AIC8800-SDK OTA Manual

Date	Version	Notes
2023-06-16	v1.1.00	Initial release
2023-07-07	v1.1.01	Update App compilation control macro
2023-07-07	v1.1.02	Added compatible version description
2023-08-11	v1.1.03	Added compression library installation instructions
2023-08-18	v1.1.04	Added instructions for deleting backup partitions
2023-10-18	v1.1.05	Updated instructions for deleting backup partitions
2023-10-18	v1.1.06	Added upgrade process diagram
2024-05-08	v1.1.07	Added WiFi protocol stack version description

Note: There are many ways to implement ***OTA* . This SDK integrates *http-OTA and host-OTA* . *http-OTA* refers to *TCP* connection, update the application in the form of wireless network, host-OTA is the main control through *the SDIO/ USB* interface to upgrade the application**

1. Description of the new bootloader

0 OTA bootloader operation process

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

    // 1. Check if cur_image is valid
    if (0 == res[IMAGE_AREA_CURRENT]) {
        /* ... */
        // Confirm that upg_image is valid and the version is updated, then upgrade!
        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. Check if upg_image is valid
    } else if (0 == res[IMAGE_AREA_UPGRADE]) {
        // cur_image is corrupted, upgrade it!
        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 are damaged
    } else {
        // Confirm bak_image
        /* ... */
```

```
    }

    // 4. Confirm the address of cur_image and jump to run
    /* ... */

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

First, check whether cur_image is valid. The check method is to calculate the crc value of cur_image and the crc value saved in image_info.

The same method is used to check the content of the upg_image area. Secondly, compare the versions of cur_image and upg_image.

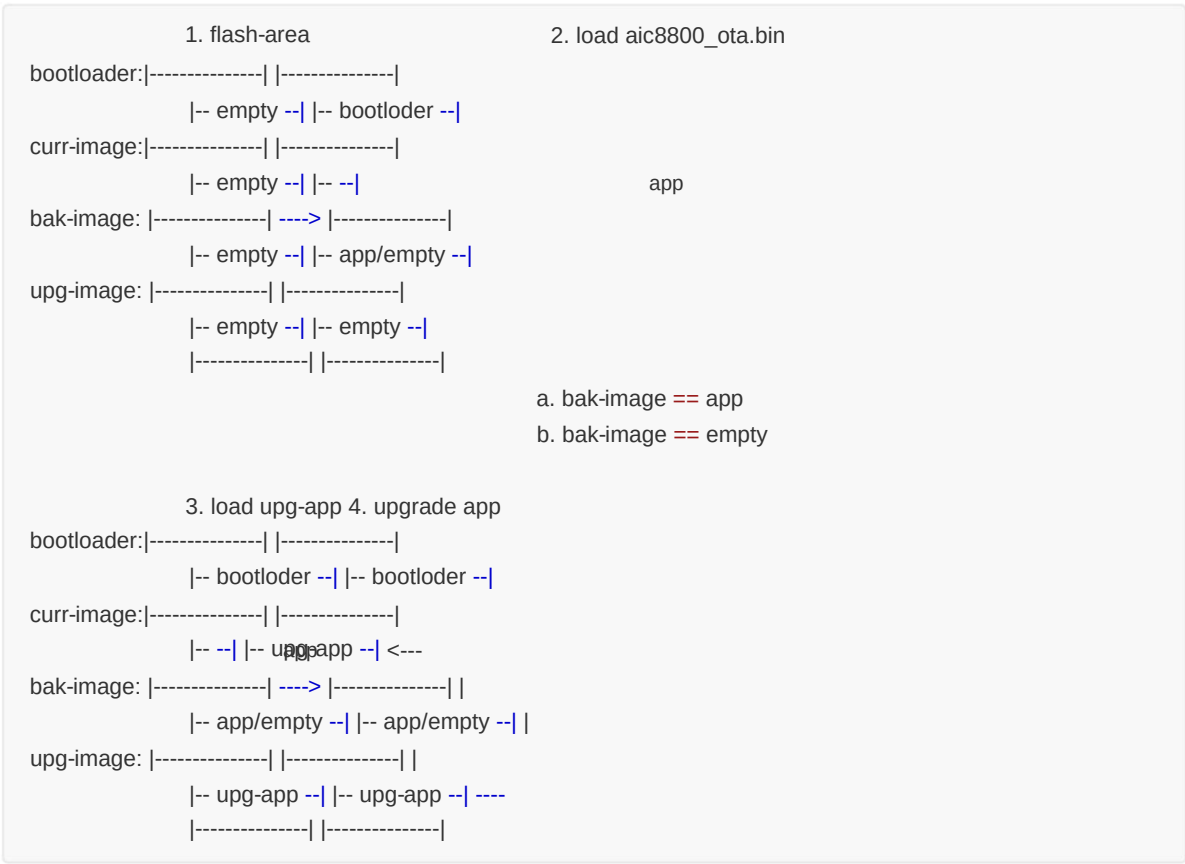
If the upg_image version is newer and valid, first copy the upg_image content to the cur_image area and jump to

