

## Introduction

The increasing pressure to deliver robust products to market in a timely manner has amplified the importance of comprehensively verifying embedded processor designs. Therefore, when choosing an embedded processor a key consideration is the verification solution supplied with the processor. Nios<sup>®</sup> II embedded processor designs support a broad range of verification solutions including:

- *Board Level Verification* — Altera offers a number of development boards that provide a versatile platform for verifying both the hardware and software of a Nios II embedded processor system. The Nios II integrated development environment (IDE) can be used to verify designs running on development or custom boards using its built-in debugger. You can find more information on the Nios II IDE debugger in the Nios II IDE online help. Hardware components that interact with the processor can further be debugged with the Signal Tap II embedded logic analyzer. For more information on Signal Tap II, see *AN 323: Using Signal Tap II Embedded Logic Analyzer in SOPC Builder Systems*.
- *Instruction Set Simulator (ISS)* — An instruction set simulator is used to model the Nios II processors instruction set in a software based simulation model. This allows designers to run the executable image from their software project on the ISS and to debug the software using the Nios II IDE debugger. The ISS is particularly useful if a development board is not available. The ISS is principally used for software debugging purposes; however, it is also capable of modeling a subset of the components available in the SOPC Builder. For more information on the Nios II ISS see the online help in the Nios II IDE.
- *Register Transfer Level (RTL) Simulation* — RTL simulation is a powerful means of debugging the interaction between a processor and its peripheral set. When debugging a target board it can often be difficult to view signals buried deep within the system. RTL simulation alleviates this problem as it enables designers to functionally probe every register and signal within the design. Nios II-based systems are easily simulated in ModelSim<sup>®</sup> using an automatically generated simulation environment created by SOPC Builder and the Nios II IDE.

This application note describes the process of generating an RTL simulation environment using Nios II example designs, SOPC Builder, and the Nios II IDE. It also describes the process of running the Nios II RTL simulation within ModelSim.

## Before You Begin

This document assumes that you have prior experience working with SOPC Builder as well as a familiarity with the ModelSim simulator. In addition if you wish to simulate a Nios II based system you should have the following software installed:

- Quartus® II version 4.0 Service Pack 1 or higher
- ModelSim Altera 5.7e or higher, or ModelSim PE, SE, EE



If you are using ModelSim Altera 5.7e with Quartus II version 4.0 SP1, you must install the ModelSim Altera pre-compiled libraries for SP1 available at:  
[www.altera.com/support/software/download/service\\_packs/quartus/dnl-qii40sp1.jsp](http://www.altera.com/support/software/download/service_packs/quartus/dnl-qii40sp1.jsp)

- Nios II embedded processor version 1.0. If you wish to simulate a first-generation Nios processor please refer to *AN 189: Simulating Nios Embedded Processor Designs*

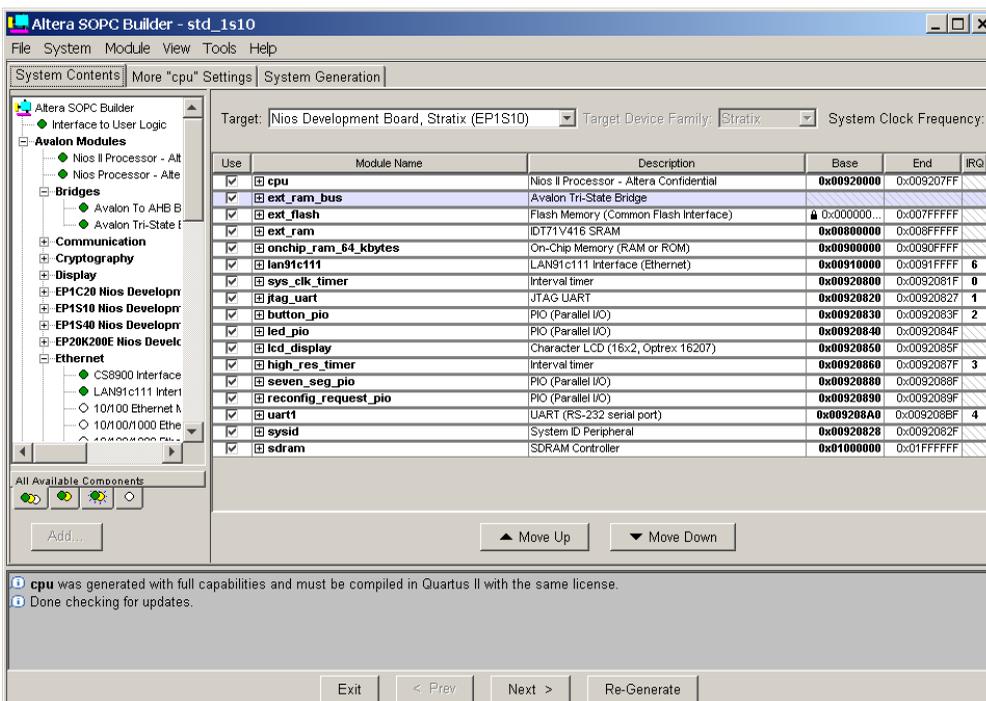
## Setting Up Your Simulation Environment in SOPC Builder

