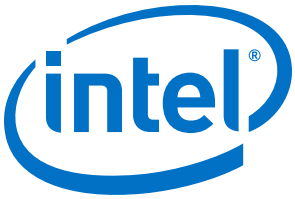


Using the NicheStack TCP/IP Stack - Nios II Edition Tutorial

Subscribe
Send Feedback

TU-01001 | 2017.10.09
Latest document on the web: PDF | HTML

Contents

1 Using the NicheStack TCP/IP Stack - Nios II Edition Tutorial

This tutorial introduces you to the Nios® II Software Build Tools (SBT) for Eclipse™ using the MicroC/OS-II and NicheStack TCP/IP Stack development flow. It shows you how to use the Nios II SBT for Eclipse to create a new Nios II project that configures, builds, and runs a MicroC/OS-II and NicheStack TCP/IP Stack program on an Intel® FPGA development board.

1.1 Introduction

This tutorial familiarizes you with the NicheStack TCP/IP Stack – Nios II Edition (NicheStack TCP/IP Stack) software component. The tutorial covers the following topics:

- Configuring and initializing the NicheStack TCP/IP Stack software component
- Managing a TCP/IP connection with MicroC/OS-II real-time operating system (RTOS) tasks
- Using the Nios II SBT for Eclipse to develop programs with the NicheStack TCP/IP Stack software component

The Nios II SBT for Eclipse offers software designers a rich development platform for Nios II applications. The Nios II SBT for Eclipse contains the MicroC/OS-II RTOS and the NicheStack TCP/IP Stack software component, providing designers with the ability to quickly build networked embedded systems applications for the Nios II processor. This tutorial provides step-by-step instructions for building a simple program based on the MicroC/OS-II RTOS and NicheStack TCP/IP Stack networking stack.

This tutorial describes C software files that demonstrate communication with a telnet client on a development host PC. The telnet client offers a convenient way of issuing commands over a TCP/IP socket to the Ethernet-connected NicheStack TCP/IP Stack running on the development board with a simple TCP/IP socket server example. The socket server example receives commands sent over a TCP/IP connection and turns LEDs on and off according to the commands. The example consists of a socket server task that listens for commands on a TCP/IP port and dispatches those commands to a set of LED management tasks.

Note: The Nios II target system does not implement a full telnet server.

For more information about MicroC/OS-II for the Nios II processor, refer to the MicroC/OS-II Real-Time Operating System chapter of the *Nios II Software Developer's Handbook*.

For more information about NicheStack TCP/IP Stack initialization and configuration for the Nios II processor, refer to the Ethernet and the NicheStack TCP/IP Stack – Nios II Edition chapter of the *Nios II Software Developer's Handbook*.

Related Links

- MicroC/OS-II Real-Time Operating System
- Ethernet and the NicheStack TCP/IP Stack - Nios II Edition

1.2 Hardware and Software Requirements

This tutorial requires the following hardware and software:

- Quartus® Prime software version 17.0 or later
- Nios II Embedded Design Suite (EDS) version 17.0 or later
- One of the following Intel FPGA development kit boards:
  - Arria 10 SoC Development Kit
  - Stratix IV GX FPGA Development Kit
- Intel FPGA Download Cable II
- RJ-45 connected Ethernet cable on the same network as the PC development host

To complete this tutorial, you must install the Nios II SBT for Eclipse and you must connect your development board to a host PC on the Ethernet and USB/JTAG ports. For hardware setup instructions, refer to the "Hardware Setup Details" section.

Related Links

Hardware Setup Details on page 22

1.3 Tutorial Files

1.3.1 Hardware Design Files

The Nios II Ethernet Standard Design Example page contains the hardware design files to use with this tutorial. Browse to the web page and locate the Nios II Ethernet Standard design example **.zip** file that corresponds to your board. Download and unzip the file in a directory of your choosing. The tutorial uses `<tutorial_files>` to refer to this directory.

Related Links

Nios II Ethernet Standard Design Example

1.3.2 Software Files

The NicheStack tutorial software files are available as an example within the EDS and do not have to be separately downloaded.

The software files constitute the Nios II Simple Socket Server application for this tutorial:

- **alt_error_handler.c**—Contains three error handlers, one each for the Nios II Simple Socket Server, NicheStack TCP/IP Stack, and MicroC/OS-II.
- **alt_error_handler.h**—Contains definitions and function prototypes for the three software component-specific error handlers.
- **iniche_init.c**—Defines main(), which initializes MicroC/OS-II and NicheStack TCP/IP Stack, processes the MAC and IP addresses, contains the PHY management tasks, and defines function prototypes.
- **led.c**—Contains the LED management tasks.
- **simple_socket_server.c**—Defines the tasks and functions that use the NicheStack TCP/IP Stack sockets interface, and creates all the MicroC/OS-II resources.
- **simple_socket_server.h**—Defines the task prototypes, task priorities, and other MicroC/OS-II resources used in this tutorial.
- **tse_my_system.c**—Defines the global structure of type "alt_tse_system_info", named "tse_mac_device" which describes the TSE configuration
- **network_utilities.c**—Contains MAC address and IP address routines to manage addressing. Routines are used by NicheStack during initialization, but are implementation-specific. You configure your MAC address to your preference or read it from your non-volatile memory.
- **network_utilities.h**—Contains prototype for function get_board_mac_addr().

1.4 Software Development Flow

The process for creating a NicheStack TCP/IP Stack and MicroC-OS/II software image for the Nios II processor consists of the following general steps:

1. Creating a new Nios II SBT for Eclipse C/C++ application project with the simple socket server project template
2. Configuring a board support package (BSP) project, including MicroC/OS-II and the NicheStack TCP/IP Stack software component
3. Building the application project
4. Running and debugging the application project

1.4.1 Creating a New Nios II Project

In this section, you create a new Nios II SBT for Eclipse project using a project template. To do so, follow these steps:

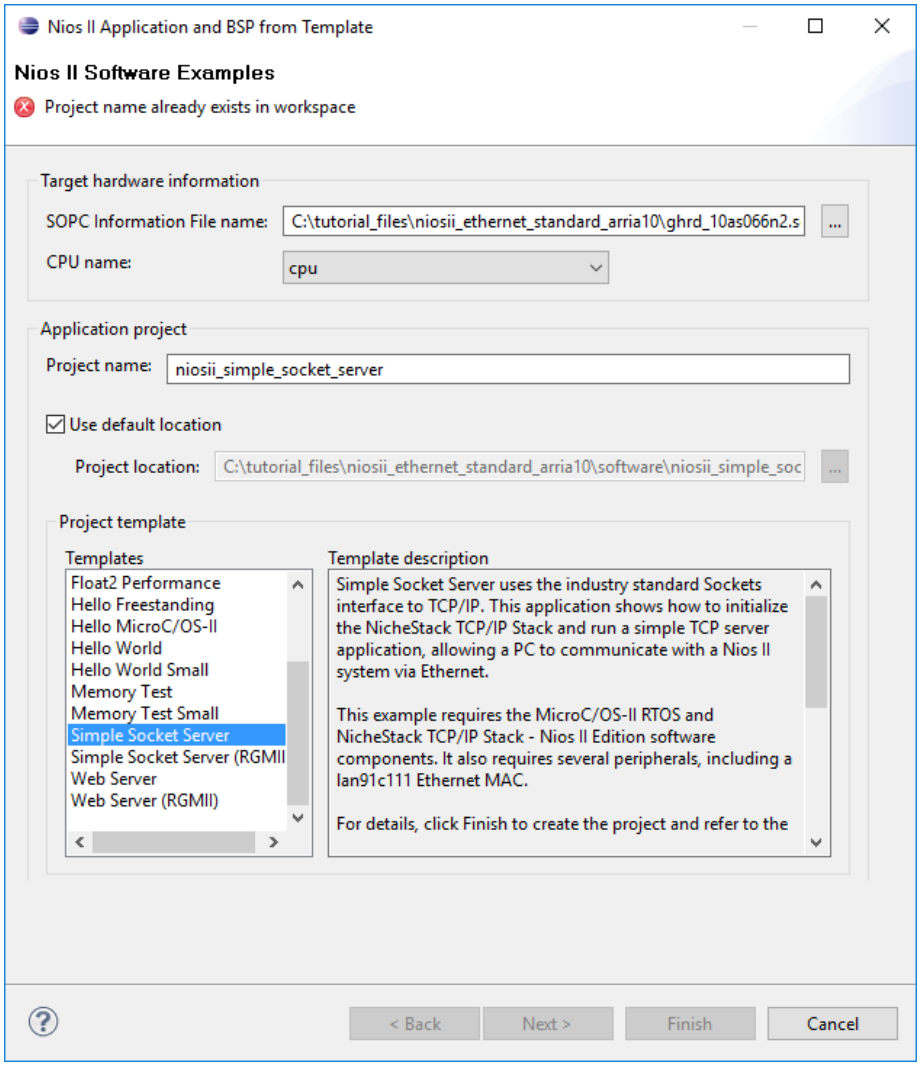
1. Start the Nios II SBT for Eclipse by performing one of the following actions:

- On Windows, on the Start menu, point to **All Apps > Intel FPGA** <*version*> **Pro Edition >**, and click **Nios II** <*version*> **Software Build Tools for Eclipse**.
- On Linux, open a Nios II Command Shell and type `eclipse-nios2`.

2. On the File menu, point to **New** and click **Nios II Application and BSP from Template**. The first page of the **Nios II Application and BSP from Template** wizard appears.
3. Under **Target hardware information**, browse to and open the <*tutorial_files*> **\niosii_ethernet_standard_**<*board*>**\eth_std_main_system.sopcinfo** SOPC Information File (**.sopcinfo**). The **SOPC Information File name** box contains the path to the **.sopcinfo** and the **CPU name** box contains the name of one of the available Nios II processors as defined in Platform Designer (Standard). The hardware design of the tutorial contains a single processor, so the software automatically selects the single processor.
4. In the **Project name** box, type `niosII_simple_socket_server`. The **Project location** fills in for you automatically.
5. Verify **Use default location** is on.
6. Under Project template, select **Simple Socket Server**

   Figure 1 on page 7 shows the state of the **Nios II Application and BSP from Template** wizard at this point in the tutorial.

Figure 1. Nios II Application and BSP from Template Wizard

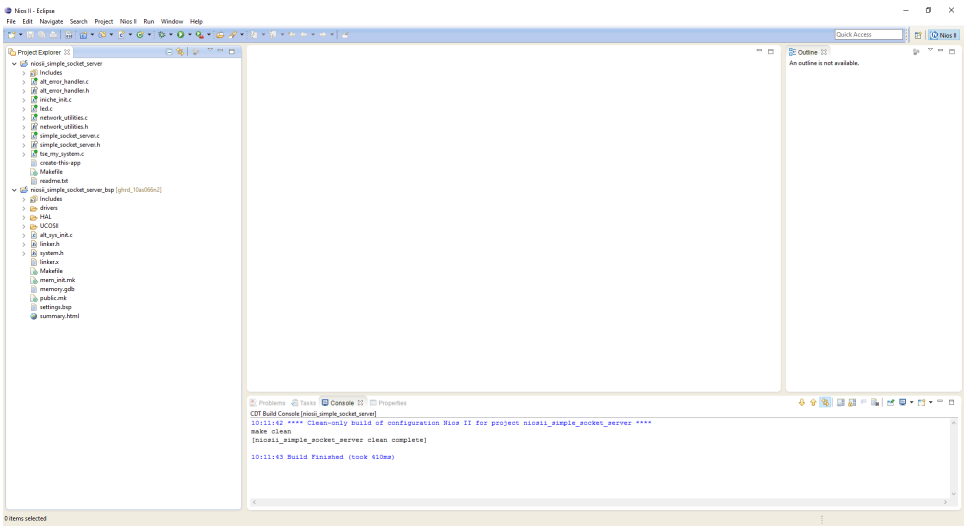


7. Click **Next**. The second page of the **Nios II Application and BSP from Template** wizard appears.
8. Select **Select an existing BSP project from your workspace**.
9. Click **Create**. The **Nios II Board Support Package** dialog box appears.
10. In the **BSP name** box, type `niosII_simple_socket_server_bsp`.
11. In the **Operating system** list, select **Micrium MicroC/OS-II**.
12. Click **Finish**. The wizard creates a BSP project and closes the **Nios II Board Support Package** dialog box.
13. Click **Finish**. The wizard creates an application project.

    Note: If the wizard prompts you to open the Nios II perspective, click **Yes**. If the **Finish** button is not available, click **Cancel** to close the previous GUI. Repeat Step 2 - 8, select **niosII_simple_socket_server_bsp**, and click **Finish**.

Figure 2 on page 8 shows the application and BSP projects in the Project Explorer view at this point in the tutorial.

