External Memory Interface Handbook Volume 5

Section II. UniPHY Design Tutorials

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

EMI_TUT_QDR-3.0

Document last updated for Altera Complete Design Suite version: 11.0
Document publication date: June 2011

Subscribe

Contents

1. Using Controllers with UniPHY in Stratix III and Stratix IV Devices

This tutorial describes how to use the design flow to implement external memory interfaces with UniPHY using Altera devices. This tutorial also provides some recommended settings to simplify the design.

The design examples use the Stratix III and Stratix IV FPGA development kits. Table 1–1 lists the design files provided by Altera for the different memory protocols and devices.

Table 1–1. Design Files

Design File	Memory Protocol	Device	Width	Frequency
emi_uniphy_ddr2_siii.zip	DDR2 SDRAM with UniPHY	EP3SL150F1152-C2	72 bits	300 MHz
emi_uniphy_ddr3_siv.zip	DDR3 SDRAM with UniPHY	EP4SGX230KF40C2	64 bits	533 MHz
emi_qdrii_plus_siii.zip	QDR II or QDR II+ SRAM with UniPHY	EP3SL150F1152C2	18 bits	400 MHz
emi_qdrii_plus_siv.zip	QDR II or QDR II+ SRAM with UniPHY	EP4SGX230KF40C2	18 bits	400 MHz
emi_rldramii_siii.zip	RLDRAM II with UniPHY	EP3SL150F1152C2	36 bits	400 MHz
emi_rldramii_siv.zip	RLDRAM II with UniPHY	EP4SE530H35C2	36 bits	533 MHz

You can also use these design examples if you are targeting an Arria® II, HardCopy® III, HardCopy IV or Stratix V device for your design.

To download the design examples, go to the External Memory Interface Design Examples page or refer to the List of Designs Using Altera External Memory IP 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 assumes that you have experience with the Quartus® II software. This tutorial requires the following software:

- Quartus II software version 11.0 or later.
- ModelSim®-Altera® version 6.6d or later.

Creating a Quartus II Project

Create a project in the Quartus II software that targets the respective device.

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

Instantiating and Parameterizing a Controller

After creating a Quartus II project, instantiate the controller and its parameters.

Instantiating a Controller

To instantiate the controller, follow these steps:

1. On the Tools menu, click **MegaWizard™ Plug-In Manager**.
2. In the MegaWizard Plug-In Manager, in the **Interfaces** folder, expand **External Memory** and select the appropriate controller.
3. Type *<variation name>* for the name of your controller.
4. Click **Next**.

Parameterizing the Controller

To parameterize the controller, apply the values shown in Table 1–2, Table 1–3, or Table 1–4.

☞ Retain the default values for the parameters not listed in these tables.

Table 1–2 shows the parameter values for DDR2 and DDR3 SDRAM controllers with UniPHY targeting Stratix III and Stratix IV devices.

Table 1–2. Parameter Values for DDR2 SDRAM and DDR3 SDRAM with UniPHY

Parameter	DDR2 SDRAM	DDR3 SDRAM
Presets	MICRON MT9HTF12872 AY-800	MT41J64M16L A-15E
PHY Settings		
Speed Grade	2	2
Memory clock frequency	300 MHz	533 MHz
PLL reference clock frequency	50 MHz	50 MHz
Full or half rate on Avalon-MM interface	Half	Half
Advanced clock phase control	Turn on	Turn on
Additional address and command clock phase	–20	0
I/O standard	—	—
Memory Parameters		
Memory format	Unbuffered DIMM	Discrete Device
Memory device speed grade	400 MHz	666.66 MHz
Total interface width	72	64
DQ/DQS group size	8	8
Number of DQS groups	1	1
Number of chip selects	1	1
Number of clocks per chip select	3	1

Table 1–2. Parameter Values for DDR2 SDRAM and DDR3 SDRAM with UniPHY

Parameter	DDR2 SDRAM	DDR3 SDRAM
Row address width	14	13
Column address width	10	10
Bank address width	3	3
Memory CAS latency setting	6	8
Output drive strength setting	Full	RZQ/7
Memory on-die termination (ODT) setting	75	—
ODT Rtt nominal value	—	RZQ/4
Memory write CAS latency setting	—	6
Dynamic ODT (Rtt_WR) value	—	RZQ/4
Board Settings		
CK/CK# slew rate (Differential)	2 V/ns	4 V/ns
Address and command slew rate	1 V/ns	1.5 V/ns
DQS/DQS# slew rate (Differential)	2 V/ns	3 V/ns
DQ slew rate	1 V/ns	1.5 V/ns
Maximum CK delay to DIMM/device	0.6 ns	0.618 ns
Maximum DQS delay to DIMM/device	0.6 ns	0.368 ns
Minimum delay difference between CK and DQS	0.098 ns	0.250
Maximum delay difference between CK and DQS	0.151 ns	0.378
Maximum skew within DQS group	0 ns	0.017 ns
Maximum skew between DQS groups	0.053 ns	0.128 ns
Average delay difference between DQ and DQS	0 ns	0.021 ns
Maximum skew within address and command bus	0.155 ns	0.072 ns
Average delay difference between address and command and CK	0.490 ns	0.015 ns
Controller Settings		
Maximum Avalon-MM burst length (Specify the burst length based on your system)	64	64
Diagnostics		
Auto calibration mode	Skip calibration	Skip calibration
Skip memory initialization	Turn on	Turn on

Table 1–3 shows the parameter values for QDR II and QDR II+ SDRAM controllers with UniPHY targeting Stratix III and Stratix IV devices.

Table 1–3. Parameter Values for QDR II and QDR II+ SRAM

Parameter	QDR II / QDR II+ SRAM	
	Stratix III	Stratix IV
Presets	CY7C1263V18-400	CY7C1563KV18-400
PHY Settings		
Speed Grade	2	2
Memory clock frequency	400 MHz	400 MHz
PLL reference clock frequency	125 MHz	50 MHz
Full or half rate on Avalon-MM interface	Half	Half
Advanced clock phase control	0	0
Additional Address/Command clock phase	0	0
I/O standard	1.8-V HSTL	1.5-V HSTL
Sequencer optimization	Performance (Nios II-based Sequencer)	Performance (Nios II-based Sequencer)
Memory Parameters		
Address width	19	20
Data width	18	18
Data-mask width	2	2
CQ width	1	1
K width	1	1
Burst length	4	4
Board Settings		
Maximum delay difference between devices	0 ps	0 ps
Maximum skew within write data group (i.e. K group)	20 ps	10 ps
Maximum skew within read data group (i.e. CQ group)	20 ps	3 ps
Maximum skew between CQ groups	20 ps	0 ps
Maximum skew within Address/Command bus	81 ps	27 ps
Average delay difference between Address/Command and K	1 ps	5 ps
Average delay difference between write data signals and K	3 ps	17 ps
Average delay difference between read data signals and CQ	9 ps	3 ps
Controller Settings		
Maximum Avalon-MM burst length (Specify the burst length based on your system)	64	64
Reduce Controller Latency by	Zero	One
Diagnostics		

Table 1–3. Parameter Values for QDR II and QDR II+ SRAM

Parameter	QDR II / QDR II+ SRAM	
	Stratix III	**Stratix IV**
Auto calibration mode	Skip calibration	Skip calibration
Skip memory initialization	Turn on	Turn on

Table 1–4 shows the parameter values for RLDRAM II controller with UniPHY targeting Stratix III and Stratix IV devices.

Table 1–4. Parameter Values for RLDRAM II

Parameter	RLDRAM II	
	Stratix III	**Stratix IV**
Presets	MT49H16M36-18	MT49H16M36 HT-18
PHY Settings		
Speed Grade	2	2
Memory clock frequency	400 MHz	533 MHz
PLL reference clock frequency	100 MHz	50 MHz
Full or half rate on Avalon-MM interface	Half	half
Additional Address/Command clock phase	0	0
I/O standard	1.8-V HSTL	1.8-V HSTL
Sequencer optimization	Area (RTL Sequencer)	Performance (Nios II-based Sequencer)
Memory Parameters		
Address width	19	19
Data width	36	36
Bank-address width	3	3
Data-mask width	1	1
QK width	2	2
DK width	2	2
Burst length	4	4
Memory Mode Register Configuration	t_{RC} = 6	t_{RC} = 8
	t_{RL} = 6	t_{RL} = 8
	t_{WL} = 7	t_{WL} = 9
	f = 400 MHz –175 MHz	f = 533 MHz –175 MHz
Board Settings		
tAS Vref to CK/CK# Crossing	25 ps	0 ps
tAS VIH MIN to CK/CK# Crossing	–100 ps	–100 ps
tAH CK/CK# Crossing to Vref	13 ps	0 ps
tAH CK/CK# Crossing to VIH MIN	–50 ps	–50 ps

Table 1–4. Parameter Values for RLDRAM II

Parameter	RLDRAM II	
	Stratix III	Stratix IV
tDS Vref to CK/CK# Crossing	11 ps	5 ps
tDS VIH MIN to CK/CK# Crossing	–100 ps	–100 ps
tDH CK/CK# Crossing to Vref	6 ps	3 ps
tDH CK/CK# Crossing to VIH MIN	–50 ps	–50 ps
Maximum CK delay to device	600 ps	309 ps
Maximum DK delay to device	600 ps	288 ps
Minimum delay difference between CK and DK	–55	21 ps
Maximum delay difference between CK and DK	0 ps	23 ps
Maximum delay difference between devices	0 ps	0 ps
Maximum skew within QK group	45 ps	70 ps
Maximum skew between QK groups	32 ps	7 ps
Maximum skew within Address/Command bus	93 ps	13 ps
Average delay difference between Address/Command and CK	87 ps	12 ps
Average delay difference between write data signals and DK	61 ps	12 ps
Average delay difference between read data signals and QK	–32 ps	0 ps
Controller Settings		
Maximum Avalon-MM burst length (Specify the burst length based on your system)	128	4
Reduce Controller Latency by	1	0
Diagnostics		
Auto calibration mode	Skip calibration	Skip calibration
Skip memory initialization	Turn on	Turn on

☞ Altera recommends that you use a third party tool to perform whole path simulation for DQ, DQS, CK, address, and command signals to obtain the slew rate and board skew parameters for your design. If you are unable to perform post-layout simulation to obtain the slew rate parameters, you can use the board trace delay model. For more information, refer to "Performing Advanced I/O Timing Analysis with Board Trace Delay Model" on page 1–10 for more information about board trace delay models.

☞ If you want to migrate your design to a HardCopy device, turn on **HardCopy Compatibility Mode**, and select the appropriate **Reconfigurable PLL Location**.
The Intersymbol Interference (ISI) parameters are not applicable for single chip-select configurations. Set these parameters to **0 ns**.

Click **Finish** to generate your MegaCore function variation. The MegaWizard Plug-In Manager generates the following two example design projects:

- Project for synthesis flow; obtain this file from the <*variation_name*>**_example_design/example_project** directory.

- Project for simulation flow; obtain this file from the *<variation_name>*_**example_design/simulation** directory.

Each project includes full design RTL and the Quartus II project files (**.qpf** and **.qsf**) that you can use with minimal modifications.

When the controller variation completes generation, close the existing project.

Performing RTL Simulation (Optional)

This section describes RTL (functional) simulation.

After you have generated your design, the Quartus II software creates a complete design example for functional simulation in the *<variation_name>*_**example_design/simulation/** directory. This design example includes the driver, testbench, and the memory model that allows you to perform functional simulation on the design. This design example also includes a script file that directs the simulator to perform the necessary compilation and simulation. Use the ModelSim-Altera simulator to perform the functional simulation.

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

1. Open the generated example project for the design example simulation, ***<variation_name>*_example_design/simulation/<variation_name>_ example_sim.qpf**.
2. Change the device in the design example to the actual device you are targeting for your design.
3. To set the absolute path to your third-party simulator executable file, follow these steps:
   a. On the Tools menu, click **Options**.
   b. In the **Category** list, select EDA Tools Options and set the default path for **ModelSim-Altera** to **C:\<version>\modelsim_ae\win32aloem**.
   c. Click **OK**.
4. To elaborate your design, on the Processing menu, point to **Start** and click **Start Analysis & Elaboration**.
5. To compile all the neccessary files and run the simulation, on the Tools menu, point to **Run EDA Simulation Tool** and click **EDA RTL Simulation**.

Adding Constraints

When you instantiate a controller, the Quartus II software generates the constraints files for the design example. Apply these constraints to the design before compilation.

Using Example Project

To use the example project, open the example project located in *<variation_name>*_**example_design/example_project**. If the device you are using does not match the device in the example project, change the device in the project.

To change the device in the example project to the actual device you are targeting for your design, follow these steps:

1. On the Assignments menu, click **Device**.
2. Select the device of your choice.
3. Click **OK**.

Adding Pin and DQ Group Assignments

The pin assignment script, <*variation_name*>**_if0_p0_pin_assignments.tcl**, sets up the I/O standards for the external memory controllers with UniPHY interface. This script does not include the assignment for the design's PLL input clock. You must create a PLL input clock for the design and provide pin assignments for the signals of both the example driver and testbench that the MegaCore variation generates.

Run the <*variation_name*>**_pin_assignments.tcl** script to add the pin, I/O standards, and DQ group assignments to the design example. To run the pin assignment scripts, follow these steps:

1. On the Processing menu, point to **Start** and click **Start Analysis & Synthesis**.
2. In the Tools menu, click **Tcl Scripts**.
3. Locate the <*variation_name*>**_pin_assignments.tcl** file.
4. Click **Run**.

☞ The PLL input clock I/O does not need to have the same I/O standard as the memory interface I/Os. However, you may see a no-fit error as the bank in which the PLL input clock I/O gets placed becomes unusable for placement of the memory interface I/Os because of the incompatible V_{CCIO} levels. Altera recommends that you assign the same I/O standard to the PLL input clock I/O.

Assigning Pins

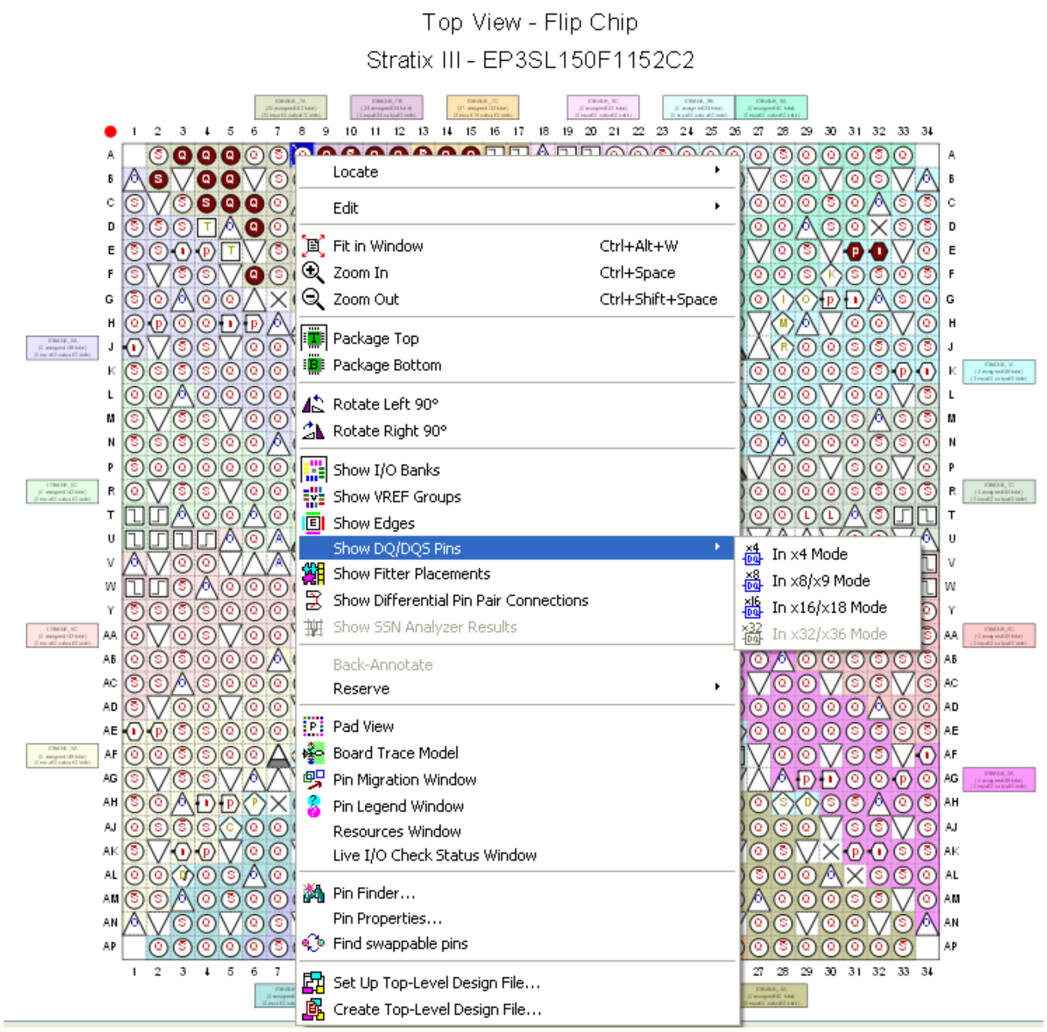
To enter the pin location assignments, assign all of your pins so that the Quartus II software fits your design correctly and gives correct timing analysis. To assign the pin locations, run the Altera-provided <*board_name*>**_pin_location.tcl** file.

