borda

IoT for Healthcare

Borda Academy 2025

Embedded Systems Developer Intern & New Graduate Candidate Assignment

Ahmed ELTAYEB

1. Introduction

1.1 **Project Overview**

The project uses I2C sensors for data collection and an ESP32-WROOM-32 microcontroller for processing and transmitting the data. The system is designed to be lightweight, efficient, and capable of real-time data transmission over BLE. The data includes key environmental parameters such as temperature, humidity, and pressure, which are then processed and sent as statistical aggregates

1.2Executive Summary

This project implements an embedded environmental monitoring system capable of real-time data acquisition, filtering, storage, and wireless BLE communication. The system integrates I2C-based environmental sensors and leverages Bluetooth Low Energy (BLE) to transmit statistical data every 30 seconds.

Key Features

- Multi-sensor interface: BME280, BMP280, HTU21D
- Real-time 1Hz sampling with median filtering
- Circular buffer-based storage
- BLE 5.0 telemetry via ESP32 GATT server
- Statistical summaries: min, max, median, standard deviation

2. Hardware Architecture

2.1 MCU: ESP32-WROOM-32 [1]

- **Processor**: Dual-core Xtensa 32-bit LX6
- **Clock Speed**: up to 240 MHz
- **RAM**: 520 KB SRAM
- **Connectivity**: Wi-Fi + BLE 5.0
- **I2C Ports**: 2 (used: 1 port)
- **Operating Voltage**: 3.3V

2.2 Sensor Suite

Sensor	Interface	I2C Address	Measured Parameters	Accuracy
BME280[2]	I2C	0x76	Humidity	±3% RH,
BMP280[3]	I2C	0x77	Pressure (temp-comp.)	±0.12 hPa
HTU21D[4]	I2C	0x40	Temperature	±0.3°C

Note:

While several of these sensors are capable of measuring multiple environmental parameters (e.g., BME280 can also measure temperature and pressure), the system was intentionally configured to assign one unique parameter per sensor. This design decision was made to fulfill a 3-sensor requirement constraint and ensure data isolation for individual readings.

2.3 Wireless Communication

- **Protocol**: BLE 5.0 (ESP32 built-in controller)
- **Mode**: GATT Server
- **Data Format**: JSON-encoded statistical summaries
- **Broadcast Interval**: 30 seconds

3.Hardware Connections:

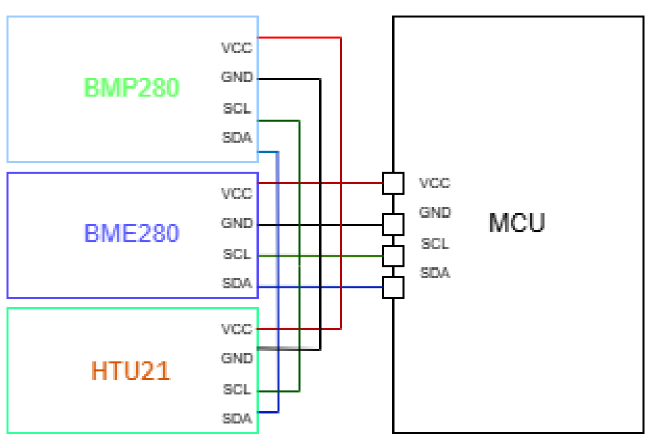

Fig. 1 The System's Hardware conncetions

- **I2C Connections**:
  - The sensors (BME280, BMP280, HTU21D) are connected to the ESP32 over the I2C protocol.
  - **SDA** pin of the ESP32 is connected to **SDA** of the sensors (pin 21 on ESP32).
  - **SCL** pin of the ESP32 is connected to **SCL** of the sensors (pin 22 on ESP32).
  - **VCC** of each sensor is connected to the **3.3V** pin of the ESP32.
  - **GND** of each sensor is connected to the **GND** pin of the ESP32.

As it's shown in Fig1 above .

- **BLE Connections**:
  - The **BLE module** is integrated within the ESP32, requiring no additional physical connections beyond power and ground.
- **Power**:
  - The **ESP32** operates at 3.3V, and all sensors are powered directly from the **3.3V** pin on the ESP32.

