Nios®

# Nios Tutorial

EXCALIBUR™

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

**Document Version:** 1.1
**Document Date:** 04/02

TU-NIOSTTRL1.1

I.S. EN ISO 9001

# 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 user guide.

*Table 1. Tutorial Revision History*

| Date | Description |
|---|---|
| April 2002 | Updates for Nios version 2.1 |
| January 2002 | Made significant changes to the tutorial, including creating a new 32-bit Nios design. The tutorial supports the Nios embedded processor version 2.0. |
| February 2001 | Updated the tutorial for Nios version 1.1 |

## 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<br>(7:30 a.m. to 5:30 p.m. Pacific Time) |
| Technical support | (800) 800-EPLD (3753)<br>(7:30 a.m. to 5:30 p.m. Pacific Time) | (408) 544-7000 *(1)*<br>(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.

## Documentation Feedback

Altera values your feedback. If you would like to provide feedback on this document—e.g., clarification requests, inaccuracies, or inconsistencies—send e-mail to nios_docs@altera.com.

## Typographic Conventions

The *Nios Tutorial* 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: $\mathbf{f_{MAX}}$, **\QuartusII** 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 Quartus II Help topics are shown in quotation marks. Example: “Configuring a FLEX 10K or FLEX 8000 Device with the BitBlaster™ Download Cable.” |
| `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`.<br><br>Anything that must be typed exactly as it appears is shown in Courier type. For example: `c:\quartusII\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:*

# Contents

# About this Tutorial

## Introduction

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

This tutorial is for Nios novices or users who are new to using embedded systems in PLDs. The tutorial guides you through the steps necessary to create and compile a 32-bit Nios system module design, called **nios_system_module**, and download it into the Nios development board. This simple, single-master Nios system module has a Nios embedded processor and associated system peripherals and interconnections.

After you create the **nios_system_module** design and connect to external pins, you can download it to the Altera® APEX™ device on the Nios development board. The external physical pins on the APEX device are in turn connected to other hardware components 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 three sections:

- "Design Entry" on page 13 teaches you how to create the Nios system module in a Block Design File (.bdf) using the MegaWizard® Plug-In Manager and the SOPC Builder. This section also teaches you how to connect the system module ports to pins in the APEX device.
- "Compilation" on page 45 teaches you how to compile the Nios design using Compiler settings, pin assignments, and EDA tool settings to control compilation processing.
- "Programming" on page 55 teaches you how to use the Quartus II Programmer and the ByteBlasterMV™ cable to download the design to an APEX device on the Nios development board. It also teaches you how to download the design to the flash memory device provided on the board.

The Nios embedded processor version 2.1 includes the SOPC Builder, which supports features such as the simultaneous multi-master Avalon bus and custom instructions for the Nios processor. This tutorial does not cover these features. For more information on these topics, refer to:

- *AN 184: Simultaneous Multi-Mastering with the Avalon Bus*
- *AN 188: Custom Instructions for the Nios Embedded Processor*

## Hardware & Software Requirements

This tutorial requires the following hardware and software:

- A PC running the Windows NT or 2000 operating system
- Nios embedded processor version 2.1
- GNUPro® Nios software development tools version 2.1
- Quartus II Limited Edition software version 1.1 or higher
- A Nios development board, set up as described in the *Nios Embedded Processor Getting Started User Guide*
- The ByteBlaster driver, installed as described in the *Quartus II Installation & Licensing for PCs* manual

☞ When you install the Nios embedded processor, the installation program also installs the LeonardoSpectrum software. The SOPC Builder uses this version of the LeonardoSpectrum software when synthesizing a Nios system module. You can request a free license file for this software from the Nios Development Software Licenses page on the Altera web site at http://www.altera.com. Your license file also contains a license for the Quartus II Limited Edition software.

## Tutorial Files

This tutorial assumes that you create and save your files in a working directory on the **c:** drive on 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 1 in the **\altera\excalibur\sopc_builder_2_5** directory by default:

*Table 1. Directory Structure*

| Directory Name | Description |
|---|---|
| **bin** | Contains the SOPC Builder components used to create a system module. |
| **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 document is **Nios_Tutorial**. |

## More Information

Refer to "Nios Documentation" in the *Nios Embedded Processor Getting Started User Guide* for a listing of the documentation provided with the Excalibur Development Kit, featuring the Nios embedded processor.

*Notes:*

The following tutorial sections guide you through the steps required to create the **nios_system_module** project, and then explain how to create a top-level BDF that contains the Nios system module. You create and instantiate the Nios system module using the SOPC Builder.

☞ The instructions in this section assume that you are familiar with the Quartus II software interface, specifically the toolbars. Refer to Quartus II Help for information that is not included in the tutorial.

## Create a Quartus II Project

### Start the Quartus II Software

In this section, you start the Quartus II software and begin creating your project.

To start the Quartus II software, use one of the following methods:

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

*or*

✓ Type `quartus` ↵ at the command prompt.

### Create a Project

Before you begin, you must create a new Quartus II project. With the **New Project** wizard, you specify the working directory for the project, assign the project name, and designate the name of the top-level design entity. To create a new project, perform the following steps:

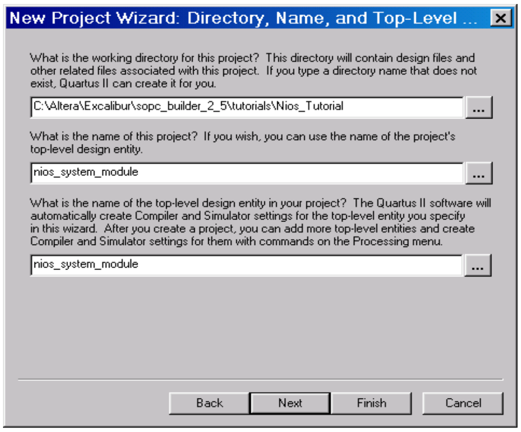
1. Choose **New Project Wizard** (File menu).

2. Click **Next** in the introduction (the introduction will not display if you turned it off previously).

3. Specify the working directory for your project. This walkthrough uses the directory **c:\altera\excalibur\sopc_builder_2_5\tutorials\Nios_Tutorial**.

4. Specify the name of the project and the top-level design entity. This tutorial uses **nios_system_module**. See Figure 1.

*Figure 1. Specifying the Project Name & Directory*



New Project Wizard: Directory, Name, and Top-Level ...

What is the working directory for this project? This directory will contain design files and other related files associated with this project. If you type a directory name that does not exist, Quartus II can create it for you.

C:\Altera\Excalibur\sopc_builder_2_5\tutorials\Nios_Tutorial

What is the name of this project? If you wish, you can use the name of the project's top-level design entity.

nios_system_module

What is the name of the top-level design entity in your project? The Quartus II software will automatically create Compiler and Simulator settings for the top-level entity you specify in this wizard. After you create a project, you can add more top-level entities and create Compiler and Simulator settings for them with commands on the Processing menu.

nios_system_module

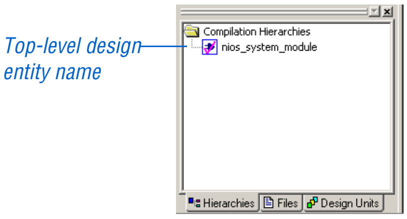
5. Click **Next**.

6. Click **Next**.

7. Click **Finish**.

You have finished creating your new Quartus II project. The top-level design entity name appears in the **Hierarchies** tab of the Project Navigator window. See Figure 2.

*Figure 2. Project Navigator Window*



## Create a Nios System Module

This section describes how to create the top-level Block Design File (**.bdf**) that contains a Nios system module. After creating a design file, you use the SOPC Builder to create the Nios embedded processor, configure system peripherals, and connect these elements to make a Nios system module. Next, you connect the Nios system module ports to the APEX device pins that are connected to hardware components on the Nios development board.

This section includes the following steps:

1. "Create a New .bdf" on page 15
2. "Start the SOPC Builder" on page 17
3. "Add CPU & Peripherals" on page 19
4. "Make Nios System Settings" on page 35
5. "Generate nios32 & Add It to the Design" on page 36
6. "Add Pins & Primitives" on page 39
7. "Name the Pins" on page 42

### Create a New .bdf

In this step you create a new BDF called **nios_system_module.bdf**. This file is the top-level design entity of the **nios_system_module** project.

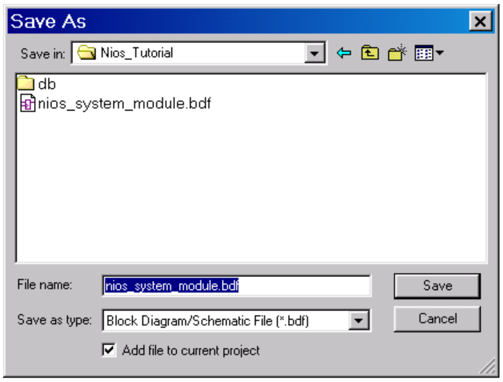
To create a new BDF, follow these steps:

1. Choose **New** (File menu). The **Device Design Files** tab of the **New** dialog box appears automatically.
2. In the **Device Design Files** tab, select **Block Diagram/Schematic File**.
3. Click **OK**. A new **Block Editor** window appears.
4. Choose **Save As** (File menu).
5. The **Save As** dialog box automatically displays the project directory **c:\altera\excalibur\sopc_builder_2_5\tutorials\Nios_Tutorial**. You will save the file into this directory.

6. In the **File name** box, type `nios_system_module` as the name of the BDF, if necessary. Figure 3 shows an existing nios_system_ module .bdf file.

7. Make sure **Add file to current project** is turned on. See Figure 3.

*Figure 3. Creating a New BDF*



File name: nios_system_module.bdf

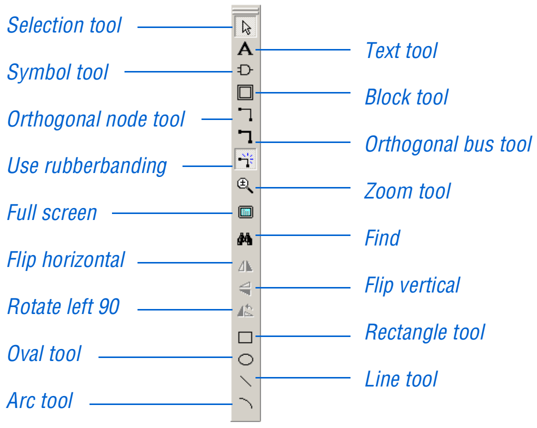
Save as type: Block Diagram/Schematic File (*.bdf)

☑ Add file to current project

8. Click **Save**. The file is saved and added to the project. Leave the **.bdf** open for the remainder of the Design Entry section.

You are now ready to create your Nios design. In the following sections, you will use various tools in the Block Editor toolbar. Figure 4 describes the tools that are available in this toolbar.

*Figure 4. Block Editor Toolbar*



## Start the SOPC Builder

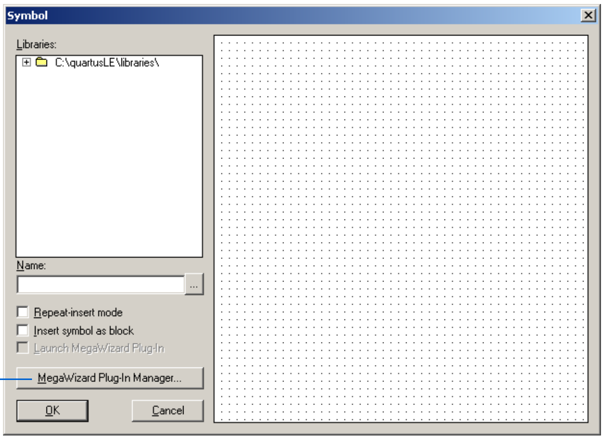
The MegaWizard Plug-In Manager (Tools menu) allows you to create (or modify) design files that contain custom variations of megafunctions. The SOPC Builder is a wizard that guides you through the process of creating a Nios system module or a more general multi-master SOPC module. A complete Nios system module contains a Nios embedded processor and its associated system peripherals.

The SOPC Builder helps you easily specify options for the Nios system module. The wizard prompts you for the values you want to set for parameters and which optional ports and peripherals you want to use. Once the wizard generates the Nios system module, you can instantiate it in the design file.

Follow these steps to start the SOPC Builder:

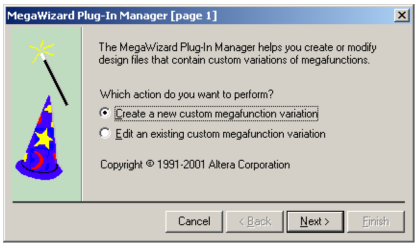
1. Double-click an empty space in the Block Editor window. The **Symbol** dialog box appears. See Figure 5.

*Figure 5. Symbol Dialog Box*



2. Click MegaWizard Plug-In Manager. The first page of the MegaWizard Plug-In Manager is displayed. See Figure 6.

*Figure 6. MegaWizard Plug-In Manager*



The MegaWizard Plug-In Manager helps you create or modify design files that contain custom variations of megafunctions.

Which action do you want to perform?

