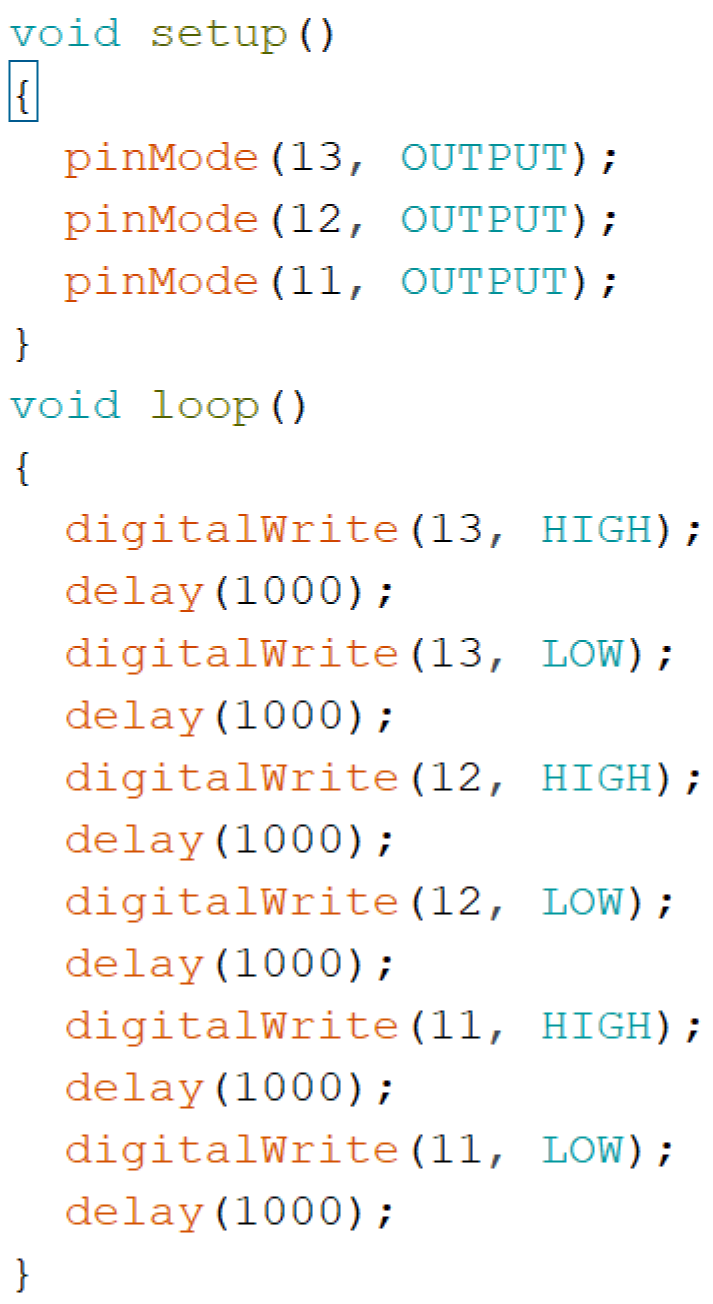
Assignment - 1

Title: Sequential LED Blinking using Arduino

Objective: The objective of this project is to demonstrate sequential LED blinking using an Arduino Uno, with two LEDs and a single resistor , with 1 sec delay.Two LEDs are connected to separate GPIO pins of the ESP32.

