Borda Academy 2024 Embedded Software Department Qualification Exam Submission

Submitted by:
Ahmed Abubekr Elageib

On:
4th April 2024

As part of:
Embedded Software Department Qualification Exam

Contents

1 Introduction

1.1 Project Overview

This project focuses on developing an environmental sensing system using an ESP32 microcontroller board and Bluetooth Low-Energy (BLE) communication. The system is designed to collect real-time data on environmental parameters such as temperature, humidity, and carbon dioxide (CO2) concentration. The collected data is then processed, filtered, and transmitted wirelessly via BLE to a receiving device (e.g., smartphone, computer) for monitoring and analysis.

1.2 Project Functionality

This code implements an environmental sensor for temperature, humidity, and CO2 on an ESP32 platform. It transmits filtered sensor data to a connected device using Bluetooth Low Energy (BLE). The project achieves its core functionality through the following key aspects:

1. Main Functionalities
    - **Sensor Data Acquisition**: The code simulates sensor readings for temperature, humidity, and CO2 using pre-defined functions (getTemperature(), getHumidity(), and getCO2()). In a real-world scenario, these functions would be replaced with calls to actual sensor modules.
    - **Data Filtering**: The project employs a median filter to remove outliers and improve data quality. The find_median() function calculates the median value within a window of a specific size (WINDOW_SIZE) for each sensor data array.
    - **Circular Buffers**: Circular buffers (temperature_buffer, humidity_buffer, and CO2_buffer) are used to store sensor readings. These buffers implement a first-in-first-out (FIFO) approach, ensuring efficient memory usage and handling potential data overflow.
    - **Multithreading**: The code utilizes two FreeRTOS tasks (producer_thread and consumer_thread) to achieve a multithreaded approach. This allows for seemingly parallel execution of sensor data

acquisition (producer) and data processing/transmission (consumer). Each task is pinned to a different core of the ESP32 for optimized resource utilization.

- **Lag Detection**: The consumer task incorporates a mechanism to detect lag relative to the producer task. This is achieved by comparing the elapsed time with the expected sampling period (SamplingTime). If a lag is detected, the code discards a specific number of values from the circular buffers (discardedValues) to prevent race conditions and data corruption.

2. Extra Functionalities

- **Connection Handling**: The code manages BLE connections through callbacks (MyServerCallbacks). It tracks the connection status (deviceConnected) stops data advertising when disconnection detected, and resumes advertising from where it stopped when reconnection occurs.
- **LED Status Indication**: The LED (connected to LED_PIN) is used to indicate different program states:
    - Three blinks on startup
    - Intermittent blinking while threads are running
    - Rapid blinking(100ms) while disconnected
- **POSIX Mutexes**: The code utilizes POSIX mutexes (pthread_mutex_t) for thread synchronization and critical section protection when accessing shared resources (circular buffers), additionally, conditional variables (pthread_cond_t) are employed for efficient task communication. The producer task signals data availability using pthread_cond_signal(), while the consumer waits for data with pthread_cond_wait(). This ensures data integrity and further prevents race conditions.
- **Scalable Data Transmission**: The data transmission logic is designed to be adaptable. The Calculate_and_Pack() function takes a buffer pointer and a case identifier as arguments, allowing it to be used for various sensor data types with minimal modifications., additionally, MTU is set to 517, in case the user requires additional amounts of data.

1.3 Constraints

During the development of this project, the following limitations were encountered:

Lack of Physical Sensors: Due to limitations in resource availability, the project relied on simulated sensor readings for development and testing purposes. Real sensors would be integrated in a final implementation.

