REFRIGERATOR MONITORING SYSTEM

April 23, 2023

1 REFRIGERATOR MONITORING SYSTEM

Submitted To:
Prof. D.V.Gadre

Submitted by:
Akshay Rana(2020UEC2547)
Kushal Kumar(2021UEC2546)
Rakesh Kumar(2021UEC2567)

Electronics And Design Workshop(EDW)Project
Netaji Subash University Of Technology
ECE SEC-1

2 ACKNOWLEDGEMENT

We would like to thank a few people without whom this project would certainly not have been possible. Gadre sir for his constant support and guidance.Thank you for sparking our interest in building something real, we had a ton of fun while putting this project together and are grateful for the hands-on approach that you took while teaching this course. Monis bhaiya and Naman bhaiya always there to help us with anything related to our project. Thank you for patiently solving our silliest doubts

3 SYNOPSIS

The Refrigerator Monitoring Device is an IOT solution designed to help households and efficiently monitor the refrigerators. With this device, users can easily track the number of times their refrigerators are opened from anywhere.Whenever the refrigerator is kept open accidentally for certain duration the buzzer start beeping.It can also help us to minimize our electricity consumption. The refrigerator monitoring device is easy to install and operate, providing a user-friendly interface.

4 PROJECT DESCRIPTION

- NodeMCU ESP8266: The brain behind our whole project, NodeMCU ESP8266 is one type of microcontroller board, designed by Espressif Systems. It is a small size board which is also flexible with a wide variety of applications. It performs all the functions ranging from calculating the tank size to sending information/status about the tank to the web app in real time. It has the capability to connect to WiFi inbuilt. Some technical specifications of the board are as follows: Microcontroller: Tensilica 32-bit RISC CPU Xtensa LX106

  Operating voltage: 3.3 volts

  Input voltage: 7-12 volts

  Digital I/O pins: 16

  Analog input pins: 1

  Flash memory: 4 MB

  SRAM: 64 KB

  Clock speed: 80 MHz

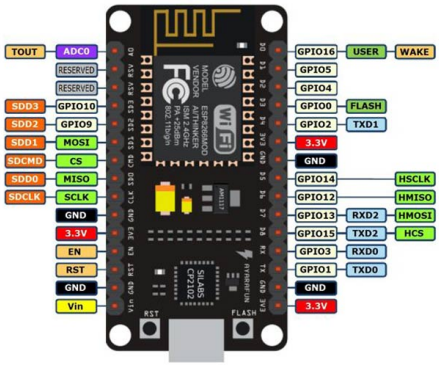


Figure 1: Pin out NodeMCU

- Opto Isolated Module(MOC7811): MOC7811 is a slotted Opto isolator module, with an IR transmitter a photodiode mounted on it. Performs Non-Contact Object Sensing. This is normally used as positional sensor switch (limit switch) or as Position Encoder sensors used to find position of the wheel. It consists of IR LED and Photodiode mounted facing each other enclosed in plastic body.

  Specifications:

  Mounting hole diameter: 3mm

  Mounting hole spacing: 19mm

  Slot width: 3mm

  Slot depth: 7mm

Figure 2: MOC7811

- : Buzzers are electric sounding devices that generate sounds. Typically powered by DC voltage, they can be categorised as Piezo buzzer and magnetic buzzer.Piezo-type buzzer's core is the piezoelectric element. The piezoelectric element is made out of piezoelectric ceramic as well as the metal plate, they are held together in or piece by the adhesive.

Figure 3: Buzzer

- LCD Display: The term LCD stands for liquid crystal display. It is one kind of electronic display module used in an extensive range of applications like various circuits devices like mobile phones, calculators, computers, TV sets, etc. These displays are mainly preferred for multi-segment light-emitting diodes and seven segments. The main benefits of using this module are inexpensive; simply programmable, animations, and there are no limitations for displaying custom characters, special and even animations, etc.

Figure 4: PIN OUT

-

I2C Connector: I2C lcd adapter is a device containing a micro-controller PCF8574 chip. This micro-controller is a I/O expander, which communicates with other micro-controller chip with two wire communication protocol. Using this adapter anyone can control an 16x2 LCD with only two wire(SDA, SCL). It saves many pins of NodeMCU/arduino or other micro-controller. It has an built in potentiometer for control lcd contrast.

Figure 5: I2C LCD Connector

- Resistors: A passive electrical component with two terminals that are used for either limiting or regulating the flow of electric current in electrical circuits. The main purpose of resistor is to reduce the current flow and to lower the voltage in any particular portion of the circuit.

