M031BT/M032BT FOTA Update User Guide

Application Note for 32-bit NuMicro® Family

Document Information

Abstract	This document introduces how to do M031BT/M032BT FOTA update.
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 3

2 FOTA OVERVIEW 3

2.1 Memory Layout 3

2.2 FOTA Operation Flow 5

3 TESTING THE FOTA DEMO 7

3.1 Creating Project Images 7

3.2 Creating an FOTA Image 7

3.3 Downloading the OTA Image to Smart Phone 9

3.4 Programming the Images to BLE Device 10

3.5 Running the FOTA Demo on BLE Device 12

3.6 Using App to do OTA Action 12

4 FOTA COMMUNICATION FLOW 16

4.1 FOTA Command 16

4.2 FOTA Data 19

5 PORTING FOTA RELATED FUNCTIONS 22

5.1 Bootloader 22

5.2 FOTA Firmware Merge 23

5.2.1 BleFota_TimerTick() 23

5.2.2 BleFota_Disconnect() 23

5.2.3 Enabling Data Length Extension Feature 23

1 Introduction

This document shows how to use NuMaker-M03xBT board and OTA demo to do the firmware over-the-air (FOTA) update.

2 FOTA Overview

The following sections describe the Flash memory layout and operation flow.

2.1 Memory Layout

The Flash memory layout is shown in Figure 2-1. The BLE SDK utilizes dual-bank Flash to perform FOTA updates to ensure a valid image is activated. Bank 0 partition of the dual-bank Flash is configured as an active firmware (FW) block, and Bank 1 partition is configured as the FW update buffer.

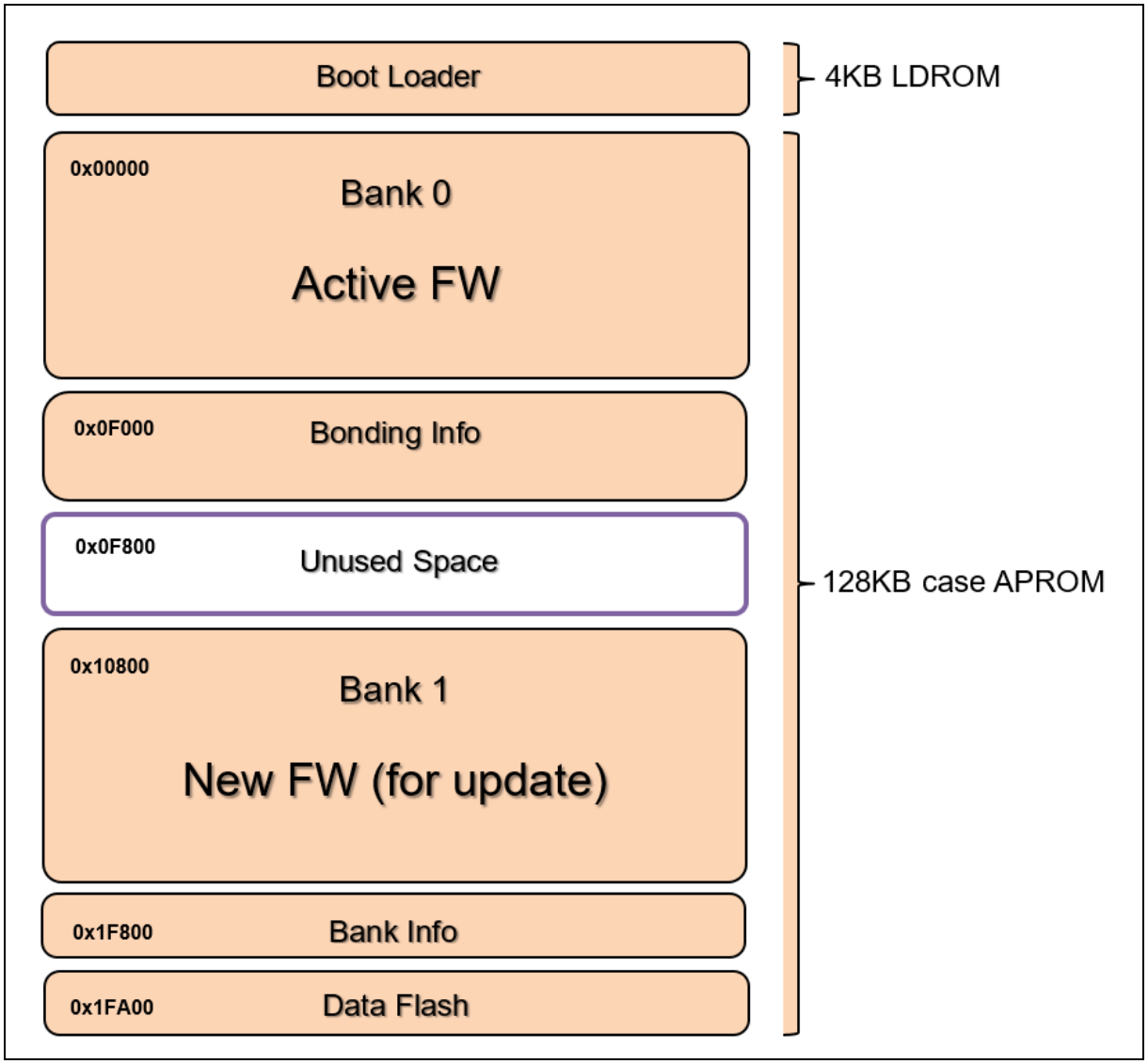

Figure 2-1 Flash Memory Layout

The Bank Info stores the information of the FW update buffer (Bank 1). After a new FW is completely written to Bank 1, the new FW information is written into Bank INFO. Then the system needs to reboot so the bootloader can check this page for completing the final step of FW update.

For the partition location, the Data Flash partition is consisted of the last three pages of APROM by default, and the Bank Info partition contains only one page. The Bonding Info is consisted of four pages, and the address is defined in the *porting_flash.h* file. The size of Bank 0 partition is from 0x0 to the head of Bonding Info partition, the Bank 1 partition is close to the Bank Info partition, and the size is equal to the Bank 0 partition.

In APROM 128KB, if you want to fully utilize the Flash space, you can change the Bonding Info start address from 0x0F000 to 0x0F800. In this case, the size of Bank0 and Bank1 is larger, and there is no more Unused Space.

2.2 FOTA Operation Flow

The FOTA operation flow is shown in Figure 2-2. A complete FOTA update is accomplished through the following steps:

1. Build the new FW image using KEIL/IAR/GCC IDE.
2. Use the FOTA PC tool to combine new FW image and system information into a FOTA image file.
3. Download the FOTA image file into a smart phone and use the App to parse the system information and new FW image.
4. Query system information of the BLE device and make a decision whether or not to update the new FW image.

