WES 269 Co-design Hardware & Software – Pannuto	**Ricardo Lizárraga** rilizarraga@ucsd.edu **PID: A69028483**	**Submit to:** (Prof) **Patrick William Pannuto** (TA) **Anthony Tarbinian** (TA) **Samir Rashid**

Post-Lab 2: BLE

What to submit?

Please use this document as a template, add your responses directly, and export it as a PDF to Gradescope. Each group should submit one post-lab.

Group name: Ricardo Lizárraga

Team member names: Ricardo Lizárraga

Link to GitHub repository:

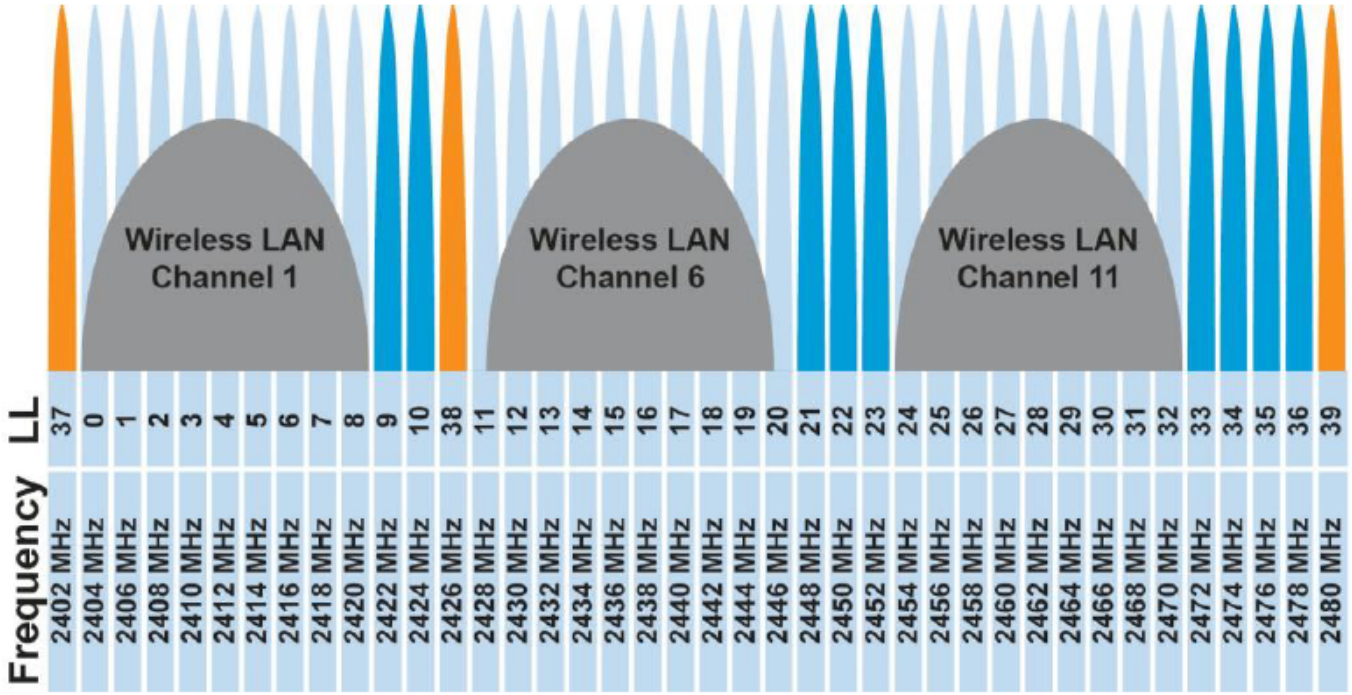
https://github.com/RicardoUCSD/Postlab2_BLE

BLE packet Protocol Data Unit (**PDU**)

The Bluetooth® Low Energy (BLE) Link Layer has only one packet format used for both advertising channel packets and data channel packets. BLE packet Protocol Data Unit (PDU) size in specifications v4. 0 and v4. 1 is 2-39 bytes

- 3 advertising channel
- 37 data channels

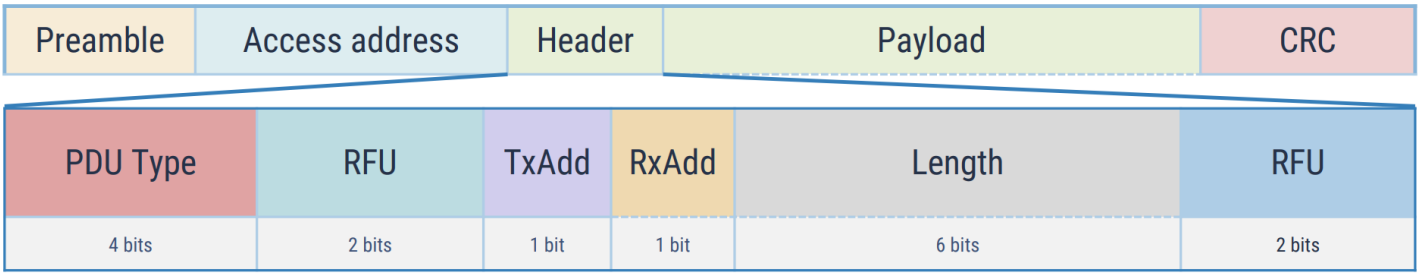
- 2.4 GHz carrier, Forty 2-MHz channels, 1 Mbps
  - 37, 38, 39 for advertising
  - 0-36 for connection (FHSS)



Preamble	Access address	PDU	CRC
1 Byte	4 Bytes	2 - 37 Byte(s)	3 Bytes

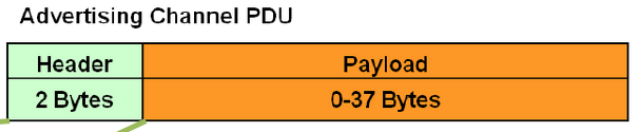
- **Preamble**: Used for sync and timing estimation (broadcast = `0xAA`)
- **Access address**: Broadcast (`0x8E89BED6`)
- **PDU**: Protocol Data Unit (header + payload)
- **CRC**: Cyclic Redundancy Check (error-detection)

ADVERTISING PDU



- **PDU type**: Indicate PDU type of advertisement
- **RFU**: Reserved for Future Use
- **TxAdd**: Indicates whether the advertiser's address (in payload) is public or random
- **RxAdd**: N/A for advertising
- **Length**: Define the size of the payload in Bytes (6 – 37 Bytes)

BLE advertising header



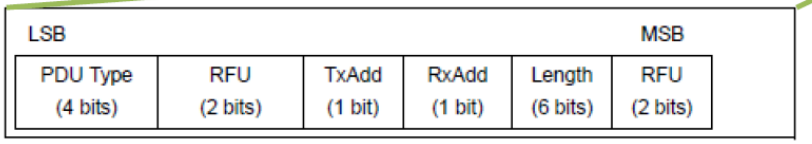


Figure 2.3: Advertising channel PDU Header

PDU Type $b_3b_2b_1b_0$	Packet Name
0000	ADV_IND
0001	ADV_DIRECT_IND
0010	ADV_NONCONN_IND
0011	SCAN_REQ
0100	SCAN_RSP
0101	CONNECT_REQ
0110	ADV_SCAN_IND
0111-1111	Reserved

Table 2.1: Advertising channel PDU Header's PDU Type field encoding

- ADV_IND
  - Advertisement
  - Allows connections and scan requests
- ADV_NONCONN_IND
  - Advertisement
  - No connections or scan requests
- ADV_SCAN_IND
  - Advertisement
  - No connections but allows scan requests
- SCAN_REQ
  - Scan request
- SCAN_RSP
  - Scan response

Setting Up the Nordic nRF Sniffer

https://dojofive.com/blog/using-the-nordic-nrf-sniffer-for-ble/

ble-sniffer command overview
https://docs.nordicsemi.com/bundle/nrfutil/page/nrfutil-ble-sniffer/guides/overview.html

Wireshark filtering used:

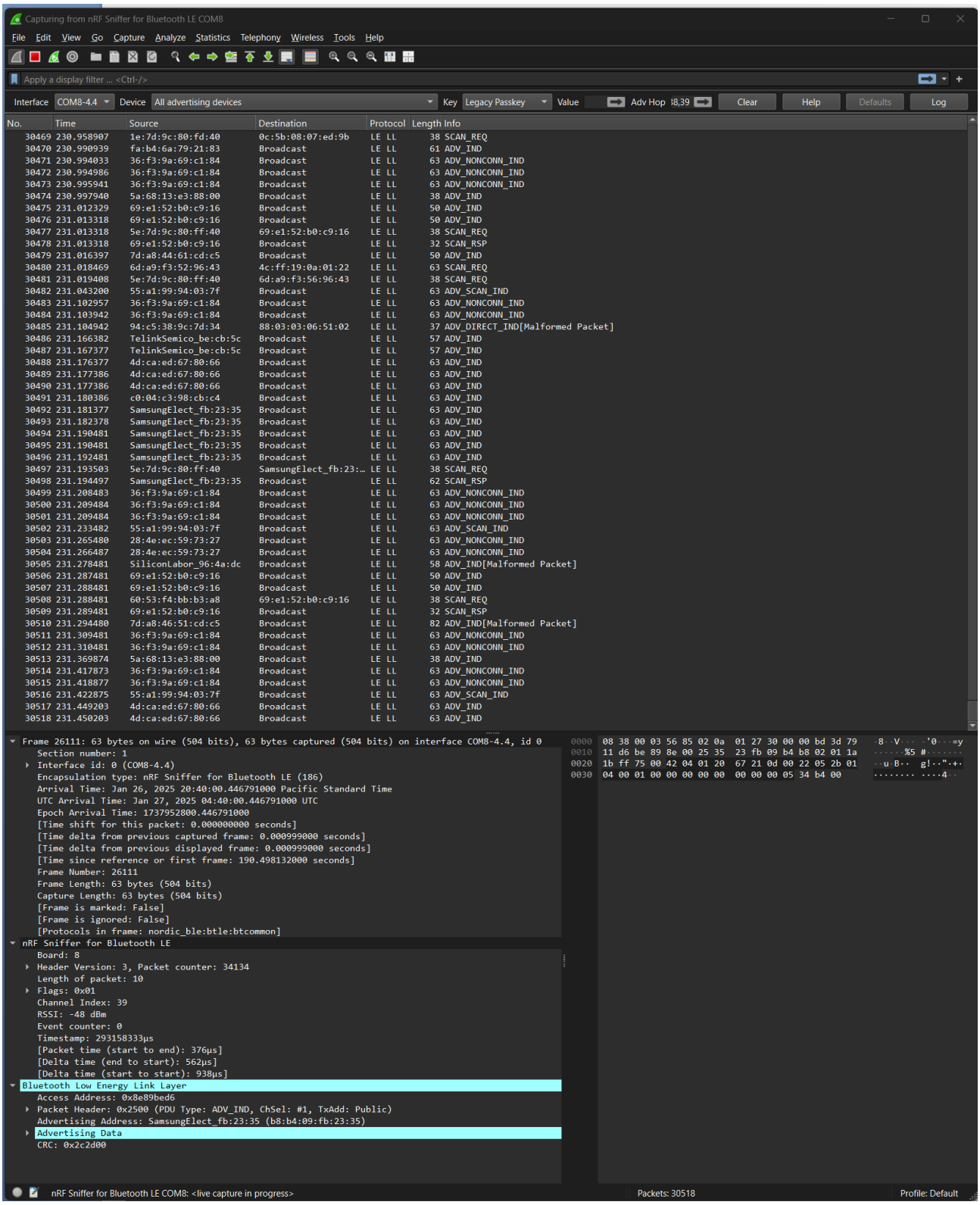
btle.advertising_address == f2:f1:d1:a1:9c:1f
btle.advertising_header.pdu_type == 0X3
btle.advertising_address: Filters packets by MAC address
nordic_ble.rssi: Filters packets by RSSI
ADV_IND: Filters packets that are undirected advertising
ADV_EXT_IND: Filters packets that are extended advertising on primary advertising channels
AUX_ADV_IND: Filters packets that are extended advertising on secondary advertising channels

btcommon.eir_ad.entry.service_data == d5:c0:ff:ee or btle.advertising_address == d8:4e:a6:cf:42:22

To filter for the dongle:
btle.advertising_address == d8:4e:a6:cf:42:22

F: Investigating BLE Advertisements

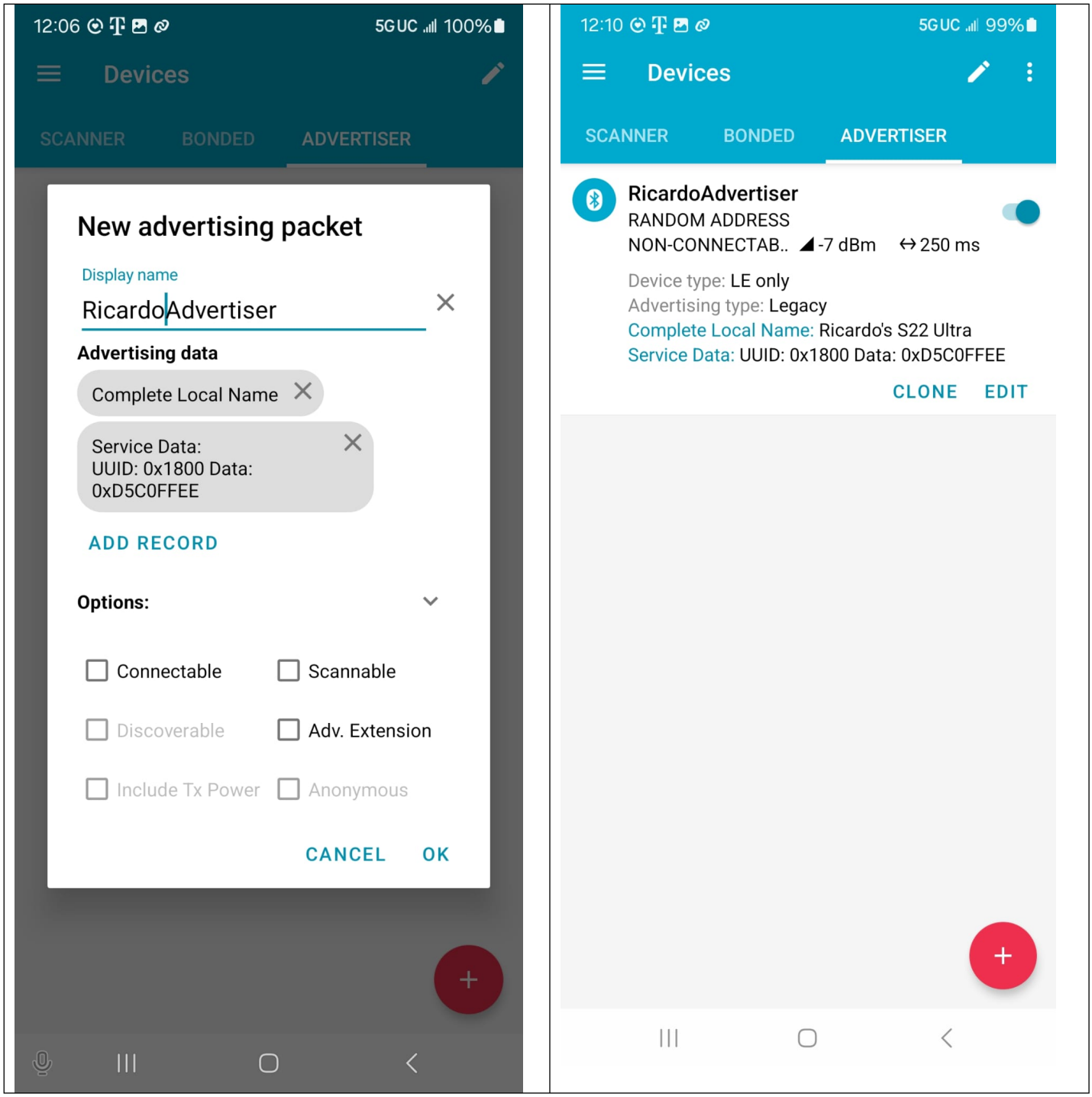
1. **How many transmissions do you see in one second?**
   I ran it in the lab on campus, and I saw hundreds per seconds, then at home and I see pretty much about the same, but definitely a number of BLE devices around. **Time (Timestamp) can be used to determine exactly the number)**



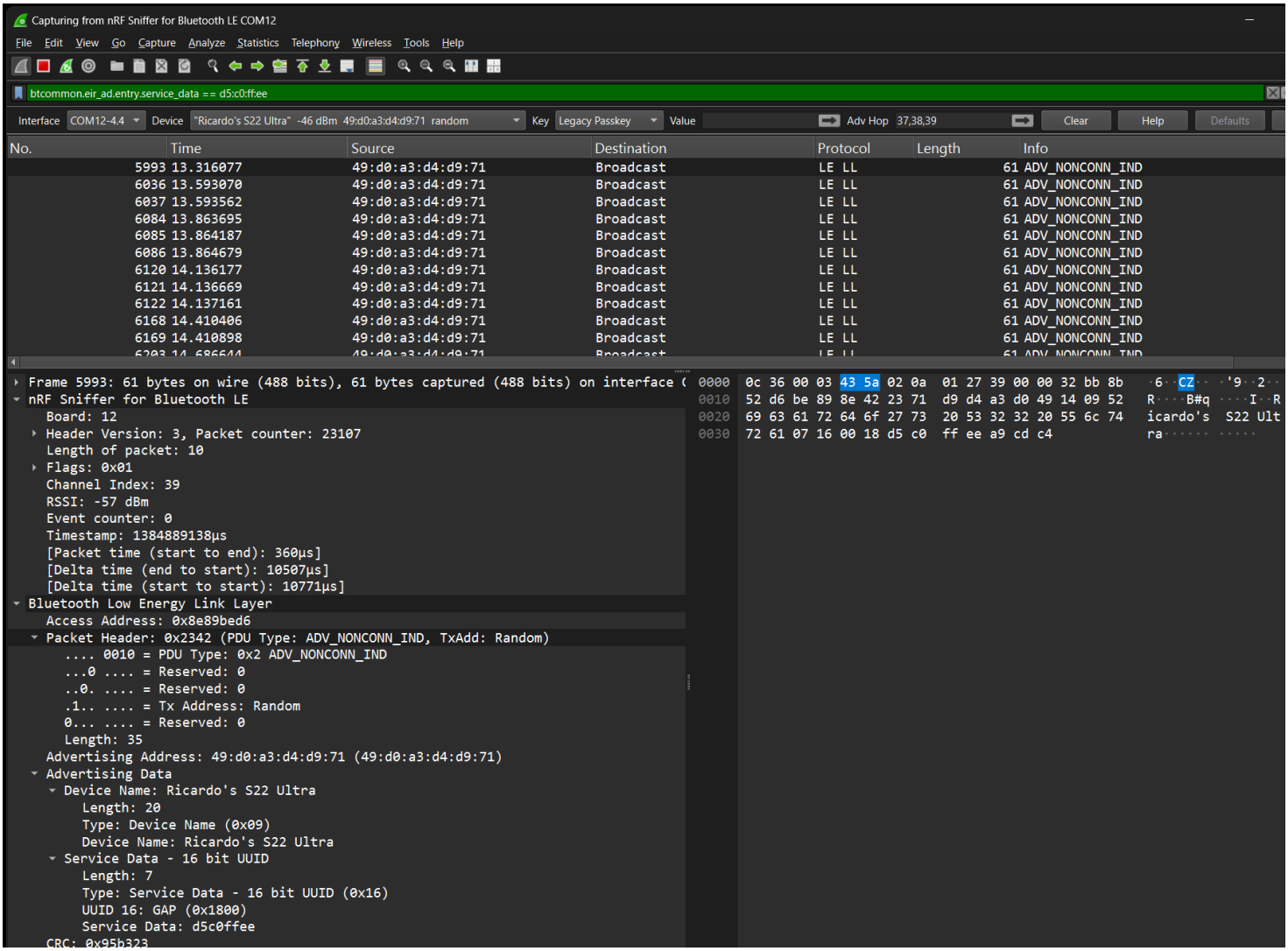
2. **Pick a received advertisement, show me the packet data, and explain the meaning of all of the bytes of it.**

