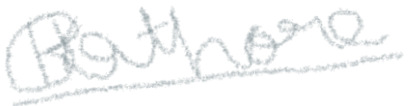
HONOR CODE PLEDGE

ECEN 5613 Final Project

By signing this sheet, students certify that the work they submit is their own, and that they have clearly identified any code, schematics, design details, documentation, pictures, or other work obtained from another source. Each student on a project team is expected to certify their work by signing this pledge.

Honor Code Pledge: "On my honor, as a University of Colorado student, I have neither given nor received unauthorized assistance on this work. I have clearly acknowledged work that is not my own."

Student 1 Name (printed): HARSH RATHORE

Student 1 Signature: 

Date: 12/14/2019

8051 IR REMOTE CONTROL

Harsh Rathore

Final Project Report
ECEN 5613 Embedded System
Design December 14, 2019

1 INTRODUCTION

Today, a lot of devices operate over the air through infrared and other mediums. In this project infrared communication is used for interacting with the peripherals on the board. This section covers new hardware elements design and driver for those hardware devices interfaced.

1.1 System Overview

In today's day and age everything around us is automated and easily accessible and controlled, which I tried to make use of by getting some hands-on experience with Infrared communication which controls several number audio, video and other electrical devices.

I have been fascinated to work on a project which involves several hardware peripherals that work together in the functioning of a system, where the peripherals can be interacted with utilizing a single remote-control device.

Through this project, I designed a simple IR controlled system wherein the peripherals are interacted with through the press of a button on their remote. This is possible because each button key pressed has a unique key that can be decoded by the microcontroller to perform the desired operations. The project consists of the following components which form the core part of the system.

2 TECHNICAL DESCRIPTION

The following sections detail the design and implementation of the 8051 based IR Remote Control.

This section covers all the technical aspects of the project including the Hardware elements involving Graphical LCD, Keypad, TSOP, and Dc motor. The firmware Design of the project covers 8051 Driver code, PWM based dc motor control, Graphical LCD code and other input and output elements. Software Compiler used is SDCC.

2.1 Board Design

The system is based on Atmel AT89C51 Microcontroller board development which we worked on in our lab 1 to lab4. The system is integrated with various other sensors, actuators, and displays that form the core of the system design. The inputs are the keypad 4*3, UART and remote control which is based on NEC protocol. TSOP 38238 is an Infrared receiver used to detect the pulses of the required frequency of 38 kHz used for communication with the microcontroller and other peripherals on the board. We use the external Rom address starting from 0XF000 to for LCD 16*4 l working and configuration. The Microcontroller Ports P0, P1, P2 and P3 are used for connecting the various board peripherals**.** The various peripherals along with the IR receiver sensor are connected to the microcontroller in the following manner as shown in the figure. 2-1**.**

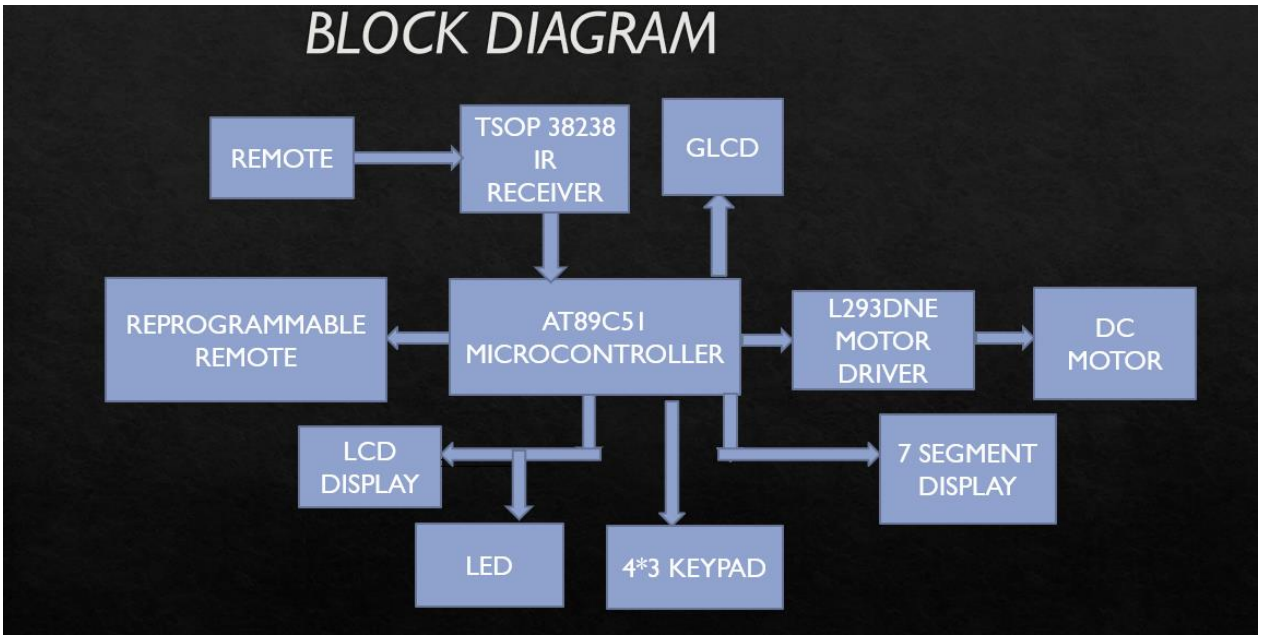


FIGURE 2.1 BLOCK DIAGRAM OF 8051 BASED IR REMOTE CONTROL

2.1.1 TSOP 38238 IR Receiver

The TSOP IR receiver is a 38 kHz sensor used to decode the pulses of a remote-based IR sensor implementing NEC protocol which is being used in the project. It uses a very low supply current and consists of a photodetector and pre-amplifier in a single package along with an internal filter for pulse code modulation. It consists of 3 pins which are connected to VCC, ground and the microcontroller pin having different pin numbers for different kinds of receivers used. The supply voltage used for the project is 5-volt VCC. however, the sensor operates on a voltage range from 2.7 to 5.5 volts. It is insensitive to noise from other environmental sources. The 2 pin IR receiver can also be used for detecting the pulses. however, it is sensitive to noise from other environmental sources and is usually just photodiodes or phototransistors. The output pin sends the captured pulses to the microcontroller.

Initially, I started with 2 pin IR receivers which are standard sensors for capturing IR signals of all frequencies, wherein I received the pulses which are continuously stream of high to low pulses for each of logic 1, logic 0 address and command bits. The TSOP uses these signals and filters them to produce a continuous stream of high pulse for Logic 1 and low pulse for Logic 0.

I worked with 2 pin IR sensor which sensed correct pulse widths some times and at other times showed incorrect pulses. The start of frame which is supposed to be 9 mili-seconds with 4.5 mili-seconds space was received as 9ms with 2.5 mili-seconds space majority of the times. When I replaced it with 3 pin TSOP receiver I could not receive any signal from the receiver. For the second TSOP sensor use, I could receive voltage at very low logic thresholds below 0.3 volts, beyond which I did not receive any signal, which would not be sufficient to be understood by the microcontroller. For this, I connected a transistor circuit consisting of led connected through a transistor as done in the lab. This was done to check the glow on the led

for every remote key pressed. the led glowed but it did not glow very brightly as it should have. With this, I connected a third TSOP sensor and it captured the correct pulses at higher voltage thresholds too. I was able to see outputs at higher logic voltage thresholds with the sensor with the pre-trigger buffer set to 10 percent.

The operating voltage of 5 volts is obtained from the voltage regulated output form 7805a which is used on the board. Sometimes the output pin is unable to produce the desired voltage for the microcontroller and can be connected to the microcontroller pin through a transistor resistor network to produce the required voltage which is needed by the microcontroller to understand it.

This package suppresses noise in signals or any disturbance present through differences in envelope duty cycle, frequency, and burst length. The data signal must be close to the bandpass center frequency of the device to operate as per requirements. Some of the other protocols which can be supported by the device are RC5, Sharp. I worked with 2 remotes working on NEC, one showed pulses consisting of 48 bits and other 32 bit pulses of address and command bits. I decided to use the latter for my project.

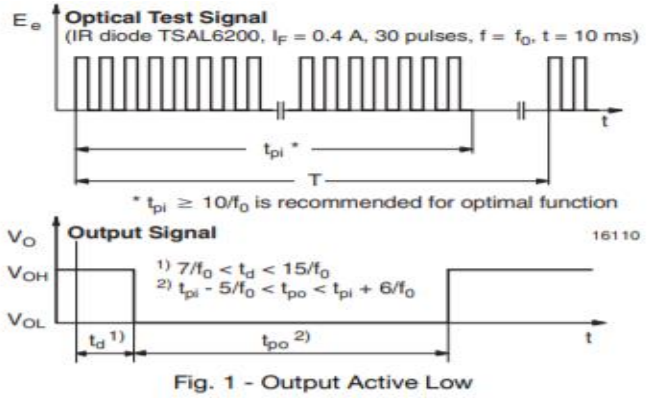


Fig. 1 - Output Active Low

FIGURE 2.2 from tsop 38238 datasheet fig 1

The minimum Burst length the Tsop uses is 10 cycles/burst. A gap time of 50 milliseconds was found when seen on the logic analyser, which is used to set the values of variables to calculate pulse width and pulse count in firmware. The tsop sensor can receive a continuous pulse burst of 1800 per second. The 38238 TSOP sensor has the following characteristics.

	TSOP382..
Minimum burst length	10 cycles/burst
After each burst of length a minimum gap time is required of	10 to 70 cycles ≥ 10 cycles
For bursts greater than a minimum gap time in the data stream is needed of	70 cycles > 4 x burst length
Maximum number of continuous short bursts/second	1800
NEC code	Yes
RC5/RC6 code	Yes
Thomson 56 kHz code	Yes
Sharp code	Yes
Suppression of interference from fluorescent lamps	Mild disturbance patterns are suppressed (example: signal pattern of Fig. 14)

FIG 2.3 CHARACTERISTICS OF TSOP datasheet fig 16

2.1.2 NOKIA 5110 GRAPHICAL LCD

The LCD used in the project is an 84*48 graphical LCD on top of the already used 16*4 LCD in the labs 1 to 4. The graphical LCD uses PCD 8544 as the internal controller. It has a serial bus interface and operates on SPI protocol. The LCD uses the voltage range from 2.7 to 3 voltage but gives the best performance with 3.3 volts. Hence I have used a voltage divider network of 100 and 50 ohms connected in series and 150 ohms connected in parallel to give the voltage drop of around 3.3 volts to operate the graphical LCD. The VCC and backlit pin are connected across resistor 150 ohms to give voltage of 3.3 volts.

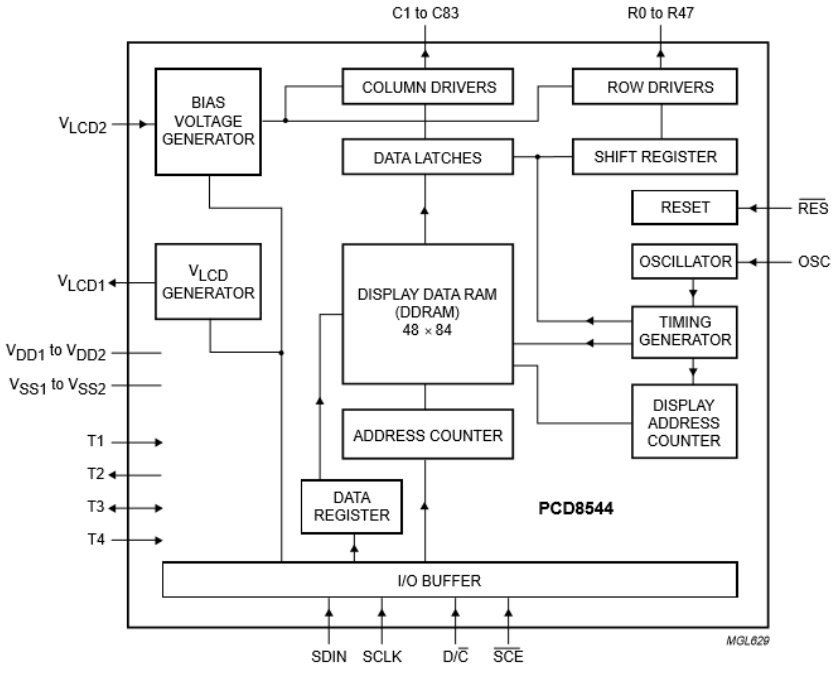


FIG 2.4 The figure show the internal circuitry of the PCD 8544 Controller from fig 1 from Nokia 5110 datasheet

The wiring of the graphical LCD involved connecting the clock, data, chip enable, data/command select as the main SPI pins. The pins are connected to the microcontroller on port 2. The pins VCC, ground and backlight control pin are connected to appropriate voltage levels. The reset pin is connected to the microcontroller to cause external reset, the signal

being active low. The maximum speed of serial operation is 4 MBits/s. The LCD voltage of 3.3V was adequate to turn on the backlit led and turn on the LCD display. We can write data into LCD consisting of x and y of 84 *6 pixels continuously where the value of DDRAM addresses will be incremented automatically, wherein each byte was written separately, with the most significant bits being send first.

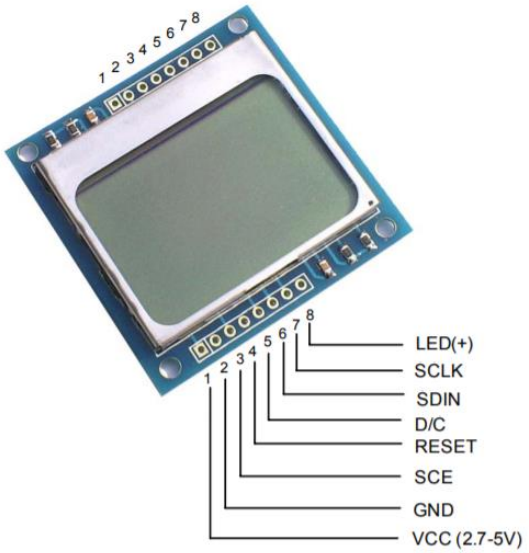


FIG 2.5 NOKIA 5110 PIN OUT from fig 5 of nokia 5110 user manual

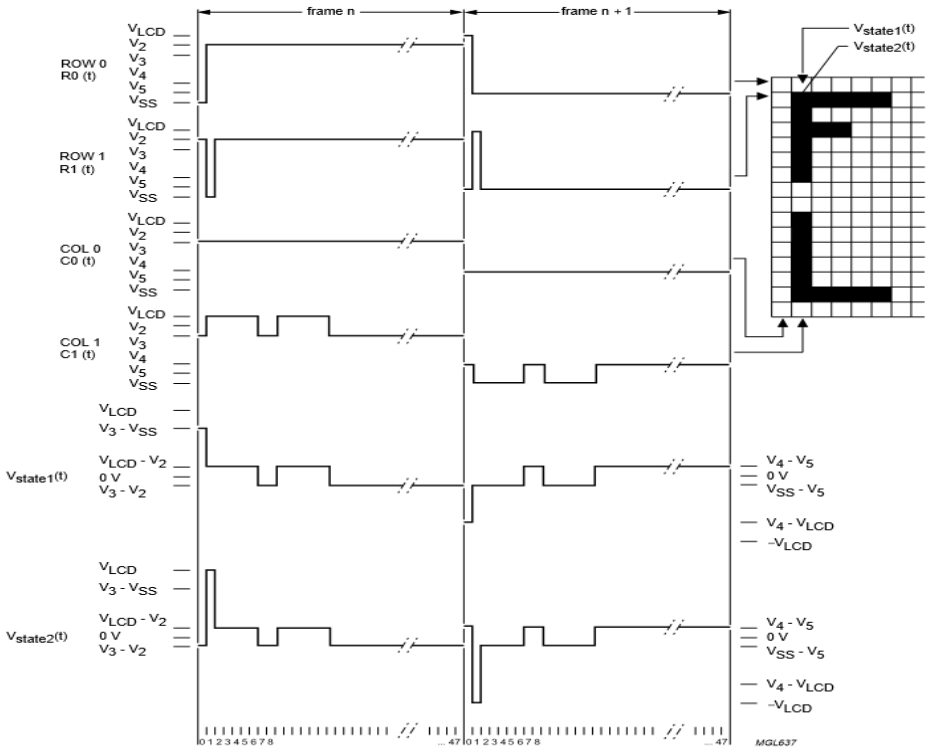


FIG 2.6 The above diagram shows the LCD driver waveform from fig 2 nokia 5110 datasheet

The Graphical LCD was soldered to the microcontroller to display the additional messages on selecting the reprogrammable functionality of the remote control. The 16*4 LCD, on the other hand, was used to display the keys that are required to be inputted for enabling a particular set of keys to perform a set of different operations.