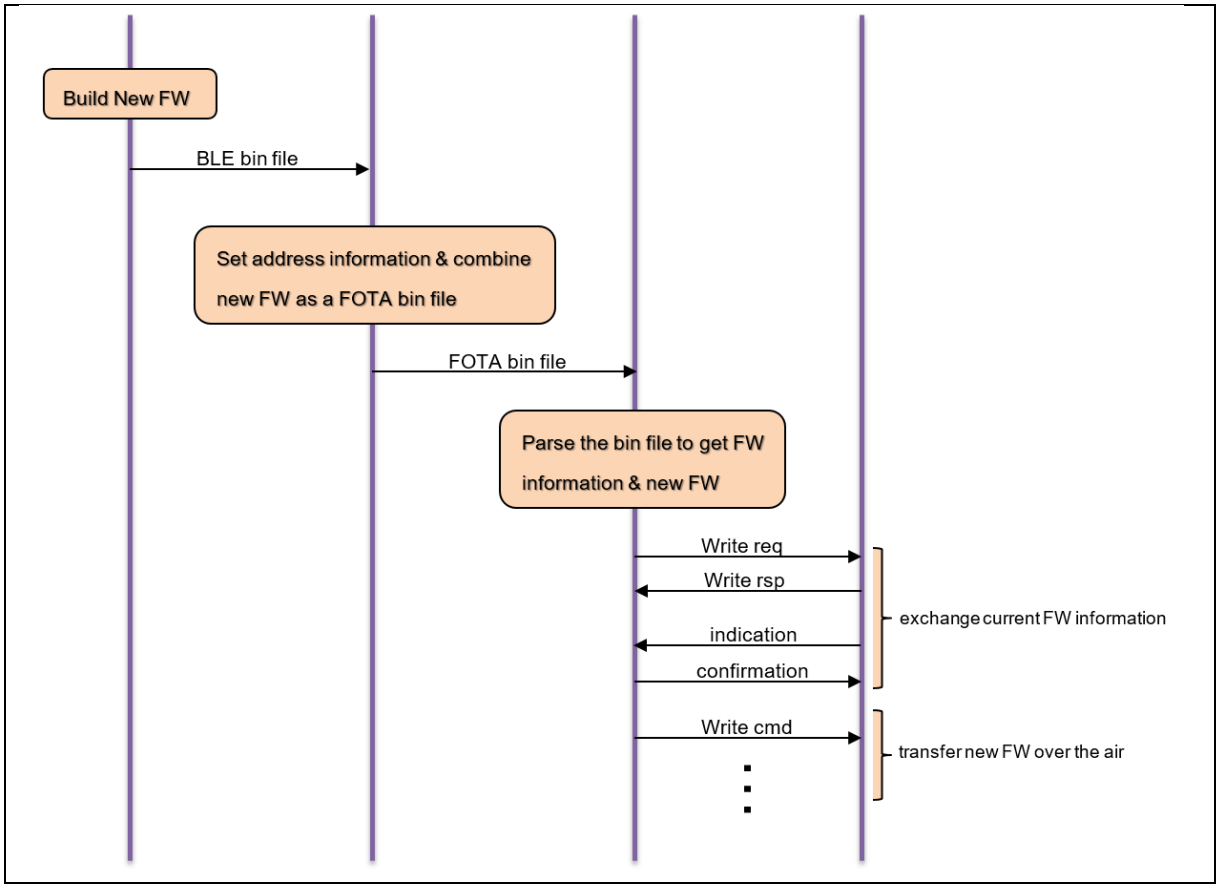

Figure 2-2 FOTA Operation Flow

3 Testing the FOTA Demo

This chapter describes how to do the FOTA test.

3.1 Creating Project Images

There are two projects used to build the related binaries. The first is the Other\FOTA_Bootloader, which is used for building the Bootloader. The second is the Peripheral\HRS_FOTA_Peripheral, which is used for building the active FW and new FOTA FW.

In HRS_FOTA_Peripheral project, the BLE System information is set by array SystemInfo[] which is defined in the *fota.c* file with 16 bytes maximum length. The string “Prefix” is defined for PC tool to search system information, and thus the string should not appear twice in the code.

```
const sys_information_t SystemInfo =
{
    {"Prefix"},
    {"sys ver 0001"}    /* BLE System Information */
};   /* An example of current system information, able to modify by user */
```

In NuBLE App, before doing a FOTA update, user must confirm the system information of new FW is different to the active FW. To do the FOTA test, you must modify the system information such as “sys ver 0002” for building the second binary. The first unmodified binary named *HRS_FOTA_Peripheral.bin* is for flashing directly, and the second binary renamed as *HRS_FOTA_Peripheral_2.bin* is for creating an FOTA image.

3.2 Creating an FOTA Image

To create an FOTA image, use the FOTA Tool as shown in Figure 3-1. In Step 1, user can specify the start address and end address of FOTA image. The tool will check the valid size and add padding zero to the end of image. For image address auto detection, user can ignore Step 1.

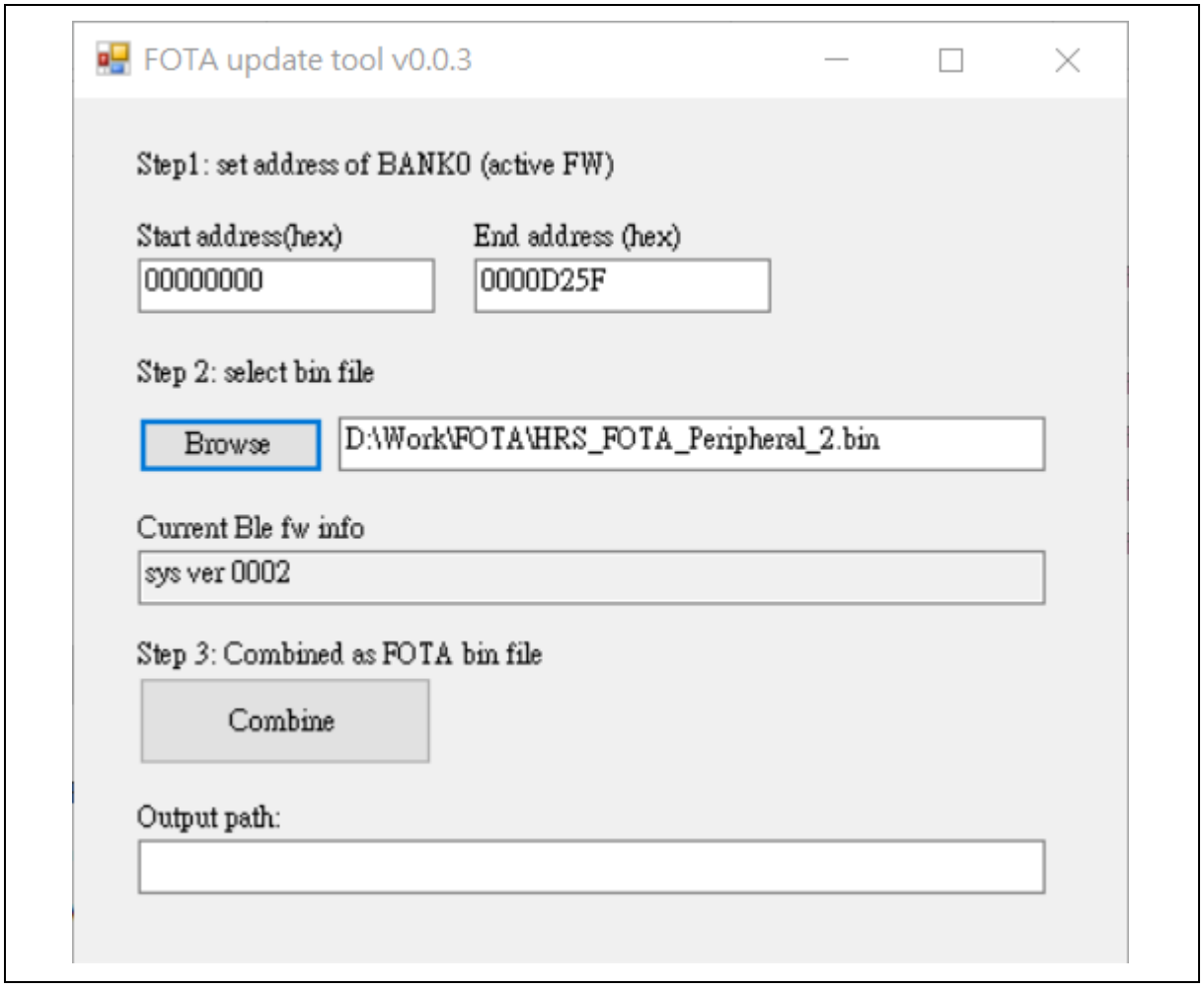

Figure 3-1 Using FOTA Tool

In Step 2, user needs to select the second binary (named *HRS_FOTA_Peripheral_2.bin*). After selecting the file, a window will pop-up as Figure 3-2 for address auto detection. It is suggested to click "**Yes**" to do address auto detection, which can create a smaller OTA image without the padding zero.