[✓ Create a new custom megafunction variation]

Edit an existing custom megafunction variation

Copyright © 1991-2001 Altera Corporation

3. Under **Which action do you want to perform?**, select **Create a new custom megafunction variation** and click **Next**. MegaWizard Plug-In Manager page 2a appears.

4. In the **Available Megafunctions** list, select **Altera SOPC Builder 2.5**.

5. Specify the following responses to the remaining wizard options:

   – **Which type of output file do you want to create?** Verilog HDL
   – **What name do you want for the output file?**
   `c:\altera\excalibur\sopc_builder_2_5\tutorials\`
   `Nios_Tutorial\nios32`

6. Click **Next**. The **Altera SOPC Builder - nios32** wizard appears and the **System Contents** tab is displayed.

You are now ready to add the Nios CPU and peripherals to your system.

## Add CPU & Peripherals

The Nios system peripherals allow the Nios embedded processor to connect and communicate with internal logic in the APEX device, or external hardware on the Nios development board. Use the 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
- Boot Monitor ROM
- Communications UART
- Debugging UART
- Timer
- Button PIO
- LCD PIO
- LED PIO
- Seven Segment PIO
- External RAM Bus (Avalon Tri-State Bridge)
- External RAM Interface
- External Flash Interface

Some of the peripherals drive components on the Nios development board. Peripheral names are shown in parentheses. See Figures 7 and 8.

*Figure 7. Nios Development Board*

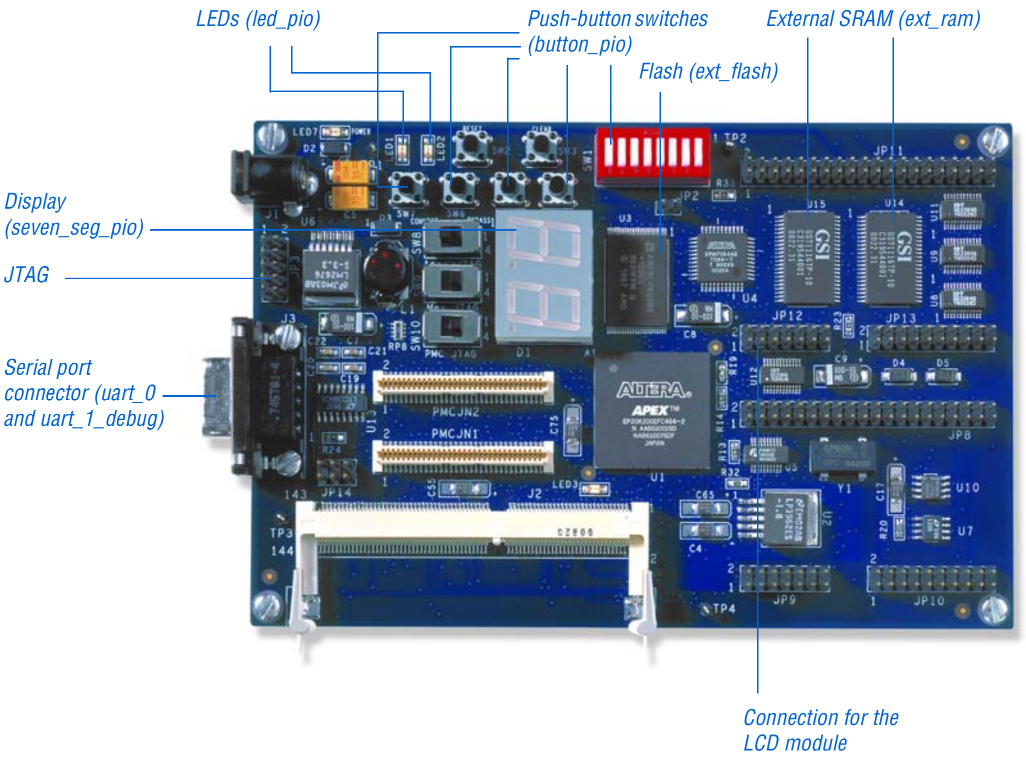

LEDs (led_pio)
Push-button switches (button_pio)
External SRAM (ext_ram)
Flash (ext_flash)
Display (seven_seg_pio)
JTAG
Serial port connector (uart_0 and uart_1_debug)
Connection for the LCD module

*Figure 8. LCD Board*

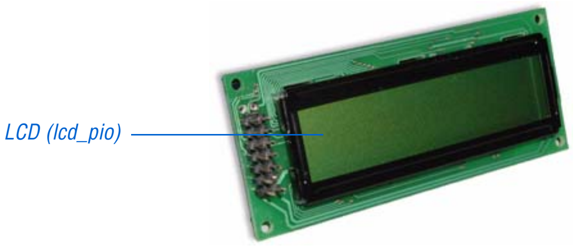
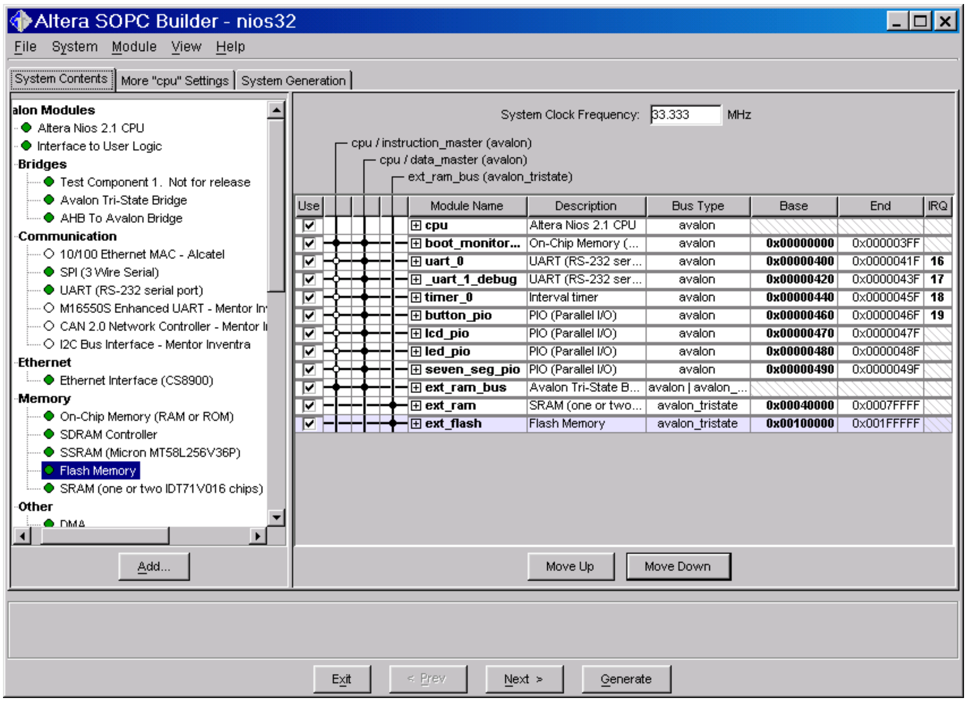


Figure 9 shows the SOPC Builder with all of the modules that you will add in the upcoming sections.

*Figure 9. SOPC Builder with Modules*



alon Modules
- Altera Nios 2.1 CPU
- Interface to User Logic

Bridges
- Test Component 1. Not for release
- Avalon Tri-State Bridge
- AHB To Avalon Bridge

Communication
- 10/100 Ethernet MAC - Alcatel
- SPI (3 Wire Serial)
- UART (RS-232 serial port)
- M16550S Enhanced UART - Mentor In
- CAN 2.0 Network Controller - Mentor I
- I2C Bus Interface - Mentor Inventra

Ethernet
- Ethernet Interface (CS8900)

Memory
- On-Chip Memory (RAM or ROM)
- SDRAM Controller
- SSRAM (Micron MT58L256V36P)
- Flash Memory
- SRAM (one or two IDT71V016 chips)

Other
- DMA

System Clock Frequency: 33.333 MHz

cpu / instruction_master (avalon)
cpu / data_master (avalon)
ext_ram_bus (avalon_tristate)

| Use | Module Name | Description | Bus Type | Base | End | IRQ |
|---|---|---|---|---|---|---|
| ✓ | cpu | Altera Nios 2.1 CPU | avalon | | | |
| ✓ | boot_monitor... | On-Chip Memory (... | avalon | 0x00000000 | 0x000003FF | |
| ✓ | uart_0 | UART (RS-232 ser... | avalon | 0x00000400 | 0x0000041F | 16 |
| ✓ | _uart_1_debug | UART (RS-232 ser... | avalon | 0x00000420 | 0x0000043F | 17 |
| ✓ | timer_0 | Interval timer | avalon | 0x00000440 | 0x0000045F | 18 |
| ✓ | button_pio | PIO (Parallel I/O) | avalon | 0x00000460 | 0x0000046F | 19 |
| ✓ | lcd_pio | PIO (Parallel I/O) | avalon | 0x00000470 | 0x0000047F | |
| ✓ | led_pio | PIO (Parallel I/O) | avalon | 0x00000480 | 0x0000048F | |
| ✓ | seven_seg_pio | PIO (Parallel I/O) | avalon | 0x00000490 | 0x0000049F | |
| ✓ | ext_ram_bus | Avalon Tri-State B... | avalon \| avalon_... | | | |
| ✓ | ext_ram | SRAM (one or two... | avalon_tristate | 0x00040000 | 0x0007FFFF | |
| ✓ | ext_flash | Flash Memory | avalon_tristate | 0x00100000 | 0x001FFFFF | |

## *Nios 32-Bit CPU*

To add the Nios 32-bit CPU, **cpu**, perform the following steps:

1. Select **Altera Nios 2.1 CPU** under **Avalon Modules**.

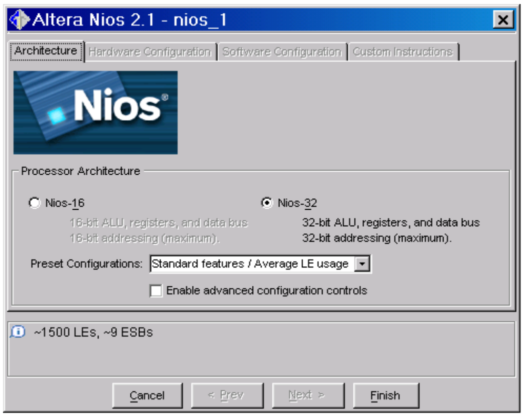
2. Click **Add**. The **Altera Nios - nios_0** wizard displays.

3. Specify the following options in the **Architecture** tab (see Figure 10):

   – **Processor Architecture:** Nios-32
   – **Preset Configurations:** Standard features/Average LE usage

☞ This preset configuration 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.

*Figure 10. Nios CPU Architecture Tab*



Processor Architecture

( ) Nios-16
16-bit ALU, registers, and data bus
16-bit addressing (maximum).

(●) Nios-32
32-bit ALU, registers, and data bus
32-bit addressing (maximum).

Preset Configurations: Standard features / Average LE usage

☐ Enable advanced configuration controls

~1500 LEs, ~9 ESBs

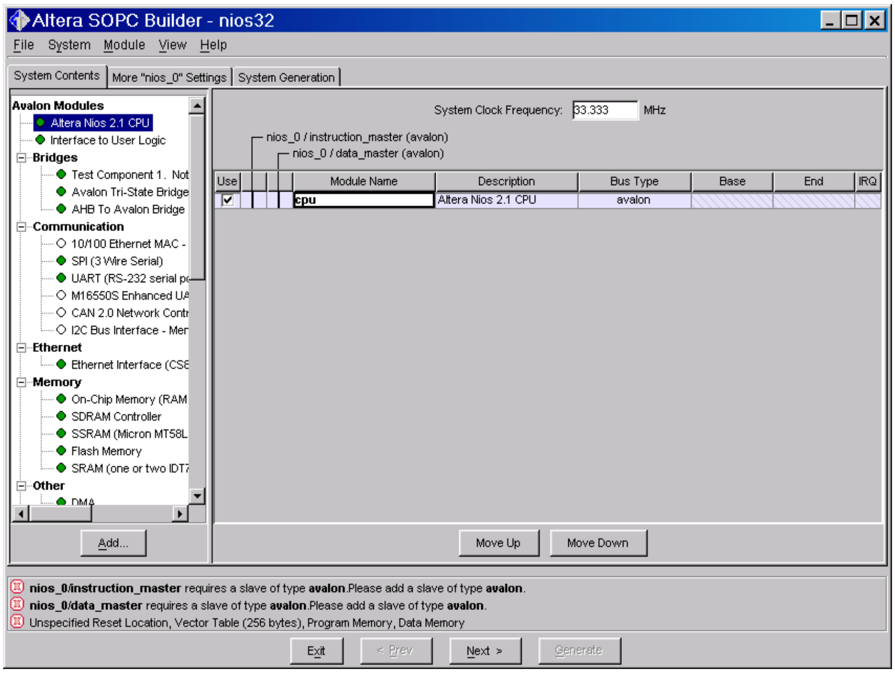
4. Click **Finish**. You are returned to the **Altera SOPC Builder - nios32** window.

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 11.