cur_image is executed; otherwise, if cur_image is valid, it will jump directly to execution, otherwise it will report an error and exit.

The following figure briefly describes the basic upgrade process. The backup partition is empty if it is not used. If the user needs to use the backup partition, he should

When burning, store the file app.bin in the backup partition. If the flash space resource is tight, you can directly delete the backup partition. See the following description for details.

State.



1 OTA Function Description

In addition to the regular application upgrade function, OTA can also **upgrade the flash version of the Wi-Fi protocol stack .**

The protocol stack upgrade must confirm the wifi protocol stack to be upgraded before upgrading. Otherwise, in the http-OTA mode, it may cause

The device can no longer connect to the Internet.

Notice:

a. The wifi protocol stack refers to the underlying wifi firmware of AIC8800, not the upper-layer lwip application;

b. In addition to the default version, other versions are for special needs, which need to be communicated with the AIC manufacturer and released separately.

Chip Model	Default version of Wi-Fi protocol stack	Wi-Fi protocol stack support version
AIC8800M	ROM	ROM/RAM/FLASH
AIC8800MC	ROM	ROM/FLASH
AIC8800M40B	RAM	RAM

WiFi Protocol Stack Edition Books	describe
ROM	The protocol stack is a fixed code and does not occupy RAM and FLASH space
RAM	The protocol stack occupies RAM space and is merged into the bin file during compilation.
FLASH	The protocol stack occupies FLASH space, a separate bin file, and requires additional burning of the protocol stack code in FLASH (bin file)

When the application compiled by the user is large, the space in the flash may not be enough to allocate it. You can consider using the compression upgrade method.

Compress the bin files of the upgrade partition and backup partition, and then write them into the corresponding partitions. The compression method is non-compression, and the compression rate is high.

The compression rate is 62.5%~71.4%. The compression rate will vary according to the actual bin file. The user should refer to the last compiled application.

Sufficient margin should be reserved when partitioning.

Based on whether the WiFi protocol stack is supported and whether compression upgrade is supported, 4 demos are currently set for the flash partition.

The actual application needs can be adjusted on the demo.

The partition table is located in the AIC8800-SDK/plf/aic8800x/src/driver/flash directory.

Version	Function	Compilation Type	Use compile options
bootloader	Wi-Fi protocol not supported Stack, compression not supported	bootloader.bin	NULL
		app.bin (host_wb.binÿÿ same)	OTA=is
bootloader_fw	Support wifi protocol stack, No compression support	bootloader.bin	FLASH_FW=on
		app.bin	OTA=isÿ FLASH_FW=on
bootloader_lma	Wi-Fi protocol not supported Stack, support compression	bootloader.bin	SOMETHING=on
		app.bin	OTA=onÿLZMA=on
bootloader_fw_lma	Support wifi protocol stack, Support compression	bootloader.bin	SOMETHING=onÿ FLASH_FW=on
		app.bin	OTA=onÿLZMA=onÿ FLASH_FW=on

1. Some of the compilation options described above are included in the scripts in the target directory, such as ALGO

2. The macro ***CFG_LZMA*** in the ALGO control code ***is used when compiling bootloader.bin***

3. The macro CFG_LZMA_APP in the LZMA control code is used ***when compiling the application*** app.bin. The following compilation description has detailed instructions.

