

STM32通用Bootloader——FOTA

kk AIoT开源项目分享 5月28日

1.固件升级简述

固件升级，通常称为 OTA（Over the Air）升级或者 FOTA（Firmware Over-The-Air）升级，即固件通过空中下载进行升级的技术。

1.1 bootloader 升级模式

bootloader 的升级模式常见有以下两种：

1. bootloader 分区 + app1 分区 + app2 分区模式该模式下，bootloader 启动后，检查 app1 和 app2 分区，哪个固件版本最新就运行哪个分区的固件。当有新版本的升级固件时，固件下载程序会将新的固件下载到另外的一个没有运行的 app 分区，下次启动的时候重新选择执行新版本的固件。优点：无需固件搬运，启动速度快。缺点：app1 分区和 app2 分区通常要大小相等，占用 Flash 资源；且 app1 和 app2 分区都只能存放 app 固件，不能存放其他固件（如 WiFi 固件）。
2. bootloader 分区 + app 分区 + download 分区模式该模式下，bootloader 启动后，检查 download 分区是否有新版本的固件，如果 download 分区内有新版本固件，则将新版本固件从 download 分区搬运到 app 分区，完成后执行 app 分区内的固件；如果 download 分区内没有新版本的固件，则直接执行 app 分区内的固件。当有新版本的升级固件时，固件下载程序会将新的固件下载到 download 分区内，重启后进行升级。优点：download 分区可以比 app 分区小很多（使用压缩固件），节省 Flash 资源，节省下载流量；download 分区也可以下载其他固件，从而升级其他的固件，如 WiFi 固件、RomFs。缺点：需要搬运固件，首次升级启动速度略慢。

RT-Thread OTA 使用的是 bootloader 升级模式 2，bootloader 分区 + app 分区 + download 分区的组合。

2.RT-OTA简介

为了能让开发者快速掌握 OTA 升级这把利器，RT-Thread 开发团队提供了通用的 Bootloader。开发者通过该 Bootloader 即可直接使用 RT-Thread OTA 功能，轻松实现对设备端固件的管理、升级与维护。

下图展示了 RT-Thread 通用 Bootloader 的软件框架：



RT-Thread 开发团队的官方Bootloader以bin文件形式提供，在线获取地址：<http://iot.rt-thread.com>

3.Flash 分区简述

通常嵌入式系统程序是没有文件系统的，而是将 Flash 分成不同的功能区块，从而形成不同的功能分区。要具备 OTA 固件升级能力，通常需要至少有两个程序运行在设备上。其中负责固件校验升级的程序称之为 bootloader，另一个负责业务逻辑的程序称之为 app。它们负责不同的功能，存储在 Flash 的不同地址范围，从而形成了 bootloader 分区和 app 分区。但多数情况下嵌入式系统程序是运行在 Flash 中的，下载升级固件的时候不会直接向 app 分区写入新的固件，而是先下载到另外的一个分区暂存，这个分区就是 download 分区，也有称之为 app2 分区，这取决于 bootloader 的升级模式。bootloader 分区、app 分区、download 分区及其他分区一起构成了分区表。分区表标识了该分区的特有属性，通常包含分区名、分区大小、分区的起止地址等。通用 Bootloader 中的分区表包含如下三个分区：

分区名	起始地址	分区大小	分区位置	介绍
app	自定义	自定义	片内 Flash	存储 app 固件
download	自定义	自定义	片内 Flash 或者片外 SPI Flash	存储待升级固件
factory	自定义	自定义	片内 Flash 或者片外 SPI Flash	存储出厂固件

4.Ymodem文件传输协议

Ymodem 是一种文本传输协议，在 OTA 应用中为空中下载技术提供文件传输的支持。基于 Ymodem协议的固件升级即为 OTA 固件升级的一个具体应用实例。

5.RT-OTA功能说明

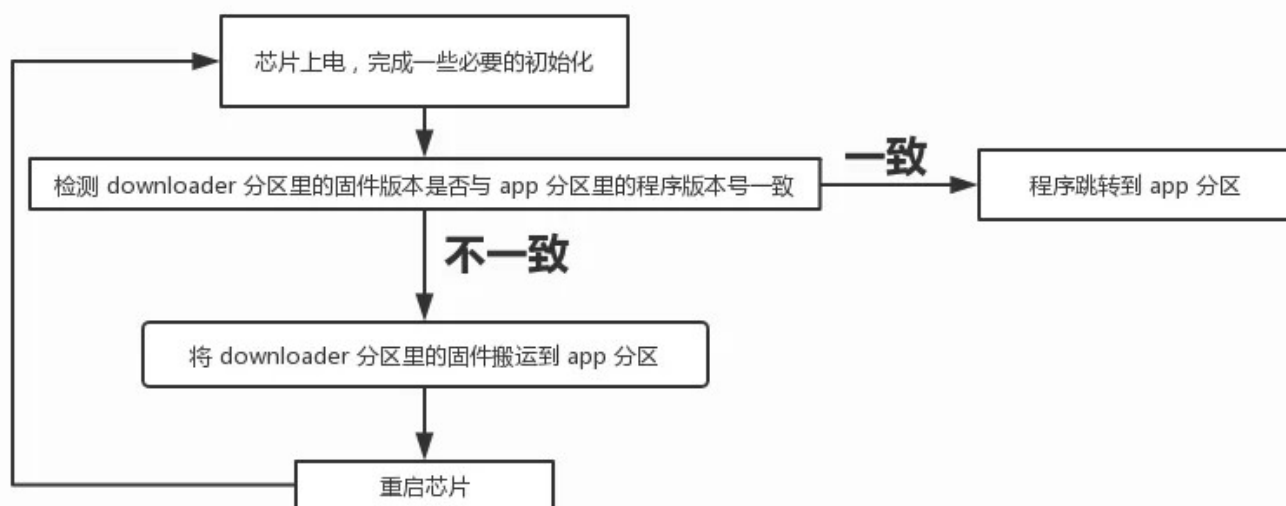
5.1 升级固件功能

当系统需要升级固件时，Bootloader 将从 download 分区将固件搬运到 app 分区，主要功能流程如下所示：

1. Bootloader 启动时检查 download 分区和 app 分区中的固件版本。
2. 如果两个固件版本相同，则跳转到 app 分区，Bootloader 运行结束。
3. 固件版本不同则将 download 分区中的固件搬运到 app 分区。
4. 在搬运的过程中 Bootloader 可以对固件进行校验、解密、解压缩等操作。
5. 搬运完毕后，删除 download 分区中存储的固件。
6. 重启系统跳转到 app 分区中的固件运行，Bootloader 运行结束。

