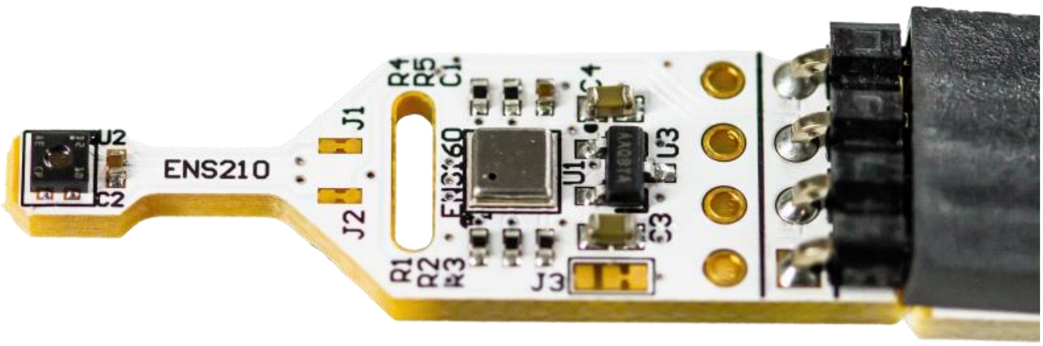

Fundamental communication

A basic guide to communicate via SPI or I²C with ENS16x sensors without specific libraries

ENS16x fundamental communication Application Note

Version: 0.7

Release Date: 2025-07-31

ScioSense

Table of Contents

1 Introduction

The ScioSense ENS16x sensors (amongst others) allow communication either via I²C or SPI interface. This application note shall give a short introduction and quick start to establish communication on one or the other protocol.

ScioSense is offering free to use firmware libraries, which can be implemented and adjusted to your own need. The latest versions can be found on our Github page including examples and should be used preferably.

https://github.com/sciosense/ens16x-arduino

This AN does support and explains fundamentals of SPI and I²C communication and can easily adapted and expanded for any application.

All software examples are based on Arduino code, compatible for a wide range of microcontrollers. Obviously proper board libraries are required as well as correct setting of wiring for MOSI/MISO/CLK/SS or SDA/SCL.

2 Hardware

2.1 Schematic

The sensor board primarily comprises the ENS16x gas sensor (U1), the ENS210 RH/T sensor (U2), a linear voltage regulator (U3), and jumper bridges for configuration (refer to 2.3.2).