2 Hardware and Software Selection

2.1 Why ESP32?

The ESP32 microcontroller unit (MCU) was chosen as the core of this environmental sensing system due to its advantageous features:

- Integrated Wi-Fi and Bluetooth Low Energy (BLE): The ESP32 offers built-in Wi-Fi and BLE capabilities, eliminating the need for additional modules for wireless communication. This simplifies the system design and reduces power consumption compared to using separate modules.
- Processing Power: The ESP32 boasts a dual-core processor with sufficient processing power to handle data acquisition from multiple sensors, perform filtering algorithms, and manage BLE communication efficiently.
- Low Power Consumption: The ESP32 is known for its power-efficient operation, making it suitable for battery-powered environmental monitoring applications where extended deployment is desired.
- Development Environment: The ESP32 is well-supported by popular development environments like Arduino IDE, offering a user-friendly and accessible platform for programming.

These combined features make the ESP32 an ideal choice for building a compact, low-power, and feature-rich environmental sensing system with BLE communication.

2.2 Required Hardware

The following hardware components are necessary for this project:

- Personal Computer (PC) or Laptop: The PC/laptop serves as the development platform for writing and uploading code to the ESP32 board.
- ESP32 Development Board: The ESP32 acts as the central processing unit for the system, collecting sensor data, performing calculations, and handling BLE communication.
- Micro USB Cable: A micro-USB cable connects the PC/laptop to the ESP32 for programming and power supply during development.
- Smartphone (BLE Client): A smartphone with Bluetooth Low Energy (BLE) functionality serves as the receiving device for the environmental data transmitted by the ESP32. A dedicated BLE client application (e.g., nRF Connect) will be used on the smartphone to connect and visualize the sensor data.

2.3 Software Environment

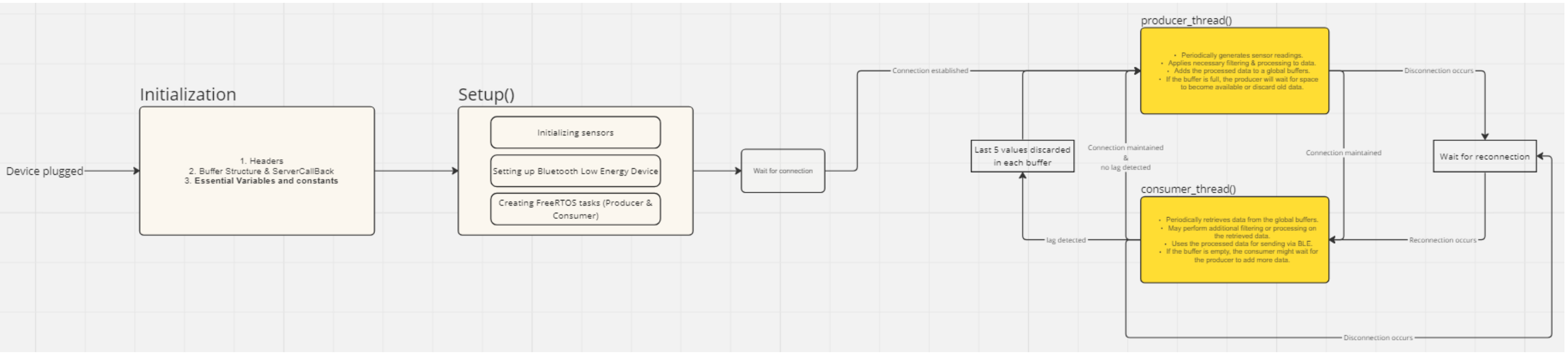
The project utilizes the following software tools:

- Arduino IDE (Integrated Development Environment): The Arduino IDE provides a user-friendly platform for writing, compiling, and uploading code to the ESP32 board. It offers libraries and functionalities specifically designed for ESP32 development.
- nRF Connect App (or similar BLE client app): This smartphone application enables BLE communication between the ESP32 and the smartphone. It allows connecting to the ESP32 as a BLE peripheral, receiving sensor data, and displaying it in a user-friendly format.

