Nios II Hardware Development

Introduction

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 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 in 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

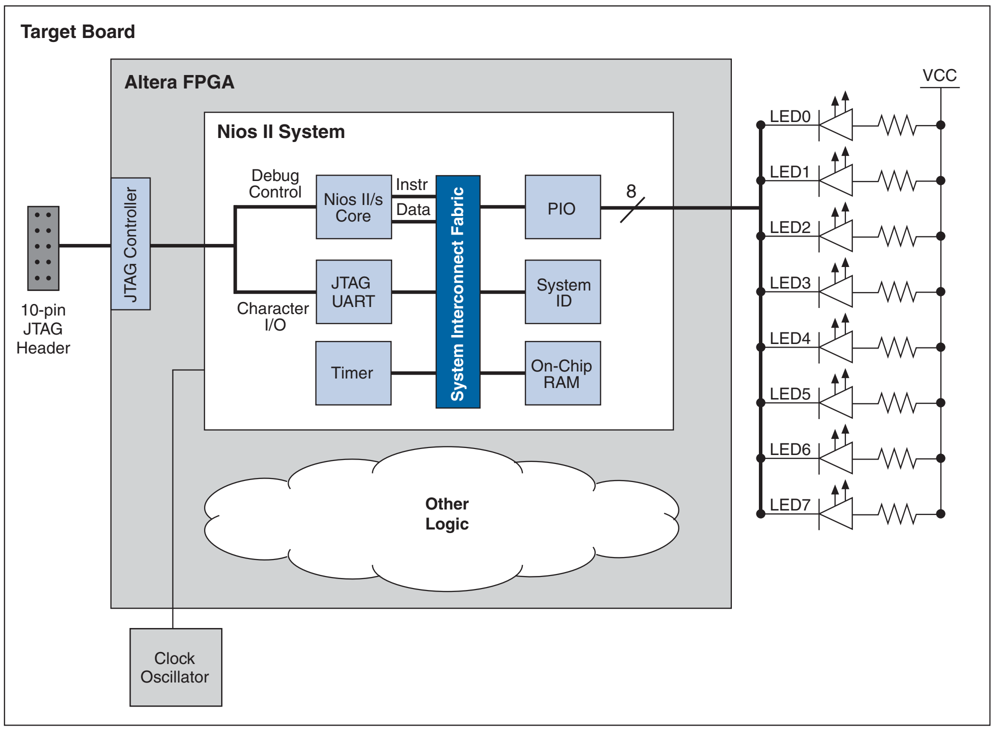
The example design you build in this tutorial demonstrates a small Nios II system for control applications, which displays character I/O output and blinks LEDs in a binary counting pattern. This Nios II system can also communicate with a host computer, allowing the host computer to control logic inside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

The example Nios II system contains the following:

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



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 example design 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 7.1 or later – The software must be installed on a Windows or Linux computer that meets the Quartus II minimum requirements.
- Nios II Embedded Design Suite version 7.1 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 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

For information on Nios II 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:
    - Any Stratix III or Stratix II device
    - Stratix EP1S10 device or larger
    - Any Cyclone III or Cyclone II device
    - 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. Revised cables have a clearly marked revision label; earlier cables do not.

OpenCore Plus Evaluation

You can perform this tutorial, even 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, as well as evaluate its size and speed 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

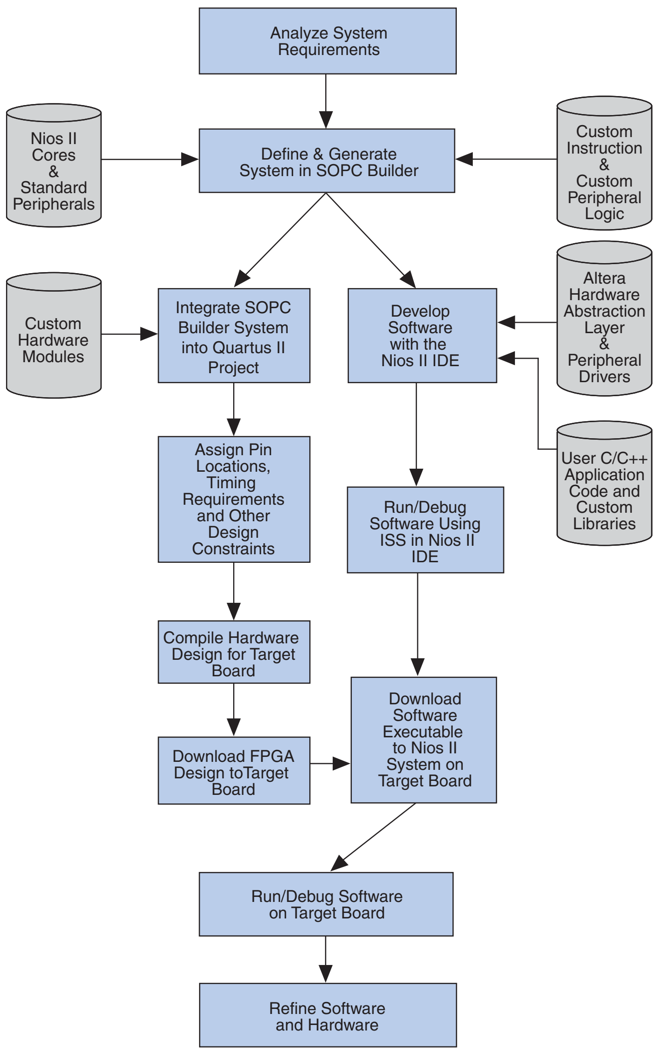
You only need to purchase a license for the Nios II processor when you are completely satisfied with its functionality and performance, and want to take your design to production.

For more information on OpenCore Plus, refer to *AN 320: 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


Analyze System Requirements
Nios II Cores & Standard Peripherals
Define & Generate System in SOPC Builder
Custom Instruction & Custom Peripheral Logic
Custom Hardware Modules
Integrate SOPC Builder System into Quartus II Project
Develop Software with the Nios II IDE
Altera Hardware Abstraction Layer & Peripheral Drivers
User C/C++ Application Code and Custom Libraries
Assign Pin Locations, Timing Requirements and Other Design Constraints
Run/Debug Software Using ISS in Nios II IDE
Compile Hardware Design for Target Board
Download FPGA Design toTarget Board
Download Software Executable to Nios II System on Target Board
Run/Debug Software on Target Board
Refine Software and Hardware

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 components 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 component eliminate wasted processor cycles copying data?
  - Could a custom instruction replace the critical loop of a DSP algorithm?
  - Could the Nios II C-to-Hardware (C2H) Acceleration Compiler improve performance?

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

Defining and 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 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 Embedded Design Suite. 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, 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 SOPC Builder System 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 into an SRAM Object File (**.sof**).

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

Quartus II Hardware Development Tasks

After you generate the Nios II system using SOPC Builder, you integrate it into the overall 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 an SRAM Object File to configure the FPGA.

You download the SRAM Object 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 both *Introduction to the Quartus II Software* and the *Quartus II Handbook,* available at **www.altera.com/literature/lit-qts.jsp**.

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 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 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 create a new Nios II C/C++ application project, the Nios II IDE requires the SOPC Builder System File. You also need the SRAM Object File 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.
- Executable and Linkable Format File (**.elf**) – An Executable and Linkable Format 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 Executable and Linkable Format 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 and 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. 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.

