SWSG

SMART WATERING SYSTEM

Prepared By

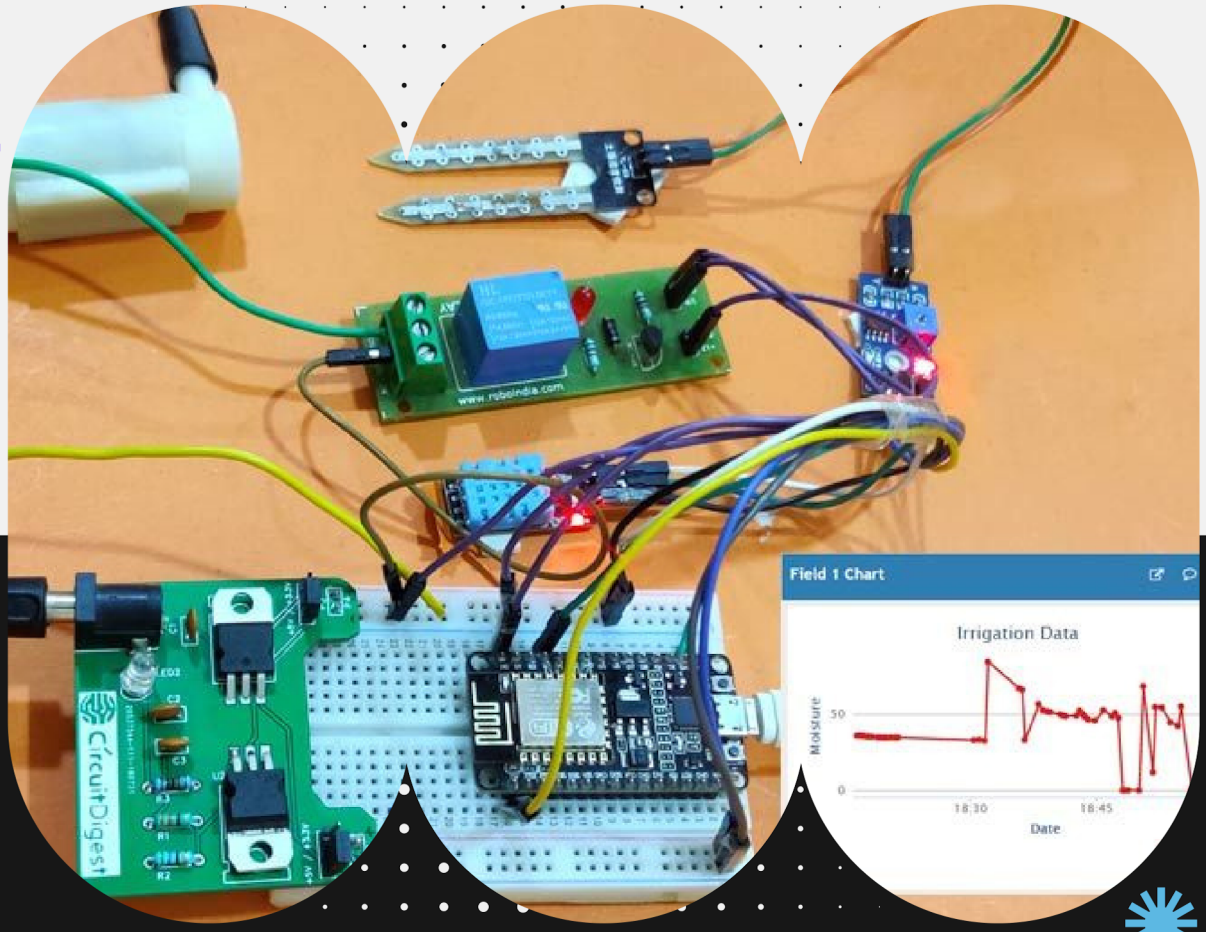
Muhammad Ahmar Shahid
Muhammad Taha Shahid

Supervised By

Sir Tehseen ul Hassan

github.com/ahmarshahid

github.com/tahashahid



Department of Computer Science

UNIVERSITY OF ENGINEERING AND TECHNOLOGY, LAHORE

TABLE OF CONTENTS

TABLE OF CONTENTS

ABSTRACT

The Smart Watering System for Gardens is a project designed to enhance the efficiency and convenience of garden irrigation by integrating Arduino Uno and ESP32 microcontrollers. The system employs a combination of sensors, actuators, and wireless communication to create an intelligent and automated watering solution.

