

Nios[®] II

Using Lightweight IP with the --- Nios II Processor Tutorial



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

Copyright © 2004 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.



Printed on recycled paper

TU-N28304-1.1





About this Tutorial	v
How to Find Information	v
How to Contact Altera	vi
Typographic Conventions.....	vii
Lightweight IP Tutorial	1-1
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	1-8
Examine the Simple Socket Server Project Files	1-15
Build & Run the Simple Socket Server Project	1-15
Interacting with the Simple Socket Server	1-17
Simple Socket Server Design Overview	1-21
Nios II Software Architecture	1-21
Software Design Naming Convention	1-23
MicroC-OS/II Resources	1-24
Tasks	1-24
Inter-Task Communication Resources	1-25
lwIP Initialization	1-26
Simple Socket Server Commands and Structures	1-27
LED Command Definitions	1-27
SSS_Socket Structure	1-27
Simple Socket Server Implementation Details	1-27
Important lwIP Concepts	1-29
Error Handling	1-29
Creating Tasks that use the lwIP Sockets Interface	1-29
Task Priorities in the Simple Socket Server Design	1-30
TCP/IP Throughput Performance	1-32
Task Stack Size	1-32
Where to Go Next	1-33
Appendix A. Hardware Setup Details	A-1
Appendix B. Optimizing lwIP Throughput	B-1
Introduction	B-1
lwIP Configuration Values	B-1
Configuring Optimization & Debug Levels	B-3

This tutorial introduces and familiarizes you with the lightweight IP (lwIP) TCP/IP software component included in your Nios® II development kit.

Table 1-1 shows the tutorial revision history.

<i>Table 1-1. Tutorial Revision History</i>	
Date	Description
December 2004	Minor updates for clarity.
September 2004	First publication.

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 world-wide 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/	altera.com/mysupport/
	(800) 800-EPLD (3753) (7:00 a.m. to 5:00 p.m. Pacific Time)	(408) 544-7000 (1) (7:00 a.m. to 5:00 p.m. Pacific Time)
Product literature	www.altera.com	www.altera.com
Altera literature services	lit_req@altera.com (1)	lit_req@altera.com (1)
Non-technical customer service	(800) 767-3753	(408) 544-7000 (7:30 a.m. to 5:30 p.m. 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, check box 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: <file name>, <project name>.pdf 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 on-line help topics are shown in quotation marks. Example: "Typographic Conventions."
Courier type	Signal and port names are shown in lowercase Courier type. Examples: data1, tdi, input. Active-low signals are denoted by suffix n, e.g., resetn. Anything that must be typed exactly as it appears is shown in Courier type. For example: c:\qdesigns\tutorial\chiptrip.gdf. Also, sections of an actual file, such as a Report File, references to parts of files (e.g., the AHDL keyword SUBDESIGN), as well as logic function names (e.g., TRI) 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.
	The caution indicates required information that needs special consideration and understanding and should be read prior to starting or continuing with the procedure or process.
	The warning indicates information that should be read prior to starting or continuing the procedure or processes
	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 lightweight IP (lwIP) TCP/IP software component included in your Nios II development kit. Topics covered include:

- Configuring and initializing the lwIP 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 lwIP 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 lightweight Internet Protocol (lwIP) 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 lwIP TCP/IP networking stack.

This tutorial provides 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 lwIP 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 lightweight Internet Protocol (lwIP) 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 lwIP initialization and configuration for the Nios II processor, refer to the *Ethernet and Lightweight IP* chapter in the *Nios II Software Developer's Handbook*

Hardware & Software Requirements

This tutorial requires the following hardware and software:

- Quartus® II version 4.2 or later
- Nios II development kit version 1.1 or later (Full kit required including MicroC/OS-II and lwIP integration)
- Nios development board, Stratix™ II Edition, Cyclone™ Edition, Stratix Edition, or Stratix Professional Edition
- Altera USB-Blaster™ Rev. B cable
- RJ-45 connected Ethernet-cable on the same network as the PC development host

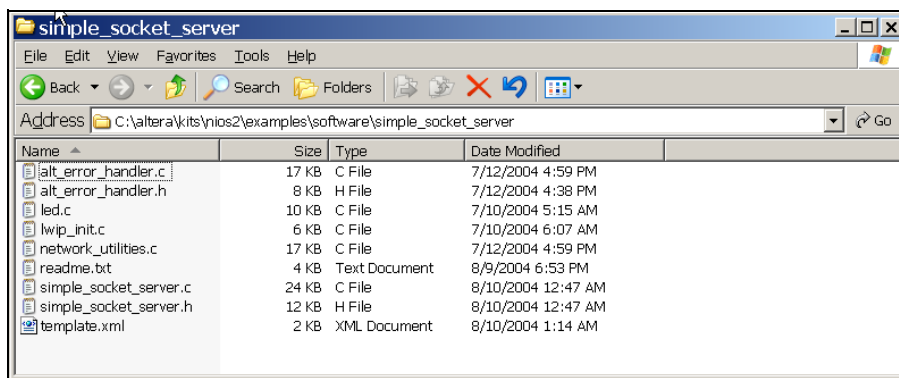


To complete this tutorial, you must have the Nios II IDE installed, and your Nios development board must be connected to a host PC. See [Appendix A, Hardware Setup Details](#) for detailed hardware-setup instructions.

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 lwIP tutorial software design files in the `<Nios II kit path>\examples\software\simple_socket_server` directory.

Figure 1–1. Simple Socket Server lwIP 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 seven source-code files make up the Simple Socket Server application for this lwIP tutorial.

- **alt_error_handler.c** - Contains the implementation of three error handlers, one each for the Simple Socket Server (SSS), lwIP, 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.
- **lwip_init.c** - Defines `main()`, which initializes MicroC/OS-II and lwIP, and `init_done_func()`; the lwIP callback function.
- **network_utilities.c** - Defines functions to manipulate the MAC and IP addresses.
- **simple_socket_server.c** - Defines all of the tasks and functions which utilize the lwIP 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 an lwIP and MicroC-OS/II software image for the Nios II processor involves the following:

- Creating a new Nios II IDE C/C++ application project with the **Simple Socket Server** project template.

- Configuring the system library project, including MicroC/OS-II and the lwIP software component.
- Building the application project.
- Running (and debugging where necessary) the application project.

Create a New Nios II IDE Project

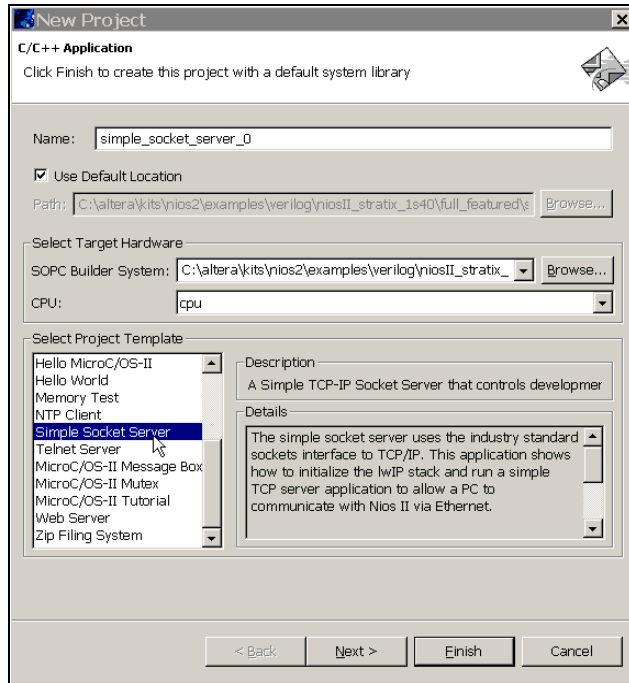
In this section, you will create a new Nios II IDE project using a project template. Perform the following steps:

1. Choose **Programs> Altera> Nios II Development Kit > Nios II IDE** (Windows Start menu) to start the Nios II IDE.
2. Choose **New> C/C++ Application** (File menu). The first page of **New Project** wizard opens.
3. Under **Select Project Template**, select **Simple Socket Server**. The project name and project path are filled in for you automatically.
4. Click **Browse** under **Select Target Hardware**. Browse to the **full_featured** hardware example directory for the Nios development board that you are targeting, e.g., *<Nios II kit path>\examples\verilog\niosII_stratix_1s40\full_featured* directory.
5. Select SOPC Builder system file (.ptf) for the full_featured design, e.g., **full_1s40.ptf**.