To open the example design, perform the following steps:

1. Depending on which HDL you use, do one of the following
  - a. If VHDL is your primary HDL, locate the *<Nios II install directory>/examples/vhdl/ niosII\_stratix\_1s10/standard* directory and copy this folder into a location where you plan to test the Nios II simulation flow. The location where you copy the folder will be referred to as *<your project directory>* throughout the remainder of this document.
  - b. If Verilog is your HDL then locate the *<Nios II install directory>/examples/verilog/ niosII\_stratix\_1s10/standard* directory and copy this folder into a location where you wish to test the Nios II simulation flow. The location where you copy the folder will be referred to as *<your project directory>* throughout the remainder of this document.
2. Run the Quartus II software

3. Choose **Open Project** (File menu)
4. Browse to < *your project directory* > \standard
5. Select **standard.qpf**
6. Click **Open**.
7. Choose **SOPC Builder** (Tools menu) in the Quartus II software. SOPC Builder software will open and resemble [Figure 1](#).

**Figure 1. Example Nios II Design**



### Memory Initialization

8. In order to run a simulation of a Nios II design, it is necessary that any memories containing software code are initialized prior to simulation. Previous generations of the Nios processor generated initialization files for memories using SOPC Builder. However, Nios II based-systems require a different technique for memory initialization. With Nios II-based systems, memory initialization

files are created in the Nios II IDE rather than SOPC Builder. This is because the Nios II IDE is used to create and compile software projects and the resulting memory initialization files. Later sections of this document explain how to use the Nios II IDE to create memory initialization files for your simulation.

### *UART Settings*

SOPC Builder can be used to customize a JTAG UART for generating a data stream to send to the host processor during simulation. There are two possible ways for the UART to send data to the Nios II processor:

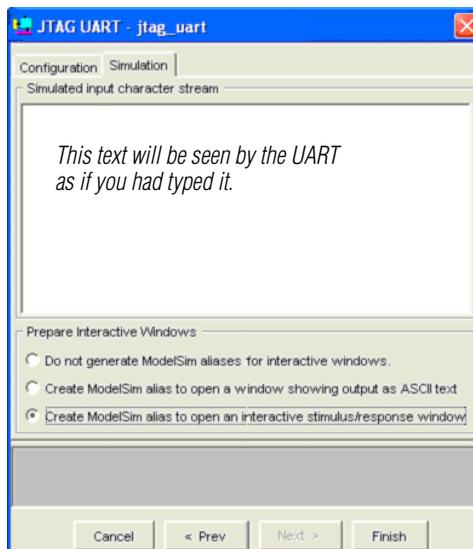
1. Using the **Simulated input character** stream dialog box.
2. Creating an **Interactive stimulus/response** window. For more information on setting up a UART for simulation, See the “UART Core with Avalon Interface” chapter in the *Nios II Processor Handbook*.

To set up the JTAG UART for our simulation, perform the following steps in the **System Contents** tab of SOPC Builder:

1. Double click the **jtag\_uart** peripheral. The JTAG UART dialog box opens.
2. Select the **Simulation** tab.
3. Turn on the **Create ModelSim alias to open an interactive stimulus/response** window.
4. Click **Finish**.

Figure 2 on page 5 shows the example JTAG UART settings.

