

Introduction

The DDR/DDR2 SDRAM Controller MegaCore[®] function version 3.4.0 and later supports SOPC Builder, enabling the function to instantiate a DDR/DDR2 SDRAM Controller inside an SOPC Builder system.

This application note discusses the following topics:

- Implementing a DDR2 SDRAM Controller within SOPC Builder
- Incorporating a Nios[®] II processor and other peripherals
- Compiling the design and generating the programming file
- Downloading the design to a board and running sample code on your design in hardware to test the read and write transactions



The example design is targeted to the Cyclone[®] II device family. You can also use these procedures for other supported device families.

Requirements

This application note requires the following hardware and software:

- An Altera[®] development board with DDR or DDR2 SDRAM memory, such as a Cyclone II DSP development board
- DDR/DDR2 SDRAM Controller MegaCore function version 3.4.0 or later
- Quartus[®] II software version 6.0 or later



For more information about the Altera DDR/DDR2 memory controllers, refer to the *DDR & DDR2 SDRAM Controller Compiler User Guide*.

Setting Up the SOPC Builder Project

To create and run an SOPC Builder project, perform the following tasks:

- “Create a New Quartus II Project” on page 2
- “Create the SOPC Builder System” on page 4
- “Create the Top-Level Verilog HDL File” on page 22
- “Update the Quartus II Project Settings” on page 32
- “Set Up the SignalTap II Logic Analyzer” on page 33
- “Compile the Project” on page 39
- “Verify the Timing Results” on page 39

- “Download the SRAM Object File” on page 39
- “Incorporate Nios II IDE” on page 40
- “Test the System” on page 44

Create a New Quartus II Project

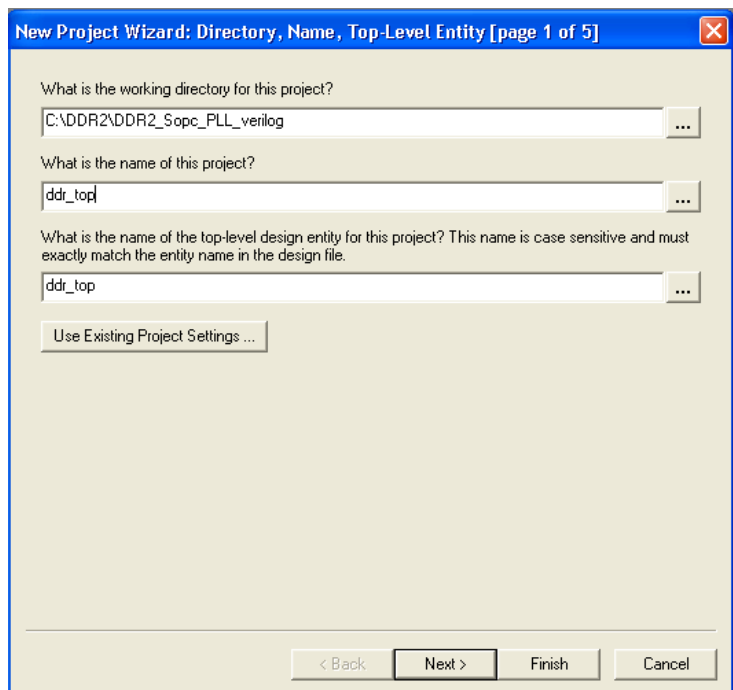
To create a new Quartus II project, follow these steps:



These instructions are for Windows-based systems. For Solaris and Linux-based systems the steps are similar, but refer to the *DDR & DDR2 SDRAM Controller Compiler User Guide* for details.

1. Start the Quartus II software.
2. On the File menu, click **New Project Wizard**.
3. Click **Next** in the introduction (the introduction will not display if you turned it off previously).
4. Specify the working directory for your project. This walkthrough uses the directory `c:\DDR2\DDR2_Sopc_PLL_verilog` (Figure 1).

Figure 1. New Project Wizard Dialog Box



5. Specify the name of the project. This walkthrough uses **ddr_top**.



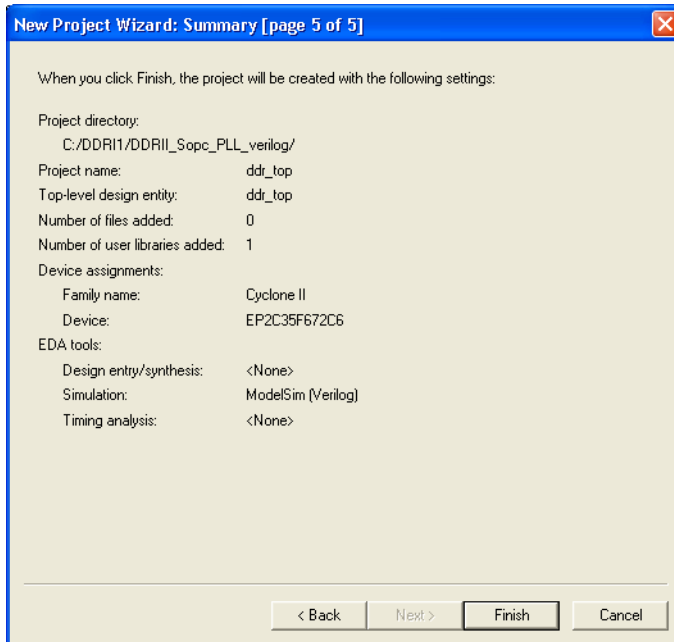
You must specify the same name for both the project and the top-level design entity.

6. Click **Next**.
7. Click **Next**.
8. In the **Family** list, select **Cyclone II**.



These instructions create a design targeting the Cyclone II device family. You can also use these procedures for other supported device families.

9. In the **Target device** box select **Specific device selected in 'Available devices' list**.
10. Under **Filters**, all fields should have the default value of **Any**.
11. In the **Available Devices** list, select **EP2C35F672C6**.
12. Click **Next**.
13. Click **Next**.
14. Check the Summary page to ensure that you have entered all the information correctly ([Figure 2](#)).

Figure 2. Summary Page

15. Click **Finish**.

You have finished creating your new Quartus II project.

Create the SOPC Builder System

In this section you will build a SOPC Builder system by adding the necessary components.

Launch SOPC Builder

To launch IP Toolbench from SOPC Builder, follow these steps:

1. On the Tools menu, click **SOPC Builder**.

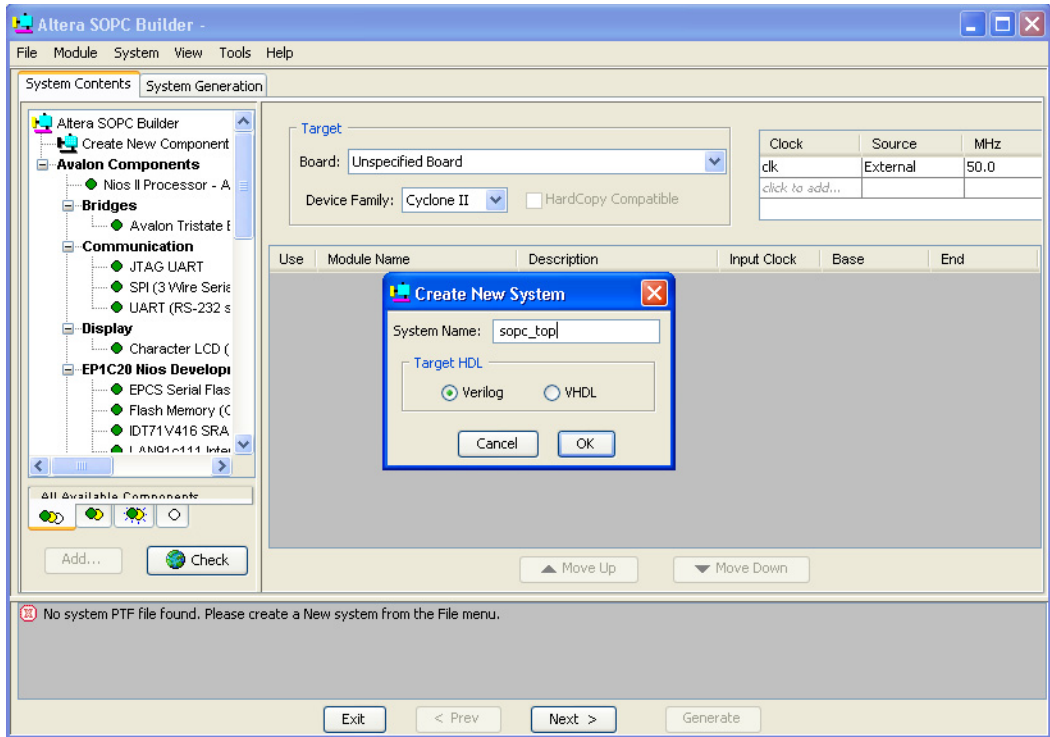
The **Altera SOPC Builder** dialog box appears.



Refer to Quartus II Help for more information on how to use SOPC Builder.

2. In the **Create New System** dialog box, type `sopc_top` in the **System Name** box and select **Verilog** under **Target HDL** (Figure 3).

