External Memory Interface Handbook Volume 5
Section I. ALTMEMPHY Design Tutorials
Chapter 1. Using DDR, DDR2, and DDR3 SDRAM Devices in Arria II GX Devices

System Requirements ................................................................. 1–1
Create a Quartus II Project ...................................................... 1–1
Instantiate and Parameterize a Controller .................................. 1–1
   Instantiate a Controller .................................................... 1–2
   Parameterize DDR2 SDRAM .............................................. 1–2
   Parameterize DDR3 SDRAM .............................................. 1–4
Add Constraints ...................................................................... 1–6
   Set the Top-Level Entity ................................................... 1–6
   Set the Optimization Technique ...................................... 1–7
   Set the Fitter Effort ......................................................... 1–7
   Add Timing Constraints .................................................... 1–7
   Add Pin and DQ Group Assignments ................................. 1–7
   Enter Pin Location Assignments ....................................... 1–8
   Assign I/O Standards ........................................................ 1–10
   Assign Virtual Pins .......................................................... 1–10
   Enter Board Trace Delay Models ....................................... 1–10
Perform RTL or Functional Simulation (Optional) ....................... 1–12
Compile Design and Verify Timing .......................................... 1–13
Verify Design on a Board ....................................................... 1–15
   Compile the Project ......................................................... 1–17
   Verify Timing ................................................................. 1–17
Download the Object File ..................................................... 1–18
Test the Design Example in Hardware .................................... 1–18

Chapter 2. Using DDR and DDR2 SDRAM Devices in Cyclone III and Cyclone IV Devices

Select the Device ................................................................. 2–1
Instantiate PHY and Controller in a Quartus II Project .............. 2–2
Perform RTL or Functional Simulation (Optional) ....................... 2–9
Add Constraints ...................................................................... 2–10
Compile Design and Verify Timing Closure ............................. 2–12
Adjust Constraints ............................................................ 2–17
Determine Board Design Constraints ................................... 2–20

Chapter 3. Using High-Performance DDR, DDR2, and DDR3 SDRAM with SOPC Builder

SOPC Builder System Considerations ....................................... 3–1
   High-Performance Controller (HPC) or High-Performance Controller II (HPC II) .......... 3–2
   Full- or Half-Rate SDRAM High-Performance Controller ............................................. 3–2
      Full-Rate Versus Half-Rate Command Operation ..................................................... 3–3
      Time-Specified Memory Parameters ................................................................. 3–3
   Clock Selection and Clock Crossing Bridges ....................................................... 3–4
   Burst Reads and Writes ........................................................... 3–6
   Multiple Masters .................................................................. 3–8
   Direct Memory Access (DMA) Controller ......................................................... 3–8
   Read and Write Addressing and Latency ......................................................... 3–9
   DDR2 SDRAM Controller with ALTMEMPHY IP with SOPC Builder Walkthrough ........ 3–10
System Requirement .......................................................... 3–10
Create Your Example Project .................................................. 3–11
1. Using DDR, DDR2, and DDR3 SDRAM Devices in Arria II GX Devices

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

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

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

System Requirements

This tutorial requires the following hardware and software:

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

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

Create a Quartus II Project

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

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

Instantiate and Parameterize a Controller

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

To instantiate a controller, perform the following steps:

- **DDR2 SDRAM controller**
  
  a. Copy the memory parameters file, `ArriaIIGX_DDR2_Kit(MT8HTF12864HDY-800G1).xml`, to your `<installation directory>\ip\ddr2_high_perf\lib` directory.

  b. Launch the MegaWizard™ Plug-in Manager.

  c. In the MegaWizard Plug-In Manager, expand **External Memory** in the **Interfaces** folder and select the **DDR2 SDRAM High Performance Controller**.

  d. Enter `ddr2_sodimm` for the name of the DDR2 SDRAM high-performance controller.

- **DDR3 SDRAM controller**

  a. Copy the memory parameters file, `ArriaIIGX_DDR3_Kit(MT41J64M16LA-15E).xml`, to your `<installation directory>\ip\ddr3_high_perf\lib` directory.

  b. Launch the MegaWizard Plug-in Manager.

  c. In the MegaWizard Plug-In Manager, expand **External Memory** in the **Interfaces** folder and select the **DDR3 SDRAM High Performance Controller**.

  d. Enter `ddr3` for the name of the DDR3 SDRAM high-performance controller.

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

Parameterize DDR2 SDRAM

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

1. In the **Memory Setting** tab, set **Speed grade** to 5.

2. For **PLL reference clock frequency**, enter 100 MHz. The input clock source, `clock_source`, supplies the **PLL reference clock frequency**.

3. For **Memory clock frequency**, enter 267 MHz. This value is the maximum frequency supported for the DDR2 SDRAM interface on Arria II GX devices.
4. For Memory Presets, select Micron ArriaIIGX_DDR2_Kit (MT8HTF12864HDY-800G1), which gives a 64-bit wide 1-GB 400-MHz DDR2 SODIMM. To create or modify a memory preset, click Modify parameters. In the Preset Editor dialog box, modify the memory presets. Refer to Figure 1–1 on page 1–3.

5. Turn on the Enable Memory Chip Calibration in Timing Analysis option in the Advanced page. This option is required for Arria II GX devices, which need post-processing script to remove timing model pessimism.

6. In the PHY Settings tab, under Advanced PHY Settings, turn on the Use differential DQS option to enhance signal to noise ratio. Turn on this option if noise margin is a concern.
7. By default, **Clock Phase** is set to 90 for **Address/Command Clock Settings**. The **Auto-Calibration Simulation Options** parameter is for RTL simulation only and is not applicable for gate-level simulation. ALTMEMPHY does not support gate-level timing simulation. In the **Board Settings** tab, enter the following values for the parameters under **Slew Rates** and **Board Skews**:

- CK/CK# slew rate (Differential) = 2.475 V/ns
- Addr/Command slew rate = 0.859 V/ns
- DQS/DQS# slew rate (Differential) = 1.386 V/ns
- DQ slew rate = 2.665 V/ns

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

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

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

8. In the **Controller Settings** tab, for **Controller Architecture**, select **High Performance Controller II** for higher efficiency and advanced features.

9. Under **Efficiency**, select the specified values for the following options:
   a. For **Command Queue Look-Ahead Depth**, select 6.
   b. For **Local-to-Memory Address Mapping**, select **CHIP-ROW-BANK-COL**.
   c. For **Local Maximum Burst Count**, select 8.

10. Click **Next**.

11. Turn on the **Generate simulation model** option to generate the simulation model for the RTL simulation.

12. Click **Finish** to generate your MegaCore function variation. The MegaWizard Plug-In Manager generates all the files necessary for your DDR2 SDRAM controller and a top-level design, which you use to test or verify the board operation.

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

**Parameterize DDR3 SDRAM**

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

1. In the **Memory Setting** tab, set **Speed grade** to 4.
2. For **PLL reference clock frequency**, enter 100 MHz, to match the on-board oscillator.
3. For Memory clock frequency, enter 300 MHz. This is the maximum frequency supported for DDR3 SDRAM interface on Arria II GX devices.

4. For Memory Presets, select Micron ArriaIIGX_DDR3_Kit(MT41J64M16LA-15E), which gives a 16-bit wide 1-GB 667-MHz DDR3 component. To create or modify a memory preset, click Modify parameters. In the Preset Editor dialog box, modify the memory presets (Figure 1–2 on page 1–5).

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

5. In Advanced page, turn on the Enable Memory Chip Calibration in Timing Analysis option. This option is required for Arria II GX devices, which need post-processing script to remove timing model pessimism.

6. Under Address/Command Clock Settings, use the default value of Clock Phase, which is 90.

7. In the Board Settings tab, set the following Slew Rates and Board Skews parameters to the specified values:
   - CK/CK# slew rate (Differential) = 4 V/ns
   - Addr/Command slew rate = 2 V/ns
   - DQS/DQS# slew rate (Differential) = 1.8 V/ns
   - DQ slew rate = 1.5 V/ns
These slew rates are obtained from simulation using the default I/O standard and drive options.

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

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

8. In the Controller Settings table, for Controller Architecture, select High Performance Controller II for higher efficiency and advanced features. Under Efficiency, select the specified values for the following options:
   a. For Command Queue Look-Ahead Depth, select 6.
   b. For Local-to-Memory Address Mapping, select CHIP-ROW-BANK-COL.
   c. For Local Maximum Burst Count, select 4.

9. Click Next.

10. Turn on the Generate simulation model option to generate the simulation model for the RTL simulation.

11. Click Finish to generate your MegaCore function variation. The MegaWizard Plug-In Manager generates all the files necessary for your DDR3 SDRAM controller and generates an example top-level design, which you use to test or verify board operation.

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

Add Constraints

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

Set the Top-Level Entity

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

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

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

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

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

Set the Fitter Effort

To set the fitter effort, perform the following steps:

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

Add Timing Constraints

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

To add the timing constraints, perform the following steps:

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

Add Pin and DQ Group Assignments

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

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

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

1) On the Tools menu, click **Tcl scripts**.
2) Under **Libraries**, select `<variation_name>_pin_assignments.tcl`
3) Click **Run**.
Enter Pin Location Assignments

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

1. Run analysis and synthesis. On the Processing menu, point to Start and click Start Analysis & Synthesis.

2. Assign all of your pins, so that the Quartus II software fits your design correctly and gives correct timing analysis. Manually assign the pin locations by using the Pin Planner or Assignment Editor. To assign the pin locations for the Arria II GX Development Kit, run the Altera-provided ArriaIIGX_DDR2_PinLocations.tcl script for the DDR2 SDRAM interface design or the ArriaIIGX_DDR3_PinLocations.tcl script for the DDR3 SDRAM interface design.