The Nios II Embedded Design Suite provides several working Nios II reference designs, which you can use as a starting point for your own designs. After installing the Nios II Embedded Design Suite, see the directory ***<Nios II EDS install path>*****/examples/verilog** or the directory ***<Nios II EDS install 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 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 1–5. If you are building a Nios II system based on the standard components 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 perform the following steps:

1. "Install the Design Files" on page 1–11.

2. "Analyze System Requirements" on page 1–12.
3. "Start the Quartus II Software and Open the Tutorial Example Design Project" on page 1–12.
4. "Create a New SOPC Builder System" on page 1–14.
5. "Define the System in SOPC Builder" on page 1–15.
6. "Integrate the SOPC Builder System into the Quartus II Project" on page 1–28.
7. "Download Hardware Design to Target FPGA" on page 1–33.
8. "Develop Software Using the Nios II IDE" on page 1–34.
9. "Run the Program" on page 1–38.

Install the Design Files

Before you proceed, you must install the Quartus II software and the Nios II Embedded Design Suite. 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 StratixII device, download the StratixII 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.
- 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 Kbytes of instruction cache
  - 20 Kbytes of on-chip memory
  - Timer
  - JTAG UART
  - Eight output-only parallel I/O (PIO) pins
  - System ID component

For complete details on these and other components, 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 SRAM Object File.

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**. Be careful not to mistake **Open** for **Open Project**. The **Open Project** dialog box appears.

3. Browse to <*Design Files Directory*>.

4. Select the file **nios2_quartus2_project.qpf** and click **Open**. The Quartus II software opens the project.

5. If the Quartus II software does not automatically display the Block Diagram File (**.bdf**) **nios2_quartus2_project.bdf** (see Figure 1–3), perform the following steps:

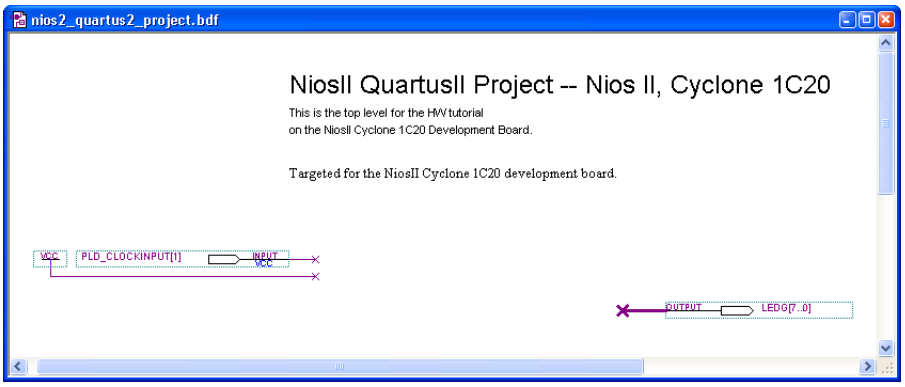
   a. On the File menu, click **Open**. The **Open** dialog box appears.

   b. Browse to <*Design Files Directory*>.

   c. Select the file **nios2_quartus2_project.bdf** and click **Open**.

Figure 1–3 shows the Block Diagram File **nios2_quartus2_project.bdf**.

Figure 1–3. Example Design Block Diagram File



NiosII QuartusII Project -- Nios II, Cyclone 1C20

This is the top level for the HW tutorial
on the NiosII Cyclone 1C20 Development Board.

Targeted for the NiosII Cyclone 1C20 development board.

The Block Diagram File 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.

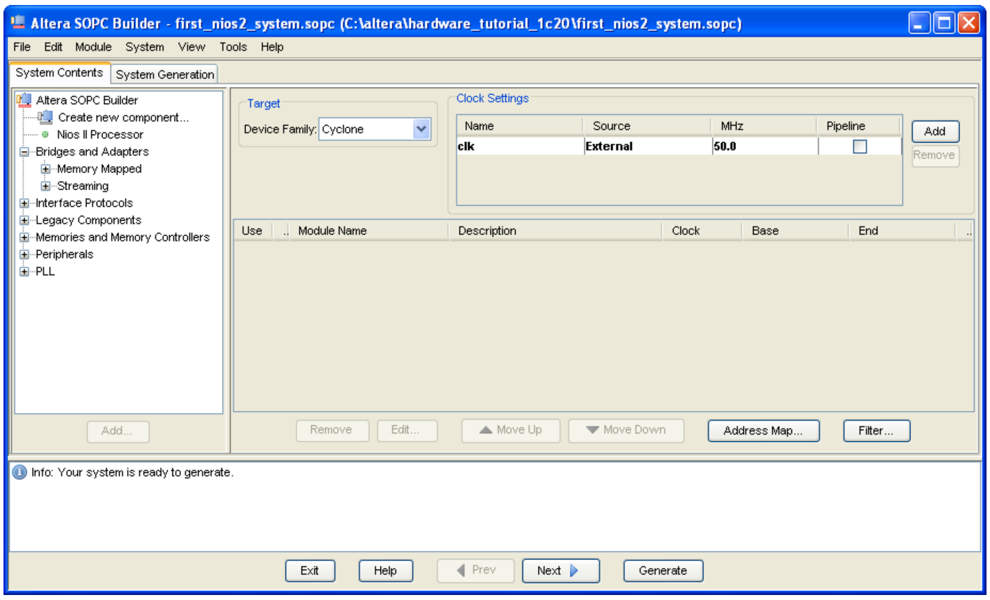
2. Type `first_nios2_system` as the **System Name**.

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

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

Figure 1–4. SOPC Builder GUI


Altera SOPC Builder - first_nios2_system.sopc (C:\altera\hardware_tutorial_1c20\first_nios2_system.sopc)

Device Family: Cyclone

Name	Source	MHz	Pipeline
clk	External	50.0	☐

Info: Your system is ready to generate.

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.
2. Double-click the clock frequency in the **MHz** column for `clk`. Type the clock frequency as shown in Table 1–1, and press **Enter**. `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 1–1. Clock Frequency for Target Boards

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

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 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 (on the left-hand side of the **System Contents** tab), expand **Memories and Memory Controllers**, expand **On-Chip**, and then click **On-Chip Memory (RAM or ROM).**

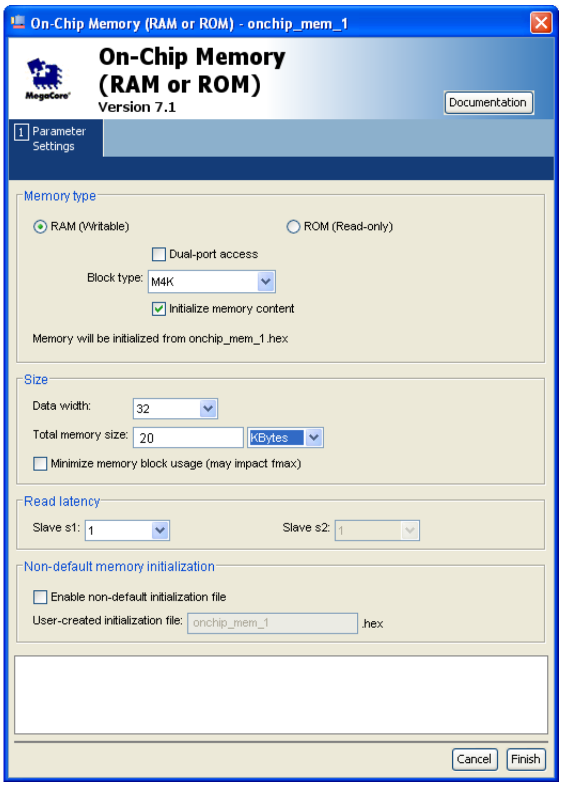
2. Click **Add**. The On-Chip Memory (RAM or ROM) MegaWizard interface appears.

