M031BT/M032BT SDK Quick Start Guide

Application Note for 32-bit NuMicro® Family

Document Information

Abstract	This document introduces how to start development with Nuvoton BLE.
Apply to	NuMicro® M031BT/M032BT BLE MCU series.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

Table of Contents

1 INTRODUCTION 4

2 SOFTWARE ARCHITECTURE 5

2.1 File Hierarchy Structure 5

2.2 Software Requirements 6

2.2.1 BLE RF Used Peripherals 6
2.2.2 Avoid Hinder the BLE RF Interrupt 6

2.3 Task Flow 6

3 SAMPLE CODE 7

3.1 Folder Structure and Demos 7

3.2 Project Structure 8

3.3 Selecting the Correct BLE Chipset 9

3.4 Starting the BLE Device 10

3.4.1 BLE_StackInit() 11
3.4.2 BleApp_Init() 12
3.4.3 BleApp_Main() 13
3.4.4 Starting the BLE Advertising 15

3.5 BLE Profile Implementation 16

3.5.1 SDK Provided Services 16
3.5.2 Implementing BLE Profile Definitions for Library 16
3.5.3 Implementing BLE Profile Definitions for Application 17
3.5.4 HRS Profile Action 18

3.6 Implementing the Non-BLE User Code 19

4 TESTING BLE SAMPLES 20

4.1 TRSP_UART_Peripheral 20

4.1.1 Testing TRSP_UART_Peripheral with Nuvoton NuBLE App 20

4.2 TRSP_LED_Peripheral 22

4.2.1 Testing TRSP_LED_Peripheral with Nuvoton NuBLE App 22

4.3 HRS_Peripheral 23

4.3.1 Testing HRS_Peripheral with nRF Toolbox App 23
4.3.2 Testing HRS_Peripheral with BLE Scanner App 25

4.4 HOGP_Peripheral 27

4.4.1 Using an Android Mobile Phone 27
4.5 DataRate_Peripheral 31
4.5.1 Testing DataRate_Peripheral with Nuvoton NuBLE App 31
4.6 TRSP_UART_Central 33
4.6.1 Testing TRSP_UART_Central with TRSP_UART_Peripheral 34
4.7 TRSP_UART_Multi_Central 35
4.7.1 Testing TRSP_UART_Multi_Central with TRSP_UART_Peripheral 35
4.8 TRSP_UART_Multi_Peripheral 37
4.8.1 Testing TRSP_UART_Multi_Peripheral with TRSP_UART_Central 37

5 BLE API 40
5.1 Commonly Used Functions 40

1 Introduction

Bluetooth Low Energy (BLE) is an emerging low-power wireless technology developed for short-range control and application monitoring.

Nuvoton provides a BLE 5.0 PHY, stack library and several demos with the M031BT/M032BT series microcontroller (MCU). This document introduces how to start development with Nuvoton BLE.

2 Software Architecture

2.1 File Hierarchy Structure

You can find the BLE material at the following path: *M031BSP\SampleCode\NuMaker-M03xBT\BLE*. There are five sub-folders, and the contents of each folder is listed as Table 2-1.

Directory Name	Contents
App	The mobile App to test BLE demos; you can also install it from store.
Demo	The BLE demos, now support KEIL project only.
Doc	Contains BLE API reference document and some user guide.
Source	Includes BLE headers, libraries, services and porting code.
Tool	The PC tool for OTA image creation

Table 2-1 Folders in BLE

Below is the BSP file hierarchy compared to the BLE architecture, as shown in Figure 2-1.

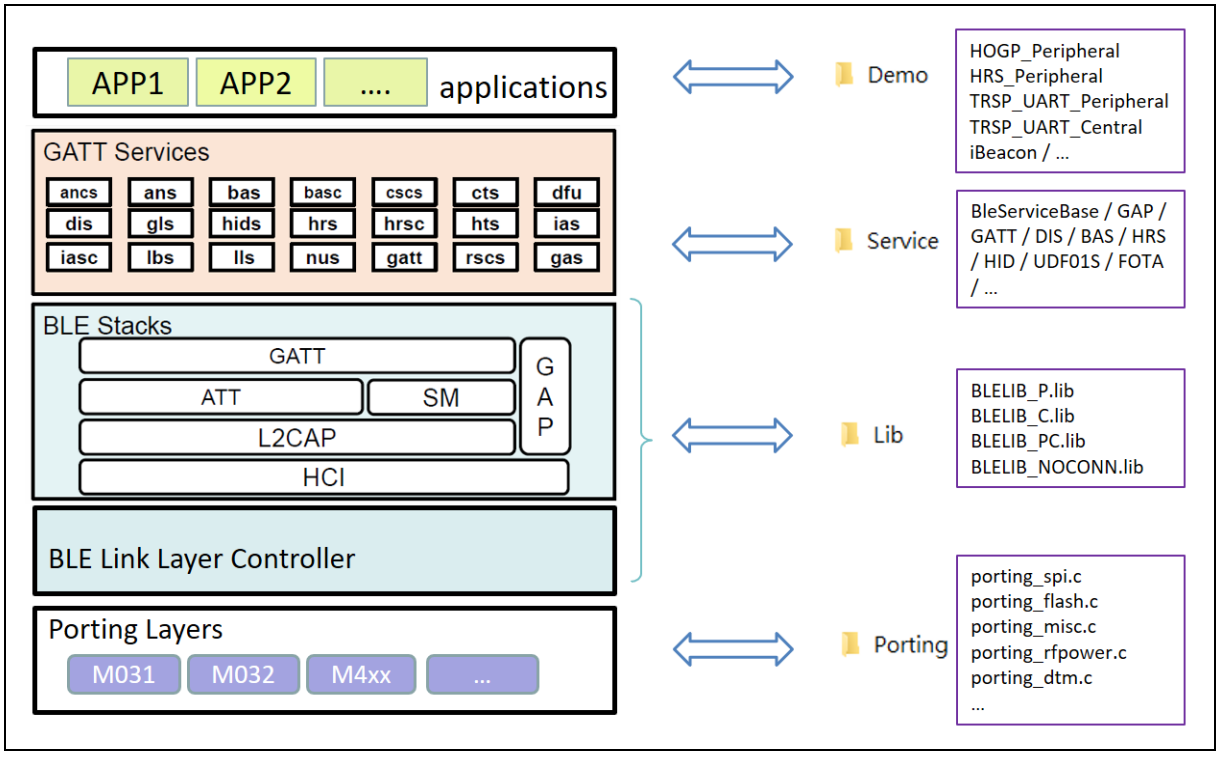

Figure 2-1 BLE BSP File Hierarchy

2.2 Software Requirements

2.2.1 BLE RF Used Peripherals

The following peripherals have been used by BLE RF. You should avoid using the same resource.

- SPI – SPI0 for BLE RF control.
- PDMA – Channel 3/4 for SPI0 TX/RX.
- GPIO_INT – GPIO pin C2/D12 as input for M031BT/M032BT BLE RF INT control.
- GPIO_RESET – GPIO pin A12/H4 as output for M031BT/M032BT BLE RF reset control.
- TIMER – Timer 3 for preventing BLE Link Layer event pending too long.
- Internal Flash – 4 pages for BLE bonding information, 1 page for OTA bank information, 3 pages for Data Flash partition to store user data.

2.2.2 Avoid Hinder the BLE RF Interrupt

The BLE RF interrupt needs to handle BLE event in time (150us), or the BLE behavior may be abnormal. The following programming rules can be used to protect the BLE RF action.

1. Only the BLE RF interrupt has the highest priority 0; other interrupts must be priority 1 or lower.
2. Since Flash page erasing will block CPU and hinder interrupts, after BLE RF is initialized, to do a flash page erase is not allowed.

2.3 Task Flow

The task flow is that the BLE stack always has the higher priority than user application, as shown in Figure 2-2.

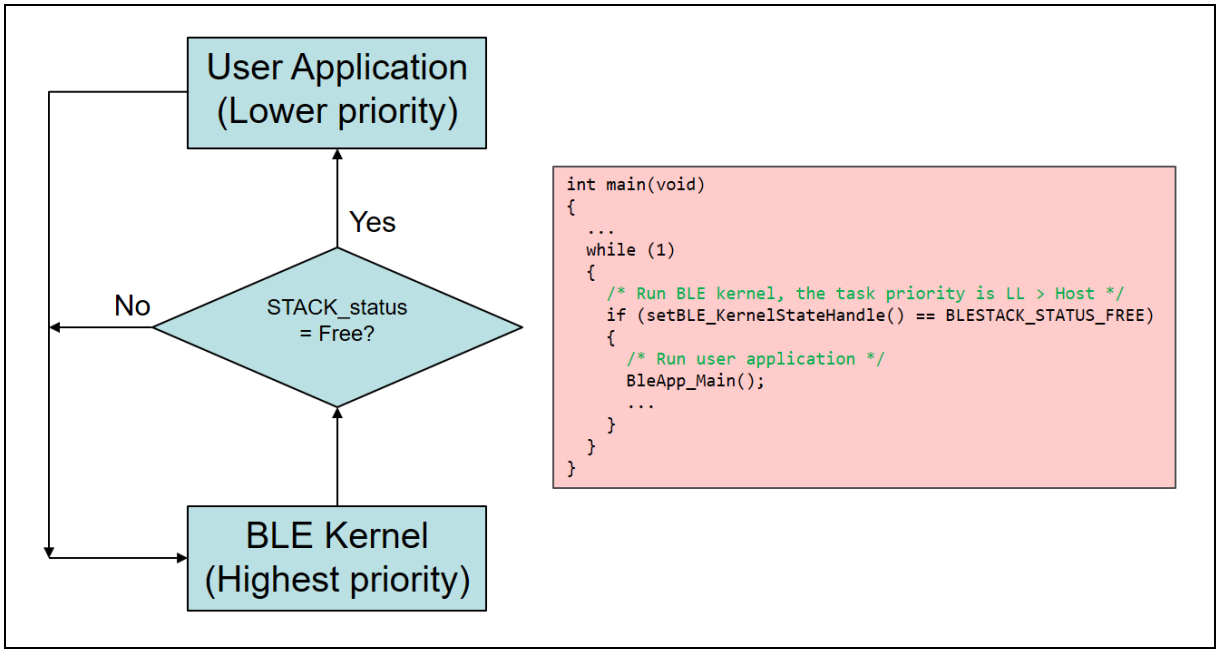

Figure 2-2 BLE BSP Task Flow

3 Sample Code

This BSP provides several samples. The following uses the HRS_Peripheral to explain the BLE project in BSP.

3.1 Folder Structure and Demos

There are five sub-folders in Demo folder, and the contents of each folder is listed as Table 3-1.

Directory Name	Contents
Peripheral	The demos use the BLE peripheral library.
Central	The demos use the BLE central library.
Central_and_Peripheral	The demos use the BLE central and peripheral library.
NoConnect	The demos use the BLE no connection library.
Other	The demos that do not use any BLE library.

Table 3-1 Folders in BLE Demo

In each folder, there are several demos with the same BLE behavior.

- Peripheral

ATCMD_Peripheral	AT Command via Transparent Service demo.
DataRate_Peripheral	Data rate test via Transparent Service demo.
HOGP_Peripheral	Human interface device over GATT Profile demo.
HRS_FOTA_Peripheral	Heart Rate Service + FOTA demo.
HRS_Peripheral	Heart Rate Service demo.
TRSP_LED_Peripheral	Transparent Service to control LED demo.
TRSP_UART_Peripheral	Transparent Service to transfer data via UART demo.
TRSP_UART_Multi_Peripheral	The same to TRSP_UART_Peripheral but can connect with multiple centrals.

- Central

TRSP_UART_Central	Transparent Service to transfer data via UART demo.
TRSP_UART_Multi_Central	The same to TRSP_UART_Central but can connect with multiple peripherals.

- Central_and_Peripheral

TRSP_UART_HRS	TRSP_UART_Central + HRS_Peripheral demo.

- NoConnect

Custom_Beacon	Customize Beacon format for data advertising demo.
DTM	Direct Test Mode demo.
iBeacon	Apple iBeacon demo.

- Other

FOTA_Bootloader	Bootloader for FOTA demo.

3.2 Project Structure

The project structure is shown in Figure 3-1. The Library group contains low level drivers and system startup code. The User group contains BLE profile definition and application sample code. The BLE Lib group contains BLE library and related functions. The Service group contains the BLE service. The Porting group contains the BLE related porting code.

