

RTL8762D Memory User Guide

V1.0

2020/06/05

修订历史 (Revision History)

日期	版本	修改	作者	Reviewer
2020/06/05	V1.0	发布第一版	Grace	Lory

Realtek Confidential

目录

修订历史 (Revision History)	2
目录	3
图目录	4
表目录	5
1 概述	6
2 ROM	7
3 RAM	8
3.1 Data RAM	8
3.2 Buffer RAM	9
3.3 APIs	10
3.4 统计 RAM 使用情况	11
3.4.1 统计 Data RAM 使用的静态区大小	11
3.4.2 统计 Cache Shared RAM 使用的静态区大小	12
3.4.3 统计动态区剩余大小	12
3.5 调整 APP 可使用 RAM 的总大小	13
4 Cache	14
5 External Flash	15
5.1 Flash 布局	15
5.2 Flash APIs	23
5.3 FTL	24
5.3.1 FTL 空间的功能划分	24
5.3.2 调整 FTL 空间的大小	25
6 设置代码执行位置	27
7 eFuse	28
Reference	29

图目录

图 3-1 Data RAM 布局	8
图 3-2 调整 Data RAM 布局	9
图 3-3 Buffer RAM 布局	10
图 5-1 Flash 布局	15
图 5-2 OTA Bank 布局	17

表目录

表 1-1 内存布局	6
表 3-1 Data RAM 用途	8
表 3-2 Buffer RAM 用途	10
表 3-3 os_mem_alloc	10
表 4-1 配置 Cache 用途	14
表 5-1 Flash 段	15

1 概述

本文主要介绍 RTL8762D 的内存系统以及怎样使用各部分内存。RTL8762D 的内存系统主要由 ROM、RAM、External Flash、Flash Cache 和 Efuse 构成，如表 1-1 所示。Cache 具有专用 RAM，并且这个专用 RAM 也可以被配置成通用 RAM。这种灵活的内存配置机制使得 RTL8762D 支持的应用更为广泛。

表 1-1 内存布局

内存类型	起始地址	结束地址	大小（字节）
ROM	0x0	0x00040000	256K
Data RAM	0x00200000	0x00224000	144K
Cache (Shared as Data RAM)	0x00224000	0x00228000	16K
Buffer RAM	0x00280000	0x00288000	32K
SPIC Flash (Cacheable)	0x00800000	0x02800000	32M
SPIC Flash (Non Cacheable)	0x04800000	0x06800000	32M

2 ROM

ROM 的地址空间为[0x0, 0x40000), 总大小是 256KB, 固化了 OS、BT stack、Flash Driver 及 Platform features 等程序。RTL8762D 为 APP 开放了一些模块, 如 RTOS、BT stack。RTL8762D SDK 包含这些 ROM 模块的头文件, 用户可以访问内置的 ROM 函数, 以减少应用程序代码大小和所需的 RAM 大小。

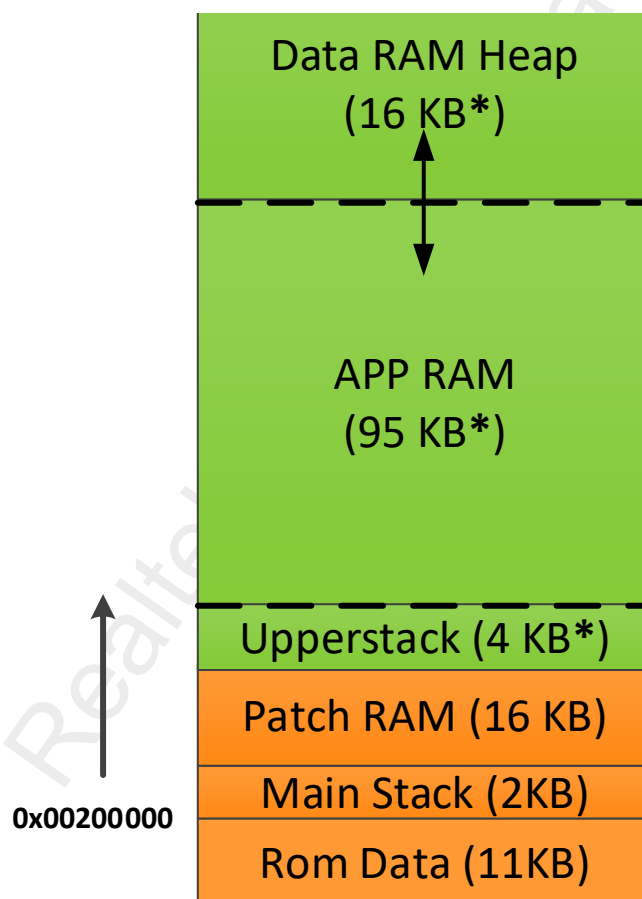
Realtek Confidential

3 RAM

RTL8762D 的 RAM 空间分为两部分，地址空间为[0x00200000, 0x00224000) 的 Data RAM 和地址空间为[0x00280000, 0x00288000)的 Buffer RAM。两块 RAM 空间都可以用来存储数据和执行代码。目前，RTL8762D SDK 中默认支持在 Data RAM 上存储数据和执行代码，而 Buffer RAM 留给 APP 可用的是默认用作 heap。

3.1 Data RAM

在 RTL8762D SDK 中，Data RAM 按其作用默认分为 6 部分空间，如图 3-1 所示。



* 表示大小可调整

图 3-1 Data RAM 布局

每个部分都有固定的排布顺序和特定的用途，如表 3-1 所示。

表 3-1 Data RAM 用途

内存类型	内存用途	内存大小 (大小是否可变)
ROM data	存放 ROM Code 中全局变量和静态变量	否
Main Stack	启动和中断服务例程中使用该 stack	否
Patch RAM	存放 Patch 全局变量和静态变量，以及在 RAM 上执行的 code	否
Upperstack RAM	存放 Upperstack 全局变量和静态变量，以及在 RAM 上执行的 code	是
APP RAM	存放 APP 中全局变量和静态变量，以及在 RAM 上执行的 code	是
Data RAM Heap	供 ROM code、Patch code 和 APP code 动态申请空间，其中总大小 16K 中的 10K 空间已经被 rom code 使用，所以目前剩余 6K 空间给 APP 用户使用。	是

