


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

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

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

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

## Design Example

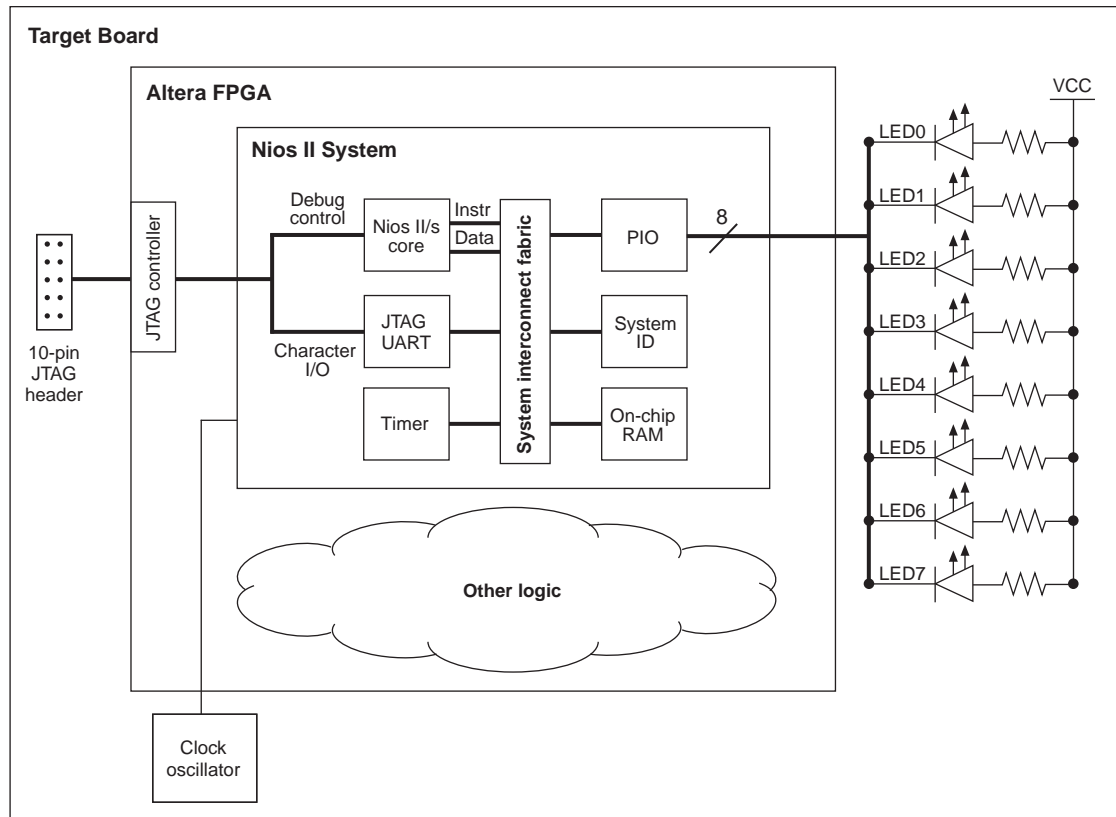
The design example you build in this tutorial demonstrates a small Nios II system for control applications, that displays character I/O output and blinks LEDs in a binary counting pattern. This Nios II system can also communicate with a host computer, allowing the host computer to control logic inside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

The example Nios II system contains the following components:

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

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

**Figure 1-1. Tutorial Design Example**

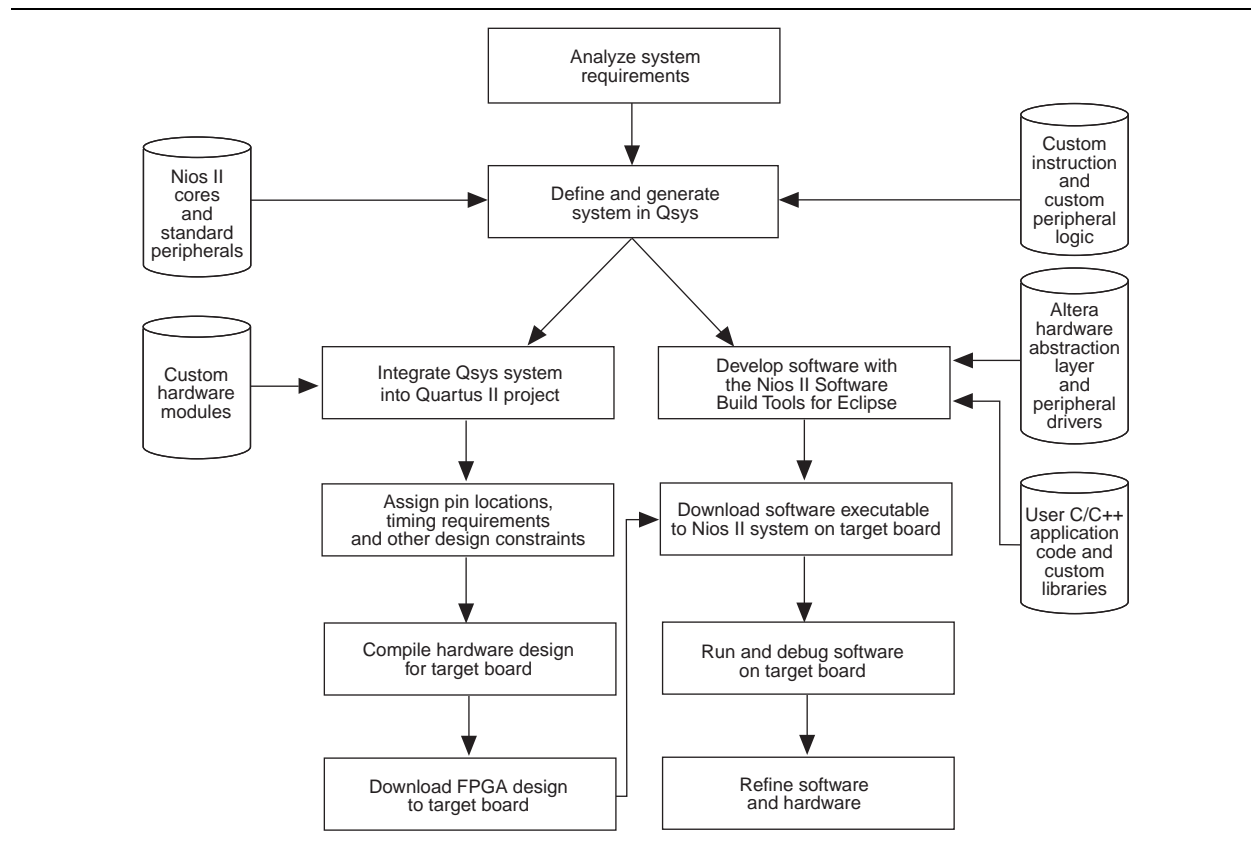


As shown in Figure 1-1, other logic can exist within the FPGA alongside the Nios II system. In fact, most FPGA designs with a Nios II system also include other logic. A Nios II system can interact with other on-chip logic, depending on the needs of the overall system. For the sake of simplicity, the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

## Nios II System Development Flow

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

**Figure 1-2. Nios II System Development Flow**



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

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



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

## Analyzing System Requirements

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

- What computational performance does the application require?
- How much bandwidth or throughput does the application require?
- What types of interfaces does the application require?
- Does the application require multithreaded software?

