AIC8800-SDK OTA手册

Date	Version	Notes
2023-06-16	v1.1.00	初始版本
2023-07-07	v1.1.01	更新App编译控制宏
2023-07-07	v1.1.02	增加兼容版本说明
2023-08-11	v1.1.03	增加压缩库安装说明
2023-08-18	v1.1.04	增加删除备份分区说明
2023-10-18	v1.1.05	更新删除备份分区说明
2023-10-18	v1.1.06	增加升级过程示意图
2024-05-08	v1.1.07	增加WiFi协议栈版本说明

说明： *OTA* 有多种实现方式， 本 SDK 集成了 *http-OTA* 、*host-OTA* 两种方式， *http-OTA* 是指通过 *TCP* 连接，以无线网络的形式更新应用程序，*host-OTA* 是主控通过 *SDIO/USB* 接口升级应用程序

一、新版 bootloader 描述

0 OTA bootloader 运行流程

```
void bootloader_test(void)
{
    /* ... */

    // 1. 检查 cur_image 是否有效
    if (0 == res[IMAGE_AREA_CURRENT]) {
        /* ... */
        // 确认 upg_image 有效，且版本更新，进行升级!
        if ((0 == res[IMAGE_AREA_UPGRADE]) && (image_is_newver(info, header))) {
            UART_PRINT("update img ver: %s -> %s\n", info->version, header-
>version);
            image_copy(header);
        }
        /* ... */
    // 2. 检查 upg_image 是否有效
    } else if (0 == res[IMAGE_AREA_UPGRADE]) {
        // cur_image 已损坏，进行升级!
        struct image_header *header = image_info_hdr_get(IMAGE_AREA_UPGRADE);
        image_copy(header);
        UART_PRINT("cur invalid, use upg: %s\n", header->version);

    // 3. cur_image & upg_image 损毁
    } else {
        // 确认 bak_image
        /* ... */
```

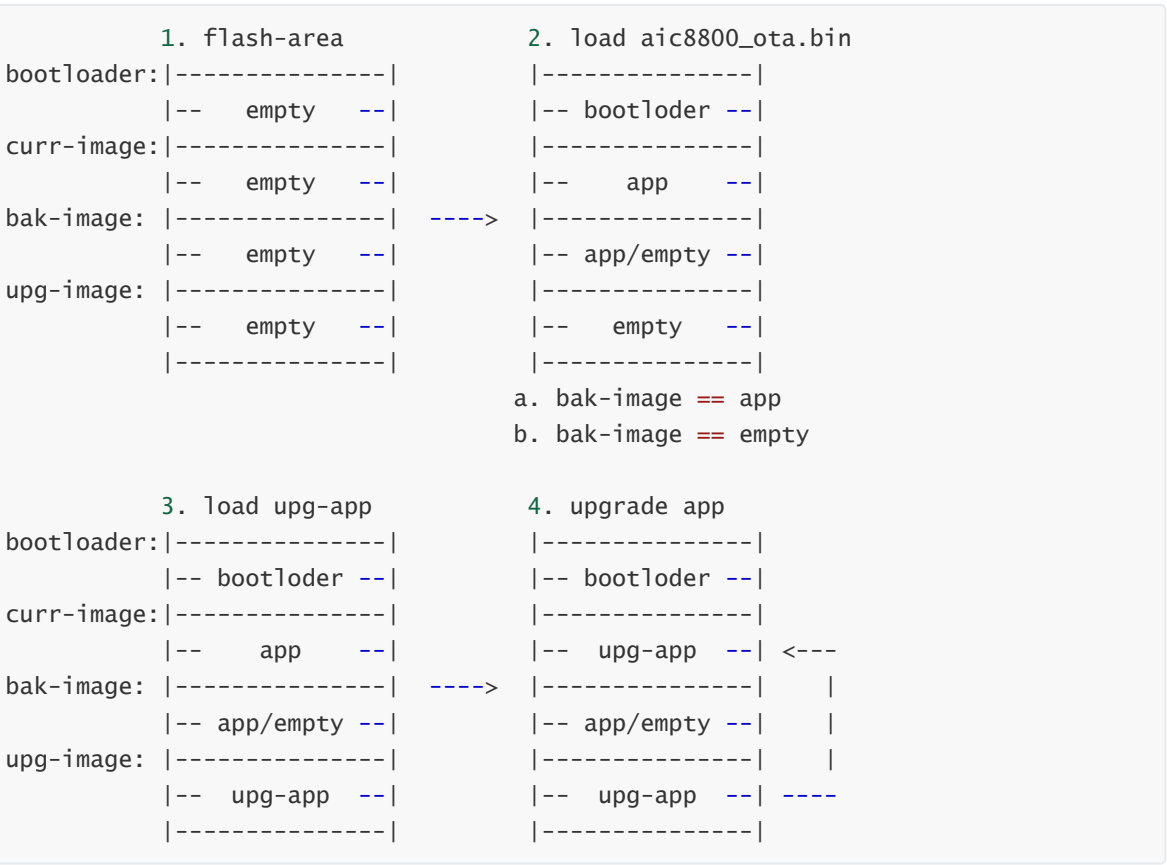
```
    }

    // 4. 确认 cur_image 地址，跳转运行
    /* ... */

    UART_PRINT("\nBootloader test done\n");
}
```

