BLE Communication between ESP32 and Python

1. Introduction

Bluetooth Low Energy (BLE) is a low-energy version of Bluetooth that sends small packets of data at regular intervals. This report explores the implementation of BLE communication between an ESP32 microcontroller and Python code running on a computer.

2. Equipment

- ESP32 module (on-board Bluetooth+Wifi)
- A computer with Python installed
- USB cable for ESP32-computer connection

3. ESP32 BLE Implementation

3.1 Required Libraries:

```
#include <BLEDevice.h>

#include <BLEUtils.h>

#include <BLEServer.h>
```

3.2 UUID Definitions:

UUID (Universally Unique Identifier) is a 128-bit number used to uniquely identify information in BLE communication. In our implementation, we define two UUIDs:

```
#define SERVICE_UUID        "4fafc201-1fb5-459e-8fcc-c5c9c331914b"
#define CHARACTERISTIC_UUID "beb5483e-36e1-4688-b7f5-ea07361b26a8"
```

- SERVICE_UUID: Identifies the BLE service our ESP32 is providing.
- CHARACTERISTIC_UUID: Identifies a specific characteristic within that service, which can be read from or written to.

UUIDs serve several important purposes in BLE:

1. Uniquely identify services and characteristics
2. Ensure interoperability between devices

3. Allow devices to discover specific services and characteristics

3.3 BLE Server Setup:

```
void setup() {
  Serial.begin(115200);
  Serial.println("1- Download and install an BLE Terminal FREE");
  Serial.println("2- Scan for BLE devices in the app");
  Serial.println("3- Connect to ESP32BLE");
  Serial.println("4- Go to CUSTOM CHARACTERISTIC in CUSTOM SERVICE and write
something");
  BLEDevice::init("ESP32BLE");
  BLEServer *pServer = BLEDevice::createServer();
  BLEService *pService = pServer->createService(SERVICE_UUID);
  BLECharacteristic *pCharacteristic = pService->createCharacteristic(
                                         CHARACTERISTIC_UUID,
                                         BLECharacteristic::PROPERTY_READ |
                                         BLECharacteristic::PROPERTY_WRITE
                                       );
  pCharacteristic->setCallbacks(new MyCallbacks());
  pCharacteristic->setValue("Hello World");
  pService->start();
  BLEAdvertising *pAdvertising = pServer->getAdvertising();
  pAdvertising->start();
  Serial.print("Server address:");
  Serial.println(BLEDevice::getAddress().toString().c_str());
}
```

3.4 Handling BLE Write Operations:

```
class MyCallbacks: public BLECharacteristicCallbacks {
    void onWrite(BLECharacteristic *pCharacteristic) {
      std::string value = pCharacteristic->getValue();
      if (value.length() > 0) {
        Serial.println("*********");
        Serial.print("New value: ");
        for (int i = 0; i < value.length(); i++)
          Serial.print(value[i]);
        Serial.println();
        Serial.println("*********");
      }
    }
};
```

3.5 Main Loop:

```
void loop() {
  // put your main code here, to run repeatedly:
  delay(2000);
}
```

4. Python BLE Implementation

4.1 Required Library:

To manage Bluetooth Low Energy communication from your PC, install the Bleak package:

```
python -m pip install bleak
```

4.2 Scanning for BLE Devices:

```
import asyncio
from bleak import BleakScanner

async def main():
    target_name = "ESP32BLE"
    target_address = None
    devices = await BleakScanner.discover()
    for d in devices:
        print(d)
        if target_name == d.name:
            target_address = d.address
            print(f"found target {target_name} bluetooth device with address {target_address}")
            break

asyncio.run(main())
```

4.3 Connecting and Communicating with ESP32:

```
import asyncio
from bleak import BleakScanner
from bleak import BleakClient

async def main():
    target_name = "ESP32BLE"
    target_address = None
    SERVICE_UUID = "4fafc201-1fb5-459e-8fcc-c5c9c331914b"
    CHARACTERISTIC_UUID = "beb5483e-36e1-4688-b7f5-ea07361b26a8"

    devices = await BleakScanner.discover()
    for d in devices:
        print(d)
```

```
        if target_name == d.name:
            target_address = d.address
            print(f"found target {target_name} bluetooth device with address {target_address}")
            break

    if target_address is not None:
        async with BleakClient(target_address) as client:
            print(f"Connected: {client.is_connected}")


              while True:
               text = input()
               if text == "quit":
                  break
               await client.write_gatt_char(CHARACTERISTIC_UUID, bytes(text, 'UTF-8'),
response=True)

               try:
                  data = await client.read_gatt_char(CHARACTERISTIC_UUID)
                  data = data.decode('utf-8') # convert byte to str
                  print(f"data: {data}")
               except Exception:
                  pass

    else:
        print("could not find target bluetooth device nearby")

asyncio.run(main())
```