Figure 3. SOPC Builder Dialog Box



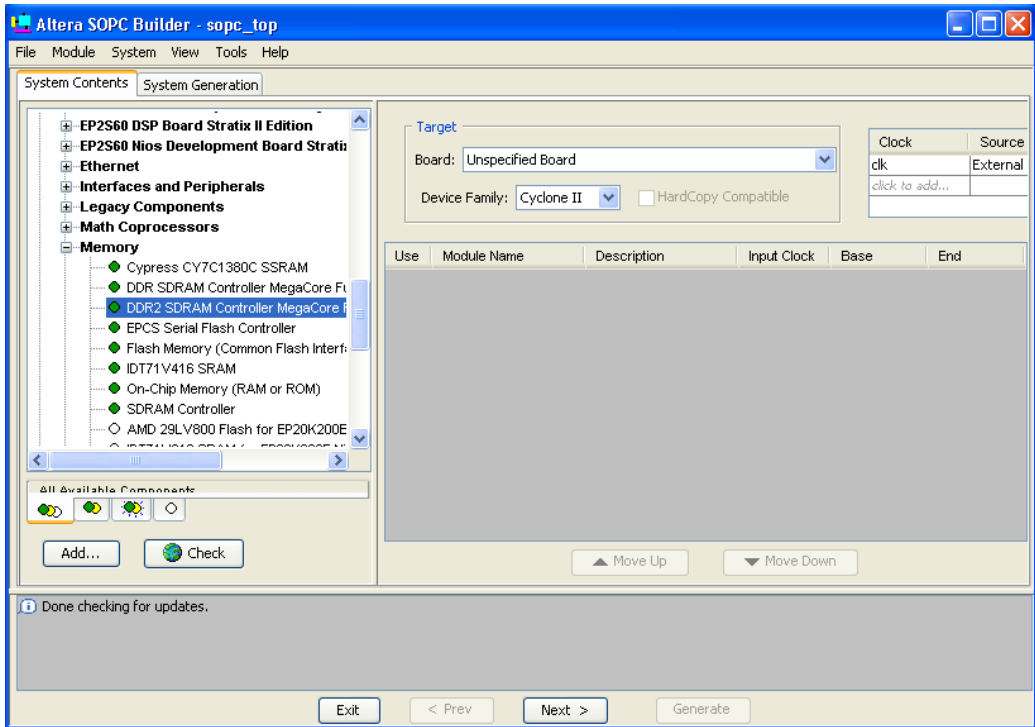
3. Click OK.

Add the DDR2 SDRAM Controller to the SOPC Builder System

Build your system by adding components from the System Contents tab.

1. Select **DDR2 SDRAM Controller MegaCore Function – Altera Corporation** in the **Memory** directory (Figure 4).

Figure 4. Select DDR2 SDRAM Controller in SOPC Builder



- Click **Add**. The DDR2 SDRAM Controller IP Toolbench appears (Figure 5).

Figure 5. DDR2 SDRAM Controller IP Toolbench



Step 1: Parameterize

To parameterize the DDR2 SDRAM Controller, follow these steps:

- Click **Step 1: Parameterize** to parameterize your custom variation. The Parameterize dialog box appears (Figure 6).
- In the Presets box select **Altera Cyclone II EP2C35 DSP Development Board**.



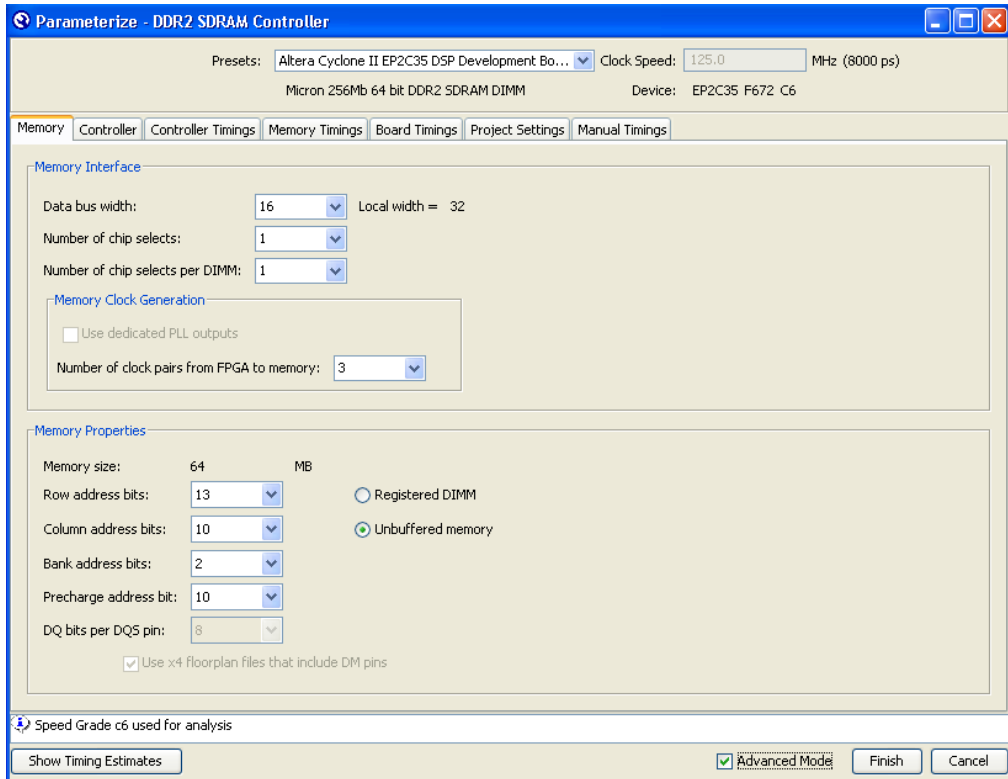
When you target an Altera board, all the settings on the Basic Settings tab and all **Advanced Mode** settings are correct for that board.

- Set **Data bus width** to 16 (the local width is automatically set to 32).



The local width is set to 32 bits to match the 32-bit Nios II processor.

Figure 6. Parameterize Dialog Box



Many other options are available for the memory properties. Turn on **Advanced Mode** and look at the available options. If you change the value of **Data Bus Width**, note that some of the values in the **Memory Properties** section also change to accommodate the new data bus width.

Turn off **Advanced Mode** when you are done.

4. Click **Finish**.

Step 2: Constraints

To set the constraints for your device, follow these steps:

1. Click **Step 2: Constraints** in IP Toolbench as shown in [Figure 7](#).

Figure 7. IP Toolbench—Constraints



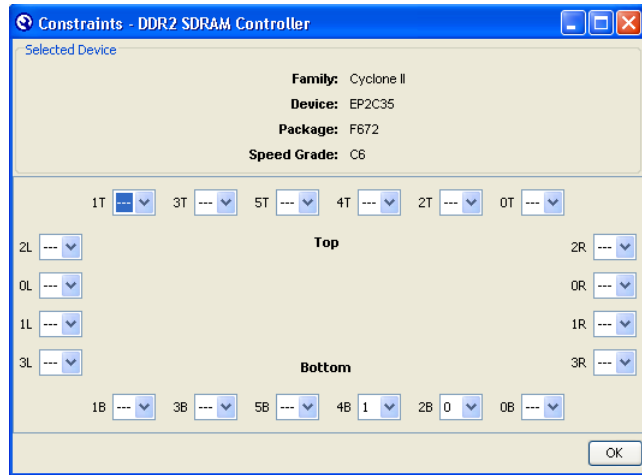
The Constraints dialog box opens ([Figure 8](#)).

2. In the Constraints dialog box, the data pins are assigned to particular banks on the Cyclone II device. In this example, the first eight dq bits (represented by “0”) are assigned to bank 2b and the next eight bits (represented by “1”) are assigned to bank 4b. These assignments are done automatically, because an Altera Cyclone II DSP Development board was selected as the target.



If you targeted an Altera board, all the constraint settings are already correct for that board.

Figure 8. Constraints Dialog Box



3. Click **OK**.

Step 3: Add/Update Component

To add or update the component and generate the system, follow these steps:

1. Click **Step 3: Add/Update Component** in IP Toolbench to add the custom variation to SOPC Builder, as shown in [Figure 9](#).

Figure 9. IP Toolbench—Add/Update Component

2. IP Toolbench closes and SOPC Builder uses the default module name (`ddr2_sdram_0`) for the variation name of your DDR2 SDRAM Controller.

Add Additional Components to the SOPC Builder System

Add the following components to the SOPC Builder system:

- **Nios II processor**—initiates the read and write transactions to the DDR2 memory
- **On-Chip Memory**—stores program data, read-only data, and other data
- **Parallel I/O**—controls the LEDs
- **JTAG UART**—sets the communication link between the PC and the FPGA through the download cable
- **PLL**—sets the input and output clock frequencies

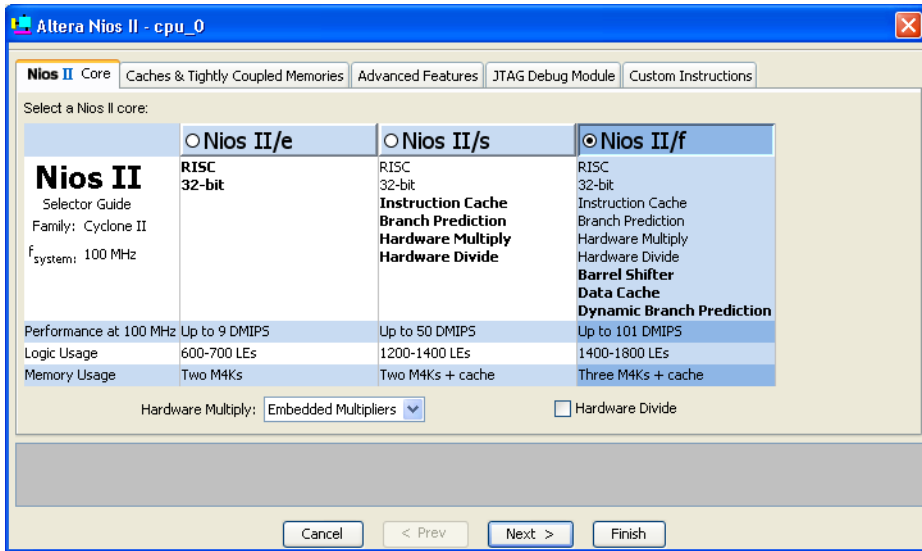
Add the Nios II Processor

To add the Nios II processor to your SOPC Builder system, perform the following steps:

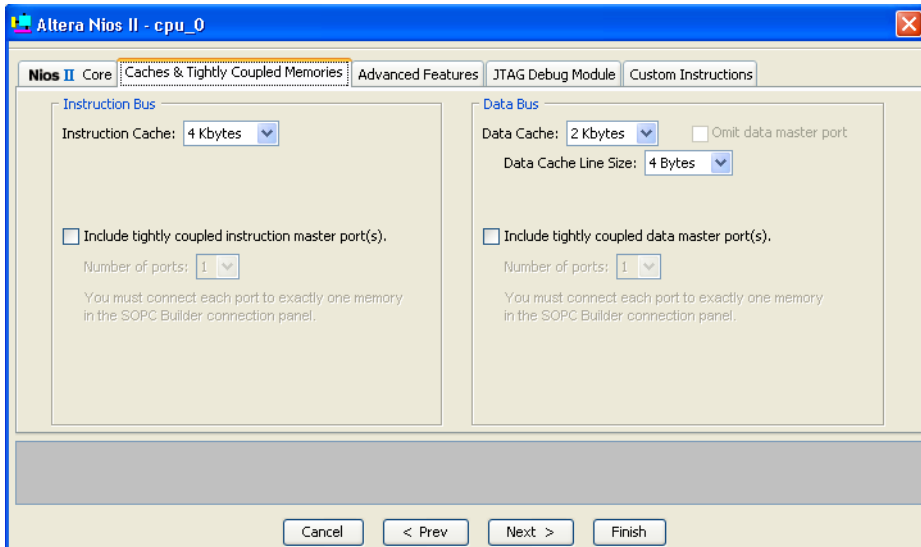
1. In the SOPC Builder dialog box, from the System Contents tab, select **Nios II Processor – Altera Corporation** in the **Avalon Components** directory.