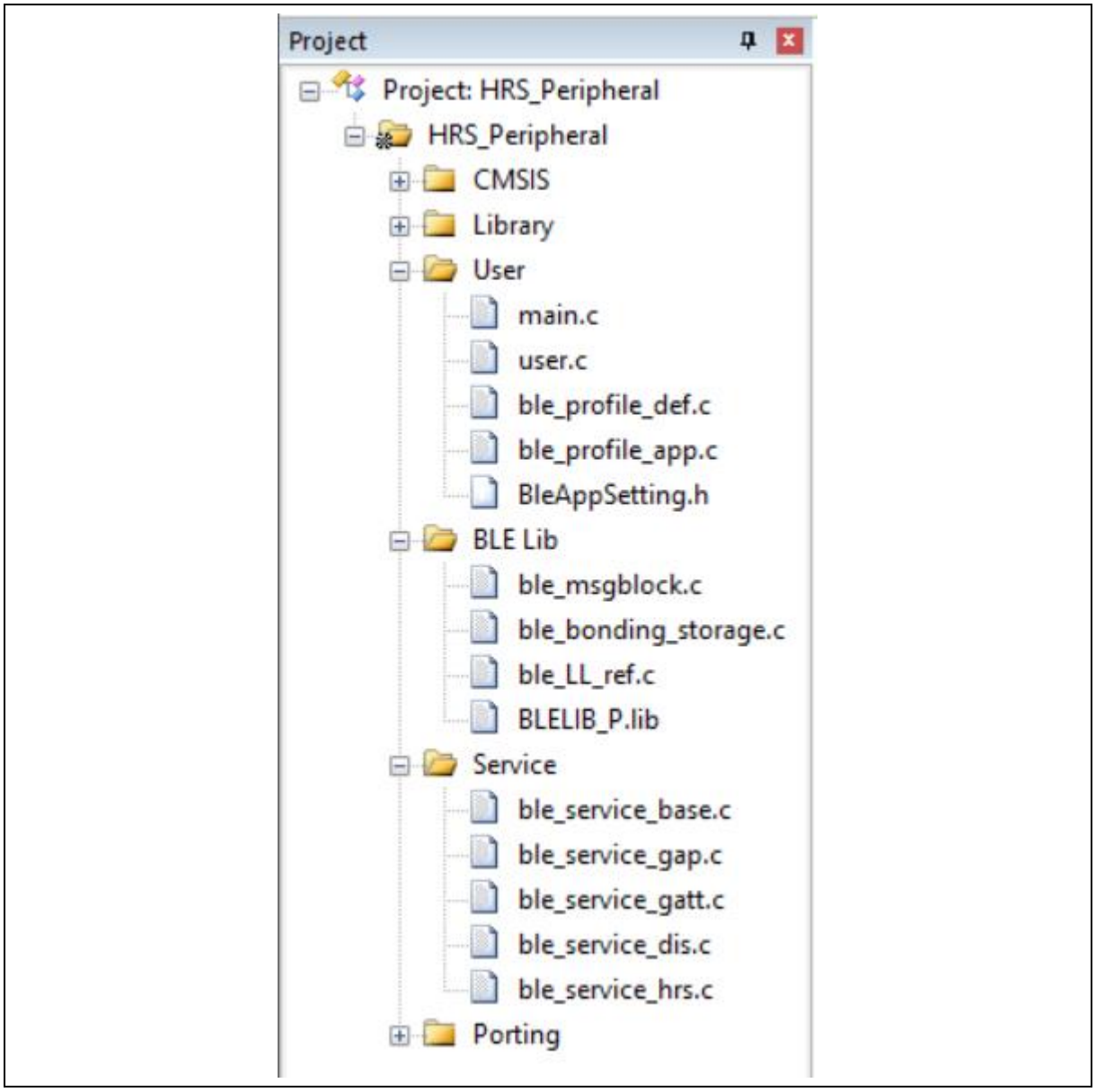

Figure 3-1 BLE Project Structure

3.3 Selecting the Correct BLE Chipset

Before building a BLE project, please select the chipset you want to do the development. Now the supported chipsets are M031BT and M032BT, and the default setting is for M031BT, which is defined in *Source\Porting\mcu_definition.h*.

```
/**
 * @ingroup mcuDef
 * @{
 * @brief Defined the MCU chip.
 */
```

```
#define _CHIP_M031BT                    0
#define _CHIP_M032BT                    1
#define _CHIP_SELECTION_                _CHIP_M031BT
```

3.4 Starting the BLE Device

To start a BLE device, please follow the steps below:

1. Call BLE_StackInit() to initial the BLE Stack.
2. Call Set_BleAddr() to set Bluetooth Device Address.
3. Call BleApp_Init() to initialize BLE application layer.
4. In while loop, call setBLE_KernelStateHandle() to run BLE tasks. In which function, the task priority is Link Layer > Host.
5. Implement BleApp_Main() for application behavior, and it will be processed when the BLE stack is free.
6. If all tasks are done, enter Power-down mode to reduce the power consumption.

```
/* Init BLE Stack */
status = BLE_StackInit();
if (status != BLESTACK_STATUS_SUCCESS)
{
    printf("BLE_StackInit() returns fail %d\n", status);
    while (1);
}

/* Set BLE device address */
Set_BleAddr();

/* Init BLE App */
status = BleApp_Init();
if (status != BLESTACK_STATUS_SUCCESS)
{
    printf("BleApp_Init() returns fail %d\n", status);
    while (1);
}

while (1)
{
    /* Run BLE kernel, the task priority is LL > Host */
    if (setBLE_KernelStateHandle() == BLESTACK_STATUS_FREE)
```

```
        {
            BleApp_Main();

            /* System enter Power Down mode & wait interrupt event. */
            setMCU_SystemPowerDown();
        }
    }
```

3.4.1 BLE_StackInit()

To initialize the BLE stack, you just need to call the BLE_StackInit(). The procedure is listed as follows:

1. Delay 200ms for waiting the RF PHY stable after power on.
2. Set GPIO pin to reset the RF PHY, and delay 50ms for reset complete.
3. Do SPI I/O re-mapping. After the re-mapping, the RF PHY is able to be controlled.
4. Initialize PDMA for SPI large packet transfer.
5. Call setRF_Init() with parameters (power regulator and crystal frequency) to initial the RF PHY, the parameters are hardware related.
6. Call setBLE_BleStackInit() to initial BLE stack.

```
    BleStackStatus BLE_StackInit()
    {
        BleStackStatus status;

        /* Wait RF PHY stable */
        CLK_SysTickDelay(200000);

        /* Do Gpio Reset */
        seBLE_GpioReset();
        CLK_SysTickDelay(50000);

        /* SPI IO remapping */
        setRF_SpiIoMapping();

        /* Init SPI PDMA */
        setBLE_SpiPDMAInit();

        /* Init RF PHY */
        status = setRF_Init(DCDC_REGULATOR, XTAL_16M);
        BLESTACK_STATUS_CHECK(status);
```

```
    /* Init BLE Stack */
    status = setBLE_BleStackInit();
    BLESTACK_STATUS_CHECK(status);

    return BLESTACK_STATUS_SUCCESS;
}
```

3.4.2 BleApp_Init()

The BleApp_Init() is used to initialize application default settings. The procedure is listed as follows:

1. Set BLE Company ID to BLE stack.
2. Register BLE event handler to receive BLE related events.
3. Initialize BLE profiles.

```
BleStackStatus BleApp_Init(void)
{
    BleStackStatus status = BLESTACK_STATUS_SUCCESS;

    /* set company Id*/
    setBLE_CompanyId(((uint16_t)BLE_COMPANY_ID_H << 8) | BLE_COMPANY_ID_L);

    /* register command event callback function */
    setBLE_RegisterBleEvent(BleEvent_Callback);

    /* initial profiles */
    status = BleApp_ProfileInit();
    if (status != BLESTACK_STATUS_SUCCESS)
    {
        printf("Error init profiles.\n");
    }

    return status;
}
```

In the BleEvent_Callback(), you can handle all of BLE events.

```
/* BLE Callback Function */
static void BleEvent_Callback(BleCmdEvent event, void *param)
{
```

```
        switch (event)
        {
        case BLECMD_EVENT_ADV_COMPLETE:
            printf("Advertising...\n");
            break;

        case BLECMD_EVENT_CONN_COMPLETE:
        {
            BLE_Event_ConnParam *connParam = (BLE_Event_ConnParam *)param;

            // set connection state
            if (connParam->hostId == CONN_HRS_LINK_HOSTID)
            {
                bleProfile_link0_info.bleState = STATE_BLE_CONNECTION;
            }
            printf("Status=%d, ID=%d, Connected to %02x:%02x:%02x:%02x:%02x:%02x\n",
                   connParam->status,
                   connParam->hostId,
                   connParam->peerAddr.addr[5],
                   connParam->peerAddr.addr[4],
                   connParam->peerAddr.addr[3],
                   connParam->peerAddr.addr[2],
                   connParam->peerAddr.addr[1],
                   connParam->peerAddr.addr[0]);
        }
        break;
        ...
        ...
    }
```

3.4.3 BleApp_Main()

The BleApp_Main() is used to implement the BLE application behavior. The procedure is listed as follows:

Step A: Handle every link in BleApp_Main().

Step B: Implement handle_AppLinkn_xxx() function for every link.

Step C: In handle_AppLinkn_xxx(), implement the corresponding actions for different BLE state. In HRS_Peripheral case, the notification action is implemented in timer interrupt

handler, so you cannot see any action for STATE_BLE_CONNECTION here.

Step D: Implement sub-function in BLE connection state if using the central mode.

The example code of HRS_Peripheral demo is shown as follows.

```
void BleApp_Main(void)
{
    // Handle Link 0 - HRS Peripheral
    handle_AppLink0_HRSP();
}
```

```
void handle_AppLink0_HRSP(void)
{
    BleStackStatus status;

    if (bleProfile_link0_info.bleState == STATE_BLE_STANDBY)
    {
        // reset preferred MTU size
        setBLEGATT_PreferredMtuSize(CONN_HRS_LINK_HOSTID, DEFAULT_MTU);

        // reset service data
        BleService_GATT_DataInit(&(bleProfile_link0_info.serviceGATT_info_s.data));
        BleService_HRS_DataInit(&bleProfile_link0_info.serviceHRS_info_s.data);

        // enable advertisement
        status = Ble_AdvStart(CONN_HRS_LINK_HOSTID);
        if (status == BLESTACK_STATUS_SUCCESS)
        {
            bleProfile_link0_info.bleState = STATE_BLE_ADVERTISING;
        }
    }
    else if (bleProfile_link0_info.bleState == STATE_BLE_CONNECTION)
    {
        if ((hrs_data_transmit_enable == 1) && (heart_rate_measurement_cccd != 0 ))
        {
            hrs_data_transmit_enable = 0;

            // process the actions of connection state
            ...
        }
```

```
        }
    }
```

In BLE central mode, the procedure should be shown as Figure 3-2.

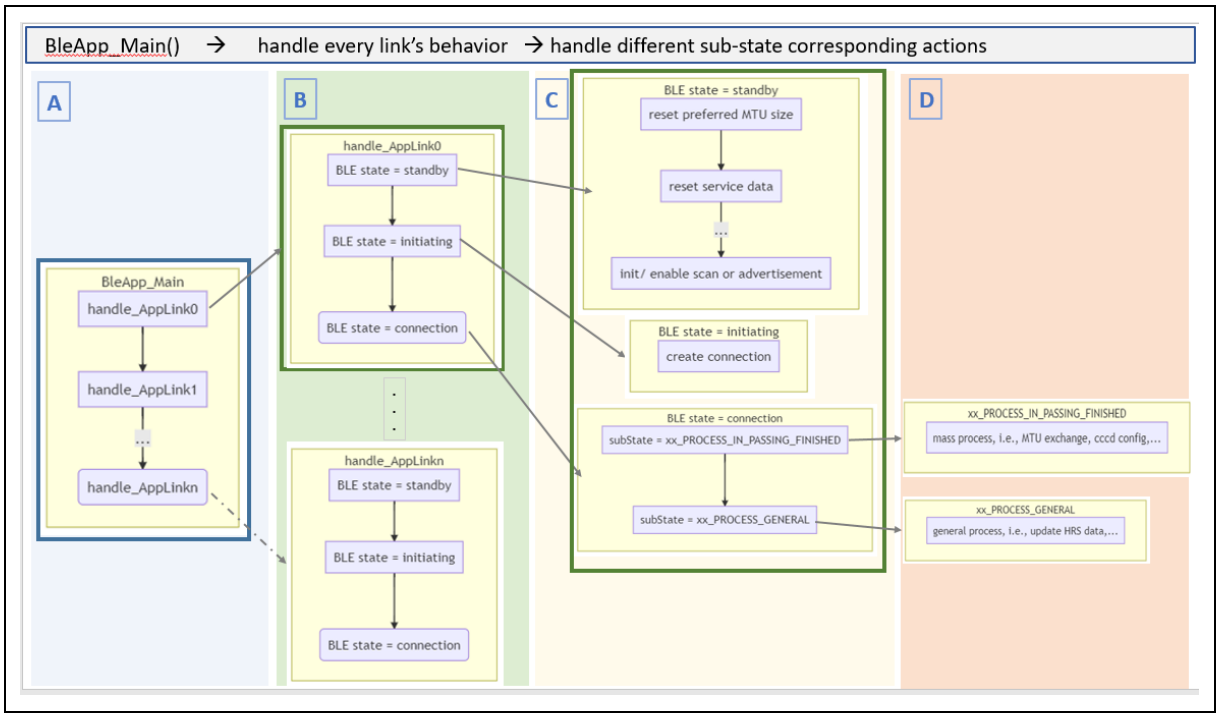

Figure 3-2 BLE Application Behavior of Central Mode

3.4.4 Starting the BLE Advertising

In the *user.c* file, you need to define the advertising interval time and scan response data. If the ADV_INTERVAL_MIN and ADV_INTERVAL_MAX have different values, the BLE stack will select an appropriate value that should be close to the middle value.

```
// Advertising device name
#define DEVICE_NAME    'N', 'u', 'v', 'o', 't', 'o', 'n', '_', 'H', 'R', 'S'

//Advertising parameters
#define APP_ADV_INTERVAL_MIN            160U    // 160*0.625ms=100ms
#define APP_ADV_INTERVAL_MAX            160U    // 160*0.625ms=100ms
```

In BleApp_Main(), call the Ble_AdvInit() to set BLE advertising data, and then call the setBLE_AdvEnable() to start BLE advertising.

```
BleStackStatus Ble_AdvStart(uint8_t hostId)
{
```

```
    BleStackStatus status;

    status = Ble_AdvInit();
    BLESTACK_STATUS_CHECK(status);

    status = setBLE_AdvEnable(hostId);
    BLESTACK_STATUS_CHECK(status);

    return BLESTACK_STATUS_SUCCESS;
}
```

3.5 BLE Profile Implementation

The procedure of four steps to implement the BLE profiles in application is listed below:

1. Implement BLE services. (*ble_service_xxx.c / ble_service_xxx.h*)
2. Implement BLE profile definitions for library. (*ble_profile_def.c*)
3. Implement BLE profile definitions for application. (*ble_profile_app.c / ble_profile.h*)
4. Implement BLE profile actions (*user.c*)

3.5.1 SDK Provided Services

The BLE SDK provides the following defined services for reference:

- SIG defined Service
  - GAP (Generic Access Profile)
  - GATT (Generic Attribute Profile)
  - DIS (Device Information Service)
  - BAS (Battery Service)
  - HID (Human Interface Device)
  - HRS (Heart Rate Service)
- Customer defined Service
  - TRSP (Transparent)
  - FOTA (Firmware Over-the-Air)

3.5.2 Implementing BLE Profile Definitions for Library

In the *ble_profile_def.c* file, you should implement the BLE profile related definitions for library. In HRS_Peripheral, you can see the profile includes the GAP, GATT, DIS and HRS services.

