

Nios[®] II

Using the NicheStack TCP/IP Stack

Nios II Edition Tutorial



101 Innovation Drive
San Jose, CA 95134
(408) 544-7000
<http://www.altera.com>

Copyright © 2007 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, maskwork rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

TU-01001-1.0





About this Tutorial	v
Revision History	v
How to Find Information	v
How to Contact Altera	vi
Typographic Conventions	vi
Chapter 1. Using the NicheStack TCP/IP Stack	
Introduction	1-1
Hardware & Software Requirements	1-2
Tutorial Design Files	1-2
Software Development Flow	1-4
Create a New Nios II IDE Project	1-4
Configure the System Library	1-6
Examine the Simple Socket Server Project Files	1-10
Build & Run the Simple Socket Server Project	1-11
Interacting with the Simple Socket Server	1-13
Simple Socket Server Design Overview	1-16
Nios II Software Architecture	1-17
Software Design Naming Convention	1-19
MicroC-OS/II Resources	1-20
Tasks	1-20
Inter-Task Communication Resources	1-21
NicheStack TCP/IP Stack Initialization	1-21
Simple Socket Server Commands and Structures	1-22
LED Command Definitions	1-22
SSS_Socket Structure	1-23
Simple Socket Server Implementation Details	1-23
Important NicheStack TCP/IP Stack Concepts	1-24
Error Handling	1-24
NicheStack TCP/IP Stack Default Task Creation	1-24
Creating Tasks that Use the NicheStack TCP/IP Stack Sockets Interface	1-25
Task Priorities in the Simple Socket Server Design	1-28
MicroC/OS-II Internal Tasks	1-29
NicheStack TCP/IP Stack Internal Tasks	1-29
Networking Initialization Task	1-29
User Networking Tasks	1-29
User Non-Networking Tasks	1-29
Task Stack Size	1-30
Where to Go Next	1-30

Appendix A. Hardware Setup Details

Introduction A-1
Network Connection A-1

Appendix B. Upgrading from lwIP to NicheStack TCP/IP Stack

Introduction B-1
Issues in Upgrading B-1
 New Method for TCP/IP Stack Initialization B-1
 New Method for Notification that the TCP/IP Stack Is Ready B-1
 New Method for Creation of Tasks that Will Use TCP/IP Stack B-2
 Different Customization Process and Include Files B-2
 New Function Prototype and Parameter Type Definitions for `Network_utilities.c` B-2
 New BOOLEAN Type Definition B-3



About this Tutorial

This tutorial introduces you to the Nios[®] II integrated development environment (IDE) and the MicroC/OS-II and NicheStack TCP/IP Stack development flow. It shows you how to use the Nios II IDE to create a new Nios II C/C++ project that configures, builds, and runs a MicroC/OS-II and NicheStack TCP/IP Stack program on the Nios development board.

Revision History The table below displays the revision history for this tutorial.

Date	Version	Changes Made
January 2007	1.0	Initial release.

How to Find Information

- The Adobe Acrobat Find feature allows you to search the contents of a PDF file. Click the binoculars toolbar icon to open the **Find** dialog box.
- Bookmarks serve as an additional table of contents.
- Thumbnail icons, which provide miniature previews of each page, provide a link to the pages.
- Numerous links, shown in green text, allow you to jump to related information.

How to Contact Altera

For the most up-to-date information about Altera products, go to the Altera® worldwide web site at www.altera.com. For technical support on this product, go to www.altera.com/mysupport. For additional information about Altera products, consult the sources shown below.

Information Type	USA & Canada	All Other Locations
Technical support	www.altera.com/mysupport/	www.altera.com/mysupport/
	(800) 800-EPLD (3753) (7:00 a.m. to 5:00 p.m. Pacific Time)	+1 408-544-8767 7:00 a.m. to 5:00 p.m. (GMT -8:00) Pacific Time
Product literature	www.altera.com	www.altera.com
Altera literature services	literature@altera.com (1)	literature@altera.com (1)
Non-technical customer service	(800) 767-3753	+ 1 408-544-7000 7:00 a.m. to 5:00 p.m. (GMT -8:00) Pacific Time
FTP site	ftp.altera.com	ftp.altera.com








Note to table:

(1) You can also contact your local Altera sales office or sales representative.

Typographic Conventions

This document uses the typographic conventions shown below.

Visual Cue	Meaning
Bold Type with Initial Capital Letters	Command names, dialog box titles, checkbox options, and dialog box options are shown in bold, initial capital letters. Example: Save As dialog box.
bold type	External timing parameters, directory names, project names, disk drive names, filenames, filename extensions, and software utility names are shown in bold type. Examples: f_{MAX} , lqdesigns directory, d: drive, chiptrip.gdf file.
<i>Italic Type with Initial Capital Letters</i>	Document titles are shown in italic type with initial capital letters. Example: <i>AN 75: High-Speed Board Design</i> .
<i>Italic type</i>	Internal timing parameters and variables are shown in italic type. Examples: <i>t_{PIA}</i> , <i>n + 1</i> . Variable names are enclosed in angle brackets (< >) and shown in italic type. Example: < <i>file name</i> >, < <i>project name</i> >.pof file.
Initial Capital Letters	Keyboard keys and menu names are shown with initial capital letters. Examples: Delete key, the Options menu.
“Subheading Title”	References to sections within a document and titles of online help topics are shown in quotation marks. Example: “Typographic Conventions.”

Visual Cue	Meaning
Courier type	Signal and port names are shown in lowercase Courier type. Examples: <code>data1</code> , <code>tdi</code> , <code>input</code> . Active-low signals are denoted by suffix <code>n</code> , e.g., <code>resetn</code> . Anything that must be typed exactly as it appears is shown in Courier type. For example: <code>c:\qdesigns\tutorial\chiptrip.gdf</code> . Also, sections of an actual file, such as a Report File, references to parts of files (e.g., the AHDL keyword <code>SUBDESIGN</code>), as well as logic function names (e.g., <code>TRI</code>) are shown in Courier.
1., 2., 3., and a., b., c., etc.	Numbered steps are used in a list of items when the sequence of the items is important, such as the steps listed in a procedure.
	Bullets are used in a list of items when the sequence of the items is not important.
	The checkmark indicates a procedure that consists of one step only.
	The hand points to information that requires special attention.
	A caution calls attention to a condition or possible situation that can damage or destroy the product or the user's work.
	A warning calls attention to a condition or possible situation that can cause injury to the user.
	The angled arrow indicates you should press the Enter key.
	The feet direct you to more information on a particular topic.

Introduction

This tutorial familiarizes you with the NicheStack TCP/IP Stack – Nios® II Edition (NicheStack TCP/IP Stack) software component included in your Nios II development kit. Topics covered include:

- 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 IDE to develop programs with the NicheStack TCP/IP Stack software component

The Nios II IDE offers software designers a rich development platform for Nios II applications. The Nios II IDE contains the MicroC/OS-II real-time operating system (RTOS) and NicheStack TCP/IP Stack software component, providing designers with the ability to build networked embedded systems applications for the Nios II processor quickly. 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 design 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 Nios II development board with a simple TCP/IP socket server example. This socket server example receives commands sent over a TCP/IP connection and manipulates LEDs 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.

Details on setup requirements for the NicheStack TCP/IP Stack software component and the MicroC/OS-II real-time operating system are covered.



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



For complete details on MicroC/OS-II for the Nios II processor, refer to the *MicroC/OS-II Real-Time Operating System* chapter in the *Nios II Software Developer's Handbook*.



For complete details on NicheStack TCP/IP Stack initialization and configuration for the Nios II processor, refer to the *Ethernet & the NicheStack TCP/IP Stack – Nios II Edition* chapter in the *Nios II Software Developer's Handbook*.

Hardware & Software Requirements

This tutorial requires the following hardware and software:

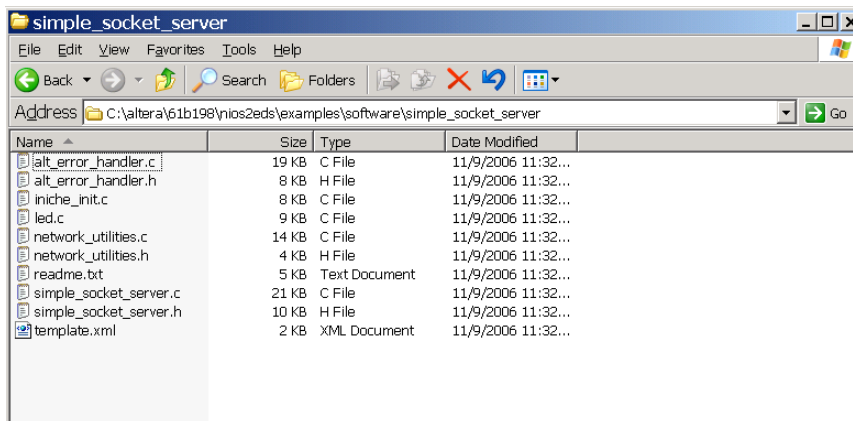
- Quartus® II software version 6.1 or later
- Nios II development kit version 6.1 or later
- Nios development board, Stratix® II Edition, Stratix Edition, Stratix Professional Edition, Cyclone® II Edition, or Cyclone Edition
- Altera® USB-Blaster™ cable
- RJ-45 connected Ethernet cable on the same network as the PC development host

To complete this tutorial, you must have the Nios II IDE installed, and your Nios development board must be connected to a host PC. Refer to Appendix A, *Hardware Setup Details*, for detailed hardware-setup instructions.

