AN 462: Implementing Multiple Memory Interfaces Using the ALTMEMPHY Megafunction

© April 2009 AN-462-1.3

Introduction

Many systems and applications use external memory interfaces as data storage or buffer mechanisms. As system applications require increasing bandwidth, become more complex, and devices get more expensive, designers prefer to have multiple memory interfaces in a single device to save cost and space on the board, along with removing partitioning complexity. To implement multiple memory interfaces on the same device that are efficient and optimized for the device architecture, designers must pay attention to the device and the physical interface (PHY) features.

This application note 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. Multiple memory interfaces may also involve different types of memory standards, for example, if you have DDR2 SDRAM and RLDRAM II interfaces in a single Stratix® II device. Width and depth expansion of a memory interface are not covered in this document as they are natively supported by the PHY. The memory interfaces described in this document use the ALTMEMPHY megafunction that is available for Arria® GX, Arria II GX, Cyclone® III, HardCopy® II, HardCopy III, HardCopy IV, Stratix II, Stratix II GX, Stratix III, and Stratix IV devices.

☞ The ALTMEMPHY megafunction supports DDR3/DDR2/DDR SDRAM and QDRII+/QDRII SRAM interfaces depending on the target device family. The ALTMEMPHY megafunction creates all the logic needed to calibrate the resynchronization clock dynamically during initialization to remove process (P) variation, and to track voltage and temperature (VT) variation when the system is running. This scheme ensures that the resynchronization data has the optimal setup and hold time margins in any PVT condition of the system. You can also use the ALTMEMPHY megafunction with your own memory controller. Arria GX, Arria II GX, Cyclone III, HardCopy II, HardCopy III, HardCopy IV, Stratix III, and Stratix IV memory interfaces require the use of the ALTMEMPHY megafunction.

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

If you are targeting HardCopy II, Stratix II, or Stratix II GX devices and are unsure whether you can use a memory controller that uses the ALTMEMPHY megafunction as a data path, as in the DDR and DDR2 SDRAM High-Performance Controller, refer to *Technical Brief 091: External Memory Interface Options for Stratix II Devices*.

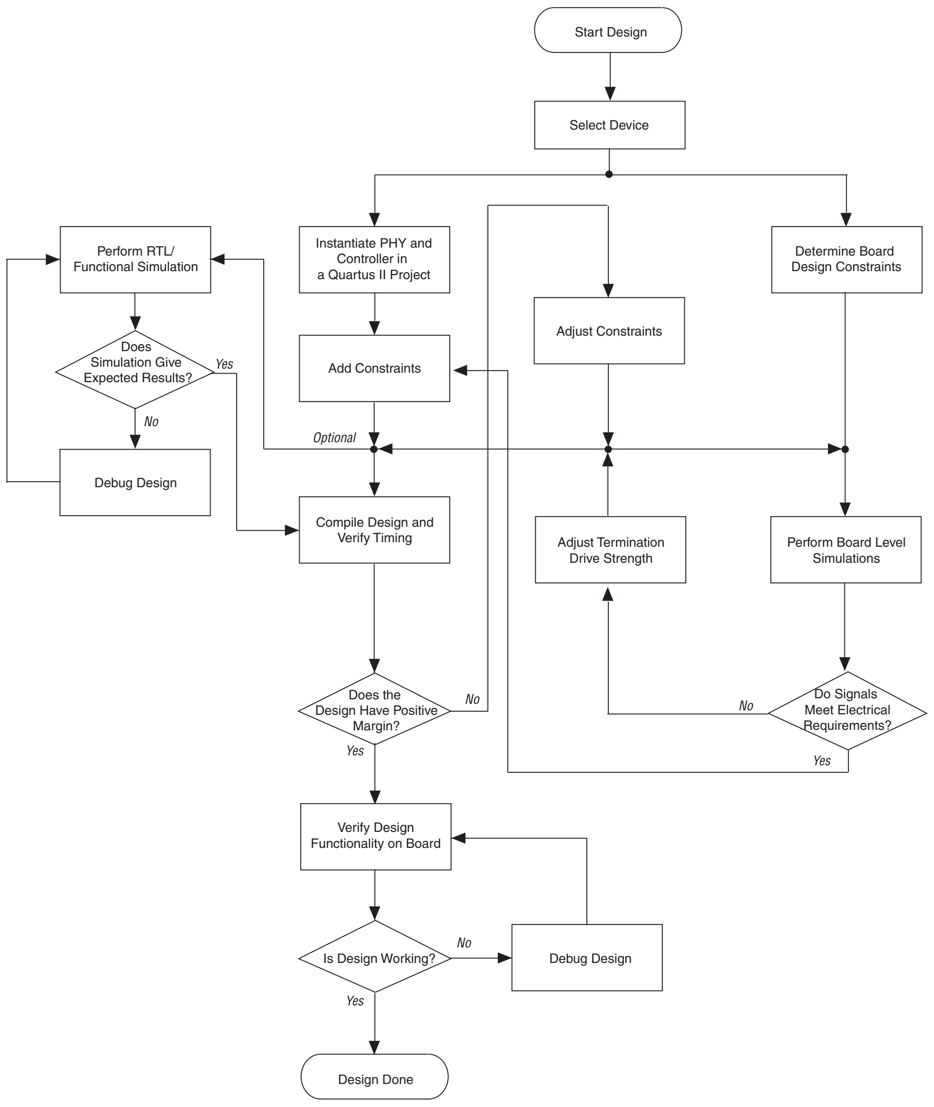
For information on multiple memory interfaces using the legacy integrated static data path and controller, refer to *AN 392: Implementing Multiple DDR/DDR2 SDRAM Controller Interfaces*.

Designs targeting the Stratix II devices are migrated to HardCopy II devices. Any discussion referring to Stratix II devices in this application note also applies Arria GX, HardCopy II, or Stratix II GX devices. However, for additional requirements imposed by HardCopy II devices, refer to *AN 463: Using ALTMEMPHY Megafunction with HardCopy II Structured ASICs*.

If the multiple memory interfaces in your design do not share any resources, you can treat them as independent modules in your design. For more information on creating your own memory controller, refer to the *ALTMEMPHY Megafunction User Guide.* For more information on using the Altera® MegaCore® function as your memory controller, refer to the *DDR and DDR2 SDRAM High-Performance Controller User Guide*.

Figure 1 shows the design flow for implementing external memory interfaces in FPGA devices. This application note covers the steps that are shaded in Figure 1, from creating a Quartus II project to verifying timing for multiple memory interface designs, assuming that the memory devices and FPGA have been chosen. This document prepares you by reviewing the device resources and creating a topology for your design prior to creating a multiple memory interface design in the Quartus II software using the DDR and DDR2 High-Performance Controller MegaCore function as an example.

Figure 1. Design Flow for Implementing External Memory Interfaces in Altera Devices


Start Design
Select Device
Perform RTL/ Functional Simulation
Instantiate PHY and Controller in a Quartus II Project
Determine Board Design Constraints
Adjust Constraints
Does Simulation Give Expected Results?
Yes
No
Add Constraints
Optional
Debug Design
Compile Design and Verify Timing
Adjust Termination Drive Strength
Perform Board Level Simulations
Does the Design Have Positive Margin?
No
Yes
Do Signals Meet Electrical Requirements?
No
Yes
Verify Design Functionality on Board
Is Design Working?
No
Yes
Debug Design
Design Done

Note to Figure 1:

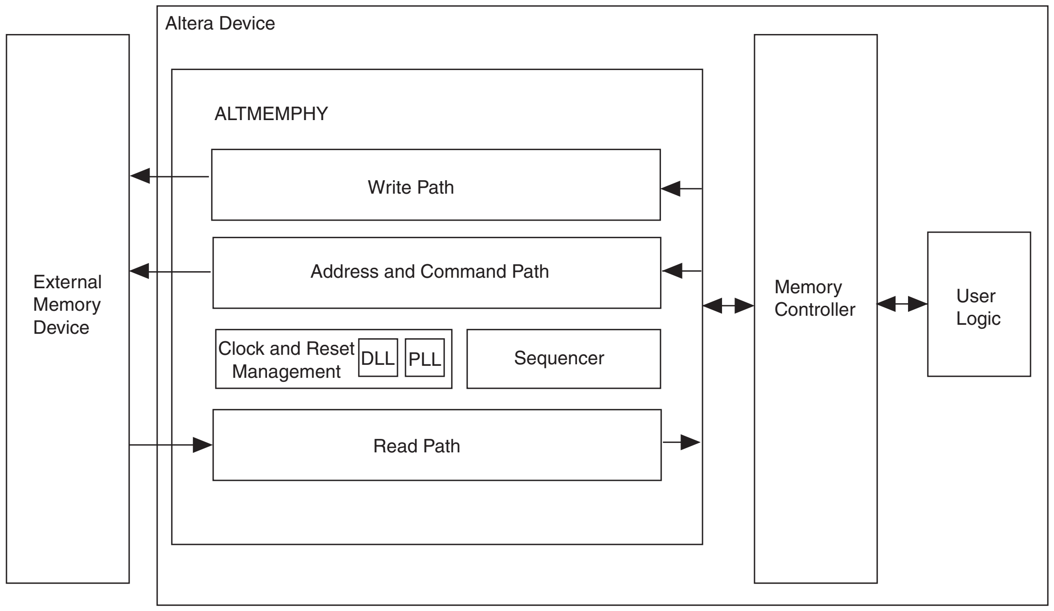
(1) Altera recommends performing this step to ensure design functionality.

This application note assumes that you are familiar with the ALTMEMPHY megafunction and Altera FPGA resources.

Before Creating a Design in the Quartus II Software

Figure 2 shows a single memory interface using the ALTMEMPHY megafunction. The interface consists of user logic connected to a memory controller that connects to the ALTMEMPHY megafunction, which connects with the memory devices via the read data, address/command, and write data path modules. These modules are unique per memory interface and cannot be shared.

Figure 2. Block Diagram of a Memory Controller Using ALTMEMPHY Megafunction Data Path



The sequencer block displayed in Figure 2 is used to calibrate and adjust the resynchronization clock. The ALTMEMPHY megafunction in version 9.0 of the Quartus II software does not allow this block to be shared.

The clock and reset management block generates all the clock and reset signals for the ALTMEMPHY megafunction. This block instantiates the DLL, the PLL, the PLL reconfiguration circuit, and the reset signals which may be shared between memory interfaces.

Note that the controllers have to be running at the same frequency when DLL, PLL, or clock networks are shared.

When creating multiple memory interfaces to be fitted in a single device, first ensure that there are enough device resources for the different interfaces. These device resources include the number of pins, DLLs, PLLs, and clock networks, where applicable. The DLLs, PLLs, and clock network resources from the clock and reset management block are shared between multiple memory interfaces, but there are sharing limitations that you must pay close attention to. These limitations are discussed in "Getting to Know Your Altera Device" on page 5.

Getting to Know Your Altera Device

Knowing the target device means:

- Comparing the requirement for the multiple interface design with the resources available on the device.
- Determining which resources limit multiple interfaces.
- Deciding 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 5
- "I/O Pins" on page 8
- "DLLs" on page 9
- "PLLs and Clock Network Resources" on page 10
- "Other Resources" on page 14

Once 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.

