XILINX®

XAPP663 (v1.1.1) August 30, 2004

TCP/IP on Virtex-II Pro Devices Using lwIP

Author: Sathyanarayanan Thammanur and Chris Borrelli

Summary

TCP/IP is a communication protocol stack designed to provide a reliable data stream between two hosts. It is a popular means of communicating data over a network. Most people use the protocol every day to check email, browse the web, instant message, and download files. TCP/IP is also becoming more utilized in embedded systems.

This application note explores the use of an open source TCP/IP stack, referred to as the light-weight Internet protocol stack (lwIP), on Virtex-II Pro™ processors (PowerPC™ and MicroBlaze™). An example reference design is provided that illustrates remote interaction with a server running on the Virtex-II Pro development board designed by Insight/Memec.

Introduction

TCP/IP is usually implemented in software as a service of an operating system (Linux, NetBSD, and so forth.). The Xilinx Virtex-II Pro family of devices contains an embedded PowerPC 405 processor and a soft core MicroBlaze processor, each capable of running an operating system. However, small projects might not require a full operating system. This application note uses a stand-alone protocol stack (without an operating system).

The TCP/IP stack used in the reference design is a light-weight Internet protocol stack (lwIP). lwIP is a TCP/IP implementation for small embedded systems where no operating system is required, although it can be used with an operating system. lwIP is briefly discussed in this application note. For more information on lwIP or to download the source contact:

`http: //savannah.nongnu.org/projects/lwip/`

The example lwIP application provides a simple demonstration of TCP/IP communication using Virtex-II Pro devices. The application allows remote access to a TCP based echo server running on the board. A system diagram of the reference design hardware and software components is shown in Figure 1.

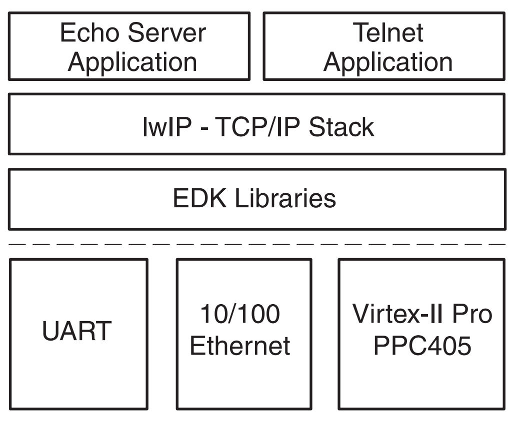


Figure 1: **Reference System Hardware and Software Block Diagram**

PowerPC Reference Design

The hardware design in the FPGA is a simplified version of the "Embedded Reference Design". The design was simplified to support only the hardware devices used in the example software application. A diagram of the hardware design involving the PowerPC processor is shown in Figure 2. lwIP stack requires a timer for its functionality. The built-in hardware timer in PowerPC is used for performing timer functionality in lwIP.

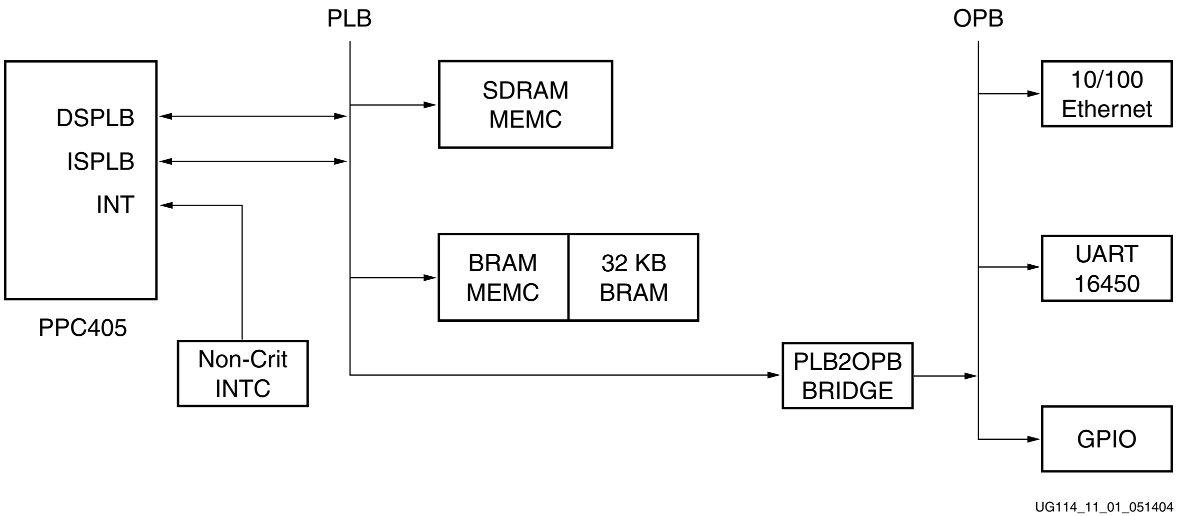


Figure 2: **PowerPC System Design for lwIP-Based ECHO Server**

MicroBlaze Reference Design

MicroBlaze hardware design is almost similar to the PowerPC design. An OPB based timer is included in the design as MicroBlaze does not provide a built-in timer functionality. A diagram of the hardware design for MicroBlaze processor is shown in Figure 3.

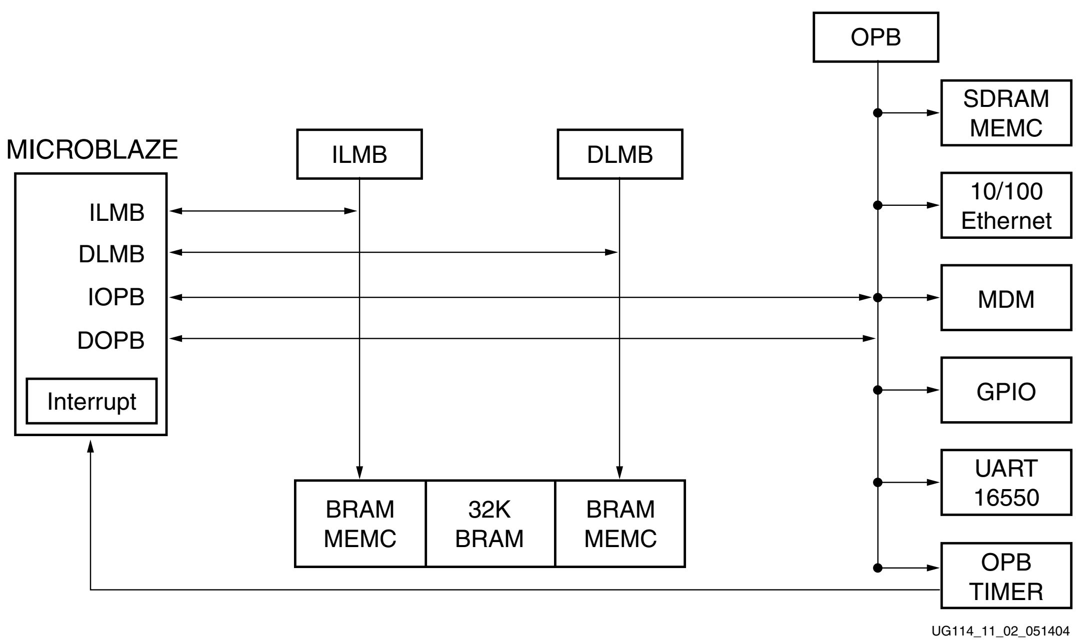


Figure 3: **MicroBlaze System Design for lwIP-Based ECHO Server**

Using lwIP to Build Network Applications

lwIP has a few different application interfaces. There are BSD-like socket interfaces available and a raw API. The raw API is the lowest level of interface to the stack. It has two different modes of operation: Event and Callback. The Callback mode is used in the accompanying reference design.