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

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

1. On the Assignments menu, click Pin Planner.

2. Assign DQ and DQS pins.

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

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

   b. Select the DQ mode to match the DQ group width (number of DQ pins/number of DQS pins) of your memory device. DQ mode is not related to the memory interface width.
DQ group order and DQ pin order within each group is not important. However, you must place the DQ pins in the same group as their respective strobe pin.

3. Place the DM pins within their respective DQ group.
4. Place address and control command pins on any spare I/O pins, ideally in the same bank or side of the device as the mem_clk pins.
5. Ensure that you place mem_clk pins on differential I/O pairs for the CK/CK# pin pair. To identify the differential I/O pairs, right-click in the Pin Planner and select Show Differential Pin Pair Connections. The pin pairs show a red line between each pin pair.
6. Ensure that the mem_clk pins use any regular adjacent I/O pins; ideally the differential I/O pairs for the CK/CK# pin pair. To identify the differential I/O pairs, right-click in Pin Planner and select Show Differential Pin Pair Connections. Pin pairs show a red line between each pin pair.
7. Place the clock_source pin on a dedicated PLL clock input pin with a direct connection to the SDRAM controller PLL and DLL pair—usually on the same side of the device as your memory interface. This recommendation reduces PLL jitter, saves a global clock resource, and eases timing and fitter effort.
8. Place the global_reset_n pin (as any high fan-out signal) on a dedicated clock pin.
9. Ensure that you place the RUP and RDN pins, termination_blk0__rdn_pad and termination_blk0__rup_pad, at locations within the same VCCIO voltage bank.
Assign I/O Standards

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

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

Assign Virtual Pins

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

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

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

Enter Board Trace Delay Models

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

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

To enter board trace information, perform the following steps:
1. In the Pin Planner, select the pin or group of pins that you want to enter the information for.
2. Right-click and select Board Trace Model, as shown in Figure 1–4.

**Figure 1–4. Board Trace Model**

![Board Trace Model Diagram]

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

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

<table>
<thead>
<tr>
<th>Net</th>
<th>Near (FPGA End of Line)</th>
<th>Far (Memory End of Line)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Length (Inch)</td>
<td>L_per_length (nH/In)</td>
</tr>
<tr>
<td>DDR2 SODIMM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Addr (1)</td>
<td>1.743</td>
<td>8</td>
</tr>
<tr>
<td>CLK0</td>
<td>2.614</td>
<td>8.66</td>
</tr>
<tr>
<td>CLK1</td>
<td>1.616</td>
<td>8.66</td>
</tr>
<tr>
<td>CKE/CS#</td>
<td>1.599</td>
<td>8</td>
</tr>
<tr>
<td>DQS0</td>
<td>1.866</td>
<td>8</td>
</tr>
<tr>
<td>DQS1</td>
<td>1.757</td>
<td>8</td>
</tr>
<tr>
<td>DQS2</td>
<td>1.437</td>
<td>8</td>
</tr>
<tr>
<td>DQS3</td>
<td>1.457</td>
<td>8</td>
</tr>
<tr>
<td>DQS4</td>
<td>1.450</td>
<td>8</td>
</tr>
</tbody>
</table>
Altera recommends that you use the Board Trace Model assignment for all memory interface signals. To apply the board trace model assignments for the Arria II GX FPGA development kit, run the Altera-provided ArriaIIGX_DDR2_BTModels.tcl script for the DDR2 SDRAM interface design, or the ArriaIIGX_DDR3_BTModels.tcl script for the DDR3 SDRAM interface design, or manually assign virtual pin assignments using the Quartus II Pin Planner.

### Perform RTL or Functional Simulation (Optional)

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

The Verilog HDL or VHDL simulation model of the PHY, `<variation_name>_{alt_mem_phy_sequencer_wrapper}.vo`/`.vho` file, is located in your project directory.

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

---

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

<table>
<thead>
<tr>
<th>Net</th>
<th>Near (FPGA End of Line)</th>
<th>Far (Memory End of Line)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Length (Inch)</td>
<td>L_per_length (nH/In)</td>
</tr>
<tr>
<td>DQS5</td>
<td>1.548</td>
<td>8</td>
</tr>
<tr>
<td>DQS6</td>
<td>1.677</td>
<td>8</td>
</tr>
<tr>
<td>DQS7</td>
<td>1.831</td>
<td>8</td>
</tr>
<tr>
<td>DDR3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Addr</td>
<td>1.879</td>
<td>8.66</td>
</tr>
<tr>
<td>CLK</td>
<td>1.359</td>
<td>8.66</td>
</tr>
<tr>
<td>CKE/CS#</td>
<td>1.775</td>
<td>8.66</td>
</tr>
<tr>
<td>DQS0</td>
<td>1.278</td>
<td>8.66</td>
</tr>
<tr>
<td>DQS1</td>
<td>1.249</td>
<td>8.66</td>
</tr>
</tbody>
</table>

**Note to Table 1–1:**

1. Addr = Addr, we#, ras#, odt, and cas.
2. The near (FPGA end of line) board trace model summary is not applicable for memory components configuration.
1. Set the absolute path to your third-party simulator executable file, by performing the following steps:
   a. On the Assignments menu, point to EDA Tool Settings.
   b. In the Category list, expand EDA Tool Settings and click Simulation.
   c. Under Tool name, select ModelSim.
   d. Under NativeLink settings, select Compile test bench and click Test Benches.
   e. Click New.
   f. Type the name of your testbench top-level module and simulation period.
   g. Click OK.
2. To elaborate your design, on the Processing menu, point to Start and click Start Analysis & Elaboration.
3. On the Tools menu, point to the Run EDA Simulation Tool and click EDA RTL Simulation. This step creates the vsimulation directory in your project directory and a script that compiles all necessary files and runs the simulation.

**Compile Design and Verify Timing**

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

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

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

Figure 1–5 shows the timing margin report for a DDR2 SDRAM interface design in the message window in the Quartus II software.

**Figure 1–5. Timing Margin Report in the Quartus II Software**
You can also obtain the timing report by running the report timing script (ddr2_sodimm_phy_report_timing.tcl or ddr3_phy_report_timing.tcl) in the TimeQuest timing analyzer. To obtain the timing report in the TimeQuest Timing Analyzer window, perform the following steps:

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

Figure 1–6 shows the timing margin report in the TimeQuest Timing Analyzer window after the report timing script runs. The results are the same as the Quartus II software results.

![Figure 1–6. Timing Margin Report in the TimeQuest Timing Analyzer](image)

You must verify the timing on every corner of the timing model. You must run the timing report script with all available timing models—slow 0°C, slow 85°C, and fast 0°C—to ensure positive margins across process, voltage, and temperature. To analyze timing on a different corner, first double-click **Set Operating Conditions** in the left pane. Select a new timing corner, then go to the Script menu and rerun the `<variation name>_report_timing.tcl` script.
For designs that target Arria II GX devices, you must run a post-processing script to remove timing model pessimism on the write and read capture path margins. For example, refer to Figure 1–7 and Figure 1–8.

**Figure 1–7. Write Margin Summary**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Setup Slack</th>
<th>Hold Slack</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Standard Write</td>
<td>0.246</td>
<td>0.246</td>
</tr>
<tr>
<td>2. Spatial correlation pessimism removal</td>
<td>0.015</td>
<td>0.015</td>
</tr>
<tr>
<td>3. Write</td>
<td>0.264</td>
<td>0.261</td>
</tr>
</tbody>
</table>

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

The final write and read capture margins are summarized in the Report DDR margin summary. Figure 1–9 shows an example of a Report DDR margin summary.

**Figure 1–8. Read Capture Path Margin Summary**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Setup Slack</th>
<th>Hold Slack</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Standard Read Capture</td>
<td>0.172</td>
<td>0.131</td>
</tr>
<tr>
<td>2. Spatial correlation pessimism removal</td>
<td>0.071</td>
<td>0.070</td>
</tr>
<tr>
<td>3. Read Capture</td>
<td>0.243</td>
<td>0.251</td>
</tr>
</tbody>
</table>

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

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

**Verify Design on a Board**

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

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

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

- local_address
- local_rdata
- local_rdata_valid
- local_read_req
- local_ready
- local_wdata
- local_wdata_req
- local_write_req
- pnf
- pnf_per_byte
- test_complete (trigger)

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

12. Click **OK**.

13. To reduce the SignalTap II logic size, turn off **Trigger Enable** on the following bus signals:

- local_address
- local_rdata
- local_wdata
- pnf_per_byte

14. Right-click **Trigger Conditions** for the **test_complete** signal and select **Rising Edge**.

15. On the File menu, click **Save**.

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

**Compile the Project**

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

**Verify Timing**

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

To perform timing analysis, run the `<variation name>_phy_report_timing.tcl` script.

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

**Download the Object File**

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

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

**Test the Design Example in Hardware**

When the design example including SignalTap II Embedded Logic Analyzer successfully downloads to your development board, click **Run Analysis** to run once, or click **Autorun Analysis** to run continuously. **Figure 1–10** shows the design analysis.

---

**Figure 1–10. SignalTap II Example DDR2 SDRAM Design Analysis**

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

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

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

Select the Device

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

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

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

Create a project in the Quartus II software targeting the EP3C120F780C7 device. Figure 2–1 shows how to target this device in the New Project Wizard.

Figure 2–1. Creating a Quartus II Project Targeting the EP3C120F780C7 Device

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

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

The subsequent files mentioned in this document that are generated by the MegaWizard Plug-In Manager and also other project files will have DDR2 as the prefix for the file names.
The rest of this subsection specifies the memory settings. Select the Cyclone III device with –7 speed grade. Then set the PLL reference clock frequency to 125 MHz and the memory clock frequency to 166.667 MHz. The 125 MHz PLL reference clock is provided by the on-board oscillator. Select the Micron MT47H32M16CC-3 333-MHz device. This 512-MB DDR2 device has a 16-bit data width. Figure 2–2 shows the memory settings panel after you make the selections.