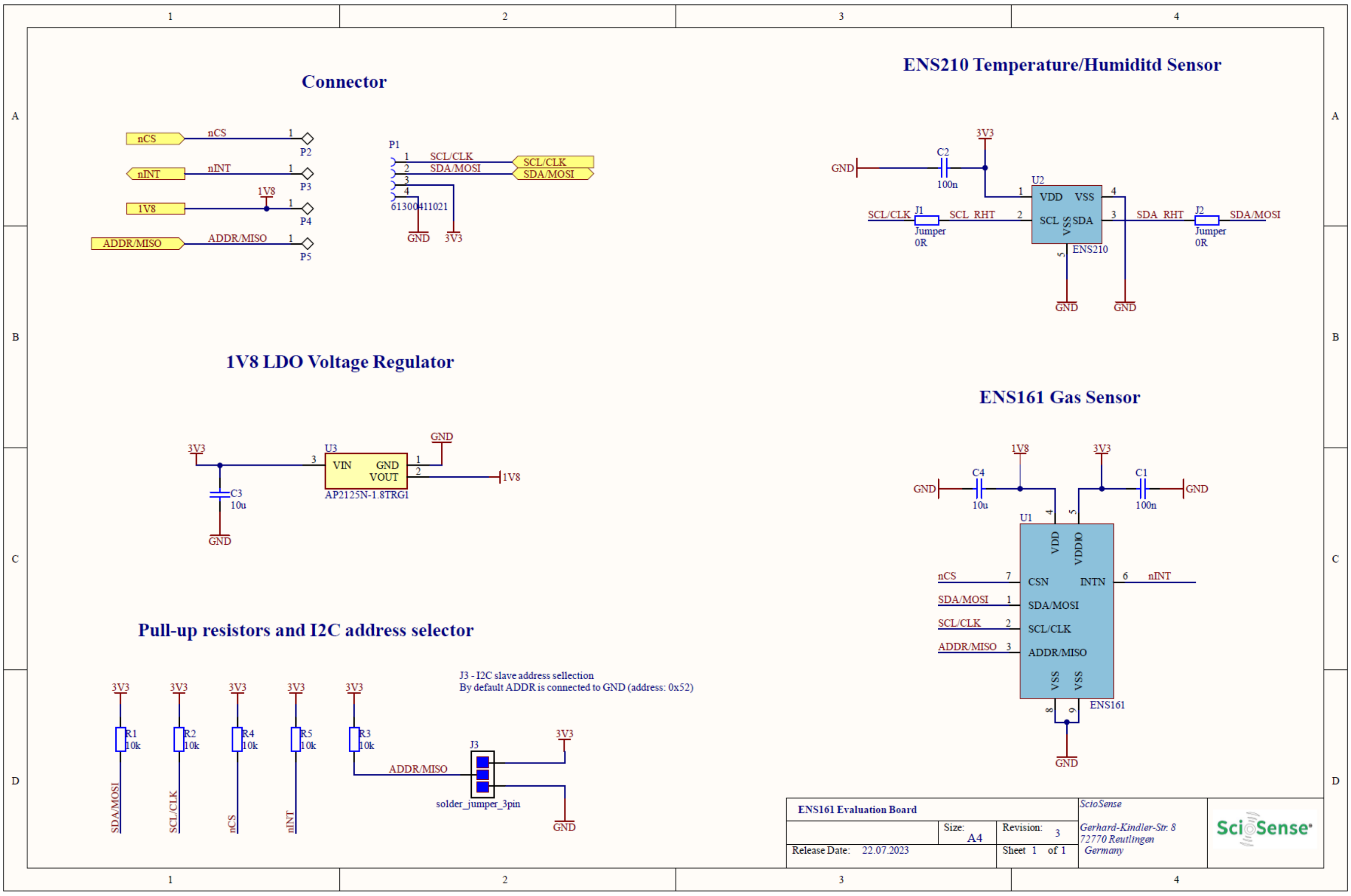

Figure 1: Schematic sensor board

2.2 Sensor board

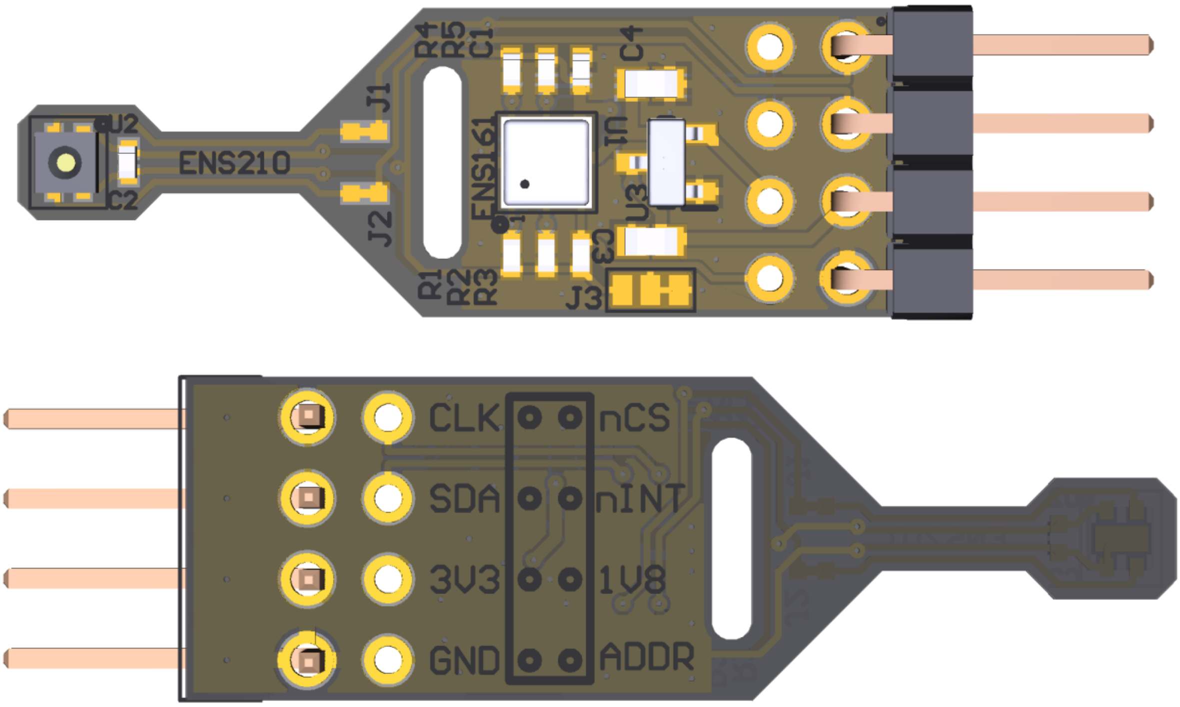

Figure 2: sensor board components and connections

2.3 Preparation

2.3.1 Disabling ENS210 RHT Sensor

J1 and J2 are normally closed, so I²C bus is connected to ENS210. However, if not needed or SPI has been selected, it is recommended to open (cut) the copper wire on both jumpers.

2.3.2 Selection I²C / SPI and I²C address

Jumper J3 provides the option to connect ENS16x pin 3 (ADDR/MISO) either to GND, 3.3V or keep open. This setting has direct impact on bus interface configuration.

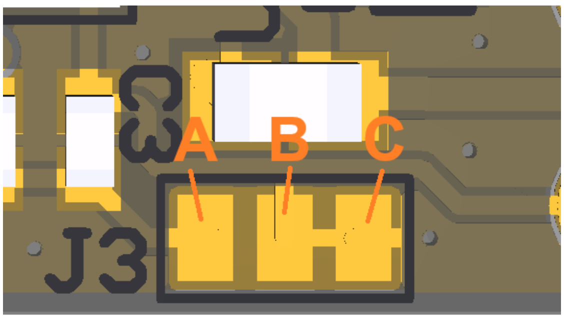

Figure 3: Jumper J3 function

By default, GND is connected to ADDR/MISO pin so the ENS16x operates in I²C mode with address 0x52. Available options are

Table 1: configuration address and bus type

Pad: A (3.3V) B (ADDR/MISO) C (GND)		function
		I²C active, address 0x53
	default	I²C active, address 0x52
		SPI activated