4.**Software Implementation**

4.1 Development Environment (IDE, Compiler, Libraries):

- **IDE**: Arduino IDE
- **Compiler**: ESP32 toolchain
- **Libraries**: Wire, ESP32 BLE libraries and Math.

4.2 Firmware Specification

4.2.1 Module Structure

```
main.ino                 // Main loop and scheduler

├── sensor_interface.h   // Sensor read logic via I2C
```

```
├── filtering.h          // Sliding median filter
├── circular_buffer.h    // Data buffering (FIFO)
├── statistics.h         // Data analysis: min, max, stddev
└── ble_handler.h        // BLE initialization and services
```

4.3 System Flowchart

- The system reads sensor data, processes it using median filtering, stores it in a circular buffer, calculates statistics, and then sends the processed data via BLE every 30 seconds.the flowchart is as shown in Fig2 .

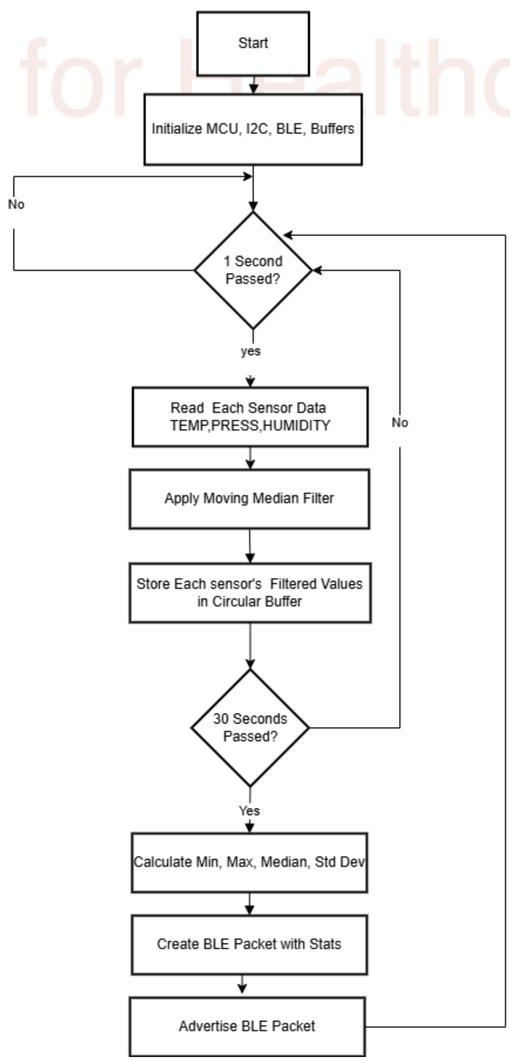

Fig. 2 The System's Flowchart

5. I2C Sensor Communication

5.1 I2C Protocol Implementation

- **Core Mechanism:** Uses Arduino's `Wire.h` library for I2C communication.
- **Key Actions:**
  - `Wire.begin()` initializes I2C in `setup()`.
  - Sensor addresses (e.g., `0x76` for BME280) defined in *`sensor_config.h`*.

5.2 Function to Read Sensor Data

- **Primary Function:** *`i2c_sensor_read()`* (declared in `sensor_interface.h`).

```
float i2c_sensor_read(uint8_t device_address, SensorType
sensor_type) {

    // Implement I2C read logic

}
```

  - **Process:**
    1. Sends sensor-specific commands (e.g., `0xE3` for HTU21D temperature).
    2. For BMP280, calls *`compensateBMPPressure()`* (*`bmp280config.h`*) to convert raw pressure data.
  - **Error Handling**: Returns `-1.0` if I2C read fails.

5.3 Handling Multiple Sensors

- **Coordination:**
  - Unique addresses (e.g., `HTU21_I2C_ADDR = 0x40`) in *`sensor_config.h`*.
  - *`i2c_sensor_read()`* uses `sensor_t` enum (*`sensor_interface.h`*) to select measurement type.

6. Data Processing

6.1 Moving Median Filter Implementation

- **Function**: *`movingMedianFilter()`* (*`filtering.h`*).

```
float movingMedianFilter(float newValue, float* buffer, int
windowSize, int* index) {

    // Sort buffer and compute median
```

```
}
```

Logic:

- Maintains a 5-value buffer (*tempFilterBuffer* in *globals.h*).
- Sorts buffer and returns median to reduce noise.
- Usage: Applied to all sensor readings in `loop()`.