2.1.3 L293DNE Motor driver and Dc motor

The motor driver L293DNE accepts the small current and amplifies to generate a larger current for the motor to be driven. L293dne IC is a16 pin H bridge driver that drives the motors bidirectionally. It can take control of 2 motors connections and drives it in the required direction.

The l293dne driver operates on a voltage range varying from 4.5 to 36 volts with the capability of providing bidirectional currents of around 600 mA. The pins 4,5 and 12, 13 are grounded. The pin 1 is used as enable and 3 and 6 as input pins of the motor. It needs 5v supply for its internal circuitry on pin number16 and to power drivers with VCC 4.5 v to 36v.

Each output consists of a Darlington source and Darlington transistor sink. The enable pin 1,2 results in enabling drivers 1,2 and 3,4 respectively. When the driver pin is high the drivers are enabled and are in phase with the inputs. On the other hand they are out of phase and in high impedance state when enable is low.

The motor can be controlled through a single switch. However, a combination of switches used helps to turn the motor on or off. the H-bridge uses 4 switches like S1, S2, S3, S4. When S1, S4 is closed a positive voltage is applied to the motor. When the switches are open are S2 S3 closed invert operation enables the motor to reverse its direction.

With the motor driver, it is possible to run the motor at an increased speed using pulse width modulation, stop a motor and also reverse its direction. The motor comes to a stop when the terminals are shorted or when the motor is detached from the circuit.

The input pins including the enable pin 1 is connected to Microcontroller pin 1.3 which also where Atmel 8051 generates PWM through PCA when enabled in module 0. The motor direction pins are connected to pin 2.0 and pin 2.1 of the microcontroller. The output pins 3,6 are connected to the motor to help drive the motor in different directions.

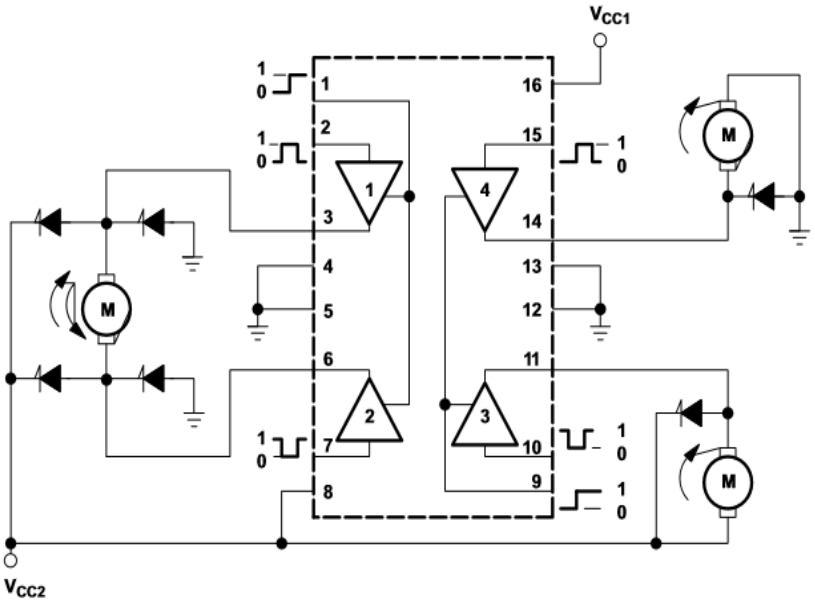


FIG 2.7 L293D INTERNAL CIRCUITARY from L293D DATASHEET 8.2

*2.1.4 3*4 Keypad Interface*

The 3*4 Keypad consists of push buttons which act as switches and are interconnected to each other. There are 3 push buttons in each of the 4 rows of our keypad. the terminals of the push button are connected in a manner where one terminal of the first row is connected together and another terminal of 3 push buttons represents the columns of the push button are connected together. In this way, we connect 7 pins of the keypad to the microcontroller. All the pins of the microcontroller are connected to port number 1 with the exception of Pins 4 and 5 which are connected to pin number 3.4 and 3.5 of the microcontroller as pins number p1.2 and pin 1.3 have led connected and dc motor driver's enable pin connected to it. These are used as input keys for the project along with the input accepted from the uart for the reprogrammable functionality of the project.

The keypad pins are connected to VCC through 4.7k resistors so that they are pulled up when in a non-active state. The keypads gave a lot of debouncing when connected as inputs to the microcontroller. The debouncing effect was observed on the graphical LCD and the seven-segment display when different keys pressed showed varying results on the GLCD and the seven-segment display. Earlier the pins for the motor and the led clashed with pin connections of the keypad which was later changed to stop the operation of these devices on the press of a key, by connecting those pins to pin 3.4 and pin 3.5 which are also used as td and rx.

When any button is pressed, we need to get the location of the button pressed, which is the corresponding row and column press of the button. Once we get the location we can use it as an input to perform other operations perfectly. As shown in the above circuit diagram, to interface Keypad, we need to connect 7 terminals of the keypad to any port (7 pins) of the

microcontroller.

We need to get the location of the button, which means the corresponding Row and Column number. Once we get the location of the button, we can print the character accordingly. The row pins of the keypad are connected to the microcontroller together and column pins are connected together. These are connected in such a way so that the column acts as input and row as output lines and location of the key pressed is found out systematically.

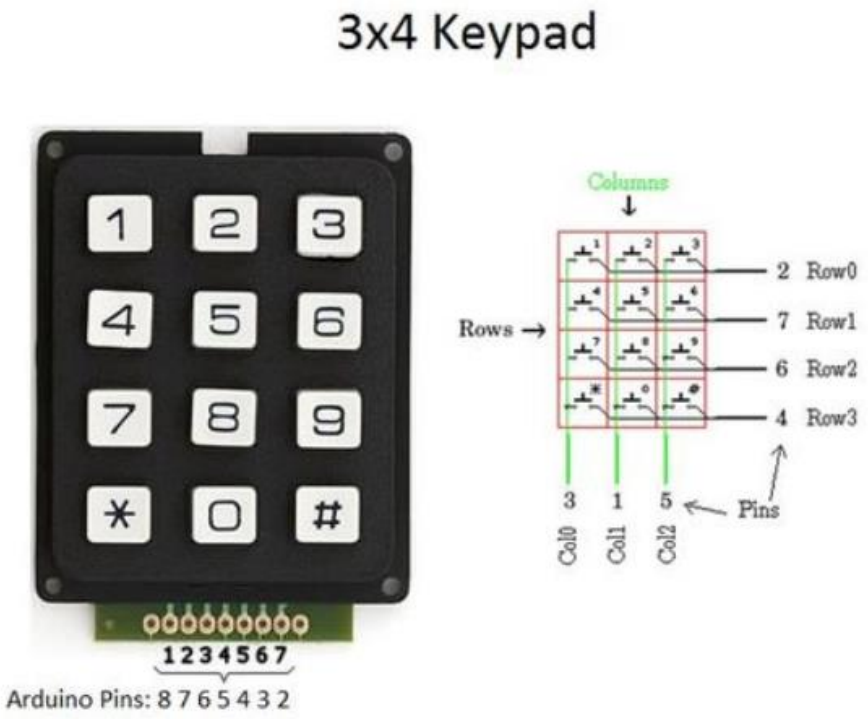


FIG 2.8 KEYPAD INTERNALLY from https://www.instructables.com

2.1.5 Seven Segment Display

The SD The seven-segment display consists of 8 LEDs to illuminate one segment of the unit and display different numbers. The numbers can be illuminated by considering each segment as a line. We can use refer these segments as a,b,c,d,e,f,g,h where the eight elements is used to represent the dot. The middle pins are common anode or cathode respectively, which are internally shorted. Therefore, we need to connect only one of these two pins. There are two types of 7 segment displays: common anode and common cathode. In common anode, all negative pins are internally shorted whereas in common cathode the positive pins are internally connected. The microcontroller does not provide enough voltage to drive the seven-segment LEDs and hence we connect LEDs cathode to microcontroller and LEDs anode to the power supply.

The connection of the seven-segment display involves connecting the different segments/leds of the display to the microcontroller pins. I have used microcontroller port 2 from pin 2.0 to 2.7 to connect the different pins of seven segement. A 470-ohm resistor is connected to the power supply.

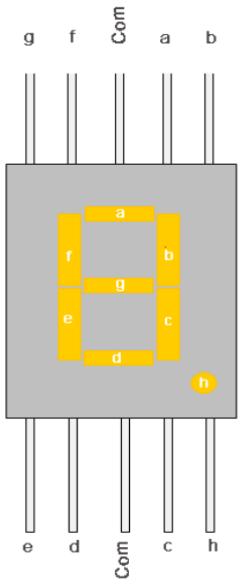


FIG 2.9 SEVEN SEGMENT DISPLAY from figure https://circuitdigest.com/microcontroller-projects/7-segment-display-interfacing-with-8051

2.3 Firmware Design

Firmware for this project included drivers for new hardware interfaces which act as input and output devices for the project, included drivers for each of the new hardware interfaces (IR remote control, keypad, Graphical LCD, PWM based motor, and seven-segment display). All firmware was developed with SDCC 3.9.0 and the PAULMON2 ROM monitor on the 8051. Appendix 8.4 contains a complete firmware source code listing.

All firmware uses SDCC's large memory model. I decided to stick with the small memory model for simplicity and efficiency.

.

2.3.1 IR Remote Control Driver

The remote driver consists of writing functions to decode the NEC protocol's pulses and pulse widths. The protocol as seen in the figure consists of 13.5 ms start of the frame where 9ms is pulse burst and 4.5ms is the space before the address and command bits. The start of frame is followed by 32-bit pulses which are in the following order:

1. 8-bit address
2. 8-bit inverse of the address
3. 8-bit command field
4. 8-bit inverse of the command
5. 562.5 uS period end pulse.

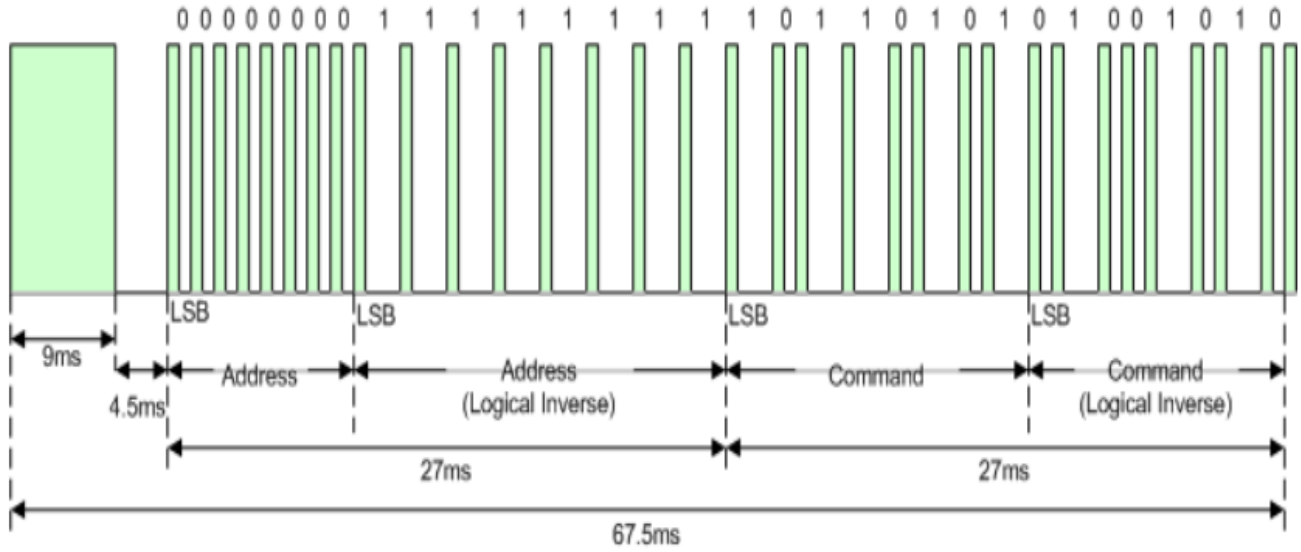


FIG 3.1 NEC MESSAGE FRAME FORMAT from
https://exploreembedded.com/wiki/NEC_IR_Remote_Control_Interface_with_8051

The logic 1 is sent with a period of 2.25 ms which consists of a 562.5 Us pulse burst followed by 1.6875 ms space.

The logic 0 is sent with a period of 1.25 ms which consists of a 562.5 Us pulse burst followed by 562.5 Us space between them.

A total of 67.5 ms are required for transmission of the message frame. Thr four bytes of data buts are sent as least significant bits first.