Alternatively, to manually assign the pin locations in the Pin Planner, follow these steps:

1. On the Assignments menu, click **Pin Planner**.
2. To view the DQS groups in the Pin Planner, right click, select **Show DQ/DQS Pins**, and click **In ×16/×18 Mode**. The Pin Planner shows each DQS group in a different color and with a different legend: S = `DQS` pin, Sbar = `DQSn` pin and Q = `DQ` pin.

Figure 1–1 shows an example of the Quartus II Pin Planner.

Figure 1–1. Quartus II Pin Planner, Show DQ/DQS Pins, In ×16/×18 Mode



3. To identify the differential I/O pairs in the Pin Planner, right-click and select **Show Differential Pin Pair Connections**. The pin pairs show a red line between each pin pair.

For more information about pin location assignments, refer to the *Device and Pin Planning* section in volume 2 of the *External Memory Interface Handbook*.

Performing Advanced I/O Timing Analysis with Board Trace Delay Model

You should use this method only if you are unable to perform post-layout simulation on the memory interface signals to obtain the slew rate parameters.

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

To perform advanced I/O timing analysis with the Quartus II software, follow these steps:

1. After the instantiation is complete, analyze and synthesize your design.
2. Add pin and DQ group assignment by running the <*variation_name*>**_if0_p0_pin_assignments.tcl** script.
3. Enter the pin location assignments.
4. Assign the virtual pins, if necessary.
5. Enter the board trace model information.

To enter board trace model information, follow these steps:

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

   Alternatively, you can run the Altera-provided <*variation_name*>**_BTModels.tcl** file to apply the board trace model information.

3. Compile your design. To compile the design, on the Processing menu, click **Start Compilation**.
4. After successfully compiling the design, perform timing analysis in the TimeQuest timing analyzer.

To perform timing analysis, follow these steps:

1. In the Quartus II software, on the Tools menu, click **TimeQuest Timing Analyzer**.
2. On the **Tasks** pane, click **Report DDR**.
3. On the **Report** pane, select **Advanced I/O Timing>Signal Integrity Metrics**.
4. In the **Signal Integrity Metrics** window, right-click and select **Regenerate** to regenerate the signal integrity metrics.
5. In the **Signal Integrity Metrics** window, note the 10–90% rise time (or fall time if fall time is worse) at the far end for CK/CK#, address, and command, DQS/DQS#, and DQ signals.
6. In the Quartus II software, launch the parameter editor of the targeted controller with UniPHY.
7. In the **Board Settings** tab, for the slew rates, type the values you obtained from the signal integrity metrics.
8. For the **Board Skews** parameters, set the maximum skew between data groups of your design. Set the other board parameters to 0 ns.
9. Compile your design.

Table 1–5 through Table 1–8 show the board trace model parameters for the Stratix III and Stratix IV GX devices for the different memory protocols.

Table 1–5. DDR2 SDRAM Board Trace Model Summary for Stratix III Devices

Net	Near (FPGA End of Line)						Far (Memory End of Line)				
	Length (Inch)	C_per_ length (pF/In)	L_per_ length (H/In)	Cn (pF)	Rns (W)	Rnh (W)	Length (Inch)	C_per_ length (pF/In)	L_per_ length (nL/In)	Cf (pF)	Rfh/Rfp (W)
Addr *(1)*	2.717	3.3	7.8	33	5.1	56	4.968	2.8	9	14	—
CLK	3.069	2.8	9.2	3.5	—	—	1.349	2.8	9	4.5	67
CKE	2.717	2.8	7.8	33	—	56	4.968	2.8	9	42	—
CS#	2.717	2.8	7.8	33	—	56	3.936	2.8	9	42	—
ODT	2.717	2.8	7.8	33	—	56	4.968	2.8	9	39.75	—
DQS0	3.017	3.4	7.8	—	22	—	0.750	2.8	9	3.5	50
DQS1	3.005	3.4	8.1	—	22	—	0.760	2.8	9	3.5	50
DQS2	2.851	3.4	7.8	—	22	—	0.760	2.8	9	3.5	50
DQS3	2.653	3.4	8.1	—	22	—	0.760	2.8	9	3.5	50
DQS4	2.686	3.4	7.8	—	22	—	0.760	2.8	9	3.5	50
DQS5	2.701	3.4	7.8	—	22	—	0.760	2.8	9	3.5	50
DQS6	2.75	3.4	7.8	—	22	—	0.760	2.8	9	3.5	50
DQS7	2.923	3.4	8.1	—	22	—	0.750	2.8	9	3.5	50
DQS8	2.536	3.4	7.8	—	22	—	0.940	2.8	9	3.5	50

Note to Table 1–5:

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

Table 1–6. DDR3 SDRAM Board Trace Model Summary for Stratix IV GX Devices

Net	Near (FPGA End of Line)						Far (Memory End of Line)				
	Length (Inch)	C_per_ length (pF/In)	L_per_ length (H/In)	Cn (pF)	Rns (W)	Rnh (W)	Length (Inch)	C_per_ length (pF/In)	L_per_ length (nL/In)	Cf (pF)	Rfh/Rfp (W)
Addr *(1)*	—	—	—	—	—	—	4.177	4.25	7.61	5.2	56
CLK	—	—	—	—	—	—	3.865	4.11	7.84	5.6	100
CKE	—	—	—	—	—	—	4.246	4.25	7.61	5.2	56
DQ/DM	—	—	—	—	—	—	2.317	4.25	7.61	2.5	60
DQS	—	—	—	—	—	—	2.299	4.11	7.84	2.5	60

Note to Table 1–6:

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

Table 1–7. QDR II+ SRAM Board Trace Model Summary for Stratix III and Stratix IV GX Devices

Net	Near (FPGA End of Line)						Far (Memory End of Line)				
	Length (in)	C_per_ length (pF/in)	L_per_ length (nH/in)	Cn (pF)	Rns (W)	Rnh (W)	Length (in)	C_per_ length (pF/in)	L_per_ length (nH/in)	Cf (pF)	Rfh (W)
Stratix III											
`Add_Cntl`	2.947	3.36	7.77	—	—	—	—	—	—	5	56
`bws_n`	2.572	3.35	7.90	—	—	—	—	—	—	5	56
`K`	2.722	3.38	8.13	—	—	—	—	—	—	4	56
D Group	3.035	3.36	7.77	—	—	—	—	—	—	5	56
Stratix IV GX											
`Add_Cntl`	2.424	4.25	7.61	—	—	—	—	—	—	5	—
`bws_n`	2.309	4.25	7.61	—	—	—	—	—	—	5	—
`K`	2.282	4.11	7.84	—	—	—	—	—	—	6	—
`doffn`	1.578	4.25	7.61	—	—	10 k	—	—	—	5	—
D Group	2.406	4.25	7.61	—	—	—	—	—	—	5	—

Table 1–8. RLDRAM II Board Trace Model Summary for Stratix III and Stratix IV GX Devices

Net	Near (FPGA End of Line)						Far (Memory End of Line)				
	Length (in)	C_per_ length (pF/in)	L_per_ length (H/In)	Cn (pF)	Rns (W)	Rnh (W)	Length (in)	C_per_ length (pF/in)	L_per_ length (nF/in)	Cf (pF)	Rfh/Rfp (W)
Stratix III											
`Addr` *(1)*	3.123	3.738	8.257	—	—	—	—	—	—	4.0	56
`CLK`	1.907	3.33	9.266	—	—	—	—	—	—	2.5	—
`DQ/DM`	1.801	3.738	8.257	—	—	—	—	—	—	4.25	—
`DK0`	1.723	3.33	9.266	—	—	—	—	—	—	2.5	56
`DK1`	1.795	3.33	9.266	—	—	—	—	—	—	2.5	56
Stratix IV											
`Addr` *(1)*	2.231	3.2	8.1	—	—	—	—	4.5	—	2.0	56
`CLK`	2.107	2.9	9.1	—	—	—	—	4.5	—	2.5	56
`DQ/DM`	1.868	2.0	8.1	—	—	—	—	4.5	—	4.3	125
`DK0`	1.895	2.9	9.1	—	—	—	—	2.3	—	2.5	56
`DK1`	1.897	2.9	9.1	—	—	—	—	2.3	—	2.5	56

Note to Table 1–8:

(1) `Addr` = `Addr`, `ba`, `cs_n`, `ref_n`, and `we_n`.

Compiling Design and Verifying Timing

To compile the design, on the Processing menu, click **Start Compilation**. After successfully compiling the design, the Quartus II software automatically runs the verified timing script of the master DDR3 SDRAM memory, <*variation_name*>**_if0_p0_report_timing.tcl**, which produces a timing margin report for the design together with the compilation report.

The report timing script performs the following tasks:

1. Creates a timing netlist.
2. Reads the <*variation_name*>**.sdc** file.
3. Updates the timing netlist.

You can also obtain the timing report by running the report timing script, <*variation_name*>**_if0_p0_report_timing.tcl**, in the TimeQuest timing analyzer. To obtain the timing report in the TimeQuest Timing Analyzer window, follow these steps:

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

The results are the same as the Quartus II software results.

For more information about the TimeQuest timing analyzer, refer to *The Quartus II TimeQuest Timing Analyzer* chapter in volume 3 of the *Quartus II Handbook*.
For information timing analysis, refer to the *Timing Analysis* section in volume 4 of the *External Memory Interface Handbook*.

Verifying FPGA Functionality

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

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

To add the SignalTap II Embedded Logic Analyzer to your Quartus II project, perform the following steps:

1. On the Tools menu, click **SignalTap II Logic Analyzer**.
2. Under **Signal Configuration** next to the **Clock** box, browse to locate sub-entities.
3. Turn on **Include subentities** to include all sub-entities in the hierarchy.
4. Type `<variation name>` in the **Named** box, for **Filter** select **SignalTap II: pre-synthesis** and click **List**.

5. Select *<variation name>*:**mem_if|afi_clk** in **Nodes Found** and click **>** to add the signal to **Selected Nodes**.
6. Click **OK**.
7. Under **Signal Configuration**, specify the following settings:
   - For **Sample depth**, select **512**.
   - For **RAM type**, select **Auto**.
   - For **Trigger flow control**, select **Sequential**.
   - For **Trigger position**, select **Center trigger position**.
   - For **Trigger conditions**, select **1**.
8. On the Edit menu, click **Add Nodes**.
9. In the **Named** box, search for specific nodes by typing `*`, for **Filter** select **SignalTap II: pre-synthesis** and click **List**.
10. In **Nodes Found**, select the appropriate nodes and click > to add to **Selected Nodes**. The following signals are common to all memory protocols:
    - `driver_status_fail`
    - `driver_status_pass`
    - `driver_status_test_complete`
    - `local_cal_fail`
    - `local_cal_success`
    - `local_init_done`
    - `global_reset_n (optional)`
    - `pnf_per_bit`
    - `pnf_per_bit_persist`
    - `rdata_valid_reg`
    - `rdata_reg`
    - `written_data_fifo|data_out`
    - `avl_addr`
    - `avl_rdata`
    - `avl_rdata_valid`
    - `avl_wdata`
    - `avl_write_req`
    - `avl_read_req`
    - `avl_ready`

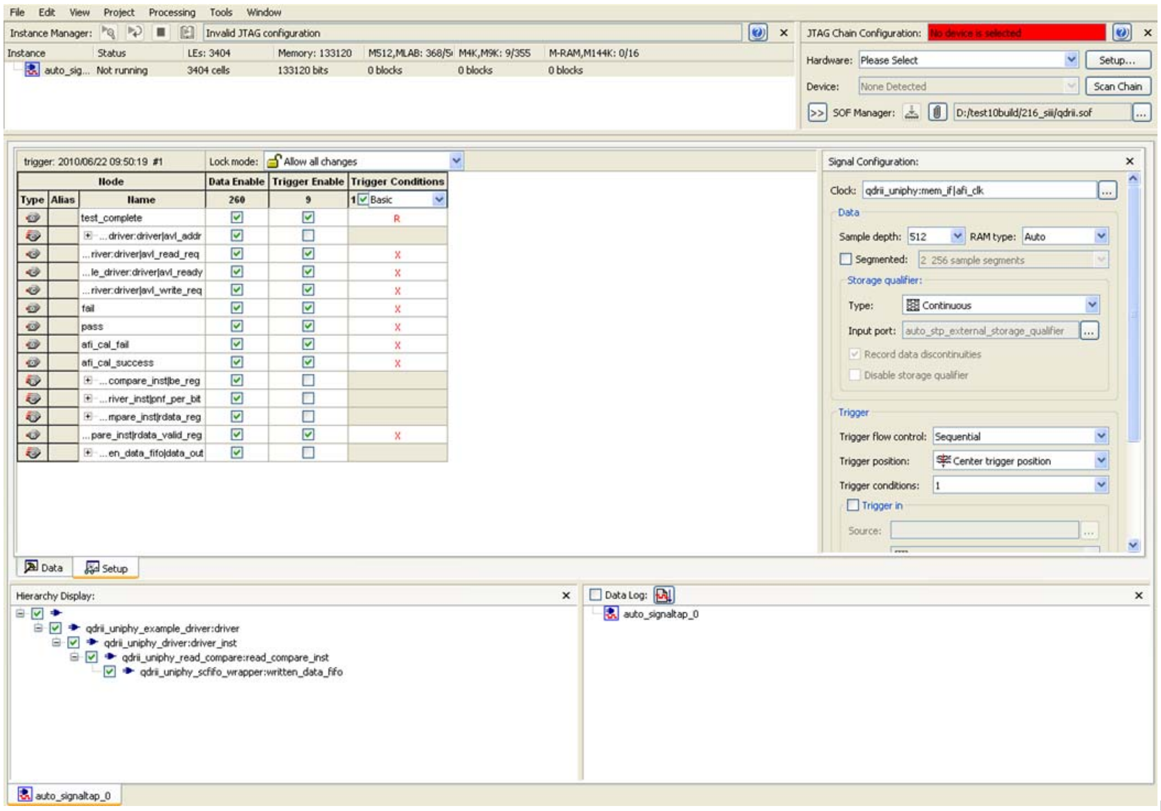
    Do not add any interface signals to the SignalTap II Embedded Logic Analyzer. The load on these signals increases and adversely affects the timing analysis.

11. Click **OK**.
12. To reduce the SignalTap II logic size, turn off **Trigger Enable** on the appropriate bus signals:
13. Right-click **Trigger Conditions** for the `driver_status_test_complete` signal and select **Rising Edge**.
14. On the File menu, click **Save** to save the SignalTap **.stp** file to your project.

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

Figure 1–2 shows the an example of a completed SignalTap II Embedded Logic Analyzer after you perform all the steps.

Figure 1–2. SignalTap II Embedded Logic Analyzer



Instance Manager: Invalid JTAG configuration

Instance	Status	LEs: 3404	Memory: 133120	M512,MLAB: 368/5	M4K,M9K: 9/355	M-RAM,M144K: 0/16
auto_sig...	Not running	3404 cells	133120 bits	0 blocks	0 blocks	0 blocks

JTAG Chain Configuration: No device is selected
Hardware: Please Select
Device: None Detected
SOF Manager: D:/test10build/216_siii/qdrii.sof

trigger: 2010/06/22 09:50:19 #1
Lock mode: Allow all changes

Node			Data Enable	Trigger Enable	Trigger Conditions
Type	Alias	Name	260	9	1 ✓ Basic
		test_complete	✓	✓	R
		...driver:driver|avl_addr	✓	☐	
		...river:driver|avl_read_req	✓	✓	X
		...le_driver:driver|avl_ready	✓	✓	X
		...river:driver|avl_write_req	✓	✓	X
		fail	✓	✓	X
		pass	✓	✓	X
		afi_cal_fail	✓	✓	X
		afi_cal_success	✓	✓	X
		...compare_inst|be_reg	✓	☐	
		...river_inst|pnf_per_bit	✓	☐	
		...mpare_inst|rdata_reg	✓	☐	
		...pare_inst|rdata_valid_reg	✓	✓	X
		...en_data_fifo|data_out	✓	☐	

Signal Configuration:
Clock: qdrii_uniphy:mem_if|afi_clk
Data
Sample depth: 512 RAM type: Auto
☐ Segmented: 2 256 sample segments
Storage qualifier:
Type: Continuous
Input port: auto_stp_external_storage_qualifier
✓ Record data discontinuities
☐ Disable storage qualifier
Trigger
Trigger flow control: Sequential
Trigger position: Center trigger position
Trigger conditions: 1
☐ Trigger in
Source:

Hierarchy Display:
- qdrii_uniphy_example_driver:driver
  - qdrii_uniphy_driver:driver_inst
    - qdrii_uniphy_read_compare:read_compare_inst
      - qdrii_uniphy_scfifo_wrapper:written_data_fifo

Data Log: auto_signaltap_0

Compiling the Project

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

Verifying Timing

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

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

Downloading the Object File

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

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

Testing the Design Example in Hardware

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

2. Implementing Multiple Memory Interfaces Using UniPHY

This tutorial describes how to use the design flow to implement a design with multiple memory interfaces, using two 16-bit wide, 1-GB Micron MT41J64M16LA-15E 667-MHz DDR3 SDRAM controllers with UniPHY interface. This tutorial also provides some recommended settings to simplify the design.

In this tutorial, you implement two DDR3 SDRAM controllers, operating at the same frequency of 553 MHz and using the UniPHY IP, in a single Stratix V device. This implementation requires device resource sharing, such as delay-locked loops (DLLs), phase-locked loops (PLLs), and on-chip termination (OCT) control blocks, and may require extra steps to create the interfaces in a Quartus II project.

The design example in this tutorial uses a Stratix V device with two DDR3 SDRAM memory components with timing closure and RTL simulation verification on the master memory.

To download the design example, **emi_multiple_ddr3_sv.zip**, go to the External Memory Interface Design Examples page. You can also use these design examples if you are targeting a HardCopy device for your design.