---

**Figure 2. UART Simulation Settings**


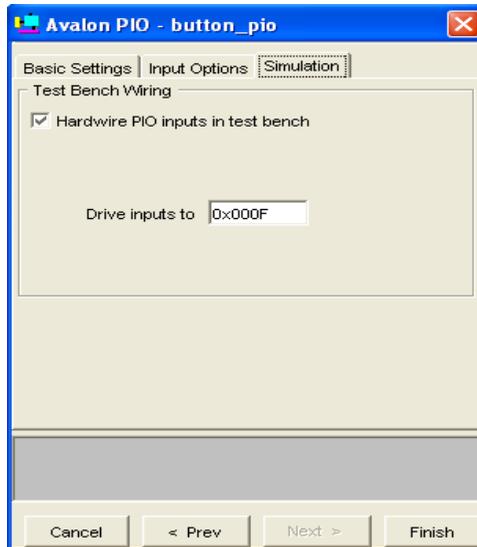

---

### *PIO Settings*

SOPC Builder can also be used to initialize the inputs of any PIO peripherals in your design which have an input port. For example, PIOs with the direction set to bidirectional (tri-state) ports, input ports only, or both input and output ports can be initialized for simulation in SOPC Builder. For more information, see the “PIO Cores with Avalon Interface” in the *Nios II Processor Handbook*. To initialize the PIO in the example design, perform the following steps:

1. Double-click **button\_pio**.
2. Select the **Simulation** tab. See [Figure 3 on page 6](#).
3. Enter the initial value that you wish to drive on the input ports.
4. Click **Finish**.

**Figure 3. PIO Simulation Settings**

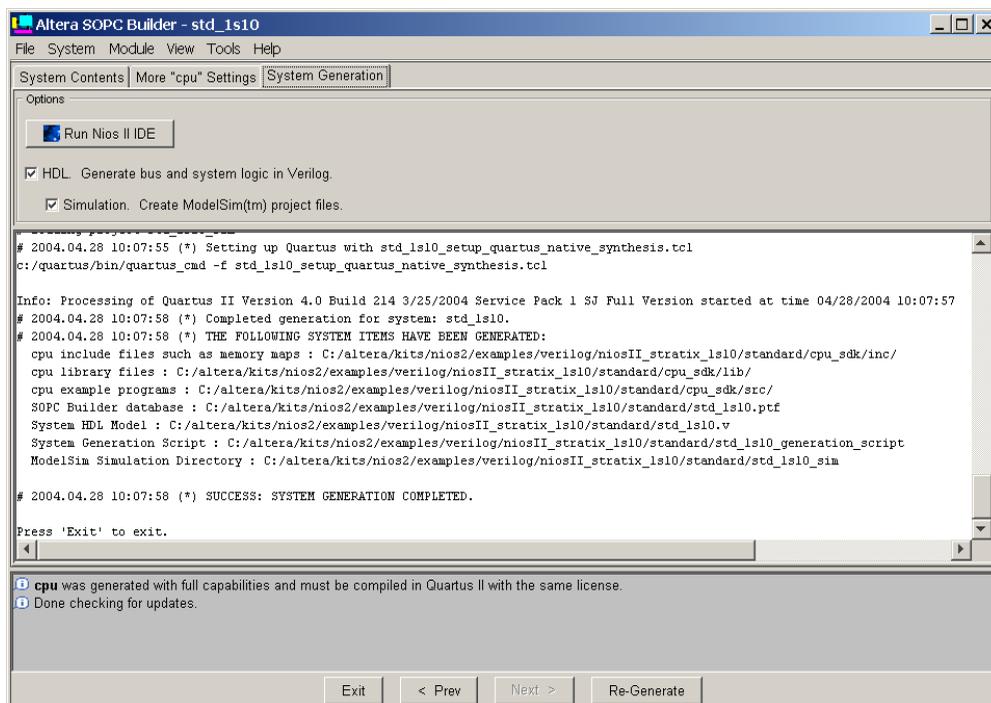


### *SOPC Builder Simulation Settings*

After you have made simulation settings to any UARTs and PIOs in your design, it is important to enable simulation file generation by performing the following steps in SOPC Builder:

1. Set up the path to the ModelSim software
2. Choose **SOPC Builder Setup > ModelSim Directory** (File menu).
3. Browse to the directory where the ModelSim executables are located. For example, for ModelSim-Altera, the directory is *<ModelSim installation directory>/win32aloem*.
4. Click **OK**.
5. Click the **System Generation** tab.
6. Enable the **Simulation** option. See [Figure 4 on page 7](#).
7. Click **Re-Generate**.

