XRADIO Image Developer Guide

Revision 1.0

Nov 12, 2019

Declaration

THIS DOCUMENTATION IS THE ORIGINAL WORK AND COPYRIGHTED PROPERTY OF XRADIO TECHNOLOGY ("XRADIO"). REPRODUCTION IN WHOLE OR IN PART MUST OBTAIN THE WRITTEN APPROVAL OF XRADIO AND GIVE CLEAR ACKNOWLEDGEMENT TO THE COPYRIGHT OWNER.

THE PURCHASED PRODUCTS, SERVICES AND FEATURES ARE STIPULATED BY THE CONTRACT MADE BETWEEN XRADIO AND THE CUSTOMER. PLEASE READ THE TERMS AND CONDITIONS OF THE CONTRACT AND RELEVANT INSTRUCTIONS CAREFULLY BEFORE USING, AND FOLLOW THE INSTRUCTIONS IN THIS DOCUMENTATION STRICTLY. XRADIO ASSUMES NO RESPONSIBILITY FOR THE CONSEQUENCES OF IMPROPER USE (INCLUDING BUT NOT LIMITED TO OVERVOLTAGE, OVERCLOCK, OR EXCESSIVE TEMPERATURE).

THE INFORMATION FURNISHED BY XRADIO IS PROVIDED JUST AS A REFERENCE OR TYPICAL APPLICATIONS, ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS DOCUMENT DO NOT CONSTITUTE A WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. XRADIO RESERVES THE RIGHT TO MAKE CHANGES IN CIRCUIT DESIGN AND/OR SPECIFICATIONS AT ANY TIME WITHOUT NOTICE.

NOR FOR ANY INFRINGEMENTS OF PATENTS OR OTHER RIGHTS OF THE THIRD PARTIES WHICH MAY RESULT FROM ITS USE. NO LICENSE IS GRANTED BY IMPLICATION OR OTHERWISE UNDER ANY PATENT OR PATENT RIGHTS OF XRADIO.THIRD PARTY LICENCES MAY BE REQUIRED TO IMPLEMENT THE SOLUTION/PRODUCT. CUSTOMERS SHALL BE SOLELY RESPONSIBLE TO OBTAIN ALL APPROPRIATELY REQUIRED THIRD PARTY LICENCES. XRADIO SHALL NOT BE LIABLE FOR ANY LICENCE FEE OR ROYALTY DUE IN RESPECT OF ANY REQUIRED THIRD PARTY LICENCE. XRADIO SHALL HAVE NO WARRANTY, INDEMNITY OR OTHER OBLIGATIONS WITH RESPECT TO MATTERS COVERED UNDER ANY REQUIRED THIRD PARTY LICENCE.

Revision History

Version	Date	Summary of Changes
1.0	2019-11-12	Initial Version

Table 1- 1 Revision History

Contents

Tables

Figures

1 概述

Image 模块主要用于管理和操作固件。系统启动过程中对固件的加载和检查，以及 OTA 升级过程中对固件的更新和验证，都用到 Image 模块提供的功能。此文档用以解释 Image 模块相关的概念并介绍接口的使用，帮助开发者了解如何使用 Image 模块管理和操作固件。

为更好理解 Image 模块的相关概念，建议先通过文档《XRadio_Flash_Layout_Developer_Guide-CN》了解固件打包和布局相关内容。此外，由于 OTA 功能在一定程度上依赖 Image 模块，因此本文档内容也有助于了解系统的 OTA 功能。

1.1 相关概念和定义

1.1.1 Image 区域

Image 区域是指在 Flash 上划分出来用于存放固件的区域。开发者如果通过 FDCM 模块或其他方式改写或擦除 Image 区域，可能导致固件损坏系统无法启动。

在不考虑 OTA 功能时，只需要在 Flash 上指定一个 Image 区域。

工程目录下 prj_config.h 文件中配置 Image 区域的位置。

```
/* image flash ID */
#define PRJCONF_IMG_FLASH             0

/* image start address, including bootloader */
#define PRJCONF_IMG_ADDR              0x00000000
```

宏 PRJCONF_IMG_FLASH 为 Flash 设备号(具体定义可参考 Flash 驱动)，用来指定 Image 区域所在的具体 Flash 设备，宏 PRJCONF_IMG_ADDR 表示 Image 区域的起始地址（包含 boot.bin）。

image.cfg 文件中配置 image 区域的大小

```
"image"    : {"max_size": "1020K"},
```

由上面三个配置，就可以确定一个 image 区域。

在考虑 OTA 功能时，需要在 Flash 上指定两个 Image 区域。其中第一个 Image 区域仍通过上述方式确定。第二个 Image 区域的 Flash 设备号、起始地址和大小通过文件 image.cfg 配置，配置示例如下：

```
"OTA"      : {"flash": "1", "addr": "1024K", "size": "4K"},
#if (__CONFIG_OTA_POLICY == 0x01)
    "image"    : {"max_size": "1020K", "xz_max_size": "600K"},
#else
    "image"    : {"max_size": "1020K"},
#endif
```

