Nios Hardware Development Tutorial

101 Innovation Drive
San Jose, CA 95134
(408) 544-7000
http://www.altera.com

Document Version: 1.2
Document Date: January 2004

About this Document

This tutorial introduces you to the Altera® Nios® system module. It shows you how to use the Quartus® II software to create and process your own Nios system module design that interfaces with components provided on the Nios development board.

Table 1 shows the tutorial revision history.

Refer to the Nios embedded processor readme file for late-breaking information that is not available in this tutorial.

Table 1. Tutorial Revision History

Date	Description
January 2004	Reflects updates for Quartus II software - version 4.0 and Nios Development Kit version 3.2
July 2003	Reflects new directory structure for SOPC Builder 3.0 and Nios Development Kit version 3.1.
May 2003	First release of this hardware tutorial for the 1S10, 1C20, and 1S40 Nios development boards.

How to Find Information

- The Adobe Acrobat Find feature allows you to search the contents of a PDF file. Click the binoculars toolbar icon to open the Find dialog box.
- Bookmarks serve as an additional table of contents.
- Thumbnail icons, which provide miniature previews of each page, provide a link to the pages.
- Numerous links, shown in green text, allow you to jump to related information.

How to Contact Altera

For the most up-to-date information about Altera products, go to the Altera world-wide web site at http://www.altera.com.

For technical support on this product, go to http://www.altera.com/mysupport. For additional information about Altera products, consult the sources shown in Table 2.

Table 2. How to Contact Altera

Information Type	USA & Canada	All Other Locations
Product literature	http://www.altera.com	http://www.altera.com
Altera literature services	lit_req@altera.com *(1)*	lit_req@altera.com *(1)*
Non-technical customer service	(800) 767-3753	(408) 544-7000 (7:30 a.m. to 5:30 p.m. Pacific Time)
Technical support	(800) 800-EPLD (3753) (7:30 a.m. to 5:30 p.m. Pacific Time)	(408) 544-7000 *(1)* (7:30 a.m. to 5:30 p.m. Pacific Time)
	http://www.altera.com/mysupport/	http://www.altera.com/mysupport/
FTP site	ftp.altera.com	ftp.altera.com

Note:
(1) You can also contact your local Altera sales office or sales representative.

Typographic Conventions

This document uses the typographic conventions shown in Table 3.

Table 3. Conventions

Visual Cue	Meaning
Bold Type with Initial Capital Letters	Command names, dialog box titles, checkbox options, and dialog box options are shown in bold, initial capital letters. Example: **Save As** dialog box.
bold type	External timing parameters, directory names, project names, disk drive names, filenames, filename extensions, and software utility names are shown in bold type. Examples: $\mathbf{f_{MAX}}$, **\qdesigns** directory, **d:** drive, **chiptrip.gdf** file.
Italic Type with Initial Capital Letters	Document titles are shown in italic type with initial capital letters. Example: *AN 75: High-Speed Board Design.*
Italic type	Internal timing parameters and variables are shown in italic type. Examples: t_{PIA}, $n + 1$. Variable names are enclosed in angle brackets (< >) and shown in italic type. Example: <*file name*>, <*project name*>**.pof** file.
Initial Capital Letters	Keyboard keys and menu names are shown with initial capital letters. Examples: Delete key, the Options menu.
"Subheading Title"	References to sections within a document and titles of on-line help topics are shown in quotation marks. Example: "Typographic Conventions."
`Courier type`	Signal and port names are shown in lowercase Courier type. Examples: `data1`, `tdi`, `input.` Active-low signals are denoted by suffix `n`, e.g., `resetn`. Anything that must be typed exactly as it appears is shown in Courier type. For example: `c:\qdesigns\tutorial\chiptrip.gdf`. Also, sections of an actual file, such as a Report File, references to parts of files (e.g., the AHDL keyword `SUBDESIGN`), as well as logic function names (e.g., `TRI`) are shown in Courier.
1., 2., 3., and a., b., c.,...	Numbered steps are used in a list of items when the sequence of the items is important, such as the steps listed in a procedure.
■	Bullets are used in a list of items when the sequence of the items is not important.
✓	The checkmark indicates a procedure that consists of one step only.
☞	The hand points to information that requires special attention.
↵	The angled arrow indicates you should press the Enter key.
👣	The feet direct you to more information on a particular topic.

Notes:

Table of Contents

About this Document iii
- How to Find Information iii
- How to Contact Altera iv
- Documentation Feedback iv
- Typographic Conventions v

Tutorial Overview 9
- Introduction 9
- Hardware & Software Requirements 9
- Tutorial Files 10
  - What This Tutorial Does Not Teach You 10
- Hardware/ Software Development Flow 11
  - Hardware Development Flow 12

Designing & Compiling 15
- Accessing a Quartus II Project 15
- Create a Nios System Module 16
  - Start SOPC Builder 17
  - System Speed 18
  - Add CPU & Peripherals 19
    - Nios 32-Bit CPU 21
    - On-Chip Boot Monitor ROM 23
    - Communications UART 24
    - Timer 25
    - Button PIO 26
    - LCD PIO 27
    - LED PIO 28
    - Seven Segment PIO 29
    - External RAM Bus (Avalon Tri-State Bridge) 30
    - External RAM Interface 30
    - External Flash Interface 32
  - Specify Base Addresses 33
    - Setting the Flash Base Address 33
  - Generate the System Module 35
  - Add the Symbol to the BDF 38

Compile the Design 39

Programming 41
Configure the FPGA 41
Custom Microcontroller —No Way! 45
Running Software on Your Nios System 46
Nios SDK Shell Tips 46
Start the Nios SDK Shell 47
Compile & Run the Sample hello_nios.srec Test Program 48
Download the Design to Flash Memory 49
Next Steps 51

Index 53

Tutorial Overview

Introduction

This tutorial introduces you to hardware development for the Nios processor and walks you through the hardware development flow. It shows you how to use SOPC Builder and the Quartus® II software to create and process your own Nios system design that interfaces with components provided on your Nios development board.

This tutorial is for users who are new to the Nios processor as well as users who are new to the concept of using embedded systems in FPGAs. The tutorial guides you through the steps necessary to create and compile a 32-bit Nios system design, called **nios_system_module**. This simple, single-master Nios system consists of a Nios embedded processor and associated system peripherals and interconnections for use with the input and output hardware available on a Nios development board.

When the FPGA device on the Nios development board is configured with the Quartus II project encapsulating **nios_system_module**, the external physical pins on the FPGA are used by the design to connect to other hardware on the Nios development board, allowing the Nios embedded processor to interface with RAM, flash memory, LEDs, LCDs, switches, and buttons.

This tutorial is divided into the following two sections:

- "Designing & Compiling" on page 15 teaches you how to use SOPC Builder to create the Nios system module in a Block Design File (.**bdf**) and how to compile the Nios design using the Quartus II Compiler.
- "Programming" on page 41 teaches you how to use the Quartus II Programmer and the ByteBlaster™ II cable to configure the FPGA on the Nios development board. It also teaches you how to store the design in the flash memory device provided on the board, so that the FPGA can be configured with your design whenever power is applied to the board.

Hardware & Software Requirements

This tutorial requires the following hardware and software:

- A PC running the Windows NT/2000/XP operating system
- Nios embedded processor version 3.02 and the SOPC Builder software version 2.82 or higher
- The Quartus II software version 2.2 SP1 or higher

- A Nios development board connected to a PC as described in the Getting Started User Guide provided with the following three kits:
  - Nios Development Kit, Stratix Edition
  - Nios Development Kit, Stratix Professional Edition
  - Nios Development Kit, Cyclone Edition

Tutorial Files

This tutorial assumes that you create and save your files in a working directory on the **C:** drive of your computer. If your working directory is on another drive, substitute the appropriate drive name.

The Nios embedded processor software installation creates the directories shown in Table 4 in the **\altera\kits\nios** directory by default:

Table 4. Directory Structure

Directory Name	Description
bin	Contains tools required for developing Nios hardware & software designs, including the GNU tool chain.
components	Contains all of the SOPC Builder peripheral components. Each peripheral has its own subdirectory with a class.ptf file that describes the component.
documents	Contains documentation for the Nios embedded processor software, Nios development board, and GNUPro Toolkit.
examples	Contains subdirectories of Nios sample designs, including the **standard_32** project on which the **nios_system_module** design is based.
tutorials	Contains tutorials with their related files for the Nios embedded processor and SOPC Builder. The directory for this tutorial is found in each of the following kit-specific directories: (1) **Nios_HW_Tutorial_Stratix_1S10** (2) **Nios_HW_Tutorial_Cyclone_1C20** (3) **Nios_HW_Tutorial_Stratix_1S40**

What This Tutorial Does Not Teach You

This tutorial starts from a pre-defined Quartus II project with components chosen, pins and other logic placed and then wired to the pins. As such, it does not teach you how to create a Quartus II project, how to set compilation settings, or how to place and assign pins.

See http://www.altera.com/literature/lit-qts.html for more information about Quartus II software.

Hardware/ Software Development Flow

Figure 1 shows a complete design flow for creating a Nios system and prototyping it on the Nios development board. The diagram includes both the hardware and software design tasks required to create a working system. The right side shows the software development flow while the left side shows the hardware design flow. This tutorial walks you through the steps "Hardware Development" and "Hardware Prototype on the Development Board" shown in Figure 1.

Refer to the *Nios Software Development Tutorial* for a complete explanation of the software flow.