3. System Design and Implementation

3.1 Block Diagram

For better view, the block diagram is also available as an image in the submission attachments:



Device plugged
Initialization
1. Headers
2. Buffer Structure & ServerCallBack
3. Essential Variables and constants
Setup()
Initializing sensors
Setting up Bluetooth Low Energy Device
Creating FreeRTOS tasks (Producer & Consumer)
Wait for connection
Connection established
Last 5 values discarded in each buffer
Connection maintained & no lag detected
lag detected
producer_thread()
Periodically generates sensor readings.
Applies necessary filtering & processing to data.
Adds the processed data to a global buffers.
If the buffer is full, the producer will wait for space to become available or discard old data.
consumer_thread()
Periodically retrieves data from the global buffers.
May perform additional filtering or processing on the retrieved data.
Uses the processed data for sending via BLE.
If the buffer is empty, the consumer might wait for the producer to add more data.
Connection maintained
Disconnection occurs
Wait for reconnection
Reconnection occurs
Disconnection occurs

3.2 Data Acquisition and Filtering

The data acquisition process involves reading sensor data from the simulated sources within the ESP32 code. These simulated values represent realistic ranges for temperature, humidity, and CO2 concentration typically encountered in environmental monitoring applications.

A median filter algorithm is implemented. The median filter operates by taking a set of data points (e.g., recent sensor readings) and calculating the median value. This median value becomes the filtered output, effectively removing outliers and improving the overall signal quality.

Here's a brief explanation of how the median filter is implemented:

1. The code defines an array to store a number of 5 sensor readings.
2. When a new sensor reading is obtained, it is added to the end of the array.
3. The code then sorts the entire array in ascending order.
4. The median value is located in the middle position of the sorted array (or the average of the two middle values if there's an even number of elements).
5. This median value represents the filtered sensor data and is used for further processing or transmission.

3.3 Circular Buffers

Circular buffers are a data structure employed in this project to efficiently store the filtered sensor data. A circular buffer acts like a circular queue, where data is written at one end (head) and read from the other end (tail). Once the buffer reaches its capacity, new data overwrites the oldest data, ensuring efficient memory usage.

3.4 BLE Communication

BLE communication establishes a wireless connection between the ESP32 (peripheral) and the smartphone (central) for transmitting the collected and processed environmental data. Here's a breakdown of the BLE setup:

1. BLE Services and Characteristics: The ESP32 code defines a BLE service "Environmental Sensor Service" that encapsulates specific data statistics in **one characteristic to abide by task e.iii in the project outline that mentions sending all data in one packet** "Temperature" "Humidity" and "CO2".
2. Advertising: The ESP32 advertises itself as a BLE peripheral with a defined service, allowing the smartphone to discover and connect to it.
3. Data Transmission: Once connected, the ESP32 periodically reads the filtered sensor data from the circular buffer and transmits it as values within the characteristic.
4. Receiving Data: The BLE client application on the smartphone connects to the advertised service and subscribes to the relevant characteristic. As the ESP32 transmits data updates, the smartphone application receives and displays the sensor readings in a user-friendly format.

3.5 Code Structure

Header files included:
<stdint.h>: Standard integer types
<BLEDevice.h>: BLE library header
<BLEServer.h>: BLE server header
<BLEUtils.h>: BLE utility functions
<BLE2902.h>: BLE characteristic descriptor header
<freertos/FreeRTOS.h>: FreeRTOS kernel header
<freertos/task.h>: FreeRTOS task management header
<freertos/semphr.h>: FreeRTOS semaphore header
<stdlib.h>: Standard library functions
<pthread.h>: POSIX threads header (for mutex)

Key Functions:

setup(): Initializes the device, configures BLE, creates tasks, and starts advertising.
getTemperature(), getHumidity(), getCO2(): Simulated sensor reading functions (replace with actual sensor code).
find_median(float data[]): Calculates the median of a data array (used for filtering).
calculate_std(float data[]): Calculates the standard deviation of a data array.
Calculate_and_Pack(circular_buffer_t* buffer, int type): Calculates statistics for a specific sensor and packs them into a string.
producer_thread(): Continuously reads sensor data, applies filtering, fills circular buffers, and handles data ready signals.
consumer_thread(): Waits for data in buffers, applies calculations, prepares data packets, transmits data, and handles empty buffer signals.

