

Nios[®] II

Nios II Hardware Development

Tutorial



101 Innovation Drive
San Jose, CA 95134
www.altera.com

Document Version: 3.0
Document Date: December 2009

TU-N2HWDV-3.0

Copyright © 2009 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, maskwork rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.



Chapter 1. Nios II Hardware Development

Introduction	1-1
Design Example	1-1
Software and Hardware Requirements	1-2
OpenCore Plus Evaluation	1-3
Nios II System Development Flow	1-3
Analyzing System Requirements	1-4
Defining and Generating the System in SOPC Builder	1-5
Integrating the SOPC Builder System into the Quartus II Project	1-6
Developing Software with the Nios II Software Build Tools for Eclipse	1-6
Running and Debugging Software on the Target Board	1-7
Varying the Development Flow	1-7
Creating the Design Example	1-8
Install the Design Files	1-8
Analyze System Requirements	1-9
Start the Quartus II Software and Open the Example Project	1-9
Create a New SOPC Builder System	1-10
Define the System in SOPC Builder	1-11
Integrate the SOPC Builder System into the Quartus II Project	1-21
Download Hardware Design to Target FPGA	1-27
Develop Software Using the Nios II Software Build Tools for Eclipse	1-28
Run the Program on Target Hardware	1-31
Taking the Next Step	1-31

Additional Information

Revision History	Info-1
How to Contact Altera	Info-1
Typographic Conventions	Info-1

Introduction

This tutorial introduces you to the system development flow for the Nios® II processor. Using the Quartus® II software and the Nios II Embedded Design Suite (EDS), you build a Nios II hardware system design and create a software program that runs on the Nios II system and interfaces with components on Altera® development boards. The tutorial is a good starting point if you are new to the Nios II processor or the general concept of building embedded systems in FPGAs.

Building embedded systems in FPGAs involves system requirements analysis, hardware design tasks, and software design tasks. This tutorial guides you through the basics of each topic, with special focus on the hardware design steps. Where appropriate, the tutorial refers you to further documentation for greater detail.

 If you are interested only in software development for the Nios II processor, refer to the tutorial in the *Getting Started with the Graphical User Interface* chapter of the *Nios II Software Developer's Handbook*.

When you complete this tutorial, you will understand the Nios II system development flow, and you will be able to create your own custom Nios II system.

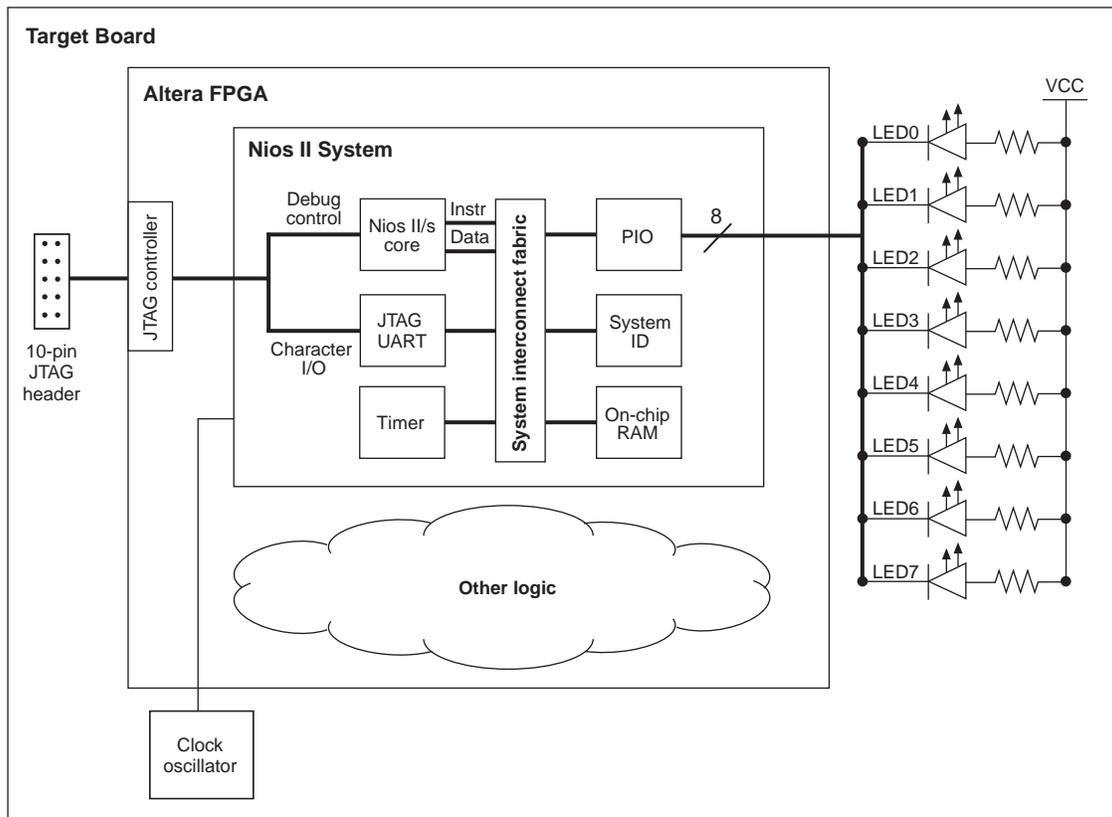
Design Example

The design example you build in this tutorial demonstrates a small Nios II system for control applications, that displays character I/O output and blinks LEDs in a binary counting pattern. This Nios II system can also communicate with a host computer, allowing the host computer to control logic inside the FPGA.

The example Nios II system contains the following components:

- Nios II/s processor core
- On-chip memory
- Timer
- JTAG UART
- 8-bit parallel I/O (PIO) pins to control the LEDs
- System identification component

[Figure 1–1](#) is a block diagram showing the relationship between the host computer, the target board, the FPGA, and the Nios II system.

Figure 1–1. Tutorial Design Example

As shown in [Figure 1–1](#), other logic can exist within the FPGA alongside the Nios II system. In fact, most FPGA designs with a Nios II system also include other logic. A Nios II system can interact with other on-chip logic, depending on the needs of the overall system. For the sake of simplicity, the design example in this tutorial does not include other logic in the FPGA.

Software and Hardware Requirements

This tutorial requires you to have the following software:

- Altera Quartus II software version 9.1 or later—The software must be installed on a Windows or Linux computer that meets the Quartus II minimum requirements.

 For system requirements and installation instructions, refer to [Altera Software Installation and Licensing](#).

- Nios II EDS version 9.1 or later.
- Design files for the design example—A hyperlink to the design files appears next to this document on the [Literature: Nios II Processor](#) page of the Altera website.

You can build the design example in this tutorial with any Altera development board or your own custom board that meets the following requirements:

- The board must have an Altera Stratix® series, Cyclone® series, or Arria® series FPGA.
- The FPGA must contain a minimum of 2500 logic elements (LE) or adaptive look-up tables (alut).
- The FPGA must contain a minimum of 50 M4K or M9K memory.
- An oscillator must drive a constant clock frequency to an FPGA pin. The maximum frequency limit depends on the speed grade of the FPGA. Frequencies of 50 MHz or less should work for most boards; higher frequencies might work.
- FPGA I/O pins can optionally connect to eight or fewer LEDs to provide a visual indicator of processor activity.
- The board must have a JTAG connection to the FPGA that provides a programming interface and communication link to the Nios II system. This connection can be either a dedicated 10-pin JTAG header for an Altera USB-Blaster download cable (revision B or higher) or a USB connection with USB-Blaster circuitry embedded on the board.



To complete this tutorial, you must refer to the documentation for your board that describes clock frequencies and pinouts. For Altera development boards, you can find this information in the associated reference manual.



For information about Altera development kits and development boards, refer to the [Literature: Development Kits](#) page of the Altera website.

OpenCore Plus Evaluation

You can perform this tutorial on hardware without a license. With Altera's free OpenCore Plus evaluation feature, you can perform the following actions:

- Simulate the behavior of a Nios II processor within your system
- Verify the functionality of your design
- Evaluate the size and speed of your design quickly and easily
- Generate time-limited device programming files for designs that include Nios II processors
- Program a device and verify your design in hardware

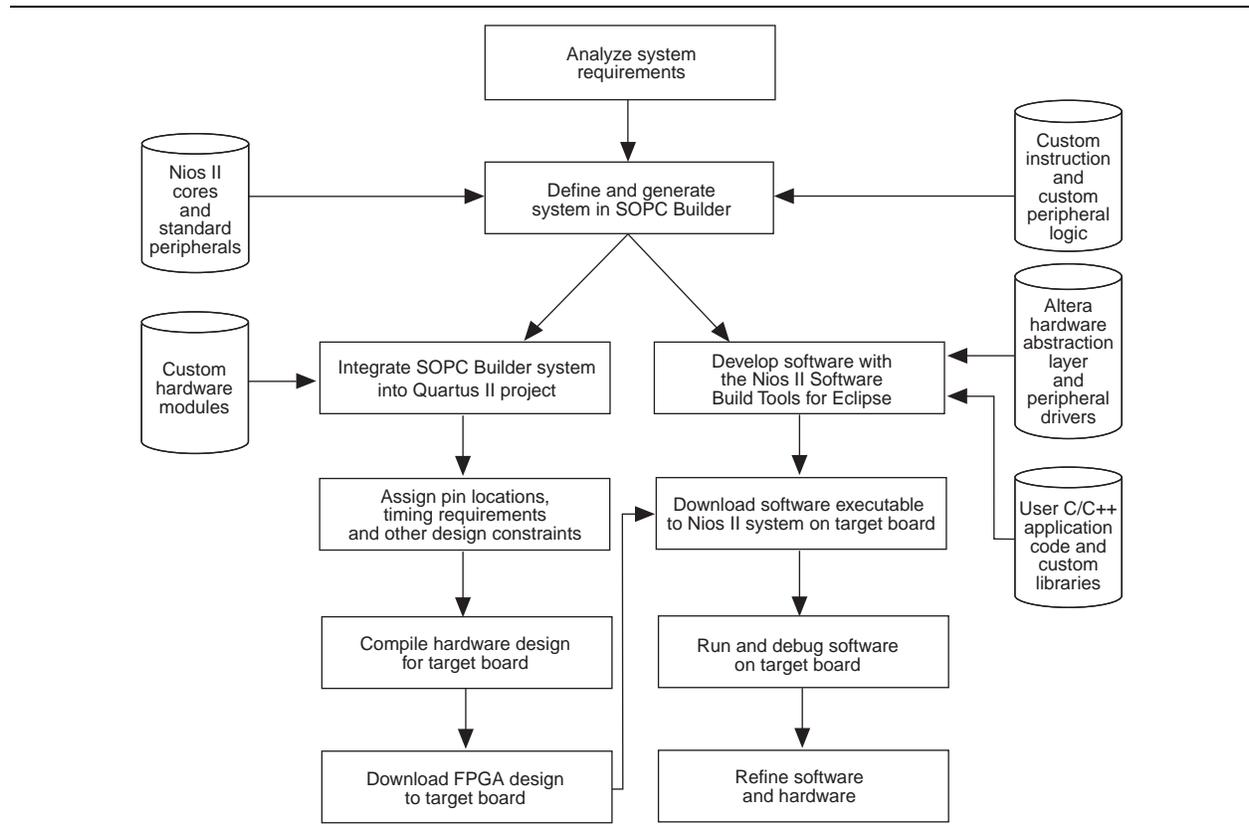
You need to purchase a license for the Nios II processor only when you are completely satisfied with its functionality and performance, and want to use your design in production.



