ESP32 Sensor LoRa Transmitter code

```
#include <LoRa.h>
#include <SPI.h>
#include <OneWire.h>
#include <DallasTemperature.h>

#define ss 5
#define rst 14
#define dio0 2
#define VREF 3.3      // analog reference voltage(Volt) of the ADC
#define SCOUNT  30    // sum of sample point

// GPIO where the DS18B20 is connected to
const int oneWireBus = 4;

// Pin for TDS sensor
#define TdsSensorPin 27

// Water flow sensor constants
#define SENSOR_PIN  32
#define LED_BUILTIN 2

long currentMillis = 0;
long previousMillis = 0;
int interval = 1000;
boolean ledState = LOW;
float calibrationFactor = 5.5;
volatile byte pulseCount;
byte pulse1Sec = 0;
float flowRate;
unsigned int flowMilliLitres;
unsigned long totalMilliLitres;

// Water pressure sensor constants
const float OffSet = 0.550;

// Setup a oneWire instance to communicate with any OneWire devices
OneWire oneWire(oneWireBus);
DallasTemperature sensors(&oneWire);

void IRAM_ATTR pulseCounter() {
  pulseCount++;
}

void setup() {
  Serial.begin(115200);
  sensors.begin();
```

```
  while (!Serial);
  Serial.println("LoRa Sender");

  LoRa.setPins(ss, rst, dio0);    // setup LoRa transceiver module

  while (!LoRa.begin(433E6)) {    // 433E6 - Asia, 866E6 - Europe, 915E6 -
North America
    Serial.println(".");
    delay(500);
  }
  LoRa.setSyncWord(0xA5);
  Serial.println("LoRa Initializing OK!");

  pinMode(LED_BUILTIN, OUTPUT);
  pinMode(SENSOR_PIN, INPUT_PULLUP);

  pulseCount = 0;
  flowRate = 0.0;
  flowMilliLitres = 0;
  totalMilliLitres = 0;
  previousMillis = 0;

  attachInterrupt(digitalPinToInterrupt(SENSOR_PIN), pulseCounter, FALLING);
}

void loop() {
  currentMillis = millis();
  if (currentMillis - previousMillis > interval) {

    pulse1Sec = pulseCount;
    pulseCount = 0;

    // Calculate flow rate
    flowRate = ((1000.0 / (millis() - previousMillis)) * pulse1Sec) /
calibrationFactor;
    previousMillis = millis();

    // Calculate flow in milliliters
    flowMilliLitres = (flowRate / 60) * 1000;
    totalMilliLitres += flowMilliLitres;

    // Temperature reading
    sensors.requestTemperatures();
    float temperatureC = sensors.getTempCByIndex(0);

    // TDS sensor reading
    float tdsValue = readTdsSensor();
```

```
// Water pressure sensor reading
float V = analogRead(27) * 5.00 / 4096.0;     // Sensor output voltage
float P = (V - OffSet) * 250;                  // Calculate water pressure

Serial.print("Flow rate: ");
Serial.print(int(flowRate));  // Print the integer part of the variable
Serial.print(" L/min\t");       // Print tab space

Serial.print("Output Liquid Quantity: ");
Serial.print(totalMilliLitres);
Serial.print(" mL / ");
Serial.print(totalMilliLitres / 1000);
Serial.println(" L");

Serial.print("Temperature: ");
Serial.print(temperatureC);
Serial.println(" C");

Serial.print("TDS Value: ");
Serial.print(tdsValue);
Serial.println(" ppm");

Serial.print("Voltage: ");
Serial.print(V, 3);
Serial.println(" V");

Serial.print("Pressure: ");
Serial.print(P, 1);
Serial.println(" kPa");

Serial.println("");

// Send LoRa packet to receiver
LoRa.beginPacket();
LoRa.print("Flow rate: ");
LoRa.print(int(flowRate));
LoRa.println(" L/min");

LoRa.print("Temperature: ");
LoRa.print(temperatureC);
LoRa.println(" C");

LoRa.print("TDS Value: ");
LoRa.print(tdsValue);
LoRa.println(" ppm");

LoRa.print("Voltage: ");
```

```
    LoRa.print(V, 3);
    LoRa.println(" V");

    LoRa.print("Pressure: ");
    LoRa.print(P, 1);
    LoRa.println(" kPa");

    LoRa.endPacket();
  }
  delay(1000);
}

int analogBuffer[SCOUNT];     // store the analog value in the array, read
from ADC
int analogBufferTemp[SCOUNT];
int analogBufferIndex = 0;
int copyIndex = 0;
float averageVoltage = 0;
float tdsValue = 0;
float temperature = 25;       // current temperature for compensation

// median filtering algorithm
int getMedianNum(int bArray[], int iFilterLen){
  int bTab[iFilterLen];
  for (byte i = 0; i<iFilterLen; i++)
  bTab[i] = bArray[i];
  int i, j, bTemp;
  for (j = 0; j < iFilterLen - 1; j++) {
    for (i = 0; i < iFilterLen - j - 1; i++) {
      if (bTab[i] > bTab[i + 1]) {
        bTemp = bTab[i];
        bTab[i] = bTab[i + 1];
        bTab[i + 1] = bTemp;
      }
    }
  }
  if ((iFilterLen & 1) > 0){
    bTemp = bTab[(iFilterLen - 1) / 2];
  }
  else {
    bTemp = (bTab[iFilterLen / 2] + bTab[iFilterLen / 2 - 1]) / 2;
  }
  return bTemp;
}

float readTdsSensor() {
  static unsigned long analogSampleTimepoint = millis();
```

```
  if (millis() - analogSampleTimepoint > 40U) { //every 40 milliseconds, read
the analog value from the ADC
    analogSampleTimepoint = millis();
    analogBuffer[analogBufferIndex] = analogRead(TdsSensorPin); //read the
analog value and store into the buffer
    analogBufferIndex++;
    if (analogBufferIndex == SCOUNT) {
      analogBufferIndex = 0;
    }
  }

  static unsigned long printTimepoint = millis();
  if (millis() - printTimepoint > 800U) {
    printTimepoint = millis();
    for (copyIndex = 0; copyIndex < SCOUNT; copyIndex++) {
      analogBufferTemp[copyIndex] = analogBuffer[copyIndex];
    }

    // read the analog value more stably by the median filtering algorithm,
and convert to voltage value
    averageVoltage = getMedianNum(analogBufferTemp, SCOUNT) * (float)VREF /
4096.0;

    // temperature compensation formula: fFinalResult(25^C) =
fFinalResult(current) / (1.0 + 0.02 * (fTP - 25.0));
    float compensationCoefficient = 1.0 + 0.02 * (temperature - 25.0);
    // temperature compensation
    float compensationVoltage = averageVoltage / compensationCoefficient;

    // convert voltage value to TDS value
    tdsValue = (133.42 * compensationVoltage * compensationVoltage *
compensationVoltage - 255.86 * compensationVoltage * compensationVoltage +
857.39 * compensationVoltage) * 0.5;


  }
  return tdsValue;
}
```