Figure 4. SOPC Builder Simulation File Generation



### SOPC Builder Generated System Simulation Files

At this point, SOPC Builder has generated your system and created all of the files necessary for simulation as shown in [Table 1](#) apart from the memory initialization files. These simulation files are located in the *<your project directory>std1S10\_sim* simulation directory.

**Table 1. SOPC Files Generated for Nios II Simulation**

File Extension	Description
.mpf	ModelSim project file. This file is created if SOPC Builder finds the ModelSim path.
.do	ModelSim macro execution scripts. The <b>setup_sim.do</b> script initializes the macros listed in <a href="#">Table 2 on page 15</a> . The <b>wave_presets.do</b> script generates a list of default signals that are displayed in the waveform window.
.dat	Memory initialization files in hexadecimal format. These files are used for simulation only. The .dat files are created to initialize components in your system such as UARTS. Additional .dat files need to be generated using the Nios II IDE to load the memories used in your design.

## Using the Nios II IDE to Generate Memory Initialization Files

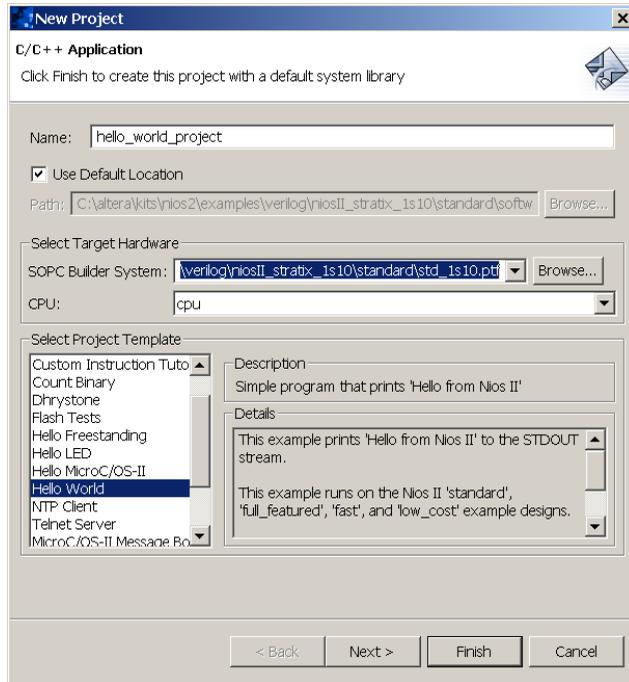
This section describes how to finish setting up your simulation by using the Nios II IDE to create a software test project and to generate the necessary files to initialize the memories used in your simulation. For further details on the operation of the Nios II IDE refer to the Nios II IDE on-line tutorials.

### Creating a Nios II IDE Project

Perform the following steps to generate and compile an example software project using the Nios II IDE.

1. Run the **Nios II IDE** software.
2. Select **New > Project** (File menu) to create a new project.
3. Select **Altera Nios II > C/C++ Application > Next**.
4. Enter **hello\_world\_project** as your software project name and browse to the location of your system's PTF file to specify **SOPC Builder System**. Your system PTF file is located in the *<your project directory>/standard* directory.
5. Select **Hello World** from the **Select Project Template** field as shown in [Figure 5 on page 9](#).

Figure 5. Nios II IDE New Project Wizard



6. Click **Next**.
7. On the next page of the New Project wizard, turn on **Creating a new library**.
8. Select **Finish**.