3. In the **Block Type** list, select **M4K**.

4. In the **Total memory size** box, type 20 and select **Kbytes** to specify a memory size of 20 Kbytes (see Figure 1–5).

5. Do not change any of the other default settings.

Figure 1–5. On-Chip Memory MegaWizard



Memory type: RAM (Writable) [✓] / ROM (Read-only)
Dual-port access: unchecked
Block type: M4K
Initialize memory content: [✓]
Memory will be initialized from onchip_mem_1.hex

Size
Data width: 32
Total memory size: 20 KBytes
Minimize memory block usage (may impact fmax): unchecked

Read latency
Slave s1: 1
Slave s2: 1

Non-default memory initialization
Enable non-default initialization file: unchecked
User-created initialization file: onchip_mem_1 .hex

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

For further details on 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.

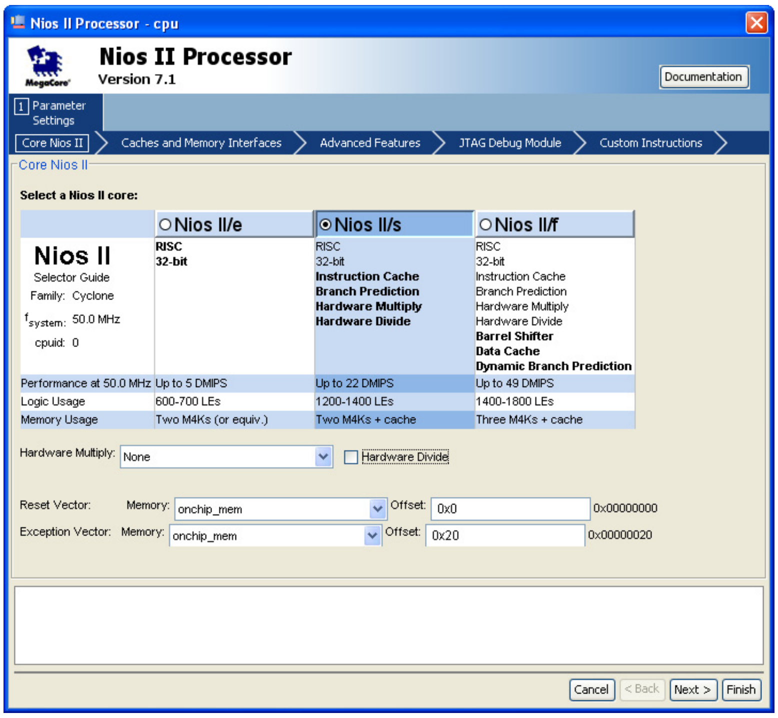
Add the Nios II Processor Core

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, select **Nios II Processor**.
2. Click **Add**. The Nios II Processor MegaWizard interface appears, displaying the **Nios II Core** page.
3. Specify the following settings (see Figure 1–6):
   - **Nios II Core:** Nios II/s
   - **Hardware Multiply:** None
   - **Hardware Divide:** Off
   - **Reset Vector: Memory:** onchip_mem **Offset:** 0x0
   - **Exception Vector: Memory:** onchip_mem **Offset:** 0x20

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



Core Nios II

Select a Nios II core:

Nios II Selector Guide Family: Cyclone f_{system}: 50.0 MHz cpuid: 0	Nios II/e	[✓ Nios II/s]	Nios II/f
	RISC 32-bit	RISC 32-bit Instruction Cache Branch Prediction Hardware Multiply Hardware Divide	RISC 32-bit Instruction Cache Branch Prediction Hardware Multiply Hardware Divide Barrel Shifter Data Cache Dynamic Branch Prediction
Performance at 50.0 MHz	Up to 5 DMIPS	Up to 22 DMIPS	Up to 49 DMIPS
Logic Usage	600-700 LEs	1200-1400 LEs	1400-1800 LEs
Memory Usage	Two M4Ks (or equiv.)	Two M4Ks + cache	Three M4Ks + cache

Hardware Multiply: None ☐ Hardware Divide

Reset Vector: Memory: onchip_mem Offset: 0x0 0x00000000

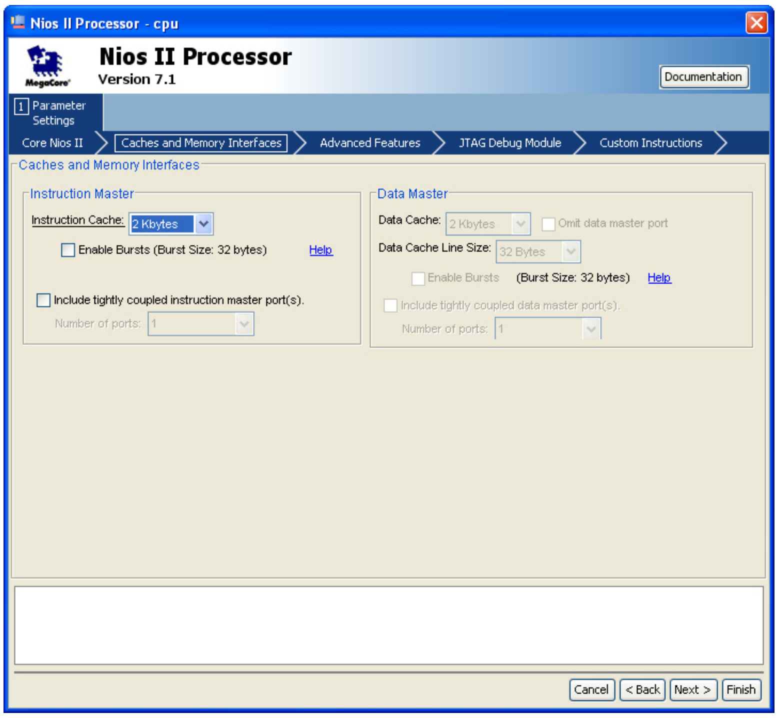
Exception Vector: Memory: onchip_mem Offset: 0x20 0x00000020

4. Click **Caches and Memory Interfaces**. The **Caches and Memory Interfaces** page appears.

5. Specify the following settings (see Figure 1–7):

   - **Instruction Cache:** 2 Kbytes
   - **Enable Bursts:** Off
   - **Include tightly coupled instruction master port(s):** Off

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



Nios II Processor - cpu

Nios II Processor
Version 7.1

Caches and Memory Interfaces

Instruction Master
Instruction Cache: 2 Kbytes
Enable Bursts (Burst Size: 32 bytes)
Include tightly coupled instruction master port(s).
Number of ports: 1

Data Master
Data Cache: 2 Kbytes
Omit data master port
Data Cache Line Size: 32 Bytes
Enable Bursts (Burst Size: 32 bytes)
Include tightly coupled data master port(s).
Number of ports: 1

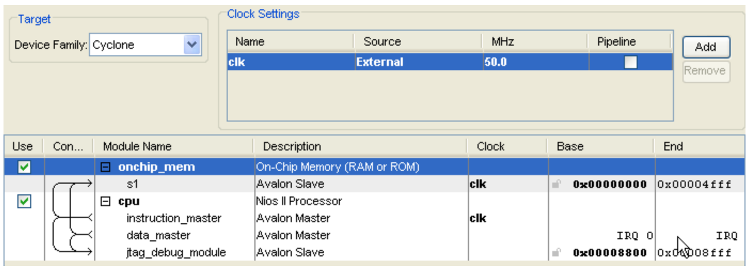
6. Do not change any settings on the **Advanced Features, JTAG Debug Module,** or **Custom Instructions** pages.

