Simulating Nios II Embedded Processor Designs

© November 2008 AN-351-1.2

Introduction

This application note describes the process of generating an RTL simulation environment using Nios II example designs, SOPC Builder, and the Nios II software build tools. It also describes the process of running the Nios II RTL simulation in the ModelSim simulator.

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

- **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. You can use the Nios II integrated development environment (IDE) with its built-in debugger to verify designs running on either development or custom boards. You can find more information about the Nios II IDE debugger in the Nios II IDE Help. You can further debug the hardware components that interact with the processor with the SignalTap® II embedded logic analyzer.

  For more information about the SignalTap II embedded logic analyzer, refer to *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*.

- **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 is often difficult to view signals buried deep in the system. RTL simulation alleviates this problem as it enables you to functionally probe every register and signal in the design. Nios II-based systems are easily simulated in the ModelSim® simulator using an automatically generated simulation environment created by SOPC Builder and the Nios II software build tools.

  For more information about Nios II software build tools, refer to the *Introduction to the Nios II Software Build Tools* chapter in the *Nios II Software Developer's Handbook*.

Before You Begin

This document assumes that you have prior experience using SOPC Builder as well as a familiarity with the ModelSim simulator. In order to simulate the Nios II design using the instructions in this document, you must have the following software installed:

- The Quartus® II software version 8.1 or later
- ModelSim-Altera 6.3g or higher, or ModelSim PE, SE, or EE
- Nios II Embedded Development Suite (EDS) 8.1 or later

Setting Up Your Simulation Environment in SOPC Builder

To open the example design, perform the following steps:

1. Locate the <*Nios II EDS install directory*>**/examples/**<*vhdl or verilog*>**/ niosII_stratixII_2s60/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 is referred to as <*your project directory*> throughout the remainder of this document.
2. Start the Quartus II software.
3. On the File menu, click **Open Project**.
4. Browse to <*your project directory*>**/standard**.
5. Select **NiosII_stratixII_2s60_standard.qpf**.
6. Click **Open**.
7. On the Tools menu, click **SOPC Builder**.

Memory Initialization

To run a simulation of a Nios II design, you must initialize any memories that contain software prior to simulation. You can create the memory initialization files using the Nios II software build tools. Later sections of this document explain how to use the Nios II software build tools to create memory initialization files for your simulation.

You can use two types of memory models for simulation purposes: generic and vendor-specific. This application note discusses the generic memory model. Table 1 shows the different types of memory and their corresponding simulation models.

For more information about simulating each of the memory types, refer to the documents listed in Table 1.

Table 1. Simulation Models Provided by Altera

Memory Type	Simulation Model	Documentation
On-chip memory	Generic memory model	*Building Memory Subsystem Development Walkthrough* chapter in volume 4 of the *Quartus II Handbook*
Off-chip SRAM	Generic memory model	*Building Memory Subsystem Development Walkthrough* chapter in volume 4 of the *Quartus II Handbook*
Flash	Generic memory model	*Building Memory Subsystem Development Walkthrough* chapter in volume 4 of the *Quartus II Handbook*
SDRAM	Generic memory model	■ *Building Memory Subsystem Development Walkthrough* chapter in volume 4 of the *Quartus II Handbook* ■ *SDRAM Controller Core* chapter in volume 5 of the *Quartus II Handbook*

JTAG UART Settings

You can use SOPC Builder to customize a JTAG UART for generating a data stream to send to the host processor during simulation.

To set up the JTAG UART for a simulation, perform the following steps in SOPC Builder:

1. In the **Module Name** column, double-click **jtag_uart**. The **JTAG UART** dialog box opens.
2. In the **JTAG UART** dialog box, click the **Simulation** tab.
3. Turn on the **Create ModelSim alias to open an interactive stimulus/response window** option.
4. Click **Finish**.

For more information about setting up a JTAG UART for simulation, refer to the *JTAG UART Core* chapter in volume 5 of the *Quartus II Handbook*.

Parallel I/O (PIO) Settings

You can also use SOPC Builder to initialize the inputs of any PIO peripherals in your design that have an input port. For example, you can initialize PIOs, with the direction set to bidirectional (tri-state) ports, input ports only, or both input and output ports, for simulation in SOPC Builder.