BLOCK DIAGRAM:

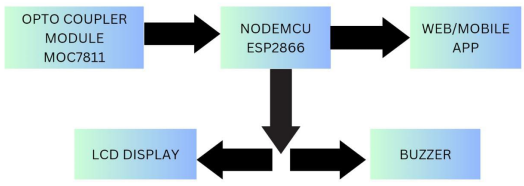


Figure 6: BLOCK DIAGRAM

Bill Of Materials:

S.No.	Component Name	Price
1	NodeMCU ESP8266	250
2	Opto Isolated Module(MOC7811)	110
3	BUZZER	50
4	LCD Display	100
5	ENCLOSING CASE	100
6	I2C Connector	400
7	Micellaneous	50

- GANTT CHART

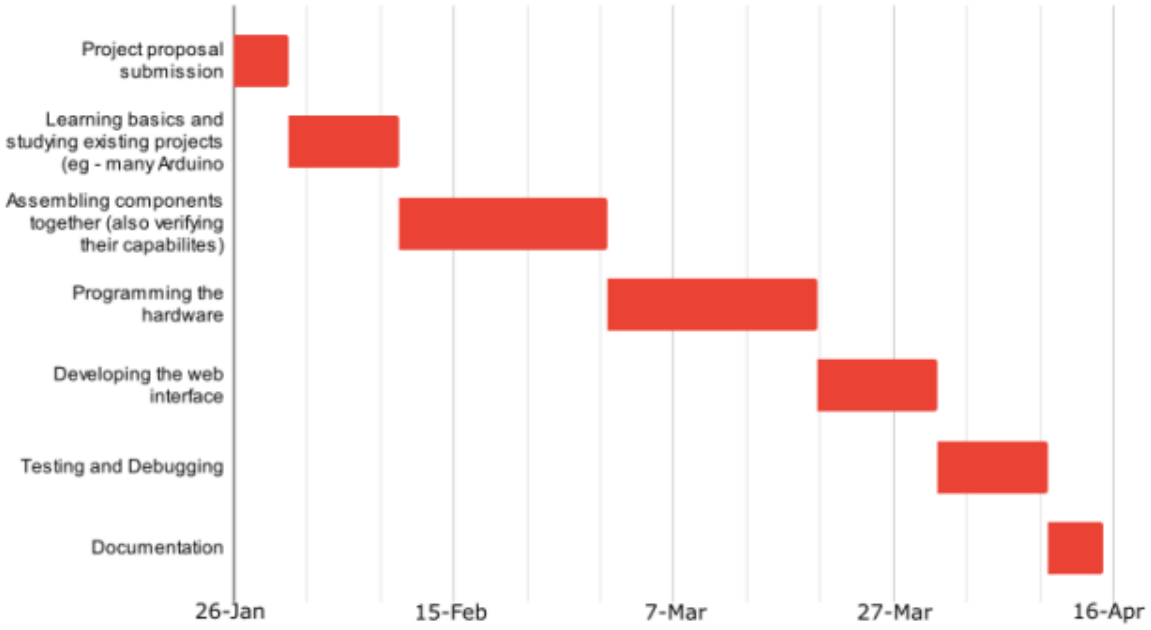


Figure 7: Gannt Chart

DIFFICULTIES FACED

- Programming the ESP8266 - We had initially started with an Arduino Nano as our main microcontroller. We had planned to use it to operate the ESP8266 module however after a bit of tinkering, we found out that the ESP8266 module cannot be operated by another microcontroller, it can only be programmed to operate a certain way such that its code is separated from the code that runs on the Arduino. This way, the Arduino essentially became an inefficient UART bridge that is used to program the ESP8266 chip. We solved this problem by switching to a NodeMCU ESP8266 microcontroller.

- Complexities on the software side - Our plan initially was to send HTTP requests through the microcontroller and use it to trigger actions on the server side using an IoT service such as IFTTT or Thingspeak. The problem with these platforms was two-fold. First, they did not store past data which made plotting previous record on a chart impossible without using another service (and increasing complexity). Secondly, they used HTTP to talk to the microcontroller which was inefficient. After some reasearch, we found a platform (Blynk) that kept logs of data and had a convenient library which abstracted the MQTT protocol for the ESP8266.