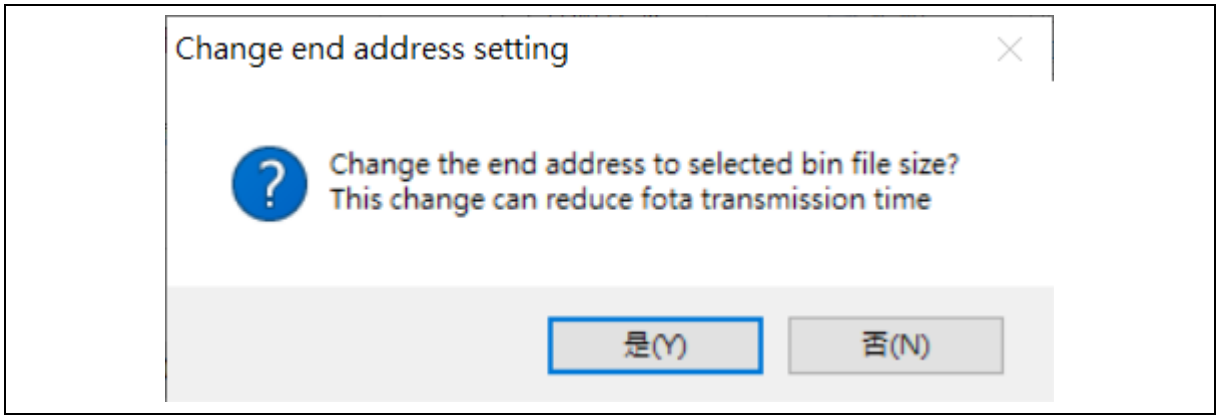

Figure 3-2 Address Auto Detection of FOTA Tool

In Step 3, user just needs to click the "**Combine**" button, and the FOTA image named *HRS_FOTA_Peripheral_2_OTA.bin* will be created in the same folder.

The FOTA Tool generates 32-bytes information for mobile App parsing, and the information is stored in the header of OTA image as Figure 3-3.

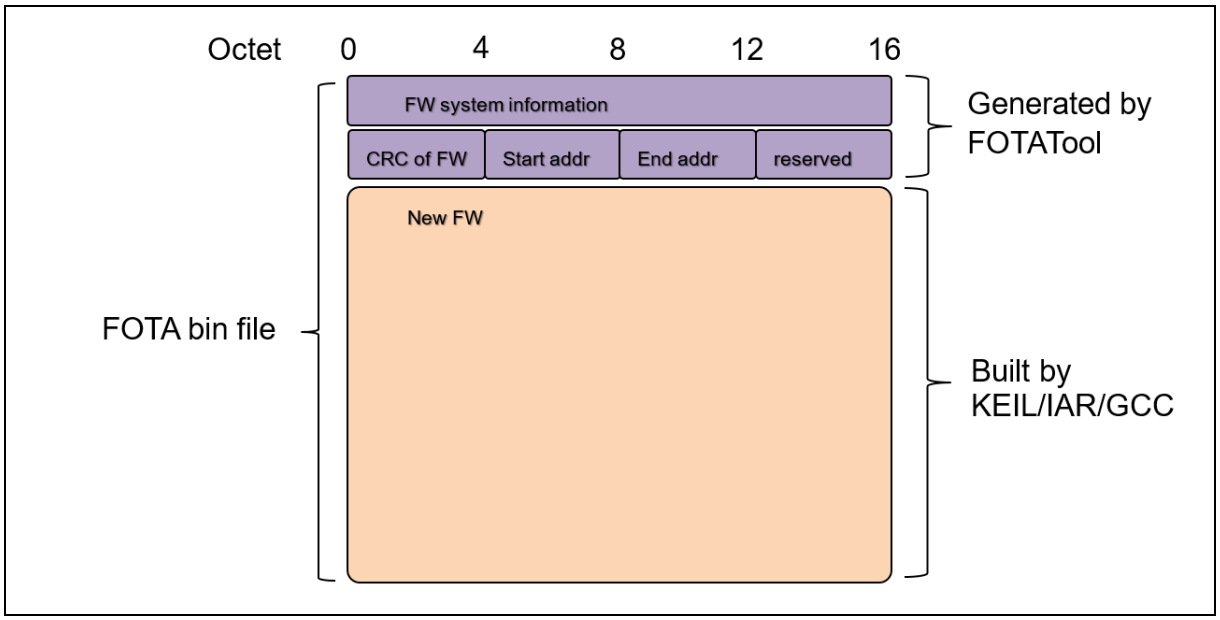

Figure 3-3 FOTA Tool Output Image

3.3 Downloading the OTA Image to Smart Phone

After downloading the OTA image to a smart phone, please move it to the specific folder *Download/fota_bin*. The detailed information is shown as Figure 3-4.

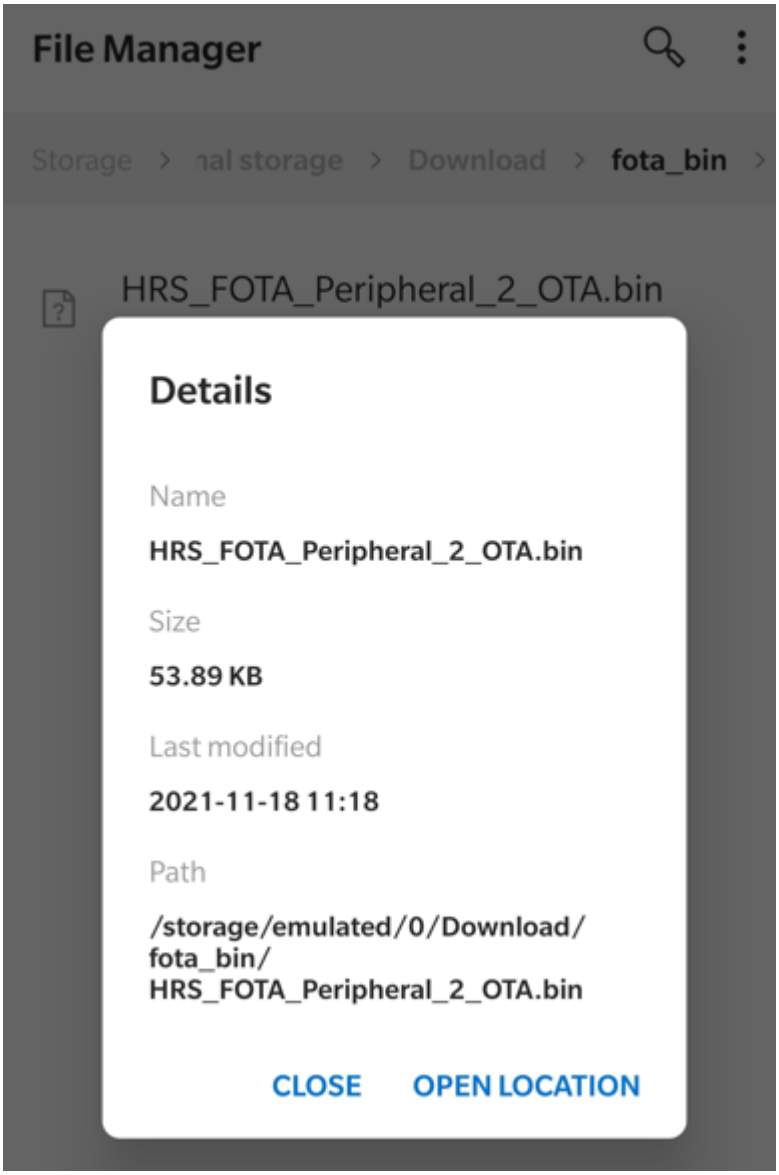

Figure 3-4 OTA Image Download File

3.4 Programming the Images to BLE Device

Use the *Nuvoton NuMicro ICP Programming Tool* to program the related images. Please enable the APROM, LDROM and Setting items. The APROM is for Bank 0 active image, and the LDROM is for Bootloader, as shown in Figure 3-5.