For more information about the design flow, refer to the *Recommended Design Flow* section in volume 1 of the *External Memory Interface Handbook*. For more information about the DDR3 SDRAM controller with UniPHY, refer to the *DDR2 and DDR3 Controller with UniPHY* section in volume 3 of the *External Memory Interface Handbook*.

Before Creating a Design in the Quartus II Software

When creating multiple memory interfaces to be fitted in a single device, first ensure that there are enough device resources for the different interfaces. These device resources include the number of pins, DLLs, PLLs, OCT control blocks, and clock networks, where applicable. You can share the DLLs, PLLs, OCT control blocks, and clock network resources between multiple memory interfaces, but there are some sharing limitations.

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

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

- Multiple controllers with no sharing. Check the number of PLLs and DLLs that are available in the targeted device, and the number of PLLs and DLLs needed in your design. If you have enough resources for the controllers, you do not need to share the DLL and PLL static clocks.

- Multiple controllers sharing PLLs and DLLs. The UniPHY supports the master-and-slave approach, with a PLL and DLL instantiated in a master controller and the PLL clocks and DLL shared with other identical UniPHY slave controllers. You require extra steps to explicitly merge the PLLs and DLLs in the top-level design file. The PLL and DLL signals from the master controller must be connected to their equivalent signals in the slave controllers.

☞ To share the PLLs and DLLs, ensure that the controllers are on the same or adjacent side of the device and the PLL and DLL are running at the same memory-clock frequency.

If you use FPGA OCT calibrated series, parallel, or dynamic termination for any I/Os in an external memory interface design, you require a calibration block for the OCT circuitry. This calibration block requires a pair of R_{UP} and R_{DN} pins, or an R_{ZQ} pin that you must place within any bank with the same I/O voltage as the memory interface signals. Route these pins to the design top-level and connect to the R_{UP} and R_{DN}, or the R_{ZQ} resistors on the board.

☞ You must prioritize the placement of the R_{UP} and R_{DN} pin pairs, or the R_{ZQ} pin in your design, because these pins may use up to two DQS or DQ pins from a group.

After understanding the device resource limitations, determine the resources to share for your multiple controller design. You can use the resources that you determined as a framework for your Quartus II design constraints.

Creating a PHY and Controller in a Quartus II Project

If you are creating multiple instances of the same controller with the same parameters (frequency of operation, data width, burst mode, etc.) and without sharing resources, you only need to generate the controller once. You can then instantiate this controller variation multiple times in your top-level design file. If you are using the controllers with different parameters, or different controllers, then you need to generate each variation of the memory controller individually. Extra steps are required to modify the RTL generated by the MegaWizard Plug-In Manager for resource sharing.

The high-performance memory controller with UniPHY is generated with a top-level design file called <*variation_name*>**_example.v** or **.vhd**, where *variation_name* is the name of the memory controller that you typed in the first page of the MegaWizard Plug-In Manager. This example top-level file instantiates the memory controller (<*variation_name*>**.v** or **.vhd**) and an example driver, which performs a comparison test after reading back data that was written to the memory.

When creating a multiple memory interface design, you have the option to combine certain aspects of the design flow illustrated in the following examples:

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

In this tutorial, you create a new top-level design schematic with an example driver instantiated in both the master and slave DDR3 SDRAM controllers.

Sharing PLLs and DLLs

The controllers with UniPHY memory interface require one PLL and one DLL to produce the clocks and delay codeword. Altera recommends that you use the master-and-slave approach to share the PLL and DLL when there are not enough resources. The PLL and DLL output signals are inputs or outputs in the top-level file, and an additional parameter, `PLL_DLL_MASTER` determines the direction of these output signals.

If you turn on **Master for PLL/DLL sharing** on the DDR2, DDR3 SDRAM, RLDRAM II, or QDR II and QDR II+ SRAM controllers with UniPHY parameter editor, the `PLL_DLL_MASTER` parameter is 1 and the RTL instantiates the PLL and DLL which drive the clock and DLL codeword inputs and outputs. If you turn off **Master for PLL/DLL sharing**, the `PLL_DLL_MASTER` parameter is 0, and the wires previously connected to the outputs of the PLL and DLL are now assigned to the clock and DLL codeword inputs and outputs. During synthesis, the direction is automatically resolved to either input or output.

By sharing the PLL output clocks, `pll_mem_clk`, `pll_write_clk`, and `pll_add_cmd_clk`, you can reduce the required number of clocks up to four clock networks.

The number of interfaces that you can share in this clock sharing scheme is limited by the number of PLLs and pins available in the device. UniPHY-based memory interfaces may share the output clocks between multiple controllers in the following two conditions:

- All UniPHY-based memory interfaces are either full-rate or half-rate designs.
- All UniPHY-based memory interfaces have the same PLL input and output frequencies.
- The `ref_clk` input of each PLL are connected to a clock signal derived from a signal clock source. You can use a clock fanout buffer to create separate PLL reference clocks from a single clock source.
- The memory controllers’ circuitry is located in the same two device quadrants.
- All the clocks are routed as required by their global network clock type.
- The general routing rules defined in the relevant memory standard layout section are met.

Sharing OCT Control Block

The controllers with UniPHY memory interface require an OCT control block to calibrate the I/O buffer on chip impedances off external resistors connected to the R_{UP} and R_{DN}, or R_{ZQ} pins on the FPGA. These resistance inputs connect directly to the OCT control block, which outputs 14- or 16-bit termination control buses.

If you turn on **Master for OCT control block** in the DDR2, DDR3 SDRAM, RLDRAM II, or QDR II and QDR II+ SRAM controllers with UniPHY parameter interface, the RTL instantiates the OCT control block in the UniPHY IP.

If you turn off **Master for OCT control block**, the UniPHY IP does not include the OCT control block and the RTL instantiates the block in the example top level. Then, you must explicitly connect the output termination control buses of the OCT control block to the PHY for proper operation.

System Requirements

This tutorial implements DDR3 SDRAM interface targeting the Stratix V device, with the master controller instantiating its own PLL to be shared with the slave controller. This tutorial requires the following software and hardware:

- Quartus II software version 11.0
- ModelSim-Altera version 6.5e or later
- Stratix V FPGA device

Creating a Quartus II Project

Create a project in the Quartus II software that targets a 5SGXMA7K2F40C2 device.

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

Instantiating and Parameterizing the Controllers

After creating a Quartus II project, instantiate and parameterize the DDR3 SDRAM controller with UniPHY.

Instantiating the Controllers

You have to instantiate the DDR3 SDRAM controller twice, one for the master controller and one for the slave controller. To instantiate the DDR3 SDRAM controller, follow these steps:

1. Start the MegaWizard Plug-In Manager.
2. In the MegaWizard Plug-In Manager, expand **External Memory** in the **Interfaces** folder and select **DDR3 SDRAM Controller with UniPHY v11.0**.
3. Type `core1` for the master and `core2` for the slave.

Parameterizing the Controllers

To parameterize the master or slave controller to interface with a 16-bit wide DDR3 SDRAM interface, perform the following steps:

1. In the **Presets** list, select **MT41J64M16LA-15E** and click **Apply**. The memory parameters are set automatically based on the memory preset settings you choose. You may change the parameters to suit your design requirements.

   CAUTION Selecting a new memory preset overwrites your existing settings. You need to enter the settings again.

2. In the **PHY Settings** tab, under **Clocks**, for **Memory clock frequency**, type **450** MHz as the system frequency.
3. For **PLL reference clock frequency**, type **100** MHz to match the on-board oscillator.
4. For **Full or half rate on Avalon-MM interface**, select **Half**. This option allows you to choose between the full-rate and half-rate controller, and define the bus data width between the controller and the PHY.

   For more information about selecting half-rate or full-rate controller, refer to the *Device and Pin Planning* section in volume 2 of the *External Memory Interface Handbook*.

5. Under **Advanced PHY Settings**, turn on **Advanced clock phase control**.
6. For **Additional address and command clock phase**, type **-22.50** for the master controller (core1), and **0** for the slave controller (core2).
7. For **Supply Voltage**, select **1.5-V DDR3**.
8. For **PLL sharing mode**, **DLL sharing mode**, and **OCT sharing mode**, select **Master** for the master controller (core1) and select **Slave** for the slave controller (core2).

   If you want to migrate your design to a HardCopy device, turn on **HardCopy Compatibility Mode**, and select the appropriate **Reconfigurable PLL Location**.

9. In the **Memory Parameters** tab, for **Total interface width**, type **16**.
10. Under **Memory Topology**, turn on **Fly-by topology**.
11. Under **Memory Initialization Options**, for **Memory CAS latency setting**, select **8**.
12. For **Output drive strength setting**, select **RZQ/7**.
13. For **ODT Rtt nominal value**, select **RZQ/4**.
14. For **Memory write CAS latency setting**, select **6**.
15. For **Dynamic ODT (Rtt_WR) value**, select **RZQ/4**.
16. In the **Board Settings** tab, for setup and hold derating, and board skews, use Altera's default settings.

    For more accurate timing analysis, assign the board trace delay model for every single pin. The board skew values are used to calculate the overall system timing margin. Change these values based on the performance of your board if you are not using the board trace delay model, or if you cannot accurately assign the board trace model for each pin. Refer to "Perform Advanced I/O Timing Analysis with Board Trace Delay Model" on page 2–11 for more information about board trace delay model.

17. In the **Controller Settings** tab, under **Avalon Interface**, for **Maximum Avalon-MM burst length**, specify the burst length based on the burst count sent from the core fabric. For example, if your system generates up to 64 beats, then select **64**.

18. In the **Diagnostics** tab, under **Simulation Options**, for **Auto-calibration mode**, select **Quick initialization and skip calibration**. This option allows you to skip memory calibration during simulation and decrease simulation time.
19. Click **Finish** to generate your MegaCore function variation. The MegaWizard Plug-In Manager generates all the files necessary for your DDR3 SDRAM controller with UniPHY, and an example top-level design, which you may use to test or verify the board operation.
20. Click **Exit** to close the MegaWizard Plug-In Manager after you have reviewed the generation report.

Adding Constraints

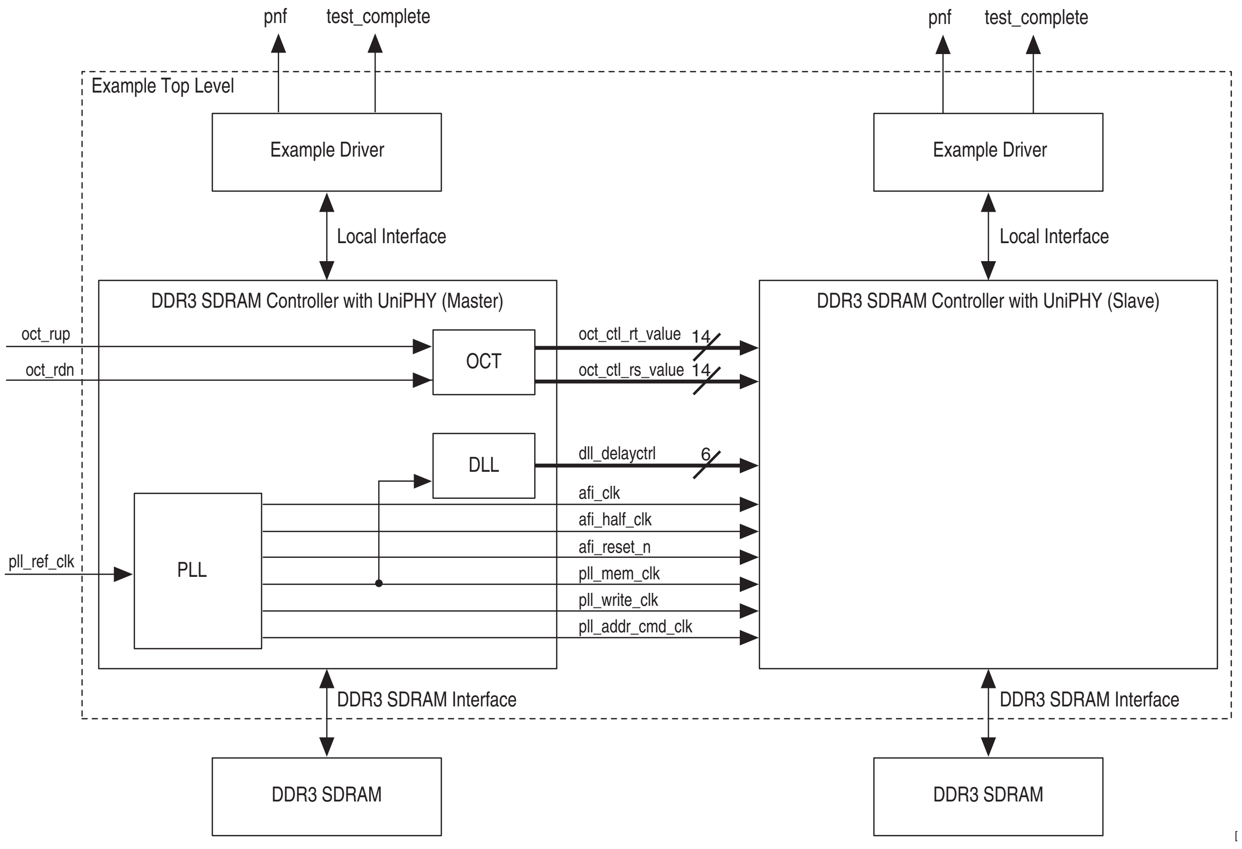
After instantiating the DDR3 SDRAM controller with UniPHY, the Quartus II software generates the constraints files for the design example. Apply these constraints to the design before compilation.

Creating Top-Level Project

To create a top-level design for multiple memory interfaces, use the top-level design with master controller instance and instantiate other slave memory interfaces, and add them to this top-level design. To demonstrate the multiple memory interfaces, instantiate an example driver with each memory controller. In this design example, you obtain the top-level design by instantiating the master DDR3 SDRAM example top-level design and adding the slave DDR3 SDRAM instance to the top-level design.

Figure 2–1 shows the top-level diagram for the multiple memory controllers.

Figure 2–1. Multiple Memory-Controller Top-Level Diagram



For detailed information about how you can create a top-level design for multiple memory interfaces, refer to the Altera-provided **sv_multiple_ddr3.v** file.

Alternatively, you can create your own new top-level design to instantiate all the memory controllers in the design.

Adding PLL Sharing Constraint

To enable the DDR3 SDRAM slave controller to share the DDR3 SDRAM master's PLL output clocks, follow these steps:

1. On the File menu, click **Open**.
2. Browse to **core2_example_design\example_project\core2_example\submodules\core2_example_if0_p0_timing.tcl.** and click **Open**.
3. Locate the following PLL clock assignments:

```
set master_corename "_MASTER_CORE_"
set master_instname "_MASTER_INST_"
```

and change the master instance and core names to match the names you specify in your design, for example:

```
set master_corename "core1_example_if0_p0"

set master_instname "if02|p0"
```

4. Click **Save**.

For systems that the MegaWizard Plug-In Manager generates, the corename is the top-level component name and the instance name is the full path to the instance. The instance name is in the <*IP core name*>**_all_pins.txt** file that generates automatically when you run the <*IP core name*>**_pin_assignments.tcl** script.

5. Browse to **core2_example_design\example_project\core2_example\submodules\core2_example_if0_p0_report_timing_core.tcl** and click **Open**.
6. Locate the following assignments:

```
set dqs_periphery_node
${inst}|controller_phy_inst|memphy_top_inst|umemphy|uio_pads|dq_ddi
o[${leveling_delay_chain_number}].ubidir_dq_dqs|altdq_dqs2_inst|the
chain|clkin

if {$isdmpin == 1} {

    set dq_periphery_node
${inst}|controller_phy_inst|memphy_top_inst|umemphy|uio_pads|dq_ddi
o[${leveling_delay_chain_number_dq}].ubidir_dq_dqs|altdq_dqs2_inst|
thechain|clkin

} else {

    set dq_periphery_node
${inst}|controller_phy_inst|memphy_top_inst|umemphy|uio_pads|dq_ddi
o[${leveling_delay_chain_number_dq}].ubidir_dq_dqs|altdq_dqs2_inst|
thechain|clkin

}
```

and change to the following:

```
set dqs_periphery_node
*|*|controller_phy_inst|memphy_top_inst|umemphy|uio_pads|dq_ddio[${
leveling_delay_chain_number}].ubidir_dq_dqs|altdq_dqs2_inst|thechai
n|clkin

if {$isdmpin == 1} {

    set dq_periphery_node
*|*|controller_phy_inst|memphy_top_inst|umemphy|uio_pads|dq_ddio[${
leveling_delay_chain_number_dq}].ubidir_dq_dqs|altdq_dqs2_inst|thec
hain|clkin

} else {

    set dq_periphery_node
*|*|controller_phy_inst|memphy_top_inst|umemphy|uio_pads|dq_ddio[${
leveling_delay_chain_number_dq}].ubidir_dq_dqs|altdq_dqs2_inst|thec
hain|clkin

}
```

7. Click **Save**.

Apply the PLL sharing constraints to all the slave controllers if you have more than one slave controller in your design.

Setting Pins

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

1. In the **Assignments** list, select **Device** and click **Device and Pin Options**.
2. Click the **Unused Pins** tab, and for **Reserve all unused pins** select **As input tri-stated with weak pull-up**.
3. Click the **Voltage** tab, and for **Default I/O standard** select **1.5 V.**
4. Click **OK**.

Adding Example Project

Add the example project files to your project, by performing the following steps:

1. On the Project menu, click **Add/Remove Files in Project**.
2. Browse to the <*variation_name*>**_example_design\example_design** directory.
3. Select <*variation_name*>**_example.qip** for the master DDR3 SDRAM component, and click **Open**.