Tutorial Design Files

The tutorial software design is a C source code file collection, provided with the Nios II development kit. You will find the NicheStack TCP/IP Stack tutorial software design files in the `<Nios II kit path>\examples\software\simple_socket_server` directory.

Figure 1–1. Simple Socket Server: Using the NicheStack TCP/IP Stack Tutorial Software Design Files



The Nios II development kit includes the reference hardware designs. The software design will work with either the **standard** or **full-featured** hardware reference design.

After you install the Nios II development kit, you can find the hardware design files in the Nios II development kit directory structure. For demonstration purposes, this tutorial uses the Nios II development kit, Stratix Professional Edition, featuring the Stratix EP1S40 device, and uses the Verilog full-featured hardware reference design. The hardware reference design files are located in the following directory:

```
<Nios II kit installation path>\examples\verilog\niosII_stratix_1s40\  
full_featured
```

Throughout this tutorial, where path names are listed, replace **nios_II_stratix_1s40** with the matching directory for your particular Nios development board, **verilog** with **vhdl**, and **full_featured** with **standard** where appropriate to match your FPGA device, hardware description language, and hardware reference design selection.

The following list of eight source-code files make up the Simple Socket Server application for this tutorial:

- **alt_error_handler.c**—Contains the implementation of three error handlers, one each for the Simple Socket Server (SSS), 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.
- **led.c**—Contains LED management tasks.
- **iniche_init.c**—Defines `main()`, which initializes MicroC/OS-II and NicheStack TCP/IP Stack.
- **network_utilities.c**—Defines functions to manipulate the MAC and IP addresses.
- **network_utilities.h**—Defines the function prototype to manipulate the MAC address.
- **simple_socket_server.c**—Defines all of the tasks and functions that utilize the NicheStack TCP/IP Stack sockets interface, and creates all of the MicroC/OS-II resources.
- **simple_socket_server.h**—Defines all of the task prototypes, task priorities, and other MicroC/OS-II resources used in this tutorial.

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 IDE C/C++ application project with the **Simple Socket Server** project template.
2. Configuring the system library project, including MicroC/OS-II and the NicheStack TCP/IP Stack software component.
3. Building the application project.
4. Running (and debugging where necessary) the application project.

Create a New Nios II IDE Project

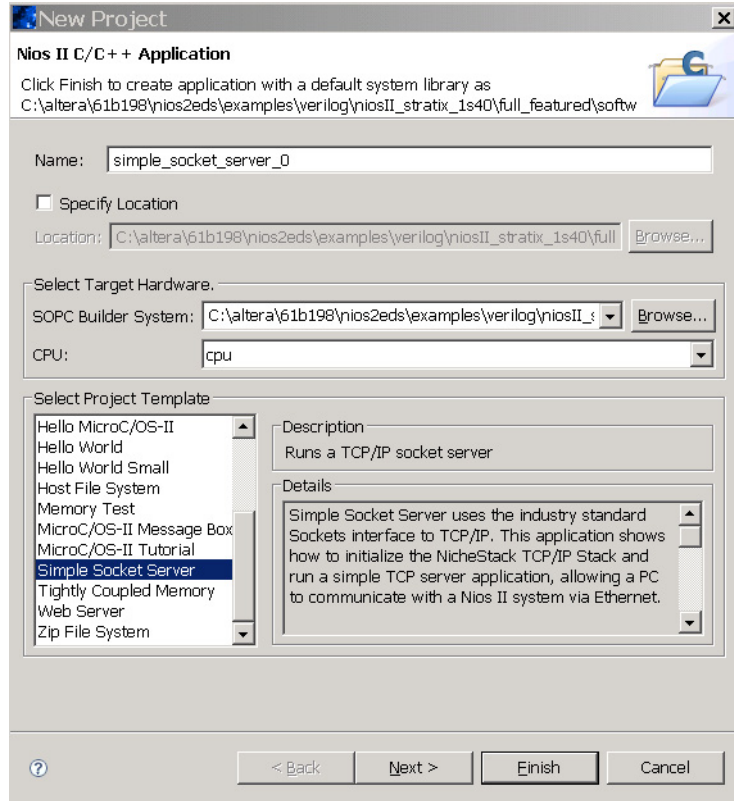
In this section, you create a new Nios II IDE project using a project template. Perform the following steps:

1. To start the Nios II IDE, on the Start menu, point to Programs, point to Altera, point to Nios II EDS 6.1, and click **Nios II 6.1 IDE**.
2. On the File menu, point to New and click **Nios II C/C++ Application**. The first page of the **New Project** wizard appears.
3. Under **Select Project Template**, select **Simple Socket Server**. The project name and project path are filled in for you automatically.
4. Under **Select Target Hardware**, click **Browse**.
5. In the **Browse** dialog box, browse to the full_featured hardware reference design directory for the Nios development board that you are targeting, for example, `<6.1_installation_path>\nios2eds\examples\verilog\niosII_stratix_1s40\full_featured`.
6. Select the SOPC Builder system file (.ptf) for the full_featured design, for example, `full_1s40.ptf`.
7. Click **Open**.

The **Browse** dialog box closes and you are returned to the **New Project** wizard. As shown in [Figure 1-2](#), the **SOPC Builder System** box under **Select Target Hardware** contains the path to the SOPC Builder system file (.ptf) for the full_featured example design. Additionally, the CPU box contains the name of one of the available Nios II CPUs as defined in SOPC Builder.

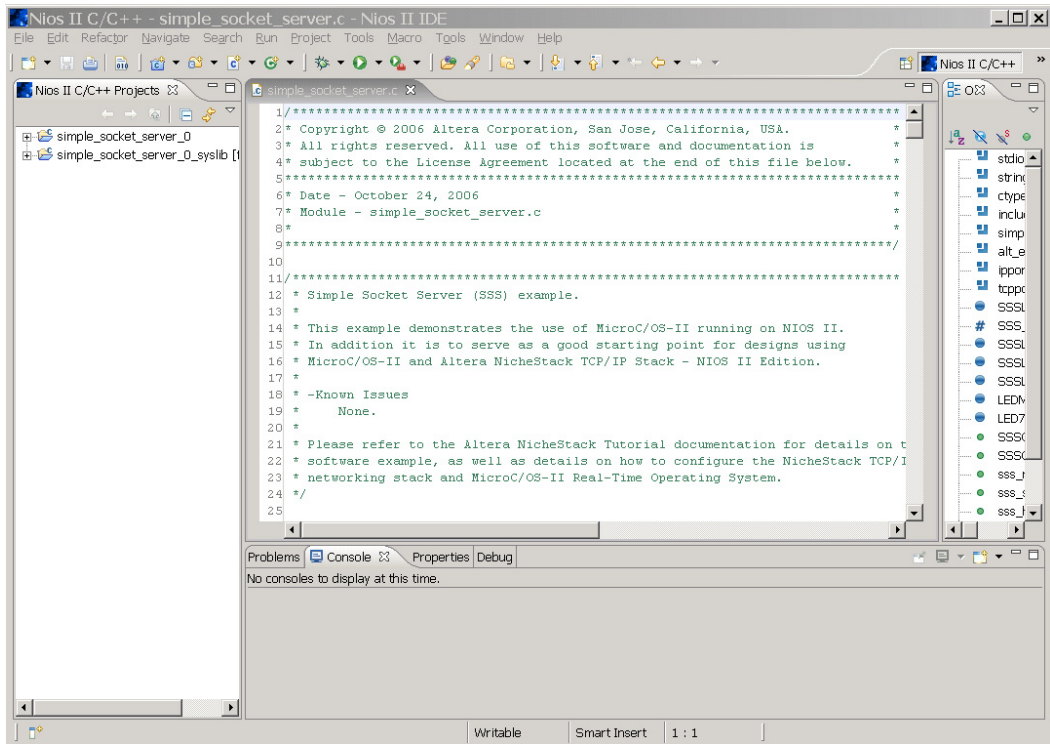
The Nios II development board hardware reference designs contain a single CPU. The single CPU is selected automatically when you choose the SOPC Builder System. Keep the default CPU as displayed in the CPU selection box.