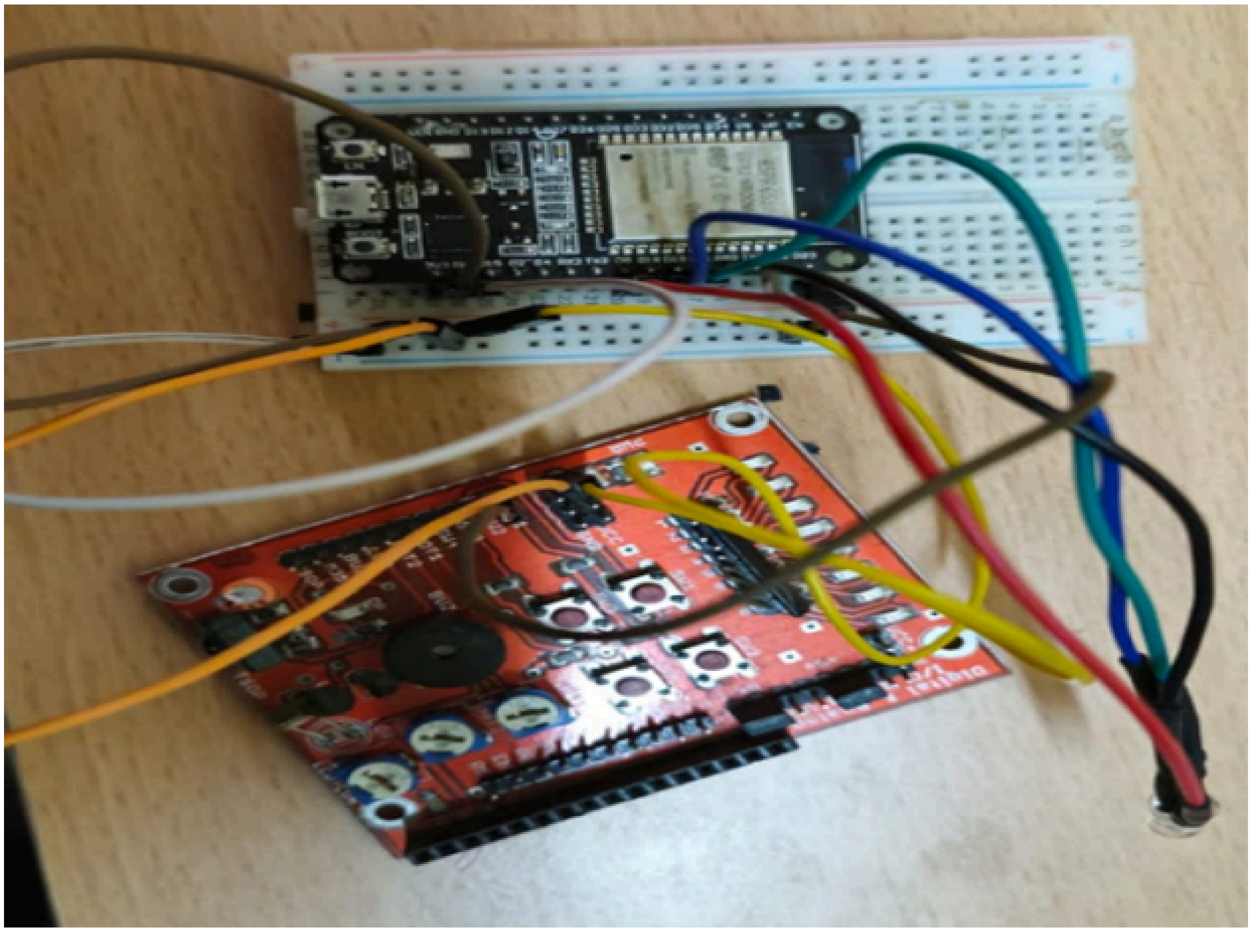
Code:



```
void setup()
{
  pinMode(13, OUTPUT);
  pinMode(12, OUTPUT);
  pinMode(11, OUTPUT);
}
void loop()
{
  digitalWrite(13, HIGH);
  delay(1000);
  digitalWrite(13, LOW);
  delay(1000);
  digitalWrite(12, HIGH);
  delay(1000);
  digitalWrite(12, LOW);
  delay(1000);
  digitalWrite(11, HIGH);
  delay(1000);
  digitalWrite(11, LOW);
  delay(1000);
}
```

Components Required: Arduino Uno, Universal Board, ESP32, 2 LEDs ,1 Resistor (220Ω) , Jumper wires

Circuit Diagram:



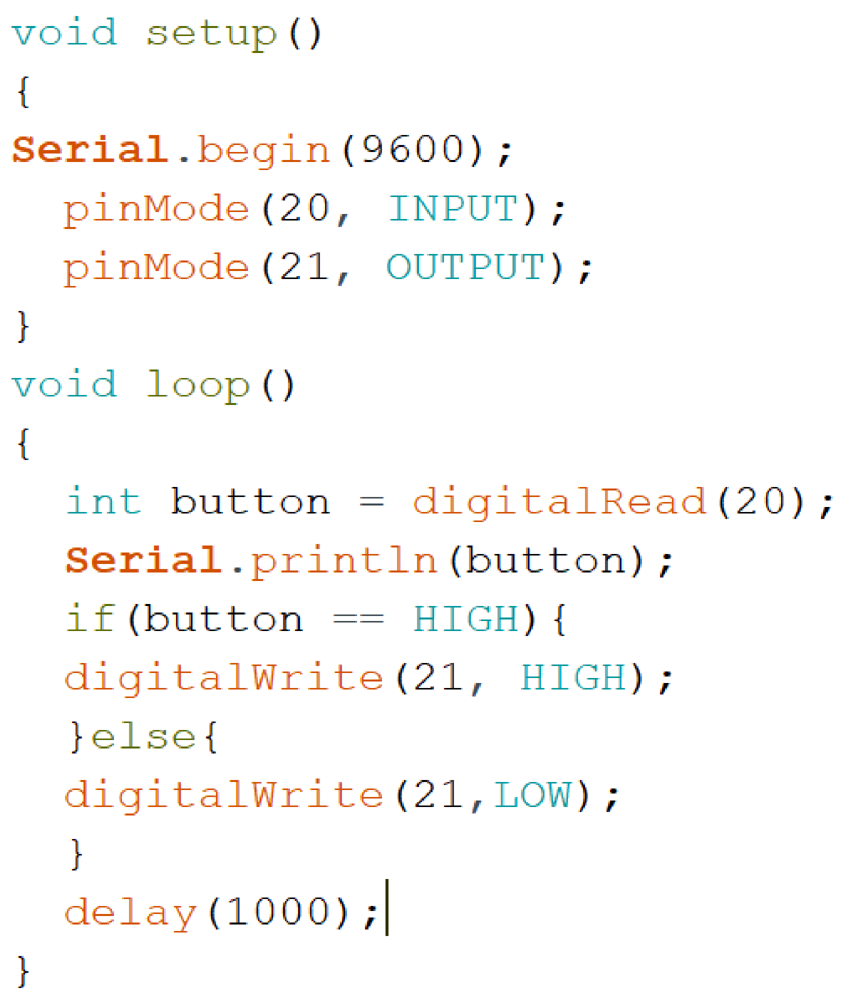
Output:

When the ESP32 runs the program, the two LEDs blink alternately—when one is ON, the other is OFF. This creates a continuous flashing effect, making it look like the LEDs are switching places. The delay between the transitions ensures that the blinking is smooth and clearly visible.

Assignment - 2

Title: ESP32 Button-Controlled LED System

Objective: The goal of this project is to use an ESP32 to control an LED with a push button. When the button is pressed, the LED turns on; when the button is released, the LED turns off. This project demonstrates digital input and output control using a microcontroller.

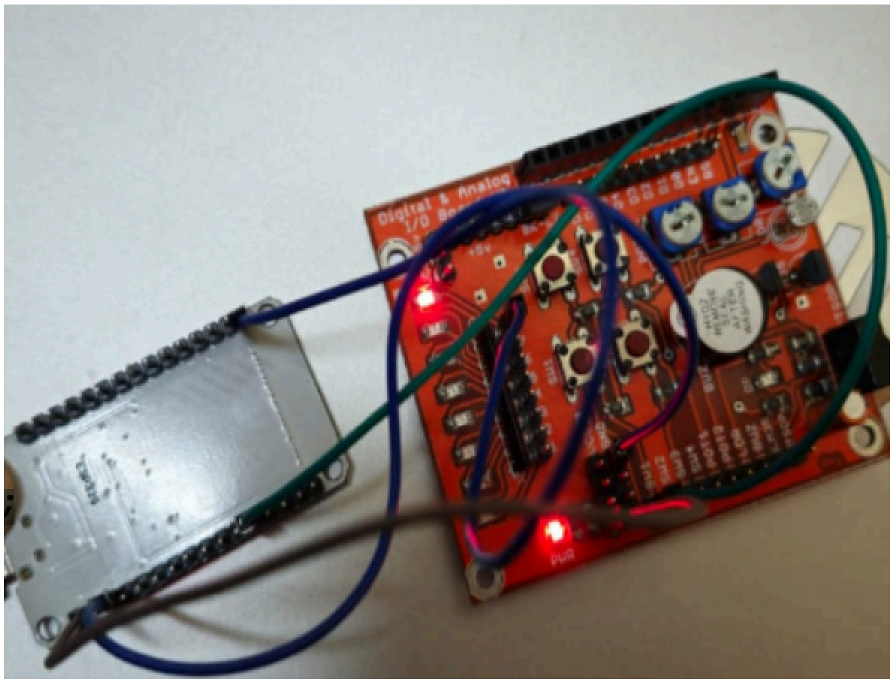


```
void setup()
{
Serial.begin(9600);
  pinMode(20, INPUT);
  pinMode(21, OUTPUT);
}
void loop()
{
  int button = digitalRead(20);
  Serial.println(button);
  if(button == HIGH){
  digitalWrite(21, HIGH);
  }else{
  digitalWrite(21,LOW);
  }
  delay(1000);
}
```