7. Click **Finish**. You return to the SOPC Builder **System Contents** tab, and an instance of the Nios II core named **cpu** now appears in the table of available components.

For further details on configuring the Nios II core, see 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 (see Figure 1–8). 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



Name	Source	MHz	Pipeline
clk	External	50.0	☐

Use	Con...	Module Name	Description	Clock	Base	End
✓		onchip_mem	On-Chip Memory (RAM or ROM)			
		s1	Avalon Slave	clk	0x00000000	0x00004fff
✓		cpu	Nios II Processor			
		instruction_master	Avalon Master	clk		
		data_master	Avalon Master		IRQ 0	IRQ
		jtag_debug_module	Avalon Slave		0x00008800	0x00008fff

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

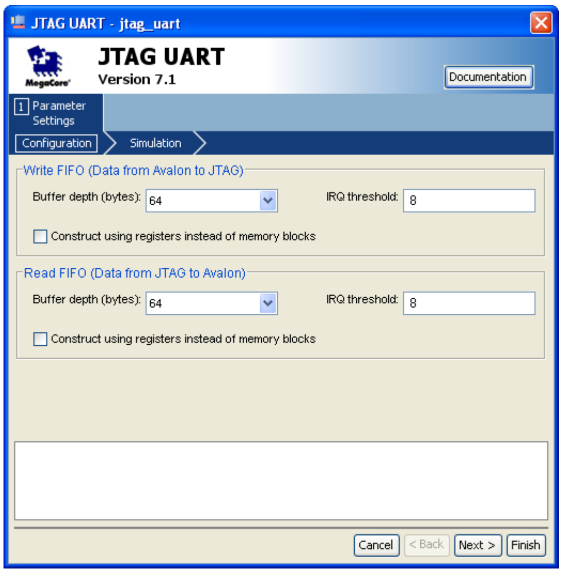
The JTAG UART provides a convenient way to communicate character data with the Nios II processor through the USB-Blaster download cable. Perform the following steps to add the JTAG UART:

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.

3. Do not change the default settings (see Figure 1–9).

Figure 1–9. JTAG UART MegaWizard



Write FIFO (Data from Avalon to JTAG)

Buffer depth (bytes): 64 IRQ threshold: 8

☐ Construct using registers instead of memory blocks

Read FIFO (Data from JTAG to Avalon)

Buffer depth (bytes): 64 IRQ threshold: 8

☐ Construct using registers instead of memory blocks

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

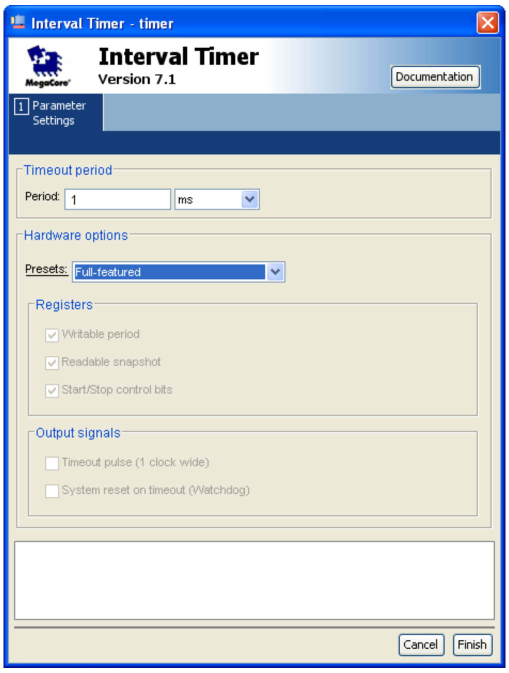
Add the Interval Timer

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.
3. In the **Presets** list, select **Full-featured**.
4. Do not change any of the other default settings (see Figure 1–10).

Figure 1–10. Interval Timer MegaWizard


Interval Timer - timer

Interval Timer

Version 7.1

Timeout period

Period: 1 ms

Hardware options

Presets: Full-featured

Registers

☑ Writable period

☑ Readable snapshot

☑ Start/Stop control bits

Output signals

☐ Timeout pulse (1 clock wide)

☐ System reset on timeout (Watchdog)

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

It is a good habit to give memorable 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.

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

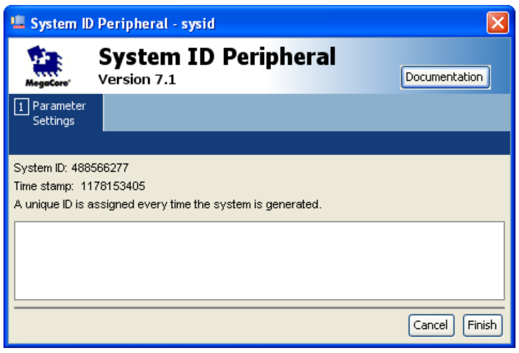
Add the System ID Peripheral

The system ID peripheral safeguards against accidentally downloading software compiled for a different Nios II system. 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 **Peripherals**, expand **Debug and Performance**, and then click **System ID Peripheral**.
2. Click **Add...**. The System ID Peripheral MegaWizard interface appears. The system ID peripheral has no user-configurable options (see Figure 1–11).

Figure 1–11. System ID Peripheral MegaWizard



System ID: 488566277
Time stamp: 1178153405
A unique ID is assigned every time the system is generated.

3. Click **Finish**. 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.

For further details on the system ID peripheral, see the *System ID Core* 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 eight PIO signals to drive LEDs on the board.

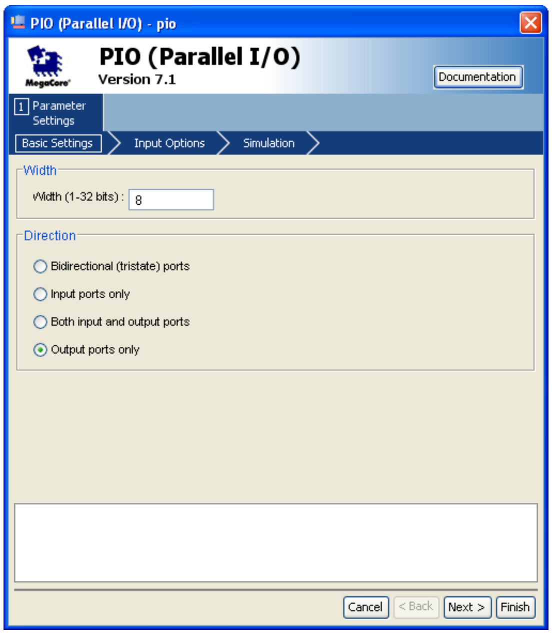
Perform the following steps to add the PIO. Perform these steps even if your target board doesn't have LEDs.