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

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

Analyzing these topics involve both the hardware and software teams.

## Defining and Generating the System in Qsys

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

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

The primary outputs of Qsys are the following file types:

- Qsys Design File (**.qsys**)—Contains the hardware contents of the Qsys system.
- SOPC Information File (**.sopcinfo**)—Contains a description of the contents of the **.qsys** file in Extensible Markup Language File (**.xml**) format. The Nios II EDS uses the **.sopcinfo** file to create software for the target hardware.
- Hardware description language (HDL) files—Are the hardware design files that describe the Qsys system. The Quartus II software uses the HDL files to compile the overall FPGA design into an SRAM Object File (**.sof**).



For more information about the following topics, refer to the related documentation:

- For Nios II processor cores, refer to the *Nios II Processor Reference Handbook*.
- For Qsys and developing custom components, refer to the *System Design with Qsys* section of *Volume 1: Design and Synthesis* of the *Quartus II Handbook*.
- For custom instructions, refer to the *Nios II Custom Instruction User Guide*.

## Integrating the Qsys System into the Quartus II Project

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

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

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

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



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

## Developing Software with the Nios II Software Build Tools for Eclipse

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

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

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

- **system.h** file—Defines symbols for referencing the hardware in the system. The Nios II SBT for Eclipse automatically create this file when you create a new board support package (BSP).
- Executable and Linking Format File (**.elf**)—Is the result of compiling a C/C++ application project, that you can download directly to the Nios II processor.

- Hexadecimal (Intel-Format) File (.hex)—Contains initialization information for on-chip memories. The Nios II SBT for Eclipse generate these initialization files for on-chip memories that support initialization of contents.
- Flash memory programming data—Is boot code and other arbitrary data you might write to flash memory. The Nios II SBT for Eclipse includes a flash programmer, which allows you to write your program to flash memory. The flash programmer adds appropriate boot code to allow your program to boot from flash memory. You can also use the flash programmer to write arbitrary data to flash memory.

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



For extensive information about developing software for the Nios II processor, refer to the *Nios II Software Developer's Handbook*.

## Running and Debugging Software on the Target Board

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



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

## Varying the Development Flow

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

### Refining the Software and Hardware

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

### Iteratively Creating a Nios II System

A common technique for building a complex Nios II system is to start with a simpler Qsys 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.

# Analyze System Requirements

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

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

These goals lead to the following design decisions:

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



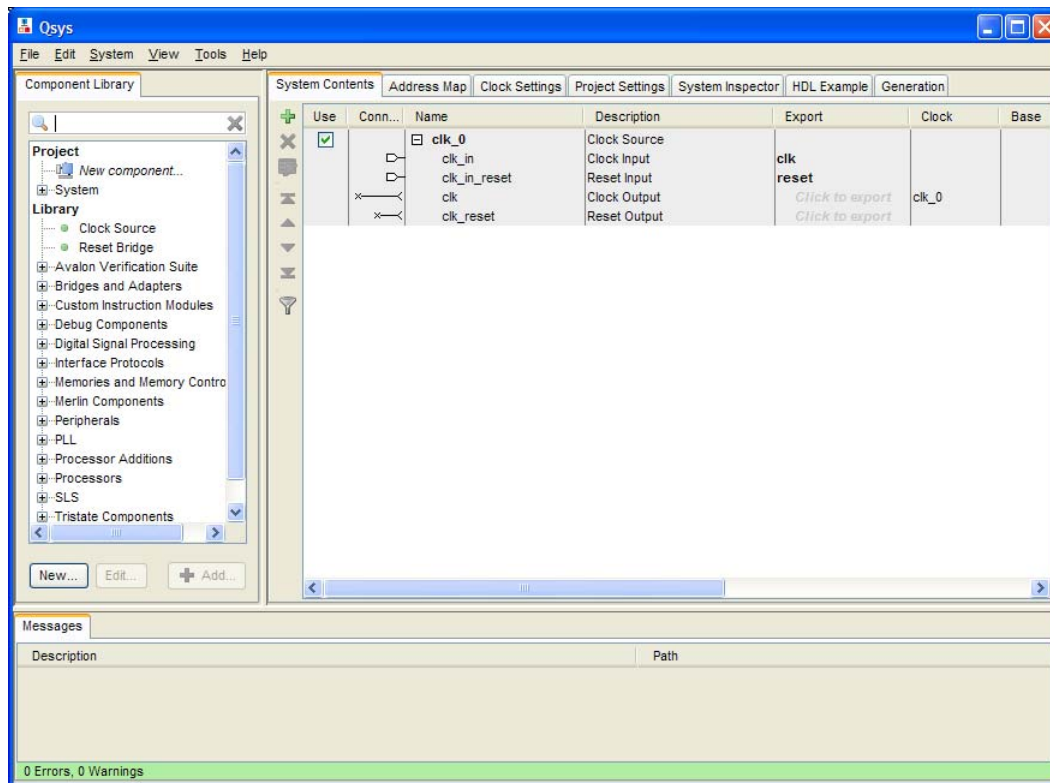
For more information about these and other components, refer to the *Embedded Peripherals IP User Guide*.

- Open the activity's template
- This is the same template as Lab1. Open in Quartus the Quartus Project Basic\_Organ\_solution.qpf
- Open the Qsys environment via the Tools->Qsys
- If a dialog box open, press "OK"
- Note that your FPGA device on the instructions in the next page should be automatically OK. Also, the 50 MHz default for the clock in new Qsys files (discussed in the next page) matches your development board so that is OK too.



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

Figure 1-4. Qsys GUI



## Define the System in Qsys

You use Qsys 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. Qsys 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 Qsys system.

The Qsys 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 Qsys design steps in a different order.

## Specify Target FPGA and Clock Settings

The **Clock Settings** and the **Project Settings** tabs specify the Qsys system's relationship to other devices in the system.

Perform the following steps:

1. On the **Project Settings** tab, select the **Device Family** that matches the Altera FPGA you are targeting.



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

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



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

3. On the **Clock Settings** tab, double-click the clock frequency in the **MHz** column for `clk_0`. `clk_0` is the default clock input name for the Qsys system. The frequency you specify for `clk_0` must match the oscillator that drives the FPGA.
4. Type the clock frequency and press Enter.