Key components of the project include soil moisture sensors, temperature sensors, and a water pump controlled by the Arduino Uno. The ESP32 is utilized for its Wi-Fi capabilities, enabling remote monitoring and control via a dedicated mobile application or web interface

The soil moisture sensors continuously measure the moisture content in the soil, providing valuable data to the Arduino Uno. Using predefined thresholds, the system intelligently triggers the water pump to irrigate the garden when the soil moisture falls below the desired level

This Smart Watering System not only promotes sustainable water usage but also simplifies garden maintenance for users. By leveraging the capabilities of Arduino Uno and ESP32, the project offers an intelligent, efficient, and user-friendly solution for garden irrigation, contributing to the advancement of smart agriculture and home automation.

COMPONENTS

- **Arduino UNO**

Arduino Uno serves as the core controller in the smart watering system for gardens, managing sensors and actuators. Its ATmega328P processor interfaces with soil moisture sensors, temperature sensors, and controls a water pump. With a user-friendly programming environment like Arduino IDE, Uno enables the implementation of intelligent algorithms to optimize watering schedules based on real-time environmental data.

- **ESP32 module**

The ESP32 is a crucial component in the smart watering system for gardens, providing wireless connectivity and remote control capabilities. Equipped with Wi-Fi, the ESP32 enables real-time monitoring and adjustment of the watering system through a dedicated mobile app or web interface. Its dual-core processor and ample memory make it suitable for handling data processing and communication tasks efficiently.

- **Soil Sensor**

The soil moisture sensor is a vital component in the smart watering system for gardens, providing real-time feedback on the moisture content of the soil. This sensor employs probes to measure the electrical conductivity in the soil, correlating it with moisture levels. The data gathered by the sensor is crucial for determining when irrigation is necessary, ensuring optimal watering based on the specific needs of the plants

- **LED**

The LED in the smart watering system for gardens serves as the actuator responsible for delivering water to the plants. Controlled by the Arduino Uno, the pump is activated based on input from soil moisture sensors, ensuring that water is supplied only when needed. This automated process optimizes watering efficiency, preventing both overwatering and underwatering.

- **MQTT Broker/Client**

An MQTT broker receives messages from **ESP-32** and frowards them to the public using MQTT Dashboard Applications. Users subscribed to relevent topics on the MQTT broker receive real-time updates on LCD screen. Users can be instantly notified of changing status of the moisture level of the soil.

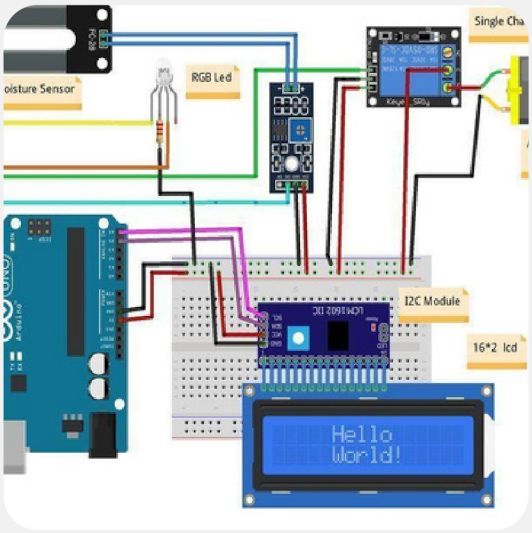
- **Google Firebase Realtime Database**

Google Firebase Realtime Database is employed in the smart watering system for gardens to store and manage real-time data generated by sensors. This cloud-based database allows seamless integration with the **ESP32** microcontroller, enabling remote access and monitoring of soil moisture levels and system status. By utilizing Firebase, users can receive instant updates through a mobile app or web interface, facilitating timely adjustments to watering schedules.

- **ThingSpeak Module**

ThingSpeak is a cloud-based Internet of Things (IoT) platform utilized in the smart watering system for gardens. It enables the ESP32 to upload and store sensor data, including soil moisture levels and temperature information. This data can be accessed in real-time through ThingSpeak's web interface, allowing users to monitor and analyze the soil’s conditions remotely.

METHODOLOGY



AVR Module

AVR Assembly Language is utilized for programming the ATMega328p microcontroller, Arduino UNO is used for this purpose.

This methodology involves low-level programming for efficient control of hardware components and direct intrection with the microcontroller.