2 OTA flash partition

```
/**
 *    Memory map with bootloader:
 *    +-------------+ <-------------- 0x08000000
 * | | * | bootloader | * | |                    ^
                                                  | 12KB/20KB (compressed version supported)
                                                  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)
 0x081FF000                                       v
 *
 * | *                                            ^
 | boot_info * |                                  | 4KB : chip reserved(RO)
                                                  v
 *    +-------------+ <-------------- 0x08200000
 */
```

The partition framework of the 2M flash version is as above. It is mainly divided into three parts: bootloader, image and system_info. Pay special attention to the different

There may be slight differences in the chips in the series, but the main framework of the partitions remains the same.

Bootloader: Two versions are distinguished based on whether compression is supported

Image: This part stores the image bin file and its header information

system_info: stores the factory configuration information of AIC8800x, erasing is not allowed

image partition description

0 Overview

A complete image includes header and body, such as the current partition image_info + cur_image, the upgraded partition image_header

+ upg_image; image_info, image_header save the basic information of the image, fixed size of 4KB, cannot be adjusted.

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 **divides the image** into 4 areas: current partition, backup partition, update partition, and data partition.

Yes, you need to ensure that the image bodies of the three partitions have the same size.

Modifiable partitions

1. Data partition is set to 100KB, increase or decrease according to actual application
2. According to user application requirements, you can choose to delete the backup partition

2. bootloader_fw.h

```
/** version: bootloader_fw.h **/
/**
 *    +--------------+ <---------- 0x08003000
 * | | * | image_info | * | |                ^
 +--------------+ <----------                 | 4KB
 0x08004000                                    v
 *
 * | *                                         ^
 | 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 supports the flash version of the wifi protocol stack, and adds a protocol stack partition. Please note that the protocol stack partition cannot be modified.

change!

Modifiable partitions

1. Data partition is set to 32KB, increase or decrease according to actual application
2. According to user application requirements, you can choose to delete the backup partition

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 does not support the flash version of the wifi protocol stack, divide the current partition, backup partition, update partition, data partition, support

Supports compression function, bak_image and upg_image can be compressed and stored.

Modifiable partitions

1. Data partition is set to 100KB, increase or decrease according to actual application
2. According to user application requirements, you can choose to delete the backup partition

4. bootloader_fw_lzma.h

```
/** version: bootloader_fw_lzma.h **/
/**
 *    +--------------+ <---------- 0x08005000
 * | | * | image_info | * | |                  ^
 +--------------+ <----------                   | 4KB
 0x08006000                                      v
 *
 * | *                                           ^
 | cur_image * |                                 | 712KB : current image
                                                 v
 *+--------------+ <---------- 0x080B8000
 * | | * | image_header | * | |                ^
 +--------------+ <----------                   | 4KB
 0x080B9000                                      v
 *
 * | *                                           ^
 | bak_image * |                                 | 500KB : backup image
                                                 v
 *+--------------+ <---------- 0x08136000
 * | | * | image_header | * | |                ^
 +--------------+ <----------                   | 4KB
 0x08137000                                      v
 *
 * | *                                           ^
 | 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 supports flash version of wifi protocol stack, supports compression upgrade, divides current partition, backup partition, update partition

Area, data partition, protocol stack partition. Pay special attention that the protocol stack partition cannot be modified!

Modifiable partitions

1. Data partition is set to 28KB, increase or decrease according to actual application
2. According to user application requirements, you can choose to delete the backup partition

