**INTERNET OF THINGS LAB RECORD**

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

|  |  |
| --- | --- |
| Name: | Srijita Kashyap |
| Registration Number: | FET-BAML-2022-2026-007 |
| Course: | Btech CSE AIML |
| Semester: | 5th |
| Faculty: | Mr. Biswajeeban Mishra &  Mr. Pritam Nanda |

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



SRI SRI UNIVERSITY

Bidyadharpur, Cuttack, Odisha.

Index

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sl. No.** | **Date** | **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** |  | Configuring MQTT in our Local Machine |  |  |
| **5** |  | Real-Time DHT11 Data on Node-RED Dashboard |  |  |
| **6** |  | Working w/ Button, LED & an UltraSonic [ HC-SR04 ] |  |  |
| **7** |  | ESPressif32 DevKit v1 [ NodeMCU: MicroController Unit ] |  |  |
| **8** |  |  |  |  |
| **9** |  |  |  |  |
| **10** |  |  |  |  |
| **11** |  |  |  |  |
| **12** |  |  |  |  |
| **13** |  |  |  |  |
| **14** |  |  |  |  |
| **15** |  |  |  |  |
| **16** |  |  |  |  |

***Experiment No.:1 Date: 9-8-2024***

***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:***

// Blink Internal LED

void setup() {

pinMode(13, OUTPUT); // Set pin 13 as an output

}

void loop() {

digitalWrite(13, HIGH); // Turn the LED on

delay(1000); // Wait for 1 second

digitalWrite(13, LOW); // Turn the LED off

delay(1000); // Wait for 1 second

}

# Explanation

In this code, the line pinMode(13, OUTPUT) configures pin 13 as an output, allowing us to control the internal LED on the Arduino board. The command digitalWrite(13, HIGH) then turns the LED on by setting pin 13 to a HIGH state, supplying voltage to the LED. This is followed by delay(1000) which pauses the program for 1000 milliseconds, or 1 second, keeping the LED on during this period. Next, digitalWrite(13, LOW) turns the LED off by setting pin 13 to a LOW state, stopping the current to the LED. Another delay(1000) pauses the program for an additional 1 second with the LED off. This sequence in the loop() function continuously repeats, making the LED blink on and off with a 1-second interval.

***Conclusion:***In conclusion, this simple LED blinking experiment demonstrates the basics of Arduino programming. By using digitalWrite() to control the LED's state and delay() to set intervals, we learned how to manipulate digital pins.



***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);

}

***Programming the LED:***

* pinMode(9, OUTPUT); configures pin 9 as an output, enabling us to control the external LED.
* digitalWrite(9, HIGH); supplies voltage to pin 9, turning the LED on.
* delay(1000); pauses the program for 1 second, keeping the LED on.
* digitalWrite(9, LOW); stops the current to pin 9, turning the LED off.
* delay(1000); pauses the program again for 1 second with the LED off

***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");

}

# Explanation:

The line DHT dht(DHTPIN, DHTTYPE) initializes the DHT22 sensor on the specified pin, setting it up for use in the program. Following this, dht.begin() starts communication with the DHT sensor, allowing the Arduino to interact with it and receive data. Inside the loop(), the code uses dht.readHumidity() and dht.readTemperature() to retrieve current humidity and temperature values from the sensor. If the sensor readings are successful, these values are printed to the Serial Monitor for observation. This process provides real-time monitoring of environmental conditions like humidity and temperature.

**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***

# Configuring MQTT in our Local Machine

To configure MQTT (Message Queuing Telemetry Transport) on a local machine, we’ll install and set up an MQTT broker. MQTT is a lightweight messaging protocol often used in IoT (Internet of Things) applications to allow multiple devices to communicate. Setting up MQTT on a local machine involves installing the broker software, configuring it, and testing it with a client.

**Steps for Configuring MQTT**

1. **Install an MQTT Broker**:
   * One of the most popular MQTT brokers is **Mosquitto**, which is free and open-source. You can install it on various platforms, including Windows, macOS, and Linux.
   * To install Mosquitto:
     + On **Windows**: Download the Mosquitto installer from the Mosquitto website.
     + On **macOS**: Using Homebrew by running brew install mosquitto.
     + On **Linux (Ubuntu)**: Use sudo apt-get install mosquitto.
2. **Configure the Broker**:
   * By default, Mosquitto is set to use port 1883. we can edit the configuration file (mosquitto.conf) to adjust settings like port, authentication, or persistence if needed.
   * On most systems, the configuration file is located at /etc/mosquitto/mosquitto.conf.
   * To require authentication, add a username and password file and enable password-based access by including password\_file /etc/mosquitto/passwd in the config file. We can create this file with the mosquitto\_passwd command.
3. **Start the MQTT Broker**:
   * On **Windows**: Run mosquitto from the command prompt or start it as a Windows service.
   * On **macOS/Linux**: Use the command sudo mosquitto -v to start the broker in verbose mode, which outputs detailed status messages.
   * Alternatively, use sudo systemctl start mosquitto to start the service if installed as a system service.
4. **Testing MQTT**:
   * Install an MQTT client like **MQTT Explorer** (GUI) or **mosquitto\_pub** and **mosquitto\_sub** (command-line tools).
   * Open two command prompts or terminal windows to test.
   * In one, run mosquitto\_sub -h localhost -t test/topic to subscribe to the test/topic.
   * In the other, publish a message to the same topic by running mosquitto\_pub -h localhost -t test/topic -m "Hello, MQTT!".
   * The message should appear in the subscriber window, confirming successful communication.

**Conclusion**

Configuring MQTT on a local machine provides a foundational setup for testing and developing IoT projects. With the broker and clients properly configured, we can build IoT applications that enable communication between devices over MQTT, making it ideal for home automation, sensor data collection, and control applications. This local MQTT setup is an efficient testing environment for validating MQTT-based interactions before deploying them in a larger network or cloud.

# Experiment No:5

# Real-Time DHT11 Data on Node-RED Dashboard

**Description:**

To display real-time DHT11 sensor data on a Node-RED dashboard, we’ll set up the hardware, connect it to an MQTT broker, and create a flow in Node-RED to read and visualize temperature and humidity data. This setup enables real-time monitoring of environmental conditions through a web-based dashboard.

**Steps to Display Real-Time DHT11 Data on Node-RED**

1. **Set Up Hardware and DHT11 Sensor**:
   * Connect the DHT11 sensor to an Arduino (or other microcontroller).
   * Wire the DHT11 sensor as follows:
     + **VCC** to **5V** on the Arduino.
     + **Data** to a digital pin (e.g., pin 2) on the Arduino.
     + **GND** to **GND** on the Arduino.
   * Use a 10kΩ pull-up resistor between VCC and the Data pin to stabilize readings.
2. **Install Node-RED and Required Nodes**:
   * Install Node-RED on your machine by following the Node-RED installation guide.
   * Open the Node-RED editor by entering http://localhost:1880 in a browser.
   * In Node-RED, install the required nodes:
     + **node-red-dashboard**: for creating a dashboard (npm install node-red-dashboard).
     + **node-red-contrib-dht-sensor**: if connecting the DHT11 directly to a Raspberry Pi (if using an Arduino with MQTT, you can skip this).
3. **Arduino Code to Publish Data to MQTT**:
   * Use the DHT11 library to read sensor data and publish it to an MQTT topic. Here’s sample Arduino code (assuming an MQTT broker is already set up):

Code:

#include <DHT.h>

#include <PubSubClient.h>

#include <WiFi.h>

// Replace with your network details

const char\* ssid = "your\_SSID";

const char\* password = "your\_PASSWORD";

const char\* mqtt\_server = "broker\_address"; // e.g., "192.168.1.10"

#define DHTPIN 2

#define DHTTYPE DHT11

DHT dht(DHTPIN, DHTTYPE);

WiFiClient espClient;

PubSubClient client(espClient);

void setup() {

Serial.begin(115200);

dht.begin();

WiFi.begin(ssid, password);

while (WiFi.status() != WL\_CONNECTED) {

delay(500);

Serial.print(".");

}

client.setServer(mqtt\_server, 1883);

}

void loop() {

if (!client.connected()) {

while (!client.connected()) {

client.connect("DHT11Client");

}

}

float humidity = dht.readHumidity();

float temperature = dht.readTemperature();

if (!isnan(humidity) && !isnan(temperature)) {

String payload = "{\"temperature\": " + String(temperature) + ", \"humidity\": " + String(humidity) + "}";

client.publish("home/dht11", payload.c\_str());

}

client.loop();

delay(2000); // Adjust for desired frequency

}

1. **Node-RED Flow Setup**:
   * Add an **MQTT input node** and configure it to listen to the home/dht11 topic on the local broker.
   * Connect the MQTT node to a **JSON node** to parse the JSON string into a JavaScript object.
   * Use **function nodes** (optional) to format data if necessary.
   * Connect the output to a **Dashboard Gauge** or **Chart node** to display temperature and humidity on the dashboard.
2. **Configure the Dashboard**:
   * Open the Node-RED dashboard editor (on the sidebar) to customize the layout.
   * Set up two gauges (or charts) to show temperature and humidity values.
   * Deploy the flow and access the dashboard by navigating to http://localhost:1880/ui.

**Conclusion**

Using Node-RED with MQTT and a DHT11 sensor, we can create a real-time, web-based dashboard for monitoring temperature and humidity. This setup provides a hands-on way to visualize sensor data, ideal for IoT applications such as smart home monitoring or environmental data logging. The modularity of Node-RED also allows easy expansion to include other sensors and devices.

# Experiment No: 6

# Description:

Working with a button, an LED, and an ultrasonic sensor (HC-SR04) allows us to interact with various input and output components on an Arduino. In this setup, the button will toggle the LED on or off, while the ultrasonic sensor measures distance. If an object is detected within a certain range, the LED will indicate it by lighting up, adding an interactive element to the circuit.

**Components Needed**

* Arduino board
* 1 LED (any color)
* 1 Button
* 1 Resistor for LED (220Ω or 330Ω)
* 1 Resistor for button (10kΩ)
* HC-SR04 ultrasonic sensor
* Jumper wires

**Wiring**

1. **Button**:
   * Connect one side of the button to a digital input pin (e.g., pin 7).
   * Connect the other side to **GND**.
   * Place a 10kΩ pull-down resistor between the button's input pin and **GND**.
2. **LED**:
   * Connect the long leg (anode) of the LED to a digital output pin (e.g., pin 13).
   * Connect the short leg (cathode) to one end of a resistor (220Ω), and the other end of the resistor to **GND**.
3. **HC-SR04 Ultrasonic Sensor**:
   * **VCC** to **5V** on the Arduino.
   * **GND** to **GND** on the Arduino.
   * **Trigger** (Trig) pin to a digital pin (e.g., pin 9).
   * **Echo** pin to another digital pin (e.g., pin 10).

code:

// Define pins

const int buttonPin = 7;

const int ledPin = 13;

const int trigPin = 9;

const int echoPin = 10;

bool ledState = false; // State of LED (on/off)

int buttonState = 0; // Variable to store button state

void setup() {

pinMode(buttonPin, INPUT);

pinMode(ledPin, OUTPUT);

pinMode(trigPin, OUTPUT);

pinMode(echoPin, INPUT);

Serial.begin(9600); // For debugging

}

void loop() {

// Button handling

buttonState = digitalRead(buttonPin);

if (buttonState == HIGH) {

ledState = !ledState; // Toggle LED state

digitalWrite(ledPin, ledState ? HIGH : LOW);

delay(300); // Debounce delay

}

// Ultrasonic distance measurement

long duration, distance;

digitalWrite(trigPin, LOW);

delayMicroseconds(2);

digitalWrite(trigPin, HIGH);

delayMicroseconds(10);

digitalWrite(trigPin, LOW);