Figure 1–2. New Project Wizard



8. Click **Finish** to complete creation of the application and system library projects.

The wizard creates two projects in the Nios II IDE **C/C++ Projects** tab, as shown in [Figure 1–3](#).

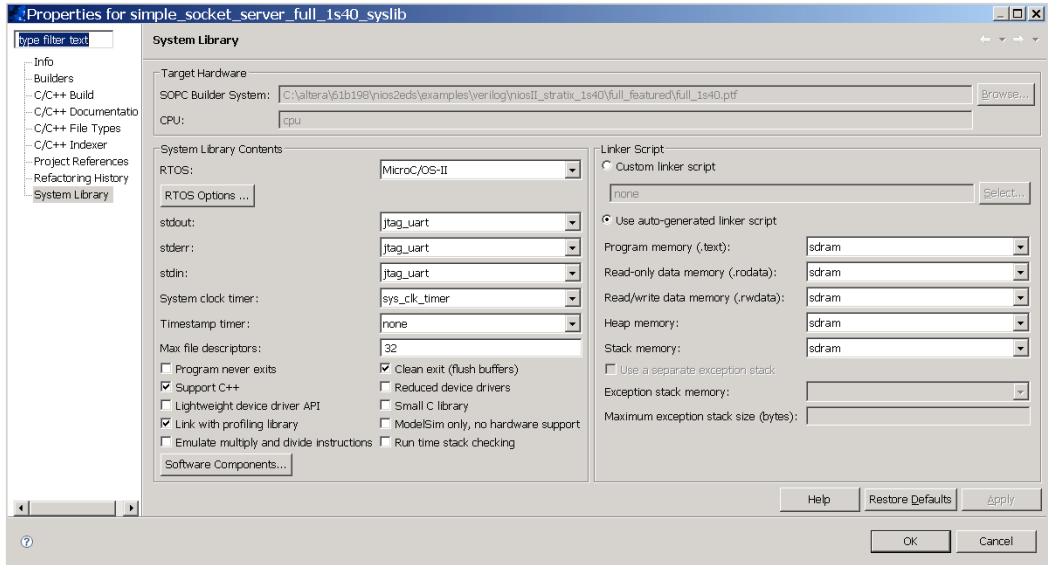
Figure 1–3. New Projects in the Nios II C/C++ Projects Perspective

Configure the System Library

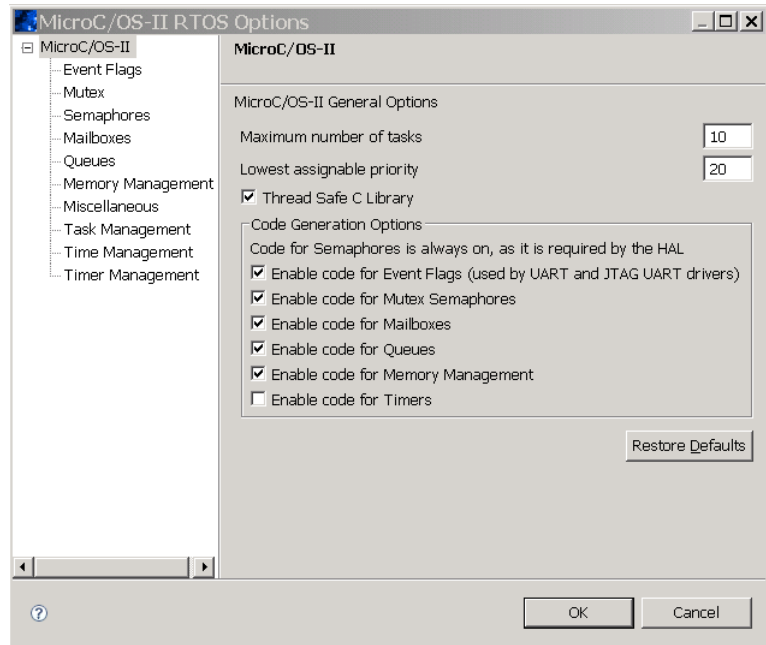
After you create a new system library, you may want to configure it further (for example, defining `stdin`, `stdout`, `stderr`, and other parameters). Refer to the Nios II IDE online *Nios II Software Development Tutorial* for more details. For this NicheStack tutorial, you must configure the MicroC/OS-II and NicheStack TCP/IP Stack software components. Perform the following steps to configure the MicroC/OS-II kernel:


1. With a left mouse click, select the **syslib** project.
2. With a right mouse click, select **System Library Properties**.
3. Read the License Notification, and then click **OK**.

Figure 1–4. System Library Properties Page



4. In the **RTOS** drop-down menu, select **MicroC/OS-II**.
5. Read the License Notification, and then click **OK**.
6. Click **RTOS Options** under **RTOS**. The **MicroC/OS-II RTOS Options** dialog box opens, as shown in [Figure 1–5](#).

Figure 1–5. MicroC/OS-II RTOS Options

- Click the  icon in the left panel to expand the contents under MicroC/OS-II, as shown in [Figure 1–5](#). The MicroC/OS-II kernel is highly configurable. The options you select in this dialog box determine which MicroC/OS-II options are included in the binary image. Examine the options you can select by clicking each of the options categories under MicroC/OS-II in the left panel of the screen.



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



For details 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*.


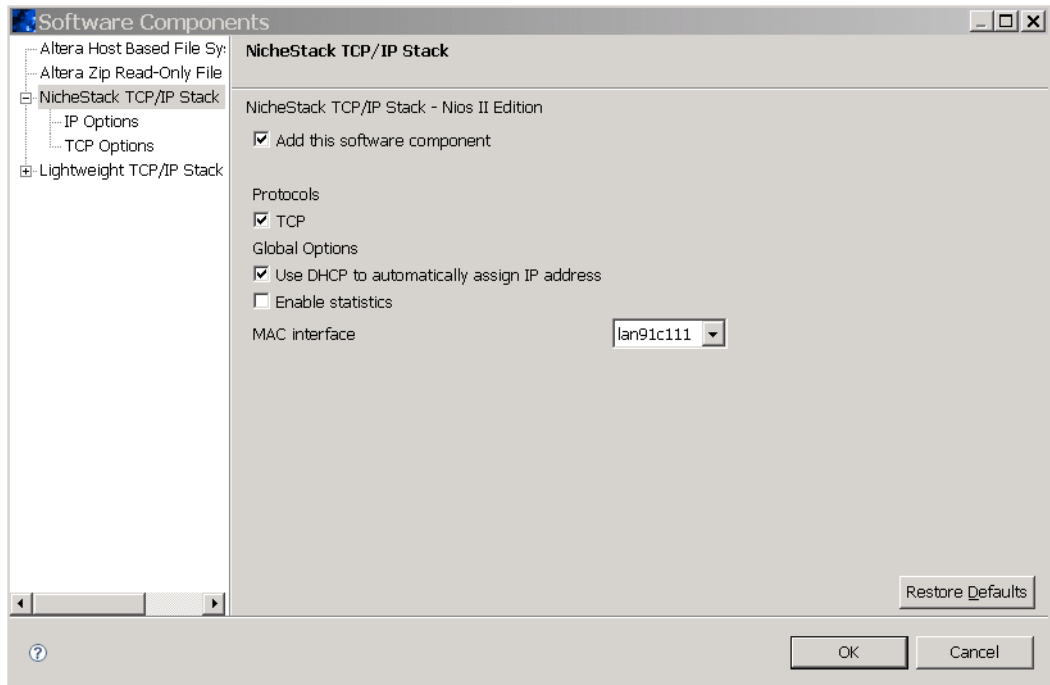

8. For this tutorial, choose the default settings and click **OK**. You are returned to the **System Library** options properties page.
9. Click **Software Components**.
10. Select **NicheStack TCP/IP Stack** in the left panel.
11. Read the License Notification, and then click **OK**.
12. Under **NicheStack TCP/IP Stack – Nios II Edition**, turn on **Add this software component**.
13. If a DHCP Server is available on your network, turn on the **Use DHCP to automatically assign IP address** option. If no DHCP server is available, make sure the option is turned off. Instead, provide IP addresses, specified in `simple_socket_server.h`, for the Nios development board, the gateway, and the network mask.
14. Click the  icon in the left panel to expand the contents under NicheStack TCP/IP Stack, as shown in [Figure 1–6](#).