Figure 2. New Projects in the Nios II Perspective



1.4.2 Configuring the BSP

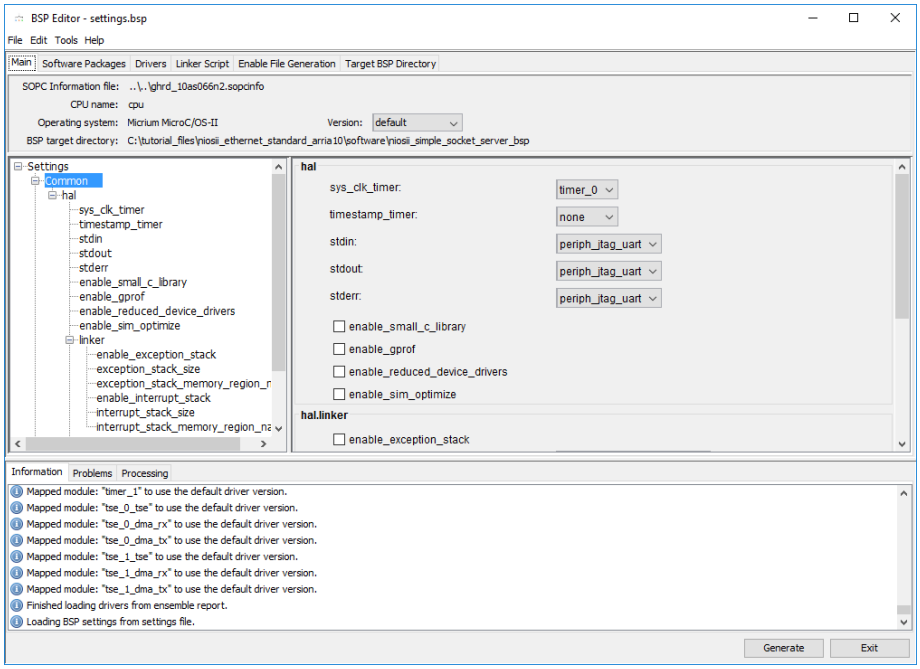
After you create a new BSP, you might want to customize its configuration (for example, defining stdin, stdout, stderr, and other parameters).

Note: For more information, refer to the Getting Started with the Graphical User Interface chapter of the *Nios II Software Developer's Handbook*.

For this tutorial, you must configure the MicroC/OS-II RTOS kernel and NicheStack TCP/IP Stack software components. To do so, follow these steps:

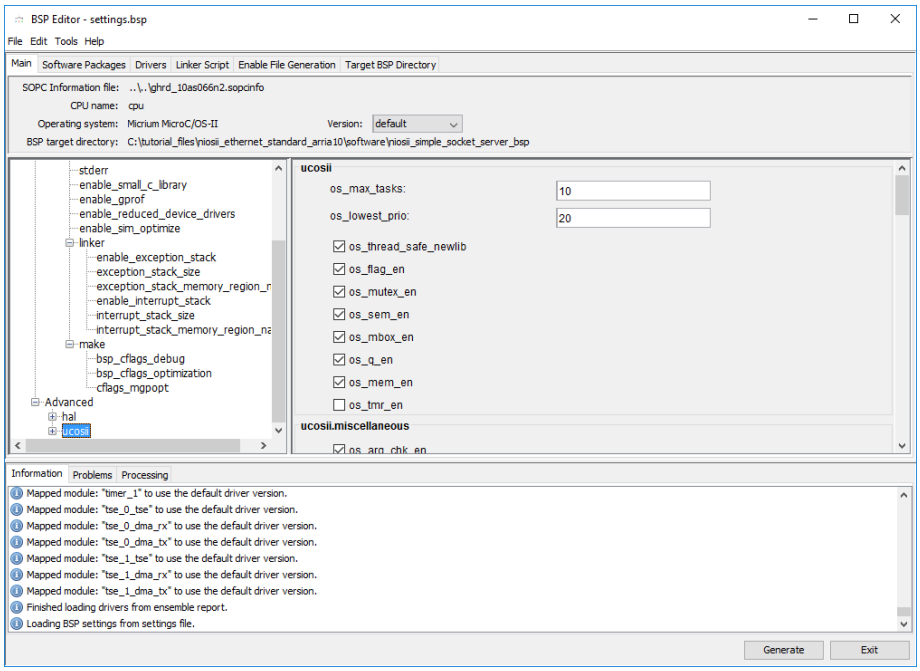
1. In the Project Explorer view, right-click the **niosII_simple_socket_server_bsp** project, point to **Nios II**, and click **BSP Editor**. The Nios II BSP Editor appears.
2. On the **Main** tab expand **Settings** in the left pane, and click **Common**. Verify that the settings for the stdout, stdin, and `stderr` parameters are **jtag_uart**, as shown in the "BSP Editor Main Tab" figure.

Figure 3. BSP Editor Main Tab



3. On the **Main** tab expand **Advanced** in the left pane, and click **ucosii**. Settings for the MicroC/OS-II RTOS appear, as shown in the "MicroC/OS-II RTOS Options" figure.

Figure 4. MicroC/OS-II RTOS Options



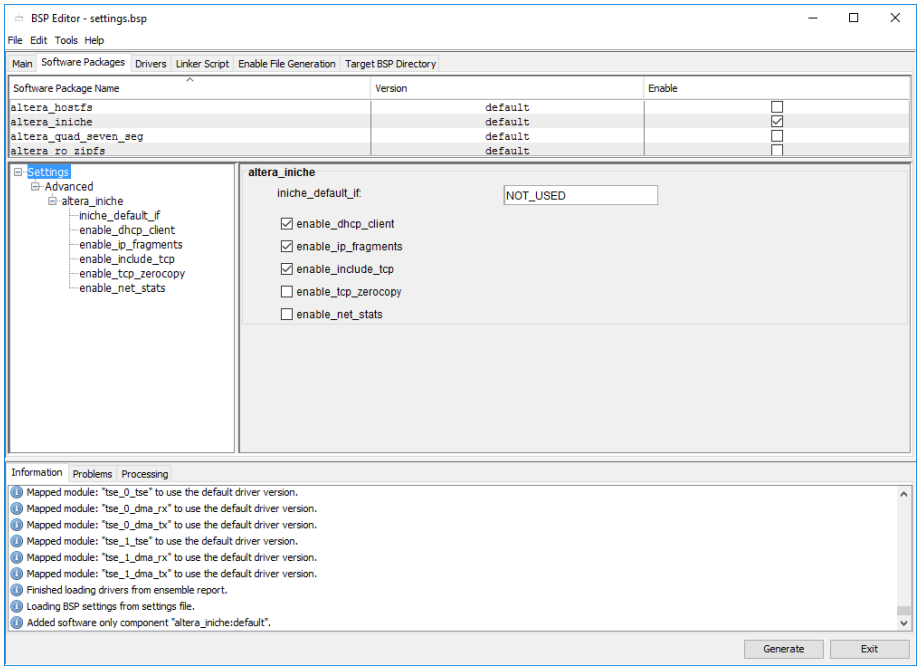
The MicroC/OS-II kernel is highly configurable. The options that you set in this dialog box determine which MicroC/OS-II options are included in the binary image. Examine the configurable options by clicking each of the options categories under ucosii in the left pane. For this tutorial, do not change any of the settings.