Figure 1. Hardware/Software Development Flow for a Nios Processor System

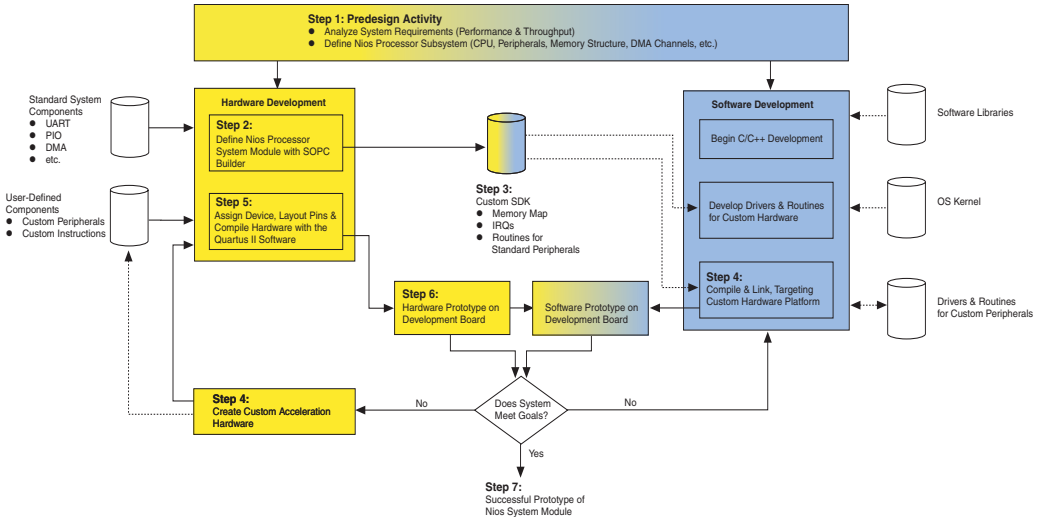


Figure 1 shows where the hardware and software flows intersect. To obtain a complete, working system, it is important to know what each side must provide for the other. Even if your development involves separate teams for hardware and software design, it is helpful to understand the design flow on both sides of the hardware-software divide.

The development flow begins with predesign activity (step 1 in Figure 1), which includes an analysis of the system requirements:

- What computational performance does the design require?
- How much bandwidth or throughput must the system handle?

Based on the answers to these questions, you can determine the concrete system requirements:

- Will the CPU need a hardware-accelerated multiplier?
- Which peripherals, and how many of each, does the design require?
- Could DMA channels be used to free up CPU cycles spent copying data?

These decisions involve both the hardware and software teams.

Hardware Development Flow

The hardware design process begins by using the SOPC Builder system integration software to choose the appropriate CPU, memory, and peripheral components such as on-chip memory, PIOs, UARTs, and off-chip memory interfaces and then customize their functionality (step 2 in Figure 1). SOPC Builder allows you to easily connect custom hardware components to your system, giving you powerful options to accelerate system performance. SOPC Builder takes this information and automatically integrates the system, outputting HDL files that describe the system hardware. SOPC Builder also generates a software development kit (SDK) that forms the foundation for software development for your custom Nios processor system (step 3 in Figure 1). The SDK can be used immediately to begin developing embedded software for the custom hardware.

You can create a custom instruction by designing hardware blocks that connect directly to the Nios CPU's arithmetic logic unit (ALU). Alternately, you can create a custom peripheral designed specifically for a critical computation or data movement task (step 4 in Figure 1).

Next, you use the Quartus II software to target a specific Altera device and place-and-route the HDL design files generated by SOPC Builder. Using the Quartus II software, you choose the target Altera FPGA device, assign pin locations to the various I/O ports on the Nios system, and apply any hardware compilation options and/or timing constraints (step 5 in Figure 1). During compilation, Quartus II integrated synthesis generates a netlist from the HDL source files, and the Quartus II fitter fits the netlist into the target device. Finally, Quartus II generates a device configuration file to configure the FPGA.

Using the Quartus II programmer and an Altera download cable, you then download the configuration file (hardware image for your custom Nios processor system) into the development board (step 6 in Figure 1). After verifying that the design works in hardware, the new hardware image can be programmed into nonvolatile memory on the development board. After the board is programmed, the software team is ready to begin using the board as a prototype platform to verify software functionality on the processor hardware.

You now have a successful prototype of a Nios processor system running on the Nios development board (step 7 in Figure 1). Most hardware design flows will continue and integrate the HDL from SOPC Builder into a larger, single-chip SOPC design. The designer's preferred synthesis tool can then synthesize the complete design and the Quartus II software will place-and-route the netlist into a target device. The overall logic design can be prototyped on the Nios development board or, the design can be migrated to the designer's custom hardware system.

Notes:

This tutorial guides you through the steps required to create and instantiate the Nios system module using the SOPC Builder software.

To use the instructions in this section you need to be familiar with the Quartus II software interface, specifically the toolbars. Refer to the Quartus II help system for more information about using the Quartus II software.

Accessing a Quartus II Project

To begin, you must open the specific Quartus II project for the Nios development board you are using. To start the Quartus II software and create a new project, perform the following steps:

1. Choose **Programs > Altera > Quartus II** <*version*> (Windows Start menu).

2. Choose **Open Project** (File menu).

3. Browse to the **tutorials** directory for your board.

   Throughout this tutorial, we refer to the tutorial directory for your board as <*board specific tutorial*>. Each of the board-specific tutorials are found in the following directory: **c:\altera\kits\nios\tutorials** <*board specific tutorial*>. Below are the names of the board-specific tutorial directories:

   – Stratix 1S10 board—**\Nios_HW_Tutorial_Stratix_1S10**
   – Cyclone 1C20 board—**\Nios_HW_Tutorial_Cyclone_1C20**
   – Stratix 1S40 board—**\Nios_HW_Tutorial_Stratix_1S40**

4. Choose the **.quartus** file **nios_system_module** and click **Open**.

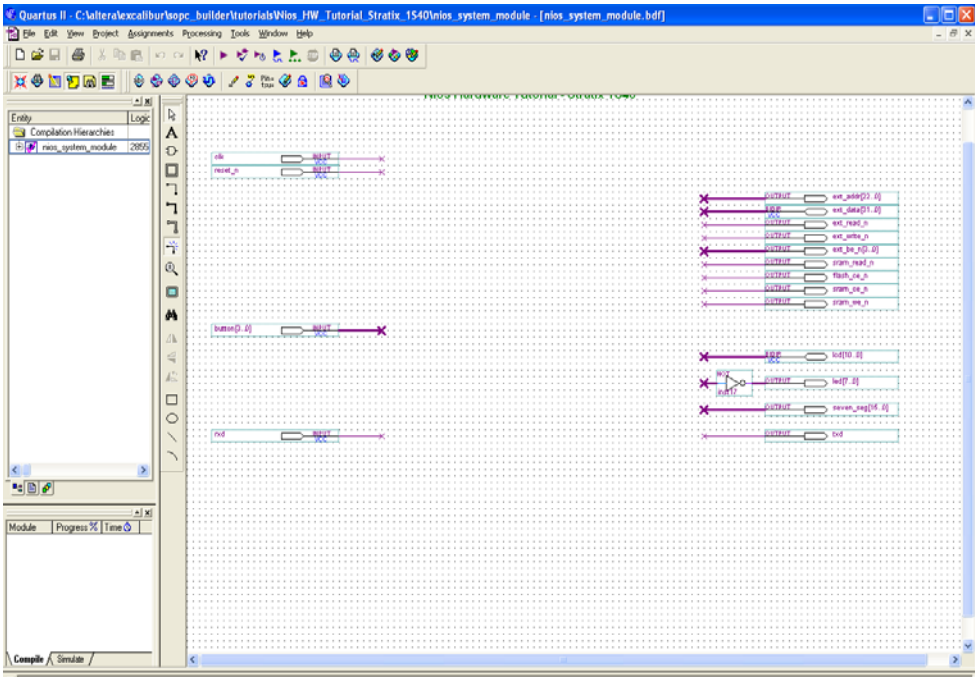
   The Quartus project will open with a block design file (BDF) showing unconnected named pin as shown in Figure 2.

5. *Stratix 1S10 board users only*: Early shipments of the board used ES (engineering sample) devices. Device configuration files for ES and non-ES devices are not compatible. The files for this tutorial use the non-ES device by default.

a. If the Stratix device on your board is labeled "EP1S10F780C6ES", then it is an ES device, and you must retarget the Quartus II design for the ES part number. **Select Device…** (Assignments menu) to change the device settings to the EP1S10F780C6ES.

b. If the device on your board is labeled "EP1S10F780C6" (no "ES"), then do not change the device settings. Starting with the Nios processor version 3.1, August 2003, the board ships with a non-ES device, and works as-is with the tutorial files.

Figure 2. Quartus II Project Block Design File



Create a Nios System Module

This section describes how to use SOPC Builder to create the Nios embedded processor, configure system peripherals, and connect these elements to make a Nios system. Next, you will connect the Nios system ports to the FPGA device pins (represented in the BDF) that correspond to the physical FPGA pins connected to hardware components on the Nios development board.

This section includes the following steps:

1. "Start SOPC Builder" on page 17.

2. "Add CPU & Peripherals" on page 19.

3. "Specify Base Addresses" on page 33.

4. "Generate the System Module" on page 35.

5. "Add the Symbol to the BDF" on page 38.

Start SOPC Builder

SOPC Builder is a software tool that allows you to create a full functioning custom embedded microcontroller called the Nios system module. A complete Nios system module contains a Nios embedded processor and its associated system peripherals. SOPC Builder allows you to easily and quickly create a multi-master system module (masters, slaves, bus arbitration logic, etc.) that is wired up and ready to use in Altera FPGAs.

SOPC Builder prompts you to select parameter settings and optional ports and peripherals. Once SOPC Builder generates the Nios system module, you instantiate it in the design file.

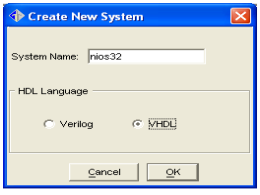
Follow these steps to start SOPC Builder:

1. In the Quartus II software, choose **SOPC Builder** (Tools menu). SOPC Builder starts, displaying the **Create New System** dialog box.

2. Type `nios32` in the **System Name** field (Figure 3) and under HDL Language choose **Verilog** or **VHDL**.

SOPC Builder generates plain text Verilog HDL or VHDL for all of its native components depending on which language you choose in this step.

