RTL8762E OTA User Guide

V1.0

2022/06/02

Revision History

Date	Version	Comments	Author	Reviewer
2021/09/02	V0.1	Draft version	Grace	
2022/06/01	V1.0	First Release version	Arthur	Grace

Contents

Table List

Figure List

1 Overview

1.1 Function Description

OTA (Over The Air) represents the technology that apply Bluetooth to update image (code and data) that runs in RTL8762E Flash.

1.2 Related Emphasis

1. Flash Layout
2. IMG format
3. OTA packaging format and tool usage
4. OTA Protocol

2 Flash layout

Flash layout of RTL8762E consists of OEM Config, OTA Bank0, OTA Bank1, FLASH Transport Layer(FTL), OTA TMP and APP defined section, as shown in Figure 2-1. Start address for accessing Flash is 0x800000. The above is a brief description. For details, please refer to Chapter 5 of the document <RTL8762E Memory User Guide>.

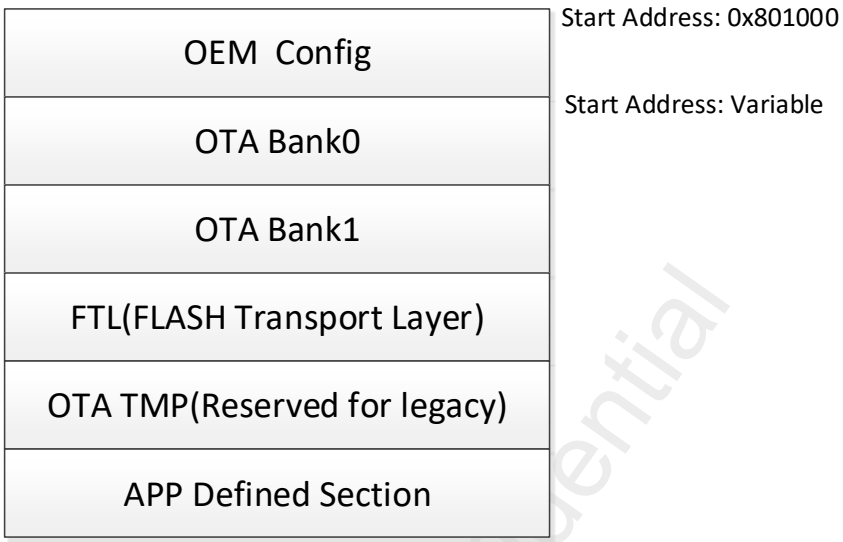

Figure 2-1 Flash Layout

Memory and corresponding function of Flash is shown in Table 2-1.

Table 2-1 FLASH Memory and Function Description

Memory Segment	Starting Address	Size (Bytes)	Functions
OEM Config	0x801000	0x1000	Storage of Config information, including Bluetooth address, AES Key and Customizable Flash Layout.
OTA Bank0	Variable (defined in OEM Config)	Variable length (defined in OEM Config)	If not in bank switching mode, this region contains the project data and codes to be executed, including OTA Header, Secure boot, Patch, Upperstack, APP, APP Data1, APP Data2, APP Data3, APP Data4, APP Data5, APP Data6. OTA_TMP is the backup region of this OTA. In bank switching mode, OTA Bank 0 and OTA Bank 1 is backup region of each other.
OTA Bank1	Variable	Variable length	This region only exists when bank switching method is applied. It has same functions and same size with Bank 0 in bank switching mode
FTL	Variable	Variable length	A software technology that access Flash with logical address. Customer no longer needs to focus on operations on Flash physical layer. This region also balances consumption.
OTA_TMP	Variable	Variable length	Used as backup region of OTA if not in bank switching mode. Its size should be no less than largest image in OTA Bank 0.
APP Defined Section	Variable	Variable length	The rest of Flash that can be customized. This region cannot be managed by OTA scheme.

OTA bank layout is shown in Figure 2-2, and description for each part is shown in Table 2-2.

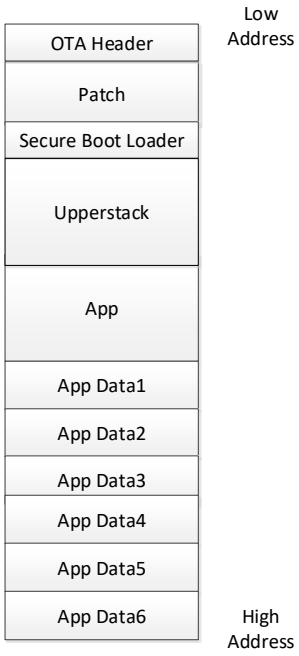

Figure 2-2 OTA Bank Layout

Table 2-2 Flash Segmentation

Memory Segment	Starting Address	Size	Functions
OTA Header	Determined in the OEM Config region	4KB	This region contains the OTA Header version and start address and size of the images in the bank
Secure Boot Loader	Determined in the OTA Header region	Variable	This region contains secure boot loader.
Patch	Determined in the OTA Header region	Variable	This region contains the code that optimize and extend the protocol stack and system in ROM.
Upperstack	Determined in the OTA Header region	Variable	This region contains the upperstack code.
App	Determined in the OTA Header region	Variable	This region contains project code.
App Data1	Determined in the OTA Header region	Variable	Data region used in project.
App Data2	Determined in the OTA Header region	Variable	Data region used in project.
App Data3	Determined in the OTA Header region	Variable	Data region used in project.
App Data4	Determined in the OTA Header region	Variable	Data region used in project.
App Data5	Determined in the OTA Header region	Variable	Data region used in project.
App Data6	Determined in the OTA Header region	Variable	Data region used in project.

3 Image Format

All images that may need to be upgraded (OTA Header, Secure boot, Patch, Upperstack, APP, APP Data1, APP Data2, APP Data3, APP Data4, APP Data5, APP Data6) are composed of a 1KB header and payload. Among them, the header part in OTA Header image and other images are slightly different. In this article, the headers of all upgradeable images except the OTA Header image are called Image Headers.

3.1 OTA Header Image Format

OTA Header image is made up of header (1KB) and dummy payload (3KB). OTA Header is generated by MPTool. Different fields of header are shown in Figure 3-1.

