IOT- FIRMWARE DEVELOPMENT

R MADESHWAR
(CEG-Anna University)

1.Write an Esp32-based code using an esp32 microcontroller that has the following tasks
a. Read MPU6050 sensor data at 200 readings per second intervals and GPS sensor data at 1-second intervals from ESP32
b. Read temperature sensor data at every 5-second interval from the Arduino microcontroller.
c. Transfer the collected sensor data from Arduino to Esp32 via UART protocol
d. Publish the combined sensor data in JSON format above to the AWS MQTT or any other MQTT endpoint every 5 seconds.

```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BMP280.h>
#include <MPU6050_tockn.h>
#include <TinyGPS++.h>
#include <WiFi.h>
#include <PubSubClient.h>

// Define pins for I2C communication
#define SDA_PIN 21
#define SCL_PIN 22

// WiFi credentials
const char* ssid = "your-SSID";
const char* password = "your-PASSWORD";

// MQTT Broker information
const char* mqtt_server = "your-MQTT-broker-URL";
const int mqtt_port = 1883;
const char* mqtt_username = "your-MQTT-username";
const char* mqtt_password = "your-MQTT-password";
const char* mqtt_topic = "your-MQTT-topic";

// Create objects for sensors
Adafruit_BMP280 bmp;
MPU6050 mpu6050(Wire);

// Create object for GPS sensor
TinyGPSPlus gps;
```

```
// Define interval constants
const unsigned long mpuInterval = 5000;  // 200 readings per second
const unsigned long gpsInterval = 1000;  // 1 reading per second
const unsigned long tempInterval = 5000; // 1 reading every 5 seconds
const unsigned long mqttInterval = 5000; // 1 publish every 5 seconds

unsigned long mpuPreviousMillis = 0;
unsigned long gpsPreviousMillis = 0;
unsigned long tempPreviousMillis = 0;
unsigned long mqttPreviousMillis = 0;

WiFiClient espClient;
PubSubClient client(espClient);

void setup() {
  Serial.begin(115200);
  Wire.begin(SDA_PIN, SCL_PIN);

  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("Connected to WiFi");

  // MQTT setup
  client.setServer(mqtt_server, mqtt_port);
  client.setCredentials(mqtt_username, mqtt_password);

  if (!bmp.begin()) {
    Serial.println(F("Could not find a valid BMP280 sensor, check
wiring!"));
    while (1);
  }

  if (!mpu6050.begin(MPU6050_SCALE_2000DPS, MPU6050_RANGE_2G)) {
    Serial.println("Could not find a valid MPU6050 sensor, check
wiring!");
    while (1);
  }
```

```
  mpu6050.calibrateGyro();

  Serial.println("Initialization complete.");
}

void loop() {
  unsigned long currentMillis = millis();

  // Read MPU6050 sensor data
  if (currentMillis - mpuPreviousMillis >= mpuInterval) {
    readMPU6050Data();
    mpuPreviousMillis = currentMillis;
  }

  // Read GPS sensor data
  if (currentMillis - gpsPreviousMillis >= gpsInterval) {
    readGPSData();
    gpsPreviousMillis = currentMillis;
  }

  // Read temperature sensor data
  if (currentMillis - tempPreviousMillis >= tempInterval) {
    readTemperatureData();
    tempPreviousMillis = currentMillis;
  }

  // Publish sensor data to MQTT
  if (currentMillis - mqttPreviousMillis >= mqttInterval) {
    publishSensorData();
    mqttPreviousMillis = currentMillis;
  }

  // Maintain MQTT connection
  if (!client.connected()) {
    reconnect();
  }
  client.loop();
}

void readMPU6050Data() {
  mpu6050.update();
  Serial.print("MPU6050 - AccX: ");
  Serial.print(mpu6050.getAccX());
```

```
  Serial.print(", AccY: ");
  Serial.print(mpu6050.getAccY());
  Serial.print(", AccZ: ");
  Serial.print(mpu6050.getAccZ());
  Serial.print(", GyroX: ");
  Serial.print(mpu6050.getGyroX());
  Serial.print(", GyroY: ");
  Serial.print(mpu6050.getGyroY());
  Serial.print(", GyroZ: ");
  Serial.println(mpu6050.getGyroZ());
}

void readGPSData() {
  while (Serial2.available() > 0) {
    if (gps.encode(Serial2.read())) {
      Serial.print("GPS - Latitude: ");
      Serial.print(gps.location.lat(), 6);
      Serial.print(", Longitude: ");
      Serial.println(gps.location.lng(), 6);
    }
  }
}

void readTemperatureData() {
  Serial.print("Temperature: ");
  Serial.println(bmp.readTemperature());
}

void publishSensorData() {
  // Create a JSON string with the sensor data
  String jsonPayload = "{";
  jsonPayload += "\"AccX\":" + String(mpu6050.getAccX()) + ",";
  jsonPayload += "\"AccY\":" + String(mpu6050.getAccY()) + ",";
  jsonPayload += "\"AccZ\":" + String(mpu6050.getAccZ()) + ",";
  jsonPayload += "\"GyroX\":" + String(mpu6050.getGyroX()) + ",";
  jsonPayload += "\"GyroY\":" + String(mpu6050.getGyroY()) + ",";
  jsonPayload += "\"GyroZ\":" + String(mpu6050.getGyroZ()) + ",";
  jsonPayload += "\"Latitude\":" + String(gps.location.lat(), 6) + ",";
  jsonPayload += "\"Longitude\":" + String(gps.location.lng(), 6) +
",";
  jsonPayload += "\"Temperature\":" + String(bmp.readTemperature());
  jsonPayload += "}";
```

```
  // Convert the JSON string to a char array
  char charArray[jsonPayload.length() + 1];
  jsonPayload.toCharArray(charArray, sizeof(charArray));

  // Publish the JSON payload to the MQTT topic
  client.publish(mqtt_topic, charArray);
  Serial.println("Published to MQTT");
}

void reconnect() {
  // Loop until we're reconnected to the MQTT broker
  while (!client.connected()) {
    Serial.println("Attempting MQTT connection...");

    // Attempt to connect
    if (client.connect("ESP32 Client")) {
      Serial.println("Connected to MQTT broker");
      client.subscribe(mqtt_topic);
    } else {
      Serial.print("Failed, rc=");
      Serial.print(client.state());
      Serial.println(" Retrying in 5 seconds");
      delay(5000);
    }
  }
}
```