OTA porting guide

1、flash 分布

目前支持的 flash 大小包括 1MB（UNO_81A 和 UNO_81AM）和 4MB（UNO_81C），不使用 OTA 和使用 OTA 时的分布情况不同。

不使用 OTA 时的 flash 分布：

```
0               4K                                      1/2/4M
-------------------------------------------------------------
| bootrom data | firmware | sys data | 3rd partner data |
-------------------------------------------------------------
        4K          n1 * 4K       4K          n2 * 4K
```

使用 OTA 时的 falsh 分布：

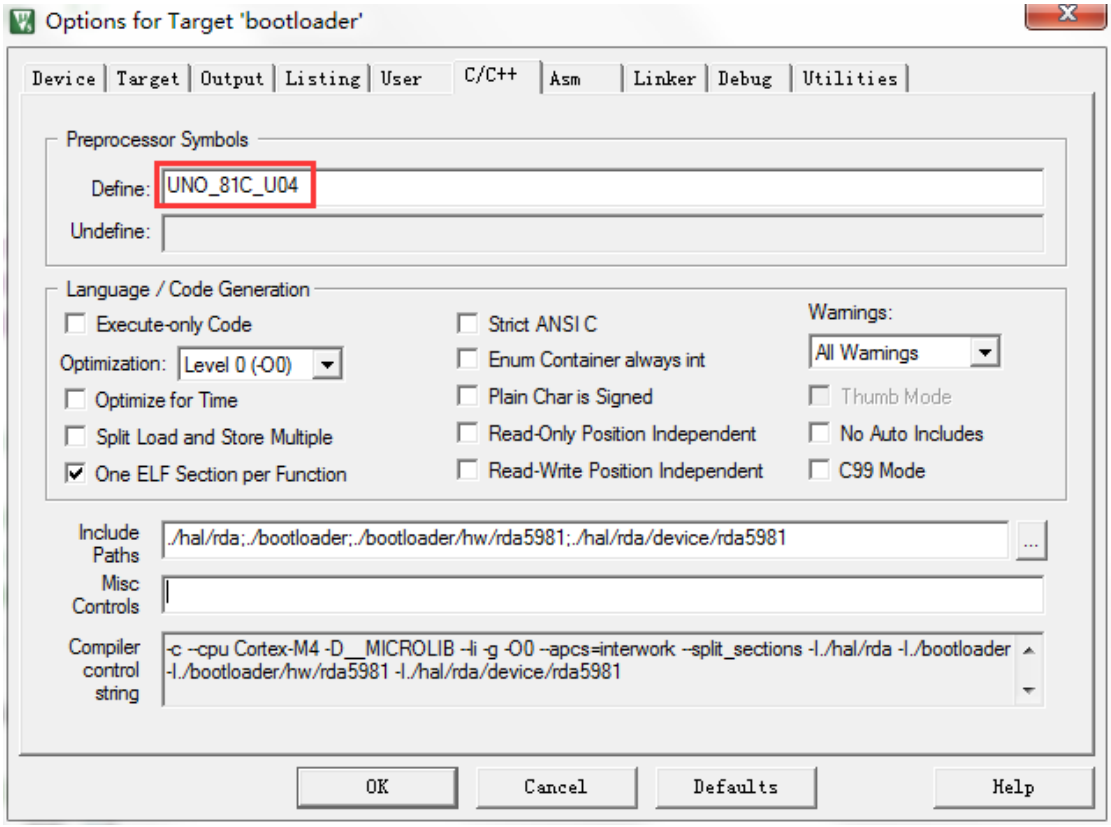
```
0               4K          12K             16K                                                 1/2/4M
-------------------------------------------------------------------------------------------------------------
| bootrom data | bootloader | firmware info | firmware | upgrade | sys data | 3rd partner data |
-------------------------------------------------------------------------------------------------------------
        4K          8K              4K          n1 * 4K    n2 * 4K      4K          n3 * 4K
```

