NETWORKING WITH THE 8051

SUBIR KUMAR PADHEE
SHRIVATHSA KESHAVA MURTHY

Final Project Report
ECEN 5613 Embedded System Design
April 30, 2016

1 INTRODUCTION

Rear Admiral Grace Murray Hopper, best known for the invention of the compiler, "the intermediate program that translates English language instructions into the language of the target computer said she did this because she was lazy and hoped that 'the programmer may return to being a mathematician.' [1]

Laziness sure is the Mother of Invention!

The Embedded System Design course had most of us spend sleepless nights working on our assignments. While in the initial part of the course, spending time in the lab was necessary as the NVRAM (used as the code memory) had to be programmed using the standalone programmer stationed in the laboratory, it became optional in the latter half when we could use the Flip utility and program the microcontroller via UART. It was still cumbersome as one had to carry the board to one's place and carefully so!

We thought, wouldn't it be convenient if we could program the board over the Ethernet or even better, over the air using wireless Internet?

Our project implements an automated programmer for our 8051-development board. The implementation of a web server hosted by the microcontroller opens up the door for a plethora of applications. We implemented an Instant Messenger application which allows a remote user to interact with the microcontroller itself or a user seated at the computer to which the 8051 is connected.

2 TECHNICAL DESCRIPTION

2.1 Board Design

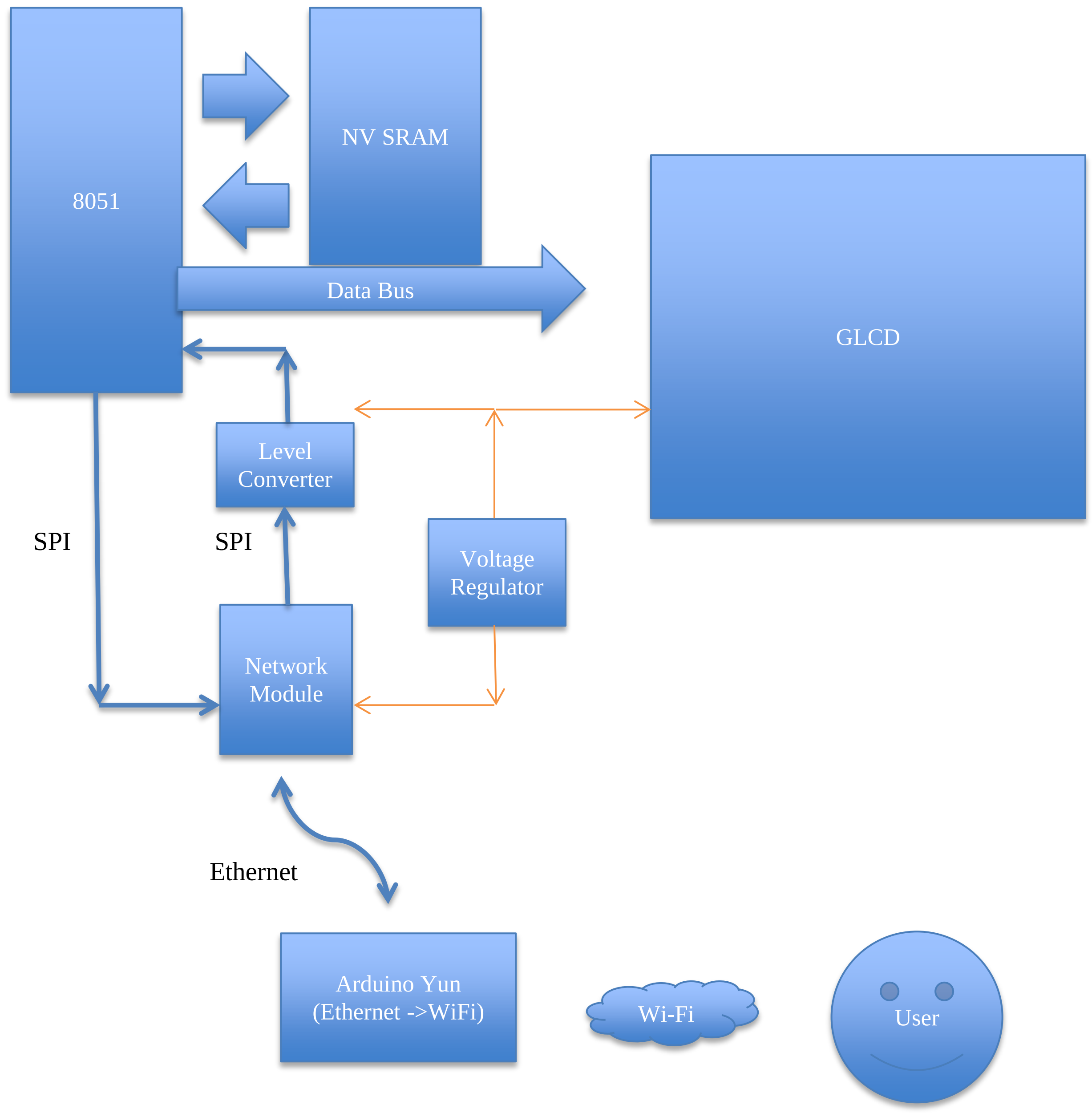


Figure 0

2.2 The Network Module (W5100)– Hardware 1

We used the WIZnet W5100 Network Module with Mag Jack - WIZ811MJ to provide network interface to our microcontroller. The W5100 is a Ethernet Controller designed for embedded applications. It has a hardwired TCP/IP stack and integrated MAC and PHY layers. The TCP/IP stack supports TCP, UDP, IPv4, ICMP, ARP, IGMP and PPPoE protocols. We have used the TCP protocol, which is a Transport layer protocol, and the IPv4 protocol, which is a Network layer protocol in our project. The chip provides for a total of 16 Kilo bytes of buffer space- 8 KB for receive buffer and the other 8 KB for the transmit buffer.
The W5100 is capable of both parallel and serial communication with the host microcontroller. Since our microcontroller – the AT89C51RC2 has a Serial Peripheral Interface module and the data and address pins on the microcontroller had been wire wrapped to the brim, we chose to use the SPI interface to interact with the network module. The module operates at 3.3 V but is tolerant to 5 V input/output.

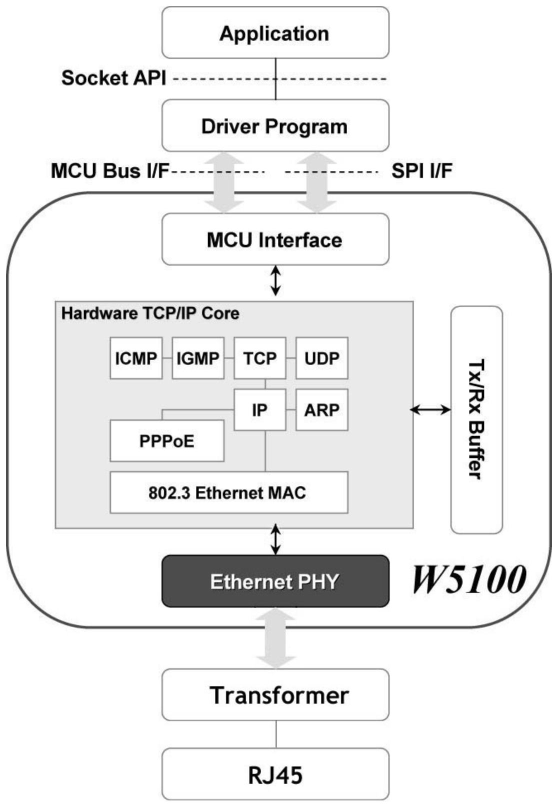


Figure 1 [2]

The W5100 provides a total of 4 sockets and hence 4 simultaneous connections, which share the Transmission and Reception buffer space. We required just one connection at a time. In fact, more than one connection at a time would be catastrophic for the microcontroller programming application if not the others. Therefore we used one socket with the entire 16 KB Rx/Tx buffer dedicated to it.

2.3 The Graphic Liquid Crystal Display (GLCD) – Hardware 2

We used a LED backlit 64x128 graphic LCD – the GDM12864HLCM module. It uses the KS0108B chip for controlling the display. The GLCD is divided into two halves-left and right and two chips are employed to control the two halves. The interface with the microcontroller is a simple 20 pin parallel interface where the 8 data pins from the microcontroller's Port 0 connect with 8 data pins of the GLCD. The Enable, Register Select, Read/Write, Chip Select 1&2 signals of the GLCD are fed by GPIO pins of the microcontroller.
The GLCD has a total of 8192 pixels divided into 8 pages along the shorter side, each page having 1024 pixels. While the GLCD provides control of each of these pixels for generating wonderful graphics, it also lets one customize text printing by allowing one to print text of custom size and font. We used it as an interface for the Instant Messenger application.

2.4 Firmware Design

2.4.1 Network Module

2.4.1.1 The Memory Map

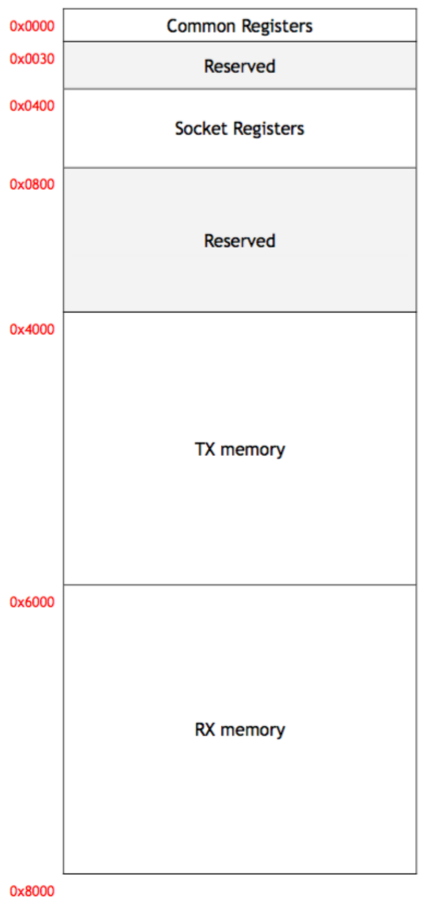


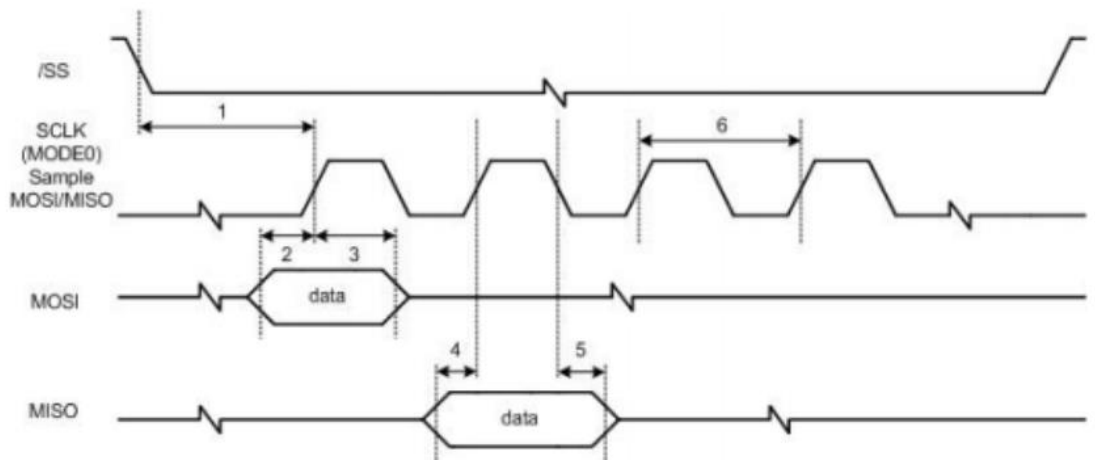
Figure 2 [2]

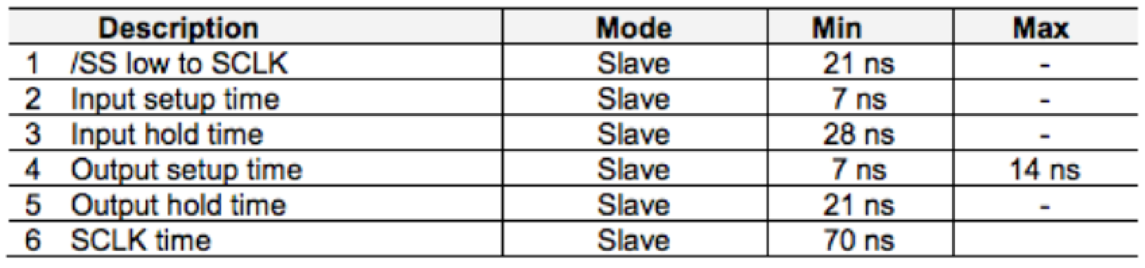
The W5100 has a number of registers, which facilitate control by the host MCU (Micro Controller Unit). These are organized as Common registers which deal with the overall functioning of the module and four sets of Socket registers one each for the four sockets. These registers are memory mapped as the above image depicts. The Transmit and Receive buffers occupy the major chunk of memory. A total of 8KBof memory is reserved for the Receive buffer from ox8000 to 0x6000 for all the four sockets taken together. Likewise for the Receive buffer- from 0x6000 to 0x4000.

2.4.1.2 Interface with the MCU

Like mentioned earlier, the MCU interacts with the network module using Serial Peripheral Interface (SPI) with the MCU acting as the master and the Network module the slave
The MCU has a dedicated SPI module; hence there was no need to employ bit banging. Since we had only one peripheral communicating on SPI, the Slave Select Disable bit was set on the SPCON register of the 8051. A ‘Zero polarity’ and ‘1 Phase’ setting of SPI was chosen to make sure the SPI Clock is low when idle and outgoing data from the MCU is sampled on the rising edge of the clock. A baud rate equaling half the peripheral clock of the MCU is chosen for the communication.

The timing requirements for the SPI of the W5100 are as follows:





Description	Mode	Min	Max
1 /SS low to SCLK	Slave	21 ns	-
2 Input setup time	Slave	7 ns	-
3 Input hold time	Slave	28 ns	-
4 Output setup time	Slave	7 ns	14 ns
5 Output hold time	Slave	21 ns	-
6 SCLK time	Slave	70 ns	

Figure 3 [2]

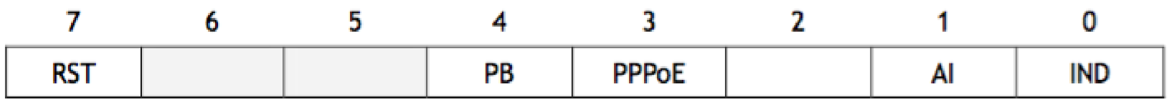
SS is the Slave Select line
MOSI is the Master Out Slave In line which carries data/instructions to the slave
MISO is the Master In Slave Out line which carries data to the master
SCLK is the clock that determines the baud rate of the communication

2.4.1.3 Initializing the W5100

Like most peripherals, the W5100 requires certain registers to be initialized with specific values based on the desired behavior before it can be used for the application it is meant for. This must be preceded by a hardware-reset sequence without which everything else becomes immaterial and the part does not respond. We learnt it the hard way!

This hardware reset is achieved by driving the RESET pin LOW for more than 2 microseconds.

This is followed by software reset by setting the 7th bit of the Mode register (the global Mode register) high. It is cleared automatically after the reset that confirms that the reset happened.



7	6	5	4	3	2	1	0
RST			PB	PPPoE		AI	IND

The 4th bit is the other bit of significance for us as it determines if the module will block ‘ping’ or not. The bit is cleared to make sure pings are responded to. Ping is a very basic way of checking connectivity between two devices connected over the network and we would not want it disabled.

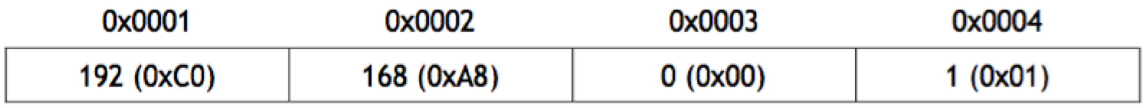
MR (Mode Register) [R/W] [0x0000] [0x00]

The Mode register is therefore set to 0x80 to start the initialization process.

The Network Module is not capable of discovering the Gateway address and hence its own IP on getting connected to a network. Therefore all the network parameters have to be manually configured.

The gateway IP address (4 bytes) is set by populating the GWR register. The default gateway was configured to be having the IP address 192.168.0.1

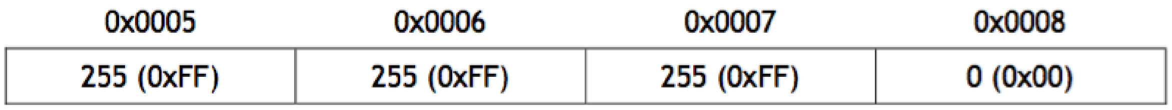
GWR (Gateway IP Address Register) [R/W] [0x0001 – 0x0004]



0x0001	0x0002	0x0003	0x0004
192 (0xC0)	168 (0xA8)	0 (0x00)	1 (0x01)

Populating the SUBR register sets the Subnet mask (4 bytes). The subnet mask is 255.255.255.0

SUBR (Subnet Mask Register) [R/W] [0x0005 – 0x0008]



0x0005	0x0006	0x0007	0x0008
255 (0xFF)	255 (0xFF)	255 (0xFF)	0 (0x00)

The SHAR is used to set the MAC address (6 bytes). The MAC address assigned must be unique in the network.

SHAR (Source Hardware Address Register) [R/W] [0x0009 – 0x000E] [0x00]

0x0009	0x000A	0x000B	0x000C	0x000D	0x000E
0x00	0x08	0xDC	0x01	0x02	0x03

A unique IP address (4 bytes) must also be assigned. This is done using the SIPR register.

SIPR (Source IP Address Register) [R/W] [0x000F – 0x0012] [0x00]

We assigned the IP address 192.168.0.3 to our board.

0x000F	0x0010	0x0011	0x0012
192 (0xC0)	168 (0xA8)	0 (0x00)	3 (0x03)

Every application on the network needs to be identified by a port number. This can be set using the UPORT register. We assigned the port number 5000 (0x1388) to our application.

UPORT (Unreachable Port Register) [R] [0x002E – 0x002F] [0x0000]

0x002E	0x002F
0x13	0x88

The Interrupt Register, Interrupt Mask Register, Retry Time Value Register, Retry Count Register can be configured to change their default values. We did not require using interrupts or retry time or count values other than the default ones, hence we let these registers retain their default values.

IR (Interrupt Register) [R] [0x0015] [0x00]

IMR (Interrupt Mask Register) [R/W] [0x0016] [0x00]

RTR (Retry Time-value Register) [R/W] [0x0017 – 0x0018] [0x07D0]

RCR (Retry Count Register) [R/W] [0x0019] [0x08]

Once the network parameters are set, the W5100 needs to be informed how many sockets will be used for the application. As discussed earlier, we need one and not more socket.
The RMSR and TMSR registers are used to configure the Rx and Tx buffer memory per socket respectively. By default, all four sockets are activated and 2 KB of memory allocated to each of them.

RMSR (RX Memory Size Register) [R/W] [0x001A] [0x55]

TMSR (TX Memory Size Register) [R/W] [0x001B] [0x55]

7	6	5	4	3	2	1	0
Socket 3		Socket 2		Socket 1		Socket 0	
S1	S0	S1	S0	S1	S0	S1	S0

S1	S0	Memory size
0	0	1KB
0	1	2KB
1	0	4KB
1	1	8KB

Figure 4 [2]

We assigned 0x03 to both RMSR and TMSR registers to ensure we can utilize the entire Rx and Tx buffer space – 8 KB each for Socket 0.