UART

UART communication is used for establishing a serial communication link between the **Arduino** (ATMega329p) and **ESP-32**.

This allows data transfer between the two devices, enabling them to exchange Sensor Values and States of the Sensors.

C Language

C Programming language is employed for coding the **ESP-32,** a widely used **IoT** (Internet of Things) device, that has the feature such as **WiFi** Connection

ESP-32 is programmed to keep record of soil values. It is also programmed to act as a bridge between cloud and **Arduino UNO** communication.

MQTT

MQTT (Message Queuing Telementry Transport) is a lightweight messaging protocol used, facilitating seamless communication between the devices and the cloud.

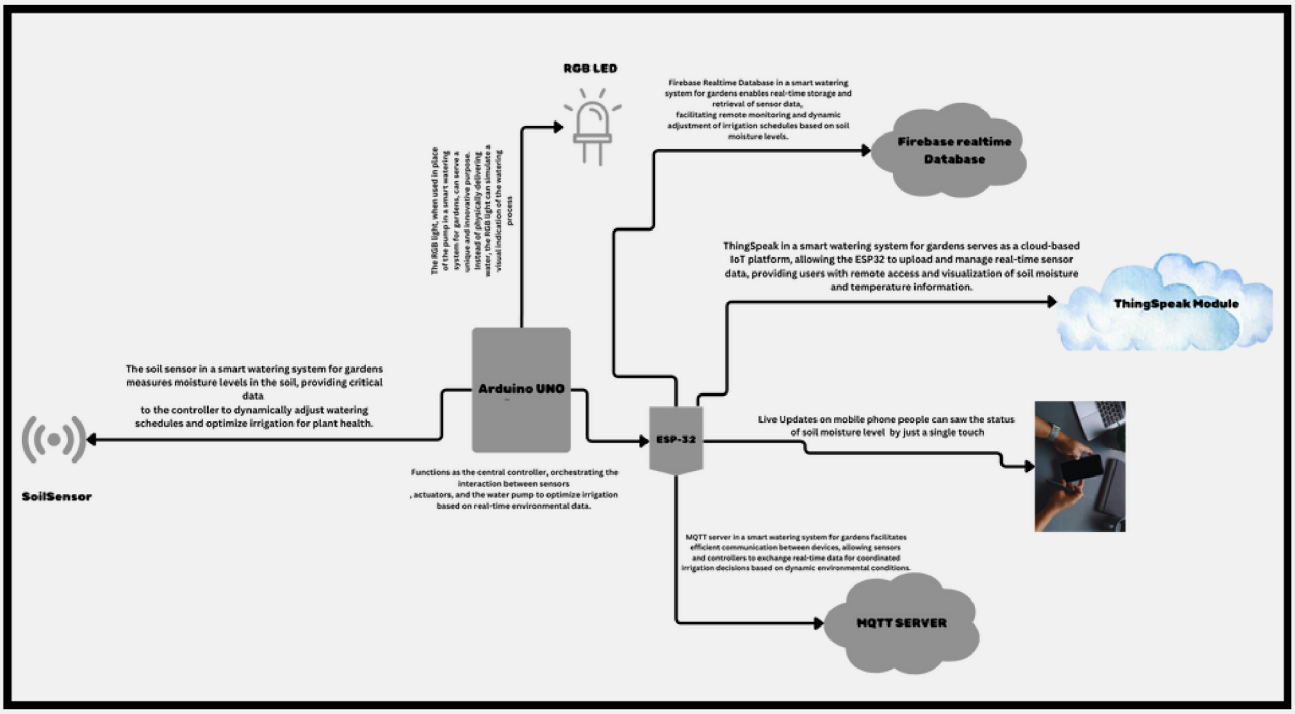
Firebase

Firebase is a backend-as-a-service platform utilized for real-time data storage, retrievel and synchronization.
It integrates in Smart Watering System for gardens for storing and managing data generated by the system, facilitating seamless communication between the devices and the cloud.