Figure 1–6. NicheStack TCP/IP Stack Options

 Do not enable the Lightweight TCP/IP Stack (lwIP). Use of the lwIP stack is not compatible with simultaneous use of the NicheStack TCP/IP Stack.

15. Click **OK** to complete the configuration of the NicheStack TCP/IP Stack.
16. Click **OK** in the System Library Properties page to complete configuration of the system library.

Examine the Simple Socket Server Project Files


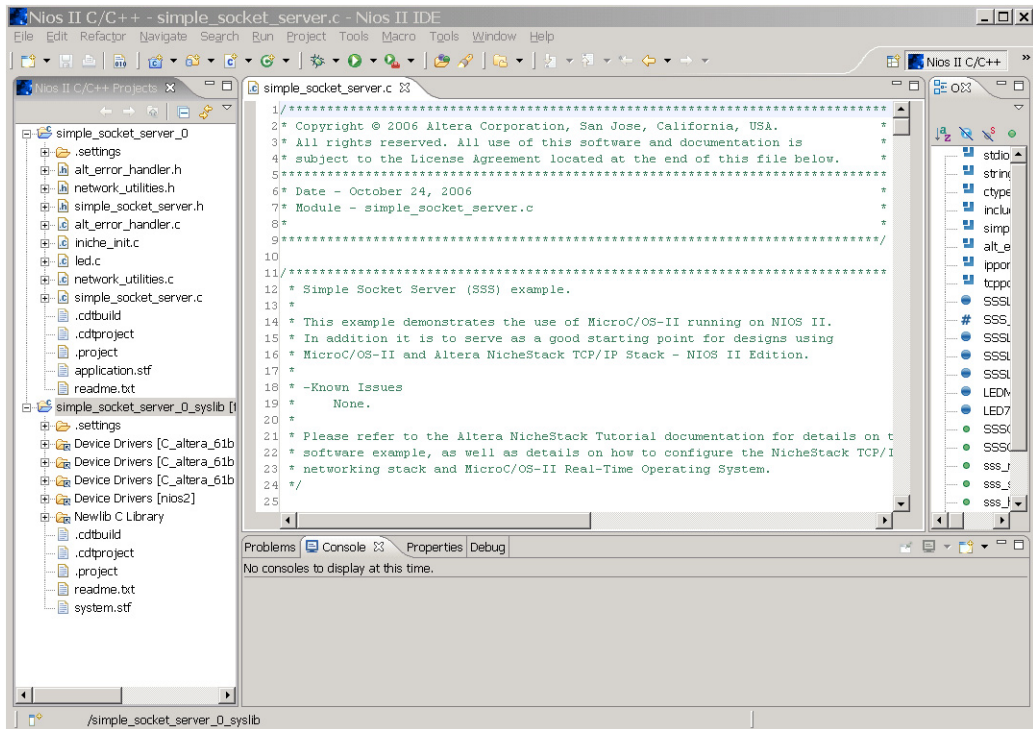
You can click the  icon to the left of the `simple_socket_server_0` folder icon to view the source files, as shown in [Figure 1–7](#).

Figure 1–7. Simple Socket Server Project Files



You have finished creating and configuring both the **simple_socket_server_0** and the associated system library project. You are now ready to build and run the example described in the following section.



For more information about building and running programs with the Nios II IDE, refer to the *Nios II Software Development Tutorial* in the Nios II IDE online help.

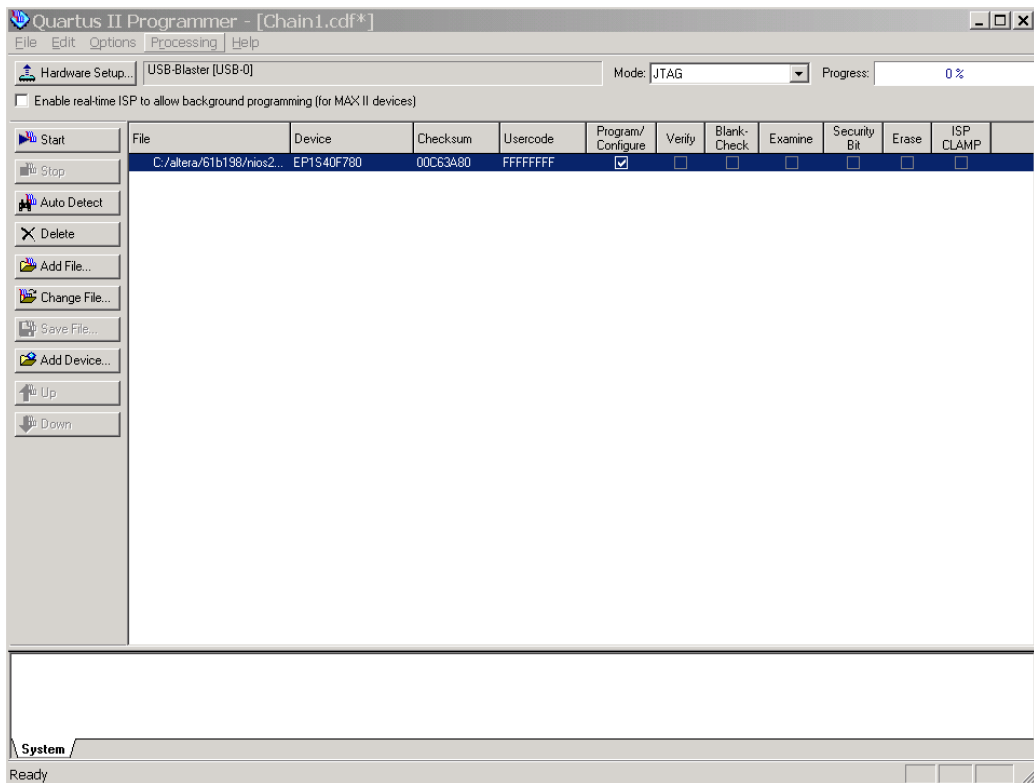
Build & Run the Simple Socket Server Project

In this section, you will run the example design on a Nios development board. You will build the application, configure the development board with the full-featured hardware design, and download the executable software file. Perform the following steps:

1. On the Tools menu, click **Quartus II Programmer**.

-
2. In the **Quartus II Programmer** dialog box, on the File menu, click **Open**.
3. Select the FPGA configuration file (.sof), for example, **full_featured.sof**.
4. Click **Open**. You return to the **Quartus II Programmer** dialog box.
5. Turn on the **Program/Configure** option, as shown in [Figure 1-8](#).

Figure 1-8. Quartus II Programmer Dialog Box



-
-
-
-
-
6. Click **Start** to configure the FPGA on the development board.
7. On the File menu, click **Exit** to close the Quartus II Programmer, or minimize the Quartus II Programmer, and return to the Nios II IDE. If you receive a message that asks if you want to save the changes to the **chain1.cdf** file, click **No**.

8. In the Nios II IDE, select the **simple_socket_server_0** project in the **Nios II C/C++ Projects** tab.
9. 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.

The build process takes several minutes. After the Nios II IDE builds the executable, it attempts to download the image to your Nios development board using the default run configuration.



For additional information about using the Nios II IDE to build projects, set up run configurations, and download programs to the board, refer to the *Nios II Software Development Tutorial* within the Nios II IDE online help.

Interacting with the Simple Socket Server

After the image is downloaded to your Nios development board, the seven-segment LED banks flash in a random pattern. The STDOUT configured console 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 is displayed after a message that indicates a DHCP IP address has been acquired by the DHCP client for the Ethernet interface.

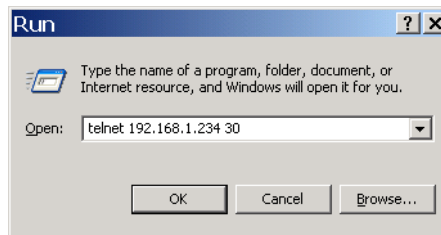
The message “Simple Socket Server starting up” is displayed when the NicheStack TCP/IP Stack is ready to accept commands.