其中 Upperstack RAM 、 APP RAM 和 Data RAM Heap 区域的大小可变，但是总大小固定为 115KB。Upperstack RAM 、 APP RAM 和 data RAM heap 三块空间大小可以通过调整 mem_config.h 中的宏 UPPERSTACK_GLOBAL_SIZE 、 APP_GLOBAL_SIZE 的大小修改，如图 3-2 所示。**UPPERSTACK_GLOBAL_SIZE 必须根据使用的 upperstack image 的版本对应设定。**不同的 upperstack 占用不同大小的 Data RAM，请注意看 upperstack 发布文件及其说明。根据应用需要调整 APP_GLOBAL_SIZE 的大小，最后剩余的是 DATA Heap 的大小。值得注意的是，DATA Heap 是整个系统共用的，包括 ROM、Patch 和 APP，初步统计系统默认使用了 10KB 的 data heap，因此需要保证 HEAP_DATA_ON_SIZE 的值不少于 10KB。

```

/*=====
*.....data ram layout configuration
*=====*/
/* Data RAM layout:.....144K
example:
...1) reserved for rom:.....13K (fixed)
...2) patch ram code:.....16K (fixed)
...3) upperstack global + ram code:.....4K (adjustable, config UPPERSTACK_GLOBAL_SIZE, must consponding to used upperstack image)
...4) app global + ram code:.....95K (adjustable, config APP_GLOBAL_SIZE)
...5) Heap ON:.....16K (adjustable, defined by UPPERSTACK_GLOBAL_SIZE and APP_GLOBAL_SIZE)
*/

/** @brief data ram size for upperstack global variables and code */
#define UPPERSTACK_GLOBAL_SIZE .....(4 * 1024)

/** @brief data ram size for app global variables and code, could be changed */
#define APP_GLOBAL_SIZE .....(95 * 1024)

/** @brief data ram size for heap, could be changed,
but (UPPERSTACK_GLOBAL_SIZE + APP_GLOBAL_SIZE + HEAP_DATA_ON_SIZE) must be 115k, and can't less than 10KB */
#define HEAP_DATA_ON_SIZE .....(115 * 1024 - APP_GLOBAL_SIZE - UPPERSTACK_GLOBAL_SIZE)

/** @brief shared cache ram size (adjustable, config SHARE_CACHE_RAM_SIZE: 0/8KB/16KB) */
#define SHARE_CACHE_RAM_SIZE .....(8 * 1024)
/*****

```

图 3-2 调整 Data RAM 布局

3.2 Buffer RAM

Buffer RAM 的地址空间为[0x00280000, 0x00288000)，总大小 32KB，其布局如图 3-3 所示。前 2KB

已经用于 ROM Global Data, 剩余 30KB 用作 Heap, 其中 ROM 中大概使用了 26KB 的动态空间, 剩余 4KB 给 APP 用, 如表 3-2 所示。

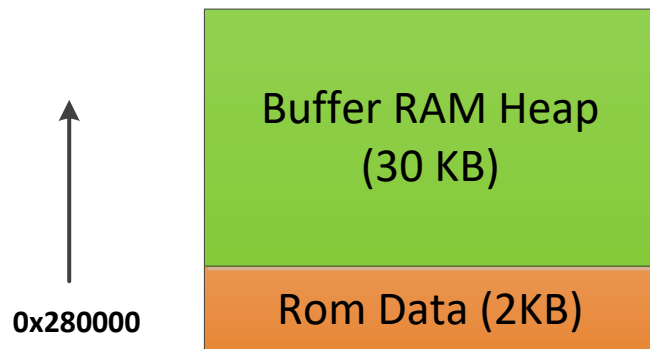


图 3-3 Buffer RAM 布局

表 3-2 Buffer RAM 用途

内存类型	内存用途	内存大小 (是否可变)
ROM data	存放 ROM Code 中全局变量和静态变量	否
Buffer RAM Heap	供 ROM code、Patch code 和 APP code 动态申请空间, 其中总大小 30K 中的 26K 空间已经被 rom code 使用, 所以目前剩余 4K 空间给 APP 用户使用。	否

3.3 APIs

通过调用 API: `os_mem_alloc` 可从 Data RAM Heap 或者 Buffer RAM heap 上动态申请内存, 由参数 `ramType` 指定申请的类型, 如表 3-3 所示。

表 3-3 `os_mem_alloc`

```
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__)

```

其他动态申请内存的 APIs 如下（详细信息参考 os_mem.h）：

- (1) os_mem_zalloc: 申请的内存被初始化为 0；
- (2) os_mem_aligned_alloc: 申请的内存按照指定的方式对齐；
- (3) os_mem_free: 释放从 data ram heap 或者 buffer ram heap 上申请到的内存；
- (4) os_mem_aligned_free: 释放按指定方式对齐申请的内存；
- (5) os_mem_peek: 统计指定 RAM 类型的未使用的内存大小。

3.4 统计 RAM 使用情况

目前 RTL8762D 的 RAM 空间是两块不连续的空间，为了 APP 使用 RAM 更方便，有对内存管理做以下优化：调用 os_mem_alloc 等 API 不管参数 ram_type 指定为 RAM_TYPE_DATA_ON 还是 RAM_TYPE_BUFFER_ON，都会优先从 Buffer RAM 申请，如果申请不到再从 Data RAM 申请，从而让静态数据和代码集中放到 Data RAM，而不是分散到 Data RAM 和 Buffer RAM。

3.4.1 统计 Data RAM 使用的静态区大小

查找 Build 生成的 app.map 文件来找到 Data RAM 中分配的首地址，结束地址以及使用的大小。如下图，通过查找加载区“RAM_DATA_ON”的 map 图，可以确定 APP DATA RAM 的首地址。Execution Region OVERLAY_C 决定着 Data RAM 的结束地址。如下示例中，Data RAM 实际使用大小为 0xd10，而在“mem_config.h”中配置的 APP_GLOBAL_SIZE 的值是 0x16c00（91KB）。通过 map 图可以进一步优化 APP 配置 Data RAM 的布局。

```

..Execution Region: RAM DATA ON (Exec-base: 0x00208400, Load-base: 0x0082fec0, Size: 0x00000d10, Max: 0x00016c00, OVERLAY)
DATA RAM的起始地址
..Exec Addr Load Addr Size Type Attr Idx E Section Name Object
实际使用的大小 mem_config.h中配置的 APP_GLOBAL_SIZE的值
0x00208400 0x0082fec0 0x00000150 Code RO 35 app.data_ram.text system_rtl876x.o
0x00208550 0x00830010 0x0000006c Data RW 229 data overlay_mgr.o
0x002085bc 0x0083007c 0x00000022 Data RW 367 data main.o
0x002085de 0x0083009e 0x00000001 Data RW 439 data dfu_flash.o
0x002085df 0x0083009f 0x0000002e Data RW 649 data dis.o
0x0020860d 0x008300cd 0x00000003 PAD
0x00208610 0x008300d0 0x000000a8 Data RW 702 data dfu_service.o
0x002086b8 0x008300d0 0x00000010 Zero RW 40 system_rtl876x.o
0x002086c8 0x008300d0 0x00000008 Zero RW 227 bss overlay_mgr.o
0x002086d0 0x008300d0 0x0000000c Zero RW 249 bss app_task.o
0x002086dc 0x008300d0 0x00000009 Zero RW 311 bss silent_ota_application.o
0x002086e5 0x008300d0 0x00000006 Zero RW 365 bss main.o
0x002086eb 0x008300d0 0x00000003 Zero RW 530 bss dfu_application.o
0x002086ee 0x00830178 0x00000002 PAD
0x002086f0 0x00830178 0x00000021 Zero RW 561 bss dfu_main.o
0x00208711 0x00830178 0x00000003 PAD
0x00208714 0x00830178 0x0000000c Zero RW 601 bss dfu_task.o
0x00208720 0x00830178 0x00000008 Zero RW 627 bss bas.o
0x00208728 0x00830178 0x00000082 Zero RW 647 bss dis.o
0x002087aa 0x00830178 0x00000002 PAD
0x002087ac 0x00830178 0x00000038 Zero RW 671 bss ota_service.o
0x002087e4 0x00830178 0x00000834 Zero RW 700 bss dfu_service.o
0x00209018 0x00830178 0x000000e4 Zero RW 1505 bss c_w.l(rand.o)
0x002090fc 0x00830178 0x00000014 Zero RW 1599 bss c_w.l(rt_locale.o)