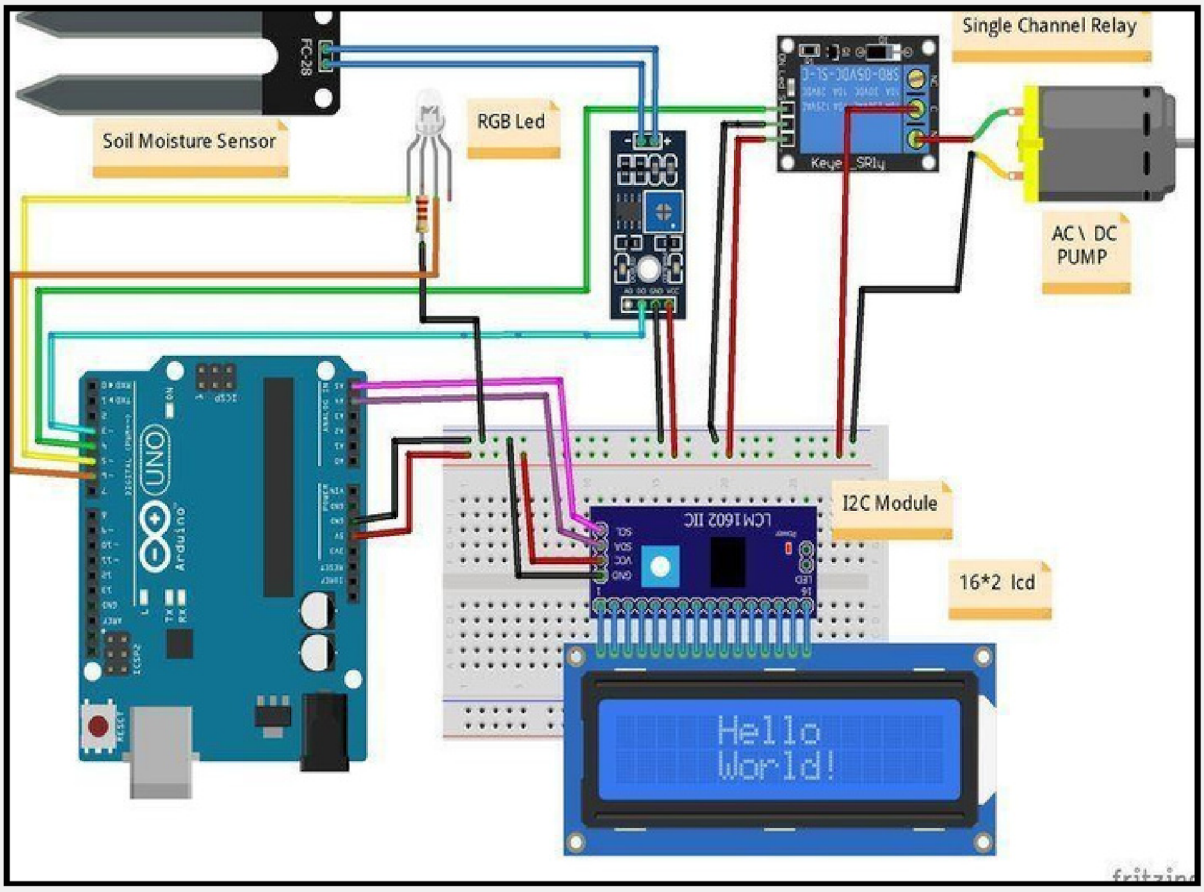
ThingSpeak Module

ThingSpeak facilitates the creation of customizable charts and graphs, aiding in the interpretation of data trends. Its integration enhances the system's scalability and provides a user-friendly means of managing and optimizing the smart irrigation setup.