5. Delete backup partition instructions

Take the botloader.h partition as an example.

```
/**
 * 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. Distribute **bak_image** and its **image_header** 640KB in the partition table equally to **cur_image** and **upg_image .**

   district

2. Recalculate **the CURRENT_START_ADDR** and **UPGRADE_START_ADDR** offset addresses

If you use the compression upgrade function, you also need to modify the compression script. The starting address of the upgrade partition in the script should be consistent with that in the bootloader header file.

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 related files

Compile and generate files

document	type	describe
aic8800_ota.bin	Initial production files	The bin file that is burned for the first time when the module is mass-produced
bootloader.bin	Compile the file	Bootloader
host_wb.bin	Compile the file	Initial version of the application
host_wb_upg.bin	Apply update compilation file	OTA update application
fmacfw.bin	Protocol stack files	AIC Provides
fmacfw_upg.bin	Update protocol stack files	AIC Provides

Different bootloader files provide different generation tools

type	name	describe
bootloader.h	ota_bin_generator.py	Combine bootloader.bin with the first version of the application aic8800_ota.bin
bootloader_fw.h	ota_bin_generator.py	Combine bootloader.bin with the first version of the application aic8800_ota.bin
	ota_fw_generator.py	Generate fmacfw.bin into aic8800_fw_ota.bin
bootloader_lzma.h	ota_bin_generator.py	Combine bootloader.bin with the first version of the application aic8800_ota.bin
	image_pack_ota.py	Compress the updated host_wb_upg.bin into host_wb_upg_lzma.bin
bootloader_fw_lzma.h	ota_bin_generator.py	Combine bootloader.bin with the first version of the application aic8800_ota.bin
	ota_fw_generator.py	Generate fmacfw.bin into aic8800_fw_ota.bin
	image_pack_user_ota.py	Compress the updated host_wb_upg.bin into host_wb_upg_lzma.bin
	image_pack_wifi_ota.py	Compress the updated fmacfw_upg.bin into fmacfw_upg_lzma.bin

4 OTA compilation description

OTA-Tool Tool Catalog

```
OTA-Tool/
 ÿÿÿ bootloader
 ÿ ÿÿÿ output #File generation directory
 ÿ ÿ ÿÿÿ aic8800_ota.bin
 ÿ ÿÿÿ source #Compiled file storage directory
 ÿ ÿ ÿÿÿ bootloader.bin
 ÿ ÿ ÿÿÿ host_wb.bin
 ÿ ÿÿÿ ota_bin_generator.py
 ÿÿÿ bootloader_fw
 ÿ ÿÿÿ output
 ÿ ÿÿÿ source
 ÿ ÿÿÿ ota_bin_generator.py
 ÿ ÿÿÿ ota_fw_generator.py
 ÿÿÿ bootloader_lzma
 ÿ ÿÿÿ output
 ÿ ÿÿÿ source
 ÿ ÿÿÿ image_pack_ota.py
 ÿ ÿÿÿ ota_bin_generator.py
 ÿÿÿ bootloader_fw_lzma
     ÿÿÿ output
     ÿÿÿ source
     ÿÿÿ image_pack_user_ota.py
     ÿÿÿ image_pack_wifi_ota.py
     ÿÿÿ ota_fw_generator.py
```

```
ÿÿÿ ota_bin_generator.py
```

a. Bootloader version description >> bootloader.h

1. Enter the config/aic8800x/target_test directory and compile to generate bootloader.bin

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

2. Enter the config/aic8800x/target_xxx directory and compile and generate the application host_wb_xxx.bin file

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

3. Enter the ota-tool/bootloade/ directory

```
a. Modify host_wb_xxx.bin to host_wb.bin b. Put the compiled
bootloader.bin and host_wb.bin in the ./source directory c. Use ota_bin_generator.py to
synthesize aic8800_ota.bin d. The generated aic8800_ota.bin is located in
the ./output directory
```

4. Burn OTA file

```
#Burn aic8800_ota.bin, need to specify the erase length, such as 100000, 1024K #Note !
Users should specify the erase length according to the actual size of the generated aic8800_ota.bin!
x 8000000 100000

#Or , make sure to clear all data, erase all info, header and image information, 1932K x 8000000
1e3000
```

5. Update the application

```
a. In the new version of the application, the user needs to update
the version number
of the OTA interface such as host_OTA , static int
custom_msg_host_ota_handler(...) {
        ...
        // The initial version is v0.1.0, and subsequent versions
        need to update the version number memcpy(hdr.version, "v0.1.1", sizeof("v0.1.1"));
        ...

