devices*
- *AN 462: Implementing Multiple Memory Interfaces Using the ALTMEMPHY Megafunction*
- *AN 463: Using the ALTMEMPHY Megafunction with HardCopy II Structured ASICs*
- *Avalon Memory-Mapped Interface Specification*
- *DDR and DDR2 SDRAM Controller Compiler User Guide*
- *DDR and DDR2 SDRAM High Performance Controller MegaCore Function User Guide*
- *DDR Timing Wizard (DTW) User Guide*
- *External Memory Interfaces* chapter of the *Stratix II, Stratix II GX,* or *Arria GX Device Handbook*
- *PCI Express Development Kit, Stratix II GX Edition Getting Started User Guide*
- *Quartus II Handbook*
- *Quartus II Online Help*
- *Quartus II Software Release Notes*
- *Stratix II DDR2 System Validation Summary* white paper
- *Stratix II GX PCI Express Development Board Reference Manual*
- *TB 091: External Memory Interface Options for Stratix II Devices*
- www.jedec.org website

Document Revision History

Table 23 shows the revision history for this document.

Table 25. Document Revision History

Date and Document Version	Changes Made	Summary of Changes
October 2009 ver. 6.0	■ Added "Multiple Memory Controllers Walkthrough" on page 86 section ■ Added On-chip termination table: Table 3 on page 5 ■ Updated two termination recommendations tables: Table 8 on page 42 and Table 10 on page 84 ■ Updated graphics with color	—
April 2009 ver. 5.0	■ Updated Table 2. ■ Updated Figures 4-24, Figure 29 -51. ■ Updated sections "Stratix II, Stratix II GX, and Arria GX Dedicated DQS Phase-Shift Circuitry", "Step 1: Select Device", "Step 3: Add Constraints", "Step 6: Adjust Constraints"	—
November 2007 ver. 4.0	■ Updated and added information for Stratix II GX, and Arria GX Devices (througout the document). ■ Added Appendices: "Stratix II GX PCI-Express Development Board Pin Assignments", "Interface Signal Description", "Legacy PHY Architecture Description", "Interface Timing Analysis", and "The relative_constraint.tcl Script". Material regarding interface signals, architecture descriptions, and timing analysis was moved from the body of the document into appendices B and C. Interface timing material was moved into appendix D. ■ Added the sections "Design Example Walkthrough for a 333-MHz DDR2 SDRAM Interface Using the ALTMEMPHY Megafunction", "Design Example Walkthrough for 267-MHz DDR2 SDRAM Interface Using the Legacy PHY", and "Design Checklist". ■ Updated Figure 1, Figure 2, Figure 8, Figure 9, Figure 10, Figure 15, Figure 18, Figure 41, Figure 62. ■ Added Figures 26, 27, 63, 73, 77, 78, and 79. ■ Updated Tables 1, 2, 5 -19. ■ Added Table 4. ■ Updated and added information for Quartus II software, version 7.2 (throughout the document).	Major update to sections, addition of new sections, and material for Arria GX.
May 2006 ver. 3.1	■ Updated sections "DDR2 SDRAM Overview", Interface Description, "Interface Signals", "Clock Signals", "Strobes, Data, DM, and Optional ECC Signals", Command & Address Signals, Data Path Architecture Without Using Dedicated DQS Circuitry, Interface Timing Analysis, "FPGA Timing Information", "Memory Timing Parameters", "FPGA Timing Parameters", "Setup and Hold Margins Calculations", "Read Timing Margins for PLL-Based Implementation", "FPGA Timing Parameters", "Write Data Timing Analysis", and "Round-Trip Delay Calculation". ■ Updated Table 1-Table 10 and Figure 5. ■ Added Figure 4, Figure 10, Figure 14, Figure 15, and Figure 16.	Major update to multiple sections, tables, and figures.

101 Innovation Drive
San Jose, CA 95134
www.altera.com
Technical Support
www.altera.com/support