COMP-SCI 5577
Internet of Things
Mini-Project3 Report

Professor: Dr Qiuye He
Term members:
Hui Jin
Jiarui Zhu
Harshitha Bashyam
Mar 10, 2025
Github Link: https://github.com/nanxuanhui/Iot_miniproject.git

Content

Introduction

This project aims to use ESP32 as a BLE (Bluetooth Low Energy) sensor device to wirelessly transmit environmental data (temperature and humidity) and battery level, while also allowing users to set a humidity threshold through a BLE client (e.g., LightBlue). The project covers hardware circuit design, software implementation, BLE communication protocol, data transmission, and testing verification.

1. Key Features

- DHT11 sensor collects temperature and humidity data and transmits it to the BLE client.
- Simulates battery level reduction and transmits battery status via BLE.
- Allows users to set a humidity threshold via the BLE client, transmitting data only when the humidity exceeds the threshold.
- Tested and validated using LightBlue or other BLE applications.

2. Design Goals

- **Low Power Consumption**

ESP32 as a BLE device should optimize power consumption for long-term operation.

- **Real-time Data Transmission**

The BLE device should transmit temperature and humidity data in real-time and allow subscription by BLE clients.

- **Stable BLE Connection**

The device should automatically re-advertise for reconnection after disconnection.

- **Support User Input**

Users should be able to dynamically set a humidity threshold, ensuring ESP32 only transmits data when the humidity exceeds this threshold.

System Design

1. Hardware Architecture

This system mainly consists of the following hardware components:

- **ESP32 Development Board**

Handles data collection and BLE communication.

- **DHT11 Sensor**

Collects temperature and humidity data.

- **Power Management**

Provides power and simulates battery level changes.

2. Circuit Connections

The connection between ESP32 and DHT11 sensor is as follows:

ESP32	DHT11
3.3V	+
GND	-
GPIO4	out

- ESP32 reads DHT11 sensor data via GPIO4.

- ESP32 transmits temperature and humidity data via BLE.
- Simulates battery level reduction by 1% per minute.

3. BLE Communication Structure

The characteristic design for the BLE device is as follows:

Service	UUID	Characteristic	UUID	Read/Write/Notify
Environmental Sensing	0x181A	Temperature	0x2A6E	Read & Notify
		Humidity	0x2A6F	Read & Notify
Battery Service	0x180F	Battery Level	0x2A19	Read & Notify
Custom Service		Humidity Threshold	0x2A1F	Read & Write

Implementation

1. Key Steps

a. Configure BLE on ESP32

- Initialize BLE device.
- Create BLE services and characteristics.
- Allow BLE clients to subscribe to characteristic value changes

b. Read DHT11 Sensor Data

- Read temperature and humidity.
- Convert to string format and transmit via BLE

c. Handle Humidity Threshold Setting

- Allow BLE clients to write humidity threshold.
- Parse user input and update the humidity filter condition

d. Simulate Battery Consumption

- Reduce battery level by 1% per minute
- Broadcast battery level information via BLE

2. Key Code Snippets

a. BLE Initialization

```
BLEDevice::init("ESP32 Sensor");
BLEServer *pServer = BLEDevice::createServer();
pServer->setCallbacks(new ServerCallbacks());data.
```

b. Reading Temperature and Humidity Data and Sending via BLE

```
float temperature = dht.readTemperature();
float humidity = dht.readHumidity();
char tempStr[8], humStr[8];
snprintf(tempStr, sizeof(tempStr), "%.2f", temperature);
snprintf(humStr, sizeof(humStr), "%.2f", humidity);
pTemperatureCharacteristic->setValue(tempStr);
pTemperatureCharacteristic->notify();
pHumidityCharacteristic->setValue(humStr);
pHumidityCharacteristic->notify();
```

c. Parsing the BLE-Set Humidity Threshold

```
void onWrite(BLECharacteristic *pCharacteristic) {
std::string value = std::string((char*)pCharacteristic->getData(),
pCharacteristic->getLength());
try {
        humidityThreshold = std::stoi(value);
        Serial.print("New Humidity Threshold Set: ");
        Serial.println(humidityThreshold);
} catch (...) {
        Serial.println("Invalid Threshold Input!");
        }
}
```

Testing & Results

1. Testing Steps

a. Connect to LightBlue

- Search for ESP32 in LightBlue.
- Subscribe to temperature, humidity, and battery characteristics.
- Ensure BLE data is transmitted correctly.

b. Modify Humidity Threshold

- Enter 40 in the Humidity Threshold characteristic
- Check if ESP32 correctly parses and updates the threshold

c. Verify Battery Simulation

- Observe battery level change per minute
- Ensure the BLE client can correctly read battery values

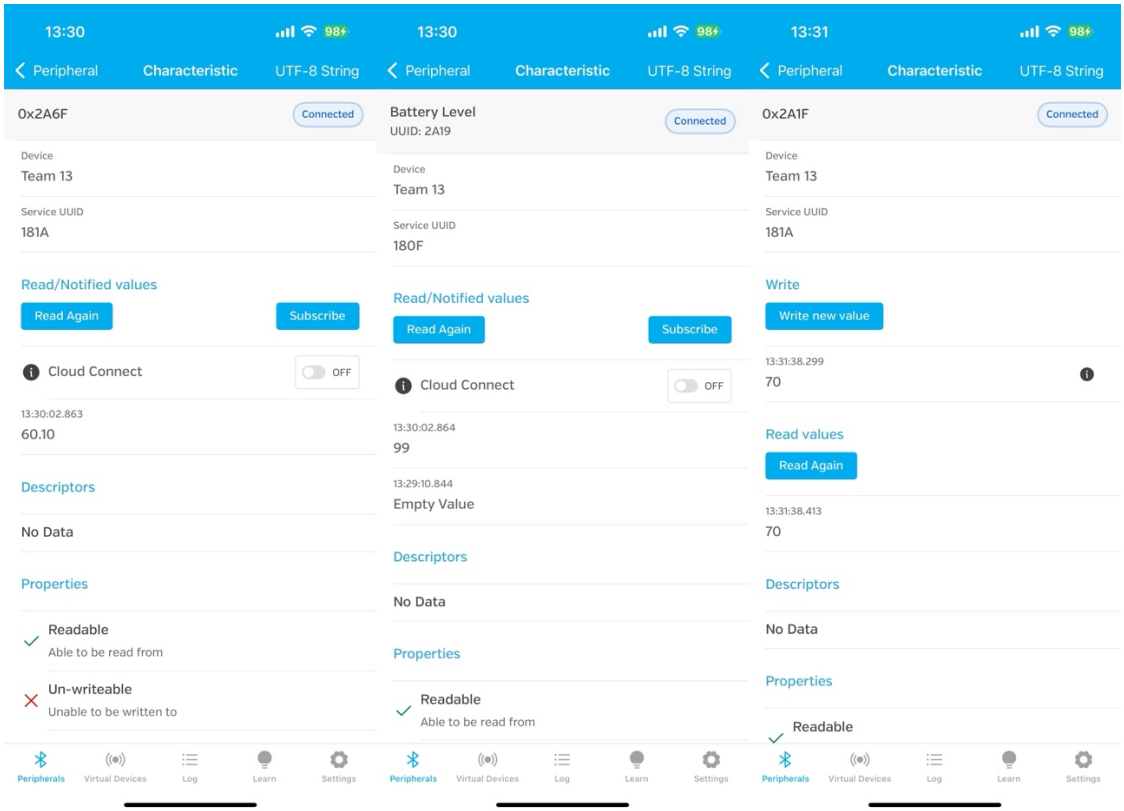
2. Testing Results

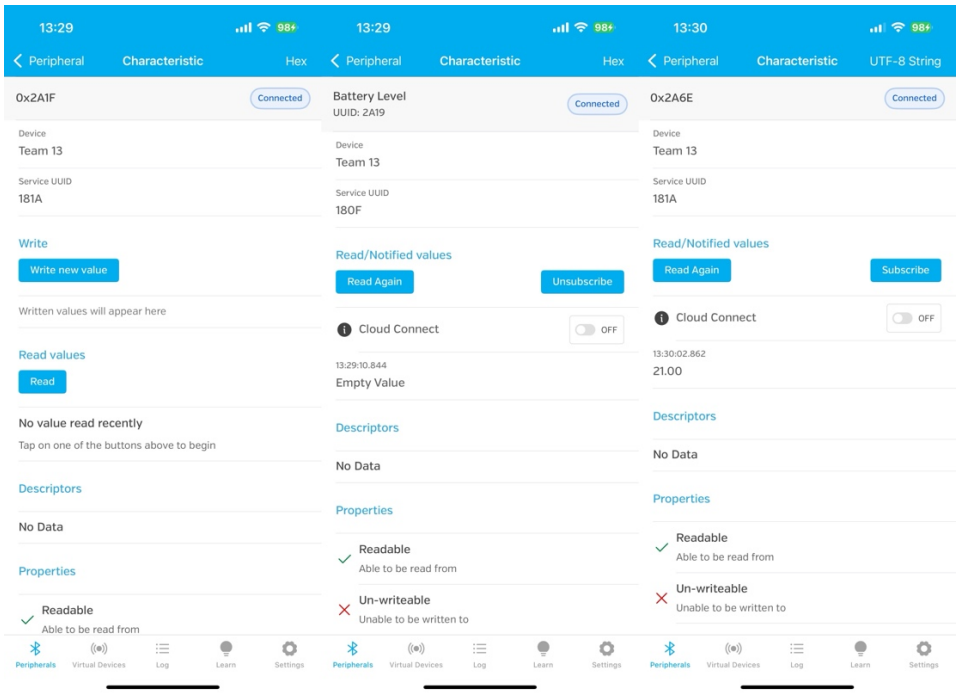
Test Item	Expected Result	Actual Result
Read Temperature & Humidity	Data is correctly retrieved	✅
Set Humidity Threshold	Can be modified and takes effect	✅
BLE Connection Management	Reconnects after disconnection	✅
Battery Simulation	Gradually decreases and transmits data	✅

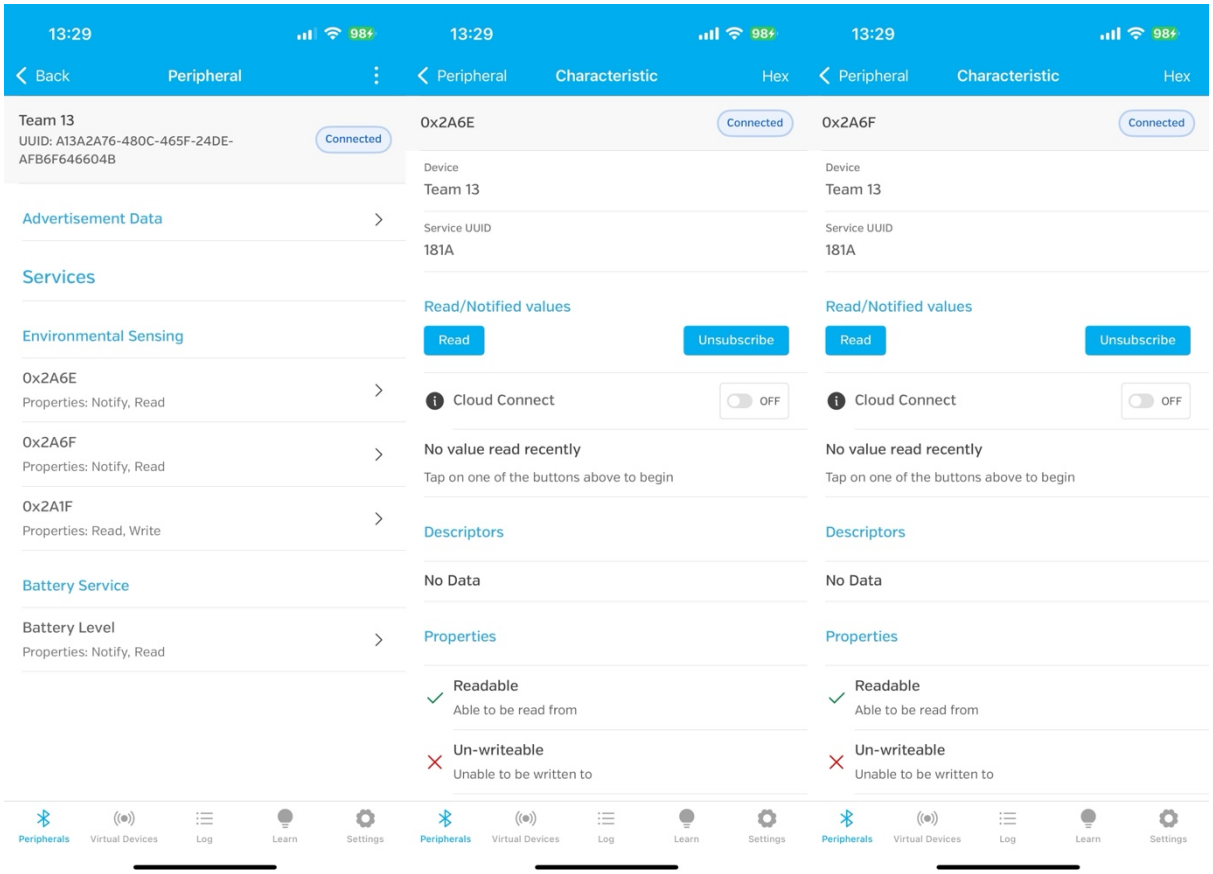
3. Screenshots

```
13:29:03.428 -> rst:BLE Advertising started.
13:29:03.428 -> Measured Temperature: 21.00°C, Device connected
13:30:02.861 -> Measured Temperature: 21.00°C, Measured Humidity: 60.10%
13:30:02.911 -> Sent Temperature: 21.00°C
13:30:02.911 -> Sent Humidity: 60.10%
13:30:02.911 -> Sent Battery Level: 99%
13:30:38.008 -> Received request to update humidity threshold...
13:30:38.008 -> Raw received value: 40
13:30:38.008 -> New Humidity Threshold Set: 40%
13:30:38.008 -> Humidity threshold updated successfully.
13:31:02.888 -> Measured Temperature: 20.00°C, Measured Humidity: 60.10%
13:31:02.889 -> Sent Temperature: 20.00°C
13:31:02.938 -> Sent Humidity: 60.10%
13:31:02.938 -> Sent Battery Level: 98%
```

```
13:31:19.942 -> Device disconnected, restarting advertising...
13:31:21.985 -> Device connected
13:31:38.332 -> Received request to update humidity threshold...
13:31:38.377 -> Raw received value: 70
13:31:38.377 -> New Humidity Threshold Set: 70%
13:31:38.377 -> Humidity threshold updated successfully.
13:32:02.909 -> Measured Temperature: 20.00°C, Measured Humidity: 60.10%
13:32:02.942 -> Sent Temperature: 20.00°C
13:32:02.942 -> Sent Battery Level: 97%
```







