lwIP for CrossCore® Embedded Studio 1.0.0
User's Guide

Revision 1.0, March 2012

Part Number
82-100119-01

Analog Devices, Inc.
One Technology Way
Norwood, Mass. 02062-9106

ANALOG DEVICES

Copyright Information

Printed in the USA.

Disclaimer

Trademark and Service Mark Notice

CONTENTS

PREFACE

Thank you for purchasing the lwIP (light-weight Internet Protocol) for CrossCore® Embedded Studio (CCES) 1.0.0 add-in, the Analog Devices implementation of the open-source TCP/IP stack for embedded platforms.

Purpose

The *lwIP for CrossCore Embedded Studio 1.0.0 User's Guide* describes how to use the lwIP add-in software.

Intended Audience

The primary audience for this manual is a programmer who is familiar with Analog Devices processors and TCP/IP protocol suite. This manual assumes that the audience has a working knowledge of the appropriate processor architecture and instruction set. Programmers who are unfamiliar with Analog Devices processors can use this manual, but should supplement it with other texts (such as the appropriate hardware reference and programming reference manuals) that describe your target architecture.

Manual Contents

The manual consists of:

- Chapter 1, "lwIP Introduction" on page 1-1
  Describes the open-source TCP/IP stack.
- Chapter 2, "lwIP Add-in Overview" on page 2-1
  Describes the product.
- Chapter 3, "Developing lwIP Applications" on page 3-1
  Describes how to use the product.
- Chapter 4, "lwIP Examples" on page 4-1
  Describes example programs included with the product.

What's New in this Manual

This is the first revision of the *lwIP for CrossCore Embedded Studio 1.0.0 User's Guide.*

Technical or Customer Support

You can reach Analog Devices, Inc. Customer Support in the following ways:

- Visit the Embedded Processing and DSP products Web site at
  `http://www.analog.com/processors/technical_support`
- E-mail tools questions to
  `processor.tools.support@analog.com`
- E-mail processor questions to
  `processor.support@analog.com (World wide support)`
  `processor.china@analog.com (China support)`

- Phone questions to **1-800-ANALOGD** (USA only)
- Contact your Analog Devices, Inc. local sales office or authorized distributor
- Send questions by mail to:
  ```
  Analog Devices, Inc.
  One Technology Way
  P.O. Box 9106
  Norwood, MA 02062-9106
  USA
  ```

Supported Processors

The lwIP for CrossCore Embedded Studio 1.0.0 supports the ADSP-BF60x Blackfin processors from Analog Devices.

Refer to the CrossCore Embedded Studio online help for a complete list of supported processors.

Product Information

Product information can be obtained from the Analog Devices Web site and CrossCore Embedded Studio online help.

Analog Devices Web Site

The Analog Devices Web site, www.analog.com, provides information about a broad range of products—analog integrated circuits, amplifiers, converters, and digital signal processors.

To access a complete technical library for each processor family, go to http://www.analog.com/processors/technical_library. The manuals selection opens a list of current manuals related to the product as well as a

link to the previous revisions of the manuals. When locating your manual title, note a possible errata check mark next to the title that leads to the current correction report against the manual.

Also note, MyAnalog.com is a free feature of the Analog Devices Web site that allows customization of a Web page to display only the latest information about products you are interested in. You can choose to receive weekly e-mail notifications containing updates to the Web pages that meet your interests, including documentation errata against all manuals. MyAnalog.com provides access to books, application notes, data sheets, code examples, and more.

Visit MyAnalog.com to sign up. If you are a registered user, just log on. Your user name is your e-mail address.

EngineerZone

EngineerZone is a technical support forum from Analog Devices. It allows you direct access to ADI technical support engineers. You can search FAQs and technical information to get quick answers to your embedded processing and DSP design questions.

Use EngineerZone to connect with other DSP developers who face similar design challenges. You can also use this open forum to share knowledge and collaborate with the ADI support team and your peers. Visit http://ez.analog.com to sign up.

Notation Conventions

Text conventions used in this manual are identified and described as follows.