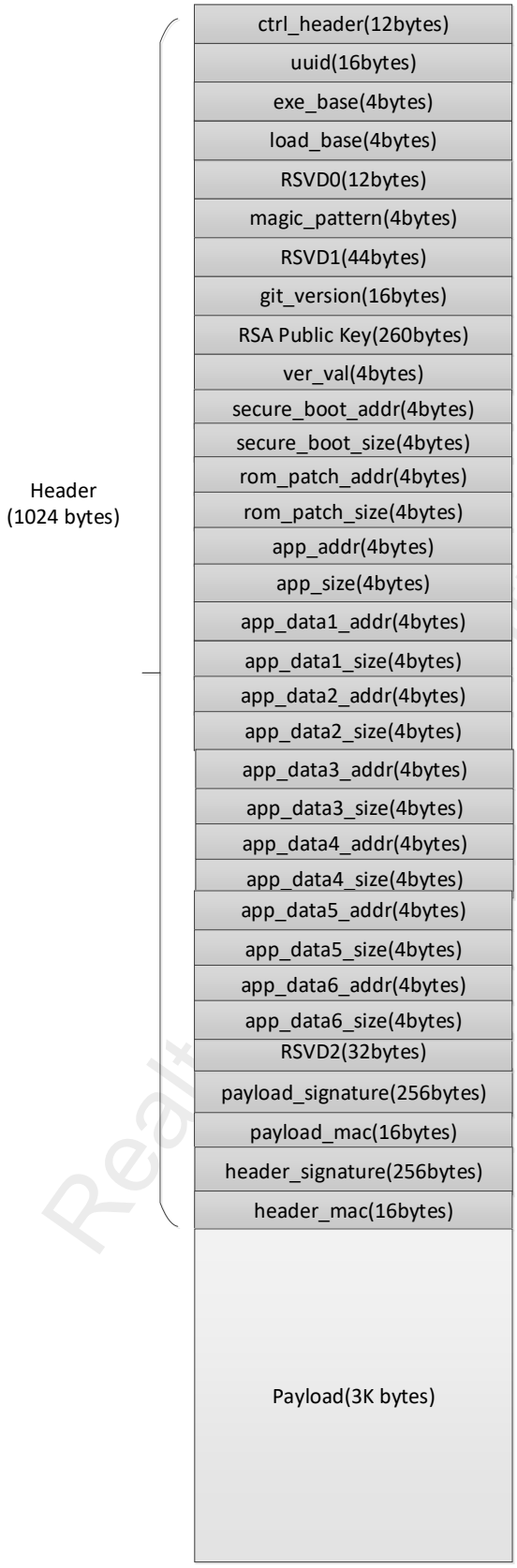

Figure 3-1 OTA Header Format

Header fields and corresponding functions are shown in Table 3-1.

Table 3-1 Fields of OTA Header

Fields	Length (Byte)	Functions
ctrl_header	12	Control message of OTA Header
secure_boot_addr	4	Start address of secure boot image
secure_boot_size	4	Size of secure boot image
rom_patch_addr	4	Start address of ROM patch image
rom_patch_size	4	Size of ROM patch image
app_addr	4	Start address of application image
app_size	4	Size of application image
app_data1_addr	4	Start address of application data1
app_data1_size	4	Size of application data1
app_data2_addr	4	Start address of application data2
app_data2_size	4	Size of application data2
app_data3_addr	4	Start address of application data3
app_data3_size	4	Size of application data3
app_data4_addr	4	Start address of application data4
app_data4_size	4	Size of application data4
app_data5_addr	4	Start address of application data5
app_data5_size	4	Size of application data5
app_data6_addr	4	Start address of application data6
app_data6_size	4	Size of application data6

3.2 Other Image Format

Image of patch, APP and App data is made up of image header (1KB) and corresponding payload. Image header of patch and APP is generated while compiling and linking, and that of App data is added by APP DATA Tool. Header fields are shown in Figure 3-2, and corresponding functions are shown in Table 3-2.

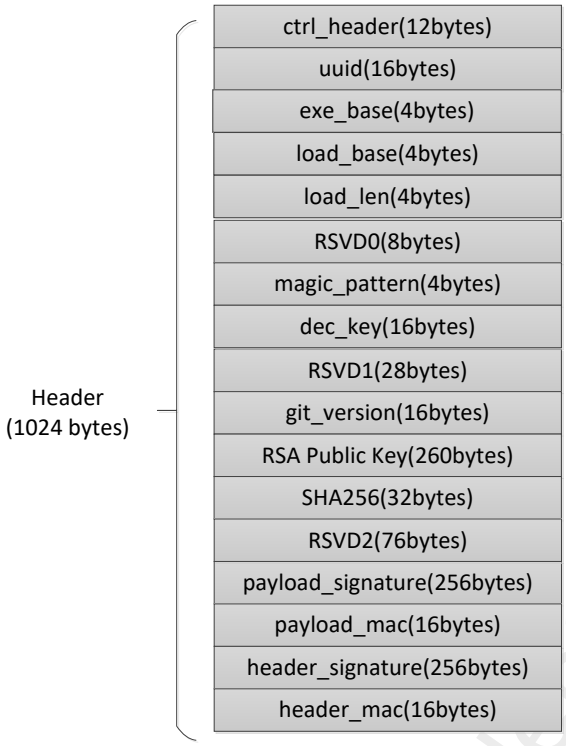

Figure 3-2 Image Header Layout

Table 3-2 Image Header Field

Fields	Length(Byte)	Functions
ctrl_header	12	Control information field of Image Header
git_version	16	Information field of version control

ctrl_header format in Image Header is shown as follows:

```
typedef struct _IMG_CTRL_HEADER_FORMAT
{
    uint8_t ic_type;
    uint8_t secure_version;
    union
    {
        uint16_t value;
        struct
        {
            uint16_t xip: 1; // payload is executed on flash
            uint16_t enc: 1; // all the payload is encrypted
            uint16_t load_when_boot: 1; // load image when boot
            uint16_t enc_load: 1; // encrypt load part or not
            uint16_t enc_key_select: 3; // referenced to ENC_KEY_SELECT
```

```
        uint16_t not_ready : 1; //for copy image in ota
        uint16_t not_obsolete : 1; //for copy image in ota
        uint16_t integrity_check_en_in_boot : 1; // enable image integrity check in boot flow
        uint16_t rsvd: 6;
    };
  } ctrl_flag;
  uint16_t image_id;
  uint16_t crc16;
  uint32_t payload_len;
} T_IMG_CTRL_HEADER_FORMAT;
```

ic_type represents IC type, which has the value of 12 when RTL8762E chip is used. secure_version indicates version of secure boot image.

image_id identifies different types of image, among which SCCD, OCCD and FactoryCode cannot be updated through OTA. The types are enumerated in T_IMG_ID.

```
typedef enum
{
  SCCD       = 0x278D,
  OCCD       = 0x278E,
  FactoryCode  = 0x278F,
  OTA        = 0x2790, /**< OTA header */
  SecureBoot  = 0x2791,
  RomPatch    = 0x2792,
  AppPatch    = 0x2793,
  AppData1    = 0x2794,
  AppData2    = 0x2795,
  AppData3    = 0x2796,
  AppData4    = 0x2797,
  AppData5    = 0x2798,
  AppData6    = 0x2799,
  UpperStack   = 0x279a,
  IMAGE_MAX   = 0x279b,
  IMAGE_USER_DATA = 0xFFFE,  /**<the image only support unsafe single bank ota*/
} T_IMG_ID;
```