For the role definition, it is the server (peripheral) mode. Please note that the role definition must be the same as the definition in BleApp_ProfileInit().

```
/* Service Combination #00 */
const ATTRIBUTE_BLE *const ATT_SERVICE_COMB00[] =
{
```

```
        &ATT_NULL_INVALID,       //mandatory, don't remove it.
        ATT_GAP_SERVICE
        ATT_GATT_SERVICE
        ATT_DIS_SERVICE
        ATT_HRS_SERVICE
    };

    /* BLE Connection Links Definition */
    const ATTR_DB_Role_by_ID ATT_DB_LINK[] =
    {
        // Link 0
        {
            ((const ATTRIBUTE_BLE **)0),   // Client Profile
            ATT_SERVICE_COMB00,            // Server Profile
        },
    };
    …
    …
```

3.5.3 Implementing BLE Profile Definitions for Application

In the *ble_profile.h* and *ble_profile_app.c* files, you should implement the BLE profile related definitions for application. At first, you need to define the maximum number of connection link for each service.

```
    /** Define the maximum number of BLE GAP service link. */
    #define MAX_NUM_CONN_GAP            1

    /** Define the maximum number of BLE GATT service link. */
    #define MAX_NUM_CONN_GATT           1

    /** Define the maximum number of BLE DIS service link. */
    #define MAX_NUM_CONN_DIS            1

    /** Define the maximum number of BLE HRS service link. */
    #define MAX_NUM_CONN_HRS            1
```

Secondly, you need to define the data structure for each link.

```
    /** BLE Application Link 0 Profile Attribute Information Structure. */
    typedef struct BLEProfile_Link0_Info
    {
```

```
        uint8_t          hostId;              /**< Host id. */
        BleMode          bleState;            /**< Current BLE mode. */
        uint8_t          subState;            /**< Current link substate. */
        BLEATT_GAP_Info  serviceGAP_info_s;   /**< GAP Service information (server). */
        BLEATT_GATT_Info serviceGATT_info_s;  /**< GATT Service information (server). */
        BLEATT_DIS_Info  serviceDIS_info_s;   /**< DIS Service information (server). */
        BLEATT_HRS_Info  serviceHRS_info_s;   /**< HRS Service information (server). */
    } BLEProfile_Link0_Info;

    /** Extern BLE Application Link 0 Profile Attribute Information Initialization. */
    extern BLEProfile_Link0_Info bleProfile_link0_info;
```

Finally, if BLE GATT role is set to BLE_GATT_ROLE_CLIENT, you still need to implement the getBLELink0_ServiceHandles() function to get all service attribute handles after receiving BLECMD_EVENT_ATT_DATABASE_PARSING_FINISHED event. You can refer to the TRSP_UART_Central demo for this step.

3.5.4 HRS Profile Action

In the *user.c* file, you can see after connected, it sends out the heart rate measurement value via notification in function handle_AppLink0_HRSP().

```
    //initial is 0x14 & 0x02 = 0, toggle "device detected" / "device not detected" information
    if ((bleProfile_link0_info.serviceHRS_info_s.data.heart_rate_measurement[0] & BIT1) == 0)
    {
        //set Sensor Contact Status bit
        bleProfile_link0_info.serviceHRS_info_s.data.heart_rate_measurement[0] |= BIT1;
    }
    else
    {
        //clear Sensor Contact Status bit
        bleProfile_link0_info.serviceHRS_info_s.data.heart_rate_measurement[0] &= ~BIT1;
    }

    if (setBLEGATT_Notification(bleProfile_link0_info.hostId,
                                bleProfile_link0_info.serviceHRS_info_s.handles.hdl_heart_rate_measurement,
                                bleProfile_link0_info.serviceHRS_info_s.data.heart_rate_measurement,
                                sizeof(bleProfile_link0_info.serviceHRS_info_s.data.heart_rate_measurement) /
sizeof(bleProfile_link0_info.serviceHRS_info_s.data.heart_rate_measurement[0])
                               ) == BLESTACK_STATUS_SUCCESS)
    {
        //+1, Heart Rate Data. Here just a simulation, increase 1 about every second
        bleProfile_link0_info.serviceHRS_info_s.data.heart_rate_measurement[1]++;
```

```
        //+1, Heart Rate RR-Interval
        bleProfile_link0_info.serviceHRS_info_s.data.heart_rate_measurement[2]++;
    }
```

The frequency of sending data is once per second; you can find it in the timer interrupt handler.

```
void TMR0_IRQHandler(void)
{
    if (TIMER_GetIntFlag(TIMER0) == 1)
    {
        /* Clear Timer0 time-out interrupt flag */
        TIMER_ClearIntFlag(TIMER0);
    }

    // send heart rate measurement value
    hrs_data_transmit_enable = 1;
}
```

3.6 Implementing the Non-BLE User Code

Before adding your own non-BLE code, please note that do NOT occupy the CPU resource too much, as it may cause Link Layer cannot handle the BLE event in time.

It is suggested that users add their own non-BLE code in the *main.c* file as follows.

```
while (1)
{
    /* Run BLE kernel, the task priority is LL > Host */
    if (setBLE_KernelStateHandle() == BLESTACK_STATUS_FREE)
    {
        BleApp_Main();

        /* Here to add Non-BLE code */
        User_Main();

        /* System enter Power Down mode & wait interrupt event. */
        setMCU_SystemPowerDown();
    }
}
```

4 Testing BLE Samples

Before testing BLE demos with a smart phone, please note that you need to enable the Bluetooth function in smart phone.

Following sections introduce how to test the BLE demos.

4.1 TRSP_UART_Peripheral

The TRSP_UART_Peripheral is a demo that lets NuMaker-M03xBT board and mobile App do bidirectional data transfer via BLE and UART. Since TRSP is not a standard BLE profile, the easiest way to test TRSP_UART_Peripheral is using the Nuvoton NuBLE App.

4.1.1 Testing TRSP_UART_Peripheral with Nuvoton NuBLE App

You can find the App at the path: *M031BSP\SampleCode\NuMaker-M03xBT\BLE\App*. The local App supports Android systems only. You can also search the keyword “Nuvotonble” in Google Play store and Apple App Store to install the NuBLE App.

Follow the steps below to test TRSP_UART_Peripheral:

1. Start a terminal software (e.g. Putty) and then open the VCOM of NuMaker-M03xBT board, where the UART baud rate is 115200. You will see the following message after booting, as shown in Figure 4-1.

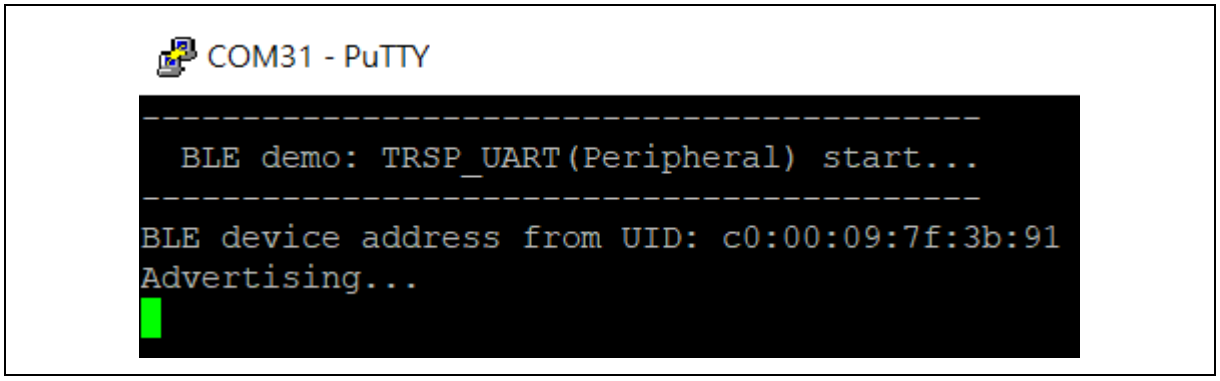

Figure 4-1 TRSP_UART_Peripheral Boot Message

2. Open NuBLE App. You can find an “SCAN BLE” icon at bottom-left, as shown in Figure 4-2. Click the icon, and you will see the “Nuvoton_UART” device in the scan list. Clicking it can connect to the device.

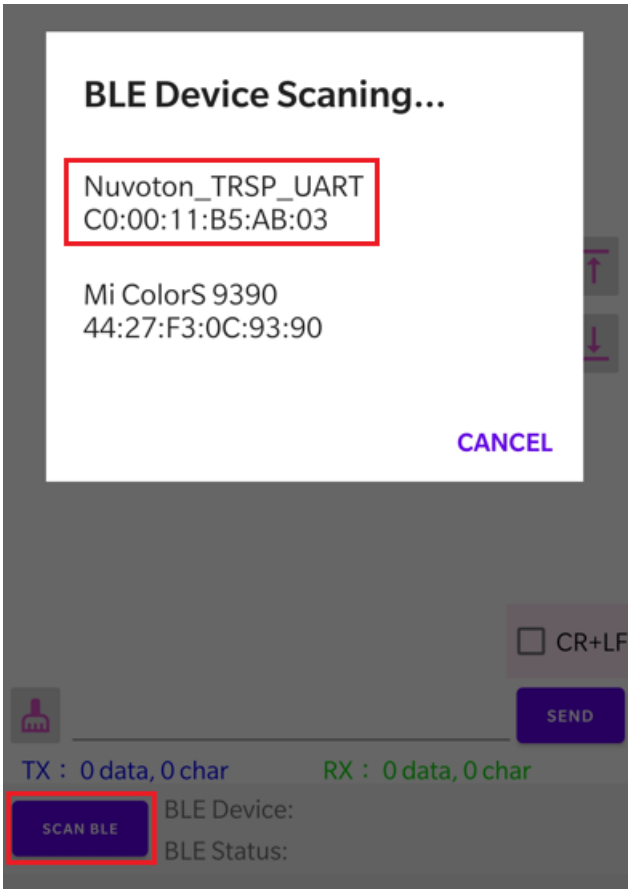

Figure 4-2 NuBLE TRSP_UART Main Screen

3. For sending data from PC to mobile App, you can type string “1234” + ENTER in debug console. You will see the string in App, as shown in Figure 4-3.

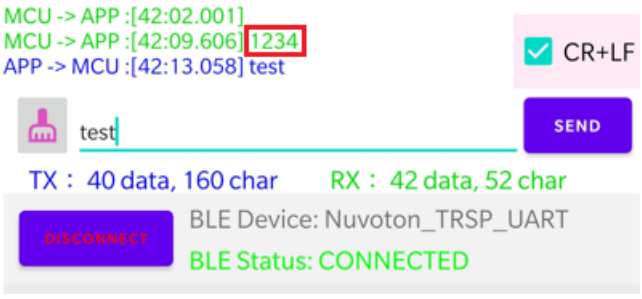

Figure 4-3 TRSP_UART PC to App Screen

4. For sending data from mobile App to PC, you can try to send string “test” in App. Then you will see the string in debug console, as shown in Figure 4-4.

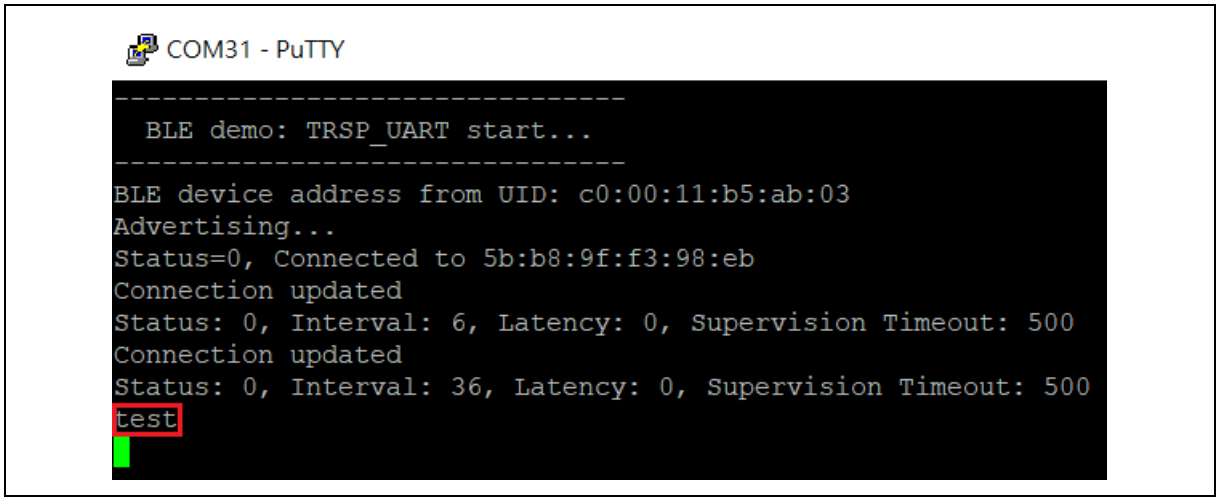

Figure 4-4 TRSP_UART App to PC Screen

4.2 TRSP_LED_Peripheral

The TRSP_LED_Peripheral is a demo for a remote device to control the LED on the NuMaker-M03xBT board via BLE. You can use the Nuvoton NuBLE App to test it.