*Figure 11. SOPC Builder with CPU*



| Use | Module Name | Description | Bus Type | Base | End | IRQ |
|---|---|---|---|---|---|---|
| ✓ | cpu | Altera Nios 2.1 CPU | avalon | | | |

**nios_0/instruction_master** requires a slave of type **avalon**.Please add a slave of type **avalon**.
**nios_0/data_master** requires a slave of type **avalon**.Please add a slave of type **avalon**.
Unspecified Reset Location, Vector Table (256 bytes), Program Memory, Data Memory

## *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.

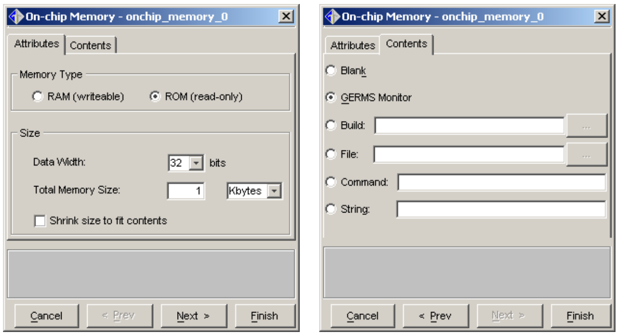
2. Specify the following options in the **Attributes** tab (see Figure 12):

   – **Memory Type:** ROM (read-only)
   – **Data Width:** 32
   – **Total Memory Size:** 1 Kbytes

3. Click the **Contents** tab.

4. Select the **GERMS Monitor** option (see Figure 12).

*Figure 12. Boot Monitor ROM Wizard Settings*



On-chip Memory - onchip_memory_0 (Attributes)

Memory Type: RAM (writeable) [✓ ROM (read-only)]

Size

Data Width: 32 bits

Total Memory Size: 1 Kbytes

☐ Shrink size to fit contents

On-chip Memory - onchip_memory_0 (Contents)

Blank [✓ GERMS Monitor] Build: File: Command: String:

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

6. Right-click **onchip_memory_0** under **Module Name**.

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

8. Rename **onchip_memory_0** as **boot_monitor_rom**. Press return when you are finished typing the new name to save your setting.

## *Communications UART*

This Nios design includes two UART peripherals:

- A UART for communication between the host terminal and the Nios system on the Nios development board.
- A UART for sending and receiving debugging messages between a debugger running on the host and a debug stub running in the Nios system.

To add the communications UART peripheral, **uart_0**, 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 the following options in the **Configuration** tab (see Figure 13):

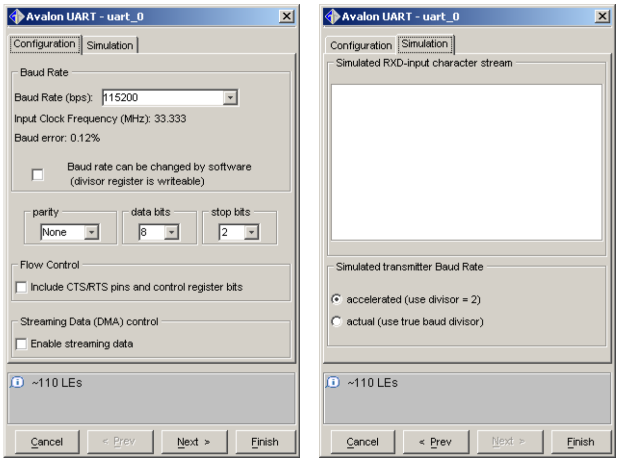
   – **Baud Rate (bps):** 115200
   – **Parity:** None
   – **Data Bits:** 8
   – **Stop Bits:** 2

3. Click the **Simulation** tab.

4. Select the **accelerated (use divisor = 2)** option (see Figure 13).

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 13. Communications UART Wizard Settings***



Baud Rate

Baud Rate (bps): 115200

Input Clock Frequency (MHz): 33.333

Baud error: 0.12%

☐ Baud rate can be changed by software (divisor register is writeable)

parity: None

data bits: 8

stop bits: 2

Flow Control

☐ Include CTS/RTS pins and control register bits

Streaming Data (DMA) control

☐ Enable streaming data

~110 LEs

Simulated RXD-input character stream

Simulated transmitter Baud Rate

● accelerated (use divisor = 2)

○ actual (use true baud divisor)

~110 LEs

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

## *Debugging UART*

To add the UART peripheral, **uart_1_debug**, perform the following steps:

1. Select **UART (RS-232 serial port)** under **Communication** and click **Add**. The **Avalon UART - uart_1** wizard displays.

2. Specify the following options in the **Configuration** tab:

   – **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.

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

6. Right-click **uart_1** under **Module Name**.

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

8. Rename **uart_1** as **uart_1_debug**. Press return when you are finished typing the new name to save your setting.

### *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.

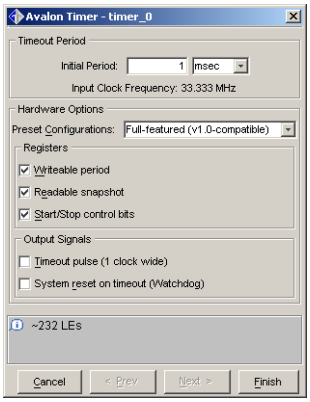
To add the timer peripheral, **timer_0**, perform the following steps:

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

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

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

*Figure 14. Timer Wizard Settings*


Avalon Timer - timer_0

Timeout Period

Initial Period: 1 msec

Input Clock Frequency: 33.333 MHz

Hardware Options

Preset Configurations: Full-featured (v1.0-compatible)

Registers

☑ Writeable period

☑ Readable snapshot

☑ Start/Stop control bits

Output Signals

☐ Timeout pulse (1 clock wide)

☐ System reset on timeout (Watchdog)

~232 LEs

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

## *Button PIO*

To add the button PIO peripheral, **button_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 (see Figure 15):

   – **Width:** 12 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.

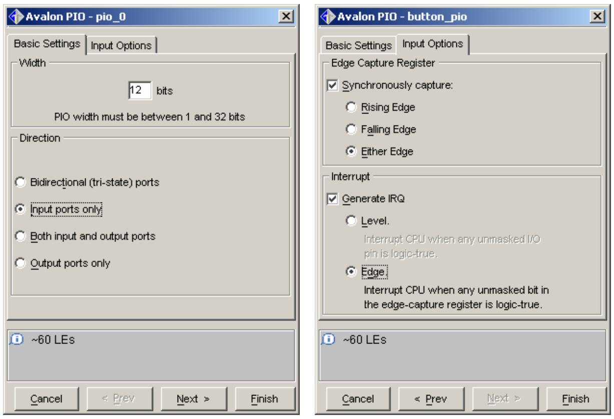
4. Turn on **Synchronously capture** under **Edge Capture Register**.

5. Select **Either Edge**.

6. Turn on **Generate IRQ** under **Interrupt**.

7. Select **Edge**. See Figure 15.

*Figure 15. Button PIO Wizard Settings*



Width

12 bits

PIO width must be between 1 and 32 bits

Direction

- Bidirectional (tri-state) ports
- [✓ Input ports only]
- Both input and output ports
- Output ports only

~60 LEs

Edge Capture Register

[✓] Synchronously capture:

- Rising Edge
- Falling Edge
- [✓ Either Edge]

Interrupt

[✓] Generate IRQ

- Level.
  Interrupt CPU when any unmasked I/O pin is logic-true.
- [✓ Edge.]
  Interrupt CPU when any unmasked bit in the edge-capture register is logic-true.

~60 LEs

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

9. Right-click **pio_0** under **Module Name**.

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

11. Rename **pio_0** as **button_pio**. Press return when you are finished typing the new name to save your setting.

## *LCD PIO*

To add the LCD PIO peripheral, **lcd_pio**, perform the following steps:

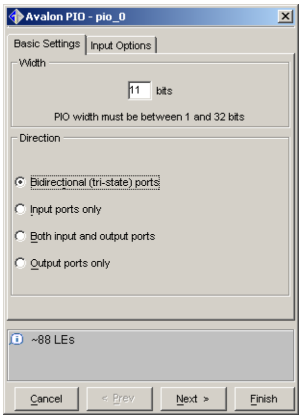
1. Select **PIO (Parallel I/O)** under **Other** and click **Add**. The **Avalon PIO - pio_0** wizard displays.

2. Specify the following options (see Figure 16):

   – **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 16. LCD PIO Wizard Settings*


Avalon PIO - pio_0

Basic Settings | Input Options

Width

11 bits

PIO width must be between 1 and 32 bits

Direction

(● Bidirectional (tri-state) ports) ○ Input ports only ○ Both input and output ports ○ Output ports only

~88 LEs

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. Right-click **pio_0** under **Module Name**.

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

7. Rename **pio_0** as **lcd_pio**. Press return when you are finished typing the new name to save your setting.

## *LED PIO*

To add the LED PIO peripheral, **led_pio**, perform the following steps:

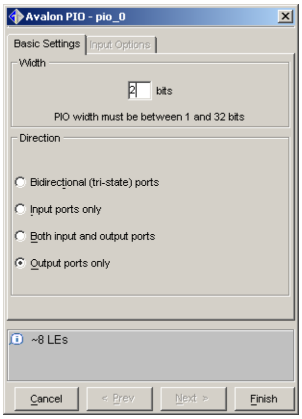
1. Select **PIO (Parallel I/O)** under **Other** and click **Add**. The **Avalon PIO - pio_0** wizard displays.

2. Specify the following options (see Figure 17):

   – **Width:** 2 bits
   – **Direction:** Output ports only

☞ The standard 32-bit reference design included with the Nios processor version 2.1 has the LED PIO set to use bidirectional pins and the LEDs power up turned off (not illuminated). 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 you power up the board, LED1 and LED2 are illuminated, indicating that the design running in the board is the tutorial design and not the factory-default design.

*Figure 17. LED PIO Wizard*



Avalon PIO - pio_0

Width: 2 bits

PIO width must be between 1 and 32 bits

Direction:
( ) Bidirectional (tri-state) ports
( ) Input ports only
( ) Both input and output ports
[✓ Output ports only]

~8 LEs

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

4. Right-click **pio_0** under **Module Name**.

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

6. Rename **pio_0** as **led_pio**. Press return when you are finished typing the new name to save your setting.

### 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. Right-click **pio_0** under **Module Name**.

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

6. Rename **pio_0** as **seven_seg_pio**. Press return when you are finished typing the new name to save your setting.

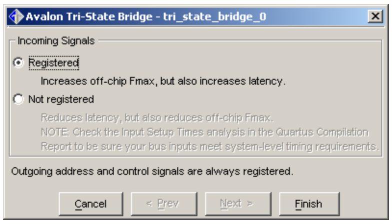
### External RAM Bus (Avalon Tri-State Bridge)

For the Nios system to communicate with memory external to the APEX device on the Nios development board, you must add a bridge between the Avalon bus and the bus or buses to which the external memory is connected.

To add the Avalon tri-state bridge, **ext_ram_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 18. The **Registered** option is turned on by default.

**Figure 18. Avalon Tri-State Bridge Wizard**


Avalon Tri-State Bridge - tri_state_bridge_0

Incoming Signals

(●) Registered
Increases off-chip Fmax, but also increases latency.
( ) Not registered
Reduces latency, but also reduces off-chip Fmax.
NOTE: Check the Input Setup Times analysis in the Quartus Compilation Report to be sure your bus inputs meet system-level timing requirements.

Outgoing address and control signals are always registered.

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

3. Right-click **tri_state_bridge_0** under **Module Name**.

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

5. Rename **tri_state_bridge_0** as **ext_ram_bus**. Press return when you are finished typing the new name to save your setting.

### *External RAM Interface*

To add the external RAM peripheral, **ext_ram**, perform the following steps:

1. Select **SRAM (one or two IDT71V016 chips)** under **Memory** and click **Add**. The **SRAM (one or two IDT71V016 chips) - nios_dev_board_sram32...** wizard displays.

2. In the **Attributes** tab, choose **32 bits wide / 256KBytes (Nios 32 default)** from the **Choose Configuration** drop-down list box (see Figure 19).

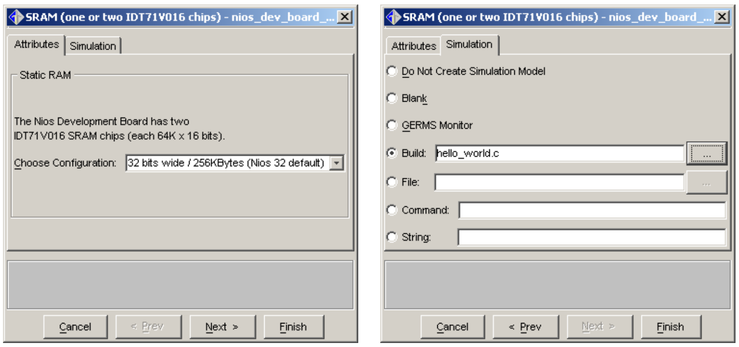
3. Click the **Simulation** tab.

