ECE LABORATORY

DREXEL UNIVERSITY

To: **Dr. Peters**

From: **Ehi Simon**

Re: **ECE 304 Lab 2 - Bluetooth and Bluetooth Low Energy**

PURPOSE:

The purpose of this week's lab is to perform rudimentary remote monitoring and control using a smart phone and/or laptop through the use of the Bluetooth and Bluetooth Low Energy (BLE) protocols. Using Bluetooth and BLE will make it easy to perform on-demand monitoring and control.

Discussion:

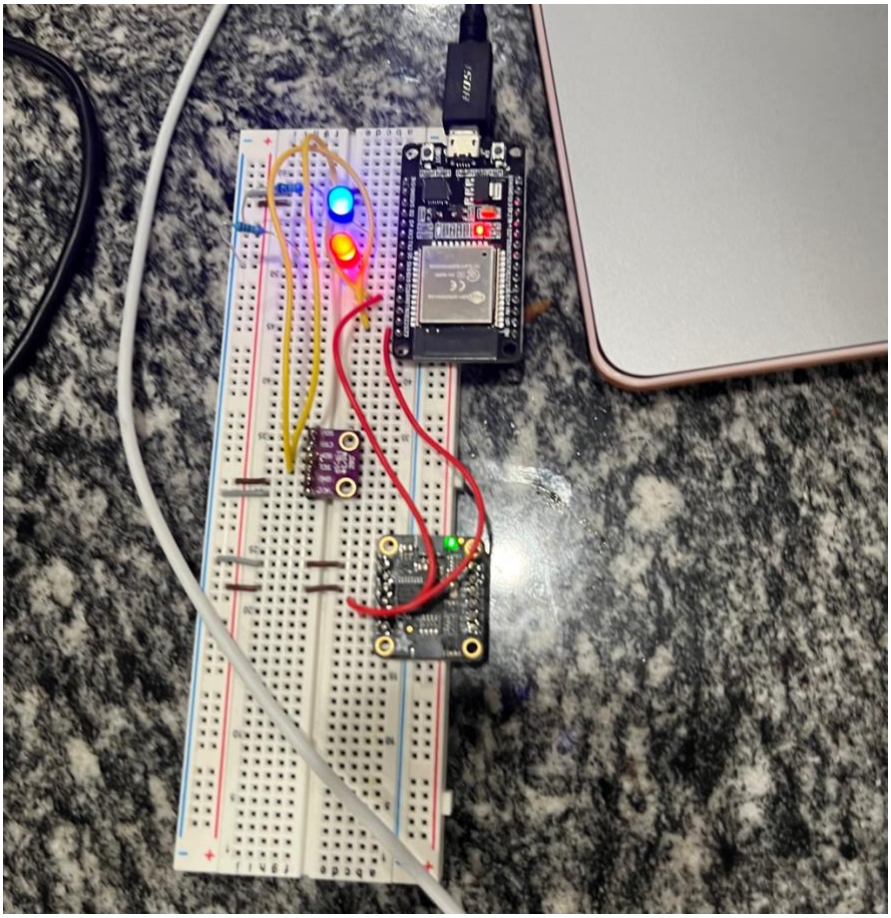


Fig. 1. Circuit Connection for Project 2

The circuit for the lab was built like the one above. It consists of 2 330Ω resistors, a red LED, a blue LED, a BME280 Environmental Sensor, an Adafruit BNO085 IMU, and an ESP32S microcontroller. Just like last week, an integer (between 0 and 8191) will be sent to the ESP32 for decoding. The breakdown for the number is based on the bit makeup:

- Bit 0(LSB): Take and display temperature measurement (1), or not (0).
- Bit 1: Take and display humidity measurement (1), or not (0)
- Bit 2: Take and display pressure measurement (1), or not (0)
- Bit 3: Take and display altitude measurement (1), or not (0)
- Bit 4: Turn blue LED ON (1) or OFF(0)
- Bits 5-12(MSB): Set the intensity of the red LED.

For this project I created two separate ESP32 programs named BluetoothMain.cpp and BLEMain.cpp.

BluetoothMain.cpp

In my main.cpp file, I initialized the multiple libraries that were needed for the sensors to work and provide readings. The figure below shows the libraries I initialized. There is an additional library from the first project and that is the BluetoothSerial header file. This file is what is going to enable me to connect the ESP32 to Bluetooth and control it from a Bluetooth serial terminal.

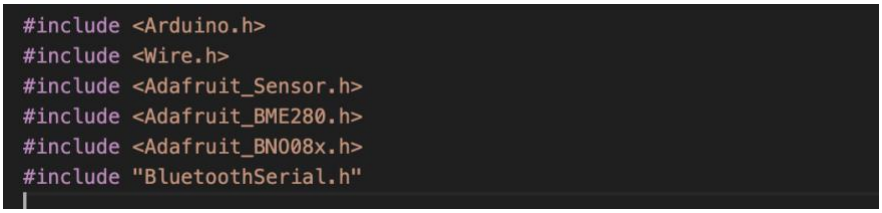


Fig. 2. Figure Showing Initialization of Libraries

The main function consisted mostly of the setup and loop functions. Before the setup functions, I did some essential things such as defining the pins for the LEDs, defining the PWM parameters, and defining a reference pressure for sea level. Also, I created both a BME object and a BluetoothSerial object. This required the BluetoothSerial header file. After this, I created a structure named euler_t that defined the yaw, pitch, and roll and set them to be floating values. Lastly, I reset the BNO085 chip. This is shown in the figure below.