The push button acts as a sensor (digital input) by detecting whether it is pressed or released and sending the corresponding signal to the ESP32 microcontroller. When the button is pressed, it sends a HIGH (1) signal, and when released, it sends a LOW (0) signal. The LED, on the other hand, functions as an actuator (digital output) by responding to the signal received from the ESP32. If the button sends a HIGH signal, the ESP32 activates the LED, turning it ON. Conversely, when the button sends a LOW signal, the ESP32 deactivates the LED, turning it OFF. This setup demonstrates the interaction between a sensor (push button) and an actuator (LED) in a basic microcontroller-based system.

Output :When the ESP32 is powered on, it continuously reads the button's state from GPIO 20. If the button is pressed (input is HIGH), the LED connected to GPIO 21 turns on. If the button is not pressed (input is LOW), the LED turns off. The button's state (0 or 1) is displayed on the Serial Monitor, and the system updates every 1 second.

```
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
```



Assignment - 3

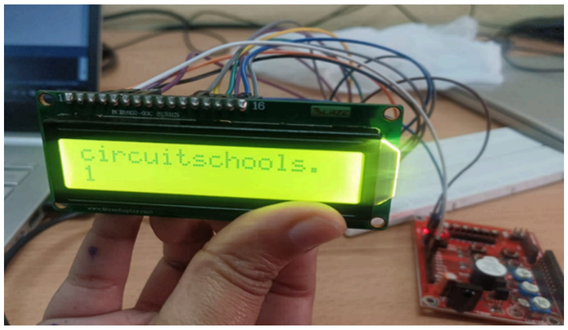
Title: ESP32-Based LCD Display with Timer

Objective: This project uses an ESP32 and an LCD to display a message and a timer. It helps in understanding how to interface an LCD with a microcontroller.

```
// include the library code:
#include <LiquidCrystal.h>
LiquidCrystal lcd(19, 23, 18, 17, 16, 15);
void setup() {
// set up the LCD's number of columns and rows:
lcd.begin(16, 2);
// Print a message to the LCD.
lcd.print("Task Done.");
}
void loop() {
lcd.setCursor(0, 1);
// print the number of seconds since reset:
lcd.print(millis() / 1000);
}
```

The LCD screen acts as an actuator, which means it takes an electrical signal from the ESP32 and performs an action—displaying messages and the timer count. Actuators are essential in IoT systems as they convert processed data into meaningful actions, such as visual feedback, motion, or sound.Interfacing a 16x2 LCD with ESP32 allows displaying text, sensor readings, or system messages. The LCD is typically connected using an I2C module (SDA & SCL) or parallel connections to ESP32 GPIO pins. The ESP32 sends data to the LCD, which then displays characters on its 16-column, 2-row screen. This setup is useful for real-time monitoring and embedded applications.

Output: When the ESP32 starts, the LCD screen displays:"Task Done." on the first row.The second row shows the number of seconds since the ESP32 was powered on.The timer continuously updates, increasing every second.