4. Select **Build**.

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

6. Select the **hello_world.c** file, and click **Open** (see Figure 19).

***Figure 19. External RAM Wizard Settings***



Static RAM

The Nios Development Board has two IDT71V016 SRAM chips (each 64K x 16 bits).

Choose Configuration: 32 bits wide / 256KBytes (Nios 32 default)

- ( ) Do Not Create Simulation Model
- ( ) Blank
- ( ) GERMS Monitor
- (●) Build: hello_world.c
- ( ) File:
- ( ) Command:
- ( ) String:

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

8. Right-click **nios_dev_board_sram32_0** under **Module Name**.

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

10. Rename **nios_dev_board_sram32_0** as **ext_ram**. Press return when you are finished typing the new name to save your setting.

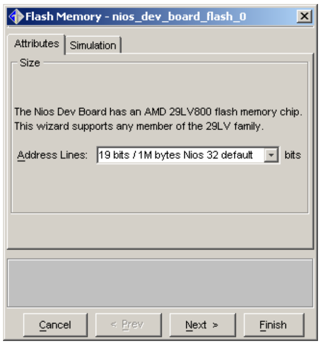
### *External Flash Interface*

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

1. Select **Flash Memory** under **Memory** and click **Add**. The **Flash Memory - nios_dev_board_flash_0** wizard displays.

2. Choose **19 bits / 1 M bytes Nios 32 default** from the **Address Lines** drop-down list box. See Figure 20.

***Figure 20. External Flash Wizard***


Flash Memory - nios_dev_board_flash_0

Attributes | Simulation

Size

The Nios Dev Board has an AMD 29LV800 flash memory chip.
This wizard supports any member of the 29LV family.

Address Lines: 19 bits / 1M bytes Nios 32 default bits

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

4. Right-click **nios_dev_board_flash_0** under **Module Name**.

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

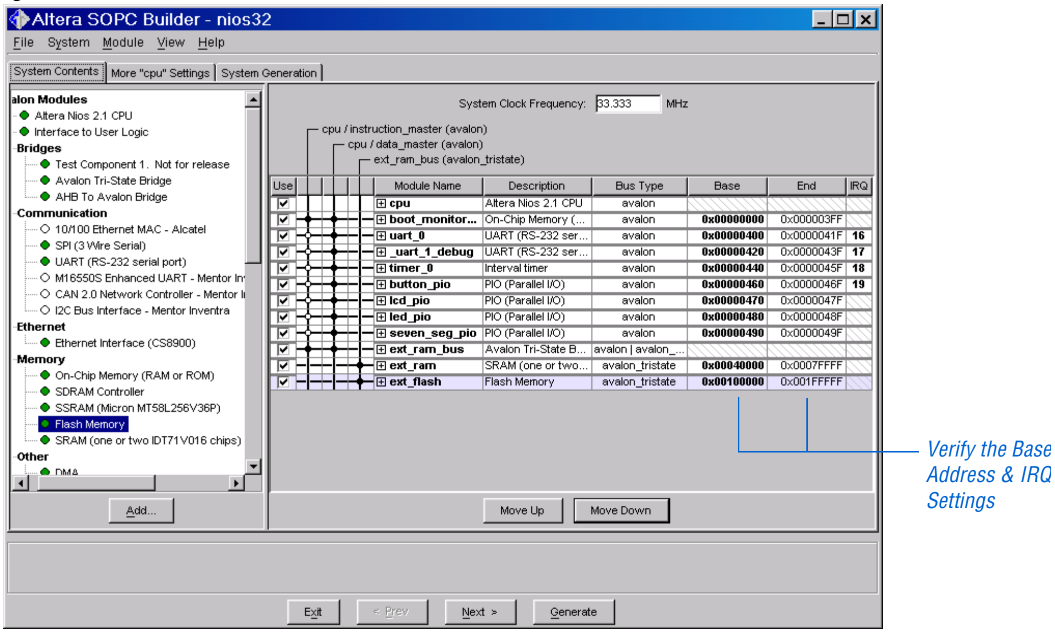
6. Rename **nios_dev_board_flash_0** as **ext_flash**. Press return when you are finished typing the new name to save your setting.

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

## Specify Base Addresses & IRQs

The SOPC Builder assigns default base address and IRQ values for the components in your Nios system module. However, you can modify these defaults. For this tutorial, make sure that the SOPC Builder uses the base addresses and IRQ values shown in Figure 21.

*Figure 21. Base Address & IRQ Values*



alon Modules
- Altera Nios 2.1 CPU
- Interface to User Logic

Bridges
- Test Component 1. Not for release
- Avalon Tri-State Bridge
- AHB To Avalon Bridge

Communication
- 10/100 Ethernet MAC - Alcatel
- SPI (3 Wire Serial)
- UART (RS-232 serial port)
- M16550S Enhanced UART - Mentor In
- CAN 2.0 Network Controller - Mentor I
- I2C Bus Interface - Mentor Inventra

Ethernet
- Ethernet Interface (CS8900)

Memory
- On-Chip Memory (RAM or ROM)
- SDRAM Controller
- SSRAM (Micron MT58L256V36P)
- Flash Memory
- SRAM (one or two IDT71V016 chips)

Other
- DMA

System Clock Frequency: 33.333 MHz

cpu / instruction_master (avalon)
cpu / data_master (avalon)
ext_ram_bus (avalon_tristate)

| Use | Module Name | Description | Bus Type | Base | End | IRQ |
|---|---|---|---|---|---|---|
| ✓ | cpu | Altera Nios 2.1 CPU | avalon | | | |
| ✓ | boot_monitor... | On-Chip Memory (... | avalon | 0x00000000 | 0x000003FF | |
| ✓ | uart_0 | UART (RS-232 ser... | avalon | 0x00000400 | 0x0000041F | 16 |
| ✓ | _uart_1_debug | UART (RS-232 ser... | avalon | 0x00000420 | 0x0000043F | 17 |
| ✓ | timer_0 | Interval timer | avalon | 0x00000440 | 0x0000045F | 18 |
| ✓ | button_pio | PIO (Parallel I/O) | avalon | 0x00000460 | 0x0000046F | 19 |
| ✓ | lcd_pio | PIO (Parallel I/O) | avalon | 0x00000470 | 0x0000047F | |
| ✓ | led_pio | PIO (Parallel I/O) | avalon | 0x00000480 | 0x0000048F | |
| ✓ | seven_seg_pio | PIO (Parallel I/O) | avalon | 0x00000490 | 0x0000049F | |
| ✓ | ext_ram_bus | Avalon Tri-State B... | avalon \| avalon_... | | | |
| ✓ | ext_ram | SRAM (one or two... | avalon_tristate | 0x00040000 | 0x0007FFFF | |
| ✓ | ext_flash | Flash Memory | avalon_tristate | 0x00100000 | 0x001FFFFF | |

☞ Roll your mouse over any of the fields for an enhanced view.

If the SOPC Builder does not use the values shown in Figure 21, you can change the values by clicking the base address or IRQ fields and typing the correct value.

## Make Nios System Settings

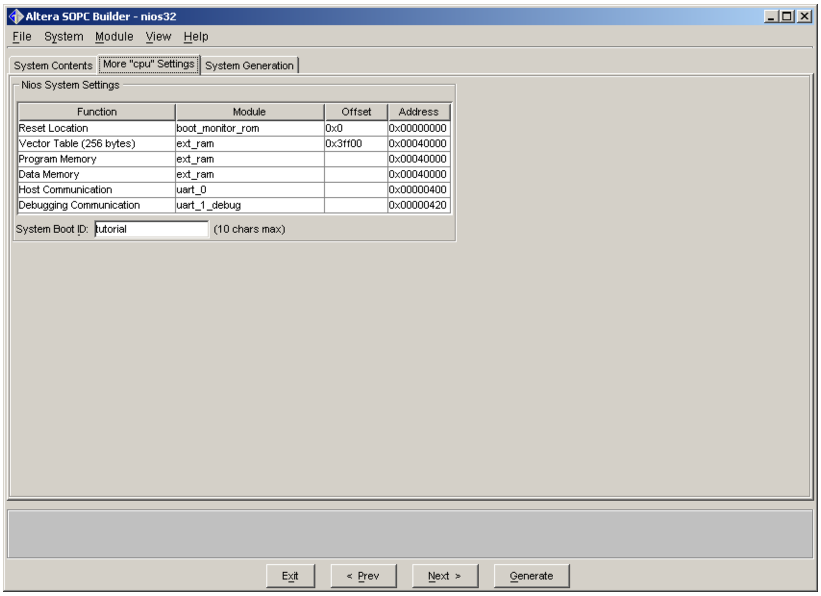
After you add the components to the system module, you must make system settings.

1. Click the **More "cpu" Settings** tab. The text in parentheses 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 22 on page 36):

   – Reset Location
     - **Module:** boot_monitor_rom
     - **Offset:** 0x0
   – Vector Table (256 bytes)
     - **Module**: ext_ram
     - **Offset:** 0x3ff00
   – Program Memory
     - **Module:** ext_ram
   – Data Memory
     - **Module:** ext_ram
   – Host Communication
     - **Module:** uart_0
   – Debugging Communication
     - **Module:** uart_1_debug
   – **System Boot ID:** tutorial

*Figure 22. SOPC Builder More 'cpu' Settings Tab*



| Function | Module | Offset | Address |
|---|---|---|---|
| Reset Location | boot_monitor_rom | 0x0 | 0x00000000 |
| Vector Table (256 bytes) | ext_ram | 0x3ff00 | 0x00040000 |
| Program Memory | ext_ram | | 0x00040000 |
| Data Memory | ext_ram | | 0x00040000 |
| Host Communication | uart_0 | | 0x00000400 |
| Debugging Communication | uart_1_debug | | 0x00000420 |

System Boot ID: tutorial (10 chars max)

## Generate nios32 & Add It to the Design

Before you compile the Nios design with the Quartus II software, you must synthesize the design logic (i.e., generate the design) using the LeonardoSpectrum software. The LeonardoSpectrum software is installed with the Nios embedded processor and runs automatically within the SOPC Builder to synthesize the design and generate the **nios32.edf** file.

Before generating the design, perform the following steps:

1. Make the following settings under **Options** in the **System Generation** tab (See Figure 23):

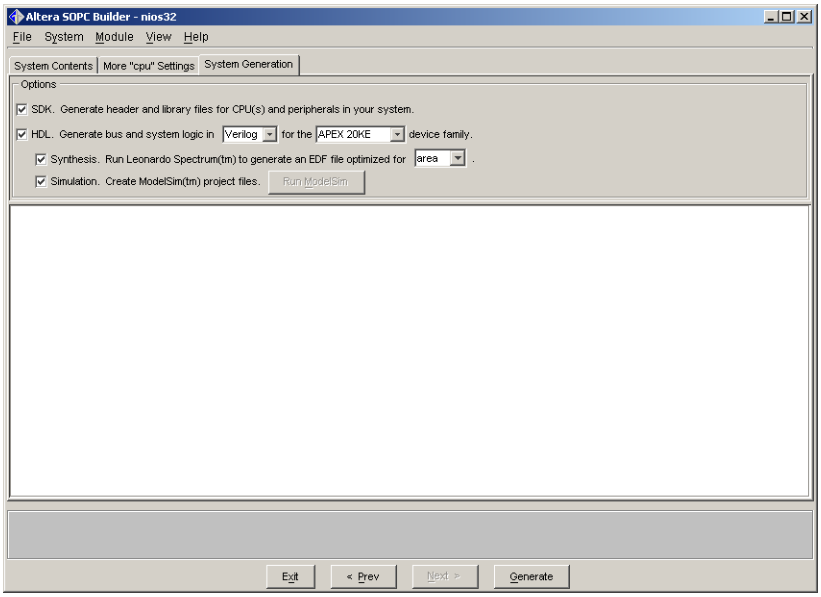
   – **SDK:** Turn on

   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 and choose Verilog and APEX 20KE
   – **Synthesis:** Turn on and choose to optimize for area
   – **Simulation:** Turn on

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

**Figure 23. SOPC Builder System Generation Tab**



Altera SOPC Builder - nios32

☑ SDK. Generate header and library files for CPU(s) and peripherals in your system.

☑ HDL. Generate bus and system logic in Verilog for the APEX 20KE device family.

☑ Synthesis. Run Leonardo Spectrum(tm) to generate an EDF file optimized for area .

☑ Simulation. Create ModelSim(tm) project files.

To synthesize the design, perform the following steps.