For more information about OpenCore Plus, refer to the [AN320: OpenCore Plus Evaluation of Megafunctions](#).

Nios II System Development Flow

This section discusses the complete design flow for creating a Nios II system and prototyping it on a target board. [Figure 1-2](#) shows the Nios II system development flow.

Figure 1-2. Nios II System Development Flow

The Nios II development flow consists of three types of development: hardware design steps, software design steps, and system design steps, involving both hardware and software. For simpler Nios II systems, one person might perform all steps. For more complex systems, separate hardware and software designers might be responsible for different steps. System design steps involve both the hardware and software, and might require input from both sides. In the case of separate hardware and software teams, it is important to know exactly what files and information must be passed between teams at the points of intersection in the design flow.

The design steps in this tutorial focus on hardware development, and provide only a simple introduction to software development.

 After completing this tutorial, refer to the *Nios II Software Developer's Handbook*, especially the tutorial in the *Getting Started with the Graphical User Interface* chapter, for further details about the software development process. The handbook is a complete reference for developing software for the Nios II processor.

Analyzing System Requirements

The development flow begins with predesign activity which includes an analysis of the application requirements, such as the following questions:

- What computational performance does the application require?
- How much bandwidth or throughput does the application require?

- What types of interfaces does the application require?
- Does the application require multithreaded software?

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

- Which Nios II processor core to use: smaller or faster.
- What components the design requires and how many of each kind.
- Which real-time operating system (RTOS) to use, if any.
- Where hardware acceleration logic can dramatically improve system performance. For example:
 - Could adding a DMA component eliminate wasted processor cycles copying data?
 - Could a custom instruction replace the critical loop of a DSP algorithm?

Analyzing these topics involve both the hardware and software teams.

Defining and Generating the System in SOPC Builder

After analyzing the system hardware requirements, you use SOPC Builder to specify the Nios II processor core(s), memory, and other components your system requires. SOPC Builder automatically generates the interconnect logic to integrate the components in the hardware system.

You can select from a list of standard processor cores and components provided with the Nios II EDS. You can also add your own custom hardware to accelerate system performance. You can add custom instruction logic to the Nios II core which accelerates CPU performance, or you can add a custom component which offloads tasks from the CPU. This tutorial covers adding standard processor and component cores, and does not cover adding custom logic to the system.

The primary outputs of SOPC Builder are the following file types:

- SOPC Builder Design File (**.sopc**)—Contains the hardware contents of the SOPC Builder system.
- SOPC Information File (**.sopcinfo**)—Contains a human-readable description of the contents of the **.sopc** file. The Nios II EDS uses the **.sopcinfo** file to compile software for the target hardware.
- Hardware description language (HDL) files—Are the hardware design files that describe the SOPC Builder system. The Quartus II software uses the HDL files to compile the overall FPGA design into an SRAM Object File (**.sof**).



For further details about the Nios II processor cores, refer to the *Nios II Processor Reference Handbook*. For further details about SOPC Builder and developing custom components, refer to *Volume 4: SOPC Builder* of the *Quartus II Handbook*. For further details about custom instructions, refer to the *Nios II Custom Instruction User Guide*.

Integrating the SOPC Builder System into the Quartus II Project

After generating the Nios II system using SOPC Builder, you integrate it into the Quartus II project. Using the Quartus II software, you perform all tasks required to create the final FPGA hardware design.

As shown in [Figure 1-1 on page 1-2](#), most FPGA designs include logic outside the Nios II system. You can integrate your own custom hardware modules into the FPGA design, or you can integrate other ready-made intellectual property (IP) design modules available from Altera or third party IP providers. This tutorial does not cover adding other logic outside the Nios II system.

Using the Quartus II software, you also assign pin locations for I/O signals, specify timing requirements, and apply other design constraints. Finally, you compile the Quartus II project to produce a `.sof` to configure the FPGA.

You download the `.sof` to the FPGA on the target board using an Altera download cable, such as the USB-Blaster™. After configuration, the FPGA behaves as specified by the hardware design, which in this case is a Nios II processor system.



For further information about using the Quartus II software, refer to [Introduction to the Quartus II Software](#), the [Quartus II Handbook](#), and the [Quartus II Software Interactive Tutorial](#) in the [Training Courses](#) section of the Altera website.

Developing Software with the Nios II Software Build Tools for Eclipse

Using the Nios II Software Build Tools for Eclipse, you perform all software development tasks for your Nios II processor system. After you generate the system with SOPC Builder, you can begin designing your C/C++ application code immediately with the Nios II Software Build Tools for Eclipse. Altera provides component drivers and a hardware abstraction layer (HAL) which allows you to write Nios II programs quickly and independently of the low-level hardware details. In addition to your application code, you can design and reuse custom libraries in your Nios II Software Build Tools for Eclipse projects.

To create a new Nios II C/C++ application project, the Nios II Software Build Tools for Eclipse uses information from the `.sopcinfo` file. You also need the `.sof` file to configure the FPGA before running and debugging the application project on target hardware.

The Nios II Software Build Tools for Eclipse can produce several outputs, listed below. Not all projects require all of these outputs.

- **system.h** file—Defines symbols for referencing the hardware in the system. The Nios II Software Build Tools for Eclipse automatically create this file when you create a new project.
- Executable and Linking Format File (**.elf**)—Is the result of compiling a C/C++ application project, that you can download directly to the Nios II processor.
- Hexadecimal (Intel-Format) File (**.hex**)—Contains initialization information for on-chip memories. The Nios II Software Build Tools for Eclipse generate these initialization files for on-chip memories that support initialization of contents.

- Flash memory programming data—Is boot code and other arbitrary data you might write to flash memory. The Nios II Software Build Tools for Eclipse includes a flash programmer, which allows you to write your program to flash memory. The flash programmer adds appropriate boot code to allow your program to boot from flash memory. You can also use the flash programmer to write arbitrary data to flash memory.

This tutorial focuses on downloading only the `.elf` directly to the Nios II system.

-  For complete details about developing software for the Nios II processor, refer to the *Nios II Software Developer's Handbook*.

Running and Debugging Software on the Target Board

The Nios II Software Build Tools for Eclipse provides complete facilities for downloading software to a target board, and running or debugging the program on hardware. The Nios II Software Build Tools for Eclipse debugger allows you to start and stop the processor, step through code, set breakpoints, and analyze variables as the program executes.

-  For details about running and debugging Nios II programs, refer to the tutorial in the *Getting Started with the Graphical User Interface* chapter of the *Nios II Software Developer's Handbook*.

Varying the Development Flow

The development flow is not strictly linear. This section describes common variations.

Refining the Software and Hardware

After running software on the target board, you might discover that the Nios II system requires higher performance. In this case, you can return to software design steps to make improvements to the software algorithm. Alternatively, you can return to hardware design steps to add acceleration logic. If the system performs multiple mutually exclusive tasks, you might even decide to use two (or more) Nios II processors that divide the workload and improve the performance of each individual processor.

Iteratively Creating a Nios II System

A common technique for building a complex Nios II system is to start with a simpler SOPC Builder system, and iteratively add to it. At each iteration, you can verify that the system performs as expected. You might choose to verify the fundamental components of a system, such as the processor, memory, and communication channels, before adding more complex components. When developing a custom component or a custom instruction, first integrate the custom logic into a minimal system to verify that it works as expected; later you can integrate the custom logic into a more complex system.

- Altera provides several working Nios II reference designs which you can use as a starting point for your own designs. After installing the Nios II EDS, refer to the <Nios II EDS install path>/**examples/verilog** or the <Nios II EDS install path>/**examples/vhdl** directory. Demonstration applications are also available in newer development kit installations.

Verifying the System with Hardware Simulation Tools

You can perform hardware simulation of software executing on the Nios II system, using tools such as the ModelSim® RTL simulator. Hardware simulation is useful to meet certain needs, including the following cases:

- To verify the cycle-accurate performance of a Nios II system before target hardware is available.
- To verify the functionality of a custom component or a Nios II custom instruction before trying it on hardware.

A hardware simulation step is not shown in [Figure 1-2 on page 1-4](#). If you are building a Nios II system based on the standard components provided with the Nios II EDS, the easiest way to verify functionality is to download the hardware and software directly to a development board.

- For details about performing hardware simulation for Nios II system, refer to the [AN351: Simulating Nios II Embedded Processor Designs](#).

Creating the Design Example

This section guides you through the Nios II development flow to create a working design example. You perform the following steps:

1. [“Install the Design Files” on page 1-8](#).
2. [“Analyze System Requirements” on page 1-9](#).
3. [“Start the Quartus II Software and Open the Example Project” on page 1-9](#).
4. [“Create a New SOPC Builder System” on page 1-10](#).
5. [“Define the System in SOPC Builder” on page 1-11](#).
6. [“Integrate the SOPC Builder System into the Quartus II Project” on page 1-21](#).
7. [“Download Hardware Design to Target FPGA” on page 1-27](#).
8. [“Develop Software Using the Nios II Software Build Tools for Eclipse” on page 1-28](#).
9. [“Run the Program on Target Hardware” on page 1-31](#).