1. In the list of available components, expand **Peripherals**, expand **Microcontroller Peripherals**, and then click **PIO (Parallel I/O)**.

2. Click **Add**. The **PIO (Parallel I/O)** MegaWizard interface appears.

3. Do not change the default settings (see Figure 1–12). The MegaWizard interface defaults to an 8-bit output-only PIO, which exactly matches the needs for the example design.

Figure 1–12. PIO MegaWizard



Width

Width (1-32 bits) : 8

Direction

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

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

5. Right-click **pio** and click **Rename**.

6. Type `led_pio` and press **Enter**.

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

For further details on the PIO, see the *PIO Core* 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 component, 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 component 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 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:

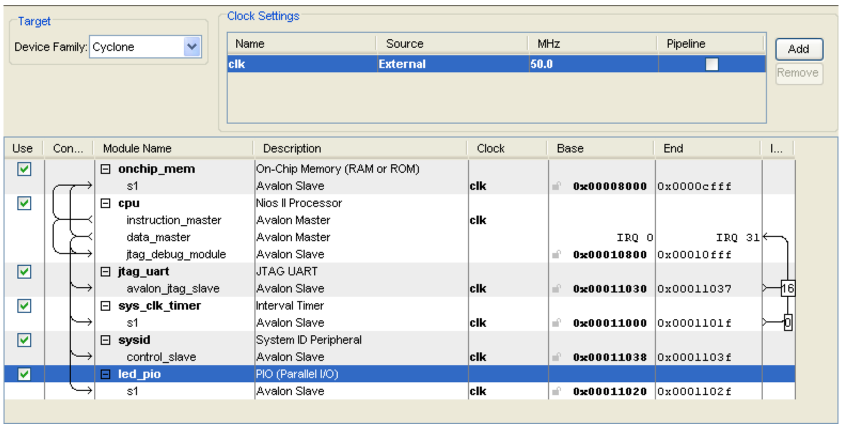
1. On the System menu, click **Auto-Assign Base Addresses** to make SOPC Builder assign functional base addresses to each component in the system. The **Base** and **End** values in the table of active components might change, reflecting the addresses that SOPC Builder reassigned.

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



Device Family: Cyclone

Clock Settings

Name	Source	MHz	Pipeline
clk	External	50.0	☐

Use	Con...	Module Name	Description	Clock	Base	End	I...
✓		onchip_mem	On-Chip Memory (RAM or ROM)				
		s1	Avalon Slave	clk	0x00008000	0x0000cfff	
✓		cpu	Nios II Processor				
		instruction_master	Avalon Master	clk			
		data_master	Avalon Master		IRQ 0	IRQ 31	
		jtag_debug_module	Avalon Slave		0x00010800	0x00010fff	
✓		jtag_uart	JTAG UART				
		avalon_jtag_slave	Avalon Slave	clk	0x00011030	0x00011037	16
✓		sys_clk_timer	Interval Timer				
		s1	Avalon Slave	clk	0x00011000	0x0001101f	0
✓		sysid	System ID Peripheral				
		control_slave	Avalon Slave	clk	0x00011038	0x0001103f	
✓		led_pio	PIO (Parallel I/O)				
		s1	Avalon Slave	clk	0x00011020	0x0001102f	

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 "Info: System generation was successful." (see Figure 1–14).

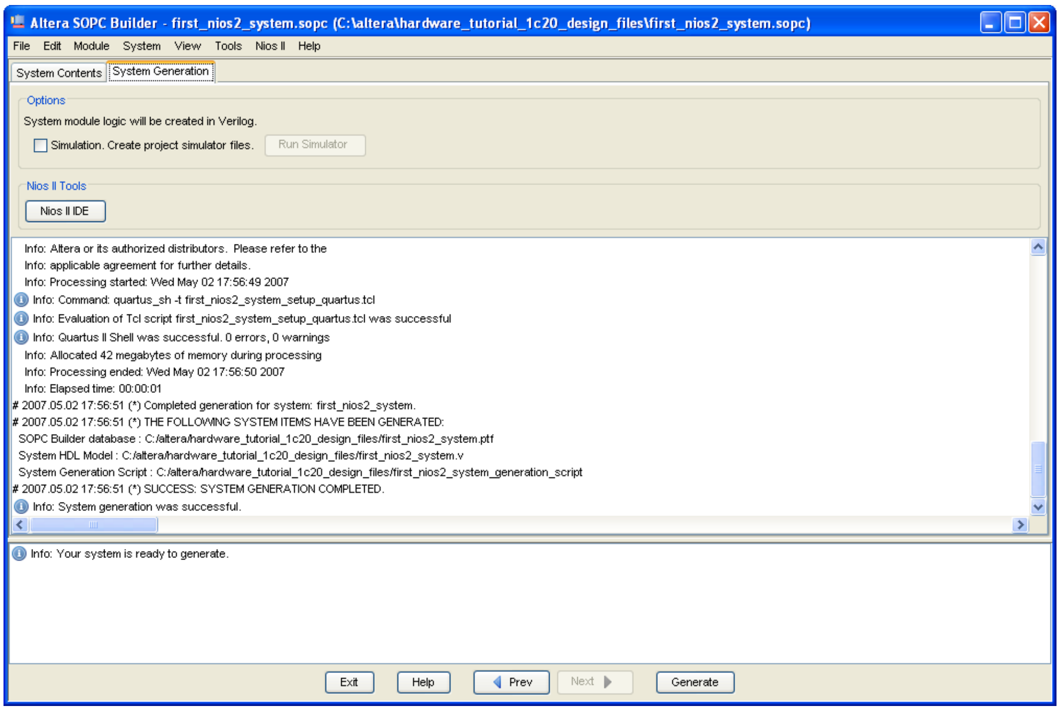


Figure 1–14. Successful System Generation

Altera SOPC Builder - first_nios2_system.sopc (C:\altera\hardware_tutorial_1c20_design_files\first_nios2_system.sopc)
File Edit Module System View Tools Nios II Help
System Contents
System Generation
Options
Nios II Tools
Nios II IDE
Run Simulator
Exit
Help
Prev
Next
Generate

System module logic will be created in Verilog.