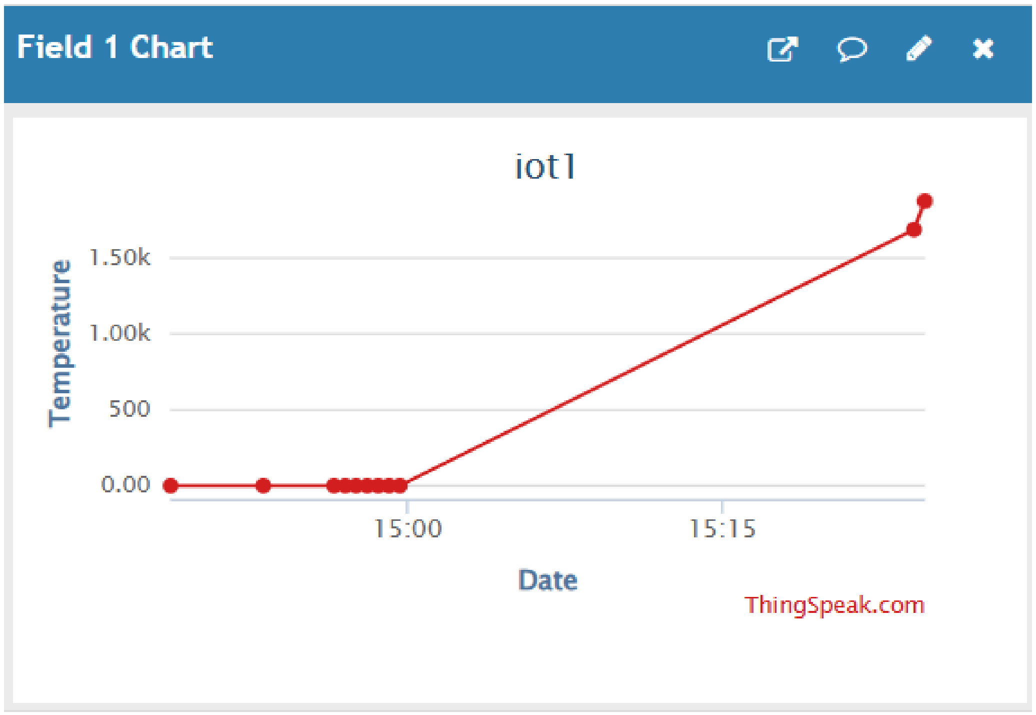
Assignment - 4

Title: ESP32-Based Temperature Monitoring and Data Logging on ThingSpeak

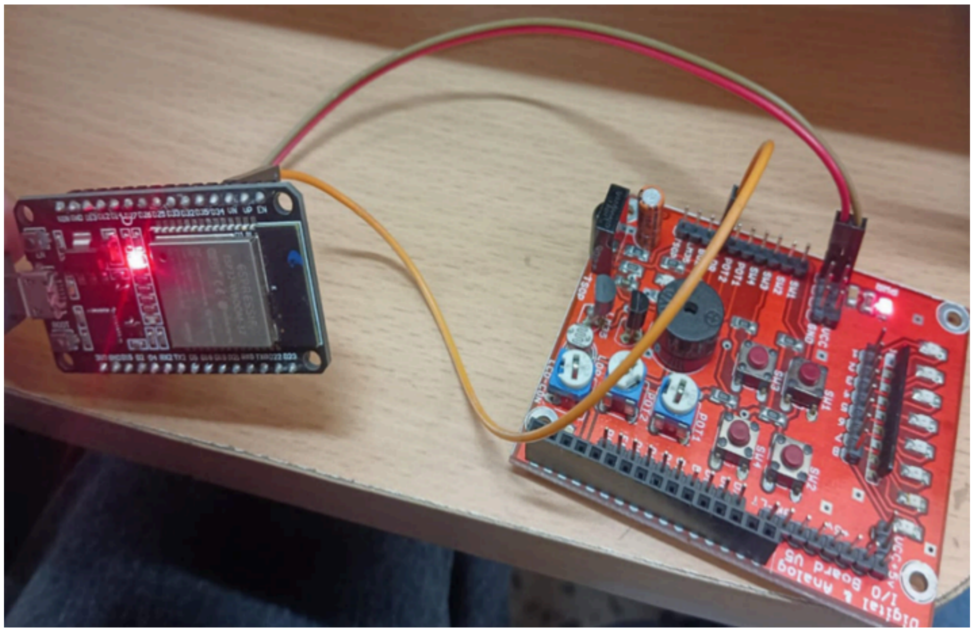
Objective: This project uses an ESP32 to read temperature data from an analog sensor and send it to ThingSpeak over WiFi. It demonstrates IoT-based data logging, enabling real-time monitoring of environmental conditions.