lwIP Reference Design Structure

Software applications built using lwIP must follow a certain structure. For the stack to maintain internal timers for TCP round trip calculations and other TCP facilities, timer functions provided by lwIP must be called periodically. The reference design uses the PPC405 64-bit hardware timer in its idle loop to generate a time base. It calls the lwIP functions after the correct number of cycles have passed since the last iteration. Interrupts are not used.

Initializing lwIP

lwIP requires the application to call several initialization functions before it can be used. The following code is an example main() function to illustrate the initialization required for lwIP and the network interface, `netif`.

```
/******************************************************************
* Call lwIP Initialization Functions
*****************************************************************/
sys_init();
mem_init();
memp_init();
pbuf_init();
netif_init();
tcp_init();
/******************************************************************
* Setup our 1 network interface (netif)
*****************************************************************/
netif = netif_add(&ipaddr,&netmask,&gw,&XEmacIF_ConfigTable[0]
                 xemacif_init, ip_input);
netif_set_default(netif);

/******************************************************************
* IDLE Loop - handle lwIP timer functions and poll EMAC
*****************************************************************/
while (1) {
    while (waiting_for_timer) {
        /* poll network interface */
        xemacif_input(default_netif);
        /* get current PPC405 timer value */
        XTime_GetTime(&ml_new);
        /* check to see if we reached the terminal count */
        if (ml_new >= ml_base) {
            waiting_for_timer = 0;
            ml_base = ml_new + ml_offset;
        }
    }
    my_tmr() /* calls the various lwIP timer functions */
    /* wait for next terminal count */
    waiting_for_timer = 1;
}
```

lwIP Raw API

The raw API of lwIP is the lowest level interface to the stack. It is also the most efficient interface because it provides direct access to the stack rather than providing extra buffering and message passing features. The following sections describe the most used interface functions. A more detailed description of the raw API is found in the lwIP source tree (lwip/doc/rawapi.txt.) Asynchronous network events (data received, connection established, etc.) are communicated to the software application through callback functions. These callbacks are registered during initialization of the TCP connection using the raw API. Table 1 lists the events and associated callback functions.

Table 1: **TCP Events, Callbacks, and How to Register the Callbacks**

Event	Callback	Register with lwIP
TCP Connection Established	* accept()	tcp_accept()
TCP Data Acknowledged by Remote Host	* sent()	tcp_sent()
TCP Data Received	* recv()	tcp_recv

void * tcp_init()

This function must be called first to initialize the lwIP TCP stack.

void tcp_tmr()

The void *tcp_tmr* function must be called every TCP_TMR_INTERVAL milliseconds. Two lower level timer functions are called inside void *tcp_tmr*. These functions are called directly by the example application: *tcp_slowtmr* and *tcp_fasttmr*.

struct tcp_pcb * tcp_new()

A new TCP protocol control block (*pcb*) structure is created by the *tcp_new* function.

void tcp_arg (struct tcp_pcb * *pcb*, void * *arg*)

An argument registered by *tcp_arg* is passed back to the application for all callback functions. This argument, *arg*, is usually used as a pointer to a structure holding the application state information.

err_t tcp_bind (struct tcp_pcb * *pcb*, struct ip_addr * *ip_addr*, u16_t *port*)

Bind the *pcb* to an IP address, *ip_addr* and TCP port number, *port* with the *tcp_bind* function.

struct tcp_pcb * tcp_listen (struct * *pcb*)

The *tcp_listen* function instructs the lwIP stack to put the *pcb* in the listen state. The *pcb* will start to listen for connections on the IP address and port number specified in *tcp_bind*. When a new connection is established, lwIP calls the callback application function specified using the following *tcp_accept* function.

void tcp_accept (struct tcp_pcb * *pcb*,
err_t (* accept)(void * *arg*, struct tcp_pcb * *newpcb* , err_t *err*))

The *tcp_accept* function allows the application to register a callback function, *accept*. The lwIP stack calls the *accept* function when a new connection is established on a *pcb* in the listening state.

err_t tcp_write (struct tcp_pcb * *pcb*, void * *dataptr*, u16_t *len*, u8_t *copy*)

The *tcp_write* function will write *len* bytes of data from *dataptr* to the transmit queue. The copy argument specifies whether or not the stack should copy the data or reference it using the pointer *dataptr*.

**void tcp_sent (struct tcp_pcb * *pcb*,
err_t (* sent)(void * *arg*, struct tcp_pcb * *tpcb*, u16_t *len*))**

The *tcp_sent* function registers the *sent* callback function. The *sent* function is called by the stack when data is acknowledged by the remote host. The *len* argument indicates the number of bytes acknowledged. Once acknowledged, the bytes are removed from the transmit buffer creating more space.

This is useful when trying to send a large amount of data using lwIP. If the *tcp_write* function fails because the transmit buffer is exhausted, this callback function allows the application to write additional data once buffer space becomes available.

For example, an application registers a function, *application_sent*, by calling *tcp_sent* and passing a pointer to it. The application then tries to send a large amount of data by calling *tcp_write*; however, *tcp_write* returns an error code if the transmit buffer is full. The *application_sent* function can continue to call *tcp_write* as transmit buffer space becomes available. The *len* argument passed to *application_sent* is the number of bytes acknowledged by the remote host. This *len* field is used as the number of bytes sent when calling *tcp_write*.

**void tcp_recv (struct tcp_pcb * *pcb*,
err_t (* recv)(void * *arg*, struct tcp_pcb * *tpcb*, struct pbuf * *p*, err_t *err*))**

The *tcp_recv* function is used to register the callback function *recv*. The *recv* function is called when new data arrives on the connection. The *p* argument is NULL when the connection is closed.

Example Software Application

The accompanying reference design example is a TCP based echo server. The server allows remote access to the Memec Design's Virtex-II Pro development board. The software starts a TCP server, listens for connections, and then echoes back the data given by the remote host. A Telnet client application is used for demonstration purposes. Telnet client connects to the development board through TCP/IP and communicates with the server running on the board. Figure 3 illustrates the basic connections. The board running the example application and a PC running the Telnet client are both connected to an Ethernet network. After board power up, it is configured for the network (MAC address, subnet mask, etc.) by connecting a PC to the serial port of the development board (serial connection not shown, use the serial connector on the main board - JD1).

