Hardware Development Tutorial

2014.09.22

AN-717 Subscribe Send Feedback

This tutorial describes the system development flow for the Nios II processor.

Using the software and the Embedded Design Suite (EDS), you can:

- build a hardware system design
- create a software program that runs on the system and interfaces with components on development boards

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.

Software and Hardware Requirements

The following are the software requirements for the tutorial:

- software version 14.0 or later—The software must be installed on a Windows or Linux computer that meets the minimum requirements.
- EDS version 14.0 or later.
- Design files for the design example—Refer related information below for the design example file.

You can build the design example with any development board or your own custom board that meets the following hardware requirements:

- The board must have either , Stratix series, Cyclone series, or Arria series FPGA.
- The FPGA must contain a minimum of 2800 logic elements (LE) or adaptive lookup tables (ALUT).
- The FPGA must contain a minimum of 40 M9K memory blocks.
- An oscillator must drive a constant clock frequency to an FPGA pin. The maximum frequency limit depends on the speed grade of the FPGA. Frequencies of 50 MHz or less should work for most boards; higher frequencies might work.
- FPGA I/O pins can optionally connect to eight or fewer LEDs to provide a visual indicator of processor activity.
- The board must have a JTAG connection to the FPGA that provides a programming interface and communication link to the system. The JTAG connection can be a dedicated 10-pin JTAG header for an FPGA USB Download Cable or a USB connection with circuitry embedded on the board.

Note: Refer to the documentation for your board that describes clock frequencies and pinouts. For development boards, refer to the related information below.

Related Information

- **Altera Development Kits Documentation**
- **Altera Software Installation and Licensing**

You can perform this tutorial on hardware without a license. With , you can perform the following actions:

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

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

Related Information

OpenCore Plus Evaluation of Megafunctions
Provides more information about OpenCore Plus.

Design Example

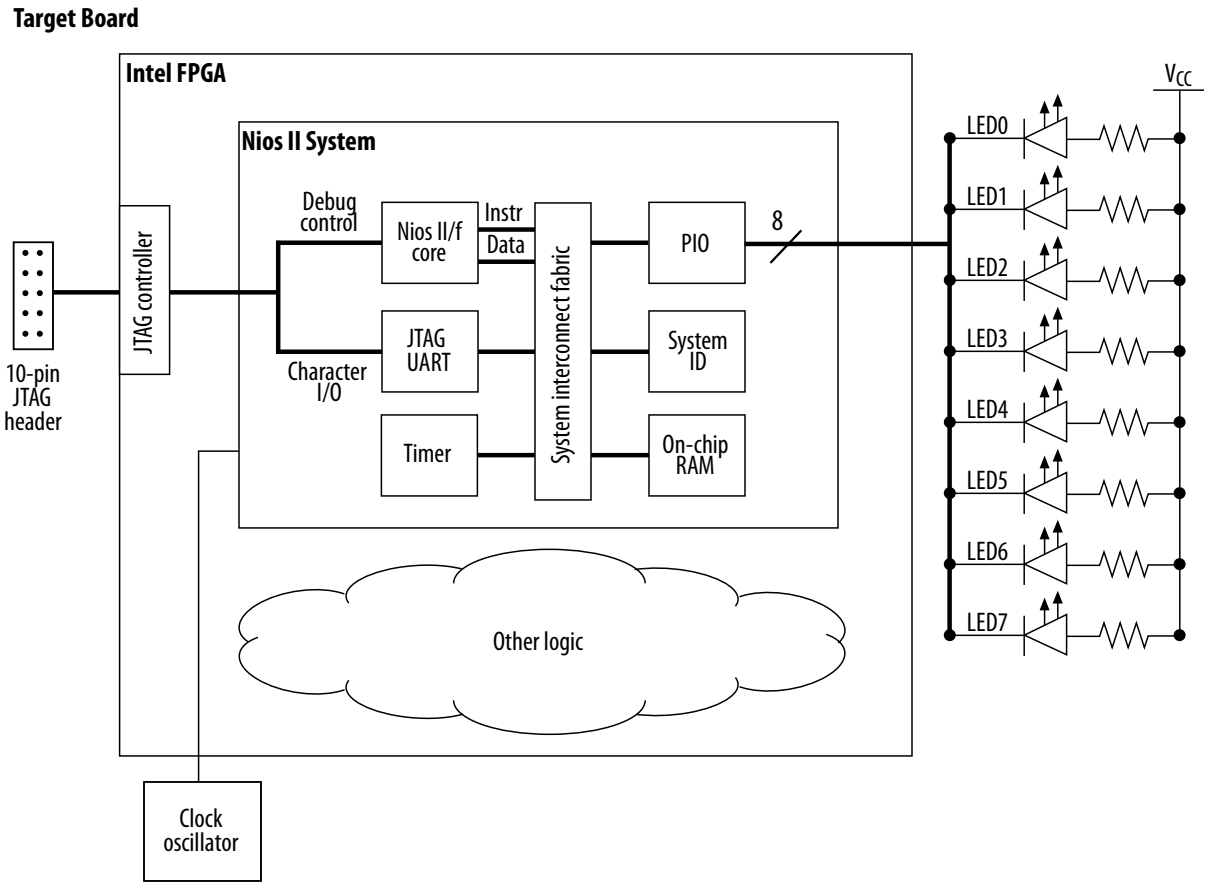
The design example you build in this tutorial demonstrates a small system for control applications, that displays character I/O output and blinks LEDs in a binary counting pattern. This system can also communicate with a host computer, allowing the host computer to control logic inside the FPGA.