Bootloader 工作过程如下图所示：

升级固件功能流程



5.2 恢复固件功能

当系统中的固件损坏，Bootloader 将从 factory 分区将固件搬运到 app 分区，主要功能流程

如下所示：

1. Bootloader 启动时检查触发固件恢复的引脚是否为有效电平。
2. 如果有效电平持续超过 10S 则将 factory 分区中的固件搬运到 app 分区中。
3. 如果有效电平没有持续超过 10S 则继续进行 2.2 小节中介绍的启动步骤。
4. 在搬运的过程中 Bootloader 可以对固件进行校验、解密、解压缩等操作。
5. 搬运完毕后，保持 factory 分区中的固件不变。
6. 重启系统跳转到 app 分区中的固件运行，Bootloader 运行结束。

6.RT-FOTA简介

RT-Thread官方推出了STM32系列单片机的通用bootloader,在其网站可以通过网页配置就可以生成bootloader的烧录文件，使广大嵌入式工程师不用编写一行代码，就能够轻松完成自己产品的bootloader功能。但是由于RTT官方的bootloader软件RT-OTA是商用性质，不公开源码，不仅仅限制了在其他平台的移植，而且也不方便加入产品的特定功能。所以就有了RT-FOTA的由来。RT-FOTA兼容RTThread官方OTA的所有功能，为了与官方的RT-OTA作于区分，所以取名为RT-FOTA。RT-FOTA的项目地址：https://gitee.com/spunky_973/rt-fotaRT-FOTA可以直接使用在RT-Thread的完整版搭载，只需要将rt_fota.c、rt_fota.h和rt_fota_crc.c放入工程中即可实现，然后用env配置相关组件即可。

7.RT-FOTA功能说明

1. 支持RTT官方的RBL打包软件，使用方式也一致。目前支持包括CRC32、AES256、quicklz和fastlz功能；
2. 支持命令行模式（FINSH组件）和出厂固件恢复；
3. 支持FLASH分区（FAL组件）；
4. 增加fota和ymdown命令；
5. 其他功能可自行方便扩展；

8.RT-FOTA应用示例

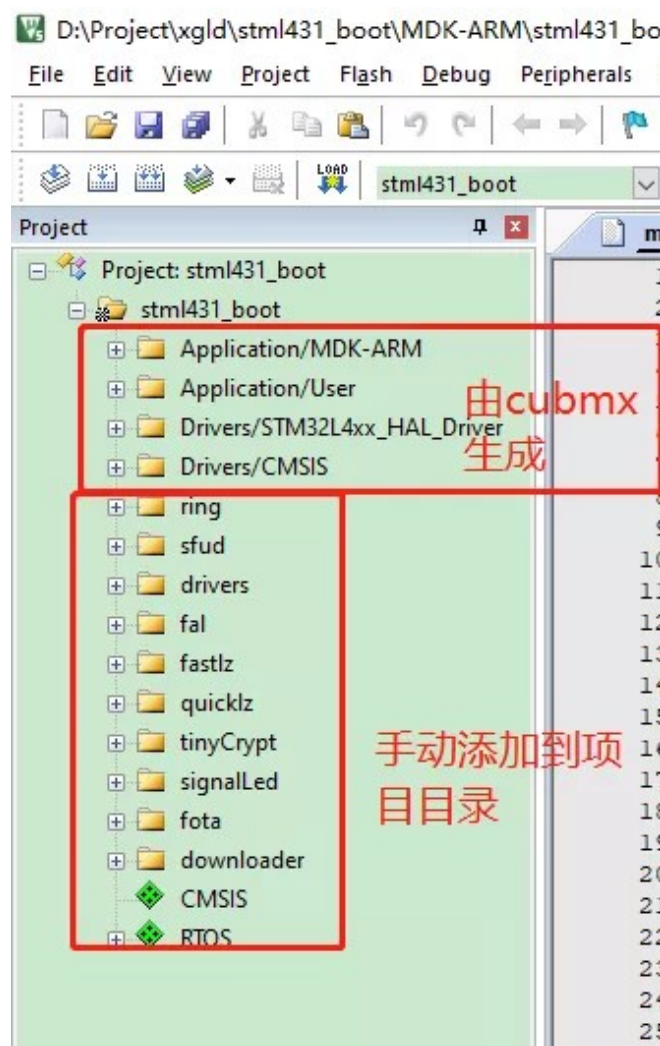
RT-FOTA的作者虽然使用的是rtthread nano版本，但是添加了完整版特有的 device 框架和finsh组件，这样的优点是很方便扩展更多的软件包，但是相应的也增加了nano的尺寸，并且对于使用者需要移植的地方相应的也会增加，如果用户使用RT-Thread的完整版搭载的话，移植虽然比较方便，但是占用flash空间就更大了。所以为了使RT-FOTA的占用空间更小，更方便

用户移植，我对RT-FOTA做了一下改进：

1. 编译环境改为Arm Compiler 6.13，优化等级改为-Oz
2. 去除device 框架，使用精简版的rtthread nano版本，只添加finsh组件
3. 使用STM32CubeMX生成对应的工程，以方便移植到其他STM32平台

对RT-FOTA重新移植后，在不影响原有功能的情况下，所占flash空间减小到42K。重新移植后的RT-FOTA项目地址：<https://gitee.com/Aladdin-Wang/RT-FOTA-STM32L431>

8.1 工程目录



8.2 软件配置说明

RT-FOTA的软件配置仍然集中在rtconfig.h中，把所有根据不同需求，需要修改的宏都集中在在了rtconfig.h中，其中需要用户修改的部分有：

```

...
#define STM32_FLASH_START_ADRESS    ((uint32_t)0x08000000)
#define STM32_FLASH_SIZE            (256 * 1024)
#define STM32_FLASH_END_ADDRESS    ((uint32_t)(STM32_FLASH_START_ADRESS +
STM32_FLASH_SIZE))

#define STM32_SRAM1_START            (0x20000000)
#define STM32_SRAM1_END              (STM32_SRAM1_START + 64 * 1024) // 结束地
址 = 0x20000000 (基址) + 64K(RAM大小)

// <<< end of configuration section >>>
/* On-chip Peripheral Drivers */
#define BSP_USING_ON_CHIP_FLASH
#define BSP_USING_LPUART1
#define BSP_USING_SPI2
/* Onboard Peripheral Drivers */
#define BSP_DATAFLASH_CS_GPIOX      GPIOB
#define BSP_DATAFLASH_CS_GPIO_PIN  GPIO_PIN_12

#define RT_FOTA_SIGNAL_LED
#define RT_FOTA_SIGNAL_LED_GPIOX    GPIOB
#define RT_FOTA_SIGNAL_LED_GPIO_PIN GPIO_PIN_1
#define RT_FOTA_SIGNAL_LED_ON_LEVEL GPIO_PIN_RESET

#define RT_FOTA_DEFAULT_KEY
#define RT_FOTA_DEFAULT_KEY_CHK_TIME 10
#define RT_FOTA_DEFAULT_KEY_GPIOX GPIOA
#define RT_FOTA_DEFAULT_KEY_GPIO_PIN GPIO_PIN_7
#define RT_FOTA_DEFAULT_KEY_LEVEL  GPIO_PIN_SET
/* package */
#define PKG_USING_FAL
#define FAL_DEBUG 1
#define FAL_PART_HAS_TABLE_CFG
#define FAL_PART_TABLE              \
{ \
    {FAL_PART_MAGIC_WROD, "app",  "onchip_flash", 64*1024,    192*1024, 0}, \