flash 中可能存在如下分区，注意每个分区的大小都是 4KB 对齐的。
bootrom data：bootrom 所使用到的数据，用户不可读写。
firmware：firmware 代码区。
firmware info：firmware 的描述信息。
upgrade：upgrade 代码区。
bootloader：执行升级操作的 bootloader。
sys data：存放 MAC 地址、路由器的 SSID/PASSWORD 等信息。
3rd partner data：只用少部分客户在使用，不建议其他客户使用。
系统启动后，会先跳转到 4K 处开始执行。如果没有使用 OTA 功能，则 4K 处放的就是 firmware。如果使用了 OTA 功能，则 4K 处放的是 bootloader，bootloader 会首先检查 firmware 区和 upgrade 区的版本，如果 upgrade 的版本比 firmware 的版本更新，则 bootloader 会将 upgrade 区的数据复制到 firmware，同时更新 firmware info，再跳转到 firmware（16K）处开始执行。否则直接跳转到 firmware（16K）处开始执行。每次进行 OTA 升级时，都是将最新的 firmware 烧录到 upgrade 区。

除了 bootrom data、bootloader、firmware info 和 sys data 区的大小是确定的，其他区的大小用户是可以自己设定的。
注意：目前芯片分为 RDA5981A、RDA5981AM 和 RDA5981C 三款，其中前 A、AM 的 flash 大小为 1MB，C 的 flash 大小为 4MB。同时目前还存在 U02 和 U04 的版本区别，目前基本上使用的都是 U04 芯片。具体使用哪款芯片和版本在程序中由宏控制，这个宏和在编译 mbed 程序时的 target 相同，分为 UNO_81A_U02、UNO_81A_U04、UNO_81AM_U02、UNO_81AM_U04、UNO_81C_U02、UNO_81C_U04。
本文假设用户使用的是 RDA5981C，U04 版本，故宏为 UNO_81C_U04。firmware 区的大小

假设为 0x001F4000（2000KB），与 mbed 中对应的 scatter file 中的 RDA_CODE_SIZE 相同。

对于使用 ARMCC 的用户，可以在打开对应的 keil 工程后，在“Options for Target”中更改。



对于使用 GCC 的用户，可以在 makefile 中更改，将 makefile 中的 DEFS += -DUNO_81C_U04 改成对应的芯片版本。

2、初次使用 OTA

2.1、生成 bootloader.bin

用 Keil v5 打开 bootloader 这个工程，并编译生成 bootloader.bin。bootloader 的运行地址必须设置到 0x1000，在此工程中已经默认使用此值，用户不需要修改。注意首先有两个宏需要根据你的实际情况修改，分别为 bootloader_config.h 中的 COPY_MODE_FW_MAX_SIZE 和 COPY_MODE_UPGRADE_ADDR。COPY_MODE_FW_MAX_SIZE 为 firmware 的最大大小，本文中设置成了 2000KB，故 COPY_MODE_FW_MAX_SIZE 为 0x001F4000。COPY_MODE_UPGRADE_ADDR 为 upgrade 区的起始地址，flash 在 memory map 中的起始地址为 0x18000000，故 COPY_MODE_UPGRADE_ADDR 为 4K + 8K + 4K + 2000K + 0x18000000 = 0x181F8000。

对于使用 GCC 的客户，请在每次 make 之前先 make clean 一下。

2.2、生成 mbed.bin

编译 mbed，生成 mbed.bin，mbed 的运行地址要和 bootloader 的跳转地址相对应。即要把 mbed 的运行地址设置为 16K 处，需要修改以下三个文件。

对于使用 ARMCC 的用户：

hal\targets\cmsis\TARGET_RDA\TARGET_UNO_91H\TOOLCHAIN_ARM_STD\TARGET_UNO_81C\RDA5981C.sct 中的
#if (0 == RDA_PARTITION_INDEX)
#define RDA_PADDR_OFST (0x00001000)
修改为
#if (0 == RDA_PARTITION_INDEX)
#define RDA_PADDR_OFST (0x00004000)

对于使用 GCC 的用户：
hal\targets\cmsis\TARGET_RDA\TARGET_UNO_91H\TOOLCHAIN_GCC_ARM\TARGET_UNO_81C\RDA5981C.ld 中的
FLASH (rx) : ORIGIN = 0x18001000 , LENGTH = 2000K
修改为
FLASH (rx) : ORIGIN = 0x18004000 , LENGTH =2000K