Although this example software design does not use all the MicroC/OS-II system calls, the NicheStack TCP/IP Stack internally uses many more MicroC/OS-II system calls, more than the Nios II Simple Socket Server application itself uses. Do not disable any system calls unless you need to be very conservative with your code size requirements. You must re-enable system calls that you try to disable if the link stage of the build fails with unresolved symbols.

For more information about the various MicroC/OS-II features, refer to the MicroC/OS-II Real-Time Operating System chapter in the *Nios II Software Developer's Handbook*.

4. On the **Software Packages** tab, turn on **Enable** for the **altera_iniche** software package, as shown in the "NicheStack TCP/IP Stack Options" figure.

Figure 5. NicheStack TCP/IP Stack Options



5. If a DHCP server is available on your network, turn on **enable_dhcp_client**. If no DHCP server is available, turn off **enable_dhcp_client** and specify your IP addresses, gateway, and network mask in *<tutorial_files>***\nichestack_tutorial\niosII_simple_socket_server.h**.

   Take care when choosing your default IP and gateway addresses. Some secure router configurations block DHCP request packets on local subnetworks such as the 192.168.X.X subnetwork. If you do encounter problems, try using 0.0.0.0 as your default IP and gateway addresses.

6. Click **Generate**. When prompted to save your changes, click **Yes**, **Save**.

7. Click **Exit** on the File menu to close the BSP Editor and return to the Nios II SBT for Eclipse.
8. In BSP project, you must add -DTSE_MY_SYSTEM to your defined symbols. Right-click the **niosII_simple_socket_server_bsp** project and click **Properties**. The Nios II BSP Properties page appears. On the left pane, click Nios II BSP Properties. In the **Defined symbols** box, type `-DTSE_MY_SYSTEM`.
9. Click **Apply** and **OK**.

Related Links

- MicroC/OS-II Real-Time Operating System
- Getting Started with the Graphical User Interface

1.4.3 Examining the Nios II Simple Socket Server Project Files

You have finished creating and configuring the **niosII_simple_socket_server** application and the associated BSP projects. Use the Project Explorer view, as shown in Figure 2 on page 8, to examine the project files in the **niosII_simple_socket_server** and **niosII_simple_socket_server_bsp** folders to understand more about the projects.

1.4.4 Building and Running the Nios II Simple Socket Server Project

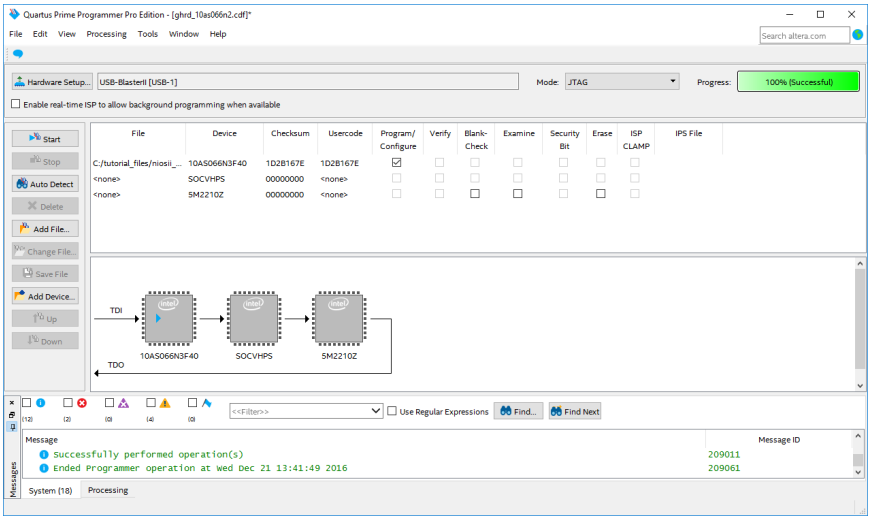
This section guides you to run the design example on an development board. This section also guides you to build the application, configure the development board with a hardware design, and download the executable software file to the FPGA on the board.

For more information about building and running programs with the Nios II SBT for Eclipse, refer to the Getting Started with the Graphical User Interface chapter of the *Nios II Software Developer's Handbook*.

To build and run the application, follow these steps:

1. Configure the FPGA on the development board by performing the following steps:
   a. On the Nios II menu, click Quartus Prime Programmer.
   b. In the Quartus Prime Programmer dialog box, on the File menu, click Open.
   c. Browse to and open the <*tutorial_files*>**\niosii_ethernet_standard_**<*board*>**\niosii_ethernet_standard_**<*board*>**.sof** SRAM Object File (**.sof**). Information for the file appears in the **Quartus Prime Programmer** dialog box.
   d. Verify Program/Configure is on, as shown in Figure 6 on page 12.

Figure 6. Quartus Prime Programmer Dialog Box



e. Click **Start** to configure the FPGA on the development board.

   If **Start** is disabled or if the Intel FPGA download cable is not listed in the **Hardware Setup** field, refer to the Quartus Prime Handbook for more information about the Quartus Prime Programmer.

f. On the File menu, click **Exit** to close the Quartus Prime Programmer and return to the Nios II SBT for Eclipse. If you receive a message that asks if you want to save the changes to the **chain1.cdf** file, click **No**.

2. In the Nios II SBT for Eclipse, select the **niosII_simple_socket_server** project in the **Project Explorer** view.
3. On the Run menu, point to **Run As** and click **Nios II Hardware** to build the program, download it to the board, and run it.

   If the **Run Configurations** dialog box appears, click the **Target Connection** tab. Then click **Refresh Connections** and **Apply** until a board connection establishes. After the board connection is established, click **Run**.

The build process takes several minutes. After the build process completes, the Nios II SBT for Eclipse downloads the executable program to your development board.

For additional information about using the Nios II SBT for Eclipse to build projects, set up run configurations, and download programs to the board, refer to the Getting Started with the Graphical User Interface chapter of the *Nios II Software Developer's Handbook*.