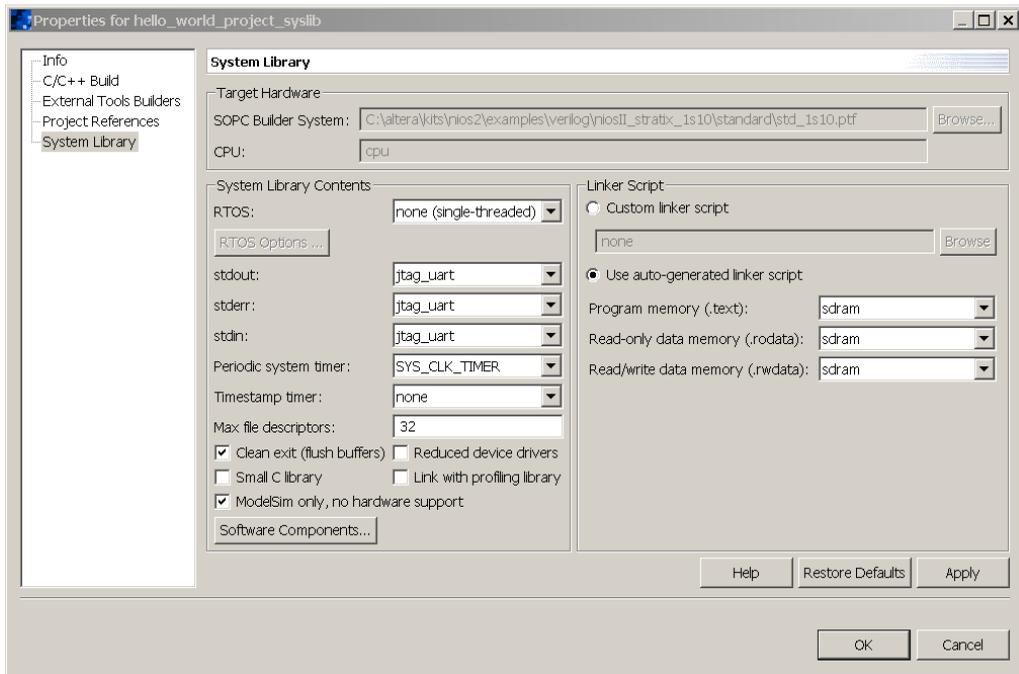
## Modify the System Library for Simulation

The next stage in preparing for simulation is to modify the system library parameters for your project. To specify which memory you want your code compiled and to increase simulation speed by reducing code overhead, perform the following steps:

1. Right click on your system library **hello\_world\_project\_syslib[std\_1s10]** located in the **C/C++ Project** panel of the Nios II IDE and select **Properties**.

2. Select **System Library** from the **Properties for hello\_world\_project\_syslib** dialog box.
3. The **System Library** page is used to select which communication devices in your design will be used for standard out and standard input. See [Figure 6](#). The **System Library** page is also used to specify which memory will be used at run time by the CPU.
  - a. Scroll to **sdram** for the **Program, Read-only, and Read/write** memory fields.
  - b. Scroll to **jtag\_uart** for the **stdout, stderr and stdin** System Library Contents fields.
  - c. Turn on **ModelSim only, no hardware support**.
4. Click **OK**.

**Figure 6. System Library Properties**



The system library has now been modified to create an executable image that can be downloaded to your program memory (in our case, SDRAM) and simulated in ModelSim. Turning on **ModelSim only, no hardware support** indicates to the compiler that the current project is being run on a simulator. In turn, the compiler will remove sections of the startup code in order to improve simulation speed. Specifically, the instruction and data caches will not be initialized during simulation and the BSS section of the Read/write data memory will not be cleared. These enhancements will greatly improve the speed at which your design will simulate in ModelSim; however, the resulting software image will not run on a target board.



Before downloading your software image to your target board; it is vital to disable the **ModelSim only, no hardware support** option inside the **System Library Contents** field and recompile your software project.

## C Code for the Hello World Project

The example design that you will simulate is based on the **hello\_world.c** file located in the *<your project directory>*\standard\software\**hello\_world\_project** directory. This file simply prints a message to the JTAG UART on the target board. You can view the source code for the project by opening the **hello\_world\_project** folder in the Nios II IDE **C/C++ projects** panel and double-clicking on the **hello\_world.c** file. The source code for this file is shown below.

### hello\_world.c Source Code

```
#include <stdio.h>

int main ()
{
    printf("Hello from Nios II!\n");

    return 0;
}
```

To demonstrate how to access hardware peripherals we will modify the **hello\_world.c** source file to write to the PIO LEDs on the development board. Modify the **hello\_world.c** to use the **IOWR\_ALTERA\_AVALON\_PIO\_DATA** macro to write values to the LEDs as shown below.