6. Click **Open**.

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 the Nios II CPU as defined in SOPC Builder.

Figure 1–2. New Project Wizard - Page 1

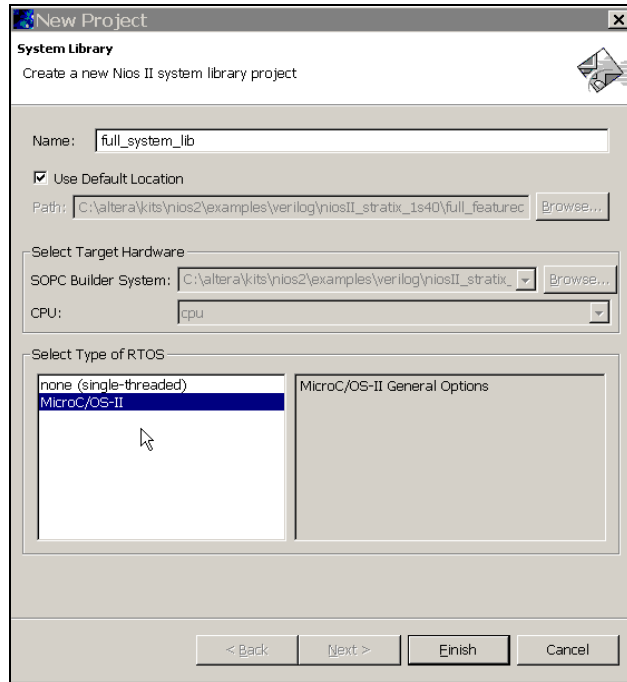


7. Click **Next** to go to the second page of the **New Project** wizard.

8. Turn on **Select or create a system library**.

9. Click **New System Library Project** to open the system library page as shown in **Figure 1-3**.

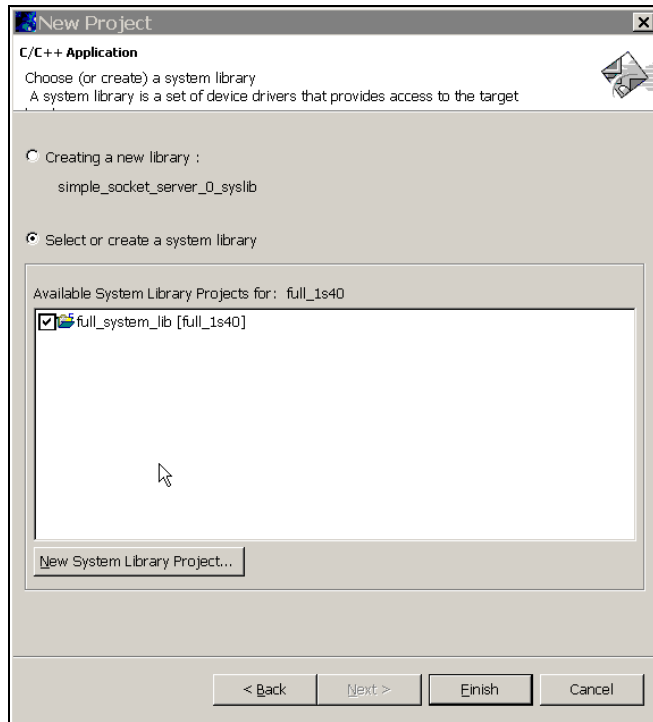
Figure 1-3. New System Library Dialog Box



10. Type full_system_lib in the **Name** box.
11. Select **MicroC/OS-II** from the **Select Type of RTOS** box.

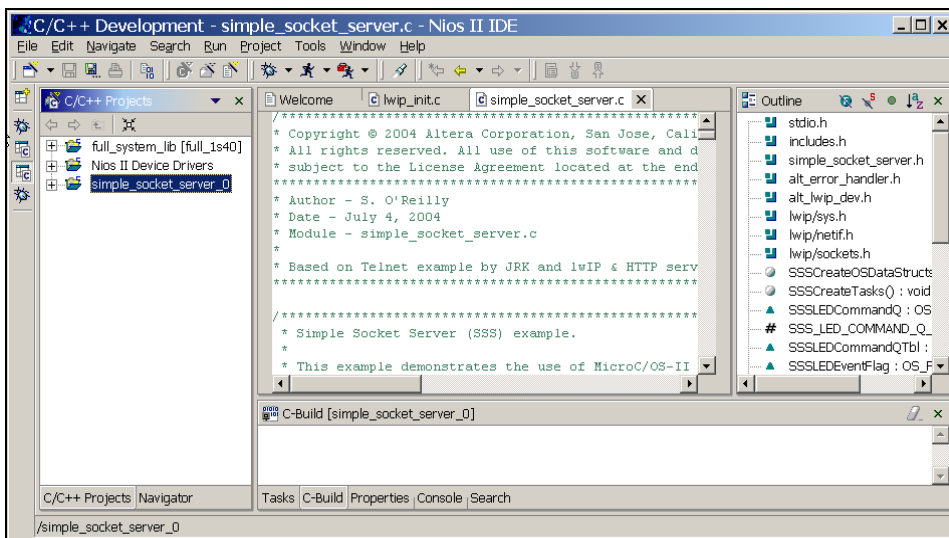
12. Click **Finish** to return to the **New Project** wizard. See [Figure 1-4](#).

Figure 1-4. New Project Wizard - Page 2



- Click **Finish** to complete creating your new projects. The wizard creates two projects in the Nios II IDE C/C++ **Projects** tab of the C/C++ **Development** perspective as shown in Figure 1-5.