Related Links

- Getting Started with the Graphical User Interface
- Quartus Prime Handbook

1.4.5 Interacting with the Nios II Simple Socket Server

The Nios II Console view displays a message with the default IP address as configured in **simple_socket_server.h**. If DHCP is enabled, the DHCP server-supplied IP address displays a message that indicates the DHCP client for the Ethernet interface acquires a DHCP IP address.

The message "Nios II Simple Socket Server starting up" displays when the NicheStack TCP/IP Stack is ready to accept commands.

After the NicheStack TCP/IP Stack is ready, you can start a telnet session to interact with the stack. To start a telnet session, follow these steps:

1. From your operating system, open a command shell.

   Note: On Windows, you can also use **Run** on the Start menu.

2. Type the following command, specifying either the static IP address or the DHCP server-provided IP address:

```
telnet <IP address> 30
```

If the connection to port 30 on the development board is successful, the menu of available commands displays in a command window. When you enter commands at the command prompt, Ethernet sends the commands over the telnet connection to a task waiting on a socket for commands. The task responds to those commands by sending instructions to another task that manipulates the LED. Figure 7 on page 13 shows the Nios II Simple Socket Server menu, along with entered commands 1, 2, 3, and Q.

Figure 7. Interacting with the Nios II Simple Socket Server Via Telnet

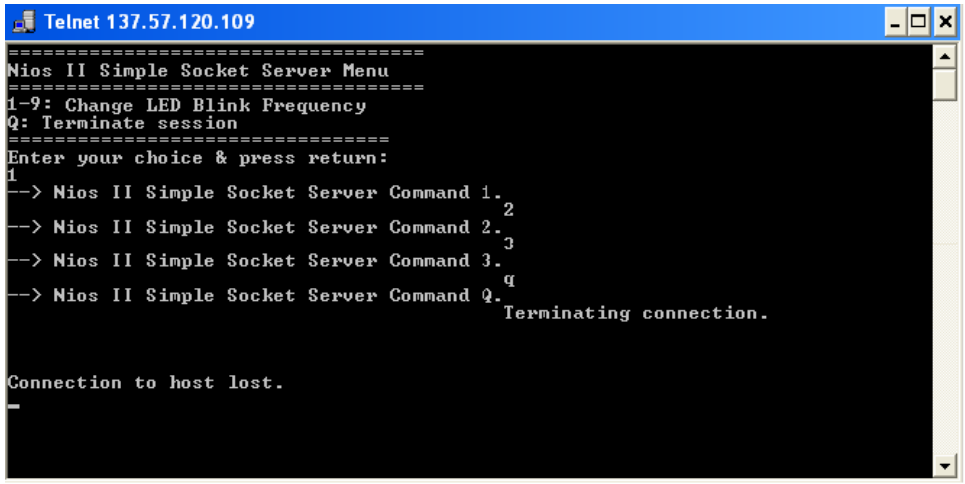
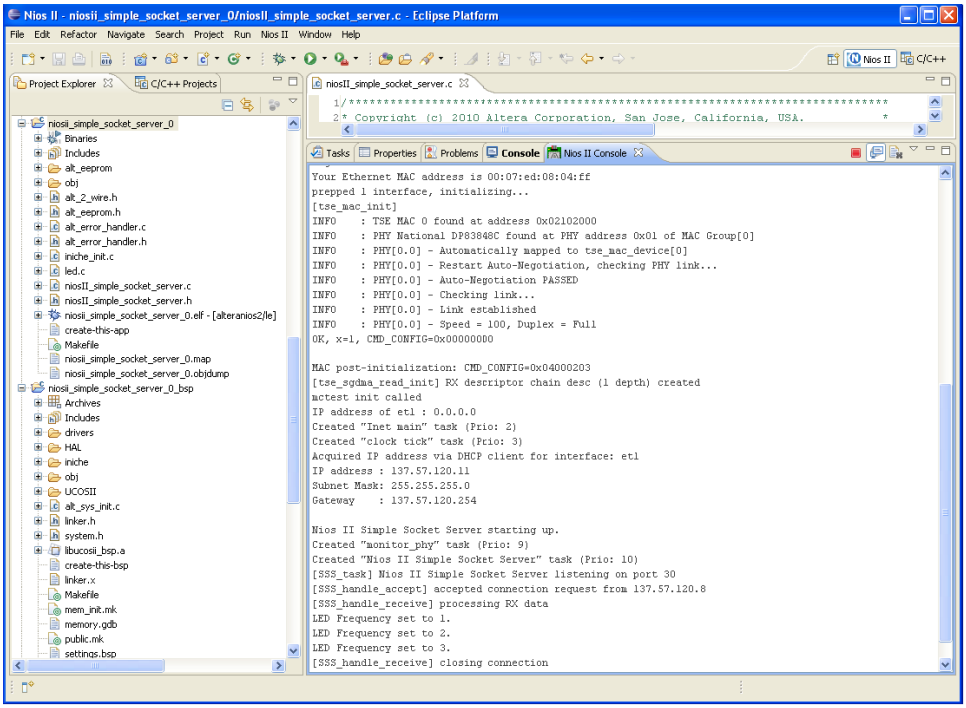


Figure 8 on page 14 shows the corresponding output that appears in the Nios II Console view.

Figure 8. Nios II Console Output During Telnet Session



To test the functionality of the Nios II Simple Socket Server, enter the following commands in the telnet session:

1. Type <n> , where <n> is a number from one through nine, to change the LED's state on your board.
2. Repeat step 1 several times, varying the number, to see the LED turn on and off.
3. Type Q to terminate the test. The socket connection on the development board terminates and the telnet command exits.

1.5 Nios II Simple Socket Server Overview

1.5.1 Software Naming Conventions

The Nios II Simple Socket Server uses capitalized acronym prefixes to identify public resources for each software module, and lowercase letters with underscores to indicate a private resource or function used internally to a software module. Table 1 on page 14 shows the software module acronym identifiers.