The example system contains the following components:

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

Figure 1: Design Example Block Diagram

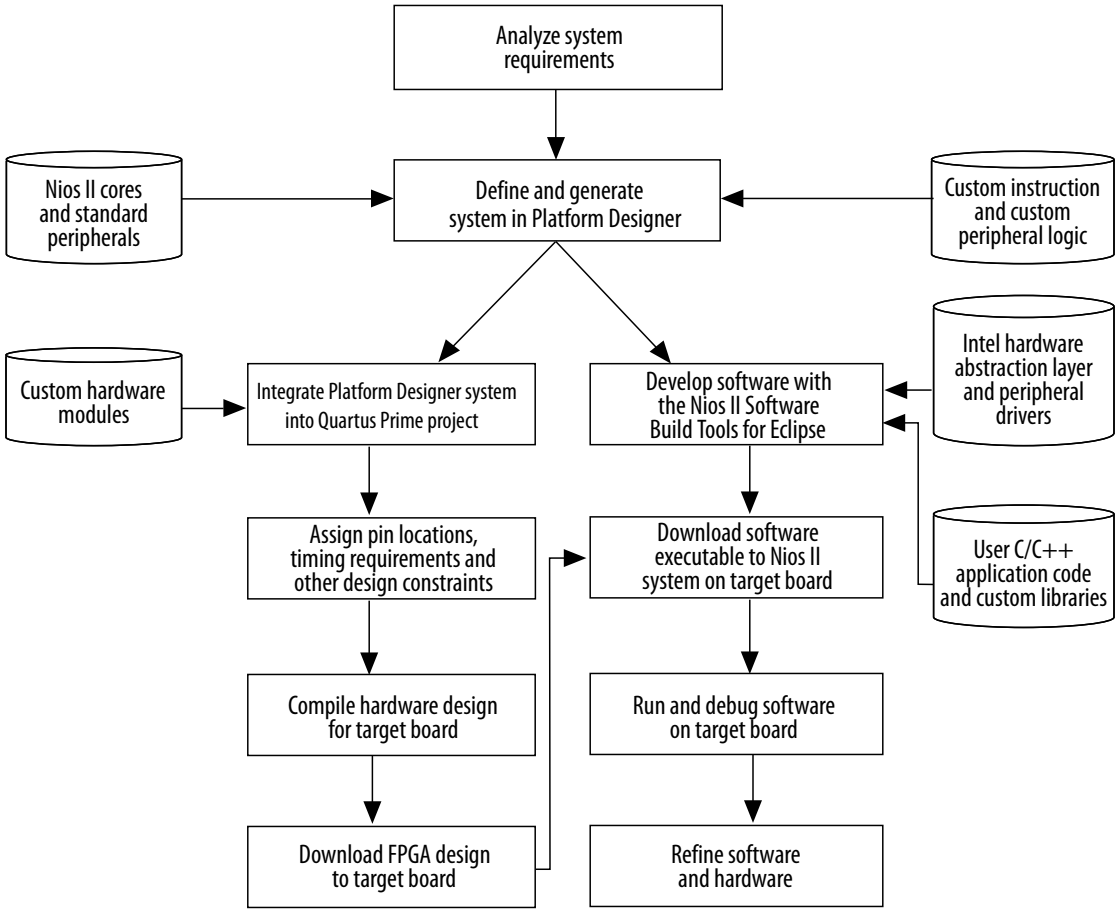
The block diagram shows the relationship between the host computer, the target board, the FPGA, and the system.