This ends the initialization process. An initialized network module is capable of responding to a 'ping' from another device on the network. A positive response to a 'ping' ensures correct initialization. The LEDs on the RJ45 jack must also glow when an Ethernet cable is plugged in and initialization has been successful. These are small tests that one can do to make sure things are going in the right track. It is at this stage that we realized that we had not performed the hardware reset in the very beginning.

2.4.1.4 Socket Setup

Once the initialization is done the socket needs to be setup and readied to receive connections from clients – We intend to set up a web server, hence we need not configure a client socket.

The last four bits of the S0_MR register are relevant and must be set to 1 to set up a socket that accepts requests based on TCP protocol.

S0_MR (Socket 0 Mode Register) [R/W] [0x0400] [0x00]

7	6	5	4	3	2	1	0
MULTI		ND / MC		P3	P2	P1	P0

P3	P2	P1	P0	Meaning
0	0	0	0	Closed
0	0	0	1	TCP

Figure 5 [2]

W5100 provides for the configuration of a separate port number and hence a separate application for each socket. This has to be configured in the S0_PORT register. We configured the port number 5000 (0x1388) for our application.

S0_PORT (Socket 0 Source Port Register) [R/W] [0x0404–0x0405] [0x00]

0x0404	0x0405
0x13	0x88

The Socket Control register and the Socket Status register then come into the picture. The former is used to control the socket state while the latter shows the socket state throughout the application. The control register takes values ranging from 0x01 to 0x40 to perform various functions, while the status register takes values ranging from 0x00 to 0x5F to indicate different states for the socket.

S0_CR (Socket 0 Command Register) [R/W] [0x0401] [0x00]
S0_SR (Socket 0 Status Register) [R] [0x0403] [0x00]

S0_CR is set to OPEN (0x01) to initialize it.
The **S0_SR** is polled till it shows the status as SOCK_INIT (0x13) and then **S0_CR** is set to LISTEN (0x02).
The socket is set up and this is reflected by the **S0_SR,** which reads SOCK_LISTEN (0x14). The web server is up at this stage!

When client sends a connection request to the server and a connection is established, the **S0_SR** changes its state to SOCK_ESTABLISHED (0x17). Once a connection has been established, the data read/ transmit APIs can be called to read or transmit data from or to the client.

2.4.2 Graphic Liquid Crystal Display (GLCD)

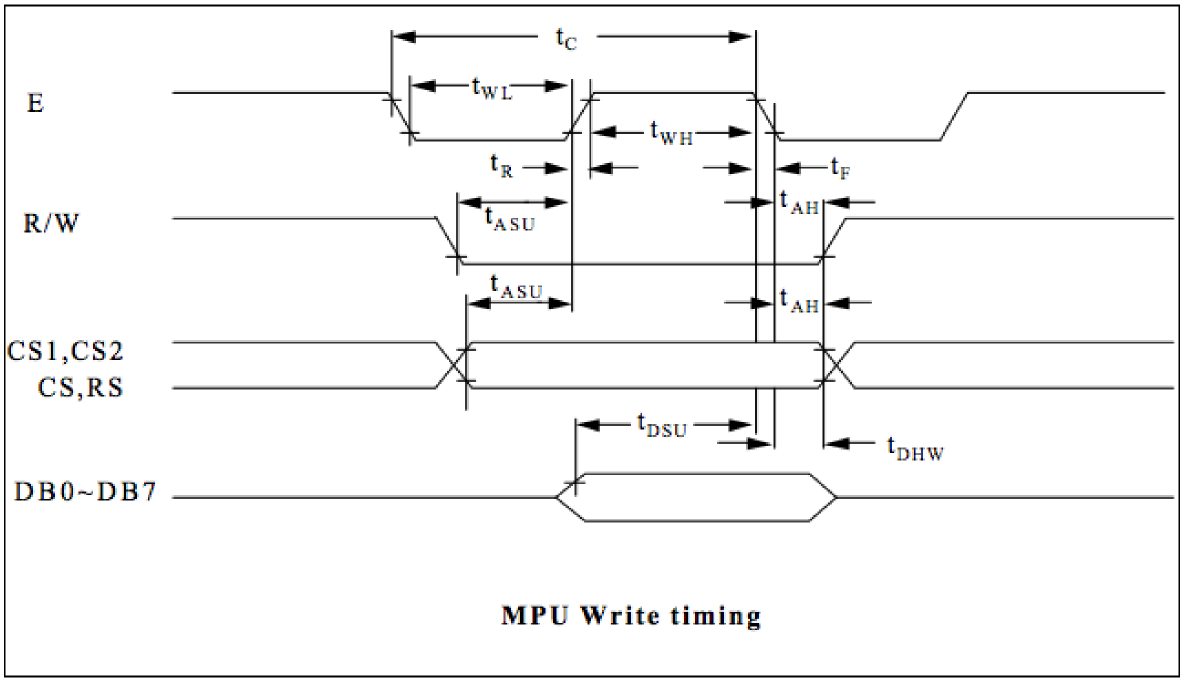
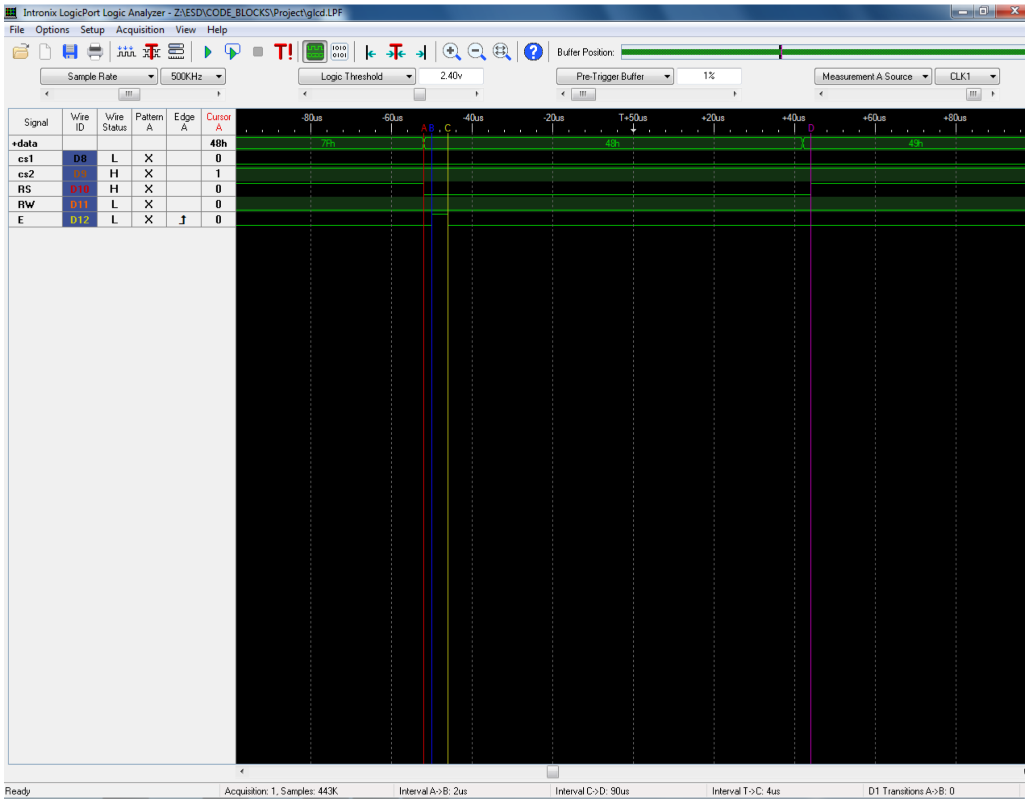


Figure 6 [3]

The timings measured on the logic port can be seen in the following screenshots



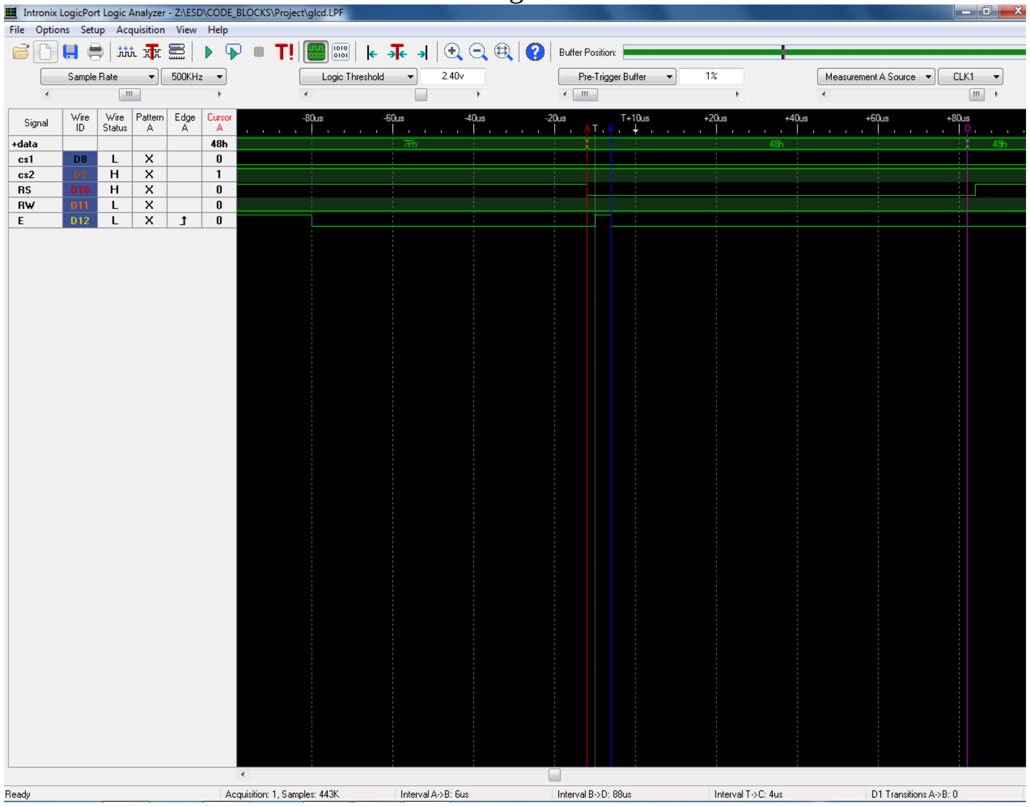
Intronix LogicPort Logic Analyzer - Z:\ESD\CODE_BLOCKS\Project\glcd.LPF

Sample Rate: 500KHz | Logic Threshold: 2.40v | Pre-Trigger Buffer: 1% | Measurement A Source: CLK1

Signal	Wire ID	Wire Status	Pattern A	Edge A	Cursor A
+data					48h
cs1	D8	L	X		0
cs2	D9	H	X		1
RS	D10	H	X		0
RW	D11	L	X		0
E	D12	L	X	↥	0

```
Ready    Acquisition: 1, Samples: 443K    Interval A->B: 2us    Interval C->D: 90us    Interval T->C: 4us    D1 Transitions A->B: 0
```

Figure 7



Intronix LogicPort Logic Analyzer - Z:\ESD\CODE_BLOCKS\Project\glcd.LPF

Sample Rate: 500KHz | Logic Threshold: 2.40v | Pre-Trigger Buffer: 1% | Measurement A Source: CLK1

Signal	Wire ID	Wire Status	Pattern A	Edge A	Cursor A
+data					48h
cs1	D8	L	X		0
cs2	D9	H	X		1
RS	D10	H	X		0
RW	D11	L	X		0
E	D12	L	X	↥	0

```
Ready    Acquisition: 1, Samples: 443K    Interval A->B: 6us    Interval B->D: 88us    Interval T->C: 4us    D1 Transitions A->B: 0
```

Figure 8

The GLCD has a parallel interface with the MPU. Apart from the 8 data pins there are bunch of control signals, which must meet the timing requirements depicted in the figure above so as to write instructions to the control registers or data to the DDRAM to be displayed on the liquid crystal display.

While the R/W signal tells the GLCD if the MPU is going to read or write information, the RS signal chooses the instruction register or the data register depending on whether it is 0 or 1.
CS1 and CS2 are used to select the left or right half of the display. The Enable pulse latches the data present on the data bus.
The RESET pin is permanently pulled up to Vcc.

There are primarily four major commands/instructions that we used on the GLCD.

- Display switch ON
- Display Switch OFF
- Page Select
- Column Select

For all commands and data issued to the GLCD, it has to be ensured that the timing requirements mentioned above are met.

Display is switched ON by sending 0x3F on the data bus. 0x3E switches OFF the display.

As discussed earlier, the display is divided into 8 pages (page 0 to 7) along the shorter side- each page being 8 pixels in height and 128 pixels in width. Evidently we need 3 bits to uniquely identify a page. To select a particular page represented by xyz, the data bus must have '10111xyz' i.e. 0xB8 OR-ed with a byte stating the page number.

Each page has 128 columns and the control automatically shifts to the next column of the same page once data is written onto the current column. This auto increment happens only till the 63rd column of the left side after which column 0 is again selected. This happens because two separate controllers control the left and right side of the display. Therefore care must be taken to manually select the right side of the display when the boundary is being crossed. OR-ing the column number with 0x40 and putting the resultant on the data bus selects a column.

2.4.3 Network Interface Module (ENC28J60) – Not included in final product

Our original plan was to use the ENC28J60 based network module and write the TCP/IP stack in software. Using the W5100 module was our fallback plan. We did manage to establish connectivity- received response for 'ping' request. However, we were not left with enough time to code the TCP/IP stack. Also given that we didn't have much luck with the SD card, we decided to go for the W5100 module so that we can concentrate on the application software once firmware was done. The following is a brief explanation of the progress made in the ENC28J60 network module. I will explain it through functions written by us without explaining about the registers which can be found in the attached datasheet.

The ENC28J60 interacts with the MCU over SPI.

ctrl_write( ) : this function is used to update a register on the ENC28J60 with the desired value. The first three bits have to be 010 followed by 5 bits of the address. This is followed by the data byte ctrl_read( ) : this function is used to read a register value on the ENC28J60. The first three bits have to be 000 followed by 5 bits of the address. The data byte follows on the MISO line.

buffer_write( ) : this function is used to write data to the transmit buffer . The first byte sent must be 0x7A. This is followed by the data bytes

buffer_read( ) : this function is used to read data from the receive buffer. At first 0x3A must be sent. The data bytes follows on the MISO line.

nic_init( ) : Unlike the W5100 module, there is no hardware reset in this module. It just requires a software reset. 0xFF must be sent on the MOSI line to reset the chip.

The ENC28J60 has 4 banks of registers, which can be selected by writing to the 0-1 bits of the ECON1 register.

The next step is to set up the receive and transmit buffer by updating the registers that are pointers to the start and end addresses of the buffer. These registers are in the Bank 0.
Bank 0 is selected by writing 0x00to ECON1.
ERXST, ERXND, ETXST and ETXND registers are updated with required values. These determine the start and end addresses of the receive and transmit buffers respectively.

The ERXFCON register is then updated with the value 0x00to allow all frames i.e. disable filtering of any kind.

The next step is to update the MAC registers which are in Bank 2.
ECON1 is updated with 0x06 to select bank 2.
MACON3 is updated with 0xB5 to enable padding, auto CRC and Full duplex connection.
Updating MAMXF with 0xEE05 sets the frame length

The MAC address (6 bytes) is configured in the MAADR0-5 registers

The initialization sequence ends by enabling frame reception. This is done by writing 0x04 to the ECON1 register.

At this stage we sent a ‘ping’ request from a computer connected to the 8051 via an Ethernet cable and could see the 8051 respond.

We wrote functions packets_send() and packets_recv() to send and receive data but these could be tested only when the upper network layers were implemented.

2.4.4 SD Card – Not implemented in the final product

Block diagram

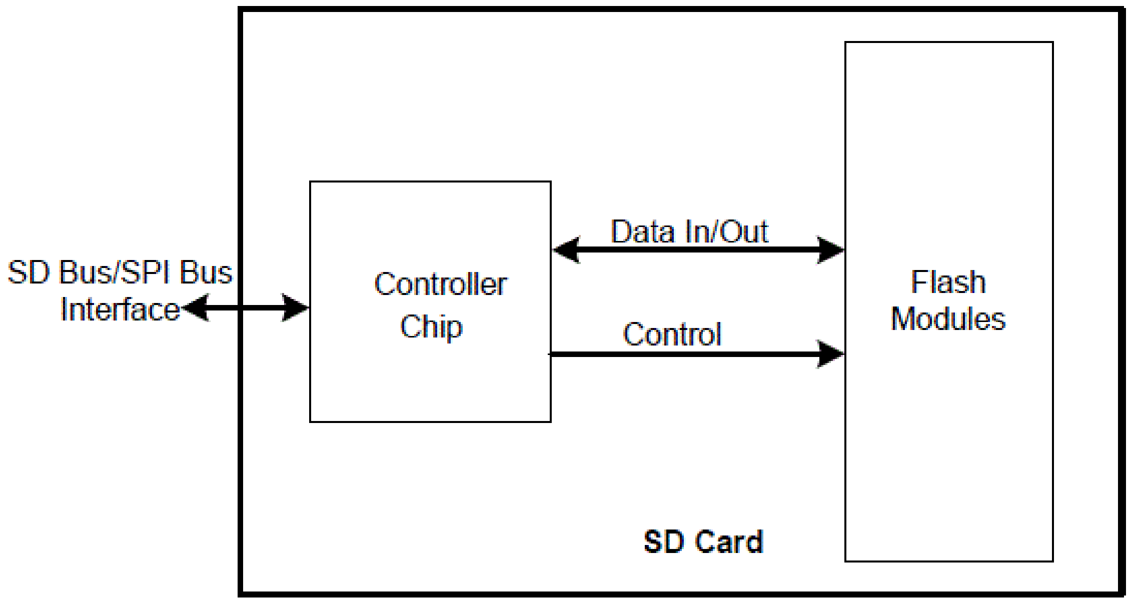


Figure 8_1 [9]

SD card works in two modes. SD mode and SPI mode. We have implemented SPI driver to initialize SD card.

SPI Protocol
It supports byte oriented protocol. Master, 8051, sends commands and the slave, SD card, responds in a given format. SD card is selected by an active low signal SS.