Example	Description
Close command (**File** menu)	Titles in bold style reference sections indicate the location of an item within the CrossCore Embedded Studio environment's menu system (for example, the **Close** command appears on the **File** menu).
`{this | that}`	Alternative required items in syntax descriptions appear within curly brackets and separated by vertical bars; read the example as `this` or `that`. One or the other is required.
`[this | that]`	Optional items in syntax descriptions appear within brackets and separated by vertical bars; read the example as an optional `this` or `that`.
`[this,…]`	Optional item lists in syntax descriptions appear within brackets delimited by commas and terminated with an ellipsis; read the example as an optional comma-separated list of `this`.
`.SECTION`	Commands, directives, keywords, and feature names are in text with `letter gothic` font.
`filename`	Non-keyword placeholders appear in text with italic style format.
	**Note:** For correct operation, ... A Note provides supplementary information on a related topic. In the online version of this book, the word **Note** appears instead of this symbol.
	**Caution:** Incorrect device operation may result if ... **Caution:** Device damage may result if ... A Caution identifies conditions or inappropriate usage of the product that could lead to undesirable results or product damage. In the online version of this book, the word **Caution** appears instead of this symbol.
	**Warning:** Injury to device users may result if ... A Warning identifies conditions or inappropriate usage of the product that could lead to conditions that are potentially hazardous for devices users. In the online version of this book, the word **Warning** appears instead of this symbol.

1 LWIP INTRODUCTION

lwIP (light-weight Internet Protocol) is the widely-accepted TCP/IP stack implementation for embedded platforms. The lwIP TCP/IP stack has been ported to Analog Devices, Inc. family of Blackfin embedded processors. The ported stack uses a standard Ethernet device driver interface, which allows the drivers to be used with different Ethernet controllers. All of the drivers adhere to the ADI device driver model and use CrossCore® Embedded Studio's System Services Libraries (SSL).

lwIP supports most of the standard protocols in the TCP/IP protocol suite and applications conforming to the Berkeley-alike socket (BSD) interface.

lwIP supports the following protocols:

- Internet Protocol (IP)
- Internet Control Message Protocol (ICMP)
- User Datagram Protocol (UDP)
- Transmission Control Protocol (TCP)
- Dynamic Host Configuration Protocol (DHCP)
- Address Resolution Protocol (ARP)
- Berkeley-alike Socket API

For more information on the lwIP stack, visit http://www.sics.se/~adam/lwIP/.

The stack sources are maintained at:
http://savannah.nongnu.org/cgi-bin/viewcvs/lwIP/lwIP/.

2 LWIP ADD-IN OVERVIEW

The lwIP add-in to CrossCore Embedded Studio (CCES) consists of the core layers of the TCP/IP protocol and low-level wrapper layer that interacts with Ethernet device drivers. The Ethernet device drivers are provided as part of the CCES installation.

The lwIP add-in support package includes various networking examples and online documentation. The add-in also includes a ready-to-use template framework that facilitates development of network applications. Refer to "Developing lwIP Applications" on page 3-1 for more information about the lwIP application framework.

The following topics are covered in this chapter:

lwIP System Architecture

Figure 2-1 is a simplified view of the lwIP system architecture.