Capture #1: RicardoAdvertiser

Using nRF Connect For Mobile to create a custom advertisement message “RicardoAdvertiser”



Filter: btcommon.eir_ad.entry.service_data == d5:c0:ff:ee



PDU: Protocol Data Unit (header + payload)

```
Bluetooth Low Energy Link Layer
  Access Address: 0x8e89bed6
  Packet Header: 0x2342 (PDU Type: ADV_NONCONN_IND, TxAdd: Random)
    .... 0010 = PDU Type: 0x2 ADV_NONCONN_IND
    ...0 .... = Reserved: 0
    ..0. .... = Reserved: 0
    .1.. .... = Tx Address: Random
    0... .... = Reserved: 0
    Length: 35
  Advertising Address: 49:d0:a3:d4:d9:71 (49:d0:a3:d4:d9:71)
  Advertising Data
    Device Name: Ricardo's S22 Ultra
      Length: 20
      Type: Device Name (0x09)
      Device Name: Ricardo's S22 Ultra
    Service Data - 16 bit UUID
      Length: 7
      Type: Service Data - 16 bit UUID (0x16)
      UUID 16: GAP (0x1800)
      Service Data: d5c0ffee
  CRC: 0x95b323
```

PDU Type	0010 = 0x2 ADV_NONCONN_IND
...0	Reserved 0
..0.	Reserved 0
.1..	Tx Address: Random
0...	Reserved 0
MAC addr	49:d0:a3:d4:d9:71
Length	20

Advertising Data:

```
Advertising Data
  Device Name: Ricardo's S22 Ultra
    Length: 20
    Type: Device Name (0x09)
    Device Name: Ricardo's S22 Ultra
```

Service Data:

```
Service Data - 16 bit UUID
  Length: 7
  Type: Service Data - 16 bit UUID (0x16)
  UUID 16: GAP (0x1800)
  Service Data: d5c0ffee
```

CRC	0X95b323

CRC calculated for this packet

I used this link to help me improve filtering techniques un Wireshark:
https://dojofive.com/blog/using-the-nordic-nrf-sniffer-for-ble/

Capture #2: Filtering by PDU type

Filter applied: btle.advertising_header.pdu_type == 0X3

btle.advertising_header.pdu_type == 0X3

Interface COM8-4.4 Device All advertising devices Key Legacy Passkey Value Adv Hop 37,38,39

No.	Time	Source	Destination	Protocol	Length	Info
7531	33.177614	AmazonTechno_3a:47:c1	TelinkSemico_f6:27:49	LE LL	38	SCAN_REQ
7534	33.231223	49:64:6d:f7:7d:05	59:c8:19:26:a8:ab	LE LL	38	SCAN_REQ
7553	33.440998	SamsungElect_fb:23:35	47:d1:54:72:c9:d7	LE LL	38	SCAN_REQ
7558	33.460266	SamsungElect_fb:23:35	55:fa:f4:3c:0c:49	LE LL	38	SCAN_REQ
7565	33.522598	49:64:6d:f7:7d:05	SamsungElect_fb:23:35	LE LL	38	SCAN_REQ
7576	33.586788	49:64:6d:f7:7d:05	59:c8:19:26:a8:ab	LE LL	38	SCAN_REQ
7594	33.765582	49:64:6d:f7:7d:05	51:7f:59:5f:b8:2f	LE LL	38	SCAN_REQ
7609	33.943301	49:64:6d:f7:7d:05	1a:a3:d8:4c:60:21	LE LL	38	SCAN_REQ
7612	33.949365	49:64:6d:f7:7d:05	SeongjiIndus_34:ac:e4	LE LL	38	SCAN_REQ
7633	34.120615	AmazonTechno_3a:47:c1	40:fc:86:6d:95:2e	LE LL	38	SCAN_REQ
7648	34.265304	49:64:6d:f7:7d:05	47:d1:54:72:c9:d7	LE LL	38	SCAN_REQ
7663	34.473125	SamsungElect_fb:23:35	65:e8:9b:f9:1f:70	LE LL	38	SCAN_REQ
7666	34.479223	49:64:6d:f7:7d:05	SamsungElect_bd:4a:e1	LE LL	38	SCAN_REQ
7669	34.494020	49:64:6d:f7:7d:05	4e:38:c0:82:66:68	LE LL	38	SCAN_REQ
7693	34.800902	49:64:6d:f7:7d:05	1a:a3:d8:4c:60:21	LE LL	38	SCAN_REQ
7708	34.956694	41:f2:92:2b:c5:f6	55:fa:f4:3c:0c:49	LE LL	38	SCAN_REQ
7718	35.025530	49:64:6d:f7:7d:05	59:c8:19:26:a8:ab	LE LL	38	SCAN_REQ
7721	35.032517	49:64:6d:f7:7d:05	7c:d6:0e:83:1e:4f	LE LL	38	SCAN_REQ
7724	35.037568	49:64:6d:f7:7d:05	SeongjiIndus_34:ac:e4	LE LL	38	SCAN_REQ
7727	35.098378	SamsungElect_fb:23:35	47:d1:54:72:c9:d7	LE LL	38	SCAN_REQ
7741	35.321990	49:64:6d:f7:7d:05	55:fa:f4:3c:0c:49	LE LL	38	SCAN_REQ
7744	35.323996	SamsungElect_fb:23:35	55:fa:f4:3c:0c:49	LE LL	38	SCAN_REQ
7750	35.376875	AmazonTechno_3a:47:c1	1a:a3:d8:4c:60:21	LE LL	38	SCAN_REQ
7752	35.397699	49:64:6d:f7:7d:05	59:c8:19:26:a8:ab	LE LL	38	SCAN_REQ
7770	35.577442	49:64:6d:f7:7d:05	7c:d6:0e:83:1e:4f	LE LL	38	SCAN_REQ
7785	35.758759	49:64:6d:f7:7d:05	40:fc:86:6d:95:2e	LE LL	38	SCAN_REQ
7788	35.768029	49:64:6d:f7:7d:05	59:c8:19:26:a8:ab	LE LL	38	SCAN_REQ
7793	35.820473	SamsungElect_fb:23:35	4c:8f:8c:7c:d1:47	LE LL	38	SCAN_REQ
7801	35.879644	49:64:6d:f7:7d:05	55:fa:f4:3c:0c:49	LE LL	38	SCAN_REQ
7815	36.005438	AmazonTechno_3a:47:c1	59:c8:19:26:a8:ab	LE LL	38	SCAN_REQ
7816	36.007362	49:64:6d:f6:7d:04	59:c8:99:26:a8:ab	LE LL	71	SCAN_REQ
7820	36.057802	49:64:6d:f7:7d:05	7f:32:d5:09:e9:c5	LE LL	38	SCAN_REQ
7824	36.062799	34:54:6d:f7:7d:05	ec:d8:ac:5e:3b:ab	LE LL	38	SCAN_REQ
7834	36.123059	49:64:6d:f7:7d:05	SeongjiIndus_34:ac:e4	LE LL	38	SCAN_REQ
7835	36.141146	61:22:44:b1:f8:a0	HP_4f:89:da	LE LL	38	SCAN_REQ
7848	36.223306	49:64:6d:f7:7d:05	SeongjiIndus_34:ac:e4	LE LL	38	SCAN_REQ
7868	36.439114	SamsungElect_fb:23:35	55:fa:f4:3c:0c:49	LE LL	38	SCAN_REQ
7887	36.604933	49:64:6d:f7:7d:05	fa:b4:6a:79:21:83	LE LL	38	SCAN_REQ
7895	36.637737	AmazonTechno_3a:47:c1	4c:8f:8c:7c:d1:47	LE LL	38	SCAN_REQ
7927	36.936906	AmazonTechno_3a:47:c1	51:7f:59:5f:b8:2f	LE LL	38	SCAN_REQ
7929	36.991810	71:75:d9:43:c1:89	7d:9a:7e:f5:fe:00	LE LL	38	SCAN_REQ
7931	37.003415	49:64:6d:f7:7d:05	55:fa:f4:3c:0c:49	LE LL	38	SCAN_REQ

PDU: Protocol Data Unit (header + payload)

```
▾ Bluetooth Low Energy Link Layer
    Access Address: 0x8e89bed6
  ▾ Packet Header: 0x0c83 (PDU Type: SCAN_REQ, TxAdd: Public, RxAdd: Random)
      .... 0011 = PDU Type: 0x3 SCAN_REQ
      ...0 .... = Reserved: 0
      ..0. .... = Reserved: 0
      .0.. .... = Tx Address: Public
      1... .... = Rx Address: Random
      Length: 12
    Scanning Address: SamsungElect_fb:23:35 (b8:b4:09:fb:23:35)
    Advertising Address: 47:d1:54:72:c9:d7 (47:d1:54:72:c9:d7)
    CRC: 0xb373c0
```

PDU Type	0011 = 0X3
...0	Reserved
..0.	Reserved
PDU Type	Tx address
PDU Type	Rx address
Length	12
Scanning Address: SamsungElect_fb:23:35 (b8:b4:09:fb:23:35) Advertising Address: 47:d1:54:72:c9:d7 (47:d1:54:72:c9:d7)	
CRC	0Xb373c0

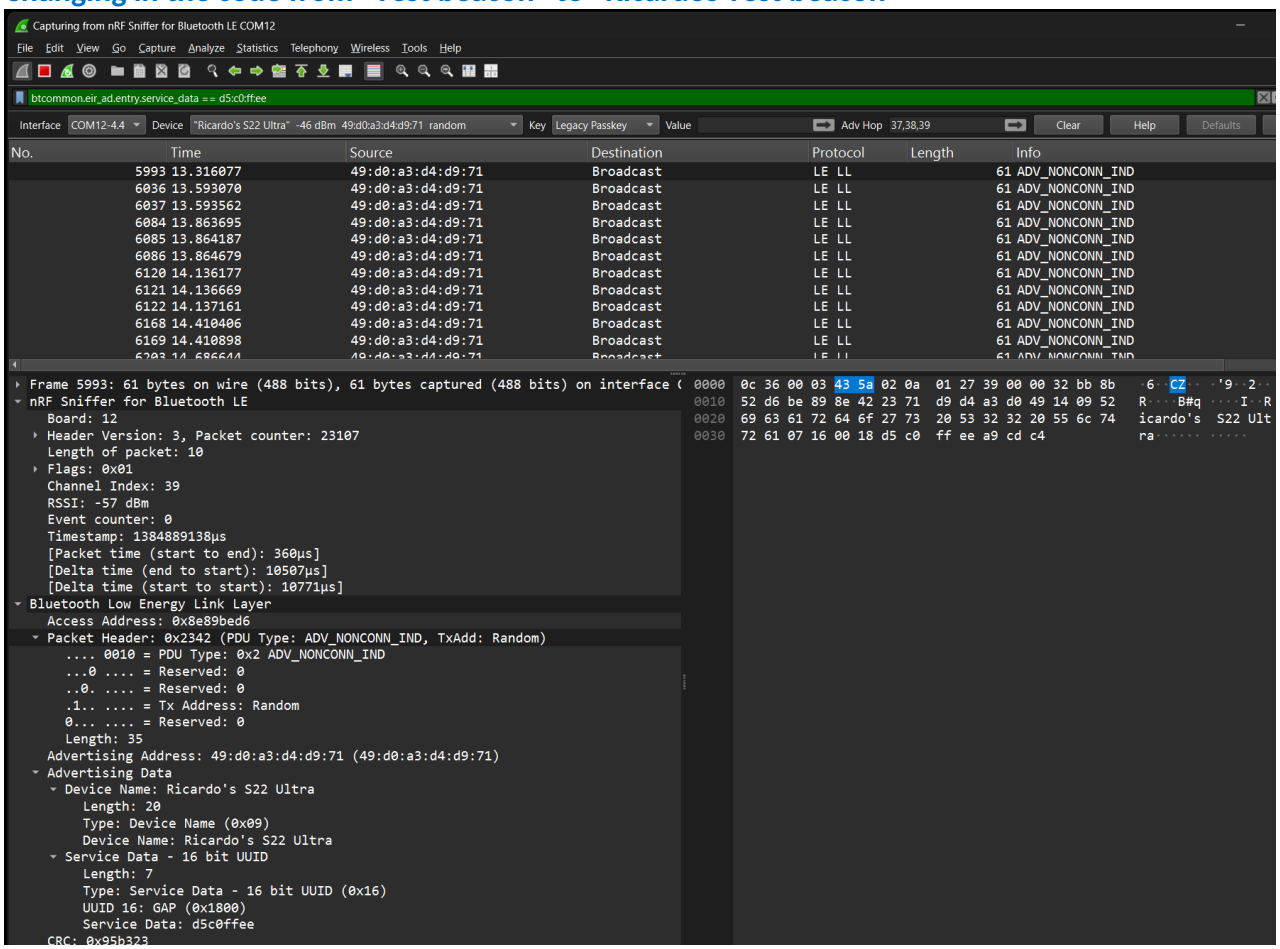
H: Programming a BLE Advertiser

3. Show evidence you were able to create an advertiser with your custom name:

Build and Flashing:

C:\Users\rliz0\beacon

Changing in the code from “Test beacon” to “Ricardos Test beacon”



PDU: Protocol Data Unit (header + payload)

```
Bluetooth Low Energy Link Layer
  Access Address: 0x8e89bed6
  Packet Header: 0x2342 (PDU Type: ADV_NONCONN_IND, TxAdd: Random)
    .... 0010 = PDU Type: 0x2 ADV_NONCONN_IND
    ...0 .... = Reserved: 0
    ..0. .... = Reserved: 0
    .1.. .... = Tx Address: Random
    0... .... = Reserved: 0
    Length: 35
  Advertising Address: 49:d0:a3:d4:d9:71 (49:d0:a3:d4:d9:71)
  Advertising Data
    Device Name: Ricardo's S22 Ultra
      Length: 20
      Type: Device Name (0x09)
      Device Name: Ricardo's S22 Ultra
    Service Data - 16 bit UUID
      Length: 7
      Type: Service Data - 16 bit UUID (0x16)
      UUID 16: GAP (0x1800)
      Service Data: d5c0ffee
  CRC: 0x95b323
```

PDU Type	0010 = 0x2 ADV_NONCONN_IND
...0	Reserved 0
..0.	Reserved 0
.1..	Tx Address: Random
0...	Reserved 0
MAC addr	49:d0:a3:d4:d9:71
Length	20

Advertising Data:

```
Advertising Data
  Device Name: Ricardo's S22 Ultra
    Length: 20
    Type: Device Name (0x09)
    Device Name: Ricardo's S22 Ultra
```

Service Data:

```
Service Data - 16 bit UUID
  Length: 7
  Type: Service Data - 16 bit UUID (0x16)
  UUID 16: GAP (0x1800)
  Service Data: d5c0ffee
```

CRC	0X95b323

CRC calculated for this packet

Using documentation: https://docs.zephyrproject.org/apidoc/latest/index.html to change the advertising interval so that packets are sent every 333 ms.

```c
/* Start advertising */
err = bt_le_adv_start(BT_LE_ADV_NCONN_IDENTITY, ad, ARRAY_SIZE(ad),
                      sd, ARRAY_SIZE(sd));
if (err) {
    printk("Advertising failed to start (err %d)\n", err);
    return;
}
```

```c
/** Non-connectable advertising with @ref BT_LE_ADV_OPT_USE_IDENTITY */
#define BT_LE_ADV_NCONN_IDENTITY BT_LE_ADV_PARAM(BT_LE_ADV_OPT_USE_IDENTITY, \
                                 BT_GAP_ADV_FAST_INT_MIN_2, \
                                 BT_GAP_ADV_FAST_INT_MAX_2, \
                                 NULL)
```

According to documentation: https://docs.zephyrproject.org/apidoc/latest/index.html

main.c | bluetooth.h | gap.h | autoconf.h

C: > ncs > v2.9.0 > zephyr > include > zephyr > bluetooth > gap.h > ...

```c
#ifndef ZEPHYR_INCLUDE_BLUETOOTH_GAP_H_
/**
/** Generic Unknown */
#define BT_APPEARANCE_UNKNOWN                                0x0000
/** Generic Phone */
#define BT_APPEARANCE_GENERIC_PHONE                          0x0040
/** Generic Computer */
#define BT_APPEARANCE_GENERIC_COMPUTER                       0x0080
/** Desktop Workstation */
#define BT_APPEARANCE_COMPUTER_DESKTOP_WORKSTATION           0x0081
/** Server-class Computer */
#define BT_APPEARANCE_COMPUTER_SERVER_CLASS                  0x0082
/** Laptop */
#define BT_APPEARANCE_COMPUTER_LAPTOP                        0x0083
/** Handheld PC/PDA (clamshell) */
#define BT_APPEARANCE_COMPUTER_HANDHELD_PCPDA                0x0084
/** Palmsize PC/PDA */
#define BT_APPEARANCE_COMPUTER_PALMSIZE_PCPDA                0x0085
/** Wearable computer (watch size) */
#define BT_APPEARANCE_COMPUTER_WEARABLE_COMPUTER             0x0086
/** Tablet */
#define BT_APPEARANCE_COMPUTER_TABLET                        0x0087
/** Docking Station */
#define BT_APPEARANCE_COMPUTER_DOCKING_STATION               0x0088
/** All in One */
#define BT_APPEARANCE_COMPUTER_ALL_IN_ONE                    0x0089
/** Blade Server */
#define BT_APPEARANCE_COMPUTER_BLADE_SERVER                  0x008A
/** Convertible */
#define BT_APPEARANCE_COMPUTER_CONVERTIBLE                   0x008B
/** Detachable */
#define BT_APPEARANCE_COMPUTER_DETACHABLE                    0x008C
/** IoT Gateway */
#define BT_APPEARANCE_COMPUTER_IOT_GATEWAY                   0x008D
/** Mini PC */
#define BT_APPEARANCE_COMPUTER_MINI_PC                       0x008E
/** Stick PC */
#define BT_APPEARANCE_COMPUTER_STICK_PC                      0x008F
```

BT_LE_ADV_PARAM

```c
#define BT_LE_ADV_PARAM ( _options,
                          _int_min,
                          _int_max,
                          _peer )
```

```c
#include <zephyr/bluetooth/bluetooth.h>
```

Value:

```c
((const struct bt_le_adv_param[]) { \
        BT_LE_ADV_PARAM_INIT(_options, _int_min, _int_max, _peer) \
 })
```

Helper to declare advertising parameters inline.

Parameters

_options	Advertising Options
_int_min	Minimum advertising interval
_int_max	Maximum advertising interval
_peer	Peer address, set to NULL for undirected advertising or address of peer for directed advertising.

bt_le_adv_param Struct Reference

Connectivity • Bluetooth APIs • Generic Access Profile (GAP)

LE Advertising Parameters. More...

`#include <bluetooth.h>`

Data Fields

`uint8_t id`

Local identity.

`uint8_t sid`

Advertising Set Identifier, valid range 0x00 - 0x0f.

`uint8_t secondary_max_skip`

Secondary channel maximum skip count.

`uint32_t options`

Bit-field of advertising options.

`uint32_t interval_min`

Minimum Advertising Interval (N * 0.625 milliseconds) Minimum Advertising Interval shall be less than or equal to the Maximum Advertising Interval.

`uint32_t interval_max`

Maximum Advertising Interval (N * 0.625 milliseconds) Minimum Advertising Interval shall be less than or equal to the Maximum Advertising Interval.

`const bt_addr_le_t * peer`

Directed advertising to peer.

Detailed Description

LE Advertising Parameters.

Field Documentation

id

`uint8_t bt_le_adv_param::id`

Local identity.