CODE EXPLANATION

```
#include <LiquidCrystal_I2C.h>          //Library for LCD Display
LiquidCrystal_I2C lcd(0x3F,16,2);
#include <ESP8266WiFi.h>                //Library for wifi and hotspot on and off
#include <ESP8266WebServer.h>           //Library for sending html
#include<EEPROM.h>                      //Library for EEPROM
#include <BlynkSimpleEsp8266.h>         //Library for Blynk.cloud server
//--------------------------------
```

1.These are the necessary libraries which would be required in our system

```
#include <DNSServer.h>                  //For Captive Portal
```

2.For captive portal we require following library.

```
IPAddress ManuallysetIP(192,168,4,1);
DNSServer SignInRequiredMsg;
```

3.Setting ip address for access point(hotspot) and user defined variable for DNSServer.

```
int AP_decider=1;
int AP_presentstate;
int AP_laststate;
int APdisable;
int AP_enablebutton=D4;
```

4. Variables required for counting switch which is used to change the mode of system from wifi to access point and vice versa. D4 pin is declare to receive signal that whether the switch is pressed or not.

```
int Apin = A0;           //A0 input signal to microcontroller- door is closed or open
int buzzer= D8;
int buttonStateAnalogSignal=0;
int Counter=0;
int seconds=0;
String DoorStatus;
String PresentState;
String LastState;
String instrucn="Stable State";
```

5.A0 pin is used to receive an analog signal from photo coupler that it is interrupted by an opaque object , sending low values as emitter junction current is low no light is detected by photodiode at base of photo transistor or not interrupted then sending high value to the micro controller as photodiode receive an infra red light that causes to produce emitter current higher.

Low signal:photo coupler is interrupted= door is closed High signal: photo coupler is not interrupted= door is open D8 pin connected to buzzer .Sends high to buzzer when door is opened since long time. Time for this can be adjusted by code. By default we have set this time for 1 minute which can be explan in further code. “String DoorStatus” stores the status of door whether it is open or closed. “Int seconds” is variable used for timer. “String instruction” stores whether the system is stable or not by default it set to “stable state”. Other variables used for requirement in code and can be explained later.

```
//--------------Variables for EEPROM-------------------
int lengthofprevssid;
int lengthofprevpass;
int lengthofprevAuth;
char ssidchar;
char passchar;
char Authchar;
String newssid="";
String newpass="";
String newAuth="";
//-----------------------------------------------------
```

6.These are the variables used for saving ssid,pass,authentication token to EEPROM when user try to change these parameters through captive portal.

```
//-----------------------Variables to store Credentials from Web Server--------
String inputssid;
String inputpass;
String inputAuth;
//-------------------------------------------------------------------------------
```

7.String used to store input parameters from web server.

```
//------------HTML Code Store in Variable--------------------------------------
const char MAIN_page[] PROGMEM = R"=====(
<!DOCTYPE html>
<html>
<body>
<center><h1>RMS7811i</h1></center>
<center><table border="1" bordercolor="blue" bgcolor="skyblue" height="400" width="400">
<td>
<center>
<h3>UPDATE CREDENTIALS</h3>
<form action="/AfterSubmit">
  New SSID:<br>
  <input type="text" name="NewSSID" placeholder="Enter your New SSID" required>
  <br>
  Password:<br>
  <input type="password" name="NewPassword" placeholder="Enter your SSID Password">
  <br>
  Authentication Token<br>
  <input type="text" name="AuthenticationToken" placeholder="Enter your BLYNK Auth" required>
  <br><br>
  <input type="submit" value="Submit">
</form><br><br>
</div>
</table>
</center>
</td>
</center>
</body>
</html>
)=====";
//-----------------------------------------------------------------------------
```

8.Constant char userdefined name[] PROGM EM= (parameters) :any thing written in the parameters of this function will considered as string, this function is used to store our html code as string. Because html has inverted commas in between the code that makes difficult to store in general way of declaring string and storing it in single variable.

```
//-----------------------------------------------------------------
ESP8266WebServer server(80); //Server on port 80
//-----------------------------------------------------------------
```

9. This is necessary to declare virtual server server port.

```
void MyServer(){

  server.onNotFound( []() {

    String s= MAIN_page;
     server.send(200, "text/html", s);
    });

server.on("/AfterSubmit", []() {

     inputssid = server.arg("NewSSID");
     inputpass = server.arg("NewPassword");
     inputAuth = server.arg("AuthenticationToken");
        //-------------writing to EEPROM-----------------------------------------------
    EEPROM.put(150,inputssid.length());
    for(int i=0 ; i<inputssid.length(); ++i){
    EEPROM.put(i,inputssid[i]);}

    EEPROM.put(155,inputpass.length());
    for(int i=0 ; i<inputpass.length(); ++i){
    EEPROM.put(i+50,inputpass[i]);}

    EEPROM.put(160,inputAuth.length());
    for(int i=0 ; i<inputAuth.length(); ++i){
    EEPROM.put(i+100,inputAuth[i]);}
    //-------------------
    EEPROM.commit(); // SAVING/FLASHING EEPROM
    //-------------------
    String content = "<!DOCTYPE HTML><html><h1>!!!UPDATED!!!<h1><br><br><h3>Restarting...<h3><br><br><h4>Please Restart Manually for better functionality<h4></html>";
    server.send(200, "text/html", content);
    delay(2000);
    ESP.reset();
     });
}
```

10.myserver() a user defined function. Calling this function when Access point is enabled. Server.onNotFount() will execute data even if user not requested . this has parameter as send html through port 200 and html is stored in “MAIN page” which is now equals to “s” this all html code send by function “server.send”.