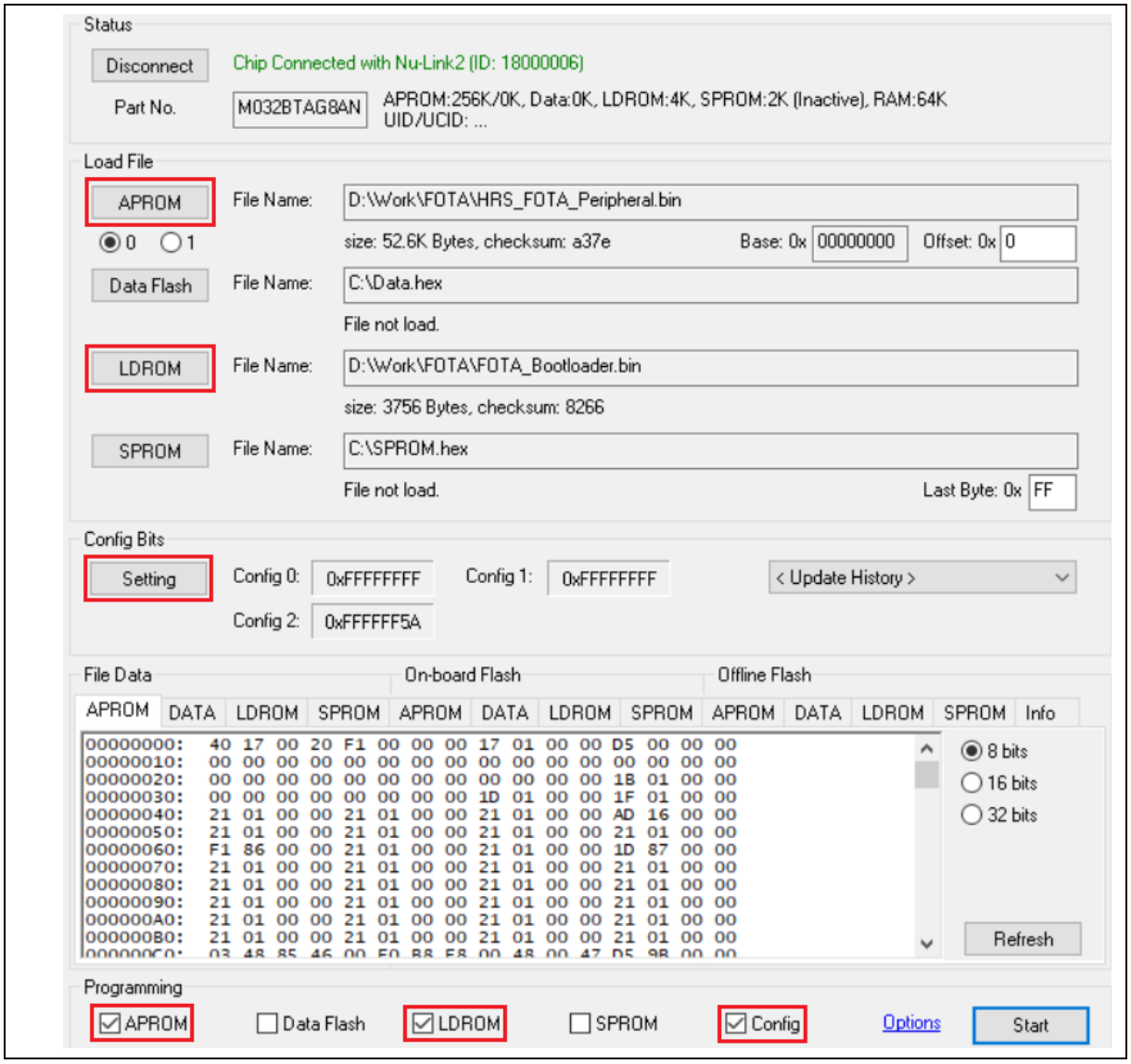

Figure 3-5 ICP Programming Tool

Before starting programming the images, please click the "**Select**" option to change the boot option to be "LDROM", shown as Figure 3-6. This makes sure the OTA Bootloader will be executed after system reset.

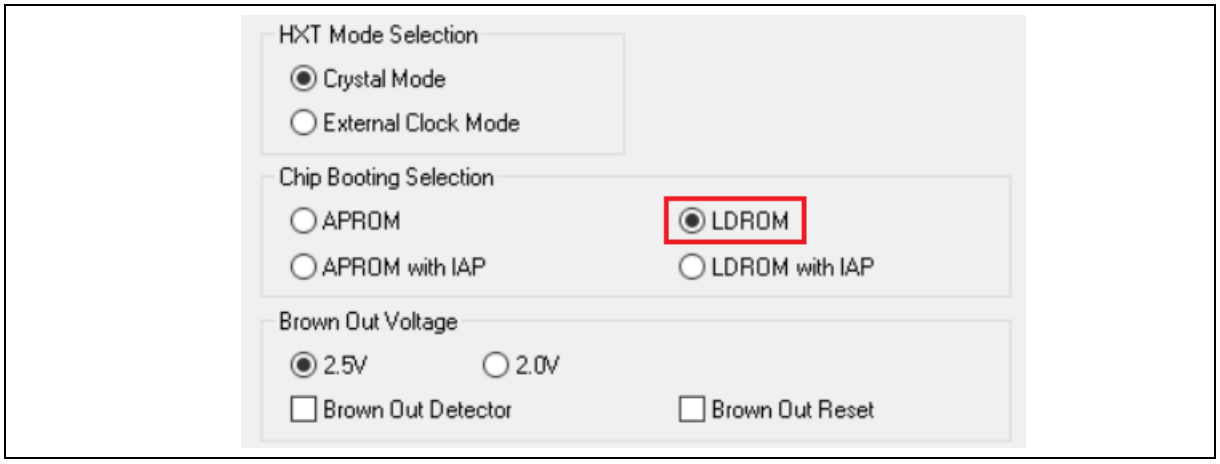

Figure 3-6 Boot Selection

3.5 Running the FOTA Demo on BLE Device

After programming the images, user can see the partition base addresses and system information of active FW, shown as Figure 3-7.

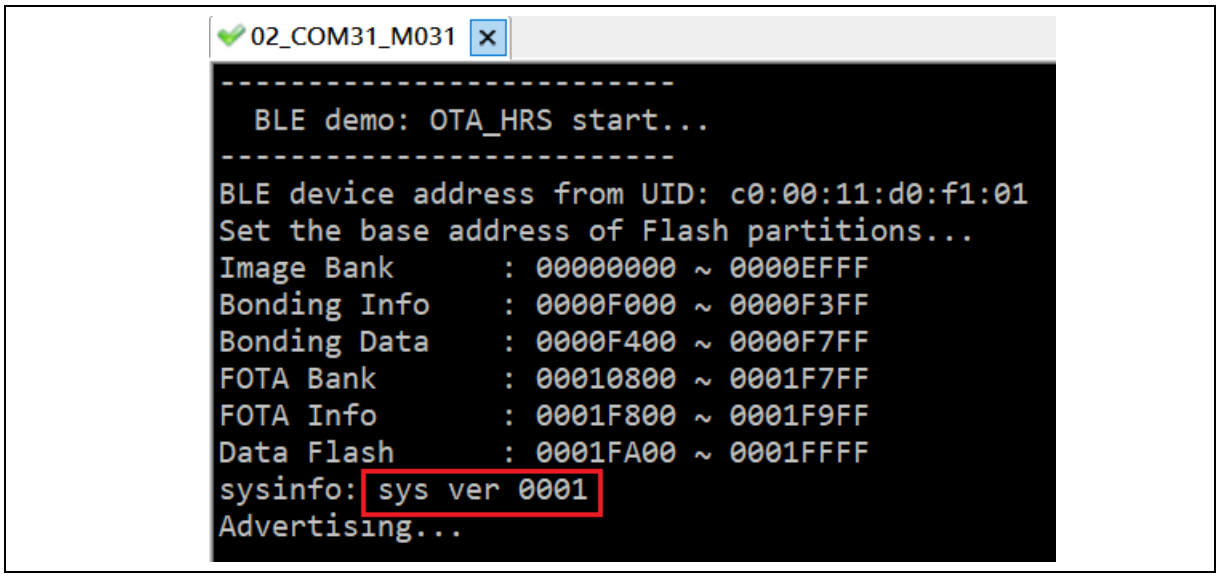

Figure 3-7 Booting Message before FOTA

3.6 Using App to do OTA Action

The NuBLE App supports several features for different demos. After starting the App, you should change the tab to OTA feature, shown as Figure 3-8.

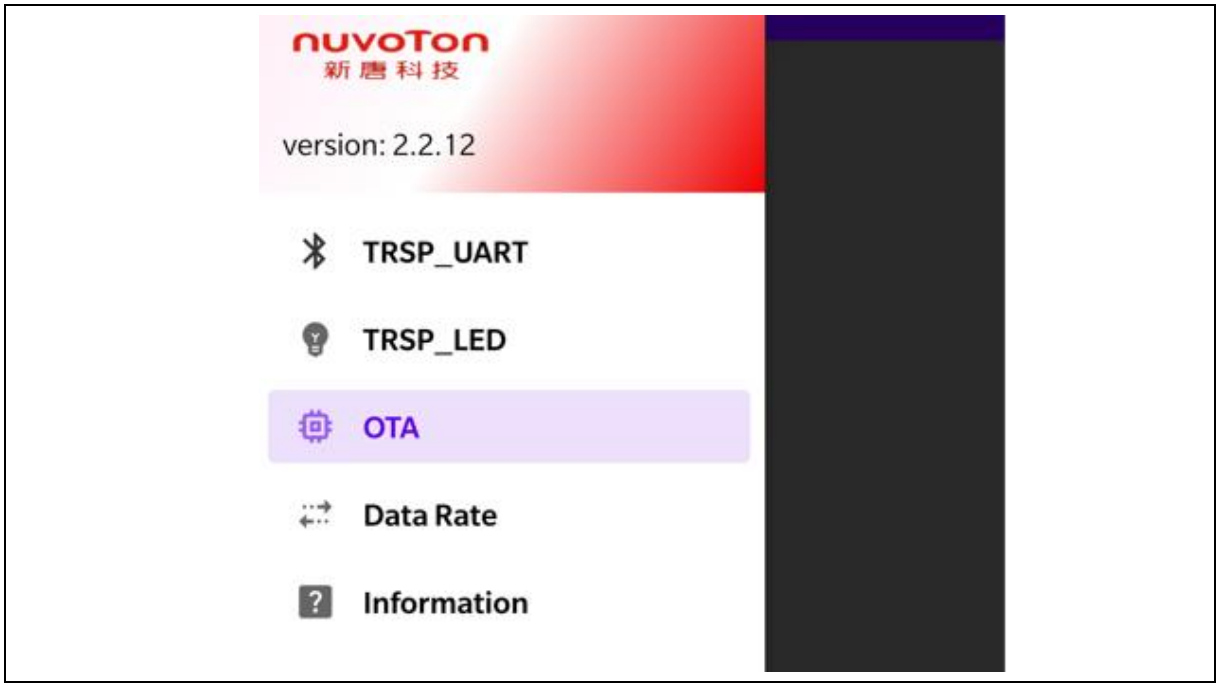

Figure 3-8 OTA Tab in App

Then you can do BLE scanning and connect to the OTA device, shown as Figure 3-9.

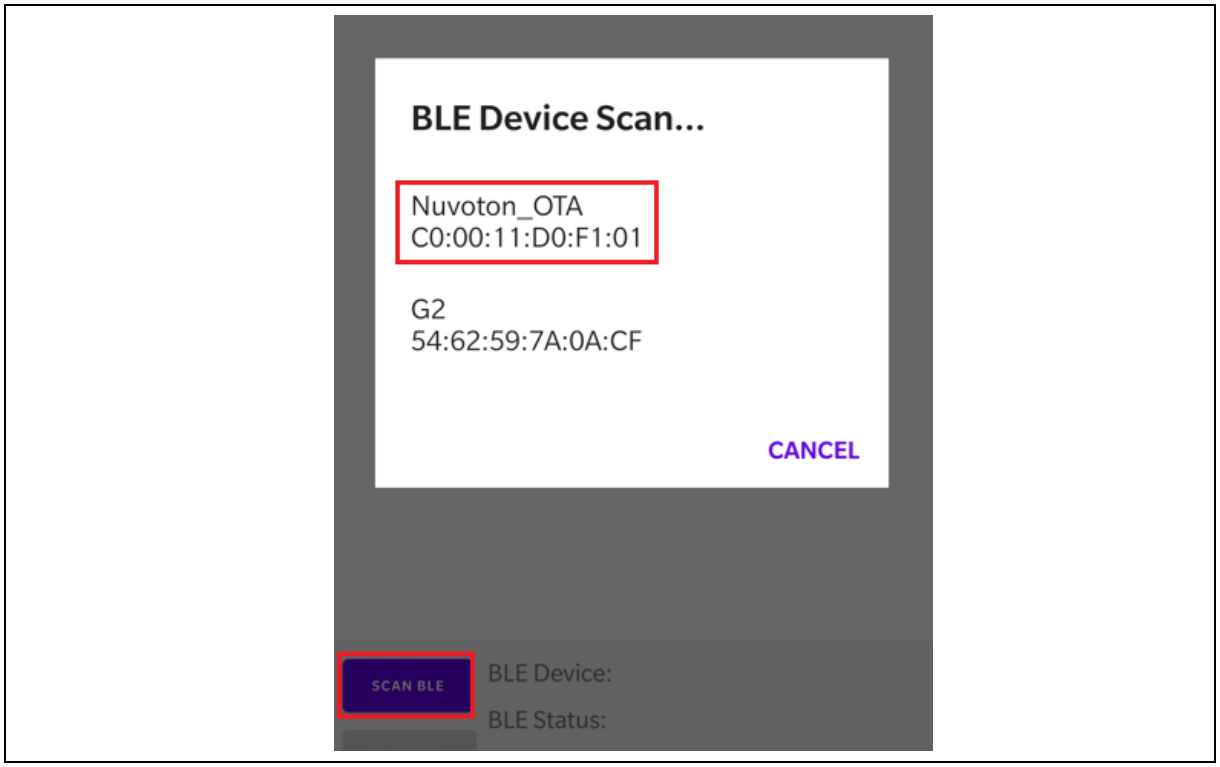

Figure 3-9 Scan and Connect in App

After connecting to the device, you can select the new OTA FW image for updating. Please confirm the image is located in folder *Download\fota_bin*.