Related instructions:

1. The unit of payload_len is bytes, indicating the size of the image, excluding the 1KB image header;
2. crc16 indicates whether to perform crc check or SHA256 check, if it is not 0, it indicates crc check, otherwise SHA256 check is performed;
3. The only bit fields related to ctrl_flag and OTA are not_ready and not_obsolete:
   1) not ready indicates whether the OTA transfer writing is completed, not_ready in the default compiled image

is 0;

(1) When the image is written to the backup area, first set not_ready to 1, until the upgrade transmission is completed, and after passing the integrity check (CRC or SHA256), not_ready is written to 0, indicating that the image is ready;

2) not_obsolete indicates whether the image is discarded, and not_obsolete is 1 in the default compiled image;

(1) When bank switching is supported, not_obsolete is invalid during the OTA process;

(2) When bank switching is not supported, after the OTA upgrade is completed, the startup code determines that not_ready is 0 and not_obsolete is 1. At this time, the image is moved from the OTA_TMP area to the designated area (App/Patch/Appdata). The not_obsolete in the image in the OTA_TMP area is written as 0;

4. The business code of Secure Boot/Patch/APP and the raw data part of APP Data are placed in the Image Payload.

4 OTA packaging format and tool usage

OTA has the following two packaging tools:

1. Flash Map Generate Tool: Generate flash map.h and flash_map.ini.
   1) flash_map.h needs to be placed in the upper-level directory of the project to participate in the compilation and generate the APP image;
   2) flash map.ini is the input file of MP Pack Tool and MP Tool, and it is necessary to ensure that the image is consistent with all the output addresses set;
2. MP Pack Tool: Pack OTA files.

4.1 Support bank switching

4.1.1 Flash Layout

Bank switching method needs two OTA Banks that are completely same to become the backup of each other. Its advantage is that program can directly jump to new bank when reboot. However, OTA update in bank switching mode takes more flash memory to speed up update, so the size of flash memory should be larger if bank switching method is applied.

If flash size is comparatively large, user can update firmware by applying bank switching method. Take 1 MB Flash as example, the suggested Flash layout is shown below:

Table 4-1 FLASH Layout Sample (support bank switch)

Sample layout for flash(total size = 1MB)	Size(bytes)	start address
1) Reserved	4K	0x00800000
2) OEM Header	4K	0x00801000
3) OTA Bank0	424K	0x00802000
a) OTA Header	4K	0x00802000
b) Secure boot loader	4K	0x0080D000
c) Patch code	40K	0x00803000
d) Upperstack code	144K	0x0080E000
e) APP code	232K	0x00832000
f) APP data1	0K	0x0086C000
g) APP data2	0K	0x0086C000
h) APP data3	0K	0x0086C000
i) APP data4	0K	0x0086C000
j) APP data5	0K	0x0086C000
k) APP data6	0K	0x0086C000
4) OTA Bank1 (size must be same with OTA Bank0)	424K	0x0086C000
a) OTA Header	4K	0x0086C000
b) Secure boot loader	4K	0x00877000
c) Patch code	40K	0x0086D000
d) Upperstack code	144K	0x00878000
e) APP code	232K	0x0089C000
f) APP data1	0K	0x008D6000
g) APP data2	0K	0x008D6000
h) APP data3	0K	0x008D6000
i) APP data4	0K	0x008D6000
j) APP data5	0K	0x008D6000
k) APP data6	0K	0x008D6000
5) FTL	16K	0x008D6000
6) OTA Temp	0K	0x008DA000
7) APP Defined Section	152K	0x008DA000

Note: Flash Layout should be determined based on actual size of image and data.

4.1.2 Usage of package tool with bank switching

1. Use FlashMapGenerateTool to flash map.ini, flash map.h, OTA Header0 and OTA Header1.

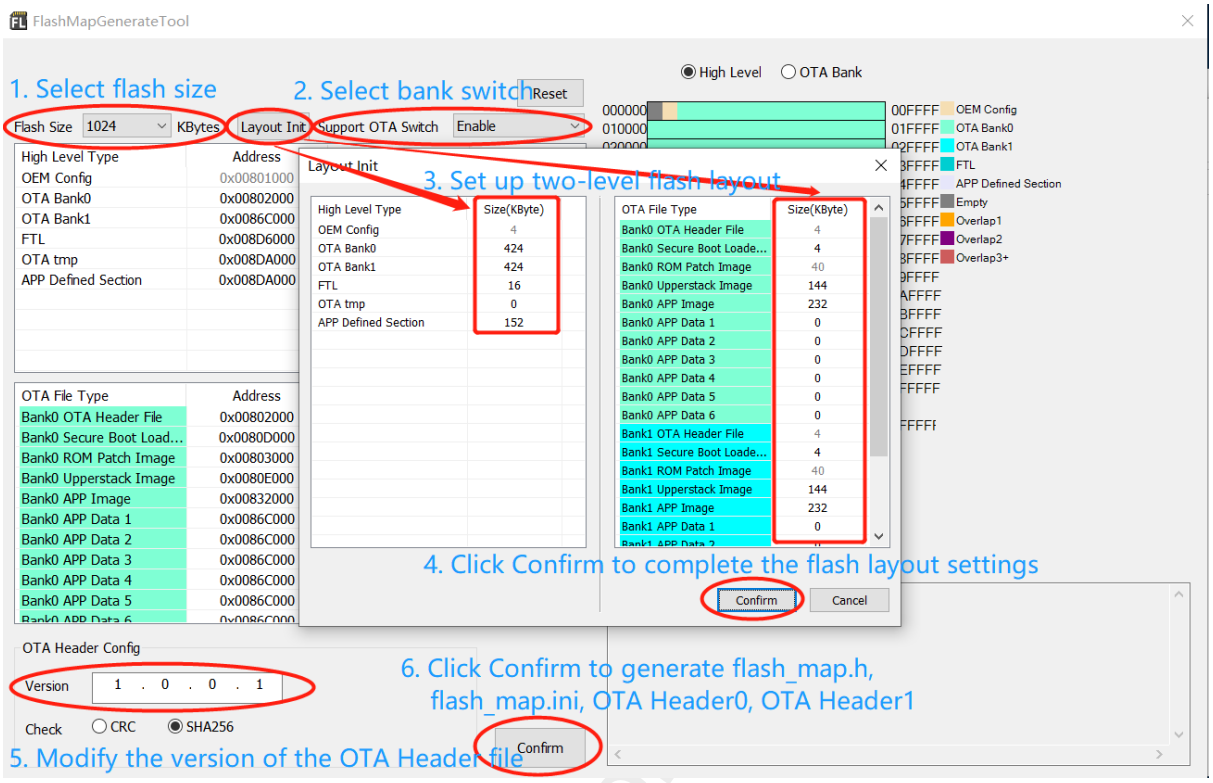

Figure 4-1 flash layout config and generate ota header image

Note: The version number of the OTA Header used for packaging is higher than the version number of the original running version, so that the new bank can take effect normally after the OTA upgrade.