- Click **Add**. The settings for the Altera Nios II processor are shown in [Figure 10](#).

Figure 10. Settings for the Nios II Processor



- Click **Next**. In [Figure 11](#) you parameterize the instruction and data cache.

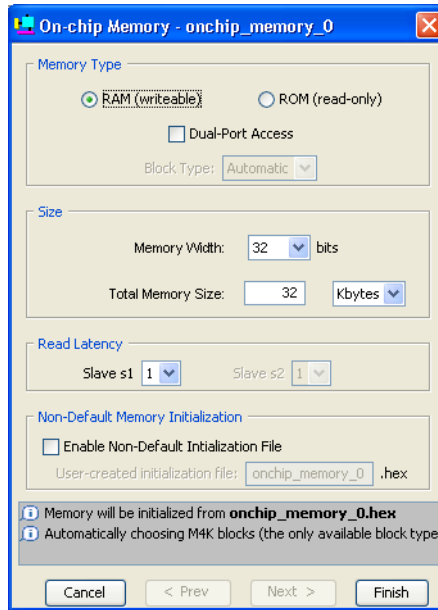
Figure 11. Instruction and Data Cache Parameterization

4. Click **Finish**. The Nios II Processor module is added to the SOPC Builder system.

Add the On-Chip Memory

To add the On-Chip Memory module to your SOPC Builder system, perform the following steps:

1. In the SOPC Builder dialog box, from the System Contents tab, select **On-Chip Memory (RAM or ROM)** in the **Memory** directory.
2. Click **Add**. The On-Chip Memory parameterization window appears (Figure 12).

Figure 12. On-Chip Memory Parameterization

3. Set Total Memory Size to **32 Kbytes**.

This size is large enough to hold both the program executable and the memory required for the read-only data memory and the read/write data memory (refer to [Figure 29 on page 44](#)).

4. On the System menu, click **Auto-Assign Base Addresses**.

Increasing the memory size to 32 Kbytes causes errors in the base address generated for the memory. **Auto-Assign Base Addresses** resolves the problem.

5. Click **Finish**. The On-Chip Memory module is added to the SOPC Builder system.

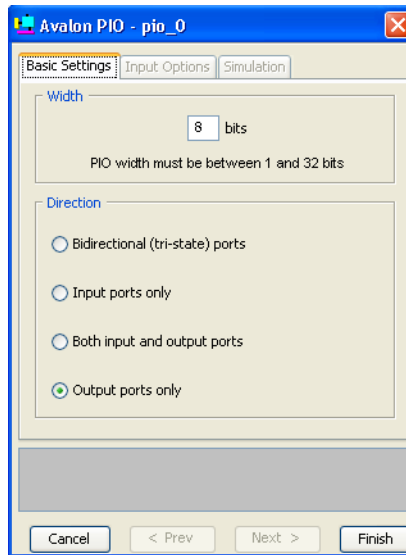
Add the Parallel I/O

To add the Parallel I/O (PIO) module to your SOPC Builder system, perform the following steps:

1. In the SOPC Builder dialog box, from the System Contents tab, select **PIO (Parallel I/O)** in the **Other** directory.

2. Click **Add**. The PIO selection window appears (Figure 13).

Figure 13. PIO Selection



3. Select **Output ports only**. The PIO only sends data to the LEDs. Nothing is read back from the LEDs.
4. Click **Finish**. The PIO module is added to the SOPC Builder system.

Add the JTAG UART

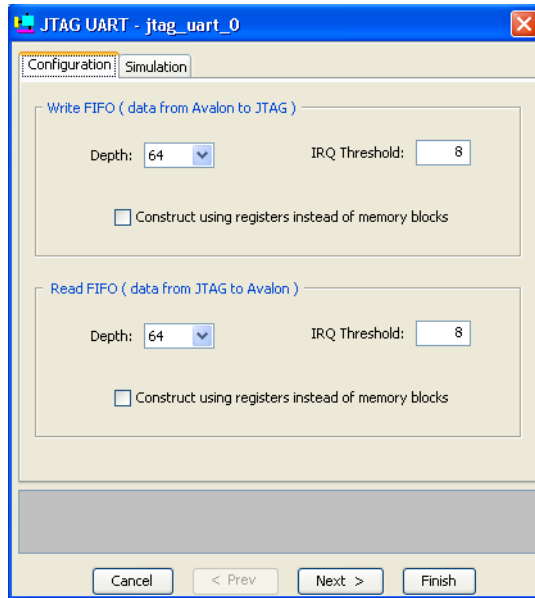
To add the JTAG UART module to your SOPC Builder system, perform the following steps:

1. In the SOPC Builder dialog box, from the System Contents tab, select **JTAG UART** in the **Communication** directory.
2. Click **Add**. The JTAG UART parameterization window appears (Figure 14).
3. Set **Depth** to 64 and **IRQ Threshold** to 8 for both the Write FIFO and the Read FIFO. These are the optimal values for this system.



For more information about these parameters, refer to the *JTAG UART Core with Avalon Interface* chapter in volume 5 of the *Quartus II Handbook*.

Figure 14. JTAG UART Parameters



4. Click **Finish**. The JTAG UART module is added to the SOPC Builder system.

Add the PLL

A Phase-Locked Loop (PLL) is generated automatically when the DDR2 SDRAM controller is generated. This example demonstrates how to add a PLL through SOPC Builder. This ability is useful when you have designed your own controller and want to add a PLL to the design.

The PLL that is generated with the DDR2 SDRAM controller in this example is removed by making the RTL modifications explained in [“Create the Top-Level Verilog HDL File” on page 22](#).

To add the PLL module to your SOPC Builder system, perform the following steps:

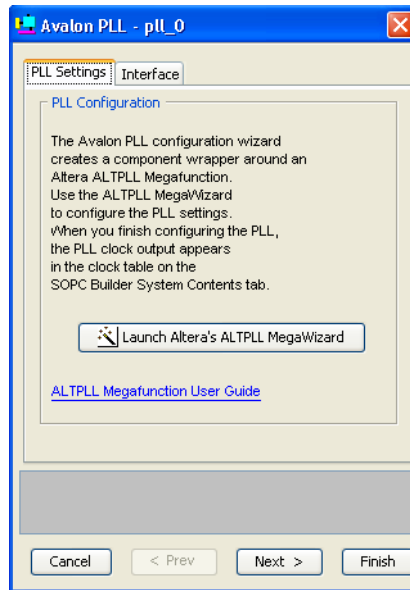
1. In the SOPC Builder dialog box, from the System Contents tab, select **PLL** in the **Other** directory.
2. Click **Add**. The PLL module is added to the SOPC Builder system.

The default input:output clock ratio for the PLL is 1:1. However, you must modify the input and output clock frequency settings of the PLL to meet the board requirements.

For the Cyclone II DSP board, perform the following steps to change the output clock:

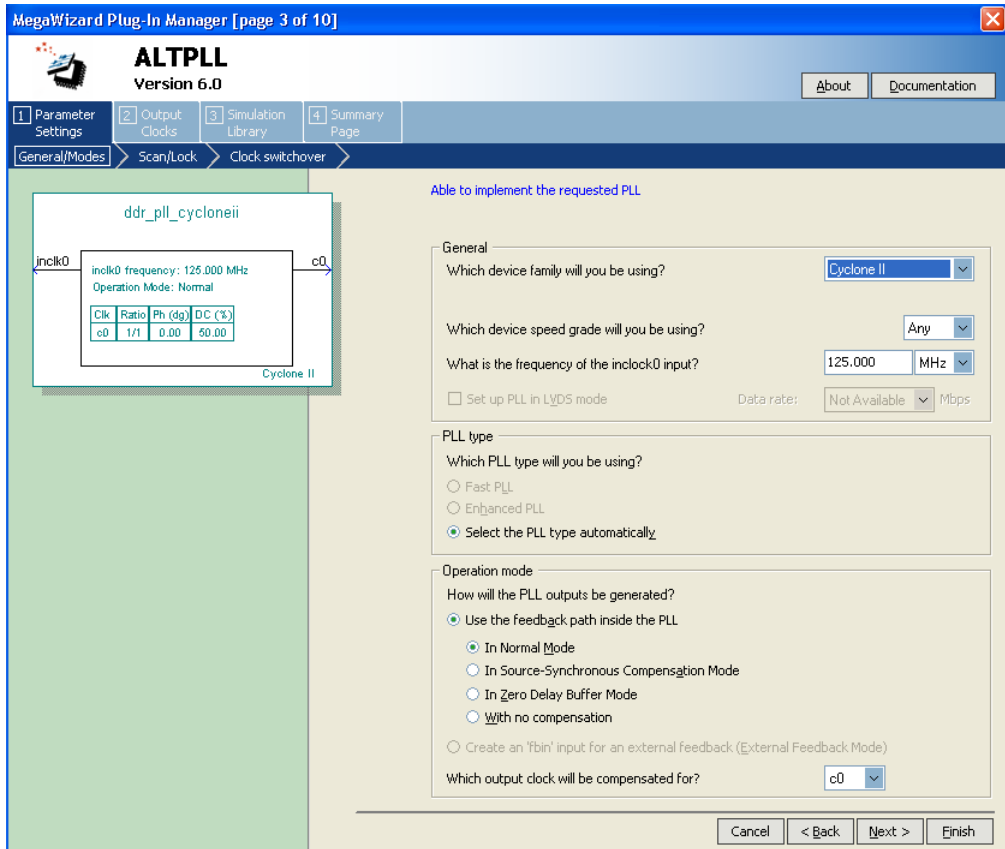
1. Double-click on the PLL in the SOPC Builder system. The PLL dialog box appears (Figure 15).