Initialization flowchart

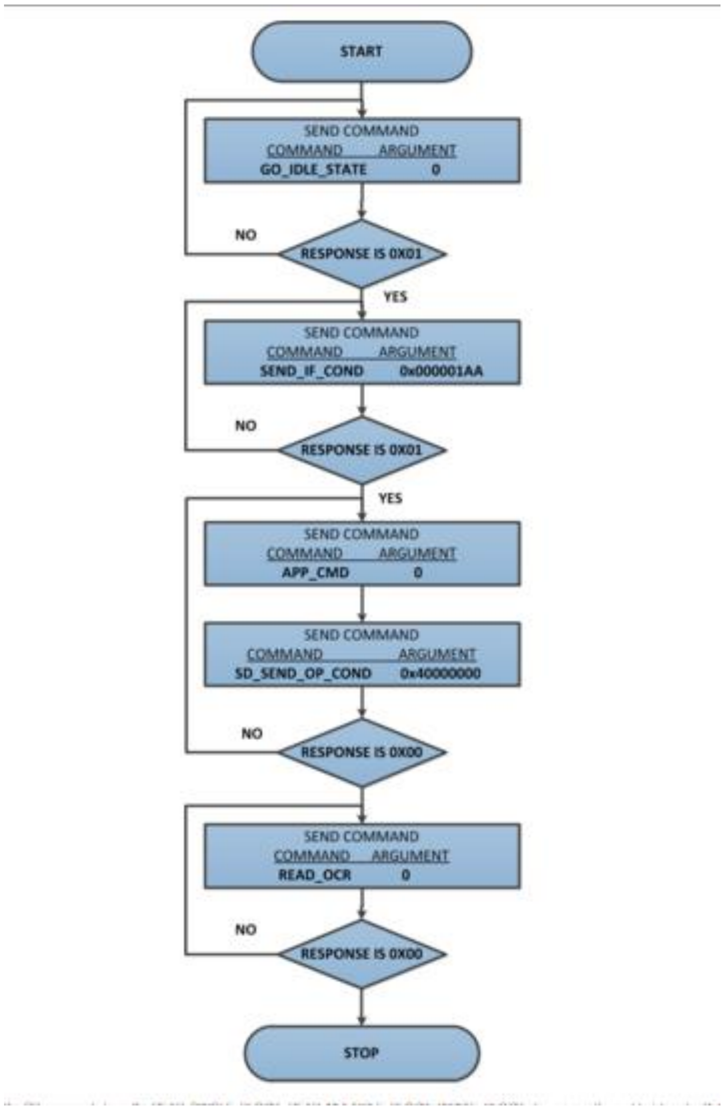


Figure 8_2 [9]

We were successfully able to initialize the SD card and get respective responses as per flowchart above. We also received OCR response and correct SD card type was interpreted. But, we could not read or write to a sector

2.5 Software Design

Software is the application running on top of our firmware. Refer to Figure 9 for the software block diagram of the entire system. The objective of the project is to wirelessly upload an Intel hex file, generated by an IDE, running on client PC, to the 8051 microcontroller. Please note that we have assumed the Arduino Yun and the client PC are on the same network. i.e. Yun's wireless module could be connected to PC's network to support file transfers between them. Following section discusses each element in detail.

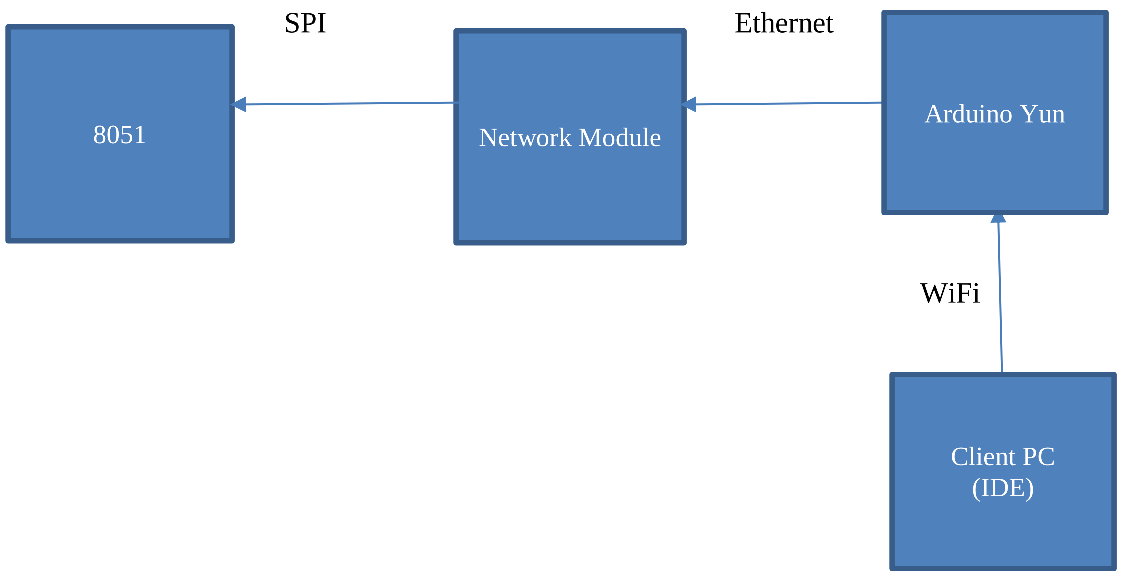


Figure 9 Software Block Diagram

2.5.1 **Network Module**

read_SPI_W5100( ): This function is called every time a value has to be read from any address location on the slave memory. The W5100 requires a protocol to be followed for reading values over SPI wherein a Write Opcode (0x0F) has to be sent first followed by the 2-byte long address (MSByte followed by the LSByte). The data follows on the MISO line after this. The Slave Select line must be driven low during this exercise. The read_SPI_W5100 executes the aforementioned protocol and returns the read data.

write_SPI_W5100( ): This function is called every time a value has to be written to any address location on the slave memory. The W5100 requires a protocol to be followed for writing values over SPI wherein a Write Opcode (0xF0) has to be sent first followed by the 2-byte long address (MSByte followed by the LSByte) and then the data on the MOSI line. The Slave Select line must be driven low during this exercise. The write_SPI_W5100 executes the aforementioned protocol.

receiveFromClient( ): This function is called once a connection is established between our web server and a remote client. The first step is to read the number of bytes received which is stored in the register S0_RX_RSR using the read_SPI_W5100 function. Once it's determined that data has

been received, the location of the first data byte is determined. This is same as the base address for the receive buffer offset by the result of the value stored in the S0_RX_RD register AND-ed with the S0_RX_MASK (0x1FFF). Once the start address is determined, received bytes are consecutively read from the receive buffer and stored in the data memory of the MPU. The S0_RX_RD is then updated to point to a location just after the last byte read from the receive buffer. The receive buffer wraps around once it gets filled. This must be considered while reading data from the buffer.
sendToClient( ): This function is called if a connection is established between our web server and a remote client and data has to be sent to the remote client. The first step is to read the free space in the transmit buffer which is stored in the register S0_TX_FSR using the read_SPI_W5100 function. If there is space in the transmit buffer, the location of write pointer is determined. This is same as the base address for the transmit buffer offset by the result of the value stored in the S0_TX_WR register AND-ed with the S0_TX_MASK (0x1FFF). Once the write location is determined, data to be transmitted is consecutively written in the buffer. The S0_TX_WR is then updated to point to a location just after the last byte written to the transmit buffer. The transmit buffer wraps around once it gets filled. This must be considered while writing data to the buffer.

nic( ) : This function is called from the main function after all other init functions have been run. It runs an infinite while loop in which it continuously reads the status of connection from the S0_SR register. If the connection is closed at any point in time, the setup_server function is called to get the 8051 server up and running again – waiting for a client request. When a connection is established with a client, the receiveFromClient function is called. If the status is any thing othe than these two, the socket is closed forcefully. As an immediate consequence the setup_server function is called and a server socket is setup again.

The Instant Messenger : The Instant Messenger application is embedded within the receiveFromClient function. Whenever data has been received, a check is made if a '*' is received. This is the que to the web server that the client is interested in exchanging texts and it calls the sendToClient function which polls the stdin for input text and sends the same to the client once the return key is pressed. This process continues till the client sends 'bye' which is a que to the webserver to end the chat session and close the socket. The socket is then setup again and its ready for more client requests.

HTTP Server: The Web server is configured to respond to HTTP GET requests with a positive response. This is done in the receiveFromClient function.

2.5.2 **Graphic LCD**

fill_column( ) : This being a graphic LCD, any character to be displayed has to created by filling pixels. The GLCD allows filling one entire column in a page i.e. 8 pixels in one go. The fill_column function does exactly that. It passes one byte of data to the GLCD, each bit of which whitens or darkens one pixel in a column – MSb corresponds to the bottom pixel and the LSb corresponds to the top pixel. Assigning 1 to a pixel whitens it while assigning 0 darkens it.

print_char( ) : We decided to have each character occupy a space of 8x6 pixels i.e. 6 consecutive columns in a page, the first column being a blank to ensure a one pixel gap between characters. Filling 6 consecutive columns in a way that the resulting pattern looks like the desired character therefore creates a character. A 6-column wide two-dimensional byte array was created where each row had 6 bytes to be written onto the aforementioned 6 columns. Each printable character thus occupied one row in the array, the row number being the 32 subtracted from the ascii value of the character. We generated this array using the open source software 'GLCD Font Creator'.

To display a character, the print_char function runs through the columns of the corresponding row of the mentioned array and calls the fill_column function 6 times. A string is displayed by running through the length of the string and calling the print_char function for each character.
clear_glcd( ) : This function was written to clear the entire display. Looping through all the pages and all the columns of each page and setting each pixel to 0 achieved it.

clear_page( ) : this function was created to clear a page before displaying a new string on it.

2.5.3 **Client Environment**

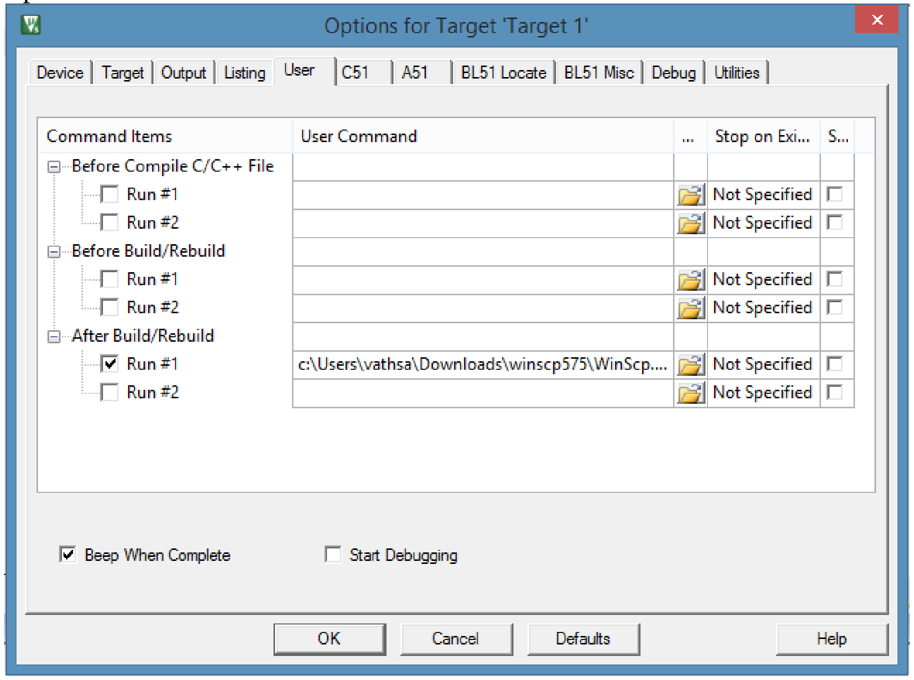
This is where the user can interact with the system to upload the Intel hex file. We have used Keil Micro Vision as the client IDE (Integrated Development Environment) in this project as it was friendlier to our script files. On clicking build button, a script is being called as part of post build arguments. Refer to Fig. 10 for the same. Post build argument gets called only if the build process of the project is successful. This meets our requirement for the project.

2.5.3.1 Windows Script

The objective here is to transfer the newly built Intel hex file from the remote PC to the Yun wirelessly. Yun has OpenWrt, a lightweight version of Debian operating system, running on it. More details on the Yun is discussed in the later sections.

2.5.3.2 WinScp

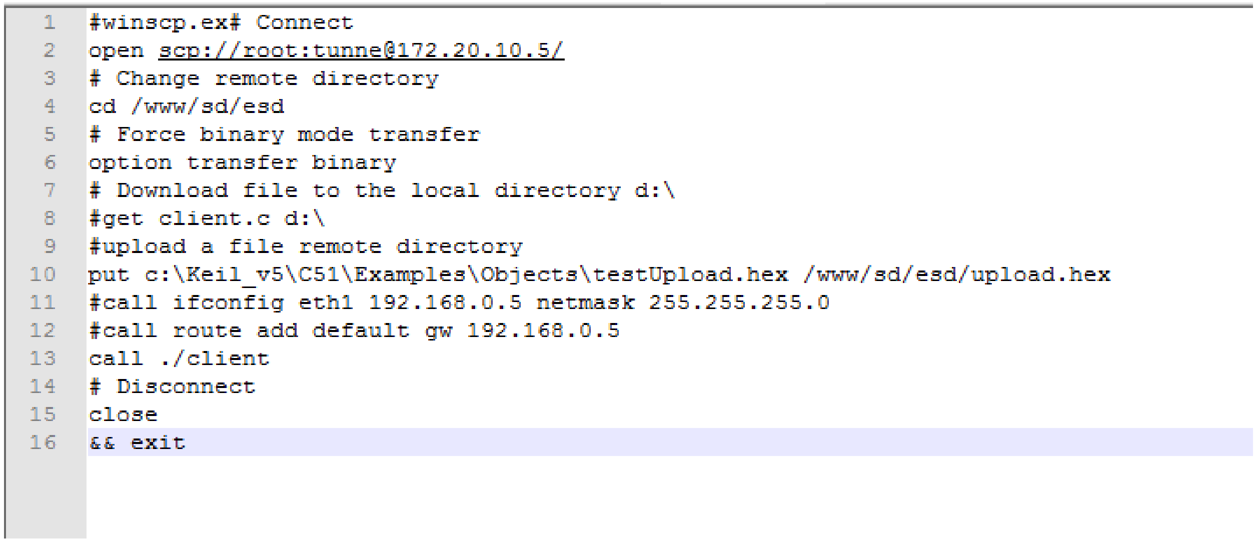
WinScp application allows you to transfer files from windows operating system to Yun’s OS. Here, we are executing this application as Keil’s post build arguments by providing a text file as an argument.
WinScp.com /script=sendfile.txt



Command Items	User Command	...	Stop on Exi...	S...
Before Compile C/C++ File				
☐ Run #1			Not Specified	☐
☐ Run #2			Not Specified	☐
Before Build/Rebuild				
☐ Run #1			Not Specified	☐
☐ Run #2			Not Specified	☐
After Build/Rebuild				
☑ Run #1	c:\Users\vathsa\Downloads\winscp575\WinScp....		Not Specified	☐
☐ Run #2			Not Specified	☐

☑ Beep When Complete ☐ Start Debugging

Figure 10 Keil Post Build arguments



```
#winscp.ex# Connect
open scp://root:tunne@172.20.10.5/
# Change remote directory
cd /www/sd/esd
# Force binary mode transfer
option transfer binary
# Download file to the local directory d:\
#get client.c d:\
#upload a file remote directory
put c:\Keil_v5\C51\Examples\Objects\testUpload.hex /www/sd/esd/upload.hex
#call ifconfig eth1 192.168.0.5 netmask 255.255.255.0
#call route add default gw 192.168.0.5
call ./client
# Disconnect
close
&& exit
```

Figure 11 sendFile.txt

sendFile.txt
This section discusses the command line argument file for the WinScp application.

Open – Opens a connection with a given host. Example, open scp://username:password@172.20.10.5/. This opens an SCP connection with the respective credentials.

Put – uploads the hex file from local build directory to remote directory. Example, put srcPath DestnPath

Call – Executes a remote shell command. We use this command to configure Ethernet network settings, eth1, of the Yun. Yun is connected to 8051's NIC through Ethernet interface and respective software abstraction is eth1. When the Yun boots up, eth1 does not have an ip. We configure using the following commands.
call ifconfig eth1 192.168.0.5 netmask 255.255.255.0
call route add default gw 192.168.0.5
This is required to identify the Yun which is essential to create a socket and talk to the 8051. Also, a client program (. /client) is executed remotely which is explained in the next section.

2.5.4 Arduino Yun

Before we go further, we would like to mention that the sole purpose of using Arduino is to take advantage of its WiFi module and the operating system which it offers. Please note that Atmega microcontroller onboard, is not executing any program here. Also, Yun is referred to Arduino's operating system.
Figure 12 shows the block diagram of the Arduino Yun.

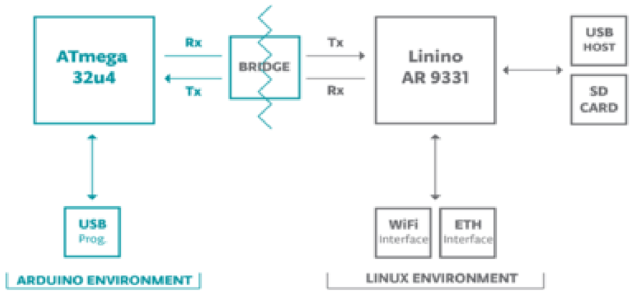


Figure 12 Arduino block diagram [4]

As shown above, Yun (Linino) is the Linux part of the Arduino. SD card is exposed to the OS as a mount point and this where the client program is stored. Client PC is connected to the WiFi interface and 8051's network interface is connected to ETH of Yun though Ethernet cable. Also, Linino can communicate with the ATmega of the Arduino through Bridge (Serial communication). This feature is not being used anywhere in the project.
We have written a client program which runs on the Yun which does two things primarily.

1. Parse the Intel Hex file received from the client computer.
2. Create a socket connection with the 8051.

2.5.5 Intel Hex File

The program parses the Intel Hex file for opcodes and places all opcodes in a buffer, assuming they are continuous. The buffer is then sent to the server, 8051, through TCP/IP sockets.
Intel Hex Format

1. Start Code – an ASCII colon character which denotes the beginning of the line.
2. Byte Count – Two hex digits which represent the number bytes of opcodes in the particular line.
3. Address – Four hex digits representing the 16 bit beginning memory address offset of data. Physical address is calculated by adding this address with a previously established base address.
4. Data Type – Two hex digits whose values ranging from 00 to 05, defining the meaning of the data field.
   00 – Data
   01 – End of File
   02 – Extended segment address
   03 – Start segment address
   04 – Extended linear address
   05 – Start linear address
5. Data – n bytes of data represented by 2n hex digits
6. Checksum – Two hex digits which could be used to verify records

Source: [5]
At this point, we have tested a program where the opcodes' address is sequential. Hence we ignore address and data type field. We read the Byte Count and extract respective number of bytes from each line.

2.5.6 Sockets

Sockets are application programming interfaces provided by the operating system which helps the user to create, control and use network sockets. A network socket is an endpoint connection in a computer network. Since most of the computers use Internet Protocol, we could say they are Internet sockets. This software abstraction provides an interface to user to send data to another network or computer and the user need not worry about implementations of layer underneath it. In a UNIX system which views everything as a file, sockets are nothing but file descriptor integers, assigned by the OS when they are created. Sockets are characterized by the following.

1. Local logical IP address and port number
2. Transport protocol – TCP or UDP protocol

Assignment of IP was done earlier as part of WinSCP script file. We have assigned 192.168.0.5 to eth1 of Yun and our client TCP application is running on port 5000. We chose TCP over UDP as the earlier provides reliable delivery service.

Stages involved in socket creation and connection

Socket
First, a socket is created using, socket (AF_INET, SOCK_STREAM, 0). First argument denotes the domain, AF_INET which is ipv4 internet protocols. Second argument, SOCK_STREAM type, denotes reliable two way connection based byte streams.

Connect
int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
Then, the newly created socket, sockfd, is connected to an address of type sockaddr which is defined by a combination of IP and port number. In this case, it is connected to 192.168.0.3, the IP of 8051 and 5000, the port number which 8051 is listening to.

Gethostbyname
This function returns a structure of type hostent for the given host name. Here, the argument for this function is either a hostname or a standard dotted decimal IP address. One of the structure members of hostent, h_addr, contains the address which is required to connect to the server, 8051.
Once a client socket is setup and connected to the server, 8051, we use send system call to send data to 8051 and recv, to receive data from the 8051. Following section explains about their implementatios.