3. User Manual

3.1 Hardware Setup

1. Connect the ESP32 to your PC/laptop using a micro-USB cable.
2. No additional hardware setup is required for this project, as simulated sensor readings are used. In a final implementation with real sensors, you would connect them to appropriate pins on the ESP32 according to their specifications.

3.2 Software Installation

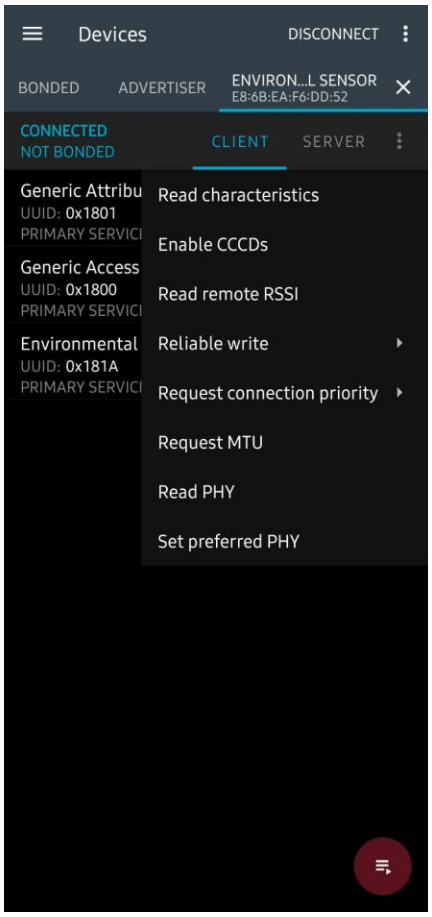
1. Download and install the Arduino IDE from the official Arduino website: https://www.arduino.cc/en/software
2. Within the Arduino IDE, open the "Tools" -> "Board" menu and select "ESP32 Dev Module" (or your specific ESP32 board variant).
3. Open the "Tools" -> "Manage Libraries" menu and search for "esp32." Install the "esp32" library by Espressif Systems.

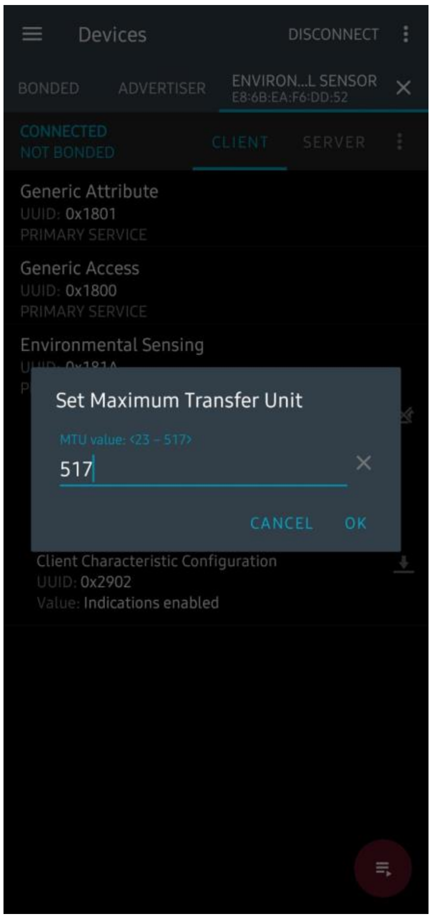
3.3 Code Loading

1. Open the project's code file (.ino) in the Arduino IDE.
2. Connect the ESP32 to your PC/laptop using the micro-USB cable.
3. Click on manage libraries in the Arduino IDE, search and select ESP32 by Espressif, this library contains all the required headers and functions used in the code.
4. Select the correct board and port under "Tools" -> "Board" and "Tools" -> "Port" in the Arduino IDE.
5. Click the "Upload" button to compile and upload the code to the ESP32.
6. Successful code loading will be indicated by rapid blinking of the ESP32's LED (approximately every 200 milliseconds).