对于无论使用 ARMCC 还是 GCC 的用户：
hal\targets\cmsis\TARGET_RDA\TARGET_UNO_91H\RDA5991H.h 中的
#if (0 == RDA_PARTITION_INDEX)
#define RDA_PADDR_OFST (0x00001000UL)
修改为
#if (0 == RDA_PARTITION_INDEX)
#define RDA_PADDR_OFST (0x00004000UL)

或者用户也可以通过自己定义的宏来控制 RDA_PADDR_OFST（ORIGIN）与 RDA_CODE_SIZE（LENGTH）的值。
对于使用 ARMCC 的用户，sct 文件里的宏 RDA_CODE_SIZE 定义了 firmware 的最大大小，与 COPY_MODE_FW_MAX_SIZE 对应，这里为 2000KB，客户可以自己定制。对于使用 GCC 的用户，则是通过 ld 文件中 FLASH 那一行的 LENGTH 来控制的，这里同样为 2000K。

注意，使用 RDA5981A 的用户应该去修改对应的 RDA5981A.sct 和 RDA5981A.ld，使用 RDA5981AM 的用户应该去修改对应的 RDA5981AM.sct 和 RDA5981AM.ld。

2.3、给 mbed.bin 添加 image header
使用 bootloader\jlink\image_pack_ota.py 给 2.2 生成的 mbed.bin 添加 image header，image header 的内容为

```
struct image_header {
    uint16_t magic;
    uint8_t  encrypt_algo;
    uint8_t  resv[1];
    uint8_t  version[VERSION_SZ];
    uint32_t crc32;
    uint32_t size;
};
```

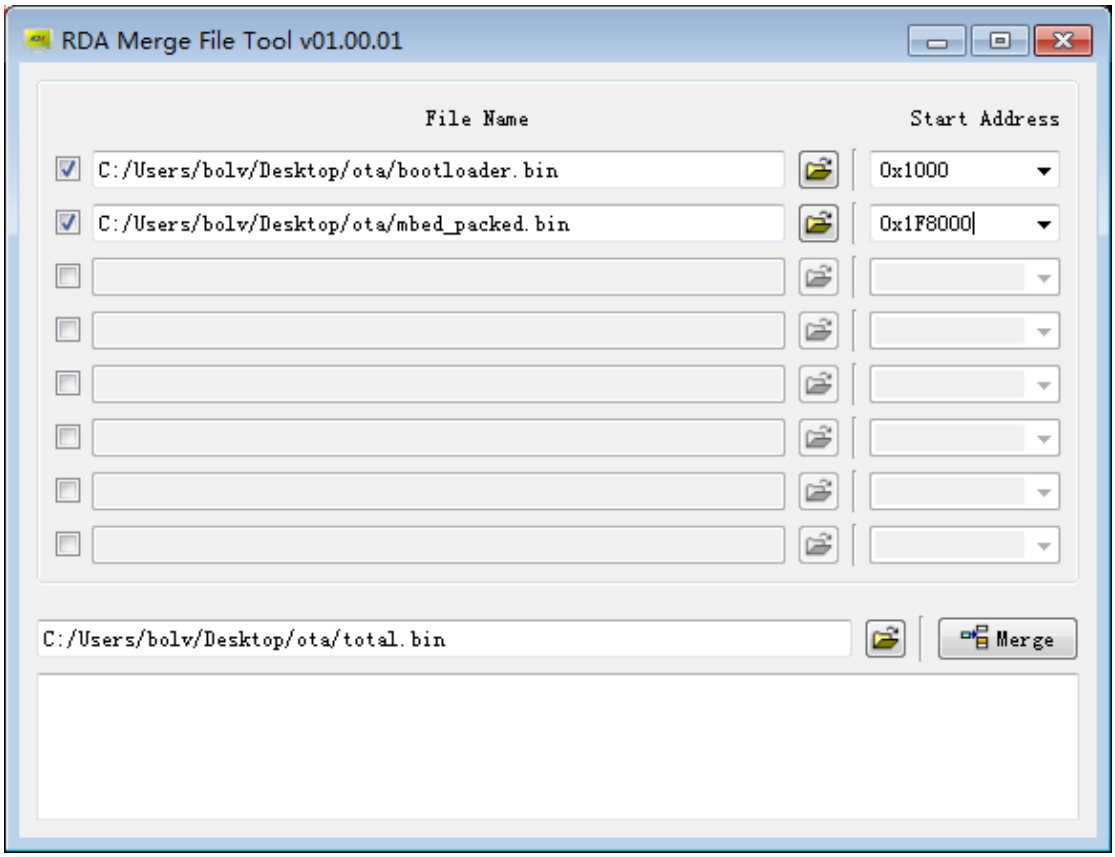
其中除了 version 之外，其它的信息都是自动生成的。执行 image_pack_ota.py 时需要传入两

