SIoT ➔HW 1 ➔BLE
Parisa Toumari
99101857

The code:
I start by initiating the required stages for BLE protocol. First, we have to initiate the memory, then the controller, and then the NimBLE definitions, and then we specify the configurations; like the server name, gap services, gatt services, and the configuration of the connections and layers.
In the 6th stage, where we run the thread, we actually run the “host_task” function which is an infinite task that is running and it ceases to exist when the NimBLE port stops working.
The advertise function defines the BLE connection. This function has two major parts. First is the “device name” definition, and second is the “device connectivity” definition; where we define whether our device will be connectable or non-connectable and other connection possibilities.
There are two handler events in the ble_gap_event function. The first event happens when the device gets connected, and the other event happens when the device performs the first action and will be able to perform the second one.
The ble_gatt_svc_def function defines the pointers to the services and some kind of definition function which is required in BLE where we define the UUID. We could enable reading of the device and writing on the device, and we can give data and any command to the ESP32.

Summary:

Setting Up Data Characteristics

In BLE communication, data characteristics define the type of data your device can send and receive. They act as containers for information and provide essential details about the data they hold. To set up characteristics in ESP-IDF, follow these steps:

Step 1: Define Your Custom Characteristic

In the ESP-IDF project, we navigate to the relevant source file and define our custom data characteristics.

Step 2: Register the Characteristic

Next, we need to register the characteristic with the ESP-IDF BLE stack. This step ensures that our device can advertise and accept data related to this characteristic.

Maintaining Persistent BLE Connections

Maintaining a stable and persistent connection is crucial in BLE applications. It ensures that our ESP32 board stays connected to other devices reliably. Here's how to achieve this:

Step 1: Set Connection Parameters

In our ESP-IDF project, we can configure the connection parameters to suit our application. These parameters include connection interval, slave latency, and supervision timeout.

Step 2: Handle Connection Events

Implement callback functions to handle connection events. We can use these callbacks to perform actions like data exchange or notifications when a connection is established or terminated. By handling connection events effectively, we can ensure that our ESP32 maintains stable connections.

Making The Device Discoverable or Non-Discoverable

BLE devices can be either discoverable or non-discoverable, depending on the application's requirements.

Define Discoverable or Non-Discoverable Mode

We can define our device's discoverability mode in our ESP-IDF project. By toggling between these modes, we can control when our ESP32 board is visible to other BLE devices.

Lastly, we can write the code to control the ESP32 light.

Using the idf.py command in the ESP-IDF CMD, we can create a project, build it, flash it, and then monitor the ESP32. Now when we turn on the Bluetooth of our phone, we can see that our server (BLE-Server) is visible and discoverable in the nrfConnect application. After connecting to the server, we are able to send commands (LIGHT ON/OFF) to control the ESP32 light.