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

### Add the On-Chip Memory

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

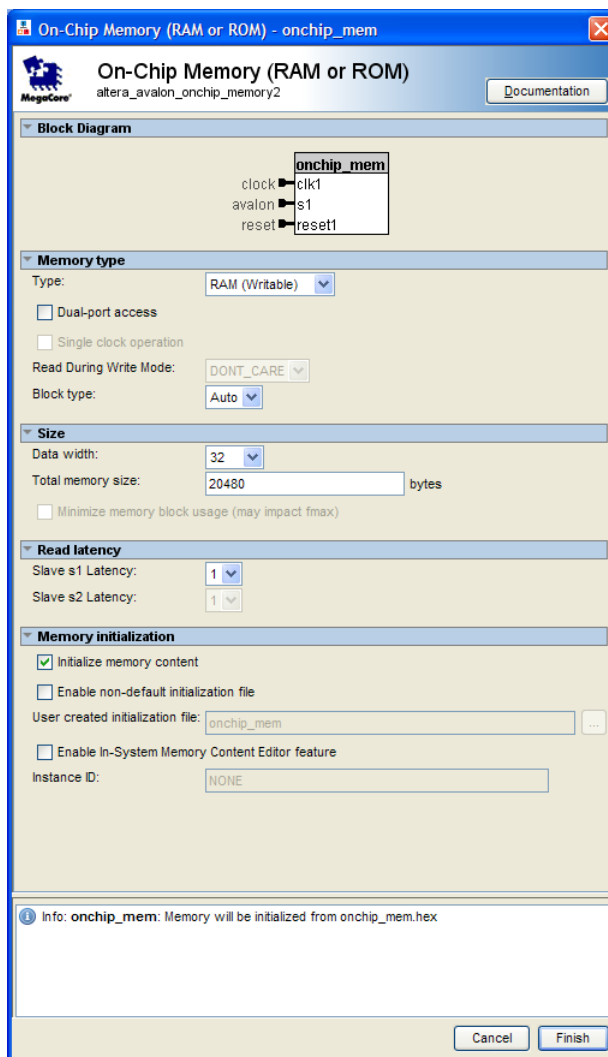
1. On the **Component Library** tab (to the left of the **System Contents** tab), expand **Memories and Memory Controllers**, expand **On-Chip**, and then click **On-Chip Memory (RAM or ROM)**.
2. Click **Add**. The On-Chip Memory (RAM or ROM) parameter editor appears. [Figure 1-5](#) shows the GUI.
3. In the **Block type** list, select **Auto**.
4. In the **Total memory size** box, type 20480 to specify a memory size of 20 KB.



Do not change any of the other default settings.

Figure 1-5 shows the On-Chip Memory (RAM or ROM) parameter editor.

**Figure 1-5. On-Chip Memory Parameter Editor**



5. Click **Finish**. You return to Qsys.
6. Click the **System Contents** tab. An instance of the on-chip memory appears in the system contents table.



For more information about on-chip memory, you can click **Documentation** in the On-Chip Memory (RAM or ROM) parameter editor. This documentation feature is available in the parameter editor for each component.

7. In the **Name** column of the system contents table, right-click the on-chip memory and click **Rename**.
8. Type `onchip_mem` and press Enter.



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

## Add the Nios II Processor Core

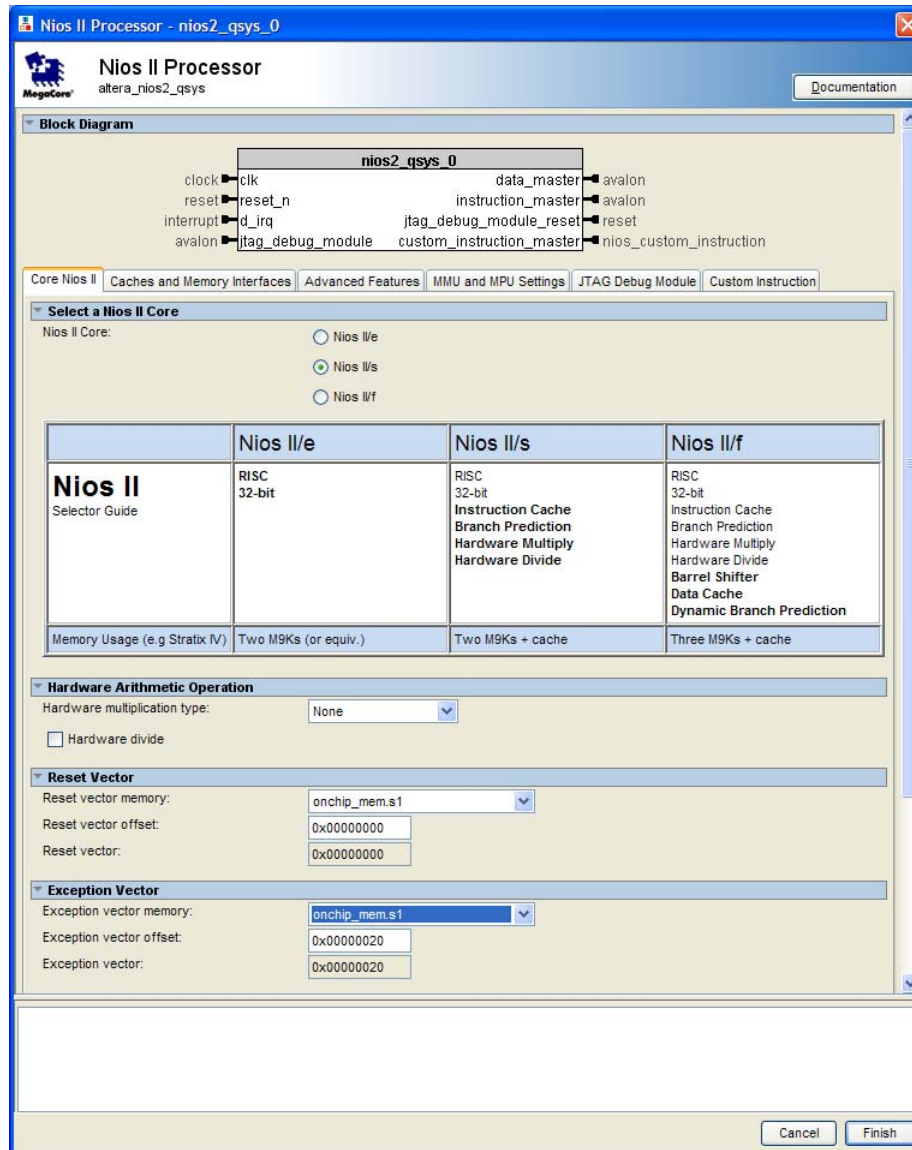
In this section you add the Nios II/s core and configure it to use 2 KB of on-chip instruction cache memory. For educational purposes, the tutorial design example uses the Nios II/s (standard) core, which provides a balanced trade-off between performance and resource utilization. In reality, the Nios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