Install the Design Files

Before you proceed, you must install the Quartus II software and the Nios II EDS. You must also download tutorial design files from the Altera web site. The design files provide a ready-made Quartus II project to use as a starting point.

- The design files appear next to this document on the [Literature: Nios II Processor](#) page of the Altera website.

Perform the following steps to set up the design environment:

1. Locate the zipped design files on the Altera web site.
2. Unzip the contents of the zip file to a directory on your computer. Do not use spaces in the directory path name.

The remainder of this tutorial refers to this directory as the *<design files directory>*.

Analyze System Requirements

This section describes the system requirements for the tutorial design example. The design example has the following goals:

- Demonstrate a simple Nios II processor system that you can use for control applications.
- Build a practical, real-world system, while providing an educational experience.
- Demonstrate the most common and effective techniques to build practical, custom Nios II systems.
- Build a Nios II system that works on any board with an Altera FPGA. The entire system must use only on-chip resources, and not rely on the target board.
- The design should conserve on-chip logic and memory resources so it can fit in a wide range of target FPGAs.

These goals lead to the following design decisions:

- The Nios II system uses only the following inputs and outputs:
 - One clock input, which can be any constant frequency.
 - Eight optional outputs to control LEDs on the target board.
- The design uses the following components:
 - Nios II/s core with 2 KB of instruction cache
 - 20 KB of on-chip memory
 - Timer
 - JTAG UART
 - Eight output-only parallel I/O (PIO) pins
 - System ID component



For complete details about these and other components, refer to *Volume 5: Embedded Peripherals* of the *Quartus II Handbook*.

Start the Quartus II Software and Open the Example Project

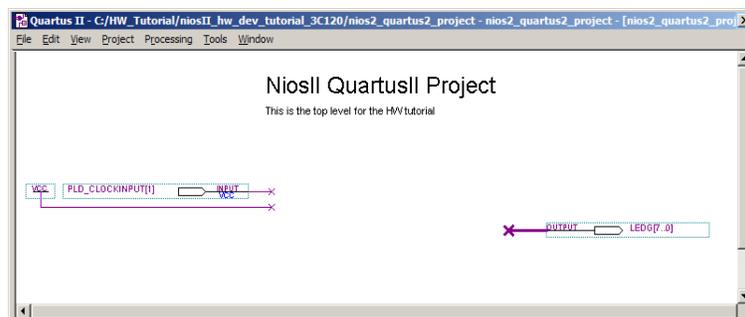
To start, you open the Quartus II project for the tutorial design example. This Quartus II project serves as an easy starting point for the Nios II development flow. The Quartus II project contains all settings and design files required to create the .sof.

To open the Quartus II project, perform the following steps:

1. Start the Quartus II software.
On Windows computers, click **Start**, point to **Programs, Altera, Quartus II <version>**, and then click **Quartus II <version>**. On Linux computers, type `quartus` at a shell command prompt, assuming the Quartus II program directory is in the search path.
2. On the File menu, click **Open Project**. The **Open Project** dialog box appears.
3. Browse to the *<design files directory>*.
4. Select the file `nios2_quartus2_project.qpf` and click **Open**.
5. If the Quartus II software does not automatically display the Block Diagram File (`.bdf`) `nios2_quartus2_project.bdf` (Figure 1-3), perform the following steps:
 - a. On the File menu, click **Open**. The **Open** dialog box appears.
 - b. Browse to the *<design files directory>*.
 - c. Select `nios2_quartus2_project.bdf` and click **Open**.

Figure 1-3 shows the `nios2_quartus2_project.bdf` file.

Figure 1-3. Design Example Block Diagram File



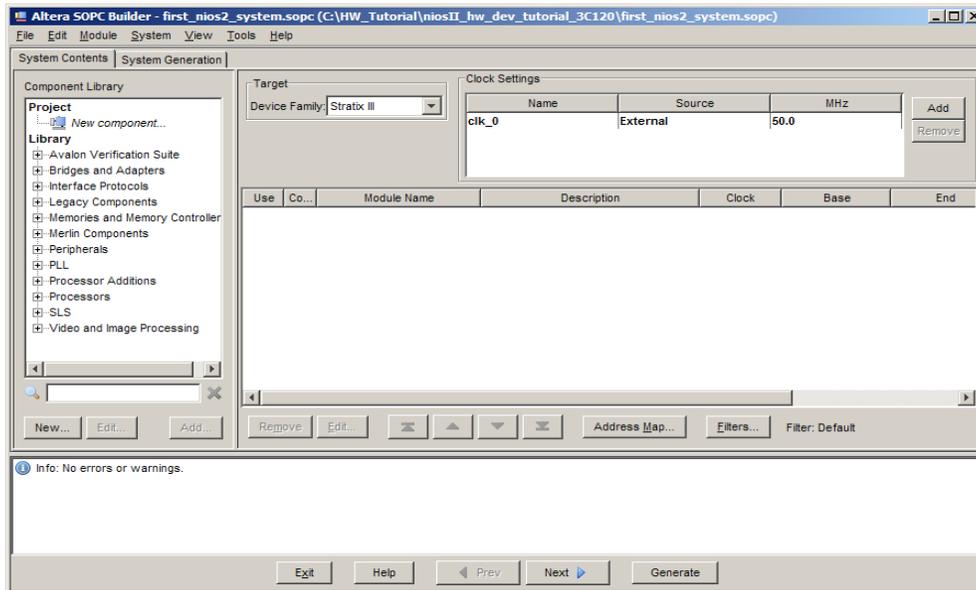
The `.bdf` contains an input pin for the clock input and eight output pins to drive LEDs on the board. Next, you create a new SOPC Builder system, which you ultimately connect to these pins.

Create a New SOPC Builder System

You use SOPC Builder to generate the Nios II processor system, adding the desired components, and configuring how they connect together. Perform the following steps to create a new SOPC Builder system:

1. On the Tools menu in the Quartus II software, click **SOPC Builder**. SOPC Builder starts and displays the **Create New System** dialog box.
2. Type `first_nios2_system` as the **System Name**.
3. Select either **Verilog** or **VHDL** as the **Target HDL**. If you do not have a preference, accept the default. Later when you generate the system, SOPC Builder outputs design files in the language you select.
4. Click **OK**. SOPC Builder opens, displaying the **System Contents** tab.

Figure 1-4 shows the SOPC Builder GUI in its initial state.

Figure 1-4. SOPC Builder GUI

Define the System in SOPC Builder

You use SOPC Builder to define the hardware characteristics of the Nios II system, such as which Nios II core to use, and what components to include in the system. SOPC Builder does not define software behavior, such as where in memory to store instructions or where to send the `stderr` character stream.

In this section, you perform the following steps:

1. Specify target FPGA and clock settings.
2. Add the Nios II core, on-chip memory, and other components.
3. Specify base addresses and interrupt request (IRQ) priorities.
4. Generate the SOPC Builder system.

The SOPC Builder design process does not need to be linear. The design steps in this tutorial are presented in the most straightforward order for a new user to understand. However, you can perform SOPC Builder design steps in a different order.

Specify Target FPGA and Clock Settings

The **Target** and **Clock Settings** sections of the **System Contents** tab specify the SOPC Builder system's relationship to other devices in the system. Perform the following steps:

1. Select the **Device Family** that matches the Altera FPGA you are targeting.



If a warning appears stating the selected device family does not match the Quartus project settings, ignore the warning. You specify the device in the Quartus project settings later in this tutorial.

2. In the documentation for your board, look up the clock frequency of the oscillator that drives the FPGA.



For Altera development board reference manuals, refer to the [Literature: Development Kits](#) page of the Altera website.

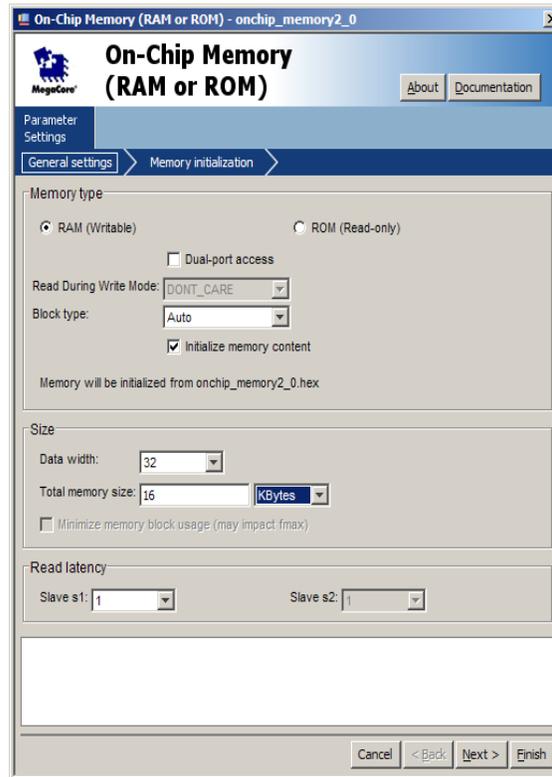
3. Double-click the clock frequency in the **MHz** column for `c1k_0`. `c1k_0` is the default clock input name for the SOPC Builder system. The frequency you specify for `c1k_0` must match the oscillator that drives the FPGA.
4. Type the clock frequency and press Enter.

Next, you begin to add hardware components to the SOPC Builder system. As you add each component, you configure it appropriately to match the design specifications.

Add the On-Chip Memory

Processor systems require at least one memory for data and instructions. This design example uses one 20 KB on-chip memory for both data and instructions. To add the memory, perform the following steps:

1. In the list of available components (on the left side of the **System Contents** tab), expand **Memories and Memory Controllers**, expand **On-Chip**, and then click **On-Chip Memory (RAM or ROM)**.
2. Click **Add**. The On-Chip Memory (RAM or ROM) MegaWizard interface appears. [Figure 1-5](#) shows the GUI.
3. In the **Block Type** list, select **Auto**.
4. In the **Total memory size** box, type 20 and select **KBytes** to specify a memory size of 20 KB.
5. Do not change any of the other default settings.

Figure 1-5. On-Chip Memory MegaWizard

- Click **Finish**. You return to the SOPC Builder **System Contents** tab, and an instance of the on-chip memory now appears in the table of available components.

 For further details about on-chip memory, you can click **Documentation** in the On-Chip Memory (RAM or ROM) MegaWizard interface. This documentation feature is available in the MegaWizard interface for each component.

- Right-click the on-chip memory and click **Rename**.
- Type `onchip_mem` and press Enter.

 You must type these tutorial component names exactly as specified. Otherwise, the tutorial programs written for this Nios II system fail in later steps. In general, it is a good habit to give descriptive names to hardware components. Nios II programs use these symbolic names to access the component hardware. Therefore, your choice of component names can make Nios II programs easier to read and understand.

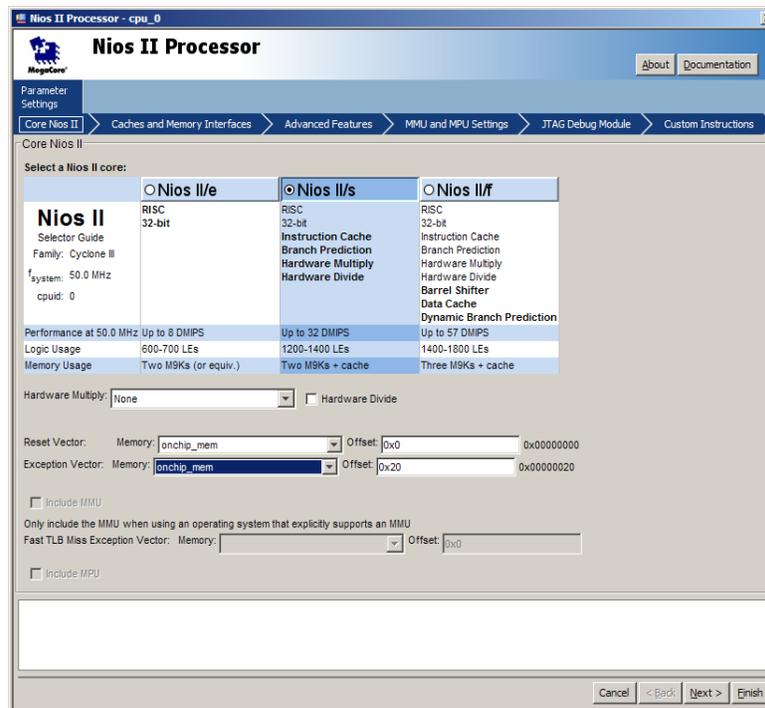
Add the Nios II Processor Core

In this section you add the Nios II/s core and configure it to use 2 KB of on-chip instruction cache memory. For educational purposes, the tutorial design example uses the Nios II/s "standard" core, which provides a balanced trade-off between performance and resource utilization. In reality, the Nios II/s core is more powerful than necessary for most simple control applications.

Perform the following steps to add a Nios II/s core to the system:

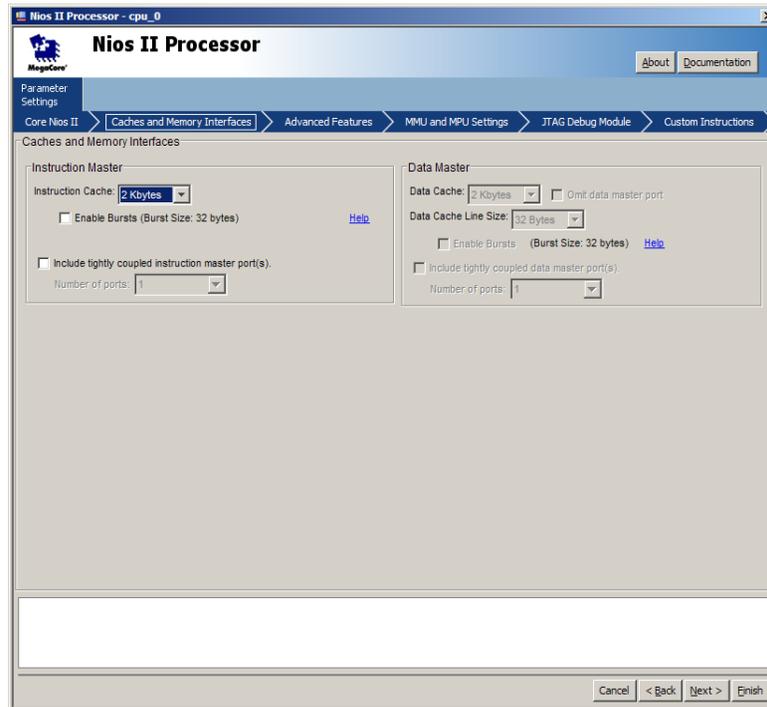
1. In the list of available components, expand **Processors**, and then click **Nios II Processor**.
2. Click **Add**. The Nios II Processor MegaWizard interface appears, displaying the **Nios II Core** page. [Figure 1-6](#) shows the GUI.
3. Under **Select a Nios II core**, select **Nios II/s**.
4. In the **Hardware Multiply** list, select **None**.
5. Turn off **Hardware Divide**.
6. Under **Reset Vector**, select **onchip_mem** in the **Memory** list and type **0x0** in the **Offset** box.
7. Under **Exception Vector**, select **onchip_mem** in the **Memory** list and type **0x20** in the **Offset** box.

Figure 1-6. Nios II MegaWizard – Nios II Core Page



8. Click **Caches and Memory Interfaces**. The **Caches and Memory Interfaces** page appears. [Figure 1-7](#) shows the GUI.
9. In the **Instruction Cache** list, select **2 Kbytes**.
10. Turn off **Enable Bursts**.
11. Turn off **Include tightly coupled instruction master port(s)**.

Figure 1-7. Nios II MegaWizard – Caches and Memory Interfaces page

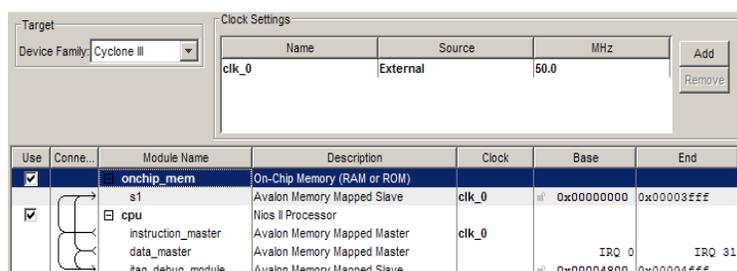


12. Do not change any settings on the **Advanced Features**, **MMU and MPU Settings**, **JTAG Debug Module**, or **Custom Instructions** pages.
13. Click **Finish**. You return to the SOPC Builder **System Contents** tab, and an instance of the Nios II core appears in the table of available components.

 For further details about configuring the Nios II core, refer to the *Instantiating the Nios II Processor in SOPC Builder* chapter of the *Nios II Processor Reference Handbook*.

 SOPC Builder automatically connects the instruction and data master ports on the Nios II core to the memory slave port. **Figure 1-8** shows the GUI. When building a system, always verify that SOPC Builder's automatic connections are appropriate for your system requirements.

Figure 1-8. System Contents Tab with the Nios II Core and On-Chip Memory



For further details about connecting memory to Nios II systems, refer to the *Building Memory Subsystems Using SOPC Builder* chapter in volume 4 of the *Quartus II Handbook*.

Add the JTAG UART

The JTAG UART provides a convenient way to communicate character data with the Nios II processor through the USB-Blaster download cable. Perform the following steps to add the JTAG UART:

1. In the list of available components, expand **Interface Protocols**, expand **Serial**, and then click **JTAG UART**.
2. Click **Add**. The JTAG UART MegaWizard interface appears. [Figure 1-9](#) shows the GUI.
3. Do not change the default settings.

Figure 1-9. JTAG UART MegaWizard



4. Click **Finish**. You return to the SOPC Builder **System Contents** tab, and an instance of the JTAG UART now appears in the table of available components.

 SOPC Builder automatically connects the data master port on the Nios II core to the JTAG UART slave port. (The instruction master port does not connect to the JTAG UART, because the JTAG UART is not a memory device and cannot feed instructions to the Nios II processor.) When building a system, always verify that SOPC Builder's automatic connections are appropriate for your system requirements.

For further details about the JTAG UART, refer to the *JTAG UART Core* chapter in volume 5 of the *Quartus II Handbook*.

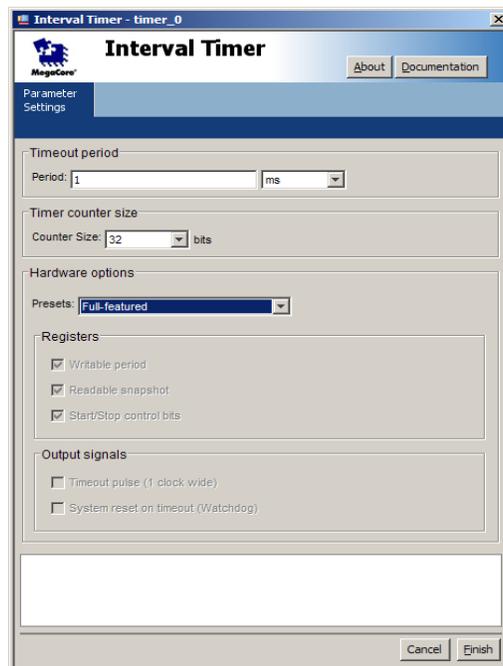
Add the Interval Timer

Most control systems use a timer component to enable precise calculation of time. To provide a periodic system clock tick, the Nios II HAL requires a timer.

Perform the following steps to add the timer:

1. In the list of available components, expand **Peripherals**, expand **Microcontroller Peripherals**, and then click **Interval Timer**.
2. Click **Add**. The Interval Timer MegaWizard interface appears. [Figure 1-10](#) shows the GUI.
3. In the **Presets** list, select **Full-featured**.
4. Do not change any of the other default settings.

Figure 1-10. Interval Timer MegaWizard



5. Click **Finish**. You return to the SOPC Builder **System Contents** tab, and an instance of the interval timer now appears in the table of available components.
6. Right-click the interval timer and click **Rename**.
7. Type `sys_clk_timer` and press Enter.

 For further details about the timer, refer to the *Timer Core* chapter in volume 5 of the *Quartus II Handbook*.

Add the System ID Peripheral

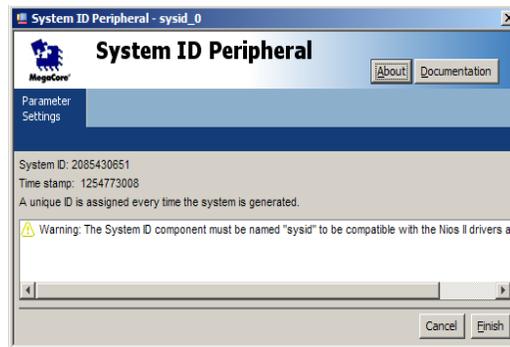
The system ID peripheral safeguards against accidentally downloading software compiled for a different Nios II system. If the system includes the system ID peripheral, the Nios II Software Build Tools for Eclipse can prevent you from downloading programs compiled for a different system.