☐ Simulation. Create project simulator files.

```
 Info: Altera or its authorized distributors.  Please refer to the
 Info: applicable agreement for further details.
 Info: Processing started: Wed May 02 17:56:49 2007
 Info: Command: quartus_sh -t first_nios2_system_setup_quartus.tcl
 Info: Evaluation of Tcl script first_nios2_system_setup_quartus.tcl was successful
 Info: Quartus II Shell was successful. 0 errors, 0 warnings
 Info: Allocated 42 megabytes of memory during processing
 Info: Processing ended: Wed May 02 17:56:50 2007
 Info: Elapsed time: 00:00:01
# 2007.05.02 17:56:51 (*) Completed generation for system: first_nios2_system.
# 2007.05.02 17:56:51 (*) THE FOLLOWING SYSTEM ITEMS HAVE BEEN GENERATED:
 SOPC Builder database : C:/altera/hardware_tutorial_1c20_design_files/first_nios2_system.ptf
 System HDL Model : C:/altera/hardware_tutorial_1c20_design_files/first_nios2_system.v
 System Generation Script : C:/altera/hardware_tutorial_1c20_design_files/first_nios2_system_generation_script
# 2007.05.02 17:56:51 (*) SUCCESS: SYSTEM GENERATION COMPLETED.
 Info: System generation was successful.
```

```
Info: Your system is ready to generate.
```

4. 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 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 systems, see *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 pins.
- Compile the Quartus II project.
- Verify timing.

You can skip ahead to "Develop Software Using the Nios II IDE" on page 1–34 if you do not have a target board. Alternatively, 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 *Introduction to the Quartus II Software* and the *Quartus II Handbook,* available at **www.altera.com/literature/lit-qts.jsp**.

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 method of design entry, so you instantiate a system module symbol **first_nios2_system** into the Block Diagram File.

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 Block Diagram File, 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 Block Diagram File 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-hand side of the Block Diagram File.

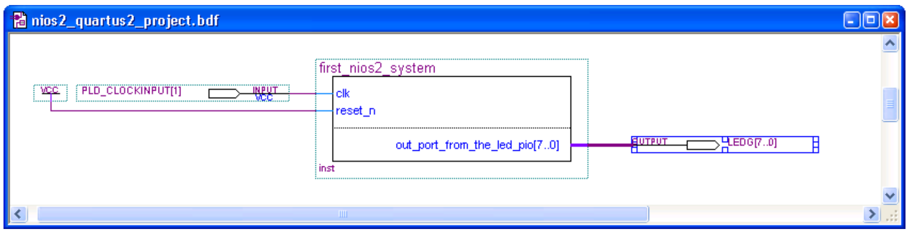
6. Click the left mouse button to drop the symbol in place.

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 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. Press **Delete**.

9. To save the completed Block Diagram 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 pinouts of your board.

☞ Skip ahead to section "Compile the Quartus II Project and Verify Timing" on page 1–32, if you are targeting a Nios development board. The provided Quartus II project files already contain appropriate assignments for Nios development boards.

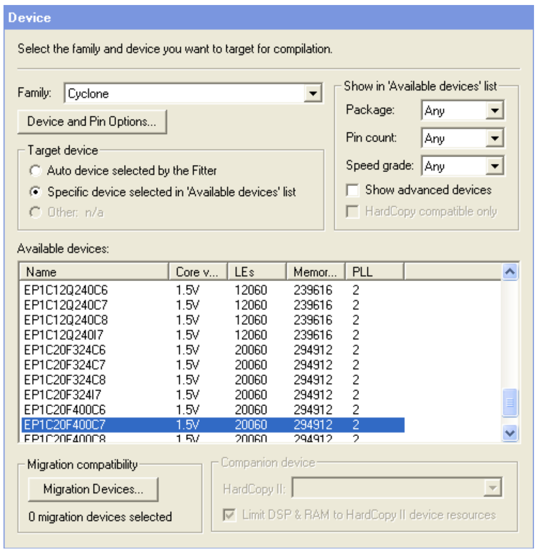
You must know the pin layout for the custom board to complete this section. You also must know other requirements for using the board, which are beyond the scope of this document. Refer to the documentation for your board.

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 1–16 shows an example of the **Settings** dialog box assigning a Cyclone device.

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



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

Family: Cyclone

Target device

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

Show in 'Available devices' list

Package: Any

Pin count: Any

Speed grade: Any

- [ ] Show advanced devices
- [ ] HardCopy compatible only

Available devices:

Name	Core v...	LEs	Memor...	PLL
EP1C12Q240C6	1.5V	12060	239616	2
EP1C12Q240C7	1.5V	12060	239616	2
EP1C12Q240C8	1.5V	12060	239616	2
EP1C12Q240I7	1.5V	12060	239616	2
EP1C20F324C6	1.5V	20060	294912	2
EP1C20F324C7	1.5V	20060	294912	2
EP1C20F324C8	1.5V	20060	294912	2
EP1C20F324I7	1.5V	20060	294912	2
EP1C20F400C6	1.5V	20060	294912	2
EP1C20F400C7	1.5V	20060	294912	2
EP1C20F400C8	1.5V	20060	294912	2

Migration compatibility

0 migration devices selected

Companion device

HardCopy II:

- [✓] Limit DSP & RAM to HardCopy II device resources

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

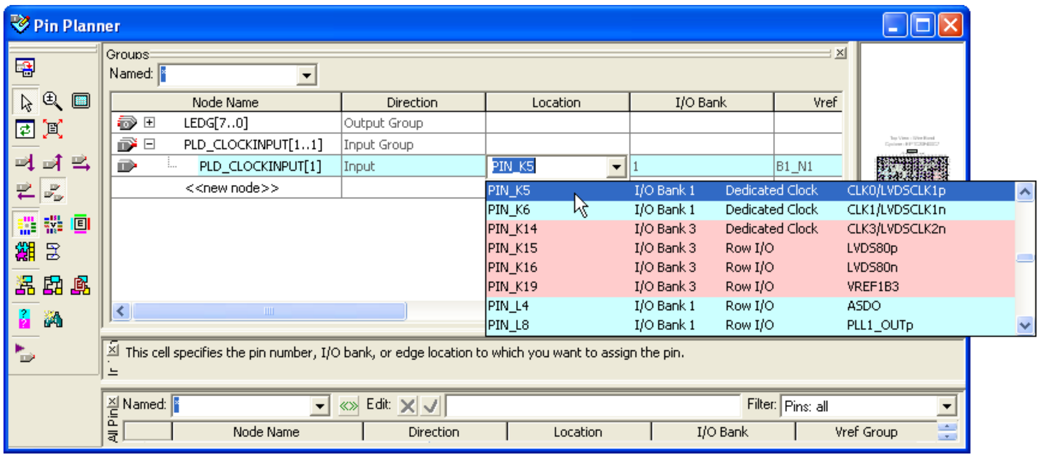
1. On the Assignments menu, click **Pins**. The Quartus II Pin Planner appears. The Quartus II project has many ready-made assignments appropriate for a Nios development board, which you must reassign to suit your board.

2. In the **Node Name** column, locate **PLD_CLOCKINPUT[1]**. You might need to expand the **PLD_CLOCKINPUT[1..1]** category to make **PLD_CLOCKINPUT[1]** visable.

3. In the **PLD_CLOCKINPUT[1]** row, double-click in the **Location** cell. A list of available pin locations appears.

4. Select the appropriate FPGA pin that connects to the oscillator on the board (see Figure 1–17).

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



Node Name	Direction	Location	I/O Bank	Vref
LEDG[7..0]	Output Group			
PLD_CLOCKINPUT[1..1]	Input Group			
PLD_CLOCKINPUT[1]	Input	PIN_K5	1	B1_N1
<<new node>>				

PIN_K5	I/O Bank 1	Dedicated Clock	CLK0/LVDSCLK1p
PIN_K6	I/O Bank 1	Dedicated Clock	CLK1/LVDSCLK1n
PIN_K14	I/O Bank 3	Dedicated Clock	CLK3/LVDSCLK2n
PIN_K15	I/O Bank 3	Row I/O	LVDS80p
PIN_K16	I/O Bank 3	Row I/O	LVDS80n
PIN_K19	I/O Bank 3	Row I/O	VREF1B3
PIN_L4	I/O Bank 1	Row I/O	ASDO
PIN_L8	I/O Bank 1	Row I/O	PLL1_OUTp

This cell specifies the pin number, I/O bank, or edge location to which you want to assign the pin.