Figure 1-5. New Project in the C/C++ Development Perspective



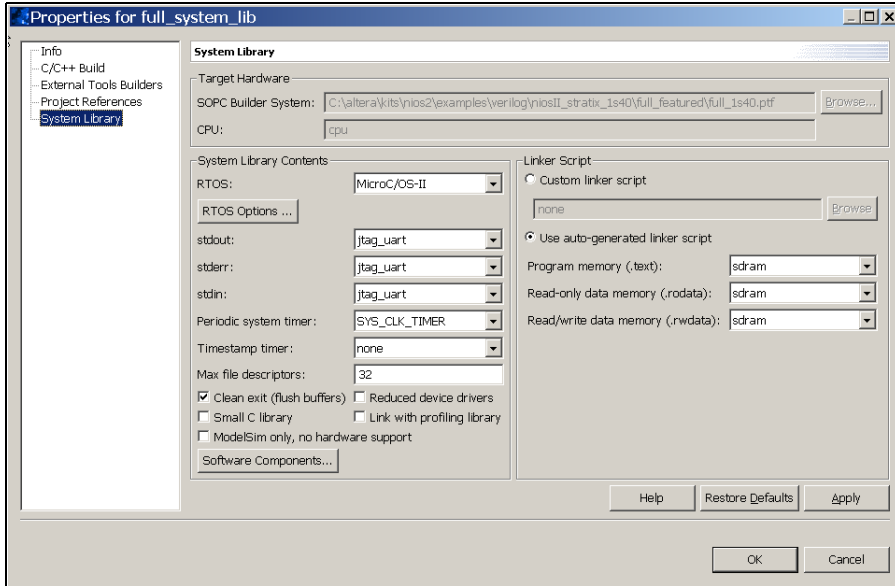
Configure the System Library

After you create a new system library, you must configure it (e.g., defining `stdin`, `stdout`, `stderr`, and other parameters). See the Nios II IDE online *Nios II Software Development Tutorial* for more details. For this tutorial, you must configure MicroC/OS-II and lwIP software components. Perform the following steps to configure the MicroC/OS-II kernel.

- Right-click on the system library, `full_system_lib`, in the Nios II IDE C/C++ **Projects** view.
- Choose **Properties** in the pop-up menu to open the properties dialog box for the system library.

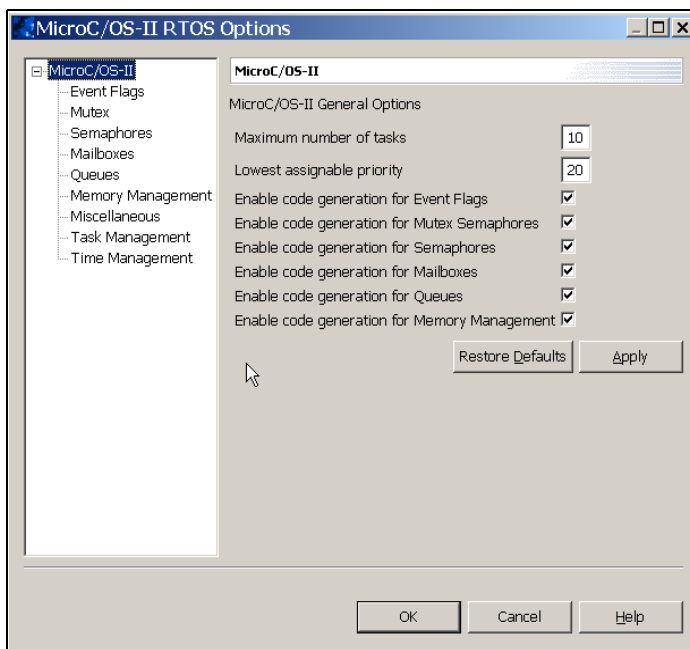
3. Click **System Library** to display the system library options as shown in **Figure 1–6**.

Figure 1–6. System Library Options



- Click **RTOS Options** under **RTOS**. The **MicroC/OS-II RTOS Options** dialog box opens, as shown in **Figure 1-7**.

Figure 1-7. MicroC/OS-II RTOS Options



- Click the “+” in the left hand panel to expand the contents under **MicroC/OS-II** as shown in **Figure 1-7**.

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.



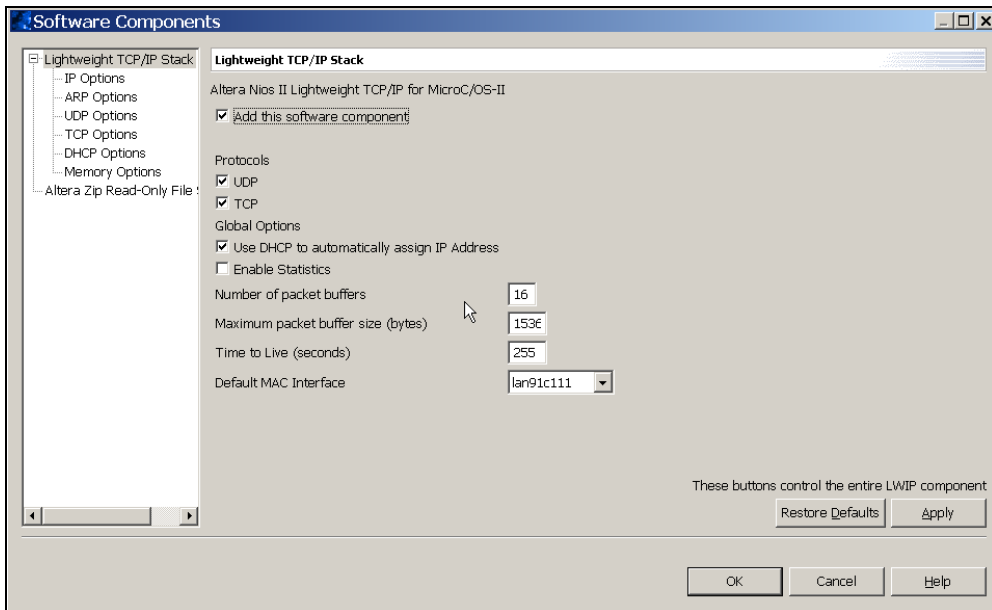
Although this example software design does not use all of the MicroC/OS-II system calls, lwIP internally uses many more MicroC/OS-II system calls than are used by the Simple Socket Server application itself. Do not disable any system calls unless you need to be very conservative with your code size requirements. 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 on 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*.

6. For this tutorial, choose the default settings and click **OK**. You are returned to the **System Library** options properties page.
7. Click **Software Components**.
8. Click the "+" in the left hand panel to expand the contents under Lightweight TCP/IP Stack as shown in [Figure 1-8](#).

Figure 1-8. Lightweight TCP/IP Stack Options



9. On the left panel, highlight **Lightweight TCP/IP Stack**.

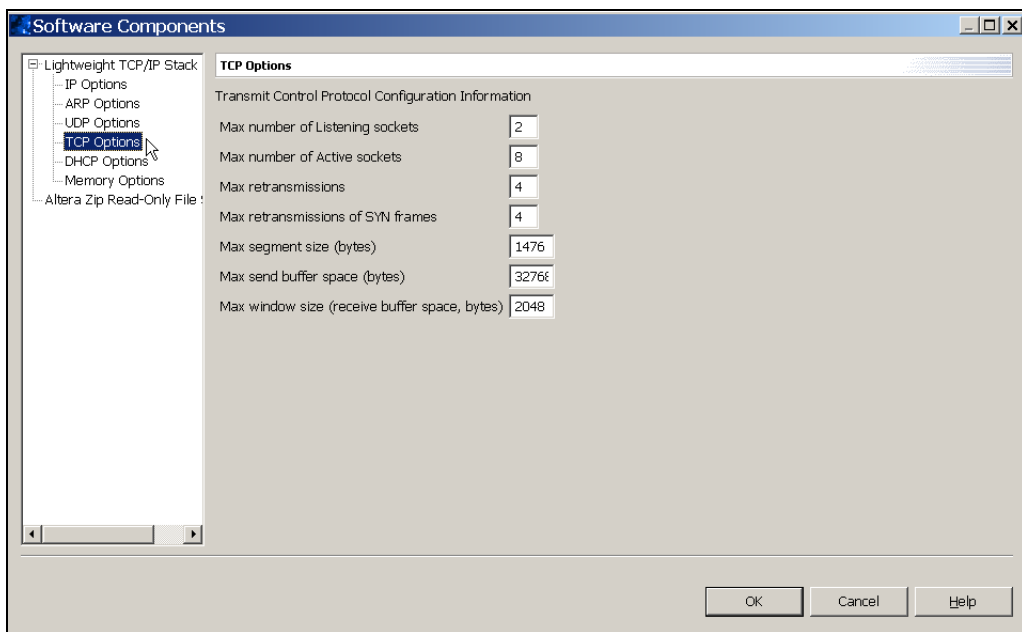
The **Software Components** window displays options available for the Lightweight TCP/IP Stack.