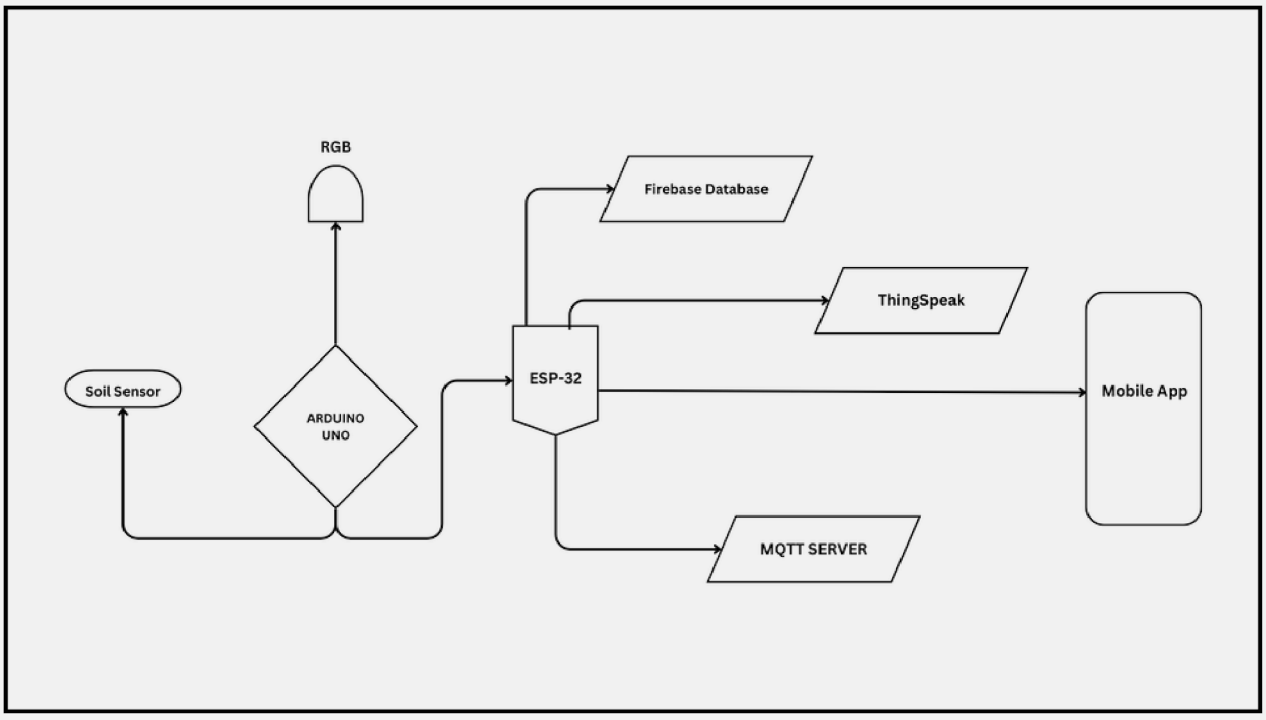
DATA FLOW DIAGRAM



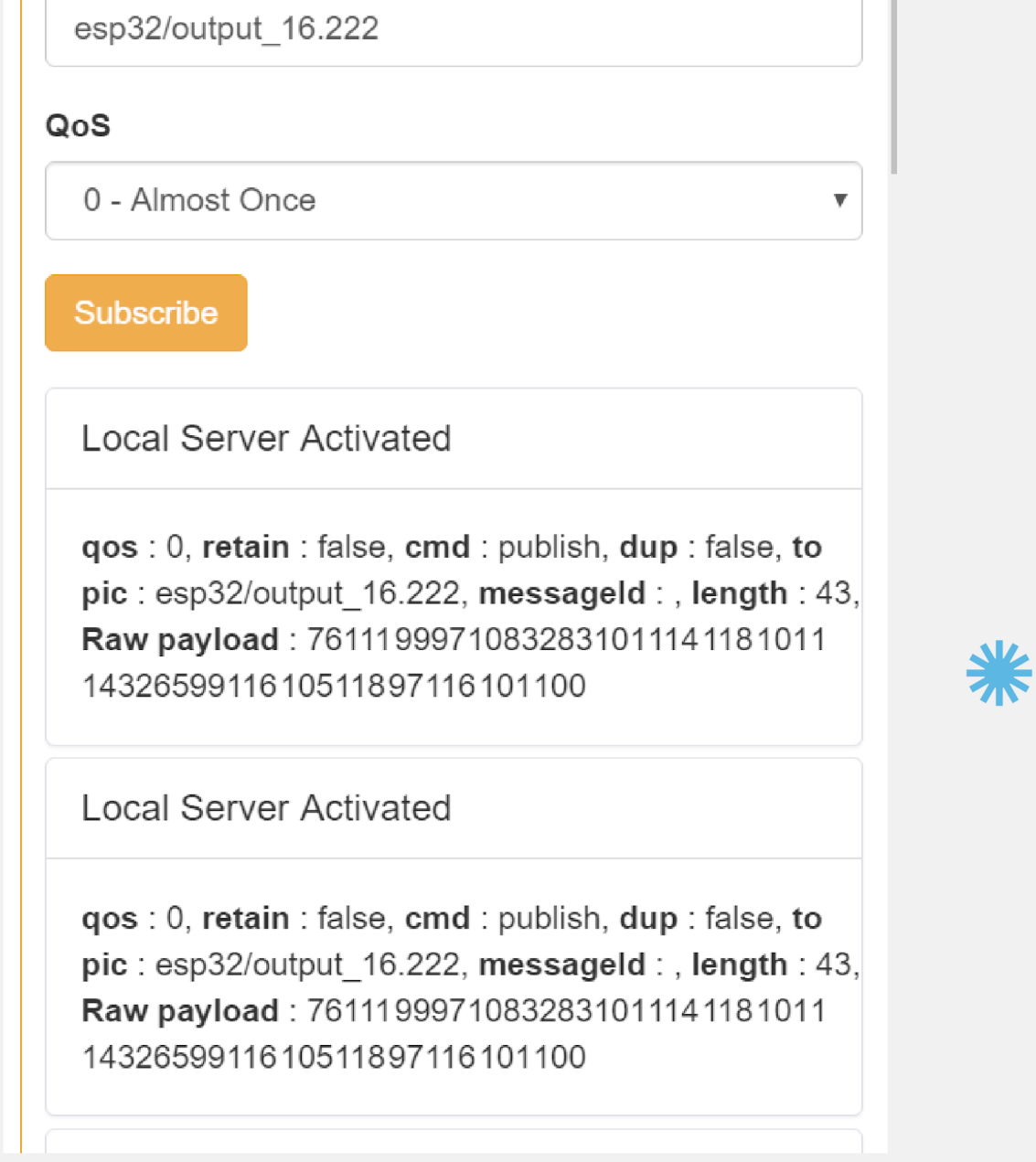
CIRCUIT DIAGRAM



FLOW DIAGRAM



MQTT SCREENSHOT



AVR CODE MODULE

```
.include "m328pdef.inc"
.include "delay.inc"
.include "UART.inc"
.include "LCD_1602.inc"

.def A = r16
.def AH = r17
.def LCD = r18
.def LCD_BAK = r19


.org 0x00
 LCD_init
 delay 1000
 LCD_send_a_command 0x01
 delay 1000
 LCD_backlight_ON

 ; 0: Controlled by Sensor
 ; 1: Controlled by App
 ; LCD[1] = Pump State
 ; 0: OFF
 ; 1: ON
 ; LCD[2] = Moisture Level
 ; 0: HIGH
 ; 1: LOW
 ldi LCD, 0b00000000 ; Init LCD Indicator
 ldi LCD_BAK, 0xFF
 ; P4 for LED
 sbi DDRB,PB4
 sbi PORTB,PB4
```

```
; ADC Configuration
 LDI A,0b11000111 ; [ADEN ADSC ADATE ADIF ADIE ADIE ADPS2 ADPS1
ADPS0]
 STS ADCSRA,A
 LDI A,0b01100000 ; [REFS1 REFS0 ADLAR – MUX3 MUX2 MUX1 MUX0]
 STS ADMUX,A ; Select ADC0 (PC0) pin
 SBI PORTC,PC0 ; Enable Pull-up Resistor
 ldi r16, 0x00
 ldi r17, 0x00
 ldi r20, 0x00

 Serial_begin ; initilize UART serial communication

loop:
 delay 1000
 call getSerialData
 call analogRead ; Read Soil Moisture Detector Sensor value in AH
 Serial_writeReg_ASCII AH ; sending the received value to UART
 Serial_writeNewLine
 ; LCD[0] is clear LCD in Sensor Mode
 SBRS LCD, 0
  call UPDATE_LCD_REG
 cp LCD, LCD_BAK
 brne DISPLAY_LCD
rjmp loop

 UPDATE_LCD_REG:
  cpi AH, 170
  brmi HIGH_MOISTURE
  rjmp LOW_MOISTURE
 END_UPDATE_LCD_REG:
  ret

 HIGH_MOISTURE:
  andi LCD, 0b11111011
  call PUMP_OFF
  rjmp END_UPDATE_LCD_REG

 LOW_MOISTURE:
  ori LCD, 0b00000100
  call PUMP_ON