### Modified hello\_world.c Source Code

```
#include <stdio.h>
#include "system.h"
#include "altera_avalon_pio_regs.h"

int main ()
{
    int i;

    printf("Hello from Nios II!\n");

    for(i=0;i<256;i++) {
        IOWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE, i);
    }

    return 0;
}
```

 If you copy and paste the **modified hello\_world.c** source code from an Adobe pdf file to the **hello\_world.c** file, you may need to delete the quotation marks and re-type them in Nios II IDE. Quotation marks from a pdf file may cause build errors in Nios II IDE.

The modifications made in “**Modified hello\_world.c Source Code**” write values to the LED using the `IOWR_ALTERA_AVALON_PIO_DATA` macro which is defined in the **altera\_avalon\_pio\_regs.h** file. The `IOWR_ALTERA_AVALON_PIO_DATA` routine accepts two arguments, the first is the base address of the PIO and the second is the value that will be written to the PIO. In our case the PIO base address is `LED_PIO_BASE` and is defined in the **system.h** file.

 When accessing a PIO peripheral you should use the `IOWR_ALTERA_AVALON_PIO_DATA` and `IORD_ALTERA_AVALON_PIO_DATA` macros defined in the **altera\_avalon\_pio\_regs.h** header file.

### *Compile the Software Project*

After modifying the `hello_world.c` source code and the system library to generate an executable image that will run faster in ModelSim you are ready to compile the project to generate the **.dat** files which will be used to initialize the memories in your system for simulation purposes.

To compile your software project in the Nios II IDE, perform the following steps:

1. Highlight **hello\_world\_project** in the **C/C++ Projects** panel of the Nios II IDE.

2. Right click on **hello\_world\_project** and select **Build Project**.

Once compilation is completed the **.dat** files in *<your project directory>/standard/std\_1s10\_sim* project will be initialized. In this case **sdram.dat** will contain the executable image for the **hello\_world.c** file that was compiled by the Nios II IDE.

## Launch ModelSim Using the Nios II IDE

The Nios II IDE can be used to launch the ModelSim simulator and to set up the simulator to run our simulation. After launching the simulator the Nios II IDE has no further role in the simulation process and all subsequent simulation commands are performed in the ModelSim simulation tool.

Launch the ModelSim simulator and set it up to run our project by performing the following steps:

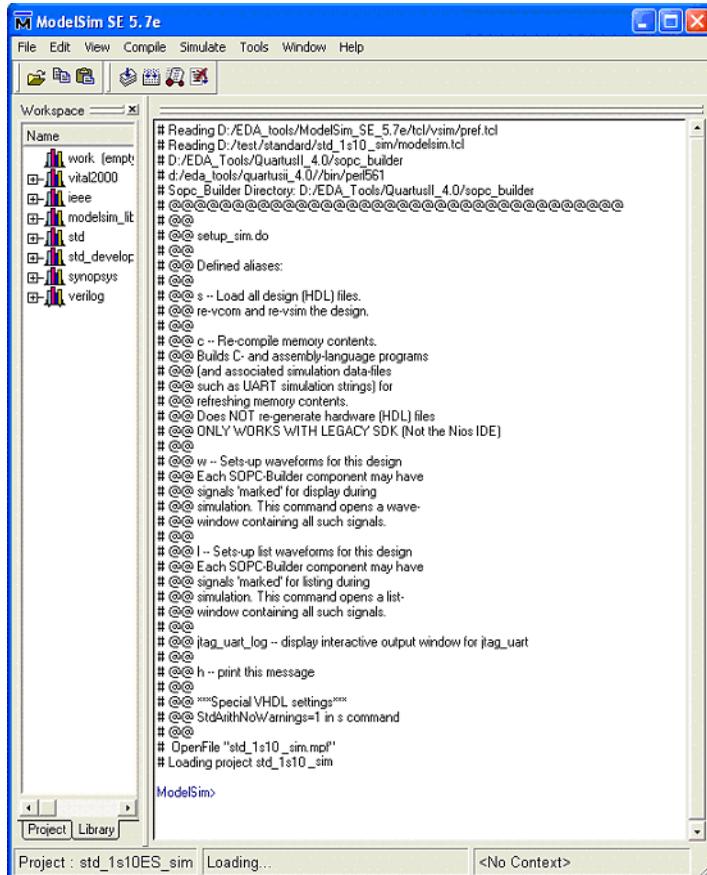
1. Highlight **hello\_world\_project** in the **C/C++ Projects** panel of the Nios II IDE.
2. Select **Run...** (Run menu).
3. Highlight the **Nios II ModelSim** icon in the **Configurations** window of the **Run** dialog box.
4. Click **New**.
5. Ensure that the **ModelSim** path points to the executable directory for ModelSim. If this path is incorrect you can specify the ModelSim path location using SOPC Builder as described on [page 6](#).
6. Select **Run**.