10. Under **Altera Nios II Lightweight TCP/IP for MicroC/OS-II**, turn on **Add this software component**.

There are 6 sub-pages of options (see [Figure 1-8](#)) in addition to the main **Lightweight TCP/IP Stack** options page for further customization of lwIP. Click on any of the 6 sub-headings in the left panel to view the various options for each sub-page shown in the right panel. For this tutorial we will use all of the defaults:

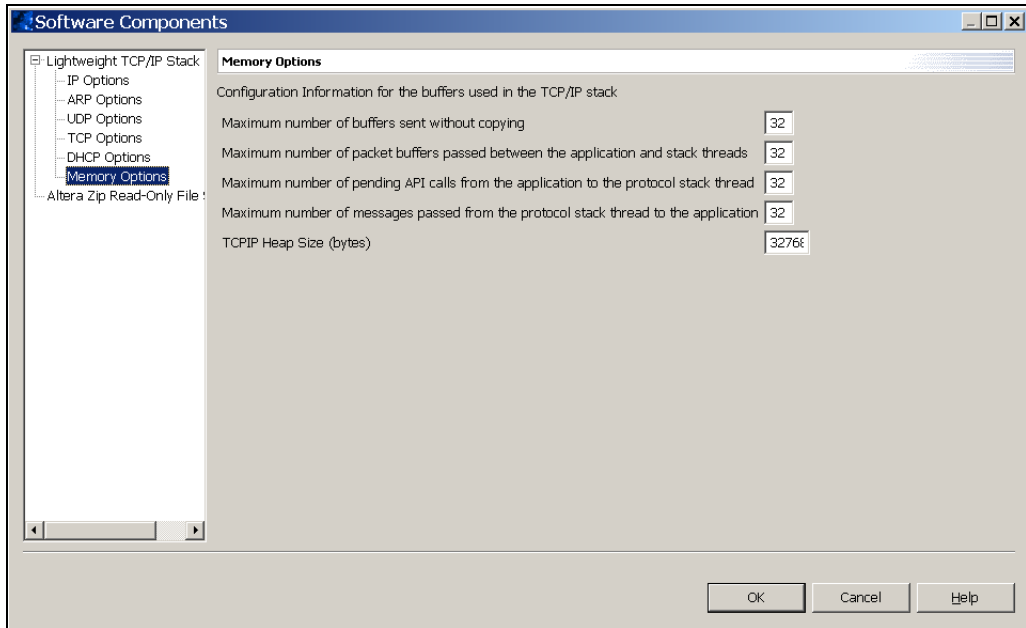
- IP Options – Forward IP Packets
- ARP Options – Size of the ARP Table
- UDP Options – Maximum number of UDP sockets
- TCP Options – Various resources maximums, as shown in [Figure 1-9](#)
- DHCP Options – ARP to check the assigned address is not in use.

Figure 1-9. lwIP TCP Options



- Memory Options – Configuration information for buffers used in the TCP/IP stack as shown in [Figure 1–10](#).

Figure 1–10. lwIP Memory Options

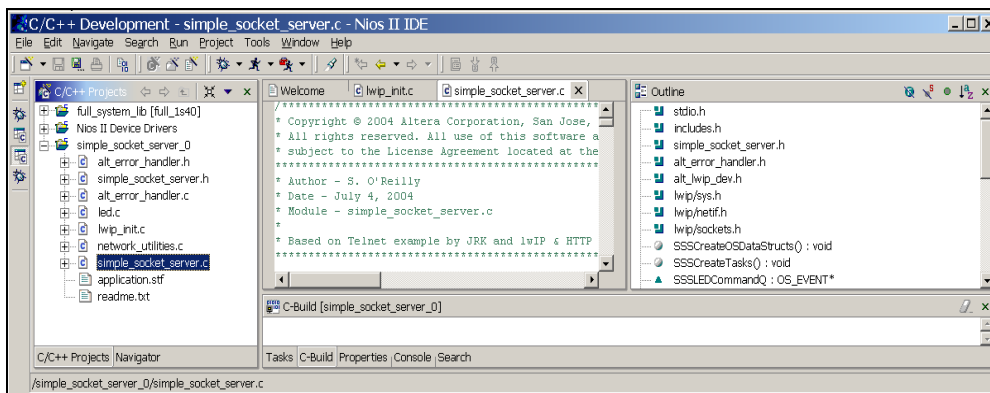


11. Click **OK** to complete configuration of lwIP.
12. Click **OK** in the system library properties page to complete configuration of the system library.

Examine the Simple Socket Server Project Files

You can click the “+” to the left of the **simple_socket_server_0** folder icon to view the source files as shown in [Figure 1-11](#).

Figure 1-11. Simple Socket Server Project Files



You have finished creating and configuring both the **simple_socket_server_0** and the **full_system_lib** projects, and are ready to build and run the example as described in the following section.



For additional details on how to build and run programs with the Nios II IDE, see the *Nios II Software Development Tutorial* in the Nios II IDE online help.

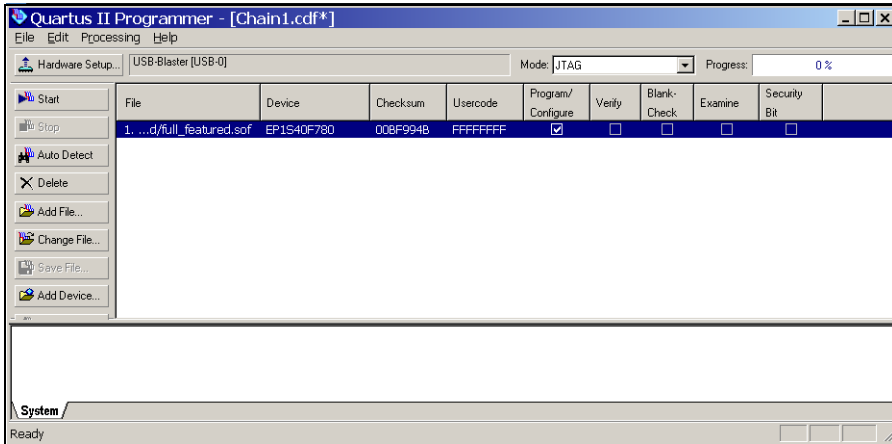
Build & Run the Simple Socket Server Project

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.


1. Choose **Quartus II Programmer** (Tools menu).
2. Choose **Open** (File menu) in the Quartus II Programmer.

3. Select the FPGA configuration file (.sof), for example, **full_featured.sof**.
4. Click **Open**. You return to the Quartus II Programmer.
5. Turn on the **Program/Configure** option as shown in **Figure 1-12**.