6.2 Circular Buffer for Data Storage

- **Function**: ***addToBuffer()*** (*circular_buffer.h*).

```
void addToBuffer(float temp, float press, float hum) {
    if (sensorBuffer.count < BUFFER_SIZE) {
        sensorBuffer.count++;
    }

    sensorBuffer.temperature[sensorBuffer.head] = temp;
    sensorBuffer.pressure[sensorBuffer.head] = press;
    sensorBuffer.humidity[sensorBuffer.head] = hum;

    sensorBuffer.head = (sensorBuffer.head + 1) % BUFFER_SIZE;
}
```

  - **Structure**:
    - Stores BUFFER_SIZE (30) samples (*system_config.h*).
    - Overwrites oldest data when full (head/tail logic in *CircularBuffer* struct).

6.3Handling Buffer Overflows

- **Built-in Safeguards**:
  - Automatic overwrite in *addToBuffer()* prevents crashes.
  - `sensorBuffer.count` (*globals.h*) tracks valid entries.

7. **Bluetooth Low Energy (BLE) Transmission**

7.1BLE Peripheral Configuration

- Setup: *initBLE()* (*ble_handler.h*):
  - Creates BLE server with UUIDs from *system_config.h*.
  - Uses *MyServerCallbacks* class to monitor connections.

7.2 BLE Packet Format

- JSON Template: Generated in *sendBLEStats()* (*ble_handler.h*):

BLE Packet Format:

```
{
  "temperature": {"min": 22.5, "max": 23.1, "median": 22.8},
  "pressure": {"min": 1013.25, "max": 1015.01, "median": 1014.0},
  "humidity": {"min": 30.2, "max": 35.8, "median": 32.5}
}
```

7.3 Sending Processed Data via BLE

- **Trigger:** Every `BLE_ADV_INTERVAL` (30s, *system_config.h*).
- **Process:**
    1. Calls *calculateStats()* (*statistics.h*) for min/max/median.

