External Memory Interface Handbook Volume 6

Section I. ALTMEMPHY Design Tutorials

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

EMI_TUT_DDR-2.1

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

Subscribe

Contents

Chapter 5. Using DDR3 SDRAM Devices in Stratix III and Stratix IV Devices

Chapter 6. Using High-Performance DDR, DDR2, and DDR3 SDRAM with SOPC Builder

Chapter 7. Implementing Multiple ALTMEMPHY-Based Controllers

Additional Information

1. Using High-Performance Controller II with Native Interface Design

This tutorial shows how to use your existing Native interface design with the high-performance controller II (HPC II) architecture. The HPC II architecture only supports the Avalon® Memory-Mapped (Avalon-MM) interface, and requires an adaptor to work with designs using the Native interface.

The HPC II architecture offers significantly more efficient memory access with better flexibility. As HPC II only supports the Avalon-MM interface, all Native interface designs migrating to HPC II architecture must adapt to the Avalon-MM interface. The only significant difference between the two interfaces is the write data timing. Using Avalon-MM interface, the user logic presents a write request, address, burst count, and the first beat of write data at the same time. The subsequent write data beats are placed into the FIFO buffer until they are needed. In the Native interface, the user logic presents a write request, address, and burst count. The controller then requests the correct number of write data beats by asserting the write data request signal.

This tutorial shows how you can compile and simulate using the native_to_avalon_adaptor design example.

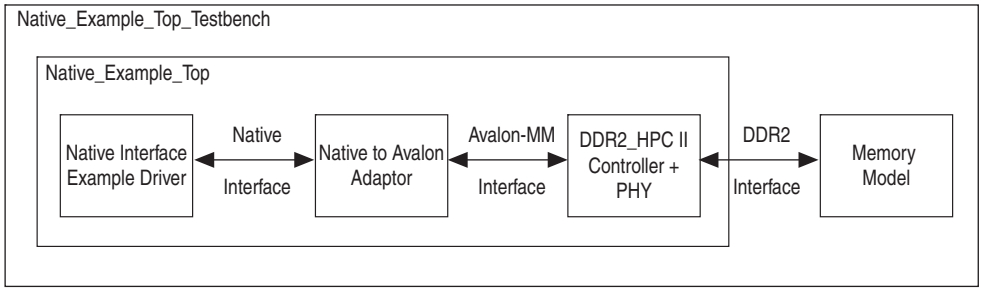
The design example implements a 200-MHz, 8-bit DDR2 SDRAM memory interface with a full-rate DDR2 SDRAM HPC II.

To download the design example, **native_to_avalon_adaptor.zip**, go to the External Memory Interface Design Examples page.

Functional Description

Figure 1–1 shows the different components of the design example and how they are connected.

Figure 1–1. Design Example Components



The native_to_avalon_adaptor module in the design example enables designs with Native interface to work with the Avalon-MM interface of HPC II. The module registers all its outputs except these signals: `ntv_ready`, `ntv_rdata`, `ntv_rdata_valid`, and `ntv_wdata_req`.

The native_to_avalon_adaptor module indicates it is ready to accept commands on the Native interface by asserting the `ntv_ready` signal. The `ntv_ready` signal is not registered, so the signal is asserted at the same cycle as the `avl_ready` signal.

The Avalon read command is the same as the Native read command; so the Native interface signals are simply registered, and connected to the Avalon interface signals. The Avalon read data and valid signals are passed to the Native interface directly without being registered.

The write command and data adaptation is slightly more complicated. The native_to_avalon_adaptor module generates data requests to the Native interface master, when it sees a valid write command request by the master. The `ntv_wdata_req` signal path is combinatorial, so the native_to_avalon_adaptor module is able to generate the data requests on the same cycle as the command requests. A state machine tracks the outstanding write data requested by the native_to_avalon_adaptor module on the Native interface.

The module has a maximum count of write data cycles that can be tracked, and deasserts the `ntv_ready` signal when the maximum count is reached. The module only handles Native interface commands with a burst of 1 or 2.

Performance

The adaptor has a command latency of one cycle for both read and write commands. Figure 1–2 on page 1–3 and Figure 1–3 on page 1–4 show how a read or write command is presented on the Avalon-MM interface on the following cycle after the command is presented on the Native interface.

Figure 1–2. Read Commands

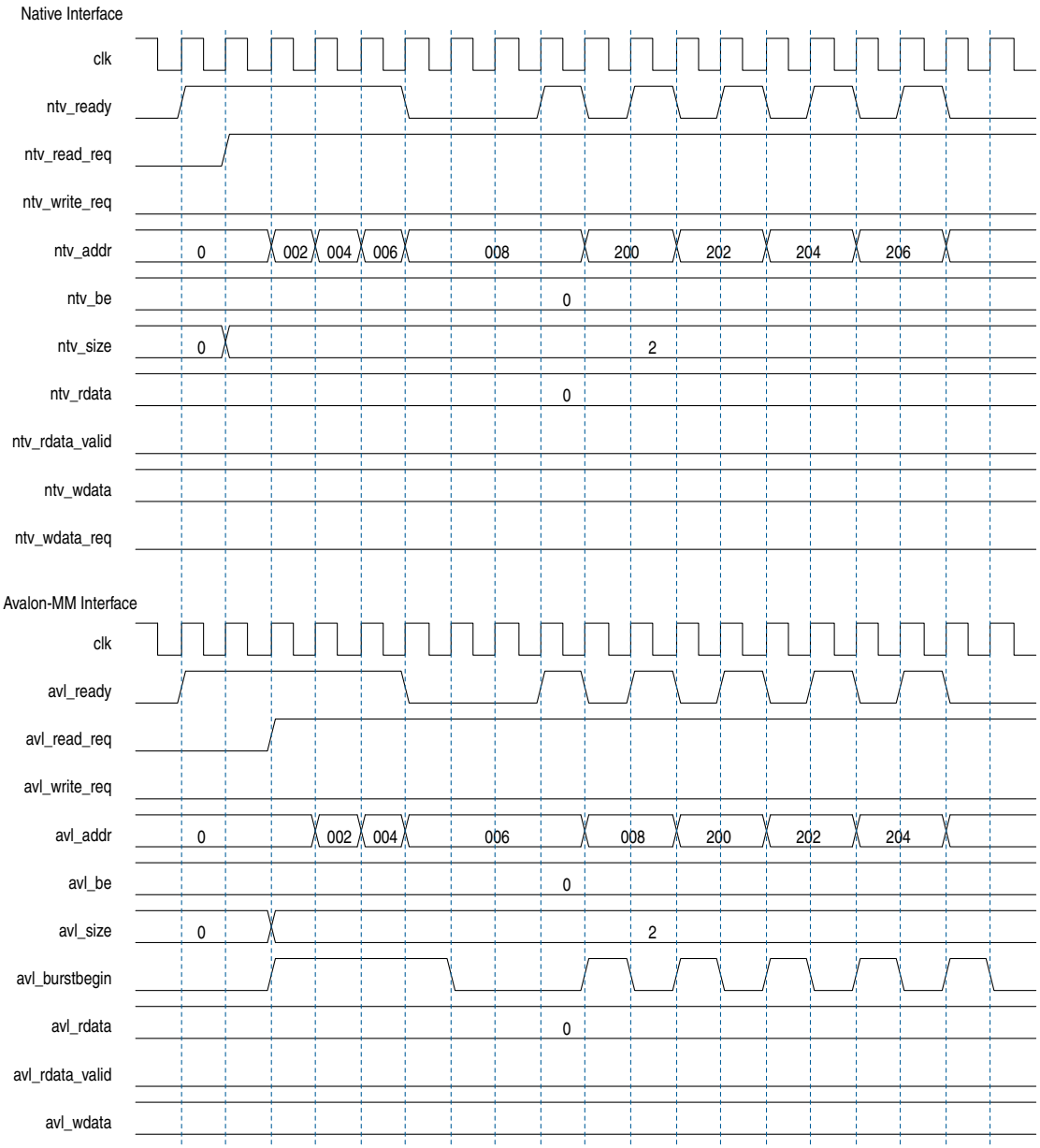
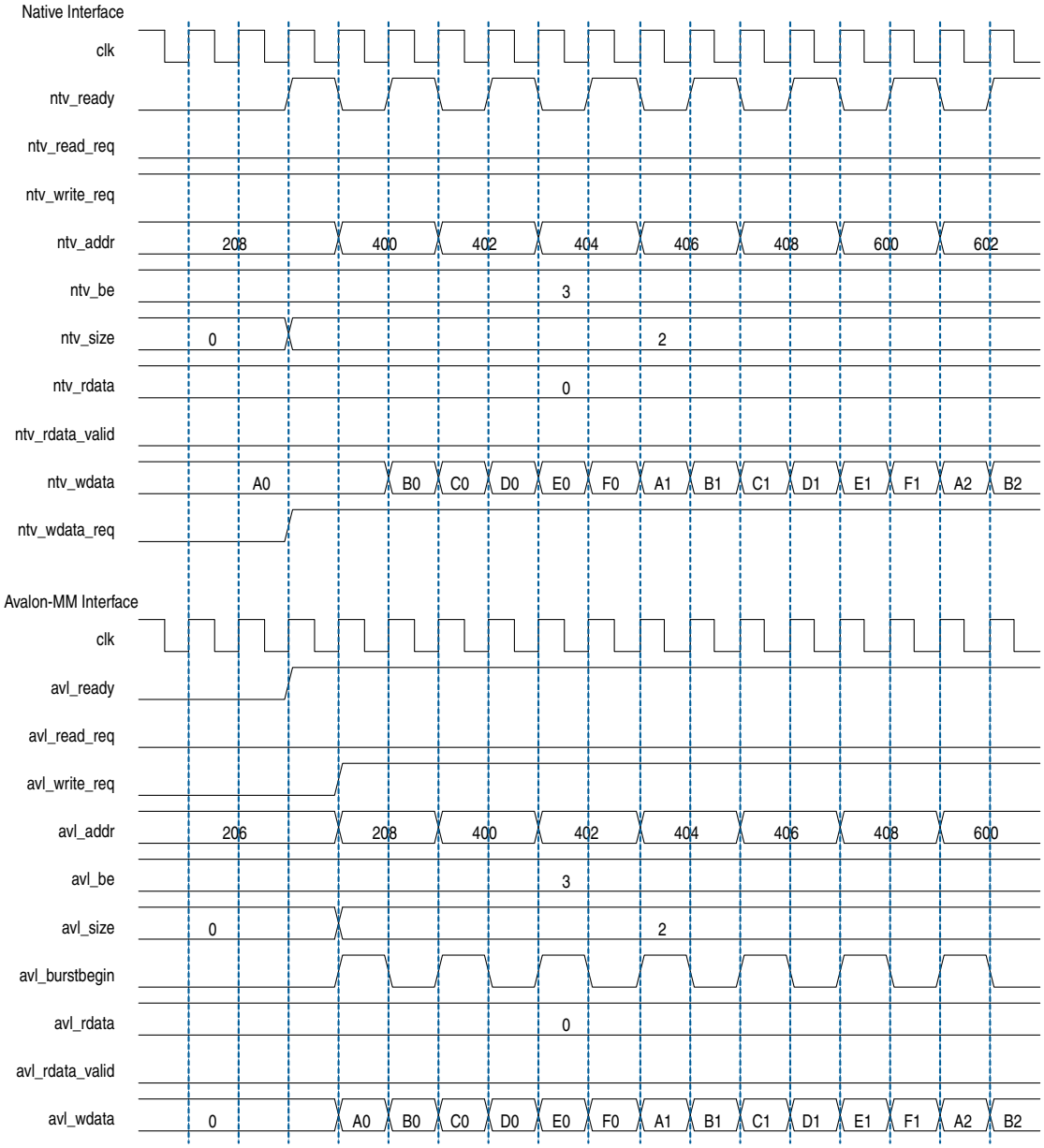
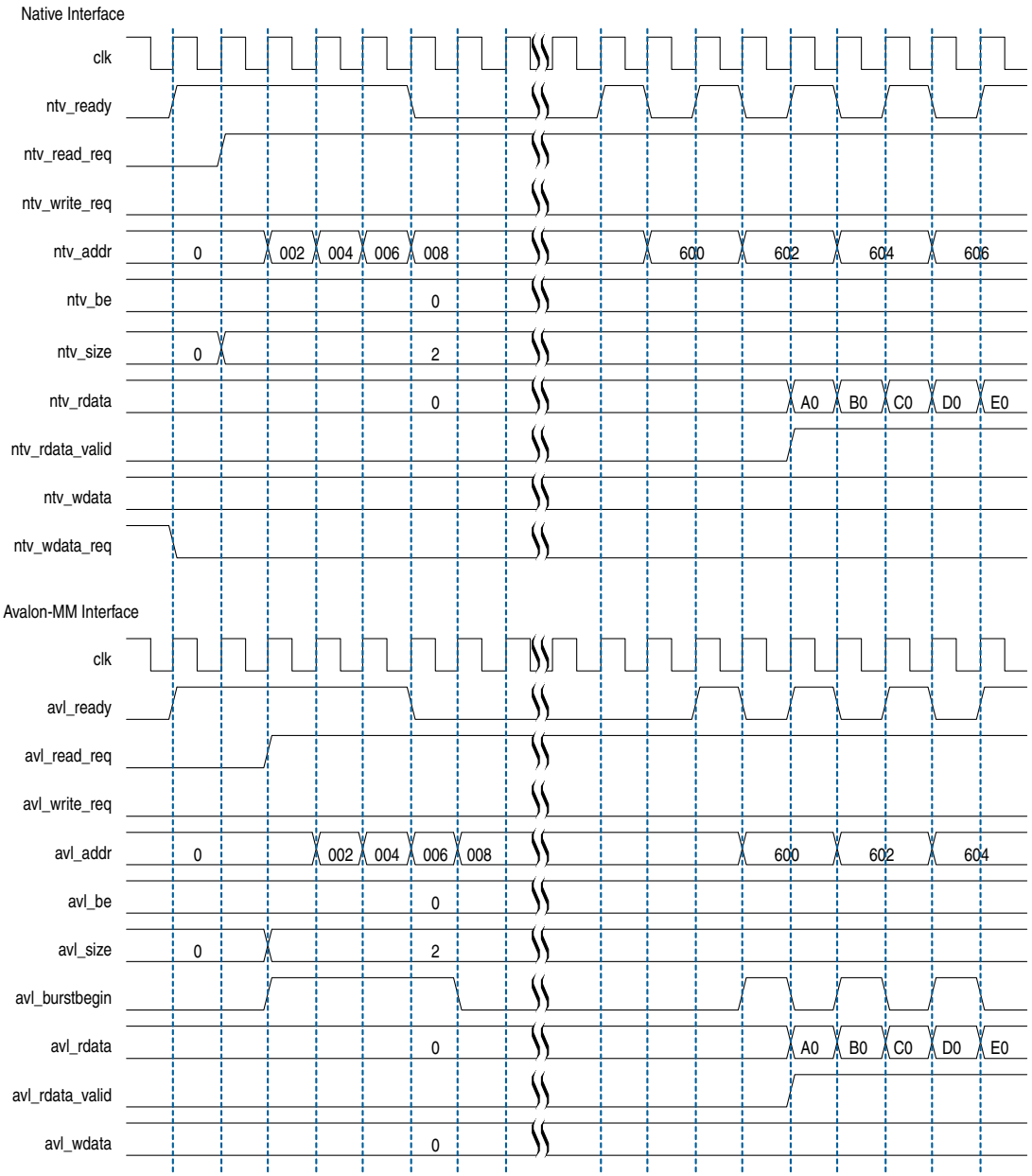


Figure 1–3. Write Commands



There are no latencies on the ntv_rdata and ntv_rdatavalid datapaths. Figure 1–4 shows that the data from a read command is presented on the Native interface, on the same cycle the data is presented on the Avalon-MM interface.

Figure 1–4. Read Data Returned



The adaptor asserts the ntv_wdata_req signal on the same cycle where the ntv_write_req signal is asserted. The Native interface master then presents the ntv_wdata signal on the subsequent cycle.

The adaptor uses a counter to keep track of outstanding write data beats that it needs to request on the Native interface. If the counter reaches a maximum value, the adaptor deasserts the ntv_ready signal to stop further commands on the Native interface. To increase the number of outstanding write data beats, increase the NTV_SIZE_COUNTER_WIDTH parameter to a sufficient value.

Parameters

Table 1–1 shows the parameters for the native_to_avalon_adaptor module.

Table 1–1. Module Parameters

Parameter	Default Setting	Description
AVL_ADDR_WIDTH	24	Set to match the HPCII local_address width.
AVL_BE_WIDTH	2	Set to match the HPCII local_be width.
AVL_LSIZE_WIDTH	2	Set to match the HPCII local_size width.
AVL_DATA_WIDTH	16	Set to match the HPCII local_wdata and local_rdata width.
NTV_SIZE_COUNTER-WIDTH	8	Controls the width of the counter that tracks outstanding write data beats on the Native interface.
NTV_STATE_WIDTH	3	Fixed value.

Signals

Table 1–2 through Table 1–4 show some of the signals for the native_to_avalon_adaptor design example.

Table 1–2. Clock and Reset Signals

Port	Direction	Description
clk	Input	Module clock
reset_n	Input	Module reset

Table 1–3. Native Interface Signals

Port	Direction
ntv_read_req	Input
ntv_write_req	Input
ntv_addr	Input
ntv_be	Input
ntv_size	Input
ntv_wdata	Input
ntv_ready	Output
ntv_rdata	Output
ntv_rdata_valid	Output
ntv_wdata_req	Output

Table 1–4. Avalon-MM Interface Signals

Port	Direction	Direction
avl_ready	Input	local_ready
avl_rdata	Input	local_rdata
avl_rdata-valid	Input	local_rdata_valid
avl_read_req	Output	local_read_req
avl_write_req	Output	local_write_req
avl_addr	Output	local_address
avl_be	Output	local_be
avl_size	Output	local_size
avl_wdata	Output	local_wdata
avl_burstbegin	Output	local_burstbegin

Getting Started

This section discusses the requirements and related procedures to compile and simulate the design example. This section contains the following topics:

- Software Requirements
- Directory Structure

- Compile the Design
- Simulate the Design

Software Requirements

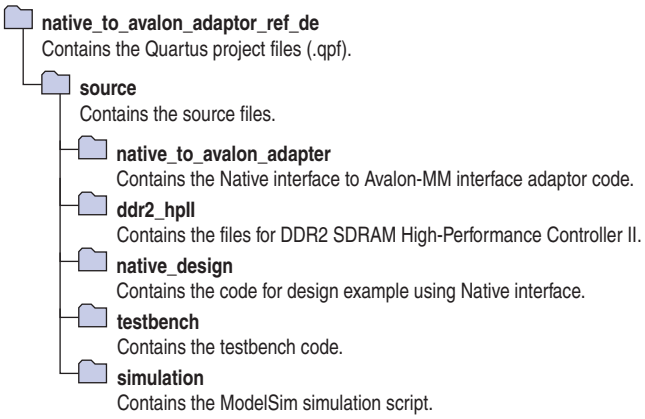
This design example requires the following software:

- Quartus® II software, version 9.1 or later
- ModelSim®-Altera® software, version 6.5b or later
- DDR2 Controller with ALTMEMPHY IP version 9.1

Directory Structure

Figure 1–5 shows the directory structure of the design example.

Figure 1–5. Directory Structure



Compile the Design

To compile the design example, perform the following steps:

1. Launch the Quartus II software.
2. On the File menu, click **Open Project**, navigate to *<your project path>***/native_to_avalon_adaptor_ref_de/native_to_avalon_adaptor_ref_de.qpf**, and click **Open**.
3. On the Processing menu, click **Start Compilation**.

Simulate the Design

To set up the simulation environment, and start the simulation, perform the following steps:

1. Launch **ModelSim-Altera**.
2. On the File menu, click **Change Directory**.
3. Select *<your project path>***/native_to_avalon_adaptor_ref_de/source/simulation/modelsim** and click **OK**.

4. On the Tools menu, click **Tcl**, then click **Execute Macro.** Select **native_example_top_run_msim_rtl_verilog.tcl**, and click **Open** to start the simulation.

Alternatively, you can run the script in the ModelSim transcript window by typing

```
source ./native_example_top_run_msim_rtl_verilog.tcl,
```

or in a terminal console by typing

```
vsim -do ./native_example_top_run_msim_rtl_verilog.tcl.
```

2. Using DDR, DDR2, and DDR3 SDRAM Devices in Arria II GX Devices

This tutorial describes how to use the design flow to design a 64-bit wide, 267-MHz, 533-Mbps DDR2 SDRAM interface, and a 16-bit wide, 300-MHz, 600-Mbps DDR3 SDRAM interface. The design examples provide some recommended settings, including termination scheme and drive strength settings, to simplify the design. You can also use the DDR2 SDRAM design example to design a DDR SDRAM interface.

The design examples target the Arria® II GX FPGA development kit, which includes a 64-bit wide 1-GB Micron MT8HTF12864HDY-800G1 400-MHz DDR2 SDRAM SODIMM, and a 16-bit wide 1-GB Micron MT41J64M16LA-15E DDR3 SDRAM component.

To download the design examples, **emi_ddr2_aii.zip** and **emi_ddr3_aii.zip**, go to the External Memory Interface Design Examples page.For more information about the design flow, refer to the *Recommended Design Flow* section in volume 1 of the *External Memory Interface Handbook*.

System Requirements

This tutorial requires the following hardware and software:

- DDR2 SDRAM interface:
  - Quartus II software version 9.1
  - DDR2 SDRAM Controller with ALTMEMPHY IP version 9.1
  - Arria II GX FPGA Development Kit with the EP2AGX125EF35C5 device
- DDR3 SDRAM interface:
  - Quartus II software version 9.1
  - DDR3 SDRAM Controller with ALTMEMPHY IP version 9.1
  - Arria II GX FPGA Development Kit with the EP2AGX260FF35C4 device

Create a Quartus II Project

Create a project in the Quartus II software that targets the respective device; EP2AGX260FF35C5 for the DDR2 SDRAM interface or EP2AGX260FF35C4 for the DDR3 SDRAM interface.

For step-by-step instructions on how to create a Quartus II project, refer to the Quartus II software tutorial by clicking the Help menu in the Quartus II window and selecting **PDF Tutorials**.

Instantiate and Parameterize a Controller

After creating a Quartus II project, you need to instantiate a controller and set its parameters.

Instantiate a Controller

To instantiate a controller, perform the following steps:

- DDR2 SDRAM controller
  a. Copy the memory parameters file, **ArriaIIGX_DDR2_Kit(MT8HTF12864HDY-800G1).xml**, to your <*installation directory*>**\91\ip\ddr2_high_perf\lib** directory.
  b. Launch the MegaWizard™ Plug-in Manager.
  c. In the MegaWizard Plug-In Manager, expand **External Memory** in the **Interfaces** folder and select the **DDR2 SDRAM High Performance Controller**.
  d. Enter `ddr2_sodimm` for the name of the DDR2 SDRAM high-performance controller.
- DDR3 SDRAM controller
  a. Copy the memory parameters file, **ArriaIIGX_DDR3_Kit(MT41J64M16LA-15E).xml**, to your <*installation directory*>**\91\ip\ddr3_high_perf\lib** directory.
  b. Launch the MegaWizard Plug-in Manager.
  c. In the MegaWizard Plug-In Manager, expand **External Memory** in the **Interfaces** folder and select the **DDR3 SDRAM High Performance Controller.**
  d. Enter `ddr3` for the name of the DDR3 SDRAM high-performance controller.

Both the DDR2 and DDR3 design examples instantiate the ALTMEMPHY megafunction automatically.

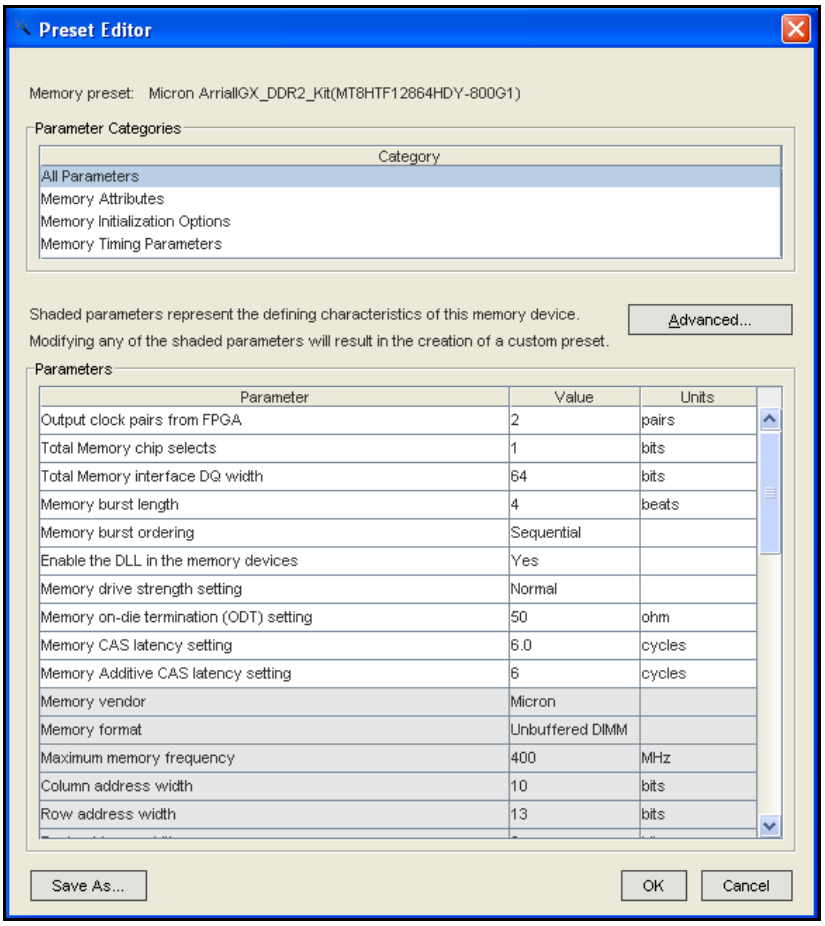
Parameterize DDR2 SDRAM

To parameterize the DDR2 high-performance controller to interface with a 267-MHz 64-bit wide DDR2 SDRAM interface, perform the following steps:

1. In the **Memory Setting** tab, set **Speed grade** to 5.
2. For **PLL reference clock frequency**, enter **100** MHz. **The input clock source,** `clock_source`, supplies the PLL **reference clock frequency.**
3. For **Memory clock frequency**, enter **267** MHz. This value is the maximum frequency supported for the DDR2 SDRAM interface on Arria II GX devices.

4. For **Memory Presets**, select **Micron ArriaIIGX_DDR2_Kit(MT8HTF12864HDY-800G1)**, which gives a 64-bit wide 1-GB 400-MHz DDR2 SODIMM.To create or modify a memory preset, click **Modify parameters**. In the **Preset Editor** dialog box, modify the memory presets. Refer to Figure 2–1 on page 2–3.

Figure 2–1. Modify the Memory Presets to Create a Custom Memory



5. Turn on the **Enable Memory Chip Calibration in Timing Analysis** option in the **Advanced** page. This option is required for Arria II GX devices, which need post-processing script to remove timing model pessimism.
6. In the **PHY Settings** tab, under **Advanced PHY Settings**, turn on the U**se differential DQS** option to enhance signal to noise ratio. Turn on this option if noise margin is a concern.

7. By default, **Clock Phase** is set to **90** for **Address/Command Clock Settings**. The **Auto-Calibration Simulation Options** parameter is for RTL simulation only and is not applicable for gate-level simulation. ALTMEMPHY does not support gate-level timing simulation.In the **Board Settings** tab, enter the following values for the parameters under **Slew Rates** and **Board Skews**:
   - **CK/CK# slew rate (Differential)** = **2.475 V/ns**
   - **Addr/Command slew rate** = **0.859 V/ns**
   - **DQS/DQS# slew rate (Differential)** = **1.386 V/ns**
   - **DQ slew rate** = **2.665 V/ns**

   
   These slew rates are obtained from a simulation using the default I/O standard and drive options.

   - Max skew within DQS group = 0.0339 ns
   - Max skew between DQS groups = 0.0824 ns
   - Addr/Command to CK skew = 0.0356 ns

   The Intersymbol Interference parameters are not applicable for single rank configurations. Set all these parameters to **0 ns**.
8. In the **Controller Settings** tab, for **Controller Architecture**, select **High Performance Controller II** for higher efficiency and advanced features.
9. Under **Efficiency**, select the specified values for the following options:
   a. For **Command Queue Look-Ahead Depth**, select **6**.
   b. For **Local-to-Memory Address Mapping**, select **CHIP-ROW-BANK-COL**.
   c. For **Local Maximum Burst Count**, select **8**.
10. Click **Next**.
11. Turn on the **Generate simulation model** option to generate the simulation model for the RTL simulation.
12. Click **Finish** to generate your MegaCore function variation. The MegaWizard Plug-In Manager generates all the files necessary for your DDR2 SDRAM controller and a top-level design, which you use to test or verify the board operation.


For more information about the DDR/DDR2 SDRAM high-performance controller, refer to the *DDR and DDR2 SDRAM Controller with ALTMEMPHY IP User Guide* in volume 3 of the *External Memory Interface Handbook*.

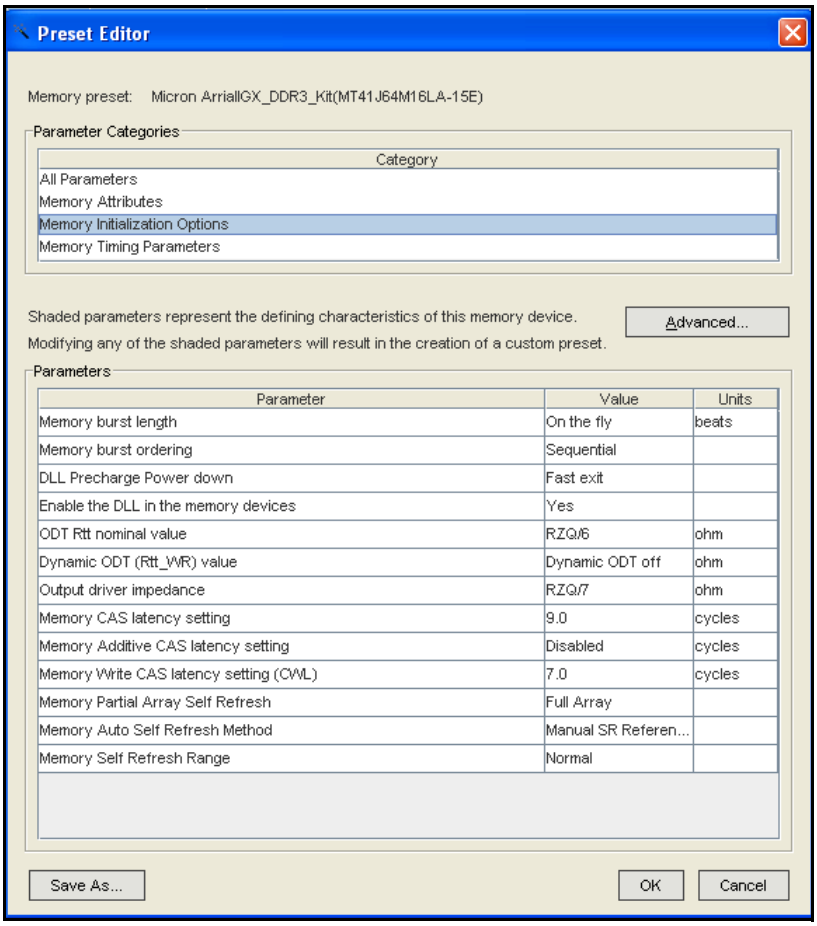
Parameterize DDR3 SDRAM

To parameterize the DDR3 high-performance controller to interface with a 300-MHz, 16-bit wide, DDR3 SDRAM interface, perform the following steps:

1. In the **Memory Setting** tab, set **Speed grade** to 4.
2. For **PLL reference clock frequency**, enter **100** MHz, to match the on-board oscillator**.**

3. For **Memory clock frequency**, enter **300** MHz. This is the maximum frequency supported for DDR3 SDRAM interface on Arria II GX devices.
4. For **Memory Presets**, select **Micron ArriaIIGX_DDR3_Kit(MT41J64M16LA-15E)**, which gives a 16-bit wide 1-GB 667-MHz DDR3 component. To create or modify a memory preset, click **Modify parameters**. In the **Preset Editor** dialog box, modify the memory presets (Figure 2–2 on page 2–5).

Figure 2–2. Modify the Memory Presets to Create a Custom Memory



5. In **Advanced** page, turn on the **Enable Memory Chip Calibration in Timing Analysis** option. This option is required for Arria II GX devices, which need post-processing script to remove timing model pessimism.
6. Under **Address/Command Clock Settings**, use the default value of **Clock Phase,** which is **90**.
7. In the **Board Settings** tab, set the following **Slew Rates** and **Board Skews** parameters to the specified values:
   - **CK/CK# slew rate (Differential) = 4 V/ns**
   - **Addr/Command slew rate = 2 V/ns**
   - **DQS/DQS# slew rate (Differential) = 1.8 V/ns**
   - **DQ slew rate = 1.5 V/ns**

These slew rates are obtained from simulation using the default I/O standard and drive options.

- **Max skew within DQS group = 0.020 ns**
- **Max skew between DQS groups = 0.020 ns**
- **Addr/Command to CK skew = -0.045 ns**

The **Intersymbol Interference** parameters are not applicable for single rank configurations. Set all these parameters to **0 ns**.

8. In the **Controller Settings** table, for **Controller Architecture**, select **High Performance Controller II** for higher efficiency and advanced features.Under **Efficiency**, select the specified values for the following options:
   a. For **Command Queue Look-Ahead Depth**, select **6**.
   b. For **Local-to-Memory Address Mapping**, select **CHIP-ROW-BANK-COL**.
   c. For **Local Maximum Burst Count**, select **4**.
9. Click **Next**.
10. Turn on the **Generate simulation model** option to generate the simulation model for the RTL simulation.
11. Click **Finish** to generate your MegaCore function variation. The MegaWizard Plug-In Manager generates all the files necessary for your DDR3 SDRAM controller and generates an example top-level design, which you use to test or verify board operation.

For more information about the DDR3 SDRAM high-performance controller, refer to the *DDR3 SDRAM Controller with ALTMEMPHY IP User Guide* in volume 3 of the *External Memory Interface Handbook*.

Add Constraints

After instantiating the DDR2 or DDR3 SDRAM high-performance controller, the ALTMEMPHY megafunction generates the constraint files for the design example. You must apply these constraints to the design before compilation.

Set the Top-Level Entity

The top-level entity of the project must be set to the correct entity. The naming convention for an ALTMEMPHY megafunction entity is <*variation_name*>**_phy.v** or **vhd**; an SDRAM high-performance controller entity is <*variation_name*>**.v** or **vhd**.

To set the top-level file, perform the following steps:

1. Open the entity file, **ddr2_sodimm_example_top.v** or **vhd** for the DDR2 design example or **ddr3_example_top.v** or **vhd** for the DDR3 design example.
2. On the Project menu, click **Set as Top-Level Entity**.

Set the Optimization Technique

To ensure the remaining unconstrained paths are routed with the highest speed and efficiency, perform the following steps to set the optimization technique:

1. On the Assignments menu, click **Settings**.
2. Select **Analysis & Synthesis Settings**.
3. Select **Speed** under **Optimization Technique**. Click **OK**.

Set the Fitter Effort

To set the fitter effort, perform the following steps:

1. On the Assignments menu, click **Settings**.
2. Select **Fitter Settings**.
3. Turn on **Optimize hold timing** and select **All Paths**.
4. Turn on **Optimize multi-corner timing**.
5. Under **Fitter effort**, select **Standard Fit (highest effort)**.
6. Click **OK**.

Add Timing Constraints

When you instantiate an SDRAM high-performance controller, it generates the timing constraints file <*variation_name*>**_phy_ddr_timing.sdc**, or <*variation_name*>**_phy_ddr3_timing.sdc**. The timing constraint file constrains the clock and input and output delay on the SDRAM high-performance controller.

To add the timing constraints, perform the following steps:

1. On the Assignments menu, click **Settings**.
2. In the **Category** list, expand **Timing Analysis Settings** and select **TimeQuest Timing Analyzer**.
3. Select the <*variation_name*>**_phy_ddr_timing.sdc** file for DDR2, or <*variation_name*>**_phy_ddr3_timing.sdc** file for DDR3 and click **Add**.
4. Click **OK**.

Add Pin and DQ Group Assignments

The pin assignment script, <*variation_name*>**_pin_assignments.tcl**, sets up the I/O standards for the memory interface.

This script does not create a clock for the design. You must create a clock for the design and provide pin assignments for the signals of both the example driver and testbench that the MegaCore variation generates.

To add the pin and I/O standards to the design example, perform the following steps:

1) On the Tools menu, click **Tcl scripts**.

2) Under **Libraries**, select <*variation_name*>**_pin_assignments.tcl**

3) Click **Run**.

Enter Pin Location Assignments

To enter the pin location assignments using the Pin Planner, perform the following steps:

1. Run analysis and synthesis. On the Processing menu, point to **Start** and click **Start Analysis & Synthesis**.
2. Assign all of your pins, so that the Quartus II software fits your design correctly and gives correct timing analysis. Manually assign the pin locations by using the Pin Planner or Assignment Editor. To assign the pin locations for the Arria II GX Development Kit, run the Altera-provided **ArriaIIGX_DDR2_PinLocations.tcl** script for the DDR2 SDRAM interface design or the **ArriaIIGX_DDR3_PinLocations.tcl** script for the DDR3 SDRAM interface design.

   If you are at the design exploration phase of your design cycle and do not have any PCB defined pin locations, you must still manually define an initial set of pin constraints, which can become more specific during your development process.

To manually assign pin locations using Pin Planner, perform the following steps:

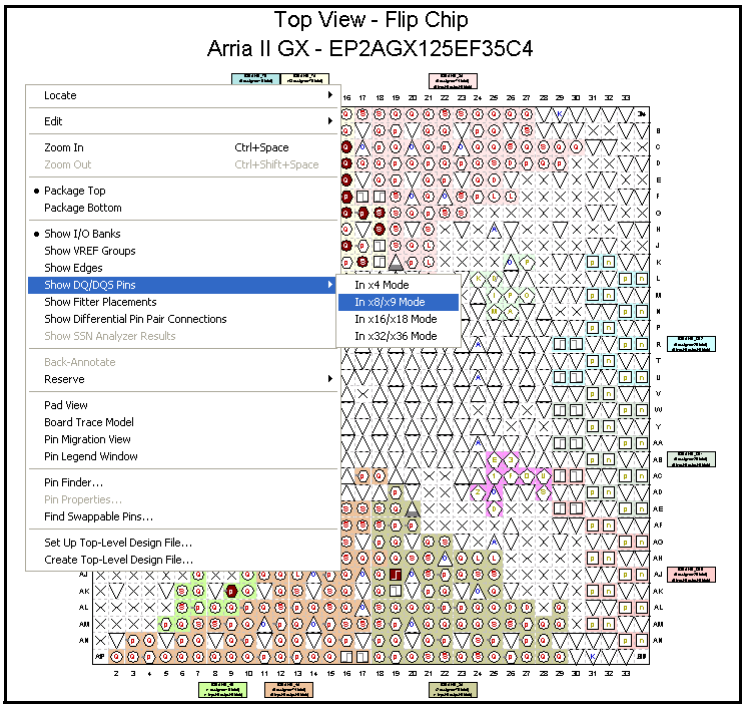
1. On the Assignments menu, click **Pin Planner**.
2. Assign DQ and DQS pins.
   a. To select the device DQS pin groups that the design uses, assign each DQS pin in your design to the required DQS pin in the Pin Planner. The Quartus II Fitter then automatically places the respective DQ signals onto suitable DQ pins within each group. To see DQS groups in Pin Planner, right click, select **Show DQ/DQS Pins**, and click **In ×8/×9 Mode**. Pin Planner shows each DQS group in a different color and with a different legend: S = DQS pin, Sbar = DQSn pin and Q = DQ pin (refer to Figure 2–3).

      Most memory devices operate in ×8/×9 mode; however, as some memory devices operate in ×4 mode, refer to your specific memory device datasheet.

   b. Select the DQ mode to match the DQ group width (number of DQ pins/number of DQS pins) of your memory device. DQ mode is not related to the memory interface width.

DQ group order and DQ pin order within each group is not important. However, you must place the DQ pins in the same group as their respective strobe pin.

Figure 2–3. Quartus II Pin Planner, Show DQ/DQS Pins in ×8/×9 Mode



3. Place the DM pins within their respective DQ group.
4. Place address and control command pins on any spare I/O pins, ideally in the same bank or side of the device as the mem_clk pins.
5. Ensure that you place mem_clk pins on differential I/O pairs for the CK/CK# pin pair. To identify the differential I/O pairs, right-click in the Pin Planner and select **Show Differential Pin Pair Connections**. The pin pairs show a red line between each pin pair.
6. Ensure that the mem_clk pins use any regular adjacent I/O pins; ideally the differential I/O pairs for the CK/CK# pin pair. To identify the differential I/O pairs, right-click in Pin Planner and select **Show Differential Pin Pair Connections**. Pin pairs show a red line between each pin pair.
7. Place the clock_source pin on a dedicated PLL clock input pin with a direct connection to the SDRAM controller PLL and DLL pair—usually on the same side of the device as your memory interface. This recommendation reduces PLL jitter, saves a global clock resource, and eases timing and fitter effort.
8. Place the global_reset_n pin (as any high fan-out signal) on a dedicated clock pin.
9. Ensure that you place the R_{UP} and R_{DN} pins, termination_blk0~_rdn_pad and termination_blk0~_rup_pad, at locations within the same V_{CCIO} voltage bank.

For more information about how to use the Quartus II Pin Planner, refer to the *I/O Management* chapter in volume 2 of the *Quartus II Handbook*.

Assign I/O Standards

To assign the I/O standards, perform the following steps:

1. On the Assignments menu, click **Assignment Editor**.
2. Specify **LVDS** as the I/O standard for `clock_source`.
3. Specify **1.8** V for DDR2 SDRAM, or **1.5** V for DDR3 SDRAM as the I/O standard for `global_reset_n` and `mem_odt[0]`.

Assign Virtual Pins

The example top-level design, which is auto-generated by the high-performance controller, includes an example driver to stimulate the interface. This example driver is not part of the SDRAM high-performance controller IP, but allows easy testing of the IP.

The example driver outputs several test signals to indicate its operation and the status of the stimulated memory interface. These signals are `pnf`, `pnf_per_byte`, `test_complete`, and `test_status`. These signals are not part of the memory interface, but are for testing. You must either take these signals to a debug header or set the signals to virtual pins using the Quartus II Assignment Editor. When using the example driver for testing, do not remove these signals from the top-level signal list; otherwise, the Quartus II software optimizes the driver away and the example driver fails.

To assign virtual pin assignments for the Arria II GX development board, run the Altera-provided **ArriaIIGX_DDR2_exdriver_vpin.tcl** file for the DDR2 SDRAM design example or **ArriaIIGX_DDR3_exdriver_vpin.tcl** file for DDR3 SDRAM design example, or manually assign the virtual pin assignments using the Assignment Editor.

Enter Board Trace Delay Models

For accurate I/O timing analysis, the Quartus II software must be aware of the board trace and loading information. This information must be derived and refined during your PCB development process of pre-layout (line) simulation through post-layout (board) simulation.

For external memory interfaces that use memory modules (DIMMs), the board trace and loading information must also include the trace and loading information of the module, which you can obtain from your memory vendor.

To enter board trace information, perform the following steps:

1. In the Pin Planner, select the pin or group of pins that you want to enter the information for.

2. Right-click and select **Board Trace Model, as shown in** Figure 2–4.

Figure 2–4. Board Trace Model

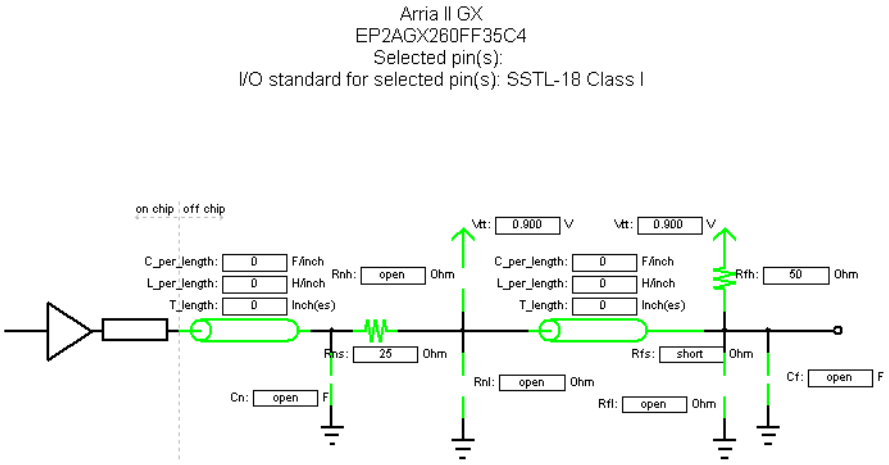


Table 2–1 shows the board trace model parameters for the Arria II GX development board.

Table 2–1. Arria II GX Development Board Trace Model Summary for DDR2 and DDR3 (Part 1 of 2)

Net	Near (FPGA End of Line)						Far (Memory End of Line)				
	Length (Inch)	L_per_length (nH/In)	C_per_length (pF/In)	Cn (pF)	Rns	Rnh	Length (Inch)	L_per_length (nH/In)	C_per_length (pF/In)	Cf (pF)	Rfh/R (pF)
DDR2 SODIMM											
Addr *(1)*	1.743	8	4.04	—	3	56	2.230	10.6	3.06	16	—
CLK0	2.614	8.66	3.74	—	—	—	1.017	10.6	3.06	8	100
CLK1	1.616	8.66	3.74	—	—	—	1.020	10.6	3.06	8	100
CKE/CS#	1.599	8	4.04	—	3	56	2.155	10.6	3.06	2	
DQS0	1.866	8	4.04	—	3 for dq, 22 for dqs	56	0.637	10.6	3.06	4	—
DQS1	1.757	8	4.04	—	3 for dq, 22 for dqs	56	0.661	10.6	3.06	4	—
DQS2	1.437	8	4.04	—	3 for dq, 22 for dqs	56	0.741	10.6	3.06	4	—
DQS3	1.457	8	4.04	—	3 for dq, 22 for dqs	56	0.761	10.6	3.06	4	—
DQS4	1.450	8	4.04	—	3 for dq, 22 for dqs	56	0.749	10.6	3.06	4	—

Table 2–1. Arria II GX Development Board Trace Model Summary for DDR2 and DDR3 (Part 2 of 2)

Net	Near (FPGA End of Line)						Far (Memory End of Line)				
	Length (Inch)	L_per_length (nH/In)	C_per_length (pF/In)	Cn (pF)	Rns	Rnh	Length (Inch)	L_per_length (nH/In)	C_per_length (pF/In)	Cf (pF)	Rfh/R (pF)
DQS5	1.548	8	4.04	—	3 for dq, 22 for dqs	56	0.667	10.6	3.06	4	—
DQS6	1.677	8	4.04	—	3 for dq, 22 for dqs	56	0.665	10.6	3.06	4	—
DQS7	1.831	8	4.04	—	3 for dq, 22 for dqs	56	0.666	10.6	3.06	4	—
DDR3 *(2)*											
Addr *(1)*	1.879	8.66	3.74	—	—	—	—	—	—	1.3	56
CLK	1.359	8.66	3.74	—	—	—	—	—	—	1.4	100
CKE/CS#	1.775	8.66	3.74	—	—	—	—	—	—	1.3	56
DQS0	1.278	8.66	3.74	—	—	—	—	—	—	2.5	56
DQS1	1.249	8.66	3.74	—	—	—	—	—	—	2.5	56

Note to Table 2–1:

(1) Addr = Addr, ba, we#, ras#, odt, and cas.