} For example , in
http_OTA , #define OTA_CURRENT_VER "v0.1.1"
#define OTA_NEW_VER                 "v0.1.2"

b. Compiling a new version of the application also
requires specifying the compilation address./build_xxx.sh OTA =on CODE_START_ADDR=0x8004000 -j8
```

6. Attention! Basic version of OTA API

```
#a. When ota_start writes a new bin file, it is necessary to start writing from the starting address
of upg_image . #b. Use ota_end to write image_header to complete OTA
```

b. bootloader_fw version description >> bootloader_fw.h

1. Enter the config/aic8800x/target_test directory and compile to generate bootloader.bin

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

2. Enter the config/aic8800x/target_xxx directory and compile and generate the application host_wb_xxx.bin file

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

3. Enter the ota-tool/bootloader_fw directory

```
a. Modify host_wb_xxx.bin to host_wb.bin b. Put the compiled
bootloader.bin, host_wb.bin, and fmacfw.bin in the ./source directory c. Use ota_bin_generator.py to
synthesize aic8800_ota.bin d. Use ota_fw_generator.py to synthesize
aic8800_fw_ota.bin e. The generated aic8800_ota.bin and aic8800_fw_ota.bin
are in the ./output directory
```

4. Burn OTA file

```
#Burn aic8800_ota.bin and specify the erase length, such as 100000, 1024K #Note ! The
user should specify the erase length according to the actual size of the generated aic8800_ota.bin!
x 8000000 100000

#Or , make sure to clear all data, erase all info, header and image information, 1932K x 8000000
1e3000

### To burn the flash version of the WiFi protocol stack, you need to write aic8800_fw_ota.bin to the specified address x
81BB000 41000 // The current fw_header is located at 0x81BB000, with a length of 260KB. Note! The storage
address of the flash version of the WiFi protocol stack may be adjusted in the future based on the specific needs of users, but it must be confirmed by the AIC manufacturer.
```

5. Update the application

```
a. In the new version of the application, the user needs to update the
version number of
the OTA interface such as host_OTA , static int
custom_msg_host_ota_handler(...) {
        ...
        // The initial version is v0.1.0, and subsequent versions
        need to update the version number memcpy(hdr.version, "v0.1.1", sizeof("v0.1.1"));
        ...
}
# host_OTA does not currently support upgrading the
wifi protocol stack such
as in http_OTA , #define OTA_CURRENT_VER "v0.1.1"
#define OTA_NEW_VER                    "v0.1.2"

b. To compile a new version of the application, you also
need to specify the compilation address./build_xxx.sh OTA =on FLASH_FW=on CODE_START_ADDR=0x8004000 -j8
```

6. Attention! Bootloader_fw version of OTA API

```
#a. When writing a new bin file, ota_start needs to start writing from the starting address of upg_image .
#b. ota_end should be replaced with ota_type_end (ota_image_t type), set OTA_USER_IMAGE or # OTA_WIFI_IMAGE to
specify the update type.
```

c. bootloader_lzma version description >> bootloader_lzma.h

1. Enter the config/aic8800x/target_test directory and compile to generate bootloader.bin

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

2. Enter the config/aic8800x/target_xxx directory and compile and generate the application host_wb_xxx.bin file

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

3. Enter the ota-tool/bootloader_lzma directory

```
a. Modify host_wb_xxx.bin to host_wb.bin b. Put the compiled
bootloader.bin and host_wb.bin in the ./source directory c. Use ota_bin_generator.py to synthesize
aic8800_ota.bin d. The generated aic8800_ota.bin is located in the ./output directory
```

4. Burn OTA file

```
#Burn aic8800_ota.bin and specify the erase length, such as 100000, 1024K #Note ! The
user should specify the erase length according to the actual size of the generated aic8800_ota.bin!
x 8000000 100000

#Or , make sure to clear all data, erase all info, header and image information, 1932K x 8000000 1e3000
```

5. Update the application

```
a. Note! bootloader_lzma (compressed version) does not need to specify the upgrade version number in the code, the
      version number will be specified by the compression tool!

b. Compiling a new version of the application also
requires specifying the compilation address./build_xxx.sh OTA =on LZMA=on CODE_START_ADDR=0x8006000 -j8
```

6. Compress updated applications