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

1. On the **Component Library** tab, expand **Processors**, and then click **Nios II Processor**.
2. Click **Add**. The Nios II Processor parameter editor appears, displaying the **Core Nios II** tab. [Figure 1-6](#) shows the GUI.
3. Under **Select a Nios II core**, select **Nios II/s**.
4. In the **Hardware multiplication type** list, select **None**.
5. Turn off **Hardware divide**.
6. Click **Finish**. You return to the **Qsys System Contents** tab, and an instance of the Nios II core appears in the system contents table. Ignore the exception and reset vector error messages. You resolve these errors in steps [13](#) and [14](#).
7. In the **Name** column, right-click the Nios II processor and click **Rename**.
8. Type `cpu` and press Enter.
9. In the **Connections** column, connect the `clk` port of the `clk_0` clock source to both the `clk1` port of the on-chip memory and the `clk` port of the Nios II processor by clicking the hollow dots on the connection line. The dots become solid indicating the ports are connected.
10. Connect the `clk_reset` port of the `clk_0` clock source to both the `reset1` port of the on-chip memory and the `reset_n` port of the Nios II processor.
11. Connect the `s1` port of the on-chip memory to both the `data_master` port and `instruction_master` port of the Nios II processor.
12. Double-click the Nios II processor row of the system contents table to reopen the Nios II Processor parameter editor.
13. Under **Reset Vector**, select `onchip_mem.s1` in the **Reset vector memory** list and type `0x0` in the **Reset vector offset** box.
14. Under **Exception Vector**, select `onchip_mem.s1` in the **Exception vector memory** list and type `0x20` in the **Exception vector offset** box.

Figure 1-6 shows the Core Nios II tab of the Nios II Processor parameter editor.

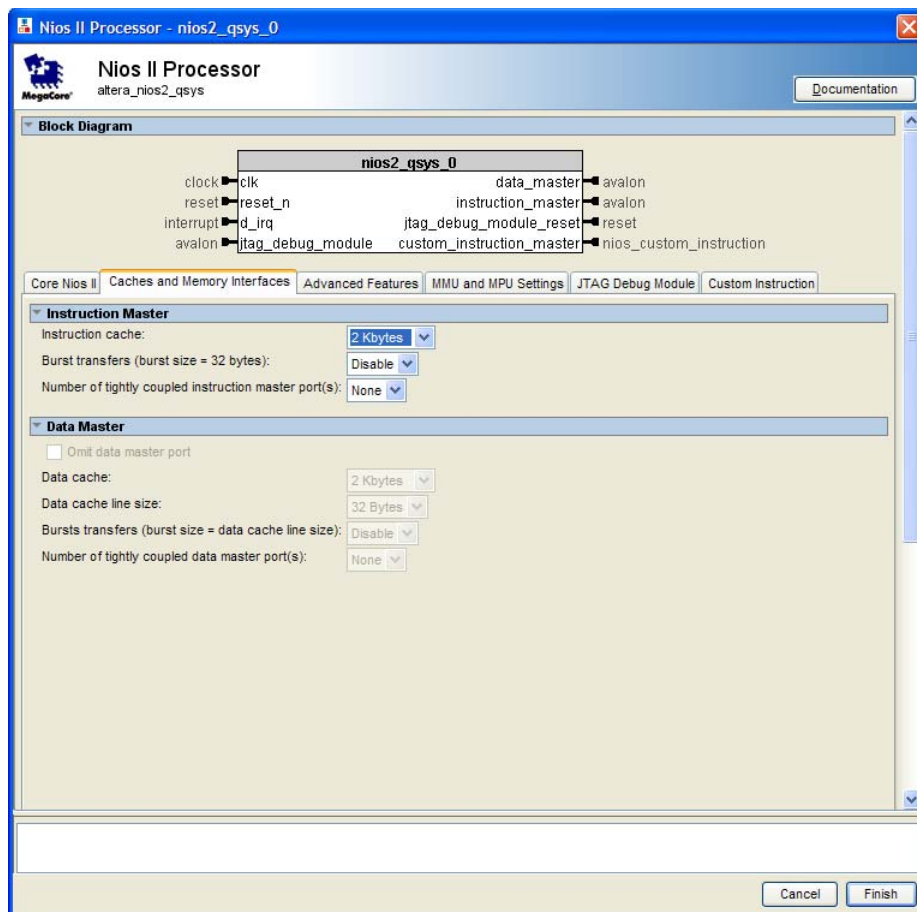
**Figure 1-6. Nios II Parameter Editor – Core Nios II Tab**



15. Click the **Caches and Memory Interfaces** tab. Figure 1-7 shows the GUI.
16. In the **Instruction cache** list, select **2 Kbytes**.
17. In the **Burst transfers** list, select **Disable**.
18. In the **Number of tightly coupled instruction master port(s)** list, select **None**.



Figure 1-7 shows the Caches and Memory Interfaces tab of the Nios II Processor parameter editor.

**Figure 1-7. Nios II Parameter Editor – Caches and Memory Interfaces Tab**



 Do not change any settings on the **Advanced Features**, **MMU and MPU Settings**, **JTAG Debug Module**, or **Custom Instruction** tabs.

19. Click **Finish**. You return to the Qsys **System Contents** tab.

-  For more information about configuring the Nios II core, refer to the *Instantiating the Nios II Processor in Qsys* chapter of the *Nios II Processor Reference Handbook*.
-  For more information about connecting memory to Nios II systems, refer to the *System Design with Qsys* section of *Volume 1: Design and Synthesis* of the *Quartus II Handbook*.

## Add the JTAG UART

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

1. On the **Component Library** tab, expand **Interface Protocols**, expand **Serial**, and then click **JTAG UART**.
2. Click **Add**. The JTAG UART parameter editor appears.


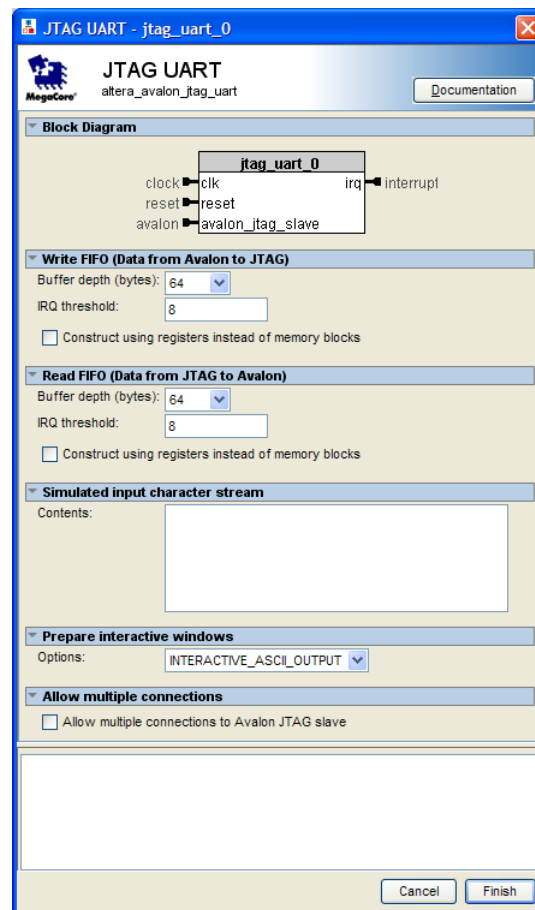
 Do not change the default settings.

