

Assignment - 1

Title: Sequential LED Blinking using Arduino

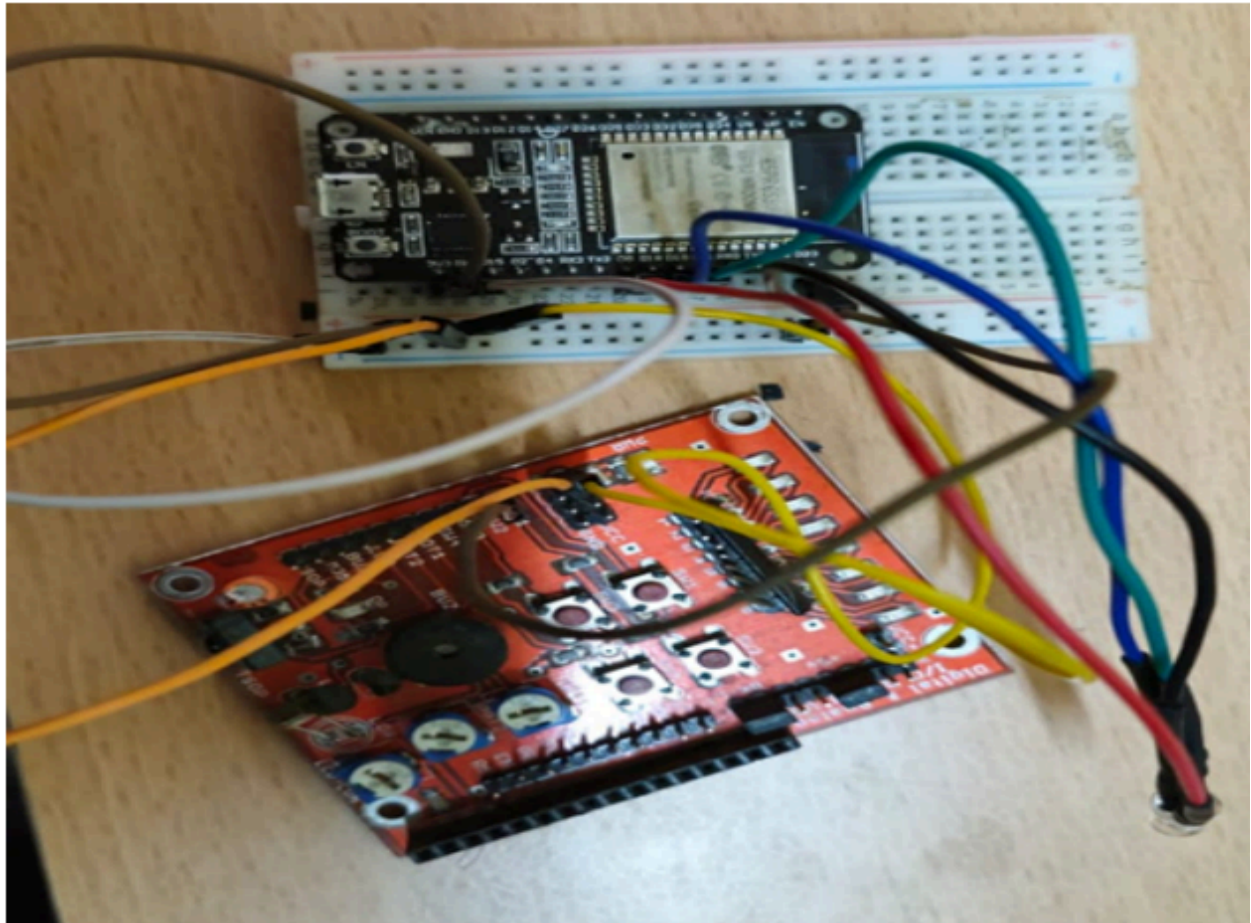
Objective: The objective of this project is to demonstrate sequential LED blinking using an Arduino Uno, with two LEDs and a single resistor , with 1 sec delay.Two LEDs are connected to separate GPIO pins of the ESP32.

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 (220 Ω) , 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

