EE 552
Final Report

Networked Appliance Controller

Web Version

Please see *http://www.ee.ualberta.ca/~cdjones/EE_552/* for the complete final report. In order to meet the 200 kB size restriction on this report, the sections on datasheets, schematics, design hierarchy, testing, and testbenches have been removed in their entirety.

8 April 2002

Group Email List *ee552@novusordo.net*

Justin Bague *jebague@telusplanet.net*
Stephen Dytiuk *dytiuks@hotmail.com*
Chris Jones *cdjones@ee.ualberta.ca*
Randy Tsen *rtsen@ualberta.ca*
Mimi Yiu *myiu@ualberta.ca*
Jones Yu *jones_yu@hotmail.com*

© 2002: Justin Bague, Steven Dytiuk, Chris Jones, Randy Tsen, Mimi Yiu, & Jones Yu.
All rights are reserved.
Permission is granted to freely redistribute this document or portions thereof, if credit is provided.
Submitted for credit in EE 552 in the winter 2002 term, at the University of Alberta.

Declaration of Original Content

The design elements of this project and report are entirely the original work of the authors and have not been submitted for credit in any other course except as follows:

> The following VHDL files have been reused from **[O2]**: *clkdiv.vhd, RS232_In.vhd, RS232_out.vhd, serialShiftRight.vhd, myShiftOut.vhd,* and *leddecoder.vhd*.
>
> The *clockDivider.vhd* file was reused from that provided in class.
>
> The LFSR_GENERIC component used in the SLIPCONTROL and NETCON test benches was taken from [O3].

8 April 2002

Justin Bague _____ Steven Dytiuk _____

Chris Jones _____ Randy Tsen _____

Mimi Yiu _____ Jones Yu _____

Abstract

The Networked Appliance Controller was designed to respond to remote requests from a local area network (LAN) and control an appliance: in this case, a relay with which to reboot a computer. The main focus of the project was the implementation of Protocol: Ethernet, IP, and UDP and the interface with the EE-100 Ethernet Module. The EE-100 provides a 10BaseT connection using a Crystal/Cirrus CS8900A Ethernet Controller, operating in 8-bit mode. A SLIP interface was also developed in parallel with the Ethernet interface as a backup scheme.

The design used two interfaces to the network inputs (Ethernet and SLIP), with an interface-agnostic engine reading the payload of the incoming UDP packets to determine whether they carried the appropriate value. In this case, it would activate the relay; otherwise, it would ignore the packet and continue waiting.

While our design simulated correctly, we were unable to completely finish interfacing the Ethernet and SLIP modules in our FPGA to their respective hardware devices, and thus have not been able to demonstrate operation of the device as a whole. However, the SLIP and Ethernet modules should be very close to being usable by a future project for their own devices.

Achievements

Network Controller

The network controller (*netController*) implements an engine that examines the payload of a *User Datagram Protocol* (UDP) packet encapsulated in an *Internet Protocol* (IP) packet [**S4**, **S5**] for a pre-defined "secret key" which provides a weak level of authorization for the appliance's control. It interfaces to a packet buffer which permits random access by the network controller so that it may examine the headers and contents of the packets without having to interface directly to the Ethernet or SLIP controllers.

The network controller implements a *highly* stripped-down version of UDP and IP, ignoring all headers. It is unsuitable for use in anything other than an environment where packets should be accepted from any source for any destination (i.e. *promiscuous mode*).

Ethernet Interface

The Ethernet interface (*ethIF* and *pktBuffer*) implements an abstract interface to an Ethernet controller chip. *ethIF* polls the chip to determine its status, and whether it has received new packets (the CS8900A chip which we used has 4 kBytes of onboard RAM to store packets for asynchronous retrieval by a host system). Upon receipt, *ethIF* stores the packet in an on-FPGA memory (the *pktBuffer*) for random-access use by the network controller.

We have demonstrated communication between *ethIF* and the CS8900A chip, but it has only been partially successful. We are able to retrieve chip revision information from an internal register on the CS8900A, but have not been able to successfully retrieve packets from the chip. In spite of this difficulty, the *ethIF* and *pktBuffer* modules implement a coherent interface to hardware: a future group should be able to use them as a basis for starting implementation of an Ethernet interface and resolve the problems which remain.

