**Week 8 Lab 5 Report**

**Name:** Javon Teo Tze Kai

**Student ID:** A0233706J

**Lab Group:** B02

**Task 1:**

“b” indicates that our Hello! message string is in bytes literal format, it means that the string is represented as a sequence of bytes.

**Task 2:**

def on\_message(client, userdata, message):

    print("Received message " + message.payload.decode())

**Task 3:**

Given that the mqtt.py code had:

def on\_connect(client, userdata, flags, rc):

...

    client.publish("hello/world", "This is a test")

I modified the HelloToMyself.ino:

char \*subscribeTopic = "hello/world";

char \*publishTopic = "hello/esp";

By changing the subscribeTopic in the HelloToMyself.ino, such that it subscribes to the topic “hello/world”, the data would then be sent to the ESP32 from the pub.py python script.

**Task 4:**

ESP32 sketch:

char \*publishTempTopic = "weather/temp";

char \*publishHumidTopic = "weather/humidity";

struct measurements {

  float temperature;

  float humidity;

};

void loop()

{

    struct measurements mmts = dht11\_loop(); // collects data DHT11 sensor.

    mqttClient.publish(publishTempTopic, String(mmts.temperature), 0, false);

    mqttClient.publish(publishHumidTopic, String(mmts.humidity), 0, false);

    delay(2000);

}

mqtt.py python script:

def on\_connect(client, userdata, flags, rc):

    client.subscribe("weather/temp")

    client.subscribe("weather/humidity")

def on\_message(client, userdata, message):

    if message.topic == "weather/temp":

        print(f"Received message: {message.payload.decode()} °C on topic: {message.topic}")

    else:

        print(f"Received message: {message.payload.decode()} % on topic: {message.topic}")

**Task 5:**

States:

* Deep Sleep mode has power consumption of 10-150 µA. The device would last 1333.33h – 20000h.
* Modem-sleep mode has power consumption of 27mA to 44mA . The device would last for 4.54h – 7.41h.
* Active mode has power consumption of 95 - 240mA. The device would last for 0.833h – 2.10h.

**Task 6:**

#define uS\_TO\_S\_FACTOR 1000000ULL

#define TIME\_TO\_SLEEP  20

char \*subscribeTopic = "hello/world";

char \*publishTopic = "hello/esp32";

ESP32MQTTClient mqttClient;

void setup(){

  Serial.begin(115200);

  esp\_sleep\_enable\_timer\_wakeup(TIME\_TO\_SLEEP \* uS\_TO\_S\_FACTOR);

  log\_i();

  log\_i("setup, ESP.getSdkVersion(): ");

  log\_i("%s", ESP.getSdkVersion());

  mqttClient.enableDebuggingMessages();

  mqttClient.setURI(server);

  mqttClient.enableLastWillMessage("lwt", "I am going offline");

  mqttClient.setKeepAlive(30);

  WiFi.begin(ssid, pass);

  WiFi.setHostname("c3test");

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

    Serial.print('.');

    delay(1000);

  }

  Serial.println("CONNECTED");

  mqttClient.loopStart();

}

void loop(){

  delay(5000);

  // Sends MQTT message

  mqttClient.publish(publishTopic, "Hello from ESP32!!!!", 0, false);

  // Prints to Serial that it has sent the message

  Serial.println("SENT");

  // We sent ESP32 to sleep.

  Serial.println("Going to sleep now");

  delay(5000);

  esp\_deep\_sleep\_start(); // This makes ESP32 go to sleep.

}

The ESP32 is set to go to deep-sleep every 20 seconds. When it wakes up, it firstly sets the parameters of the MQTT client, then connects to the WiFi network with the provided SSID and password. It then goes to the loop(), where after 5 seconds delay, sends an MQTT message (“Hello from ESP32!!!!”), prints “SENT” to the Serial, then prints “Going to sleep now” to the Serial before setting the ESP32 to deep sleep.

**Task 7:**

def on\_message(client, userdata, message):

    if message.topic == "weather/temp":

        float\_temp = float(message.payload.decode('utf-8'))

        reply = classify\_temp(float\_temp)

        if reply:

            client.publish("weather/temp", reply)

def classify\_temp(temp):

    reply = None

    if temp > 30:

        print("Too hot! Opening window...")

        reply = 0

    elif temp < 25:

        print("Too cold! Closing window...")

        reply = 1

    else:

        print("Between 25-30. Partially opening window...")

        reply = 2

    return reply

**Task 8:**

Arduino Code:

char \*subscribeTempTopic = "classification/temp";

char \*publishTempTopic = "weather/temp";