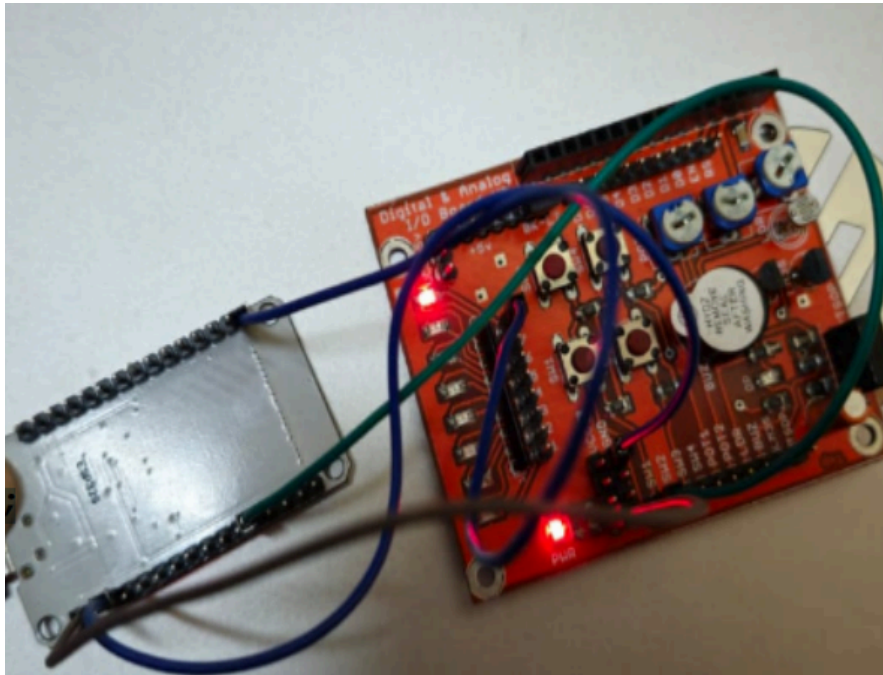
Objective: The goal of this project is to use an ESP32 to control an LED with a push button. When the button is pressed, the LED turns on; when the button is released, the LED turns off. This project demonstrates digital input and output control using a microcontroller.

```
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
1
1
```



Assignment - 3

Title: ESP32-Based LCD Display with Timer

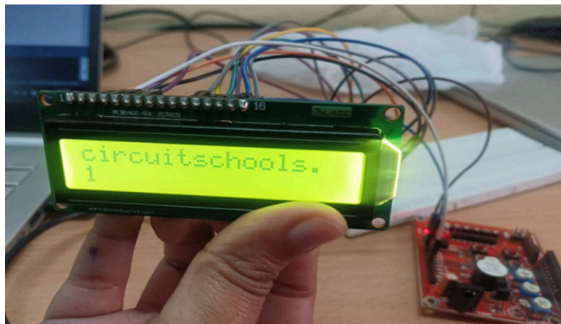
Objective: This project uses an ESP32 and an LCD to display a message and a timer. It helps in understanding how to interface an LCD with a microcontroller.

```
// 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.

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.



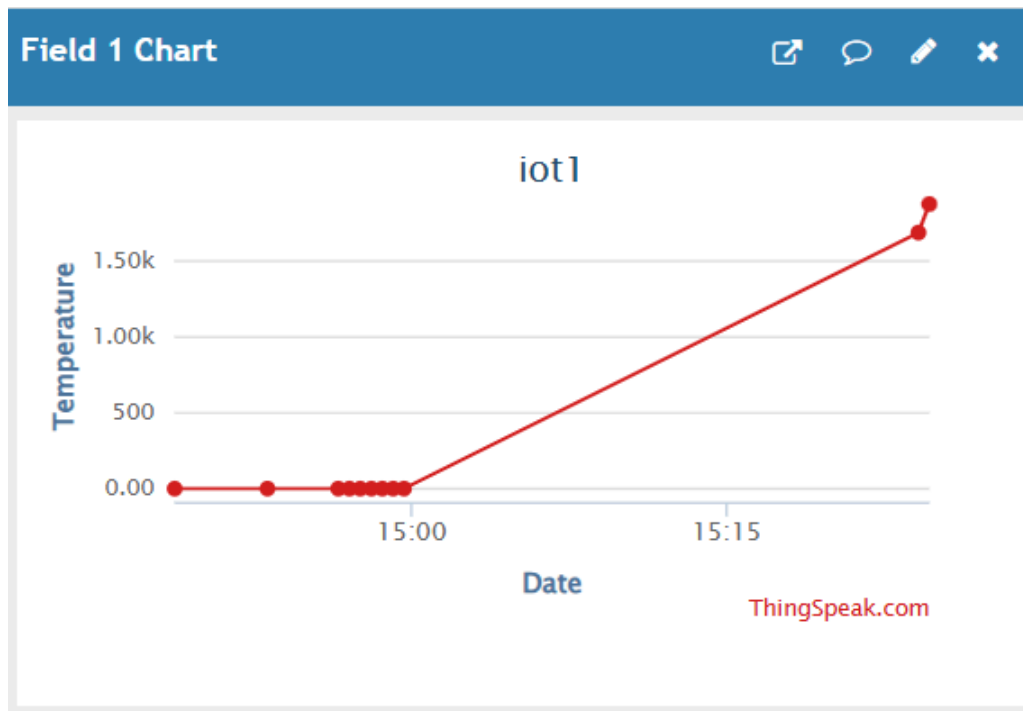
Assignment - 4

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 M3lsE1E0"; // 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();
}
}

```