Figure 15. Launching the PLL



2. Click **Launch Altera's ALTPLL MegaWizard**.
3. In the MegaWizard® Plug-In Manager – ALTPLL [page 1of 8] dialog box, set the frequency of the `inclock0` input to 100 MHz (Figure 16).

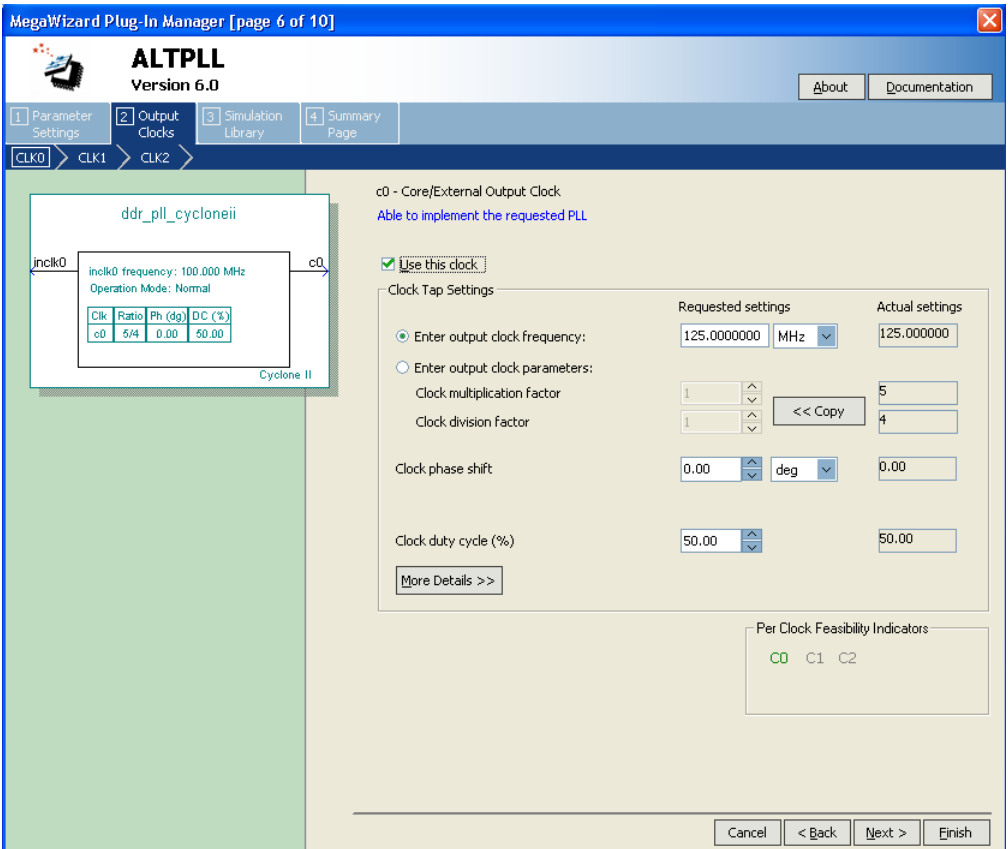
Figure 16. Setting the Input Clock Frequency



- Click **Next** three times.
- In the MegaWizard Plug-In Manager – ALTPLL [page 4 of 8] dialog box, set the frequency for output clock c0 (Figure 17). Clock c0 is used as both the system clock and the resync clock.

Set **Enter output clock frequency** to 125 MHz.

Figure 17. Setting the Output Clock c0

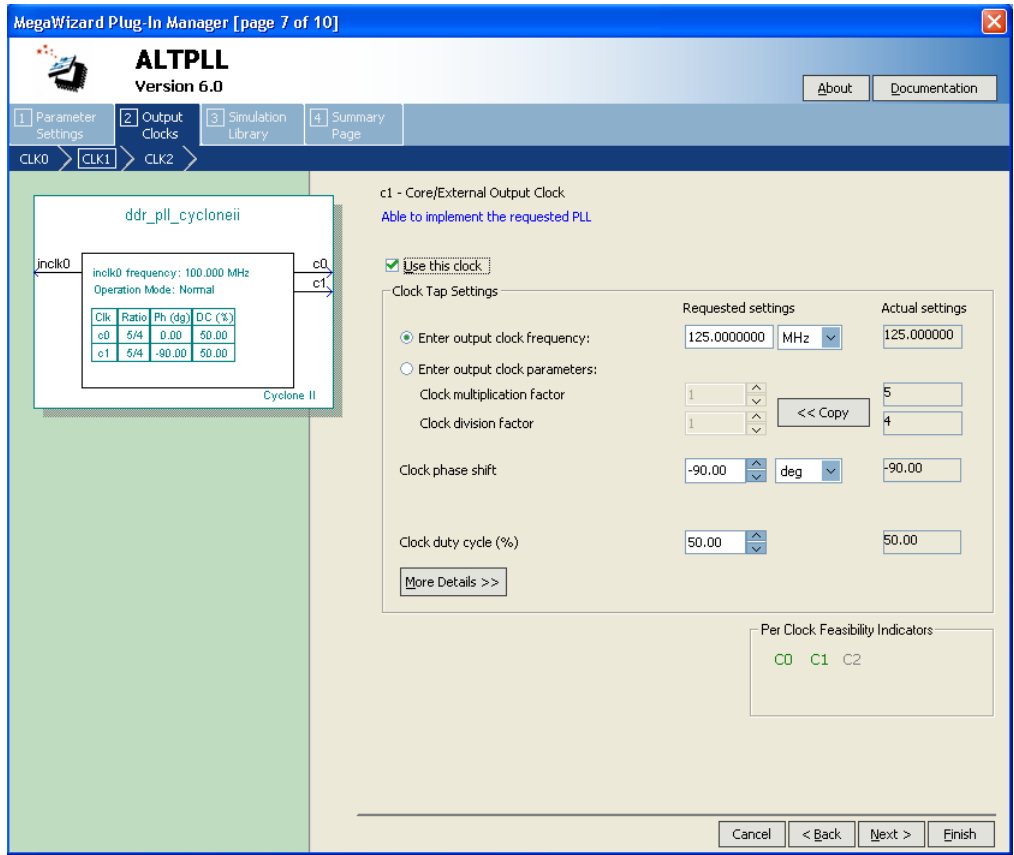


6. Click **Next**.
7. In the MegaWizard Plug-In Manager – ALTPLL [page 5 of 8] dialog box, set the parameters for output clock c1 (Figure 18). Clock c1 is used as the write clock to the memory.

Set **Enter output clock frequency** to 125 MHz.

Set **Clock phase shift** to -90 degrees.

Figure 18. Setting the Output Clock c1



8. Click **Next**.
9. Click **Finish**.
10. If a dialog box appears asking if you want to overwrite the current file, click **OK**.

If you rerun the DDR2 SDRAM Controller wizard, turn off the auto update PLL option under **Project Settings** or your PLL settings will be overwritten.

Complete the SOPC Builder System

First, set the bus links (refer to [Figure 19](#)).

1. Break the link between the `ddr2_sdram_0` and `cpu_0/instruction_master` modules.

In this example, the DDR2 SDRAM memory is only for data and does not contain any instruction code.

2. Create a link between the `ddr2_sdram_0` and `cpu_0/data_master` modules.
3. Create a link between the `pll_0` and `cpu_0/data_master` modules. (This is necessary because all slaves must be driven by at least one master.)
4. Set `clk_1` as the input clock to `ddr2_sdram_0`.

[Figure 19](#) shows the bus links and the input clock `clk_1`.

Figure 19. SOPC Builder Final System

Use	Module Name	Description	Input Clock	Base	End	IRQ
<input checked="" type="checkbox"/>	ddr2_sdram_0	DDR2 SDRAM Controller ...	clk_1	0x00000000	0x03FFFFFF	
<input checked="" type="checkbox"/>	cpu_0	Nios II Processor - Altera ...	clk			
	instruction_master	Master port				
	data_master	Master port				
	jtag_debug_module	Slave port				
<input checked="" type="checkbox"/>	onchip_memory_0	On-Chip Memory (RAM or ...)	clk	0x04008000	0x04007FFF	
<input checked="" type="checkbox"/>	pio_0	PIO (Parallel I/O)	clk	0x04008800	0x0400880F	
<input checked="" type="checkbox"/>	jtag_uart_0	JTAG UART	clk	0x04008810	0x04008817	0
<input checked="" type="checkbox"/>	pll_0	PLL (Phase-Locked Loop)	clk	0x04008820	0x0400883F	

Next, set the clock frequency to 125 MHz.

1. In SOPC Builder, change the clock frequency to 125 MHz.

A 125 MHz frequency is used because the f_{MIN} of the DDR2 SDRAM memory is 125 MHz. (SOPC Builder defaults to 50 MHz.)

2. Double-click the `ddr2_sdram_0` module. IP Toolbench appears.
3. Click **Step 1: Parameterize**.

In the Parameterization window, the clock speed has been automatically set to 125 MHz.