Figure 1-8 shows the JTAG UART parameter editor.

Figure 1-8. JTAG UART Parameter Editor



3. Click **Finish**. You return to the Qsys **System Contents** tab, and an instance of the JTAG UART appears in the system contents table.
4. In the **Name** column, right-click the JTAG UART and click **Rename**.
5. Type `jtag_uart` and press Enter.
6. Connect the `clk` port of the `clk_0` clock source to the `clk` port of the JTAG UART.

7. Connect the `clk_reset` port of the `clk_0` clock source to the `reset` port of the JTAG UART.
8. Connect the `data_master` port of the Nios II processor to the `avalan_jtag_slave` port of the JTAG UART.



The `instruction_master` port of the Nios II processor does not connect to the JTAG UART because the JTAG UART is not a memory device and cannot send instructions to the Nios II processor.



For more information about the JTAG UART, refer to the *JTAG UART Core* chapter in the *Embedded Peripherals IP User Guide*.

### 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. On the **Component Library** tab, expand **Peripherals**, expand **Microcontroller Peripherals**, and then click **Interval Timer**.
2. Click **Add**. The Interval Timer parameter editor appears. [Figure 1-9](#) shows the GUI.
3. In the **Presets** list, select **Full-featured**.

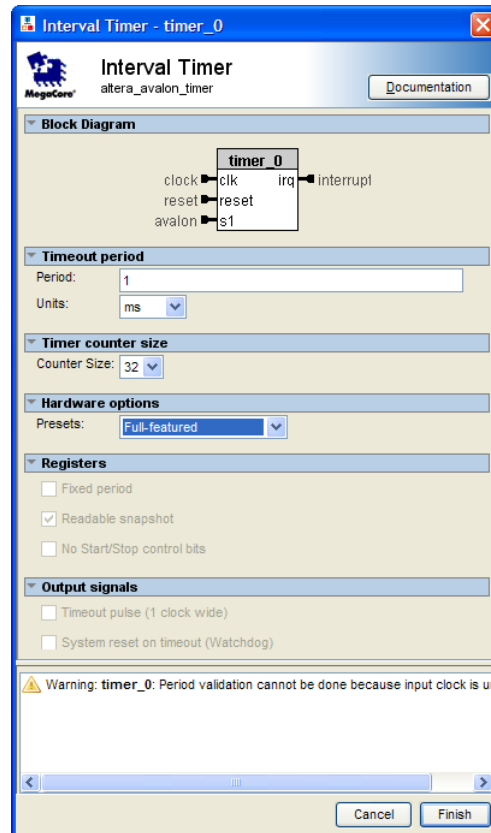


Do not change any of the other default settings.




Figure 1-9 shows the Interval Timer parameter editor.

**Figure 1-9. Interval Timer Parameter Editor**



4. Click **Finish**. You return to the Qsys **System Contents** tab, and an instance of the interval timer appears in the system contents table.
5. In the **Name** column, right-click the interval timer and click **Rename**.
6. Type `sys_clk_timer` and press Enter.
7. Connect the `clk` port of the `clk_0` clock source to the `clk` port of the interval timer.
8. Connect the `clk_reset` port of the `clk_0` clock source to the `reset` port of the interval timer.
9. Connect the `data_master` port of the Nios II processor to the `s1` port of the interval timer.

 For more information about the timer, refer to the *Timer Core* chapter in the *Embedded Peripherals IP User Guide*.

## 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 SBT for Eclipse can prevent you from downloading programs compiled for a different system.

Perform the following steps to add the system ID peripheral:

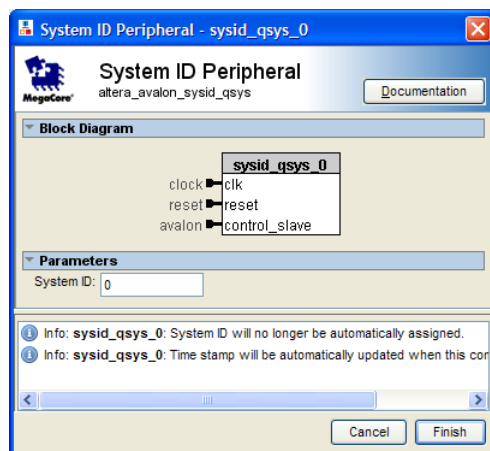
1. On the **Component Library** tab, expand **Peripherals**, expand **Debug and Performance**, and then click **System ID Peripheral**.
2. Click **Add**. The System ID Peripheral parameter editor appears.



Do not change the default setting.

Figure 1–10 shows the System ID Peripheral parameter editor.

**Figure 1–10. System ID Peripheral Parameter Editor**



3. Click **Finish**. You return to the Qsys **System Contents** tab, and an instance of the system ID peripheral appears in the system contents table.
4. In the **Name** column, right-click the system ID peripheral and click **Rename**.
5. Type `sysid` and press Enter.
6. Connect the `clk` port of the `clk_0` clock source to the `clk` port of the system ID peripheral.
7. Connect the `clk_reset` port of the `clk_0` clock source to the `reset` port of the system ID peripheral.
8. Connect the `data_master` port of the Nios II processor to the `control_slave` port of the system ID peripheral.



For more information about the system ID peripheral, refer to the *System ID Core* chapter in the *Embedded Peripherals IP User Guide*.

## Add the PIO

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

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

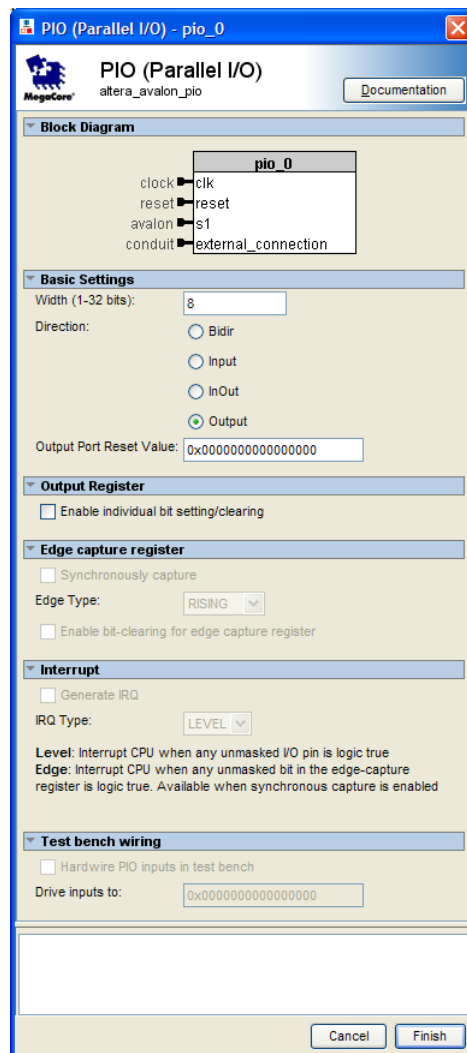
1. On the **Component Library** tab, expand **Peripherals**, expand **Microcontroller Peripherals**, and then click **PIO (Parallel I/O)**.
2. Click **Add**. The PIO (Parallel I/O) parameter editor appears. Figure 1-11 shows the GUI.