```

```
{FAL_PART_MAGIC_WROD, "ef", FAL_USING_NOR_FLASH_DEV_NAME, 0 , 1024 * 1024,
0}, \
{FAL_PART_MAGIC_WROD, "download", FAL_USING_NOR_FLASH_DEV_NAME, 1024 *
1024 , 512 * 1024, 0}, \
{FAL_PART_MAGIC_WROD, "factory", FAL_USING_NOR_FLASH_DEV_NAME, (1024 +
512) * 1024 , 512 * 1024, 0}, \
}
#define FAL_USING_SFUD_PORT
#define FAL_USING_NOR_FLASH_DEV_NAME "w25q64"

#define PKG_USING_YMODEM_OTA
#define TINY_CRYPT_AES
#define PKG_USING_QUICKLZ
#define QLZ_COMPRESSION_LEVEL 3

/* RT-FOTA module define */
#define RT_FOTA_SW_VERSION    "1.0.0"
/* FOTA application partition name */
#ifndef RT_FOTA_APP_PART_NAME
#define RT_FOTA_APP_PART_NAME  "app"
#endif

/* FOTA download partition name */
#ifndef RT_FOTA_FM_PART_NAME
#define RT_FOTA_FM_PART_NAME  "download"
#endif

/* FOTA default partition name */
#ifndef RT_FOTA_DF_PART_NAME
#define RT_FOTA_DF_PART_NAME  "factory"
#endif

/* AES256 encryption algorithm option */
#define RT_FOTA_ALGO_AES_IV "0123456789ABCDEF"
#define RT_FOTA_ALGO_AES_KEY "0123456789ABCDEF0123456789ABCDEF"

#define SOC_SERIES_STM32L4
#endif
```


8.3 RBL文件说明

使用过RTT官方的RT-OTA组件的朋友都知道，下载的不是bin文件，而是需要通过RTT打包软件“装饰”成rbl文件之后，才能被RT-OTA识别。



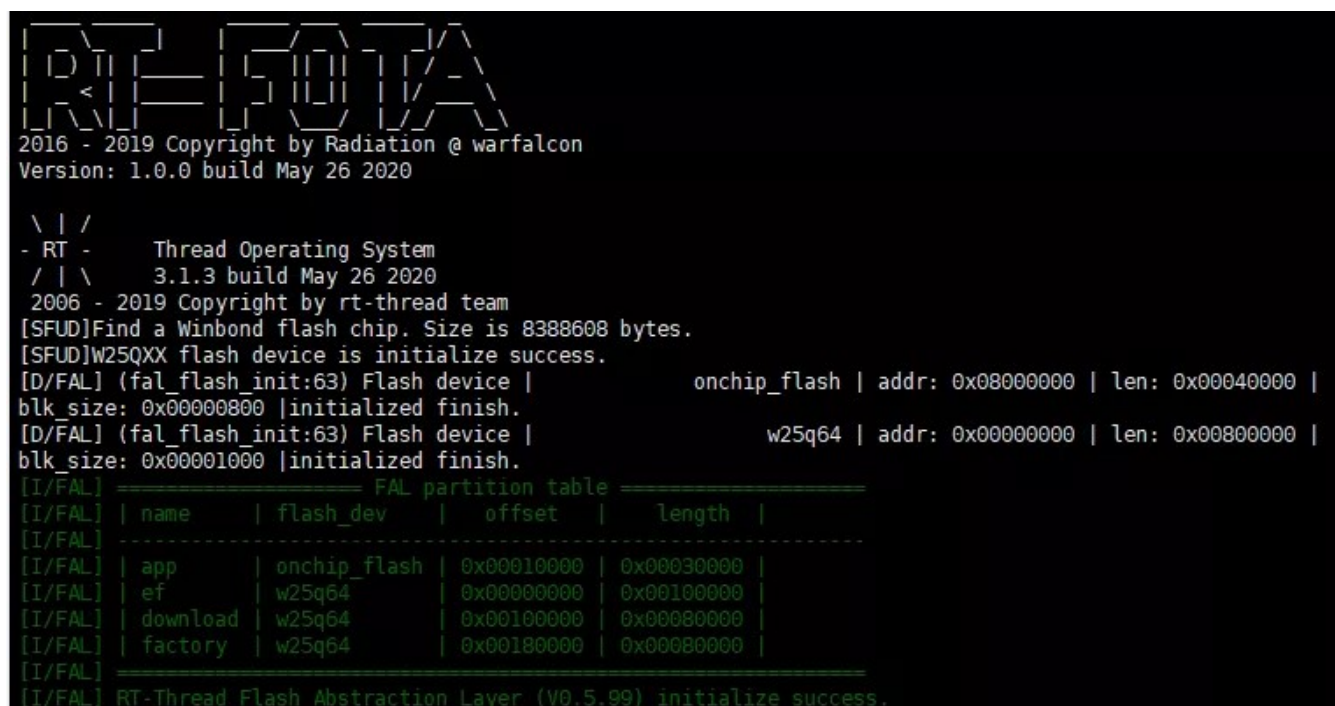
软件可以设置代码加密和压缩，其配置信息都存在rbl文件前96字节中：

```
rt-fota />fota show download 0 96
00000000: 52 42 4C 00 02 02 00 00 59 34 CB 5E 61 70 70 00
00000010: 00 00 00 00 00 00 00 00 00 00 00 00 30 2E 30 2E
00000020: 33 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000030: 00 00 00 00 30 2E 30 2E 33 00 00 00 00 00 00 00
00000040: 00 00 00 00 00 00 00 00 00 00 00 00 25 51 9A 76
00000050: 9D F9 7E DE 78 99 02 00 10 AE 01 00 5A AA C4 BE
```


其具体含义如下：

```
typedef struct {
char type[4];/* RBL字符头 */
rt_uint16_t fota_algo;/* 算法配置: 表示是否加密或者使用了压缩算法 */
rt_uint8_t fm_time[6];/* 原始bin文件的时间戳, 6位时间戳, 使用了4字节, 包含年月日信息 */
char app_part_name[16];/* app执行分区名 */
char download_version[24];/* 固件代码版本号 */
char current_version[24];/* 这个域在rbl文件生成时都是一样的, 我用于表示app分区当前运行固件的版本号, 判断是否固件需要升级 */
rt_uint32_t code_crc;/* 代码的CRC32校验值,它是的打包后的校验值,即rbl文件96字节后的数据 */
rt_uint32_t hash_val;/* 估计这个域是指的原始代码本身的校验值, 但不知道算法, 无法确认, 故在程序中未使用 */
rt_uint32_t raw_size;/* 原始代码的大小 */
rt_uint32_t com_size;/* 打包代码的大小 */
rt_uint32_t head_crc;/* rbl文件头的CRC32校验值, 即rbl文件的前96字节 */
} rt_fota_part_head, *rt_fota_part_head_t;
```