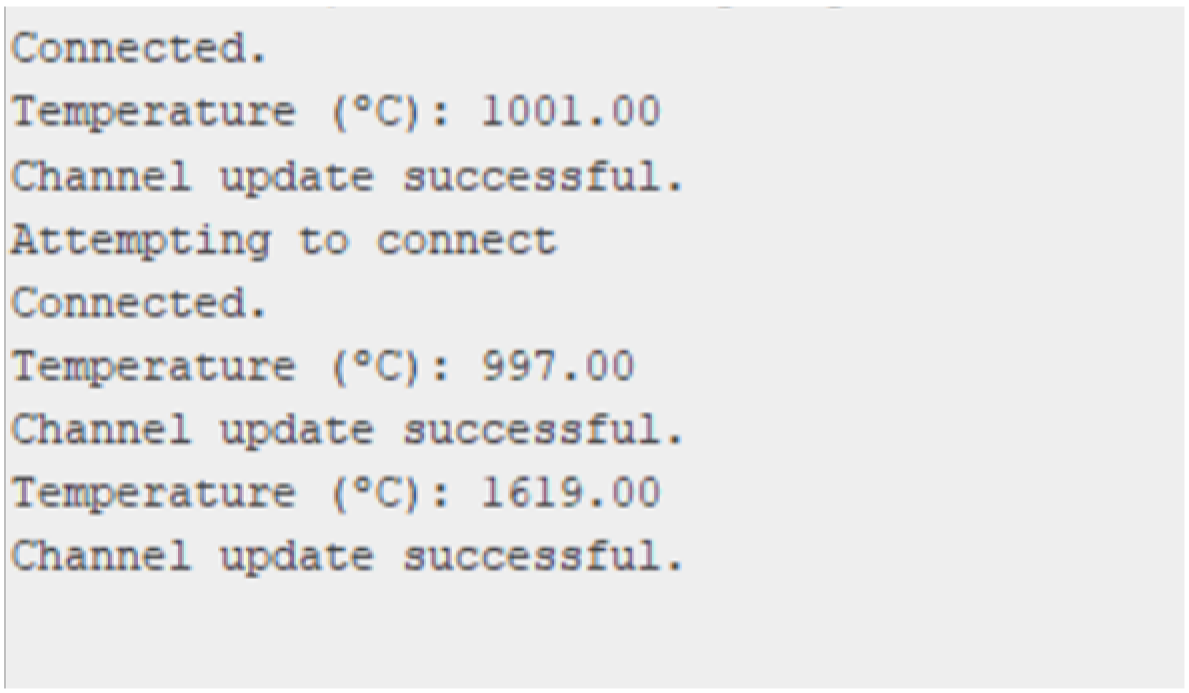
Output:

The ESP32 connects to WiFi and establishes a link to ThingSpeak.It reads temperature data from analog pin 34 (simulating a sensor).The temperature value is printed to the Serial Monitor for real-time viewing.The ESP32 uploads this temperature data to ThingSpeak every 30 seconds



```
#include <WiFi.h>
#include "ThingSpeak.h"

const char* ssid = "Galaxy M31sE1E0";   // your network SSID (name)
const char* password = "12345678";   // your network password

WiFiClient  client;

unsigned long myChannelNumber = 1;
const char * myWriteAPIKey = "OHJF0TMH41DWBURX";

// Timer variables
unsigned long lastTime = 0;
unsigned long timerDelay = 30000;

// Variable to hold temperature readings
float temperatureC;
//uncomment if you want to get temperature in Fahrenheit
//float temperatureF;

// Create a sensor object
// Adafruit_BME280 bme; //BME280 connect to ESP32 I2C (GPIO 21 = SDA, GPIO 22 = SCL)


// void initBME(){
//   if (!bme.begin(0x76)) {
//     Serial.println("Could not find a valid BME280 sensor, check wiring!");
//     while (1);
//   }
// }

void setup() {
  Serial.begin(115200);  //Initialize serial
  // initBME();

  WiFi.mode(WIFI_STA);

  ThingSpeak.begin(client);  // Initialize ThingSpeak
}

void loop() {
  if ((millis() - lastTime) > timerDelay) {

    // Connect or reconnect to WiFi
    if(WiFi.status() != WL_CONNECTED){
      Serial.print("Attempting to connect");
      while(WiFi.status() != WL_CONNECTED){
        WiFi.begin(ssid, password);
        delay(5000);
      }
      Serial.println("\nConnected.");
    }

    // Get a new temperature reading
    temperatureC = analogRead(34);
    Serial.print("Temperature (°C): ");
    Serial.println(temperatureC);

    //uncomment if you want to get temperature in Fahrenheit
    /*temperatureF = 1.8 * bme.readTemperature() + 32;
    Serial.print("Temperature (°F): ");
    Serial.println(temperatureF);*/

    // Write to ThingSpeak. There are up to 8 fields in a channel, allowing you to store up to 8 different
    // pieces of information in a channel.  Here, we write to field 1.
    int x = ThingSpeak.writeField(myChannelNumber, 1, temperatureC, myWriteAPIKey);
    //uncomment if you want to get temperature in Fahrenheit
    //int x = ThingSpeak.writeField(myChannelNumber, temperatureC, 1, myWriteAPIKey);

    if(x == 200){
        Serial.println("Channel update successful.");
      }
      else{
        Serial.println("Problem updating channel. HTTP error code " + String(x));
      }
      lastTime = millis();
  }
}
```




```
Connected.
Temperature (°C): 1001.00
Channel update successful.
Attempting to connect
Connected.
Temperature (°C): 997.00
Channel update successful.
Temperature (°C): 1619.00
Channel update successful.
```