```

```

..Execution Region: OVERLAY_A (Exec-base: 0x00209110, Load-base: 0x00830178, Size: 0x00000000, Max: 0xffffffff, OVERLAY)
****No section assigned to this execution region****

..Execution Region: OVERLAY_B (Exec-base: 0x00209110, Load-base: 0x00830178, Size: 0x00000000, Max: 0xffffffff, OVERLAY)
****No section assigned to this execution region****

..Execution Region: OVERLAY_C (Exec-base: 0x00209110, Load-base: 0x00830178, Size: 0x00000000, Max: 0xffffffff, OVERLAY)
****No section assigned to this execution region****

```

3.4.2 统计 Cache Shared RAM 使用的静态区大小

通过 SHARE_CACHE_RAM_SECTION 修饰可以指定函数或者静态、全局变量到 SHARE CACHE RAM 上。通过查找 Build 生成的 app.map 文件来找到 Cache Shared RAM 中分配的首地址（固定是 0x00224000）和结束地址，如下示例中，SHARE CACHE RAM 实际使用大小为 0x158，而在“mem_config.h”中配置的 SHARE_CACHE_RAM_SIZE 的值是 16KB，通过 map 图可以进一步优化 APP 配置 Data RAM 的布局。

```

..Execution Region: CACHE_DATA_ON (Exec-base: 0x00224000, Load-base: 0x0083001c, Size: 0x00000158, Max: 0x00004000, OVERLAY)
..Exec Addr Load Addr Size Type Attr Idx E Section Name Object
0x00224000 0x0083001c 0x00000154 Code RO 363 app.sharecache_ram.text main.o
0x00224154 0x00830170 0x00000004 Data RW 366 app.sharecache_ram.text main.o

```

3.4.3 统计动态区剩余大小

通过 os_mem_peek 函数可以获取指定 RAM 类型的剩余 heap size。

3.5 调整 APP 可使用 RAM 的总大小

在默认 config（最大链路数目为 1，master 为 0，slave 为 1）配置下，对于 APP 开发可使用的 RAM 总大小为 95（Data RAM Global）+ 6（Data RAM Heap）+ 4（Buffer RAM Heap）+ 16（Cache share ram）= 121 KB。如果想进一步增加 APP 可使用的 RAM 的总大小，可通过以下两种方式：

- 1) 设置蓝牙支持的最大链路数目为 1，同时关闭 AE 和 PSD 功能：通过 MPTool 中 Config Set 选项中的“stack”页面修改如图 3-4 配置可增加 3KB Data RAM Heap 和 1KB Buffer RAM Heap。因此，APP 开发可使用的 RAM 总大小增大到 95（Data RAM Global）+ 9（Data RAM Heap）+ 5（Buffer RAM Heap）+ 16（Cache share ram）= 125KB。

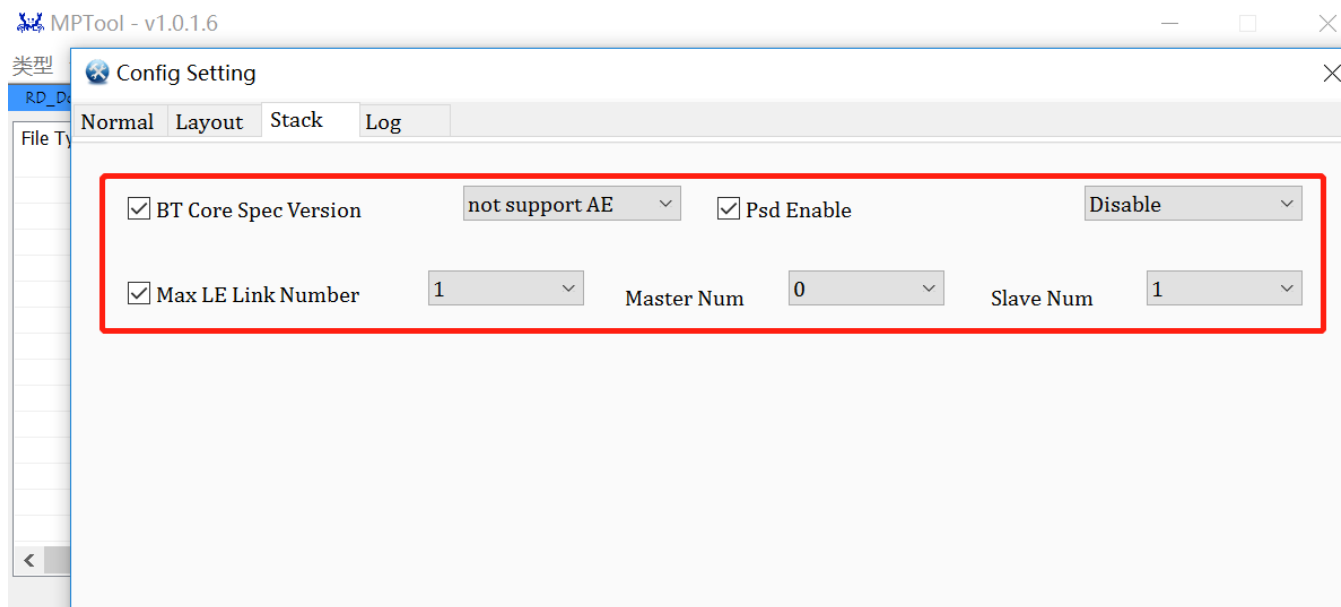


图 3-4 修改 config 参数以增大 APP 可用 RAM 的总大小

- 2) 关闭 log：在 APP 工程头文件 platform_autoconf.h 中打开宏“RELEASE_VERSION”将会关闭 log 的打印，还可以额外增加 3KB 的 Buffer RAM Heap。

4 Cache

RTL8762D 有一个 16K 大小的 cache，对应地址空间为[0x00224000, 0x00228000)，配合 SPI Flash 控制器（SPIC）来加速 flash 的读写访问。同时，该部分空间也可以配置成 Data RAM 使用，用于存储数据和执行代码。用户可通过 mem_config.h 中的宏 SHARE_CACHE_RAM_SIZE 来控制 cache 配置成 Data RAM 的大小，如表 4-1 所示。

表 4-1 配置 Cache 用途

SHARE_CACHE_RAM_SIZE	Flash Cache 大小	Data RAM 大小	适用场景
0 KB	16 KB	0 KB	跑大量的 flash code
8 KB	8 KB	8 KB	跑部分 Flash code，且对 flash code 运行效率有要求
16 KB	0 KB	16 KB	基本不需要运行 flash code，或对 flash code 的运行效率无要求。

