



Table of Contents

Introduction	2
Example Design.....	2
Software & Hardware Requirements	3
Nios II System Development Flow	4
Analyzing System Requirements	6
Defining & Generating the System in SOPC Builder	6
Quartus II Hardware Development Tasks.....	7
Nios II IDE Software Development Tasks.....	7
Running & Debugging Software on the Target Board.....	8
Varying the Development Flow.....	8
Refining the Software and Hardware	8
Iteratively Creating a Nios II System.....	8
Verifying the System with Hardware Simulation Tools	9
Creating the Example Design.....	9
Install the Design Files	9
Analyze System Requirements	10
Start the Quartus II Software and Open the Tutorial Example Design Project	10
Start a New SOPC Builder System	11
Define the System in SOPC Builder	12
Specify Target FPGA & Clock Settings	13
Add the Nios II Processor Core	13
Add the On-Chip Memory	15
Add the JTAG UART Peripheral.....	16
Add the Timer Peripheral.....	17
Add the System ID Peripheral	18
Add the PIO	19
Specify Base Addresses and Interrupt Request Priorities	20
Specify More Nios II Settings.....	22
Generate the SOPC Builder System.....	22
Integrate the SOPC Builder System into Quartus II Project	23
Instantiate the SOPC Builder System Module in the Quartus II Project.	24
Assign FPGA pins	25
Compile the Quartus II Project and Verify Timing.....	27
Download Hardware Design to Target FPGA	28
Develop Software Using the Nios II IDE	29
Create a New C/C++ Application Project.....	29
Compile the Project	30
Run the Program	32
Run the Program on Target Hardware	32
Run the Program on the ISS.....	32
Taking the Next Step.....	33

Introduction

Thank you for evaluating the Nios II processor! This tutorial introduces you to the system development flow for the Nios II processor. This 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. In this tutorial you will build a Nios II hardware system and create a software program to run on the Nios II system.

Building embedded systems in FPGAs is a broad subject, involving 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, see the *Software Development Tutorial* available from the Nios II IDE help system.

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.

Example Design

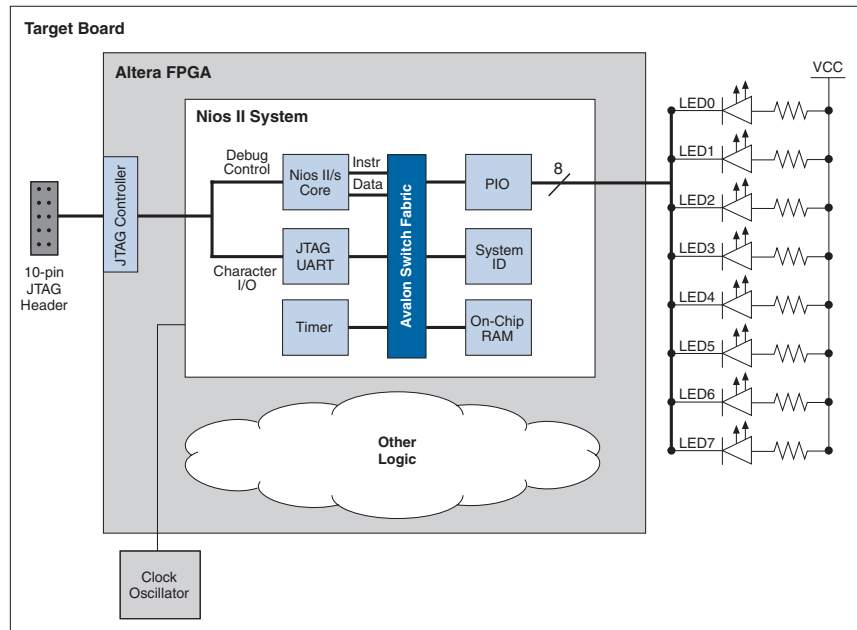
The example design you build in this tutorial demonstrates a small Nios II system for control applications, which can monitor input stimuli and respond by turning on or off output signals. This Nios II system can also communicate with a host computer, allowing the host computer to control logic inside the FPGA.

The example Nios II system contains the following:

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

Figure 1 is a block diagram showing the relationship between the host computer, the target board, the FPGA, and the Nios II system.

Figure 1. Tutorial Example Design



As shown in Figure 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 example design in this tutorial does not include other logic in the FPGA.

Software & Hardware Requirements

This tutorial requires you to have the following software:

- Altera Quartus II software version 5.0 or later – The software must be installed on a Windows or Linux computer that meets the Quartus II minimum requirements.
- Nios II development tools version 5.0 or later
- Design files for the example design – A hyperlink to the design files appears next to this document on the Nios II literature page. Visit www.altera.com/literature/lit-nio2.jsp.

You can build the example design in this tutorial whether you own a development board or not. This tutorial allows you to choose from the following target board options:

- **No board** – If you do not have a target board, you can still use the tutorial, but you will not witness the example design running on hardware. Instead, you will simulate software running on the Nios II instruction set simulator (ISS).
- **Nios development board** – If you have an Altera Nios II development kit, use the board included in the kit. In this case, you also must have the DC power supply and download cable provided with the kit, such as the USB-Blaster™ cable. The following Altera kits are supported:
 - Nios II Development Kit, Stratix® II Edition
 - Nios II Development Kit, Stratix Edition
 - Nios II Development Kit, Stratix Professional Edition

- Nios II Development Kit, Cyclone™ II Edition
- Nios II Development Kit, Cyclone Edition
- Nios II Evaluation Kit



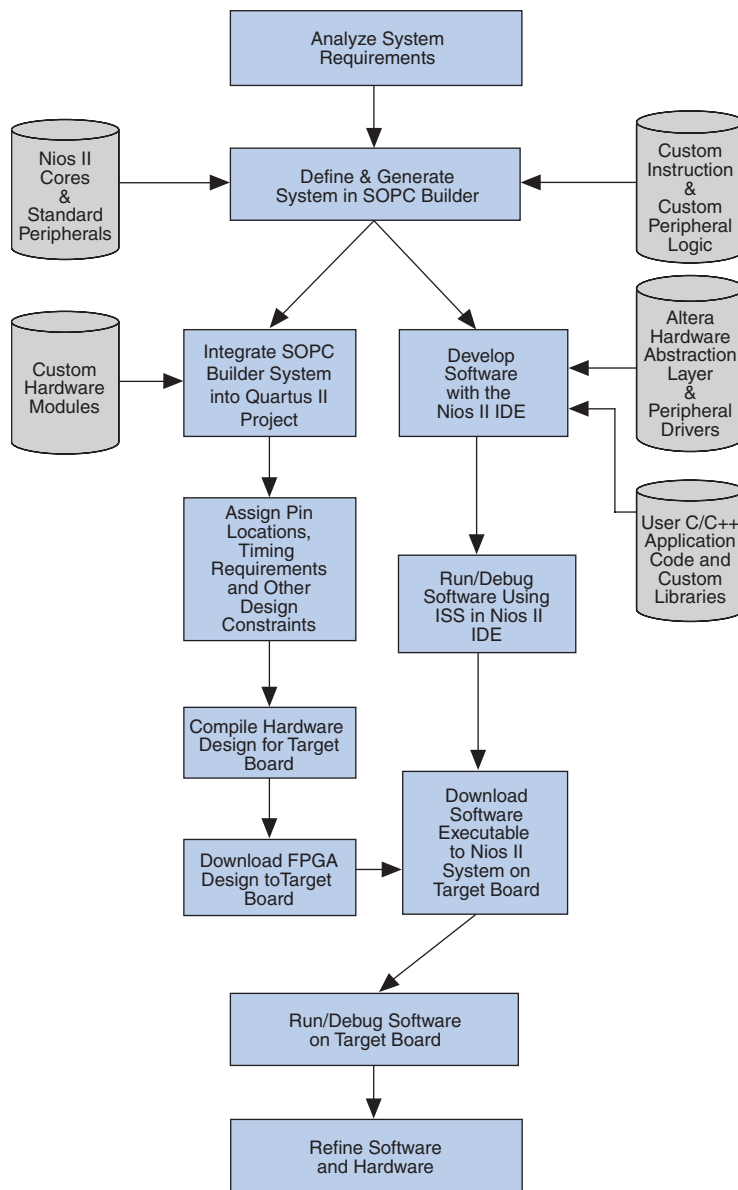
For information on Nios development kits, visit www.altera.com/devkits.