Do not change the default settings. The parameter editor defaults to an 8-bit output-only PIO, which exactly matches the needs for the design example.

Figure 1-11 shows the PIO (Parallel I/O) parameter editor.

**Figure 1-11. PIO Parameter Editor**



3. Click **Finish**. You return to the Qsys **System Contents** tab, and an instance of the PIO appears in the system contents table.
4. In the **Name** column, right-click the PIO and click **Rename**.
5. Type `led_pio` and press Enter.
6. Connect the `clk` port of the `clk_0` clock source to the `clk` port of the PIO.

7. Connect the `clk_reset` port of the `clk_0` clock source to the `reset` port of the PIO.
8. Connect the `data_master` port of the Nios II processor to the `s1` port of the PIO.
9. In the `external_connection` row, click **Click to export** in the **Export** column to export the PIO ports.



For more information about the PIO, refer to the *PIO Core* chapter in the *Embedded Peripherals IP User Guide*.

## Specify Base Addresses and Interrupt Request Priorities

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

Qsys provides the **Assign Base Addresses** command which makes assigning component base addresses easy. For many systems, including this design example, **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.
- Qsys 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.

Qsys also provides an **Assign Interrupt Numbers** command which connects IRQ signals to produce valid hardware results. However, assigning IRQs effectively requires an understanding of how software responds to them. Because Qsys does not know the software behavior, Qsys 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 **Assign Base Addresses** to make Qsys assign functional base addresses to each component in the system. Values in the **Base** and **End** columns might change, reflecting the addresses that Qsys reassigned.
2. In the **IRQ** column, connect the Nios II processor to the JTAG UART and interval timer.
3. Click the IRQ value for the `jtag_uart` component to select it.
4. Type 16 and press Enter to assign a new IRQ value.

5. Click the IRQ value for the **sys\_clk\_timer** component to select it.
6. Type 1 and press Enter to assign a new IRQ value.

Figure 1–12 shows the Qsys **System Contents** tab with the complete system.

**Figure 1–12. System Contents Tab with Complete System**

Use	Connections	Name	Description	Export	Clock	Base	End	IRQ
<input checked="" type="checkbox"/>		<b>clk_0</b>	Clock Source					
		clk_in	Clock Input	clk				
		clk_in_reset	Reset Input	reset				
		clk	Clock Output	Click to export	clk_0			
		clk_reset	Reset Output	Click to export				
<input checked="" type="checkbox"/>		<b>onchip_mem</b>	On-Chip Memory (RAM or ROM)					
		clk1	Clock Input	Click to export	clk_0			
		s1	Avalon Memory Mapped Slave	Click to export	[clk1]	0x00008000	0x0000cfff	
		reset1	Reset Input	Click to export				
<input checked="" type="checkbox"/>		<b>cpu</b>	Nios II Processor					
		clk	Clock Input	Click to export	clk_0			
		reset_n	Reset Input	Click to export	[clk]			
		data_master	Avalon Memory Mapped Master	Click to export	[clk]			
		instruction_master	Avalon Memory Mapped Master	Click to export	[clk]			
		jtag_debug_module_re...	Reset Output	Click to export	[clk]			
		jtag_debug_module	Avalon Memory Mapped Slave	Click to export	[clk]			
		custom_instruction_m...	Custom Instruction Master	Click to export	[clk]			
<input checked="" type="checkbox"/>		<b>jtag_uart</b>	JTAG UART					
		clk	Clock Input	Click to export	clk_0			
		reset	Reset Input	Click to export	[clk]			
		avalon_jtag_slave	Avalon Memory Mapped Slave	Click to export	[clk]	0x00011030	0x00011037	
<input checked="" type="checkbox"/>		<b>sys_clk_timer</b>	Interval Timer					
		clk	Clock Input	Click to export	clk_0			
		reset	Reset Input	Click to export	[clk]			
		s1	Avalon Memory Mapped Slave	Click to export	[clk]	0x00011000	0x0001101f	
<input checked="" type="checkbox"/>		<b>sysid</b>	System ID Peripheral					
		clk	Clock Input	Click to export	clk_0			
		reset	Reset Input	Click to export	[clk]			
		control_slave	Avalon Memory Mapped Slave	Click to export	[clk]	0x00011038	0x0001103f	
<input checked="" type="checkbox"/>		<b>led_pio</b>	PIO (Parallel I/O)					
		clk	Clock Input	Click to export	clk_0			
		reset	Reset Input	Click to export	[clk]			
		s1	Avalon Memory Mapped Slave	Click to export	[clk]	0x00011020	0x0001102f	
		external_connection	Conduit Endpoint	led_pio_external_...				

## Generate the Qsys System

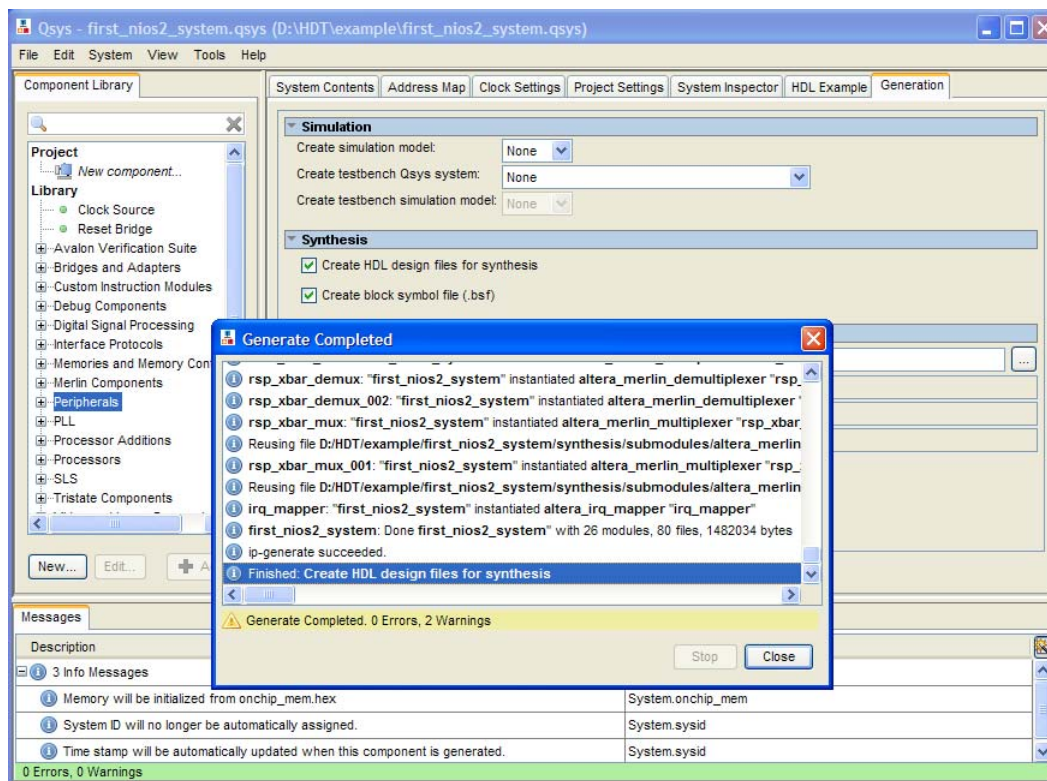
You are now ready to generate the Qsys system. Perform the following steps:

1. Click the **Generation** tab.
2. Select **None** in both the **Create simulation model** and **Create testbench Qsys system** lists. Because this tutorial does not cover the hardware simulation flow, you can select these settings to shorten generation time.
3. Click **Generate**. The **Save changes?** dialog box appears, prompting you to save your design.

4. Type `first_nios2_system` in the **File name** box and click **Save**. The **Generate** dialog box appears and system generation process begins.


The generation process can take several minutes. Output messages appear as generation progresses. When generation completes, the final "Info: Finished: Create HDL design files for synthesis" message appears. Figure 1-13 shows the successful system generation.

**Figure 1-13. Successful System Generation**



5. Click **Close** to close the dialog box.
6. On the File menu, click **Exit** to close Qsys and 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 SBT for Eclipse to develop software.

 For more information about generating systems with Qsys, refer to the *System Design with Qsys* section of *Volume 1: Design and Synthesis* of the *Quartus II Handbook*. For information about hardware simulation for Nios II systems, refer to *Simulating Nios II Embedded Processor Designs*.

## Integrate the Qsys System into the Quartus II Project

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

- Instantiate the Qsys system module in the Quartus II project.
- Assign FPGA device and pin locations.

## Add IP Variation File

In this section, you add the Quartus II IP File (**.qip**) to the your Quartus II project. To add the **.qip**, perform the following steps:

1. On the Assignments menu, click **Settings**. The **Settings** dialog box appears.
2. Under **Category**, click **Files**. The **Files** page appears.
3. Next to **File name**, click the browse (...) button.
4. In the **Files of type** list, select **Script Files (\*.tcl, \*.sdc, \*.qip)**.
5. Browse to locate *<design files directory>/first\_nios2\_system/synthesis/first\_nios2\_system.qip* and click **Open** to select the file.
6. Click **Add** to include **first\_nios2\_system.qip** in the project.
7. Click **OK** to close the **Settings** dialog box.

- Add the Qsys file to your project. The Verilog top level of the Qsys system is found in the file:

```
first_nios2_system/synthesis/first_nios2_system.v
```

You can look inside that file and understand what was auto-generated by the Qsys tool (further submodules are in the first\_nios2\_system/synthesis/submodules directory).

The module declaration of the file first\_nios2\_system.v should look something like:

```
module first_nios2_system (
    input wire      clk_clk,
    input wire      reset_reset_n,
    output wire [7:0] led_pio_external_connection_export
);
```

Instantiate this module inside the top-level file of your template (Basic\_Organ\_Solution.v) and connect clk\_clk to your 50 MHz clock, reset\_reset\_n to "1'b1", and led\_pio\_external\_connection\_export to the red LEDs 7:0 (LEDR[7:0]).

It should look something like this:

```
first_nios2_system
first_nios2_system_inst (
    .clk_clk(CLK_50M),
    .reset_reset_n(1'b1),
    .led_pio_external_connection_export(LEDR[7:0])
);
```



## **Compile the Quartus II Project and Verify Timing**

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

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

## Develop Software Using the Nios II SBT for Eclipse

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



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

In this section, you perform the following actions:

- Create new Nios II C/C++ application and BSP projects.
- Compile the projects.

To perform this section, you must have the `.sopcinfo` file you created in “*Define the System in Qsys*” on page 1–11.

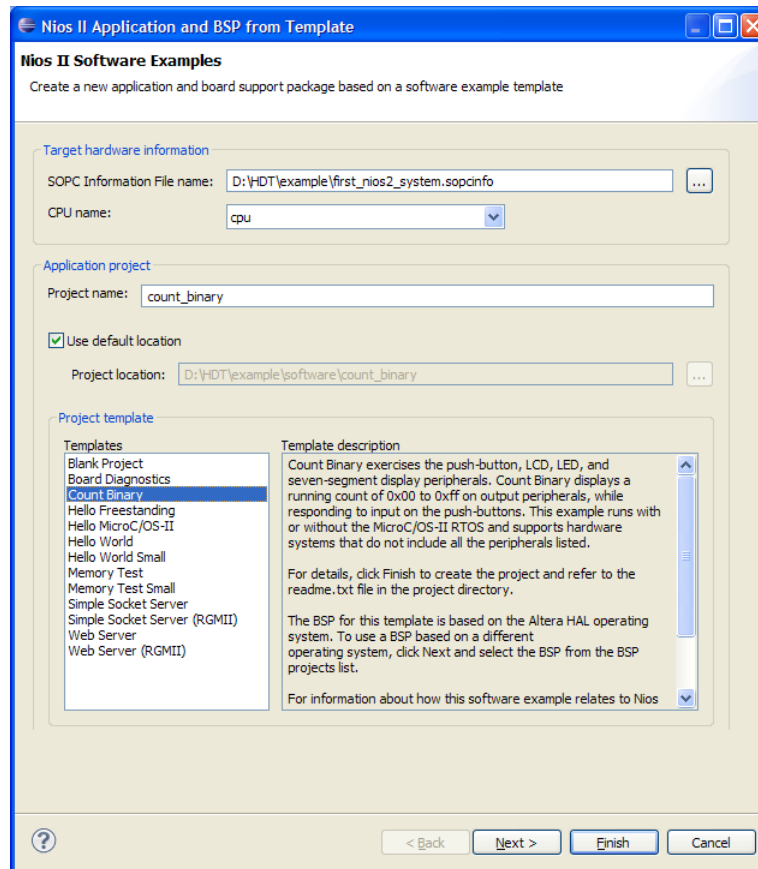
### Create a New Nios II Application and BSP from Template

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

1. Start the Nios II SBT for Eclipse. On Windows computers, click **Start**, point to **Programs**, **Altera**, **Nios II EDS <version>**, and then click **Nios II <version> Software Build Tools for Eclipse**. On Linux computers, run the executable file `<Nios II EDS install path>/bin/eclipse-nios2`.
2. If the **Workspace Launcher** dialog box appears, click **OK** to accept the default workspace location.
3. On the Window menu, point to **Open Perspective**, and then either click **Nios II**, or click **Other** and then click **Nios II** to ensure you are using the Nios II perspective.

