RTL8762E Memory User Guide

V1.0

2022/06/01

Revision History

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

Contents

Table List

Figure List

1 Overview

This document describes the memory system of RTL8762E and how to use them. RTL8762E memory consists of ROM, RAM, external SPI Flash and eFuse, as shown in Table 1-1. Cache has dedicated RAM, and the dedicated RAM also can be configured as general RAM. This flexible memory configuration mechanism makes RTL8762E support a wide range of applications.

Table 1-1 Memory Layout

Memory Type	Start Addr	End Addr	Size(bytes)
ROM	0x0	0x00044000	272K
Data RAM	0x00200000	0x00216000	88K
Cache (Shared as Data RAM)	0x00216000	0x00218000	8K
Buffer RAM	0x00280000	0x00284000	16K
SPIC Flash (Cacheable)	0x00800000	0x01000000	8M
SPIC Flash (Non Cacheable)	0x01800000	0x02000000	8M

2 ROM

The ROM code is located at [0x0, 0x44000) with a total size of 272KB, in which Bootloader, RTOS, BT Stack, Flash Driver and other platform modules are built in. RTL8762E opens some modules such as RTOS, BT Stack for application to use. RTL8762E SDK contains the header files of these ROM modules which enables users to access built in ROM functions. This reduces both application code size and RAM size.

3 RAM

RTL8762E has two pieces of RAM, the Data RAM located at [0x00200000, 0x00216000) with a total size of 88KB and the Buffer RAM located at [0x00280000, 0x00284000) with a total size of 16KB. Both RAM memories can be used to store data and execute code. In the current SDK, data RAM has been used for storing data and executing code, while Buffer RAM reserved for APP has been used as the buffer heap.

3.1 Data RAM

In the SDK, Data RAM is divided into 6 parts by default, as shown in Figure 3-1. Each part has its fixed arrangement order dedicated usage, as shown in

Table 3-1.

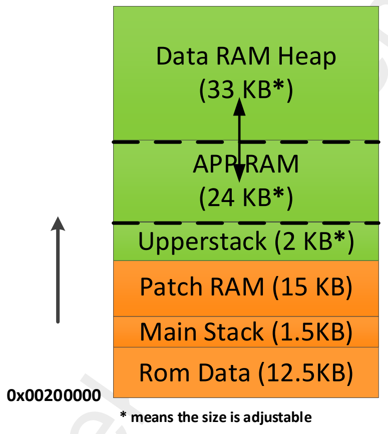


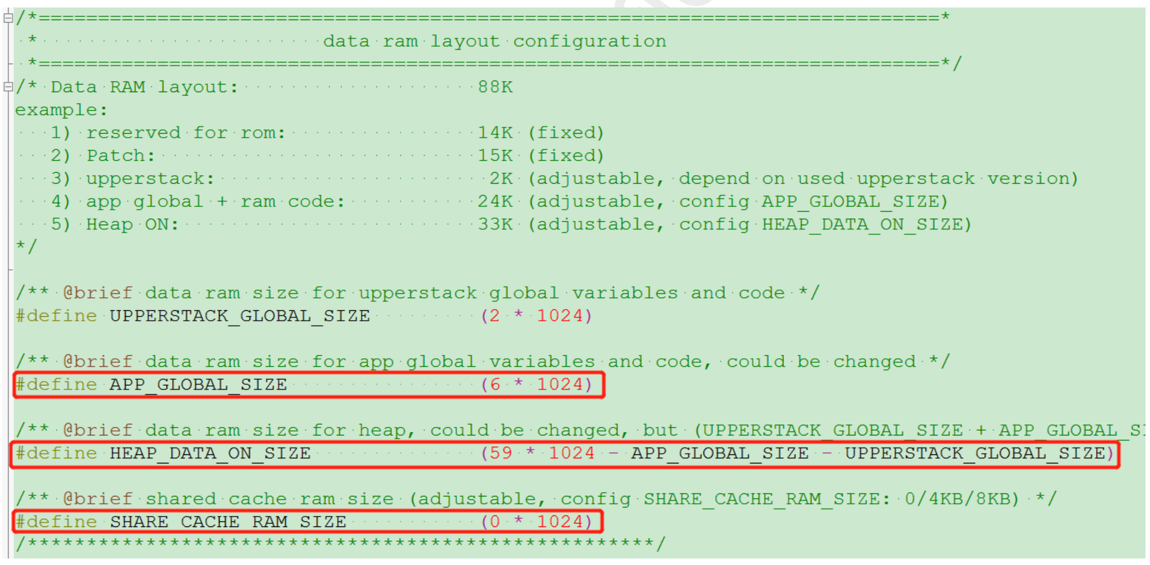
Figure 3-1 Data RAM Layout

Table 3-1 Data RAM Usage

Memory Type	Memory Usage	Memory size changeable or not
ROM data	for all global and static variables used by ROM code	NO
Main Stack	For the stack of boot code and ISR (interrupt service routine)	NO
Patch RAM	for global/static variables and RAM code of patch	NO
Upperstack RAM	for global/static variables and RAM code of Upperstack	YES
APP RAM	for global/static variables and RAM code of APP	YES

Data RAM Heap	for dynamic memory allocation of ROM code, Patch code and APP code, 33K of the 25.5K data RAM heap(Default stack configuration: link num 1, master link 0, slave link 1, enable advertising extension and PSD, all the following data are under this config parameter) has been used by rom code, so according to the above configuration the remaining 7.6K is left for APP users	YES

The total size of Upperstack RAM, APP RAM and data RAM heap are 59K, which is unchangeable, while the three space sizes of Upperstack RAM, APP RAM and data RAM heap can be modified by adjusting the size of the macros UPPERSTACK_GLOBAL_SIZE and APP_GLOBAL_SIZE in mem_config.h, as shown in Figure 3-2. UPPERSTACK_GLOBAL_SIZE value must be set according to the version of the used upperstack image. Different upperstack occupy different Data RAM size, please pay attention to the upperstack release files and notes. Adjust the value of APP_GLOBAL_SIZE according to application, and the remain size equal to DATA Heap size. It is worth noting that data heap is shared by the whole system, including ROM, Patch and APP. Preliminary statistics, the system has used 25.4KB DATA Heap by default, so it is necessary to ensure that HEAP_DATA_ON_SIZE value is not less than 25.4KB.



```
/*============================================================================*
 *                        data ram layout configuration
 *============================================================================*/
/* Data RAM layout:                   88K
example:
   1) reserved for rom:               14K (fixed)
   2) Patch:                          15K (fixed)
   3) upperstack:                      2K (adjustable, depend on used upperstack version)
   4) app global + ram code:          24K (adjustable, config APP_GLOBAL_SIZE)
   5) Heap ON:                        33K (adjustable, config HEAP_DATA_ON_SIZE)
*/

/** @brief data ram size for upperstack global variables and code */
#define UPPERSTACK_GLOBAL_SIZE        (2 * 1024)

/** @brief data ram size for app global variables and code, could be changed */
#define APP_GLOBAL_SIZE               (6 * 1024)

/** @brief data ram size for heap, could be changed, but (UPPERSTACK_GLOBAL_SIZE + APP_GLOBAL_SI
#define HEAP_DATA_ON_SIZE             (59 * 1024 - APP_GLOBAL_SIZE - UPPERSTACK_GLOBAL_SIZE)

/** @brief shared cache ram size (adjustable, config SHARE_CACHE_RAM_SIZE: 0/4KB/8KB) */
#define SHARE_CACHE_RAM_SIZE          (0 * 1024)
/******************************************************/
```

Figure 3-2 Adjust Data RAM Layout

3.2 Buffer RAM

In the current SDK, the address space of Buffer RAM is located in [0x00280000, 0x00284000] which is divided into 2 parts by default, as shown in Figure 3-3. The first 1.9 KB is used for ROM Global Data, while the other 14.1 KB is used as heap. In the 14.1 KB heap, ROM occupies about 13.5 KB dynamic spaces, while the remaining 0.6KB is used by APP, as shown in Table 3-2.

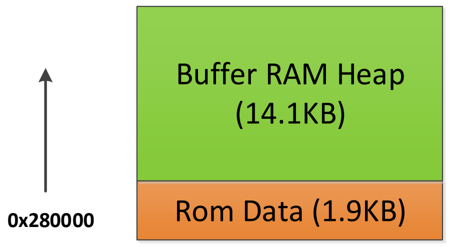


Figure 3-3 Buffer RAM Layout

Table 3-2 Buffer RAM Usage

Memory Type	Memory Usage	Memory size changeable or not
ROM data	for all global and static variables used by ROM code	NO
Buffer RAM Heap	for dynamic memory allocation of ROM code, Patch code and APP code. 13.5K of the 14.1K buffer RAM heap has been used by rom code, so 0.6K is left for application.	NO

3.3 APIs

os_mem_alloc can be used to allocate memory dynamically from Data RAM Heap or Buffer RAM heap according to the parameter of ram_type. Specific codes are as follows:

```
typedef enum
{
        RAM_TYPE_DATA_ON             = 0,
        RAM_TYPE_BUFFER_ON           = 1,
} RAM_TYPE;
/**
        * @brief  Allocate memory dynamically from Data RAM Heap or Buffer RAM heap
        * @param  ram_type : specify which heap to allocate memory from
        * @param  size: memory size in bytes to be allocated
        * @retval pointer to the allocated memory
 */
#define os_mem_alloc(ram_type, size)  os_mem_alloc_intern(ram_type, size, __func__, __LINE__)
```

Other APIs are list as below: (Refer to os_mem.h for more details)

1. os_mem_zalloc: allocated memory will be initialized to 0.
2. os_mem_aligned_alloc: allocated memory will be aligned to the specified alignment.
3. os_mem_free: free a memory block that had been allocated from data ram heap or buffer ram heap.
4. os_mem_aligned_free: free an aligned memory block that had been allocated.
5. os_mem_peek: peek the unused memory size of the specified RAM type.

