**5GSM WITH MQTT, CALL ,SMS USING INTERUPT**

**ABSTRACT:**

This project integrates a GSM module with an Arduino to simultaneously manage MQTT communication, voice calls, and SMS notifications, utilizing an interrupt-driven approach. The system is designed to handle multiple communication methods in real-time while prioritizing incoming GSM events like calls or SMS.

The setup includes a SIM800A GSM module connected via software serial to the Arduino, with an interrupt pin configured to monitor the Ring Indicator (RI) pin of the GSM module. When an incoming call or SMS is detected, the interrupt is triggered, and the system pauses MQTT operations to manage the GSM event. The buzzer, connected to pin 6 of the Arduino, can be controlled via MQTT messages, incoming calls, or SMS, with the state change communicated back via SMS.

Key components include:

* **MQTT Communication**: Managed using the TinyGsmClient and PubSubClient libraries, allowing remote control of the buzzer via MQTT messages.
* **GSM Handling**: The GSM module is initialized and connected to a GPRS network for MQTT communication. Incoming calls or SMS messages trigger an interrupt, causing the system to prioritize handling the GSM event.
* **Interrupt-Driven Design**: The interrupt pin monitors the RI signal from the GSM module, allowing the system to respond immediately to GSM events without polling.
* **Event Handling**: The system can process MQTT messages and handle GSM events such as incoming calls (which can toggle the buzzer state) and SMS messages (which can turn the buzzer on or off).

**COMPONENTS:**

**ARDUINO UNO:**The Arduino Uno is a popular microcontroller board based on the ATmega328P, ideal for beginners and hobbyists. It features 14 digital I/O pins, 6 analog inputs, and operates at 5V. The board is compatible with various shields and sensors, making it versatile for a wide range of DIY projects.



**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.



**GSM 800/900:**The SIM800/900 modules are GSM/GPRS-based communication modules commonly used with microcontrollers like Arduino. They allow devices to send SMS, make calls, and connect to the internet via GPRS. These modules are widely used in IoT projects for remote communication and control.



**CODE:**

int interruptPin = 2;//set the interupt pin

volatile bool interruptFlag = false;

int call\_count=1;//flags

int first\_interupt\_flag=1;//intilizing gsm interuptflag

String phoneNumber="+919080991747";

#define TINY\_GSM\_MODEM\_SIM800

//we cannot directly uses the hardware serial of arduino(serial data of arduino and gsm collide ) so i use the software serial

#include <SoftwareSerial.h>

/\*Sending and receiving SMS messages

Making and receiving voice calls

Connecting to the internet via GPRS

Using TCP/UDP protocols for communication

Handling MQTT communication\*/

#include <TinyGsmClient.h>

//PubSubClient is a popular MQTT client library that allows you to connect your Arduino to an MQTT broker and publish or subscribe to MQTT topics.

#include <PubSubClient.h>

// Define software serial pins for GSM module

SoftwareSerial gsmSerial(7, 8); // RX, TX

/\*Debugging Output: When you define TINY\_GSM\_DEBUG and set it to Serial, the TinyGSM library will print debug information to the serial monitor.

This can include details about the AT commands being sent to the GSM module, responses received, and other internal processes.

Serial Monitor Viewing: By setting it to Serial, you can view the debugging information on your computer's serial monitor

(e.g., using the Arduino IDE's Serial Monitor) to understand what's happening in the background when your Arduino communicates with the GSM module.\*/

#define TINY\_GSM\_DEBUG Serial

// Buzzer Pin

const int buzzerPin = 6;

// GSM and MQTT settings

const char\* broker = "new-iot.cloud.shiftr.io";

const char\* mqttUsername = "new-iot";

const char\* mqttPassword = "dn3WXRPJD77svMgG";

const char\* apn = "airtelgprs.com";

const char\* gprsUser = "";

const char\* gprsPass = "";

const char\* topicOutput1 = "/buzz/sig";//to to give input

//const char\* topicState = "/buzz/state";//to get the state

// GSM and MQTT clients

TinyGsm modem(gsmSerial);

TinyGsmClient gsmClient(modem);

PubSubClient mqttClient(gsmClient);

void setup() {

pinMode(interruptPin, INPUT);// set as interupt pin in input

Serial.begin(9600); // Initialize serial communication

// Attach interrupt to pin 2, trigger on RISING signal and FALLING signal

attachInterrupt(digitalPinToInterrupt(interruptPin), handleInterrupt, RISING);// interupt activate when the state chages

gsmSerial.begin(9600); // Start serial communication for GSM module

pinMode(buzzerPin, OUTPUT); // Set buzzer pin as output

Serial.println("Initializing GSM...");

if (!initializeGSM()) {

Serial.println("GSM initialization failed.");

while (true)

;

}

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

if (!connectGPRS()) {

Serial.println("GPRS connection failed.");

while (true)

;

}

mqttClient.setServer(broker, 1883);

mqttClient.setCallback(mqttCallback);

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

if (!connectMQTT()) {

Serial.println("MQTT connection failed.");

while (true)

;

}

Serial.println("Setup complete");

}

void loop() {

// Handle MQTT connection

if (!mqttClient.connected()) {

reconnectMQTT();

}

mqttClient.loop();

if (interruptFlag) {//only interupt flag is true

Serial.println("Interrupt detected!");

interruptFlag = false; // Reset interrupt flag

handleGSMEvent(); // Handle GSM event

}

delay(500); // Add a small delay to prevent flooding

if (gsmSerial.available() > 0) {// check for the data in gsmserial

String incomingData = gsmSerial.readString();// read data and store it as string

//call first time buzzer on and call second time buzzer off

if (incomingData.indexOf("RING") != -1) {// string is RING then call in occured

Serial.println("Incoming call detected.");

if(call\_count%2==1){// odd for on

digitalWrite(buzzerPin, HIGH);// buzzer on

Serial.println("buzzer on");

delay(1000);

gsmSerial.println("ATH\r");

SendMessage("buzzer on",phoneNumber);//send on message

}

else// even for off

{

digitalWrite(buzzerPin, LOW);// buzzer off

Serial.println("buzzer off");

delay(1000);

gsmSerial.println("ATH\r");

SendMessage("buzzer off",phoneNumber);// send off message

}

call\_count++;// add the 1 to call\_count

}

else if(incomingData.indexOf("ON") != -1)// if on through sms

{

digitalWrite(buzzerPin, HIGH); // Turn the buzzer on

Serial.println("Buzzer on sms");

// Optional: Send an SMS indicating that the buzzer is on

SendMessage("Buzzer on sms",phoneNumber);

}

else if(incomingData.indexOf("OFF") != -1)// if off through sms

{

digitalWrite(buzzerPin, LOW); // Turn the buzzer on

Serial.println("Buzzer off sms");

// Optional: Send an SMS indicating that the buzzer is on

SendMessage("Buzzer off sms",phoneNumber);

delay(3000);// small dely

reconnectMQTT();// connect to mqtt

}

}

}

bool initializeGSM() {

modem.restart();//used to reset the gsm andstarts with a clean state after an unexpected error or disconnection

//this was the reson why the ri pin get high after intiliztion but we need this function for unexpected events

delay(1000);

//optional

String modemInfo = modem.getModemInfo();// get the modem info

Serial.print("Modem Info: ");

Serial.println(modemInfo);

return true;

}

bool connectGPRS() {//for connected to mqtt we need gprs

Serial.print("Connecting to APN: ");

Serial.println(apn);

for (int i = 0; i < 5; i++) { // Retry 5 times

if (modem.gprsConnect(apn, gprsUser, gprsPass)) {// connecting using the apn

Serial.println("GPRS connected");

return true;

}

Serial.println("GPRS connection failed, retrying...");// if not connected wait for 5 sec and retry

delay(5000);

}

return false;// return false

}

bool connectMQTT() {// connected to the mqtt

Serial.print("Connecting to MQTT broker: ");

Serial.println(broker);

for (int i = 0; i < 5; i++) { // Retry 5 times

if (mqttClient.connect("ArduinoClient", mqttUsername, mqttPassword)) {// connecting to the mqt using the username and password (ArduinoClient)is the name of the client

Serial.println("MQTT connected");

mqttClient.subscribe(topicOutput1);// subscribe to the topic to get msg publish in the topic

//mqttClient.subscribe(topicState);

return true;

}

//fails retry 5 times in the interval of 5 sec

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

Serial.print(mqttClient.state());

Serial.println(" retrying...");

delay(5000);

}

return false;

}

void reconnectMQTT() {

while (!mqttClient.connected()) {// check the mqtt is not connected

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

if (connectMQTT()) {// connected to mqtt

Serial.println("connected");

} else {// fails retry every 5 sec

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

Serial.print(mqttClient.state());

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

delay(5000);

}

}

}

// read data from the mqtt broker

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

String message;

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

message += (char)payload[i];//add the data to messge string

}

Serial.print("Message arrived [");

Serial.print(topic);

Serial.print("]: ");

Serial.println(message);

if (String(topic) == topicOutput1) {//check for the messge published topic and subscribed topic are same

if (message == "ON") {// the is ON on the buzzer

digitalWrite(buzzerPin, HIGH);

Serial.println("Buzzer ON via MQTT");

SendMessage("Buzzer on mqtt",phoneNumber);

} else if (message == "OFF") {// if the mmsg is off off the buzzer

digitalWrite(buzzerPin, LOW);

Serial.println("Buzzer OFF via MQTT");

SendMessage("Buzzer off mqtt",phoneNumber);

}

}

}

void handleInterrupt() {// interupt handling

first\_interupt\_flag++;//add 1

if(first\_interupt\_flag>3)//the interupt gets activatd only after the first because the initilization of gsm trigger the gsm at first time

interruptFlag = true; // Set interrupt flag when the interrupt occurs

}

// for interupt we cannot able to do all operation during interupt spo i set flag

void handleGSMEvent() {

//Disconnect MQTT to handle GSM event

mqttClient.disconnect();

delay(1000);

Serial.println("Handling GSM event...");

}

void SendMessage(String message,String phoneNumber) {

gsmSerial.println("AT+CMGF=1"); // Sets the GSM Module in Text Mode

delay(1000); // Delay of 1000 milliseconds or 1 second

gsmSerial.println("AT+CMGS=\"" + phoneNumber + "\"\r"); // Use the provided phone number

delay(1000);

gsmSerial.println(message); // The SMS text you want to send

delay(100);

Serial.println("Finish");

gsmSerial.println((char)26); // ASCII code of CTRL+Z

delay(1000);

Serial.println("-> SMS Sent");

}

### Code Explanation

This code controls a buzzer using an Arduino, a SIM800 GSM module, and an MQTT protocol. The buzzer can be turned on or off through incoming calls, SMS, and MQTT messages. Let's break down the code and explain how it works, including the use of variables s and c, and how the interrupt is handled.

#### 1. ****Variable Declarations:****

* **interruptPin**: The pin connected to the interrupt signal (pin 2).
* **interruptFlag**: A flag to indicate if an interrupt has occurred.
* **callActive:** A flag to indicate if a call is currently active.
* **s, c, i:** These are integer variables used for various control purposes in the code.
* **buzzerPin:** The pin connected to the buzzer (pin 6).

#### 2. ****Library Inclusions:****

* **SoftwareSerial:** To create a serial communication interface for the GSM module on pins 7 (RX) and 8 (TX).
* **TinyGsmClient and PubSubClient:** These libraries handle GSM communication and MQTT messaging.

#### 3. ****Setup Function:****

**Pin Configurations:**

* + pinMode(interruptPin, INPUT): Sets the interrupt pin as an input.
  + pinMode(buzzerPin, OUTPUT): Sets the buzzer pin as an output.

**Interrupt Setup**:

* + attachInterrupt(digitalPinToInterrupt(interruptPin), handleInterrupt, CHANGE): Attaches an interrupt to the interrupt pin. It triggers on any change (rising or falling edge), executing the handleInterrupt() function.

**GSM Initialization**:

* + initializeGSM(): Initializes the GSM module.
  + connectGPRS(): Connects to the GPRS network using the provided APN.
  + connectMQTT(): Connects to the MQTT broker and subscribes to the topics.

#### 4. ****Loop Function:****

**MQTT Handling**:

* + The code checks if the MQTT connection is active. If not, it attempts to reconnect using reconnectMQTT().

**Interrupt Handling:**

* + If the interruptFlag is set (due to an interrupt), the code processes the GSM event by calling handleGSMEvent(), which manages incoming calls or SMS.

**GSM Event Handling**:

* + If s is set to 1 (indicating that a GSM event needs to be handled), the code reads incoming data from the GSM module.
  + If an incoming call is detected (incomingData.indexOf("RING") != -1), the buzzer toggles its state based on the value of c. The c variable alternates between odd and even, allowing the buzzer to be turned on and off with each call.

**SMS Handling**:

* + The code checks if an SMS contains "ON" or "OFF" and turns the buzzer on or off accordingly. An SMS response is sent to confirm the buzzer's state.

#### 5. ****Function Implementations:****

**initializeGSM():** Initializes the GSM module by restarting it and checking the modem information.

**connectGPRS():** Connects to the GPRS network, retrying up to 5 times if it fails.

**connectMQTT():** Connects to the MQTT broker, retrying up to 5 times if it fails.

**reconnectMQTT():** Repeatedly tries to reconnect to the MQTT broker if the connection is lost.

**mqttCallback():** Handles incoming MQTT messages and controls the buzzer based on the message content ("ON" or "OFF").

**handleInterrupt():** This function is triggered by the interrupt. It increments i and sets the interruptFlag to true when i exceeds 2, indicating an event that needs to be processed.

**handleGSMEvent():** Disconnects MQTT, processes the GSM event, and then reconnects MQTT.

**SendMessage():** Sends an SMS with a specified message.

#### 6. ****Variables**** s ****and**** c****:****

**s:** This variable is used to control the state of the GSM event handler. When handleGSMEvent() is called, s is set to 1, indicating that the loop should process GSM events.

**c:** This variable is used to alternate the state of the buzzer during incoming calls. If c is odd, the buzzer is turned on; if c is even, the buzzer is turned off. c is incremented each time an incoming call is detected.

#### 7. ****Interrupt Handling in the Code:****

* The interrupt is triggered on the interruptPin when there's a change in state (rising or falling edge). The handleInterrupt() function is executed, which increments i. When i exceeds 2, the interruptFlag is set to true, signaling the main loop to handle a GSM event by calling handleGSMEvent().

#### 8. ****Events That Trigger the Buzzer:****

* **Incoming Call**: If an incoming call is detected, the buzzer toggles between on and off states.
* **SMS**: If an SMS contains "ON" or "OFF", the buzzer is turned on or off accordingly.
* **MQTT Message**: If an MQTT message contains "ON" or "OFF", the buzzer is turned on or off accordingly.

The code utilizes the **Ring Indicator (RI)** pin of the GSM module for detecting interrupts. The RI pin is connected to interruptPin on the Arduino. The RI pin behaves as follows:

* **During Initialization**: The RI pin normally goes high when the GSM module is initialized. This causes an interrupt to be triggered immediately after setup.
* **Handling the Initial Interrupt**: To avoid handling this initial, unintentional interrupt, the code uses a counter variable i. Initially, i is set to 1. In the handleInterrupt() function, i is incremented each time the interrupt is triggered. The interrupt is only considered valid and handled when i exceeds 2, effectively ignoring the first two interruptions.

This ensures that the first, unintended interrupt caused by the RI pin going high during GSM initialization is ignored, and only subsequent, meaningful interrupts (like an incoming call or SMS) are processed.

**PROGRAM FLOW:**

**1. Global Variables and Setup**

**Interrupt Handling:**

* interruptPin = 2: Sets pin 2 as the interrupt pin.
* volatile bool interruptFlag = false: A flag to signal when an interrupt has occurred.
* int s, c = 1: Flags for handling specific GSM events (s) and keeping track of call occurrences (c).
* int i = 1: Initialization flag to avoid handling the first interrupt caused by the GSM module's initialization.

**Libraries and Definitions:**

#define TINY\_GSM\_MODEM\_SIM800: Specifies the use of the SIM800 GSM module.

#include <SoftwareSerial.h>: Enables serial communication on arbitrary digital pins (pin 7 for RX, pin 8 for TX).

#include <TinyGsmClient.h>: Handles GSM and GPRS communication.

#include <PubSubClient.h>: Manages MQTT communication with the broker.

**Serial Communication Setup:**

SoftwareSerial: gsmSerial(7, 8) initializes software serial communication with the GSM module.

Debugging: #define TINY\_GSM\_DEBUG Serial allows debugging information to be printed to the serial monitor.

**Buzzer Pin Setup:**

const int buzzerPin = 6: Sets pin 6 as the buzzer pin, which will be controlled by the program.

**GSM and MQTT Settings:**

const char\* broker, mqttUsername, mqttPassword, apn, etc., define the broker, APN, and credentials for GPRS and MQTT communication.

**GSM and MQTT Clients:**

* TinyGsm modem(gsmSerial): Initializes the GSM modem.
* TinyGsmClient gsmClient(modem): Associates the GSM modem with the GSM client.
* PubSubClient mqttClient(gsmClient): Sets up the MQTT client to use the GSM client for communication.

**2. Setup Function**

**Pin Configuration:**

pinMode(interruptPin, INPUT): Sets the interrupt pin as input.

pinMode(buzzerPin, OUTPUT): Sets the buzzer pin as output.

Serial Communication Initialization:

Serial.begin(9600): Starts serial communication at 9600 baud for debugging.

**Interrupt Attachment:**

attachInterrupt(digitalPinToInterrupt(interruptPin), handleInterrupt, CHANGE): Attaches an interrupt to the specified pin that triggers on any state change (rising or falling edge).

**GSM Initialization:**

gsmSerial.begin(9600): Initializes the GSM module communication at 9600 baud.

initializeGSM(): Restarts the GSM modem and retrieves modem information. This step is crucial as it resets the GSM modem to a clean state, which triggers the RI (Ring Indicator) pin.

**GPRS Connection:**

connectGPRS(): Connects to the internet using the provided APN settings, retrying up to 5 times.

**MQTT Connection:**

mqttClient.setServer(broker, 1883): Sets up the MQTT broker and port.

mqttClient.setCallback(mqttCallback): Defines the callback function to handle incoming MQTT messages.

connectMQTT(): Connects to the MQTT broker, retrying up to 5 times.

**Completion Message:**

Serial.println("Setup complete"): Indicates that the setup is finished.

### 3. ****Loop Function (continued)****

#### ****Incoming Call Detection:****

* if (incomingData.indexOf("RING") != -1): Detects if the incoming data string contains the word "RING," which indicates an incoming call.
  + **Buzzer Control:**
    - if (c % 2 == 1): If c is odd, the buzzer is turned on:
      * digitalWrite(buzzerPin, HIGH): Activates the buzzer.
      * Serial.println("buzzer on"): Prints a message to the serial monitor indicating the buzzer is on.
      * SendMessage("buzzer on"): Sends an SMS to notify that the buzzer is turned on.
    - else: If c is even, the buzzer is turned off:
      * digitalWrite(buzzerPin, LOW): Deactivates the buzzer.
      * Serial.println("buzzer off"): Prints a message to the serial monitor indicating the buzzer is off.
      * SendMessage("buzzer off"): Sends an SMS to notify that the buzzer is turned off.
      * delay(2000): Adds a 2-second delay to allow time for the GSM module to process.
      * reconnectMQTT(): Reconnects to the MQTT broker after handling the call.
    - c++: Increments the call counter to alternate between turning the buzzer on and off with each call.

#### ****SMS Handling:****

**Buzzer Control via SMS:**

* + else if (incomingData.indexOf("ON") != -1): Checks if the incoming SMS contains the word "ON":
    - digitalWrite(buzzerPin, HIGH): Turns the buzzer on.
    - Serial.println("Buzzer on sms"): Prints a message to the serial monitor indicating the buzzer is on via SMS.
    - SendMessage("Buzzer on sms"): Sends a confirmation SMS that the buzzer is turned on.
  + else if (incomingData.indexOf("OFF") != -1): Checks if the incoming SMS contains the word "OFF":
    - digitalWrite(buzzerPin, LOW): Turns the buzzer off.
    - Serial.println("Buzzer off sms"): Prints a message to the serial monitor indicating the buzzer is off via SMS.
    - SendMessage("Buzzer off sms"): Sends a confirmation SMS that the buzzer is turned off.
    - delay(3000): Adds a 3-second delay to ensure smooth handling of the GSM module.
    - reconnectMQTT(): Reconnects to the MQTT broker after processing the SMS.
* s = 0: Resets the flag s to 0, indicating that the GSM event has been handled.

### 4. ****GSM Initialization (initializeGSM)****

* modem.restart(): Resets the GSM module to ensure it starts in a clean state.
* delay(1000): Adds a 1-second delay to allow the GSM module to stabilize.
* **Modem Information:**
  + String modemInfo = modem.getModemInfo(): Retrieves information about the GSM module.
  + Serial.print("Modem Info: "); Serial.println(modemInfo): Prints the modem information to the serial monitor.
* return true: Returns true to indicate successful initialization.

### 5. ****GPRS Connection (connectGPRS)****

* Serial.print("Connecting to APN: "); Serial.println(apn): Prints the APN being used for the GPRS connection.
* **Connection Attempt Loop:**
  + for (int i = 0; i < 5; i++): Tries to connect to GPRS up to 5 times.
  + if (modem.gprsConnect(apn, gprsUser, gprsPass)): Attempts to connect using the APN, username, and password.
    - Serial.println("GPRS connected"): Prints a success message if connected.
    - return true: Returns true to indicate successful GPRS connection.
  + **Retry Logic:**
    - Serial.println("GPRS connection failed, retrying..."): Prints a retry message if the connection fails.
    - delay(5000): Waits 5 seconds before retrying.
* return false: Returns false if the connection fails after 5 attempts.

### 6. ****MQTT Connection (connectMQTT)****

* Serial.print("Connecting to MQTT broker: "); Serial.println(broker): Prints the MQTT broker being connected to.
* **Connection Attempt Loop:**
  + for (int i = 0; i < 5; i++): Tries to connect to the MQTT broker up to 5 times.
  + if (mqttClient.connect("ArduinoClient", mqttUsername, mqttPassword)): Attempts to connect to the broker with the given credentials:
    - Serial.println("MQTT connected"): Prints a success message if connected.
    - mqttClient.subscribe(topicOutput1): Subscribes to the specified topic to receive messages.
    - return true: Returns true to indicate a successful MQTT connection.
  + **Retry Logic:**
    - Serial.print("MQTT connection failed, rc="); Serial.print(mqttClient.state()); Serial.println(" retrying..."): Prints the reason for the connection failure and retries after a 5-second delay.
* return false: Returns false if the connection fails after 5 attempts.

### 7. ****Reconnect MQTT (reconnectMQTT)****

* **Reconnection Loop:**
  + while (!mqttClient.connected()): Continuously attempts to reconnect to the MQTT broker if the connection is lost.
  + Serial.print("Attempting MQTT connection..."): Prints a message indicating an attempt to reconnect.
  + **Successful Reconnection:**
    - if (connectMQTT()): Calls the connectMQTT function to attempt a connection:
      * Serial.println("connected"): Prints a success message if reconnected.
  + **Retry on Failure:**
    - else: If the reconnection fails:
      * Serial.print("failed, rc="); Serial.print(mqttClient.state()); Serial.println(" try again in 5 seconds"): Prints the failure reason and retries after a 5-second delay.

### 8. ****MQTT Callback (mqttCallback)****

* **Message Processing:**
  + void mqttCallback(char\* topic, byte\* payload, unsigned int length): Handles incoming MQTT messages.
  + **Message Construction:**
    - String message; for (unsigned int i = 0; i < length; i++) { message += (char)payload[i]; }: Constructs the message string from the payload.
  + Serial.print("Message arrived ["); Serial.print(topic); Serial.print("]: "); Serial.println(message): Prints the received topic and message.
  + **Buzzer Control via MQTT:**
    - if (String(topic) == topicOutput1): Checks if the topic matches the expected topic for controlling the buzzer:
      * if (message == "ON"): Turns the buzzer on if the message is "ON":
        + digitalWrite(buzzerPin, HIGH); Serial.println("Buzzer ON via MQTT").
      * else if (message == "OFF"): Turns the buzzer off if the message is "OFF":
        + digitalWrite(buzzerPin, LOW); Serial.println("Buzzer OFF via MQTT").

### 9. ****Interrupt Handling (handleInterrupt)****

* **Interrupt Logic:**
  + void handleInterrupt(): Handles the interrupt triggered by the GSM module.
  + i++: Increments the initialization flag.
  + if (i > 2): Ensures the interrupt is only processed after the first two (initialization-related) triggers:
    - interruptFlag = true: Sets the interrupt flag to indicate that an interrupt has occurred.

### 10. ****GSM Event Handling (handleGSMEvent)****

* **GSM Event Processing:**
  + int handleGSMEvent(): Handles the event triggered by the GSM module (like an incoming call or SMS).
  + mqttClient.disconnect(): Disconnects from the MQTT broker to prioritize GSM event handling.
  + delay(1000): Adds a delay to ensure the GSM module has time to process the event.
  + Serial.println("Handling GSM event..."): Prints a message indicating that a GSM event is being handled.
  + s = 1: Sets the flag s to 1 to indicate that a GSM event is in progress.
  + return s: Returns the value of s.

### 11. ****Sending SMS (SendMessage)****

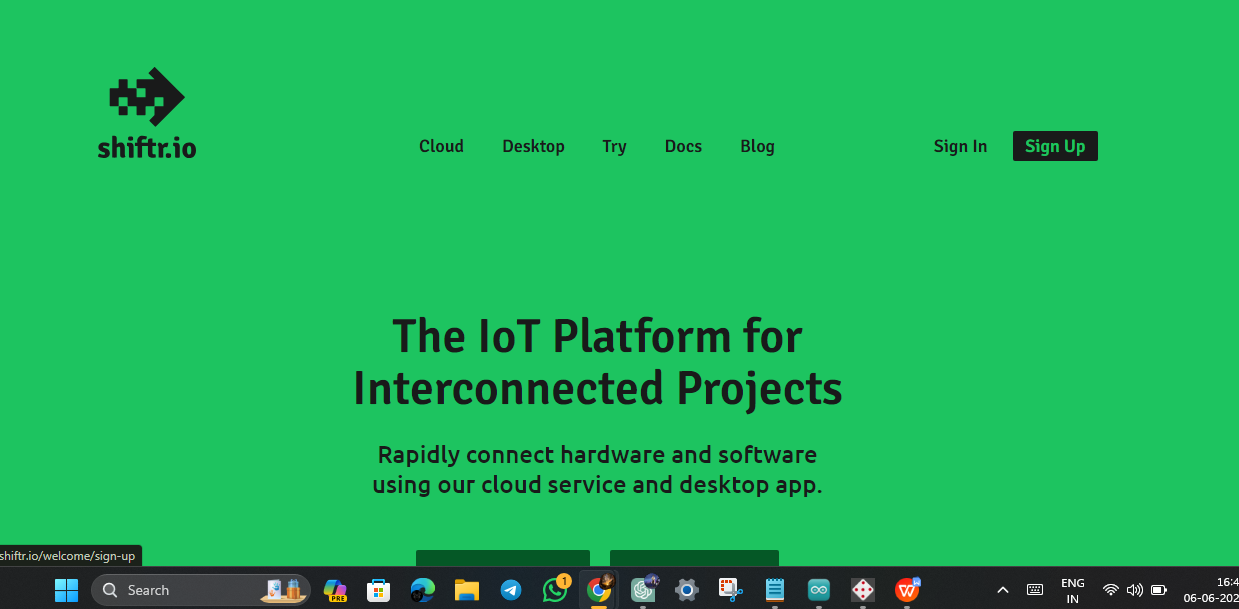
* **SMS Sending Process:**
  + void SendMessage(String message): Sends an SMS with the specified message.
  + gsmSerial.println("AT+CMGF=1"): Sets the GSM module to text mode.
  + delay(1000): Waits for the command to be processed.
  + gsmSerial.println("AT+CMGS=\"+919080991747\"\r"): Sends the SMS command with the recipient's phone number.
  + delay(1000): Waits for the command to be processed.
  + gsmSerial.println(message): Sends the message content.
  + delay(100): Waits briefly before finishing the SMS.
  + gsmSerial.println((char)26): Sends the ASCII code for CTRL+Z to indicate the end of the SMS.
  + delay(1000): Waits for the message to be sent.

**STEPS TO CREATE A PROJECT**

**Step 1: Set Up Shiftr.io**

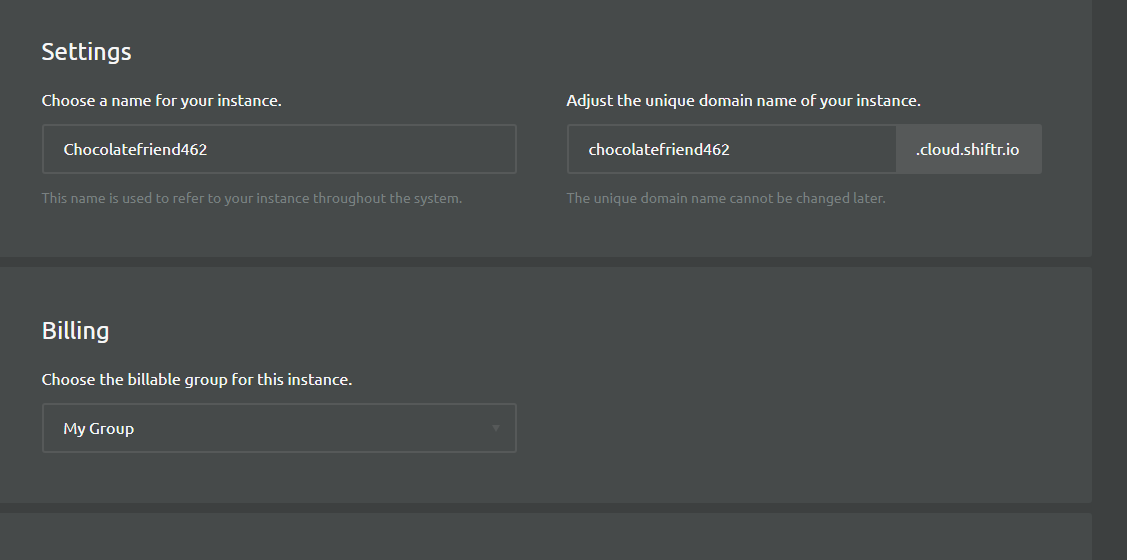
**Create an Account:**

* Visit Shiftr.io and create an account.



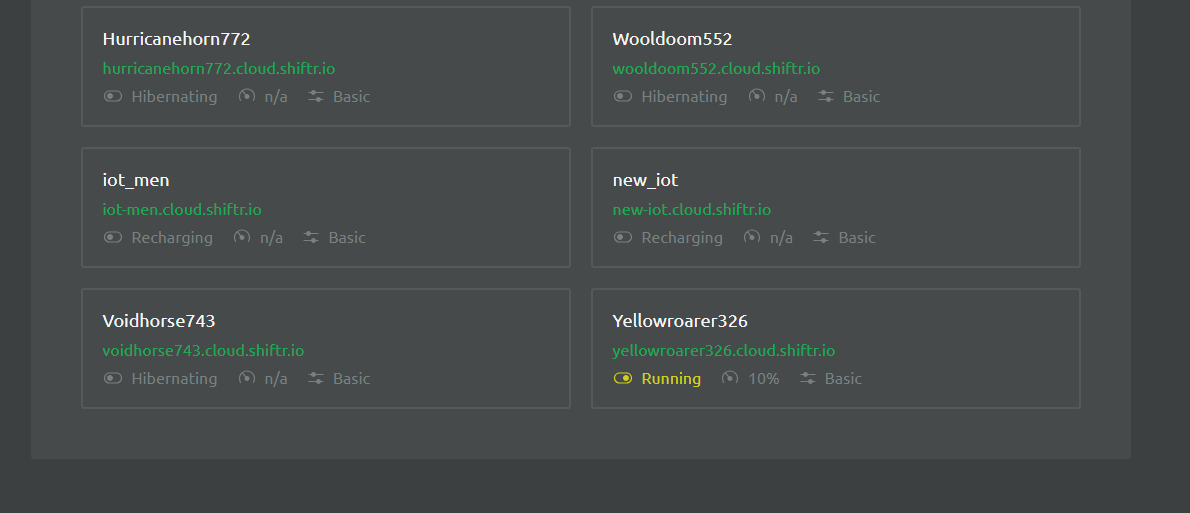
**Create a New Space:**

* After logging in, create a new space. This space will be your MQTT broker.

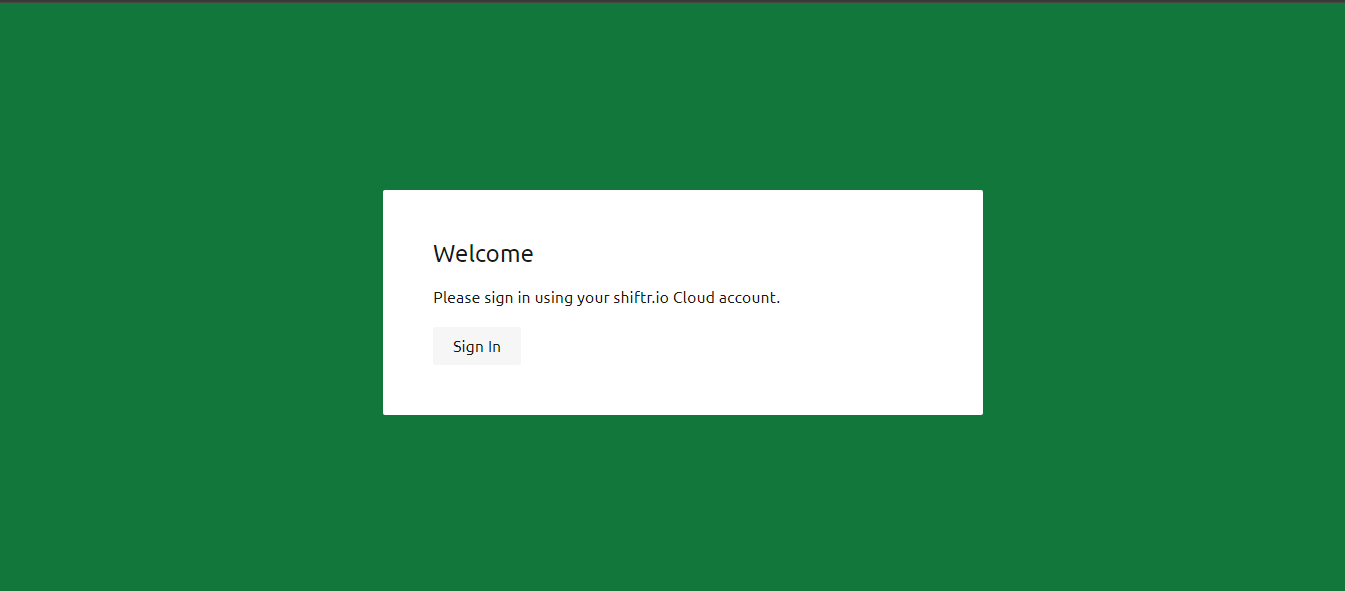


**Get Connection Details:**

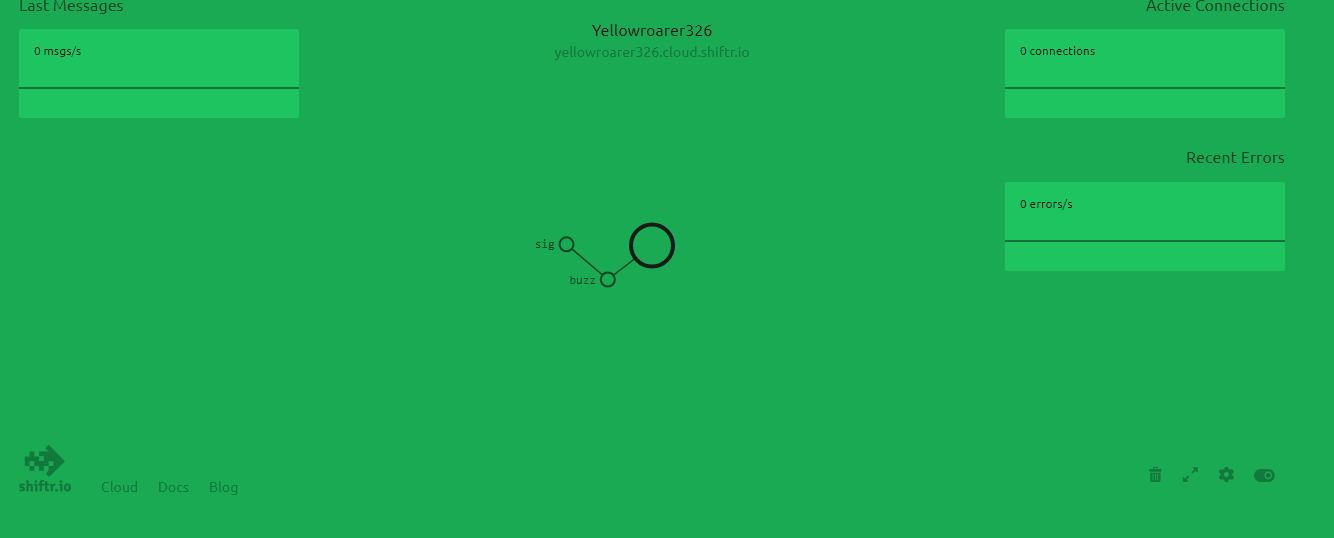
* Note the broker URL, username, and password provided by Shiftr.io. You will need these for your ESP32 and MQTT Box setup.
* Click the url of the instance



* Sign in



* You will go to the broker connection page



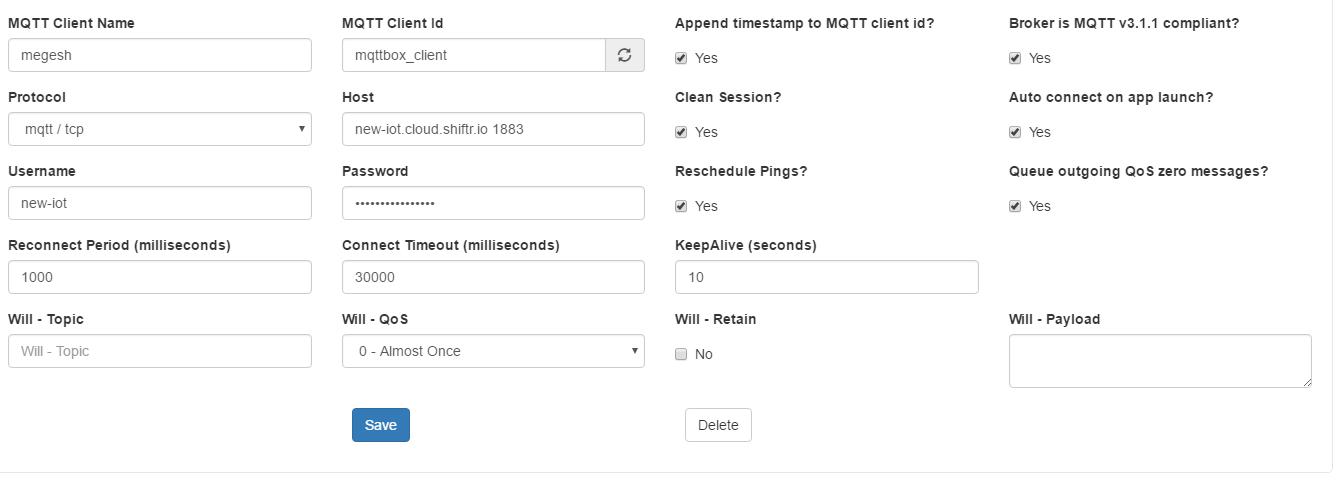
**Step 2: Set Up MQTT Box**

**Install MQTT Box:**

* Download and install MQTT Box from MQTT Box.

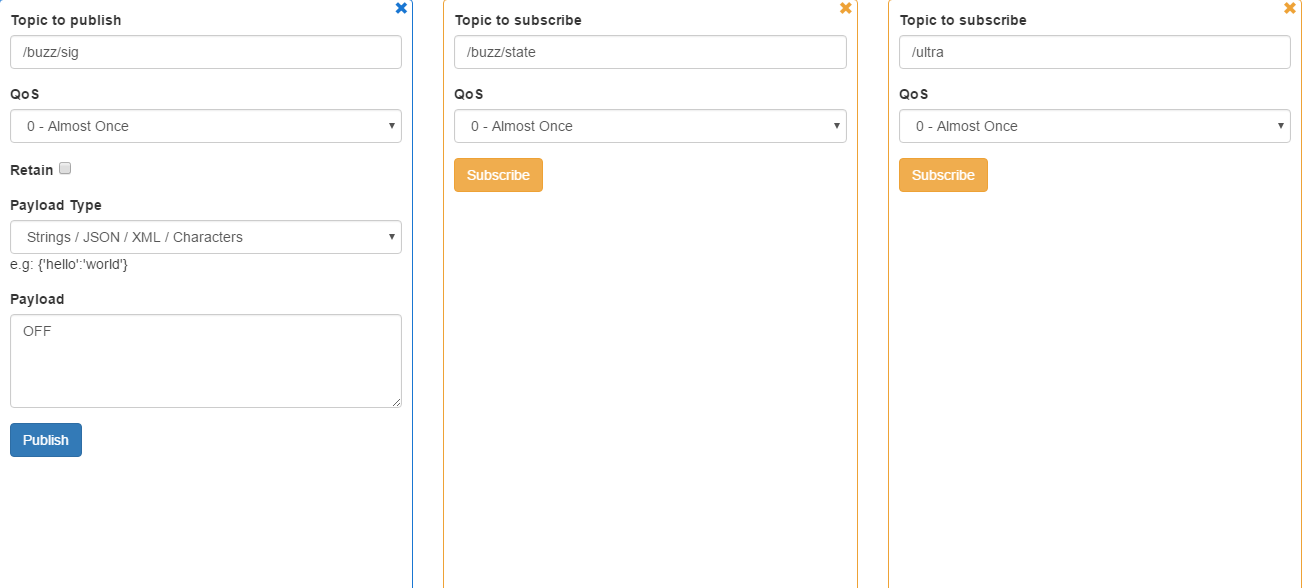
**Create a New Connection:**

* Open MQTT Box and create a new connection.
* Enter the broker URL, username, and password you got from Shiftr.io.

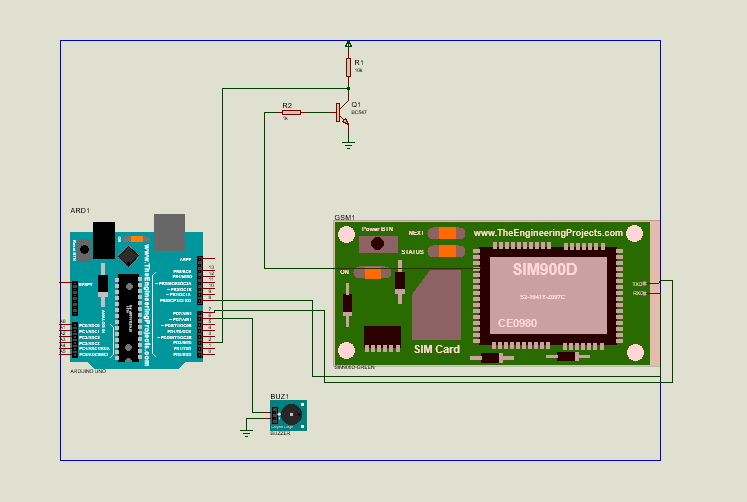


**Subscribe to Topics:**

* Subscribe to the following topics to monitor and control the buzzer:



**CIRCUIT DIAGRAM:**



**CONCLUSION:**

This project effectively integrates GSM communication with MQTT to remotely control a buzzer using an Arduino. The key functionalities include:

* **GSM Module Integration**: The SIM800A module is utilized for handling incoming calls and SMS, allowing the user to control the buzzer through both voice and text messages. The program is designed to prioritize GSM events (such as calls and SMS) over MQTT operations by using an interrupt-driven approach.
* **MQTT Communication**: The Arduino connects to an MQTT broker, allowing remote control of the buzzer via MQTT messages. The system subscribes to a specific topic, enabling the buzzer to be turned on or off based on the received MQTT commands.
* **Interrupt Handling**: The project effectively manages interrupts triggered by the GSM module, ensuring that the system can promptly respond to incoming calls or SMS. The use of the UART1\_RI pin as an interrupt source allows the program to detect these events without constant polling, making it more efficient.
* **Buzzer Control**: The buzzer can be controlled through three methods—MQTT messages, incoming calls, and SMS commands. This flexibility provides multiple ways to interact with the system, enhancing its utility in different scenarios.
* **Reliability and Robustness**: The program includes mechanisms to handle GSM module initialization, GPRS connection for internet access, and MQTT broker reconnection, ensuring the system remains operational even after disruptions.

Overall, the project demonstrates a comprehensive approach to integrating GSM and MQTT for remote device control. It is a practical solution for applications requiring remote monitoring and control, such as in home automation, security systems, or industrial monitoring.