An 8051 Based Web Server

Project by Mason Kidd

Submitted to Dr. Donald Schertz

EE 452 Senior Laboratory II

May 14th, 2002

Table of Contents

Abstract

A web server was designed using an 8051 based micro controller. The micro controller was interfaced to an Ethernet network using an Ethernet Transceiver chip. This allows the user to connect to the micro controller using a standard web browser and receive data in the form of a web page.

Functional Description

The goal of this project is to design a web server based on an 8051 micro controller board. The micro controller board was interfaced to a 10BaseT Ethernet network. A TCP/IP protocol stack and web server software were written for the micro controller board. This allows the user to connect to the 8051 micro controller using a web browser and receive a web page with data from the micro controller. An Ethernet Transceiver chip provides the interface from the micro controller to the Ethernet network.

The main input and output of this project are the Ethernet line to the rest of the network. Packets of data are sent and received using this line. Data acquisition and external system control are done using the inputs and outputs available to the 8051 micro controller board, which include, but not limited to, Analog/Digital converters, Digital/Analog converters, digital inputs and outputs, a keypad, and an LCD display. In the future, any of these may be utilized. For the testing purposes of this project a simple temperature sensor using the A/D converter is used.

When a user attempts to connect to the micro controller, the 10BaseT Transformer receives line signals from the Ethernet network. The Ethernet Transceiver then converts these line signals to packets, and stores them in internal buffers. The micro controller reads these packets from the Ethernet Transceiver for decoding. The micro controller decodes these packets, processes them, and sends a response. The response goes back the same way, through the Ethernet Transceiver and the 10BaseT Transformer, to the Ethernet Network and finally to the user.

In order to operate as a web server, the software on the micro controller must implement a subset of the TCP/IP (Transmission Control Protocol/Internet Protocol) protocol stack (see Appendix 2). TCP/IP is a protocol standard that transmits data using packets of information. TCP/IP is based on the OSI (Open Systems Interconnection) model for networks. The protocols that are implemented are shown in Table 1, along with their corresponding OSI layers. The ICMP (Internet Control Message Protocol) protocol is implemented in order to test connectivity to the micro controller board. IP and ARP (Address Resolution Protocol) are implemented because TCP and UDP (User Datagram Protocol) rely on them. UDP is implemented to provide a simple server so that a client can receive data. This was done because UDP is much easier to implement than TCP, so the micro controller's connectivity and data processing could be tested before the web server is implemented. TCP is the protocol that a web browser uses to send and receive data with a web server. Each protocol has it's own header that must be processed. The Ethernet Transceiver provides the OSI Layers 1 (Physical) and 2 (Data link). The Layer 3 packet is sent to the Ethernet Transceiver, where the Layer 2 header and trailer are added, and then the frame is sent to the network.

Table 1 – Protocols Implemented by Software

Protocol/Standard	**OSI Layer**	**Function**
Internet Protocol (IP)	3 (Network)	Decide how to get data to its destination
Address Resolution Protocol (ARP)	3	Determine host's MAC address using its IP address
Internet Control Message Protocol (ICMP)	3	Provide network reliability information
User Datagram Protocol (UDP)	4(Transport)	Provides message exchange using datagrams
Transmission Control Protocol (TCP)	4	Provide end-to-end reliable communication
Hypertext Markup Language (HTML)	6 (Presentation)	Format data for display in a web browser
Web Server	7 (Application)	Process requests for HTML data

Block Diagram

The complete system level block diagram is shown in Figure 1. Table 2 lists the various subsystems and their inputs and outputs. Each subsystems function is described below.