Figure 3. Create New System Dialog Box

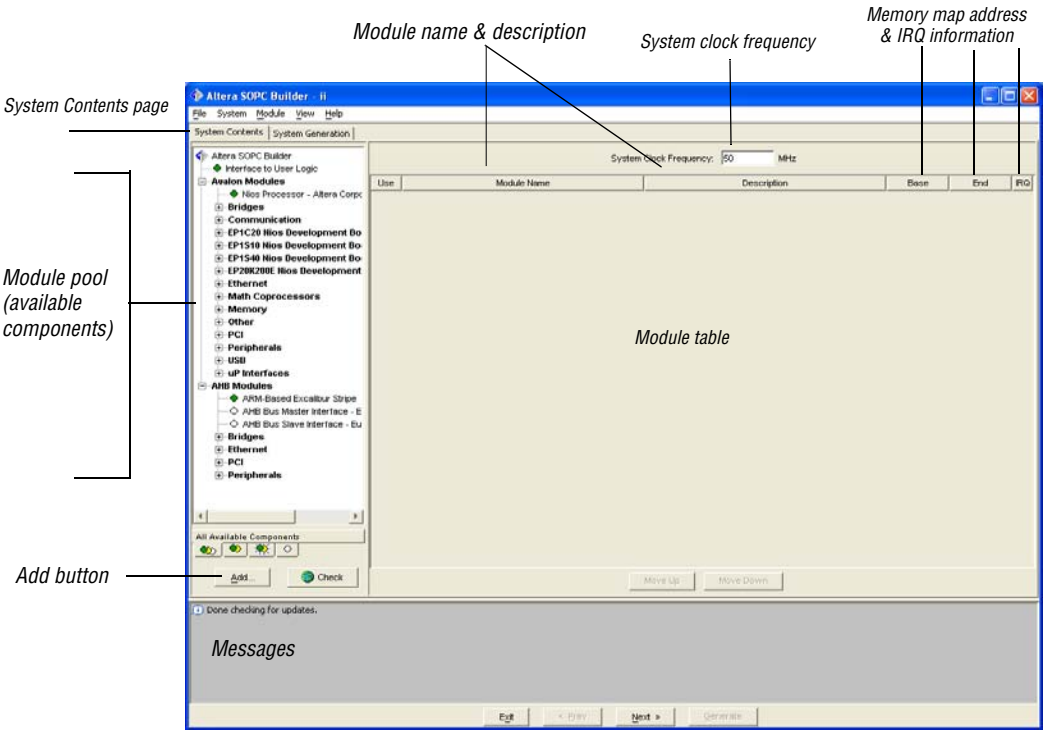


3. Click **OK**. The **Altera SOPC Builder - nios32** window appears and the **System Contents** tab is displayed.

You are now ready to add the Nios CPU and peripherals to your system. You will be populating the SOPC Builder table with the components that go into your final microcontroller system. Figure 4 on page 18 shows the SOPC Builder **System Contents** tab and SOPC Builder table for **nios32.**

For more information on the SOPC Builder, refer to the *SOPC Builder Data Sheet*.

Figure 4. SOPC Builder System Contents Page



System Speed

Before adding any peripherals, enter 50MHz in **System Clock Frequency** found at the top of the SOPC Builder page as shown in Figure 4. This value is used to achieve accurate timing in both hardware generation and software SDKs. It is critical that this clock frequency matches the frequency going into the Nios system module.

For this tutorial, we are simply using the clock input from the 50MHz oscillator on the board. However, if a PLL is used to change this frequency, or an external clock input is used, then the frequency going into the system module should be entered into **System Clock Frequency**.

Add CPU & Peripherals

The Nios system peripherals allow the Nios embedded processor to connect and communicate with internal logic in the FPGA, or external hardware on the Nios development board. Use SOPC Builder to specify the name, type, memory map addresses, and interrupts of the system peripherals for your Nios system module.

The following specifications ensure that the **nios_system_module** design functions correctly on the Nios development board, and allow you to run the software examples provided in your project's software development kit (SDK) directory.

You will add the following modules to the SOPC Builder:

- Nios 32-Bit CPU
- On-Chip Boot Monitor ROM
- Communications UART
- Timer
- Button PIO
- LCD PIO
- LED PIO
- Seven Segment PIO
- External RAM Bus (Avalon Tri-State Bridge)
- External RAM Interface
- External Flash Interface

The Cyclone development board example in Figure 5 on page 20 and Figure 6 on page 20 shows you the components you will be adding to SOPC Builder to create the Nios system design in this tutorial. In Figure 5 on page 20 and Figure 6 on page 20, the SOPC Builder component module names are called out in parentheses.

See the *Nios Development Board Reference Manual* for more detailed board component information.

The components you will be adding are located in the module pool located on the left-hand side of the **System Contents** tab in the **Altera SOPC Builder - nios32** window. See Figure 4 on page 18.

Figure 5. Nios Development Board

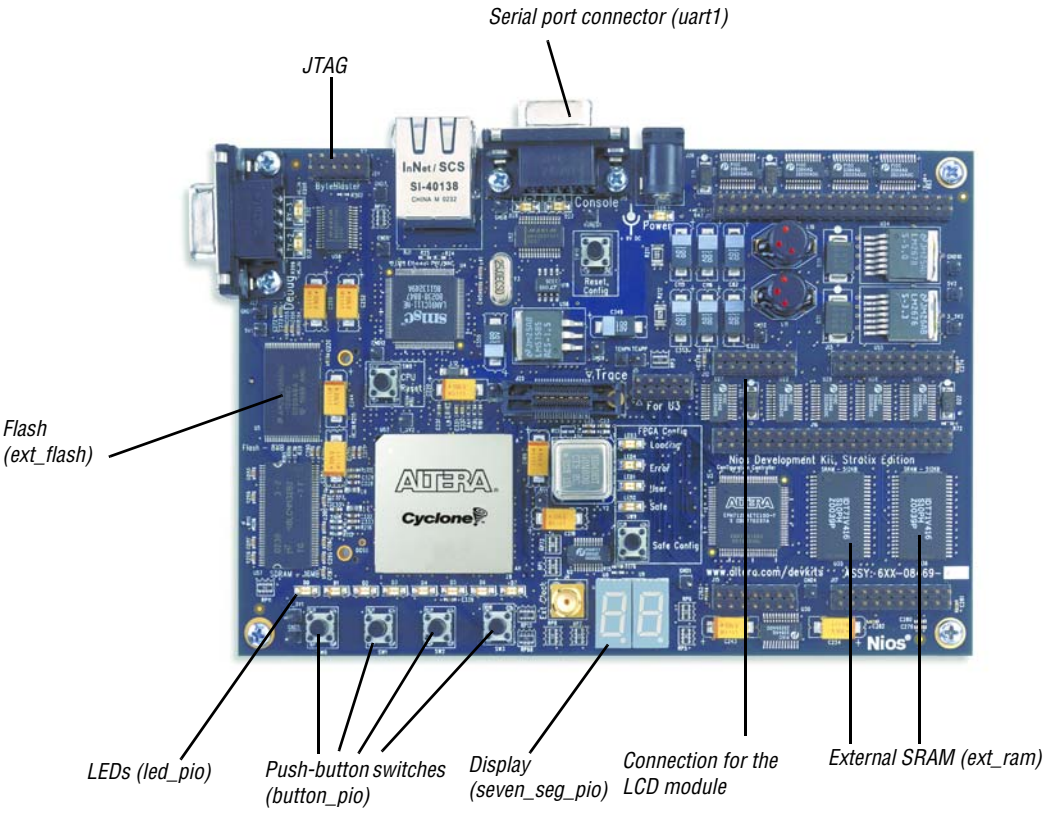
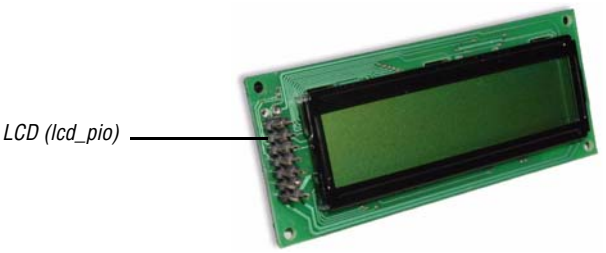


Figure 6. LCD Board



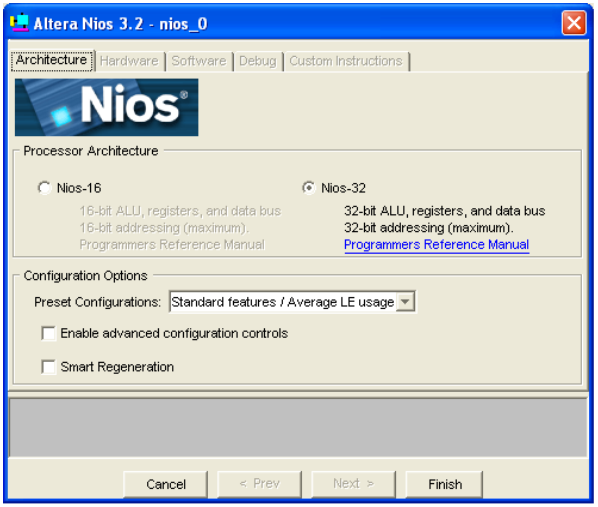
Nios 32-Bit CPU

To add the Nios 32-bit CPU, named **cpu**, to the `nios32` system, perform the following steps:

1. Under Avalon Modules, Select **Nios Processor**.
2. Click **Add**. The Nios configuration wizard titled **Altera Nios 3.0 - nios_0** displays.
3. Specify the following options in the **Architecture** tab (see Figure 7):
   - **Processor Architecture:** Nios-32
   - **Preset Configurations:** Standard features/Average LE usage

Selecting these configuration options automatically sets the options in the remaining Nios wizard tabs. If you want to view these options or to customize the settings, turn on the **Enable advanced configuration controls** option. For this tutorial, the default configuration is acceptable.

Figure 7. Nios CPU Architecture Tab



4. Click **Finish**. You are returned to the **Altera SOPC Builder - nios32** window.
5. Right-click **nios_0** under **Module Name**.