Note When extended advertising `CONFIG_BT_EXT_ADV` is not enabled or not supported by the controller it is not possible to scan and advertise simultaneously using two different random addresses.

interval_max

`uint32_t bt_le_adv_param::interval_max`

Maximum Advertising Interval (N * 0.625 milliseconds) Minimum Advertising Interval shall be less than or equal to the Maximum Advertising Interval.

The Minimum Advertising Interval and Maximum Advertising Interval should not be the same value (as stated in Bluetooth Core Spec 5.2, section 7.8.5) Range: 0x0020 to 0x4000

interval_min

`uint32_t bt_le_adv_param::interval_min`

Minimum Advertising Interval (N * 0.625 milliseconds) Minimum Advertising Interval shall be less than or equal to the Maximum Advertising Interval.

The Minimum Advertising Interval and Maximum Advertising Interval should not be the same value (as stated in Bluetooth Core Spec 5.2, section 7.8.5) Range: 0x0020 to 0x4000

options

`uint32_t bt_le_adv_param::options`

Bit-field of advertising options.

peer

`const bt_addr_le_t* bt_le_adv_param::peer`

Directed advertising to peer.

When this parameter is set the advertiser will send directed advertising to the remote device.

The advertising type will either be high duty cycle, or low duty cycle if the BT_LE_ADV_OPT_DIR_MODE_LOW_DUTY option is enabled. When using BT_LE_ADV_OPT_EXT_ADV then only low duty cycle is allowed.

In case of connectable high duty cycle if the connection could not be established within the timeout the connected() callback will be called with the status set to BT_HCI_ERR_ADV_TIMEOUT.

secondary_max_skip

`uint8_t bt_le_adv_param::secondary_max_skip`

Secondary channel maximum skip count.

Maximum advertising events the advertiser can skip before it must send advertising data on the secondary advertising channel.

Note Requires BT_LE_ADV_OPT_EXT_ADV

sid

`uint8_t bt_le_adv_param::sid`

Advertising Set Identifier, valid range 0x00 - 0x0f.

Note Requires BT_LE_ADV_OPT_EXT_ADV

Change the advertising interval so that packets are sent every 333 ms.

- You'll need to change the first argument to `bt_le_adv_start()`
- Look up the BT_LE_ADV_PARAM macro by searching the Zephyr docs

Note, the advertising intervals are specified in multiples of 0.625mS

Default parameters:

```
BT_LE_ADV_OPT_USE_IDENTITY = BIT(2),
#define BT_GAP_ADV_FAST_INT_MIN_2              0x00a0  /* 100 ms   */
#define BT_GAP_ADV_FAST_INT_MAX_2              0x00f0  /* 150 ms   */

0x00f0 = 240d
So, 240d * 0.625mS = 150mS
```

Changed parameters to reflect 333.125mS interval:

```
BT_LE_ADV_OPT_USE_IDENTITY = BIT(2),
#define BT_GAP_ADV_FAST_INT_MIN_2              0x00a0  /* 100 ms   */
#define BT_GAP_ADV_FAST_INT_MAX_2              0x00f0  /* 150 ms   */

0x0215 = 533d
So, 533d * 0.625mS = 333.125mS
```

Add appearance to the advertising payload. The value 0x0040 should make the device claim to be a "Generic Phone" per the BLE specification: https://www.bluetooth.com/wp-content/uploads/Files/Specification/HTML/Assigned_Numbers/out/en/Assigned_Numbers.pdf?v=1707335302187

0x0040:

2025-01-24 **Assigned Numbers** / Document

2.6.3 Appearance Sub-category values

These are the assigned values within the assigned ranges of appearance categories.

Last Modified: 2024-10-16

Filename: assigned_numbers/core/appearance_values.yaml

Category (bits 15 to 6)	Subcatgeory (bits 5 to 0)	Value	Name
0x000	0x00	0x0000	Generic Unknown
0x001	0x00	0x0040	Generic Phone
0x002	0x00	0x0080	Generic Computer
	0x01	0x0081	Desktop Workstation
	0x02	0x0082	Server-class Computer
	0x03	0x0083	Laptop
	0x04	0x0084	Handheld PC/PDA (clamshell)
	0x05	0x0085	Palm-size PC/PDA
	0x06	0x0086	Wearable computer (watch size)
	0x07	0x0087	Tablet
	0x08	0x0088	Docking Station
	0x09	0x0089	All in One
	0x0A	0x008A	Blade Server
	0x0B	0x008B	Convertible
	0x0C	0x008C	Detachable
	0x0D	0x008D	IoT Gateway
	0x0E	0x008E	Mini PC
	0x0F	0x008F	Stick PC

Changing to "Mini PC":

2025-01-24 **Assigned Numbers** / Document

2.6.3 Appearance Sub-category values

These are the assigned values within the assigned ranges of appearance categories.

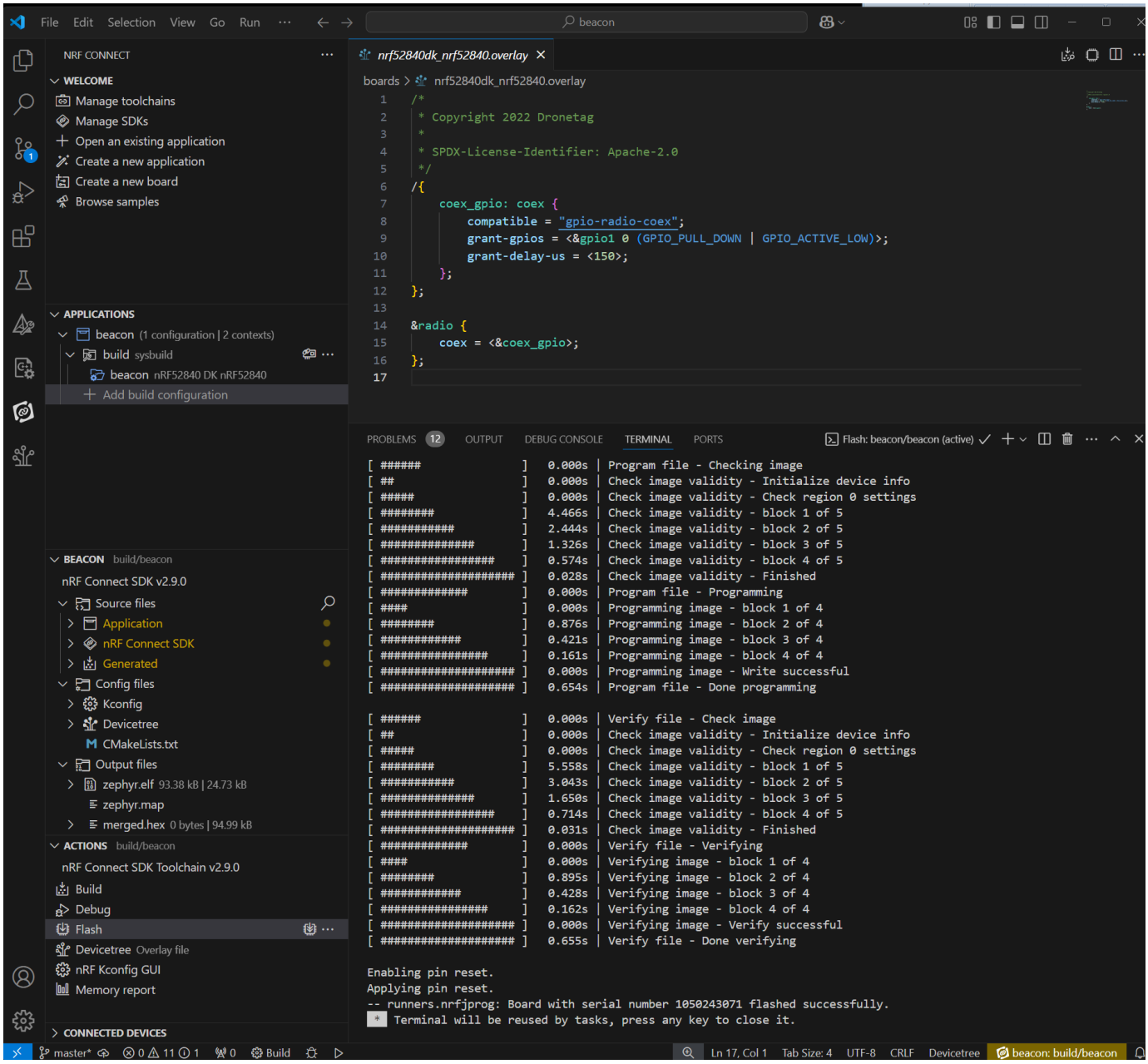
Last Modified: 2024-10-16

Filename: assigned_numbers/core/appearance_values.yaml

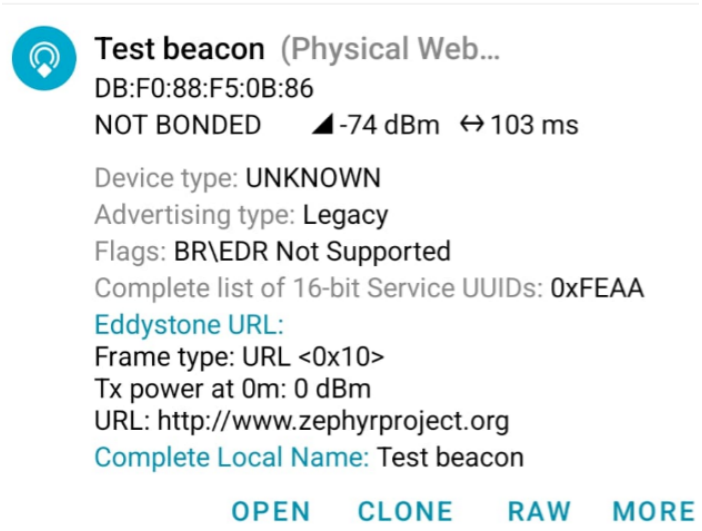
Category (bits 15 to 6)	Subcatgeory (bits 5 to 0)	Value	Name
0x000	0x00	0x0000	Generic Unknown
0x001	0x00	0x0040	Generic Phone
0x002	0x00	0x0080	Generic Computer
	0x01	0x0081	Desktop Workstation
	0x02	0x0082	Server-class Computer
	0x03	0x0083	Laptop
	0x04	0x0084	Handheld PC/PDA (clamshell)
	0x05	0x0085	Palm-size PC/PDA
	0x06	0x0086	Wearable computer (watch size)
	0x07	0x0087	Tablet
	0x08	0x0088	Docking Station
	0x09	0x0089	All in One
	0x0A	0x008A	Blade Server
	0x0B	0x008B	Convertible
	0x0C	0x008C	Detachable
	0x0D	0x008D	IoT Gateway
	0x0E	0x008E	Mini PC
	0x0F	0x008F	Stick PC

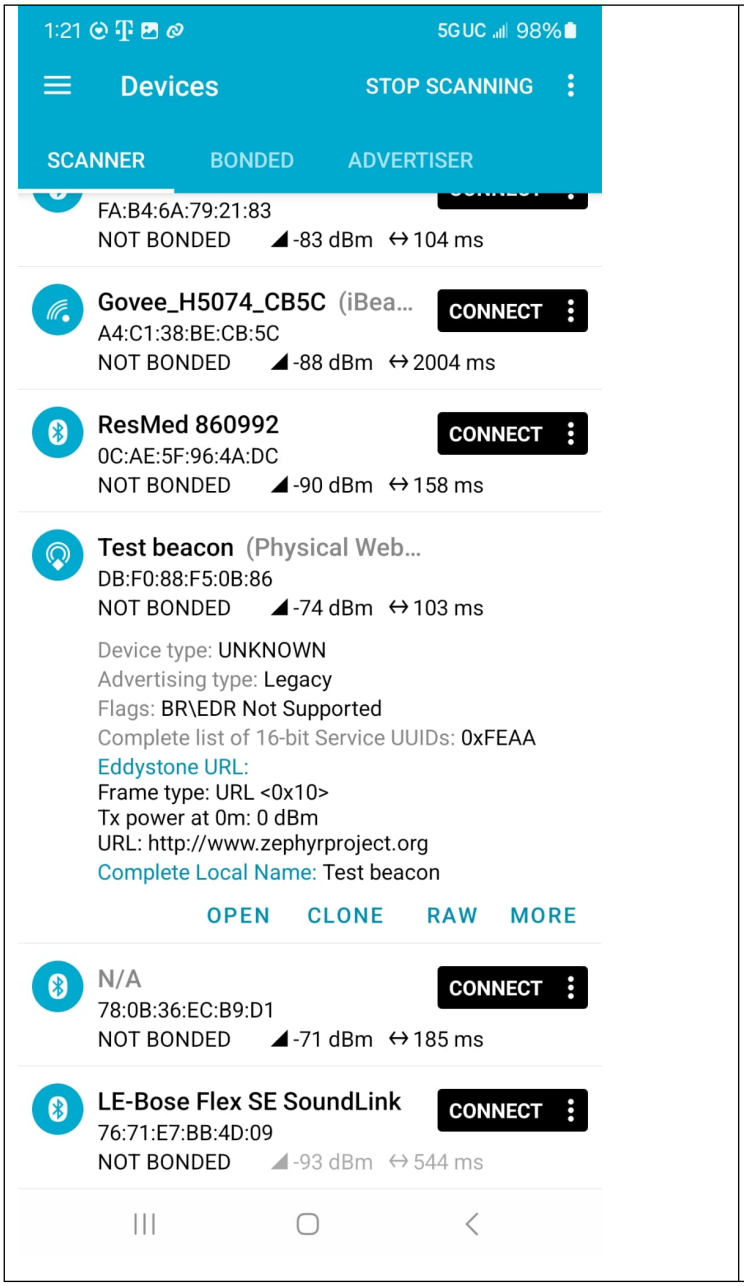
```
726   #define BT_GAP_ADV_FAST_INT_MIN_2              0x00a0  /* 100 ms   */
727   #define BT_GAP_ADV_FAST_INT_MAX_2              0x0215  /* Was 0x00f0 for 150 ms, Now 0x0215(533d) So, 533d * 0.625mS = 333.125mS */
```

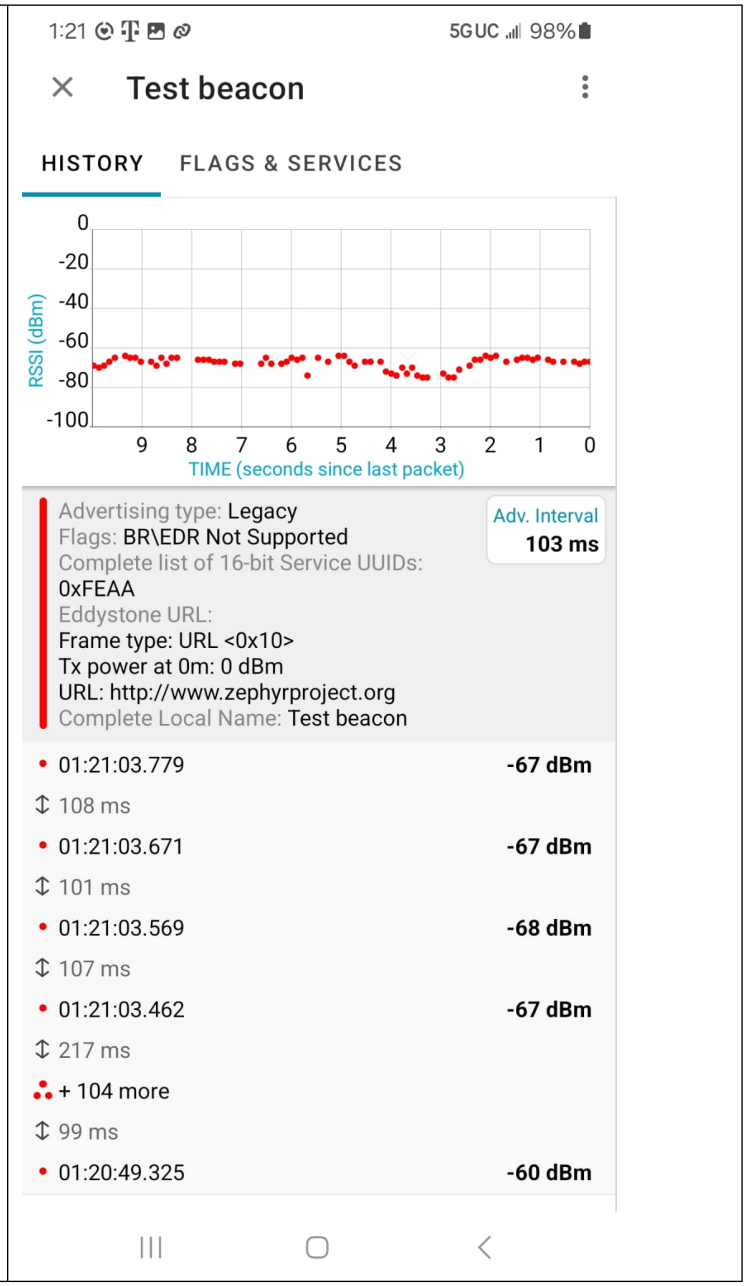
```
180   #define CONFIG_BT_DEVICE_NAME "Ricardo Test beacon"
```