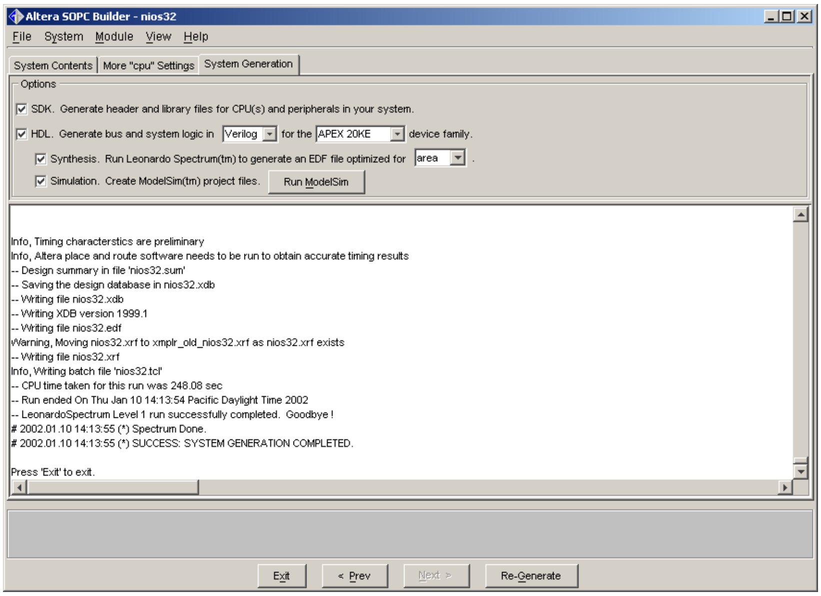
1. Click the **System Generation** tab if you have not already.
2. Click **Generate** in the SOPC Builder. The SOPC Builder performs a variety of actions during design generation, depending on which options you have specified. For the design created using this tutorial, which has all available SOPC Builder options turned on, the SOPC Builder performs the following actions:

   – Generates the SDK, C, and Assembly language header and source files
   – Compiles the library for your system
   – Compiles the GERMS monitor used in the boot monitor ROM
   – Generates the HDL source files
   – Creates the simulation project and source files
   – Synthesizes the HDL files using the LeonardoSpectrum software and produces an EDIF netlist file

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

3. When generation is complete (see Figure 24), the `SYSTEM GENERATION COMPLETED` message displays. Click **Exit** to exit the SOPC Builder, which returns you to the **Symbol** dialog box.

*Figure 24. System Generation Completes*



SDK. Generate header and library files for CPU(s) and peripherals in your system. [✓]
HDL. Generate bus and system logic in Verilog for the APEX 20KE device family. [✓]
Synthesis. Run Leonardo Spectrum(tm) to generate an EDF file optimized for area . [✓]
Simulation. Create ModelSim(tm) project files. [✓]

```
Info, Timing characterstics are preliminary
Info, Altera place and route software needs to be run to obtain accurate timing results
-- Design summary in file 'nios32.sum'
-- Saving the design database in nios32.xdb
-- Writing file nios32.xdb
-- Writing XDB version 1999.1
-- Writing file nios32.edf
Warning, Moving nios32.xrf to xmplr_old_nios32.xrf as nios32.xrf exists
-- Writing file nios32.xrf
Info, Writing batch file 'nios32.tcl'
-- CPU time taken for this run was 248.08 sec
-- Run ended On Thu Jan 10 14:13:54 Pacific Daylight Time 2002
-- LeonardoSpectrum Level 1 run successfully completed.  Goodbye !
# 2002.01.10 14:13:55 (*) Spectrum Done.
# 2002.01.10 14:13:55 (*) SUCCESS: SYSTEM GENERATION COMPLETED.

Press 'Exit' to exit.
```

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

## Add the Symbol to the BDF

During generation, the 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. In the **Symbol** dialog box, click **Browse** (**...**). The **Open** dialog box appears.

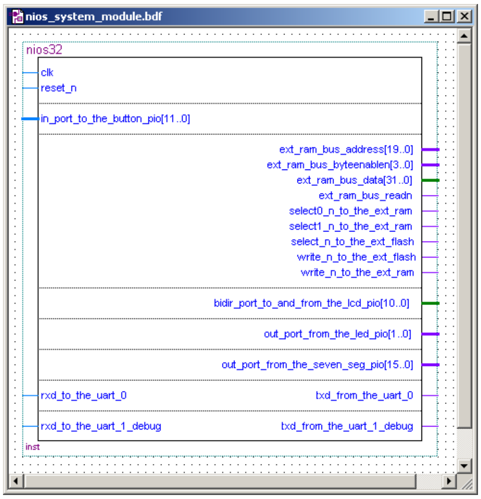
2. In the **c:\altera\excalibur\sopc_builder_2_5\tutorials\Nios_Tutorial** directory, select the newly created symbol, **nios32.bsf**.

3. Click **Open**. A preview of the `nios32` symbol appears in the **Symbol** dialog box.

4. Click **OK** to instantiate the `nios32` symbol in the BDF. An outline of the `nios32` symbol is attached to the pointer.

5. To place the symbol, click an empty space in the **Block Editor** window. The `nios32` symbol is instantiated in the BDF as shown in Figure 25.

*Figure 25. Adding the nios32 Symbol*


nios_system_module.bdf
nios32
clk
reset_n
in_port_to_the_button_pio[11..0]
ext_ram_bus_address[19..0]
ext_ram_bus_byteenablen[3..0]
ext_ram_bus_data[31..0]
ext_ram_bus_readn
select0_n_to_the_ext_ram
select1_n_to_the_ext_ram
select_n_to_the_ext_flash
write_n_to_the_ext_flash
write_n_to_the_ext_ram
bidir_port_to_and_from_the_lcd_pio[10..0]
out_port_from_the_led_pio[1..0]
out_port_from_the_seven_seg_pio[15..0]
rxd_to_the_uart_0
txd_from_the_uart_0
rxd_to_the_uart_1_debug
txd_from_the_uart_1_debug
inst

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

## Add Pins & Primitives

To enter input, output, and bidirectional pins and primitives, perform the following steps:

1. Turn on rubberbanding if it is not already turned on by clicking the Use rubberbanding icon on the Block Editor toolbar (refer to Figure 4 on page 17 for a description of the tools in the Block Editor toolbar).

2. Click the **Symbol Tool** button on the Block Editor toolbar. The **Symbol** dialog box appears. Opening the **Symbol** dialog box using the toolbar button enables the **Repeat-insert mode** option. This option makes it easy for you to add multiple instances of a symbol.

☞ Refer to Quartus II Help if you are unfamiliar with the Quartus II toolbar buttons.

3. In the **Libraries** list of the **Symbol** dialog box, click the + icon to expand the **c:\quartus\libraries** folder. Expand the **primitives** folder and then expand the **pin** folder.

4. In the **pin** folder, select the **input** primitive.

5. Click **OK**.

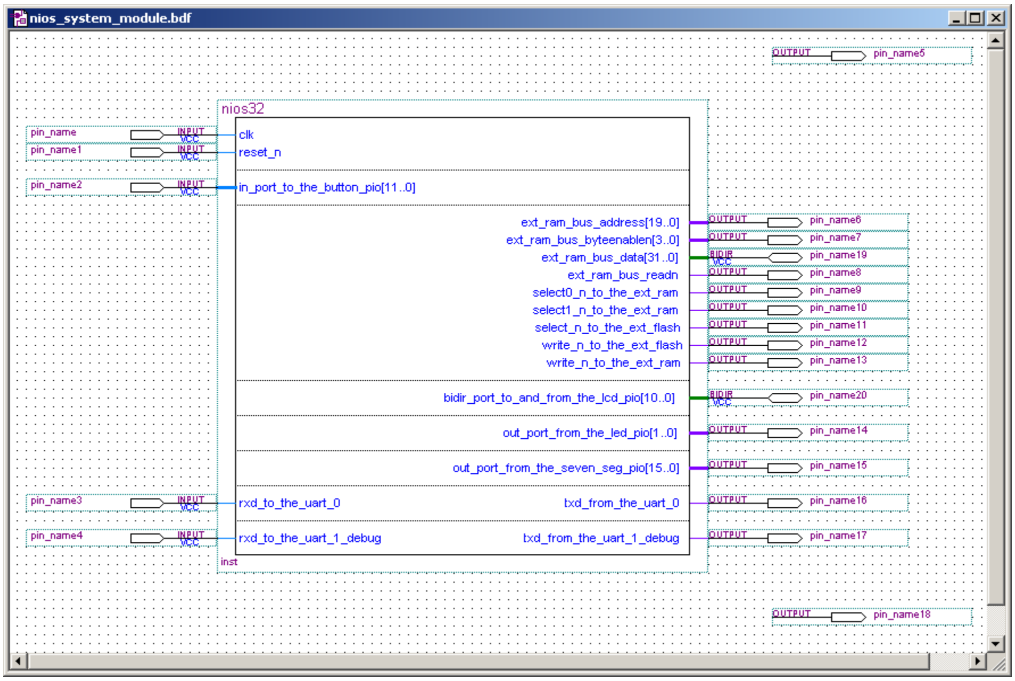
6. Click an empty space five times to insert a total of 5 `INPUT` pin symbols on the left-hand side of the `nios32` symbol. Position each symbol so that the right side of it touches the `nios32` symbol and the horizontal line in each `INPUT` pin symbol meets a horizontal line on the `nios32` symbol. See Figure 26.

   By positioning the pins using this method (with rubberbanding turned on), you are connecting them to the `nios32` symbol. If you select one of the pin symbols and move it away from the `nios32` symbol, a connection line appears between them.

   ☞ Symbols are automatically named as `pin_name`<*number*> in sequence. Press the Esc key when you are done adding the symbols.

7. Repeat steps 1 through 5 to insert and position a total of 14 `OUTPUT` pins and 2 `BIDIR` pins in the file in the locations shown in Figure 26.

*Figure 26. Adding Input, Output & Bidirectional Pins*



8. Double-click a blank space in the BDF. The **Symbol** dialog box appears.

9. In the **Libraries** list of the **Symbol** dialog box, click the + icon to expand the **c:\quartus\libraries** folder. Expand the **primitives** folder and then expand the **other** folder.

10. In the **other** folder, select the **gnd** primitive.

11. Click an empty space in the BDF next to the `OUTPUT` pin at the bottom of the BDF to insert the `GND` symbol.

12. Double-click a blank space in the BDF. The **Symbol** dialog box appears.

13. In the **Libraries** list of the **Symbol** dialog box, click the **+** icon to expand the **c:\quartus\libraries** folder. Expand the **primitives** folder and then expand the **logic** folder.

14. In the **logic** folder, select the **not** primitive.

15. Click **OK**.

16. Click an empty space in the BDF to insert the `NOT` symbol.

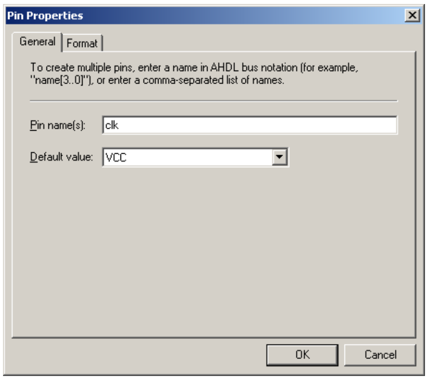
17. Choose **Save** (File menu).

## Name the Pins

You can now name the pins. To name the clock pin, perform the following steps:

1. With the Selection Tool, double-click the first input pin symbol that you entered. The **General** tab of the **Pin Properties** dialog box appears. See Figure 27.

*Figure 27. Pin Properties Dialog Box*



To create multiple pins, enter a name in AHDL bus notation (for example, "name[3..0]"), or enter a comma-separated list of names.

Pin name(s): clk

Default value: VCC

2. In the **Pin name(s)** box, type `clk` to replace the default name of the pin, i.e., to replace `pin_name`.

3. Click **OK**.

4. Repeat steps 1 through 3 to rename each of the pins with the names listed in Table 1.

   For the design to work properly, you must name the pins as shown in Table 1. Later in this tutorial you will use a Tcl script to make pin assignments that reflect these names.

*Table 1. Input, Output & Bidirectional Pins Names*

| Pin Name | Type | Connected to nios32 Signal |
|---|---|---|
| `clk` | Input | `clk` |
| `reset_n` | Input | `reset_n` |
| `button_pio[11..0]` | Input | `in_port_to_the_button_pio[11..0]` |
| `rxd` | Input | `rxd_to_the_uart_0` |
| `rxd_debug` | Input | `rxd_to_the_uart_1_debug` |
| `ext_addr[19..0]` | Output | `ext_ram_bus_address[19..0]` |
| `ext_be_n[3..0]` | Output | `ext_ram_bus_byteenable[3..0]` |
| `ext_data[31..0]` | Bidirectional | `ext_ram_bus_data[31..0]` |
| `SRAM_Lo_oe_n` | Output | `ext_ram_bus_readn` |
| `SRAM_Hi_cs_n` | Output | `select0_n_to_the_ext_ram` |
| `SRAM_Lo_cs_n` | Output | `select1_n_to_the_ext_ram` |
| `FLASH_ce_n` | Output | `select_n_to_the_ext_flash` |
| `FLASH_we_n` | Output | `write_n_to_the_ext_flash` |
| `SRAM_Lo_we_n` | Output | `write_n_to_the_ext_ram` |
| `lcd_pio[10..0]` | Bidirectional | `bidir_port_to_and_from_the_lcd_pio[10..0]` |
| `led_pio[1..0]` | Output | `out_port_from_the_led_pio[1..0]` |
| `seven_seg_pio[15..0]` | Output | `out_port_from_the_seven_seg_pio[15..0]` |
| `txd` | Output | `txd_from_the_uart_0` |
| `txd_debug` | Output | `txd_from_the_uart_1_debug` |
| `JP12_sel_n` | Output | None. This pin selects the 5-V proto-card pins so that JP12 uses the LCD module. |
| `FLASH_a16` | Output | None. This pin allows the flash and SRAM to be used for both 32- and 16-bit designs on the same Nios development board. |

## Make the Final Connections

To connect the remaining symbols in the BDF, perform the following steps:

1. Select the **Orthogonal Node** tool on the **Block Editor** toolbar.

2. To create a node that can be connected by name, draw a node line from the `FLASH_a16` pinstub to an empty space to the left.