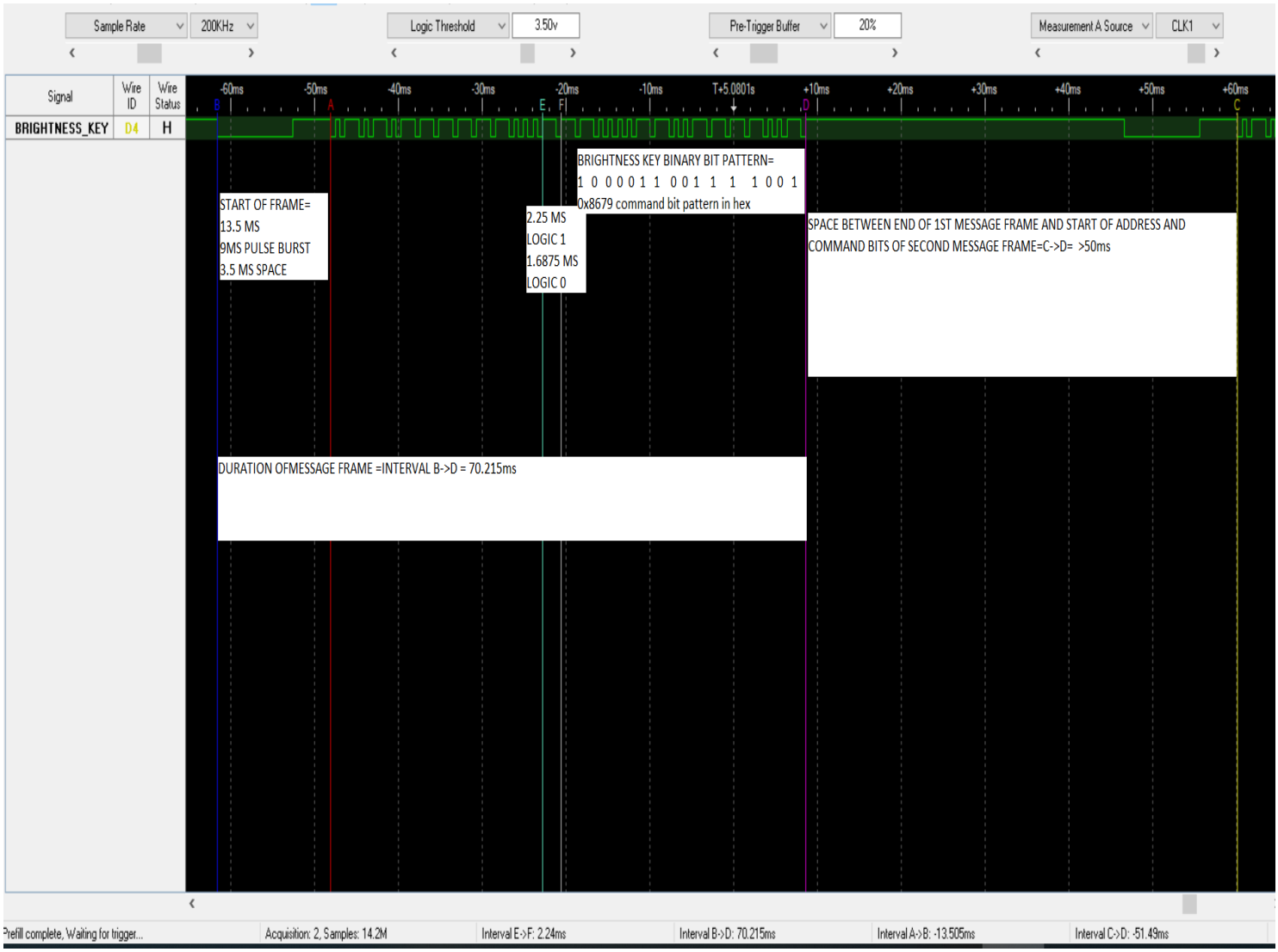


FIG 3.2 NEC MESSAGE FRAME AS OBSERVED ON THE LOGIC ANALYZER

For this driver I have made use of 2 interrupts functions, one is used for detecting the pulses and the other is used for detecting the pulse widths. As observed on the logic analyzer that each successive message is transmitted after 50 ms. Hence, the timer0_isr() __interrupt 1 function which is triggered after every 1ms period, has a variable that maintains the milliseconds count. This variable is reset after it is greater than 50ms period.

The external hardware interrupt void externalIntr0_ISR() __interrupt 0 is triggered on every button press of the remote as it is configured to trigger on the falling edge of the pulses. This interrupt function when triggers increment a variable count pulse count to maintain the count of the pulses which are being received. This function resets the timer value once the pulse width is determined. If the pulse width is greater than 50 ms then the start of the frame is counted which is not accumulated as part of the new bit pattern and only consists of address and command bits. If the pulse count is greater than 0 and less than 32, the bit pattern is formed which is stored in a new variable after the 32 bits are over to determine a new key of the remote press.

The issue I faced during decoding the IR pulses was that the pulses were received for their sensor were different each time I added the newkey bit pattern in a variable in the main loop and printed on putty through paulmon. Also, taking inputs for reprogrammable functionality which use getchar and putchar functions to transmit the message serially went on continuously. As a result, even though the user pressed the remote button key, it waited for the user to enter the input for the getchar function before executing the operation the remote key was mapped to.

This was resolved by doing that operation in the ISR. When the corresponding key is pressed, the microcontroller performs operations real-time depending on the operations desired.

In addition to using these functions, application code consisted of remote key function macros that were found out by calculating the binary values of remote pattern through the logic analyzer. The driver includes separate header files for remote key and remote functions which defines macros and variables for all register addresses and register bits.

2.3.2 GRAPHICAL LCD DRIVER

The graphical LCD works on SPI protocol t transmit messages to the screen. Whenever the D/C line is low, the data byte is selected and the following data bits are stored in the display data ram. After each data byte the address increments automatically.

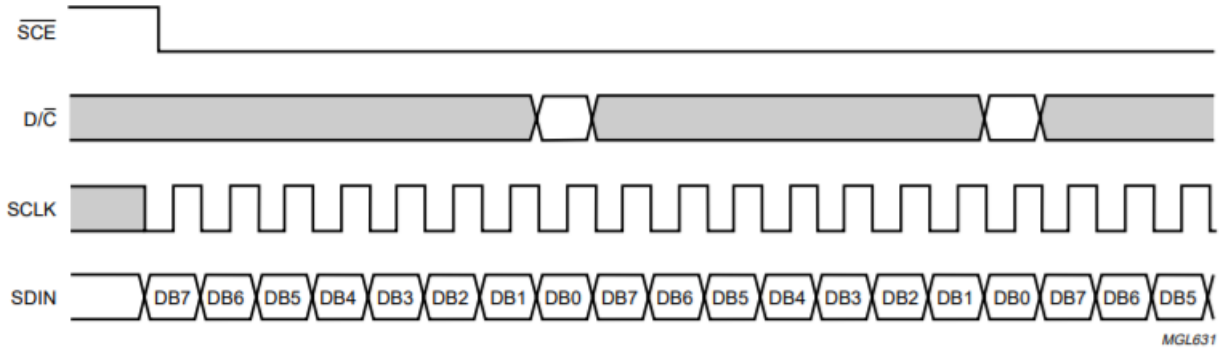


FIG 3.3 TIMING DIAGRAM OF SPI PROTOCOL from fig 8 of NOKIA 5110 datasheet

The data or instruction field is sent byte by byte from MSB to LSB.

The serial interface is not initialized when SCE/ is high. The clock pulses have no effect on the serial interface when SCE is high. A negative edge on SCE enables serial interface and it signals the start of data transmission.

Data in pulses get sampled at every positive edge of the clock. After completion of one byte of data transfer, the next byte of data transfer depends on SCE/ pin. It stays low even after the last byte of data transfer and serial data transfer continues with D/C pin determining whether the data byte transfer is a command instruction or data instruction. A reset in the

middle of transmission can interrupt the serial data transmission in between and the data display address is cleared from any data present. If SCE/ pin is low after the positive edge of the reset pin, the serial transmission continues again with MSB of data byte being ready to be send. In SPI the bytes written and read are MSB first. I verified the signals on logic analyzer to see for the correct sequence of data transmission.

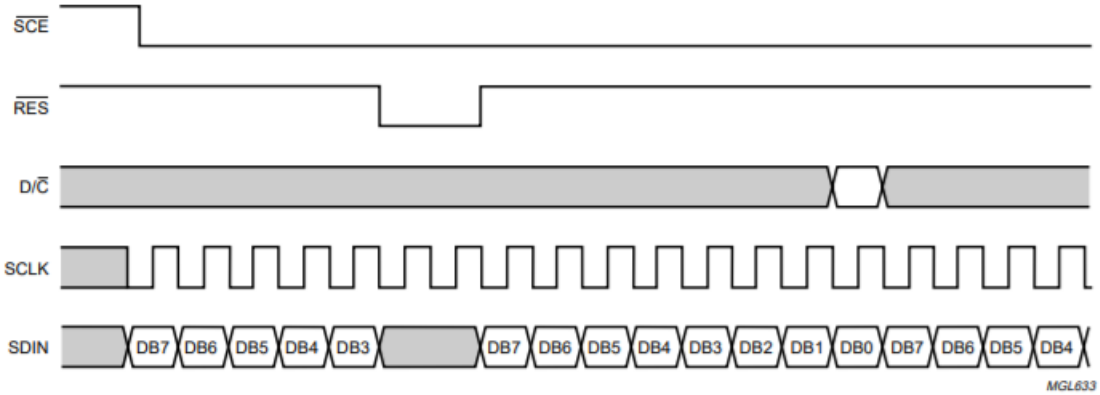


FIG 3.4 RESET PULSE TIMING DIAGRAM from fig 9 of NOKIA 5110 datasheet

Immediately after the power on the contents of ram internal registers is undefined. The reset pin is needed to be applied at the start to avoid the possibility of a great damage. After reset the LCD goes on the following state:

*Power-down mode
*Blank display
*The x and y fields of addresses are in the reset state
*The addressing is horizontal
*Bias system is set to zero.

The initializing sequence forms an important part to work with the LCD display. Initially, we send a reset signal, and the chip is disabled. We activate the chip and set it addressing as horizontal addressing. Following that command is sent to adjust the led contrast, voltage bias and keep horizontal addressing in use. This is followed by sending the command instruction to activate all segments of the display, clear the pixels and blank the display, and finally, the display is set to normal and the cursor is brought to the start. All these initialization routines are sent to the instruction register through the command() function when D/C pin is made low. The LCD did not start normally when initialized. Later on changing the backlit led contrast from 0XB2 to 0XBF the LED displayed things as required.

After this initialization, the LCD display started normally and sent data bits one by one including the single characters and string of characters at the specified location from the lookup table. However, when I tried to display images by specifying the image bit map, the images get displayed in an incorrect position with only half of it getting displayed. The lower half of the image got displayed on the upper half of LCD and the lower half of LCD showed a distorted pixeled picture.

The lookup table consists of only those characters which can be displayed on the LCD. The other characters form ASCII value beyond the range 32 to 126 are printed as a backslash in the function charDisp(). The row and column together combine to make up 504 pixels on the graphical LCD. 84 pixels on the x-axis and 6 pixels on the y-axis. The write() function sends all the instructions and data on the data line. The data is sent one byte at a time when with very clock transition from low to high. The setcursor() and setpixel() function determine the position of the pointer on the ddram address and allclear() clear the LCD display.

INSTRUCTION	D/$\overline{C}$	COMMAND BYTE								DESCRIPTION
		DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
(H = 0 or 1)										
NOP	0	0	0	0	0	0	0	0	0	no operation
Function set	0	0	0	1	0	0	PD	V	H	power down control; entry mode; extended instruction set control (H)
Write data	1	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	writes data to display RAM
(H = 0)										
Reserved	0	0	0	0	0	0	1	X	X	do not use
Display control	0	0	0	0	0	1	D	0	E	sets display configuration
Reserved	0	0	0	0	1	X	X	X	X	do not use
Set Y address of RAM	0	0	1	0	0	0	Y_2	Y_1	Y_0	sets Y-address of RAM; $0 \le Y \le 5$
Set X address of RAM	0	1	X_6	X_5	X_4	X_3	X_2	X_1	X_0	sets X-address part of RAM; $0 \le X \le 83$
(H = 1)										
Reserved	0	0	0	0	0	0	0	0	1	do not use
	0	0	0	0	0	0	0	1	X	do not use
Temperature control	0	0	0	0	0	0	1	TC_1	TC_0	set Temperature Coefficient (TC_x)
Reserved	0	0	0	0	0	1	X	X	X	do not use
Bias system	0	0	0	0	1	0	BS_2	BS_1	BS_0	set Bias System (BS_x)
Reserved	0	0	1	X	X	X	X	X	X	do not use
Set V_{OP}	0	1	V_{OP6}	V_{OP5}	V_{OP4}	V_{OP3}	V_{OP2}	V_{OP1}	V_{OP0}	write V_{OP} to register

Table 2 Explanations of symbols in Table 1

BIT	0	1
PD	chip is active	chip is in Power-down mode
V	horizontal addressing	vertical addressing
H	use basic instruction set	use extended instruction set
D and E		
00	display blank	
10	normal mode	
01	all display segments on	
11	inverse video mode	
TC_1 and TC_0		
00	V_{LCD} temperature coefficient 0	
01	V_{LCD} temperature coefficient 1	
10	V_{LCD} temperature coefficient 2	
11	V_{LCD} temperature coefficient 3	

FIG 3.5 Set of instruction or commands from table 4 of NOKIA 5110 datasheet

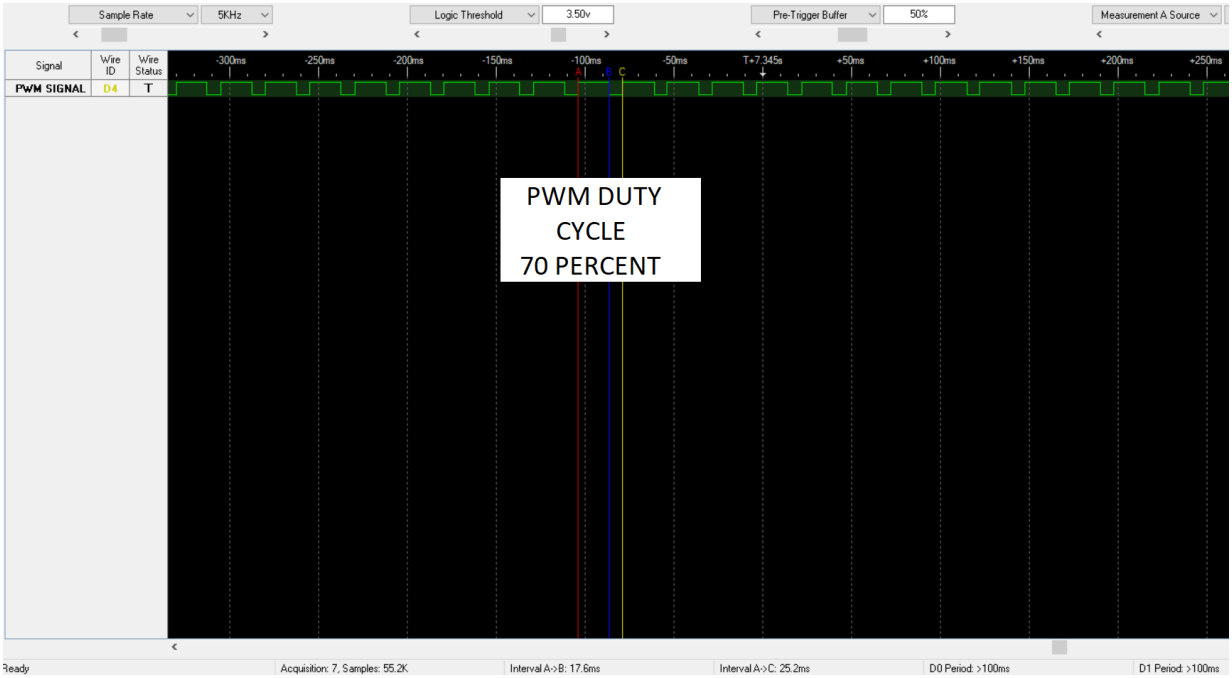
2.3.3 DC MOTOR CONTROL USING PWM

DC motor converts electrical energy into mechanical energy. The direction of the dc motor can be reversed by changing the direct current through the motor. This can be done by using a motor driver which has a H- bridge circuit as is present in Motor driver L293d.
The change in voltage applied to the motor results in changing the speed with the motor rotates, for which we use the PWM technique.