Other logic can exist within the FPGA alongside the system. In fact, most FPGA designs with a system also include other logic. A system can interact with other on-chip logic, depending on the needs of the overall system. This design example does not include other logic in the FPGA.

System Development Flow

Figure 2: System Development Flow



The development flow consists of three types of development:

- hardware design steps
- software design steps
- system design steps, involving both hardware and software

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

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 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 point of view.

Defining and Generating the System in

After analyzing the system hardware requirements, you use to specify the processor core(s), memory, and other components your system requires. 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 EDS. You can also add your own custom hardware to accelerate system performance. You can add custom instruction logic to the 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 are the following file types:

Table 1: Primary Output File Types

File Types	Description
Design File (`.qsys`)	Contains the hardware contents of the system
SOPC Information File (`.sopcinfo`)	Contains a description of the contents of the .qsys file in Extensible Markup Language File (`.xml`) format. The 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 system. The software uses the `HDL` files to compile the overall FPGA design into an SRAM Object File (`.sof`).

Related Information

Volume 1: Design and Synthesis of the Quartus II Handbook
Provides more information about Qsys and developing custom components

Integrating the System into the Project

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

Using the software, you can:

- assign pin locations for I/O signals
- specify timing requirements
- apply other design constraints
- compile the project to produce a `.sof` to configure the FPGA

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

Developing Software with the Software Build Tools for Eclipse

You can perform all software development tasks for your processor system using the Software Build Tools (SBT) for Eclipse™.

After you generate the system with , you can begin designing your C/C++ application code immediately with the SBT for Eclipse. provides component drivers and a hardware abstraction layer (HAL) which allows you to write 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 SBT for Eclipse projects.

To create a new C/C++ application project, the 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 SBT for Eclipse can produce several outputs, listed below. Not all projects require all of these outputs.

Table 2: SBT for Eclipse Outputs

The SBT for Eclipse can produce several outputs but not all projects require all of these outputs.

Output	Description
`system.h` file	• Defines symbols for referencing the hardware in the system. • The 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 processor.

Output	Description
Hexadecimal (-Format) File (`.hex`)	• Contains initialization information for on-chip memories. • The SBT for Eclipse generate these initialization files for on-chip memories that support initialization of contents.
Flash memory programming data	• Boot code and other arbitrary data you might write to flash memory. • The flash programmer adds appropriate boot code to allow your program to boot from flash memory. • The SBT for Eclipse includes a flash programmer to allow you to write your program or arbitrary data to flash memory.

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

Running and Debugging Software on the Target Board

The SBT for Eclipse has the capability to download software to a target board, and run or debug the program on hardware. The SBT for Eclipse debugger allows you to start and stop the processor, step through code, set breakpoints, and analyze variables as the program executes.

Varying the Development Flow

The development flow is not strictly linear. The following lost the common variations:

- Refining the Software and Hardware
- Iteratively Creating a System
- Verifying the System with Hardware Simulation Tools

Refining the Software and Hardware

After running software on the target board, you might discover that the system requires higher performance. In this case, you can:

- return to software design steps to make improvements to the software algorithm; or
- 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) processors that divide the workload and improve the performance of each individual processor.

Iteratively Creating a System

A common technique for building a complex system is to start with a simpler 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; then integrate the custom logic into a more complex system.

Verifying the System with Hardware Simulation Tools

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

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

If you are building a system based on the standard components provided with the EDS, the easiest way to verify functionality is to download the hardware and software directly to a development board.

Creating the Design Example

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

Install the Design Files

Perform the following steps to set up the design environment:

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

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

Analyze System Requirements

The system requirements are derived from the following goals of the tutorial design example:

- Demonstrate a simple 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 systems.
- Build a system that works on any board with an 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 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:
  - /f core with 2 KB of instruction cache with static branch prediction
  - 20 KB of on-chip memory
  - Timer
  - JTAG UART
  - Eight output-only parallel I/O (PIO) pins
  - System ID component

Related Information

Embedded Peripheral IP User Guide
Provides more information about the JTAG UART, timer, system ID peripheral, and PIO.