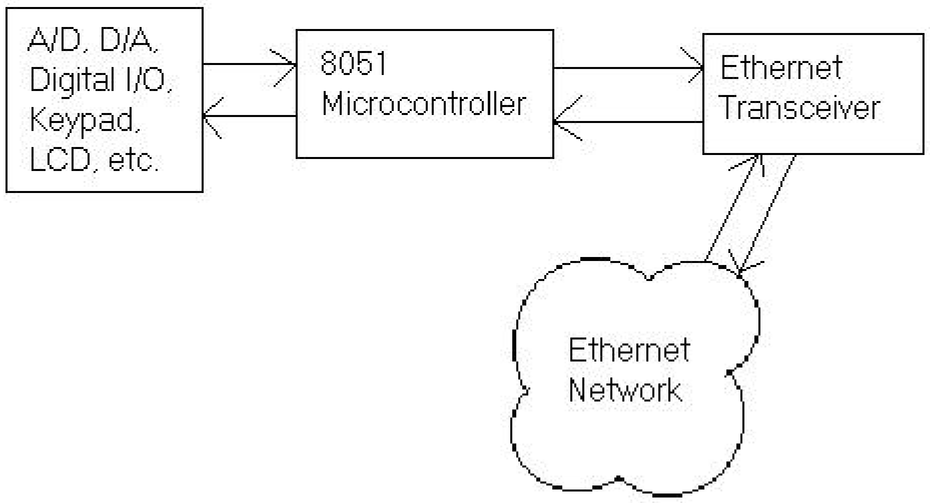


Figure 1 - Complete System Level Block Diagram

Table 2 - System Inputs and Outputs

Subsystem	Inputs	Outputs
A/D, D/A, Digital I/O, etc.	Control signals	Acquired data
8051 Microcontroller	Acquired Data, Packets	Control Signals, Packets
Ethernet Transceiver	Packets	Packets

<u>A/D, D/A, Digital I/O, etc.</u>
The various data acquisition components of the micro controller are used to provide data to the micro controller for processing.

<u>8051 Micro controller</u>
The micro controller receives packets from the Ethernet Transceiver. It processes these packets, and if needed, retrieves data from the I/O. The micro controller responds by sending a packet of data to the Ethernet Transceiver to be sent to the network.

<u>Ethernet Transceiver</u>
When a user on the other side of the Ethernet Network contacts the micro controller, the Ethernet Transceiver receives a packet from the network. It processes this packet and stores it so that the micro controller can retrieve it. When the micro controller is finished processing the packet, it transfers a response packet to the Ethernet Transceiver, which then passes the packet back to the Ethernet Network. The purpose of the Ethernet Transceiver is to translate packets of data into voltage levels for transmission.

Software Flow Chart

The software for the micro controller is composed of three main parts, as shown in the flow chart in Figure 2. The first is a loop that checks if a packet is waiting in the Ethernet Transceiver, and if there is a packet there it retrieves it. The next part decodes the packet and transfers the pertinent information to the appropriate packet processing function. These functions are the last part, and they process each type of packet and send the appropriate response. Table 3 shows the packet type, its function, and the response. TCP is the protocol used by the user's web browser to retrieve information from the web server in the micro controller. Since not all of the functions supported by TCP are needed for this design, only a subset of the full TCP protocol is implemented.

