Arin Jain
0801CS221034

LAB Assignment IOT

Experiment 1: Blinking RGB LED with Delay using ESP32

Aim: To control an RGB LED and make it blink in various colors using an ESP32 microcontroller.

Objective:

1. Understand how to interface an RGB LED with an ESP32.
2. Learn to control different LED colors using GPIO pins.
3. Implement timing delays to create a blinking effect.

Hardware/IDE/Software used:

1. ESP32 Development Board
2. Universal Board
3. Jumper Wires
4. USB Cable to power the ESP32
5. Arduino IDE (with ESP32 board support installed).

Outcomes:

1. Ability to program an ESP32 to control an LED.
2. Knowledge of GPIO pin configuration for output.
3. Understanding of timing functions in embedded programming.

Principle: The ESP32 controls the RGB LED by switching its individual red, green, and blue components on and off using digital output signals.

Code:- void setup()

{

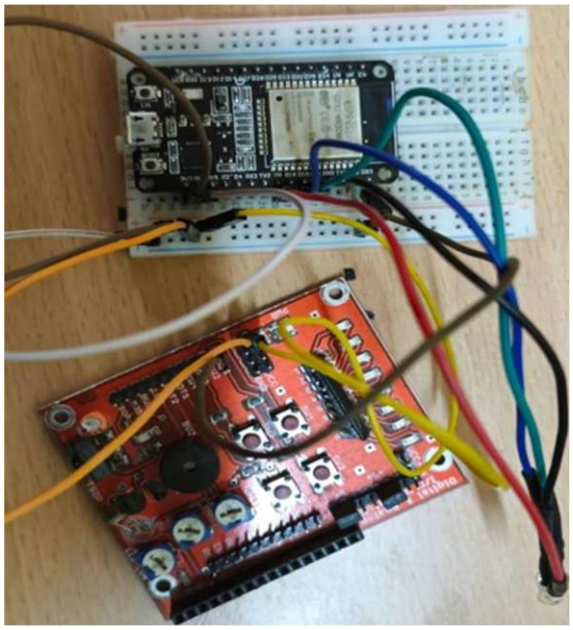
pinMode(13, OUTPUT);

pinMode(12, OUTPUT);

pinMode(11, OUTPUT);

}

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);
}
```

Experiment 2: Push Button Controlled LED with ESP32

Aim: To control an LED using a push button with the ESP32.

Objective:

1. Learn how to interface a push button with an ESP32.
2. Implement digital input reading for button presses.
3. Control an LED based on user input.

Hardware/IDE/Software used:

1. ESP32 Development Board
2. Universal Board
3. Jumper Wires

Arin Jain
0801CS221034

4. USB Cable to power the ESP32
5. Breadboard
6. Arduino IDE (with ESP32 board support installed)

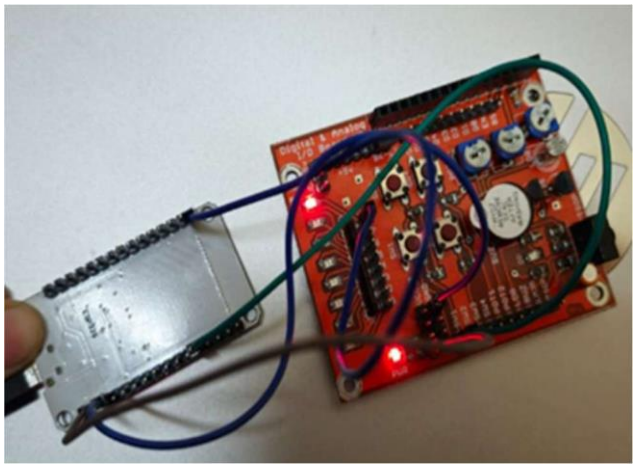
Outcomes:

1. Understanding of input and output configurations in ESP32.
2. Ability to read digital inputs and trigger events.
3. Implementation of real-time user interaction with hardware components.

Principle: A push button acts as a digital input device that toggles the LED state based on user interaction, utilizing GPIO input and output logic.

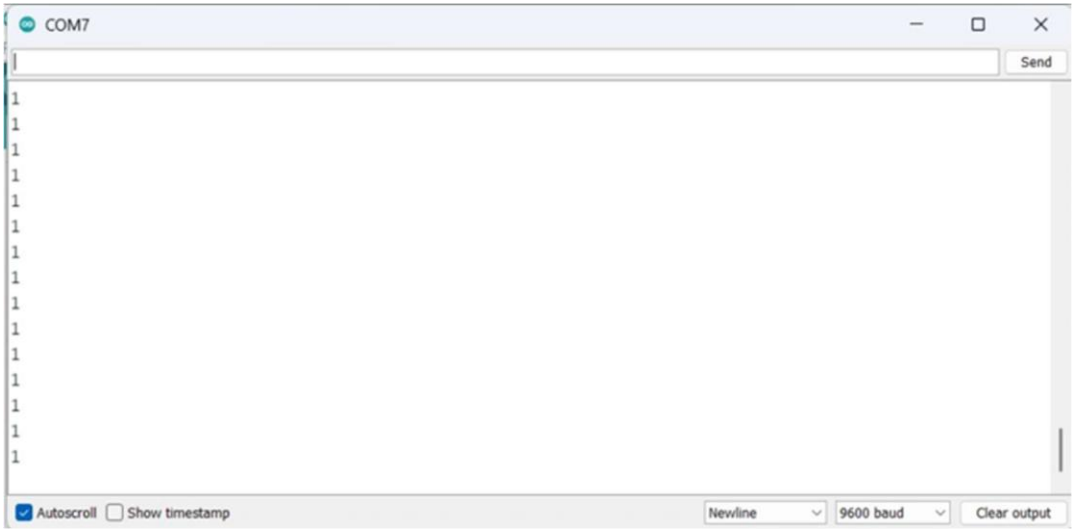
Code:

```
void setup()
{
Serial.begin(9600);
pinMode(35, INPUT);
pinMode(23, OUTPUT);
}
void loop()
{
int button = digitalRead(35);
Serial.println(button);
if(button == HIGH){
digitalWrite(23, HIGH);
}else{
digitalWrite(23,LOW);
}
delay(1000);
}
```



Arin Jain
0801CS221034

Output:



Experiment 3: Light Dependent Resistor (LDR) with ESP32

Aim: To measure light intensity using an LDR connected to an ESP32.

Objective:

1. Understand the working principle of an LDR.
2. Learn how to interface an LDR sensor with an ESP32.
3. Use analog input readings to control outputs based on light intensity.

Hardware/IDE/Software used:

1. ESP32 Development Board
2. Universal Board
3. Jumper Wires
4. USB Cable to power the ESP32
5. Breadboard
6. Arduino IDE (with ESP32 board support installed)

Libraries Used:

1. #include<stdio.h>

Outcomes:

Arin Jain
0801CS221034

1. Ability to read analog signals from an LDR.
2. Understanding of threshold-based decision making in embedded systems.
3. Implementation of automation based on environmental conditions.

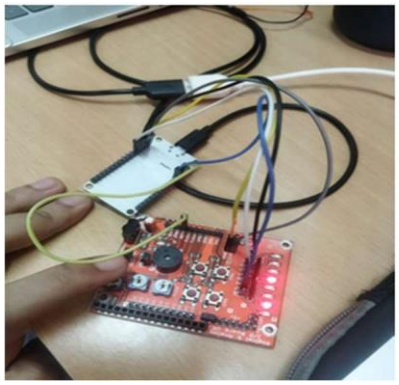
Principle: An LDR changes resistance based on light intensity, which is converted to an analog voltage read by the ESP32 for processing and decision-making.

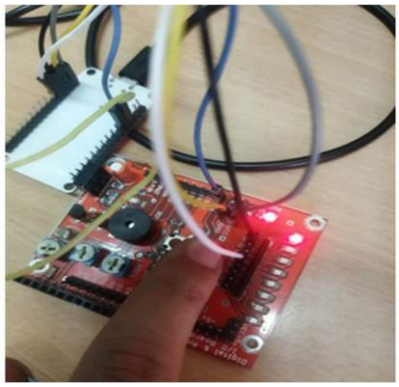
Code:

```
#include <stdio.h>
void setup() {
  pinMode(14, INPUT);
  pinMode(34, OUTPUT);
  pinMode(33, OUTPUT);
  pinMode(35, OUTPUT);
}
void loop() {
  Serial.println(analogRead(14));
  delay(700);
  if (analogRead(14) <= 100) {
    digitalWrite(33, HIGH);
    digitalWrite(34, HIGH);
    digitalWrite(35, HIGH);
  }
  else if (analogRead(14) < 1000 && analogRead(14) > 100) {
    digitalWrite(33, HIGH);
    digitalWrite(34, HIGH);
    digitalWrite(35, LOW);
```

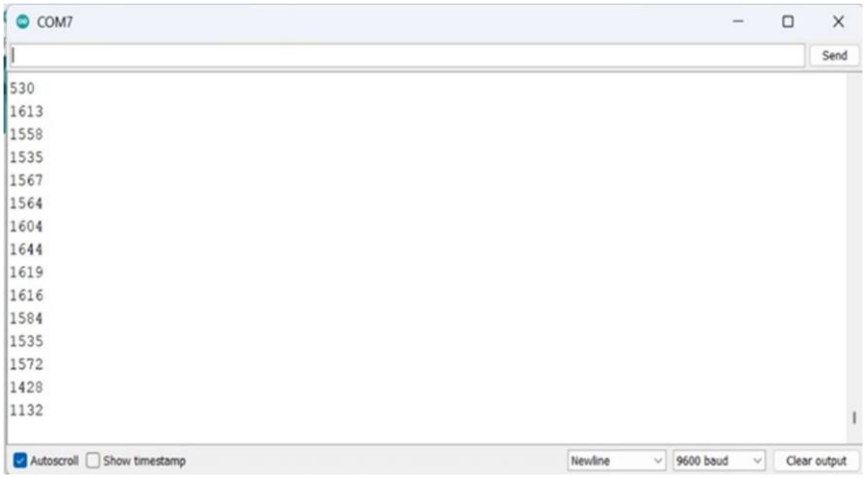
Arin Jain
0801CS221034