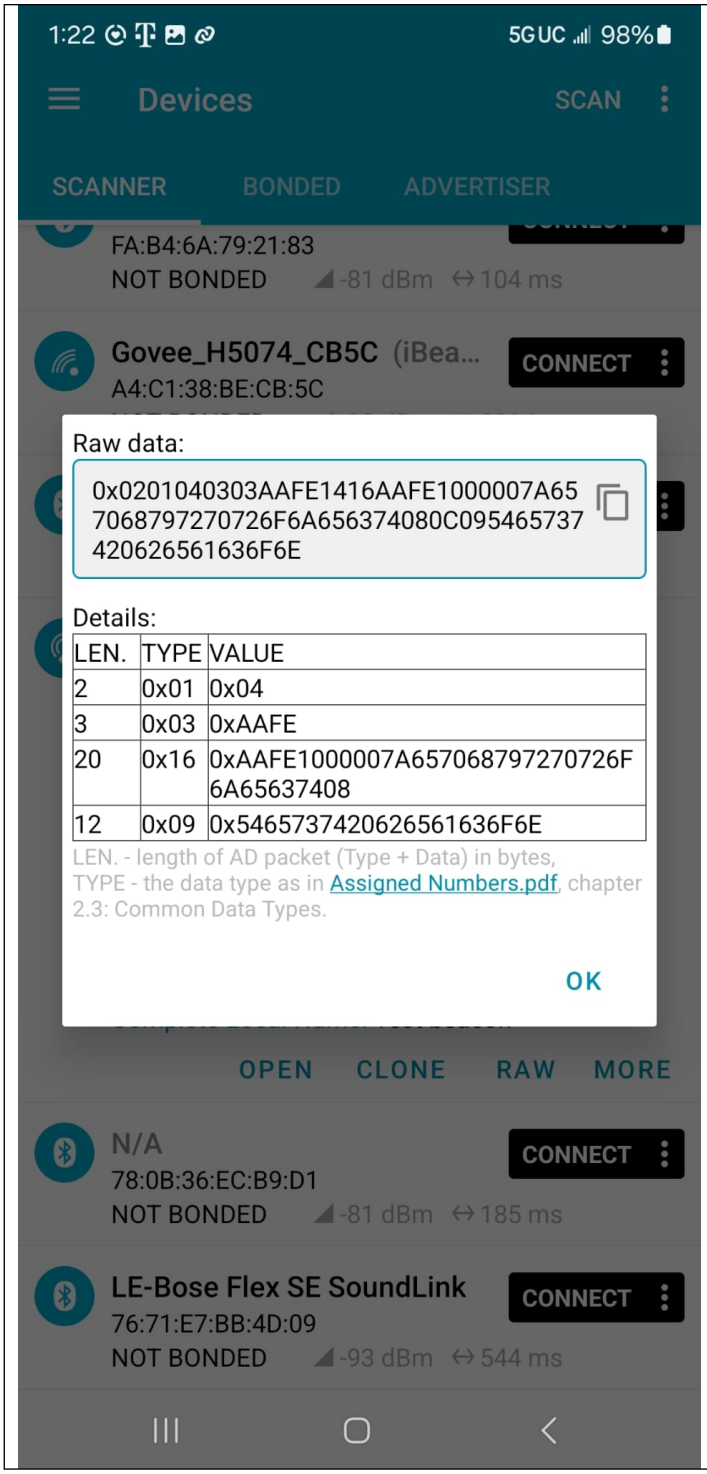

File Edit Selection View Go Run
beacon
NRF CONNECT
WELCOME
Manage toolchains
Manage SDKs
Open an existing application
Create a new application
Create a new board
Browse samples
APPLICATIONS
beacon (1 configuration | 2 contexts)
build sysbuild
beacon nRF52840 DK nRF52840
Add build configuration
BEACON build/beacon
nRF Connect SDK v2.9.0
Source files
Application
nRF Connect SDK
Generated
Config files
Kconfig
Devicetree
CMakeLists.txt
Output files
zephyr.elf 93.38 kB | 24.73 kB
zephyr.map
merged.hex 0 bytes | 94.99 kB
ACTIONS build/beacon
nRF Connect SDK Toolchain v2.9.0
Build
Debug
Flash
Devicetree Overlay file
nRF Kconfig GUI
Memory report
CONNECTED DEVICES
nrf52840dk_nrf52840.overlay
boards > nrf52840dk_nrf52840.overlay
/*
* Copyright 2022 Dronetag
*
* SPDX-License-Identifier: Apache-2.0
*/
/{
coex_gpio: coex {
compatible = "gpio-radio-coex";
grant-gpios = <&gpio1 0 (GPIO_PULL_DOWN | GPIO_ACTIVE_LOW)>;
grant-delay-us = <150>;
};
};
&radio {
coex = <&coex_gpio>;
};
PROBLEMS 12 OUTPUT DEBUG CONSOLE TERMINAL PORTS
Flash: beacon/beacon (active)
[######] 0.000s | Program file - Checking image
[##] 0.000s | Check image validity - Initialize device info
[#####] 0.000s | Check image validity - Check region 0 settings
[########] 4.466s | Check image validity - block 1 of 5
[##########] 2.444s | Check image validity - block 2 of 5
[############] 1.326s | Check image validity - block 3 of 5
[################] 0.574s | Check image validity - block 4 of 5
[##################] 0.028s | Check image validity - Finished
[############] 0.000s | Program file - Programming
[####] 0.000s | Programming image - block 1 of 4
[########] 0.876s | Programming image - block 2 of 4
[##########] 0.421s | Programming image - block 3 of 4
[################] 0.161s | Programming image - block 4 of 4
[##################] 0.000s | Programming image - Write successful
[##################] 0.654s | Program file - Done programming
[######] 0.000s | Verify file - Check image
[##] 0.000s | Check image validity - Initialize device info
[#####] 0.000s | Check image validity - Check region 0 settings
[########] 5.558s | Check image validity - block 1 of 5
[##########] 3.043s | Check image validity - block 2 of 5
[############] 1.650s | Check image validity - block 3 of 5
[################] 0.714s | Check image validity - block 4 of 5
[##################] 0.031s | Check image validity - Finished
[############] 0.000s | Verify file - Verifying
[####] 0.000s | Verifying image - block 1 of 4
[########] 0.895s | Verifying image - block 2 of 4
[##########] 0.428s | Verifying image - block 3 of 4
[################] 0.162s | Verifying image - block 4 of 4
[##################] 0.000s | Verifying image - Verify successful
[##################] 0.655s | Verify file - Done verifying
Enabling pin reset.
Applying pin reset.
-- runners.nrfjprog: Board with serial number 1050243071 flashed successfully.
* Terminal will be reused by tasks, press any key to close it.
master* 0 11 1 0 Build
Ln 17, Col 1 Tab Size: 4 UTF-8 CRLF Devicetree beacon: build/beacon

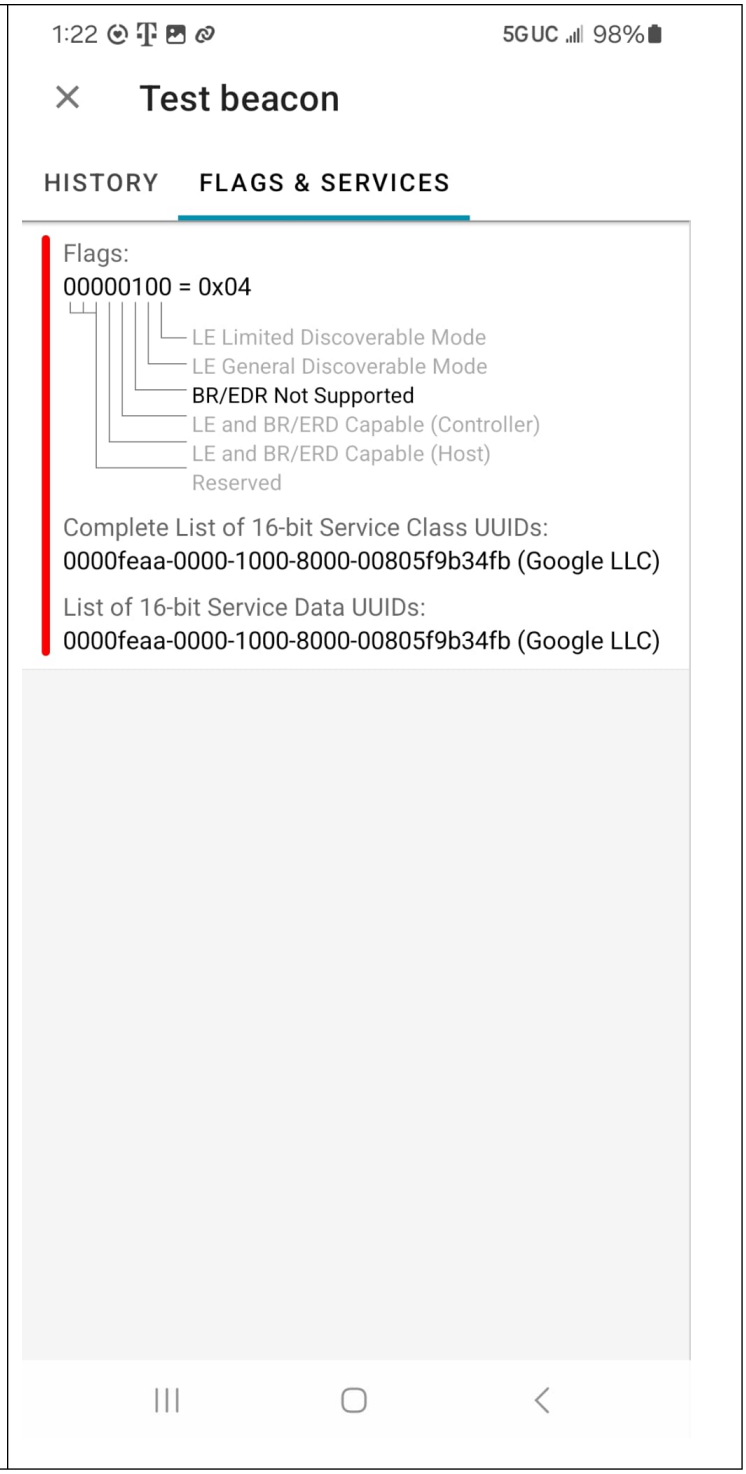
4. **Link to your updated code:**








1:22
5GUC 98%
Devices
SCAN
SCANNER
BONDED
ADVERTISER
FA:B4:6A:79:21:83
NOT BONDED
-81 dBm
104 ms
Govee_H5074_CB5C (iBea...
A4:C1:38:BE:CB:5C
CONNECT
Raw data:
0x0201040303AAFE1416AAFE1000007A65
7068797270726F6A656374080C095465737
420626561636F6E
Details:
LEN. TYPE VALUE
2 0x01 0x04
3 0x03 0xAAFE
20 0x16 0xAAFE1000007A657068797270726F
6A65637408
12 0x09 0x5465737420626561636F6E
LEN. - length of AD packet (Type + Data) in bytes,
TYPE - the data type as in Assigned Numbers.pdf, chapter
2.3: Common Data Types.
OK
OPEN
CLONE
RAW
MORE
N/A
78:0B:36:EC:B9:D1
NOT BONDED
-81 dBm
185 ms
CONNECT
LE-Bose Flex SE SoundLink
76:71:E7:BB:4D:09
NOT BONDED
-93 dBm
544 ms
CONNECT


1:22
5GUC 98%
Test beacon
HISTORY
FLAGS & SERVICES
Flags:
00000100 = 0x04
LE Limited Discoverable Mode
LE General Discoverable Mode
BR/EDR Not Supported
LE and BR/ERD Capable (Controller)
LE and BR/ERD Capable (Host)
Reserved
Complete List of 16-bit Service Class UUIDs:
0000feaa-0000-1000-8000-00805f9b34fb (Google LLC)
List of 16-bit Service Data UUIDs:
0000feaa-0000-1000-8000-00805f9b34fb (Google LLC)

I: Programming a BLE Scanner

I have programmed the dongle using 2 methods:

1. **Via Console terminal method and**

Load firmware for dongle

```
PS C:\_RLS\nRF_Sniffer_Install\nrf_sniffer_for_bluetooth_le_4.1.1\hex> nrfutil device list
id:5
ports           COM4, vcom: 0
                COM5, vcom: 1
traits          serialPorts, usb

A1B470112683F189
product         USB-DEV
ports           COM11
traits          serialPorts, usb

Found 2 supported device(s)
```

```
nrfutil pkg generate --hw-version 52 --sd-req=0x00  --application
sniffer_nrf52840dongle_nrf52840_4.1.1.hex --application-version 1 app.zip
```

C:_RLS\nRF_Sniffer_Install\nrf_sniffer_for_bluetooth_le_4.1.1\hex\ sniffer_nrf52840dongle_nrf52840_4.1.1.hex

Holding the reset button to enter into boot up mode, then program:

```
PS C:\_RLS\nRF_Sniffer_Install\nrf_sniffer_for_bluetooth_le_4.1.1\hex> nrfutil device program --firmware app.zip --traits nordicDfu
[00:00:04] ###### 100% [2/2 D84EA6CF4222] Programmed
```

> nrfutil device program –firmware app.zip –traits nordicDfu

Video used that helped to program the dongle:

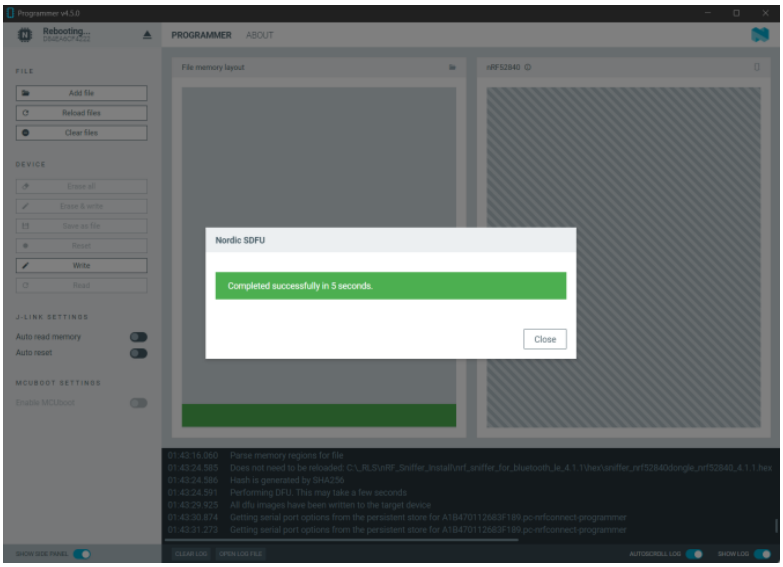
Unlocking the NRF52840 Dongle: Your 2024 Starter Guide! https://www.youtube.com/watch?v=TeBvb645NZA

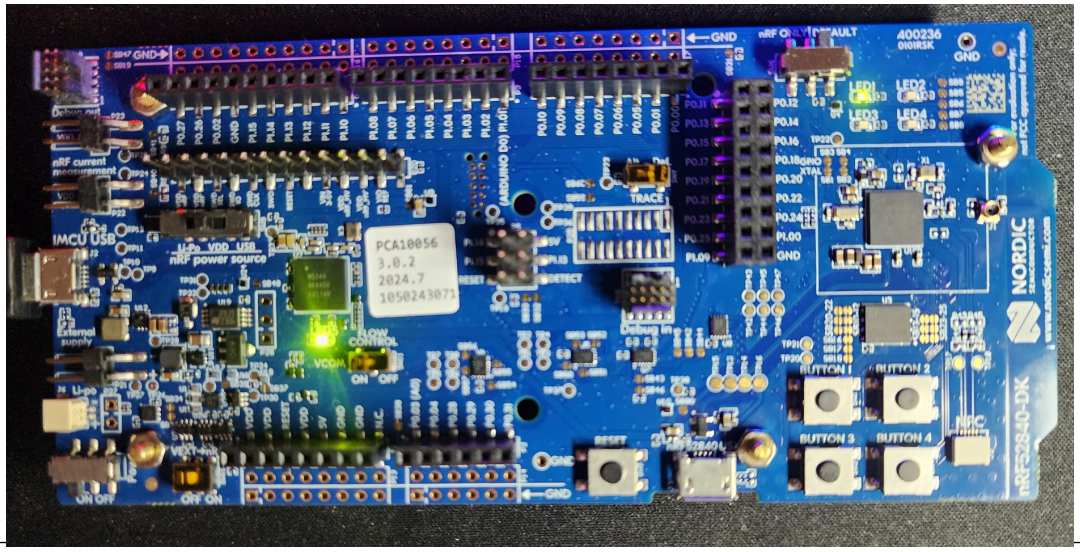
Another example of programming manually

>> nrfutil pkg generate --hw-version 52 --sd-req=0x00 --application zephyr.hex --application-version 1 app.zip

>> nrfutil device program –firmware app.zip –traits nordicDfu

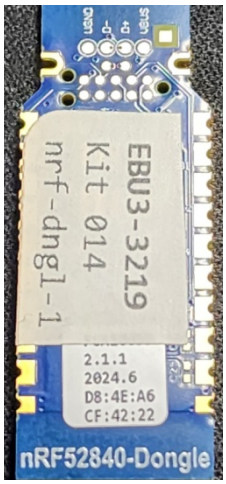
2. **Using the Programmer app**



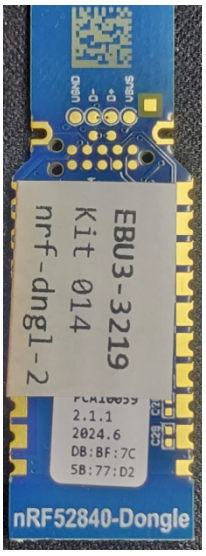


This dongle was programmed as **sniffer**: **External device connected to Wireshark, to sniff BT traffic**

WireShark filter for **nrf52840dk:**
btle.advertising_address == **fa:b4:6a:79:21:83**



btle.advertising_address == d8:4e:a6:cf:42:22



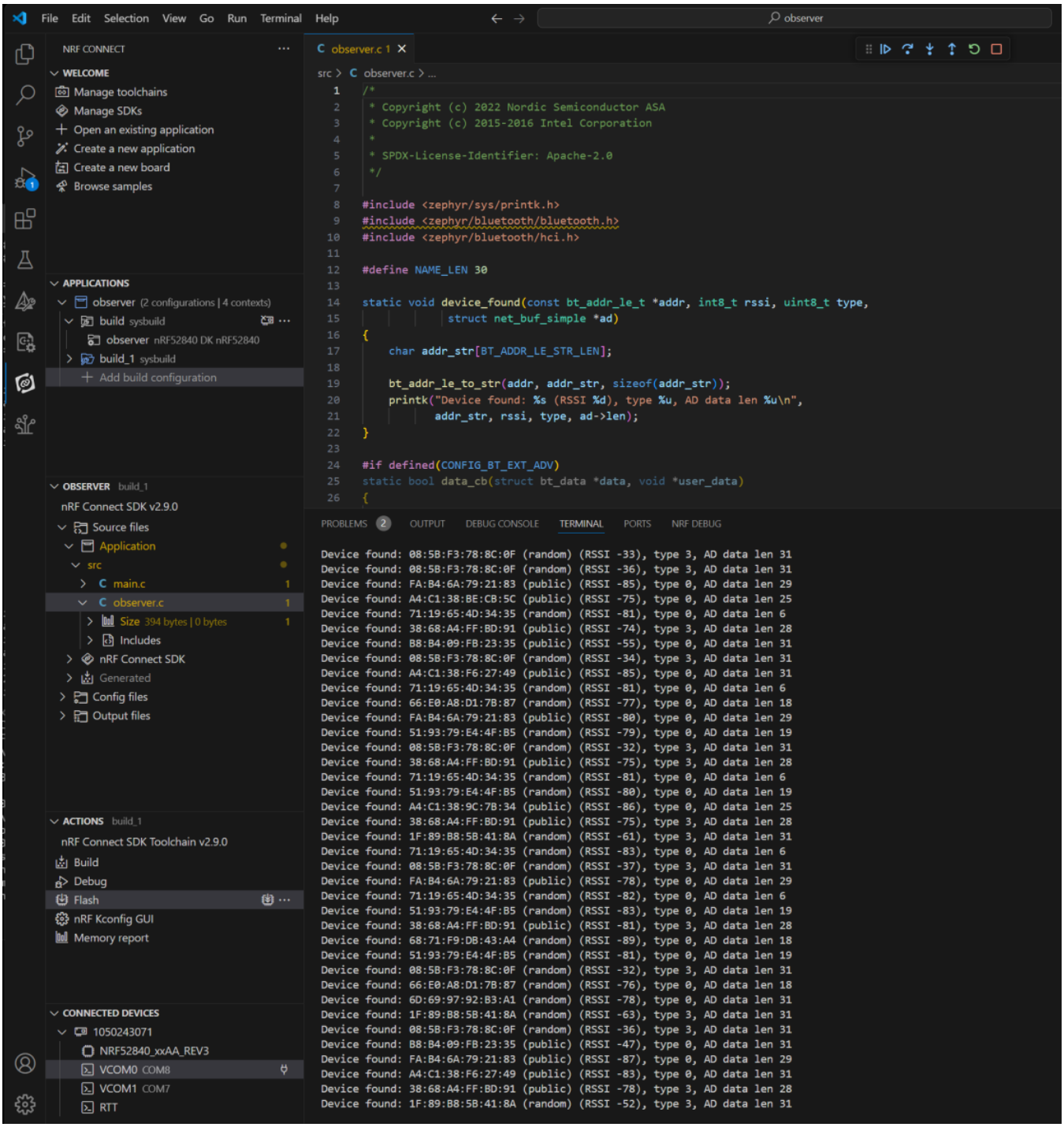
btle.advertising_address == db:bf:7c:5b:77:d2

The **nrf52840dk was programmed as Observer: It continuously scans for nearby BT devices. {prints out the DEVICE-NAME & Manufacturing-data}**