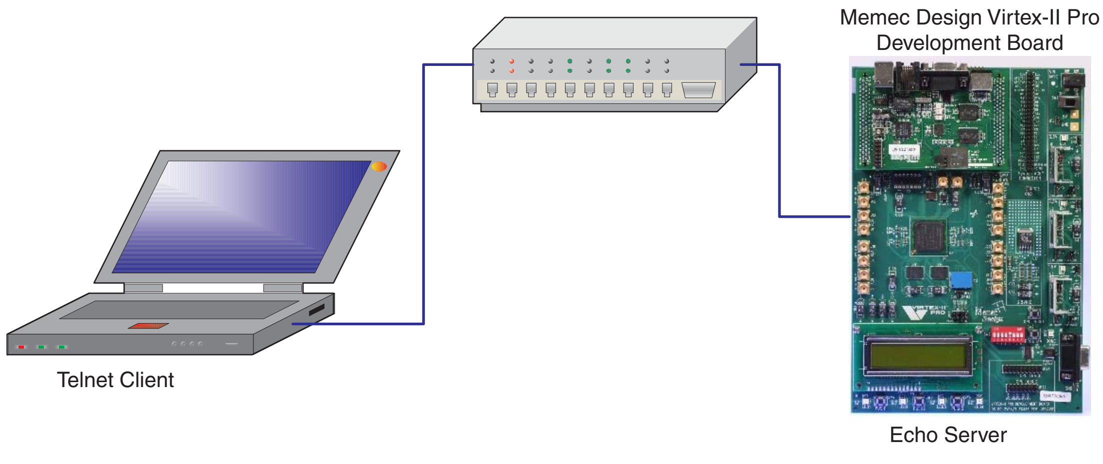


Figure 4: **Telnet Client and lwIP Echo Server**

The server echoes back the data sent by the client on the telnet terminal. This uses the standard TCP protocol .

Reference Design

Obtaining and Using the Necessary Files/Libraries

lwIP Files

The lwIP source files are integrated into the EDK as a library. The lwIP source tree has two components: the core lwIP source and the contrib source. The core lwIP source is hardware independent and contains all of the core functionality of the stack. The contrib source contains all of the ports for different hardware platforms. The Virtex-II Pro platform is used for the EDK reference design.

Currently, the core lwIP source is lwip-0.6.4; therefore, this is the tested version. The latest contrib source is in the lwIP CVS repository. Use the following commands in a Solaris or Xygwin shell to download the contrib source into the current directory:

```
$ cvs -z3 -d:pserver:anoncvs@subversions.gnu.org:/cvsroot/lwip co
  -r STABLE-0_6_4 contrib
```

More information about the anonymous lwIP CVS server is available at the following site.

http://savannah.nongnu.org/cvs/?group=lwip

Download the /lwip and /contrib directories into the same directory tree. Figure 5 shows the directory structure of the lwIP source installation in EDK.

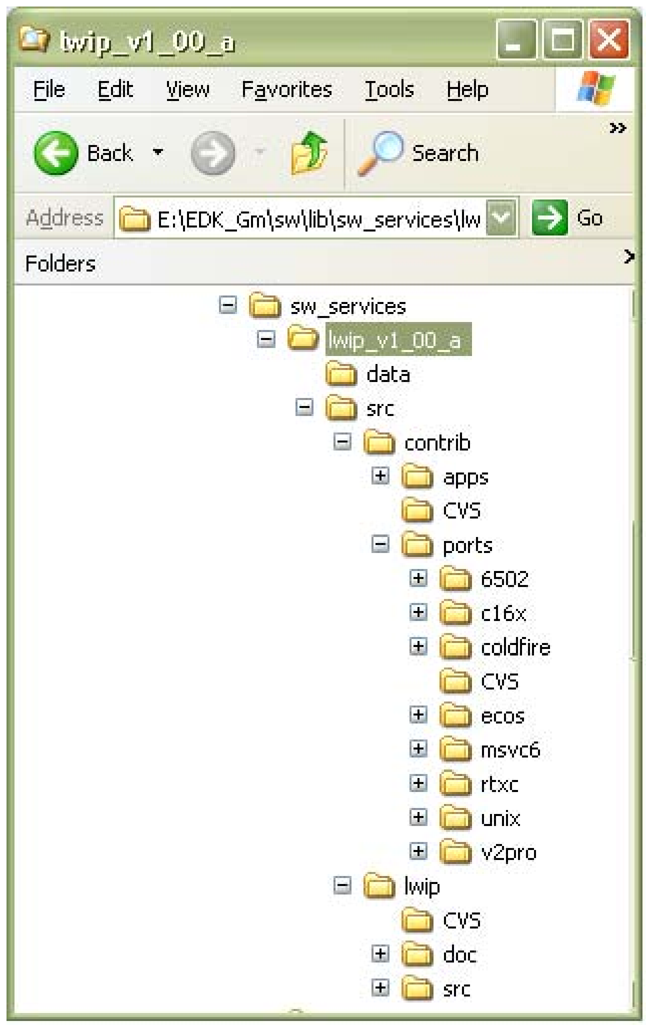


Figure 5: **lwIP Directory Structure**

Design Files

The reference design files are in the form of a Xilinx EDK project. An EDK project contains both the hardware and software descriptions for an embedded Virtex-II Pro design. The EDK project can be downloaded from the Xilinx site at http://www.xilinx.com/bvdocs/appnotes/xapp663.zip.

EDK Project Directory Structure

The directory structure for this EDK project is shown in Figure 6. The software application source code is located in the code directory. The software and hardware systems are specified in the `system.mss` and `system.mhs` files, respectively.