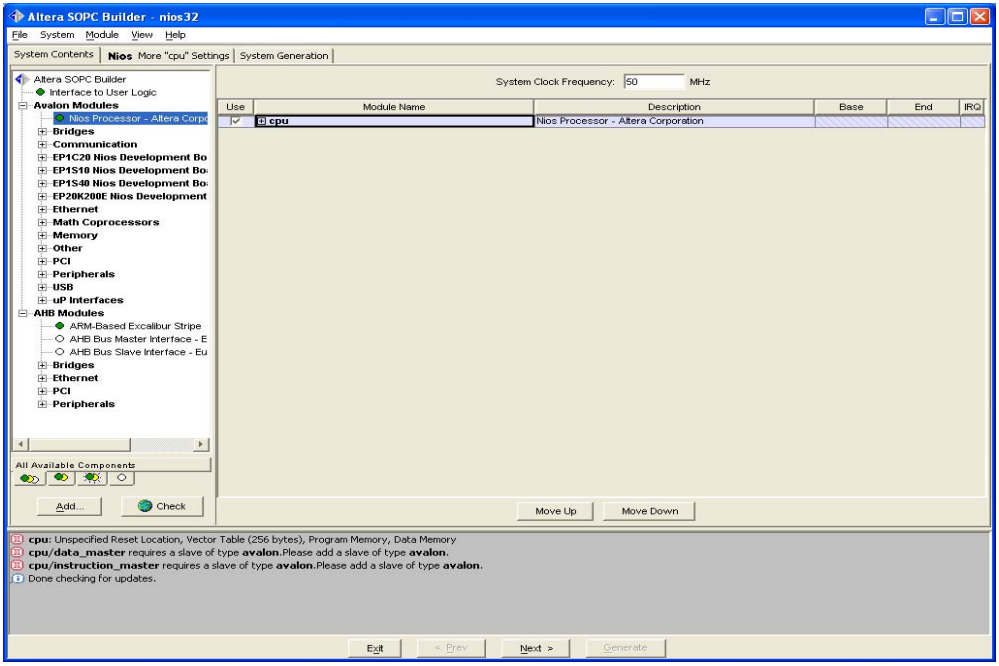
6. Choose **Rename** from the pop-up menu.

7. Rename **nios_0** as **cpu**. Press return when you are finished typing the new name to save your setting. See Figure 8.

Some of the peripherals in this tutorial MUST be renamed with the EXACT name provided. Because renaming a few of the peripherals is critical for successful tutorial completion, you will rename all the peripherals to avoid potential difficulties.

When designing your own Nios system, it is not necessary for you to rename peripherals, but helpful as a method for quickly identifying problem occurrences based on the peripheral's name. Also, assigning meaningful names helps the user understand the memory map peripheral-related problems during software development.

Figure 8. SOPC Builder with CPU



You will see errors in the SOPC Builder message display (Figure 8). The issues causing the errors to appear after adding the Nios CPU are resolved when the rest of the elements are added to the design. At this stage these error messages can be safely ignored.

On-Chip Boot Monitor ROM

To add the boot monitor ROM peripheral, **boot_monitor_rom**, perform the following steps:

1. Select **On-Chip Memory (RAM or ROM)** under **Memory** and click **Add**. The **On-chip Memory - onchip_memory_0** wizard displays.

2. Specify the following options in the **Attributes** tab (see Figure 9):

   – **Memory Type:** ROM (read-only)
   – **Block Type:** Automatic
   – **Data Width:** 32
   – **Total Memory Size:** 2 KBytes

   After specifying these attributes, a warning that the ROM is empty will be displayed in the bottom of the **On-chip Memory - onchip_memory_0** wizard. This is resolved in Steps 4.

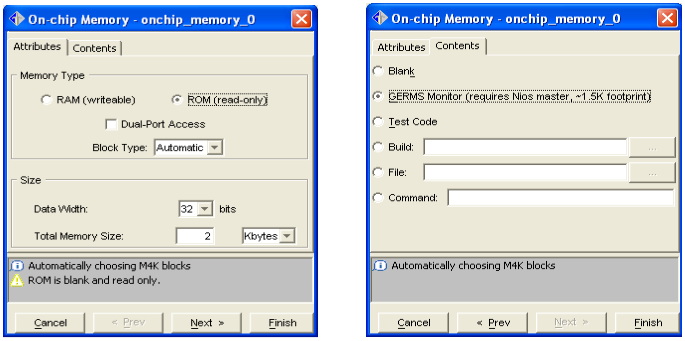
3. Click the **Contents** tab.

4. Select the **GERMS Monitor** option (see Figure 9).

   The GERMS monitor is a simple boot monitor program that allows you to look at memory and communicate with the processor on boot up. This will be explored later in this tutorial. You can find more information in the *Nios Software Development Reference Manual*.

Figure 9. Boot Monitor ROM Wizard Settings



5. Click **Finish**. You are returned to the **Altera SOPC Builder - nios32** window.

6. Rename **onchip_memory_0** to **boot_ROM**. See Steps 5–7 on page 21.

Communications UART

This Nios design includes one UART peripheral for output from the processor. To add the communications UART peripheral, **uart1**, perform the following steps:

1. Select **UART (RS-232 serial port)** under **Communication** and click **Add**. The **Avalon UART - uart_0** wizard displays.

2. Specify these options in the **Configuration** tab (see Figure 10):

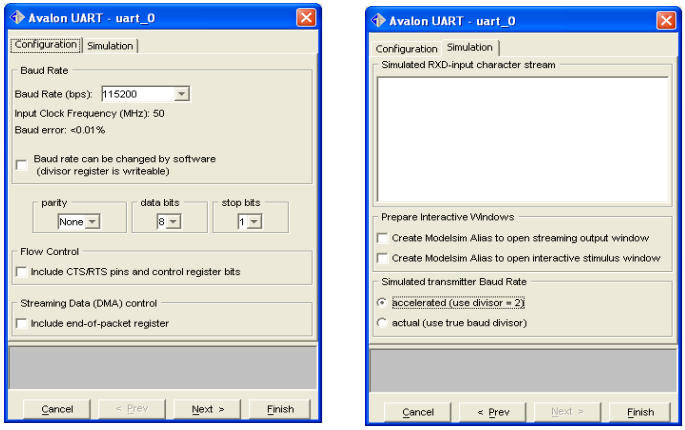
   – **Baud Rate (bps):** 115200
   – **Parity:** None
   – **Data Bits:** 8
   – **Stop Bits:** 1

3. Click the **Simulation** tab.

4. Select the **accelerated (use divisor = 2)** option (see Figure 10).

UARTS run slowly compared to the typical Nios system clock speed. Even at 115,200 baud, simulating sending or receiving a single character to a UART takes a long time. The UART peripheral has an option, **accelerated (use divisor = 2)**, that allows you to simulate UART transactions as if the baud rate was 1/2 the system clock speed (making much faster simulation times).

Figure 10. Communications UART Wizard Settings



1. Click **Finish**. You are returned to the **Altera SOPC Builder - nios32** window.

2. Rename **uart_0** to **uart1**. See Steps 5–7 on page 21.

Timer

Like the Nios CPU, the timer peripheral has several preset configurations that trade off between logic element (LE) usage and configurability. For example, a simple interval timer uses few LEs, but it is not configurable. In contrast, the full-featured timer is configurable, but uses more LEs. You can use one of the preset configurations or make your own custom settings.

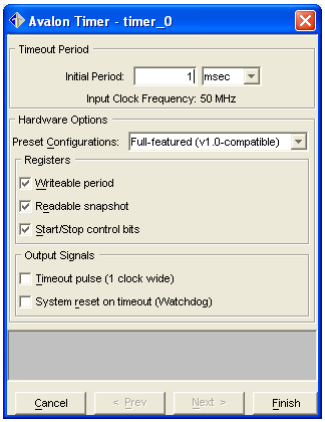
To add the timer peripheral, **timer1**, perform the following steps:

1. Select **Interval timer** under **Other** and click **Add**. The **Avalon Timer - timer_0** wizard displays.

2. Specify the following options (see Figure 11):

   – **Initial Period** under **Timeout Period:** 1 msec
   – **Preset Configurations:** Full-featured (v1.0-compatible)

Figure 11. Timer Wizard Settings



3. Click **Finish**. You are returned to the **Altera SOPC Builder - nios32** window.

4. Rename **timer_0** to **timer1**. See Steps 5–7 on page 21.

The timer software library subroutines rely on a timer named **timer1**. If you do not use the name **timer1**, these routines will not be compiled into the library.

Button PIO

To provide an interface for the buttons on the Nios development board, add the button PIO peripheral, **button_pio**, by performing the following steps:

1. Select **PIO (Parallel I/O)** under **Other** and click **Add**. The **Avalon PIO - pio_0** wizard displays.

2. Specify the following options (see Figure 12):

   – **Width:** 4 bits
   – **Direction:** Input ports only

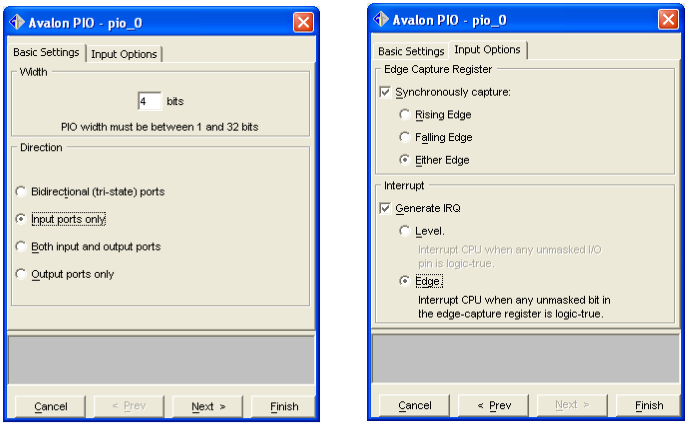
   When you select the **Input ports only** option, the **Input Options** tabs is enabled.

3. Click the **Input Options** tab.

4. Turn on **Synchronously capture** under **Edge Capture Register**.

5. Select **Either Edge**.
6. Turn on **Generate IRQ** under **Interrupt**.
7. Select **Edge**.

Figure 12. Button PIO Wizard Settings



8. Click **Finish**. You are returned to the **Altera SOPC Builder - nios32** window.
9. Rename **pio_0** to **button_pio**. See Steps 5–7 on page 21.

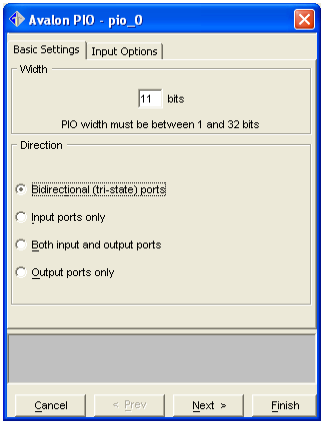
LCD PIO

To provide an interface to the LCD panel, add the LCD PIO peripheral, **lcd_pio**, by performing the following steps:

1. Select **PIO (Parallel I/O)** under **Other** and click **Add**. The **Avalon PIO - pio_0** wizard displays.
2. Specify the following options (see Figure 13):
   - **Width:** 11 bits
   - **Direction:** Bidirectional (tri-state) ports

   The LCD module that comes with the Nios development kit can be written to and read from. Therefore, the LCD PIO uses bidirectional pins.

