ACE6263

EMBEDDED IoT SYSTEMS & APP

Assignment

Group 4

No	Name	Student ID	Major	Email
1	Tang Chin Wen	1191201278	LE	1191201278@student.mmu.edu.my
2	THANESHWARAN A/L RAVICHANDRAN	1201101420	LE	1201101420@student.mmu.edu.my
3	MAHENDRAN A/L SURESH	1191101755	TE	1191101755@student.mmu.edu.my

Smart Soil Moisture Monitoring System

Introduction:

The Embedded IoT-based Smart Soil Moisture Monitoring System is a project aimed at automating plant care through real-time monitoring of environmental and soil conditions. This system seeks to enhance farming productivity and decrease manual work by ensuring that plants are given sufficient water according to real soil moisture conditions. It is especially beneficial for home gardening, greenhouse management, and small-scale agricultural uses.

At the heart of this project is the ESP32-WROOM-32 microcontroller, offering processing capabilities and wireless communication. A soil moisture sensor measures the amount of water in the soil, whereas a DHT11 sensor tracks the surrounding temperature and humidity. When the ground gets too arid, the system automatically triggers a small water pump to water the plant. Visual information is displayed on a 0.96" OLED screen that presents real-time sensor data. Moreover, an LED and a buzzer serve to offer visual and auditory notifications for low moisture levels or system operations.

By combining these elements, the system delivers an effective, immediate solution for intelligent irrigation, assisting in water conservation and encouraging healthier plant development. This project showcases the real-world use of embedded systems and IoT in precision agriculture

Appendix:

1 ESP32 Dev Board

2 DHT11 Sensor

3 Soil Moisture Sensor

4 OLED Display (SSD1306)

5 NPN Transistor or Relay

6 DC Water Pump

7 Buzzer

8 LED (Red & White)

9 Push Buttons

10 Resistors (for buttons)

11 Breadboard & Wires

12 Power Supply

13 WiFi.h – For ESP32 WiFi

14 PubSubClient – For MQTT communication

15 SimpleTimer – Lightweight interval timer

16 DHT sensor library by Adafruit

17 Adafruit Unified Sensor

18 Adafruit SSD1306

19 Adafruit GFX

Bil of component:

NO	Name of component	Quantity	Justification
1	ESP32	1	Acts as the main microcontroller with built-in WiFi, ideal for IoT applications. It handles all sensor reading, actuator control, OLED display, and MQTT communication.
2	Soil Moisture Sensor	1	Provides real-time soil moisture data. Essential for determining when to activate the water pump, ensuring optimal irrigation.
3	DHT11 Sensor	1	Measures ambient temperature and humidity. Helps to monitor environmental conditions that can affect soil moisture evaporation rate.
4	OLED Display (SSD1306)	1	Locally displays real-time data (temperature, humidity, soil status, emergency state).
5	DC Water Pump	1	Main actuator that controls irrigation. It responds to both automatic and manual commands.
6	Buzzer	1	Provides audible alerts in dry soil conditions, wet soil conditions, increasing system interactivity.
7	Red and White LEDs	4	Provide a visual indication of soil condition: red for dry another one is power supply, white for wet another one is pump system if pump work will light up.
8	STOP and RESET Buttons	2	Physical safety interface. STOP instantly disables the pump and buzzer; RESET restores normal operation. Ensures user control even if remote control fails.
9	Relay or NPN Transistor	1	Interface between ESP32 (low power) and high-power pump. Essential to safely control pump operation.
10	Buttery (3.7V)	3	Power supply
11	WiFi LED (GPIO2)		Indicates WiFi and ThingsBoard connection status for user awareness and debugging.
12	Arduino IDE	1	Using to coding for the system
13	ThingsBoard Cloud Platform	1	Provides a cloud-based dashboard for real-time data monitoring and control.

NO	Name of component	Quantity	Price (RM)
1	ESP32	1	35.00
2	Soil Moisture Sensor	1	0.90
3	DHT11 Sensor	1	10
4	OLED Display (SSD1306)	1	6.80
5	DC Water Pump	1	2.80
6	Buzzer	1	0.30
7	Red and White LEDs	4	1.40
8	STOP and RESET Buttons	2	5.00
9	Relay or NPN Transistor	1	2.00
10	breadboard	2	9.00
11	resistor	2	0.20
12	Buttery (3.7V)	3	2.71
		Total:	74.11

Conclusion

The Smart Soil Moisture Monitoring System effectively accomplished its objective of automating irrigation according to current environmental conditions. The system integrated essential sensors and actuators with the ESP32-WROOM-32, allowing it to accurately detect soil moisture, monitor temperature and humidity, and activate the mini water pump as needed to respond appropriately.

