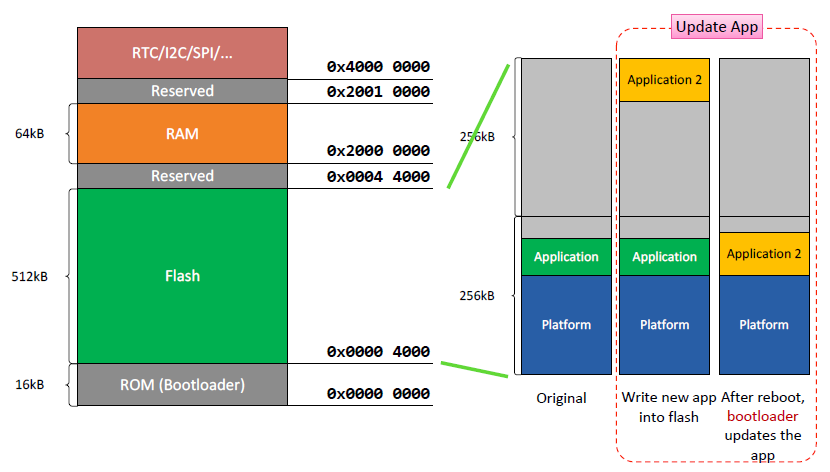
# 1. 概述

本文介绍一种将Ingchips OTA service移植到客户私有OTA service的时间方法，Ingchips OTA service的开发和演示请参考<https://ingchips.github.io/blog/fota_demo_zh/>。

# 2. Ingchips OTA

1. 下载新的程序到flash的空闲区域
2. 提供新程序的启动地址
3. 重启，BootLoader会自动搬移并完成更新



# 3. 实现细节

创建新文件ota\_private\_service.c

## 3.1 准备状态记录信息

// 包含必须的头文件

#include <stdio.h>

#include <stdint.h>

#include <string.h>

#include "ingsoc.h"

#include "platform\_api.h"

#include "rom\_tools.h"

#include "eflash.h"

#include "ota\_service.h"

// 创建状态记录信息

#define PAGE\_SIZE (8192)

typedef enum

{

OTA\_SERVICE\_DISABLED = 0,

OTA\_SERVICE\_START = (1 << 0),

OTA\_SERVICE\_END = (1 << 1)

} ota\_service\_flag\_e;

typedef enum

{

OTA\_SERVICE\_PAGE\_STATUS\_UNINIT = 0,

OTA\_SERVICE\_PAGE\_STATUS\_BEGIN,

OTA\_SERVICE\_PAGE\_STATUS\_END

} ota\_service\_page\_status\_e;

typedef struct

{

ota\_ver\_t ota\_ver;

ota\_service\_flag\_e ota\_service\_flag;//start/end

// static data from controller

uint32\_t ota\_load\_addr;//load address of new app

uint32\_t ota\_flash\_base\_addr;//flash address to save ota data

uint32\_t ota\_file\_size;//total size of new app bin

// variables to hold downloading progess

ota\_service\_page\_status\_e ota\_curr\_page\_stat;

uint32\_t ota\_curr\_page\_addr;

uint32\_t ota\_curr\_page\_size;

uint8\_t page\_buffer[PAGE\_SIZE];

uint32\_t ota\_total\_size;//size that has been program to flash

} ota\_service\_data\_s;

ota\_service\_data\_s ota\_service\_data =

{

.ota\_service\_flag = OTA\_SERVICE\_DISABLED,

.ota\_ver = { .app = {.major = 0, .minor = 0, .patch = 0} },

.ota\_curr\_page\_addr = 0,

.ota\_total\_size = 0

};

## 3.2 创建初始化接口

在开始下载之前，有三个信息需要提供：

1. load addr：新程序的启动地址。

2. flash addr：重启之前，用来放置新程序的flash基地址。

3.file size: 新程序的大小。

//需要注意的是所有地址都必须是word对齐的，文件大小不能超过flash的空闲区域大小。

uint8\_t ota\_service\_init(uint32\_t load\_addr, uint32\_t flash\_addr, uint32\_t file\_size)

{

if ((load\_addr & 0x3) || (flash\_addr & 0x3) || (!file\_size) ||

((flash\_addr + file\_size) > 0x84000))

{

return 0;