Under no circumstances should pad A and pad C be connected, as this will short the supply voltage to GND and may cause damage to the board

Before working on the wiring and code, please make sure to make yourself familiar with the datasheet. Especially chapter 14.2 describes SPI specific considerations, while the following chapter 15 and 16 focusing on mode of operation and register handling

3 SPI Operation

3.1 Wiring

Wiring is already described as below in the datasheet for direct sensor usage; make sure not to exceed voltage limits (max V_{DDIO}=3.6V, max V_{DD} = 1.98V)

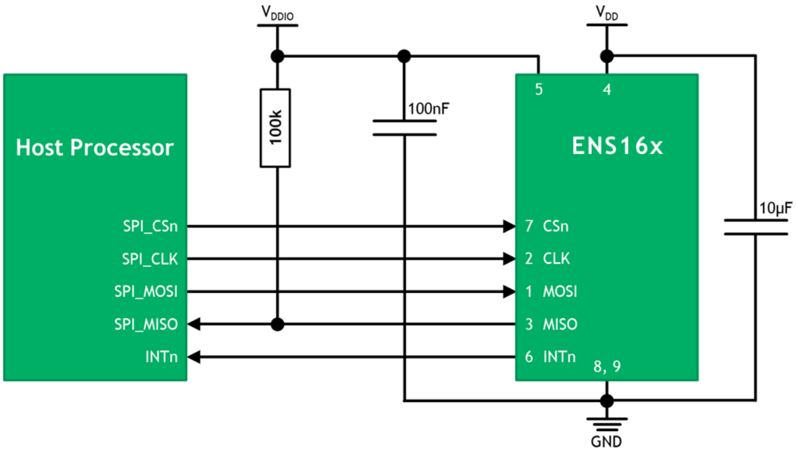

Figure 4: Recommended application circuit for direct sensor usage (SPI operation)

If you're using the ENS16x EVKit, please ensure that you've carefully followed the steps outlined in Section 2.3 to configure SPI communication correctly. Additionally, make sure the ENS210 sensor is disabled as specified in the documentation to avoid any conflicts during operation.

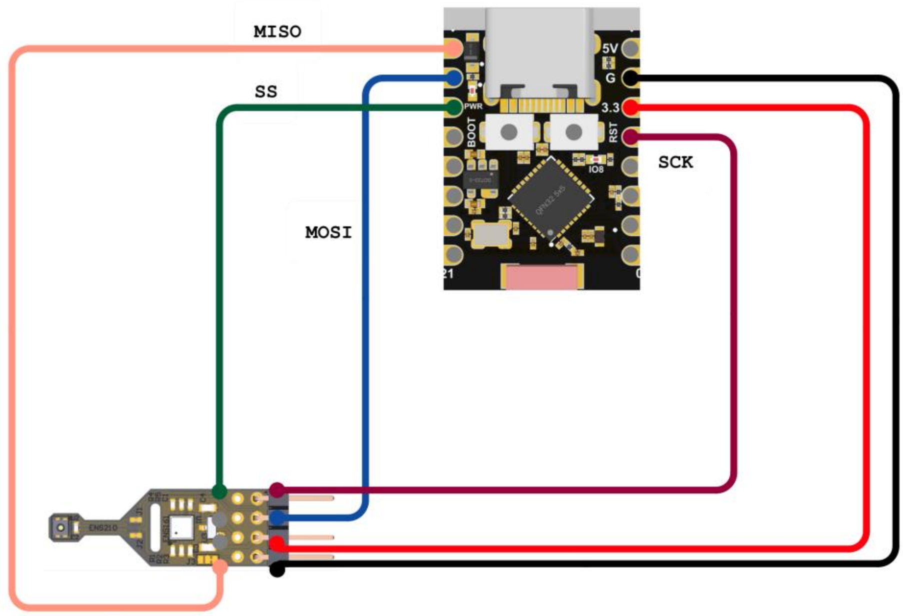

Figure 5: SPI Wiring example on ESP32-C3 SuperMini

3.2 Code example

The following code is an Arduino based example to demonstrate fundamental communication

```
/*
*       Basic function ENS16x SPI
*       this example code provides a basic communication via SPI between µC and ENS16x
*
*       code provided as it is. Note the sensor is working on 3.3V communication level
*       - level shifter might be required
*       V1: 2025-07-07 TR
*/

#include <SPI.h>

// register definition
#define PARTID_REGISTER 0x00
#define OPMODE_REGISTER 0x10
#define DEVICE_STATUS_REGISTER 0x20
#define AQI_U_REGISTER 0x21
#define TVOC_REGISTER 0x22
#define ECO2_REGISTER 0x24
#define AQI_S_REGISTER 0x26     //available on ENS161 only
#define GPR_READ_REGISTER 0x48

// structure definition
#define OPMODE_IDLE 0x01
#define OPMODE_STANDARD 0x02
#define OPMODE_LOWPOWER 0x03        //available on ENS161 only
#define OPMODE_ULTRALOWPOWER 0x04   //available on ENS161 only
#define OPMODE_RESET 0xF0

// pin on Hardware
// make sure MISO, MOSI, CLK are properly connected
const int CS_PIN = 5;  // Chip Select

void setup() {
  Serial.begin(9600);
  SPI.begin();
  pinMode(CS_PIN, OUTPUT);
  digitalWrite(CS_PIN, HIGH);

  Serial.println("----------- Start ENS16x SPI-Communication... -----------");
  ens16x_init();
}

void loop() {
  // get values from register
  uint16_t eco2 = readRegister(ECO2_REGISTER);
  uint16_t tvoc = readRegister(TVOC_REGISTER);
  uint16_t aqi_s = readRegister(AQI_S_REGISTER);        //for ENS161 only


  // display results
  Serial.print("eCO₂: ");
  Serial.print(eco2);
```

```
  Serial.print("ppm\tTVOC: ");
  Serial.print(tvoc);
  Serial.print(" ppb\tAQI_S: ");        //for ENS161 only
  Serial.println(aqi_s);

  delay(1000);
}

void ens16x_init() {
  // initialize sensor
  // execute reset
  writeRegister(OPMODE_REGISTER, OPMODE_RESET);
  delay(100);

  // read sensor type
  Serial.print("Device Type: ENS");
  Serial.println(readRegister(PARTID_REGISTER), HEX);

  // switch to standard mode
  writeRegister(OPMODE_REGISTER, OPMODE_STANDARD);
  delay(100);
}

uint16_t readRegister(uint8_t reg) {
  // read two byte register
  digitalWrite(CS_PIN, LOW);
  SPI.transfer((reg << 1) + 1);   //read request
  uint8_t low = SPI.transfer(0x00);
  uint8_t high = SPI.transfer(0x00);
  digitalWrite(CS_PIN, HIGH);

  return (high << 8) | low;
}

void writeRegister(uint8_t reg, uint8_t payload) {
  // write one byte into register
  digitalWrite(CS_PIN, LOW);
  SPI.transfer(reg << 1 );  //write request
  SPI.transfer(payload);

  digitalWrite(CS_PIN, HIGH);
}
```