Figure 13. LCD PIO Wizard Settings



3. Leave the settings in the **Input Options** tab at the defaults.

4. Click **Finish**. You are returned to the **Altera SOPC Builder - nios32** window.

5. Rename **pio_0** to **lcd_pio**. See Steps 5–7 on page 21.

LED PIO

To provide an interface for the LEDs on the Nios development board, add the LED PIO peripheral, **led_pio**, by performing the following steps:

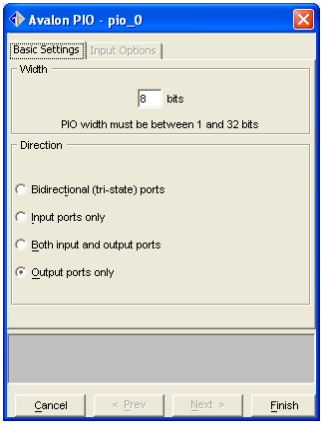
1. Select **PIO (Parallel I/O)** under **Other** and click **Add**. The **Avalon PIO - pio_0** wizard displays.

2. Specify the following options (see Figure 14):

   – **Width:** 8 bits
   – **Direction:** Output ports only

   In this tutorial, the LED PIO uses outputs only and has an inverter between the system module and the pins that are connected to the LEDs. Therefore, when the tutorial design is downloaded to the FPGA, the LEDs are illuminated, indicating that the correct design is running on the board.

Figure 14. LED PIO Wizard



3. Click **Finish**. You are returned to the **Altera SOPC Builder - nios32** window.

4. Rename **pio_0** to **led_pio**. See Steps 5–7 on page 21.

Seven Segment PIO

To add the seven segment PIO peripheral, **seven_seg_pio**, perform the following steps:

1. Select **PIO (Parallel I/O)** under **Other** and click **Add**. The **Avalon PIO - pio_0** wizard displays.

2. Specify the following options:

   – **Width:** 16 bits
   – **Direction:** Output ports only

3. Click **Finish**. You are returned to the **Altera SOPC Builder - nios32** window.

4. Rename **pio_0** to **seven_seg_pio**. See Steps 5–7 on page 21.

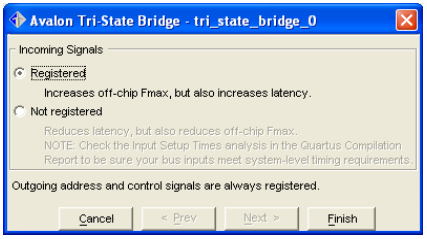
External RAM Bus (Avalon Tri-State Bridge)

SOPC Builder uses the Avalon interface to connect on-chip components with Avalon master or slave ports. For example, the Nios CPU has an Avalon master port and the UART component has an Avalon slave port. For the Nios system to communicate with memory external to the FPGA on the Nios development board, you must add a bridge between the Avalon bus (the local connection interface for SOPC Builder-generated systems) and the bus or buses to which the external memory is connected.

To add the Avalon tri-state bridge, **ext_Shared_Bus**, perform the following steps:

1. Select **Avalon Tri-State Bridge** under **Bridges** and click **Add**. The **Avalon Tri-State Bridge - tri_state_bridge_0** wizard displays. See Figure 15. The **Registered** option is turned on by default.

Figure 15. Avalon Tri-State Bridge Wizard



2. Click **Finish**. You are returned to the **Altera SOPC Builder - nios32** window.

3. Rename **tri_state_bridge_0** to **Ext_Shared_Bus**. See Steps 5–7 on page 21.

External RAM Interface

The development board choices in SOPC Builder include interface selections for various components included on the development board. One of the board interface selections is memory devices. You will now choose the memory device for your board. To add the external RAM peripheral, **SRAM_1MByte**, perform the following steps:

1. Click the "+" next to your development board from the choice of development boards in the **System Contents** tab and select the SRAM component for your board as shown below:

- Stratix 1S10 board: IDT71V416 SRAM
- Stratix 1C20 board: IDT71V416 SRAM
- Stratix 1S40 board: IDT71V416 SRAM

2. Click **Add**. The **SRAM (two IDT71V416 chips) - nios_dev_board_sram32...** wizard displays.

3. In the **Attributes** tab, make sure the memory size is set at 1024 kb (Figure 16).

4. Click the **Simulation** tab.

5. Select **Build**.

The build option specifies a Nios program that will automatically compile when you generate your system.

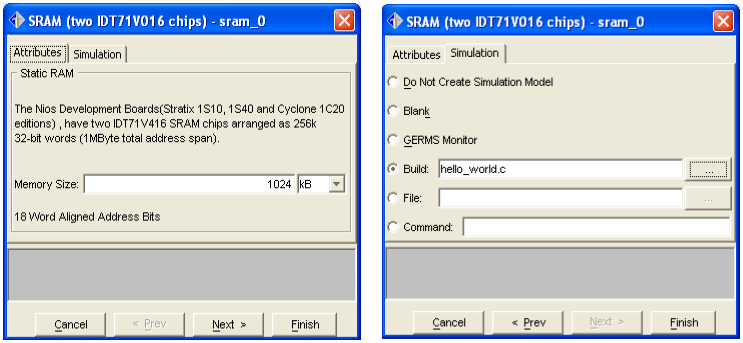
6. Click the **Browse** (**...**) button.

7. Select the **hello_world.c** file, and click **Open** (see Figure 16).

The purpose of this step is to show you how easily you can add a program file that you want to simulate running on your Nios system. By adding your **.c** file here, SOPC Builder will take care of creating the test bench needed to watch an executing software program in hardware simulation.

See *AN 189: Simulating Nios Embedded Processor Designs* for more information.

Figure 16. External RAM Wizard Settings



8. Click **Finish**. You are returned to the **Altera SOPC Builder - nios32** window.

9. Rename **sram_0** to **SRAM_1MByte**. See Steps 5–7 on page 21.

External Flash Interface

To add the external flash peripheral, **Flash_8MByte**, perform the following steps:

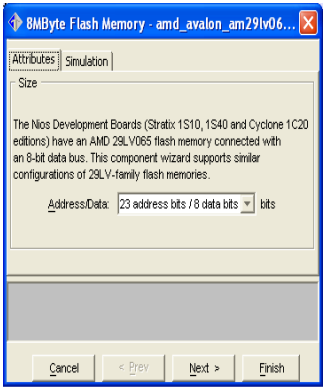
1. Click the "+" next to the development board you have and then select the external flash peripheral for your development board as shown below.

   Stratix 1S10 board: AMD29LV065D flash
   Cyclone 1C20 board: AMD29LV065D flash
   Stratix 1S40 board: AMD29LV065D flash

2. Click **Add** and the **8Mbyte Flash Memory - amd_avalon_am29LV065d_flash_0** wizard displays.

3. Choose **23 address bits / 8 data bits** from the **Address/Data** drop-down list box (Figure 17).

Figure 17. External Flash Wizard



4. Click **Finish**. You are returned to the **Altera SOPC Builder - nios32** window.

5. Rename **amd_avalon_amLV065d_flash_0** to **Flash_8MByte**. See Steps 5–7 on page 21.

You are finished adding peripherals. In the remaining Design Entry sections, you will set options in SOPC Builder.

Specify Base Addresses

SOPC Builder assigns default base address values for the components in your Nios system module. However, you can modify these defaults. For this tutorial, you will set and lock the Flash base address to 0 but allow SOPC Builder to change the other addresses.

This tutorial sets the base address of the Flash to 0x0000 to make it simple to talk about hardware and software configurations stored at certain address locations in the Flash memory, without worrying about offsets.

Setting the Flash Base Address

To set the Flash base address, do the following.

1. In the SOPC Builder module table, click on the **Base** for the **Flash_8MByte** peripheral, type `0x0` and press Enter.

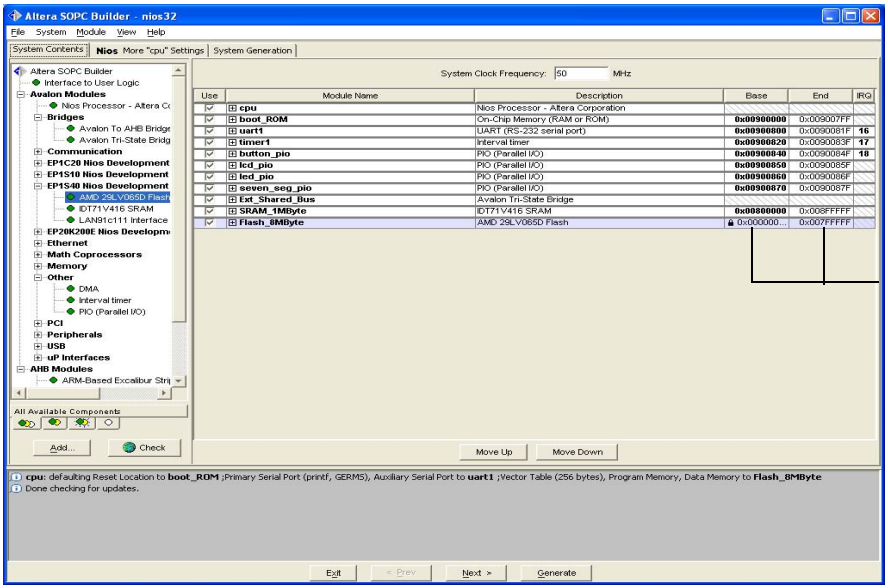
   Errors will appear in the SOPC Builder message area because the new Flash base address has conflicts with another peripheral's base address. This is resolved in Steps 3.

   Verify Flash is selected in the SOPC Builder Module table. The Flash peripheral will be highlighted.

2. Choose **Lock Base Address** (Module menu). A padlock icon appears next to the Flash Base Address.

3. Choose **Auto Assign Base Address** (System menu). **Auto Assign Base Address** causes SOPC Builder to reassign base addresses for any unlocked address to avoid address map conflicts. The errors that occurred in Steps 1 should now be resolved and no errors should appear in the message area. Figure 18 shows the completed system with its address map.

Figure 18. Final System with Address Map& Nios System Settings



Verify the Base Address & IRQ Settings

After you add the components to the system module, you must make the following system settings.

1. Click the **Nios More "cpu" Settings** tab. The text in quotation marks is the name of the Nios CPU module, which in this example is **cpu**.