- **Custom board** – You can use this tutorial with any board that meets the following requirements:
 - The board must have an Altera FPGA.
 - The FPGA must meet the following density requirements, depending on the device family:
 - Stratix II EP2S15 device or larger
 - Stratix EP1S10 device or larger
 - Cyclone II EP2C20 device or larger
 - Cyclone EP1C12 device or larger
 - 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.
 - The board must have a 10-pin header connected to the dedicated JTAG pins on the FPGA to provide a communication link to the Nios II system.
 - FPGA I/O pins can optionally connect to 8 (or fewer) LEDs to provide a visual indicator of processor activity.
 - You must have an Altera USB Blaster download cable, revision B or higher. Prior cables might exhibit communication errors when connecting to the Nios II processor. Revision B cables (or later) have a clearly-marked revision label; earlier cables do not.

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 2 shows the Nios II system development flow.

Figure 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. For further details on the software development process, Altera recommends that you read the *Software Development Tutorial* available from the Nios II IDE help system after you complete this tutorial.



The *Software Development Tutorial* and complete IDE reference are included in the Nios II IDE help system. To open the Nios II IDE help system, click **Help Contents** on the Help menu. To see the tutorials, click **Nios II IDE Help** in the **Contents** pane, and then click **Tutorials**.

Analyzing System Requirements

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

- 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 peripherals does the design require? How many of each kind?
- Which real-time operating system (RTOS) to use, if any?
- Where can hardware acceleration logic dramatically improve system performance? For example:
 - Could adding a DMA peripheral eliminate wasted processor cycles copying data?
 - Could a custom instruction replace the critical loop of a DSP algorithm?

Answers to these questions involve both the hardware and software teams.

Defining & Generating the System in SOPC Builder

After analyzing the system hardware requirements, you use the SOPC Builder tool which is included in the Altera Quartus II software. Using SOPC Builder you specify the Nios II processor core(s), memory, and other peripherals 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 peripherals provided with the Nios II development tools. 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 peripheral which offloads tasks from the CPU. This tutorial covers adding standard processor and peripheral cores, but does not cover adding custom logic to the system.

The primary outputs of SOPC Builder are the following:

- SOPC Builder system file (**.ptf**) – This file stores the hardware contents of the SOPC Builder system. The Nios II IDE requires the **.ptf** file to compile software for the target hardware.
- Hardware description language (HDL) files – These files are the hardware design files which describe the SOPC Builder system. The Quartus II software uses the HDL files to compile the overall FPGA design.



For further details on the Nios II processor, see the *Nios II Processor Reference Handbook*. For further details on SOPC Builder and developing custom peripherals, see the *Quartus II Handbook Volume 1: Design & Synthesis*. For further details on custom instructions, see the *Nios II Custom Instruction User Guide*.

Quartus II Hardware Development Tasks

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

As shown in Figure 1, 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 3rd 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 an FPGA configuration file (**.sof**).

You download the FPGA configuration file 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 on using the Quartus II software, see the *Quartus II Tutorial* in the Quartus II help system, and the *Introduction to Quartus II Manual* and the *Quartus II Handbook*, available at www.altera.com.

Nios II IDE Software Development Tasks

Using the Nios II IDE, 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 IDE. Altera provides peripheral 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 IDE projects.

If you do not have a target board for software development, you can run and debug your code with the Nios II instruction set simulator (ISS). The ISS simulates the processor, memory, and `stdin/stdout/stderr` streams, which allows you to verify program flow and algorithm correctness. As soon as you have a target board with an Altera FPGA configured with the Nios II system, you can download your software to the board using an Altera download cable, such as the USB-Blaster.

To start a new C/C++ application project, the Nios II IDE requires the SOPC Builder system file (**.ptf**). Software designers might also require the FPGA configuration file (**.sof**) to configure the FPGA before running and debugging the application project on target hardware.

The IDE can produce several outputs, listed below. Not all projects require all of these outputs.

- **system.h** file – **system.h** defines symbols for referencing the hardware in the system. The IDE automatically creates this file when you create a new project.
- Software executable (**.elf**) – An **.elf** file is the result of compiling a C/C++ application project, which you can download directly to the Nios II processor.

- Memory initialization files (**.hex**) – Some on-chip memories can power up with predefined memory contents. The IDE generates initialization files for on-chip memories that support initialization of contents.
- Flash programming data – The IDE 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 only on downloading the **.elf** file directly to the Nios II system.



For complete details on developing software for the Nios II processor, see the *Nios II Software Developer's Handbook* and the Nios II IDE help system.

Running & Debugging Software on the Target Board

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



For details on running and debugging Nios II programs, see the *Software Development Tutorial* available from the Nios II IDE help system.

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. Alternately, 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 peripheral 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.



The Nios II development tools provide several working Nios II reference designs, which you can use as a starting point for your own designs. After installing the Nios II development tools, see the directory `<Nios II kit path>/examples/verilog` or the directory `<Nios II kit path>/examples/vhdl`.

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 for certain cases, including the following:

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

A hardware simulation step is not shown in Figure 2. If you are building a Nios II system based on the standard peripherals provided with the Nios II Embedded Design Suite, the easiest way to verify functionality is to download the hardware and software directly to a development board.



For details on performing hardware simulation for Nios II system, see *AN351: Simulating Nios II Embedded Processor Designs*.

Creating the Example Design

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

1. Install the Design Files.
2. Analyze System Requirements (see page 10).
3. Start the Quartus II Software and Open the Tutorial Example Design Project (see page 10).
4. Start a New SOPC Builder System (see page 12).
5. Define the System in SOPC Builder (see page 12).
6. Integrate the SOPC Builder System into Quartus II Project (see page 23).
7. Download Hardware Design to Target FPGA (see page 28).
8. Develop Software Using the Nios II IDE (see page 29).
9. Run the Program (see page 32).

Install the Design Files

Before you proceed, you must install the Quartus II software and the Nios II development tools. 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 are associated with the link to this document on the Nios II literature page at www.altera.com/literature/lit-nio2.jsp.

Perform the following steps to set up the design environment:

1. Locate the zipped design files on the Altera web site. A different set of design files exists for each Altera Nios development board.
2. Download the design files by performing one of the following steps:

- a. If you have a Nios development board, download the files that match your development board.
 - b. If you have a custom board, download the files that most closely match your board. For example, if your board has a Stratix II device, download the Stratix II design files (**niosII_stratixII_2s60_es.zip**).
 - c. If you do not have a board, you can use any of the design files.
3. 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 example design. The goals for the design are the following:

- 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 for the reader.
- 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 peripherals:
 - Nios II/s core with 2 Kbytes of instruction cache
 - 20 Kbytes of on-chip memory
 - Timer
 - JTAG UART
 - 8 output-only parallel I/O (PIO) pins
 - System ID register



For complete details on these and other peripherals, see the *Quartus II Handbook Volume 5: Embedded Peripherals*.

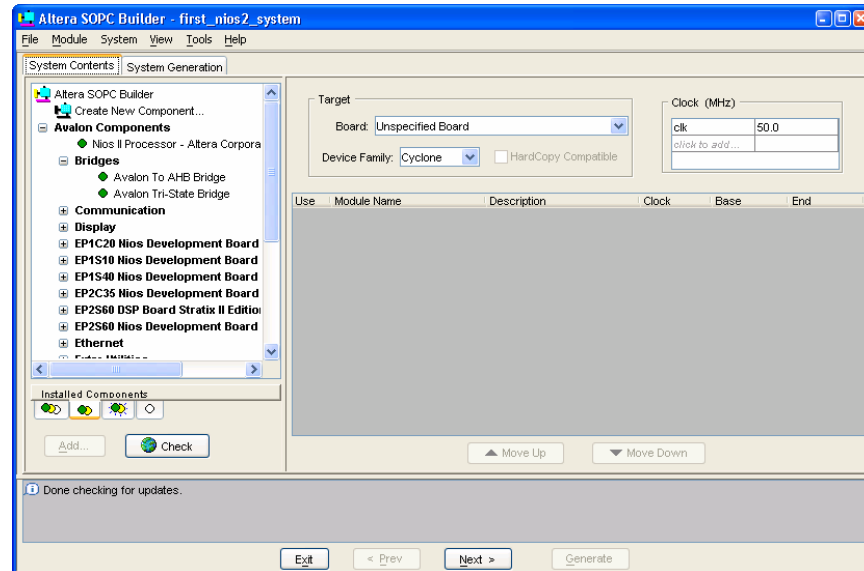
Start the Quartus II Software and Open the Tutorial Example Design Project

To start, you open the Quartus II project for the tutorial example design. 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 FPGA configuration file.