个参数，文件名和版本号，如果要给 mbed.bin 加上版本号为 1.00.00，则执行
python image_pack_ota.py mbed.bin 1.00.00
生成了带 image head 的 mbed_packed.bin。

```
bolv@BJLP000815 MINGW64 /e/RDA/rda5981_bootloader_v1.2/jlink
$ python image-pack.py mbed.bin 1.00.00
firmware: mbed.bin
    size: 26828
   version: 1.00.00
   crc32: 40b63318
```

2.4、合并 bootloader 与 firmware

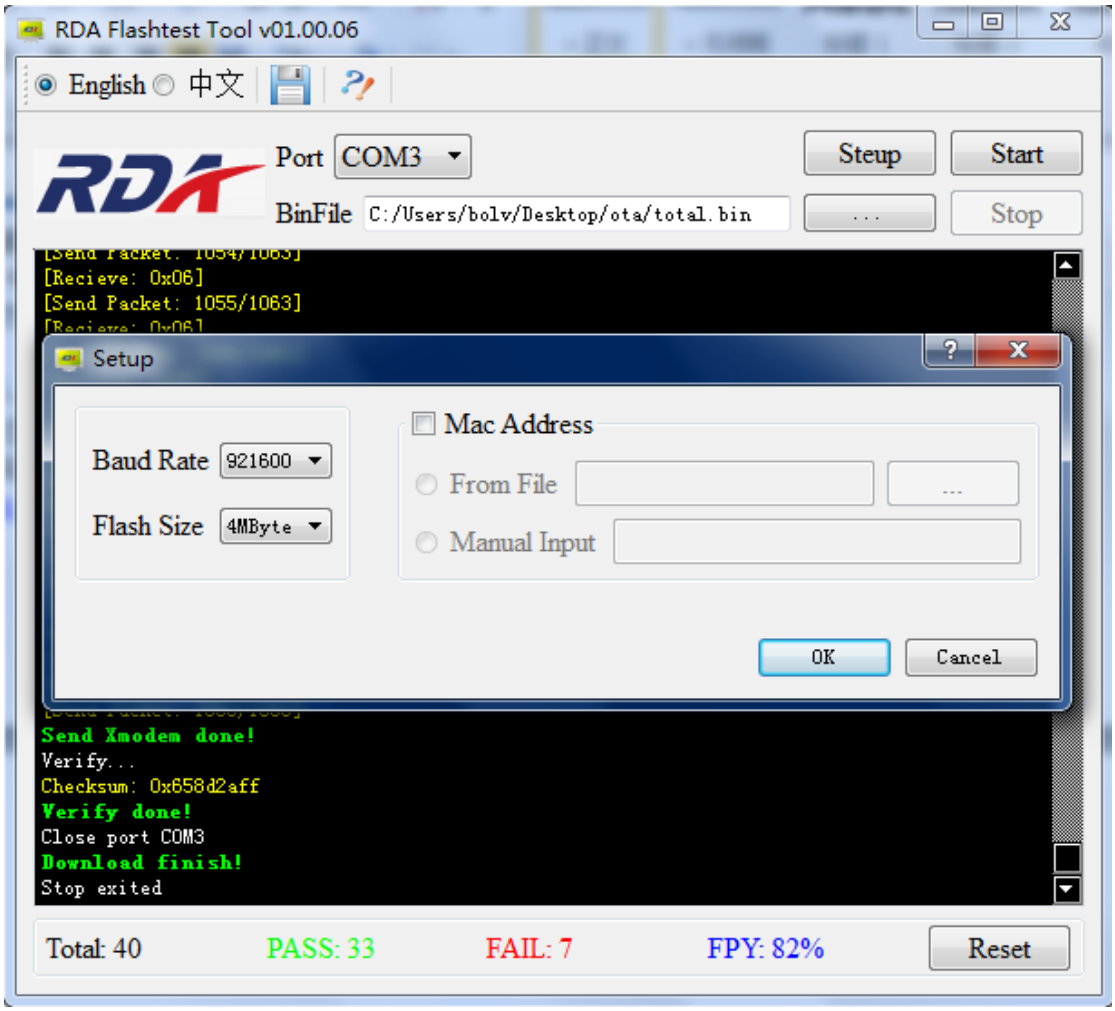
将 2.2 里生成的 bootloader.bin 与 2.3 里生成的 mbed_packed.bin 通过“RDA5981 bin 文件合并工具”合并，生成 total.bin，工具下载地址为：
http://bbs.rdamicro.com/forum.php?mod=viewthread&tid=201



注意，图中地址的设置，这里要设置成相对于 flash 开始处的相对地址，bootloader.bin 要设置在 4K 处。mbed_packed.bin 要烧录在 upgrade 区，故设置的地址与 upgrade 区的地址相同，本文为 0x1F8000（2000K + 16K）处，即（COPY_MODE_UPGRADE_ADDR – 0x18000000）。

2.5、烧录

通过“RDA5981 单口下载工具”将 2.4 中生成 total.bin 烧录到 RDA5981 的 flash，工具下载地址：
http://bbs.rdamicro.com/forum.php?mod=viewthread&tid=108



注意要在 Setup 中把 Flash Size 设置成对应的大小。

2.6、重启

重启开发板，bootrom 自动跳转到 4K 处执行 bootloader，bootloader 经过比较后发现 upgrade 区的版本比较新，将 upgrade 区的内容更新到 firmware 区，再跳转到 firmware 区开始执行。

3、后续使用 OTA

在初次使用 OTA 之后，就可以不使用下载工具，而是使用 RDA5981 提供的 OTA 升级接口进行升级。但是首先要通过 2.3 的方法给要给要升级的 bin 加上 image header。

参考 SDK 里面的 ota case

http://bbs.rdamicro.com/forum.php?mod=viewthread&tid=132&extra=page%3D1

及 RDA5981 flash 分区信息里面提供的 flash 操作接口

http://bbs.rdamicro.com/forum.php?mod=viewthread&tid=284&extra=page%3D1

首先要调用 int rda5981_write_partition_start(u32 addr, u32 img_len)这个接口。

addr 为 upgrade 区的地址，必须 4K 对齐，地址为在 memory map 中的地址，故此处应该为 0x181F8000。

image_len 为 bin 的长度，必须为 4K 对齐。假如你实际 bin 的长度为 39K，那么这里传入的

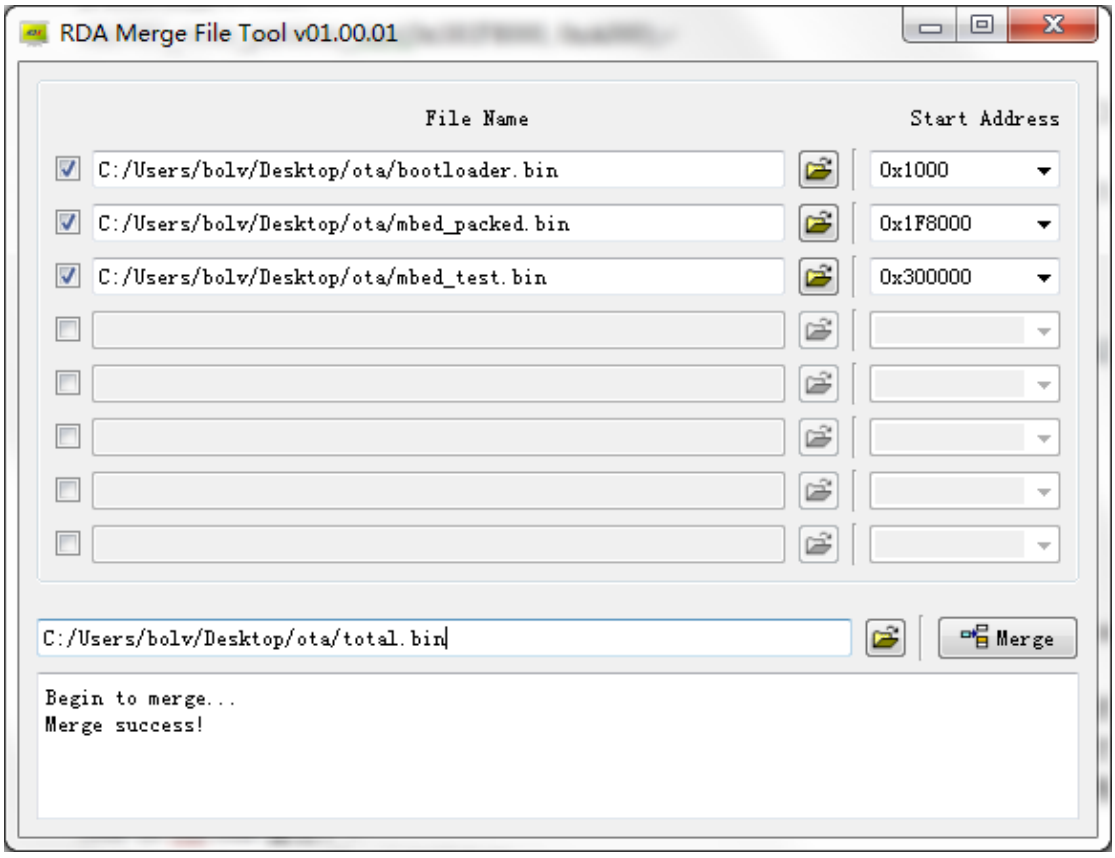
参数要设置为 40K。
rda5981_write_partition_start(0x181F8000, 0xA000);
接下来循环调用 int rda5981_write_partition(u32 offset, const u8 *buf, u32 len)这个接口往 upgrade 区写入数据直至所有的 bin 写入 upgrade 区。