2. Copy 'flash map.h' to the upper-level directory with project and open project with Keil. Link and compile the project to generate "app_MP_sdk#####+version+MD5.bin" file for packaging. Bank switching method needs to compile the app images of OTA BANK0 and OTA BANK1.

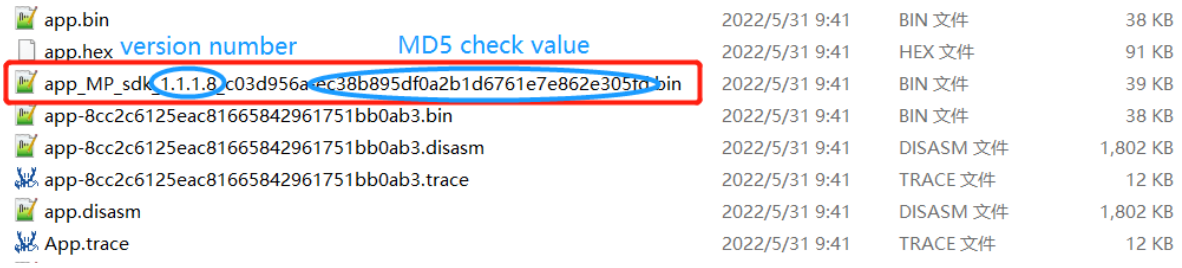

Figure 4-2 builded APP image

Note:The demo app project in the Realtek released SDK can only compile the app image of OTA BANK0 by default. Compiling app image of OTA BANK1 need modify the macro 'APP_BANK' in mem_config.h in the same directory as the project file to 1.

```
1.  /** @brief set app bank to support OTA: 1 is ota bank1, 0 is ota bank0 */
2.  #define APP_BANK                    1
```

3. Get patch and Upperstack image that runs runs in OTA Bank1.

   Note: The default released patch and Upperstack image can only run in OTA Bank0. Please consult Realtek to get patch and Upperstack image that runs in OTA Bank1, when you choose bank swithing method.

4. Generate packet file of ImgPacketFile-xxxxxx.bin in current directory, which is used for updating.

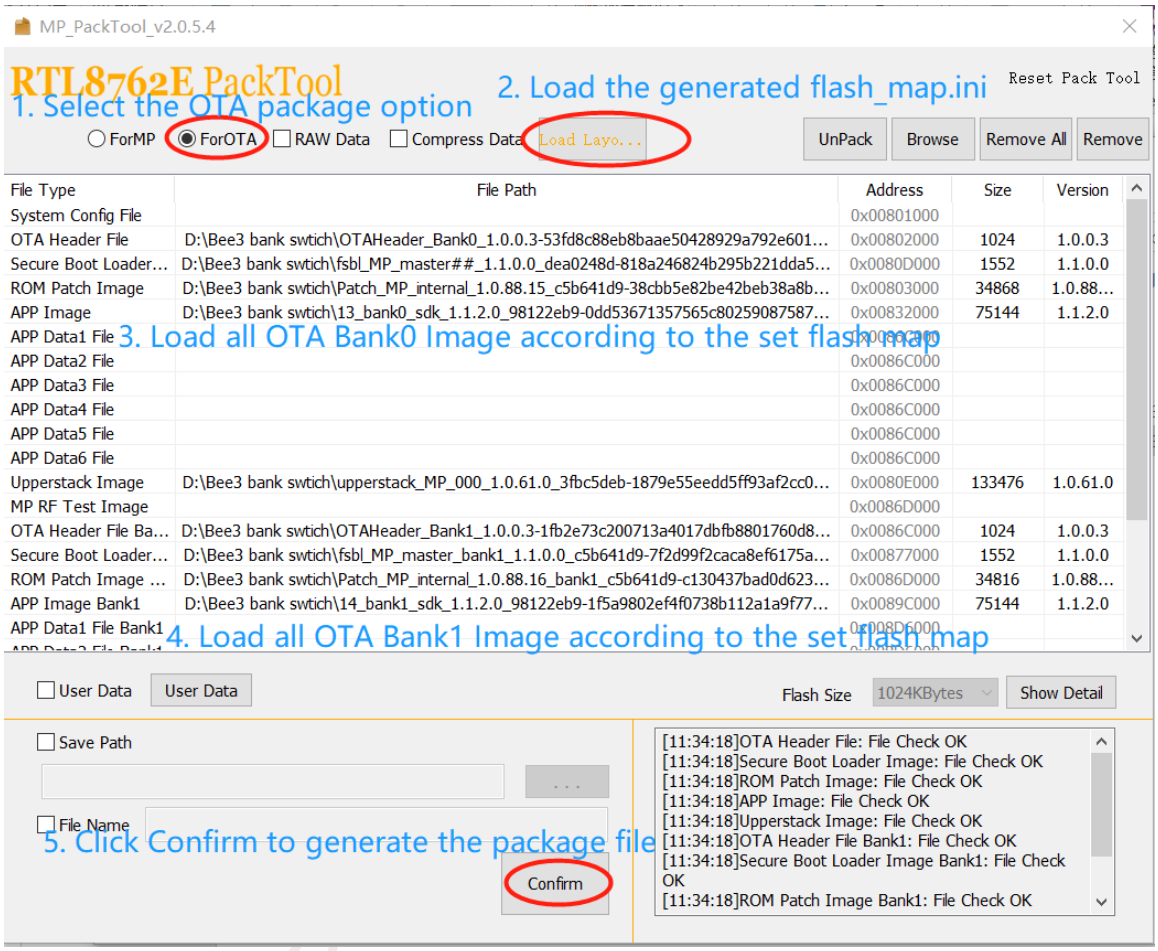

Figure 4-3 Package to generate PACK

1) Note:

(1) Both OTA Head0 and OTA Header1 need to be packaged to PACK, different from the mode without bank switching.

(2) All the contents defined in Flash layout need to be packaged, especially Header, Patch and APP of OTA.

(3) It is recommended that package both bank0 and bank1 in PACK.

*(4) APP DATA file is generated with APP DATA generation script, for detailed information refer to <**RTL8762E App DataTools User Guide**>.*

4.2 Do not support bank switching

4.2.1 Flash Layout

The following three points should be paid attention to in the Flash layout that does not switch the Bank scheme:

1. There is no need to allocate capacity in OTA Bank1 area;
2. The OTA Temp area needs to allocate capacity, and it should not be smaller than the image size with the largest capacity in OTA Bank0.