(2) The near (FPGA end of line) board trace model summary is not applicable for memory components configuration.

Altera recommends that you use the **Board Trace Model** assignment for all memory interface signals. To apply the board trace model assignments for the Arria II GX FPGA development kit, run the Altera-provided **ArriaIIGX_DDR2_BTModels.tcl** script for the DDR2 SDRAM interface design, or the **ArriaIIGX_DDR3_BTModels.tcl** script for the DDR3 SDRAM interface design, or manually assign virtual pin assignments using the Quartus II Pin Planner.

Perform RTL or Functional Simulation (Optional)

After instantiating the DDR2 or DDR3 SDRAM high-performance controller, the Quartus II software generates a design example that includes driver, test bench, and memory model that allows you to perform functional simulation on your design.

The Verilog HDL or VHDL simulation model of the PHY, *<variation_name>***_alt_mem_phy_sequencer_wrapper.vo/.vho** file, is located in your project directory.

To run the simulation with NativeLink, perform the following steps:

1. Set the absolute path to your third-party simulator executable file, by performing the following steps:
   a. On the Assignments menu, point to **EDA Tool Settings**.
   b. In the **Category** list, expand **EDA Tool Settings** and click **Simulation**.
   c. Under **Tool name**, select **ModelSim**.
   d. Under **NativeLink settings**, select **Compile test bench** and click **Test Benches**.
   e. Click **New**.
   f. Type the name of your testbench top-level module and simulation period.
   g. Click **OK**.
2. To elaborate your design, on the Processing menu, point to **Start** and click **Start Analysis & Elaboration**.
3. On the Tools menu, point to the **Run EDA Simulation Tool** and click **EDA RTL Simulation**. This step creates the **\simulation** directory in your project directory and a script that compiles all necessary files and runs the simulation.

Compile Design and Verify Timing

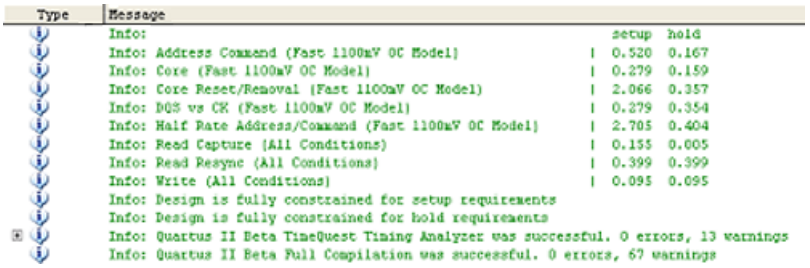
To compile the design, on the Processing menu, click **Start Compilation**. After successfully compiling the design, the Quartus II software automatically runs the verify timing script, which produces a timing report for the design together with the compilation report.

The report timing script provides information about the following margins and paths:

- Address and command setup and hold margin
- Core setup and hold margin
- Core reset, removal setup, and hold margin
- Read capture setup and hold margin
- Write setup and hold margin
- Read resync setup and hold margin
- DQS vs CK setup and hold margin

Figure 2–5 shows the timing margin report for a DDR2 SDRAM interface design in the message window in the Quartus II software.

Figure 2–5. Timing Margin Report in the Quartus II Software

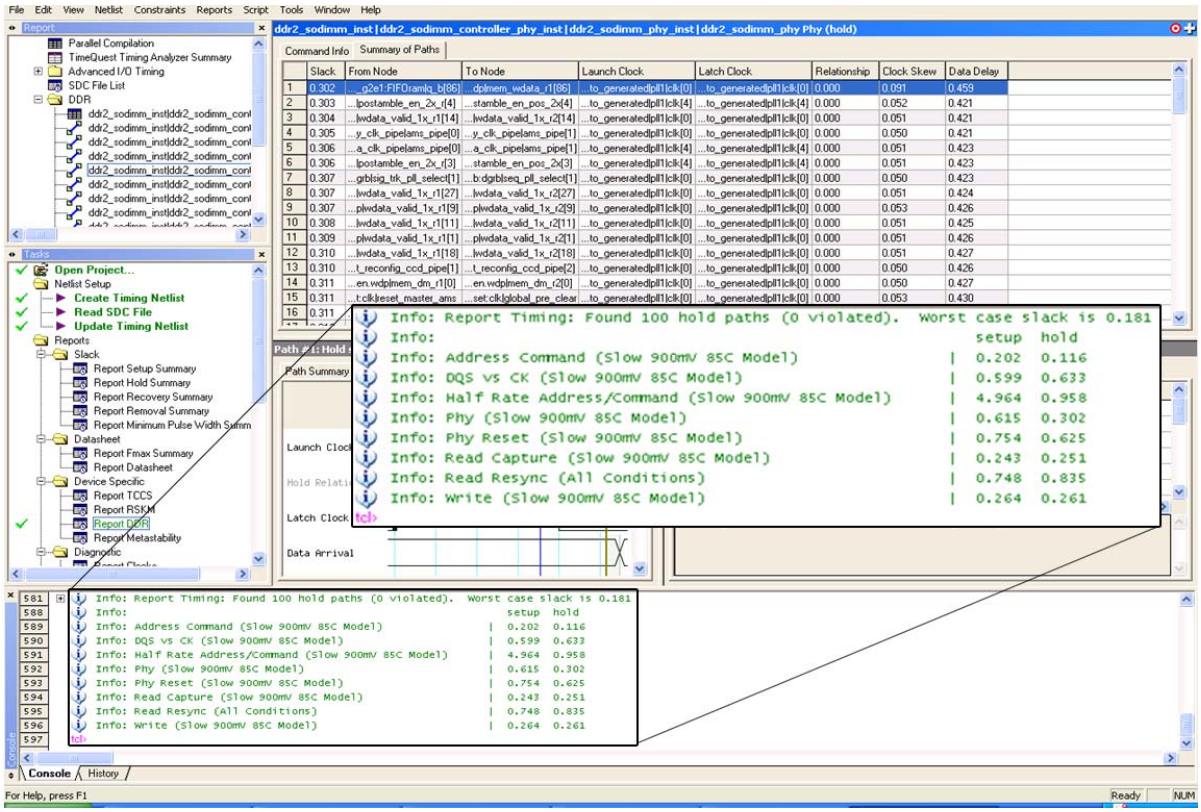


You can also obtain the timing report by running the report timing script (**ddr2_sodimm_phy_report_timing.tcl** or **ddr3_phy_report_timing.tcl)** in the TimeQuest timing analyzer. To obtain the timing report in the TimeQuest Timing Analyzer window, perform the following steps:

1. In the Quartus II software, on the Tools menu, click **TimeQuest Timing Analyzer**.
2. On the **Tasks** pane, double-click **Report DDR** to run **Update Timing Netlist**, **Create Timing Netlist**, and **Read SDC File**. This command subsequently executes the report timing script to generate the timing margin report.

Figure 2–6 shows the timing margin report in the TimeQuest Timing Analyzer window after the report timing script runs. The results are the same as the Quartus II software results.

Figure 2–6. Timing Margin Report in the TimeQuest Timing Analyzer



You must verify the timing on every corner of the timing model. You must run the timing report script with all available timing models—slow 0°C, slow 85°C, and fast 0°C—to ensure positive margins across process, voltage, and temperature. To analyze timing on a different corner, first double-click **Set Operating Conditions** in the left pane. Select a new timing corner, then go to the Script menu and rerun the <*variation name*>**_report_timing.tcl** script.

For designs that target Arria II GX devices, you must run a post-processing script to remove timing model pessimism on the write and read capture path margins. For example, refer to Figure 2–7 and Figure 2–8.

Figure 2–7. Write Margin Summary

	Operation	Setup Slack	Hold Slack
1	⊟ Standard Write	0.249	0.246
2	Spatial correlation pessimism removal	0.015	0.015
3	Write	0.264	0.261

Figure 2–8. Read Capture Path Margin Summary

	Operation	Setup Slack	Hold Slack
1	⊟ Standard Read Capture	0.172	0.181
2	Spatial correlation pessimism removal	0.071	0.070
3	Read Capture	0.243	0.251

The standard write and read capture margins are initially calculated using the FPGA timing model and adjusted to account for the effects not modeled by either the timing model or TimeQuest timing analyzer. These effects include memory calibration, deskew topologies, quantization error and calibration uncertainties.

The final write and read capture margins are summarized in the Report DDR margin summary. Figure 2–9 shows an example of a Report DDR margin summary.

Figure 2–9. Report DDR Margin Summary

	Path	Operating Condition	Setup Slack	Hold Slack
1	Address Command (Slow 900mV 85C Model)	Slow 900mV 85C Model	0.202	0.116
2	DQS vs CK (Slow 900mV 85C Model)	Slow 900mV 85C Model	0.599	0.633
3	Half Rate Address/Command (Slow 900mV 85C Model)	Slow 900mV 85C Model	4.964	0.958
4	Phy (Slow 900mV 85C Model)	Slow 900mV 85C Model	0.615	0.302
5	Phy Reset (Slow 900mV 85C Model)	Slow 900mV 85C Model	0.754	0.625
6	Read Capture (Slow 900mV 85C Model)	Slow 900mV 85C Model	0.243	0.251
7	Read Resync (All Conditions)	All Conditions	0.748	0.835
8	Write (Slow 900mV 85C Model)	Slow 900mV 85C Model	0.264	0.261

For more information about the TimeQuest Timing Analyzer window, refer to *The Quartus II TimeQuest Timing Analyzer* chapter in volume 7 of the *Quartus II Handbook*. For information about timing analysis, refer to the *Timing Analysis* section in volume 4 of the *External Memory Interface Handbook*.

Verify Design on a Board

The SignalTap® II Embedded Logic Analyzer shows read and write activity in the system.

For more information about using the SignalTap II Embedded Logic Analyzer, refer to the *Design Debugging Using the SignalTap II Embedded Logic Analyzer* chapter in the *Quartus II Handbook*, *AN 323: Using SignalTap II Embedded Logic Analyzers in SOPC Builder Systems* and *AN 446: Debugging Nios II Systems with the SignalTap II Embedded Logic Analyzer*.

To add the SignalTap II Embedded Logic Analyzer, perform the following steps:

1. On the Tools menu, click **SignalTap II Logic Analyzer**.
2. Under **Signal Configuration**, next to the **Clock** box click **...** (**Browse Node Finder**).
3. In the **Named** box, type `*phy_clk`.
4. For **Filter**, select **SignalTap II: pre-synthesis** and click **List**.
5. In **Nodes Found**, for a DDR2 SDRAM design, select **ddr2_sodimm_example_top | ddr2_sodimm:ddr2_sodimm_inst | ddr2_sodimm_controller_phy:ddr2_sodimm_controller_phy_inst | phy_clk | phy_clk**, or for a DDR3 SDRAM design, select **ddr3_example_top | ddr3:ddr3_inst | ddr3_controller_phy:ddr3_controller_phy_inst | phy_clk | phy_clk**, and click **>** to add the signal to **Selected Nodes**.
6. Click **OK**.
7. Under **Signal Configuration**, specify the following settings:
   - For **Sample depth**, select **512**
   - For **RAM type**, select **Auto**
   - For **Trigger flow control**, select **Sequential**
   - For **Trigger position**, select **Center trigger position**
   - For **Trigger conditions**, select **1**
8. On the Edit menu, click **Add Nodes**.
9. In the **Named** box, search for specific nodes by typing `*local*`.
10. For **Filter**, select **SignalTap II: pre-synthesis** and click **List**.

11. Select the following nodes in **Nodes Found** and click **>** to add to **Selected Nodes**:
    - **local_address**
    - **local_rdata**
    - **local_rdata_valid**
    - **local_read_req**
    - **local_ready**
    - **local_wdata**
    - **local_wdata_req**
    - **local_write_req**
    - **pnf**
    - **pnf_per_byte**
    - **test_complete** (trigger)

    Do not add any DDR2 SDRAM or DDR3 SDRAM interface signals to the SignalTap II Embedded Logic Analyzer. The load on these signals increases and adversely affects the timing analysis.

12. Click **OK**.
13. To reduce the SignalTap II logic size, turn off **Trigger Enable** on the following bus signals:
    - **local_address**
    - **local_rdata**
    - **local_wdata**
    - **pnf_per_byte**
14. Right-click **Trigger Conditions** for the `test_complete` signal and select **Rising Edge**.
15. On the File menu, click **Save**.

    If you see the message **Do you want to enable SignalTap II file "stp1.stp" for the current project**, click **Yes**.

Compile the Project

After you add signals to the SignalTap II Embedded Logic Analyzer, to recompile your design, on the Processing menu, click **Start Compilation**.

Verify Timing

After the design compiles, ensure that TimeQuest timing analysis passes successfully. In addition to this FPGA timing analysis, check your PCB or system SDRAM timing. To perform timing analysis, run the *<variation name>*_**phy_report_timing.tcl** script.

1. On the Tools menu, click **Tcl Scripts**.

2. Select <*variation name*>**_phy_report_timing.tcl**.
3. Click **Run**.

Download the Object File

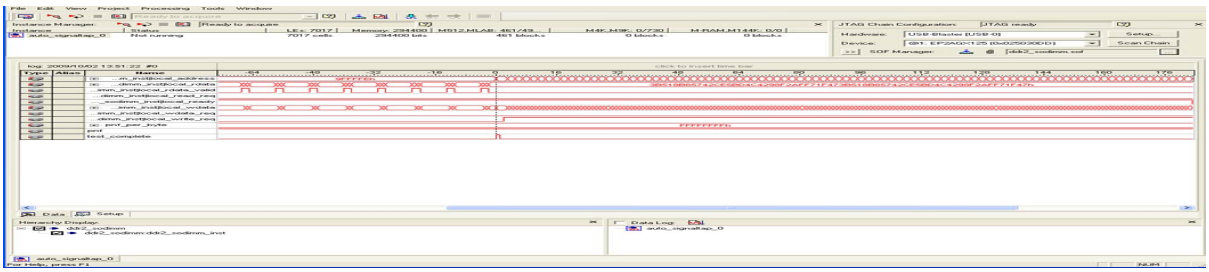
To download the object file to the device, perform the following steps:

1. Connect the development board to your computer.
2. On the Tools menu, click **SignalTap II Logic Analyzer**. The SignalTap II dialog box appears.
3. Click **...** to open the **Select Program Files** dialog box.
4. Select <*your project name*>**.sof**.
5. Click **Open**.
6. To download the file, click the **Program Device** button.

Test the Design Example in Hardware

When the design example including SignalTap II Embedded Logic Analyzer successfully downloads to your development board, click **Run Analysis** to run once, or click **Autorun Analysis** to run continuously. Figure 2–10 shows the design analysis.

Figure 2–10. SignalTap II Example DDR2 SDRAM Design Analysis



3. Using DDR and DDR2 SDRAM Devices in Cyclone III and Cyclone IV Devices

This tutorial shows how to use the design flow described in the preceding sections to design a 32-bit wide 167-MHz DDR2 SDRAM memory interface targeted for the Cyclone® III FPGA development kit. The design example also provides some recommended settings, including termination scheme and drive strength settings, to simplify your design flow. Although the design example is specifically for the DDR2 SDRAM memory interface and Cyclone III devices, the design flow is identical to that for a DDR SDRAM memory interface or using Cyclone IV devices.

This design example targets a memory interface frequency of 167 MHz because the targeted development kit uses an EP3C120F780 device. This device is available in –7 and –8 speed grades only. You can achieve a higher clock rate, up to 200 MHz, for DDR2 SDRAM if you select a –6 speed grade device from the Cyclone III family. This design example uses the DDR2 SDRAM Controller with ALTMEMPHY IP and is compiled in the Quartus II software version 9.0.

To download the design example, **emi_ddr2_ciii.zip**, go to the External Memory Interface Design Examples page.

Select the Device

Cyclone III and Cyclone IV devices support various data widths for DDR2 and DDR SDRAM memory interfaces. This walkthrough uses the Cyclone III EP3C120F780C7 device with 32-bit wide DDR2 SDRAM interface up to 167 MHz on the bottom I/O banks. The 32-bit wide interface uses four ×8 groups. For the DDR2 SDRAM memory device, select Micron's 512-MB MT47H32M16CC-3 333-MHz DDR2 SDRAM device, because it is the device used on the development board.

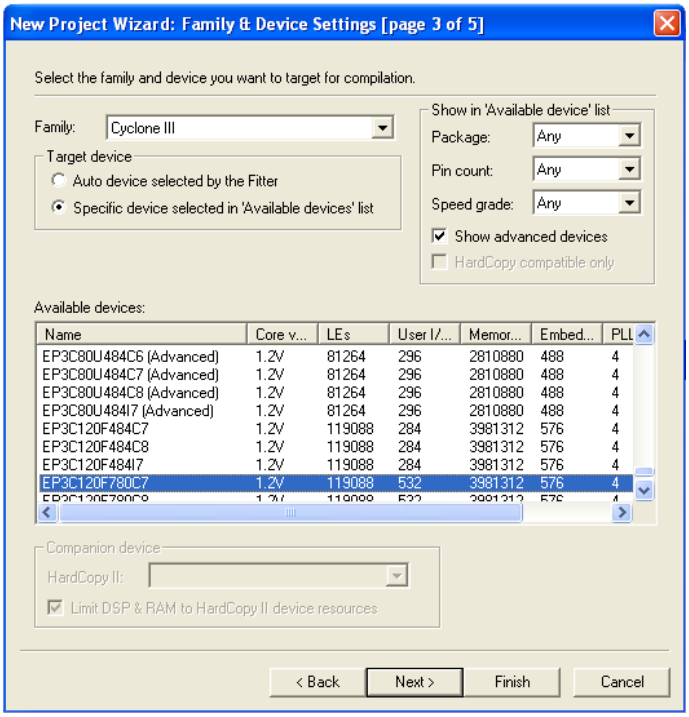
Top/bottom DQ groups provide the fastest performance. The Quartus II software automatically uses the top/bottom DQ groups if they are available. Combining top/bottom DQ groups with left/right DQ groups for a single interface is not recommended and may result in a degraded performance.‘

For more information about the DQ/DQS bus groups for different densities, packages, and sides of the Cyclone III or Cyclone IV device, refer to the *External Memory Interfaces in the Cyclone III Device Family* chapter in volume 1 of the *Cyclone III Device Handbook*, or *External Memory Interfaces in the Cyclone IV Devices* chapter in volume 1 of the *Cyclone IV Device Handbook*.

Instantiate PHY and Controller in a Quartus II Project

Create a project in the Quartus II software targeting the EP3C120F780C7 device. Figure 3–1 shows how to target this device in the **New Project Wizard**.

Figure 3–1. Creating a Quartus II Project Targeting the EP3C120F780C7 Device



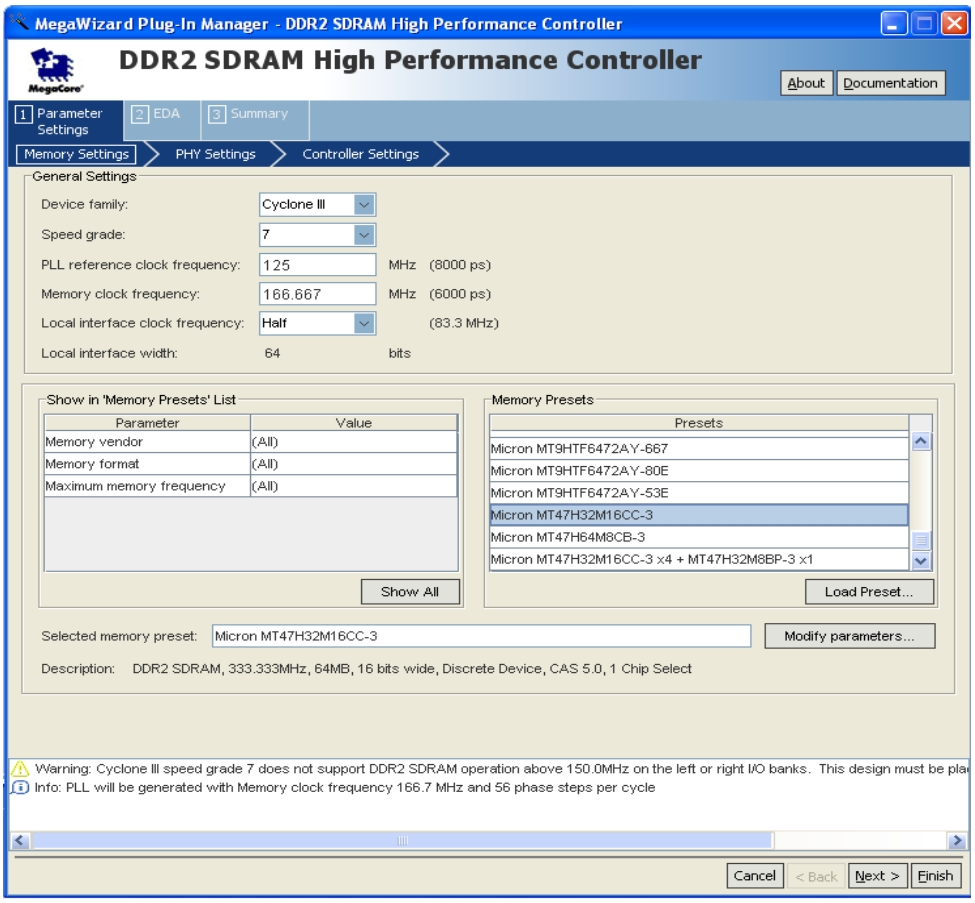
For detailed step-by-step instructions about how to create a Quartus II project, refer to the Quartus II software tutorial by clicking the Help menu in the Quartus II window and selecting **PDF Tutorials**.

After creating a Quartus II project, instantiate the DDR2 SDRAM Controller with ALTMEMPHY IP. This example uses the DDR2 SDRAM high-performance controller, which instantiates the ALTMEMPHY megafunction automatically. Select the **DDR2 SDRAM Controller with ALTMEMPHY IP** in the **Interfaces** section of the MegaWizard™ Plug-In Manager. Name the controller `DDR2`.

The subsequent files mentioned in this document that are generated by the MegaWizard Plug-In Manager and also other project files will have **DDR2** as the prefix for the file names.

The rest of this subsection specifies the memory settings. Select the Cyclone III device with –7 speed grade. Then set the PLL reference clock frequency to 125 MHz and the memory clock frequency to 166.667 MHz. The 125 MHz PLL reference clock is provided by the on-board oscillator. Select the Micron **MT47H32M16CC-3** 333-MHz device. This 512-MB DDR2 device has a 16-bit data width. Figure 3–2 shows the memory settings panel after you make the selections.

Figure 3–2. Configuring the DDR2 SDRAM Controller with ALTMEMPHY IP for the DDR2 Memory Interface



The DDR2 SDRAM controller MegaWizard interface uses the default parameters for the memory when you instantiate the controller. The default parameters are based on the memory datasheets. You can customize your memory presets by modifying the parameters. Click the **Modify parameters** button to modify the memory attributes, memory initialization options, or memory timing parameters in the Preset Editor dialog box. In this design example, modify the memory interface DQ width to 32 to match the targeted memory width on the development kit. Figure 3–3 shows the Preset Editor dialog box after you perform the customization.

Figure 3–3. Editing the Memory Presets to Create a Custom Memory

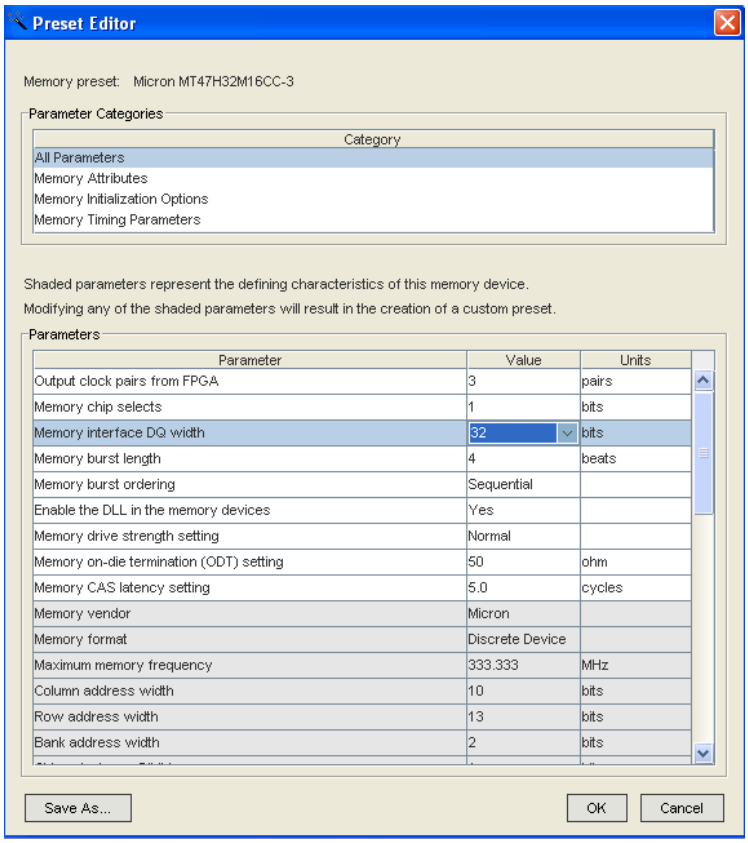
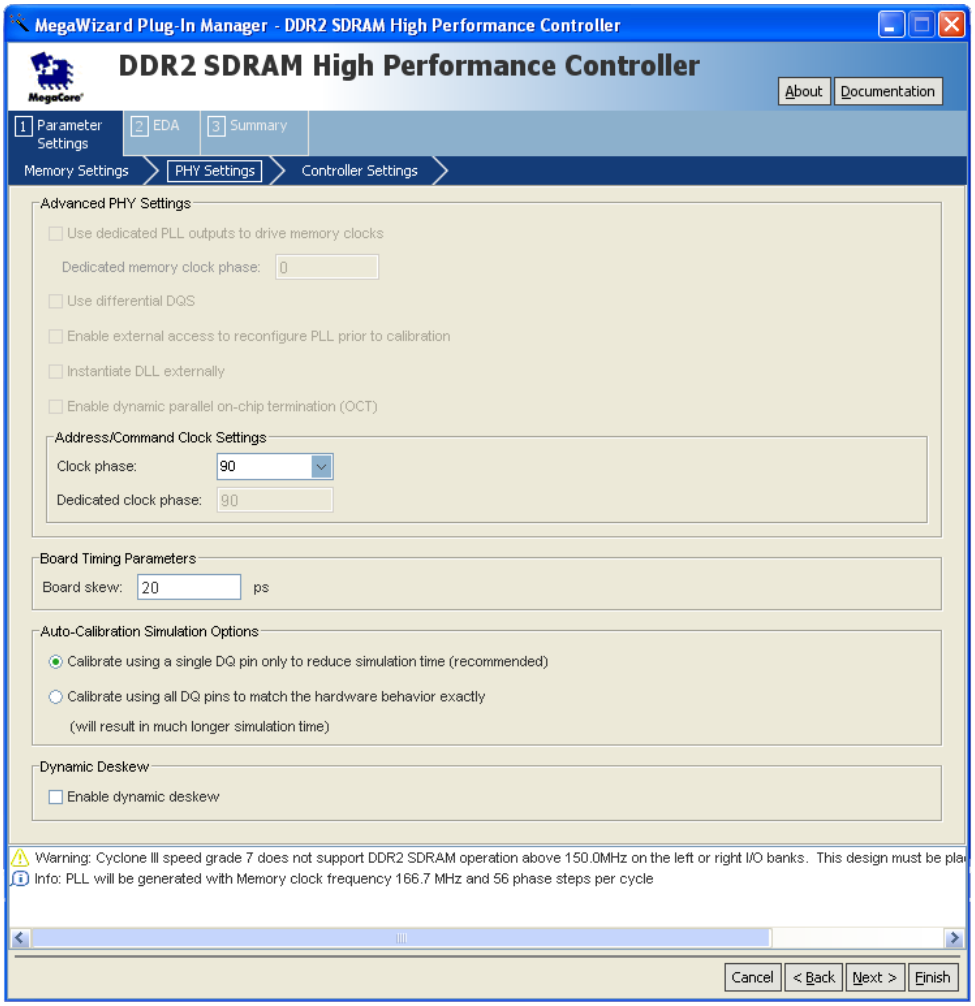


Figure 3–4 shows the PHY Settings tab. In the PHY Settings tab, under **Board Timing Parameters**, you must parameterize the **Board Skew** settings. The specified skew is across all memory interface signal types including data, strobe, clock, address and command, and is used to generate the PHY timing constraints for all paths. The default value is set to 20 ps. You must update this number based on your board specification, because this number is used to calculate the overall system timing margin.

The DDR2 SDRAM device uses the `CK` and `CK#` signals to clock the command and address signals into the memory. The controller names the `CK` and `CK#` signals `mem_clk` and `mem_clk_n`, respectively. The skew between the `CK` or `CK#` and the DDR2 SDRAM-generated `DQS` signal is the value of t_{DQSCK} in the DDR2 SDRAM data sheet.

The DDR2 SDRAM has a write requirement (t_{DQSS}) that the positive edge of the DQS signal on writes be within ± 25% (± 90°) of the positive edge of the DDR2 SDRAM clock input. t_{DQSS} is defined as the time between the DQS latching edge to its associated clock edge. The controller generates the mem_clk and mem_clk_n signals using the DDR registers in the input/output element (IOE) to match the DQS signal and reduce any variations across process, voltage, and temperature. The positive edge of the DDR2 SDRAM clock, mem_clk, is aligned with the DQS write to satisfy t_{DQSS}.

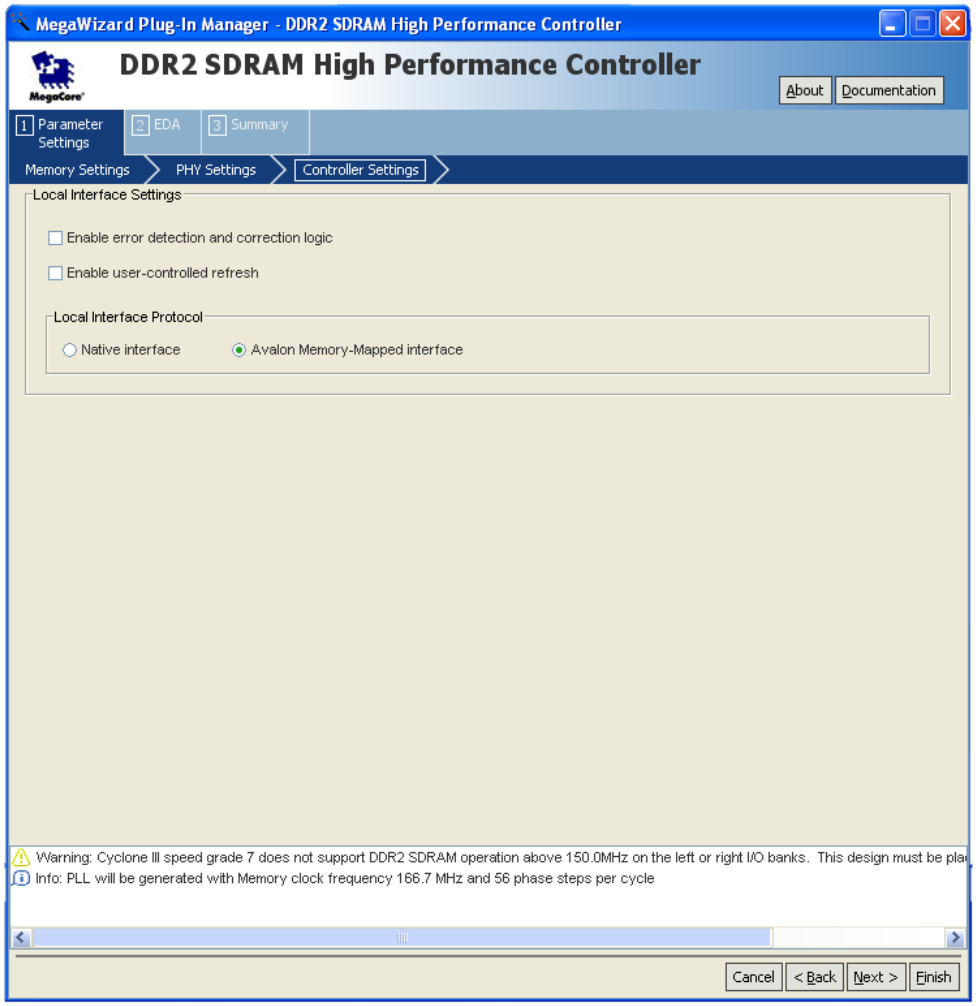
Figure 3–4. DDR2 SDRAM Controller with ALTMEMPHY IP PHY Settings



The settings in the **Auto-Calibration Simulation Options** section are for RTL simulation only and are not applicable for gate-level simulation.

Figure 3–5 shows the **Controller Settings** panel.

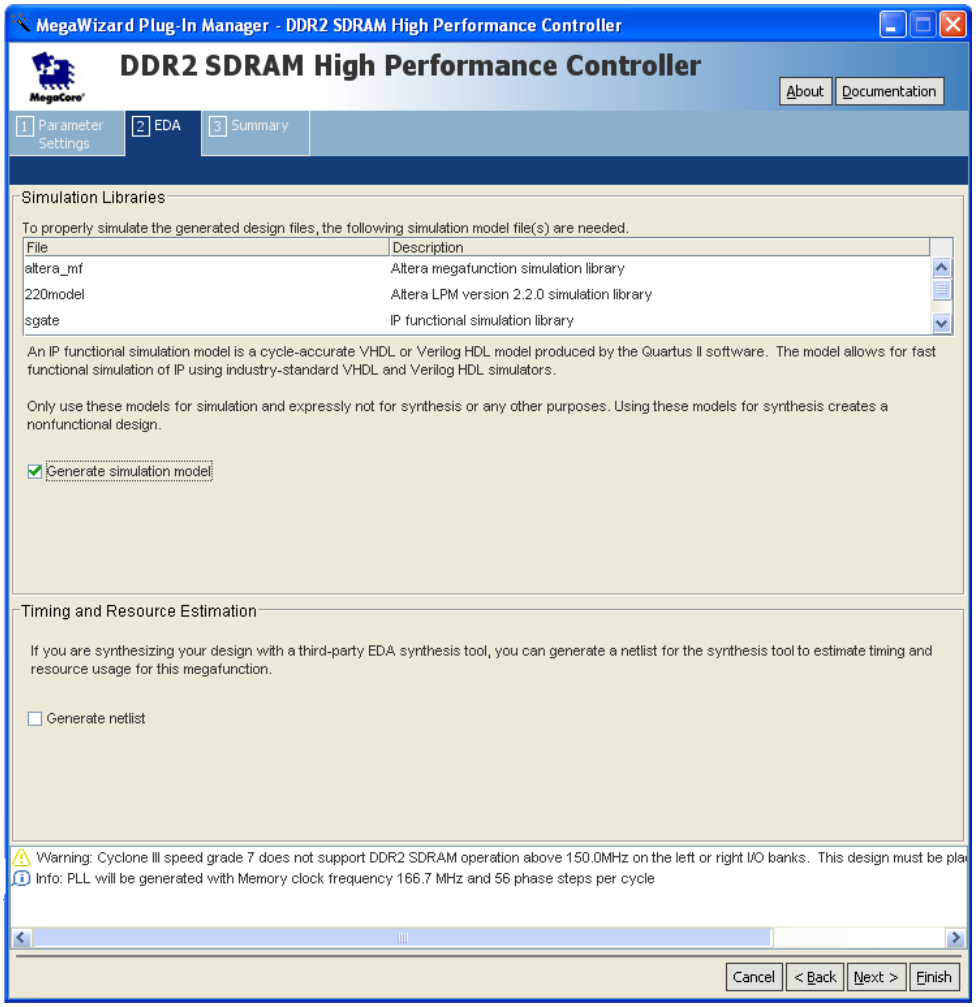
Figure 3–5. DDR2 SDRAM Controller with ALTMEMPHY IP Settings



Choose your local interface setting in the **Controller Settings** panel. Turn on the **Enable error detection and correction logic** option if you want to use error code correction (ECC). If you have your own refresh requirements, turn on **Enable user-controlled refresh**. Next, select the **Local Interface Protocol** for the memory interface. The default interface is the **Avalon Memory-Mapped Interface**. This interface allows you to easily connect to other Avalon-MM peripherals.

Figure 3–6 on page 3–7 shows the **EDA** panel. The MegaWizard can generate the simulation model for simulating the memory controller in either Verilog HDL or VHDL.

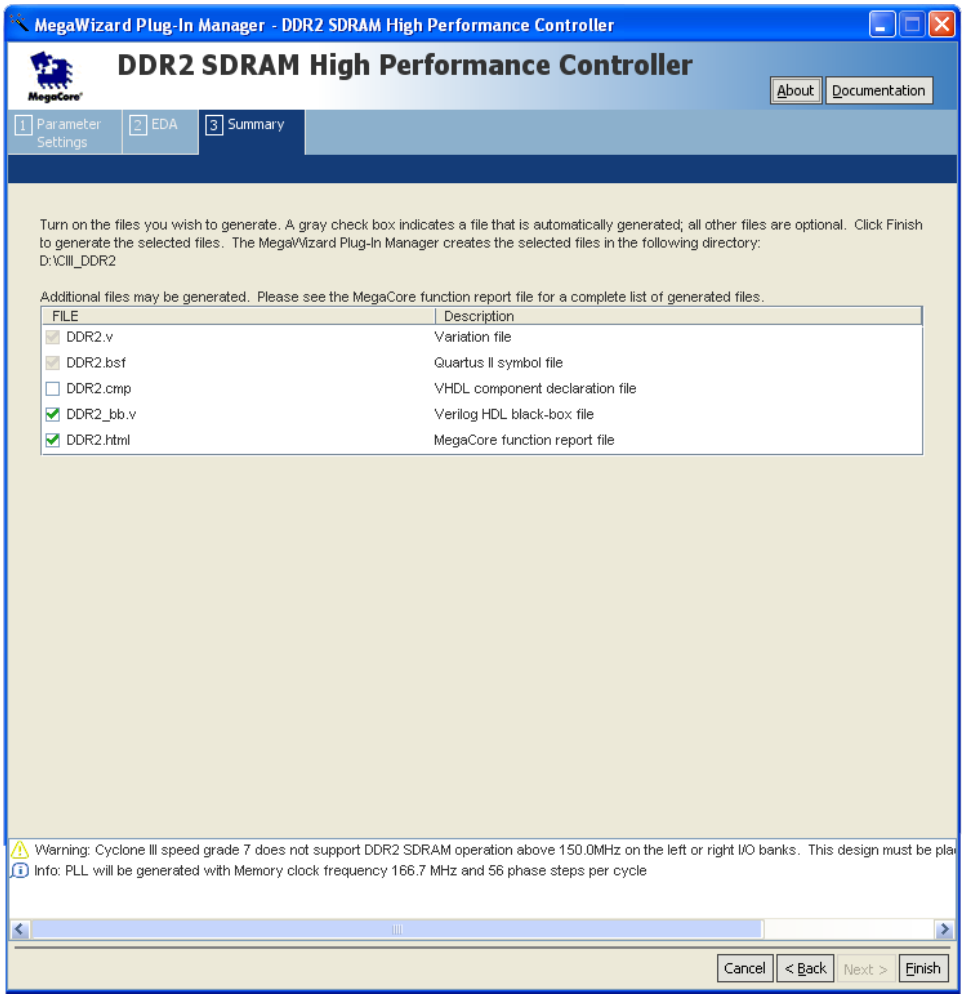
Figure 3–6. DDR2 SDRAM Controller with ALTMEMPHY IP EDA



In the **Timing and Resource Estimation**, you can choose to generate a netlist if you are synthesizing your design with a third-party EDA synthesis tool.

Figure 3–7 shows the **Summary** panel.

Figure 3–7. Summary



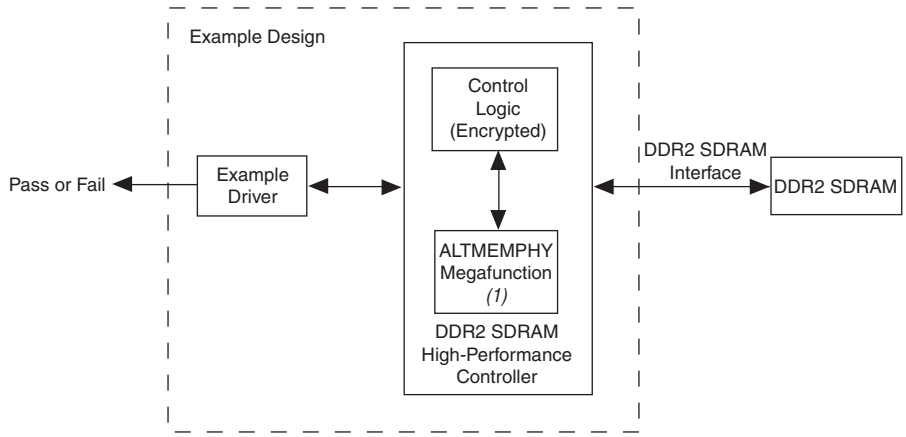
For detailed step-by-step instructions about configuring the DDR2 SDRAM Controller with ALTMEMPHY IP, refer to the *DDR and DDR2 SDRAM Controller with ALTMEMPHY IP User Guide* section in volume 3 of the *External Memory Interface Handbook*.

Click **Finish** to generate the controller.

Perform RTL or Functional Simulation (Optional)

After instantiating the DDR2 SDRAM Controller with ALTMEMPHY IP, the MegaWizard Plug-In Manager generates a design example and driver for testing the memory interface. Figure 3–8 shows a system-level diagram of the design example that the DDR2 SDRAM Controller with ALTMEMPHY IP MegaWizard creates for you.

Figure 3–8. DDR2 SDRAM Controller with ALTMEMPHY IP System-Level Diagram



Note to Figure 3–8:

(1) The ALTMEMPHY megafunction automatically generates the PLL. The PLL is part of the ALTMEMPHY megafunction.

For more information about the different files generated with the DDR2 SDRAM Controller with ALTMEMPHY IP, refer to the *DDR and DDR2 SDRAM Controller with ALTMEMPHY IP User Guide* section in volume 3 of the *External Memory Interface Handbook*.

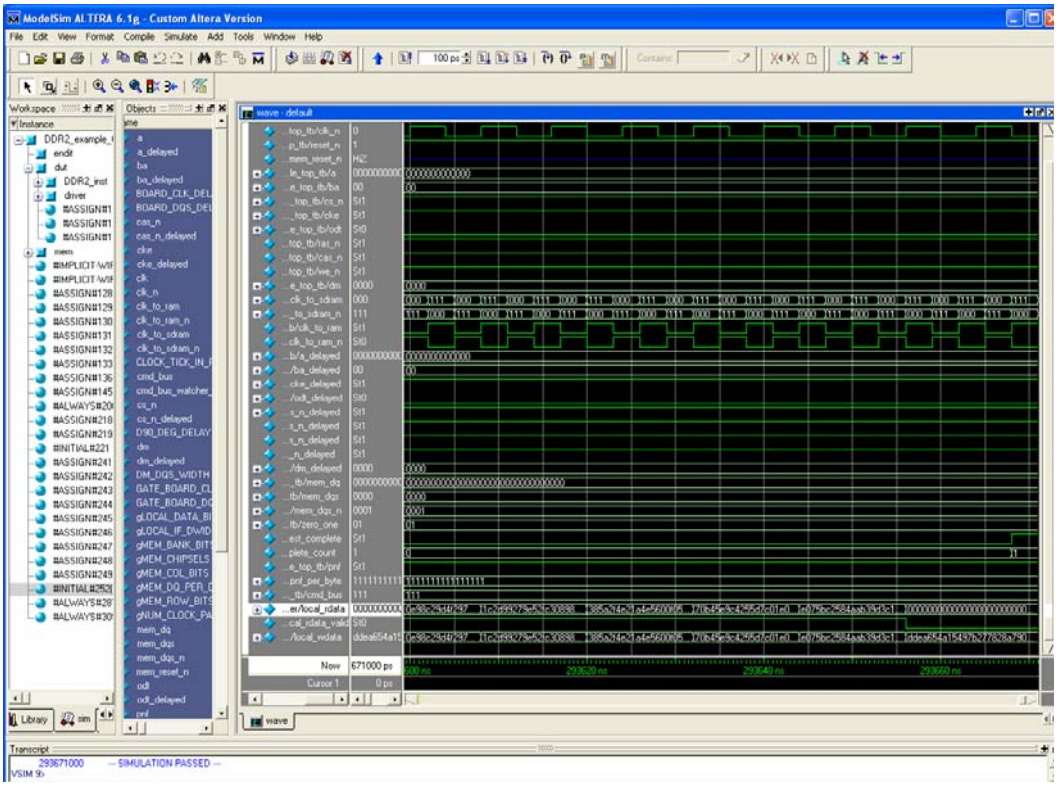
You can simulate the memory interface with the MegaWizard Plug-In Manager-generated IP functional simulation model. You should use this model in conjunction with your own driver or the testbench generated by the MegaWizard Plug-In Manager that issues read and write operations. The memory model file is also automatically generated by the MegaWizard Plug-In Manager in the testbench.

Use the functional simulation model with any Altera-supported VHDL or Verilog HDL simulator. This walkthrough uses the Quartus II NativeLink feature to run the Altera-ModelSim software to perform the simulation.

For more information about how to set up the simulation in Quartus II software using NativeLink, refer to the *Simulation Walkthrough* chapter in volume 4 of the *External Memory Interface Handbook*.

Figure 3–9 shows an Altera-ModelSim RTL simulation.

Figure 3–9. Altera-ModelSim RTL/Functional Simulation



Add Constraints

When you create your memory controller, the MegaWizard Plug-In Manager generates the following constraint files for timing constraint and pin assignment.

