IoT & Automation Lab. Record

**Lab#1**

**Blinking the InBuilt LED**

void setup() {

pinMode(LED\_BUILTIN, OUTPUT);

}

void loop() {

digitalWrite(LED\_BUILTIN, HIGH);

delay(500);

digitalWrite(LED\_BUILTIN, LOW);

delay(500);

****}

Figure 1 Uno R3: InBuilt LED Blink

[**Wokwi Link #1.0**](https://wokwi.com/projects/406490864427470849)

**Lab#2**

**Blinking an External LED ( Red ) w/ Resistor**

#define light 12

void setup() {

  pinMode(light, OUTPUT);

}

void loop() {

  digitalWrite(light, HIGH);

  delay(200);

  digitalWrite(light, LOW);

  delay(500);

  }

****

Figure 2 Uno R3: External LED Blink

[**Wokwi Link #1**](https://wokwi.com/projects/406483824958897153)

**Lab#3**

**Using a Digital Humidity & Temperature Sensor**

#include <DHT.h>

#define light 7

#define DHTTYPE DHT22

DHT dht(light, DHTTYPE);

float humid, temp;

void setup() {

  Serial.begin(9600);

  dht.begin();

}

void loop() {

Figure 3 Uno R3: DHT22

  delay(200);

  humid = dht.readHumidity();

  temp = dht.readTemperature();

  Serial.print("Humidity: ");

  Serial.print(humid);

  Serial.print(" %  Temperature: ");

  Serial.print(temp);

  Serial.println("°C");

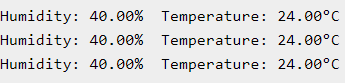
  delay(1000);

Figure 4 Wokwi Output [ DHT Sensor ]

}

[**Wokwi Link #2**](https://wokwi.com/projects/406487282785269761)

**Lab#4**

**Configuring MQTT Service in my Machine**

* **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**

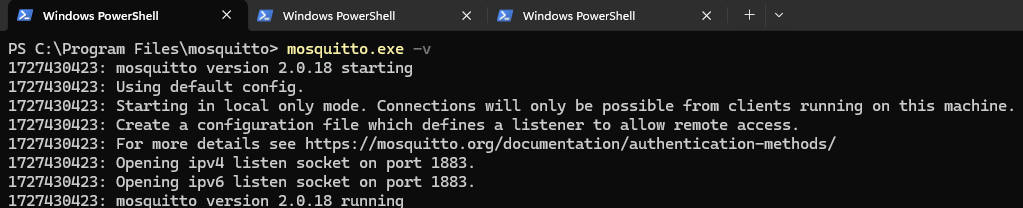
 *// -v is a Verbose Output flag, that enables us to see the backend processes, log messages, that'd help us to debug whenever necessary.*

Figure 5 Mosquitto Initialization in WinOS [ Windows 11 ]

* **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:

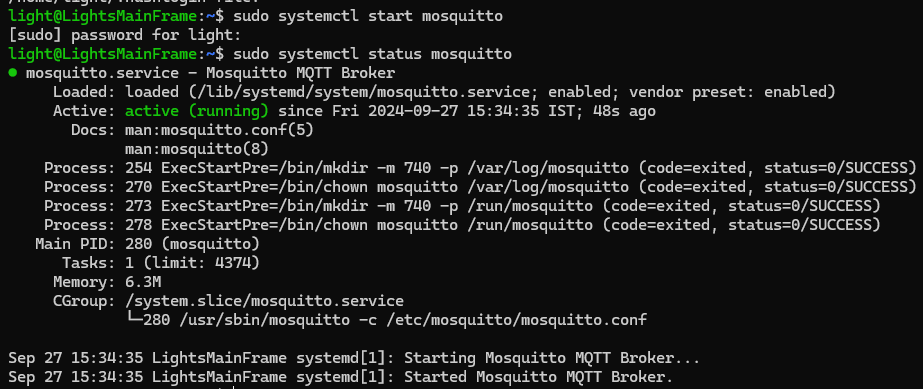
 - **$sudo systemctl status mosquitto**

Figure 6 Mosquitto Initialization in Ubuntu 22.04

- Once verified service status, transmission can be carried on.

* Stopping mosquitto services:

- **$sudo systemctl stop mosquitto**

* **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! "**

\*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/Lokesh Patra/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

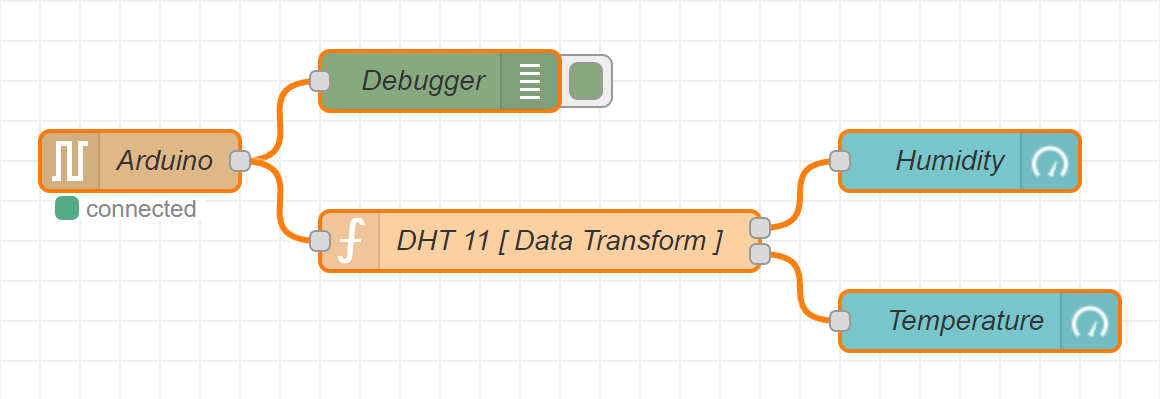
            > 2 GAUGES ( *Humidity & Temperature* )

Figure 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; }**

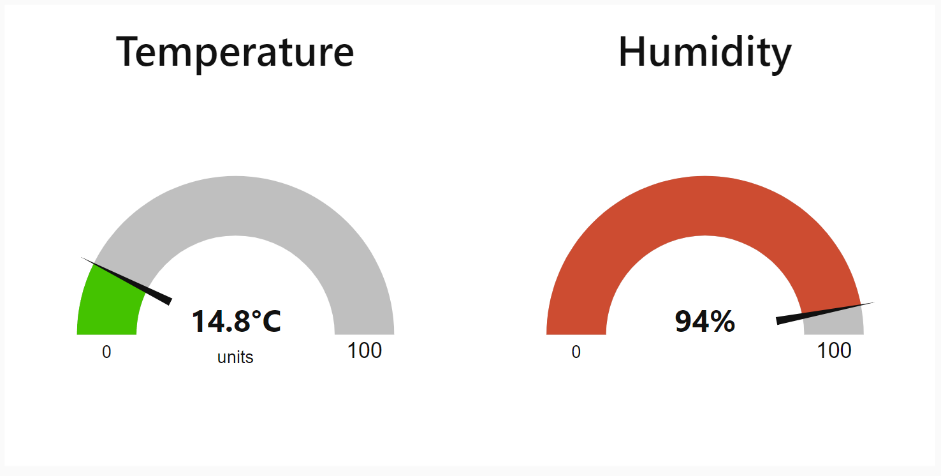
* Adjusting Gauge Nodes:

Figure 10 NodeRED Dashboard [ Temperature + Humidity ]

Humidity:

        - Title as “ **Humidity** ”.

        - Value format as ‘ **{{value}}%** ’.

        - Range Value: 0 ~ 100 %.

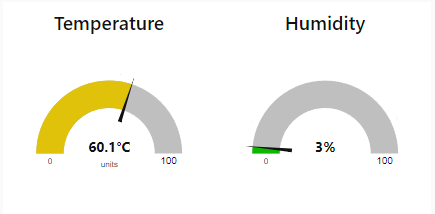
Temperatue:

        - 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();**

Figure 11 Temperature maxxed @ 60.1° & Humidity @ min. (3% )

**}**

**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.

LAB 6: Working With Ultrasonic Sensors: Measuring Distance of a Somethings Using Ultrasonic Sensors

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

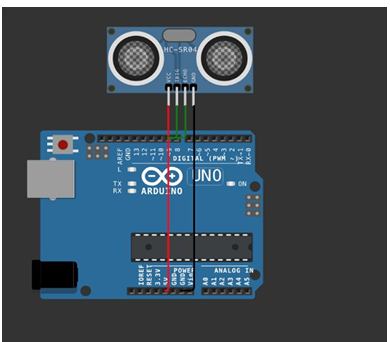
// Calculate distance in inches:

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

LAB 7: Use of Breadboard

Breadboard Usage Guide

A breadboard is an essential tool for building and testing electronic circuits without soldering. It allows you to prototype and experiment with circuits quickly and conveniently.

Key Features:

Rows and Columns: Breadboards are organized into numbered rows and lettered columns. The central rows are connected horizontally, while the outer rows, or power rails, run vertically to help distribute power.

Power Rails: Located along the edges and marked with red (+) and blue (-) lines, power rails are designed to supply power across the board.

How to Use a Breadboard:

Insert Components: Place the leads of each component into the breadboard holes.

Make Connections: Use jumper wires to connect components as needed.

Power Connections: Connect the power source to the power rails to easily distribute power across the circuit.

Avoid Overload: Breadboards are suitable for low-power circuits. High currents can damage the contacts, so avoid overloading them.

Tips:

Plan the Layout: Arrange components and wires to keep the layout organized and clear.

Check Connections: Ensure each wire and component is securely inserted to maintain good connections.

This guide should help you get started with using a breadboard effectively. Happy prototyping!

LAB 8: Use Of Esp32, Upload Code on Esp 32 To Blink Onboard Led

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:

// 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.**

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:

#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**