offset 为相对于 upgrade 区开始处的偏移地址，假如要往 0x181F8000 处写入数据，则 offset 应该为 0。
buf 为数据指针。
len 为数据长度，必须 1K 对齐，最长 4K。
注意，如前所述，如果 bin 的实际大小是 39K，由于 4K 对齐原因，我们仍然要写入 40K 数据，只是把最后 1K 数据全部写成 0xFF 即可。
最后调用 int rda5981_write_partition_end()结束 OTA 升级，成功后返回 0。

4、在 OTA 区存放测试 code

部分用户可能希望在 OTA 区存放测试 code，烧录时将正式 code 和测试 code 同时烧录到 flash。在运行过程中可以通过 AT 命令在测试 code 和正式 code 之间进行切换。
对于这种用法，首先你要确定你的测试 code 在 flash 中的存放位置，假设我们要存放在 0x300000（3072KB）处，则要按照 2.2 的步骤，先把对应的 RDA_PADDR_OFST 都改为 0x300000，再编译出 mbed_test.bin，存放位置要根据你自己的分区灵活改变。注意，mbed_test.bin 不需要添加 image header。
你可以把正式 code 先放在 upgrade 区，也可以把正式 code 放在 firmware 区。

4.1、把正式 code 放在 upgrade 区

编译出 mbed_test.bin 后，将 bootloader.bin、mbed_packed.bin 与 mbed_test.bin 合并并烧录。



重启开发板后，bootloader 首先仍然要执行 ota 升级，将正式 code 从 upgrade 区复制到 firmware 区，然后再跳转到 firmware 区开始执行正式 code。如果要跳转到测试 code，则需要你在正式 code 里面集成我们的 AT 指令。参考 uartwifi 这个 case 下有个 AT 指令，AT+BOOTADDR=<addr>，如果要跳转到测试 code，则在 console 输入 AT+BOOTADDR=0x18300000 即可。如果要从测试 code 跳转到正式 code，同样需要在测试 code 里面集成这个 AT 指令，然后在 console 输入 AT+BOOTADDR=0x18004000 即可。注意，addr 为绝对地址，对于正式 code，这个是 0x18004000，对于测试 code 则是客户自己指定的。

4.2、把正式 code 放在 firmware 区

如果把正式 code 放在 firmware 区，可以在 firmware 的 firmware info 中指定你要启动的地址 bootaddr，bootloader 会检查这个地址进行跳转。如果设置为 0，则这个地址是无效的，bootloader 会先进行 ota 升级检查，再确定跳转地址。