To start a telnet session, click **Run** in the Windows Start menu. In the **Run** dialog box, enter the following command:

```
telnet <IP_address> 30
```

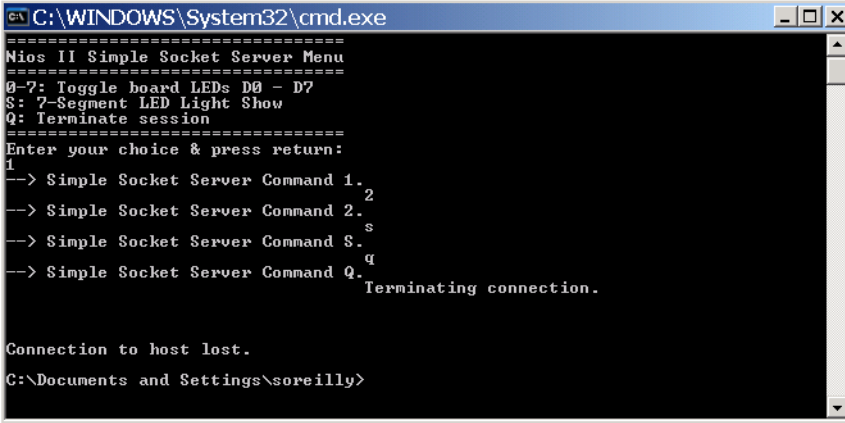
Specify either the static IP address or the DHCP server-provided IP address, as shown in [Figure 1–9](#). Click **OK**.

Figure 1–9. Connecting to the Simple Socket Server



If the connection to port 30 on the Nios development board is successful, the menu of available commands is displayed in a DOS command window, as shown in [Figure 1-10](#).

Figure 1-10. Interacting with the Simple Socket Server Via Telnet



```
C:\WINDOWS\System32\cmd.exe
=====
Nios II Simple Socket Server Menu
=====
0-7: Toggle board LEDs D0 - D7
S: 7-Segment LED Light Show
Q: Terminate session
=====
Enter your choice & press return:
1
--> Simple Socket Server Command 1.
2
--> Simple Socket Server Command 2.
s
--> Simple Socket Server Command S.
q
--> Simple Socket Server Command Q.
Terminating connection.

Connection to host lost.
C:\Documents and Settings\soreilly>
```

Commands entered at the DOS command prompt are sent over the telnet connection via Ethernet to a task waiting on a socket for commands. This task responds to those commands by sending instructions to another task that manipulates the LEDs.

[Figure 1-10](#) shows the Simple Socket Server menu, along with commands 1, 2, S, and Q. [Figure 1-11](#) shows the corresponding output on the Nios II Terminal Window during the telnet session.

Figure 1–11. Nios II Terminal Window Output During Telnet Session

The screenshot shows the Nios II IDE with the source code for `simple_socket_server.c` in the editor and the terminal window displaying the output of a telnet session. The source code includes copyright information for Altera Corporation (2006) and the module name `simple_socket_server.c`. The terminal output shows the Nios II hardware configuration, software license reminder for the NicheStack TCP/IP, and the execution of the Simple Socket Server. The server starts listening on port 30, receives a connection from 137.57.234.135, and processes RX data. It sets LED_PIO_BASE values to 2 and 6, and then closes the connection.

```

1/*****
2* Copyright © 2006 Altera Corporation, San Jose, California, USA.
3* All rights reserved. All use of this software and documentation is
4* subject to the License Agreement located at the end of this file below.
5/*****
6* Date - October 24, 2006
7* Module - simple_socket_server.c
8*
9/*****/

Problems Console Properties Debug
simple_socket_server_0 Nios II HW configuration [Nios II Hardware] Nios II Terminal Window (11/15/06 10:53 AM)
nios2-terminal: (Use the IDE stop button or Ctrl-C to terminate)

***** Software License Reminder *****
This software project uses an unlicensed version of the NicheStack TCP/IP
Network Stack - Nios II Edition. If you want to ship resulting object
code in your product, you must purchase a license for this software from
Altera. For information go to: "http://www.altera.com/nichestack"
*****
InterNiche Portable TCP/IP, v3.0

Copyright 1996-2003 by InterNiche Technologies. All rights reserved.
Your Ethernet MAC address is 00:07:ed:ff:e3:d7
prepared 1 interface, initializing...
Created "inet main" task (Prio: 2)
Created "clock tick" task (Prio: 3)
smsc91c11 Auto-negotiation: 100 Mbps, Full Duplex
SMSC ethernet Rev: 0x3391, ram: 8192
IP address of et1 : 137.57.234.136
Acquired IP address via DHCP client for interface: et1
IP address : 137.57.235.9
Subnet Mask: 255.255.255.0
Gateway : 137.57.235.254

Simple Socket Server starting up
[sss_task] Simple Socket Server listening on port 30
Created "simple socket server" task (Prio: 4)
[sss_handle_accept] accepted connection request from 137.57.234.135
[sss_handle_receive] processing RX data
Value for LED_PIO_BASE set to 2.
Value for LED_PIO_BASE set to 6.
[sss_handle_receive] closing connection

```

To test the functionality of the Simple Socket Server, enter commands in the telnet session. Entering a number from zero through seven, followed by a return, causes the corresponding LEDs D0 – D7 to toggle on or off on the Nios development board. Entering the letter S stops the random blinking LED pattern on the seven-segment LED bank. Entering the S command again restarts the light show.

To reproduce the specific run-time behavior shown in [Figures 1–10 and 1–11](#), do the following at the DOS command prompt:

1. Type 1 ↵

The LED D1 is toggled. The Nios II Terminal Window displays two messages:

```
processing RX data
Value for LED_PIO_BASE set to 2.
```

2. Type 2 ↵

The LED D2 is toggled. The Nios II Terminal Window displays the following message:

```
Value for LED_PIO_BASE set to 6.
```

The value for LED_PIO_BASE is displayed on the LEDs in binary format.

3. Type the letter S ↵

The seven-segment LED display stops flashing.

4. Type the letter Q ↵

The socket connection on the Nios development board is terminated and the telnet command exits.

Simple Socket Server Design Overview

The following sections describe the Simple Socket Server design:

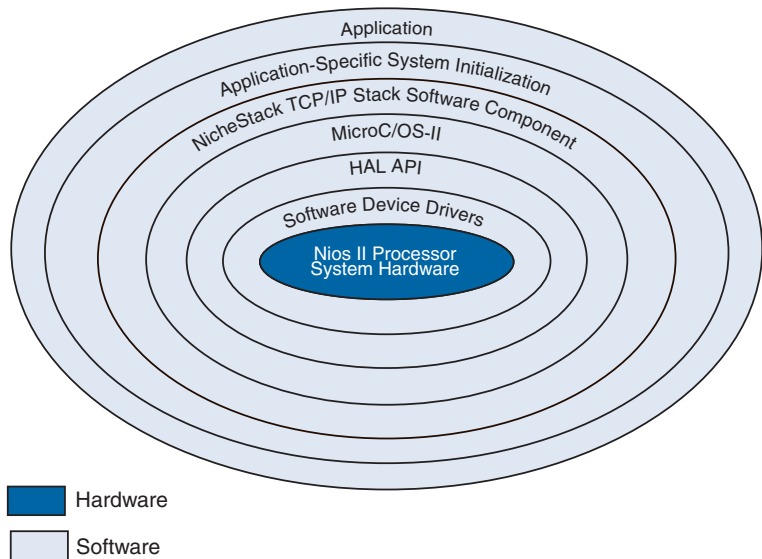
- [“Nios II Software Architecture” on page 1–17](#)
Describes the architectural model of a Nios II software application and how it fits in with the rest of the Nios II system software components.
- [“Software Design Naming Convention” on page 1–19](#)
Identifies the naming convention used in the example design source code files.
- [“MicroC-OS/II Resources” on page 1–20](#)
Describes the tasks, queue, event flag, and semaphores used to implement the Simple Socket Server software application.

- [“NicheStack TCP/IP Stack Initialization” on page 1-21](#)
Describes the tutorial’s tasks and functions that are required to establish and maintain the Ethernet TCP/IP socket connection.
- [“Simple Socket Server Commands and Structures” on page 1-22](#)
Details the actual commands passed over Ethernet to the socket server task and on to the LED management tasks, as well as the structure used to maintain the socket connection.
- [“Simple Socket Server Implementation Details” on page 1-23](#)
Details each of the functions for each software component, including `main()`, MicroC/OS-II initialization, and the details of each of the SSS, LED, and NETUTIL software modules.

Nios II Software Architecture

The onion model in [Figure 1-12](#) shows the architectural layers of a Nios II software application.