```
a. Modify the updated application host_wb_xxx.bin to host_wb_upg.bin b. Put host_wb_upg.bin in
the ./source directory c. Use image_pack_ota.py to compress
host_wb_upg.bin to generate host_wb_upg_lzma.bin and specify the updated version number, such as python
      image_pack_ota.py v0.1.1
      windows command line cmd execute exe, ./
      image_pack_ota.exe v0.1.1
d. The generated host_wb_upg_lzma.bin is located in the ./output directory
```

7. Attention! Bootloader_lzma version of OTA API

```
#a. When ota_start writes a new bin file, it is necessary to start writing from the image_header address of upg_image . #The
difference is that the compression tool has synthesized the header information into the
compressed bin file . #b. Use ota_end to write image_header to complete OTA
```

d. bootloader_fw_lzma version description >> bootloader_fw_lzma.h

1. Enter the config/aic8800x/target_test directory and compile to generate bootloader.bin

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

2. Enter the config/aic8800x/target_xxx directory and compile and generate the application host_wb_xxx.bin file

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

3. Enter the ota-tool/bootloader_fw_lzma directory

```
a. Modify host_wb_xxx.bin to host_wb.bin b. Put the compiled
bootloader.bin, host_wb.bin and fmacfw.bin in the ./source directory c. Use ota_bin_generator.py to synthesize
aic8800_ota.bin d. Use ota_fw_generator.py to synthesize aic8800_fw_ota.bin e.
The generated aic8800_ota.bin and aic8800_fw_ota.bin are in the ./output directory
```

4. Burn OTA file

```
#Burn aic8800_ota.bin and specify the erase length, such as 100000, 1024K #Note ! The
user should specify the erase length according to the actual size of the generated aic8800_ota.bin!
x 8000000 100000

#Or , make sure to clear all data, erase all info, header and image information, 1932K x 8000000 1e3000


### To burn the flash version of the WiFi protocol stack, you need to write aic8800_fw_ota.bin to the specified address x
81BB000 41000 // The current fw_header is located at 0x81BB000, with a length of 260KB. Note! The storage
address of the flash version of the WiFi protocol stack may be adjusted in the future based on the specific needs of users, but it must be confirmed by the AIC manufacturer.
```

5. Update the application

```
a. Note! bootloader_fw_lzma (compressed version) does not need to specify the upgrade version number in the code.
    The version number will be specified by the compression tool!

b. To compile a new version of the application, you
also need to specify the compilation address./build_xxx.sh OTA =on LZMA=on FLASH_FW=on CODE_START_ADDR=0x8006000 -j8
```

6. Compress updated applications

a. **Modify the updated application** host_wb_xxx.bin **to** host_wb_upg.bin b. **Put** host_wb_upg.bin in the ./
source directory c. **Use** image_pack_user_ota.py **to compress**
host_wb_upg.bin **to generate** host_wb_upg_lzma.bin
At the same time, you need to
specify the updated version number, such as python
image_pack_user_ota.py v0.1.1 **windows command line cmd execute exe, ./**image_pack_user_ota.exe v0.1.1
d. **The generated** host_wb_upg_lzma.bin is located in the ./output directory

7. Compressed and updated Wi-Fi protocol stack

a. **Modify the updated protocol stack** fmacfw.bin **to** fmacfw_upg.bin b. **Put** fmacfw_upg.bin
in the ./source directory c. **Use** image_pack_wifi_ota.py **to compress**
fmacfw_upg.bin **to generate** fmacfw_upg_lzma.bin
At the same time, you need to
specify the updated version number, such as python
image_pack_wifi_ota.py v0.1.1 **windows command line cmd execute exe, ./**image_pack_wifi_ota.exe
v0.1.1 d. **The generated** fmacfw_upg_lzma.bin is located in the ./output directory

8. Attention! Bootloader_fw_lzma version of OTA API

#a. **When** ota_start **writes a new** bin file, it needs to **start writing** from the image_header address **of** upg_image . #The **difference is that the compression tool has** synthesized the header **information into the compressed** bin **file .**
#b. ota_end **should be replaced with** ota_type_end (ota_image_t type), set OTA_USER_IMAGE **or** # OTA_WIFI_IMAGE to specify the update type

5 OTA Compression Library Description

1. **The above describes** the ***python*** **script tool. Users can modify it according to the partition offset address.**

2. Before **using the** ***python*** **script compression tool, you need to install** ***pylzma (python2 environment)***

3. **All tools provide** ***windows*** **version** ***exe*** **program** according to the default configuration

4. The pylzma library is recommended to be installed from the source package. The pylzma-0.5.0 **source package link is** ***pylzma · PyPI***

5. Windows **Installation**

win10/11 depends on
python2 **and may need to install** python-2.7.amd64.msi, which can be downloaded from
the official website #
Source code compilation and
installation python setup.py build python setup.py install

6. Linux **Installation**