Figure 1-12. Quartus II Programmer



6. Click **Start** to configure the FPGA on the development board.
7. Choose **Exit** (File menu) to close the Quartus II Programmer, or minimize the Quartus II Programmer, and return to the Nios II IDE. If the Quartus II Programmer asks if you want to save changes to the **chain1.cdf** file, click **No**.
8. In the Nios II IDE, select the **simple_socket_server_0** project in the **C/C++ Projects** view of the **C/C++ Development** perspective.

 You must have the **simple_socket_server_0** project selected in the **C/C++ Projects** view. You cannot build a project from the **Navigator** view of the **C/C++ Development** perspective.
9. Choose **Run As > Nios II Hardware** (Run menu) to build the program, download it to the board, and then run it.

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



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

Interacting with the Simple Socket Server

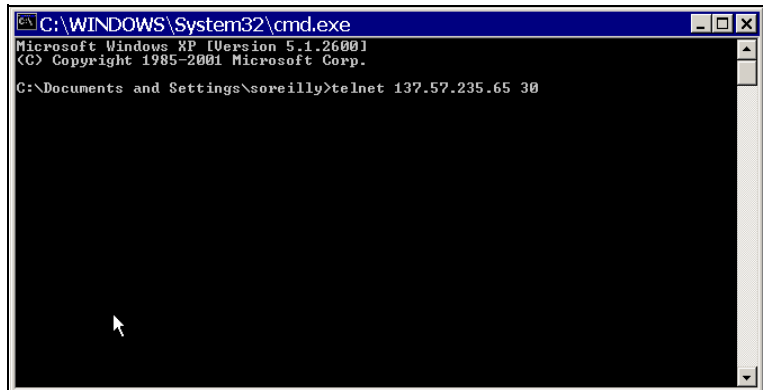
After download, the seven-segment LED banks will begin to flash with random patterns. The JTAG console and the LCD panel will display a message stating that the program is waiting for the IP address to be assigned by the DHCP server.

After two minutes, both the JTAG console and the LCD panel will display either the obtained IP address from the DHCP server, or the static IP address as defined in `simple_socket_server.h` due to the DHCP two-minute time out.

- ✓ After the IP address is assigned, execute the following command from a DOS command prompt, as shown in [Figure 1-13](#).

```
telnet <IP_address> 30 ↵
```

Figure 1-13. Connecting to the Simple Socket Server



If the connection to port 30 on the Nios development board is successful, the menu will be displayed in the DOS command window, as shown in [Figure 1-14](#).

Figure 1-14. 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.
3
--> Simple Socket Server Command S.
q
--> Simple Socket Server Command Q.
Terminating connection.