  rjmp END_UPDATE_LCD_REG
```

```
PUMP_ON:
 ori LCD , 0b00000010
 ret

PUMP_OFF:
 andi LCD,0b11111101
 ret

DISPLAY_LCD:
 SBRS LCD, 0
  call UPDATE_LCD_REG
 mov LCD_BAK, LCD
 ; Clear LCD
 LCD_send_a_command 0x01
 ; LCD[0] is set LCD in Manual Mode
 SBRC LCD, 0
  call display_manual_mode
 ; LCD[0] is clear LCD in Sensor Mode
 SBRS LCD, 0
  call display_moisture_level
 ; Move to Next Line
 LCD_send_a_command 0xC0
 ; Display Pump
 call display_pump
 ; LCD[1] is set LED ON
 SBRC LCD, 1
  call LED_ON
 ; LCD[1] is clear LED OFF
 SBRS LCD, 1
  call LED_OFF
 rjmp loop
```

```
display_moisture_level:
 call display_moisture
 ; LCD[2] is set Moisture Level Low
 SBRC LCD, 2
  call display_low
 ; LCD[2] is clear LCD in Moisture Level High
 SBRS LCD, 2
  call display_high
 ret

LED_ON:
 LDI ZL, LOW (2 * on_string)
 LDI ZH, HIGH (2 * on_string)
 Serial_writeStr

 SBI PORTB, PB4 ; LED ON
 call display_on
 ret

LED_OFF:
 LDI ZL, LOW (2 * off_string)
 LDI ZH, HIGH (2 * off_string)
 Serial_writeStr

 CBI  PORTB, PB4 ; LED ON
 call display_off
 ret

analogRead:
 LDS A,ADCSRA ; Start Analog to Digital Conversion
 ORI A,(1<<ADSC)
 STS ADCSRA,A
 wait:
  LDS A,ADCSRA ; wait for conversion to complete
  sbrc A,ADSC
  rjmp wait
```

```
LDS A,ADCL ; Must Read ADCL before ADCH
 LDS AH,ADCH
 delay 100
 ret

getSerialData:
 Serial_read
 cpi r20, '1'
 breq MANUAL_ON
 cpi r20, '2'
 breq MANUAL_OFF
 cpi r20, '3'
 breq MANUAL_PUMP_ON
 cpi r20, '4'
 breq MANUAL_PUMP_OFF

 endGetSerialData:
 ret

MANUAL_ON:
 ori LCD, 0b00000001
 LDI ZL, LOW (2 * manual_string)
 LDI ZH, HIGH (2 * manual_string)