4. Click the **Project Settings** tab.
5. Turn off the **Update the example design PLLs** option.



If this option is turned on, the PLL settings are overwritten by the default settings.

6. Click **Finish**.
7. In IP Toolbench, click **Step 3: Add/Update Component**.

Generate the System

To generate the system, perform the following steps:

1. In SOPC Builder, click **Next**.
2. Click **Next**.
3. Turn off **Simulation. Create simulator project files**.
4. Click **Generate**.

The generation has finished when you get the message

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

5. Click **Exit**.

Create the Top-Level Verilog HDL File

This section describes the RTL modifications required for the Verilog HDL version of the project. These changes are necessary to update the top-level file generated by SOPC Builder (represented in [Figure 20](#)) to have the design hierarchy shown in [Figure 21](#). In the updated design, the example driver is replaced by the Nios II processor. In addition, the PLL that was generated with the DDR2 SDRAM core is removed. Instead, the PLL module that was added in SOPC Builder is used.

Similar changes are required for the VHDL version of the project.

When SOPC Builder generates the system, the following two files are created in your working directory, `c:\DDR2\DDR2_Sopc_PLL_verilog`.

- **ddr2_sdram_0_debug_design.v**—This top-level file does not contain the SOPC Builder component.
- **sopc_top.v**—This file contains the component declaration for your SOPC Builder system.

In this section, you create a top-level file (**ddr_top**) that contains the SOPC Builder component. This file, in turn, instantiates the PLL component. To create the top-level file, you merge the two files by replacing the example driver and the DDR/DDR2 SDRAM controller in **ddr2_sdram_0_debug_design.v** with the SOPC Builder components in **sopc_top.v**. [Figure 20](#) is a representation of the current file **ddr2_sdram_0_debug_design.v**. [Figure 21](#) is a representation of the new top-level file, called **ddr_top.v**.

Figure 20. Current File Representation

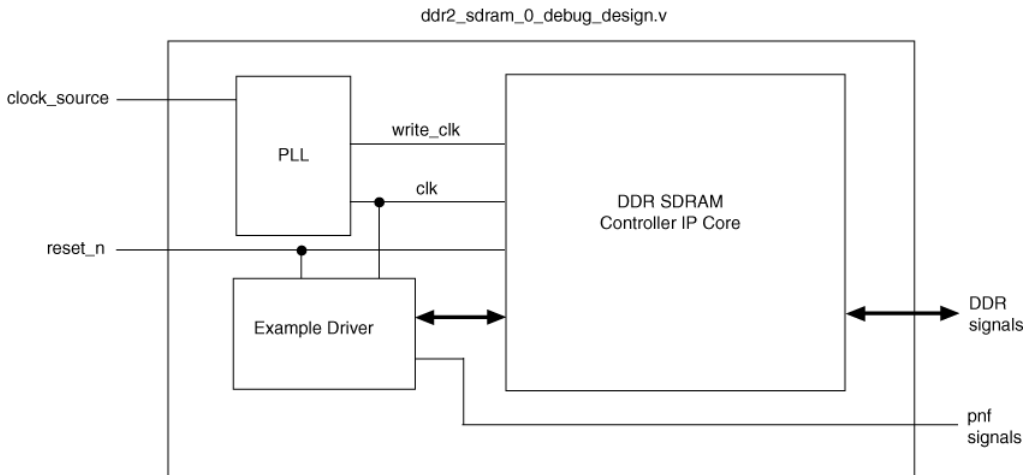
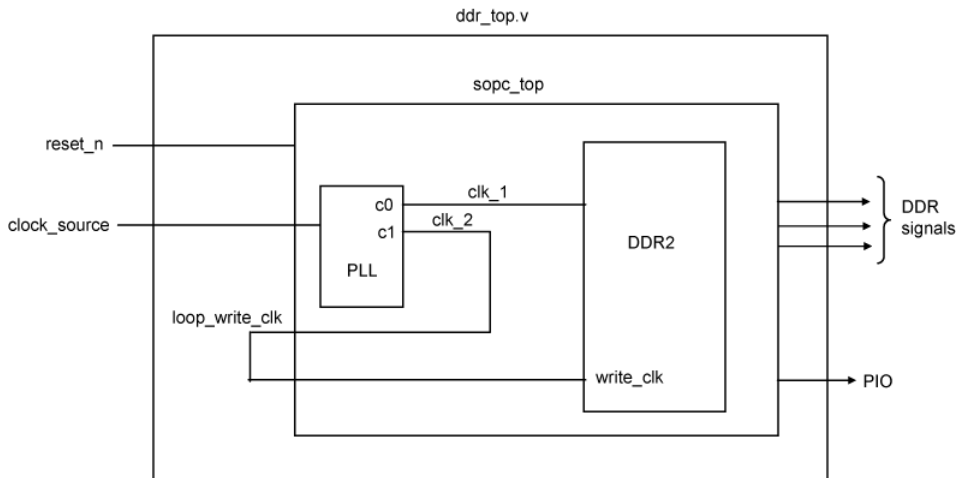


Figure 21. New Top-Level File Representation

To create the new top-level file, **ddr_top.v**, you can either copy the attached **ddr_top.v** file to your project location or make the changes manually by following these steps:

1. Open **ddr2_sdr2m_0_debug_design.v** in a text editor. Go to the first port definition ([Example 1](#)). This component defines the signals in the debug shell.

The following steps explain the changes needed to make the component compatible with SOPC Builder.

Example 1. Original Port Definition

```
module ddr2_sram_0_debug_design (
    // inputs:
    clock_source,
    reset_n,

    // outputs:
    clk_to_sram,
    clk_to_sram_n,
    ddr2_a,
    ddr2_ba,
    ddr2_cas_n,
    ddr2_cke,
    ddr2_cs_n,
    ddr2_dm,
    ddr2_dq,
    ddr2_dqs,
    ddr2_odt,
    ddr2_ras_n,
    ddr2_we_n,
    pnf,
    pnf_per_byte,
    test_complete
)
```

2. Change the name of the port definition from `ddr2_sram_0_debug_design` to your top-level name, `ddr_top`.
3. Insert the `pio_0` port. This port toggles the LEDs.
4. Remove the `pnf`, `pnf_per_byte`, and `test_complete` signals.

The revised definition is shown in [Example 2](#).

Example 2. Revised Port Definition

```
module ddr_top (
    // inputs:
    clock_source,
    reset_n,

    // outputs:
    clk_to_sdram,
    clk_to_sdram_n,
    ddr2_a,
    ddr2_ba,
    ddr2_cas_n,
    ddr2_cke,
    ddr2_cs_n,
    ddr2_dm,
    ddr2_dq,
    ddr2_dqs,
    ddr2_odt,
    ddr2_ras_n,
    ddr2_we_n,
    pio_0
)
```

Immediately following the port definitions are the port descriptions. Make the following changes to the port descriptions (refer to [Example 3](#)):

5. Remove the size description [0:0] on the ports `ddr2_cs_n`, `ddr2_cke`, and `ddr2_odt`.
6. Add the output port `pio_0` and size description [7:0].
7. Change the size of the port `ddr2_a` to 16 bits.

Example 3. Changes to Port Descriptions

```

output [ 2: 0] clk_to_sdram;
output [ 2: 0] clk_to_sdram_n;
output [ 15: 0] ddr2_a;
output [ 1: 0] ddr2_ba;
output          ddr2_cas_n;
output          ddr2_cke;
output          ddr2_cs_n;
output [ 1: 0] ddr2_dm;
inout  [ 15: 0] ddr2_dq;
inout  [ 1: 0] ddr2_dqs;
output          ddr2_odt;
output          ddr2_ras_n;
output          ddr2_we_n;
output [ 7: 0] pio_0;
input          clock_source;
input          reset_n;

```

8. Open **sopc_top.v**. Scroll to the bottom of the file, then scroll up slightly to the **sopc_top** instantiation (**Example 4**).

Example 4. Original SOPC_TOP Code

```

sopc_top DUT(
    .clk          (clk),
    .clk_1        (clk_1),
    .clk_2        (clk_2),
    .clk_3        (clk_3),
    .clk_to_sdram_from_the_ddr2_sdram_0 (clk_to_sdram_from_the_ddr2_sdram_0),
    .clk_to_sdram_n_from_the_ddr2_sdram_0 (clk_to_sdram_n_from_the_ddr2_sdram_0),
    .ddr2_a_from_the_ddr2_sdram_0 (ddr2_a_from_the_ddr2_sdram_0),
    .ddr2_ba_from_the_ddr2_sdram_0 (ddr2_ba_from_the_ddr2_sdram_0),
    .ddr2_cas_n_from_the_ddr2_sdram_0 (ddr2_cas_n_from_the_ddr2_sdram_0),
    .ddr2_cke_from_the_ddr2_sdram_0 (ddr2_cke_from_the_ddr2_sdram_0),
    .ddr2_cs_n_from_the_ddr2_sdram_0 (ddr2_cs_n_from_the_ddr2_sdram_0),
    .ddr2_dm_from_the_ddr2_sdram_0 (ddr2_dm_from_the_ddr2_sdram_0),
    .ddr2_dq_to_and_from_the_ddr2_sdram_0 (ddr2_dq_to_and_from_the_ddr2_sdram_0),
    .ddr2_dqs_to_and_from_the_ddr2_sdram_0 (ddr2_dqs_to_and_from_the_ddr2_sdram_0),
    .ddr2_odt_from_the_ddr2_sdram_0 (ddr2_odt_from_the_ddr2_sdram_0),
    .ddr2_ras_n_from_the_ddr2_sdram_0 (ddr2_ras_n_from_the_ddr2_sdram_0),
    .ddr2_we_n_from_the_ddr2_sdram_0 (ddr2_we_n_from_the_ddr2_sdram_0),
    .out_port_from_the_pio_0 (out_port_from_the_pio_0),
    .reset_n (reset_n),
    .write_clk_to_the_ddr2_sdram_0 (write_clk_to_the_ddr2_sdram_0)
);

```

9. Copy this **test_bench** instantiation into **ddr2_sdram_0_debug_design.v**, replacing the **ddr2_sdram_0** and **ddr2_sdram_0_example_driver** components.