The incorporation of a 0.96" OLED display enhanced system engagement, enabling users to access live data easily. The addition of an LED and buzzer enhanced reliability with visual and audio notifications. Throughout the testing phase, the system reliably reacted to varying soil conditions and showcased the effectiveness of integrated IoT in precision farming.

In summary, this project emphasizes that cost-effective parts and basic logic can produce an effective and expandable smart agriculture system. It encourages water-saving practices, decreases physical work, and establishes the groundwork for more sophisticated IoT-driven farming systems. Upcoming enhancements might feature a mobile app interface, automated scheduling, or solar-powered functionality to improve system sustainability and user-friendliness.

Coding:

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <Ticker.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Adafruit_Sensor.h>
#include <DHT.h>
// WiFi + MQTT
char ssid[] = "";
char pass[] = "";
#define THINGSBOARD_SERVER "demo.thingsboard.io"
#define TOKEN              "RoSjfcHzFBSP2AUVQQOn"
WiFiClient espClient;
PubSubClient client(espClient);
Ticker readingTimer;
// OLED
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);
// DHT
#define DHTPIN   14
#define DHTTYPE  DHT11
DHT dht(DHTPIN, DHTTYPE);
// Pins
#define SENSOR_PIN    34
#define RED_LED       18
#define WHITE_LED     19
```

```
#define PUMP_PIN      25
#define BUZZER_PIN    26
#define BUTTON_STOP   32
#define BUTTON_RESET  33
#define WIFI_LED      2
// Moisture calibration
#define DRY_RAW       3500
#define WET_RAW       1200
// State
volatile bool pumpOn       = false;
volatile bool manualPump   = false;
volatile bool emergency    = false;
int moisturePct            = 0;
float temperature          = 0.0;
float humidity             = 0.0;
// WiFi
void connectWiFi() {
  WiFi.begin(ssid, pass);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nWiFi Connected!");
}
// MQTT
void connectMQTT() {
  while (!client.connected()) {
    Serial.print("Connecting to ThingsBoard...");
    if (client.connect("ESP32Client", TOKEN, NULL)) {
```

```
      Serial.println("connected");
      client.subscribe("v1/devices/me/rpc/request/+");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      delay(2000);
    }
  }
}
// RPC Callback
void callback(char* topic, byte* payload, unsigned int length) {
  String data;
  for (int i = 0; i < length; i++) {
    data += (char)payload[i];
  }
  String topicStr = String(topic);
  if (data.indexOf("method") > 0) {
    if (data.indexOf("setPump") > 0) {
      if (!emergency) {
        manualPump = data.indexOf("true") > 0;
        pumpOn = manualPump;
      }
    } else if (data.indexOf("emergencyStop") > 0) {
      emergency = data.indexOf("true") > 0;
      if (emergency) {
        pumpOn = false;
        manualPump = false;
        noTone(BUZZER_PIN);
      }
```

```
    }
  }
}
// Buttons
void checkButtons() {
  static bool lastStop = HIGH, lastReset = HIGH;
  bool nowStop = digitalRead(BUTTON_STOP);
  bool nowReset = digitalRead(BUTTON_RESET);
  if (lastStop == HIGH && nowStop == LOW) {
    emergency = true;
    pumpOn = false;
    manualPump = false;
    noTone(BUZZER_PIN);
    Serial.println(">> EMERGENCY STOP (Button)");
  }
  if (lastReset == HIGH && nowReset == LOW) {
    emergency = false;
    Serial.println(">> EMERGENCY RESET (Button)");
  }
  lastStop = nowStop;
  lastReset = nowReset;
}
// Buzzer beep
void wetBeep() {
  tone(BUZZER_PIN, 2000, 150);
  delay(200);
  tone(BUZZER_PIN, 2000, 150);
}
```

```
// Sensor + Logic + Display + MQTT
void updateReading() {
  checkButtons();
  // Moisture sensor
  int raw = analogRead(SENSOR_PIN);
  raw = constrain(raw, WET_RAW, DRY_RAW);
  moisturePct = map(raw, DRY_RAW, WET_RAW, 0, 100);
  // DHT
  humidity = dht.readHumidity();
  temperature = dht.readTemperature();
  // Pump Logic
  if (emergency) {
    pumpOn = false;
    noTone(BUZZER_PIN);
  } else {
    if (!manualPump) {
      if (!pumpOn && moisturePct < 30) pumpOn = true;
      else if (pumpOn && moisturePct >= 70) {
        pumpOn = false;
        wetBeep();
      }
    }
    if (moisturePct < 30) tone(BUZZER_PIN, 1000);
    else if (moisturePct < 70) noTone(BUZZER_PIN);
  }
  digitalWrite(PUMP_PIN, pumpOn ? HIGH : LOW);
  digitalWrite(RED_LED, (moisturePct < 30) ? HIGH : LOW);
  digitalWrite(WHITE_LED, (moisturePct >= 70) ? HIGH : LOW);
```

```
  // OLED
  display.clearDisplay();
  display.setCursor(0, 0);
  display.setTextSize(1);
  display.setTextColor(SSD1306_WHITE);
  display.printf("Moisture: %3d%%\n", moisturePct);
  display.printf("Temp:     %.1f C\n", temperature);
  display.printf("Humidity: %.1f %%\n", humidity);
  display.printf("Pump:     %s\n", pumpOn ? "ON" : "OFF");
  if (moisturePct < 30) display.println("Soil: DRY");
  else if (moisturePct >= 70) display.println("Soil: WET");
  else display.println("Soil: NORMAL");
  if (emergency) display.println("!! EMERGENCY !!");
  display.display();
  // MQTT Publish
  String payload = "{";
  payload += "\"moisture\":" + String(moisturePct) + ",";
  payload += "\"temperature\":" + String(temperature) + ",";
  payload += "\"humidity\":" + String(humidity) + ",";
  payload += "\"pumpOn\":" + String(pumpOn ? "true" : "false") + ",";
  payload += "\"emergency\":" + String(emergency ? "true" : "false");
  payload += "}";
  client.publish("v1/devices/me/telemetry", payload.c_str());
}
// Setup
void setup() {
  Serial.begin(115200);
  pinMode(RED_LED, OUTPUT);
  pinMode(WHITE_LED, OUTPUT);
```

```
  pinMode(PUMP_PIN, OUTPUT);
  pinMode(BUZZER_PIN, OUTPUT);
  pinMode(BUTTON_STOP, INPUT_PULLUP);
  pinMode(BUTTON_RESET, INPUT_PULLUP);
  pinMode(WIFI_LED, OUTPUT);
  dht.begin();
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println("OLED init failed");
    while (1);
  }
  display.clearDisplay();
  display.setCursor(0, 0);
  display.setTextSize(1);
  display.setTextColor(SSD1306_WHITE);
  display.println("Connecting to WiFi...");
  display.display();
  connectWiFi();
  client.setServer(THINGSBOARD_SERVER, 1883);
  client.setCallback(callback);
  connectMQTT();
  for (int i = 0; i < 3; i++) {
    digitalWrite(WIFI_LED, HIGH); delay(300);
    digitalWrite(WIFI_LED, LOW); delay(300);
  }
  readingTimer.attach(1.0, updateReading);  // Every 1 second
}
void loop() {
  if (!client.connected()) {
    connectMQTT();
```