}

ota\_service\_data.ota\_load\_addr = load\_addr;

ota\_service\_data.ota\_flash\_base\_addr = flash\_addr;

ota\_service\_data.ota\_file\_size = file\_size;

ota\_service\_data.ota\_curr\_page\_addr = flash\_addr;

return 1;

}

## 3.3 创建初始化接口

准备数据的下载接口：

// 将接受到的数据暂时放在buffer中，满足一个page的大小才可以擦写到flash

#define OTA\_SERVICE\_SAVE\_PAGE\_DATA(data, len) \

{ \

memcpy(ota\_service\_data.page\_buffer + ota\_service\_data.ota\_curr\_page\_size, data, len); \

ota\_service\_data.ota\_curr\_page\_size += len; \

}

// 将一个page的数据写到flash

#define OTA\_SERVICE\_FLUSH\_PAGE\_DATA \

{ \

if(ota\_service\_data.ota\_curr\_page\_size > 0) {\

program\_flash(ota\_service\_data.ota\_curr\_page\_addr, ota\_service\_data.page\_buffer, ota\_service\_data.ota\_curr\_page\_size); \

ota\_service\_data.ota\_curr\_page\_addr += 0x2000; \

ota\_service\_data.ota\_curr\_page\_size = 0; }\

}

// 数据处理接口，下载完成后，返回1，否则返回0

uint8\_t ota\_service\_handle(const uint8\_t\* data, uint16\_t len)

{

uint8\_t complete = 0;

uint32\_t part1\_size = 0;

uint32\_t part2\_size = 0;

if((ota\_service\_data.ota\_curr\_page\_size + len) > PAGE\_SIZE)

{

part1\_size = PAGE\_SIZE - ota\_service\_data.ota\_curr\_page\_size;

part2\_size = len - part1\_size;

}

else

{

part1\_size = len;

}

if(part1\_size > 0)

{

OTA\_SERVICE\_SAVE\_PAGE\_DATA(data, part1\_size);

}

if((part2\_size > 0) || (ota\_service\_data.ota\_curr\_page\_size == PAGE\_SIZE))

{

OTA\_SERVICE\_FLUSH\_PAGE\_DATA;

}

if(part2\_size > 0)

{

OTA\_SERVICE\_SAVE\_PAGE\_DATA(data+part1\_size, part2\_size);

}

ota\_service\_data.ota\_total\_size += len;

if(ota\_service\_data.ota\_total\_size == ota\_service\_data.ota\_file\_size)

{

complete = 1;

OTA\_SERVICE\_FLUSH\_PAGE\_DATA;

}

return(complete);

}

## 3.4 创建重启接口

// 数据下载完成后，重启系统

void ota\_service\_reboot()

{

uint8\_t buffer[50];

ota\_meta\_t \*meta = (ota\_meta\_t \*)(buffer + 0);

meta->entry = 0;//(0 if use old entry address)

meta->blocks[0].dest = ota\_service\_data.ota\_load\_addr;

meta->blocks[0].src = ota\_service\_data.ota\_flash\_base\_addr;

meta->blocks[0].size = ota\_service\_data.ota\_total\_size;

program\_fota\_metadata(meta->entry, 1, meta->blocks);

platform\_reset();

}

# 4. 实现细节

调用流程如下：

1. 在profile.c中找到以下API，以及确定OTA使用到的handle（eg. HANDLE\_OF\_DATA）

static int att\_write\_callback(……)

{

switch (att\_handle)

{

case HANDLE\_OF\_DATA:

// add your code

return 0;

default:

return 1;

}

}

2. 主设备发送start cmd，启动OTA，并传递相关信息(load\_addr, flash\_addr, file\_size)

case HANDLE\_OF\_DATA:

if(start cmd)

{

ret = ota\_service\_init(load\_addr, flash\_addr, file\_size);

}

return 0;

3. 主设备发送data cmd，开始OTA下载

case HANDLE\_OF\_DATA:

if(data cmd)

{

ret &= ota\_service\_handle(buffer, buffer\_size);

}

return 0;

4. 主设备发送reboot cmd ，重启设备，完成OTA下载

case HANDLE\_OF\_DATA:

if(reboot cmd) && (ret == 1)

{

ota\_service\_reboot();

}

return 0;

# 5. 可能的问题

1. 下载程序没有添加应用层的校验，视情况可以自行实现，比如CRC。

2. 没有验证功能（将下载后的程序读取到主设备，对比验证），读取部分可以用memcpy实现，可以根据情况自行添加。