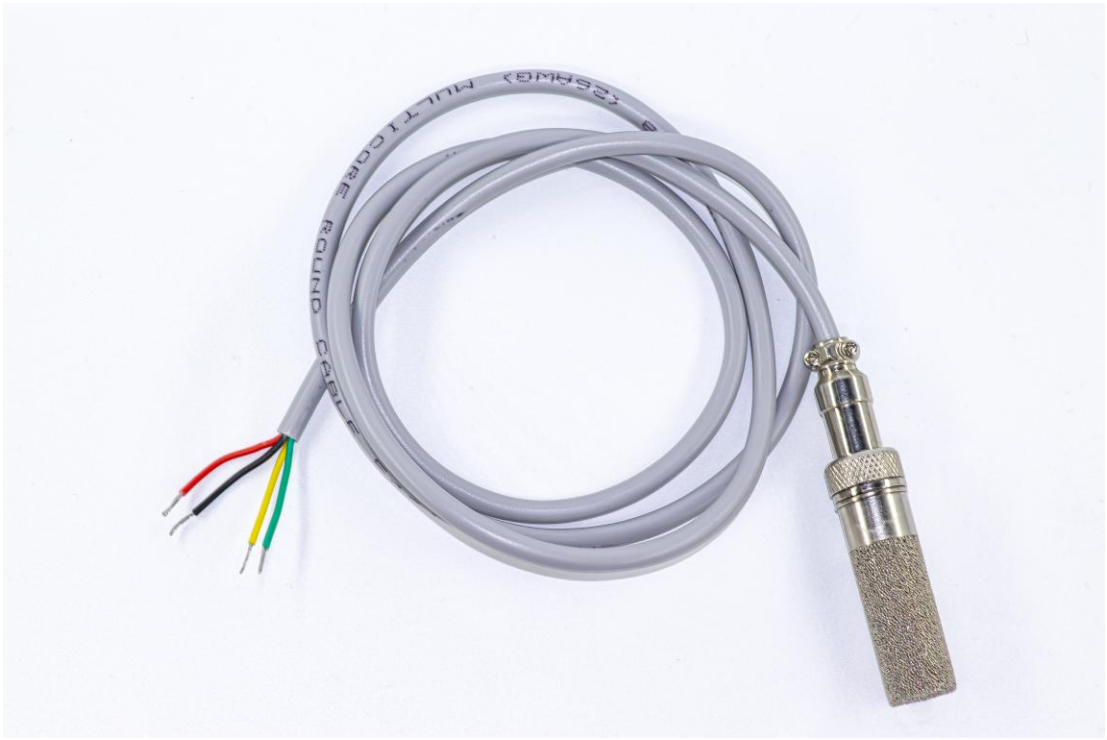
IST

IST

Industrial UART Humidity, Temperature and VOC Sensor Probe

Version 1.0



IST +91 8655821342 info@7semi.com www.7semi.com

Table of Contents

1.Introduction 2
1.1 Features 2
2.Technical specification 3
2.1 General specifications 4
3. Hardware setup 5
3.1 Pinouts 5
4.Sample code and explanation 7
4.1 Sample code 7
4.2 Code explanation 12
4.3 Serial output 14
4.4 Sample code link 15
5.Mechanical specification 16

1.Introduction

This all-in-one sensor is an excellent choice for developers needing accurate and low-power environmental measurements. Its compact design and versatile sensing capabilities make it suitable for everything from consumer wearables to industrial monitoring systems.

1.1 Features

- Digital temperature, humidity and VOC sensor probe with multicore cable.
- Compact and robust design with stainless steel body.
- Suitable for indoor and outdoor applications.
- Seamless compatibility with common development platforms.
- Ideal for HVAC, agriculture, weather stations, and smart home systems.
- Long-term stability for reliable performance over time.

2.Technical specification

This compact board is designed to provide the collection and processing of various environmental parameters, all in an ultra-low-power form factor. Its integrated design, featuring both data acquisition and on-board computation, makes it well-suited for IoT applications, and home automation projects. Flexible interfaces help ensure easy integration into existing systems, while the board's accuracy and reliability support demanding use cases.

The board outputs I2C data containing temperature, humidity, VOC values simplifying integration with external systems. Its energy-efficient design and compact form factor make it ideal for both portable and embedded applications.

Applications:-

- **Home Automation & control**
- **IoT Integration**
- **Weather Forecasting**
- **Educational Tools**
- **Indoor & outdoor monitoring systems**

This sensor board is the ideal solution for applications that demand high accuracy, low power consumption, and easy integration into existing systems.