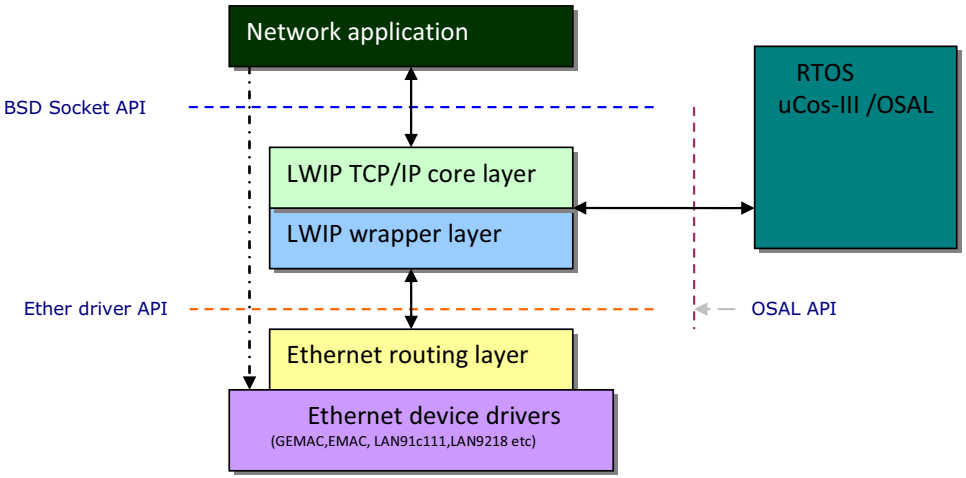

Figure 2-1. lwIP System Architecture

Ethernet Device Drivers

Every Ethernet controller has a hardware-specific driver. All Ethernet drivers interact with the lwIP subsystem via a standard driver interface. The driver exports a driver entry-point. The lwIP subsystem uses the entry-point to access the Ethernet driver functionality. During system initialization, applications open the underlying device driver and supply the handle to the lwIP subsystem. Drivers can export additional functions for applications to use. These functions are accessed directly by applications without going through the entry-point.

The Ethernet driver entry-point includes the following functions.

- adi_ether_Open – opens the Ethernet driver
- adi_ether_Read – supplies receive buffers to the driver
- adi_ether_Write – transmits data in the supplied buffer
- adi_ether_Close – closes the device driver
- adi_ether_GetLinkStatus – returns the network link status
- adi_ether_AddMulticastFilter – enables multicast for the given group address
- adi_ether_DelMulticastFilter – disables multicast for the given group address
- adi_ether_GetBufferPrefix – returns the buffer prefix of underlying driver
- adi_ether_GetMACAddress – returns the MAC address
- adi_ether_SetMACAddress – sets the MAC address
- adi_ether_EnableMAC – enables the MAC (starts the Ethernet driver)

Most of the listed driver functions are used by an lwIP subsystem. A typical application uses these three functions:

- adi_ether_Open
- adi_ether_GetLinkStatus
- adi_ether_EnableMAC

lwIP Core

The lwIP core layer supports TCP/IP protocols. The lwIP core is distributed as a library with default protocol options. Advanced users can configure the protocol options using the easy-to-use library plug-in software.

The project directory is:
`<lwip_install_path>\lwip\blackfin\lib\lwip\contrib\ports\blackfin\projects\lwipv4lib`.

See the CCES online help for more information about the library plug-in.

lwIP Wrapper

The lwIP wrapper is an interface between the lwIP core and low-level Ethernet drivers. The wrapper provides functionality for initializing and starting a stack. As part of the initialization process, the wrapper supplies buffers to the Ethernet driver. The stack callback handler is part of the wrapper function, which receives frames from the underlying Ethernet driver and queues it for further processing by the stack. Similarly, the stack-supplied frames are transmitted using the Ethernet driver.

The lwIP wrapper exports initialization APIs for the application to set up an lwIP subsystem. The following APIs are exported by the wrapper:

```
int adi_lwip_Initstack(
           const unsigned int buffer_length, char *buffer_start);
int adi_lwip_Startstack(void);
void adi_lwip_Stopstack(void);
int adi_lwip_Setdrvhandles(
           int num_services, ADI_ETHER_HANDLE *pdd_handles);
void adi_lwip_Stackcallback(
           void *arg1, unsigned int event, void* pack_list);
```

The project directory is
`<lwip_install_path>\lwip\blackfin\lib\lwip\contrib\ports\blackfin\projects\lwIPwrapperlib`.

lwIP RTOS Interface

The lwIP TCP/IP stack with a socket-level interface requires an operating system. Analog Devices' port of lwIP utilizes operating system functions via a standard interface. The current lwIP distribution uses the µC/OS-III™ for the CrossCore Embedded Studio, the operating system from Micrium, Inc.

See the CCES online help for more information about µC/OS-III.

3 DEVELOPING LWIP APPLICATIONS

Adding network functionality to a Blackfin processor-based application is a fairly easy task with the lwIP add-in to the CrossCore Embedded Studio development tools.