Challenges Faced

1. When transmitting data via BLE, the initial format used was binary (int16_t), but LightBlue defaults to UTF-8 string interpretation, causing data to appear as hexadecimal values like 0x7A17 instead of readable temperature and humidity values.
2. When selecting UTF-8 String parsing, the data was incorrectly interpreted, leading to incorrect characters (e.g., z).
3. When setting the humidity threshold through LightBlue (e.g., inputting 40), ESP32 misinterpreted the value, often setting it to 0 or another incorrect value.
4. The initial implementation used atoi() for parsing the string, which failed to correctly interpret the UTF-8 data sent via BLE.
5. After disconnection, the BLE device did not automatically re-advertise, making it impossible for BLE clients to rediscover the ESP32 without manually restarting it.
6. When a BLE client unexpectedly disconnected, the ESP32 still believed the device was connected, causing data transmission failures.
7. The BLE device consumed high power during long-term operation.
8. The battery simulation feature correctly reduced battery levels over time, but ESP32 continued running at high power consumption levels.

Conclusion

This project successfully demonstrates the integration of an ESP32 board with a DHT11 sensor and BLE technology to enable real-time environmental monitoring and wireless data transmission. The system efficiently reads temperature and humidity data, transmits them to a BLE client, and allows users to set humidity thresholds dynamically. Additionally, the project includes a battery level simulation, ensuring a comprehensive demonstration of BLE-based sensor communication.

Through careful design and implementation, the system ensures stable BLE communication, with automatic reconnection upon disconnection and real-time notifications for subscribed clients. The ability to set and dynamically adjust the humidity threshold allows for selective data transmission, optimizing both power consumption and data relevance. The project also demonstrates effective power management strategies, such as simulated battery drain and the potential for deep sleep functionality, which can significantly extend battery life in real-world applications.

During the development process, several challenges were encountered, including BLE data format inconsistencies, parsing issues with humidity threshold settings, and maintaining stable BLE connections after disconnection. These challenges provided valuable learning opportunities, leading to improved data handling, enhanced debugging techniques, and optimized BLE communication strategies. The project now reliably transmits environmental data, accurately interprets user inputs, and ensures seamless connectivity with BLE clients.

The successful completion of this project highlights the versatility of ESP32 as a low-power, cost-effective solution for wireless sensor applications. The knowledge gained through overcoming challenges will be instrumental in further refining BLE communication and power optimization strategies, making this system even more efficient and scalable in future iterations.

Reference

[1] https://lastminuteengineers.com/esp32-pinout-reference/