5 External Flash

RTL8762D 支持外挂的 SPI Flash 并且集成了 SPI Flash 控制器 (SPIC)。Realtek 提供底层的 Flash driver 接口和 FTL 接口给用户。SPIC 支持最大的地址映射空间为 32M bytes，并且有两组地址空间，对应 cache 地址空间 [0x00800000, 0x02800000)和 non-cache 地址空间[0x04800000, 0x06800000)。当使能 cache 时，CPU 访问 cache 地址空间，将提升数据读取效率和代码执行效率。

5.1 Flash 布局

RTL8762D SDK 中，flash layout 是由 7 部分组成，包括“Reserved”，“OEM Header”，“OTA Bank 0”，“OTA Bank 1”，“FTL”，“OTA Tmp”和“APP Defined Section”，如图 5-1 所示。

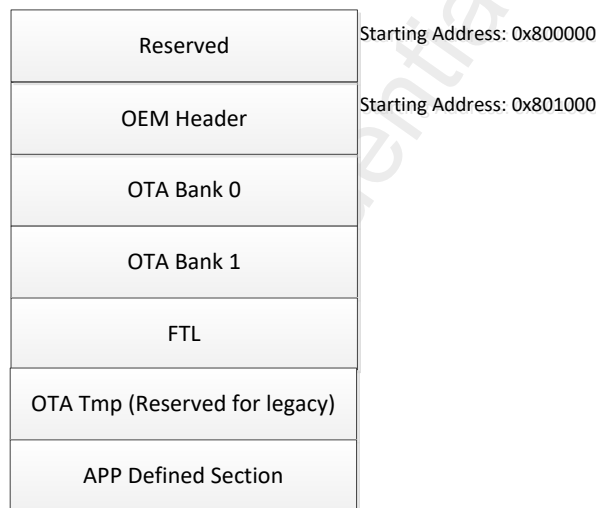


图 5-1 Flash 布局

Flash 布局中各个部分功能介绍如表 5-1 所示。

表 5-1 Flash 段

内存段	起始地址	大小 (字节)	作用
Reserved	0x800000	0x1000	系统保留区域
OEM Header	0x801000	0x1000	config 信息存储区域，包括蓝牙地址，AES Key，和客户可修改的 Flash 布局等
OTA Bank 0	可变的 (由 OEM Header 决定)	可变的 (由 OEM Header 决定)	存储数据和代码运行区，进一步细分为 OTA Header , Secure boot, Patch, APP, APP Data1, APP Data2, APP Data3, APP Data4, APP Data5, APP Data6。

			如果选择不切换 bank 的 OTA 方案，OTA_TMP 区用作 Ota 备份区；相反，如果选择切换 bank 的 OTA 方案，OTA bank0 和 OTA bank1 作用相同，互为备份区。
OTA Bank 1	可变的 (由 OEM Header 决定)	可变的 (由 OEM Header 决定)	选择切换 bank 的 OTA 方案时才存在，假设 Ota Bank0 为运行区，则 Ota Bank1 为备份区；相反，如果 Ota Bank1 为运行区，则 Ota Bank0 为备份区。其大小必须和 OTA Bank 0 一致。
FTL	可变的 (由 OEM Header 决定)	可变的 (由 OEM Header 决定)	该区域空间支持以逻辑地址访问 flash，用户可以最小以 4bytes 为单位读写 flash。
OTA TMP	可变的 (由 OEM Header 决定)	可变的 (由 OEM Header 决定)	不切换 bank 方式下，作为 OTA 备份区使用，大小必须不小于 OTA bank0 最大的 image 大小。
APP Defined Section	可变的 (由 OEM Header 决定)	可变的 (由 OEM Header 决定)	Flash 剩余未划分区域，客户可以自由使用。但不可以进行 OTA 升级。

OTA Bank 中包括 10 种类型的 image，分别是 OTA Header，Secure boot，Patch，APP，APPData1，APP Data2，APP Data3，APP Data4，APP Data5 和 APP Data6，其中各个部分的说明如图 5-2 所示，布局中各部分描述说明如表 5-2 所示。OTA Bank 这一级的布局是由其中的 OTA header 决定，通过 MPPack Tool 可以生成不同 OTA Bank 布局的 OTA header。

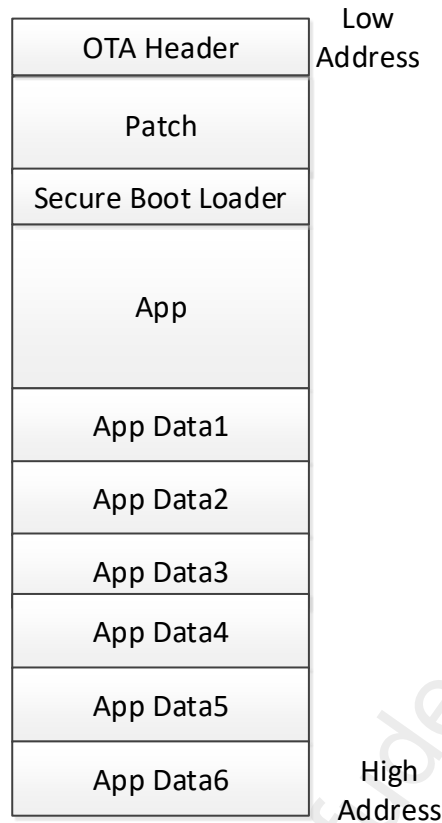


图 5-2 OTA Bank 布局

表 5-2 OTA Bank 各 image 说明

内存段	起始地址	大小	作用
OTA Header	由 OEM header 决定	4KB	存储 OTA Header 的版本信息、Bank 中各 image 的起始地址和大小。
Secure Boot Loader	由 OTA header 决定	可变的	启动过程中对代码安全级别检查的代码。
Patch	由 OTA header 决定	可变的	对 rom 中蓝牙协议栈，系统的优化和扩展代码。
App	由 OTA header 决定	可变的	开发方案的运行代码
App Data1	由 OTA header 决定	可变的	开发方案中需要升级的数据区。
App Data2	由 OTA header 决定	可变的	开发方案中需要升级的数据区。
App Data3	由 OTA header 决定	可变的	开发方案中需要升级的数据区。
App Data4	由 OTA header 决定	可变的	开发方案中需要升级的数据区。
App Data5	由 OTA header 决定	可变的	开发方案中需要升级的数据区。
App Data6	由 OTA header 决定	可变的	开发方案中需要升级的数据区。

RTL8762D 支持根据不同的应用场景灵活配置 Flash 布局，用户可通过 MPtool 中“config set”选项配置自定义的 flash 布局。为了方便用户使用，Realtek 还提供了 FlashMapGenerateTool 用于生成 flash_map.ini

和 flash_map.h 文件。flash_map.ini 文件可以导入到 MPTool 和 MPPack Tool，分别用于生成 config file 和 OTA header。而 flash_map.h 文件则需要拷贝到 APP 的工程目录下，编译出正确可运行的 APP image。

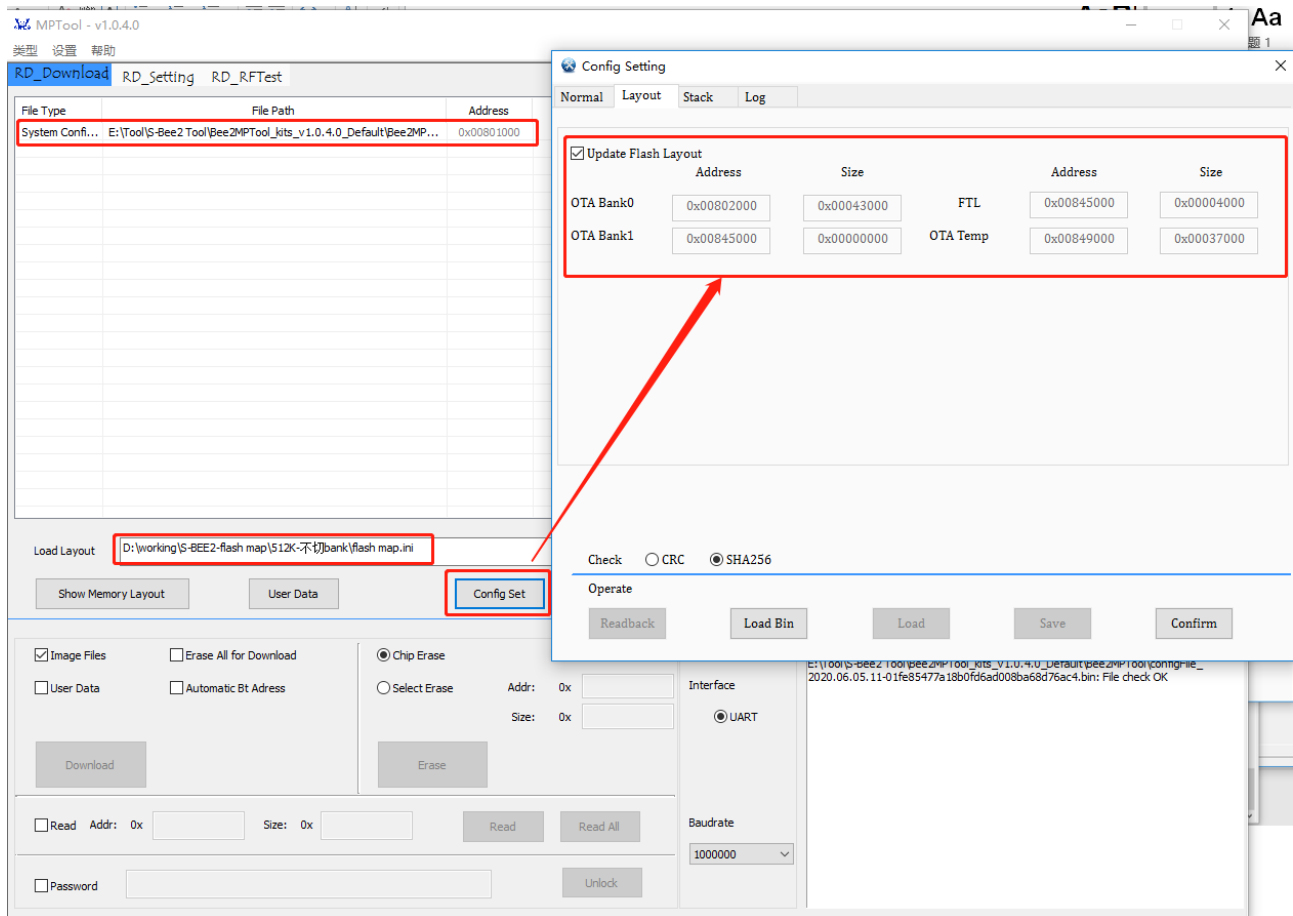


图 5-3 通过 MPTool 生成 config 文件

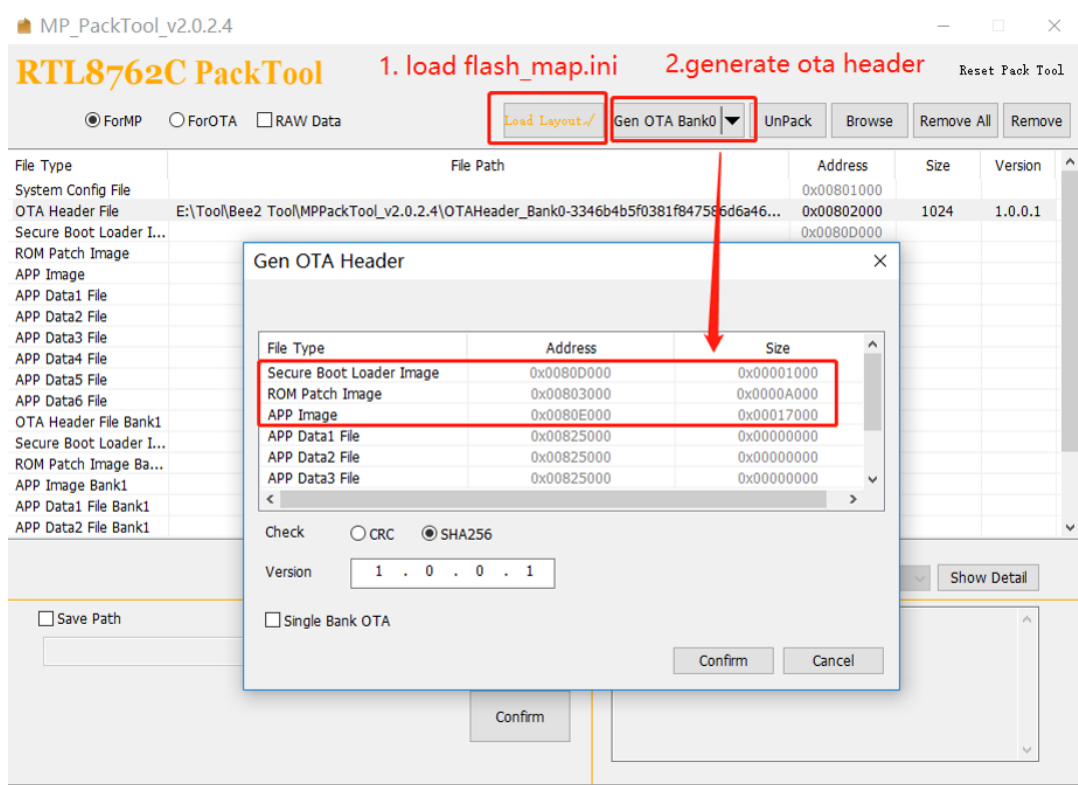


图 5-4 通过 MPackTool 生成 ota header 文件

但与此同时，必须遵从以下的原则来调整 flash 布局。

- (1) 如果选择支持 bank 切换的 OTA 方案，保证 OTA bank0 的大小等于 OTA bank1 的大小，并且将 OTA tmp 区域的大小设置为 0。
- (2) 如果选择支持 bank 切换的 OTA 方案，可以使用不同的 OTA bank 布局来生成 OTA header0 和 OTA header1，但是 OTA bank0 和 OTA bank1 的大小不可变，并且不小于 OTA bank 内各个 image 大小的总和。
- (3) 如果选择不支持 bank 切换的 OTA 方案，将 OTA bank1 的大小设置为 0，OTA tmp 区域的大小不小于 OTA bank0 中最大的 image 的大小。
- (4) 如果选择不支持 bank 切换的 OTA 方案，除 OTA Header、patch、app 外，还必须烧录 secure boot image，secure boot image 的烧录地址固定为 0x80D0000，大小至少为 4KB。secure boot image 是由 Realtek 发布。
- (5) 原始的 APP Data bin 文件需要经过 RTL8762D SDK 中提供的 App Data tool 处理之后才可以经过 MPTool 烧录和 MPackTool 打包。该工具路径在 sdk/tool/AppData，处理之后会添加一个满足特定格式的 1KB 的 image header。注意，OTA bank 内的 APP Data1、APP Data2、APP Data3、APP Data4、APP Data5 或 APP Data6 区域是存储需要进行 OTA 的 App Data，如果数据无需升级，应当放在“APP Defined Section”区域。

(6) 尽可能地将 OTA bank1 结束地址相对于 flash 起始地址（0x800000）的偏移对齐到实际所选用的 flash 所支持的某个保护级别对应的范围，如 64KB、128KB、256KB、512KB 等。目的是将所有的 code 区域都上锁，以防止非预期的 flash 写入和擦除操作。

(7) 因为存放在 OTA bank 内的 APP Data1、APP Data2、APP Data3、APP Data4、APP Data5 或 APP Data6 区域的数据是有可能落在 flash 上锁范围内的，所以该区域内一般是存储只读的数据。如果需要存储需要改写的的数据应该放在“APP Defined Section”区域。

针对上述原则中的第（6）点，以下简述其原因。RTL8762D 支持一种名为 Flash 软件块保护的机制来锁定 Flash，以防止非预期的写入和擦除操作。Flash 软件块保护机制的原理是根据 Flash 状态寄存器中的一些 BP 位来选择级别（范围）来保护，如图 5-3 所示。

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 – 1FFFFFFh	64KB	Upper 1/32
0	0	0	1	0	30 and 31	1E0000h – 1FFFFFFh	128KB	Upper 1/16
0	0	0	1	1	28 thru 31	1C0000h – 1FFFFFFh	256KB	Upper 1/8
0	0	1	0	0	24 thru 31	180000h – 1FFFFFFh	512KB	Upper 1/4
0	0	1	0	1	16 thru 31	100000h – 1FFFFFFh	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 – 0FFFFFFh	1MB	Lower 1/2
X	X	1	1	X	0 thru 31	000000h – 1FFFFFFh	2MB	ALL

图 5-5 flash 软件块保护

Flash 使用状态寄存器中的 BP(X)位来识别要锁定的块的数量，以及 TB 位来决定锁定的方向。但是，RTL8762D 的 flash 布局是从地址开始存放配置的数据和执行代码，因此 RTL8762D 上只支持了从低地址锁定 Flash。为了更好地支持这一特性，Realtek 同步发布一个合格的适配 RTL8762D Flash 的供应商列表（AVL），以保证外接的 flash 支持软件块保护的特性。AVL 中的大多数 flash 都支持按级别保护 Flash，如包含前 64KB、128KB、256KB、512KB 等不同大小的区域。