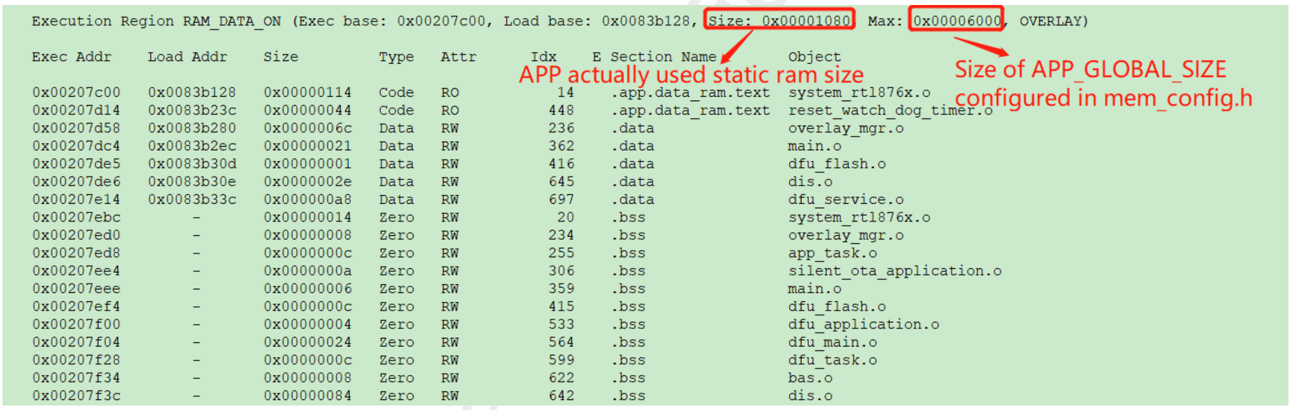
3.4 Memory Usage Calculation

At present, the RAM space of RTL8762E is two discontinuous Spaces. In order to make RAM use more convenient for APP, the memory management is optimized as follows:

1. If the VALUE of RAM_type is RAM_TYPE_DATA_ON, the system preferentially applies for the Data RAM and then applies for the Buffer RAM when the Data RAM is insufficient.
2. If ram_type is RAM_TYPE_BUFFER _ON, the request will be made from the Buffer RAM. If there is insufficient space in the Buffer RAM, an error will occur.

3.4.1 Statistics of Static Zone Size on Data RAM

Find the first address, the end address, and the size used in data RAM by the built “app.map” file. As shown in Figure 3-4, checking the map of load area “RAM_DATA_ON” can determine the first address of app data RAM. Execution Region “OVERLAY_C” determines the end address of data RAM. In the following example, the actual usage size of data RAM is 0x1080, while the configured value of APP_GLOBAL_SIZE in “mem_config.h” file is 0x6000 (24KB). APP configured Data RAM layout can be further optimized through map.

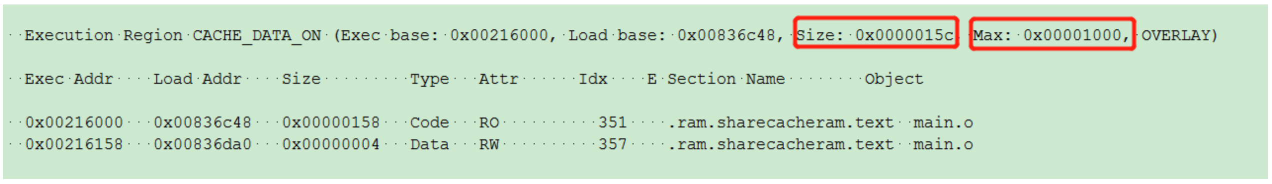

APP actually used static ram size
Size of APP_GLOBAL_SIZE configured in mem_config.h

```
Execution Region RAM_DATA_ON (Exec base: 0x00207c00, Load base: 0x0083b128, Size: 0x00001080, Max: 0x00006000, OVERLAY)

Exec Addr    Load Addr    Size         Type   Attr      Idx    E Section Name        Object

0x00207c00   0x0083b128   0x00000114   Code   RO         14     .app.data_ram.text   system_rtl876x.o
0x00207d14   0x0083b23c   0x00000044   Code   RO        448     .app.data_ram.text   reset_watch_dog_timer.o
0x00207d58   0x0083b280   0x0000006c   Data   RW        236     .data                overlay_mgr.o
0x00207dc4   0x0083b2ec   0x00000021   Data   RW        362     .data                main.o
0x00207de5   0x0083b30d   0x00000001   Data   RW        416     .data                dfu_flash.o
0x00207de6   0x0083b30e   0x0000002e   Data   RW        645     .data                dis.o
0x00207e14   0x0083b33c   0x000000a8   Data   RW        697     .data                dfu_service.o
0x00207ebc        -       0x00000014   Zero   RW         20     .bss                 system_rtl876x.o
0x00207ed0        -       0x00000008   Zero   RW        234     .bss                 overlay_mgr.o
0x00207ed8        -       0x0000000c   Zero   RW        255     .bss                 app_task.o
0x00207ee4        -       0x0000000a   Zero   RW        306     .bss                 silent_ota_application.o
0x00207eee        -       0x00000006   Zero   RW        359     .bss                 main.o
0x00207ef4        -       0x0000000c   Zero   RW        415     .bss                 dfu_flash.o
0x00207f00        -       0x00000004   Zero   RW        533     .bss                 dfu_application.o
0x00207f04        -       0x00000024   Zero   RW        564     .bss                 dfu_main.o
0x00207f28        -       0x0000000c   Zero   RW        599     .bss                 dfu_task.o
0x00207f34        -       0x00000008   Zero   RW        622     .bss                 bas.o
0x00207f3c        -       0x00000084   Zero   RW        642     .bss                 dis.o
```

Figure 3-4 Static ram size actually used in app.map

3.4.2 Statistics of Static Zone Size on Cache Shared RAM Space

Through SHARE_CACHE_RAM_SECTION decoration can specify functions or static and global variables to share cache RAM. As shown in Figure 3-, find the first address (fixed to 0x00216000) and the end address allocated in the cache shared RAM by looking up built “app.map” file. In the following example, the actual use size of share cache RAM is 0x15c, while the configured value of SHARE_CACHE_RAM_SIZE in “mem_config.h” file is 4KB, and the layout of APP configuration Cache Shared RAM can be further optimized through map.



```
  Execution Region CACHE_DATA_ON (Exec base: 0x00216000, Load base: 0x00836c48, Size: 0x0000015c, Max: 0x00001000, OVERLAY)

  Exec Addr    Load Addr    Size         Type   Attr      Idx    E Section Name        Object

  0x00216000   0x00836c48   0x00000158   Code   RO          351    .ram.sharecacheram.text  main.o
  0x00216158   0x00836da0   0x00000004   Data   RW          357    .ram.sharecacheram.text  main.o
```

Figure 3-5 Size of Cache Shared RAM in app.map

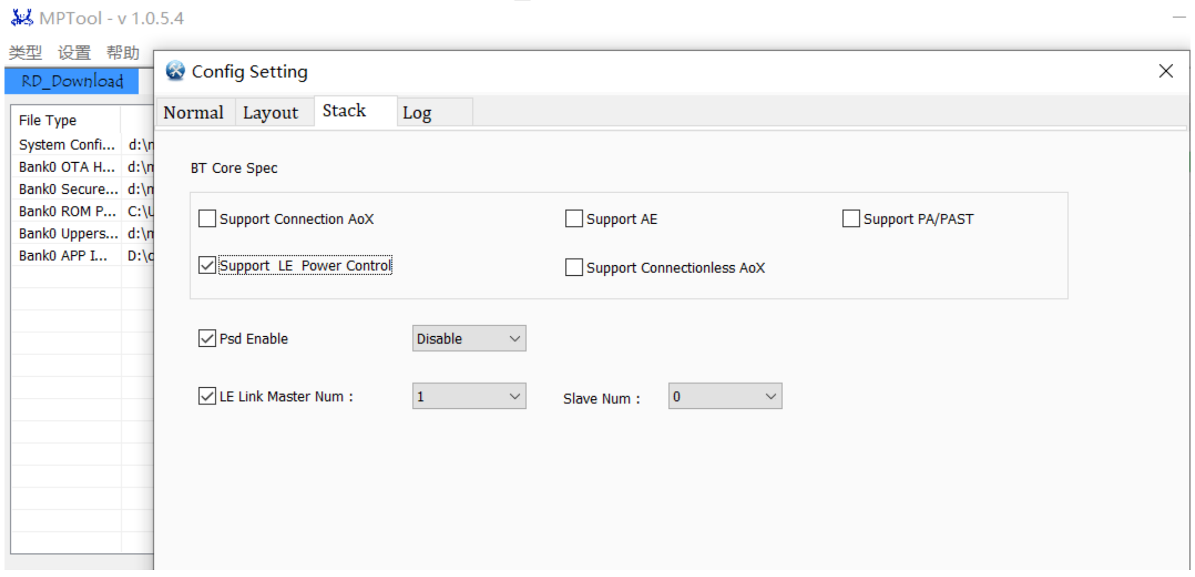
3.4.3 Statistics of Remaining Heap Size

The remaining heap size of the specified RAM type can be obtained through the os_mem_peek function.