// Measure the distance

duration = pulseIn(echoPin, HIGH);

distance = duration \* 0.034 / 2; // Calculate distance in cm

// LED control based on distance

if (distance < 10) {

digitalWrite(ledPin, HIGH); // Turn on LED if object is close

} else if (!ledState) {

digitalWrite(ledPin, LOW); // Keep LED off if not toggled on by button

}

// Print distance for debugging

Serial.print("Distance: ");

Serial.print(distance);

Serial.println(" cm");

delay(100); // Short delay for stability

}

**Explanation of the Code:**

1. **Button Control**:
   * digitalRead(buttonPin) reads the button's state. If it’s pressed, the ledState variable is toggled, which changes the LED’s state (on/off).
2. **Distance Measurement with Ultrasonic Sensor**:
   * The ultrasonic sensor sends a pulse with digitalWrite(trigPin, HIGH) and measures the time for the pulse to return using pulseIn(echoPin, HIGH).
   * The distance is calculated with distance = duration \* 0.034 / 2, where 0.034 is the speed of sound in cm/µs, and dividing by 2 accounts for the pulse's round trip.
3. **LED Control Based on Distance**:
   * If the measured distance is below 10 cm, the LED lights up as a warning.
   * If the button hasn’t turned on the LED and the distance condition isn’t met, the LED remains off.

**Conclusion**

This setup demonstrates how to integrate multiple components with an Arduino to create an interactive project. The button lets you manually control the LED, while the HC-SR04 ultrasonic sensor automatically triggers the LED based on object proximity. This type of circuit is foundational in robotics and IoT, where sensor-based automation is common.

# Experiment 7:

# ESPressif32 DevKit v1 [ NodeMCU: MicroController Unit ]

**Description**

The ESP32 DevKit v1 is a popular development board that features the ESP32 microcontroller, which is well-known for its Wi-Fi and Bluetooth capabilities. Here’s a brief overview of its key features and specifications:

**Key Features**

1. **Microcontroller**: ESP32 chip, dual-core, 32-bit processor.
2. **Connectivity**:
   * Wi-Fi 802.11 b/g/n
   * Bluetooth v4.2 (classic and BLE)
3. **GPIO Pins**: Typically 30 GPIO pins available, which can be used for various interfaces like I2C, SPI, UART, PWM, etc.
4. **Memory**:
   * Flash memory: Usually 4MB (may vary with the model)
   * RAM: 520KB
5. **Operating Voltage**: 3.3V (do not apply 5V directly)
6. **Programming**: Compatible with the Arduino IDE, PlatformIO, and other environments. Supports MicroPython as well.
7. **Power Management**: Various sleep modes for power saving.

**Typical Applications**

* IoT projects (Internet of Things)
* Smart home devices
* Wearable technology
* Robotics
* Sensor data collection and processing

**Getting Started**

1. **Setup**:
   * Install the Arduino IDE or any other compatible IDE.
   * Add the ESP32 board to the board manager.
2. **Example Code**: Start with basic examples like blinking an LED or connecting to Wi-Fi.
3. **Libraries**: Utilize libraries for Wi-Fi, Bluetooth, and other peripherals.

**Code:**

**Blink an LED**

// Define the pin where the LED is connected

const int ledPin = 2; // Change to your pin if necessary

void setup() {

// Initialize the LED pin as an output

pinMode(ledPin, OUTPUT);

}

void loop() {

// Turn the LED on

digitalWrite(ledPin, HIGH);

// Wait for a second

delay(1000);

// Turn the LED off

digitalWrite(ledPin, LOW);

// Wait for a second

delay(1000);

}

# Connect to Wi-Fi

#include <WiFi.h>

// Replace with your network credentials

const char\* ssid = "YOUR\_SSID";

const char\* password = "YOUR\_PASSWORD";

void setup() {

// Start the Serial communication

Serial.begin(115200);

// Connect to Wi-Fi

WiFi.begin(ssid, password);

Serial.print("Connecting to WiFi");

while (WiFi.status() != WL\_CONNECTED) {

delay(1000);

Serial.print(".");

}

Serial.println();

Serial.print("Connected to WiFi. IP address: ");

Serial.println(WiFi.localIP());

}

void loop() {

// Nothing to do here, but you can add your main code

}