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

To initialize the PIO in the example design, perform the following steps in SOPC Builder:

1. In the **Module Name** column, double-click **button_pio**. The **PIO (Parallel I/O)** dialog box opens.
2. In the **PIO (Parallel I/O)** dialog box, click the **Simulation** tab.
3. In the **Drive inputs to** field, type the initial value that you want to drive on the input ports; for example type `0xF`.
4. Click **Finish**.

SOPC Builder Simulation Settings

After you set up the simulation settings for the JTAG-UART and PIO peripherals in your design, you must enable simulation file generation before you start generating the system. To enable simulation file generation in SOPC Builder, perform the following steps:

1. On the **System Generation** tab, turn on the **Simulation. Create project simulator files** option.
2. Click **Generate**. Click **Save** when prompted to save changes.
3. Click **Exit** when system generation is complete.

SOPC Builder Generated System Simulation Files

At this point in the design, SOPC Builder has generated your system and created all of the files necessary for simulation listed in Table 2, except for the memory initialization files. These simulation files are located in the <*your project directory*>**/standard/ NiosII_stratixII_2s60_standard_sim** directory.

Table 2. SOPC Files Generated for Nios II Simulation

File Extension	Description
.mpf	ModelSim project file. This file is created if SOPC Builder detects that the ModelSim simulator is installed in the machine.
.do	ModelSim macro execution scripts. The **setup_sim.do** script initializes the macros listed in Table 3. The **wave_presets.do** 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 software build tools to load the memories used in your design.

Using the Nios II Software Build Tools to Generate Memory Initialization Files

This section describes how to finish setting up your simulation by using the Nios II software build tools to create a software test project and to generate the necessary files for initializing the memories used in your simulation.

For further details about the Nios II software build tools, refer to the *Introduction to the Nios II Software Build Tools* chapter in the *Nios II Software Developer's Handbook*.

Creating a Nios II Software Build Tools Project

The software test project you are going to simulate is a simple **Hello World** application. Download and unzip the **AN351_software_file.zip** file into the <*your project directory*>**/standard** directory. A hyperlink to the **AN351_software_file.zip** file appears next to this application note on the Application Notes web page.

This software prints a message to the JTAG UART on the target board and displays an incrementing binary value on the LEDs. For further details about this software, refer to **hello_world.c** source file located in the <*your project directory*>**/standard/ software/app/hello_world** directory.

To create and build the software project using the Nios II software build tools, perform the following steps:

1. Open a Nios II command shell. On the Windows Start menu, point to **Programs>Altera>Nios II EDS 8.1** and then click **Nios II 8.1 Command Shell**.
2. Change the directory to <*your project directory*>/**standard/software/ app/hello_world**.

3. To create and build the application with the **create-this-app** script, type the following in the command shell:

```
./create-this-app
```

Building the software project takes some time.

☞ The `mem_init_install` makefile target in the **create-this-app** script tells the software build tools to generate the memory initialization files. When the compilation is complete, the **.dat** files in the <*your project directory*>/**standard/ NiosII_stratixII_2s60_standard_sim** directory are updated.

Board Support Package (BSP) Settings

To improve the simulation speed, change the BSP settings for the software project. To specify which memory you want your code compiled for and to increase simulation speed by reducing code overhead, perform the following steps:

1. Open the **create-this-bsp** script located in <*your project directory*>/ **standard/software/bsp/hal_default** directory using any text editor of your choice.
2. Search for `NIOS2_BSP_ARGS` variable in the script.

   There are some **nios2-bsp** utility arguments assigned to the variable. Table 3 lists these arguments and their purpose.

Table 3. Purposes for the nios2-bsp Utility Arguments

Arguments	Purpose
--set hal.enable_sim_optimize 1	This argument turns on **ModelSim only, no hardware support** option. This option tells the compiler that the current project is being run on a simulator. In turn, the compiler removes sections of the startup code to improve simulation speed. Specifically, the instruction and data caches are not initialized during simulation and the `.bss` section of the read/write data memory is not cleared. These enhancements greatly improve the speed at which your design is simulated in the ModelSim simulator; however, the resulting software image will not run on a target board. Before downloading your software image to your target board, you must disable the **ModelSim only, no hardware support** option by setting this argument value to zero and rerun the **create-this-app** script.
--default_sections_mapping sdram	This argument sets the Program, Read-only data, Read/write data, Heap, and Stack memories to `sdram`. This specifies that the software code is compiled for this memory module.
--default_stdio jtag_uart	This argument sets the `stdout`, `stderr`, and `stdin` devices to `jtag_uart`.