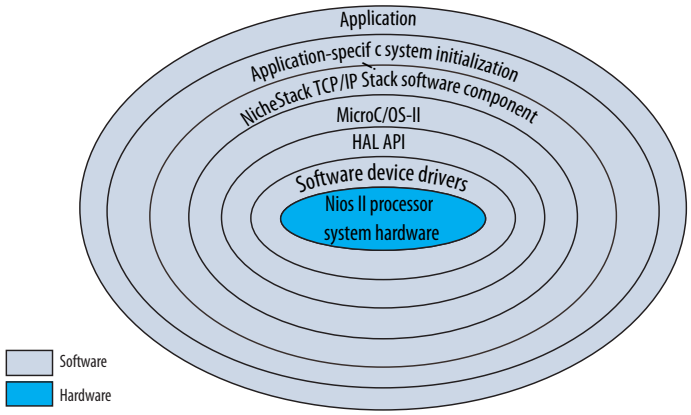
Table 1. Software Module Acronyms and Names

Acronym	Name
SSS	Nios II Simple Socket Server software module
LED	LED management software module
OS	MicroC/OS-II RTOS software component

1.5.2 Software Architecture

The onion diagram shows the architectural layers of a Nios II MicroC/OS-II software application.

Figure 9. Layered Software Model

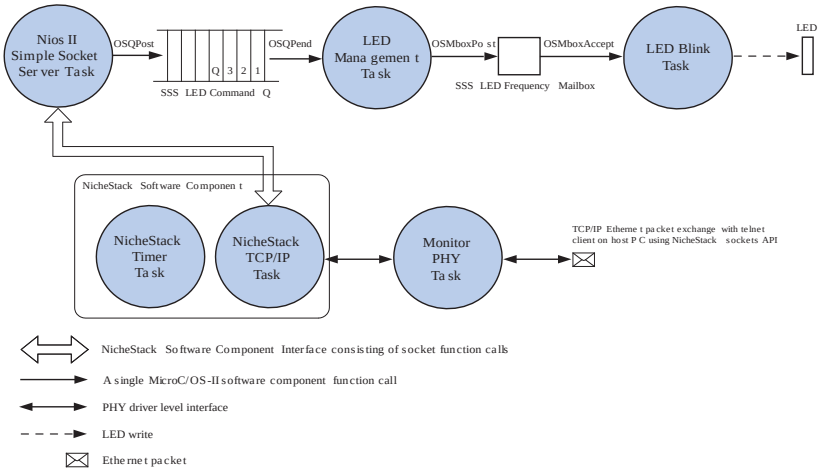


Each layer encapsulates the specific implementation details of that layer, abstracting the data for the next outer layer. The following list describes each layer:

- Nios II processor system hardware—The core of the onion diagram represents the Nios II soft core processor and hardware peripherals implemented in the FPGA.
- Software device drivers—The software device drivers layer contains the software functions that manipulate the Ethernet and hardware peripherals. These drivers know the physical details of the peripheral devices, abstracting those details from the outer layers.
- HAL API—The Hardware Abstraction Layer (HAL) application programming interface (API) provides a standardized interface to the software device drivers, presenting a POSIX-like API to the outer layers.
- MicroC/OS-II—The MicroC/OS-II RTOS layer provides multitasking and intertask communication services to the NicheStack TCP/IP Networking Stack and the Nios II Simple Socket Server.
- NicheStack TCP/IP Stack software component—The NicheStack TCP/IP Stack software component layer provides networking services to the application layer and application-specific system initialization layer through the sockets API.
- Application-specific system initialization—The application-specific system initialization layer includes the MicroC/OS-II and NicheStack TCP/IP Stack software component initialization functions invoked from main(), as well as creates all application tasks, and all the semaphores, queue, and event flag RTOS inter-task communication resources.
- Application—The outermost application layer contains the Nios II Simple Socket Server task and LED management tasks.

The "Nios II Simple Socket Server Data Flow Diagram" shows the structure of the design example described in the “Introduction” section. The sections following the figure describe the tasks in the figure.

Figure 10. Nios II Simple Socket Server Data Flow Diagram



The figure shows the state of the system after initialization. When the NicheStack TCP/IP Stack software component receives an Ethernet packet that contains an LED command sent from a telnet client program, the NicheStack TCP/IP Stack processes the incoming Ethernet packet with the TCP/IP protocol and presents the data packet to the socket server task using the sockets API. The LED management tasks then extract and post the LED command contained in the data packet to the LED command queue for processing.

Related Links

Introduction on page 3

1.5.3 MicroC-OS/II Resources

This section describes the tasks, queue, event flag, and semaphores that implement the Nios II Simple Socket Server application.

1.5.3.1 Tasks

The table below lists the MicroC/OS-II tasks that implement the Nios II Simple Socket Server application.

Table 2. MicroC/OS-II Tasks for the Nios II Simple Socket Server

The application creates the tasks listed in this table.

Task	Description
`SSSInitialTask()`	Creates an instance of all the MicroC/OS-II resources. Initializes the NicheStack TCP/IP Stack and the Nios II Simple Socket Server example RTOS structures and tasks.
`SSSSimpleSocketServerTask()`	Manages the socket server connection, and calls relevant subroutines to manage the socket connection.
`LEDManagementTask()`	Manages LEDBlinkTask, driven by commands received from a MicroC/OS-II queue, named SSSLEDCommandQ. The MicroC/OS-II mailbox, named SSSLEDFreqMailbox, passes blink rate values to the LED blink task.

The NicheStack TCP/IP Networking Stack creates two additional software component layer tasks: a main task that operates the networking stack, and a time-keeping task that the main task uses. The application creates the NicheStack TCP/IP Stack main task (tk_netmain) in the `netmain()` function with a priority of TK_NETMAIN_TPRIO. The application creates the time-keeping task (tk_nettick) in the `netmain()` call with a priority level of TK_NETTICK_TPRIO. For more information about these tasks, and how to set their priorities and stack sizes, refer to "Important NicheStack TCP/IP Stack Concepts".

Related Links

Important NicheStack TCP/IP Stack Concepts on page 17

1.5.3.2 Inter-Task Communication Resources

The following global handles (or pointers) create and manipulate your MicroC/OS-II inter-task communication resources. All the resources begin with SSS, indicating a public resource provided by the Nios II Simple Socket Server that is shared between software modules. The `SSSCreateOSDataStructs` function, invoked from `SSSInitialTask()` declares and creates these resources in **simple_socket_server.c**.

- **SSSLEDCommandQ**—SSSLEDCommandQ is a MicroC/OS-II message queue that sends commands from the simple socket server task to the development board LED control task, LEDManagementTask().

1.6 Important NicheStack TCP/IP Stack Concepts

The following topics could have a significant impact on your design.

1.6.1 Error Handling

A suite of error-handling functions defined in `alt_error_handler()` check error handling of the Nios II Simple Socket Server application, NicheStack TCP/IP Stack, and MicroC-OS/II system call error-codes. All system, socket, and application calls check for error conditions whenever an error could exist.