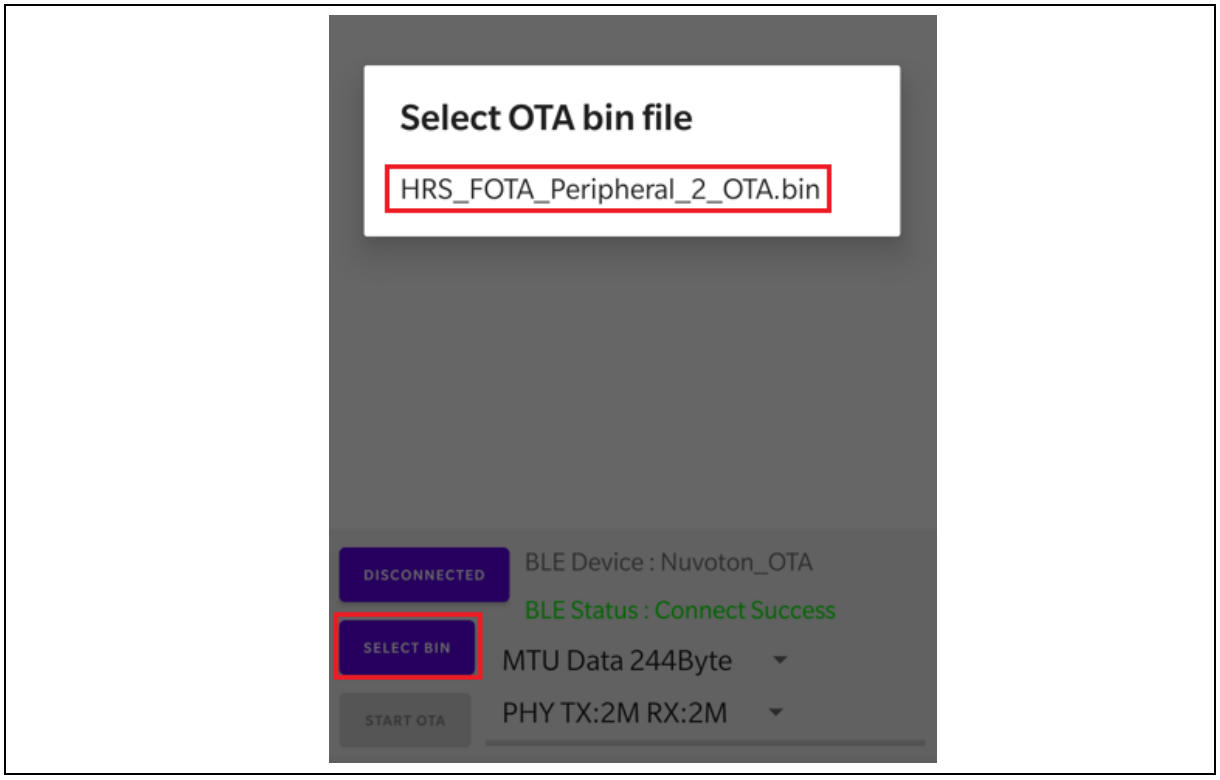

Figure 3-10 Select OTA Image in App

After loading the image file, you can click the "**START OTA**" button to start the FOTA update. Then the running message will be displayed, shown as Figure 3-11.

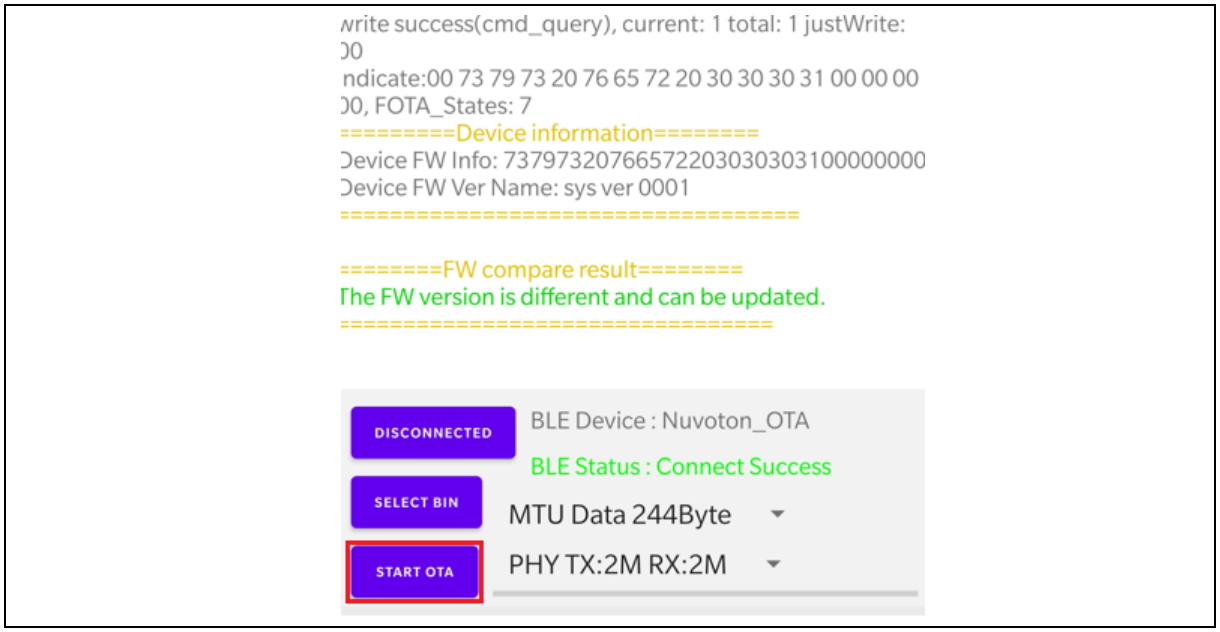

Figure 3-11 Doing FOTA Update

After the FOTA update is done, the BLE device will reboot automatically. After the device reboots, you can find the new FW has the different system information, shown as Figure 3-12.

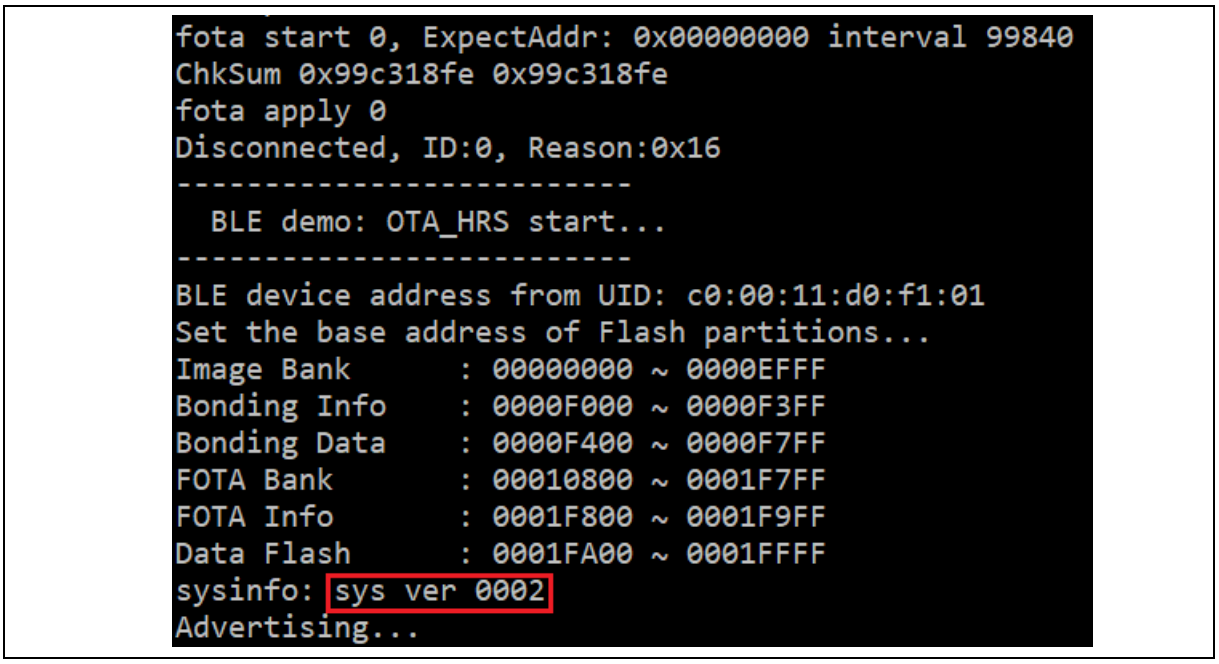

Figure 3-12 Booting Message after FOTA

4 FOTA Communication Flow

After the BLE device is connected with a smart phone, the compete FOTA update handshake flow proceeds as shown in Figure 4-1.