- On the File menu, point to **New**, and then click **Nios II Application and BSP from Template**. The Nios II Application and BSP from Template wizard appears.  
Figure 1–20 shows the GUI.

**Figure 1–20. Nios II Application and BSP from Template Wizard**



- Under **Target hardware information**, next to **SOPC Information File name**, browse to locate the *<design files directory>*.
- Select **first\_nios2\_system.sopcinfo** and click **Open**. You return to the Nios II Application and BSP from Template wizard showing current information for the **SOPC Information File name** and **CPU name** fields.
- In the **Project name** box, type `count_binary`.
- In the **Templates** list, select **Count Binary**.
- Click **Finish**.

The Nios II SBT for Eclipse creates and displays the following new projects in the Project Explorer view, typically on the left side of the window:

- count\_binary**—Your C/C++ application project
- count\_binary\_bsp**—A board support package that encapsulates the details of the Nios II system hardware

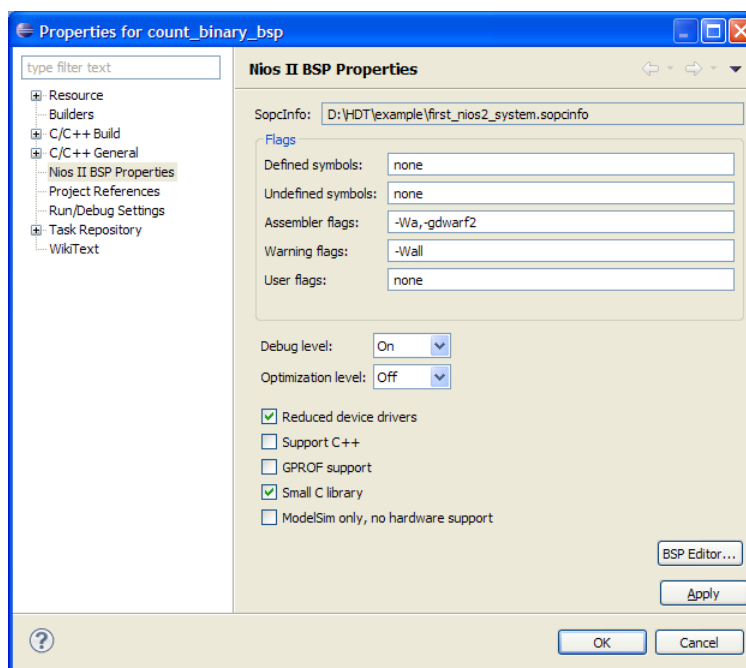
## Compile the Project

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

Perform the following steps:

1. In the Project Explorer view, right-click **count\_binary\_bsp** and click **Properties**. The **Properties for count\_binary\_bsp** dialog box appears.
2. Click the **Nios II BSP Properties** page. The **Nios II BSP Properties** page contains basic software build settings. Figure 1-21 shows the GUI.

**Figure 1-21. System Library Properties**



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

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



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


4. Click **OK**. The BSP regenerates, the **Properties** dialog box closes, and you return to the Nios II SBT for Eclipse.
5. In the Project Explorer view, right-click the **count\_binary** project and click **Build Project**.

The **Build Project** dialog box appears, and the Nios II SBT for Eclipse begins compiling the project. When compilation completes, a "count\_binary build complete" message appears in the Console view.

## Run the Program on Target Hardware

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

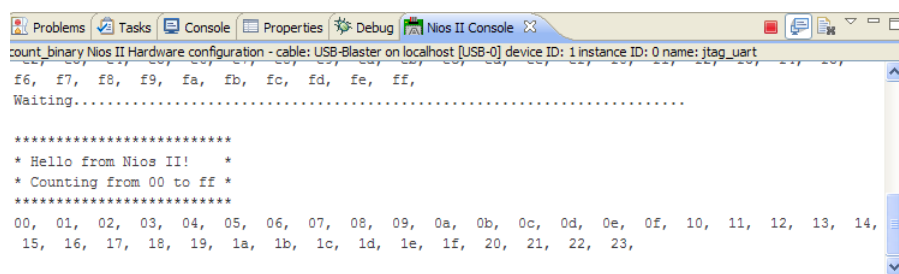
1. Right-click the **count\_binary** project, point to **Run As**, and then click **Nios II Hardware**. The Nios II SBT for Eclipse downloads the program to the FPGA on the target board and the program starts running.

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

When the target hardware starts running the program, the Nios II Console view displays character I/O output. [Figure 1-22](#) shows the output. If you connected LEDs to the Nios II system in "[Integrate the Qsys System into the Quartus II Project](#)" on [page 1-24](#), then the LEDs blink in a binary counting pattern.

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

**Figure 1-22. Console View Displaying Nios II Hardware Output**



```
count_binary Nios II Hardware configuration - cable: USB-Blaster on localhost [USB-0] device ID: 1 instance ID: 0 name: jtag_uart
f6, f7, f8, f9, fa, fb, fc, fd, fe, ff,
Waiting.....

*****
* Hello from Nios II! *
* Counting from 00 to ff *
*****
00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 0a, 0b, 0c, 0d, 0e, 0f, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 1a, 1b, 1c, 1d, 1e, 1f, 20, 21, 22, 23,
```

You can edit the **count\_binary.c** program in the Nios II SBT for Eclipse text editor and repeat these two steps to witness your changes executing on the target board. If you rerun the program, buffered characters from the previous run session might display in the Console view before the program begins executing.

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

## Taking the Next Step

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

- Analyzing system requirements
- Defining and generating Nios II system hardware in Qsys
- Integrating the Qsys system into a Quartus II project
- Compiling the Quartus II project and verifying timing
- Creating a new project in the Nios II SBT for Eclipse
- Compiling the project
- Running the software on target hardware

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

- *Nios II Software Developer's Handbook*—This handbook provides complete reference for developing software for the Nios II processor.
- The software development tutorial in the *Getting Started with the Graphical User Interface* chapter of the *Nios II Software Developer's Handbook*—This tutorial teaches in detail how to use the Nios II SBT for Eclipse to develop, run, and debug new Nios II C/C++ application projects.
- *Nios II Processor Reference Handbook*—This handbook provides complete reference for the Nios II processor hardware.
- The *System Design with Qsys* section of *Volume 1: Design and Synthesis* of the *Quartus II Handbook*—This volume provides complete reference on using Qsys, including topics such as building memory subsystems, creating custom components, and automatically generating interconnect fabric based on a network-on-a-chip topology.
- The *Embedded Peripherals IP User Guide*—This user guide contains details about the components provided free as part of the Nios II EDS.

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