EEL 4924 Electrical Engineering Design
(Senior Design)

Final Report

21 April 2009

Project Title: Initech V7.017b Lab Entry System

Team Name: Capacitive Inductance

Team Members:

Name: Jesse Black
Email: monsterh@ufl.edu
Phone: (772) 538-0643

Name: Benjamin Chai
Email: la3z3nt@ufl.edu
Phone: (407) 716-2998

Project Abstract:

The goal of this project is to design and build a system to allow access to selected laboratory rooms by authorized persons using their UF ID card. The system will be divided into two major components, a PC to manage the lists of people allowed entry, and the card reader, microprocessor and adjoining components. The biggest challenge will be communicating between the network server and the magnetic card reader/ microprocessor unit. Also we intend to make the user interface with card reader as user friendly as possible with features such as a LCD display and LED's to show red or green depending if the ID was valid, and possibly a piezoelectric speaker to emit a sound if the card swipe is successful.

Table of Contents:

List of Tables and Figures

Project Features:

The door entry system will be installed on any door that requires restricted access and replace all of the keypads currently controlling entry into rooms. The main features include:

- A magnetic strip card reader to read UF IDs
- An electronic door strike that unlocks when a card on the list is swiped
- Communication via network between card reader and server PC.
- A master list to allow access in case of network failure
- Server Software to both manage access lists and confirm access on card swipes
- Log files to record ID, time, and location of card swipes granted access and denied.

Technical Objectives:

The final design will eventually be wired and powered from the wall, so there are no power restrictions on the design, except for the 12VDC required to power the locking mechanism. The format the digital information is stored on the user's UF ID needs to be determined. The communication between the reader and server will be very small and over the building's LAN so bandwidth is also not a limiting factor. The only specification is that the door must be unlocked within a reasonable amount of time of a card swipe, and the door will only open in the event of a valid student ID number.

The Competition:

A programmable card reader that can control a single electric door strike or lock costs approximately $10-$20. A mountable system with the door strike built in can cost up to $1000 and more. Both of these systems are independently programmed devices that must be individually reprogrammed when access needs to be updated. In this sense it is not any better than the original keypad on each door. Other card reader systems are used to communicate with a server on a network such as the card readers on the campus vending machines. The ID is read and the machine communicates with the database to determine how much money is left and to deduct the appropriate amount if a purchase is made. Our system works in a similar way but instead of selling snacks, the information is used to determine whether or not to unlock the door.

Technical Concepts:

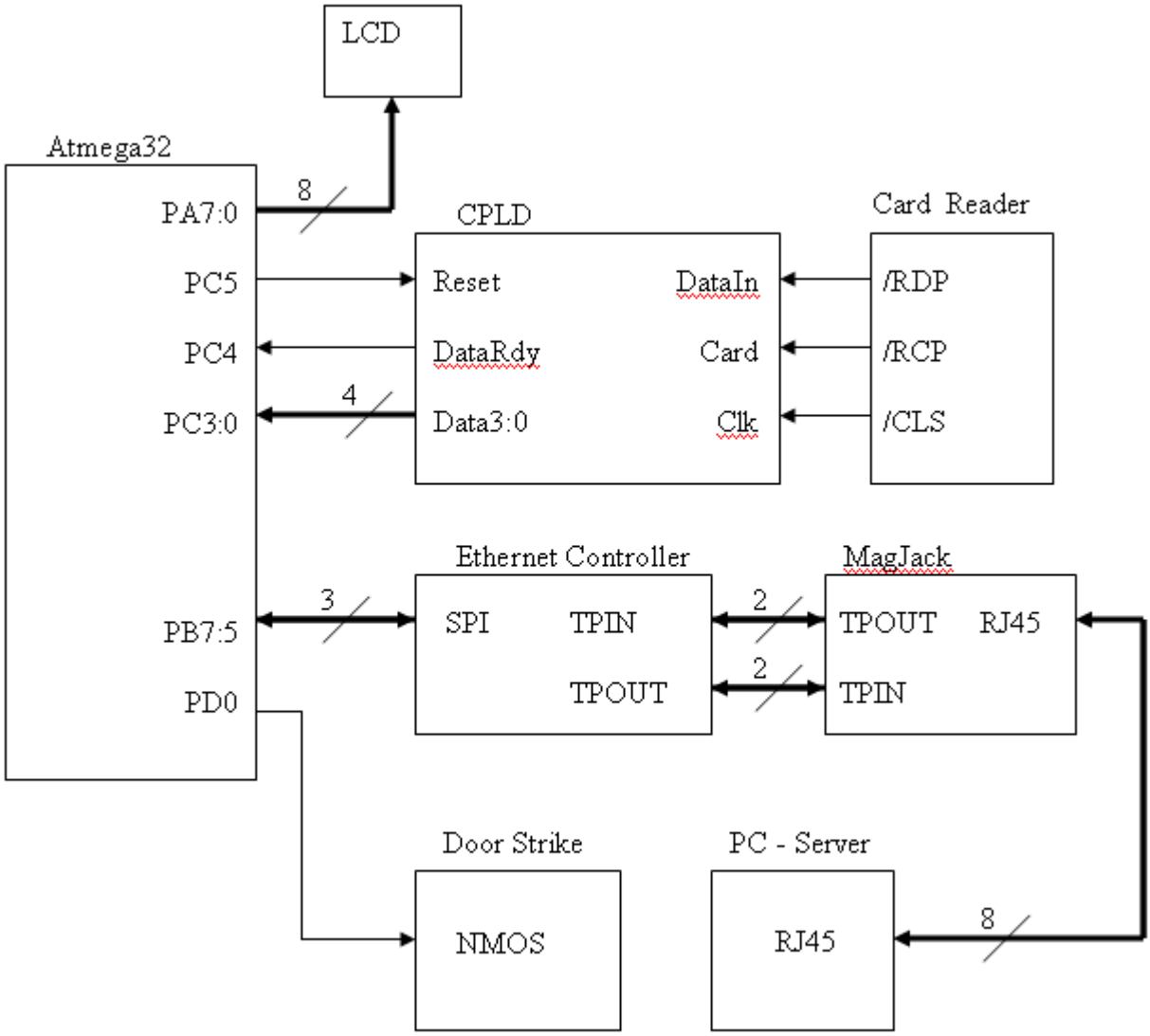
The specification of the project was to implement communication between card readers and the server via Ethernet. This was the biggest challenge to overcome. Each card reader must transmit to the server its ID (to know which door is being accessed), and the ID read from the card. The server must then reply whether access is granted or not. To simplify things, an Ethernet Controller Chip or Board can be used to interface between the microprocessor controlling the card reader and all network communication. The network communication for the server will have to be handled by software. The rest of the parts are relatively simple to use. The door strike (example pictured below) requires only a simple on off signal we can control with a power supply and a transistor. The card reader (example pictured below) communicates with an SPI interface that the ATMEL supports, however because the data shifted out is in 5 bit "bytes" (4 data 1 parity) the built in SPI with the ATMEL (reading 8 bits at a time) would be inconvenient to use. Instead a CPLD is used to shift data in from the card reader and out in parallel (4 bits at a time) and assert a signal when data is ready to be read. The operation for the microprocessor controlling the card reader module and the operation of the server are both simple and will operate according to their respective state charts, however as previously stated, implementation of the system will not be so easy. Another piece of software to manage the access lists is also required, using a designated card reader module ID to add new IDs to the list.