Perform the following steps to add the system ID peripheral:

1. In the list of available components, expand **Peripherals**, expand **Debug and Performance**, and then click **System ID Peripheral**.

- Click **Add**. The System ID Peripheral MegaWizard interface appears. [Figure 1-11](#) shows the GUI. The system ID peripheral has no user-configurable options.

Figure 1-11. System ID Peripheral MegaWizard



- Click **Finish**. You return to the SOPC Builder **System Contents** tab, and an instance of the system ID peripheral now appears in the table of available components.
- Right-click the system ID peripheral and click **Rename**.
- Type `sysid` and press Enter.

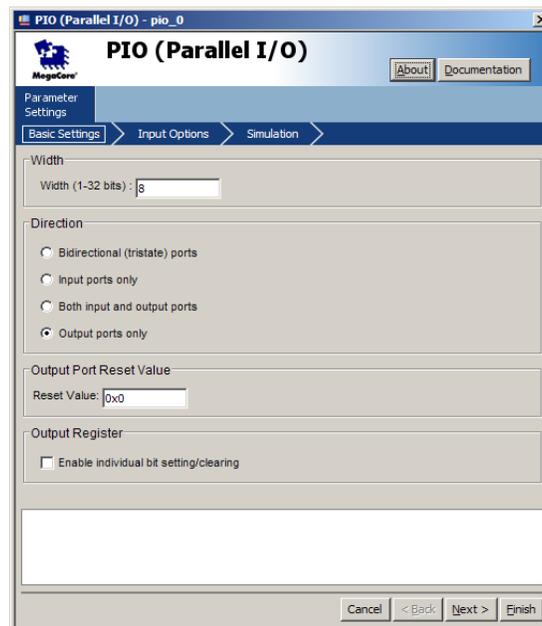
 For further details about the system ID peripheral, refer to the [System ID Core](#) chapter in volume 5 of the *Quartus II Handbook*.

Add the PIO

PIO signals provide an easy method for Nios II processor systems to receive input stimuli and drive output signals. Complex control applications might use hundreds of PIO signals which the Nios II processor can monitor. This design example uses eight PIO signals to drive LEDs on the board.

Perform the following steps to add the PIO. Perform these steps even if your target board doesn't have LEDs.

- In the list of available components, expand **Peripherals**, expand **Microcontroller Peripherals**, and then click **PIO (Parallel I/O)**.
- Click **Add**. The PIO (Parallel I/O) MegaWizard interface appears. [Figure 1-12](#) shows the GUI.
- Do not change the default settings. The MegaWizard interface defaults to an 8-bit output-only PIO, which exactly matches the needs for the design example.

Figure 1-12. PIO MegaWizard

4. Click **Finish**. You return to the SOPC Builder **System Contents** tab, and an instance of the PIO now appears in the table of available components.
5. Right-click the PIO and click **Rename**.
6. Type `led_pio` and press Enter.



For further details about the PIO, refer to the *PIO Core* chapter in volume 5 of the *Quartus II Handbook*.

Specify Base Addresses and Interrupt Request Priorities

At this point, you have added all the necessary hardware components to the system. Now you must specify how the components interact to form a system. In this section, you assign base addresses for each slave component, and assign interrupt request (IRQ) priorities for the JTAG UART and the interval timer.

SOPC Builder provides the **Auto-Assign Base Addresses** command which makes assigning component base addresses easy. For many systems, including this design example, **Auto-Assign Base Addresses** is adequate. However, you can adjust the base addresses to suit your needs. Below are some guidelines for assigning base addresses:

- Nios II processor cores can address a 31-bit address span. You must assign base address between 0x00000000 and 0x7FFFFFFF.
- Nios II programs use symbolic constants to refer to addresses. Do not worry about choosing address values that are easy to remember.
- Address values that differentiate components with only a one-bit address difference produce more efficient hardware. Do not worry about compacting all base addresses into the smallest possible address range, because this can create less efficient hardware.

- SOPC Builder does not attempt to align separate memory components in a contiguous memory range. For example, if you want an on-chip RAM and an off-chip RAM to be addressable as one contiguous memory range, you must explicitly assign base addresses.

SOPC Builder also provides an **Auto-Assign IRQs** command which connects IRQ signals to produce valid hardware results. However, assigning IRQs effectively requires an understanding of how software responds to them. Because SOPC Builder does not deal with software behavior, it cannot make educated guesses about the best IRQ assignment.

The Nios II HAL interprets low IRQ values as higher priority. The timer component must have the highest IRQ priority to maintain the accuracy of the system clock tick.

To assign appropriate base addresses and IRQs, perform the following steps:

1. On the System menu, click **Auto-Assign Base Addresses** to make SOPC Builder assign functional base addresses to each component in the system. The **Base** and **End** values in the table of active components might change, reflecting the addresses that SOPC Builder reassigned.
2. Click the **IRQ** value for the **jtag_uart** component to select it.
3. Type 16 and press Enter to assign a new IRQ value.

Figure 1-13 shows the state of the SOPC Builder **System Contents** tab with the complete system.

Figure 1-13. System Contents Tab with Complete System

The screenshot shows the SOPC Builder System Contents tab. At the top, there is a 'Target' section with 'Device Family' set to 'Cyclone III'. Below it is a 'Clock Settings' table with one entry: 'clk_0' from 'External' source at '50.0' MHz. The main table lists components with columns for 'Use', 'Conne...', 'Module Name', 'Description', 'Clock', 'Base', 'End', and 'IRQ'. The 'jtag_uart' component is selected, and its IRQ value is being edited to 16.

Use	Conne...	Module Name	Description	Clock	Base	End	IRQ
<input checked="" type="checkbox"/>		onchip_mem s1	On-Chip Memory (RAM or ROM) Avalon Memory Mapped Slave	clk_0	0x00004000	0x00007fff	
<input checked="" type="checkbox"/>		cpu	Nios II Processor				
		instruction_master	Avalon Memory Mapped Master	clk_0			
		data_master	Avalon Memory Mapped Master				
		jtag_debug_module	Avalon Memory Mapped Slave		IRQ 0	IRQ 31	
<input checked="" type="checkbox"/>		jtag_uart	JTAG UART				16
		avalon_jtag_slave	Avalon Memory Mapped Slave	clk_0	0x00009030	0x00009037	
<input checked="" type="checkbox"/>		sys_clk_timer s1	Interval Timer				
		sysid	Avalon Memory Mapped Slave	clk_0	0x00009000	0x0000901f	
<input checked="" type="checkbox"/>		sysid control_slave	System ID Peripheral				
		control_slave	Avalon Memory Mapped Slave	clk_0	0x00009038	0x0000903f	
<input checked="" type="checkbox"/>		led_pio s1	PIO (Parallel IO)				
		s1	Avalon Memory Mapped Slave	clk_0	0x00009020	0x0000902f	

Generate the SOPC Builder System

You are now ready to generate the SOPC Builder system. Perform the following steps:

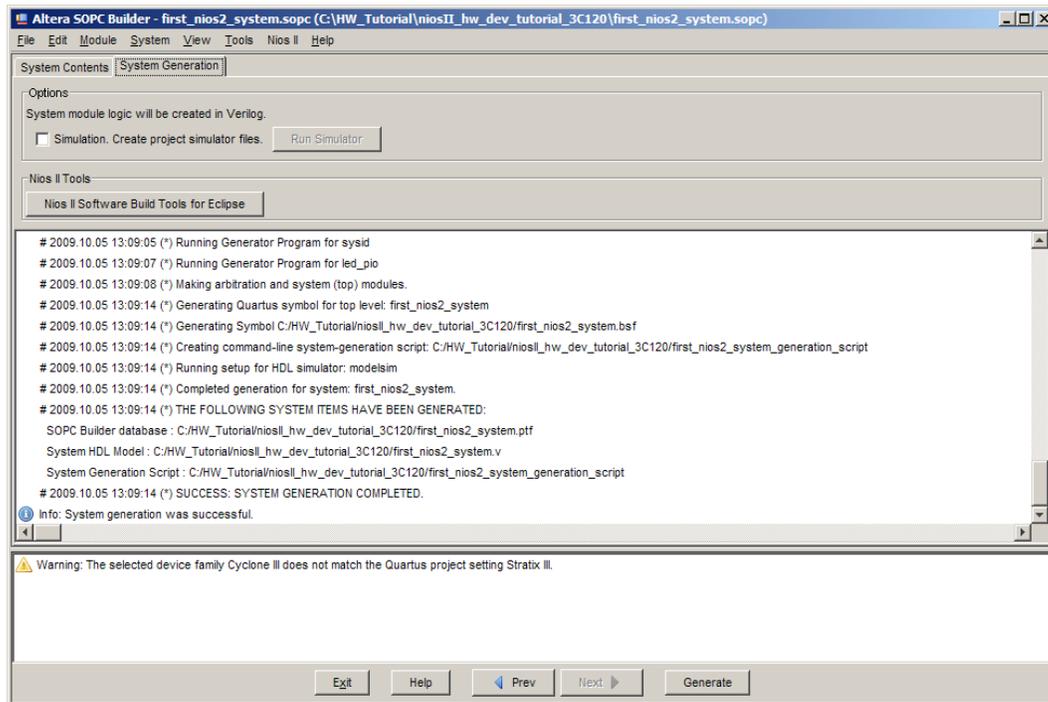
1. Click the **System Generation** tab.
2. Turn off **Simulation. Create simulator project files.**, which saves time because this tutorial does not cover the hardware simulation flow.
3. Click **Generate**. The **Save Changes** dialog box appears, prompting you to save your design.

- Click **Save**. The system generation process begins.

The generation process can take several minutes. When it completes, the **System Generation** tab displays a message "Info: System generation was successful."

Figure 1-14 shows the successful system generation.

Figure 1-14. Successful System Generation



- Click **Exit** to return to the Quartus II software.

Congratulations! You have finished creating the Nios II processor system. You are ready to integrate the system into the Quartus II hardware project and use the Nios II Software Build Tools for Eclipse to develop software.