Figure 1-12. Layered Software Model

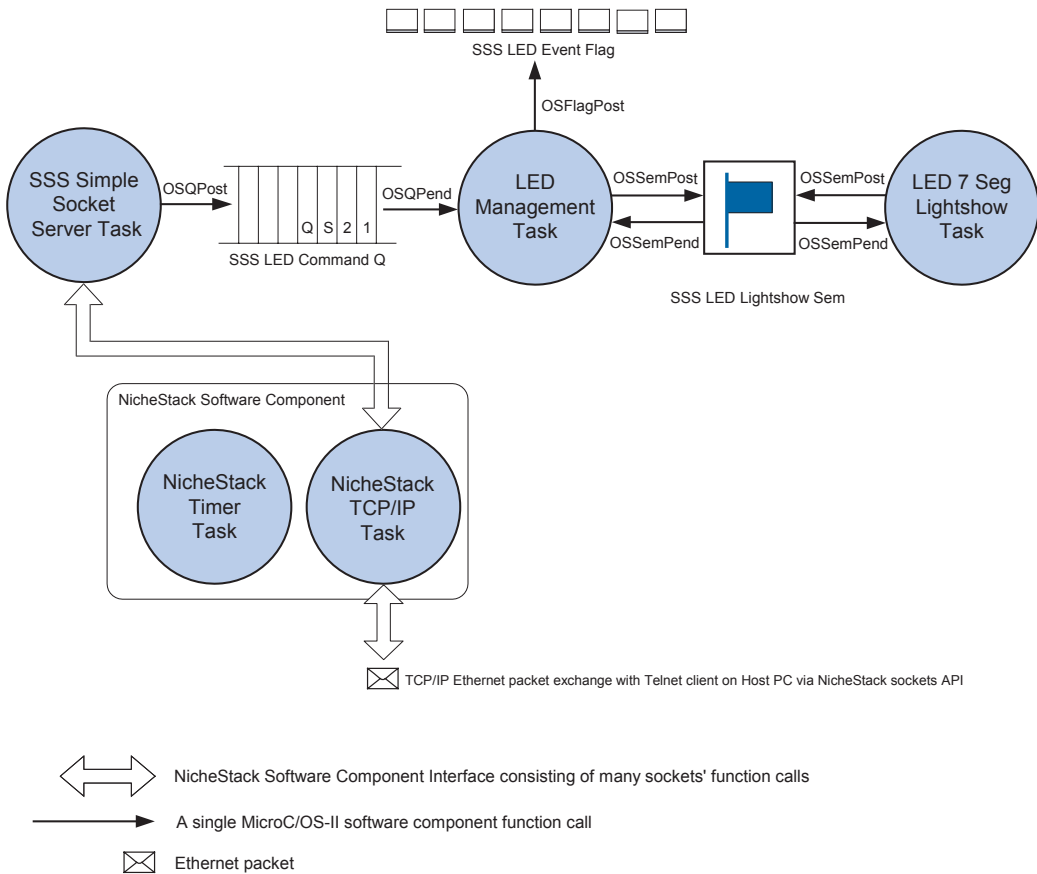


Each layer encapsulates the specific implementation details of that layer, providing a data abstraction for the next outer layer. Following is a description of each layer:

- *Nios II Processor System Hardware*—The core of the onion model contains 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 other hardware peripherals. These drivers contain the physical details of the peripheral devices, abstracting those details from the outer layers.
- *HAL API*—The hardware abstraction layer applications 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 real-time operating system layer provides multi-tasking and inter-task communication services to the NicheStack TCP/IP Networking Stack and the 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 via 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 creation of all application tasks, and all of the semaphores, queue, and event flag real-time operating system inter-task communication resources.
- *Application*—The outermost application layer contains the Simple Socket Server task and LED management tasks.

Figure 1–13 illustrates the structure of the example design. The diagram shows the state of the system after everything has been initialized. The `Iniche_net_ready` global variable is set when the NicheStack TCP/IP Stack is ready. The Ethernet packet containing an LED command sent from a telnet client program is received by the NicheStack TCP/IP Stack software component. The NicheStack TCP/IP Stack processes the incoming Ethernet packets via the TCP/IP protocol, and presents the data packet to the socket server task via the sockets API. The LED command contained within the data packet is then extracted and posted to the LED command queue for processing by the LED management tasks.

Figure 1–13. Simple Socket Server Data Flow Diagram



The following sections describe in detail the function of each element in the diagram.

Software Design Naming Convention

The naming convention used in the Simple Socket Server design employs capitalized acronyms for software module references as prefixes to variables to identify public resources for each software module, while lowercase variables with underscores indicate a private resource or function used internally to a software module. The software modules are named and have capitalized acronym identifiers, as shown in [Table 1–1](#).

Table 1–1. Software Module Acronyms & Names

Acronym	Name
SSS	Simple Socket Server software module
LED	Light Emitting Diode Management software module
NETUTILS	Network Utilities software module
OS	MicroC/OS-II Real-Time Operating System software component

MicroC-OS/II Resources

This section describes the tasks, queue, event flag, and semaphores used to implement the Simple Socket Server application.

Tasks

The MicroC/OS-II tasks shown in [Table 1–2](#) implement the simple socket server application.

Table 1–2. MicroC/OS-II Tasks for the Simple Socket Server

Task	Description
SSSInitialTask()	Initializes the NicheStack TCP/IP Stack, calls functions to create operating system data structures and other tasks.
SSSSimpleSocketServerTask()	Listens for a socket connection and handles the connection. This task only handles one connection at a time.
LEDManagementTask()	Receives and executes commands via SSSLEDCommandQ passed from SSSSimpleSocketServerTask().
LED7SegLightshowTask()	Blinks random patterns on the seven-segment LED display.

The tasks listed in [Table 1–2](#) are all created directly by the application. There are two additional software component layer tasks that are created by the NicheStack TCP/IP Networking Stack: a main task used to operate the networking stack, and a time-keeping task that is used by the main task. The NicheStack TCP/IP Stack main task (`tk_netmain`) is created in the `netmain()` call with a priority of `TK_NETMAIN_TPRIO`. The time-keeping task (`tk_nettick`) is also created in the `netmain()` call, and is assigned 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” on page 1–24](#).

Inter-Task Communication Resources

The following global handles (or pointers) are used to create and manipulate your MicroC/OS-II inter-task communication resources. All of the resources begin with `SSS`, indicating a public resource provided by the Simple Socket Server that is shared between software modules. These resources are declared and created in the `simple_socket_server.c` file by the `SSSCreateOSDataStructs` function, which is invoked from `SSSInitialTask()`.

- **SSSLEDCommandQ**

`SSSLEDCommandQ` is a MicroC/OS-II message queue used to send commands from the simple socket server task to the Nios development board LED control task, `LEDManagementTask()`.

- **SSSLEDEventFlag**

`SSSLEDEventFlag` is the handle to the MicroC/OS-II LED Event Flag Group. Each flag corresponds to one of the LEDs (D0 – D7) on the Nios development board.

- **SSSLEDLightshowSem**

`SSSLEDLightshowSem` is the handle to the MicroC/OS-II LED Lightshow Semaphore. The semaphore is checked by the `LED7SegLightshowTask` each time it updates the seven-segment LED displays U8 and U9. The `LEDManagementTask()` takes the semaphore, via `pend`, away from the `LED7SegLightshowTask()` to toggle the lightshow off, and gives up the semaphore, via `post`, to toggle the lightshow back on. The `LEDManagementTask()` does this in response to the `CMD_LEDS_LIGHTSHOW` command sent from the `SSSSimpleSocketServerTask()` when you send the toggle lightshow command over the TCP/IP socket.

NicheStack TCP/IP Stack Initialization

As described in the “NicheStack TCP/IP Stack Tasks” and “Initializing the Stack” sections of the *Ethernet & the NicheStack TCP/IP Stack – Nios II Edition* chapter in the *Nios II Software Developer’s Handbook*, the NicheStack TCP/IP Stack must be initialized from the Simple Socket Server application code as follows.

Two NicheStack functions must be called:

- `alt_niche_init()`, called from `SSSInitialTask` in `niche_init.c`
- `netmain()`, called from `SSSInitialTask` in `niche_init.c`

Two NicheStack functions must be provided, `get_mac_addr()` and `get_ip_addr()`, which are defined in `network_utilities.c` for this example.

An initialization task called `SSSInitialTask` has been provided that calls both `alt_iniche_init()` and `netmain()` initialization functions in the proper sequence, and then waits until the NicheStack TCP/IP Stack has become fully operational by waiting for the global variable `iniche_net_ready` to be set to `TRUE` before creating the application level task `SSSSimpleSocketServerTask()`.

`SSSSimpleSocketServerTask()` is defined in `simple_socket_server.c` and created with priority `SSS_SIMPLE_SOCKET_SERVER_TASK_PRIORITY`.



You are encouraged to re-utilize the task `SSSInitialTask()` in your own networking application using MicroC/OS-II and the NicheStack TCP/IP Stack.