8.4 开机界面



```

RT-FOTA
2016 - 2019 Copyright by Radiation @ warfalcon
Version: 1.0.0 build May 26 2020

\ | /
- RT -   Thread Operating System
/ | \    3.1.3 build May 26 2020
2006 - 2019 Copyright by rt-thread team
[SFUD]Find a Winbond flash chip. Size is 8388608 bytes.
[SFUD]W25QXX flash device is initialize success.
[D/FAL] (fal_flash_init:63) Flash device | onchip_flash | addr: 0x08000000 | len: 0x00040000 |
blk_size: 0x00000800 |initialized finish.
[D/FAL] (fal_flash_init:63) Flash device | w25q64 | addr: 0x00000000 | len: 0x00800000 |
blk_size: 0x00001000 |initialized finish.
[I/FAL] ----- FAL partition table -----
[I/FAL] | name      | flash_dev | offset | length |
[I/FAL] |-----|-----|-----|-----|
[I/FAL] | app       | onchip_flash | 0x00010000 | 0x00030000 |
[I/FAL] | ef        | w25q64      | 0x00000000 | 0x00100000 |
[I/FAL] | download  | w25q64      | 0x00100000 | 0x00080000 |
[I/FAL] | factory   | w25q64      | 0x00180000 | 0x00080000 |
[I/FAL] -----
[I/FAL] RT-Thread Flash Abstraction Layer (V0.5.99) initialize success.
```

可以看出使用了RTT的SFUD和FAL组件，同时列出了分区表信息。这个地方与原作者的使用方式稍微做了更改，原作者的rt-fota是在开机5秒钟内，按下Enter键，即0x0d，就可以进入命令行模式。我改为了开机检测到按键有效，但小于十秒，就进入命令行模式，如果检测到有效电

检测到有效电平小于十秒：进入finsh模式：

```

RT-FOTA
2016 - 2019 Copyright by Radiation @ warfalcon
Version: 1.0.0 build May 26 2020

\ / /
- RT -      Thread Operating System
/ | \      3.1.3 build May 26 2020
2006 - 2019 Copyright by rt-thread team
[SFUD]Find a Winbond flash chip. Size is 8388608 bytes.
[SFUD]W25QXX flash device is initialize success.
[D/FAL] (fal_flash_init:63) Flash device |          onchip_flash | addr: 0x08000000 | len: 0x00040000 | blk_size: 0x00000800 | initialized finish.
[D/FAL] (fal_flash_init:63) Flash device |          w25q64      | addr: 0x08000000 | len: 0x00800000 | blk_size: 0x00001000 | initialized finish.
[I/FAL] ===== FAL partition table =====
[I/FAL] | name       | flash_dev | offset | length |
[I/FAL] |-----|-----|-----|-----|
[I/FAL] | app        | onchip_flash | 0x00010000 | 0x00030000 |
[I/FAL] | ef         | w25q64      | 0x00000000 | 0x00100000 |
[I/FAL] | download   | w25q64      | 0x00100000 | 0x00000000 |
[I/FAL] | factory    | w25q64      | 0x00180000 | 0x00000000 |
[I/FAL] |-----|-----|-----|-----|
[I/FAL] RT-Thread Flash Abstraction Layer (V0.5.99) initialize success.
rt-fota ./>The effective level continues for more than ten seconds to start to restore the default firmware.
>>>>rt-fota ./>

```

[illegible]

开机没有检测到有效电平：正常启动：

```

RT-FOTA
2016 - 2019 Copyright by Radiation @ warfalcon
Version: 1.0.0 build May 26 2020

\ | /
- RT -      Thread Operating System
/ | \      3.1.3 build May 26 2020
2006 - 2019 Copyright by rt-thread team
[SFUD]Find a Winbond flash chip. Size is 8388608 bytes.
[SFUD]W25QXX flash device is initialize success.
[D/FAL] (fal_flash_init:63) Flash device |          onchip_flash | addr: 0x08000000 | len: 0x00040000 | blk_size: 0x00000800 | initialized finish.
[D/FAL] (fal_flash_init:63) Flash device |          w25q64 | addr: 0x00000000 | len: 0x00800000 | blk_size: 0x00001000 | initialized finish.
[I/FAL] ----- FAL partition table -----
[I/FAL] | name | flash_dev | offset | length |
[I/FAL] -----
[I/FAL] | app | onchip_flash | 0x00010000 | 0x00030000 |
[I/FAL] | ef | w25q64 | 0x00000000 | 0x00100000 |
[I/FAL] | download | w25q64 | 0x00100000 | 0x00080000 |
[I/FAL] | factory | w25q64 | 0x00180000 | 0x00080000 |
[I/FAL] -----
[I/FAL] RT-Thread Flash Abstraction Layer (V0.5.99) initialize success.
[I/fota] Implement application now.

\ | /
- RT -      Thread Operating System
/ | \      4.0.2 build May 25 2020
2006 - 2019 Copyright by rt-thread team
(2) I/drv rtc: RTC hasn't been configured, please use <date> command to config.
[SFUD] Find a Winbond flash chip. Size is 8388608 bytes.
[SFUD] W25Q64 flash device is initialize success.
[D/FAL] (fal_flash_init:63) Flash device |          onchip_flash | addr: 0x08000000 | len: 0x00100000 | blk_size: 0x00020000 | initialized finish.
[D/FAL] (fal_flash_init:63) Flash device |          W25Q64 | addr: 0x00000000 | len: 0x00800000 | blk_size: 0x00001000 | initialized finish.
[I/FAL] ----- FAL partition table -----
[I/FAL] | name | flash_dev | offset | length |
[I/FAL] -----
[I/FAL] | boot | onchip_flash | 0x00000000 | 0x00010000 |
[I/FAL] | app | onchip_flash | 0x00010000 | 0x00030000 |
[I/FAL] | ef | W25Q64 | 0x00000000 | 0x00100000 |
[I/FAL] | download | W25Q64 | 0x00100000 | 0x00080000 |
[I/FAL] | factory | W25Q64 | 0x00180000 | 0x00080000 |
[I/FAL] -----
[I/FAL] RT-Thread Flash Abstraction Layer (V0.5.99) initialize success.
[Flash] (./packages/EasyFlash-latest/src/ef_env.c:1820) ENV start address is 0x00000000, size is 409600 bytes.
[Flash] EasyFlash V4.1.99 is initialize success.
[Flash] You can get the latest version on https://github.com/armink/EasyFlash .
The system now boot 19 times

```

bootloader

APP

8.5 命令行模式

RT-FOTA的命令行模式使用的RTT的FINSH组件, 除了RTT系统自带命令外, 还增加fota和ymdown命令: **fota命令**键入fota命令后回车即可看到帮助命令:

```
rt-fota /> fota
```

Usage:

```

fota probe                - probe RBL file of partiton
fota show partition addr size  - show 'size' bytes starting at 'addr'
fota clone des_part src_part  - clone src partition to des partiton
fota exec                  - execute application program

```

1. probe参数可以打印出当分区的RBL信息:

```

rt-fota />fota probe
[I/fota] ===== RBL of download partition =====
[I/fota] | App partition name |      app |
[I/fota] | Algorithm mode   |  AES && QLZ |
[I/fota] | Firmware version  |      0.0.3 |
[I/fota] | Code raw size     |    170360 |
[I/fota] | Code package size  |    110096 |
[I/fota] | Build Timestamp   | 1590375513 |
[I/fota] ===== RBL of factory partition =====
[I/fota] | App partition name |      app |
[I/fota] | Algorithm mode   |  AES && QLZ |
[I/fota] | Firmware version  |      0.0.3 |
[I/fota] | Code raw size     |    170368 |
[I/fota] | Code package size  |    110160 |
[I/fota] | Build Timestamp   | 1590393573 |

```

这里列出了fm_area和df_area分区中

RBL文件的主要信息项,便于开发者查询:

- App partition name: 指的是RTT打包文件时设置的分区名
- Algorithm mode : 指的是RTT打包文件使用那些算法: AES256/Quicklz/Fastlz
- Firmware version : 指的是RTT打包文件设置的固件版本号
- Code raw size : 指的代码原始大小
- Code package size : 指的代码打包后的大小
- Build Timestamp : 指的代码生成的时间戳

2. show参数可以显示分区的具体实际数据, 方便调试与检查:

```

rt-fota />fota show download 0 96
00000000: 52 42 4C 00 02 02 00 00 59 34 CB 5E 61 70 70 00
00000010: 00 00 00 00 00 00 00 00 00 00 00 00 30 2E 30 2E
00000020: 33 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000030: 00 00 00 00 30 2E 30 2E 33 00 00 00 00 00 00 00
00000040: 00 00 00 00 00 00 00 00 00 00 00 00 25 51 9A 76
00000050: 9D F9 7E DE 78 99 02 00 10 AE 01 00 5A AA C4 BE

```

3. clone参数是实现分区数据克隆: 这里是将factory分区数据完整的克隆到download分区中。

```

rt-fota />fota clone download factory
Clone factory partition to download partition:
#####
Clone partition success, total 524288 bytes!
rt-fota />

```

4. exec参数是用于执行app分区的应用代码:


```

rt-fota />fota exec
[I/fota] Implement application now.

\ | /
- RT -      Thread Operating System
/ | \      4.0.2 build May 25 2020
2006 - 2019 Copyright by rt-thread team
[2] I/drv.rtc: RTC hasn't been configured, please use <date> command to config.
[SFUD] Find a Winbond flash chip. Size is 8388608 bytes.
[SFUD] W25Q64 flash device is initialize success.
[D/FAL] (fal_flash_init:63) Flash device |          onchip_flash | addr: 0x08000000 | len: 0x00100000 |
blk_size: 0x00020000 | initialized finish.
[D/FAL] (fal_flash_init:63) Flash device |          W25Q64 | addr: 0x00000000 | len: 0x00800000 |
blk_size: 0x00001000 | initialized finish.

```

ymdown命令：ymdown是基于Ymodem协议的下载命令，使用RTT的ymodem和ymodem_ota组件实现，其中将ymodem_ota.c中的DEFAULT_DOWNLOAD_PART设置为需要默认使用分区名，即在使用ymdown不带参数的情况下就下载到DEFAULT_DOWNLOAD_PART分区，也可加分区名作为参数指定下载位置。

1. ymodem_ota命令

```

rt-fota />ymodem_ota
Default save firmware on download partition.
Warning: Ymodem has started! This operator will not recovery.
Please select the ota firmware file and use Ymodem to send.
CCCCCYmodem file_size:110256
[I/ymodem] Start erase. Size (110256)
Download firmware to flash success.
System now will restart...

```

下载固件成功

```

RT-Thread
2016 - 2019 Copyright by Radiation @ warfalcon
Version: 1.0.0 build May 26 2020

\ | /
- RT -      Thread perating System
/ | \      3.1.3 build May 26 2020
2006 - 2019 Copyright
by rt-thread team
[SFUD]Find a Winbond flash chip. Size is 8388608 bytes.
[SFUD]W25QXX flash device is initialize success.
[D/FAL] (fal_flash_init:63) Flash device |          onchip_flash | addr: 0x08000000 | len: 0x00040000 |
blk_size: 0x00000800 | initialized finish.
[D/FAL] (fal_flash_init:63) Flash device |          w25q64 | addr: 0x00000000 | len: 0x00800000 |
blk_size: 0x00001000 | initialized finish.
[I/FAL] ===== FAL partition table =====
[I/FAL] | name      | flash_dev | offset | length |
[I/FAL] |-----|-----|-----|-----|
[I/FAL] | app       | onchip_flash | 0x00010000 | 0x00030000 |
[I/FAL] | ef        | w25q64      | 0x00000000 | 0x00100000 |
[I/FAL] | download  | w25q64      | 0x00100000 | 0x00800000 |
[I/FAL] | factory   | w25q64      | 0x00180000 | 0x00800000 |
[I/FAL] =====
[I/FAL] RT-Thread Flash Abstraction Layer (V0.5.99) initialize success.
[I/fota] Partition[app] erase start:
[I/fota] Start to copy firmware from download to app partition:
#####
[I/fota] Upgrade success, total 170368 bytes.
rt-fo[I/fota] Copy firmware version Success!
[I/fota] Implement application now.