3. Click the **Selection Tool** button on the toolbar.

4. Double-click the node line you just drew. The **General** tab of the **Node Properties** dialog box appears.

5. In the **Name** box, type `ext_addr[17]` as the name of the node.

6. Click **OK**. The name appears above the node line. Adding this name creates a logical connection between the `FLASH_a16` pin and the `ext_addr[17]` pin that is connected to the SRAM device on the Nios development board.

7. Position the pointer at the bounding box of the `GND` symbol, aligned with the vertical line in the symbol. The pointer turns into a crosshair with which you can draw an orthogonal node.

8. Draw a node line from the `GND` symbol to the left side of `JP12_sel_n`.

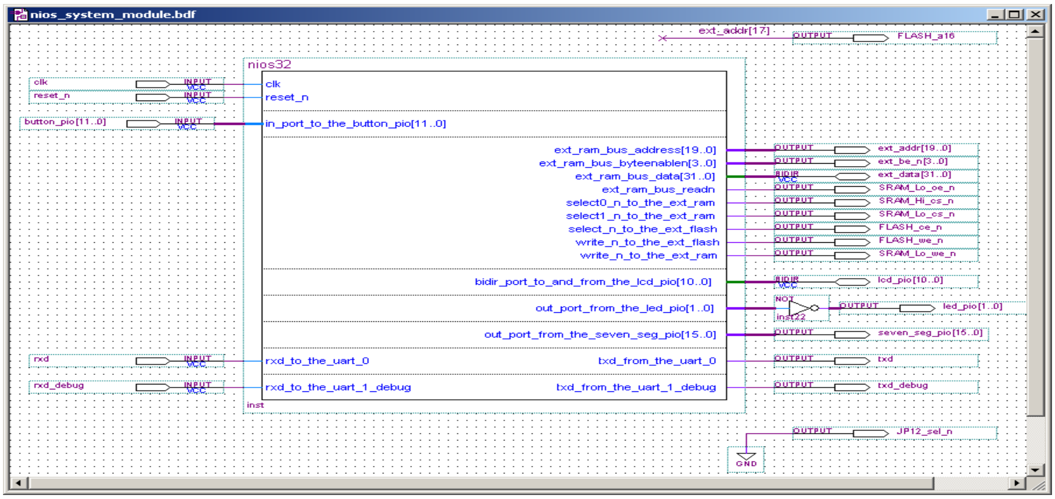
9. Draw a node line from `out_port_from_the_led_pio[1..0]` to the `NOT` primitive.

10. Draw a node line from the `NOT` primitive to the `led_pio[1..0]` signal.

11. Move the input, output, and bidirectional pins away from the `nios32` symbol so that the node lines are visible.

12. Choose **Save** (File menu). The BDF is complete. See Figure 28.

*Figure 28. Completed BDF*



# Compilation

The Quartus II Compiler consists of a series of modules that check a design for errors, synthesize the logic, fit the design into an Altera device, and generate output files for simulation, timing analysis, and device programming.

The following tutorial sections guide you through the steps necessary to create Compiler settings, assign signals to device pins, specify EDA tool settings, and compile the design. The compilation tutorial sections include:

## Create Compiler Settings

You can create Compiler settings to control the compilation process. The Compiler settings specify the compilation focus, the type of compilation to perform, the device to target, and other options. This section includes the following steps:

1. "View the Compiler General Settings" on page 46
2. "Specify the Device Family & Device" on page 47

☞ The procedures below explain how to view and edit Compiler settings using menu commands and dialog boxes. However, you can also easily specify Compiler settings by following the steps in the **Compiler Settings Wizard** (Processing menu).

## View the Compiler General Settings

The **General** tab of the **Compiler Settings** dialog box (Processing menu) allows you to select an existing group of Compiler settings for use during compilation, define and save a new group of Compiler settings, specify the compilation focus, or delete existing settings.

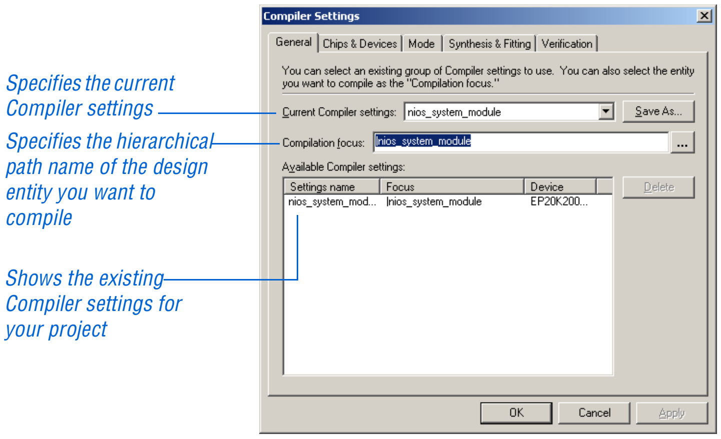
To view the default Compiler general settings created for the current project, follow these steps:

1. Make sure that you are in Compile mode by selecting **Compile Mode** (Processing menu) *or* click the Compile toolbar button.

2. Choose **Compiler Settings** (Processing menu). The **General** tab of the **Compiler Settings** dialog box appears automatically.

At this point in the tutorial, the **General** tab displays only the default Compiler general settings created by the Quartus II software when the project was initially created. These default settings are given the name of the top-level design entity in the project, **nios_system_module**. See Figure 1.

***Figure 1. Default Compiler Settings***


Specifies the current Compiler settings
Specifies the hierarchical path name of the design entity you want to compile
Shows the existing Compiler settings for your project
Compiler Settings
General
Chips & Devices
Mode
Synthesis & Fitting
Verification
Save As...
...
Delete
OK
Cancel
Apply

You can select an existing group of Compiler settings to use. You can also select the entity you want to compile as the "Compilation focus."

Current Compiler settings: nios_system_module

Compilation focus: |nios_system_module

Available Compiler settings:

| Settings name | Focus | Device |
|---|---|---|
| nios_system_mod... | \|nios_system_module | EP20K200... |

## Specify the Device Family & Device

The Nios development board includes an APEX EP20K200EFC484-2X device. In this section, you will target this device in the Compiler settings.

The **Chips & Devices** tab of the **Compiler Settings** dialog box allows you to select the family and device you want to target for compilation.

To select the device family and device, follow these steps:

1. In the **Compiler Settings** dialog box, click the **Chips & Devices** tab.

2. In the **Family** list, select **APEX 20KE**.

3. If you receive a message that asks if you want the Quartus II software to choose a device automatically, click **No**.

4. Under **Target device**, select **Specific device selected in 'Available devices' list**.

5. Under **Show in 'Available devices' list**, select the following options:

   a. In the **Package** list, select **FBGA**.

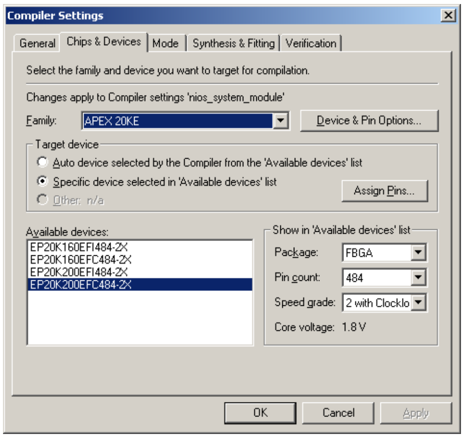
   b. In the **Pin count** list, select **484**.

   c. In the **Speed grade** list, choose **2 with ClockLock**.

   ☞ As you change these options from **Any**, the number of available device listed decreases, making it easier for you to find your target device.

6. In the **Available devices** list, select **EP20K200EFC484-2X**. See Figure 2.

*Figure 2. Chips & Devices Settings*



Select the family and device you want to target for compilation.

Changes apply to Compiler settings 'nios_system_module'

Family: APEX 20KE

Device & Pin Options...

Target device

- Auto device selected by the Compiler from the 'Available devices' list
- [✓ Specific device selected in 'Available devices' list]
- Other: n/a

Assign Pins...

Available devices:

EP20K160EFI484-2X
EP20K160EFC484-2X
EP20K200EFI484-2X
[✓ EP20K200EFC484-2X]

Show in 'Available devices' list

Package: FBGA
Pin count: 484
Speed grade: 2 with Clocklo
Core voltage: 1.8 V

7. To accept the defaults for the remaining Compiler settings, click **OK**.

## Assign Signals to Device Pins

During compilation, the Compiler assigns the logic of your design to physical device resources. You can also make pin assignments to direct the Compiler to assign signals in your design to specific pins in the target device. Because the targeted APEX device is already mounted on the Nios development board, you must assign the signals of the design to specific device pins.

The Quartus II software provides several methods for making pin assignments. You can assign pins individually with the **Assignment Organizer** (Tools menu) or with the **Pin Assignments** dialog box, or you can assign all necessary pins at once with a Tcl script. Because of the number of specific pin assignments to be made for this tutorial, you should use the Altera-provided Tcl script, **pin_assign.tcl**, which is located in **c:\altera\excalibur\sopc_builder_2_5\tutorial\Nios_Tutorial**, to make the appropriate pin assignments easily.

☞ Be sure to compare the pin names in your BDF with the pin assignments to make sure the names are the same. The design will not run on the board if pin names are misspelled or swapped.

This session includes the following steps:

1. "Assign Pins with a Tcl Script" on page 49

2. "Verify the Pin Assignments" on page 49

## Assign Pins with a Tcl Script

To make pin assignments with the Altera-provided **pin_assign.tcl** Tcl script, follow these steps:

1. To open the Quartus II Tcl Console window, choose **Auxiliary Windows > Tcl Console** (View menu). The Quartus II Tcl Console window appears.

2. At the Quartus II Tcl Console command prompt, type the following command:

   ```
   source pin_assign.tcl ↵
   ```

3. The Tcl script is executed and assigns all necessary pins. When the assignments are complete, the `assignment made` message appears in the Tcl Console window.

## Verify the Pin Assignments

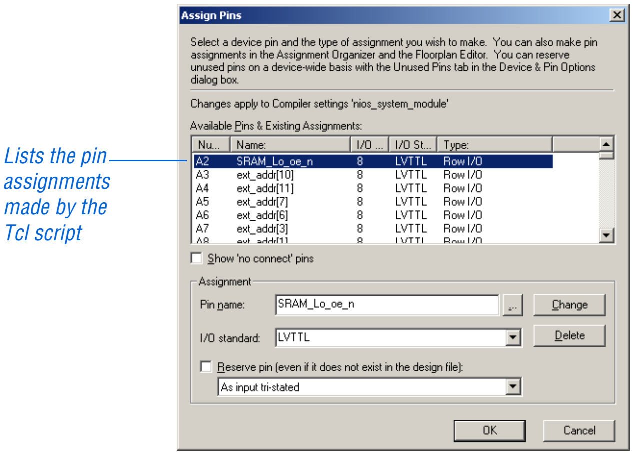
To verify the pin assignments, follow these steps.

1. Choose **Compiler Settings** (Processing menu). The **General** tab of the **Compiler Settings** dialog box appears automatically.

2. Click the **Chips & Devices** tab.

3. Click **Assign Pins**. The **Pin Assignments** dialog box appears with the new pin assignments listed in the **Available Pins & Existing Assignments** list. See Figure 3.

*Figure 3. Verify Pin Assignments*



Select a device pin and the type of assignment you wish to make. You can also make pin assignments in the Assignment Organizer and the Floorplan Editor. You can reserve unused pins on a device-wide basis with the Unused Pins tab in the Device & Pin Options dialog box.

Changes apply to Compiler settings 'nios_system_module'

Available Pins & Existing Assignments:

| Nu... | Name: | I/O ... | I/O St... | Type: |
|---|---|---|---|---|
| A2 | SRAM_Lo_oe_n | 8 | LVTTL | Row I/O |
| A3 | ext_addr[10] | 8 | LVTTL | Row I/O |
| A4 | ext_addr[11] | 8 | LVTTL | Row I/O |
| A5 | ext_addr[7] | 8 | LVTTL | Row I/O |
| A6 | ext_addr[6] | 8 | LVTTL | Row I/O |
| A7 | ext_addr[3] | 8 | LVTTL | Row I/O |
| A8 | ext_addr[1] | 8 | LVTTL | Row I/O |

☐ Show 'no connect' pins

Assignment

Pin name: SRAM_Lo_oe_n

I/O standard: LVTTL

☐ Reserve pin (even if it does not exist in the design file):

As input tri-stated

4. When you are done viewing the pin assignments, click **OK**. You are returned to the **Chips & Devices** tab in the **Compiler Settings** dialog box.