   The parameter interface generates the **.qip** file, which contains information about the generated IP core. In most cases, the **.qip** file contains all of the necessary assignments and information required to process the MegaCore function or system in the Quartus II compiler. The parameter interface generates a single **.qip** file for each MegaCore function.

4. Click **OK**.
5. Open <*variation_name*>**_example.qip** for the slave DDR3 SDRAM component, and remove the duplicate file in the slave DDR3 SDRAM component,or use the Altera-provided **.qip** file.
6. Click **Save**.

Set Top-Level Entity

The top-level entity of the project must be set to the correct entity. To set the top-level file, perform the following steps:

1. On the File menu, click **Open**.
2. Browse to the <*top_level_name*>**.v** or **.vhd** file and click **Open**, or open the Altera-provided top-level file.
3. On the Project menu, click **Set as Top-Level Entity**.

Adding Pin and DQ Group Assignments

The pin assignment scripts, **core1_example_if0_p0_pin_assignment.tcl** and **core2_example_if0_p0_pin_assignment.tcl**, set up the I/O standards for the DDR3 SDRAM and QDR II+ SRAM with UniPHY interface. These scripts do not include assignments for the design's PLL input clock. You must create a PLL clock for the design, and provide pin assignments for the signals of both the example driver and testbench that the MegaCore variations generate.

Run the **core1_example_if0_p0_pin_assignment.tcl** and **core2_example_if0_p0_pin_assignment.tcl** scripts to add the pins, I/O standards, and DQ group assignments to the design example. To run the pin assignment scripts, perform the following steps:

1. On the Processing menu, point to **Start** and then click **Start Analysis & Synthesis**.
2. In the Tools menu, click **Tcl Scripts**.
3. Locate the **core1_example_if0_p0_pin_assignments.tcl** file.
4. Click **Run**.
5. Repeat steps 1 to 4 to locate and run the **core2_example_if0_p0_pin_assignments.tcl** script.

The PLL input clock I/O does not need to have the same I/O standard as the memory interface I/Os. However, you may see a no-fit error as the bank in which the PLL input clock I/O gets placed becomes unusable for placement of the memory interface I/Os because of the incompatible V_{CCIO} levels. Altera recommends that you assign the same I/O standard to the PLL input clock I/O.

Assigning Pins

To enter the pin location assignments, assign all of your pins, so that the Quartus II software fits your design correctly and performs correct timing analysis. To assign the pin locations, run the Altera-provided **sv_multiple_ddr3_pin_location.tcl**.

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

1. On the Assignments menu, click **Pin Planner**.
2. To view the DQS groups in Pin Planner, right-click and select **Show DQ/DQS Pins**, and click **In ×16/×18 Mode**. The Pin Planner shows each DQS group in a different color and with a different legend: S = `DQS` pin, Sbar = `DQSn` pin and Q = `DQ` pin.
3. To identify differential I/O pairs, right-click in Pin Planner and select **Show Differential Pin Pair Connections**. The Pin Planner shows a red line between each pin pair.

For more information about pin location assignments, refer to the *Device and Pin Planning* section in volume 2 of the *External Memory Interface Handbook*. For more information about how to use the Quartus II Pin Planner, refer to the *I/O Management* chapter in volume 2 of the *Quartus II Handbook*.

Perform Advanced I/O Timing Analysis with Board Trace Delay Model

You should use this method only if you are unable to perform post-layout simulation on the memory interface signals to obtain the slew rate parameters.

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

To perform advanced I/O timing analysis with the Quartus II software, follow these steps:

1. After the instantiation is complete, analyze and synthesize your design.
2. Add pin and DQ group assignment by running the *<variation_name>***_p0_pin_assignments.tcl** script.
3. Enter the pin location assignments.
4. Assign the virtual pins, if necessary.
5. Enter the board trace model information.

To enter board trace model information, follow these steps:

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

   Alternatively, you can run the Altera-provided *<variation_name>***_BTModels.tcl** file to apply the board trace model information.

3. Compile your design. To compile the design, on the Processing menu, click **Start Compilation**.
4. After successfully compiling the design, perform timing analysis in the TimeQuest timing analyzer.

To perform timing analysis, follow these steps:

1. In the Quartus II software, on the Tools menu, click **TimeQuest Timing Analyzer**.
2. On the **Tasks** pane, click **Report DDR**.
3. On the **Report** pane, select **Advanced I/O Timing>Signal Integrity Metrics**.
4. In the **Signal Integrity Metrics** window, right-click and select **Regenerate** to regenerate the signal integrity metrics.
5. In the **Signal Integrity Metrics** window, note the 10–90% rise time (or fall time if fall time is worse) at the far end for CK/CK#, address, and command, DQS/DQS#, and DQ signals.
6. In the DDR3 SDRAM controller parameter editor, in the Board Settings tab, type the values you obtained from the signal integrity metrics.

7. For the board skew parameters, set the maximum skew between DQS groups of your design. Set the other board parameters to 0 ns.
8. Compile your design.

Compiling Design and Verifying Timing

To compile the design, on the Processing menu, click **Start Compilation**. After successfully compiling the design, the Quartus II software automatically runs the verified timing script of the master DDR3 SDRAM memory, <*variation_name*>**_p0_report_timing.tcl**, which produces a timing margin report for the design together with the compilation report.

The report timing script performs the following tasks:

1. Creates a timing netlist.
2. Reads the <*variation_name*>**.sdc** file.
3. Updates the timing netlist.

You can also obtain the timing report by running the report timing script in the TimeQuest timing analyzer. To obtain the timing report in the TimeQuest Timing Analyzer window, perform the following steps:

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

The results are the same as the Quartus II software results.

For more information about the TimeQuest timing analyzer, refer to *The Quartus II TimeQuest Timing Analyzer* chapter in volume 3 of the *Quartus II Handbook*.
For information timing analysis, refer to the *Timing Analysis* section in volume 4 of the *External Memory Interface Handbook*.

Performing RTL Simulation (Optional)

This section describes RTL simulation. The Quartus II software automatically generates a RTL simulation project file for both the master and slave DDR3 SDRAM instances with UniPHY. You can obtain the RTL simulation project file from the <*variation_name*>**_example_design\simulation** folder.

When creating a multiple-memory interface design, you must modify the testbench to perform functional simulation. You can create a testbench that combines all the memory interfaces in the design or simulate the master component. You cannot use the RTL simulation project file for a stand-alone slave component, because the slave component requires you to provide a master interface. You can use this project file to verify your design or to create an RTL simulation on the top-level project file.

To run the RTL simulation with NativeLink, follow these steps:

1. Set the absolute path to your third-party simulator executable, by performing the following steps:
   a. On the Tools menu, click **Options**.
   b. In the **Category** list, select EDA Tools Options and set the default path for **ModelSim-Altera** to **C:\<version>\modelsim_ae\win32aloem**.
   c. Click **OK**.
2. On the Assignments menu, click **Settings**.
3. In the **Category** list, expand **EDA Tool Settings** and click **Simulation**.
4. Under **Tool name**, select **ModelSim-Altera**.
5. Under **NativeLink settings**, select **Compile test bench** and click **Test Benches**.
6. Click **New**.
7. In the **Edit Test Bench Settings** dialog box, perform the following steps:
   a. For **Test bench name**, type `core1_example_sim` for the master DDR3 SDRAM.
   b. For **Top level module in test bench**, type `core1_example_sim_tb` for the master DDR3 SDRAM.
   c. Under **Simulation period**, select **Run simulation until all vector stimuli are used**.
   d. In the **Test bench files** field, include the testbench files listed in the **testbench.txt** file from the **emi_multiple_ddr3_sv.zip** folder.
   e. Click **OK**.
8. To elaborate your design, on the Processing menu, point to **Start** and click **Start Analysis & Elaboration**.
9. On the Tools menu, point to the **Run EDA Simulation Tool** and click **EDA RTL Simulation**. This step creates the **\simulation** directory in your project directory and a script that compiles all necessary files and runs the simulation.
10. Add all the signal to the wave window and click **Run all** to run the simulation.

3. DDR3 SDRAM Controller with UniPHY Using Qsys

This tutorial shows how to use the Qsys design flow to design an 8-bit wide, 400-MHz DDR3 SDRAM interface. The design example also provides some recommended settings to simplify the design.

The design example targets the Stratix IV FPGA development kit which includes a 16-bit wide, 1-Gb Micron MT41J64M16LA-15E 667-MHz DDR3 SDRAM component.

System Requirement

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

- Quartus II software version 10.1
- ModelSim-SE 6.6c or higher
- DDR3 SDRAM Controller with UniPHY version 11.0
- Nios II Embedded Design Suite (EDS) version 11.0

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

Creating Your Example Project

This section shows you how to create a new Quartus II project and a Qsys system.

Creating a New Quartus II Project

In the Quartus II software, create a new project that targets an EP4SGX230KF40C2 device.

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

Creating the Qsys System

To create a Qsys system, follow these steps:

1. On the Tools menu, click **Qsys**.
2. Under **Component Library**, expand **Memories and Memory Controllers>External Memory Interfaces>DDR3**, select **DDR3 SDRAM Controller with UniPHY** and click **Add**.
3. The **DDR3 SDRAM Controller with UniPHY** parameter editor appears.
4. In the **Presets** list, select **MICRON MT41J64M16LA-15E** and click **Apply**. The memory parameters are set automatically based on the selected memory preset settings. You may change the parameters to suit your design requirements.

   Selecting a new memory preset overwrites your existing settings. You need to enter the settings again.

5. In the **PHY Settings** tab, under **FPGA**, for **Speed Grade** select **2** to match the chosen device.
6. Under **Clocks**, for **Memory clock frequency**, type `400` MHz as the system frequency.
7. For **PLL reference clock frequency**, type `100` MHz to match the on-board oscillator.

   ☞ DDR3 SDRAM controllers only support half-rate mode.

8. Under **Advanced PHY Settings**, turn on **Advanced clock phase control** and key in the values for **Additional address and command clock phase** and **Additional CK/CK# phase** if you want to increase or decrease the amount of phase shift on the clocks. However, Altera recommends that you retain the default values.
9. For **PLL sharing mode**, **DLL sharing mode**, and **OCT sharing mode**, select **No Sharing**.
10. Turn off **HardCopy compatibility**. You cannot migrate this design to HardCopy.
11. In the **Memory Parameters** tab, for **Total Interface Width**, select **8**.
12. Under **Memory Initialization Options**, for **Memory CAS latency setting**, select **8**.
13. For **Output drive strength setting**, select **RZQ/7**.
14. For **ODT Rtt nominal value**, select **RZQ/4**.
15. For **Memory write CAS latency setting**, select **6**.
16. For **Dynamic ODT (Rtt_WR) value**, select **RZQ/4**.
17. In the **Board Settings** tab under **Setup and Hold Derating**, for **Derating method**, select **Specify slew rates to calculate setup and hold times**.
18. Set the slew rate parameters to the specified values in Table 3–1.

Table 3–1. Slew Rate Parameters for Stratix IV Devices

Slew Rate Parameter	Slew Rate (V/ns)
CK/CK# slew rate (Differential)	5.117
Address and command slew rate	5.745
DQS/DQS# slew rate (Differential)	2.257
DQ slew rate	4.581

☞ If your design is targeting a Stratix III device, you have to manually derate t_{DS}, t_{DH}, t_{IS}, and t_{IH} parameters in the **Memory Timing** tab. For more information about how to derate memory setup and hold timing , refer to the *DDR2 and DDR3 SDRAM Controller with UniPHY User Guide* in volume 3 of the *External Memory Interface Handbook*.

☞ The Intersymbol Interference (ISI) parameters are not applicable for single chip-select configurations or designs that target Stratix III devices. Set these parameters to **0 ps**.

19. Set the **Board Skews** parameters to the specified values in Table 3–2.

Table 3–2. Board Skews Parameters for Stratix IV Devices

Board Skews Parameter	Skew (ns)
Maximum CK delay to DIMM/device	0.600
Maximum DQS delay to DIMM/device	0.600
Minimum delay difference between CK and DQS	–0.079
Maximum delay difference between CK and DQS	–0.021
Maximum delay difference between DIMMs/devices	0.017
Maximum skew within DQS group	0.017
Maximum skew between DQS groups	0.058
Average delay difference DQ and DQS	–0.009
Maximum skew within address and command bus	0.084
Average delay difference between address and command and CK	–0.025

Altera recommends that you use a third party tool to perform whole path simulation for DQ, DQS, CK, address, and command signals to obtain the slew rate and board skew parameters for your design. If you are unable to perform post-layout simulation to obtain the slew rate parameters. You can use the board trace delay model. For more information, refer to the "Performing Advanced I/O Timing Analysis with Board Trace Delay Model" on page 3–14.

20. In the **Controller Settings** tab, under **Avalon Interface**, turn on **Generate power-of-2 data bus widths for SOPC Builder**.
21. For **Maximum Avalon-MM burst length**, specify the burst length based on the burst count sent from the core fabric. For example, if your system generates up to 16 beats, then select **16**.
22. Under **Efficiency**, for **Local-to-Memory Address Mapping**, select **CHIP-ROW-BANK-COL**.
23. For **Command Queue Look-Ahead Depth**, select **6**.
24. In the **Diagnostics** tab, under **Simulation Options**, for **Auto-calibration mode**, select **Skip calibration** and turn on **Skip memory initialization**. Specifying these options allows you to skip memory initialization and calibration during simulation so that you can reduce the simulation time.
25. Click **Finish**.

Adding Qsys Components

To add components from the **System Contents** tab, follow these 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 `65536` bytes.
   b. Click **Finish**.

2. Select **On-Chip Memory (RAM or ROM)** again, click **Add**.
   a. For **Total memory size**, type `4096` bytes**.**
   b. Click **Finish.**
   c. In the **System Contents** tab, in the **Name** column, right click on this second on-chip memory and click **Rename**.
   d. Type `dma_read_memory` and press Enter.
3. On the System menu, click **Assign Base Addresses**.
4. Under **Component Library**, expand **Processors**, select **Nios II Processor** and click **Add**.
   a. Select **Nios II/s**.
   b. Set **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 resets and in doing so writes to addresses `0x00` to `0x1f`. If you want your memory contents to remain intact through a system reset, avoid using the memory addresses below `0x20`. This step is not necessary if you reload your SDRAM contents from flash every time you reset.

   c. Click **Finish**.
5. Expand **Interface Protocols** and expand **Serial**. Select **JTAG UART** and click **Add**.
   a. Under **Write FIFO** and **Read FIFO**, for the **Buffer depth (bytes)** select **64**, and for **IRQ threshold** type `8`.
   b. For **Prepare interactive windows**, select **INTERACTIVE_INPUT_OUTPUT** to open the interactive display window during simulation.
   c. Click **Finish**.
6. Expand **Peripherals** and expand **Microcontroller Peripherals**. Select **PIO (Parallel I/O)** and click **Add**.
   a. For **Width**, type `8` bits.
   b. For **Direction**, select **Output**.
   c. Click **Finish**.
7. Expand **Bridges and Adapters** and expand **DMA**. Select **DMA Controller** and click **Add**.
   a. In the **Advanced** tab, turn off **byte**, **halfword**, **doubleword** and **quadword**.
   b. Click **Finish.**

   Do not to add a PLL component to your Qsys design because the DDR3 SDRAM Controller with UniPHY IP already includes one.

8. In the **Project Settings** tab, for **Clock Crossing Adapter Type**, select **Auto**.

9. In the **Sytem Contents** tab, to set the bus links, follow these steps:
   a. Make the appropriate bus connections as shown in Table 3–3.