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**. The SOPC Builder GUI appears, displaying the **System Contents** tab.

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

Figure 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 peripherals 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 will perform the following steps:

1. Specify target FPGA and clock settings.
2. Add the Nios II core, on-chip memory, and other peripherals.
3. Specify base addresses and interrupt request (IRQ) priorities.
4. Specify more Nios II settings.
5. 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 straight-forward order for a new user to understand. However, you can perform SOPC Builder design steps in a different order.

Specify Target FPGA & Clock Settings

The **Target** and **Clock** settings specify the SOPC Builder system's relationship to other devices in the system. Perform the following steps:

1. Specify the target board and device family by performing one of the following steps:
 - a. If you are targeting an Altera Nios development board, select the appropriate board in the **Board** list. SOPC Builder automatically fills in **Device Family**.
 - b. If you are targeting a custom board, select **Unspecified Board** in the **Board** list, then select the **Device Family** that matches the Altera FPGA on your board.
 - c. If you do not have a target board, select the board in the **Board** list that matches the tutorial design files you are using.
2. Specify the **Clock (MHz)** frequency as shown in Table 1. `clk` is the default clock input name for the SOPC Builder system. The frequency you specify for `clk` must match the oscillator that drives the FPGA.

Table 1. Clock Frequency for Target Boards

Target Board	Frequency
Nios Development Board (all versions), or no board	50
Nios Evaluation Board (Cyclone EP1C12)	24
Custom board	Same as oscillator on board

If you specify an Altera Nios development board, SOPC Builder presents a new tab labeled **Board Settings**. This tutorial uses all the default values on the **Board Settings** tab.

Add the Nios II Processor Core

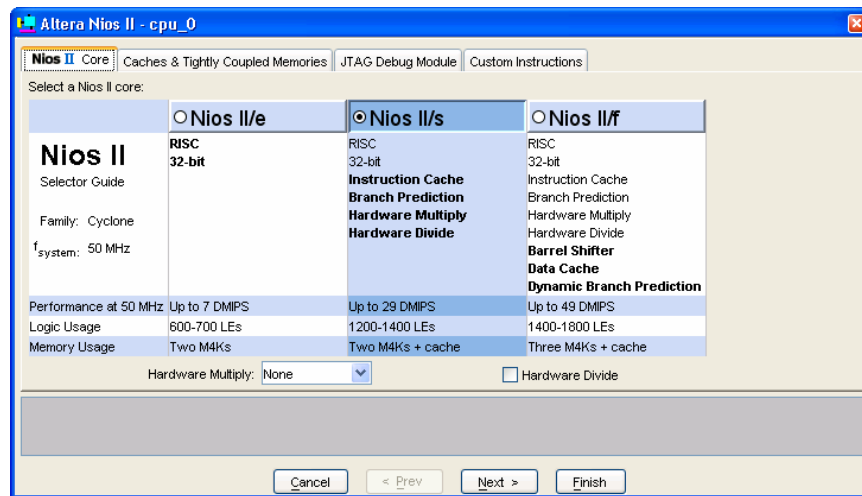
Next, you begin to add hardware components to the SOPC Builder system. As you add each component, you configure it appropriately to match the design specifications.

In this section you add the Nios II/s core and configure it to use 2 Kbytes of on-chip instruction cache memory. For educational purposes, the tutorial example design 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:

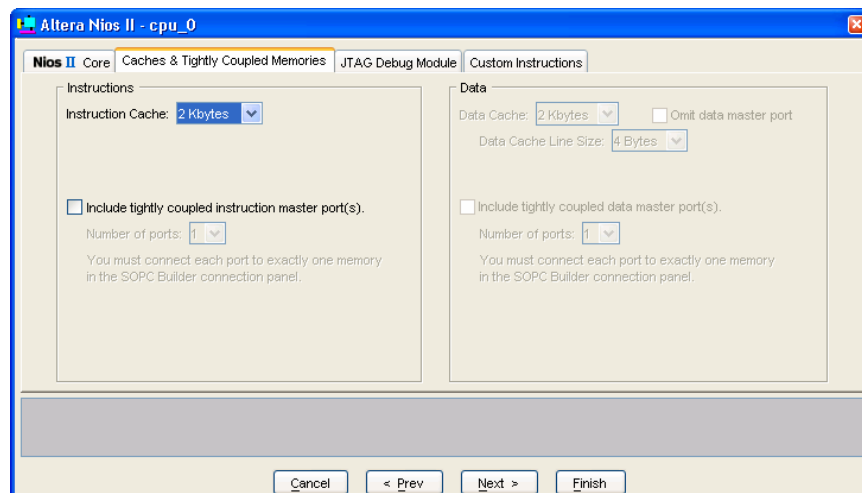
1. In the list of available components (on the left-hand side of the **System Contents** tab), select **Nios II Processor – Altera Corporation** under the **Avalon Components** category.
2. Click **Add**. The Altera Nios II configuration wizard appears, displaying the **Nios II Core** tab.
3. Specify the following settings (see Figure 5):
 - **Nios II Core:** Nios II/s
 - **Hardware Multiply:** None
 - **Hardware Divide:** Off

Figure 5. Nios II Configuration Wizard – Nios II Core Tab



4. Click **Next**. The **Caches & Tightly Coupled Memories** tab appears.
5. Specify the following settings (see Figure 6):
 - **Instruction Cache:** 2 Kbytes
 - **Include Tightly Coupled Instruction Master Port(s):** Off

Figure 6. Nios II Configuration Wizard – Caches & Tightly Coupled Memories Tab



6. Do not change any settings on the **JTAG Debug Module** tab or the **Custom Instructions** tab.
7. Click **Finish**. You return to the SOPC Builder **System Contents** tab, and an instance of the Nios II core named **cpu_0** now appears in the table of available components.

As a result of adding the Nios II core, SOPC Builder presents an additional tab titled **Nios II More "cpu_0" Settings**, which allows you to further configure the Nios II core. Error messages appear in the SOPC Builder Messages window. These messages are normal; you will fix them in later steps.



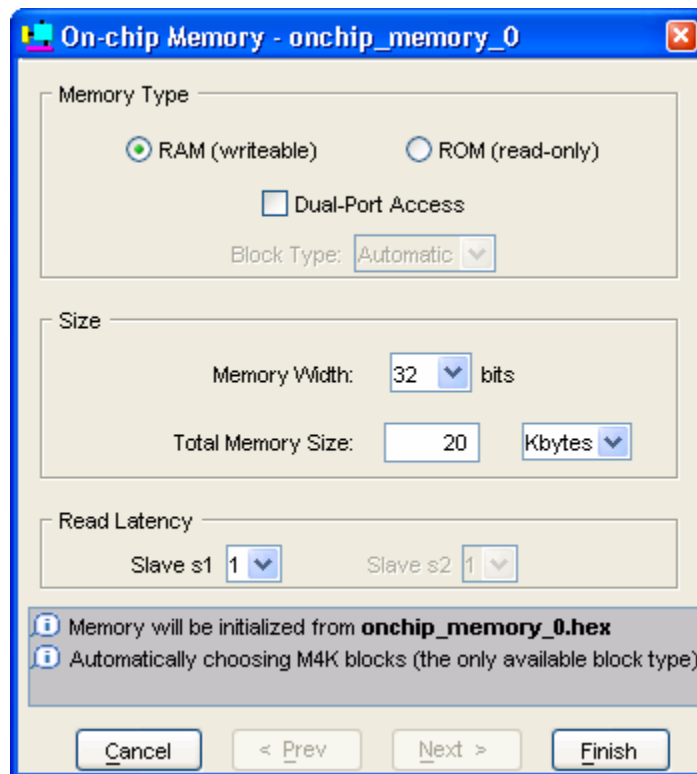
For further details on configuring the Nios II core, see the *Implementing the Nios II Processor in SOPC Builder* chapter of the *Nios II Processor Reference Handbook*. You can right-click on **cpu_0** for a list of documentation links. This SOPC Builder feature is available for all components.