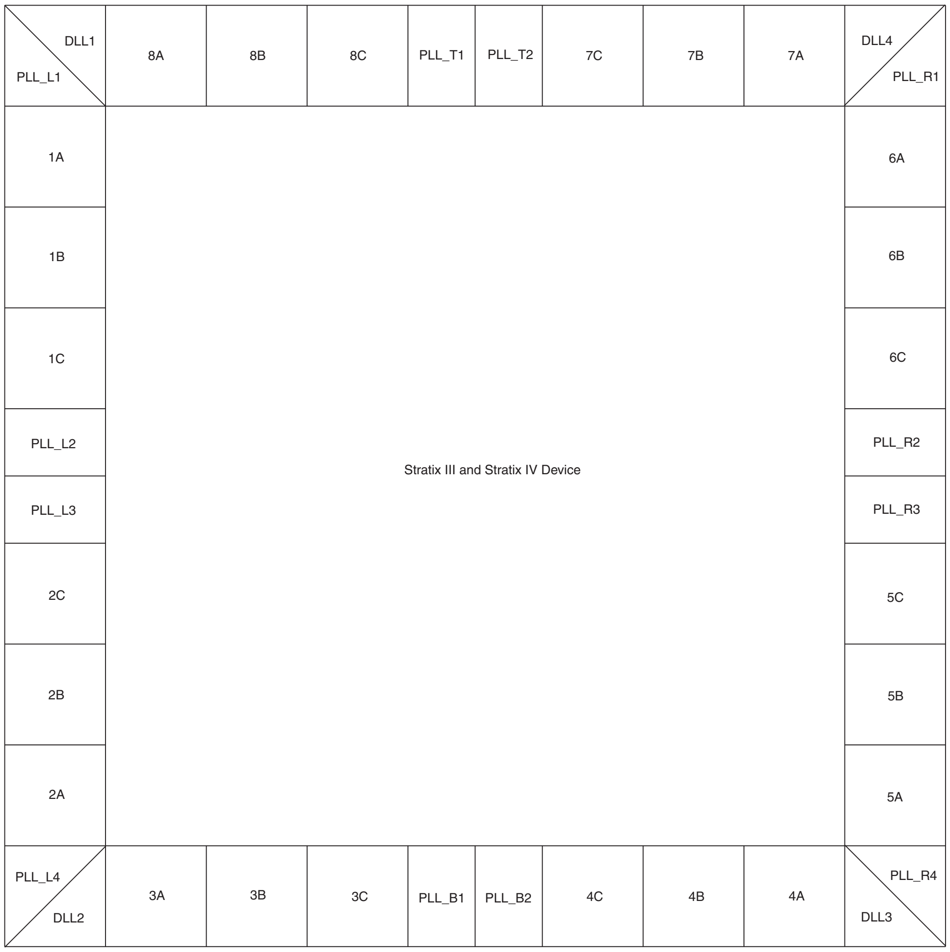
I/O Banks

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.

Stratix III and Stratix IV devices support memory interfaces in all I/O banks. These two device families have up to 24 I/O banks, as shown in Figure 3. You can also have multiple memory interfaces in a Stratix III or Stratix IV I/O banks, provided that you have enough pins and clock network resources. However, Stratix III and Stratix IV devices have a restriction whereby you can only have one memory interface in each I/O sub-bank (for example, I/O sub-banks 1A, 1B, and 1C) if you are using leveling delay chain. This is because there is only one leveling delay chain per I/O sub-bank.

Figure 3 shows the Stratix III and Stratix IV device families that have up to 24 I/O banks.

Figure 3. Stratix III and Stratix IV External Memory Support *(Note 1)*



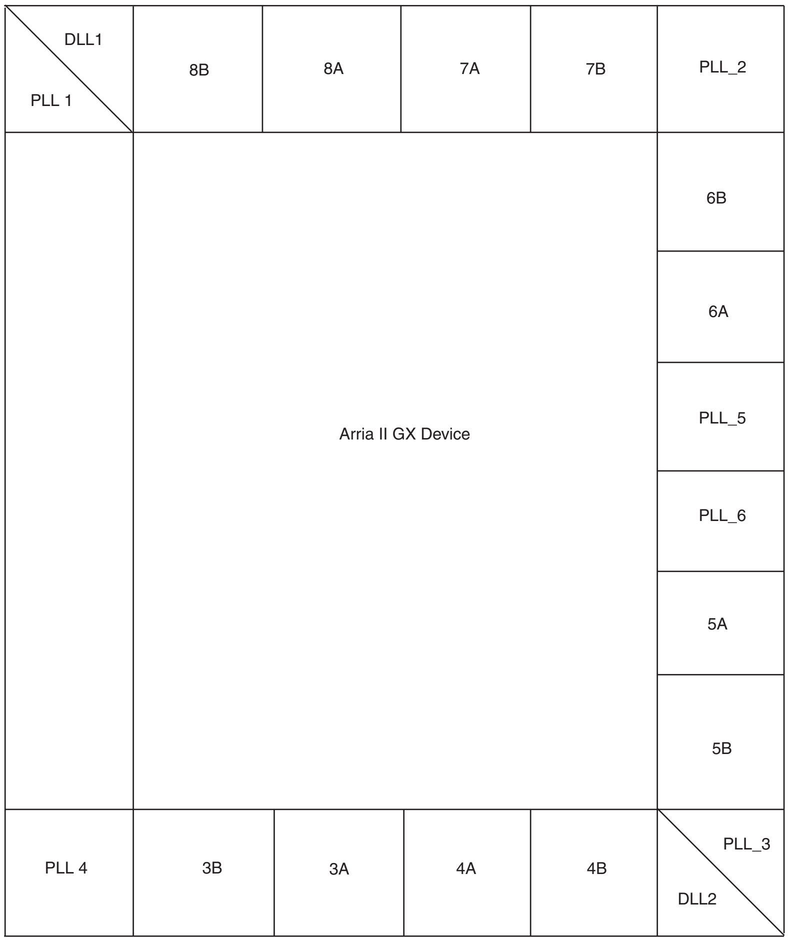
Note to Figure 3:

(1) The number of I/O banks and PLLs available depends on the device density. For more information on the availability of the I/O bank and PLL for each device, refer to Stratix III or Stratix IV pin-out files at www.altera.com/literature/lit-dp.jsp.

Arria II GX devices support memory interfaces in top, bottom, and right sides of the I/O banks. This device family has up to 12 I/O banks, as shown in Figure 4. You can have multiple memory interfaces in an Arria II GX I/O bank, provided that you have enough pins and clock network resources.

Figure 4 shows the Arria II GX device family that has up to 12 I/O banks.

Figure 4. Arria II GX External Memory Support *(Note 1)*



Note for Figure 4:

(1) The number of I/O banks and PLLs available depends on the device density. For more information on the availability of I/O banks and PLLs for each device, refer to the Arria II GX pin-out file at www.altera.com/literature/lit-dp.jsp.

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.

Table 1 shows a summary of the number of pins required for various memory interfaces.

Table 1 assumes that series on-chip termination (OCT) with calibration, parallel OCT with calibration, or dynamic calibrated OCT are used. The type of OCT selected is shown by the usage of R_{UP} and R_{DN} pins.

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

Memory Interface	FPGA Bus Width	Number of DQ Pins	Number of DQS Pins	Number of DM/BWSn Pins	Number of Address Pins *(3)*	Number of Command Pins	Number of Clock Pins	R_{UP}/R_{DN} Pins *(10)*	Total Pins
DDR3 SDRAM *(4)*, *(5)*	×4	4	2	0 *(6)*	14	10	2	2	34
	×8	8	2	1	14	10	2	2	39
	×16	16	4	2	14	10	2	2	50
	×72	72	18	9	16	15	4	2	136
DDR2 SDRAM *(7)*	×4	4	1	0 *(6)*	15	9	2	2	33
	×8	8	1	1	15	9	2	2	38
	×16	16	2	2	14	9	2	2	47
	×72	72	9	9	14	12	6	2	124
DDR SDRAM *(5)*	×4	4	1	0 *(6)*	14	7	2	2	28
	×8	8	1	1	14	7	2	2	33
	×16	16	2	2	14	7	2	2	43
	×72	72	9	9	13	9	6	2	118
QDRII+ SRAM	×9	18	2	1	19	3 *(8)*	4	2	49
	×18	36	2	2	18	3 *(8)*	4	2	67
	×36	72	2	4	17	3 *(8)*	4	2	104
QDRII SRAM	×9	18	2	1	19	2	4	2	48
	×18	36	2	2	18	2	4	2	66
	×36	72	2	4	17	2	4	2	103

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

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

Notes to Table 1:

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

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

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

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

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

(6) Altera FPGAs do not support DM pins in ×4 mode.

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

(8) The QVLD pin, used to indicate read data valid from the QDRII+ SRAM or RLDRAM II device, is included in this number.

(9) The ALTMEMPHY megafunction in version 8.0 of the Quartus II software does not support RLDRAM II interfaces.

(10) Some DQ/DQS pins are dual purpose and can also be used as RUP, RDN, or configuration pins. A DQ/DQS group is lost if you use these pins for configuration or as RUP or RDN pins for calibrated OCT. Pick RUP/RDN pins in a DQS/DQ group that is not being used for memory interface purposes. You may need to place the DQS and DQ pins manually if you place the RUP and RDN pins in the same DQS/DQ group pins.

For the device DQS/DQ pin count, refer to the *External Memory Interfaces* chapter of the respective device family handbook.

When using the ALTMEMPHY megafunction for Arria GX, HardCopy II, Stratix II, and Stratix II GX devices, you can only target the top and bottom of the I/O banks (I/O banks 3, 4, 7, and 8).

Arria II GX, HardCopy III, HardCopy IV, Stratix III, and Stratix IV devices support external memory interfaces in every I/O bank.

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 information on the DLL in each device family and to decide whether you must share the DLLs for your multiple memory controllers based on your frequency and memory data width requirements, refer to "HardCopy III, HardCopy IV, Stratix III, and Stratix IV Devices" on page 10, "Arria II GX Devices" on page 10, and "Arria GX, HardCopy II, Stratix II, and Stratix II GX Devices" on page 10.

HardCopy III, HardCopy IV, Stratix III, and Stratix IV Devices

HardCopy III, HardCopy IV, Stratix III, and Stratix IV device DLLs are located in each corner of the device, allowing them to shift any DQS pins located on the adjacent sides to the DLLs. There are a total of four DLLs per device, so HardCopy III, HardCopy IV, Stratix III, and Stratix IV devices can have a maximum of four memory interfaces with unique frequencies. For information on the method to share DLLs in these devices, refer to "Sharing DLLs" on page 18.

Arria II GX Devices

There are two DLLs in an Arria II GX device, located in the top-left and bottom-right corners of the device. These two DLLs can support a maximum of two unique frequencies, with each DLL running at one frequency. Each DLL can access the top, bottom, and right side of the device. This means that each I/O bank is accessible by two DLLs, giving more flexibility to create multiple frequencies and multiple-type interfaces. The DLL outputs the same DQS delay settings for the different sides of the device. For information on the method to share DLLs in the Arria II GX device, refer to "Sharing DLLs" on page 18.

☞ Version 9.0 of the Quartus II software only supports the hybrid mode for top and right sides and bottom and right sides of the device. The hybrid mode for top, bottom and right sides of the device will be supported in the future version of the Quartus II software.

Arria GX, HardCopy II, Stratix II, and Stratix II GX Devices

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. For information on the method to share DLLs in these devices, refer to "Sharing DLLs" on page 18.

Cyclone III Devices

Cyclone III devices do not use a DLL to interface with memory devices.

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.

Table 2 shows a summary of the clocks used in the ALTMEMPHY megafunction. All the clocks run at the same frequency as the I/O frequency, except for the `phy_clk_1x` clock, which runs at half the frequency of the interface if the controller is in half-rate mode, or at the same frequency as that of the memory interface in full-rate mode. The `resynch_clk_2x` and `measure_clk_2x` clocks use the PLL reconfiguration ability to change the phase shift at run-time to adjust for voltage and temperature (VT) variations.

☞ Half-rate DDR/DDR2 ALTMEMPHY megafunction for Stratix III devices cannot interleave memory controller within each other due to the half-rate resynchronization clock (`resynch_clk_1x`) which is cascaded from DQ group to DQ group.