SLIP Interface

The SLIP interface (*slipcontrol*) module correctly receives serial data using the *Serial Line Internet Protocol* (SLIP) over an asynchronous serial line, at a rate of 9600 baud. The data is stored in the FPGA RAM, and is accessible by sending an address to it. When the end character is received, it allows read access to the RAM. This behaviour has been verified through simulation and by programming the FPGA and actually sending it data. The *slipcontrol* module is compatible with the *ethIF* module, and they can be substituted with a simple recompilation of the overall system.

The *slipcontrol* module is also setup to send data using SLIP over the serial line. The controller has been simulated: it correctly formats and sends the data. This was out of the scope of our project, and so has not been tested.

This module should be usable for future groups to use in their projects when IP connectivity is required and a full-bore Ethernet connection is overkill.

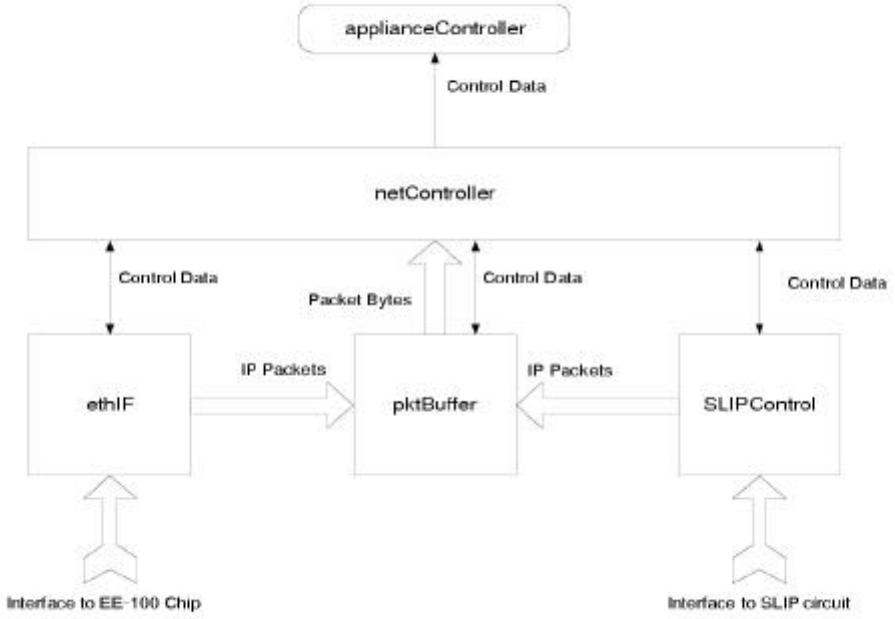
Appliance Controller

The original design called for the use of the FPGA as a supplier of 3.3 volts to power the relay. However, upon hooking up the appliance, it was found that a single pin could not supply enough current to activate the diode in the optical coupler. It was then decided that the pins would be used as a current sink instead of a source. The optical coupler power line was attached to V_{cc} on the FPGA, while the ground of the coupler was attached to our output pin. When the appliance is to be activated, a logic zero is applied to the pin, thus creating a ground. When it is to be shut off, a logic '1' is placed on the pin, resulting in a zero voltage drop across the diode, and therefore disabling it. This was tested, and worked perfectly.

Description of Operation

Network Controller

The network controller is the "brains" of the networked device: it is responsible for parsing inbound UDP packets and determining whether they carry the required secret payload which authorizes the user to toggle the state of the appliance. It communicates to a packet buffer (*pktBuffer* in the case of Ethernet, and to a similar buffer encapsulated within *slipcontrol* in the case of SLIP) to access bytes within a packet with random access, in order to be able to determine whether the packet is appropriate.