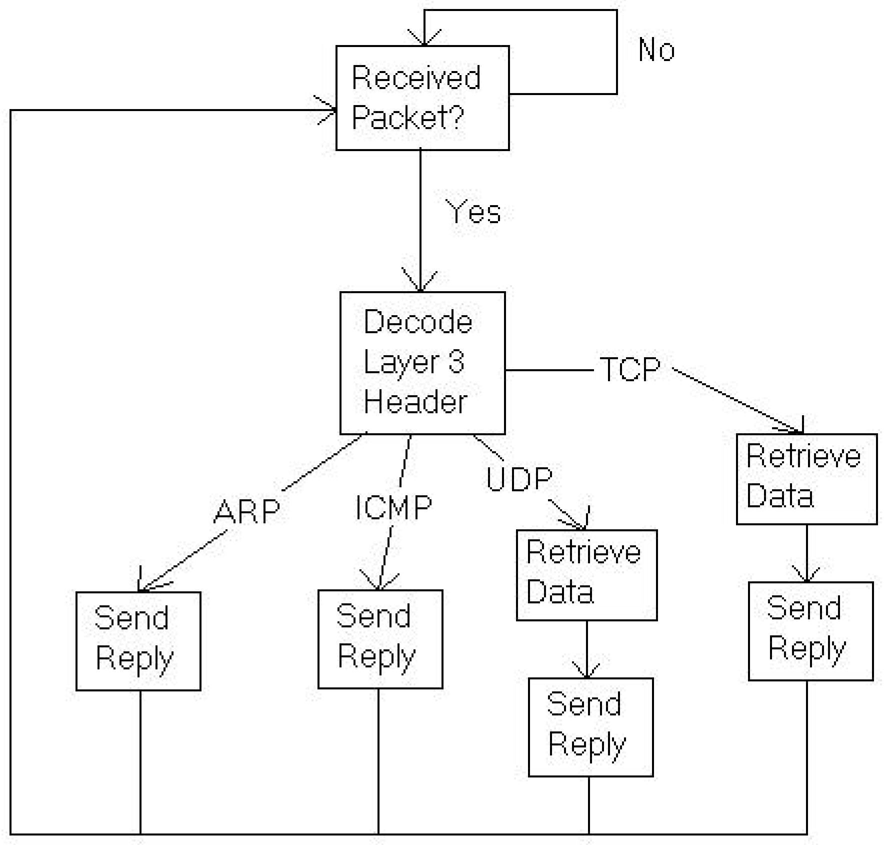


Figure 2 - Software Flow Chart

Table 3 - Packet Types

Packet Type	**Function**	**Response**
ARP	Translation from IP address to MAC address	The micro controller's MAC address
ICMP	Data link connection testing	An Echo Reply packet
UDP	Simple data retrieval	Requested data from I/O
TCP	Web browsing	HTML data packets

Hardware Design

The goal of the hardware design is to interface an Ethernet Transceiver chip to the 8051 micro controller board. After some searching on the Internet, the Crystal CS8900A Ethernet Transceiver was chosen. This chip is designed to be interfaced to a PC using the ISA bus. This allowed it to be easily interfaced to the 8051 micro controller. Crystal also provides an application note (AN181.pdf) that describes using the CS8900A in 8-bit mode. The CS8900A is used in I/O mode, meaning that it is an I/O device attached to the bus of the micro controller. The schematic shown in the application note AN181.pdf (See Appendix 1) was followed. This device has 9 address lines for direct connection to the ISA bus. Since the CS8900A has 8 2-byte ports (see Table 4), it only uses 4 of those address lines. The chip defaults to starting at address 0300h, so the remaining 5 address lines were connected to Vcc and Ground appropriately. The EMAC board provides for selecting external devices using the EXTIO pin. This pin is activated whenever a device at address 7000h is accessed. This pin was used as the Chip Select for the CS8900A. Only the upper byte of the address matters, so A8-A11 were connected to the 4 address lines on the CS8900A. To send a byte of data to the TxCMD port for instance, one would load 74h in DPH, put the data in the accumulator, then do a movx @DPTR, A instruction.

Table 4 - CS8900A IO Ports

Offset	**Type**	**Description**
0000h	Read/Write	Receive/Transmit Data (Port 0)
0002h	Read/Write	Receive/Transmit Data (Port 1)
0004h	Write Only	TxCMD (Transmit Command)
0006h	Write Only	TxLength (Transmit Length)
0008h	Read Only	Interrupt Status Queue
000Ah	Read/Write	PacketPage Pointer
000Ch	Read/Write	PacketPage Data (Port 0)
000Eh	Read/Write	PacketPage Data (Port 1)

The CS8900A uses an internal structure called PacketPage. This is an internal set of registers that provide status and configuration of the chip. It is accessed in IO mode by writing the register number to the PacketPage Pointer port, then either reading or writing the data to PacketPage Data (Port 0). This will be discussed more in the software section.

A loading and timing analysis was done to ensure that the device would operate in the given constraints. The analysis showed that the device would operate within the limits of the 80515 micro controller. These figures are in the laboratory notebook on pages 15-25.

Software Design

The software design began with the writing of the low-level functions to communicate with the CS8900A chip. These were written in assembly language because it was easier and faster. The first two functions written were the read_byte and write_byte functions. These functions read or write a byte of data to or from the giving IO Port of the CS8900A. These were first used to test the communication with the CS8900A. Next a write_word function and two different read_word functions were written. The write_word function writes a word of data to a CS8900A IO Port. The two read_word functions read a byte of data from the CS8900A IO Port. One of them, read_wordL, reads the low-order byte first, and the other, read_wordH, reads the high-order byte first. This is done because during packet reception, the Status and Length of the packet are read from the Receive/Transmit Data (Port 0) high-order byte first, but the rest of the data is read low-order byte first, as per AN181.pdf.

Next functions to test and initialize the CS8900A were written. First a test is done to ensure that the CS8900A is there and functional. This is done by reading from PacketPage Data Port 0, and checking that 0x630E is read. Next the PacketPage register RxCFG is read and a check is done for a return value of 0x0003. If both of these succeed, then the CS8900A is there. Next it is configured by setting values in certain PacketPage registers. First, the chip is reset by writing the

RESET value to PacketPage register SelfCTL. Then the SelfST register is read until the chip is finished initializing. Next the chip is configured for what packets to receive. The values RX_OK_ACCEPT, RX_IA_ACCEPT, and RX_BROADCAST_ACCEPT are written to the RxCTL register, to turn on reception, accept packets destined for the current address, and accept broadcast packets, respectively. Then the MAC address of the server is written to the Individual Address register. Then the 10BaseT receive and transmit are turned on by writing SERIAL_RX_ON and SERIAL_TX_ON to the LineCTL register.

Once the chip is initialized, the server enters the main loop, where it is in listening mode. Here is the psuedo-code for the main loop:

```
Poll CS8900A
if  packet_received then
    read_packet into receive_buffer
    switch receive_buffer->packet_type
        case IP:
            process_ip_packet
                switch ip_packet->packet_type
                    case ICMP:
                        send_ICMP_reply
                    case UDP:
                        retrieve_data
                        send_UDP_reply
                    case TCP:
                        process_TCP_packet
        case ARP:
            send_ARP_reply
end
```

The read_packet function (rx_packet) was then written. This function reads the Receive Status and Receive Length from the Receive/Transmit Data Port 0. It does this by calling the read_wordH function to read both of these words high-byte first. It then reads from this port using read_world until it reads a number words equal to Receive Length divided by 2. It stores this data in a global receive buffer to save memory.

In order to decode the packet, structures were created to represent the different headers of the protocols, as defined in the RFC for that protocol. A pointer of each type of structure is created to point to the receive buffer. This is a mapping process to map the headers to the information in the receive buffer. So for instance, the Ethernet header, the first header, contains the source MAC address, destination MAC address, and packet type (either IP or ARP). So a structure containing this is created:

```
struct eth_hdr
{
  unsigned char dhost[MAC_ADDR_LEN];          // destination MAC address
  unsigned char shost[MAC_ADDR_LEN];          // source MAC address
  unsigned int type;                          // packet type
};
```

Then a pointer to the receive buffer is created:

```
struct eth_hdr *rx_eth_hdr = (struct eth_hdr *)rx_buffer;
```

So now the packet type can be referenced by using rx_eth_hdr->type. This is the process that was done for each of the headers (Ethernet, ARP, IP, ICMP, UDP, and TCP)

Next the IP packet is processed. First, the source MAC and IP addresses are saved for when a reply is sent, so that ARP does not need to be used to discover the MAC address. Then one computes the checksum of the header to ensure that there were no transmission errors. This is done by taking the one's complement of the one's complement sum of all the words in the header, skipping the existing checksum. So a 32-bit integer is used to store the sum of all the 16-bit words in the header. The complement of this is created and stored in a 16-bit integer using:

```
checksum = (~((sum >> 16) + (sum & 0xffff))} & 0xffff
```

If the checksum matches what is stored in the header, then the packet is valid, and the protocol type is returned.

Then the ICMP, UDP, or TCP handler, depending on the protocol type, processes the packet.

For an ICMP packet, the header is checked to see that it is an ICMP Echo Request. The header checksum is calculated to ensure that the packet was received without error. Then the source and destination MAC and IP address are switched. The packet is sent to the ICMP transmit handler, with an Echo Reply type and the data from the original packet. A new checksum is created, and the packet is then sent to the CS8900A chip.

The process is similar for a UDP packet. The header checksum is verified, and the destination port checked to see if it is the port defined for the UDP server, in this case port 0x6969. The handler then retrieves the data. For testing this is a text message. The transmit handler is called and passed this data. The transmit handler updates the information in the header, and computes a new checksum. The packet is then sent to the CS8900A chip.

For a TCP packet, special consideration must be taken to keep track of an open connection. A table is kept that keeps track of the open TCP connections. For now this table only holds data for one connection, but could be increased to allow for more. This table lists the connection's current local sequence number, remote sequence number, local port, remote port, and current state. The TCP handler first checks the packet checksum, then looks in the connection table to see if this is an open connection. If it is not an open connection, a new connection is created and set to the listening state. The packet is then processed based on what the current state is. The current connection information is updated and a response packet is sent. If the connection is in the ESTABLISHED state, then a flag is set meaning that the data should be processed and a response sent. If this is so, then the port is checked to see if it is the HTTP port, then the data is sent to the HTTP handler.