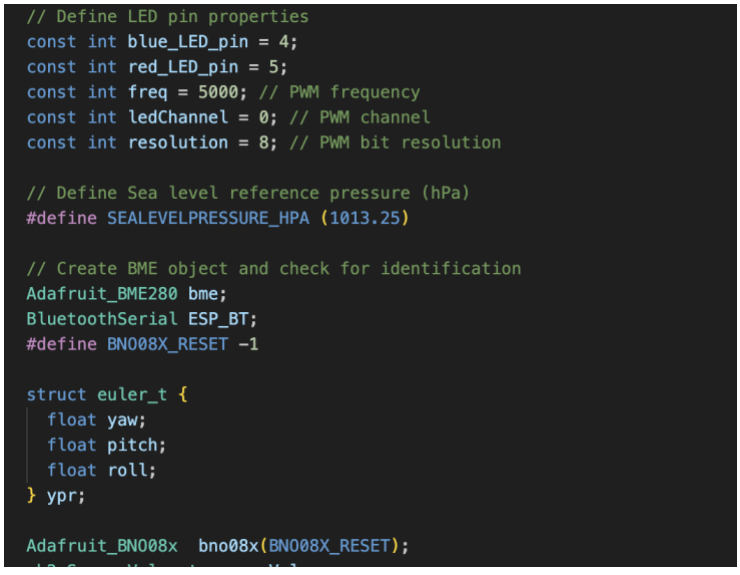


Fig. 3. Figure Showing Definition of Variables and Creation of Objects

```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  ESP_BT.begin("ESP32_Peters");
  pinMode(blue_LED_pin, OUTPUT);
  pinMode(red_LED_pin, OUTPUT);
  ledcSetup(ledChannel, freq, resolution);
  ledcAttachPin(red_LED_pin, ledChannel);

  bool status;
  status = bme.begin(0x76);
  if (!status) {
    Serial.println("Could not find a valid BME280 sensor, check wiring!");
  while (1);
  }
  Serial.println("Enter A Number between 0 and 8191: ");

    while (!Serial) delay(10);     // will pause Zero, Leonardo, etc until serial console opens

  Serial.println("Adafruit BNO08x test!");

  // Try to initialize!
  if (!bno08x.begin_I2C()) {
    Serial.println("Failed to find BNO08x chip");
    while (1) { delay(10); }
  }
  Serial.println("BNO08x Found!");


  setReports(reportType, reportIntervalUs);

  Serial.println("Reading events");
  delay(100);
}
```

Fig. 4. Figure Showing Setup Function

The figure above is concerned with the setup of the ESP32. The “serial.begin” line starts serial communications with a 115200-bps baud rate. The ESP_BT.begin(“ESP32_Peters”) line starts the Bluetooth communication and names the device ESP32_Peters. The next two lines set the pins for the LEDs for OUTPUT. The two lines below those are specially for the ESP32. They are used to set up the LED behavior for the red LED. The next few lines start the BME280, and check if the BME280 is found. They also start the BNO085 chip, and check if it is found. After this in the setup function, we also print out a prompt for the user to enter an integer between 0 and 8191.

The figure below shows the first main section of the loop function. It starts with the line “int command = Serial.parseInt()” reading a sequence of characters from the serial monitor, converting them into an integer value, and assigning it to the variable command. Then, it prompts the user to enter a number between 0 and 8191. These commands are read using the bitRead function and assigned to variables based on their bits. Based on the command entered, the function will also

check if the blue LED is on or off and print the output to the serial terminal, as well as the Bluetooth serial terminal. It will also print the value of the red LED to both terminals. After this, there are if functions that will read the temperature, pressure, humidity, and altitude from the BME sensor and print the value to both terminals.

```
void loop() {
  // Read command from serial monitor
  if (ESP_BT.available() >0){
    int command = ESP_BT.parseInt();
    Serial.flush();
    Serial.print("Received: ");
    Serial.println(command);
    if ((command >= 0) & (command <= 8191)){
      int temp_bit = bitRead(command,0);
      int humid_bit = bitRead(command,1);
      int press_bit = bitRead(command,2);
      int elev_bit = bitRead(command,3);
      int blue_LED_bit = bitRead(command,4);
      int red_LED_value = command >> 5;

      digitalWrite(blue_LED_pin,blue_LED_bit);
      if (blue_LED_bit){
        Serial.println("Blue LED: ON");
        ESP_BT.println("Blue LED: ON");
      }
      else{
        Serial.println("Blue LED: OFF");
        ESP_BT.println("Blue LED: OFF");
      }

      ledcWrite(ledChannel, red_LED_value);
      String red_str = "Red LED Value: " + String(red_LED_value);
      Serial.println(red_str);
      ESP_BT.println(red_str);

      if (temp_bit){
        float temp_val = bme.readTemperature();
        String temp_str = "Temperature: " + String(temp_val,2) + " C";
        Serial.println(temp_str);
        ESP_BT.println(temp_str);
      }
```

Fig. 5. Section of the Loop Function

A table mapping the command bits to their descriptions can be found below:

Bit	Description
0 (LSB)	Temperature: (0/1 for no measurement/measurement)
1	Humidity: (0/1 for no measurement/measurement)
2	Pressure: (0/1 for no measurement/measurement)
3	Altitude: (0/1 for no measurement/measurement)
4	Blue LED: (0/1 for OFF/ON)
5-12 (MSB)	Blue LED intensity (Leftmost 8 bits for 0-255)