The speed control pin which is also called the enable pin in motor driver l293d, is connected to pin 1.3 of microcontroller, where we also generate PWM through Programmable counter array, which is set at module 0 to perform Pulse width modulation where CMOD is used to set up the PCA timer, CCAP0X is used to load the values in the timer for PWM control and CR is used to run the timer. The PCA is used to generate 8 bit PWM signals. The duty cycle of the output depends on the value which is loaded in the high byte. the values of the duty cycle can be calculated using the equation

$$CCAPnH = 256(1-\text{Duty Cycle})$$

where the duty cycle is a fraction and CCAPnH is a 8-bit integer.



FIR 3.6 PWM output from pin INT0 observed on logic analyzer

Initially, in the PWM Based dc motor control, we load the value that is to be loaded in the PWM based timer. The value is loaded for a PWM period of 25 milliseconds. we use Timer interrupt 2 in capture compare mode. Initially, since old 8051 had only 2 timers of which only one timer could be used, and timer 0 was being used for remote control pulse width measurements, I had difficulty in deciding how PWM based control could be possible, without using PCA. Then I looked at the datasheet of Atmel 89c51 and found that it also

contains an extra timer that could be used. I used this timer for my PWM based operation.

At the start of the operation, the speed and duty cycle is set to zero. As the remote upper key is pressed, the speed is incremented by changing the on and off load values of PWM. Until the duty cycle value reaches 100 percent the speed is continuously incremented by increasing the count value. Similarly for decreasing the speed of PWM, the on and off value is changed accordingly in the Set_DutyCycle_to() function which is reflected in the speed of the motor as a result of interrupt trigger after every 25 milliseconds. The direction of the DC motor is controlled through motor direction pins p1.0 and pin 2.0 which are toggled for changing the direction. The replay key and okay key on the remote are used to control the direction of the PWM based dc motor rotation.

The period variable represents the maximum duty cycle of PWM, which is changed from the following calculation:

```
float period = 65535 - PWM_Period;
ON_Period = ((period/100.0) * duty_cycle);
OFF_Period = (period - ON_Period);
ON_Period = 65535 - ON_Period;
OFF_Period = 65535 - OFF_Period;
```

The dc motor can also be stopped from running by right arrow key and started again by using the left arrow key.

One problem encountered was that the PWM started and stopped while running and was observed on the logic analyzer that it stopped only during the time timer 0 interrupt was operational that is for a period of 1 millisecond. Later on, it was found that I had initially made PWM control to toggle in timer 0 interrupt before I used timer 2 interrupt because of which it showed a stop-start behavior.

Also, one more problem was that the dc motor should run continuously until it is stopped by decreasing the speed or stopped by pressing the stop button (right arrow key) on the remote control. If I kept it in the same while (1) loop as my user input for UART, the motor would stop for a brief period before restarting after some key was pressed. For this, i initialized the motor speed control in the while loop of the getchar function. as a result of that change, the motor also ran during the time the program was waiting for input from the user.

*2.3.4 4*3 Keypad*

The 4*3 Keypad is used for taking input from the user along with the keyboard input which is required for the reprogrammable functionality of the remote. In this driver code, we make all rows zero initially and all columns as one. Whenever a button is pressed column and the row corresponding to the button gets shorted and hence the column related to that row becomes zero.

This is how we get the column number. To find the rows, we have created four functions called row1(), row2(), row3() and row4().

These functions are called corresponding to the column which switched to zero on the press of the button. In these functions, we initialize the columns as zero and make the rows as 1. Whenever a button on the keypad is pressed the corresponding row on which the button lies becomes zero. This is how we are able to find the right row and column of the button pressed. The same procedure is followed for every button press and we are able to get the row and column of this button pressed.

The issue with Keypad was the effect of debouncing. Also, when observed on the logic analyzer the buttons were in the floating state for which the resistors of value 4.7 k ohms were connected in hardware.

This, however, did not solve the problem of debouncing. The microcontroller has limited pins as a result of which the other peripherals like dc motor connected to the same pin showed uneven behavior. I made the software adjustment by making the program to wait before the key was stopped being pressed and it showed better results when the key pressed was displayed on the GCD and the seven-segment display.

2.3.1 Seven segment Display

Seven segment display consists of a number of LED segments, which need to switch on individually to display different patterns or numerical digits. It can be a common anode or common cathode. WE use the common anode 7 segment where the positive elements are left alone and the led segments which are switched on are made 0. For example, the binary code for digit zero will be 11000000 that is 0xC0 in hexadecimal. This needs to be sent to the port pin where seven segment display is connected to display the correct number pressed. In the project, seven segment display is used to display all the numbers or characters which are taken as input from the user.

I tested this by checking if all led segments are being pulled up to VCC.
After connecting all the segments in the proper order, I ran a for loop to test of numbers from 0 to 9 to see if all of the led segments worked properly.

2.4 Testing Process

Throughout the project, I used an incremental approach in adding and testing components. I took small steps by testing the hardware for the particular peripheral by checking the continuity and its connections before going onto the software. I checked if the components worked separately before integrating all the elements of the project. I ran a basic working code for all the components to test its operation. The signals received by all components were tested on the logic analyzer.

As previously mentioned, I went through a sequential approach in collecting and integrating my peripherals which also helped me study individual components in detail. Mostly the testing involved was manually done without any case of regression type testing. I tried to solve each problem through hardware and then jumped to software to handle

issues that could not be solved in hardware.

Testing of my project started from checking for the correct IR pulses received on a 2 pin IR receiver. Then, switching to TSOP, Testing the graphical lcd through normal initialization routines, then using keypad and seven-segment display to check for the correct output, and finally worked on PWM based dc motor generation.

3 RESULTS AND ERROR ANALYSIS

The system came together rather nicely and I was able to get the outputs of majority of things I planned to implement during the project presentation. Most of the debugging steps were defined in Section 2. However, some which are important part of error analysis is the logic analyzer outputs and component testing.

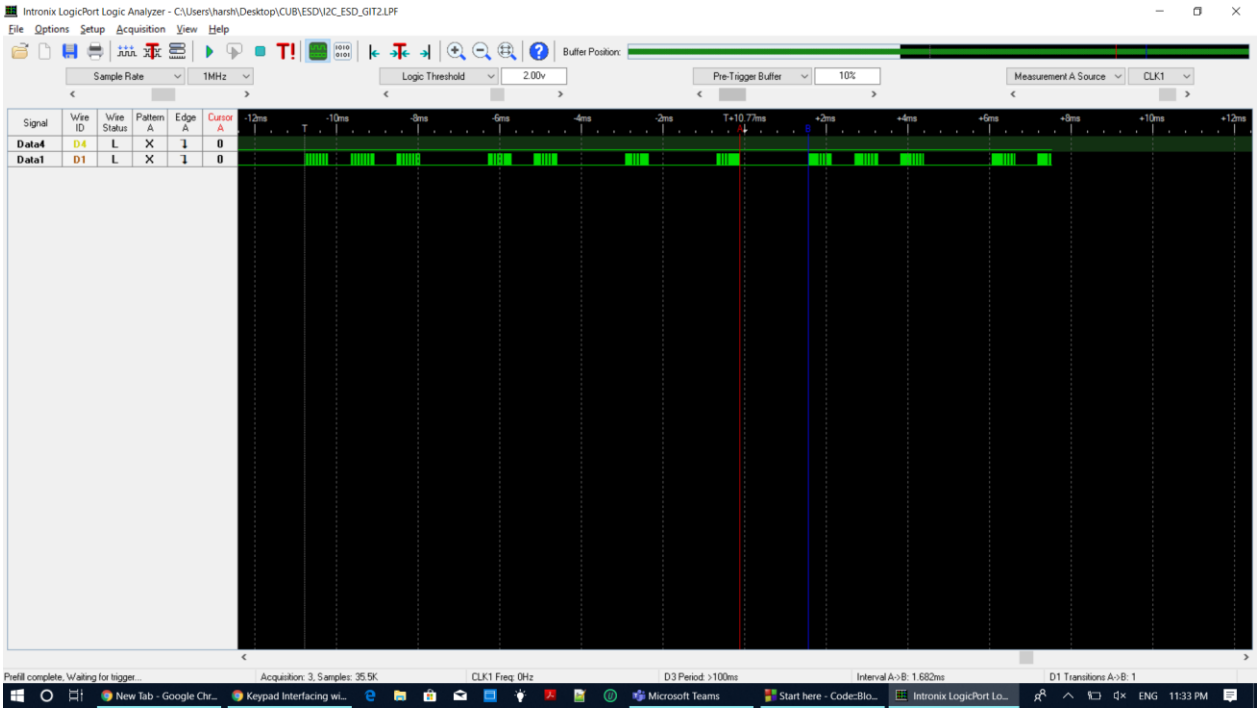


FIG 4.1 LOGIC ANALYZER OUTPUT FOR 2 Pin IR Receiver WHERE LOGIC 1 AND LOGIC 0 are stream of high and low pulses, something which TSOP filters out to make it a high or low pulse only

3.1 Component Testing

It is very essential before starting to work with the component that the component is thoroughly tested. I observed this in my project at the start when I was observing the pulses of IR. I observed that the TSOP sensor was faulty and gave outputs at very low logic threshold levels. When I tested it by connecting led across it through a transistor, I observed that the led also did not glow properly. On changing the component with other TSOP, I received outputs at correct threshold levels.

3.2 Paulmon

Paulmon uses timer 0 and timer 1 to calculate auto baud rates. Then it turns off timer 0 and uses timer 1 to generate the required baud rate. Therefore, in the initialization of timer 0 interrupt when I made the T register as TMOD = value rather than TMOD |= value, the led inside the interrupt function toggled but the printf statements did not print anything on putty. Therefore, actually the timer interrupt for IR remote which was used for calculating the pulse widths was not initialized and led was blinking because of paulmon enabled timer. By making this small change, the IR remote detected proper pulses too.

3.3 Logic Analyzer

The use of logic analyzer was very beneficial in tracking errors with respect to IR sensors, keypad response, graphical lcd spi bit banging. It helped in analyzing things which were difficult to debug without knowing the signal response from the device.

4 CONCLUSION

The project was a great learning experience to deal with software, the integration, and the functioning of several hardware peripherals working together as part of a single system.

It helped me learn about several aspects like how infrared communication happens between various devices in our day to day life and how are we able to interact with so many devices present around us by simply a press of a button. It helped me understand the working of graphical LCD and how SPI bit-banging helped in data transfer between the master and the slave. It also helped me understand in detail the working of dc motor and how is the motor driver useful in driving the motor, through the built in h- bridge driver, and how we need to be careful about the variables we use in our systems, specifically in case of interrupts.

I learned a lot of soft skills throughout the project. The process of incremental work worked wonders throughout the project. Although, I wanted to work on a project in a group, doing the project alone helped me in managing time and working independently. It helped me be more responsible for the entire work I took ownership of. Working with independent pieces of the project proved to be fruitful at the end of the project helping me learn about key aspects of the system's hardware and software techniques.

I was successfully able to work on the majority of the things I planned to do at the beginning of the semester and gave me great satisfaction in successfully implementing things. However, some aspects like graphical LCD and reprogrammable functionality could be improved upon in future projects with shift to arm or other advanced controllers.

5 FUTURE DEVELOPMENT IDEAS

This project involved interfacing various peripherals to limited pins of the microcontroller. In future projects, the 8051 microcontrollers can be replaced by high-end microcontrollers having an adequate number of pins to run all the peripherals smoothly.

Also, the functionality of the project can be enhanced by displaying the key pressed on the

remote in an image form on the graphical LCD. The reprogrammable functionality can be expanded by giving control of it to the remote itself. In this case is when a particular sequence of a button press is done, the user with the remote alone can be allowed to reprogram the functionality of the remote. The set of operations he can change can be visible on the LCD. Also, the keypad can be used as security to avoid change of operations using different keys of the remote by unauthorized users. The project can also be used in home automation to control the fan, light and other electrical devices that operate on Infrared radiations. The possibilities of adding new elements to this project are immense and it is upon the interest of the user to explore it.

6 ACKNOWLEDGEMENTS

I would like to thank Prof. Linden McClure, the course instructor for his amazing course which helped me work on a lot of embedded and c concepts throughout the semester and his advice to work on pieces in a structured manner by incrementally increasing the complexity of the project.

I also wish to thank Tanmay Chaturvedi, my TA for the specific course to help me give motivation and inspiration to work harder on the project and see things come through.

To my Family members for their continuous support and helping me stay sane during tough times

Finally, I would like to acknowledge and thank the authors of the various sources cited in the references below.

7 REFERENCES

[1] https://www.ardumotive.com/how-to-use-a-keypad-en.html
[2] https://www.snrelectronicsblog.com/8051/8051-based-remote-for-home-appliance/http://www.circuitstoday.com/nokia-graphic-lcd-display-8051
[3] http://www.circuitstoday.com/nokia-graphic-lcd-display-8051
[4] http://www.vishay.com/docs/82491/tsop382.pdf TSOP38238 DATASHEET
[5] https://cdn.sparkfun.com/datasheets/Robotics/M260.pdf T
[6] https://circuitdigest.com/microcontroller-projects/keypad-interfacing-with-8051-microcontroller
[7] https://circuitdigest.com/microcontroller-projects/7-segment-display-interfacing-with-8051
[8] http://skpang.co.uk/catalog/images/lcd/graphic/docs/User_Manual_ET_LCD5110.pdf
[9] https://www.electronicwings.com/8051/dc-motor-interfacing-with-8051
[10] https://exploreembedded.com/wiki/NEC_IR_Remote_Control_Interface_with_8051
[11] http://www.diymodules.org/eagle-search?text=LCD
[12] https://www.sparkfun.com/datasheets/Sensors/Infrar

ed/tsop382.pdf
[13] https://stackoverflow.com/questions/39625579/c-overflow-in-implicit-constant-conversion-woverflow/39625680
[14] https://sites.google.com/site/controlandelectronics/pwm-tutorial-using-8051
[15] http://www.ti.com/lit/ds/symlink/l293d.pdf
[16] http://ww1.microchip.com/downloads/en/appnotes/s72052.pdf