首先 check cur_image 是否有效，check 方法是计算出 cur_image 的 crc 值跟 image_info 中保存的 crc 值作比较。同样方法对 upg_image 区域内容做 check。 其次，比较 cur_image 和 upg_image 的版本，若 upg_image 版本较新且有效则先把 upg_image 内容拷贝到 cur_image 区域，并跳转到 cur_image 执行；否则，若 cur_image 有效直接跳转到执行，否则报错退出。

下图简要描述了升级的基本过程，其中备份分区不使用则为空，用户若需要使用备份分区，则应在初次烧录时将文app.bin存放至备份分区中。若flash空间资源紧张，则可以直接删除备份分区，详见后文描述。



1 OTA 功能描述

目前 OTA 除了常规的升级应用功能外，还可针对 flash 版本的 wifi 协议栈进行升级，**特别说明**，wifi 协议栈升级必须对即将升级的 wifi 协议栈进行确认，方可进行升级，否则在 http-OTA 方式中，可能使得设备无法再进行联网。

注意：

a. wifi协议栈是指AIC8800的底层wifi固件，不是上层lwip应用；

b. 除了默认版本，其他版本为特殊需求，需跟AIC原厂沟通，单独release库文件

芯片型号	wifi 协议栈默认版本	wifi 协议栈支持版本
AIC8800M	ROM	ROM/RAM/FLASH
AIC8800MC	ROM	ROM/FLASH
AIC8800M40B	RAM	RAM

WiFi协议栈版本	描述
ROM	协议栈为固化代码，不占用RAM与FLASH空间
RAM	协议栈占用RAM空间，编译时合并在bin文件中
FLASH	协议栈占用FLASH空间，单独bin文件，需要额外在FLASH中烧录协议栈代码（bin文件）

当用户编译的应用程序较大时，flash 中的空间可能不够分配，可考虑使用压缩升级的方式，压缩升级会将升级分区和备份分区的 bin 文件进行压缩，再写入对应的分区。压缩采用无所压缩的方式，压缩率大致是62.5%~71.4%，依据实际的 bin 文件压缩率会有差异，用户应根据最后编译的应用程序为准，在调整分区时应预留足够的余量。

依据是否支持更新 wifi 协议栈、是否支持压缩升级，目前对 flash 分区设置了**4个 demo**，用户结合自己的实际应用需求，可以在 demo 上进行调整。

分区表位于 AIC8800-SDK/plf/aic8800x/src/driver/flash 目录下，

版本	功能	编译类型	使用编译选项
bootloader	不支持 wifi 协议栈，不支持压缩	bootloader.bin	NULL
		app.bin (host_wb.bin，下同)	OTA=on
bootloader_fw	支持 wifi 协议栈，不支持压缩	bootloader.bin	FLASH_FW=on
		app.bin	OTA=on、FLASH_FW=on
bootloader_lma	不支持 wifi 协议栈，支持压缩	bootloader.bin	ALGO=on
		app.bin	OTA=on、LZMA=on
bootloader_fw_lma	支持 wifi 协议栈，支持压缩	bootloader.bin	ALGO=on、FLASH_FW=on
		app.bin	OTA=on、LZMA=on、FLASH_FW=on

1. 以上描述的编译选项，部分包含在*target*目录下的脚本中，如 *ALGO*

2. ALGO 控制代码中的宏 *CFG_LZMA*，在编译 *bootloader.bin*时使用

3. LZMA 控制代码中的宏 CFG_LZMA_APP，在编译应用程序 app.bin时使用，以下编译描述有详细说明

2 OTA flash 分区

```
/**
 * Memory map with bootloader:
 * +--------------+  <---------------- 0x08000000
 * |              |     ^
 * | bootloader   |     |  12KB/20KB(支持压缩版本)
 * |              |     v
 * +--------------+  <---------------- 0x08003000
                                      (0x08005000)
 ...
 // image_header and image_body //
 ...
 * +--------------+  <---------------- 0x081FC000
 * |              |     ^
 * | bt_ble_info  |     |  4KB
 * |              |     v
 * +--------------+  <---------------- 0x081FD000
 * |              |     ^
 * | wifi_info    |     |  4KB
 * |              |     v
 * +--------------+  <---------------- 0x081FE000
 * |              |     ^
 * | calib_info   |     |  4KB : factory calib(RO)
 * |              |     v
 * +--------------+  <---------------- 0x081FF000
 * |              |     ^
 * | boot_info    |     |  4KB : chip reserved(RO)
 * |              |     v
 * +--------------+  <---------------- 0x08200000
 */
```