   Therefore, the advantage of this scheme is that it relatively saves the Flash space. After the OTA transfer is completed, restarting the boot program will move the data in the OTA Temp area to the image area designated by OTA Bank0, and then restart to take effect, so the restart time for the OTA upgrade is relatively increased.
3. Combined image upgrade is also supported in the non-switching bank scheme. This function is controlled by the macro definition SUPPORT_TEMP_COMBINED_OTA in board.h, which is enabled by default. When using the combined image upgrade, when writing the image data to the OTA temp area, it will calculate whether the remaining space in the OTA temp area can put down the next image file that needs to be written, and if it can be put down, continue to write the next image file. OTA temp area, when it cannot fit, move the data in the OTA Temp area to the OTA Bank0 area. The advantage of the combined upgrade is that it reduces the number of restarts and speeds up the transfer rate.

The following takes the 512K byte flash as an example to introduce the non-switching Bank scheme. The recommended Flash layout is shown in the following table:

Table 4-2 FLASH Layout Sample(don't support bank switch)

Sample flash layout(total size is 512KB)	Size(bytes)	Start address
1) Reserved	4K	0x00800000
2) OEM Header	4K	0x00801000
3) OTA Bank0	340K	0x00802000
a) OTA Header	4K	0x00802000
b) Secure boot loader	4K	0x0080D000
c) Patch code	40K	0x00803000
d) Upperstack code	144K	0x0080E000
e) APP code	148K	0x00832000
f) APP data1	0K	0x00857000
g) APP data2	0K	0x00857000
h) APP data3	0K	0x00857000
i) APP data4	0K	0x00857000
j) APP data5	0K	0x00857000
k) APP data6	0K	0x00857000
4) OTA Bank1	0K	0x00857000
5) FTL	16K	0x00857000
6) OTA Temp	156K	0x0085B000
7) APP Defined Section	0K	0x00880000

Note: The space for APP data is not allocated in this sample; FLASH Layout should be distributed based on actual size of image and data.

4.2.2 Usage of package tool without bank switching

1. Use Flash Map Generate Tool to generate flash map.ini, flash map.h and OTA Header0 files, as shown in the following figure:

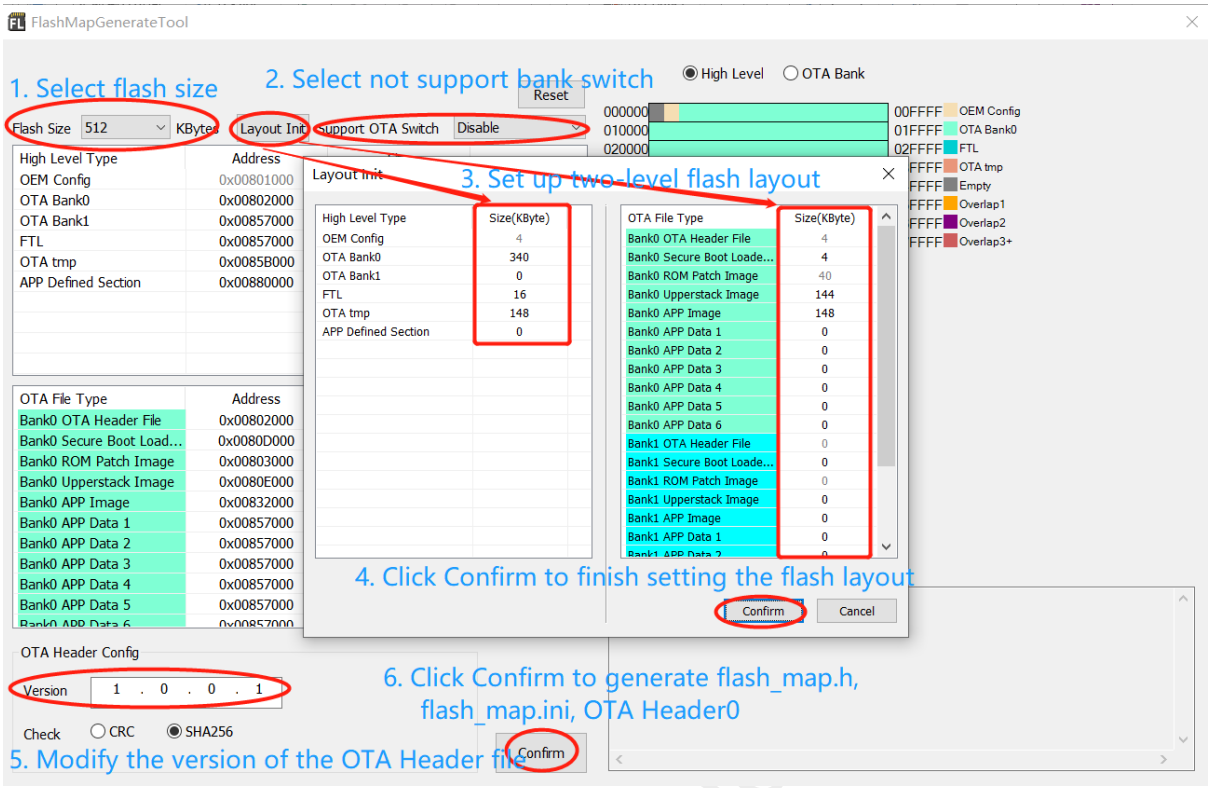

Figure 4-4 flash layout config and generate ota header image

Note: The flash map.ini generated here needs to be consistent with the flash map.ini used in the MP phase.

2. Copy ‘flash map.h’ to the upper-level directory with project and open project with Keil. Link and compile the project to generate “app_MP_sdk#####+version+MD5.bin” file for packaging. To apply Without Bank switching method, “mem_config.h” in the upper-level directory with project should be modified.

```
1.  /** @brief set app bank to support OTA: 1 is ota bank1, 0 is ota bank0 */
2.  #define APP_BANK        0
```

3. Open MPPackTool to load flash_map.ini generated in previous step and load corresponding image.

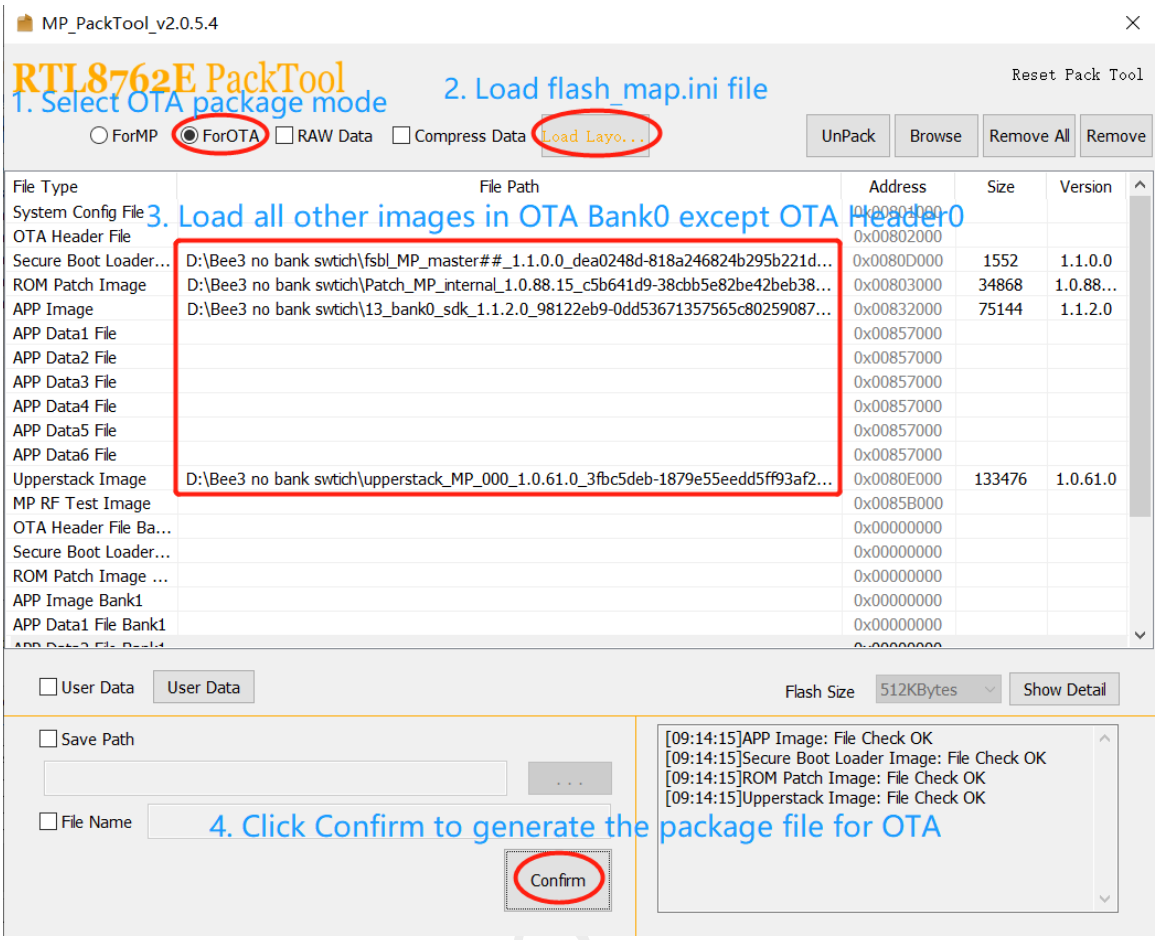

Figure 4-5 pack OTA images using MP PACK Tool

Note:

1) *OTA Head0 doesn't need to be packaged to PACK, different from the mode with bank switching.*
2) *Content of Secure boot loader Image is defined in Flash Layout, but it's not recommended to package if there isn't any new version of Secure boot loader Image.*
3) *If only ROM Patch Image or APP Image, either of them can be packaged.*

5 OTA Protocol

5.1 DFU Service

DFU Service uuid: { 0x12, 0xA2, 0x4D, 0x2E, 0xFE, 0x14, 0x48, 0x8e, 0x93, 0xD2, 0x17, 0x3C, ***0x87, 0x62***, *0x00, 0x00*}.

DFU Service defines two Characteristics：

1. Data Characteristic accepts image data (write no response);
2. Control Point Characteristic accepts control commands (write/notification);

Control points supported by DFU Service:

Table 5-1 DFU opcode

Procedure	Requirement	Properties	Parameter Description	Applicable Response Value(s)	Response Parameter
Start DFU ①	M	Write	ic_type(UINT8) secure_version (UINT8) ctrl_flag.value(UINT16) image_id (UINT16) crc16((UINT16) payload_len (UINT32)	ARV	None
Receive FW image	M	Write	image_id (2byte-UINT16) nImageLength(4Byte-INT32)	ARV	None
Validate FW	M	Write	image_id (2byte-UINT16)	ARV	None
Activate Image and Reset	M	Write	None	None	None
Reset System	M	Write	None	None	None
Report Received Image Information	M	Write	image_id(UINT16)	ARV	origin_image_version (UINT32) cur_offset (UINT32)
Connection parameter update	M	Write	connIntervalMin(UINT16) connIntervalMax(UINT16) connLatency(UINT16) supervisionTimeout(UINT16)	ARV	None
Buffer check enable	M	Write	None	ARV	Max buffer size(UINT16) Mtu size(UINT16)
Buffer check size&crc	M	Write	mBufferSize(UINT16) mCrc(UINT16)	ARV	Next send offset(UINT32)
IC type	O	Write	None	ARV	ic_type(UINT8)
Copy Img②	M	Write	image_id(UINT16) destination_addr(UINT32) copysize(UINT32)	ARV	None

Precautions:

1. The parameter of Start DFU is the ctrlheader of img. After receiving the ctrlheader, it needs to be written into the flash as part of the upgrade file, so these 12 bytes are processed the same as the data transmission. If aes encryption is enabled, after receiving the Start DFU After the parameter, it needs to be decrypted and then parsed, and written into the flash;
2. When the app data is upgraded, the Bank switching scheme is adopted, and it is judged that the secure version is the same as the APP DATA version. This command can be used to directly copy the contents of the original Bank to the destination Bank, eliminating the OTA transmission link;
3. During data transmission, if buffer check is enabled, its size must be (16*2^n)bytes and less than or equal to max buffer Size (returned by the Buffer check enable command). If aes encryption is supported, every 16 bytes is encrypted. After the receiving end receives the data, it needs to be decrypted first. The last part less than 16 bytes is sent without encryption. When the buffer check size is full, write to flash;
4. If buffer check is not enabled, you need to send 20*n(n=1,2,4,5,10) bytes at a time, until the receiving end collects 2000bytes, can write to flash, and write the file after receiving the last time . If aes encryption is supported, the part of the packet data that is divisible by 16 needs to be encrypted and sent, and the remainder is less than 16 bytes and does not need to be encrypted.

5.2 OTA Service

OTA Service uuid: { 0x12, 0xA2, 0x4D, 0x2E, 0xFE, 0x14, 0x48, 0x8e, 0x93, 0xD2, 0x17, 0x3C, ***0xFF, 0xD0***, *0x00, 0x00*}.