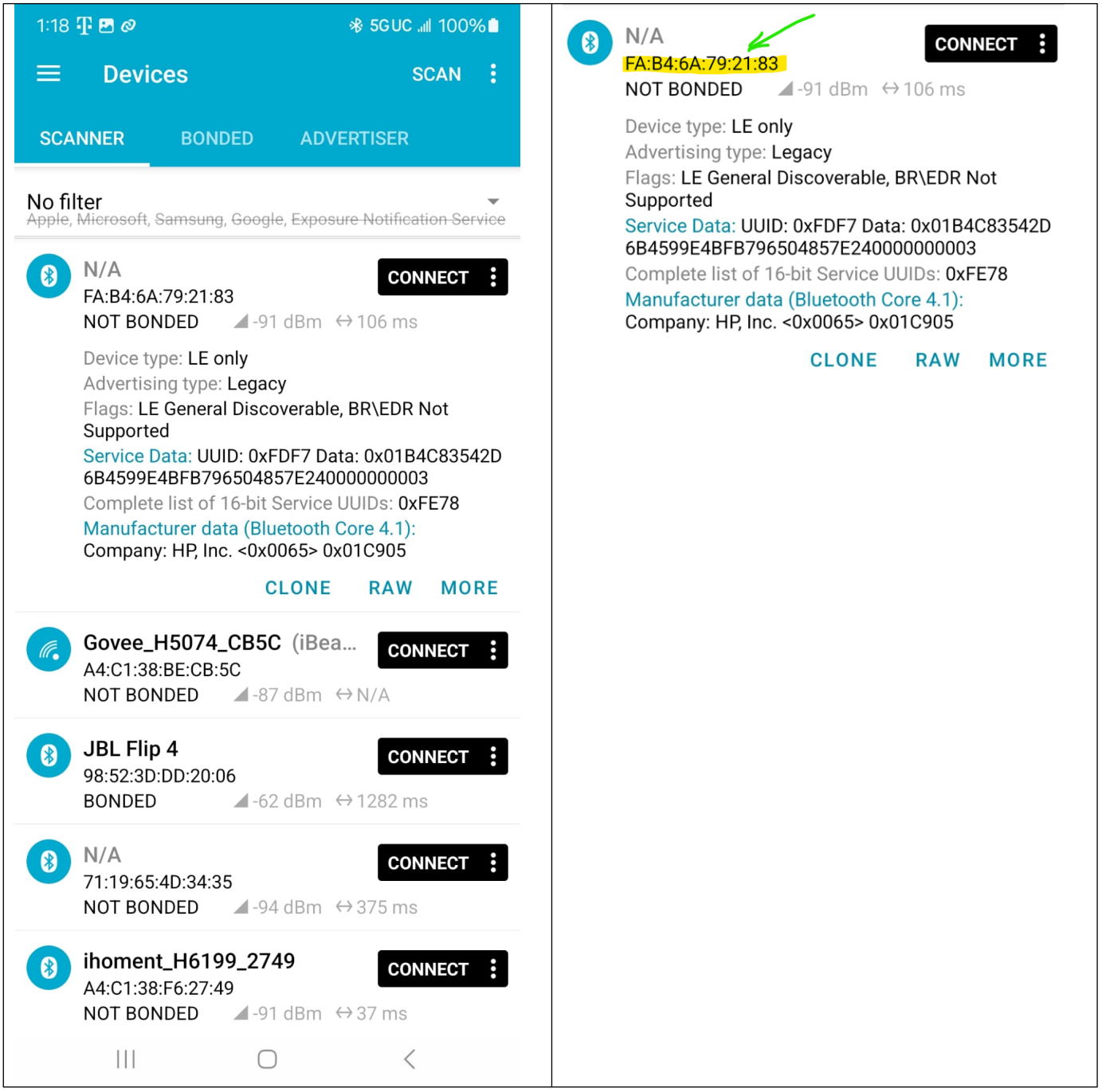
PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS NRF DEBUG

```
Device found: B8:B4:09:FB:23:35 (public) (RSSI -46), type 0, AD data len 31
Device found: 08:5B:F3:78:8C:0F (random) (RSSI -33), type 3, AD data len 31
Device found: 38:68:A4:FF:BD:91 (public) (RSSI -78), type 3, AD data len 28
Device found: 08:5B:F3:78:8C:0F (random) (RSSI -35), type 3, AD data len 31
Device found: 88:57:1D:34:AC:E4 (public) (RSSI -87), type 0, AD data len 31
Device found: 70:B6:E4:89:87:8B (random) (RSSI -64), type 0, AD data len 31
Device found: 7C:10:A1:DA:36:1D (random) (RSSI -80), type 0, AD data len 18
Device found: 51:93:79:E4:4F:B5 (random) (RSSI -85), type 0, AD data len 19
Device found: 38:68:A4:FF:BD:91 (public) (RSSI -81), type 3, AD data len 28
Device found: 68:71:F9:DB:43:A4 (random) (RSSI -84), type 0, AD data len 18
Device found: 22:8C:5A:4E:FA:48 (random) (RSSI -68), type 3, AD data len 31
Device found: 08:5B:F3:78:8C:0F (random) (RSSI -34), type 3, AD data len 31
Device found: 08:5B:F3:78:8C:0F (random) (RSSI -34), type 3, AD data len 31
Device found: 71:19:65:4D:34:35 (random) (RSSI -83), type 0, AD data len 6
Device found: 51:93:79:E4:4F:B5 (random) (RSSI -86), type 0, AD data len 19
Device found: 64:5D:4A:D0:27:EE (random) (RSSI -87), type 0, AD data len 18
Device found: 38:68:A4:FF:BD:91 (public) (RSSI -74), type 3, AD data len 28
Device found: 71:19:65:4D:34:35 (random) (RSSI -83), type 0, AD data len 6
Device found: B8:B4:09:FB:23:35 (public) (RSSI -45), type 0, AD data len 31
Device found: B8:B4:09:FB:23:35 (public) (RSSI -44), type 0, AD data len 31
Device found: 08:5B:F3:78:8C:0F (random) (RSSI -35), type 3, AD data len 31
Device found: CF:3F:D6:70:8F:E2 (random) (RSSI -85), type 3, AD data len 8
Device found: 4B:55:7E:27:9E:42 (random) (RSSI -86), type 3, AD data len 17
Device found: 71:19:65:4D:34:35 (random) (RSSI -82), type 0, AD data len 6
Device found: 08:5B:F3:78:8C:0F (random) (RSSI -36), type 3, AD data len 31
Device found: 68:71:F9:DB:43:A4 (random) (RSSI -85), type 0, AD data len 18
Device found: 70:B6:E4:89:87:8B (random) (RSSI -70), type 0, AD data len 31
Device found: 88:57:1D:34:AC:E4 (public) (RSSI -86), type 0, AD data len 31
Device found: 7C:10:A1:DA:36:1D (random) (RSSI -76), type 0, AD data len 18
Device found: A4:C1:38:BE:CB:5C (public) (RSSI -72), type 0, AD data len 25
Device found: 38:68:A4:FF:BD:91 (public) (RSSI -78), type 3, AD data len 28
Device found: 98:52:3D:DD:20:06 (public) (RSSI -48), type 0, AD data len 25
Device found: 43:E8:1D:96:00:56 (random) (RSSI -85), type 2, AD data len 31
Device found: 08:5B:F3:78:8C:0F (random) (RSSI -36), type 3, AD data len 31
Device found: 22:8C:5A:4E:FA:48 (random) (RSSI -62), type 3, AD data len 31
Device found: 38:68:A4:FF:BD:91 (public) (RSSI -82), type 3, AD data len 28
Device found: 08:5B:F3:78:8C:0F (random) (RSSI -36), type 3, AD data len 31
Device found: 68:71:F9:DB:43:A4 (random) (RSSI -85), type 0, AD data len 18
```

1. **Show evidence you were able to implement a scanner with the nRF52840DK:**



2. **Show evidence <u>your device</u> asked for a scan response**



3. **Show evidence another device another device sent a scan response:**

```
evice found: 32:8C:A1:54:BD:F5 (random) (RSSI -33), type 3, AD data len 31
evice found: 76:A2:54:51:B7:10 (random) (RSSI -77), type 0, AD data len 18
evice found: FA:B4:6A:79:21:83 (public) (RSSI -82), type 0, AD data len 29
evice found: 0F:74:2E:0E:EE:10 (random) (RSSI -67), type 3, AD data len 31
evice found: 38:68:A4:FF:BD:91 (public) (RSSI -73), type 3, AD data len 28
evice found: 32:8C:A1:54:BD:F5 (random) (RSSI -33), type 3, AD data len 31
evice found: 32:8C:A1:54:BD:F5 (random) (RSSI -35), type 3, AD data len 31
evice found: 88:57:1D:34:AC:E4 (public) (RSSI -85), type 0, AD data len 31
evice found: FA:B4:6A:79:21:83 (public) (RSSI -77), type 0, AD data len 29
evice found: B8:B4:09:FB:23:35 (public) (RSSI -46), type 0, AD data len 31
evice found: 5B:B4:2F:0B:F3:C7 (random) (RSSI -65), type 0, AD data len 31
evice found: 38:68:A4:FF:BD:91 (public) (RSSI -73), type 3, AD data len 28
evice found: 76:A2:54:51:B7:10 (random) (RSSI -77), type 0, AD data len 18
evice found: 61:40:FD:DE:DD:80 (random) (RSSI -83), type 2, AD data len 31
evice found: 32:8C:A1:54:BD:F5 (random) (RSSI -34), type 3, AD data len 31
evice found: 61:40:FD:DE:DD:80 (random) (RSSI -84), type 2, AD data len 31
evice found: 38:68:A4:FF:BD:91 (public) (RSSI -78), type 3, AD data len 28
evice found: 32:8C:A1:54:BD:F5 (random) (RSSI -36), type 3, AD data len 31
evice found: FA:B4:6A:79:21:83 (public) (RSSI -78), type 0, AD data len 29
evice found: B8:B4:09:FB:23:35 (public) (RSSI -45), type 0, AD data len 31
evice found: 61:40:FD:DE:DD:80 (random) (RSSI -85), type 2, AD data len 31
evice found: 5B:B4:2F:0B:F3:C7 (random) (RSSI -64), type 0, AD data len 31
evice found: 76:A2:54:51:B7:10 (random) (RSSI -78), type 0, AD data len 18
evice found: 32:8C:A1:54:BD:F5 (random) (RSSI -34), type 3, AD data len 31
evice found: 61:40:FD:DE:DD:80 (random) (RSSI -84), type 2, AD data len 31
evice found: 5B:B4:2F:0B:F3:C7 (random) (RSSI -69), type 0, AD data len 31
evice found: 38:68:A4:FF:BD:91 (public) (RSSI -74), type 3, AD data len 28
evice found: 88:57:1D:34:AC:E4 (public) (RSSI -85), type 0, AD data len 31
evice found: 32:8C:A1:54:BD:F5 (random) (RSSI -34), type 3, AD data len 31
evice found: 32:8C:A1:54:BD:F5 (random) (RSSI -38), type 3, AD data len 31
evice found: 38:68:A4:FF:BD:91 (public) (RSSI -74), type 3, AD data len 28
evice found: FA:B4:6A:79:21:83 (public) (RSSI -78), type 0, AD data len 29
evice found: CE:53:B0:3F:09:49 (random) (RSSI -91), type 3, AD data len 8
evice found: 76:A2:54:51:B7:10 (random) (RSSI -78), type 0, AD data len 18
evice found: 5B:B4:2F:0B:F3:C7 (random) (RSSI -69), type 0, AD data len 31
evice found: A4:C1:38:F6:27:49 (public) (RSSI -90), type 0, AD data len 31
evice found: 38:68:A4:FF:BD:91 (public) (RSSI -75), type 3, AD data len 28
```

A **BLE scan response** is the packet that **is sent by the advertising device (peripheral)** upon the reception of scanning requests (i.e. yes, it is a response to a device scan). ***The scan response usually has more data than the advertising packets. In other words, central devices send scan requests to the advertising device in order to get additional user data through the scan response***.

4. **Link to your updated code:**

- Print something special when you receive an advertisement from your own advertiser.

I modified the code, so when found MAC ADDRESS fa:b4:6a:79:21:83 will print out messsage

```
static void device_found(const bt_addr_le_t *addr, int8_t rssi, uint8_t type,
                         struct net_buf_simple *ad)
{
    char addr_str[BT_ADDR_LE_STR_LEN];
    char addr_str2[BT_ADDR_LE_STR_LEN] = "FA:B4:6A:79:21:83";

    bt_addr_le_to_str(addr, addr_str, sizeof(addr_str));
    printk("Device found: %s (RSSI %d), type %u, AD data len %u\n",
           addr_str, rssi, type, ad->len);

    bool res = true;
    for(int i=0;i<16;i++)
    {
        if(addr_str[i]!=addr_str2[i])
        {
            res = false;
        }
    }

    if (res)
        printk(">>> advertisement received from my own advertiser: { %s }\n\n\n",addr_str);

}
```

```
PROBLEMS 1   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   NRF DEBUG

Device found: 4D:DC:C8:70:5C:98 (random) (RSSI -61), type 0, AD data len 18
Device found: 66:2A:AC:C1:8E:C4 (random) (RSSI -47), type 0, AD data len 6
Device found: B8:B4:09:FB:23:35 (public) (RSSI -50), type 0, AD data len 31
Device found: B8:B4:09:FB:23:35 (public) (RSSI -50), type 0, AD data len 31
Device found: FA:B4:6A:79:21:83 (public) (RSSI -80), type 0, AD data len 29
>>> advertisement received from my own advertiser: { FA:B4:6A:79:21:83 (public) }


Device found: E2:46:4B:9F:BC:D2 (random) (RSSI -62), type 3, AD data len 8
Device found: 66:2A:AC:C1:8E:C4 (random) (RSSI -48), type 0, AD data len 6
Device found: 54:AF:6B:DF:99:36 (random) (RSSI -76), type 0, AD data len 31
Device found: 39:D9:7E:11:54:98 (random) (RSSI -37), type 3, AD data len 31
Device found: 66:2A:AC:C1:8E:C4 (random) (RSSI -48), type 0, AD data len 6
Device found: 39:D9:7E:11:54:98 (random) (RSSI -33), type 3, AD data len 31
Device found: 11:8D:42:BD:57:D0 (random) (RSSI -69), type 0, AD data len 29
Device found: 41:90:C8:15:18:82 (random) (RSSI -46), type 0, AD data len 18
Device found: 4D:DC:C8:70:5C:98 (random) (RSSI -61), type 0, AD data len 18
Device found: 66:2A:AC:C1:8E:C4 (random) (RSSI -50), type 0, AD data len 6
Device found: 39:D9:7E:11:54:98 (random) (RSSI -34), type 3, AD data len 31
Device found: 4D:DC:C8:70:5C:98 (random) (RSSI -54), type 0, AD data len 18
Device found: 54:AF:6B:DF:99:36 (random) (RSSI -75), type 0, AD data len 31
Device found: 38:68:A4:FF:BD:91 (public) (RSSI -74), type 3, AD data len 28
Device found: 39:D9:7E:11:54:98 (random) (RSSI -36), type 3, AD data len 31
Device found: B8:B4:09:FB:23:35 (public) (RSSI -41), type 0, AD data len 31
Device found: 41:90:C8:15:18:82 (random) (RSSI -46), type 0, AD data len 18
Device found: 54:AF:6B:DF:99:36 (random) (RSSI -74), type 0, AD data len 31
Device found: 39:D9:7E:11:54:98 (random) (RSSI -33), type 3, AD data len 31
Device found: 4D:DC:C8:70:5C:98 (random) (RSSI -54), type 0, AD data len 18
Device found: 11:8D:42:BD:57:D0 (random) (RSSI -77), type 0, AD data len 29
Device found: 39:D9:7E:11:54:98 (random) (RSSI -36), type 3, AD data len 31
Device found: FA:B4:6A:79:21:83 (public) (RSSI -81), type 0, AD data len 29
>>> advertisement received from my own advertiser: { FA:B4:6A:79:21:83 (public) }


Device found: 5F:13:5F:C2:32:D6 (random) (RSSI -81), type 0, AD data len 18
Device found: 66:2A:AC:C1:8E:C4 (random) (RSSI -48), type 0, AD data len 6
Device found: B8:B4:09:FB:23:35 (public) (RSSI -45), type 0, AD data len 31
Device found: 11:8D:42:BD:57:D0 (random) (RSSI -78), type 0, AD data len 29
Device found: 66:2A:AC:C1:8E:C4 (random) (RSSI -48), type 0, AD data len 6
Device found: 39:D9:7E:11:54:98 (random) (RSSI -33), type 3, AD data len 31
Device found: 39:D9:7E:11:54:98 (random) (RSSI -33), type 3, AD data len 31
```

- Filter which device information prints based on RSSI.
    - Pick whatever RSSI value you think makes sense, and only print data from devices with an RSSI value greater than that (RSSI is negative, so smaller magnitude is greater signal strength received).

With the following code change the scanner will list only found devices with power equal or greater to -60 RSSI

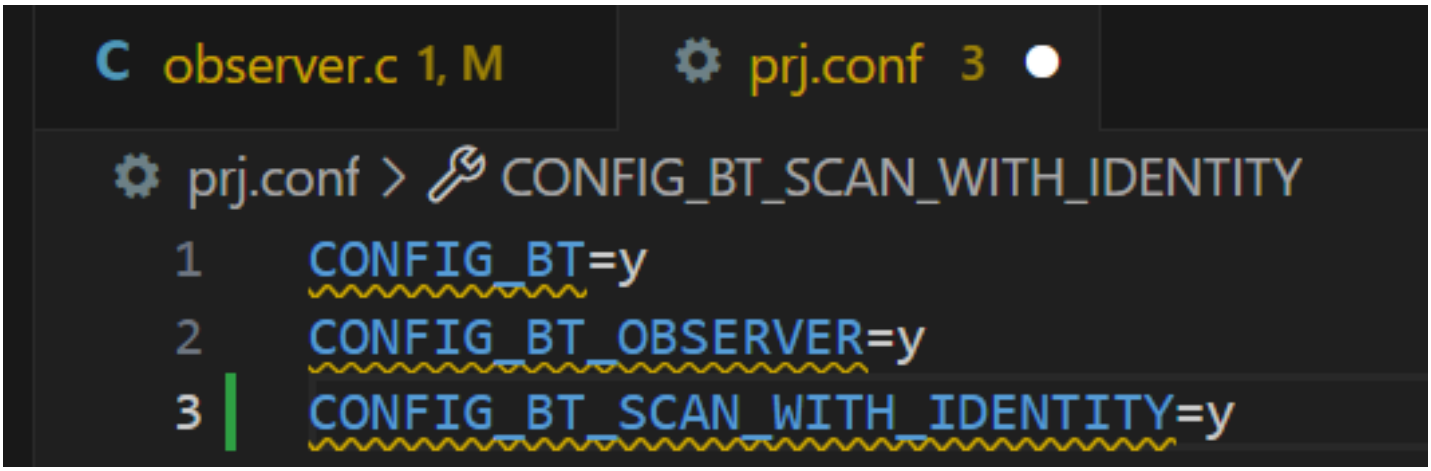
```
if (rssi >= -60){
    printk("Device found: %s (RSSI %d), type %u, AD data len %u\n",
        addr_str, rssi, type, ad->len);

    bool res = true;
    for(int i=0;i<16;i++)
    {
        if(addr_str[i]!=addr_str2[i])
            res = false;
    }
    if (res)
        printk(">>> advertisement received from my own advertiser: { %s }\n\n\n",addr_str);
}
```

```
15                  struct net_buf_simple *ad)
18      char addr_str2[BT_ADDR_LE_STR_LEN] = "FA:B4:6A:79:21:83";
19
20      bt_addr_le_to_str(addr, addr_str, sizeof(addr_str));
21
22      if (rssi >= -60){
23          printk("Device found: %s (RSSI %d), type %u, AD data len %u\n",
24              addr_str, rssi, type, ad->len);
25
26          bool res = true;
27          for(int i=0;i<16;i++)
28          {
29              if(addr_str[i]!=addr_str2[i])
30                  res = false;
31          }
32          if (res)
33              printk(">>> advertisement received from my own advertiser: { %s }\n\n\n",addr_str);
34      }
35
36
37  }
38
39  #if defined(CONFIG_BT_EXT_ADV)
40  static bool data_cb(struct bt_data *data, void *user_data)
```

