Department of Electronic and Telecommunication Engineering
University of Moratuwa

EN2160: Electronic Design Realization

# Product Report of Soil Moisture Meter

Name- D.P.C.L.Dombawala.

Index- 200148M

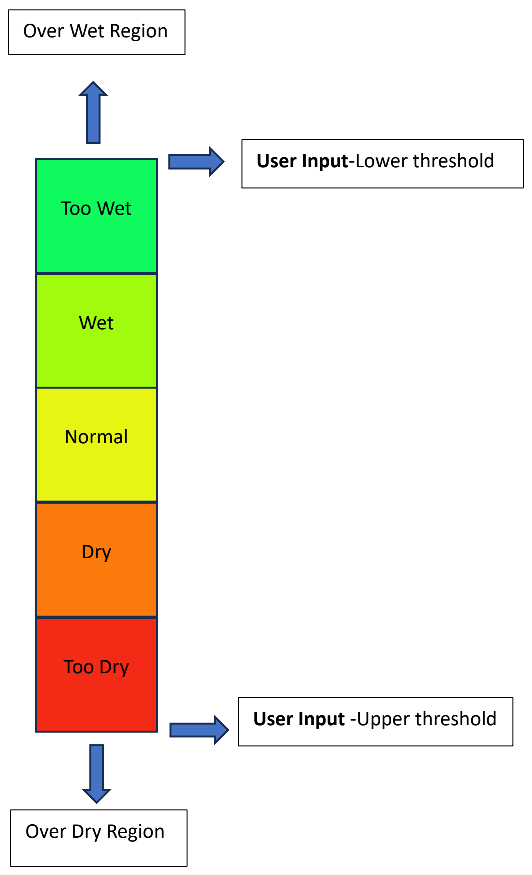
## Contents

# Soil Moisture Meter

## Product Description

A soil moisture meter with temperature and humidity measurement is a handy tool for measuring both the moisture level and temperature of soil. It usually consists of a probe or sensor that is inserted into the soil, which provides readings of the moisture level and temperature of the soil. This tool helps users make informed decisions about watering practices, soil management, and plant care, resulting in healthy and thriving plants. By maintaining optimal soil moisture, temperature and humidity level, gardeners and farmers can ensure the best possible growing conditions for their plants.

## Moisture Scale

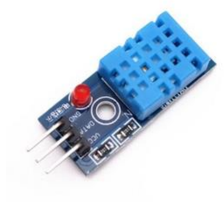


**Soil Moisture Sensor:**

The soil moisture sensor is a fundamental component of your Soil Moisture Meter, designed to accurately measure the water content in the soil. It employs innovative technology to assess soil moisture levels based on conductivity or capacitance changes in the soil. The sensor features a sturdy probe that can be conveniently inserted into the soil to obtain precise readings. With its high sensitivity and reliability, this sensor plays a crucial role in providing essential data for optimizing irrigation schedules, preventing water wastage, and promoting healthy plant growth. It empowers users to make informed decisions about watering their plants, ensuring they receive the ideal amount of moisture to thrive.

**DHT11 Temperature and Humidity Sensor:**

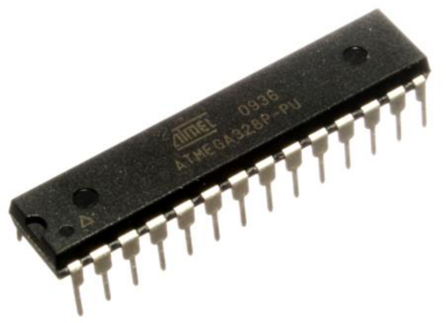
The DHT11 sensor is a sophisticated component integrated into your Soil Moisture Meter, designed to measure both ambient temperature and relative humidity. Its advanced sensing elements enable accurate and rapid detection of temperature changes in the surrounding environment. Simultaneously, it provides real-time readings of the humidity level, offering valuable insights into the moisture content of the air. This sensor's exceptional accuracy and stability make it an indispensable tool for monitoring environmental conditions that impact soil moisture levels and plant health. By leveraging the data from the DHT11 sensor, users can create the optimal environment for their plants, ensuring they thrive in the most suitable temperature and humidity conditions.



- Humidity measurement range: 0-100% with accuracy of +/- 2%
- Temperature measurement range: -10°C to 70°C with accuracy of +/- 0.5°C

**Microcontroller - ATmega328P:**

At the heart of your Soil Moisture Meter lies the ATmega328P microcontroller, which serves as the brain of the device. This powerful and versatile microcontroller efficiently processes data from the soil moisture sensor, DHT11 sensor, and any other connected peripherals. Its advanced capabilities enable the implementation of various smart features, such as menu navigation, user interface management, and precise control over the device's functionalities. The ATmega328P microcontroller ensures seamless interaction with users, allowing them to set custom moisture thresholds, access different modes, and receive timely alerts when soil moisture levels exceed specified limits. Its presence enhances the overall user experience and enables the Soil Moisture Meter to be a sophisticated and reliable tool for agriculture, gardening, and environmental monitoring.

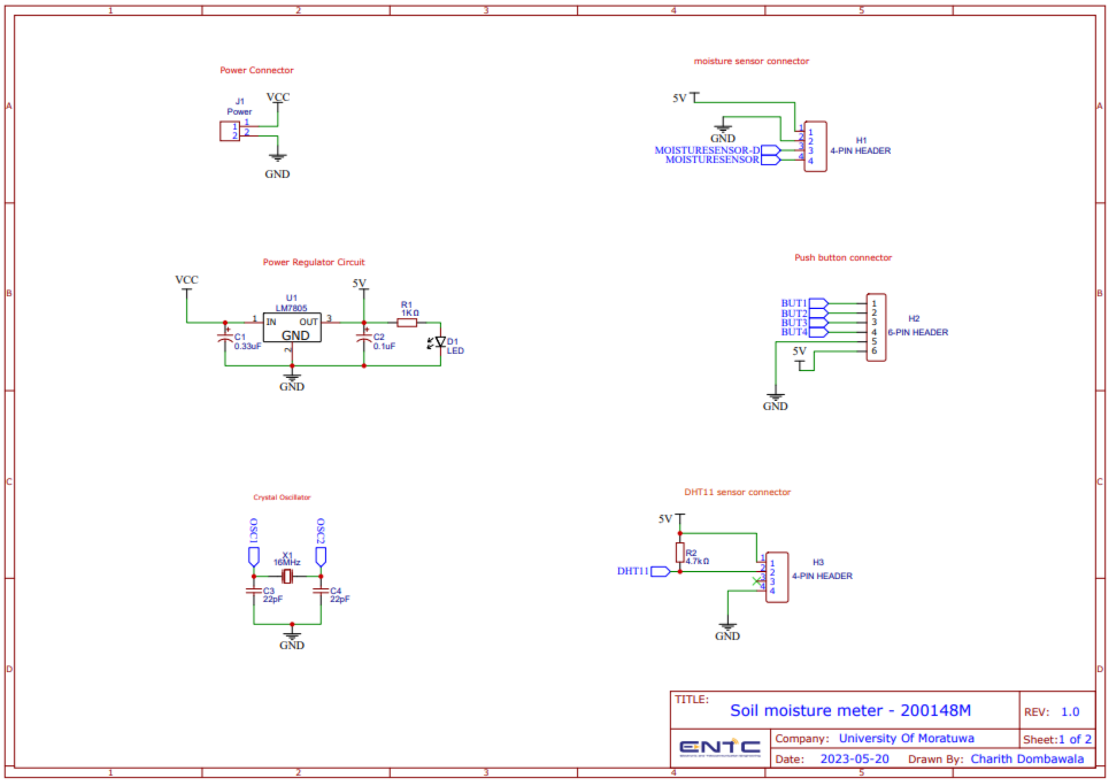


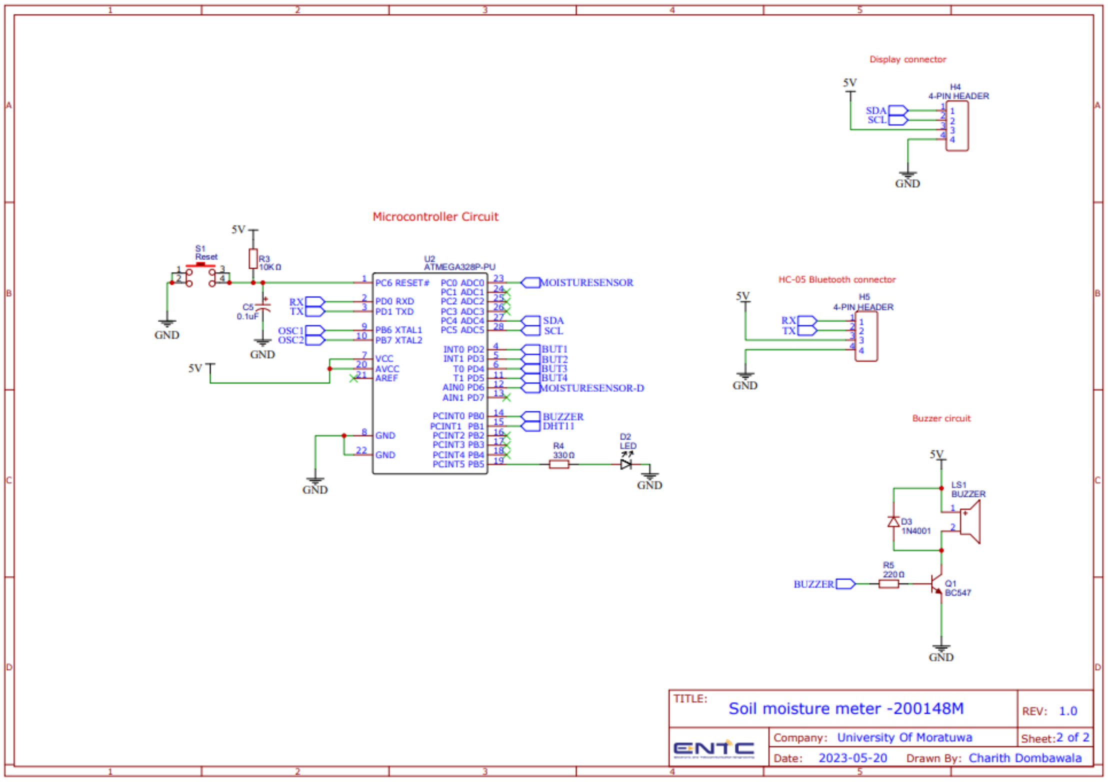
By combining the soil moisture sensor, DHT11 temperature and humidity sensor, and the ATmega328P microcontroller, your Soil Moisture Meter boasts a powerful and comprehensive design. It empowers users with precise and real-time information on soil moisture, environmental temperature, and humidity, enabling them to make well-informed decisions for efficient plant care and sustainable agricultural practices. The integration of these cutting-edge components ensures that your Soil Moisture Meter is an invaluable asset for plant enthusiasts, farmers, and anyone seeking to optimize their gardening and farming endeavors.

## Features and Specifications

- Power ON/OFF button: Allows the user to turn the device on and off as needed to conserve battery life.
- LCD display: Shows the current moisture level, temperature, humidity, and alarm status in an easy-to-read format.
- Buzzer: Provides an audible alert when an alarm is triggered, indicating that the moisture, humidity or temperature levels have exceeded the user-defined thresholds**.**
- LED indicators: Several different colored LED lights indicate the status of the soil moisture, humidity, and temperature levels, with different colors indicating different levels of each.
- Keypad: Allows the user to input specific information, such as the desired moisture or humidity thresholds and the alarm settings.
- Rechargeable battery: The device is equipped with a rechargeable battery.
- Soil moisture sensor: Measures the moisture content of the soil and displays it on the LED display.
- humidity: Measures the humidity level of the environment displays it on the LED display.
- Temperature sensor: Measures the temperature of the environment and displays it on the LED display.
- Moisture measurement range: 0-100% with accuracy of +/- 2%
- Humidity measurement range: 0-100% with accuracy of +/- 2%
- Temperature measurement range: -10°C to 70°C with accuracy of +/- 0.5°C
- Alarms and thresholds: User can set multiple alarms/thresholds for different plants or soil types.
- Material: Made with durable, water resistant material for long-term use

## Schematic Diagram





## Bill of material

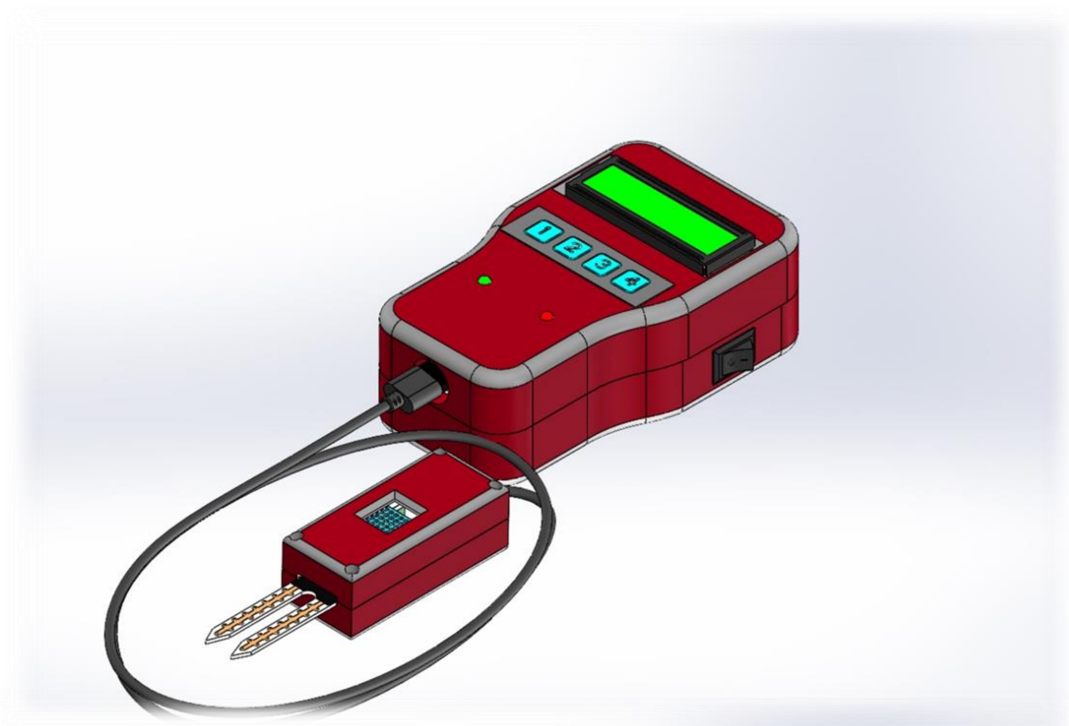
| ID | Name | Designator | Quantity | Supplier | Unit price (USD) | Price (USD) |
|---|---|---|---|---|---|---|
| 1 | 0.33uF | C1 | 1 | LCSC | 0.007 | 0.007 |
| 2 | 0.1uF | C2, C6 | 2 | LCSC | 0.007 | 0.014 |
| 3 | 22pF | C3, C4 | 2 | LCSC | 0.007 | 0.014 |
| 4 | 100nF | C5 | 1 | LCSC | 0.007 | 0.007 |
| 5 | LED-0603_R | D1, D2 | 2 | LCSC | 0.016 | 0.032 |
| 6 | LED-0603_G | D3 | 1 | LCSC | 0.027 | 0.027 |
| 7 | LED | D4 | 1 | LCSC | 0.027 | 0.027 |
| 8 | 1N4001 | D5 | 1 | LCSC | 0.034 | 0.034 |
| 9 | 4-PIN HEADER | H4, H6, H7 | 3 | LCSC | 0.072 | 0.216 |
| 10 | 6-PIN HEADER | H8 | 1 | LCSC | 0.072 | 0.072 |
| 11 | Power | J1 | 1 | LCSC | 0.01 | 0.01 |
| 12 | AC Power Input | J2 | 1 | LCSC | 0.01 | 0.01 |
| 13 | Output | J3 | 1 | LCSC | 0.01 | 0.01 |
| 14 | BUZZER | LS1 | 1 | LCSC | 0.272 | 0.272 |
| 15 | BC547 | Q1 | 1 | LCSC | 0.092 | 0.092 |
| 16 | 1kΩ | R1 | 1 | LCSC | 0.007 | 0.007 |
| 17 | 2kΩ | R2 | 1 | LCSC | 0.007 | 0.007 |
| 18 | 470Ω | R3, R4 | 2 | LCSC | 0.007 | 0.014 |
| 19 | 10kΩ | R5 | 1 | LCSC | 0.007 | 0.007 |
| 20 | 4.7kΩ | R6 | 1 | LCSC | 0.007 | 0.007 |
| 21 | 10KΩ | R7 | 1 | LCSC | 0.007 | 0.007 |
| 22 | 330Ω | R8 | 1 | LCSC | 0.007 | 0.007 |
| 23 | 220Ω | R9 | 1 | LCSC | 0.007 | 0.007 |
| 24 | K2-3.6×6.1_SMD | S1 | 1 | LCSC | 0.069 | 0.069 |
| 25 | Reset | S2 | 1 | LCSC | 0.016 | 0.016 |
| 26 | LM7805 | U1 | 1 | LCSC | 0.259 | 0.259 |
| 27 | MCP73831 | U2 | 1 | LCSC | 1.034 | 1.034 |
| 29 | ATMEGA328P-PU | U4 | 1 | LCSC | 4.249 | 4.249 |
| 30 | 16MHz | X1 | 1 | LCSC | 0.123 | 0.123 |
|  |  |  |  |  | **Total Price =** | 6.657 |

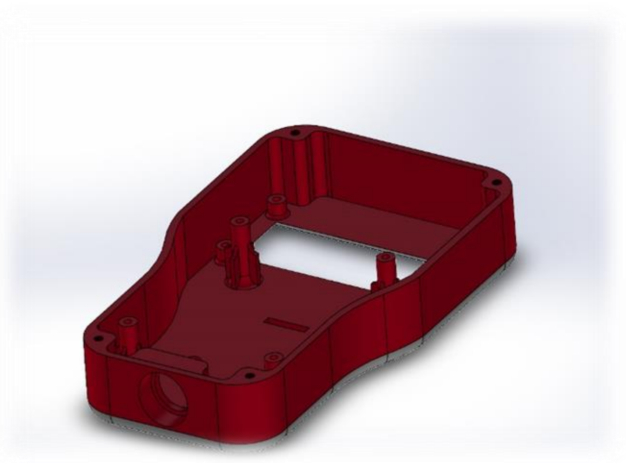
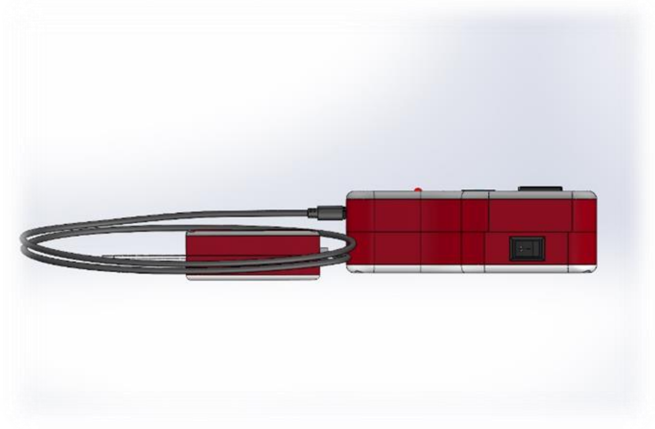
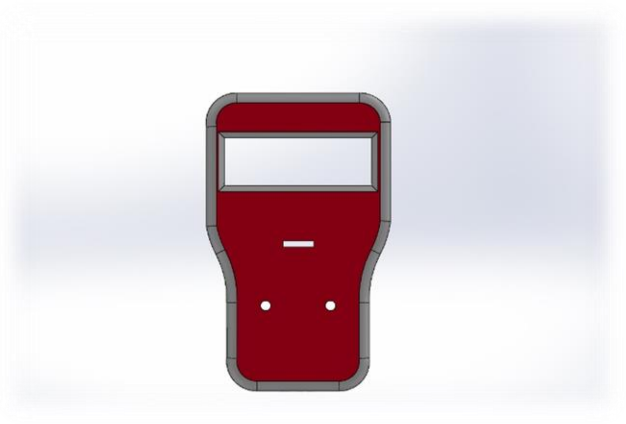
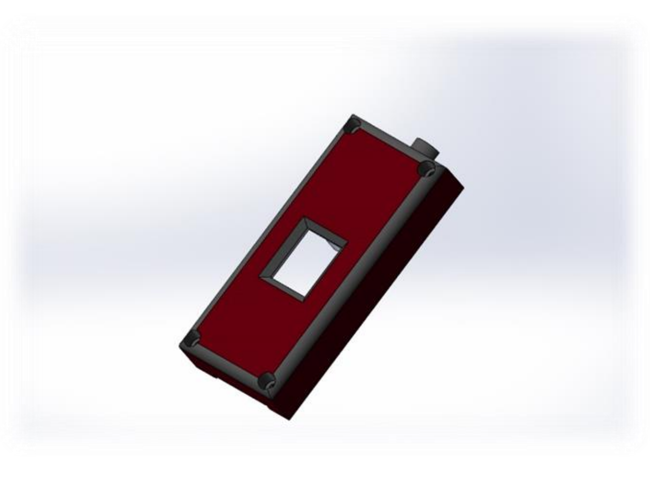
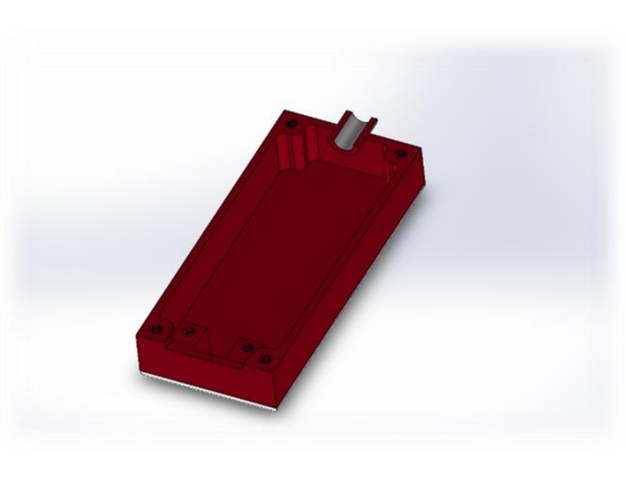
| | |
|---|---|
| PCB manufacturing Cost (from JLCPCB) | 4.7 USD |
| FedEx shipping cost | 2.72 USD |
| Discount | 1.02 USD |
| Total | 6.4 USD |
| Total in LKR | 2167.168 |
| Tax in LKR | 330 |
| Total PCB cost in LKR | 2497.168 |
| Total PCB cost in USD | 7.37 USD |

| | |
|---|---|
| Enclosure manufacturing Cost (from Spatia) | 3300 LKR |
| Delivery cost | 500 LKR |
| Total cost for enclosure | 3800 LKR |
| Total cost for enclosure (USD) | 11.22 USD |

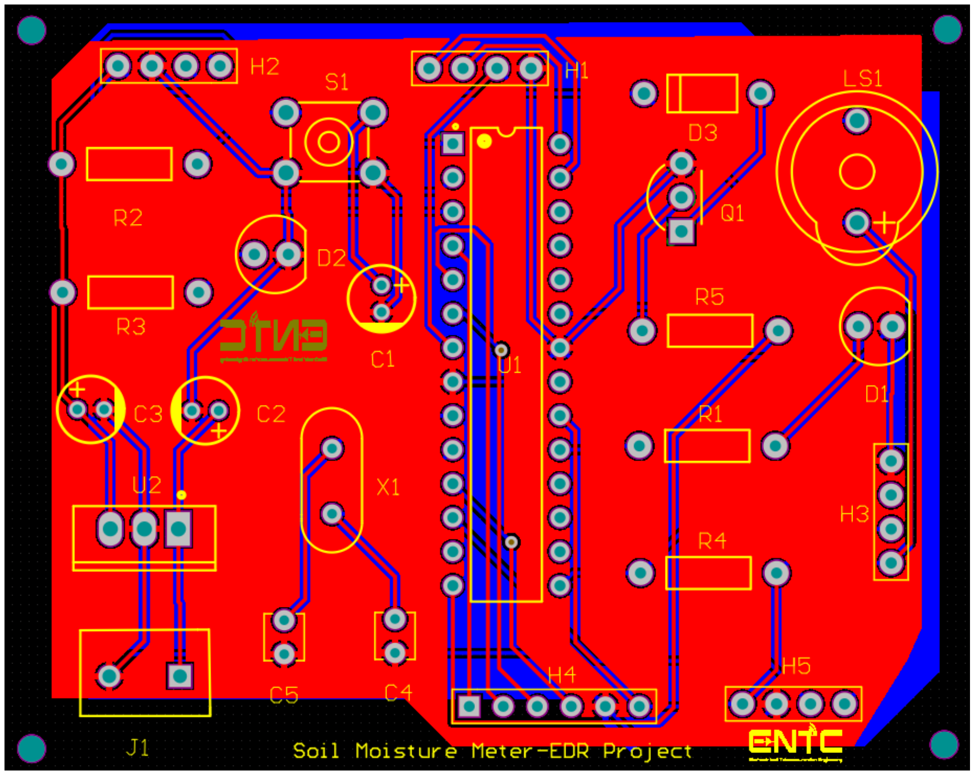
| | |
|---|---|
| **Total cost of Product** | **8549.13 LKR** |
| **Total cost of Product (USD)** | **25.247 USD** |

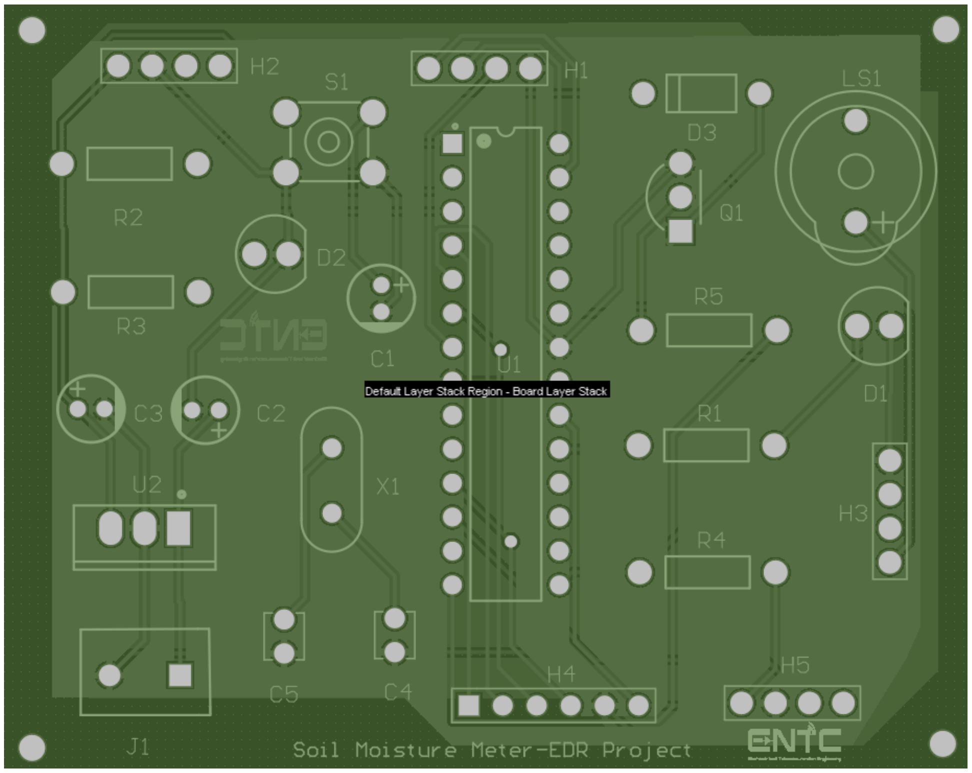
## Enclosure Design



## PCB Design


H2
S1
H1
D3
LS1
R2
D2
Q1
R5
R3
C1
U1
C3
C2
D1
R1
U2
X1
H3
R4
H4
H5
C5
C4
J1
Soil Moisture Meter-EDR Project
ENTC


Default Layer Stack Region - Board Layer Stack
Soil Moisture Meter-EDR Project
ENTC

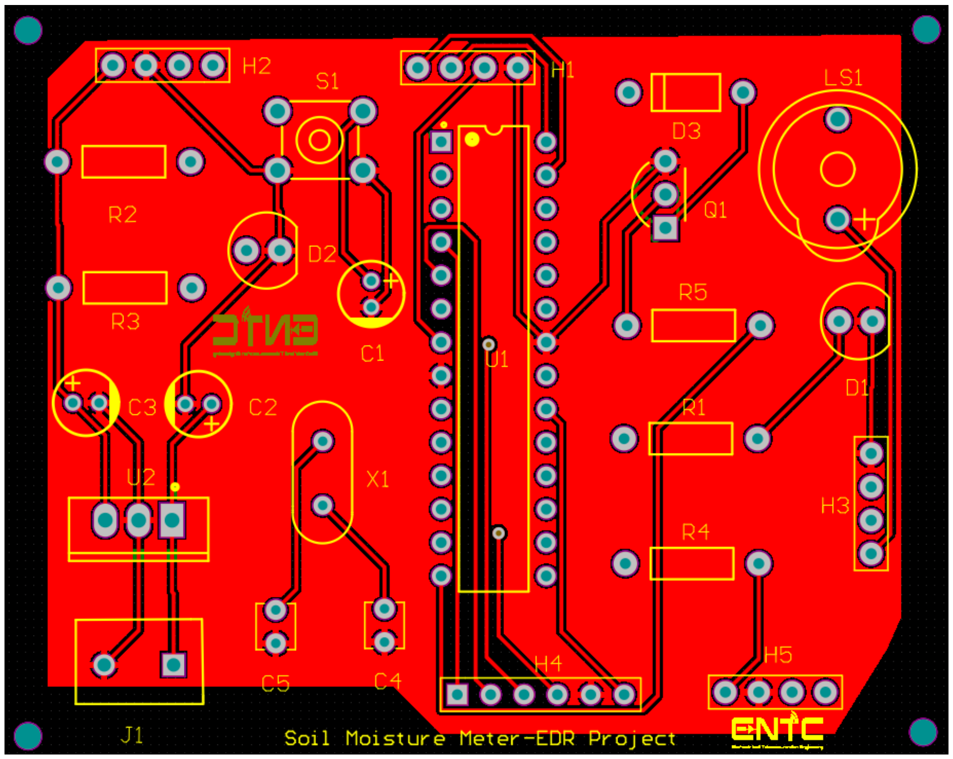


*Figure 1 - Top Layer*

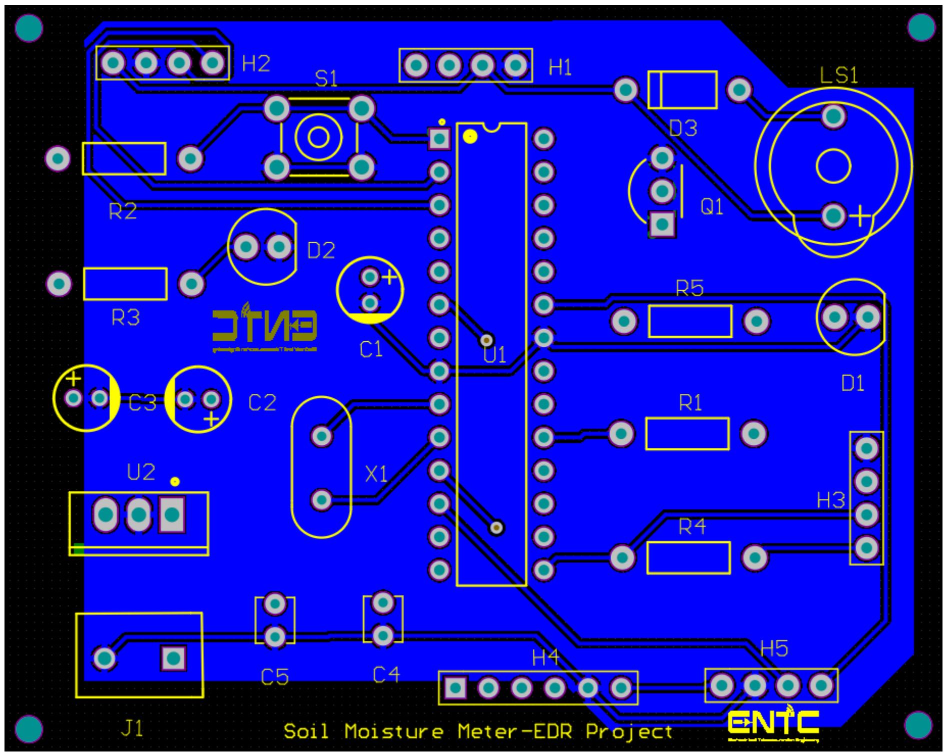


*Figure 2 - Bottom layer*

## Compiled PCB Gerber files and NC drill files

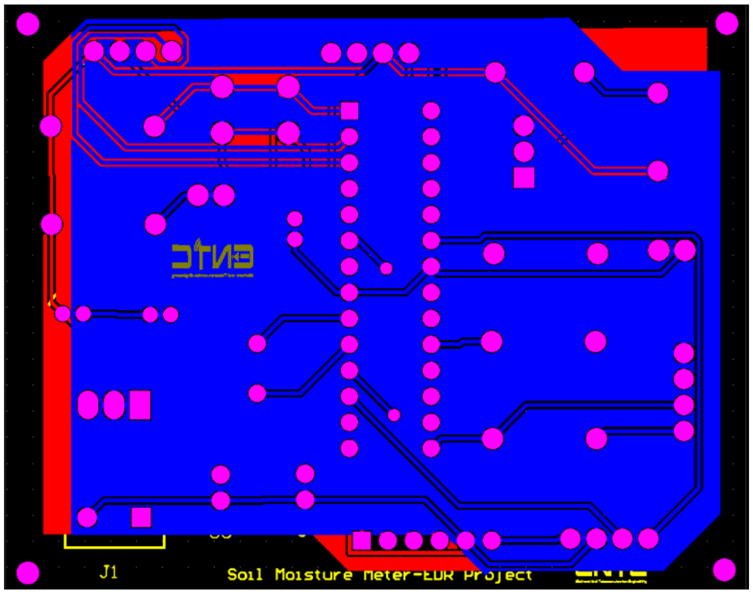


*Figure 3- Gerber files*

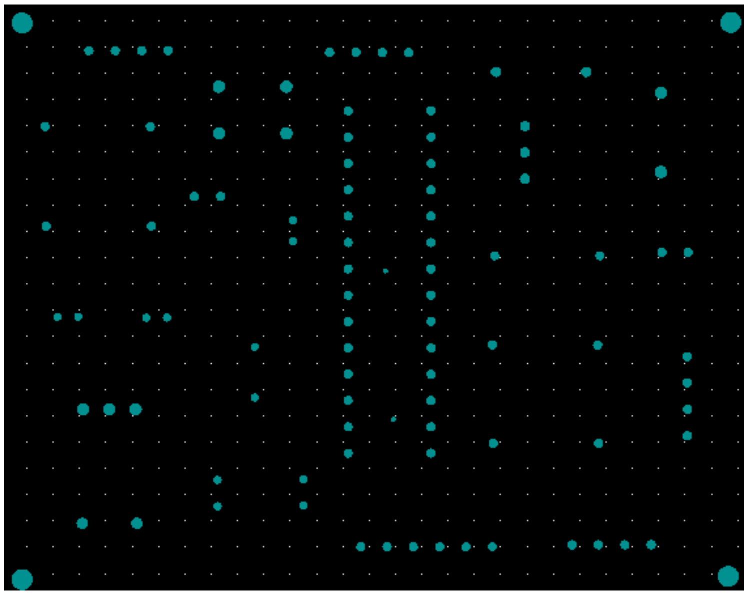


*Figure 4- NC drill drawing*

## Instructions For assembling.

1. Gather Components: Ensure you have all the necessary components and tools before starting the assembly process. These may include the PCB, enclosure, soil moisture sensor, temperature and humidity sensors, microcontroller (if applicable), resistors, capacitors, connectors, and any other electronic components required for your specific design.
2. PCB Assembly: Begin by soldering the electronic components onto the PCB. Start with the smaller components first, such as resistors and capacitors, and then proceed to larger components like connectors and sensors. Double-check the component values and orientations during this process to avoid any errors.
3. Sensor Integration: Carefully attach the soil moisture sensor, temperature sensor, and humidity sensor to their designated locations on the PCB. Ensure proper connections and follow the datasheets or guidelines provided for each sensor.
4. Microcontroller (if applicable): If your design includes a microcontroller, now is the time to integrate it onto the PCB. Make sure to program the microcontroller with the appropriate firmware before soldering it in place.
5. Power Supply: Add the power supply components and make sure to follow safety guidelines for any power-related connections. Double-check for any short circuits or incorrect connections that could potentially damage the circuit.
6. Testing: Before proceeding further, conduct a thorough testing process to ensure the functionality of the assembled circuit. Test the soil moisture sensor, temperature, and humidity sensors to verify their accuracy. Additionally, check if the alarm and indicator systems work as expected.
7. Enclosure Assembly: Once the PCB assembly is verified, carefully place it inside the enclosure. Ensure all the necessary holes for connectors, buttons, and displays align properly with the PCB and components.
8. Secure the PCB: Use appropriate mounting hardware to secure the PCB inside the enclosure, preventing any movement that may cause damage during usage or transportation.
9. Final Connections: Connect any external interfaces, such as USB ports, power jacks, or display modules, as per the design requirements. Double-check all connections for accuracy and security.
10. Final Testing: Perform a comprehensive functional test of the fully assembled Soil Moisture Meter. Verify that all components work together seamlessly and meet the desired specifications.

## How to Test for Functionality

Here are the menu options in the device,

1. Measure moisture
2. Set Moisture
3. Set Temperature
4. Set Humidity
5. Set Time
6. Set Alarm
7. Bluetooth Mode

1. **Power On the Device**: Ensure the device is powered on and the display is functional.
2. **Default Menu**: Upon startup, the device should display the default menu, which is likely the "Measure moisture" option.
3. **Navigation**: Use the "Up" and "Down" buttons to navigate through the menu options. Each press of these buttons should move the highlighted selection up or down the menu list.
4. **Select Menu Item**: Press the "OK/Menu" button to select the highlighted menu item. The action of selecting a menu item should take you to the respective submenu or execute the function associated with that menu item.
5. **Measure Moisture**: Select the "Measure moisture" option from the menu, and the device should display the current soil moisture level.
6. **Set Moisture Threshold**: Select the "Set Moisture" option from the menu, and the device should prompt you to enter the desired moisture threshold. You can use the "Up" and "Down" buttons to adjust the value and the "OK/Menu" button to confirm the threshold setting.
7. **Set Temperature and Humidity**: Repeat the above step for "Set Temperature" and "Set Humidity" options to configure the desired temperature and humidity thresholds.
8. **Set Time**: Select the "Set Time" option to set the device's internal clock. Use the navigation buttons to adjust the hours, minutes, and other time-related settings.
9. **Set Alarm**: Select the "Set Alarm" option to configure the alarm settings for watering or monitoring purposes. Set the alarm time and other relevant parameters as needed.

10. **Bluetooth Mode**: If your Soil Moisture Meter has Bluetooth functionality, select the "Bluetooth Mode" option. This should enable the device to connect with a Bluetooth-enabled device or application for data transmission or remote monitoring.
11. **Cancel/Go Back**: If at any point during the menu navigation you want to cancel an action or go back to the previous menu level, use the "Cancel" button.
12. **Testing Scenarios**: Test various scenarios, such as setting different thresholds, verifying the correct display of values, and ensuring the alarm triggers correctly when thresholds are exceeded.

## Appendix

```
#include <DHT.h>
#include <LiquidCrystal_I2C.h>


byte BUZZER = 8;     //14        // sudu- button 2, kalu button 1,duburu-
button 4, rathu-button 3    //  1 - gnd , 2- button 3, 3- button 4, 4- button
1, 5-button 2
byte LED_1 = 13 ;    //19
byte pb_cancel = 4;  //3
byte pb_ok = 5;      //2
byte pb_down = 2;    //5
byte pb_up = 3 ;     //4
byte DHTpin = 9 ;    //15


byte moisturesensorPower = 6;  //12   //3-nil-moisture  4-kaha-temp  2-rathu-
vcc 1 -kola-gnd
byte moisturesensor= A0;      //23


DHT dht(DHTpin, DHT11);

LiquidCrystal_I2C lcd(0x3F, 16, 2);
//Global variable

byte hours = 10;    // Range: 0-255
byte minutes = 0;  // Range: 0-255
byte seconds = 0;

int timeNow = 0;
int timeLast = 0;

float moisture[] = { 800, 300 };
byte temperature[] = { 40, 15 };
byte humidity[] = { 85, 60 };

bool alarm_enabled = false;
byte alarm_hours[] = { 5 };
byte alarm_minutes[] = { 54 };
bool alarm_triggered = true;

int BUZZERFrequency = 5000;
byte BUZZERDuration = 750;

byte current_mode = 0;
byte n_mode = 7;
String modes[] = { "Measure moisture","Set moisture", "Set Temparature", "Set
Humidity","Set Time", "Set Alarm", "Bluetooth mode" };

void setup() {
  // put your setup code here, to run once:


  pinMode(BUZZER, OUTPUT);
  pinMode(LED_1, OUTPUT);
  pinMode(pb_cancel, INPUT_PULLUP);
  pinMode(pb_ok, INPUT_PULLUP);
  pinMode(pb_down, INPUT_PULLUP);
  pinMode(pb_up, INPUT_PULLUP);
  pinMode(DHTpin, INPUT);
  pinMode(moisturesensorPower, OUTPUT);
  pinMode(moisturesensor, INPUT);

  dht.begin();
  // Initially keep the sensor OFF
  digitalWrite(moisturesensorPower, LOW);

  lcd.init();
  lcd.clear();
  lcd.backlight();  // Make sure backlight is on

  // Print a message on both lines of the LCD.
  lcd.setCursor(5, 0);  //Set cursor to character 2 on line 0
  lcd.print("Welcome");
  delay(2000);
}

void loop() {
  //  // put your main code here, to run repeatedly:
  update_time_with_check_alarm();


  if (digitalRead(pb_up)==LOW){
       delay(200);
       go_to_menu();
  }
  check_sensors();
}


void print_text(String text, int column, int row) {
  lcd.clear();
  lcd.setCursor(column, row);  //Set cursor to character 2 on line 0
  lcd.print(text);
}


void update_time() {
  timeNow = millis() / 1000;
  seconds = timeNow - timeLast;
  if (seconds >= 60) {
    minutes += 1;
    timeLast += 60;
  }
  if (minutes == 60) {
    hours += 1;
    minutes = 0;
  }
  if (hours == 24) {

    hours = 0;
  }
}


void ring_alarm() {

  lcd.clear();
  print_text(F("Alarm"), 0, 0);
  digitalWrite(LED_1, HIGH);

  bool break_happened = false;

  while (break_happened == false && digitalRead(pb_cancel) == HIGH ) {

    if (digitalRead(pb_cancel) == LOW) {
      delay(200);
      break_happened = true;

      break;
    }
    digitalWrite(BUZZER, HIGH);
    tone(BUZZER, BUZZERFrequency);
    delay(BUZZERDuration);
    noTone(BUZZER);
    delay(BUZZERDuration);
  }
  digitalWrite(LED_1, LOW);
  lcd.clear();
}

void update_time_with_check_alarm() {

  update_time();
  if (alarm_enabled) {

    if (alarm_triggered == false && alarm_hours[0] == hours &&
alarm_minutes[0] == minutes && alarm_enabled==true) {
      ring_alarm();
      alarm_triggered = true;
    }
  }
}


int wait_for_button_press() {
  while (true) {
    if (digitalRead(pb_ok) == LOW) {
      delay(200);
      return pb_ok;
    } else if (digitalRead(pb_up) == LOW) {
      delay(200);
      return pb_up;
    } else if (digitalRead(pb_down) == LOW) {
      delay(200);
      return pb_down;
    } else if (digitalRead(pb_cancel) == LOW) {
      delay(200);
      return pb_cancel;
    }
    update_time();
  }
}

int wait_for_button_press2() {
  while (true) {

    lcd.setCursor(8,0);
    lcd.print(String(hours)+":"+String(minutes)+":"+String(seconds));

    if (digitalRead(pb_ok) == LOW) {
      delay(200);
      return pb_ok;
    } else if (digitalRead(pb_up) == LOW) {
      delay(200);
      return pb_up;
    } else if (digitalRead(pb_down) == LOW) {
      delay(200);
      return pb_down;
    } else if (digitalRead(pb_cancel) == LOW) {
      delay(300);
      return pb_cancel;
    }
    update_time();
  }
}



void go_to_menu() {
  while (digitalRead(pb_cancel) == HIGH) {
    lcd.clear();
    print_text((modes[current_mode]), 0, 1);
    lcd.setCursor(12, 0);
    lcd.print("Menu");


    int pressed = wait_for_button_press();

    if (pressed == pb_up) {
      delay(200);
      current_mode += 1;
      current_mode = current_mode % n_mode;
    }

    else if (pressed == pb_down) {
      delay(200);
      current_mode -= 1;
      if (current_mode < 0) {
        current_mode = n_mode - 1;
      }
    }

    else if (pressed == pb_ok) {
      delay(200);
      run_mode(current_mode);
    }

    else if (pressed == pb_cancel) {
      delay(200);
      break;
    }
    update_time();
  }
}


void run_mode(int current_mode) {       //{"Measure moisture","Set moisture",
"Set Temparature", "Set Humidity","Set Time", "Set Alarm", "Bluetooth mode"}
  if (current_mode == 0) {
    Measure_moisture();
  } else if (current_mode == 1) {
    set_moisture();
  } else if (current_mode == 2) {
    set_temp();
  } else if (current_mode == 3) {
    set_humidity();
  } else if (current_mode == 4) {
    set_time();
  } else if (current_mode == 5) {
    alarm();
  } else if (current_mode == 6) {
    print_text(F("BluetoothMode"), 40, 40);
  }
}

void Measure_moisture(){

while(true){
     update_time();
     float step= ((moisture[0]-moisture[1])/5) ;
     int reading=readMoistureSensor();
    if (moisture[1]>reading) {
         print_text("Saturated (" +String(reading)+")",0,1);
         lcd.setCursor(9,0);
         lcd.print(String(reading/10)+"%"+" Dry");
         BlinkLED();
         delay(1000);

  } else if ((moisture[1]<=reading) && (reading <(moisture[1]+step))) {
         print_text("Too Wet ("+String(reading)+")",0,1);
         lcd.setCursor(9,0);
         lcd.print(String(reading/10)+"%"+" Dry");
        delay(1000);

  } else if ((moisture[1]+step<=reading) && (reading <(moisture[1]+2*step)))
{
         print_text("Wet ("+String(reading)+")",0,1);
         lcd.setCursor(9,0);
         lcd.print(String(reading/10)+"%"+" Dry");
         delay(1000);

  } else if ((moisture[1]+2*step<=reading) && (reading<moisture[1]+3*step)) {
         print_text("Noramal ("+String(reading)+")",0,1);
         lcd.setCursor(9,0);
         lcd.print(String(reading/10)+"%"+" Dry");
         delay(1000);
  } else if ((moisture[1]+3*step<=reading ) && (reading<moisture[1]+4*step))
{
         print_text("Dry ("+String(reading)+")",0,1);
         lcd.setCursor(9,0);
         lcd.print(String(reading/10)+"%"+" Dry");
        delay(1000);
  } else if ((moisture[1]+4*step<reading ) && (reading<moisture[1]+5*step)) {
         print_text("Too Dry ("+String(reading)+")",0,1);
         lcd.setCursor(9,0);
         lcd.print(String(reading/10)+"%"+" Dry");
```

```
            delay(1000);
     } else if ((moisture[0]<=reading) && (reading<=1000)) {
            print_text("Over Dry ("+String(reading)+")",0,1);
            lcd.setCursor(9,0);
            lcd.print(String(reading/10)+"%"+" Dry");
            BlinkLED();
            delay(1000);
     } else if (reading>1000) {
            print_text("Over Dry ("+String(reading)+")",0,1);
            lcd.setCursor(8,0);
            lcd.print("100% Dry");
            BlinkLED();
            delay(1000);
      }

    if ((digitalRead(pb_up)==LOW) || (digitalRead(pb_cancel)==LOW)){
          delay(200);
          break;
    }
  }
  }


void set_time() {
  int temp_hour = hours;
  while (true) {

    print_text(("Hour :" + String(temp_hour)), 0, 1);
    lcd.setCursor(8,0);
    lcd.print(String(hours)+":"+String(minutes)+":"+String(seconds));

    int pressed = wait_for_button_press2();

    if (pressed == pb_up) {
      delay(200);
      temp_hour += 1;
      temp_hour = temp_hour % 24;
    }

    else if (pressed == pb_down) {
      delay(200);
      temp_hour -= 1;
      if (temp_hour < 0) {
        temp_hour = 23;
      }
    }

    else if (pressed == pb_ok) {
      delay(200);
      hours = temp_hour;
      break;
    }
```

```
      else if (pressed == pb_cancel) {
        delay(200);
        break;
      }
      update_time();
    }

    int temp_minute = minutes;
    while (true) {

      print_text(("Minute :" + String(temp_minute)), 0, 1);
      lcd.setCursor(8,0);
      lcd.print(String(hours)+":"+String(minutes)+":"+String(seconds));

      int pressed = wait_for_button_press2();

      if (pressed == pb_up) {
        delay(200);
        temp_minute += 1;
        temp_minute = temp_minute % 60;
      }

      else if (pressed == pb_down) {
        delay(200);
        temp_minute -= 1;
        if (temp_minute < 0) {
          temp_minute = 59;
        }
      }

      else if (pressed == pb_ok) {
        delay(200);
        minutes = temp_minute;
        break;
      }

      else if (pressed == pb_cancel) {
        delay(200);
        break;
      }
      update_time();
    }

    lcd.clear();
    print_text(F("Time is set"), 0, 0);
    delay(1500);
  }

  void alarm(){
    while (true){

   print_text("To enable- Up",0,0);
   lcd.setCursor(0,1);
```

```
 lcd.print("To disable- Down");

 int pressed = wait_for_button_press();

 if (pressed == pb_up) {
      delay(200);
      alarm_enabled=true;
      set_alarm();
      break;
    }

  else if (pressed == pb_down) {
      delay(200);
      alarm_enabled=false;
      break;
      }
 update_time();
}
}

void set_alarm() {
  int temp_hour = alarm_hours[0];
  while (true) {
    print_text(("Hour :" + String(temp_hour)), 0, 1);
    lcd.setCursor(8,0);
    lcd.print(String(hours)+":"+String(minutes)+":"+String(seconds));

    int pressed = wait_for_button_press2();

    if (pressed == pb_up) {
      delay(200);
      temp_hour += 1;
      temp_hour = temp_hour % 24;
    }

    else if (pressed == pb_down) {
      delay(200);
      temp_hour -= 1;
      if (temp_hour < 0) {
        temp_hour = 23;
      }
    }

    else if (pressed == pb_ok) {
      delay(200);
      alarm_hours[0] = temp_hour;
      break;
    }

    else if (pressed == pb_cancel) {
      delay(200);
      break;
    }
```

```
      update_time();
    }

    int temp_minute = alarm_minutes[0];
    while (true) {

      print_text(("Minute :" + String(temp_minute)), 0, 1);
      lcd.setCursor(8,0);
      lcd.print(String(hours)+":"+String(minutes)+":"+String(seconds));

      int pressed = wait_for_button_press2();

      if (pressed == pb_up) {
        delay(200);
        temp_minute += 1;
        temp_minute = temp_minute % 60;
      }

      else if (pressed == pb_down) {
        delay(200);
        temp_minute -= 1;
        if (temp_minute < 0) {
          temp_minute = 59;
        }
      }

      else if (pressed == pb_ok) {
        delay(200);
        alarm_minutes[0] = temp_minute;
        break;
      }

      else if (pressed == pb_cancel) {
        delay(200);
        break;
      }
      update_time();
    }

    lcd.clear();
    print_text(F("Alarm is set"), 0, 0);
    alarm_triggered= false;
    delay(1500);
  }

  void set_moisture() {
    int m_upper_threshold = moisture[0];
    while (true) {
      print_text(("Upper Level: " + String(m_upper_threshold)), 0, 1);
      lcd.setCursor(6, 0);
      lcd.print("M_Now: " + String(readMoistureSensor()));
      int pressed = wait_for_button_press();

      if (pressed == pb_up) {
        delay(200);
        m_upper_threshold += 25;
        m_upper_threshold = m_upper_threshold % 1000;
      }

      else if (pressed == pb_down) {
        delay(200);
        m_upper_threshold -= 25;
        if (m_upper_threshold < 0) {
          m_upper_threshold = 1000;
        }
      }

      else if (pressed == pb_ok) {
        delay(200);
        moisture[0] = m_upper_threshold;
        break;
      }

      else if (pressed == pb_cancel) {
        delay(200);
        break;
      }
      update_time();
    }

    int m_lower_threshold = moisture[1];
    while (true) {
      print_text(("Lower Level:" + String(m_lower_threshold)), 0, 1);
      lcd.setCursor(6, 0);
      lcd.print("M_Now: " + String(readMoistureSensor()));

      int pressed = wait_for_button_press();

      if (pressed == pb_up) {
        delay(200);
        m_lower_threshold += 25;
        m_lower_threshold = m_lower_threshold % 1000;
      }

      else if (pressed == pb_down) {
        delay(200);
        m_lower_threshold -= 25;
        if (m_lower_threshold < 0) {
          m_lower_threshold = 1000;
        }
      }

      else if (pressed == pb_ok) {
        delay(200);
```

```
        moisture[1] = m_lower_threshold;
        break;
      }

      else if (pressed == pb_cancel) {
        delay(200);
        break;
      }
      update_time();
    }

    lcd.clear();
    print_text(F("Moisture Levels"), 0, 0);
    lcd.setCursor(0, 1);
    lcd.print("are set");

    delay(1500);
  }


  void set_temp() {
    int upper_threshold = temperature[0];
    while (true) {
      print_text(("Upper Level: " + String(upper_threshold)), 0, 1);
      lcd.setCursor(6, 0);
      lcd.print("T_Now: " + String(dht.readTemperature()));

      int pressed = wait_for_button_press();

      if (pressed == pb_up) {
        delay(200);
        upper_threshold += 1;
        upper_threshold = upper_threshold % 50;
      }

      else if (pressed == pb_down) {
        delay(200);
        upper_threshold -= 1;
        if (upper_threshold < 0) {
          upper_threshold = 50;
        }
      }

      else if (pressed == pb_ok) {
        delay(200);
        temperature[0] = upper_threshold;
        break;
      }

      else if (pressed == pb_cancel) {
        delay(200);
        break;
      }
      update_time();
    }

    int lower_threshold = temperature[1];
    while (true) {

      print_text(("Lower Level: " + String(lower_threshold)), 0, 1);
      lcd.setCursor(6, 0);
      lcd.print("T_Now: " + String(dht.readTemperature()));

      int pressed = wait_for_button_press();

      if (pressed == pb_up) {
        delay(200);
        lower_threshold += 1;
        lower_threshold = lower_threshold % 50;
      }

      else if (pressed == pb_down) {
        delay(200);
        lower_threshold -= 1;
        if (lower_threshold < 0) {
          lower_threshold = 50;
        }
      }

      else if (pressed == pb_ok) {
        delay(200);
        temperature[1] = lower_threshold;
        break;
      }

      else if (pressed == pb_cancel) {
        delay(200);
        break;
      }
      update_time();
    }

    print_text(F("Temp Levels are"), 0, 0);
    lcd.setCursor(0, 1);
    lcd.print("set");
    delay(1500);
  }
  void set_humidity() {
    int h_upper_threshold = humidity[0];
    while (true) {

      print_text(("Upper Level: " + String(h_upper_threshold)), 0, 1);
      lcd.setCursor(6, 0);
      lcd.print("H_Now: " + String(dht.readHumidity()));

      int pressed = wait_for_button_press();
      if (pressed == pb_up) {
        delay(200);
        h_upper_threshold += 1;
        h_upper_threshold = h_upper_threshold % 100;
      }

      else if (pressed == pb_down) {
        delay(200);
        h_upper_threshold -= 1;
        if (h_upper_threshold < 0) {
          h_upper_threshold = 100;
        }
      }

      else if (pressed == pb_ok) {
        delay(200);
        temperature[0] = h_upper_threshold;
        break;
      }

      else if (pressed == pb_cancel) {
        delay(200);
        break;
      }
      update_time();
    }

    int h_lower_threshold = humidity[1];
    while (true) {

      print_text(("Lower Level: " + String(h_lower_threshold)), 0, 1);
      lcd.setCursor(6, 0);
      lcd.print("H_Now: " + String(dht.readHumidity()));

      int pressed = wait_for_button_press();

      if (pressed == pb_up) {
        delay(200);
        h_lower_threshold += 1;
        h_lower_threshold = h_lower_threshold % 50;
      }

      else if (pressed == pb_down) {
        delay(200);
        h_lower_threshold -= 1;
        if (h_lower_threshold < 0) {
          h_lower_threshold = 50;
        }
      }

      else if (pressed == pb_ok) {
        delay(200);
        humidity[1] = h_lower_threshold;
        break;
      }

      else if (pressed == pb_cancel) {
        delay(200);
        break;
      }
      update_time();
    }

    lcd.clear();
    print_text(F("Humidity Levels"), 0, 0);
    lcd.setCursor(0, 1);
    lcd.print("are set");
    delay(1500);
  }
  int readMoistureSensor() {
    digitalWrite(moisturesensorPower, HIGH);  // Turn the sensor ON
    delay(100);                                // Allow power to settle
    int val = analogRead(moisturesensor);     // Read the analog value form
  sensor
    digitalWrite(moisturesensorPower, LOW);   // Turn the sensor OFF
    return val;                               // Return analog moisture value
  }

  void check_sensors() {

    print_text("Moisture: " + String(readMoistureSensor()), 0, 0);
    if (readMoistureSensor() > moisture[0]) {
        BlinkLED();
        lcd.setCursor(0, 1);
        lcd.print("M_Level High");
      }
    else if (readMoistureSensor() < moisture[1]) {
        lcd.setCursor(0, 1);
        lcd.print("M_Level LOW");
        ring_Buzzer();
    }
    delay(1000);

    print_text("Temperature:" + String(int(dht.readTemperature())), 0, 0);
    lcd.setCursor(14, 0);
    lcd.print("'C");
    if (dht.readTemperature() > temperature[0]) {
        lcd.setCursor(0, 1);
        lcd.print("T_Level High");
        BlinkLED();
    }
    else if (dht.readTemperature() < temperature[1]) {
        lcd.setCursor(0, 1);
        lcd.print("T_Level LOW");
        BlinkLED();
    }
    delay(1000);

    print_text("Humidity: " + String(int(dht.readHumidity())), 0, 0);
    lcd.setCursor(12, 0);
    lcd.print("%");
    if (dht.readHumidity() > humidity[0]) {
        lcd.setCursor(0, 1);
        lcd.print("H_Level High");
        BlinkLED();
    }
    else if (dht.readHumidity() < humidity[1]) {
        lcd.setCursor(0, 1);
        lcd.print("H_Level LOW");
        BlinkLED();
    }
    delay(1000);
  }

  void BlinkLED() {
    digitalWrite(LED_1, HIGH);
    delay(1000);
    digitalWrite(LED_1, LOW);
    delay(1000);
  }
  void ring_Buzzer() {
    print_text(F("Please water"), 1, 0);
    lcd.setCursor(2, 1);
    lcd.print("your plants");


    bool break_happened = false;

    while (break_happened == false && digitalRead(pb_cancel) == HIGH) {

      if (digitalRead(pb_cancel) == LOW) {
        delay(200);
        break_happened = true;
        set_moisture();
        break;
      }
      BlinkLED();
      digitalWrite(BUZZER, HIGH);
      tone(BUZZER, BUZZERFrequency);
      delay(BUZZERDuration);
      noTone(BUZZER);
      delay(BUZZERDuration);
    }

  }
```