3.3 Signal plot

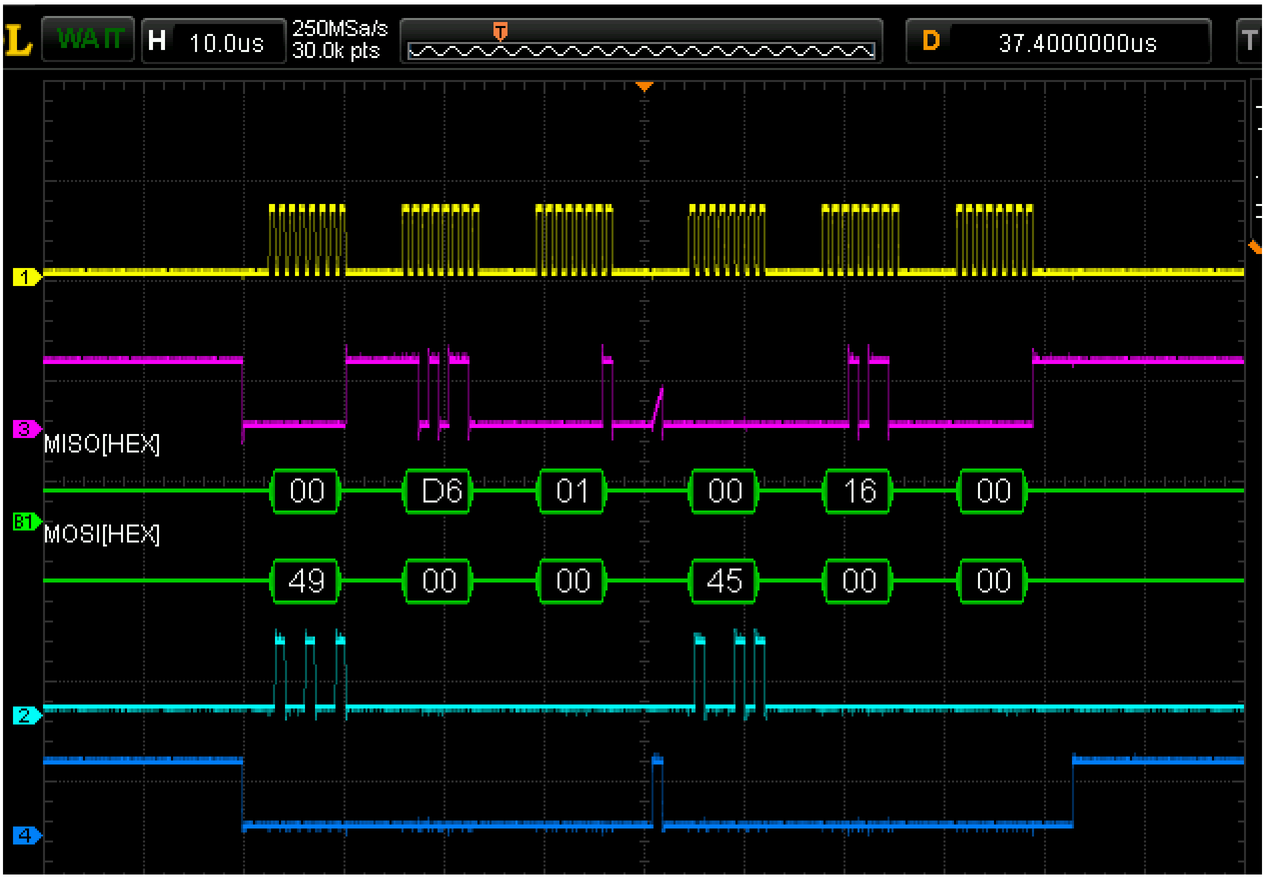

Figure 6: signal plot for eCO2 and TVOC reading (example)

Description:

1. Register 0x24 (eCO2) is addressed (0x24 << 1 + 1 = 0x49)
2. Two bytes are sent from the slave, low byte 0xD6 followed by high byte 0x01
3. Chip select (SS) disabled and then enabled the sensor
4. Register 0x22 (TVOC) is addressed (0x22 << 1 + 1 = 0x45)
5. Two bytes are sent from the slave, low byte 0x16 followed by high byte 0x00

4 I²C Operation

4.1 Wiring

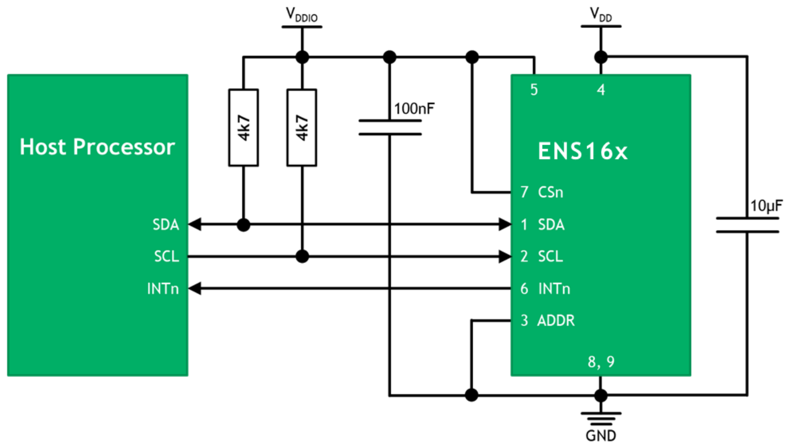

Figure 7: Recommended application circuit for direct sensor usage (I²C operation)

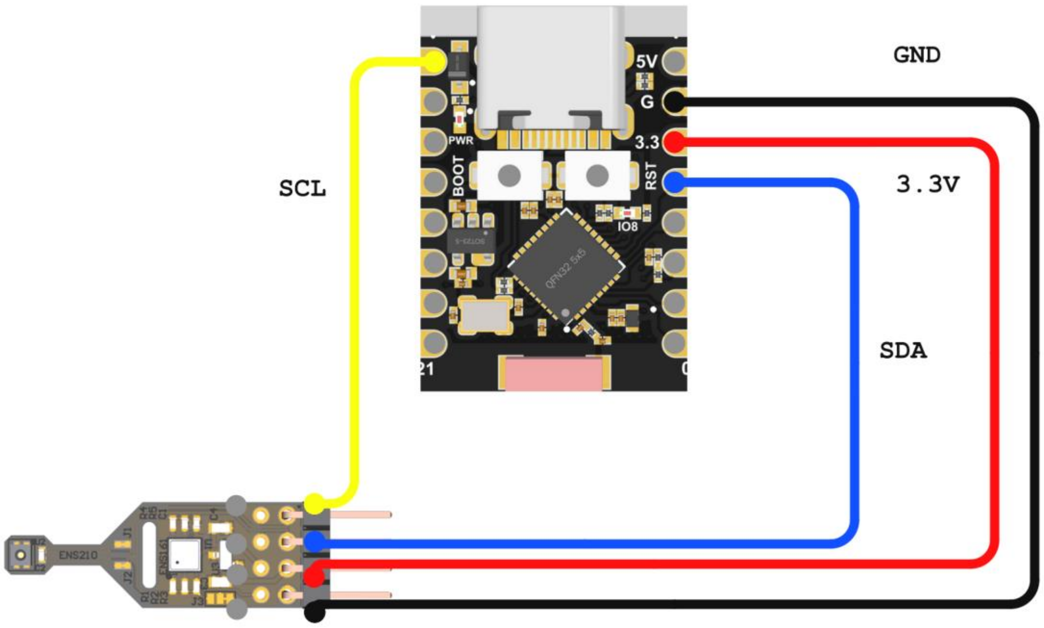

Figure 8: I²C Wiring example on ESP32-C3 SuperMini

4.2 Code example

The following code is an Arduino based example to demonstrate fundamental communication. For for full functionality please refer to official libraries provided on GitHub

```
/*      Basic function ENS16x I2C
*       this example code provides a basic communication via I²C between µC and ENS16x
*
*       code provided as it is. Note the sensor is working on 3.3V communication level
*       - level shifter might be required
*       V1: 2025-07-08 TR
*/

#include <Wire.h>

// I²C-address of ENS16x
#define ENS16x_I2C_ADDRESS 0x52

#define PARTID_REGISTER 0x00
#define OPMODE_REGISTER 0x10
#define DEVICE_STATUS_REGISTER 0x20
#define AQI_U_REGISTER 0x21
#define TVOC_REGISTER 0x22
#define ECO2_REGISTER 0x24
#define AQI_S_REGISTER 0x26         //available on ENS161 only
#define GPR_READ_REGISTER 0x48

#define OPMODE_IDLE 0x01
#define OPMODE_STANDARD 0x02
#define OPMODE_LOWPOWER 0x03        //available on ENS161 only
#define OPMODE_ULTRALOWPOWER 0x04   //available on ENS161 only
#define OPMODE_RESET 0xF0

byte receivedData[5];

void setup() {
    Wire.begin();  // make sure to use correct pins for SDA and SCL
    Serial.begin(9600);
    Serial.println("----------- Start ENS16x I2C-Communication... -----------");
    ens16x_init();
}
void loop() {
    // get TVOC bytes from register
    readRegister(ENS161_I2C_ADDRESS, TVOC_REGISTER, receivedData, 2);
    Serial.print("TVOC: \t");
    Serial.println((uint16_t)(receivedData[1] << 8) + receivedData[0]);

    // get eCO2 bytes from register
    readRegister(ENS161_I2C_ADDRESS, ECO2_REGISTER, receivedData, 2);
    Serial.print("eCO2: \t");
    Serial.println((uint16_t)(receivedData[1] << 8) + receivedData[0]);

    delay(1000);
}
```

```
void ens16x_init() {

    // initialize sensor
    // execute reset
    writeRegister(ENS161_I2C_ADDRESS, OPMODE_REGISTER, OPMODE_RESET);

    // read sensor type
    readRegister(ENS161_I2C_ADDRESS, PARTID_REGISTER, receivedData, 2);
    Serial.print("Device Type: ENS");
    Serial.print(receivedData[1], HEX);
    Serial.println(receivedData[0], HEX);

    // switch to standard mode
    writeRegister(ENS161_I2C_ADDRESS, OPMODE_REGISTER, OPMODE_STANDARD);
}


void writeRegister(byte deviceAddress, byte registerAddress, byte data) {
    // routines to write I²C sequence
    Wire.beginTransmission(deviceAddress);
    Wire.write(registerAddress);
    Wire.write(data);
    Wire.endTransmission();
}

int readRegister(byte deviceAddress, byte registerAddress, byte* buffer, int bufferSize) {
    // routines to read I²C sequence
    Wire.beginTransmission(deviceAddress);
    Wire.write(registerAddress);
    Wire.endTransmission(false);
    Wire.requestFrom(deviceAddress, bufferSize);
    int bytesRead = 0;
    while (Wire.available() && bytesRead < bufferSize) {
        buffer[bytesRead] = Wire.read();
        bytesRead++;
    }
    return bytesRead;  // returns number of received bytes
}
```

