ES ASSIGNMENT 3

I2C bus

Rens Pastoor
Thijs van de Weijer
Gijs van Maanen
Max van Oers

Inhoudsopgave

Introduction

This report documents the development and integration of I2C (Inter-Integrated Circuit) communication in a series of embedded systems assignments as outlined in the practicum document *"ES2_WK10+11+12 - Practicum 4"*.

Each assignment focuses on practical I2C applications using Arduino Uno boards:

- **Assignment A:** Writing a device driver for the BME280 environmental sensor.
- **Assignment B:** Establishing two-way communication between two Arduinos over I2C.
- **Assignment C:** Implementing a virtual register-based slave device.
- **Assignment D:** Demonstrating all components together on a shared I2C bus.

Equipment Used

(As specified in the practicum instructions)

Hardware:

- 2 × Arduino Uno boards
- 1 × Bosch BME280 sensor (for temperature, humidity, and pressure)
- Breadboard and jumper wires

Software:

- Arduino IDE for firmware upload and serial debugging
- PlatformIO (Visual Studio Code) for structured project management
- Teleplot / Arduino Serial Plotter for visualizing data streams

Assignment A: BME280 Device Driver

Objective:

Implement a low-level C device driver for the BME280 sensor using I2C communication. No external libraries were allowed; only register-level access based on the datasheet.

Key Implementation:

- The sensor uses address 0x76 and connects via SDA (A4) and SCL (A5) on the Arduino.
- Accessed registers like:
  - 0xD0: Chip ID (via BME280_GetID)
  - 0xE0: Soft reset (BME280_Reset)
  - 0xF2: Humidity control (CtrlHum)
  - 0xF4: Measurement control (CtrlMeas)
  - 0xF7–0xFE: Raw ADC data for pressure, temperature, humidity
- Raw values converted to physical units using calibration coefficients.

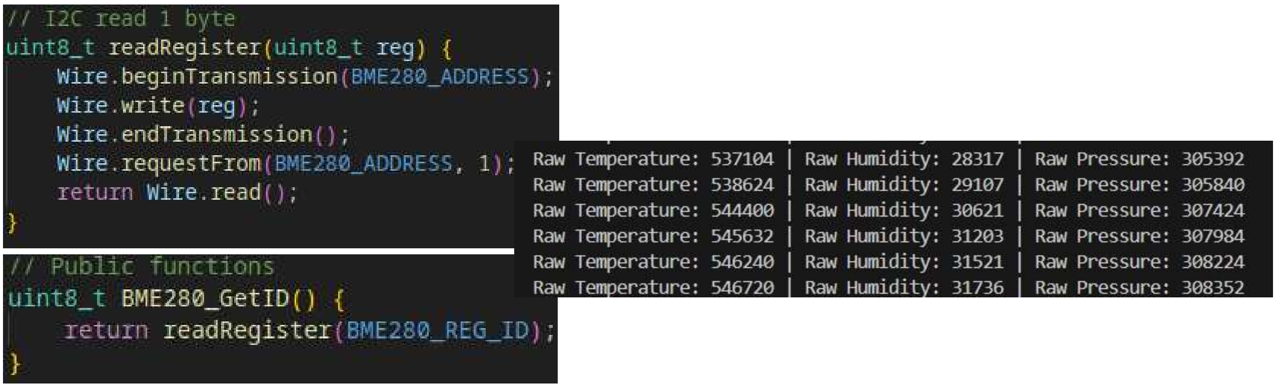
Improvements:

- Used enum types for control bit patterns to prevent errors and improve readability.

Testing:

Data was displayed using the Serial Plotter in real-time, confirming functional sensor reading.

Code Snippet:



```
// I2C read 1 byte
uint8_t readRegister(uint8_t reg) {
    Wire.beginTransmission(BME280_ADDRESS);
    Wire.write(reg);
    Wire.endTransmission();
    Wire.requestFrom(BME280_ADDRESS, 1);
    return Wire.read();
}
```

```
// Public functions
uint8_t BME280_GetID() {
    return readRegister(BME280_REG_ID);
}
```

```
Raw Temperature: 537104 | Raw Humidity: 28317 | Raw Pressure: 305392
Raw Temperature: 538624 | Raw Humidity: 29107 | Raw Pressure: 305840
Raw Temperature: 544400 | Raw Humidity: 30621 | Raw Pressure: 307424
Raw Temperature: 545632 | Raw Humidity: 31203 | Raw Pressure: 307984
Raw Temperature: 546240 | Raw Humidity: 31521 | Raw Pressure: 308224
Raw Temperature: 546720 | Raw Humidity: 31736 | Raw Pressure: 308352
```

Assignment B: Arduino Master-Slave I2C

Objective:

Enable full-duplex data exchange between two Arduino Unos using the I2C protocol.