```

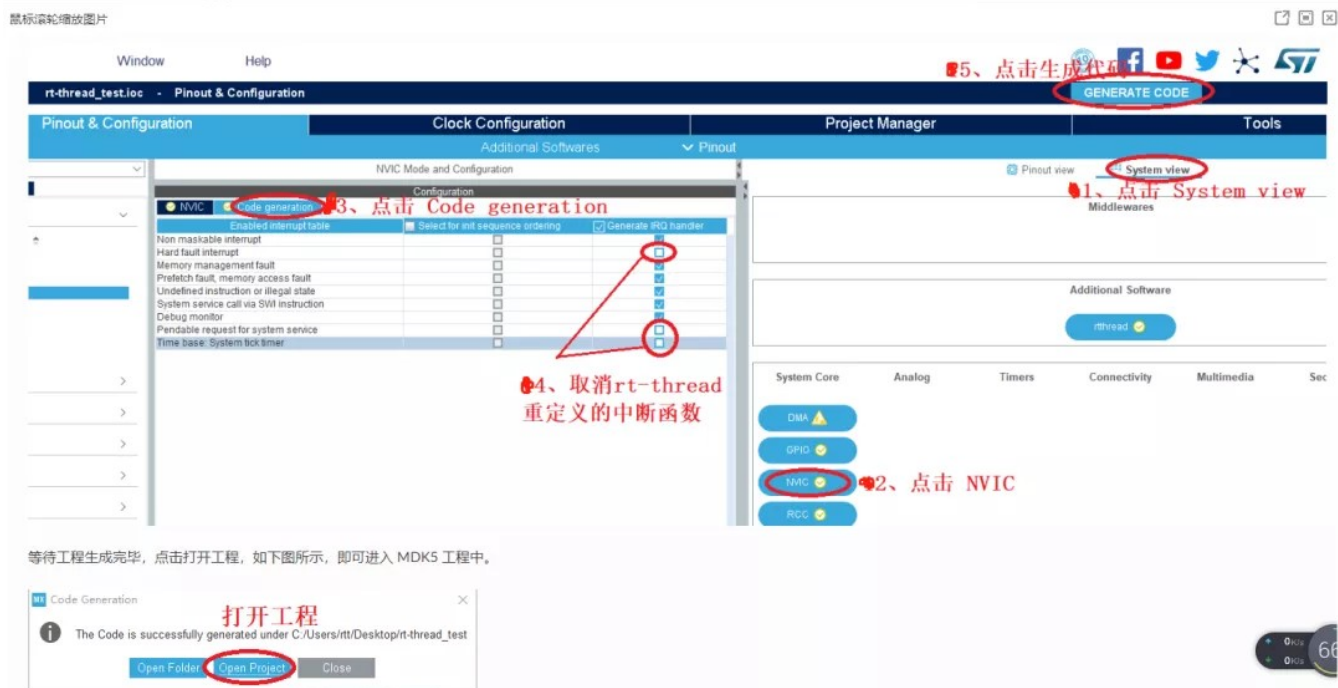
升级成功

2. ymodem_ota -p命令将固件下载到factory分区：

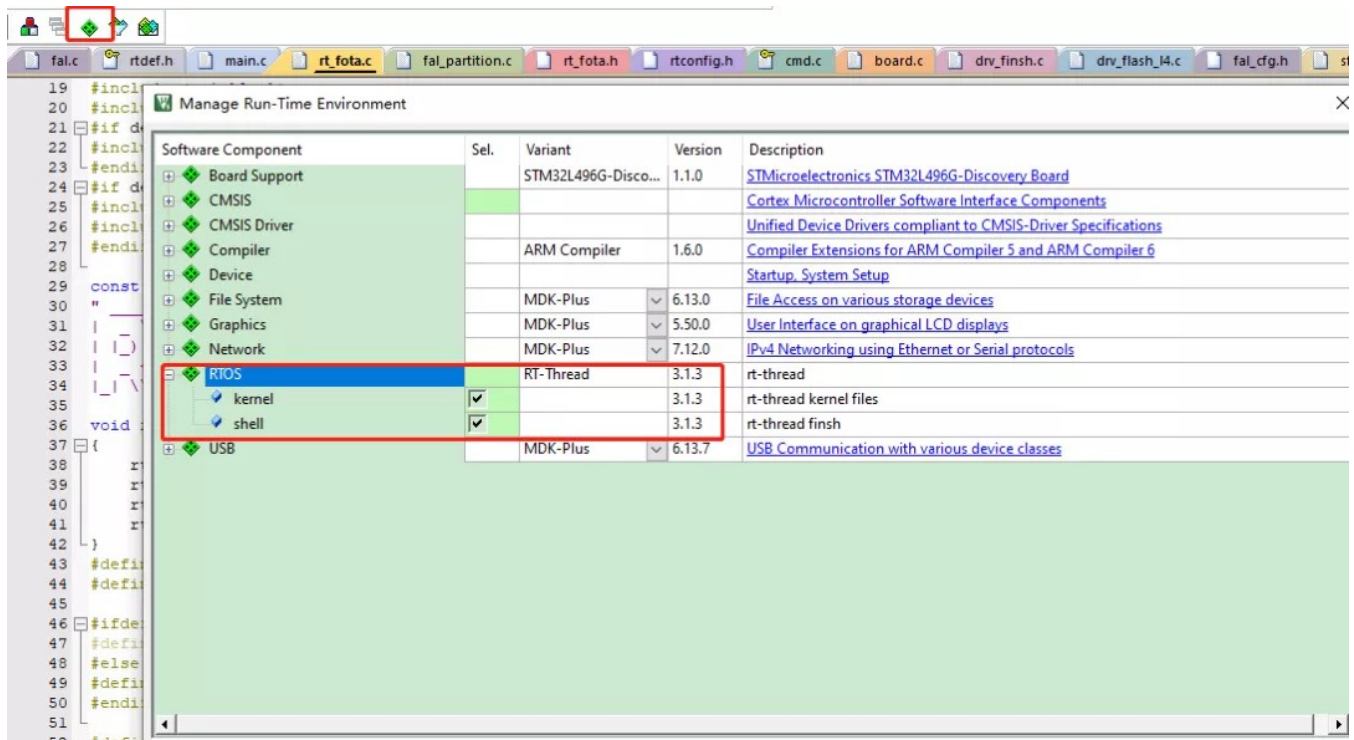
```
rt-fota />ymodem_ota -p factory
Warning: Ymodem has started! This operator will not recovery.
Please select the ota firmware file and use Ymodem to send.
CCCYmodem file_size:110256
[I/ymodem] Start erase. Size (110256)
Download firmware to flash success.
System now will restart...
```

9.如何移植

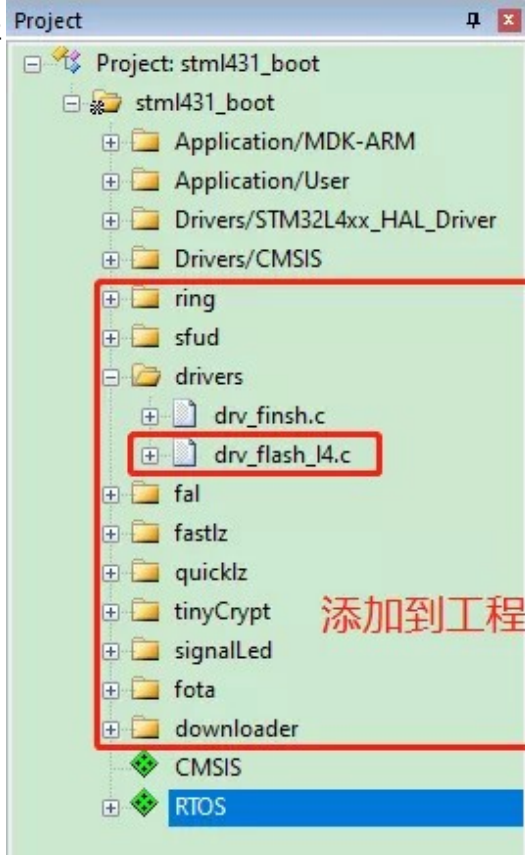
步骤1：通过STM32CubeMX生成工程：RT-Thread 操作系统重定义 HardFault_Handler、PendSV_Handler、SysTick_Handler 中断函数，为了避免重复定义的问题，在生成工程之前，需要在中断配置中，代码生成的选项中，取消选择三个中断函数（对应注释选项是 Hard fault interrupt, Pendable request, Time base :System tick timer），最后点击生成代码，具体操作如下图中步骤：



步骤2：基于 Keil MDK 移植 RT-Thread NanoMDK需要先获取 RT-Thread Nano pack 安装包并进行安装。RT-Thread Nano 离线安装包下载，下载结束后双击文件进行安装。RT-Thread Nano pack安装完成后，勾选 kernel和shell。



步骤3：将所有文件添加到工程



其中的drv_flash_l4.c要根

据自己的工程，选择对应的文件，其他的都不需要改动。

> stm1431_boot > rt_drv

步骤

名称

drv_finsh.c
drv_flash.h
drv_flash_f0.c
drv_flash_f1.c
drv_flash_f2.c
drv_flash_f4.c
drv_flash_f7.c
drv_flash_l4.c
drv_log.h

4：更改编译选项为AC6AC6的编译速度更快，尺寸更小。

Options for Target 'stm1431_boot'

×

Device Target Output Listing User C/C++ (AC6) Asm Linker Debug Utilities

STMicroelectronics STM32L431RCTx

Xtal (MHz): 80.0

Operating system: None

System Viewer File: STM32L4x1.svd

☐ Use Custom File

Code Generation

ARM Compiler: V6.13.1

☒ Use MicroLIB ☐ Big Endian

Floating Point Hardware: Single Precision

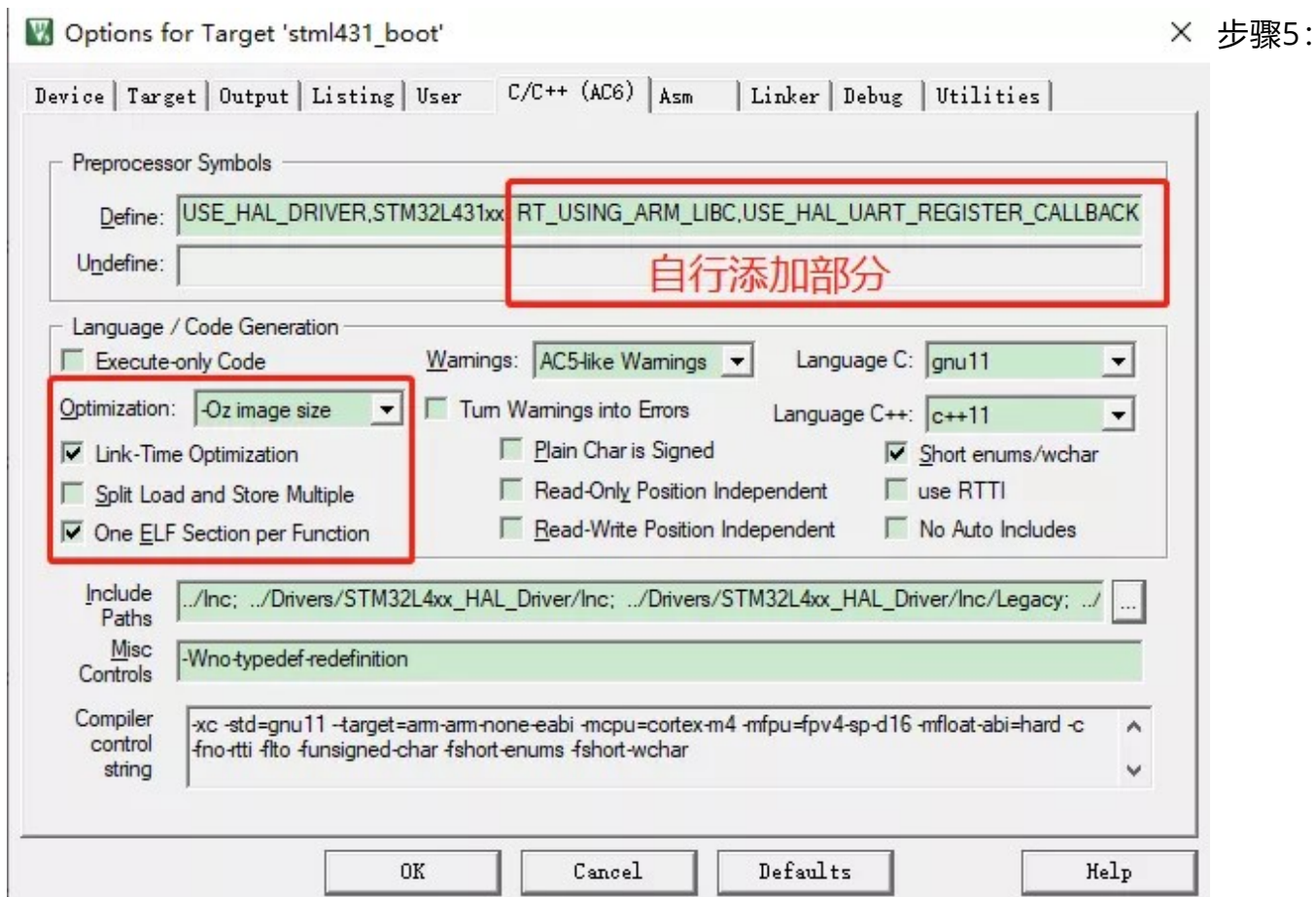
Read/Only Memory Areas

default	off-chip	Start	Size	Startup
<input type="checkbox"/>	ROM1:			<input type="radio"/>
<input type="checkbox"/>	ROM2:			<input type="radio"/>
<input type="checkbox"/>	ROM3:			<input type="radio"/>
	on-chip			
<input checked="" type="checkbox"/>	IROM1:	0x8000000	0x40000	<input checked="" type="radio"/>
<input type="checkbox"/>	IROM2:			<input type="radio"/>

Read/Write Memory Areas

default	off-chip	Start	Size	NoInit
<input type="checkbox"/>	RAM1:			<input type="checkbox"/>
<input type="checkbox"/>	RAM2:			<input type="checkbox"/>
<input type="checkbox"/>	RAM3:			<input type="checkbox"/>
	on-chip			
<input checked="" type="checkbox"/>	IRAM1:	0x20000000	0x10000	<input type="checkbox"/>
<input type="checkbox"/>	IRAM2:			<input type="checkbox"/>