The following topics are covered in this chapter:

Creating Network Applications

The following topics are covered in this section:

Adding and Removing an lwIP Add-in

Network functionality is added to an application by adding lwIP to the CCES project. Once you install the lwIP add-in, you can add the lwIP components to a project.

In CCES, double-click the project's `system.svc` file in a project view, choose **Analog Devices' lwIP TCP/IP stack**, and click **Add.** The `sys-`

tem.svc in your project is core-specific, so for a dual-core processor (such as ADSP-BF60x), you can add lwIP to one of the cores.

Adding lwIP functionality to a project also adds template code, lwIP libraries, and include paths required for network applications. Refer to "lwIP Template Files" on page 3-2 and "Required User Changes to lwIP Projects" on page 3-4 for more information.

Similarly, you can remove the **Analog Devices' lwIP TCP/IP stack** functionality from a CCES project by clicking **Remove** in the system.svc file editor.

lwIP Template Files

By default, the lwIP add-in adds template code to your project. The lwIP template code facilitates development of network application. It generates code that initializes and starts an lwIP subsystem. By default, the lwIP uses the Dynamic Host Configuration Protocol (DHCP) to get the IP address from a DHCP server. As a result, you need to connect your hardware to a network that has an available DHCP server.

The following files are added to the project with the lwIP add-in.

lwip_sysboot_task.c

The lwip_sysboot_task.c file contains the core routines and lwIP system boot task for the lwIP subsystem booting. The boot task is responsible for opening and configuring the Ethernet drivers, initializing, and starting the network stack.

The adi_lwip_Init() function is responsible for creating the lwIP system task. The adi_lwip_Init() function is called from adi_init_Components(). Once the lwIP add-in becomes part of the project, the adi_lwip_Init() function is added to the adi_init_Components() routine automatically. The adi_init_Components() function typically is called from the application's main().

Upon successful initialization, the lwIP system boot task prints the IP address obtained from the DHCP server. It also posts the `g_semLWIPBootComplete` semaphore. Other network-dependent application boot tasks can pend on this semaphore for the network boot to complete. All tasks pending on this boot semaphore will be released.

If case of a network connection problem, the lwIP system boot task pends on a semaphore and periodically checks for a network link.

lwip_sysboot_task.h

The `lwip_sysboot_task.h` file contains global configuration settings for an lwIP subsystem and exported functions. The critical configuration settings are:

```
/*! LWIP task stack size */
#define APP_OS_CFG_LWIP_TASK_STK_SIZE (2048)

/*! LWIP task priority   */
#define APP_OS_CFG_LWIP_TASK_PRIO      (6)

/*! Number of receive DMA descriptors */
#define NO_RCVES    (10)

/*! Number of transmit DMA descriptors */
#define NO_XMITS    (10)
```

For DMA-based Ethernet drivers, an application can increase the number of supplied DMA descriptors to the driver by changing the `NO_RCVES` and `NO_XMITS` macros.

softswitch_cfg.c

The `softswitch_cfg.c` file contains the processor-specific switch configuration routines.

lwip_app.c

The `lwip_app.c` file contains the application-specific configuration settings that can be modified via the lwIP graphical-user interface (the **lwIP** tab). Refer to "lwIP Add-in Overview" and the CCES online help for more information.

The template code creates the lwIP boot task but does not start the operating system. It is the user's responsibility to start the operating system. Refer to "Required User Changes to lwIP Projects" on page 3-4 for details.

Required User Changes to lwIP Projects

Once you add the lwIP add-in to your application project, the lwIP system boot task, lwIP library, header files, and relevant include paths are added to the project as well.

To complete your network application, the following steps are required.

1. Ensure the `main()` function is present in the project; otherwise, add the function to the project.
2. Ensure the µC/OS-III component is added to the project; otherwise, add µC/OS-III to the project.
3. Start the µC/OS-III using `OSStart()`. The code template is as follows:

```
#include <os.h>
int main(void)
{
 OS_ERR osErr;
```

```
/*
The default startup code does not include any functiona-
lity to allow core 0 to enable core 1. A convenient way to
enable core 1 is to use 'adi_core_1_enable' function.
*/

adi_core_1_enable();

/* Initialize managed drivers and/or services */
adi_initComponents();

/* Begin adding your custom code here */
 OSStart(&osErr);

 return 0;
}
```

4. Place the default heap in external memory and increase it to a minimum of 4 MB. Follow these steps:

    a. Double click the project's `system.svc` file. The **System Configuration Overview** page of the system configuration utility appears.

    b. If the **Startup Code/LDF** tab does not show at the bottom of the page, click **Add** to install the **Startup Code/LDF** component.

    c. Click the **Startup Code/LDF** tab. The **Startup Code/LDF** page appears.

d. Click **LDF** to open the **LDF Configuration** page (Figure 3-1).

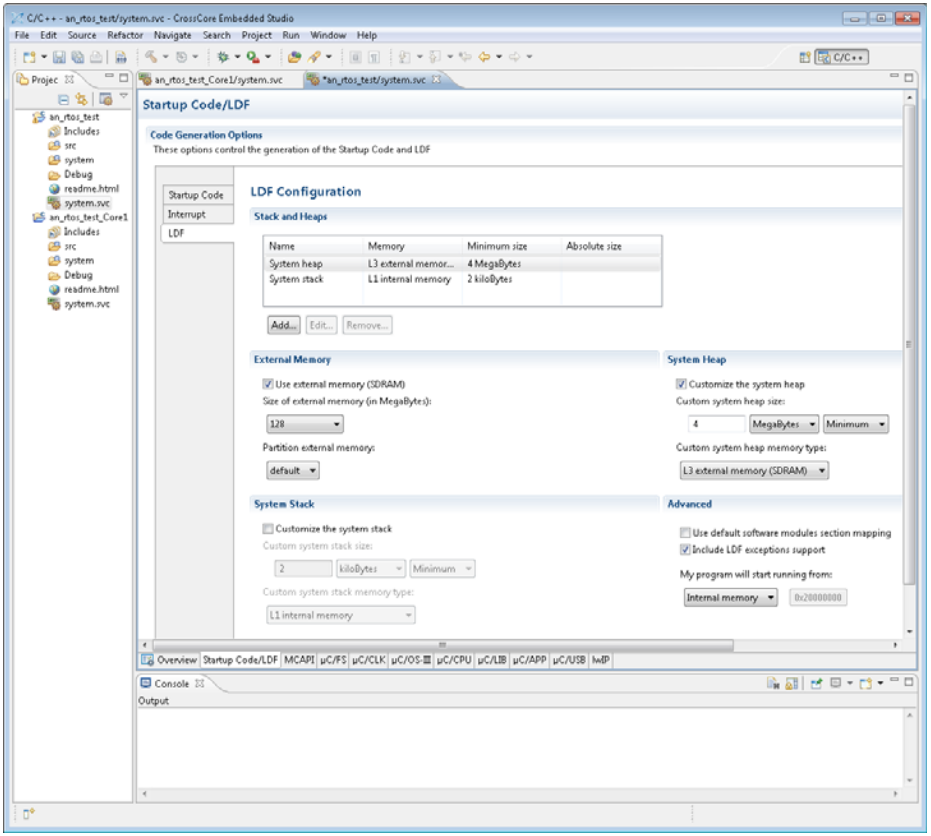

Figure 3-1. Heap Configuration

e. Enable **Use external memory (SDRAM)**.

f. Enable **Customize the system heap**.

g. Set **Custom system heap memory type to L3 external memory (SDRAM)**.

h. Set **Custom system heap size** to **4 MegaBytes Minimum**

5. Build and run your network application.

lwIP Application Graphical Interface

Use the lwIP's graphical user interface (GUI) to configure various application-specific network parameters, such as IP addresses, subnet masks, and gateway addresss. Use the interface to configure the Media Access Control (MAC) address and the number of driver buffers.

Double-click the project's `system.svc` file and click the **lwIP** tab. The **lwIP Application Configuration** page appears (see Figure 3-2).