如果第二个 Image 区域与第一个 Image 区域在相同的 Flash 设备，则配置可简化为：

```
"OTA"      : {"addr": "1024K", "size": "4K"},
#if (__CONFIG_OTA_POLICY == 0x01)
    "image"    : {"max_size": "1020K", "xz_max_size": "600K"},
#else
```

```
    "image"   : {"max_size": "1020K"},
#endif
```

其中”addr”表示 OTA 区域的起始地址，”size”表示 OTA 区域的大小，那么第二个 Image 区域的起始地址就是”add + size”（不包含 boot.bin）。如果采用 OTA 压缩升级，则第二个 Image 区域的大小由”xz_max_size”决定，否则由”max_size”决定。

1.1.2 Image sequence

支持 OTA 功能时，要求在 Flash 上有两个 Image 区域来存放固件。Image sequence 用于指定不同 Image 区域的固件。Image sequence 在代码中定义如下：

```
typedef uint8_t image_seq_t;
```

1.1.3 Section ID

从 image.cfg 中可以看到，一个 IMAGE 由多个 Section 组成。一个 Section 包含了头部和 bin 文件，也可能在尾部包含证书。Section ID 是每个 Section 的唯一标识。Image 模块根据 Section ID 找到对应的 Section 以及 Section 内部的 bin。打包时固件中的 Section ID 来自固件配置文件 image.cfg，代码中的 Section ID 定义在文件 image.h 中（示例如下），应保证这两处定义的一致性。

```
#define IMAGE_BOOT_ID       (0xA5FF5A00)
#define IMAGE_APP_ID        (0xA5FE5A01)
#define IMAGE_APP_XIP_ID     (0xA5FD5A02)
#define IMAGE_NET_ID        (0xA5FC5A03)
#define IMAGE_NET_AP_ID     (0xA5FB5A04)
#define IMAGE_WLAN_BL_ID      (0xA5FA5A05)
#define IMAGE_WLAN_FW_ID   (0xA5F95A06)
#define IMAGE_WLAN_SDD_ID   (0xA5F85A07)
#define IMAGE_APP_EXT_ID     (0xA5F75A08) /* for XR32 only */
#define IMAGE_APP_PSRAM_ID  (0xA5F65A09)
```

1.1.4 Segment

打包生成的固件除了包含 bin 文件本身以外，在每个 bin 文件前还增加了一个长 64 个字节的头部，每个 bin 文件的尾部也可以包含证书等信息。Segment 用来指定 Section 中的头部（HEADER）、bin 文件本身（BODY）或是尾部（TAILER）。Segment 在文件 image.h 中的定义如下：

```
typedef enum image_segment {
    IMAGE_SEG_HEADER = 0,
    IMAGE_SEG_BODY   = 1,
    IMAGE_SEG_TAILER = 2,
} image_seg_t;
```

1.1.5 Image validity

在 Section HEADER 中包含了 HEADER 部分的累加和，以及 BODY 加 TAILER 部分的累加和。固件累加和校验的结果用 Image validity 表示，在文件 image.h 中定义如下：

```
typedef enum image_validity {
    IMAGE_INVALID    = 0,
    IMAGE_VALID      = 1,
} image_val_t;
```

1.2 Image 与 OTA

OTA 功能的初始化参数（定义如下）可从 Image 模块提供的接口获得。参数主要包括两个 Image 区域的位置、大小以及 Bootloader 区域大小等信息。

```
typedef struct image_ota_param {
    uint32_t  ota_flash : 8;   /* flash ID of OTA area */
    uint32_t  ota_size  : 24;  /* size of OTA area */
    uint32_t  ota_addr;        /* start addr of OTA area */
    uint16_t  img_max_size;    /* image max size (excluding bootloader, the unit
is K) */
    uint16_t  img_xz_max_size; /* compressed image max size (the unit is K)*/
    uint32_t  bl_size;         /* bootloader size */
    image_seq_t running_seq;   /* running image sequence */
    uint8_t   flash[IMAGE_SEQ_NUM];   /* flash ID which the image on */
    uint32_t  addr[IMAGE_SEQ_NUM];    /* image start addr, excluding bootloader
*/
} image_ota_param_t;
```

2 使用说明

2.1 代码位置

相关代码请参考：

sdk-code/include/sys/image.h

sdk-code/src/image/image.c

2.2 接口说明

下面对 Image 模块提供的接口进行简要说明。

表 2-1 image 模块 api 说明

function	detail
image_init();	声明：int image_init(uint32_t flash, uint32_t addr, uint32_t max_size); 目的：初始化 image 模块 参数：flash：flash 设备号 addr：flash 区域的起始地址 max_size：flash 区域的大小 返回值：返回状态（0：成功；-1：失败）
image_deinit();	声明：void image_deinit(void); 目的：反初始化 image 模块 参数：无 返回值：无
image_get_ota_param();	声明：const image_ota_param_t *image_get_ota_param(void); 目的：获取 OTA 功能初始化的参数 参数：无 返回值：OTA 参数结构体的指针
image_set_running_seq();	声明：void image_set_running_seq(image_seq_t seq); 目的：设置此次系统能够启动将加载的 Image sequence 参数：seq：加载固件的 Image sequence 返回值：无