Start the Software and Open the Example Project

The project serves as an easy starting point for the development flow. The project contains all settings and design files required to create the `.sof`. To open the project, perform the following steps:

1. Start the software.
2. Click **Open Existing Project** on the splash screen, or, on the **File** menu, click **Open Project**. The **Open Project** dialog box appears.
3. Browse to the `<design files directory>`.
4. Select the file `nios2_quartus2_project.qpf` and click **Open**.
5. To display the Block Diagram File (`.bdf`) `nios2_quartus2_project.bdf`, perform the following steps:
   a. On the **File** menu, click **Open**. The **Open** dialog box appears.
   b. Browse to the `<design files directory>`.
   c. Select `nios2_quartus2_project.bdf` and click **Open.**

   The `.bdf` contains an input pin for the clock input and eight output pins to drive LEDs on the board.

Next, you create a new system, which you ultimately connect to these pins.

Create a New System

You use to generate the processor system, adding the desired components, and configuring how they connect together. To create a new system, click on the **Tools** menu in the software. starts and displays the **System Contents** tab.

Define the System in

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

The 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 design steps in a different order.

Specify Target FPGA and Clock Settings

To specify target FPGA and clock settings, perform the following steps:

1. On the **Project Settings** tab, select the **Device Family** that matches the 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.
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 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 system. As you add each component, you configure it appropriately to match the design specifications.

Related Information

Altera Development Kits Documentation

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 **IP Catalog** tab (to the left of the **System Contents** tab), expand **Basic Functions**, expand **On-Chip Memory**, and then click **On-Chip Memory (RAM or ROM)**.
2. Click **Add**.
   The On-Chip Memory (RAM or ROM) parameter editor appears.
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.
5. Click **Finish**. You return to .
6. Click the **System Contents** tab.
   An instance of the on-chip memory appears in the system contents table.
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 system fail in later steps. In general, it is a good habit to give descriptive names to hardware components. programs use these symbolic names to access the component hardware. Therefore, your choice of component names can make programs easier to read and understand.

Add the Processor Core

You add the /f core and configure it to use 2 KB of on-chip instruction cache memory, no data cache and use static branch prediction. For this tutorial, the /f core is configured to provide a balanced trade-off between performance and resource utilization. To add a /f core to the system, perform the following steps:

1. On the **IP Catalog** tab, expand **Processors and Peripherals**, and then click **Gen2 Processor**.
2. Click **Add**.
   The Processor parameter editor appears, displaying the **Core Nios II** tab.
3. In the **Main Tab** under **Select an Implementation**, select **/f**.
4. Click **Finish** and return to the **System Contents** tab.
   The core instance appears in the system contents table. Ignore the exception and reset vector error messages. You resolve these errors in future steps.
5. In the **Name** column, right-click the processor and click **Rename**.
6. Type `cpu` and press **Enter**.
7. 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 processor by clicking the hollow dots on the connection line. The dots become solid indicating the ports are connected.
8. 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 processor.
9. Connect the `s1` port of the on-chip memory to both the `data_master` port and `instruction_master` port of the processor.
10. Double-click the processor row of the system contents table to reopen the Processor parameter editor.
11. Under **Reset Vector** in **Vectors** tab, select **onchip_mem.s1** in the **Reset vector memory** list and type `0x0` in the **Reset vector offset** box.
12. Under **Exception Vector**, select **onchip_mem.s1** in the **Exception vector memory** list and type `0x20` in the **Exception vector offset** box.
13. Click the Caches and Memory Interfaces tab.
14. In the Instruction cache list, select 2 Kbytes.
15. Choose **None** for **Data Cache size** and do not change other default settings.
16. In **Advanced Features** tab, select **Static branch prediction** type.
17. Click **Finish**. You return to the System Contents tab.

    Do not change any settings on the **MMU and MPU Settings** and **JTAG Debug** tabs.

Add the JTAG UART

The JTAG UART provides a convenient way to communicate character data with the processor through the . To add the JTAG UART, perform the following steps:

1. On the **IP Catalog** tab, expand **Interface Protocols**, expand **Serial**, and then click **JTAG UART**.
2. Click **Add**.
   The **JTAG UART** parameter editor appears and do not change the default settings.
3. Click **Finish** and return to the **System Contents** tab.
   The JTAG UART instance 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 processor to the `avalan_jtag_slave` port of the **JTAG UART**.

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

Related Information

Embedded Peripheral IP User Guide
Provides more information about the JTAG UART, timer, system ID peripheral, and PIO.