Setup:

- Master Arduino sends data using Wire.write() and receives responses using Wire.requestFrom().
- Slave Arduino listens at address 0x42 using:
  - Wire.onReceive() for incoming data
  - Wire.onRequest() to send responses

Logic:

- Master sends an incrementing byte.
- Slave responds with:
  - 2 if byte > 100
  - 4 otherwise

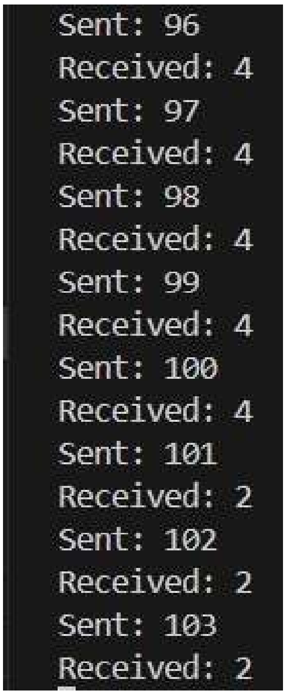
Testing:

Serial output confirmed accurate communication both ways.

Code Snippet (Slave):

```
void receiveEvent(int bytes) {
  inputVal = Wire.read();
}

void requestEvent() {
  if (inputVal > 100) Wire.write(2);
  else Wire.write(4);
}
```



```
Sent: 96
Received: 4
Sent: 97
Received: 4
Sent: 98
Received: 4
Sent: 99
Received: 4
Sent: 100
Received: 4
Sent: 101
Received: 2
Sent: 102
Received: 2
Sent: 103
Received: 2
```

Assignment C: Arduino as Register-Based Slave

Objective:

Simulate a custom slave device with virtual registers that compute and store min(a, b) and max(a, b).

Register Map (From Practicum):

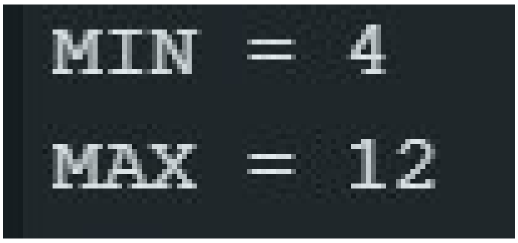
Address	Register	Function	Access
0x21	INA	Input A	R/W
0x22	INB	Input B	R/W
0x23	MIN	min(A, B)	Read Only
0x24	MAX	max(A, B)	Read Only

Implementation:

- On receiving data, INA and INB are updated.
- On request from master, the slave computes and sends either MIN or MAX.
- Registers implemented using a switch-case on the address.

Testing:

Values were verified using simple tests like A=12, B=4 → MIN=4, MAX=12.



```
MIN = 4
MAX = 12
```

Assignment D: Multi-Slave I2C Demonstration

(Referenced from Assignment D, Page 4)

Objective:

Combine all components (sensor + register slave) on the same I2C bus to demonstrate bus sharing.

I2C Setup:

- BME280 Sensor at address 0x76
- Register Slave at address 0x77
- Shared SDA and SCL lines across all devices