OTA Service defines the following Characteristics:

Table 5-2 OTA Characteristic

Characteristic Name	Requirement	Mandatory Properties	Description
OTA CMD	M/O	WriteWithoutResponse	Refer to OTA CMD
Device Mac	M	Read	Refer to Device Mac
Patch Version	M	Read	Refer to Patch Version
App Version	M	Read	Refer to App Version
Patch Extension Version	O	Read	Refer to Patch Extension Version
Test Mode	O	WriteWithoutResponse	Refer to Test Mode
Device Info	M	Read	Refer to Device Info
Image Counter	O	WriteResponse	Refer to Image Counter
Image Version	M	Read	Refer to Image Version

5.2.1 OTA CMD

UUID: 0xFFD1

This feature is the control endpoint that allows the device to enter OTA mode.

This command is used to enter DFU mode. The code for entering DFU mode is as follows:

```
1. dfu_switch_to_ota_mode();
2. WDG_SystemReset(RESET_ALL_EXCEPT_AON, DFU_SWITCH_TO_OTA);
```

Table 5-3 OTA CMD characteristics

Name	Field Requirement	Format	Value
OTA CMD	Mandatory	Uint8	1

5.2.2 Device Mac

UUID: 0xFFD2

This characteristic is used to read BDA (Bluetooth Device Address) of RTL8762E to compare with the scanned BDA in OTA mode.

Table 5-4 Device Mac characteristics

Name	Field Requirement	Format	Value
Device Mac	Mandatory	Uint8*6	XX:XX:XX:XX:XX:XX

5.2.3 Patch Version

UUID: 0xFFD3

This characteristic is used to read patch version and compatible with RTL8762A. Patch version information is described in ‘Image version’ in RTL8762C/RTL8762E.

Table 5-5 Patch Version characteristic for RTL8762C/RTL8762E

Name	Field Requirement	Format	Value
Patch Version	Mandatory	Uint32	0xNNNNNNNN

5.2.4 APP Version

UUID: 0xFFD4

This characteristic is used to read APP version and compatible with RTL8762A (not recommended in RTL8762C/RTL8762E). APP version information is described in “Image version” in RTL8762C/RTL8762E.

Table 5-6 APP Version characteristic for RTL8762C/RTL8762E

Name	Field Requirement	Format	Value
APP Version	Mandatory	Uint32	0xNNNNNNNN

5.2.5 Patch Extension Version

UUID: 0xFFD5

This characteristic is used to read patch extension version. It is only used for RTL8762A but not for RTL8762C/RTL8762E.

Table 5-7 错误!未找到引用源。Patch Extension Version characteristic

Name	Field Requirement	Format	Value
Patch extension Version	Optional	Uint16	0xNNNN

5.2.6 Test Mode

UUID: 0xFFD8

This characteristic allow device to exit control point in test mode and write ‘1’ to clear test flag to quit MP mode.

Table 5-8 Test Mode characteristics

Name	Field Requirement	Format	Value
Test mode	Optional	Uint8	1

Note: This characteristic is not related to OTA.

5.2.7 Device Info

UUID: 0xFFF1

This characteristic is used to read device information, and its description is shown below:

For the other BT SoC chip, the characteristic is as below.

Table 5-9 Device info characteristic for RTL8762C/RTL8762E

Name	Field Requirement	Format	Value
Device info	Mandatory	As Table 5-10	As Table 5-10

Table 5-10 Device info Format For RTL8762C/RTL8762E (OTA version = 1)

Format	IC Type	Version	Secure Version	MODE		Max Buffer Size	temp_bank_size
Size	8bit	8bit	8bit	8bit		16bit	8bit
Value	RTL8763:4 RTL8762C:5 RTL8762E:12	Bit3~0: OTA version:0x1 Bit7~4: Reserved:0x0		Bit0	0:normal mode 1:Support buffer check	0xNNNN	0xNNNN
				Bit1	0:Aes flag not set 1:Aes flag Set		
				Bit2	0: Only encrypt first 16 bytes of OTA data in normal mode 1:Encrypt 16*N bytes of OTA date in normal mode		
				Bit3	0: Disable Copy Image 1: Enable Copy Image		
				Bit4	0: Update one Image at a time 1: Update multiple Images at a time		

Format(Attach to above table)	Image Version Indicator 32bit
Value(Attach to above table)	0xNNNNNNNN Indications for each image version. Each indication uses 2 bits. 00: image does not exist. 01: image exists in bank0, OTA should update image for bank1. 10: image exists in bank1, OTA should update image for bank0. 11: image is standalone. OTA should update image for standalone. bit[1:0]: Image 0 … bit[2N+1:2N]:Image N Image indicator for RTL8762C/RTL8762E is as below:

Image 0	OTA Header Image
Image 1	Secure Boot Loader Image
Image 2	ROM Patch Image
Image 3	APP Image
Image 4	APP Data1 Image
Image 5	APP Data2 Image
Image 6	APP Data3 Image
Image 7	APP Data4 Image
Image 8	APP Data5 Image
Image 9	APP Data6 Image
Image 10	Upperstack Image

5.2.8 Image Counter

UUID: 0xFFF2

This characteristic is used to write response and inform device how many image files are about to be written.

Table 5-11 Image Counter characteristics

Name	Field Requirement	Format	Value
Image Counter	Optional	Uint8	0xNN

5.2.9 Image Version

UUID: 0xFFE0~FFEF

This characteristic is used to read image versions of device. Each image version occupies 4 bytes. Limited to MTU

size (20 bytes), user needs to define another characteristic (**UUID: 0xFFE0~FFEF**) to read next image version when number of image is greater than 5. The number of device img versions is indicated by Image Version Indicator, which is defined in Device Info (0xfff1).

Table 5-12 Image Version characteristics

Name	Field Requirement	Format	Value
Image Version	mandatory	Uint32*N	

5.3 OTA Procedure

The OTA procedure is divided into buffer check and no buffer check. The specific process is shown in the following figure:

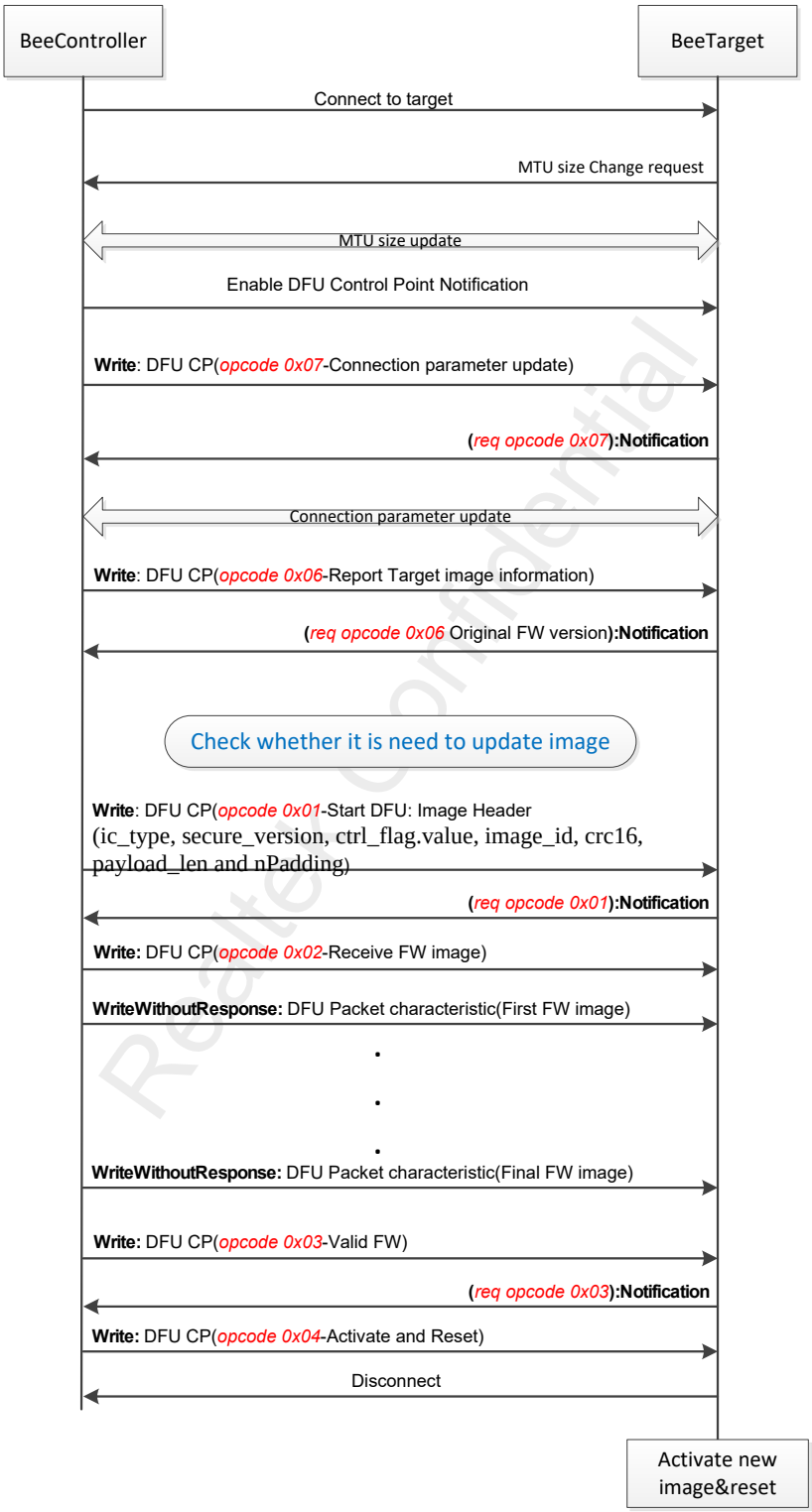

Figure 5-1 OTA procedure without buffer check

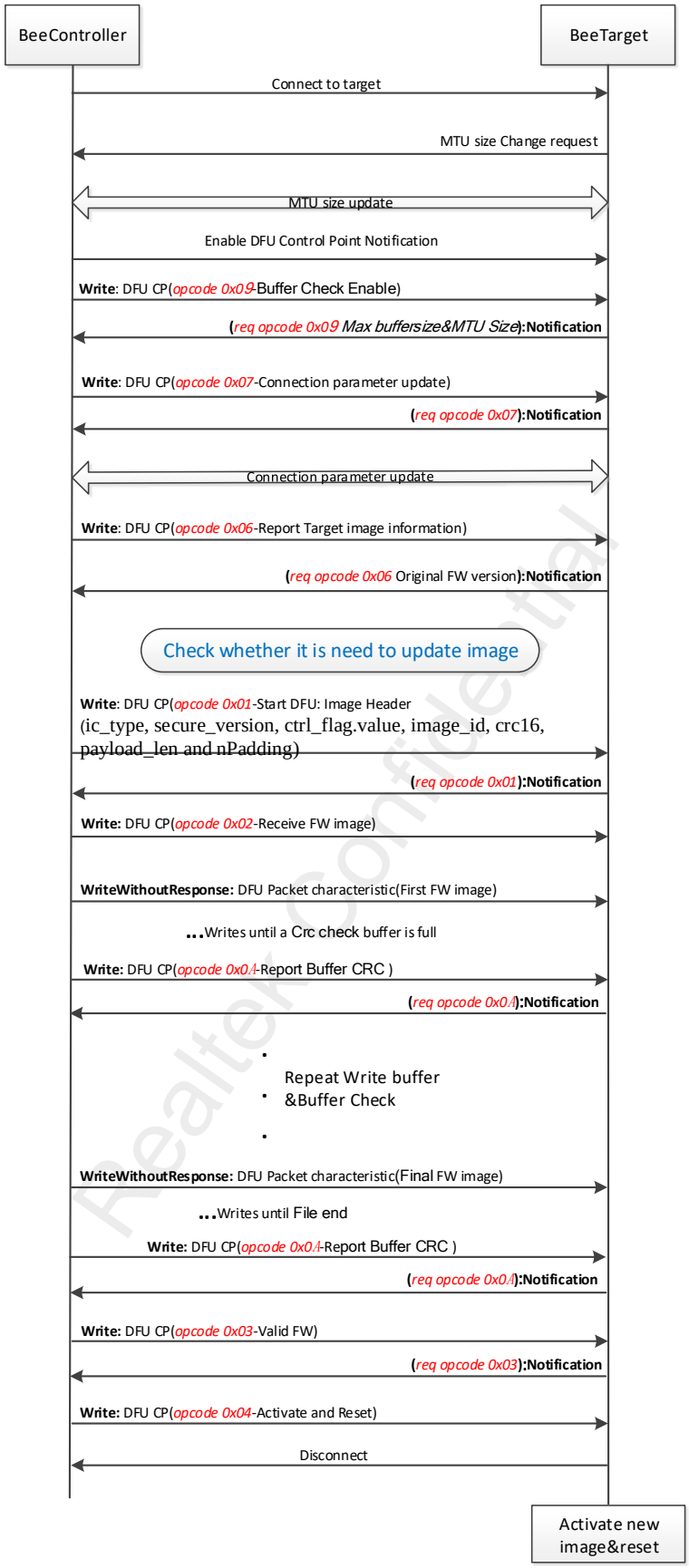

Figure 5-2 OTA procedure with buffer check

5.3.1 Multiple File Update

1. Without bank switching, a new file cannot be updated until the previous file has been verified and program has been rebooted when packaged file includes Patch, Upperstack, APP or APPDATA.
2. With bank switching, program cannot be rebooted until all the files have been updated and verified when the packaged file includes OTA Header, Patch, Upperstack, APP or APPDATA. Otherwise, this update will be invalid for that all the files in bank region must come into effect to ensure the program is running properly with bank switching.

Reference

[1]. <RTL8762E Memory User Guide>