Electronic Door Strike (Right)

Card Reader (Below)

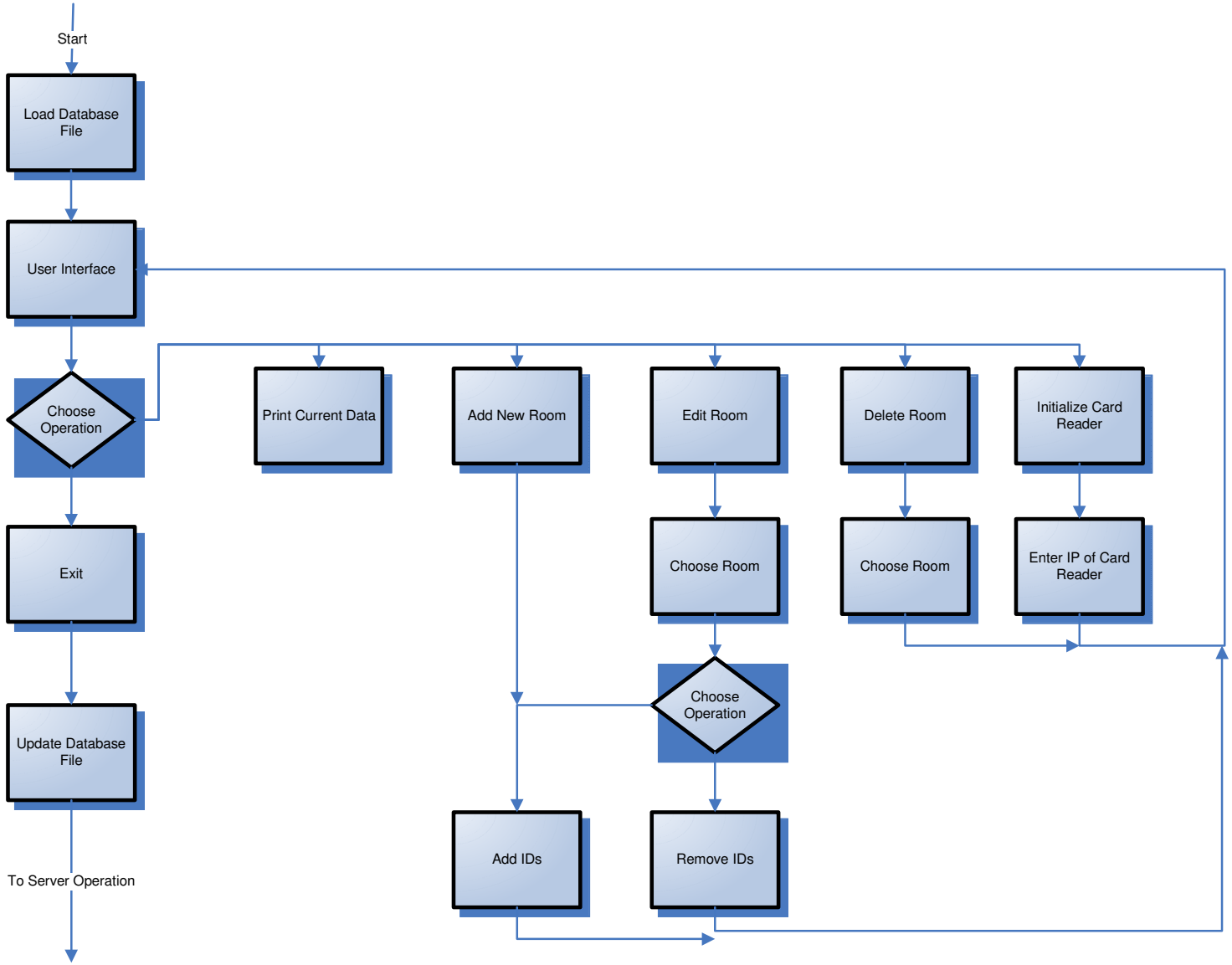
.

Project Architecture:

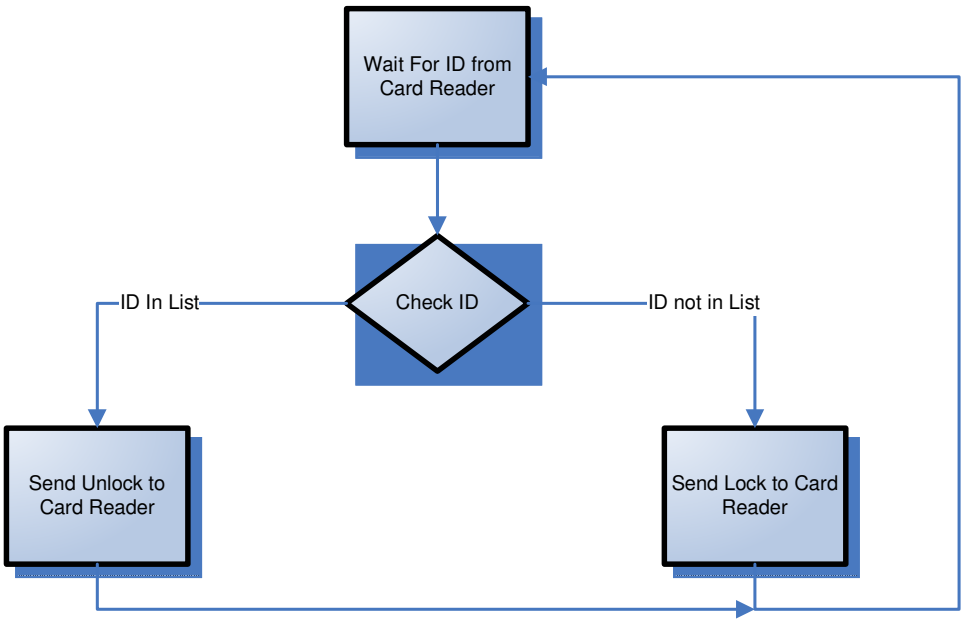


The Atmega32 controls most of the operation of the card reader module. The card reader shifts data out serially with its three signals (RDP, RCP, CLS) that the CPLD shifts in. The CPLD filters out unnecessary bits and places 4 bits out in parallel along with a Data Ready strobe for the Atmega32 to read. The ENC28J60 Ethernet Controller communicates to the Atmega32 through SPI and to the network through the Magjack RJ45 connection. It has an input data buffer that fills when it detects data for it and an output buffer that can be written to by the microcontroller. An SPI command can be sent to read or write data from and to the Ethernet controller to communicate with the network. Packets with appropriate encapsulation must be manually constructed before being sent. The server side PC program communicates with the Atmega32 with UDP packets as opposed to TCP.