	Node Name	Direction	Location	I/O Bank	Vref Group

5. If you connected the LED pins in the Board Design File schematic, repeat steps 2 to 4 with **LEDG[7..0]** 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]**.

6. On the File menu, click **Save** to save the assignments.

7. Close the Pin Planner.

CAUTION 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 enter a high-impedance state.
- The IO banks are configured for the 3.3V LVTTL I/O standard. The board must supply 3.3V to the FPGA's VCCIO pins.
- The **LEDG[7..0]** outputs drive 3.3V.

For further details on making assignments in the Quartus II software, see the *Quartus II Handbook Volume 2: Design Implementation and 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 SRAM Object File 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.

Perform the following steps:

1. 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_CLOCKINPUT[1]**. This is the maximum frequency (F_{MAX}) that this FPGA design is capable of running.

If the **Actual Time** frequency for **PLD_CLOCKINPUT[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 SRAM Object 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 and Synthesis*.

Download Hardware Design to Target FPGA

In this section you download the SRAM Object File 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.

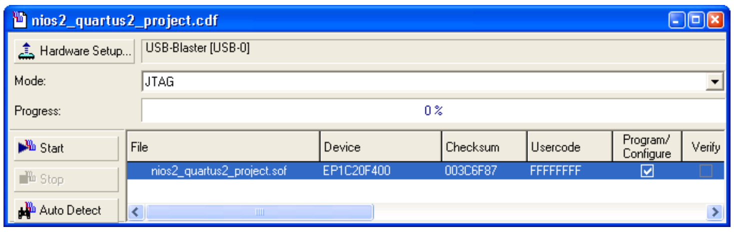
4. Select the appropriate download cable in the **Currently selected hardware** list. If the appropriate download cable does not appear in the list, you must first install drivers for the cable.

5. Click **Close**.

6. Turn on **Program/Configure** for **nios2_quartus2_project.sof** (see Figure 1–18).

7. Click **Start**. The **Progress** meter sweeps to 100% as the Quartus II software configures the FPGA.

Figure 1–18. Quartus II Programmer Window



Hardware Setup... USB-Blaster [USB-0]

Mode: JTAG

Progress: 0 %

File	Device	Checksum	Usercode	Program/ Configure	Verify
nios2_quartus2_project.sof	EP1C20F400	003C6F87	FFFFFFFF	☑	☐

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 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 perform the following actions:

- Create a new Nios II C/C++ application project (see page 1–35).
- Compile the project (see page 1–36).

To perform this section, you must have the SOPC Builder System File you created in "Define the System in SOPC Builder" on page 1–15.

Create a New Nios II C/C++ Application Project

In this section you 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** <*version*>. On Linux computers, run the executable file <*Nios II EDS install path*>**/bin/nios2-ide**.

2. If the **Workspace Launcher** dialog box appears, click **OK** to accept the default workspace location.

3. If you are not already in the Nios II C/C++ perspective, point to **Open Perspective** on the Window menu, and then either click **Nios II C/C++**, or click **Other** and then click **Nios II C/C++**.

4. On the File menu, point to **New**, and then click **Nios II C/C++ Application** to open the **New Project** wizard.

5. Click **Browse** under **Select Target Hardware**. The **Select Target Hardware** dialog box opens.

6. Browse to **<***Design Files Directory***>**.

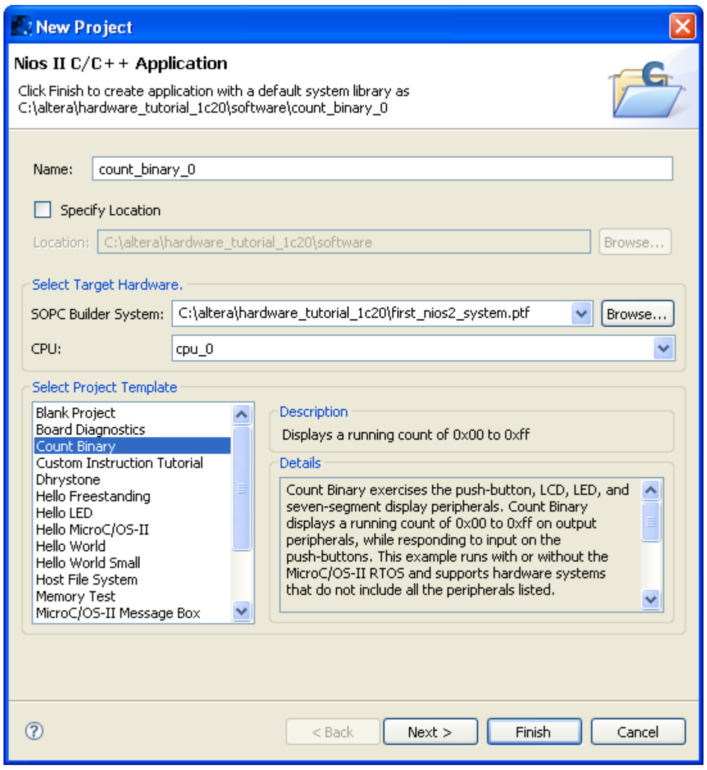
7. Select **first_nios2_system.ptf**.

8. Click **Open**. You return to the **New Project** wizard, and the **SOPC Builder System** and **CPU** fields are now filled in.

9. Select **Count Binary** in the **Select Project Template** list. The **Name** field automatically updates to **count_binary_0** (see Figure 1–19).

10. Click **Finish**.

Figure 1–19. Nios II IDE New Project Wizard



Click Finish to create application with a default system library as C:\altera\hardware_tutorial_1c20\software\count_binary_0

Name: count_binary_0

Specify Location

Location: C:\altera\hardware_tutorial_1c20\software

Select Target Hardware.

SOPC Builder System: C:\altera\hardware_tutorial_1c20\first_nios2_system.ptf

CPU: cpu_0

Select Project Template

Blank Project
Board Diagnostics
Count Binary
Custom Instruction Tutorial
Dhrystone
Hello Freestanding
Hello LED
Hello MicroC/OS-II
Hello World
Hello World Small
Host File System
Memory Test
MicroC/OS-II Message Box

Description

Displays a running count of 0x00 to 0xff

Details

Count Binary exercises the push-button, LCD, LED, and seven-segment display peripherals. Count Binary displays a running count of 0x00 to 0xff on output peripherals, while responding to input on the push-buttons. This example runs with or without the MicroC/OS-II RTOS and supports hardware systems that do not include all the peripherals listed.