**Figure 2–2. Configuring the DDR2 SDRAM Controller with ALTMEMPHY IP for the DDR2 Memory Interface**
The DDR2 SDRAM controller MegaWizard interface uses the default parameters for the memory when you instantiate the controller. The default parameters are based on the memory datasheets. You can customize your memory presets by modifying the parameters. Click the Modify parameters button to modify the memory attributes, memory initialization options, or memory timing parameters in the Preset Editor dialog box. In this design example, modify the memory interface DQ width to 32 to match the targeted memory width on the development kit. Figure 2–3 shows the Preset Editor dialog box after you perform the customization.

Figure 2–3. Editing the Memory Presets to Create a Custom Memory

Figure 2–4 shows the PHY Settings tab. In the PHY Settings tab, under Board Timing Parameters, you must parameterize the Board Skew settings. The specified skew is across all memory interface signal types including data, strobe, clock, address and command, and is used to generate the PHY timing constraints for all paths. The default value is set to 20 ps. You must update this number based on your board specification, because this number is used to calculate the overall system timing margin.

The DDR2 SDRAM device uses the CK and CK# signals to clock the command and address signals into the memory. The controller names the CK and CK# signals mem_clk and mem_clk_n, respectively. The skew between the CK or CK# and the DDR2 SDRAM-generated DQS signal is the value of tDQSCk in the DDR2 SDRAM data sheet.
The DDR2 SDRAM has a write requirement (tDQSS) that the positive edge of the DQS signal on writes be within ±25% (±90°) of the positive edge of the DDR2 SDRAM clock input. tDQSS is defined as the time between the DQS latching edge and its associated clock edge. The controller generates the mem_clk and mem_clk_n signals using the DDR registers in the input/output element (IOE) to match the DQS signal and reduce any variations across process, voltage, and temperature. The positive edge of the DDR2 SDRAM clock, mem_clk, is aligned with the DQS write to satisfy tDQSS.

The settings in the Auto-Calibration Simulation Options section are for RTL simulation only and are not applicable for gate-level simulation.
Choose your local interface setting in the **Controller Settings** panel. Turn on the **Enable error detection and correction logic** option if you want to use error code correction (ECC). If you have your own refresh requirements, turn on **Enable user-controlled refresh**. Next, select the **Local Interface Protocol** for the memory interface. The default interface is the **Avalon Memory-Mapped Interface**. This interface allows you to easily connect to other Avalon-MM peripherals.
Figure 2–6 on page 2–7 shows the EDA panel. The MegaWizard can generate the simulation model for simulating the memory controller in either Verilog HDL or VHDL.

Figure 2–6. DDR2 SDRAM Controller with ALTMEMPHY IP EDA

In the Timing and Resource Estimation, you can choose to generate a netlist if you are synthesizing your design with a third-party EDA synthesis tool.
Figure 2–7 shows the Summary panel.

**Figure 2–7. Summary**

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

Click **Finish** to generate the controller.
Perform RTL or Functional Simulation (Optional)

After instantiating the DDR2 SDRAM Controller with ALTMEMPHY IP, the MegaWizard Plug-In Manager generates a design example and driver for testing the memory interface. Figure 2–8 shows a system-level diagram of the design example that the DDR2 SDRAM Controller with ALTMEMPHY IP MegaWizard creates for you.

Figure 2–8. DDR2 SDRAM Controller with ALTMEMPHY IP System-Level Diagram

Note to Figure 2-8:
(1) The ALTMEMPHY megafunction automatically generates the PLL. The PLL is part of the ALTMEMPHY megafunction.

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

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

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

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

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

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

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

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

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

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

**Figure 2–10. Specifying the Timing Constraint SDC File for the Design Example**
The DDR2_pin_assignments.tcl file specifies the I/O standard assignment for the memory interface pins. To run the Tcl file, on the Tools menu, click Tcl Scripts. Select the Tcl file and click Run (Figure 2–11 on page 2–12). Running the script adds the information into your Quartus II project. Alternatively, you can also use the Tcl Console to run the Tcl file.

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

**Compile Design and Verify Timing Closure**

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

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

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

1. On the Assignments menu, click Settings. The Settings dialog box appears.
2. In the Category list, click Analysis & Synthesis Settings.
3. Define the **Optimization Technique** setting. The default setting is **Balanced** (Figure 2–12 on page 2–13).

**Figure 2–12. Default Setting for Optimization Technique**
4. In the Category list, click Fitter Settings, and set the Fitter effort. The default setting is Auto Fit (Figure 2–13 on page 2–14).

To compile your design, perform the following steps:

1. On the Processing menu, click Start Compilation.

2. After compilation is successful, on the Tools menu, select TimeQuest Timing Analyzer.

3. Generate the timing margin report for your memory interface design by executing the Report DDR function from the Tasks pane of the TimeQuest Timing Analyzer window (Figure 2–14).

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

Figure 2–14. TimeQuest Timing Analyzer Window

For more information about the TimeQuest timing analyzer, refer to the *Quartus II TimeQuest Timing Analyzer* chapter in volume 3 of the *Quartus II Handbook*.
Figure 2–15 shows the output of the Report DDR task. The Report pane contains a new folder titled DDR with detailed timing information about the most critical paths, and a timing margin summary similar to the summary on the TimeQuest Console.

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

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

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

For more information about the timing analysis, refer to the Timing Analysis section in volume 4 of the External Memory Interface Handbook.
the hold margin on the address and command datapath. To determine the clock that is clocking the address and command registers, in TimeQuest timing analyzer, on the Report panel, click on the address/command report, and select the path for the setup or hold time for the address and command datapath as Path Summary (Figure 2–16) or in Waveform View (Figure 2–17).

Figure 2–16. Path Summary for the Address and Command Datapath
The report indicates that clk2 of the PLL is clocking the address and command registers. Edit PLL megafunction to change the phase setting of clk2. For this design, the initial phase setting of clk2 is –90° with reference to the system clock. By adjusting clk2 to later than –90° (specifying that the address and command launch later), the setup margin is decreased and the hold margin is increased.
Modify the \texttt{clk2} phase setting to $-55^\circ$, and recompile the design for the new PLL setting to take effect. Run the report timing script again. Figure 2–18 shows the timing margin reported in the Quartus II software after adjusting the phase setting of \texttt{clk2}. The address and command datapath now has a more balanced 2.117 ns setup and 2.128 ns hold margin.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{Figure2-18.png}
\caption{Timing Margin Reported After Adjusting \texttt{clk2}}
\end{figure}

\section*{Determine Board Design Constraints}

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

The DDR2 SDRAM supports dynamic parallel on-die termination (ODT). This feature allows you to turn on ODT when the FPGA is writing to the DDR2 SDRAM memory and turn off ODT when the FPGA is reading from the DDR2 SDRAM memory. The ODT features are available in settings of 150 $\Omega$, 75 $\Omega$, and 50 $\Omega$. The 50-$\Omega$ setting is only available in DDR2 SDRAM with operating frequencies greater than 267 MHz.
Refer to the respective memory data sheet for additional information about the available settings for the ODT and the output driver impedance features, and the timing requirements for driving the ODT pin in DDR2 SDRAM.

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

Figure 2–19 shows the setup for write operation to the DDR2 SDRAM memory with Class I termination together with the drive strength setting of the Cyclone III or Cyclone IV device. In this setup, the driver’s (FPGA) output impedance matches that of the transmission line, resulting in optimal signal transmission to the DDR2 SDRAM memory. On the receiver (DDR2 SDRAM memory) side, the receiving pin is properly terminated with matching impedance to the transmission line, through the external pull-up resistor to \( V_{TT} \), to eliminate any ringing or reflection.

Figure 2–20 shows the setup for read operation from the DDR2 SDRAM memory.

If you choose not to use the drive strength setting, you can use the OCT feature of the Cyclone III or Cyclone IV device instead.
Finally, the loading seen by the FPGA during writes to the memory is different in a system using dual-inline memory modules (DIMMs) and a system using components. The additional loading from the DIMM connector can reduce the edge rates of the signals arriving at the memory, thus affecting available timing margin.

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

For detailed information about different effects on signal integrity design, refer to the *Board Layout Guidelines* section in volume 2 of the *External Memory Interface Handbook*. 
3. Using High-Performance DDR, DDR2, and DDR3 SDRAM with SOPC Builder

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

This walkthrough discusses the following topics:

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

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

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

SOPC Builder System Considerations

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

- High-Performance Controller (HPC) or High-Performance Controller II (HPC II)
- Full- or Half-Rate SDRAM High-Performance Controller
- Clock Selection and Clock Crossing Bridges
- Burst Reads and Writes
- Multiple Masters
- Direct Memory Access (DMA) Controller
Read and Write Addressing and Latency

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

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

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

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

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

Full- or Half-Rate SDRAM High-Performance Controller

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

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

Implementing the SDRAM high-performance controllers in half-rate mode gives the highest possible SDRAM clock frequency while allowing the more complex core logic to operate at half this frequency. You can simplify the complexity of your design by permitting your Nios II processor to run at the slower, half-rate memory speed while still achieving the required SDRAM bandwidth per I/O pin.
However, where possible it is generally more optimal to configure the controller in full-rate mode with the core operating at the same clock frequency as your SOPC Builder system.

**Full-Rate Versus Half-Rate Command Operation**

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

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

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

**Time-Specified Memory Parameters**

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

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

For example, if:

\[ t_{RC\min} = 20 \text{ ns} \]

For a 133-MHz controller:

\[ t_{CK} = 7.5 \text{ ns} \]

\[ 20/7.5 = 2.667 \text{ rounded up to 3 clock cycles (22.5ns).} \]

For a half-rate 66-MHz controller:

\[ t_{CK} = 15 \text{ ns} \]

\[ 20/15 = 1.33 \text{ rounded up to 2 clock cycles (30ns).} \]

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

Ideally every component in the SOPC Builder system should be clocked using the same clock, to prevent SOPC Builder automatically adding clock domain crossing logic, which adds latency. The SDRAM high-performance controllers already include a PLL, so you should set the SOPC Builder system clock to `altmemddr.sysclk`, which is the clock the controller and local interface logic use. SOPC Builder only shows the input clock to the controller PLL, not the interface clock, refer to Figure 3–1 on page 3–4.