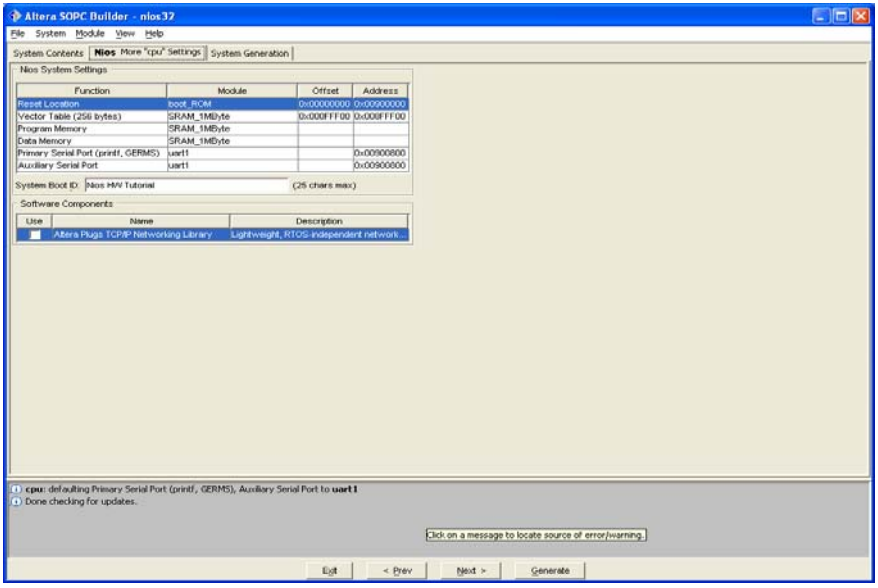
2. Make the following settings under **Nios System Settings** see Figure 19 on page 35):

   – Reset Location
     - **Module:** boot_ROM
     - **Offset:** 0x0
   – Vector Table (256 bytes)
     - **Module**: SRAM_1MByte
     - **Offset:** 0x000FFF00
   – Program Memory
     - **Module:** SRAM_1MByte
   – Data Memory
     - **Module:** SRAM_1MByte
   – Primary Serial Port (printf, GERMS)
     - **Module:** uart1
   – Auxiliary Serial Port
     - **Module:** uart1
   – **System Boot ID:** Nios HW Tutorial

For the **System Boot ID**, specify any 25-character string you prefer. This text is sent over the UART by the Nios CPU on boot-up.

3. Under **Software Components** make sure none of the options are selected. The other software components are not used in this tutorial.

Figure 19. SOPC Builder More 'cpu' Settings Tab



Generate the System Module

To make your Nios design a part of the Quartus II project that will be compiled for the FPGA device on the development board, you must first generate the design logic.

To generate the design, perform the following steps:

1. Click the **System Generation** tab if you have not already.

2. Make the following settings under **Options** in the **System Generation** tab (See Figure 20):

   – **SDK:** Turn on

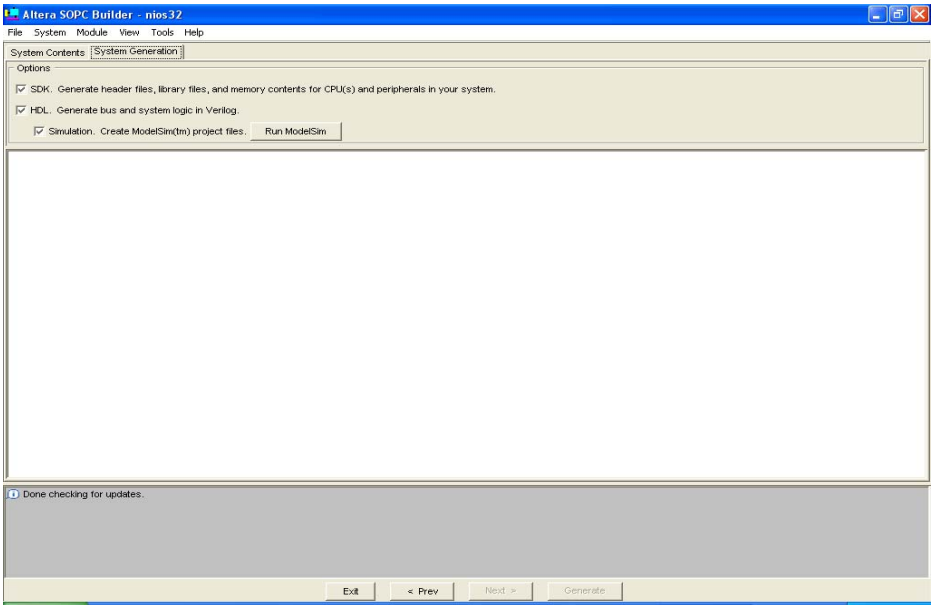
For more information on the files that are generated in the SDK, refer to the *Nios Embedded Processor Software Development Reference Manual*.

– **HDL:** Turn on.

– **Simulation:** Turn on if you have the ModelSim software installed on your PC.

For more information on the simulation files that are generated, refer to *AN 189: Simulating Nios Embedded Processor Designs*.

Figure 20. SOPC Builder System Generation Tab



3. Click **Generate**.

   SOPC Builder performs a variety of actions during design generation, depending on which options you have specified. For the design created using this tutorial, which has all available system generation options turned on, the SOPC Builder performs the following actions:

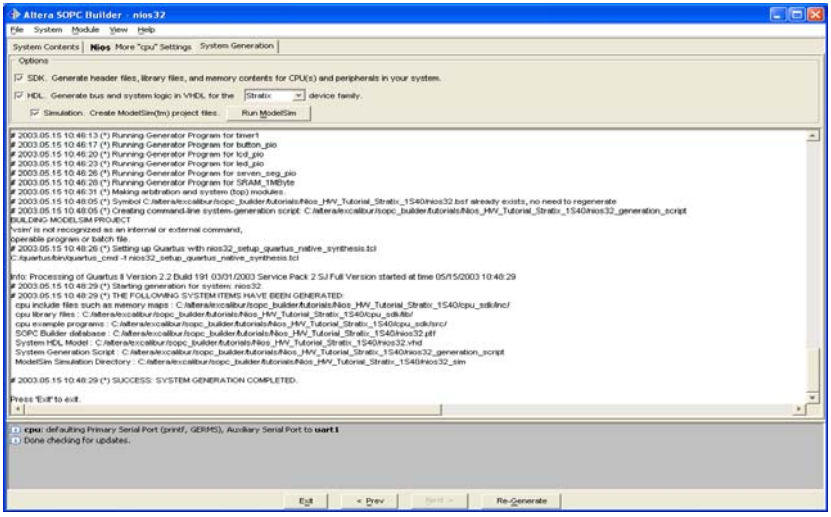
   – Generates Verilog HDL or VHDL source files
   – Creates the simulation project and source files
   – Generates the SDK, C, and Assembly language header and source files for the options chosen in the **Nios More "cpu" settings** tab.
   – Compiles the custom software library for the hardware in your system
   – Compiles the GERMS monitor used in the boot monitor ROM

   During generation, information and messages appear in the message box in the **System Generation** tab.

   System generation will probably take between three to five minutes.

4. When generation is complete (see Figure 21), the `SYSTEM GENERATION COMPLETED` message displays. Click **Exit** to exit SOPC Builder.

Figure 21. System Generation Completes

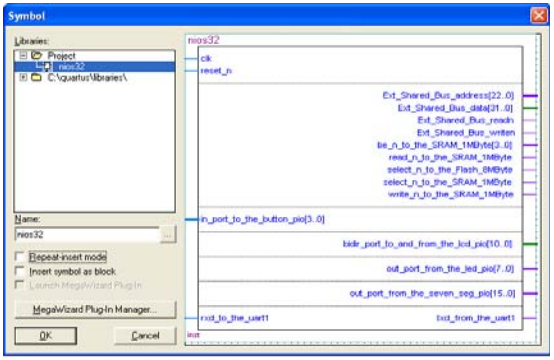


Add the Symbol to the BDF

During generation, SOPC Builder creates a symbol for your Nios system module. You can add the `nios32` symbol to your BDF. To add the symbol, perform the following steps:

1. Return to the Quartus II software and double-click anywhere inside the BDF window. The **Symbol** dialog box appears (Figure 22).

Figure 22. Symbol Dialog Box



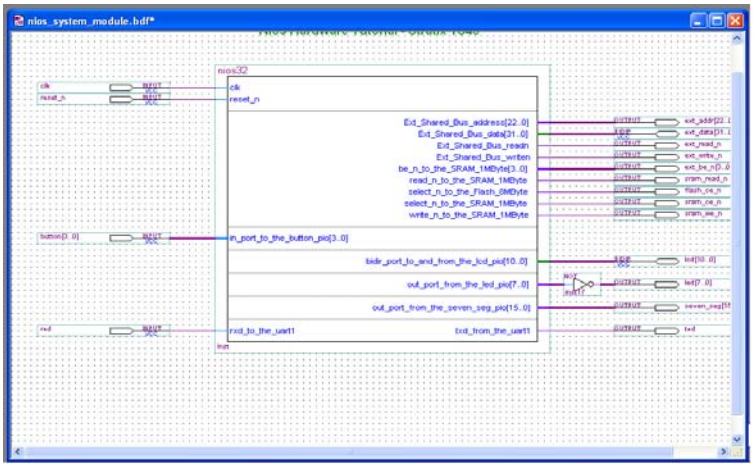
2. In the **Symbol** dialog box, click **Project** to expand the Project symbol directory.

3. Under Project, choose **nios32**. A large symbol will appear representing the Nios system you just created.

4. Click **OK**. The **Symbol** dialog box closes and an outline of the `nios32` symbol is attached to the pointer.

5. Place the symbol so it lines up with the pins that are already in the block design schematic files as seen in Figure 23 on page 39.

Figure 23. Adding the nios32 Symbol



6. Choose **Save** (File menu).

Compile the Design

During compilation the Compiler locates and processes all design and project files, generates messages,and reports related to the current compilation, and creates the SOF file and any optional programming files.

To compile the **nios_system_module** design, follow these steps:

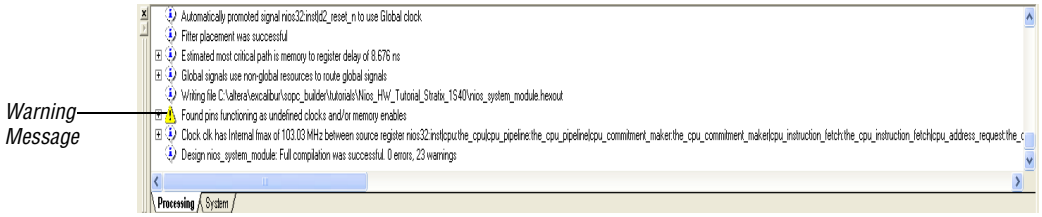
1. Choose **Start Compilation** (Processing menu). You can optionally click the Compile toolbar button.

   If you get a message asking if you want to save the changes you made to the BDF file, choose **Yes**.

   The Compiler immediately begins to compile the **nios_system_module** design entity, and any subordinate design entities, using the **nios_system_module** Compiler settings. As the design compiles, the Status window automatically displays, as a percentage, the total compilation progress and the time spent in each stage of the compilation. The results of the compilation are updated in the Compilation Report window. The total compilation time may be 10 minutes or more, depending on the processing power of your computer, its amount of available memory, and the complexity of your SOPC design.

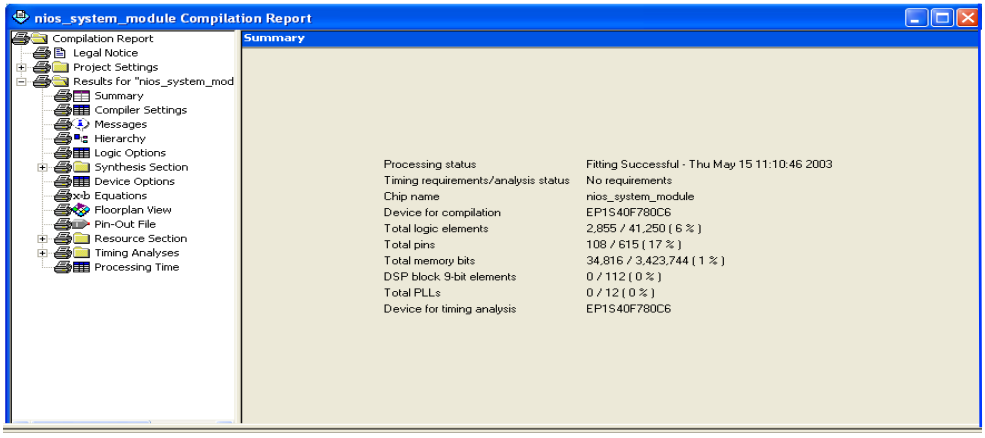
The Compiler may generate one or more of the following warning messages that do not affect the outcome of your design (see Figure 24).

Figure 24. Compiler Messages



2. When compilation completes, you can view the results in the **nios_system_module Compilation Report** window. See Figure 25.

Figure 25. Compilation Report



If the Compiler displays any error messages, you should correct them in your design and recompile it until it is error-free before proceeding with the tutorial. You can select the message and choose **Locate** (right button pop-up menu) to find its source(s), and/or choose **Help** (right button pop-up menu) to display help on the message.

Refer to the Compilation module in the Quartus II on-line tutorial for more information about viewing compilation results.

After a successful compilation, the Quartus II Compiler generates one or more programming files that the Programmer can use to program or configure a device. You can download configuration data directly into the FPGA with the ByteBlaster communications cable connected to the JTAG port (J24) on your Nios development board. You can also download configuration data to the flash memory device on the Nios development board over either the JTAG or serial port using the utilities provided. This method allows you to configure the FPGA using the data stored in flash memory.

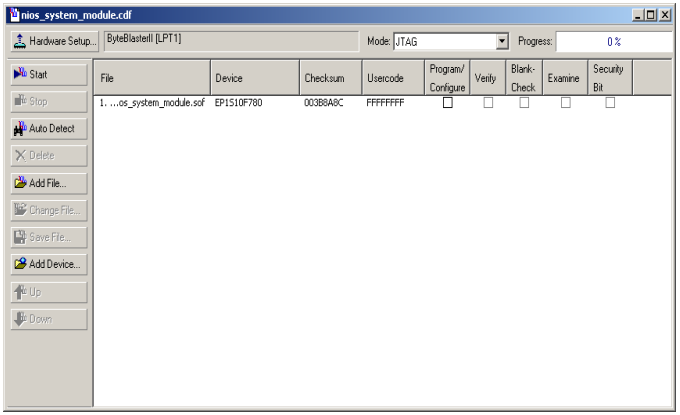
Configure the FPGA

Once you have properly connected and set up the ByteBlaster cable to transmit configuration data over the JTAG port, you can configure the FPGA immediately upon power up on the Nios development board with your design.

To configure the FPGA on the Nios development board with the **nios_system_module** design, follow these steps:

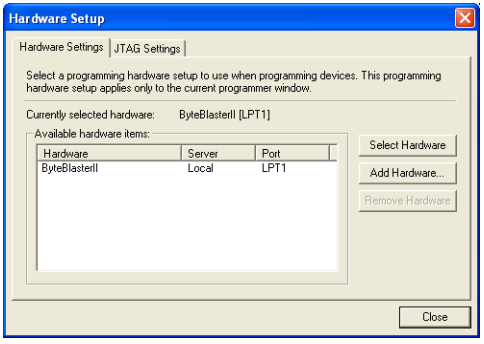
1. Choose **Programmer** (Tools menu). The Programmer window opens as shown in Figure 26.

Figure 26. JTAG Chain



2. Choose **Save As** (File menu).
3. In the **Save As** dialog box, type `nios_system_module.cdf` in the **File name** box.
4. In the **Save as type** list, make sure **Chain Description File** is selected.
5. Click **Save**.
6. In the **Mode** list of the Programmer window, make sure **JTAG** is selected.
7. Click **Hardware Setup...** to configure the programming hardware. The **Hardware Setup** dialog box appears as shown in Figure 27.

Figure 27. Hardware Setup

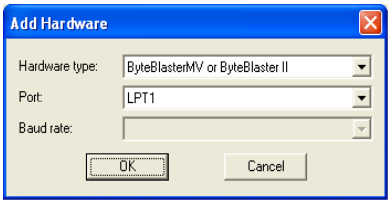


8. In the **Hardware Type** list, select **ByteBlaster or ByteBlaster II**.

   If the **ByteBlasterMV or ByteBlaster II** option does not appear in the **Hardware Type** list, do the following:

   a. Click **Add Hardware**. The **Add Hardware** dialog box appears as shown in Figure 28 on page 43.

Figure 28. Add Hardware



b. Select **ByteBlasterMV or ByteBlaster II** from the **Hardware type** list and click **OK**.

c. Now Select **ByteBlaster** and then click **Select Hardware**. The Programming Hardware will now display your selection.

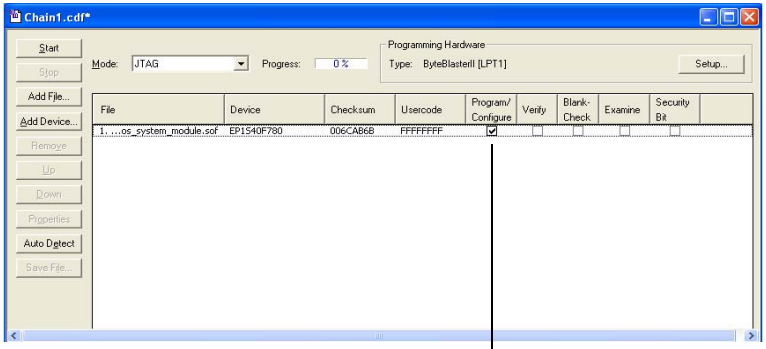
Refer to "Installing the ByteBlasterMV & ByteBlaster II Parallel Port Download Cable" in the *Quartus II Installation & Licensing Manual for PCs* for more information.

8. In the **Port** list, select the port that is connected to the ByteBlaster cable. Click **OK**.

9. Click **Close** to exit the **Hardware Setup** window.

10. In the Programmer window, turn on **Program/Configure**. See Figure 29 on page 44.

Figure 29. Turn on Program/Configure Option



11. Click **Start**. The Programmer begins to download the configuration data to the FPGA. The **Progress** field displays the percentage of data that is downloaded. A message appears when the configuration is complete.

When the design is successfully downloaded to the Altera device, the following events occur:

- If hardware configuration completes successfully, the D0-D7 LEDs are illuminated on the Nios development board.
- The GERMS monitor, which is stored in the boot_monitor_rom peripheral runs. The GERMS monitor performs the following system initialization tasks:
  - Disables interrupts on the UART, timer, and switch PIO.
  - Sets the Stack Pointer register to the top of RAM.
  - Examines two flash memory bytes at `Flash_base` + `0x4000C` for executable code (it looks for `N` and `i`, the first two letters spelling Nios).
- When the GERMS monitor determines that the flash memory bytes at `Flash_base` + `0x40000` contain `N` and `i`, it executes a call to location `Flash_base` + `0x40000` and runs the program here.

  The check at `Flash_base` + `0x4000C` can be changed in the GERMS source code if desired.

If you are unable to configure the device correctly, you can press the Safe Config or Force Config button (SW9) on the Nios development board to reload the factory default reference design and continue the tutorial.

Custom Microcontroller —No Way!

Think about it! In a short period of time, you created a custom microcontroller from scratch and compiled it to run on a FPGA!

The microprocessor lights the LEDs by default and runs the GERMS monitor which is a boot monitor program that waits for commands transmitted over the serial port. Such commands are often stored in an executable file that you download and run on your microprocessor. You will be using some of those in the next section. The important point is **you have created your own microcontroller**.

Running Software on Your Nios System

Now that you have downloaded the tutorial hardware design to the Nios development board, you must verify that it works properly and runs compiled code. Then, you can store the tutorial design in the on-board flash memory. In this section, you will compile sample code that the SOPC Builder generated. This code is automatically placed in your project's SDK directory. After you compile the code, you will download it and run it on the tutorial system module that you loaded into the FPGA.

This section contains the following:

1. "Nios SDK Shell Tips" on page 46
2. "Start the Nios SDK Shell" on page 47
3. "Compile & Run the Sample hello_nios.srec Test Program" on page 48

Nios SDK Shell Tips

If you like typing, skip this section. Below are some shortcuts and tips when using the **Nios SDK Shell** in the following sections:

- The **Nios SDK Shell** opens to the **/altera/kits/nios/examples** directory by default. To change to your SDK directory, type:

  `cd ../tutorials/<board specific tutorial>/cpu_sdk/src` ↵

- The **Nios SDK Shell** supports command completion of unique commands with the Tab key and pattern matching with the * key. Therefore, instead of typing a whole string, you can type a few letters. For example:
  - Instead of typing the word tutorials, type `tut` and press the Tab key.
  - Instead of typing <*Nios CPU name*>`_sdk`, type `*sdk`.

- Using these keyboard shortcuts, the command to change to the Nios tutorial directory is:

  `cd ../tut`<*Tab Key*>`/Nios_HW`<*Tab Key*>`/*sdk`<*Tab Key*>`/src` ↵

- As a shortcut, you can type **nb** instead of **nios-build**. For example:

  `nb hello_nios.c` ↵

- As a shortcut, you can type **nr** instead of **nios-run**. For example:

```
nr hello_nios.srec ↵
```

Start the Nios SDK Shell

The **Nios SDK Shell** is a UNIX-like command shell that allows you to do the following:

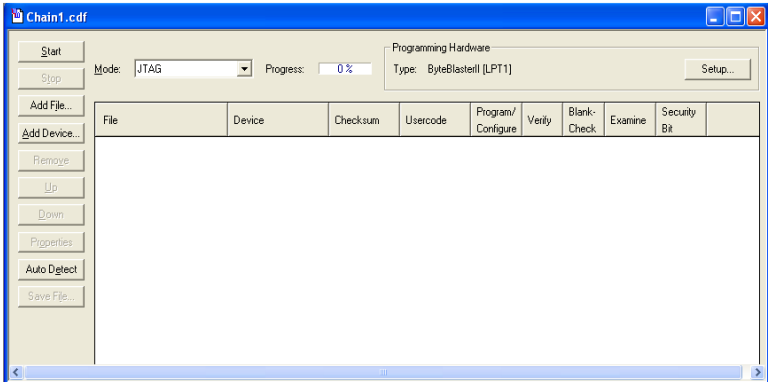
- Build code
- Download code to the Nios development board
- Run utilities (e.g. **nios-build** and **nios-run)**
- Run various test programs on the Nios development board

For more detailed information about software utilities, refer to the *Nios Software Development Reference Manual*.

To start the **Nios SDK Shell**, do the following:

Choose **Programs > Altera > Nios Development Kit** <*installed version*> **> Nios SDK Shell** (Windows Start menu). The **Nios SDK Shell** window appears. The **Nios SDK Shell** window displays some text, including path information and some messages about sample programs. See Figure 30.

Figure 30. Nios SDK Shell



Compile & Run the Sample hello_nios.srec Test Program

You can compile and run the Altera-provided **hello_nios.c** program to test the functionality of the **nios_system_module** design you downloaded into the FPGA. The **hello_nios.c** program is located in the **c:\altera\kits\nios\tutorials\Nios_HW_Tutorial_**<*Nios board version*>**\cpu_sdk\src** project subdirectory. You can use the **nios-build** and **nios-run** utilities to compile the **hello_nios.c** program and run it on your Nios system module.

To compile and run the sample **hello_nios.c** test program from the **Nios SDK Shell**, follow these steps:

1. To change to the appropriate project subdirectory, type the following command at the **Nios SDK Shell** prompt:

   ```
   cd c:/Altera/kits/nios/tutorials/
      Nios_HW_Tutorial_<Nios board version>/cpu_sdk/src ↵
   ```

   You must use the "/" character instead of the "\" character as a directory separator in the **Nios SDK Shell** window.

2. Type the following command at the **Nios SDK Shell** command prompt:

   ```
   nios-build hello_nios.c ↵
   ```

   The **nios-build** utility compiles the C code in the **hello_nios.c** file and generates the **hello_nios.srec** file.

3. To download to the board and run the **hello_nios.srec** program, type the following command at the **Nios SDK Shell** prompt:

   ```
   nios-run -r -p com<com port number> hello_nios.srec ↵
   ```

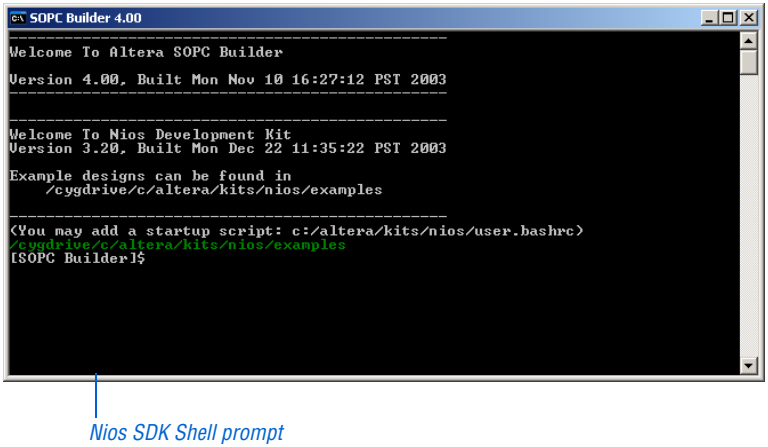
   

   If you do not specify a COM port with the -p com<*com port number*> text, the **nios-run** utility uses COM1 by default.

   The **nios-run** utility sends the executable code via the COM port, then runs the **hello_nios.srec** program on the Nios system module you created. This program generates the message `Hello, from Nios!` and causes the dual 7-segment LEDs (U8 and U9) to count down from 99 to 00. The Nios processor resumes execution of the GERMS monitor when the **hello_nios.srec** program is complete. See Figure 31 on page 49.

For more information on the nios-run and nios-build commands, refer to the *Nios Embedded Processor Software Development Reference Manual*.

Figure 31. Run hello_nios.srec Program



✓ When the **hello_nios.srec** program is complete, press the CPU Reset button (SW8) on the Nios development board to clear the **hello_nios.srec** program from the Nios embedded processor.

The CPU Reset button (SW8) is tied to the reset pin in the Nios system module. Pressing the CPU Reset button is the same as performing a power-on-reset on the microprocessor. Pressing the CPU Reset button does not reconfigure the FPGA.

Download the Design to Flash Memory

You can store configuration data in the flash memory device provided on the Nios development board. This section describes how to use the **hexout2flash** utility to convert the **.hexout** hardware configuration file created in the Quartus II software into a new file for the GERMS monitor. The GERMS monitor uses this new file to erase the user-configuration section of the flash memory on the Nios development board. Once erased, the GERMS monitor then downloads the **nios system module** configuration data to the user-configuration section of flash memory device on the Nios development board.

To download configuration data to the flash memory device on the Nios development board, follow these steps:

1. Make sure you have the tutorial hardware design running on the Nios development board. The easiest way to tell if the design is running is by looking at the D0 -D7 LEDs. They should all be illuminated.

2. Change to your Quartus II project directory from the **cpu_sdk/src** directory by moving up two levels:

   ```
   cd ../..
   ```

3. Type one of the following commands depending on which board you are using to create a `hexout2flash nios_system_module.hexout` file.

   For the Nios Stratix 1S10 board type:

   ```
   hexout2flash -b 0x600000 -s 0x06bde6 nios_system_
   module.hexout
   ```

   For the Nios Cyclone 1C20 board type:

   ```
   hexout2flash -b 0x600000 -s 0x06c9cb nios_system_
   module.hexout
   ```

   For a Nios Stratix 1S40 board type:

   ```
   hexout2flash -b 0x400000 -s 0x17a1a0 nios_system_
   module.hexout
   ```

   Generating the `nios_system_module.hexout` file creates a new file called **nios_system_module.hexout.flash**.

The hexout2flash utility simply prepends the following GERMS commands to the **.hexout** file and renames it to a **.hexout.flash** file.

```
e600000
e610000
e620000
e630000
e640000
e650000
e660000
e670000
e680000
e690000
```

```
e6a0000 ↵
r600000 ↵
```

The **e** command in GERMS is used to erase flash at the location following `e` (in the first case, `e6000000` means erase the flash sector containing 6000000). The last command r6000000 is a relocation command that affects all addresses in the hexout file. By default the hexout file written starts at 0x000000, but to write this data into the flash starting at address location 0x600000 we use the (**r**) relocate command.

4. To download configuration data for your project to the flash memory device on the Nios development board, type the following command at the **Nios SDK Shell** prompt:

```
nios-run -r nios_system_module.hexout.flash ↵
```

The **nios-run** utility begins to download the configuration data to the flash memory device on the Nios development board, beginning at memory address `0x600000`. This task may require a few minutes to complete. The **nios-run** utility returns to terminal mode when the download is complete.

If downloading is proceeding successfully, the **Nios SDK Shell** displays a string of periods (`.`). If the shell displays an exclamation point (!), an error was encountered during the flash programming. If you receive an exclamation point, go try Figure 4 on page 51 again.

5. To configure the FPGA with the **nios_system_module** data stored in the flash memory, press the Power-On Reset button (SW10) on the Nios development board. The FPGA is configured with the **nios_system_module** data stored in the flash memory device. When the configuration is complete, LED0 through LED7 are lit on the Nios development board and the GERMS monitor displays the text `Nios Hardware Tutorial` or whatever custom message you chose.

Next Steps

Congratulations! You have finished creating, verifying, and using your first Nios system. To learn more about the Nios embedded processor and the SOPC Builder, refer to the following sources:

- For information on using the advanced features of the Nios processor, refer to:
  - *AN 184: Simultaneous Multi-Mastering with the Avalon Bus*
  - *AN 188: Custom Instructions for the Nios Embedded Processor*
  - *AN 189: Simulating Nios Embedded Processor Designs*

- *Nios Custom Instructions Tutorial*

- For additional Nios software development information, refer to:
  - *Nios Software Development Tutorial*
  - *Nios Embedded Processor Software Development Reference Manual*
  - *Nios Embedded Processor 16-Bit Programmer's Reference Manual*
  - *Nios Embedded Processor 32-Bit Programmer's Reference Manual*
  - *AN 284: Implementing Interrupt Service Routines in Nios Systems*

- A variety of reference designs that can be downloaded and used on the Nios development board are located in the **c:\altera\kits\nios\examples** directory.