2M flash 版本的分区框架如上，主要分三部分，bootloader、image 和 system_info，**特别注意**，不同系列的芯片可能会有细微的差异，但分区的主体框架保持一致。

bootloader: 依据是否支持压缩，区分两个版本

image: 这部分存放镜像 bin 文件及其 header 信息

system_info: 存放 AIC8800x 的出厂配置信息，不允许擦除

image 分区描述

0 概要

一份完整的镜像包括 header 和 body，如当前分区 image_info + cur_image，升级分区 image_header + upg_image；image_info、image_header 保存镜像的基本信息，固定4KB大小，不可调节。

1. bootloader.h

```
/** base version: bootloader.h **/
/**
 * +--------------+  <--------------- 0x08003000
 * |              |     ^
 * | image_info   |     |  4KB
 * |              |     v
 * +--------------+  <--------------- 0x08004000
 * |              |     ^
 * | cur_image    |     |  636KB : current image
 * |              |     v
 * +--------------+  <--------------- 0x080A3000
 * |              |     ^
 * | image_header |     |  4KB
 * |              |     v
 * +--------------+  <--------------- 0x080A4000
 * |              |     ^
 * | bak_image    |     |  636KB : backup image
 * |              |     v
 * +--------------+  <--------------- 0x08143000
 * |              |     ^
 * | image_header |     |  4KB
 * |              |     v
 * +--------------+  <--------------- 0x08144000
 * |              |     ^
 * | upg_image    |     |  636KB : upgrade image
 * |              |     v
 * +--------------+  <--------------- 0x081E3000
 * |              |     ^
 * | user_data    |     |  100KB : user specified
 * |              |     v
 * +--------------+  <--------------- 0x081FC000
 */
```

bootloader.h 将 **image** 划分成4个区域，当前分区、备份分区、更新分区、数据分区，由于不带压缩功能，需要保证三个分区的 image body 大小一致。

可修改分区

1. 数据分区设置100KB，结合实际应用增减
2. 根据用户应用需求，可以选择删除备份分区

2. bootloader_fw.h

```
/** version: bootloader_fw.h **/
/**
 * +--------------+  <--------------- 0x08003000
 * |              |     ^
 * | image_info   |     |  4KB
 * |              |     v
 * +--------------+  <--------------- 0x08004000
 * |              |     ^
 * | cur_image    |     |  572KB : current image
 * |              |     v
 * +--------------+  <--------------- 0x08093000
 * |              |     ^
```

```
 * | image_header |     |  4KB
 * |              |     v
 * +--------------+  <--------------- 0x08094000
 * |              |     ^
 * | bak_image    |     |  572KB : backup image
 * |              |     v
 * +--------------+  <--------------- 0x08123000
 * |              |     ^
 * | image_header |     |  4KB
 * |              |     v
 * +--------------+  <--------------- 0x08124000
 * |              |     ^
 * | upg_image    |     |  572KB : upgrade image
 * |              |     v
 * +--------------+  <--------------- 0x081B3000
 * |              |     ^
 * | user_data    |     |  32KB : user specified
 * |              |     v
 * +--------------+  <--------------- 0x081BB000
 * |              |     ^
 * | fw_header    |     |  4KB
 * |              |     v
 * +--------------+  <--------------- 0x081BC000
 * |              |     ^
 * | wifi_fw      |     |  256KB : fmacfw image
 * |              |     v
 * +--------------+  <--------------- 0x081FC000
 */
```

bootloader_fw.h 支持 flash 版本的 wifi 协议栈，增加了协议栈分区， **特别注意，**协议栈分区不可修改!

可修改分区

1. 数据分区设置32KB，结合实际应用增减
2. 根据用户应用需求，可以选择删除备份分区

3. bootloader_lzma.h

```
/** version: bootloader_lzma.h **/
/**
 * +--------------+  <--------------- 0x08005000
 * |              |     ^
 * | image_info   |     |  4KB
 * |              |     v
 * +--------------+  <--------------- 0x08006000
 * |              |     ^
 * | cur_image    |     |  780KB : current image
 * |              |     v
 * +--------------+  <--------------- 0x080C9000
 * |              |     ^
 * | image_header |     |  4KB
 * |              |     v
 * +--------------+  <--------------- 0x080CA000
 * |              |     ^
 * | bak_image    |     |  560KB : backup image
 * |              |     v
 * +--------------+  <--------------- 0x08156000
```

```
 * |              |     ^
 * | image_header |     |  4KB
 * |              |     v
 * +--------------+  <--------------- 0x08157000
 * |              |     ^
 * | upg_image    |     |  560KB : upgrade image
 * |              |     v
 * +--------------+  <--------------- 0x081E3000
 * |              |     ^
 * | user_data    |     |  100KB : user specified
 * |              |     v
 * +--------------+  <--------------- 0x081FC000
 */
```

bootloader_lzma.h 不支持 flash 版 wifi 协议栈，划分当前分区、备份分区、更新分区、数据分区，支持压缩功能，可以将 bak_image 和 upg_image 压缩存放。