Pressing the Run button launches Modelsim and causes ModelSim to compile the **setup\_sim.do** script and wait for you to run the simulation.

### *Running the Simulation Using ModelSim*

After you have launched ModelSim from the Nios II IDE the **setup\_sim.do** script is executed and the available macros are shown in the ModelSim console window. See [Figure 7 on page 14](#). These macros make it easy for you to load your design files and to view the default signals for your design. The individual macros are defined in [Table 2 on page 15](#).

Figure 7. ModelSim After Pressing Run in the Nios II IDE



**Table 2. Nios Simulation Macros**

Macros	Description
s	Recompiles the Nios processor and peripheral source code and then reloads the design into the ModelSim work library for simulation. This macro resets the entire simulation.
c	This is a legacy command that is not supported for designs generated using the Nios II IDE. It's purpose was to recompile source code and reinitialize the system memories. For Nios II the recommended approach to reinitializing memories is to recompile your design in Nios II IDE and then restart your design in ModelSim after the Nios II IDE compilation is complete.
w	Loads the wave_presets.do file, which contains predefined ModelSim waveform window information. The wave_presets.do file loads the common signals from all of the processors and peripherals that reside on-chip and displays the ModelSim waveform window.
l	Loads the wave_presets.do file, which contains predefined ModelSim waveform window information. The wave_presets.do file loads the common signals from all of the processors and peripherals that reside on-chip and displays the ModelSim waveform window.
<UART name>_log	Optional. For each UART in the system, this macro is created if you turned on the display interactive output window inside SOPC Builder before system generation. When you run this macro it opens a window, similar to a terminal screen, showing TXD data from the UART.
<UART name>_drive	Optional. For each UART in the system, this macro is created if you turned on the interactive/stimulus response window inside SOPC Builder before system generation. When you run this macro it opens a window, similar to a terminal screen, where you can send virtual data to the UART RXD signal during simulation.
h	Help. Displays the available macros and their functions.

Run the simulation within ModelSim by performing the following steps:

1. Type **s** in the ModelSim console window to execute the **s** macro to load the design.
2. Execute the **jtag\_uart\_drive** macro to launch the interactive terminal window that displays the output of the `printf` statement in the **hello\_world.c** source code.
3. Execute the **w** macro to display the ModelSim waveform window with example signals that were automatically generated for your system. These signals are separated by function and include signals useful for debugging. [Table 3 on page 16](#) lists signals included in the default waveform.

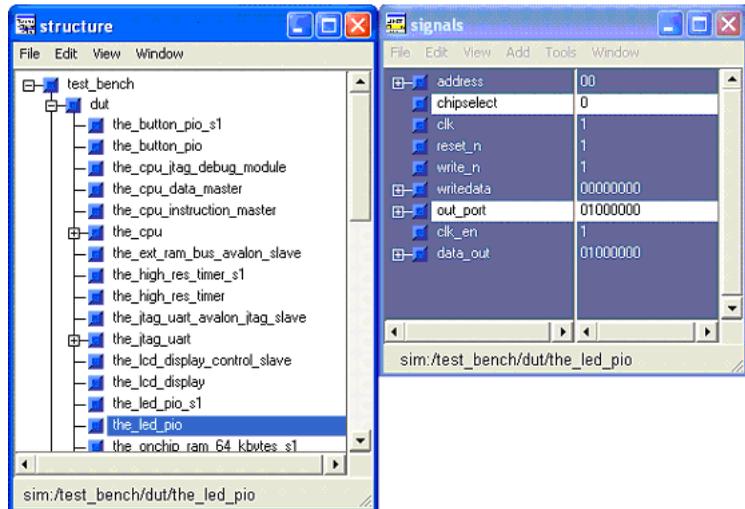
4. You could now run the design in ModelSim using the standard ModelSim commands. However, we will add a few additional signals to the wave window so that the operation of the PIO peripheral can be observed. In order to view the operation of the PIO peripheral open the **Structures** and **Signals** windows in ModelSim (these windows are accessed via ModelSim's View menu).