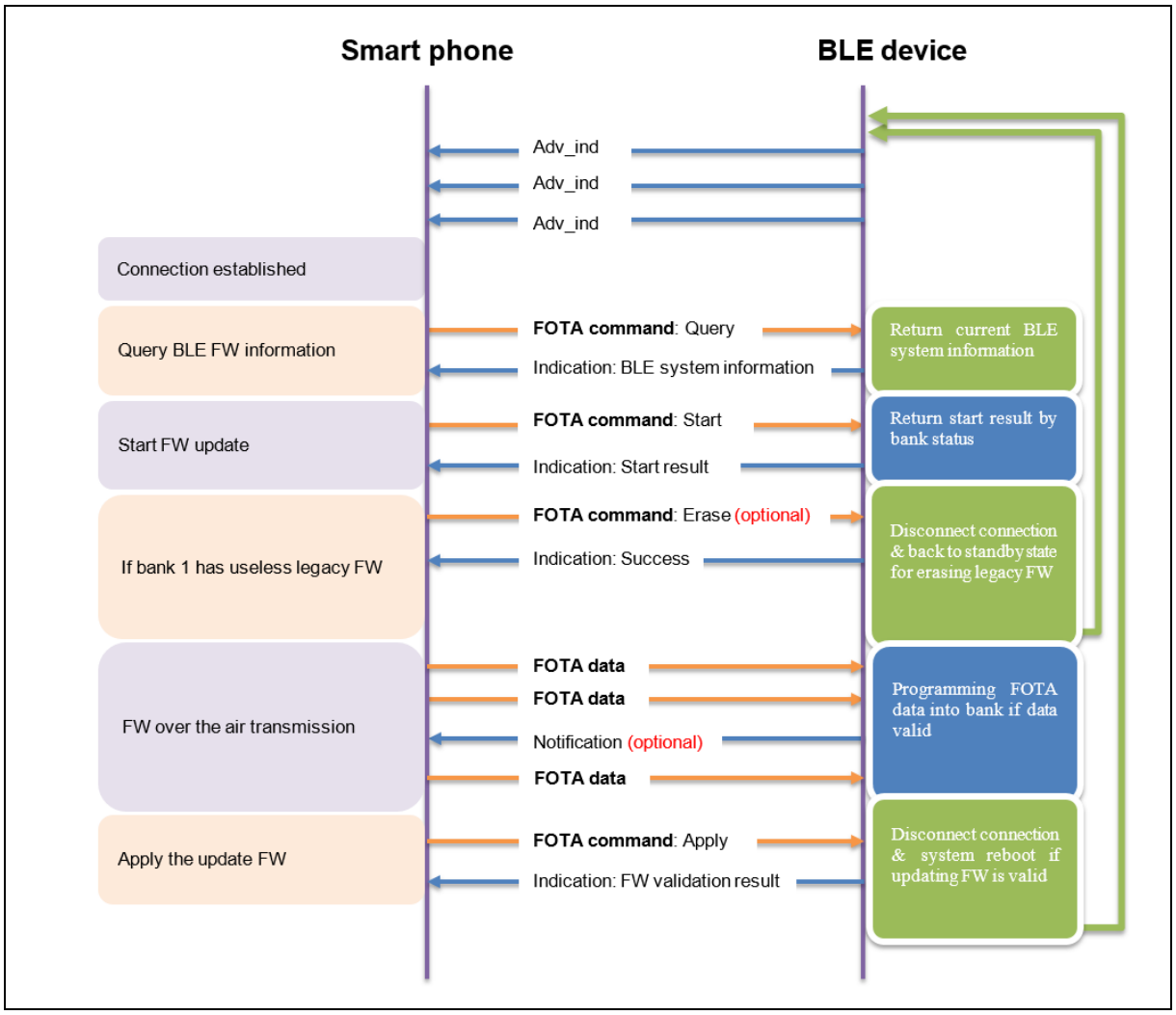

Figure 4-1 FOTA Communication Flow

From the handshake flow, you can find that there are two major functions **FOTA command** and **FOTA data** to handle FOTA update procedure. The two functions are defined in the *fota.c* file.

4.1 FOTA Command

The FOTA commands are parsed by function “BleFota_Cmd()”.

```
void BleFota_Cmd(uint8_t length, uint8_t *data);
```

The first byte of command payload contains command ID and each command ID may contain different information behind. We defined four kind of commands here:

```
typedef uint8_t OtaCmd;
#define OTA_CMD_QUERY           0x00  /* Query current system information */
#define OTA_CMD_START           0x01  /* Start FW OTA update */
#define OTA_CMD_ERASE           0x02  /* Erase legacy FW */
#define OTA_CMD_APPLY           0x03  /* FW transmission completed, apply new FW */
```

1. Query: Query device current system information.

octets	1
parameter	Command ID (0x00)

2. Start: Start FW update, this command contains new FW length and CRC.

octets	1	4	4	4
parameter	Command ID (0x01)	FW length	FW CRC	Notify interval

3. Erase: Terminate the connection and erase legacy bank 1 FW and information.

octets	1
parameter	Command ID (0x02)

4. Apply: Apply the new FW if receiving FW length and CRC matched with FOTA command “Start”.

octets	1	4
parameter	Command ID (0x03)	Bank0 address

When the device receives FOTA command, an indication with error code will return to smart phone for deciding how to continue the FOTA procedure. These error codes have been defined in the “*fota.c*” file:

```
/*Indication error codes for FOTA command*/
typedef uint8_t OtaErrCode;
#define OTA_ERR_CODE_NO_ERR                     0x00
```

```
#define OTA_ERR_CODE_CMD_ERR                0x01
#define OTA_ERR_CODE_ALREADY_START          0x02
#define OTA_ERR_CODE_UPDATE_NOT_START       0x03
#define OTA_ERR_CODE_FLASH_ERASE_ERR        0x04
#define OTA_ERR_CODE_FW_CRC_ERR             0x05
#define OTA_ERR_CODE_FW_LEN_ERR             0x06
#define OTA_ERR_CODE_OUT_OF_BANK_SIZE       0x07
```

5. No error: Command success.

octets	1
parameter	Error code (0x00)

6. Command error: Unsupported command ID.

octets	1
parameter	Error code (0x01)

7. Already start: FOTA procedure already start.

octets	1	4	4	3
parameter	Error code (0x02)	Updating FW length	Updating FW CRC	Updating FW next expect address

8. Update not start: FOTA procedure was not start.

octets	1
parameter	Error code (0x03)

9. Flash erase error: Bank 1 flash erase fail.

octets	1
parameter	Error code (0x04)

10. FW CRC error: Receiving FW's CRC incorrect.

octets	1
parameter	Error code (0x05)

11. FW length error: Receiving FW's length incorrect.

octets	1
parameter	Error code (0x06)

12. Out of bank size: Updating FW's length larger than bank size.

octets	1	3
parameter	Error code (0x07)	Current FW bank size

4.2 FOTA Data

The function “BleFota_Data()” is implemented to handle FOTA data.

```
void BleFota_Data(uint8_t length, uint8_t *data);
```

The first 4 bytes of data payload is data header that contains the FOTA data programming address (3 bytes) and length (1byte).