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

```

You will press commands at the keyboard. Keys pressed at the DOS command prompt get sent over the telnet connection via Ethernet to a task listening on a socket for commands. This task responds to those commands by sending instructions to another task to manipulate the LEDs.

To exercise the functionality of the Simple Socket Server, enter commands to the telnet session. Entering a number from zero through seven, followed by a return, will cause the corresponding LEDs D0 – D7, to toggle on–or–off on the Nios development board. Entering the letter `s` will stop the random blinking LED pattern on the seven- segment LED bank. Entering this `S` command again will restart the light show. To reproduce the specific run-time behavior shown in [Figure 1-14](#) and [Figure 1-15](#), do the following at the DOS command prompt:

1. Type the number 1 ↵

LED D1 is toggled. The Nios II IDE Console displays 2 messages: “processing RX data” followed by “Value for LED_PIO_BASE set to 2”.

2. Type the number 2 ↵

LED D2 is toggled. The Nios II IDE Console displays 2 messages: “processing RX data” followed by “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↵

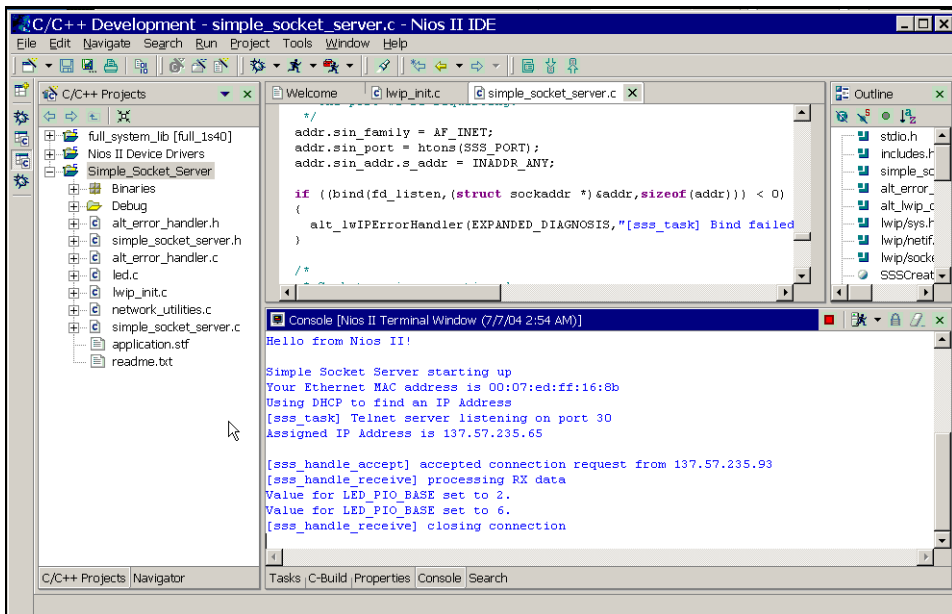
The seven-segment LED display stops flashing.

4. Type the letter Q↵

This terminates the socket connection on the Nios development board, and the telnet command exits.

Figure 1-14 on page 1-17 shows the initial state of the **Simple Socket Server Menu**, along with commands 1, 2, S, and Q. Figure 1-15 shows the corresponding output on the Nios II IDE console during the telnet session.

Figure 1-15. Nios II IDE Console Output During Telnet Session



Simple Socket Server Design Overview

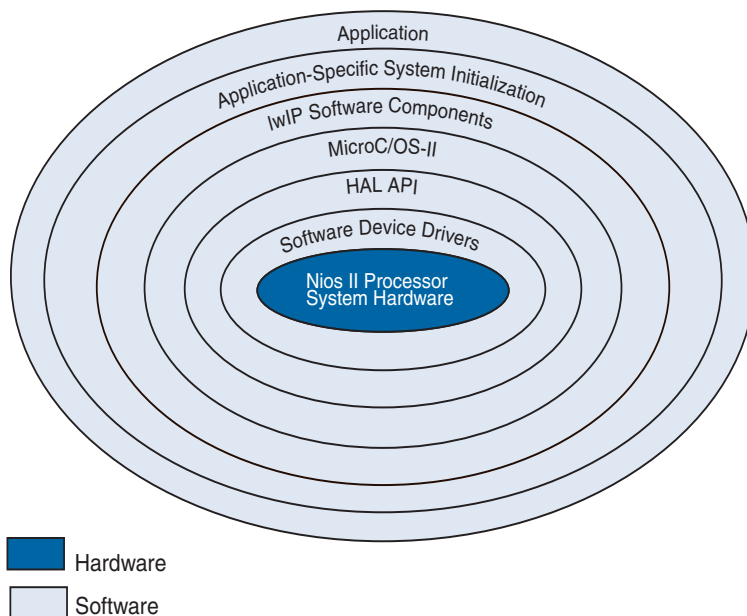
This section describes the simple socket server design. The discussion is divided into the following subsections:

- [“Nios II Software Architecture” on page 1-20](#)
This section 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 1-22](#)
This section identifies the naming convention used in the example design source code files.
- [“MicroC-OS/II Resources” on page 1-23](#)
This section describes the tasks, queue, event flag, and semaphores used to implement the **Simple Socket Server** software application.
- [“Simple Socket Server Commands and Structures” on page 1-26](#)
This section 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.
- [“lwIP Initialization” on page 1-25](#)
This section describes the tutorial’s tasks and functions which are required to establish and maintain the Ethernet TCP/IP socket connection.
- [“Simple Socket Server Implementation Details” on page 1-26](#)
This section 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 1-16](#) shows the architectural layers of a Nios II software application.

Figure 1-16. Layered Software Model



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

- *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 which 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 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 lwIP networking stack and **Simple Socket Server** application.
- *lwIP Software Component* — The lwIP 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 lwIP 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, LED management tasks, and network utility DHCP timeout task.

Figure 1-17. Simple Socket Server Data Flow Diagram

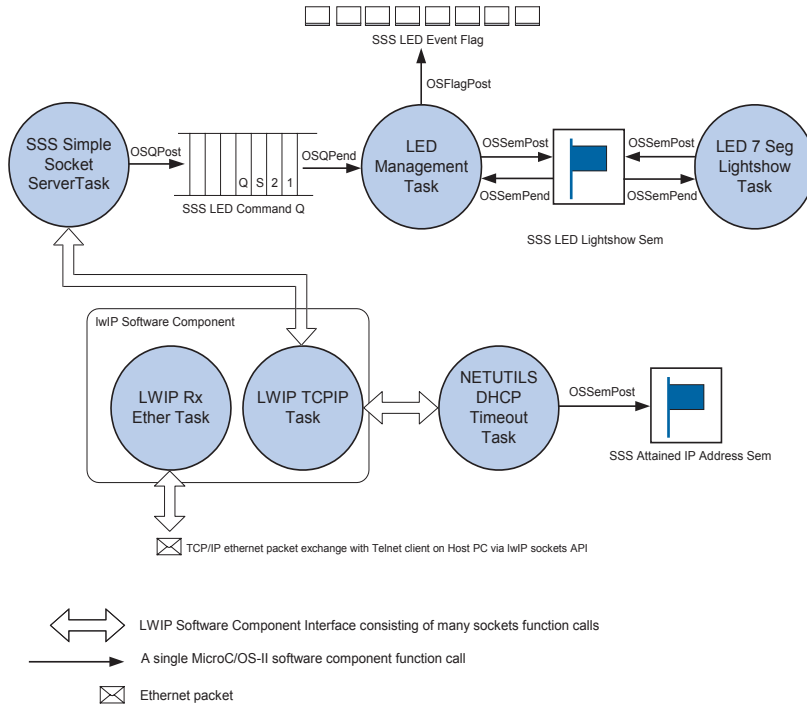


Figure 1-17 illustrates the structure of the example design. The following sections will describe in detail the function of each element in the diagram. The diagram shows the state of the system after everything has been initialized. The Ethernet packet containing a LED command sent from a Telnet client program is received by the lwIP software component. lwIP 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 into the LED command queue for processing by the LED management tasks.

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

lower-case 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](#).

Acronym	Name
SSS	Simple Socket Server software module
LED	Light Emitting Diode Management software module
NETUTILS	Network Utilities software module
LWIP	Lightweight Internet Protocol software component
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.

Task	Description
<code>SSSInitialTask()</code>	Initializes the operating system data structures and creates the other tasks.
<code>SSSSimpleSocketServerTask()</code>	Listens for a socket connection and handles the connection. This task is written to handle only one connection at a time.
<code>NETUTILSDHCPTimeoutTask()</code>	Sets a static IP address after two minutes if lwIP has not been able to set a dynamic IP address due to lack of a successful DHCP server response.
<code>LEDManagementTask()</code>	Receives and executes commands via <code>SSSLEDCommandQ</code> passed from <code>SSSSimpleSocketServerTask()</code> .
<code>LED7SegLightshowTask()</code>	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 which implement lwIP network stack and process incoming packets. The lwIP network stack implementation task is created in the `lwip_stack_init()` call with a priority of

LWIP_TCPIP_TASK_PRIORITY. The packet receive-processing task gets created in `lwip_devices_init` with a priority level of `LWIP_RX_ETHER_TASK_PRIORITY`.

Inter-Task Communication Resources

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

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 our 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 our MicroC/OS-II LED Lightshow Semaphore. The semaphore is checked by the `LED7SegLightshowTask` each time it updates seven-segment LED displays, U8 and U9. The `LEDManagementTask()` grabs 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 the user sends the toggle lightshow command over the TCP/IP socket.

SSSAttainedIPAddressSem

`SSSAttainedIPAddressSem` is the handle to our MicroC/OS-II IP address semaphore. The semaphore is posted when an IP address has been set, either using a static value, or in response to a reply from a DHCP server. An application can `pend` on this semaphore in order to wait for a valid IP address before opening sockets, if desired.

lwIP Initialization

As described in “The lwIP Tasks” and “Initializing the Stack” sections of the *Ethernet & LightWeight IP* chapter in the *Nios II Software Developer’s Handbook*, lwIP must be initialized from the **Simple Socket Server** application code as follows:

- 2 lwIP functions must be called:
 - `lwip_stack_init()`, called from `main()` in **lwip_init.c**
 - `lwip_devices_init()`, called from `init_done_func()` in **lwip_init.c**
- 3 lwIP functions must be provided:
 - `init_done_func()` which is defined in **lwip_init.c** for this example
 - `get_mac_addr()` and `get_ip_addr()` which are defined in **network_utilities.c** for this example

The following tasks are created by `sys_thread_new()` in `init_done_func()`. This function is an excellent place to create any additional application tasks, via `sys_thread_new()`, which will use the lwIP sockets interface.

- `SSSSimpleSocketServerTask()`, defined in **simple_socket_server.c**, is created with priority `SSS_SIMPLE_SOCKET_SERVER_TASK_PRIORITY`.
- `NETUTILSDHCPTimeoutTask()`, defined in **network_utilities.c**, is created with priority `NETUTILS_DHCP_TIMEOUT_TASK_PRIORITY`.

Two more lwIP tasks are created by `sys_thread_new()`. These two system tasks make up the implementation of the lwIP networking stack sockets interface.

In `lwip_stack_init()`, the main TCP/IP networking stack processing task is created with priority `LWIP_TCPIP_TASK_PRIORITY`.

In `lwip_devices_init()`, the Ethernet packet receive-processing task is created with priority `LWIP_RX_ETHER_TASK_PRIORITY`. This lwIP task acts as the bottom half of a UNIX-style driver. When an Ethernet packet is received, the Ethernet receive-interrupt handler clears the bit and puts a message on the queue for this task. The high priority `lwip_dev_rx()` task is responsible for reading in packets and dispatching them to the lwIP main TCP/IP networking stack task.

Simple Socket Server Commands and Structures

The lwIP 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 which flow through the data flow diagram shown in [Figure 1-17 on page 1-22](#).

```
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

Below are the details of all of the simple socket server tasks and functions.

- `main()` (**lwip_init.c**)
 - Opens the LCD device
 - Calls `lwip_stack_init()`
 - Creates `SSSInitialTask()`
 - Calls `OS_start()` to begin multithreading
- `init_done_func()` (**lwip_init.c**)
 - Calls `lwip_devices_init()`

- Creates tasks which uses lwIP with `sys_thread_new()`:
`SSSSimpleSocketServerTask()`,
`NETUTILSDHCPTimeoutTask()`
- `SSSInitialTask()` (**simple_socket_server.c**) is used to initialize the operating system data structures and to create the other tasks. The task deletes itself because it is not needed after initialization completes. 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 will initialize correctly if that feature is enabled.