3.5 Total RAM Size Available for APP Configuration

Under the default config setting(max link num is 1, master num is 0, and slave num is 1), the total RAM size available for APP development is 24 (Data RAM Global) + 7.5 (Data RAM Heap) + 0.6 (Buffer RAM Heap) + 8 (Cache share ram) = 40.1 KB. If you want to further increase the total size of the RAM that can be used by the APP, there are two ways:

1. Configure Bluetooth as single link: Modify the “stack” page in the Config Set option in MPTool to modify the configuration as shown in Figure 3- to add 5.5KB Data RAM Heap and 3KB Buffer RAM Heap. Therefore, the total RAM size available for APP development is increased to 24 (Data RAM Global) + 13 (Data RAM Heap) + 3.6 (Buffer RAM Heap) + 8 (Cache share ram) =48.6 KB.
2. Disable log function. Open the macro "RELEASE_VERSION" in the APP project header file “platform_autoconf.h” will disable log function, and APP can add an additional 1.5 KB Buffer RAM Heap.



☐ Support Connection AoX ☐ Support AE ☐ Support PA/PAST
☑ Support LE Power Control ☐ Support Connectionless AoX

☑ Psd Enable: Disable

☑ LE Link Master Num : 1 Slave Num : 0

Figure 3-6 Total RAM Size Available for APP Configuration

4 Cache

RTL8762E has a 8K bytes cache, it co-works with SPIC (SPI Flash Controller) to speed up the SPI Flash read and write operation. And it also can be used as data RAM. If it is configured as data RAM, it can be used for Data Storage or Code Execution. If Cache is configured as data RAM, its range is [0x00216000, 0x00218000). This range is just at the end of data RAM.

The Data RAM size of cache could be configured by setting SHARE_CACHE_RAM_SIZE micro in mem_config.h, as shown in Table 4-1.

Table 4-1 Configure Cache Usage

SHARE_CACHE_ RAM_SIZE	Flash Cache Size	Data RAM Size	Scenario
0 KB	8 KB	0 KB	Run large amounts of flash Code that requires a large piece of cache
8 KB	4 KB	4 KB	Run small amounts of flash code that requires a small piece of cache
8 KB	0 KB	8 KB	Not run flash code

5 External Flash

RTL8762E supports external SPI Flash by integrating a SPI Flash Controller (SPIC). Realtek offers the bottom level API of Flash driver, FTL for user application. SPIC supports memory mapping to SPI Flash on board and the maximum Flash size is 16M bytes. There are two range of memory mapping spaces. The range of [0x00800000, 0x01000000) is with cache, and [0x01800000, 0x02000000) is without cache. When enabled, cache will work while CPU accesses this space. This improves the data read and code execution efficiency of SPI Flash greatly.

5.1 Flash Layout

The Flash layout is set in two levels. The High Level Flash Map is determined by the config file, and the OTA Bank Map is determined by the OTA Header file.

5.1.1 High Level Flash Map

In the current SDK, the FLASH memory layout is summarized as in Figure 5-1 and consists of 7 fields : Reserved, OEM Header, OTA Bank 0, OTA Bank 1, FTL, OTA Tmp and APP Defined Section. Note that the defined starting address of the FLASH memory accessible by MCU is 0x800000. Flash layout could be adjusted by MP tool. The description of each field in the Flash layout is summarized in Table 5-1.

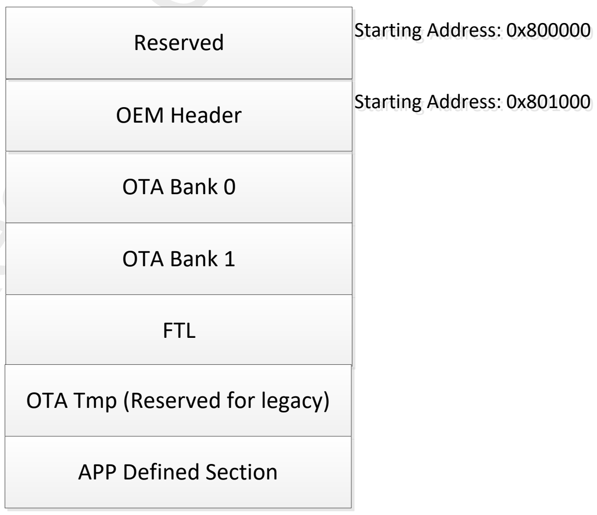

Reserved
Starting Address: 0x800000
OEM Header
Starting Address: 0x801000
OTA Bank 0
OTA Bank 1
FTL
OTA Tmp (Reserved for legacy)
APP Defined Section

Figure 5-1 Flash Layout

Table 5-1 Flash Section

Memory Segment	Starting Address	Size (Bytes)	Functions
Reserved	0x800000	0x1000	Reserved
OEM Header	0x801000	0x1000	Store configure information which includes BT Address, AES Key and user defined Flash layout.
OTA Bank0	Variable (defined in OEM Header)	Variable length (defined in OEM Header)	Store data and executable code. It can be divided into several sections: OTA Header, Secure boot, Patch, Upperstack, APP, APP Data1, APP Data2, APP Data3, APP Data4, APP Data5, APP Data6. If bank switch of OTA update is not supported, OTA_TMP is used for backup. If bank switch of OTA update is supported, one of bank0 and bank1 is executable zone, while the other is back-up zone.
OTA Bank1	Variable	Variable length	The same as bank0, and the size of bank1 must also be the same with bank0.
FTL	Variable	Variable length	Support accessing flash with logic address. User can read/write flash with unit size of 4 bytes at least.
OTA_TMP	Variable	Variable length	Used as backups for OTA when bank switch of OTA update is not supported. Its size can't be less than the largest image of OTA bank0.
APP Defined Section	Variable	Variable length	The remaining zone of the Flash. User can use it freely except for OTA update.

5.1.2 OTA Bank Map

There are 11 types of images in OTA Bank: OTA Header, Secure boot, Patch, Upperstack, APP, APP Data1, APP Data2, APP Data3, APP Data4, APP Data5 and APP Data6, the description of each part is shown in Figure 5-2, and the description of each part in the layout is shown in Table 5-2. The layout of the OTA Bank level is determined by OTA header. The Flash Map Tool in MPTool can generate OTA header with different OTA Bank layouts, as shown in Figure 5-3.

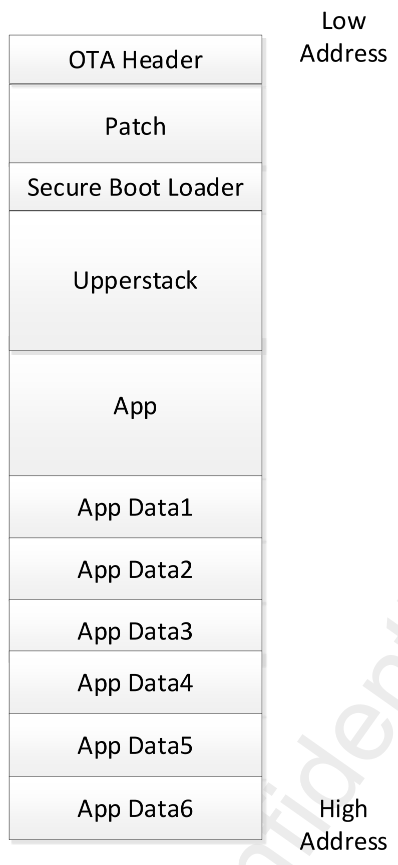
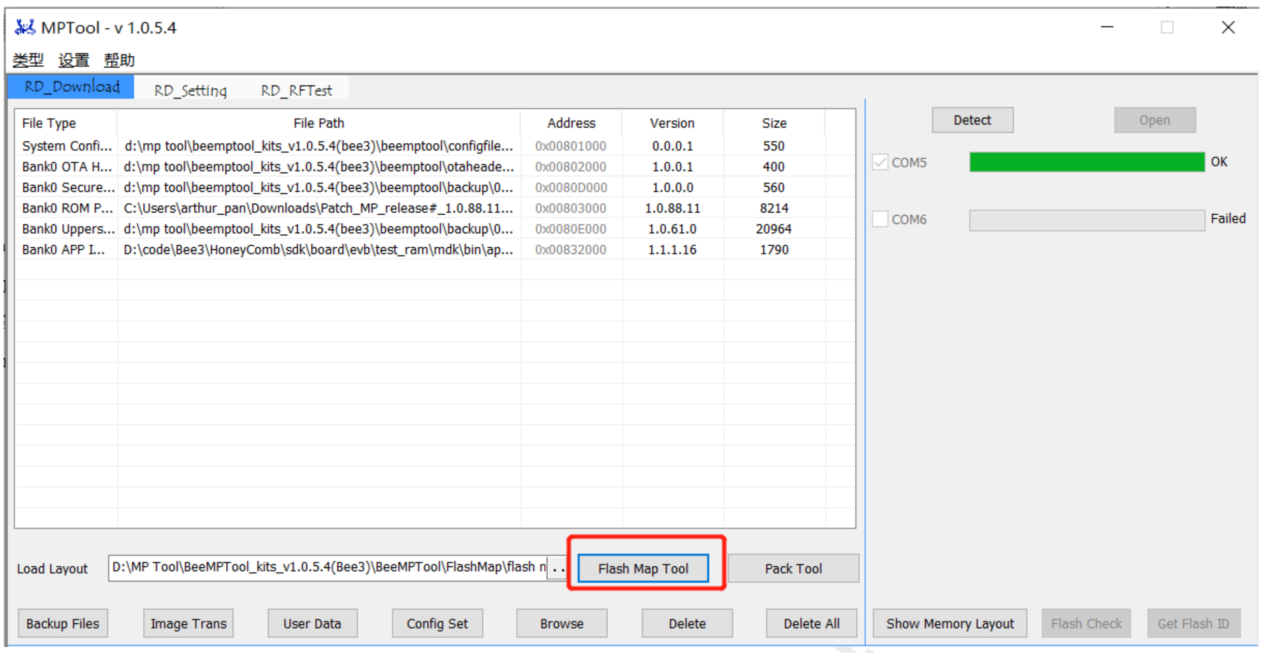


