**INTERNET OF THINGS Lab. Record**

**Subject Code: BTCS-AMDS-009T**

|  |  |
| --- | --- |
| Name: | Meheak Sahu |
| Registration Number: | FET-BAML-2022-26-025 |
| Course: | IOT |
| Semester: | 5th |
| Faculty: | Biswajeeban Mishra  Pritam Nanda |

|  |  |
| --- | --- |
| Remarks |  |
| Signature |  |



SRI SRI UNIVERSITY

Bidyadharpur, Cuttack, Odisha.

Index

|  |  |  |  |
| --- | --- | --- | --- |
| **Sl. No.** | **Experiment/Case Study** | **Page No.** | **Remark** |
| **1** | Blinking of Internal LED |  |  |
| **2** | Blinking of External LED |  |  |
| **3** | Read Humidity and Temperature value using DHT22 |  |  |
| **4** | A case study on a communication Protocol: MQTT |  |  |
| **5** | Building a web app using Node-Red to fetch DHT sensor data and display it on the web app dashboard |  |  |
| **6** | Interfacing Ultrasonic Sensor |  |  |
| **7** | Use of Breadboard |  |  |
| **8** | Use of ESP32, upload code on ESP 32 to blink onboard LED |  |  |
| **9** |  |  |  |
| **10** |  |  |  |

***Experiment 1:***

**Aim of the experiment:**

Blink Inbuilt Led of your Arduino Board.

**Requirements:**

* Arduino UNO board
* USB cable
* Computer with Arduino IDE

**Description:**

Arduino boards come with a built-in LED connected to digital pin 13. This LED can be controlled through programming to turn it ON and OFF, which creates a blinking effect. We use digitalWrite() function to control the LED state and delay() function to create time intervals between states and the variables high and low represent the voltage when the led turns on and off.

**Procedure:**

1. Connect your Arduino board to the computer using the USB cable.
2. Open the Arduino IDE on your computer.
3. Create a new sketch and type the following code:

**Code:**

void setup() {

// Set pin 13 (built-in LED) as output

pinMode(LED\_BUILTIN, OUTPUT);

}

void loop() {

digitalWrite(LED\_BUILTIN, HIGH); // Turn LED on

delay(1000); // Wait for 1 second

digitalWrite(LED\_BUILTIN, LOW); // Turn LED off

delay(1000); // Wait for 1 second

}

1. On the above click on Tools menu and select:

* The correct board (Arduino UNO)

1. Click the "Upload" button (arrow icon) to upload the code to your Arduino.

**Observations:**

We can observe that the built-in LED on your Arduino board start blinking with a 1-second interval.

**Conclusion:**

Through this experiment we demonstrated how setup() and loop() function work and the usage of digitalWrite() and delay() functions. And how we can control the digital outputs timing sequence.

***Experiment 2:***

**Aim of the experiment:**

To control an external LED with Arduino board, making it blink at regular intervals.

**Requirements:**

* Arduino UNO board
* USB cable
* External LED
* Two wires
* Computer with Arduino IDE

**Description:**

In this experiment an external LED is connected to a digital pin of the Arduino, and the program will make the LED blink at regular intervals. Also demonstrates Arduino's ability to control external hardware. By making an LED blink through programmed commands.

**Procedure:**

1. Connect your Arduino board to the computer using the USB cable.
2. Connect the longer leg (anode) of the LED to pin 8 on the Arduino.
3. Connect the shorter leg (cathode to the ground (GND) of the Arduino.
4. Open the Arduino IDE on your computer.
5. Create a new sketch and type the following code:

**Code:**

// Pin number for the external LED

int ledPin = 8;

void setup() {

// Initialize the external LED as an output

pinMode(ledPin, OUTPUT);

}

void loop() {

digitalWrite(ledPin, HIGH); // Turn the LED on

delay(1000); // Wait for 1 second

digitalWrite(ledPin, LOW); // Turn the LED off

delay(1000); // Wait for 1 second

}

1. Select "Arduino UNO" from Tools → Board menu
2. Click upload button

**Observations:**

We can observe that the external LED start blinking with a 1-second interval.

**Conclusion:**

Through this experiment we understood the concept of controlling the external LED with the Arduino board.And how to establish the connections between the resistors ,pins and the LEDs.

****

***Experiment 3:***

**Aim of the experiment:**

Measuring temperature and pressure using DHT sensor

**Requirements:**

* Arduino UNO board
* USB cable
* External LED
* Two wires
* Computer with Arduino IDE

**Description:**

The DHT22 sensor is a basic digital temperature and humidity sensor. It uses a capacitive humidity sensor and a thermistor to measure the surrounding air, and sends a digital signal to display on serial monitor.

**Procedure:**

1. Connect your Arduino board to the computer using the USB cable.
2. Connect the VCC pin of the DHT22 sensor to the 5V pin on the Arduino.
3. Connect the GND pin to the ground (GND) of the Arduino.
4. Connect the data pin to pin 2 on the Arduino.
5. Open the Arduino IDE on your computer.
6. Install DHT library (Sketch → Include Library → Manage Libraries).
7. Search for "DHT sensor library" by Adafruit and install it
8. Create a new sketch and type the following code:

**Code:**

#include "DHT.h"

#define DHTPIN 2 // DHT sensor connected to digital pin 2

#define DHTTYPE DHT22 // Define the type of DHT sensor

DHT dht(DHTPIN, DHTTYPE);

float humid, temp;

void setup() {

Serial.begin(9600); // Start the serial monitor at 9600 baud rate

dht.begin(); // Initialize the DHT sensor

}

void loop() {

delay(200); // Wait a few seconds between measurements

humid = dht.readHumidity(); // Read humidity

temp = dht.readTemperature(); // Read temperature in Celsius

// Print the results to the Serial Monitor

Serial.print("Humidity: ");

Serial.print(humid);

Serial.print(" %\t");

Serial.print("Temperature: ");

Serial.print(temp);

Serial.println(" °C");

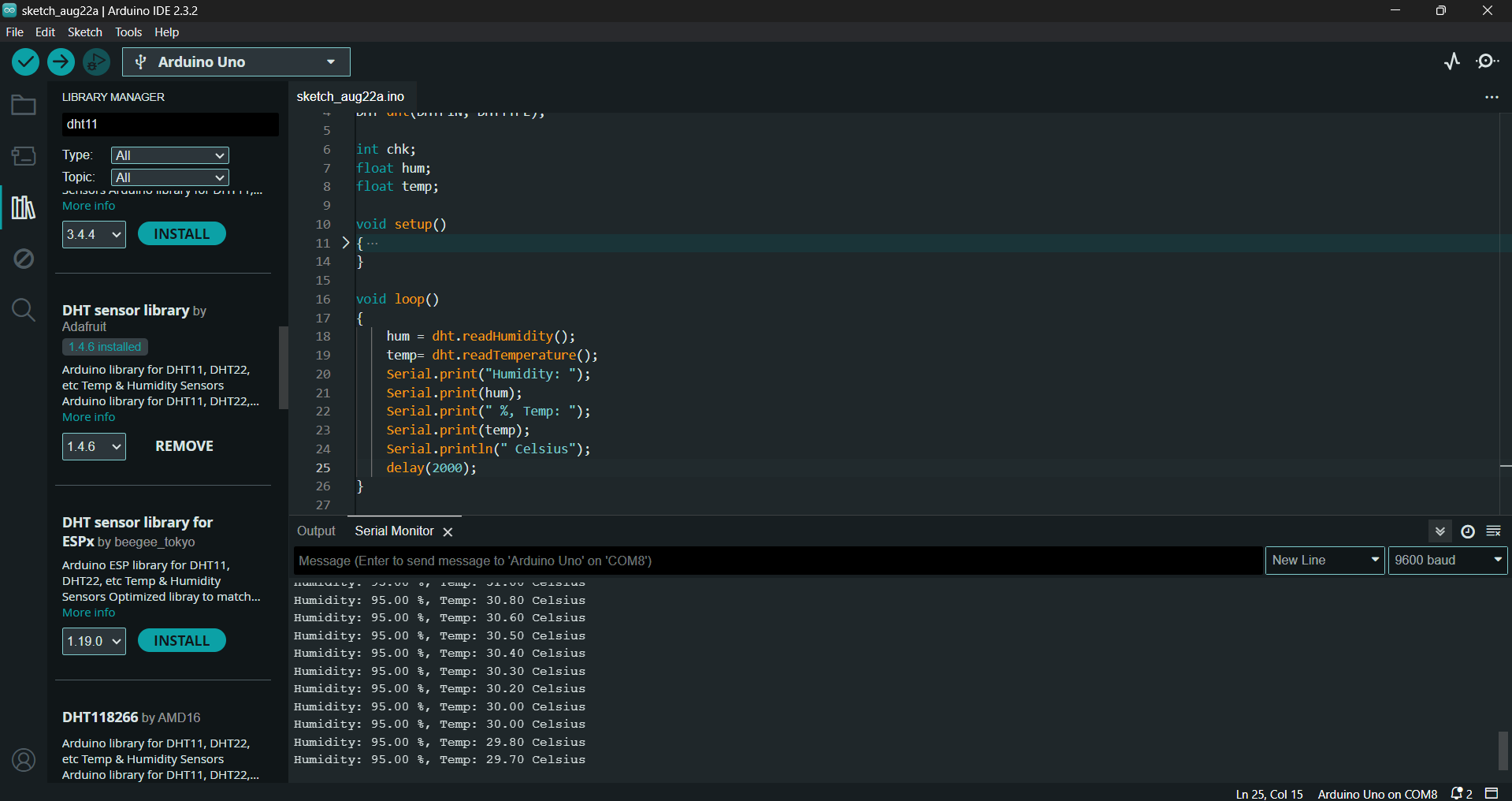
delay(1000);

}

**Observations:**

The DHT sensor provides continuous monitoring of environmental conditions, updating both temperature and humidity readings every two seconds. Temperature values are shown in Celsius, while humidity readings are displayed as percentages.

Output:



**Conclusion:**

In this experiment, we successfully interfaced the DHT22 sensor with the Arduino to measure humidity and temperature. This data was displayed on the serial monitor. The DHT library simplifies sensor interaction, allowing us to focus on gathering and displaying environmental data. This experiment is useful for weather-related projects or environmental monitoring systems. We also learn to use external libraries, making it a perfect introduction to Arduino-based environmental monitoring projects.

***Experiment 4:***

**Aim of the experiment:**

A case study on a communication Protocol: MQTT

**Description:**

MQTT is like a messaging system designed specifically for Internet of Things (IoT) devices. Think of it as a postal service for smart devices. Mainly it has 3 components. They are Broker, Subscriber, Publisher. These 3 play a vital role in the communication.

In this experiment we are going to configure the MQTT in our local Machine.

**Procedure:**

**In SystemOS [ Windows11 ]:**

* Installed Mosquitto as a Service from Official Eclipse Page [ https://mosquitto.org/download/ ].
* This allows the MQTT Broker to run automatically in the background.
* Added mosquittio.exe to the System Environment Variables PATH [ ' C:\Program Files\mosquitto ' ], which allows us to use MQTT commands directly in the Command Prompt or, Terminal.
  + Starting @ boot byDefault:

net start mosquitto

* + Stopping:

In Elevated CMD > net stop mosquitto

* For Transmission: Navigate to [ cd C:/Program Files/mosquitto ]

mosquitto.exe -v

* // -v is a Verbose Output flag, that enables us to see the backend processes, log messages, that'd help us to debug whenever necessary.

**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

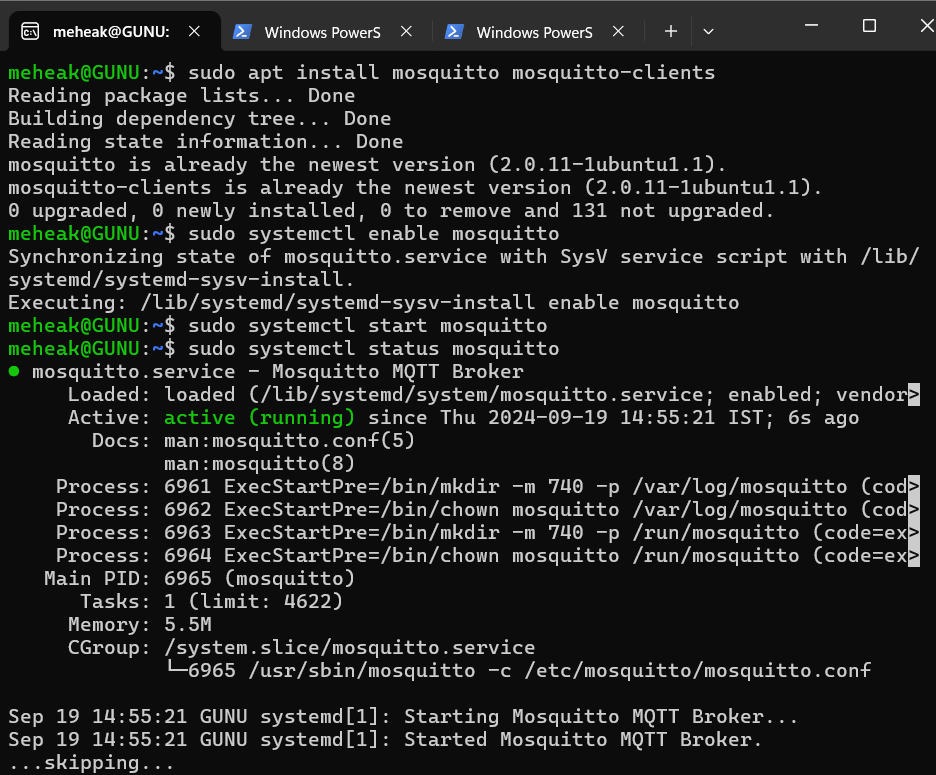
**Practical implementation**:

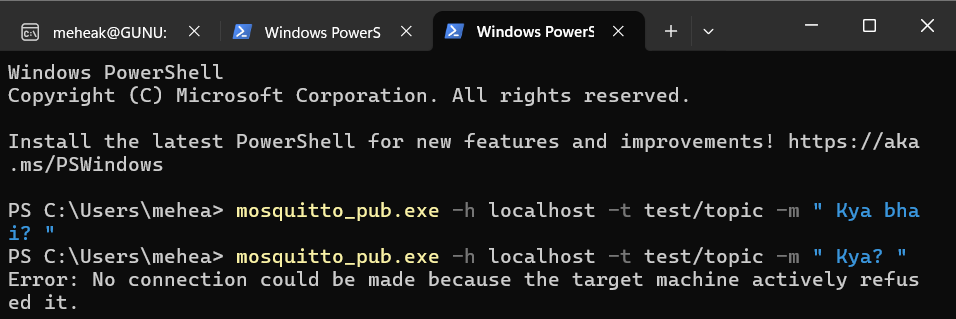
**Testing MQTT Services [ Message Transmission: WinOS11 + Ubuntu 22.04 ]:**

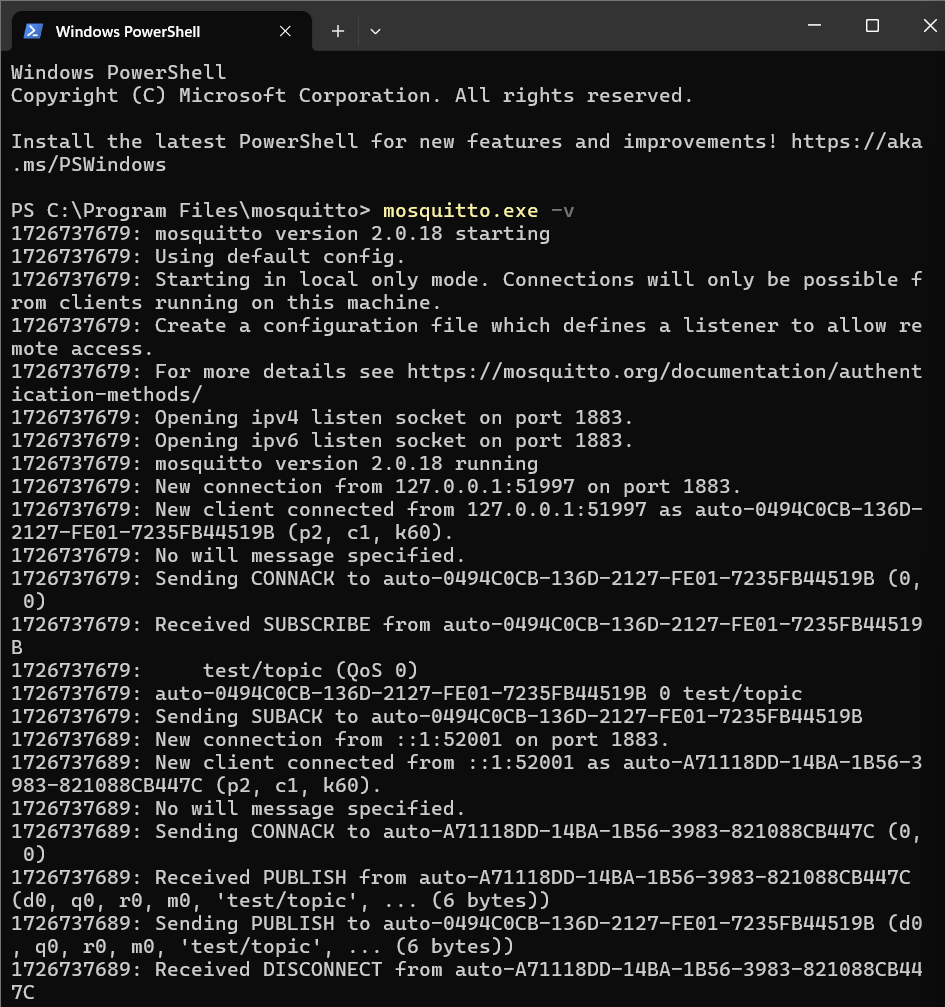
* Open 2 Terminals:
* 1st: mosquitto\_sub.exe -h localhost -t test/topic
* 2nd: mosquitto\_pub.exe -h localhost -t test/topic -m " Light was here! "

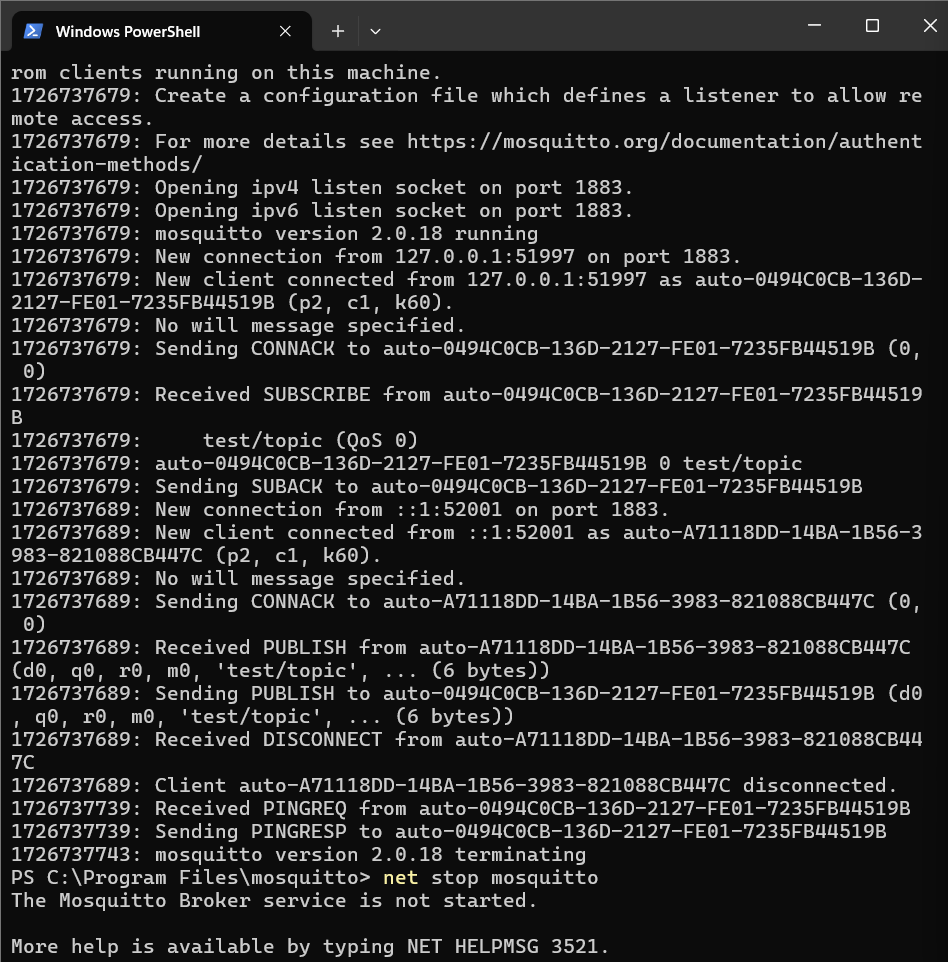
**Conclusion:** Through this experiment we got an idea how mosquito protocol really works. And understood how to perform those activities.

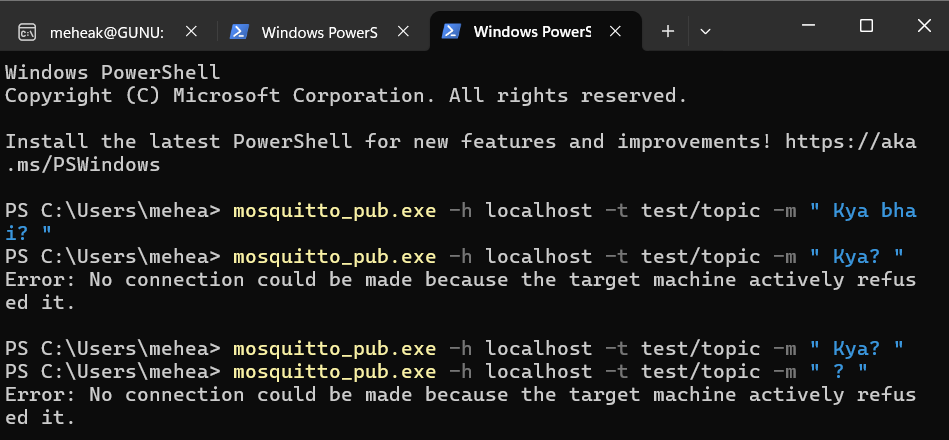
**Output Images:**

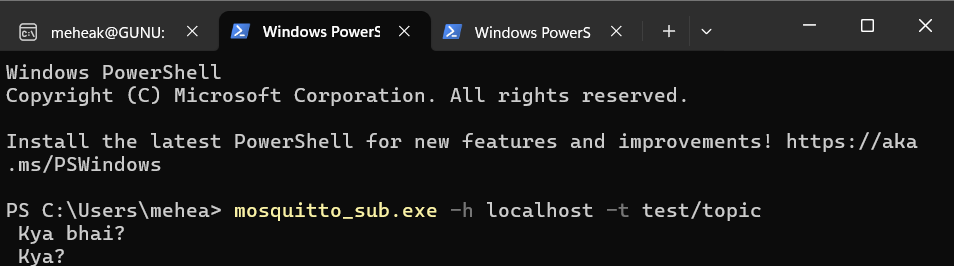












***Experiment 5:***

**Aim of the experiment:**

Building a web app using Node-Red to fetch DHT sensor data and display it on the web app dashboard.

**Requirements:**

* Node.js platform
* NodeRED

**Procedure:**

**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\Hi\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 )
* Serial In Node: Configured it to read from the correct serial port where my Arduino is connected (e.g., COM11) > 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}}%`.

- Minimum value to 0 and the maximum to 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.

***Experiment 6:***

**Aim of the experiment:**

Interfacing Ultrasonic Sensorand working with LED

**Requirements:**

Arduino UNO, USB cable, External LED,Two jumper wires,Computer with Arduino IDE installed

**Procedure:**

**1.Button: Held or, Released?**

Step#1: Connect 2 jumper wires *diagonally* to the Button.

Step#2: One wire to Digital Pin [ *here, 2* ], and the other wire to GND.

Step#3: After configuring the Arduino UNO R3, this sketch is to be uploaded and executed in the IDE:

Step#4: Create a new sketch and type the following code:

**Code:**

const int buttonPin = 2; // Pin where the button is connected

void setup() {

Serial.begin(115200); // Initialize serial communication at 115200 baud

pinMode(buttonPin, INPUT\_PULLUP); // Set the button pin as input with internal pull-up resistor

}

void loop() {

int buttonState = digitalRead(buttonPin); // Read the state of the button

if (buttonState == LOW) { // Check if the button is pressed

Serial.println("Button Held!"); // Print message to Serial Monitor

}

else {

Serial.println("Button Released!"); // Print message to Serial Monitor

}

delay(500); // Add a small delay to debounce the button

}

**Output:** Button Held! ; Button Released

**Procedure:**

**2.UltraSonic [ HC-SR04 ]: Measures Distance**

Step#1: Connect 4 jumper wires to the UltraSonic Sensor as [VCC: 5v], [Trigger: 9], [Echo: 8], & **GND**.

Step#2: After configuring the **HC-SR04** w/ **UNO R3**, this sketch is to be uploaded and executed in the IDE:

Code:

#define PIN\_TRIG 9 // Define the pin for the trigger

#define PIN\_ECHO 8 // Define the pin for the echo

void setup() {

Serial.begin(9600); // Initialize serial communication at 9600 baud

pinMode(PIN\_TRIG, OUTPUT); // Set the trigger pin as output

pinMode(PIN\_ECHO, INPUT); // Set the echo pin as input

}

void loop() {

// Start a new measurement:

digitalWrite(PIN\_TRIG, HIGH); // Set the trigger pin high

delayMicroseconds(10); // Wait for 10 microseconds

digitalWrite(PIN\_TRIG, LOW); // Set the trigger pin low

// Read the result:

int duration = pulseIn(PIN\_ECHO, HIGH); // Read the duration of the pulse from the echo pin

// Calculate distance in centimeters:

Serial.print("Distance in CM: ");

Serial.println(duration / 58); // Print the distance in centimeters

// Calculate distance in inches:

Serial.print("Distance in inches: ");

Serial.println(duration / 148); // Print the distance in inches

delay(1000); // Wait for 1 second before taking the next measurement

}

**Output:**Distance in CM: 121 ; Distance in inches: 47

**Procedure:**

**3.HC-SR04 + PushButton**

Step#3: Now, a button can be connected to Digital 2 and diagonally to **GND**.

Step#4: After configuring the Button & the sensor, this sketch is to be uploaded and executed in the IDE:

Code:

#define PIN\_TRIG 9 // Define the pin for the trigger

#define PIN\_ECHO 8 // Define the pin for the echo

#define buttonPin 2 // Define the pin for the button

void setup() {

Serial.begin(9600); // Initialize serial communication at 9600 baud

pinMode(PIN\_TRIG, OUTPUT); // Set the trigger pin as output

pinMode(PIN\_ECHO, INPUT); // Set the echo pin as input

pinMode(buttonPin, INPUT\_PULLUP); // Set the button pin as input with internal pull-up resistor

}

void loop() {

int buttonState = digitalRead(buttonPin); // Read the state of the button

if (buttonState == LOW) { // Check if the button is pressed

// Start a new measurement:

digitalWrite(PIN\_TRIG, HIGH); // Set the trigger pin high

delayMicroseconds(10); // Wait for 10 microseconds

digitalWrite(PIN\_TRIG, LOW); // Set the trigger pin low

// Read the result:

int duration = pulseIn(PIN\_ECHO, HIGH); // Read the duration of the pulse from the echo pin

// Calculate distance in centimeters:

Serial.print("Distance in CM: ");

Serial.println(duration / 58); // Print the distance in centimeters

// Calculate distance in inches:

Serial.print("Distance in Inches: ");

Serial.println(duration / 148); // Print the distance in inches

} else {

Serial.println("Button Released!"); // Print message to Serial Monitor

}

delay(500); // Add a small delay to debounce the button

}

**Output:**Button Released! ... ; Distance in CM: 24 ; Distance in Inches: 9

**Procedure:**

**4.Combining PushButton + UltraSonic Sensor + External LED + DHT Sensor**

Step#1: Get 9 Jumper wires [ M2F ], and connect them as:

* HC-SR04:
  + Trigger [ *Digital 9* ]
  + Echo [ *Digital 8* ]
  + VCC [ *5v* ]
  + GND
* DHT 11:
  + VCC [ *3.3v* ]
  + Data [ *Digital 4* ]
  + GND
* Button:
  + Corner 1 [ *Digital 2* ]
  + Diagonal Corner 4 [ GND ]
* External LED:
  + Anode [ +ve, *Digital 13* ]
  + Cathode [ -ve, GND]

Step#2: After configuring the entire Arduino Uno R3 with proper wire connections, the following could be executed in the IDE:

Code:

#include <DHT.h>

#define DHTPIN 4 // Pin where the DHT sensor is connected

#define DHTTYPE DHT11 // Define the type of DHT sensor (DHT11 in this case)

#define PIN\_TRIG 9 // Define the pin for the ultrasonic sensor trigger

#define PIN\_ECHO 8 // Define the pin for the ultrasonic sensor echo

#define BUTTON\_PIN 2 // Define the pin for the button

#define LED\_PIN 13 // Define the pin for the LED

DHT dht(DHTPIN, DHTTYPE);

void setup() {

Serial.begin(9600); // Initialize serial communication at 9600 baud

dht.begin(); // Initialize the DHT sensor

pinMode(PIN\_TRIG, OUTPUT); // Set the ultrasonic sensor trigger pin as output

pinMode(PIN\_ECHO, INPUT); // Set the ultrasonic sensor echo pin as input

pinMode(BUTTON\_PIN, INPUT\_PULLUP); // Set the button pin as input with internal pull-up resistor

pinMode(LED\_PIN, OUTPUT); // Set the LED pin as output

}

void loop() {

int buttonState = digitalRead(BUTTON\_PIN); // Read the state of the button

if (buttonState == LOW) { // Check if the button is pressed

// Ultrasonic Sensor Measurement:

digitalWrite(PIN\_TRIG, HIGH); // Set the trigger pin high

delayMicroseconds(10); // Wait for 10 microseconds

digitalWrite(PIN\_TRIG, LOW); // Set the trigger pin low

int duration = pulseIn(PIN\_ECHO, HIGH); // Read the duration of the pulse from the echo pin

float distanceCm = duration / 58.0; // Calculate the distance in centimeters

Serial.print("Distance in CM: ");

Serial.println(distanceCm);

if (distanceCm < 20) { // Example condition: if distance is less than 20 cm

digitalWrite(LED\_PIN, HIGH); // Turn the LED on

} else {

digitalWrite(LED\_PIN, LOW); // Turn the LED off

}

// DHT Sensor Measurement:

float humidity = dht.readHumidity();

float temperature = dht.readTemperature();

if (isnan(humidity) || isnan(temperature)) {

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

} else {

Serial.print("Humidity: ");

Serial.print(humidity);

Serial.print(" %\t");

Serial.print("Temperature: ");

Serial.print(temperature);

Serial.println(" \*C");

}

} else {

Serial.println("Button Released!");

}

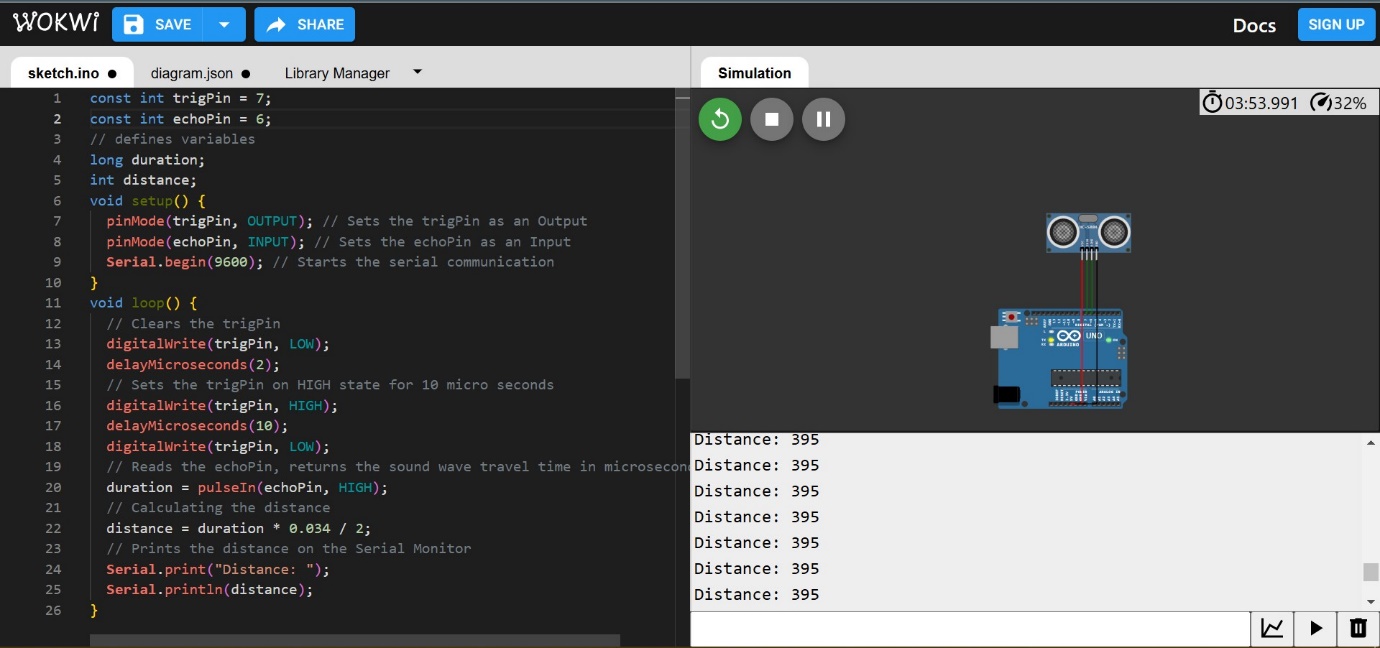
delay(1500); // Delay for 1.5 seconds before the next loop

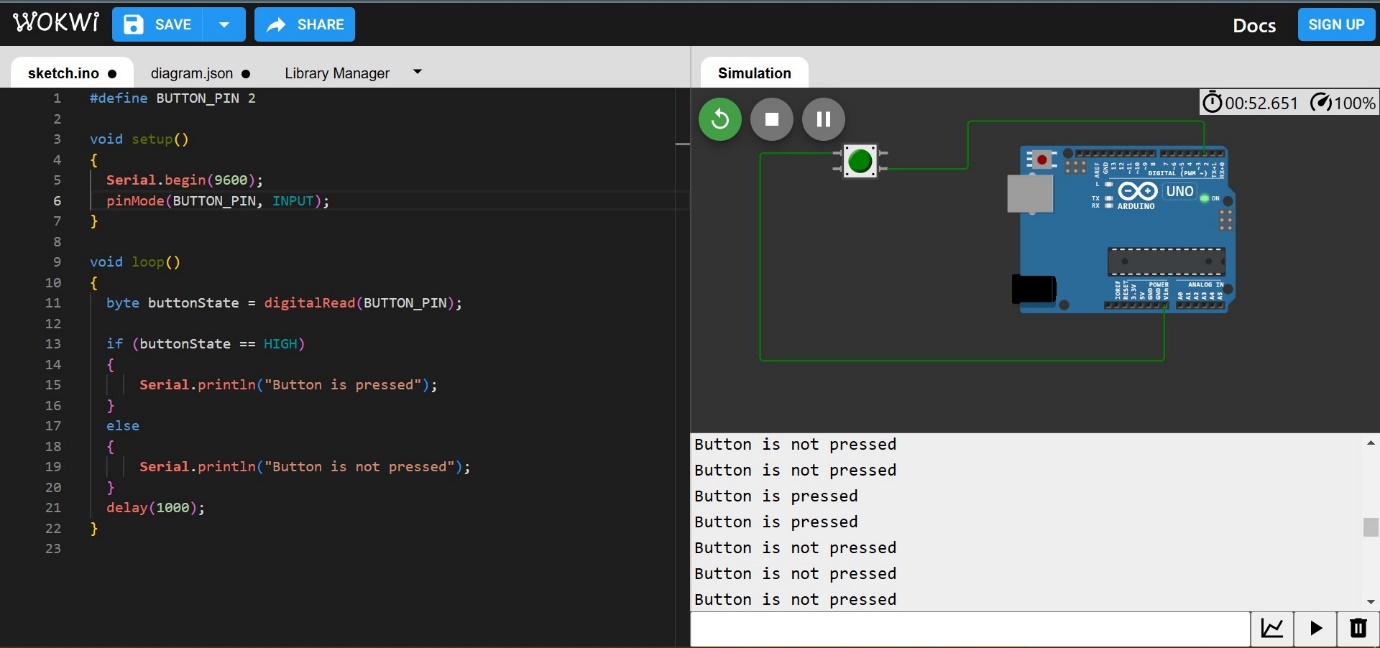
}

**Output:**Button Released!; Distance in CM: 18.45 -- Humidity: 79.00 % -- Temperature: 30.80 \*C [ LED TURNED ON ] ; Distance in CM: 24.28 -- Humidity: 65.00 % -- Temperature: 30.80 \*C [ LED TURNED OFF ] ; Button Released!

**Conclusion:** Through this experiment we understood the concept of working with button, LED and Ultrasonic sensor.

**Output Images:**

****

****

***Experiment 7:***

**Aim of the experiment:**

Setting UpESPressif32 DevKit v1 [ NodeMCU: MicroController Unit ].

**Requirements:** Arduino IDE, USB cable.

**Procedure:**

**Setting Up ESPressif32**

Step#1: Inside Arduino IDE, Navigate to Files > Preferences > Additional Boards Manager URL > <https://dl.espressif.com/dl/package_esp32_index.json> < *paste this and click OK*

Step#2: Then head into BOARDS MANAGER > and Install esp32 by ESPressif Systems.

Step#3: Now, for establishing the connection, we would need to configure our mainframe w/ the CP210x USB-to-UART Bridge Virtual COM Port (VCP) Driver avaliable at [CP210x USBtoUART Driver](https://www.silabs.com/documents/public/software/CP210x_Windows_Drivers.zip).

Step#4: After a superfluous reboot, the mainframe is ready to be used w/ an ESPressif32, as in the Arduino IDE, we first select the correct COM port (*here, COM12*), and ESP32 Dev Module as the board.

* NOTE: While executing a sketch, the Board reuires to be in DOWNLOAD MODE /BOOT MODE, so for, the BOOT button is to be pressed while uploading the code onto the board, exactly post Connecting... for 3-4 seconds.
  + A simple way around for this redundancy, would be:
    - Holding the BOOT button (GPIO0), the EN (Enable Pin) button is pressed for a second.
    - Post releasing the EN button, finally, the BOOT button is let go.
  + *This would keep the Espressif32 in the bootloader mode, so it could be equipped w/ any further sketch executions.*

Step#5: To blink the internal LED, the following is to be executed in the IDE:

**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(1000);

// Turn the LED off by making the voltage LOW

digitalWrite(LED\_PIN, LOW);

// Wait for a second

delay(1000);

}

**Output**: Internal LED (blue) blinks @ 1s duration

**Procedure:**

**Configuring the ESPressif to SCAN Wireless-Fidelity Signals!**

Step#6: The ESP32 can be used to scan nearby Wi-Fi signals using the following sketch:

**Code:**

#include "WiFi.h"

void setup() {

Serial.begin(115200);

// Set WiFi to station mode and disconnect from an AP if it was previously connected.

WiFi.mode(WIFI\_STA);

WiFi.disconnect();

delay(100);

Serial.println("Setup done");

}

void loop() {

Serial.println("Scan start");

// WiFi.scanNetworks will return the number of networks found.

int n = WiFi.scanNetworks();

Serial.println("Scan done");

if (n == 0) {

Serial.println("no networks found");

} else {

Serial.print(n);

Serial.println(" networks found");

Serial.println("Nr | SSID | RSSI | CH | Encryption");

for (int i = 0; i < n; ++i) {

// Print SSID and RSSI for each network found

Serial.printf("%2d", i + 1);

Serial.print(" | ");

Serial.printf("%-32.32s", WiFi.SSID(i).c\_str());

Serial.print(" | ");

Serial.printf("%4ld", WiFi.RSSI(i));

Serial.print(" | ");

Serial.printf("%2ld", WiFi.channel(i));

Serial.print(" | ");

switch (WiFi.encryptionType(i)) {

case WIFI\_AUTH\_OPEN: Serial.print("open"); break;

case WIFI\_AUTH\_WEP: Serial.print("WEP"); break;

case WIFI\_AUTH\_WPA\_PSK: Serial.print("WPA"); break;

case WIFI\_AUTH\_WPA2\_PSK: Serial.print("WPA2"); break;

case WIFI\_AUTH\_WPA\_WPA2\_PSK: Serial.print("WPA+WPA2"); break;

case WIFI\_AUTH\_WPA2\_ENTERPRISE: Serial.print("WPA2-EAP"); break;

case WIFI\_AUTH\_WPA3\_PSK: Serial.print("WPA3"); break;

case WIFI\_AUTH\_WPA2\_WPA3\_PSK: Serial.print("WPA2+WPA3"); break;

case WIFI\_AUTH\_WAPI\_PSK: Serial.print("WAPI"); break;

default: Serial.print("unknown");

}

Serial.println();

delay(10);

}

}

Serial.println("");

// Delete the scan result to free memory for code below.

WiFi.scanDelete();

// Wait a bit before scanning again.

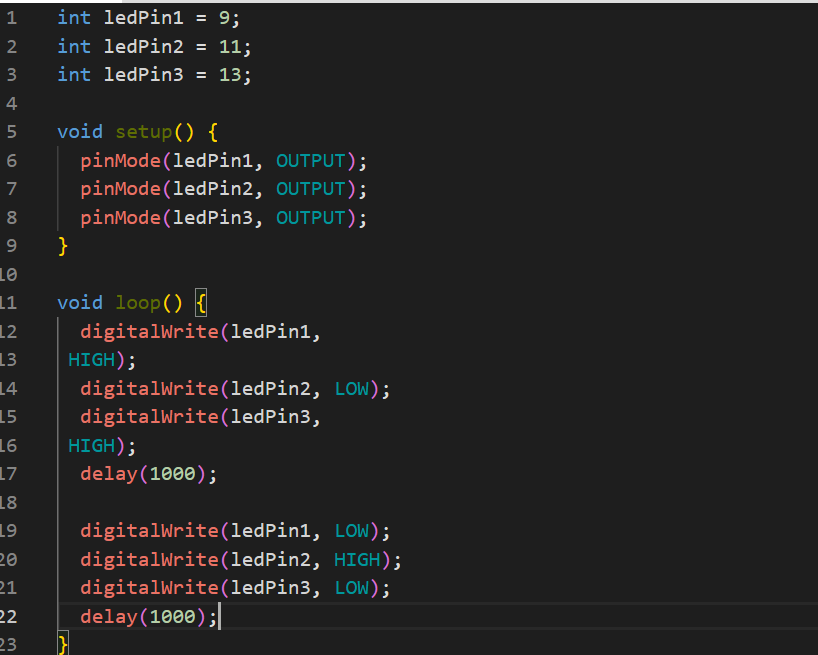
delay(5000);

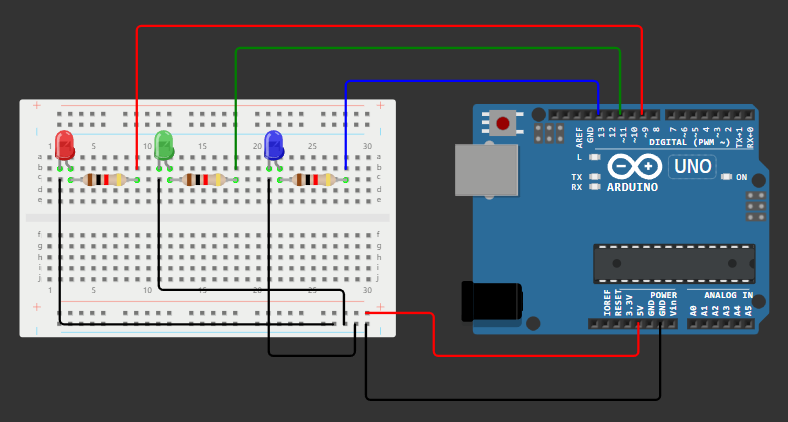
}

**Output: *n Networks Found! ; Light's Space Stone ; Airtel77***

**Conclusion:** Through this experiment we understood how to work with ESP32 and blink the internal LED ,Configuring ESPressif to scan wireless signals.

**Output Images:**





***Experiment No.:8***

**Aim of the experiment:**Use of ESP32, upload code on ESP 32 to blink onboard LED  
**Requirements:**

* **ESP32 Board**: The primary microcontroller for the lab.
* **USB Cable**: To connect the ESP32 to the computer for programming.
* **Arduino IDE**: Software environment for writing, uploading, and testing the code.

**Procedure:**The ESP32 is a popular microcontroller known for its low-power capabilities, WiFi, and Bluetooth connectivity, making it ideal for IoT applications. In this lab, the ESP32’s onboard LED will be programmed to blink at 1-second intervals. The LED is commonly connected to GPIO pin 2 on most ESP32 boards.

**ESP32 LED Blinking Program**

The following code will make the onboard LED blink on and off every second:

#define LED\_PIN 2 // GPIO2 is where the internal LED is connected on most ESP32 boards

Code:

void setup() {

  pinMode(LED\_PIN, OUTPUT); // Set the LED pin as an output

}

void loop() {

  digitalWrite(LED\_PIN, HIGH); // Turn on the LED

  delay(1000); // Wait for 1 second (1000 milliseconds)

  digitalWrite(LED\_PIN, LOW); // Turn off the LED

  delay(1000); // Wait for 1 second (1000 milliseconds)

}

**Setup the Arduino IDE for ESP32**

1. **Install ESP32 Board**: In the Arduino IDE, go to **File > Preferences**. In the "Additional Board Manager URLs" field, add:

https://dl.espressif.com/dl/package\_esp32\_index.json

1. Go to **Tools > Board > Boards Manager**, search for "ESP32," and install the **ESP32 by Espressif Systems** package.

**Connect ESP32 and Upload Code**

1. Select your ESP32 board in **Tools > Board**.
2. Select the correct COM port in **Tools > Port**.
3. Paste the provided code into the Arduino IDE editor.
4. Click the **Upload** button. The code will be compiled and uploaded to the ESP32.

**Output:** Eperiment Link **:[Wokwi](https://wokwi.com/projects/413193606617609217)**



**Conclusion:**

This lab provided hands-on experience with setting up the ESP32 in the Arduino IDE and writing a simple program to control the onboard LED. Blinking an LED is a foundational step in microcontroller programming, as it allows users to test the basic operation and configuration of the device. The ESP32’s onboard LED responded as expected, indicating a successful setup and execution.