String classification = "";

int pos = 0;

int servoPin = 13;

void setup()

{

    Serial.begin(115200);

    esp\_sleep\_enable\_timer\_wakeup(TIME\_TO\_SLEEP \* uS\_TO\_S\_FACTOR);

    dht.begin();

    sensor\_t sensor;

    delayMS = sensor.min\_delay / 1000;

    // Sets up the Servo

    ESP32PWM::allocateTimer(0);

    ESP32PWM::allocateTimer(1);

    ESP32PWM::allocateTimer(2);

    ESP32PWM::allocateTimer(3);

    myservo.setPeriodHertz(50);    // standard 50 hz servo

    myservo.attach(servoPin, 1000, 2000); // attaches the servo on pin 13 to the servo object

    log\_i();

    log\_i("setup, ESP.getSdkVersion(): ");

    log\_i("%s", ESP.getSdkVersion());

    mqttClient.enableDebuggingMessages();

    mqttClient.setURI(server);

    mqttClient.enableLastWillMessage("lwt", "I am going offline");

    mqttClient.setKeepAlive(30);

    WiFi.begin(ssid, pass);

    WiFi.setHostname("c3test");

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

      Serial.print('.');

      delay(1000);

    }

    Serial.println("CONNECTED");

    mqttClient.loopStart();

}

void loop()

{

    struct measurements mmts = dht11\_loop(); // collects data from DHT11.

    Serial.print("mmts:");

    Serial.println(String(mmts.temperature));

    mqttClient.publish(publishTempTopic, String(mmts.temperature), 0, false);

    delay(5000);

    actuateServo(classification); // actuates the servo.

    Serial.println("Going to sleep now");

    delay(5000);

    esp\_deep\_sleep\_start(); // This makes ESP32 go to sleep.

}

struct measurements dht11\_loop() {

  delay(delayMS);

  struct measurements mmts;

  sensors\_event\_t event;

  dht.temperature().getEvent(&event);

  if (isnan(event.temperature)) {

    Serial.println(F("Error reading temperature!"));

  }

  else {

    mmts.temperature = event.temperature;

  }

  dht.humidity().getEvent(&event);

  if (isnan(event.relative\_humidity)) {

    Serial.println(F("Error reading humidity!"));

  }

  else {

    mmts.humidity = event.relative\_humidity;

  }

  return mmts;

}

void actuateServo(String reply) {

  if (reply == "HOT") {

    if (pos != 0) {

      myservo.write(0);

      pos = 0;

    }

  } else if (reply == "COLD") {

    if (pos != 180) {

      myservo.write(180);

      pos = 180;

    }

  } else if (reply == "BETWEEN") {

    if (pos != 90) {

      myservo.write(90);

      pos = 90;

    }

  } else {

  }

}

void onConnectionEstablishedCallback(esp\_mqtt\_client\_handle\_t client)

{

    if (mqttClient.isMyTurn(client)) // can be omitted if only one client

    {

        mqttClient.subscribe(subscribeTempTopic, [](const String &payload)

                             {

                              Serial.println(String(subscribeTempTopic)+String("")+String(payload.c\_str()));

                              classification = String(payload.c\_str());

                              });

}

mqtt.py

import paho.mqtt.client as mqtt

from time import sleep

def on\_connect(client, userdata, flags, rc):

    client.subscribe("weather/temp")

def on\_message(client, userdata, message):

    float\_temp = float(message.payload.decode('utf-8'))

    reply = classify\_temp(float\_temp)

    if reply != None:

        client.publish("classification/temp", reply)

def classify\_temp(temp):

    reply = None

    if temp > 30:

        print("Too hot! Opening window...")

        reply = "HOT"

    elif temp < 25:

        print("Too cold! Closing window...")

        reply = "COLD"

    else:

        print("Between 25-30. Partially opening window...")

        reply = "BETWEEN"

    return reply

client = mqtt.Client()

client.on\_connect = on\_connect

client.on\_message = on\_message

client.connect("localhost", 1883, 60)

client.loop\_forever()

The setup function is used to set up all the peripherals of the system, such as the servo, the dht11, the MQTT client and the deep sleep functionality of the ESP32. It then moves on to the loop function, it senses the temperature via the DHT11, then publishes the temperature to the “weather/temp” channel. The mqtt.py is able to read in this message, and classifies and replies back with “HOT”, “COLD” or “BETWEEN”. The ESP32 then takes in the message (published in “classification/temp” channel), then runs actuateServo to actuate the resultant action on the servo. The ESP32 is then sent into deep-sleep mode, and then this process repeats when it wakes up again.