Testing

At first there were many problems with the hardware. First the circuitry for the connection to the 10BaseT connector was constructed, and tested by plugging in a cable and seeing if there was a link light. There was none, so the hardware connecting the chip to the EMAC board was constructed and tested. Eventually the EMAC was able to communicate with the CS8900A using the test function discussed earlier. Once the chip was determined to be working, the 10BaseT circuitry was reworked, and eventually a link light was established. This light turns on whenever a cable is plugged into the 10BaseT connector and into either a hub with a straight-through cable, or another computer with a crossover cable. Then the initialization functions were run, and the polling function was put into a loop to check if the RxEvent register was read properly. Then a computer was connected through a hub to the CS8900A board. First the MAC address for the CS8900A must be put into the ARP cache on the computer, to ensure that an ARP packet is not generated when the computer attempts to contact the CS8900A board. This is done with the command 'ARP –s 192.168.1.1 00-08-07-06-05-04' where 192.168.1.1 is the IP address of the EMAC board, and 00-08-07-06-05-04 is the MAC address. This MAC address was picked at random, and the IP address was used because it is an address for use in internal and testing networks. The computer was programmed with the IP address 192.168.1.2 so that they are on the same subnet. The ping was then issued with the command 'ping –n 1 192.168.1.1', which issues 1 ping packet to the address 192.168.1.1.

The code on the EMAC board was sent to run until the RxEvent register detects that a packet is received. The code did stop and the RxEvent register was set to the correct value(0x0504). Next the code was stepped through to see what was read from Receive/Transmit Data Port 0. The Status and Length read as 0x0000 or 0x0003, and so did the data. So the received packet could not be read from the chip. This code was then changed to read the Status and Length from the RxStatus and RxLength registers (0x400 and 0x402 respectively). The correct status and length were read. Then the data was read from the RxData register, starting at 0x404. The first 8 bytes were read correctly, then garbage was received. Crystal Semiconductor's Application Engineers were contacted, but the people that worked on the CS8900A no longer work there, and Crystal no longer takes support over the phone. So the problem was sent over the web using their technical support home page, but no response has been received.

