XRADIO OTA 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

OTA 模块为系统提供在线升级固件的功能。此文档用以解释说明 OTA 模块相关概念和定义，介绍并指导开发者使用 SDK 中的 OTA 方案。OTA 模块在一定程度上依赖 Image 模块。与 Image 相关的内容可以参考文档《XRADIO_Image_Developer_Guide-CN》。目前 OTA 模块支持两种升级方式：ping pong 模式和压缩升级模式。本文将介绍这两种升级方式。

1.1 OTA 升级原理

SDK 中的 OTA 方案通过对两个 Image 区域进行操作实现对固件的升级，整个 flash 区域布局如下图所示。

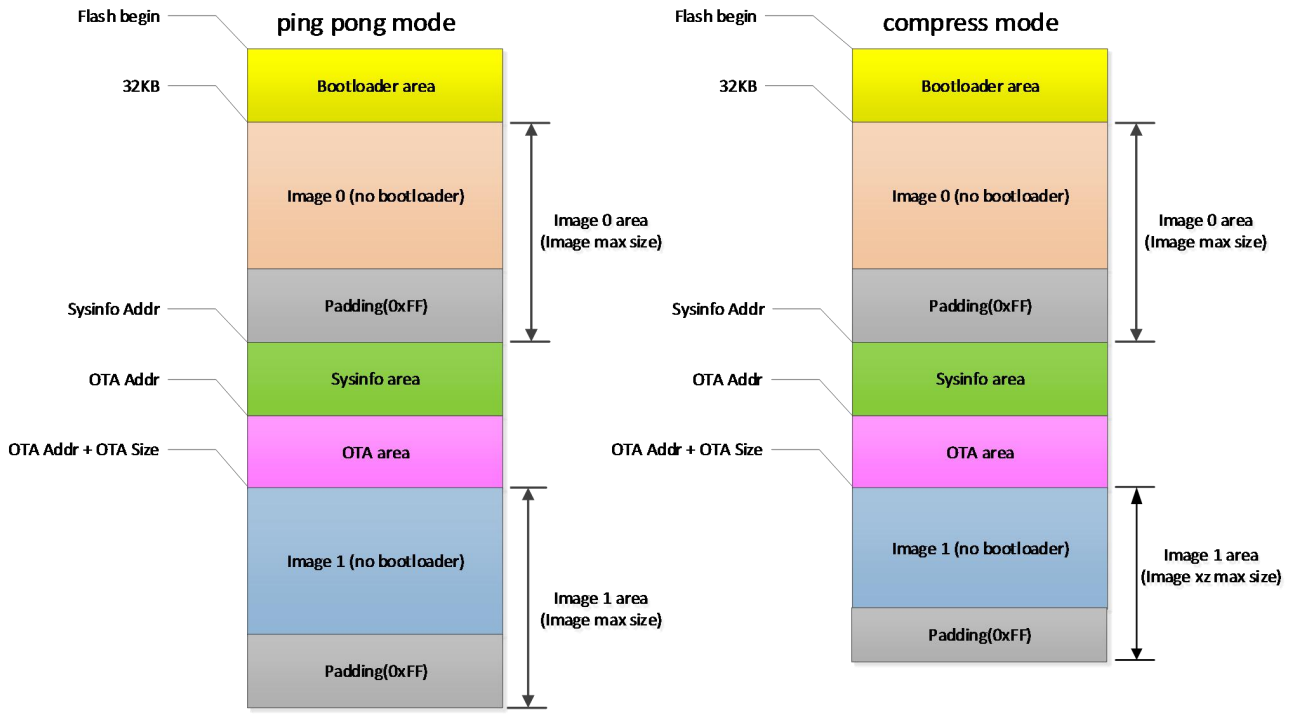

图 1-1 Flash 布局示意图

OTA 方案 flash 分布说明如下所示：

1. 在 ping pong 模式下，image0 区域和 image1 区域大小相同，均为 Image max size（不包含 BootLoader）。Image0 和 image1 交替存储 image，当 image0 作为启动项时，image1 作为新 image 的升级区域；当 image1 作为启动项时，image0 作为新 image 的升级区域。