Send – sends message to another socket
send(int sockfd, const void *buf, size_t len, int flags)
sockfd – newly created socket descriptor.
Buf – buffer to be transmitted to another socket.
Len – Length of the above buffer
Flags – Null, not used.

Recv – Receives message from another socket.
size_t recv(int sockfd, void *buf, size_t len, int flags)
size_t – number of bytes of buffer received. Description for the rest of the arguments remain the same.
So far we have discussed parsing Intel hex file and various concepts involved in sending data to the server. Now, let us take a look on how we use above implementations to our requirements.

Modes of client-server operation - Project works in 3 modes.

1. Wifi programmer – Sends the entire opcodes received from the client to 8051
2. Chat interface – Implements a TCP/IP chat client-server application. Half duplex communication between client and server to exchange messages. Messages received at the server is reflected on a Graphical LCD.
3. WebServer – A tiny web server hosted at the 8051 on port 5000 which returns a simple PHP page to client.

Let us discuss each mode in detail.

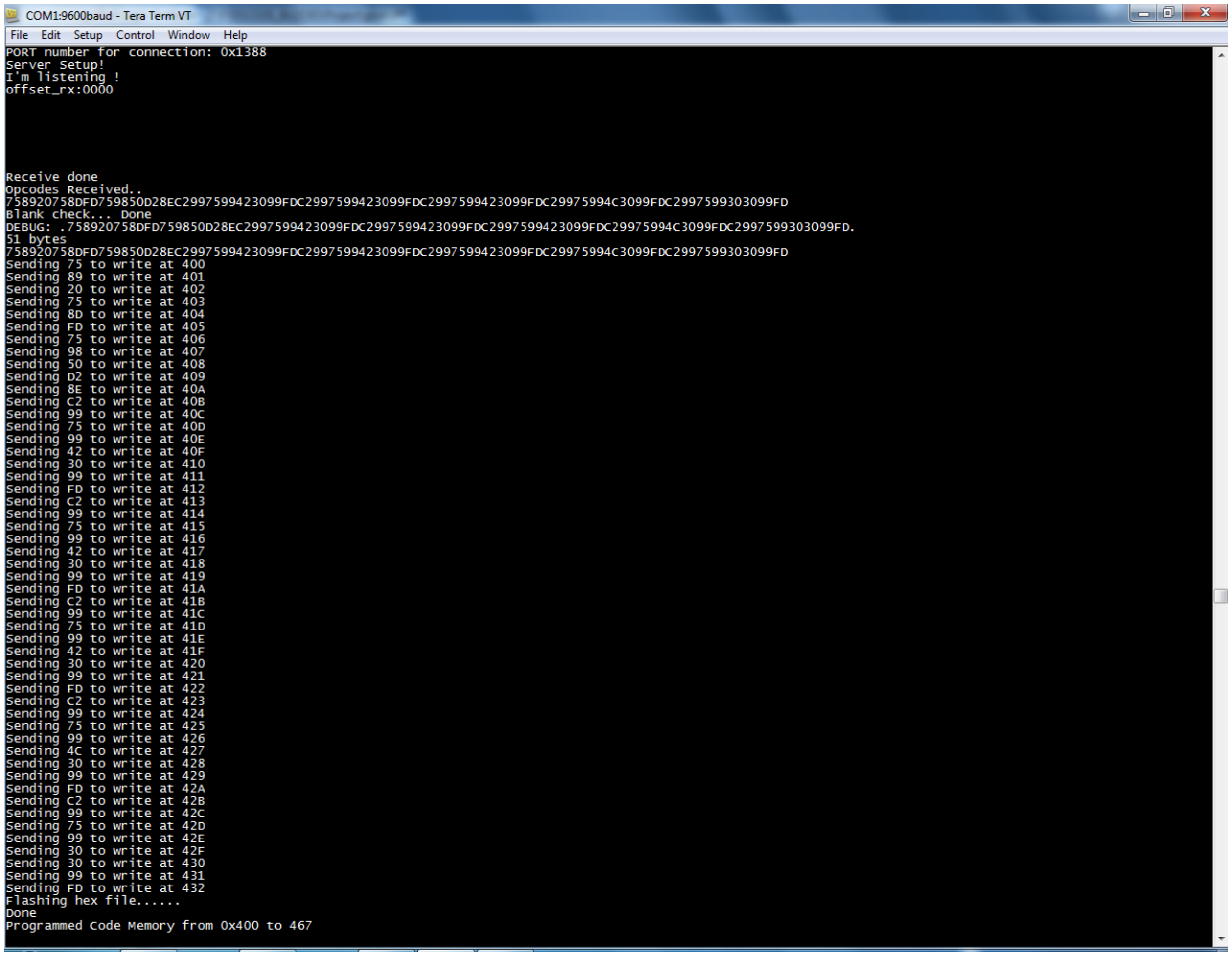
2.5.7 Wifi programmer

Client program

This mode is identified by the client program when there is no command line argument. At this point, buffer would have the parsed opcodes. This entire buffer is sent to the server using send system call. Refer to Figure 1.5 for the reception of buffer snapshot at 8051. Once done, pull EA to ground to execute the new code. Figure 1.6 shows the snapshot of the newly loaded program which writes a string “BBBL0” to serial terminal.
Entire project operates in two modes.

Mode	**EA**	**Data Memory**	**Code Memory**
1	1	NVSRAM 0x0000 – 0x7FFF	On-Chip 0x0000 – 0x7FFF
2	0	NVSRAM 0x2000- 0x7FFF	NVSRAM 0x400 – 0x1FFF

*EA is achieved using a jumper between VCC, GND and EA
*When EA=0, Data memory partition is achieved by specifying –data-loc 0x2000 in the SDCC linker settings.


COM1:9600baud - Tera Term VT
File Edit Setup Control Window Help

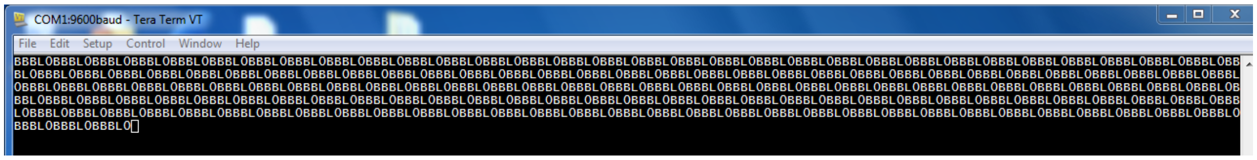
```
PORT number for connection: 0x1388
Server Setup!
I'm listening !
offset_rx:0000





Receive done
Opcodes Received..
758920758DFD759850D28EC2997599423099FDC2997599423099FDC2997599423099FDC29975994C3099FDC2997599303099FD
Blank check... Done
DEBUG: .758920758DFD759850D28EC2997599423099FDC2997599423099FDC2997599423099FDC29975994C3099FDC2997599303099FD.
51 bytes
758920758DFD759850D28EC2997599423099FDC2997599423099FDC2997599423099FDC29975994C3099FDC2997599303099FD
Sending 75 to write at 400
Sending 89 to write at 401
Sending 20 to write at 402
Sending 75 to write at 403
Sending 8D to write at 404
Sending FD to write at 405
Sending 75 to write at 406
Sending 98 to write at 407
Sending 50 to write at 408
Sending D2 to write at 409
Sending 8E to write at 40A
Sending C2 to write at 40B
Sending 99 to write at 40C
Sending 75 to write at 40D
Sending 99 to write at 40E
Sending 42 to write at 40F
Sending 30 to write at 410
Sending 99 to write at 411
Sending FD to write at 412
Sending C2 to write at 413
Sending 99 to write at 414
Sending 75 to write at 415
Sending 99 to write at 416
Sending 42 to write at 417
Sending 30 to write at 418
Sending 99 to write at 419
Sending FD to write at 41A
Sending C2 to write at 41B
Sending 99 to write at 41C
Sending 75 to write at 41D
Sending 99 to write at 41E
Sending 42 to write at 41F
Sending 30 to write at 420
Sending 99 to write at 421
Sending FD to write at 422
Sending C2 to write at 423
Sending 99 to write at 424
Sending 75 to write at 425
Sending 99 to write at 426
Sending 4C to write at 427
Sending 30 to write at 428
Sending 99 to write at 429
Sending FD to write at 42A
Sending C2 to write at 42B
Sending 99 to write at 42C
Sending 75 to write at 42D
Sending 99 to write at 42E
Sending 30 to write at 42F
Sending 30 to write at 430
Sending 99 to write at 431
Sending FD to write at 432
Flashing hex file......
Done
Programmed Code Memory from 0x400 to 467
```

Figure 13 EA=1. 8051, is listening to client connections. Once a client connection is accepted, debug logs show that respective opcodes starting from 0x400 address have been written to the NVSRAM, which is now both code and data memory.


COM1:9600baud - Tera Term VT
File Edit Setup Control Window Help

```
BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BB
BL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL
0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0B
BBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBB
L0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0BBBL0
BBBL0BBBL0BBBL0
```

Figure 14 EA=0. 8051 executing newly loaded program which writes a string to the serial terminal in an infinite loop.

Server program
void parseTcpOpcodes(unsigned int baseAddress, unsigned char *buffer)

baseAddress – Starting address where the program to be loaded. We are starting at 0x400 of the NVRAM as we cannot write before 0x400 by software. This is because of the fact that the processor, at the minimum, uses 256 bytes of XRAM and we use 1KB of XRAM and hence address generated in that range cannot be used to select external memory. Also, we programmed the NVRAM to have 0x81, 0x00 at 0x000, 0x0001 respectively, using Needham's programmer. These are the opcodes which correspond to AJMP 0x400. This is to make sure we jump to our new location every time the processor boots up.

Buffer – Raw unsigned characters received from the client. This function will internally convert 2 unsigned characters in to a byte and calls another function 'dataout' to write to NVSRAM.

```
void dataout(unsigned int oAddress, unsigned char oData){
    ptr = oAddress;
   *ptr = oData;
}
```

The above function writes an unsigned character, oData, to a specified address, oAddress. Ptr is pointing to external data memory.

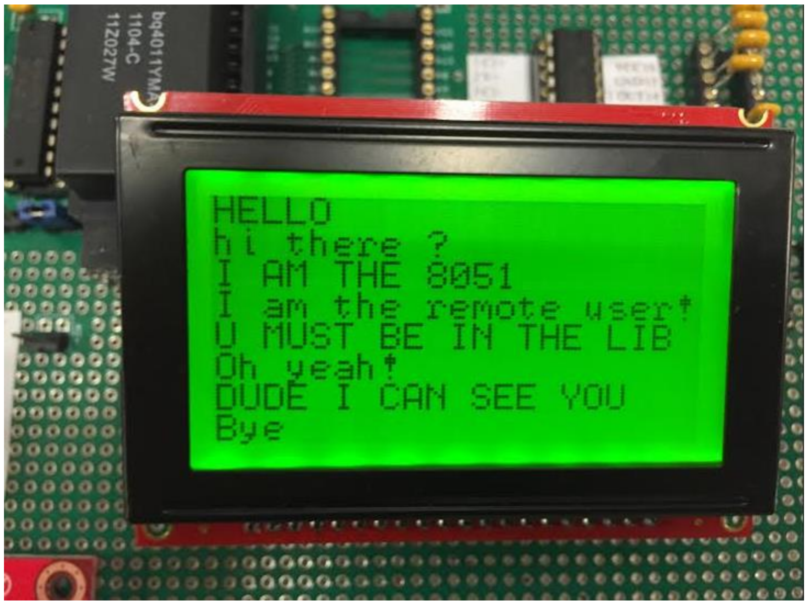
Chat interface

At the client side, this mode is identified by passing a command line argument string, /client chat. And then an ASCII character '*' is sent followed by the messages sent by the user. Server identifies this mode by this '*' character. User at the client side executes the above program and a terminal is prompted where messages can be entered. "Bye" string can be used to terminate the session at both client and server.

At the server side, user can input messages in the terminal and the same is reflected on a graphical LCD. Send and recv system calls are used at both client and server to transfer data. Also, note that this is half duplex communication. Figure 1.6.

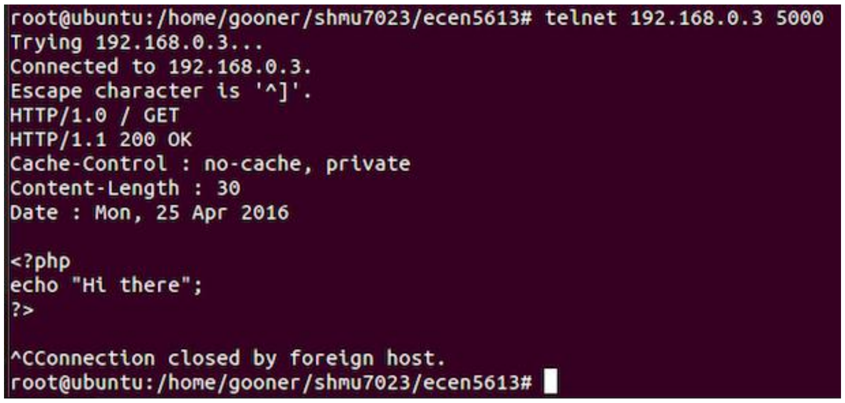
WebServer

8051 is hosting a tiny webserver on port 5000. Server identifies this mode by looking for "HTTP" in the input string. If found, it returns a GET response with a "Hi there" PHP page. This feature is to demonstrate a webserver could be hosted. A telnet session was created at the client and response was received. Figure 1.7.



```
HELLO
hi there ?
I AM THE 8051
I am the remote user!
U MUST BE IN THE LIB
Oh yeah!
DUDE I CAN SEE YOU
Bye
```

Figure 15 Graphical LCD chat snapshot



```
root@ubuntu:/home/gooner/shmu7023/ecen5613# telnet 192.168.0.3 5000
Trying 192.168.0.3...
Connected to 192.168.0.3.
Escape character is '^]'.
HTTP/1.0 / GET
HTTP/1.1 200 OK
Cache-Control : no-cache, private
Content-Length : 30
Date : Mon, 25 Apr 2016

<?php
echo "Hi there";
?>

^CConnection closed by foreign host.
root@ubuntu:/home/gooner/shmu7023/ecen5613#
```

Figure 16 HTTP GET response as seen in terminal

2.6 Testing Process

We have used modular approach in this project to develop and test each module. Each module was unit tested individually and checked in. Also, we have manually tested each unit and no regression tests have been implemented. Every time we checked in a piece of code, we made sure to test the previously implemented features and verified it does not break anything.

3 RESULTS AND ERROR ANALYSIS

We were successful in implementing features we have proposed altogether. Even though it was not part of our proposed idea, we could wirelessly program 8051 which turned out to be a better solution. This eliminates the use of Needham's programmer and we were successful in making it happen. We setup the environment and hit the build button in Keil IDE and we were able to see the new code output on the serial monitor. We could chat between 8051 and the remote user as seen in the screenshot above. Also, we could get the HTTP GET response in a Linux terminal after creating a telnet session. We faced occasional malfunction in the newly loaded code which was suspicious.

SD Card

Initial idea was to write driver for SD card to store the hex files. Although, we wrote bit-banged SPI driver and got the SD card initialized, we could not read and write data as we encountered inconsistencies in finding a common datasheet and sequence of SPI commands. We have written SPI drivers and received response from the SD card during initialization sequence.

Software based TCP/IP stack

We could not implement a software based TCP/IP stack due to time constraints.

4 CONCLUSION

This project demonstrates 8051 processor could be wirelessly programed. As long as the Yun and client PC are on the same network, you can sit in any remote location and program your development board which eliminates the use of Flip and Serial communication. We as a team learned about SPI drivers and the development of hardware drivers by reading datasheets. This also helped us understand bootloaders, sharing code and data memory, timing analysis in great detail. We had an opportunity to understand the functionality of low level NIC drivers as well as high level Windows scripting involved in application development. We could conclude that dealing with opcodes and loading them to memory by software methods was a challenging task.

5 FUTURE DEVELOPMENT IDEAS

We have demonstrated so far by programming a simple code which displays a string. This could be extended for bigger programs involving interrupt service routines and conditional jumps. We could add a mechanism in the code to look for ISRs and service them in new address as the program memory starts from 0x400. This project incorporates parsing opcodes without taking checksum in the hex file into consideration. We could add a feature to look for error in each line. This is very crucial because change in a bit value could eventually hang the processor. Writing our own driver for a WiFi module instead of using Yun can also be considered.

6 ACKNOWLEDGEMENTS

We would like to acknowledge Professor Dr. Linden Mcclure for his guidance and support throughout the project.

We also would like to acknowledge and thank the authors of various sources cited in the reference below.