4.3 Signal plot

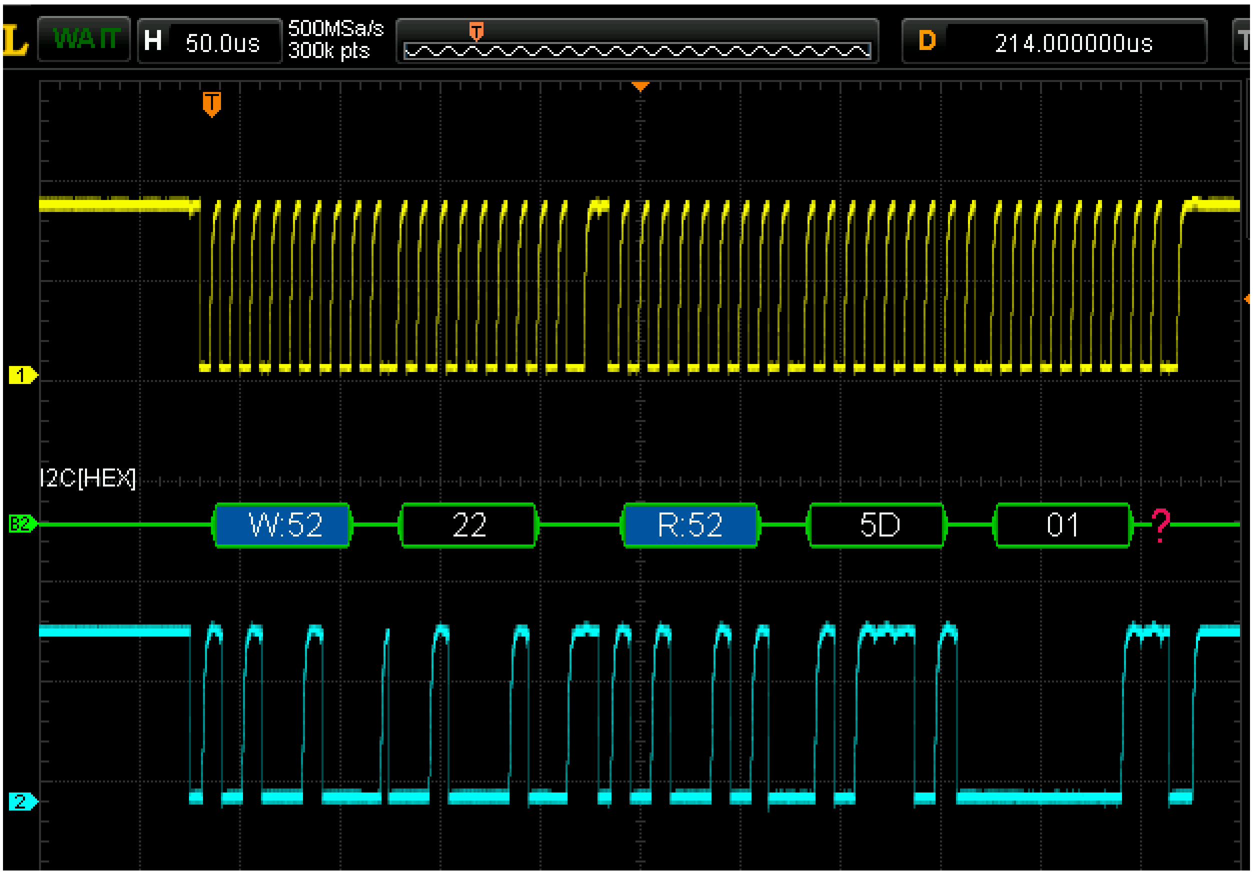

Figure 9: signal plot for TVOC reading

Description

1. Sending write instruction to device address 0x52
2. Address register 0x22 (TVOC)
3. Sending read request to device address 0x52
4. Reading two bytes: low byte 0x5D, high byte 0x01

5 Further information

5.1 Links

For more information on the sensor please find on our website http://www.sciosense.com

ENS160 Datasheet ↓	ENS161 Datasheet ↓

Application notes

ENS16x Evaluation Kit ↓

Code libraries and examples

https://github.com/sciosense/ens16x-arduino

6 Copyrights & Disclaimer

Devices sold by Sciosense B.V. are covered by the warranty and patent indemnification provisions appearing in its General Terms of Trade. Sciosense B.V. makes no warranty, express, statutory, implied, or by description regarding the information set forth herein. Sciosense B.V. reserves the right to change specifications and prices at any time and without notice. Therefore, prior to designing this product into a system, it is necessary to check with Sciosense B.V. for current information. This product is intended for use in commercial applications. Applications requiring extended temperature range, unusual environmental requirements, or high reliability applications, such as military, medical life-support or life-sustaining equipment are specifically not recommended without additional processing by Sciosense B.V. for each application. This product is provided by Sciosense B.V. "AS IS" and any express or implied warranties, including, but not limited to the implied warranties of merchantability and fitness for a particular purpose are disclaimed.

This product is not intended for use as safety device and/or in applications where failure of the product could result in personal injury. Sciosense B.V. shall not be liable to recipient or any third party for any damages, including but not limited to personal injury, property damage, loss of profits, loss of use, interruption of business or indirect, special, incidental or consequential damages, of any kind, in connection with or arising out of the furnishing, performance or use of the technical data herein. No obligation or liability to recipient or any third party shall arise or flow out of Sciosense B.V. rendering of technical or other services.

7 Revision information

Table 2: Revision history

Revision	Date	Comment	Page
0.7	2025-07-18	Initial Preliminary Version	All

Note(s) and/or Footnote(s):

1. Page and figure numbers for the previous version may differ from page and figure numbers in the current revision.
2. Correction of typographical errors is not explicitly mentioned.

Address: Sciosense B.V.
High Tech Campus 10
5656 AE Eindhoven
The Netherlands

Contact: www.sciosense.com
info@sciosense.com