OK Cancel Defaults Help



更改boart.c和rtconfig.hboart.c可以直接复制使用

```
/*
 * Copyright (c) 2006-2019, RT-Thread Development Team
 *
 * SPDX-License-Identifier: Apache-2.0
 *
 * Change Logs:
 * Date       Author    Notes
 * 2017-07-24  Tanek     the first version
 * 2018-11-12  Ernest Chen modify copyright
 */

#include <stdint.h>
#include <rthw.h>
#include <rtthread.h>
#include "main.h"
#define _SCB_BASE      (0xE000E010UL)
#define _SYSTICK_CTRL  (*(rt_uint32_t *)(_SCB_BASE + 0x0))
#define _SYSTICK_LOAD  (*(rt_uint32_t *)(_SCB_BASE + 0x4))
#define _SYSTICK_VAL   (*(rt_uint32_t *)(_SCB_BASE + 0x8))
#define _SYSTICK_CALIB (*(rt_uint32_t *)(_SCB_BASE + 0xC))
#define _SYSTICK_PRI   (*(rt_uint8_t *) (0xE000ED23UL))

// Updates the variable SystemCoreClock and must be called
// whenever the core clock is changed during program execution.
extern void SystemCoreClockUpdate(void);
extern void SystemClock_Config(void);
extern void MX_GPIO_Init();
// Holds the system core clock, which is the system clock
// frequency supplied to the SysTick timer and the processor
// core clock.
extern uint32_t SystemCoreClock;

static uint32_t _SysTick_Config(rt_uint32_t ticks)
{
    if ((ticks - 1) > 0xFFFFF)
    {
        return 1;
    }
}
```

```

}

_SYSTICK_LOAD = ticks - 1;
_SYSTICK_PRI = 0xFF;
_SYSTICK_VAL = 0;
_SYSTICK_CTRL = 0x07;

return 0;
}

#if defined(RT_USING_USER_MAIN) && defined(RT_USING_HEAP)

#if defined(__CC_ARM) || defined(__CLANG_ARM)
extern int Image$$RW_IRAM1$Z$$Limit;           // RW_IRAM1, 需与链接脚本中运
行时域名相对应
#define HEAP_BEGIN ((void *)&Image$$RW_IRAM1$Z$$Limit)
#endif

#define HEAP_END          STM32_SRAM1_END
#endif

/**
 * This function will initial your board.
 */
void rt_hw_board_init()
{
    HAL_Init();
    SystemClock_Config();
    /* System Clock Update */
    SystemCoreClockUpdate();

    /* System Tick Configuration */
    _SysTick_Config(SystemCoreClock / RT_TICK_PER_SECOND);
    MX_GPIO_Init();
    extern int uart_init(void);
    uart_init();
    /* Call components board initial (use INIT_BOARD_EXPORT()) */

```

```

#ifdef RT_USING_COMPONENTS_INIT
    rt_components_board_init();
#endif

extern void rt_fota_print_log(void);
rt_fota_print_log();

#if defined(RT_USING_USER_MAIN) && defined(RT_USING_HEAP)
    rt_system_heap_init((void *)HEAP_BEGIN, (void *)HEAP_END);
#endif
}

void SysTick_Handler(void)
{
    /* enter interrupt */
    rt_interrupt_enter();

    rt_tick_increase();

    /* leave interrupt */
    rt_interrupt_leave();
}

```

rtconfig.h配置文件:

```

160 // <<< end of configuration section >>>
161 /* On-chip Peripheral Drivers */
162 #define BSP_USING_ON_CHIP_FLASH
163 #define BSP_USING_LPUART1
164 #define BSP_USING_SPI2
165 /* Onboard Peripheral Drivers */
166 #define BSP_DATAFLASH_CS_GPIOX GPIOB
167 #define BSP_DATAFLASH_CS_GPIO_PIN GPIO_PIN_12
168
169 #define RT_FOTA_SIGNAL_LED
170 #define RT_FOTA_SIGNAL_LED_GPIOX GPIOB
171 #define RT_FOTA_SIGNAL_LED_GPIO_PIN GPIO_PIN_1
172 #define RT_FOTA_SIGNAL_LED_ON_LEVEL GPIO_PIN_RESET
173
174 #define RT_FOTA_DEFAULT_KEY
175 #define RT_FOTA_DEFAULT_KEY_CHK_TIME 10
176 #define RT_FOTA_DEFAULT_KEY_GPIOX GPIOA
177 #define RT_FOTA_DEFAULT_KEY_GPIO_PIN GPIO_PIN_7
178 #define RT_FOTA_DEFAULT_KEY_LEVEL GPIO_PIN_SET

```

根据cubmx的配置选择对应的串口和SPI

根据cubmx的配置选择对应的LED、KEY和SPI的片选引脚

```

183 #define FAL_PART_TABLE
184 {
185     {FAL_PART_MAGIC_WROD, "app", "onchip_flash", 64*1024, 192*1024, 0}, \
186     {FAL_PART_MAGIC_WROD, "ef", FAL_USING_NOR_FLASH_DEV_NAME, 0, 1024 * 1024, 0}, \
187     {FAL_PART_MAGIC_WROD, "download", FAL_USING_NOR_FLASH_DEV_NAME, 1024 * 1024, 512 * 1024, 0}, \
188     {FAL_PART_MAGIC_WROD, "factory", FAL_USING_NOR_FLASH_DEV_NAME, (1024 + 512) * 1024, 512 * 1024, 0}, \
189 }
190 #define FAL_USING_SFUD_PORT
191 #define FAL_USING_NOR_FLASH_DEV_NAME "w25q64"
192
193 #define PKG_USING_YMODEM_OTA
194 #define TINY_CRYPT_AES
195 #define PKG_USING_QUICKLZ
196 #define QLZ_COMPRESSION_LEVEL 3
197
198 /* RT-FOTA module define */
199 #define RT_FOTA_SW_VERSION "1.0.0"
200 /* FOTA application partition name */
201 #ifndef RT_FOTA_APP_PART_NAME
202 #define RT_FOTA_APP_PART_NAME "app"
203 #endif
204
205 /* FOTA download partition name */
206 #ifndef RT_FOTA_FM_PART_NAME
207 #define RT_FOTA_FM_PART_NAME "download"
208 #endif
209
210 /* FOTA default partition name */
211 #ifndef RT_FOTA_DF_PART_NAME
212 #define RT_FOTA_DF_PART_NAME "factory"
213 #endif
214
215 /* AES256 encryption algorithm option */
216 #define RT_FOTA_ALGO_AES_IV "0123456789ABCDEF"
217 #define RT_FOTA_ALGO_AES_KEY "0123456789ABCDEF0123456789ABCDEF"

```

FLASH分区信息，需要与APP部分的分区表一致

可裁剪项

分区的名字

加密部分需要与加密工具设置的一致

10.注意事项

1. 如果APP部分已经使用了Ymodem或者其他文件传输方式，bootloader可以不使能Ymodem
2. app也可以使用裸机开发，对系统无依赖，对于app只需要更改中断向量表部分，IAP可以由bootloader的Ymodem完成
3. 本项目示例代码中使用的硬件有Ipuart1、spi2 (W25Q64)、PA7(key)、PB1 (led)、PB12 (片选)
4. 本项目地址：<https://gitee.com/Aladdin-Wang/RT-FOTA-STM32L431>

本项目更多的教程请参考博客内容：

https://blog.csdn.net/sinat_31039061/article/details/106344081



扫码关注我們

CSDN博客：Aladdin Wang

微信号：wlk13298337053

[阅读原文](#)