LED blinking :

The system continuously monitors temperature readings and controls LEDs based on predefined thresholds. If the temperature exceeds 30°C, an alert message is displayed, and an LED blinks to indicate the threshold breach. Otherwise, the LEDs respond accordingly to different temperature ranges.

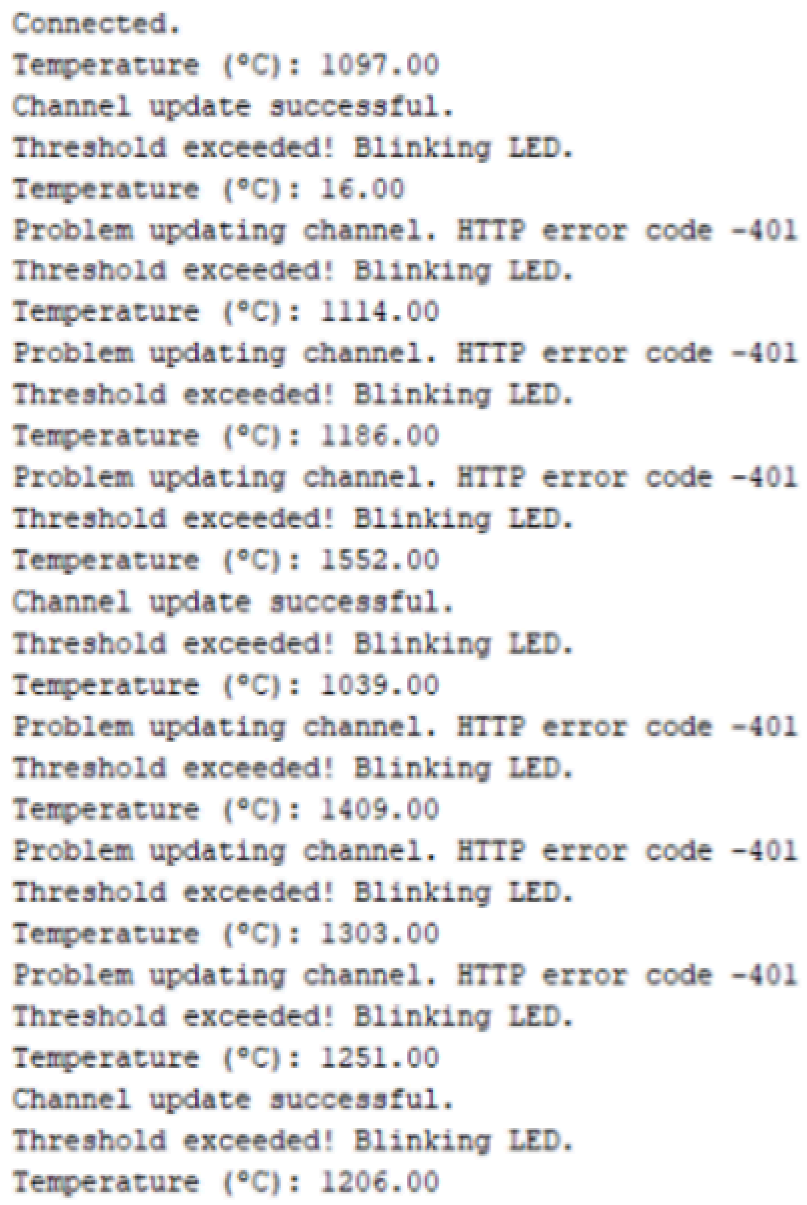
Code:

```
#include <WiFi.h>

#include <HTTPClient.h>



#define LED1 4  // GPIO for LED 1

#define LED2 5  // GPIO for LED 2



const char* ssid = "Galaxy M31sE1E0";

const char* password = "12345678";

const char* apiKey = "CRVSYEYVN1CLX3UM";
const char* server = "http://api.thingspeak.com";
const int channelID = 2824552;  // Replace with your ThingSpeak channel ID
const int fieldNumber = 1;      // Field number for temperature


void setup() {
   Serial.begin(115200);
   pinMode(LED1, OUTPUT);
   pinMode(LED2, OUTPUT);
   WiFi.begin(ssid, password);
   Serial.println("Connected.");


   while (WiFi.status() != WL_CONNECTED) {
      delay(1000);
      Serial.print(".");
   }


   Serial.println("\nConnected to Wi-Fi");
}
void loop() {
   if (WiFi.status() == WL_CONNECTED) {
      HTTPClient http;
      String url = String(server) + "/channels/" + String(channelID) + "/fields/" +
                  String(fieldNumber) + "/last.json?api_key=" + apiKey;
      http.begin(url);
```

```
int httpResponseCode = http.GET();


if (httpResponseCode > 0) {
  String response = http.getString();
  float temperature = extractTemperature(response);


  Serial.print("(°C): ");
  Serial.println(temperature);


  // Control LEDs based on temperature
  if (temperature > 30.0) {
    Serial.println("Threshold exceeded! Blinking LED.");
    digitalWrite(LED1, HIGH);
    digitalWrite(LED2, LOW);
    blinkLED(LED1, 3);  // Blink LED1
  } else if (temperature <= 30.0 && temperature >= 20.0) {
    digitalWrite(LED1, LOW);
    digitalWrite(LED2, HIGH);
  } else {
    digitalWrite(LED1, LOW);
    digitalWrite(LED2, LOW);
  }


  // Try updating ThingSpeak
```

```
            if (!updateThingSpeak(temperature)) {

                Serial.println("Problem updating channel. HTTP error code —401");

            }

        } else {

