BLE Weather Station with WiFi Connectivity

Overview

This project uses a Blutooth Low Energy(IEEE 802.15.1) enabled weather station using ESP32 microcontroller. It reads data from DHT11 temperature and humidity sensor and sends the data to a BLE client. The data can be seen in nRF Connect app in the client-side. On top of that, WiFi credentials could be seen in the respective characterstics.

Components used

- ESP32 Microcontroller
- DHT11 Temperature and Humidity sensor
- Mobile with nRF Connect app

Libraries Used

1. **BLEDevices.h :** Initializes the BLE functionality and used as the main entry point for setting up a BLE environment.
2. **BLEServer.h :** Provides BLEServer class, which represents the server instance.
3. **BLEUtils.h :** Includes various BLE functions/operations such as data conversions, address formatting, and other helper methods.
4. **BLE2902.h :** Provides descriptor class, which represents the client instance.
5. **DFRobot_DHT11.h :** Offers interface for DHT11 sensor.
6. **WiFi.h :** Grants basic WiFi functions such as Connecting to WiFi Networks, Managing IP address, Web Server and Client Functionalities.

Project Setup

1. Hardware Setup

1. **Connect the DHT11 Sensor to the ESP32**:
   - **VCC**: Connect to 3.3V on ESP32.
   - **GND**: Connect to ground.
   - **Data**: Connect to GPIO2 of the ESP32.
2. **ESP32 Setup**:

    - Flash the ESP32 with firmware that includes BLE and WiFi connectivity.
    - Set up the ESP32 for BLE advertising, so it can be detected by the nRF Connect app.

2. Software Setup

1. **Programming the ESP32**:
    - Install the **ESP-IDF** or **Arduino IDE**.
    - Import necessary libraries for DHT11 sensor and BLE functionality.
2. **BLE Configuration**:
    - Configure the ESP32 as a BLE server.
    - Set up characteristics for temperature and humidity data.

3. Viewing Data on the Client

1. Open the **nRF Connect app** on a BLE-enabled smartphone or tablet.
2. Connect to the ESP32 BLE weather station.
3. View live temperature and humidity readings transmitted from the ESP32.

BLE Service and Characteristics

In BLE, characteristics are attributes that represent specific type of data such as Blood Pressure Measurement, Force, and Heart Rate. The characteristics ID can be seen in assigned numbers section in bluetooth SIG website. The characteristics for this project are mentioned below. Each characteristic has a unique **UUID** that allows other devices to recognize its usecase.

- **Service UUID :** 00000002-0000-0000-FDFD-FDFDFDFDFDFD
    - **Temperature Characteristic:**
        - **UUID :** 0x2A6E
        - **Properties :** NOTIFY, READ
    - **Humidity Characteristic:**
        - **UUID :** 0x2A6F
        - **Properties:** NOTIFY, READ
    - **WiFi Credentials Characteristic:**
        - **UUID :** 0x2AF7

- **Properties :** NOTIFY, READ, WRITE

Code

```
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>
#include <DFRobot_DHT11.h>
#include <WiFi.h>

DFRobot_DHT11 DHT;
#define DHT11_PIN 2

uint16_t hum;
int temp;
bool deviceConnected = false;
bool oldDeviceConnected = false;
String pwd = "";
String ssid = "";
BLEServer *pServer;
BLECharacteristic *tempChar, *humChar, *wifiChar;

class MyServerCallbacks: public BLEServerCallbacks {
  void onConnect(BLEServer* pServer) {
    deviceConnected = true;
  }

  void onDisconnect(BLEServer* pServer) {
    deviceConnected = false;
  }

};

class charCallback: public BLECharacteristicCallbacks {
  void onWrite(BLECharacteristic *pCharacteristic) {
    String value = pCharacteristic->getValue();
    if (value.length() > 0) {
      if(ssid == "" && pwd == ""){
        Serial.print("SSID: ");
        ssid = value;
        Serial.println(ssid);
      }
      else if(pwd==""){
        pwd = value;
        Serial.print("Password: ");
        Serial.println(pwd);
        WiFi.begin(ssid, pwd);
        while (WiFi.status() != WL_CONNECTED) {
            delay(500);
            Serial.print(".");
        }
        Serial.println();
        Serial.print("Wifi Connected Sucessfully!");
        wifiChar->setValue("WiFi Successfully connected");
        wifiChar->notify();
        pwd="";
        ssid="";
      }
    }
  }
};

void setup() {
  Serial.begin(115200);
  BLEDevice::init("Weather Station");
  // Create the BLE Server
  pServer = BLEDevice::createServer();
  pServer->setCallbacks(new MyServerCallbacks());
  BLEService *DHTService = pServer->createService("00000002-0000-0000-FDFD-
FDFDFDFDFDFD");
  //creating temperature characteristic
  tempChar = DHTService->createCharacteristic(BLEUUID((uint16_t)0x2A6E),
BLECharacteristic::PROPERTY_NOTIFY | BLECharacteristic::PROPERTY_READ);
  tempChar->addDescriptor(new BLE2902());
  //creating humidity characteristic
  humChar = DHTService->createCharacteristic(BLEUUID((uint16_t)0x2A6F),
BLECharacteristic::PROPERTY_NOTIFY | BLECharacteristic::PROPERTY_READ);
  humChar->addDescriptor(new BLE2902());
  //creating characteristic for transfering WiFi credentials
  wifiChar = DHTService->createCharacteristic(BLEUUID((uint16_t)0x2AF7),
BLECharacteristic::PROPERTY_NOTIFY | BLECharacteristic::PROPERTY_READ |
BLECharacteristic::PROPERTY_WRITE);
  wifiChar->setCallbacks(new charCallback());
  wifiChar->addDescriptor(new BLE2902());
  //Starting the service
  DHTService->start();
  BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();
  pAdvertising->addServiceUUID("00000002-0000-0000-FDFD-FDFDFDFDFDFD");
  pServer->getAdvertising()->start();
  Serial.println("Waiting a client connection to notify...");
  if(deviceConnected){
    Serial.println("Device Connected");
  }
}

void loop() {
  DHT.read(DHT11_PIN);
  temp = DHT.temperature*100;
  hum = DHT.humidity*100;
  if(deviceConnected){
    humChar->setValue(hum);
    humChar->notify();
    tempChar->setValue(temp);
    tempChar->notify();
    delay(100);
  }
  if (!deviceConnected && oldDeviceConnected) {
    delay(500);                   // give the bluetooth stack the chance to get
things ready
    pServer->startAdvertising();  // restart advertising
    Serial.println("start advertising");
    oldDeviceConnected = deviceConnected;
  }
  if (deviceConnected && !oldDeviceConnected) {
      oldDeviceConnected = deviceConnected;
  }
  delay(1000);
}
```

Code Explanation

1. **Global Variables**
    - **temp** and **hum :** temperature and humidity readings from the DHT11 sensor.
    - **deviceConnected** and **oldDeviceConnected :** BLE connection status.
    - **ssid** and **pwd :** Contains WiFi credentials sent from the BLE client.
2. **BLE Callbacks**
    - **MyServerCallbacks:**

        - Updates **deviceConnected** when a BLE client connects or disconnects.
    - **charCallback:**
        - Try to connect WiFi connection with the received credentials and notifies the client if the connection is successful.
3. **setup() function**
    - Initializes serial communication and BLE device.
    - Creates the BLE server and service for the weather station.
    - Create instances for temperature, humidity, and WiFi credentials characteristics.
    - Advertising BLE beacon frames.
4. **loop() function**
    - Reads temperature and humidity data from the DHT11 sensor.
    - Sends temperature and humidity readings through BLE, if the client is connected with the server.
    - Checks BLE connection status to restart advertising if the device disconnects.