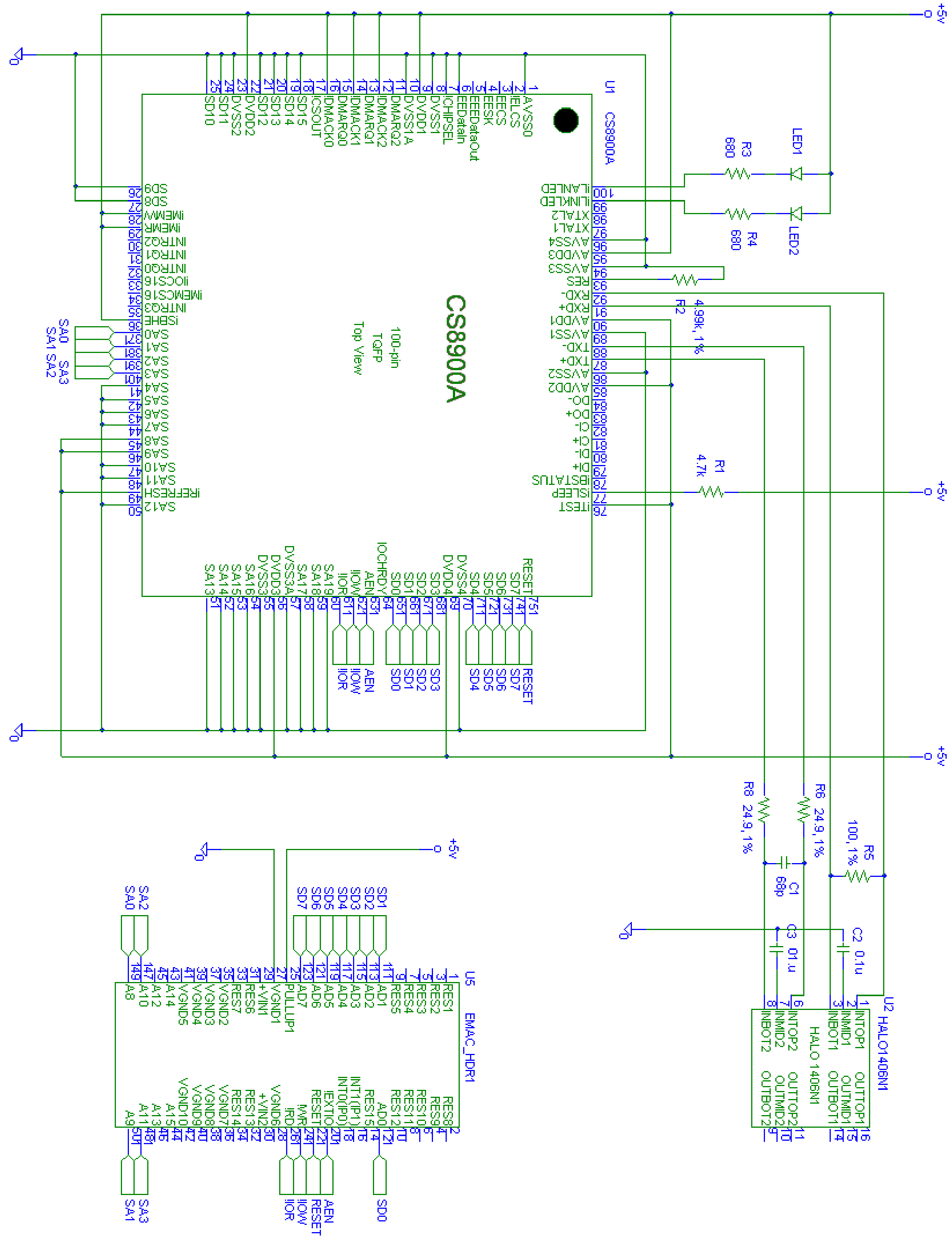
Since the receive did not work, the transmit capabilities were tested. The EMAC board was setup so that when the 1 key on the keypad was pressed, a ping packet would be sent to the other computer. The computer received that ping packet, and it was looked at in a protocol sniffer (Ethereal, see www.ethereal.com). The protocol sniffer displayed the packet and showed that it was the correct structure. Therefore the transmit functionality works correctly.

In order to test the protocol handlers, a packet was decoded using the packet sniffer, then hard coded into the receive buffer. The software was stepped through the protocol handlers, then through the construction of the response packet. This packet was compared against the packets taken from the protocol sniffer. The packets were shown to be correct for the ICMP, UDP, and TCP protocols.

Results

The complete system does not work as expected. This is because the receive functionality does not work properly. However, it was shown that the software would process the packets correctly and construct a correct response packet. A packet can also be transmitted from the server. Once the receive problem is corrected, the server will be fully functional.

Appendix 1: System Schematic



Appendix 2: Overview of TCP/IP and HTTP

TCP/IP is the suite of protocols used on the Internet to allow communication between computers. It is a layered protocol based on the Open Systems Interconnect (OSI). Data is sent from one computer to another in packets. The data starts in the Application at Layer 7. The application sends the data to the next layer, Presentation Layer 6, where it is formatted and headers/footers are added. This continues until the data reaches Layer 1, the Physical layer. It is translated to electrical signals and sent across a network. When it reaches the other side, the reverse process happens to return the data to the destination application.

The protocols included in TCP/IP include the Transmission Control Protocol, the User Datagram Protocol, the Internet Control Message Protocol, the Address Resolution Protocol, and the Internet Protocol. The Hypertext Transport Protocol operates on top of the TCP/IP protocol.

The ARP, IP, and ICMP protocols operate with each other. The IP protocol provides the network-addressing scheme and allows end-to-end connectivity. The ARP protocol allows a client to find the hardware address on a LAN segment that corresponds to an IP network address. The ICMP protocol allows for testing of connectivity using “ping” messages.

The UDP and TCP protocols operate on top of IP and allow for data transfer. UDP provides a means for connectionless data transfer, which allows sending data without the overhead of initiating a connection. TCP provides for reliable, connection oriented data transfer. A connection is established with a question and answer communication. Data is transferred, with a response after a pre-established “window.” If a packet is not received, it can be retransmitted.

Appendix 3: Software Availability

All software discussed in this paper can be downloaded in zip format at:
http://cegt201.bradley.edu/projects/proj2002/8051web