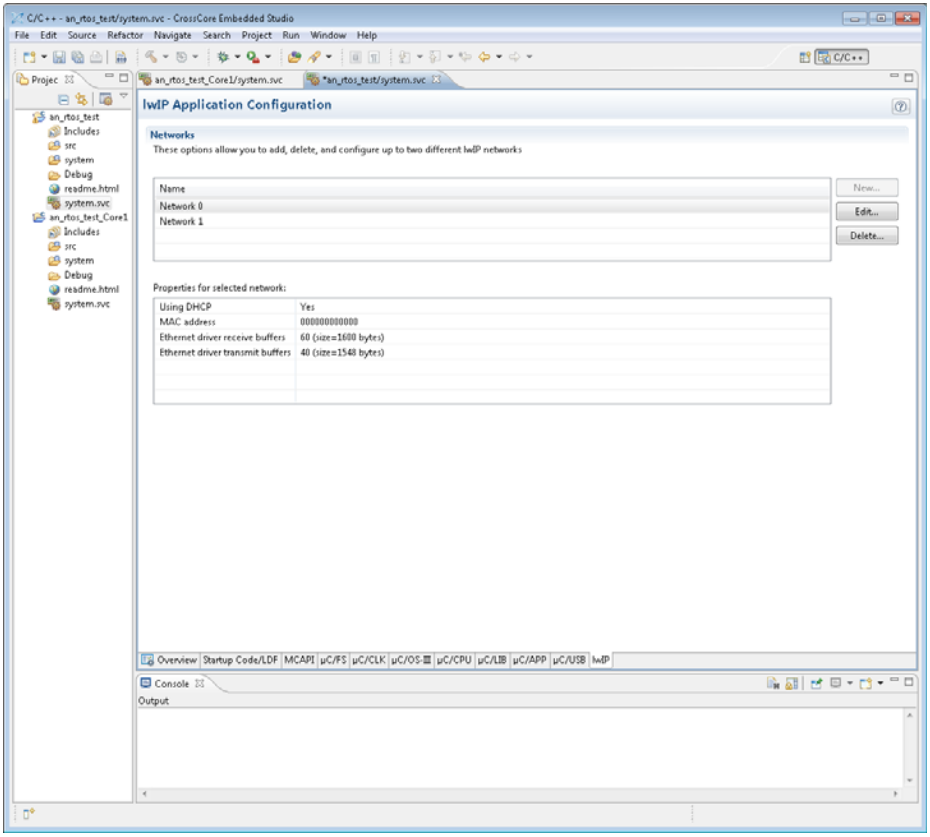

Figure 3-2. lwIP Application Configuration Page

Use the page to configure the network parameters, as described in:

- “Configuring a Static IP Address” on page 3-9
- “Configuring a Subnet Mask” on page 3-9

Configuring a Static IP Address

To configure the static IP address:

1. Click the project's `system.svc` file to open the system configuration utility.
2. Click the **lwIP** tab to open the **lwIP Application Configuration** page (see Figure 3-2).
3. Double click the network ID (**Network 0** is shown in Figure 3-2) to open a dialog box for the selected network.
4. Disable the **Use DHCP** option and enter the address in **IP address**.
5. Click **OK** to save the changes.

Configuring a Subnet Mask

A typical configuration of the subnet mask is `255.255.255.0`. To configure the subnet mask:

1. Click the project's `system.svc` file to open the system configuration utility.
2. Click the **lwIP** tab to open the **lwIP Application Configuration** page (see Figure 3-2).
3. Double click the network ID (**Network 0** is shown in Figure 3-2) to open a dialog box for the selected network.
4. Enter the mask value in **Subnet Mask**.
5. Click **OK** to save the changes.

Configuring a MAC Address

Typically, a MAC address is picked up from the EEPROM or flash memory of the EZ-KIT Lite®/EZ-Board® hardware.

To configure the MAC address:

1. Click the project's `system.svc` file to open the system configuration utility.
2. Click the **lwIP** tab to open the **lwIP Application Configuration** page (see Figure 3-2).
3. Double click the network ID (**Network 0** is shown in Figure 3-2) to open a dialog box for the selected network.
4. Enter the address in **MAC address (hex)**.
5. Click **OK** to save the changes.

Configuring a Driver Buffer

Receive and transmit driver buffers are used by an Ethernet device driver to send and receive Ethernet frames.