 Serial_writeStr
 rjmp endGetSerialData

MANUAL_OFF:
 andi LCD, 0b11111110
 LDI ZL, LOW (2 * sensor_string)
 LDI ZH, HIGH (2 * sensor_string)
 Serial_writeStr
 rjmp endGetSerialData

MANUAL_PUMP_ON:
 call PUMP_ON
 rjmp endGetSerialData

MANUAL_PUMP_OFF:
 call PUMP_OFF
 rjmp endGetSerialData
```

```
display_moisture:
 LDI ZL, LOW (2 * moisture_string)
 LDI ZH, HIGH (2 * moisture_string)
 LCD_send_a_string
 ret

display_low:
 LDI ZL, LOW (2 * low_string)
 LDI ZH, HIGH (2 * low_string)
 LCD_send_a_string
 ret

display_high:
 LDI ZL, LOW (2 * high_string)
 LDI ZH, HIGH (2 * high_string)
 LCD_send_a_string
 ret

display_pump:
 LDI ZL, LOW (2 * pump_string)
 LDI ZH, HIGH (2 * pump_string)
 LCD_send_a_string
 ret

display_on:
 LDI ZL, LOW (2 * on_string)
 LDI ZH, HIGH (2 * on_string)
 LCD_send_a_string
 ret

display_off:
 LDI ZL, LOW (2 * off_string)
 LDI ZH, HIGH (2 * off_string)
 LCD_send_a_string
 ret

display_manual_mode:
 LDI ZL, LOW (2 * manual_mode_string)
 LDI ZH, HIGH (2 * manual_mode_string)
 LCD_send_a_string
 ret
```

```
manual_mode_string: .db " MANUAL MODE ON",0
off_string: .db "OFF",0x0D,0x0A,0
on_string: .db "ON",0x0D,0x0A,0
pump_string: .db "   Pump: ",0x0D,0x0A,0
high_string: .db "HIGH",0x0D,0x0A,0
low_string: .db "LOW",0x0D,0x0A,0
moisture_string: .db " Moisture: ",0x0D,0x0A,0
manual_string: .db "Manual",0x0D,0x0A,0
sensor_string: .db "Sensor",0x0D,0x0A,0
```

IOT MODULE CODE

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <Firebase_ESP_Client.h>
#include "addons/TokenHelper.h"
#include "addons/RTDBHelper.h"