- *DDR2_phy_ddr_timing.sdc*
- *DDR2_pin_assignments.tcl*

The DDR2_phy_ddr_timing.sdc file is used to constrain the clock and input/output delays in the ALTMEMPHY megafunction. Enable the TimeQuest timing analyzer before compiling your design. To enable the TimeQuest timing analyzer, perform the following steps:

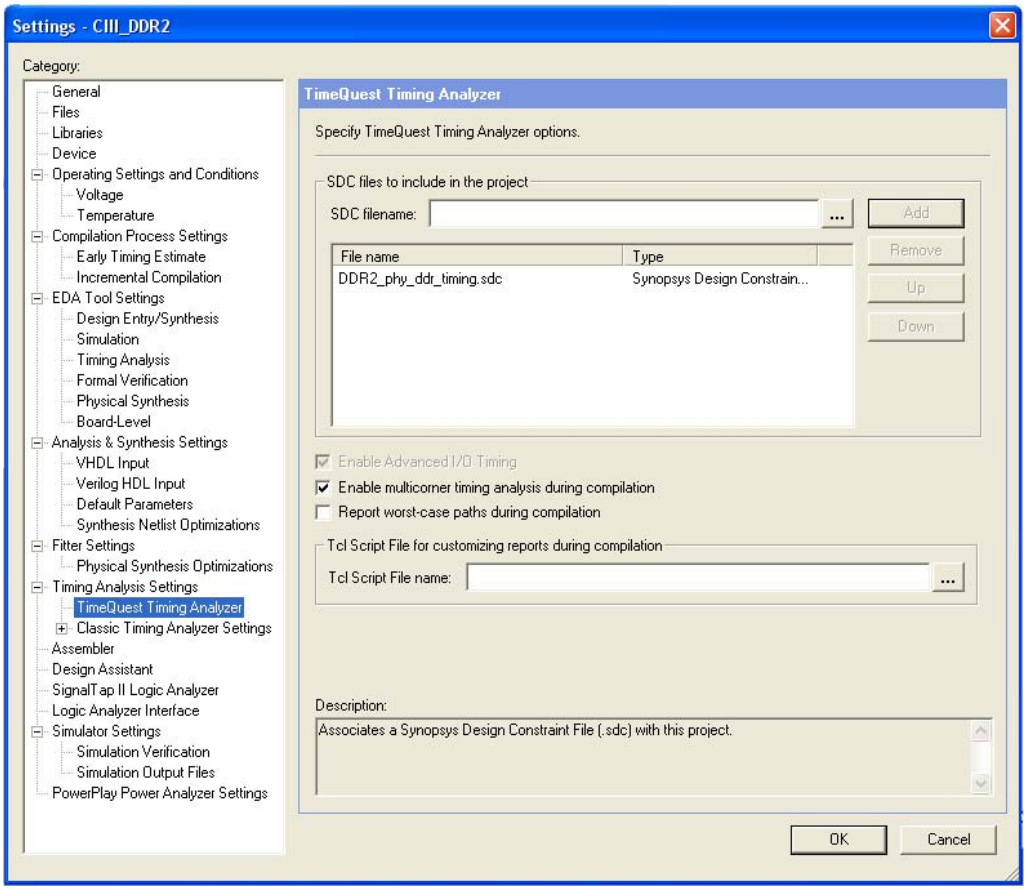
1. On the Assignments menu, click **Settings**. The **Settings** dialog box appears.
2. From the **Category** list, click **Timing Analysis Settings** and select **Use TimeQuest Timing Analyzer during compilation**.
3. Click **OK**.

Next, to add the timing constraints, perform the following steps:

1. On the **Settings** dialog box, click **Timing Analysis Settings** and select **TimeQuest Timing Analyzer**. The **TimeQuest Timing Analyzer** page appears.
2. Specify the SDC file and click **Add** (Figure 3–10).

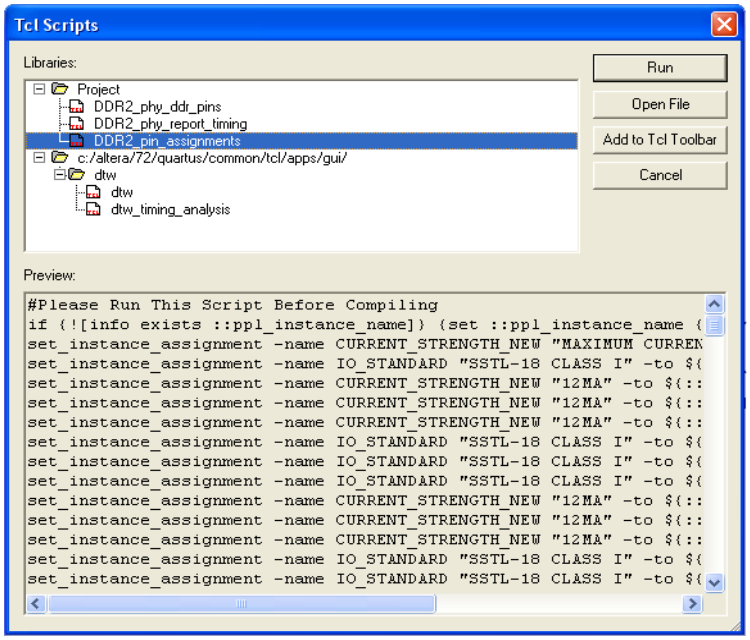
3. Click **OK**.

Figure 3–10. Specifying the Timing Constraint SDC File for the Design Example



The **DDR2_pin_assignments.tcl** file specifies the I/O standard assignment for the memory interface pins. To run the Tcl file, on the **Tools** menu, click **Tcl Scripts**. Select the Tcl file and click **Run** (Figure 3–11 on page 3–12). Running the script adds the information into your Quartus II project. Alternatively, you can also use the Tcl Console to run the Tcl file.

Figure 3–11. Running Pin Assignment Constraint Script in the Tcl Script Panel



After running the **DDR2_pin_assignments.tcl** file, you can assign the I/O locations based on your board design in the **Assignment Editor** or **Pin Planner**. In this design example, assign the I/O locations based on the Cyclone III FPGA development kit board. The 32-bit interface is located at the bottom I/O banks of the device.

Compile Design and Verify Timing Closure

Before you compile the design, set the top level entity of the project to the design example created by the high-performance controller in the previous section, by performing the following steps:

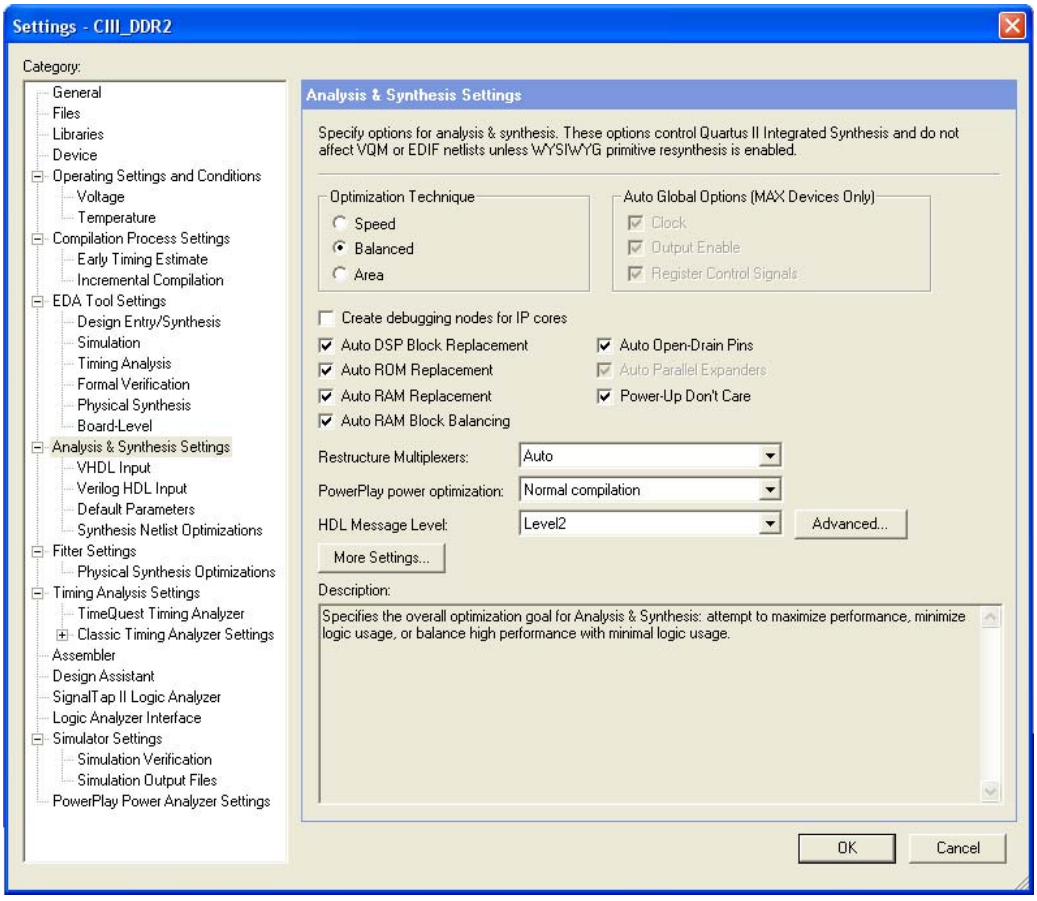
1. On the File menu, click **Open**.
2. Browse to *<variation name>*_**example_top** and click **Open**.
3. On the Project menu, click **Set as Top-Level Entity**.

After you specify that the design example is the top level entity, set the Quartus II software to ensure the remaining unconstrained paths are routed with the highest speed and efficiency. To do so, perform the following steps:

1. On the Assignments menu, click **Settings**. The **Settings** dialog box appears.
2. In the **Category** list, click **Analysis & Synthesis Settings**.

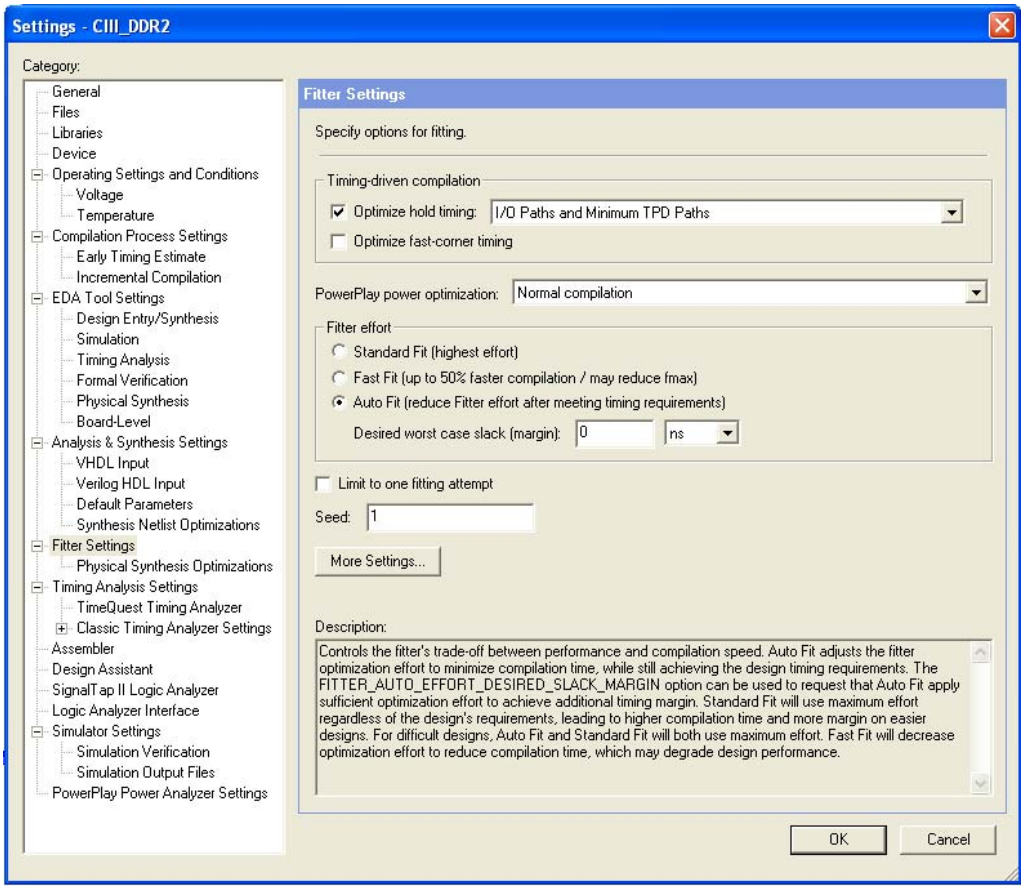
3. Define the **Optimization Technique** setting. The default setting is **Balanced** (Figure 3–12 on page 3–13).

Figure 3–12. Default Setting for Optimization Technique



4. In the **Category** list, click **Fitter Settings**, and set the **Fitter effort**. The default setting is **Auto Fit** (Figure 3–13 on page 3–14).

Figure 3–13. Default Setting for Fitter Effort



To compile your design, perform the following steps:

1. On the Processing menu, click **Start Compilation**.
2. After compilation is successful, on the Tools menu, select **TimeQuest Timing Analyzer**.
3. Generate the timing margin report for your memory interface design by executing the **Report DDR** function from the **Tasks** pane of the TimeQuest Timing Analyzer window (Figure 3–14).

Executing the **Report DDR** task automatically runs the *<variation_name>***_phy_report_timing.tcl** timing margin report script generated by the MegaWizard Plug-In Manager when the megafunction variation was created.

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

Figure 3–14. TimeQuest Timing Analyzer Window

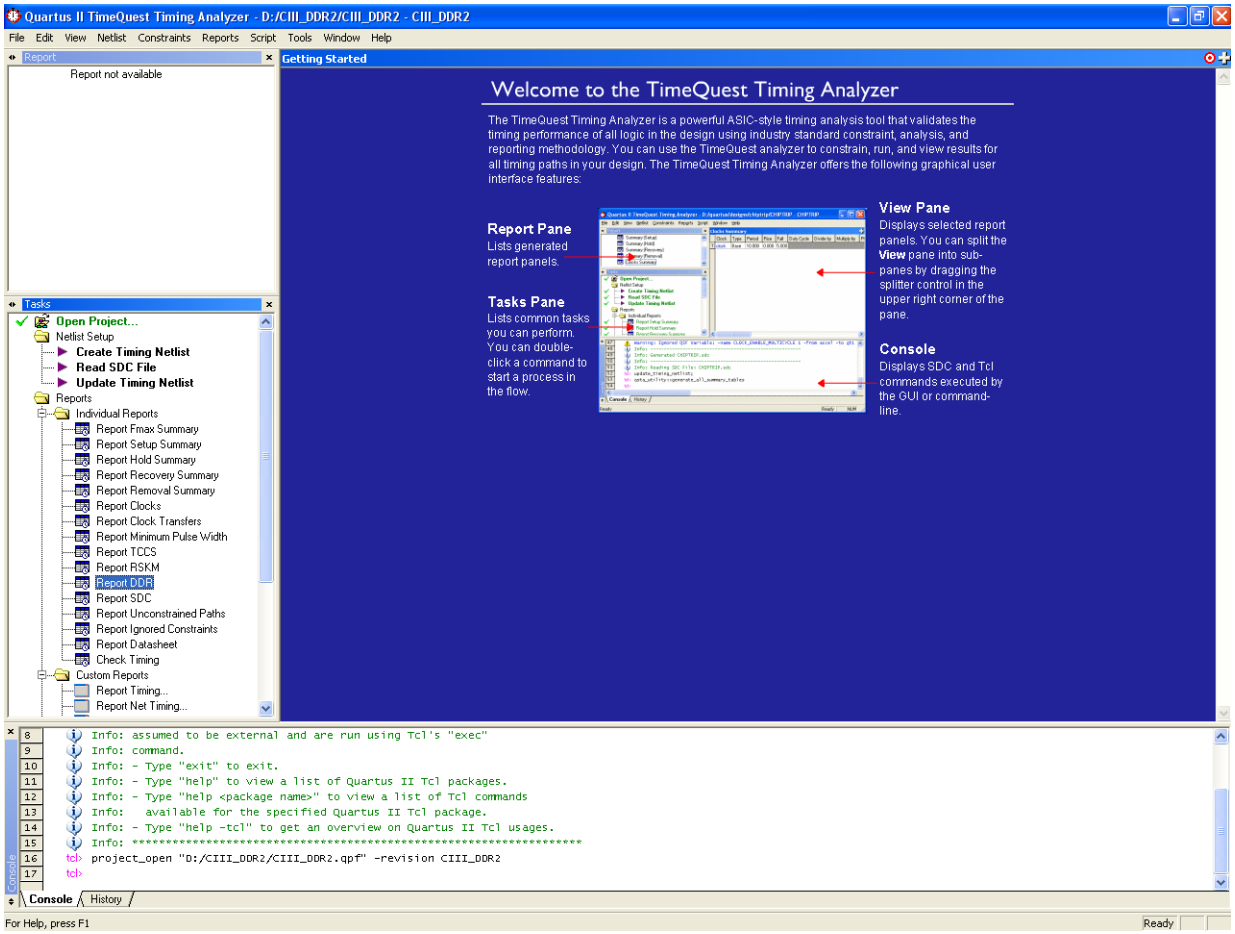
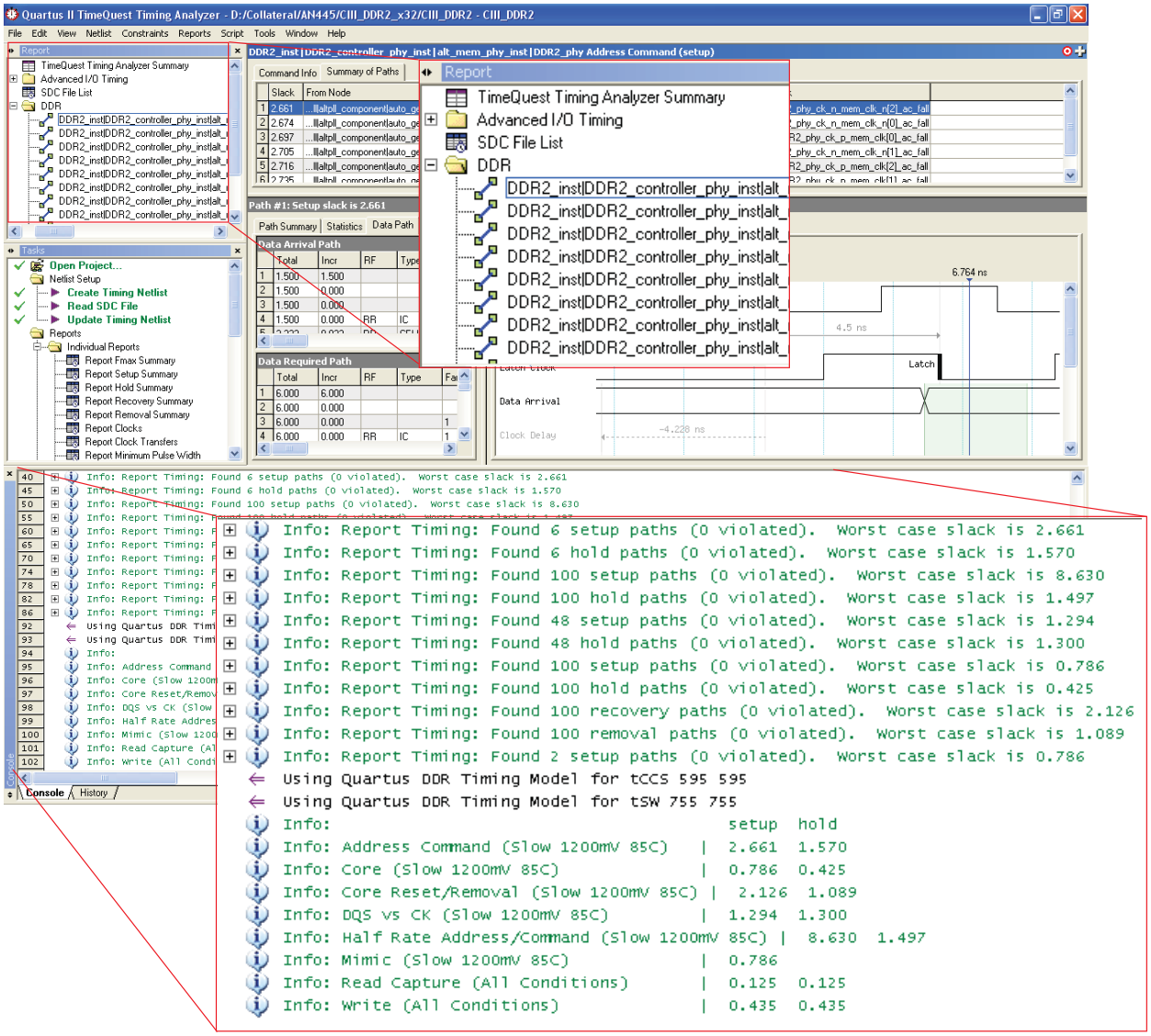


Figure 3–15 shows the output of the Report DDR task. The Report pane contains a new folder titled **DDR** with detailed timing information about the most critical paths, and a timing margin summary similar to the summary on the TimeQuest Console.

Figure 3–15. DDR2 Report Panel and Timing Margin in TimeQuest Timing Analyzer



The report timing script provides information about the following margins and paths:

- Address/command setup and hold margin
- Half-rate address/command setup and hold margin
- Core setup and hold margin
- Core reset/removal setup and hold margin
- DQS vs CK setup and hold margin
- Mimic path

- Write setup and hold margin
- Read capture setup and hold margin

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

Adjust Constraints

Even if the MegaWizard Plug-In Manager generates the controller with the correct settings and you do not see any timing violation, you can manually adjust some of the constraints to improve the timing for your system. The timing margin report shows the address and command datapath with a 2.668 ns setup and 1.601 ns hold margin. Adjusting the clock that is regulating the address and command output registers can balance the setup and hold time better by decreasing the setup margin and increasing

the hold margin on the address and command datapath. To determine the clock that is clocking the address and command registers, in TimeQuest timing analyzer, on the Report panel, click on the address/command report, and select the path for the setup or hold time for the address and command datapath as Path Summary (Figure 3–16) or in Waveform View (Figure 3–17).

Figure 3–16. Path Summary for the Address and Command Datapath

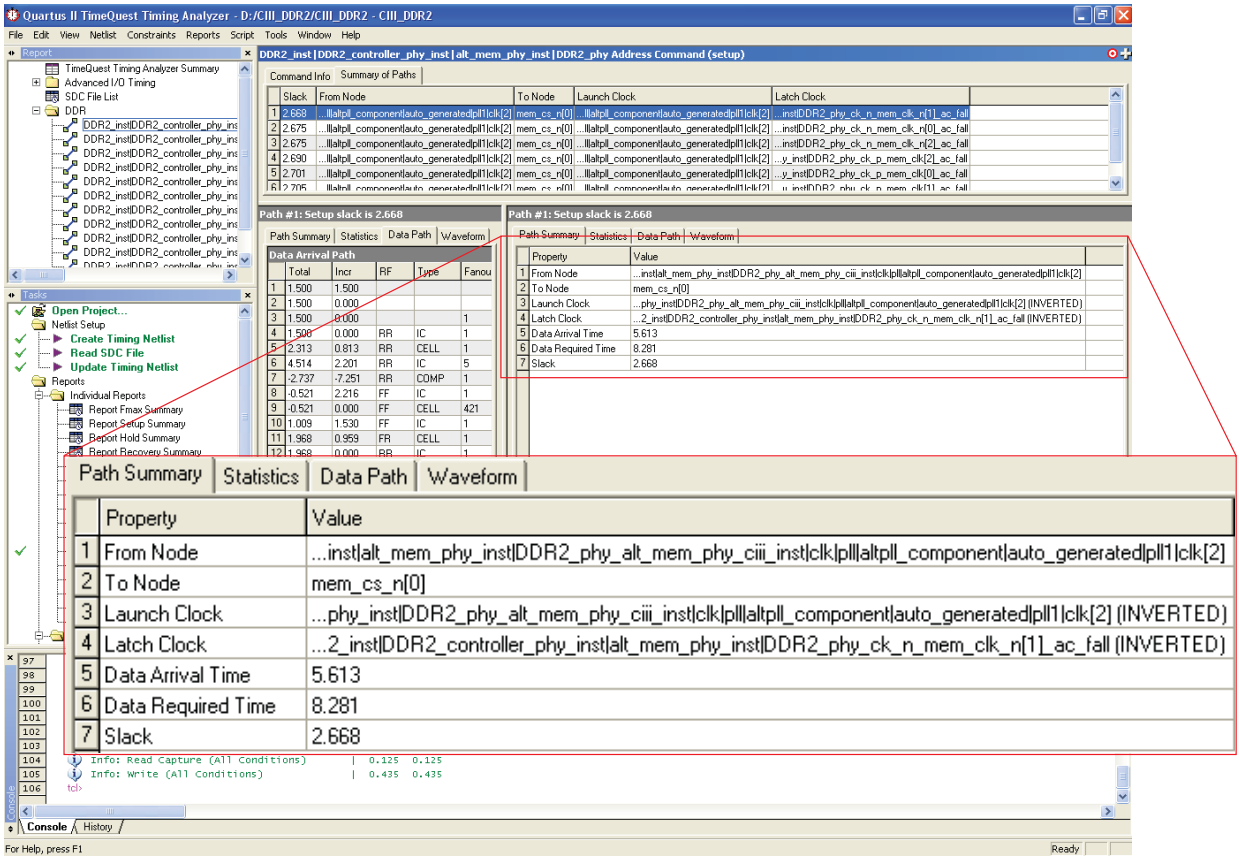
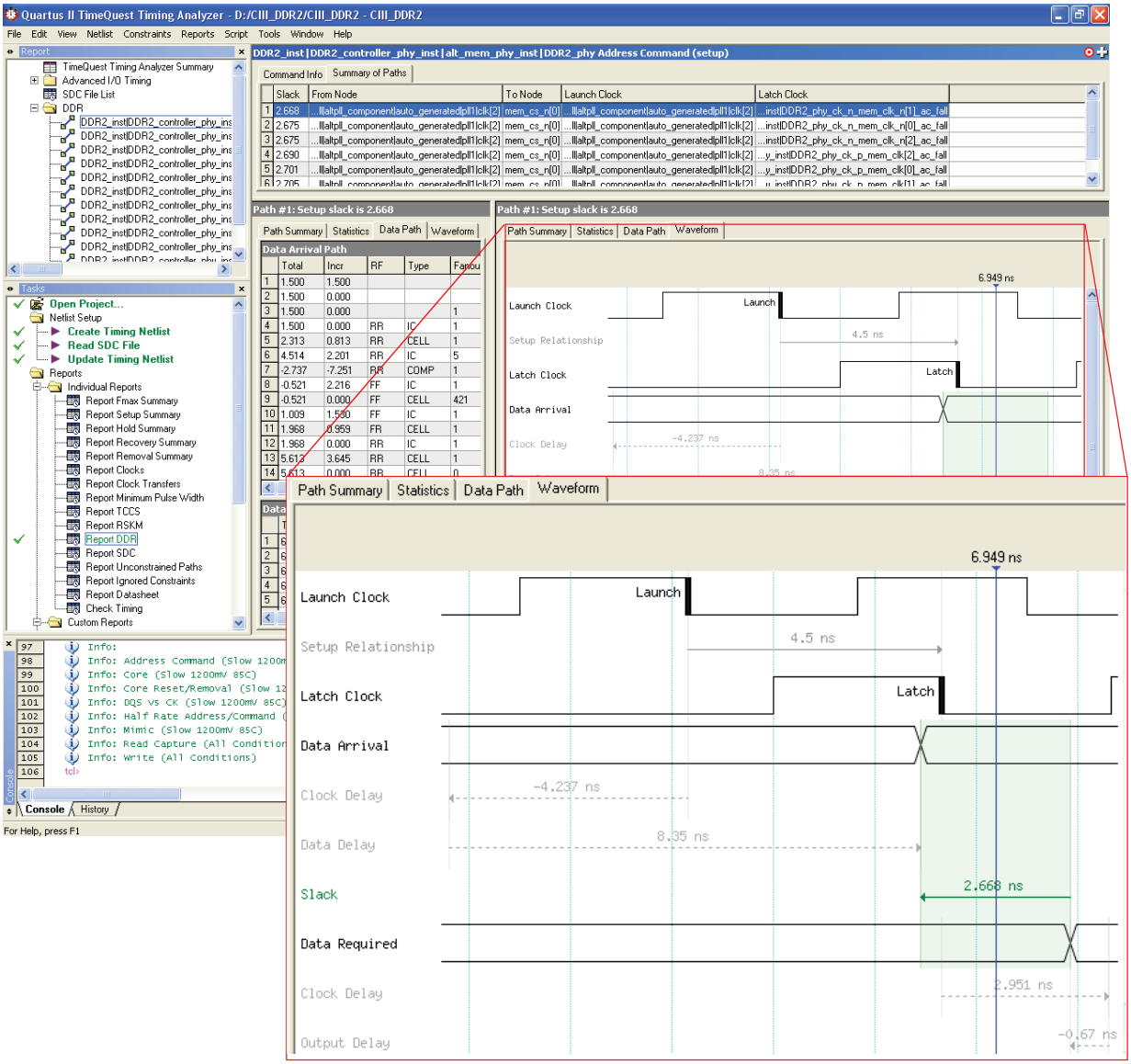


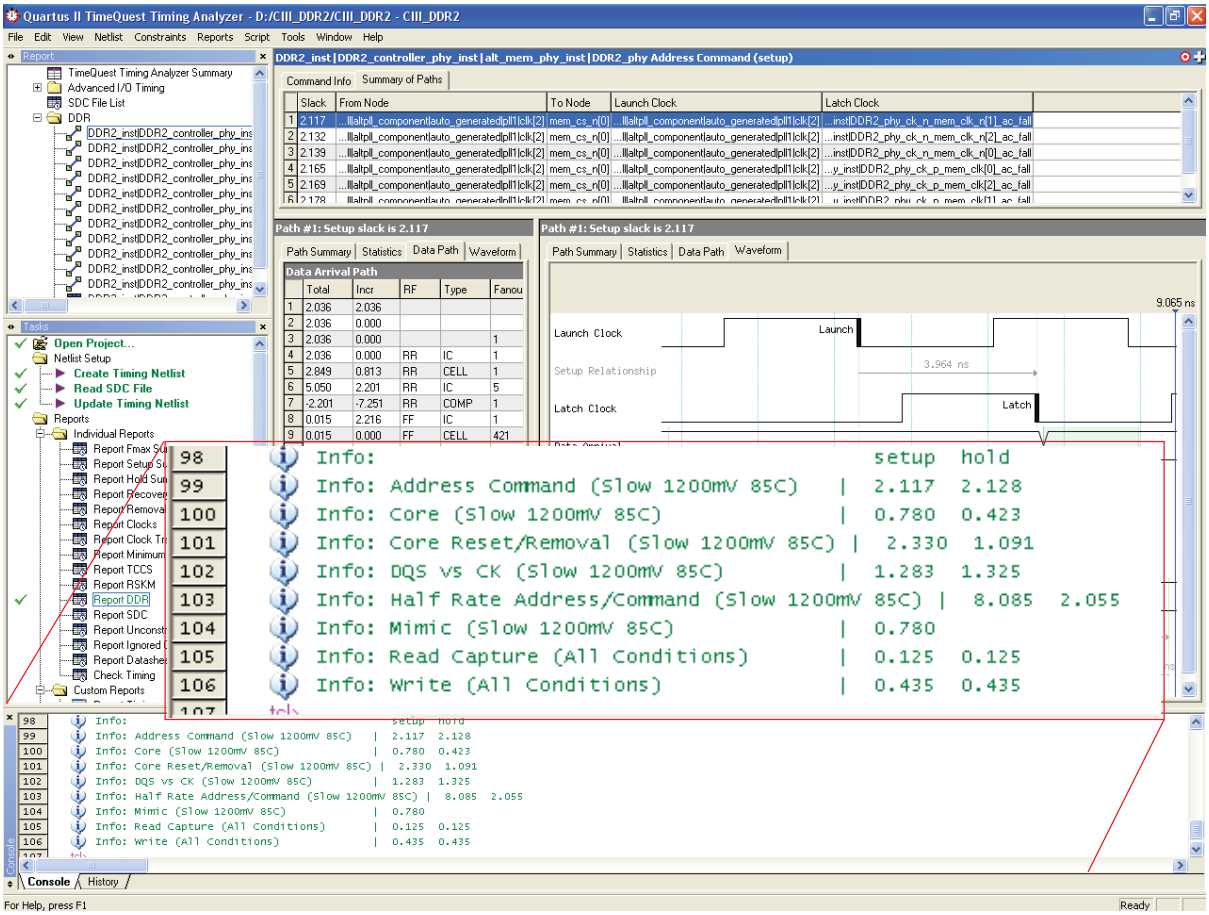
Figure 3–17. Waveform View for the Address and Command Datapath



The report indicates that `clk2` of the PLL is clocking the address and command registers. Edit PLL megafunction to change the phase setting of clk2. For this design, the initial phase setting of `clk2` is –90° with reference to the system clock. By adjusting `clk2` to later than –90° (specifying that the address and command launch later), the setup margin is decreased and the hold margin is increased.

Modify the clk2 phase setting to –55°, and recompile the design for the new PLL setting to take effect. Run the report timing script again. Figure 3–18 shows the timing margin reported in the Quartus II software after adjusting the phase setting of clk2. The address and command datapath now has a more balanced 2.117 ns setup and 2.128 ns hold margin.

Figure 3–18. Timing Margin Reported After Adjusting clk2



Determine Board Design Constraints

Both Cyclone III and Cyclone IV FPGA support the series on-chip termination (OCT) to improve signal integrity and simplify the board design. The Cyclone III and Cyclone IV devices support OCT with or without calibration. In addition, you can choose the resistance value 25 Ω or 50 Ω depending on the I/O standards you use for the memory interface and the termination scheme.

The DDR2 SDRAM supports dynamic parallel on-die termination (ODT). This feature allows you to turn on ODT when the FPGA is writing to the DDR2 SDRAM memory and turn off ODT when the FPGA is reading from the DDR2 SDRAM memory. The ODT features are available in settings of 150 Ω, 75 Ω, and 50 Ω. The 50-Ω setting is only available in DDR2 SDRAM with operating frequencies greater than 267 MHz.

Refer to the respective memory data sheet for additional information about the available settings for the ODT and the output driver impedance features, and the timing requirements for driving the ODT pin in DDR2 SDRAM.

For this design walkthrough, which targets the Cyclone III FPGA development kit, drive strength setting is used together with Class I termination. Using Class I termination allows a higher maximum DDR2 SDRAM clock frequency and reduces the number of external resistors required on the board. You can adjust the current strength of the output pin of the Cyclone III and Cyclone IV devices according to your board setup. If the current strength is too high, you might see excessive overshoot and undershoot of your signal. Use the oscilloscope to check the signal overshoot and undershoot.

Figure 3–19 shows the setup for write operation to the DDR2 SDRAM memory with Class I termination together with the drive strength setting of the Cyclone III or Cyclone IV device. In this setup, the driver's (FPGA) output impedance matches that of the transmission line, resulting in optimal signal transmission to the DDR2 SDRAM memory. On the receiver (DDR2 SDRAM memory) side, the receiving pin is properly terminated with matching impedance to the transmission line, through the external pull-up resistor to V_{TT}, to eliminate any ringing or reflection.

Figure 3–19. Write Operation to DDR2 SDRAM Memory with Class I Termination

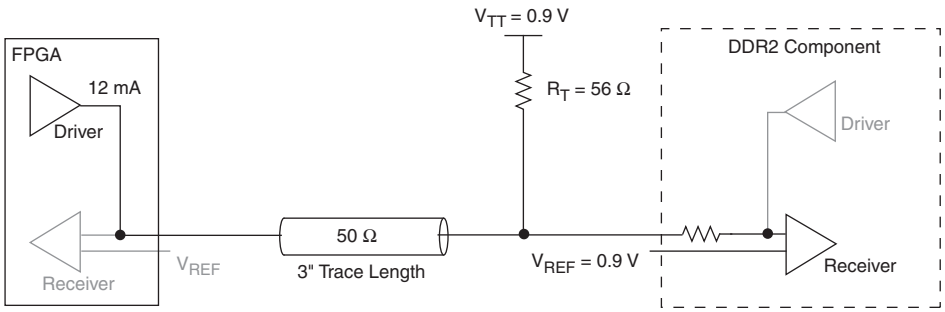
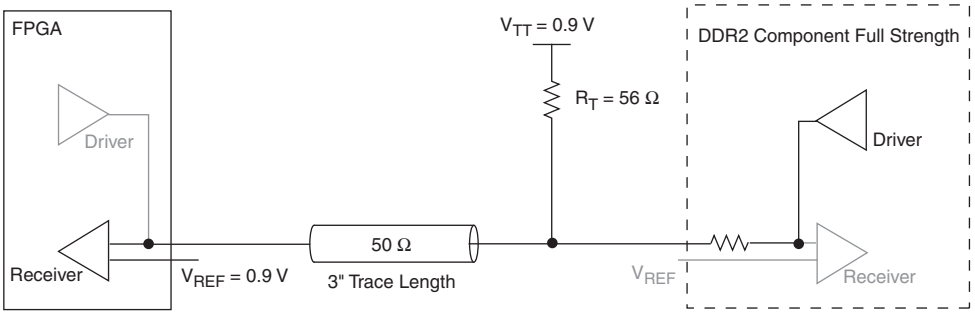


Figure 3–20 shows the setup for read operation from the DDR2 SDRAM memory.

Figure 3–20. Read Operation from DDR2 SDRAM Memory with Class I Termination



If you choose not to use the drive strength setting, you can use the OCT feature of the Cyclone III or Cyclone IV device instead.

Finally, the loading seen by the FPGA during writes to the memory is different in a system using dual-inline memory modules (DIMMs) and a system using components. The additional loading from the DIMM connector can reduce the edge rates of the signals arriving at the memory, thus affecting available timing margin.

For more information about Cyclone III and Cyclone IV OCT, refer to the *I/O Features in the Cyclone III Device Family* chapter in volume 1 of the *Cyclone III Device Handbook*, and the *I/O Features in Cyclone IV Devices* chapter in volume 1 of the *Cyclone IV Device Handbook*.

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

4. Using DDR and DDR2 SDRAM Devices in Stratix III and Stratix IV Devices

This tutorial describes how to use the design flow to design a 72-bit wide, 400-MHz, 800-Mbps DDR2 SDRAM interface with Stratix® III devices. This design example also provides some recommended settings, including termination scheme and drive strength setting, to simplify the design. Although the design example is specifically for the DDR2 SDRAM interface, the design flow for a DDR SDRAM interface is the same.

The design example targets the Stratix III FPGA development kit, which includes a 72-bit wide 1-GB Micron MT9HTF12872AY-800E 400-MHz DDR2 SDRAM DIMM.

To download the design example, **emi_ddr2_siii.zip**, go to the External Memory Interface Design Examples page. You can also use this design example if you are targeting a HardCopy® device for your design. For ALTMEMPHY megafunction supported devices, refer to mmmm section in volume 3 of the *External Memory Interface Handbook*.

For more information about design flow, refer to the *Recommended Design Flow* section in volume 1 of the *External Memory Interface Handbook*.

Software Requirements

This tutorial assumes that you have experience with the Quartus II software. This tutorial requires the following hardware and software:

- Quartus II software version 9.0
- DDR and DDR2 SDRAM Controllers with ALTMEMPHY IP version 9.0
- Stratix III FPGA development kit

Create a Quartus II Project

Create a project in the Quartus II software that targets the EP3SL150F1152-C2 device.

For detailed step-by-step instructions about how to create a Quartus II project, refer to the Quartus II software tutorial by clicking the Help menu in the Quartus II window and selecting **PDF Tutorials**.

Instantiate and Parameterize a Controller

After creating a Quartus II project, you need to instantiate a controller and its parameters.

Instantiate a Controller

To instantiate a controller, perform the following steps:

1. Start the MegaWizard™ Plug-In Manager.

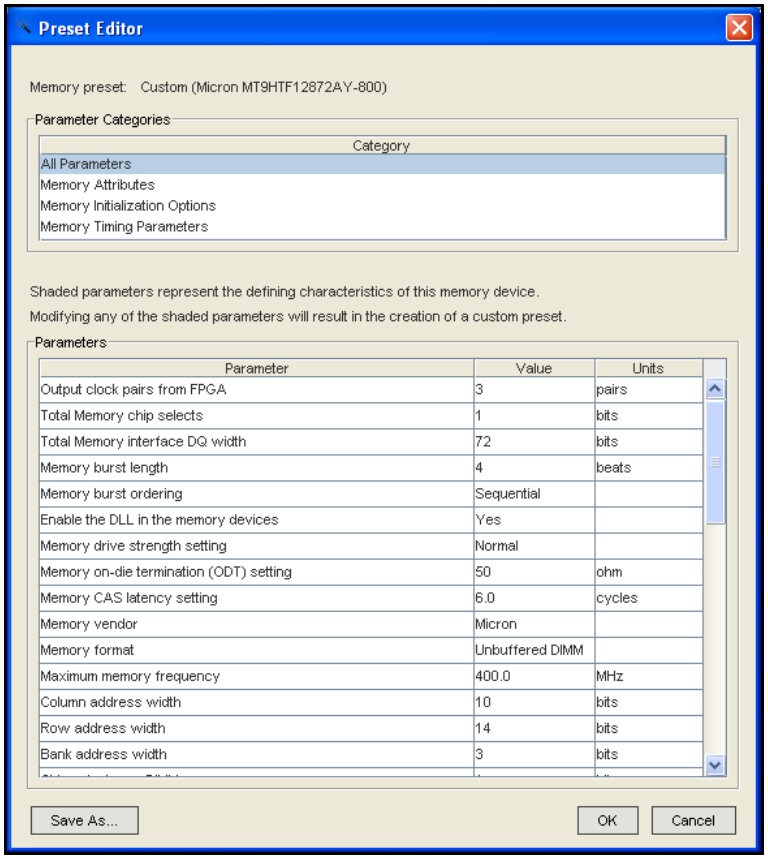
2. In the MegaWizard Plug-In Manager, expand **External Memory** in the **Interfaces** folder and select the **DDR2 SDRAM Controller with ALTMEMPHY**.
3. Type `ddr2_dimm` for the name of the DDR2 SDRAM high-performance controller.

Parameterize DDR2 SDRAM with Stratix III

To parameterize the DDR2 SDRAM high-performance controller to interface with a 400-MHz, 72-bit wide DDR2 SDRAM interface.

1. In the **Memory Setting** tab, set **Speed grade** to **2**.
2. For **PLL reference clock frequency**, type **50** MHz (to match the on-board oscillator).
3. For **Memory clock frequency**, type **400** MHz.
4. For the **Memory Presets**, select **Micron MT9HTF12872AY-800**, which gives a 72-bit wide 1-GB 400-MHz DDR2 DIMM.
5. To create or modify a memory preset, click **Modify parameters**. In the **Preset Editor** dialog box, you can modify the memory presets, see Figure 4–1.

Figure 4–1. Modify the Memory Presets to Create a Custom Memory



The t_{IS}, t_{IH}, t_{DS}, and t_{DH} parameters typically require slew rate derating.

For more information on slew rate derating and how to perform slew rate derating calculations, refer to the memory vendor datasheet.

Simulation and measurement show the following slew rate for the clock, address and command, DQ, and DQS pins on the Stratix III Development Board when using the default I/O standard and drive options:

- Address and command = 0.5 V/ns
- CLK and CLK# = 1.5 V/ns (differential)
- DQ = 1.5 V/ns
- DQS and DQSn = 2.8 V/ns (differential)

Hence, the correct t_{IS}, t_{IH}, t_{DS}, and t_{DH} values for this design are:

- $t_{IS} = t_{ISb} + \Delta t_{IS} + (V_{IHAC} - V_{REF})$/address and command rising slew rate
  $= 175 + (-30) + 400 = 545$ ps
- $t_{IH} = t_{IHb} + \Delta t_{IH} + (V_{REF} - V_{ILDC})$/address and command rising slew rate
  $= 250 + (-95) + 250 = 405$ ps
- $t_{DS} = t_{DSa} + \Delta t_{DS} + (V_{IHAC} - V_{REF})$/DQ rising slew rate
  $= 50 + 67 + 133 = 250$ps
- $t_{DH} = t_{DHa} + \Delta t_{DH} + (V_{REF} - V_{ILDC})$/DQ rising slew rate
  $= 125 + 42 + 83 = 250$ ps

You should always simulate or measure your own design and topology to ensure accurate timing information and analysis.

For information about how to derate these numbers, refer to *Derate Memory Setup and Hold Timing* in the *DDR and DDR2 SDRAM Controller with ALTMEMPHY IP User Guide* section in volume 3 of the *External Memory Interface Handbook*.

The DDR2 SDRAM has a write requirement (t_{DQSS}) that states the positive edge of the DQS signal on writes must be within ± 25% (± 90°) of the positive edge of the DDR2 SDRAM clock input. To achieve this skew requirement, ALTMEMPHY-based designs always use DDR IOE registers to generate the CK and CK# signals.

6. In the **PHY Settings** tab, under **Advanced PHY Settings** turn on the **Use differential DQS** option to enhance signal to noise ratio. Turn on this option where noise margin is a concern. Differential DQS is recommended for DDR2 SDRAM interfaces operating at above 266 MHz and enhances signal to noise ratio. The Stratix III development board is routed for differential DQS.
7. Turn on the **Enable dynamic parallel on-chip termination (OCT)** option for this example. The Stratix III development board does not include discrete external termination on the DQ, DQS, or DM pins, as it was designed to use OCT.
8. Under **Address/Command Clock Settings**, for **Dedicated clock phase** type **240**. Timing analysis shows that 240° is optimal for the Stratix III development board.

9. Under **Board Timing Parameters**, for **Board skew** type **20 ps**. This timing parameter is the board trace variation between the DQ and DQS pins. If your board can perform better or worse than this value, update it accordingly. The wizard uses this number to calculate the overall system timing margin. For this design example, type the value of 20 ps as the board skew tolerance target is 20 ps.

The **Auto-Calibration Simulation Options** parameter is for RTL simulation only and is not applicable for gate-level simulation.

10. Click **Next**.
11. Click **Next**.
12. Turn the **Generate simulation model** option.
13. Click **Finish** to generate your MegaCore function variation. The MegaWizard Plug-In generates all the files necessary for your DDR2 SDRAM controller, and generates an example top-level design, which you may use to test or verify board operation.

For detailed step-by-step instructions for parameterizing the DDR2 SDRAM Controller with ALTMEMPHY IP, refer to *Volume 3: Implementing Altera Memory Interface IP* of the *External Memory Interface Handbook*.

Perform RTL or Functional Simulation (Optional)

This section describes RTL and functional simulation.

Set Up Simulation Options

To set up simulation option, perform the following steps:

1. Obtain and copy the vendors memory model to a suitable location. For example, obtain the **ddr2.v** and **ddr2_parameters.vh** memory model files from the Micron website and save them in the testbench directory.

   Some vendor DIMM models do not use DM pin operation, which can cause calibration failures. In these cases, use the vendors component models directly.

2. Open the memory model file in a text editor and add the following define statements to the top of the file:

   ```
   'define sg25
   'define x8
   ```

   The two define statements prepare the DDR2 SDRAM interface model.

   The first statement specifies the memory device speed grade as –25. The second statement specifies the memory device width per DQS.

3. Make sure the following statement is included in the model file:

   ```
   `include "ddr2_parameters.vh"
   ```

4. Open the testbench in a text editor, instantiate the downloaded memory model, and connect its signals to the rest of the design.

5. Delete the whole section between the START and END MEGAWIZARD comments.

```
// << START MEGAWIZARD INSERT MEMORY_ARRAY

    // This will need updating to match the memory models you are
using.


   // Instantiate a generated DDR memory model to match the datawidth
& chipselect requirements


    ddr2_dimm_mem_model mem (
        .mem_dq       (mem_dq),
        .mem_dqs      (mem_dqs),
        .mem_dqs_n    (mem_dqs_n),
        .mem_addr     (a_delayed),
        .mem_ba       (ba_delayed),
        .mem_clk      (clk_to_ram),
        .mem_clk_n    (clk_to_ram_n),
        .mem_cke      (cke_delayed),
        .mem_cs_n     (cs_n_delayed),
        .mem_ras_n    (ras_n_delayed),
        .mem_cas_n    (cas_n_delayed),
        .mem_we_n     (we_n_delayed),
        .mem_dm       (dm_delayed),
        .mem_odt      (odt_delayed)
    );


    // << END MEGAWIZARD INSERT MEMORY_ARRAY
```

6. Instantiate the first instance of the DDR2 SDRAM model by editing the following section in the testbench file:

```
ddr2 memory_0 (
    .clk      (clk_to_ram),
    .clk_n    (clk_to_ram_n),
    .cke      (cke_delayed),
    .cs_n     (cs_n_delayed),
    .ras_n    (ras_n_delayed),
    .cas_n    (cas_n_delayed),
    .we_n     (we_n_delayed),
    .dm_rdqs       (dm_delayed[0]),
    .ba       (ba_delayed),
    .addr     (a_delayed),
    .dq       (mem_dq[7:0]),
    .dqs      (mem_dqs[0]),
```

```
    .dqs_n    (mem_dqs_n[0]),
    .rdqs_n (),
    .odt      (odt_delayed)
    );
```


Make sure that port names in the memory model match with those in the testbench. Also the names and case of the names should be the same.

7. Similarly, create the other eight instances of the DDR2 module by changing the DQ, DQS, DQS_N, and DM bus indices, and the instance name `memory_0`. The following code shows the second ×8 memory instance of the DDR2 module:

```
ddr2 memory_1 (
    .clk      (clk_to_ram),
    .clk_n    (clk_to_ram_n),
    .cke      (cke_delayed),
    .cs_n     (cs_n_delayed),
    .ras_n    (ras_n_delayed),
    .cas_n    (cas_n_delayed),
    .we_n     (we_n_delayed),
    .dm_rdqs       (dm_delayed[1]),
    .ba       (ba_delayed),
    .addr     (a_delayed),
    .dq       (mem_dq[15:8]),
    .dqs      (mem_dqs[1]),
    .dqs_n    (mem_dqs_n[1]),
    .rdqs_n (),
    .odt      (odt_delayed)
    );
```

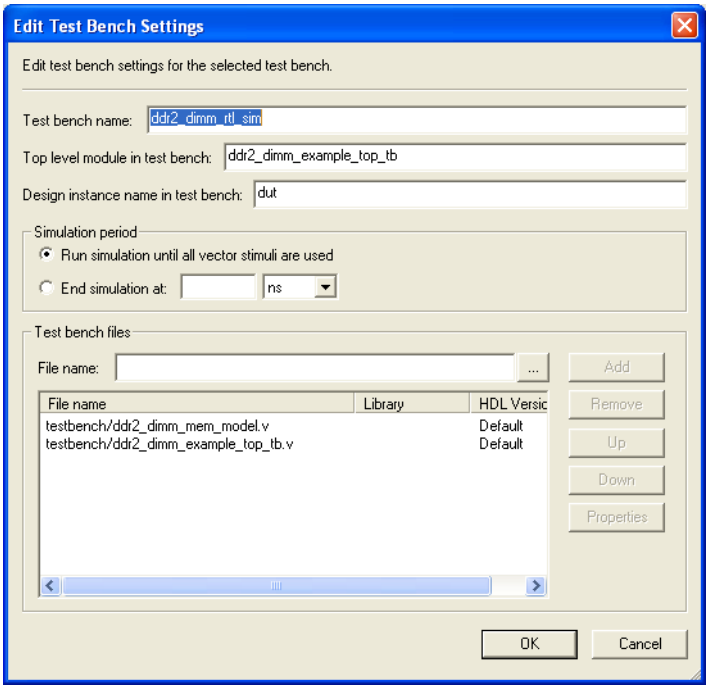
8. To launch the ModelSim simulator from the Quartus II software, set the path to the ModelSim simulator in the Quartus II software:
   a. On the Tools menu click **Options**.
   b. Click **EDA Tools Options** in the **Category** list.
   c. Set the path to the simulator.

Run Simulation with NativeLink

To run the simulation with NativeLink, perform the following steps:

1. Set the absolute path to your third-party simulator executable file, by performing the following steps:
   a. On the Assignments menu, click **EDA Tool Settings**.
   b. In the **Category** list, expand **EDA Tool Settings** and click **Simulation**.
   c. Under **Tool Name**, select **ModelSim-Altera**.
   d. Under **NativeLink settings**, select **Compile test bench** and click **Test Benches**.
   e. Click **New**.
2. In the **Edit Test Bench Settings** dialog box, perform the following steps:
   a. Type the testbench name, testbench top-level module, design instance name, and simulation period.
   b. In the **Test bench files** field, include the testbench file and the memory model file (Figure 4–2).
   c. Click **OK**.

Figure 4–2. Testbench Files



3. To elaborate your design, on the Processing menu, point to **Start** and click **Start Analysis & Elaboration**.
4. On the Tools menu, point to the **Run EDA Simulation Tool** and click **EDA RTL Simulation**. This step creates the **\simulation** directory in your project directory and a script that compiles all necessary files and runs the simulation.

For example timing diagrams, refer to *Volume 3: Implementing Altera Memory Interface IP* of the *External Memory Interface Handbook*.

Add Constraints

After instantiating the DDR2 SDRAM high-performance controller, the ALTMEMPHY megafunction generates the constraints files for the design example. Apply these constraints to the design before compilation.

Add Timing Constraints

When you instantiate an SDRAM high-performance controller, it generates a timing constraints file, <*variation_name*>**_phy_ddr_timing.sdc**. The timing constraint file constrains the clock and input and output delay on the SDRAM high-performance controller.

To add timing constraints, perform the following steps:

1. On the Assignments menu click **Settings**.
2. In the **Category** list, expand **Timing Analysis Settings**, and select **TimeQuest Timing Analyzer**.
3. Select the <*variation_name*>**_phy_ddr_timing.sdc** file and click **Add**.
4. Click **OK**.

Add Pin and DQ Group Assignments

The pin assignment script, <*variation_name*>**_pin_assignments.tcl**, sets up the I/O standards for the DDR2 SDRAM interface. It also launches the DQ group assignment script, <*variation_name*>**_phy_assign_dq_groups.tcl**, which relates the `DQ` and `DQS` pin groups together for the fitter to place them correctly in the Quartus II software.

This script does not create a clock for the design. You need to create a clock for the design and provide pin assignments for the signals of both the example driver and testbench that the MegaCore variation generates.

Run the <*variation_name*>**_pin_assignments.tcl** to add the pin, I/O standards, and DQ group assignments to the design example.

Set Top-Level Entity

Before compiling the design, set the top-level entity of the project to the correct entity. The ALTMEMPHY megafunction entity is <*variation_name*>**_phy.v** or **vhd**; the SDRAM high-performance controller entity is <*variation_name*>**.v** or **vhd**.

The example top-level design, which instantiates the SDRAM high-performance controller and an example driver, is <*variation_name*>**_example_top.v** or **vhd**.

To set the top-level file, perform the following steps:

1. Open the top-level entity file, <*variation_name*>**_example_top.v** or **vhd**.
2. On the Project menu click **Set as Top-Level Entity**.

Set Optimization Technique

To ensure the remaining unconstrained paths are routed with the highest speed and efficiency, perform the following steps to set the optimization technique:

1. On the Assignments menu, click **Settings**.
2. Select **Analysis & Synthesis Settings**.
3. Select **Speed** under **Optimization Technique**. Click **OK**.

Set Fitter Effort

To set the fitter effort, perform the following steps:

1. On the Assignments menu, click **Settings**.
2. Expand **Fitter Settings**.
3. Turn on **Optimize hold timing** and select **All Paths**.
4. Turn on **Optimize multi-corner timing**.
5. Select **Standard Fit (highest effort)** under **Fitter effort**.
6. Click **OK**.

Enter Pin Location Assignments

To enter the pin location assignments, perform the following steps:

1. Run analysis and synthesis. On the Processing menu, point to **Start** and click **Start Analysis & Synthesis**.
2. Assign all of your pins, so the Quartus II software fits your design correctly and gives correct timing analysis. To assign pin locations for the Stratix III development board, run the Altera-provided **S3_Host_DDR2_PinLocations.tcl** file or manually assign pin locations by using the Pin Planner.

   The SDRAM high-performance controller auto-generated scripts do not make any pin location assignments.

   If you are at the design exploration phase of your design cycle and do not have any PCB defined pin locations, you should still manually define an initial set of pin constraints, which can become more specific during your development process.

To manually assign pin locations using the Pin Planner, perform the following steps:

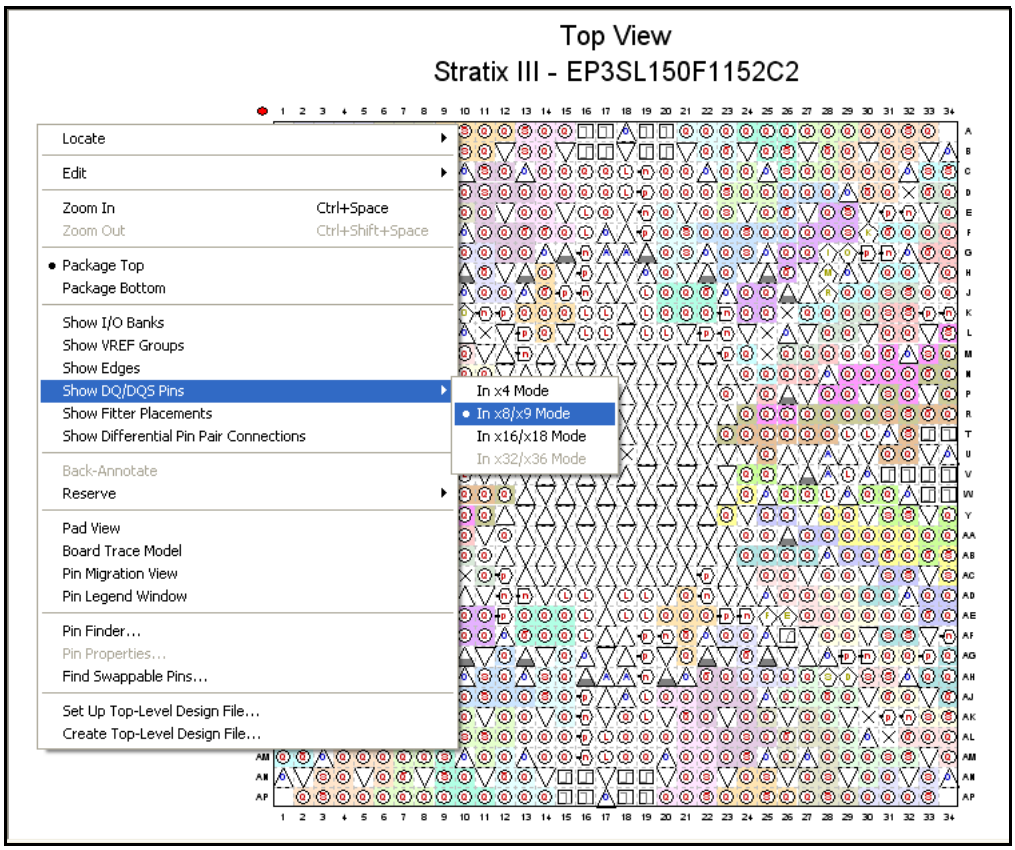
1. On the Assignments menu, click **Pin Planner**.
2. Assign `DQ` and `DQS` pins.
   a. To select the device DQS pin groups that the design uses, assign each `DQS` pin in your design to the required `DQS` pin in the Pin Planner. The Quartus II Fitter then automatically places the respective `DQ` signals onto suitable `DQ` pins within each group. To see DQS groups in the Pin Planner, right-click, select **Show DQ/DQS Pins**, and click **In x8/x9 Mode**. The Pin Planner shows each DQS group in a different color and with a different legend: S = `DQS` pin, Sbar = `DQSn` pin and Q = `DQ` pin, refer to Figure 4–3 on page 4–10.

Most DDR2 SDRAM devices operate in ×8/×9 mode, however as some DDR2 SDRAM devices operate in ×4 mode, refer to your specific memory device datasheet.

b. Select the DQ mode to match the DQ group width (number of DQ pins/number of DQS pins) of your memory device. DQ mode is not related to the memory interface width.

DQ group order and DQ pin order within each group is not important. However, you must place the DQ pins in the same group as their respective strobe pin.

Figure 4–3. Quartus II Pin Planner, Show DQ/DQS Pins, In x8/x9 Mode



3. Place the DM pins within their respective DQ group.
4. Place address and control command pins on any spare I/O pins ideally within the same bank or side of the device as the mem_clk pins.
5. Ensure that you place mem_clk pins on differential I/O pairs for the CK/CK# pin pair. To identify differential I/O pairs, right-click in the Pin Planner and select **Show Differential Pin Pair Connections**. The pin pairs show a red line between each pin pair.

You must place mem_clk[0] and mem_clk_n[0] on a DIFFIO_RX pin pair, if your design uses differential DQS signaling.

6. Ensure `mem_clk` pins use any regular adjacent I/O pins—ideally differential I/O pairs for the `CK/CK#` pin pair. To identify differential I/O pairs, right-click in the Pin Planner and select **Show Differential Pin Pair Connections**. The pin pairs show a red line between each pin pair.
7. Place the `clock_source` pin on a dedicated PLL clock input pin with a direct connection to the SDRAM controller PLL and DLL pair—usually on the same side of the device as your memory interface. This recommendation reduces PLL jitter, saves a global clock resource, and eases timing and fitter effort.
8. Place the `global_reset_n` pin (like any high fan-out signal) on a dedicated clock pin.

For more information on how to use the Quartus II Pin Planner, refer to the *I/O Management* chapter in volume 2 of the *Quartus II Handbook*.

Assign Virtual Pins

The example top-level design, which is auto-generated by the high- performance controller, includes an example driver to stimulate the interface. This example driver is not part of the SDRAM high-performance controller IP, but allows easy testing of the IP.

The example driver outputs several test signals to indicate its operation and the status of the stimulated memory interface. These signals are `pnf`, `pnf_per_byte`, and `test_complete`. These signals are not part of the memory interface, but are to facilitate testing. You should connect these signals to either a debug bus or assign the signals to virtual pins using the Quartus II Assignment Editor. When using the example driver for testing, do not remove these signals from the top-level signal list. If you remove these signals from the top-level module, the Quartus II software optimizes the driver away and the example driver fails.

To assign virtual pin assignments for the Stratix III development board, run the Altera-provided **s3_Host_ddr2_exdriver_vpin.tcl** file or manually assign virtual pin assignments using the Assignment Editor.

Advanced I/O Timing

ALTMEMPHY-based designs assume that the memory address and command signals are matched length to the memory clock signals. Typically, this length match is not true for DIMM-based designs. You should verify the difference in your design. For the Stratix III development board fitted with the MT9HTF12872AY DIMM, the address and command signals remain asserted 750 ps longer than the clock signals.

To amend the TimeQuest **.sdc** file, <*variation name*>**_phy_ddr_timing.sdc**, to include this difference, perform the following steps:

1. Open the **ddr2_dimm_phy_ddr_timing.sdc** file in a text editor and find the following line (usually line 31):

```
set t(additional_addresscmd_tpd) 0.000
```

2. Change the line to the following text:

```
set t(additional_addresscmd_tpd) 0.750
```

3. Save the file.

If the DDR2 SDRAM controller **.sdc** file is regenerated, this change is lost and you must re-edit the file.

Enter Board Trace Delay Models

For accurate I/O timing analysis, the Quartus II project must include the board trace and loading information. This information should be derived and refined during your PCB development process of prelayout (line) simulation and final post-layout (board) simulation. For external memory interfaces that use memory modules (DIMMs), this information should include the trace and loading information of the module in addition to the main and host platform. You can obtain the information from your memory vendor.

To enter board trace information, perform the following steps:

1. In the Pin Planner, select the pin or group of pins that you want to enter the information for.
2. Right-click and select **Board Trace Model**.

Figure 4–4 through 4–14 show a typical board trace model for an address, memory clock, DQ, and DQS pin on the Stratix III development board including the data for the MT9HTF12872AY-800E memory module that is included with the kit.

Figure 4–4. Stratix III Development Board Address Signal Board Trace Model

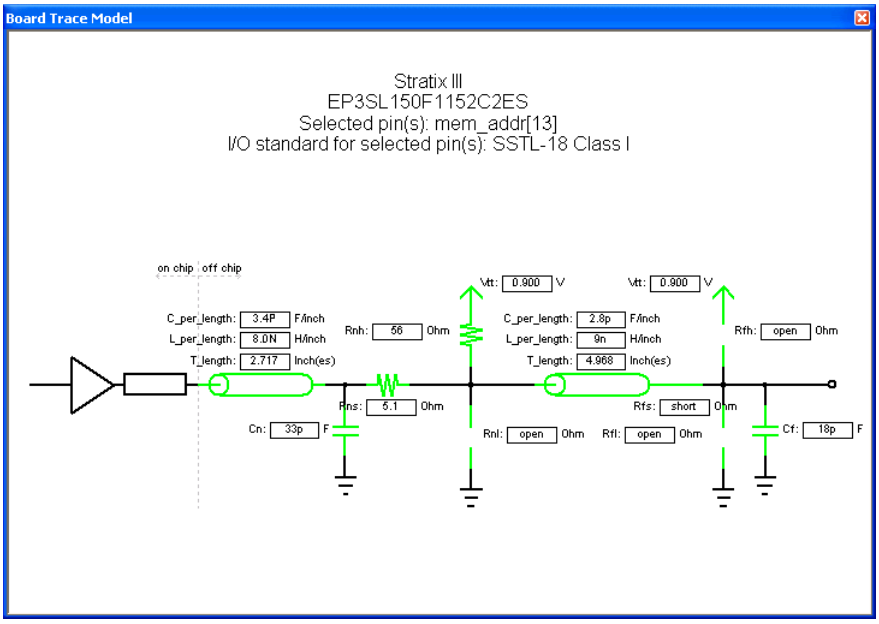


Figure 4–5. Stratix III Development Board Memory Clock Signal Board Trace Model

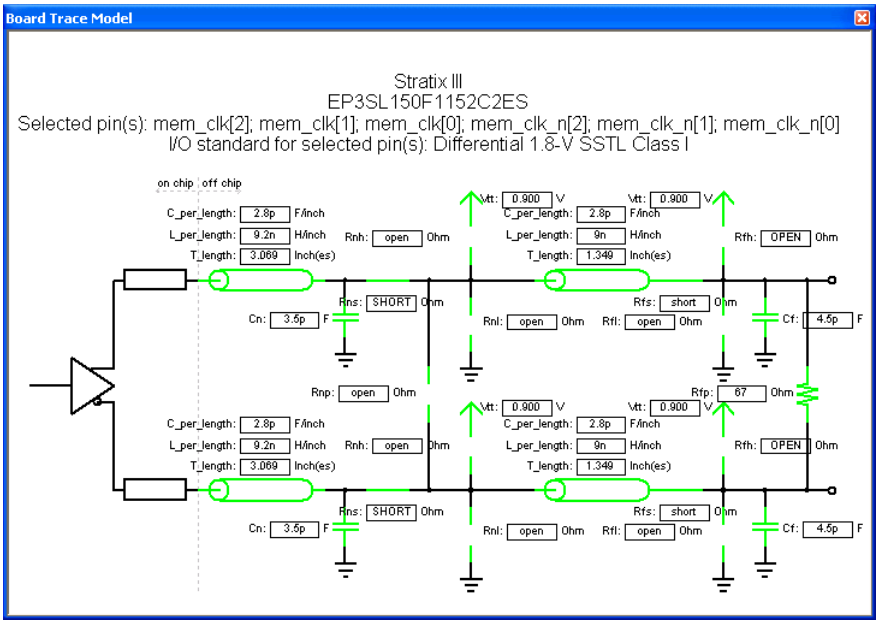


Figure 4–6. Stratix III Development Board DQ Signal Board Trace Model

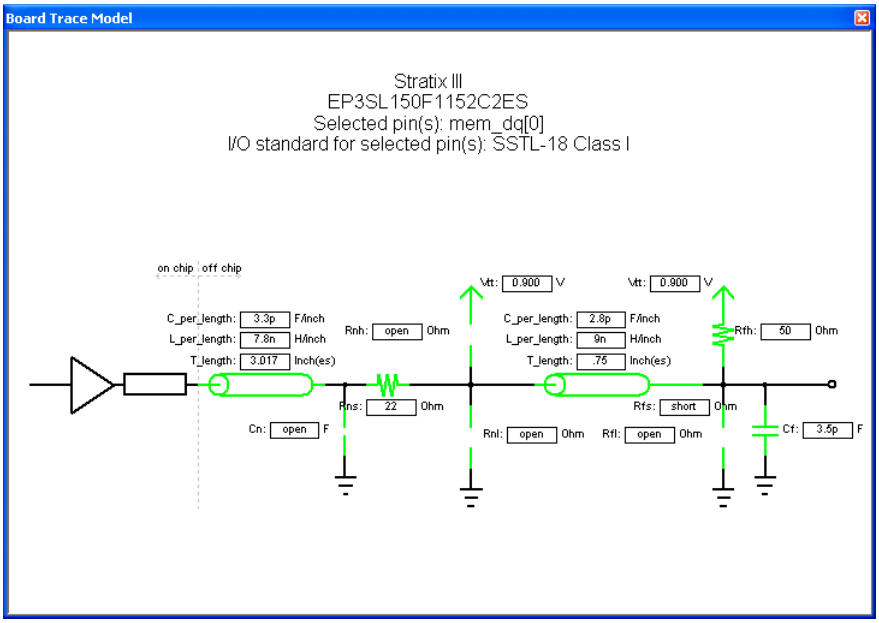


Figure 4–7. Stratix III Development Board DQS Signal Board Trace Model

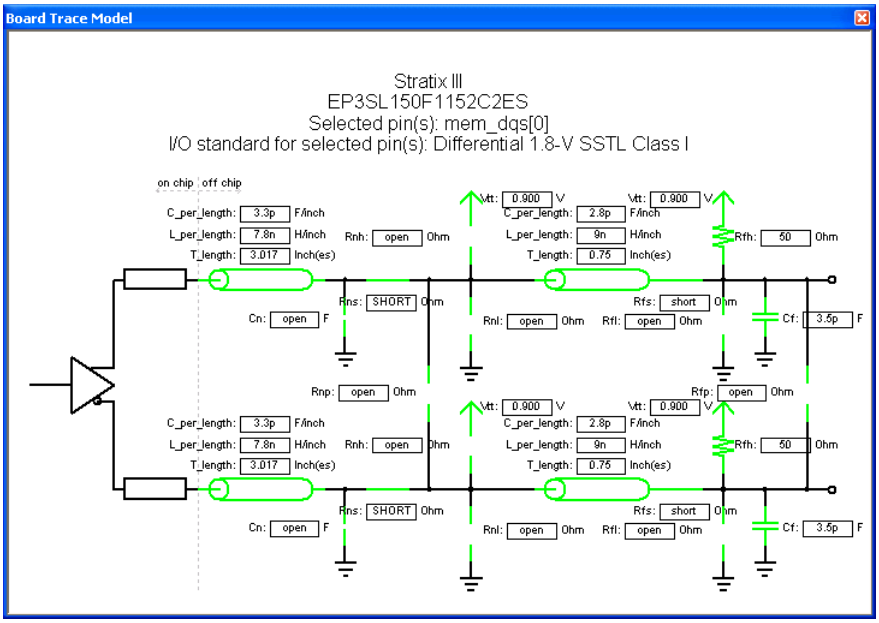


Table 4–1 shows the board trace model parameters for the Stratix III development board.

Table 4–1. Stratix III Development Board Trace Model Summary

Net	Near (FPGA End of Line)						Far (Memory End of Line)				
	Length	C_per_length (pF/In)	L_per_length (nH/In)	Cn (pF)	Rns	Rnh	Length	C_per_length (pF/In)	L_per_length (nH/In)	Cf (pF)	Rfh/Rfp
Addr *(1)*	2.717	3.3	7.8	33	5.1	56	4.968	2.8	9	18	—
CLK	3.069	2.8	9.2	3.5	—	—	1.349	2.8	9	4.5	67
CKE/CS#	2.717	3.3	7.8	33	—	56	3.936	2.8	9	42	—
ODT	2.717	3.3	7.8	33	—	56	3.936	2.8	9	39.75	—
DQS0	3.017	3.4	7.8	—	22	—	0.750	2.8	9	3.5	50
DQS1	3.005	3.4	8.1	—	22	—	0.760	2.8	9	3.5	50
DQS2	2.851	3.4	7.8	—	22	—	0.760	2.8	9	3.5	50
DQS3	2.653	3.4	8.1	—	22	—	0.760	2.8	9	3.5	50
DQS4	2.686	3.4	7.8	—	22	—	0.760	2.8	9	3.5	50
DQS5	2.701	3.4	7.8	—	22	—	0.760	2.8	9	3.5	50
DQS6	2.750	3.4	7.8	—	22	—	0.760	2.8	9	3.5	50
DQS7	2.923	3.4	8.1	—	22	—	0.750	2.8	9	3.5	50
DQS8	2.536	3.4	7.8	—	22	—	0.940	2.8	9	3.5	50

Note to Table 4–1:

(1) Addr = A, BA, WE#, RAS#, ODT, and CAS#.

Altera recommends you use the **Board Trace Model** assignment on all DDR and DDR2 SDRAM interface signals. To apply board trace model assignments for the Stratix III development board, run the Altera-provided **S3_Host_DDR2_BTModels.tcl** file or manually assign virtual pin assignments using the Quartus II Pin Planner.

The Stratix III development board has the following compensation capacitors fitted to its DDR2 SDRAM address and command, and CLK and CLK# signals:

- Address and command = 33 pF compensation capacitors
- CLK and CLK# = 7 pF (differential) compensation capacitors.

These capacitors are typically fitted to designs that use asymmetric DIMM designs. You should simulate your design to see if compensation capacitors are required. Stratix III devices have various programmable drive strength and OCT I/O options, so compensation capacitors should not usually be required. Fitting compensation capacitors reduces the edge rate of your signals, so you should observe memory vendor derating guidelines.

For more information on compensation capacitors, refer to *Micron Technical Note TN_47_01*.

Compile Design and Verify Timing

To compile the design, on the Processing menu, click **Start Compilation**.

Figure 4–8 shows the timing margin report in the message window in the Quartus II software.

Figure 4–8. Timing Margin Report in the Quartus II Software

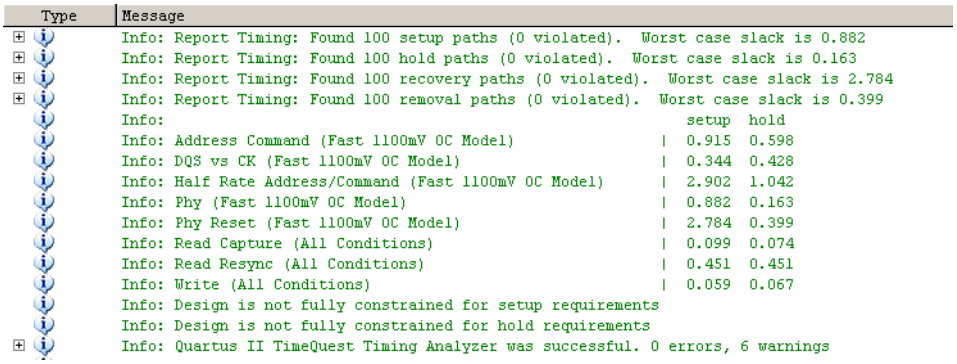
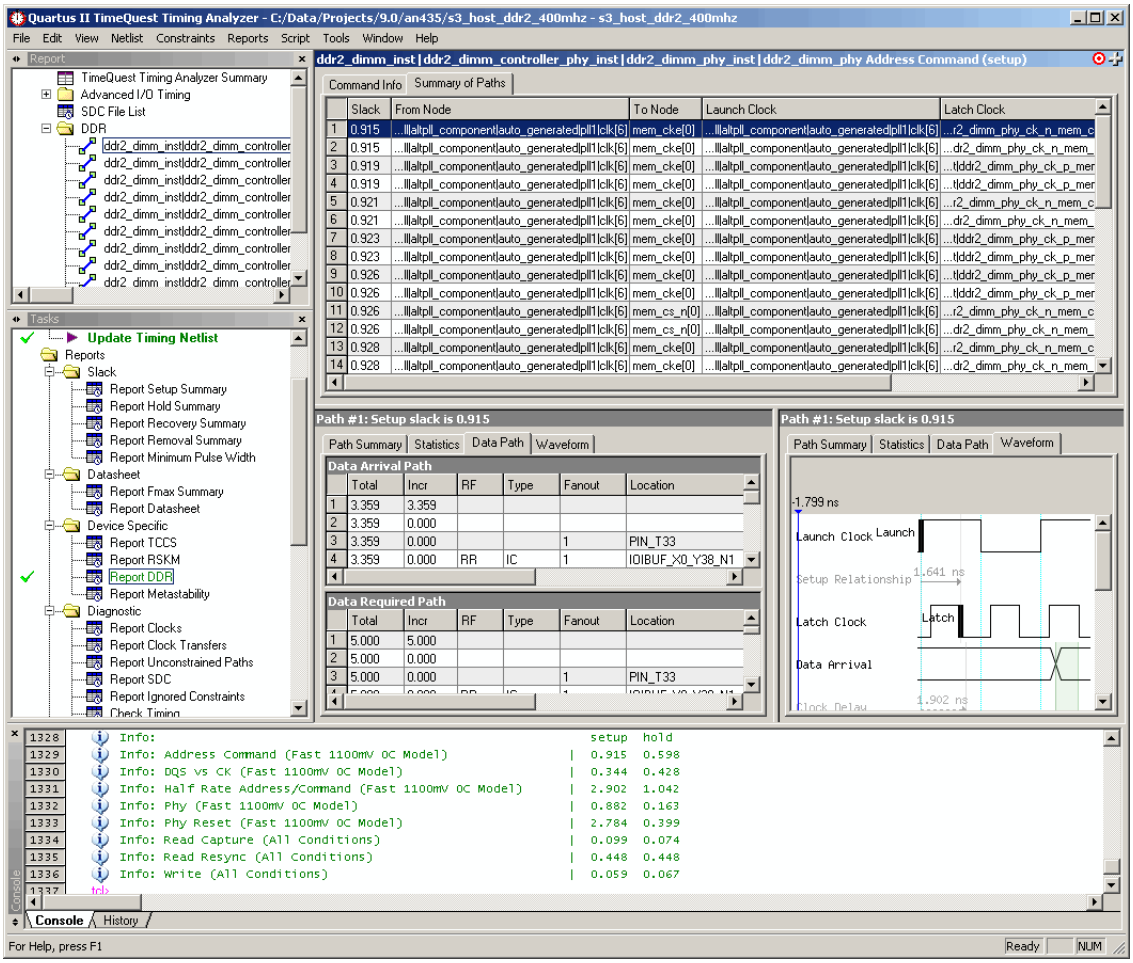


Figure 4–9 on page 4–17 shows the timing margin report in the TimeQuest Timing Analyzer window after running the Report DDR Task. The results are the same as the Quartus II software results.

Figure 4–9. Timing Margin Report in TimeQuest Timing Analyzer



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

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

Adjust Constraints

As the design meets timing, there is no need to adjust any constraints.

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

Determine Board Design Constraints and Perform Board-Level Simulations

Stratix III devices support both series and parallel OCT resistors to improve signal integrity. Another benefit of the Stratix III OCT resistors is eliminating the need for external termination resistors on the FPGA side. This feature simplifies board design and reduces overall board cost. You can dynamically switch between the series and parallel OCT resistor depending on whether the Stratix III devices are performing a write or a read operation. The OCT features offer user-mode calibration to compensate for any variation in voltage and temperature during normal operation to ensure that the OCT values remain constant. The parallel and series OCT features of the Stratix III devices are available in either a 25-Ω or 50-Ω setting.

Refer to the *Stratix III Device I/O Features* chapter of the *Stratix III Device Handbook*, for more information about the OCT features.

Refer to the respective memory data sheet, for more information about the available settings of the ODT and the output driver impedance features, and the timing requirements for driving the ODT pin in DDR2 SDRAM.

Figure 4–10 illustrates the write operation to the DDR2 SDRAM with the ODT feature turned on and using the 50-Ω series OCT feature of the Stratix III FPGA device. In this setup, the transmitter (FPGA) is properly terminated with matching impedance to the transmission line, thus eliminating any ringing or reflection. The receiver (DDR2 SDRAM) is also properly terminated when the dynamic ODT setting is at 75 Ω

Figure 4–10. Write Operation Using Parallel ODT and 50-Ω Series OCT of the Stratix III FPGA Device

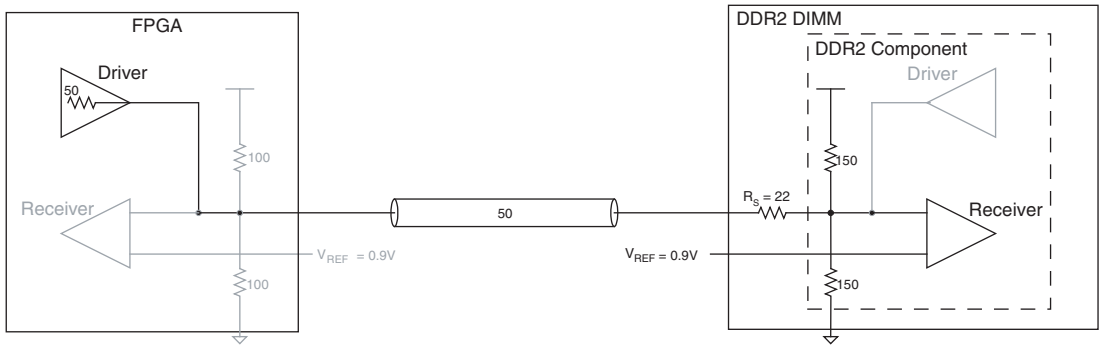
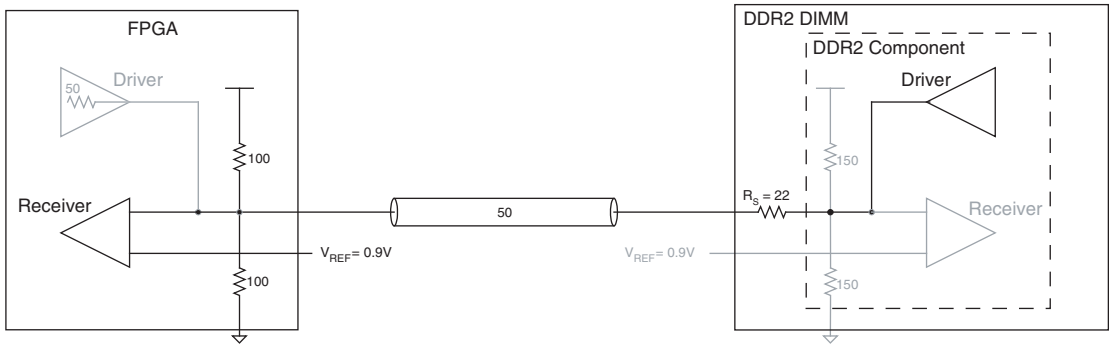


Figure 4–11 illustrates the read operation from the DDR2 SDRAM using the parallel OCT feature of the Stratix III device. In this setup, the driver's (DDR2 SDRAM) output impedance is not larger than 21 Ω. This impedance is in keeping with SSTL-18 JEDEC specification JESD79-2. Combined with an on dual-inline memory modules (DIMM) series resistor, the impedance matches that of the transmission line resulting in optimal signal transmission to the receiver (FPGA). On the receiver (FPGA) side, it is properly terminated with 50-Ω, which matches the impedance of the transmission line, thus eliminating any ringing or reflection.

Figure 4–11. Read Operation From DDR2 SDRAM Using the Parallel OCT Feature of the Stratix III FPGA Device



Finally, the loading seen by the FPGA during writes to the memory is different between a system using DIMMs versus a system using components. The additional loading from the DIMM connector can reduce the edge rates of the signals arriving at the memory thus affecting available timing margin.

For more information about Stratix III devices signal integrity, refer to the Stratix III Device Signal and Power Integrity page.

Adjust Termination and Drive Strength

Due to the loading of the line, the Quartus II software may report that the default or chosen drive strength cannot drive the line to the specified toggle rate or minimum pulse width. If you encounter this error, use the stronger drive strength I/O standard. Ensure that you re-simulate your design with the new drive strength to ensure that signal quality is still acceptable.

Verify Design on a Board

The SignalTap II Embedded Logic Analyzer shows read and write activity in the system.

For more information on using the SignalTap II Embedded Logic Analyzer, refer to the *Design Debugging Using the SignalTap II Embedded Logic Analyzer* chapter in the *Quartus II Handbook*, *AN 323: Using SignalTap II Embedded Logic Analyzers in SOPC Builder Systems* and *AN 446: Debugging Nios II Systems with the SignalTap II Logic Analyzer*.

To add the SignalTap II Embedded Logic Analyzer, perform the following steps:

For this design, Altera provides a Tcl file, **S3_Host_DDR2_SignalTap.tcl**, to automate the following steps. The **.stp** file is included with the design example file.

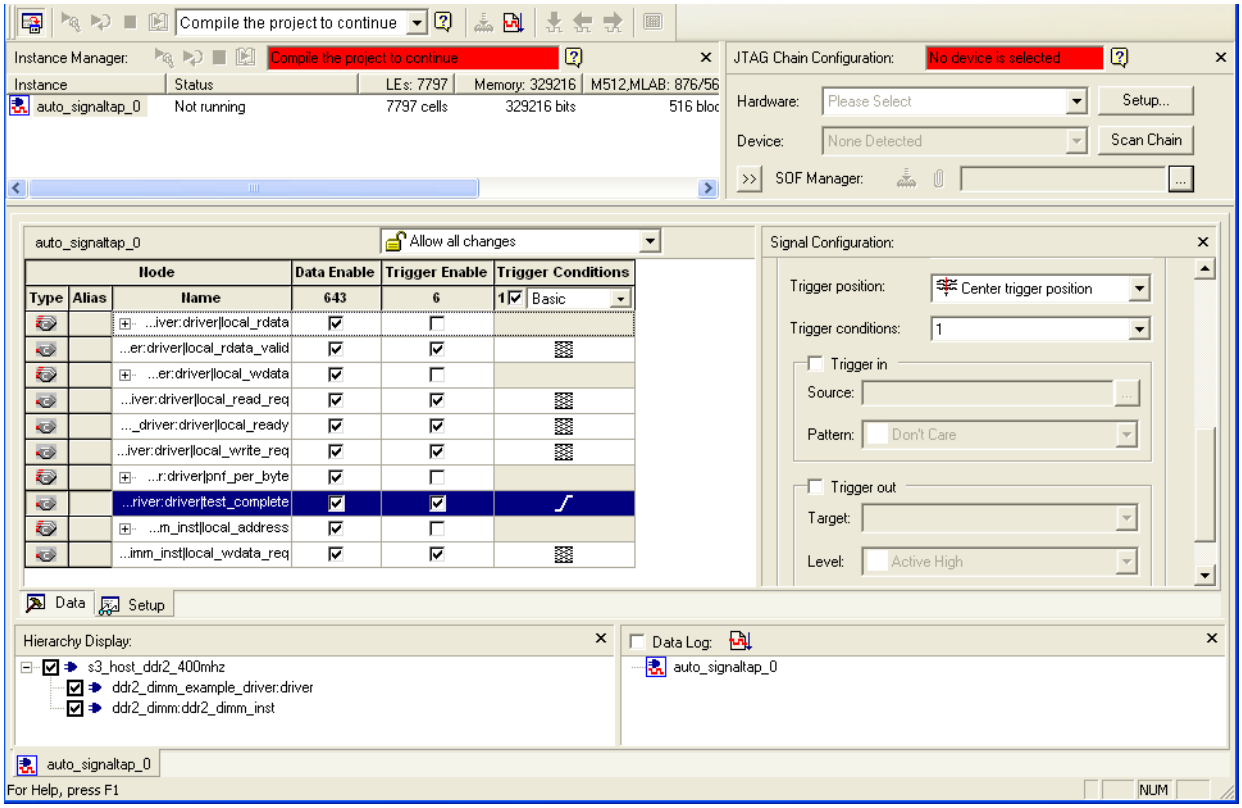
1. On the Tools menu, click **SignalTap II Logic Analyzer**.
2. In the **Signal Configuration** window next to the **Clock** box, click **…** (**Browse Node Finder**).
3. Type `*phy_clk` in the **Named** box, for **Filter** select **SignalTap II: pre-synthesis** and click **List**.
4. Select **ddr2_dimm_example_top | ddr2_dimm:ddr2_dimm_inst | ddr2_dimm_controller_phy:ddr2_dimm_controller_phy_inst | phy_clk | phy_clk** in **Nodes Found** and click **>** to add the signal to **Selected Nodes**.
5. Click **OK**.
6. Under **Signal Configuration**, specify the following settings:
   - For **Sample depth**, select **512**
   - For **RAM type**, select **Auto**
   - For **Trigger flow control**, select **Sequential**
   - For **Trigger position**, select **Center trigger position**
   - For **Trigger conditions**, select **1**
7. On the Edit menu, click **Add Nodes**.
8. Search for specific nodes by typing `*local*` in the **Named** box, for **Filter** select **SignalTap II: pre-synthesis** and click **List**.
9. Select the following nodes in **Nodes Found** and click **>** to add to **Selected Nodes**:
   - **local_address**
   - **local_rdata**
   - **local_rdata_valid**
   - **local_read_req**
   - **local_ready**
   - **local_wdata**
   - **local_wdata_req**
   - **local_write_req**
   - **pnf**
   - **pnf_per_byte**
   - **test_complete** (trigger)

   Do not add any DDR SDRAM interface signals to the SignalTap II Embedded Logic Analyzer. The load on these signals increases and adversely affects the timing analysis.
10. Click **OK**.

11. To reduce the SignalTap II logic size, turn off **Trigger Enable** on the following bus signals:
    - **local_address**
    - **local_rdata**
    - **local_wdata**
    - **pnf_per_byte**
12. Right-click **Trigger Conditions** for the `test_complete` signal and select **Rising Edge**.

    Figure 4–12 shows the completed SignalTap II Embedded Logic Analyzer.

Figure 4–12. SignalTap II Embedded Logic Analyzer



13. On the File menu, click **Save**, to save the SignalTap II **.stp** file to your project.

    If you see the message **Do you want to enable SignalTap II file "stp1.stp" for the current project**, click **Yes**.

Compile the Project

Once you add signals to the SignalTap II Embedded Logic Analyzer, recompile your design, on the Processing menu, click **Start Compilation**.

Verify Timing

Once the design compiles, ensure that TimeQuest timing analysis passes successfully. In addition to this FPGA timing analysis, check your PCB or system SDRAM timing. To run timing analysis, run the <*variation name*>**_phy_report_timing.tcl** script.

1. On the Tools menu, click **Tcl Scripts**.
2. Select <*variation name*>**_phy_report_timing.tcl** and click **Run**.

Download the Object File

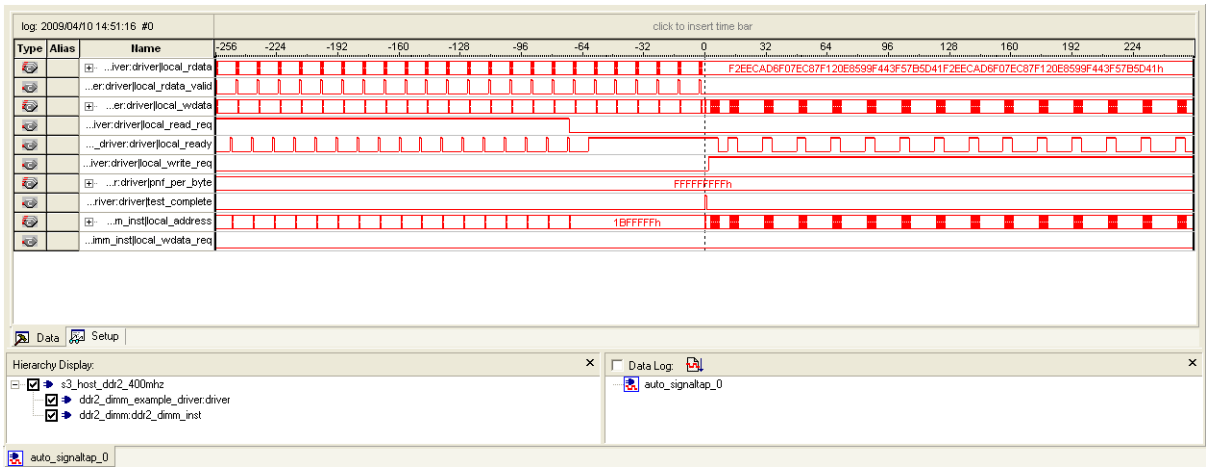
To download the object file to the device, perform the following steps:

1. Connect the development board to your computer.
2. On the Tools menu, click **SignalTap II Logic Analyzer**. The SignalTap II dialog box appears.
3. Click **...** to open the **Select Program Files** dialog box.
4. Select <*your project name*>**.sof**.
5. Click **Open**.
6. To download the file, click the **Program Device** button.

Test the Design Example in Hardware

When the design example including SignalTap II successfully downloads to your development board, click **Run Analysis** to run once, or click **Autorun Analysis** to run continuously. Figure 4–13 shows the design analysis.

Figure 4–13. SignalTap II Example DDR2 SDRAM Design Analysis



5. Using DDR3 SDRAM Devices in Stratix III and Stratix IV Devices

This tutorial describes how to use the design flow to implement a 64-bit wide, 533-MHz, 1,066-Mbps DDR3 SDRAM interface with Stratix III devices, and a 72-bit wide, 400-MHz, 800-Mbps DDR3 SDRAM interface with Stratix IV devices. The design examples also provide some recommended settings to simplify the design.

The design examples target the Stratix III memory demonstration kit, which includes a 72-bit wide 1-GB Micron MT9JSF12872AY-1G1BZES 533-MHz DDR3 SDRAM DIMM, and the Stratix IV GX FPGA development kit, which includes four 16-bit wide 1-GB Micron MT41J64M16LA-187E 533-MHz DDR3 SDRAM devices.

To download the design examples, **emi_ddr3_siii.zip** and **emi_ddr3_siv.zip**, go to the External Memory Interface Design Examples page. You can also use this design example if you are targeting a HardCopy device for your design. For ALTMEMPHY megafunction supported devices, refer to mmmm section in volume 3 of the *External Memory Interface Handbook*.