1.6.2 NicheStack TCP/IP Stack Default Task Creation

The NicheStack TCP/IP Stack creates one or more system level tasks during system initialization, when you call the `netmain()` function. Users have complete control over these system level tasks through a global configuration file named `ipport.h`, located in the directory structure for the BSP project, in the `iniche/src/h/nios2` folder.

You can edit the `#define` statements in `ipport.h` to configure the following options for the NicheStack TCP/IP Stack:

- Module Inclusion—Identifies which built-in NicheStack modules should be started
- Module Priority—Identifies what MicroC/OS-II priority the module task should use
- Module Stack Size—Identifies what MicroC/OS-II stack size the module should use

For more information about other NicheStack TCP/IP Stack options that can be enabled at run-time, refer to the NicheStack TCP/IP Stack documentation in the **NicheStackRef.zip** file located in the `<Nios II EDS install path>/components/altera_iniche/UCOSII/31src` directory.

In the Nios II Simple Socket Server design example, only the minimum required NicheStack TCP/IP Stack tasks have been configured to run. These tasks are as follows:

- `tk_netmain`—Initializes the stack, including networking interfaces
- `tk_nettick`—A time management task that the networking stack uses

For more information about these NicheStack TCP/IP Stack tasks, refer to the "Task Priorities in the Nios II Simple Socket Server Design" section.

Related Links

Task Priorities in the Nios II Simple Socket Server Design on page 20

1.6.3 Creating Tasks that Use the NicheStack TCP/IP Stack Sockets Interface

You must use the function call TK_NEWTASK to create any tasks that use the NicheStack networking services. You must create tasks that do not use networking services with the MicroC/OS-II function `OSTaskCreate()`.

The NicheStack Networking Stack uses the TK_NEWTASK (defined in **osportco.c**) function to launch MicroC/OS-II tasks that use the networking services. TK_NEWTASK accepts a single argument, `struct inet_taskinfo * nettask` (defined in osport.h), which specifies the task name, the MicroC/OS-II thread priority, and the stack size. You can locate these files in the `<Nios II EDS install path>/components/altera_iniche/UCOSII/src/nios2` directory. The `struct inet_taskinfo` structure is defined as follows:

```
struct inet_taskinfo {
TK_OBJECT_PTR(tk_ptr);/* pointer to static task object */
char * name;/* name of task */
TK_ENTRY_PTR(entry); /* pointer to code that starts task*/
int priority; /* MicroC/OS-II priority of the task */
```

```
int stacksize;/* size (bytes) of task's stack */
char* stackbase; /* base of task's stack */
};
```

For every networking task you create in your application, you must declare a local struct inet_taskinfo structure with the elements defined. These elements are listed in the following bullets, along with a brief explanation of their function:

- `TK_OBJECT_PTR(tk_ptr)`—A pointer to a static task object, defined for a given task by the `TK_OBJECT` macro. The NicheStack Networking Stack makes use of the `tk_ptr` element during the operation. After declaring the variable name through the `TK_OBJECT` and populating the `TK_OBJECT_PTR(tk_ptr)`, you do not need to do anything more.
- `char * name`—This element contains a character string that corresponds to the name of the task. You can set it with any character string you choose.
- `TK_ENTRY_PTR(entry)`—This element corresponds to the entry point or defined function name of the task that you want to run.
- `int priority`—The MicroC/OS-II priority level for the task.
- `int stacksize`—The MicroC/OS-II stack size for the task.
- `char* stackbase`—This element in the structure is used by the NicheStack software. You must not change the element in the structure.

In addition to declaring the struct inet_taskinfo structure, you must invoke two macro definitions: `TK_OBJECT` and `TK_ENTRY`. These macros have the following uses:

- `TK_OBJECT(name)`—Creates the static task object named name, which is used by NicheStack during operation. The static task object is also set in `TK_OBJECT_PTR(tk_ptr)`. A NicheStack naming convention for the name parameter is to set it to the string "to_", followed by the declared name of the struct inet_taskinfo instance.
- `TK_ENTRY(name)`—Used to create a declaration of the task's entry point, or function name. The name parameter is identical to the function name you specified for the task that you want to create, which must have the form void name (void). The name parameter sets the `TK_ENTRY_PTR(entry)` macro.

To create your own application tasks that use the services offered by the NicheStack TCP/IP Stack, follow these steps:

1. **Invoke Task Macros**—Include the `TK_OBJECT` and `TK_ENTRY` macros, with information about your task.
2. **Define Task Parameters**—Define your task application by filling in a local `inet_taskinfo` structure in your code.
3. **Wait for Stack Initialization**—Before launching your task, wait until the external variable `iniche_net_ready` is set to TRUE. This variable sets to FALSE at run time and changes to TRUE when the NicheStack TCP/IP Networking Stack is operational.
4. **Launch Task**—Call `TK_NEWTASK` while passing in a pointer to the `inet_taskinfo` structure for your task.

Following is a code sample for creating your own application task:

```
// Declaration of SSSNiosIISimpleSocketServerTask
void SSSNiosIISimpleSocketServerTask(void){
// task specific code
}
// Creation of NicheStack networking task
TK_OBJECT(to_ssstask);
TK_ENTRY(SSSNiosIISimpleSocketServerTask);
struct inet_taskinfo ssstask = {
&to_ssstask,
"simple socket server",
SSSNiosIISimpleSocketServerTask,
TASK_PRIORITY,
APP_STACK_SIZE,
};
while (!iniche_net_ready)
TK_SLEEP(1);
/* Create the main simple socket server task. */
TK_NEWTASK(&ssstask);
```

Networking tasks can hand off large processing jobs that are independent of networking to other tasks. This task load segmentation has the advantage of increasing control over memory usage for task stacks, as well as greater control over prioritization of jobs.

Be careful not to over utilize job distribution among several tasks at the same time, for the following reasons:

- Additional tasks require additional CPU execution time to do task context-switching.
- Limited number of priorities. Each task must have its own unique priority in MicroC/OS-II, and you do not want to run out of task priorities.

1.6.4 Task Priorities in the Nios II Simple Socket Server Design

Task priorities in the application directly affect how the application runs, or if the task functions correctly at all. The MicroC/OS-II operating system uses a unique priority number scheme for running its tasks, in which tasks assigned a lower priority number are treated as higher priority tasks. Because the NicheStack TCP/IP Stack requires the use of the MicroC/OS-II RTOS for operation, all tasks run on the system must be assigned a unique priority number. For the Nios II Simple Socket Server demo application, all tasks have been assigned non-conflicting priorities. For your own application, however, you should verify that all tasks in your system are assigned unique priority numbers at run-time.

