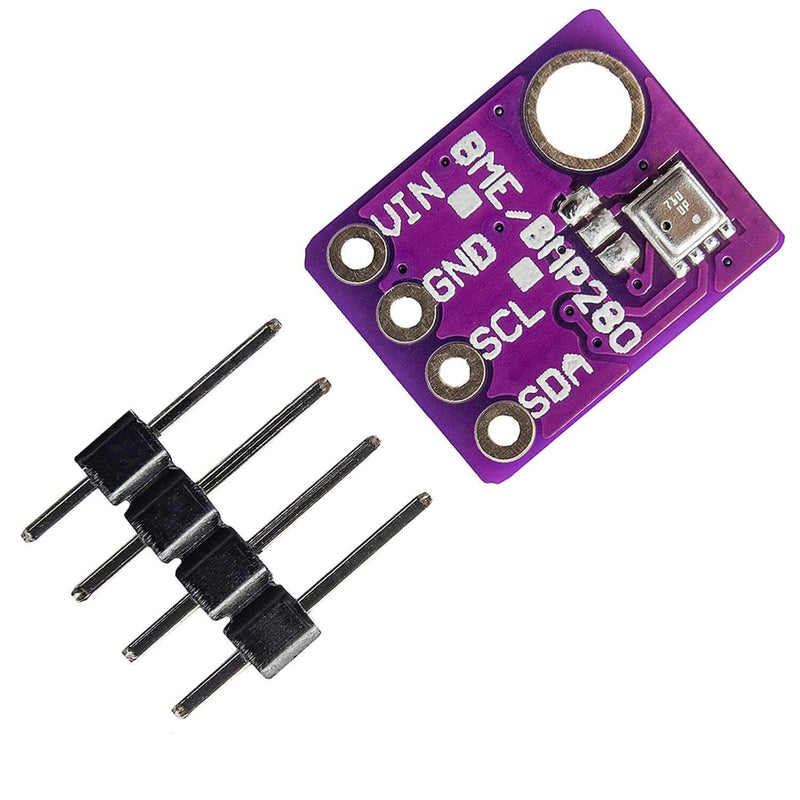
Technical Report

for

Assignment 4



Fontys University of Applied Sciences

ICT & Technology

Embedded Systems

Group 5

Serhiy Medvedyev & Peter Szilagyi

23.05.2023

# **Abstract**

This assignment focuses on the utilization of I2C communication to control both a BME280 sensor and a secondary Arduino board by accessing and modifying their respective registers based on specific requirements. A significant aspect of the assignment involved developing a dedicated library for the BME280 sensor. This process entailed thorough research of datasheets, filtering and extracting the necessary data, and performing crucial calculations to obtain easily understandable values. The final phase of the assignment involved merging the first and third parts, ensuring the code and communication functioned together and errorless. This integrated approach served to validate the code's robustness and effectiveness in facilitating efficient communication between devices.

Contents

[**1.** **Abstract** 2](#_Toc135775499)

[**2.** **Introduction** 3](#_Toc135775500)

[**3.** **Procedure** 3](#_Toc135775501)

[a. Setup 3](#_Toc135775502)

[b. Assignment A 3](#_Toc135775503)

[c. Assignment B 5](#_Toc135775504)

[d. Assignment C 6](#_Toc135775505)

[e. Assignment D 7](#_Toc135775506)

[**4. Conclusion** 8](#_Toc135775507)

[**5.References** 10](#_Toc135775508)