2. 在压缩升级模式下，image0 区域和 image1 区域大小不相同，image0 要大于 image1，image0 的值为 Image max size（不包含 BootLoader），image1 的值为 Image xz max size（不包含 BootLoader）。image0 区域固定作为启动项，image1 区域用于升级时存储压缩后的 image。这样，image1 区域的空间就可以大大减小，减少 flash 的占用。

3. Image max size 加 BootLoader 后应该大于实际编译打包出来的固件大小，Image xz max size 应该大于实际

固件压缩后的大小，这样做的目的是为了给后期开发扩展预留一定的空间，预留出的部分即为 padding 部分。

4. Image 区域应该与 Flash 可擦除块对齐（比如 4K 对齐）。
5. 编译链接打包生成的固件包含了 Bootloader 和 image，通过刷机工具，固件会被烧录到 Bootloader 区域和 image0 区域，其中 Bootloader 区域大小为 32k。
6. Image1 区域前面是 OTA 区域，OTA 区域的主要功能是存放 OTA 参数，只存放了两个参数：一个是 image 区域启动项序号（即是 image0 为启动项，还是 image1 为启动项），另一个是启动项的 image 是否校验通过的状态标志。OTA 区域的大小在 image.cfg 文件定义，一般定义为 4k。
7. OTA 区域后的是 image1 区域，该区域的 image 是不包含 BootLoader 的（因为 BootLoader 只需要一个即可）。

OTA 方案原理说明如下所示：

1. 系统启动时，加载 Bootloader，Bootloader 根据 OTA 区域的参数，可以判断出采用的是什么升级模式以及应该运行哪个 image 区域的固件。
2. OTA 升级固件时，系统根据 OTA 区域的参数，判断本次擦除并更新哪个 Image 区域的固件（压缩模式下，固定更新 image1 区域的固件）。固件更新成功并校验通过后，更新 OTA 区域的参数，并重启系统。
3. 系统重启后会根据更新后的 OTA 区域的参数，判断出采用的是什么升级模式，如果是 ping pong 模式则判断并加载最新且校验通过的固件；如果是压缩升级模式，则擦除 image0 区域数据，将 image1 数据解压到 image0 区域，解压完后进行校验。如果以上所有流程都正常，则更新 OTA 区域参数，如果固件更新或校验失败，则不修改 OTA 区域，系统下次启动仍然执行以上操作。

1.2 相关概念和定义

1.2.1 OTA 区域

OTA 区域存储了 OTA 升级的记录。系统启动时，Bootloader 获取 OTA 区域的信息，就能判断该加载哪个 Image 区域固件的信息。OTA 区域中主要存储结构体类型 image_raw_cfg_t 的数据。定义如下：

```
typedef struct image_raw_cfg {
    uint16_t raw_seq;    /* magic number for image_seq_t */
    uint16_t raw_state; /* magic number for image_state_t */
} image_raw_cfg_t;
```

raw_seq 存储了 image 区域的序号。raw_state 表示该 image 区域的校验结果。

备注：在压缩升级模式中，raw_seq 固定为 1。

1.2.2 OTA protocol

OTA protocol 表示 OTA 升级时下载固件的协议。在 ota.h 中定义如下：

```
typedef enum ota_protocol {
```

```
    OTA_PROTOCOL_FILE    = 0,
    OTA_PROTOCOL_HTTP    = 1,
} ota_protocol_t;
```

当前 SDK 中支持通过 Http 和 File 两种协议下载固件。

1.2.3 OTA verify

OTA verify 表示对下载完固件的校验算法。为保证固件在 OTA 下载和烧写 Flash 过程中不出错，可采用 CRC32、MD5、SHA1 或 SHA256 算法对固件进行校验。

目前 sdk 支持的校验算法有：

```
typedef enum ota_verify {
    OTA_VERIFY_NONE      = 0,
    OTA_VERIFY_CRC32 = 1,
    OTA_VERIFY_MD5       = 2,
    OTA_VERIFY_SHA1      = 3,
    OTA_VERIFY_SHA256    = 4,
} ota_verify_t;
```

当启动 OTA 校验功能时，编译打包生成的固件末尾有额外的 verify data。应用代码可根据 verify data 对固件进行校验。

```
typedef struct ota_verify_data {
    uint32_t ov_magic;           /* OTA Verify Header Magic Number */
    uint16_t ov_length;           /* OTA Verify Data Length */
    uint16_t ov_version;          /* OTA Verify Version: 0.0 */
    uint16_t ov_type;            /* OTA Verify Type */
    uint16_t ov_reserve;
    uint8_t ov_data[OTA_VERIFY_DATA_SIZE];
} ota_verify_data_t;
```