Launch the ModelSim Simulator Using the Nios II Command Shell

You can use the Nios II command shell to launch and set up the ModelSim simulator to run your simulation. After launching the ModelSim simulator, the Nios II command shell has no further role in the simulation process. All subsequent simulation commands are performed in the ModelSim simulator.

Set up the ModelSim simulator to run your project by performing the following steps:

1. In the Nios II command shell, change the directory to <*your project directory*>/**standard/NiosII_stratixII_2s60_standard_sim**.
2. To launch the ModelSim simulator, type the following in the command shell:

```
vsim setup_sim.do &
```

Once the ModelSim simulator is launched, it then compiles the **setup_sim.do** script and waits for you to run the simulation.

Running the Simulation Using the ModelSim Simulator

After you have launched the ModelSim simulator from the Nios II command shell, the **setup_sim.do** script is run and the available macros are shown in the Nios II command shell. These macros make it easy for you to load your design files and view the default signals for your design. The available macros are described in Table 4.

Table 4. Nios II Simulation Macros

Macros	Description
`s`	Loads all design (HDL) files into the ModelSim work library, recompiles the design files, and starts the simulation.
`c`	Recompiles memory contents. Builds C- and assembly-language programs (and associated simulation data-files such as UART simulation strings) for refreshing memory contents. Works only with the Legacy Software Development Kit (SDK).
`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 **list_presets.do** file, which contains predefined ModelSim list window information. The **list_presets.do** file loads the common signals from all of the processors and peripherals that reside on-chip and displays the ModelSim list window.
`h`	Help. Displays a list of the available macros and their functions.
*<UART name>***_drive**	Optional. For each UART in the system, this macro is created if you turned on the **Create ModelSim alias to open an interactive/stimulus response window** option inside SOPC Builder before system generation. When you run this macro, it opens an interactive display window.

Run the simulation in the ModelSim simulator by performing the following steps:

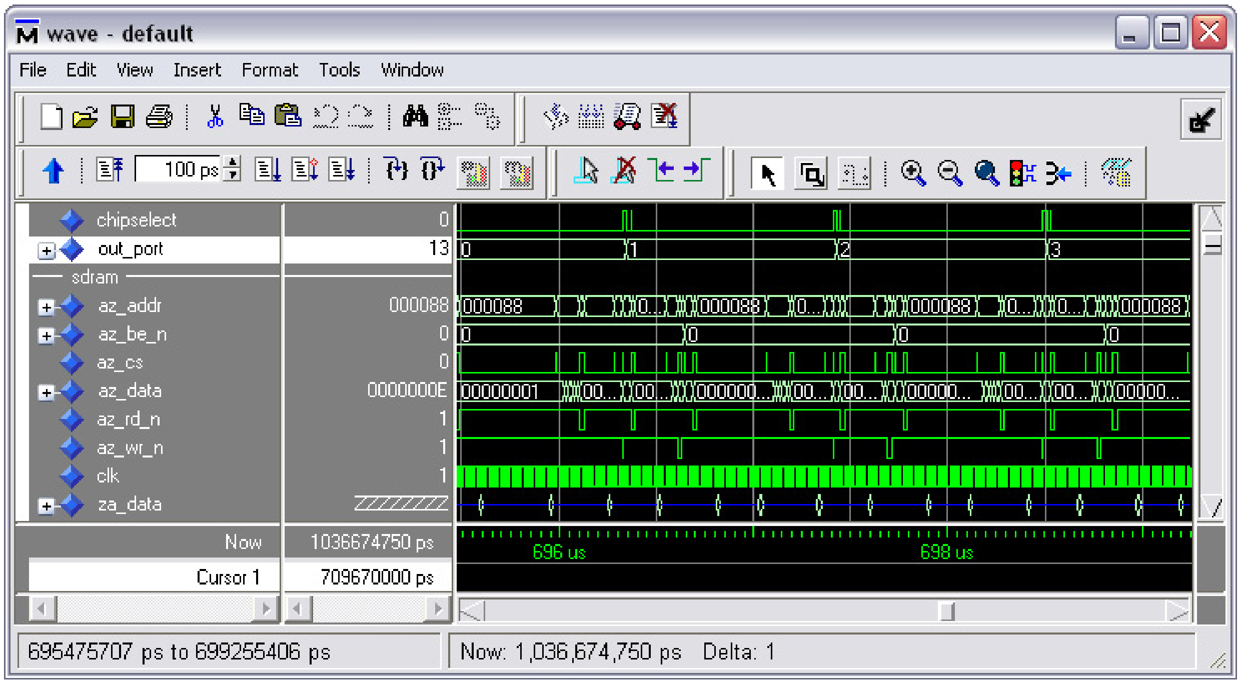
1. Type `s` in the ModelSim Transcript window to run the **s** macro to load the design.
2. Run the **jtag_uart_drive** macro to launch the interactive terminal window that displays the output of the `printf` statement in the **hello_world.c** source code.
3. Run the **w** macro to display the ModelSim waveform window with the example signals that were automatically generated for your system. These signals are separated by function and include signals useful for debugging. Table 5 lists the signals included in the default waveform.
4. Run the design in the ModelSim simulator using the standard ModelSim commands. However, by adding a few additional signals to the wave window, you can observe the operation of the PIO peripheral. To view the operation of the PIO peripheral, click on the **sim** tab inside the Workspace window. (If the Workspace window is not open, on the View menu, click **Workspace**.)

