

IoT & Automation Lab. Record

Lab#1

Blinking the InBuilt LED

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

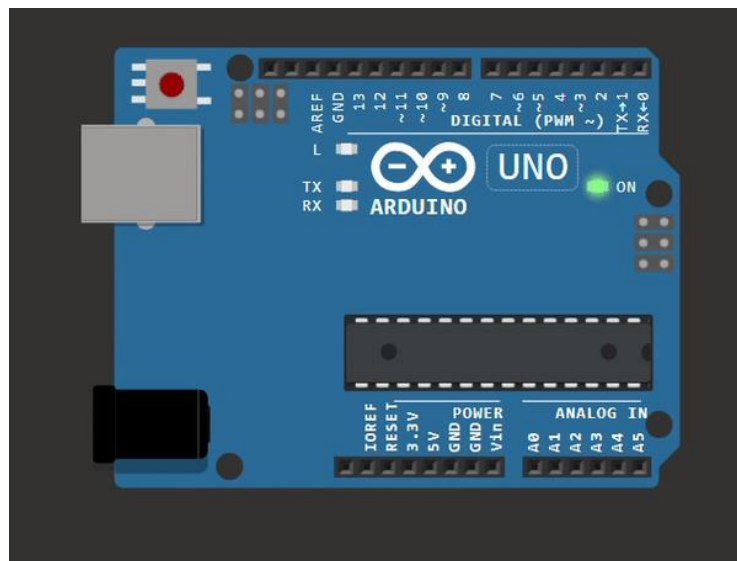


Figure 1 Uno R3: InBuilt LED Blink

Link of the project: [WOKWI LAB1](#)

Lab#2

Blinking an External LED (Green) w/ Resistor

```
#define light 10

void setup() {
  pinMode(light, OUTPUT);
}

void loop() {
  digitalWrite(light, HIGH);
  delay(300);
  digitalWrite(light, LOW);
  delay(600);
}
```

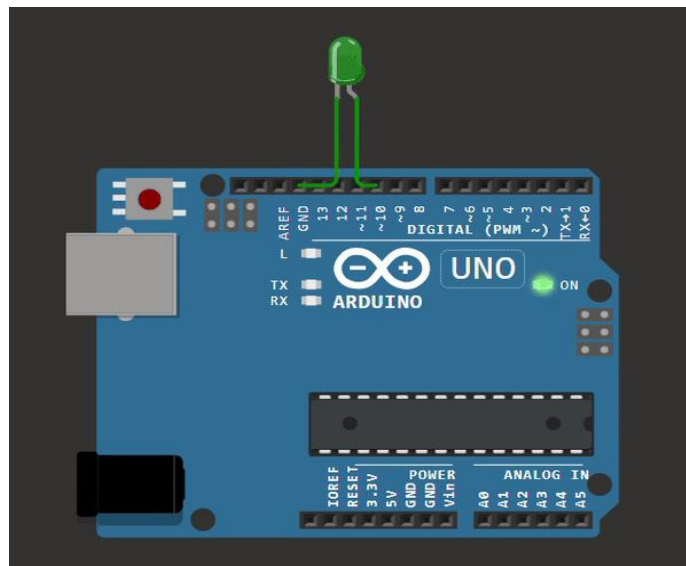


Figure 2 Uno R3: External LED Blink

Link of the project: [WOKWI LAB 2](#)

Lab#3

Using a Digital Humidity & Temperature Sensor

```
#include <DHT.h>

#define pin 7

#define DHTTYPE DHT22

DHT dht(pin, DHTTYPE);

float humid, temp;

void setup() {
    Serial.begin(9600);
    dht.begin();
}

void loop() {
    delay(200);

    humid = dht.readHumidity();
    temp = dht.readTemperature();
    Serial.print("Humidity: ");
    Serial.print(humid);
    Serial.print(" % Temperature: ");
    Serial.print(temp);
    Serial.println("°C");
    delay(1000);
}
```

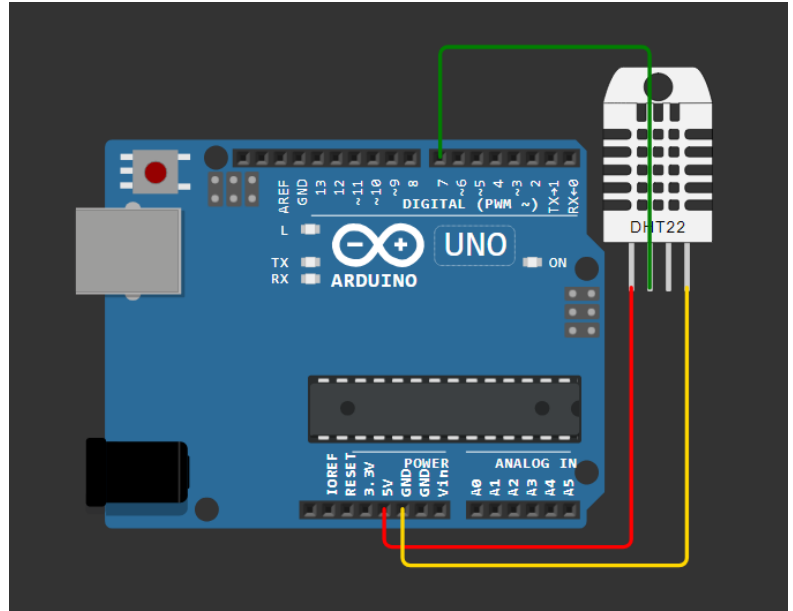


Figure 3 Uno R3: DHT22

```
Humidity: 40.00 % Temperature: 24.00°C
Humidity: 40.00 % Temperature: 24.00°C
Humidity: 40.00 % Temperature: 24.00°C
Humidity: 40.00 % Temperature: 24.00°C
Humidity: 40.00 % Temperature: 24.00°C
Humidity: 40.00 % Temperature: 24.00°C
Humidity: 40.00 % Temperature: 24.00°C
```

Figure 4 Wokwi Output [DHT Sensor]

Link of the Project: [WOKWI LAB3](#)

Lab#4

Configuring MQTT Service in my Machine

In Linux [WSL*: Ubuntu 22.04 LTS]:

- In Terminal > wsl --install -d Ubuntu-22.04 > \ E / N \ T / E \ R /
- Restart the machine, and Launch Ubuntu 22.04
 - \$sudo apt update
 - \$sudo apt install mosquitto mosquitto-clients
- Starting mosquitto services:
 - \$sudo systemctl (enable /start) mosquitto
- Mosquitto Broker Service Status can be checked here:
 - \$sudo systemctl status mosquitto
 - Once verified service status, transmission can be carried on.
- Stopping mosquitto services:
 - \$sudo systemctl stop mosquitto

```
suraj@Suraj:~$ sudo systemctl enable mosquitto
sudo systemctl start mosquitto
[sudo] password for suraj:
Synchronizing state of mosquitto.service with SysV service script with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable mosquitto
suraj@Suraj:~$ sudo systemctl status mosquitto
● mosquitto.service - Mosquitto MQTT Broker
   Loaded: loaded (/lib/systemd/system/mosquitto.service; enabled; vendor preset: enabled)
   Active: active (running) since Thu 2024-10-17 02:53:50 IST; 3min 10s ago
     Docs: man:mosquitto.conf(5)
           man:mosquitto(8)
  Main PID: 209 (mosquitto)
    Tasks: 1 (limit: 3474)
   Memory: 3.6M
   CGroup: /system.slice/mosquitto.service
           └─209 /usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf

Oct 17 02:53:50 Suraj systemd[1]: Starting Mosquitto MQTT Broker...
Oct 17 02:53:50 Suraj systemd[1]: Started Mosquitto MQTT Broker.
suraj@Suraj:~$ sudo systemctl stop mosquitto
suraj@Suraj:~$ sudo systemctl status mosquitto
○ mosquitto.service - Mosquitto MQTT Broker
   Loaded: loaded (/lib/systemd/system/mosquitto.service; enabled; vendor preset: enabled)
   Active: inactive (dead) since Thu 2024-10-17 03:07:39 IST; 15s ago
     Docs: man:mosquitto.conf(5)
           man:mosquitto(8)
  Process: 209 ExecStart=/usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf (code=exited, status=0/SUCCESS)
 Main PID: 209 (code=exited, status=0/SUCCESS)

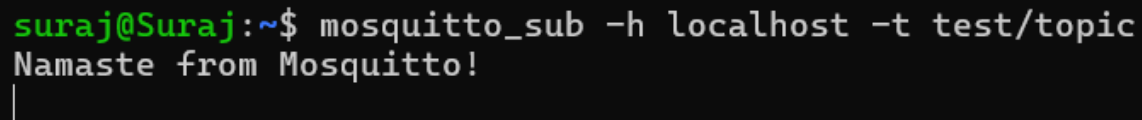
Oct 17 02:53:50 Suraj systemd[1]: Starting Mosquitto MQTT Broker...
Oct 17 02:53:50 Suraj systemd[1]: Started Mosquitto MQTT Broker.
Oct 17 03:07:39 Suraj systemd[1]: Stopping Mosquitto MQTT Broker...
Oct 17 03:07:39 Suraj systemd[1]: mosquitto.service: Deactivated successfully.
Oct 17 03:07:39 Suraj systemd[1]: Stopped Mosquitto MQTT Broker.
suraj@Suraj:~$
```

Figure 5 Mosquitto Initialization in Ubuntu 22.04

Testing MQTT Services [Message Transmission: Ubuntu 22.04]:

- Open 2 Terminals:

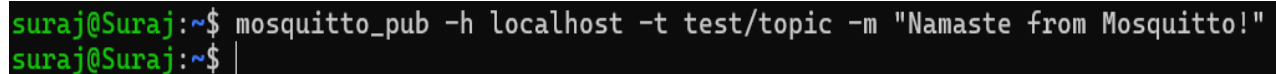
1st: mosquitto_sub.exe -h localhost -t test/topic

A terminal window with a black background. The prompt is 'suraj@Suraj:~\$'. The command 'mosquitto_sub -h localhost -t test/topic' is entered. The output 'Namaste from Mosquitto!' is displayed on the next line, followed by a cursor on a new line.

```
suraj@Suraj:~$ mosquitto_sub -h localhost -t test/topic
Namaste from Mosquitto!
|
```

Figure 6 MQTT Subscriber Testing [Message Transmission]

2nd: mosquitto_pub.exe -h localhost -t test/topic -m " Namaste from Mosquitto! "

A terminal window with a black background. The prompt is 'suraj@Suraj:~\$'. The command 'mosquitto_pub -h localhost -t test/topic -m "Namaste from Mosquitto!"' is entered. The prompt 'suraj@Suraj:~\$' is shown on the next line with a cursor.

```
suraj@Suraj:~$ mosquitto_pub -h localhost -t test/topic -m "Namaste from Mosquitto!"
suraj@Suraj:~$ |
```

Figure 7 MQTT Publisher Testing

*Windows Subsystem for Linux

Lab#5

Realtime DHT Sensor Data on NodeRED

Install Node.js :

- Installed NodeJS from Official Eclipse Page [<https://nodejs.org/en/download/package-manager>].
- Added node.js to the System Environment Variables PATH [C:/Users/suraj/AppData/Roaming/npm '], which allows us to use npm commands directly in the Command Prompt or, Terminal.

Installing & Initialising NodeRED:

- Open Node.js > npm install node-red-dashboard
- [PostInstallation] > Elevated CMD: node-red
 - In Client Application, browsed localhost:1880 [Accessing NodeRED]
- Inside the NodeRED window, a flow was created w/ the nodes as:
 - > SERIAL-IN (Arduino Uno R3 Board)
 - > DEBUGGER
 - > DHT FUNCTION
 - > 2 GAUGES (*Humidity & Temperature*)

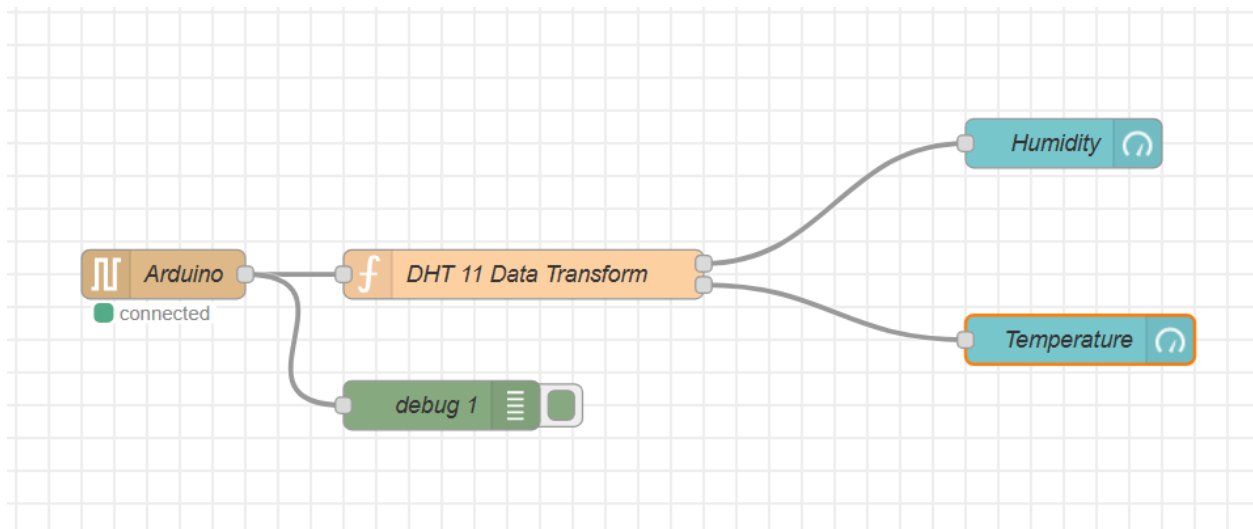


FIG 9: Node-RED Flow Diagram

Serial In Node: Configured it to read from the correct serial port where my Arduino is connected (e.g., COM7) > Set the baud rate to 9600.

- Configure the DHT Function as:

```
var m = msg.payload.split(',');
```

```
if (m.length === 2) {
```

```
var H = { payload: parseFloat(m[0]) };
```

```
    var T = { payload: parseFloat(m[1]) };
```

```
    return [H, T];
```

```
} else {
```

```
return null; }
```

- Adjusting Gauge Nodes:

Humidity:

- Title as “ Humidity ”.
- Value format as ‘ {{value}}% ’.
- Range Value: 0 ~ 100 %.

Temperatue:

- Title as ' Temperature '.
- Value format as ' {{value}}°C '.

***Ensure that Humidity & Temperature are in the same group.*

Deployment:

- Uploaded DHT11 /22 Sketch to the Arduino Board through its IDE:

```
#include <DHT.h>
```

```
#define DHTPIN 3
```

```
#define DHTTYPE DHT11
```

```
DHT dht(DHTPIN, DHTTYPE);
```

```
void setup() {
```

```
  Serial.begin(9600);
```

```
  dht.begin();
```

```
}
```

```
void loop() {
```

```
  float H = dht.readHumidity();
```

```
  float T = dht.readTemperature();
```

```
  if (isnan(H) || isnan(T)) {
```

```
    Serial.println("Failed to read from DHT sensor!");
```

```
  } else {
```

```
    Serial.println(String(H) + "," + String(T));
```

```
  }
```

```
  delay(2000);
```


}

- After uploading this sketch, close the IDE.
- Deploy the flow in NodeRED.
- Check the Dashboard in the upper-right corner, for the Humidity and Temperature Gauge.
- OUTPUT:

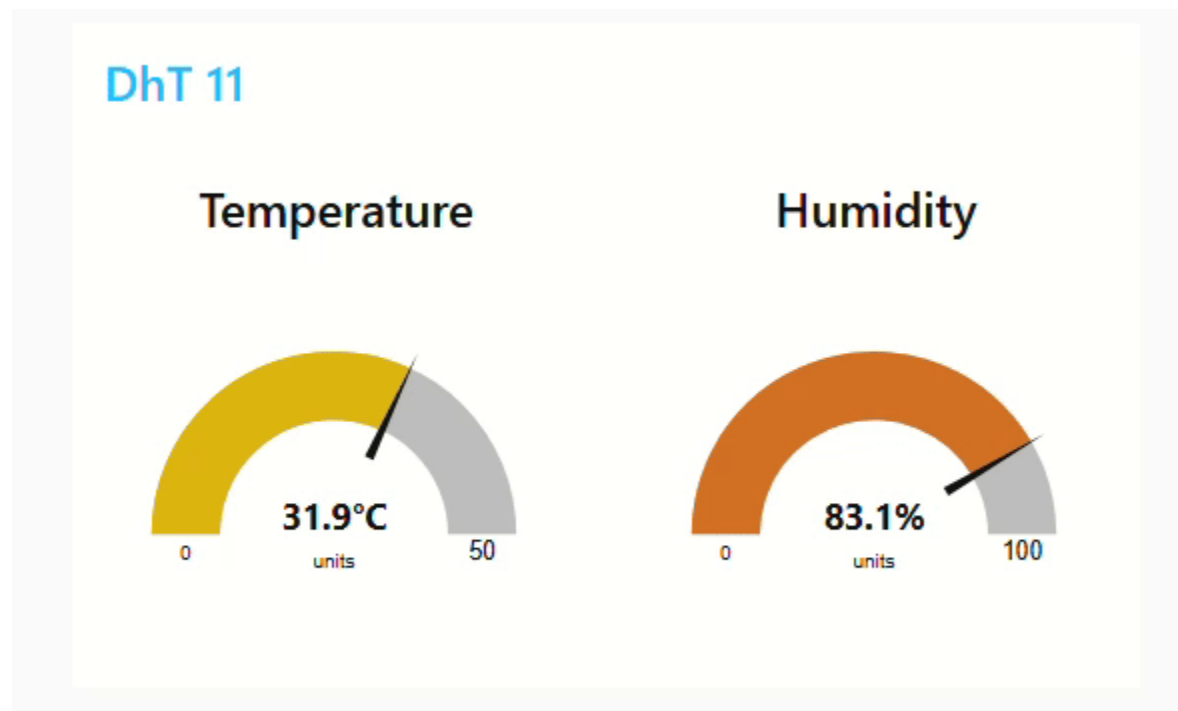


FIG 10: Dashboard measuring the temperature and humidity

Lab#6

Measuring the distances using Ultrasonic Sensors

```
const int trigPin = 10;
```

```
const int echoPin = 6;
```

```
float duration, distance;
```

```
void setup() {
```

```
    pinMode(trigPin, OUTPUT);
```

```
    pinMode(echoPin, INPUT);
```

```
    Serial.begin(9600);
```

```
}
```

```
void loop() {
```

```
    digitalWrite(trigPin, LOW);
```

```
    delayMicroseconds(2);
```

```
    digitalWrite(trigPin, HIGH);
```

```
    delayMicroseconds(10);
```

```
    digitalWrite(trigPin, LOW);
```

```
    duration = pulseIn(echoPin, HIGH);
```

```
    distance = (duration*.0343)/2;
```

```
    Serial.print("Distance: ");
```

```
    Serial.println(distance);
```

```
    delay(1000);
```

```
}
```

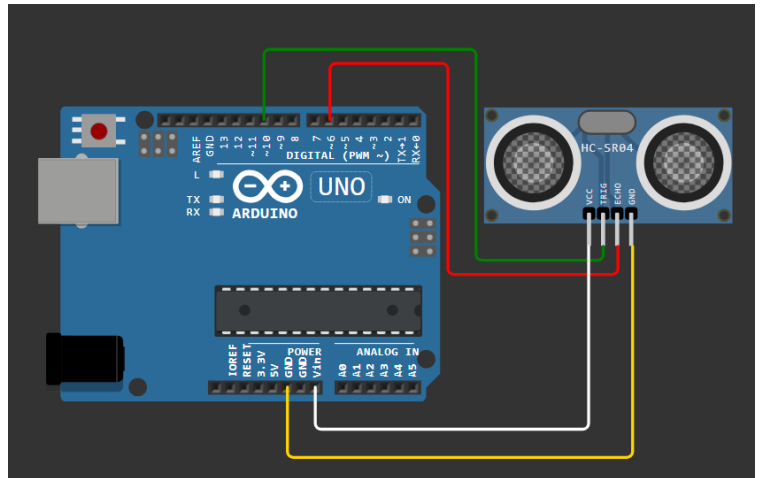


Fig 11: Ultrasonic Distance Sensor

```
Distance: 401.16
Distance: 401.16
Distance: 401.16
Distance: 401.14
Distance: 401.16
Distance: 401.16
Distance: 401.16
Distance: 401.14
```

Fig12: Wokwi Output

Link of the Project: [WOKWI LAB 6](#)

Lab#7

Use of Breadboard

A breadboard is a device for creating temporary electronic circuits without soldering. It is an essential tool for prototyping and testing electronic circuits quickly and easily.

Key Features

1. Rows and Columns: Breadboards have rows (numbered) and columns (lettered). The rows in the middle are typically connected horizontally, while the side rows (power rails) run vertically and are useful for connecting power supplies.
2. Power Rails: These are long rows along the edges, usually marked with a red (+) and blue (-) line, used for distributing power across the board.

HOW to use a Breadboard

1. Insert Components: Push the leads of components into the holes.
2. Make Connections: Use jumper wires to connect components.
3. Power Connections: Connect your power source to the power rails first to distribute power across the board easily.
4. Avoiding Overload: Breadboards can handle only low current and low power. High currents can damage the contacts, so avoid overloading.

Tips

- Plan the Layout: Organize the layout of components and wires to minimize clutter.

- Check Connections: Ensure each component and wire is fully inserted.

Experiment to understand breadboard (Blinking Multiple LEDs)

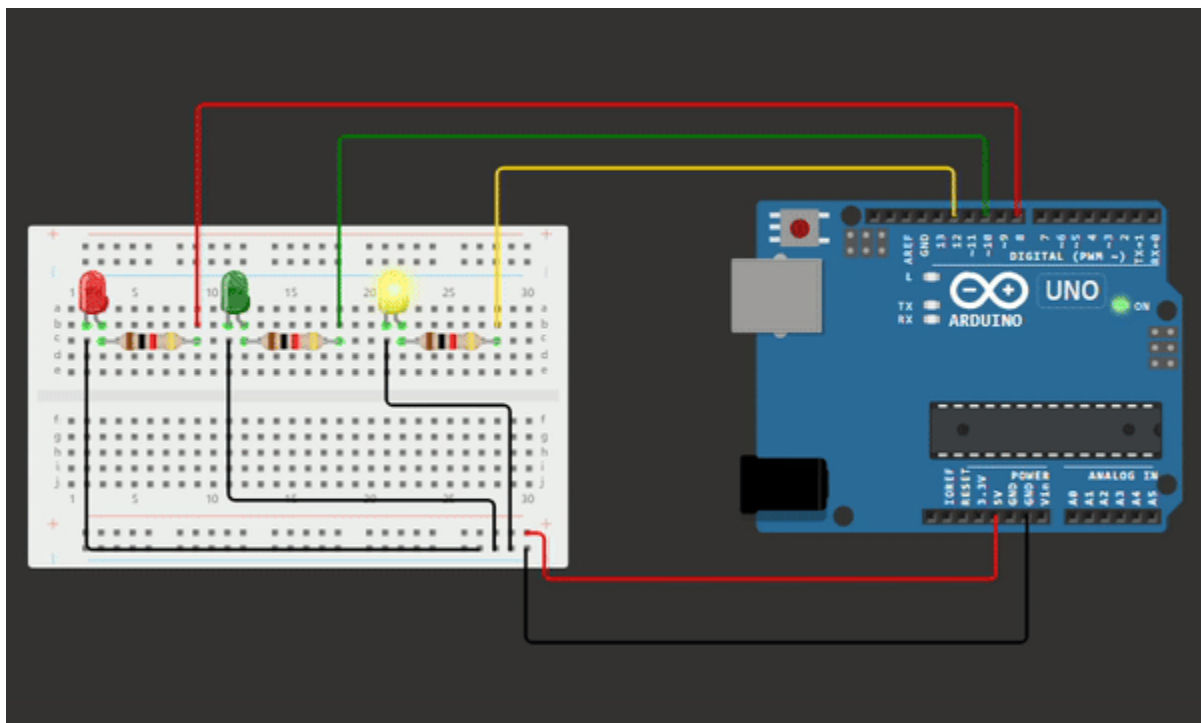


Fig 13: Blinking of Multiple LEDs

Link of the Project: [WOKWI LAB 7](#)

Lab#8

Use of ESP32, upload code on ESP 32 to blink onboard LED

Procedure:

1. Install ESP32 Support in Arduino IDE:

- Go to File > Preferences in the Arduino IDE.
- In Additional Boards Manager URLs, paste:
https://dl.espressif.com/dl/package_esp32_index.json and click OK.
- Open Boards Manager (Tools > Board > Boards Manager) and search for ESP32 by Espressif Systems. Click Install.

2. Install USB Driver:

- Download and install the CP210x USB-to-UART Bridge VCP Driver to enable communication between your computer and the ESP32 board.

3. Select Board and Port in Arduino IDE:

- After restarting, select ESP32 Dev Module from the Tools > Board menu.
- Choose the correct COM port for your ESP32 (e.g., COM7).

4. Put ESP32 in Boot Mode:

- Press and hold the BOOT button while uploading the code.
- Release the BOOT button once the upload begins (when you see "Connecting...").

5. Code Execution for LED Blink:

- Use the following code in the Arduino IDE to blink the onboard LED.

CODE:

```
// Define the LED pin
#define LED_PIN 2 // The onboard LED is connected to GPIO 2 on most ESP32 boards

void setup() {
  // Initialize the LED pin as an output
  pinMode(LED_PIN, OUTPUT);
}

void loop() {
  // Turn the LED on (HIGH is the voltage level)
  digitalWrite(LED_PIN, HIGH);
  // Wait for a second
  delay(2000);
  // Turn the LED off by making the voltage LOW
  digitalWrite(LED_PIN, LOW);
  // Wait for a second
  delay(2000);
}
```

OUTPUT:

