# NAMIT LOHARKAR BT22CIV101

# EXPERIMENT NO. - 10

# Aim: Transfer of information to and from an ESP32 using MQTT protocol.

# Apparatus:

- ESP32 Development Board  
- Jumper Wires  
- Breadboard  
- USB Cable (for connecting ESP32 to laptop/PC)  
- Wi-Fi Connectivity

**Theory -**MQTT (Message Queuing Telemetry Transport) is a lightweight, publish-subscribe messaging protocol designed for low-bandwidth and resource-constrained devices. It operates on a client-server model, with a central broker facilitating communication between devices.

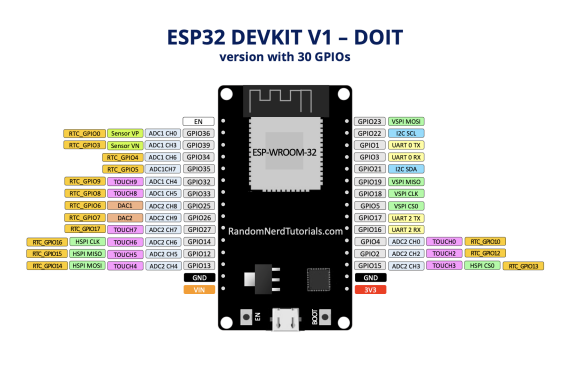
**ESP32 and MQTT Communication**  
The ESP32, a popular IoT microcontroller with built-in Wi-Fi and Bluetooth, uses MQTT for seamless communication between devices or cloud servers. It can act as a publisher or subscriber, connecting to an MQTT broker via TCP/IP. The broker facilitates communication by routing messages based on **topics**, which are hierarchical channels for categorizing data. For instance, an ESP32 publishing sensor data might use the topic home/livingroom/temperature, while another device subscribing to this topic receives the updates instantly.

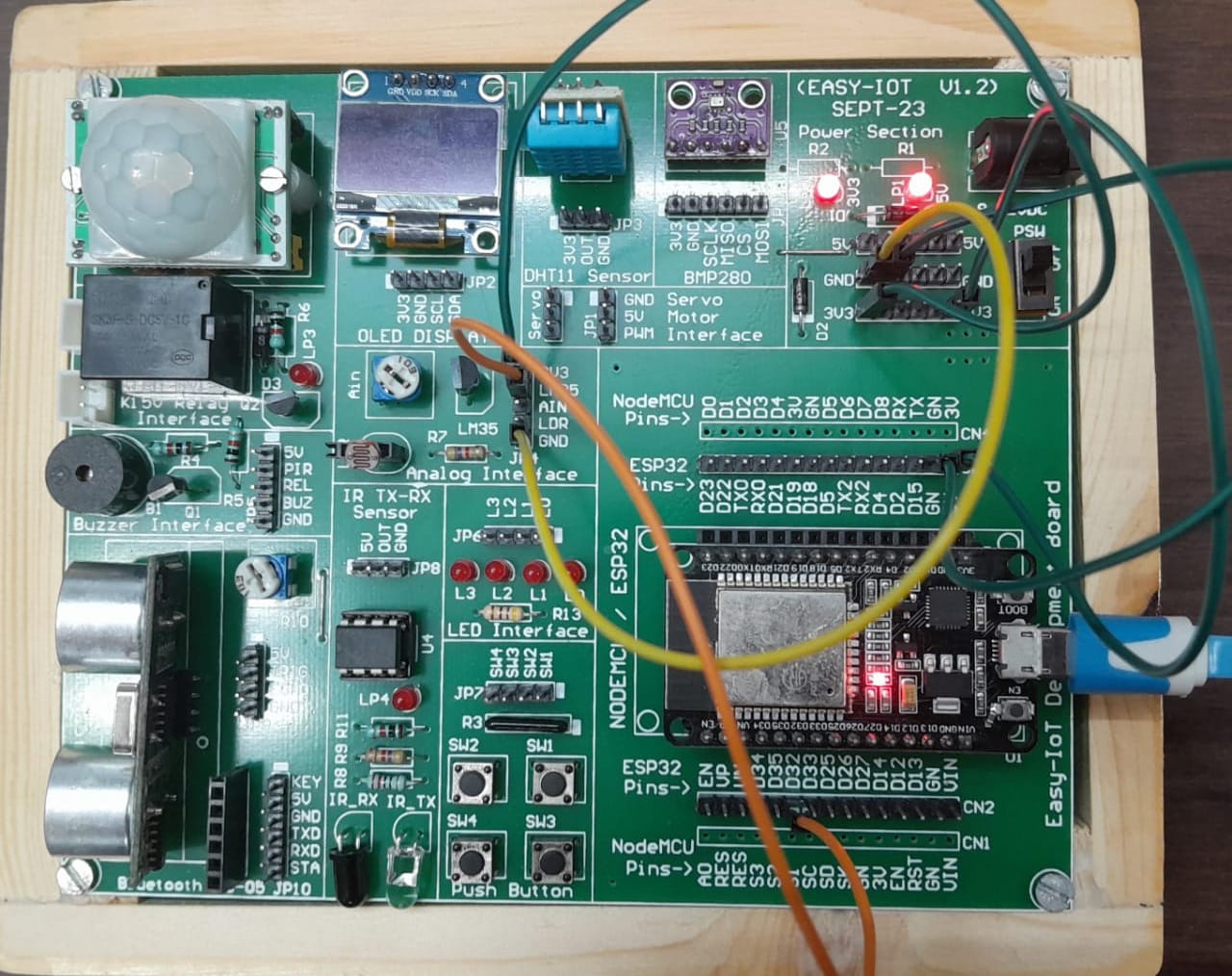
**Applications and Workflow**  
Smart home automation.

Real-time monitoring (e.g., weather stations, health data).

Industrial IoT systems.

**ESP32 required pin diagram-**



**Connection diagram-**  
  
 

**Procedure:**

1. Set Up the Environment

Install Arduino IDE.

Add ESP32 board support to the IDE.

Install the PubSubClient library.

2. Configure Wi-Fi and MQTT

Update the ssid and password variables with your Wi-Fi credentials.

Set the broker details (broker: broker.emqx.io, topic: emqx/esp32).

3. Upload the Code

Connect the ESP32 to your computer.

In the Arduino IDE, select the appropriate board and port.

Upload the code to the ESP32.

4. Monitor Serial Output

Open the Serial Monitor in Arduino IDE with a baud rate of 115200.

Observe the connection status for Wi-Fi and MQTT, published messages, and updates to subscribe topics

5. Test MQTT Communication

Use an MQTT client (e.g., MQTT Explorer) to publish messages to the topic emqx/esp32.

Check the ESP32’s response on the Serial Monitor.

6. Debugging

Verify Wi-Fi and MQTT credentials.

Check network connectivity.

Ensure the MQTT broker is accessible if any issues occur. **CODE-**

#include <WiFi.h>

#include <PubSubClient.h>

// WiFi

const char \*ssid = "Emb\_Lab"; // Enter your Wi-Fi name

const char \*password = "emb@1234"; // Enter Wi-Fi password

// MQTT Broker

const char \*mqtt\_broker = "broker.emqx.io";

const char \*topic = "emqx/esp32";

const char \*mqtt\_username = "emqx";

const char \*mqtt\_password = "public";

const int mqtt\_port = 1883;

WiFiClient espClient;

PubSubClient client(espClient);

void setup() {

// Set software serial baud to 115200;

Serial.begin(115200);

// Connecting to a WiFi network

WiFi.begin(ssid, password);

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

delay(500);

Serial.println("Connecting to WiFi..");

}

Serial.println("Connected to the Wi-Fi network");

//connecting to a mqtt broker

client.setServer(mqtt\_broker, mqtt\_port);

client.setCallback(callback);

while (!client.connected()) {

String client\_id = "esp32-client-";

client\_id += String(WiFi.macAddress());

Serial.printf("The client %s connects to the public MQTT broker\n", client\_id.c\_str());

if (client.connect(client\_id.c\_str(), mqtt\_username, mqtt\_password)) {

Serial.println("Public EMQX MQTT broker connected");

} else {

Serial.print("failed with state ");

Serial.print(client.state());

delay(2000);

}

}

// Publish and subscribe

client.publish(topic, "Hi, I'm ESP32 ^^");

client.subscribe(topic);

}

void callback(char \*topic, byte \*payload, unsigned int length) {

Serial.print("Message arrived in topic: ");

Serial.println(topic);

Serial.print("Message:");

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

Serial.print((char) payload[i]);

}

Serial.println();

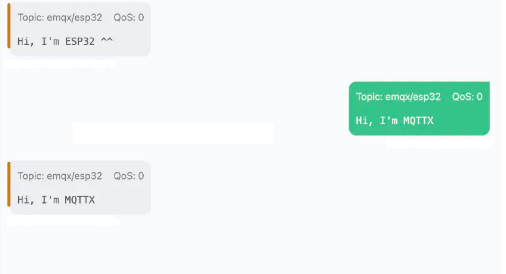
Serial.println("-----------------------");

}

void loop() {

client.loop();

}

**Photos of results :  
  
**

**Observations:**

* **Wi-Fi Connectivity**:  
  The ESP32 attempts to connect to the specified Wi-Fi network and continuously retries until successful. The connection status is printed on the Serial Monitor.
* **MQTT Broker Connection**:  
  The ESP32 connects to the MQTT broker using the provided credentials. If the connection fails, the client.state() function prints the failure reason for debugging.
* **Message Publishing**:  
  Upon successful connection, the ESP32 publishes the message Hi, I'm ESP32 ^^ to the topic emqx/esp32.
* **Subscription and Callback**:  
  The ESP32 subscribes to the topic emqx/esp32 and processes any incoming messages through the callback function, displaying the topic name and message on the Serial Monitor.
* **Loop Functionality**:  
  The client.loop() in the loop() function ensures the MQTT client continuously checks for new messages or maintains its connection to the broker.

# Conclusion:

This code demonstrates how to establish seamless communication between an ESP32 microcontroller and an MQTT broker using the publish-subscribe model. By leveraging the lightweight MQTT protocol, the ESP32 can efficiently publish messages and subscribe to topics, enabling real-time data exchange. The integration of Wi-Fi connectivity ensures that the ESP32 can interact with cloud-based or local MQTT brokers, making it a versatile solution for IoT applications such as smart homes, industrial monitoring, and automation. The use of callback functions to handle incoming messages highlights the event-driven nature of MQTT, ensuring prompt and reliable processing of data. This implementation is a foundational step toward building robust IoT systems with low-power microcontrollers like the ESP32.