octets	3	1	Variable
parameter	Data programming address	Data programming length	Data

If there is invalid data header, device will send notification to response it. Hence, for ideal FOTA data transmission, there is no notification sent from device unless smart phone had configured periodic notification. The notification reason is contained in the first byte of notification packet and you can find the definition in the “*fota.c*” file as follows:

```
/*Notification reasons for FOTA data*/
typedef uint8_t OtaNotify;
#define OTA_DATA_NOTIFY_PERIODIC                0x00
#define OTA_DATA_NOTIFY_TIMEOUT                 0x01
#define OTA_DATA_NOTIFY_ADDRESS_UNEXPECTED      0x02
#define OTA_DATA_NOTIFY_LEN_ERROR               0x03
```

```
#define OTA_DATA_NOTIFY_TOTAL_LEN_ERR          0x04
#define OTA_DATA_NOTIFY_ADDRESS_ERR            0x05
#define OTA_DATA_NOTIFY_NOT_START              0x06
#define OTA_DATA_NOTIFY_NONE                   0xFF
```

1. Periodic: Periodic notification, the notify interval set from FOTA start command.

octets	1	3
parameter	Notify reason (0x00)	Updating FW next expect address

2. Timeout: FOTA data was not received in a specific interval.

octets	1	3
parameter	Notify reason (0x01)	Updating FW next expect address

3. Address unexpected: Received FOTA data's address was not continuously.

octets	1	3
parameter	Notify reason (0x02)	Updating FW next expect address

4. Length error: Received FOTA data length incorrect.

octets	1	1
parameter	Notify reason (0x03)	Received FOTA data length

5. Total length error: Total received FOTA data length is larger than bank size.

octets	1	3
parameter	Notify reason (0x04)	Total Receiving FOTA data length

6. Address error: Received FOTA data's address is larger than bank size.

octets	1
parameter	Notify reason (0x05)

7. Not start: FOTA data received but FOTA procedure was not start.

octets	1
parameter	Notify reason (0x06)

5 Porting FOTA Related Functions

This section describes additional functions needed for a complete FOTA update procedure.

5.1 Bootloader

The main purpose of bootloader is to activate new FW if a valid image is present in Bank 1. The following steps must be performed by the bootloader:

1. Check whether the new FW exists and go to Step 2 if new FW is found in Bank 1; otherwise, go to Step 4.
2. Utilize the CRC32 checksum to determine whether the new FW is valid or not, then go to Step 3 if valid; otherwise, go to Step 4.
3. Copy the new FW to Bank 0.
4. Boot from Bank 0.

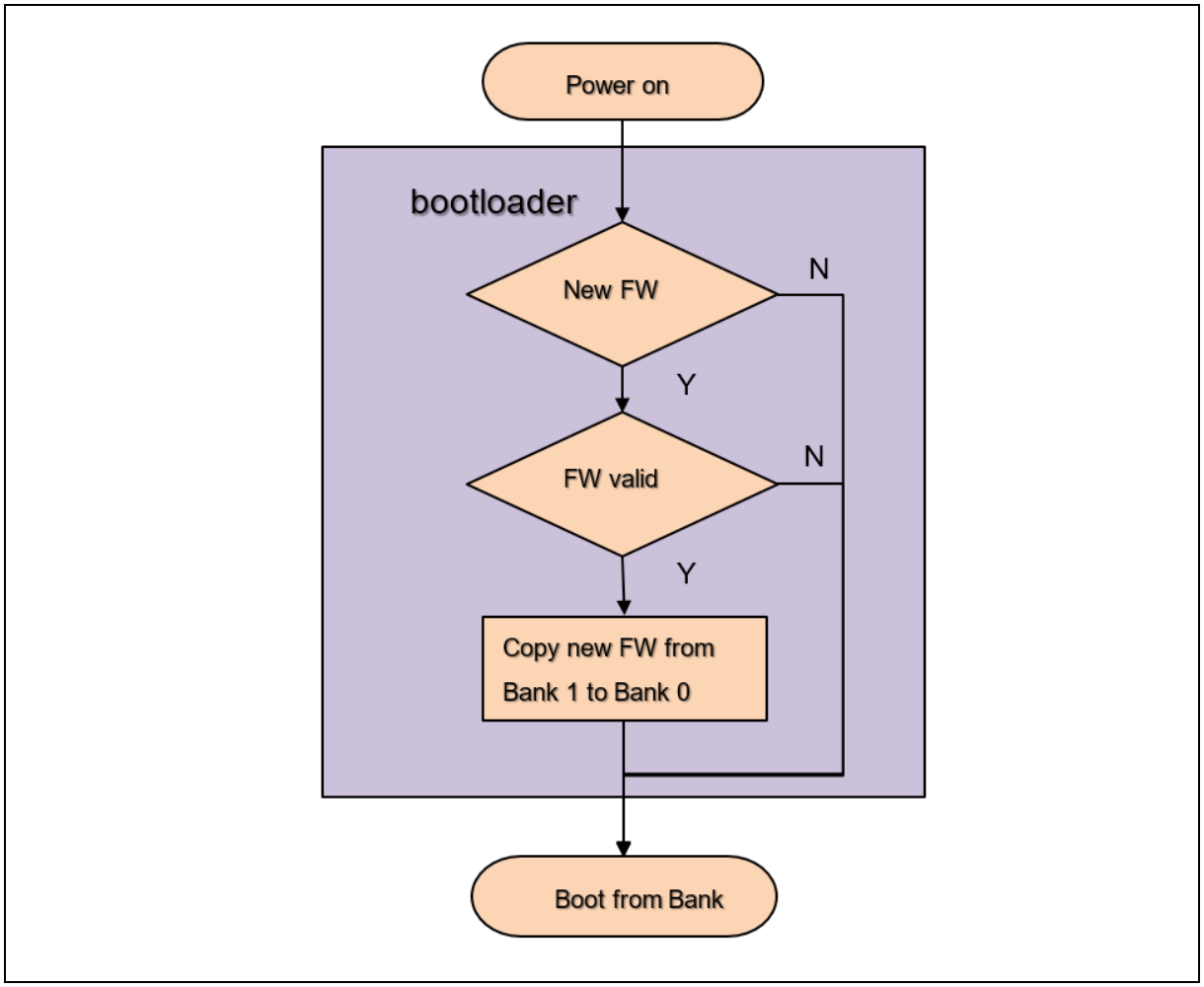

Figure 5-1 Bootloader Flow

5.2 FOTA Firmware Merge

The complete functions of the FOTA are implemented in the *fota.c* file. To merge it into your firmware, you just need to add some codes in your project. For details, you can compare the sources between HRS and OTA_HRS projects.

5.2.1 BleFota_TimerTick()

A timer with 1 second interval is used to call BleFota_TimerTick() per-tick to implement a failsafe mechanism. When the timer expires in BleFota_TimerTick(), a notification is sent to the App indicating packets missing from the transfer.

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

    BleFota_TimerTick();
}
```

5.2.2 BleFota_Disconnect()

In BLECMD_EVENT_DISCONN_COMPLETE event handle in the *user.c* file, please call the BleFota_Disconnect() to process the FOTA related actions after disconnection.

```
case BLECMD_EVENT_DISCONN_COMPLETE:
{
    BLE_Event_DisconnParam *disconnParam = (BLE_Event_DisconnParam *)param;
    ble_state = STATE_BLE_STANDBY;

    BleFota_Disconnect();
}
```

5.2.3 Enabling Data Length Extension Feature

To speed up the OTA image transfer performance via BLE, you can enable the data length extension feature with maximum MTU 247 bytes.

Revision History

Date	Revision	Description
2021.07.20	1.00	1. Initially issued.
2021.08.31	2.00	1. Supported SDK v2.
2021.12.23	2.01	1. 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.