2 使用说明

2.1 代码位置

相关代码请参考：

sdk-code/include/sys/ota.h

sdk-code/src/ota/

2.2 接口说明

下面对 OTA 模块提供的接口进行简要说明。

表 2-1 OTA 模块 api 说明

function	detail
ota_init();	声明：ota_status_t ota_init(void); 目的：初始化 OTA 模块 参数：无 返回值：返回执行状态
ota_deinit();	声明：void ota_deinit(void); 目的：反初始化 OTA 模块 参数：无 返回值：无
ota_get_image();	声明：ota_status_t ota_get_image(ota_protocol_t protocol, void *url); 目的：下载获取固件 参数：protocol：下载固件的协议，目前支持 http 和 file 两种协议 url：下载的地址 返回值：返回下载状态
ota_get_verify_data();	声明：ota_status_t ota_get_verify_data(ota_verify_data_t *data); 目的：获取 verify data 参数：data：verify data 的指针 返回值：返回状态
ota_verify_image();	声明：ota_status_t ota_verify_image(ota_verify_t verify, uint32_t *value);

	目的：通过指定算法校验下载的固件，并根据校验结果更新 OTA 区域信息 参数：verify：校验算法 value：正确固件通过指定校验算法得到的校验值 返回值：返回校验结果
ota_reboot();	声明：void ota_reboot(void); 目的：重启系统 参数：无 返回值：无

2.3 使用示例

2.3.1 升级固件

以 Http 协议下载固件为例，假设固件的 url 为 http://192.168.1.100/OTA/xr_system.img，升级固件的示例如下：

```
char url[] = "http://192.168.1.100/OTA/xr_system.img";
uint32_t *verify_value;
ota_verify_t verify_type;
ota_verify_data_t verify_data;

//通过 Http 协议下载固件
if (ota_get_image(OTA_PROTOCOL_HTTP, url) != OTA_STATUS_OK) {
    ……                    //出错处理
}

//获取校验数据和校验算法
if (ota_get_verify_data(&verify_data) != OTA_STATUS_OK) {
    verify_type = OTA_VERIFY_NONE;
    verify_value = NULL;
} else {
    verify_type = verify_data.ov_type;
    verify_value = (uint32_t*)(verify_data.ov_data);
}

//校验固件，该接口也会更新 OTA 区域信息
if (ota_verify_image(verify_type, verify_value) != OTA_STATUS_OK) {
    ……                    //出错处理
}

//重启系统
ota_reboot();
```

2.3.2 协议扩展

OTA 模块已提供 Http 和 File 两种协议下载固件，此外还支持扩展其他协议。扩展协议的方式为：

1. 在文件 ota.h 中将扩展的协议补充到数据类型 ota_protocol_t。

```
typedef enum ota_protocol {
    OTA_PROTOCOL_FILE    = 0,
    OTA_PROTOCOL_HTTP    = 1,
    OTA_PROTOCOL_XXXX    = 2,
} ota_protocol_t;
```

2. 实现扩展协议下载固件的两个回调函数，回调函数类型在文件 ota_i.h 中定义如下。可参考 Http 协议和 File 协议对两个回调函数的实现（ota_http.h、ota_http.c、ota_file.h、ota_file.c）。

```
typedef ota_status_t (*ota_update_init_t)(void *url);
typedef ota_status_t (*ota_update_get_t)(uint8_t *buf, uint32_t buf_size,
                                         uint32_t *recv_size, uint8_t *eof_flag);
```

3. 将扩展协议的两个回调函数注册到文件 ota.c 的函数 ota_get_image()中。

```
case OTA_PROTOCOL_FILE:
    return ota_update_image(url, ota_update_file_init, ota_update_file_get);
case OTA_PROTOCOL_HTTP:
    return ota_update_image(url, ota_update_http_init, ota_update_http_get);
case OTA_PROTOCOL_XXXX:
    return ota_update_image(url, ota_update_xxxx_init, ota_update_xxxx_get);
```