// # define the firebase realtime database API Key and DATABASE URL
#define API_KEY "AIzaSyBshho8Db3vcAsDUglSZyBkBssOy68RuMA"
#define DATABASE_URL "https://coal-lab-default-rtdb.europe-
west1.firebasedatabase.app/"

//#define mqttServer "broker.hivemq.com"
const char *ssid = "Homé";
const char *password = "90009000";
const char *mqttServer = "broker.hivemq.com";
const int mqttPort = 1883;
const char *mqttClientId = "COAL_PROJECT_ID_16.222";
const char *outputTopic = "esp32/output_16.222";
const char *inputTopic = "esp32/input_16.222";
static unsigned long lastMillis = 0;


FirebaseData fbdo;
FirebaseAuth auth;
FirebaseConfig config;
unsigned long sendDataPrevMillis = 0;
bool signupOK = false;

// #define the ThingSpeak Module
const char *thingSpeakApiKey = "VPRIHK24BN2YUILV";
const char *thingSpeakUrl = "https://api.thingspeak.com/update?
api_key=VPRIHK24BN2YUILV&field1=0";

WiFiClient espClient;
PubSubClient client(espClient);
```

```
void setup() {
  Serial.begin(115200);
  Serial2.begin(9600);
  WifiSetup();

  // Configure MQTT
  client.setServer(mqttServer, mqttPort);
  client.setCallback(callBack);
  connectToMQTT();

   /* Assign the api key (required) */
  config.api_key = API_KEY;

  /* Assign the RTDB URL (required) */
  config.database_url = DATABASE_URL;

  /* Sign up */
  if (Firebase.signUp(&config, &auth, "", "")){
    Serial.println("ok");
    signupOK = true;
  }
  else{
    Serial.printf("%s\n", config.signer.signupError.message.c_str());
  }

  config.token_status_callback = tokenStatusCallback;

  Firebase.begin(&config, &auth);
  Firebase.reconnectWiFi(true);
}

void WifiSetup(){
  // Connect to Wi-Fi
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
```

```
 Serial.println("");
  Serial.print("WiFi connected with IP address: ");
  Serial.println(WiFi.localIP());
}

void callBack(char* inputTopic, byte* message, unsigned int length) {
  Serial.print("Message arrived on topic: ");
  Serial.print(inputTopic);
  Serial.print(". Message : ");
  String messageTemp;
  char tempPressed;

  for (int i = 0; i < length; i++) {
    messageTemp += (char)message[i];
  }
  Serial.println(messageTemp);
  Serial.print("Message Sent to UART : ");
}


void loop()
{
  if (WiFi.status() != WL_CONNECTED)
  {
    WifiSetup();
  }
  if (Firebase.ready() && signupOK && (millis() - sendDataPrevMillis >
4000 || sendDataPrevMillis == 0)){
    sendDataPrevMillis = millis();

// Now use buttonChar in your Firebase.RTDB.setInt call
    if (Firebase.RTDB.setInt(&fbdo, "data/sensor", 12))
    {
      Serial.print("PASSED   to ");
      Serial.println("PATH: " + fbdo.dataPath());
      Serial.println("TYPE: " + fbdo.dataType());
    }
    else {
      Serial.println("FAILED");
      Serial.println("REASON: " + fbdo.errorReason());
    }
  }
```

```
if(Serial2.available()> 0)
  {
     //String receivedChar = Serial2.readString();
     //Serial.print("Message Received through UART  :   ");
     //Serial.println(receivedChar);
  }

  // Handle MQTT events
  if (!client.connected()) {
    connectToMQTT();
  }
  client.loop();

  // Message Publishing on app with delay of 5s
  if (millis() - lastMillis > 5000) {


    const char* message = "Local Server Activated";
    publishMessage(message);
    lastMillis = millis();
  }

  // Data Sending to FireBase
}


void connectToMQTT() {
  while (!client.connected()) {
    Serial.println("Connecting to MQTT...");
    if (client.connect(mqttClientId)) {
      Serial.println("Connected to MQTT");
      client.subscribe(inputTopic);
    }
    else {
      Serial.print("Failed with state   :  ");
      Serial.println(client.state());
      delay(2000);
    }
  }
}

}
```

```
void publishMessage(const char* message) {
  if (client.connected()) {
    client.publish(outputTopic, message);
    Serial.print("Message Published :  ");
    Serial.println(message);
  }
```

LINKS