Table 3–3. Bus Connections

Source Port	Destination Port
`dma.read_master`	`uniphy_ddr3.avl` `dma_read_memory.s1`
`dma.write_master`	`uniphy_ddr3.avl` `dma_read_memory.s1`

`cpu.data_master`	`cpu.jtag_debug_module`
	`jtag_uart.avalon_jtag_slave`
	`Led_pio.s1`
	`dma.control_port_slave`
	`dma_read_memory.s1`
	`on_chip_memory.s1`
	`uniphy_ddr3.avl`

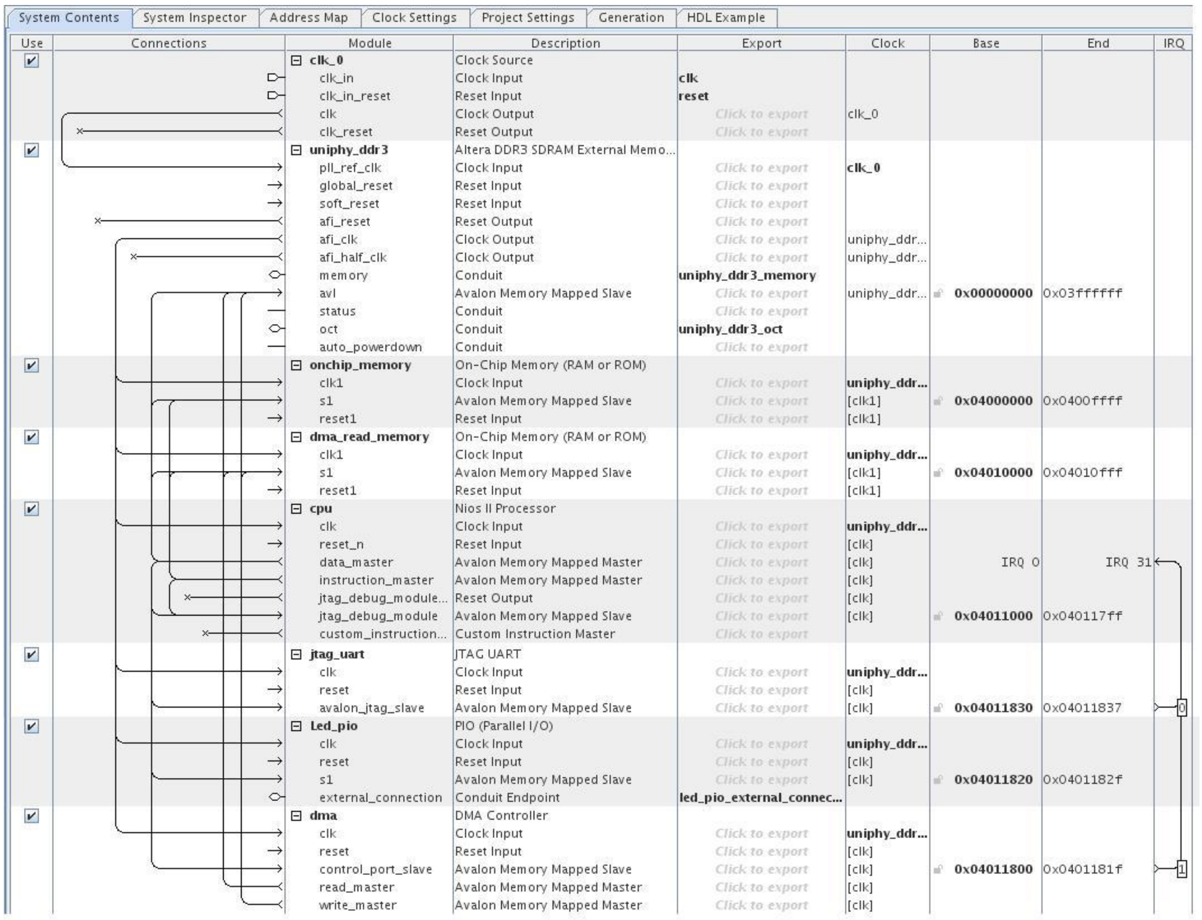
`afi.clk`	`on_chip_memory.clk1`
	`Led_pio.clk`
	`dma_read_memory.clk1`
	`cpu.clk`
	`jtag_uart.clk`
	`dma.clk`

`cpu.instruction_master`	`cpu.jtag_debug_module`
	`dma_read_memory.s1`
	`on_chip_memory.s1`

`clk_0.clk`	`uniphy_ddr3.pll_ref_clk`

b. Connect the interrupt request (IRQ) lines as shown in Figure 3–1.

Figure 3–1. Setting Bus Links



System Contents | System Inspector | Address Map | Clock Settings | Project Settings | Generation | HDL Example

Use	Connections	Module	Description	Export	Clock	Base	End	IRQ
✓		**clk_0**	Clock Source					
		clk_in	Clock Input	**clk**				
		clk_in_reset	Reset Input	**reset**				
		clk	Clock Output	Click to export	clk_0			
		clk_reset	Reset Output	Click to export				
✓		**uniphy_ddr3**	Altera DDR3 SDRAM External Memo...					
		pll_ref_clk	Clock Input	Click to export	**clk_0**			
		global_reset	Reset Input	Click to export				
		soft_reset	Reset Input	Click to export				
		afi_reset	Reset Output	Click to export				
		afi_clk	Clock Output	Click to export	uniphy_ddr...			
		afi_half_clk	Clock Output	Click to export	uniphy_ddr...			
		memory	Conduit	**uniphy_ddr3_memory**				
		avl	Avalon Memory Mapped Slave	Click to export	uniphy_ddr...	**0x00000000**	0x03ffffff	
		status	Conduit	Click to export				
		oct	Conduit	**uniphy_ddr3_oct**				
		auto_powerdown	Conduit	Click to export				
✓		**onchip_memory**	On-Chip Memory (RAM or ROM)					
		clk1	Clock Input	Click to export	**uniphy_ddr...**			
		s1	Avalon Memory Mapped Slave	Click to export	[clk1]	**0x04000000**	0x0400ffff	
		reset1	Reset Input	Click to export	[clk1]			
✓		**dma_read_memory**	On-Chip Memory (RAM or ROM)					
		clk1	Clock Input	Click to export	**uniphy_ddr...**			
		s1	Avalon Memory Mapped Slave	Click to export	[clk1]	**0x04010000**	0x04010fff	
		reset1	Reset Input	Click to export	[clk1]			
✓		**cpu**	Nios II Processor					
		clk	Clock Input	Click to export	**uniphy_ddr...**			
		reset_n	Reset Input	Click to export	[clk]			
		data_master	Avalon Memory Mapped Master	Click to export	[clk]	IRQ 0	IRQ 31	
		instruction_master	Avalon Memory Mapped Master	Click to export	[clk]			
		jtag_debug_module...	Reset Output	Click to export	[clk]			
		jtag_debug_module	Avalon Memory Mapped Slave	Click to export	[clk]	**0x04011000**	0x040117ff	
		custom_instruction...	Custom Instruction Master	Click to export				
✓		**jtag_uart**	JTAG UART					
		clk	Clock Input	Click to export	**uniphy_ddr...**			
		reset	Reset Input	Click to export	[clk]			
		avalon_jtag_slave	Avalon Memory Mapped Slave	Click to export	[clk]	**0x04011830**	0x04011837	0
✓		**Led_pio**	PIO (Parallel I/O)					
		clk	Clock Input	Click to export	**uniphy_ddr...**			
		reset	Reset Input	Click to export	[clk]			
		s1	Avalon Memory Mapped Slave	Click to export	[clk]	**0x04011820**	0x0401182f	
		external_connection	Conduit Endpoint	**led_pio_external_connec...**				
✓		**dma**	DMA Controller					
		clk	Clock Input	Click to export	**uniphy_ddr...**			
		reset	Reset Input	Click to export	[clk]			
		control_port_slave	Avalon Memory Mapped Slave	Click to export	[clk]	**0x04011800**	0x0401181f	1
		read_master	Avalon Memory Mapped Master	Click to export	[clk]			
		write_master	Avalon Memory Mapped Master	Click to export	[clk]			

If there are warnings about overlapping addresses, on the System menu click **Assign Base Addresses**.
If there are warnings about overlapping IRQ, on the System menu click **Assign Interrupt Numbers**.

c. Click **Export as** column to export the listed signals as input, output, or bidirectional pins for the following modules :

- **Led_pio**—`external_connection`
- **uniphy_ddr3**—`memory`, `oct`
- **clk_0**—`clk_in`, `clk_in_reset`
- **cpu**—`reset_n`

d. Make the appropriate reset connections as shown in Table 3–4.

Table 3–4. Reset Connections

Source Port	Destination Port
`clk0.clk.reset`	`uniphy_ddr3.global_reset`
	`onchip_memory.reset1`
	`dma_read_memory.reset1`
	`cpu.reset_n`
	`jtag_uart.reset`
	`Led_pio.reset`
	`dma.reset`
`uniphy_ddr3.afi_reset`	`onchip_memory.reset1`
	`dma_read_memory.reset1`
	`cpu.reset_n`
	`jtag_uart.reset`
	`Led_pio.reset`
	`dma.reset`
`cpu. jtag_debug_module_reset`	`uniphy_ddr3.global_reset`
	`onchip_memory.reset1`
	`dma_read_memory.reset1`
	`cpu.reset_n`
	`jtag_uart.reset`
	`Led_pio.reset`
	`dma.reset`

e. Ensure that all other modules are clocked on **uniphy_ddr3.afi_clk** to avoid any unnecessary clock-domain crossing logic.

10. After setting up the connections, right-click the **cpu** module and select **Edit** to open the **Nios II Processor** parameter editor. In the **Core Nios II** tab, for **Reset vector memory** and **Exception vector memory**, select **onchip_memory.s1**, and click **Finish**.

☞ If the local on-chip memory holds the Nios II instruction code, the SDRAM interface requires less arbitration and results in a more optimal Avalon-MM structure.

11. On the File menu, click **Save** to save the Qsys system.
12. In the **Generation** tab, under **Synthesis**, turn on **Create HDL design files for synthesis** and **Create block symbol file (.bsf)**.
13. Click **Generate** to generate the design.

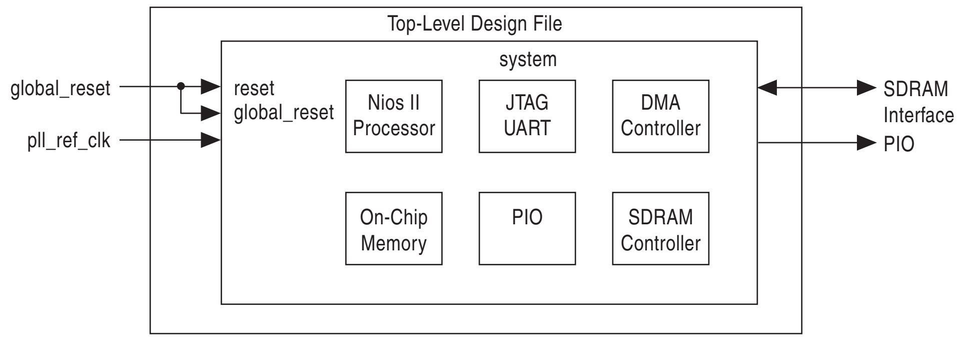
Creating the Top-Level Design File

Conceptually, you can consider the Qsys system as a component in your design. The Qsys system can be the only component; or one of many components. Hence, when your Qsys 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 Qsys system with no additional components. The top-level design file just defines the pin naming convention and port connections. Figure 3–2 shows the Qsys top-level block diagram.

Figure 3–2. Qsys Top-Level Block Diagram



To create a top-level design for your Qsys 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 **system**, click **OK**.

6. Position the Qsys system component outline in the <*project*>**.bdf** and left-click.
7. Right-click on the Qsys system component and click **Generate Pins for Symbol Ports**, to automatically add pins and nets to the schematic symbol.
8. Rename the following pins to the modified pin names as shown Table 3–5.

Table 3–5. Rename Pin

Existing Pin Name	Modified Pin Name
clk_clk	pll_ref_clk
reset_reset_n	global_reset_n
led_pio_external_connection_export[7..0]	pio[7..0]
oct_rup	rup
oct_rdn	rdn
uniphy_ddr3_memory_mem_reset_n	mem_reset_n
uniphy_ddr3_memory_mem_dqs_n	mem_dqs_n[0] *(1)*
uniphy_ddr3_memory_mem_ck_n	mem_ck_n[0] *(1)*
uniphy_ddr3_memory_mem_a[12..0]	mem_a[12..0] *(1)*
uniphy_ddr3_memory_mem_ras_n	mem_ras_n[0] *(1)*
uniphy_ddr3_memory_mem_odt	mem_odt[0] *(1)*
uniphy_ddr3_memory_mem_we_n	mem_we_n[0] *(1)*
uniphy_ddr3_memory_mem_cs_n	mem_cs_n[0] *(1)*
uniphy_ddr3_memory_mem_dq[7..0]	mem_dq[7..0] *(1)*
uniphy_ddr3_memory_mem_ck	mem_ck[0] *(1)*
uniphy_ddr3_memory_mem_dm	mem_dm[0] *(1)*
uniphy_ddr3_memory_mem_dqs	mem_dqs[0] *(1)*
uniphy_ddr3_memory_mem_cas_n	mem_cas_n[0] *(1)*
uniphy_ddr3_memory_mem_ba[2..0]	mem_ba[2..0] *(1)*
uniphy_ddr3_memory_mem_cke	mem_cke[0] *(1)*

Note to Table 3–5:

(1) The single vector signals must have [0] vectored pin names, otherwise your design may fail to simulate.

9. On the File menu, click **Save**, to save your changes.
10. On the Project menu, click **Set as Top-Level Entity**.

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

Adding Constraints

After generating the DDR3 SDRAM Controller with UniPHY, the UniPHY IP generates the constraint files for the design. You need to apply these constraints to the design before compilation.

Setting Pins

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. On the Assignment menu, select **Device** and click **Device and Pin Options**.
2. Click the **Unused Pins** tab, and for **Reserve all unused pins** select **As input tri-stated with weak pull-up**.
3. Click the **Voltage** tab, and for **Default I/O standard** select the same VCCIO voltage as the chosen SDRAM interface; for DDR3 SDRAM **select 1.5 V**.
4. Click **OK**.

Assigning I/O Standards

To assign I/O standards, follow these steps:

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

Adding the Quartus II IP File

When you instantiate an SDRAM controller with UniPHY, it automatically generates a Quartus II IP File (**.qip**), **system.qip**.

To add **.qip**, follow these steps:

1. On the Project menu, click **Add/Remove Files in Project**.
2. Browse to the **system/synthesis** directory, and add <*memory_instance*>**.qip**.
3. Click **OK**.

Setting 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, follow these steps:

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

Setting Fitter Effort

To set the Fitter effort, follow these 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**.

Adding Pin, DQ Group, and IO Standard Assignments

The pin assignment script, **uniphy_ddr3_pin_assignments.tcl**, sets up the I/O standards for the DDR3 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 **uniphy_ddr3_pin_assignments.tcl** script to add the pin, I/O standards, and DQ group assignments to the design example, follow these steps:

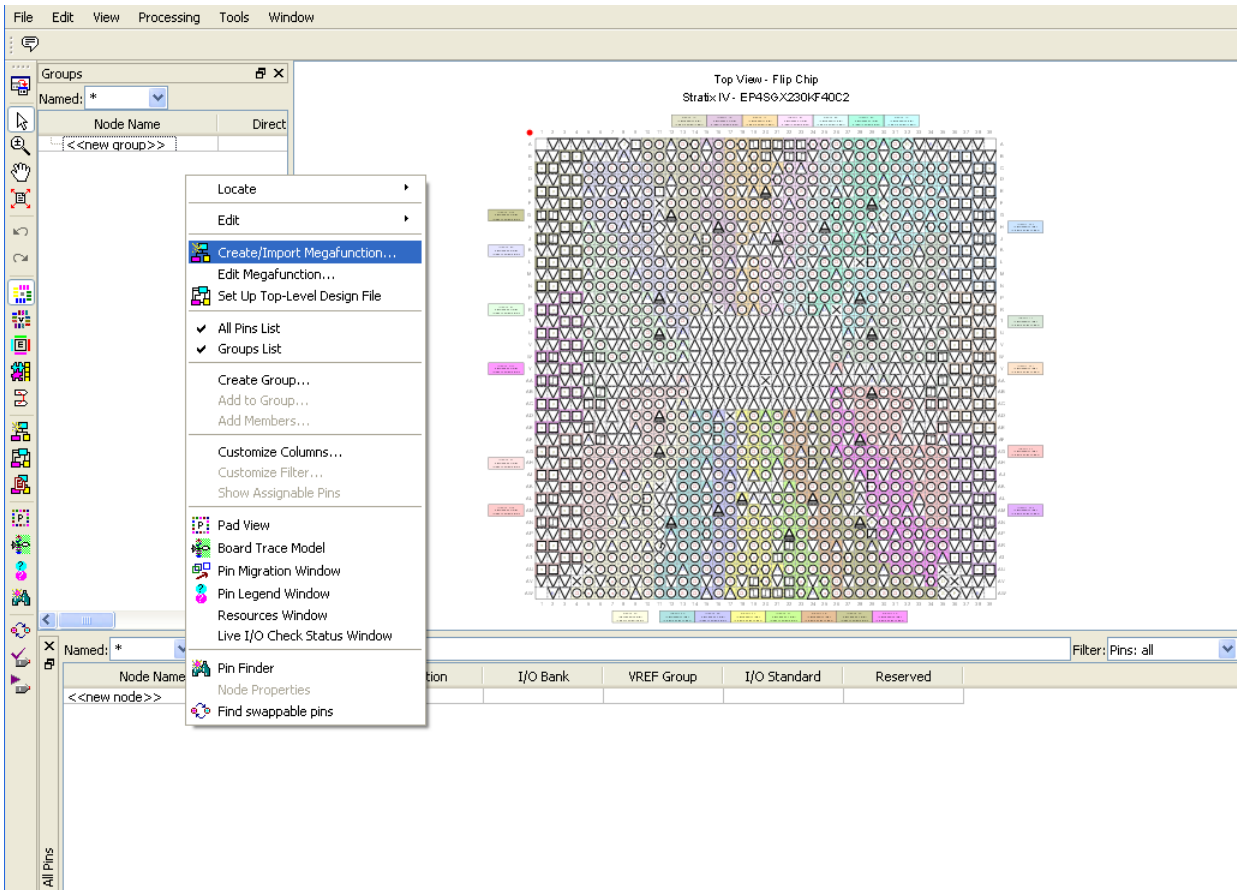
1. On the Processing menu, point to **Start** and click **Start Analysis & Synthesis** to perform synthesis.
2. On the Tools menu, click **Tcl Scripts**.
3. Select **uniphy_ddr3_pin_assignments.tcl**.
4. Click **Run**.

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 3–3.
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.

Figure 3–3. Create/Import Megafunction in Pin Planner



Assigning Pins

To assign pin locations, follow these steps:

1. 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 IV development board, run the Altera-provided **S4_DDR3_Qsys_Pin_Location.tcl** file or manually assign pin locations by using either the Pin Planner or Assignment Editor.