Figure 5-2 OTA Bank Layout

Table 5-2 Image Description of OTA Bank

Memory Segment	Starting Address	Size	Functions
OTA Header	Depend by OEM Header	4KB	OTA version, start address and size of each bank
Secure Boot Loader	Depend by OTA Header	changeable	Security check of code in boot process
Patch	Depend by OTA Header	changeable	Extended function of BT protocol stack and system in ROM
Upperstack	Depend by OTA Header	changeable	Part of code to implement HCI and above BT protocol stack
App	Depend by OTA Header	changeable	User application code
App Data1	Depend by OTA Header	changeable	APP Data need to be updated by OTA
App Data2	Depend by OTA Header	changeable	APP Data need to be updated by OTA
App Data3	Depend by OTA Header	changeable	APP Data need to be updated by OTA
App Data4	Depend by OTA Header	changeable	APP Data need to be updated by OTA
App Data5	Depend by OTA Header	changeable	APP Data need to be updated by OTA
App Data6	Depend by OTA Header	changeable	APP Data need to be updated by OTA



File Type	File Path	Address	Version	Size
System Confi...	d:\mp tool\beemptool_kits_v1.0.5.4(bee3)\beemptool\configfile...	0x00801000	0.0.0.1	550
Bank0 OTA H...	d:\mp tool\beemptool_kits_v1.0.5.4(bee3)\beemptool\otaheade...	0x00802000	1.0.0.1	400
Bank0 Secure...	d:\mp tool\beemptool_kits_v1.0.5.4(bee3)\beemptool\backup\0...	0x0080D000	1.0.0.0	560
Bank0 ROM P...	C:\Users\arthur_pan\Downloads\Patch_MP_release#_1.0.88.11...	0x00803000	1.0.88.11	8214
Bank0 Uppers...	d:\mp tool\beemptool_kits_v1.0.5.4(bee3)\beemptool\backup\0...	0x0080E000	1.0.61.0	20964
Bank0 APP I...	D:\code\Bee3\HoneyComb\sdk\board\evb\test_ram\mdk\bin\ap...	0x00832000	1.1.1.16	1790

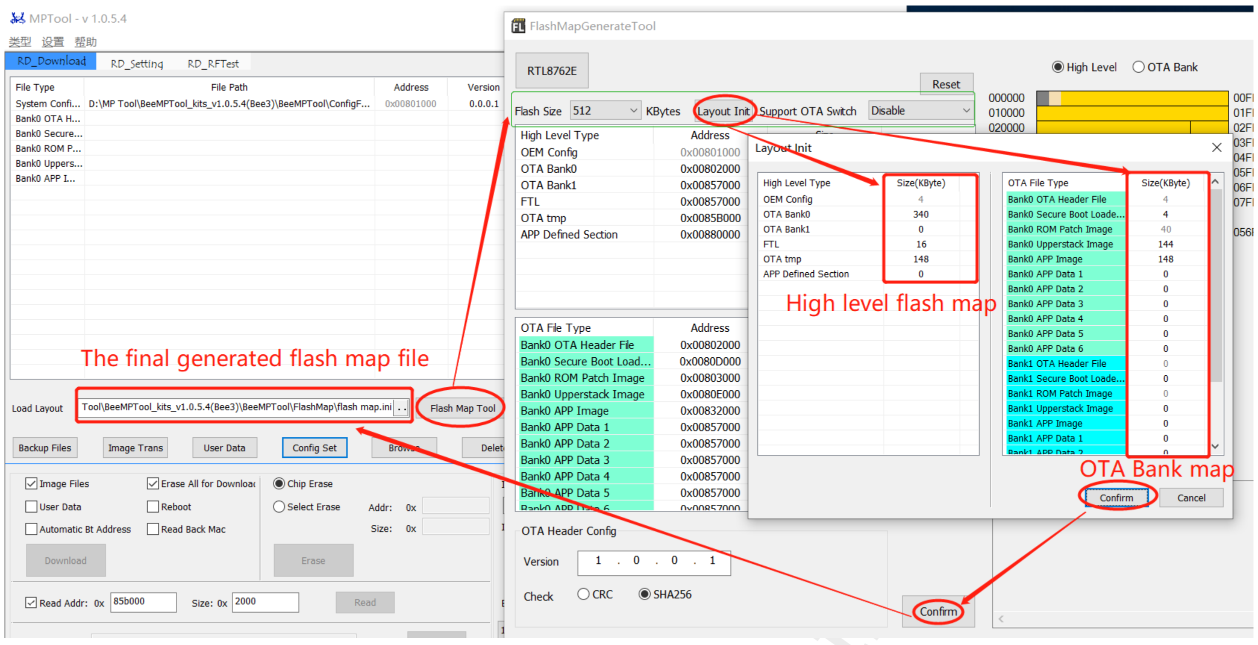
Load Layout: D:\MP Tool\BeeMPTool_kits_v1.0.5.4(Bee3)\BeeMPTool\FlashMap\flash n

Figure 5-3 The Flash Map Tool in MPTool

5.1.3 Method of configuring Flash layout

The Flash layout is set in two levels. The High Level Flash Map is determined by the config file, and the OTA Bank Map is determined by the OTA Header file.

RTL8762E supports flexible configuration of flash layout according to different application scenarios. In order to facilitate users to flexibly set the flash map, the FlashMapGenerateTool included in the BeeMPTool_kits released by Realtek can be used to set up a customized flash map and generate two flash map configuration files-"flash map.ini" and "flash_map.h" files (BeeMPTool_kits_v1.0.xx\BeeMPTool\FlashMap), and generate the OTA Header file of OTA BANK0 (BeeMPTool_kits_v1.0.xx\BeeMPTool\OTAHeader) at the same time, as shown in Figure 5-4. The "flash_map.h" file needs to be copied to the APP project path (sdk/evb/board/xxx) to generate the APP image with the correct loading address.



File Type	File Path	Address	Version
System Confi...	D:\MP Tool\BeeMPTool_kits_v1.0.5.4(Bee3)\BeeMPTool\ConfigF...	0x00801000	0.0.0.1
Bank0 OTA H...			
Bank0 Secure...			
Bank0 ROM P...			
Bank0 Uppers...			
Bank0 APP I...			

Load Layout: Tool\BeeMPTool_kits_v1.0.5.4(Bee3)\BeeMPTool\FlashMap\flash map.ini

[✓] Image Files [✓] Erase All for Download (●) Chip Erase
[ ] User Data [ ] Reboot ( ) Select Erase Addr: 0x
[ ] Automatic Bt Address [ ] Read Back Mac Size: 0x

[✓] Read Addr: 0x 85b000 Size: 0x 2000

Flash Size 512 KBytes Support OTA Switch Disable

High Level Type	Address
OEM Config	0x00801000
OTA Bank0	0x00802000
OTA Bank1	0x00857000
FTL	0x00857000
OTA tmp	0x0085B000
APP Defined Section	0x00880000

OTA File Type	Address
Bank0 OTA Header File	0x00802000
Bank0 Secure Boot Load...	0x0080D000
Bank0 ROM Patch Image	0x00803000
Bank0 Upperstack Image	0x0080E000
Bank0 APP Image	0x00832000
Bank0 APP Data 1	0x00857000
Bank0 APP Data 2	0x00857000
Bank0 APP Data 3	0x00857000
Bank0 APP Data 4	0x00857000
Bank0 APP Data 5	0x00857000
Bank0 APP Data 6	0x00857000

OTA Header Config
Version 1 . 0 . 0 . 1
Check ( ) CRC (●) SHA256

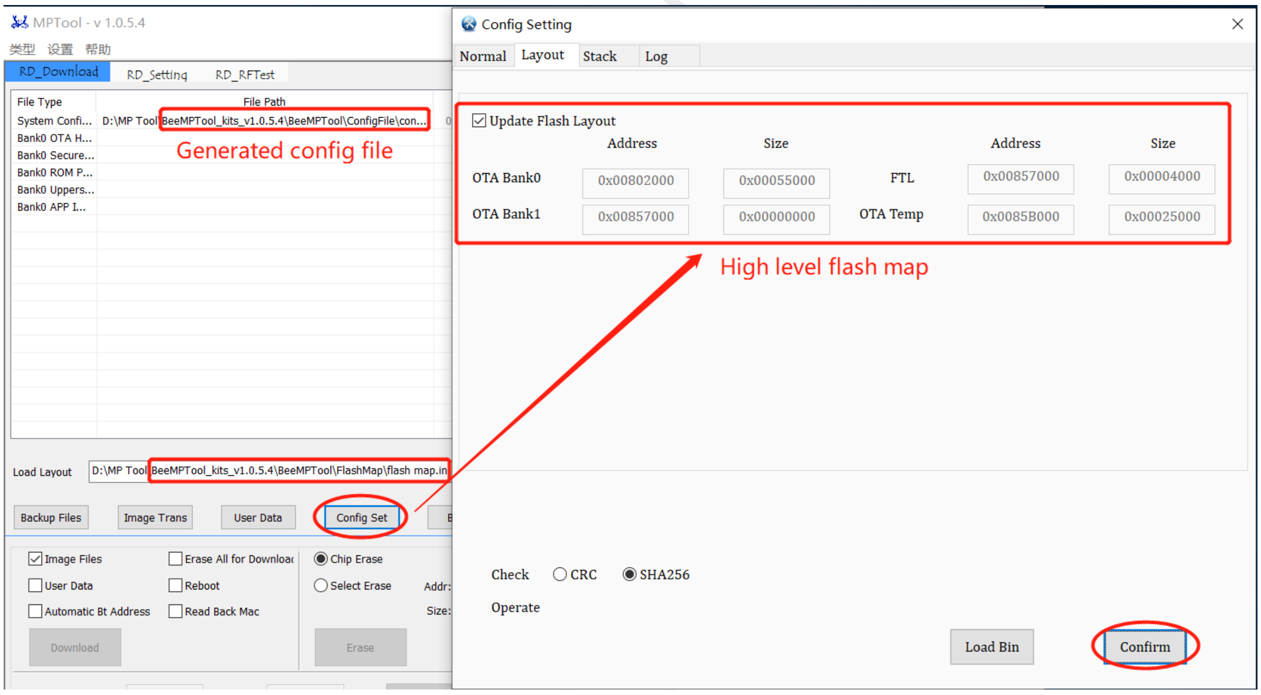
Layout Init

High Level Type	Size(KByte)
OEM Config	4
OTA Bank0	340
OTA Bank1	0
FTL	16
OTA tmp	148
APP Defined Section	0

OTA File Type	Size(KByte)
Bank0 OTA Header File	4
Bank0 Secure Boot Loade...	4
Bank0 ROM Patch Image	40
Bank0 Upperstack Image	144
Bank0 APP Image	148
Bank0 APP Data 1	0
Bank0 APP Data 2	0
Bank0 APP Data 3	0
Bank0 APP Data 4	0
Bank0 APP Data 5	0
Bank0 APP Data 6	0
Bank1 OTA Header File	0
Bank1 Secure Boot Loade...	0
Bank1 ROM Patch Image	0
Bank1 Upperstack Image	0
Bank1 APP Image	0
Bank1 APP Data 1	0
Bank1 APP Data 2	0

Figure 5-4 Configure Flash Map

After configuring the flash map, use the flash_map.ini file to generate a config file and burn it to the flash to configure the High level Flash map, as shown in Figure 5-5. Load the generated OTA Header file to configure the OTA Bank map, as shown in Figure 5-7.



File Type	File Path
System Confi...	D:\MP Tool\BeeMPTool_kits_v1.0.5.4\BeeMPTool\ConfigFile\con...
Bank0 OTA H...	
Bank0 Secure...	
Bank0 ROM P...	
Bank0 Uppers...	
Bank0 APP I...	

Load Layout: D:\MP Tool\BeeMPTool_kits_v1.0.5.4\BeeMPTool\FlashMap\flash map.ini

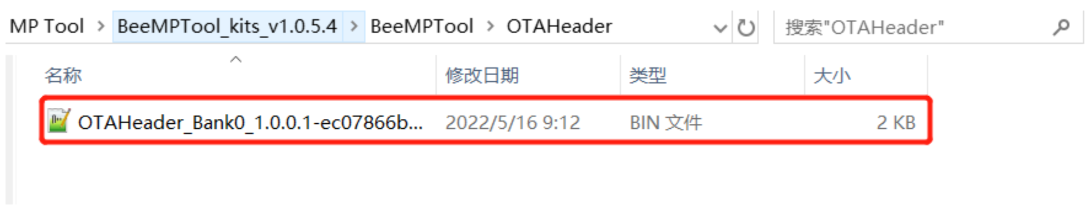
[✓] Image Files [ ] Erase All for Download (●) Chip Erase
[ ] User Data [ ] Reboot ( ) Select Erase Addr:
[ ] Automatic Bt Address [ ] Read Back Mac Size:

[✓] Update Flash Layout

	Address	Size		Address	Size
OTA Bank0	0x00802000	0x00055000	FTL	0x00857000	0x00004000
OTA Bank1	0x00857000	0x00000000	OTA Temp	0x0085B000	0x00025000

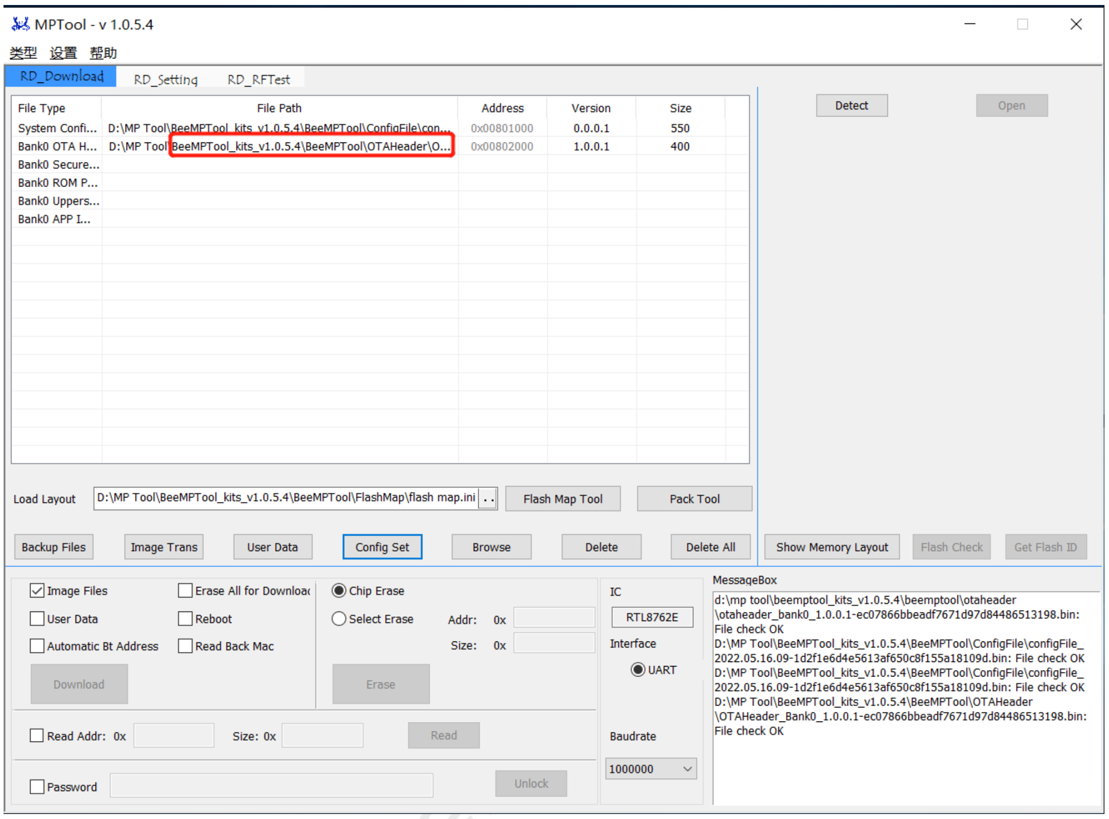
Check ( ) CRC (●) SHA256
Operate

Figure 5-5 Generate config file by MPTool



名称	修改日期	类型	大小
OTAHeader_Bank0_1.0.0.1-ec07866b...	2022/5/16 9:12	BIN 文件	2 KB

Figure 5-6 OTA Header bank0



File Type	File Path	Address	Version	Size
System Confi...	D:\MP Tool\BeeMPTool_kits_v1.0.5.4\BeeMPTool\ConfigFile\con...	0x00801000	0.0.0.1	550
Bank0 OTA H...	D:\MP Tool\BeeMPTool_kits_v1.0.5.4\BeeMPTool\OTAHeader\O...	0x00802000	1.0.0.1	400
Bank0 Secure...				
Bank0 ROM P...				
Bank0 Uppers...				
Bank0 APP I...				

Load Layout: D:\MP Tool\BeeMPTool_kits_v1.0.5.4\BeeMPTool\FlashMap\flash map.ini

[✓] Image Files
[ ] User Data
[ ] Automatic Bt Address
[ ] Erase All for Download
[ ] Reboot
[ ] Read Back Mac
(●) Chip Erase
( ) Select Erase
Addr: 0x
Size: 0x
[ ] Read Addr: 0x
Size: 0x
[ ] Password
IC: RTL8762E
Interface: (●) UART
Baudrate: 1000000

MessageBox

```
d:\mp tool\beemptool_kits_v1.0.5.4\beemptool\otaheader
\otaheader_bank0_1.0.0.1-ec07866bbeadf7671d97d84486513198.bin:
File check OK
D:\MP Tool\BeeMPTool_kits_v1.0.5.4\BeeMPTool\ConfigFile\configFile_
2022.05.16.09-1d2f1e6d4e5613af650c8f155a18109d.bin: File check OK
D:\MP Tool\BeeMPTool_kits_v1.0.5.4\BeeMPTool\ConfigFile\configFile_
2022.05.16.09-1d2f1e6d4e5613af650c8f155a18109d.bin: File check OK
D:\MP Tool\BeeMPTool_kits_v1.0.5.4\BeeMPTool\OTAHeader
\OTAHeader_Bank0_1.0.0.1-ec07866bbeadf7671d97d84486513198.bin:
File check OK
```

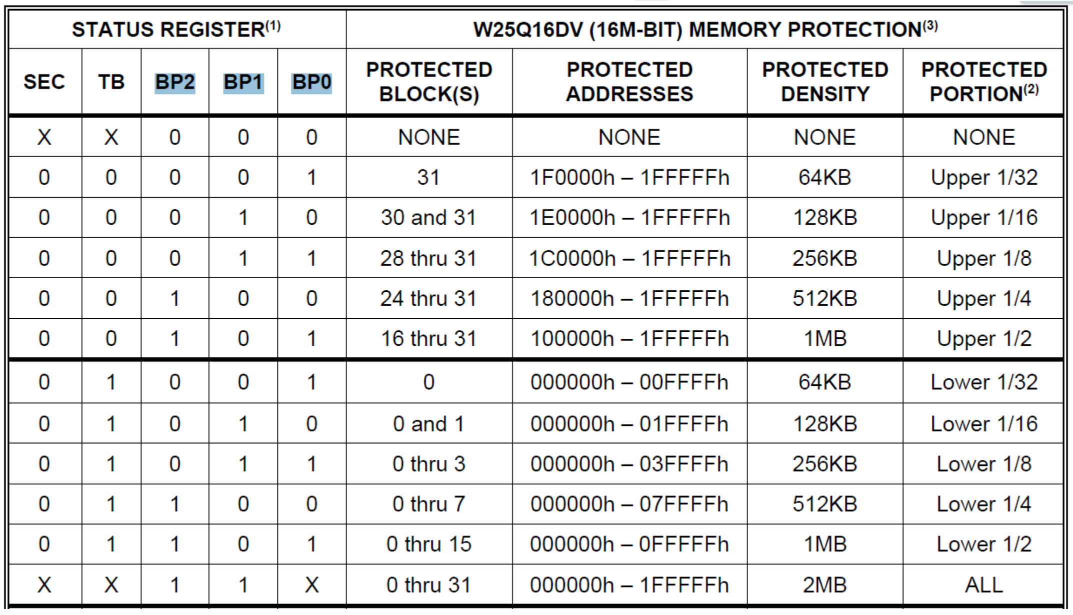
Figure 5-7 OTA Header file

But you must pay attention to some of the principles of adjusting flash layout as described below.

1. If OTA supports bank switch, size of OTA bank0 and bank1 must be equal, while size of OTA tmp should be set to 0.
2. If OTA supports bank switch, layout of OTA bank0 and bank1 can be different. But the size of bank0 and bank1 cannot be changed and less than the total size of internal images.
3. If OTA doesn't support bank switch, size of OTA bank1 is set to 0. The size of OTA tmp can't be less than size of the largest image in OTA bank0.
4. **If OTA doesn't support bank switch, besides OTA Header, patch, Upperstack and APP, user should burn secure boot image. The address should be fixed to 0x80D0000, and size must be less than 4KB. The OTA solution that supports bank switching also needs to burn the secure boot image of bank1. The burning address is determined according to the flash_map, and the size is at least 4KB.The secure boot image is released by Realtek.**

5. The original APP Data bin file needs to be processed by the App Data tool provided in the RTL8762E SDK before it can be burned with MPTool and packaged with MPPackTool. The tool path is in sdk/tool/AppData. After processing, a 1KB image header meeting a specific format will be added. Note that, sections of APP Data1, APP Data2, APP Data3, APP Data4, APP Data5 and APP Data6 are used as storage of app data for OTA. If the data don't need to update, it should be placed in the APP Defined Section.
6. Make sure the offset between end address of OTA bank1 image and flash start address (0x800000) aligned to flash protected address range, such as 64KB, 128KB, 256KB and 512KB. This can lock all the code zone to protect from unexpected flash write and erase operation.
7. Because the data stored in the APP Data1, APP Data2, APP Data3, APP Data4, APP Data5 and APP Data6 areas in the OTA bank may fall within the flash lock range, the data stored in this area is generally read-only. If you need to store the data that needs to be overwritten, you should put it in the App Defined Section area.

As to the 6th principle, RTL8762E supports a mechanism named flash software Block Protect to lock flash to prevent writing and erasing operations. Flash software block protect uses some BP bits in flash status register to select the level (range) to protect as described below.



STATUS REGISTER[1]					W25Q16DV (16M-BIT) MEMORY PROTECTION[3]			
SEC	TB	BP2	BP1	BP0	PROTECTED BLOCK(S)	PROTECTED ADDRESSES	PROTECTED DENSITY	PROTECTED PORTION[2]
X	X	0	0	0	NONE	NONE	NONE	NONE
0	0	0	0	1	31	1F0000h – 1FFFFFh	64KB	Upper 1/32
0	0	0	1	0	30 and 31	1E0000h – 1FFFFFh	128KB	Upper 1/16
0	0	0	1	1	28 thru 31	1C0000h – 1FFFFFh	256KB	Upper 1/8
0	0	1	0	0	24 thru 31	180000h – 1FFFFFh	512KB	Upper 1/4
0	0	1	0	1	16 thru 31	100000h – 1FFFFFh	1MB	Upper 1/2
0	1	0	0	1	0	000000h – 00FFFFh	64KB	Lower 1/32
0	1	0	1	0	0 and 1	000000h – 01FFFFh	128KB	Lower 1/16
0	1	0	1	1	0 thru 3	000000h – 03FFFFh	256KB	Lower 1/8
0	1	1	0	0	0 thru 7	000000h – 07FFFFh	512KB	Lower 1/4
0	1	1	0	1	0 thru 15	000000h – 0FFFFFh	1MB	Lower 1/2
X	X	1	1	X	0 thru 31	000000h – 1FFFFFh	2MB	ALL

Figure 5-8 Flash Software Block Protect

Flash uses BP(x) bits in status register to identify number of blocks to lock, and TB bit to decide the direction to lock. However, Realtek only supports lock flash from low address to protect some important data such as configuration, security, and code sections. In order to support this feature, it is necessary to pass some checking rules to guarantee selected flash meets our requirement for software Block Protect. That is why Approved Vendor List (AVL) exists. Most flash in Qualified Vendor List supports protecting flash by level for different size, such as 64KB, 128KB, 256KB, 512KB, etc.

Therefore, when we divide flash layout, we need to ensure that the end address offset of OTA bank1 segment can be aligned to a certain level of protection that the selected flash supports as far as possible. Then RTL8762E will parse flash layout configuration parameters and query selected flash information to set block protect value. In order to maximize the use of BP, some flash layout examples are as follows. But what you have to notice is that there are two kinds of flash in AVL, which does not support protecting flash by block level. More details please refer to Qualified Vendor List.

The protected flash zone can't be written and erased. If necessary, user can unlock flash first, and then write or erase, finally lock the flash to the previous level. But this operation is not recommended, as the flash status register is accessed by way of NVRAM. There is 100K times limit, so frequently unlocking may make the flash unavailable.

5.1.4 Flash layout example

Based on the rules for adjusting the flash layout described above, this section provides flash layout examples under two OTA schemes.

Table 5-3 OTA doesn't support bank switch Sample Flash layout (total flash size is 512KB)

Sample Flash Layout (total size is 512KB)	Size	Start Address	Block Protect size
1) Reserved	4K	0x00800000	The front 256KB space starting from flash low address (OTA Bank1 end address offset is 340KB. It is not aligned so just lock 256KB.)
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	Unlocked region
6) OTA Temp	148K	0x0085B000	
7) APP Defined Section	0K	0x00880000	

Table 5-4 OTA supports bank switch Sample Flash layout (total flash size is 1MB)

Sample Flash Layout (Total size is 1MB)	Size	Start Address	Block Protect Size
1) Reserved	4K	0x00800000	The front 512KB space starting from flash low address (OTA Bank1 end address offset is 808KB. It is not aligned so just lock 512KB)
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 as 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	Unlocked region
6) OTA Temp	0K	0x008DA000	
7) APP Defined Section	152K	0x008DA000	

5.2 Flash APIs

Flash operation APIs are listed as follows, refer to Bee3-SDK.chm and RTL8762E Flash User Guide for more details. The APIs with “auto” will access flash with auto mode, the others will access flash with user mode.

1. Basic Operation APIs:
   1) bool flash_auto_read_locked(uint32_t addr, uint32_t *data);
   2) bool flash_read_locked(uint32_t start_addr, uint32_t data_len, uint8_t *data);
   3) bool flash_auto_write_locked(uint32_t start_addr, uint32_t data);
   4) bool flash_auto_write_buffer_locked(uint32_t start_addr, uint32_t *data, uint32_t len);
   5) bool flash_write_locked(uint32_t start_addr, uint32_t data_len, uint8_t *data);
   6) bool flash_erase_locked(T_ERASE_TYPE type, uint32_t addr);

2. Flash High Speed Read APIs:
   1) bool flash_auto_dma_read_locked(T_FLASH_DMA_TYPE dma_type, FlashCB flash_cb,uint32_t src_addr, uint32_t dst_addr, uint32_t data_len);
   2) bool flash_auto_seq_trans_dma_read_locked(T_FLASH_DMA_TYPE dma_type, FlashCB flash_cb, uint32_t src_addr, uint32_t dst_addr, uint32_t data_len);
   3) bool flash_split_read_locked(uint32_t start_addr, uint32_t data_len, uint8_t *data, uint32_t *counter);

If you need to use the above three interface functions for high-speed reading of flash, you need to include the header file "flash_device.h". For detailed usage, please refer to RTL8762E Flash User Guide.

5.3 FTL

FTL (flash transport layer) is used as abstraction layer for BT stack and user application to read/write data in flash, and convert the physical space of the flash into the logical space of the FTL storage without worrying about the constraints of the flash driver. Through FTL interface, user can read or write the responding data in flash space for FTL by logic address.

Generally, FTL space is more suitable for storing data that needs to be rewritten frequently. On the one hand, it is simpler to use FTL interface. On the other hand, FTL operation will actually recycle the allocated physical space so as not to write multiple times to a flash sector and affect the life of flash, and due to the conversion from physical space to logical space, about half of the flash space is lost, which is not suitable for storing large amounts of data. If you need to store a large amount of data and only need to overwrite occasionally, or store read-only data but do not need to upgrade, you can call flash APIs in the ‘App Defined Section’ area for management. If you need to store read-only data and upgrade it, place it in App Data1, App Data2, App Data3, App Data4, App Data5 and App Data6 areas in OTA bank.

5.3.1 FTL Storage Space

The FTL space can be divided into 2 spaces according to functions. Take the default physical space size of FTL as 16K:

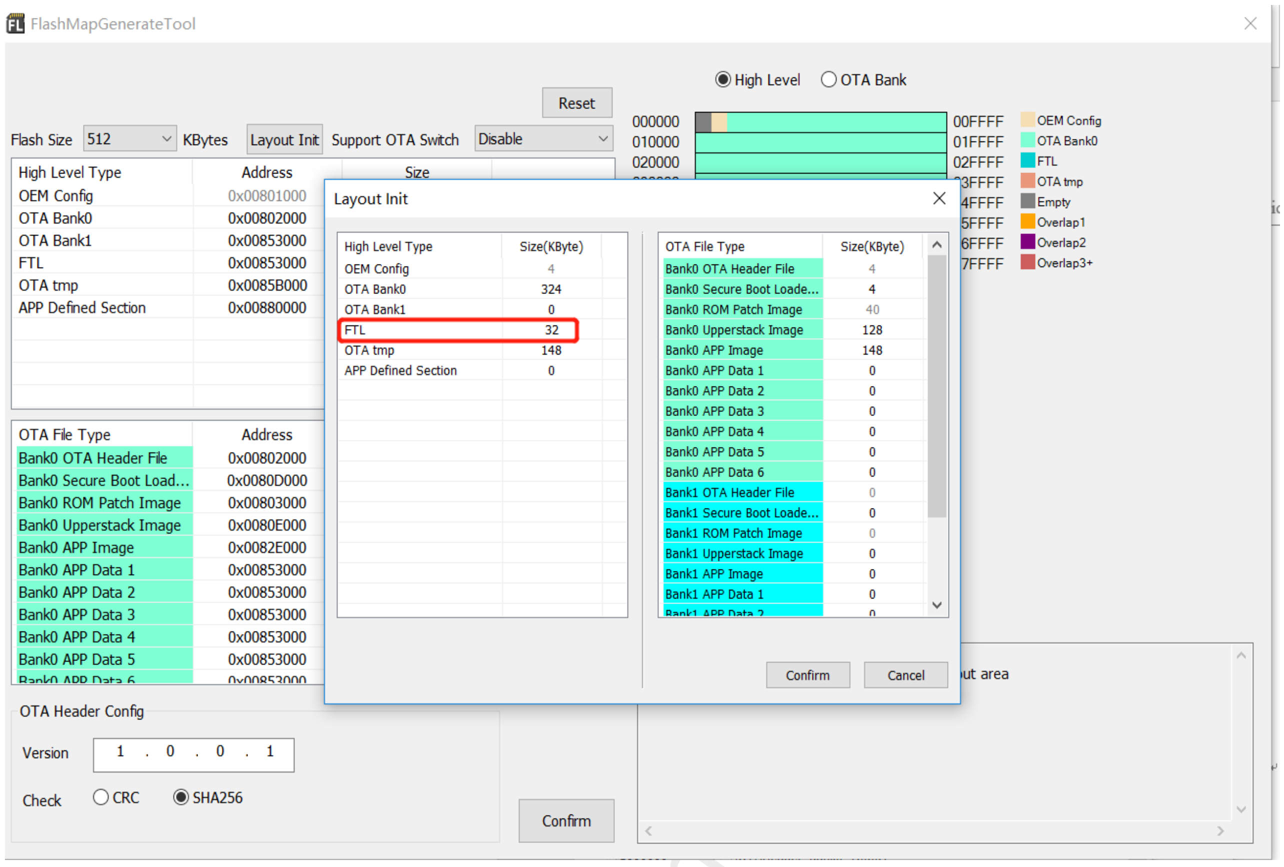
1. BT storage space
   1) Logic address range: [0x0000, 0x0400). But this space size can be changed by otp parameter.
   2) This region is used to store BT information such as device address, link key, etc.
   3) Refer to RTL8762E BLE Stack User Manual for more details.
2. APP storage space
   1) The start of the logical address is 0x0400, and the end address is related to the otp configuration. The default FTL_REAL_LOGIC_ADDR_SIZE is 3K.
   2) APP can use this region to store user defined information.
   3) The following APIs can be called to read/write data in this region, and they are defined in ftl.h. Please refer to Bee3-SDK.chm for more details. Note that when calling ftl_save or ftl_load, the bottom layer has already encapsulated an offset, which is 0x0400 by default.

```
uint32_t ftl_save(void * p_data, uint16_t offset, uint16_t size)
uint32_t ftl_load(void * p_data, uint16_t offset, uint16_t size)
```

5.3.2 Adjust FTL Space Size

The physical space and logical space size of FTL are configurable. The physical space of FTL is adjusted by modifying the configuration parameters of config file, and the logical space size of FTL is adjusted by adjusting OTP configuration. The size adjustment of logical space is limited by the configured physical space size.

1. The steps to adjust the physical space are as follows:
   1) First, use FlashMapGenerateTool which released with MPTool to generate "flash map.ini" and "flash_map.h" file.
   2) Copy the "flash_map.h" file to the app project directory, for example "\sdk\board\evb\silent_ota", so that the correct load address can be obtained when the app is compiled.
   3) Load "flash map.ini" into MPTool to generate config file for download. As shown in Figure 5-9, the physical space of FTL will be adjusted to 32K.



High Level Type	Address	Size
OEM Config	0x00801000	
OTA Bank0	0x00802000	
OTA Bank1	0x00853000	
FTL	0x00853000	
OTA tmp	0x0085B000	
APP Defined Section	0x00880000	

OTA File Type	Address
Bank0 OTA Header File	0x00802000
Bank0 Secure Boot Load...	0x0080D000
Bank0 ROM Patch Image	0x00803000
Bank0 Upperstack Image	0x0080E000
Bank0 APP Image	0x0082E000
Bank0 APP Data 1	0x00853000
Bank0 APP Data 2	0x00853000
Bank0 APP Data 3	0x00853000
Bank0 APP Data 4	0x00853000
Bank0 APP Data 5	0x00853000
Bank0 APP Data 6	0x00853000

OTA Header Config

Version 1 . 0 . 0 . 1

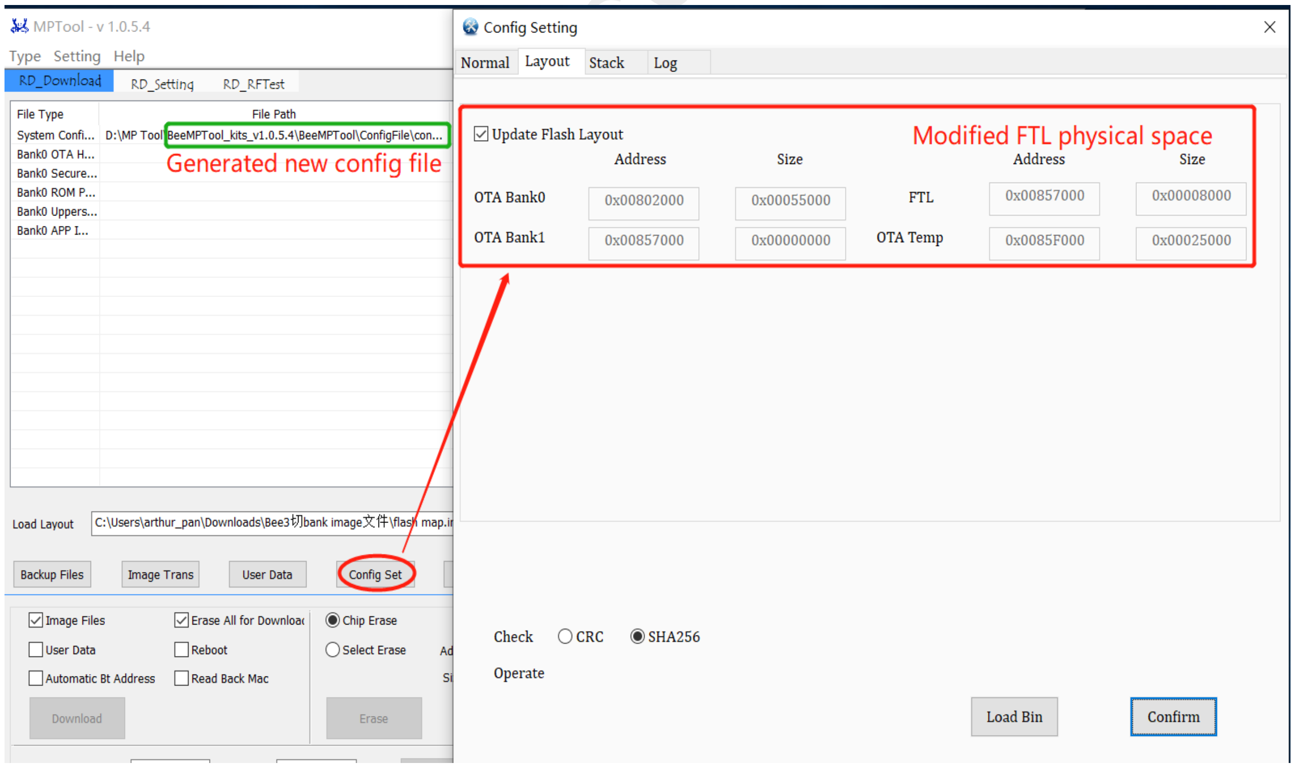
Check ( ) CRC [✓ SHA256]

Layout Init

High Level Type	Size(KByte)
OEM Config	4
OTA Bank0	324
OTA Bank1	0
FTL	32
OTA tmp	148
APP Defined Section	0

OTA File Type	Size(KByte)
Bank0 OTA Header File	4
Bank0 Secure Boot Loade...	4
Bank0 ROM Patch Image	40
Bank0 Upperstack Image	128
Bank0 APP Image	148
Bank0 APP Data 1	0
Bank0 APP Data 2	0
Bank0 APP Data 3	0
Bank0 APP Data 4	0
Bank0 APP Data 5	0
Bank0 APP Data 6	0
Bank1 OTA Header File	0
Bank1 Secure Boot Loade...	0
Bank1 ROM Patch Image	0
Bank1 Upperstack Image	0
Bank1 APP Image	0
Bank1 APP Data 1	0
Bank1 APP Data 2	0

Figure 5-9 Adjust Flash Map Config File



File Type	File Path
System Confi...	D:\MP Tool\BeeMPTool_kits_v1.0.5.4\BeeMPTool\ConfigFile\con...
Bank0 OTA H...	
Bank0 Secure...	
Bank0 ROM P...	
Bank0 Uppers...	
Bank0 APP I...	

Generated new config file

Load Layout C:\Users\arthur_pan\Downloads\Bee3切bank image文件\flash map.i

[✓] Image Files [✓] Erase All for Download [✓ Chip Erase]
[ ] User Data [ ] Reboot ( ) Select Erase
[ ] Automatic Bt Address [ ] Read Back Mac

[✓] Update Flash Layout

Modified FTL physical space

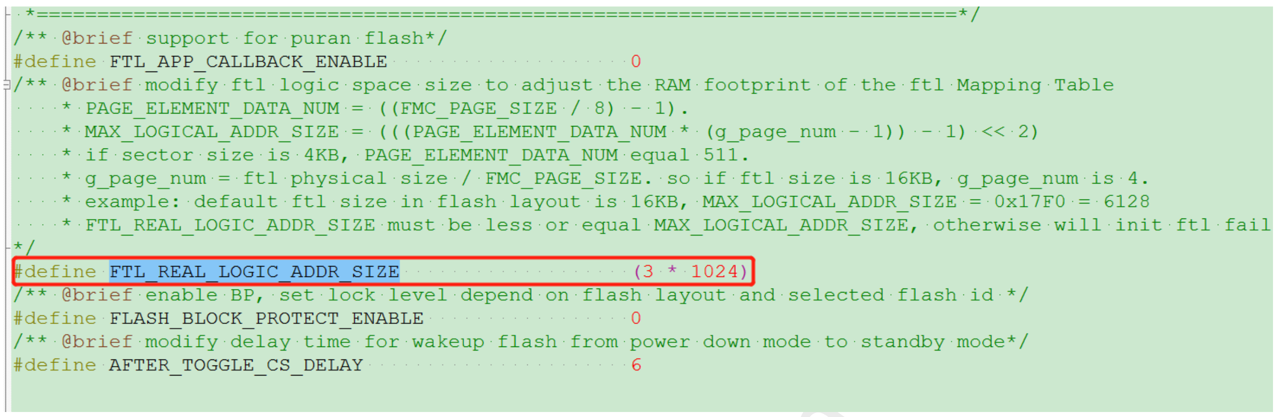
	Address	Size		Address	Size
OTA Bank0	0x00802000	0x00055000	FTL	0x00857000	0x00008000
OTA Bank1	0x00857000	0x00000000	OTA Temp	0x0085F000	0x00025000

Check ( ) CRC [✓ SHA256]

Operate

Figure 5-10 Generate config file

2. The steps to adjust the logical space are as follows:
    1) Modify the size of the macro ‘FTL_REAL_LOGIC_ADDR_SIZE’ in otp_config.h, the value is 3KB by default, and the FTL mapping table occupies 1248 bytes of RAM resources.
    2) In the case that the FTL physical space size is set to 16K by default, the maximum logical space size is 0x17f0.



```
 *========================================================================*/
/** @brief support for puran flash*/
#define FTL_APP_CALLBACK_ENABLE                    0
/** @brief modify ftl logic space size to adjust the RAM footprint of the ftl Mapping Table
    * PAGE_ELEMENT_DATA_NUM = ((FMC_PAGE_SIZE / 8) - 1).
    * MAX_LOGICAL_ADDR_SIZE = (((PAGE_ELEMENT_DATA_NUM * (g_page_num - 1)) - 1) << 2)
    * if sector size is 4KB, PAGE_ELEMENT_DATA_NUM equal 511.
    * g_page_num = ftl physical size / FMC_PAGE_SIZE. so if ftl size is 16KB, g_page_num is 4.
    * example: default ftl size in flash layout is 16KB, MAX_LOGICAL_ADDR_SIZE = 0x17F0 = 6128
    * FTL_REAL_LOGIC_ADDR_SIZE must be less or equal MAX_LOGICAL_ADDR_SIZE, otherwise will init ftl fail
*/
#define FTL_REAL_LOGIC_ADDR_SIZE                   (3 * 1024)
/** @brief enable BP, set lock level depend on flash layout and selected flash id */
#define FLASH_BLOCK_PROTECT_ENABLE                 0
/** @brief modify delay time for wakeup flash from power down mode to standby mode*/
#define AFTER_TOGGLE_CS_DELAY                      6
```

Figure 5-11 Adjust the size of FTL logical space

Note:

1. When adjusting the size of FTL physical space, the logical space available to app will also change accordingly. Assuming that the physical space size of the actual FTL is set to MK (M is an integral multiple of 4), the logical space available for the corresponding app is equal to ((511 * (M-4) - 4) - 1024) bytes.
2. In addition, in order to improve the efficiency of FTL reading, a set of mapping mechanism of physical address and logical address is designed in the bottom layer. This mapping table will occupy a certain amount of RAM space. When the default FTL physical space size is 16K, the mapping table takes up 2298 bytes of buffer RAM heap space. Assuming that the physical space size of the actual FTL is set to MK (M is an integral multiple of 4), the RAM space occupied by its mapping table is equal to ((511 * (M-4) - 4) * 0.375) bytes.
3. Therefore, users need to adjust the size of FTL space reasonably according to specific application scenarios. If the size is too large, some RAM resources will be wasted. On the other hand, if you choose a smaller flash and want to compress the space occupied by FTL, you must ensure that the physical space of FTL is not smaller than 12K.
4. At the same time, since the physical address corresponding to each logical address in the mapping table is represented by 12 bits by default, the maximum physical space of FTL can be adjusted to 36K. If the physical space of 36KB is not enough to meet the application requirements, the number of bits representing the physical address can also be set to 16bit. At the same time, because the logical address stored in flash is 16bit, now the maximum physical space corresponding to FTL can be adjusted to 132K.

6 Flash Code and RAM Code Setting

The code can run on Flash or on RAM. This section describes how to place code in a specific memory to execute.

1. Modify the macro FEATURE_RAM_CODE definition:
   1) 1 indicates that the code without any section modified runs on RAM.
   2) 0 indicates that the code without any section modified runs on Flash.
2. If you want to specify a function to place on a specific memory, use the section macro in app_section.h. For example:
   1) APP_FLASH_TEXT_SECTION means putting the function into Flash to execute.
   2) DATA_RAM_FUNCTION means putting the function into RAM to execute.
   3) SHARE_CACHE_RAM_SECTION means putting functions, global and static variables on shared cache RAM.

7 eFuse

eFuse is a block of one-time programming memory which is used to store the important and fixed information, such as UUID, security key and other one-time programming configuration. The single bit of eFuse cannot be changed from 0 to 1, and there is no erase operation to eFuse, so be careful to update eFuse. Realtek offers MP Tool to update certain eFuse sections.

Reference

[1] RTL8762E BLE Stack User Manual.

[2] Bee3-SDK.chm

[3] RTL8762E Flash User Guide.