## Specify Device, Programming & EDA Tool Settings

Before compiling the design, you must specify options that control the use of unused pins, optional programming file generation, and EDA tool settings. This section includes the follow steps:

1. "Reserve Unused Pins" on page 50
2. "Specify Optional Programming Files" on page 51
3. "Specify EDA Tool Settings" on page 51

### Reserve Unused Pins

To specify options for reserving unused pins, follow these steps:

If you do not follow these steps, your design will not run on the Nios development board.

1. In the **Chips & Devices** tab of the **Compiler Settings** dialog box, click **Device & Pin Options**. The **General** tab of the **Device & Pin Options** dialog box appears automatically.
2. Click the **Unused Pins** tab.
3. Under **Reserve all unused pins**, select **As inputs, tri-stated**.

## Specify Optional Programming Files

By default, the Compiler always generates an SRAM Object File (**.sof**). The Programmer uses the SOF to configure an APEX device with your design. However, you can also direct the Compiler to generate other optional programming files during compilation. For example, you can generate a Hexadecimal (Intel-Format) Output File (**.hexout**) that can be used to download your design to the user-configuration area of the flash memory device provided on the Nios development board.

☞ You will use the.hexout file in a later step to download your design to the flash memory on the Nios development board.

To specify optional programming files, follow these steps:

1. In the **Device & Pin Options** dialog box, click the **Programming Files** tab.

2. Turn on **Hexadecimal (Intel-Format) Output File (.hexout)**.

3. To accept the remaining defaults and save the device and pin options, click **OK**.

4. In the **Chips & Devices** tab, click **OK**.

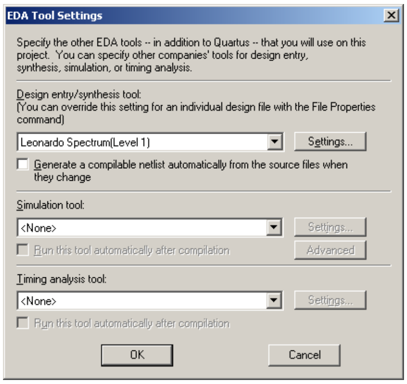
## Specify EDA Tool Settings

To specify the appropriate EDA tool settings for use when compiling a design synthesized with the LeonardoSpectrum software, follow these steps:

1. Choose **EDA Tool Settings** (Project menu). The **EDA Tool Settings** dialog box appears.

2. Under **Design entry/synthesis tool**, select **Leonardo Spectrum (Level 1)**. See Figure 4.

*Figure 4. EDA Tool Settings*


EDA Tool Settings

Specify the other EDA tools -- in addition to Quartus -- that you will use on this project. You can specify other companies' tools for design entry, synthesis, simulation, or timing analysis.

Design entry/synthesis tool:
(You can override this setting for an individual design file with the File Properties command)

Leonardo Spectrum(Level 1) | Settings...

☐ Generate a compilable netlist automatically from the source files when they change

Simulation tool:

<None> | Settings... | Advanced

☐ Run this tool automatically after compilation

Timing analysis tool:

<None> | Settings...

☐ Run this tool automatically after compilation

OK | Cancel

3. Click **OK**.

## 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 and any optional programming files.

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

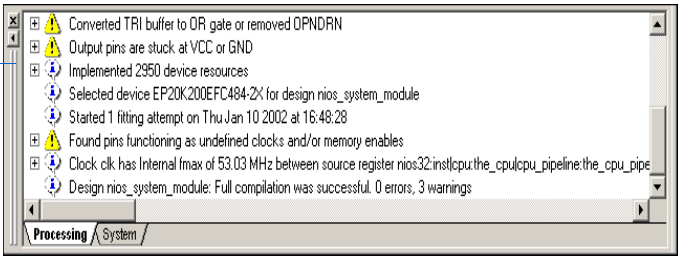
1. Choose **Start Compilation** (Processing menu). You also click the Compile toolbar button.

   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 your system.

The Compiler may generate one or more of the following warning messages that do not affect the outcome of your design (see Figure 5).

**Figure 5. Compiler Messages**

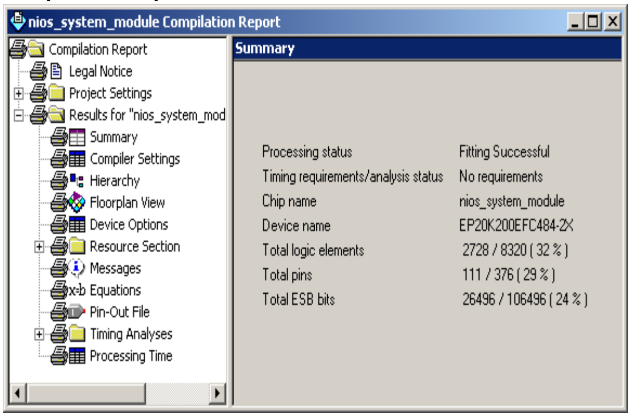
*Warning message*



```
Converted TRI buffer to OR gate or removed OPNDRN
Output pins are stuck at VCC or GND
Implemented 2950 device resources
Selected device EP20K200EFC484-2X for design nios_system_module
Started 1 fitting attempt on Thu Jan 10 2002 at 16:48:28
Found pins functioning as undefined clocks and/or memory enables
Clock clk has Internal fmax of 53.03 MHz between source register nios32:inst|cpu:the_cpu|cpu_pipeline:the_cpu_pipe
Design nios_system_module: Full compilation was successful. 0 errors, 3 warnings
```

2. When compilation completes, you can view the results in the nios_system_module Compilation Report window. See Figure 6.

**Figure 6. Compilation Report**



| | |
|---|---|
| Processing status | Fitting Successful |
| Timing requirements/analysis status | No requirements |
| Chip name | nios_system_module |
| Device name | EP20K200EFC484-2X |
| Total logic elements | 2728 / 8320 ( 32 % ) |
| Total pins | 111 / 376 ( 29 % ) |
| Total ESB bits | 26496 / 106496 ( 24 % ) |

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.

*Notes:*

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 APEX device with the ByteBlasterMV communications cable connected to the JTAG port (JP3), located next to the power supply connector (J1) on the Nios development board. If you have a Nios design running a GERMS monitor, you can also download configuration data to a flash memory device on the Nios development board over a serial port using a terminal emulation program. Then, you can configure the APEX device using the data stored in flash memory.

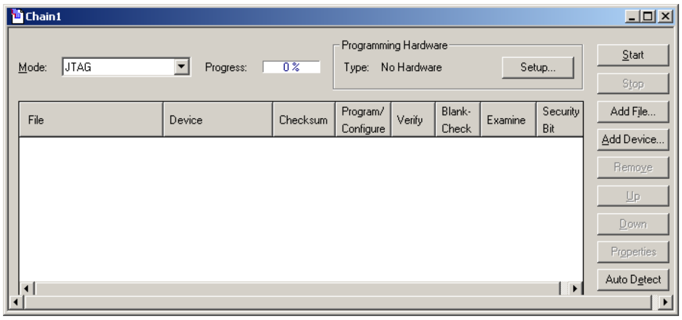
## Configure an APEX Device

Once you have properly connected and set up the ByteBlasterMV cable to transmit configuration data over the JTAG port, you can configure the APEX device on the Nios development board with your design.

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

1. Choose **Open Programmer** (Processing menu). The Programmer window opens a blank Chain Description File (**.cdf**), as shown in Figure 1.

*Figure 1. 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. Under **Programming Hardware**, click **Setup**. The **Hardware Setup** dialog box appears.

8. In the **Hardware Type** list, select **ByteBlasterMV**.

   ☞ If the ByteBlasterMV cable does not appear in the **Hardware Type** list, you may not have set up the ByteBlaster driver for your system. Refer to "Installing the ByteBlasterMV Parallel Port Download Cable" in the *Quartus II Installation & Licensing Manual for PCs* for instructions on setting up the driver.

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

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

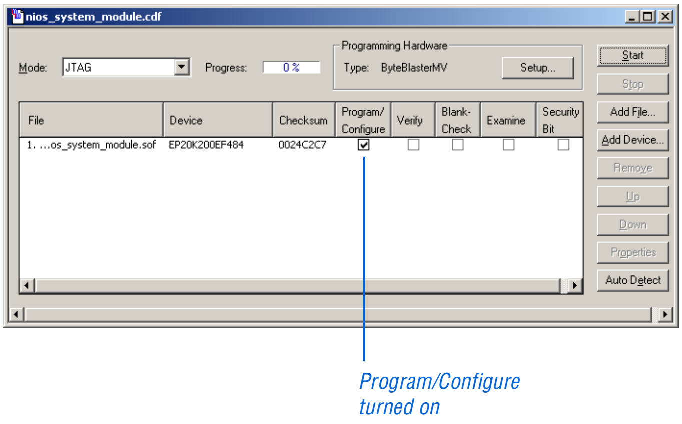
11. Click **Add File**. The **Select Programming File** dialog box appears.

12. Specify the **nios_system_module.sof** file in the **File name** box.

13. Click **Open**. The SOF is listed in the Programmer window.

14. In the Programmer window, turn on **Program/Configure**. See Figure 2.

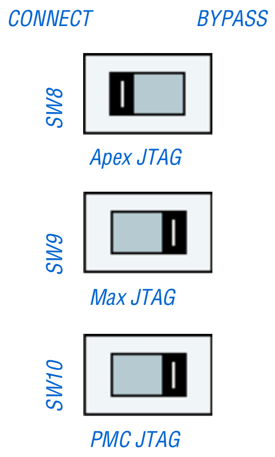
*Figure 2. Turn on Program/Configure Option*



| File | Device | Checksum | Program/ Configure | Verify | Blank-Check | Examine | Security Bit |
|---|---|---|---|---|---|---|---|
| 1. ...os_system_module.sof | EP20K200EF484 | 0024C2C7 | ✓ | ☐ | ☐ | ☐ | ☐ |

15. On the Nios development board, make sure switch SW8 is in the CONNECT position, and that switches SW9 and SW10 are in the BYPASS position. Figure 3 illustrates the correct configuration of the JTAG switches:

*Figure 3. Nios Development Board JTAG Switch Configuration*



16. Click **Start**. The Programmer begins to download the configuration data to the APEX device. 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 APEX device, the following events occur:

- The instructions specified in the boot_monitor_rom system peripheral are executed. The GERMS Monitor performs the following system initialization tasks:

  – Disables interrupts on the UART, timer, and switch PIO.
  – Sets the Stack Pointer register to `0x080000` (top of the RAM for this design).
  – Examines two flash memory bytes at `0x14000C` 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 `0x14000C` contain `N` and `i`, it executes a call to location `0x140000`. By default, this call loads a sample peripheral test program that you use later in this tutorial.

- If configuration completes successfully, LED1 and LED2 are illuminated on the Nios development board.

  ☞ If you are unable to configure the device correctly, you can press the RESET button (SW2) on the Nios development board to reload the factory default reference design and continue the tutorial.

## Running Software in 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 and 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 APEX device.

This section contains the following steps:

1. "Nios SDK Shell Tips" on page 59
2. "Start the Nios SDK Shell" on page 59
3. "Run the Sample hello_nios.srec Test Program" on page 60

## Nios SDK Shell Tips

The following tips make using the **Nios SDK Shell** easier:

- The **Nios SDK Shell** opens to the **/altera/excalibur/sopc_builder_2_5/examples** directory by default. To change to your SDK directory, type:

  ```
  cd ../tutorials/Nios_Tutorial/<CPU name>_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_2<Tab Key>/*_sdk/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

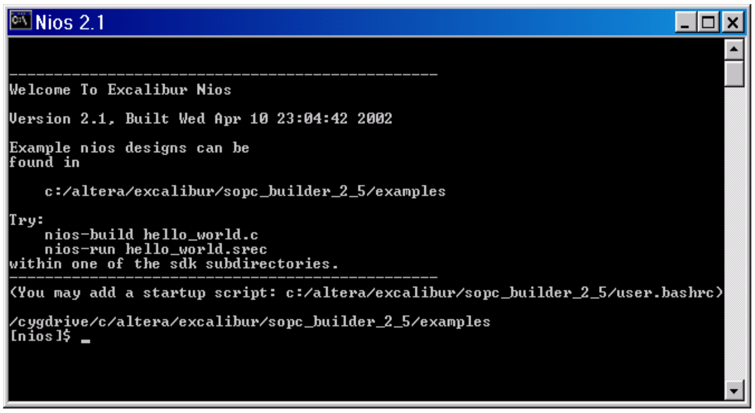
The **Nios SDK Shell** is a UNIX-like command shell that allows you to run the **nios-build** and **nios-run** utilities and various test programs on the Nios development board.

To start the **Nios SDK Shell**, follow these steps:

1. Make sure to connect the 9-pin serial cable to the serial connector on the Nios development board (J3) and to a serial port on your PC, as described in the *Nios Embedded Processor Getting Started User Guide*.

2. Choose **Programs > Altera > Excalibur Nios 2.1 > 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 4.

*Figure 4. Nios SDK Shell*