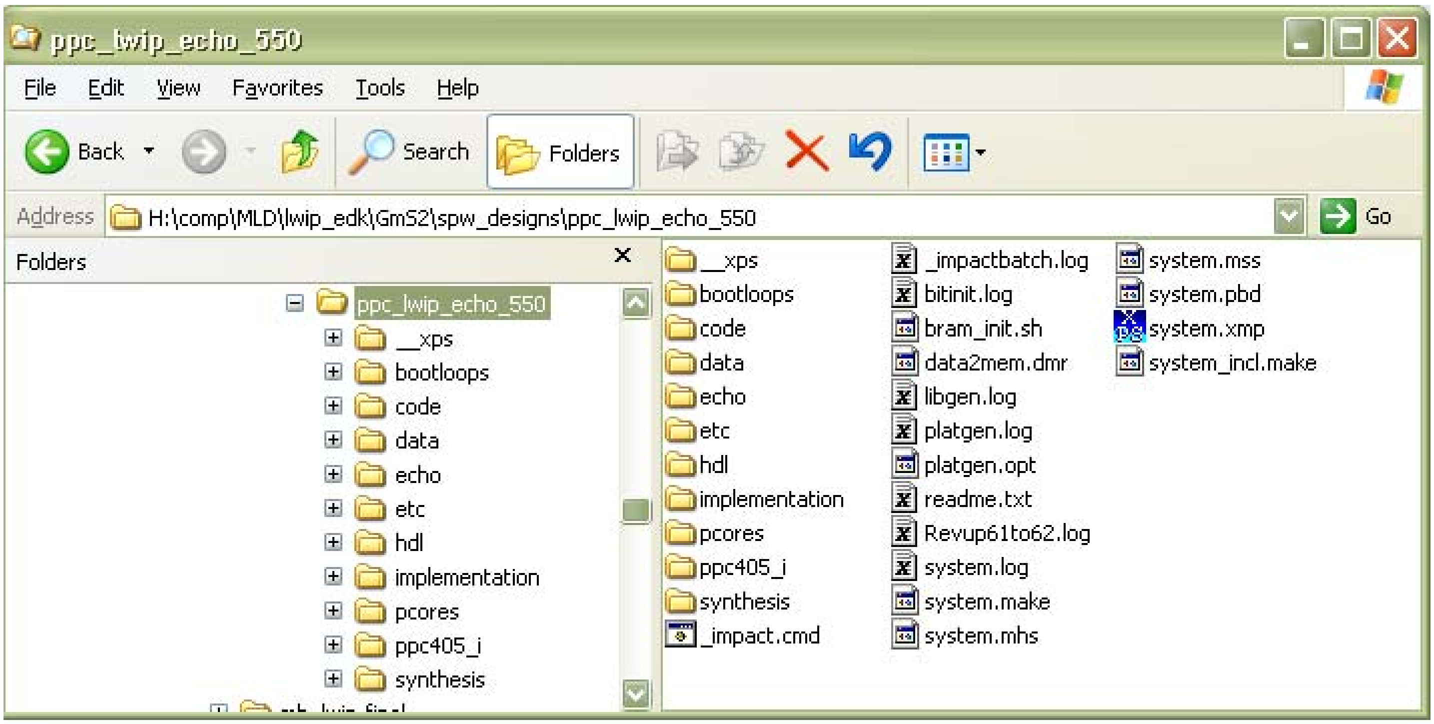


Figure 6: **EDK Project Directory Structure**

Executing EDK GUI Instructions

The following instructions are valid on Solaris or Windows versions of the Xilinx Platform Studio (XPS) GUI.

1. Unzip the EDK project files to your local file system.

   ```
   $ cd /<download_dir>/
   $ unzip xapp663.zip
   ```

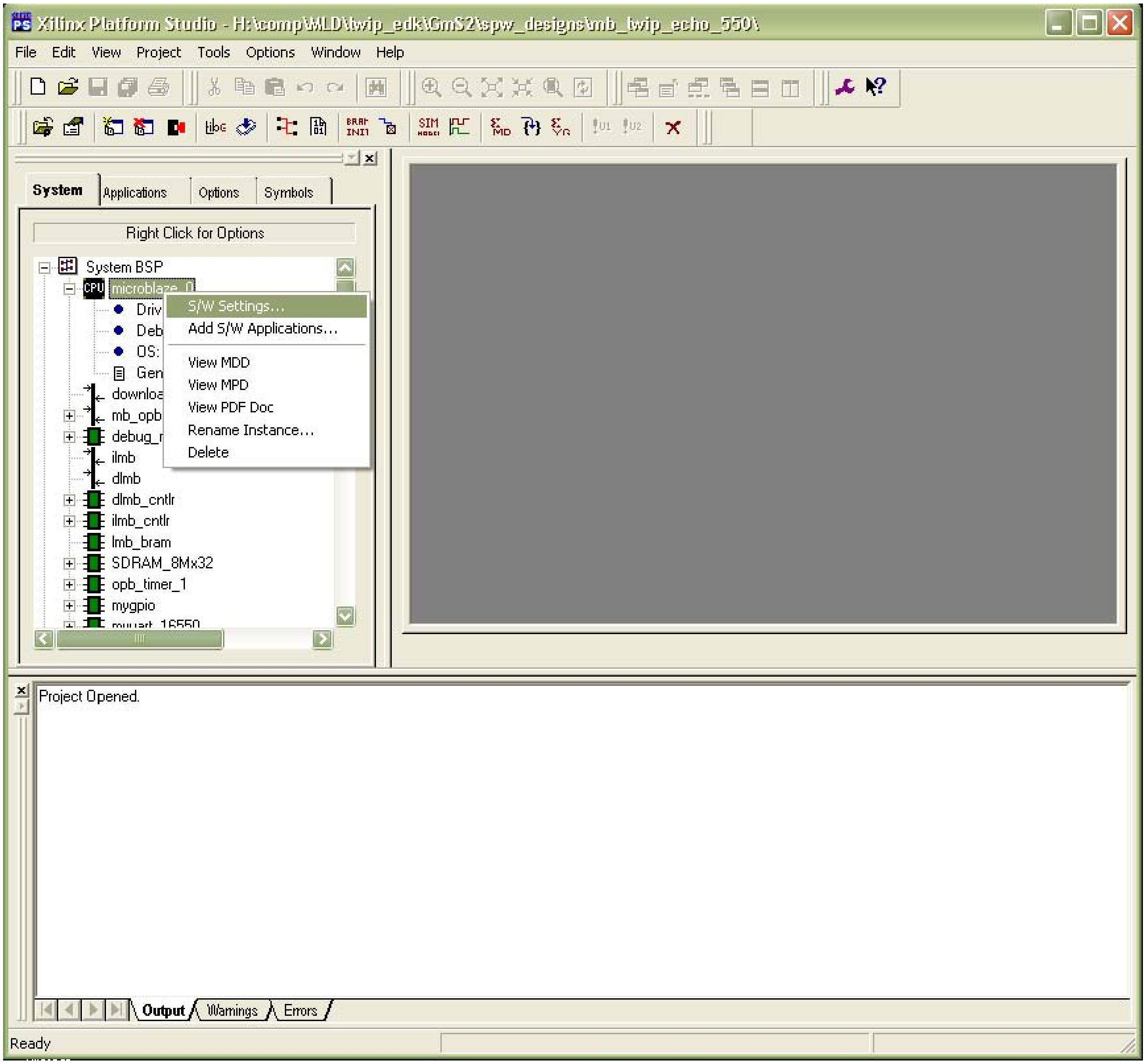
   The unzipped file has two directories: one for MicroBlaze and one for PowerPC, as shown below:

   ```
   xapp663/PowerPC
   xapp663/MicroBlaze
   ```

   Based on the processor used in the design, **`cd`** to the appropriate directory.

2. Update the EDK project for lwIP specific information.

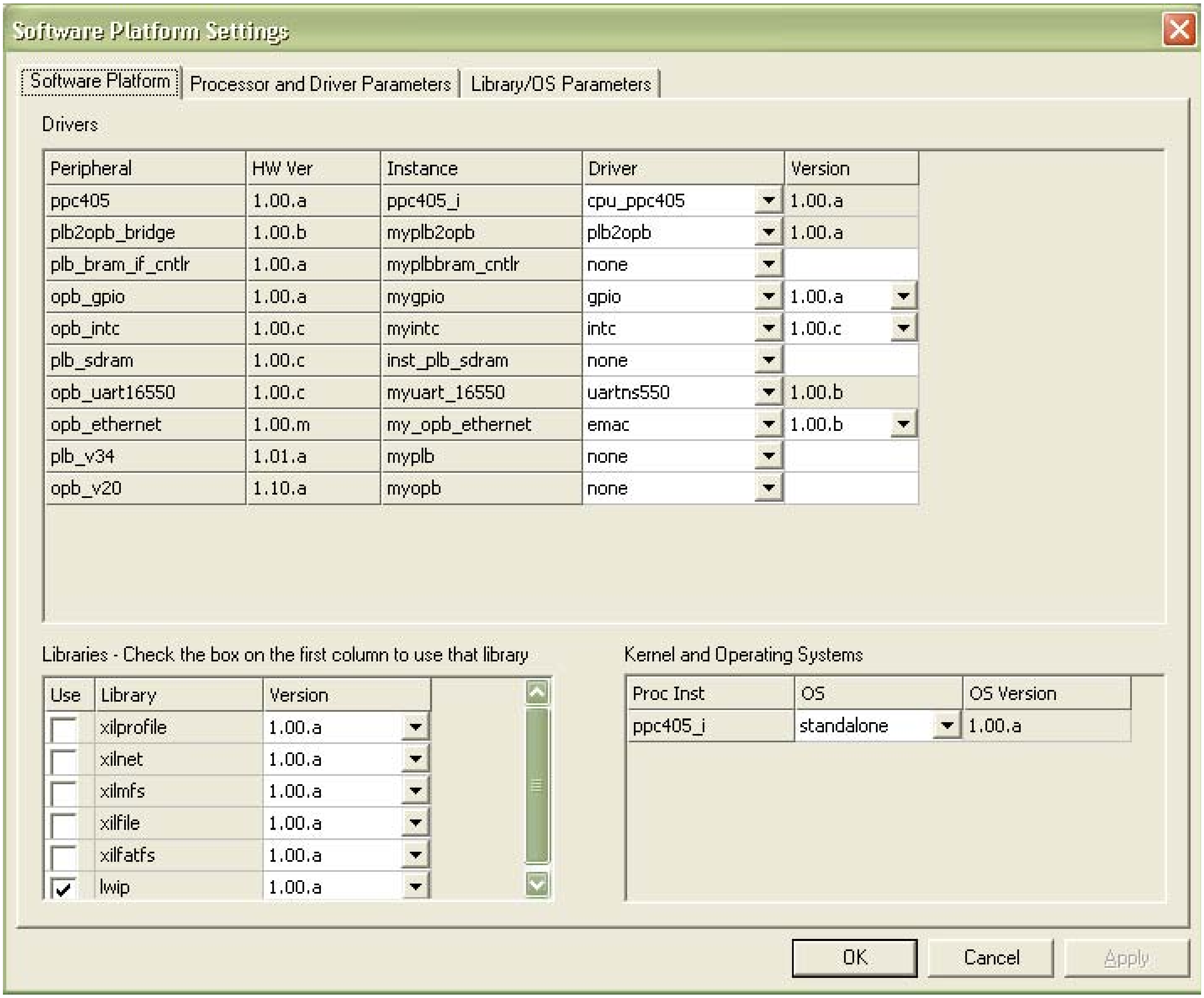
   The software specifications are updated by configuring the software components of the system. From the tree view of the EDK project, right click on any peripheral to see the menu shown in Figure 7.



x663_23_072004

Figure 7: **Specifying Software Specifications in XPS**

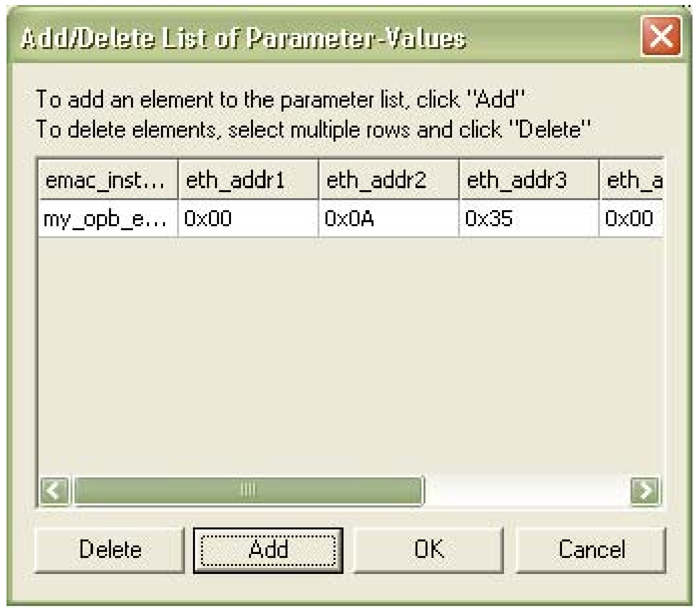
3. Select the *S/W Settings* option to open up the software settings dialog shown in Figure 8.



x663_24_072004

Figure 8: **Software Settings Dialog**

4. Select the *Library/OS Parameters* tab and look for lwip in the list. In the lwIP library configuration dialog, select the ... button that is present in the *emac_instances* configuration to display the *emac_instances* dialog shown in Figure 9.



x663_25_072004

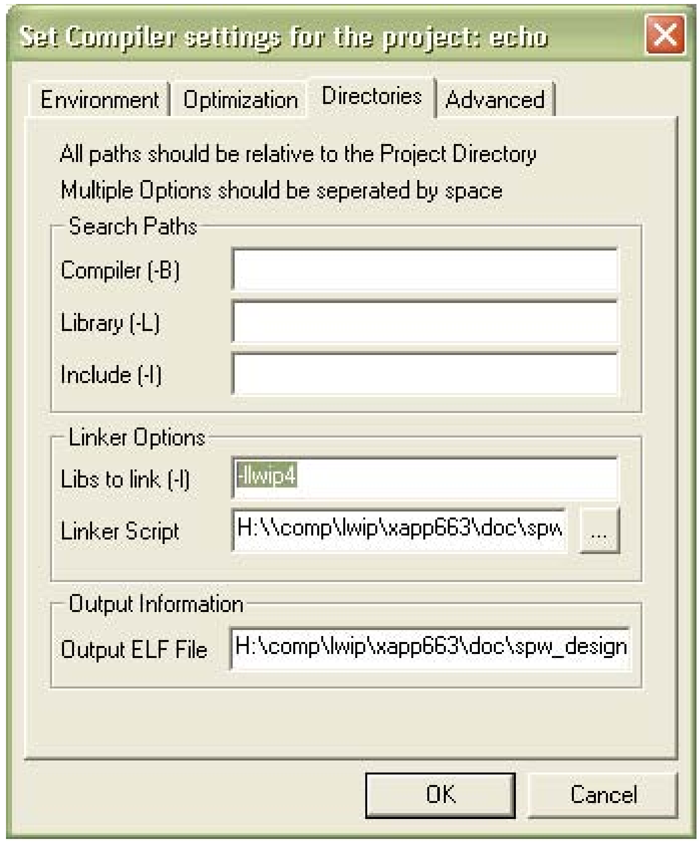
Figure 9: **Specifiying Ethernet Instance Information for lwIP**