# **Introduction**

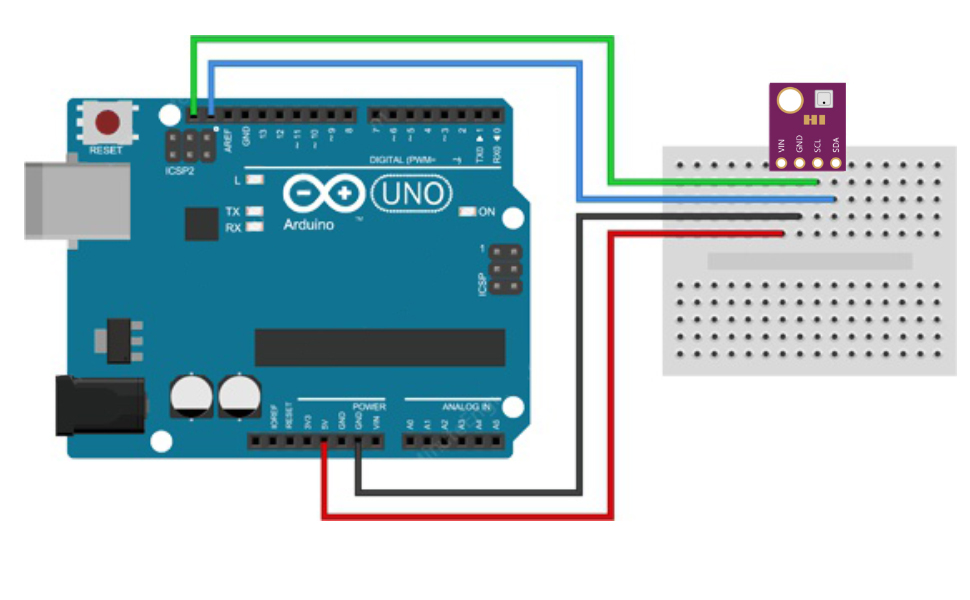
The practicum consists of four parts which are logical continuations of each other. Assignment A featured writing a driver for BME280 sensor to get temperature, humidity and pressure as well as changing oversampling settings. Assignment B included researching OnReceive() and OnRequest() asynchronous events in I2C communication and their application in master <-> slave Arduino communications. Third assignment required creating an own model of master <-> slave communication. It was needed to simulate the register system where we have 2 Read-Write registers to store values which come from master and 2 Read-only registers which are calculated depending on first two registers values and are then sent on request from master Arduino. The last assignment required combining BME280 sensor with master <-> slave Arduino communication to work simultaneously using I2C communication.

# **Procedure**

## Setup

• Laptop + VS Code + PlatformIO + Arduino IDE  
• 2 x Arduino + USB cable  
• 1 x breadboard  
• 1 x BME280 sensor  
• Jumper wires

## Assignment A



*Figure 1*

This assignment required creating a driver for BME280 sensor. Communication with the master device (Arduino) and BME280 sensor is handled by I2C bus which makes use of *Wire.h* library. The main amount of work was dedicated to reading datasheets and adapting algorithms and concepts described there for the team’s needs. The list of functions needed to be implemented:

uint8\_t BME280\_GetID();  
void BME280\_Reset();  
uint8\_t BME280\_CtrlHum();  
void BME280\_CtrlHum(uint8\_t bitpattern);  
uint8\_t BME280\_CtrlMeas ();  
void BME280\_CtrlMeas(uint8\_t bitpattern);  
long BME280\_ReadTemperature ();  
int BME280\_ReadHumidity();  
long BME280\_ReadPressure ();

The first two functions are general control functions which access relative registers(dig\_T#, dig\_H#, dig\_P#) and either get or set values to perform needed functionality: getting the device’s ID and resetting the sensor. Next four functions control data oversampling for humidity, temperature and pressure. It means the frequency with which data is sent to Arduino via I2C. It can be set to 0 (the data sending is skipped, so no data is acquired), or to set values starting from one to specify the needed frequency. This is done by accessing related registers (0xF2, 0xF4) which can be seen on Figure 2. The last three functions provide human-readable data from sensor on humidity, temperature and humidity and require the most manipulation to execute. At the beginning, the raw data extracted from these registers(0xF7-0xFE) at once by recommendation from datasheets.(Bosch© BME280 - Combined humidity and pressure sensor (BST-BME280-DS001-18, November 2020), p.23). This data is not human readable as it is just a set of meaningless numbers. To convert them to actual values it was needed to follow the protocol described in the datasheets. For each type of data (temperature, humidity, pressure) the manufacturer provided so-called compensators – constant values which can be found on specific registers(0x88 -0xA1, 0xE1-0xE7) inside the sensor to perform needed calculations. The last step to get actual measurements was to calculate them using formulas found in the datasheets using raw data, bit manipulation and compensators(Bosch© BME280 - Combined humidity and pressure sensor (BST-BME280-DS001-18, November 2020), p.24-25).

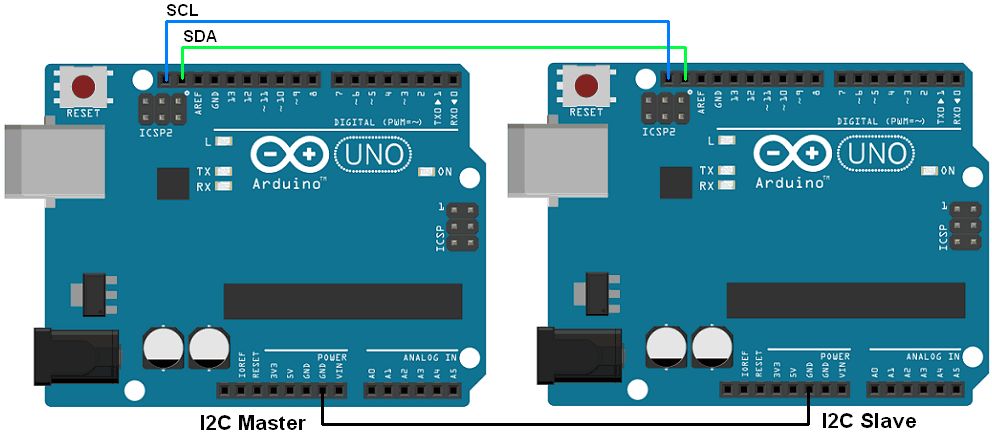
A picture containing text, screenshot, line, number

Description automatically generated

*Figure 2*

## Assignment B

Assignment B featured using an Arduino as a slave in I2C bus communication. It was required to research different communication methods and events using provided materials, especially OnReceive() and OnRequest() events. After that, a simple application was built to handle communication between two Arduinos. The whole application loop is represented by the idea that slave device is a listener and runs only when there is some request from master device. Master device would send an incrementing byte. Whenever slave receives this byte, it should calculate the response by the condition set. If the input byte is less than one hundred it sends 4, otherwise – 2. However, slave does not perform the transmission right after receive event. On the contrary, it waits (listens) for the request from master and only then transmits the response.



*Figure 3*

## Assignment C

Assignment C is a continuation of Assignment B. Two Arduinos were connected via I2C bus as pictured in Figure 3. The slave Arduino connected to the bus with 0x50 address. Its main responsibility was to handle made up registers which are either Read – Write or Read-Only. These registers are changed depending either on values received on OnReceive() method which changes the values on Read-Write registers or sends data on request from master. The flow for receive function is represented on Figure 4.

A screen shot of a computer program

Description automatically generated with medium confidence

*Figure 4*

Whenever master sends data to slave, master specifies the register in the first byte sent and the actual data in the next byte. For this reason, slave reads the register to a global variable and the value depending on the register received.