**Table 3. Signals Shown in Simulation Waveform**

Signal Group	Description
cpu	Signals related to instruction fetching and read and writing data. Signals in this group beginning with a <code>d_</code> prefix are associated with the CPU data master. These signals provide information on when the CPU data master is performing read or write access to memory or memory mapped peripherals. Signals beginning with a <code>i_</code> prefix are associated with the CPU instruction master. These signals indicate when the CPU instruction master is accessing instructions from memory.
sdram	Signals showing the interface between the Avalon bus module and the SDRAM controller, SDRAM controller and SDRAM device(s), and signals internal to the SDRAM controller logic. Signals from the Avalon bus module to the SDRAM controller have the prefix <code>az_</code> , e.g., <code>az_addr</code> for the address bus input. Signals from the SDRAM controller to the Avalon bus module have the prefix <code>za_</code> , e.g., <code>za_data</code> for the data from the controller to the Avalon bus module. Signals between the SDRAM controller and external SDRAM device(s) have the prefix <code>zs_</code> , e.g., <code>zs_ras_n</code> for the row address strobe signal. Signals internal to the SDRAM controller logic include the system clock ( <code>clk</code> ) and the current operation that the SDRAM controller is performing (code).
onchip_ram_64_kbytes	Address, data and control signals for the <code>onchip_ram_64</code> kbytes memory.
jtag_uart	Displays the Avalon interface signals to the JTAG UART.
uart1 Bus Interface	Signals displaying the UART bus interface. Display the Avalon address and data signals for the UART.
uart1 Internals	Internal UART signals showing the UART transmit (TX) and receive (RX) data registers. The signals decode the 8-bit TX and RX registers to ASCII text so that you can view the characters in the simulation waveform. The TX ready and RX character ready signals are also shown.

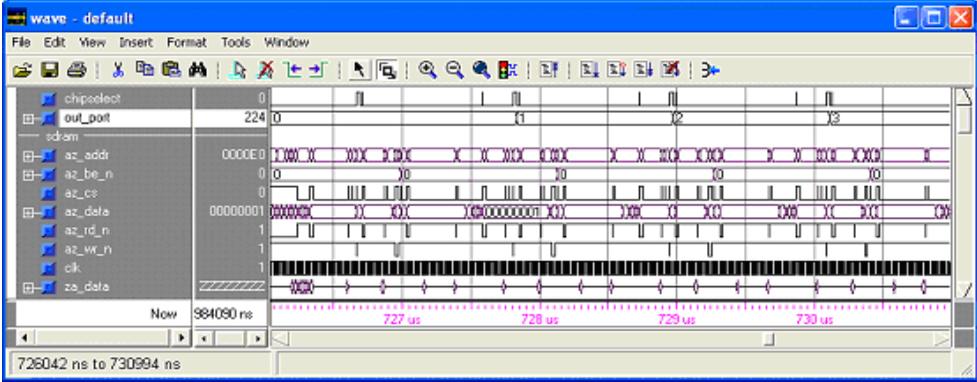
5. In the **Structure** window open the `test_bench` folder followed by the `dut` folder and select `the_led_pio`.
6. In the **Signals** window, select `chipselect` and `out_port`. See [Figure 8](#).

Figure 8. ModelSim Structure and Signals Windows



7. Then drag the highlighted signals into the waveform window.
8. Run the simulation for 800 microseconds by typing `run 800 us` in the ModelSim console.
9. After the simulation has completed the terminal window should display the `printf` statement from the `hello_world.c` file. Also, zoom in on the PIO signals that you added to the waveform window and observe the CPU making write access to the PIO as shown in [Figure 9 on page 18](#).

Figure 9. Simulation Results



## Conclusion

Simulation and verification are vital parts of the design process. Nios II processors can be verified comprehensively using board-level debugging, software emulation using the Nios II ISS and RTL simulation using ModelSim. RTL simulation is an important part of the design process particularly for configurable systems as it allows you to probe deeply embedded signals within the processor and your peripheral set.



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

Copyright © 2004 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.

 Printed on recycled paper