可修改分区

1. 数据分区设置100KB，结合实际应用增减
2. 根据用户应用需求，可以选择删除备份分区

4. bootloader_fw_lzma.h

```
/** version: bootloader_fw_lzma.h **/
/**
 * +--------------+  <--------------- 0x08005000
 * |              |     ^
 * | image_info   |     |  4KB
 * |              |     v
 * +--------------+  <--------------- 0x08006000
 * |              |     ^
 * | cur_image    |     |  712KB : current image
 * |              |     v
 * +--------------+  <--------------- 0x080B8000
 * |              |     ^
 * | image_header |     |  4KB
 * |              |     v
 * +--------------+  <--------------- 0x080B9000
 * |              |     ^
 * | bak_image    |     |  500KB : backup image
 * |              |     v
 * +--------------+  <--------------- 0x08136000
 * |              |     ^
 * | image_header |     |  4KB
 * |              |     v
 * +--------------+  <--------------- 0x08137000
 * |              |     ^
 * | upg_image    |     |  500KB : upgrade image
 * |              |     v
 * +--------------+  <--------------- 0x081B4000
 * |              |     ^
 * | user_data    |     |  28KB : user specified
 * |              |     v
 * +--------------+  <--------------- 0x081BB000
 * |              |     ^
 * | fw_header    |     |  4KB
 * |              |     v
```

```
 * +--------------+  <--------------- 0x081BC000
 * |              |     ^
 * | wifi_fw      |     |  256KB : fmacfw image
 * |              |     v
 * +--------------+  <--------------- 0x081FC000
 */
```

bootloader_fw_lzma.h 支持 flash 版 wifi 协议栈，支持压缩升级，划分当前分区、备份分区、更新分区、数据分区、协议栈分区，**特别注意，**协议栈分区不可修改！

可修改分区

1. 数据分区设置28KB，结合实际应用增减
2. 根据用户应用需求，可以选择删除备份分区

5. 删除备份分区说明

以 botloader.h 分区为例进行说明，

```
/**
 * Memory map with bootloader:
 * +--------------+ <---------------- 0x08000000
 * |              |      ^
 * | bootloader   |      | 12KB
 * |              |      v
 * +--------------+ <---------------- 0x08003000
 * |              |      ^
 * | image_info   |      | 4KB
 * |              |      v
 * +--------------+ <---------------- 0x08004000
 * |              |      ^
 * | cur_image    |      | 636KB :: current image
 * |              |      v
 * +--------------+ <---------------- 0x080A3000
 * |              |      ^
 * | image_header |      | 4KB
 * |              |      v
 * +--------------+ <---------------- 0x080A4000
 * |              |      ^
 * | bak_image    |      | 636KB :: backup image
 * |              |      v
 * +--------------+ <---------------- 0x08143000
 * |              |      ^
 * | image_header |      | 4KB
 * |              |      v
 * +--------------+ <---------------- 0x08144000
 * |              |      ^
 * | upg_image    |      | 636KB :: upgrade image
 * |              |      v
 * +--------------+ <---------------- 0x081E3000
 * |              |      ^
 * | user_data    |      | 100KB :: user specified
 * |              |      v
 * +--------------+ <---------------- 0x081FC000
 * |              |      ^
 * | bt_ble_info  |      | 4KB
 * |              |      v
 * +--------------+ <---------------- 0x081FD000
 * |              |      ^
 * | wifi_info    |      | 4KB
 * |              |      v
 * +--------------+ <---------------- 0x081FE000
 * |              |      ^
 * | calib_info   |      | 4KB :: factory calib(RO)
 * |              |      v
 * +--------------+ <---------------- 0x081FF000
 * |              |      ^
 * | boot_info    |      | 4KB :: chip reserved(RO)
 * |              |      v
 * +--------------+ <---------------- 0x08200000
 */

#define IMAGE_INFO_SIZE             0x1000

#define CURRENT_START_ADDR          0x08003000
#define CURRENT_INFO_ADDR           (CURRENT_START_ADDR)
#define CURRENT_IMAGE_ADDR          (CURRENT_START_ADDR + IMAGE_INFO_SIZE)

#define UPGRADE_START_ADDR          0x08143000
#define UPGRADE_INFO_ADDR           (UPGRADE_START_ADDR)
#define UPGRADE_IMAGE_ADDR          (UPGRADE_START_ADDR + IMAGE_INFO_SIZE)

#define BACKUP_START_ADDR           0x080A3000
#define BACKUP_INFO_ADDR            (BACKUP_START_ADDR)
#define BACKUP_IMAGE_ADDR           (BACKUP_START_ADDR + IMAGE_INFO_SIZE)

#define USERDATA_START_ADDR         0x081E3000

#define CURRENT_IMAGE_MAX_SIZE      (BACKUP_START_ADDR - CURRENT_IMAGE_ADDR)
#define BACKUP_IMAGE_MAX_SIZE       (UPGRADE_START_ADDR - BACKUP_IMAGE_ADDR)
#define UPGRADE_IMAGE_MAX_SIZE      (USERDATA_START_ADDR - UPGRADE_IMAGE_ADDR)
```

删除

修改偏移地址

1. 将分区表中 **bak_image** 及其 **image_header** 640KB，平均分配给 **cur_image** 和 **upg_image** 分区
2. 重新计算 **CURRENT_START_ADDR** 和 **UPGRADE_START_ADDR** 偏移地址

若是使用压缩升级功能，还需要修改压缩脚本，脚本中升级分区起始地址和bootloader头文件中保持一致

image_pack_ota.py

文件 编辑 查看

```
def pack_image_lzma(filename, version):
    try:
            import pylzma
    except ImportError:
        print 'could import pylzma module'
        exit(0)

    fname = os.path.splitext(filename)

    print 'firmware:', filename

    f = file('./source/' + filename, 'rb')
    data = f.read()
    f.close()

    magic = 0x48474D49
    # upgrade addr
    addr = 0x08157000
    encrypt_algo = 1

    crc32 = getCrc32(bytearray(data)) & 0xffffffff
    size = len(data)

    lzma_data = pylzma.compress(data, dictionary=12, eos=0)
    uncompressed_size = struct.pack("II", size&0xffffffff, size>>32)
    lzma_data = insert(lzma_data, uncompressed_size, 5)

    f = file('./output/' + fname[0] + '.bin.lzma', "wb")
    f.write(lzma_data)
    f.close()

    lzma_crc32 = getCrc32(bytearray(lzma_data)) & 0xffffffff
    lzma_size = len(lzma_data)

    print '   magic: %08x' % magic
    print '   size: ', lzma_size
    print '   version: ', version
    print '   loadaddr: %08x' % addr
    print '   crc32: %08x' % lzma_crc32
    print '   uncompress size: ', size
    print '   uncompress crc32: %08x' % crc32

    # image_user_ota
    header = struct.pack("<IIII16sIII", magic, addr, lzma_size, lzma_crc32, version, encrypt_algo, crc32, size)

    f = file('./output/' + fname[0] + '_lzma.bin', "wb")
    f.write(header)
    f.tell()
    f.seek(0x1000, 0)
    f.write(lzma_data)
    f.close()

    # delete useless file
    os.remove('./output/' + fname[0] + '.bin.lzma')
```

压缩脚本中需修改此处地址
修改为: 升级分区起始地址 + 0x1000

3 OTA 相关文件

编译及生成文件

文件	类型	描述
aic8800_ota.bin	量产初始文件	模组量产时初次烧写的bin文件
bootloader.bin	编译文件	bootloader程序
host_wb.bin	编译文件	初版的应用程序
host_wb_upg.bin	应用更新编译文件	OTA升级的应用程序
fmacfw.bin	协议栈文件	AIC 提供
fmacfw_upg.bin	更新协议栈文件	AIC 提供

不同的 bootloader 文件提供不同的生成工具

类型	名称	描述
bootloader.h	ota_bin_generator.py	将bootloader.bin与初版应用合成aic8800_ota.bin
bootloader_fw.h	ota_bin_generator.py	将bootloader.bin与初版应用合成aic8800_ota.bin
	ota_fw_generator.py	将fmacfw.bin生成aic8800_fw_ota.bin
bootloader_lzma.h	ota_bin_generator.py	将bootloader.bin与初版应用合成aic8800_ota.bin
	image_pack_ota.py	将更新的host_wb_upg.bin压缩成host_wb_upg_lzma.bin
bootloader_fw_lzma.h	ota_bin_generator.py	将bootloader.bin与初版应用合成aic8800_ota.bin
	ota_fw_generator.py	将fmacfw.bin生成aic8800_fw_ota.bin
	image_pack_user_ota.py	将更新的host_wb_upg.bin压缩成host_wb_upg_lzma.bin
	image_pack_wifi_ota.py	将更新的fmacfw_upg.bin压缩成fmacfw_upg_lzma.bin

4 OTA 编译描述

OTA-Tool 工具目录

```
OTA-Tool/
├── bootloader
│   ├── output # 文件生成目录
│   │   └── aic8800_ota.bin
│   ├── source # 编译文件存放目录
│   │   ├── bootloader.bin
│   │   └── host_wb.bin
│   └── ota_bin_generator.py
├── bootloader_fw
│   ├── output
│   ├── source
│   ├── ota_bin_generator.py
│   └── ota_fw_generator.py
├── bootloader_lzma
│   ├── output
│   ├── source
│   ├── image_pack_ota.py
│   └── ota_bin_generator.py
└── bootloader_fw_lzma
    ├── output
    ├── source
    ├── image_pack_user_ota.py
    ├── image_pack_wifi_ota.py
    ├── ota_fw_generator.py
```

```
└── ota_bin_generator.py
```

a. bootloader版描述 >> bootloader.h

1. 进入 config/aic8800x/target_test 目录，编译生成 bootloadr.bin

```
./build_bootloader.sh -j8
```