```
  }
  client.loop();
}
```

Tang chin wen (1191201278)

I am Tang Chin Wen; I am the leader of Group 4. The project we are working on is a smart watering system. In this project, I am responsible for the soil moisture sensor and the code of the humidity sensor. I understand that the humidity of the air will affect the humidity of the soil. The lowest soil humidity of plants is (20-30%) and the highest soil humidity of plants is (70-80%). So I set the soil humidity code of our soil humidity sensor to (30-70%). If the soil humidity is lower than 30%, the red LED will light up as a reminder. If the soil humidity is higher than 70%, the white LED will light up as a reminder. I also set up two button systems in this project. The first is an emergency button system to prevent the water pump from continuing to discharge water when the soil humidity is higher than 70%. Currently, the emergency button is needed. After the pump is repaired, we can press the reset button to resume the operation of the pump.

Because of this project, I had the opportunity to use thingsboard. This was my first time using thingsboard and I was not very familiar with it, so I gave it to my teammate Thanesh to write the code to connect to thingsboard. Since it was not his first time using thingsboard, I learned from him how to use esp32 to connect to Wi-Fi or Bluetooth and then connect to thingsboard to check the moisture of our soil directly from the phone or computer. Although I did not write the code, I was also responsible for designing the dashboard because I wrote the codes for soil moisture sensor, humidity sensor, emergency button and reset, so I knew what widgets were needed in the dashboard.

In this project, I also learned how to be a good leader. Aside from the usual tasks like assigning works to the members and leading them during the progress, I also learned how to be patient in communicating with teammates. Even though it started unpleasantly, everyone was cooperative in doing the circuit all together. The most difficult part of the circuit is the water pump part because our water pump is 5 voltages but the supply voltage of our entire circuit is higher than 5V. The voltage supplied by our entire circuit is 12V, so we discussed using npn transistors to step down the voltage to protect all components.

THANESHWARAN A/L RAVICHANDRAN (1201101420)

In the Smart Soil Moisture Monitoring System project, my main task was to guarantee the effective deployment of the Wi-Fi communication system with the ESP32-WROOM-32 microcontroller and to establish the cloud-based monitoring interface via the ThingsBoard platform. This position was essential for facilitating the immediate transmission and display of soil moisture data gathered by our sensor system. Initially, I was responsible for coding the ESP32 to establish a connection to a Wi-Fi network. This task required creating and adjusting Arduino code that enabled the ESP32 to connect to our local Wi-Fi using designated SSID and password details.

Simultaneously with the microcontroller configuration, I focused on the ThingsBoard platform to display the gathered data. I established a new device on ThingsBoard and produced an access token to connect the ESP32 securely. After establishing communication, I created a dashboard featuring digital widgets like gauges and time-series graphs that refreshed in real time. These visual aids were created to show the soil moisture level in a clear and intuitive manner. I also set up fundamental alert functions like emergency stop to activate a warning whenever the soil moisture fell below a specified limit, mimicking how automated irrigation systems can operate based on real-time data

During this journey, I faced numerous technical difficulties, especially in ensuring the ESP32 reliably communicated with ThingsBoard. These experiences greatly improved my problem-solving abilities and highlighted the significance of thorough testing in embedded systems. his project provided me with hands-on experience in implementing agricultural solutions based on IoT technology. I discovered how microcontrollers with Wi-Fi capabilities function in a cloud-connected environment, and I gained knowledge of cloud platforms such as ThingsBoard, commonly used for data visualization in intelligent applications.

in summary, my involvement in creating the Wi-Fi and ThingsBoard elements of the Smart Soil Moisture Monitoring System was equally demanding and fulfilling. It enhanced my technical base in IoT systems and provided me with a greater appreciation for collaboration and system integration.

MAHENDRAN A/L SURESH (1191101755)

For this project, I helped put together a live alert-and-monitoring setup with an ESP32, a small buzzer, and a handy OLED screen. I wired the buzzer to GPIO 26 and wrote code for three main alerts: (1) a constant 1000-Hz tone when soil moisture drops under 30%, (2) two brief 2000-Hz beeps when the pump shuts off after the soil hits 70%, and (3) a noTone() command that cuts the sound during an emergency stop, whether I trigger it by hand or through a ThingsBoard RPC. At the same time, I set up an SSD1306 display on the same I2C bus (address 0x3C); it refreshes every second and shows the soil moisture level, temperature, humidity, pump status (ON/OFF), soil condition (DRY, NORMAL, WET), and a big red warning-!! EMERGENCY !!-whenever the buzzer goes into crisis mode.

While putting both systems to work, a few bumps cropped up along the way:

Synchronization Issues: I had to make sure the buzzer and OLED talked at the same time and didnt step on rapid sensor updates.

Display Flickering: At first, the screen flashed because I wiped the whole buffer every second; switching to partial redraws stopped the jumpiness.

Timing Conflicts: The blocking delay() calls made behavior erratic, so I rewrote the timing with millis() and kept everything responsive.

Tone Overlap: I also needed the buzzer to cut out the instant emergency mode fired, so I tracked state carefully and dropped noTone() everywhere it mattered.

I2C Communication Lag: Because the OLED runs over I2C, its updates added tiny delays that threw off the buzzer, so I lined the displays refresh with quieter moments.