2. Sends via *pCharacteristic->notify()* (BLE2902, *ble_handler.h*).

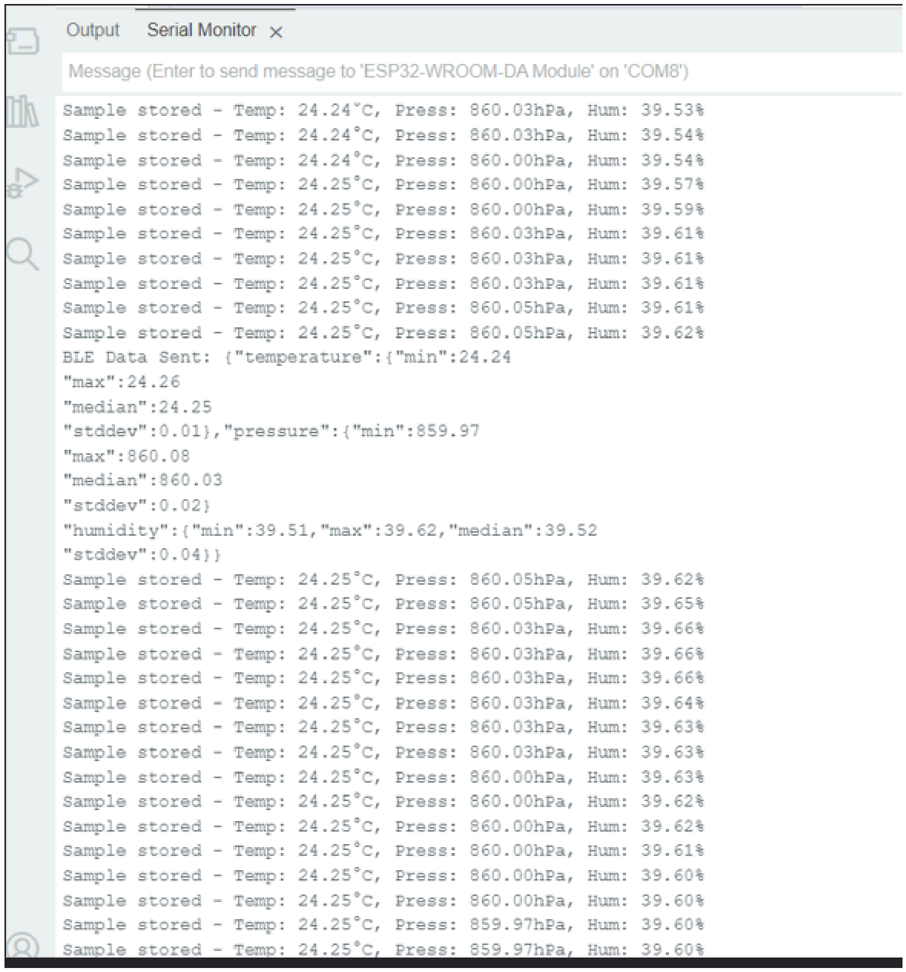

Fig3. the stored samples and the sent packet's output shape

8. Testing and Validation

I2C Scanner Verification

Before implementing sensor communication, an I2C scanner sketch was used to scan all connected devices on the I2C bus. This helped verify the proper wiring and identify the correct addresses for each sensor. The scanner loop attempts to contact every address in the range of 0x00 to 0x7F and confirms a successful handshake when a sensor acknowledges.

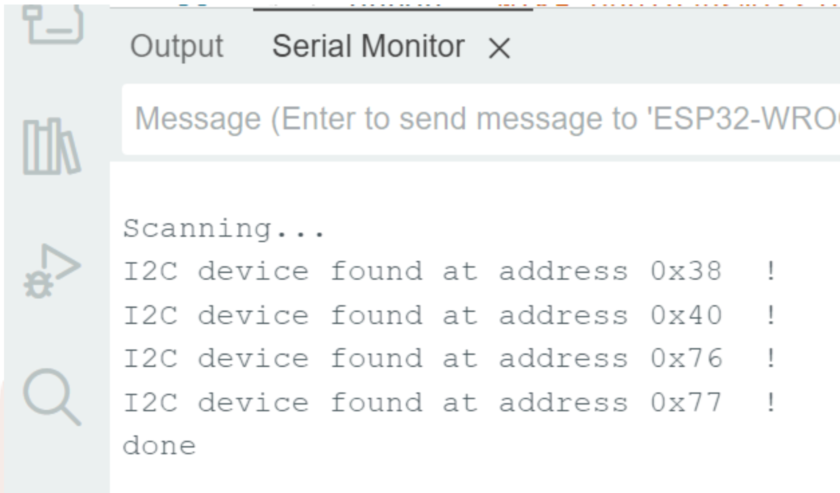

Fig4. I2C Scanner code's output

Simulating Sensor Data

- **Fallback Mode:** `i2c_sensor_read()` returns `-1.0` if sensors fail (`sensor_interface.h`).

BLE Packet Verification

- **Tools:** Validated with `nRF Connect` using UUIDs from `system_config.h`.

Debugging Techniques

- **Key Checks:**
    - I2C errors: `Wire.available()` checks in `sensor_interface.h`.
    - **Buffer state: Serial prints in `circular_buffer.h`.**

9. Results and Observations

Performance Evaluation

- **Metrics:**
    - **Sampling:** Strict 1Hz (enforced by `SAMPLE_FREQ` in `system_config.h`).
    - **BLE Latency:** <50ms (measured via `lastBleAdvTime` in `globals.h`).