Add the On-Chip Memory

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

1. In the list of available components, expand the **Avalon Components, Memory** category, and then click **On-Chip Memory (RAM or ROM)**.
2. Click **Add**. The On-Chip Memory configuration wizard appears.
3. Type **20** in the **Total Memory Size** box to specify a memory size of 20 Kbytes (see Figure 7)
4. If you are targeting a Stratix or Stratix II device, select **M4K** in the **Block Type** list. (For other device families you cannot change **Block Type**.)
5. Do not change any of the other default settings.

Figure 7. On-Chip Memory Configuration Wizard



6. Click **Finish**. You return to the SOPC Builder **System Contents** tab, and an instance of the on-chip memory named **onchip_memory_0** now appears in the table of available components.


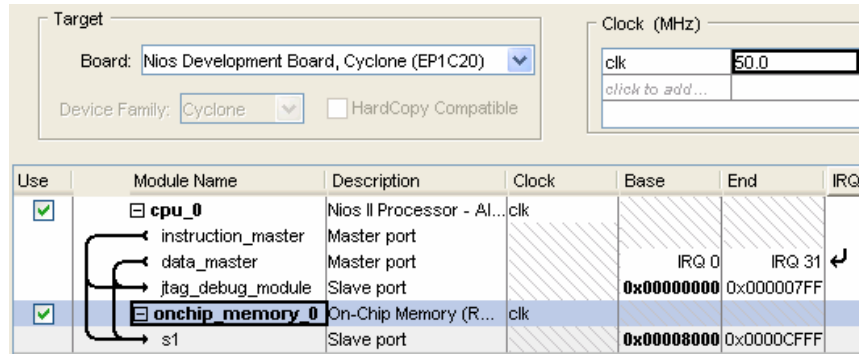

 SOPC Builder automatically connects the instruction and data master ports on the Nios II core to the memory slave port (see Figure 8). When building a system, always verify that SOPC Builder's automatic connections are appropriate for your system requirements.

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



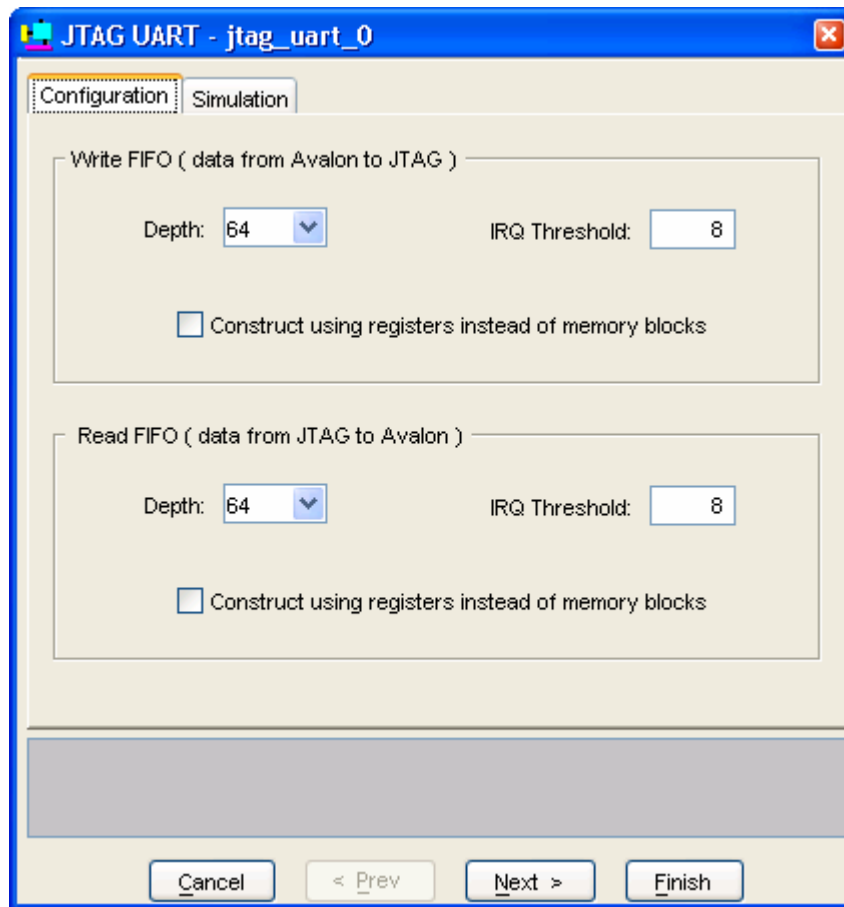
 For further details on connecting memory to Nios II systems, see the *Building Memory Subsystems Using SOPC Builder* chapter of the *Quartus II Handbook Volume 4: SOPC Builder*.

Add the JTAG UART Peripheral


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:

1. In the list of available components, expand the **Avalon Components, Communication** category, and then click **JTAG UART**.
2. Click **Add**. The JTAG UART configuration wizard appears.
3. Do not change the default settings (see Figure 9).

Figure 9. JTAG UART Configuration Wizard



4. Click **Finish**. You return to the SOPC Builder **System Contents** tab, and an instance of the JTAG UART named `jtag_uart_0` 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 on the JTAG UART, see the *JTAG UART Core with Avalon Interface* chapter of the *Quartus II Handbook Volume 5: Embedded Peripherals*.

Add the Timer Peripheral

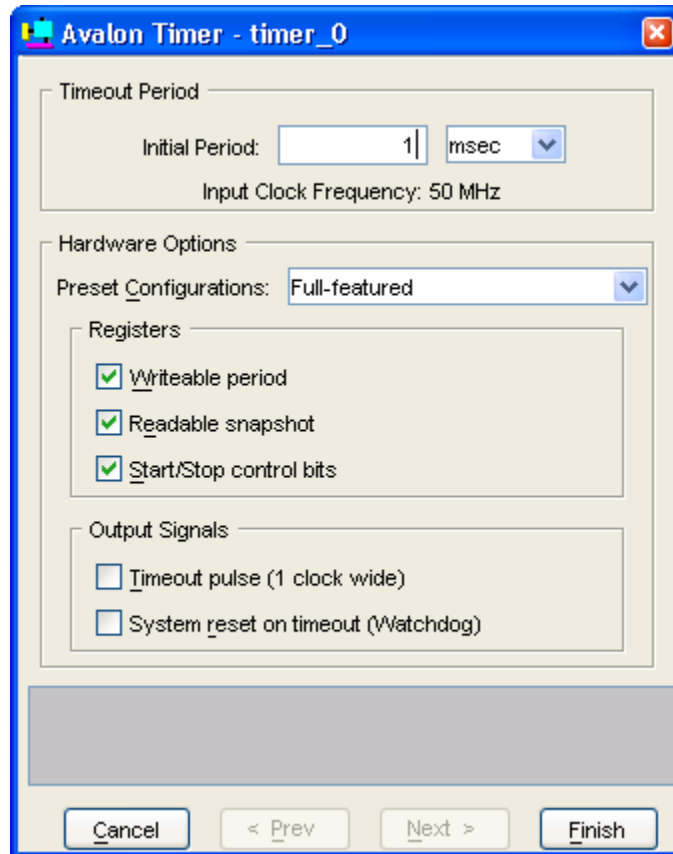
Most control systems use a timer peripheral to enable precise calculation of time. The Nios II hardware abstraction layer requires a timer to provide a periodic system clock tick.

Perform the following steps to add the timer:

1. In the list of available components, expand the **Avalon Components, Other** category, and then click **Interval timer**.

2. Click **Add**. The Avalon Timer configuration wizard appears.
3. Do not change the default settings (see Figure 10).

Figure 10. Timer Configuration Wizard



4. Click **Finish**. You return to the SOPC Builder **System Contents** tab, and an instance of the interval timer named **timer_0** now appears in the table of available components.
5. Right-click **timer_0** and click **Rename**.
6. Type `sys_clk_timer` and press Enter.



It is a good habit to give memorable names to hardware peripherals. Nios II programs use these symbolic names to access the peripheral hardware. Therefore, your choice of peripheral names can make Nios II programs easier to read and understand.



For further details on the timer, see the *Timer Core with Avalon Interface* chapter of the *Quartus II Handbook Volume 5: Embedded Peripherals*

Add the System ID Peripheral

The system ID peripheral safeguards against accidentally downloading software compiled for a different Nios II system. For example, web server software cannot run on a Nios II system lacking an Ethernet MAC. If the system

includes the system ID peripheral, the Nios II IDE prevents 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 the **Avalon Components, Other** category, and then click **System ID Peripheral**.
2. Click **Add...** You return to the SOPC Builder **System Contents** tab, and an instance of the system ID peripheral named **sysid** now appears in the table of available components.