```
    }
    else if (analogRead(14) < 2000 && analogRead(14) > 1000) {
      digitalWrite(33, HIGH);
      digitalWrite(34, LOW);
      digitalWrite(35, LOW);
    }
    else {
      digitalWrite(33, LOW);
      digitalWrite(34, LOW);
      digitalWrite(35, LOW);
    }
}
```





Output:



Arin Jain
0801CS221034

Experiment 4: Interfacing an LCD with ESP32

Aim: To interface a 16x2 LCD with the ESP32 and display text.
Objective:

1. Understand how an LCD communicates with ESP32.
2. Learn to display and manipulate text on an LCD.
3. Implement dynamic text updates on an LCD screen.

Hardware/IDE/Software used:

1. ESP32 Development Board
2. Universal Board
3. Jumper Wires
4. USB Cable to power the ESP32
5. Breadboard
6. Arduino IDE (with ESP32 board support installed)
7. 16x2 LCD

Libraries Used:

1. #include<stdio.h>
2. #include <LiquidCrystal.h>

Outcomes:

1. Ability to connect and configure an LCD with ESP32.
2. Knowledge of data transmission using parallel/serial communication.
3. Display of meaningful text messages for user interaction.

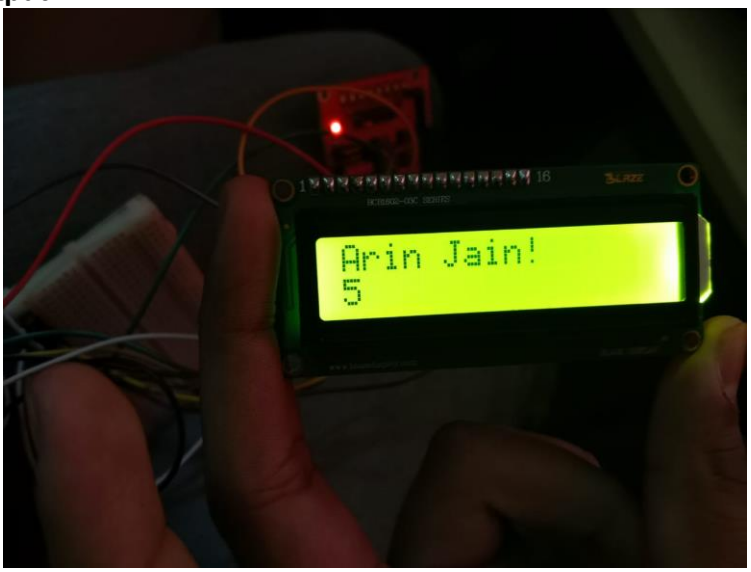
Principle: The ESP32 communicates with the LCD using digital output signals to display characters and messages dynamically.

Code:

```
#include <LiquidCrystal.h>

const int rs = 12, en = 11, d4 = 5, d5 = 4, d6 = 3, d7 = 2;

LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

void setup() {

// set up the LCD's number of columns and rows:
```

```
lcd.begin(16, 2);

// Print a message to the LCD.

lcd.print("Circuit schools");

}

void loop() {

// Turn off the blinking cursor:

lcd.noBlink();

delay(3000);

// Turn on the blinking cursor:

lcd.blink();

delay(3000);

}
```

Output:



Arin Jain
0801CS221034

Experiment 5: ThingSpeak Data Communication

Aim: To send and receive sensor data from ThingSpeak using ESP32.
Objective:

1. Learn how to interface ESP32 with the ThingSpeak IoT platform.
2. Understand Wi-Fi communication for data transmission.
3. Retrieve sensor data from the cloud and process it.

Hardware/IDE/Software used:

1. ThingSpeak Platform
2. Arduino IDE.

Libraries Used:

1. #include<stdio.h>
2. #include <WiFi.h>
3. #include "ThingSpeak.h"

Outcomes:

1. Ability to connect ESP32 to the internet via Wi-Fi.
2. Successful transmission and reception of data on ThingSpeak.
3. Real-time monitoring and analysis of sensor values.

Principle: The ESP32 connects to a Wi-Fi network and communicates with the ThingSpeak server using HTTP requests, enabling cloud-based data monitoring.

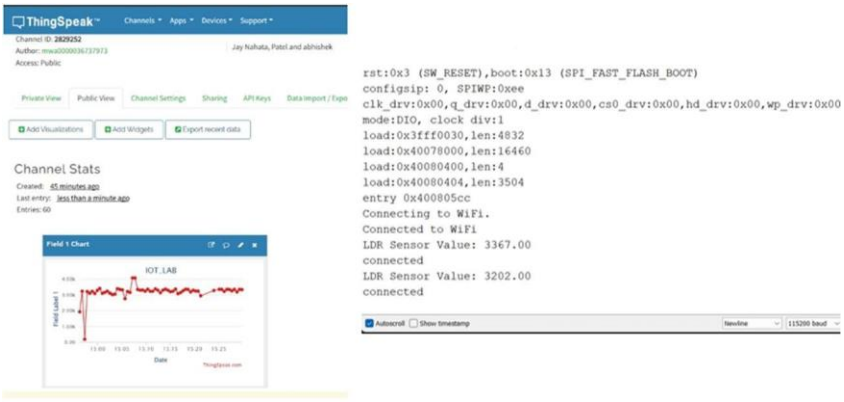
Code:

```
#include <WiFi.h>

#include "ThingSpeak.h"


const char* ssid = "MotoAKV";      // Your network SSID

const char* password = "123456789"; // Your network password


WiFiClient client;
```

Arin Jain
0801CS221034

```
unsigned long myChannelNumber = 2829252;  // Replace with your ThingSpeak Channel Number
const char * myReadAPIKey = "JHGKCR0ETDLB9DQR"; // Replace with your Read API Key


void setup() {
  Serial.begin(115200);  // Start Serial Monitor

  // Connect to WiFi
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);

  Serial.print("Connecting to WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(500);
  }
  Serial.println("\nConnected to WiFi");

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

void loop() {
  // Read LDR sensor value from ThingSpeak Field 1
  float ldrValue = ThingSpeak.readFloatField(myChannelNumber, 1, myReadAPIKey);

  // Display the received LDR value
```

```
  Serial.print("LDR Sensor Value: ");
  Serial.println(ldrValue);
  int statusCode = ThingSpeak.getLastReadStatus();
  if(statusCode == 200){
  Serial.println("connected");
  }
  else{
    Serial.println("Problem reading channel. HTTP error code " + String(statusCode));
  }

  delay(30000);  // Wait for 30 seconds before next read
}
```



Experiment 6: Bluetooth Serial Communication and Servo Motor Control with ESP32

Aim: To control a servo motor using Bluetooth communication with an ESP32.

Objective:

1. Learn to establish Bluetooth communication with ESP32.
2. Understand how to control a servo motor using received Bluetooth commands.

Arin Jain
0801CS221034

3. Implement angle-based servo movement via user input.

Hardware/IDE/Software used:

1. ESP32 Development Board
2. Universal Board
3. Jumper Wires
4. USB Cable to power the ESP32
5. Breadboard
6. Arduino IDE (with ESP32 board support installed)
7. Servo Motor
8. Bluetooth-enabled Mobile Device (Smartphone)

Libraries Used:

1. #include <BluetoothSerial.h>
2. #include <ESP32Servo.h>

Outcomes:

1. Ability to receive and process Bluetooth commands on ESP32.
2. Understanding of servo motor operation and angle control.
3. Real-time user interaction for remote-controlled movement.

Principle: The ESP32 receives Bluetooth data from a mobile device, interprets the received commands, and adjusts the servo motor angle accordingly.

Code:

```
#include <BluetoothSerial.h>

#include <ESP32Servo.h>


BluetoothSerial SerialBT; // Initialize Bluetooth Serial

Servo myServo; // Create a Servo object

const int servoPin = 13; // Define the GPIO pin for the servo

void setup() {

  Serial.begin(115200);

  SerialBT.begin("ana"); // Start Bluetooth with name "ESP32_Servo"
```

Arin Jain
0801CS221034

```
    myServo.attach(servoPin); // Attach the servo motor to pin 13
    myServo.write(90); // Set servo to initial position (90 degrees)
    Serial.println("Bluetooth Started! Waiting for input...");
}
void loop() {
    if (SerialBT.available()) { // Check if data is received from Bluetooth
        String data = SerialBT.readStringUntil('\n'); // Read the incoming string
        data.trim(); // Remove any extra spaces or newlines

        if (data.length() == 0) return; // Ignore empty data

        if (isDigit(data[0]) || (data[0] == '-' && isDigit(data[1]))) { // Ensure it's a valid number
            int angle = data.toInt(); // Convert received string to an integer

            if (angle >= 0 && angle <= 180) { // Check if it's a valid servo angle
                myServo.write(angle); // Move servo to the specified angle
                Serial.print("Servo moved to: ");
                Serial.println(angle);
            } else {
                Serial.println("Invalid angle! Enter a value between 0-180.");
            }}
    }
    delay(50);
}
```

Output: