**INTERNET OF THINGS LAB RECORD**

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

|  |  |
| --- | --- |
| Name: | Rudra Prasanna Mishra |
| Registration Number: | FET-BAML-2022-2026-032 |
| Course: | Btech CSE AIML |
| Semester: | 5th |
| Faculty: | Mr. Biswajeeban Mishra &  Mr. 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 No.:1***

***Description:***

The internal LED on the Arduino (usually connected to pin 13) is a useful component for testing basic programming concepts. In this experiment, we will write a program that makes the internal LED blink on and off with a delay.

***Code:***

// Pin number for the internal LED on most Arduino boards

int ledPin = 13;

void setup() {

// Initialize the internal LED as an output

pinMode(ledPin, OUTPUT);

}

void loop() {

// Turn the LED on (HIGH is the voltage level)

digitalWrite(ledPin, HIGH);

// Wait for 1 second

delay(1000);

// Turn the LED off by making the voltage LOW

digitalWrite(ledPin, LOW);

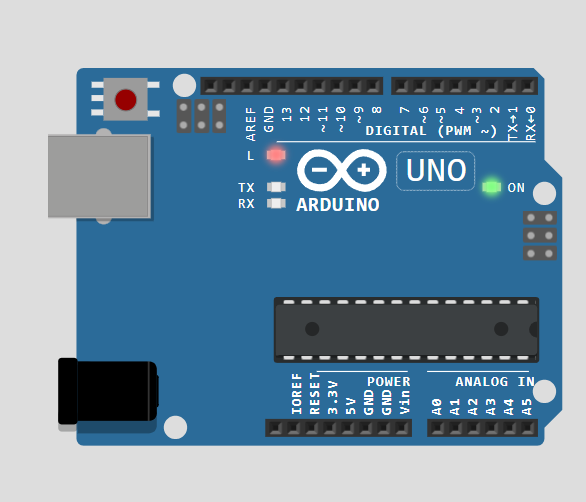
// Wait for 1 second

delay(1000);

}

***Conclusion:***

This experiment demonstrates the basic use of the digitalWrite() and delay() functions in Arduino programming to control the internal LED. It provides a good starting point for understanding how to control outputs and introduces the structure of an Arduino sketch.

******

***Experiment No.:2***

**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. This shows how external components can be controlled using Arduino*.*

***Circuit:***

* *Connect the longer leg (anode) of the LED to pin 8 on the Arduino.*
* *Connect the shorter leg (cathode to the ground (GND) of the Arduino.*

***Code:***

// Pin number for the external LED

int ledPin = 8;

void setup() {

// Initialize the external LED as an output

pinMode(ledPin, OUTPUT);

}

void loop() {

// Turn the LED on

digitalWrite(ledPin, HIGH);

// Wait for 1 second

delay(1000);

// Turn the LED off

digitalWrite(ledPin, LOW);

// Wait for 1 second

delay(1000);

}

***Conclusion:***

This experiment reinforces the concepts of controlling external components with the Arduino. By blinking an external LED, we learned how to work with basic electronic components and establish simple connections using resistors and LEDs.



***Experiment No.:3***

**Description:**

The DHT22 sensor is used to measure both temperature and humidity. This experiment will read these values from the sensor and display them in the serial monitor of the Arduino IDE.

**Circuit:**

* Connect the VCC pin of the DHT22 sensor to the 5V pin on the Arduino.
* Connect the GND pin to the ground (GND) of the Arduino.
* Connect the data pin to pin 2 on the Arduino.

**Code:**

#include "DHT.h"

// Pin to which the DHT22 sensor is connected

#define DHTPIN 2

// Define the type of DHT sensor

#define DHTTYPE DHT22

DHT dht(DHTPIN, DHTTYPE);

void setup() {

// Start the serial monitor at 9600 baud rate

Serial.begin(9600);

// Initialize the DHT sensor

dht.begin();

}

void loop() {

// Wait a few seconds between measurements

delay(2000);

// Read humidity

float humidity = dht.readHumidity();

// Read temperature in Celsius

float temperature = dht.readTemperature();

// Check if any reads failed and exit early (to try again)

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

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

return;

}

// Print the results to the Serial Monitor

Serial.print("Humidity: ");

Serial.print(humidity);

Serial.print(" %\t");

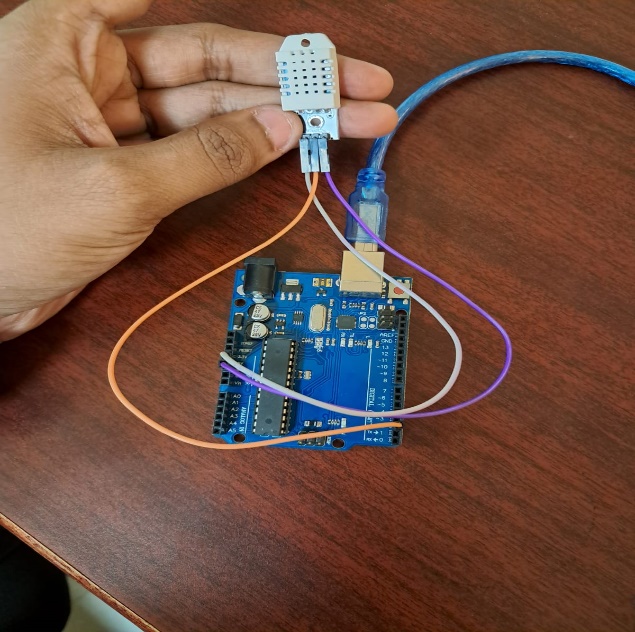
Serial.print("Temperature: ");

Serial.print(temperature);

Serial.println(" °C");

}

**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.

***Experiment No.:4***

**Description:**

A case study on a communication Protocol: MQTT

**Introduction**

This lab explores MQTT, a lightweight messaging protocol commonly used in IoT networks, and Mosquitto, one of its popular implementations. MQTT’s simplicity and efficiency make it ideal for IoT environments where devices have limited resources and network bandwidth. Mosquitto is an open-source MQTT broker that facilitates communication between IoT devices through publish/subscribe mechanisms.

**Understanding MQTT Protocol**

* **Overview**
* MQTT (Message Queuing Telemetry Transport) is a lightweight, publish/subscribe messaging protocol designed for low-bandwidth, high-latency, or unreliable networks.
* It works on a client-broker architecture where clients (publishers and subscribers) communicate through a central broker that manages message distribution.
* **Key Features**
* **Publish/Subscribe Model**: Allows devices to broadcast messages to multiple receivers through a broker.
* **Quality of Service (QoS) Levels**:
  + **QoS 0** - At most once delivery
  + **QoS 1** - At least once delivery
  + **QoS 2** - Exactly once delivery
* **Retained Messages**: Enables storing the last message for each topic.
* **Topics**: Messages are organized by topics, enabling specific message routing.

**Overview of Mosquitto**

* **About Mosquitto**
* Mosquitto is a widely used open-source MQTT broker that supports both standard and encrypted connections, offering a secure and efficient way to transmit data between IoT devices.
* **Popular Usage**
* Commonly implemented in smart home systems, industrial IoT, and real-time data collection systems, Mosquitto is known for its reliability and low power consumption, which makes it ideal for battery-powered devices.

**Installation and Testing**

**Ubuntu Installation**

* Followed the [Vultr Guide](https://docs.vultr.com/install-mosquitto-mqtt-broker-on-ubuntu-20-04-server) for Ubuntu 20.04 installation.
* Steps:
  1. Update the system and install Mosquitto via terminal.
  2. Enable and start the Mosquitto service.
  3. Test by running Mosquitto on localhost and confirming it’s active.

**Windows Installation**

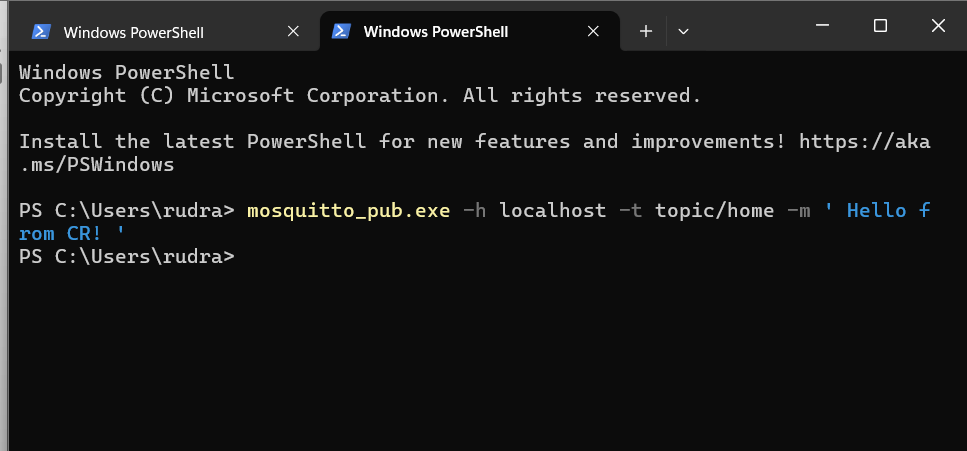
* Followed the [Cedalo Blog](https://cedalo.com/blog/how-to-install-mosquitto-mqtt-broker-on-windows/) instructions.
* Steps:
  1. Download and install Mosquitto on Windows.
  2. Configure Mosquitto to allow message publishing and subscribing.
  3. Test using command line tools provided by Mosquitto.

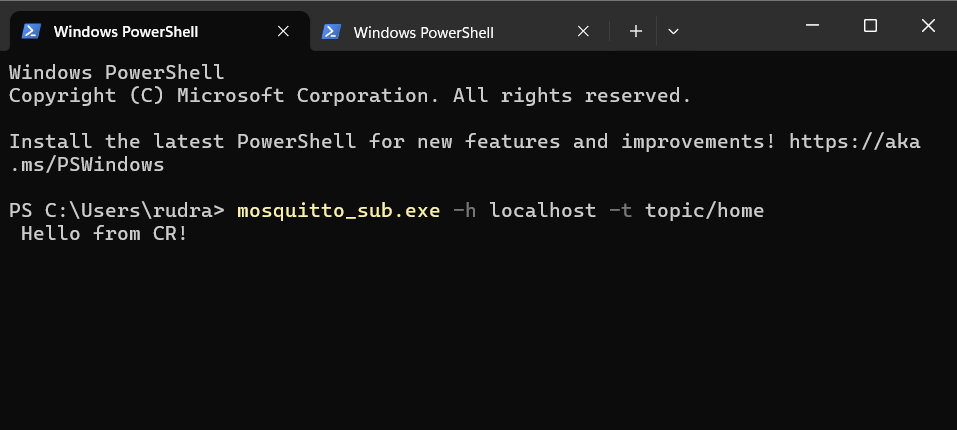
**Testing Mosquitto**

* **Starting the Broker**:
  + Command: mosquitto or mosquitto -v for verbose mode.
  + Verified the broker is running on the default port (1883).
* **Publishing a Message**:
  + Command: mosquitto\_pub -h localhost -t test/topic -m "Hello, MQTT!"
  + Published a message to the topic test/topic and verified it on the subscriber end.
* **Subscribing to a Topic**:
  + Command: mosquitto\_sub -h localhost -t test/topic
  + Observed that the subscribed client received the message "Hello, MQTT!" from the publisher.

**Results**

Successful message transmission between publisher and subscriber via Mosquitto broker.





**Conclusion**

This lab provided hands-on experience with the MQTT protocol through Mosquitto, demonstrating its functionality and efficiency for IoT communication. Mosquitto proved to be an effective broker, capable of handling lightweight communication across different devices, making it ideal for IoT applications where network and power resources are limited.

***Experiment No.:5***

**Description:**

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

**Procedure:  
a. Installation and Setup**

**Install Node.js**

* Download and install Node.js, which is required for running Node-RED. The installation steps depend on the operating system:
  + Download link: [Node.js Download](https://nodejs.org/en/download/package-manager)
  + Verify the installation by running node -v and npm -v in the command line to check for the version numbers.

**Install Node-RED**

* Install Node-RED using the command:

npm install -g --unsafe-perm node-red

* Start Node-RED by running the command node-red in the terminal.
* Access the Node-RED editor by navigating to http://localhost:1880 in a web browser.
* Reference: [Node-RED Installation Guide](https://nodered.org/docs/getting-started/windows)

**Setting Up Node-RED for DHT Sensor Data**

* Use a DHT sensor like the DHT11 or DHT22, which measures temperature and humidity.
* Follow instructions to wire the DHT sensor with Arduino and connect the setup to the system that will run Node-RED.
* Reference Guide for Arduino and DHT Sensor Integration: [Node-RED IoT with Arduino DHT11](https://www.ee-diary.com/2022/07/node-red-iot-with-arduino-dht11.html)

**b. Node-RED Flow Setup**

**Configure Sensor Node**

* Use an Arduino node in Node-RED to read data from the DHT sensor.
* Install any necessary Node-RED libraries or nodes for DHT sensor compatibility:

npm install node-red-contrib-dht-sensor

* In Node-RED, configure the sensor node to receive data from the correct pin on the Arduino. Ensure that the DHT sensor data is set to read both temperature and humidity.

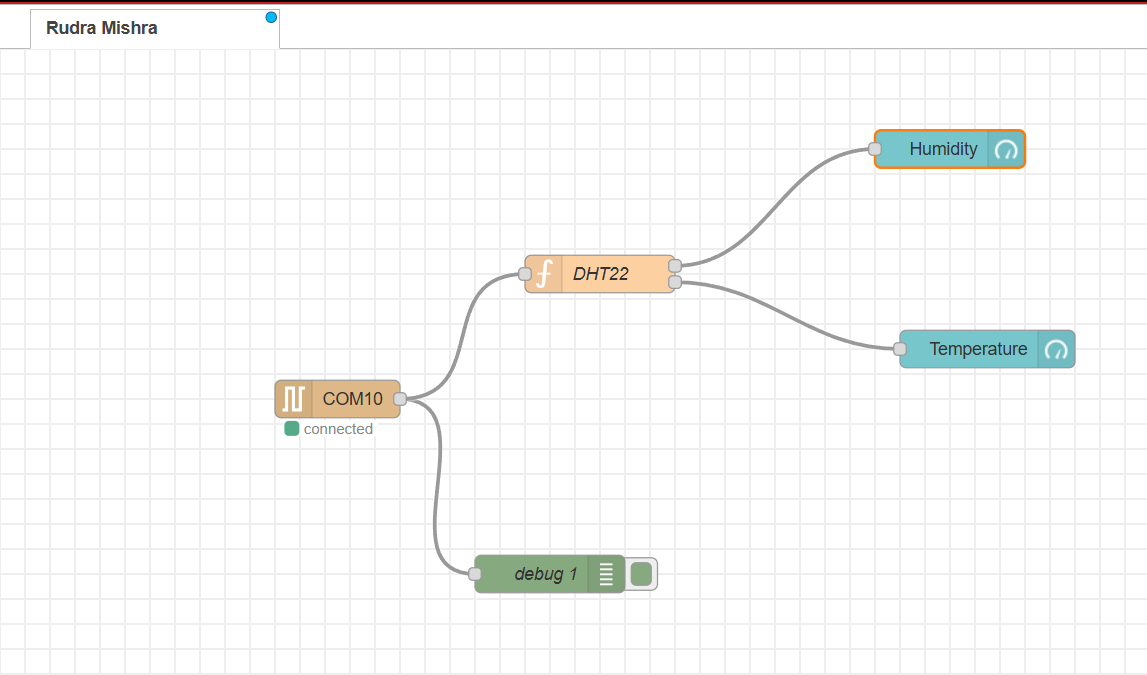
**Create the Web Dashboard**

* Install the Node-RED dashboard package**:**

npm install node-red-dashboard

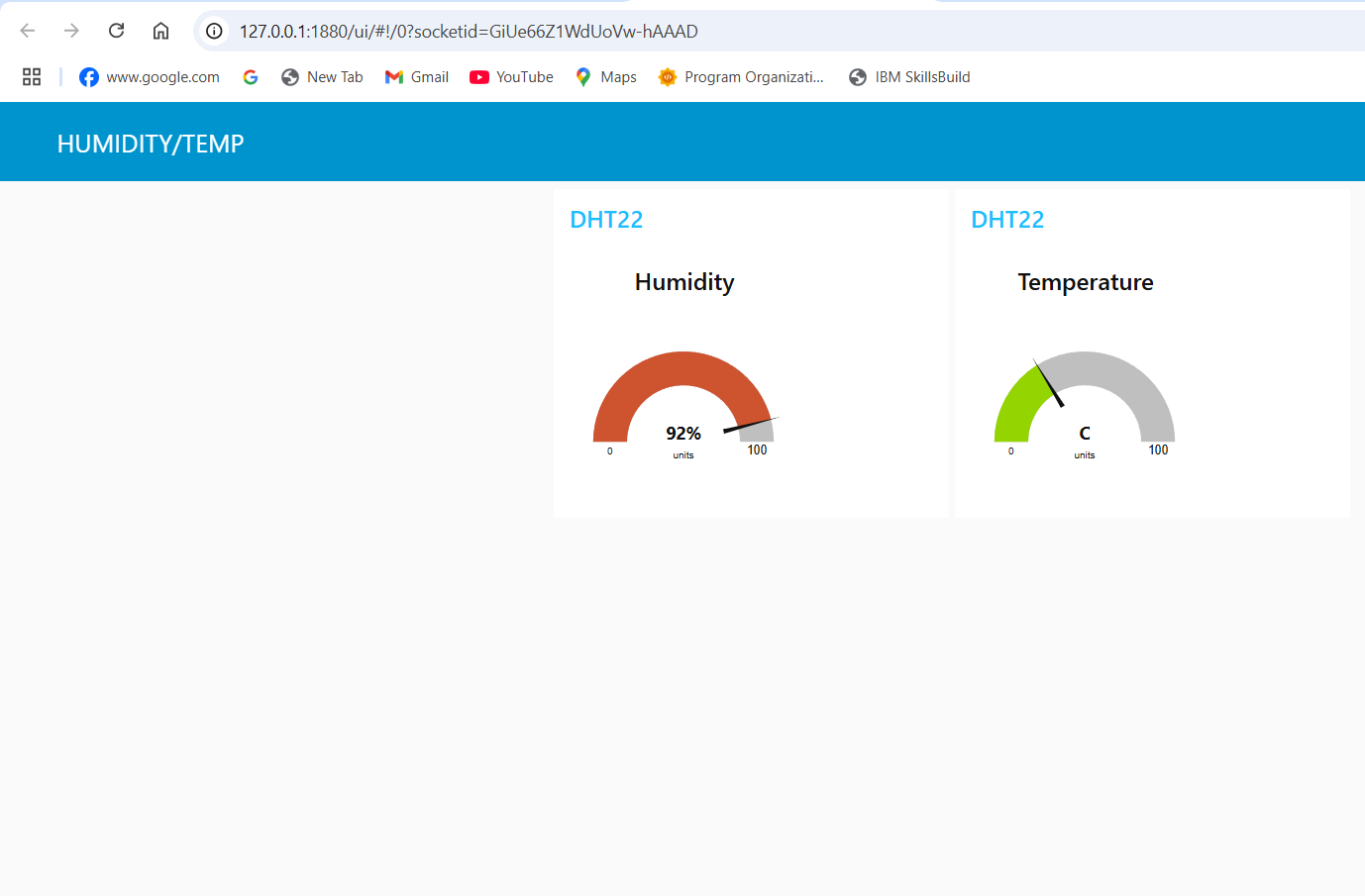
* Create a simple dashboard layout by dragging and dropping UI elements such as gauges and charts onto the flow.
* Configure the dashboard elements to display the temperature and humidity data coming from the DHT sensor node.

**Testing and Troubleshooting**

* Deploy the flow and check if the DHT sensor data is updating correctly on the dashboard.
* If issues arise, refer to the Node-RED troubleshooting guide for DHT sensors:
  + DHT Sensor Troubleshooting: [Node-RED DHT Fix](https://subsequent-friday-236.notion.site/Experiment-6-Real-Time-DHT11-Sensor-Data-Display-on-Node-RED-Dashboard-10716cf4630080cbb588fde5532748fe?pvs=4)
* Test the data flow by observing real-time data changes on the web dashboard as the sensor environment changes (e.g., exposure to warm or cool air).

**Results**

* The Node-RED dashboard successfully displays temperature and humidity data fetched from the DHT sensor.
* Real-time updates are visible on the dashboard, with gauge elements showing the current temperature and humidity levels, and a chart for historical data visualization.



**Conclusion**

* This lab demonstrates how to use Node-RED to develop a simple IoT web application to monitor environmental conditions via a DHT sensor.
* The setup highlights Node-RED's flexibility in IoT applications, especially for real-time data monitoring and easy web visualization.

***Experiment No.:6***

**Description:**Interfacing Ultrasonic Sensor (HC - SR04).  
  
**Procedure:  
Introduction to Ultrasonic Sensing**

Ultrasonic sensing is a technique that uses high-frequency sound waves to detect the distance to an object. The sensor emits ultrasonic pulses and measures the time it takes for the echo to return, calculating distance based on the speed of sound. Ultrasonic sensors are widely used in various applications, such as robotics, automotive systems, and obstacle detection.

**Ultrasonic Ranging Module HC-SR04**

The HC-SR04 is a popular ultrasonic sensor module used for distance measurement. It has two main components: a transmitter that emits an ultrasonic wave and a receiver that detects the returning echo. This module calculates the time it takes for the echo to return to estimate the distance to an object.

**Specifications:**

Operating Voltage: 5V

Measuring Range: 2cm to 400cm

Accuracy: ±3mm

Measuring Angle: 15 degrees

Working Principle:

The HC-SR04 sends out an ultrasonic pulse and listens for its echo. The time delay between sending and receiving the pulse is used to calculate the distance based on the formula:

*Distance = ½ \*Time \* speed of sound*

**Experimental Setup**

**Wiring the HC-SR04 to the Microcontroller**

VCC: Connect to the 5V pin on the Arduino.

GND: Connect to the ground (GND) pin.

Trig: Connect to any digital pin (e.g. 5).

Echo: Connect to another digital pin (e.g. 4).

**Code for Distance Measurement**

const int trigPin = 9;

const int echoPin = 10;

void setup() {

Serial.begin(9600);

pinMode(trigPin, OUTPUT);

pinMode(echoPin, INPUT);

}

void loop() {

digitalWrite(trigPin, LOW);

delayMicroseconds(2);

digitalWrite(trigPin, HIGH);

delayMicroseconds(10);

digitalWrite(trigPin, LOW);

long duration = pulseIn(echoPin, HIGH);

int distance = duration \* 0.034 / 2;

Serial.print("Distance: ");

Serial.print(distance);

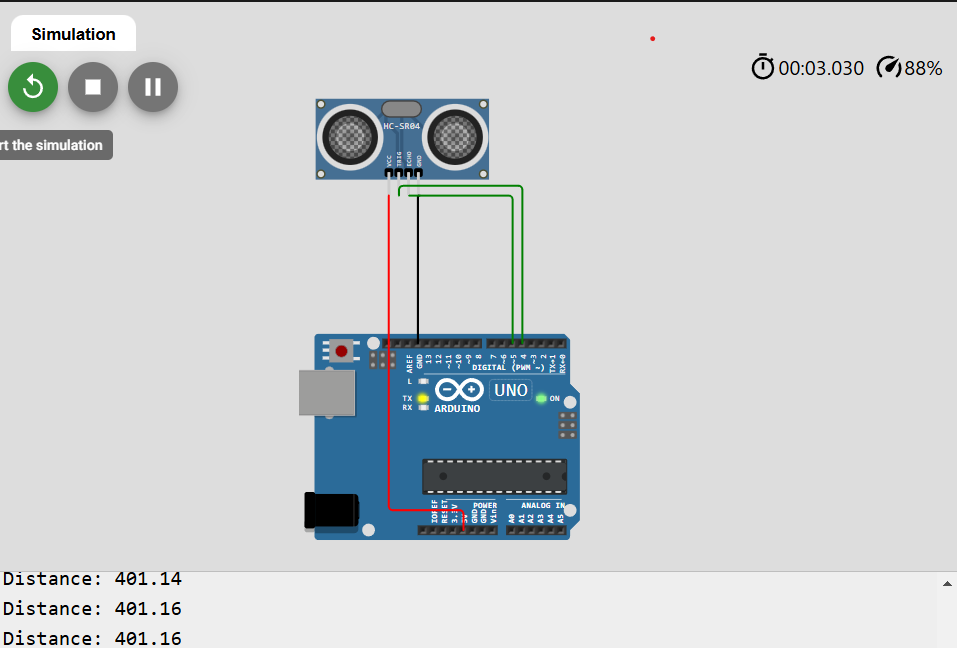
Serial.println(" cm");

delay(500);

}

**Results**

**Experiment Link :** [**Wokwi**](https://wokwi.com/projects/409993842168307713)

****

**Conclusion**

The HC-SR04 ultrasonic sensor effectively measured distances up to 400 cm with reasonable accuracy. This lab reinforced the basics of ultrasonic sensing and the practical use of sensors like the HC-SR04 in distance measurement applications.

***Experiment No.:7***

**Description:**Use of Breadboard  
**Introduction to Breadboards**

A breadboard is a reusable tool used to build and test electronic circuits without soldering. It is commonly used in prototyping because it allows components to be easily connected and removed. Breadboards have a grid of holes in which electronic components and wires can be inserted to create circuits.

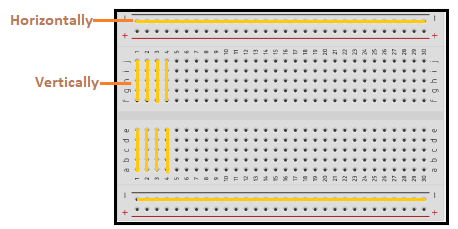
* **Breadboard Layout**:
  + Breadboards have two main areas:
    1. **Power Rails**: Run along the top and bottom for power supply connections (labeled "+" and "-").
    2. **Terminal Strips**: The central grid where components are placed; each row in the terminal strip connects a set of holes.

**Breadboard Configuration and Structure**

**Power Rails**

* Power rails run along the long edges and are used for connecting the positive (VCC) and negative (GND) terminals of a power supply.
* Each rail is internally connected, allowing the same voltage to be available at any point along the rail.

**Terminal Strip (Connecting Strip)**

* The central section is divided into two halves by a gap, with each half containing rows of interconnected holes.
* Each row of five holes on either side of the gap is connected horizontally. This allows for easy component placement and wiring between the two halves.  
  

**Conclusion**

The breadboard is an essential tool for prototyping circuits quickly and easily. Breadboards are invaluable for experimenting and learning circuit designs in electronics.

***Experiment No.:8***

**Description:**Use of ESP32, upload code on ESP 32 to blink onboard LED  
  
**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.

**Components and Tools Required**

* **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.

**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

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.

**Result** 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.