The Stratix III memory demonstration kit is not available for purchase. The early versions of the Stratix III memory demonstration kit include a MT16JTF25664AY-1G1D1 DDR3 SDRAM DIMM, which is a dual-rank DIMM and is not supported by ALTMEMPHY-based solutions. Make sure that the MT9JSF12872AY-1G1BZES DDR3 SDRAM DIMM is fitted.

For more information about the design flow, refer to the *Recommended Design Flow* section in volume 1 of the *External Memory Interface Handbook*.

System Requirements

This tutorial requires the following hardware and software for the DDR3 SDRAM interface with a Stratix III device:

- Quartus II software version 9.1
- DDR3 SDRAM Controller with ALTMEMPHY IP version 9.1
- Stratix III memory demonstration kit, with Micron MT9JSF12872AY-1G1BZES device

This tutorial requires the following hardware and software for the DDR3 SDRAM interface with a Stratix IV device:

- Quartus II software version 9.1
- DDR3 SDRAM Controller with ALTMEMPHY IP version 9.1
- Stratix IV GX FPGA development kit, with Micron MT41J64M16LA-187E device

Create a Quartus II Project

Create a project in the Quartus II software that targets the appropriate device; the EP3SL150F1152-C2ES device for the Stratix III device family or the EP4SGX230KF40C3ES device for the Stratix IV device family.

For step-by-step instructions to create a Quartus II project, refer to the Quartus II software tutorial by clicking the Help menu in the Quartus II window and selecting **PDF Tutorials**.

Instantiate and Parameterize a Controller

After you create a Quartus II project, you must instantiate a controller and its parameters.

Instantiate a Controller

To instantiate a controller, perform the following steps:

- DDR3 SDRAM controller with a Stratix III device:
  a. Copy the memory parameters files, **S3MB1_Derated (Micron MT9JSF12872AY-1G1BZES).xml**, to your <*installation directory*>**\91\ip\ddr3_high_perf\lib** directory.
  b. Start the MegaWizard™ Plug-In Manager.
  c. In the MegaWizard Plug-In Manager, expand **External Memory** in the **Interfaces** folder and select the **DDR3 SDRAM High Performance Controller**.
  d. Type `ddr3_dimm` for the name of the DDR3 SDRAM high-performance controller.
- DDR3 SDRAM controller with a Stratix IV device:
  a. Copy the memory parameters files, **SD_PCIe_DDR3_Kit (4xMicron MT41J64M16LA-187E).xml**, to your <*installation directory*>**\91\ip\ddr3_high_perf\lib** directory.
  b. Start the MegaWizard Plug-In Manager.
  c. In the MegaWizard Plug-In Manager, expand **External Memory** in the **Interfaces** folder and select the **DDR3 SDRAM High Performance Controller**.
  d. Type `ddr3_bot_x64` for the name of the DDR3 SDRAM high-performance controller.

Both design examples instantiate the ALTMEMPHY megafunction automatically.

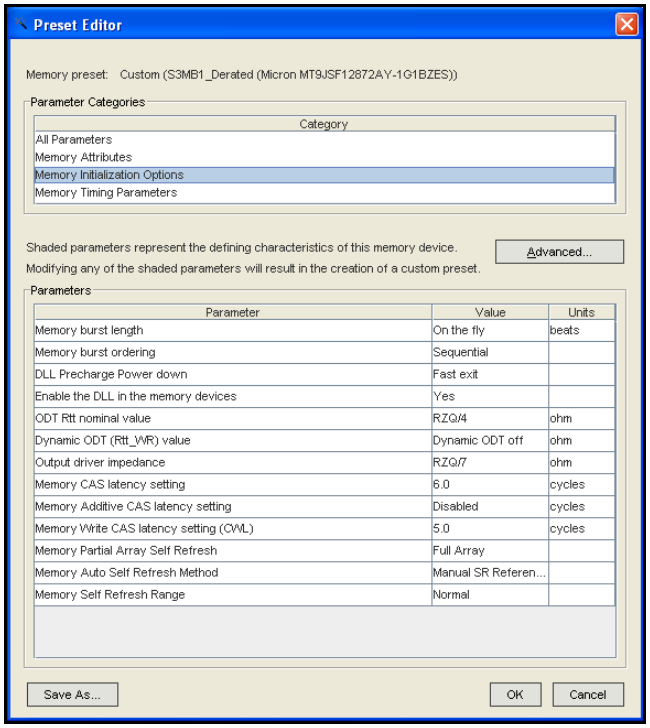
Parameterize DDR3 SDRAM with Stratix III

To parameterize the DDR3 SDRAM high-performance controller to interface with a 533-MHz, 72-bit wide DDR3 SDRAM interface, perform the following steps:

1. In the **Memory Setting** tab, set **Speed grade** to **2**.
2. For **PLL reference clock frequency**, type **100** MHz (to match the on-board oscillator).
3. For **Memory clock frequency**, type **533** MHz (the maximum frequency supported for DDR3 SDRAM interfaces on Stratix III devices).
4. For **Memory Presets**, select **S3MB1_Derated (Micron MT9JSF12872AY-1G1BZES)**, which gives a 72-bit wide, 1,152-Mbps 533-MHz DDR3 unbuffered DIMM.

5. To create or modify a memory preset, click **Modify parameters**. In the **Preset Editor** dialog box, you can modify the memory presets. Refer to Figure 5–1.

Figure 5–1. Modify the Memory Presets to Create a Custom Memory



The t_{AC} and t_{QHS} parameters are often not defined by memory vendors, as these values are only used in static RTD calculations and non-DQS capture mode. The wizard does not require these parameters, so use the default values.

The t_{IS}, t_{IH}, t_{DS}, and t_{DH} parameters typically require slew rate derating.

Simulation and measurement show the following slew rate for the clock, address and command, and DQ and DQS pins on the Stratix III memory demonstration board when using the default I/O standard and drive options:

- Address and command = 1.5 V/ns
- CLK and CLK# = 3 V/ns (differential)
- DQ = 2 V/ns
- DQS = 3 V/ns (differential)

Hence, the correct t_{IS}, t_{IH}, t_{DS}, and t_{DH} values for this design are:

- $t_{IS} = t_{ISb} + \Delta t_{IS} + (V_{IHAC} - V_{REF})$/address and command rising slew rate
  $=125 + 59 + 175/1.5 = 301$ ps
- $t_{IH} = t_{IHb} + \Delta t_{IH} + (V_{REF} - V_{ILDC})$/address and command rising slew rate
  $= 200 + 34 + 100/1.5 = 301$ ps
- $t_{DS} = t_{DSa} + \Delta t_{DS} + (V_{IHAC} - V_{REF})$/DQ rising slew rate
  $= 25 + 88 + 175/2 = 201$ ps
- $t_{DH} = t_{DHa} + \Delta t_{DH} + (V_{REF} - V_{ILDC})$/DQ rising slew rate

= 100 + 50 + 100/2 = 200 ps

For more information about how to derate these numbers, refer to *Derate Memory Setup and Hold Timing* in the *DDR3 SDRAM Controller with ALTMEMPHY IP User Guide* in volume 3 of the *External Memory Interface Handbook*.

The DDR3 SDRAM has a write requirement (t_{DQSS}) that states the positive edge of the DQS signal on writes must be within ±25% (±90°) of the positive edge of the DDR3 SDRAM clock input. To achieve this skew requirement, the ALTMEMPHY-based designs always use the DDR IOE registers to generate the CK and CK# signals.

6. To set the ODT settings for the DDR3 SDRAM interface on your board, in the **Preset Editor** dialog box, select **Memory Initialization Options**.
7. In the **Memory Initialization Options** dialog box, perform the following steps:
   a. For **Output driver impedance**, select **RZQ/7** (which is 34).
   b. For **Dynamic ODT (Rtt_WR) value**, select **Dynamic ODT off**.
   c. For **ODT Rtt nominal value**, select **RZQ/4** (which is 60).
   d. Click **OK** to apply the settings and exit the dialog box.
8. In the **PHY Settings** tab, under **Advanced PHY Settings** turn on the **Enable dynamic parallel on-chip termination (OCT)** option for this example.
9. Under **Address/Command Clock Setting**s, for **Dedicated clock phase** type **240**.
10. Under **Board Timing Parameters**, for **Board skew** type **20 ps**. This timing parameter is the board trace variation between the CK, CK#, CAC, DQ, DQS, and DQS# pins. If your board can perform better or worse than this value, update it accordingly.

The **Auto-Calibration Simulation Options** parameter is for RTL simulation only and is not applicable for gate-level simulation.

11. In the **Controller Settings** tab, for **Controller Architecture** select **High Performance Controller II** for higher efficiency and advanced features.
12. Under **Efficiency**, select the specified values for the following options:
    a. For **Command Queue Look-Ahead Depth**, select **6**.
    b. For **Local-to-Memory Address Mapping**, select **CHIP-ROW-BANK-COL**.
    c. For **Local Maximum Burst Count**, select **4**.
13. Click **Next**.
14. Turn on the **Generate simulation model** option to generate the simulation model for the RTL simulation.
15. Click **Finish** to generate your MegaCore function variation. The MegaWizard Plug-In Manager generates all the files necessary for your DDR3 SDRAM controller, and generates an example top-level design, which you use to test or verify board operation.

Parameterize DDR3 SDRAM with Stratix IV

To parameterize the DDR3 SDRAM high-performance controller to interface with a 400-MHz, 64-bit wide DDR3 SDRAM interface, perform the following steps:

1. In the **Memory Setting** tab, set **Speed grade** to 3.
2. For **PLL reference clock frequency**, type **100** MHz to match the on-board oscillator.
3. For **Memory clock frequency**, type **400** MHz, the maximum frequency supported for DDR3 SDRAM interfaces on Stratix IV devices.
4. For **Memory Presets**, select **SD_PCIe_DDR3_Kit (4xMicron MT41J64M16LA-187E)**, which is a 64-bit wide 512-MB 400-MHz DDR3 unbuffered DIMM.

   The memory format in the **SD_PCIe_DDR3_Kit (4xMicron MT41J64M16LA-187E).xml** file is changed from discrete device to unbuffered DIMM to enable leveling functionality.

5. To create or modify a memory preset, click **Modify parameters**. In the **Preset Editor** dialog box, you can modify the memory presets.

   The DDR3 SDRAM has a write requirement (t_{DQSS}) that states the positive edge of the DQS signal on writes must be within ±25% (±90°) of the positive edge of the DDR3 SDRAM clock input. To achieve this skew requirement, the ALTMEMPHY-based designs always use the DDR IOE registers to generate the CK and CK# signals.

6. Turn on the **Enable Memory Chip Calibration in Timing Analysis** option on the **Advanced** page. This option is required for Stratix IV devices, which need post-processing script to remove timing model pessimism.
7. To specify the ODT settings for the DDR3 SDRAM interface on your board, in the **Preset Editor** dialog box, select **Memory Initialization Options**.
8. In the **Memory Initialization Options** dialog box, perform the following steps:
   a. For **Output driver impedance**, select **RZQ/7** (which is 34).
   b. For **Dynamic ODT (Rtt_WR) value**, select **RZQ/4** (which is 60).
   c. For **ODT Rtt nominal value**, select **RZQ/4** (which is 60).
   d. Click **OK** to apply the settings and exit the dialog box.
9. In the **PHY Settings** tab, turn on the **Enable dynamic parallel on-chip termination (OCT)** option for this example. The Stratix IV GX FPGA development kit does not include discrete external termination on the DQ, DQS, DQS#, or DM pins as the board was designed to use OCT.
10. Under **Address/Command Clock Setting**s, for **Dedicated clock phase** type **240**.

    The **Auto-Calibration Simulation Options** parameter is for RTL simulation only and is not applicable for gate-level simulation.

11. In the **Board Settings** tab, set the following **Slew Rates** and **Board Skews** parameters to the specified values:
    - **CK/CK# slew rate (Differential)** = **4 V/ns**
    - **Addr/Command slew rate** = **1.5 V/ns**
    - **DQS/DQS# slew rate (Differential)** = **3 V/ns**
    - **DQ slew rate** = **1.5 V/ns**

    
    These slew rates are obtained from simulation using the default I/O standard and drive options.

    - **Max skew within DQS group** = **0.015 ns**
    - **Max skew between DQS groups** = **0.128 ns**
    - **Addr/Command to CK skew** = **- 0.05 ns**

    The Intersymbol Interference (ISI) parameters are not applicable for single rank configurations. Set these parameters to **0 ns**.
12. In the **Controller Settings** tab, for **Controller Architecture** select **High Performance Controller II** for higher efficiency and advanced features.
13. Under **Efficiency**, select the specified values for the following options:
    a. For **Command Queue Look-Ahead Depth**, select **6**.
    b. For **Local-to-Memory Address Mapping**, select **CHIP-ROW-BANK-COL**.
    c. For **Local Maximum Burst Count**, select **4**.
14. Click **Next**.
15. Turn on the **Generate simulation model** option to generate the simulation model for the RTL simulation.
16. Click **Finish** to generate your MegaCore function variation. The MegaWizard Plug-In Manager generates all the files necessary for your DDR3 SDRAM controller, and an example top-level design, which you use to test or verify board operation.

For more information about the DDR3 SDRAM high-performance controller, refer to the *DDR3 SDRAM Controller with ALTMEMPHY IP User Guide* in volume 3 of the *External Memory Interface Handbook*.

Add Constraints

After instantiating the DDR3 SDRAM high-performance controller, the ALTMEMPHY megafunction generates the constraints files for the design example. You must apply these constraints to the design before compilation.

Set Top-Level Entity

The top-level entity of the project must be set to the correct entity. The naming convention for an ALTMEMPHY megafunction entity is <*variation_name*>**_phy.v** or **vhd**; an SDRAM high-performance controller entity is <*variation_name*>**.v** or **vhd**.

To set the top-level file, perform the following steps:

1. Open the entity file, <*variation_name*>**_example_top.v** or **vhd**.
2. On the Project menu, click **Set as Top-Level Entity**.

Set Optimization Technique

To ensure the remaining unconstrained paths are routed with the highest speed and efficiency, perform the following steps to set the optimization technique:

1. On the Assignments menu, click **Settings**.
2. Select **Analysis & Synthesis Settings**.
3. Select **Speed** under **Optimization Technique**. Click **OK**.

Set Fitter Effort

To set the fitter effort, perform the following steps:

1. On the Assignments menu, click **Settings**.
2. Select **Fitter Settings**.
3. Turn on **Optimize hold timing** and select **All Paths**.
4. Turn on **Optimize multi-corner timing**.
5. Select **Standard Fit (highest effort)** under **Fitter effort**.
6. Click **OK**.

Add Timing Constraints

When you instantiate an SDRAM high-performance controller, it generates a timing constraints file, <*variation_name*>**_phy_ddr_timing.sdc**. The timing constraint file constrains the clock, input, and output delay on the SDRAM high-performance controller.

To add timing constraints, perform the following steps:

1. On the Assignments menu click **Settings**.
2. In the **Category** list, expand **Timing Analysis Settings**, and select **TimeQuest Timing Analyzer**.
3. Select the <*variation_name*>**_phy_ddr_timing.sdc** file and click **Add**.
4. Click **OK**.

Add Pin and DQ Group Assignments

The pin assignment script, <*variation_name*>**_pin_assignments.tcl**, sets up the I/O standards for the DDR3 SDRAM interface. This script also launches the DQ group assignment script, <*variation_name*>**_phy_assign_dq_groups.tcl**, which relates the `DQ` and `DQS` pin groups together for the fitter to place them correctly in the Quartus II software.

This script does not create a clock for the design. You must create a clock for the design and provide pin assignments for the signals of both the example driver and testbench that the MegaCore variation generates.

To add the pin and I/O standards to the design example, perform the following steps:

1. On the Tools menu, click **Tcl scripts**.
2. Under **Libraries**, select <*variation_name*>**_pin_assignments.tcl**.
3. Click **Run**.

Enter Pin Location Assignments

To enter the pin location assignments using the Pin Planner, perform the following steps:

1. Run analysis and synthesis. On the Processing menu, point to **Start** and click **Start Analysis & Synthesis**.
2. Assign all of your pins, so the Quartus II software fits your design correctly and performs correct timing analysis. To assign pin locations for the Stratix III memory demonstration kit, run the Altera-provided **S3_MB1_DDR3_PinLocations.tcl** file, and for the Stratix IV GX FPGA development kit, run the Altera-provided **SIV_DDR3×64_PinLocations.tcl** file, or manually assign the pin locations by using the Pin Planner.

   The SDRAM high-performance controller auto-generated scripts do not make any pin location assignments.

   If you are at the design exploration phase of your design cycle and do not have any PCB defined pin locations, you should manually define an initial set of pin constraints, which can become more specific during your development process.

To manually assign pin locations using the Pin Planner, perform the following steps:

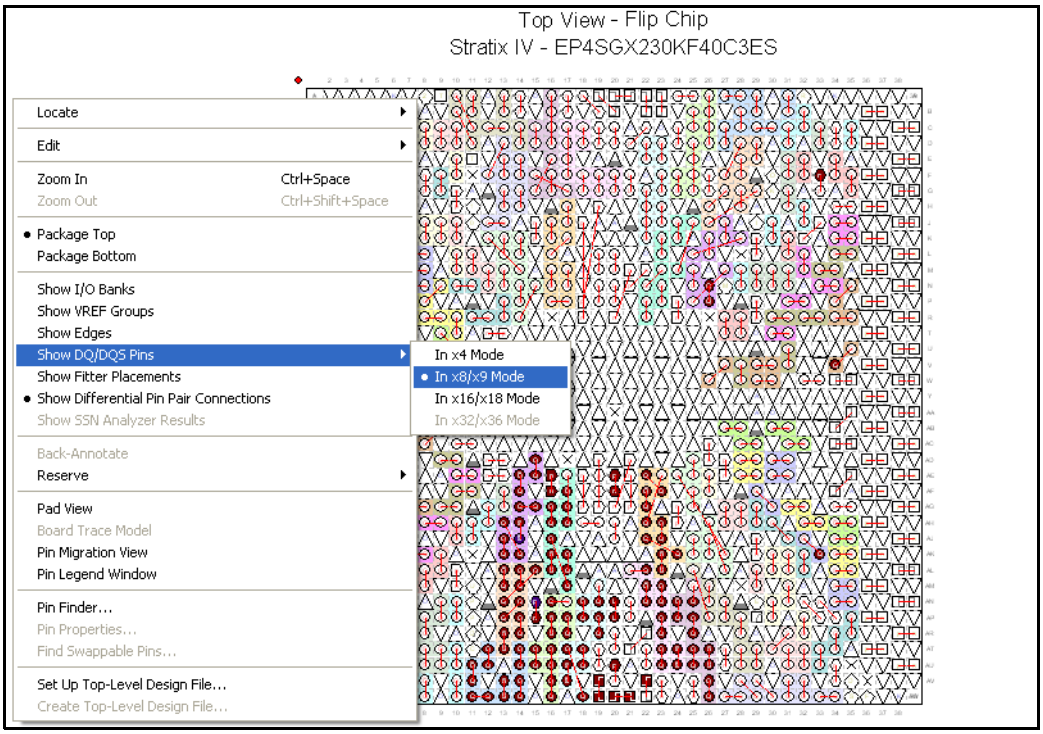
1. On the Assignments menu, click **Pin Planner**.
2. Assign `DQ` and `DQS` pins.
   a. To select the device DQS pin groups that the design uses, assign each `DQS` pin in your design to the required `DQS` pin in the Pin Planner. The Quartus II Fitter automatically assigns the respective `DQ` signals to suitable `DQ` pins within each group. To view the DQS groups in the Pin Planner, right-click, select **Show DQ/DQS Pins**, and click **In ×8/×9 Mode**. The Pin Planner shows each DQS group in a different color and with a different legend: S = `DQS` pin, Sbar = `DQSn` pin and Q = `DQ` pin, as shown in Figure 5–2.

   Most DDR3 SDRAM devices operate in ×8/×9 mode. However, some DDR3 SDRAM devices operate in ×4 mode. Refer to your specific memory device datasheet.

   b. Select the DQ mode to match the DQ group width (number of `DQ` pins/number of `DQS` pins) of your memory device. DQ mode is not related to the memory interface width.

DQ group order and DQ pin order within each group is not important. However, you must place the DQ pins in the same group as their respective strobe pin.

Figure 5–2. Quartus II Pin Planner, Show DQ/DQS Pins, In x8/x9 Mode for Stratix IV



3. Place the DM pins within their respective DQ group.
4. Place the address and control command pins on any spare I/O pins; ideally within the same bank or side of the device as the mem_clk pins.
5. Ensure that you place the mem_clk pins on differential I/O pairs for the CK/CK# pin pair. To identify differential I/O pairs, right-click in the Pin Planner and select **Show Differential Pin Pair Connections**. The pin pairs show a red line between each pin pair.

   You must place the mem_clk[0] and mem_clk_n[0] pins on an unused DQ or DQS pin pairs with DIFFIO_RX capability.

6. Place the clock_source pin on a dedicated PLL clock input pin with a direct connection to the SDRAM controller PLL and DLL pair—usually on the same side of the device as your memory interface. This recommendation reduces PLL jitter, saves a global clock resource, and eases timing and fitter effort.
7. Place the global_reset_n pin (like any high fan-out signal) on a dedicated clock pin.
8. Ensure that you place the R_{UP} and R_{DN} pins, termination_blk0~_rdn_pad and termination_blk0~_rup_pad, at locations within the same V_{CCIO} voltage bank.

For more information about the Quartus II Pin Planner, refer to the *I/O Management* chapter in volume 2 of the *Quartus II Handbook*.

Assign I/O Standards

To assign the I/O standards, perform the following steps:

1. On the Assignments menu, click **Assignment Editor**.
2. Specify **LVDS** as the I/O standard for `clock_source`.
3. Specify **2.5** V as the I/O standard for `global_reset_n`.

Assign Virtual Pins

The example top-level design, which is auto-generated by the high- performance controller, includes an example driver to simulate the interface. This example driver is not part of the SDRAM high-performance controller IP, but allows easy testing of the IP.

The example driver outputs several test signals to indicate its operation and the status of the simulated memory interface. These test signals are `pnf`, `pnf_per_byte`, and `test_complete`. The test signals are not part of the memory interface, but are to facilitate testing. You must connect these signals to either a debug bus or assign the signals to virtual pins using the Quartus II Assignment Editor. When using the example driver for testing, do not remove the test signals from the top-level signal list. If you remove the test signals from the top-level module, the Quartus II software optimizes the driver away, and the example driver fails.

To assign virtual pin assignments for the Stratix III memory demonstration board, run the Altera-provided **s3_MB1_ddr3_exdriver_vpin.tcl** file, and for the Stratix IV GX FPGA development kit, run the Altera-provided **SIV_DDR3×64_exdriver_vpin.tcl** file, or manually assign the virtual pin assignments using the Assignment Editor.

The memory interface pins (`DQ`, `DQS`, `DM`, `CK`, `CK#`, address and command) cannot be assigned as virtual pins.

Advanced I/O Timing

The ALTMEMPHY-based Stratix III designs assume that the memory address and command signals are length-matched to the memory clock signals. Typically, this length match is not true for DIMM-based designs. You must verify the difference in your design. To edit the TimeQuest **.sdc** file, <*variation name*>**_phy_ddr_timing.sdc**, to include this difference, perform the following steps:

1. Open the **ddr3_dimm_phy_ddr_timing.sdc** file in a text editor and find the following command line (usually line 31):

```
set t(additional_addresscmd_tpd) 0.000
```

   and change to the following command line:

```
set t(additional_addresscmd_tpd) -0.300
```

2. Save the file.

If the DDR3 SDRAM controller **.sdc** file is regenerated, this change is lost and you must edit the file again.

Enter Board Trace Delay Models

For accurate I/O timing analysis, the Quartus II software must include the board trace and loading information. This information should be derived and refined during your PCB development process of prelayout (line) simulation and final post-layout (board) simulation.

For external memory interfaces that use memory modules (DIMMs), the board trace and loading information must also include the trace and loading information of the module. You can obtain these information from your memory vendor.

To enter board trace information, perform the following steps:

1. In the Pin Planner, select the pin or group of pins that you want to enter the information for.
2. Right-click and select **Board Trace Model**. Figure 5–3 shows the board trace model.

Figure 5–3. Board Trace Model for Stratix IV

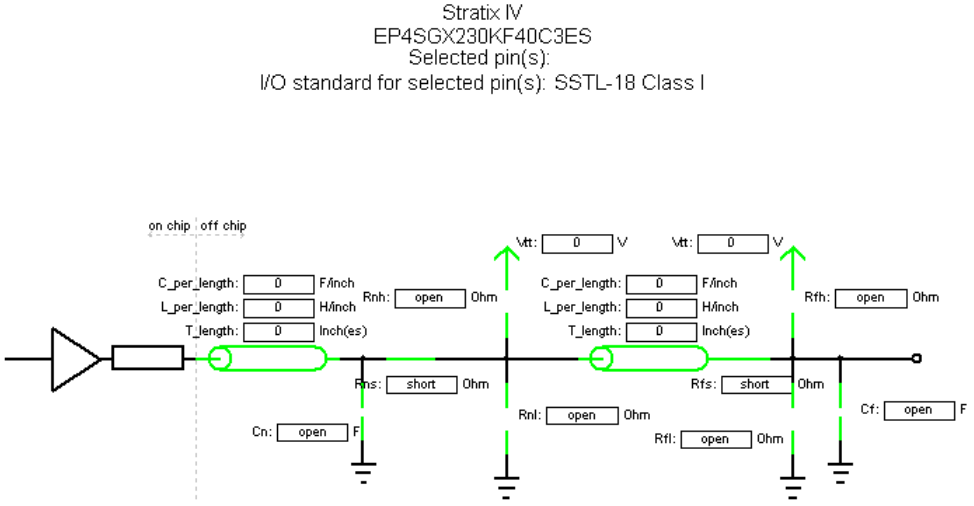


Table 5–1 shows the board trace model parameters for the Stratix III and Stratix IV GX development boards.

Table 5–1. Stratix III and Stratix IV Development Board Trace Model Summary

Net	Near (FPGA End of Line)						Far (Memory End of Line)				
	Length (Inch)	C_per_ length (pF/In)	L_per_ length (nH/In)	Cn (pF)	Rns	Rnh	Length (Inch)	C_per_ length (pF/In)	L_per_ length (nH/In)	Cf (pF)	Rfh/Rfp
Stratix III											
Addr *(1)*	2.904	3.5	8.3	—	—	—	8.488	3.75	8.9	13.5	39
CLK	3.069	3.1	9.3	4.6 *(2)*	—	—	8.488	3.75	8.9	7.2	36
CKE/CS#	2.937	3.5	8.3	—	—	—	8.480	3.75	8.9	13.5	39
ODT	2.853	3.5	8.3	—	—	—	8.480	3.75	8.9	13.5	39
DQS0	2.905	3.5	8.3	—	15	—	0.661	3.0	10.7	3.0	60
DQS1	2.973	3.5	8.3	—	15	—	0.780	3.0	10.7	3.0	60
DQS2	2.893	3.5	8.3	—	15	—	0.913	3.0	10.7	3.0	60
DQS3	2.778	3.5	8.3	—	15	—	1.106	3.0	10.7	3.0	60
DQS4	2.877	3.5	8.3	—	15	—	1.051	3.0	10.7	3.0	60
DQS5	2.936	3.5	8.3	—	15	—	0.870	3.0	10.7	3.0	60
DQS6	3.072	3.5	8.3	—	15	—	0.728	3.0	10.7	3.0	60
DQS7	3.080	3.5	8.3	—	15	—	0.665	3.0	10.7	3.0	60
DQS8	2.708	3.5	8.3	—	15	—	0.661	3.0	10.7	3.0	60
Stratix IV											
Addr *(1)*	—	—	—	—	—	—	3.568	4.2	7.6	5.2	56
CLK	—	—	—	—	—	—	3.73	4.1	7.8	5.2	100
CKE/CS#	—	—	—	—	—	—	3.568	4.2	7.6	5.2	56
ODT	—	—	—	—	—	—	3.51	4.2	7.6	5.2	56
DQS	—	—	—	—	—	—	2.3	4.1	7.8	2.5	60

Notes to Table 5–1:

(1) Addr = Addr, ba, we#, ras#, odt, and cas.

(2) Cn value of 4.6 pF comprises 7 pF on memory demonstration board plus 2.2 pF on the DIMM, which is 9.2 pF differential or 4.6 pF SE.

Altera recommends you to use the **Board Trace Model** assignment for all DDR3 SDRAM interface signals. To apply the board trace model assignments for the Stratix III memory demonstration board, run the Altera-provided **S3_MB1_DDR3_BTModels.tcl** script, and for the Stratix IV GX FPGA development kit, run the Altera-provided **SIV_DDR3×64_BoardTraceModels.tcl** script, or manually assign the virtual pin assignments using the Quartus II Pin Planner.

The Stratix III demonstration board has the 7 pF (differential) compensation capacitors fitted to its DDR3 SDRAM CLK and CLK# signals. These capacitors are typically fitted to designs that use asymmetric DIMM designs. Simulate your design to check if the compensation capacitors are required. Stratix III devices have various programmable drive strength and OCT I/O options, so compensation capacitors are not usually required. Because fitting compensation capacitors reduces the edge rate of your signals, you should observe the memory vendor derating guidelines.

For more information on compensation capacitors, refer to *Micron Technical Note TN_47_01*.

Perform RTL or Functional Simulation (Optional)

After instantiating the DDR3 SDRAM high-performance controller, the MegaWizard Plug-In Manager generates a design example that includes a driver, a test bench, and a memory model that allows you to perform functional simulation on your design.

The Verilog HDL or VHDL simulation model of the PHY, <*variation_name*>**_alt_mem_phy_sequencer_wrapper.vo/.vho** file, is located in your project directory.

To run the simulation with NativeLink, perform the following steps:

1. Set the absolute path to your third-party simulator executable file by performing the following steps:
   a. On the Assignments menu, click **EDA Tool Settings**.
   b. In the **Category** list, expand **EDA Tool Settings** and click **Simulation**.
   c. Under **Tool name**, select **ModelSim-Altera**.
   d. Under **NativeLink settings**, select **Compile test bench** and click **Test Benches**.
   e. Click **New**.
   f. Type the name of your testbench top-level module and simulation period.
   g. Click **OK**.
2. To elaborate your design, on the Processing menu, point to **Start** and click **Start Analysis & Elaboration**.
3. On the Tools menu, point to the **Run EDA Simulation Tool** and click **EDA RTL Simulation**. This step creates the **\simulation** directory in your project directory and a script that compiles all necessary files and runs the simulation.

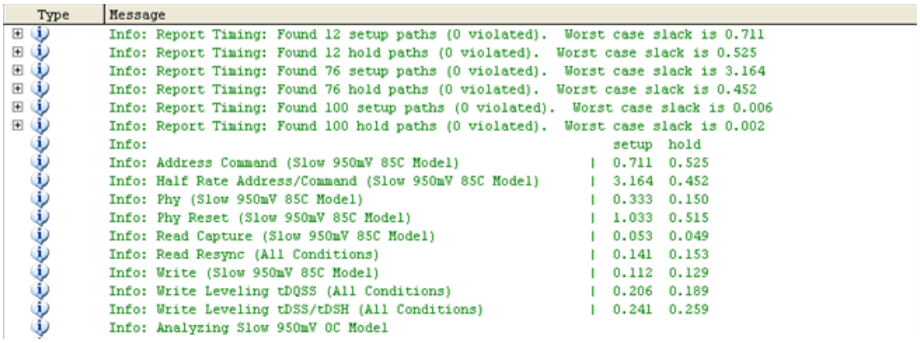
For example timing diagrams, refer to the *DDR3 SDRAM Controller with ALTMEMPHY IP User Guide* in volume 3 of the *External Memory Interface Handbook*.

Compile Design and Verify Timing

To compile the design, on the Processing menu, click **Start Compilation**. After successfully compiling the design, the Quartus II software automatically runs the verify timing script, <*variation_name*>**_timing.tcl**, which produces a timing report for the design together with the compilation report.

Figure 5–4 shows the timing margin report in the message window in the Quartus II software.

Figure 5–4. Timing Margin Report in the Quartus II Software

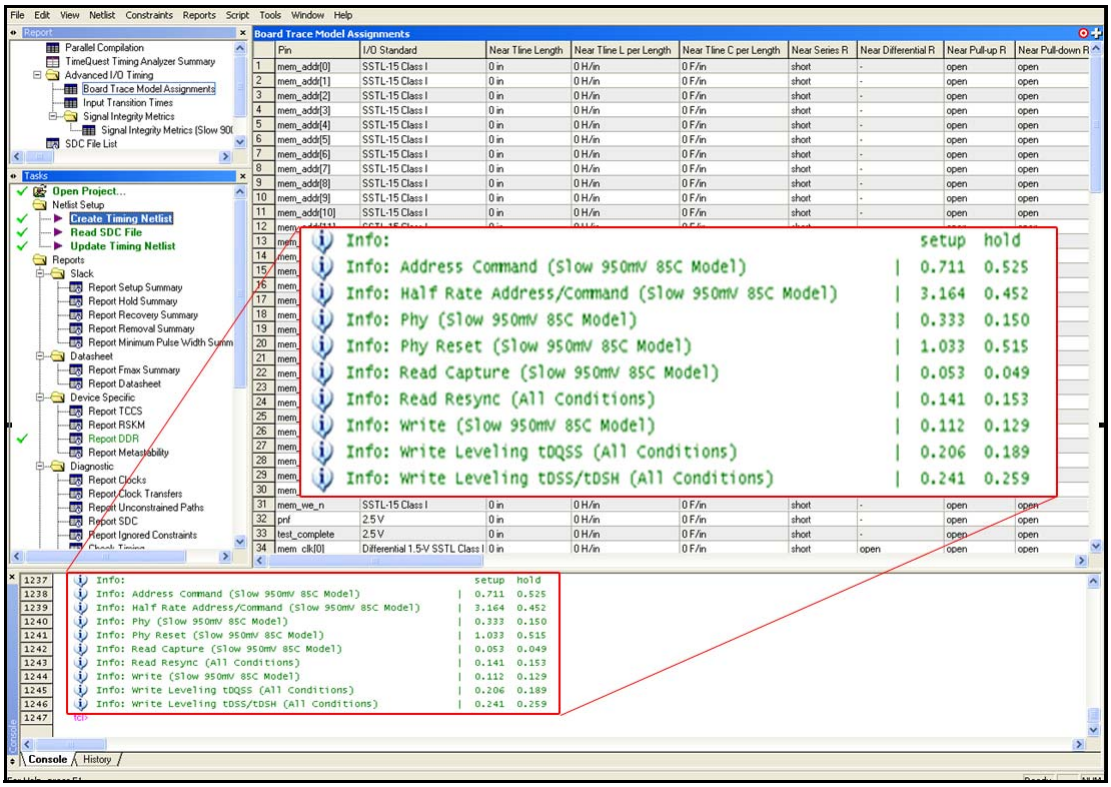


You can also obtain the timing report by running the report timing script, <*variation_name*>**_timing.tcl**, in the TimeQuest Timing Analyzer window. To obtain the timing report in the TimeQuest Timing Analyzer window, perform the following steps:

1. On the Tools menu, click **TimeQuest Timing Analyzer**.
2. On the **Tasks** pane, double-click on **Report DDR** to run **Update Timing Netlist**, **Create Timing Netlist,** and **Read SDC File**. This command subsequently executes the report timing script to generate the timing margin report.

Figure 5–5 shows the timing margin report in the TimeQuest Timing Analyzer window after running the report timing script. The results are the same as those obtained from the Quartus II software directly.

Figure 5–5. Timing Margin Report in TimeQuest Timing Analyzer



You must verify the timing at every corner of the timing model. You must run the timing report script with all available timing models—slow 0°C, slow 85°C, and fast 0°C—to ensure positive margins across the process, voltage, and temperature variations. To analyze timing on a different corner, double-click **Set Operating Conditions** in the left pane. Select a new timing corner, then go to the Script menu and rerun the <*variation_name*>**_report_timing.tcl** script.

The Stratix IV devices need post-processing script to remove timing model pessimism on the write and read capture path margins, refer to Figure 5–6 and Figure 5–7.

Figure 5–6. Write Margin Summary

	Operation	Setup Slack	Hold Slack
1	⊟ Standard Write	-0.044	-0.025
2	Spatial correlation pessimism removal	0.065	0.065
3	More clock pessimism removal	0.096	0.094
4	Write	0.117	0.134

Figure 5–7. Read Capture Path Margin Summary

	Operation	Setup Slack	Hold Slack
1	⊟ Standard Read Capture	0.011	0.007
2	Spatial correlation pessimism removal	0.047	0.047
3	Read Capture	0.058	0.054

The standard write and read capture margins are initially calculated using the FPGA timing model and adjusted to account for the effects not modeled by either the timing model or TimeQuest timing analyzer. These effects include memory calibration, deskew topologies, quantization error and calibration uncertainties.

The final write and read capture margins are summarized in the Report DDR margin summary, refer to Figure 5–8.

Figure 5–8. Report DDR Margin Summary

	Path	Operating Condition	Setup Slack	Hold Slack
1	Address Command (Slow 950mV 85C Model)	Slow 950mV 85C Model	0.717	0.528
2	Half Rate Address/Command (Slow 950mV 85C Model)	Slow 950mV 85C Model	3.170	0.444
3	Phy (Slow 950mV 85C Model)	Slow 950mV 85C Model	0.333	0.121
4	Phy Reset (Slow 950mV 85C Model)	Slow 950mV 85C Model	1.042	0.519
5	Read Capture (Slow 950mV 85C Model)	Slow 950mV 85C Model	0.058	0.054
6	Read Resync (All Conditions)	All Conditions	0.142	0.154
7	Write (Slow 950mV 85C Model)	Slow 950mV 85C Model	0.117	0.134
8	Write Leveling tDQSS (All Conditions)	All Conditions	0.206	0.189
9	Write Leveling tDSS/tDSH (All Conditions)	All Conditions	0.241	0.259

For more information about the TimeQuest timing analyzer, refer to *The Quartus II TimeQuest Timing Analyzer* chapter in volume 7 of the *Quartus II Handbook*. For information about timing analysis, refer to the *Timing Analysis* section in volume 4 of the *External Memory Interface Handbook*.

Determine Board Design Constraints and Perform Board-Level Simulations

The Stratix III and Stratix IV devices support both series and parallel OCT resistors to improve signal integrity. The Stratix III and Stratix IV OCT resistors also eliminate the need for external termination resistors on the FPGA. This feature simplifies board design and reduces overall board cost. The series ODT features are available in settings of 34 Ω and 40 Ω Although 40 Ω is not supported by all vendors.

All DDR3 SDRAM interfaces use the following two classes of signal type:

- Unidirectional class I terminated signals, which include clocks, and address and command signals.
- Bidirectional class II terminated signals, which include `DQS`, `DQ`, and `DM` signals.

For bidirectional signals, Class II termination is recommended to ensure proper termination for the following operation:

- Reads from the memory component, termination at the Stratix III or Stratix IV GX FPGA side.
- Writes to the memory component, termination at the memory side.

The Stratix III and Stratix IV devices include on-chip series and parallel termination. So generally, discrete termination at the FPGA end of the line is not required.

The DDR3 SDRAM devices support dynamic parallel ODT at the memory end of the line. So typically, discrete termination is not required. For these reasons, Class I termination are used for bidirectional signals in this tutorial.

To understand better about the different effects on signal integrity design, refer to the *Board Layout Guidelines* section in volume 2 of the *External Memory Interface Handbook*. For more information about the signal integrity of the Stratix III and Stratix IV devices, refer to Stratix III Device Signal and Power Integrity and Stratix IV FPGA Signal and Power Integrity pages.

Verify Design on a Board

The SignalTap II Embedded Logic Analyzer shows read and write activity in the system.

For more information about using the SignalTap II Embedded Logic Analyzer, refer to the For more information about using the SignalTap II Embedded Logic Analyzer, refer to the *Design Debugging Using the SignalTap II Embedded Logic Analyzer* chapter in the *Quartus II Handbook*, *AN 323: Using SignalTap II Embedded Logic Analyzers in SOPC Builder Systems* and *AN 446: Debugging Nios II Systems with the SignalTap II Logic Analyzer*.

To add the SignalTap II Embedded Logic Analyzer, perform the following steps:

1. On the Tools menu, click **SignalTap II Logic Analyzer**.
2. Under **Signal Configuration**, next to the **Clock** box click **…** (**Browse Node Finder**).

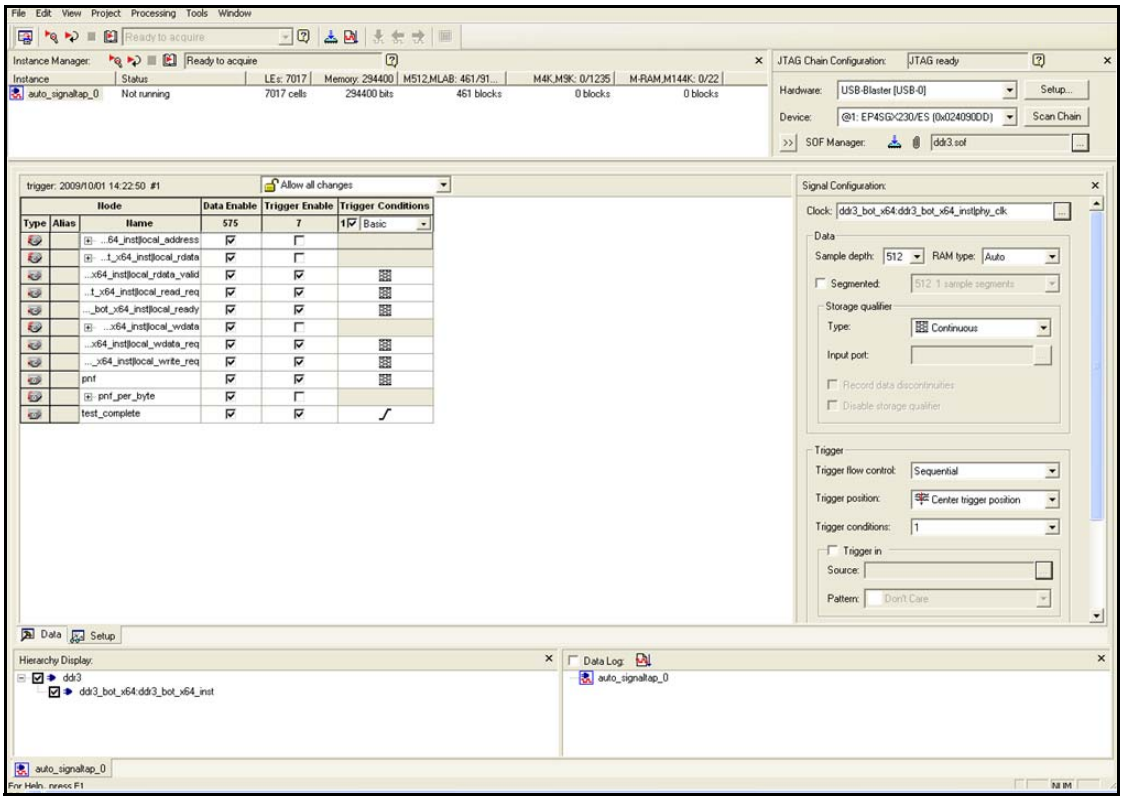
3. Type `*phy_clk` in the **Named** box, for **Filter** select **SignalTap II: pre-synthesis** and click **List**.
4. Select **ddr3_dimm | ddr3_dimm_inst | phy_clk** for a Stratix III design, or **ddr3_bot_x64_example_top | ddr3_bot_x64:ddr3_bot_x64 | phy_clk** for a Stratix IV design, in **Nodes Found** and click **>** to add the signal to **Selected Nodes**.
5. Click **OK**.
6. Under **Signal Configuration**, specify the following settings:
   - For **Sample depth**, select **512**
   - For **RAM type**, select **Auto**
   - For **Trigger flow control**, select **Sequential**
   - For **Trigger position**, select **Center trigger position**
   - For **Trigger conditions**, select **1**
7. On the Edit menu, click **Add Nodes**.
8. Search for specific nodes by typing `*local*` in the **Named** box, for **Filter** select **SignalTap II: pre-synthesis** and click **List**.
9. Select the following nodes in **Nodes Found** and click **>** to add to **Selected Nodes**:
   - **local_address**
   - **local_rdata**
   - **local_rdata_valid**
   - **local_read_req**
   - **local_ready**
   - **local_wdata**
   - **local_wdata_req**
   - **local_write_req**
   - **pnf**
   - **pnf_per_byte**
   - **test_complete** (trigger)
   - **ctl_cal_success**
   - **ctl_cal_fail**
   - **ctl_wlat**
   - **ctl_rlat**

   
   Do not add any DDR3 SDRAM interface signals to the SignalTap II Embedded Logic Analyzer. The load on these signals increases and adversely affects the timing analysis.
10. Click **OK**.

11. To reduce the SignalTap II logic size, turn off **Trigger Enable** on the following bus signals:
    - **local_address**
    - **local_rdata**
    - **local_wdata**
    - **pnf_per_byte**
    - **ctl_wlat**
    - **ctl_rlat**
12. Right-click **Trigger Conditions** for the `test_complete` signal and select **Rising Edge**.

    Figure 5–9 shows the completed SignalTap II Embedded Logic Analyzer.

Figure 5–9. SignalTap II Embedded Logic Analyzer



13. On the File menu, click **Save**, to save the SignalTap II **.stp** file to your project.

    If you see the message **Do you want to enable SignalTap II file "stp1.stp" for the current project**, click **Yes**.

Compile the Project

After you add signals to the SignalTap II Embedded Logic Analyzer, to recompile your design, on the Processing menu, click **Start Compilation**.

Verify Timing

After the design compiles, ensure that TimeQuest timing analysis passes successfully. In addition to this FPGA timing analysis, check your PCB or system SDRAM timing. To run timing analysis, perform the following steps to run the <*variation name*>**_phy_report_timing.tcl** script:

1. On the Tools menu, click **Tcl Scripts**.
2. Select <*variation name*>**_phy_report_timing.tcl**. \
3. Click **Run**.

Download the Object File

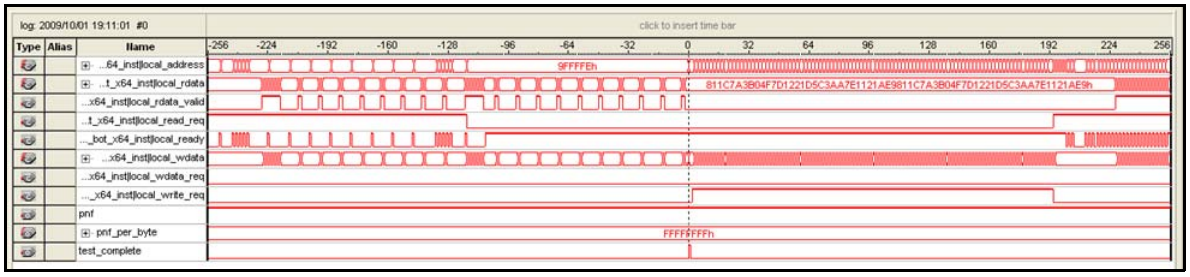
To download the object file to the device, perform the following steps:

1. Connect the development board to your computer.
2. On the Tools menu, click **SignalTap II Logic Analyzer**. The SignalTap II dialog box appears.
3. Click **...** to open the **Select Program Files** dialog box.
4. Select <*your project name*>**.sof**.
5. Click **Open**.
6. To download the file, click the **Program Device** button.