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

To prevent SOPC Builder from auto-inserting clock adapters:

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

Notes to Figure 3–1:

(1) The `clk` clock is the input clock to the controller PLL.
(2) The `s1` clock is not stated, but is on the `altmemddr.sysclk` clock domain, not on the `clk` domain.
If the connected SOPC Builder components and SDRAM high-performance controller operate at different frequencies, manually insert an Avalon-MM clock crossing bridge between the SDRAM high-performance controller and the other SOPC Builder components.

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

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

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

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

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

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

The Avalon-MM DDR memory half-rate bridge has the same functionality as the Avalon-MM clock crossing bridge but provides lower latency. Altera recommends you use the Avalon-MM DDR memory half-rate bridge for Nios II Processors that require low latency access to high-speed memory.
Figure 3–2 shows a block diagram of how to use an Avalon-MM clock crossing bridge or an Avalon-MM DDR memory half-rate bridge.

**Figure 3–2. Clock Crossing Bridge**

![Clock Crossing Bridge Diagram](image)

Burst reads and writes differ in the following ways:

**Burst Reads and Writes**

Burst reads and writes differ in the following ways:

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

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

![Half-Rate Bridge Diagram](image)
In half-rate designs, the Avalon-MM slave interface is four times the width of the SDRAM device. Hence four transactions are performed on the SDRAM for every single Avalon-MM transaction. Avalon-MM burst requests on the local side serve no purpose as the memory interface is already using the maximum supported memory burst size for every single Avalon-MM transaction.

In full-rate designs, you can use Avalon-MM bursts of one or two with the SDRAM high-performance controller, as each Avalon-MM transaction results in only two SDRAM transactions. So two Avalon-MM burst transactions may be combined to support the four supported on the SDRAM high-performance controller.

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

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

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

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

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

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

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

Table 3–1 and Table 3–2 show the burst length support for each type of DDR SDRAM HPC and HPC II.

**Table 3–1. Burst Length Support for DDR SDRAM HPC**

<table>
<thead>
<tr>
<th>Memory Type</th>
<th>Full-Rate Mode</th>
<th>Half-Rate Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDR3</td>
<td>—</td>
<td>8</td>
</tr>
<tr>
<td>DDR2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>DDR</td>
<td>2 or 4</td>
<td>4</td>
</tr>
</tbody>
</table>

**Table 3–2. Burst Length Support for DDR SDRAM HPC II**

<table>
<thead>
<tr>
<th>Memory Type</th>
<th>Full-Rate Mode</th>
<th>Half-Rate Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDR3</td>
<td>—</td>
<td>8</td>
</tr>
<tr>
<td>DDR2</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>DDR</td>
<td>4</td>
<td>8</td>
</tr>
</tbody>
</table>
HPC II consists of a built-in burst adapter that automatically splits or merges burst request to match the memory’s native burst length. This built-in adapter allows the Nios II processor to request any burst length and maps the burst length to the most efficient memory burst. HPC II accepts up to 64 burst count.

**Multiple Masters**

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

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

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

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

**Direct Memory Access (DMA) Controller**

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

For more information, refer to the *DMA Controller Core* chapter in volume 5 of the *Quartus II Handbook*. 
The DMA controller can perform bulk data transfers, reading data from a source address range and writing data to a different address range. The DMA controller boosts the memory bandwidth and alleviates the multi-master bottleneck. With the use of a DMA controller, you can reduce the number of masters that access the memory. Figure 3–4 shows an example where there are multiple masters access to the memory. This increases the complexity of the connection and latency to the system.

**Figure 3–4. Multiple Masters Accessing a Single Memory**

You can use the DMA controller to reduce the bottleneck for the multi-master access to the memory. The DMA controller acts as the only master that accesses the memory but as a slave for other components that wants access to the memory. DMA controller uses interrupt request to notify the masters of other components that the transfers are completed without wasting time with polling. Figure 3–5 shows an example of how DMA controller is used to enhance the memory bandwidth for multiple masters.

**Figure 3–5. A Single Master Accessing Memory with DMA Controller**

**Read and Write Addressing and Latency**

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

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

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

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

### DDR2 SDRAM Controller with ALTMEMPHY IP with SOPC Builder Walkthrough

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

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

### System Requirement

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

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

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

Create Your Example Project

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

Create a New Quartus II Project

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

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

Create the SOPC Builder System

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

1. On the Tools menu, click SOPC Builder.

2. In the Create New System dialog box, type sopc_top for the System Name. In the Target HDL, select Verilog, then click OK.

3. In the System Contents tab, under Component Library, expand Memories and Memory Controllers. Expand SDRAM. Select DDR2 SDRAM High Performance Controller and click Add. The DDR2 SDRAM Controller with ALTMEMPHY IP wizard opens, refer to Figure 3–6 on page 3–11.

Figure 3–6. System Content
4. In the DDR2 High Performance Controller wizard interface, select 7 for the Speed grade to match the chosen device.

5. For the PLL reference clock frequency, type 50 MHz, to match signal CLKIN50.

6. For the Memory clock frequency, type 150 MHz.

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

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

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

8. For the Memory vendor, select Micron.

9. For the Memory format, select Discrete Device.

10. For the Memory Presets, select Micron MT47H32M16CC-3 x4 + MT47H32M8BP-3 x1, refer to Figure 3–7 on page 3–13.
This memory preset has been included by Altera in the DDR2 SDRAM Controller with ALTMEMPHY IP as it matches the exact configuration that both the Cyclone III development board and the Stratix II GX PCIe development kit use.

Figure 3-7. Memory Settings

**DDR2 SDRAM High Performance Controller**

- **Device family:** Cyclone III
- **Speed grade:** N/A
- **PLL reference clock frequency:** 50 MHz (20000 ps)
- **Memory clock frequency:** 150 MHz (66 ps)
- **Controller data rate:** Full
- **Local interface clock frequency:** 75.0 MHz
- **Local interface width:** 32 bits

**Memory Presets**

- **Preset:**
  - Micron MT47H512M1600-3.3V
  - Micron MT47H512M1600-3.3V
  - Micron MT47H512M1600-3.3V
  - Micron MT47H512M1600-3.3V
  - Micron MT47H512M1600-3.3V

**Description:** DDR2 SDRAM, 333-333MHz, 32MB, 32 bits wide, DQ = 3.0, 1 Chip Select

Info: The PLL is generated with Memory clock frequency 1000 MHz and 41 phase steps per cycle.
11. Click **Modify parameters** and ensure that you change the following parameters:

- **Output clock pairs from FPGA**, select **1 pair**
- **Total Memory interface DQ width**, select **8 bits**, to give in an Avalon-MM width of 32 bits, which is ideal to connect to a Nios II processor
- **Memory drive strength setting**, select **Reduced** (DQ are low load point-to-point connections)
- **Memory on-die termination (ODT) setting**, select **Disabled** (discrete class I termination is already fitted to the development board)
- **Memory CAS latency setting**, select **3.0 cycles** (this DDR2 SDRAM supports \( \text{CAS} = 3 \) for frequencies of 200 MHz or lower)

12. Click **OK**, then click **Finish** in the DDR2 SDRAM Controller with ALTMEMPHY IP interface.

**Figure 3–8. Preset Editor**
If you are using a device that requires timing model pessimism removal, ensure that you turn on the Enable Memory Chip Calibration in Timing Analysis option on the Advanced page. This option is not supported by Cyclone III devices.

**Figure 3–9. Advanced Memory Settings**

For Cyclone III devices ensure that you enter a value for the Board skew parameter in the PHY Settings tab based on your board simulation results, and fill in the board trace and loading information to achieve accurate timing analysis. Refer to “Enter Board Trace Delay Model” on page 3–29 for more information about the board trace and loading information.

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

13. In the Controller Settings tab, select High-Performance Controller II.
15. For Local-to-Memory Address Mapping, select CHIP-ROW-BANK-COL.
16. For Local Maximum Burst Count, select 1, refer to Figure 3–10.
Set the maximum burst count based on the burst count of the core logic system.

Figure 3–10. DDR2 SDRAM HPC II Settings

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

For more information about how to share the clock for multiple memory interfaces, refer to the **Implementing Multiple Memory Interfaces** section in volume 5 of the *External Memory Interface Handbook*.

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

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

1. Under Component Library, expand Memories and Memory Controllers and expand On-Chip. Select On-Chip Memory (RAM or ROM), and click Add.
   a. For Total memory size type 64 KBs.
   b. Click Finish.

2. Select On-Chip Memory again, click Add.
   a. Set Total memory size to 4096 Bytes.
   b. Click Finish.
   c. Right click on this second on-chip memory, click Rename and type dma_read_memory and press Enter.


   a. Select Nios II/s.
   b. For Reset Vector Memory and Exception Vector Memory, select onchip_memory. If the local on-chip memory holds the Nios II instruction code, less arbitration is required to the SDRAM interface resulting in a more optimal Avalon-MM structure.
   c. Change Reset Vector Offset to 0x20 and Exception Vector Offset to 0x40.

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

5. Expand Interface Protocols and expand Serial, select JTAG UART and click Add.
   a. In the Configuration tab, under Write FIFO and Read FIFO, for the Buffer depth (bytes) select 64 and for IRQ threshold type 8.
   b. In the Simulation tab, select Create Modelsim alias to open an interactive stimulus/response window, to open the interactive display window during simulation.
   c. Click Finish.

6. Expand Peripherals and expand Microcontroller Peripherals, select PIO (Parallel I/O), click Add.
   a. For the Width type 8 bits.
   b. For Direction select Output ports only.
   c. Click Finish.
7. Expand Memories and Memory Controllers and expand DMA, select DMA Controller and click Add.
   a. In the DMA Parameters tab, turn on Enable burst transfers, and for the Maximum burst size select 512 words.
   b. In the Advanced tab, turn off byte, halfword, doubleword and quadword.
   c. Click Finish.