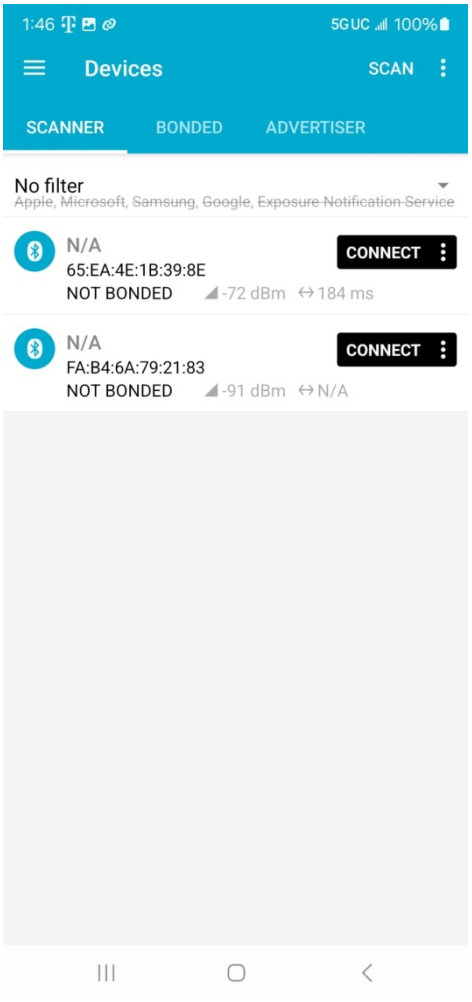
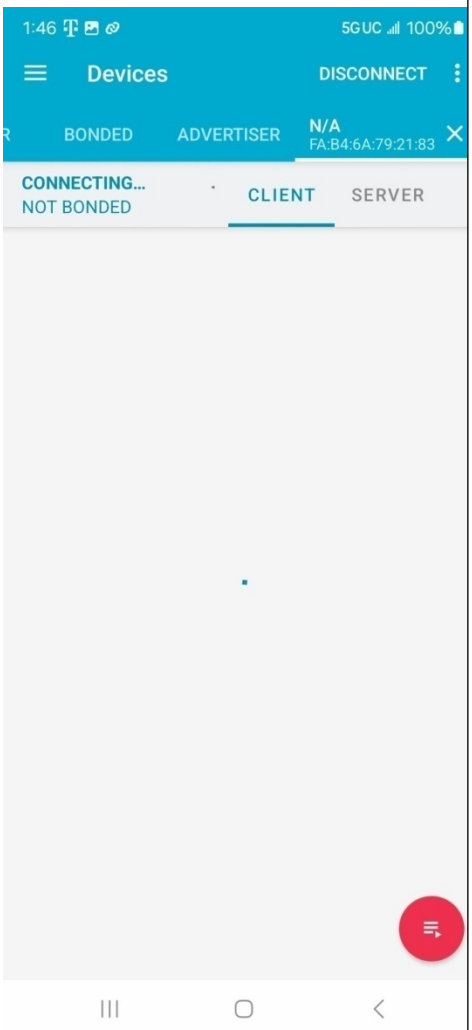
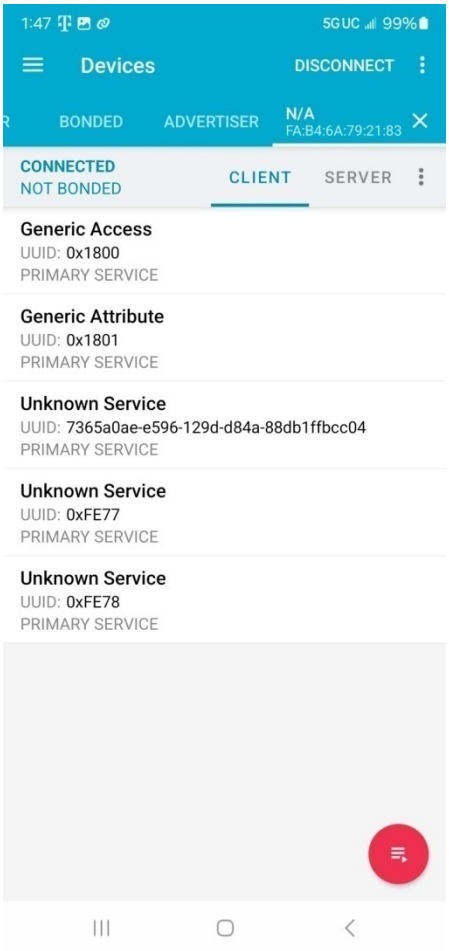
```
PROBLEMS 1  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  NRF DEBUG

Device found: 1B:F9:1F:D7:BC:5A (random) (RSSI -39), type 3, AD data len 31
Device found: 67:E6:A5:34:2C:7A (random) (RSSI -47), type 0, AD data len 18
Device found: 6B:E3:B5:95:12:BA (random) (RSSI -56), type 0, AD data len 18
Device found: 66:2A:AC:C1:8E:C4 (random) (RSSI -47), type 0, AD data len 6
Device found: 1B:F9:1F:D7:BC:5A (random) (RSSI -35), type 3, AD data len 31
Device found: 66:2A:AC:C1:8E:C4 (random) (RSSI -46), type 0, AD data len 6
Device found: 1B:F9:1F:D7:BC:5A (random) (RSSI -33), type 3, AD data len 31
Device found: 66:2A:AC:C1:8E:C4 (random) (RSSI -47), type 0, AD data len 6
Device found: 6B:E3:B5:95:12:BA (random) (RSSI -55), type 0, AD data len 18
Device found: B8:B4:09:FB:23:35 (public) (RSSI -49), type 0, AD data len 31
Device found: 67:E6:A5:34:2C:7A (random) (RSSI -46), type 0, AD data len 18
Device found: 1B:F9:1F:D7:BC:5A (random) (RSSI -36), type 3, AD data len 31
Device found: 6B:E3:B5:95:12:BA (random) (RSSI -48), type 0, AD data len 18
Device found: 1B:F9:1F:D7:BC:5A (random) (RSSI -33), type 3, AD data len 31
Device found: B8:B4:09:FB:23:35 (public) (RSSI -47), type 0, AD data len 31
Device found: 67:E6:A5:34:2C:7A (random) (RSSI -46), type 0, AD data len 18
Device found: 6B:E3:B5:95:12:BA (random) (RSSI -48), type 0, AD data len 18
Device found: 66:2A:AC:C1:8E:C4 (random) (RSSI -45), type 0, AD data len 6
Device found: 1B:F9:1F:D7:BC:5A (random) (RSSI -33), type 3, AD data len 31
Device found: 1B:F9:1F:D7:BC:5A (random) (RSSI -38), type 3, AD data len 31
Device found: 67:E6:A5:34:2C:7A (random) (RSSI -54), type 0, AD data len 18
Device found: 66:2A:AC:C1:8E:C4 (random) (RSSI -44), type 0, AD data len 6
Device found: B8:B4:09:FB:23:35 (public) (RSSI -47), type 0, AD data len 31
Device found: 66:2A:AC:C1:8E:C4 (random) (RSSI -45), type 0, AD data len 6
Device found: 6B:E3:B5:95:12:BA (random) (RSSI -48), type 0, AD data len 18
Device found: 1B:F9:1F:D7:BC:5A (random) (RSSI -33), type 3, AD data len 31
Device found: 66:2A:AC:C1:8E:C4 (random) (RSSI -45), type 0, AD data len 6
Device found: 67:E6:A5:34:2C:7A (random) (RSSI -56), type 0, AD data len 18
Device found: 6B:E3:B5:95:12:BA (random) (RSSI -49), type 0, AD data len 18
Device found: B8:B4:09:FB:23:35 (public) (RSSI -57), type 0, AD data len 31
Device found: 1B:F9:1F:D7:BC:5A (random) (RSSI -36), type 3, AD data len 31
Device found: 1B:F9:1F:D7:BC:5A (random) (RSSI -34), type 3, AD data len 31
Device found: 67:E6:A5:34:2C:7A (random) (RSSI -56), type 0, AD data len 18
```

Next try to use Scan Requests and Scan Responses.

- Enable Scan Requests for your scanner. In BLE terms, this is known as “active scanning” and is a configuration you can apply at setup time. Go check the API for the parameter.
    - To get the scan requests to use your actual BLE address, you will also need to add `CONFIG_BT_SCAN_WITH_IDENTITY=y` to the prj.conf file.
    - You can edit the file by choosing in the NRF CONNECT side panel: “Config files->Kconfig->prj.conf”

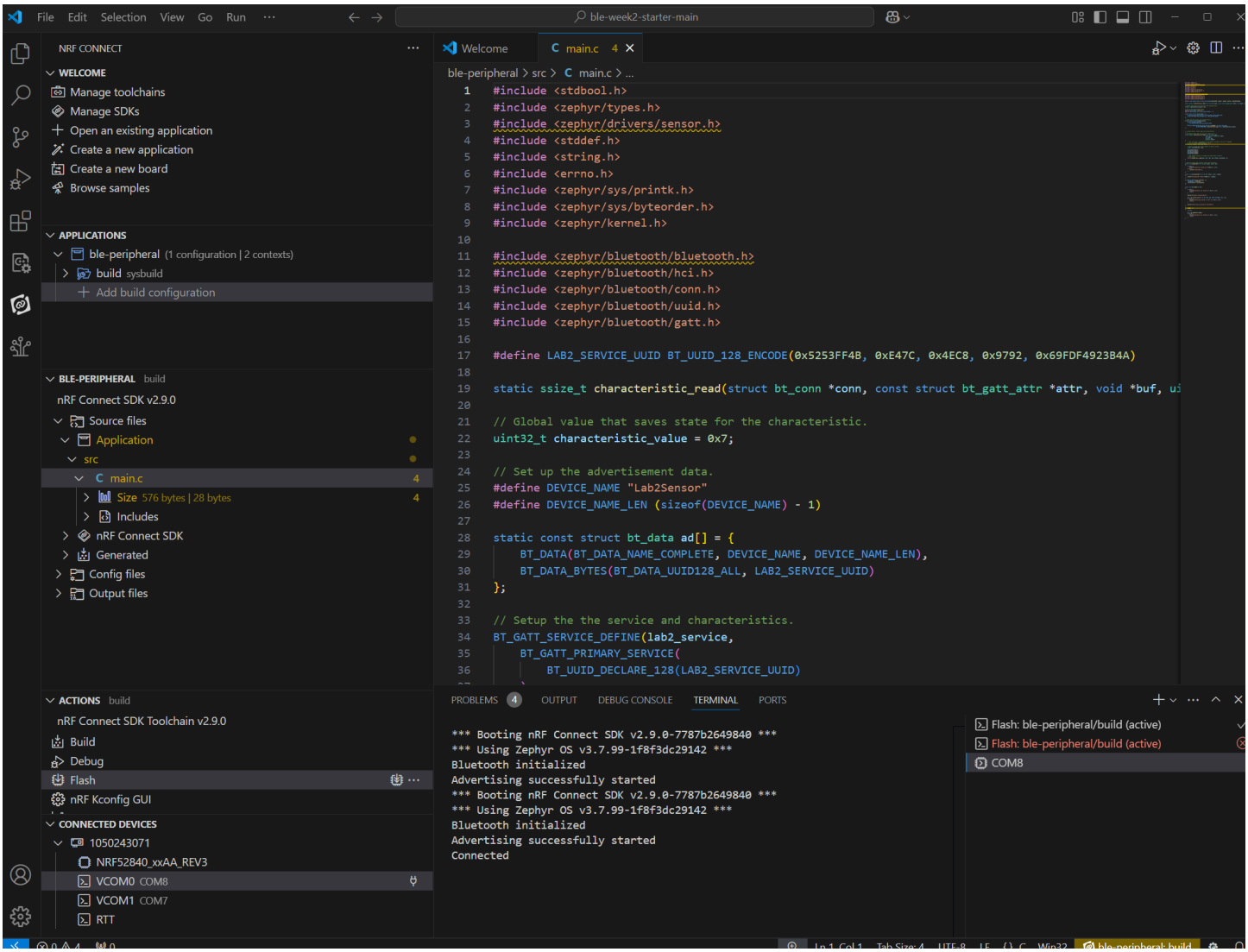


26546 238.198265	fa:b4:6a:79:21:83	Broadcast	LE LL	61	ADV_IND
26568 238.408873	61:20:3d:f5:23:d2	fa:b4:6a:79:21:…	LE LL	38	SCAN_REQ
26582 238.515675	fa:b4:6a:79:21:83	Broadcast	LE LL	61	ADV_IND
26593 238.621186	63:a9:a2:35:78:7d	fa:b4:6a:79:21:…	LE LL	38	SCAN_REQ
26668 239.346937	fa:b4:6a:79:21:83	Broadcast	LE LL	61	ADV_NONCON
26669 239.348129	fa:b4:6a:79:21:83	Broadcast	LE LL	61	ADV_IND[Ma
26681 239.454535	fa:b4:6a:79:21:83	Broadcast	LE LL	61	ADV_IND
26690 239.555692	fa:b4:6a:79:21:83	Broadcast	LE LL	61	ADV_IND[Ma
26699 239.658818	fa:b4:6a:79:21:83	Broadcast	LE LL	61	ADV_IND
26700 239.660009	fa:b4:6a:79:21:83	Broadcast	LE LL	61	ADV_IND[Ma
26730 239.980698	fa:b4:6a:79:21:83	Broadcast	LE LL	61	ADV_IND[Ma
26731 239.981419	fa:b4:6a:79:21:83	Broadcast	LE LL	61	ADV_IND
26745 240.087670	fa:b4:6a:79:21:83	Broadcast	LE LL	61	ADV_IND
26763 240.295798	fa:b4:6a:79:21:83	Broadcast	LE LL	61	ADV_IND
26788 240.505707	fa:b4:6a:79:21:83	Broadcast	LE LL	61	ADV_IND
26833 241.025695	a3:a9:a2:35:78:7d	fa:b4:6a:79:21:…	LE LL	38	SCAN_REQ
26854 241.233312	fa:b4:6a:79:21:83	Broadcast	LE LL	61	ADV_IND
26906 241.761975	fa:b4:6a:79:21:83	Broadcast	LE LL	61	ADV_IND
26943 242.185826	fa:b4:6a:79:21:83	Broadcast	LE LL	61	ADV_IND
26996 242.698333	fa:b4:6a:79:21:83	Broadcast	LE LL	61	ADV_IND
26997 242.699563	63:a9:a2:35:78:7d	fa:b4:6a:79:21:…	LE LL	38	SCAN_REQ
27073 243.436449	63:a9:a2:35:78:7d	fa:b4:6a:79:21:…	LE LL	38	SCAN_REQ

Step1: Using the smartphone, scanned and found dk board (**fa:b4:6a:79:21:83**). Clicked connect	Step2: Connecting	Step3: Now connected to **fa:b4:6a:79:21:83**), Here we can see UUID services offered
		

J: BLE Peripheral

Loaded **“ble-peripheral”** from repository
https://github.com/ucsd-wxiot-wi25/ble-week2-starter ble-week2-starter



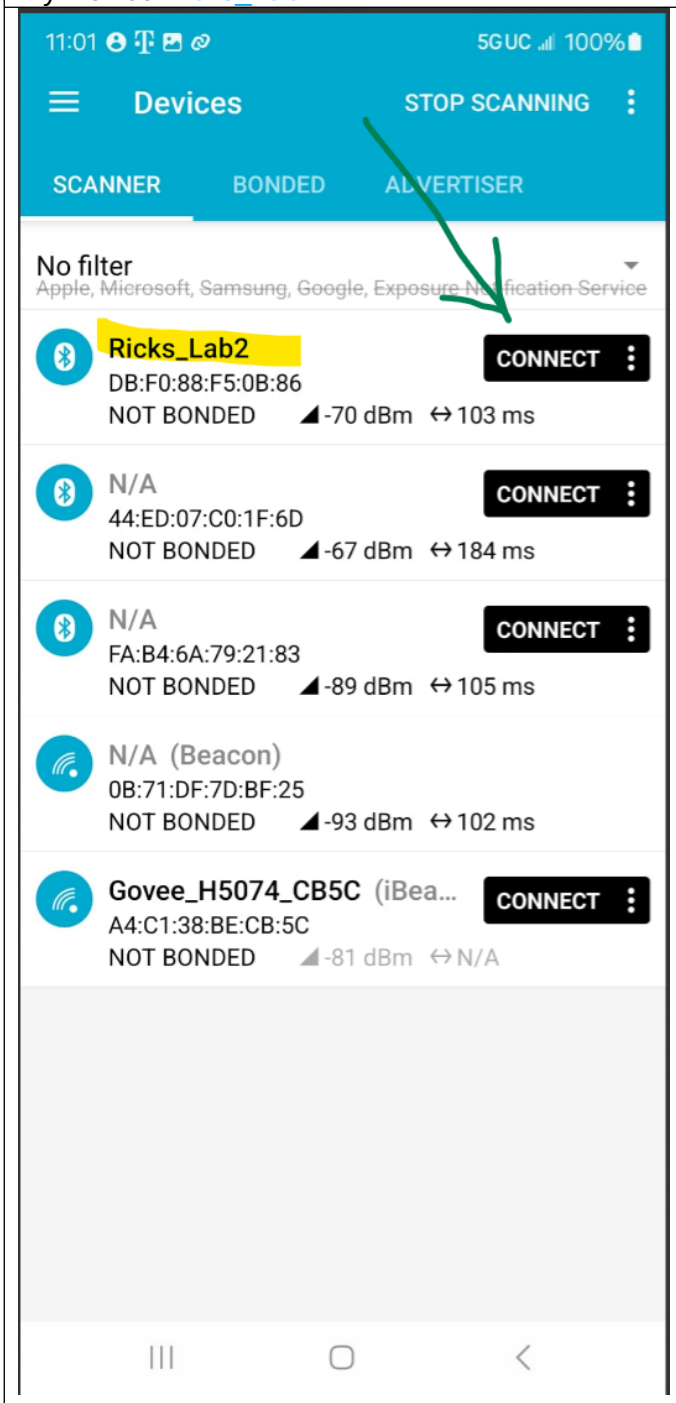
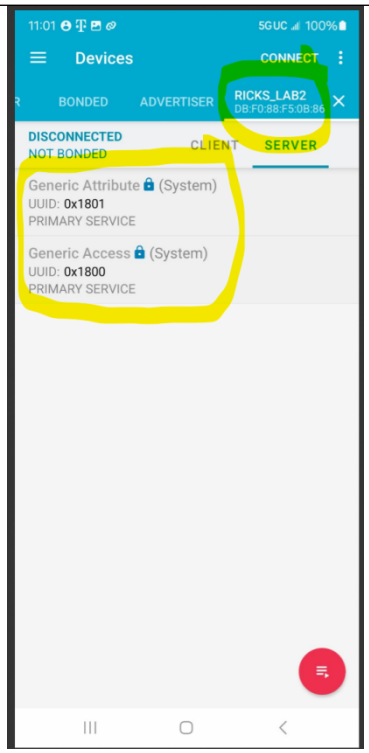
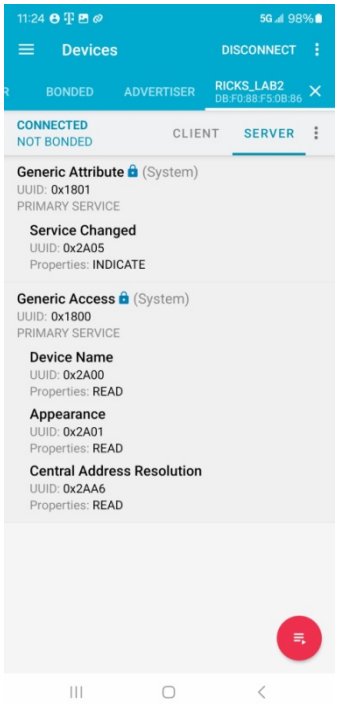
1. **What is the initial value of the characteristic in the provided code?**

