Table of Contents

1. nRF52840 Zephyr BLE client 2
1.1. BLE Health Temperature peripheral 2
1.2. BLE central 2
1.2.1. Scanning the neighbourhood 2
1.2.2. Searching advertising data for info on service 3
1.2.3. Establishing the connection 3
1.2.4. Discovering the service 3
1.2.5. Searching for the characteristic 3
1.2.6. Subscribing to the data with indication 3
1.2.7. Ready! 3
1.3. Tasks to do during the laboratory 4

1. nRF52840 Zephyr BLE client

Usually, the BLE client role plays some user device eg. mobile phone, tablet or computer. In IoT systems sometimes BLE is used for M2M (machine-to-machine) data transmission. In such a situation an automatic BLE client must be implemented to connect to the peripherals and read the data eg. coming from BLE-capable sensors. BLE client must be configured to connect to the peripheral which exposes some specific service and characteristic. To do the lab exercise there is a need to create the peripheral device first.

1.1. BLE Health Temperature peripheral

The default Health Temperature example is sufficient to make the laboratory. How to import the project, flash, and execute is shown in peripheral example.

1.2. BLE central

Among the examples provided with Zephyr OS, there is also one complementary to the Health Temperature peripheral. It is named central_ht [1)]. The program is quite complex, here we will explain the most important elements and the process of establishing the connection.
The process of establishing and configuring a connection has a few steps:

1. scanning the neighbourhood to find connectable devices
2. comparing the advertising data to look for information on the required service
3. establishing the connection to peripheral
4. retrieving information on services and characteristics
5. subscribing to the interesting data with a notification or indication mechanism (optional)
6. reading data (by reading the characteristic value or automatically with notify/indicate)

1.2.1. Scanning the neighbourhood

In the main function, after the successful initialisation of the Bluetooth subsystem, scanning for nearby devices is started.

```
err = bt_enable(NULL);
...
err = scan_start();
...
```

The “scan_start()” function calls “bt_le_scan_start(&scan_param, device_found)” function. The second parameter is the callback function which processes scan results.

1.2.2. Searching advertising data for info on service

The “device_found()” function parses received advertising data with the “bt_data_parse()” function to the function “eir_found()”. The latter function searches advertising data for the UUID of the Health Thermometer (0x1809).

1.2.3. Establishing the connection

After a successful search establishes a connection with the remote device by calling the function “bt_conn_le_create()”.
Further processing is done by callback functions registered by the macro:

```
BT_CONN_CB_DEFINE(conn_callbacks) = {
    .connected = connected,
    .disconnected = disconnected,
};
```

Zephyr OS keeps the “bt_conn_cb” structure that allows the programmer to attach callback functions to chosen events. “BT_CONN_CB_DEFINE” macro attaches “connected” and “disconnected” functions to appropriate events. The “connected()” function is executed after successfully establishing the connection to the peripheral device.

1.2.4. Discovering the service

This function registers the callback function “discover_func()”, and starts the discovery process by searching for the Health Thermometer service (UUID = 0x1809).

1.2.5. Searching for the characteristic

After finding this service discovery_func() is executed which performs further searching. First, it looks for the Temperature Measurement characteristic (UUID = 0x2a1c).

1.2.6. Subscribing to the data with indication

This characteristic should have the descriptor Client Characteristic Configuration (UUID = 0x1902) which makes it possible to enable notifications or indications. The indication enabling and registering the “notify_func()” as the callback for incoming indications finishes the establishing connection process.

1.2.7. Ready!

Notice that after every successful discovery the function “discovery_func()” is called as the callback. While the peripheral device updates the thermometer sensor reading, our central device receives this

data and calls the “notify_func()” callback which prints the data using the UART port.

1.3. Tasks to do during the laboratory

1. Create the Health Thermometer peripheral device and test if it operates properly
2. Import central device program “central_ht” (main.c and prj.conf files must be imported)
3. Compile and execute it observing its behaviour with and without peripheral running
4. Modify the peripheral code and central code to enable LED control on the central board with a button press on the peripheral board



If you find problems with undefined BT_ATT_FIRST_ATTRIBUTE_HANDLE and BT_ATT_LAST_ATTRIBUTE_HANDLE define them as in the following code.

```
#define BT_ATT_FIRST_ATTRIBUTE_HANDLE 0x0001
#define BT_ATT_LAST_ATTRIBUTE_HANDLE 0xFFFF
```



To send simple binary data you can use a binary sensor service (UUID = 0x183B) with a binary sensor response characteristic (UUID = 0x2B2C).

[1)]

..\.platformio\packages\framework-zephyr\samples\bluetooth\central_ht

From:
https://dokuwiki.zmitac.aei.polsl.pl/ - **RAU6 Dokuwiki**

Permanent link:
https://dokuwiki.zmitac.aei.polsl.pl/en:edu:zmitac:s2:aih:lab:nrf52zephyrblecentral

Last update: **2023/12/19 23:19**