2. 进入 config/aic8800x/target_xxx 目录，编译生成应用 host_wb_xxx.bin 文件

```
./build_xxx.sh OTA=on CODE_START_ADDR=0x8004000 -j8
```

3. 进入 ota-tool/bootloade/ 目录

```
a. 将 host_wb_xxx.bin 统一修改成 host_wb.bin
b. 将编译后的 bootloader.bin 与 host_wb.bin 放在./source目录下
c. 利用 ota_bin_generator.py 合成 aic8800_ota.bin
d. 生成的 aic8800_ota.bin 位于./output目录下
```

4. 烧录 OTA 文件

```
# 烧录aic8800_ota.bin，需指定擦除长度，比如100000，1024K
# 注意！用户应根据生成aic8800_ota.bin的实际大小，指定擦除长度！
x 8000000 100000

# 或者，确保要清除所有数据，擦除所有的info、header和image信息，1932K
x 8000000 1e3000
```

5. 更新应用程序

```
a. 新版本的应用程序中，用户需要对OTA接口更新版本号
如 host_OTA 中，
static int custom_msg_host_ota_handler(...)
{
    ...
    // 初始版本为v0.1.0，后续版本均需要更新版本号
    memcpy(hdr.version, "v0.1.1", sizeof("v0.1.1"));
    ...
}
如 http_OTA 中，
#define OTA_CURRENT_VER    "v0.1.1"
#define OTA_NEW_VER        "v0.1.2"

b. 编译新版应用程序同样需要指定编译地址
./build_xxx.sh OTA=on CODE_START_ADDR=0x8004000 -j8
```

6. **注意！**基础版的 OTA api

```
#a. ota_start 写入新的 bin 文件时，需要从 upg_image 的起始地址开始写入
#b. 使用 ota_end 写入 image_header 完成 OTA
```

b. bootloader_fw版描述 >> bootloader_fw.h

1. 进入 config/aic8800x/target_test 目录，编译生成 bootloader.bin

```
./build_bootloader.sh FLASH_FW=on -j8
```

2. 进入 config/aic8800x/target_xxx 目录，编译生成应用 host_wb_xxx.bin 文件

```
./build_xxx.sh OTA=on FLASH_FW=on CODE_START_ADDR=0x8004000 -j8
```

3. 进入 ota-tool/bootloader_fw 目录

```
a. 将 host_wb_xxx.bin 统一修改成 host_wb.bin
b. 将编译后的 bootloader.bin、host_wb.bin、fmacfw.bin 放在./source目录下
c. 利用 ota_bin_generator.py 合成 aic8800_ota.bin
d. 利用 ota_fw_generator.py 合成 aic8800_fw_ota.bin
e. 生成的 aic8800_ota.bin、aic8800_fw_ota.bin 位于./output目录下
```

4. 烧录 OTA 文件

```
# 烧录aic8800_ota.bin，并指定擦除长度，比如100000，1024K
# 注意！用户应根据生成aic8800_ota.bin的实际大小，指定擦除长度！
x 8000000 100000

# 或者，确保要清除所有数据，擦除所有的info、header和image信息，1932K
x 8000000 1e3000

### 烧录flash版本wifi协议栈，需要将 aic8800_fw_ota.bin 写入指定地址
x 81BB000 41000  // 当前fw_header位于0x81BB000，长度260KB
注意！后续结合用户具体需求，flash版wifi协议栈存放地址可能调整，但是必须经过AIC原厂确认
```

5. 更新应用程序

```
a. 新版本的应用程序中，用户需要对 OTA 接口更新版本号
如 host_OTA 中，
static int custom_msg_host_ota_handler(...)
{
    ...
    // 初始版本为v0.1.0，后续版本均需要更新版本号
    memcpy(hdr.version, "v0.1.1", sizeof("v0.1.1"));
    ...
}
# host_OTA 暂时不支持升级wifi协议栈
如 http_OTA 中，
#define OTA_CURRENT_VER   "v0.1.1"
#define OTA_NEW_VER       "v0.1.2"

b. 编译新版应用程序同样需要指定编译地址
./build_xxx.sh OTA=on FLASH_FW=on CODE_START_ADDR=0x8004000 -j8
```

6. **注意！** bootloader_fw版的 OTA api

```
#a. ota_start 写入新的 bin 文件时，需要从 upg_image 的起始地址开始写入
#b. ota_end 应替换成 ota_type_end(ota_image_t type)，设置 OTA_USER_IMAGE 或
#   OTA_WIFI_IMAGE，指定更新的类型
```

c. bootloader_lzma版描述 >> bootloader_lzma.h

1. 进入 config/aic8800x/target_test 目录，编译生成 bootloader.bin

```
./build_lzma_bootloader.sh -j8
```