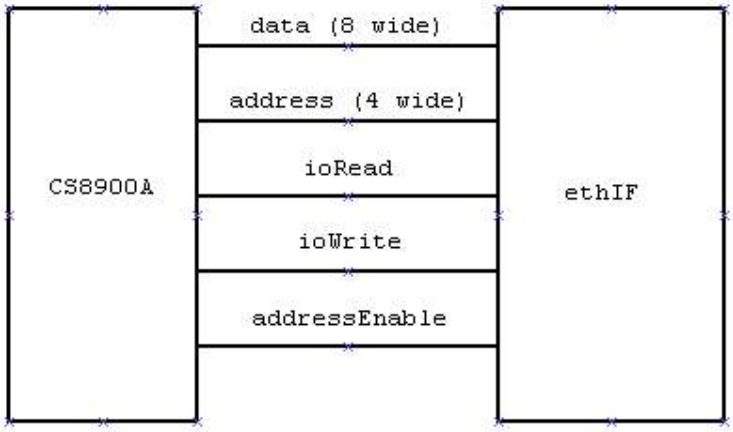


Ethernet Interface

The network controller communicates with a network through the CS8900A chip on the EE-100 board: the VHDL module used to do this is *ethIF.vhd*. An Ethernet (IEEE 802.3 [**S10**]) packet, encapsulating an IP packet [**S4**, **S7**] which in turn encapsulates a UDP packet [S65], is received by the CS8900A and automatically stored in its onboard memory (capacity 4 kB). The *netController* polls *ethIF* when it is idle, enquiring as to whether a new packet has been received. If so, *netController* asks *ethIF* to load the newly received packet into a packet buffer (*pktBuffer*), subsequently sending an acknowledgement to the *netController*.

The Ethernet interface operates at a substantially higher clock frequency (presently 1024 times faster) than the network controller, so that the new packet can be loaded into the packet buffer in one *netController* clock cycle. It interfaces directly to the CS8900A chip as illustrated below:

Upon receiving a request from the network controller to check for the receipt of new packets in the CS8900A, the interface communicates with the chip to determine whether a new packet has been received. If this is the case, the Ethernet interface acknowledges the controller's request and advises it that a new packet has been received.



ethIF / CS8900A Interface

The network controller then may request that the Ethernet interface load the new packet into the packet buffer, destroying the current contents of the buffer. Upon receiving this request and obtaining the communication lock described above, the interface communicates with the Ethernet chip to determine the length of the packet, so that it can copy the appropriate number of bytes into the packet buffer. See the CS8900A 8-Bit Operation Reference Manual [**O1**] and the CS8900A Product Data Sheet [**D2**] for the precise details of operation.

Once the packet has been loaded into the buffer, the Ethernet interface acknowledges the network controller's request. The network controller may then communicate with the packet buffer in random-access mode to determine the contents of each byte. As the layout of the Ethernet, IP, and UDP packets are known, the network controller need not examine all bytes in the buffer, but may instead simply look for the packet lengths, and use them as offsets into the RAM to find the UDP payload.

SLIP Interface

The SLIP component takes in 7 inputs, and produces 8 outputs. The *soutclr* and *reset* ports are both connected to *slipcontrol*'s *reset* because they are two separate resets that we don't need to differentiate against in this project. The serial input (using the SLIP protocol) we receive from the serial port is *serialin*. The FPGA clock at 25.175 MHz is input as *sysclock*.

The serial output transmitted using the SLIP protocol based on the *serial_out_data* input we receive from EnetControl is *serialout*, while *outready* is a signal which indicates when we are ready to either receive or transmit data. *Out_load* must be pulsed high when a new word needs to be outputted from *serial_out_data*.