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

8. Set the bus links, refer to Figure 3–11 on page 3–19.
   a. Connect the DMA read_master and write_master to both the altmemddr and the dma_read_memory.

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

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

   b. Ensure that altmemddr is clocked on the external clock clk and that the frequency matches the external oscillator (50 MHz for the Cyclone III development board).

   c. Ensure that all other modules are clocked on the altmemddr_sysclk, to avoid any unnecessary clock-domain crossing logic.
Generate the SOPC Builder System

To generate the system, perform the following steps:

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

   SUCCESS: SYSTEM GENERATION COMPLETED.

4. In SOPC Builder, click Exit.

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

Create the Top-Level Design File

Conceptually, you can consider the SOPC Builder system as component in your design. It can be the only component; or one of many components. Hence, when your SOPC Builder system is complete, you must add it to your top-level design.
The top-level design can be in your preferred HDL language, or simply a `.bdf` schematic design.

In this walkthrough, the top-level design is a simple wrapper file around the SOPC Builder system with no additional components. The top-level design file just defines the pin naming convention and port connections. Figure 3–12 shows the SOPC Builder top-level block diagram.

![Figure 3–12. SOPC Builder Top-Level Block Diagram](image)

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

All the SDRAM high-performance controllers generate the standalone design example `altmemddr_example_top`. This file only includes the controller and an example driver, refer to Figure 3–13. This file does not instantiate your SOPC Builder system but you can use the file in one of the following ways:

- Identify the default memory controller pin names
Use as a starting point for the rest of the design.

**Figure 3–13. Design Example**

![Diagram of the design example](image.png)

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

As a reference, Altera provides an example `.bdf` top-level design with the correct pin names, refer to **Figure 3–14**. The `mem_clk[0]` and `mem_clk_n[0]` pins are bidirectional because of the mimic path in the ALTMEMPHY megafunction.

**Figure 3–14. Quartus II Top-Level Project**

![Quartus II top-level project](image.png)
To create a top-level design for your SOPC Builder system using a Quartus II .bdf schematic, perform the following steps:

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

   ![The Quartus II software automatically sets the .bdf file name to your project name.]

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

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

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

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

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

8. The SOPC Builder system has two reset inputs, reset_n and global_reset_n_to_the_altmemddr. Connect both these signals to a single pin, global_reset_n.

9. Ensure that you rename the clock signal as clock_source.

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

10. Rename out_port_from_the_Led_pio[7...0] to pio[7...0].

11. On the File menu, click Save, to save your changes.

12. On the Project menu, click Set as Top-Level Entity.

Add Constraints

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

Device Settings

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

1. In the Category list, select Device and click Device and Pin Options.

2. Click the Unused Pins tab, and for Reserve all unused pins select As input tri-stated with weak pull-up resistor.

3. Click the Voltage tab, and for Default I/O standard select the same VCCIO voltage as the chosen SDRAM interface; for DDR2 SDRAM select 1.8 V.

4. Click OK.

Add Timing Constraints

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

To add timing constraints, perform the following steps:

1. On the Assignments menu, click Settings.
2. In the Category list, expand Timing Analysis Settings, and select TimeQuest Timing Analyzer.

3. Select the altmemddr_phy_ddr_timing.sdc, altera_avalon_half_rate_bridge_constraints.sdc cpu.sdc, and cycloneIII_3c120_generic.sdc files and click Add.

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

4. Click OK.

Set Optimization Technique

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

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

Set Fitter Effort

To set the Fitter effort, perform the following steps:

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

Add Pin, DQ Group, and IO Standard Assignments

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

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

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

   If you require pin name changes (prefix changes only), then in the Quartus II Pin Planner, perform the following steps:
1. On the Assignment menu, click Pin Planner.
2. Right-click in any area under Node Name and select Create/Import Megafunction, refer to Figure 3–15.
3. Select Import an existing custom megafuction and select the <variation name>-.ppf file.
4. In the Instance name field, type the prefix that you want to use, refer to Figure 3–16 on page 3–25.

Figure 3–15. Create/Import Megafunction in Pin Planner

Alternatively, you may edit the altmemddr_pin_assignments.tcl file to change the prefix, and assign the pins without the _from_to_ pin names, by following these steps:
1. Open the altmemddr_pin_assignments.tcl file.
2. To create assignments that have _from_to_ pin names, change to the following code:

![Figure 3–16. Adding Prefix](image)
switch $sopc_mode {
    YES {
        set output_suffix _from_the_${instance_name}
        set bidir_suffix _to_and_from_the_${instance_name}
        set input_suffix _to_the_${instance_name}
    }
    default {
        set output_suffix ""
        set bidir_suffix ""
        set input_suffix ""
    }
}

to

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

3. Type your preferred prefix in the pin_prefix variable. For example, to add prefix `mem_`, change following code:
   
   ```tcl```
   if {![info exists set_prefix]{set pin_prefix "my mem_"}
   ```tcl```
   
4. Save the `altmemddr_pin_assignments.tcl` file.
5. Run the `.tcl` script.

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

**Pin Location Assignments**

To assign pin locations, perform the following steps:

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

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

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

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

1. On the Assignments menu, click Pin Planner.
2. Assign DQ and DQS pins.
   
a. To select the device DQS pin groups that the design uses, assign each DQS pin in your design to the required DQS pin in the Pin Planner. The Quartus II Fitter then automatically places the respective DQ signals onto suitable DQ pins within each group. To see DQS groups in the Pin Planner, right click, select Show DQ/DQS Pins, and click In x8/x9 Mode. The Pin Planner shows each DQS group in a different color and with a different legend: S = DQS pin, Sbar = DQSn pin and Q = DQ pin, refer to Figure 3–17 on page 3–28.

Most DDR2 SDRAM devices operate in x8/x9 mode, however as some DDR2 SDRAM devices operate in x4 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.

3. Place the DM pins within their respective DQ group.

4. Place the address and control/command pins on any spare I/O pins ideally within the same bank or side of the device as the mem_clk pins.

5. Ensure the mem_clk pins use any regular adjacent I/O pins—ideally differential I/O pairs for the CK/CK# pin pair. To identify differential I/O pairs, right-click in Pin Planner and select Show Differential Pin Pair Connections (DIFFIO in Pin Planner). Pin pairs show a red line between each pin pair.

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

   The DDR2 SDRAM in Stratix III and Stratix IV devices with differential DQS mode require the mem_clk[0]/mem_clk_n[0] pin on a DIFFIO_RX capable pin pair. The DDR3 SDRAM interfaces in Stratix III and Stratix IV devices require the mem_clk[0]/mem_clk_n[0] pin in any DQ or DQS pin pair with DIFFIO_RX capability.
For more information about memory clock pin placement and external memory pin selection, refer to the Device and Pin Planning section in volume 2 of the External Memory Interface Handbook.

6. Place the clock_source pin on a dedicated PLL clock input pin with a direct connection to the SDRAM controller PLL/DLL pair—usually on the same side of the device as your memory interface. This recommendation reduces PLL jitter, saves a global clock resource, and eases timing and fitter effort.

7. Place the global_reset_n pin (like any high fan-out signal) on a dedicated clock pin.

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

Enter Board Trace Delay Model

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

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

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

2. Right-click and select Board Trace Model.
Figure 3–18 on page 3–30 shows the board trace model. Enter the values such as the trace length, capacitance/length, inductance/length, pull-up, pull-down and others based on your board.

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

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

<table>
<thead>
<tr>
<th>Net</th>
<th>Near (FPGA End of Line)</th>
<th>Far (Memory End of Line)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Length</td>
<td>C_Per_Length</td>
</tr>
<tr>
<td>Address (1)</td>
<td>1.764</td>
<td>3.8p</td>
</tr>
<tr>
<td>CLK</td>
<td>1.872</td>
<td>3.4p</td>
</tr>
<tr>
<td>CKE/CSn</td>
<td>1.862</td>
<td>3.7p</td>
</tr>
<tr>
<td>ODT</td>
<td>1.527</td>
<td>3.8p</td>
</tr>
<tr>
<td>DQS (2)</td>
<td>1.165</td>
<td>3.6p</td>
</tr>
</tbody>
</table>

Note to Table 3–3:
(1) Address = addr, ba, we#, ras#, and cas.
(2) DQS = to DM, DQ, and DQS pins.

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

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

Figure 3–19 shows the timing margin report in the message window in the Quartus II software.

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

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

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

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

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

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

Launch the Nios II IDE

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

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

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

To see the `DDR_TEST.c` example test program, refer to “Example DDR_TEST.c Test Program File” on page 3–45.

This program consists of a simple loop that takes commands from the JTAG UART and executes them. Table 3–4 shows the commands that are sent to the Nios II C code.

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

**Table 3–4. Commands**

<table>
<thead>
<tr>
<th>Command</th>
<th>Format</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Switch A</td>
<td>Switch</td>
<td>Data</td>
<td>Controls the LEDs.</td>
</tr>
<tr>
<td>Switch B</td>
<td>Switch</td>
<td>Address</td>
<td>Data</td>
</tr>
<tr>
<td>Switch C</td>
<td>Switch</td>
<td>Address</td>
<td>Performs a single read to an address offset location in memory.</td>
</tr>
<tr>
<td>Switch D</td>
<td>Switch</td>
<td>From Address</td>
<td>To Address</td>
</tr>
</tbody>
</table>
You can read the SOPC Builder component address locations directly from the SOPC Builder Window, refer to Figure 3–21.