5. Edit the MAC address in the GUI or by editing the lwIP library definition block in `system.mss`, as follows:

```
###LWIP LIBRARY####
BEGIN LIBRARY
  PARAMETER LIBRARY_NAME = lwip
  PARAMETER LIBRARY_VER = 1.00.a
  PARAMETER emac_instances
  =(my_opb_ethernet,0x00,0x0A,0x35,0x00,0x22,0x21)
END
```

6. Include the lwIP library (`liblwip4.a`) in the linker search path.

   The EDK project includes the lwIP library as an option to the compiler/linker (gcc). This is done as part of the Compiler Options to the Software Application *echo,* which is part of the EDK project. The Compiler Option dialog for *echo* is shown in Figure 10.



x663_26_072004

Figure 10: **Compiler Option Dialog for Echo Server Project**

Adding lwIP as an EDK User Library

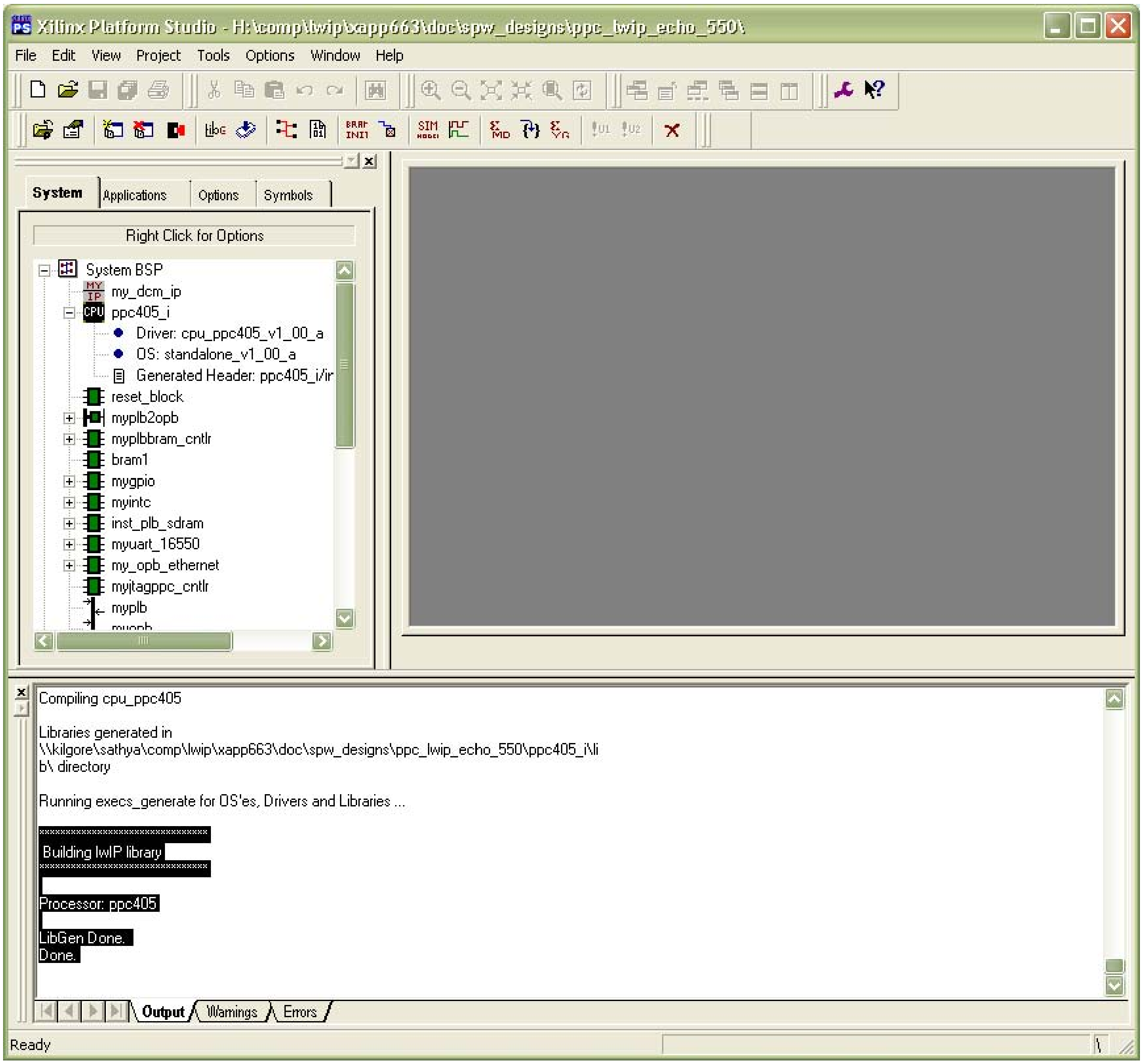
As of version 6.2.2, EDK supports lwIP as a library. lwIP can be compiled automatically during the standard build process of EDK. The example project already has lwIP as a user library. To add lwIP to a brand new EDK project, do the following:

1. Open the *S/W Settings* dialog (see the previous section).
2. Select the *lwip 1.00.a* checkbox from the Libraries section.
3. Add the *emac_instances* parameter for lwIP in the *Library/OS Parameters* tab of *S/W Settings* dialog (see the previous section).
4. Include `liblwip4.a` in the linker path (see the previous section).

Compiling lwIP as an EDK User Library

The lwIP library is compiled automatically when `libgen` is executed. There are several ways to run `libgen`. The most common way is through the XPS GUI (Tools -> Generate Libraries). Figure 11 illustrates the `libgen` run in XPS.

After these steps are completed, a `liblwip4.a` library will be located in the `ppc405_i/lib/` directory of the EDK project.