2.1 General specifications

Operating Range:

- Temperature: -10°C to +50°C
- Humidity: 0 to 90% RH

Accuracy:

- Temperature: ±0.2°C
- Relative humidity: ±1.8% RH

Operating Voltage:

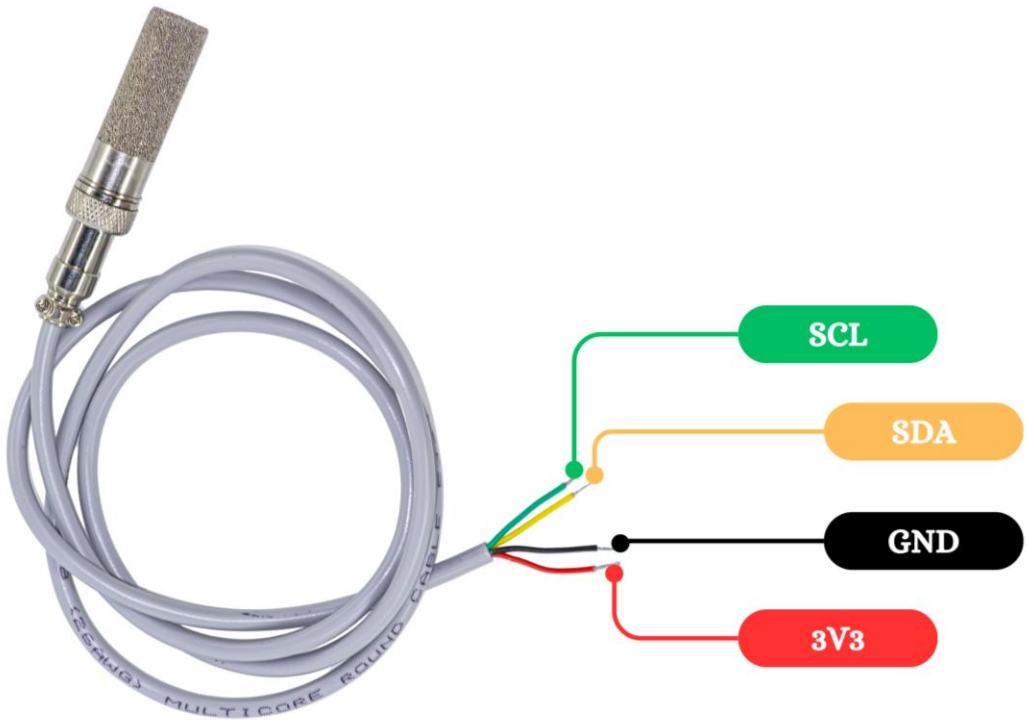
- VDD (supply voltage): 1.71 V to 3.6 V
- VDDIO (interface voltage): 1.2 V to 3.6 V

Communication protocol: I2C

VOC Index, processed value from SRAW_VOC: 1 to 500

3. Hardware setup

3.1 Pinouts



SCL: I2C Communication Serial clock.

SDA: I2C Communication Serial data.

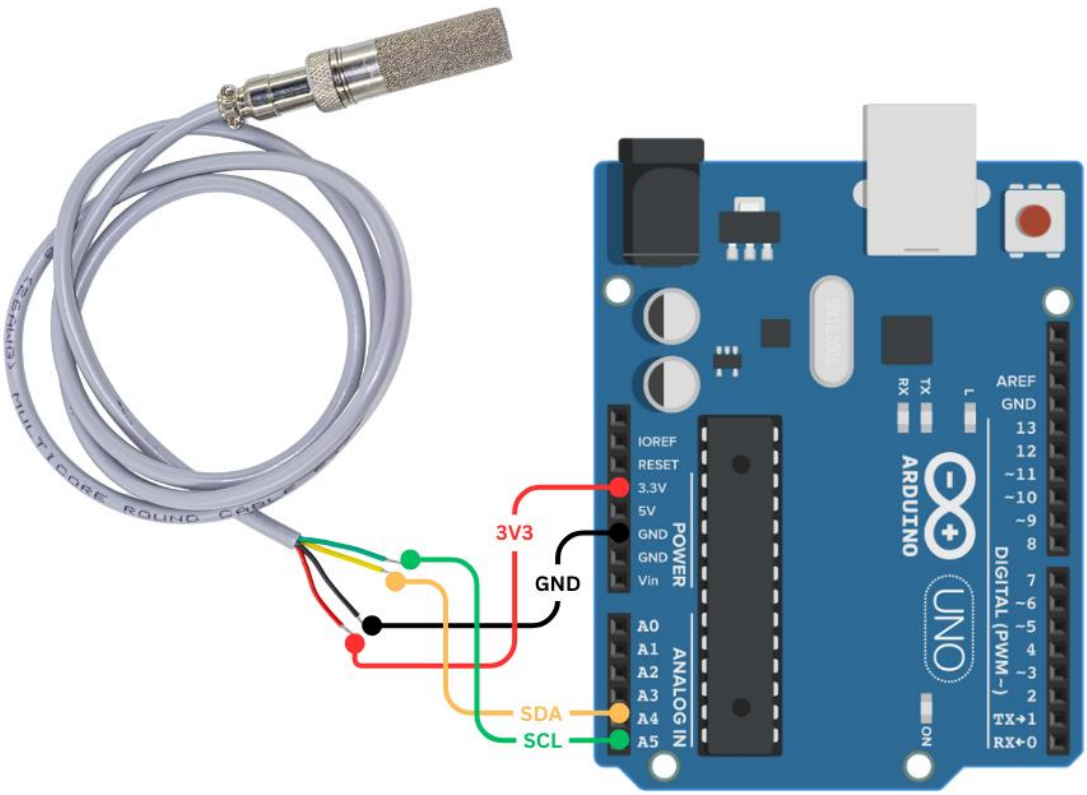
GND: Ground pin, connect to the system's ground.

3V3: Power input, connect to a 3.3V DC supply.

Connection Guidelines

- **Power Supply**: Ensure a clean and stable 3.3V DC power source is connected to the 3V3 pin, with GND tied to the power supply's ground.
- **I2C connection**: Connect SDA of probe to SDA of controller and SCL of probe to SCL of controller.

3.2 Hardware connection



4.Sample code and explanation

4.1 Sample code

```
#include <Wire.h>


#include <Arduino.h>
#include <SensirionI2CSgp40.h>
#include <SensirionI2cSht4x.h>
#include <VOCGasIndexAlgorithm.h>

SensirionI2cSht4x sht4x;
SensirionI2CSgp40 sgp40;

// Sampling interval in seconds
// This code uses a fixed heating pulse of ca. 200 ms for the measurement
and
// thus, the sampling interval defines the duty cycle
float sampling_interval = 1.f;

VOCGasIndexAlgorithm voc_algorithm(sampling_interval);

char errorMessage[256];

void setup() {
    Serial.begin(115200);
    while (!Serial) {
        delay(100);
```

```
    }
    Wire.begin();
    sht4x.begin(Wire, SHT40_I2C_ADDR_44);
    sgp40.begin(Wire);
    delay(1000);  // needed on some Arduino boards in order to have Serial
ready


    Serial.print("Sampling interval (sec):\t");
    Serial.println(voc_algorithm.get_sampling_interval());
    Serial.println("");
}


void sgp40MeasureRawSignalLowPower(uint16_t compensationRh,
                                   uint16_t   compensationT,   uint16_t*
error) {
    uint16_t srawVoc = 0;
    int32_t voc_index = 0;
    // Request a first measurement to heat up the plate (ignoring the
result)
    *error   =   sgp40.measureRawSignal(compensationRh,   compensationT,
srawVoc);
    if (*error) {
        return;
    }


    // Delaying 170 msec to let the plate heat up.
    // Keeping in mind that the measure command already include a 30ms
delay
    delay(140);
```

```
    // Request the measurement values
    *error    =    sgp40.measureRawSignal(compensationRh,    compensationT,
srawVoc);
    if (*error) {
        return;
    }

    Serial.print("\t");
    Serial.print("srawVOC: ");
    Serial.println(srawVoc);

    // Turn heater off
    *error = sgp40.turnHeaterOff();
    if (*error) {
        return;
    }

    // Process raw signals by Gas Index Algorithm to get the VOC index
values
    voc_index = voc_algorithm.process(srawVoc);
    Serial.print("\t");
    Serial.print("VOC Index: ");
    Serial.println(voc_index);
}

void getCompensationValuesFromSHT4x(uint16_t* compensationRh,
                                    uint16_t*  compensationT,  uint16_t*
error) {
    float humidity = 0;     // %RH
```

```
    float temperature = 0;  // degreeC
    *error = sht4x.measureHighPrecision(temperature, humidity);
    if (*error) {
        return;
    }
    Serial.print("T:");
    Serial.print(temperature);
    Serial.print("\t");
    Serial.print("RH:");
    Serial.print(humidity);

    // convert temperature and humidity to ticks as defined by SGP40
    // interface
    // NOTE: in case you read RH and T raw signals check out the
    // ticks specification in the datasheet, as they can be different for
    // different sensors
    *compensationT = static_cast<uint16_t>((temperature + 45) * 65535 /
175);
    *compensationRh = static_cast<uint16_t>(humidity * 65535 / 100);
}

void loop() {

    uint16_t error;
    uint16_t compensationRh =

        0x8000;  // initialized to default value in ticks as defined by
SGP40
    uint16_t compensationT =
```

```
        0x6666;  // initialized to default value in ticks as defined by
SGP40


    // 1. Sleep: We need the delay to match the desired sampling interval
    // In low power mode, the SGP40 takes 200ms to acquire values.
    // SHT4X also includes a delay of 10ms
    delay(int(sampling_interval) * 1000 - 210);


    // 2. Measure temperature and humidity for SGP internal compensation
    getCompensationValuesFromSHT4x(&compensationRh,      &compensationT,
&error);
    if (error) {
        Serial.print(
            "SHT4x - Error trying to execute measureHighPrecision(): ");
        errorToString(error, errorMessage, sizeof(errorMessage));
        Serial.println(errorMessage);
        Serial.println("Fallback to use default values for humidity and "
                       "temperature compensation for SGP40");
    }


    // 3. Measure SGP40 signals using low power mode
    sgp40MeasureRawSignalLowPower(compensationRh, compensationT, &error);
    if (error) {
        Serial.print(
            "SGP40 - Error trying to acquire data in low power mode: ");
        errorToString(error, errorMessage, sizeof(errorMessage));
        Serial.println(errorMessage);
    }
}
```

4.2 Code explanation

```
#include <Wire.h>
#include <Arduino.h>
#include <SensirionI2CSgp40.h>
#include <SensirionI2cSht4x.h>
#include <VOCGasIndexAlgorithm.h>
```

Wire: I2C communication.

SensirionI2CSgp40: For interacting with SGP40 sensor.

SensirionI2cSht4x: For SHT40 sensor.

VOCGasIndexAlgorithm: To convert raw SGP40 output into a meaningful VOC Index.

```
Serial.begin(115200);
Wire.begin();
sht4x.begin(Wire, SHT40_I2C_ADDR_44);
sgp40.begin(Wire);
```

Initializes Serial, I2C, and the sensors.

Prints the sampling interval using the VOC algorithm object.

```
delay(int(sampling_interval) * 1000 - 210);
```

Waits for the correct sampling interval, subtracting sensor delays.

```
getCompensationValuesFromSHT4x(...)
```

Reads temperature and humidity.

Converts these to tick values for SGP40 compensation.

`sgp40MeasureRawSignalLowPower(...)`

Measures srawVOC (raw gas signal).

Turns off heater to save power.

Uses VOCGasIndexAlgorithm to compute VOC Index from raw signal.

4.3 Serial output

```
T:25.98.RH:55.55.srawVOC: 28335
.VOC Index: 73
T:25.97.RH:55.53.srawVOC: 28326
.VOC Index: 74
T:25.97.RH:55.53.srawVOC: 28332
.VOC Index: 75
T:25.99.RH:55.52.srawVOC: 28337
.VOC Index: 76
T:25.99.RH:55.50.srawVOC: 28341
.VOC Index: 77
T:25.98.RH:55.46.srawVOC: 28340
.VOC Index: 78
T:25.98.RH:55.48.srawVOC: 28342
.VOC Index: 79
T:25.97.RH:55.46.srawVOC: 28343
.VOC Index: 80
T:25.98.RH:55.45.srawVOC: 28334
.VOC Index: 81
T:25.98.RH:55.50.srawVOC: 28338
.VOC Index: 81
T:25.97.RH:55.52.srawVOC: 28334
.VOC Index: 82
T:26.02.RH:55.53.srawVOC: 28334
.VOC Index: 83
T:25.97.RH:55.54.srawVOC: 28333
.VOC Index: 84
T:25.99.RH:55.51.srawVOC: 28347
.VOC Index: 84
T:25.99.RH:55.56.srawVOC: 28345
.VOC Index: 85
```

4.4 Sample code link

Sample code link:- [ES-50103-I4040]

5.Mechanical specification