```
### Installation test based on ubuntu-22.04
#rely
sudo apt-get install python2
sudo apt-get install python2-dev
sudo apt-get install gcc

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

6 OTA Function Description

http_ota

AIC8800 obtains the upgrade package through the network for OTA. The routine is located in fhost_http_ota.c. The macro SOFTAP_MODE_OTA is used

In the setup mode, when the definition is 1, the application will start the AP and Http_server to receive the upgrade file; when the definition is 0, it will connect to the specified

AP and Http_server actively obtain the upgrade file.

host_ota

AIC8800 is connected to the main control, and the main control sends the upgrade package for OTA. The routine is located in fhostif_cmd.c, where

custom_msg_host_ota_handler is the handler for OTA operations. The master controller sends upgrades through the custom_msg application.

OTA instructions.

7 OTA version number

OTA upgrade version number description

Compile the code for the first time, default next time

Upgraded version number "v0.1.1"

V

host_wb.bin + bootloader.bin

ÿ ---------- ÿ

< ------ **The synthesis tool sets the initial version number to "v0.1.0"**

V

aic8800_ota.bin | (Current software version number is "v0.1.0")

< ------ **http-OTA or host-OTA upgrade write to host_wb_upg.bin**

The version number in the header is "v0.1.1"

When compiling host_wb_upg.bin, you need to update the software written next time

The version number is "v0.1.2"

(When compressing and upgrading, the version number is included in host_wb_upg_lzma.bin

and fmacfw_upg_lzma.bin files, **no need to pass API**

Write separately)

V

host_wb_upg.bin | (Current software version number is "v0.1.1")

< ------ **Iterative upgrade**

V

host_wb_upg.bin (Current software version number "vx.xx")

2. Original bootloader description

The original version of the bootloader only supports compressed upgrade applications, but does not support the upgrade of the Wi-Fi protocol stack.
Compatibility processing. Users can choose whether to use the original bootloader OTA version and control it through related macros. The following modifications are made
Users should note:

1. **The original version only has one bootloader.h file, which is now modified to bootloader_ago.h**
2. **The original OTA tool is located in the /docs/Tool/OTA-Tool/bootloader_ago directory**
3. **When compiling the original OTA, both the bootloader and the app must set BOOT_AGO=on and reference the original bootloader_ago.h header file**
4. **When compiling the original target app, ALGO=on is no longer used, but LZMA=on is used instead.**

Version	Function compilation type		Use compile options
bootloader_ago	No pressure Shrink	bootloader.bin	NULLÿBOOT_AGO=on
		app.bin (host_wb.binÿ The same below)	OTA=onÿBOOT_AGO=on
bootloader_ago compressed	bootloader.bin		ALGO=onÿBOOT_AGO=on
		app.bin	OTA=onÿBOOT_AGO=onÿ LZMA=on

Original OTA documentation

Function	Files	illustrate	path
No pressure Shrink	OTA_Tools_Userguide.pdf		/docs/Tool/OTA-Tool/bootloader_ago/NO_LZMA
	ota_bin_generator.py	synthesis aic8800_ota.bin	
Compressed	OTA_ZLMA_Tools_Userguide.pdf		/docs/Tool/OTA-Tool/bootloader_ago/LZMA
	ota_bin_generator_lzma.py	synthesis aic8800_ota.bin	
	image_pack_ota.py	Compress Application	