**BUZZER CONTROL**

**ABSTRACT:**

This project demonstrates the development of a WiFi-connected buzzer system using the ESP32 microcontroller and the MQTT protocol. The objective is to remotely control a buzzer via a wireless network, providing real-time status updates through MQTT messages. The ESP32, equipped with built-in WiFi capabilities, connects to a specified WiFi network and an MQTT broker to facilitate communication.

Upon receiving MQTT messages, the system interprets these commands to either turn the buzzer on or off. Feedback on the buzzer's status is then published back to a designated MQTT topic. This setup allows for seamless remote control and monitoring, making it ideal for applications such as security alarms, notification systems, and other IoT-based control mechanisms.

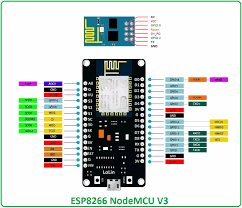
The implementation involves configuring the ESP32 to connect to a WiFi network, setting up MQTT communication using the PubSubClient library, and programming the microcontroller to handle buzzer control based on received MQTT messages. The system subscribes to a command topic and publishes feedback messages to a feedback topic, ensuring real-time updates on the buzzer's status.

In testing, the system demonstrated reliable performance, successfully turning the buzzer on and off in response to MQTT commands and providing accurate feedback. This highlights the efficiency of using the MQTT protocol for IoT applications, demonstrating how a simple device like a buzzer can be controlled and monitored remotely with minimal hardware and straightforward software implementation.

Future enhancements could include integrating additional sensors and actuators, improving security features to ensure data integrity and privacy, and developing user-friendly interfaces, such as mobile apps or web dashboards, for more intuitive control and interaction. This project underscores the potential of IoT technologies in creating smart, connected devices that can be managed effortlessly over the internet.

**COMPONENTS:**

**ESP8266**: The ESP8266 is a low-cost Wi-Fi microcontroller chip with full TCP/IP stack and microcontroller capability. It is widely used in IoT projects due to its ability to connect to a Wi-Fi network and process data. In this project, the ESP8266 handles Wi-Fi connectivity, receives control commands, and interfaces with both the servo motor and the LCD display.



**BUZZER:**The buzzer system integrated with the Arduino IoT Cloud enables remote toggling of its state, providing a seamless user experience for activating or deactivating auditory feedback. This functionality is achieved through the synchronization of the NodeMCU microcontroller with the cloud platform, facilitating real-time control and monitoring of the buzzer's operations from anywhere with an internet connection.



**Jumper Wires**: Jumper wires are insulated wires with connectors (typically male or female pins) at each end, used to create temporary or semi-permanent connections between different components on a breadboard or between a breadboard and other devices.



**CODE:**

#include <WiFi.h>

#include <PubSubClient.h>

#define BUZZER\_PIN 2

const char\* ssid = "moto g54 5G\_3480";

const char\* password = "9500697232";

const char\* mqtt\_server = "broker.mqttdashboard.com";

#define COMMAND\_TOPIC "OsoyooCommand"

#define FEEDBACK\_TOPIC "BuzzerFeedback"

WiFiClient espClient;

PubSubClient client(espClient);

void setup\_wifi() {

delay(100);

Serial.print("Connecting to ");

Serial.println(ssid);

WiFi.begin(ssid, password);

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

delay(500);

Serial.print(".");

}

randomSeed(micros());

Serial.println("");

Serial.println("WiFi connected");

Serial.println("IP address: ");

Serial.println(WiFi.localIP());

}

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

Serial.print("Command from MQTT broker is : [");

Serial.print(topic);

int p = (char)payload[0]-'0';

// if MQTT comes a 0 turn off the buzzer

if (p == 0) {

digitalWrite(BUZZER\_PIN, LOW);

client.publish(FEEDBACK\_TOPIC, "Buzzer turned off");

Serial.println(" Turn Off BUZZER! ");

}

// if MQTT comes a 1, turn on the buzzer

else if (p == 1) {

digitalWrite(BUZZER\_PIN, HIGH);

client.publish(FEEDBACK\_TOPIC, "Buzzer turned on");

Serial.println(" Turn On BUZZER! ");

}

Serial.println();

}

void reconnect() {

while (!client.connected()) {

Serial.print("Attempting MQTT connection...");

String clientId = "ESP32Client-";

clientId += String(random(0xffff), HEX);

if (client.connect(clientId.c\_str())) {

Serial.println("connected");

client.subscribe(COMMAND\_TOPIC);

} else {

Serial.print("failed, rc=");

Serial.print(client.state());