   The SDRAM controller with UniPHY 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, follow these steps:

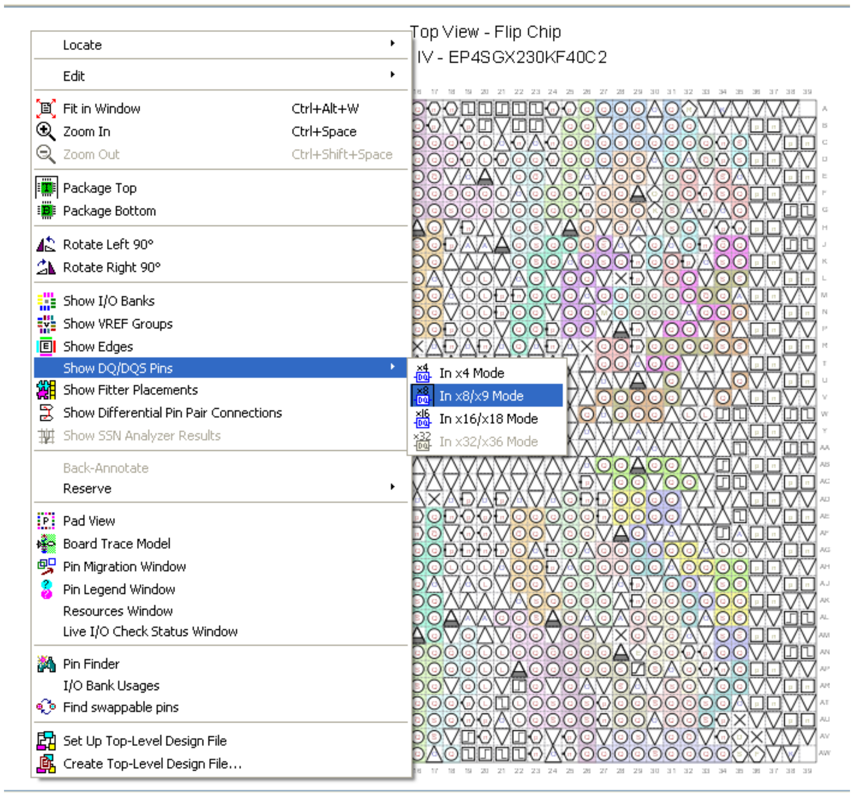
1. On the Assignments menu, click **Pin Planner**.
2. Assign DQ and DQS pins.
   a. To select the device DQS pin groups that the design uses, assign each DQS pin in your design to the required DQS pin in the Pin Planner. The Quartus II Fitter then automatically places the respective DQ signals onto suitable 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 3–4 on page 3–13.

   Most DDR3 SDRAM devices operate in ×8/×9 mode, however as 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 DQ pins in the same group as their respective strobe pin.

Figure 3–4. 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_ck pins.
5. Ensure the mem_ck 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.

   The DDR3 SDRAM in Stratix III and Stratix IV devices with differential DQS mode require the mem_ck[0]/mem_ck_n[0] pin on a DIFFIO_RX capable pin pair. The DDR3 SDRAM interfaces in Stratix III and Stratix IV devices require the mem_ck[0]/mem_ck_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 pll_ref_clk 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 Stratix IV external memory pin selection, refer to the *External Memory Interfaces in Stratix IV Devices* chapter in the *Stratix IV Device Handbook*.

Performing Advanced I/O Timing Analysis with Board Trace Delay Model

You should only use this method if you are unable to perform post-layout simulation on the memory interface signals to obtain the slew rate parameters.

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 perform advanced I/O timing analysis with the Quartus II software, follow these steps:

1. Instantiate and parameterize the Altera DDR3 SDRAM External Memory Interfaces with the required values. However, for **Setup and Hold Derating**, **Inter Symbol Interfence**, and **Board Skew** parameters maintain the default values.
2. When your design is complete, analyze and synthesize your design.

3. Add pin and DQ group assignment by running <*variation_name*>**_pin_assignments.tcl** script.
4. Enter the pin location assignments.
5. Assign the virtual pins, if necessary.
6. Enter the board trace model information. To include the board trace model information, follow these steps:
   a. In the Pin Planner, select the pin or group of pins that you want to enter the information for.
   b. Right-click and select **Board Trace Model**.

   Table 3–6 shows the board trace model parameters for the Stratix IV development board.

Table 3–6. Stratix IV 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_ Length	Cf	Rfh/Rfp
Address (1)	1.668	4.25p	7.61n	—	—	—	—	—	—	5.2p	—
CLK	1.437	4.11p	7.84n	—	—	—	—	—	—	5.6p	100
DQ (2)	1.921	4.25p	7.61n	—	—	—	—	—	—	2.5p	60
DQS	1.921	4.11p	7.84n	—	—	—	—	—	—	2.5p	60

Note to Table 3–6:

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

(2) `DQ` = `DQ` and `DM` pins.

   To apply board trace model assignments for the Stratix IV development board, run the Altera-provided **S4_DDR3_BTModels.tcl** file or manually assign virtual pin assignments using the Quartus II Pin Planner.

7. Compile your design. When compilation completes, launch the TimeQuest timing analyzer. To launch the timing analyzer, follow these steps:
   a. On the Tools menu, click **TimeQuest Timing Analyzer**.
   b. In **Tasks** panel, click **Report DDR**.
   c. In **Report** panel, select **Advanced I/O Timing >Signal Integrity Metrics**.
   d. In the **Signal Integrity Metrics** window, right-click and select **Regenerate** to regenerate the signal integrity metrics.
   e. In the **Signal Integrity Metrics** window, take note of the 10-90% rise time (or fall time if fall time is worse) at the far end for CK/CK#, address and command, DQS/DQS#, and DQ signals.

8. Parameterize the DDR3 SDRAM Controller with UniPHY, follow these steps:
   a. Launch the **DDR3 SDRAM Controller with UniPHY** parameter editor.
   b. In the **DDR3 SDRAM Controller with UniPHY** parameter editor, in the Board Settings tab, for the slew rates, key in with the values you obtain from the signal integrity metrics.
   c. For the board skew parameters, set the maximum skew between DQS groups of your design. Set the other board parameters to 0 ns.
9. Compile your design.

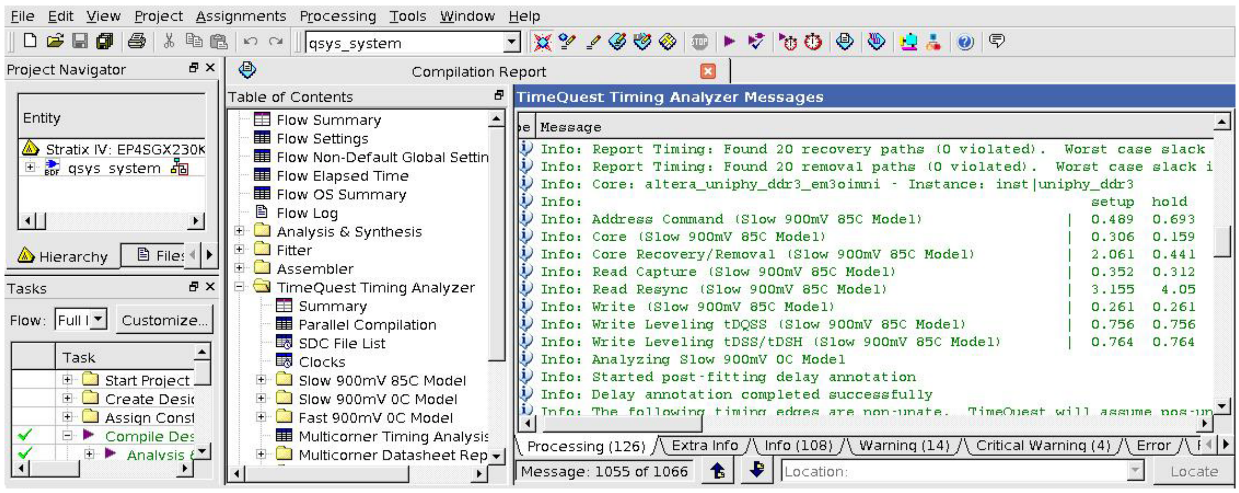
Compiling the Design

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

After successfully compiling the design, the Quartus II software automatically runs the **uniphy_ddr3_report_timing.tcl** file, which produces a timing report for the design together with the compilation report.

Figure 3–5 shows the timing margin report in the message window in the Quartus II software.

Figure 3–5. Timing Margin Report in the Quartus II Software



```
Message
Info: Report Timing: Found 20 recovery paths (0 violated).  Worst case slack
Info: Report Timing: Found 20 removal paths (0 violated).  Worst case slack i
Info: Core: altera_uniphy_ddr3_em3oimni - Instance: inst|uniphy_ddr3
Info:                                                         setup  hold
Info: Address Command (Slow 900mV 85C Model)                |  0.489  0.693
Info: Core (Slow 900mV 85C Model)                           |  0.306  0.159
Info: Core Recovery/Removal (Slow 900mV 85C Model)          |  2.061  0.441
Info: Read Capture (Slow 900mV 85C Model)                   |  0.352  0.312
Info: Read Resync (Slow 900mV 85C Model)                    |  3.155   4.05
Info: Write (Slow 900mV 85C Model)                          |  0.261  0.261
Info: Write Leveling tDQSS (Slow 900mV 85C Model)           |  0.756  0.756
Info: Write Leveling tDSS/tDSH (Slow 900mV 85C Model)       |  0.764  0.764
Info: Analyzing Slow 900mV 0C Model
Info: Started post-fitting delay annotation
Info: Delay annotation completed successfully
Info: The following timing edges are non-unate.  TimeQuest will assume pos-un
```

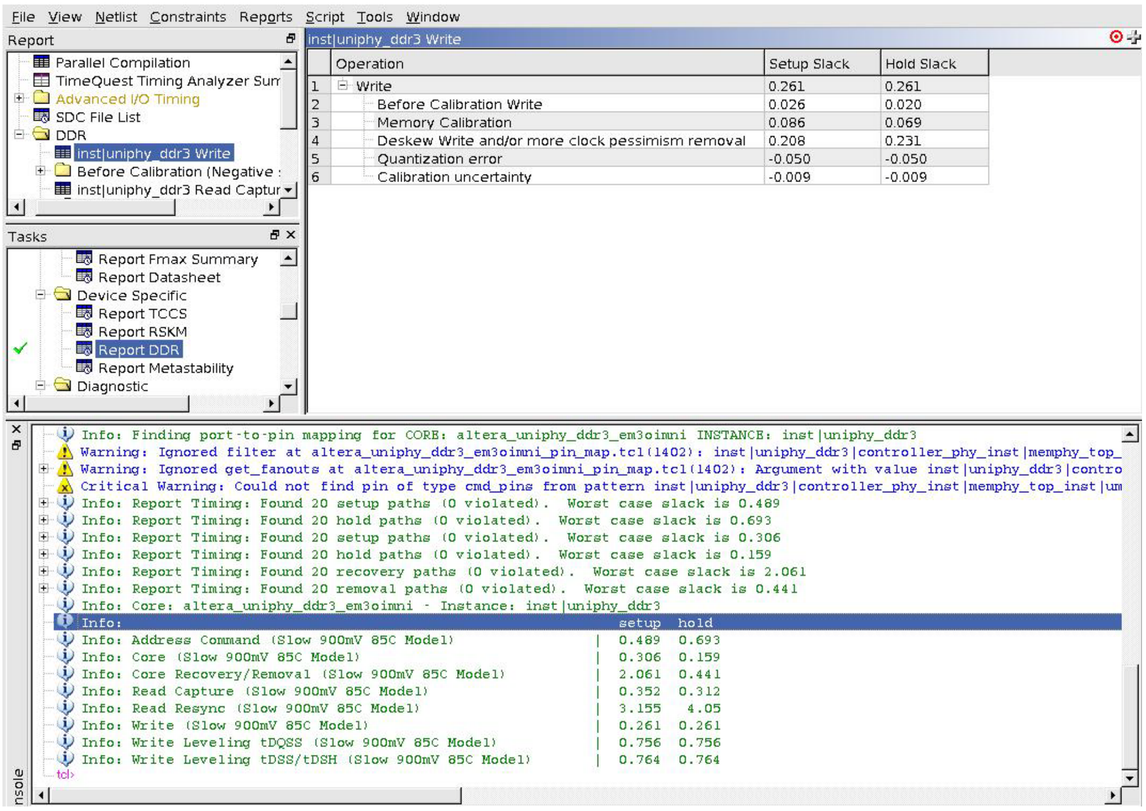
You may also run the report timing script in the TimeQuest Timing Analyzer window, To run the timing script, perform the following steps:

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 3–6 shows the timing margin report in the TimeQuest Timing Analyzer window after running the report timing script. The results are the same as the Quartus II software results.

Figure 3–6. Timing Margin Report in the TimeQuest Timing Analyzer



	Operation	Setup Slack	Hold Slack
1	Write	0.261	0.261
2	Before Calibration Write	0.026	0.020
3	Memory Calibration	0.086	0.069
4	Deskew Write and/or more clock pessimism removal	0.208	0.231
5	Quantization error	-0.050	-0.050
6	Calibration uncertainty	-0.009	-0.009

```
Info: Finding port-to-pin mapping for CORE: altera_uniphy_ddr3_em3oimni INSTANCE: inst|uniphy_ddr3
Warning: Ignored filter at altera_uniphy_ddr3_em3oimni_pin_map.tcl(1402): inst|uniphy_ddr3|controller_phy_inst|memphy_top_
Warning: Ignored get_fanouts at altera_uniphy_ddr3_em3oimni_pin_map.tcl(1402): Argument with value inst|uniphy_ddr3|contro
Critical Warning: Could not find pin of type cmd_pins from pattern inst|uniphy_ddr3|controller_phy_inst|memphy_top_inst|um
Info: Report Timing: Found 20 setup paths (0 violated).  Worst case slack is 0.489
Info: Report Timing: Found 20 hold paths (0 violated).  Worst case slack is 0.693
Info: Report Timing: Found 20 setup paths (0 violated).  Worst case slack is 0.306
Info: Report Timing: Found 20 hold paths (0 violated).  Worst case slack is 0.159
Info: Report Timing: Found 20 recovery paths (0 violated).  Worst case slack is 2.061
Info: Report Timing: Found 20 removal paths (0 violated).  Worst case slack is 0.441
Info: Core: altera_uniphy_ddr3_em3oimni - Instance: inst|uniphy_ddr3
Info:                                                       setup  hold
Info: Address Command (Slow 900mV 85C Model)            |  0.489  0.693
Info: Core (Slow 900mV 85C Model)                       |  0.306  0.159
Info: Core Recovery/Removal (Slow 900mV 85C Model)      |  2.061  0.441
Info: Read Capture (Slow 900mV 85C Model)               |  0.352  0.312
Info: Read Resync (Slow 900mV 85C Model)                |  3.155   4.05
Info: Write (Slow 900mV 85C Model)                      |  0.261  0.261
Info: Write Leveling tDQSS (Slow 900mV 85C Model)       |  0.756  0.756
Info: Write Leveling tDSS/tDSH (Slow 900mV 85C Model)   |  0.764  0.764
tcl>
```

Incorporating the Nios II Eclipse

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

When doing memory tests, you must read and write from the external DDR3 SDRAM and not from cached memory. The following two 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.

Launching the Nios II Eclipse

To launch the Nios II Eclipse, follow these steps:

1. One the Windows Start menu, point to **All Programs**, **Altera**, **Nios II EDS** <*version*>, and then click **Nios II** <*version*> **Software Build Tool for Eclipse**.
2. On the File menu, point to **Switch Workspace**, click **Browse**, and select your project directory.

3. On the File menu, point to **New** and click **Project**.
4. Select **Nios II Application and BSP from Template**, and click **Next**.
5. Select **Blank Project** under Project Templates, browse and locate the SOPC Information File, **system.sopcinfo**, and type `siv_ddr3_uniphy` as project name under the **Application project**.
6. Click **Next** and click **Finish**.
7. In Windows Explorer, drag Altera-supplied **DDR_TEST.c** to the **siv_ddr3_uniphy** directory.

To see the **DDR_TEST.c** example test program, refer to "Example DDR_TEST.c Test Program File" on page 3–27.

This program consists of a simple loop that takes commands from the JTAG UART and executes them. Table 3–7 shows the commands that are sent to the Nios II C code.

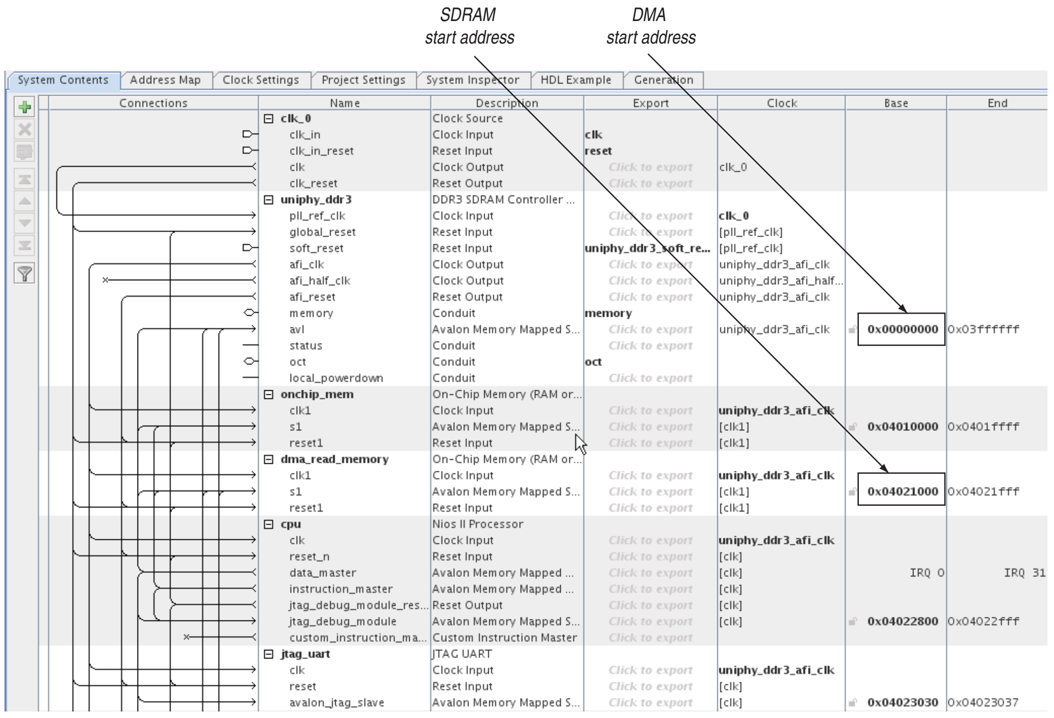
The commands must be in the following format and the switches A, B, C, and D must be entered in upper case. Both address and data strings must always be 8 characters long.

Table 3–7. 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 `12345678`, writes `12345678` 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 `00000001`.
Switch D	Switch | From Address | To Address	Performs an incremental write to the first address location, followed by a DMA transfer from the first address location to the second address location. The burst is a fixed length of 512 words or 2048 bytes.	`D0802100000000000` loads the DMA read memory with the incrementing pattern, then DMA transfers it to the SDRAM high-performance controller.

You can read the Qsys component address locations directly from the Qsys Window, refer to Figure 3–7.

Figure 3–7. Qsys Component Address



Name	Description	Export	Clock	Base	End
clk_0	Clock Source				
clk_in	Clock Input	**clk**			
clk_in_reset	Reset Input	**reset**			
clk	Clock Output	Click to export	clk_0		
clk_reset	Reset Output	Click to export			
uniphy_ddr3	DDR3 SDRAM Controller ...				
pll_ref_clk	Clock Input	Click to export	**clk_0**		
global_reset	Reset Input	Click to export	[pll_ref_clk]		
soft_reset	Reset Input	**uniphy_ddr3_soft_re...**	[pll_ref_clk]		
afi_clk	Clock Output	Click to export	uniphy_ddr3_afi_clk		
afi_half_clk	Clock Output	Click to export	uniphy_ddr3_afi_half...		
afi_reset	Reset Output	Click to export	uniphy_ddr3_afi_clk		
memory	Conduit	**memory**			
avl	Avalon Memory Mapped S...	Click to export	uniphy_ddr3_afi_clk	**0x00000000**	0x03ffffff
status	Conduit	Click to export			
oct	Conduit	**oct**			
local_powerdown	Conduit	Click to export			
onchip_mem	On-Chip Memory (RAM or...				
clk1	Clock Input	Click to export	**uniphy_ddr3_afi_clk**		
s1	Avalon Memory Mapped S...	Click to export	[clk1]	**0x04010000**	0x0401ffff
reset1	Reset Input	Click to export	[clk1]		
dma_read_memory	On-Chip Memory (RAM or...				
clk1	Clock Input	Click to export	**uniphy_ddr3_afi_clk**		
s1	Avalon Memory Mapped S...	Click to export	[clk1]	**0x04021000**	0x04021fff
reset1	Reset Input	Click to export	[clk1]		
cpu	Nios II Processor				
clk	Clock Input	Click to export	**uniphy_ddr3_afi_clk**		
reset_n	Reset Input	Click to export	[clk]		
data_master	Avalon Memory Mapped ...	Click to export	[clk]	IRQ 0	IRQ 31
instruction_master	Avalon Memory Mapped ...	Click to export	[clk]		
jtag_debug_module_res...	Reset Output	Click to export	[clk]		
jtag_debug_module	Avalon Memory Mapped S...	Click to export	[clk]	**0x04022800**	0x04022fff
custom_instruction_ma...	Custom Instruction Master	Click to export			
jtag_uart	JTAG UART				
clk	Clock Input	Click to export	**uniphy_ddr3_afi_clk**		
reset	Reset Input	Click to export	[clk]		
avalon_jtag_slave	Avalon Memory Mapped S...	Click to export	[clk]	**0x04023030**	0x04023037

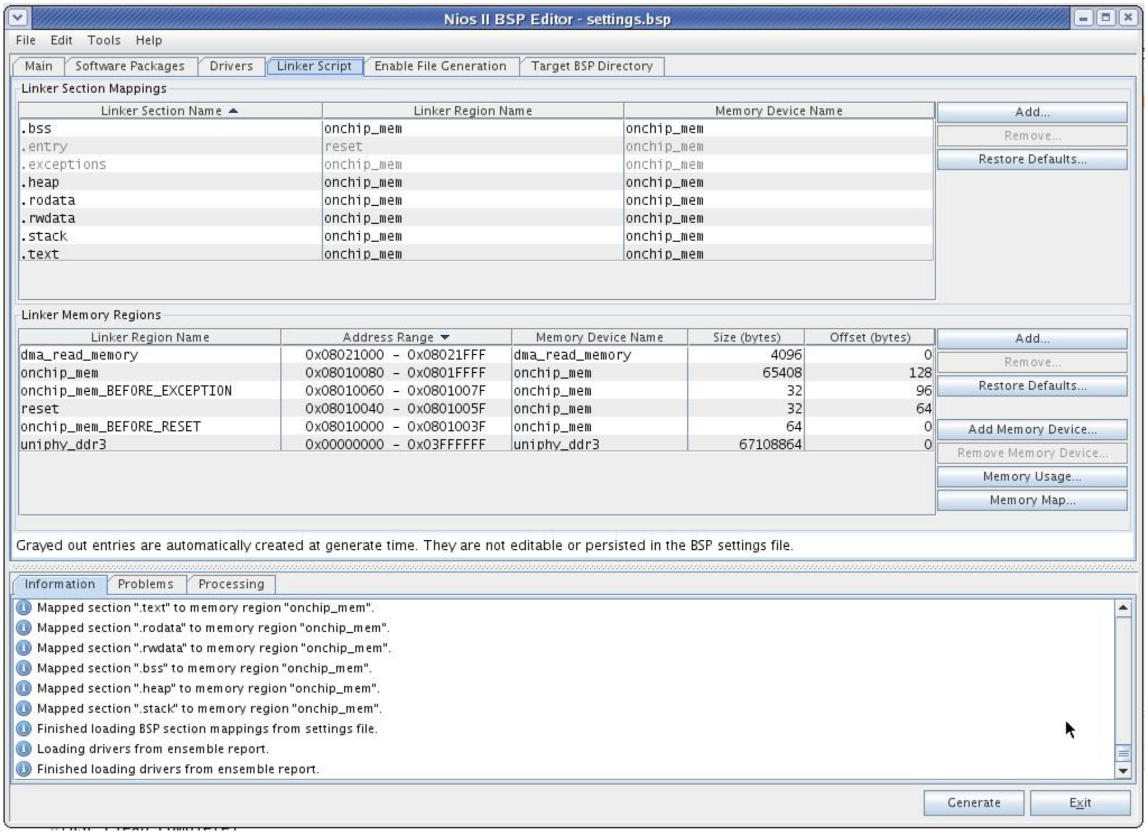
Setting Up the Project Settings

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

1. In the **Project Explorer** tab, right-click <*project_name*> and select **Nios II**, and click **BSP Editor** to launch the **Nios II BSP Editor**.
2. To optimize the footprint size of the library project, in the **Main** tab, point to **Settings**, and turn on **enabled_reduced_device_drivers**.
3. To reduce the memory size allocated for the system library, for **Max file descriptors**, type `4`.
4. In the **Linker Script** tab, select **onchip_mem** as the linker region name for **.bss**, **.heap**, **.rodata**, **.rwdata**, **.stack**, and **.text**.

Figure 3–8 shows the **Nios II BSP Editor** dialog box.

Figure 3–8. Nios II Project Settings



Linker Section Mappings

Linker Section Name	Linker Region Name	Memory Device Name
.bss	onchip_mem	onchip_mem
.entry	reset	onchip_mem
.exceptions	onchip_mem	onchip_mem
.heap	onchip_mem	onchip_mem
.rodata	onchip_mem	onchip_mem
.rwdata	onchip_mem	onchip_mem
.stack	onchip_mem	onchip_mem
.text	onchip_mem	onchip_mem

Linker Memory Regions

Linker Region Name	Address Range	Memory Device Name	Size (bytes)	Offset (bytes)
dma_read_memory	0x08021000 - 0x08021FFF	dma_read_memory	4096	0
onchip_mem	0x08010080 - 0x0801FFFF	onchip_mem	65408	128
onchip_mem_BEFORE_EXCEPTION	0x08010060 - 0x0801007F	onchip_mem	32	96
reset	0x08010040 - 0x0801005F	onchip_mem	32	64
onchip_mem_BEFORE_RESET	0x08010000 - 0x0801003F	onchip_mem	64	0
uniphy_ddr3	0x00000000 - 0x03FFFFFF	uniphy_ddr3	67108864	0

Grayed out entries are automatically created at generate time. They are not editable or persisted in the BSP settings file.

```
Mapped section ".text" to memory region "onchip_mem".
Mapped section ".rodata" to memory region "onchip_mem".
Mapped section ".rwdata" to memory region "onchip_mem".
Mapped section ".bss" to memory region "onchip_mem".
Mapped section ".heap" to memory region "onchip_mem".
Mapped section ".stack" to memory region "onchip_mem".
Finished loading BSP section mappings from settings file.
Loading drivers from ensemble report.
Finished loading drivers from ensemble report.
```

5. Click **Generate**.

Verifying 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**.

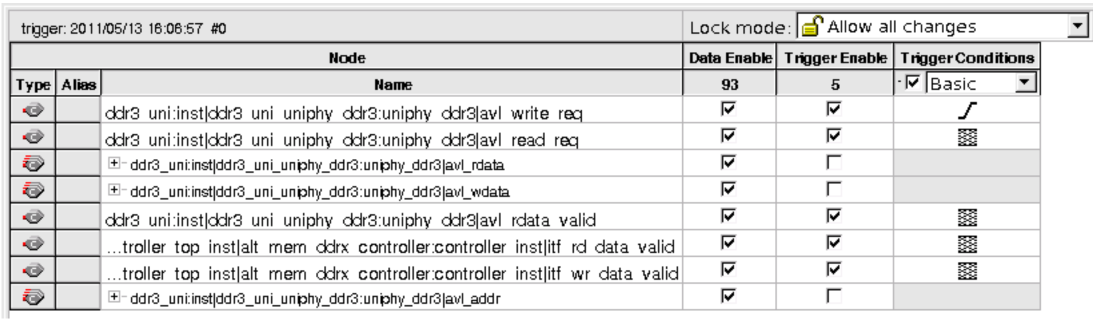
4. Select **ddr3_uni:inst|ddr3_uni_uniphy_ddr3:uniphy_ddr3|ddr3_uni_uniphy_ddr3_p0:p0|ddr3_uni_uniphy_ddr3_p0_controller_phy:controller_phy_inst|ddr3_uni_uniphy_ddr3_p0_memphy_top:memphy_top_inst|ddr3_uni_uniphy_ddr3_p0_memphy:umemphy|ddr3_uni_uniphy_ddr3_p0_nios_sequencer:usequencer|ddr3_uni_uniphy_ddr3_p0_qsys_sequencer:sequencer_inst|phy_clk** 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 `*uniphy_ddr3|avl*` in the **Named** box, for **Filter** select **SignalTap II: pre-synthesis** and click **List**.
9. In **Nodes Found**, select the following nodes and click **>** to add to **Selected Nodes**:
   - **avl_addr**
   - **avl_rdata**
   - **avl_rdata_valid** (alternative trigger to compare read/write data)
   - **avl_read_req**
   - **avl_wdata**
   - **avl_write_req** (trigger)
10. On the Edit menu, click **Add Nodes** again.
11. Search for specific nodes by typing `*controller_inst|itf*data_valid` in the **Named** box, for **Filter** select **SignalTap II: pre-synthesis** and click **List**.
12. In **Nodes Found**, select the following nodes and click **>** to add to **Selected Nodes**:
    - **itf_rd_data_valid**
    - **itf_wr_data_valid**
13. Click **OK**.

☞ 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.

14. To reduce the SignalTap II logic size, turn off **Trigger Enable** on the following buses:
    - **avl_addr**
    - **avl_rdata**
    - **avl_wdata**
15. Right-click **Trigger Conditions** for the `avl_write_req` signal and select **Rising Edge**.

    Figure 3–9 shows the completed SignalTap II Embedded Logic Analyzer.

Figure 3–9. SignalTap II Embedded Logic Analyzer



trigger: 2011/05/13 16:06:57 #0

Lock mode: Allow all changes

Node			Data Enable	Trigger Enable	Trigger Conditions
Type	Alias	Name	93	5	✓ Basic
		ddr3 uni:inst|ddr3 uni uniphy ddr3:uniphy ddr3|avl write req	✓	✓	╱
		ddr3 uni:inst|ddr3 uni uniphy ddr3:uniphy ddr3|avl read req	✓	✓	▨
		⊞ ddr3_uni:inst|ddr3_uni_uniphy_ddr3:uniphy_ddr3|avl_rdata	✓	☐	
		⊞ ddr3_uni:inst|ddr3_uni_uniphy_ddr3:uniphy_ddr3|avl_wdata	✓	☐	
		ddr3 uni:inst|ddr3 uni uniphy ddr3:uniphy ddr3|avl rdata valid	✓	✓	▨
		...troller top inst|alt mem ddrx controller:controller inst|itf rd data valid	✓	✓	▨
		...troller top inst|alt mem ddrx controller:controller inst|itf wr data valid	✓	✓	▨
		⊞ ddr3_uni:inst|ddr3_uni_uniphy_ddr3:uniphy_ddr3|avl_addr	✓	☐	

16. 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**.

Compiling the Project

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

Verifying 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**.

Connecting the Development Board

Connect the Stratix IV development board to your computer.

For more information on the Stratix IV development board, refer to the *Stratix IV GX FPGA Development Kit User Guide*.

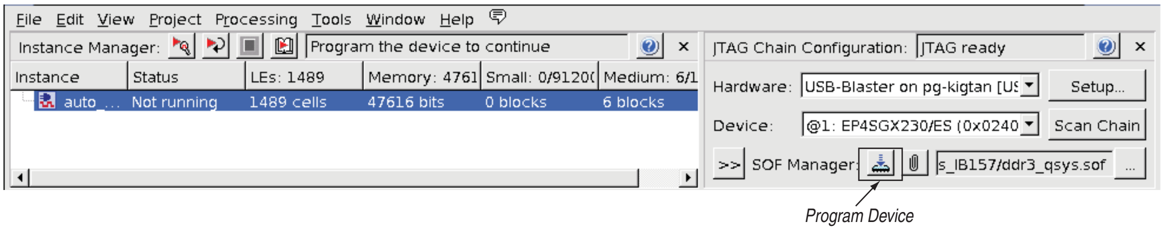
Downloading the Object File

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 3–10.
2. Select <*your project name*>**.sof**.
3. Click **Open**.
4. To download the file, click the **Program Device** button.

Figure 3–10. Install the SRAM Object File in the SignalTap II Dialog Box


File Edit View Project Processing Tools Window Help
Instance Manager:
Program the device to continue
JTAG Chain Configuration: JTAG ready
Hardware: USB-Blaster on pg-kigtan [US
Setup...
Device: @1: EP4SGX230/ES (0x0240
Scan Chain
>> SOF Manager:
s_IB157/ddr3_qsys.sof
...
Program Device

Instance	Status	LEs: 1489	Memory: 4761	Small: 0/9120(	Medium: 6/1
auto_...	Not running	1489 cells	47616 bits	0 blocks	6 blocks

Verifying Design with Nios II

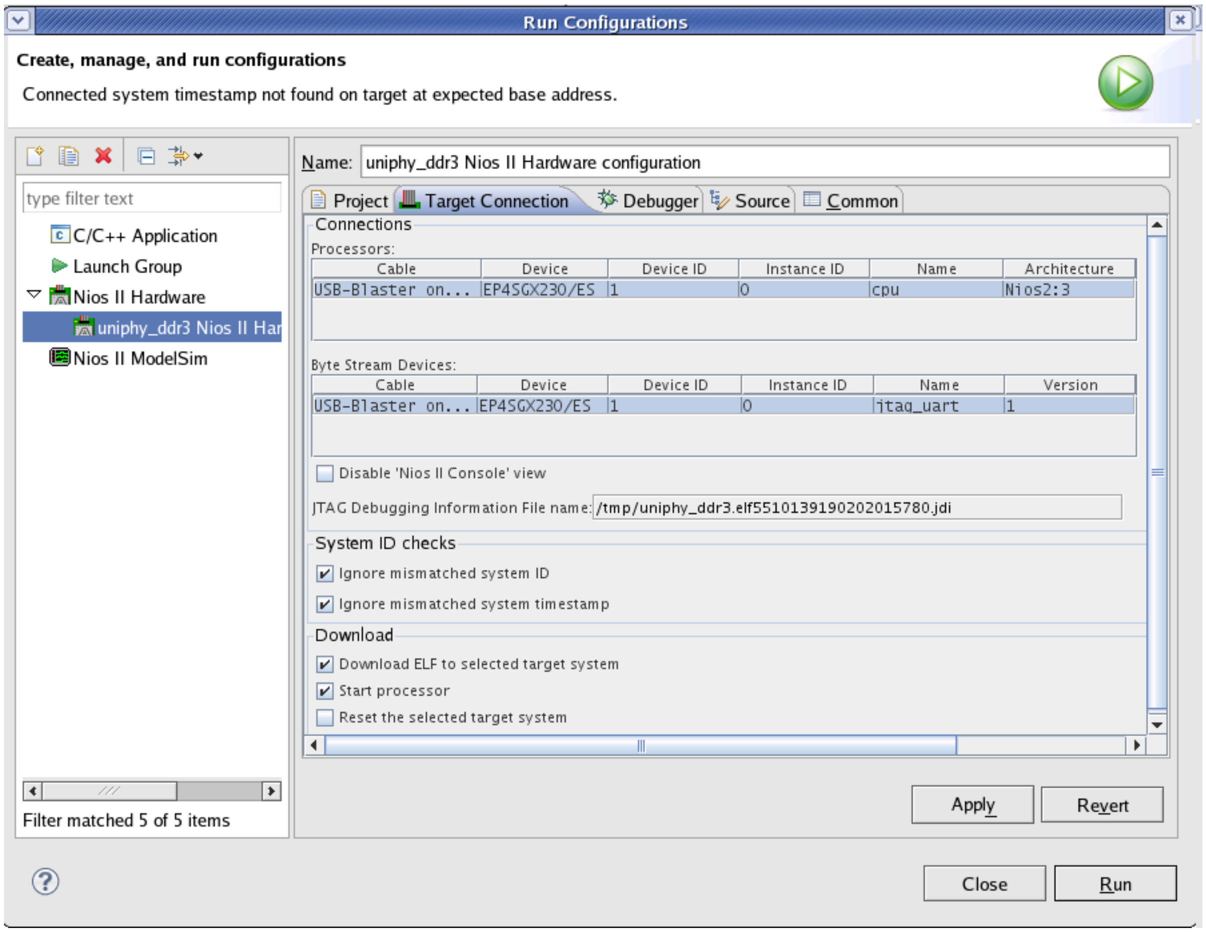
Right-click on **blank_project**, point to **Run As**, and click **Nios II Hardware** for the Nios II C/C++ Eclipse to compile the example test program.

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 Eclipse, on the Run menu click **RUN**. Click the **Target Connection** tab and correct the **JTAG cable connection**, refer to Figure 3–11.

2. Right click on <*target*>**_project**, point to **Run As**, and click **Nios II Hardware**.

Figure 3–11. JTAG Download Target Connection Settings



Run Configurations

Create, manage, and run configurations

Connected system timestamp not found on target at expected base address.

Name: uniphy_ddr3 Nios II Hardware configuration

Connections

Processors:

Cable	Device	Device ID	Instance ID	Name	Architecture
USB-Blaster on...	EP4SGX230/ES	1	0	cpu	Nios2:3

Byte Stream Devices:

Cable	Device	Device ID	Instance ID	Name	Version
USB-Blaster on...	EP4SGX230/ES	1	0	jtag_uart	1

☐ Disable 'Nios II Console' view

JTAG Debugging Information File name: /tmp/uniphy_ddr3.elf5510139190202015780.jdi

System ID checks

☑ Ignore mismatched system ID

☑ Ignore mismatched system timestamp

Download

☑ Download ELF to selected target system

☑ Start processor

☐ Reset the selected target system

Filter matched 5 of 5 items

Testing the System

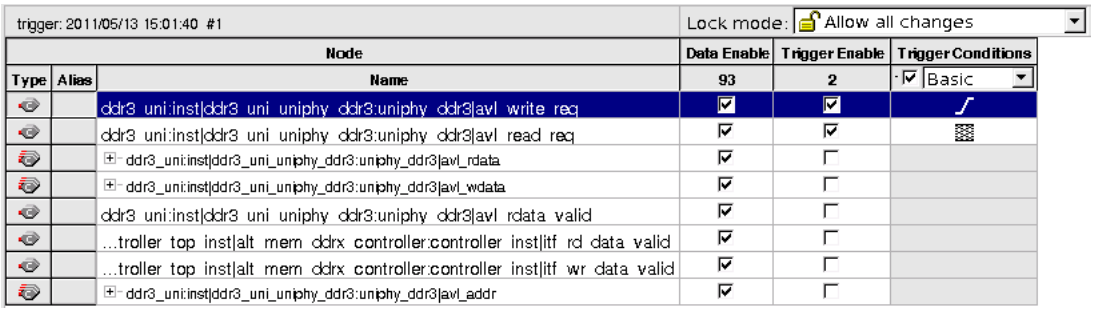
Perform the following tests to verify your system is operating correctly.

Setting SignalTap II Trigger to avl_write_req

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

1. In the **Setup** tab, select to trigger on a rising edge of `avl_write_req`, refer to Figure 3–12.

Figure 3–12. Set Trigger for avl_write_req Signal



trigger: 2011/05/13 15:01:40 #1 — Lock mode: Allow all changes

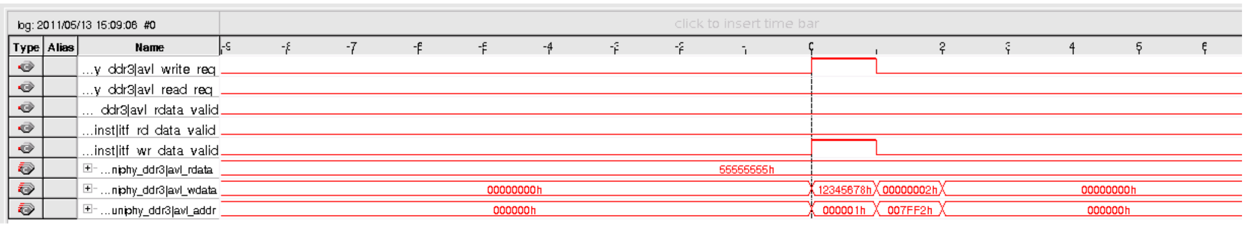
Type	Alias	Name	Data Enable	Trigger Enable	Trigger Conditions
			93	2	✓ Basic
		ddr3 uni:inst|ddr3 uni uniphy ddr3:uniphy ddr3|avl write req	✓	✓	⌐
		ddr3 uni:inst|ddr3 uni uniphy ddr3:uniphy ddr3|avl read req	✓	✓	▨
		⊞ ddr3_uni:inst|ddr3_uni_uniphy_ddr3:uniphy_ddr3|avl_rdata	✓	☐	
		⊞ ddr3_uni:inst|ddr3_uni_uniphy_ddr3:uniphy_ddr3|avl_wdata	✓	☐	
		ddr3 uni:inst|ddr3 uni uniphy ddr3:uniphy ddr3|avl rdata valid	✓	☐	
		...troller top inst|alt mem ddrx controller:controller inst|itf rd data valid	✓	☐	
		...troller top inst|alt mem ddrx controller:controller inst|itf wr data valid	✓	☐	
		⊞ ddr3_uni:inst|ddr3_uni_uniphy_ddr3:uniphy_ddr3|avl_addr	✓	☐	

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

```
B123456780000001
```

4. Return to the SignalTap II Embedded Logic Analyzer window and check that at time 0 the following occur (refer to Figure 3–13):
   - `avl_write_req` signal goes high for a single cycle
   - `avl_wdata` shows `12345678h`
   - `avl_address` shows `000001h`

Figure 3–13. Waveform for Write Request

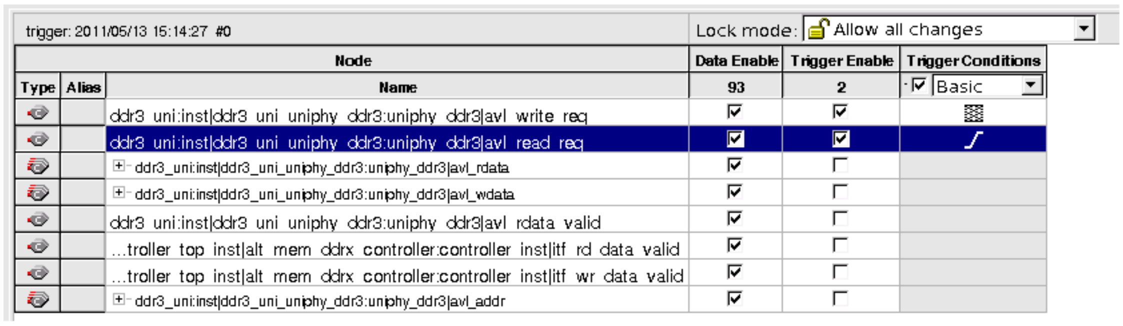


Setting SignalTap II Trigger to avl_read_req

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 `avl_read_req`, refer to Figure 3–14.

Figure 3–14. Set Trigger for avl_read_req Signal



trigger: 2011/05/13 15:14:27 #0

Lock mode: Allow all changes

Type	Alias	Node Name	Data Enable	Trigger Enable	Trigger Conditions
			93	2	✓ Basic
		ddr3 uni:inst|ddr3 uni uniphy ddr3:uniphy ddr3|avl write req	✓	✓	▨
		ddr3 uni:inst|ddr3 uni uniphy ddr3:uniphy ddr3|avl read req	✓	✓	⟋
		ddr3_uni:inst|ddr3_uni_uniphy_ddr3:uniphy_ddr3|avl_rdata	✓	☐	
		ddr3_uni:inst|ddr3_uni_uniphy_ddr3:uniphy_ddr3|avl_wdata	✓	☐	
		ddr3 uni:inst|ddr3 uni uniphy ddr3:uniphy ddr3|avl rdata valid	✓	☐	
		...troller top inst|alt mem ddrx controller:controller inst|itf rd data valid	✓	☐	
		...troller top inst|alt mem ddrx controller:controller inst|itf wr data valid	✓	☐	
		ddr3_uni:inst|ddr3_uni_uniphy_ddr3:uniphy_ddr3|avl_addr	✓	☐	

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

   `C00000001`

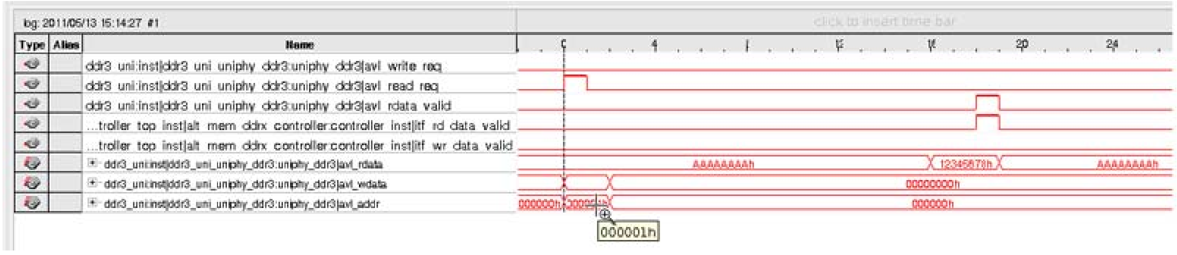
4. Return to the SignalTap II Embedded Logic Analyzer window and check that at time 0 the following occur (refer to Figure 3–8):

- `avl_read_req` signal goes high
- `avl_address` shows `000001h`

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

- `avl_rdata_valid` goes high for one cycle
- `avl_rdata` shows `12345678h`

Figure 3–15. Waveform for Read Request



log: 2011/05/13 15:14:27 #1

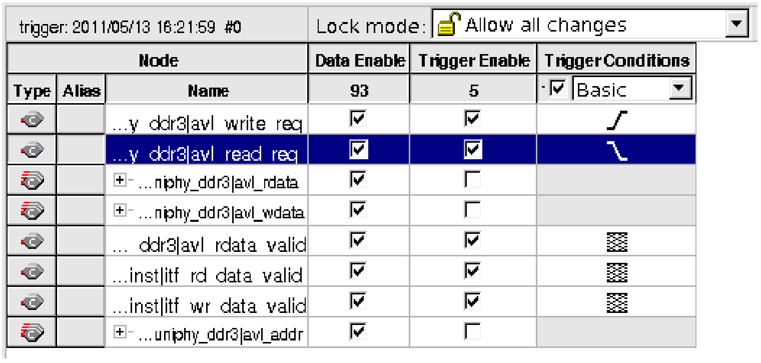
Type	Alias	Name
		ddr3 uni:inst|ddr3 uni uniphy ddr3:uniphy ddr3|avl write req
		ddr3 uni:inst|ddr3 uni uniphy ddr3:uniphy ddr3|avl read req
		ddr3 uni:inst|ddr3 uni uniphy ddr3:uniphy ddr3|avl rdata valid
		...troller top inst|alt mem ddrx controller:controller inst|itf rd data valid
		...troller top inst|alt mem ddrx controller:controller inst|itf wr data valid
		ddr3_uni:inst|ddr3_uni_uniphy_ddr3:uniphy_ddr3|avl_rdata
		ddr3_uni:inst|ddr3_uni_uniphy_ddr3:uniphy_ddr3|avl_wdata
		ddr3_uni:inst|ddr3_uni_uniphy_ddr3:uniphy_ddr3|avl_addr

Testing DMA Read and Write Operation

Use the SignalTap II Embedded Logic Analyzer to capture DMA read and write activity. To show activity, follow these steps:

1. In the **Setup** tab, select to trigger on a rising edge of `avl_write_req` and on a falling edge of `avl_read_req`, refer to Figure 3–16.

Figure 3–16. Set Trigger for avl_write_req and avl_read_req Signal



trigger: 2011/05/13 16:21:59 #0 | Lock mode: Allow all changes

Type	Alias	Name	Data Enable 93	Trigger Enable 5	Trigger Conditions ✓ Basic
		...y ddr3|avl write req	✓	✓	rising edge
		...y ddr3|avl read req	✓	✓	falling edge
		...niphy_ddr3|avl_rdata	✓		
		...niphy_ddr3|avl_wdata	✓		
		... ddr3|avl rdata valid	✓	✓	▨
		...inst|itf rd data valid	✓	✓	▨
		...inst|itf wr data valid	✓	✓	▨
		...uniphy_ddr3|avl_addr	✓		

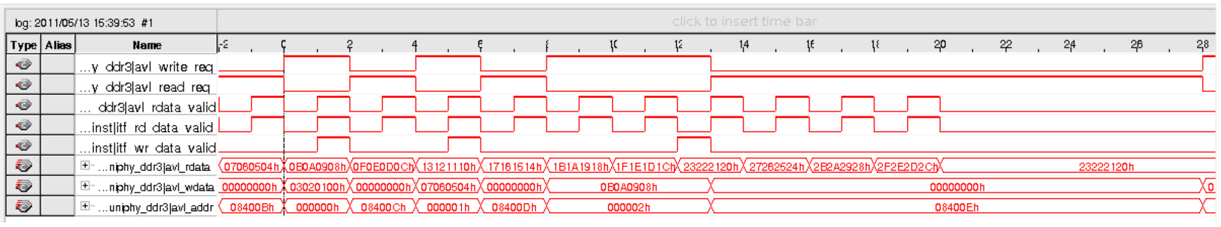
2. Click **Run Once Analysis** to capture the write request.
3. Type the following command in the Nios II command console:

```
D802100000000000
```

4. Return to the SignalTap II Embedded Logic Analyzer window and check that at time 0, the following occur (refer to Figure 3–8):
   - `avl_write_req` and `avl_read_req` signal goes high for multiple cycles
   - `avl_wdata` shows `03020100h` followed by `07060504h` and so on
   - `avl_address` shows `000000h` followed by `000001h` and so on

The current `avl_address` increases by 1 compared to the `avl_address` of previous read or write request.
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 3–17. Waveform for Continuous Read and Write Activity


log: 2011/05/13 15:39:53 #1
click to insert time bar
Type
Alias
Name
-2 0 2 4 6 8 10 12 14 16 18 20 22 24 26 28
...y ddr3|avl write req
...y ddr3|avl read req
... ddr3|avl rdata valid
...inst|itf rd data valid
...inst|itf wr data valid
...niphy_ddr3|avl_rdata
...niphy_ddr3|avl_wdata
...uniphy_ddr3|avl_addr
07060504h 0B0A0908h 0F0E0D0Ch 13121110h 17161514h 1B1A1918h 1F1E1D1Ch 23222120h 27262524h 2B2A2928h 2F2E2D2Ch 23222120h
00000000h 03020100h 00000000h 07060504h 00000000h 0B0A0908h 00000000h 0
08400Bh 000000h 08400Ch 000001h 08400Dh 000002h 08400Eh

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>=(UNIPHY_DDR3
_SPAN/4)))
  {
    printf(" \n Invalid Offset \n");
    printf( "\n You have entered wrong offset address : \n");
    return;
  }
  IOWR(UNIPHY_DDR3_BASE,DDR_write_OFFSET_ADDRESS,write_data);
  if (IORD(UNIPHY_DDR3_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>=(UNIPHY_DDR3_SPAN/4)))
  {
    printf(" \n Invalid Offset \n");
  }
  else
  {
    read_data=IORD(UNIPHY_DDR3_BASE,DDR_read_OFFSET_ADDRESS);
    printf("Read %08x from address %08x
\n",read_data,(UNIPHY_DDR3_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 ;

}
```

Additional Information

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
June 2011	3.0	■ Updated the *DDR3 SDRAM Controller with UniPHY Using Qsys* chapter. ■ Removed all memory-protocol specific tutorials and added a new generic tutorial applicable for all memory protocols. ■ Updated the *Implementing Multiple Memory Interfaces Using UniPHY* chapter which now uses two components of DDR3 SDRAM with UniPHY.
December 2010	2.1	■ Added a new chapter: *DDR3 SDRAM Controller with UniPHY using Qsys*. ■ Updated *DDR2 and DDR3 SDRAM Controller with UniPHY in Stratix III and Stratix IV Devices* chapter. ■ Updated *Implementing Multiple Memory Interfaces Using UniPHY* chapter with OCT information.
July 2010	2.0	Added two new chapters: ■ *DDR3 SDRAM Controller with UniPHY in Stratix IV Devices*. ■ *Implementing Multiple Memory Interfaces Using UniPHY*.
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*.
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.
⚠ CAUTION	A caution calls attention to a condition or possible situation that can damage or destroy the product or your work.
⚠ WARNING	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.