11. When user enter all the credentials and submit it by clicking on submit button, all input from server (html page) is now fetched and stored in respective variables.Size of parameters of Ssid will store at location 150 of EEPROM ,password at 155 and auth token at 160. These sizes have information how long our parameters are and saved at this location. And these data is used when we fetch these parameters for wifi connection. For loop is used to write every character of parameters at specific location one by one. Ssid will store from 0 to its length. Password store from location 50 to its length. Auth token stores from location 100 to its length. "EEPROM.commit" will save the data which was written in the process. Even if the system is off the data will not erase. After 2 second . the ESP8266 will restart and begin wifi connection with new ssid,password,authentication token id.

```
void ReadingDataFromEEPROM(){
  //---------Reading SSID,PASS and AUTH from EEPROM to begin wifi IOT Connection---------
EEPROM.get(150,lengthofprevssid);//Storing lenth value of ssid to given variable
for(int i=0 ; i<lengthofprevssid; ++i){
  EEPROM.get(i,ssidchar);
  newssid += ssidchar;}

EEPROM.get(155,lengthofprevpass);//Storing lenth value of pass to given variable
for(int i=0 ; i<lengthofprevpass; ++i){
  EEPROM.get(i+50, passchar);
  newpass += passchar;}

EEPROM.get(160,lengthofprevAuth);//Storing lenth value of Auth to given variable
for(int i=0 ; i<lengthofprevAuth; ++i){
  EEPROM.get(i+100,Authchar);
  newAuth += Authchar;}

//-------------------------------------------------------------------------------------
}
```

12. "ReadingDataFromEEPROM" user defined function is used to read data from EEPROM. First length of ssid is reading from location 150, there we can find the size of our previous ssid that is stored. Now using loop from location 0 to its length we are storing character one by one to "newssid " variable. Similarly we can read for other stored parameters.

```
 83  void setup() {
 84  EEPROM.begin(512);
 85  ReadingDataFromEEPROM();
 86  server.begin();
 87  Serial.begin(9600);
 88  pinMode(AP_enablebutton,INPUT_PULLUP);
 89  pinMode(Apin,INPUT);
 90  pinMode(buzzer,OUTPUT);
 91
 92  lcd.backlight();
 93  lcd.begin();
 94  lcd.clear();
 95  lcd.print("--System Is ON--");  // initial display of lcd
 96  lcd.setCursor(3,1);// LCD Cursor set at- 3rd coloumn 1st row
 97  lcd.print("RMS7811i");
 98  delay(2000);
 99  lcd.clear();
100  lcd.setCursor(0,0);// LCD Cursor set at- 0th coloumn 0th row
101  lcd.print("Your System is Now Monitered with IoT   ");
102  delay(1000);
103  for (int pos=0; pos<24;pos++)
104    {
105      lcd.scrollDisplayLeft();
106      delay(500);
107      }
```

13.Inside the void setup.some Necessary initialisation, EEPROM begin with 512

kb(ESP8266 has total 1kb memory in EEPROM). Calling function “Reading-DataFromEEPROM” for reading ssid,password,authentication token which was saved in EEPROM. Initialising server lcd, by“server.begin” and “lcd.begin”. Declaring modes of various pins. “Lcd.backlight” will turn on backlight of lcd . “lcd.clear” clear the Lcd screen. “Lcd.print” is used to print the characters on lcd. Lcd has dimension of 2 rows and 16 columns “lcd.setcursor” is used to set the cursor at specific location on lcd display. Then for loop is used to scroll lcd text which is currently on displayed.

```
110  void loop() {
111  //----------------------AP Decider----------------------------------------
112  char newAuthCharrArray[newAuth.length()+1];
113  newAuth.toCharArray(newAuthCharrArray,newAuth.length()+1); //string to charr array
114  if(digitalRead(AP_enablebutton)== HIGH){
115    AP_presentstate=1;
116  }
117  if(digitalRead(AP_enablebutton)== LOW){
118    AP_presentstate=0;
119  }
120  if(AP_decider %2==0){
121     lcd.setCursor(14,1);
122     lcd.print("AP");
123     APdisable--;
124      if (APdisable==0){
125      AP_presentstate=0;
126        }
127  }
128  if(AP_presentstate!= AP_laststate){
129    if(AP_presentstate==0){
130    AP_decider++;}
131     if(AP_decider %2==0){
132       APdisable=120;
133       WiFi.disconnect();
134       WiFi.softAP("RMS78111","12345678");
135       SignInRequiredMsg.start(53, "*", ManuallysetIP);
136       MyServer();
137     }
138     if(AP_decider %2 != 0){
139      WiFi.softAPdisconnect();
140      WiFi.begin(newssid,newpass);
```