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

Clock Name	Usage Description
`phy_clk_1x`	Static system clock for the half-rate data path and controller.
`mem_clk_2x`	Static DQS output clock used to generate DQS, CK/CK# signals, the input reference clock to the DLL, and the system clock for the full-rate datapath and controller.
`mem_clk_1x`	This clock drives the `aux_clk` output or clocking DQS and as a reference clock for the memory devices.
`write_clk_2x`	Static DQ output clock used to generate DQ signals at 90^o earlier than DQS signals. Also may be used to generate the address and command signals.
`mem_clk_ext_2x`	This clock is only used if the memory clock generation uses dedicated output pins. Applicable only in HardCopy II or Stratix II prototyping for HardCopy II designs.
`resynch_clk_2x`	Dynamic-phase clock used for resynchronization and postamble paths. Currently, this clock cannot be shared by multiple interfaces.
`measure_clk_2x/` `measure_clk_1x` *(3)*	Dynamic-phase clock used for VT tracking purposes. Currently, this clock cannot be shared by multiple interfaces.
`ac_clk_2x` `ac_clk_1x`	Dedicated static clock for address and command signals.
`scan_clk`	Static clock to reconfigure the PLL
`seq_clk`	Static clock for the sequencer logic

Notes to Table 2:

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

(2) For more information on the clock usage and clock network type, refer to the *ALTMEMPHY Megafunction User Guide*.

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

Table 3 and Table 4 show a comparison of the number of PLLs and dedicated clock outputs available respectively in Arria GX, Arria II GX, Cyclone III, HardCopy II, HardCopy III, HardCopy IV, Stratix II, Stratix II GX, Stratix III, and Stratix IV devices.

Table 3. Number of PLLs Available in Altera Device Families *(Note 1)*

Device Family	Fast PLL *(2)*	Enhanced PLL
Arria GX	2-4	2-4
Arria II GX	N/A	4-6 *(3)*
Cyclone III	N/A	2-4 *(3)*
HardCopy II	2-8	2-4
HardCopy III	N/A	4-12 *(3)*
HardCopy IV	N/A	4-12 *(3)*
Stratix II	4-8	2-4
Stratix II GX	2-4	2-4
Stratix III	N/A	4-12 *(3)*
Stratix IV	N/A	3-12 *(3)*

Notes to Table 3:

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

(2) You cannot use Stratix II and Stratix II GX Fast PLLs in the ALTMEMPHY megafunction, as they do not have enough clock outputs to meet the requirements of the ALTMEMPHY megafunction.

(3) Arria II GX, Cyclone III, HardCopy III, Stratix IV, HardCopy IV, and Stratix III PLLs are not the same as Stratix II Enhanced PLLs, but compared with Stratix II PLLs, they are more similar to the Enhanced PLLs than to the Fast PLLs.

Table 4. Number of Enhanced PLL Clock Outputs and Dedicated Clock Outputs Available in Altera Device Families *(Note 1)* (Part 1 of 2)

Device Family	Number of Enhanced PLL Clock Outputs	Number Dedicated Clock Outputs
Arria GX	6 clock outputs each	Three differential or six single-ended total (top and bottom I/O banks only)
Arria II GX *(2)*	7 clock outputs each	Single-ended or 1 differential pair 3 single-ended or 3 differential pairs *(3)*
Cyclone III	5 clock outputs each	1 single-ended or 1 differential pair total (not for memory interface use)
HardCopy II	6 clock outputs each	Three differential or six single-ended total (top and bottom I/O banks only)
HardCopy III	Left/right: 7 clock outputs Top/bottom: 10 clock outputs	Left/right: 2 single-ended or 1 differential pair Top/bottom: 6 single-ended or 4 single-ended and 1 differential pair

Table 4. Number of Enhanced PLL Clock Outputs and Dedicated Clock Outputs Available in Altera Device Families *(Note 1)* (Part 2 of 2)

Device Family	Number of Enhanced PLL Clock Outputs	Number Dedicated Clock Outputs
HardCopy IV	Left/right: 7 clock outputs Top/bottom: 10 clock outputs	Left/right: 2 single-ended or 1 differential pair Top/bottom: 6 single-ended or 4 single-ended and 1 differential pair
Stratix II	6 clock outputs each	Three differential or six single-ended total (top and bottom I/O banks only)
Stratix II GX	6 clock outputs each	Three differential or six single-ended total (top and bottom I/O banks only)
Stratix III	Left/right: 7 clock outputs Top/bottom: 10 clock outputs	Left/right: 2 single-ended or 1 differential pair Top/bottom: 6 single-ended or 4 single-ended and 1 differential pair
Stratix IV	Left/right: 7 clock outputs Top/bottom: 10 clock outputs	Left/right: 2 single-ended or 1 differential pair Top/bottom: 6 single-ended or 4 single-ended and 1 differential pair

Note to Table 4:

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

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

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

Table 5 shows the number of clock networks available in the Altera device families.

Table 5. Number of Clock Networks Available in Altera Device Families *(Note 1)* (Part 1 of 2)

Device Family	Global Clock Network	Regional Clock Network
Arria GX	16	32
Arria II GX	16	48
Cyclone III	10-20	N/A
HardCopy II	16	32
HardCopy III	16	88
HardCopy IV	16	88

Table 5. Number of Clock Networks Available in Altera Device Families *(Note 1)* (Part 2 of 2)

Device Family	Global Clock Network	Regional Clock Network
Stratix II	16	32
Stratix II GX	16	32
Stratix III	16	64-88
Stratix IV	16	64-88

Note to Table 5:

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

You must decide whether you need to share clock networks, PLL clock outputs, or PLLs 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.

Resources Planning for DLL and PLL Static Clocks

Below are the different scenarios for the resources sharing for DLL and PLL static clocks:

- Multiple Controllers with No Sharing — You need to check the number of PLLs and DLLs that available in the targeted device and the number of PLLs and DLLs that needed in your design. If you have enough resources for the controllers, you do not need to share the DLL and PLL static clocks.
- Multiple Controllers with Sharing DLL only — If you have enough resources on PLL clocks output and only need to share DLL, ensure that the controllers are running at the same memory clock frequency.
- Multiple Controllers with Sharing PLL Static Clocks only — If the controllers are on the opposite device edge, you can only share the PLL static clocks. Each controller has their own DLL as the DLL can only access the adjacent sides from its location. Cyclone III devices do not use DLL to interface with memory devices, thus only PLL static clocks are shared.
- Multiple Controllers with Sharing DLL and PLL Static Clocks — The PLL static clocks and DLL are shared when the controllers are on the same side or adjacent side of the device and provided they are running at the same memory-clock frequency. Arria GX, Stratix II, and Stratix II GX devices only support memory interfaces on the top and bottom I/O banks. Only Arria II GX, Stratix III, and Stratix IV devices can support hybrid mode as the DLL for these devices families can access the adjacent side from its location in the device.

It is recommended to use dedicated clock input pin for the PLL reference clock input. If you do not use dedicated clock input pin, Quartus II software will give critical warning due to increased jitter and worse timing margin.

For Stratix III and Stratix IV devices, the adjacent PLLs on the same side of the device can share a dedicated input clock for their PLL reference clock input. If LVDS is used on the left/right side of the device (which must be powered by 2.5 V), you can also use a 2.5-V LVDS input clock for PLL reference input clock for PLL at top or bottom side of the device.

Creating the Project Topology

After understanding the device resource limitations, you must determine the resources to share for your multiple controller design. You can use this as a framework for your Quartus II design constraints. For example, Stratix II external memory interfaces with an ALTMEMPHY megafunction data path are supported only on the top and bottom side of the device. If you also have a PCI interface in your design, you must leave at least one bank (either on the top or the bottom of the device) for the PCI interface. For this particular example interface, you may need to share a DLL.

Creating PHY and Controller in a Quartus II Project

You can instantiate multiple controllers using either one of the following flows:

- "SOPC Builder Flow"
- "MegaWizard Plug-in Manager Flow"

The SOPC Builder flow provides useful validation and automatically generates all the wiring in SOPC Builder system and assignment. For the MegaWizard Plug-in Manager flow, you must manually set the assignment for the PLL sharing but this flow gives you more flexibility.

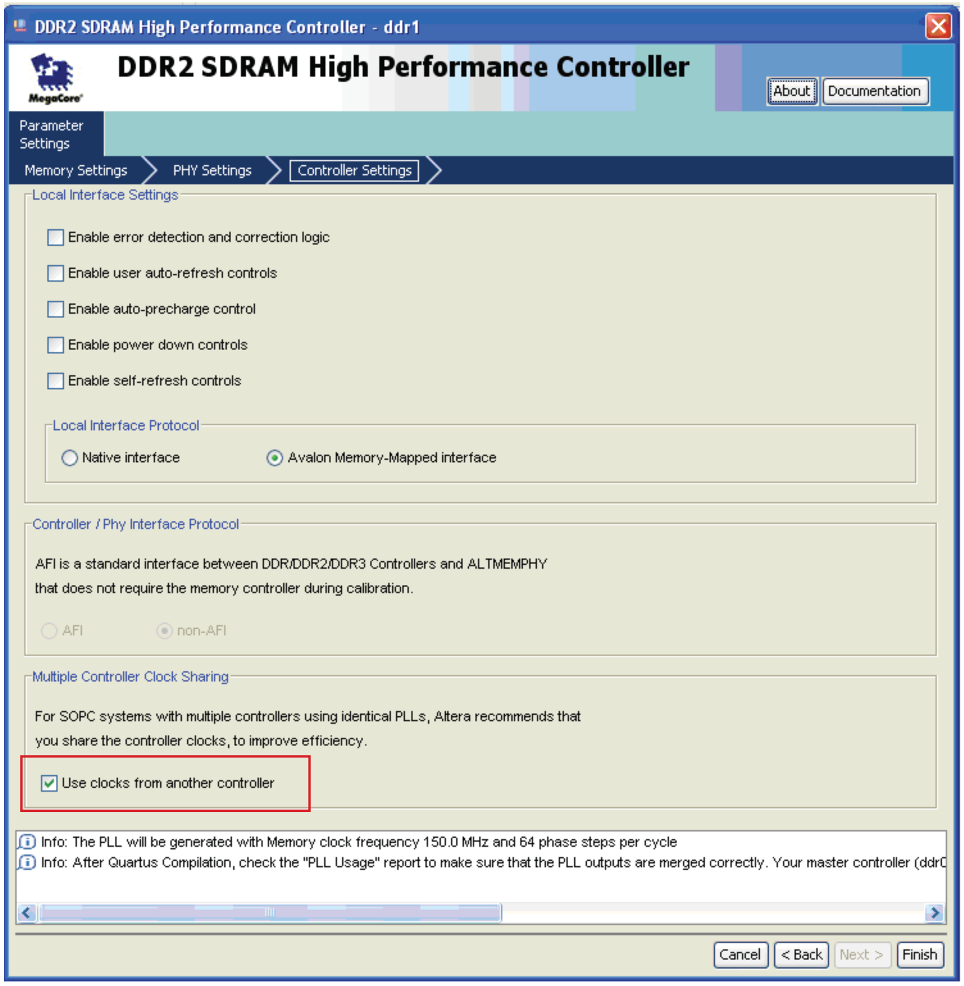
There are some regulations that you must follow when you want to share the clock between controllers:

- The controllers must be configured to have the same local interface clock frequency, either in full-rate or half-rate mode
- The controllers must have the same PLL input and output frequencies
- The controllers must have compatible memory types (DDR SDRAM and DDR2 SDRAM are mixed in the same system but the DDR3 SDRAM controller is not compatible with DDR/DDR2 SDRAM)
- The controllers must have their `ref_clk` input connected to the same signal

SOPC Builder Flow

The SOPC Builder in version 8.1 and onwards of the Quartus II software allows you to add multiple controllers directly to a new or existing SOPC Builder system. The SOPC Builder supports sharing static PHY clocks between multiple controllers in the SOPC Builder that are running on the same frequency and sharing the same PLL reference clock. To share the static clocks, you must turn on the **Clock source from another controller** option in the **Controller Settings** page of the DDR3/DDR2/DDR SDRAM High Performance Controller, as shown in Figure 5. This option must be turned on for the slave controllers. Turning on this option adds a new clock input connection point to the slave controller named `shared_sys_clk`. You must connect the `sys_clk` signal from the master controller to the `shared_sys_clk` signal of the slave controller as shown in Figure 6. The **Force Merging of PLL Clock Fanouts** assignment will automatically be assigned in the Quartus II software.