 For further details about generating systems with SOPC Builder, refer to *Volume 4: SOPC Builder* of the *Quartus II Handbook*. For details about hardware simulation for Nios II systems, refer to the *AN351: Simulating Nios II Embedded Processor Designs*.

Integrate the SOPC Builder System into the Quartus II Project

In this section you perform the following steps to complete the hardware design:

- Instantiate the SOPC Builder system module in the Quartus II project.
- Assign FPGA device and pin locations.
- Compile the Quartus II project.
- Verify timing.

For further information about using the Quartus II software, refer to *Introduction to the Quartus II Software*, the *Quartus II Handbook*, and the *Quartus II Software Interactive Tutorial* in the **Training** section of the Altera website.

Instantiate the SOPC Builder System Module in the Quartus II Project

SOPC Builder outputs a design entity called the system module. The tutorial design example uses the block diagram method of design entry, so you instantiate a system module symbol `first_nios2_system` into the `.bdf`.

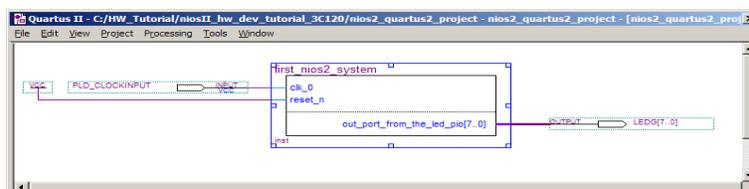
How you instantiate the system module depends on the design entry method of the overall Quartus II project. For example, if you were using Verilog HDL for design entry, you would instantiate the Verilog module `first_nios2_system` defined in the file `first_nios2_system.v`.

To instantiate the system module in the `.bdf`, perform the following steps:

1. Double click in the empty space between the input and output pins. The **Symbol** dialog box appears.
2. Under **Libraries**, expand **Project**.
3. Click `first_nios2_system`. The **Symbol** dialog box displays the `first_nios2_system` symbol.
4. Click **OK**. You return to the `.bdf` schematic. The `first_nios2_system` symbol tracks with your mouse pointer.
5. Position the symbol so the inputs on the symbol align with the wires on the left side of the schematic.
6. Click the left mouse button to drop the symbol in place.
7. If your target board has LEDs that the Nios II system can drive, click and drag `LEDG[7..0]` and connect it with the port `out_port_from_the_led_pio[7..0]` on the `first_nios2_system` symbol. This action connects the `LEDG[7..0]` output pins to the `first_nios2_system`.

Figure 1-15 shows the completed Board Design File schematic using the LED pins.

Figure 1-15. Completed Board Design File Schematic



8. If you are targeting a custom board that does not have LEDs, you must delete the `LEDG[7..0]` pins. To delete the pins, perform the following steps:
 - a. Click the output symbol `LEDG[7..0]` to select it.
 - b. On your keyboard, press **Delete**.
9. To save the completed `.bdf`, click **Save** on the File menu.

Assign FPGA Device and Pin Locations

In this section, you assign a specific target device and then assign FPGA pin locations to match the pinouts of your board.

1. You must know the pin layout for the board to complete this section. You also must know other requirements for using the board, which are beyond the scope of this document. Refer to the documentation for your board.

 For Altera development board reference manuals, refer to the [Literature: Development Kits](#) page of the Altera website.

To assign the device, perform the following steps:

1. On the Assignments menu, click **Device**. The **Settings** dialog box appears.
2. In the **Family** list, select the FPGA family that matches your board.

 If prompted to remove location assignments, do so.

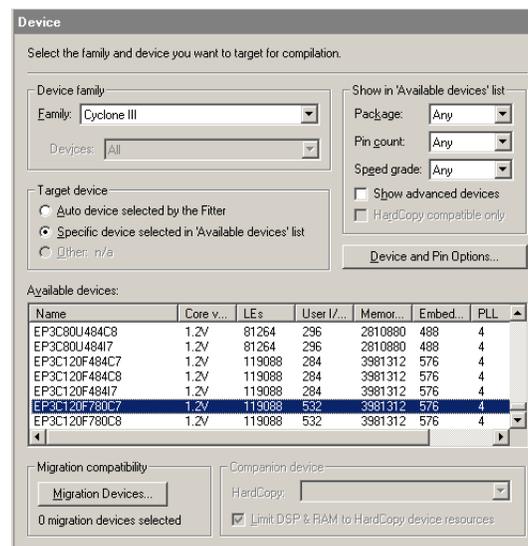
3. Under **Target Device**, select **Specific device selected in 'Available devices' list**.
4. Under **Available devices**, select the exact device that matches your board.

 If prompted to remove location assignments, do so.

5. Click **OK** to accept the device assignment.

[Figure 1-16](#) shows an example of the **Device** page of the **Settings** dialog box.

Figure 1-16. Assigning a Device in the Quartus II Settings Dialog Box



To assign the FPGA pin locations, perform the following steps:

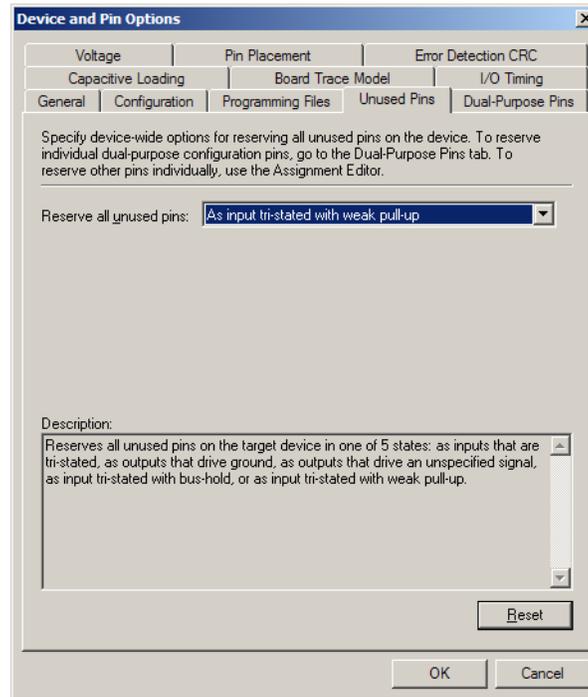
1. On the Processing menu, point to **Start**, and click **Start Analysis & Elaboration** to prepare for assigning pin locations. A confirmation message appears when analysis and elaboration completes.

2. Click **OK**.
3. On the Assignments menu, click **Pins**. The Quartus II Pin Planner appears.
4. In the **Node Name** column, locate **PLD_CLOCKINPUT**.
5. In the **PLD_CLOCKINPUT** row, double-click in the **Location** cell. A list of available pin locations appears. [Figure 1-17](#) shows the GUI.

Figure 1-17. Assigning Pins with the Quartus II Pin Planner

	Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved
1	LEDG[7]	Output				2.5 V (default)	
2	LEDG[6]	Output				2.5 V (default)	
3	LEDG[5]	Output				2.5 V (default)	
4	LEDG[4]	Output				2.5 V (default)	
5	LEDG[3]	Output				2.5 V (default)	
6	LEDG[2]	Output				2.5 V (default)	
7	LEDG[1]	Output				2.5 V (default)	
8	LEDG[0]	Output				2.5 V (default)	
9	PLD_CLOCKINPUT	Input	PIN_AH15	4	B4_N2	1.8 V	
10	<<new node>>						

6. Select the appropriate FPGA pin that connects to the oscillator on the board.
 -  If your design fails to work, recheck your board documentation for this step first.
7. In the **PLD_CLOCKINPUT** row, double-click in the **I/O Standard** cell. A list of available I/O standards appears.
8. Select the appropriate I/O standard that connects to the oscillator on the board.
9. If you connected the LED pins in the board design schematic, repeat steps 4 to 8 for each of the LED output pins (**LEDG[0]**, **LEDG[1]**, **LEDG[2]**, **LEDG[3]**, **LEDG[4]**, **LEDG[5]**, **LEDG[6]**, **LEDG[7]**) to assign appropriate pin locations.
10. On the File menu, click **Close** to save the assignments.
11. On the Assignments menu, click **Device**. The **Settings** dialog box appears.
12. Click **Device and Pin Options**. The **Device and Pin Options** dialog box appears.
13. Click the **Unused Pins** tab. [Figure 1-18](#) shows the GUI.

Figure 1-18. The Unused Pins Tab of the Device and Pin Options Dialog Box

14. In the **Reserve all unused pins** list, select **As input tri-stated with weak pull-up**. With this setting, all unused I/O pins on the FPGA enter a high-impedance state after power-up.



Unused pins are set as input tri-stated with weak pull-up to remove contention which might damage the board. Depending on the board, you might have to make more assignments for the project to function correctly. You can damage the board if you fail to account for the board design. Consult with the maker of the board for specific contention information.

15. Click **OK** to close the **Device and Pin Options** dialog box.
16. Click **OK** to close the **Settings** dialog box.



For further details about making assignments in the Quartus II software, refer to the *Volume 2: Design Implementation and Optimization* of the *Quartus II Handbook*.

Compile the Quartus II Project and Verify Timing

At this point you are ready to compile the Quartus II project and verify that the resulting design meets timing requirements.

You must compile the hardware design to create a **.sof** that you can download to the board. After the compilation completes, you must analyze the timing performance of the FPGA design to verify that the design will work in hardware.