14.Here in void loop , the instructions in void loop will executed repeatedly. Line 112 and 113: this code is to change type from char to char array because “blynk.begin” take char array as parameters. Now we are setting push button for changing mode to Access Point or Wifi mode, working behind this to count push button if push button pressed for even times then it turn on hotspot and act as Access point . where user can connect the hotspot, if any device is connected to its access point ,captive portal will bring the html page automatically if not then user can type ip address of it on any browser, an web server will load where

user can set the new credentials by filling the required field on web page. As we can see in the code “AP decider” where count is stored, if it is even it disconnect the current wifi connection and on hotspot and captive portal begin given ip address in the code. “Myserver” function calling at this point to send server instruction which was defined in the function. If the the push button is pressed odd times then it turn off the Access point and start wifi connection. Setting lcd cursor at bottom corner. Clearing “AP” mode indicator from the lcd. In the wifi mode we need to establish connection with blynk server to send data on internet “Blynk config” is used with authentication token as its parameters (This authentication token is available on www. blynk .cloud when an user sign up there).

```
141       lcd.setCursor(14,1);
142       lcd.print("  ");//Clearing AP mode display
143       Blynk.config(newAuthCharrArray,"blynk.cloud", 8080);
144      }
145    }
146    AP_laststate=AP_presentstate;
147    //-------------------------------------------------------------------------
148    buttonStateAnalogSignal= analogRead(Apin); //Analog Signal from photoCoupler
149    if (buttonStateAnalogSignal>700){
150      DoorStatus="Open";
151      PresentState=DoorStatus;
152      }
153    else{
154      DoorStatus="Close";
155      PresentState=DoorStatus;
156      instrucn="Stable State";
157     }
158
159    if(PresentState != LastState){
160       if(PresentState=="Open"){
161         Counter++;
162         lcd.clear();
163         lcd.setCursor(1,0);
164         lcd.print("Door is open");// display that door is currently open
165
166       }
167       if(PresentState =="Close"){
168         digitalWrite(buzzer,LOW);
169         seconds=seconds*0;
170         lcd.clear();
171         lcd.setCursor(1,0);
```

15.Line 148: this code reads the analog signal from photo coupler from the pin A0. if the analog signal is high then door is open other wise the signal is low it means the photo coupler is interrupted and door is closed. The status is stored in “DoorStatus” variable. This variable can be used to send data to internet,

and count the number of times door is open. If door is open then it counts in counter variable also display this on lcd. If it is closed then timer("seconds" variable in the code) which is started when door is open is now stoped by the equation on line 169 in code, make buzzer off clearing the lcd and display on lcd counter and door status.