Figure 5. Enabling Multiple Controllers Sharing Clock for Slave Controllers



Local Interface Settings

- [ ] Enable error detection and correction logic
- [ ] Enable user auto-refresh controls
- [ ] Enable auto-precharge control
- [ ] Enable power down controls
- [ ] Enable self-refresh controls

Local Interface Protocol

Native interface [✓ Avalon Memory-Mapped interface]

Controller / Phy Interface Protocol

AFI is a standard interface between DDR/DDR2/DDR3 Controllers and ALTMEMPHY that does not require the memory controller during calibration.

AFI [✓ non-AFI]

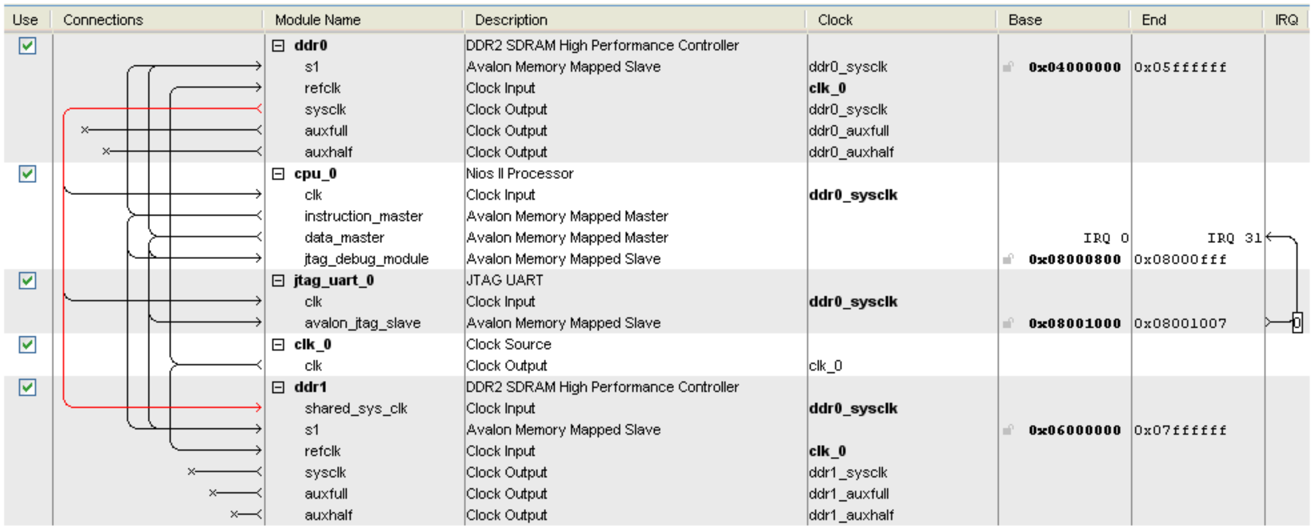
Multiple Controller Clock Sharing

For SOPC systems with multiple controllers using identical PLLs, Altera recommends that you share the controller clocks, to improve efficiency.

- [✓] Use clocks from another controller

```
Info: The PLL will be generated with Memory clock frequency 150.0 MHz and 64 phase steps per cycle
Info: After Quartus Compilation, check the "PLL Usage" report to make sure that the PLL outputs are merged correctly. Your master controller (ddr0
```

Figure 6. Connection Between Master and Slave Controller



Use	Connections	Module Name	Description	Clock	Base	End	IRQ
✓		ddr0	DDR2 SDRAM High Performance Controller				
		s1	Avalon Memory Mapped Slave	ddr0_sysclk	0x04000000	0x05ffffff	
		refclk	Clock Input	clk_0			
		sysclk	Clock Output	ddr0_sysclk			
		auxfull	Clock Output	ddr0_auxfull			
		auxhalf	Clock Output	ddr0_auxhalf			
✓		cpu_0	Nios II Processor				
		clk	Clock Input	ddr0_sysclk			
		instruction_master	Avalon Memory Mapped Master				
		data_master	Avalon Memory Mapped Master		IRQ 0	IRQ 31	
		jtag_debug_module	Avalon Memory Mapped Slave		0x08000800	0x08000fff	
✓		jtag_uart_0	JTAG UART				
		clk	Clock Input	ddr0_sysclk			
		avalon_jtag_slave	Avalon Memory Mapped Slave		0x08001000	0x08001007	0
✓		clk_0	Clock Source				
		clk	Clock Output	clk_0			
✓		ddr1	DDR2 SDRAM High Performance Controller				
		shared_sys_clk	Clock Input	ddr0_sysclk			
		s1	Avalon Memory Mapped Slave		0x06000000	0x07ffffff	
		refclk	Clock Input	clk_0			
		sysclk	Clock Output	ddr1_sysclk			
		auxfull	Clock Output	ddr1_auxfull			
		auxhalf	Clock Output	ddr1_auxhalf			

After the system generation, you must add the SDC file and run the pin assignment TCL scripts as usual.

When an SOPC Builder multiple controllers system does not share the `sys_clk`, the SOPC Builder inadvertently insert clock-crossing adaptors between the controller and the NIOS II processor. In this situation, the system cannot achieve the maximum performance for both controllers.

The SOPC Builder checks all the conditions listed in the regulation section and prevents you from generating the system when they are not met.

For more information on how to use controllers with SOPC Builder, refer to *AN 517: Using High-Performance DDR, DDR2, and DDR3 SDRAM With SOPC Builder*.

For more information on DDR, DDR2, and DDR3 SDRAM High Performance Controllers, refer to *DDR and DDR2 SDRAM high-Performance Controller User Guide* and the *DDR3 SDRAM High-Performance Controller User Guide*.

For more information on simulating SOPC Builder systems, refer to *AN 351: Simulating Nios II Systems*.

MegaWizard Plug-in Manager Flow

If you are creating multiple instances of the same controller with the same parameters (frequency of operation, data width, burst mode, etc.), you only need to generate the controller once. You can then instantiate this controller variation multiple times in your top-level design file. If you are using controllers with different parameters, or plan to modify the RTL generated by the MegaWizard Plug-In Manager (as required for resource sharing in some cases), you must generate each variation of the memory controller individually.

The high-performance memory controller is generated with a top-level design, **<***variation_name***>_example_top.v/.vhd**, in which *variation_name* is the name of the memory controller that you entered in the first page of the MegaWizard Plug-In Manager. This top-level design example instantiates the memory controller (**<***variation_name***>.v/.vhd**) and an example driver which performs a comparison test after reading back data that was written to the memory. You can also generate simulation files for each high-performance controller or ALTMEMPHY megafunction.

When creating a multiple memory interface design, you have the option to combine certain aspects of the design flow, for example:

- You can use one of the top-level designs to instantiate all the memory controllers in the design, or create a new top-level design.
- You can either modify one of the example drivers to test all the memory interfaces in the design, or instantiate an example driver with each memory controller.
- You can simulate each memory interface separately, or create a testbench which combines all the memory interfaces in the design.

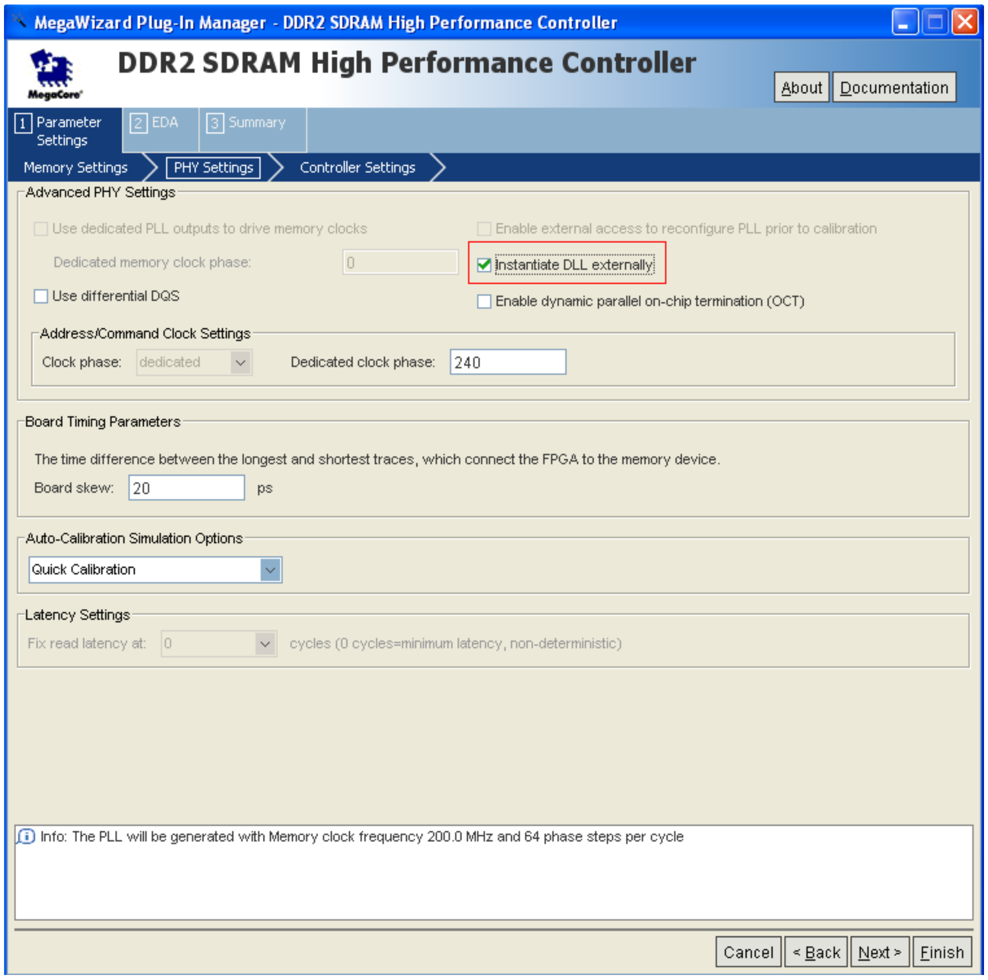
For more information on how to use the ALTMEMPHY megafunction, refer to the *ALTMEMPHY Megafunction User Guide* and the *DDR and DDR2 SDRAM High-Performance Controller User Guide*.

Sharing DLLs

If the controllers are sharing a DLL, ensure that the **Instantiate DLL externally** option is turned on in the **PHY Settings** page of the DDR3/DDR2/DDR SDRAM High-Performance Controller or the ALTMEMPHY MegaWizard Plug-In Manager, as shown in Figure 4. This option must be checked for every interface that is sharing a DLL.

Figure 7 shows you how to allow the DLL to be instantiated externally for controllers sharing DLLS.

Figure 7. Allowing the DLL to be Instantiated Externally for Controllers Sharing DLLs



Advanced PHY Settings

- [ ] Use dedicated PLL outputs to drive memory clocks
- Dedicated memory clock phase: 0
- [ ] Use differential DQS
- [ ] Enable external access to reconfigure PLL prior to calibration
- [✓] Instantiate DLL externally
- [ ] Enable dynamic parallel on-chip termination (OCT)

Address/Command Clock Settings

Clock phase: dedicated   Dedicated clock phase: 240

Board Timing Parameters

The time difference between the longest and shortest traces, which connect the FPGA to the memory device.