7 REFERENCES

1. http://innovators.vassar.edu/innovator.html?id=8)
2. https://www.sparkfun.com/datasheets/DevTools/Arduino/W5100_Datasheet_v1_1_6.pdf
3. https://www.sparkfun.com/datasheets/LCD/GDM12864H.pdf
4. https://www.arduino.cc/en/Main/ArduinoBoardYun
5. https://en.wikipedia.org/wiki/Intel_HEX
6. http://www.engineersgarage.com/microcontroller/8051projects/graphics-lcd-interfacing-at89c52-circuit
7. http://www.circlemud.org/jelson/sdcard/SDCardStandardv1.9.pdf
8. http://www.mikrocontroller.net/attachment/21920/SDHC_SDM04G7B7_08G7B7.pdf
9. http://www.engineersgarage.com/embedded/avr-microcontroller-projects/sd-card-interfacing-project

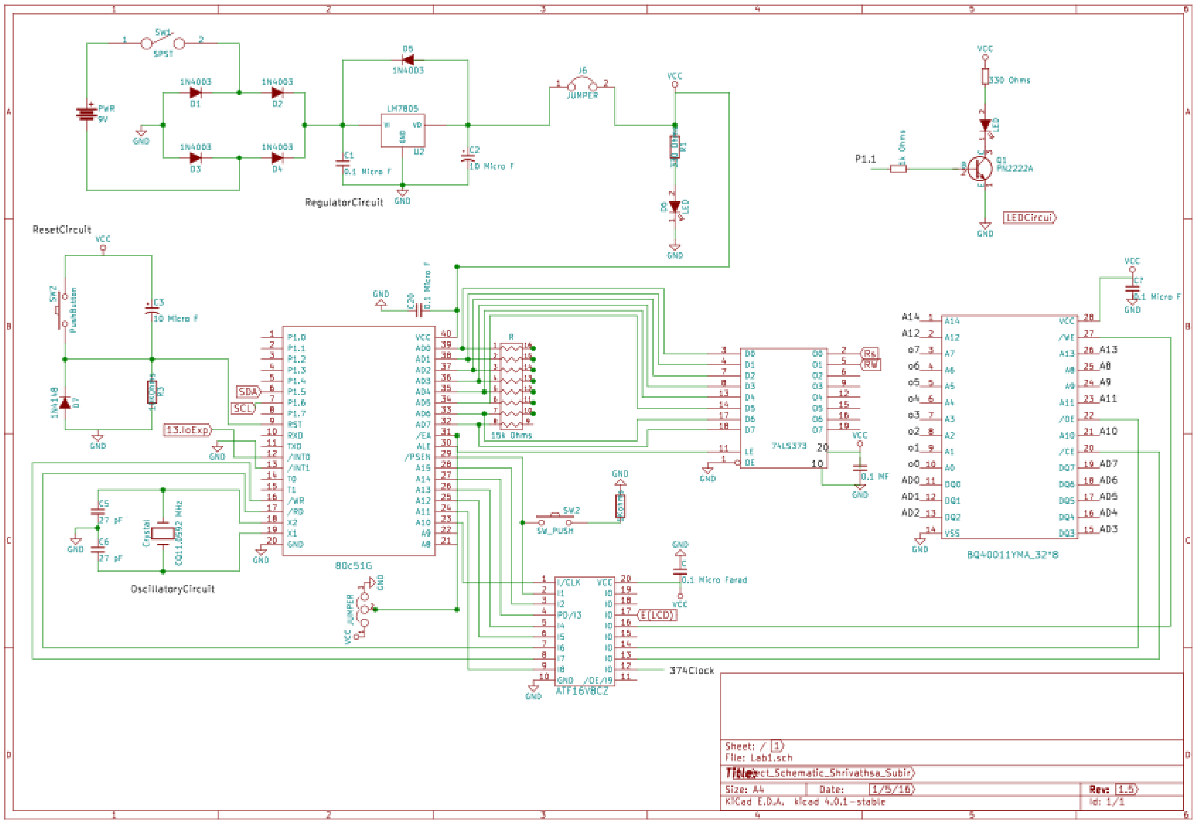
8 APPENDICES

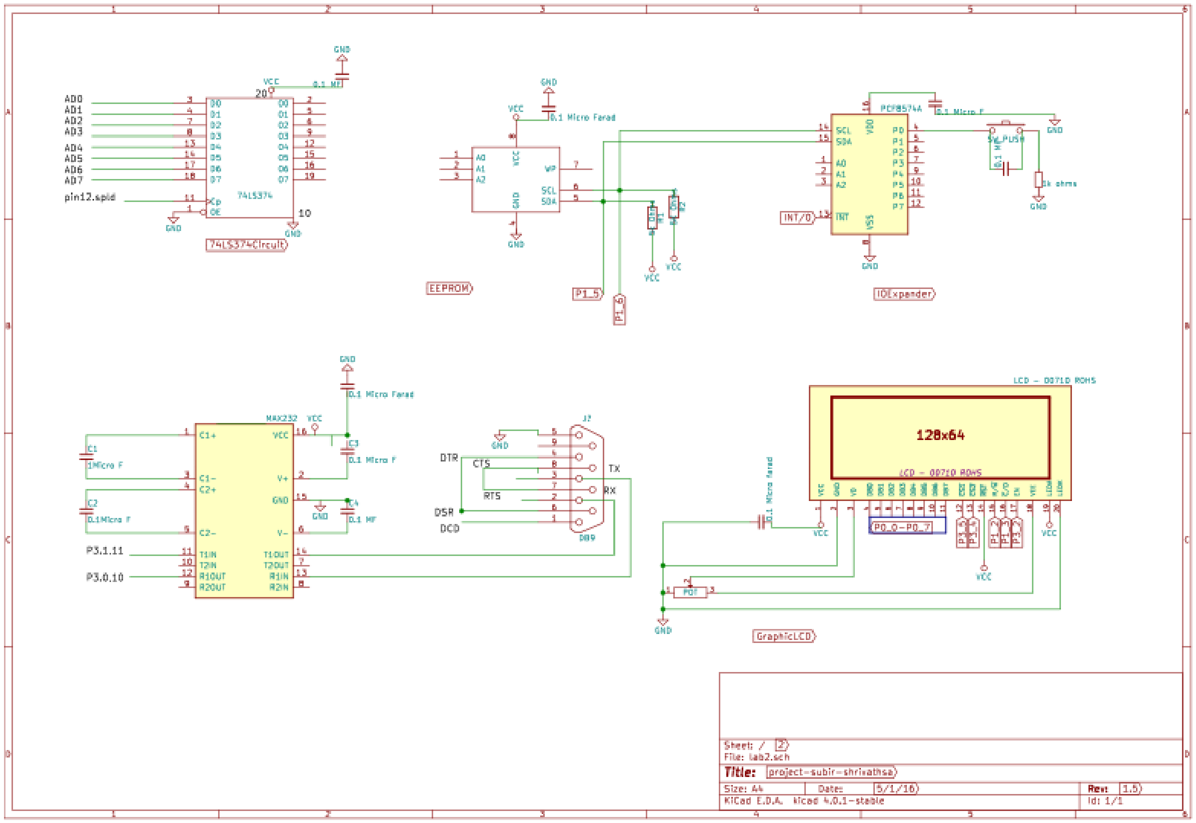
Several appendices have been attached to this report in the order shown below.

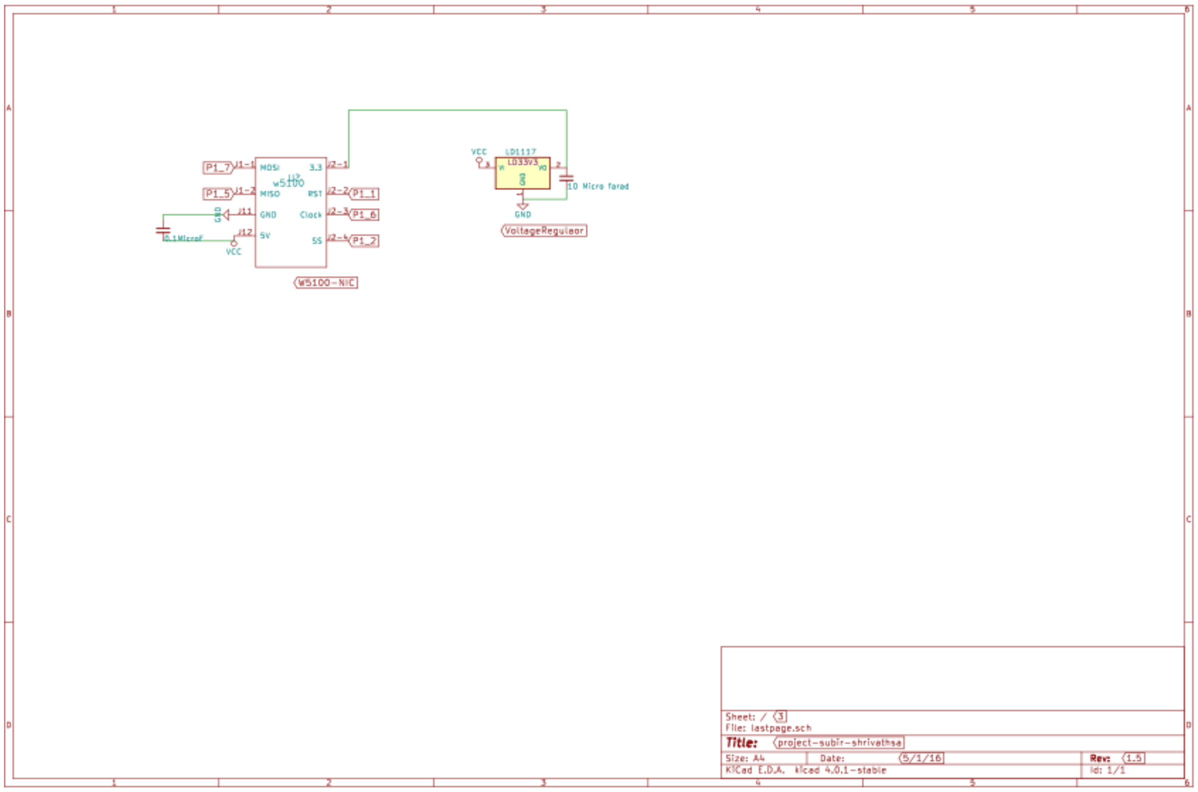
8.1 Appendix - Bill of Materials

Part Description	Source	Cost
1. WIZnet W5100 Network Module with Mag Jack - WIZ811MJ	Spark Fun www.sparkfun.com	$24.95
2. Graphic LCD	Spark Fun www.sparkfun.com	$19.95
3. Logic level converter	Spark Fun www.sparkfun.com	$2.95
4. Voltage Regulator 3.3 V	Spark Fun www.sparkfun.com	$1.95
5. Capacitor/Resistor	E-Store (CU Boulder EE Dept.)	$2.00
TOTAL		**$51.80**

8.2 Appendix – Schematics




74LS374Circuit
EEPROM
IOExpander
PCF8574A
MAX232
DB9
128x64
LCD - 0071D ROHS
GraphicLCD
Sheet: / 2
File: lab2.sch
Title: project-subir-shrivathsa
Size: A4 Date: 5/1/16 Rev: 1.5
KiCad E.D.A. kicad 4.0.1-stable Id: 1/1


W5100-NIC
w5100
LD1117
VoltageRegulator
10 Micro farad
Sheet: / 3
File: lastpage.sch
Title: project-subir-shrivathsa
Size: A4 Date: 5/1/16 Rev: 1.5
KiCad E.D.A. kicad 4.0.1-stable Id: 1/1

8.3 Appendix - Firmware Source Code

```
/*  ESD-Final Project main src file
    SUBIR KUMAR PADHEE
    SHRIVATHSA KESHAVA MURTHY
    ECEN-5613
*/

#include "main.h"
#include "w5100.h"
#include "glcd.h"

/* Globals */
unsigned char RCVD[1600] = {'\0'};


_sdcc_external_startup()
{
    TI = 0;
    RI = 0;
    SCON = 0x50;
    TMOD = 0x20;
    TH1 = 253;
    TL1 = 253; // for 9600 baud rate of the RS-232 communication
    TR1 = 1;
    AUXR |= 0x0C;

    return 0;
}

void serialInit()
{

    TMOD = 0x20;
    SCON = 0x50;

    TH1 = -3;
    TI = 0;
    RI = 0;
    TR1 = 1;
}

int main(void)
{
    delay_custom(1000);

    //serialInit();
    printf_tiny("\r\nInitializing...");
    SPI_init(); //Initialize the SPI

    /* glcd code starts here */
    glcd_init(); //Initialize the glcd
    delay_1ms();
    /* glcd code ends here */
    /* W5100 code starts */
    RESET_W5100 = 0; //Hardware reset for the w5100 network module
    delay_1ms();
    RESET_W5100 = 1;

    nic(); //call the network module. Its an infinite loop over there
    /* W5100 code ends */

    return 0;
}
```

```
/*  ESD-Final Project header – main.h
    SUBIR KUMAR PADHEE
    SHRIVATHSA KESHAVA MURTHY
    ECEN-5613
*/

#ifndef main_h
#define main_h

/* File includes */
#include <stdio.h>
#include <stdlib.h>
#include <at89c51ed2.h>
#include <mcs51reg.h>
#include <mcs51/8051.h>
#include <string.h>
#include <math.h>


/* Function Prototypes*/
unsigned char getChar1();
void putcharac(char c);
void putstring(char * str);
unsigned char getcharac();
unsigned char getstring(unsigned char * str);
unsigned char error(unsigned char c);
void delay_custom(unsigned char t);
void delay_1ms();
void delay_custom_us(unsigned int delay);

/* SPI */
void SPI_init();
//void isr_SPI(void) __interrupt (9);
unsigned char send_SPI(unsigned char value);
unsigned char read_data();

/* NIC */
//void isr_INT1(void) __interrupt (2);
void nic_init();
void ctrl_write(unsigned char addr, unsigned char data_b);
unsigned char ctrl_read(unsigned char addr);
unsigned char sd_single_read(unsigned long sector);
void readBytesSd();
unsigned char read_buf();
unsigned char read_data_nic();
unsigned char send_SPI_nic(unsigned char value);
void buffer_write(unsigned char *data_array, unsigned int length);
unsigned char buffer_read(unsigned char *data_array, unsigned int length);
void bfs(unsigned char addr, unsigned char data_b);
void bfc(unsigned char addr, unsigned char data_b);
void packets_send(unsigned char *str, unsigned int length);
unsigned char packets_recv(unsigned char *str);
void nic();
unsigned char getcharac_chat();
```

```
/* SD card */
void sdcard_init();

/* Defines */
#define time20_ms 20
#define time5_ms 5
#define time1_ms 1
#define time100_ms 100

/* SPI */
#define SS P1_4
#define RESET_W5100 P1_1

#define MISO P1_5
#define SCL P1_6
#define MOSI P1_7


/* NIC */
/* MAC ADDRESS */
#define MAC5 0xFC
#define MAC4 0x3F
#define MAC3 0xDB
#define MAC2 0x38
#define MAC1 0x3A
#define MAC0 0x6A
/* OPCODES */
#define CTRL_WR 0x02
#define CTRL_RD 0x00
#define BUF_WR 0x7A
#define BUF_RD 0x3A
#define BFS 0x04
#define BFC 0x05
#define PPCTRL_BYTE 0x00

/* ALL BANKS */
#define EIE 0x1B
#define EIR 0x1C
#define ESTAT 0x1D
#define ECON2 0x1E
#define ECON1 0x1F
/*BANK 0 */
#define EIE 0x1B
#define ERXSTL 0x08
#define ERXSTH 0x09
#define ERXNDL 0x0A
#define ERXNDH 0x0B
#define ERXRDPTL 0x0C
#define ERXRDPTH 0x0D
#define EWRPTL 0x02
#define EWRPTH 0x03
#define ERDPTL 0x00
#define ERDPTH 0x01

#define ETXSTL 0x04
#define ETXSTH 0x05
```

```
#define ETXNDL 0x06
#define ETXNDH 0x07

/*BANK 1 */
#define ERXFCON 0x18
#define EPKTCNT 0x19

/*  BANK 2*/
#define MACON1 0x00
#define MACON3 0x02
#define MACON4 0x03
#define MAMXFLL 0x0A
#define MAMXFLH 0x0B
#define MABBIPG 0x04
#define MAIPGL 0x06

/* BANK 3 */
#define MAADR1 0x04
#define MAADR2 0x05
#define MAADR3 0x02
#define MAADR4 0x03
#define MAADR5 0x00
#define MAADR6 0x01



/* Macros */
#define NOP (P1_0 ^= 1)

/* Externs*/
extern unsigned char errorFlag; /* declared as extern due to possible use in multiple files */
extern unsigned char buffer;
extern volatile unsigned char tx_over;
void delay_custom_ms(unsigned int delay);

unsigned char errorFlag = 0; //ERROR FLAG-SET  WHEN AN ERROR OCCURS

#endif // main
```

```
/*  ESD-Final Project library - W5100.c
    SUBIR KUMAR PADHEE
    SHRIVATHSA KESHAVA MURTHY
    ECEN-5613
*/
#include "main.h"
#include "w5100.h"
#include "glcd.h"

/* Globals */
__xdata unsigned int* ptr;
__xdata at 0x7000 unsigned char buffer_rx[4095];

/* Function definitions */

void nic()
{
    volatile unsigned char connection_st = 0;

    w5100_init();
    delay_1ms();
    setup_server(); // SET UP THE SERVER
    //while(!setup_server());// SET UP THE SERVER
    while(1)
    {
        connection_st = read_SPI_W5100(S0_SR);
        switch(connection_st)
        {
        case SOCK_CLOSED:
            setup_server(); // SET UP THE SERVER
            break;

        case SOCK_ESTABLISHED:
            //printf("\r\n\tESTD");
            receiveFromClient();// STORE THE PACKETS RECEIVED AND ACT ACCORDINGLY
            break;

        case SOCK_FIN_WAIT:
        case SOCK_CLOSING:
        case SOCK_TIME_WAIT:
        case SOCK_CLOSE_WAIT:
        case SOCK_LAST_ACK:
            write_SPI_W5100(S0_CR,CLOSE); // CLOSE THE SOCKET IF THE CONNECTION STATUS IS
ANYTHING BUT CLOSED OR ESTABLISHED
            while(read_SPI_W5100(S0_SR));

            break;

        default:
            //printf_tiny("\r\nSome status not handled!");
            break;
        }
    }
}

void w5100_init()
```

```
{
   write_SPI_W5100(MR, 0x80); //RESET
   delay_1ms();
   //printf("\r\nReading MR: %d\n\n",read_SPI_W5100(MR));

       /* Set Network information */
   //GATEWAY SET TO 192.168.1.0
   write_SPI_W5100(GAR0, 192);
   write_SPI_W5100((GAR0+1), 168);
   write_SPI_W5100((GAR0+2), 0);
   write_SPI_W5100((GAR0+3), 1);

   delay_1ms();

    printf("\r\nDefault Gateway: %d.%d.%d.%d\n\n",read_SPI_W5100(GAR0 + 0),read_SPI_W5100(GAR0 +
1),\
        read_SPI_W5100(GAR0 + 2),read_SPI_W5100(GAR0 + 3));

   //SOURCE MAC ADDRESS SET TO FC.3F.DB.38.3A.6A
   write_SPI_W5100((SHAR0), 0xFC);
   write_SPI_W5100((SHAR0+1), 0x3F);
   write_SPI_W5100((SHAR0+2), 0xDB);
   write_SPI_W5100((SHAR0+3), 0x38);
   write_SPI_W5100((SHAR0+4), 0x3A);
   write_SPI_W5100((SHAR0+5), 0x6A);

   delay_1ms();
   printf("\r\nMAC Address: %.2X:%.2X:%.2X:%.2X:%.2X:%.2X\n\n",read_SPI_W5100(SHAR0 +
0),read_SPI_W5100(SHAR0 + 1),\
        read_SPI_W5100(SHAR0 + 2),read_SPI_W5100(SHAR0 + 3),read_SPI_W5100(SHAR0 +
4),read_SPI_W5100(SHAR0 + 5));

   //SUBNET MASK SET TO 255.255.255.0
   write_SPI_W5100(SUBR0, 255);
   write_SPI_W5100((SUBR0+1), 255);
   write_SPI_W5100((SUBR0+2), 255);
   write_SPI_W5100((SUBR0+3), 0);

   delay_1ms();
   printf("\r\nSubnet Mask: %d.%d.%d.%d\n\n",read_SPI_W5100(SUBR0 + 0),read_SPI_W5100(SUBR0 +
1),\
        read_SPI_W5100(SUBR0 + 2),read_SPI_W5100(SUBR0 + 3));

   //SOURCE IP SET TO 192.168.0.3
   write_SPI_W5100(SIP0, 192);
   write_SPI_W5100((SIP0+1), 168);
   write_SPI_W5100((SIP0+2), 0);
   write_SPI_W5100((SIP0+3), 3);

   delay_1ms();

   printf("\r\nConfigured IP: %d.%d.%d.%d\n\n",read_SPI_W5100(SIP0 + 0),read_SPI_W5100(SIP0 + 1),\
        read_SPI_W5100(SIP0 + 2),read_SPI_W5100(SIP0 + 3));

   //MEMORY FOR EACH SOCKET - S0:8KB REST:0
   write_SPI_W5100(RMSR, 0x03);
```

```
    write_SPI_W5100(TMSR, 0x03);

    delay_1ms();

    //MODE REG
    write_SPI_W5100(IMR, 0xCF); // IP CONFLICT, DEST UNREACHABLE, ALL SOCKETS
    write_SPI_W5100((RTR0), 0x0F); // RTR SET TO 400ms
    write_SPI_W5100(RTR0+1, 0xA0); //
    write_SPI_W5100(RCR, 0x05); // RCR SET TO 5 TRIES

    printf_tiny("Initialization routine over!\n");
}

/* FUNCTION TO SETUP THE SERVER */
unsigned char setup_server()
{
    // Send Close Command
    write_SPI_W5100(S0_CR,CLOSE);
    // Waiting until the S0_CR is clear
    while(read_SPI_W5100(S0_CR) == OPEN)
    {
        read_SPI_W5100(S0_SR);
        return;
    }

    write_SPI_W5100(S0_MR, 0x01); //SOCKET 0 - ON TCP
    //printf("\r\nReading SO_MR: %02X",read_SPI_W5100(S0_MR));
    write_SPI_W5100(S0_PORT0, 0x13);//S0_PORT 5097 - 0X13E9
    write_SPI_W5100((S0_PORT0+1), 0x88);//S0_PORT 5097 - 0X13E9
    printf("\r\nPORT number for connection: 0x%X%X",read_SPI_W5100(S0_PORT0),
read_SPI_W5100(S0_PORT0+1));

    write_SPI_W5100(S0_CR, OPEN); // INITIALIZE S0 AS TCP SERVER SOCKET
    delay_1ms();

    if(read_SPI_W5100(S0_SR) != SOCK_INIT)
    {
        write_SPI_W5100(S0_CR, CLOSE);
        printf("\r\nCould not setup server.  S0_SR:%02X ",read_SPI_W5100(S0_SR));
        return 0;//FAIL
    }
    else
        printf_tiny("\r\nServer Setup!");
    write_SPI_W5100(S0_CR, LISTEN); //MAKE THE SERVER READY TO ACCEPT CONNECTIONS
    delay_1ms();
    if(read_SPI_W5100(S0_SR) != SOCK_LISTEN)
    {
        write_SPI_W5100(S0_CR, CLOSE);
        printf_tiny("\r\nCould not Listen");
        return 0;//FAIL
    }
    else
        printf_tiny("\r\nI'm listening !");

    return 1; //SUCCESSFULLY SETUP THE SERVER
}
```

```
/* FUNCTION TO SEND PACKETS TO THE CLIENT */
unsigned char sendToClient(unsigned char *buffer_tx)
{
   unsigned int buf_sz = 0, upper_size = 0, remaining_size = 0; //HOLD SIZE OF DATA TO BE SENT
   unsigned int offset_tx = 0; // OFFSET TO THE BASE OF TX BUFFER TO WHICH DATA TO BE
TRANSMITTED IS WRITTEN
   unsigned int offset_NOmask = 0; // RAW OFFSET TO THE BASE OF TX BUFFER TO WHICH DATA
TO BE TRANSMITTED IS WRITTEN
   unsigned int start_addr = 0; //ABSOLUTE ADDRESS TO WHICH DATA TO BE TRANSMITTED IS
WRITTEN
   unsigned int j = 0; // LOOP VARIABLE
   unsigned char lb = 0; //TEMPORARY VARIABLE
   unsigned int ub = 0, free_sz = 0; //TEMPORARY VARIABLE
   unsigned int i = 0; // LOOP VARIABLE

   ub = read_SPI_W5100(S0_TX_FSR0);
   delay_1ms();
   lb = read_SPI_W5100(S0_TX_FSR0+1);

   free_sz = ((ub & 0x00FF) << 8) | lb; //READ THE FREE SPACE AVAILABLE IN THE TX BUFFER
   NOP;

   if(free_sz != 0)
   {
      offset_NOmask = (((unsigned int)read_SPI_W5100(S0_TX_WR0) & 0x00FF) << 8) |
read_SPI_W5100(S0_TX_WR0 + 1);
      offset_tx = offset_NOmask & S0_TX_MASK;
      NOP;
      start_addr = S0_TX_BASE + offset_tx; //DETERMINE THE ABSOLUTE ADDRESS TO WHICH
DATA TO BE TRANSMITTED IS WRITTEN
      i=0;
      buf_sz = strlen(buffer_tx);

      if(offset_tx + buf_sz > S0_TX_MASK + 1) //IF DATA TO BE TRANSMITTED NEEDS TO BE
WRAPPED AROUND THE BUFFER
      {
         upper_size = S0_TX_MASK +1 -offset_tx;
         for(i= 0; i < upper_size; i++)
         {
            write_SPI_W5100(start_addr+i, buffer_tx[j++]);
         }
         remaining_size = buf_sz - upper_size;
         for(i= 0; i < remaining_size; i++)
         {
            write_SPI_W5100(S0_TX_BASE+i, buffer_tx[j++]);
         }
      }
      else
      {
         for(i= 0; i < buf_sz; i++)
         {
            write_SPI_W5100(start_addr+i, buffer_tx[j++]);
            delay_1ms();
         }
      }
```

```
        write_SPI_W5100(S0_TX_WR0,((offset_NOmask + buf_sz) & 0xFF00) >> 8 ); // UPDATE THE
WRITE POINTER AFTER THE WRITE
        write_SPI_W5100(S0_TX_WR0 + 1,((offset_NOmask + buf_sz) & 0x00FF));

        write_SPI_W5100(S0_CR, SEND); //INDICATE TO THE CHIP TO INITIATE TRANSMISSION
        NOP;
    }
    else
    {
        printf_tiny("\r\nCan't Send-No memory");
    }
}

unsigned char receiveFromClient()
{
    unsigned int rcvd_size = 0, upper_size = 0, remaining_size = 0; //HOLD SIZE OF DATA RECEIVED
    unsigned int offset_rx = 0; // OFFSET TO THE BASE OF RX BUFFER FROM WHICH DATA IS TO BE
READ
    unsigned int offset_NOmask = 0;
    unsigned int start_addr = 0; // ABSOLUTE ADDRESS OF THERX BUFFER FROM WHICH DATA IS
TO BE READ
    unsigned int j = 0; //LOOP VARIABLE
    unsigned char buffer_tx[100] = {'\0'}; //HOLDS DATA TO BE TRANSMITTED- INSTANT
MESSENGER FUNCTIONALITY
    volatile static unsigned char CHAT_FLAG = 0,WEB_PAGE_FLAG=0; //FLAGS FOR IM AND HTTP
SERVER APPLICATIONS
    unsigned int ub = 0, lb = 0; //TEMPORARY VARIABLES
    unsigned int i = 0; //LOOP VARIABLE

    for(i = 0; i < 100; i++)
    {
        buffer_tx[i] = '\0';
    }
    ub = read_SPI_W5100(S0_RX_RSR0);
    delay_1ms();
    lb = read_SPI_W5100(S0_RX_RSR0+1);
    rcvd_size = (ub << 8) | (lb & 0x00FF); // DETERMINE RECEIVED SIZE
    delay_1ms();
    if(rcvd_size == 0);
        printf_tiny("\r\nReceived nothing");

    if(rcvd_size != 0)
    {
        offset_NOmask = ((read_SPI_W5100(S0_RX_RD0) & 0x00FF) << 8) | read_SPI_W5100(S0_RX_RD0
+ 1);
        offset_rx = offset_NOmask & S0_RX_MASK;
        if(CHAT_FLAG == 0)
            printf("\r\noffset_rx:%04X", offset_rx);

        start_addr = S0_RX_BASE + offset_rx; // DETERMINE START ADDRESS

        if(offset_rx + rcvd_size > S0_RX_MASK + 1) // IF RECEIVED PACKETS WRAP AROUND THE
RECEIVE BUFFER
        {
            upper_size = S0_RX_MASK +1 -offset_rx;
            for(i= 0; i < upper_size; i++)
            {
               delay_1ms();
               buffer_rx[j++] = read_SPI_W5100(start_addr+i); //STORE THE DATA RECEIVED
            }
            remaining_size = rcvd_size - upper_size;
            for(i= 0; i < remaining_size; i++)
            {
               delay_1ms();
               buffer_rx[j++] = read_SPI_W5100(start_addr+i);
            }
         }
         else
         {
            for(i= 0; i < rcvd_size; i++)
            {
               delay_1ms();
               if(i%20 == 0)
                  printf("\r\n");
               buffer_rx[j++] = read_SPI_W5100(start_addr+i); //STORE THE DATA RECEIVED
               delay_1ms();
            }
         }
         buffer_rx[j] = '\0';

         write_SPI_W5100(S0_RX_RD0,((offset_NOmask + rcvd_size) & 0xFF00) >> 8 ); //UPDATE THE
READ POINTER
         write_SPI_W5100(S0_RX_RD0 + 1,((offset_NOmask + rcvd_size) & 0x00FF));

         write_SPI_W5100(S0_CR, RECV); //INDICATE TO THE CHIP THAT RECEPTION IS COMPLETE.
THE STATUS REGISTER CHANGES ITS VALUE ACCORDINGLY

         /* CHAT */
         if((buffer_rx[0] == '*') || (CHAT_FLAG == 1))
         {
            if(buffer_rx[0] == '*')
            {
               printf("\r\nGUEST:\tLET'S TALK!\r\n");
            }
            else
            {
               printf("GUEST> %s \r\n",buffer_rx);
               glcd(buffer_rx, 0); //DISPLAY RECEIVED STRING ON THE GLCD
            }

            CHAT_FLAG = 1;
         }

         else if ( (buffer_rx[1] == 'T') && (buffer_rx[2] == 'T') && (buffer_rx[3] == 'P')){
            printf_tiny("\r\nWebServer..\r\n");
            WEB_PAGE_FLAG =1;
         }
         //OPCODE
         //THE PRINTED STRINGS EXPLAIN THE CODE FLOW
         if((CHAT_FLAG == 0) && (WEB_PAGE_FLAG == 0))
         {
            printf_tiny("\r\nReceive done \r\n");
            delay_1ms();
            printf_tiny("Opcodes Received.. \r\n%s\r\n",buffer_rx);
            printf_tiny("Blank check... ");
            blankCheck();
            printf_tiny("Done\r\n");
            printf_tiny("DEBUG: .%s.\r\n",buffer_rx);
            parseTcpOpcodes(0x400,buffer_rx);
            printf_tiny("Programmed Code Memory from 0x400 to %x\r\n",0x400+rcvd_size);
            printf_tiny("\r\nDone.. Connect EA=0 to run the new code\r\n");
        }
        // CHAT
        if(CHAT_FLAG)
        {
            printf_tiny("\r\n.%s.\r\n", buffer_rx);
            if((buffer_rx[0] == 'b') && (buffer_rx[1] == 'y') && (buffer_rx[2] == 'e'))
            {
                CHAT_FLAG = 0;
                write_SPI_W5100(S0_CR,CLOSE);
                while(read_SPI_W5100(S0_SR));
                printf_tiny("\r\n Connection Closed!");
                return 1;
            }
            printf("ME>\t");
            getstring(buffer_tx);
            buffer_tx[strlen(buffer_tx)] = '\0';

            glcd(buffer_tx, 0); //DISPLAY THE TRANSMITTED STRING ONTHE GLCD
            printf("\r\n");
            sendToClient(buffer_tx);
            for(i = 0; i < 100; i++)
            {
                buffer_tx[i] = '\0';
                buffer_rx[i] = ' ';
            }
        }
        if(WEB_PAGE_FLAG)
        {
            printf_tiny("\r\n%s\r\n",buffer_rx);
            sendToClient("HTTP/1.1 200 OK\r\nCache-Control : no-cache, private\r\nContent-Length : 30\r\nDate
: Mon, 25 Apr 2016 \r\n\r\n<?php\necho \"Hi there\";\n?>\r\n\r\n");
            delay_custom_ms(1000);
            write_SPI_W5100(S0_CR,CLOSE);
            while(read_SPI_W5100(S0_SR));
            delay_custom_ms(1000);
        }
    }
}

void parseTcpOpcodes(unsigned int baseAddress, unsigned char *buffer){
    int  i=0;
    int n=0;
    unsigned char inputBuffer=0,j;
    n = strlen(buffer);

    printf_tiny("%d bytes\r\n%s\r\n",n/2,buffer);
    for ( i=0; buffer[i]!='\0';){
      // printf_tiny("DEBUG: buffer[%d]=%x\r\n",i,buffer[i]);
        inputBuffer=0;
        for(j=0;j<2;j++){
            unsigned char temp=0;
            if ( buffer[i] > ALPHA_F){
                temp = buffer[i]-0x57;
            }
            else if ( buffer[i] > NUM9){
                temp = buffer[i]-NUM7;
            }
            else{
                temp = buffer[i]-NUM0;
            }
            // convert input characters to integer
            inputBuffer = inputBuffer+(temp*powf(16,2-j-1));
            i++;
            }
    printf_tiny("Sending %x to write at %x\r\n",inputBuffer,baseAddress);
    dataout(baseAddress,inputBuffer);
    if ( i == 2)
        dataout(0x400,0x75); // backdoor method . don't do this. bug
    baseAddress++;
    }
    // we have written opcodes
    // write an infinite loop, so that it will not go further to execute
    // 80FE is the opcode
    //printf_tiny("Appended application opcodes of 0x80, 0xFE bytes in the end\r\n");

    // dataout(0x401,0x89); // backdoor method . don't do this. bug
    printf_tiny("Flashing hex file......");
    for ( i=baseAddress;i<0x8000;i++){
        dataout(baseAddress,0xFF);
        baseAddress++;
    }
    printf_tiny("Done\r\n");
    //dataout(baseAddress,0xFE);
    //baseAddress++;

    /* __asm
     mov a, 0x55
     mov dptr,#0x0200
     movc @dptr,a
     __endasm;
     */

}

 void dataout(unsigned int x, unsigned char y){
    /* __asm