10. Comment out `clk_1` and `clk_3`, because they are not used in the top-level file.
11. Change the signal name for `clk` to `clock_source`.
12. Add a signal `loop_write_clk` as a wire.
13. Connect the wire `loop_write_clk` from `clk_2` to `write_clk_to_the_ddr2_sdram_0`.
14. Change the signal names in the `sopc_top` instantiation to the shorter names shown in [Example 5](#).

Example 5. Revised SOPC_TOP Module

```
sopc_top DUT(  
    .clk                                (clock_source),  
    // .clk_1                            (clk_1),  
    .clk_2                              (loop_write_clk),  
    // .clk_3                            (clk_3),  
    .clk_to_sdram_from_the_ddr2_sdram_0 (clk_to_sdram),  
    .clk_to_sdram_n_from_the_ddr2_sdram_0 (clk_to_sdram_n),  
    .ddr2_a_from_the_ddr2_sdram_0      (ddr2_a [12:0]),  
    .ddr2_ba_from_the_ddr2_sdram_0     (ddr2_ba),  
    .ddr2_cas_n_from_the_ddr2_sdram_0  (ddr2_cas_n),  
    .ddr2_cke_from_the_ddr2_sdram_0    (ddr2_cke),  
    .ddr2_cs_n_from_the_ddr2_sdram_0   (ddr2_cs_n),  
    .ddr2_dm_from_the_ddr2_sdram_0     (ddr2_dm),  
    .ddr2_dq_to_and_from_the_ddr2_sdram_0 (ddr2_dq),  
    .ddr2_dqs_to_and_from_the_ddr2_sdram_0 (ddr2_dqs),  
    .ddr2_odt_from_the_ddr2_sdram_0    (ddr2_odt),  
    .ddr2_ras_n_from_the_ddr2_sdram_0  (ddr2_ras_n),  
    .ddr2_we_n_from_the_ddr2_sdram_0   (ddr2_we_n),  
    .out_port_from_the_pio_0           (pio_0),  
    .reset_n                           (reset_n),  
    .write_clk_to_the_ddr2_sdram_0     (loop_write_clk)  
);
```

15. Change the `test_bench` name in the architecture statement to `ddr_top`.
16. Comment out the PLL instantiation at the bottom of the file, because the PLL is part of the SOPC Builder system.
17. Immediately following the `sopc_top` instantiation in **`ddr2_sdram_0_debug_design.v`**, insert the following line:

```
assign ddr2_a [15:13] = 3'b000;
```

This statement sets the top bits of the `ddr2_a` register (that are not used in this design) to zero.

18. Rename the file to the top-level Quartus II project name, `ddr_top.v`, and save it. The new file is shown in [Example 6](#) (most of the comments have been left out).

Example 6. File ddr_top.v

//Legal Notice: (C)2006 Altera Corporation. All rights reserved.

```
`timescale 1ps / 1ps
```

```
//-----
module ddr_top (
    // inputs:
    clock_source,
    reset_n,

    // outputs:
    clk_to_sdram,
    clk_to_sdram_n,
    ddr2_a,
    ddr2_ba,
    ddr2_cas_n,
    ddr2_cke,
    ddr2_cs_n,
    ddr2_dm,
    ddr2_dq,
    ddr2_dqs,
    ddr2_odt,
    ddr2_ras_n,
    ddr2_we_n,
    pio_0
)
;

output [ 2: 0] clk_to_sdram;
output [ 2: 0] clk_to_sdram_n;
output [ 15: 0] ddr2_a;
output [ 1: 0] ddr2_ba;
output        ddr2_cas_n;
output ddr2_cke;
output ddr2_cs_n;
output [ 1: 0] ddr2_dm;
inout  [ 15: 0] ddr2_dq;
inout  [ 1: 0] ddr2_dqs;
output  ddr2_odt;
output        ddr2_ras_n;
output        ddr2_we_n;
output [ 7: 0] pio_0;
input    clock_source;
input    reset_n;

wire        clk;
wire [ 2: 0] clk_to_sdram;
wire [ 2: 0] clk_to_sdram_n;
wire [ 15: 0] ddr2_a;
wire [ 1: 0] ddr2_ba;
```

```

wire          ddr2_cas_n;
wire [ 0: 0] ddr2_cke;
wire [ 0: 0] ddr2_cs_n;
wire [ 1: 0] ddr2_dm;
wire [15: 0] ddr2_dq;
wire [ 1: 0] ddr2_dqs;
wire [23: 0] ddr2_local_addr;
wire [ 3: 0] ddr2_local_be;
wire          ddr2_local_burstbegin;
wire [ 9: 0] ddr2_local_col_addr;
wire          ddr2_local_cs_addr;
wire [31: 0] ddr2_local_rdata;
wire          ddr2_local_rdata_valid;
wire          ddr2_local_read_req;
wire          ddr2_local_ready;
wire          ddr2_local_refresh_req;
wire [ 1: 0] ddr2_local_size;
wire [31: 0] ddr2_local_wdata;
wire          ddr2_local_write_req;
wire [ 0: 0] ddr2_odt;
wire          ddr2_ras_n;
wire          ddr2_we_n;
wire          dedicated_resynch_or_capture_clk;
wire          pnf;
wire [ 3: 0] pnf_per_byte;
wire          test_complete;
wire          write_clk;
wire          loop_write_clk;

sopc_top DUT(
    .clk          (clock_source),
    // .clk_1      (clk_1),
    .clk_2        (loop_write_clk),
    // .clk_3      (clk_3),
    .clk_t0_s dram_from_the_ddr2_s dram_0 (clk_t0_s dram),
    .clk_to_s dram_n_from_the_ddr2_s dram_0 (clk_to_s dram_n),
    .ddr2_a_from_the_ddr2_s dram_0 (ddr2_a [12:0]),
    .ddr2_ba_from_the_ddr2_s dram_0 (ddr2_ba),
    .ddr2_cas_n_from_the_ddr2_s dram_0 (ddr2_cas_n),
    .ddr2_cke_from_the_ddr2_s dram_0 (ddr2_cke),
    .ddr2_cs_n_from_the_ddr2_s dram_0 (ddr2_cs_n),
    .ddr2_dm_from_the_ddr2_s dram_0 (ddr2_dm),
    .ddr2_dq_to_and_from_the_ddr2_s dram_0 (ddr2_dq),
    .ddr2_dqs_t0_and_from_the_ddr2_s dram_0 (ddr2_dqs),
    .ddr2_odt_from_the_ddr2_s dram_0 (ddr2_odt),
    .ddr2_ras_n_from_the_ddr2_s dram_0 (ddr2_ras_n),
    .ddr2_we_n_from_the_ddr2_s dram_0 (ddr2_we_n),
    .out_port_from_the_pio_0 (pio_0),
    .reset_n      (reset_n),
    .write_clk_to_the_ddr2_s dram_0 (loop_write_clk)
);

```

```
assign ddr2_a [15:13] = 3'b000;

//<< START MEGAWIZARD INSERT REFRESH_REQ
assign ddr2_local_refresh_req = 1'b0;

//connect up the column address bits
assign ddr2_local_addr[8 : 0] = ddr2_local_col_addr[9 : 1];

/*
//<< START MEGAWIZARD INSERT PLL
ddr_pll_cycloneii g_cyclonepll_ddr_pll_inst
(
    .c0 (clk),
    .c1 (write_clk),
    .c2 (dedicated_resynch_or_capture_clk),
    .inclk0 (clock_source)
);
*/

endmodule
```

Update the Quartus II Project Settings

Update the following Quartus II project settings to obtain optimal results.

First, for the best timing results, set the Fitter to **Standard** instead of **Auto** and set the Optimization Technique to **Speed** instead of **Balanced**. Perform the following steps:

1. On the Assignments menu, click **Settings**.
2. In the Category list, select **Fitter Settings**.
3. In the Fitter effort box, select **Standard Fit**.
4. In the Category list, select **Analysis & Synthesis Settings**.
5. In the Optimization Technique box, select **Speed**.
6. Click **OK**.

Next, set the unused pins to input tristate. If this is not done, the SignalTap® II logic analyzer may fail. Perform the following steps:

1. On the Assignments menu, click **Settings**.
2. In the Category list, select **Device**.
3. In the Device dialog box, click **Device & Pin Options**.

4. In the Unused Pins tab, select **As inputs, tri-stated**.
5. Click **OK**.
6. In the Device dialog box, click **OK**.

Set Up the SignalTap II Logic Analyzer

The SignalTap II logic analyzer is used to show read and write activity in the system.

Compile the Design

Compile the design. This step is necessary to select the nodes in the SignalTap II logic analyzer.

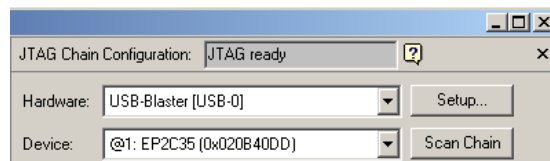
Install Hardware in the SignalTap II Logic Analyzer

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

In the Hardware field, select **USB-Blaster (USB-0)** from the drop-down menu (Figure 22).

If the necessary hardware driver is missing, go to the Altera website and download an appropriate driver.