4.2.1 Testing TRSP_LED_Peripheral with Nuvoton NuBLE App

Follow the steps below to test TRSP_LED_Peripheral:

1. Open the NuBLE App, you can find an icon in the top-left. Click the icon, and you can see several tabs, which are TRSP_UART, TRSP_LED, OTA and DataRate, as shown in Figure 4-5. The default tab is the TRSP_UART, and you need to change to TRSP_LED.

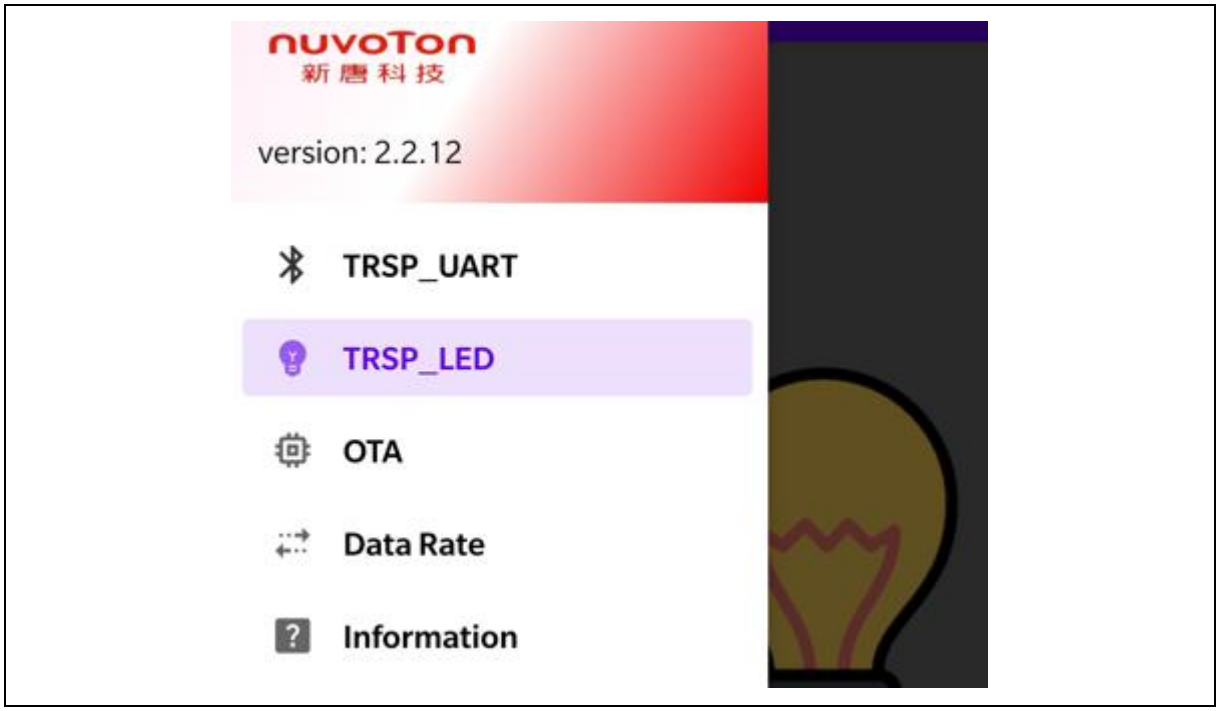

Figure 4-5 NuBLE Tabs

2. In TRSP_LED tab, you can see two LED control icons, as shown in Figure 4-6. After connect to device, you can click the ON/OFF icons to control the LED on the NuMaker-M03xBT board.

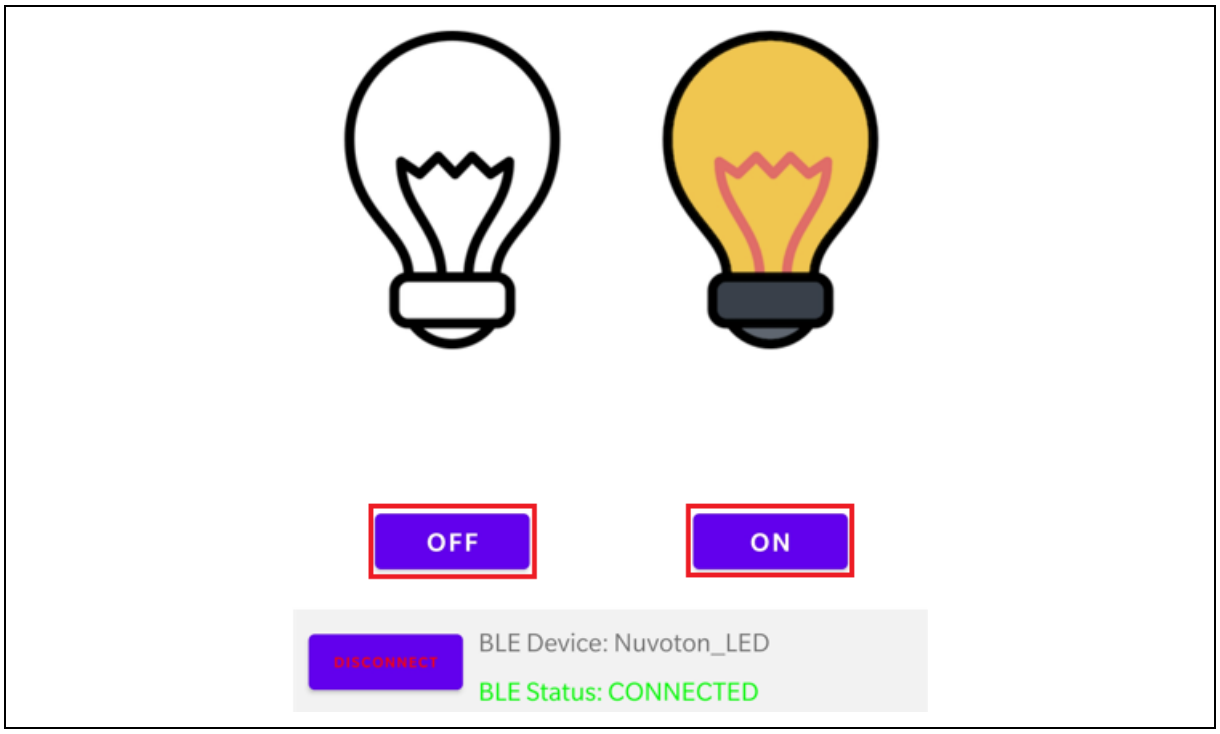

Figure 4-6 NuBLE TRSP_LED Control Screen

4.3 HRS_Peripheral

The Heart Rate Profile is a standard profile of BLE. You can use a BLE App that supports HRS (e.g. nRF Toolbox) to test demo easily. Or you can use a BLE App that supports generic actions (e.g. BLE Scanner or nRF Connect) to get the detailed information.

After connected to HRS device, the App can receive the heart rate measurement value. In this demo, the measurement value will be increased by 1 per second.

4.3.1 Testing HRS_Peripheral with nRF Toolbox App

The nRF Toolbox works with a wide range of the BLE profiles. You can follow the steps below to test HRS_Peripheral demo:

1. Click the HRM icon, as shown in Figure 4-7.

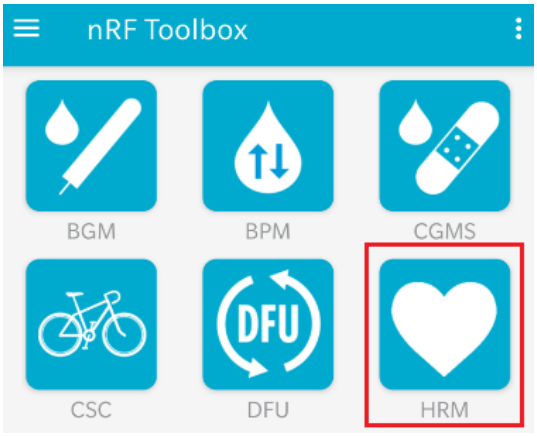

Figure 4-7 nRF Toolbox Main Screen

2. Click the “CONNECT” button at bottom to scan device. You can see the “Nuvoton_HRS” device in AVAILABLE DEVICES list, as shown in Figure 4-8. Then you can click the “Nuvoton_HRS” to connect the device.

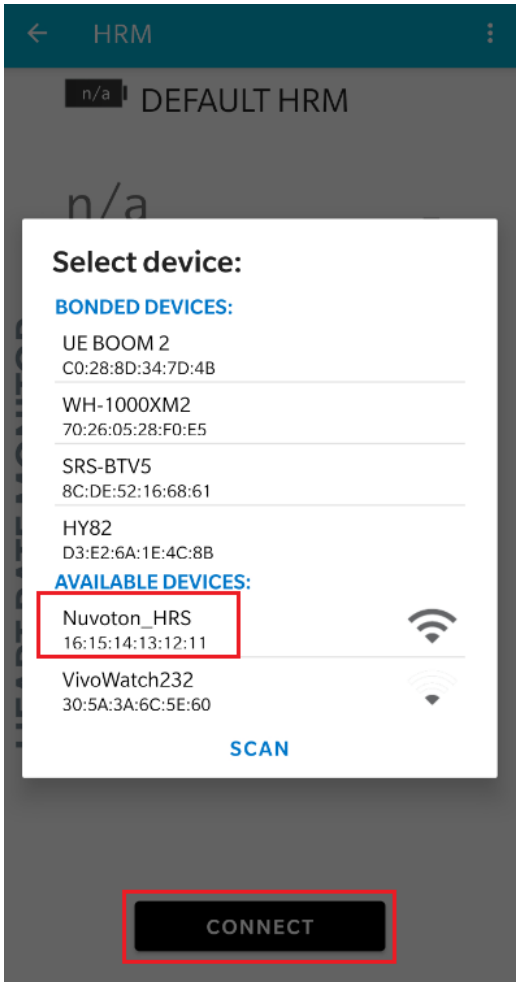

Figure 4-8 nRF Toolbox Scan and Connect

3. After connected to HRS device, you can see the heart rate measurement value, sensor position and battery information, as shown in Figure 4-9.

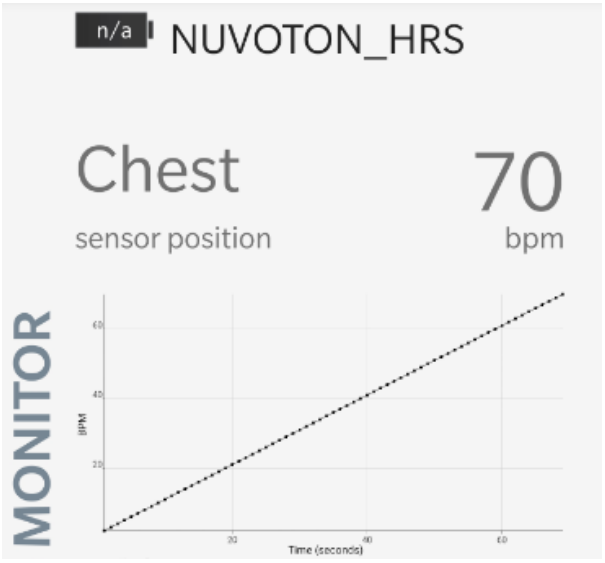

Figure 4-9 Heart Rate Measurement Screen

4.3.2 Testing HRS_Peripheral with BLE Scanner App

BLE Scanner is a popular tool used by developers to find the BLE devices. You can follow the steps below to test HRS_Peripheral demo:

1. BLE Scanner lists the near BLE devices in the main screen. You can find the device “Nuvoton_HRS”, as shown in Figure 4-10.