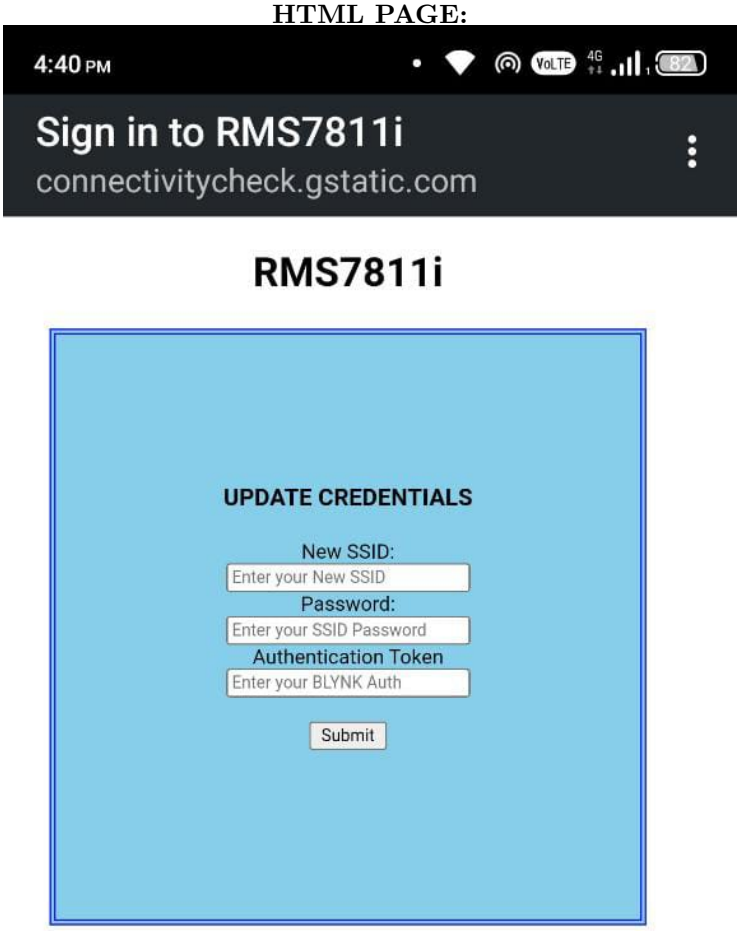
```
171         lcd.setCursor(1,0);
172         lcd.print("Door Closed!");
173         lcd.setCursor(0,1);
174         lcd.print("DoorOpened:"); //display door counter- part1
175         lcd.print(Counter);  //display door counter- part2
176         delay(1000);
177       }
178   LastState=PresentState;
179     }
180   if(PresentState=="Open"){
181         seconds++;
182         delay(1000);
183         lcd.setCursor(0,1); // second row of display
184         lcd.print("Since:");
185         lcd.print(seconds);
186         lcd.print("s");
187         if(seconds>=60){
188           instrucn="Buzzer is ON";
189           digitalWrite(buzzer,HIGH);
190           }
191   }
192   if(PresentState =="Close"){
193     delay(1000);
194   }
195
196   //----------------------
197   if (WiFi.status()!=WL_CONNECTED){
198     lcd.setCursor(13,1);
199     lcd.print("   ");
```

16.Other thing is explained here, the timer started here by increment seconds variable in delay with 1000 milli seconds. Line 187: When door is open for 60 seconds or more the buzzer is start beeping. This buzzer will stop when door will closed. We know that when door is open timer starts and delay of 1000 milli seconds to balance this delay we have delayed 1000 milli seconds in closed state also. Line 197: if wifi is not connected then iot indicator is cleared from lcd.

```
200  }
201  if (WiFi.status()==WL_CONNECTED){
202    lcd.setCursor(13,1);
203    lcd.print("IoT");
204    int buzzerindicatortoiot=seconds;
205    Blynk.virtualWrite(V1,DoorStatus);//String Variable open or closed
206    Blynk.virtualWrite(V2,seconds);//timer
207    Blynk.virtualWrite(V3,Counter);//doorcounter
208    Blynk.virtualWrite(V4,instrucn);
209    Blynk.virtualWrite(V5,buzzerindicatortoiot);
210    Blynk.run();//Run the Blynk library
211   }
212  //-----------------
213
214
215  //--------------------------
216  //--------------------------
217  //----------------------for continously handle server when it request--
218  server.handleClient();
219  //-----------------------------
220  //-------------DNSServer Always available to AP mode-------
221  SignInRequiredMsg.processNextRequest();
222  //----------------------
223  Serial.println(newssid);
224  Serial.println(newpass);
225  Serial.println(newAuth);
226  }
```

17.Line 201: if wifi is connected , iot indicator is displayed on lcd."blynk.virtualwrite" is used to send data to Blynk server on different virtual pins(V1,V2,V3,V4,V5). "Blynk.run" is necessary to send data continuously to the server and run all the blynk function. "server.handleClient" for continuously handle server when it request in Access point mode. "userdefinedname. processNextRequest" for DNS server always available to Access point. "Serial.println" is used for debugging can see on serial monitor 9600 to check which ssid,password and authentication id is used by ESP8266 or we can say our system name "RMS7811i".

HTML PAGE:


4:40 PM
Sign in to RMS7811i
connectivitycheck.gstatic.com
RMS7811i
UPDATE CREDENTIALS
New SSID:
Enter your New SSID
Password:
Enter your SSID Password
Authentication Token
Enter your BLYNK Auth
Submit