Test the Design Example in Hardware

When the design example including SignalTap II successfully downloads to your development board, click **Run Analysis** to run once, or click **Autorun Analysis** to run continuously. Figure 5–10 shows the design analysis.

Figure 5–10. SignalTap II Example DDR3 SDRAM Design Analysis



6. Using High-Performance DDR, DDR2, and DDR3 SDRAM with SOPC Builder

The Altera DDR, DDR2, and DDR3 SDRAM Controllers with ALTMEMPHY IP version 9.1 support SOPC Builder, enabling you to instantiate a DDR, DDR2, or DDR3 SDRAM Controller with ALTMEMPHY IP in an SOPC Builder system.

This walkthrough discusses the following topics:

- Consider SOPC Builder system interconnect fabric and performance implications:
  - High-performance controller (HPC) or high-performance controller II (HPC II)
  - Full- or half-rate SDRAM high-performance controller
  - System and component clock selection, and half-rate bridges
  - Burst reads and writes
  - Latency and how to optimize read or write addressing
- Implement a DDR2 SDRAM high-performance controller in SOPC Builder
- Incorporate a Nios® II Processor and other peripherals
- Compile the design and generate the programming file
- Download the design to a development board and run sample code on your design to test read and write transactions

The design in this walkthrough ensures an optimum SOPC Builder Avalon-MM architecture and demonstrates how a simple Nios II C program, when combined with a SignalTap II Embedded Logic Analyzer, can verify both hardware and software operation.

For more information on functional simulation of an SOPC Builder system, refer to *AN 351: Simulating Nios II Embedded Processor Designs*. For more information about the DDR, DDR2, and DDR3 SDRAM high-performance controllers, refer to the *DDR and DDR2 SDRAM Controller with ALTMEMPHY IP User Guide* section and the *DDR3 SDRAM Controller with ALTMEMPHY IP User Guide* section in volume 3 of the *External Memory Interface Handbook*.

SOPC Builder System Considerations

Always consider the following caveats and limitations when integrating a DDR, DDR2, or DDR3 SDRAM high-performance controller in SOPC Builder:

- High-Performance Controller (HPC) or High-Performance Controller II (HPC II)
- Full- or Half-Rate SDRAM High-Performance Controller
- Clock Selection and Clock Crossing Bridges
- Burst Reads and Writes
- Multimasters
- Direct Memory Access (DMA) Controller

- Read and Write Addressing and Latency

High-Performance Controller (HPC) or High-Performance Controller II (HPC II)

Select either HPC or HPC II based on your system design. HPC II is an enhanced version of HPC with higher efficiency, and provides the following features:

- Supports higher efficiency look-ahead bank management with in-order read and write, out of order management.
- Supports bank interleaving with additive latency and auto precharge to provide higher efficiency.
- Provides optional run-time programmability to configure the behavior of the controller such as memory timing, size and mode register settings, allowing you to reconfigure and read the controller parameters in user mode, and reduce compilation time.
- Has half-bridge integrated in the controller to reduce the memory access latency.
- Supports multi-cast write to mitigate t_{RC}.
- Has integrated flexible built-in burst adapter to automatically split or merge user burst request to match memory's native burst length, allowing you to set a maximum of 64 beats at the local side.
- Supports integrated error correction coding (ECC), which supports 40-bit and 72-bit interfaces with sub-word writes, and optional automatic write-back on error.
- Supports Avalon-MM interface.

For more information about HPC II, the *DDR and DDR2 SDRAM Controller with ALTMEMPHY IP User Guide* section and the *DDR3 SDRAM Controller with ALTMEMPHY IP User Guide* section in volume 3 of the *External Memory Interface Handbook*.

Altera recommends that you use the HPC II, which provides more flexibility and higher efficiency.

Full- or Half-Rate SDRAM High-Performance Controller

Full- or half-rate SDRAM high-performance controllers have the following definitions:

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

Implementing the SDRAM high-performance controllers in half-rate mode gives the highest possible SDRAM clock frequency while allowing the more complex core logic to operate at half this frequency. You can simplify the complexity of your design by permitting your Nios II processor to run at the slower, half-rate memory speed while still achieving the required SDRAM bandwidth per I/O pin.

However, where possible it is generally more optimal to configure the controller in full-rate mode with the core operating at the same clock frequency as your SOPC Builder system.

Full-Rate Versus Half-Rate Command Operation

Commands can be slower using a half-rate controller. For example, a DDR SDRAM device can have a number of banks open at once. Each bank has a currently selected row. Changing the column within the selected row of an open bank requires no additional bank management commands to be issued. Changing the row in an active bank, or changing the bank both incur a protocol penalty that requires the precharge (PCH) command closes the active row or bank, and the active (ACT) command then opens or activates the new row or bank combination.

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

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

Time-Specified Memory Parameters

For a half-rate SDRAM high-performance controller, the control circuitry is clocked at half rate and so control operations are slower than in full-rate mode. However, the memory's clock frequency and physical properties are not affected.

When you use half-rate mode, any time-specified memory parameters in the controller are modified.

For example, if:

t_{RCD}min = 20 ns

For a 133-MHz controller:

t_{CK} = 7.5 ns

20/7.5 = 2.666 rounded up to 3 clock cycles (22.5ns).

For a half-rate 66-MHz controller:

t_{CK} = 15 ns

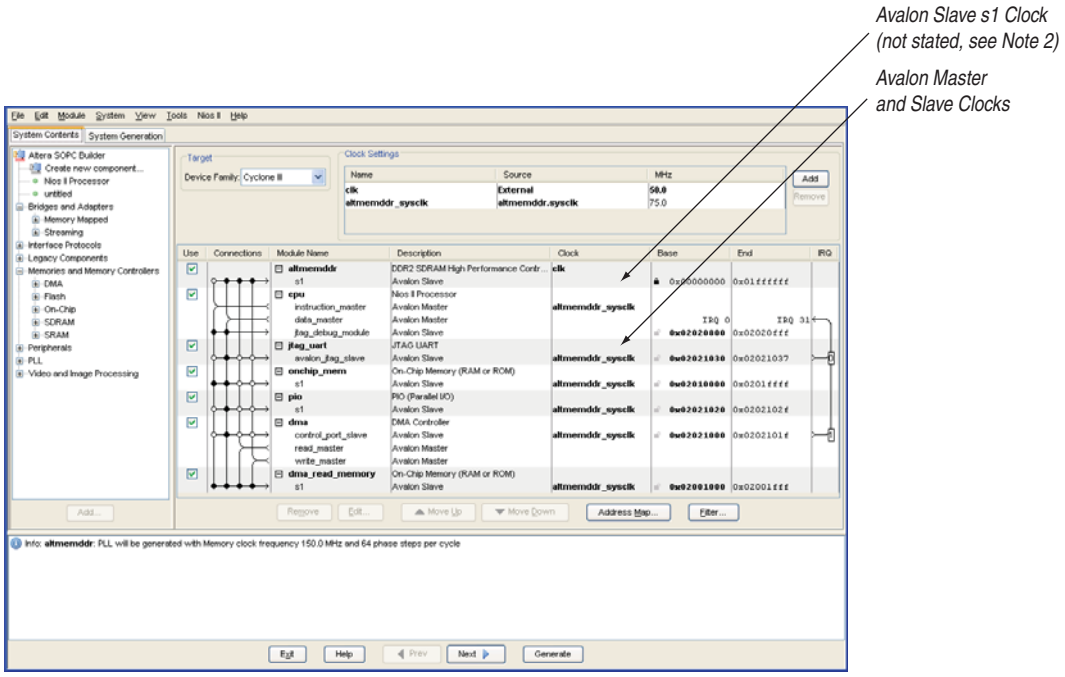
20/15 = 1.33 rounded up to 2 clock cycles (30ns).

Thus bank and row changes are slower in half-rate mode, but are not twice as slow. The easiest way to measure this effect for your chosen memory device and interface clock speed is to simulate both half-rate and full-rate designs and record the increased latency when switching rows. Typically a full-rate controller is around 14% more efficient when switching rows within a bank. As a half-rate controller has less read latency and if the masters in your system do not cause bank switching to occur often, the half-rate controller still gives higher performance. In SOPC Builder, you can alter arbitration shares to prevent masters from switching memory banks, thus creating a more optimal system.

Clock Selection and Clock Crossing Bridges

Ideally every component in the SOPC Builder system should be clocked using the same clock, to prevent SOPC Builder automatically adding clock domain crossing logic, which adds latency. The SDRAM high-performance controllers already include a PLL, so you should set the SOPC Builder system clock to `altmemddr.sysclk`, which is the clock the controller and local interface logic use. SOPC Builder only shows the input clock to the controller PLL, not the interface clock, refer to Figure 6–1 on page 6–4.

Figure 6–1. SOPC Builder Avalon-MM Slave and Master Clock Selection *(Note 1)* and *(2)*



Notes to Figure 6–1:

(1) The `clk` clock is the input clock to the controller PLL.

(2) The `s1` clock is not stated, but is on the `altmemddr.sysclk` clock domain, not on the `clk` domain.

If a different Avalon-MM clock is specified for connected components and a clock crossing bridge is not used, SOPC Builder automatically adds clock crossing adapters between any data masters of your SOPC Builder system and the SDRAM high-performance controller slave interface. Clock crossing adapters provide robust and safe transactions between different clock domains, however they increase latency and limit total bandwidth.

To prevent SOPC Builder from auto-inserting clock adapters:

- If the connected SOPC Builder components and SDRAM high-performance controller operate at the same frequency, use `altmemddr.sysclk` as the clock for the rest of your system. No clock domain crossing adapters are added by SOPC Builder.

- If the connected SOPC Builder components and SDRAM high-performance controller operate at different frequencies, manually insert an Avalon-MM clock crossing bridge between the SDRAM high-performance controller and the other SOPC Builder components.
- If the connected SOPC Builder components and SDRAM high-performance controller operate at different frequencies but are in the same clock phase, manually insert an Avalon-MM DDR memory half-rate bridge between the SDRAM high-performance controller and the other SOPC Builder components. Alternatively, turn on the **Enable Half Rate Bridge** option on the **Memory Settings** tab to use the embedded half-rate bridge in HPC II.

To use the Avalon-MM clock crossing bridge or the Avalon-MM DDR memory half-rate bridge, set the slave clock to your SOPC Builder system clock, and the master clock to `altmemddr.sysclk`.

For more information about the Avalon-MM clock crossing bridge and Avalon-MM DDR memory half-rate bridge, refer to the *Avalon Memory-Mapped Bridges* chapter in volume 4 of the *Quartus II Handbook*.

The Avalon-MM clock crossing bridge also works when you are using two different clocks at the same frequency but with different phases. However, the Avalon-MM DDR memory half-rate bridge must meet the following requirements:

- The clock frequency of the memory-side master must be in the same phase and twice of the processor-side slave frequency.
- The memory-side master is half as wide as the processor-side slave.

Use either the Avalon-MM clock crossing bridge or the Avalon-MM DDR memory half-rate bridge to allow a full-rate controller to connect to a half-rate SOPC builder design. As the memory controller still operates at full rate, twice the Nios II frequency, the memory interface commands operate at the faster rate.

The Avalon-MM DDR memory half-rate bridge has the same functionality as the Avalon-MM clock crossing bridge but provides lower latency. Altera recommends you use the Avalon-MM DDR memory half-rate bridge for Nios II Processors that require low latency access to high-speed memory.

Figure 6–2 shows a block diagram of how to use an Avalon-MM clock crossing bridge or an Avalon-MM DDR memory half-rate bridge.

Figure 6–2. Clock Crossing Bridge

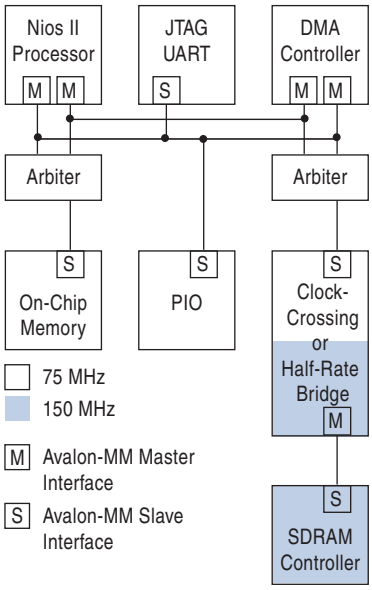
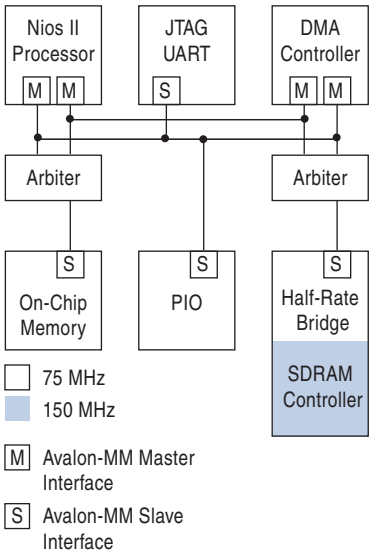


Figure 6–3 shows a block diagram of how a half-rate bridge is embedded in HPC II.

Figure 6–3. Half-Rate Bridge in DDR SDRAM High-Performance Controller II



Burst Reads and Writes

Burst reads and writes differ in the following ways:

- In half-rate designs, the Avalon-MM slave interface is four times the width of the SDRAM device. Hence four transactions are performed on the SDRAM for every single Avalon-MM transaction. Avalon-MM burst requests on the local side serve no purpose as the memory interface is already using the maximum supported memory burst size for every single Avalon-MM transaction.
- In full-rate designs, you can use Avalon-MM bursts of one or two with the SDRAM high-performance controller, as each Avalon-MM transaction results in only two SDRAM transactions. So two Avalon-MM burst transactions may be combined to support the four supported on the SDRAM high-performance controller.

When a burst capable master supports larger burst lengths than the slave, SOPC Builder automatically places a burst length adapter into the path.

To obtain performance advantages, ensure that the data widths of master and slave pairs are matched in SOPC Builder. Whenever a master port is connected to a slave port of a different width, SOPC Builder automatically inserts adapter logic to convert between the different data widths.

Ideally, the data size of an Avalon-MM interface is 32 bits for a Nios II processor:

- For a full-rate SDRAM high-performance controller, the double-data rate stage doubles the data width, thus a 16-bit external memory width is best.
- For a half-rate SDRAM high-performance controller, the double-data rate and half-rate stages both double the data width, thus an 8-bit external memory width is best.

Ideally, match the following data cache line sizes to the memory controller burst length:

- 4-byte line = bursts of 1 (32-bit word)
- 16-byte line = bursts of 4
- 32-byte line = bursts of 8

You can set the data bus arbitration priority to avoid using the burst signal.

Table 6–1 and Table 6–2 show the burst length support for each type of DDR SDRAM HPC and HPC II.

Table 6–1. Burst Length Support for DDR SDRAM HPC

Memory Type	Full-Rate Mode	Half-Rate Mode
DDR3	—	8
DDR2	4	4
DDR	2 or 4	4

Table 6–2. Burst Length Support for DDR SDRAM HPC II

Memory Type	Full-Rate Mode	Half-Rate Mode
DDR3	—	8
DDR2	4	8
DDR	4	8

HPC II consists of a built-in burst adapter that automatically splits or merges bust request to match the memory's native burst length. This built-in adapter allows the Nios II processor to request any burst length and maps the burst length to the most efficient memory burst. HPC II accepts up to 64 burst count.

Multimasters

To achieve best throughput and latency of the SDRAM, you should connect the controller to the smallest number of masters and share those masters with the smallest number of slaves. Fewer connections reduce the complexity of the SOPC Builder auto-inserted data multiplexers and increase the f_{MAX} of the Avalon-MM interface.

If the Avalon-MM interface f_{MAX} becomes the limiting factor, insert pipeline bridges to increase f_{MAX} the (at the expense of latency).

Master and slave pairs are locked for the whole duration of a burst—no other master is granted access to the slave target of a burst until the burst is completed. You should isolate critical memory connections from the rest of the system.

One master may lock an SDRAM high-performance controller until a write burst is completed, which may take several cycles of time, during which the SDRAM high-performance controller cannot accept any other request. The write burst command is complete when all burst data is passed to the SDRAM high-performance controller. For a read burst, the controller quickly transfers back the whole burst at full bandwidth and becomes immediately available for new requests.

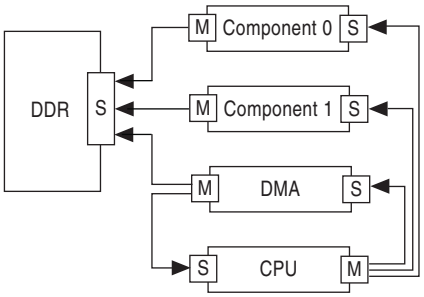
Direct Memory Access (DMA) Controller

The DMA controller enables the memory data transfer directly without being preformed through the processor to achieve better performance and parallelism between operations. By enabling the direct transfer, the processor can perform other tasks in parallel.

For more information, refer to the *DMA Controller Core* chapter in volume 5 of the *Quartus II Handbook*.

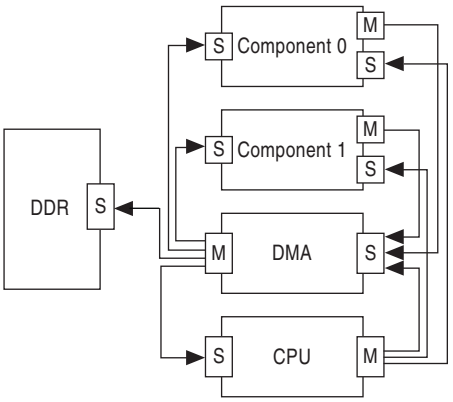
The DMA controller can perform bulk data transfers, reading data from a source address range and writing data to a different address range. The DMA controller boosts the memory band-width and alleviates the multimaster bottleneck. With the use of a DMA controller, you can reduce the number of masters that access the memory. Figure 6–4 shows an example where there are multiple masters access to the memory. This increases the complexity of the connection and latency to the system.

Figure 6–4. Multiple Masters Accessing a Single Memory



You can use the DMA controller to reduce the bottleneck for the multimaster access to the memory. The DMA controller acts as the only master that accesses the memory but as a slave for other components that wants access to the memory. DMA controller uses interrupt request to notify the masters of other components that the transfers are completed without wasting time with polling. Figure 6–5 shows an example of how DMA controller is used to enhance the memory bandwidth for multimaster.

Figure 6–5. A Single Master Accessing Memory with DMA Controller



Read and Write Addressing and Latency

Systems with deterministic access patterns can minimize the number of bank and row changes. For example, by suitably arranging the memory map. In systems with more random access patterns (often typical in embedded SOPC Builder-type systems), minimizing bank and row changes is more difficult and the increased latency (by constantly changing the row in an active bank, or changing the bank) has a greater effect. Non-optimal cache implementations can waste cycles with half-rate controllers.

You should always consider the following actions:

- Match controller Avalon-MM interface width to Avalon-MM master interface width.
- Minimize non-sequential addressing to reduce row addressing time.
- Match the Nios II cache line size to memory burst length.
- Set arbitration priorities correctly.
- Insert bridges to increase f_{MAX} at the expense of latency.
- Minimize multimaster designs where arbitration stalls may take place.

In multi-mastering designs, the memory is not available to all masters concurrently.

Setting the arbitration priorities correctly prevents unnecessary memory accesses. For example, always set the Nios II instruction master arbitration priority to eight, because it always tries to access eight sequential addresses. Set the data master arbitration priorities depending on the cache line size; do not leave the arbitration priority value as default.

DDR2 SDRAM Controller with ALTMEMPHY IP with SOPC Builder Walkthrough

This walkthrough shows how to use the SOPC Builder design flow to design a 8-bit wide, 150-MHz, 300-Mbps DDR2 SDRAM interface. The design example also provides some recommended settings to simplify the design. Although the design example is specifically for the DDR2 SDRAM interface, the design flow for DDR and DDR3 SDRAM interfaces are the same.

The design example targets the Cyclone III FPGA development kit with four MT47H32M16CC-3 components and an 8-bit wide 512-MB Micron MT47H32M16CC-3 333-MHz DDR2 SDRAM component.

System Requirement

This application note assumes that you are familiar with the Quartus II software and SOPC Builder. This tutorial requires the following hardware and software:

- Quartus II software version 9.1
- ModelSim-SE 6.5b or higher
- DDR2 SDRAM Controller with ALTMEMPHY IP version 9.1
- Nios II Embedded Design Suite (EDS) version 9.1

The design is targeted to the Cyclone III FPGA development kit. You can target other supported device families or development kits.

For more information on the Cyclone III FPGA development kit, refer to www.altera.com/products/devkits/altera/kit-cyc3.html.

Create Your Example Project

This section shows you how to create a new Quartus II project and an SOPC Builder system.

Create a New Quartus II Project

In the Quartus II software, create a new project with the **New Project Wizard**, ensure that the device type is set to Cyclone III, EP3C120F780C7.

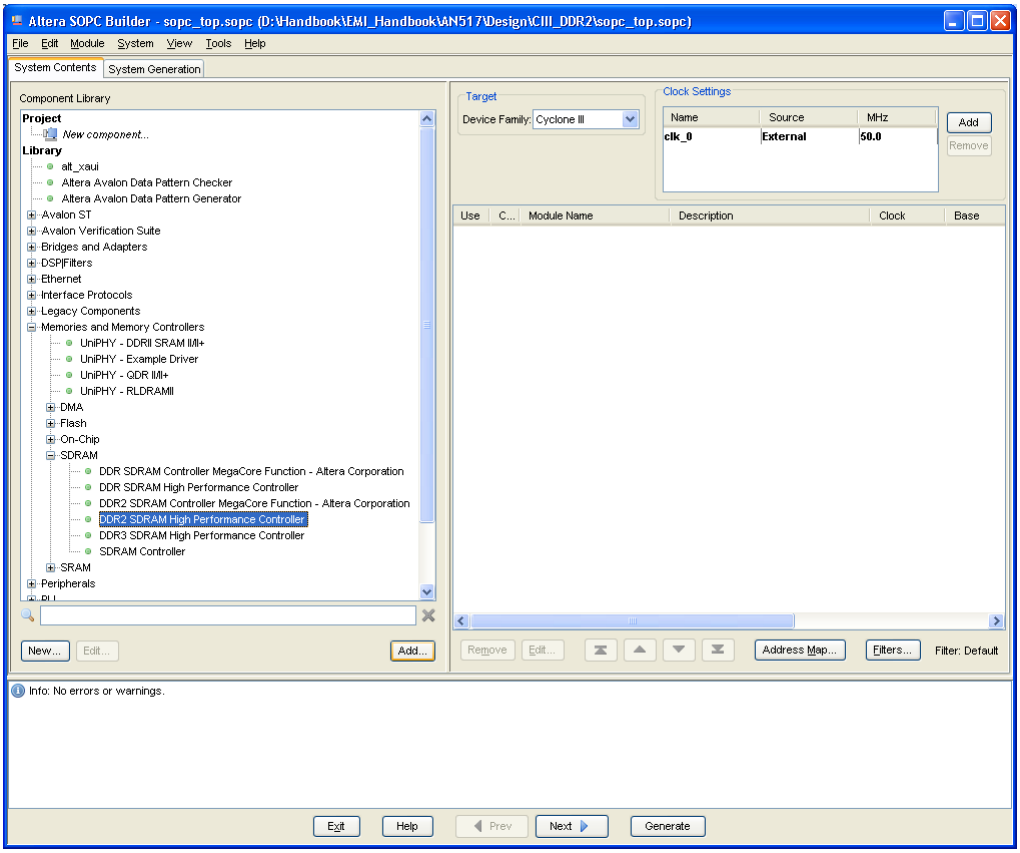
Ensure that your project path does not include any spaces or extended characters.

Create the SOPC Builder System

To create an SOPC Builder system, perform the following steps:

1. On the Tools menu, click **SOPC Builder**.
2. In the **Create New System** dialog box, type `sopc_top` for the **System Name**. In the **Target HDL**, select **Verilog**, then click **OK**.
3. In the **System Contents** tab, under **Component Library**, expand **Memories and Memory Controllers**. Expand **SDRAM**. Select **DDR2 SDRAM High Performance Controller** and click **Add**. The DDR2 SDRAM Controller with ALTMEMPHY IP wizard opens, refer to Figure 6–6 on page 6–11.

Figure 6–6. System Content



4. In the **DDR2 High Performance Controller** wizard interface, select **7** for the **Speed grade** to match the chosen device.
5. For the **PLL reference clock frequency**, type **50** MHz, to match signal `CLKIN50`.
6. For the **Memory clock frequency**, type **150** MHz.

☞ 150 MHz is the maximum supported frequency for DDR2, SSTL-18 class I, in top and bottom I/O, in a C7 speed grade Cyclone III device.

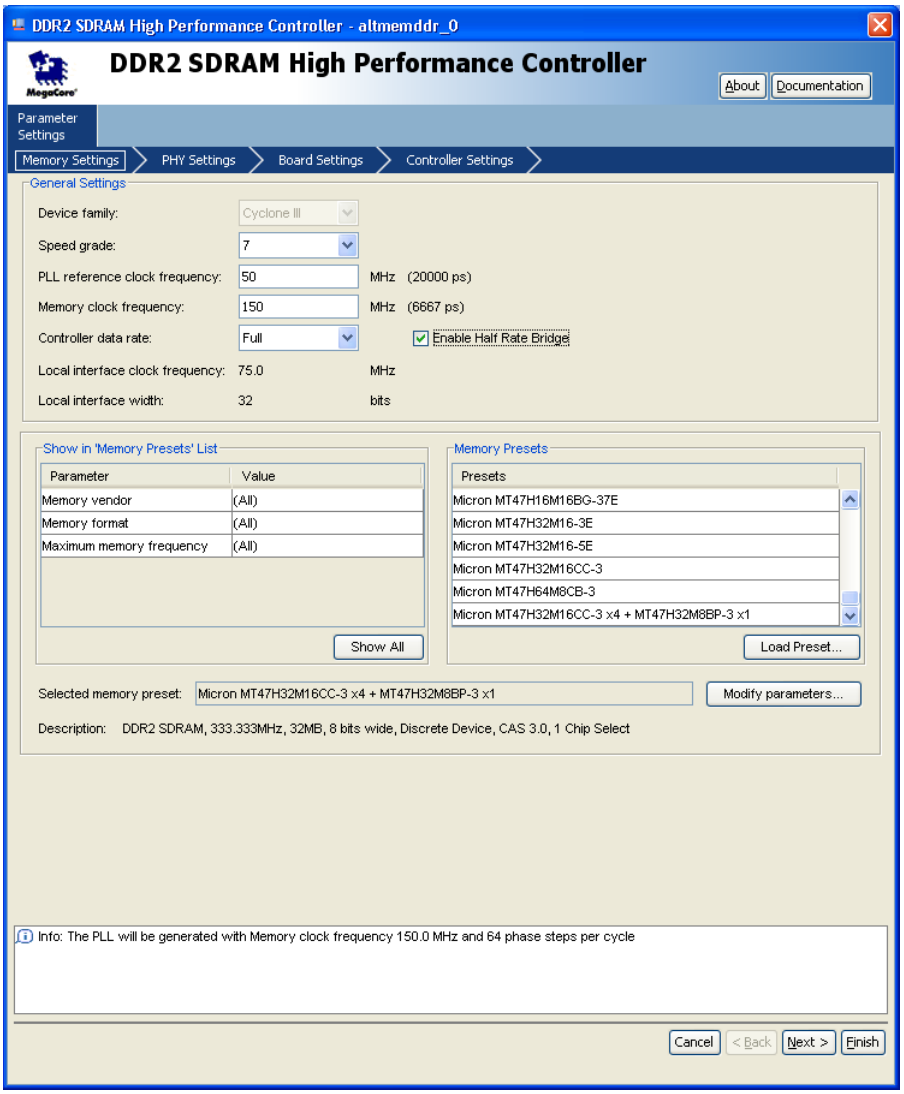
7. For the **Controller data rate**, select **Full**. Full-rate frequency is selected because the half-rate bridge is used in HPC II to reduce latency by driving the PHY and controller to work faster.

   ☞ The **Enable Half Rate Bridge** option is only available with the full-rate memory controllers.

8. For the **Memory vendor**, select **Micron**.
9. For the **Memory format**, select **Discrete Device**.
10. For the **Memory Presets**, select **Micron MT47H32M16CC-3 x4 + MT47H32M8BP-3 x1**, refer to Figure 6–7 on page 6–13.

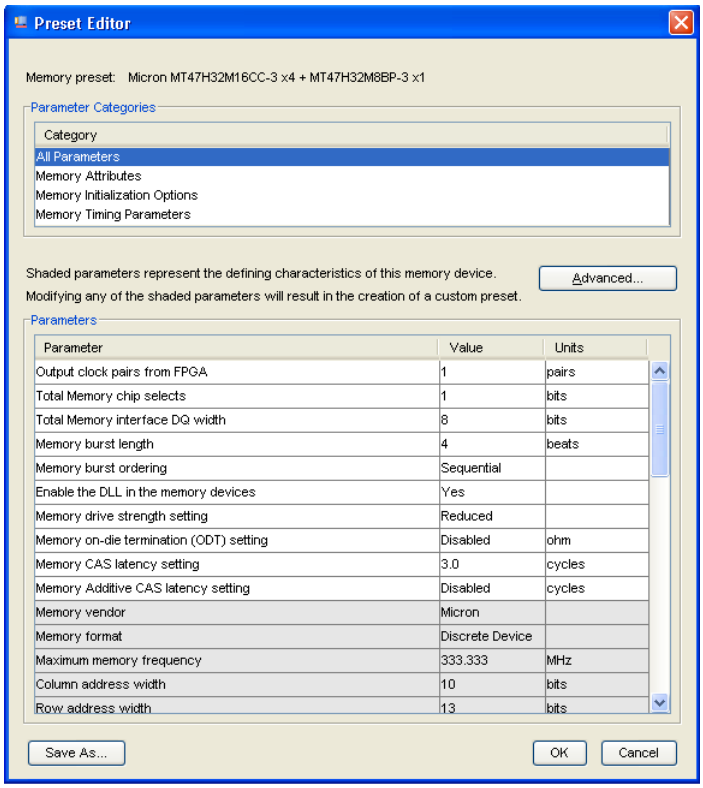
This memory preset has been included by Altera in the DDR2 SDRAM Controller with ALTMEMPHY IP as it matches the exact configuration that both the Cyclone III development board and the Stratix II GX PCIe development kit use.

Figure 6–7. Memory Settings



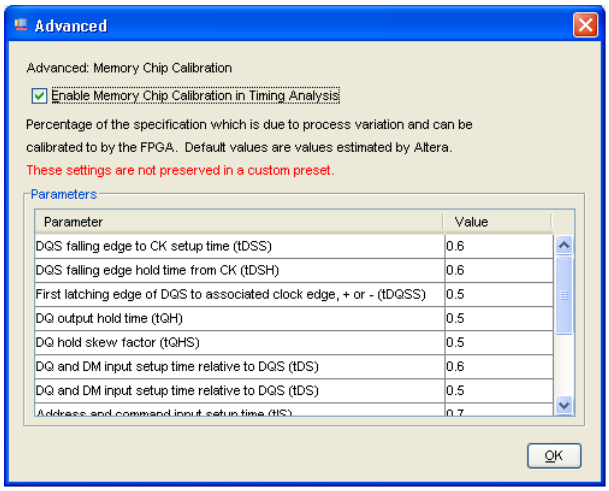
11. Click **Modify parameters** and ensure that you change the following parameters:
    - **Output clock pairs from FPGA**, select **1 pair**
    - **Total Memory interface DQ width**, select **8 bits**, to give in an Avalon-MM width of 32 bits, which is ideal to connect to a Nios II processor
    - **Memory drive strength setting**, select **Reduced** (DQ are low load point-to-point connections)
    - **Memory on-die termination (ODT) setting**, select **Disabled** (discrete class I termination is already fitted to the development board)
    - **Memory CAS latency setting,** select **3.0 cycles** (this DDR2 SDRAM supports CAS = 3 for frequencies of 200 MHz or lower)
12. Click **OK**, then click **Finish** in the DDR2 SDRAM Controller with ALTMEMPHY IP interface.

Figure 6–8. Preset Editor



If you are using a device that requires timing model pessimism removal, ensure that you turn on the **Enable Memory Chip Calibration in Timing Analysis** option on the **Advanced** page. This option is not supported by Cyclone III devices.

Figure 6–9. Advanced Memory Settings



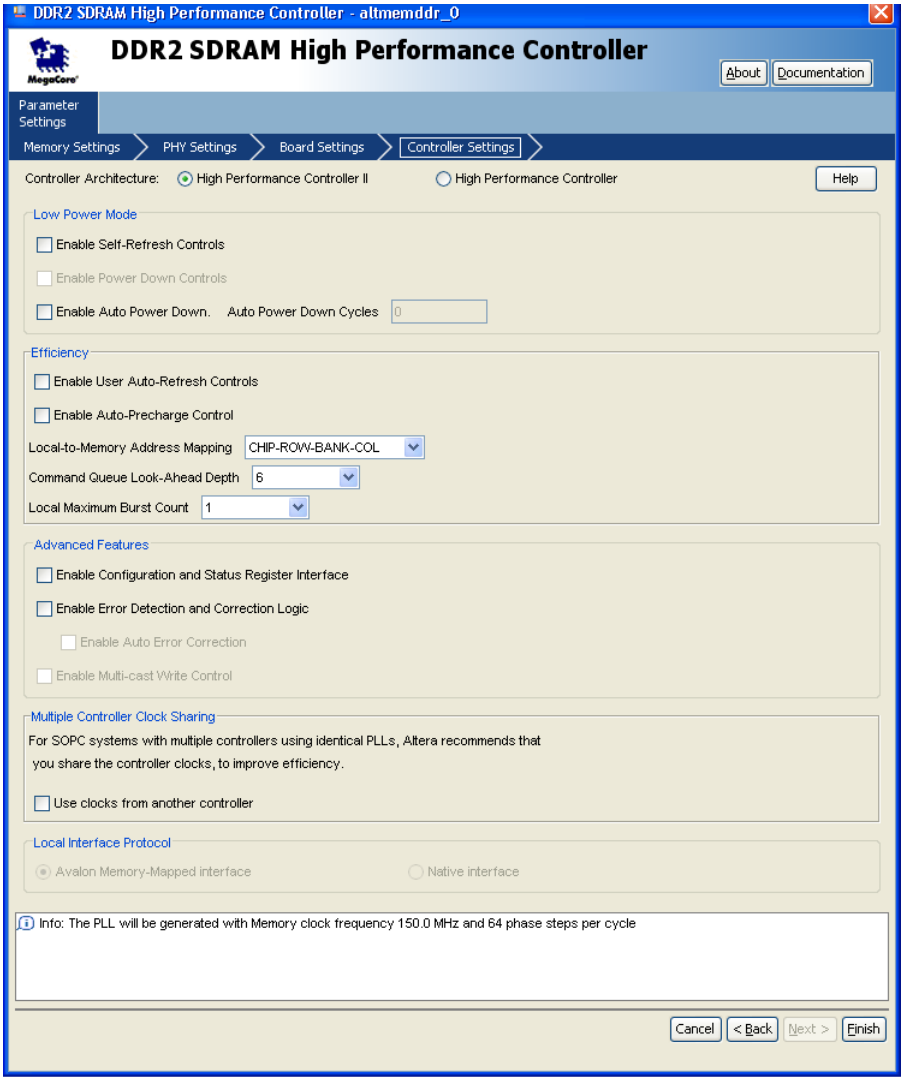
Because Cyclone III devices do not support flexible timing methodology, ignore the board settings. If you are using devices that support flexible timing methodology, ensure that you fill in the **Board Settings** parameters based on your board simulation results to achieve accurate timing analysis.

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

13. In the **Controller Settings** tab, select **High-Performance Controller II**.
14. For **Command Queue Look-Ahead Depth**, select **6**.
15. For **Local-to-Memory Address Mapping**, select **CHIP-ROW-BANK-COL**.
16. For **Local Maximum Burst Count**, select **1**, refer to Figure 6–10.

Set the maximum burst count based on the burst count of the core logic system.

Figure 6–10. DDR2 SDRAM HPC II Settings



If you are using multiple controllers, turn on the **Use clocks from another controller** option under **Multiple Controller Clock Sharing** to save clock resources, and improve system efficiency and latency. This option allows the controllers to share the static PHY clocks between multiple controllers that run on the same frequency and share the same PLL reference clock.

For more information about how to share the clock for multiple memory interfaces, refer to the *Implementing Multiple Memory Interfaces* section in volume 6 of the *External Memory Interface Handbook*.

17. Click the **Memory Settings** tab, and turn on the **Enable Half Rate Bridge** option.
18. Click **Finish**.

Add SOPC Builder Components

To add components from the **System Contents** tab, perform the following steps:

1. Under **Component Library**, expand **Memories and Memory Controllers** and expand **On-Chip**. Select **On-Chip Memory (RAM or ROM)**, and click **Add**.
   a. For **Total memory size** type **64 KBs**.
   b. Click **Finish**.
2. Select **On-Chip Memory** again, click **Add**.
   a. Set **Total memory size** to **4096 Bytes.**
   b. Click **Finish.**
   c. Right click on this second on-chip memory, click **Rename** and type `dma_read_memory` and press Enter.
3. On the System menu, click **Auto-Assign Base Addresses**.
4. Under **Component Library**, expand **Processors**, select **Nios II Processor** and click **Add**.
   a. Select **Nios II/s**.
   b. For **Reset Vector Memory** and **Exception Vector Memory**, select **onchip_memory**. If the local on-chip memory holds the Nios II instruction code, less arbitration is required to the SDRAM interface resulting in a more optimal Avalon-MM structure.
   c. Change **Reset Vector Offset** to **0x20** and **Exception Vector Offset** to **0x40**.

   CAUTION: If you select SDRAM as reset and exception vector, the controller performs memory interface calibration every time it is reset and in doing so writes to addresses `0x00` to `0x1f`. If you want your memory contents to remain intact through a system reset, you should avoid using the memory addresses below `0x20`. This step is not necessary, if you reload your SDRAM contents from flash every time you reset.

   d. Click **Finish**.
5. Expand **Interface Protocols** and expand **Serial**, select **JTAG UART** and click **Add**.
   a. In the **Configuration** tab, under **Write FIFO** and **Read FIFO**, for the **Buffer depth (bytes)** select **64** and for **IRQ threshold** type **8**.
   b. In the **Simulation** tab, select **Create Modelsim alias to open an interactive stimulus/response window**, to open the interactive display window during simulation.
   c. Click **Finish**.
6. Expand **Peripherals** and expand **Microcontroller Peripherals**, select **PIO (Parallel I/O)**, click **Add**.
   a. For the **Width** type **8 bits**.
   b. For **Direction** select **Output ports only**.
   c. Click **Finish**.

7. Expand **Memories and Memory Controllers** and expand **DMA**, select **DMA Controller** and click **Add**.
   a. In the **DMA Parameters** tab, turn on **Enable burst transfers**, and for the **Maximum burst size** select **512 words**.
   b. In the **Advanced** tab, turn off **byte**, **halfword**, **doubleword** and **quadword**.
   c. Click **Finish.**

   ☞ Do not to add a PLL component to your SOPC Builder design as the DDR2 SDRAM Controller with ALTMEMPHY IP includes one.

8. Set the bus links, refer to Figure 6–11 on page 6–19.
   a. Connect the DMA **read_master** and **write_master** to both the **altmemddr** and the **dma_read_memory**.

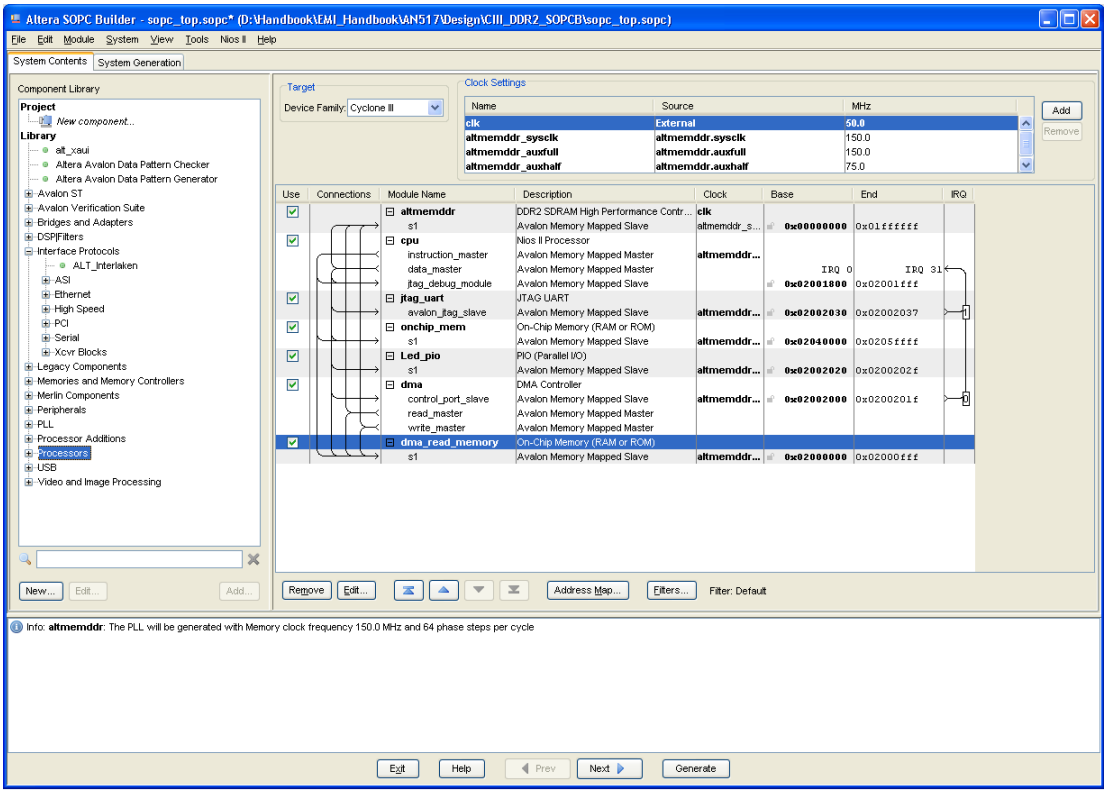
   ☞ If there are warnings about overlapping addresses, on the System menu click **Auto-Assign Base Addresses**.

   ☞ If there are warnings about overlapping IRQ, on the System menu click **Auto-Assign IRQs**.

   b. Ensure that **altmemddr** is clocked on the external clock **clk** and that the frequency matches the external oscillator (50 MHz for the Cyclone III development board).
   c. Ensure that all other modules are clocked on the **altmemddr_sysclk**, to avoid any unnecessary clock-domain crossing logic.

For more information on SOPC Builder system interconnect fabric, refer to the *System Interconnect Fabric for Memory-Mapped Interfaces* chapter in the *Quartus II Handbook*.

Figure 6–11. SOPC Builder Final System Connection



Generate the SOPC Builder System

To generate the system, perform the following steps:

1. In SOPC Builder, click **Next**.
2. Turn on the **Simulation. Create simulator project files** option.
3. In SOPC Builder, click **Generate**. Click **Save**. When the generation is completed, the following message appears:

   ```
   SUCCESS: SYSTEM GENERATION COMPLETED.
   ```

4. In SOPC Builder, click **Exit**.

For more information on SOPC simulation and testbench options, refer to the *SOPC Builder User Guide* and *AN 351: Simulating Nios II Systems*.

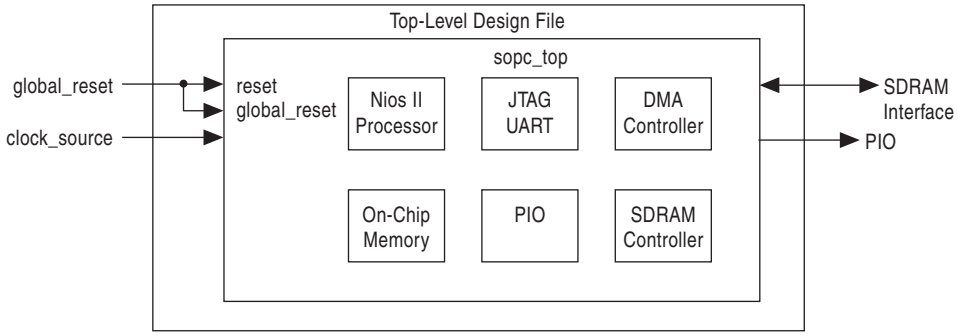
Create the Top-Level Design File

Conceptually, you can consider the SOPC Builder system as component in your design. It can be the only component; or one of many components. Hence, when your SOPC Builder system is complete, you must add it to your top-level design.

The top-level design can be in your preferred HDL language, or simply a **.bdf** schematic design.

In this walkthrough, the top-level design is a simple wrapper file around the SOPC Builder system with no additional components. The top-level design file just defines the pin naming convention and port connections. Figure 6–12 shows the SOPC Builder top-level block diagram.

Figure 6–12. SOPC Builder Top-Level Block Diagram



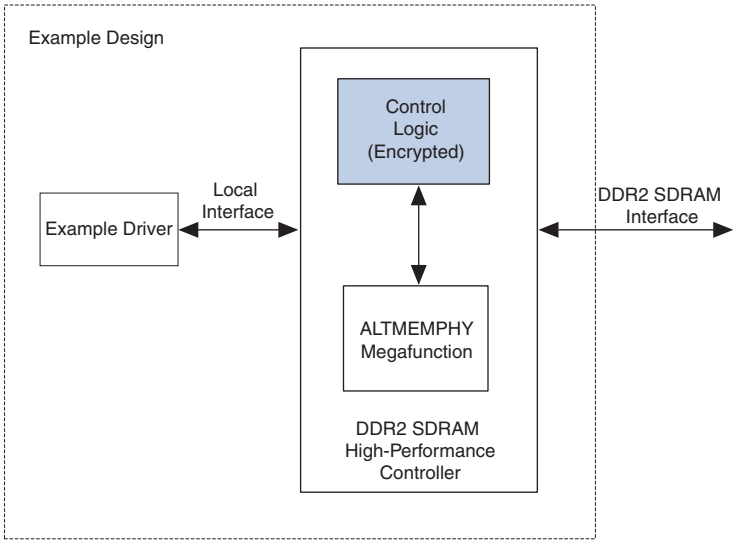
The SDRAM high-performance controller must make assignments to the top-level memory interface pins. These assignments are applied with the auto-generated script and constraint files. To ensure the constraints work, you must ensure the pin names and pin group assignments match, otherwise you get a no fit when you compile your design. The pin names default to `mem_*`. You can see the expected default pin names in the generated **altmemddr_example_top** file. You may optionally apply a prefix to the default pin naming convention.