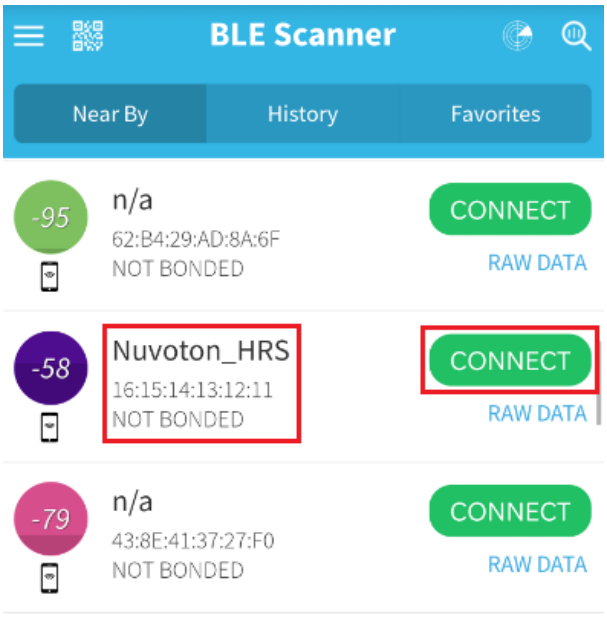

Figure 4-10 BLE Scanner Main Screen

2. After clicking the CONNECT button to connect device, you can see all services, as shown in Figure 4-11.

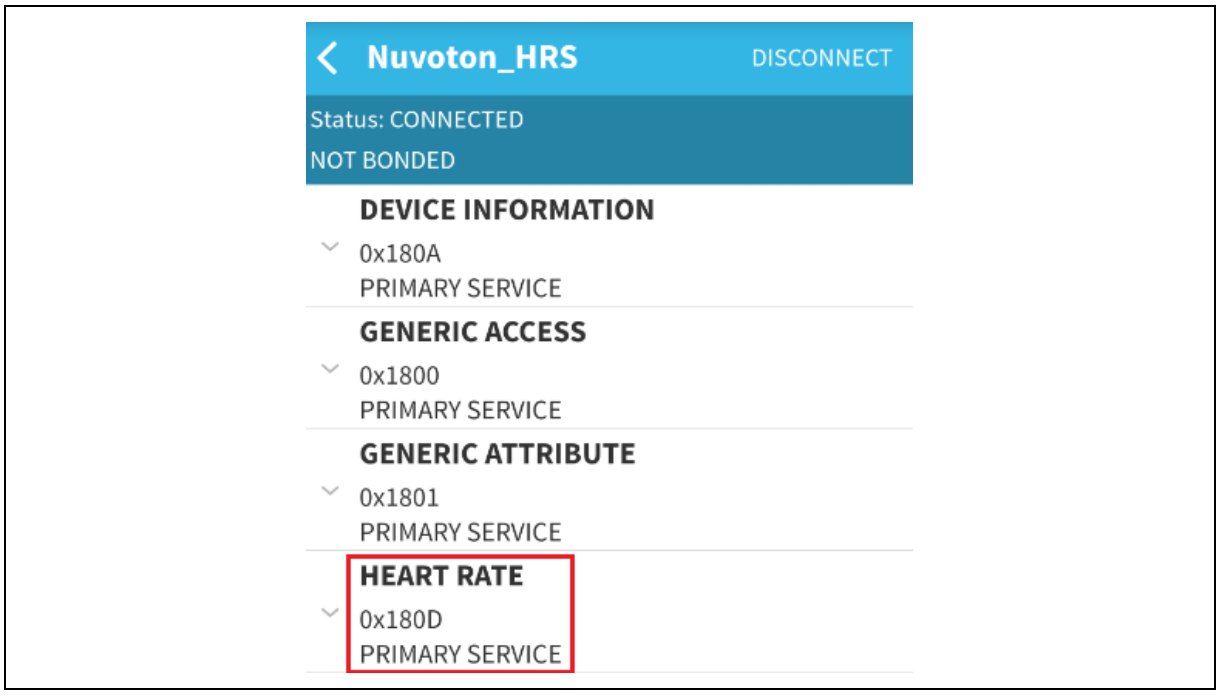

Figure 4-11 Services in Nuvoton_HRS

3. Click the “HEART RATE” service, and you can see all characteristics. Find the “HEART RATE MEASUREMENT” and click the NOTIFY button, then you can get the heart rate measurement value. Find the “BODY SENSOR LOCATION” and click the READ button, then you can get the sensor location is CHEST, as shown in Figure 4-12.

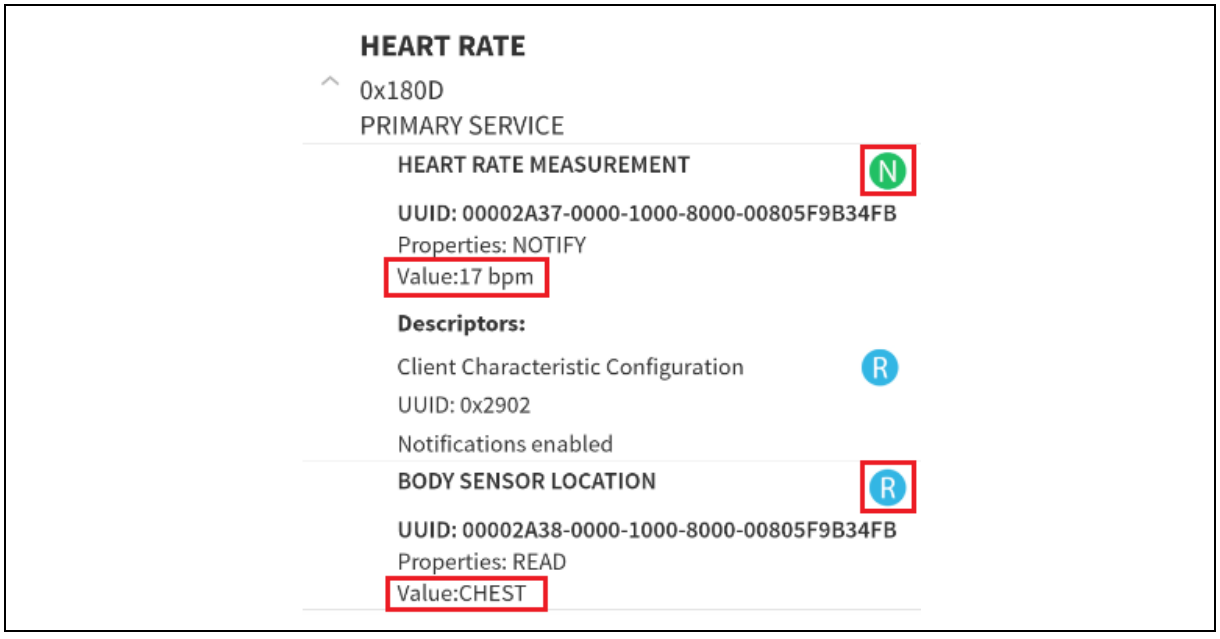

Figure 4-12 Heart Rate Service

4.4 HOGP_Peripheral

The HOGP (HID Over GATT Protocol) is used to connect human interface devices such as keyboard and mouse to a host device. This demo contains mouse, keyboard and multimedia key features.

4.4.1 Using an Android Mobile Phone

Follow the steps below to test HOGP_Peripheral:

1. Run the demo. In the *BleAppSetting.h* file, if you select the IOCAPABILITY_SETTING to be DISPLAY_ONLY, you can see the following message in debug console, as shown in Figure 4-13. The message includes the Bluetooth Pairing Key. In this case, it is “654321”.

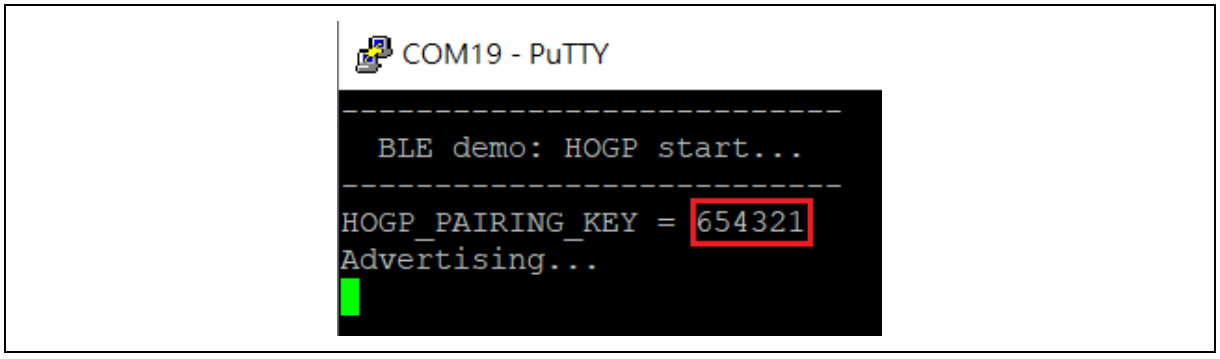

Figure 4-13 Pairing Key in Debug Message

2. Enter “Settings” screen and select “Bluetooth & device connection”. In Bluetooth page, select “Pair new device”, and then you can see the available devices, as shown in Figure 4-14.

Figure 4-14 Available Devices for BLE Pairing

3. Find the device “Nuvoton_HOGP” and click it. You can see that it asks you to enter the PIN code to pair, as shown in Figure 4-15. Please input the pairing key you get in Step 1 to connect the device.

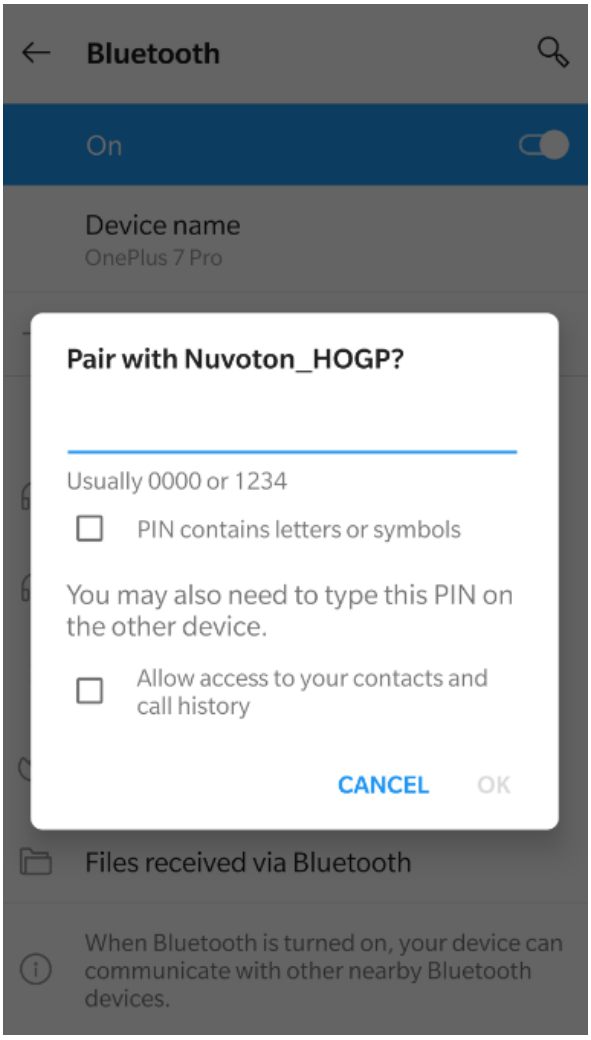

Figure 4-15 Input Bluetooth Pairing PIN in Mobile

4. If you select the IOCAPABILITY_SETTING to be KEYBOARD_ONLY, you will see the Bluetooth Pairing Key is shown in mobile, as shown in Figure 4-16. Then, you need to input the pairing key in debug console of device side to connect.

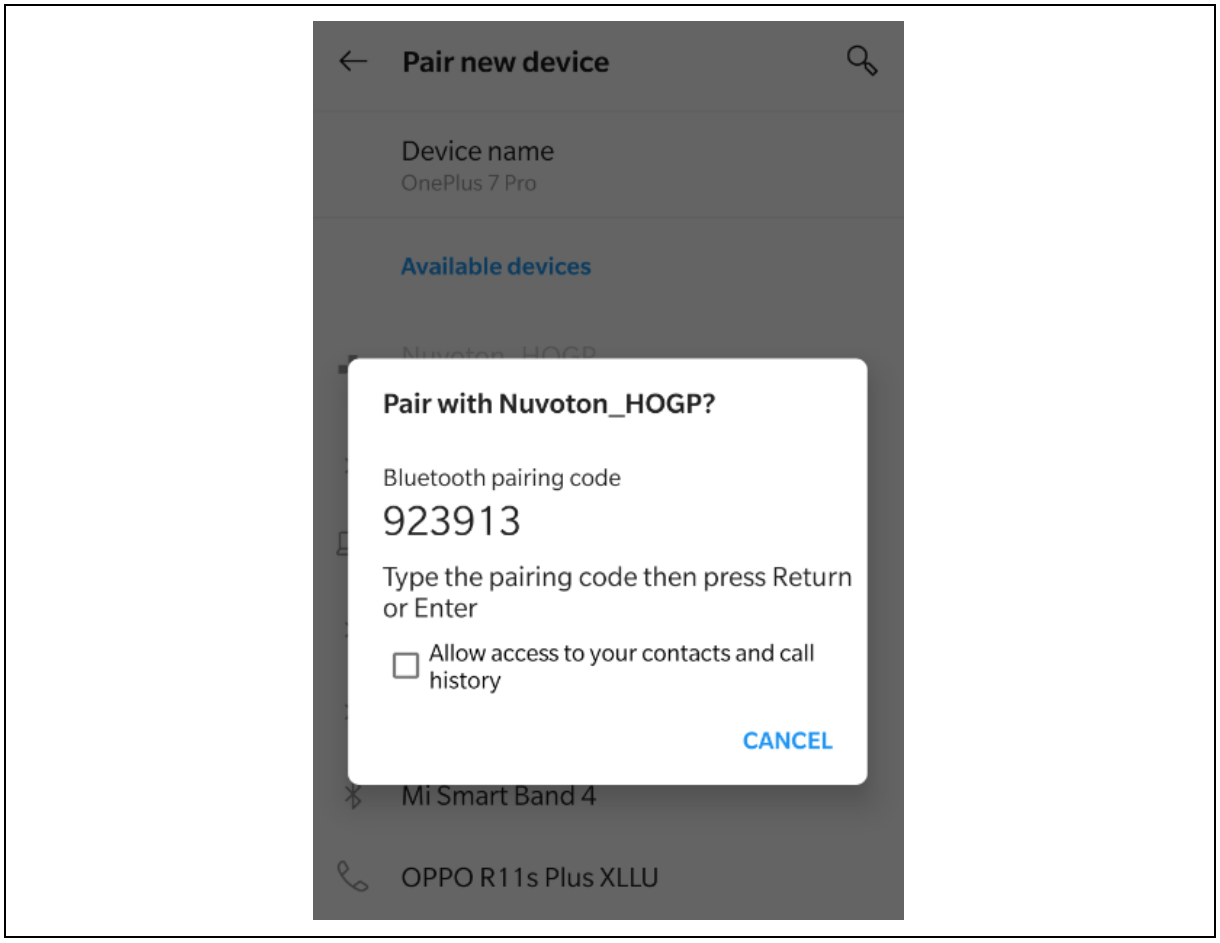

Figure 4-16 Show Bluetooth Pairing Code in Mobile

5. After connected to the device, you can see the HID demonstrate automatically. In the screen, you can see the mouse cursor and volume control screen, as shown in Figure 4-17. The HOGP_mouse controls cursor to draw a diamond slowly, at the moment cursor moved to the top, and HOGP_consumer controls volume.