Serial.println(" try again in 5 seconds");

delay(6000);

}

}

}

void setup() {

Serial.begin(115200);

setup\_wifi();

client.setServer(mqtt\_server, 1883);

pinMode(BUZZER\_PIN, OUTPUT);

digitalWrite(BUZZER\_PIN, LOW);

}

void loop() {

if (!client.connected()) {

reconnect();

}

client.setCallback(callback);

client.loop();

}

}

**STEPS TO CREATE A PROJECT IN ARDUINO IOT CLOUD**

1. **Set Up the Hardware:**

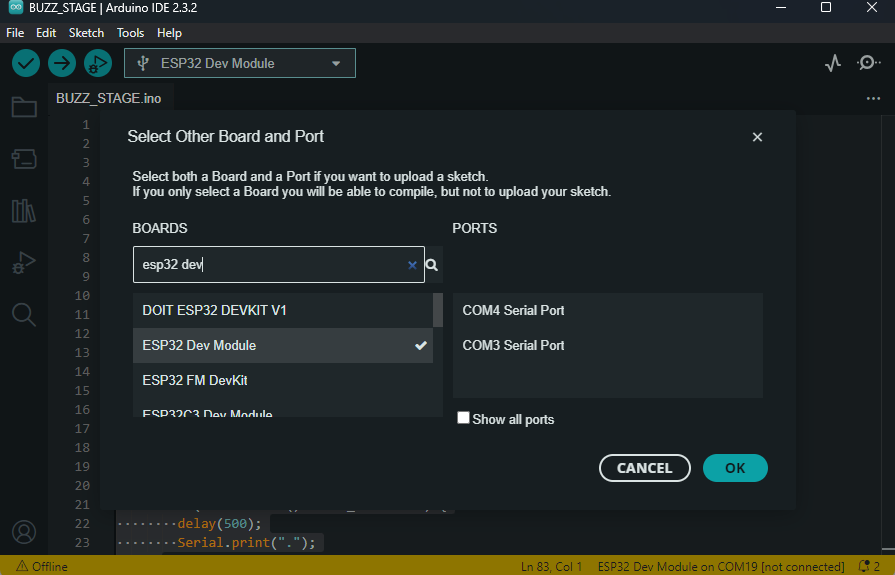
* Connect the piezo buzzer to GPIO pin 2 of the ESP32.
* Connect the positive pin of the buzzer to GPIO pin 2.
* Connect the negative pin of the buzzer to a GND pin on the ESP32.

**Install Arduino IDE:**

* Download and install the Arduino IDE from the official website (https://www.arduino.cc/en/software).

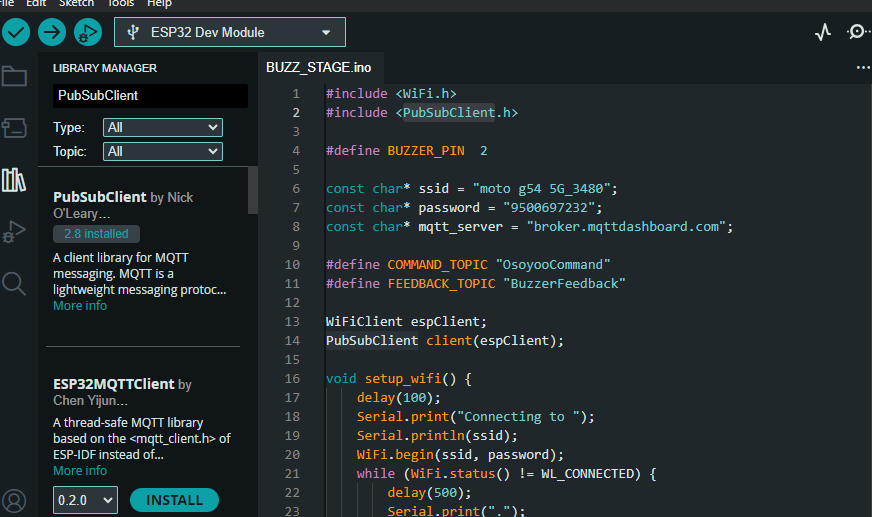
**Install ESP32 Board in Arduino IDE:**

* Open the Arduino IDE.
* Go to Tools > Board > Boards Manager.
* Search for ESP32 and install the esp32 by Espressif Systems.



**Install PubSubClient Library:**

* In the Arduino IDE, go to Sketch > Include Library > Manage Libraries.
* Search for PubSubClient and install it.

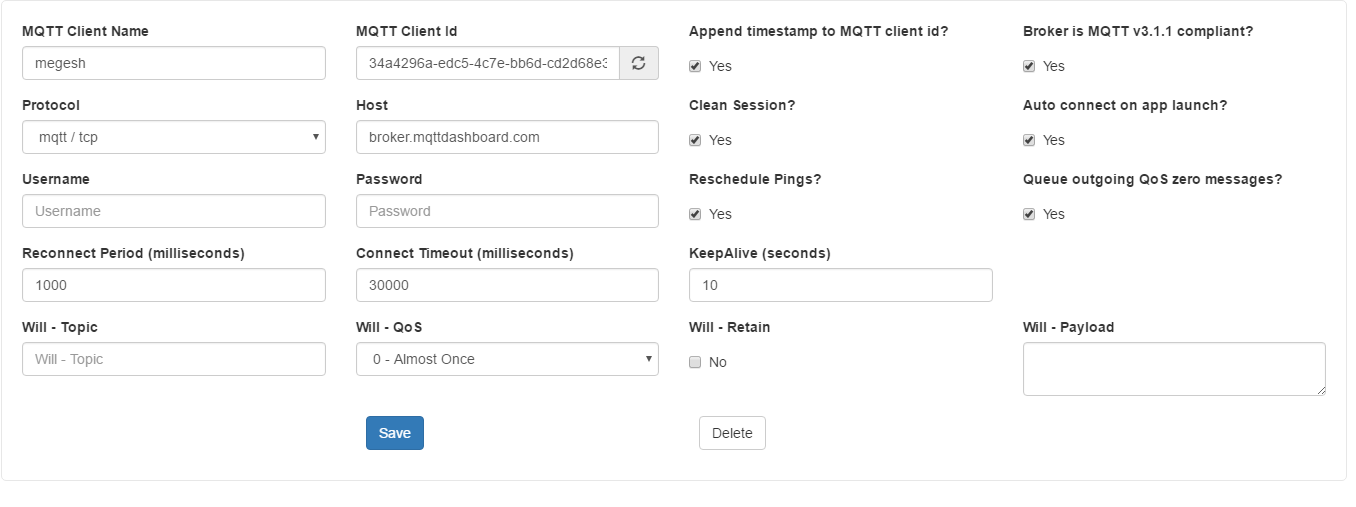


**Write the Code:**

* Open a new sketch in Arduino IDE and copy the following code:
* Connect your ESP32 to your computer using a USB cable.
* Select the correct board and port in the Arduino IDE:
* Go to Tools > Board > ESP32 Dev Module.
* Go to Tools > Port and select the COM port to which the ESP32 is connected.
* Click on the upload button to compile and upload the code to the ESP32.

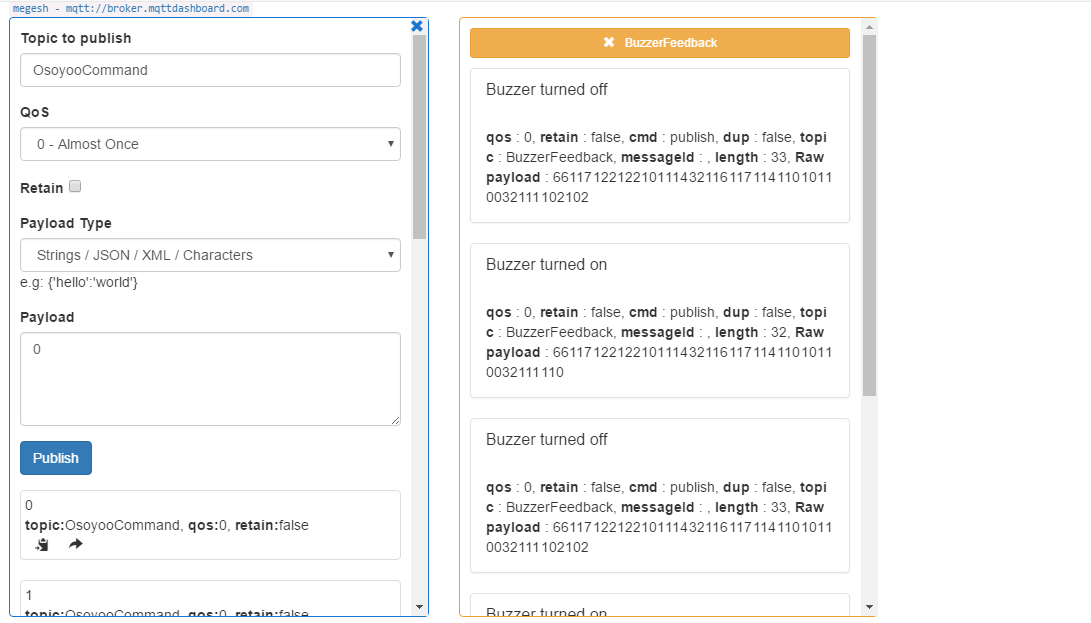
**Set Up MQTT Box:**

* Download and install MQTT Box (available as a browser extension or standalone application).
* Open MQTT Box and create a new MQTT client.
* Configure the client with the following settings:
* Client ID: Any unique identifier (e.g., MQTTBoxClient).
* Protocol: MQTT/TCP.
* Host: broker.mqttdashboard.com.
* Port: 1883.
* Connect the client.

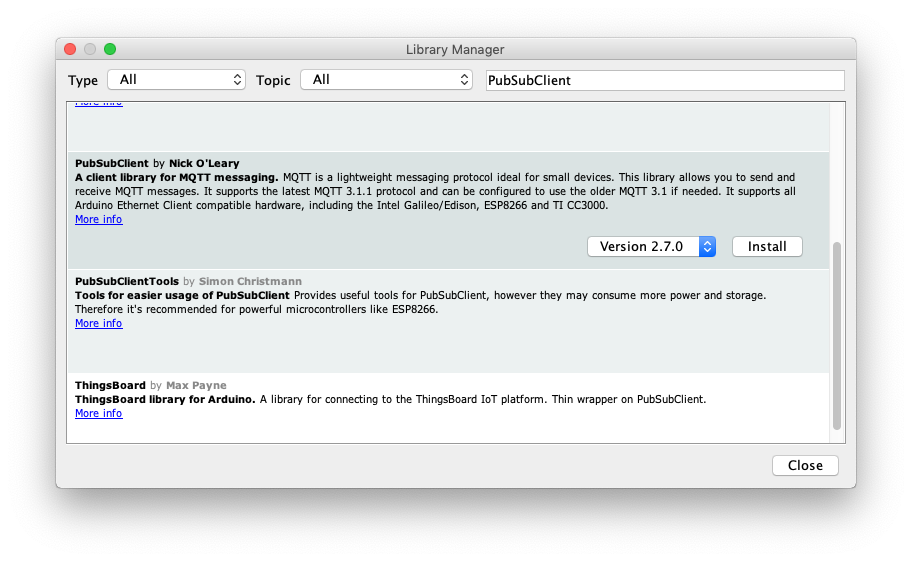


**Send Commands Using MQTT Box:**

* Subscribe to the feedback topic (BuzzerFeedback) to see the buzzer status.
* Publish messages to the command topic (OsoyooCommand):
* To turn the buzzer on, publish the message 1 to OsoyooCommand.
* To turn the buzzer off, publish the message 0 to OsoyooCommand.
* Observe the feedback messages in the subscribed feedback topic to confirm the buzzer status.



**PubSubClient:**



The PubSubClient library is a lightweight and efficient library designed for Arduino-compatible boards to facilitate MQTT communication. It allows these microcontrollers, such as the ESP32, to connect to an MQTT broker, publish messages to topics, and subscribe to topics to receive messages. This library is crucial for implementing IoT applications, enabling seamless and reliable data exchange between devices over the MQTT protocol.



**CODE EXPALNATION**

· **Setup Wi-Fi Connection:**

* The setup\_wifi() function initializes the ESP32's Wi-Fi connection by calling WiFi.begin() with the provided SSID and password.
* It then enters a loop, continuously checking the Wi-Fi status (WL\_CONNECTED) until the connection is established.
* The function also initializes the random number generator using randomSeed(micros()), which is commonly used for generating unique client IDs for MQTT connections.

· **MQTT Connection:**

* The PubSubClient library is utilized to establish an MQTT connection with the broker specified by mqtt\_server.
* The reconnect() function handles reconnection attempts to the MQTT broker in case the connection is lost. It generates a random client ID for the ESP32 and attempts to connect to the broker using client.connect().
* If the connection attempt fails, the function prints an error message along with the current connection state (client.state()) and retries after a delay of 6 seconds.

· **Buzzer Control:**

* The callback() function is invoked whenever a message is received on the MQTT topic specified by COMMAND\_TOPIC.
* It parses the received message (payload) to determine the command (0 for turning off the buzzer, 1 for turning it on).
* If the command is 0, it sets the buzzer pin (BUZZER\_PIN) low to turn off the buzzer and publishes a feedback message to the FEEDBACK\_TOPIC.
* If the command is 1, it sets the buzzer pin high to turn on the buzzer and publishes a corresponding feedback message.

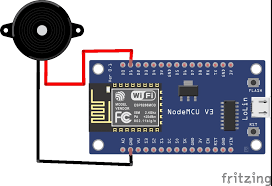
· **Main Loop:**

* The loop() function continuously checks the MQTT connection status using client.connected().
* If the connection is lost, it invokes the reconnect() function to attempt reconnection.
* It also sets the callback function for handling incoming MQTT messages (client.setCallback()), allowing the ESP32 to respond to commands in real-time.
* Finally, it calls client.loop() to maintain the MQTT connection and handle incoming/outgoing messages.

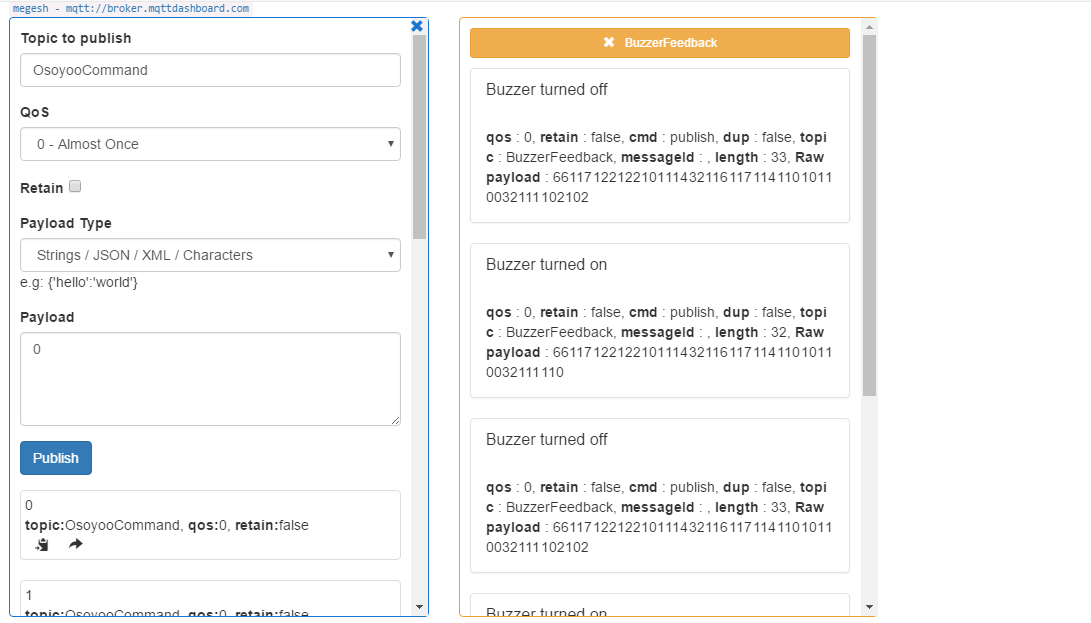
· **Serial Output:**

* Serial messages are utilized for debugging purposes to monitor the execution flow of the program.
* Messages are printed during Wi-Fi connection setup, MQTT connection attempts, and whenever a command is received and executed, providing visibility into the device's operational status and response to MQTT commands.

**CIRCUIT DIAGRAM:**



**MQTT DASH BOARD**



**CONCLUSION:**

In conclusion, the provided code demonstrates a practical implementation of an IoT (Internet of Things) project using an ESP32 microcontroller, Wi-Fi connectivity, and MQTT (Message Queuing Telemetry Transport) protocol. By leveraging these technologies, the project achieves remote control of a buzzer, allowing users to send commands over the internet to turn the buzzer on or off.

The key components of the project include:

1. **Wi-Fi Connectivity:** The ESP32 connects to a Wi-Fi network, enabling internet access for communication with external MQTT brokers.
2. **MQTT Communication:** The PubSubClient library facilitates communication with an MQTT broker, enabling the ESP32 to subscribe to specific topics to receive commands and publish feedback messages.
3. **Buzzer Control:** The ESP32 responds to MQTT commands received on a predefined topic, allowing users to remotely control the state of a connected buzzer.
4. **Serial Debugging:** Serial output provides valuable insights into the program's execution, including Wi-Fi connection status, MQTT connection attempts, and command execution feedback, aiding in troubleshooting and monitoring.

This project exemplifies the versatility and practicality of IoT applications, showcasing how simple hardware components and communication protocols can be leveraged to create remote-controlled devices. With further development and integration, similar principles can be applied to various IoT scenarios, ranging from home automation to industrial monitoring and control systems