Table 1. Table Showing Command Bits and Their Descriptions

The last main section of the loop function deals with obtaining the yaw, pitch and roll from the BNO08X chip and printing them to the serial monitor. This can be found in the figure below.

```
    //Clear the buffer to remove line feeds and carriage returns
    while(ESP_BT.available()>0){
      int temp=ESP_BT.parseInt();
    }
    if (bno08x.wasReset()) {
    Serial.print("sensor was reset ");
    setReports(reportType, reportIntervalUs);
  }

    if (bno08x.getSensorEvent(&sensorValue)) {
      quaternionToEulerRV(&sensorValue.un.arvrStabilizedRV, &ypr, true);

      static long last = 0;
      long now = micros();
      Serial.print(now - last);             Serial.print("\t");
      last = now;
      Serial.print(sensorValue.status);     Serial.print("\t");  // This is accuracy in the range of 0 to 3
      Serial.print(ypr.yaw);                Serial.print("\t");
      Serial.print(ypr.pitch);              Serial.print("\t");
      Serial.println(ypr.roll);
    }
  }
  delay(20);
}
```

Fig. 6. Section of the Loop Function

The final result was output in both serial terminal and Bluetooth serial terminal that can be found below.

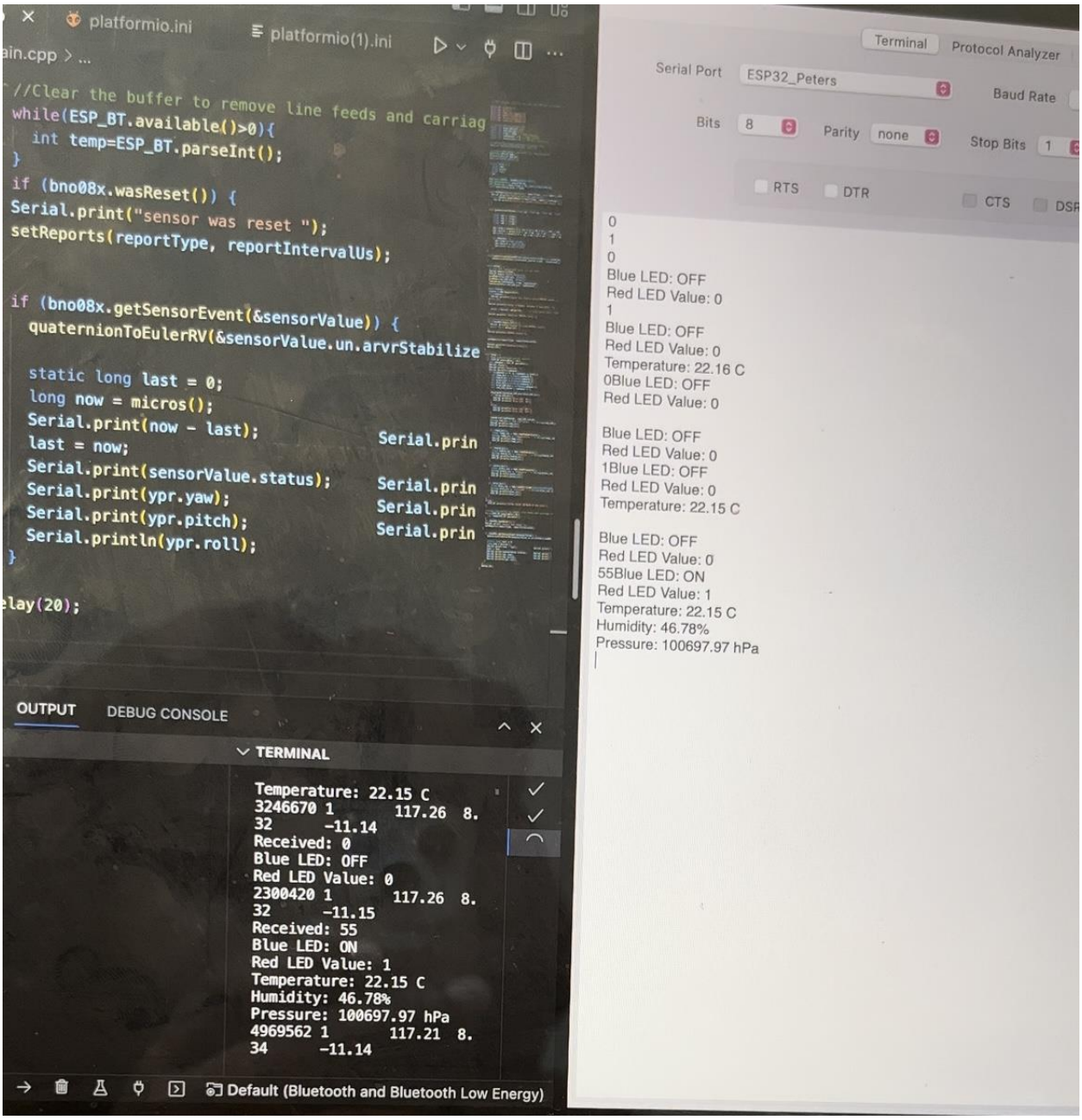
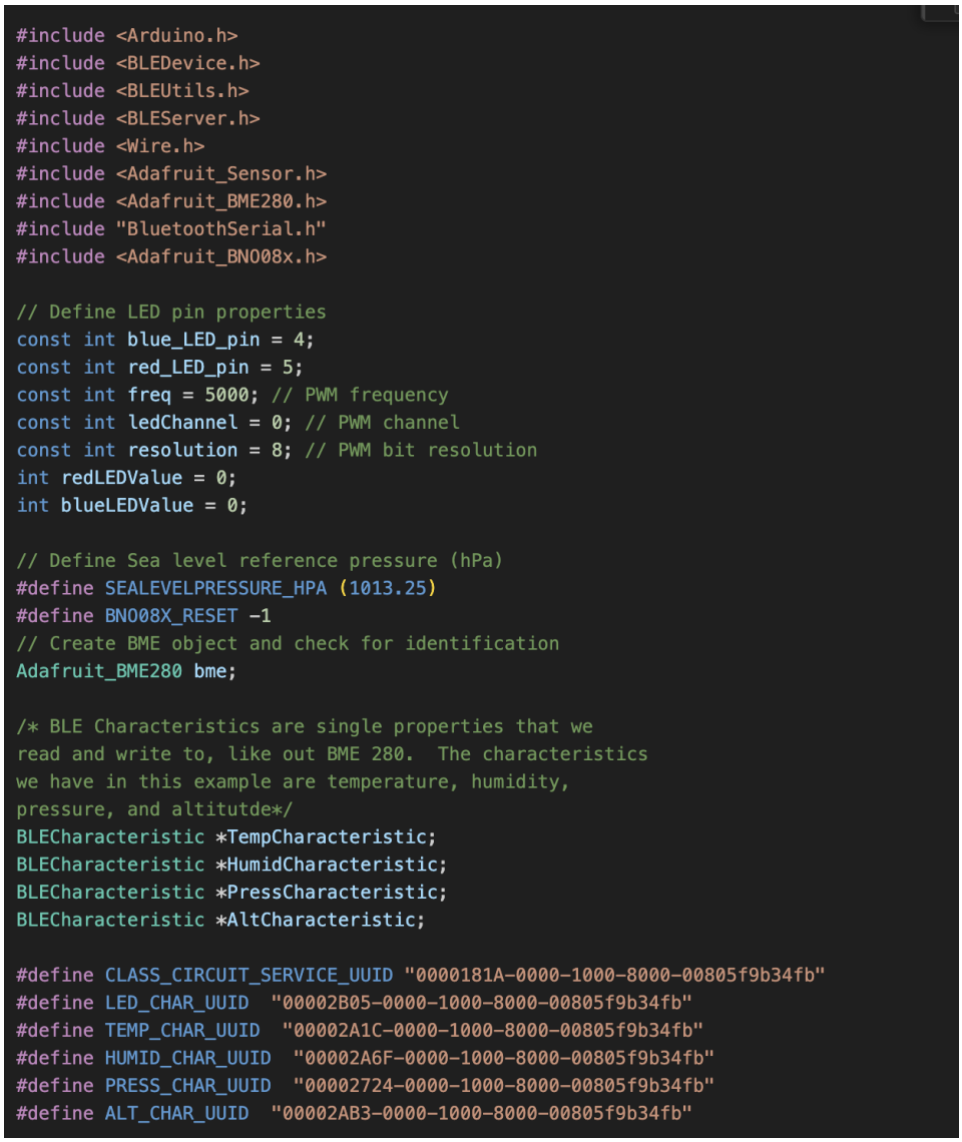