     mov a, 0x55
     mov dptr,#0xCCCC
     movx @dptr,a
     __endasm; */
     // assign the address in x and value in y
    // printf("Writing %x to 0x%x\r\n",y,x);
```

```
     ptr =x;
    *ptr =y;
}

void  blankCheck(){
    unsigned int i=0;
    for ( i=0x400; i< 0x500; i++){
        dataout(i,0x0FF);
    }

}
```

```
/*  ESD-Final Project header - w5100.h
    SUBIR KUMAR PADHEE
    SHRIVATHSA KESHAVA MURTHY
    ECEN-5613
*/
#ifndef w5100_h
#define w5100_h

/* REGISTERS */
/* Common Registers*/

#define MR 0x0000 //MODE REG
#define GAR0 0x0001 //GATEWAY
#define SUBR0 0x0005 //SUBNET MASK
#define SHAR0 0x0009 //SOURCE MAC ADDRESS
#define SIP0 0x000F //SOURCE IP
#define IR 0x0015
#define IMR 0x0016
#define RTR0 0x0017
#define RCR 0x0019
#define RMSR 0x001A
#define TMSR 0x001B
#define SHAR0 0x0009

/* Socket Registers */
//SOCKET 0
#define S0_MR 0x0400
#define S0_CR 0x0401
#define S0_IR 0x0402
#define S0_SR 0x0403
#define S0_PORT0 0x0404
#define S0_DHAR0 0x0406
#define S0_DIPR0 0x040C
#define S0_DPORT0 0x0410
#define S0_TOS 0x0415
#define S0_TTL 0x0416
#define S0_TX_FSR0 0x0420 //Tx FREE SIZE
#define S0_TX_RD0 0x0422 //Tx READ POINTER
#define S0_TX_WR0 0x0424 //Tx WRITE POINTER
#define S0_RX_RSR0 0x0426 //RECEIVED SIZE
#define S0_RX_RD0 0x0428 //RX READ POINTER


#define S0_RX_BASE 0x6000
#define S0_TX_BASE 0x4000
#define S0_RX_MASK 0x1FFF
#define S0_TX_MASK 0x1FFF

//SOCKET 1
#define S1_MR 0x0400
#define S1_CR 0x0401
#define S1_IR 0x0402
#define S1_SR 0x0403
#define S1_PORT0 0x0404
#define S1_DHAR0 0x0406
#define S1_DIPR0 0x040C
```

```
#define S1_DPORT0 0x0410
#define S1_TOS 0x0415
#define S1_TTL 0x0416
#define S1_TX_FSR0 0x0420
#define S1_TX_RD0 0x0422
#define S1_TX_WR0 0x0424
#define S1_TX_RSR0 0x0426
#define S1_RX_RD0 0x0428

//SOCKET 2
#define S2_MR 0x0400
#define S2_CR 0x0401
#define S2_IR 0x0402
#define S2_SR 0x0403
#define S2_PORT0 0x0404
#define S2_DHAR0 0x0406
#define S2_DIPR0 0x040C
#define S2_DPORT0 0x0410
#define S2_TOS 0x0415
#define S2_TTL 0x0416
#define S2_TX_FSR0 0x0420
#define S2_TX_RD0 0x0422
#define S2_TX_WR0 0x0424
#define S2_TX_RSR0 0x0426
#define S2_RX_RD0 0x0428

//SOCKET 3
#define S3_MR 0x0400
#define S3_CR 0x0401
#define S3_IR 0x0402
#define S3_SR 0x0403
#define S3_PORT0 0x0404
#define S3_DHAR0 0x0406
#define S3_DIPR0 0x040C
#define S3_DPORT0 0x0410
#define S3_TOS 0x0415
#define S3_TTL 0x0416
#define S3_TX_FSR0 0x0420
#define S3_TX_RD0 0x0422
#define S3_TX_WR0 0x0424
#define S3_TX_RSR0 0x0426
#define S3_RX_RD0 0x0428

/* OPCODES */
#define OP_WR 0xF0
#define OP_RD 0x0F

/* COMMANDS  */
#define OPEN 0x01
#define LISTEN 0x02
#define CONNECT 0x04
#define DISCON 0x08
#define CLOSE 0x10
#define SEND 0x20
#define SEND_KEEP 0x22
#define RECV 0x40
```

```
/* SOCKET STATUS */
#define SOCK_CLOSED 0x00
#define SOCK_INIT 0x13
#define SOCK_LISTEN 0x14
#define SOCK_ESTABLISHED 0x17
#define SOCK_FIN_WAIT 0x18
#define SOCK_CLOSING 0x1A
#define SOCK_TIME_WAIT 0x1B
#define SOCK_CLOSE_WAIT 0x1C
#define SOCK_LAST_ACK 0x1D

/* NUMBERS AND ALPHA IN HEX*/

#define QUIT 0x1B
#define NUM0 0x30
#define NUM1 0x31
#define NUM3 0x33
#define NUM4 0x34
#define NUM7 0x37
#define NUM9 0x39

#define ALPHA_F 0x46
#define ALPHA_a 0x61
#define ALPHA_f 0x66
#define ALPHA_A 0x41

/* Function Definitions */
void w5100_init();
void write_SPI_W5100(unsigned int addr, unsigned char value);
unsigned char read_SPI_W5100(unsigned int addr);
unsigned char setup_server();
unsigned char receiveFromClient();
void parseTcpOpcodes(unsigned int baseAddress, unsigned char *buffer);
void dataout(unsigned int x, unsigned char y);
void blankCheck();
unsigned char sendToClient(unsigned char *buffer_tx);

#endif //w5100
```

```
/*  ESD-Final Project library - glcd.c
    SUBIR KUMAR PADHEE
    SHRIVATHSA KESHAVA MURTHY
    ECEN-5613
*/

#include "main.h"
#include "glcd.h"


/* FUNCTION DEFINITIONS */

void glcd_init()
{
    CS1 = 1;
    CS2 = 1;
    send_command(0x3F);
    delay_1ms();
    delay_1ms();
    delay_1ms();
    clear_glcd();

    column_select(0);
    page_select(0);
}

void glcd(unsigned char str[], unsigned char c)
{
    volatile static unsigned char p = 0;
    unsigned int i = 0, len = 0, j = 0;
    static unsigned char counter = 0;
    unsigned char record_screen[7][22] = {{'\0'}};
    unsigned char array_char[][6] =   {
                                    {0x00, 0x00, 0x00, 0x00, 0x00, 0x00},  // Code for char
                                    {0x00, 0x00, 0x06, 0x5F, 0x06, 0x00},  // Code for char !
                                    {0x00, 0x07, 0x03, 0x00, 0x07, 0x03},  // Code for char "
                                    {0x00, 0x24, 0x7E, 0x24, 0x7E, 0x24},  // Code for char #
                                    {0x00, 0x24, 0x2B, 0x6A, 0x12, 0x00},  // Code for char $
                                    {0x00, 0x63, 0x13, 0x08, 0x64, 0x63},  // Code for char %
                                    {0x00, 0x36, 0x49, 0x56, 0x20, 0x50},  // Code for char &
                                    {0x00, 0x00, 0x07, 0x03, 0x00, 0x00},  // Code for char '
                                    {0x00, 0x00, 0x3E, 0x41, 0x00, 0x00},  // Code for char (
                                    {0x00, 0x00, 0x41, 0x3E, 0x00, 0x00},  // Code for char )
                                    {0x00, 0x08, 0x3E, 0x1C, 0x3E, 0x08},  // Code for char *
                                    {0x00, 0x08, 0x08, 0x3E, 0x08, 0x08},  // Code for char +
                                    {0x00, 0x00, 0xE0, 0x60, 0x00, 0x00},  // Code for char ,
                                    {0x00, 0x08, 0x08, 0x08, 0x08, 0x08},  // Code for char -
                                    {0x00, 0x00, 0x60, 0x60, 0x00, 0x00},  // Code for char .
                                    {0x00, 0x20, 0x10, 0x08, 0x04, 0x02},  // Code for char /
                                    {0x00, 0x3E, 0x51, 0x49, 0x45, 0x3E},  // Code for char 0
                                    {0x00, 0x00, 0x42, 0x7F, 0x40, 0x00},  // Code for char 1
                                    {0x00, 0x62, 0x51, 0x49, 0x49, 0x46},  // Code for char 2
                                    {0x00, 0x22, 0x49, 0x49, 0x49, 0x36},  // Code for char 3
                                    {0x00, 0x18, 0x14, 0x12, 0x7F, 0x10},  // Code for char 4
                                    {0x00, 0x2F, 0x49, 0x49, 0x49, 0x31},  // Code for char 5
                                    {0x00, 0x3C, 0x4A, 0x49, 0x49, 0x30},  // Code for char 6
```

```
{0x00, 0x01, 0x71, 0x09, 0x05, 0x03},  // Code for char 7
{0x00, 0x36, 0x49, 0x49, 0x49, 0x36},  // Code for char 8
{0x00, 0x06, 0x49, 0x49, 0x29, 0x1E},  // Code for char 9
{0x00, 0x00, 0x6C, 0x6C, 0x00, 0x00},  // Code for char :
{0x00, 0x00, 0xEC, 0x6C, 0x00, 0x00},  // Code for char ;
{0x00, 0x08, 0x14, 0x22, 0x41, 0x00},  // Code for char <
{0x00, 0x24, 0x24, 0x24, 0x24, 0x24},  // Code for char =
{0x00, 0x00, 0x41, 0x22, 0x14, 0x08},  // Code for char >
{0x00, 0x02, 0x01, 0x59, 0x09, 0x06},  // Code for char ?
{0x00, 0x3E, 0x41, 0x5D, 0x55, 0x1E},  // Code for char @
{0x00, 0x7E, 0x11, 0x11, 0x11, 0x7E},  // Code for char A
{0x00, 0x7F, 0x49, 0x49, 0x49, 0x36},  // Code for char B
{0x00, 0x3E, 0x41, 0x41, 0x41, 0x22},  // Code for char C
{0x00, 0x7F, 0x41, 0x41, 0x41, 0x3E},  // Code for char D
{0x00, 0x7F, 0x49, 0x49, 0x49, 0x41},  // Code for char E
{0x00, 0x7F, 0x09, 0x09, 0x09, 0x01},  // Code for char F
{0x00, 0x3E, 0x41, 0x49, 0x49, 0x7A},  // Code for char G
{0x00, 0x7F, 0x08, 0x08, 0x08, 0x7F},  // Code for char H
{0x00, 0x00, 0x41, 0x7F, 0x41, 0x00},  // Code for char I
{0x00, 0x30, 0x40, 0x40, 0x40, 0x3F},  // Code for char J
{0x00, 0x7F, 0x08, 0x14, 0x22, 0x41},  // Code for char K
{0x00, 0x7F, 0x40, 0x40, 0x40, 0x40},  // Code for char L
{0x00, 0x7F, 0x02, 0x04, 0x02, 0x7F},  // Code for char M
{0x00, 0x7F, 0x02, 0x04, 0x08, 0x7F},  // Code for char N
{0x00, 0x3E, 0x41, 0x41, 0x41, 0x3E},  // Code for char O
{0x00, 0x7F, 0x09, 0x09, 0x09, 0x06},  // Code for char P
{0x00, 0x3E, 0x41, 0x51, 0x21, 0x5E},  // Code for char Q
{0x00, 0x7F, 0x09, 0x09, 0x19, 0x66},  // Code for char R
{0x00, 0x26, 0x49, 0x49, 0x49, 0x32},  // Code for char S
{0x00, 0x01, 0x01, 0x7F, 0x01, 0x01},  // Code for char T
{0x00, 0x3F, 0x40, 0x40, 0x40, 0x3F},  // Code for char U
{0x00, 0x1F, 0x20, 0x40, 0x20, 0x1F},  // Code for char V
{0x00, 0x3F, 0x40, 0x3C, 0x40, 0x3F},  // Code for char W
{0x00, 0x63, 0x14, 0x08, 0x14, 0x63},  // Code for char X
{0x00, 0x07, 0x08, 0x70, 0x08, 0x07},  // Code for char Y
{0x00, 0x71, 0x49, 0x45, 0x43, 0x00},  // Code for char Z
{0x00, 0x00, 0x7F, 0x41, 0x41, 0x00},  // Code for char [
{0x00, 0x02, 0x04, 0x08, 0x10, 0x20},  // Code for char BackSlash
{0x00, 0x00, 0x41, 0x41, 0x7F, 0x00},  // Code for char ]
{0x00, 0x04, 0x02, 0x01, 0x02, 0x04},  // Code for char ^
{0x80, 0x80, 0x80, 0x80, 0x80, 0x80},  // Code for char _
{0x00, 0x00, 0x03, 0x07, 0x00, 0x00},  // Code for char `
{0x00, 0x20, 0x54, 0x54, 0x54, 0x78},  // Code for char a
{0x00, 0x7F, 0x44, 0x44, 0x44, 0x38},  // Code for char b
{0x00, 0x38, 0x44, 0x44, 0x44, 0x28},  // Code for char c
{0x00, 0x38, 0x44, 0x44, 0x44, 0x7F},  // Code for char d
{0x00, 0x38, 0x54, 0x54, 0x54, 0x08},  // Code for char e
{0x00, 0x08, 0x7E, 0x09, 0x09, 0x00},  // Code for char f
{0x00, 0x18, 0xA4, 0xA4, 0xA4, 0x7C},  // Code for char g
{0x00, 0x7F, 0x04, 0x04, 0x78, 0x00},  // Code for char h
{0x00, 0x00, 0x00, 0x7D, 0x40, 0x00},  // Code for char i
{0x00, 0x40, 0x80, 0x84, 0x7D, 0x00},  // Code for char j
{0x00, 0x7F, 0x10, 0x28, 0x44, 0x00},  // Code for char k
{0x00, 0x00, 0x00, 0x7F, 0x40, 0x00},  // Code for char l
{0x00, 0x7C, 0x04, 0x18, 0x04, 0x78},  // Code for char m
{0x00, 0x7C, 0x04, 0x04, 0x78, 0x00},  // Code for char n
```

```
                                        {0x00, 0x38, 0x44, 0x44, 0x44, 0x38},  // Code for char o
                                        {0x00, 0xFC, 0x44, 0x44, 0x44, 0x38},  // Code for char p
                                        {0x00, 0x38, 0x44, 0x44, 0x44, 0xFC},  // Code for char q
                                        {0x00, 0x44, 0x78, 0x44, 0x04, 0x08},  // Code for char r
                                        {0x00, 0x08, 0x54, 0x54, 0x54, 0x20},  // Code for char s
                                        {0x00, 0x04, 0x3E, 0x44, 0x24, 0x00},  // Code for char t
                                        {0x00, 0x3C, 0x40, 0x20, 0x7C, 0x00},  // Code for char u
                                        {0x00, 0x1C, 0x20, 0x40, 0x20, 0x1C},  // Code for char v
                                        {0x00, 0x3C, 0x60, 0x30, 0x60, 0x3C},  // Code for char w
                                        {0x00, 0x6C, 0x10, 0x10, 0x6C, 0x00},  // Code for char x
                                        {0x00, 0x9C, 0xA0, 0x60, 0x3C, 0x00},  // Code for char y
                                        {0x00, 0x64, 0x54, 0x54, 0x4C, 0x00},  // Code for char z
                                        {0x00, 0x08, 0x3E, 0x41, 0x41, 0x00},  // Code for char {
                                        {0x00, 0x00, 0x00, 0x77, 0x00, 0x00},  // Code for char |
                                        {0x00, 0x00, 0x41, 0x41, 0x3E, 0x08},  // Code for char }
                                        {0x00, 0x02, 0x01, 0x02, 0x01, 0x00},  // Code for char ~
                                        {0x00, 0x3C, 0x26, 0x23, 0x26, 0x3C}  // Code for char ☐*/

    };

    if(p == 8)
    {
        p = 0; //WRAP AROUND THE DISPLAY BY SELECTING PAGE 0
        clear_glcd();
    }
    page_select(p++); //INCREMENT TO THE NEXT PAGE FOR DISPLAYING THE NEXT STRING
    clear_page(p-1);

    column_select(c);
    len = strlen(str);
    len -= (counter++ % 2);//TRANSMITTED AND RECEIVED STRINGS ARE FORMATTED
DIFFERNTLY. THIS TAKES CARE OF THAT

    for(i = 0; (i < len) && (i < 22); i++)
    {
        printf("\r\ni=%X, %c len:%d - %s\r",i,str[i], len, str);
        print_char(c+(i*6), array_char[str[i] -32]);
    }
}

/* DISPLAYS A SINGLE CHARACTER ON GLCD */
void print_char(unsigned char column_curr, unsigned char *array_char)
{
    unsigned char i = 0, old = 0;
    unsigned char current_c = 0;

    current_c = column_curr;

    for(i = 0; i < 6; i++)
    {
        column_select(current_c+i);
        fill_column(current_c+i, array_char[i]);
    }
}

/* SEND A COMMAND TO THE GLCD  CONTROLLER */
```

```
void send_command(unsigned char command)
{
   P0 = command;
   RWbar = 0;
   RS = 0;
   E = 0;
   delay_custom(300);
   E = 1;
   delay_custom(300);
   E = 0;
}

/* SEND DATA TO FILL ONE COLUMN- 8 PIXELS */
void fill_column(unsigned char column_curr, unsigned char data_b)
{
   //static unsigned char flag = 1;

   if(column_curr < 64)
   {
      CS1 = 1;
      CS2 = 0;
   }
   else
   {
      CS1 = 0;
      CS2 = 1;
   }

   P0 = data_b;
   RWbar=0;
   RS=1;
   E=0;
   delay_custom(300);
   E = 1;
   delay_custom(300);
   E = 0;
}

/* SELECTS THE DESIRED PAGE ON GLCD */
void page_select(unsigned char page)
{
   unsigned char inst_byte = 0xB8;

   inst_byte |= page;
   P0 = inst_byte;
   CS1 = 1;
   CS2 = 1;
   RS = 0;
   RWbar = 0;

   E=1;
   NOP;
   E=0;
}

/* SELECTS THE DESIRED COLUMN ON GLCD */
```

```
void column_select(unsigned char column)
{
   unsigned char inst_byte = 0x40;

   inst_byte |= column;


   if(column < 64)
   {
      CS1 = 1;
      CS2 = 0;
   }
   else
   {
      CS1 = 0;
      CS2 = 1;
   }
   P0 = inst_byte;

   RS = 0;
   RWbar = 0;

   E=1;
   NOP;
   E=0;
}

/* CLEARS THE ENTIRE GLCD */
void clear_glcd()
{
   unsigned char i = 0, j = 0;

   for(i = 0; i < 8; i++)
   {
      page_select(i);
      column_select(0);
      for(j = 0; j < 128; j++)
      {
         fill_column(j, 0x00);
         delay_1ms();
      }
   }
}

/* CLEARS A PAGE ON THE GLCD */
void clear_page(unsigned char page)
{
   unsigned char i = 0, j = 0;

   page_select(page);
   column_select(0);
   for(j = 0; j < 128; j++)
      fill_column(j, 0x00);
}
```

```
/*  ESD-Final Project header - glcd.h
    SUBIR KUMAR PADHEE
    SHRIVATHSA KESHAVA MURTHY
    ECEN-5613
*/

#ifndef glcd_h
#define glcd_h

#define RS P1_3
#define RWbar P1_2
#define BF P0_7
#define CS2 P3_4//T0
#define CS1 P3_5 //T1
#define E P3_2


void glcd(unsigned char *str, unsigned char c);
void glcd_init();
void page_select(unsigned char page);
void column_select(unsigned char column);
void fill_column(unsigned char column_curr, unsigned char data_b);
void print_char(unsigned char column_curr, unsigned char *arrary_char);
void send_command(unsigned char command);
void delay(unsigned int delay_time);
void clear_glcd();
void clear_page(unsigned char page);







#endif //glcd.h
```

```
/*  ESD-Final Project library
    SUBIR KUMAR PADHEE
    SHRIVATHSA KESHAVA MURTHY
    ECEN-5613
*/

#include "main.h"
#include "w5100.h"
/* Globals */
unsigned char buffer = 0;
volatile unsigned char tx_over = 0;


/* Function definitions */

/* the error() function prints the error messages based on the error code it is called with. It then sets the error
flag */
unsigned char error(unsigned char c)
{
    switch(c)
    {
    case 1:
        putstring("\r\n Some Error!");
        errorFlag = 1;
        break;

        default:
            putstring("\r\nUnforeseen error");
            errorFlag = 1;
    }
    return errorFlag;
}

/*  delay function */
void delay_custom(unsigned char t)
{
    unsigned char i, j;
    for(i = 0; i <t; i++)
        for(j= 0; j < 5; j++);
}

/*  1 ms delay */
void delay_1ms()
{
    unsigned char i, j;
    for(i = 0; i <30; i++)
        for(j= 0; j < 12; j++);
}

/*  1 us delay */
void delay_custom_us(unsigned int delay)
{
    unsigned int i = 0;
    for(i = 0; i < delay; i++)
        NOP;
}
```

```
/*   ms delay */
void delay_custom_ms(unsigned int delay)
{
    unsigned int i = 0;
    for(i = 0; i < delay; i++)
        delay_1ms();
}


/* Initialize SPI */
void SPI_init()
{
    SPCON = 0x74; //Clear- set clk to 000 mode, CPOL-CPHA to 00
    SPCON |= 0x10; //Set as Master
    SPCON |= 0x20; //Set SSDIS
    SPCON |= 0x40; //Enable SPI
}

/* SEND VALUE TO SLAVE ON SPI */
unsigned char send_SPI(unsigned char value)
{
    unsigned char i = 0;
    SS = 0;
    printf("\r\nSending.. %02X, ",value);

    for(i = 0; i < 8; i++)
    {
        if (value & 0x80)
            MOSI = 1;
        else
            MOSI = 0;
        SCL = 1;
        SCL = 0;
        value <<= 1;
    }
    return SS; //return 1 on success
}

/* READ VALUE FROM SLAVE ON SPI */
unsigned char read_data()
{
    unsigned char i = 0;
    for(i = 0; i < 8; i++)
    {
        SCL = 1;
        buffer <<= 1;
        SCL = 1;
        buffer |= MISO;
        SCL = 0;
        SCL = 0;
    }
    printf("Received :%02X.\r\n",buffer);
    return buffer;
}
```

```
/* SEND VALUE TO SLAVE ON SPI - CUSTOMIZED FOR THE NIC*/
unsigned char send_SPI_nic(unsigned char value)
{
   SS = 0;
   printf("\r\nSending.. %02X, ",value);

   SPDAT = value;
   while(SPSTA != 0x80);
   SPSTA &= 0x7F;

   return SS; //return 1 on success
}

/* READ VALUE FROM SLAVE ON SPI - CUSTOMIZED FOR THE NIC */
unsigned char read_data_nic()
{
   SS = 0;
   while(SPSTA != 0x80);
   buffer = SPDAT;
   SPSTA &= 0x7F;
   printf("Received :%02X.\r\n",buffer);
   return buffer;
}

/* SEND VALUE TO SLAVE ON SPI - CUSTOMIZED FOR THE W5100*/
void write_SPI_W5100(unsigned int addr, unsigned char value)
{
   SS = 0;
   //printf("\r\nSending %02X to %04X ",value, addr);

   SPDAT = OP_WR;
   while(SPSTA != 0x80);
   SPSTA &= 0x7F;

   SPDAT = ((addr & 0xFF00) >> 8);
   while(SPSTA != 0x80);
   SPSTA &= 0x7F;

   SPDAT = (addr & 0x00FF);
   while(SPSTA != 0x80);
   SPSTA &= 0x7F;

   SPDAT = value;
   while(SPSTA != 0x80);
   SPSTA &= 0x7F;

   SS = 1;
}

/* READ VALUE FROM SLAVE ON SPI - CUSTOMIZED FOR THE W5100 */
unsigned char read_SPI_W5100(unsigned int addr)
{
   SS = 0;

   SPDAT = OP_RD;
   while(SPSTA != 0x80);
```

```
    SPSTA &= 0x7F;

    SPDAT = ((addr & 0xFF00) >> 8);
    while(SPSTA != 0x80);
    SPSTA &= 0x7F;

    SPDAT = (addr & 0x00FF);
    while(SPSTA != 0x80);
    SPSTA &= 0x7F;

    SPDAT = 0x00;
    while(SPSTA != 0x80);
    buffer = SPDAT;
    SPSTA &= 0x7F;

    SS = 1;

    return buffer;
}
```