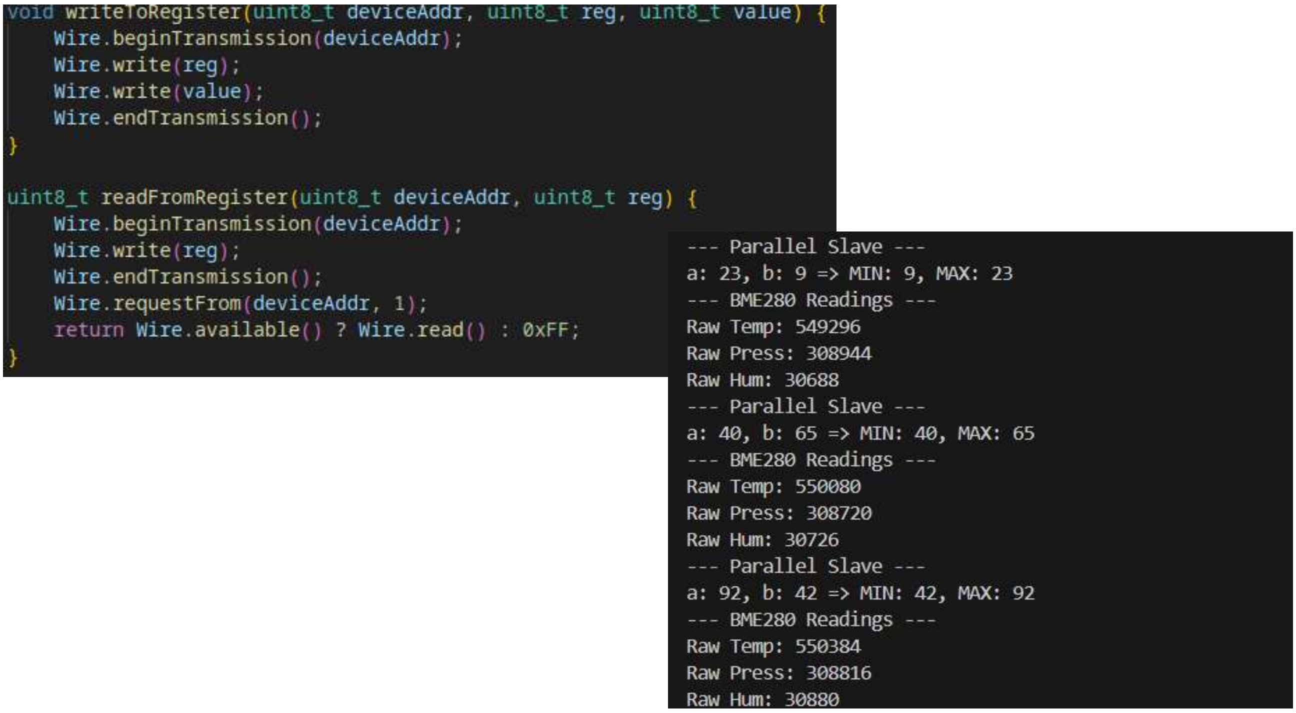
Master Operation:

1. Initializes BME280, reads sensor data
2. Sends INA/INB values to the computation slave
3. Reads back MIN/MAX results
4. Displays all data to Serial

Scalability:

- Successfully tested with two slaves
- Can be extended by adding more computation slaves at different addresses

Code Snippet (Master Overview):



```
void writeToRegister(uint8_t deviceAddr, uint8_t reg, uint8_t value) {
    Wire.beginTransmission(deviceAddr);
    Wire.write(reg);
    Wire.write(value);
    Wire.endTransmission();
}

uint8_t readFromRegister(uint8_t deviceAddr, uint8_t reg) {
    Wire.beginTransmission(deviceAddr);
    Wire.write(reg);
    Wire.endTransmission();
    Wire.requestFrom(deviceAddr, 1);
    return Wire.available() ? Wire.read() : 0xFF;
}
```

```
--- Parallel Slave ---
a: 23, b: 9 => MIN: 9, MAX: 23
--- BME280 Readings ---
Raw Temp: 549296
Raw Press: 308944
Raw Hum: 30688
--- Parallel Slave ---
a: 40, b: 65 => MIN: 40, MAX: 65
--- BME280 Readings ---
Raw Temp: 550080
Raw Press: 308720
Raw Hum: 30726
--- Parallel Slave ---
a: 92, b: 42 => MIN: 42, MAX: 92
--- BME280 Readings ---
Raw Temp: 550384
Raw Press: 308816
Raw Hum: 30880
```

Conclusion

This practicum effectively demonstrated:

- Direct I2C sensor interfacing from datasheet
- Multi-device communication over shared I2C lines
- Custom register-mapped logic in software
- Master-slave synchronization and data flow

All tasks were implemented using the Arduino platform, and testing confirmed the stability and scalability of the I2C-based system.