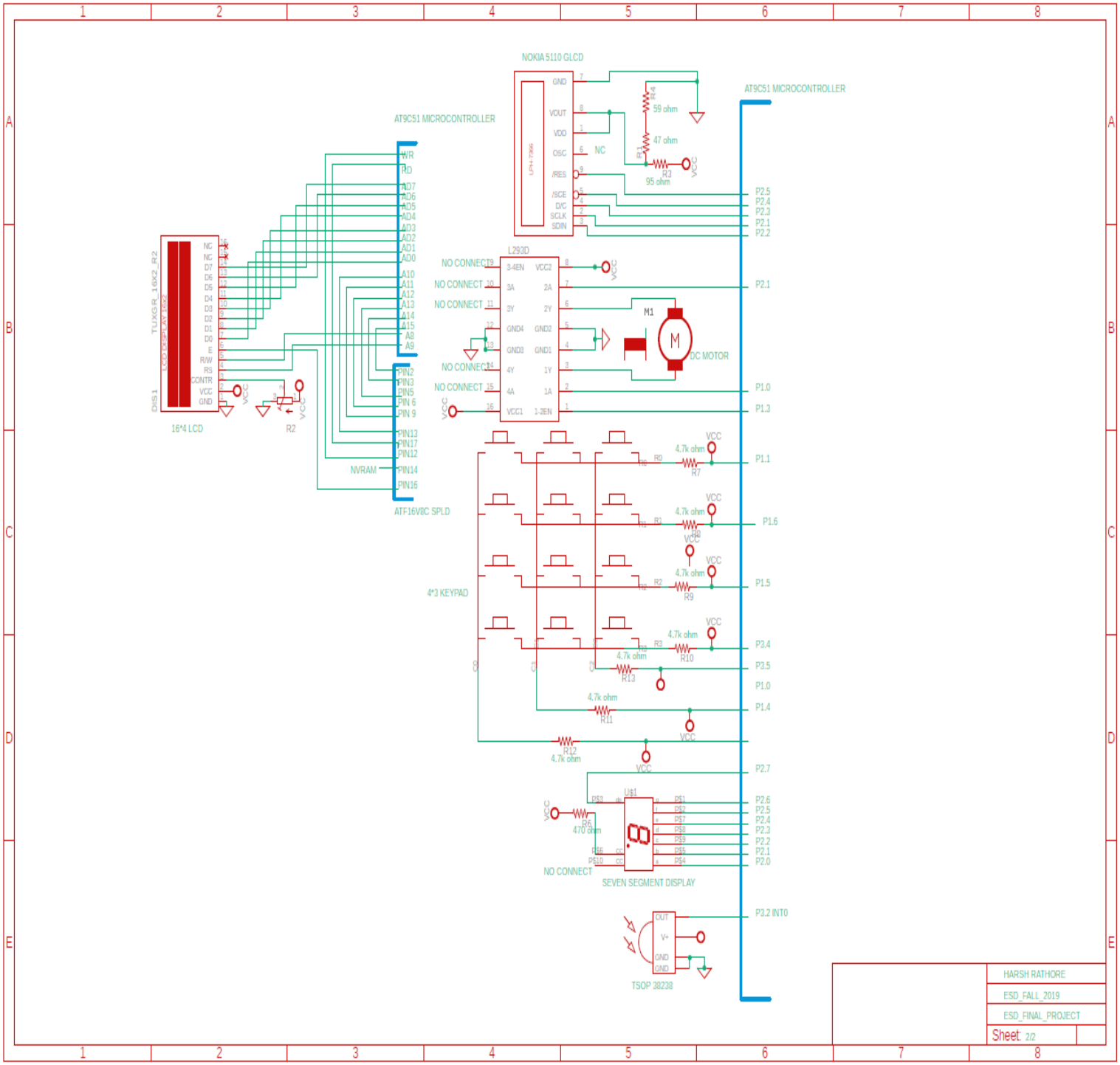
8 APPENDICES

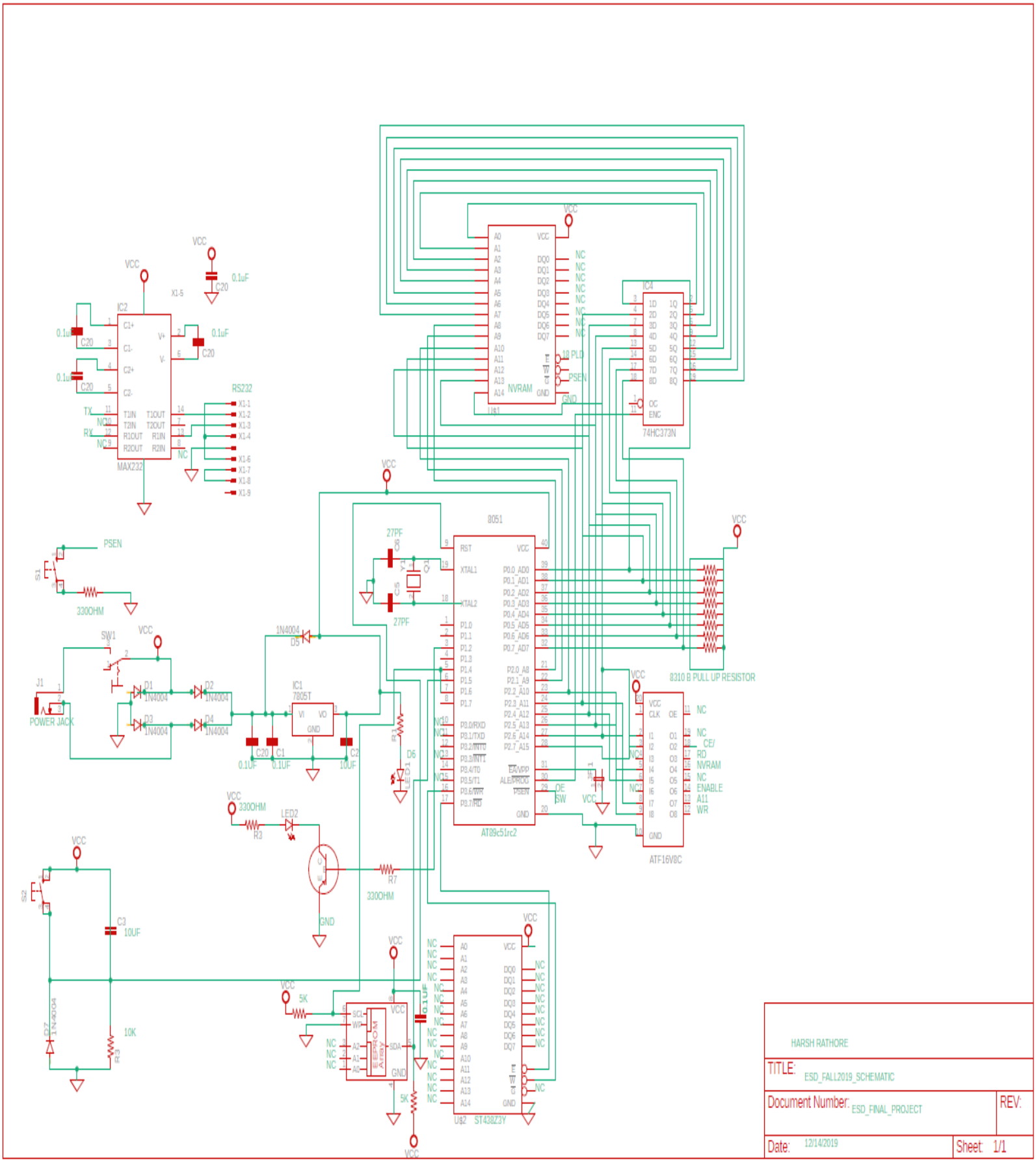
Several appendices have been attached to this report in the order shown below.

8.1 Appendix - Bill of Materials

Part Description	Source	Cost
2 PIN IR RECEIVERS AND TRANSMITTERS (10 EACH)	www.amazon.com	$4.9
NOKIA 5110 GLCD	(9.95$) Borrowed from a friend's Components Bag	$0.00
TSOP 38328 (6)	Sparkfun www.sparkfun.com	$11.7
PERF BOARD	CU Boulder Electronics Centre	$4.00
L293D MOTOR DRIVER	CU Boulder Electronics Centre	$1.00
4.7K Pull-up Resistors, (10)	Borrowed from a Friend's Components bag	$0.00
100 (1) and 50(2) Resistors	Borrowed from a Friend's Components bag	$0.00
DC MOTOR	CU Boulder Electronics Centre	$1.95
3*4 KEYPAD	Sparkfun www.sparkfun.com	$4.50
20-pin IC Holder	CU Boulder Electronics Centre	$1.00
Pin Strip Header, Single Row, Contacts	Embedded Systems Lab	$0.05
Pin Strip Header, Single Row,	Embedded Systems Lab	$0.20
TOTAL		**$29.30**

8.2 Appendix - Schematics




HARSH RATHORE
TITLE: ESD_FALL2019_SCHEMATIC
Document Number: ESD_FINAL_PROJECT
REV:
Date: 12/14/2019
Sheet: 1/1

8.3 Appendix - Firmware Source Code

ESD FINAL PROJECT SOURCE CODE

MAIN.C

```
#include <stdio.h>
#include <stdlib.h>
#include<at89c51ed2.h>          ///include library functions needed FOR input
#include <mcs51/8051.h>
#include <stdint.h>
#include "remote_key.h"
#include "LCD.h"
#include "GLCD.h"
#include "Dc_motor_pwm.h"
#include "Keypad.h"
#include "Remote_functions.h"
#include "Sevensegment.h"
volatile uint32_t bitPattern=0,newKey=0;          ///bitpattern, newkey is set t zero as initialized as extern
volatile uint8_t motor_direction_pin1=1;
volatile uint8_t motor_direction_pin2=0;

unsigned char seven_seg[]= {0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90};    ///hex values
for displaying 0 to 9
unsigned char seven_seg_alp[]= {0x88,0x80,0xC6};
#define Bulb P1_2
#define ir P1_7


volatile uint32_t keyCode;                ///keypad key
volatile char mode;                     ///mode for re-programmable function
volatile char input1;
extern volatile int8_t power_key;
extern volatile int8_t back_key;          ///back key, power key and back key on remote
extern volatile int8_t home_key;
volatile uint8_t motor;
volatile uint8_t remote_flag=0;

volatile uint8_t keypad=0;
volatile uint16_t OFF_Period=0, ON_Period=0,Speed=0;

#define M1_Pin1                 P1_0              /// Motor Pin 1 set pin
#define M1_Pin2                 P2_1              /// Motor Pin 2  set pin



void msdelay(unsigned int time)           /// Function for creating delay in milliseconds.
{
   unsigned i,j ;
   for(i=0; i<time; i++)
      for(j=0; j<1275; j++);
}

/// Function to provide delay of 1ms at 11.0592 MHz
```

```
void delay2(unsigned int count)
{
   int i,j;
   for(i=0; i<count; i++)
      for(j=0; j<112; j++);
}

void delay1(unsigned int count)                    ///delay for glcd
{
   int i,j;
   for(i=0; i<count; i++)
      for(j=0; j<112; j++);
}

void switchs(char keys)
{
   printf("\n\r mode=%c\n\r",mode);
   switch(keys)
   {
   case('A'):
   {
      P0=seven_seg_alp[0];
      setCursor(0,0);

      stringDisp(" POWER BUTTON     SELECTED");              ///Print string to GLCD
      if (mode=='1')
         power_key=1;
      else if (mode=='2')                          ///to find the particular key THE OPERATION needs to be
mapped to
         power_key=2;                              ///MODE1 = power, MODE2 = BACK KEY, MODE3 =
HOME KEY
      else if(mode=='3')
         power_key=3;
      break;
   }
   case('B'):
   {
      P0=seven_seg_alp[1];
      setCursor(0,0);
      ///Print string to LCD
      stringDisp("  BACK BUTTON     SELECTED");              ///Print string to GLCD
      if (mode=='1')
         back_key=1;
      else if (mode=='2')                          ///to find the particular key THE OPERATION needs to be
mapped to
         back_key=2;
      else if(mode=='3')                           ///MODE1 = power, MODE2 = BACK KEY, MODE3 =
HOME KEY
         back_key=3;
      break;
   }
   case('C'):
   {
```

```
      P0=seven_seg_alp[2];
      setCursor(0,0);
      ///Print string to LCD
      stringDisp("  HOME BUTTON     SELECTED");              ///Print string to GLCD
      if (mode=='1')
         home_key=1;
      else if (mode=='2')                          ///to find the particular key THE OPERATION needs to be
mapped to
         home_key=2;
      else if(mode=='3')                           ///MODE1 = power, MODE2 = BACK KEY, MODE3 =
HOME KEY
         home_key=3;
      break;
   }
   default:
   {
      printf_tiny("\n\rINVALID INPUT\n\r");
   }

   }


}

void main()
{
   M1_Pin1=1;
   M1_Pin2=0;
   lcdinit();                  ///LCD INITIALIZATION
   lcdbusywait();
   lcd_instruction_write(0x40);   ///setting cgram address
   lcdbusywait();
   lcdputch(0x00);              ///the custom  characters first row
   lcdbusywait();
   lcd_instruction_write(0x41);   ///first custom character row 1
   lcdbusywait();
   lcdputch(0x1B);              ///the custom  characters second row
   lcdbusywait();
   lcd_instruction_write(0x42); ///first custom character row 2
   lcdbusywait();
   lcdputch(0x00);             ///the custom  characters third row
   lcdbusywait();
   lcd_instruction_write(0x43); ///first custom character row 3
   lcdbusywait();
   lcdputch(0x04);             ///the custom  characters fourth row
   lcdbusywait();
   lcd_instruction_write(0x44); ///first custom character row 4
   lcdbusywait();
   lcdputch(0x04);           ///the custom  characters fifth row
   lcdbusywait();
   lcd_instruction_write(0x45); ///first custom character row 5
   lcdbusywait();
   lcdputch(0x11);           ///the custom  characters sixth row
```

```
    lcdbusywait();
    lcd_instruction_write(0x46); ///first custom character row 6
    lcdbusywait();
    lcdputch(0x0E);          ///the custom  characters seventh row
    lcdbusywait();
    lcd_instruction_write(0x47);
    lcdbusywait();
    lcdputch(0x00);
    lcdbusywait();
    lcdgotoxy(0,7);         ///print the custom character at various locations
    lcdbusywait();
    lcdputch(0);            ///print the 0th custom character
    lcdgotoxy(1,7);
    lcdbusywait();
    lcdputch(0);
    lcdgotoxy(2,7);
    lcdbusywait();
    lcdputch(0);
    lcdgotoxy(3,7);
    lcdbusywait();
    lcdputch(0);

    lcdgotoxy(0,0);
    lcdbusywait();
    lcdputstr("1.MOTOR A.POWER");   ///display the re-programmable functions for the different keys on
lcd for ease of input
    lcdgotoxy(1,0);
    lcdbusywait();
    lcdputstr("2.LIGHT B.BACK");
    lcdgotoxy(2,0);
    lcdbusywait();
    lcdputstr("3.GLCD  C.HOME");
    lcdbusywait();
    lcdgotoxy(3,1);            ///print the custom character at different location
    lcdbusywait();
    lcdputch(0);
    lcdgotoxy(3,3);
    lcdbusywait();
    lcdputch(0);
    lcdgotoxy(3,5);
    lcdbusywait();
    lcdputch(0);
    lcdgotoxy(3,7);
    lcdbusywait();
    lcdputch(0);
    lcdgotoxy(3,9);
    lcdbusywait();
    lcdputch(0);
    lcdgotoxy(3,11);
    lcdbusywait();
    lcdputch(0);
    lcdgotoxy(3,13);
    lcdbusywait();
```

```
   lcdputch(0);
   lcdgotoxy(3,15);
   lcdbusywait();
   lcdputch(0);
   delay(1000);
   Initialize_LCD();

   Bulb = 0;            /// Turn OFF the relay during Init
   IR_RemoteInit();  /// Initialize the INTO and Timer0 for decoding the IR pulses

   Timer_init();     ///timer init for pwm motor
   Motor_Init();
   msdelay(100);
   setPixel(0,0);

   while(1)
   {

       printf_tiny("Enter the operation which needs to mapped to a specific button\n\r");
       printf_tiny("\n\r1=MOTOR\n\r");
       printf_tiny("\n\r2=LIGHT\n\r");
       printf_tiny("\n\r3=GLCD\n\r");
       printf_tiny("\n\rM=MENU\n\r");
       mode=getchar();                        ///TAKE INPUT IN FORM A CHARACTER FORM USER
       putchar(mode);
       if(mode!=0)
       {

           if(mode=='1' || keypad==1)          ///if  mode is 1 or keypad pressed is 1 display on 7 segment
display
           {
              P0=seven_seg[1];
              allClear();
              setCursor(0,0);
              ///Print string to LCD
              stringDisp("MOTOR SELECTED");

printf_tiny("\n\r~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n\r");
              printf_tiny("\n\rKEY to which the operation needs to be mapped to\n\r");
              printf_tiny("\n\rA=POWER\n\r");
              printf_tiny("\n\rB=BACK\n\r");
              printf_tiny("\n\rC=HOME\n\r");

printf_tiny("\n\r~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n\r");
              input1=getchar();                        ///TAKE INPUT IN FORM A CHARACTER FORM USER
              putchar(input1);
              switchs(input1);                  ///when motor is selected go to switchs, to find the particular key it
needs to be mapped to

           }
           else if(mode=='2' || keypad==2)
           {
              P0=seven_seg[2];
```

```
            allClear();
            setCursor(0,0);
            ///Print string to LCD
            stringDisp("LIGHT SELECTED");

printf_tiny("\n\r~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n\r");
            printf_tiny("\n\rKEY to which the operation needs to be mapped to\n\r");
            printf_tiny("\n\rA=POWER\n\r");
            printf_tiny("\n\rB=BACK\n\r");
            printf_tiny("\n\rC=HOME\n\r");

printf_tiny("\n\r~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n\r");
            input1=getchar();                  ///TAKE INPUT IN FORM A CHARACTER FORM USER
            putchar(input1);
            switchs(input1);           ///when light is selected go to switchs, to find the particular key it
needs to be mapped to

         }
         else if(mode=='3' || keypad==3)
         {
            P0=seven_seg[3];
            allClear();
            setCursor(0,0);
            ///Print string to LCD
            stringDisp(" GLCD SELECTED");

printf_tiny("\n\r~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n\r");
            printf_tiny("\n\rKEY to which the operation needs to be mapped to\n\r");
            printf_tiny("\n\rA=POWER\n\r");
            printf_tiny("\n\rB=BACK\n\r");
            printf_tiny("\n\rC=HOME\n\r");

printf_tiny("\n\r~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n\r");
            input1=getchar();                  ///TAKE INPUT IN FORM A CHARACTER FORM USER
            putchar(input1);
            switchs(input1);          ///when glcd is selected go to switchs, to find the particular key it
needs to be mapped to

         }
         else if(mode=='M')            ///to print the menu again
         {
            printf_tiny("\n\r1=MOTOR\n\r");
            printf_tiny("\n\r2=LIGHT\n\r");
            printf_tiny("\n\r3=\n\r");
         }
         else
         {
            printf_tiny("\n\rINVALID INPUT\n\r");
         }
      }
   }
}
```

```
int putchar (int c)
{
   while (!TI);
                                /// wait for TX ready, spin on TI
   SBUF = c;                    /// load serial port with transmit value
   TI = 0;                      /// clear TI flag
   return 0;
}

int getchar ()
{
   while (!RI)
   {   /// Increment Duty cycle i.e. speed by 10% for Speed_Inc Switch
      if( newKey== up_arrow || newKey==up_arrow1)  ///if the button pressed on remote is up
      {
         if(Speed < 100)              ///if pwm speed is less than 100
            Speed += 10;              ///increase the pwm speed by 10 percent
         Set_DutyCycle_To(Speed);     ///set the new value of speed for the dc motor
         while(newKey==up_arrow);     ///wait till up arrow is being pressed
         delay(200);
      }

      /// Decrement Duty cycle i.e. speed by 10% for Speed_Dec Switch
      if(newKey == down_arrow || newKey==down_arrow1) ///if the button pressed on remote is up
      {
         if(Speed > 0)                ///if pwm speed is greater than 0
            Speed -= 10;              ///decrease speed by 10 percent
         Set_DutyCycle_To(Speed);     ///set the new value of speed for the dc motor
         while(newKey == down_arrow);    ///wait till down arrow is being pressed
         delay(200);
      }
      /// Change rotation direction for Change_Dir Switch
      if(newKey == replay|| newKey == replay1 || newKey==ok_key || newKey==ok_key1) ///if the button
pressed on remote is ok or replay
      {
         M1_Pin1 = !motor_direction_pin1;  ///toggle the pins which set the direction of pwm dc motor
         M1_Pin2 = !motor_direction_pin2;
         while(newKey == replay);     ///wait till replay is being pressed
         delay(200);
      };


   }                            /// wait for RI ready, spin on RI
   /// wait for character to be received, spin on RI
   RI = 0;                                  /// clear RI flag
   return SBUF;                 /// return character from SBUF
}
```