            Serial.println("Problem updating channel. HTTP error code —" +
String(httpResponseCode));

        }



        http.end();

    } else {

        Serial.println("Wi-Fi disconnected, reconnecting...");

        WiFi.reconnect();

        delay(1000);

    }



    delay(5000);  // Wait before the next request

}



float extractTemperature(String json) {

    int index = json.indexOf("\"field" + String(fieldNumber) + "\":\"");



    if (index != -1) {

        int start = index + String("\"field" + String(fieldNumber) + "\":\"").length();

        int end = json.indexOf("\"", start);

        return json.substring(start, end).toFloat();
```

```
  }

  return -1;  // Return -1 if parsing fails
}
bool updateThingSpeak(float temperature) {
  HTTPClient http;
  String url = String(server) + "/update?api_key=" + apiKey + "&field" + String(fieldNumber) +
"=" + String(temperature);

  http.begin(url);
  int httpResponseCode = http.GET();
  http.end();

  if (httpResponseCode == 200) {
    Serial.println("Channel update successful.");
    return true;
  }
  return false;
}
void blinkLED(int pin, int times) {
  for (int i = 0; i < times; i++) {
    digitalWrite(pin, HIGH);
    delay(500);
    digitalWrite(pin, LOW);
    delay(500);
  }
}
```



```
Connected.
Temperature (°C): 1097.00
Channel update successful.
Threshold exceeded! Blinking LED.
Temperature (°C): 16.00
Problem updating channel. HTTP error code -401
Threshold exceeded! Blinking LED.
Temperature (°C): 1114.00
Problem updating channel. HTTP error code -401
Threshold exceeded! Blinking LED.
Temperature (°C): 1186.00
Problem updating channel. HTTP error code -401
Threshold exceeded! Blinking LED.
Temperature (°C): 1552.00
Channel update successful.
Threshold exceeded! Blinking LED.
Temperature (°C): 1039.00
Problem updating channel. HTTP error code -401
Threshold exceeded! Blinking LED.
Temperature (°C): 1409.00
Problem updating channel. HTTP error code -401
Threshold exceeded! Blinking LED.
Temperature (°C): 1303.00
Problem updating channel. HTTP error code -401
Threshold exceeded! Blinking LED.
Temperature (°C): 1251.00
Channel update successful.
Threshold exceeded! Blinking LED.
Temperature (°C): 1206.00
```