GALLERY

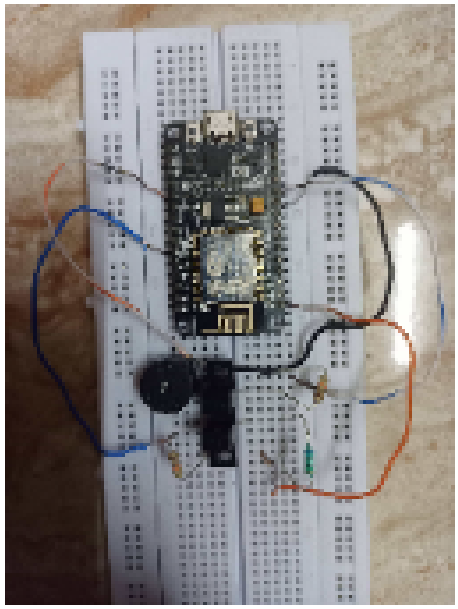


Figure 8: Testing Circuit 1

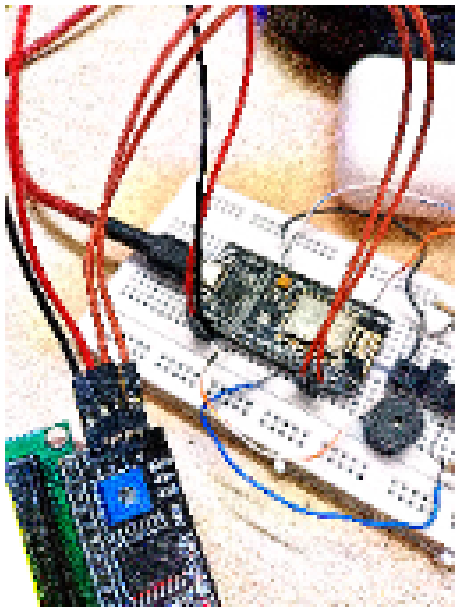
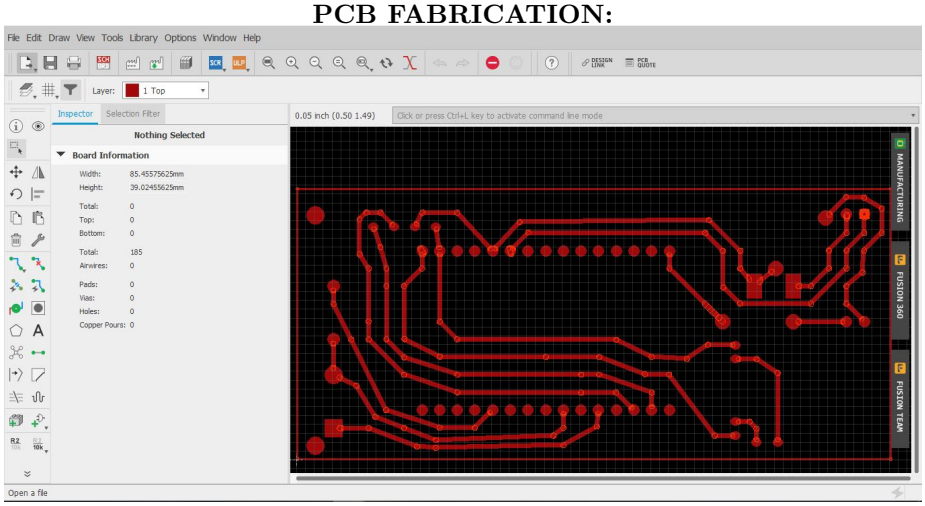
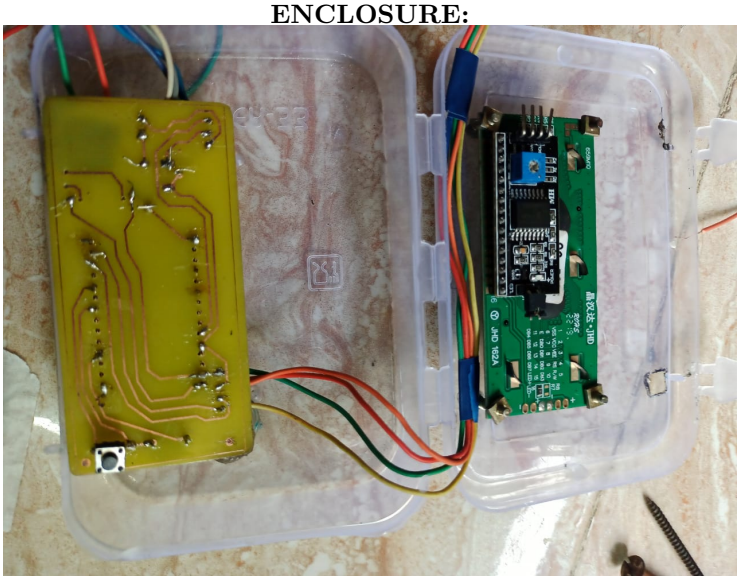