The system ID peripheral has no user-configurable options, and therefore it doesn't have a configuration wizard.



For further details on the system ID peripheral, see the *System ID Core with Avalon Interface* chapter of the *Quartus II Handbook Volume 5: Embedded Peripherals*.

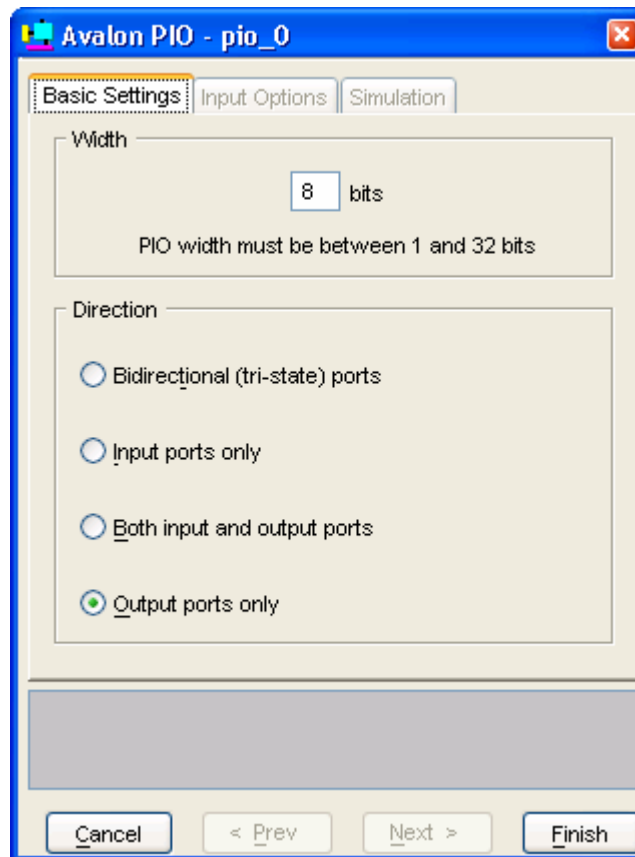
Add the PIO

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 example design uses 8 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.

1. In the list of available components, expand the **Avalon Components, Other** category, and then click **PIO (Parallel I/O)**.
2. Click **Add**. The Avalon PIO configuration wizard appears.
3. Do not change the default settings (see Figure 11). By coincidence, the configuration wizard defaults to an 8-bit output-only PIO, which exactly matches the needs for the example design.

Figure 11. PIO Configuration Wizard



4. Click **Finish**. You return to the SOPC Builder **System Contents** tab, and an instance of the PIO named **pio_0** now appears in the table of available components.
5. Right-click **pio_0** and click **Rename**.
6. Type `led_pio` and press Enter.



Nios II software uses this name to access the peripheral. You must name the PIO **led_pio**, or else programs written for this Nios II system will fail to work in later steps.



It is a good habit to give descriptive names to hardware peripherals.



For further details on the PIO, see the *PIO Core with Avalon Interface* chapter of the *Quartus II Handbook Volume 5: Embedded Peripherals*.

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 peripheral, and assign interrupt request (IRQ) priorities for the JTAG UART and the timer.

SOPC Builder provides the **Auto-Assign Base Addresses** command which makes assigning peripheral base addresses easy. For many systems, including this example design, **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 peripherals 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 will connect 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.

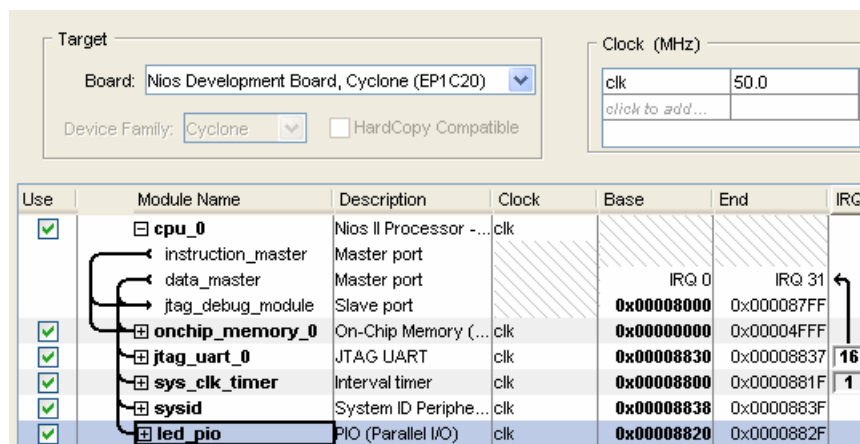
The Nios II hardware abstraction layer interprets low IRQ values as higher priority. The timer peripheral must have the highest IRQ priority to maintain the accuracy of the system clock tick.

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_0** component to select it.
3. Type 16 and press Enter to assign a new IRQ value.

Figure 12 shows the state of the SOPC Builder **System Contents** tab with the complete system.

Figure 12. System Contents Tab with Complete System

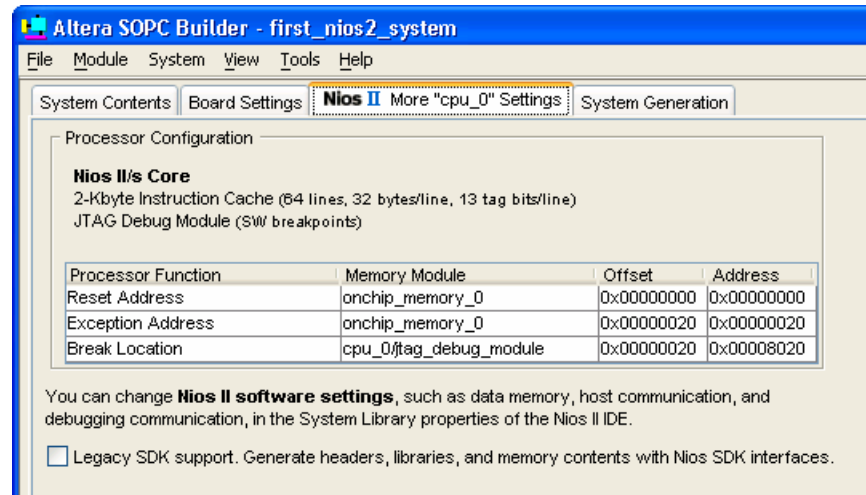


Specify More Nios II Settings

At this point, you have configured the peripherals to form a coherent system. There are a few final Nios II processor settings that depend on the overall configuration of the system. For each Nios II processor in the system SOPC Builder displays a separate tab for these system-dependent Nios II settings.

For the tutorial example design, there is only one Nios II core named **cpu_0**, and therefore SOPC Builder presents one tab labeled **Nios II More "cpu_0" Settings** (see Figure 13).

Figure 13. Nios II More "cpu_0" Settings Tab



- ✓ Do not change the default settings on the **Nios II More "cpu_0" Settings** tab; they are appropriate for the tutorial example design.

For further details on configuring the Nios II core, see the *Implementing the Nios II Processor in SOPC Builder* chapter of the *Nios II Processor Reference Handbook*.

Generate the SOPC Builder System

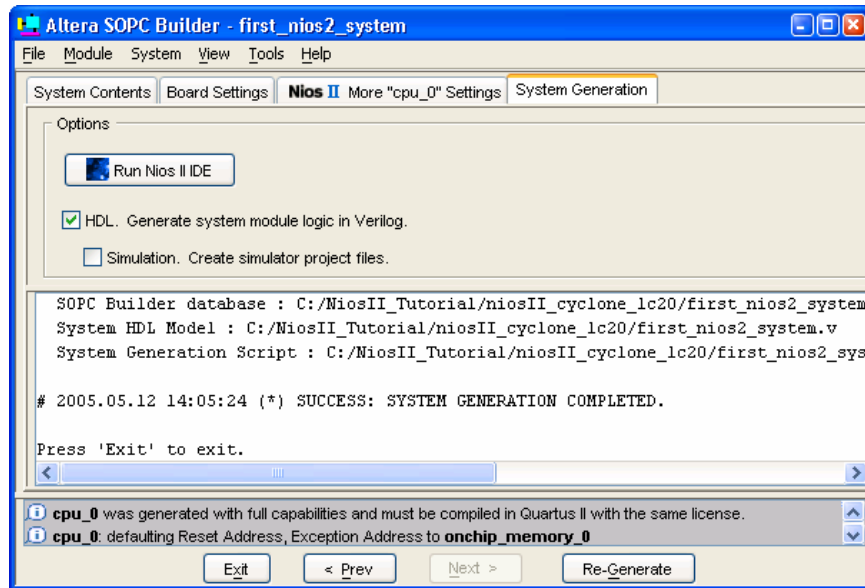
You are now ready to generate the SOPC Builder system. Perform the following steps:

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 system generation process begins.

The generation process can take several minutes. When it completes, the System Generation tab displays a message "SUCCESS: SYSTEM GENERATION COMPLETED." (see Figure 14).

4. Click **Exit** and return to the Quartus II software.

Figure 14. Successful System Generation



Congratulations! You have finished creating the Nios II processor system. After generation completes, you are ready to integrate the system into the Quartus II hardware project and use the Nios II IDE to develop software.

For further details on generating systems with SOPC Builder, see the *Quartus II Handbook Volume 4: SOPC Builder*. For details on hardware simulation for Nios II system, see *AN351: Simulating Nios II Embedded Processor Designs*.

Integrate the SOPC Builder System into Quartus II Project

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


- Instantiate the SOPC Builder system module in the Quartus II project.
- Assign FPGA pins.
- Compile the Quartus II project
- Verify timing

You can skip ahead to "Develop Software Using the Nios II IDE" on page 29 if you do not have a target board. Alternately, you can read this section to familiarize yourself with more of the hardware design flow. However, the steps in this section do not affect the outcome of the tutorial if you do not have a target board.

For further information on using the Quartus II software, see the *Quartus II Tutorial* in the Quartus II help system, and both the *Introduction to Quartus II Manual* and the *Quartus II Handbook*, available at www.altera.com.

Instantiate the SOPC Builder System Module in the Quartus II Project.

SOPC Builder outputs a design entity called the system module. The tutorial example design uses the block diagram file (BDF) method of design entry, so you will instantiate a system module symbol **first_nios2_system** into the BDF.

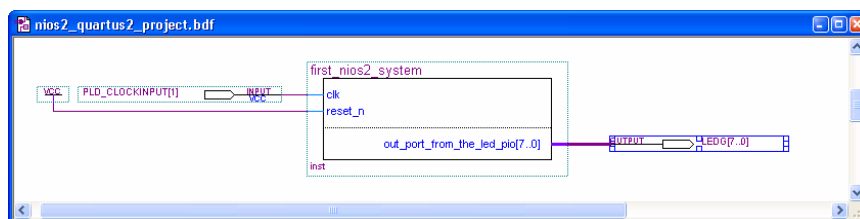
 How you instantiate the system module depends on the design entry method of the overall Quartus II project. For example, if you were using Verilog HDL for design entry, you would instantiate the Verilog module **first_nios2_system** defined in the file **first_nios2_system.v**.

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. The first_nios2_system symbol tracks with your mouse pointer.
5. Connect the inputs on the symbol with the wires on the left-hand side of the BDF.
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, perform the following step to connect the **LEDG[7..0]** output pins to the first_nios2_system.
 - ✓ Click and drag **LEDG[7..0]** to connect it with the port **out_port_from_the_led_pio[7..0]** on the **first_nios2_system** symbol.

Figure 15 shows the complete BDF using the LED pins.

Figure 15. Complete BDF



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. Left click the output symbol **LEDG[7..0]** to select it.
 - b. Press Delete.
9. To save the completed BDF file, click **Save** on the File menu.

Assign FPGA pins

If you are targeting a custom board, you must assign a specific target device and then assign FPGA pin locations to match the pinout of your board.



Skip ahead to section “Compile the Quartus II Project and Verify Timing” on page 27. If you are targeting a Nios development board. The provided Quartus II project files already contain appropriate assignments for Nios development boards.

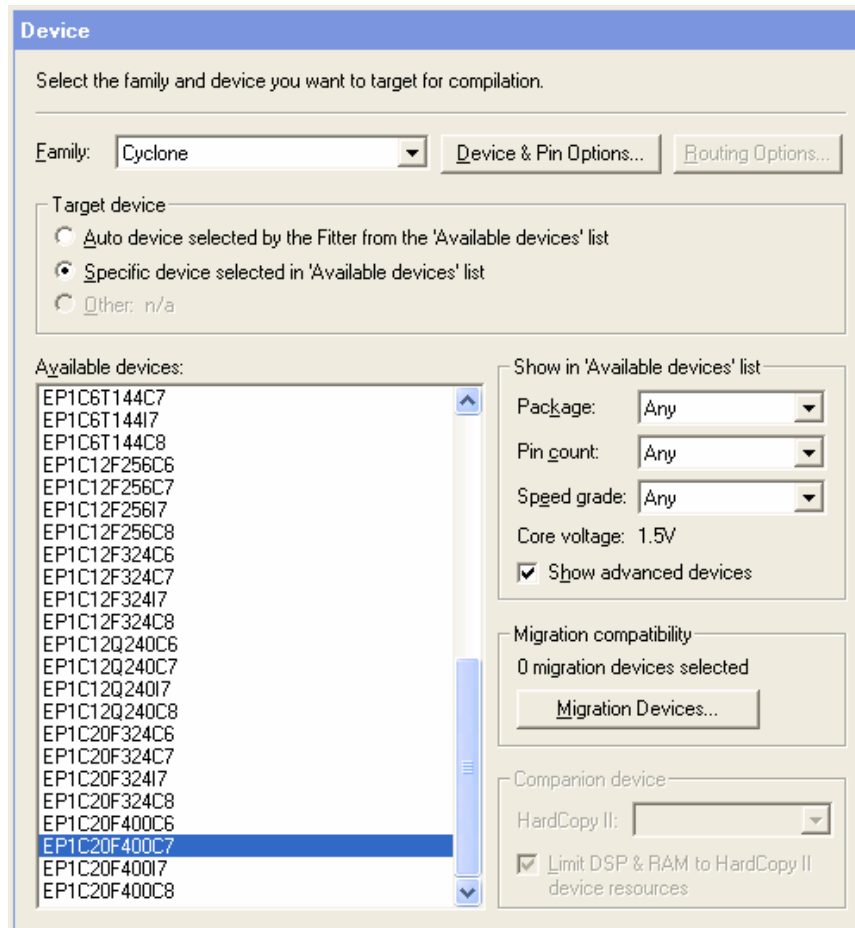
You must know the pin layout for the custom board to complete this section. You also must know other requirements for using the board, which are beyond the scope of this document.

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.
3. Click **No** if a dialog box asks, "Device family selection has changed. Do you want to remove all location assignments?"
4. Under **Target Device** select **Specific device selected in 'Available devices' list**.
5. Under **Available devices** select the exact device that matches your board.
6. Click **No** if a dialog box asks, "Altera recommends removing all location assignments when changing the device. Do you want to remove all location assignments?"
7. Click **OK** to accept the device assignment.

Figure 16 shows an example of the **Settings** dialog box assigning a Cyclone device.

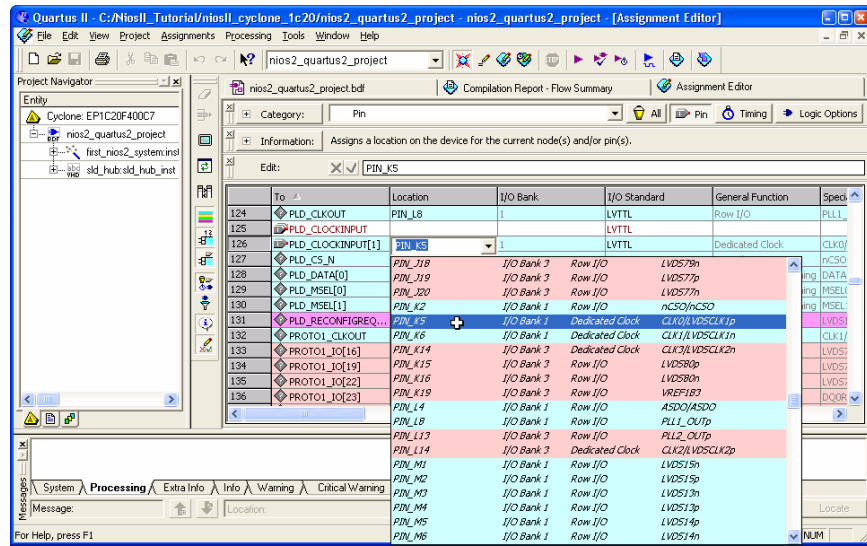
Figure 16. Assigning a Device in the Quartus II Settings Dialog Box



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

1. On the Assignments menu, click **Pins**. The Quartus II Assignment Editor appears. The Quartus II project has many ready-made assignments appropriate for a Nios development board, which you must reassign to suit your board.
2. In the Assignment Editor click the **To** column heading to sort the pin assignments by name.
3. Scroll down until the pin **PLD_CLOCKINPUT[1]** appears in the Assignment Editor.
4. Double-click in the **Location** cell for pin **PLD_CLOCKINPUT[1]**. A list of available pin locations appears.
5. Select the appropriate FPGA pin that connects to the oscillator on the board (see Figure 17.).