   Renamed device name:

```
// Set up the advertisement data.
#define DEVICE_NAME "Ricks_Lab2"
```

2. **Link to your updated code:**

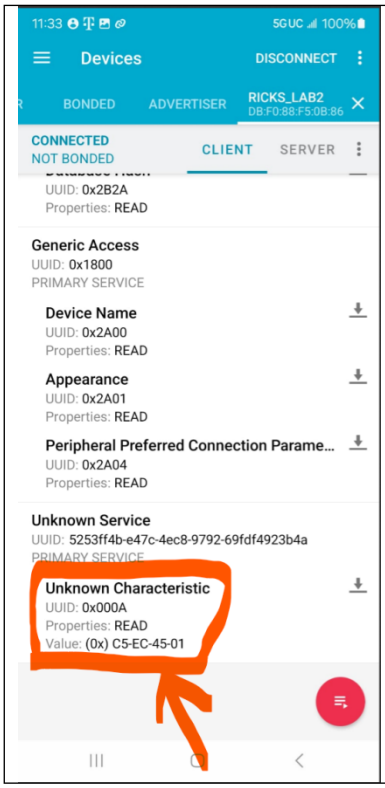
C:_RLS\ble-week2-starter-main\ble-week2-starter-main\ble-peripheral

#1: Scanning in smartphone (nRF for Mobile), finds my Device Ricks_Lab2"	#2 Connected Device
	 

```
PROBLEMS 4  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  NRF DEBUG

*** Booting nRF Connect SDK v2.9.0-7787b2649840 ***
*** Using Zephyr OS v3.7.99-1f8f3dc29142 ***
Bluetooth initialized
Advertising successfully started
*** Booting nRF Connect SDK v2.9.0-7787b2649840 ***
*** Using Zephyr OS v3.7.99-1f8f3dc29142 ***
Bluetooth initialized
Advertising successfully started
```

powershell
Flash: ble-peripheral/build (active)
COM8
COM7

Original code



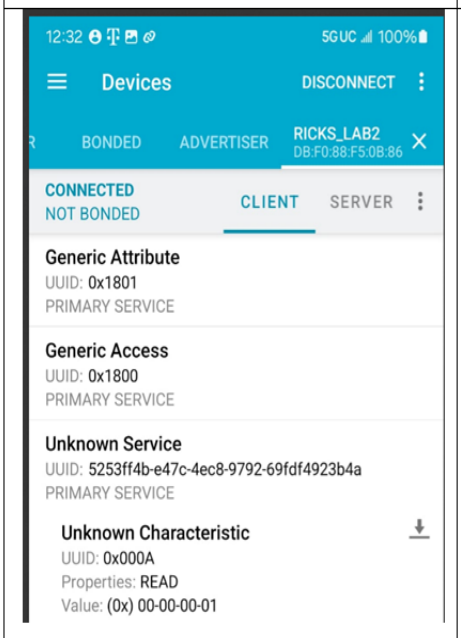

```c
// Callback when a client reads the characteristic.
//
// Documented under name "bt_gatt_attr_read_chrc()"
static ssize_t characteristic_read(struct bt_conn *conn,
                                   const struct bt_gatt_attr *attr,
                                   void *buf,
                                   uint16_t len,
                                   uint16_t offset)
{
    // The `user_data` corresponds to the pointer provided as the last "argument"
    // to the `BT_GATT_CHARACTERISTIC` macro.
    uint32_t *value = attr->user_data;

    // Need to encode data into a buffer to send to client.
    uint8_t out_buffer[4] = {0};

    out_buffer[0]=0xc5;
    out_buffer[1]=0xec;
    out_buffer[2]=0x45;
    out_buffer[3]=0x01;

    // User helper function to encode the output data to send to
    // the client.
    return bt_gatt_attr_read(conn, attr, buf, len, offset, out_buffer, 4);
}
```

Changed code

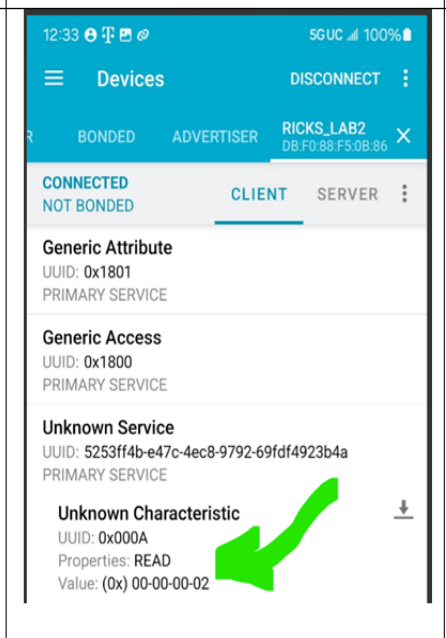
```c
// Documented under name "bt_gatt_attr_read_chrc()"
static ssize_t characteristic_read(struct bt_conn *conn,
                                   const struct bt_gatt_attr *attr,
                                   void *buf,
                                   uint16_t len,
                                   uint16_t offset)
{
    // The `user_data` corresponds to the pointer provided as the last "argument"
    // to the `BT_GATT_CHARACTERISTIC` macro.
    uint32_t *value = attr->user_data;

    static uint32_t counter = 0;
    counter++;      // Increment the counter by one

    // Need to encode data into a buffer to send to client.
    uint8_t out_buffer[4] = {0};

    out_buffer[3] = (uint8_t)(counter & 0xFF);          // Most significant byte
    out_buffer[2] = (uint8_t)((counter >> 8) & 0xFF);
    out_buffer[1] = (uint8_t)((counter >> 16) & 0xFF);
    out_buffer[0] = (uint8_t)((counter >> 24) & 0xFF); // Least significant byte

    // User helper function to encode the output data to send to
    // the client.
    return bt_gatt_attr_read(conn, attr, buf, len, offset, out_buffer, 4);
}
```

Counter testing

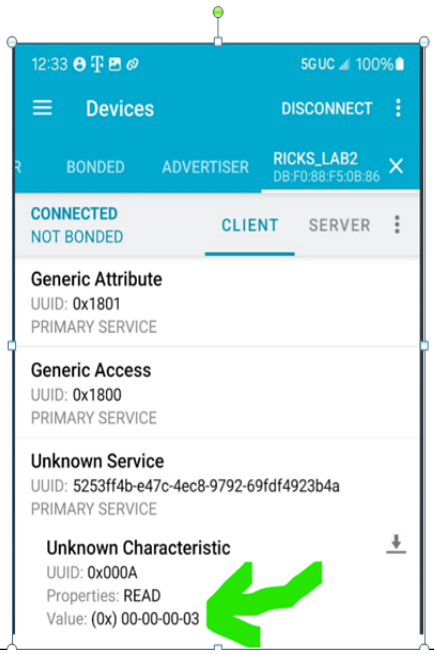
Counter value =1



Counter value =2



Counter value =3

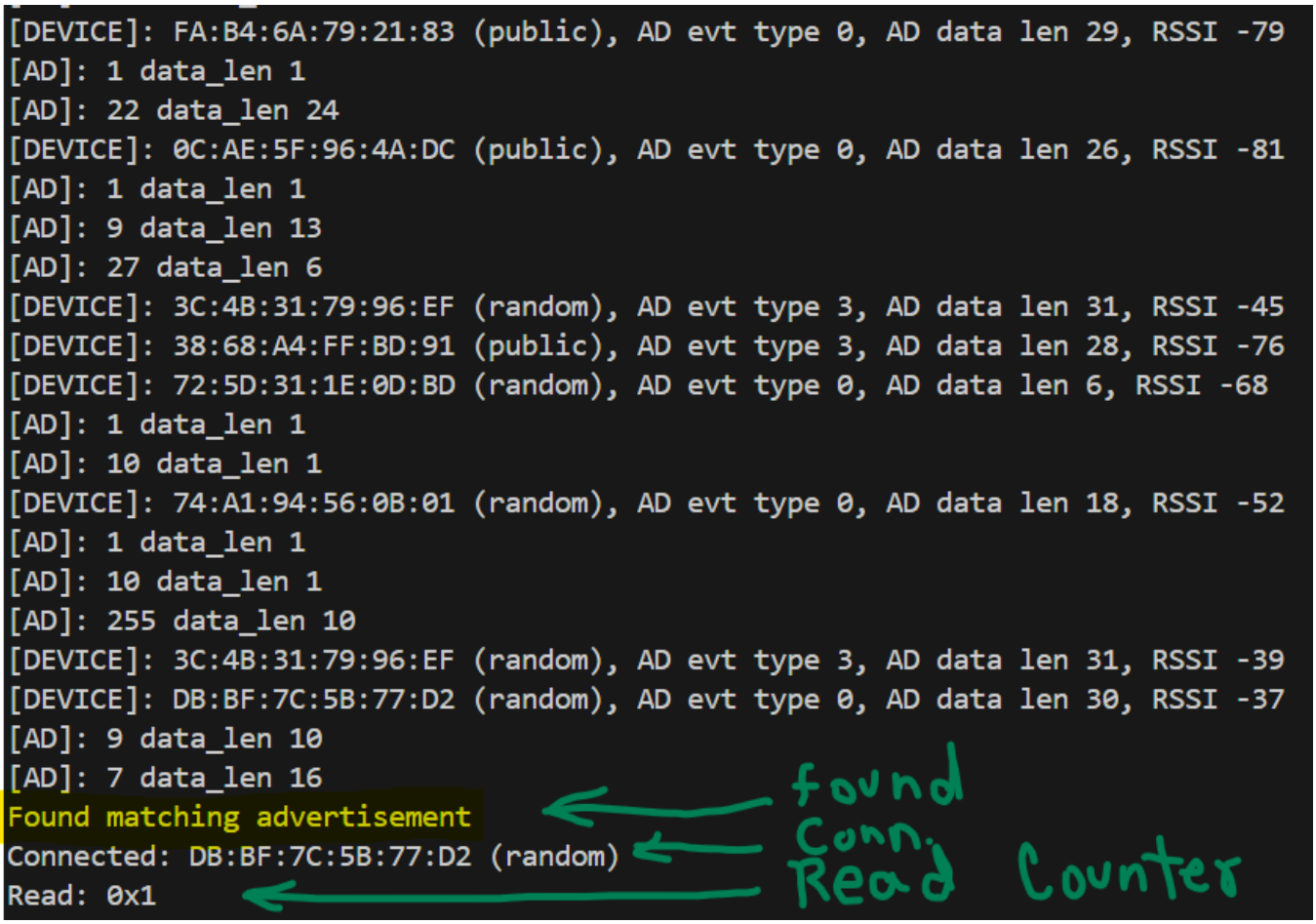


K: BLE Central
C:_RLS\ble-week2-starter-main\ble-week2-starter-main\ble-central

Functional Architecture:

Device	Functionality
Dongle (Peripheral) - (Ricks_Lab2)	32-bit unsigned int counter
DK (CENTRAL)	**The central device will connect to your peripheral and display the current count.** ○ Scan for advertisements. ○ Find an advertisement from your device ○ Connect to your device. ○ Read the characteristic. ○ Print the count value. • bt_gatt_discover(). This function is used to discover services and attributes. • the bt_gatt_discover() function is called multiple times. This allows finding the service first, and then finding the contained characteristic bt_gatt_discover_params Struct Reference: **https://docs.zephyrproject.org/apidoc/latest/structbt__gatt__discover__params.html**

1. **Show the terminal output showing the count increasing:**

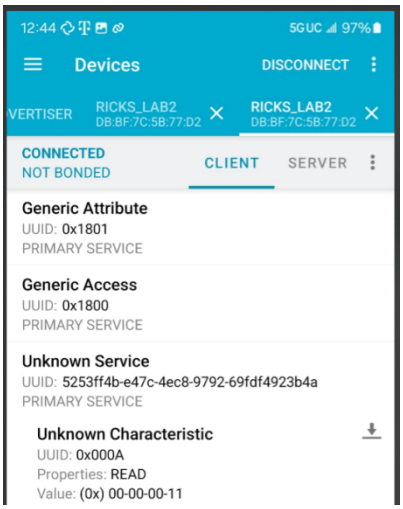


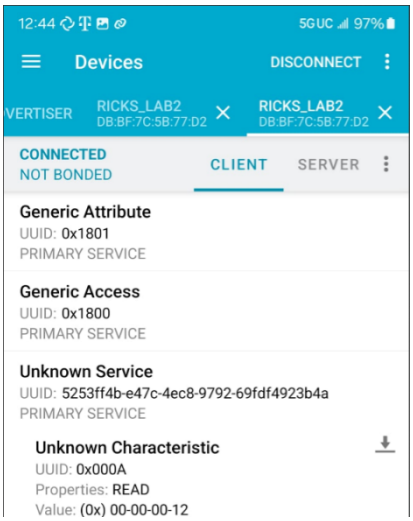
```
Read: 0xd
*** Booting nRF Connect SDK v2.9.0-7787b26498
*** Using Zephyr OS v3.7.99-1f8f3dc29142 ***
Bluetooth initialized
Scanning successfully started
[DEVICE]: 5D:B4:BA:89:BA:9A (random), AD evt
[AD]: 1 data_len 1
[AD]: 10 data_len 1
[AD]: 255 data_len 10
[DEVICE]: 0B:71:DF:7D:BF:25 (random), AD evt
[DEVICE]: 38:68:A4:FF:BD:91 (public), AD evt
[DEVICE]: 5D:3F:84:B4:9D:63 (random), AD evt
[AD]: 1 data_len 1
[AD]: 10 data_len 1
[DEVICE]: 3C:35:3A:BE:23:C3 (random), AD evt
[DEVICE]: FA:B4:6A:79:21:83 (public), AD evt
[AD]: 1 data_len 1
[AD]: 22 data_len 24
[DEVICE]: 5D:B4:BA:89:BA:9A (random), AD evt
[AD]: 1 data_len 1
[AD]: 10 data_len 1
[AD]: 255 data_len 10
[DEVICE]: 5D:3F:84:B4:9D:63 (random), AD evt
[AD]: 1 data_len 1
[AD]: 10 data_len 1
[DEVICE]: DB:BF:7C:5B:77:D2 (random), AD evt
[AD]: 9 data_len 10
[AD]: 7 data_len 16
Found matching advertisement
Connected: DB:BF:7C:5B:77:D2 (random)
Read: 0xe
*** Booting nRF Connect SDK v2.9.0-7787b26498
*** Using Zephyr OS v3.7.99-1f8f3dc29142 ***
Bluetooth initialized
Scanning successfully started
[DEVICE]: 5D:3F:84:B4:9D:63 (random), AD evt
[AD]: 1 data_len 1
[AD]: 10 data_len 1
[DEVICE]: 29:3B:91:BE:53:B4 (random), AD evt
[DEVICE]: 38:68:A4:FF:BD:91 (public), AD evt
[DEVICE]: DB:BF:7C:5B:77:D2 (random), AD evt
[AD]: 9 data_len 10
[AD]: 7 data_len 16
Found matching advertisement
Connected: DB:BF:7C:5B:77:D2 (random)
Read: 0xf
*** Booting nRF Connect SDK v2.9.0-7787b26498
*** Using Zephyr OS v3.7.99-1f8f3dc29142 ***
Bluetooth initialized
Scanning successfully started
[DEVICE]: 0B:71:DF:7D:BF:25 (random), AD evt
[DEVICE]: FA:74:34:75:3C:14 (random), AD evt
[DEVICE]: 5D:3F:84:B4:9D:63 (random), AD evt
[AD]: 1 data_len 1
[AD]: 10 data_len 1
[DEVICE]: 29:3B:91:BE:53:B4 (random), AD evt
[DEVICE]: FA:B4:6A:79:21:83 (public), AD evt
[AD]: 1 data_len 1
[AD]: 22 data_len 24
[DEVICE]: 5D:B4:BA:89:BA:9A (random), AD evt
[AD]: 1 data_len 1
[AD]: 10 data_len 1
[AD]: 255 data_len 10
[DEVICE]: FA:B4:6A:79:21:83 (public), AD evt
[AD]: 1 data_len 1
[AD]: 22 data_len 24
[DEVICE]: 38:68:A4:FF:BD:91 (public), AD evt
[DEVICE]: B8:B4:09:FB:23:35 (public), AD evt
[AD]: 1 data_len 1
[AD]: 255 data_len 26
[DEVICE]: 50:AD:D2:A2:4A:D6 (random), AD evt
[AD]: 1 data_len 1
[AD]: 255 data_len 26
[DEVICE]: 5D:B4:BA:89:BA:9A (random), AD evt
[AD]: 1 data_len 1
[AD]: 10 data_len 1
[AD]: 255 data_len 10
[DEVICE]: 0B:71:DF:7D:BF:25 (random), AD evt
[DEVICE]: 29:3B:91:BE:53:B4 (random), AD evt
[DEVICE]: FF:CE:06:C1:9F:6B (random), AD evt
[DEVICE]: 0B:71:DF:7D:BF:25 (random), AD evt
[DEVICE]: 0C:AE:5F:96:4A:DC (public), AD evt
[AD]: 1 data_len 1
[AD]: 9 data_len 13
[AD]: 27 data_len 6
[DEVICE]: 5D:3F:84:B4:9D:63 (random), AD evt
[AD]: 1 data_len 1
[AD]: 10 data_len 1
[DEVICE]: 62:D4:C3:80:01:DA (random), AD evt
[AD]: 1 data_len 1
[AD]: 10 data_len 1
[AD]: 255 data_len 9
[DEVICE]: 29:3B:91:BE:53:B4 (random), AD evt
[DEVICE]: FA:B4:6A:79:21:83 (public), AD evt
[AD]: 1 data_len 1
[AD]: 22 data_len 24
[DEVICE]: 5D:B4:BA:89:BA:9A (random), AD evt
[AD]: 1 data_len 1
[AD]: 10 data_len 1
[AD]: 255 data_len 10
[DEVICE]: 38:68:A4:FF:BD:91 (public), AD evt
[DEVICE]: 29:3B:91:BE:53:B4 (random), AD evt
[DEVICE]: FA:B4:6A:79:21:83 (public), AD evt
[AD]: 1 data_len 1
[AD]: 22 data_len 24
[DEVICE]: 5D:3F:84:B4:9D:63 (random), AD evt
[AD]: 1 data_len 1
[AD]: 10 data_len 1
[DEVICE]: 3C:35:3A:BE:23:C3 (random), AD evt
[DEVICE]: DB:BF:7C:5B:77:D2 (random), AD evt
[AD]: 9 data_len 10
[AD]: 7 data_len 16
Found matching advertisement
Connected: DB:BF:7C:5B:77:D2 (random)
Read: 0x10
```

After disconnecting and connecting multiple times from CENTRAL, observe that the counter goes from
`0x1, …, 0x10,`

`Then`

`I connect my smartphone (nRF for Mobile), click for 2 reads, then the counter increments :`
`0x11, 0x12`





Link to your updated code:
C:_RLS\ble-week2-starter-main\ble-week2-starter-main\ble-central\myBLECentral

L: LED Controller Application

LOOK at examples:
https://github.com/zephyrproject-rtos/zephyr/tree/main/samples
https://github.com/zephyrproject-rtos/zephyr/tree/main/samples/basic/button
See examples: https://github.com/zephyrproject-rtos/zephyr/blob/main/samples

1. **Write a few sentences on what you had to do to make this work. Particularly note anything that was especially challenging to get working.**
   Basically I had to do 2 projects (see in below table):

CENTRAL (DK): **1050243071** C:_RLS\ble-week2-starter-main\ble-week2-starter-main\ble-central\myBLECentral3 → press button here, then see the PERIPHERAL to toggle the LED on or off (for every write comand) Program this board as a **CENTRAL** which controls LED(s) **based on button presses** **Steps:** <x> Create CENTRAL project, ready to connect to Peripheral <x> Setup button0, so if presses, show a Trace message <x > implement write char. Into PERIPHERAL
PERIPHERAL (DK2) **1050209793**: C:_RLS\ble-week2-starter-main\ble-week2-starter-main\ble-peripheral2 Program this board as a **PERIPHERAL** with controllable **LEDs** <x> Create Peripheral project <x > turn LED on when connecting to CENTRAL (for now) <x > Implement that for every write command received from CENTRAL, just toggle the LED state