```
------------------------------------------------
Welcome To Excalibur Nios

Version 2.1, Built Wed Apr 10 23:04:42 2002

Example nios designs can be
found in

    c:/altera/excalibur/sopc_builder_2_5/examples

Try:
    nios-build hello_world.c
    nios-run hello_world.srec
within one of the sdk subdirectories.
------------------------------------------------
(You may add a startup script: c:/altera/excalibur/sopc_builder_2_5/user.bashrc)

/cygdrive/c/altera/excalibur/sopc_builder_2_5/examples
[nios]$ _
```

## 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 APEX device. The **hello_nios.c** program is located in the **c:\altera\excalibur\sopc_builder_2_5\tutorials\Nios_Tutorial\nios32_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/Excalibur/sopc_builder_2_5/tutorials/
   Nios_Tutorial/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.

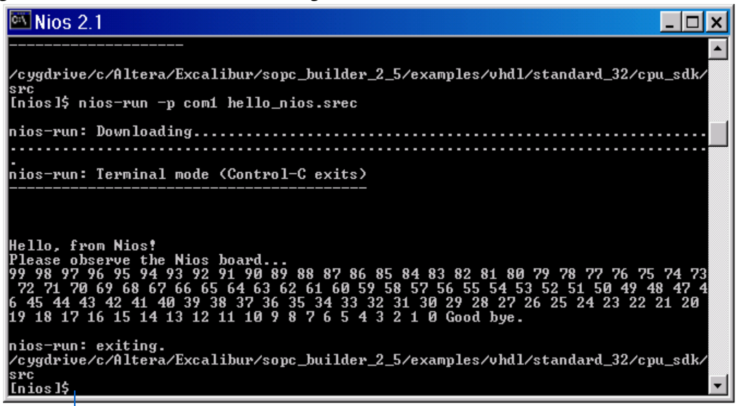
3. To run the **hello_nios.srec** program, type the following command at the **Nios SDK Shell** prompt:

   `nios-run -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 attempts to use COM1 by default.

   The **nios-run** utility runs the **hello_nios.srec** program on the Nios system module you created. This program generates the message `Hello, from Nios!` and causes the 2-digit 7-segment LED (D1) to count down from 99 to 00. The **Nios SDK Shell** prompt appears when the **hello_nios.srec** program is complete. See Figure 5.

*Figure 5. Run hello_nios.srec Program*



```
--------------------

/cygdrive/c/Altera/Excalibur/sopc_builder_2_5/examples/vhdl/standard_32/cpu_sdk/
src
[nios]$ nios-run -p com1 hello_nios.srec

nios-run: Downloading...........................................................
................................................................................
.
nios-run: Terminal mode (Control-C exits)
-----------------------------------------



Hello, from Nios!
Please observe the Nios board...
99 98 97 96 95 94 93 92 91 90 89 88 87 86 85 84 83 82 81 80 79 78 77 76 75 74 73
 72 71 70 69 68 67 66 65 64 63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 4
6 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20
19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 Good bye.

nios-run: exiting.
/cygdrive/c/Altera/Excalibur/sopc_builder_2_5/examples/vhdl/standard_32/cpu_sdk/
src
[nios]$
```

4. When the **hello_nios.srec** program is complete, press the CLEAR button (SW3) on the Nios development board to clear the **hello_nios.srec** program from the Nios embedded processor.

☞ The CLEAR button (SW3) is tied to the reset pin in the Nios system module. Pressing the CLEAR button is the same as performing a power-on-reset on the microprocessor.

## 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 GERMS Monitor to erase the user-configurable section of the flash memory device on the Nios development board. Then, you can use the **nios-run** utility to download the configuration data to the flash memory device on the Nios development board.

You can use the **nios-run** utility to switch the **Nios SDK Shell** to terminal mode. The terminal mode allows you to communicate with the Nios development board through the GERMS Monitor. By sending commands through the GERMS Monitor, you can erase the user section of the flash memory device and download the **nios_system_module.hexout** file that stores hardware configuration data for your project.

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

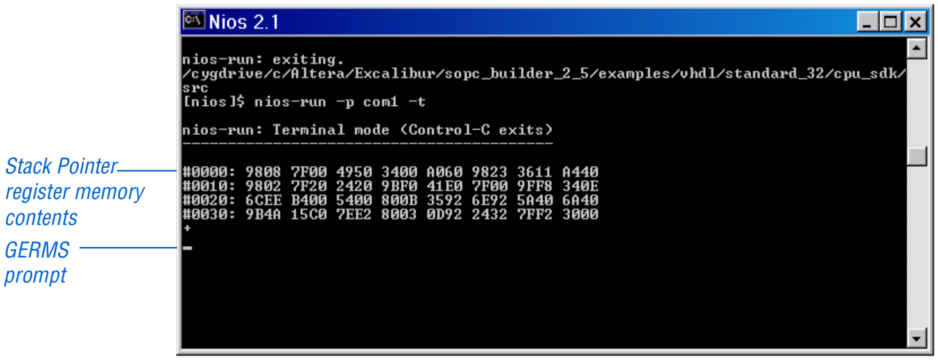
1. To use the **nios-run** utility in terminal mode, type the following command at the **Nios SDK Shell** prompt:

   `nios-run -p com`*<com port number>* `-t` ↵

   The **nios-run** utility is now in terminal mode.

2. At the command prompt, press ↵. The GERMS Monitor displays the four lines of eight half-words located at the Stack Pointer register memory address, followed by the GERMS Monitor command prompt (+). See Figure 6.

*Figure 6. Downloading Configuration Data to Flash Memory*



```
nios-run: exiting.
/cygdrive/c/Altera/Excalibur/sopc_builder_2_5/examples/vhdl/standard_32/cpu_sdk/
src
[nios]$ nios-run -p com1 -t

nios-run: Terminal mode (Control-C exits)
-----------------------------------------

#0000: 9808 7F00 4950 3400 A060 9823 3611 A440
#0010: 9802 7F20 2420 9BF0 41E0 7F00 9FF8 340E
#0020: 6CEE B400 5400 800B 3592 6E92 5A40 6A40
#0030: 9B4A 15C0 7EE2 8003 0D92 2432 7FF2 3000
+
_
```

3. To erase the user section of the flash memory contents, beginning at memory address 0x180000 for this design, type the following commands at the GERMS Monitor prompt:

```
e180000 ↵
e190000 ↵
e1a0000 ↵
e1b0000 ↵
r180000 ↵
```

☞ If you make a mistake entering any of these commands, you cannot use the Backspace key to correct the error. Instead, you must press Esc and re-enter the entire line.

The **e** command erases the block of flash memory at the specified hexadecimal flash memory address. The **r** command relocates the Stack Pointer register to the specified hexadecimal flash memory address.

☞ For more information on the GERMS monitor commands, refer to the *Nios Embedded Processor Software Development Reference Manual*.

4. To exit terminal mode, press Ctrl + C. The **Nios SDK Shell** prompt appears.

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

```
cd ../.. ↵
```

6. 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 -p com<com port number>
  nios_system_module.hexout ↵
```

The **nios-run** utility begins to download the configuration data to the flash memory device on the Nios development board, beginning at memory address `0x180000`. 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 a `!`, an error was encountered during the flash programming. If you receive a `!`, go back to step 4 and begin the programming process again.

7. To configure the APEX device with the **nios_system_module** data stored in the flash memory, press the RESET button (SW2) on the Nios development board. The APEX device is configured with the **nios_system_module** data stored in the flash memory device. When the configuration is complete, LED1 and LED2 are lit on the Nios development board and the GERMS monitor displays the text `tutorial`.

## Restore the Factory Default Configuration

After you are finished using the **nios_system_module** design on the Nios development board, you can optionally restore the board to its factory default state and run a peripheral test program.

To restore the Nios development board to its factory default configuration and run the peripheral test program, follow these steps:

1. If the pins at JP2 on the Nios development board do not have a shunt across them, add a shunt.

2. Using the Quartus II Programmer, download the **standard_32.sof** file, which is located in the **\examples\**<*verilog or vhdl*>**\standard_32** directory to the Nios development board.

3. Open the **Nios SDK Shell**.

4. Change to the **/altera/excalibur/examples** directory.

5. Download the factory default flash file to the board using the **nios-run** command:

```
nios-run -p com<com port number>
   standard_32_devboard.flash ↵
```

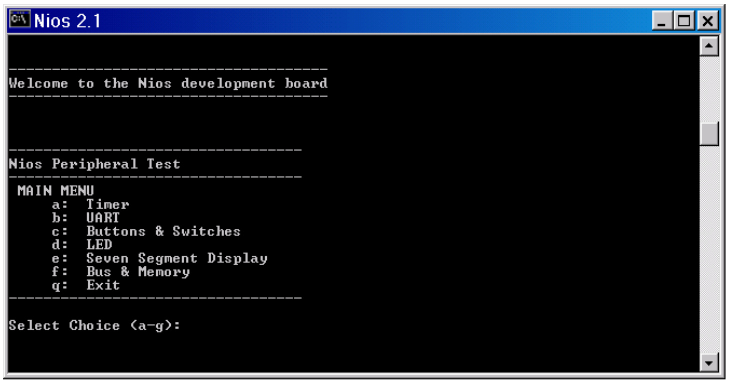
   The **nios-run** utility downloads the configuration data to the flash memory device on the Nios development board. This task may require a few minutes to complete. The **nios-run** utility returns to terminal mode when the download is complete.

☞ If downloading is proceeding successfully, the **Nios SDK Shell** displays a string of periods (`.`). If the shell displays a `!`, an error was encountered during the flash programming. If you receive a `!`, go back to step 2 and begin the programming process again.

6. When the file finishes downloading, press the RESET button (SW2). The **nios_system_module** design is cleared from the APEX device and the factory default design **reference_design_32_bit** is loaded. The Nios Peripheral Test Main Menu appears in the **Nios SDK Shell** window, as shown in Figure 7.

☞ If you press the RESET button without connecting the jumper pins at JP2, your Nios system module design is configured into the APEX device. The configuration controller tries to load this section of flash data into the APEX device first if the jumper pins at JP2 are not connected.

***Figure 7. Restoring Factory Defaults***

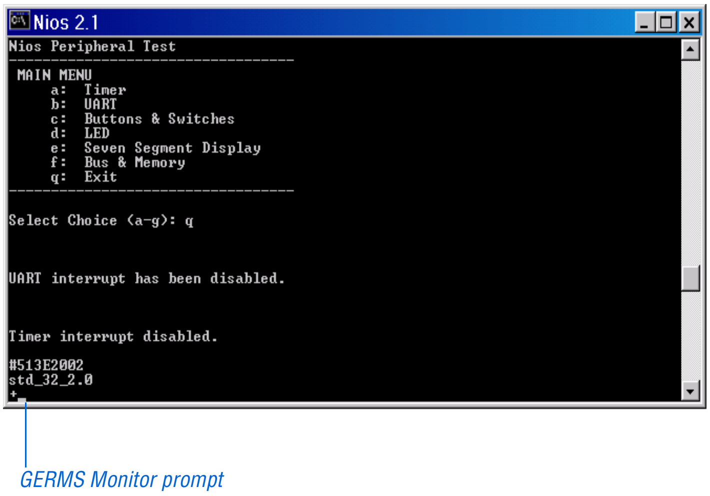


```
----------------------------------------
Welcome to the Nios development board
----------------------------------------



------------------------------------
Nios Peripheral Test
------------------------------------
 MAIN MENU
     a:  Timer
     b:  UART
     c:  Buttons & Switches
     d:  LED
     e:  Seven Segment Display
     f:  Bus & Memory
     q:  Exit
------------------------------------

Select Choice (a-g):
```

7. If you want to test any of the peripherals on the development board, type **a** through **f** to select a peripheral from the Main Menu.

8. When you are done testing the development board peripherals, return to the Main Menu and type **q** to exit the peripheral test program and return to terminal mode. The GERMS Monitor prompt appears, as shown in Figure 8.

*Figure 8. GERMS Monitor Prompt*



```
Nios Peripheral Test
-----------------------------------
 MAIN MENU
     a:  Timer
     b:  UART
     c:  Buttons & Switches
     d:  LED
     e:  Seven Segment Display
     f:  Bus & Memory
     q:  Exit
-----------------------------------

Select Choice (a-g): q


UART interrupt has been disabled.


Timer interrupt disabled.

#513E2002
std_32_2.0
+_
```

You can stop the GERMS Monitor from automatically running the peripheral test program stored in flash memory by pressing and releasing the CLEAR (SW3) button while pressing and holding the SW4 button on the Nios development board.

## 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 new hardware features provided with the Nios processor version 2.1, 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*
- For additional software information, refer to:
  - *Nios Embedded Processor Software Development Reference Manual*
  - *Nios 16-Bit Programmer's Reference Manual*
  - *Nios 32-Bit Programmer's Reference Manual*
- A variety of reference designs that can be downloaded and used on the Nios development board are located in the **c:\altera\excalibur\sopc_builder_2_5\examples** directory.

# Index

*Notes:*