Table 5. Signals Shown in Simulation Waveform

Signal Group	Description
`cpu`	Signals related to instruction fetching and reading and writing data. Signals in this group beginning with a `d_` 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 an i_ prefix are associated with the CPU instruction master. These signals indicate when the CPU instruction master is reading instructions from memory.
`sdram`	Signals that show the following: ■ The interface between the system interconnect fabric and the SDRAM controller ■ The interface between the SDRAM controller and SDRAM device(s) ■ Signals internal to the SDRAM controller logic Signals from the system interconnect fabric to the SDRAM controller have the prefix `az_`. For example, `az_addr` is the address bus input. Signals from the SDRAM controller to the system interconnect fabric have the prefix `za_`. For example, `za_data` is the data from the controller to the system interconnect fabric. Signals between the SDRAM controller and external SDRAM device(s) have the prefix `zs_`. For example, `zs_ras_n` is the row address strobe signal. Signals internal to the SDRAM controller logic include the system clock (`clk`) and the current operation that the SDRAM controller is performing (code).
`onchip_ram`	Address, data, and control signals for the `onchip_ram` memory.
`jtag_uart`	Displays the Avalon® Memory-Mapped (MM) slave port interface signals of the JTAG UART.
`uart1 Bus Interface`	Signals displaying the UART bus interface. These signals 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 Workspace window, expand **test_bench** and expand **DUT** to select **the_led_pio**.
6. In the Objects window, select **chipselect** and **out_port**. (If the Objects window is not open, on the View menu, click **Objects**.)
7. Drag the selected signals into the waveform window.
8. Run the simulation for 800 microseconds by typing `run 800 us` in the ModelSim Transcript window.
9. After the simulation has completed, the terminal window should display `Hello from Nios II!` message. Also, zoom in on the PIO signals that you added to the waveform window and observe the CPU writing data to the PIO as shown in Figure 1.

Figure 1. Simulation Results



Conclusion

Simulation and verification are vital parts of the design process. You can comprehensively verify the Nios II processors with board-level debugging, and RTL simulation using the ModelSim simulator. RTL simulation is an important part of the design process, particularly for configurable systems, because it enables you to probe deeply embedded signals in the processor and your peripheral set. RTL simulation also helps verify your system before you try out your design in the actual hardware.

Revision History

Table 6 shows the revision history for this application note.

Table 6. Document Revision History

Date and Revision	Changes Made	Summary of Changes
November 2008, version 1.2	Updated for the Nios II processor 8.1 release.	Updated the tutorial section.
November 2007, version 1.1	Updated for the Nios II processor 7.2 release.	Updated the screenshots.
May 2004 version 1.0	Initial Release.	—

101 Innovation Drive
San Jose, CA 95134
www.altera.com
Technical Support
www.altera.com/support