Fig. 7. Output on Both Serial Terminals

BlEMain.cpp

In my main.cpp file, I initialized the multiple libraries that were needed for the sensors to work and provide readings. The figure below shows the libraries I initialized. There are additional libraries other than those form the previous program and those are the BLEDevice, BLEServer, and BLEUtils header files. This file is what is going to enable me to connect the ESP32 to Bluetooth and control it from a Bluetooth serial terminal. The LEDs are defined, the sea level reference pressure is defined and the BNO08X chip is reset. Also, BLE characteristics are created for temperature, altitude, humidity, and pressure to be read from the BME, which has its object created. UUIDs are then created for these characteristics. This can all be seen in the figure below.



```
#include <Arduino.h>
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEServer.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>
#include "BluetoothSerial.h"
#include <Adafruit_BNO08x.h>

// Define LED pin properties
const int blue_LED_pin = 4;
const int red_LED_pin = 5;
const int freq = 5000; // PWM frequency
const int ledChannel = 0; // PWM channel
const int resolution = 8; // PWM bit resolution
int redLEDValue = 0;
int blueLEDValue = 0;

// Define Sea level reference pressure (hPa)
#define SEALEVELPRESSURE_HPA (1013.25)
#define BNO08X_RESET -1
// Create BME object and check for identification
Adafruit_BME280 bme;

/* BLE Characteristics are single properties that we
read and write to, like out BME 280.  The characteristics
we have in this example are temperature, humidity,
pressure, and altitutde*/
BLECharacteristic *TempCharacteristic;
BLECharacteristic *HumidCharacteristic;
BLECharacteristic *PressCharacteristic;
BLECharacteristic *AltCharacteristic;

#define CLASS_CIRCUIT_SERVICE_UUID "0000181A-0000-1000-8000-00805f9b34fb"
#define LED_CHAR_UUID  "00002B05-0000-1000-8000-00805f9b34fb"
#define TEMP_CHAR_UUID  "00002A1C-0000-1000-8000-00805f9b34fb"
#define HUMID_CHAR_UUID  "00002A6F-0000-1000-8000-00805f9b34fb"
#define PRESS_CHAR_UUID  "00002724-0000-1000-8000-00805f9b34fb"
#define ALT_CHAR_UUID  "00002AB3-0000-1000-8000-00805f9b34fb"
```