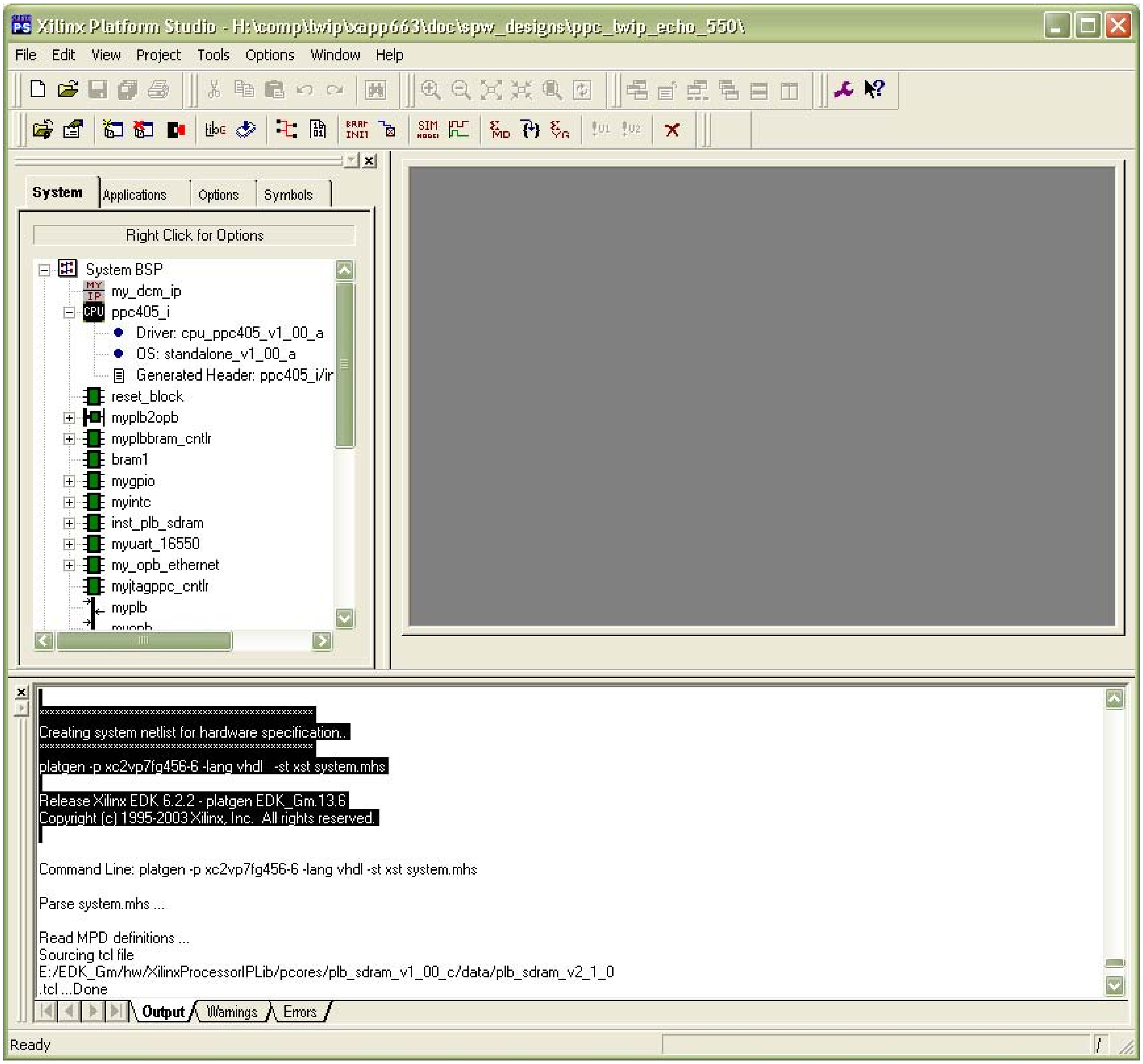


x663_27_072004

Figure 11: **Generating Software Libraries in XPS**

Implementing the EDK Hardware Flow

The flow for building the hardware netlist and bitstream is the same as for any EDK project. The most common way is through XPS GUI (Tools-> Generate Bitstream), as shown in Figure 12. Refer to the EDK documentation for more detail.



x663_28_072004

Figure 12: **Generating Hardware Bitstream in XPS**

Implementing the EDK Software Flow

The steps for building the software libraries are repeated here for completeness. To compile the software application, choose Tools-> Generate Libraries from XPS GUI as given in Figure 11.

Downloading the FPGA Bitstream

This section covers downloading the FPGA bitstream to the Memec Design Virtex-II Pro demonstration board. The design was tested on the XC2VP7, revision 4 board.

Downloading the design to the Memec Design Virtex-II Pro demonstration board requires a Xilinx parallel cable (both version III and IV of the cable are appropriate). A straight through RS-232 serial cable is needed to configure and use the design with a PC.

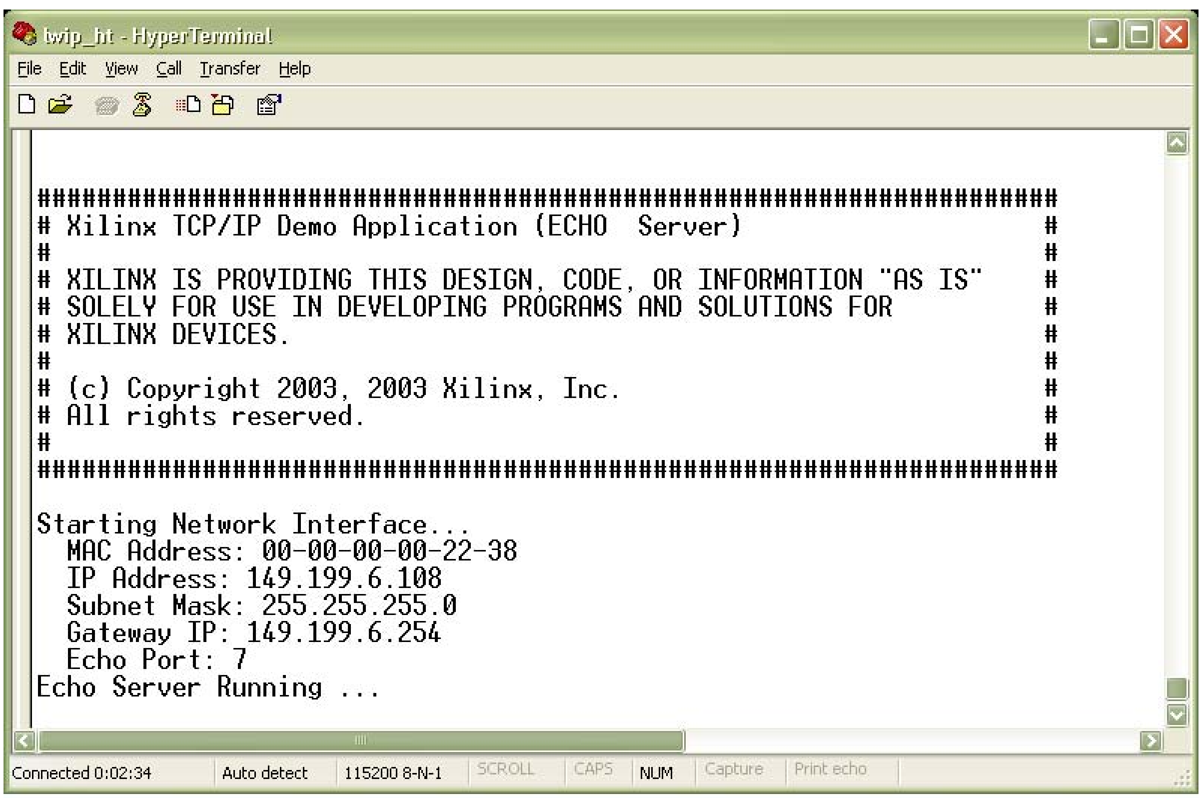
The footprints of the lwIP library and application are too large to fit in the internal block RAM of an XC2VP7. Download the software into the external SDRAM as a separate step from downloading the bit file (bitstream) into the FPGA.