DC_MOTOR_PWM.C

```
#include "Dc_motor_pwm.h"
#include <stdio.h>
```

```
#include <stdlib.h>
#include<at89c51ed2.h>          ///include library functions needed for dc motor control
#include <mcs51/8051.h>
#include <stdint.h>
#define PWM_Out_Pin  P1_3        /// PWM Out Pin for speed control

#define PWM_Period 0xA5FD       ///PWM period of 25 ms set at the start

#define M1_Pin1                     P1_0      /// Motor Pin 1
#define M1_Pin2                     P2_1      /// Motor Pin 2
extern volatile uint16_t OFF_Period, ON_Period,Speed;  ///Variables defined for pwm speed control at
ever
                                    ///key pressed on the remote by changing the timer count for pwm in
th2 and tl2 register
volatile uint8_t duty_cycle=0;  ///duty cycle of pwm
int period = 65535 - PWM_Period;  ///100 percent period of pwm set as period


/*****************************Timer 2 INIT***********************************
THIS FINCTION CONTAINS ALL INIT REQUIREMENTS FOR PWM speed and direction
specification of the dc motor
TIMER 2 IS USED FOR GENERATING capture compare mode of timer 2 interrupt where value is
loaded as 81 hex in T2CON
*****************************************************************************/
void Timer_init()
{
   M1_Pin1=1;               ///m1 and m2 pin are used to set direction of motor pin as clockwise initially
   M1_Pin2=0;
   T2CON = 0x81;             ///to set the timer in capture compare mode
   PWM_Out_Pin=0;             ///initially set the output pin as 0


   TH2 = (0xA5);             /// 25ms timer value
   TL2 = 0xFD;


   ET2 = 1;                  /// Enable Timer 2 Interrupts
   TR2 = 1;                  /// Start Timer 2 Running
   EA = 1;                  ///enable global interrupt


}

/****************************************Timer 2
interrupt*******************************************
This function is used for changing the speed of dc motor on particular key presses on the remote
It changes the on and off time period by loading different values in the th2 and tl2 register
*****************************************************************************************
****************/
void Timer0_ISR() __interrupt 5
{
   M1_Pin1=motor_direction_pin1;      ///initially set the motor pin as 1 and 0 for clockwise rotation
   M1_Pin2=motor_direction_pin2;
```

```
   TF2=0;                        ///Timer overflow flag is reset on visit to the interrupt
   PWM_Out_Pin =!PWM_Out_Pin;        ///toggle outpin pin of motor

   if(PWM_Out_Pin)                  ///if pwm out pin is high we will load the value of th2 and tl2 as defined
by on period
   {

      TH2 = ((ON_Period)>>8);
      TL2 = ON_Period;
   }
   else                        ///if pwm out pin is low we will load the value of th2 and tl2 as defined by off
period
   {
      TH2 = (OFF_Period>>8);
      TL2 = OFF_Period;
   }

}


/// Calculate period & duty cycle for pwm by changing the on and off period
void Set_DutyCycle_To(float duty_cycle)
{
   float period = 65535 - PWM_Period;          ///period of 100 percent duty cycle is set as period at start
   ON_Period = ((period/100.0) * duty_cycle);   ///on period is period divided by 100 into duty
cycle,changed which every key pressed which corresponds to the specific function
   OFF_Period = (period - ON_Period);          ///off period is period found by subtracting on period from
period
   ON_Period = 65535 - ON_Period;             ///the value to be loaded in the register is 0xffff minus the
on period
   OFF_Period = 65535 - OFF_Period;           ///the value to be loaded in the register is 0xffff minus the
off period
}

/// Initially Speed & Duty cycle  of the motor is set to zero and is in clockwise direction
void Motor_Init()
{
   Speed = 0;          ///speed zero
   M1_Pin1 = 1;        ///clockwise direction
   M1_Pin2 = 0;
   Set_DutyCycle_To(Speed);
}
```

GLCD.C

```
#include "GLCD.h"

#include <stdint.h>


///lookup table defined for displaying the different characters on graphical lcd from 32 to 126 which can
be displayed on the nokia glcd
const unsigned char LookUpTable [][5] =
{
```

```
{ 0x00, 0x00, 0x00, 0x00, 0x00 },  // space
{ 0x00, 0x00, 0x2f, 0x00, 0x00 },   // !
{ 0x00, 0x07, 0x00, 0x07, 0x00 },   // "
{ 0x14, 0x7f, 0x14, 0x7f, 0x14 },   // #
{ 0x24, 0x2a, 0x7f, 0x2a, 0x12 },   // $
{ 0xc4, 0xc8, 0x10, 0x26, 0x46 },   // %
{ 0x36, 0x49, 0x55, 0x22, 0x50 },   // &
{ 0x00, 0x05, 0x03, 0x00, 0x00 },   // '
{ 0x00, 0x1c, 0x22, 0x41, 0x00 },   // (
{ 0x00, 0x41, 0x22, 0x1c, 0x00 },   // )
{ 0x14, 0x08, 0x3E, 0x08, 0x14 },   // *
{ 0x08, 0x08, 0x3E, 0x08, 0x08 },   // +
{ 0x00, 0x00, 0x50, 0x30, 0x00 },   // ,
{ 0x10, 0x10, 0x10, 0x10, 0x10 },   // -
{ 0x00, 0x60, 0x60, 0x00, 0x00 },   // .
{ 0x20, 0x10, 0x08, 0x04, 0x02 },   // /
{ 0x3E, 0x51, 0x49, 0x45, 0x3E },   // 0
{ 0x00, 0x42, 0x7F, 0x40, 0x00 },   // 1
{ 0x42, 0x61, 0x51, 0x49, 0x46 },   // 2
{ 0x21, 0x41, 0x45, 0x4B, 0x31 },   // 3
{ 0x18, 0x14, 0x12, 0x7F, 0x10 },   // 4
{ 0x27, 0x45, 0x45, 0x45, 0x39 },   // 5
{ 0x3C, 0x4A, 0x49, 0x49, 0x30 },   // 6
{ 0x01, 0x71, 0x09, 0x05, 0x03 },   // 7
{ 0x36, 0x49, 0x49, 0x49, 0x36 },   // 8
{ 0x06, 0x49, 0x49, 0x29, 0x1E },   // 9
{ 0x00, 0x36, 0x36, 0x00, 0x00 },   // :
{ 0x00, 0x56, 0x36, 0x00, 0x00 },   // ;
{ 0x08, 0x14, 0x22, 0x41, 0x00 },   // <
{ 0x14, 0x14, 0x14, 0x14, 0x14 },   // =
{ 0x00, 0x41, 0x22, 0x14, 0x08 },   // >
{ 0x02, 0x01, 0x51, 0x09, 0x06 },   // ?
{ 0x32, 0x49, 0x59, 0x51, 0x3E },   // @
{ 0x7E, 0x11, 0x11, 0x11, 0x7E },   // A
{ 0x7F, 0x49, 0x49, 0x49, 0x36 },   // B
{ 0x3E, 0x41, 0x41, 0x41, 0x22 },   // C
{ 0x7F, 0x41, 0x41, 0x22, 0x1C },   // D
{ 0x7F, 0x49, 0x49, 0x49, 0x41 },   // E
{ 0x7F, 0x09, 0x09, 0x09, 0x01 },   // F
{ 0x3E, 0x41, 0x49, 0x49, 0x7A },   // G
{ 0x7F, 0x08, 0x08, 0x08, 0x7F },   // H
{ 0x00, 0x41, 0x7F, 0x41, 0x00 },   // I
{ 0x20, 0x40, 0x41, 0x3F, 0x01 },   // J
{ 0x7F, 0x08, 0x14, 0x22, 0x41 },   // K
{ 0x7F, 0x40, 0x40, 0x40, 0x40 },   // L
{ 0x7F, 0x02, 0x0C, 0x02, 0x7F },   // M
{ 0x7F, 0x04, 0x08, 0x10, 0x7F },   // N
{ 0x3E, 0x41, 0x41, 0x41, 0x3E },   // O
{ 0x7F, 0x09, 0x09, 0x09, 0x06 },   // P
{ 0x3E, 0x41, 0x51, 0x21, 0x5E },   // Q
{ 0x7F, 0x09, 0x19, 0x29, 0x46 },   // R
{ 0x46, 0x49, 0x49, 0x49, 0x31 },   // S
{ 0x01, 0x01, 0x7F, 0x01, 0x01 },   // T
```

```
  { 0x3F, 0x40, 0x40, 0x40, 0x3F },   // U
  { 0x1F, 0x20, 0x40, 0x20, 0x1F },   // V
  { 0x3F, 0x40, 0x38, 0x40, 0x3F },   // W
  { 0x63, 0x14, 0x08, 0x14, 0x63 },   // X
  { 0x07, 0x08, 0x70, 0x08, 0x07 },   // Y
  { 0x61, 0x51, 0x49, 0x45, 0x43 },   // Z
  { 0x00, 0x7F, 0x41, 0x41, 0x00 },   // [
  { 0x55, 0x2A, 0x55, 0x2A, 0x55 },   // 55
  { 0x00, 0x41, 0x41, 0x7F, 0x00 },   // ]
  { 0x04, 0x02, 0x01, 0x02, 0x04 },   // ^
  { 0x40, 0x40, 0x40, 0x40, 0x40 },   // _
  { 0x00, 0x01, 0x02, 0x04, 0x00 },   // '
  { 0x20, 0x54, 0x54, 0x54, 0x78 },   // a
  { 0x7F, 0x48, 0x44, 0x44, 0x38 },   // b
  { 0x38, 0x44, 0x44, 0x44, 0x20 },   // c
  { 0x38, 0x44, 0x44, 0x48, 0x7F },   // d
  { 0x38, 0x54, 0x54, 0x54, 0x18 },   // e
  { 0x08, 0x7E, 0x09, 0x01, 0x02 },   // f
  { 0x0C, 0x52, 0x52, 0x52, 0x3E },   // g
  { 0x7F, 0x08, 0x04, 0x04, 0x78 },   // h
  { 0x00, 0x44, 0x7D, 0x40, 0x00 },   // i
  { 0x20, 0x40, 0x44, 0x3D, 0x00 },   // j
  { 0x7F, 0x10, 0x28, 0x44, 0x00 },   // k
  { 0x00, 0x41, 0x7F, 0x40, 0x00 },   // l
  { 0x7C, 0x04, 0x18, 0x04, 0x78 },   // m
  { 0x7C, 0x08, 0x04, 0x04, 0x78 },   // n
  { 0x38, 0x44, 0x44, 0x44, 0x38 },   // o
  { 0x7C, 0x14, 0x14, 0x14, 0x08 },   // p
  { 0x08, 0x14, 0x14, 0x18, 0x7C },   // q
  { 0x7C, 0x08, 0x04, 0x04, 0x08 },   // r
  { 0x48, 0x54, 0x54, 0x54, 0x20 },   // s
  { 0x04, 0x3F, 0x44, 0x40, 0x20 },   // t
  { 0x3C, 0x40, 0x40, 0x20, 0x7C },   // u
  { 0x1C, 0x20, 0x40, 0x20, 0x1C },   // v
  { 0x3C, 0x40, 0x30, 0x40, 0x3C },   // w
  { 0x44, 0x28, 0x10, 0x28, 0x44 },   // x
  { 0x0C, 0x50, 0x50, 0x50, 0x3C },   // y
  { 0x44, 0x64, 0x54, 0x4C, 0x44 },   // z
  { 0x00, 0x08, 0x3e, 0x41, 0x00 },   // {
  { 0x00, 0x00, 0x7f, 0x00, 0x00 },   //  |
  { 0x00, 0x41, 0x36, 0x08, 0x00 },   // }
  { 0x10, 0x08, 0x08, 0x10, 0x08 },   //  ~

};


void Write(unsigned int write)              ///glcd transferring the character strings bit by bit
{
  uint8_t i;
  for(i=0; i<8; i++)                  ///the characters are transmitted bit by bit and each byte is send one at a
time
  {
    CLK = 0;                        ///the data  bit is send when clock is low
```

```
      data_in = ((write & 0x80) ? 1 : 0); ///checking if bit to write is 1 or 0
      CLK = 1;                          ///after data pin send, set clock high
      write <<= 1;                      ///for sending second bit
   }
}



void Data(unsigned char x)              ///data send function where data is send on glcd when glcd is in data
send mode
{
   Data_command = 1;                  ///data send mode
   chip_enable = 0;                  ///when there is a transition of chip enable from low to high data is send to
the pin
   Write(x);
   chip_enable = 1;
}

void Command(unsigned char x)         ///command send function where data is send on glcd when glcd is
in command send mode
{
   Data_command = 0;                  ///command send mode
   chip_enable = 0;                  ///when there is a transition of chip enable from low to high, command is
send to the pin
   Write(x);
   chip_enable = 1;
}
/*************************************************************
  Brings ram pointer to X,Y pixel position
   Arguments for the input: x-> X cordinate range from 0 to 83
  Arguments  for the input: y-> Y cordinate range from 0 to 5  */
/*************************************************************/
void setPixel(unsigned char x, unsigned char y)
{
   Command(0x40|(y&0x07));   /// Y axis
   Command(0x80|(x&0x7f));    /// X axis
}

/// Clears the screen
void allClear(void)
{
   int pixel;
   setPixel(0,0);                                                                    ///bring the
Cursor Home.
   for (pixel=504; pixel>0; pixel--)              ///clear all 504 pixels on the display 84*6 i.e x*y 504
DDRAM addresses.
   {
      Data(0x00);
   }
}

///GLCD NOKIA 5110 initialisation
void Initialize_LCD(void)
```

```
{
   RST = 1;              /// Set Reset pin HIGH.
   chip_enable = 1;    /// Chip Disabled

   Command(0x21);              /// Activate Chip and H=1.
   Command(0xB2);              /// Set value for correct lcd contrast
   Command(0x13);              /// Adjust voltage bias.
   Command(0x20);              /// Use horizontal addressing
   Command(0x09);              /// Activate all segments.
   allClear();                 /// Clear all pixels
   Command(0x08);              /// Blank the Display.
   Command(0x0C);              /// Normally display on lcd
   setPixel(0,0);           /// Bring the cursor home
}


void setCursor(unsigned char row, unsigned char col)       ///the glcd is x*y=14*6 and if row and column is
set more than that return
{
   if((row>6) || (row<1) || (col<1) || (col>14))        ///maximum of 6 lines of text is possible on the lcd
where each address increments
                                        ///  automatically for string to be displayed
      return;
   setPixel(((col-1)*6),(row-1));
}

/// Writes single character on LCD */

void charDisp(unsigned char a)            ///input argument is the character to be displayed on the glcd
{
   unsigned char i, b;
   if ( (a < 32) || (a > 126) )        ///for non displayable characters print character as backslash
   {
      a = 92;
   }
   for(i=0; i<5; i++)                  ///print the characters by sending to data write function from the binary
values
                                ///of characters in the lookuptable
   {
      b = LookUpTable[a - 32][i] << 1;
      Data(b);
   }
   Data(0x00);
}

/********************************** Writes character string on
LCD***************************/

void stringDisp(unsigned char *p)          ///arguments are pointer to string to be displayed
{
   while(*p)
      charDisp(*p++);
}
```