因此，当划分 Flash 布局时，需要确保尽可能地将 OTA bank1 结束地址偏移对齐到实际所选用的 flash 所支持的某个保护级别对应的范围。这样做的目的是，将重要数据区和代码区完全上锁，且不影响紧随其后的存放需要改写的的数据区，如“FTL”、“OTA Tmp”区。一旦 Flash 布局确定之后，RTL8762D 将自动解析 Flash 布局配置参数并查询所选 Flash 信息设置 BP 保护级别。

被保护的区域无法直接执行写和擦的操作。如果需要，先做解锁 flash，再执行写或者擦除访问，最后将 flash 锁回原来的级别。但是，这样的操作并不推荐，因为 flash 的状态寄存器是 NVRAM 方式访问时，

具有 100K 的编程次数限制，频繁解锁将有可能导致 flash 不可用。

为了最大限度地利用 BP，表 5-3，5-4,5-5 提供了一些 Flash 布局示例。但是必须注意的是，AVL 中有两种特殊的 Flash，它们不支持通过按级别来保护 Flash，更多关于 flash BP 的详情请参阅 Realtek 提供的 RTL8762D 所支持的 AVL。

表 5-3 flash 总大小为 512KB 的示例布局

flash 总大小为 512KB 的示例布局	大小（字节）	起始地址	BP 保护的大小
1) Reserved	4K	0x800000	从 flash 低地址开始的前 256KB（OTA Bank1 结束地址的偏移为 276KB，因此在这种布局下依然有部分 OTA Bank0 区域未被上锁）
2) OEM Header	4K	0x801000	
3) OTA Bank0	268K	0x802000	
a) OTA Header	4K	0x802000	
b) Secure boot loader	4K	0x80D000	
c) Patch code	40K	0x803000	
d) APP code	220K	0x80E000	
e) APP data1	0K	0x00845000	
f) APP data2	0K	0x00845000	
g) APP data3	0K	0x00845000	
h) APP data4	0K	0x00845000	
i) APP data5	0K	0x00845000	
j) APP data6	0K	0x00845000	
4) OTA Bank1	0K	0x00845000	未上锁区域
5) FTL	16K	0x00845000	
6) OTA Temp	220K	0x00849000	
7) APP Defined Section	0K	0x00880000	

表 5-4 flash 总大小为 1MB 的示例布局

flash 总大小为 1MB 的示例布局	大小（字节）	起始地址	BP 保护的大小
1) Reserved	4K	0x00800000	从 flash 低地址开始的前 512KB（OTA Bank1 结束地址的偏移对齐在 808KB，一部分的代码未加锁）
2) OEM Header	4K	0x00801000	
3) OTA Bank0	400K	0x00802000	
a) OTA Header	4K	0x00802000	
b) Secure boot loader	4K	0x0080D000	
c) Patch code	40K	0x00803000	

d) APP code	352K	0x0080E000	
e) APP data1	0K	0x00866000	
f) APP data2	0K	0x00866000	
g) APP data3	0K	0x00866000	
h) APP data4	0K	0x00866000	
i) APP data5	0K	0x00866000	
j) APP data6	0K	0x00866000	
4) OTA Bank1 (大小必须和 OTA Bank0 相同)	400K	0x00866000	
a) OTA Header	4K	0x00866000	
b) Secure boot loader	0K	0x00871000	
c) Patch code	40K	0x00867000	
d) APP code	352K	0x00872000	
e) APP data1	0K	0x008CA000	
f) APP data2	0K	0x008CA000	
g) APP data3	0K	0x008CA000	
h) APP data4	0K	0x008CA000	
i) APP data5	0K	0x008CA000	
j) APP data6	0K	0x008CA000	
5) FTL	16K	0x008CA000	未上锁区域
6) OTA Temp	0K	0x008CE000	
7) APP Defined Section	200K	0x008CE000	

表 5-5 flash 总大小为 2MB 的示例布局

flash 总大小为 1MB 的示例布局	大小 (字节)	起始地址	BP 保护的 大小
1) Reserved	4K	0x00800000	从 flash 低地址开始的前 1MB (OTA Bank1 结束地址的偏移对齐在 1MB)
2) OEM Header	4K	0x00801000	
3) OTA Bank0	508K	0x00802000	
a) OTA Header	4K	0x00802000	
b) Secure boot loader	0K	0x0080D000	
c) Patch code	40K	0x00803000	
d) APP code	464K	0x0080E000	
e) APP data1	0K	0x00881000	

f) APP data2	0K	0x00881000	
g) APP data3	0K	0x00881000	
h) APP data4	0K	0x00881000	
i) APP data5	0K	0x00881000	
j) APP data6	0K	0x00881000	
4) OTA Bank1 (大小必须和 OTA Bank0 相同)	508K	0x00881000	
a) OTA Header	4K	0x00881000	未上锁区域
b) Secure boot loader	0K	0x0088C000	
c) Patch code	40K	0x00882000	
d) APP code	464K	0x0088D000	
e) APP data1	0K	0x00900000	
f) APP data2	0K	0x00900000	
g) APP data3	0K	0x00900000	
h) APP data4	0K	0x00900000	
i) APP data5	0K	0x00900000	
j) APP data6	0K	0x00900000	
5) FTL	16K	0x00900000	
6) OTA Temp	0K	0x00904000	
7) APP Defined Section	200K	0x00904000	

5.2 Flash APIs

操作 flash 的 APIs 如下，其中带有“auto”字样的表示以 auto mode 访问 flash，其他则是通过 user mode 访问 flash。详细信息可以参考 SBee2-SDK.chm 和 RTL8762D Flash User Guide。

1. 基本操作

```
bool flash_auto_read_locked(uint32_t addr, uint32_t *data);
```

```
bool flash_read_locked(uint32_t start_addr, uint32_t data_len, uint8_t *data);
```

```
bool flash_auto_write_locked(uint32_t start_addr, uint32_t data);
```

```
bool flash_auto_write_buffer_locked(uint32_t start_addr, uint32_t *data, uint32_t len);
```

```
bool flash_write_locked(uint32_t start_addr, uint32_t data_len, uint8_t *data);
```

```
bool flash_erase_locked(T_ERASE_TYPE type, uint32_t addr);
```

2. Flash 高速读取的 APIs

```
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);
```

```
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);
```

```
bool flash_split_read_locked(uint32_t start_addr, uint32_t data_len, uint8_t *data, uint32_t *counter);
```

如果需要使用以上三个高速读取 flash 的接口函数，需要将 sdk\src\flash\flash_hs_read.c 文件添加到工程中，并包含头文件“flash_device.h”。

5.3 FTL

FTL 是提供给 bt stack 和 APP 用户读写 flash 上数据的抽象层，通过 FTL 接口可以通过逻辑地址直接读写 flash 上分配出 FTL 空间的数据，操作更为简便。

通常 FTL 空间更适用于存储需要频繁改写的的数据，一方面使用 FTL 接口更为简单，另一方面 FTL 操作实际上会在分配的物理空间上循环使用从而不会对某一个 flash sector 擦写多次而影响 flash 的寿命。如果需要存储数据量比较大并且只是偶尔需要改写，或者是存储只读数据但不需要升级，可以在“APP Defined Section”区域调用 flash APIs 来进行管理。如果需要存储只读数据并且需要升级，放在 OTA bank 内的 App Data1、App Data2、App Data3、App Data4、App Data5、App Data6 区域。

5.3.1 FTL 空间的功能划分

目前，FTL 空间按照其功能划分为以下两块存储空间，以默认设置的 FTL 的物理空间大小为 16K 为例。

(1) BT 存储空间

- (1) 逻辑地址范围[0x0000, 0x0C00)，但是这部分空间的大小可通过 otp 参数调整；
- (2) 用于存储 BT 相关的信息，如蓝牙设备地址、配对密钥等；
- (3) 更多的详细信息可以参考 RTL8762D BLE Stack User Manual。

(2) APP 存储空间

- (1) 逻辑地址范围[0x0C00, 0x17F0)；
- (2) APP 可以使用这部分空间存储自定义的信息；
- (3) 调用以下两个 API（参考 ftl.h）可以读写这部分空间的数据，更多详细信息参考 SBee2-SDK.chm。注意，在调用 ftl_save 或者 ftl_load 时，底层已经封装了一个 offset，默认是 0x0C00。


```
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 调整 FTL 空间的大小

FTL 的物理空间大小是可配置的，通过修改 config 文件的配置参数来调整。操作步骤如下：

- (1) 首先利用 FlashMapGenerateTool（随着 MPackTool 一起发布）来生成 flash_map.ini 和 flash_map.h 文件。
- (2) 将 flash_map.h 文件拷贝到 APP 工程目录下，例如：\sdk\board\evb\ota，这样 APP 编译的时候可以获取正确的加载地址。
- (3) 将第一步生成的 flash_map.ini 加载到 MPTool 中生成 config 文件用于烧录，如图 5-7 所示将会将 FTL 的物理空间大小调整到 32K。

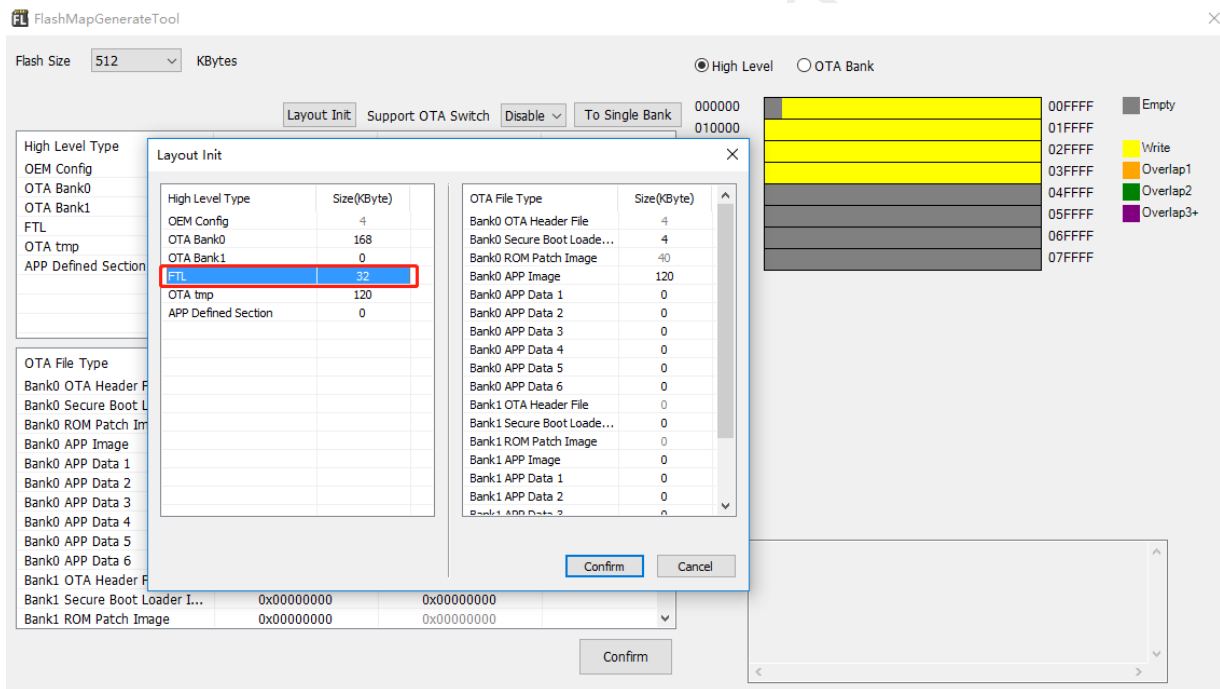


图 5-6 修改 flash map 配置文件

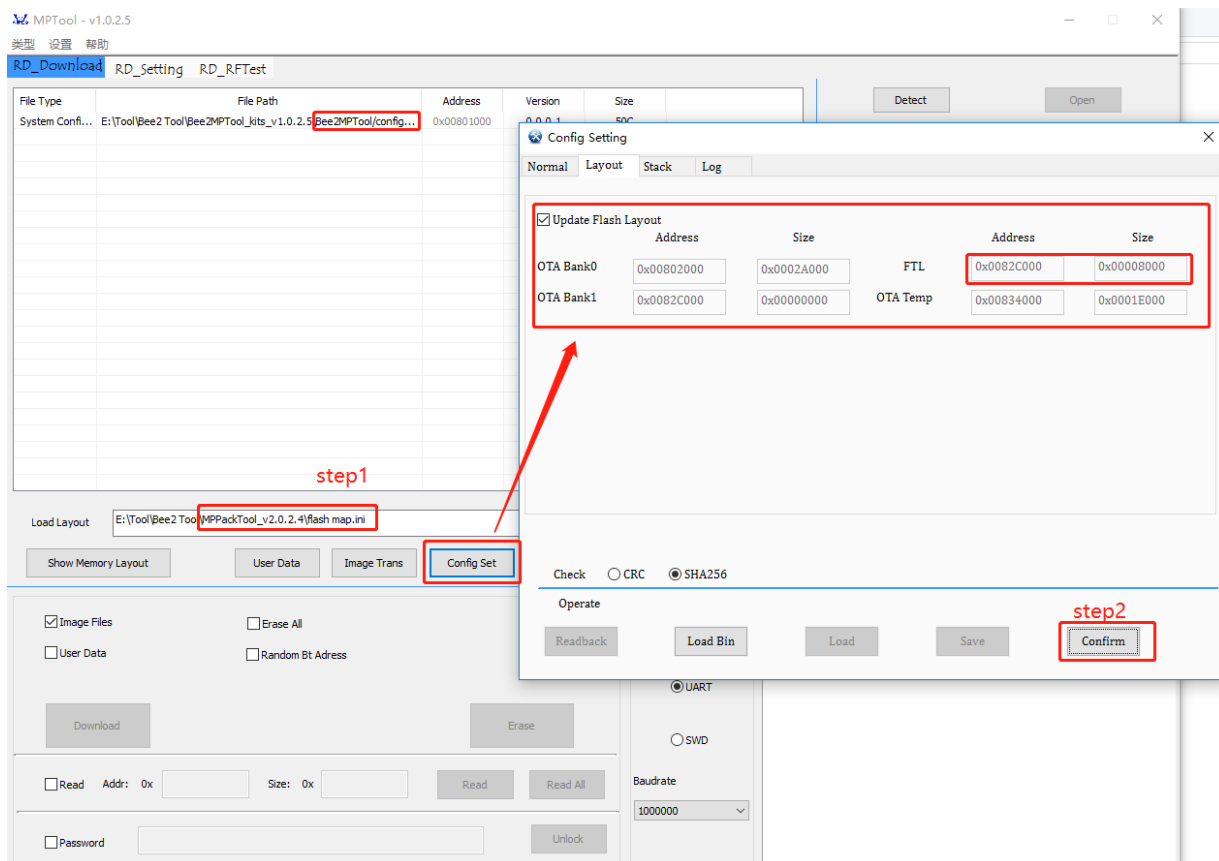


图 5-7 生成 config 文件

当调整 FTL 物理空间大小，APP 可使用的逻辑空间也会对应改变。假设实际 FTL 的物理空间大小设置的 mK（m 是 4 的整数倍），对应 APP 可使用的逻辑空间等于 $((511 * (m - 4) - 4) - 3072)$ bytes。此外，为了提升 FTL 读取的效率，底层做了一套物理地址和逻辑地址的映射机制，这个映射表会占用一定的 RAM 空间。在默认设置 FTL 物理空间大小为 16K 情况下，这个映射表占用 2298byte 的 buffer ram heap 的空间。假设实际 FTL 的物理空间大小设置的 mK（m 是 4 的整数倍），其映射表占用的 RAM 空间等于 $((511 * (m - 4) - 4) * 0.375)$ bytes。所以，用户在调整 FTL 空间大小需要依据具体的应用场景来合理调整，如果调至过大将会浪费一部分 RAM 资源。另一方面，如果对于选用的比较小的 flash 并且想压缩 FTL 的占用的空间，必须保证 FTL 的物理空间不比 12K 小。同时由于映射表中每个逻辑地址只占 12 个 bit 的限制，FTL 的物理空间最大可以调整到 36K。

6 设置代码执行位置

代码可以放在 Flash 上运行，也可以放在 RAM 上运行。本节将介绍如何把代码放到特定的内存位置运行。

1. 修改宏 FEATURE_RAM_CODE 定义：
 - (1) 1 表示默认不加任何 section 的代码是放到 RAM 运行；
 - (2) 0 表示默认不加任何 section 的代码是放到 Flash 运行。
2. 如果想要特别指定某个函数是放到特定的内存位置上，使用 app_section.h 中的 section 宏定义。
例如：
 - (1) APP_FLASH_TEXT_SECTION 表示把函数放到 Flash 运行；
 - (2) DATA_RAM_FUNCTION 表示把函数放到 RAM 运行。
 - (3) SHARE_CACHE_RAM_SECTION 表示把函数和全局、静态变量放到 SHARE CACHE RAM 上运行。

7 eFuse

eFuse 是一个一次性存储程序块，用于存储重要的和固定的信息，如 UUID、安全密钥和其他一次性编程配置。eFuse 的单个位不能从 0 变为 1，并且没有擦除操作，所以要小心更新 eFuse。RealTek 提供 MP Tool 工具来更新某些 eFuse 部分。

Realtek Confidential

Reference

1. RTL8762D BLE Stack User Manual.
2. SBee2-SDK.chm
3. RTL8762D Flash User Guide.

Realtek Confidential