To download both the hardware and software to the board:

1. Use Impact to download the `implementation/system.bit` bit file to the FPGA.
2. Use XMD to download and run the software application in the SDRAM. From the XPS GUI, choose Tools-> XMD. This displays an XMD shell, as shown below. From an EDK Xygwin shell (PC only), run the following commands:

```
Xilinx Microprocessor Debug (XMD) Engine
Xilinx EDK 6.2.2 Build EDK_Gm.13.6
Copyright (c) 1995-2004 Xilinx, Inc.  All rights reserved.
XMD% ppcconnect
XMD% dow echo/executable.elf
XMD% con
```

A menu (illustrated in Figure 13) is generated over the serial connection to the PC (use the 115200,8-N-1 serial port setting). Use the menu to configure the network addresses, and start the Echo application server. The network IP address and subnet mask must be set to legal network configuration values. Your network administrator can supply further details.

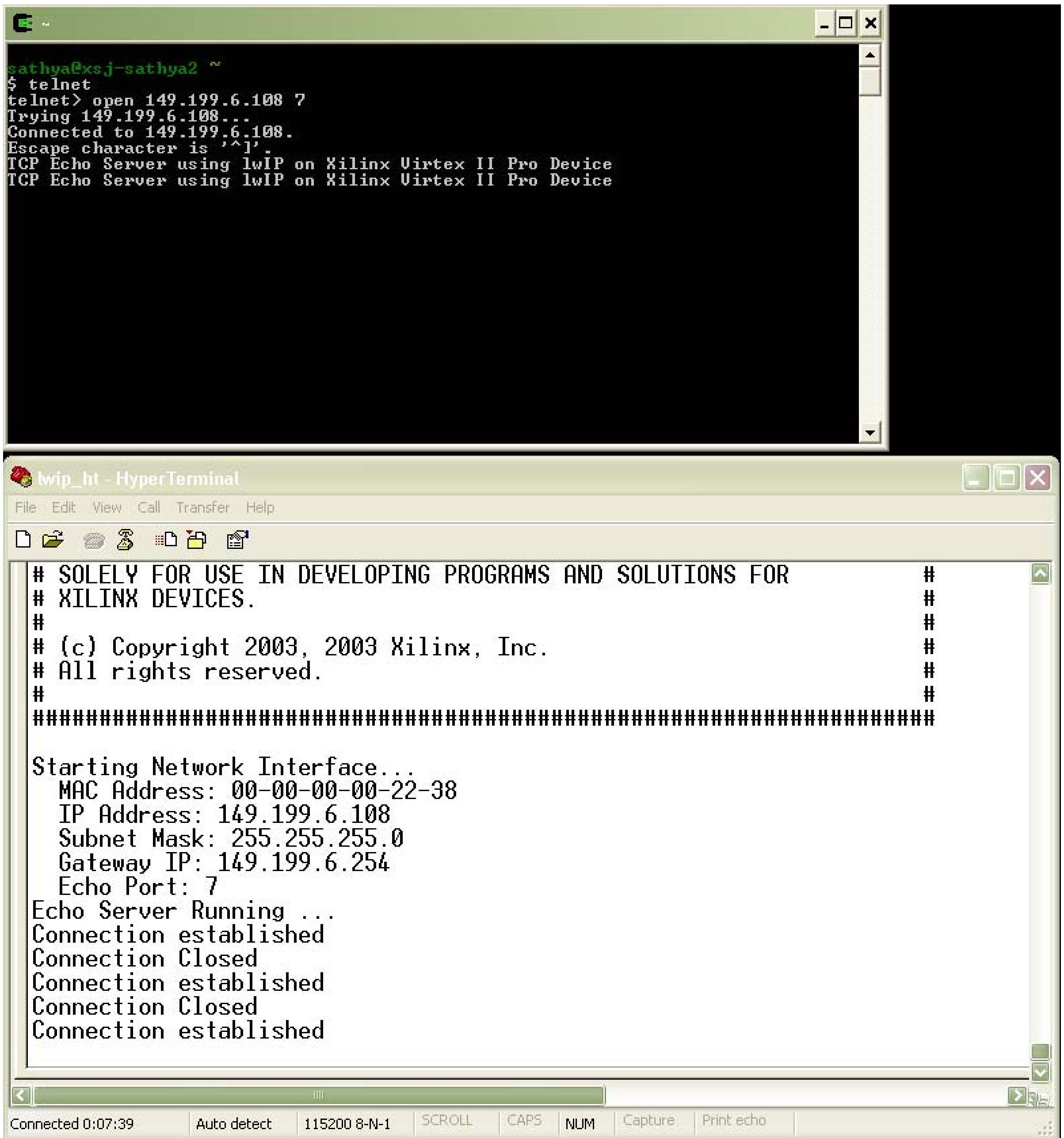


x663_29_072004

Figure 13: **HyperTerminal**

Running Telnet as an Echo Client on a Windows or Solaris Host

Telnet application (typically available in both Windows as well as Solaris platforms) can be used as an echo client to connect to the echo server running on the board. Once the application is running, connect to the board by entering its IP address and port number of the echo server - 7 by default. The application connects to the echo server running on the Virtex-II Pro device. Figure 14 shows the transactions between the telnet client and the echo server running on the Virtex-II Pro device.



x663_30_072004

Figure 14: **Telnet Client and Echo Server Transactions**

Conclusion

This application note illustrates how to use lwIP in an EDK Virtex-II Pro Design. The software application server, TCP based Echo, is an example of remotely exchanging data using the lwIP TCP/IP stack. The design is used when data movement to or from a Virtex-II Pro device is required over a reliable network (TCP/IP over Ethernet).

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
08/05/03	1.0	Initial Xilinx release.
08/23/04	1.1	Updates for the MicroBlaze reference design and using lwIP with the EDK 6.2.2 release.
08/30/04	1.1.1	Cosmetic changes.