This task:

- Creates `SSSLEDCommandQ`, `SSSLEDLightshowSemaphore`, and `SSSLEDEventFlag` real-time operating system resources.
- Creates non-lwIP using tasks, including the LED tasks.
- Network utility task (**network_utilities.c**)
 - `NETUTILSDHCPTimeoutTask()` sets a static IP address after 2 minutes if an IP address has not been set due to a DHCP server response.
- `SSSSimpleSocketServerTask()` (**simple_socket_server.c**)
 - 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 could modify this task to create other tasks to 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 the `LEDManagementTask()` via the `SSSLEDCommandQ`.
- LED Tasks (**leds.c**)

- `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, controlled by a single command sent to the `LEDManagementTask()`, `CMD_LEDS_LIGHTSHOW`.

Important lwIP Concepts

The following topics are not easily categorized, but may have a significant impact on your design.

Error Handling

Error Handling of SSS (our application), lwIP, 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.

Creating Tasks that use the lwIP Sockets Interface

`sys_thread_new()` must be used to create any tasks which will use lwIP networking services. Tasks which do not use networking services should be created with `OSTaskCreate()`. Since all networking tasks are created with 2048-byte task stacks, care should be taken to consolidate networking functionality into networking tasks. 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.

On the other hand, be careful not to over utilize job distribution among several tasks at the same time. There are two reasons:

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

Task Priorities in the Simple Socket Server Design

The priority of the tasks in the application have an affect on how the application runs, or if the task functions correctly at all. The priorities of the tasks in the simple socket server design are discussed below:

- `LWIP_RX_ETHER_TASK_PRIORITY` sets the priority to a value of 3 for a task launched in `lwip_devices_init()`, called `lwip_dev_rx()`. This task is a lwIP task which acts as the second half of a UNIX-type Ethernet driver responsible for collecting packets from the Ethernet device and passing them to the TCP/IP stack. The priority of this task must be very high to avoid dropping any packets.
- `LWIP_TCPIP_TASK_PRIORITY` sets the priority to a value of 6 for the lwIP main TCP/IP networking stack task, called `tcPIP_thread()`, launched by `lwip_stack_init()`. This lwIP task processes packets passed from the `lwip_dev_rx()` task. In order to maximize TCP/IP packet-throughput rate, the priority of this task should be higher than application tasks that use lwIP.
- `NETUTILS_DHCP_TIMEOUT_TASK_PRIORITY` sets a value for priority of 2. The timeout task is intended for use during development only, and sets a static IP address (defined by `IPADDR{0-3}` macros in `simple_socket_server.h`) if no dynamic IP address has been assigned after two minutes. Such a delay is not appropriate for a deployed embedded application, which will benefit from assignment of a static IP address.
- `SSS_INITIAL_TASK_PRIORITY` is set to a very high value of 1 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 simply to get all the real-time operating system resources and tasks created before the other tasks start using the resources or interacting with each other.
- `SSS_SIMPLE_SOCKET_SERVER_TASK_PRIORITY` is set to a value of 10, a priority which is lower than the consumer task `LEDManagementTask()`. The priority of this application task is set to be lower than all of the software components' system service tasks. In general, this practice allows for the best overall scheduling latency, since the software component tasks are designed to operate for as short a period of time as possible.

At the same time, `SSSSimpleSocketServer_Task()` remains a high priority among the group of application task priorities, since it needs to service incoming Ethernet packets via the socket interface

which could come in at a high rate. The priority of this producer task is lower than the priority of its complementary queue message-consuming partner, `LEDManagementTask()`. This relative priority selection means that the `SSSSimpleSocketServerTask()` will likely never fill up the queue, even given a high rate of packet input. Thus, encounter of a queue full condition from the post to `SSSLEDCommandQ` is avoided.

`LEDManagementTask()` is scheduled to remove one of the messages from the `SSSLEDCommandQ` as soon as a command is posted. The queue full condition must still be handled though, due to a rare exception which is described in [Appendix B, Optimizing lwIP Throughput](#).

- `LED_MANAGEMENT_TASK_PRIORITY` has a priority value of 7. The `LEDManagementTask()` is a consumer task, and is assigned a priority higher than its complementary producer task, `SSSSimpleSocketServerTask()`. After `SSSSimpleSocketServerTask()` posts a message to the `SSSLEDCommandQueue`, the `LEDManagementTask()` task will get scheduled to consume that queue message before the producer task `SSSSimpleSocketServerTask()` can post another queue message.
- `LED_7SEG_LIGHTSHOW_TASK_PRIORITY` has a low priority value of 18. This task should be our lowest priority task, since it acts like our idle task. When the system is doing nothing else, it has time to perform the minimally important job of flashing random number patterns to the seven-segment LEDs. Using a low-priority task that blinks LEDs can be a handy debug tool to check your system for task starvation. Updated by the lowest priority task, the random pattern will be changed on the seven-segment LEDs only if all other higher priority tasks (with ready-to-run status) are getting scheduled for CPU processing time. This test can identify task starvation, but this test cannot detect a task dead-lock condition.

TCP/IP Throughput Performance

The following UDP throughput rates (in megabits per second) have been measured for lwIP on a closed network consisting of only a PC connected to a Nios development board, Stratix Edition. The closed network does not have any other network devices connected, so no extraneous traffic is generated. The benchmark program uses the lwIP sockets interface and

the MicroC/OS-II real-time operating system. The board runs at 50 MHz and is configured with the standard reference design, with the exception that the Nios II/s core is changed to a Nios II/f core. See [Table 1-3](#).

Type	Rate	Measurement
UDP Transmit	5.16	Mbps
UDP Receive	3.44	Mbps

TCP throughput performance will be lower due to the overhead of the protocol.

Task Stack Size

Task stack space requirements will vary depending on how the Nios II processor, HAL, RTOS, and individual software components are configured. A quick empirical check of the `Stk[]` array values at run-time, 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 will reveal differing values representing the used portion of the stack followed by a lot of zeros where the stack has not yet reached. The amount of zeros until the beginning of the next adjacent task stack shows how deep the stack has grown since the last system reset.

Each task that will use the lwIP networking stack must be created with the function `sys_thread_new()`. The implementation for `sys_thread_new()` is located in `sys_arch.c`. `sys_thread_new()` creates all tasks with a stack size of 2 kilowords (8192 bytes). Since this parameter is not configurable through the `sys_thread_new()` function, be aware that if you created a networking task which requires a large amount of stack, you will need to modify `sys_thread_new()`, defined in `sys_arch.c`, to use a larger stack size.

All tasks which make run-time library calls have space allocated from the top of 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 stack space, leaving just over 1 Kbyte for all the other stack needs of the task. Allocation of large local structures of a Kbyte or more in a task created with `sys_thread_new()` would cause all variables locally declared to be positioned in another task's stack! No compiler, linker, or run-time error messages are generated to warn you in this scenario, which would result in indeterminate run-time behavior that is difficult to diagnose. Therefore,

tasks which utilize the sockets API should be designed such that any processing of packet data is handed off to other tasks which have been created with sufficient task stack sizes.

For more details on the `_reent` structure, see both 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 `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 for 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.