Simple Socket Server Commands and Structures

The Simple Socket Server example design uses the following data elements:

LED Command Definitions

These definitions are the actual commands passed from the telnet client to the socket on the Nios development board, and on to the LED management tasks. These commands are the elements that flow through the data flow diagram shown in [Figure 1–13 on page 1–19](#).

```
■ CMD_LEDS_BIT_0_TOGGLE '0'
■ CMD_LEDS_BIT_1_TOGGLE '1'
■ CMD_LEDS_BIT_2_TOGGLE '2'
■ CMD_LEDS_BIT_3_TOGGLE '3'
■ CMD_LEDS_BIT_4_TOGGLE '4'
■ CMD_LEDS_BIT_5_TOGGLE '5'
■ CMD_LEDS_BIT_6_TOGGLE '6'
■ CMD_LEDS_BIT_7_TOGGLE '7'
■ CMD_LEDS_LIGHTSHOW 'S'
■ CMD_QUIT 'Q'
```

SSS_Socket Structure

This structure is used to manage a single socket connection.

```
typedef struct SSS_SOCKET
{
enum { READY, COMPLETE, CLOSE } state;
int fd;
int close;
INT8U rx_buffer[SSS_RX_BUF_SIZE]; /* circular buffer */
INT8U *rx_rd_pos; /* position we've read up to */
INT8U *rx_wr_pos; /* position we've written up to */
} SSSConn;
```

Simple Socket Server Implementation Details

This section provides details about the simple socket server tasks and functions.

```
main() (iniche_init.c)
    Calls OSTimeSet()
    Calls SSSInitialTask() (via OSTaskCreateExt)
    Calls alt_uCOSIIErrorHandler()
    Calls OSStart() to begin multithreading
```

SSSInitialTask() (**iniche_init.c**) is used to initialize the NicheStack TCP/IP Stack software, initialize the operating system data structures, and launch any user-defined networking tasks and regular tasks. The convention of creating a task that is used to initialize the rest of the application is advocated by Micrium's MicroC/OS-II examples. This ensures that stack checking initializes correctly if that feature is enabled. This task does the following:

- Calls alt_iniche_init() to perform pre-initialization of the NicheStack Networking Stack
- Calls netmain() to initialize and start the NicheStack Networking Stack
- Instantiates ssstask (via TK_NEWTASK) to start the Simple Socket Server networking task
- Calls SSSCreateOSDataStructs() to create data structures (SSSLEDCommandQ, SSSLEDLightshowSemaphore, and SSSLEDEventFlag real-time operating system resources) for the Simple Socket Server application
- Calls SSSCreateTasks() to create non-NicheStack TCP/IP Stack dependent tasks, including the LED tasks
- Calls OSTaskDel() to delete itself as a task

`SSSSimpleSocketServerTask()` (`simple_socket_server.c`) does the following:

- Creates a socket to serve a TCP/IP connection, binds to the socket, and listens for TCP/IP connection requests from a client.
- Calls `sss_handle_accept()` for an incoming TCP/IP connection.
- Calls `sss_handle_receive()` to serve the TCP/IP connection. If you require multiple TCP/IP connections, you can modify this task to create other tasks that handle each individual TCP/IP connection.
- Calls `sss_reset_connection()`, `sss_send_menu()`, and `sss_exec_command()`.
- When data packets are received, the LED commands are extracted and passed to `LEDManagementTask()` via the `SSSLEDCommandQ`.

LED Tasks (`leds.c`) include the following:

- `LEDManagementTask()` consumes LED commands received on the `SSSLEDCommandQ`. The commands received are executed by toggling the `SSSLEDLightshowSem` semaphore in response to the command `CMD_LEDS_LIGHTSHOW`, or posting to the `SSSLEDEventFlag` to manipulate LEDs D0 – D7 in response to `CMD_LEDS_BIT_TOGGLE` commands. The application is terminated in response to the `CMD_QUIT` command.
- `LED7SegLightshowTask()` blinks random patterns on the seven-segment LED display. This task suspends and resumes its LED update based on the `SSSLEDLightshowSem` semaphore, which is controlled by a single command sent to the `LEDManagementTask()`, `CMD_LEDS_LIGHTSHOW`.

Important NicheStack TCP/IP Stack Concepts

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

Error Handling

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

NicheStack TCP/IP Stack Default Task Creation

The NicheStack TCP/IP Stack creates one or more system level tasks during system initialization, when the `netmain()` function is called. Users have complete control over these system level tasks through a global configuration file called `ipport.h`, located in the directory structure for the system library project, in the `Debug/system_description` path.

You can edit the `#define` statements in `ippport.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 details on 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/src/downloads/packages` directory.

In the “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 used by the networking stack

For more information about these NicheStack TCP/IP Stack tasks, refer to [“Task Priorities in the Simple Socket Server Design”](#) on page 1–28.

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

The function call `TK_NEWTASK` must be used to create any tasks that will use the NicheStack networking services. Tasks that do not use networking services should be created with the MicroC/OS-II function `OSTaskCreate()`.