KEYPAD.C

```
#include "Keypad.h"
#include "GLCD.h"          ///libraries required for keypad
#include "Sevensegment.h"

#include <stdint.h>
#define  C2  P1_0          /// Connecting keypad to Port 1
#define  R1  P1_1
#define  C1  P3_5
#define  R4  P3_4
#define  C3  P1_4
#define  R3  P1_5
#define  R2  P1_6


extern volatile uint8_t keypad;

void row_finder1()          ///Function used for finding the row for column 1
{
   R1=R2=R3=R4=1;
   C1=C2=C3=0;

   if(R1==0)              ///if c1 are zero and r1 is zero then 1 is pressed
   {
      if(R1==0);
      keypad=1;
      P0=seven_seg[1];
      charDisp('1');
   }
   if(R2==0)            ///if c1 are zero and r2 is zero then 4 is pressed
   {
      if(R2==0);
      P0=seven_seg[4];
      charDisp('4');
   }

   if(R3==0)         ///if c1 are zero and r3 is zero then 7 is pressed
   {
      if(R3==0);
      P0=seven_seg[7];
      charDisp('7');
   }

   if(R4==0)        ///if c1 are zero and r4 is zero then * is pressed
   {
      if(R4==0);
      P0=seven_seg[0];
   }

}

void row_finder2() ///Function used for finding the row for column 2
```

```
{
  R1=R2=R3=R4=1;
  C1=C2=C3=0;

  if(R1==0)     ///if c2 is zero and r1 is zero then 2 is pressed
  {
    if(R1==0);
    keypad=2;
    P0=seven_seg[2];
  }
  if(R2==0)    ///if c2 is zero and r2 is zero then 5 is pressed
  {
    if(R2==0);
    P0=seven_seg[5];
  }
  if(R3==0)   ///if c2 is zero and r2 is zero then 8 is pressed
  {
    if(R3==0);
    P0=seven_seg[8];
  }

  if(R4==0)   ///if c2 is zero and r2 is zero then 0 is pressed
  {
    if(R4==0);
    P0=seven_seg[0];
  }
}

void row_finder3() ///Function used for finding the row for column 3
{
  R1=R2=R3=R4=1;
  C1=C2=C3=0;

  if(R1==0)  ///if c3 is zero and r1 is zero then 3 is pressed
  {
    if(R1==0);
    keypad=3;
    P0=seven_seg[3];
  }

  if(R2==0) ///if c3 is zero and r2 is zero then 6 is pressed
  {
    if(R2==0);
    P0=seven_seg[6];
  }

  if(R3==0) ///if c3 is zero and r3 is zero then 9 is pressed
  {
    if(R3==0);
    P0=seven_seg[9];
  }

  if(R4==0) ///if c3 is zero and r4 is zero then # is pressed
```

```
   {
      if(R4==0);
   }

}
```

LCD.C

```
#include "LCD.h"

#include <stdint.h>


/**************************************************************************************
*HARSH RATHORE
*UNIVERSITY OF COLORADO BOULDER
*FALL 2019
*EMBEDDED SYSTEMS DESIGN
**************************************************************************************/
///***@file name LCD.c
///*@LCD commands usage
///**this header file contains all the timer calculations usage
///**@author Harsh Rathore
///*@ date 21st nov 2019
///*@version1.0



volatile char ch=0;

volatile int8_t row2=0;                    ///rows and column value defined as global so that value of the teh
string gets printed even when timer is running
/// hence defined its values inside the timer function
volatile int8_t column2=0;
volatile char *string=0;                  ///input string which will be taken from the user is defined as global
and defined inside timer function so that
///it gets printed

volatile int8_t address1=0;                        ///defining the address for input from the user to write a
particular character

volatile char input=0;                     ///input of ui made to make the user take charge of lcd display
volatile uint8_t count=0;
volatile uint8_t row=0;
volatile uint8_t column=0;

void delay(uint32_t n)
{
   int i,j;                           ///delay function called between different lcd instruction commands

   for(i=0; i<n; i++)
   {
      for(j=0; j<1275; j++)
      {
```

```
        ;
      }
   }
}
void lcd_instruction_write(uint16_t write)
{
   instruction_reg_write=write;
   lcdbusywait();
}
void lcdbusywait()
{
   while(busy_flag_read&0x80);                    ///checking for the busy flag, if it is 1
}

void lcdinit(void)
{
   delay(15);                              ///15ms delay
   instruction_reg_write = unlock;

   delay(4);                              ///4.1ms delay
   instruction_reg_write = unlock;

   delay(1);                             ///100uS delay
   instruction_reg_write = unlock;

   lcdbusywait();                          /// wait for busy flag to clear

   instruction_reg_write = Function_set;

   lcdbusywait();                          /// wait for busy flag to clear

   instruction_reg_write = display_off;

   lcdbusywait();                           /// wait for busy flag to clear

   instruction_reg_write = display_on;

   lcdbusywait();                          /// wait for busy flag to clear

   instruction_reg_write = entrymode_set;

   lcdbusywait();                          /// wait for busy flag to clear

   instruction_reg_write = clearscreen;
}

void lcd_clear_screen()
{
   instruction_reg_write = clearscreen;        ///function to clear the lcd screen
}
void lcdgotoaddr(uint8_t addr)
{
   instruction_reg_write = addr;             ///function to write to the lcd ddram address
```

```
}
void lcdgotoxy(uint8_t rows, uint8_t columns)
{
   row=rows;
   column=columns;
   int address=0;                    ///function to specify the required location to go to specified by the row
and column
   int a=0x80;
   int b=0xC0;
   int c=0x90;
   int d=0xD0;

   if (rows==0)
      address=a+columns;             ///increment the value of the address of each row and column address
as specified as row and column values
   if (rows==1)                      ///for the address
      address=b+columns;
   if (rows==2)
      address=c+columns;
   if (rows==3)
      address=d+columns;
   lcdgotoaddr(address);             ///using lcd go to address to go to a required location
}

void lcdputch(char cc)
{
   data_register_write=cc;           ///putting character on the lcd display at the desired location
   lcdbusywait();
   delay(10);

}
void lcdputstr(char *string)
{
   count=column;
   while(*string!='\0' && (*string!='\n' && *string!='\r'))          ///till string ends keep putting characters
on lcd
   {
      lcdputch(*string++);           ///send characters one by one

      count++;
      if (count==16)                 ///check the count of the characters
      {
         row=row+1;
         count=0;
         if(row==4)                  ///if column value is 16 goto the next line and to the next row
         {
            lcdgotoxy(0,0);
            row=0;
         }

         else
         {
```

```
            lcdgotoxy(row,0);    ///if reached the last column and last row, jump to the first location
          }
        }
    }
}
```

REMOTE_FUNCTIONS.C

```
#include "Remote_functions.h"
#include "remote_key.h"
#include "Dc_motor_pwm.h"
#include "GLCD.h"
#include <stdio.h>
#include <stdlib.h>
#include<at89c51ed2.h>          ///include library functions needed for remote functions
#include <mcs51/8051.h>
#include <stdint.h>
volatile uint8_t timer_mili=0;  ///used for storing the count value of timer interrupt for pulse width
volatile uint8_t mili_count=0;    ///used for calculating 1ms timer interrupt
volatile int pulseCount=0;    ///calculating number of pulses
extern volatile uint32_t bitPattern;  ///forming the binary bit pattern
extern volatile uint32_t newKey;     ///storing value of bit pattern into variable
extern volatile uint8_t remote_flag;
volatile int8_t power_key=0;         ///used for reprogrammable remote functionality
volatile int8_t back_key=0;
volatile int8_t home_key=0;

#define Bulb P1_2             ///defining port at which led is placed


/****************************************Timer T0
interrupt****************************************
This function is used for detecting the pulse width by incrementing the count of a variable that passes
through the interrupt every 1 ms and
hence we find the pulse width by measuring this variable count with the time the external edge of arrives,
which gives us the pulse width
*********************************************************************************************
****************/
void timer0_isr() __interrupt 1
{
   if(mili_count<50)
      mili_count++;

   TH0 = 0xFC;   // Reload the timer value for 1ms Delay
   TL0 = 0x67;
}

/****************************************External INT0
interrupt****************************************
This function is used for detecting the number of pulses through the external hardware pin
The interrupt triggers on every falling edge of the pulse
*********************************************************************************************
****************/
```

```
void externalIntr0_ISR()  __interrupt 0
{

   timer_mili = mili_count;
   mili_count = 0;

   TH0 = 0xFC;  /// Reload the timer value for 1ms Delay
   TL0 = 0x67;

   pulseCount++;

   if((timer_mili>=50)) /// If the pulse width is greater than 50ms, the new pulse start of frame is present
   {
      pulseCount = -2; ///the value of first 2 counts is not included as part of the 32 bit address and
command bits,therefore pulse count is -2
      bitPattern = 0;  ///and therefore bit pattern is 0
   }
   else if((pulseCount>0) && (pulseCount<=32)) ///Accumulate  0-31 bit values
   {
      if(timer_mili>=2)                  ///pulse width greater than 2ms is considered as LOGIC1
      {
         bitPattern |=(uint32_t)1<<(31-pulseCount);
      }
      else
      {

      }
   }
   else if(pulseCount>=32)               ///32 pulses are received, hence we store it into the bit pattern
   {
      newKey = (bitPattern>>4);              /// Copy the newKey(patter) and set the pulse count to 0;

      pulseCount = 0;
      remote_flag=1;

      printf("newKey=%lu\n\r",newKey);       ///display the pattern on uart
      if (newKey!=0)
      {

         if (newKey==power || newKey==power1 ||(newKey==power && newKey==32763231 &&
newKey==66580479))
         {

            if(power_key==1)                    ///if power key is programmed as motor, motor will turn on
            {
               motor_direction_pin1=0;
               motor_direction_pin1=1;
               printf_tiny("\n\rmotor\n\r");
            }
            else if(power_key==2)               ///if power key is programmed as light, light will turn on
            {
               Bulb=Bulb^1;
```

```
            printf_tiny("\n\rlight\n\r");
            CR=0;
          }
          else if (power_key==3)          ///if power key is programmed as to display a message on
glcd, glcd will turn on
          {
            allClear();
            setCursor(1,0);
            stringDisp(" POWER BUTTON");

          }
        }
        if(newKey==back || newKey==back1)
        {

          if(back_key==1)          ///if back key is programmed as motor, motor will turn on
          {
            motor_direction_pin1=0;
            motor_direction_pin1=1;
            printf_tiny("\n\rmotor\n\r");

          }
          else if(back_key==2)     ///if back key is programmed as light, light will turn on
          {
            Bulb=Bulb^1;

            CR = 0;

          }
          else if (back_key==3)    ///if back key is programmed as to display a message on glcd, glcd will
turn on
          {
            allClear();
            setCursor(1,0);
            stringDisp(" BACK KEY");
            CR = 0;
          }
        }
        if(newKey==home || newKey==home1)   ///if home key is programmed as motor, motor will turn
on
        {
          if(home_key==1)
          {
            motor_direction_pin1=0;
            motor_direction_pin1=1;
            printf_tiny("\n\rmotor\n\r");

          }
          else if(home_key==2)     ///if home key is programmed as light, light will turn on
          {
            Bulb=Bulb^1;
            CR = 0;
```

```
            }
            else if (home_key==3)      ///if home key is programmed as to display a message on glcd, glcd
will turn on
            {
               setCursor(1,0);
               stringDisp("  HOME BUTTON");

            }

         }
         if(newKey==up_arrow || newKey==up_arrow1) ///set to increase speed of dc motor
         {
            allClear();
            setCursor(0,0);
            stringDisp("INCREASE SPEED ");
         }
         if(newKey==right_arrow || newKey==right_arrow1)  ///set to stop dc motor
         {
            printf_tiny("\n\rright\n\r");
            motor_direction_pin1=1;
            motor_direction_pin1=0;

            Bulb=Bulb^1;

         }
         if(newKey==down_arrow || newKey==down_arrow1)   ///set to decrease speed of dc motor
         {
            allClear();
            setCursor(0,0);
            stringDisp("REDUCE SPEED");


         }
         if(newKey==left_arrow || newKey==left_arrow1)  ///set to start dc motor
         {
            printf_tiny("\n\rleft\n\r");
            motor_direction_pin1=0;
            motor_direction_pin1=1;

         }
         if(newKey==ok_key || newKey==ok_key1)         ///set to turn light on
         {
            Bulb=Bulb^1;
         }

         if(newKey==night_mode_key || newKey==night_mode_key1) ///set to turn light on
         {

            Bulb=Bulb^1;

         }
         if(newKey==brightness_key || newKey==brightness_key1) ///set to turn light on
         {
```

```
            Bulb=Bulb^1;

        }
        if(newKey==rewind_key || newKey==rewind_key1)  ///set to switch direction of dc motor
        {

            Bulb=Bulb^1;
        }
        if(newKey==fast_forward_key || newKey==fast_forward_key1) ///set to turn light on
        {
            Bulb=Bulb^1;

        }
        if(newKey==advance_key || newKey==advance_key1) ///set to turn light on
        {

            Bulb=Bulb^1;
        }
      }

   }
}


/*****************************IR REMOTE INIT************************************
THIS FINCTION CONTAINS ALL INIT REQUIREMENTS FOR DETECTION OF NEC PROTOCOL
TIMER 0 IS USED FOR GENERATING 1 ms interrupt
INT0 is used for triggering an interrupt every falling edge of the pulse
*******************************************************************************/
void IR_RemoteInit(void)
{
   TMOD |= 0x01;  /// Timer0 MODE1(16-bit timer)
   TH0 = 0xFC;    /// Timer value for 1ms at 11.0592Mhz clock
   TL0 = 0x67;
   TR0 = 1;       /// Start the Timer
   ET0 = 1;       /// Enable the Timer0 Interrupt

   IT0 = 1;       /// Configure external INT0 falling edge interrupt
   EX0 = 1;       /// Enable the INT0 External Interrupt

   EA  = 1;       /// Enable the Global Interrupt bit
}
```