Figure 9: Testing Circuit 2

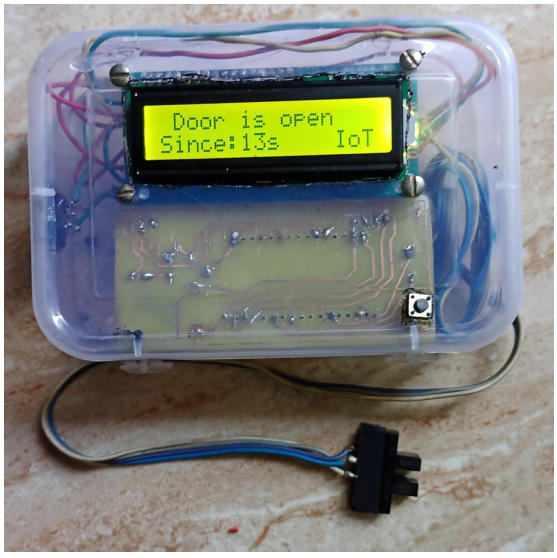
PCB FABRICATION:



ENCLOSURE:



FINAL ASSEMBLED PROJECT:



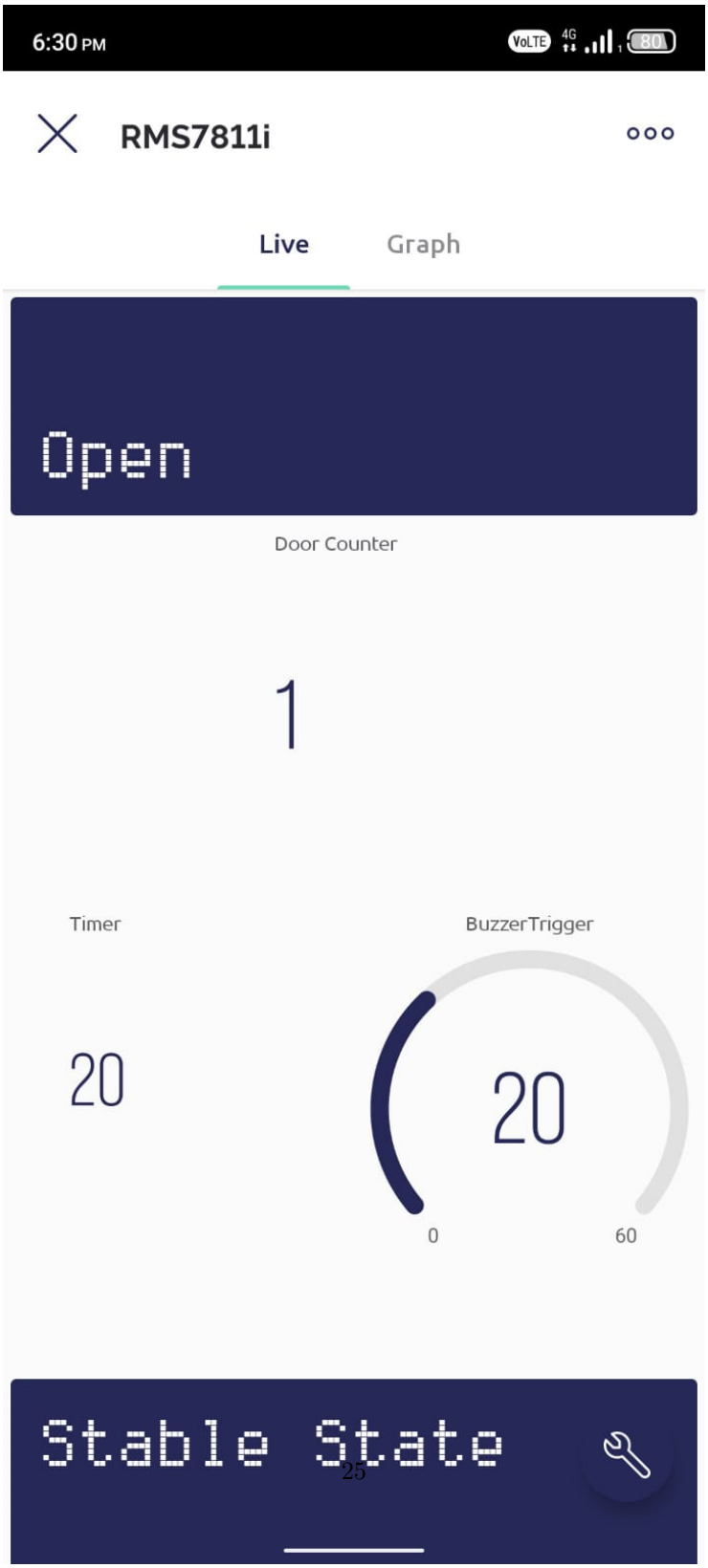


Figure 10: Blynk App Interface:

Figure 11: Blynk App Interface: