**IoT & Automation Lab Record**

**Lab-1**

**Lab Exercise: Blinking the In-Built LED**

**Step 1: Connect and Set Up the Arduino**

1. **Connect the Arduino to Your Computer**: Use a USB cable to connect the Arduino board to your computer. The Arduino’s onboard LED is connected to *pin 13*, so no external components are required.
2. **Open the Arduino IDE**: Launch the Arduino IDE on your computer.

**Step 2: Write the Code**

1. **Open a New Sketch**: In the Arduino IDE, open a new sketch by going to *File > New*.
2. **Enter the Code**: Type the following code, which will turn the LED on and off at one-second intervals:

void setup() {

pinMode(LED\_BUILTIN, OUTPUT);

}

void loop() {

digitalWrite(LED\_BUILTIN, HIGH);

delay(1000);

digitalWrite(LED\_BUILTIN, LOW);

delay(1000);

}



**Step 3: Select the Board and Port**

1. Choose the Board: Go to *Tools > Board* and select your Arduino model (e.g., *Arduino Uno*).
2. Select the Port: Go to *Tools > Port* and choose the port your Arduino is connected to. This is usually labeled with "Arduino" and a COM number (e.g., COM3 on Windows or /dev/cu.usbmodem on macOS).

**Step 4: Upload the Code**

1. Verify the Code: Click the *Verify* button (the checkmark icon) to compile the code and check for any errors.
2. Upload the Code: Click the *Upload* button (the right-arrow icon) to upload the code to your Arduino.

**Step 5: Observe the In-Built LED Blinking**

Once the upload is complete, the onboard LED should start blinking on and off every second. The LED will turn on for one second, then off for one second, in a continuous loop**.**

[**Wokwi Lab-1 @ project**](https://wokwi.com/projects/408548736050079745)

**Lab-2**

**Lab Exercise: Blinking an External Red LED**

**Components Required**

* 1 x Red LED
* 1 x 220Ω Resistor (to limit the current and protect the LED)
* Jumper wires
* Breadboard (optional, but makes it easier to connect the components)
* Arduino board (e.g., Arduino Uno)

**Step 1: Connect the LED and Resistor to the Arduino**

1. **Identify the LED Pins:**
   * The anode (longer leg) is the positive terminal and should connect to the Arduino output pin.
   * The cathode (shorter leg) is the negative terminal and should connect to ground.
2. **Connect the Resistor:**
   * Insert the resistor into the breadboard or directly connect it to the anode (longer leg) of the LED. Connect the other end of the resistor to a row that you’ll connect to *pin 9* on the Arduino (or any digital pin you choose to use).
3. **Connect the Cathode (Ground):**
   * Insert the LED’s cathode (shorter leg) into a row connected to the Arduino ground (GND).
4. **Use Jumper Wires to Connect:**
   * Use a jumper wire to connect the row with the resistor to *pin 9* on the Arduino.
   * Use another jumper wire to connect the row with the LED’s cathode to the Arduino’s *GND* pin.

**Step 2: Write the Code**

1. Open the Arduino IDE: Launch the Arduino IDE on your computer.
2. Enter the Code: Use the following code to blink the external LED on and off:

#define k 12

void setup() {

  pinMode(k, OUTPUT);

}

void loop() {

  digitalWrite(k, HIGH);

  delay(500);

  digitalWrite(light, LOW);

  delay(500);

  }





**Step 3: Select the Board and Port**

1. Select the Board: Go to *Tools > Board* and choose your Arduino model (e.g., *Arduino Uno*).
2. Choose the Port: Go to *Tools > Port* and select the port where the Arduino is connected.

**Step 4: Upload the Code**

1. Upload the Code: Click the *Upload* button (right-arrow icon) to upload the code to your Arduino.

**Step 5: Observe the External LED Blinking**

Once the upload is complete, the external LED should start blinking on and off every second. The LED will turn on for one second, then off for one second, repeating in a loop.