Figure 17. Assigning Pins with the Quartus II Assignment Editor



- If you connected the LED pins in the BDF, repeat steps 3 to 5 to assign appropriate pin locations for each of the LED outputs pins: **LEDG[0], LEDG[1], LEDG[2], LEDG[3], LEDG[4], LEDG[5], LEDG[6], LEDG[7]**.
- On the File menu, click **Save** to save the assignments.
- Close the Assignment Editor.



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 to ensure that the following conditions will not damage the board:

- After power-up all unused I/O pins on the FPGA will enter a high-impedance state.
- The IO banks are configured for the 3.3V LVTTTL I/O standard. The board must supply 3.3V to the FPGA's VCCIO pins.
- The **LEDG[7..0]** outputs will drive 3.3V.



For further details on making assignments in the Quartus II software, see the *Quartus II Handbook Volume 2: Design Implementation & Optimization*.

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 an FPGA configuration file that you can download to the board. After the compilation completes, you must analyze the timing performance of FPGA design to verify that the design will work in hardware.

Perform the following steps:

- On the Processing menu, click **Start Compilation**.

2. The Quartus II Status utility 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 window.
4. Expand the **Timing Analyzer** category of the Compilation Report window.
5. Click **Summary**.
6. Check the frequency listed in the **Actual Time** cell associated with **PLD_CLOCK[1]**. This is the maximum frequency (F_{MAX}) that this FPGA design is capable of running.



If the **Actual Time** frequency for **PLD_CLOCK[1]** is less than the oscillator frequency on the board, this design will not operate in hardware. You must make Quartus II timing assignments to optimize the clock, or reduce the oscillator frequency driving the FPGA.

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



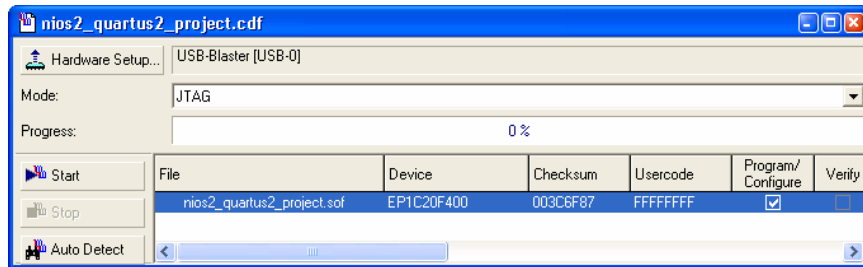
For further details on meeting timing requirements in the Quartus II software, see the *Quartus II Handbook Volume 1: Design & Synthesis*.

Download Hardware Design to Target FPGA

In this section you will download the FPGA configuration file (**.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 Programmer window appears and automatically displays the appropriate configuration file (**nios2_quartus2_project.sof**).
3. Click **Hardware Setup** in the top-left corner of the Programmer window 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.
5. Click **Close**.
6. Turn on **Program/Configure** for **nios2_quartus2_project.sof** (see Figure 18).
7. Click **Start**. The **Progress** meter sweeps to 100% as the Quartus II software configures the FPGA.

Figure 18. Quartus II Programmer Window



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

Develop Software Using the Nios II IDE

In this section you will start the Nios II integrated development environment (IDE) 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 IDE to develop programs, see the *Software Development Tutorial* available from the IDE help system.

In this section you will perform the following actions:

1. Create a New C/C++ Application Project
2. Compile the Project.

To perform this section, you must have the SOPC Builder system file (.ptf) you created in “Define the System in SOPC Builder” on page 12.

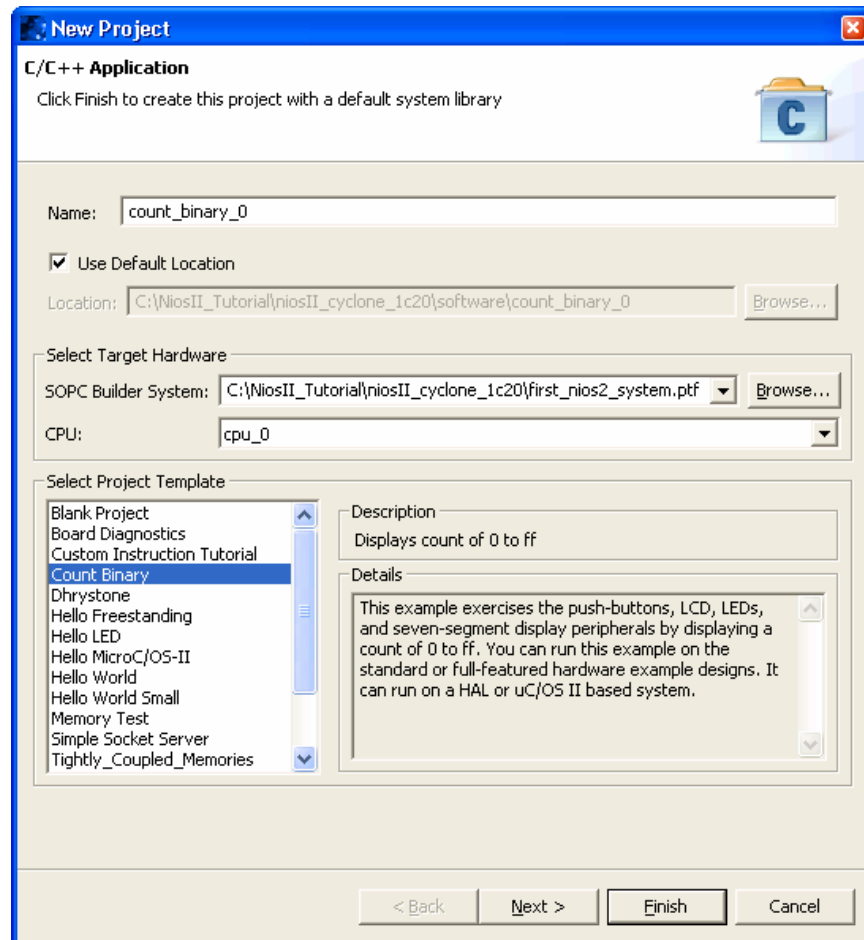
Create a New C/C++ Application Project

In this section you will create a new Nios II C/C++ Application Project. Perform the following steps:

1. Start the Nios II IDE. On Windows computers, click **Start**, point to **Programs, Altera, Nios II EDS <version>**, and then click **Nios II IDE**. On Linux computers, run the executable file **<Nios II kit path>/bin/nios2-ide**.
2. When the **Workspace Launcher** dialog box appears, click **OK** to accept the default workspace location.
3. On the File menu, point to **New**, and then click **C/C++ Application** to open the **New Project** wizard.
4. Click **Browse** under **Select Target Hardware**. The Select Target Hardware dialog box opens.
5. Browse to **<Design Files Directory>**.
6. Select **first_nios2_system.ptf**.
7. Click **Open**. You return to the **New Project** wizard, and the **SOPC Builder System** and **CPU** fields are now filled in.

8. Select **Count Binary** in the **Select Project Template** list. The **Name** field automatically updates to **count_binary_0**. See Figure 19.
9. Click **Finish**.

Figure 19. Nios II IDE New Project Wizard



The Nios II IDE displays two new projects in the C/C++ Projects view on the left-hand side of the workbench: **count_binary_0** and **count_binary_0_syslib**. **count_binary_0** is your C/C++ application project, and **count_binary_0_syslib** is a system library that encapsulates the details of the Nios II system hardware.