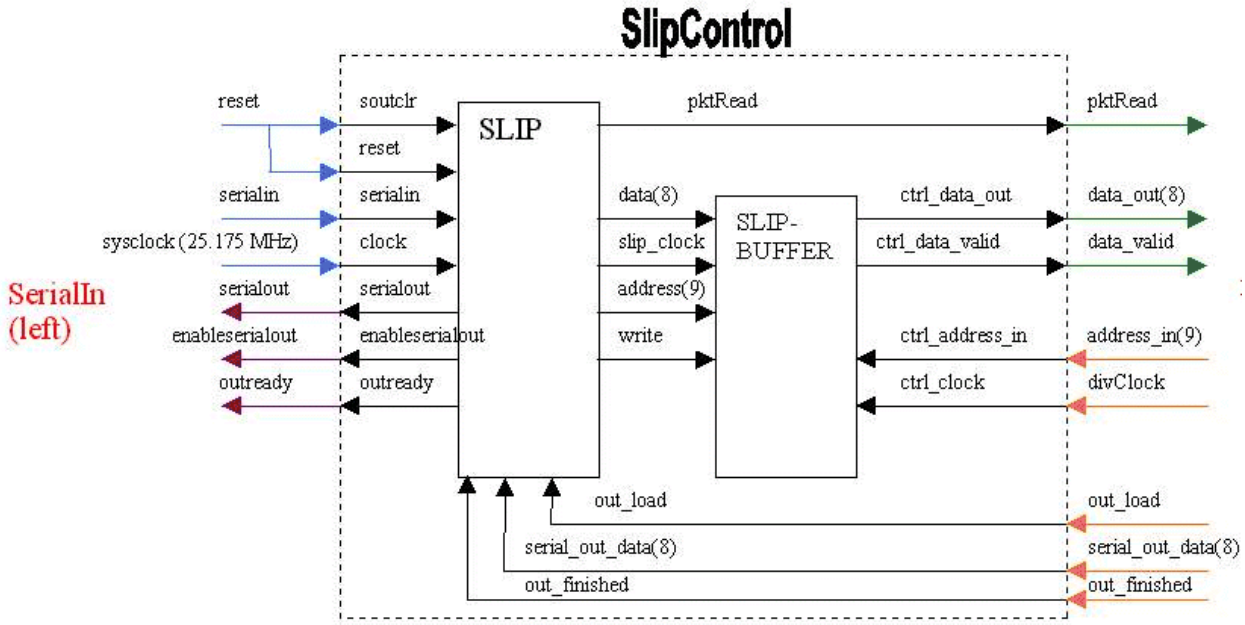


FIGURE 8: SLIP CONTROL

SLIP Control Design

The SLIP component uses the RS232 transmission files, as specified in [**O2**]. It converts the serial input to byte-sized words, keeps track of the address that the word should be stored in *SLIP_BUFFER*, and when the END character is received, pulses *pktRead* high. Each word is stored into *SLIP_BUFFER* as soon as it is completely received. To send, the SLIP component translates the *serial_out_data* into serial data conforming to the SLIP standard, and sends it out over the *serialout* line.

The *SLIPBUFFER* component uses onboard RAM to store and retrieve the data sent to us over the serial connection. Whenever the SLIP component has something to write to *SLIPBUFFER*, *write* is set high, and *SLIPBUFFER* will store the data into the address indicated by *ADDRESS*. Whenever *write* is low, *SLIPBUFFER* will send the data from the address specified by ctrl_address_in out to the *netController*.

The SLIP protocol has four special characters:

- END: %11000000 (0xC0, 192_{10})
- ESC: %11011011 (0xDB, 218_{10})
- ESC_END: %11011100 (0xDC, 219_{10})

- ESC_ESC: %11011101 (0xDD, 220_{10})

The END character is used to indicate the end of an arbitrary sized packet. The ESC character is used so that we can transmit the same data as the END character without conveying the END message. When an END character appears in the packet, it is translated to be a (ESC, ESC_END) pair; similarly, when an ESC character appears naturally in the packet, it is translated to (ESC, ESC_ESC) for transmission. [**S1**]

Appliance Controller

The appliance interface will require a single pin from the FPGA. Due to the nature of the appliance, and the high voltages and currents involved, we have chosen to replace the relay in our original design with an Optical Coupler. This new IC will allow complete voltage and current separation from the appliance side and the FPGA.

Whenever 3 volts (Logic 1) are applied to the Optical Coupler, it will allow a 12-volt current to flow through the other half of the IC. This acts as a relay. The datasheet for this device is included in the appendix. This 12-volt signal will then trigger another actual relay, which will complete the circuit within the extension cord. Intuitively, it can be seen that without power to our FPGA, there will be no activity in the appliance. The 12-volt signal will be generated with a simple battery. Most likely, two 9 volt batteries in series will allow for a longer current draw before replacement. Alternatively, a 12-volt AC-DC adapter could be used in place of the battery.

The maximum speed of this device is rated as 73 MHz, and thus will not have any detrimental impact on the operations of the FPGA. This VHDL could be completely left out, if the signal from the Ethernet side of the FPGA was declared as negative logic.

Netcon Datasheet

Netcon consists of the Altera UP-1 FPGA board, an EE-100 Embedded Ethernet module with a CS8900A Ethernet chip onboard, the appliance relay system, and the serial communication system. Offering a remote reboot capability for any network-connected computer, Netcon is the perfect solution for when your computer locks up while you're not at the desk.

Specifications

- 5V power supply
- 10BaseT and RS-232 connections

Pin Name	Pin Type	Pin Number	Purpose
address0	Output	45	Connection to EE-100 Module.
address1		49	
address2		48	
address3		51	
ioRead		50	
ioWrite		54	
aEnable		53	
data0	Bidirectional	56	
data1		55	
data2		62	
data3		61	
data4		64	
data5		63	
data6		66	
data7		65	
output		71	To relay subsystem

Design Size: Logic Blocks Used

Network Controller

```
Total dedicated input pins used:                6/6      (100%)
Total I/O pins used:                           36/183    ( 19%)
Total logic cells used:                       141/1152   ( 12%)
Total embedded cells used:                      0/48     (  0%)
Total EABs used:                                0/6      (  0%)
Average fan-in:                              3.56/4      ( 89%)
Total fan-in:                               502/4608     ( 10%)
```

Ethernet Interface

```
Total dedicated input pins used:                6/6      (100%)
Total I/O pins used:                           35/183    ( 19%)
Total logic cells used:                       274/1152   ( 23%)
Total embedded cells used:                      0/48     (  0%)
Total EABs used:                                0/6      (  0%)
Average fan-in:                              3.50/4      ( 87%)
Total fan-in:                                959/4608    ( 20%)
```

SLIP Interface

```
Total dedicated input pins used:           1/6      ( 16%)
Total I/O pins used                        1/183    (  0%)
Total logic cells used                     1/1152   (  0%)
Total embedded cells used                  0/48     (  0%)
Total EABs used                            0/6      (  0%)
Average fan-in                          1.00/4      ( 25%)
Total fan-in                               1/4608   (  0%)
```

Appliance Controller

```
Total dedicated input pins used:                6/6      (100%)
Total I/O pins used:                           30/183    ( 16%)
Total logic cells used:                       319/1152   ( 27%)
Total embedded cells used:                     16/48     ( 33%)
Total EABs used:                                2/6      ( 33%)
Average fan-in:                              3.23/4      ( 80%)
Total fan-in:                               1032/4608    ( 22%)
```

Results of Experiments

Appliance Controller

The original design called for a system of components arranged in the following manner.

FPGA = > Buffer = > OPAMP (If Necessary) = > Relay = > Computer

This was ultimately decided against, due to the unnecessary and prohibitive costs of the specific relay we were using. Also, since there was a requirement of a power supply to bring the opamp up to 12 volts anyway, it was decided that for simplicity's sake we would just use two relays, thus changing the design to the following.

FPGA = > Relay1 = > 12-volt supply = > Relay2 = > Computer

This design however, had one notable drawback. By attaching a standard electro mechanical relay directly to the FPGA, we risked backwards EMF interference / and or destruction of the FPGA. To get around this dilemma, it was chosen to use an optical coupler to isolate the FPGA from the appliance circuitry, resulting in the following.

FPGA = > Optical Coupler = > 12-volt supply = > 12-volt Relay = > Computer

As for the code to run these devices, a single output pin is all that is required. The VHDL code for a working interface to the appliance is included in the appendix, but may be summarized as:

If disable signal goes high, Reset_Computer_Relay signal goes low.

This VHDL code is simply an inverter, so that the appliance is always running, until the FPGA actively decides to shut it off.

References

Books

B1. Stevens, W. Richard: "*TCP/IP Illustrated, Volume 1: The Protocols*"
ISBN 0-201-63346-9, Addison-Wesley, 1994.

Diagrams & Data Sheets

D1. Cirrus Logic: "*CS8900A: 10Base-T Ethernet Controller*",
http://www.cirrus.com/design/products/overview/index.cfm?ProductID=46

D2. Cirrus Logic: "*CS8900A Product Data Sheet*",
http://www.cirrus.com/pubs/cs8900a-4.pdf

D3. Li, Timmy: **"***Schematic of TTL to RS232 Interface*"
http://www.ee.ualberta.ca/~elliott/ee552/studentAppNotes/2002_w/circuits/TTL_to_Serial/images/Drawing3.jpg

D4. MAXIM: "*+5V-Powered, Multichannel RS-232 Drivers/Receivers*",
http://pdfserv.maxim-ic.com/arpdf/MAX220-MAX249.pdf

Other Resources

O1. Ayres, Jim: "*Using the Crystal(R) CS8900A in 8-bit Mode*",
http://www.cirrus.com/pubs/an181.pdf

O2. Ng, Dave; Mel Lumague; Ben Talbot; Emy Egbogah; Nitin Parimi: "*RS-232 Serial Port*"
http://www.ee.ualberta.ca/~elliott/ee552/studentAppNotes/2002_w/vhdl/rs232/RS232.html

O3. EE 552: "*Lab 5: Pipelining, Handshaking & Test Benches*"
http://www.ee.ualberta.ca/~elliott/ee552/labs/lab5/lab5.html.
LFSR_GENERIC file created by Raymond Sung; modified by Raymond Sung and John Koob.

O4. Desrosiers, Gary T.: "*CS8900A Packet Sniffer*",
http://www.embeddedethernet.com/appnotes/sniffer.bs2

O5. Bensler, Tim & Eric Chan: "RS-232 Serial Port"
http://www.ee.ualberta.ca/~elliott/ee552/studentAppNotes/1999_w/RS232/

Standards & Specifications

S1. Romkey, J.: "*A Nonstandard For Transmission of IP Datagrams Over Serial Lines: SLIP*"
(RFC 1055, STD 47), The Internet Society, 1988.
http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc1055.html

S2. Socolofsky, Theodore J. & Claudia J. Kale: "*A TCP/IP Tutorial*"
(RFC 1180), The Internet Society, 1991.
http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc1180.html

S3. Internet Engineering Task Force: "*Requirements for Internet Hosts – Communication Layers*"
(RFC 1122, STD 3), The Internet Society, 1989.
http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc1122.html

S4. Deering, Steve: "*Internet Protocol*"
(RFC 791, STD 5), USC/Information Sciences Institute, 1981.
http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc791.html

S5. Postel, Jon (Ed.): "*User Datagram Protocol*"
(RFC 768, STD 6), USC/Information Sciences Institute, 1980.
http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc768.html

S6. Postel, Jon (Ed.): "*Transmission Control Protocol*"
(RFC 793, STD 7), USC/Information Sciences Institute, 1981.
http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc793.html

S7. Hornig, C: ""*Standard for the transmission of IP datagrams over Ethernet networks*"
(RFC 894, STD 41), USC/Information Sciences Institute, 1984.
http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc894.html

S8. Postel, Jon, & J.K. Reynolds: "*Standard for the transmission of IP datagrams over IEEE 802 networks*", (RFC 1042, STD 43), USC/Information Sciences Institute, 1988.
http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc1042.html

S9. IEEE: "*IEEE Standards for Local and Metropolitan Area Networks: Overview and Architecture*"
(IEEE Std. 802-1990), ISBN 1-55937-052-1,
Institute of Electrical & Electronics Engineers, New York City, 1990.

S10. IEEE: "*Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications*", (IEEE Std. 802.3, 2000 Ed.), ISBN 0-7381-2673-8 [PDF Vers.], Institute of Electrical & Electronics Engineers, Piscataway NJ, 2000.

S11. TechFest: "*Ethernet Technical Summary*"
http://www.techfest.com/networking/lan/ethernet2.htm#2.1

S12. Fairhurst, Gorry: "*IP Packet Header*":
http://www.erg.abdn.ac.uk/users/gorry/course/inet-pages/ip-packet.html

S13 The University of North Carolina at Chapel Hill:
http://www.cs.unc.edu/Courses/comp249-s02/lectures/comp249_s02_3/sld009.htm

Web Pages

W1. Elliott, Duncan: "*EE 552 Project*",
http://www.ee.ualberta.ca/~elliott/ee552/reports.html

W2. Radio Shack: "*Mount Electro-Mechanical Relay*",
http://www.radioshack.com/searchsku.asp?find=900–8653

W3. TechFest.com: "*TechFest Ethernet Technical Summary*",
http://www.techfest.com/networking/lan/ethernet.htm

W4 www.embeddedether.net: "*EE-100 Ethernet Module Technical Specifications*",
http://www.embeddedether.net/specs.html

W5. Thornton, Mitch: "*Lab4: Using LPMs in VHDL*",
http://www.ece.msstate.edu/~mitch/class/ee4743/labs/lab4/lab4.html

Design Verification

Ethernet Interface

The Ethernet interface was tested using the *nettester.vhd* file to attempt to debug the connection between the CS8900A and the FPGA. The design could not be verified in its entirety, but communication between the chips was established.

SLIP Interface

The *sliptester.vhd* file was created to verify the *slipcontrol* design, using the *LEDDecoder.vhd* file from [**O2**].

This testing program is useful to physically test the *slipcontrol*. It accepts serial data, stores it in the RAM, and displays the contents of the RAM onto the LED display. The pushbuttons are used to look at the next memory address, and go back to address 0. In addition, the two decimal points on the LED display the data_valid line, and the packet_received line.

The design was verified by sending characters from Hyperterminal on the Windows computer, and checking to see that the characters were received and stored properly on the FPGA. To test whether the END character would be received, we changed the definition of END to be ASCII 'e' so that I could send it the END char. Otherwise we would not have been able to use Hyperterminal to send the data. Also used was the *jSLIP* program listed later: it sends UDP packets over Ethernet or SLIP. This works correctly using the correct end character for the SLIP protocol.

The SlipControl functioned correctly. It received all the data, stored it, and retrieved it when requested.

Other verification of the smaller components was performed. Waveforms and explanations of those verifications are included.

Appliance Controller

The appliance controller was tested manually, using a push-button switch.