Figure 22. Hardware Installation in the SignalTap II Dialog Box

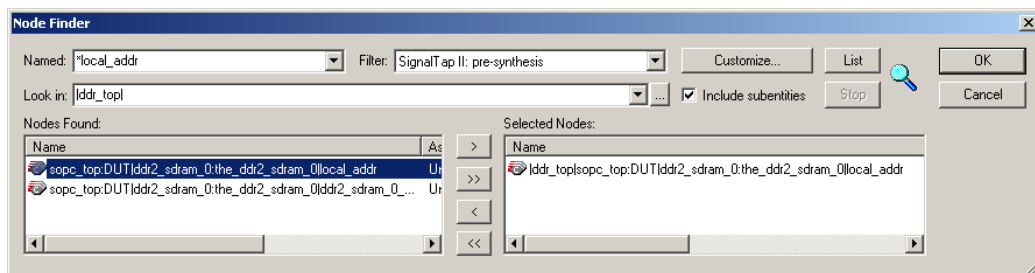


Add Signals in the SignalTap II Logic Analyzer


To add signals to the SignalTap II logic analyzer, follow these steps:

1. Double-click in the node selection box to bring up the Node Finder dialog box (Figure 23).

Figure 23. SignalTap II Node Finder



2. To add the `local_addr` signal, type `*local_addr` in the **Named** box and click **List**.
3. Select `sopc_top:DUT|ddr2_sdram_0|the_ddr2_sdram_0|local_addr` in the **Nodes Found** list and click **>** to add the signal to the **Selected Nodes** list (Figure 23).
4. Repeat steps 2-3 to add the following signals:
 - `local_rdata`
 - `local_rdata_valid`
 - `local_read_req`
 - `local_ready`
 - `local_wdata`
 - `local_write_req`

 For more information about these signals, refer to the *DDR and DDR2 SDRAM Controller Compiler User Guide*.

5. Click **OK**.
6. Turn off the trigger enables on the `local_addr`, `local_rdata`, and `local_wdata` signals. This reduces the trigger logic required for the SignalTap II logic analyzer.
7. Right-click the **Trigger Levels** cell for the `local_write_req` signal and select the **Rising Edge** trigger.



Do not add any of the `ddr` interface signals. Doing so will increase the load on those signals and adversely affect the timing analysis.

Figure 24 shows the completed node section for the SignalTap II logic analyzer. You will add a clock signal in the next step.

Figure 24. Signals Added to the SignalTap II Logic Analyzer

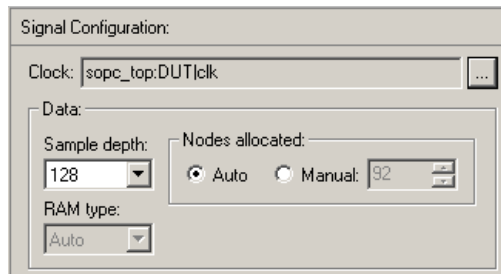
		Node		Incremental Route	Debug Port Out	Data Enable	Trigger Enable	Trigger Levels
Type	Alias	Name				92/Auto	4/Auto	1 Basic
		<input type="checkbox"/>	..._ddr2_sdram_0jlocal_addr	<input type="checkbox"/>		<input checked="" type="checkbox"/>	<input type="checkbox"/>	
		<input type="checkbox"/>	..._ddr2_sdram_0jlocal_rdata	<input type="checkbox"/>		<input checked="" type="checkbox"/>	<input type="checkbox"/>	
			...dr2_sdram_0jlocal_rdata_valid	<input type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
			...ddr2_sdram_0jlocal_read_req	<input type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
			...the_ddr2_sdram_0jlocal_ready	<input type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
		<input type="checkbox"/>	...dr2_sdram_0jlocal_wdata	<input type="checkbox"/>		<input checked="" type="checkbox"/>	<input type="checkbox"/>	
			...ddr2_sdram_0jlocal_write_req	<input type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	

Add the Clock Signal

To add a clock signal to the Signal Configuration box, perform the following steps:

1. Click the **Browse Node Finder** button in the Signal Configuration box to bring up the Node Finder dialog box (Figure 23).
2. Type `*clk` in the **Named** box and click **List**.
3. Select `sopc_top:DUT|clk` in the **Nodes Found** list and click **>** to add the signal to the **Selected Nodes** list.
4. Click **OK**.
5. Save the file.
6. If you get a message asking whether you want to enable SignalTap II file "stp1.stp" for the current project, click **Yes**.

Figure 25 shows the completed signal configuration section for the SignalTap II logic analyzer.

Figure 25. Signal Configuration Dialog Box

Source the Pin-Out File

On the Tools menu, click **Tcl Scripts**. The Tcl Scripts dialog box appears.

To source the Pin-out file, follow these steps:

1. In the Libraries box, scroll down to the C:/MegaCore/ddr_dds2_sdram-v3.4.0/lib/ directory and select **cycloneii_dsp_rev_a_pins**.
2. Click **Run**.

If the action is successful, the following message appears:

```
Info: Successfully loaded and ran Tcl Script File
"C:\MegaCore\ddr_dds2_sdram-v3.4.0\lib\
cycloneii_dsp_rev_a_pins.tcl"
```

Pin Changes

The Tcl script that was sourced in the previous section contains a number of pin assignments that are not used in this project. These pins are grayed out and have a question mark next to them in the Assignment Editor (refer to [Figure 26](#)). In this section, you delete several of these pin assignments and modify the names of several others.

On the Assignments menu, click **Assignment Editor**. The Assignment Editor dialog box appears.

To change the pin assignments, follow these steps:

1. In the Assignment Editor dialog box, click the **To** bar to arrange the pins alphabetically.

2. Delete the eight pins `user_dipsw[0]` through `user_dipsw[7]`. These pins are not used in the project.
3. Delete the pins that start with `dig_1` and `dig_2`, such as `dig_1_a` and `dig_2_a`.

The `ddr2_cke` and `ddr2_cs_n` pins must be renamed. [Figure 26](#) shows the current names.











Figure 26. Original Pin Names in Assignment Editor

	From	To ▲	Assignment Name	Value	Enabled
91		ddr2_cas_n	Output Pin Load	24	Yes
92		ddr2_cke[0]	Fast Output Register	On	Yes
93		ddr2_cke[0]	Location	PIN_AE21	Yes
94		ddr2_cke[0]	I/O Standard	SSTL-18 Class I	Yes
95		ddr2_cke[0]	Output Pin Load	24	Yes
96		ddr2_cke[1]	Location	PIN_AC19	Yes
97		ddr2_cs_n[0]	Fast Output Register	On	Yes
98		ddr2_cs_n[0]	Location	PIN_AF22	Yes
99		ddr2_cs_n[0]	I/O Standard	SSTL-18 Class I	Yes
100		ddr2_cs_n[0]	Output Pin Load	24	Yes
101		ddr2_cs_n[1]	Location	PIN_AB18	Yes
102		ddr2_dm	Output Enable Group	1	Yes

4. Change pin name `ddr2_cke[0]` to `ddr2_cke` for all instances of `ddr2_cke[0]`. Check that `ddr2_cke` is on location `PIN_AE21`.
5. Delete pin `ddr2_cke[1]`.
6. Change pin name `ddr2_cs_n[0]` to `ddr2_cs_n` for all instances of `ddr2_cs_n[0]`. Check that `ddr2_cs_n` is on location `PIN_AF22`.
7. Delete pin `ddr2_cs_n[1]`.

[Figure 27](#) shows the new names of these pins.

Figure 27. Pin Changes in Assignment Editor

	From	To ▲	Assignment Name	Value	Enabled
91		 ddr2_cas_n	Output Pin Load	24	Yes
92		 ddr2_cke	Fast Output Register	On	Yes
93		 ddr2_cke	Location	PIN_AE21	Yes
94		 ddr2_cke	I/O Standard	SSTL-18 Class I	Yes
95		 ddr2_cke	Output Pin Load	24	Yes
96		 ddr2_cs_n	Fast Output Register	On	Yes
97		 ddr2_cs_n	Location	PIN_AF22	Yes
98		 ddr2_cs_n	I/O Standard	SSTL-18 Class I	Yes
99		 ddr2_cs_n	Output Pin Load	24	Yes
100		 ddr2_dm	Output Enable Group	1	Yes

8. Change pin name `ddr2_odt[0]` to `ddr2_odt` for all instances of `ddr2_odt[0]`. Check that `ddr2_odt` is on location `PIN_AF21`.
9. Delete pin `ddr2_odt[1]`.
10. Create three new assignments, one each for pins `ddr2_a[13]`, `ddr2_a[14]`, and `ddr2_a[15]`.
 - a. Double-click on the <<new>> field at the bottom of the **To** column.
 - b. On the drop-down menu, click **Node Finder**.
 - c. In the Node Finder dialog box, type `ddr2_a*` in the **Named** box and click **List**.
 - d. Select `ddr2_a[13]`, `ddr2_a[14]`, and `ddr2_a[15]` in the **Nodes Found** list and click > to add the pins to the **Selected Nodes** list.
 - e. Click **OK**.
11. For each of the new assignments, set the Assignment Name field and the Value field. To set a field, double-click on the cell. Select **I/O Standard** from the Assignment Name drop-down menu and **SSTL-18 Class I** from the Value drop-down menu.
12. Change pin name `user_led[0]` to `pio_0[0]`. Change pin name `user_led[1]` to `pio_0[1]` and so on through `user_led[7]`.
13. Save the file.

Compile the Project

On the Processing menu, click **Start Compilation** to compile the project.

Verify the Timing Results

Any failed timings will be highlighted in blue in the System tab of the Quartus II message window.

Download the SRAM Object File

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

Make sure the correct **.sof** file is installed. The SOF Manager box should contain the file **ddr_top.sof**. If it doesn't, perform the following steps:


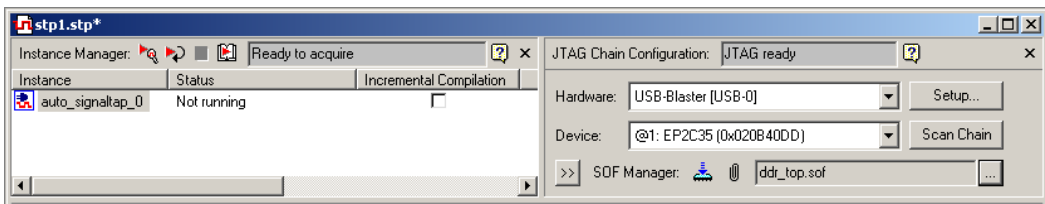
1. Open the Browse Program Files dialog box.
2. Select **ddr_top.sof**.
3. Click **Open**.
4. To download the file, click the **Program Device** button:  (Figure 28).