**Figure 3–21. SOPC Builder Component Address**

Set Up the Project Settings
To set up the Nios II project settings, perform the following steps:

1. In the Nios II C/C++ Projects tab, right-click blank_project_0 and click System Library Properties.
2. Click System Library in the list on the left side of the Properties dialog box.
3. To optimize the footprint size of the library project, under System Library Contents, turn on Reduced device drivers.
4. Under Linker Script, for all the memory options select onchip_mem. The on-chip memory is the target memory space for the executable file.
5. To reduce the memory size allocated for the system library, for Max file descriptors type 4.
   Figure 3-22 shows the Properties dialog box.

   Figure 3-22. Nios II Project Settings

   6. Click OK.

   Perform RTL or Functional Simulation (Optional) with Nios II

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

   1. From the Nios II IDE, on the Run menu click RUN.
   2. Under Nios II ModelSim, select blank_project_0.
   3. Specify your ModelSim installation path in ModelSim Path.
   4. Click Run. ModelSim is launched.
   5. Type s in the ModelSim Transcript window to run the s macro to load the design.
   6. Type w in the ModelSim Transcript window to run the w macro to display the waveform.
   7. Type jtag_uart_drive to launch the interactive terminal.
   8. Type Run -all to run the design.
9. In the interactive terminal, type your desired operation, data, and address, refer to Figure 3–23.

Figure 3–23. The JTAG UART Interactive Terminal
Figure 3–24 shows a single write operation where data 00000001 is written to address 00000001.

Verify Design on a Development Platform

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

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

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

1. On the Tools menu, click SignalTap II Logic Analyzer.
2. In the Signal Configuration window next to the Clock box, click … (Browse Node Finder).
3. Type *phy_clk in the Named box, for Filter select SignalTap II: pre-synthesis and click List.
4. Select `sopc_top:inst|altmemddr:the_altmemddr|phy_clk` in Nodes Found and click > to add the signal to Selected Nodes.

5. Click OK.

6. Under Signal Configuration, specify the following settings:
   - For **Sample depth**, select 512
   - For **RAM type**, select Auto
   - For **Trigger flow control**, select Sequential
   - For **Trigger position**, select Center trigger position
   - For **Trigger conditions**, select 1

7. On the Edit menu, click Add Nodes.

8. Search for specific nodes by typing `*local*` in the Named box, for Filter select SignalTap II: pre-synthesis and click List.

9. In Nodes Found, select the following nodes and click > to add to Selected Nodes:
   - `local_address`
   - `local_rdata`
   - `local_rdata_valid` (alternative trigger to compare read/write data)
   - `local_read_req`
   - `local_ready`
   - `local_wdata`
   - `local_write_req` (trigger)

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

10. Click OK.

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

Figure 3–25 shows the completed SignalTap II Embedded Logic Analyzer.

**Figure 3–25. SignalTap II Embedded Logic Analyzer**

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

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

**Compile the Project**

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

**Verify Timing**

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

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

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

**Connect the Development Board**

Connect the Cyclone III development board to your computer.

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

**Download the Object File**

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

The SRAM Object File (SOF) Manager should contain the `<your project name>.sof` file.

To add the correct file to the SOF Manager, perform the following steps:

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

![Figure 3–26. Install the SRAM Object File in the SignalTap II Dialog Box](image)

**Verify Design with Nios II**

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

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

1. From the Nios II IDE, on the Run menu click RUN. Click the Target Connection tab and correct the JTAG cable connection, refer to Figure 3–27.
2. Right click on `blank_project_0`, point to `Run As`, and click `Nios II Hardware`.

**Figure 3–27. JTAG Download Target Connection Settings**

![JTAG Download Target Connection Settings](image)

**Test the System**

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

**Set SignalTap II Trigger to `local_write_req`**

Use the SignalTap II Embedded Logic Analyzer to capture write activity. To show write activity, perform the following steps:
1. In the Setup tab, select to trigger on a rising edge of local_write_req, refer to Figure 3–28.

**Figure 3–28. Set Trigger for local_write_req Signal**

<table>
<thead>
<tr>
<th>Node</th>
<th>Data Enable</th>
<th>Trigger Enable</th>
<th>Trigger Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>...en DDR[local_address]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>...mil DDR[local_data]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>...tm DDR[local_data_valid]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>...atm DDR[local_read_req]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>...he alt DDR[local_ready]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>...tm DDR[local_wdata]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>...tm DDR[local_wdata req]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>...al DDR[local_write_req]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

2. Click Run Once Analysis to capture the write request.

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

   B000000010000001

4. Return to the SignalTap II Embedded Logic Analyzer window and check that at time 0 (refer to Figure 3–29):
   - local_write_req signal goes high for a single cycle
   - local_wdata shows 00000001h
   - local_address shows 000001h

**Figure 3–29. Waveform for Write Request**

Set SignalTap II Trigger to local_read_req

Use the SignalTap II Embedded Logic Analyzer to capture read activity. To show read activity, perform the following steps:
1. In the **Setup** tab, select to trigger on a rising edge of `local_read_req`, refer to Figure 3–30.

**Figure 3–30. Set Trigger for local_read_req Signal**

<table>
<thead>
<tr>
<th>Node</th>
<th>Data Enable</th>
<th>Trigger Enable</th>
<th>Trigger Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>`...emdddi</td>
<td>local_address`</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>`...emdddi</td>
<td>local_rdata`</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>`...lmnddi</td>
<td>local_rdata_valid`</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>`...lmnddi</td>
<td>local_read_req`</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>`...lmnddi</td>
<td>local_ready`</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>`...lmnddi</td>
<td>local_wdata`</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>`...lmnddi</td>
<td>local_write_req`</td>
<td>✔️</td>
<td>✔️</td>
</tr>
</tbody>
</table>

2. Click **Run Once Analysis** to capture the read request.

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

   C00000001

4. Return to the SignalTap II Embedded Logic Analyzer window and check that at time 0 (refer to Figure 3–22):
   - `local_read_req` signal goes high
   - `local_address` shows 000001h
   
   Several clock cycles later, depending on the system read latency:
   - `local_rdata_valid` goes high for one cycle
   - `local_rdata` shows 00000001h

**Figure 3–31. Waveform for Read Request**

**Test Burst Write Operation**

Use the SignalTap II Embedded Logic Analyzer to capture write activity. To show write activity, perform the following steps:
1. In the **Setup** tab, select to trigger on a rising edge of `local_write_req`, refer to Figure 3–32.

**Figure 3–32. Set Trigger for `local_write_req` Signal**

<table>
<thead>
<tr>
<th>Node</th>
<th>Data Enable</th>
<th>Trigger Enable</th>
<th>Trigger Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>local_write_req</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2. Click **Run Once Analysis** to capture the write request.

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

\[
D0200000000000000
\]

4. Return to the SignalTap II Embedded Logic Analyzer window and check that at time 0 (refer to Figure 3–22):

- `local_write_req` signal goes high for multiple cycles
- `local_wdata` shows `03020100h` followed by `07060504h` and so on
- `local_address` shows `000000h` followed by `000002h` and so on

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

**Figure 3–33. Waveform for Burst Write Request**

---

**Test Burst Read Operation**

Performing a burst read operation is similar to performing a burst write operation (refer to “**Test Burst Write Operation**” on page 3–43), but with the memory locations swapped. To perform a burst read operation, perform the following steps:

1. In the **Setup** tab, set to trigger on a rising edge of `local_read_req`.
2. Click **Run Once Analysis** to capture the read request.
3. Type the following command in the Nios II command console:

```
D0000000002000000
```

Example DDR_TEST.c Test Program File

```c
#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 Select One of the following Commands \n\n");
    printf(" Enter 'A' : Controls the LEDS: \n\n");
    printf(" Enter 'B' : Single Write to an address location in Memory: \n\n");
    printf(" Enter 'C' : Single Read to an address location in Memory: \n\n");
    printf(" Enter 'D' : Performs DMA operation with burst length of 512 words: \n\n");
    printf(" Enter your command now \n\n");
}

/**************************************************************************
 * Function: LED_Control
 * Purpose: Controls the LEDs..
 *
**************************************************************************/
void LED_Control(void)
{
Example DDR_TEST.c Test Program File

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

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

/**************************************************************************
 * Function: Single_Read
 * Purpose: Performs a single read to an address location in memory
 ***************************************************************************/
void Single_Read(void)
{
 unsigned char read_offset[8];
 unsigned int DDR_read_OFFSET_ADDRESS;
```
unsigned int read_data;
printf(" \n Single Read operation \n");
printf( "\n Enter the offset address from where you want to read in
the Memory:(i.e. 00000010) \n");
gets(read_offset);
DDR_read_OFFSET_ADDRESS = to_hex(&(read_offset[0]));
if ((DDR_read_OFFSET_ADDRESS<0) ||
    (DDR_read_OFFSET_ADDRESS>(ALTMEMDDR_SPAN/4)))
{
    printf(" \n Invalid Offset \n");
}
else
{
    read_data=IORD(ALTMEMDDR_BASE,DDR_read_OFFSET_ADDRESS);
    printf("Read %08x from address %08x
",read_data,(ALTMEMDDR_BASE+DDR_read_OFFSET_ADDRESS));
}