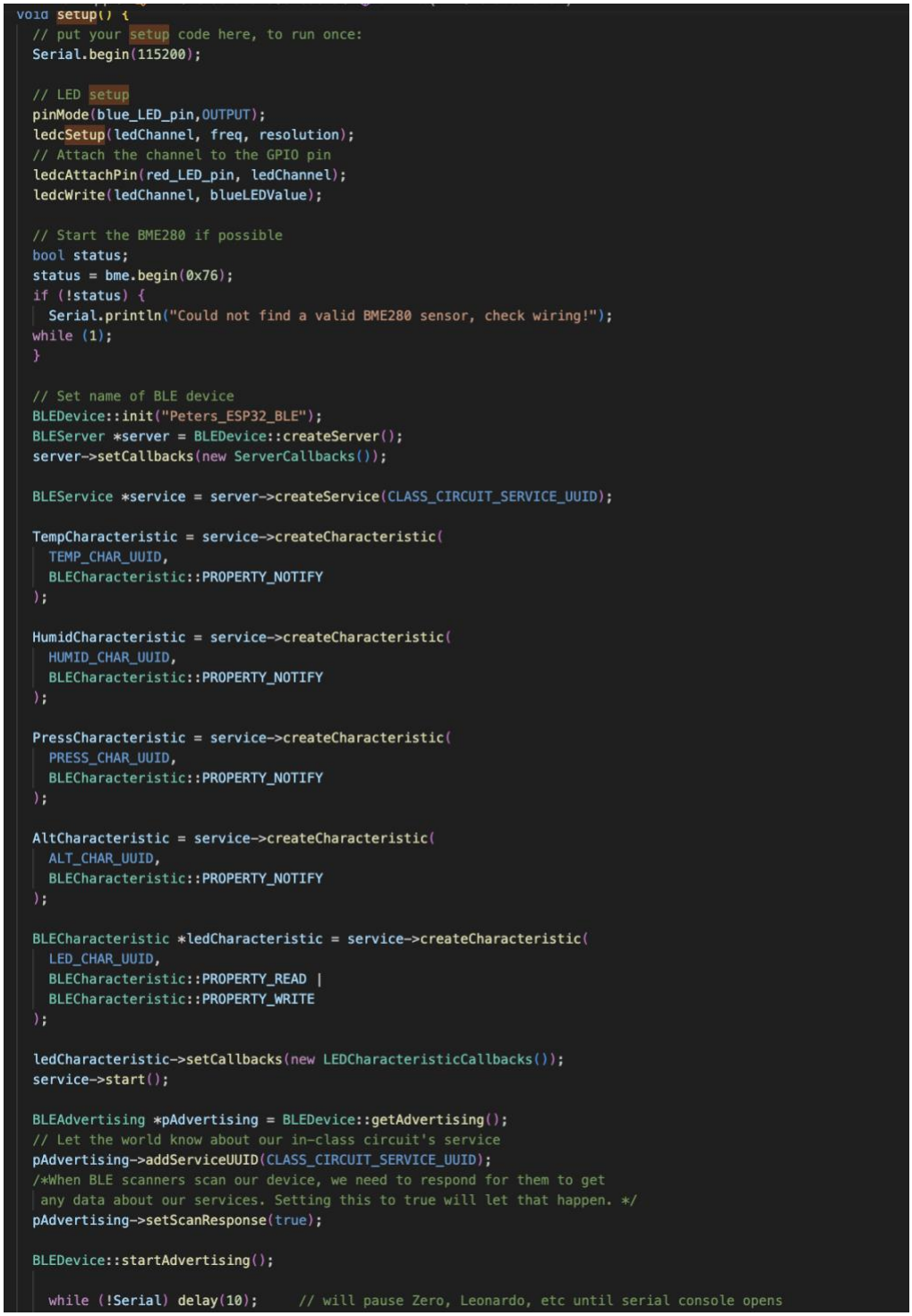
Fig. 8. Figure Showing Initialization of Libraries and Definition of Variables

The BLEMain.cpp file has multiple custom classes that were created for it to work. These can be found below:

```
class ServerCallbacks: public BLEServerCallbacks {
    // code runs automatically when a device connects
    void onConnect(BLEServer* pServer) {
      Serial.print("New Device connected with connection ID: ");
      Serial.println(pServer->getConnId());
    };

    // code runs automatically when a device disconnects
    void onDisconnect(BLEServer* pServer) {
      Serial.print("Device disconnected with connection ID: ");
      Serial.println(pServer->getConnId());
    }
};

class LEDCharacteristicCallbacks: public BLECharacteristicCallbacks{
  // called when a connected device writes to this characteristic
  void onWrite(BLECharacteristic *characteristic) {


    std::string data = characteristic->getValue();
    Serial.print("Received Value: ");
    Serial.println(data.c_str());

    if (data.length() == 0) {
      // an empty string was sent, there's nothing to do
      return; // returning from a callback "ends" the write operation
    }
    String valueStr = data.c_str();
    int value = valueStr.toInt();

    // 0 = blue is off, red is value 0
    // 511 = blue is on, red is value 255
    if (value >= 0 && value <= 511) {
      // grab the least significant bit from the data for the red LED
      blueLEDValue = value & 0b1;
      Serial.print("Blue LED Value: ");
      Serial.println(blueLEDValue);
      // write to red LED
      digitalWrite(blue_LED_pin, blueLEDValue);

      // right shift the value by one and use that for the blue LED
      redLEDValue = value >> 1;
      Serial.print("Red LED Value: ");
      Serial.println(redLEDValue);
      // write to blue LED
      ledcWrite(ledChannel, redLEDValue);
    }
    else{
      Serial.println("VALUE NOT IN RANGE");
    }
  }

  // called when a connected device reads from this characteristic
  void onRead(BLECharacteristic *characteristic) {
    // this is the string the connected device will read
    String ledInfo = "BLUE: " + String(blueLEDValue) + ", RED: " + String(redLEDValue) + "\n";
    Serial.print(ledInfo);
    characteristic->setValue(ledInfo.c_str());
  }
};
```

Fig. 9. Figure Showing Custom Classes and Functions to Interact with BLE Devices

The setup and loop functions are very similar to that of the previous program except that they now include these characteristics and classes that were created to interact with BLE devices. They can be found in the figures below.



```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);

  // LED setup
  pinMode(blue_LED_pin,OUTPUT);
  ledcSetup(ledChannel, freq, resolution);
  // Attach the channel to the GPIO pin
  ledcAttachPin(red_LED_pin, ledChannel);
  ledcWrite(ledChannel, blueLEDValue);

  // Start the BME280 if possible
  bool status;
  status = bme.begin(0x76);
  if (!status) {
    Serial.println("Could not find a valid BME280 sensor, check wiring!");
  while (1);
  }

  // Set name of BLE device
  BLEDevice::init("Peters_ESP32_BLE");
  BLEServer *server = BLEDevice::createServer();
  server->setCallbacks(new ServerCallbacks());

  BLEService *service = server->createService(CLASS_CIRCUIT_SERVICE_UUID);

  TempCharacteristic = service->createCharacteristic(
    TEMP_CHAR_UUID,
    BLECharacteristic::PROPERTY_NOTIFY
  );

  HumidCharacteristic = service->createCharacteristic(
    HUMID_CHAR_UUID,
    BLECharacteristic::PROPERTY_NOTIFY
  );

  PressCharacteristic = service->createCharacteristic(
    PRESS_CHAR_UUID,
    BLECharacteristic::PROPERTY_NOTIFY
  );

  AltCharacteristic = service->createCharacteristic(
    ALT_CHAR_UUID,
    BLECharacteristic::PROPERTY_NOTIFY
  );

  BLECharacteristic *ledCharacteristic = service->createCharacteristic(
    LED_CHAR_UUID,
    BLECharacteristic::PROPERTY_READ |
    BLECharacteristic::PROPERTY_WRITE
  );

  ledCharacteristic->setCallbacks(new LEDCharacteristicCallbacks());
  service->start();

  BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();
  // Let the world know about our in-class circuit's service
  pAdvertising->addServiceUUID(CLASS_CIRCUIT_SERVICE_UUID);
  /*When BLE scanners scan our device, we need to respond for them to get
   any data about our services. Setting this to true will let that happen. */
  pAdvertising->setScanResponse(true);

  BLEDevice::startAdvertising();

    while (!Serial) delay(10);     // will pause Zero, Leonardo, etc until serial console opens
```

Fig. 9. Figure Showing Setup Function

```
void loop() {
  // Read BME280 values
  float temp = bme.readTemperature();
  float humid = bme.readHumidity();
  float press = bme.readPressure()/1000.0;
  float alt = bme.readAltitude(SEALEVELPRESSURE_HPA);

  // Create strings that will appear on the connected device
  String tempString = String(temp, 2) + " C";
  String humidString = String(humid,2) + "%";
  String pressString = String(press,2) + " kPa";
  String altString = String (alt,2) + " m";

  // we need to set the characteristic's value before we notify
  TempCharacteristic->setValue(tempString.c_str());
  HumidCharacteristic->setValue(humidString.c_str());
  PressCharacteristic->setValue(pressString.c_str());
  AltCharacteristic->setValue(altString.c_str());

  // send the new value to any connected device that is listening.
  TempCharacteristic->notify();
  HumidCharacteristic->notify();
  PressCharacteristic->notify();
  AltCharacteristic->notify();

  // space out our notifications so we don't spam the connected devices
  delay(1000);


  if (bno08x.wasReset()) {
    Serial.print("sensor was reset ");
    setReports(reportType, reportIntervalUs);
  }

  if (bno08x.getSensorEvent(&sensorValue)) {
    quaternionToEulerRV(&sensorValue.un.arvrStabilizedRV, &ypr, true);

    static long last = 0;
    long now = micros();
    Serial.print(now - last);             Serial.print("\t");
    last = now;
    Serial.print(sensorValue.status);     Serial.print("\t");  // This is accuracy in the range of 0 to 3
    Serial.print(ypr.yaw);                Serial.print("\t");
    Serial.print(ypr.pitch);              Serial.print("\t");
    Serial.println(ypr.roll);
  }


}
```

Fig. 10. Figure Showing Loop Function

The final result came as notifications sent to the BLE device as well as values printed on the serial monitor. The output can be found below.

```
Received Value: 78
Blue LED Value: 0
Red LED Value: 39
BLUE: 0, RED: 39
1012360 0       136.29  13.04   -16.87
1027604 0       136.30  13.00   -16.80
1011701 0       136.29  13.00   -16.77
1027968 0       136.28  13.00   -16.85
1011589 0       136.26  12.96   -16.89
1027905 0       136.24  12.93   -16.91
1011551 0       136.23  12.96   -16.95
1027472 0       136.24  12.98   -16.90
1012594 0       136.24  12.97   -16.89
1028269 0       136.24  12.98   -16.88
1012954 0       136.24  12.98   -16.88
1029490 0       136.21  12.98   -16.87
1011216 0       136.27  12.71   -16.97
1027325 0       136.18  12.62   -17.08
```

Fig. 11. Output in Serial Monitor Showing LED Values and Yaw, Pitch, and Roll

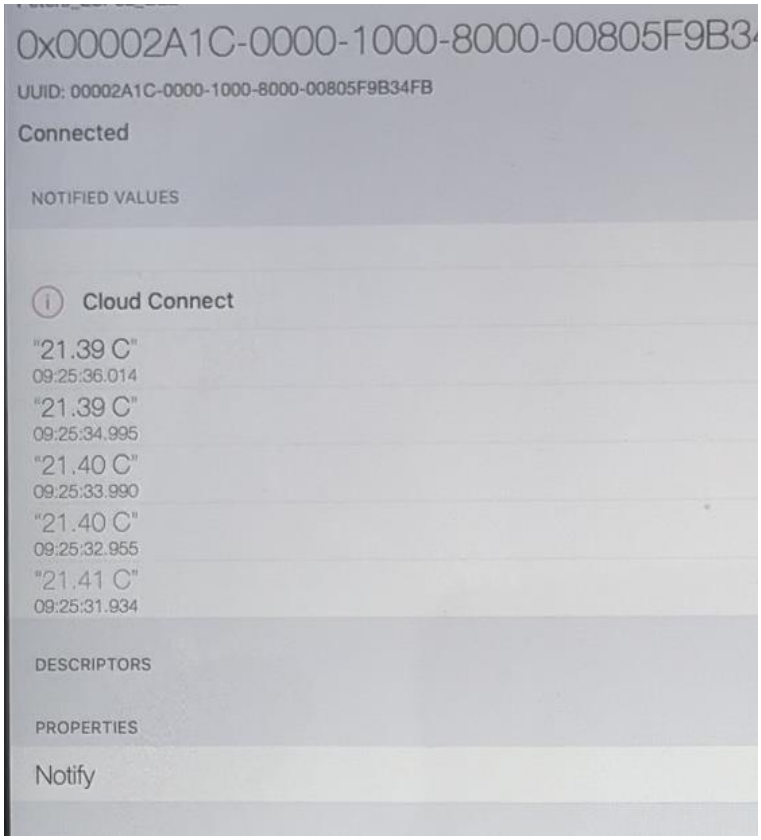


Fig. 12. Figure Showing Output in BLE Terminal for Temperature

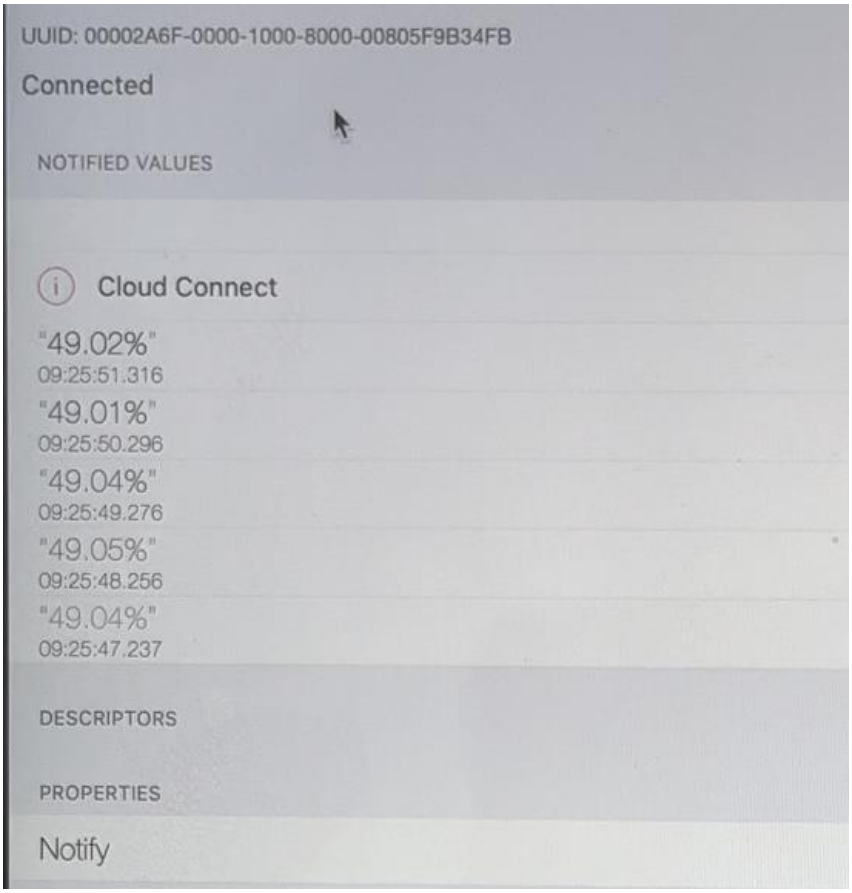


Fig. 13. Figure Showing Output in BLE Terminal for Humidity

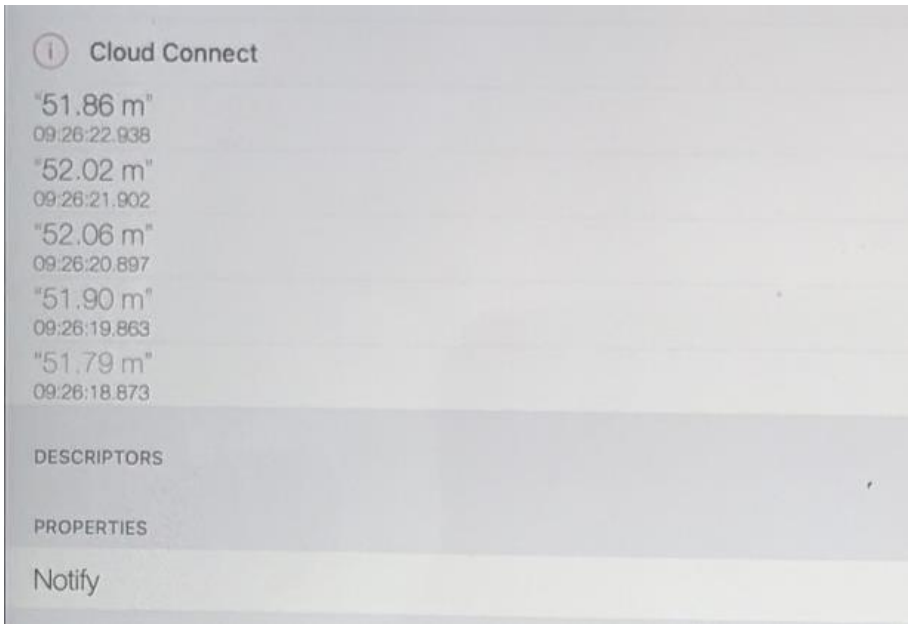


Fig. 14. Figure Showing Output in BLE Terminal for Altitude

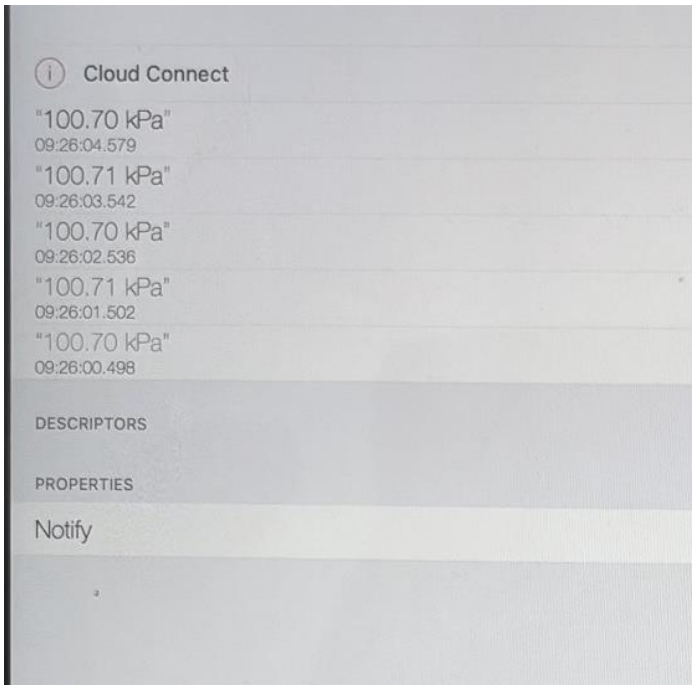


Fig. 15. Figure Showing Output in BLE Terminal for Pressure

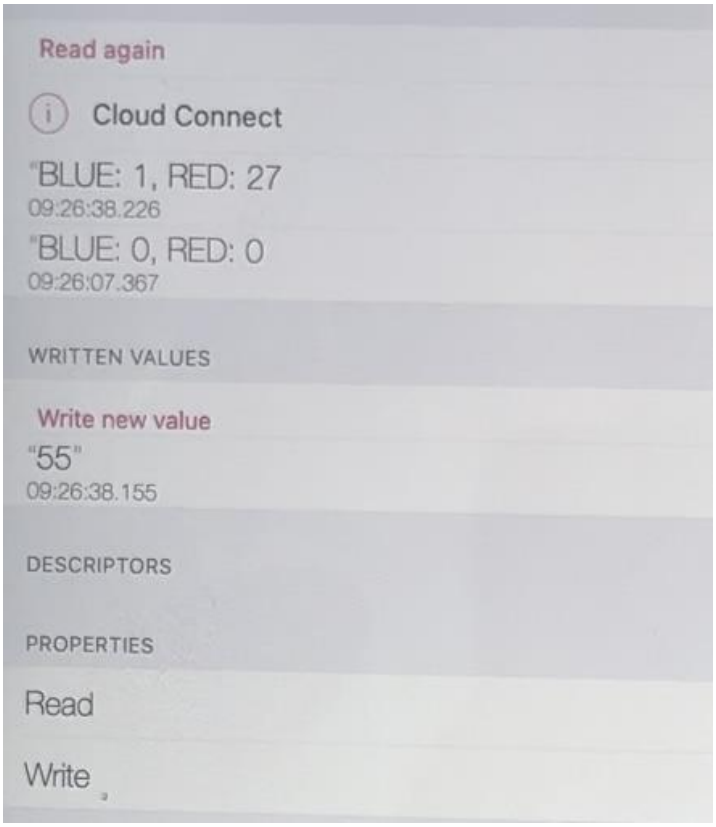


Fig. 16. Figure Showing Output in BLE Terminal for LED Values

Conclusion

In this experiment, I learned about the integration of an ESP32 microcontroller with various sensors and Bluetooth connectivity. By initializing libraries and creating objects, I was able to read data from sensors, control LEDs, and communicate via Bluetooth.