DC_MOTOR.H

```
#ifndef _DC_MOTOR_PWM_H_
#define _DC_MOTOR_PWM_H_
/*******************************************************************************
*HARSH RATHORE
```

```
*UNIVERSITY OF COLORADO BOULDER
*FALL 2019
*EMBEDDED SYSTEMS DESIGN
**********************************************************************************/
/// ***@file name DC_motor_pwm.h
/// *@pwm commands and variables
/// **this header file contains all the dc motor functions,its prototypes and variables
/// **@author Harsh Rathore
///@reference-https://www.electronicwings.com/8051/dc-motor-interfacing-with-8051
/// *@ date 1st Dec 2019
/// *@version1.0
/// */

#include <stdint.h>
extern volatile uint8_t motor_direction_pin1;
extern volatile uint8_t motor_direction_pin2;

/**
*@brief sets the initial values of the timer2 low and high register and starts the timer
**Given none

*@param none
**@return void.
*/

void Timer_init();

/**
*@brief sets the initial values of the timer2 low and high register and starts the timer
**Given none

*@param none
**@return void.
*/
void Motor_Init();                          ///prototype declaration of dc motor functions
/**
*@brief sets the values of the duty cycle for speed control and pins for direction control
**Given a float duty cycle


*@param  duty cycle value to be set for pwm dc motor
**@return void.
*/
void Set_DutyCycle_To(float duty_cycle);
/**
*@brief the interrupt handler function which triggers depending on the timer load value
**Given none

*@param none
**@return void.
*/
void Timer0_ISR() __interrupt 5;
```

```
#endif
```

GLCD.H

```
#ifndef _GLCD_H_
#define _GLCD_H_

/**************************************************************************************
*HARSH RATHORE
*UNIVERSITY OF COLORADO BOULDER
*FALL 2019
*EMBEDDED SYSTEMS DESIGN
**************************************************************************************/
/// ***@file name GLCD.h
/// *@Graphical lcd
/// **this header file contains all the intructions,variables and commands of nokia 5110 graphical lcd
/// **@author Harsh Rathore
/// @reference-http://www.circuitstoday.com/nokia-graphic-lcd-display-8051
/// *@ date 1st Dec 2019
/// *@version1.0
/// */

#include<at89c51ed2.h>          ///include library functions needed for lcd display and 8051
#include <mcs51/8051.h>
#include <stdint.h>
#define CLK P2_1
#define data_in P2_2
#define Data_command P2_3
#define chip_enable P2_4
#define RST P2_5


extern const unsigned char LookUpTable [][5];          ///lookup table to print characters

/**
*@brief the write to pixel through spi protocol
**Given an unsigned int write to write the pixel value

**@param write the pixel value

**@return void.
*/

void Write(unsigned int write);
/**
*@brief the data value to be written
**Given an unsigned char value which is send to the ddram address

**@param char x

**@return void.
*/
```

```
void Data(unsigned char x);
/**
*@brief the command instruction to be sent
**Given a n unsigned char

**@param x set the command in hex, mostly used at start to initialise the lcd

**@return void.
*/

void Command(unsigned char x);
/**
*@brief the function to set the pixel address
**Given a n unsigned char x
*Given a n unsigned char y
**@param x to set x axis
*@param y to set y axis
**@return void.
*/

void setPixel(unsigned char x, unsigned char y);        ///prototype declaration of glcd functions
/**
*@brief the lcd clear function
**Given none, clears the entire lcd


*@param void
**@return void.
*/

void allClear(void);
/**
*@brief the function is used to initialize the lcd
**Given a void value

*@param value
**@return void.
*/

void Initialize_LCD(void);

/**
*@brief the set the pixel position
**Given a char row which is the x axis
*given a char value col which is y axis
**@paramp char row value
*@param  char col value
**@return void.
*/

void setCursor(unsigned char row, unsigned char col);
```

```
/**
*@ brief the function to print character on glcd
**Given a unsigned char to variable
**@param value of data item to be displayed
**@return void.
*/

void charDisp(unsigned char a);

/**
*@brief the function to print strings on glcd
**Given a char pointer to a variable p
**@param pointer to a data iterm
**@return void.
*/
void stringDisp(unsigned char *p);

#endif
```

KEYPAD.H

```
#ifndef _KEYPAD_H_
#define _KEYPAD_H_

/*****************************************************************************
*HARSH RATHORE
*UNIVERSITY OF COLORADO BOULDER
*FALL 2019
*EMBEDDED SYSTEMS DESIGN
*****************************************************************************/
/// ***@file name Keypad.h
/// *@KEYPAD 4*3
/// **this header file contains all the intructions,variables and commands of 4*3 keypad
/// **@author Harsh Rathore
/// @reference-https://circuitdigest.com/microcontroller-projects/keypad-interfacing-with-8051-
microcontroller
/// *@ date 1st Dec 2019
/// *@version1.0
/// */


/**
*@brief the row finder function for a given column
**Given none

**@param none

**@return void.
*/
void row_finder1();          ///prototype declaration of keypad functions
/**
*@brief the row finder function for a given column
```

```
**Given none

**@paramp none

**@return void.
*/
void row_finder2();
/**
*@brief the row finder function for a given column
**Given none

**@paramp noneem

**@return void.
*/
void row_finder3();

extern unsigned char seven_seg[];    ///hex values for displaying 0 to 9
extern unsigned char seven_seg_alp[];

#endif
```

LCD.H

```
#ifndef _LCD_H_
#define _LCD_H_

/*********************************************************************************
*HARSH RATHORE
*UNIVERSITY OF COLORADO BOULDER
*FALL 2019
*EMBEDDED SYSTEMS DESIGN
*********************************************************************************/
/// ***@file name LCD.h
/// *@LCD commands
/// **this header file contains all the lcd command and display instructions
/// **@author Harsh Rathore
/// *@ date 21st nov 2019
/// *@version1.0



#include <stdlib.h>
#include<at89c51ed2.h>          ///include library functions needed for lcd display and 8051
#include <mcs51/8051.h>
#include <stdint.h>
#define unlock (0x30)
#define Function_set (0x38)      ///macro defined for different sets of lcd commands
#define display_off (0x08)
#define display_on (0x0E)
#define entrymode_set (0x06)
#define clearscreen (0x01)

__xdata __at (0xF000) volatile unsigned char instruction_reg_write;  ///Instruction Register Write address
```

```
__xdata __at (0xF100) volatile unsigned char data_register_write;   ///Data register write address

__xdata __at (0xF200) volatile unsigned char busy_flag_read;        ///  Busy Flag Read address

__xdata __at (0xF300) volatile unsigned char data_register_read;   ///Data register read address

/**
*@brief the lcd put string function to print the string
**Given a pointer string

**@paramptr string value to be displayed

**@return void.
*/
void lcdputstr(char *string);
/**
*@brief the character to be displayed
**Given a char variable


*@param value to write to the lcd
**@return void.
*/
void lcdputch(char cc);
/**
*@brief the rows and columns the cursor needs to be set to
**Given a uint8_t value of row
***Given a uint8_t value of col
**@param sets row
*@param sets column
**@return void.
*/
void lcdgotoxy(uint8_t rows, uint8_t columns);
/**
*@brief the address of ddram
**Given a uint8_t value of address

*@param value of the address the character should be displayed at
**@return void.
*/
void lcdgotoaddr(uint8_t addr);                         ///prototype declaration of lcd functions
/**
*@brief to clear the screen
**Given none

**@param none

**@return void.
*/

void lcd_clear_screen();
/**
```

```
*@brief the lcd init function to initialize the lcd correctly
**Given a void

*@param void
**@return void.
*/

void lcdinit(void);

/**
*@brief the lcd busy wait function to wait for command to process

**Given none

*@param void
**@return void.
*/

void lcdbusywait();

/**
*@brief the delay of 1 ms
**Given a uint32_t value

**@param provide delay in multiple of 1ms

**@return void.
*/

void delay(uint32_t n);

/**
*@brief the instruction commands for lcd
**Given a uint16_t write

**@param write instruction commands

**@return void.
*/

void lcd_instruction_write(uint16_t write);

#endif
```

REMOTE_FUNCTIONS.H

```
#ifndef _REMOTE_FUNCTIONS_H_
#define _REMOTE_FUNCTIONS_H_

/*****************************************************************************
*HARSH RATHORE
*UNIVERSITY OF COLORADO BOULDER
*FALL 2019
```

```
*EMBEDDED SYSTEMS DESIGN
*********************************************************************************/
/// ***@file name Remote_functions.h
/// *@Ir remmote control
/// **this header file contains all the intructions,variables and commands of nec based ir remote
/// **@author Harsh Rathore
/// @references-https://exploreembedded.com/wiki/NEC_IR_Remote_Control_Interface_with_8051
/// https://www.snrelectronicsblog.com/8051/8051-based-remote-for-home-appliance/
/// *@ date 1st Dec 2019
/// *@version1.0
/// */


/**
*@brief the external interrupt handler function which triggers depending on the external hardware
interrupt
.
**@paramptr none

**@return void.
*/
void externalIntr0_ISR()  __interrupt 0;
/**
*@brief the remote init function initializing remote timer interrupt values

**@paramptr void

**@return void.
*/
void IR_RemoteInit(void);                    ///prototype declaration of remote functions
/**
*@brief the time r0 interrupt for ms count which runs according to timer value loaded

**@param none

**@return void.
*/

void timer0_isr() __interrupt 1;

#endif
```

REMOTE_KEY.H

```
#ifndef _REMOTE_KEY_H
#define _REMOTE_KEY_H
/*********************************************************************************
*HARSH RATHORE
*UNIVERSITY OF COLORADO BOULDER
*FALL 2019
*EMBEDDED SYSTEMS DESIGN
*********************************************************************************/
```

```
/// ***@file name Remote_key.h
/// *@Remote_key
/// **this header file contains all the intructions,variables and commands of Remote_key
/// **@author Harsh Rathore
/// @reference-https://exploreembedded.com/wiki/NEC_IR_Remote_Control_Interface_with_8051
/// *@ date 1st Dec 2019
/// *@version1.0
/// */

#define  power 0x1E3E817
#define power1 0x3E3E817
#define power_long_press
#define back  0x1E36699
#define back1  0x3E36699
#define home  0x3E3C03F
#define home1  0x1E3C03F
#define up_arrow  0x3E39867
#define up_arrow1  0x1E39867
#define right_arrow  0x3E3B44B
#define right_arrow1  0x1E3B44B
#define down_arrow   0x3E3CC33
#define down_arrow1   0x1E3CC33              ///values of bitpatterns for the different keys pressed
#define  left_arrow 0x3E37887
#define  left_arrow1 0x1E37887
#define  ok_key  0x3E354AB
#define  ok_key1  0x1E354AB
#define  replay  0x3E31EE1
#define  replay1  0x1E31EE1
#define   night_mode_key  0x3E346B9
#define   night_mode_key1  0x1E346B9
#define  brightness_key   0x3E38679
#define  brightness_key1   0x1E38679
#define  rewind_key  0x3E32CD3
#define  rewind_key1  0x1E32CD3
#define  fast_forward_key   0x3E332CD
#define  fast_forward_key1   0x3E332CD
#define  advance_key  0x3E3AA55
#define  advance_key1  0x1E3AA55
#define  netflix_key 0x3E34AB5
#define  netflix_key1 0x1E34AB5
#define  espn_key 0x3E34EB1
#define  espn_key1 0x1E34EB1
#define  hulu_key 0x3E3B24D
#define  hulu_key1 0x1E3B24D
#define  roku_tv 0x3E37689
#define  roku_tv1 0x1E37689
#define pow 0xE817

#endif
```

SEVENSEGMENT.H

```
#ifndef _SEVEN_SEGMENT_H_
#define _SEVEN_SEGMENT_H_

/*****************************************************************************
*HARSH RATHORE
*UNIVERSITY OF COLORADO BOULDER
*FALL 2019
*EMBEDDED SYSTEMS DESIGN
*****************************************************************************/
/// ***@file name Seven_Segment.h
/// *@Seven_Segment
/// **this header file contains all the array for numbers from 0 to 9 and alphabets from A to C
/// **@author Harsh Rathore
/// *@ date 1st Dec 2019
/// *@version1.0


extern unsigned char seven_seg[];    ///hex values for displaying 0 to 9
extern unsigned char seven_seg_alp[];                    ///hex values to display a to c
#endif
```

8.4 Appendix - Data Sheets and Application Notes

Attached as pdf