2. 进入 config/aic8800x/target_xxx 目录，编译生成应用 host_wb_xxx.bin 文件

```
./build_xxx.sh OTA=on LZMA=on CODE_START_ADDR=0x8006000 -j8
```

3. 进入 ota-tool/bootloader_lzma 目录

```
a. 将 host_wb_xxx.bin 统一修改成 host_wb.bin
b. 将编译后的 bootloader.bin、host_wb.bin 放在./source目录下
c. 利用 ota_bin_generator.py 合成 aic8800_ota.bin
d. 生成的 aic8800_ota.bin 位于./output目录下
```

4. 烧录 OTA 文件

```
# 烧录aic8800_ota.bin，并指定擦除长度，比如100000，1024K
# 注意！用户应根据生成aic8800_ota.bin的实际大小，指定擦除长度！
x 8000000 100000

# 或者，确保要清除所有数据，擦除所有的info、header和image信息，1932K
x 8000000 1e3000
```

5. 更新应用程序

```
a. 注意！bootloader_lzma（压缩版）不需要在代码中指定升级版本号，
   版本号将通过压缩工具指定！

b. 编译新版应用程序同样需要指定编译地址
./build_xxx.sh OTA=on LZMA=on CODE_START_ADDR=0x8006000 -j8
```

6. 压缩更新的应用程序

```
a. 将更新的应用程序 host_wb_xxx.bin 统一修改成 host_wb_upg.bin
b. 将 host_wb_upg.bin 放在./source目录下
c. 利用 image_pack_ota.py 压缩 host_wb_upg.bin 生成 host_wb_upg_lzma.bin
   同时需指定更新的版本号，
   如 python image_pack_ota.py v0.1.1
   windows命令行cmd执行exe，./image_pack_ota.exe v0.1.1
d. 生成的 host_wb_upg_lzma.bin 位于./output目录下
```

7. **注意！** bootloader_lzma版的 OTA api

```
#a. ota_start 写入新的 bin 文件时，需要从 upg_image 的 image_header 地址开始写入
#   区别在于压缩工具已经将 header 信息合成在压缩后的 bin 文件中
#b. 使用 ota_end 写入 image_header 完成 OTA
```

d. bootloader_fw_lzma版描述 >> bootloader_fw_lzma.h

1. 进入 config/aic8800x/target_test 目录，编译生成 bootloader.bin

```
./build_lzma_bootloader.sh FLASH_FW=on -j8
```

2. 进入 config/aic8800x/target_xxx 目录，编译生成应用 host_wb_xxx.bin 文件

```
./build_xxx.sh OTA=on LZMA=on FLASH_FW=on CODE_START_ADDR=0x8006000 -j8
```

3. 进入 ota-tool/bootloader_fw_lzma 目录

```
a. 将 host_wb_xxx.bin 统一修改成 host_wb.bin
b. 将编译后的 bootloader.bin、host_wb.bin fmacfw.bin 放在./source目录下
c. 利用 ota_bin_generator.py 合成 aic8800_ota.bin
d. 利用 ota_fw_generator.py 合成 aic8800_fw_ota.bin
e. 生成的 aic8800_ota.bin、aic8800_fw_ota.bin 位于./output目录下
```

4. 烧录 OTA 文件

```
# 烧录aic8800_ota.bin，并指定擦除长度，比如100000，1024K
# 注意！用户应根据生成aic8800_ota.bin的实际大小，指定擦除长度！
x 8000000 100000

# 或者，确保要清除所有数据，擦除所有的info、header和image信息，1932K
x 8000000 1e3000

### 烧录flash版本wifi协议栈，需要将 aic8800_fw_ota.bin 写入指定地址
x 81BB000 41000  // 当前fw_header位于0x81BB000，长度260KB
注意！ 后续结合用户具体需求，flash版wifi协议栈存放地址可能调整，但是必须经过AIC原厂确认
```

5. 更新应用程序

```
a. 注意！bootloader_fw_lzma（压缩版）不需要在代码中指定升级版本号，
   版本号将通过压缩工具指定！

b. 编译新版应用程序同样需要指定编译地址
./build_xxx.sh OTA=on LZMA=on FLASH_FW=on CODE_START_ADDR=0x8006000 -j8
```

6. 压缩更新的应用程序

```
a. 将更新的应用程序 host_wb_xxx.bin 统一修改成 host_wb_upg.bin
b. 将 host_wb_upg.bin 放在./source目录下
c. 利用 image_pack_user_ota.py 压缩 host_wb_upg.bin 生成 host_wb_upg_lzma.bin
   同时需指定更新的版本号，
   如 python image_pack_user_ota.py v0.1.1
   windows命令行cmd执行exe，./image_pack_user_ota.exe v0.1.1
d. 生成的 host_wb_upg_lzma.bin 位于./output目录下
```