image_get_running_seq();	声明：image_seq_t image_get_running_seq(void); 目的：获取加载固件的 Image sequence 参数：无 返回值：加载固件的 Image sequence
image_get_section_addr();	声明：uint32_t image_get_section_addr(uint32_t id); 目的：获取已加载生效固件中指定 Section 的 HEADER 在 Flash 中的起始地址 参数：id：指定的 Section ID 返回值：返回指定 section 的起始地址
image_read();	声明：uint32_t image_read(uint32_t id, image_seg_t seg, uint32_t offset, void *buf, uint32_t size); 目的：读取已加载生效固件中指定区域的数据 参数：id：待读取的 section 的 id seg：segment offset：表示所读区域起始地址相对于指定 Segment 起始地址的偏移 buf：读取数据存放 Buffer 的指针 size：所读数据的长度 返回值：所读数据的长度
image_write();	声明：image_write(uint32_t id, image_seg_t seg, uint32_t offset, void *buf, uint32_t size); 目的：向已加载生效固件中指定区域写入数据 参数：id：待写入的 section 的 id seg：segment offset：表示所写区域起始地址相对于指定 Segment 起始地址的偏移 buf：写入数据存放 Buffer 的指针 size：所写数据的长度 返回值：写入数据的长度
image_get_checksum();	声明：uint16_t image_get_checksum(void *buf, uint32_t len); 目的：计算 Buffer 中指定长度数据的 16-bit 累加和 参数：buf：待计算的 Buffer 的指针 len：计算的长度 返回值：返回 16-bit 累加和计算结果
image_check_header();	声明：image_val_t image_check_header(section_header_t *sh);

	目的：校验 HEADER 部分的 16-bit 累加和 参数：sh：HEADER 结构体的指针 返回值：返回校验结果
image_check_data();	声明： image_val_t image_check_data(section_header_t *sh, void *body, uint32_t body_len, void *tailer, uint32_t tailer_len); 目的：校验 BODY 加 TAILER 部分的 16-bit 累加和 参数：sh：HEADER 结构体的指针 body：BODY 部分数据的 Buffer 指针 body_len：BODY 部分数据的长度 tailer：TAILER 部分数据的 Buffer 指针 tailer_len：TAILER 部分数据的长度 返回值：返回校验结果
image_check_section();	声明：image_val_t image_check_section(image_seq_t seq, uint32_t id); 目的：校验 Flash 上指定 Image 区域固件的指定 Section 的 HEADER、BODY 和 TAILER 部分的 16-bit 累加和 参数：seq：指定校验的 image 区域的 image sequence id：section id 返回值：返回校验结果
image_check_sections();	声明：image_val_t image_check_sections(image_seq_t seq); 目的：校验 Flash 上指定 Image 区域固件的所有 Section 的 HEADER、BODY 和 TAILER 部分的 16-bit 累加和 参数：seq：指定校验的 image 区域的 image sequence 返回值：返回校验结果
image_get_cfg();	声明：int image_get_cfg(image_cfg_t *cfg); 目的：获取 image 配置 参数：cfg：存储 image 配置的 buffer 指针 返回值：返回状态（0：成功；-1：失败）
image_set_cfg();	声明：int image_set_cfg(image_cfg_t *cfg); 目的：设置 image 配置 参数：cfg：待写入的 image 配置 返回值：返回状态（0：成功；-1：失败）

2.3 使用示例

下面是 bootloader 中调用 Image 模块接口的例子：

1. 初始化 Image 模块。

```
image_init(PRJCONF_IMG_FLASH, PRJCONF_IMG_ADDR, 0);
```

2. 获取 OTA 功能初始化的参数

```
const image_ota_param_t *iop = image_get_ota_param();
```

3. 根据 OTA 功能初始化参数判断是否支持 OTA 功能，并确定加载哪个 Image 区域的固件后，设置 Image sequence。

```
image_set_running_seq(*image_seq);
```

4. 从 Flash 中读取 IMAGE_APP_ID 对应 Section 的 HEADER，并校验 HEADER 的 16-bit 累加和。

```
image_read(IMAGE_APP_ID, IMAGE_SEG_HEADER, 0, &sh, IMAGE_HEADER_SIZE);
image_check_header(&sh);
```

5. 从 Flash 中读取 IMAGE_APP_ID 对应 Section 的 BODY（即 bin 文件数据），并校验 BODY 的 16-bit 累加和。

```
image_read(IMAGE_APP_ID, IMAGE_SEG_BODY, 0, (void *)sh.load_addr, sh.data_size);
image_check_data(&sh, (void *)sh.load_addr, sh.data_size, NULL, 0)
```

6. 反初始化 Image 模块。

```
image_deinit();
```