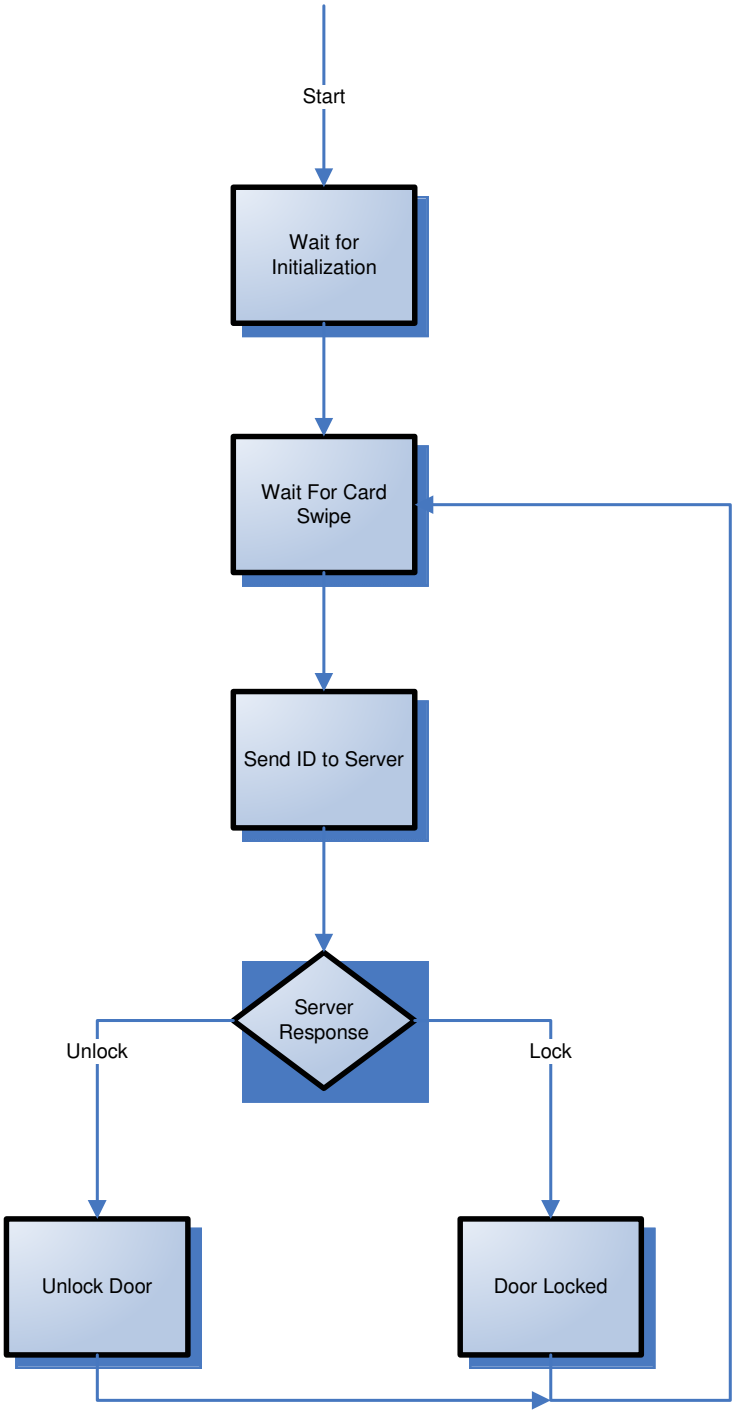
Server State Chart



Server State Chart Second Part



Card Reader Module State Chart



Division of Labor:

Task	Jesse Black	Ben Chai
Acquire Parts	50%	50%
Interface with Card Reader	30%	70%
Build Doorstrike Enclosure	100%	0%
Design PCB	90%	10%
Program Server	0%	100%
Solder PCB	100%	0%
Interface with Ethernet Controller	10%	90%
Re-Design PCB	100%	0%
Re-Solder PCB	100%	0%
Program Server	0%	100%
Debugging	50%	50%

User Manual:

Assembly:

Care must be taken when following the schematic because parts may have different footprints from the ones that were used in our project. Areas to pay special attention to:

- Magjack – Each model has a different pinout and possibly a different footprint.
- Potentiometer – Make sure the variable resistance is going to the LCD.
- NFET – A 3 to 10k resistor should be placed between the Atmega32 and the gate. Ensure gate goes to the Atmel, Source to Gnd, Drain to header pin.

Programming:

The PC software can be run on any windows computer on the same network as the card readers. The card reader modules must be given static IPs, and unique MAC addresses if they are to communicate with the server. A switch on each module determines the ID of the card reader module (0x0 – 0xF) with 0xF being the programmer module.

Operation:

When the server is started it automatically loads the current rooms and lists of IDs into memory. It then has a user interface to edit the lists of rooms and IDs granted access. Before a card reader can communicate with the server it must be initialized. On startup of the card reader it first waits for initialization from the server. Initialization is simply a packet sent to the card reader so that it will know where to send replies. This is necessary to allow the server to be run on any PC and consequently have any IP. Card reader modules MUST have a FIXED IP and a unique MAC address programmed into them. The server has an option to initialize card readers by typing in their IP address. After initialization the card readers enter the service loop where they wait for cards to be swiped and then proceed to send the data to the server. The server must first exit the list editing operation to begin the servicing operation which waits for an ID to be sent by the card reader then responds with whether or not the ID is in the list for that room. A card reader must be set to 'F' in order for the server to recognize it as a programmer during list editing. Afterwards the card reader ID must be set to the ID corresponding to the room number of the list in the server.

Bill of Materials:

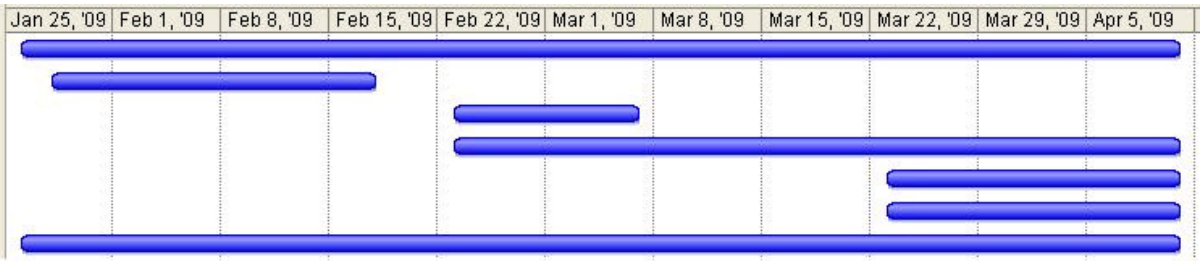
ENC28J60 Ethernet Controller - $3
Magjack RJ45 Socket - $5
Atmega32, EPM7064SLC44-10 (CPLD), LCD – Salvaged from previous classes (FREE)
Card Reader – Provided by Mike ($25 Approximate)
Door Strike – Provided by Mike ($20 Approximate)
Voltage Regulators – Provided by Mike (FREE)
PCB – Provided by lab (FREE)

References:

Beej's Guide to Network Programming - http://beej.us/guide/bgnet/
Tuxgraphics avr microcontroller electronics http://www.tuxgraphics.org/electronics/

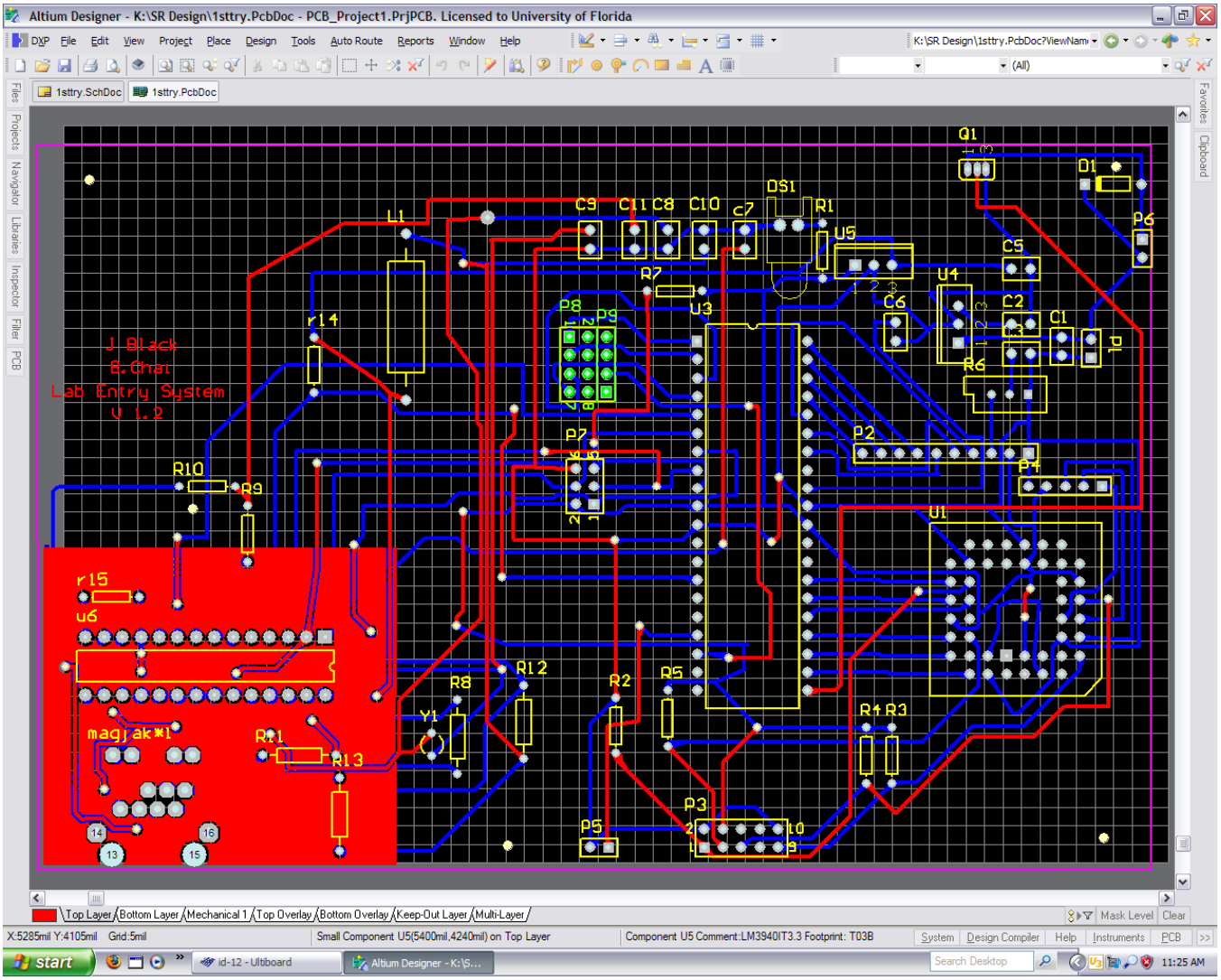
Gantt Chart

	Task Name	Duration	Start	Finish
1	Collect Parts	75 days?	Mon 1/26/09	Fri 4/10/09
2	ID from Card Reader	21 days	Wed 1/28/09	Tue 2/17/09
3	Make PCB	12 days	Mon 2/23/09	Fri 3/6/09
4	Server Program	47 days	Mon 2/23/09	Fri 4/10/09
5	ENC28J60	19 days	Mon 3/23/09	Fri 4/10/09
6	Make New PCB	19 days	Mon 3/23/09	Fri 4/10/09
7	Debugging	75 days	Mon 1/26/09	Fri 4/10/09

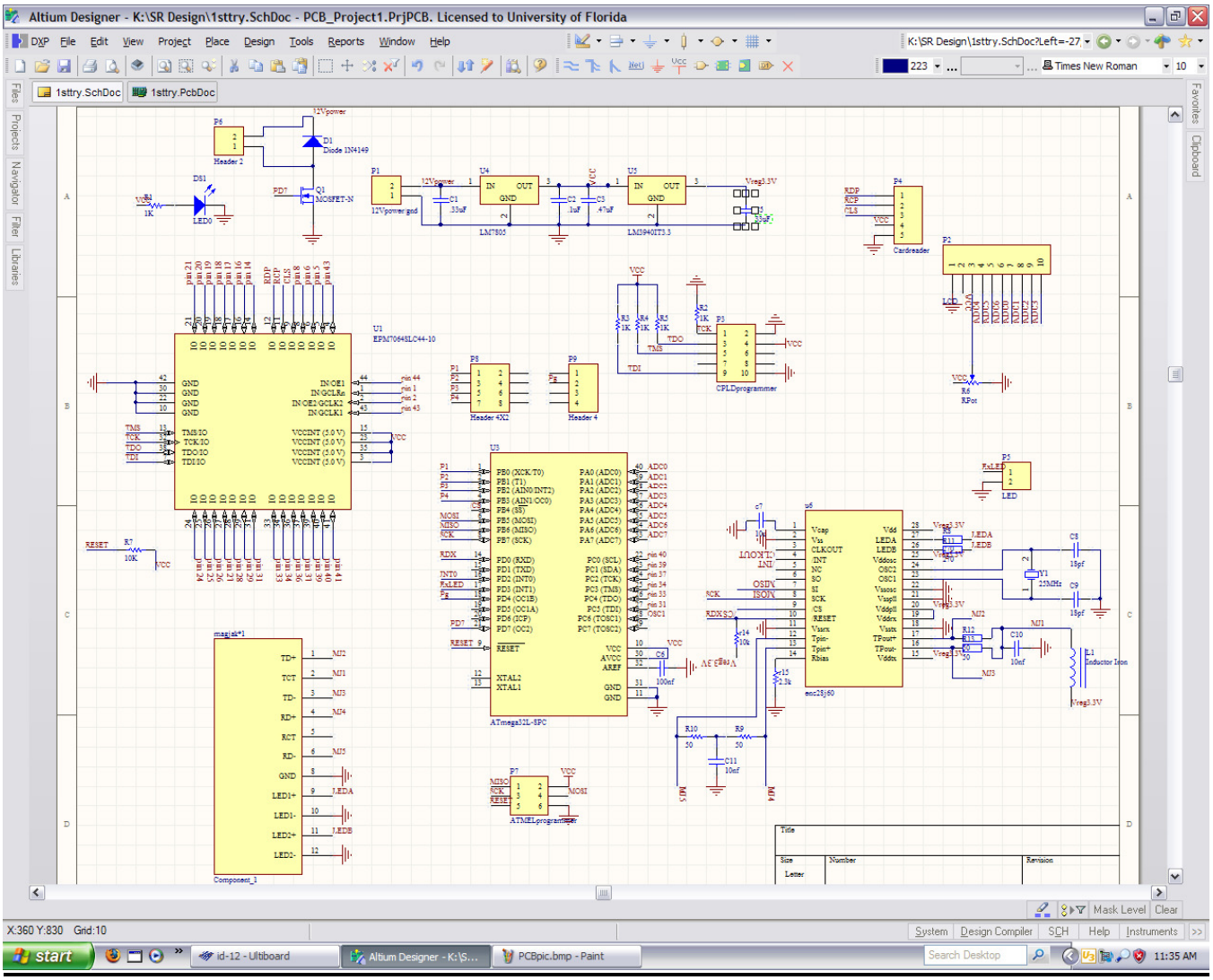


APPENDIX:

PCB Layout



PCB Schematic



CPLD VHDL Code

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE   ieee.std_logic_arith.all;
--USE ieee.std_logic_unsigned.all;
USE ieee.STD_LOGIC_SIGNED.all;

ENTITY Shift_Register IS
      PORT(       Din,Card_Present,Clk,RESET          : IN STD_LOGIC;
            Data_Rdy                      : OUT STD_LOGIC;
            --D               :     OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
            Dout              : OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END Shift_Register;

ARCHITECTURE behavior OF Shift_Register IS
      SIGNAL data : STD_LOGIC_VECTOR(4 DOWNTO 0);
      SIGNAL count : STD_LOGIC_VECTOR(2 DOWNTO 0);
      SIGNAL start : STD_LOGIC;

BEGIN
      PROCESS(Clk,RESET)
      BEGIN
            IF(RESET = '1') THEN
                  data <= "00000";
                  count <= "000";
                  start <= '0';
            ELSIF(Clk'EVENT AND Clk = '0') THEN
                  IF(Card_Present = '0') THEN
                        data(4) <= not(Din);
                        loop1: FOR i IN 0 TO 3 LOOP
                              data(i) <= data(i+1);
                        END LOOP;
                        IF(start = '1') THEN
                              IF(count = "100") THEN
                                    count <= "000";
                              ELSE
                                    count <= count + 1;
                              END IF;
                        END IF;
                  END IF;
                  IF(data = "01011") THEN
                        start <= '1';
                  END IF;

            END IF;

      END PROCESS;
      --D <= count;
      Dout <= data(3 DOWNTO 0);
      WITH (start & count) SELECT
            Data_Rdy <= '1' WHEN "1100",
                              '0' WHEN OTHERS;

END behavior;
```

Atmega32 C Code: (not including headers)

```
/*********************************************
 *
 *
 * Author: Benjamin Chai
 *
 *
 * Card Reader Module Firmware
 *
 * Title: Microchip ENC28J60 Ethernet Interface Driver
 * Chip type           : ATMEGA32 with ENC28J60
 *********************************************/
#define F_CPU 8000000UL  // 8 MHz
#include <avr/io.h>
#include <util/delay.h>
#include <stdio.h>
#include "ip_arp_udp.h"
#include "enc28j60.h"
#include "timeout.h"
#include "avr_compat.h"
#include "net.h"
#include "avrutil.h"
#include "lcd.h"

/****************************
PortC Pin Assignments

PC0 - Input - Data0
PC1 - Input - Data1
PC2 - Input - Data2
PC3 - Input - Data3
PC4 - Input - DataRdy (Active Hi)
PC5 - Output - RESET (Active Hi)


****************************/

// please modify the following two lines. mac and ip have to be unique
// in your local area network. You can not have the same numbers in
// two devices:
static uint8_t mymac[6] = {0x54,0x55,0x58,0x10,0x00,0x24};
static uint8_t myip[4] = {192,168,1,10};
static uint16_t myport =1337; // listen port for udp
// how did I get the mac addr? Translate the first 3 numbers into ascii is: TUX

#define BUFFER_SIZE 250
static uint8_t buf[BUFFER_SIZE+1];
static uint8_t init_packet[BUFFER_SIZE+1];

uint16_t recv();
void send(char * data, uint8_t len, uint16_t plen, uint16_t myport);
void get_ID(char * ID);
void Print_Data();
char Check_ID();
void Write_EEPROM (char data, uint16_t address);
char Read_EEPROM (uint16_t address);
uint8_t In_Master_List(char * ID);
void Open_Door();

int main(void){
```

```
  //Local Variables
  char ID[9];             //ID[0] - Card Reader ID, ID[1:8] -  Card Swipe ID
  uint16_t plen,init_plen,address;    //Holds Packet Length
uint8_t i,j;                    //Iterators
  i=0;
  j=0;

  init_LCD();

  //Setup Card Reader ID DDR
  DDRB &= (0xf0); //Set PortB (0-3) as input others unchanged
  PORTB |= (0x0f); //Internal Pullup Resistors on PortB (0-3)
  DDRD |= (0x10); //PD4 output
  PORTD &= (0xef); //PD4 = 0 others unchanged

  //Setup DoorStrike DDR
  DDRD |= (0x88); //PD3 output (Piezo) PD7 output (Doorstrike)
  PORTD &= (0x77); //PD3 PD7 = 0

  //Setup CPLD DDR
  DDRC = 0x20; //Set portc.5 as an output... for reset signal
  PORTC &= ~(0x20);       //c.5 = 1

/* enable PD0, reset as output */
DDRD |= 0x01;

/* set output to gnd, reset the ethernet chip */
PORTD &= ~(0x01);
delay_ms(100);
/* set output to Vcc, reset inactive */
PORTD|= (0x01);
delay_ms(100);


/*initialize enc28j60*/
enc28j60Init(mymac);

  ////////////////////////BLINK LEDS ON MAGJACK//////////////////////////
  //Tests Working SPI to enc28j60

  /* Magjack leds configuration, see enc28j60 datasheet, page 11 */
  // LEDA=greed LEDB=yellow
  //
  // 0x880 is PHLCON LEDB=on, LEDA=on
  // enc28j60PhyWrite(PHLCON,0b0000 1000 1000 00 00);
  enc28j60PhyWrite(PHLCON,0x880);
  delay_ms(500);
  //
  // 0x990 is PHLCON LEDB=off, LEDA=off
  // enc28j60PhyWrite(PHLCON,0b0000 1001 1001 00 00);
  enc28j60PhyWrite(PHLCON,0x990);
  delay_ms(500);
  //
  // 0x880 is PHLCON LEDB=on, LEDA=on
  // enc28j60PhyWrite(PHLCON,0b0000 1000 1000 00 00);
  enc28j60PhyWrite(PHLCON,0x880);
  delay_ms(500);
  //
  // 0x990 is PHLCON LEDB=off, LEDA=off
  // enc28j60PhyWrite(PHLCON,0b0000 1001 1001 00 00);
  enc28j60PhyWrite(PHLCON,0x990);
```

```
      delay_ms(500);

      ////////////////////END BLINKING LEDS////////////////////


      // 0x476 is PHLCON LEDA=links status, LEDB=receive/transmit
      // enc28j60PhyWrite(PHLCON,0b0000 0100 0111 01 10);
      enc28j60PhyWrite(PHLCON,0x476);
      delay_ms(100);


    //init the ethernet/ip layer:
    init_ip_arp_udp(mymac,myip);


      ////////////////////////INITIALIZING PHASE//////////////////////////
      lcd_home(0);
      lcd_print("Waiting for \nInitialization");
      plen = recv();

      //Save packet information to init packet for sends
      init_plen = plen;
      for(i=0; i<plen; i++)
            init_packet[i] = buf[i];

      send("Initialized",11,init_plen,myport);

      lcd_home(1);
      lcd_print("Initialized");

      ////////////////////////END INITIALIZING PHASE//////////////////////////


      ////////////////////////CARD READER PHASE////////////////////////

      //Main Loop to Read Card Swipes
      while(1)
      {
            get_ID(ID);
            ID[0] = Check_ID();
            send(ID,9,init_plen,myport);  //Send ID
            plen = recv();                              //Get Reply
            Print_Data();
            if(buf[UDP_DATA_P]=='U')
                  Open_Door();
            delay_ms(1000);

      }

      ////////////////////////END CARD READER PHASE//////////////////

    return (0);
}

//Loops until a valid UDP packet is recieved and stored in buf
//Blocking recv function
uint16_t recv()
{
      uint16_t plen;
      uint8_t i, payloadlen;

      char Message[20];
```

```
	while(1){
		// get the next new packet:
		plen = enc28j60PacketReceive(BUFFER_SIZE, buf);

	  /*plen will ne unequal to zero if there is a valid
	  * packet (without crc error) */
	  if(plen==0){
		continue;
	  }

	  // arp is broadcast if unknown but a host may also
	  // verify the mac address by sending it to
	  // a unicast address.
	  if(eth_type_is_arp_and_my_ip(buf,plen)){
		make_arp_answer_from_request(buf,plen);
		continue;
	  }
	  // check if ip packets (icmp or udp) are for us:
	  if(eth_type_is_ip_and_my_ip(buf,plen)==0){
		continue;
	  }

		//If pinged, reply
	  if(buf[IP_PROTO_P]==IP_PROTO_ICMP_V &&
buf[ICMP_TYPE_P]==ICMP_TYPE_ECHOREQUEST_V){
		// a ping packet, let's send pong
		make_echo_reply_from_request(buf,plen);
		continue;
	  }

		//If packet is a UDP packet, exit
		if (buf[IP_PROTO_P]==IP_PROTO_UDP_V){

				Print_Data();
		return plen;
	  }
	}
}

//Sends data of length len without modifying init packet contents
void send(char * data, uint8_t len, uint16_t plen, uint16_t myport)
{
	uint16_t i;
	for(i=0; i<plen; i++)
		buf[i] = init_packet[i];

	make_udp_reply_from_request(buf,data,len,myport);
}

//Loops Until Card Swiped and stores ID into ID array
void get_ID(char * ID)
{
	char output[9];
	int i;
	//pulses the reset for the cpld
	PORTC |= (1<<5);	//c.5 = 1
	delay_ms(5);
	PORTC &= ~(0x20);	//c.5 = 0
	delay_ms(1000);
	lcd_home(0);
	lcd_print("Ready for Card Swipe");
	for(i=0; i<11; i++)
```

```
	{
		while(1)
			if((PINC & 0x10))
			{
				if(i>2)
					ID[i-2] = ((PINC & 0x0F) | 0x30);
				break;
			}

		while((PINC & 0x10));