The table below lists the tasks that might be running in your system, and the mechanism for configuring the priority of these tasks.

Table 3. Simple Socket Server Tasks and Configuration Mechanisms

Task Type	Configuration Mechanism
MicroC/OS-II internal tasks	**ucosii** settings, located on the **Main** tab of the Nios II BSP Editor
NicheStack TCP/IP Stack internal tasks	**ipport.h**, located in the **iniche/src/h/nios2** directory of your BSP project

continued...

Task Type	Configuration Mechanism
Networking initialization task	**iniche_init.c**, located in the **<tutorial_files>\software/<app files>** directory
User networking tasks (calls to TK_NEWTASK)	Created in the user application code
User non-networking tasks (calls to OSTaskCreate)	Created in the user application code

The following sections discuss the priorities of the tasks in the Nios II Simple Socket Server design:

1.6.4.1 MicroC/OS-II Internal Tasks

The Nios II Simple Socket Server application has been configured to not use any MicroC/OS-II internal tasks.

1.6.4.2 NicheStack TCP/IP Stack Internal Tasks

TK_NETMAIN_TPRIO, defined in **ipport.h**, sets the priority to a value of 2 for the main NicheStack TCP/IP Stack task, launched by `netmain()`. This task implements the core functionality of the NicheStack TCP/IP Stack. To maximize the TCP/IP packet throughput rate, the priority of this task should be higher than application tasks that use the NicheStack TCP/IP Networking Stack.

TK_NETTICK_TPRIO, defined in **ipport.h**, sets the priority to a value of 3 for the NicheStack TCP/IP Stack time-keeping task, launched by `netmain()`. The NicheStack TCP/IP Stack uses this task to keep track of time-based events in the networking stack. It is recommended that you set the priority of this task to one priority level lower than TK_NETMAIN_TPRIO.

1.6.4.3 Networking Initialization Task

SSS_INITIAL_TASK_PRIORITY is set to a value of 5 for the first task that MicroC/OS-II runs. This task creates the resources and all the other tasks before deleting itself. This task is given a high priority, not due to its high time-period rate or low latency requirement, but to create all the real-time operating system resources and tasks before the other tasks start using the resources.

1.6.4.4 User Networking Tasks

SSS_SIMPLE_SOCKET_SERVER_TASK_PRIORITY is set to a value of 10, a priority that is lower than the consumer task `LEDManagementTask()`. The priority of this application task is set lower than all of the software components' system service tasks. This practice allows for the best overall scheduling latency, because the software component tasks are designed to operate for as short a period of time as possible.

1.6.4.5 User Non-Networking Tasks

LED_MANAGEMENT_TASK_PRIORITY is set to a value of 7. This task's function is to receive LED command messages from the `SSSSimpleSocketServerTask`.

1.6.5 Task Stack Size

Task stack space requirements depend on how the Nios II processor, HAL, RTOS, and individual software components are configured. A quick empirical check of the `Stk[]` array values at runtime, through the Nios II SBT for Eclipse memory window, is an easy way to examine the top of a task stack. Examination of a task's `Stk[]` array reveals differing values representing the used portion of the stack followed by multiple zeros where the stack has not yet reached. The number of zeros until the beginning of the next adjacent task stack shows how deep the stack has grown since the last system reset.

All tasks that make run-time library calls have space allocated from the top of the stack for the approximately 900-byte `_reent` structure. Each task has its own copy of the structure positioned on the task's stack. The size of this structure alone reduces the amount of available stack space.

For more information about the _reent structure, refer to the "The Newlib ANSI C Standard Library" and the "Implementing MicroC/OS-II Projects for the Nios II Processor" sections of the MicroC/OS-II Real-Time Operating System chapter of the *Nios II Software Developer's Handbook*.

Related Links

MicroC/OS-II Real-Time Operating System

1.7 Where to Go Next

This example is easily expandable to handle multiple TCP connections on a single port. The `SSSSimpleSocketServerTask()` task could be modified to create separate socket connection instance tasks to handle multiple telnet connections.

There are many uses for an Ethernet connection in an embedded system. A connection to the Internet can allow the addition of many powerful features for any embedded product, such as remote configurability using a web browser, or remote software upgrade for corrections or feature enhancements to a product already in the field.

1.8 Hardware Setup Details

To complete this tutorial, you must have the Nios II SBT for Eclipse installed, and your development board must be connected to a host PC on both the Ethernet and USB/JTAG ports.

Note: For information about download cables and drivers, refer to the Cable and Adapter Drivers Information page.

The Nios II Ethernet standard hardware design examples for development boards include the Ethernet device required by this NicheStack tutorial. The Ethernet device included in these design examples, along with the physical MAC/PHY on your development boards, is the Triple Speed Ethernet MAC peripheral. The Ethernet peripheral base address settings for the design examples are defined in **system.h**.

The app file **tse_my_system.c** reflects the IP names in the **system.h** file. If you are not using the default hardware project then you may have to update the following:

- The TSE IP used in the example project is TSE_0_TSE
- IP name for the TX MSGDMA is TSE_0_DMA_TX
- IP name for the RX MSGDMA is TSE_0_DMA_RX

Related Links

Cable and Adapter Drivers Information

1.8.1 Network Connection

If you are using a DHCP server to assign IP addresses, connect your development board to your Ethernet network.

If the development board is connected directly to your PC with a crossover Ethernet cable, or a DHCP server is not available, specify the IP addresses manually in **simple_socket_server.h**.

The default IP addresses in **simple_socket_server.h** are set to all zeros so the DHCP server packets can pass through secure routers. If you are not using a DHCP server, specify valid static addresses, such as an IP address of 192.168.1.234, with a gateway of 192.168.1.1 and a subnet mask of 255.255.255.0.

Be sure to turn the **enable_dhcp_client** setting on or off accordingly on the **Software Packages** tab of the BSP Editor. For details, refer to the "Configuring the BSP" section.

Related Links

Configuring the BSP on page 8

1.9 Document Revision History

The following table shows the revision history for this document.

Table 4. Document Revision History

Date	Version	Changes
October 2017	2017.10.09	Replaced Qsys with Platform Designer.
December 2016	2016.12.19	• Updated for the Arria 10 SoC Development Kit. Obsolete boards: • Nios II Embedded Evaluation Kit (NEEK), Cyclone III Edition • Embedded Systems Development Kit (ESDK), Cyclone III Edition
June 2011	3.0	Revised for Quartus II Software 11.0 release.
May 2010	2.0	Revised for Nios II Software Build Tools for Eclipse.
January 2007	1.0	Initial release.