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 HAL requires a timer. To add the timer, perform the following steps:

1. On the **IP Catalog** tab, expand **Processors and Peripherals**, expand **Peripherals**, and then click **Interval Timer**.
2. Click **Add**.
   The **Interval Timer** parameter editor appears.
3. Click **Finish** return to the **System Contents** tab.
   The interval timer instance appears in the system contents table.
4. In the **Name** column, right-click the **interval timer** and click **Rename.**
5. Type `sys_clk_timer` and press **Enter.**
6. Connect the `clk` port of the **clk_0** clock source to the `clk` port of the **interval timer**.
7. Connect the `clk_reset` port of the **clk_0** clock source to the `reset` port of the **interval timer**.
8. Connect the `data_master port` of the **processor** to the `s1` port of the **interval timer**.

Related Information

Embedded Peripheral IP User Guide
Provides more information about the JTAG UART, timer, system ID peripheral, and PIO.

Add the System ID Peripheral

The system ID peripheral safeguards against accidentally downloading software compiled for a different system. If the system includes the system ID peripheral, the SBT for Eclipse can prevent you from downloading programs compiled for a different system. To add system ID peripheral, perform the following steps:

1. On the **IP Catalog** tab, expand **Basic Functions**, expand **Simulations; Debug and Verifications** and then click **System ID Peripheral**.
2. Click **Add**.
   The **System ID Peripheral** parameter editor appears and do not change the default setting.
3. Click **Finish** and return to the **System Contents** tab.
   The system ID peripheral instance 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 **processor** to the `control_slave` port of the **system ID peripheral**.

Related Information

Embedded Peripheral IP User Guide
Provides more information about the JTAG UART, timer, system ID peripheral, and PIO.

Add the PIO

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

Note: Perform these steps even if your target board does not have LEDs.

1. On the **IP Catalog** tab, expand **Processors and Peripherals**, expand **Peripherals**, and then click **PIO** .
2. Click **Add**.
   The **PIO (Parallel I/O)** parameter editor appears and do not change the default settings.
3. Click **Finish** and return to the **System Contents** tab.
   The PIO instance 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 **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.

Related Information

Embedded Peripheral IP User Guide
Provides more information about the JTAG UART, timer, system ID peripheral, and PIO.

Specify Base Addresses and Interrupt Request Priorities

To specify how the components added in the design to interact to form a system, you need assign base addresses for each slave component, and assign interrupt request (IRQ) priorities for the JTAG UART and the interval timer.

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:

- processor cores can address a 31-bit address span. You must assign base address between `0x00000000` and `0x7FFFFFFF`.

  Note: The **Use most-significant address bit in processor to bypass data cache** option is enable by default. If disabled, the processor cores supports full 32-bit address.
- programs use symbolic constants to refer to addresses. You do not have to choose address values that are easy to remember.
- Address values that differentiate components with only a one-bit address difference produce more efficient hardware. You do not have to compact all base addresses into the smallest possible address range, because this can create less efficient hardware.
- 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.

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 does not know the software behavior, cannot make educated guesses about the best IRQ assignment.

The 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 assign functional base addresses to each component in the system. Values in the **Base** and **End** columns might change, reflecting the addresses that reassigned.
2. In the **IRQ** column, connect the 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.

Generate the System

To generate the system, perform the following steps:

1. Click the **Generation** tab.
2. Select **None** in both the **Create simulation model** and **Create testbench system** lists.

   Because this tutorial does not cover the hardware simulation flow, you can select these settings to shorten generation time.
3. Click **Generate**. Click **Yes** when the **Save changes?** dialog box appears.
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. When generation completes, prompt: `Create HDL design files for synthesis`.

5. Click **Close** to close the dialog box.
6. On the **File** menu, click **Exit** to close and return to the software.

You are ready to integrate the system into the hardware project and use the SBT for Eclipse to develop software.

Integrate the System into the Project

To complete the hardware design, you need to perform the following tasks:

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

Instantiate the System Module in the Project

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

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

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

1. Double-click in the empty space to the right of the input and output wires.
   The **Symbol** dialog box appears.
2. Under **Libraries**, expand **Project**.
3. Click **first_nios2_system**.
   The **Symbol** dialog box displays the **first_nios2_system** symbol.
4. Click **OK**. You return to the `.bdf` schematic. The **first_nios2_system** symbol tracks with your mouse pointer.
5. Position the symbol so the pins on the symbol align with the wires on the schematic.
6. Click to anchor the symbol in place.
7. If your target board does not have LEDs that the system can drive, you must delete the **LEDG[7..0]** pins. To delete the pins, perform the following steps:
   a. Click the output symbol **LEDG[7..0]** to select it.
   b. On your keyboard, press **Delete.**
8. To save the completed `.bdf`, click **Save** on the **File** menu.