	}

	//Uncomment this block to print ID to lcd
	for(i=0; i<8; i++)
		output[i] = ID[i+1];
	output[8] = '\0';
	lcd_home(0);
	lcd_print("                    ");
	lcd_home(1);
	lcd_print("ID: ");
	lcd_print(output);

}

//Prints the data the received packet buf to lcd
void Print_Data()
{
	char Message[20];
	uint8_t payloadlen, i;

	payloadlen=buf[UDP_LEN_L_P]-UDP_HEADER_LEN;
	lcd_home(0);
	sprintf(Message,"UDP Packet! %d",payloadlen);
	lcd_print(Message);
	lcd_home(1);
	for(i=0;i<payloadlen;i++)
		Message[i] = buf[UDP_DATA_P+i];
	Message[i] = '\0';
	lcd_print(Message);
}

//Checks PortB switch to Read Card Reader ID
//ID is ascii value of hex number 0-F
char Check_ID()
{
	char c = 0;
	uint8_t sum;
	sum = (PINB & (0x0f));
	if(sum == (0x0f))
		c = '0';
	if(sum == (0x07))
		c = '1';
	if(sum == (0x0b))
		c = '2';
	if(sum == (0x03))
		c = '3';
	if(sum == (0x0d))
		c = '4';
	if(sum == (0x05))
		c = '5';
	if(sum == (0x09))
```

```
            c = '6';
      if(sum == (0x01))
            c = '7';
      if(sum == (0x0e))
            c = '8';
      if(sum == (0x06))
            c = '9';
      if(sum == (0x0a))
            c = 'A';
      if(sum == (0x02))
            c = 'B';
      if(sum == (0x0c))
            c = 'C';
      if(sum == (0x04))
            c = 'D';
      if(sum == (0x08))
            c = 'E';
      if(sum == (0x00))
            c = 'F';
      return c;
}

//Unlocks DoorStrike for 2 seconds
void Open_Door()
{
      PORTD |= (0x80);  //PD7 = 1 DoorStrike Unlock
      lcd_home(0);
      lcd_print("Access     \nGranted!    ");
      //Loop to sound piezo here or
      delay_ms(2000);
      PORTD &= (0x7f); //PD7 = 0 Locked
      lcd_home(0);
      lcd_print("           \n           ");
}
```

Server C++ Code:

```
// UDP_Server.cpp : Defines the entry point for the console application.

#include "stdafx.h"

#define PORT 1337
#define MAXBUFFER 1024


//Data Structure for Room Data
//Each Room contains necessary data for a single Card Reader
struct Room
{
      string RoomName;
      vector <string> IDList;
};


//Prototypes for UDP related Functions
bool UDP_Server();
bool Bind_Socket(SOCKET &MySocket);
bool Server(SOCKET &MySocket, vector <Room> &L, ofstream &AccessLog, ofstream
&DeniedLog);
void Receive_Packet(SOCKET &MySocket, string &Message, SOCKADDR &Remote, int &len);
void Send_Packet(SOCKET &MySocket, string &Message, SOCKADDR &Remote, int &len);
void Init_Card_Reader(vector <Room> &L,SOCKET &MySocket);

//Prototypes for List related Functions
void User_Interface(vector <Room> &RoomList, SOCKET &MySocket);
void Print_Menu();
void Print_Room_Data(vector <Room> &L);
void Add_New_Room(vector <Room> &L, SOCKET &MySocket);
void Delete_Room(vector <Room> &L);
void Edit_Room(vector <Room> &L, SOCKET &MySocket);
void Add_IDs(vector <Room> &L, int room, SOCKET &MySocket);
void Remove_IDs(vector <Room> &L, int room, SOCKET &MySocket);
int ID_Present(vector <Room> &L, int room, string ID);

//Prototypes for Logfile related Functions
int Get_Date(string &Date);
bool New_Day(int &Today);
void New_File(ofstream &AccessLog, ofstream &DeniedLog);
string Get_Time();



int _tmain(int argc, _TCHAR* argv[])
{
    // Start Winsock up
    WSAData wsaData;
      int nCode;
    if ((nCode = WSAStartup(MAKEWORD(1, 1), &wsaData)) != 0) {
            cerr << "WSAStartup() returned error code " << nCode << "." <<
                        endl;
        return 255;
    }

      //Run the Program
      if(!UDP_Server())
      {
```

```
			"ERROR!\n";
		}

		system("pause");
		//Shut Winsock down
		WSACleanup();
		return 0;
}

//Calls other functions to perform the all operations
bool UDP_Server()
{
		//Local Variables Here
		ofstream AccessLog,DeniedLog; //To Write Logfiles
		int Today;
		string Date;
		bool exit = false;					//Used to exit main service loop
		vector <Room> RoomList;			//Will hold all the data for rooms and
IDs
		SOCKET MySocket;					//Socket for UDP communications
		fd_set ReadFd;						//File Descriptor only used in
select() statement
		struct timeval tv;					//Time Value only used in select()
statement
		tv.tv_sec = 0;
		tv.tv_usec = 0;

		Today = Get_Date(Date);				//Get Todays Date

		//Create Datagram Socket
		MySocket = socket(AF_INET, SOCK_DGRAM, 0);
		if(!MySocket)
		{
			cout << "Socket Creation Failed!\n";
			return false;
		}

		//Bind the Port to the Socket
		if(!Bind_Socket(MySocket))
		{
			cout << "Failed to Bind!\n";
			return false;
		}

		//Load the Roomlist data file and allow modifications
		User_Interface(RoomList,MySocket);

		//Open Logfiles
		New_File(AccessLog,DeniedLog);

		//infinite loop to service clients
		system("cls");
		cout << "Monitoring Card Readers\nHit Enter key twice to Shut Down\n";
		while(true)
		{
			//On New Dates create a new logfile for better organization
			if(New_Day(Today))
				New_File(AccessLog,DeniedLog);

			FD_ZERO(&ReadFd);					//Zero out the fd_set
			FD_SET(MySocket, &ReadFd);			//(Re)Add Socket to fd_set
necessary because select() modifies ReadFd
```

```
            if(select(MySocket+1, &ReadFd, NULL, NULL, &tv)==-1)  //tv = 0 instant
timeout polls for nonblocking operation
            {
                  cout << "Select Error\n";
                  return false;
            }


            if(FD_ISSET(MySocket,&ReadFd))                  //Receive the Packet if
one was sent
                  if(!Server(MySocket,RoomList,AccessLog,DeniedLog))
                        break;
            //check for exit
            if(kbhit())
            {
                  if(exit)
                        return true;
                  system("cls");
                  cout << "Warning Pressing Enter key again will shut down
program\n";
                  exit = true;
            }

            //Do Other Stuff Here If Necessary
      }

      AccessLog.close();
      DeniedLog.close();
      return true;
}

//Attempts to make socket connection (Still UDP)
//Binds the Port to the socket
bool Bind_Socket(SOCKET &MySocket)
{
      sockaddr_in SocketAddress;

      SocketAddress.sin_family = AF_INET;
      SocketAddress.sin_addr.s_addr = INADDR_ANY;
      SocketAddress.sin_port = htons(PORT);


      if(bind(MySocket, (sockaddr*)&SocketAddress, sizeof(sockaddr_in)) ==
SOCKET_ERROR)
      {
            MySocket = INVALID_SOCKET;
            cout << "Failed to Bind!\n";
            return false;
    }
      return true;
}

//Receives a UDP Packet
//Data stored in Message, Return address stored in Remote
void Receive_Packet(SOCKET &MySocket, string &Message, SOCKADDR &Remote, int &len)
{
      char MessageBuffer[MAXBUFFER];
      int Mlength;
      Mlength = recvfrom(MySocket, MessageBuffer, MAXBUFFER, 0, &Remote, &len);

      if(Mlength == SOCKET_ERROR)
```

```
	{
		cout << "Failed to Receive Message!\n";
		return;
	}
	MessageBuffer[Mlength] = '\0';
	cout << "Received " << Mlength << " bytes\n";
	cout << "Message: " << MessageBuffer << endl;

	Message = MessageBuffer;
}

//Sends a UDP packet with data in Message to Remote
void Send_Packet(SOCKET &MySocket, string &Message, SOCKADDR &Remote, int &len)
{
	int Mlength;

	Mlength = sendto(MySocket, Message.c_str(), Message.length(), 0,
(sockaddr*)&Remote, len);

	if(Mlength == SOCKET_ERROR)
	{
		cout << "Failed to send Message!\n";
		return;
	}
	cout << "Sent " << Mlength << " bytes\n";

}
//Main Server Program to Unlock/Lock Doors
bool Server(SOCKET &MySocket, vector <Room> &L, ofstream &AccessLog, ofstream
&DeniedLog)
{
	SOCKADDR Remote;
	int len = sizeof(SOCKADDR);
	int CardReaderID;
	string ID,Message;

	//Get a Packet with Card ID
	Receive_Packet(MySocket,Message,Remote,len);

	/* Old Code ignore
	Mlength = recvfrom(MySocket, MessageBuffer, MAXBUFFER, 0, &Remote, &len);

	if(Mlength == SOCKET_ERROR)
	{
		cout << "Failed to Receive Message!\n";
		return false;
	}
	MessageBuffer[Mlength] = '\0';

	cout << "Received " << Mlength << " bytes\n";
	cout << "Message: " << MessageBuffer << endl;
	ID = (char*)(MessageBuffer+1);
	CardReaderID = atoi(&MessageBuffer[0]);
	*/

	//Decode Packet to ID and CardReader ID
	ID = Message.substr(1,8);
	CardReaderID = Message[0];
	if(CardReaderID>64)
		CardReaderID-=55;
	else
		CardReaderID-=48;
```

```
	//Check if ID is in that room
	if(ID_Present(L,CardReaderID,ID)!=-1)
	{
		Message = "Unlock!";
		AccessLog << "ID: " << ID << " Room: " << CardReaderID << " Time: " <<
Get_Time() << endl;
	}
	else
	{
		Message = "Lock!";
		DeniedLog << "ID: " << ID << " Room: " << CardReaderID << " Time: " <<
Get_Time() << endl;
	}

	cout << "Room: " << CardReaderID << " " << Message << endl;

	//Send Reply
	Send_Packet(MySocket,Message,Remote,len);

	/*
	Mlength = sendto(MySocket, MessageBuffer, Mlength, 0, (sockaddr*)&Remote,
len);

	if(Mlength == SOCKET_ERROR)
	{
		cout << "Failed to send Message!\n";
		return false;
	}
	cout << "Sent " << Mlength << " bytes\n";
	*/

	return true;
}

//Loads the Data from the file to RoomList
//Allows Modifications of file
//Updates stored file
//Initializes Card Readers To communicate to this machine
void User_Interface(vector <Room> &RoomList, SOCKET &MySocket)
{
	int NumRooms, NumIDs, i, j, choice;

	ifstream infile("RoomList.dat");
	infile >> NumRooms;
	infile.ignore();

	//////////////Read Room Data In from file//////////////////
	RoomList.resize(NumRooms);

	for(i=0; i<NumRooms; i++)
	{
		getline(infile,RoomList[i].RoomName);
		infile >> NumIDs;
		RoomList[i].IDList.resize(NumIDs);

		for(j=0; j<NumIDs; j++)
			infile >> RoomList[i].IDList[j];
	}

	////////////////////////////////////////////////
```

```
//////////Root Menu/////////////////
while(true)
{
	Print_Menu();

	if(!(cin >> choice))
	{
		cout << "Bad Input! Integers Only Please!\n";
		cin.clear();
		cin.ignore();
		choice = 10;
		continue;
	}

	///////Print Room Data/////////
	if(choice==0)
	{
		Print_Room_Data(RoomList);
		system("pause");
		system("cls");
		continue;
	}
	///////Create New Room/////////
	else if(choice==1)
	{
		Add_New_Room(RoomList, MySocket);
		continue;
	}
	///////Delete Room//////////////
	else if(choice==2)
	{
		Delete_Room(RoomList);
		continue;
	}
	///////Edit Room Data///////////
	else if(choice==3)
	{
		Edit_Room(RoomList,MySocket);
		continue;
	}
	////////Initialize Card Reader/////
	else if(choice==4)
	{
		Init_Card_Reader(RoomList,MySocket);
		continue;
	}
	////////Exit//////////////////
	else if(choice==5)
		break;
	else
	{
		system("cls");
		continue;
	}
}

infile.close();

////////Overwrite File with data//////////////
ofstream outfile;
outfile.open("RoomList.dat",ofstream::trunc);
```

```
	outfile << RoomList.size() << endl;

	for(i=0; i<RoomList.size(); i++)
	{
		outfile << RoomList[i].RoomName << endl;
		outfile << RoomList[i].IDList.size() << endl;

		for(j=0; j<RoomList[i].IDList.size(); j++)
			outfile << RoomList[i].IDList[j] << endl;
	}

	outfile.close();

	return;
}

//Prints the menu for user input
void Print_Menu()
{
	cout <<		"Enter Selection:\n"
		<<		"0: Print Current Room Data\n"
		<<		"1: Add New Room\n"
		<<		"2: Delete Room\n"
		<<		"3: Edit Room Data\n"
		<<		"4: Initialize Card Reader and Edit Master List\n"
		<<		"5: Exit (Will Update Room Data to file)\n"
		<<		"NOTE: Card Reader Must be Initialized when powered on
before use\n";
}

//Prints all the current rooms and the ID's Associated with them
void Print_Room_Data(vector <Room> &L)
{
	for(int i=0; i<L.size(); i++)
	{
		cout << "Room ID: " << i << " - " << L[i].RoomName << endl;

		for(int j=0; j<L[i].IDList.size(); j++)
			cout << "ID " << j << " : " << L[i].IDList[j] << endl;
	}
}

//Adds a new room to the list and adds ID's to the room
void Add_New_Room(vector <Room> &L, SOCKET &MySocket)
{
	char c;
	Room temp;
	system("cls");

	cin.ignore();
	do
	{
		cout << "Enter Room Name (ie: Senior Design Lab)\n";
		getline(cin,temp.RoomName);
		cout << "You Entered <" << temp.RoomName << "> Is This Correct? Y/N\n";
		cin >> c;
		cin.ignore();
	}while((c!='y')&&(c!='Y'));
	L.push_back(temp);
	Add_IDs(L,L.size()-1,MySocket);
}
```

```
//Removes a Room and All ID's from the list
void Delete_Room(vector <Room> &L)
{
	char c;
	int room;
	do
	{
		system("cls");
		cout << "Current Rooms:\n";
		for(int i=0; i<L.size(); i++)
			cout << i << ": " << L[i].RoomName << endl;
		cout << "Enter Room ID To Delete\n";
		if(!(cin >> room))
		{
			cin.clear();
			cin.ignore();
			room = 1337;
			continue;
		}
		if(room>=L.size())
		{
			continue;
		}
		cout << "You selected: " << L[room].RoomName << " Delete FOREVER?!
Y/N\n";
		cin >> c;
		cin.ignore();

		if((c=='N')||(c=='n'))
		{
			system("cls");
			return;                                  //Exits if you dont want to
delete a room
		}

	} while((c!='Y')&&(c!='y'));
	L.erase(L.begin()+room);
}

//Allows ID's to be added or removed from the list
void Edit_Room(vector <Room> &L, SOCKET &MySocket)
{
	int room, choice;
	while(true)
	{
		system("cls");
		Print_Room_Data(L);
		cout << "Enter Room Number\n";
		if(!(cin >> room))
		{
			cin.clear();
			cin.ignore();
			room = 1337;
			continue;
		}
		if(room>=L.size())
			continue;
		system("cls");
		cout << "0: Add IDs\n1: Remove IDs\n";
		if(!(cin >> choice))
		{
```

```
                  cin.clear();
                  cin.ignore();
                  choice = 1337;
                  continue;
            }
            if(choice==0)
            {
                  Add_IDs(L,room,MySocket);
                  break;
            }
            else if(choice==1)
            {
                  Remove_IDs(L,room,MySocket);
                  break;
            }
      }
}

//Loops to Add Ids to a room
void Add_IDs(vector <Room> &L, int room, SOCKET &MySocket)
{
      char CardReaderID;
      string Message;
      SOCKADDR Remote;
      struct timeval tv;
      tv.tv_sec = 0;
      tv.tv_usec = 0;
      int len = sizeof(SOCKADDR);
      fd_set ReadFd;
      system("cls");
      cout << "Press Enter key when finished\n";
      cout << "Swipe Cards now to ADD ID to Room: " << L[room].RoomName << endl;
      do
      {
            FD_ZERO(&ReadFd);
            FD_SET(MySocket, &ReadFd);

            if(select(MySocket+1, &ReadFd, NULL, NULL, &tv)==-1)
            {
                  cout << "Select Error\n";
                  return;
            }
            if(FD_ISSET(MySocket,&ReadFd))
            {
                  Receive_Packet(MySocket, Message, Remote, len);
                  CardReaderID = Message[0];

                  //Check if packet is not from Programming Reader
                  if(CardReaderID!='F')
                  {
                        Message = "Lock!";
                        Send_Packet(MySocket,Message,Remote,len);
                        continue;
                  }

                  Message = Message.substr(1,8);
                  if(ID_Present(L,room,Message)>=0)
                  {
                        cout << "ID: " << Message << " already exists in the
list!\n";

                        Message = "BAD!";
```

```
                        Send_Packet(MySocket,Message,Remote,len);
                        continue;
                }

                cout << "ID Added: " << Message << endl;
                L[room].IDList.push_back(Message);

                Message = "Good!";
                Send_Packet(MySocket,Message,Remote,len);
        }

    }while(!kbhit());
    cin.ignore();
}

//Loops to Remove IDs one at a time from a Room
void Remove_IDs(vector <Room> &L, int room, SOCKET &MySocket)
{
    int index;
    while(true)
    {
        system("cls");
        cout << "Enter ID Index to Remove (0 to Exit)\n";
        for(int i=0; i<L[room].IDList.size(); i++)
            cout << i+1 << " - " << L[room].IDList[i] << endl;
        if(!(cin >> index))
        {
            cin.clear();
            cin.ignore();
            index = 1337;
            continue;
        }
        if(index==0)
        {
            system("cls");
            return;
        }
        if((index<0)||(index>L[room].IDList.size()+1))
            continue;

        index--;

        cout << "ID Removed: " << L[room].IDList[index] << endl;
        L[room].IDList.erase(L[room].IDList.begin()+index);
        system("pause");
    }
}

//Checks if an ID is in the room returns the index
int ID_Present(vector <Room> &L, int room, string ID)
{
    if(room>L.size())
        return -1;
    for(int i=0; i<L[room].IDList.size(); i++)
        if(ID == L[room].IDList[i])
            return i;
    return -1;
}

//Sends a Packet to the Card Reader
//This is necessary so Card Readers will have the information to reply to server
```

```
//Card Readers have static IPs but this allows the server to be portable to any
machine.
void Init_Card_Reader(vector <Room> &L,SOCKET &MySocket)
{
	int choice;
	bool edit = false;
	char c = '0';
	string IP, Message;

	system("cls");
	cout << "Do You Want to See/Edit Master Room List? Y/N\n";
	cin >> c;
	if((c=='Y')||(c=='y'))
	{
		Message = "Yes";
		//edit = true;
	}
	else
		Message = "No";

	cout << "Enter Card Reader IP Address (xxx.xxx.xxx.xxx)\n";
	cin >> IP;
	//cin.ignore();

	int len = sizeof(SOCKADDR);
	SOCKADDR Remote;
	sockaddr_in SocketAddress;
	u_long	ServerAddress;

	ServerAddress = inet_addr(IP.c_str());
	if(ServerAddress == INADDR_NONE)
	{
		cout << "Invalid IP address!\n";
		return;
	}

	SocketAddress.sin_family = AF_INET;
	SocketAddress.sin_addr.s_addr = ServerAddress;
	SocketAddress.sin_port = htons(PORT);

	sendto(MySocket, Message.c_str(), Message.size(), 0,
(SOCKADDR*)&SocketAddress,sizeof(sockaddr_in));
	Receive_Packet(MySocket, Message, Remote,len);
	if(!edit)
		return;
	while(true)
	{
		system("cls");
		cout << "0 - Clear List (Note: Must Be Cleared at least ONCE *BEFORE*
reading or adding)\n"
			<< "1 - Print Current List\n"
			<< "2 - Add ID\n"
			<< "3 - Exit\n";
		if(!(cin >> choice))
		{
			cin.clear();
			cin.ignore();
			choice = 1337;
			continue;
		}
		//Clear List
		if(choice == 0)
```

```
		{
			Message = '0';
			Send_Packet(MySocket,Message,Remote,len);
			Receive_Packet(MySocket,Message,Remote,len);
			system("pause");
		}
		//Print List
		else if(choice == 1)
		{
			Message = '1';
			Send_Packet(MySocket,Message,Remote,len);
			Receive_Packet(MySocket,Message,Remote,len);
			int i = Message[0]-48;
			while(i>0)
			{
				Receive_Packet(MySocket,Message,Remote,len);
				cout << Message << endl;
			}
			system("pause");
		}
		//Add ID
		else if(choice == 2)
		{
			Message = '2';
			Send_Packet(MySocket,Message,Remote,len);
			cout << "Enter UF ID (XXXXXXXX)\n";
			cin >> Message;
			if(Message.length()>8)
				Message.resize(8);
			if(Message.length()<8)
			{
				Message += "00000000";
				Message.resize(8);
			}
			cout << Message << " Added to Master List!\n";
			Send_Packet(MySocket,Message,Remote,len);
			system("pause");
		}
		//Exit
		else if(choice == 3)
		{
			Message = '3';
			Send_Packet(MySocket,Message,Remote,len);
			system("cls");
			break;
		}
	}
	//cout << Message << endl;
}

//Puts the Date into string and returns day of the month
int Get_Date(string &Date)
{
	time_t rawtime;
	struct tm * timeinfo;
	char buffer [80];

	time ( &rawtime );
	timeinfo = localtime ( &rawtime );

	strftime (buffer,80,"%Y-%m-%d",timeinfo);
	Date = buffer;
```

```
	return timeinfo->tm_mday;
}

//Checks if the current day is the same as Today
bool New_Day(int &Today)
{
	time_t rawtime;
	struct tm * timeinfo;
	char buffer [80];

	time ( &rawtime );
	timeinfo = localtime ( &rawtime );
	if(Today==timeinfo->tm_mday)
		return false;

	Today = timeinfo->tm_mday;
	return true;
}

//Open Files For Log Files Always append if file exists
void New_File(ofstream &AccessLog, ofstream &DeniedLog)
{
	if(AccessLog.is_open())
		AccessLog.close();
	if(DeniedLog.is_open())
		DeniedLog.close();
	string Date,FileName;
	int Today = Get_Date(Date);
	FileName = Date;
	FileName += ".in";
	AccessLog.open(FileName.c_str(),ofstream::app);
	FileName = Date;
	FileName += ".out";
	DeniedLog.open(FileName.c_str(),ofstream::app);
}

//Returns string with formatted current time
string Get_Time()
{
	string temp;
	time_t rawtime;
	struct tm * timeinfo;
	char buffer [80];

	time ( &rawtime );
	timeinfo = localtime ( &rawtime );

	strftime (buffer,80,"%H:%M.%S",timeinfo);
	temp = buffer;
	return temp;
}
```