The left-hand pane of the IDE workbench has two tabbed views: The C/C++ Projects view and the Navigator view. Click the **C/C++ Projects** tab to display the C/C++ Projects view. This view is appropriate for most C/C++ development activity.

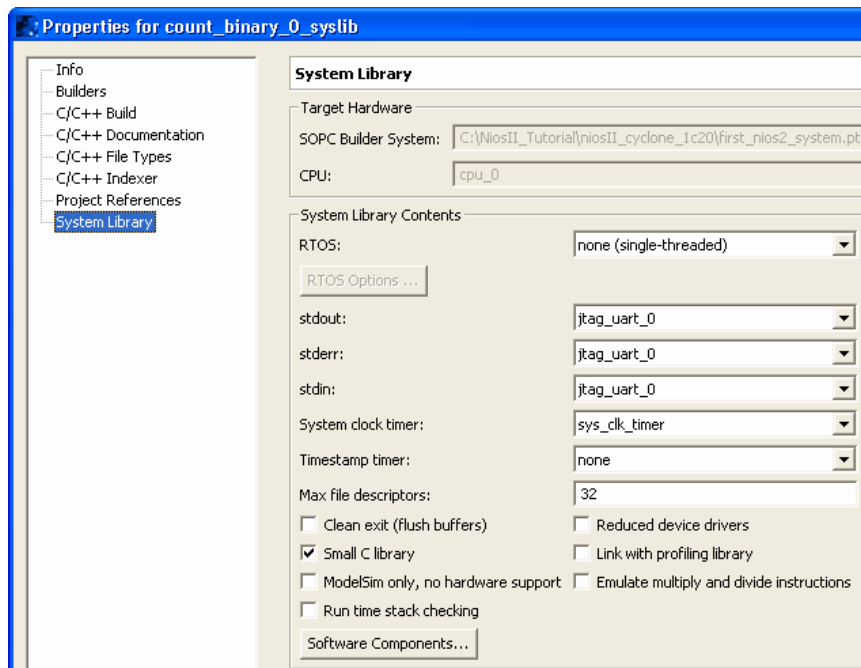
Compile the Project


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

Perform the following steps:

1. Right-click `count_binary_0` and click **System Library Properties**. The **Properties for count_binary_0_syslib** dialog box opens.
2. Click the **System Library** page.
3. The **System Library** page contains all settings related to how the program interacts with the underlying hardware. Therefore, the settings here reflect names you specified when creating the Nios II hardware in section "Define the System in SOPC Builder" on page 12).
4. Change the following settings which affect the size of the compiled executable (see Figure 20).
 - a. Turn off **Clean exit (flush buffers)**.
 - b. Turn on **Small C library**.
5. Click **OK** to close the **Properties** dialog box and return to the IDE workbench.

Figure 20. System Library Properties



 For further details on the system library see the *Nios II Software Developer's Handbook*.

To compile the project, perform the following step:

- ✓ Right-click the `count_binary_0` project in the C/C++ Projects view and click **Build Project**.


The **Build Project** dialog box appears, and the IDE begins compiling the project. When compilation completes, the message "Build completed" appears in the Console view.

Run the Program

In this section you will run the program to see the compiled code execute. You can run the program on target hardware, on the Nios II instruction set simulator (ISS), or both.

Run the Program on Target Hardware

In this section you will download the program to target hardware and execute it.

 If you do not have a target board, skip ahead to "Run the Program on Target Hardware" on page 32. To proceed, you must have completed the steps in "Download Hardware Design to Target FPGA" on page 28.

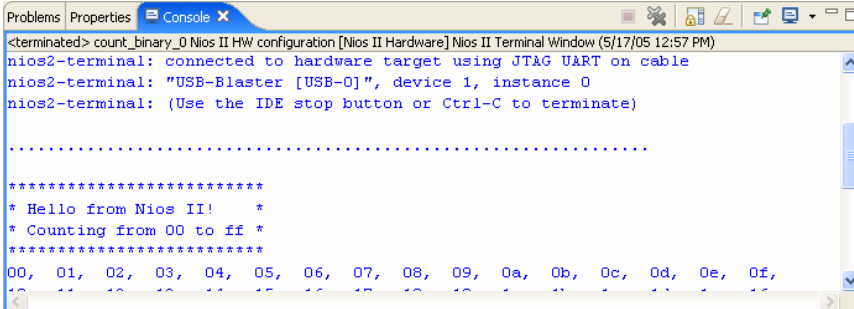
To download the software executable to the target board, perform the following steps:

1. Right-click the **count_binary_0** project, point to **Run As**, and then click **Nios II Hardware**. The IDE downloads the program to the FPGA on the target board and starts execution.

When the target hardware starts executing the program, the Console view displays character I/O output (see Figure 21). If you connected LEDs to the Nios II system in "Integrate the SOPC Builder System into Quartus II Project" on page 23, then the LEDs blink in a binary counting pattern.

2. Click **Terminate** (the red square) on the toolbar at the upper-right hand corner of the Console view to terminate the run session. When you click **Terminate**, the IDE disconnects from the target hardware and leaves the Nios II processor running.

Figure 21. Console View Displaying Nios II Hardware Output



```


<terminated> count_binary_0 Nios II HW configuration [Nios II Hardware] Nios II Terminal Window (5/17/05 12:57 PM)
nios2-terminal: connected to hardware target using JTAG UART on cable
nios2-terminal: "USB-Blaster [USB-0]", device 1, instance 0
nios2-terminal: (Use the IDE stop button or Ctrl-C to terminate)

.....

*****
* 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,

```

You can make edits to the **count_binary.c** program in the IDE 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, see the *Software Development Tutorial* available from the Nios II IDE help system.

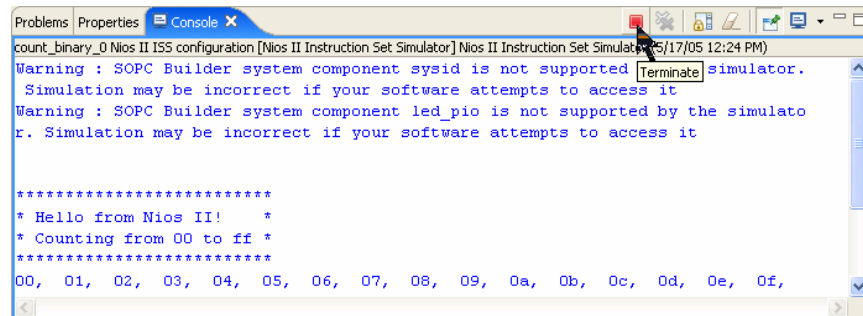
Run the Program on the ISS

In this section you will run the **count_binary_0** program on the Nios II ISS. Perform the following steps:

1. Right-click the **count_binary_0** project, point to **Run As**, and then click **Nios II Instruction Set Simulator**.

2. When the ISS starts executing the program, the Console view displays character I/O output from the program. See Figure 22. The count output appears very slowly because there are delay loops in the code.
3. Click the **Terminate** button (the red square) on the toolbar at the upper-right hand corner of the Console view to terminate the ISS session.

Figure 22. Console View Displaying Instruction Set Simulator Output



You can make edits to the `count_binary.c` program in the IDE and repeat these two steps to witness your changes executing on the ISS.



For information on running and debugging programs on the ISS, see the *Software Development Tutorial* available from the Nios II IDE help system.

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 & verifying timing
- Creating a new project in the Nios II IDE
- Compiling the project
- Running the software on the ISS and 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 on developing software for the Nios II processor.
- *Software Development Tutorial* available from the Nios II IDE help system – This tutorial teaches in detail how to use the Nios II IDE to develop, run, and debug new C/C++ application projects.
- *Nios II IDE Help System* – The help system in the IDE provides complete reference on features of the IDE. To open the help system, click **Help Contents** on the Help menu, then click the **Nios II IDE Help** book in the **Contents** pane.

- *Nios II Processor Reference Handbook* – This handbook provides complete reference for the Nios II processor hardware.
- *Quartus II Handbook Volume 5: Embedded Peripherals* – This handbook contains details on the peripherals provided free as part of the Nios II development tools.

Quartus II Handbook Volume 4: SOPC Builder – This volume provides complete reference on using SOPC Builder, including topics such as building memory subsystems and creating custom components.

For a complete list of all documents available for the Nios II processor, visit the Nios II literature page at www.altera.com/nios2.



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

© 2006 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. 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.