Add IP Variation File

You can add the IP File (`.qip`) to the your project by performing 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.

Assign FPGA Device

Before assigning FPGA pin locations to match the pinouts of your board, you need to first assign a specific target device. To assign the device, perform the following steps:

1. On the **Assignments** menu, click **Device**.
   The **Device** dialog box appears.
2. In the **Family** list, select the FPGA family that matches your board.
   If prompted to remove location assignments, do so.
3. Under **Target device**, select **Specific device selected in Available devices list**.
4. Under **Available devices**, select the exact device that matches your board.
   If prompted to remove location assignments, do so.
5. Click **OK** to accept the device assignment.

Assign FPGA Pin Locations

Before assigning the FPGA pins, you must know the pin layout for the board. You also must know other requirements for using the board, refer to related information below. To assign the FPGA pin locations, perform the following steps:

1. On the Processing menu, point to **Start**, and click **Start Analysis & Elaboration** to prepare for assigning pin locations.
   The analysis starts by displaying a `data not available` message and can take several minutes. A confirmation message box appears when analysis and elaboration completes.
2. Click **OK.**
3. On the **Assignments** menu, click **Pin Planner**.
   The **Pin Planner** appears.
4. In the **Node Name** column, locate **PLD_CLOCKINPUT**.
5. In the **PLD_CLOCKINPUT** row, double-click in the **Location** cell to access a list of available pin locations.
6. Select the appropriate FPGA pin that connects to the oscillator on the board.
   If your design fails to work, recheck your board documentation for this step first.
7. In the **PLD_CLOCKINPUT** row, double-click in the **I/O Standard** cell to access a list of available I/O standards.
8. Select the appropriate I/O standard that connects to the oscillator on the board.
9. If you connected the LED pins in the board design schematic, repeat steps 4 to 8 for each of the LED output pins (**LEDG[0]**, **LEDG[1]**, **LEDG[2]**, **LEDG[3]**, **LEDG[4]**, **LEDG[5]**, **LEDG[6]**, **LEDG[7]**) to assign appropriate pin locations.
10. On the **File** menu, click **Close** to save the assignments.
11. On the **Assignments** menu, click **Device**.

The **Device** dialog box appears.

12. Click **Device and Pin Options**.
    The **Device and Pin Options** dialog box appears.
13. Click the **Unused Pins** page.
14. In the **Reserve all unused pins** list, select **As input tri-stated with weak pull-up**. With this setting, all unused I/O pins on the FPGA enter a high-impedance state after power-up.

    Unused pins are set as input tri-stated with weak pull-up to remove contention which might damage the board. Depending on the board, you might have to make more assignments for the project to function correctly. You can damage the board if you fail to account for the board design. Consult with the maker of the board for specific contention information.
15. Click **OK** to close the **Device and Pin Options** dialog box.
16. Click **OK** to close the **Device** dialog box.

Related Information

Altera Development Kits Documentation

Set Timing

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

1. On the **File** menu, click **Open**.
2. In the **Files of type** list, select **Script Files (*.tcl, *.sdc, *.qip)**.
3. Browse to locate `<design files directory>/hw_dev_tutorial.sdc` and click **Open**. The file opens in the text editor.
4. Locate the following `create_clock` command:`create_clock -name sopc_clk -period 20 [get_ports PLD_CLOCKINPUT]`
5. Change the period setting from 20 to the clock period (1/frequency) in nanoseconds of the oscillator driving the clock pin.
6. On the **File** menu, click **Save.**
7. On the **Assignments** menu, click **Settings**.
   The **Settings** dialog box appears.
8. Under **Category**, click **TimeQuest Timing Analyzer**.
9. Next to File name, click the browse **(...)** button.
10. Browse to locate `<design files directory>/hw_dev_tutorial.sdc` and click **Open** to select the file.
11. Click **Add** to include `hw_dev_tutorial.sdc` in the project.
12. Turn on **Enable multicorner timing analysis during compilation**.
13. Click **OK**.