By default, there are 60 (1600 bytes each) Ethernet receive buffers and 40 (1548 bytes each) transmit buffers. Applications can increase or decrease the number of buffers and buffer size, depending on requirements.

The number of receive and transmit driver buffers can be configured through the lwIP graphical interface or application code. In order to do so, applications must change the `ETHER_STACK_SIZE` macro in the `lwip_sysboot_task.c` file. See "lwIP Template Files" on page -2 for more information.

To configure a driver buffer through the graphical interface:

1. Click the project's `system.svc` file to open the system configuration utility.
2. Click the **lwIP** tab to open the **lwIP Application Configuration** page (see Figure 3-2).
3. Double click the network ID (**Network 0** is shown in Figure 3-2) to open a dialog box for the selected network.
4. Enter the number of buffers and size under **Ethernet driver buffers**.
5. Click **OK** to save the changes.

4 LWIP EXAMPLES

The lwIP examples are installed with the lwIP add-in package. The examples have the same directory structure and use a pre-built demo version of the Micrium µC/OS-III libraries. Applications are expected to use the µC/OS-III add-in. Refer to the `readme.html` file provided with each example for information on how to convert an example to use Micrium µC/OS-III add-in/sources instead of pre-built demo libraries.

The lwIP example directory is `<lwip_install_path>\lwip\Blackfin\examples`.

Figure 4-1 is a pictorial representation of the directory structure of an lwIP example.

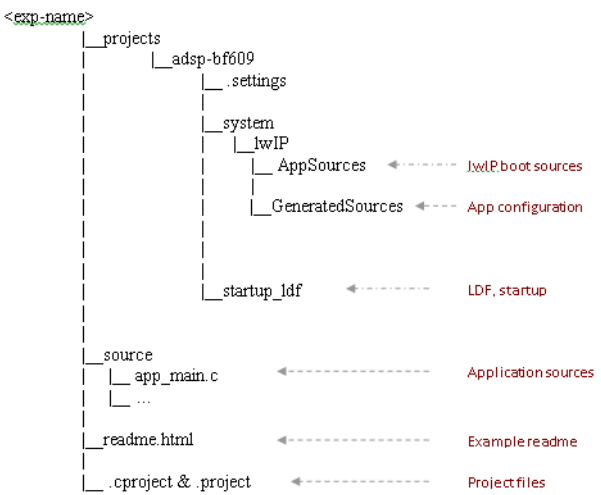

Figure 4-1. An lwIP Example Directory Structure

inetd Example

The `inetd` example has three servers: character generator (port #19), echo (port #9), and discard (port #7). Upon startup, the example starts the three servers and waits for the connections on the specified ports for the client connection. For each client connection, the servers spawn a task, and the client task handles the client connection. The example uses TCP sockets and exercises TCP socket API calls. Refer to the `readme.html` file in the example's directory for details.

dns_client Example

The `dns_client` example demonstrates the Domain Name System (DNS) client support. The DNS client running on a Blackfin processor obtains the IP address of domain name `www.analog.com`. The example uses UDP sockets. Refer to the `readme.html` file in the example's directory for details.

fileserver Example

The `fileserver` example provides Windows host support to enable a standard C/C++ file input/output from the Blackfin processor to be received and sent to the host over TCP/IP connection. The application consists of a Windows MFC-based host program and a Blackfin fileserver program. The example uses TCP sockets. Refer to the `readme.html` file in the example's directory for details.

multicast Example

The `multicast` example demonstrates how a single node can send data to many destinations by making a single call on the multicast transport ser-

vice. The multicast application consists of a Windows MFC-based host program, multicast receiver, and a multicast sender. The multicast sender is a Blackfin processor-based application that sends out a "`Hello World`" message to the multicast IP address "`225.0.0.37`" at port 12345. The Windows application (multicast receiver) receives the message from the same multicast IP address. The multicast receiver can be run on several different PCs to receive the message simultaneously. Refer to the `readme.html` file in the example's directory for details.

5 LWIP LIBRARY CONFIGURATION

Advanced users can configure the lwIP library according to their application requirements. Refer to the online help for details.

I INDEX

U