2.3.3 打包命令

在压缩升级模式中，需要用命令来将 image 压缩和打包。

命令名称：make image_xz

使用方式：在对应工程的 gcc 目录中执行命令。（需要配置为压缩升级模式）

打包成功后，会在 gcc 目录下生成“xr_system_img_xz.img”文件，该文件是升级的 image。同时会有如下打印：

```
$ make image_xz
cd ../image/xr872 && \
       dd if=xr_system.img of=xr_system.img.temp skip=32768 bs=1c && \
       xz       -f       -k       --no-sparse       --armthumb       --check=none
--lzma2=preset=6,dict=8KiB,lc=3,lp=1,pb=1 xr_system.img.temp && \
       mv xr_system.img.temp.xz image.xz && \
       rm xr_system.img.temp && \
       ../../../../../tools/mkimage                                                -O
-c ../../../../../project/image_cfg/image_img_xz.cfg -o xr_system_img_xz.img
记录了 1007448+0 的读入
记录了 1007448+0 的写出
1007448 字节(1.0 MB)已复制，1.02673 秒，981 kB/秒
cfg string:
{
    "magic"   :"AWIH",
    "version" :"0.4",
```

```
    "OTA"     :{"addr": "1024K", "size": "32K"},
    "image"   :{"max_size": "1020K", "xz_max_size": "600K"},
    "extern"  :{"chk_ota_area": "true", "add_ota_area":"flase","verify":"md5"},
    "count"   :1,
    "section" :[
        {"id": "0xa5fe5a01", "bin": "image.xz",   "cert": "null", "flash_offs":
"0K",   "sram_offs": "0xffffffff", "ep": "0xffffffff", "attr": "0x11"}
    ]
}


******************************** verify data ********************************
magic code: 0055AAFF
length: 16
version: 1.0
type: 2
data:
3E 0F 74 64 B4 47 04 63 3C B5 92 FC E3 02 66 6F
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

*****************************************************************************

generate image: xr_system_img_xz.img
```

2.4 配置选项

2.4.1 开启 OTA 升级

1. 开启工程对 OTA 升级的支持

   在工程的 localconfig.mk 文件中添加如下选项：

```
# enable/disable OTA, default to n
export __CONFIG_OTA := y
```

2. 选择 OTA 升级模式

   在工程的 localconfig.mk 文件中添加如下选项来选择升级的模式：

```
# ota policy, choose ota mode
#   - 0x00: ping-pong mode
#   - 0x01: image compression mode

# choose ota mode, default to 0x00
export __CONFIG_OTA_POLICY := 0x00
```

   默认为 0x00，即 ping pong 模式，如果要选择压缩模式，则选择 0x01。

2.4.2 设置 OTA 升级的地址

修改 image.cfg 文件：

```
"OTA"     : {"addr": "1024K", "size": "4K"},
```

addr 为 OTA 区域的起始地址，size 为 OTA 区域的大小，则 image1 的地址为 addr+size。

备注：addr 和 size 都要和 Flash 可擦除块对齐（比如 4K 对齐）。

2.4.3 设置 Image max size 和 Image xz max size

修改 image.cfg 文件：

```
#if (__CONFIG_OTA_POLICY == 0x01)
    "image"   : {"max_size": "1020K", "xz_max_size": "600K"},
#else
    "image"   : {"max_size": "1020K"},
#endif
```

max_size – 32K 为 Image max size，即 image0 区域的大小；xz_max_size 为 Image xz max size，即 image1 区域的大小。

备注：

1. max_size 和 Image max size 含义不一样，max_size 作用于 image.cfg 文件中，用来表示整个 image 的最大值，是包含 BootLoader（32K）的；Image max size 作用于代码中，用来表示 OTA 时需要擦除空间的最大值，是不包含 BootLoader 的，因为 OTA 升级时，BootLoader 不能被擦除。
2. 设置这两个参数非常重要，必须设置正确，否则会导致擦除过量或者 image 覆盖而无法正常启动问题。
3. 在 ping pong 模式时（__CONFIG_OTA_POLICY 为 0x00），只能存在 max_size 项；在压缩升级模式时（__CONFIG_OTA_POLICY 为 0x01），max_size 和 xz_max_size 必须都存在。因为 BootLoader 为了兼容两种模式，通过 xz_max_size 的值是否有效来判断采用什么升级模式，有效有效为压缩模式，无效则为 ping pong 模式。