Figure 28. Installing the SRAM Object File In the SignalTap II Dialog Box



Incorporate Nios II IDE

The final part of this walkthrough consists of adding a Nios II processor to the project and running a simple program on it.



When doing memory tests, you must read and write from the external DDR memory and not from cached memory. There are three ways to avoid using cached memory:

- Use a Nios II processor that doesn't have cache memory
- Use the `alt_remap_uncached` function
- Set your addresses beyond 31 bits; for example, `0x80000000`

Launch Nios II IDE

To launch the Nios II IDE, follow these steps:

1. On the Tools menu, click **SOPC Builder**.
2. Click the **System Generation** tab.
3. Click **Run Nios II IDE**.
4. In the Workspace Launcher dialog box, set the Workspace to `c:\temp` and click **OK**.



Make sure the Nios II IDE and SOPC Builder have the same version. Use the Help menu to check the version level.

5. On the File menu, point to New and click **C/C++ Application**. The New Project dialog box appears.
6. In the Select Project Template box, select the **Hello World** project.
7. Click **Finish**.

Implement the Test Code

To implement the test code, follow these steps:

1. Replace the code in the `hello_world.c` program with the program shown in [Example 7](#).
2. Save the `hello_world.c` file.

Example 7. New hello_world.c File

```

#include <stdio.h>
#include "sys/alt_dma.h"
#include "sys/alt_cache.h"

int to_hex(char* pkt)
{
    int value[8];
    int value1;
    int q;
    for (q=0; q <= 7; q++)
    {
        value[q] = (pkt[q] > 0x39) ? (pkt[q] - 0x37) : (pkt[q] - 0x30);
        if (q == 0)
            value1 = (0 + value[q]);
        else
            value1 = ((value[q-1] << 4) + value[q]);
    }
    return value1;
}

int main()
{
    unsigned int *led_address;
    unsigned long *DDR_address; //long is 32 bits.
    int led_value;
    int addr;
    int datar;
    int ddr_data_out;

    char packet[32];

    DDR_address = (unsigned long *)0x80000000; //make non-cache
    led_address = (unsigned int *)0x84008800; //make non-cache

    iprintf("DDR test installed \n");

    while (1)
    {
        gets(packet);

        switch(packet[0])
        {
            case 'A':
                led_value = to_hex(&packet[1]);
                iprintf("LED test %x \n", led_value);
                *led_address = led_value;
                break;

            case 'B': // write operation

```

```
    addr = to_hex(&packet[1]);
    datar = to_hex(&packet[9]);
    DDR_address[addr] = datar;
    iprintf("Written %08x to address %08x \n", datar, addr);
    break;

case 'C':          // Read operation
    addr = to_hex(&packet[1]);
    ddr_data_out = DDR_address[addr];
    iprintf("Read %08x from address %08x \n", ddr_data_out, addr);
    break;

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

}

return 0;
}
```

This program consists of a simple loop that takes commands from the JTAG UART and executes them. The commands that are sent to the Nios/C code must be in one of the following three formats:

- Switch A controls the LEDs. The command is in the form

Switch | Data

For example, `A000000FF`. The last two bytes control all eight LEDs. The LEDs are active low on the board.

- Switch B writes to the DDR/DDR2 SDRAM memory. The command is in the form

Switch | Address | Data

For example, `B0000000000000001` stores `00000001` in memory address `00000000`.

- Switch C reads from the DDR/DDR2 SDRAM memory. The command is in the form

Switch | Address

For example, `C00000000` reads the contents of memory address `00000000`.



The switches A, B, and C must be entered in upper case.

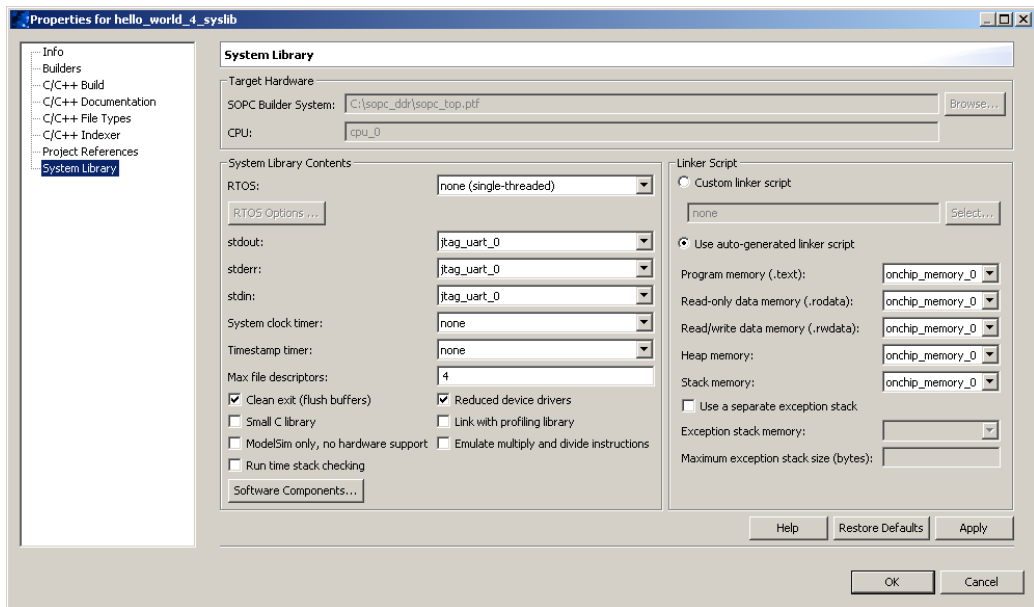
Set Up the Project Settings

To set up the Nios II project settings, follow these steps:

1. Right-click **hello_world_0** in the C/C++ Projects tab and click **System Library Properties** from the menu.
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, in the System Library Contents box, turn on **Reduced device drivers**.
4. In the Linker Script box, set all of the memory options to **onchip_memory_0**. The on-chip memory is the target memory space for the executable.
5. To reduce the memory size allocated for the system library, set **Max file descriptors** to 4.

[Figure 29](#) shows the Properties dialog box with the selected options.

Figure 29. Nios II Project Settings



6. Click **OK**.
7. On the Project menu, click **Build Project**.

Run the Nios II Project

On the Run menu, point to Run As and click **Nios II Hardware**.

Test the System

Perform the following tests to make sure the system is set up properly.

Write to an LED

Type the following in the command console:

```
A00000FE
```

Because the LEDs are active low on the Cyclone II DSP board, all but one of the LEDs should turn off.

The following message should appear in the command console:

```
LED test fe
```

Repeat the test with the following command:

```
A000000AA
```

Every other LED should turn off.

Write to the DDR/DDR2 SDRAM Memory

Perform the following steps to test the DDR/DDR2 SDRAM memory:

1. To see what is in memory address 00000000, type the following in the Nios II command console:

```
C00000000
```

2. Next, write to the same address:

```
B000000000000000001
```

This command stores a 1 in memory address 00000000.

3. Finally, read back the contents of memory address 00000000 and make sure the 1 is there. Enter the following command:

```
C00000000
```

Capturing Activity With the SignalTap II Logic Analyzer

Use the SignalTap II logic analyzer to show read and write activity. To show write activity, perform the following steps:

1. In the SignalTap II window, set the trigger for the `local_write_req` signal to rise on a write request (Figure 30).

Figure 30. Set Trigger for local_write_req Signal



2. Click the **Run Analysis** button to capture the write request once:

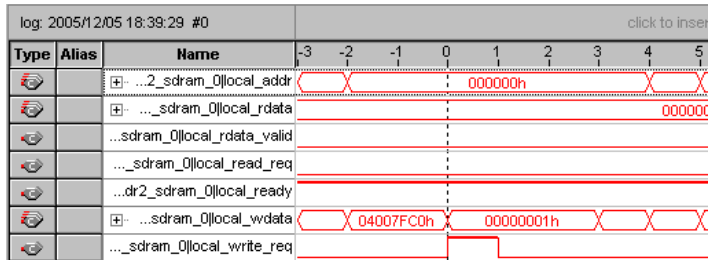


3. Type the following in the Nios II command console:

```
B0000000000000001
```

4. Return to the SignalTap II window and check that the `local_write_req` signal shows a rise in the signal at time 0. The `local_wdata` signal receives a value of 1 at time 0 (Figure 31).

Figure 31. Waveforms for Write Signals



To show read activity, perform the following steps:

1. Set the trigger for the `local_write_req` signal to **Don't care**.
2. Set the trigger for the `local_read_req` signal to rise on a read request (Figure 32).

Figure 32. Set Trigger for local_read_req Signal

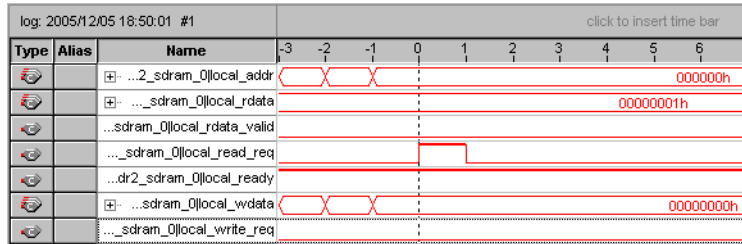


3. Click the **Run Analysis** button to capture the read request once.
4. Type the following in the Nios II command console:

```
C00000000
```

5. Return to the SignalTap II window and check that the `local_read_req` signal shows a rise in the signal at time 0. The `local_rdata` signal receives a value of 1 at time 0 (Figure 33).

Figure 33. Waveforms for Read Signals



101 Innovation Drive
 San Jose, CA 95134
 (408) 544-7000
www.altera.com
Applications Hotline:
 (800) 800-EPLD
Literature Services:
literature@altera.com

Copyright © 2006 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, maskwork rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.



I.S. EN ISO 9001