All the SDRAM high-performance controllers generate the standalone design example **altmemddr_example_top**. This file only includes the controller and an example driver, refer to Figure 6–13. This file does not instantiate your SOPC Builder system but you can use the file in one of the following ways:

- Identify the default memory controller pin names

- Use as a starting point for the rest of the design

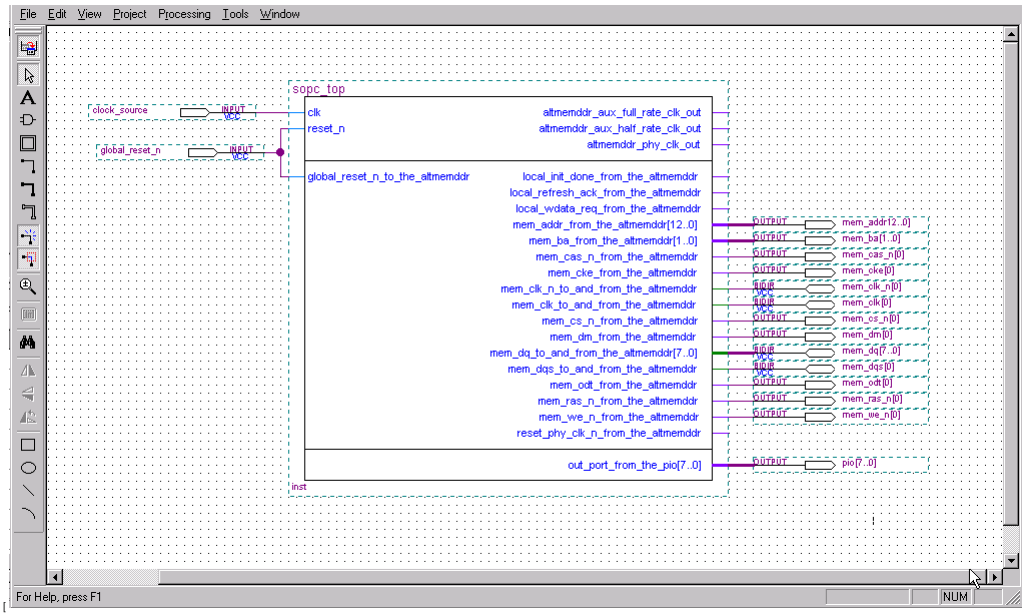
Figure 6–13. Design Example



You can create a HDL top-level design using the **altmemddr_example_top** as a template. You may edit the pin names (if required) in the **altmemddr_example_top** file only with the addition of a prefix. You must replace the example driver and the DDR2 SDRAM high-performance controller and instantiate your SOPC Builder-generated system.

As a reference, Altera provides an example**.bdf** top-level design with the correct pin names, refer to Figure 6–14. The `mem_clk[0]` and `mem_clk_n[0]` pins are bidirectional because of the mimic path in the ALTMEMPHY megafunction.

Figure 6–14. Quartus II Top-Level Project



To create a top-level design for your SOPC Builder system using a Quartus II **.bdf** schematic, perform the following steps:

1. In the Quartus II software, on the File menu click **New**.
2. Select **Block Diagram/Schematic File** and click **OK**. A blank **.bdf**, **Block1.bdf**, opens.
3. On the File menu click **Save As**. In the **Save As** dialog window, click **Save**.

   The Quartus II software automatically sets the **.bdf** file name to your project name.

4. Right-click in the blank **.bdf**, point to **Insert** and click **Symbol** to open the **Symbol** dialog box.
5. Expand **Project**, under **Libraries** select **sopc_top**, click **OK**.
6. Position the SOPC Builder system component outline in the <*project*>**.bdf** and left-click.
7. Right-click on the SOPC Builder system component and click **Generate Pins for Symbol Ports**, to automatically add pins and nets to the schematic symbol.

   The SOPC Builder system includes the following signals, which are not required for this design example and can be disconnected:

   - `altmemddr_phy_clk_out`
   - `local_init_done_from_the_altmemddr`
   - `local_refresh_ack_from_the_altmemddr`
   - `local_wdata_req_from_the_altmemddr`
   - `reset_phy_clk_n_from_the_altmemddr`
   - `altmemddr_aux_full_rate_clk_out`
   - `altmemddr_aux_half_rate_clk_out`

   The following single vector signals must have a `[0]` vectored pin name, otherwise the simulation may fail:

   - `mem_dm`
   - `mem_dqs`
   - `mem_clk`
   - `mem_cke`
   - `mem_cs_n`
   - `mem_odt`

The `altmemddr_aux_full_rate_clk_out` and `altmemddr_aux_half_rate_clk_out` clock signals instantiate the full-rate (`altmemddr_auxfull`) and half-rate (`altmemddr_auxfhalf`) clocks that are exported from the DDR2 SDRAM high-performance controller to the top level of SOPC Builder. These two clocks are used to clock the half-rate bridge if the half-rate bridge is instantiated externally. These clocks are also used to clock other components within the system.

8. The SOPC Builder system has two reset inputs, `reset_n` and `global_reset_n_to_the_altmemddr`. Connect both these signals to a single pin, `global_reset_n`.
9. Ensure that you rename the clock signal as `clock_source`.

For more information on the signals, refer to the *DDR and DDR2 SDRAM Controller with ALTMEMPHY IP User Guide* section and the *DDR3 SDRAM Controller with ALTMEMPHY IP User Guide* section in volume 3 of the *External Memory Interface Handbook*.

10. Rename `out_port_from_the_Led_pio[7...0]` to `pio[7...0]`.
11. On the File menu, click **Save**, to save your changes.
12. On the Project menu, click **Set as Top-Level Entity**.

Add Constraints

After generating the DDR2 SDRAM Controller with ALTMEMPHY IP, the ALTMEMPHY megafunction generates the constraint files for the design. You need to apply these constraints to the design before compilation.

Device Settings

Set the unused pin and device voltage correctly before adding the constraint and pin assignment. Perform the following steps to set the device voltage and unused pin:

1. In the **Category** list, select **Device** and click **Device and Pin Options**.
2. Click the **Unused Pins** tab, and for **Reserve all unused pins** select **As input tri-stated with weak pull-up resistor**.
3. Click the **Voltage** tab, and for **Default I/O standard** select the same VCCIO voltage as the chosen SDRAM interface; for DDR2 SDRAM **select 1.8 V**.
4. Click **OK**.

Add Timing Constraints

When you instantiate an SDRAM high-performance controller, it automatically generates a timing constraints file, **altmemddr_phy_ddr_timing.sdc**. The timing constraint file constrains the clock, and the input and output delay on the SDRAM high-performance controller.

To add timing constraints, perform the following steps:

1. On the Assignments menu, click **Settings**.

2. In the **Category** list, expand **Timing Analysis Settings**, and select **TimeQuest Timing Analyzer**.
3. Select the **altmemddr_phy_ddr_timing.sdc**, **altera_avalon_half_rate_bridge_constraints.sdc cpu.sdc**, and **cycloneIII_3c120_generic.sdc** files and click **Add**.

   Add `derive_pll_clocks` command in the **altera_avalon_half_rate_bridge_constraints.sdc** file before you add this file.

4. Click **OK**.

Set Optimization Technique

To ensure that the remaining unconstrained paths are routed with the highest speed and efficiency, set the optimization technique. To set the optimization technique, perform the following steps:

1. On the Assignments menu, click **Settings**.
2. Select **Analysis & Synthesis Settings**.
3. Select **Speed** under **Optimization Technique**.
4. Click **OK**.

Set Fitter Effort

To set the Fitter effort, perform the following steps:

1. On the Assignments menu click **Settings**.
2. In the **Category** list, expand **Fitter Settings**.
3. Turn on **Optimize Hold Timing** and select **All Paths**.
4. Turn on **Optimize Multi-Corner Timing**.
5. Select **Standard Fit** under **Fitter Effort**.
6. Click **OK**.

Add Pin, DQ Group, and IO Standard Assignments

The pin assignment script, **altmemddr_pin_assignments.tcl**, sets up the I/O standards for the DDR2 SDRAM interface. It also does the DQ group assignment for the Fitter to place them correctly. However, the pin assignment does not create a clock for the design. You need to create the clock for the design.

To run the **altmemddr_pin_assignments.tcl** script to add the pin, I/O standards, and DQ group assignments to the design example, perform the following steps:

1. On the Tools menu, click **Tcl Scripts**.
2. Select **altmemddr_pin_assignments.tcl**.
3. Click **Run**.

If you require pin name changes (prefix changes only), then in the Quartus II Pin Planner, perform the following steps:

1. On the Assignment menu, click **Pin Planner**.
2. Right-click in any area under **Node Name** and select **Create/Import Megafunction**, refer to Figure 6–15.
3. Select **Import an existing custom megafunction** and select the **<variation name>.ppf** file.
4. In the **Instance name** field, type the prefix that you want to use, refer to Figure 6–16 on page 6–25.

Figure 6–15. Create/Import Megafunction in Pin Planner

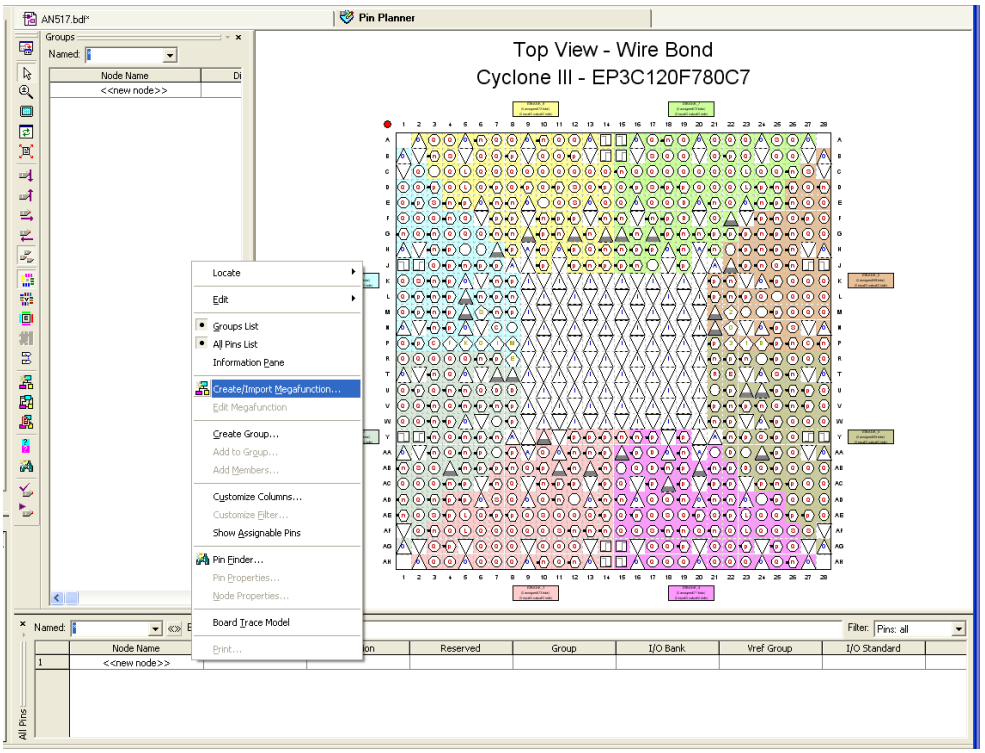
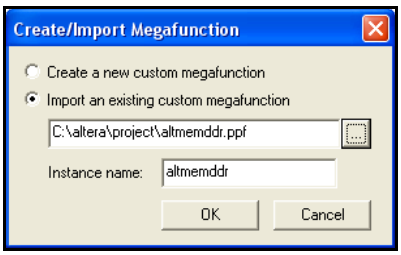


Figure 6–16. Adding Prefix



Alternatively, you may edit the **altmemddr_pin_assignments.tcl** file to change the prefix, and assign the pins without the **_from_to_** pin names, by following these steps:

1. Open the **altmemddr_pin_assignments.tcl** file.
2. To create assignments that have **_from_to_** pin names, change to the following code:

```
switch $sopc_mode {
  YES {
    set output_suffix _from_the_${instance_name}
    set bidir_suffix _to_and_from_the_${instance_name}
    set input_suffix _to_the_${instance_name}
  }
  default {
    set output_suffix ""
    set bidir_suffix ""
    set input_suffix ""
  }
}

to

switch $sopc_mode {
  NO {
    set output_suffix _from_the_${instance_name}
    set bidir_suffix _to_and_from_the_${instance_name}
    set input_suffix _to_the_${instance_name}
  }
  default {
    set output_suffix ""
    set bidir_suffix ""
    set input_suffix ""
  }
}
```

3. Type your preferred prefix in the `pin_prefix` variable. For example, to add prefix **mem_**, change following code:

```
if {![info exists set_prefix]}{set pin_prefix "my mem_"}
```

4. Save the **altmemddr_pin_assignments.tcl** file.
5. Run the **.tcl** script.

For more information about the DDR, DDR2, and DDR3 SDRAM high-performance controllers pin naming, refer to the *DDR and DDR2 SDRAM Controller with ALTMEMPHY IP User Guide* and the *DDR3 SDRAM Controller with ALTMEMPHY IP User Guide* in volume 3 of the *External Memory Interface Handbook*.

Pin Location Assignments

To assign pin locations, perform the following steps:

1. On the Processing menu, point to **Start** and click **Start & Synthesis**.

2. Assign all of your pins, so the Quartus II software fits your design correctly and gives correct timing analysis. To assign pin locations for the Cyclone III development board, run the Altera-provided **C3_Dev_DDR2_Sopc_Pin_Location.tcl** file or manually assign pin locations by using either the Pin Planner or Assignment Editor.

   The SDRAM high-performance controller auto-generated scripts do not make any pin location assignments.

   If you are at the design exploration phase of your design cycle and do not have any PCB defined pin locations, you should still manually define an initial set of pin constraints, which can become more specific during your development process.

To manually assign pin locations in the Pin Planner, perform the following steps:

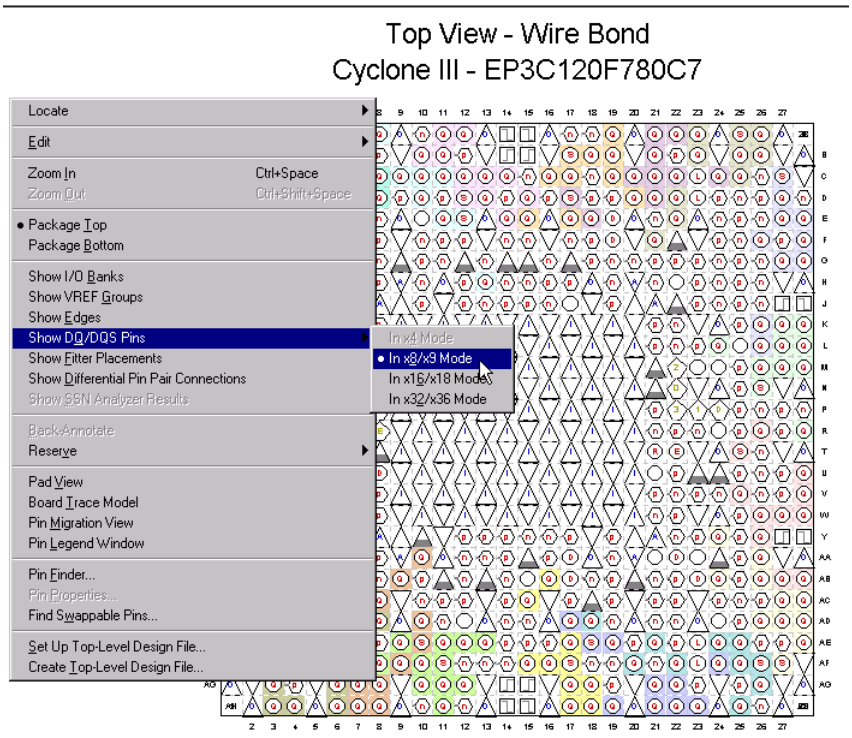
1. On the Assignments menu, click **Pin Planner**.
2. Assign DQ and DQS pins.
   a. To select the device DQS pin groups that the design uses, assign each DQS pin in your design to the required DQS pin in the Pin Planner. The Quartus II Fitter then automatically places the respective DQ signals onto suitable DQ pins within each group. To see DQS groups in the Pin Planner, right click, select **Show DQ/DQS Pins**, and click **In x8/x9 Mode**. The Pin Planner shows each DQS group in a different color and with a different legend: S = DQS pin, Sbar = DQSn pin and Q = DQ pin, refer to Figure 6–17 on page 6–28.

      Most DDR2 SDRAM devices operate in ×8/×9 mode, however as some DDR2 SDRAM devices operate in ×4 mode, refer to your specific memory device datasheet.

   b. Select the DQ mode to match the DQ group width (number of DQ pins/number of DQS pins) of your memory device. DQ mode is not related to the memory interface width.

DQ group order and DQ pin order within each group is not important. However, you must place DQ pins in the same group as their respective strobe pin.

Figure 6–17. Quartus II Pin Planner, Show DQ/DQS Pins, In x8/x9 Mode



3. Place the DM pins within their respective DQ group.
4. Place the address and control/command pins on any spare I/O pins ideally within the same bank or side of the device as the mem_clk pins.
5. Ensure the mem_clk pins use any regular adjacent I/O pins—ideally differential I/O pairs for the CK/CK# pin pair. To identify differential I/O pairs, right-click in Pin Planner and select **Show Differential Pin Pair Connections** (**DIFFIO** in Pin Planner). Pin pairs show a red line between each pin pair.

For Cyclone III and Cyclone IV devices, the mem_clk[0] pin cannot be located in the same row and column pad group as any of the interfacing DQ pins.

The DDR2 SDRAM in Stratix III and Stratix IV devices with differential DQS mode require the mem_clk[0]/mem_clk_n[0] pin on a DIFFIO_RX capable pin pair. The DDR3 SDRAM interfaces in Stratix III and Stratix IV devices require the mem_clk[0]/mem_clk_n[0] pin in any DQ or DQS pin pair with DIFFIO_RX capability.

For more information about memory clock pin placement and external memory pin selection, refer to the *Device and Pin Planning* section in volume 2 of the *External Memory Interface Handbook*.

6. Place the `clock_source` pin on a dedicated PLL clock input pin with a direct connection to the SDRAM controller PLL/DLL pair—usually on the same side of the device as your memory interface. This recommendation reduces PLL jitter, saves a global clock resource, and eases timing and fitter effort.
7. Place the `global_reset_n` pin (like any high fan-out signal) on a dedicated clock pin.

For more information on how to use the Quartus II Pin Planner, refer to the *I/O Management* chapter in volume 2 of the *Quartus II Handbook*. For more information on Cyclone III external memory pin selection, refer to the *External Memory Interface in Cyclone III Devices* chapter in the *Cyclone III Device Handbook*.

Enter Board Trace Delay Model

For accurate I/O timing analysis, the Quartus II must be aware of the board trace and loading information. This information should be derived and refined during your PCB development process of prelayout (line) simulation and finally post-layout (board) simulation. For the external memory interfaces that use memory modules (DIMMs), the board trace and loading information should include the trace and loading information of the module in addition to the main and host platform, which you can obtain from your memory vendor.

To include the board trace information, perform the following steps:

1. In the Pin Planner, select the pin or group of pins that you want to enter the information for.
2. Right-click and select **Board Trace Model**.

Figure 6–18 on page 6–30 shows the board trace model. Enter the values such as the trace length, capacitance/length, inductance/length, pull-up, pull-down and others based on your board.

Figure 6–18. Board Trace Model

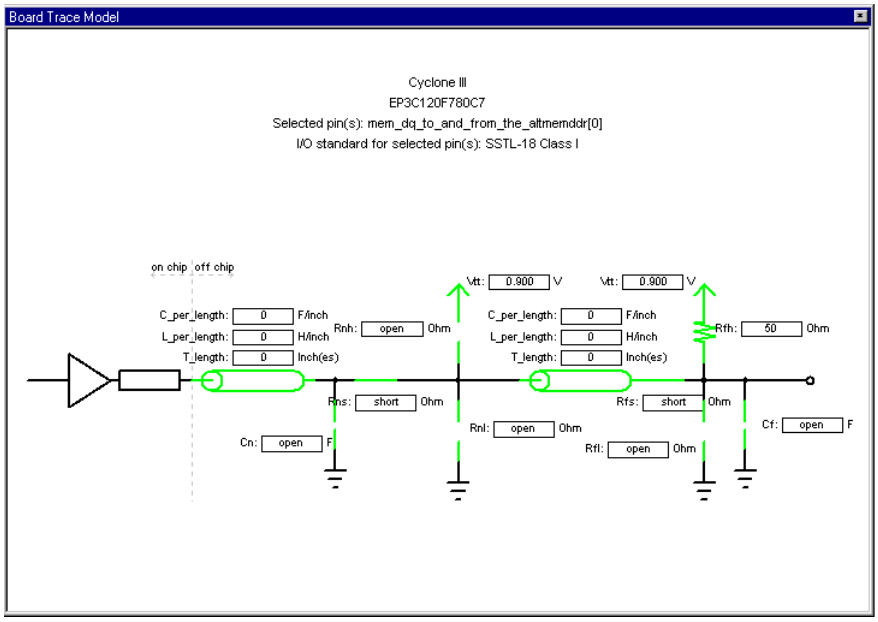


Table 6–3 shows the board trace model parameters for the Cyclone III development board.

Table 6–3. Cyclone III Development Board Trace Model Summary

Net	Near (FPGA End of Line)						Far (Memory End of Line)				
	Length	C_Per_ Length	L_Per_ Length	Cn	Rns	Rnh	Length	C_Per_ Length	L_Per_ Lenght	Cf	Rfh/Rfp
Address *(1)*	1.764	3.8p	8.0n	—	—	56	0.456	3.6p	8.0n	4.0p	—
CLK	1.872	3.4p	8.4n	—	—	—	0.566	3.4p	8.4n	4.0p	—
CKE/CSn	1.862	3.7p	8.1n	—	—	56	0.418	3.6p	8.0n	4.0p	—
ODT	1.527	3.8p	8.0n	—	—	56	0.449	3.8p	8.0n	4.0p	—
DQS *(2)*	1.165	3.6p	8.0n	—	—	56	0	0p	0n	3.5p	—

Note to Table 6–3:

(1) Address = `addr`, `ba`, `we#`, `ras#`, and `cas`.

(2) `DQS` = to `DM`, `DQ`, and `DQS` pins.

Altera recommends you use the **Board Trace Model** assignment for all DDR, DDR2 and DDR3 SDRAM interface signals. To apply board trace model assignments for the Cyclone III development board, run the Altera-provided **C3_Dev_DDR2_BTModels.tcl** file or manually assign virtual pin assignments using the Quartus II Pin Planner.

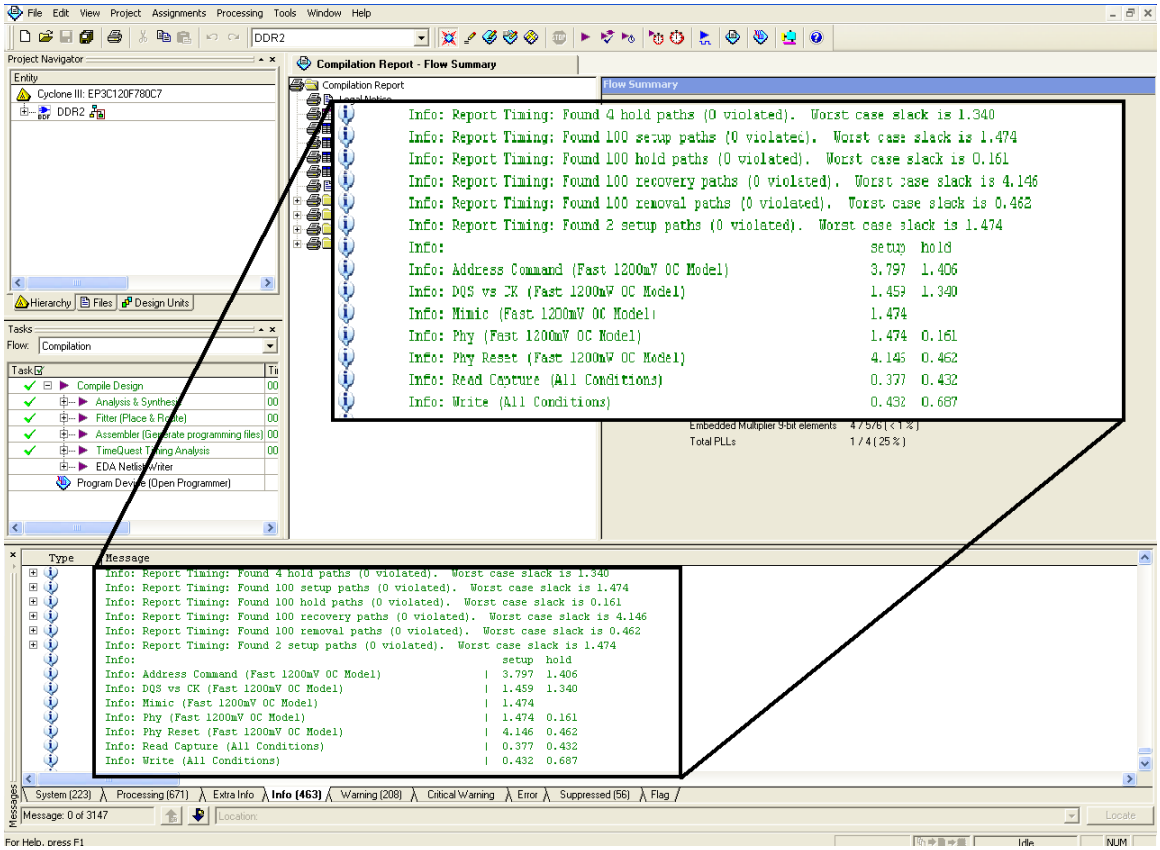
Compile the Design

To compile the design, on the Processing menu, click **Start Compilation**.

After successfully compiling the design, the Quartus II software automatically runs the **altmemddr_phy_report_timing.tcl** file, which produces a timing report for the design together with the compilation report.

Figure 6–19 shows the timing margin report in the message window in the Quartus II software.

Figure 6–19. Timing Margin Report in the Quartus II Software



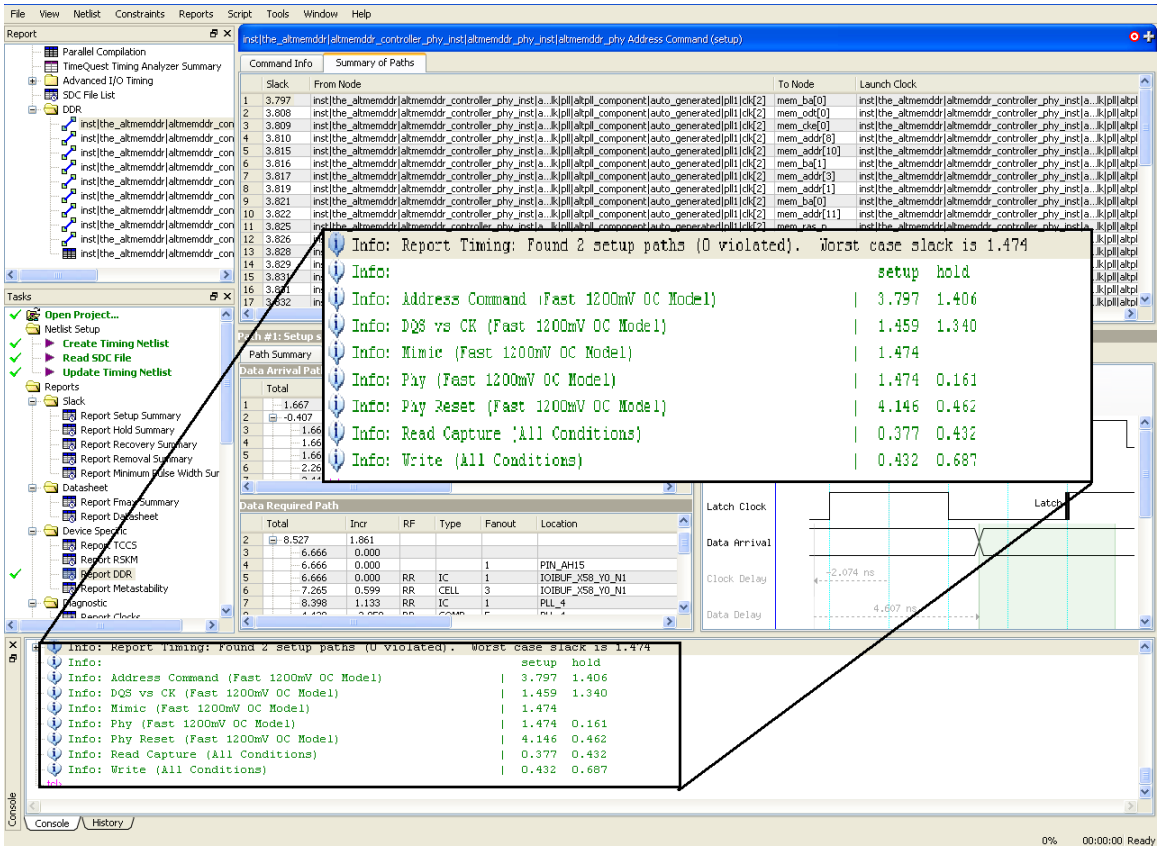
You may also run the report timing script in the TimeQuest Timing Analyzer window, To run the timing script, perform the following steps:

1. On the Tools menu, click **TimeQuest Timing Analyzer**.
2. Double-click **Update Timing Netlist** in the **Tasks** pane, which automatically runs **Create Timing Netlist** and **Read SDC File**. After a task is executed, it turns green.
3. After completing the tasks, run the report timing script by going to the Script menu and clicking **Run Tcl Script**.

Alternatively, you can directly double-click on **Report DDR** in the **Tasks** pane to get the timing report.

Figure 6–20 shows the timing margin report in the TimeQuest Timing Analyzer window after running the report timing script. The results are the same as the Quartus II software results.

Figure 6–20. Timing Margin Report in the TimeQuest Timing Analyzer



Incorporate the Nios II IDE

You can now add test code to the project's Nios II processor and use this program to run some simple tests.

When doing memory tests, you must read and write from the external DDR SDRAM and not from cached memory. The following three methods avoid using Nios II cached memory:

- Use a Nios II processor that does not have cache memory, like the Nios II/s used in this example project.
- Use the `alt_remap_uncached` function.

Launch the Nios II IDE

To launch the Nios II IDE, perform the following steps:

1. One the Windows Start menu, point to **All Programs**, **Altera**, **Nios II EDS** <*version*>, **Legacy Nios II Tools**, and then click **Nios II** <*version*> **IDE**.

   If it is the first time you have run the IDE, click **Workbench**.

2. On the File menu, click **Switch Workspace** and select your project directory.
3. On the File menu, point to **New** and click **Project**.
4. Expand **Altera Nios II** select **Nios II C/C++ Application** and click **Next**.
5. Select **Blank Project** under Select Project Templates and click **Next**.
6. Ensure that SOPC Builder System **.ptf** is *<your project path>*/*<your sopc top>*.**ptf**.
7. Click **Next** and click **Finish**.
8. In Windows Explorer, drag Altera-supplied **DDR_TEST.c** to the **blank_project_0** directory.

To see the **DDR_TEST.c** example test program, refer to "Example DDR_TEST.c Test Program File" on page 6–45.

This program consists of a simple loop that takes commands from the JTAG UART and executes them. Table 6–4 shows the commands that are sent to the Nios II C code.

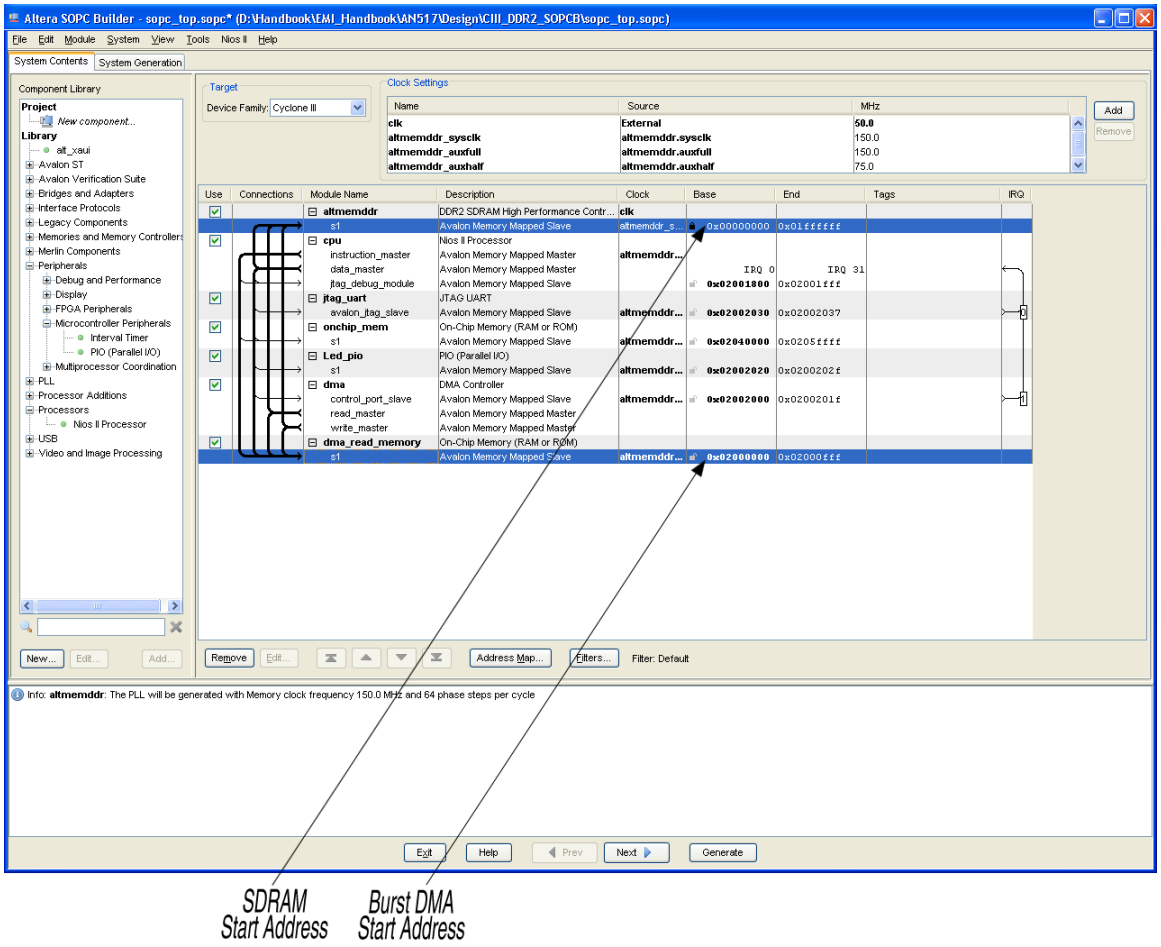
The commands must be in the following format and the switches A, B, C, and D must be entered in upper case. Both address and data strings must always be 8 characters long.

Table 6–4. Commands

Command	Format	Description	Example
Switch A	Switch | Data	Controls the LEDs.	`A000000FF`. The last two bytes control all eight LEDs. The LEDs are active low on the board.
Switch B	Switch | Address | Data	Performs a single write to an address offset location in memory.	Enter `B`, followed by `00000001`, then `00000001`, writes `00000001` at SDRAM memory address location `00000001`.
Switch C	Switch | Address	Performs a single read to an address offset location in memory.	Enter `C`, followed by `00000001`, reads the contents of the memory at SDRAM address offset location `000000010`.
Switch D	Switch | From Address | To Address	Performs an incremental write to the first address location, followed by a DMA burst transfer from the first address location to the second address location. The burst is a fixed length of 512 words or 2048 bytes.	`D0200000000000000` loads the DMA read memory with the incrementing pattern, then DMA burst transfers it to the SDRAM high-performance controller.

You can read the SOPC Builder component address locations directly from the SOPC Builder Window, refer to Figure 6–21.

Figure 6–21. SOPC Builder Component Address



Set Up the Project Settings

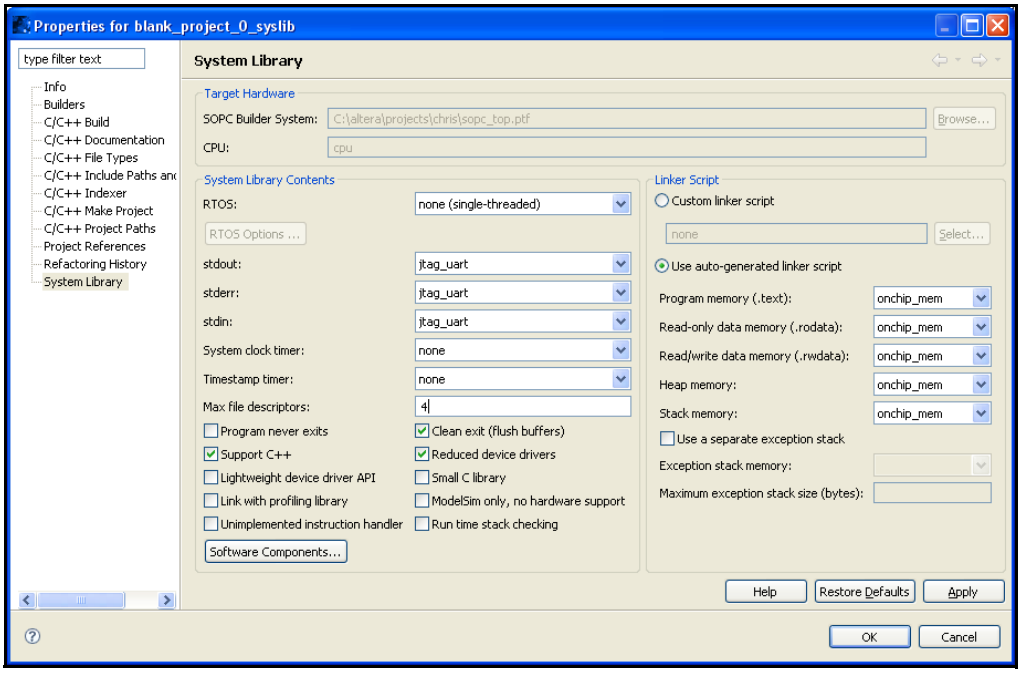
To set up the Nios II project settings, perform the following steps:

1. In the **Nios II C/C++ Projects** tab, right-click **blank_project_0** and click **System Library Properties.**
2. Click **System Library** in the list on the left side of the **Properties** dialog box.
3. To optimize the footprint size of the library project, under **System Library Contents**, turn on **Reduced device drivers**.
4. Under **Linker Script**, for all the memory options select **onchip_mem**. The on-chip memory is the target memory space for the executable file.

5. To reduce the memory size allocated for the system library, for **Max file descriptors** type 4.

   Figure 6–22 shows the **Properties** dialog box.

Figure 6–22. Nios II Project Settings



6. Click **OK**.

Perform RTL or Functional Simulation (Optional) with Nios II

Perform the following steps to run RTL or functional simulation using the Nios II processor:

1. From the Nios II IDE, on the Run menu click **RUN**.
2. Under Nios II ModelSim, select **blank_project_0**.
3. Specify your ModelSim installation path in **ModelSim Path**.
4. Click **Run**. ModelSim is launched.
5. Type `s` in the ModelSim Transcript window to run the **s** macro to load the design.
6. Type `w` in the ModelSim Transcript window to run the **w** macro to display the waveform.
7. Type `jtag_uart_drive` to launch the interactive terminal.
8. Type `Run -all` to run the design.

9. In the interactive terminal, type your desired operation, data, and address, refer to Figure 6–23.

Figure 6–23. The JTAG UART Interactive Terminal

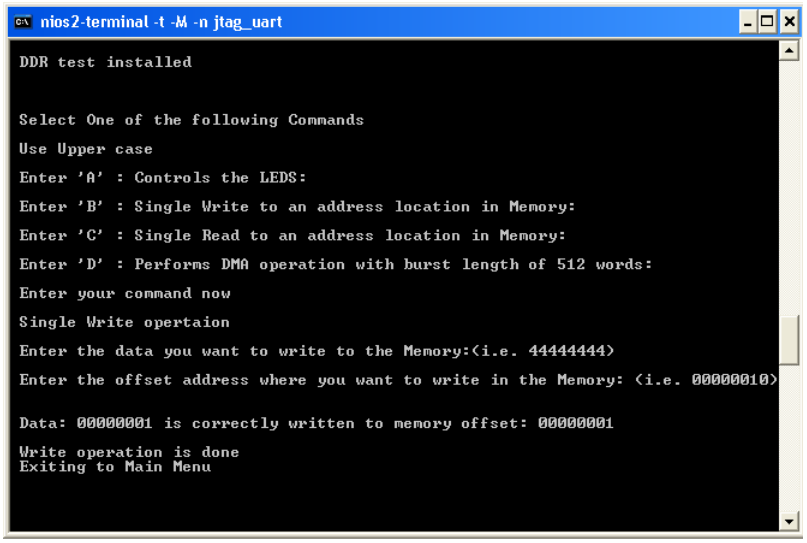
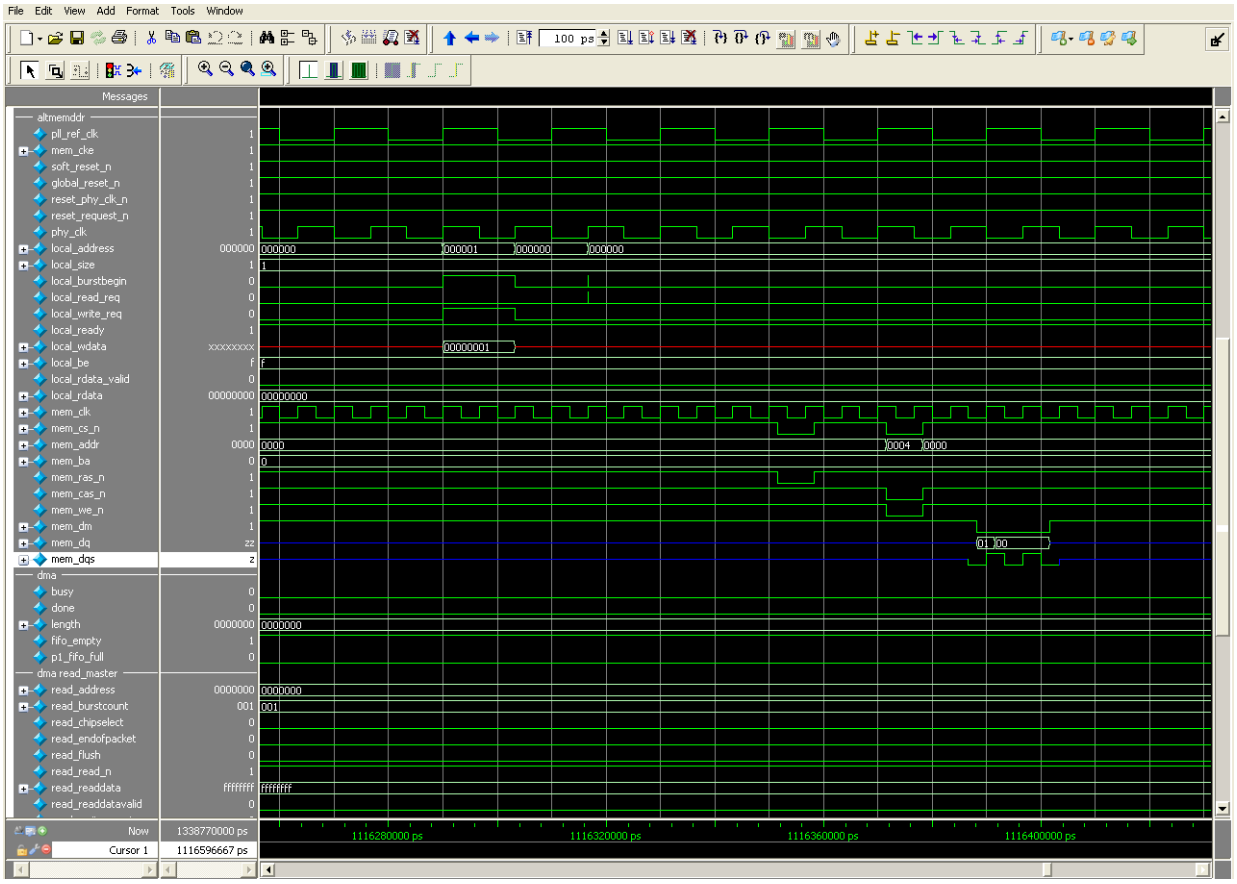


Figure 6–24 shows a single write operation where data `00000001` is written to address `00000001`.

Figure 6–24. Waveform for Single Write Operation



Verify Design on a Development Platform

The SignalTap II Embedded Logic Analyzer shows read and write activity in the system.

For more information about using the SignalTap II Embedded Logic Analyzer, refer to the *Design Debugging Using the SignalTap II Embedded Logic Analyzer* chapter in the *Quartus II Handbook*, *AN 323: Using SignalTap II Embedded Logic Analyzers in SOPC Builder Systems* and *AN 446: Debugging Nios II Systems with the SignalTap II Logic Analyzer*.

To add the SignalTap II Embedded Logic Analyzer, perform the following steps:

1. On the Tools menu, click **SignalTap II Logic Analyzer**.
2. In the **Signal Configuration** window next to the **Clock** box, click **...** (Browse Node Finder).
3. Type `*phy_clk` in the **Named** box, for **Filter** select **SignalTap II: pre-synthesis** and click **List**.