A screen shot of a computer program

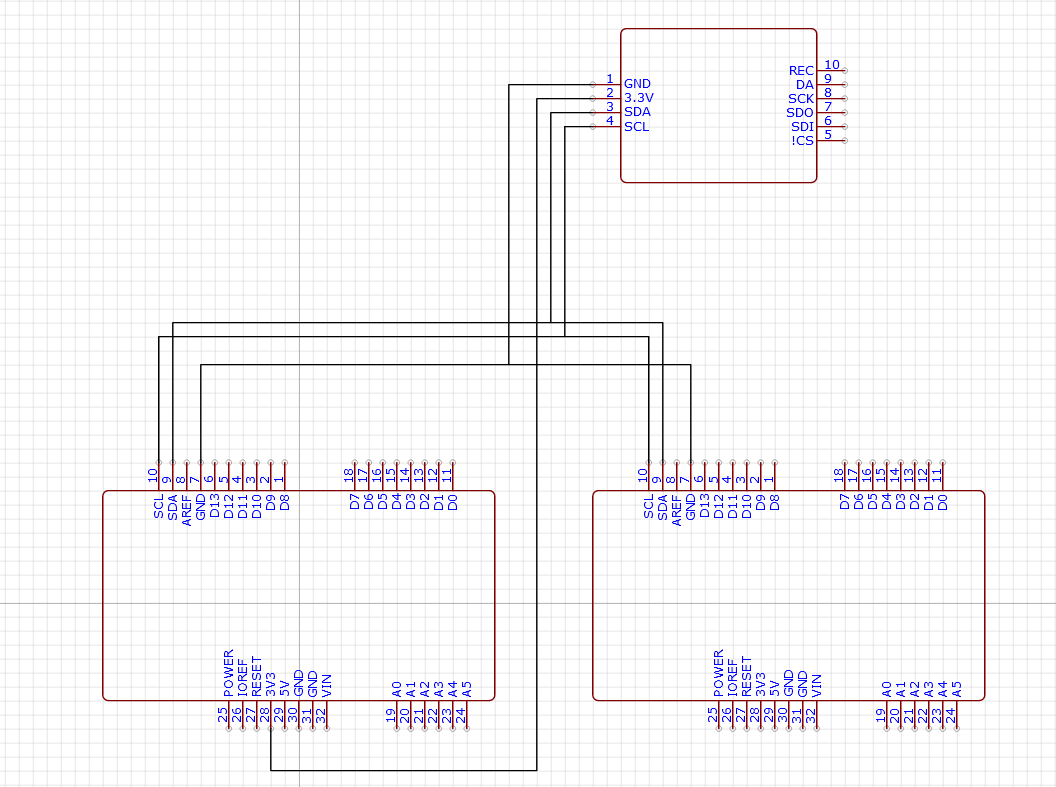
Description automatically generated with low confidence

*Figure 5*

Figure 5 pictures the request function on slave which sends data to master on request. Before the request master writes to the needed register to specify the one it will request from. That’s why the register used in this function is already preset in the global variable in the receive function. Lastly, slave writes to master the value from the specified register.

## Assignment D

The last assignment is a combination of Assignment A and C implying that both BME280 and slave Arduino are connected to the same I2C bus and communicate with master Arduino. This assignment aimed at proofing the concept of using the premade driver created for BME 280 and the library for I2C communication between two Arduinos. After combining all the libraries in one project it was possible to use functions from both. The hardware was connected as pictured in Figure 4.



*Figure 6*

As soon as the addresses of the devices are predefined and different it is possible to use functionality of two libraries at once: get the atmospheric data from BME280 and communicate with slave by accessing premade registers.

# **4. Conclusion**

The primary objective of this assignment was to further explore the utilization of device registers at an advanced level and establish I2C connections between multiple devices. The initial assignment involved working with datasheets, extracting essential information, and integrating it into the software development process to ensure proper device functionality. Additionally, the I2C bus was employed to establish communication with the sensor. The analysis of datasheets facilitated the extraction and organization of relevant sensor data, enabling the calculation of actual values displayed to the user. The Wire.h library was utilized to establish communication with connected devices via their respective addresses. Subsequent assignments focused on enhancing knowledge of the Wire.h library and I2C communication techniques, culminating in the establishment of a connection between two Arduino boards. These assignments involved implementing event-driven techniques for the I2C bus, where slave devices only respond upon receiving information or requests from the master. The final assignment played a conclusive role by integrating all the acquired concepts and knowledge. The BME280 sensor (representing Assignment 3) and the slave Arduino (representing Assignments B & C) were successfully connected to a single I2C bus, demonstrating their intended behavior and communication with the master device.

# **5.References**

(1) BME280 – Data sheet - <https://www.bosch-sensortec.com/products/environmental-sensors/humidity-sensors-bme280/#documents>

(2) Arduino circuit images - <https://easyeda.com/editor#id=!ececebe837ab446d8a691057f2b68227|!d6fbfb769d6246d1a38267a38fe99568>

(3)Bosch© BME280 - Combined humidity and pressure sensor (BST-BME280-DS001-18, November 2020)