/***************************************************************
*  Function: Verify Operation                                     *
*  Purpose: Compares the memory contents of the read data master and write
*  data master                                                    *
***************************************************************
void Call_verify(unsigned char* source, unsigned char* destination)
{
    if (memcmp(source,destination,(length*4))==0)
    {
        printf("\n DMA operation successful \n");
    }
    else
    {
        printf("\n DMA operation failed \n");
        printf( "\n Please check that DMA is correctly setup \n ");
    }
}
/***************************************************************
*  Function: DMA_Operation                                        *
*  Purpose: Performs an incremental write to the first address
*  location, followed by a DMA burst transfer from the first address
*  location to the second address location.
The burst is a fixed length of 512 words or 2048 bytes.
***************************************************************
void DMA_operation(void)
{
    unsigned int read_address;
    unsigned char* read_DMA_address;
    unsigned char *write_DMA_address;
    unsigned int write_address;
    unsigned char verify[8];
    unsigned char read[8];
    unsigned char write[8];
    unsigned char status_reg;
    unsigned int i;
    printf(" \n DMA setup operation \n");
    printf( "\n Enter DMA read master address:(i.e. 00000010) \n ");
    gets(read);
    read_address =to_hex(&(read[0]));
    read_DMA_address=read_address;
    printf( " \n DMA read master address is %08x \n",read_DMA_address);
printf(" \n Enter DMA write master address: (i.e. 00000010) \n ");
gets(write);
write_address = to_hex(&write[0]);
write_DMA_address = write_address;
printf(" \n DMA write master address is %08x \n", write_DMA_address);
printf(" \n Using %d bytes to verify the DMA operation \n", (length*4));
printf(" \n Initializing Read memory with incremental data starting
 from 00 \n");
for (i=0; i<(length*4); i++)
{
    *read_DMA_address = i;
    read_DMA_address++;
}
read_DMA_address = read_address;
printf(" \n Writing to the DMA control registers \n");
IOWR_ALTERA_AVALON_DMA_CONTROL(DMA_BASE, 0x00000084);
//DMA go bit is set to zero.
IOWR_ALTERA_AVALON_DMA_STATUS(DMA_BASE, 0x00000000);
//Clearing done bit
IOWR_ALTERA_AVALON_DMA_RADDRESS(DMA_BASE, read_DMA_address);
IOWR_ALTERA_AVALON_DMA_WADDRESS(DMA_BASE, write_DMA_address);
IOWR_ALTERA_AVALON_DMA_LENGTH(DMA_BASE, (length*4));
printf(" \n Starting DMA engine \n");
IOWR_ALTERA_AVALON_DMA_CONTROL(DMA_BASE, 0x0000008C);
printf(" \n DMA operation is in progress \n");
status_reg = (IORD_ALTERA_AVALON_DMA_STATUS(DMA_BASE) &
ALTERA_AVALON_DMA_STATUS_DONE_MSK);
if (status_reg == 1)
{
    printf(" \n DMA operation is Done \n ");
    printf(" \n Do you want to verify the DMA operation \n");
    printf(" \n Enter Y for yes, N for No \n");
    gets(verify);
    switch(verify[0])
    {
        case 'N':
            break;
        case 'Y':
            CallVerify(read_DMA_address, write_DMA_address);
            break;
        default:
            printf(" \n 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 ;
}
Many systems and applications use external memory interfaces as data storage or buffer mechanisms. As system applications require increasing bandwidth, become more complex, and devices get more expensive, designers prefer to have multiple memory interfaces in a single device to save cost and space on the board, along with removing partitioning complexity. To implement multiple memory interfaces on the same device that are efficient and optimized for the device architecture, designers must pay attention to the device and the physical interface (PHY) features.

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

While you may not be able to share one PLL between multiple ALTMEMPHY-based controllers, you may be able to share the static clocks to reduce the number of clock networks used in the design.

You cannot merge dynamic clocks of the ALTMEMPHY megafunction or high-performance controllers. The Quartus II software may not give a warning, but this merging does not work in the hardware.

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

Before Creating a Design in the Quartus II Software

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

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

The following different scenarios exist for the resources sharing DLL and PLL static clocks:
Multiple controllers with no sharing. You need to check the number of PLLs and DLLs that available in the targeted device and the number of PLLs and DLLs that needed in your design. If you have enough resources for the controllers, you do not need to share the DLL and PLL static clocks.

Multiple controllers sharing DLL only. If you have enough resources on PLL clocks output and only need to share DLL, ensure that the controllers are running at the same memory clock frequency.

Multiple controllers sharing PLL static clocks only. Ensure that all the controllers sharing the PLL static clocks are located in the same half of the device, as ALTMEMPHY requires the use of regional and dual-regional clocks, instead of global clocks. Each controller can have their own DLL as the DLL can only access the adjacent sides from its location.

Multiple controllers sharing DLL and PLL static clocks. The PLL static clocks and DLL can be shared when the controllers are on the same side or adjacent side of the device and provided they are running at the same memory-clock frequency.

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

Creating PHY and Controller in a Quartus II Project

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

■ “SOPC Builder Flow”
■ “MegaWizard Plug-In Manager Flow”

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

SOPC Builder Flow

The SOPC Builder in version 8.1 and onwards of the Quartus II software allows you to add multiple controllers directly to a new or existing SOPC Builder system. The SOPC Builder supports sharing static PHY clocks between multiple controllers in the SOPC Builder that are running on the same frequency and sharing the same PLL reference clock. To share the static clocks, you must turn on the Clock source from another controller option in the Controller Settings page of the DDR, DDR2, or DDR3 SDRAM High Performance Controller wizard. This option must be turned on for the
slave controllers. Turning on this option adds a new clock input connection point to the slave controller named shared_sys_clk. You must connect the sys_clk signal from the master controller to the shared_sys_clk signal of the slave controller (Figure 4–1). The Force Merging of PLL Clock Fanouts assignment is automatically assigned in the Quartus II software.

Figure 4–1. Connection Between Master and Slave Controller

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

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

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

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

For more information on simulating SOPC Builder systems, refer to AN 351: Simulating Nios II Systems.
MegaWizard Plug-In Manager Flow

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

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

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

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

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

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

Sharing DLLs

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

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

- siii when targeting HardCopy® III, HardCopy IV, Stratix III, or Stratix IV devices
- aii when targeting Arria II GX devices
The example top-level file then instantiates the DLL in addition to instantiating the memory controller and the example driver. You can then instantiate the other memory controllers for the design and either modify the example driver to create the test pattern for all controllers or create another design example for each controller instantiated.

If you do not need to share any PLL static clocks, you can continue to “Adding Constraints to the Design” on page 4–9.

**Sharing PLL Clock Outputs or Clock Networks**

The ALTMEMPHY sources a system clock, for use in your design, to clock your ALTMEMPHY interface. This same clock is used by Altera High-Performance controllers, SOPC Builder systems, and the Avalon-MM interface connected to the memory controller. If you use more than one similarly configured ALTMEMPHY in a design, you can share these system clocks, which allows the ALTMEMPHY interfaces to operate synchronously to each other. PLL static clock sharing has the following benefits:

- Although a PLL instance is used for each ALTMEMPHY instance, the static clocks are shared, which uses fewer clocking resources
- The multiple ALTMEMPHYs operate on the same system clock so can be directly connected to logic on the same clock (for example, Avalon-MM interface) without any clock domain crossing logic required, which causes an increase in logic usage and read latency
PLL static clock sharing reduces the required number of clocks by up to four clock networks, which are used in the memory interface logic to clock the controller and generate DQ, DQS, CK/CK#, and address and command signals. Figure 4–2 shows a diagram of the connection between the DLL and the PLL for this scheme.

**Figure 4–2. Two Controllers Static Clocks with Separate Resynchronization Clocks from Different PLLs**

The maximum number of interfaces that can be shared in this scheme is limited by the number of PLLs and the number of pins available in the device. ALTMEMPHY-based memory interfaces may share static clocks between multiple controllers when the following rules are observed:

- All ALTMEMPHYs are full-rate or all half-rate designs
- All ALTMEMPHYs have the same PLL input and output frequencies
- The ALTMEMPHY memory types are compatible. DDR and DDR2 SDRAM can be mixed in the same system; DDR3 SDRAM controllers may be compatible with DDR or DDR2 SDRAM if the static clock phases are identical.

DDR3 with and without levelling implementations have different static clock phases. Check for the ALTMEMPHY static clock phase compatibility in the clocking tables of the appropriate device and memory interface standard.

- The ref_clk input of each ALTMEMPHY PLL must be connected to a clock signal derived from a single clock source. You can use a clock fanout buffer to create separate PLL reference clocks from a single clock source.
- All of the memory controller’s circuitry is located in the same two device quadrants.
- All of the clocks can be routed as required by their **Global Network Clock Type**. To see the global clocks, look in the Fitter Report > Resource selection > Global and other fast signals.
- The general routing rules defined in the relevant memory standard layout section are met.

Static clocks which can be shared are divided into two categories:
- Static clocks with a fixed phase in the IP (all static clocks except the address and command clock `ac_clk_1x`)
- Static clocks where the phase is set in the IP (the address and command clock `ac_clk_1x`)

Static clocks with a fixed phase in the IP do not require each controller’s interface PCB traces to have the same delays. Static clocks with the phase set in the IP have several options where they may be shared between multiple controllers:
- The address and command clock traces all have the same respective delays from the FPGA to each device
- The TPD delay between the address and command signals and the memory clock of each of the interfaces is the same to all devices
- The TPD delay between the address and command signals and the memory clock of each of the interfaces is similar and the optimal address and command phase for each separate interface is within one to two PLL phase steps. Most designs should be able to use a single `ac_clk_1x` clock shared between all controllers and accept less margin.

For multiple controllers, where not all of the memory interface pins are on the same device edge, evaluating whether you can share static clocks is a complex process and there are many possible combinations of pin outs. In this case, you must evaluate the skews between signals across all controllers and factor this into the rules above for sharing clocks.

When static clocks are to be shared select which controller is to be the master. The slave controllers are made to share the master’s static clocks by using the **Force Merging of PLL Clock Fanouts** assignments. You need to manually add the two PLL merging assignments using the Assignment Editor or directly update the `.qsf` file.

---

The **Force Merging of PLL Clock Fanouts** option requires both the master and slave PLL to have the same input reference clock in the RTL. This option limits the placement of the controllers to be only on the same side where one input clock can provide the reference clock for both center PLLs. If your multiple memory controllers’ PLLs cannot share the same input reference clock pin, you need to manually connect the clocks together via RTL, instead of using this assignment.

If you are not sharing the address and command clocks, generate the slave controller address and command clocks from the master controller’s PLL spare outputs. Create these in the PLL wizard. Modify the PHY source code to bring these clocks out to the top-level file and then modify the slave controllers to connect in the new address and command clock.
Table 4–1 shows a list of PLL merging assignments.

<table>
<thead>
<tr>
<th>Device</th>
<th>Rate</th>
<th>Assignments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arria II GX</td>
<td>Half</td>
<td>set_instance_assignment -name FORCE_MERGE_PLL_FANOUTS ON -from</td>
</tr>
<tr>
<td></td>
<td></td>
<td>*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>set_instance_assignment -name FORCE_MERGE_PLL_FANOUTS ON -from</td>
</tr>
<tr>
<td></td>
<td></td>
<td>*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>Full</td>
<td>set_instance_assignment -name FORCE_MERGE_PLL_FANOUTS ON -from</td>
</tr>
<tr>
<td></td>
<td></td>
<td>*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>Cyclone III</td>
<td>Half</td>
<td>set_instance_assignment -name FORCE_MERGE_PLL_FANOUTS ON -from</td>
</tr>
<tr>
<td></td>
<td></td>
<td>*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>Full</td>
<td>set_instance_assignment -name FORCE_MERGE_PLL_FANOUTS ON -from</td>
</tr>
<tr>
<td></td>
<td></td>
<td>*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>set_instance_assignment -name FORCE_MERGE_PLL_FANOUTS ON -from</td>
</tr>
<tr>
<td></td>
<td></td>
<td>*</td>
</tr>
</tbody>
</table>
|              |      | *|<MASTER>_phy_alt_mem_phy_clk_reset:clk|<MASTER>_phy_alt_mem_phy_pll:*:altpll_component|*:auto_generated|clk[2]
Adding Constraints to the Design

To check that the Quartus II software has applied the assignments, read the Compilation Report - PLL Usage (available only if the fitter step is successful), which shows the two clocks merged. This report allows you to compare the PLL usage before PLL merging and after PLL merging.

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

Adding Constraints to the Design

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

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

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

1. Open the Assignment menu and click on Pin Planner.

Notes to Table 4–1:

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

---

**Table 4–1. PLL Merging Assignments (Note 1) (Part 2 of 2)**

<table>
<thead>
<tr>
<th>Device</th>
<th>Rate</th>
<th>Assignments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stratix III and Stratix IV</td>
<td>Half and Full</td>
<td><code>set_instance_assignment -name FORCE_MERGE_PLL_FANOUTS ON -from</code>&lt;br&gt;`*</td>
</tr>
</tbody>
</table>
2. Right-click in any area under Node Name, and select Create/Import Megafunction (Figure 4–3).

Figure 4–3. Create/Import Megafunction Option in the Pin Planner

3. Turn on the Import an existing custom megafunction radio button and choose the <variation_name>.ppf file.

4. Type the prefix that you want to use under the Instance name (Figure 4–4).

Figure 4–4. Adding Prefix to the Memory Interface Pin Names
You may also want to set the default I/O standard for your design in the Voltage section of the Device and Pin Options, by navigating to Assignment and clicking Setting (Figure 4–5).

Figure 4–5. Setting a Default Voltage for Your Design

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

Compiling the Design to Generate a Timing Report

During design compilation, the Quartus II software analyzes the timing margins for the memory controllers across the three timing model corners. As an alternative, you can also open the Tools menu and click **TimeQuest Timing Analyzer**. Double click on Report DDR to get the timing report for all ALTMEMPHY-based memory controllers in the design for the timing model that you select in the TimeQuest timing analyzer.

For more information about analyzing memory interface timing in Stratix III and Cyclone III devices, refer to the Timing Analysis section in volume 4 of the *External Memory Interface Handbook*.

Timing Closure

You may encounter some of the following timing failures:
If read capture setup time is negative, you need to decrease the delay chain used in the DQ pins by setting the Input Delay from Pin to Input Register to 0 or a lower number than the current setting in the Assignment Editor. However, if hold time is negative, you need to do the opposite; increment the Input Delay from Pin to Input Register setting to a higher number than the current setting in the Assignment Editor.

If you are experiencing any write and address/command timing failures, you may be able to resolve them by setting:

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

If the resynchronization path is not meeting timing, you must move the resynchronization registers closer to the IOE. Report DDR shows information messages with the names of the source and destination registers of the worst case setup path. You must copy the name of the destination register from the message and move it as close as possible to the source register using the Assignment Editor. Similarly, if the postamble path is not meeting recovery timing, move the postamble registers closer to the IOE.

Setup failures for transfers from the measure clock (clk5) to the PHY clock (clk0 for half-rate interfaces or clk1 for full-rate interfaces) should be ignored. Due to the dynamic nature of the measure clock these should be treated as asynchronous transfers.

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

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>June 2011</td>
<td>3.0</td>
<td>Removed the following chapters to List of Designs Using Altera External Memory IP page:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Using High-Performance Controller II with Native Interface Design</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Using DDR and DDR2 SDRAM in Stratix III and Stratix IV Devices</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Using DDR3 SDRAM in Stratix III and Stratix IV Devices</td>
</tr>
<tr>
<td>December 2010</td>
<td>2.1</td>
<td>Updated for 10.1 release.</td>
</tr>
<tr>
<td>July 2010</td>
<td>2.0</td>
<td>Updated for 10.0 release.</td>
</tr>
<tr>
<td>February 2010</td>
<td>1.1</td>
<td>Updated for 9.1 SP1 release.</td>
</tr>
<tr>
<td>November 2009</td>
<td>1.0</td>
<td>First published.</td>
</tr>
</tbody>
</table>

How to Contact Altera

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

<table>
<thead>
<tr>
<th>Contact (1)</th>
<th>Contact Method</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technical support</td>
<td>Website</td>
<td><a href="http://www.altera.com/support">www.altera.com/support</a></td>
</tr>
<tr>
<td>Technical training</td>
<td>Website</td>
<td><a href="http://www.altera.com/training">www.altera.com/training</a></td>
</tr>
<tr>
<td></td>
<td>Email</td>
<td><a href="mailto:custrain@altera.com">custrain@altera.com</a></td>
</tr>
<tr>
<td>Product literature</td>
<td>Website</td>
<td><a href="http://www.altera.com/literature">www.altera.com/literature</a></td>
</tr>
<tr>
<td>Non-technical support (General)</td>
<td>Email</td>
<td><a href="mailto:nacomp@altera.com">nacomp@altera.com</a></td>
</tr>
<tr>
<td>(Software Licensing)</td>
<td>Email</td>
<td><a href="mailto:authorization@altera.com">authorization@altera.com</a></td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Visual Cue</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bold Type with Initial Capital Letters</strong></td>
<td>Indicate command names, dialog box titles, dialog box options, and other GUI labels. For example, <strong>Save As</strong> dialog box. For GUI elements, capitalization matches the GUI.</td>
</tr>
<tr>
<td><strong>bold type</strong></td>
<td>Indicates directory names, project names, disk drive names, file names, file name extensions, software utility names, and GUI labels. For example, <code>\qdesigns</code> directory, <strong>D:</strong> drive, and <strong>chiptrip.gdf</strong> file.</td>
</tr>
<tr>
<td><strong>Italic Type with Initial Capital Letters</strong></td>
<td>Indicate document titles. For example, <em>Stratix IV Design Guidelines</em>.</td>
</tr>
<tr>
<td><strong>italic type</strong></td>
<td>Indicates variables. For example, <em>n + 1</em>. Variable names are enclosed in angle brackets (&lt;&gt;). For example, <code>&lt;file name&gt;</code> and <code>&lt;project name&gt;.pof</code> file.</td>
</tr>
<tr>
<td><strong>Initial Capital Letters</strong></td>
<td>Indicate keyboard keys and menu names. For example, the <strong>Delete</strong> key and the <strong>Options</strong> menu.</td>
</tr>
<tr>
<td>“Subheading Title”</td>
<td>Quotation marks indicate references to sections within a document and titles of Quartus II Help topics. For example, “Typographic Conventions.”</td>
</tr>
<tr>
<td><strong>Courier type</strong></td>
<td>Indicates signal, port, register, bit, block, and primitive names. For example, <strong>datal</strong>, <strong>tdi</strong>, and <strong>input</strong>. The suffix <code>n</code> denotes an active-low signal. For example, <strong>resetn</strong>. Indicates command line commands and anything that must be typed exactly as it appears. For example, <code>c:\qdesigns\tutorial\chiptrip.gdf</code>. Also indicates sections of an actual file, such as a Report File, references to parts of files (for example, the AHDL keyword <strong>SUBDESIGN</strong>), and logic function names (for example, <strong>TRI</strong>).</td>
</tr>
<tr>
<td><img src="arrow.png" alt="An angled arrow" /></td>
<td>An angled arrow instructs you to press the <strong>Enter</strong> key.</td>
</tr>
<tr>
<td>1., 2., 3., and a., b., c., and so on</td>
<td>Numbered steps indicate a list of items when the sequence of the items is important, such as the steps listed in a procedure.</td>
</tr>
<tr>
<td><img src="bullet.png" alt="Bullets" /></td>
<td>Bullets indicate a list of items when the sequence of the items is not important.</td>
</tr>
<tr>
<td><img src="hand.png" alt="The hand" /></td>
<td>The hand points to information that requires special attention.</td>
</tr>
<tr>
<td><img src="question_mark.png" alt="A question mark" /></td>
<td>A question mark directs you to a software help system with related information.</td>
</tr>
<tr>
<td><img src="feet.png" alt="The feet" /></td>
<td>The feet direct you to another document or website with related information.</td>
</tr>
<tr>
<td><img src="caution.png" alt="A caution" /></td>
<td>A caution calls attention to a condition or possible situation that can damage or destroy the product or your work.</td>
</tr>
<tr>
<td><img src="warning.png" alt="A warning" /></td>
<td>A warning calls attention to a condition or possible situation that can cause you injury.</td>
</tr>
<tr>
<td><img src="envelope.png" alt="The envelope" /></td>
<td>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.</td>
</tr>
</tbody>
</table>