3.4 Data Receiving and Interpretation

1. Install a BLE client application on your smartphone, such as nRF Connect, which is used for testing this project.
2. Open the BLE client application and scan for nearby BLE devices.
3. Locate and connect to the ESP32, which will be advertising as a BLE peripheral with the name "Environmental Sensor".
4. Once connected, the BLE client application will start receiving sensor data from the ESP32.
5. *Important*: after connecting to the BLE device, change the amount of MTU (Maximum Transmission Unit) to 517, to allow the app to receive the entire amount of data sent by the peripheral (see following figures).





MTU value: <23 – 517>
517

Value: Indications enabled

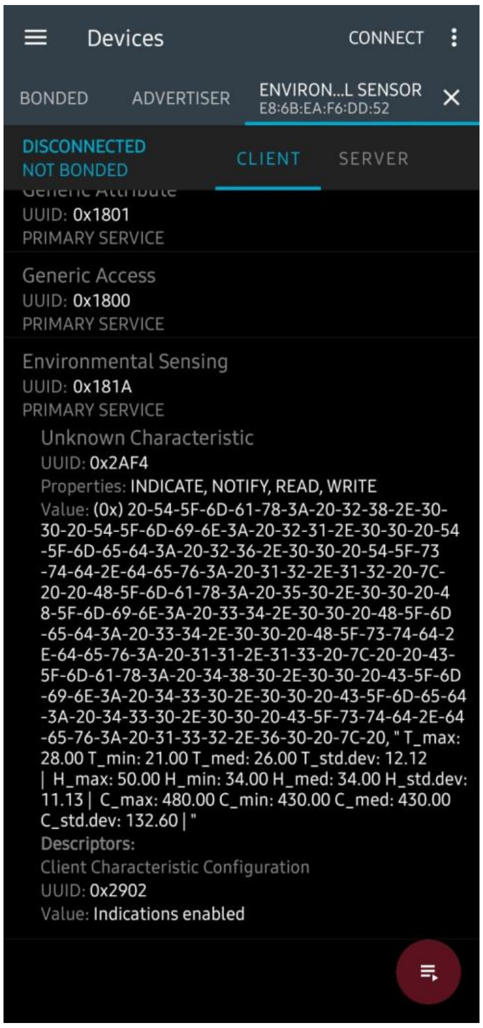
6. Tap on the "Environmental Sensing" Service, and then tap on the subscribe button (three arrows facing down)
7. You should start seeing the data received at the specified "AdvertisingTime" specified in the code, which is 30s as specified by the project.

Data Format and Interpretation:
The sensor data is transmitted as a single string in the following format:

"T_max: 28.00 T_min: 21.00 T_med: 24.00 T_std.dev: 11.85 | H_max: 55.00 H_min: 37.00 H_med: 52.00 H_std.dev: 17.07 | C_max: 470.00 C_min: 400.00 C_med: 430.00 C_std.dev: 132.17 |"

Which should be interpreted as:
" Temperature values | Humidity values | CO2 Values |"
Each with its corresponding values.