Appendix A. Hardware Setup Details

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 on how to install the software and connect the Nios development board with the USB Blaster cable, see the *Nios II Development Kit Getting Started User Guide*.

For the Ethernet connection, connect both your PC and the Nios development board to the same subnet on a network. To assign IP addresses to the MAC addresses, use either a DHCP server, or set a static IP address in the source header file `simple_socket_server.h`.

The full-featured reference hardware design for the Nios development boards includes the Ethernet device required by this lwIP tutorial. The Ethernet device included in these reference designs, along with the physical MAC/PHY on each of the Stratix II, Stratix, Stratix Professional, 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+0x030f`. The Ethernet peripheral base address settings for the full-featured hardware reference designs, along with IRQ setting, can be examined in `system.h`.



Introduction

This section discusses methods to improve throughput performance of the lwIP TCP/IP stack. The following factors affect TCP/IP throughput:

- The configuration values of the lwIP software component
- Compiler flags to improve over-all code efficiency
- Network congestion

The following sections discuss configuring the lwIP software component and setting compiler flags.

lwIP Configuration Values

This tutorial used the default values for the lwIP software component, which are a good starting point for most lwIP applications. The values in [Table B-1](#) provide higher throughput performance. However, these values present a trade-off between speed and memory size.

The most significant parameter affecting TCP receive performance in the table below is `TCP_WND`. The most significant parameter affecting TCP transmit performance in the table below is `TCP_SND_BUF`.

To access the lwIP configuration values in the Nios II IDE, open the **System Properties** page for the system library project, select **Software Components** and expand the lwIP entry to reveal each of these pages as shown in [Figure 1-8 on page 1-11](#), [Figure 1-9 on page 1-12](#), and [Figure 1-10 on page 1-13](#).

<i>Table B-1. lwIP Speed-Optimized Configuration Values (Part 1 of 3)</i>				
system.h Macro	lwIP Options Page	Default Value	Optimal Value	Description
<code>MEM_SIZE</code>	Memory	32768	65536	TCP/IP Heap Size
<code>MEMP_NUM_PBUF</code>	Memory	32	32	Maximum number of buffers sent without copying.
<code>MEMP_NUM_NETBUF</code>	Memory	32	32	Maximum number of packet buffers passed between the application and the stack threads.

<i>Table B-1. lwIP Speed-Optimized Configuration Values (Part 2 of 3)</i>				
system.h Macro	lwIP Options Page	Default Value	Optimal Value	Description
MEMP_NUM_UDP_PCB	UDP	8	8	Maximum number of UDP sockets.
MEMP_NUM_TCP_PCB	TCP	8	8	Maximum number of active sockets.
MEMP_NUM_TCP_PCB_LISTEN	TCP	2	2	Maximum number of listening sockets.
MEMP_NUM_API_MSG	Memory	32	32	Maximum number of pending API calls from the application to the protocol stack thread.
MEMP_NUM_TCPIP_MSG	Memory	32	32	Maximum number of messages passed from the protocol stack thread to the application.
ARP_TABLE_SIZE	ARP	10	10	Size of the ARP table
IP_FORWARD	IP	0	0	Forward IP packets (only useful with 2 or more network interfaces).
DHCP_DOES_ARP_CHECK	DHCP	1	1	Use ARP protocol to verify that the DHCP assigned address is not already in use.
LWIP_UDP	Lightweight TCP/IP Stack (Main page)	1	1	Enable UDP Protocol
LWIP_TCP	Lightweight TCP/IP Stack (Main page)	1	1	Enable TCP Protocol
TCP_WND	TCP	2048	32768	Maximum window size (receive buffer space in bytes).
TCP_MAXRTX	TCP	4	4	Maximum retransmissions
TCP_SYNMAXRTX	TCP	4	4	Maximum retransmissions of SYN frames.
TCP_MSS	TCP	1476	1476	Maximum Segment Size

Table B-1. lwIP Speed-Optimized Configuration Values (Part 3 of 3)

system.h Macro	lwIP Options Page	Default Value	Optimal Value	Description
TCP_SND_BUF	TCP	32768	32768	Maximum Send Buffer Space
LWIP_STATS	Lightweight TCP/IP Stack (Main page)	0	0	Enable Statistics
LWIP_DHCP	Lightweight TCP/IP Stack (Main page)	1	1	Use DHCP
ICMP_TTL	Lightweight TCP/IP Stack (Main page)	255	255	Time to Live
PBUF_POOL_BUFSIZE	Lightweight TCP/IP Stack (Main page)	1536	1536	Maximum packet buffer size
PBUF_POOL_SIZE	Lightweight TCP/IP Stack (Main page)	16	16	Number of packet buffers
LWIP_DEFAULT_IF	Lightweight TCP/IP Stack (Main page)	lan91c111	lan91c111	Default MAC Interface

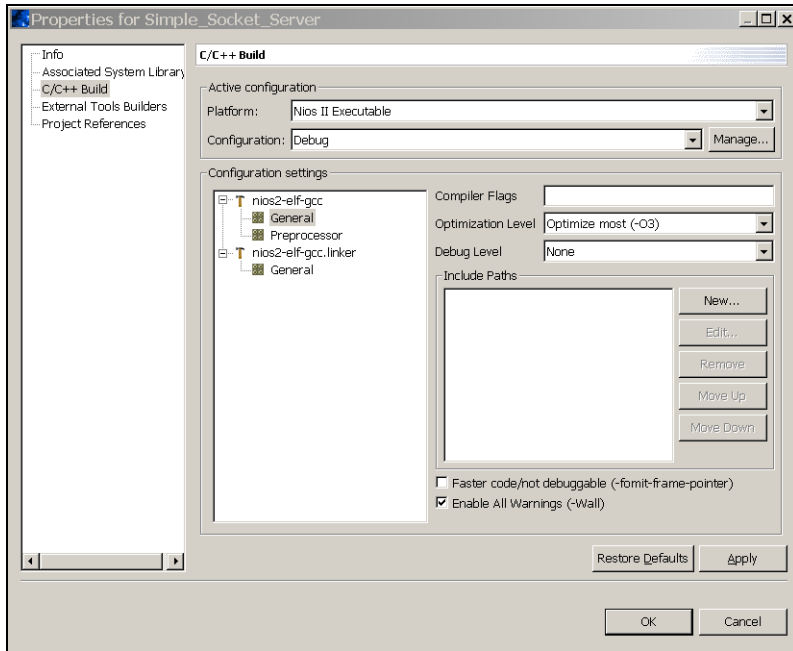
Configuring Optimization & Debug Levels

In addition to configuring lwIP for performance, you can also tune the compiler flag for high speed performance by selecting the highest level of optimization, which is 3. You can reduce the code size footprint by setting the level of debugging information to **None**. Both of these flags are controlled by Nios II IDE fields on the project **Properties** page, under the **C/C++ Build** section highlighted on the left pane. The **Debug Level** and **Optimization Level** must be set twice to adjust both the application project and the system library project.

1. In the Nios II IDE **C/C++ Projects** view, right-click the **simple_socket_server_0** application project and choose **Properties**. The **Properties** page appears.
2. Select **C/C++ Build** in the left-hand pane of the **Properties** page.
3. Select **General** under **Configuration** settings on the **Properties** page.

4. Select **Optimize Most (-O3)** as the **Optimization Level** value, and select **None** as the **Debug Level** value, as shown in **Figure B-1**.

Figure B-1. Compiler Flag Optimized Configuration Values



Set the same options in the properties page for the **full_system_lib** system library project.