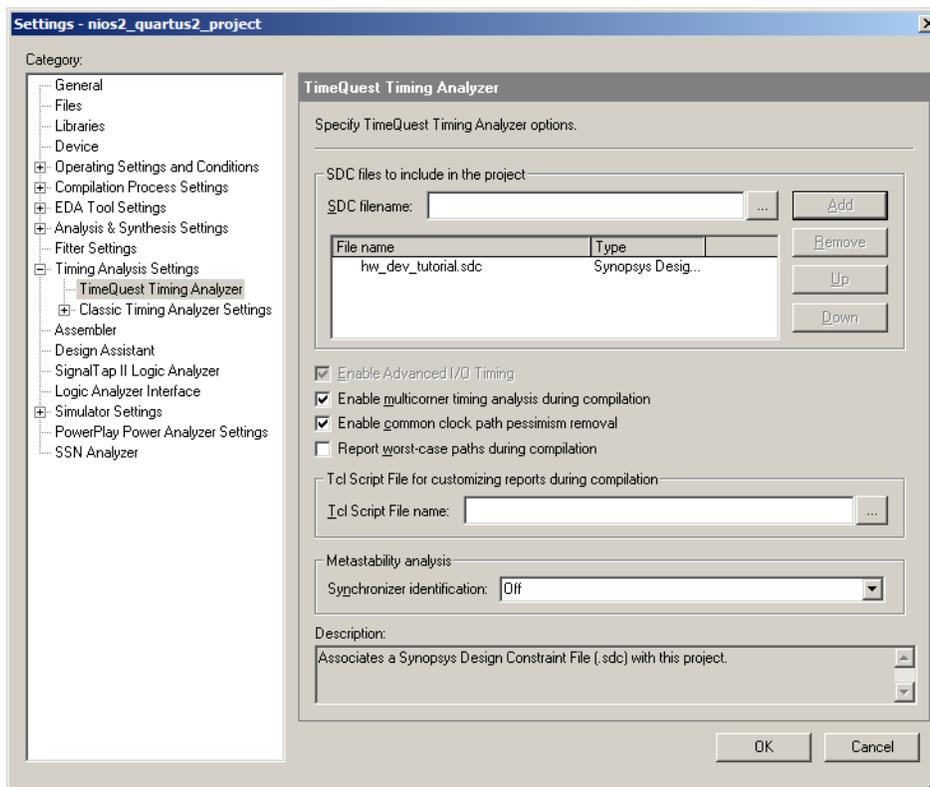
To ensure the design meets timing, perform the following steps:

1. On the File menu, click **Open**.

2. In the **Files of type** list, select **Script Files (*.tcl, *.sdc, *.qip)**.
3. Select **hw_dev_tutorial.sdc** and click **Open**. The file opens in the Text Editor.
4. Locate the following `create_clock` command:


```
create_clock -name sopc_clk -period 20 [get_ports PLD_CLOCKINPUT]
```
5. Change the period setting from 20 to the clock period (1/frequency) in nanoseconds of the oscillator driving the clock pin.
6. On the File menu, click **Save**.
7. On the Assignments menu, click **Settings**. The **Settings** dialog box appears.
8. Under **Category**, select **Timing Analysis Settings**.
9. Turn on **Use TimeQuest Timing Analyzer during compilation**.
10. Under **Category**, expand **Timing Analysis Settings**, and click **TimeQuest Timing Analyzer**. [Figure 1-19](#) shows the GUI.

Figure 1-19. TimeQuest Timing Analyzer Settings



11. Next to **SDC Filename**, click the browse (...) button.
12. Select **hw_dev_tutorial.sdc** and click **Open** to select the file.
13. Click **Add** to include **hw_dev_tutorial.sdc** in the project.
14. Turn on **Enable multicorner timing analysis during compilation**.
15. Click **OK**.

To compile the Quartus II project, perform the following steps:

1. On the Processing menu, click **Start Compilation**.
2. The Tasks window displays progress. The compilation process can take several minutes. When compilation completes, a dialog box displays the message "Full compilation was successful."
3. Click **OK**. The Quartus II software displays the Compilation Report.
4. Expand the **TimeQuest Timing Analyzer** category in the Compilation Report.
5. Click **Multicorner Timing Analysis Summary**.
6. Verify that the **Worst-case Slack** values are positive numbers for **Setup**, **Hold**, **Recovery** and **Removal**. If any of these values are negative, the design might not operate properly in hardware. To meet timing, adjust Quartus II assignments to optimize fitting, or reduce the oscillator frequency driving the FPGA.

 For further details about meeting timing requirements in the Quartus II software, refer to the *Volume 1: Design and Synthesis* of the *Quartus II Handbook*.

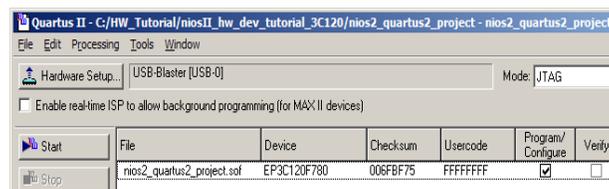
Congratulations! You have finished integrating the Nios II system into the Quartus II project. You are ready to download the **.sof** to the target board.

Download Hardware Design to Target FPGA

In this section you download the **.sof** to the target board. Perform the following steps:

1. Connect the board to the host computer with the download cable, and apply power to the board.
2. On the Tools menu in the Quartus II software, click **Programmer**. The Quartus II Programmer appears and automatically displays the appropriate configuration file (**nios2_quartus2_project.sof**). [Figure 1–20](#) shows a portion of the GUI.

Figure 1–20. Quartus II Programmer



3. Click **Hardware Setup** in the upper left corner of the Quartus II Programmer to verify your download cable settings. The **Hardware Setup** dialog box appears.
4. Select the appropriate download cable in the **Currently selected hardware** list. If the appropriate download cable does not appear in the list, you must first install drivers for the cable.

 For information about download cables and drivers, refer to the [Download Cables](#) page of the Altera website.

5. Click **Close**.

6. Turn on **Program/Configure** for `nios2_quartus2_project.sof`.
7. Click **Start**. The **Progress** meter sweeps to 100% as the Quartus II software configures the FPGA.

At this point, the Nios II system is configured and running in the FPGA, but it does not yet have a program in memory to execute.

Develop Software Using the Nios II Software Build Tools for Eclipse

In this section, you start the Nios II Software Build Tools for Eclipse and compile a simple C language program. This section presents only the most basic software development steps to demonstrate software running on the hardware system you created in previous sections.

 For a complete tutorial on using the Nios II Software Build Tools for Eclipse to develop programs, refer to the *Getting Started with the Graphical User Interface* chapter of the *Nios II Software Developer's Handbook*.

In this section, you perform the following actions:

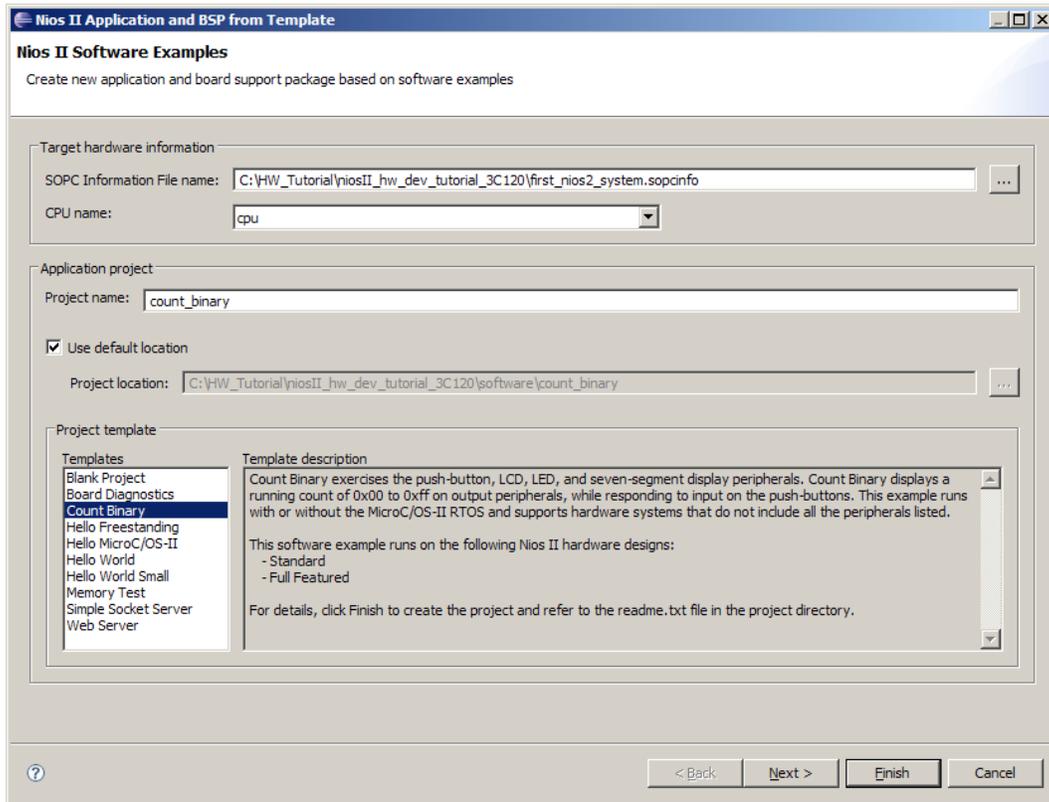
- Create new Nios II C/C++ application and board support package (BSP) projects.
- Compile the projects.

To perform this section, you must have the `.sopcinfo` file you created in “[Define the System in SOPC Builder](#)” on page 1-11.

Create a New Nios II Application and BSP from Template

In this section you create new Nios II C/C++ application and BSP projects. Perform the following steps:

1. Start the Nios II Software Build Tools for Eclipse. On Windows computers, click **Start**, point to **Programs, Altera, Nios II EDS <version>**, and then click **Nios II <version> Software Build Tools for Eclipse**. On Linux computers, run the executable file `<Nios II EDS install path>/bin/eclipse-nios2`.
2. If the **Workspace Launcher** dialog box appears, click **OK** to accept the default workspace location.
3. On the Windows menu, point to **Open Perspective**, and then either click **Nios II**, or click **Other** and then click **Nios II** to ensure you are using the Nios II perspective.
4. On the File menu, point to **New**, and then click **Nios II Application and BSP from Template**. The Nios II Application and BSP from Template wizard appears. [Figure 1-21](#) shows the GUI.

Figure 1–21. Nios II Application and BSP from Template Wizard

5. Under **Target hardware information**, next to **SOPC Information File name**, browse to the *<design files directory>*.
6. Select **first_nios2_system.sopcinfo** and click **Open**. You return to the Nios II Application and BSP from Template wizard showing current information for the **SOPC Information File name** and **CPU name** fields.
7. In the **Project name** box, type `count_binary`.
8. In the **Templates** list, select **Count Binary**.
9. Click **Finish**.