4. Select **sopc_top:inst|altmemddr:the_altmemddr|phy_clk** in **Nodes Found** and click **>** to add the signal to **Selected Nodes**.
5. Click **OK**.
6. Under Signal Configuration, specify the following settings:
   - For **Sample depth**, select **512**
   - For **RAM type**, select **Auto**
   - For **Trigger flow control**, select **Sequential**
   - For **Trigger position**, select **Center trigger position**
   - For **Trigger conditions**, select **1**
7. On the Edit menu, click **Add Nodes**.
8. Search for specific nodes by typing `*local*` in the **Named** box, for **Filter** select **SignalTap II: pre-synthesis** and click **List**.
9. In **Nodes Found**, select the following nodes and click **>** to add to **Selected Nodes**:
   - **local_address**
   - **local_rdata**
   - **local_rdata_valid** (alternative trigger to compare read/write data)
   - **local_read_req**
   - **local_ready**
   - **local_wdata**
   - **local_write_req** (trigger)

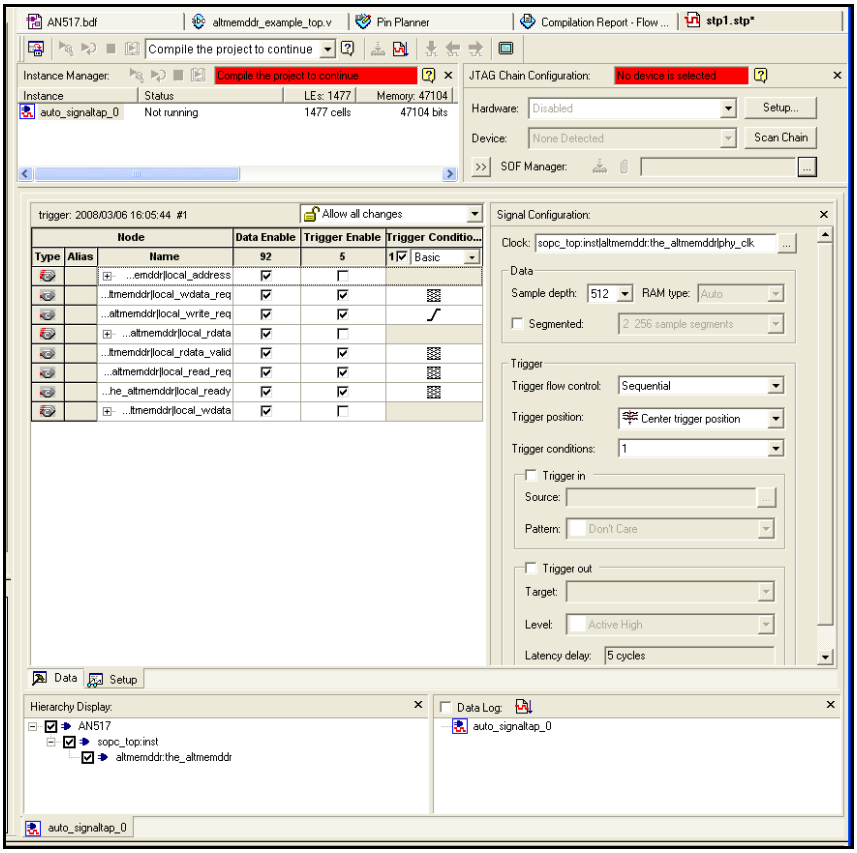
   
   Do not add any DDR SDRAM interface signals to the SignalTap II Embedded Logic Analyzer. The load on these signals increases and adversely affects the timing analysis.

10. Click **OK**.
11. To reduce the SignalTap II logic size, turn off **Trigger Enable** on the following bus signals:
    - **local_address**
    - **local_rdata**
    - **local_wdata**

12. Right-click **Trigger Conditions** for the `local_write_req` signal and select **Rising Edge**.

    Figure 6–25 shows the completed SignalTap II Embedded Logic Analyzer.

Figure 6–25. SignalTap II Embedded Logic Analyzer



13. On the File menu, click **Save**, to save the SignalTap II **.stp** file to your project.

    If you get the message **Do you want to enable SignalTap II file "stp1.stp" for the current project**, click **Yes**.

Compile the Project

Once you add signals to the SignalTap II Embedded Logic Analyzer, recompile your design, on the Processing menu, click **Start Compilation**.

Verify Timing

Once the design compiles, ensure that the TimeQuest timing analysis passes successfully. In addition to this FPGA timing analysis, check your PCB or system SDRAM timing.

To run timing analysis, run the <*variation name* >**_phy_report_timing.tcl** script by performing the following steps:

1. On the Tools menu, click **Tcl Scripts**.

2. Select <*variation name* >**_phy_report_timing.tcl** and click **Run**.

Connect the Development Board

Connect the Cyclone III development board to your computer.

For more information on the Cyclone III development board, refer to the *Cyclone III Development Kit User Guide*.

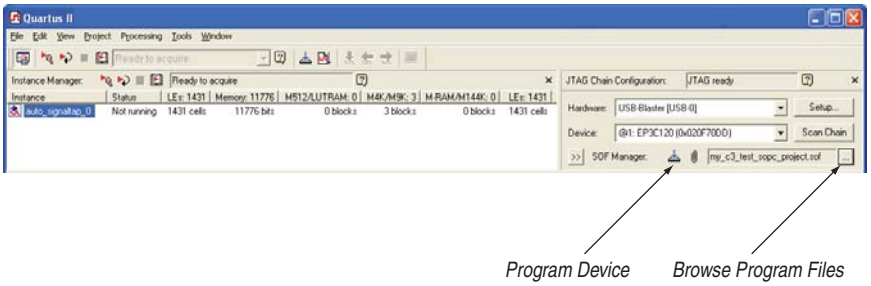
Download the Object File

On the Tools menu, click **SignalTap II Logic Analyzer**. The SignalTap II dialog box appears.

The SRAM Object File (SOF) Manager should contain the <*your project name*>**.sof** file. To add the correct file to the SOF Manager, perform the following steps:

1. Click **...** to open the **Select Program Files** dialog box, refer to Figure 6–25.
2. Select <*your project name*>**.sof**.
3. Click **Open**.
4. To download the file, click the **Program Device** button.

Figure 6–26. Install the SRAM Object File in the SignalTap II Dialog Box



Verify Design with Nios II

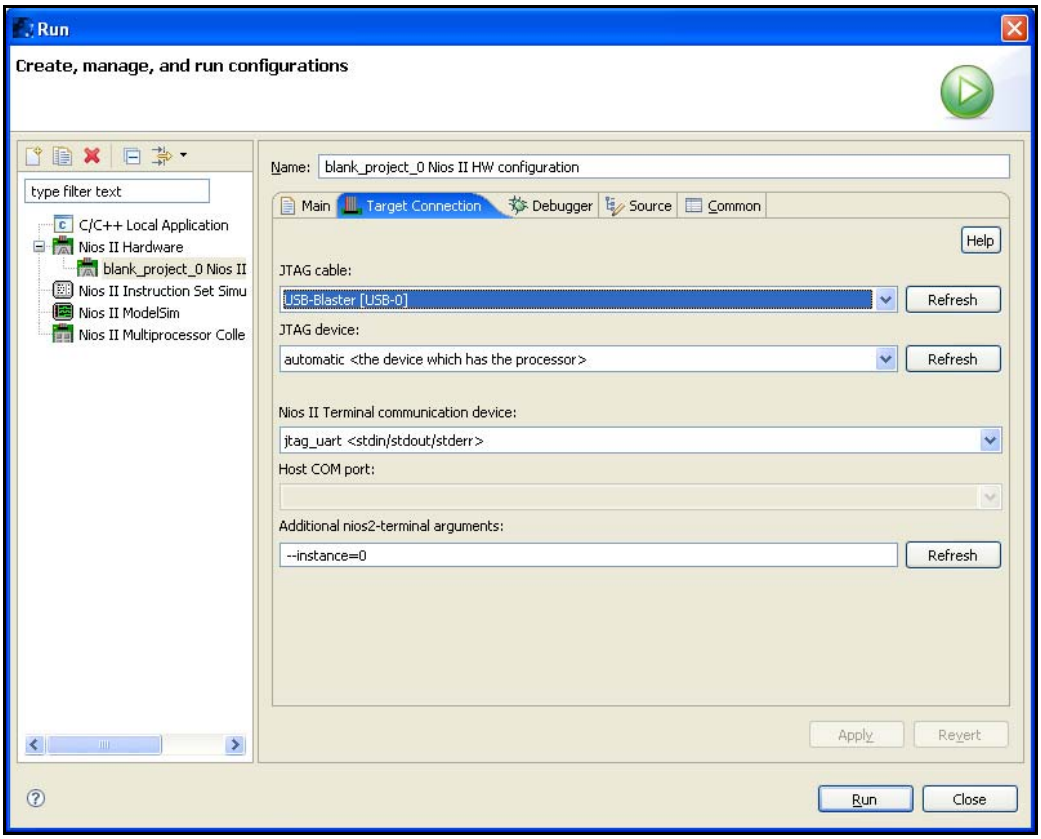
Right-click on **blank_project_0**, point to **Run As**, and click **Nios II Hardware** for the the Nios II C/C++ IDE to compile the example test program.

If you have more than one JTAG download cable connected to your computer you may see an JTAG error. If you receive a JTAG error, perform the following steps:

1. From the Nios II IDE, on the Run menu click **RUN**. Click the **Target Connection** tab and correct the **JTAG cable connection**, refer to Figure 6–27.

2. Right click on **blank_project_0**, point to **Run As**, and click **Nios II Hardware**.

Figure 6–27. JTAG Download Target Connection Settings



Test the System

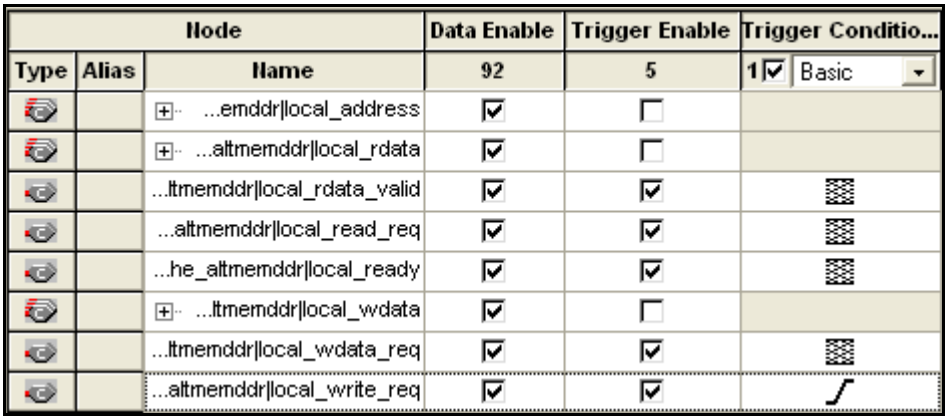
Perform the following tests to verify your system is operating correctly.

Set SignalTap II Trigger to local_write_req

Use the SignalTap II Embedded Logic Analyzer to capture write activity. To show write activity, perform the following steps:

1. In the **Setup** tab, select to trigger on a rising edge of `local_write_req`, refer to Figure 6–28.

Figure 6–28. Set Trigger for local_write_req Signal

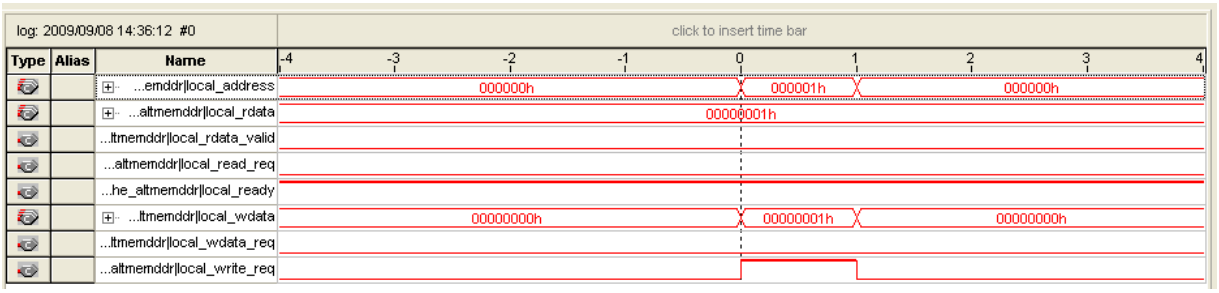


2. Click **Run Once Analysis** to capture the write request.
3. Type the following command in the Nios II command console:

```
B0000000010000001
```

4. Return to the SignalTap II Embedded Logic Analyzer window and check that at time 0 (refer to Figure 6–29):
   - `local_write_req` signal goes high for a single cycle
   - `local_wdata` shows `00000001h`
   - `local_address` shows `000001h`

Figure 6–29. Waveform for Write Request

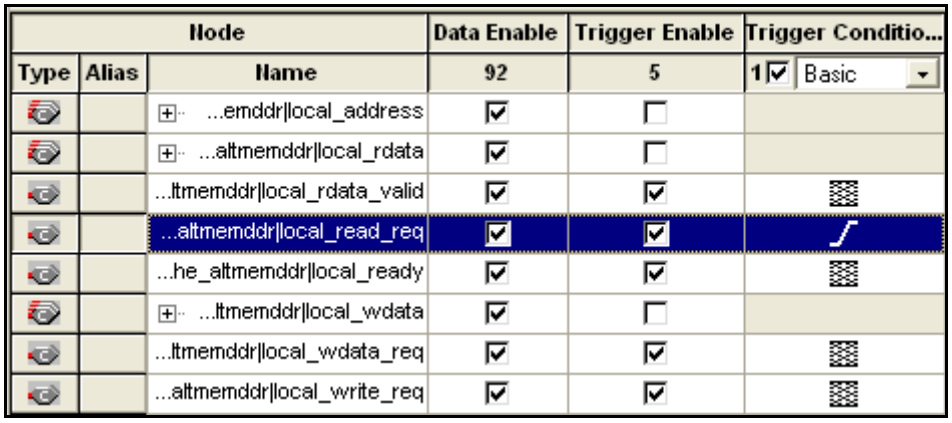


Set SignalTap II Trigger to local_read_req

Use the SignalTap II Embedded Logic Analyzer to capture read activity. To show read activity, perform the following steps:

1. In the **Setup** tab, select to trigger on a rising edge of `local_read_req`, refer to Figure 6–30.

Figure 6–30. Set Trigger for local_read_req Signal



2. Click **Run Once Analysis** to capture the read request
3. Type the following command in the Nios II command console:

```
C00000001
```

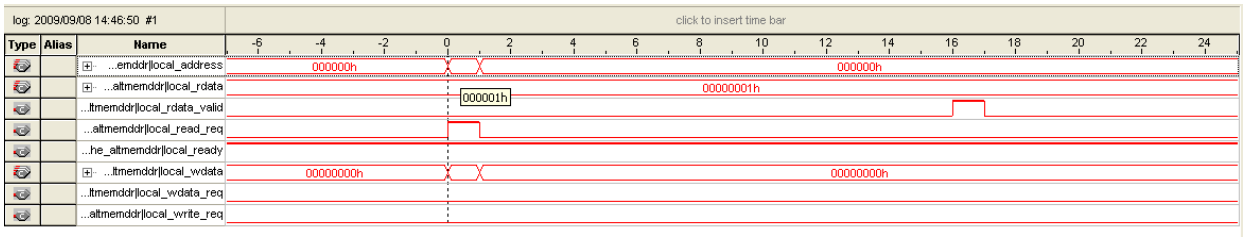
4. Return to the SignalTap II Embedded Logic Analyzer window and check that at time 0 (refer to Figure 6–22):

- `local_read_req` signal goes high
- `local_address` shows `000001h`

Several clock cycles later, depending on the system read latency:

- `local_rdata_valid` goes high for one cycle
- `local_rdata` shows `00000001h`

Figure 6–31. Waveform for Read Request

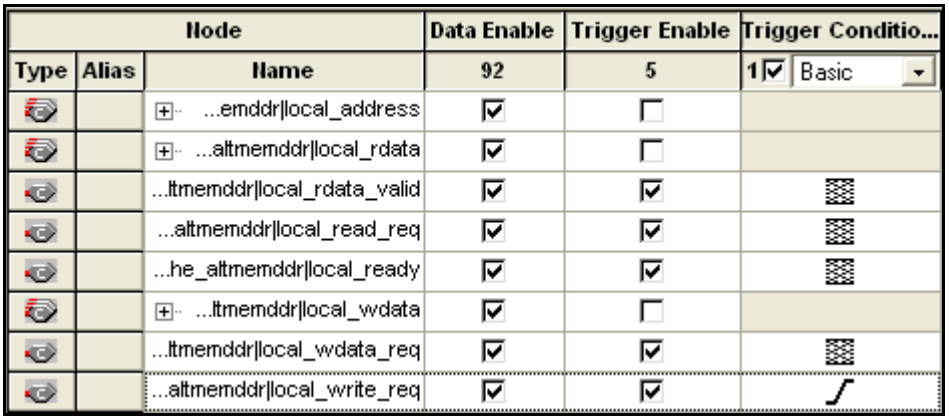


Test Burst Write Operation

Use the SignalTap II Embedded Logic Analyzer to capture write activity. To show write activity, perform the following steps:

1. In the **Setup** tab, select to trigger on a rising edge of `local_write_req`, refer to Figure 6–32.

Figure 6–32. Set Trigger for local_write_req Signal



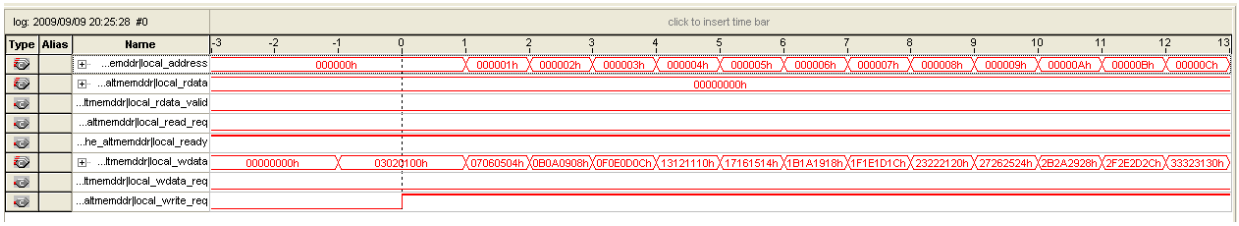
2. Click **Run Once Analysis** to capture the write request.
3. Type the following command in the Nios II command console:

```
D0200000000000000
```

4. Return to the SignalTap II Embedded Logic Analyzer window and check that at time 0 (refer to Figure 6–22):
   - `local_write_req` signal goes high for multiple cycles
   - `local_wdata` shows `03020100h` followed by `07060504h` and so on
   - `local_address` shows `000000h` followed by `000002h` and so on

The write data is first to last, most significant byte (MSB) to the least significant byte (LSB) count format. For example, a count of `00,01,02,03` = `03020100h`.

Figure 6–33. Waveform for Burst Write Request



Test Burst Read Operation

Performing a burst read operation is similar to performing a burst write operation (refer to "Test Burst Write Operation" on page 6–43), but with the memory locations swapped. To perform a burst read operation, perform the following steps:

1. In the **Setup** tab, set to trigger on a rising edge of `local_read_req`.
2. Click **Run Once Analysis** to capture the read request:

3. Type the following command in the Nios II command console:

```
D0000000002000000
```

Example DDR_TEST.c Test Program File

```
#include<stdio.h>
#include"sys/alt_dma.h"
#include "sys/alt_cache.h"
#include "system.h"
#include "altera_avalon_dma_regs.h"
#define length 512
int to_hex(char* pkt)
{
    unsigned int value[8];
    unsigned int value1=0;
    unsigned int q;
    for (q=0;q<=7;q++)
       {
            value[q]=(pkt[q]>0x39)?(pkt[q]-0x37):(pkt[q]-0x30);

        if (q==0)
            {
            value1=(value1+value[q]);
            }
        else
            {
            value1=((value1<<4)+value[q]);
             }

     }
        return value1;
}

/*****************************************************************
*  Function: Main_menu
*
*  Purpose: Prints the main menu commands
*
*****************************************************************/
static void Main_menu(void)
{
  printf("\n\n");
  printf(" \n Select One of the following Commands \n");
  printf( "\n Use Upper case \n");
  printf(" \n Enter 'A' : Controls the LEDS:\n");
  printf(" \n Enter 'B' : Single Write to an address location in
Memory:\n");
  printf(" \n Enter 'C' : Single Read to an address location in
Memory:\n");
  printf(" \n Enter 'D' : Performs DMA operation with burst length of
512 words:\n");
  printf( "\n Enter your command now \n");

}
 /*****************************************************************
*  Function: LED_Control
*
*  Purpose: Controls the LEDs..
*
*****************************************************************/
void LED_Control(void)
{
```

```
  unsigned int led_value;
  unsigned char led[8];
  printf(" \n LED Test operation \n");
  printf( "\n Enter the value in Hex you want to write to the LEDs:(i.e.
000000FF)\n");
  printf(" \n The last two bytes control all eight LEDs. \n");
  gets(led);
  led_value=to_hex(&led[0]);
  printf(" \n Value to be displayed on LEDs is: %08x \n",led_value);
  IOWR_32DIRECT(LED_PIO_BASE,0, led_value);

}

 /*****************************************************************
*  Function: Single_Write
*
*  Purpose: Performs a single write to an address location in memory.
*
*****************************************************************/
void Single_Write(void)
{
  unsigned char write_offset[8];
  unsigned char data[8];
  unsigned int DDR_write_OFFSET_ADDRESS;
  unsigned int write_data;
  printf(" \n Single Write operation \n");
  printf( "\n Enter the data you want to write to the Memory:(i.e.
44444444) \n");
  gets(data);
  write_data=to_hex(&data[0]);
  printf( "\n Enter the offset address where you want to write in the
Memory: (i.e. 00000010)\n");
  gets(write_offset);
  DDR_write_OFFSET_ADDRESS = to_hex(&write_offset[0]);
  if
((DDR_write_OFFSET_ADDRESS<0)||(DDR_write_OFFSET_ADDRESS>=(ALTMEMDDR_S
PAN/4)))
  {
    printf(" \n Invalid Offset \n");
    printf( "\n You have entered wrong offset address : \n");
    return;
  }
  IOWR(ALTMEMDDR_BASE,DDR_write_OFFSET_ADDRESS,write_data);
  if (IORD(ALTMEMDDR_BASE,DDR_write_OFFSET_ADDRESS)==write_data)
  {
    printf("\n Data: %08x is correctly written to memory offset: %08x
\n", write_data,DDR_write_OFFSET_ADDRESS);
    printf("\n Write operation is done \n");
  }
  else
  {
     printf("\n Write operation is Failed \n");
  }
}
 /*****************************************************************
*  Function: Single_Read
*
*  Purpose: Performs a single read to an address location in memory
*
*****************************************************************/
void Single_Read(void)
{
  unsigned char read_offset[8];
  unsigned int DDR_read_OFFSET_ADDRESS;
```

```
  unsigned int read_data;
  printf(" \n Single Read operation \n");
  printf( "\n Enter the offset address from where you want to read in
the Memory:(i.e. 00000010) \n");
  gets(read_offset);
  DDR_read_OFFSET_ADDRESS =  to_hex(&(read_offset[0]));
  if ((DDR_read_OFFSET_ADDRESS<0) ||
(DDR_read_OFFSET_ADDRESS>=(ALTMEMDDR_SPAN/4)))
  {
    printf(" \n Invalid Offset \n");
  }
  else
  {
    read_data=IORD(ALTMEMDDR_BASE,DDR_read_OFFSET_ADDRESS);
    printf("Read %08x from address %08x
\n",read_data,(ALTMEMDDR_BASE+DDR_read_OFFSET_ADDRESS));
  }
}
/*****************************************************************
*  Function: Verify Operation
*
* Purpose: Compares the memory contents of the read data master and write
data master
*
*****************************************************************/
void Call_verify(unsigned char* source, unsigned char* destination)
{
    if (memcmp(source,destination,(length*4))==0)
    {
     printf("\n DMA operation successful \n");

    }
    else
    {
     printf("\n DMA operation failed \n");
     printf( "\n Please check that DMA is correctly setup \n ");
     }
}
/*****************************************************************
*  Function: DMA_Operation
*
* Purpose: Performs an incremental write to the first address
location, followed by a DMA burst transfer from the first address
location to the second address location.
The burst is a fixed length of 512 words or 2048 bytes.
*
*****************************************************************/

void DMA_operation(void)
{
  unsigned int read_address;
  unsigned char* read_DMA_address;
  unsigned char *write_DMA_address;
  unsigned int write_address;
  unsigned char verify[8];
  unsigned char read[8];
  unsigned char write[8];
  unsigned char status_reg;
  unsigned int i;
  printf(" \n DMA setup operation \n");
  printf( "\n Enter DMA read master address:(i.e. 00000010) \n ");
  gets(read);
  read_address =to_hex(&read[0]);
  read_DMA_address=read_address;
  printf( " \n DMA read master address is %08x \n",read_DMA_address);
```

```
  printf( "\n Enter DMA write master address:(i.e. 00000010) \n ");
  gets(write);
  write_address =to_hex(&write[0]);
  write_DMA_address=write_address;
  printf( " \n DMA write master address is %08x \n",write_DMA_address);
  printf("\n Using %d bytes to verify the DMA operation \n",(length*4));
  printf( "\n Initializing Read memory with incremental data starting
from 00 \n");
  for (i=0;i<=(length*4);i++)
  {
   *read_DMA_address=i;
   read_DMA_address++;
  }
  read_DMA_address=read_address;
  printf( "\n Writing to the DMA control registers \n");
  IOWR_ALTERA_AVALON_DMA_CONTROL(DMA_BASE,0x00000084);
//DMA go bit is set to zero.
  IOWR_ALTERA_AVALON_DMA_STATUS(DMA_BASE, 0x00000000);
//clearing  done bit
  IOWR_ALTERA_AVALON_DMA_RADDRESS(DMA_BASE,read_DMA_address);
  IOWR_ALTERA_AVALON_DMA_WADDRESS(DMA_BASE,write_DMA_address);
  IOWR_ALTERA_AVALON_DMA_LENGTH(DMA_BASE,(length*4));
  printf( "\n Starting DMA engine \n");
  IOWR_ALTERA_AVALON_DMA_CONTROL(DMA_BASE, 0x0000008C);
  printf(" \n DMA operation is in progress \n ");
  status_reg= (IORD_ALTERA_AVALON_DMA_STATUS(DMA_BASE)&
ALTERA_AVALON_DMA_STATUS_DONE_MSK) ;
  if (status_reg ==1)
  {
  printf("\n DMA operation is Done \n ");
  printf("\n Do you want to verify the DMA operation \n");|
  printf("\n Enter Y for yes, N for No \n ");
  gets(verify);
  switch( verify[0])
  {
   case 'N':
       break;
   case 'Y':
       Call_verify(read_DMA_address, write_DMA_address);
       break;
   default :
      printf(" Error: unexpected command\n");
      break;

  }

  }

  else

  {
   printf(" \n DMA operation is in progress \n ");

  }

}

/*****************************************************************

*  Function: Main

*

* Purpose: Output the main menu and selects the app. function.

*

*****************************************************************/
```

```
int main()
{
  char packet[1];
  printf("\n DDR test installed \n");
  while (1)
  {
   Main_menu();
   gets (packet);
   switch(packet[0])
   {
     case 'A' :

       LED_Control();
       printf(" Exiting to Main Menu \n");
       break;

     case 'B' :

       Single_Write();
       printf(" Exiting to Main Menu \n");
       break;

     case 'C':

       Single_Read();
       printf(" Exiting to Main Menu \n");
       break;

     case 'D':

       DMA_operation();
       printf(" Exiting to Main Menu \n");
       break;

     default :
       printf(" Error: unexpected command. Switch was -%C\n",
packet[0]);
       break;

   }               //switch loop closure

  }               // while loop closure

  return 0 ;

}
```

7. Implementing Multiple ALTMEMPHY-Based Controllers

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

This chapter describes the steps to implement multiple memory interfaces where there are independent memory transactions, but the interfaces are operating at the same frequency. Such implementations require multiple instantiations of the PHY, which in turn may necessitate device resource sharing, such as the delay-locked loops (DLLs) and clock networks, and may require extra steps to create the interfaces in a Quartus II project. Multiple memory interfaces may also involve different types of memory standards, for example, if you have DDR2 SDRAM and RLDRAM II interfaces in a single Stratix IV device. Width and depth expansion of a memory interface are not covered in this document as they are natively supported by the PHY. The memory interfaces described in this chapter use the ALTMEMPHY megafunction.

While you may not be able to share one PLL between multiple ALTMEMPHY-based controllers, you may be able to share the static clocks to reduce the number of clock networks used in the design.
You cannot merge dynamic clocks of the ALTMEMPHY megafunction or high-performance controllers. The Quartus II software may not give a warning, but this merging does not work in the hardware.

This chapter assumes that you are familiar with the ALTMEMPHY megafunction and Altera FPGA resources. There is no design example associated with this tutorial.

Before Creating a Design in the Quartus II Software

When creating multiple memory interfaces to be fitted in a single device, first ensure that there are enough device resources for the different interfaces. These device resources include the number of pins, DLLs, PLLs, and clock networks, where applicable. The DLLs, PLLs, and clock network resources from the clock and reset management block can be shared between multiple memory interfaces, but there are sharing limitations that you need to pay close attention to.

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

The following different scenarios exist for the resources sharing DLL and PLL static clocks:

- Multiple controllers with no sharing. You need to check the number of PLLs and DLLs that available in the targeted device and the number of PLLs and DLLs that needed in your design. If you have enough resources for the controllers, you do not need to share the DLL and PLL static clocks.
- Multiple controllers sharing DLL only. If you have enough resources on PLL clocks output and only need to share DLL, ensure that the controllers are running at the same memory clock frequency.
- Multiple controllers sharing PLL static clocks only. Ensure that all the controllers sharing the PLL static clocks are located in the same half of the device, as ALTMEMPHY requires the use of regional and dual-regional clocks, instead of global clocks. Each controller can have their own DLL as the DLL can only access the adjacent sides from its location.
- Multiple controllers sharing DLL and PLL static clocks. The PLL static clocks and DLL can be shared when the controllers are on the same side or adjacent side of the device and provided they are running at the same memory-clock frequency.

After understanding the device resource limitations, determine the resources to share for your multiple controller design. You can use the resources that you determined as a framework for your Quartus II design constraints. If you also have a PCI interface in your design, you need to leave at least one bank (either on the top or the bottom of the device) for the PCI interface. For this particular example interface, you may need to share a DLL.

Creating PHY and Controller in a Quartus II Project

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

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

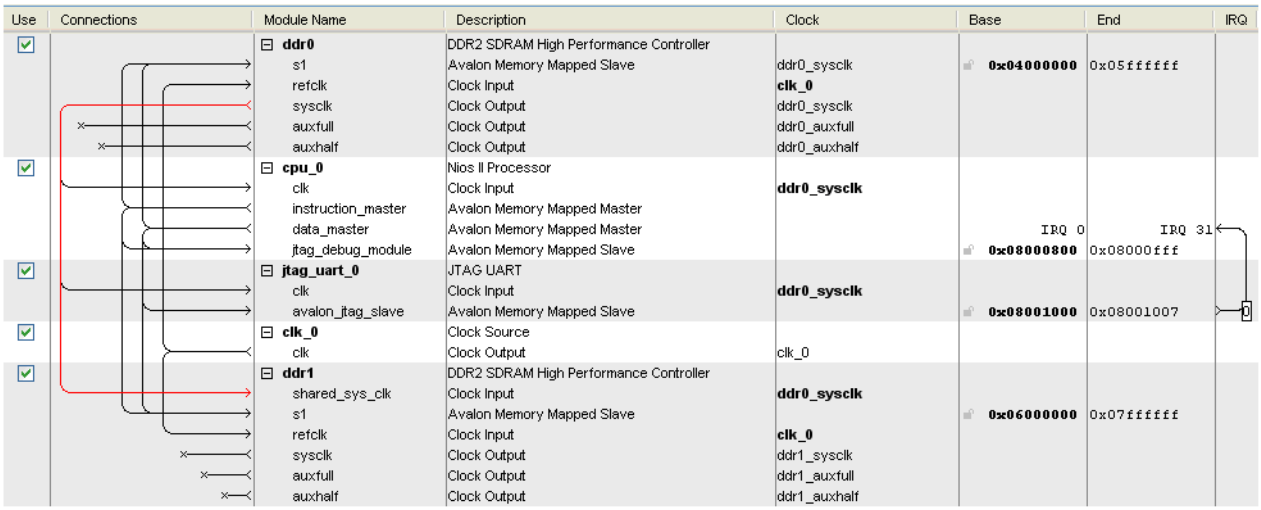
The SOPC Builder flow provides useful validation and automatically generates all the wiring in SOPC Builder system and assignment. For the MegaWizard™ Plug-In Manager flow, you must manually set the assignment for the PLL static clock sharing but this flow gives you more flexibility.

SOPC Builder Flow

The SOPC Builder in version 8.1 and onwards of the Quartus II software allows you to add multiple controllers directly to a new or existing SOPC Builder system. The SOPC Builder supports sharing static PHY clocks between multiple controllers in the SOPC Builder that are running on the same frequency and sharing the same PLL reference clock. To share the static clocks, you must turn on the **Clock source from another controller** option in the **Controller Settings** page of the DDR, DDR2, or DDR3 SDRAM High Performance Controller wizard. This option must be turned on for the

slave controllers. Turning on this option adds a new clock input connection point to the slave controller named `shared_sys_clk`. You must connect the `sys_clk` signal from the master controller to the `shared_sys_clk` signal of the slave controller (Figure 7–1). The **Force Merging of PLL Clock Fanouts** assignment is automatically assigned in the Quartus II software.

Figure 7–1. Connection Between Master and Slave Controller



After the system generation, you must add the **.sdc** file and run the pin assignment Tcl scripts as usual.

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

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

For more information on DDR, DDR2, and DDR3 SDRAM High Performance Controllers, refer to the *DDR and DDR2 SDRAM Controller with ALTMEMPHY IP User Guide* section and the *DDR3 SDRAM Controller with ALTMEMPHY IP User Guide* section in volume 3 of the *External Memory Interface Handbook*.

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

MegaWizard Plug-In Manager Flow

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

If the controllers are generated individually Altera recommends you generate each controller's files in a separate directory. However, you may get compilation errors if you add all the **.qip** files to the project. You may remove the following line in all the **.qip** files except for one to remove the compilation error:

```
set_global_assignment -name VHDL_FILE [file join $::quartus(qip_path)
"auk_ddr3_hp_controller.vhd"]
```

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

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

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

Sharing DLLs

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

When you turn on **Instantiate DLL externally**, the MegaWizard Plug-In Manager generates a file called <*variation_name*>**_phy_alt_mem_phy_dll_**<*device_family*>**.v/.vhd**, where <*device_family*> is:

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

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

If you do not need to share any PLL static clocks, you can continue to "Adding Constraints to the Design" on page 7–9.

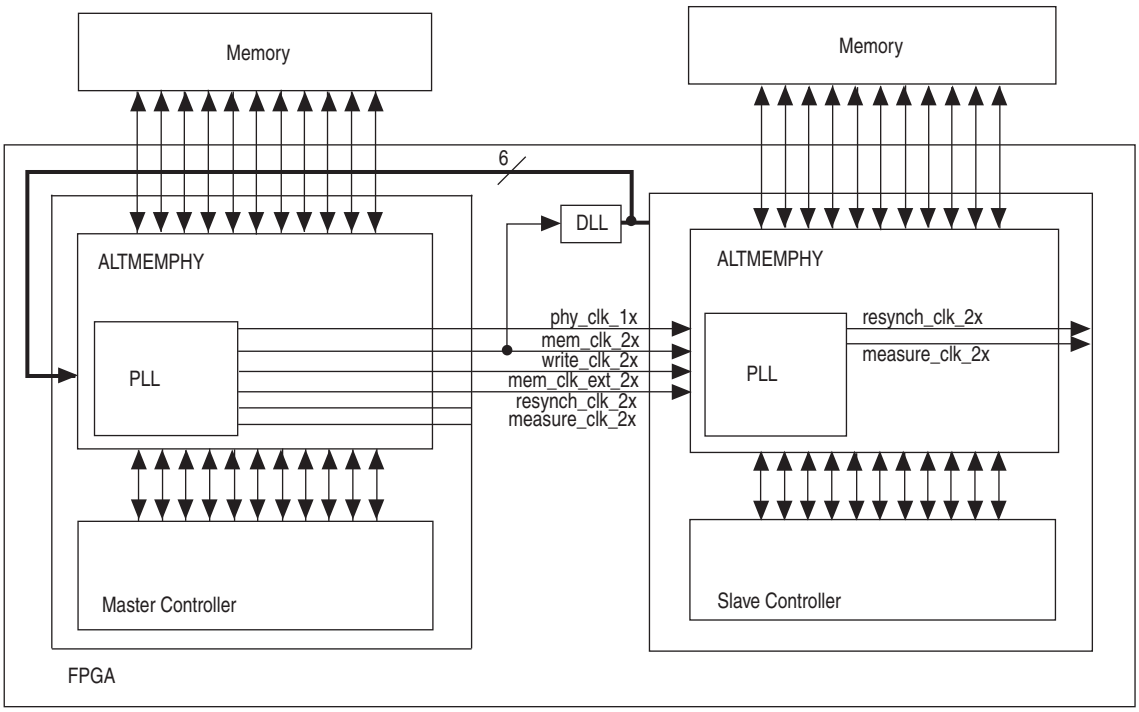
Sharing PLL Clock Outputs or Clock Networks

The ALTMEMPHY sources a system clock, for use in your design, to clock your ALTMEMPHY interface. This same clock is used by Altera High-Performance controllers, SOPC Builder systems, and the Avalon-MM interface connected to the memory controller. If you use more than one similarly configured ALTMEMPHY in a design, you can share these system clocks, which allows the ALTMEMPHY interfaces to operate synchronously to each other. PLL static clock sharing has the following benefits:

- Although a PLL instance is used for each ALTMEMPHY instance, the static clocks are shared, which uses fewer clocking resources
- The multiple ALTMEMPHYs operate on the same system clock so can be directly connected to logic on the same clock (for example, Avalon-MM interface) without any clock domain crossing logic required, which causes an increase in logic usage and read latency

PLL static clock sharing reduces the required number of clocks by up to four clock networks, which are used in the memory interface logic to clock the controller and generate DQ, DQS, CK/CK#, and address and command signals. Figure 7–2 shows a diagram of the connection between the DLL and the PLL for this scheme.

Figure 7–2. Two Controllers Static Clocks with Separate Resynchronization Clocks from Different PLLs



The maximum number of interfaces that can be shared in this scheme is limited by the number of PLLs and the number of pins available in the device. ALTMEMPHY-based memory interfaces may share static clocks between multiple controllers when the following rules are observed:

- All ALTMEMPHYs are full-rate or all half-rate designs
- All ALTMEMPHYs have the same PLL input and output frequencies
- The ALTMEMPHY memory types are compatible. DDR and DDR2 SDRAM can be mixed in the same system; DDR3 SDRAM controllers may be compatible with DDR or DDR2 SDRAM if the static clock phases are identical.

  DDR3 with and without levelling implementations have different static clock phases. Check for the ALTMEMPHY static clock phase compatibility in the clocking tables of the appropriate device and memory interface standard.

- The `ref_clk` input of each ALTMEMPHY PLL must be connected to a clock signal derived from a single clock source. You can use a clock fanout buffer to create separate PLL reference clocks from a single clock source.

- All of the memory controller's circuitry is located in the same two device quadrants
- All of the clocks can be routed as required by their **Global Network Clock Type**. To see the global clocks, look in the Fitter Report > Resource selection > Global and other fast signals.
- The general routing rules defined in the relevant memory standard layout section are met.

Static clocks which can be shared are divided into two categories:

- Static clocks with a fixed phase in the IP (all static clocks except the address and command clock `ac_clk_1x`)
- Static clocks where the phase is set in the IP (the address and command clock `ac_clk_1x`)

Static clocks with a fixed phase in the IP do not require each controller's interface PCB traces to have the same delays. Static clocks with the phase set in the IP have several options where they may be shared between multiple controllers:

- The address and command clock traces all have the same respective delays from the FPGA to each device
- The TPD delay between the address and command signals and the memory clock of each of the interfaces is the same to all devices
- The TPD delay between the address and command signals and the memory clock of each of the interfaces is similar and the optimal address and command phase for each separate interface is within one to two PLL phase steps. Most designs should be able to use a single `ac_clk_1x` clock shared between all controllers and accept less margin.

For multiple controllers, where not all of the memory interface pins are on the same device edge, evaluating whether you can share static clocks is a complex process and there are many possible combinations of pin outs. In this case, you must evaluate the skews between signals across all controllers and factor this into the rules above for sharing clocks.

When static clocks are to be shared select which controller is to be the master. The slave controllers are made to share the master's static clocks by using the **Force Merging of PLL Clock Fanouts** assignments. You need to manually add the two PLL merging assignments using the Assignment Editor or directly update the **.qsf** file.

The **Force Merging of PLL Clock Fanouts** option requires both the master and slave PLL to have the same input reference clock in the RTL. This option limits the placement of the controllers to be only on the same side where one input clock can provide the reference clock for both center PLLs. If your multiple memory controllers' PLLs cannot share the same input reference clock pin, you need to manually connect the clocks together via RTL, instead of using this assignment.

If you are not sharing the address and command clocks, generate the slave controller address and command clocks from the master controller's PLL spare outputs. Create these in the PLL wizard. Modify the PHY source code to bring these clocks out to the top-level file and then modify the slave controllers to connect in the new address and command clock.

Table 7–1 shows a list of PLL merging assignments.

Table 7–1. PLL Merging Assignments *(Note 1)* (Part 1 of 2)

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

Table 7–1. PLL Merging Assignments *(Note 1)* (Part 2 of 2)

Device	Rate	Assignments

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

Notes to Table 7–1:

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

To check that the Quartus II software has applied the assignments, read the Compilation Report - PLL Usage (available only if the fitter step is successful), which shows the two clocks merged. This report allows you to compare the PLL usage before PLL merging and after PLL merging.

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

Adding Constraints to the Design

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

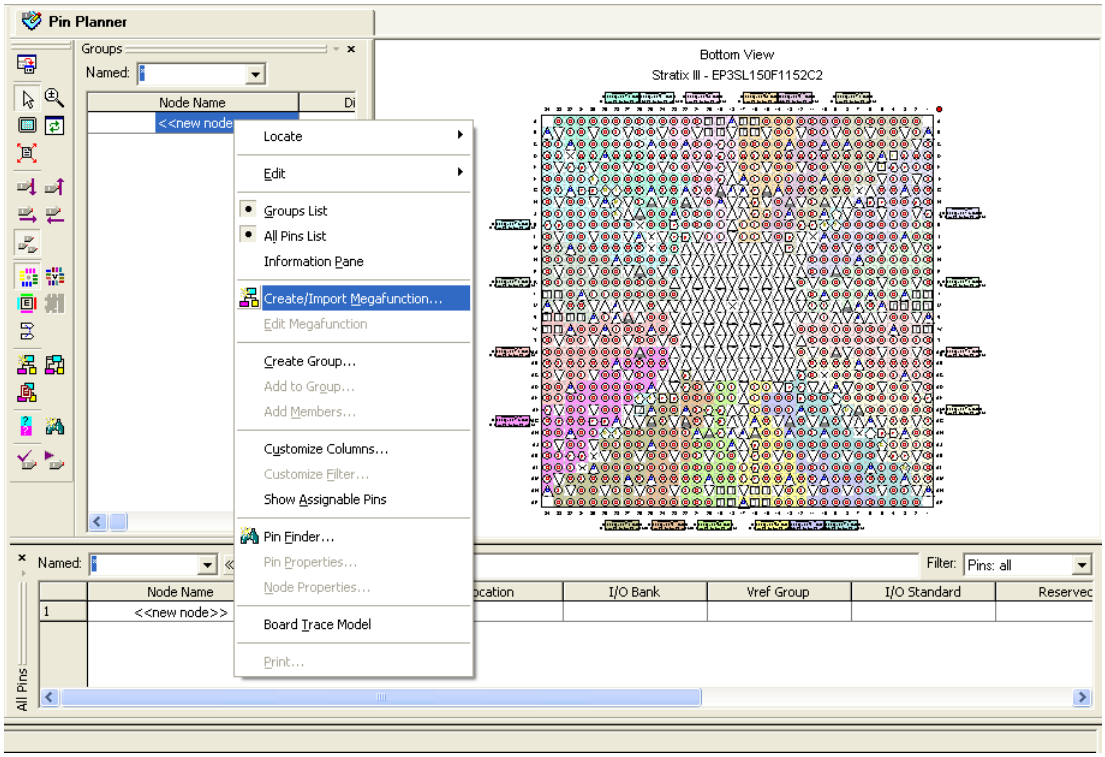
You must then add the **.sdc** file for each controller. If you are using two or more different memory controller variations, then you need to add the .sdc files generated for each variation. If, however, your design consists of multiple instantiations of the same controller, you only need to add the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

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

1. Open the Assignment menu and click on Pin Planner.

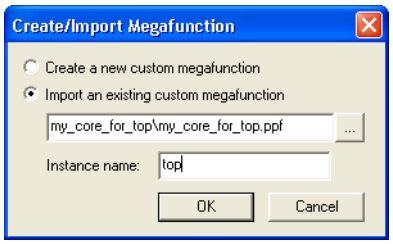
2. Right-click in any area under Node Name, and select Create/Import Megafunction (Figure 7–3).

Figure 7–3. Create/Import Megafunction Option in the Pin Planner



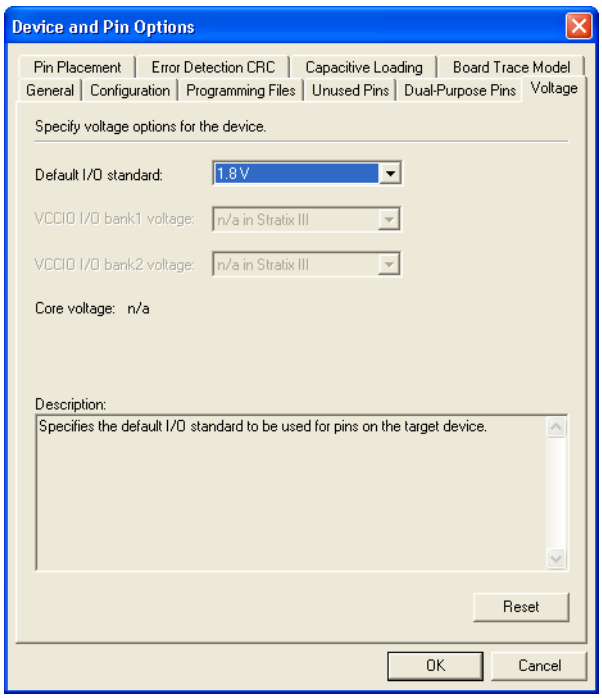
3. Turn on the Import an existing custom megafunction radio button and choose the *<variation_name>***.ppf** file.
4. Type the prefix that you want to use under the Instance name (Figure 7–4).

Figure 7–4. Adding Prefix to the Memory Interface Pin Names



You may also want to set the default I/O standard for your design in the **Voltage** section of the **Device and Pin** Options, by navigating to **Assignment and clicking Setting** (Figure 7–5).

Figure 7–5. Setting a Default Voltage for Your Design



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

Compiling the Design to Generate a Timing Report

During design compilation, the Quartus II software analyzes the timing margins for the memory controllers across the three timing model corners. As an alternative, you can also open the Tools menu and click **TimeQuest Timing Analyzer**. Double click on **Report DDR** to get the timing report for all ALTMEMPHY-based memory controllers in the design for the timing model that you select in the TimeQuest timing analyzer.

For more information about analyzing memory interface timing in Stratix III and Cyclone III devices, refer to the *Timing Analysis* section in volume 4 of the *External Memory Interface Handbook*.

Timing Closure

You may encounter some of the following timing failures:

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

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

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

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

This chapter provides additional information about the document and Altera.

Document Revision History

The following table shows the revision history for this document.

Date	Version	Changes
December 2010	2.1	Updated for 10.1 release.
July 2010	2.0	Updated for 10.0 release.
February 2010	1.1	Updated for 9.1 SP1 release.
November 2009	1.0	First published.

How to Contact Altera

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

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

Note to Table:

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

Typographic Conventions

The following table shows the typographic conventions this document uses.

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

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