首先在执行完步骤 2.2 后，使用 bootloader\jlink\image_pack_firmware.py 给 mbed.bin 增加 firmware info，firmware info 的内容为

```
struct image_header {
    uint16_t magic;
    uint8_t  version[VERSION_SZ];
    uint32_t addr;
    uint32_t size;
    uint32_t crc32;
    uint32_t bootaddr;
    uint32_t bootmagic;
};
```

与 2.3 类似，这里需要多传入一个参数 bootaddr。

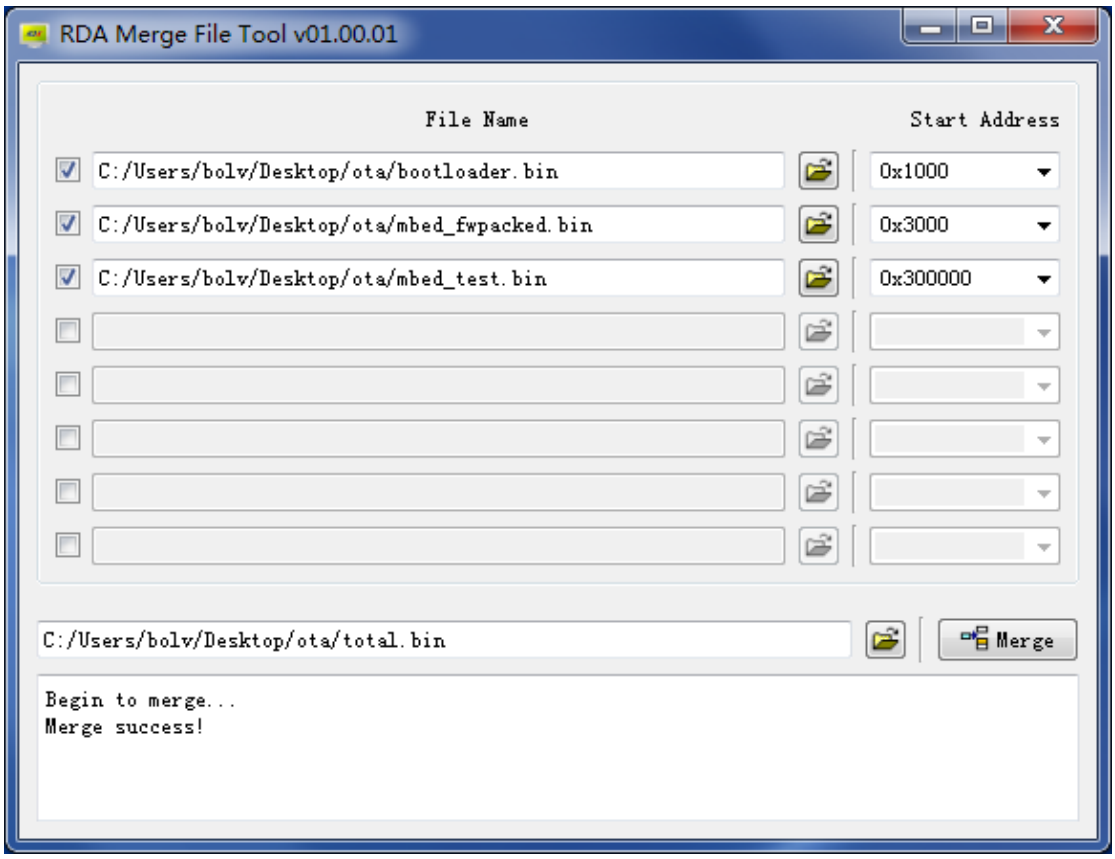
注意，在 bootloader 在向 bootaddr 执行完跳转后，flash 中保存的 bootaddr 就被清零了。如果你想在 code 启动后再切换 code，请参考 4.1 中的 AT+BOOTADDR。

假如首次启动要从测试 code 开始执行，则传入 0x18300000。

python image_pack_firmware.py mbed.bin 1.00.00 0x18300000

生成了带 firmware info 的 mbed_fwpacked.bin，版本号为 1.00.00，bootaddr 为 0x18300000。

将 bootloader.bin、mbed_fwpacked.bin、mbed_test.bin 合并并烧录。



注意这里地址的变化，因为 firmware 加上了 4K 长的 firmware info，故这里的偏移地址设置为 0x3000。但是如果要跳转到 firmware，发送的 AT 依然是 AT+BOOTADDR=0x18004000。