The Nios II Software Build Tools for Eclipse creates and displays the following new projects in the Project Explorer view, typically on the left side of the workbench:

- **count_binary**—Your C/C++ application project
- **count_binary_bsp**—A board support package that encapsulates the details of the Nios II system hardware

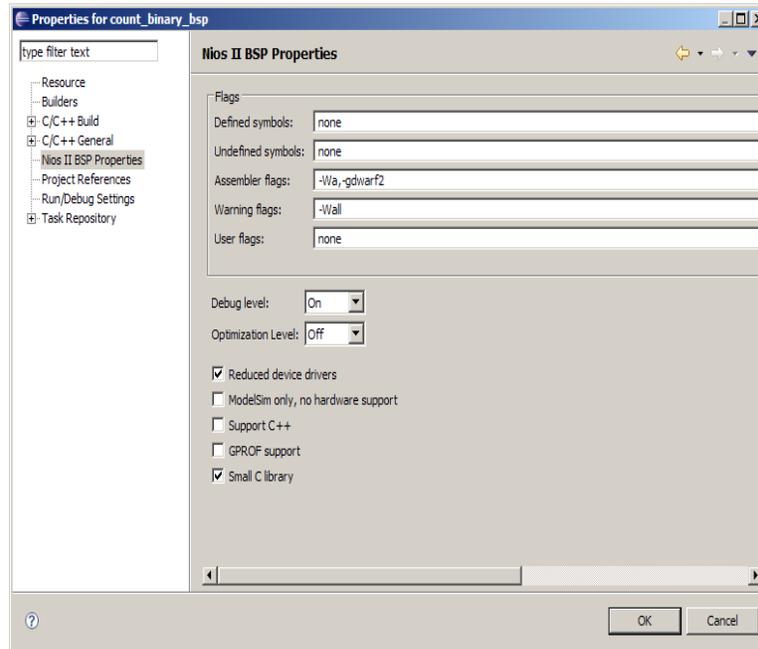
Compile the Project

In this section you compile the project to produce an executable software image. For the tutorial design example, you must first adjust the project settings to minimize the memory footprint of the software, because your Nios II hardware system contains only 20 KB of memory.

Perform the following steps:

1. In the Project Explorer view, right-click **count_binary_bsp** and click **Properties**. The **Properties for count_binary_bsp** dialog box opens.
2. Click the **Nios II BSP Properties** page. The **Nios II BSP Properties** page contains basic software build settings. [Figure 1-22](#) shows the GUI.

Figure 1-22. System Library Properties



 Though not needed for this tutorial, note the **BSP Editor** button in the lower right corner of the dialog box. You use the Nios II BSP Editor to access advanced BSP settings.

3. Adjust the following settings to reduce the size of the compiled executable:
 - a. Turn on **Reduced device drivers**.
 - b. Turn off **ModelSim only, no hardware support**.
 - c. Turn off **Support C++**.
 - d. Turn off **GPROF support**.
 - e. Turn on **Small C library**.

 For further details about BSPs, refer to the *Nios II Software Developer's Handbook*.

4. Click **OK**. The BSP regenerates, the **Properties** dialog box closes, and you return to the workbench.
5. In the Project Explorer view, right-click the **count_binary** project and click **Build Project**.

The **Build Project** dialog box appears, and the Nios II Software Build Tools for Eclipse begins compiling the project. When compilation completes, a "build completed" message appears in the Console view.

Run the Program on Target Hardware

In this section you download the program to target hardware and run it. To download the software executable to the target board, perform the following steps:

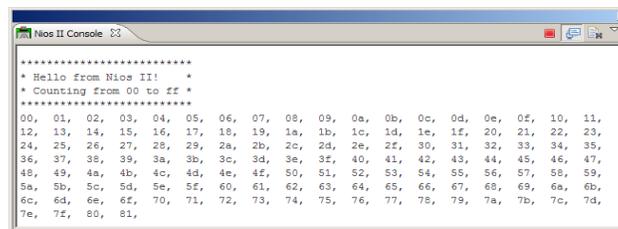
1. Right-click the **count_binary** project, point to **Run As**, and then click **Nios II Hardware**. The Nios II Software Build Tools for Eclipse downloads the program to the FPGA on the target board and the program starts running.

 If the **Run Configurations** dialog box appears, verify that **Project name** and **ELF file name** contain relevant data, then click **Run**.

When the target hardware starts running the program, the Nios II Console view displays character I/O output. [Figure 1-23](#) shows the output. If you connected LEDs to the Nios II system in [“Integrate the SOPC Builder System into the Quartus II Project”](#) on page 1-21, then the LEDs blink in a binary counting pattern.

2. Click the **Terminate** icon (the red square) on the toolbar of the Nios II Console view to terminate the run session. When you click the **Terminate** icon, the Nios II Software Build Tools for Eclipse disconnects from the target hardware.

Figure 1-23. Console View Displaying Nios II Hardware Output



```

*****
* Hello from Nios II! *
* Counting from 00 to ff *
*****
00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 0a, 0b, 0c, 0d, 0e, 0f, 10, 11,
12, 13, 14, 15, 16, 17, 18, 19, 1a, 1b, 1c, 1d, 1e, 1f, 20, 21, 22, 23,
24, 25, 26, 27, 28, 29, 2a, 2b, 2c, 2d, 2e, 2f, 30, 31, 32, 33, 34, 35,
36, 37, 38, 39, 3a, 3b, 3c, 3d, 3e, 3f, 40, 41, 42, 43, 44, 45, 46, 47,
48, 49, 4a, 4b, 4c, 4d, 4e, 4f, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59,
5a, 5b, 5c, 5d, 5e, 5f, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 6a, 6b,
6c, 6d, 6e, 6f, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 7a, 7b, 7c, 7d,
7e, 7f, 80, 81,

```

You can make edits to the **count_binary.c** program in the Nios II Software Build Tools for Eclipse and repeat these two steps to witness your changes executing on the target board. If you rerun the program, buffered characters from the previous run session might display in the Console view before the program begins executing.

 For information on running and debugging programs on target hardware, refer to the tutorial in the [Getting Started with the Graphical User Interface](#) chapter of the *Nios II Software Developer's Handbook*.

Taking the Next Step

Congratulations! You have completed building a Nios II hardware system and running software on it. Through this tutorial, you have familiarized yourself with the steps for developing a Nios II system:

- Analyzing system requirements
- Defining and generating Nios II system hardware in SOPC Builder

- Integrating the SOPC Builder system into a Quartus II project
- Compiling the Quartus II project and verifying timing
- Creating a new project in the Nios II Software Build Tools for Eclipse
- Compiling the project
- Running the software on target hardware

The following documents provide next steps to further your understanding of the Nios II processor:

- *Nios II Software Developer's Handbook*—This handbook provides complete reference for developing software for the Nios II processor.
- The software development tutorial in the *Getting Started with the Graphical User Interface* chapter of the *Nios II Software Developer's Handbook*—This tutorial teaches in detail how to use the Nios II Software Build Tools for Eclipse to develop, run, and debug new Nios II C/C++ application projects.
- *Nios II Processor Reference Handbook*—This handbook provides complete reference for the Nios II processor hardware.
- *Volume 4: SOPC Builder* of the *Quartus II Handbook* – This volume provides complete reference on using SOPC Builder, including topics such as building memory subsystems and creating custom components.
- *Volume 5: Embedded Peripherals* of the *Quartus II Handbook* – This handbook contains details about the components provided free as part of the Nios II EDS.

For a complete list of all documents available for the Nios II processor, refer to the [Literature: Nios II Processor](#) page of the Altera website.

Revision History

The following table shows the revision history for this tutorial.

 Refer to the [Nios II Embedded Design Suite Release Notes](#) page of the Altera website for the latest features, enhancements, and known issues in the current release.

Date	Version	Changes Made
December 2009	3.0	Revised entire document to use Nios II Software Build Tools for Eclipse.
October 2007	2.5	<ul style="list-style-type: none"> ■ Added altera.components project information. ■ Minor text changes.
May 2007	2.4	<ul style="list-style-type: none"> ■ Updated to describe new SOPC Builder MegaWizard design flow. ■ Added OpenCore Plus information.
March 2007	2.3	Maintenance release for v7.0 software.
November 2006	2.2	Minor text changes.
May 2006	2.1	Revised and simplified the tutorial flow.
May 2005	2.0	Revised the introductory information.
December 2004	1.1	Updated for the Nios II 1.1 release.
September 2004	1.01	Updated for the Nios II 1.01 release.
May 2004	1.0	Initial release.

How to Contact Altera

For the most up-to-date information about Altera products, refer to the following table.

Contact <i>(Note 1)</i>	Contact Method	Address
Technical support	Website	www.altera.com/support
Technical training	Website	www.altera.com/training
	Email	custrain@altera.com
Product literature	Website	www.altera.com/literature
Non-technical support (General) (Software Licensing)	Email	nacomp@altera.com
	Email	authorization@altera.com

Note to Table:

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

Typographic Conventions

This document uses the typographic conventions shown in the following table.

Visual Cue	Meaning
Bold Type with Initial Capital Letters	Indicates command names, dialog box titles, dialog box options, and other GUI labels. For example, Save As dialog box.
bold type	Indicates directory names, project names, disk drive names, file names, file name extensions, and software utility names. For example, qdesigns directory, d: drive, and chiptrip.gdf file.
<i>Italic Type with Initial Capital Letters</i>	Indicates document titles. For example, <i>AN 519: Stratix IV Design Guidelines</i> .
<i>Italic type</i>	Indicates variables. For example, $n + 1$. Variable names are enclosed in angle brackets (<>). For example, <file name> and <project name>.pof file.
Initial Capital Letters	Indicates keyboard keys and menu names. For example, Delete key and the Options menu.
“Subheading Title”	Quotation marks indicate references to sections within a document and titles of Quartus II Help topics. For example, “Typographic Conventions.”
Courier type	Indicates signal, port, register, bit, block, and primitive names. For example, data1, tdi, and input. Active-low signals are denoted by suffix n. For example, resetn. Indicates command line commands and anything that must be typed exactly as it appears. For example, c:\qdesigns\tutorial\chiptrip.gdf. Also indicates sections of an actual file, such as a Report File, references to parts of files (for example, the AHDL keyword SUBDESIGN), and logic function names (for example, TRI).
1., 2., 3., and a., b., c., and so on.	Numbered steps indicate a list of items when the sequence of the items is important, such as the steps listed in a procedure.
■ ■	Bullets indicate a list of items when the sequence of the items is not important.
	The hand points to information that requires special attention.
	A caution calls attention to a condition or possible situation that can damage or destroy the product or your work.
	A warning calls attention to a condition or possible situation that can cause you injury.
	The angled arrow instructs you to press Enter.
	The feet direct you to more information about a particular topic.