7. 压缩更新的wifi协议栈

```
a. 将更新的协议栈 fmacfw.bin 统一修改成 fmacfw_upg.bin
b. 将 fmacfw_upg.bin 放在./source目录下
c. 利用 image_pack_wifi_ota.py 压缩 fmacfw_upg.bin 生成 fmacfw_upg_lzma.bin
   同时需指定更新的版本号，
   如 python image_pack_wifi_ota.py v0.1.1
   windows命令行cmd执行exe，./image_pack_wifi_ota.exe v0.1.1
d. 生成的 fmacfw_upg_lzma.bin 位于./output目录下
```

8. **注意！**bootloader_fw_lzma版的 OTA api

```
#a. ota_start 写入新的 bin 文件时，需要从 upg_image 的 image_header 地址开始写入
#   区别在于压缩工具已经将 header 信息合成在压缩后的 bin 文件中
#b. ota_end 应替换成 ota_type_end(ota_image_t type)，设置 OTA_USER_IMAGE 或
#   OTA_WIFI_IMAGE，指定更新的类型
```

5 OTA 压缩库说明

1. 以上描述 python 脚本工具，用户可自行根据分区偏移地址进行修改

2. 在使用 python 脚本压缩工具之前，需安装 pylzma (python2环境)

3. 所有的工具均按照默认的配置提供 windows 版本 exe 程序

4. pylzma 库推荐源码包安装，pylzma-0.5.0 源码包链接 [pylzma · PyPI]

5. Windows 安装

```
# win10/11依赖
python2 可能需要安装 python-2.7.amd64.msi，可官网下载
# 源码编译安装
python setup.py build
python setup.py install
```

6. Linux 安装

```
### 基于ubuntu-22.04安装测试
# 依赖
sudo apt-get install python2
sudo apt-get install python2-dev
sudo apt-get install gcc

# 编译
python2 ./setup.py build
sudo python2 ./setup.py install
```

6 OTA 功能描述

http_ota

AIC8800通过网络获取升级包进行OTA，例程位于fhost_http_ota.c，其中，宏SOFTAP_MODE_OTA用于设置模式，当定义1时，应用会启动AP以及Http_server，接收升级文件；当定义0时，则会连接指定的AP以及Http_server，主动获取升级文件。

host_ota

AIC8800连接主控，由主控下发升级包进行OTA，例程位于fhostif_cmd.c，其中custom_msg_host_ota_handler是进行OTA操作的handler，主控通过 custom_msg 应用程序下发升级OTA指令。

7 OTA 版本号

OTA 升级版本号说明

```
初次编译代码，默认下次
 升级的版本号“v0.1.1”
       |
       V
 host_wb.bin  +  bootloader.bin
       |              |
       └ ------------ ┘
              |   < ------ 合成工具将初次的版本号设置为“v0.1.0”
              V
       aic8800_ota.bin     (当前软件版本号“v0.1.0”)
              |
              |   < ------ http-OTA或host-OTA升级写入host_wb_upg.bin
              |            的header中的版本号为“v0.1.1”
              |            编译host_wb_upg.bin时需要更新下次写入的软件
              |            版本号为“v0.1.2”
              |            (压缩升级时，版本号包含在host_wb_upg_lzma.bin
              |            和fmacfw_upg_lzma.bin文件中，不需要再通过API
              |            单独写入)
              V
       host_wb_upg.bin     (当前软件版本号“v0.1.1”)
              |
              |   < ------ 依次迭代升级
              V
       host_wb_upg.bin     (当前软件版本号“vx.x.x”)
```

二、原版 bootloader 描述

原始版本的 bootloader 只支持压缩升级应用程序，不支持 wifi 协议栈的升级，为维护原先的代码，做兼容处理。用户可以选择是否使用原先的 bootloader OTA 版本，通过相关的宏进行控制。以下修改点用户需注意：

1. **原版只有一个bootloader.h 文件，现修改为 bootloader_ago.h**
2. **原版OTA工具位于 /docs/Tool/OTA-Tool/bootloader_ago 目录下**
3. **编译原版的OTA时，bootloader 和 app 都必须设置 BOOT_AGO=on，引用原版 bootloader_ago.h 头文件**
4. **编译原版的target app 时不再使用 ALGO=on，替换成 LZMA=on**

版本	功能	编译类型	使用编译选项
bootloader_ago	不压缩	bootloader.bin	NULL、BOOT_AGO=on
		app.bin (host_wb.bin，下同)	OTA=on、BOOT_AGO=on
bootloader_ago	压缩	bootloader.bin	ALGO=on、BOOT_AGO=on
		app.bin	OTA=on、BOOT_AGO=on、LZMA=on

原版OTA文档

功能	文件	说明	路径
不压缩	OTA_Tools_Userguide.pdf		/docs/Tool/OTA-Tool/bootloader_ago/NO_LZMA
	ota_bin_generator.py	合成 aic8800_ota.bin	
压缩	OTA_ZLMA_Tools_Userguide.pdf		/docs/Tool/OTA-Tool/bootloader_ago/LZMA
	ota_bin_generator_lzma.py	合成 aic8800_ota.bin	
	image_pack_ota.py	压缩应用程序	