`TK_NEWTASK` (defined in the file `osportco.c`) is a function used by the NicheStack Networking Stack 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 is used to specify the task name, the MicroC/OS-II thread priority, and the stack size. Both files are located in the `<Nios II EDS install path>/components/altera_iniche/UCOSII/src/downloads/30src/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 */
};
```

A local `struct inet_taskinfo` structure with the elements defined must be declared for every networking task you create in your application. These elements are listed below, 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 via the `TK_OBJECT` macro. The NicheStack Networking Stack makes use of the `tk_ptr` element during the operation. After declaring the variable name via 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 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 and should not be changed by you.

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 called `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 you want to create, which must have the form `void name (void)`. The `name` parameter is also used to set `TK_ENTRY_PTR(entry)`.

To create your own application tasks that use the services offered by the NicheStack TCP/IP Stack, perform the following 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 is set to `FALSE` at run-time and is changed 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 SSSSimpleSocketServerTask
void SSSSimpleSocketServerTask(void) {
    // task specific code
}

// Creation of NicheStack networking task
TK_OBJECT(to_ssstask);
TK_ENTRY(SSSSimpleSocketServerTask);

struct inet_taskinfo ssstask = {
    &to_ssstask,
    "simple socket server",
    SSSSimpleSocketServerTask,
    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 overutilize 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.
- There are a 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.

Task Priorities in the 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, where tasks assigned a lower priority number are treated as higher priority tasks. Because the Altera version of 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 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.

Table 1–3 lists the tasks that might be running in your system, as well as the mechanism for configuring the priority of these tasks.

Table 1–3. MicroC/OS-II Tasks for the Simple Socket Server	
Task Type	Configuration Mechanism
MicroC/OS-II Internal Tasks	Nios II IDE “RTOS Options” menu (found by selecting <Nios II system_library> and clicking Properties on the File menu in the Nios II IDE)
NicheStack TCP/IP Stack Internal Tasks	ippport.h source file (found in <Nios II system_library>/ Debug/system_description directory using Nios II IDE)
Networking Initialization Task	iniche_init.c source file
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 priorities of the tasks in the simple socket server design are discussed in the following sections:

MicroC/OS-II Internal Tasks

The Simple Socket Server application has been configured not to use any MicroC/OS-II Internal Tasks.

NicheStack TCP/IP Stack Internal Tasks

`TK_NETMAIN_TPRIO`, defined in `ippport.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 `ippport.h`, sets the priority to a value of 3 for the NicheStack TCP/IP Stack time-keeping task, launched by `netmain()`. This task is used by the NicheStack TCP/IP Stack to keep track of time-based events in the networking stack. Altera recommends that the priority of this task should be set to one priority level lower than `TK_NETMAIN_TPRIO`.

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 of the other tasks before deleting itself. It 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.

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. In general, 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.

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`.

`LED_7SEG_LIGHTSHOW_TASK_PRIORITY` is set to a value of 18. The priority of this application task is set lower than the rest of the tasks in the system because it requires very little of the Nios II CPU's cycles to operate. Additionally, it only needs to operate once every 50 milliseconds to update the LED patterns. This task should be set to the lowest priority task in your system. `LED7SegLightshowTask()` then acts as a task starvation monitor, because the LEDs will blink only if all other higher priority tasks have had a chance to be scheduled.

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, via the Nios II IDE 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 details about the `_reent` structure, refer to the “The Newlib ANSI C Standard Library” and the “Implementing MicroC/OS-II Projects in the Nios II IDE” sections of the *MicroC/OS-II Real-Time Operating System* chapter in the *Nios II Software Developer's Handbook*.

Where to Go Next

This example is easily expandable to handle multiple TCP connections on a single port. The `SSSimpleSocketServerTask()` 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 via a web browser, or remote software upgrade for corrections or feature enhancements to a product already in the field.

Introduction

To complete this tutorial, you must have the Nios® II IDE installed, and your Nios development board must be connected to a host PC on both the Ethernet and USB/JTAG ports. For details about installing the software and connecting the Nios development board to the USB-Blaster™ cable, refer to the *Nios II Development Kit Getting Started User Guide*.

The full-featured reference hardware design for the Nios development boards includes the Ethernet device required by this NicheStack tutorial. The Ethernet device included in these reference designs, along with the physical MAC/PHY on each of the Stratix® II, Stratix, Stratix Professional, Cyclone® II, and Cyclone Edition Nios development boards, is the LAN91C111 Ethernet peripheral. The full 14-bit address width of the chip is used, with the 8 peripheral registers accessible at locations `base+0x300` through `base+0x30f`. The Ethernet peripheral base address settings for the full-featured hardware reference designs, along with IRQ setting, can be examined in `system.h`.

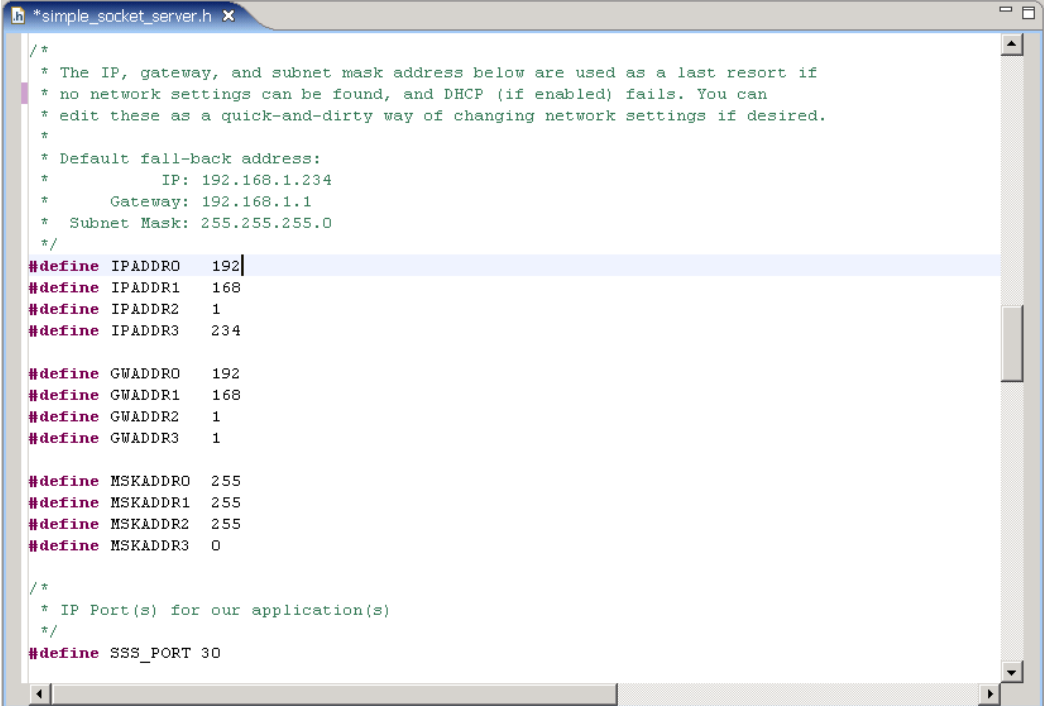
Network Connection

If a DHCP server is used to assign an IP address, connect your Nios development board to your Ethernet network.

If the Nios development board is connected directly to your PC with a crossover Ethernet cable, or a DHCP server is not available, the IP addresses can be specified manually by entering the IP address values into `simple_socket_server.h`. Be sure to turn off the **Use DHCP to automatically assign IP address** option on the **NicheStack Software Components** dialog box (shown turned on in [Figure 1–6 on page 1–10](#)).

[Figure A–1](#) shows the default IP address definitions in `simple_socket_server.h`. The default values shown represent an IP address for the Nios development board of 192.168.1.234, with a gateway of 192.168.1.1, and a subnet mask of 255.255.255.0 (a class C network). In a crossover Ethernet cable configuration, specify the IP address of your PC as the gateway.

Figure A-1. Excerpt from `simple_socket_server.h`



```
*simple_socket_server.h x
/*
 * The IP, gateway, and subnet mask address below are used as a last resort if
 * no network settings can be found, and DHCP (if enabled) fails. You can
 * edit these as a quick-and-dirty way of changing network settings if desired.
 *
 * Default fall-back address:
 *      IP: 192.168.1.234
 *      Gateway: 192.168.1.1
 *      Subnet Mask: 255.255.255.0
 */
#define IPADDR0 192
#define IPADDR1 168
#define IPADDR2 1
#define IPADDR3 234

#define GWADDR0 192
#define GWADDR1 168
#define GWADDR2 1
#define GWADDR3 1

#define MSKADDR0 255
#define MSKADDR1 255
#define MSKADDR2 255
#define MSKADDR3 0

/*
 * IP Port(s) for our application(s)
 */
#define SSS_PORT 30
```


Introduction

The process for upgrading to NicheStack TCP/IP Stack from lightweight IP (lwIP) involves changing your lwIP-based source code to accommodate the following issues:

- “New Method for TCP/IP Stack Initialization”
- “New Method for Notification that the TCP/IP Stack Is Ready”
- “New Method for Creation of Tasks that Will Use TCP/IP Stack”
- “Different Customization Process and Include Files”
- “New Function Prototype and Parameter Type Definitions for `Network_utilities.c`”
- “New BOOLEAN Type Definition”



Refer to the Simple Socket Server software example for a source code example that uses the NicheStack TCP/IP Stack software component.

Issues in Upgrading

New Method for TCP/IP Stack Initialization

lwIP uses a callback function, `init_done_func()`, which is invoked by the TCP/IP stack when initialization is complete. NicheStack TCP/IP Stack initialization does not provide a callback function. Instead, the `inich_net_ready` global variable should be checked by a task to determine when the stack is ready.

NicheStack TCP/IP Stack requires two initialization calls, `alt_iniche_init()` and `netmain()`. These two calls should be made from a task that executes with a higher priority than any task that uses the sockets interface. Refer to `iniche_init.c` in the Simple Socket Server software example. As in lwIP, the NicheStack TCP/IP Stack initialization creates two tasks, but they have different names and capabilities.

New Method for Notification that the TCP/IP Stack Is Ready

The NicheStack TCP/IP Stack sets the global variable `iniche_net_ready` to TRUE when the TCP/IP stack has obtained an IP address for the configured Ethernet device and is ready to accept socket

calls. The lwIP software example used a task called `NETUTILSDHCPTIMEOUTTASK` to determine when a DHCP IP address was provided, and posted to a semaphore called `SSSATTAINEDIPADDRESSSEM` when the lwIP stack was ready. The default timeout for waiting on a response from the DHCP server was 120 seconds, but is now 30 seconds for NicheStack.

New Method for Creation of Tasks that Will Use TCP/IP Stack

lwIP used `sys_thread_new()` to create tasks with a fixed stack size of 2048 bytes. The NicheStack TCP/IP Stack allows for the creation of tasks with variable stack sizes. Refer to [“Creating Tasks that Use the NicheStack TCP/IP Stack Sockets Interface”](#) on page 1–25.

Different Customization Process and Include Files

lwIP had a graphical configuration page accessible from the System Properties page in the Nios II IDE that enabled customizations for protocols and memory values. For additional NicheStack customizations, modify the `ippport.h` C header file, found in the `system_description` folder under the system library project build directory (for example, `simple_socket_server_0_syslib\Debug\system_description`).

lwIP tasks utilizing sockets needed to include the following files:

```
#include "alt_lwip_dev.h"
#include "lwip/sys.h"
#include "lwip/netif.h"
#include "lwip/sockets.h"
```

NicheStack tasks utilizing sockets should instead include the following two C header files:

```
#include "ippport.h"
#include "tcpport.h"
```

New Function Prototype and Parameter Type Definitions for `Network_utilities.c`

The `get_mac_addr()` prototype has changed from `get_mac_addr(alt_lwip_dev *lwip_dev)` to `get_mac_addr(NET net, unsigned char mac_addr[6])`.

`get_ip_addr()` uses a different structure definition for `struct ip_addr`.

New BOOLEAN Type Definition

The BOOLEAN type values are provided by MicroC/OS-II and defined in **ucosii.h**. The valid enumerated type values for a NicheStack TCP/IP Stack BOOLEAN structure are OS_TRUE and OS_FALSE. These values have replaced the older enumeration values of TRUE and FALSE. This change is not unique to the NicheStack TCP/IP Stack, just to MicroC/OS-II. The change is described here because the Simple Socket Server software example uses BOOLEAN variable types.