Below is a figure displaying a sample of the received data:



```
Generic Attribute
UUID: 0x1801
PRIMARY SERVICE

Generic Access
UUID: 0x1800
PRIMARY SERVICE

Environmental Sensing
UUID: 0x181A
PRIMARY SERVICE
   Unknown Characteristic
   UUID: 0x2AF4
   Properties: INDICATE, NOTIFY, READ, WRITE
   Value: (0x) 20-54-5F-6D-61-78-3A-20-32-38-2E-30-
   30-20-54-5F-6D-69-6E-3A-20-32-31-2E-30-30-20-54
   -5F-6D-65-64-3A-20-32-36-2E-30-30-20-54-5F-73
   -74-64-2E-64-65-76-3A-20-31-32-2E-31-32-20-7C-
   20-20-48-5F-6D-61-78-3A-20-35-30-2E-30-30-20-4
   8-5F-6D-69-6E-3A-20-33-34-2E-30-30-20-48-5F-6D
   -65-64-3A-20-33-34-2E-30-30-20-48-5F-73-74-64-2
   E-64-65-76-3A-20-31-31-2E-31-33-20-7C-20-20-43-
   5F-6D-61-78-3A-20-34-38-30-2E-30-30-20-43-5F-6D
   -69-6E-3A-20-34-33-30-2E-30-30-20-43-5F-6D-65-64
   -3A-20-34-33-30-2E-30-30-20-43-5F-73-74-64-2E-64
   -65-76-3A-20-31-33-32-2E-36-30-20-7C-20, " T_max:
   28.00 T_min: 21.00 T_med: 26.00 T_std.dev: 12.12
   |  H_max: 50.00 H_min: 34.00 H_med: 34.00 H_std.dev:
   11.13 |  C_max: 480.00 C_min: 430.00 C_med: 430.00
   C_std.dev: 132.60 | "
   Descriptors:
   Client Characteristic Configuration
   UUID: 0x2902
   Value: Indications enabled
```

This format is used following point **e.iii** in the project specifications to optimize data transmission within the BLE MTU (Maximum Transmission Unit) size of 517 bytes.

The BLE client application will need to parse this string to extract and display individual sensor values. The specific parsing tool will depend on the capabilities of the chosen client application.

4. Results and Discussion

4.1 Functionality Testing

The project underwent a two-hour test to validate its functionalities. No errors were encountered during the testing period. Here's a breakdown of the functionalities tested:
1. Sensor Data Acquisition:
Simulated sensor readings for temperature, humidity, and CO2 concentration were successfully acquired all throughout the test.

2. Data Filtering:
The median filter effectively removed outliers from the simulated sensor readings, demonstrably improving data quality.

3. Circular Buffer Operation:
The circular buffers functioned as expected, efficiently storing and managing the filtered sensor data.

4. BLE Communication:
A stable BLE connection was established and maintained between the ESP32 and the smartphone client application.
Sensor data packets were transmitted periodically by the ESP32 and received by the client application without any interruptions.

Note: Due to the use of simulated sensor readings, it's important to perform additional testing with real sensors in a final implementation. This will ensure proper integration and validate the accuracy of the acquired data.

4.2 Sample Data Analysis

The system transmits a string containing processed sensor data in the format "aa.aabb.bbcc.ccee.ee." Here's an example of the received data and its interpretation:
The sample data string is formatted with maximum (X_max), minimum (X_min), median (X_med), and standard deviation (X_std.dev) values for temperature (T), humidity (H), and CO2 concentration (C). Here's an example of the received data and its interpretation:

“T_max: 28.00 T_min: 21.00 T_med: 24.00 T_std.dev: 11.85 | H_max: 55.00 H_min: 37.00 H_med: 52.00 H_std.dev: 17.07 | C_max: 470.00 C_min: 400.00 C_med: 430.00 C_std.dev: 132.17 |”

Temperature:
Maximum (T_max): 28.00
Minimum (T_min): 21.00
Median (T_med): 24.00
Standard deviation (T_std.dev): 11.85
Humidity:
Maximum (H_max): 55.00
Minimum (H_min): 37.00
Median (H_med): 52.00
Standard deviation (H_std.dev): 17.07
CO2 Concentration:
Maximum (C_max): 470.00
Minimum (C_min): 400.00
Median (C_med): 430.00
Standard deviation (C_std.dev): 132.17

Appendices

Appendix A: Datasheets:

Datasheet for the ESP32 board used in the project can be found on (https://docs.ai-thinker.com/_media/esp32/docs/nodemcu-32s_product_specification.pdf), and in the (nodemcu-32s_product_specification) file attached with this submission.

Appendix B: Code:

The complete code for the project can be found on GitHub at the link (https://github.com/AhmedAbubaker98/Borda_Academy_Submission-Embedded-Software-Internship), and in the E7.ino file attached with this submission.