The Nios II IDE creates and displays these new projects in the Nios II C/C++ Projects view on the left-hand side of the workbench:

- **count_binary_0** - Your C/C++ application project
- **count_binary_0_syslib** - A board support package that encapsulates the details of the Nios II system hardware
- **altera.components** - Links to source code for all Altera-provided components, for use during debug sessions

Compile the Project

In this section you 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** dialog box for **count_binary_0_syslib** opens.

2. Click the **System Library** page. 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 1–15.

3. Change the following settings to reduce the size of the compiled executable (see Figure 1–20).

   a. Turn on **Program never exits**.

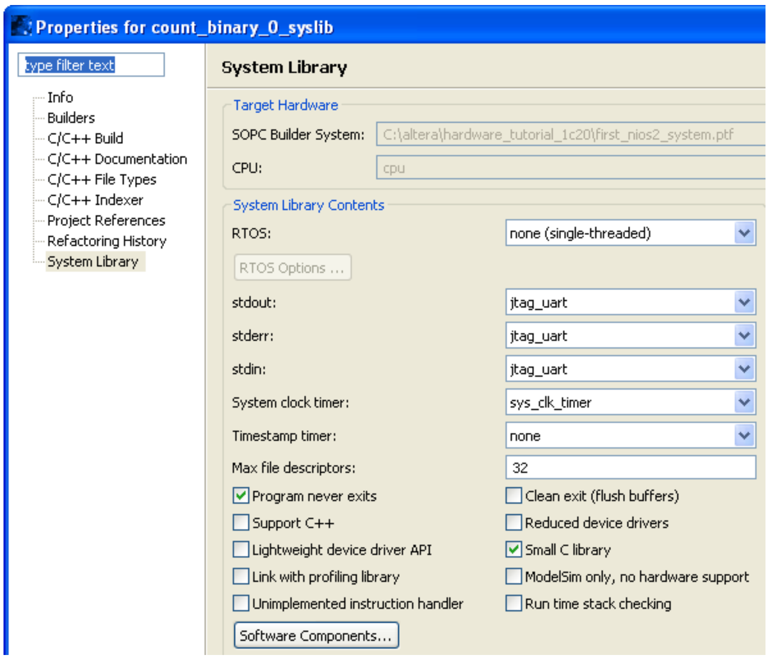
   b. Turn off **Support C++**.

   c. Turn off **Clean exit (flush buffers)**.

   d. Turn on **Small C library**.

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

Figure 1–20. System Library Properties



System Library

Target Hardware

SOPC Builder System: C:\altera\hardware_tutorial_1c20\first_nios2_system.ptf

CPU: cpu

System Library Contents

RTOS: none (single-threaded)

stdout: jtag_uart

stderr: jtag_uart

stdin: jtag_uart

System clock timer: sys_clk_timer

Timestamp timer: none

Max file descriptors: 32

- [✓] Program never exits
- [ ] Clean exit (flush buffers)
- [ ] Support C++
- [ ] Reduced device drivers
- [ ] Lightweight device driver API
- [✓] Small C library
- [ ] Link with profiling library
- [ ] ModelSim only, no hardware support
- [ ] Unimplemented instruction handler
- [ ] Run time stack checking

4. Click **OK** to close the **Properties** dialog box and return to the IDE workbench.

5. Right-click the **count_binary_0** project in the Nios II C/C++ Projects view and click **Build Project**.

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

Run the Program

In this section you 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.

Older versions of count_binary.h need a small modification to keep the code from hanging on some newer devices and terminating in the ISS. Be sure line 18 looks like this:

```
# define LCD_PRINTF(lcd, args...)  /* Do Nothing */
```

Run the Program on Target Hardware

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

> If you do not have a target board, skip ahead to "Run the Program on the ISS" on page 1–40. To proceed, you must have completed the steps in "Download Hardware Design to Target FPGA" on page 1–33.

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

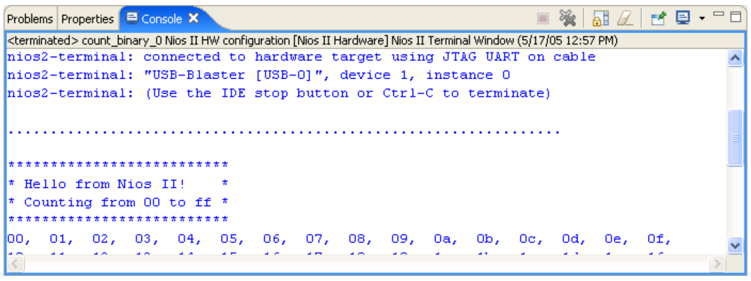
1. Right-click the **count_binary_0** project, point to **Run As**, and then click **Nios II Hardware**. The IDE downloads the program to the FPGA on the target board and starts execution.

   > If you get a dialog box warning that the IDE needs to finish indexing the **altera.components** project before you can proceed, wait a few seconds for it to finish. The source code is indexed for debug purposes.

   When the target hardware starts executing the program, the Console view displays character I/O output (see Figure 1–21). If you connected LEDs to the Nios II system in "Integrate the SOPC Builder System into the Quartus II Project" on page 1–28, 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 1–21. 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 run the **count_binary_0** program on the Nios II ISS. Perform the following steps:

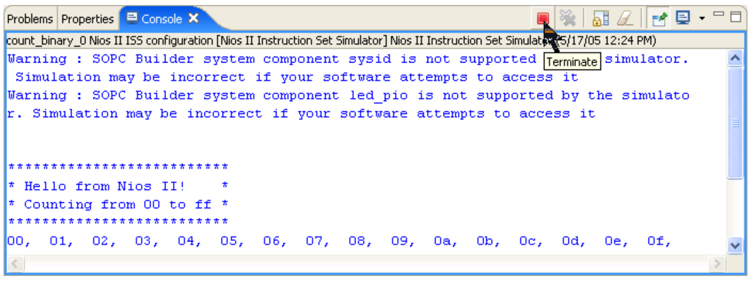
1. Right-click the **count_binary_0** project, point to **Run As**, and then click **Nios II Instruction Set Simulator**.

   If you get a dialog box warning that the IDE needs to finish indexing the **altera.components** project before you can proceed, wait a few seconds for it to finish. The source code is indexed for debug purposes.

   When the ISS starts executing the program, the Console view displays character I/O output from the program (see Figure 1–22). The count output appears very slowly because there are delay loops in the code.

2. 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 1–22. Console View Displaying Instruction Set Simulator Output



count_binary_0 Nios II ISS configuration [Nios II Instruction Set Simulator] Nios II Instruction Set Simulat (5/17/05 12:24 PM)

```
Warning : SOPC Builder system component sysid is not supported  simulator.
 Simulation may be incorrect if your software attempts to access it
Warning : SOPC Builder system component led_pio is not supported by the simulato
r. Simulation may be incorrect if your software attempts to access it


**************************
* 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 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 and 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 in the Nios II IDE help system – This tutorial teaches in detail how to use the Nios II IDE to develop, run, and debug new Nios II 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 4: SOPC Builder* – This volume provides complete reference on using SOPC Builder, including topics such as building memory subsystems and creating custom components.
- *Quartus II Handbook Volume 5: Embedded Peripherals* – This handbook contains details on the components provided free as part of the Nios II Embedded Design Suite.

For a complete list of all documents available for the Nios II processor, visit the Nios II literature page at **www.altera.com/literature/lit-nio2.jsp**.