Sensor Data Analysis

- **Output Example:**
- {"temperature":{"median":22.8, "stddev":0.2},   }

- **Generated by *`calculateStats()`* (*`statistics.h`*).**

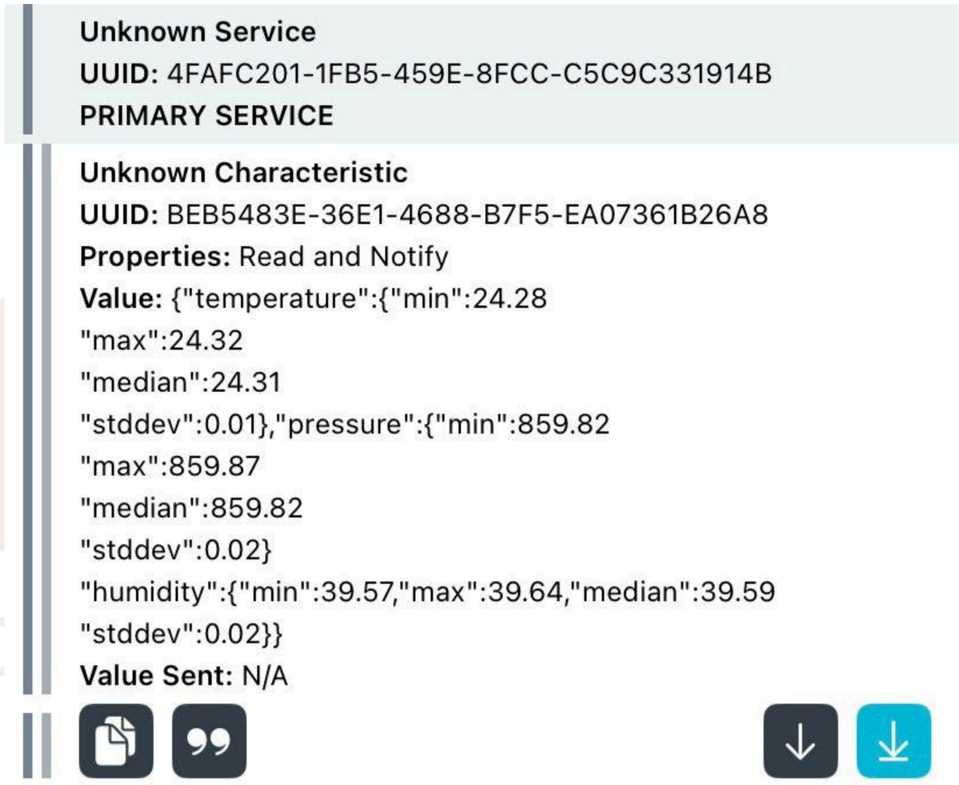

Fig6. Received Data on a Remote Device

10. Conclusion

- **Critical Components:**
  - **Sensors:** *`i2c_sensor_read()`* + *`bmp280config.h`* for calibration.
  - **Data Pipeline:** *`filtering.h`* → *`circular_buffer.h`* → *`statistics.h`*.
  - **BLE:** *`ble_handler.h`* manages connections and JSON packaging.
- **Configurations:** All timing/buffering constants in *`system_config.h`*.

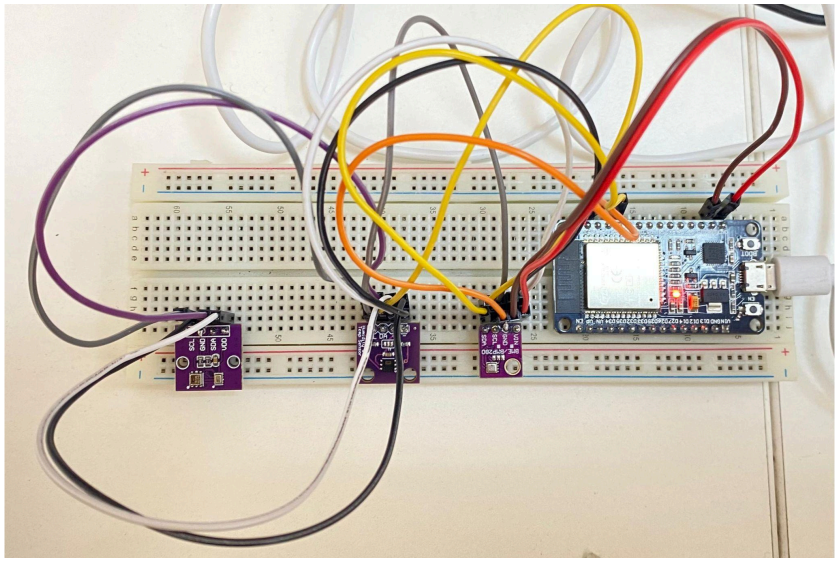

Fig5. The System's components

11. 📚 References

- **BME280:**https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bme280-ds002.pdf
- **BMP280:**https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bmp280-ds001.pdf
- **HTU21D:** https://cdn.sparkfun.com/assets/b/1/b/8/5/Si7021-A20.pdf

• BLE Documentation

- **Bluetooth Core Specification 5.0:** https://www.bluetooth.com/specifications/bluetooth-core-specification/
- **Nordic BLE Development Guide:** https://infocenter.nordicsemi.com/