8.4 Appendix - Software Source Code

```
#	Filename	:sendFile.txt
#	Author		: Subir Kumar Padhee, Shrivathsa Murthy
#	Date		: 20/4/16
#	Objective	: script file which transfers hex file from client PC to arduino Yun over Wifi using
scp

#	 bug reports? write to shmu7023@colorado.edu, supa2799@colorado.edu



#winscp.ex# Connect
open scp://root:password@192.168.0.5/
# Change remote directory
cd /www/sd/esd/
# Force binary mode transfer
option transfer binary
# Download file to the local directory d:\
#get client.c d:\
#upload a file remote directory
put c:\Keil_v5\C51\Examples\Objects\testUpload.hex /www/sd/esd/

#system calls to configure network settings
call ifconfig eth1 192.168.0.5 netmask 255.255.255.0
call route add default gw 192.168.0.5

#execute client program
call ./client
# Disconnect session =
close

#this will terminate windows prompt
&& exit
```

```
/*  ESD-Final Project library - client.c
    SUBIR KUMAR PADHEE
    SHRIVATHSA KESHAVA MURTHY
    ECEN-5613
*/
/*Author                    : Subir Kumar Padhee, Shrivathsa Murthy
  Date                      : 20/4/16
  Objective         : client Arduino Yun program to parse Intel Hex file and create
                                  sockets to communicate with the 8051

  bug reports? write to shmu7023@colorado.edu, supa2799@colorado.edu

*/


#include <sys/types.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <signal.h>
#include <stdbool.h>
#include <math.h>

#define NUM0 0x30
#define NUM7 0x37
#define NUM9 0x39

#define ALPHA_F 0x46

/*   Routine Description:
//   Function name : myAtoi - converts 2 unsigned chars into an integer
//   Arguments       : char * - 2 hex digits
//   Return            : int - converted integer value
*/

int
myAtoi(char *inputBufferArray){
  int inputBuffer=0,i;

  // loop through 2 times as there are 2 chars
  for(i=0;i<2;i++){
    unsigned char temp;
    if ( inputBufferArray[i] > ALPHA_F){
      temp = inputBufferArray[i]-0x57;
      }
    else if ( inputBufferArray[i] > NUM9){
      temp = inputBufferArray[i]-NUM7;
      }
    else{
      temp = inputBufferArray[i]-NUM0;
      }

    // convert input characters to integer
    inputBuffer = inputBuffer+(temp*pow(16,2-i-1));
    }
    return inputBuffer;
}



/* Routine Description: main function

 * Parses hex file and Connects to HTPP remote server (192.168.0.3)on port 5000 to send the contents
 * Implements Chat client server application as well
 *
*/

int
main(int argc , char **argv )
{
  FILE *filePointer;

  char setting[25];
  char value[100],value1[100];
  char portNum[5][6];
  int i=0;
  char *returnValue;
  struct sockaddr_in sin;
  struct hostent *hp;
  int s;
  char buffer[200]="\0",temp[200]="\0";
  int length,opcodeLength=0;
  char opcodes[10000]="\0";
  char lengthChar[3];


  if (  argc == 1){
              // if no args, it is programmer mode
              // read the file in the PWD
              filePointer = fopen("upload.hex","r");
        if ( filePointer == 0){
                perror("ERROR in Opening file! Input file doesn't exist");
                exit(1);
        }

              lengthChar[2]='\0';
        while(fgets(buffer,sizeof(buffer),filePointer)) {
        //printf("Entering while\n");
        if(strlen(buffer) <= 1 )                              /* if it's an empty line, continue */
                continue;

        printf("%s",buffer);

        lengthChar[0]=buffer[1]; lengthChar[1]=buffer[2];
        length = myAtoi(lengthChar); // convert to hex
        if ( length == 0){
                continue;
        }
        strncpy(temp,&buffer[9],length*2);
        temp[2*length]='\0';
        printf("%s\n",temp);
        strcat(opcodes,temp);
        opcodeLength += length;

        }
        opcodes[strlen(opcodes)]='\0';
        printf("***********Parsed opcodes = %d bytes*******\n%s\n",opcodeLength,opcodes);

 }
// we have the parsed file at this point
// convert destn ip to an address

hp = gethostbyname("192.168.0.3");
if( !hp){
  fprintf(stderr, "Unknown Host : \n");
  exit(1);
  }

//create address
bzero( (char*)&sin, sizeof(sin) );
sin.sin_family = AF_INET;
bcopy(hp->h_addr,(char*)&sin.sin_addr, hp->h_length );

// we have address at this point

  if ((sin.sin_port=htons((unsigned short)atoi("5000" ))) == 0){
    printf("can't get  port number\n");
    exit(1);
    }

// create a new socket
if ( ( s = socket( AF_INET, SOCK_STREAM, 0 )) <0 ){
    printf("Error creating a socket\n");
    exit(1);
    }

// connect socket to 8051
if (  connect( s, (struct sockaddr *)&sin, sizeof(sin) ) < 0 ){
    printf("Server listening on Port number is dead\n");
    exit(1);
    }

printf("Connected to 8051 server\n");
// argc = 1 means no external command line args. i.e. programmer mode
// if that's the case, send the opcodes to server , close socket and return
if ( argc == 1 ){
   opcodes[9999] = '\0';
   printf("Sent this \n%s\n",opcodes);
   send(s,opcodes,strlen(opcodes)+1,0);
   close(s);
   return 0;
   }
 // If it comes here, it's a
 // chat server

 printf("\nEntering 8051 TCP chat client\n\nMe:        LET'S TALK\n");

 // Send '*' so that 8051 identifies we are in CHAT mode

 send(s,"*",2,0);

 char sendBuf[100];
 char buff[200];
 int n=0;
 while(1){
         // empty it to be on the safer side
         strcpy(sendBuf,"");
         strcpy(buff,"");

         // receive response from server

         n=recv(s,buff,sizeof buff, 0);
         buff[n]='\0';
         printf("\n8051    :            %s\n",buff);

         // if message == "bye" close session
         if(!strncmp(buff,"bye",3)){
                 printf("\n            Exit from chat client\n");
                 break;
         }
         printf("\nMe      :            ");
         // o else, get input from user here and send it to 8051. this process continues

         fgets ( sendBuf, sizeof(sendBuf), stdin );
         sendBuf[99] = '\0';
         printf("\n");
         send(s,sendBuf,strlen(sendBuf)+1,0);
         //printf("Waiting..");
         if ( !strncmp(sendBuf,"bye",3)){
           printf("\n      Exit from chat client\n");
       break;
       }
         }

  close(s);
  return 0;

}
```

```
/*  ESD-Final Project library - testUpload.asm
    SUBIR KUMAR PADHEE
    SHRIVATHSA KESHAVA MURTHY
    ECEN-5613
*/
/*Author                    : Subir Kumar Padhee, Shrivathsa Murthy
  Date                      : 20/4/16
  Objective         : asm code to print a string "BBBLO" to terminal

  bug reports? write to shmu7023@colorado.edu, supa2799@colorado.edu

*/
// We start from 0x400
ORG 400H
MOV TMOD,#20H
MOV TH1,#-3
MOV SCON,#50H
SETB TR1

// write 5 characters
CLR TI
MOV SBUF,#'B'
HERE1:
JNB TI,HERE1

CLR TI
MOV SBUF,#'B'
HERE2:
JNB TI,HERE2

CLR TI
MOV SBUF,#'B'
HERE3:
JNB TI,HERE3

CLR TI
MOV SBUF,#'L'
HERE4:
JNB TI,HERE4

CLR TI
MOV SBUF,#'0'
HERE5:
JNB TI,HERE5
END
```

```
/*  LAB-4 library source file - TERMINAL
    SUBIR KUMAR PADHEE
    ECEN5613
*/

#include "main.h"

/* Function definitions */

/* putcharac prints a single character on the terminal */
void putcharac(char c)
{
    SBUF = c;
    while(TI == 0);
    TI = 0;
}
/* used bu printf() */
void putchar( char c)
{
    SBUF = c;
    while(!TI);
    TI = 0;
}

/* putstring prints a string on the terminal */
void putstring(char * str)
{
    while(*str)
    {
        putcharac(*str++);
    }
}

/* getcharac reads a single character from the terminal */
unsigned char getcharac()
{
    while(RI == 0);
    RI = 0;
    putcharac(SBUF);
    return SBUF;
}

/* getcharac reads a single character from the terminal */
unsigned char getcharac_chat()
{
    while(RI == 0);
    RI = 0;
    putcharac(SBUF);
    return SBUF;
}

/*unsigned char getChar1(){
    while(!RI);
    RI=0;
   // putchar(SBUF);
    return SBUF;
}*/

/* getstring reads a string from the terminal - customized for the Instant Messenger Application */
unsigned char getstring(unsigned char * str)
{
   unsigned char i = 0;
   do
   {
      *str = getcharac_chat();
      str++;
      i++;
      if(i == 99)
      {
         //error(7);
         break;//return 0;
      }
   }while(*(str-1) != '\r');
   *(str-1) = '\0';

   return 1;
}
```

```
/*  ESD-Final Project library - NIC ENC28J60
    SUBIR KUMAR PADHEE
    SHRIVATHSA KESHAVA MURTHY
    ECEN-5613
*/


#include "main.h"
/* Globals */
static unsigned int RXreadPtr = 0;

/* Function definitions */


void nic_init()
{
    SS = 0;
    printf_tiny("\r\nIn NIC init");

    send_SPI_nic(0xFF); // to reset the chip
    delay_1ms();

    ctrl_write(ECON1, 0x04); //select bank 0
    //ctrl_write(EIE, 0X88); //Enable interrupt
    //ctrl_read(ECON1);

    ctrl_write(ERXSTL, 0x00); //ERXSTL = 0x00;
    //ctrl_read(ERXSTL);
    ctrl_write(ERXSTH, 0x00); //ERXSTH = 0x00;
    //ctrl_read(ERXSTH);
    RXreadPtr = 0x0000;

    //getcharac();
    ctrl_write(ERXNDL, 0x19); //ERXNDL = 0x19;
    ctrl_write(ERXNDH, 0xAD); //ERXNDH = 0xAD;

    ctrl_write(ERXRDPTL, 0x00); //ERXRDPTL = 0x00; - check
    ctrl_write(ERXRDPTH, 0x00); //ERXRDPTH = 0x00; - check

    ctrl_write(ETXSTL, 0x19); //ETXSTL = 0x19;
    ctrl_write(ETXSTH, 0xAE); //ETXSTH = 0xAE;

    ctrl_write(ETXNDL, 0x1F); //ETXNDL = 0x1F;
    ctrl_write(ETXNDH, 0xFF); //ETXNDH = 0xFF;

    ctrl_write(EWRPTL, 0x19); //ETXSTL = 0x19;
    ctrl_write(EWRPTH, 0xAE); //ETXSTH = 0xAE;

    ctrl_write(ECON1, 0x05); //select bank 1
    ctrl_write(ERXFCON, 0x00);// CLEAR ERXFCON-ALLOW ALL FRAMES

    //delay_1ms();

    ctrl_write(ECON1, 0x06); //select bank 2
    //ctrl_write(MACON1, 0x0C);// SET TXPAUS, RXPAUS
    ctrl_write(MACON1, 0x01);// SET MARXEN
    ctrl_write(MACON3, 0xB5);// 1 0 1 1 0 1 0 1 AUTO 64/60 PADDING;APPEND CRC;NO PROP
HEADER;HUGE FR;FR LEN NOT COMP;FULL DUP

    ctrl_write(MACON4, 0x40); // 0 1 0 0 0 0 0 0 - SET DEFER BIT

    ctrl_write(MAMXFLL, 0x05);
    ctrl_write(MAMXFLH, 0xEE); // MAX FRAME LENGTH 1518 BYTES

    ctrl_write(MABBIPG, 0x15); //B2B INTER PKT GAP FOR FULL DUPLEX

    //ctrl_write(MAIPGL, 0x12);

    ctrl_write(MAADR1, MAC5); // SET MAC ADDRESS
    ctrl_write(MAADR2, MAC4); // SET MAC ADDRESS
    ctrl_write(MAADR3, MAC3); // SET MAC ADDRESS
    ctrl_write(MAADR4, MAC2); // SET MAC ADDRESS
    ctrl_write(MAADR5, MAC1); // SET MAC ADDRESS
    ctrl_write(MAADR6, MAC0); // SET MAC ADDRESS

    ctrl_write(ECON1, 0x04); //ENABLE RX
}

void ctrl_write(unsigned char addr, unsigned char data_b)
{
    addr |= (CTRL_WR << 5);
    printf("\r\nADDR: %02X DATA:%02X", addr, data_b);
    //send_SPI(addr);
    //send_SPI(data_b);
    SS = 0;
    send_SPI_nic(addr);
    send_SPI_nic(data_b);
    SS = 1;
}

unsigned char ctrl_read(unsigned char addr)
{
    addr |= (CTRL_RD << 5);

    SS = 0;
    send_SPI_nic(addr);
    send_SPI_nic(0x00);
    send_SPI_nic(0x00);
    read_data_nic();
    SS = 1;

    return buffer;
}


void bfs(unsigned char addr, unsigned char data_b)
{
    addr |= (BFS << 5);
    printf("\r\nADDR: %02X DATA:%02X", addr, data_b);
    //send_SPI(addr);
    //send_SPI(data_b);
    SS = 0;
    send_SPI_nic(addr);
    send_SPI_nic(data_b);
    SS = 1;
}
void bfc(unsigned char addr, unsigned char data_b)
{
    addr |= (BFC << 5);
    printf("\r\nADDR: %02X DATA:%02X", addr, data_b);
    //send_SPI(addr);
    //send_SPI(data_b);
    SS = 0;
    send_SPI_nic(addr);
    send_SPI_nic(data_b);
    SS = 1;
}

void buffer_write(unsigned char *data_array, unsigned int length)
{
    unsigned int i = 0;

    send_SPI_nic(BUF_WR);

    for(i = 0; i < length; i++)
    {
        send_SPI_nic(data_array[i]);
    }

    SS = 1;
}

unsigned char buffer_read(unsigned char *data_array, unsigned int length)
{
    unsigned int i = 0;

    send_SPI_nic(BUF_RD);
    for(i = 0; i < length; i++)
    {
        send_SPI_nic(0x00);
        send_SPI_nic(0x00);
        data_array[i] = read_data_nic();
    }

    return 1;
}

void packets_send(unsigned char *str, unsigned int length)
{
    unsigned char ppc_byte = PPCTRL_BYTE;

    buffer_write(&ppc_byte, 1);
    buffer_write(str, length);
    ctrl_write(ETXNDL, (0x19 + length +1));
    ctrl_write(ETXNDH, (0xAE + length +1));
    bfs(ECON1,0x80);
    bfc(ECON1,0x80);
    bfc(EIR, 0x03);
    bfs(ECON1, 0x08);//ENABLE TX
    printf("\r\n EIR-%02X", ctrl_read(EIR));
    /*printf("\r\n ECON1-%02X", ctrl_read(ECON1));
    printf("\r\n ECON2-%02X", ctrl_read(ECON2));*/

    while(ctrl_read(EIR) & 0x08 != 0x08);
    if((ctrl_read(ESTAT) & 0x02) == 0x02)
        printf("\r\nTx Aborted!");
    else
        printf("\r\nTx Done!");
    printf("\r\nETAT:%02X", ctrl_read(ESTAT));
    bfc(ECON1, 0x08);//DISABLE TX
    //printf("\r\nPhysical status reg %02X %02X", ((PHSTAT2>>8)&0XFF), ((PHSTAT2)&0XFF));
}
unsigned char packets_recv(unsigned char *str)
{
    unsigned int length = 0, i = 0;

    bfs(ECON1, 0x01);// SELECT BANK 1
    if((length = ctrl_read(EPKTCNT)) == 0)
    {
        printf("\r\nNothing received");
        return 1;
    }
    ctrl_write(ERDPTL, (RXreadPtr & 0xFF)); //POINT TO THE LOCATION TO READ FROM;
    ctrl_write(ERDPTH, ((RXreadPtr >> 8) & 0xFF)); //POINT TO THE LOCATION TO READ FROM;

    //Read packets
    buffer_read(str, length);
    //Read length
    //buffer_read(str, length);

    printf("\r\nNumber of pkts: %d", length);

    for(i = 0; i < length; i++)
    {
        if(i%16 == 0)
            printf_tiny("\r\n");
        printf("%02X ", str[i]);
    }

    return 1;
}
```

```
/*  ESD-Final Project library - SD CARD
   SUBIR KUMAR PADHEE
   SHRIVATHSA KESHAVA MURTHY
   ECEN-5613
*/

#include "main.h"


/* Function definitions */
typedef union           //Union to send address bytes one by one
{
   unsigned char b[4];
   unsigned long ul;
} b_ul;

b_ul fsz;

void sdcard_init()
{
   unsigned char i = 0;
   //CMD0
   for(i = 0; i<11 ;i++)
   {
      send_SPI(0xFF);
   }

   send_SPI(0x40);
   send_SPI(0x00);
   send_SPI(0x00);
   send_SPI(0x00);
   send_SPI(0x00);
   send_SPI(0x95);
   send_SPI(0xFF);
   read_data();

   //CMD1
   SS =1;
      send_SPI(0xFF);
      delay_1ms();
      SS=0;
      send_SPI(0x48);
      send_SPI(0x00);
      send_SPI(0x00);
      send_SPI(0x01);
      send_SPI(0xAA);
      send_SPI(0x87);

   do
   {
/*
      send_SPI(0x41);
      send_SPI(0x00);
      send_SPI(0x00);
      send_SPI(0x00);
      send_SPI(0x00);
```

```
        send_SPI(0xFF);


        send_SPI(0x69);
        send_SPI(0x00);
        send_SPI(0x00);
        send_SPI(0x00);
        send_SPI(0x00);
        send_SPI(0xFF);

        send_SPI(0x7A);
        send_SPI(0x00);
        send_SPI(0x00);
        send_SPI(0x00);
        send_SPI(0x00);
        send_SPI(0xFF);
*/



        send_SPI(0xFF);
        printf_tiny("\r\n****************");
        read_data();
    }while(buffer != 0x00);
    // OCR

      send_SPI(0x7A);
        send_SPI(0x00);
        send_SPI(0x00);
        send_SPI(0x00);
        send_SPI(0x00);
        send_SPI(0xFF);

    for(i = 0; i< 5;i++)
    {
        send_SPI(0xFF);
        printf_tiny("\r\n****************");
        read_data();
      //  delay_1ms();
    }



}

unsigned char sd_single_read(unsigned long sector)
{
    int counter,i,ctr0,ctr1,ctr2,ctr3;
    xdata unsigned char *buff;

    char save_data;
    b_ul temp1;

    temp1.ul = sector;

    //CMD 17 Single Data block read
    send_SPI(0xFF);          // Dummy byte & Wait for end of transmission
    send_SPI(0XFF);

    counter = 3;             // Argument 4 bytes
    while(counter >=0)
    {
         send_SPI(temp1.b[counter]);
         counter--;

    }

    for(i=0;i<6;i++)
    {
       send_SPI(0xFF);
       //save_data = serial_data;
    }

    ctr0=(temp1.b[1] & 0xff)-1;
    ctr1= (temp1.b[1] & 0xff);
    ctr2= (temp1.b[2] & 0xff);
    ctr3= (temp1.b[3] & 0xff);

    // Print address
    printf("\n\r%02x%02x%02x%02x: \n\r",(temp1.b[3] & 0xff),(temp1.b[2] & 0xff),(temp1.b[1] &
0xff),(temp1.b[0] & 0xff));

    for(i=0;i<512;i++)
    {
       send_SPI(0xFF);
       save_data = read_data();

       delay_1ms();

       buff[i] = save_data;

       ctr0++;
       if(i == 256)
       {
          ctr1++;
          ctr0=0;
       }
       if(((i%16)==0) && (i!=0))
       {
         putchar('\n');
         putchar('\r');

         printf("0x%02x  ",read_data());
       }
       else
       {
          printf("0x%02x  ",read_data());
       }

    }
    for(i=0;i<4;i++)
    {
        send_SPI(0xFF);            // Dummy byte & Wait for end of transmission
        //save_data = serial_data;
    }
    return 1;
}


void readBytesSd()
{
    unsigned int i;
    send_SPI(0x51);
    for ( i=0; i<4;i++)
        send_SPI(0x00);
    send_SPI(0xFF);
    //CMD1
    SS =1;
    send_SPI(0xFF);
    delay_1ms();
    SS=0;

    for ( i=0; i < 514;i++)
    {
        send_SPI(0xFF);
        read_data();
    }

}
```

8.5 Appendix - Data Sheets and Application Notes

1. https://www.sparkfun.com/datasheets/DevTools/Arduino/W5100_Datasheet_v1_1_6.pdf
2. https://www.sparkfun.com/datasheets/LCD/GDM12864H.pdf