[**Wokwi Lab-2 @ Sketch**](https://wokwi.com/projects/408550788966095873)

**Lab-3**

**Lab Exercise: Reading and Displaying Humidity and Temperature Data**

**Components Required**

* 1 x DHT11 or DHT22 sensor
* 1 x 10kΩ resistor (for DHT22, optional for DHT11)
* Jumper wires
* Breadboard (optional)
* Arduino board (e.g., Arduino Uno)

**Step 1: Install the DHT Library in the Arduino IDE**

1. Open the Arduino IDE.
2. Go to *Sketch > Include Library > Manage Libraries...*
3. In the Library Manager, search for “DHT” and install the DHT sensor library by Adafruit.
4. Also, install the Adafruit Unified Sensor library, as it's a dependency for the DHT library.

**Step 2: Wire the DHT Sensor to the Arduino**

1. Identify the Pins on the DHT Sensor:
   * Most DHT11 or DHT22 sensors have three or four pins: *VCC*, *DATA*, *(NC or nothing)*, and *GND*.
2. Make Connections:
   * Connect the *VCC* pin on the sensor to the *5V* pin on the Arduino.
   * Connect the *GND* pin on the sensor to the *GND* pin on the Arduino.
   * Connect the *DATA* pin on the sensor to *digital pin 2* on the Arduino.
3. **Add a Resistor (if needed):**
   * For DHT22, place a 10kΩ pull-up resistor between the *VCC* and *DATA* pin. This is optional for DHT11 but improves signal stability.

**Step 3: Write the Code**

1. Open the Arduino IDE and create a new sketch.
2. Enter the Code: Use the following code to read and display temperature and humidity data.

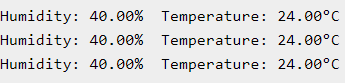
#include <DHT.h>

#define k 7

#define DHTTYPE DHT22

DHT dht(k, 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);

}

[**wokwi Lab-3 @ Sketch**](https://wokwi.com/projects/408552551823923201)

**Lab 4: Using Mosquitto MQTT (Pub-Sub):**

**Step 1: Installing and Configuring Mosquitto as an MQTT Broker**

1. Download and Install Mosquitto:
   * Go to the [official Mosquitto download page](https://mosquitto.org/download/).
   * Download the Windows installer and run the installation.
   * During installation, Mosquitto will be added as a Windows service, allowing it to run in the background automatically.
2. Add Mosquitto to the System PATH:
   * Open *System Properties* and go to *Environment Variables*.
   * Locate and edit the Path variable to include C:\Program Files\mosquitto.
   * Adding Mosquitto to the PATH allows you to run mosquitto commands directly from the Command Prompt.
3. Starting and Stopping Mosquitto as a Service:
   * To start Mosquitto manually, open Command Prompt with administrator privileges and enter:

bash

net start mosquitto

* + To stop the service, run:

bash

net stop mosquitto

1. Running Mosquitto with Verbose Logging:
   * Navigate to the Mosquitto installation directory to start the broker with verbose logging:

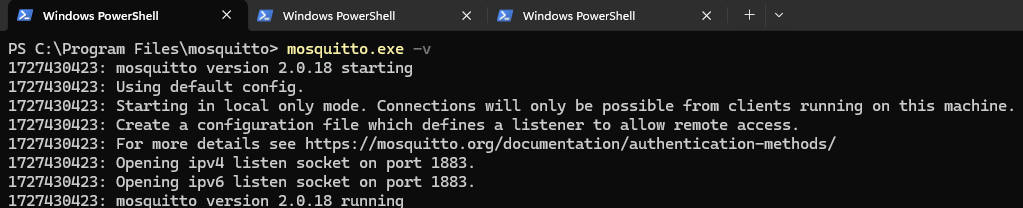
bash

Copy code

cd "C:\Program Files\mosquitto"

mosquitto.exe -v

* + The -v flag provides detailed output on the broker’s activities, which is useful for debugging.



* **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

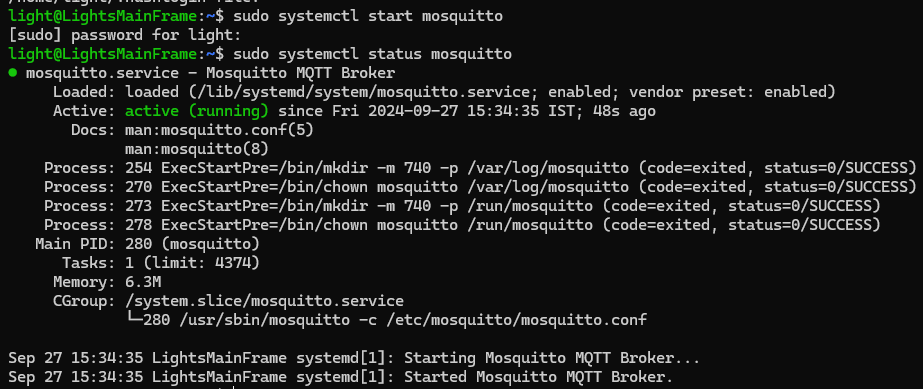
- **$sudo apt update**

- **$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**

**Step 2: Setting Up Publisher and Subscriber (Pub/Sub) on Windows**

1. **Open Command Prompt**:
   * After adding Mosquitto to the PATH, open a Command Prompt or PowerShell window to run MQTT commands.
2. **Start a Subscriber**:
   * The subscriber will listen to a topic for messages. In the Command Prompt, enter:

bash

Copy code

mosquitto\_sub -h localhost -t "test/topic"

* + Here, -h localhost specifies the broker’s address (your local machine) and -t "test/topic" specifies the topic to subscribe to.

1. **Start a Publisher**:
   * Open another Command Prompt window, and enter:

bash

Copy code

mosquitto\_pub -h localhost -t "test/topic" -m "Hello MQTT"

* + This command sends the message "Hello MQTT" to the test/topic. Any subscribers to this topic will receive the message.

**Step 3: Accessing Mosquitto Broker from a Linux Environment**

1. **Locate the IP Address of Your Windows Machine**:
   * Find your machine’s IP by running the following command in Command Prompt:

ipconfig

Note the IP address under your network connection (usually 192.168.x.x).

**Connect to the Broker from Linux**:

In a Linux environment with Mosquitto installed, use the mosquitto\_sub and mosquitto\_pub commands to connect to the broker running on your Windows machine.

To subscribe, use:

*mosquitto\_sub -h <Windows\_IP> -t "test/topic"*

* + To publish a message from Linux to the Windows broker:

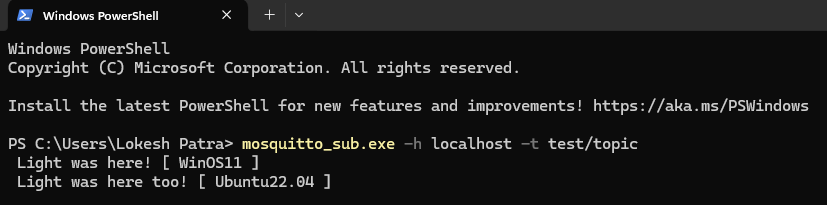
*mosquitto\_pub -h <Windows\_IP> -t "test/topic" -m "Hello from Linux"*

* + Replace <Windows\_IP> with your Windows machine’s IP address.

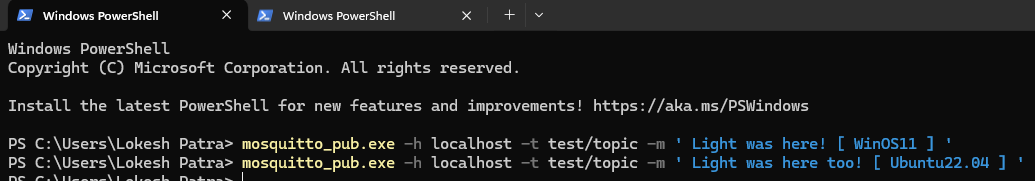
**Step 4: Testing and Troubleshooting**

* **Verify Broker Activity**:
  + In the Command Prompt on your Windows machine, use mosquitto.exe -v to view incoming connections, message activity, and potential issues.
* **Firewall Considerations**:
  + Ensure that your Windows firewall allows connections on port **1883**, the default port for MQTT. Configure this by adding a firewall rule for the Mosquitto executable or allowing inbound connections on port 1883.
* **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: Building a web app using Node-Red to fetch DHT sensor data and display it on the web app dashboard**

**Installing & Initialising node red:**

* Open node.js > npm install node-red-dashboard
* [postinstallation] > elevated cmd: node-red

**In client application, browsed localhost:1880 [ accessing node red]:**

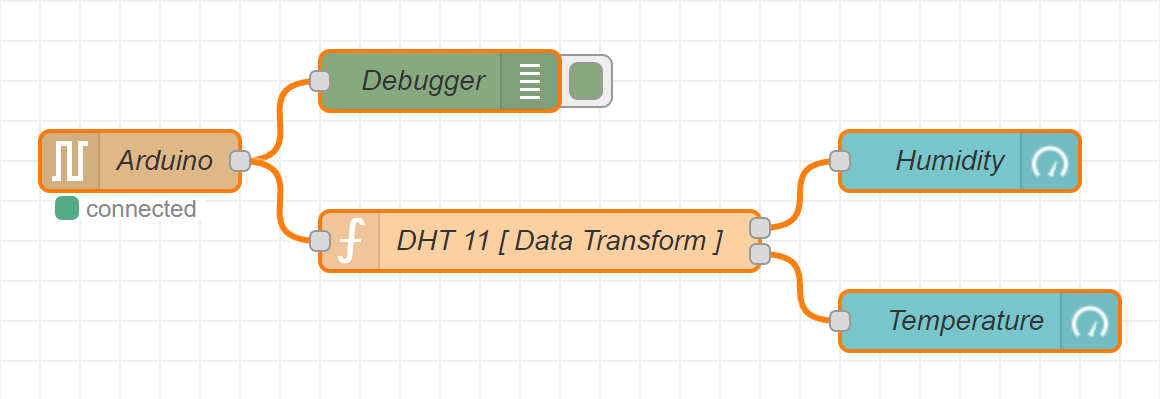
* Inside the nodered window, a flow was created w/ the nodes as:

            > 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., 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:

**Code:**

#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));

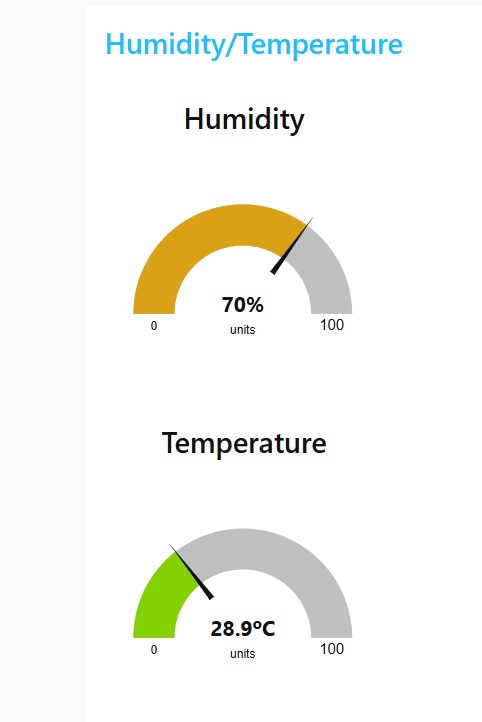
}

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 ON THE DASHBOARD:**

****

**LAB 6: Working with ultrasonic sensors .**

Measuring distance of a somethings using ultrasonic sensors

**Step 1: Installing the Arduino IDE and Selecting the Board**

To get started with programming the Arduino Uno, you first need the Arduino IDE. You can download it from the official Arduino website (https://www.arduino.cc/en/software) and install it on your computer. Once installed, open the Arduino IDE.

In the Arduino IDE, you’ll need to select the correct board type. Go to *Tools > Board* and choose *Arduino Uno* from the list. Selecting the board tells the IDE which type of Arduino you’re using and helps it compile code properly for that specific model.

**Step 2: Connecting the Arduino Uno to Your Computer and Selecting the Port**

Connect your Arduino Uno to your computer using a USB cable. After connecting, you’ll need to select the correct port in the Arduino IDE so that the computer can communicate with the board. Go to *Tools > Port* and look for a port labeled with something like "COM3" (on Windows) or "/dev/cu.usbmodem" (on macOS), often with "Arduino Uno" next to it. Select this port.

**Step 3: Writing or Selecting Code**

You can now write your code or open an existing sketch. For example, to test your setup, open the "Blink" example that comes with the Arduino IDE. Go to *File > Examples > 01.Basics > Blink*. This program will make the onboard LED blink on and off every second.

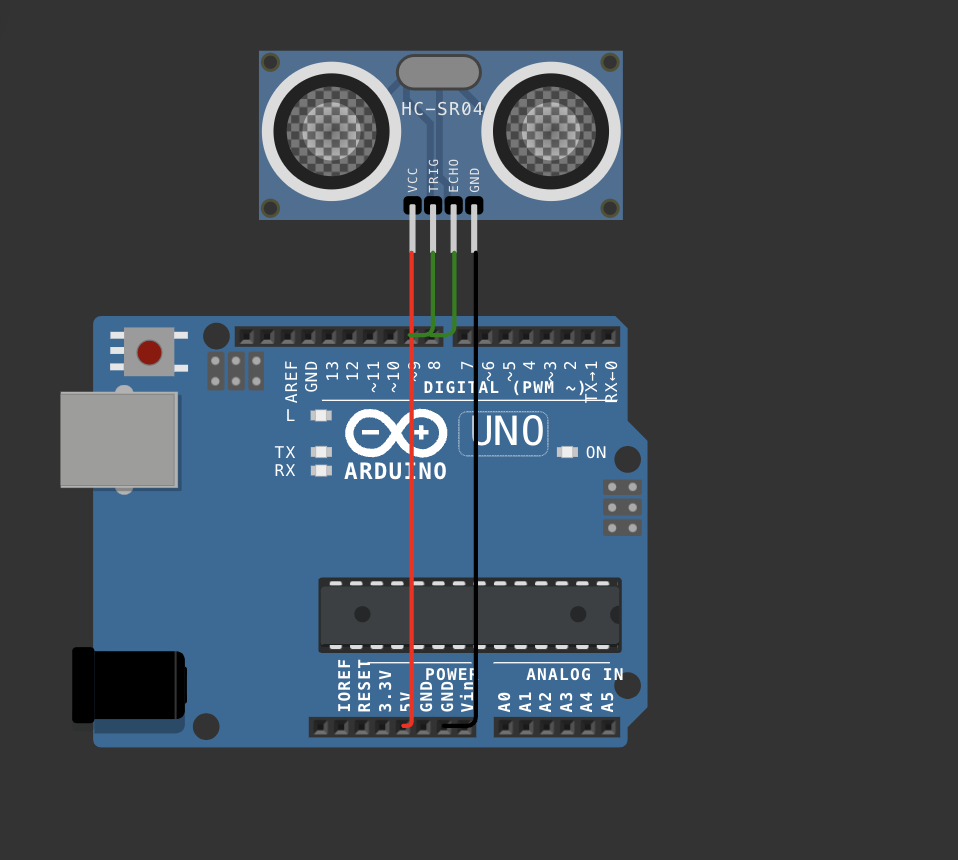
**Step 4: Compiling and Uploading the Code**

Once your code is ready, the next step is to compile and upload it to the Arduino Uno. First, click the *Verify* button (the checkmark icon) to compile your code. This process checks for any syntax errors or other issues in your code.

Once the code is verified, you can upload it to the Arduino Uno by clicking the *Upload* button (the right-pointing arrow icon). You should see the LED on the board blink rapidly as the code is uploaded. When finished, the board will reset, and the code will start running—if you’re using the Blink example, the onboard LED should start blinking at one-second intervals.

**Step 5: Testing and Running the Code**

After uploading, observe your Arduino Uno to ensure that the code is running as expected. If you’re running the Blink example, the built-in LED should now blink on and off, confirming that your setup is working. From here, you can experiment with other code examples or create your own sketches to explore more functionalities.



**CODE:**

#include <DHT.h>

#define PIN\_TRIG 9

#define PIN\_ECHO 8

void setup() {

Serial.begin(9600);

pinMode(PIN\_TRIG, OUTPUT);

pinMode(PIN\_ECHO, INPUT);

}

void loop() {

digitalWrite(PIN\_TRIG, HIGH);

delayMicroseconds(10);

digitalWrite(PIN\_TRIG, LOW);

int duration = pulseIn(PIN\_ECHO, HIGH);

float distanceCm = duration / 58.0;

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

Serial.println(distanceCm);

delay(1500);

}

**Lab#7 Use of Breadboard**

A breadboard is an essential tool for building and testing electronic circuits without soldering. It allows you to connect components easily by inserting their leads into holes connected by conductive strips. Here’s a detailed overview of how a breadboard works and the steps to connect it with an Arduino board.

**Understanding the Breadboard Layout**

A standard breadboard has rows and columns of small holes used to insert electronic components. The layout typically includes:

1. **Power Rails**: These are the long rows on either side of the breadboard marked with “+” and “-” symbols, commonly used for power distribution. They run vertically and are used to provide power and ground connections to multiple components in the circuit.
2. **Terminal Strips**: These are the short rows in the central area where you insert the components. Each row consists of five connected holes, allowing components to share connections horizontally across the row. However, these strips are isolated from each other, so the left side and right side of the breadboard are not connected by default.
3. **Gap or Divider**: In the middle of the breadboard, there’s usually a gap or divider. This separation is useful for placing integrated circuits (ICs), which straddle the gap, allowing each pin to connect to its own row.

**How to Use a Breadboard with an Arduino: Step-by-Step Guide**

**Step 1: Connect Power and Ground from the Arduino to the Breadboard**

1. **Identify the Arduino Power Pins**: Locate the *5V* and *GND* (ground) pins on the Arduino.
2. **Connect Power to the Breadboard**: Using jumper wires, connect the *5V* pin on the Arduino to the positive (+) power rail on the breadboard. This rail will now serve as the power source for your components.
3. **Connect Ground to the Breadboard**: Similarly, connect the *GND* pin on the Arduino to the negative (-) power rail on the breadboard. This establishes a common ground connection for all components connected to the ground rail.

**Step 2: Adding Components to the Breadboard**

1. **Insert Components**: Place components (such as LEDs, resistors, sensors, or other electronic parts) into the terminal strip rows. For example, you might place one leg of an LED in a row connected to an output pin on the Arduino and the other leg in a row connected to ground through a resistor.
2. **Use Resistors with LEDs**: To prevent an LED from drawing too much current, always use a resistor in series with the LED. Connect one end of the resistor to the LED’s anode (positive leg) and the other end to the row where you plan to connect the Arduino output.

**Step 3: Connect Components to the Arduino Using Jumper Wires**

1. **Set Up Signal Connections**: Using jumper wires, connect each component to the appropriate Arduino pins according to your circuit design. For example, if you’re controlling an LED, connect one wire from an Arduino digital output pin (e.g., *pin 9*) to the row on the breadboard where the LED’s resistor is connected.
2. **Check Connections**: Ensure all components are connected correctly. The positive rail should connect to *5V*, the negative rail to *GND*, and components should have their designated pins connected to the correct Arduino pins.

**Step 4: Powering the Circuit and Running Code**

1. **Power the Arduino**: Connect the Arduino to your computer via USB, or use an external power source. This will power both the Arduino and the connected breadboard circuit through the *5V* and *GND* connections.
2. **Upload Code to the Arduino**: In the Arduino IDE, write or upload a sketch that controls your components. For instance, if you connected an LED to *pin 9*, you could write a program that makes it blink on and off.
3. **Test and Troubleshoot**: Once the code is uploaded, your circuit should begin functioning. If the circuit doesn’t work as expected, double-check connections and code.

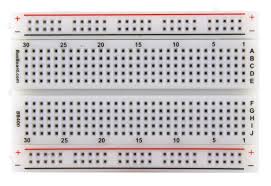
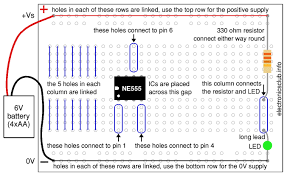
**Example: Simple LED Circuit on a Breadboard**

1. **Place the LED**: Insert the LED into the breadboard with its anode (longer leg, positive) connected to a row.
2. **Add a Resistor**: Connect a resistor (220Ω to 330Ω) from the LED anode’s row to another row, where it will connect to an Arduino output pin.
3. **Connect to Arduino**: Use a jumper wire to connect the row with the resistor to *pin 9* on the Arduino. Connect the LED’s cathode (shorter leg, negative) to the ground rail, which is connected to the Arduino’s *GND*.
4. **Upload Code**: Write a simple program to turn on *pin 9* and make the LED blink.

Using a breadboard with an Arduino enables easy experimentation and modifications to circuits, making it ideal for prototyping and testing.

Top of Form

Bottom of Form

**Lab#8 Use of ESP32, upload code on ESP 32 to blink onboard LED**

**Step 1: Installing the ESP32 Board from the Board Manager**

To get started with programming the ESP32, you need to install the appropriate board files in the Arduino IDE. First, open your Arduino IDE and navigate to *File > Preferences*. In the "Additional Boards Manager URLs" field, add this link: https://dl.espressif.com/dl/package\_esp32\_index.json. This link will allow the Arduino IDE to download and install the necessary ESP32 files.

Next, go to *Tools > Board > Boards Manager* and type "ESP32" into the search bar. You should see the "ESP32 by Espressif Systems" option in the list. Select this option and click the *Install* button. This process will download all the relevant files for the ESP32 platform. Once installation is complete, you can program various ESP32 models directly from the Arduino IDE.

**Step 2: Selecting the ESP32 Board and Port**

Once you’ve installed the ESP32 board files, you need to choose the specific model of the ESP32 you’re working with. In the Arduino IDE, go to *Tools > Board* and scroll down until you see the ESP32 options. Select the model that matches your board (for instance, “ESP32 Dev Module” is a common choice).

After selecting the board, you need to assign the correct communication port so the IDE can connect to the ESP32. Connect your ESP32 board to your computer via USB, then go to *Tools > Port*. Here, you should see a COM port listed, which represents the USB port where the ESP32 is connected. Select this port to establish a connection with the board.

**Step 3: Compiling and Uploading the Code**

With the board and port selected, you’re now ready to write, compile, and upload code to your ESP32. Write or open an Arduino sketch you’d like to upload. To make sure your code compiles without errors, click on the *Verify* button in the Arduino IDE. This will check for syntax errors or any issues in your code and compile it.

Once verified, you can upload the code to the ESP32 by pressing the *Upload* button (a right-pointing arrow icon in the toolbar). During the upload, you might see the LED on the ESP32 flashing, indicating that it’s receiving data. Some ESP32 models require you to press a "BOOT" button during the initial seconds of the upload process to put the board into programming mode. Once the upload is complete, the ESP32 will automatically restart, and your code should begin executing.

Following these steps will enable you to program and work with your ESP32 board easily. Once you've completed these setup steps, you can experiment with different code examples and features that take advantage of the ESP32's capabilities. Now let’s write the codes and test it out.

First for blinking inbuilt LED

Second for finding nearby WIFI using ESP 32 boards

**Code 1 : blinking inbuilt led**

#define LED\_PIN 2

void setup() {

pinMode(LED\_PIN, OUTPUT);

}

void loop() {

digitalWrite(LED\_PIN, HIGH);

delay(1000);

digitalWrite(LED\_PIN, LOW);

delay(1000);

}

**Code 2 : Finding nearby WIFI using esp32**

#include "WiFi.h"

void setup() {

Serial.begin(9600);

WiFi.mode(WIFI\_STA);

WiFi.disconnect();

delay(100);

Serial.println("Setup done");

}

void loop() {

Serial.println("Scan start");

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) {

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(“");

WiFi.scanDelete();

delay(5000);

}