Board skew: 20 ps

Auto-Calibration Simulation Options

Quick Calibration

Latency Settings

Fix read latency at: 0 cycles (0 cycles=minimum latency, non-deterministic)

Info: The PLL will be generated with Memory clock frequency 200.0 MHz and 64 phase steps per cycle

For more information on instantiating the controllers, refer to the *DDR and DDR2 SDRAM High-Performance Controller User Guide*.

When the **Instantiate DLL externally** option is checked, the MegaWizard Plug-In Manager generates a file called <*variation_name*>**_phy_alt_mem_phy_dll_**<*device_family*>**.v/.vhd**, in which the device family is:

- **sii** when targeting Arria GX, HardCopy II, Stratix II, or Stratix II GX devices
- **siii** when targeting HardCopy III, HardCopy IV, Stratix III, or Stratix IV devices
- **aii** when targeting Arria II GX devices

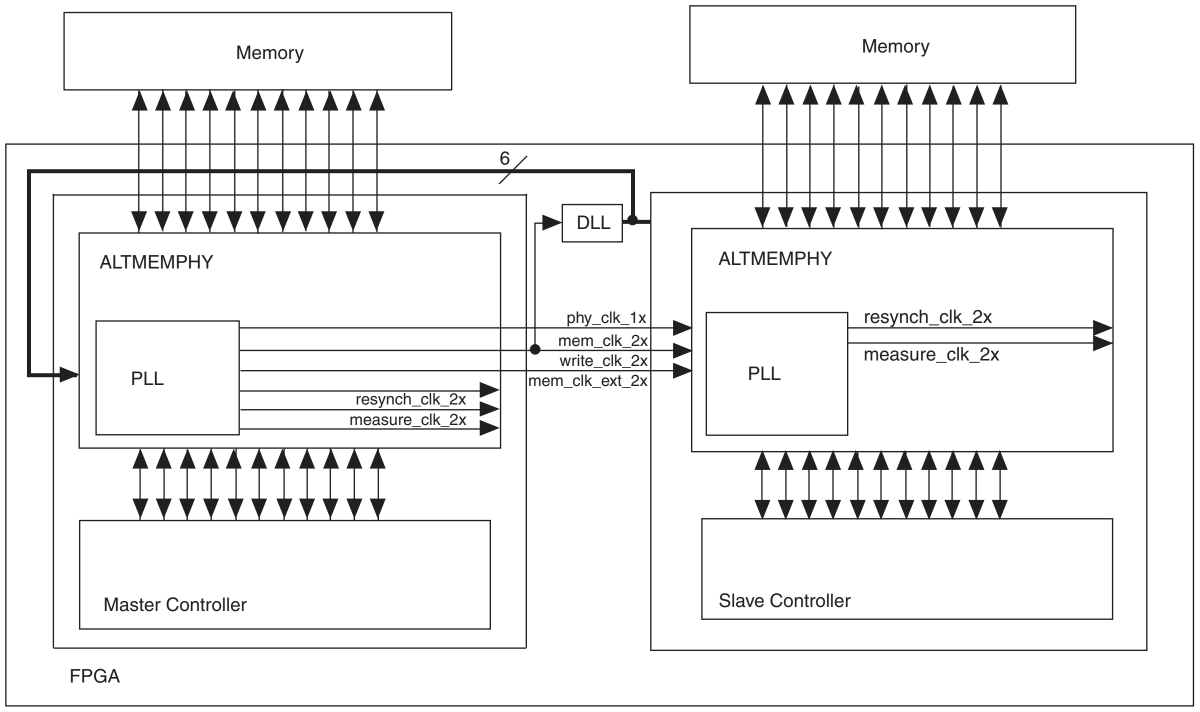
The example top design then instantiates the DLL in addition to instantiating the memory controller and the example driver. You can then instantiate the other memory controllers for the design and either modify the example driver to create the test pattern for all controllers or create another design example for each controller that is instantiated.

☞ If you do not need to share any other clock networks or PLLs, you can continue to "Adding Constraints to the Design" on page 24.

Sharing PLL Clock Outputs or Clock Networks

You can share the static clock networks in multiple memory controllers using ALTMEMPHY megafunction. Although a PLL instance is used for each ALTMEMPHY instance, the static clocks are shared, which uses fewer clocking resources. This is an easy-to-implement option that reduces the required number of clocks. Figure 8 shows a diagram of the connection between the DLL and the PLL for this scheme.

Figure 8. Two Controllers Static Clocks with Separate Resynchronization Clocks from Different PLLs



The maximum number of interfaces that are shared in this scheme is limited by the number of PLLs and the number of pins available in the device. You must ensure that the memory devices that are accessed by the different controllers are laid out on the board with the same trace lengths.

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.

Table 6 shows a list of PLL merging assignments.

Table 6. PLL Merging Assignments *(Note 1)* (Part 1 of 3)

Device *(2)*	**Rate**	**Assigments**

Arria II GX	Half	`set_instance_assignment -name FORCE_MERGE_PLL_FANOUTS ON -from *|<SLAVE>_phy_alt_mem_phy_clk_reset:clk|<SLAVE>_phy_alt_mem_phy_pll:*:altpll_component|*:auto_generated|clk[0] -to *|<MASTER>_phy_alt_mem_phy_clk_reset:clk|<MASTER>_phy_alt_mem_phy_pll:*:altpll_component|*:auto_generated|clk[0]`
		
		 `set_instance_assignment -name FORCE_MERGE_PLL_FANOUTS ON -from *|<SLAVE>_phy_alt_mem_phy_clk_reset:clk|<SLAVE>_phy_alt_mem_phy_pll:*:altpll_component|*:auto_generated|clk[3] -to *|<MASTER>_phy_alt_mem_phy_clk_reset:clk|<MASTER>_phy_alt_mem_phy_pll:*:altpll_component|*:auto_generated|clk[3]`

	Full	`set_instance_assignment -name FORCE_MERGE_PLL_FANOUTS ON -from *|<SLAVE>_phy_alt_mem_phy_clk_reset:clk|<SLAVE>_phy_alt_mem_phy_pll:*:altpll_component|*:auto_generated|clk[1] -to *|<MASTER>_phy_alt_mem_phy_clk_reset:clk|<MASTER>_phy_alt_mem_phy_pll:*:altpll_component|*:auto_generated|clk[1]`
		
		 `set_instance_assignment -name FORCE_MERGE_PLL_FANOUTS ON -from *|<SLAVE>_phy_alt_mem_phy_clk_reset:clk|<SLAVE>_phy_alt_mem_phy_pll:*:altpll_component|*:auto_generated|clk[3] -to *|<MASTER>_phy_alt_mem_phy_clk_reset:clk|<MASTER>_phy_alt_mem_phy_pll:*:altpll_component|*:auto_generated|clk[3]`

Table 6. PLL Merging Assignments *(Note 1)* (Part 2 of 3)

Device *(2)*	**Rate**	**Assigments**
Cyclone III	Half	set_instance_assignment -name FORCE_MERGE_PLL_FANOUTS ON -from *|<SLAVE>_phy_alt_mem_phy_clk_reset:clk|<SLAVE>_phy_alt_mem_phy_pll:*:altpll_component|*:auto_generated|clk[0] -to *|<MASTER>_phy_alt_mem_phy_clk_reset:clk|<MASTER>_phy_alt_mem_phy_pll:*:altpll_component|*:auto_generated|clk[0] set_instance_assignment -name FORCE_MERGE_PLL_FANOUTS ON -from *|<SLAVE>_phy_alt_mem_phy_clk_reset:clk|<SLAVE>_phy_alt_mem_phy_pll:*:altpll_component|*:auto_generated|clk[2] -to *|<MASTER>_phy_alt_mem_phy_clk_reset:clk|<MASTER>_phy_alt_mem_phy_pll:*:altpll_component|*:auto_generated|clk[2]
	Full	set_instance_assignment -name FORCE_MERGE_PLL_FANOUTS ON -from *|<SLAVE>_phy_alt_mem_phy_clk_reset:clk|<SLAVE>_phy_alt_mem_phy_pll:*:altpll_component|*:auto_generated|clk[1] -to *|<MASTER>_phy_alt_mem_phy_clk_reset:clk|<MASTER>_phy_alt_mem_phy_pll:*:altpll_component|*:auto_generated|clk[1] set_instance_assignment -name FORCE_MERGE_PLL_FANOUTS ON -from *|<SLAVE>_phy_alt_mem_phy_clk_reset:clk|<SLAVE>_phy_alt_mem_phy_pll:*:altpll_component|*:auto_generated|clk[2] -to *|<MASTER>_phy_alt_mem_phy_clk_reset:clk|<MASTER>_phy_alt_mem_phy_pll:*:altpll_component|*:auto_generated|clk[2]
Stratix II	Half	set_instance_assignment -name FORCE_MERGE_PLL_FANOUTS ON -from *|<SLAVE>_phy_alt_mem_phy_clk_reset:clk|<SLAVE>_phy_alt_mem_phy_pll:*:altpll_component|*clk0 -to *|<MASTER>_phy_alt_mem_phy_clk_reset:clk|<MASTER>_phy_alt_mem_phy_pll:*:altpll_component|*clk0 set_instance_assignment -name FORCE_MERGE_PLL_FANOUTS ON -from *|<SLAVE>_phy_alt_mem_phy_clk_reset:clk|<SLAVE>_phy_alt_mem_phy_pll:*:altpll_component|*clk2 -to *|<MASTER>_phy_alt_mem_phy_clk_reset:clk|<MASTER>_phy_alt_mem_phy_pll:*:altpll_component|*clk2
	Full	set_instance_assignment -name FORCE_MERGE_PLL_FANOUTS ON -from *|<SLAVE>_phy_alt_mem_phy_clk_reset:clk|<SLAVE>_phy_alt_mem_phy_pll:*:altpll_component|*clk1 -to *|<MASTER>_phy_alt_mem_phy_clk_reset:clk|<MASTER>_phy_alt_mem_phy_pll:*:altpll_component|*clk1 set_instance_assignment -name FORCE_MERGE_PLL_FANOUTS ON -from *|<SLAVE>_phy_alt_mem_phy_clk_reset:clk|<SLAVE>_phy_alt_mem_phy_pll:*:altpll_component|*clk2 -to *|<MASTER>_phy_alt_mem_phy_clk_reset:clk|<MASTER>_phy_alt_mem_phy_pll:*:altpll_component|*clk2

Table 6. PLL Merging Assignments *(Note 1)* (Part 3 of 3)

Device *(2)*	**Rate**	**Assigments**

Stratix III	Half and Full	`set_instance_assignment -name FORCE_MERGE_PLL_FANOUTS ON -from *|<SLAVE>_phy_alt_mem_phy_clk_reset:clk|write_clk_2x -to *|<MASTER>_phy_alt_mem_phy_clk_reset:clk|write_clk_2x`
		
		`set_instance_assignment -name FORCE_MERGE_PLL_FANOUTS ON -from *|<SLAVE>_phy_alt_mem_phy_clk_reset:clk|phy_clk_1x -to *|<MASTER>_phy_alt_mem_phy_clk_reset:clk|phy_clk_1x`

Notes to Table 6:

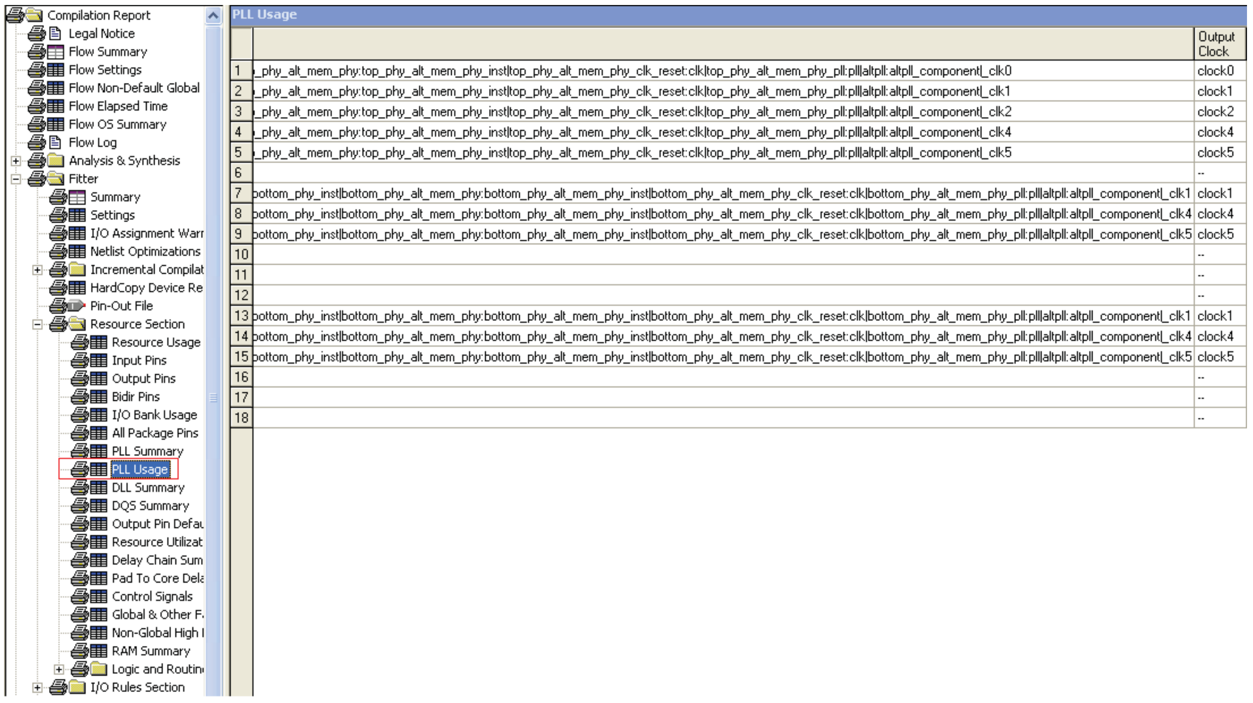
(1) In these assignments, if you are using the ALTMEMPHY megafunction, you must replace `<SLAVE>_phy` and `<MASTER>_phy` by the variation names of your two variations. If you are using the high performance controller, you must replace `<SLAVE>` and `<MASTER>` by the variation names of your two variations.

(2) Stratix II includes Arria GX, HardCopy II, Stratix II, or Stratix II GX devices; Stratix III includes HardCopy III, HardCopy IV, Stratix III, or Stratix IV devices.

To check if the Quartus II software has applied the assignments, you need to read the Compilation Report - PLL Usage window (available only if the Fitter step is successful), which shows the two merged clocks.

Figure 9 shows PLL usage in the compilation report which shows the two clocks have been merged (clk0 and clk2)

Figure 9. Compilation Report PLL Usage



		Output Clock
1	_phy_alt_mem_phy:top_phy_alt_mem_phy_inst|top_phy_alt_mem_phy_clk_reset:clk|top_phy_alt_mem_phy_pll:pll|altpll:altpll_component|_clk0	clock0
2	_phy_alt_mem_phy:top_phy_alt_mem_phy_inst|top_phy_alt_mem_phy_clk_reset:clk|top_phy_alt_mem_phy_pll:pll|altpll:altpll_component|_clk1	clock1
3	_phy_alt_mem_phy:top_phy_alt_mem_phy_inst|top_phy_alt_mem_phy_clk_reset:clk|top_phy_alt_mem_phy_pll:pll|altpll:altpll_component|_clk2	clock2
4	_phy_alt_mem_phy:top_phy_alt_mem_phy_inst|top_phy_alt_mem_phy_clk_reset:clk|top_phy_alt_mem_phy_pll:pll|altpll:altpll_component|_clk4	clock4
5	_phy_alt_mem_phy:top_phy_alt_mem_phy_inst|top_phy_alt_mem_phy_clk_reset:clk|top_phy_alt_mem_phy_pll:pll|altpll:altpll_component|_clk5	clock5
6		--
7	bottom_phy_inst|bottom_phy_alt_mem_phy:bottom_phy_alt_mem_phy_inst|bottom_phy_alt_mem_phy_clk_reset:clk|bottom_phy_alt_mem_phy_pll:pll|altpll:altpll_component|_clk1	clock1
8	bottom_phy_inst|bottom_phy_alt_mem_phy:bottom_phy_alt_mem_phy_inst|bottom_phy_alt_mem_phy_clk_reset:clk|bottom_phy_alt_mem_phy_pll:pll|altpll:altpll_component|_clk4	clock4
9	bottom_phy_inst|bottom_phy_alt_mem_phy:bottom_phy_alt_mem_phy_inst|bottom_phy_alt_mem_phy_clk_reset:clk|bottom_phy_alt_mem_phy_pll:pll|altpll:altpll_component|_clk5	clock5
10		--
11		--
12		--
13	bottom_phy_inst|bottom_phy_alt_mem_phy:bottom_phy_alt_mem_phy_inst|bottom_phy_alt_mem_phy_clk_reset:clk|bottom_phy_alt_mem_phy_pll:pll|altpll:altpll_component|_clk1	clock1
14	bottom_phy_inst|bottom_phy_alt_mem_phy:bottom_phy_alt_mem_phy_inst|bottom_phy_alt_mem_phy_clk_reset:clk|bottom_phy_alt_mem_phy_pll:pll|altpll:altpll_component|_clk4	clock4
15	bottom_phy_inst|bottom_phy_alt_mem_phy:bottom_phy_alt_mem_phy_inst|bottom_phy_alt_mem_phy_clk_reset:clk|bottom_phy_alt_mem_phy_pll:pll|altpll:altpll_component|_clk5	clock5
16		--
17		--
18		--

If the report does not show the clocks merged as expected you must check the `FORCE_MERGE_PLL_FANOUTS` assignment carefully for incorrect clock names. You can also open your <*projectname*>**_fit.rpt** file and look for Merged PLL.

☞ QDRII ALTMEMPHY does not have dynamic clocks, thus you can share the static clocks among the QDRII ALTMEMPHYs. Use regional clock networks for high-speed interfaces. For lower-speed interfaces, use global clock network if the FPGA output clock meets the QDRII SRAM input jitter specification.

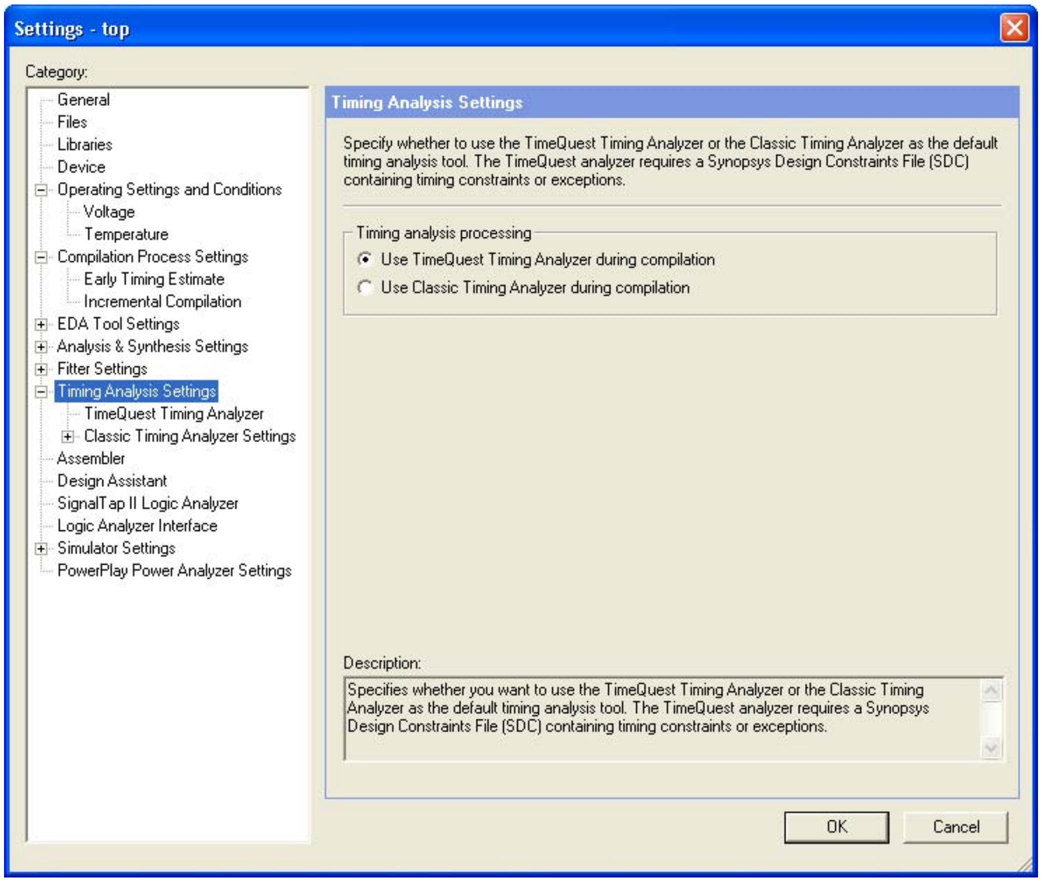
☞ You cannot merge dynamic clocks of the ALTMEMPHY megafunction or High-Performance Controller. The Quartus II software may not give a warning, but this will not work in the hardware.

Adding Constraints to the Design

The MegaWizard Plug-In Manager generates an **.sdc** file for timing constraints and **.tcl** scripts for I/O standard and DQS/DQ grouping constraints of each controller. The TimeQuest timing analyzer is the default timing analyzer for the Arria GX, Arria II GX, Cyclone III, Stratix III, and Stratix IV device families. To manually optimize timing with the Quartus II compiler and to use the timing report script created by the MegaWizard Plug-In Manager, you must enable the TimeQuest timing analyzer. To enable the TimeQuest timing analyzer manually for Stratix II and Stratix II GX designs, you must perform the following steps:

1. On the Assignments menu, click **Settings.** The **Settings** dialog box appears.
2. In the **Category** list, select **Timing Analysis Settings.** The **Timing Analysis Settings** page appears.
3. Under **Timing analysis processing**, select **Use TimeQuest Timing Analyzer** during compilation (Figure 10 shows how the **Settings** dialog box).
4. Click **OK.**

Figure 10. Enabling TimeQuest Timing Analyzer



Settings - top

Category:

Timing Analysis Settings

Specify whether to use the TimeQuest Timing Analyzer or the Classic Timing Analyzer as the default timing analysis tool. The TimeQuest analyzer requires a Synopsys Design Constraints File (SDC) containing timing constraints or exceptions.

Timing analysis processing

[✓ Use TimeQuest Timing Analyzer during compilation]
Use Classic Timing Analyzer during compilation

Description:

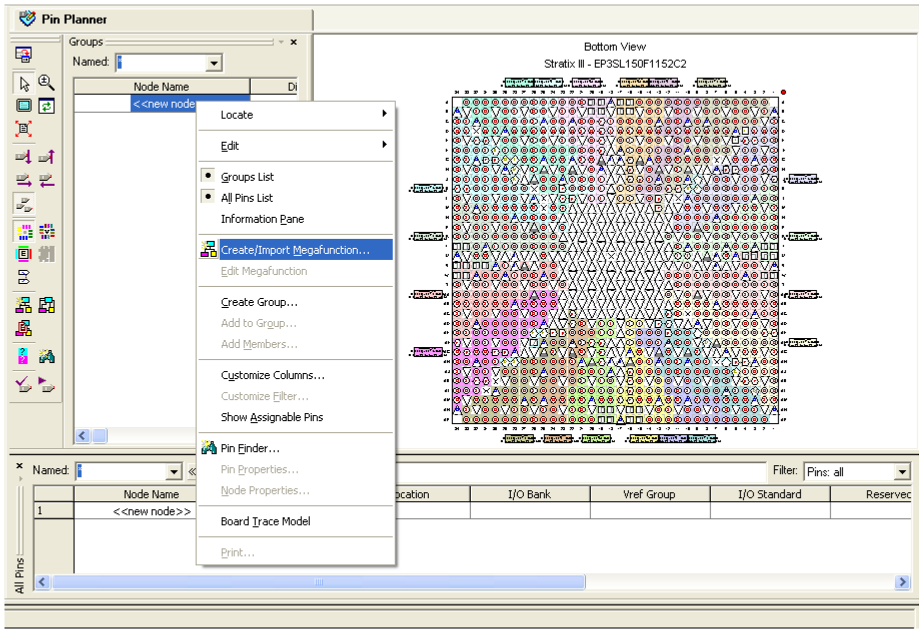
Specifies whether you want to use the TimeQuest Timing Analyzer or the Classic Timing Analyzer as the default timing analysis tool. The TimeQuest analyzer requires a Synopsys Design Constraints File (SDC) containing timing constraints or exceptions.

You must then add the **.sdc** file for each controller. If you are using two or more different memory controller variations, you must add the **.sdc** files generated for each variation. However, if your design consists of multiple instantiations of the same controller, you only need to add the **.sdc** file once without modification. The Quartus II software is able to apply the **.sdc** file for all the interfaces using the same variation.

The High-Performance Memory Controller MegaWizard Plug-In Manager also generates two **.tcl** scripts per variation to set the I/O standard, output enable grouping, termination, and current strength for the memory interface pins. These **.tcl** scripts use the default MegaWizard Plug-In Manager names, for example, `mem_dq[71..0]`, `local_ready`, and `mem_ras_n`. Every variation of the high-performance memory controller uses this naming convention. However, if there are multiple memory controllers in the design, you must change the names of the interface pins in the top-level file to differentiate each controller. You can add prefixes to the controller pin names in the Quartus II Pin Planner by performing the following steps:

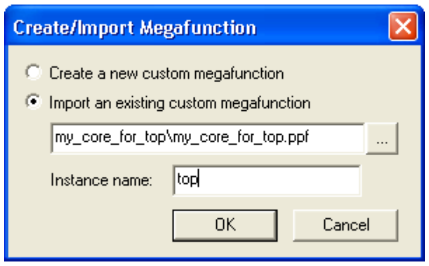
1. Open the Assignment menu and click on **Pin Planner**.
2. Right-click in any area under **Node Name**, and select **Create/Import Megafunction** as shown in Figure 11.

Figure 11. Create/Import Megafunction Option in the Pin Planner



3. Turn on the **Import an existing custom megafunction** radio button and choose the <*variation_name*>**.ppf** file.
4. Type the prefix that you want to use under the **Instance name** as shown in Figure 12.

Figure 12. Adding Prefix to the Memory Interface Pin Names

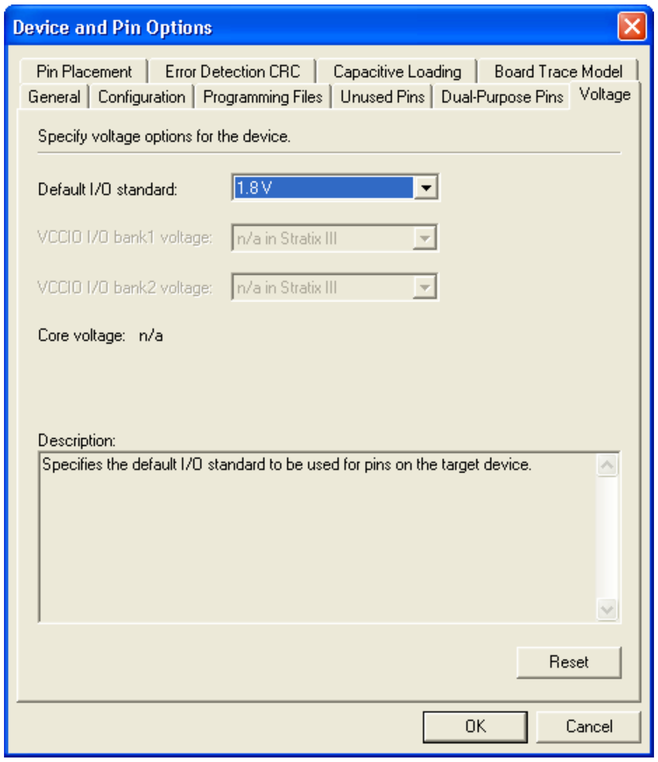

Create/Import Megafunction
Create a new custom megafunction
Import an existing custom megafunction
OK
Cancel

Import an existing custom megafunction: my_core_for_top\my_core_for_top.ppf

Instance name: top

You may also want to set the default I/O standard for your design in the **Voltage** section of the **Device and Pin Options**, by navigating to **Assignment** and clicking **Setting** as shown in Figure 13.

Figure 13. Setting a Default Voltage for Your Design



Specify voltage options for the device.

Default I/O standard: 1.8 V

VCCIO I/O bank1 voltage: n/a in Stratix III

VCCIO I/O bank2 voltage: n/a in Stratix III

Core voltage: n/a

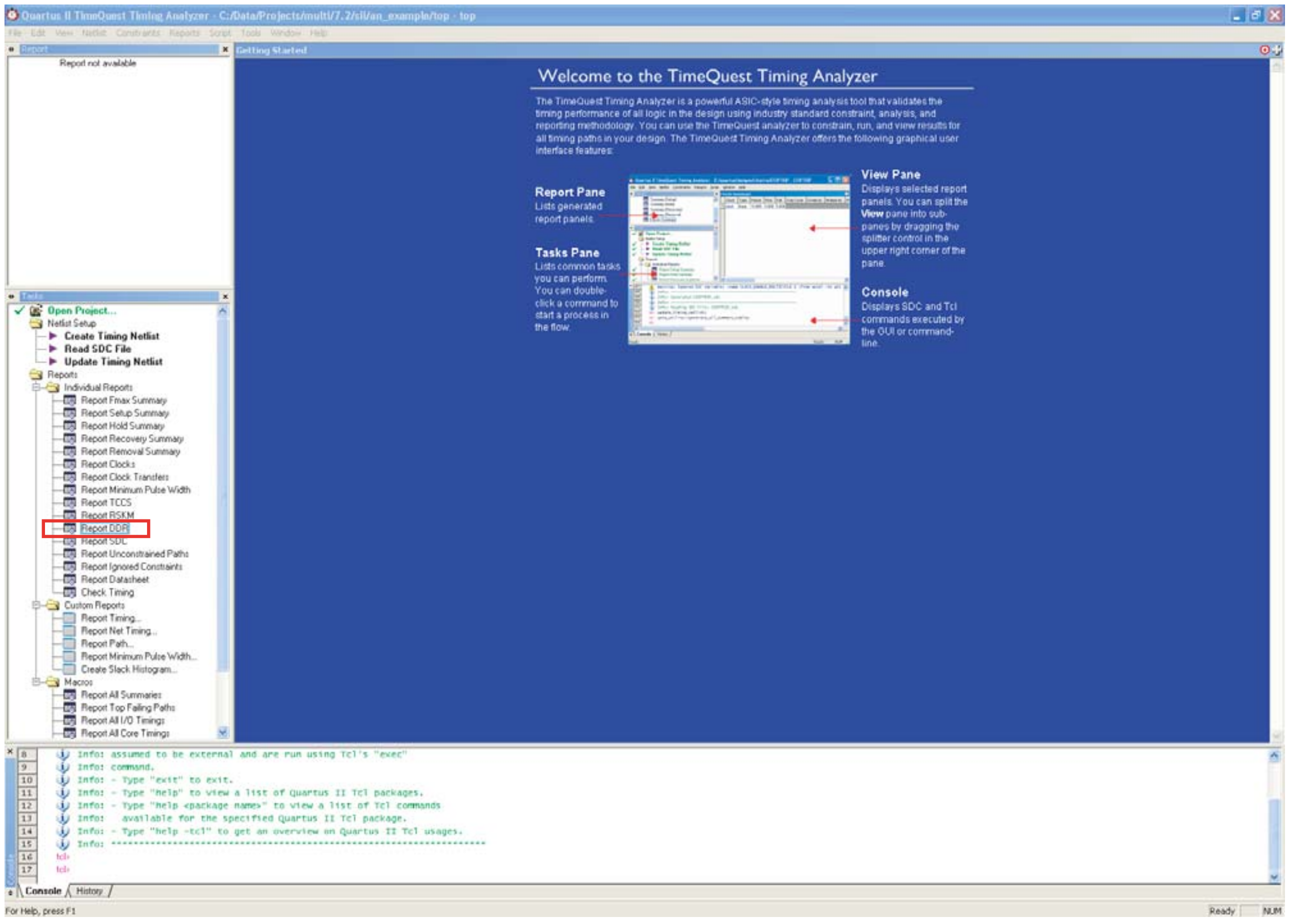
Description:
Specifies the default I/O standard to be used for pins on the target device.

Take note of the number of available pins per I/O bank to ensure that the Quartus II software can successfully compile the design.

Compiling the Design to Generate a Timing Report

After compiling the design, open the Tools menu and select **TimeQuest Timing Analyzer**. Double click on **Report DDR** (refer to Figure 14) to get the timing report for all ALTMEMPHY-based memory controllers in the design.

Figure 14. Report DDR in TimeQuest Timing Analyzer for ALTMEMPHY-Based Controller


Quartus II TimeQuest Timing Analyzer - C:/Data/Projects/multi/7.2/sil/an_example/top - top
File Edit View Netlist Constraints Reports Script Tools Window Help
Report
Getting Started
Tasks
Console
History
Ready
NUM

Report not available

- Open Project...
- Netlist Setup
  - Create Timing Netlist
  - Read SDC File
  - Update Timing Netlist
- Reports
  - Individual Reports
    - Report Fmax Summary
    - Report Setup Summary
    - Report Hold Summary
    - Report Recovery Summary
    - Report Removal Summary
    - Report Clocks
    - Report Clock Transfers
    - Report Minimum Pulse Width
    - Report TCCS
    - Report RSKM
    - Report DDR
    - Report SDC
    - Report Unconstrained Paths
    - Report Ignored Constraints
    - Report Datasheet
    - Check Timing
  - Custom Reports
    - Report Timing...
    - Report Net Timing...
    - Report Path...
    - Report Minimum Pulse Width...
    - Create Slack Histogram...
  - Macros
    - Report All Summaries
    - Report Top Failing Paths
    - Report All I/O Timings
    - Report All Core Timings

Welcome to the TimeQuest Timing Analyzer

The TimeQuest Timing Analyzer is a powerful ASIC-style timing analysis tool that validates the timing performance of all logic in the design using industry standard constraint, analysis, and reporting methodology. You can use the TimeQuest analyzer to constrain, run, and view results for all timing paths in your design. The TimeQuest Timing Analyzer offers the following graphical user interface features:

Report Pane Lists generated report panels.

Tasks Pane Lists common tasks you can perform. You can double-click a command to start a process in the flow.

View Pane Displays selected report panels. You can split the **View** pane into sub-panes by dragging the splitter control in the upper right corner of the pane.

Console Displays SDC and Tcl commands executed by the GUI or command-line.

```
Info: assumed to be external and are run using Tcl's "exec"
Info: command.
Info: - Type "exit" to exit.
Info: - Type "help" to view a list of Quartus II Tcl packages.
Info: - Type "help <package name>" to view a list of Tcl commands
Info:   available for the specified Quartus II Tcl package.
Info: - Type "help -tcl" to get an overview on Quartus II Tcl usages.
Info: ****************************************************************
tcl>
tcl>
```

For Help, press F1

For more information on analyzing memory interface timing in the Cyclone III and Stratix III devices, refer to *AN 438: Constraining and Analyzing Timing for External Memory Interfaces in Stratix III and Cyclone III Devices*.

Some of the timing failures that you may encounter are as follow:

- If read capture setup time is negative, you must decrease the delay chain used in the DQ pins by setting the **Input Delay from Pin to Input Register** to **0** or a lower number than the current setting in the **Assignment Editor**. However, if hold time is negative, you must do the opposite; increment the **Input Delay from Pin to Input Register** setting to a higher number than the current setting in the **Assignment Editor**.
- If you are experiencing any write and address/command timing failures, you can resolve them by setting:

```
set_instance_assignment -name CLOCK_TO_OUTPUT_DELAY 0 -to
<address/command/CK/CK# pin>
```

- If the resynchronization path does not meet the timing, you must move the resynchronization registers closer to the IOE. **Report DDR** shows information messages with the names of the source and destination registers of the worst case setup path. You must copy the name of the destination register from the message and move it as close as possible to the source register using the **Assignment Editor**. Similarly, if the postamble path is not meeting recovery timing, move the postamble registers closer to the IOE.
- Setup failures for transfers from the measure clock (`clk5`) to the PHY clock (`clk0` for half-rate interfaces or `clk1` for full-rate interfaces) must be ignored. Due to the dynamic nature of the measure clock, these must be treated as asynchronous transfers.

For more information on other issues that may cause timing failures in your design, refer to the latest version of the *Quartus II Release Notes*.

For an example of step-by-step flow for closing timing with ALTMEMPHY-based controllers, refer to *AN 328: Interfacing DDR2 SDRAM with Stratix II, Stratix II GX, and Arria GX Devices*.

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

Stratix II Design Example

To illustrate the flow described above, you must 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 devices 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

Table 1 indicates that 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 more information on instantiating the controllers, refer to the *DDR and DDR2 SDRAM High-Performance Controller User Guide*.

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

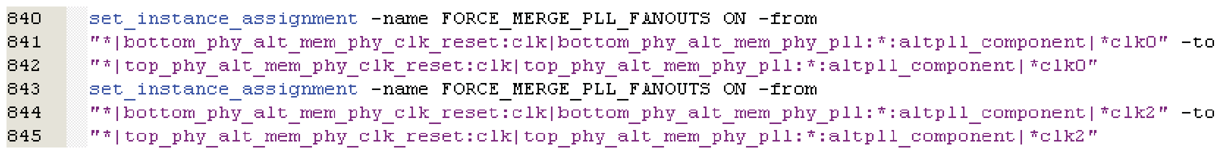
The DLL is shared between the two bottom controllers and the steps for sharing are in the "Sharing DLLs" section.

Sharing the PLL Clock Outputs

The design in this example is in half-rate mode. Refer to the "Sharing PLL Clock Outputs or Clock Networks" section for the Force Merge PLL assignments.

Figure 15 shows the snapshot of the assignment for Force Merge PLLs in **.qsf** file.

Figure 15. Assignments for Force Merge PLLs



```
set_instance_assignment -name FORCE_MERGE_PLL_FANOUTS ON -from
"*|bottom_phy_alt_mem_phy_clk_reset:clk|bottom_phy_alt_mem_phy_pll:*:altpll_component|*clk0" -to
"*|top_phy_alt_mem_phy_clk_reset:clk|top_phy_alt_mem_phy_pll:*:altpll_component|*clk0"
set_instance_assignment -name FORCE_MERGE_PLL_FANOUTS ON -from
"*|bottom_phy_alt_mem_phy_clk_reset:clk|bottom_phy_alt_mem_phy_pll:*:altpll_component|*clk2" -to
"*|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** and follow the instructions described on "Adding Constraints to the Design" on page 24 to add prefixes to match your actual top-level pin names for each controller (use prefixes `top`, `bot1`, and `bot2` for the design example).

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.

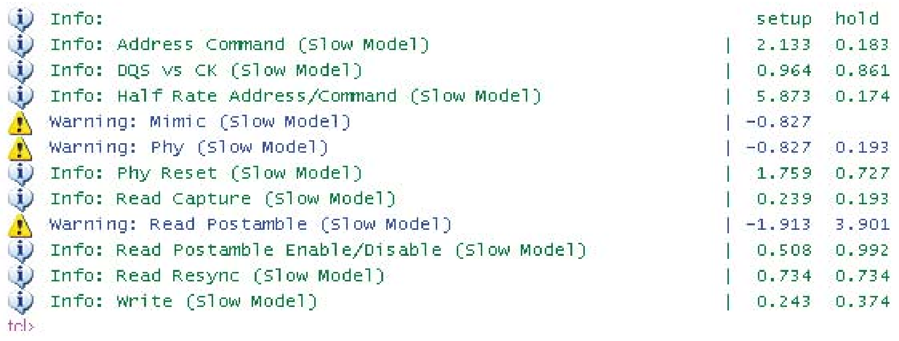
Compiling the Design Example to Verify Timing

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

Figure 16. Initial Timing Analysis Result for the Design Example

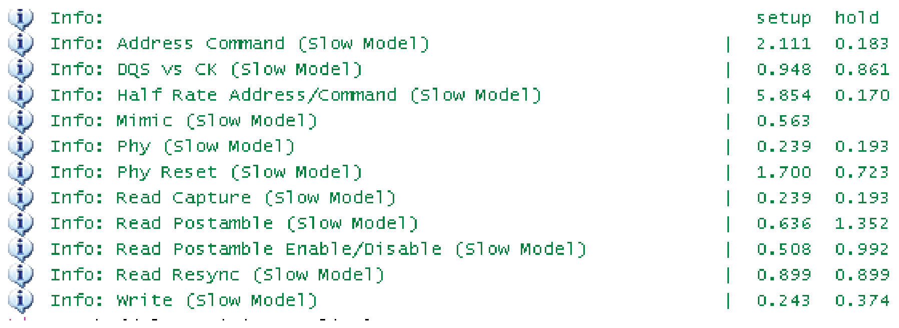


```
Info:                                                      setup   hold
Info: Address Command (Slow Model)                       |  2.133   0.183
Info: DQS vs CK (Slow Model)                             |  0.964   0.861
Info: Half Rate Address/Command (Slow Model)             |  5.873   0.174
Warning: Mimic (Slow Model)                              | -0.827
Warning: Phy (Slow Model)                                | -0.827   0.193
Info: Phy Reset (Slow Model)                             |  1.759   0.727
Info: Read Capture (Slow Model)                          |  0.239   0.193
Warning: Read Postamble (Slow Model)                     | -1.913   3.901
Info: Read Postamble Enable/Disable (Slow Model)         |  0.508   0.992
Info: Read Resync (Slow Model)                           |  0.734   0.734
Info: Write (Slow Model)                                 |  0.243   0.374
tcl>
```

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 17 shows the final timing analysis result for the design.

Figure 17. Final Timing Analysis Result for the Design Example



```
Info:                                                      setup   hold
Info: Address Command (Slow Model)                       |  2.111   0.183
Info: DQS vs CK (Slow Model)                             |  0.948   0.861
Info: Half Rate Address/Command (Slow Model)             |  5.854   0.170
Info: Mimic (Slow Model)                                 |  0.563
Info: Phy (Slow Model)                                   |  0.239   0.193
Info: Phy Reset (Slow Model)                             |  1.700   0.723
Info: Read Capture (Slow Model)                          |  0.239   0.193
Info: Read Postamble (Slow Model)                        |  0.636   1.352
Info: Read Postamble Enable/Disable (Slow Model)         |  0.508   0.992
Info: Read Resync (Slow Model)                           |  0.899   0.899
Info: Write (Slow Model)                                 |  0.243   0.374
```

Once 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 ALTMEMPHY megafunction allows Altera FPGAs to interface with high-performance external memory interfaces. The architecture of the device further allows the FPGAs to interface with multiple memory interfaces.

As shown in this application note, there are several ways to create multiple-controller design using the ALTMEMPHY megafunction data path. Pick one that is optimal for your system as both the ALTMEMPHY megafunction data path and the high-performance controller are modified to meet your needs.

Knowing the requirements and limitations for the multiple memory interface in the initial design allows you to architect your system better.

Document Revision History

Table 7 shows the revision history for this application note.

Table 7. Document Revision History (Part 1 of 2)

Date and Document Version	Changes Made	Summary of Changes
April 2009 v1.3	■ Updated the "Introduction" section. ■ Updated the Figure 1, Figure 2, Figure 3, Figure 4, Figure 5, Figure 7, Figure 8, Figure 15, Figure 16, and Figure 17. ■ Updated the Table 1, Table 2, Table 3, Table 4, and Table 5. ■ Updated the "Before Creating a Design in the Quartus II Software" section. ■ Updated the "Getting to Know Your Altera Device" section. ■ Updated the "HardCopy III, HardCopy IV, Stratix III, and Stratix IV Devices" section. ■ Updated the "Arria II GX Devices" section. ■ Updated the "Arria GX, HardCopy II, Stratix II, and Stratix II GX Devices" section. ■ Updated the "PLLs and Clock Network Resources" section. ■ Added the "Resources Planning for DLL and PLL Static Clocks" section. ■ Updated the "Creating PHY and Controller in a Quartus II Project" section. ■ Added the "SOPC Builder Flow" section. ■ Updated the "MegaWizard Plug-in Manager Flow" section. ■ Updated the "Sharing DLLs" section. ■ Updated the "Sharing PLL Clock Outputs or Clock Networks" section. ■ Updated the "Adding Constraints to the Design" section. ■ Updated the "Compiling the Design to Generate a Timing Report" section. ■ Updated the "Deciding DLL Requirements" section. ■ Updated the "Creating PHY and Controller for the Design Example" section. ■ Updated the "Sharing the DLLs Between the Two Bottom Controllers" section. ■ Updated the "Sharing the PLL Clock Outputs" section. ■ Updated the "Adding Constraints to the Design" section. ■ Updated the "Compiling the Design Example to Verify Timing" section.	—
May 2008 v1.2	■ Removed "Appendix A: Stratix III RTL Modification for DLL Sharing". ■ Updated the "Stratix III PLL Static Clock Sharing" section. ■ Updated the "Sharing PLL Clock Outputs or Clock Networks" section. ■ Removed original Figure 5. ■ Completed minor text edits.	—

Table 7. Document Revision History (Part 2 of 2)

Date and Document Version	Changes Made	Summary of Changes
October 2007 v1.1	■ Updated the sections “Introduction”, “Before Creating a Design in the Quartus II Software”, “PLLs and Clock Network Resources”, “Creating PHY and Controller in a Quartus II Project”, “Sharing DLLs”, “Sharing PLL Clock Outputs or Clock Networks”, “Adding Constraints to the Design”, “Compiling the Design to Generate a Timing Report”, “Creating PHY and Controller for the Example Design”, and “Sharing the PLL Clock Outputs” ■ Added the section “Stratix II, Stratix II GX, Arria GX, HardCopy II, and Cyclone III PLL Static Clock Sharing”, “Stratix II Design Example” (removed section Verifying Timing for the Design), “Appendix A: Stratix III RTL Modification for DLL Sharing”, and “Referenced Documents”. ■ Updated Figure 1, Figure 5, Figure 14, Figure 15, Figure 16, and Figure 17. ■ Added Figure 7, Figure 12. ■ Removed original Figure 4 and Figure 8. ■ Updated Table 2.	Added text, removed text, included new sections, section titles, updates to figures and tables, and added new figures.
June 2007 v1.0	Initial Release.	—