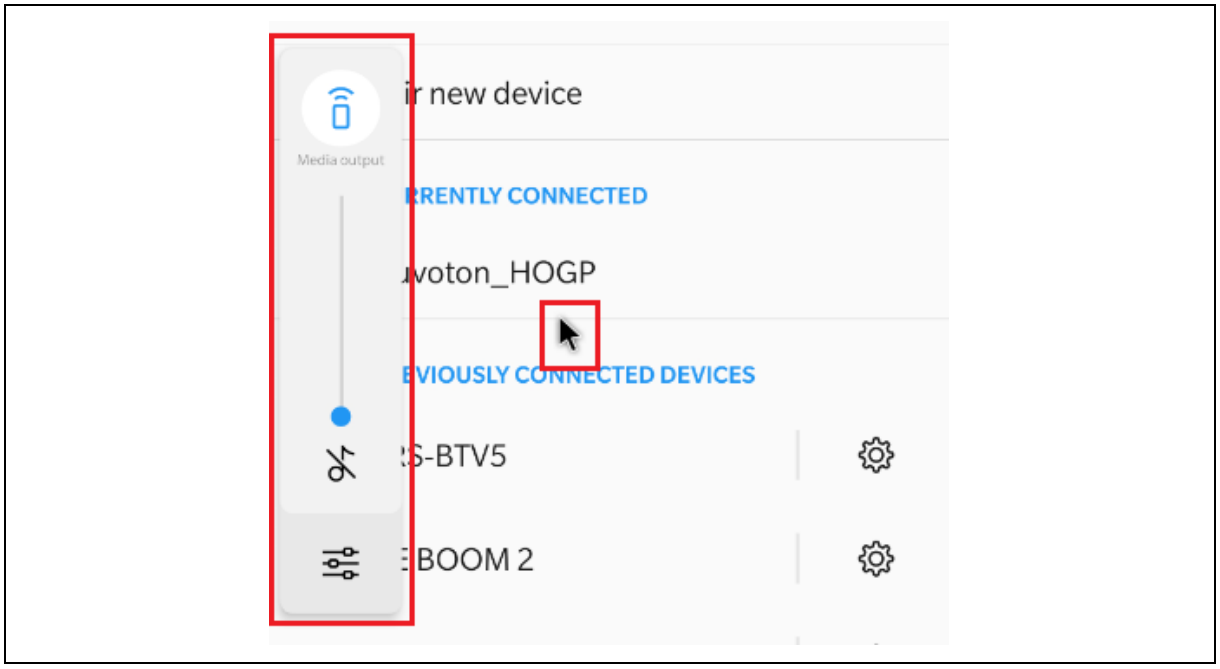

Figure 4-17 HID Mouse and Consumer Demonstration

6. If user opens a keyboard input-able application (e.g. Google search) on mobile phone, HOGP_keyboad repeats to input keys 'a' 'b' ~ 'z' ~ '1' '2'~ '9', as shown in Figure 4-18.

Figure 4-18 HID Keyboard Demonstration

7. The pairing security key is stored in the embedded Flash. If you re-flash the image, it will erase the security. Please note that you need to forget the old device first and then connect the new device to pair again. To forget the device, you can do it after clicking Nuvoton_HOGP item, as shown in Figure 4-19.

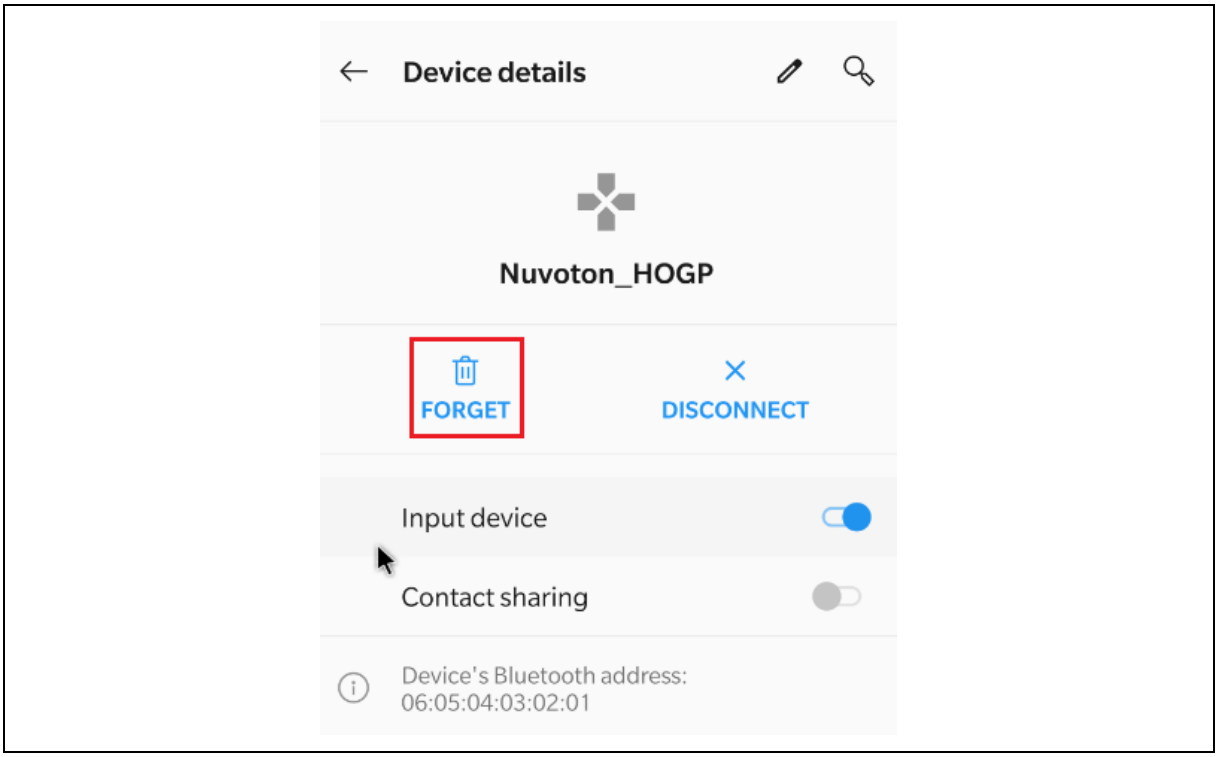

Figure 4-19 Forget the HOGP Device

4.5 DataRate_Peripheral

The DataRate_Peripheral is a demo that lets NuMaker-M03xBT board and mobile App do bidirectional data transfer via BLE. Since TRSP is not a standard BLE profile, please use the Nuvoton NuBLE App to do test.

4.5.1 Testing DataRate_Peripheral with Nuvoton NuBLE App

Follow the steps below to test DataRate_Peripheral:

1. Start a terminal software (e.g. Putty) and then open the VCOM of NuMaker-M03xBT board, where the UART baud rate is 115200. You will see the following message after booting, as shown in Figure 4-20.

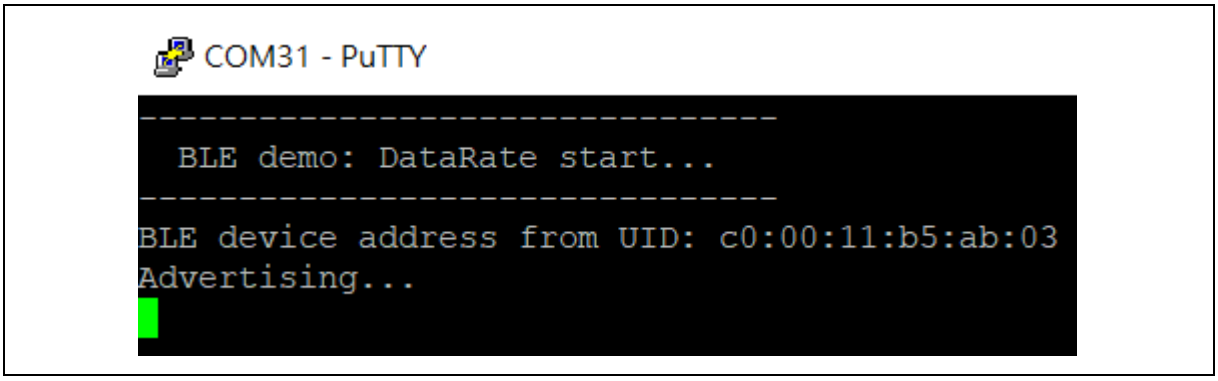

Figure 4-20 DataRate Boot Message

2. Open NuBLE App and change to the DataRate tab, and you can see the following screen, as shown in Figure 4-21. For the performance measurement, changing a setting may cause a different test result. There are some factors that can get better performance, such as larger packet length, lower packet interval and higher PHY mode.

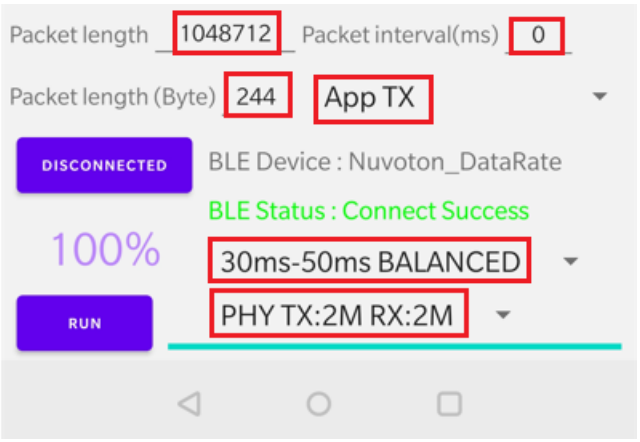

Figure 4-21 NuBLE DataRate Settings

3. Before doing the testing, you must select either “App TX” or “Device TX” to decide the data transfer direction, as shown in Figure 4-22.

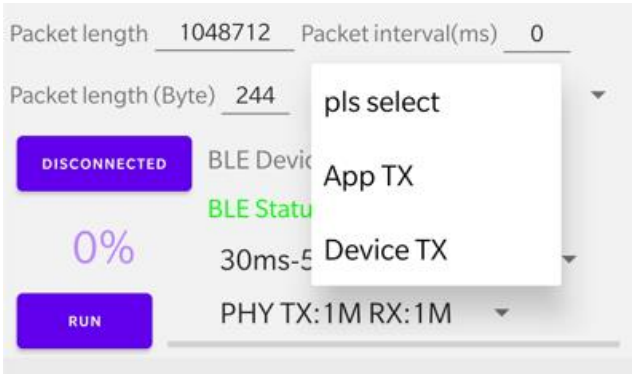

Figure 4-22 DataRate Transfer Mode

4. Clicking the RUN button can start a test. After the test is done, you can see the result in console, as shown in Figure 4-23.

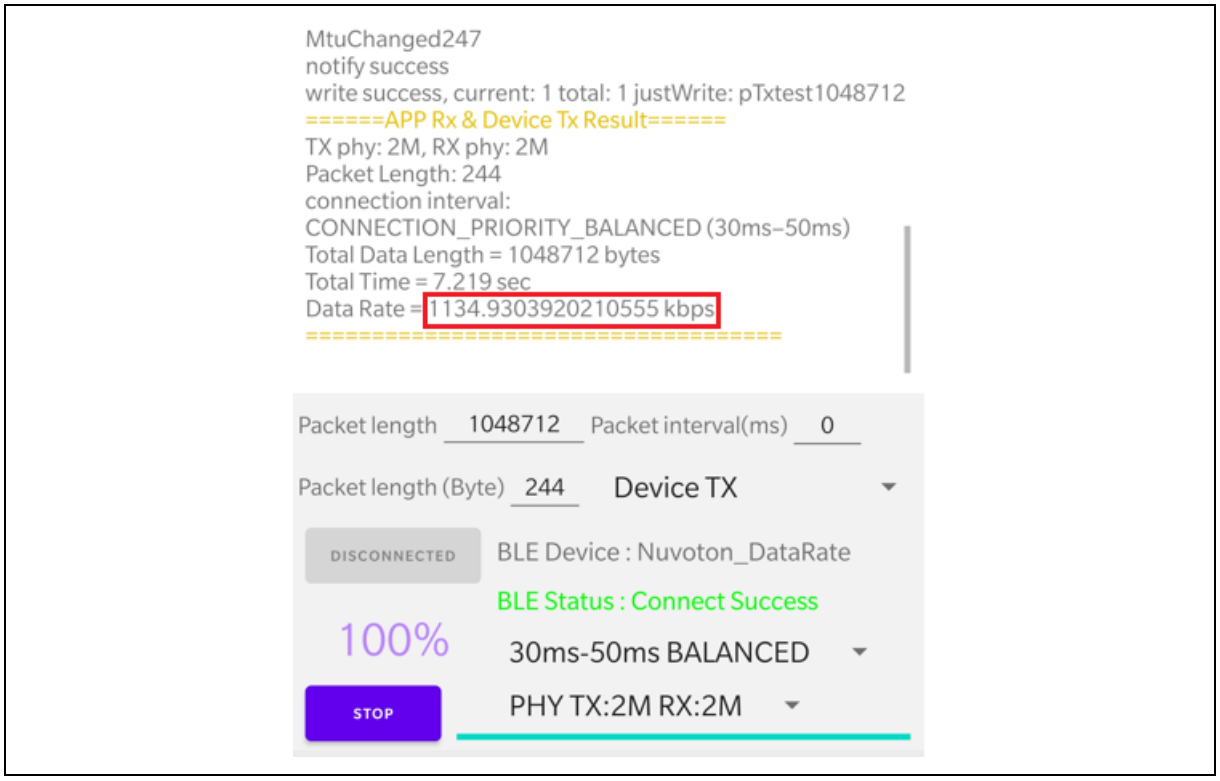

Figure 4-23 DataRate Device TX Test

4.6 TRSP_UART_Central

The TRSP_UART_Central is a demo that works with the TRSP_UART_Peripheral demo. These two demos can make two NuMaker-M03xBT boards do bidirectional data transfer via BLE and UART. The central and peripheral test block diagram is shown as Figure 4-24.

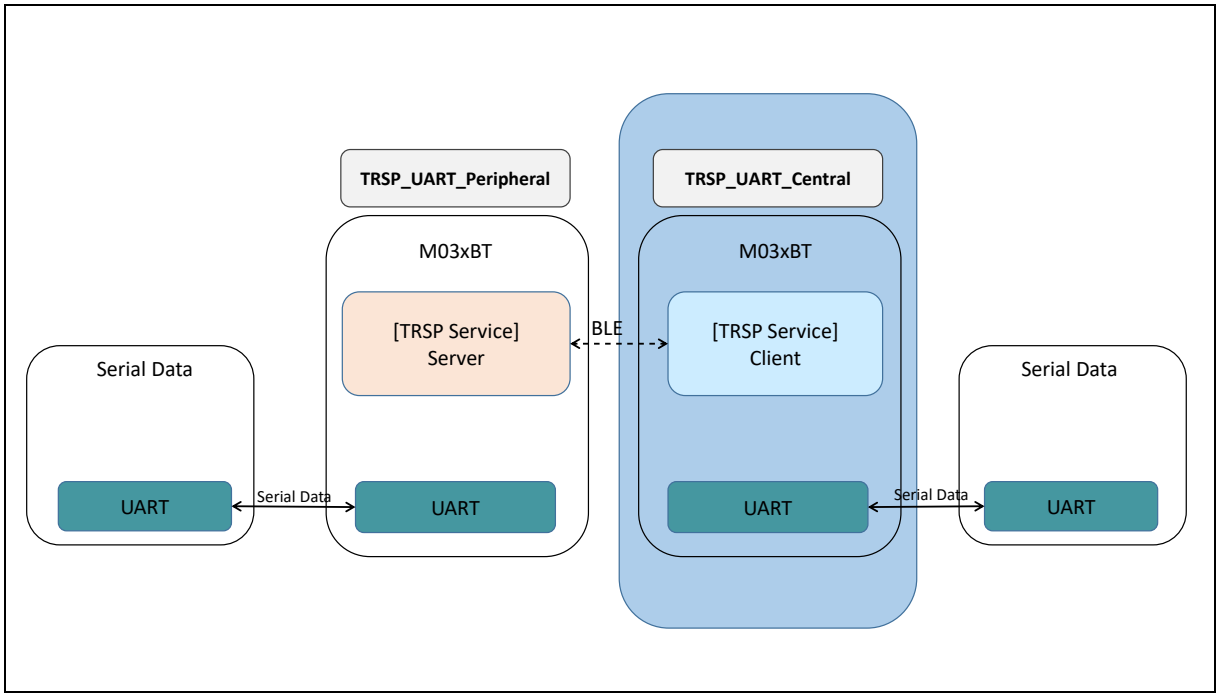

Figure 4-24 TRSP_UART Central and Peripheral Test

4.6.1 Testing TRSP_UART_Central with TRSP_UART_Peripheral

Follow the steps below to test TRSP_UART_Central:

1. Start the first NuMaker-M03xBT board that is running the TRSP_UART_Peripheral demo.
2. Start the second NuMaker-M03xBT board that is running the TRSP_UART_Central demo. You will see the TRSP_UART central enables scan and creates BLE connection if the target peripheral device found automatically, as shown in Figure 4-25.

```
----------------------------------------
  BLE demo: TRSP_UART(Central) start.....
----------------------------------------
BLE device address from UID: c0:00:09:7f:3b:91
Scanning...
Found [Name:Mi ColorS 9390] [Address: 44:27:f3:0c:93:90]
Found [Name:Nuvoton_UART] [Address: c0:00:11:d0:f1:01]
Found device and stop scanning...
Connecting...
Found [Name:CX 400BT TW] [Address: 00:1b:66:dc:88:7c]
Status=0, ID=0, Connected to c0:00:11:d0:f1:01
Exchange MTU, ID:0, size: 23
Data Length, Tx=27, Rx=27
cccd write completed!
Manu Name:Nuvoton
Firmware Rev:01.1
```

Figure 4-25 TRSP_UART_Central Boot Message

3. For sending data from central to peripheral, you can type string “client send test” + ENTER

in central console. You will see the string in peripheral console, as shown in Figure 4-26.

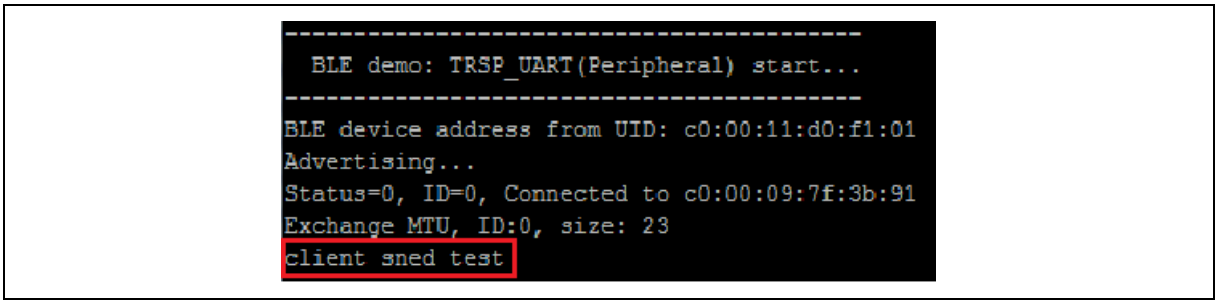

Figure 4-26 TRSP_UART Peripheral to Central Screen

4. For sending data from peripheral to central, you can type string “server send test” + ENTER in peripheral console. You will see the string in central console, as shown in Figure 4-27.

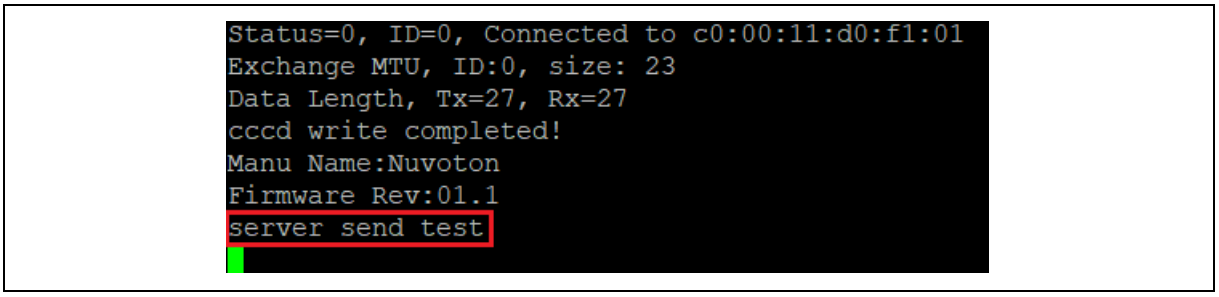

Figure 4-27 TRSP_UART Central to Peripheral Screen

4.7 TRSP_UART_Multi_Central

The TRSP_UART_Multi_Central is a demo that can work with multiple (at most 3) TRSP_UART_Peripheral demos. These two kinds of demos can make NuMaker-M03xBT board do bidirectional data transfer via BLE and UART. For each peripheral side, it can send data to the central side. For the central side, it can broadcast data to all of connected peripheral sides.

4.7.1 Testing TRSP_UART_Multi_Central with TRSP_UART_Peripheral

Follow the steps below to test TRSP_UART_Multi_Central:

1. Start the first NuMaker-M03xBT board that is running the TRSP_UART_Multi_Central demo. You will see the central creates the link 0 to do scan, as shown in Figure 4-28.

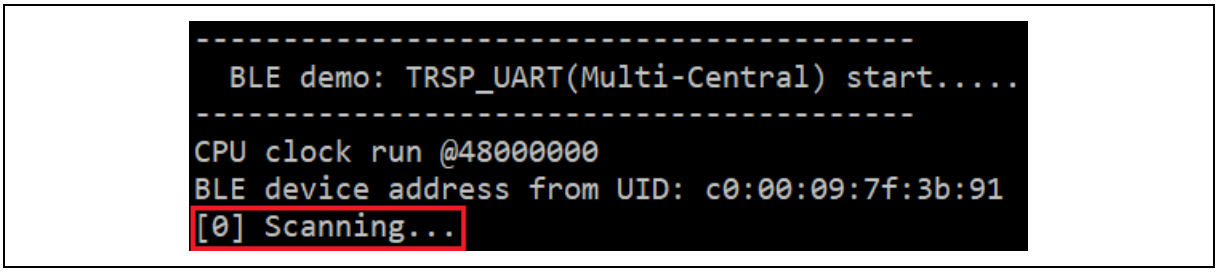

Figure 4-28 TRSP_UART_Multi_Central Boot Message

2. Start the new NuMaker-M03xBT board that is running the TRSP_UART_Peripheral demo. After the central is scanned and connected to the first peripheral, you will see the central creates the link 1 to do scan again, as shown in Figure 4-29.

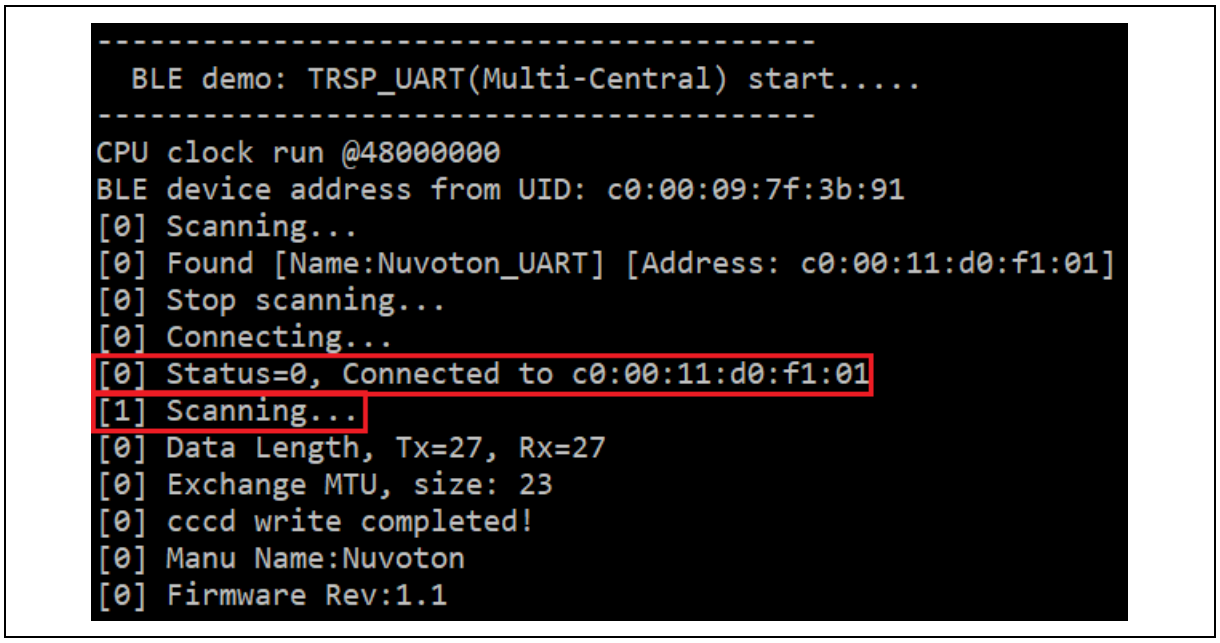

Figure 4-29 TRSP_UART_Multi_Central Connected to Peripheral

3. Repeat Step 2 to add another new peripheral; the maximum number of the peripheral is 3.

4. For sending data from peripheral to central, you can type string “server send test” + ENTER in first peripheral console. You will see the string in central console with the link number prefixed, as shown in Figure 4-30.

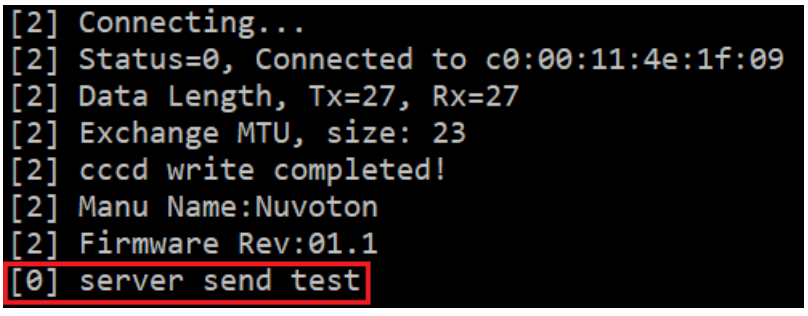

Figure 4-30 TRSP_UART Peripheral to Central Screen

5. For sending data from central to peripheral, you can type string “client send test” + ENTER in central console. You will see the string in all of connected peripheral consoles, as shown

in Figure 4-31.

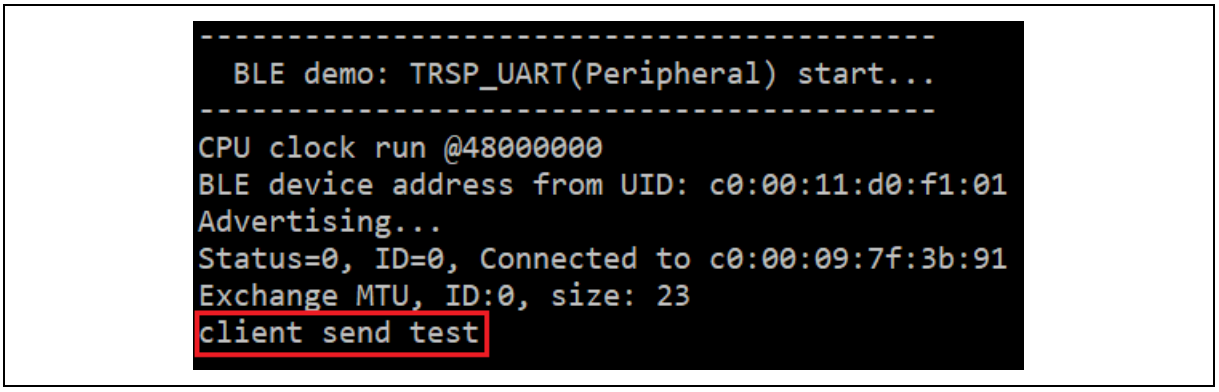

Figure 4-31 TRSP_UART Central to Peripheral Screen

4.8 TRSP_UART_Multi_Peripheral

The TRSP_UART_Multi_Peripheral is a demo that can work with multiple (at most 3) TRSP_UART_Central demos or NuBLE Apps. These two kinds of demos can make NuMaker-M03xBT boards or smart phones do bidirectional data transfer via BLE and UART. For each central side, it can send data to the peripheral side. For the peripheral side, it can broadcast data to all of connected central sides.