The challenging part for me was to figure out the WRITE characteristic, by looking into the READ example that we did already

2. **Link to your new code:**
   https://github.com/RicardoUCSD/Postlab2_BLE

3. **CHECKOFF: Showcase the central board controlling the LEDs of the peripheral board to a TA during lab or office hours**

Video is available on https://github.com/RicardoUCSD/Postlab2_BLE

CENTRAL

```
*** Booting nRF Connect SDK v2.9.0-7787b2649840 ***
*** Using Zephyr OS v3.7.99-1f8f3dc29142 ***
Set up button at gpio@50000000 pin 11
Bluetooth initialized
Scanning successfully started
Bluetooth init successfully (err 0)
[DEVICE]: 4A:B0:6B:AE:B2:D4 (random), AD evt type 0, AD data len 18, RSSI -76
[AD]: 1 data_len 1
[AD]: 3 data_len 2
[AD]: 255 data_len 9
[DEVICE]: B0:D2:78:D7:FD:AB (public), AD evt type 0, AD data len 30, RSSI -83
[AD]: 1 data_len 1
[AD]: 255 data_len 25
[DEVICE]: CB:F3:01:AE:31:A4 (random), AD evt type 0, AD data len 30, RSSI -29
[AD]: 9 data_len 10
[AD]: 7 data_len 16
Found matching advertisement
Connected: CB:F3:01:AE:31:A4 (random)
Discover successful(err 0)
[ATTRIBUTE] handle 16
Found service
Discover successful (err 0)
[ATTRIBUTE] handle 17
Found characteristic
Read successful (err 0)
read successful (err 0)
Read: 0x3
Button pressed at 688374
Write successful after button pressed (err 0)
Write successful on callback function (err 0)
Button pressed at 769712
Write successful after button pressed (err 0)
Write successful on callback function (err 0)
Button pressed at 804131
Write successful after button pressed (err 0)
Write successful on callback function (err 0)
Button pressed at 851492
Write successful after button pressed (err 0)
Write successful on callback function (err 0)
Button pressed at 892590
Write successful after button pressed (err 0)
Write successful on callback function (err 0)
```

PERIPHERAL

```
Disconnected (reason 0x08)
Connected
led0 will be turned on by writting to the service characteristic from remote CENTRAL device
remote CENTRAL device sent write characteristic command
LED state on
remote CENTRAL device sent write characteristic command
LED state off
remote CENTRAL device sent write characteristic command
LED state on
remote CENTRAL device sent write characteristic command
LED state off
remote CENTRAL device sent write characteristic command
LED state on
remote CENTRAL device sent write characteristic command
LED state off
remote CENTRAL device sent write characteristic command
LED state on
remote CENTRAL device sent write characteristic command
LED state off
remote CENTRAL device sent write characteristic command
LED state on
```

CODE

CENTRAL

```
#include <zephyr/types.h>
#include <stddef.h>
#include <errno.h>
#include <zephyr/kernel.h>
#include <zephyr/sys/printk.h>

#include <zephyr/bluetooth/bluetooth.h>
#include <zephyr/bluetooth/hci.h>
#include <zephyr/bluetooth/conn.h>
#include <zephyr/bluetooth/uuid.h>
#include <zephyr/bluetooth/gatt.h>
#include <zephyr/sys/byteorder.h>

#define LAB2_SERVICE_UUID BT_UUID_128_ENCODE(0xBDFC9792, 0x8234,
0x405E, 0xAE02, 0x35EF4174B299)
#define LAB2_SERVICE_CHARACTERISTIC_UUID 0x0001 // update for
uuid/characteristic for LED characteristc advertising. Need 1 LED

//~ button implementation
          #include <zephyr/drivers/gpio.h>
          #define SW0_NODE    DT_ALIAS(sw0)
          #if !DT_NODE_HAS_STATUS_OKAY(SW0_NODE)
                    #error "Unsupported board: sw0 devicetree alias is
not defined"
          #endif
          static const struct gpio_dt_spec button =
GPIO_DT_SPEC_GET_OR(SW0_NODE, gpios,{0});
          static struct gpio_callback button_cb_data;
          static uint8_t led_value = 0x01;
```

PERIPHERAL

```
#include <stdbool.h>
#include <zephyr/types.h>
#include <zephyr/drivers/sensor.h>
#include <stddef.h>
#include <string.h>
#include <errno.h>
#include <zephyr/sys/printk.h>
#include <zephyr/sys/byteorder.h>
#include <zephyr/kernel.h>

#include <zephyr/bluetooth/bluetooth.h>
#include <zephyr/bluetooth/hci.h>
#include <zephyr/bluetooth/conn.h>
#include <zephyr/bluetooth/uuid.h>
#include <zephyr/bluetooth/gatt.h>

//LED implementation
          #include <zephyr/drivers/gpio.h> //for led0 libs
          /* 1000 msec = 1 sec */
          #define SLEEP_TIME_MS   1000

          /* The devicetree node identifier for the "led0" alias. */
          #define LED0_NODE DT_ALIAS(led0)
          static const struct gpio_dt_spec led = GPIO_DT_SPEC_GET(LED0_NODE,
gpios);

//#define LAB2_WRITE_SERVICE_UUID BT_UUID_128_ENCODE(0xBDFC9792, 0x8234,
0x405E, 0xAE02, 0x35EF4174B299)
#define LAB2_SERVICE_UUID       BT_UUID_128_ENCODE(0xBDFC9792, 0x8234,
0x405E, 0xAE02, 0x35EF4174B299)
```

```
static void start_scan(void);
static struct bt_conn *default_conn;

static struct bt_uuid* search_service_uuid =
BT_UUID_DECLARE_128(LAB2_SERVICE_UUID);
static struct bt_uuid* search_characteristic_uuid =
BT_UUID_DECLARE_16(LAB2_SERVICE_CHARACTERISTIC_UUID);
static struct bt_gatt_discover_params discover_params;
static struct bt_gatt_read_params read_params;
static struct bt_gatt_write_params write_params;

static uint8_t write_func(struct bt_conn *conn, uint8_t err, struct
bt_gatt_write_params *params, const void *data, uint16_t length)
{
          if(err){
                    printk("write failed on callback function (err
%d)\n", err);
          }
          else{
                    printk("Write successful on callback function (err
%d)\n", err);
          }
          return BT_GATT_ITER_STOP;
}

void button_pressed(const struct device *dev, struct gpio_callback
*cb, uint32_t pins)
{
          printk("Button pressed at %" PRIu32 "\n", k_cycle_get_32());
          int err = bt_gatt_write(default_conn, &write_params);
          if (err) {
                    printk("Write failed after button pressed (err
%d)\n", err);
          }
          else{
                    printk("Write successful after button pressed (err
%d)\n", err);
          }
}

// Callback after reading characteristic value.
static uint8_t read_func(struct bt_conn *conn, uint8_t err, struct
bt_gatt_read_params *params, const void *data, uint16_t length)
{
          if (err) {
                    printk("read failed (err %d)\n", err);
          }
          else{
                    printk("read successful (err %d)\n", err);
          }

          uint8_t* buf = (uint8_t*) data;

          if (length == 4) {
                    uint32_t val = (((uint32_t) buf[0]) << 24) |
                                                        (((uint32_t)
buf[1]) << 16) |
                                                        (((uint32_t)
buf[2]) <<  8) |
                                                        (((uint32_t)
buf[3]) <<  0);

                    printk("Read: 0x%x\n", val);
          }
          return BT_GATT_ITER_STOP;
}

static uint8_t discover_func(struct bt_conn *conn, const struct
bt_gatt_attr *attr, struct bt_gatt_discover_params *params)
{
          int err;
          if (!attr) {
                    printk("Discover complete\n");
                    (void)memset(params, 0, sizeof(*params));
                    return BT_GATT_ITER_STOP;
          }
          printk("[ATTRIBUTE] handle %u\n", attr->handle);
          if (bt_uuid_cmp(discover_params.uuid,
BT_UUID_DECLARE_128(LAB2_SERVICE_UUID)) == 0) {
                    printk("Found service\n");
                    discover_params.uuid = search_characteristic_uuid;
                    discover_params.start_handle = attr->handle + 1;
                    discover_params.type =
BT_GATT_DISCOVER_CHARACTERISTIC;
                    err = bt_gatt_discover(conn, &discover_params);
                    if (err) {
                              printk("Discover failed (err %d)\n",
err);
                    }
                    else{
                              printk("Discover successful (err %d)\n",
err);
                    }
          }
          else if (bt_uuid_cmp(discover_params.uuid,
BT_UUID_DECLARE_16(LAB2_SERVICE_CHARACTERISTIC_UUID)) == 0) {
                    printk("Found characteristic\n");
                    //Add write values:
                    write_params.func = write_func;
                    write_params.handle =
bt_gatt_attr_value_handle(attr);
                    write_params.offset = 0;
          write_params.data = &led_value;
          write_params.length = sizeof(led_value);
```

```
//#define LAB2_SERVICE_CHARACTERISTIC_WRITE_UUID 0x0001
static uint8_t custom_value = 11;
static uint8_t led_state = 0x0;

//~ this characteristic READS the counter
static ssize_t characteristic_read(struct bt_conn *conn, const struct
bt_gatt_attr *attr, void *buf, uint16_t len, uint16_t offset);
static ssize_t characteristic_write(struct bt_conn *conn, const struct
bt_gatt_attr *attr, const void *buf, uint16_t len, uint16_t offset, uint8_t
flags);
// Global value that saves state for the characteristic.
uint32_t characteristic_value2 = 0x1;

// Set up the advertisement data.
#define DEVICE_NAME "Ricks_Lab2"
#define DEVICE_NAME_LEN (sizeof(DEVICE_NAME) - 1)

static const struct bt_data ad[] = {
          BT_DATA(BT_DATA_NAME_COMPLETE, DEVICE_NAME, DEVICE_NAME_LEN),
          BT_DATA_BYTES(BT_DATA_UUID128_ALL, LAB2_SERVICE_UUID)
};

// Setup the the service and characteristics.
BT_GATT_SERVICE_DEFINE(lab2_service,
          BT_GATT_PRIMARY_SERVICE(
                    BT_UUID_DECLARE_128(LAB2_SERVICE_UUID)
          ),
          BT_GATT_CHARACTERISTIC(BT_UUID_DECLARE_16(0x0001),
BT_GATT_CHRC_WRITE | BT_GATT_CHRC_READ,
          //0x0001 for write characteristic
                                        BT_GATT_PERM_WRITE | BT_GATT_PERM_READ,
characteristic_read, characteristic_write, &characteristic_value2),  //need
to imcrement characteristic value inside char..._read
);

// Callback when a client reads the characteristic.
//
// Documented under name "bt_gatt_attr_read_chrc()"
static ssize_t characteristic_read(struct bt_conn *conn,
                                                                  const struct
bt_gatt_attr *attr,

             void *buf,
                                                                  uint16_t len,

             uint16_t offset)
{
          // The `user_data` corresponds to the pointer provided as the last
"argument"
          // to the `BT_GATT_CHARACTERISTIC` macro.
          uint32_t *value = attr->user_data;

          static uint32_t counter = 0;
          counter++;                    // Increment the counter by one

          // Need to encode data into a buffer to send to client.
          uint8_t out_buffer[4] = {0};

    out_buffer[3] = (uint8_t)(counter & 0xFF);         // Most significant
byte
    out_buffer[2] = (uint8_t)((counter >> 8) & 0xFF);
    out_buffer[1] = (uint8_t)((counter >> 16) & 0xFF);
    out_buffer[0] = (uint8_t)((counter >> 24) & 0xFF); // Least significant
byte

          // User helper function to encode the output data to send to
          // the client.
          return bt_gatt_attr_read(conn, attr, buf, len, offset, out_buffer,
4);
}

//~ Add callback function to write into Characteristic
// Documented under name "bt_gatt_attr_read_chrc()"
static ssize_t characteristic_write(struct bt_conn *conn,
          const struct bt_gatt_attr *attr,
          const void *buf,
          uint16_t len,
          uint16_t offset,
          uint8_t flags)
{
          if (offset + len > sizeof(custom_value)) {
                    return BT_GATT_ERR(BT_ATT_ERR_INVALID_OFFSET);
          }

          memcpy(&custom_value, buf, len);
          //LOG_INF("Custom characteristic updated: %d", custom_value);
          printk("remote CENTRAL device sent write characteristic
command\n");
          gpio_pin_configure_dt(&led, GPIO_OUTPUT);  // Configure LED pin as
output
          led_state = (led_state>0x0)?0x0:0x1;
          gpio_pin_set_dt(&led, led_state);  // Toggle LED on
          printk("LED state %s\n", (led_state)>0?"on":"off");
          return len;
}

// Setup callbacks when devices connect and disconnect.
static void connected(struct bt_conn *conn, uint8_t err)
{
          if (err) {
                    printk("Connection failed (err 0x%02x)\n", err);
          } else {
                    printk("Connected\n");
                    printk("led0 will be turned on by writting to the
```

```
                    read_params.func = read_func;
                    read_params.handle_count = 1;
                    read_params.single.handle =
bt_gatt_attr_value_handle(attr);
                    read_params.single.offset = 0U;
                    err = bt_gatt_read(conn, &read_params);
                    if (err) {
                              printk("Read failed (err %d)\n", err);
                    }
                    else{
                              printk("Read successful (err %d)\n",
err);
                    }
          }
          return BT_GATT_ITER_STOP;
}

static void connected(struct bt_conn *conn, uint8_t conn_err)
{
          char addr[BT_ADDR_LE_STR_LEN];
          int err;
          bt_addr_le_to_str(bt_conn_get_dst(conn), addr,
sizeof(addr));
          if (conn_err) {
                    printk("Failed to connect to %s (%u)\n", addr,
conn_err);
                    bt_conn_unref(default_conn);
                    default_conn = NULL;
                    start_scan();
                    return;
          }

          printk("Connected: %s\n", addr);
          if (conn == default_conn) {
                    discover_params.uuid = search_service_uuid;
                    discover_params.func = discover_func;
                    discover_params.start_handle =
BT_ATT_FIRST_ATTRIBUTE_HANDLE;
                    discover_params.end_handle =
BT_ATT_LAST_ATTRIBUTE_HANDLE;
                    discover_params.type = BT_GATT_DISCOVER_PRIMARY;

                    err = bt_gatt_discover(default_conn,
&discover_params);
                    if (err) {
                              printk("Discover failed(err %d)\n",
err);
                              return;
                    }
                    else{
                              printk("Discover successful(err %d)\n",
err);
                    return;
                    }
          }
}

static void disconnected(struct bt_conn *conn, uint8_t reason)
{
          char addr[BT_ADDR_LE_STR_LEN];
          bt_addr_le_to_str(bt_conn_get_dst(conn), addr,
sizeof(addr));
          printk("Disconnected: %s (reason 0x%02x)\n", addr, reason);
          if (default_conn != conn) {
                    return;
          }
          bt_conn_unref(default_conn);
          default_conn = NULL;
          start_scan();
}

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

// Called for each advertising data element in the advertising data.
static bool ad_found(struct bt_data *data, void *user_data)
{
          bt_addr_le_t *addr = user_data;
          printk("[AD]: %u data_len %u\n", data->type, data-
>data_len); //UNCOMMENT TO PRINT ALL ADS
          switch (data->type) {
          case BT_DATA_UUID128_ALL:
                    if (data->data_len != 16) {
                              printk("AD malformed\n");
                              return true;
                    }
                    struct bt_le_conn_param *param;
                    struct bt_uuid uuid;
                    int err;
                    bt_uuid_create(&uuid, data->data, 16);
                    if (bt_uuid_cmp(&uuid,
BT_UUID_DECLARE_128(LAB2_SERVICE_UUID)) == 0) {
                              printk("Found matching
advertisement\n");
                              err = bt_le_scan_stop();
                              if (err) {
                                        printk("Stop LE scan failed
(err %d)\n", err);
                                        return false;
                              }
                              param = BT_LE_CONN_PARAM_DEFAULT;
                              err = bt_conn_le_create(addr,
BT_CONN_LE_CREATE_CONN, param, &default_conn);
service characteristic from remote CENTRAL device\n");
          }
}

static void disconnected(struct bt_conn *conn, uint8_t reason)
{
          printk("Disconnected (reason 0x%02x)\n", reason);
}

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

static void bt_ready(int err)
{
          if (err) {
                    printk("Bluetooth init failed (err %d)\n", err);
                    return;
          }
          printk("Bluetooth initialized\n");
          err = bt_le_adv_start(BT_LE_ADV_CONN, ad, ARRAY_SIZE(ad), NULL,
0);
          if (err) {
                    printk("Advertising failed to start (err %d)\n", err);
                    return;
          }
          printk("Advertising successfully started\n");
}

void main(void)
{
          int err;
          err = bt_enable(bt_ready);
          if (err) {
                    printk("Bluetooth init failed (err %d)\n", err);
                    return;
          }
}
```

```
                                if (err) {
                                        printk("Create conn failed
(err %d)\n", err);
                                        start_scan();
                                }
                        }
                        return false;
                }
                return true;
}

static void device_found(const bt_addr_le_t *addr, int8_t rssi,
uint8_t type, struct net_buf_simple *ad)
{
                char dev[BT_ADDR_LE_STR_LEN];
                bt_addr_le_to_str(addr, dev, sizeof(dev));
                printk("[DEVICE]: %s, AD evt type %u, AD data len %u, RSSI
%i\n", dev, type, ad->len, rssi);
                if (type == BT_GAP_ADV_TYPE_ADV_IND || type ==
BT_GAP_ADV_TYPE_ADV_DIRECT_IND) {
                        bt_data_parse(ad, ad_found, (void*) addr);
                }
}

static void start_scan(void)
{
                int err;
                struct bt_le_scan_param scan_param = {
                        .type       = BT_LE_SCAN_TYPE_PASSIVE,
                        .options    = BT_LE_SCAN_OPT_NONE,
                        .interval   = BT_GAP_SCAN_FAST_INTERVAL,
                        .window     = BT_GAP_SCAN_FAST_WINDOW,
                };
                err = bt_le_scan_start(&scan_param, device_found);
                if (err) {
                        printk("Scanning failed to start (err %d)\n",
err);
                        return;
                }
                printk("Scanning successfully started\n");
}

static void bt_ready(int err)
{
                if (err) {
                        printk("Bluetooth init failed (err %d)\n", err);
                        return;
                }
                printk("Bluetooth initialized\n");
                start_scan();
}

void main(void)
{
                //~ Setup BT
                int ret;
                if (!gpio_is_ready_dt(&button)) {
                        printk("Error: button device %s is not
ready\n",button.port->name);
                        return;
                }
                ret = gpio_pin_configure_dt(&button, GPIO_INPUT);
                if (ret != 0) {
                        printk("Error %d: failed to configure %s pin
%d\n",ret, button.port->name, button.pin);
                        return;
                }
                ret =
gpio_pin_interrupt_configure_dt(&button,GPIO_INT_EDGE_TO_ACTIVE);
                if (ret != 0) {
                        printk("Error %d: failed to configure interrupt on
%s pin %d\n",ret, button.port->name, button.pin);
                        return;
                }
                gpio_init_callback(&button_cb_data, button_pressed,
BIT(button.pin));
                gpio_add_callback(button.port, &button_cb_data);
                printk("Set up button at %s pin %d\n", button.port->name,
button.pin);

                //~ Setup BT
                int err;
                err = bt_enable(bt_ready);
                if (err) {
                        printk("Bluetooth init failed (err %d)\n", err);
                        return;
                }
                else{
                        printk("Bluetooth init successfully (err %d)\n",
err);
                        return;
                }
}
```