Compile the Project and Verify Timing

To create a `.sof` file, you have to compile the hardware design and then it download to the board. After the compilation completes, you must analyze the timing performance of the FPGA design to verify that the design works in hardware. To compile the project, perform the following steps:

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

The **Tasks** window and percentage and time counters in the lower-right corner display progress. The compilation process can take several minutes. When compilation completes, a dialog box displays the message "Full Compilation was successful."

2. Click **OK**. The Quartus Prime software displays the **Compilation Report** tab.
3. Expand the **TimeQuest Timing Analyzer** category in the compilation report.
4. Click **Multicorner Timing Analysis Summary**.
5. Verify that the **Worst-case Slack** values are positive numbers for **Setup**, **Hold**, **Recovery**, and **Removal**.

   If any of these values are negative, the design might not operate properly in hardware. To meet timing, adjust assignments to optimize fitting, or reduce the oscillator frequency driving the FPGA.

Download the Hardware Design to the Target FPGA

To download the `.sof` to the target board, perform the following steps:

1. Connect the board to the host computer with the download cable, and apply power to the board.
2. On the **Tools** menu in the software, click **Programmer**.
   The Programmer appears and automatically displays the appropriate configuration file (`nios2_quartus2_project.sof`).
3. Click **Hardware Setup** in the upper left corner of the Programmer to verify your download cable settings.
   The **Hardware Setup** dialog box appears.
4. Select the appropriate download cable in the **Currently selected hardware** list.

   If the appropriate download cable does not appear in the list, you must first install drivers for the cable.
5. Click **Close**.
6. In the **nios2_quartus2_project.sof** row, turn on **Program/Configure**.
7. Click **Start**.
   The Progress meter sweeps to 100% as the software configures the FPGA.

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

Related Information

Altera Programming Cables Documentation

Develop Software Using the SBT for Eclipse

Developing software using the SBT for Eclipse consists the following tasks:

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

To perform these steps, you must have the `.sopcinfo` file you created earlier in this tutorial. Refer to related information for more information.

Note: This tutorial presents only the most basic software development steps to demonstrate software running on the hardware system you created in previous sections.

Related Information

Define the System in on page 9

Create a New Application and BSP from Template

To create new C/C++ application and BSP projects, perform the following steps:

1. Start the SBT for Eclipse. On Windows computers, click **Start**, point to **Programs**, **EDS <version>**, and then click **<version> Software Build Tools for Eclipse**. On Linux computers, run the executable file `<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 , or click **Other** and then click to ensure you are using the perspective.
4. On the **File** menu, point to **New**, and then click **Application and BSP from Template**.
   The **Nios II Application and BSP from Template** wizard appears.
5. Under **Target hardware information**, next to **SOPC Information File name**, browse to locate the `<design files directory>`.
6. Select `first_nios2_system.sopcinfo` and click **Open**.
   Nios II Application and BSP from Template wizard shows the current information for the **SOPC Information File name** and **CPU name** fields.
7. In the **Project name** box, type `count_binary`.
8. In the **Templates** list, select **Count Binary**
9. Click **Finish**.

The 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 system hardware

Compile the Project

You have to 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 hardware system contains only 20 KB of memory. To adjust the project settings and compile the project, 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 **BSP Properties** page. The BSP Properties page contains basic software build settings.
   Though not needed for this tutorial, note the BSP Editor button in the lower right corner of the dialog box. You use the BSP Editor to access advanced BSP settings.
3. Adjust the following settings to reduce the size of the compiled executable:
   a. Turn on **enable_reduced_device_drivers**.
   b. Turn off **enable_gprof**.
   c. Turn on **enable_small_c_library**.
   d. Turn off **enable_sim_optimize**.
4. Click **OK**.
   The BSP regenerates, the **Properties** dialog box closes, and you return to the 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 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

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 **Hardware**.

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

   The SBT for Eclipse downloads the program to the FPGA on the target board and the program starts running. When the target hardware starts running the program, the Console view displays character I/O output. If you connected LEDs to the system in previous section, then the LEDs blink in a binary counting pattern.
2. Click the **Terminate** icon (the red square) on the tool bar of the Console view to terminate the run session and the SBT for Eclipse disconnects from the target hardware.

   You can edit the `count_binary.c` program in the 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.

Related Information

Integrate the System into the Project on page 15

Document Revision History

Date	Version	Changes
September 2014	2014.09.22	Initial release.