4.8.1 Testing TRSP_UART_Multi_Peripheral with TRSP_UART_Central

Follow the steps below to test TRSP_UART_Multi_Peripheral:

1. Start the first NuMaker-M03xBT board that is running the TRSP_UART_Multi_Peripheral demo. You will see the peripheral creates the link 0 to do advertising, as shown in Figure 4-32.

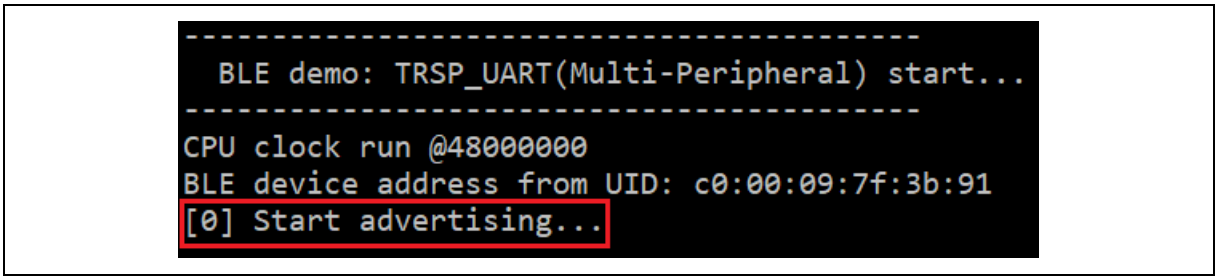

Figure 4-32 TRSP_UART_Multi_Peripheral Boot Message

2. Start the new NuMaker-M03xBT board that is running the TRSP_UART_Central demo. After the first central is scanned and connected to the peripheral, you will see the

peripheral creates the link 1 to do advertising again, as shown in Figure 4-33.

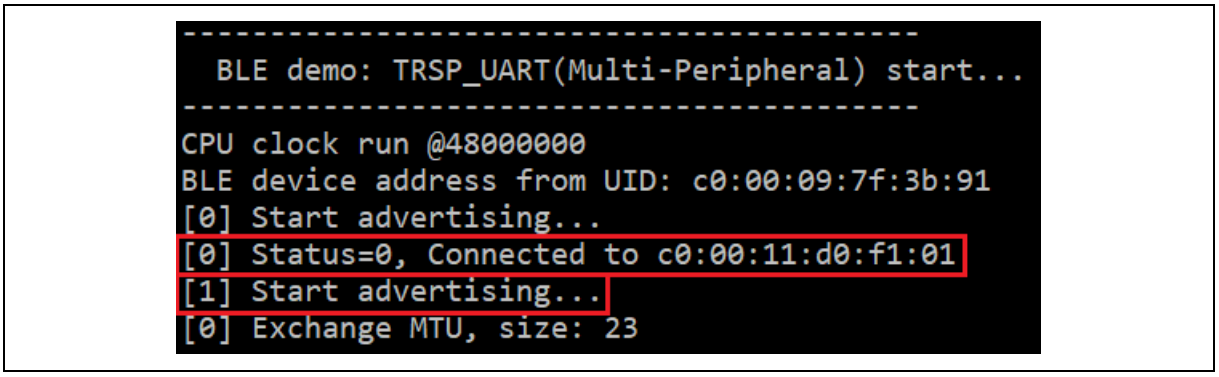

Figure 4-33 TRSP_UART_Multi_Peripheral Connected from Central

3. Repeat Step 2 to add another new central; the maximum number of the central is 3.
4. For sending data from central to peripheral, you can type string “client send test” + ENTER in first central console. You will see the string in peripheral console with the link number prefixed, as shown in Figure 4-34.

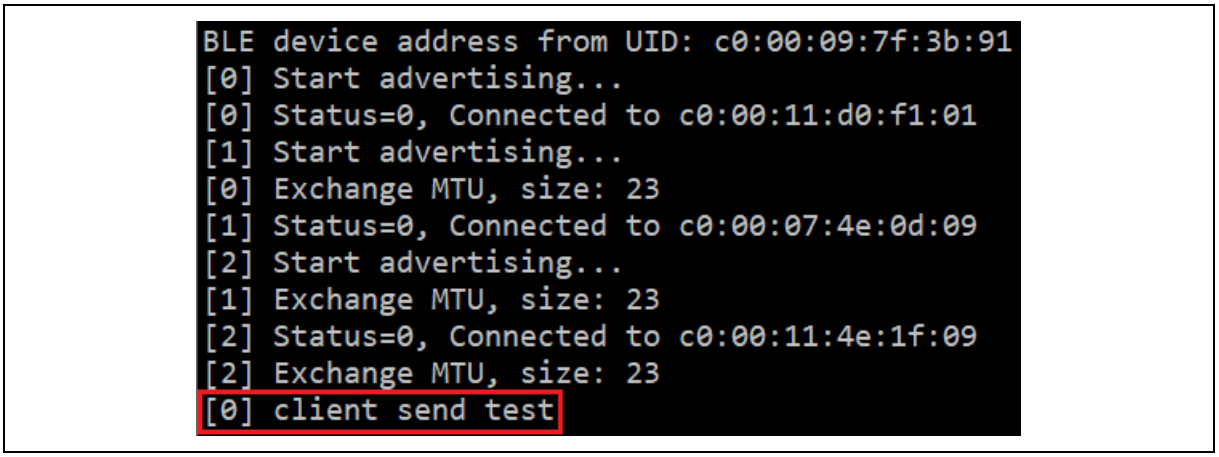

Figure 4-34 TRSP_UART Central to Peripheral Screen

5. For sending data from peripheral to central, you can type string “server send test” + ENTER in peripheral console. You will see the string in all of connected central consoles,

as shown in Figure 4-35.

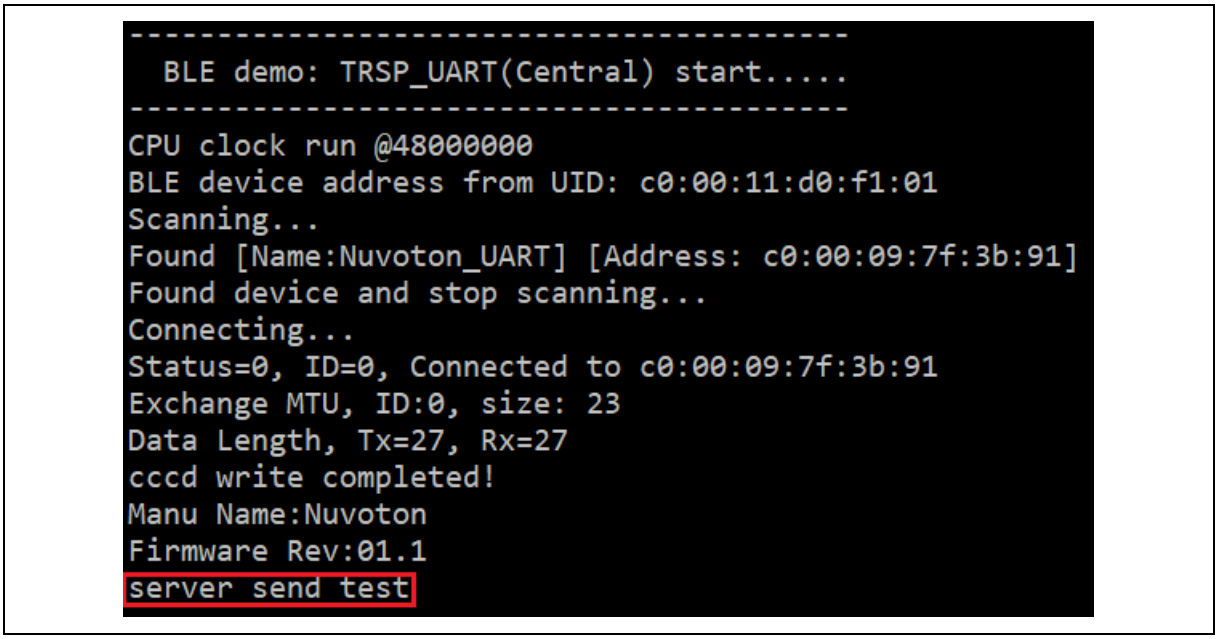

Figure 4-35 TRSP_UART Peripheral to Central Screen

5 BLE API

To see the Nuvoton BLE API documentation, please start a browser (e.g. Chrome) and open the file from *M031BSP\SampleCode\NuMaker-M03xBT\BLE\Doc\API Reference\index.html*.

5.1 Commonly Used Functions

You can find the commonly used functions at links below, as shown in Figure 5-1.

- BLE_API → Modules → RF PHY → RF PHY Function.
- BLE_API → Modules → BLE Common → BLE Command Function → BLE Command Function.
- BLE_API → Modules → BLE Common → BLE Command Function → BLE DTM Function for MP.
- BLE_API → Modules → BLE Common → BLE Event Definition.
- BLE_API → Modules → BLE Services → BLE Service Based Definitions → BLE Based Function.

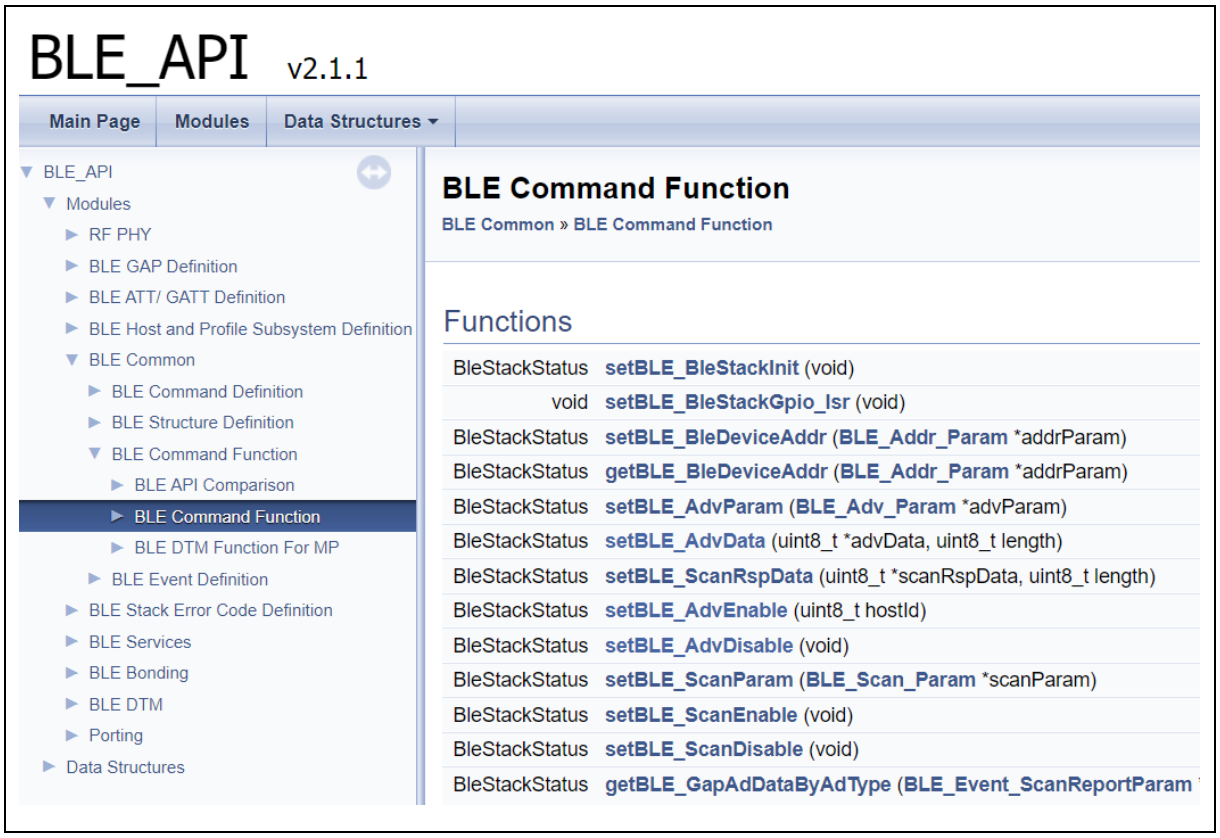

Figure 5-1 BLE API Documentation

Revision History

Date	Revision	Description
2020.04.30	1.00	1. Initially issued.
2020.05.29	1.01	1. Updated section 3.5.
2020.08.27	1.02	1. Supported installing NuBLE App from Internet. 2. Supported standard BLE security management
2021.07.22	1.03	1. Supported M032BT. 2. Supported DataRate demo. 3. Supported using NuBLE App v2.2.x.
2021.09.01	2.00	1. Supported SDK v2.
2021.09.15	2.01	1. Fixed the title and picture style.
2021.12.23	2.02	1. Added demo description. 2. Supported TRSP_UART_Multi_Central and TRSP_UART_Multi_Peripheral demo. 3. Made editorial changes.

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.