本资料仅供内部使用！

STM32 μC/OS-II 稳定性测试

子模块 IAP 实现

2013 年 6 月 24 日

修改记录

制定日期	生效日期	制定 / 修订 内容摘要	页数	版本	拟稿	审查	批准
2013.06.24		μC/OS-II 稳定性测试子模块 IAP	9	0.01	朱正晶		

目 录

1 简介

本节将简要说明手册的目的、范围。

1.1 手册目的

本手册的目的在于说明 STM32F207 IAP（在应用编程），本子系统为 STM32 稳定性测试的一部分。

1.2 手册范围

本手册首先简要地介绍 IAP 方法的优点，然后说明 IAP 具体的实现方法。

本手册的使用者包括：

程序编写、维护者

…

2 IAP 简介

IAP（In Application Programming）即在应用编程。IAP 是用户自己的程序在运行过程中对 User Flash 的部分区域进行烧写，目的是为了在产品发布后可以方便地通过预留的通信口对产品中的固件程序进行更新升级。

3 IAP 实现

通常实现 IAP 功能时，即用户程序运行中作自身的更新操作，需要在设计固件程序时编写两个项目代码，第一个项目程序不执行正常的功能操作，而只是通过某种通信方式(如 USB、USART)接收程序或数据，执行对第二部分代码的更新；第二个项目代码才是真正的功能代码。这两部分项目代码都同时烧录在 User Flash 中，当芯片上电后，首先是第一个项目代码开始运行，它作如下操作：

1）检查是否需要对第二部分代码进行更新

2）如果不需要更新则转到 4）

3）执行更新操作

4）跳转到第二部分代码执行

第一部分代码必须通过其它手段，如 JTAG 或 ISP 烧入；第二部分代码可以使用第一部分代码 IAP 功能烧入，也可以和第一部分代码一起烧入，以后需要程序更新是再通过第一部分 IAP 代码更新。

我们将第一个项目代码称之为 Bootloader 程序，第二个项目代码称之为 APP 程序，他们存放在 STM32 FLASH 的不同地址范围，一般从最低地址区开始存放 Bootloader，紧跟其后的就是 APP 程序（注意，如果 FLASH 容量足够，是可以设计很多 APP 程序的，这里我们只讨论一个 APP 程序的情况）。这样我们就是要实现两个程序：Bootloader 和 APP。

我们先来看看 STM32 正常的程序运行流程，如图 3-1 所示：

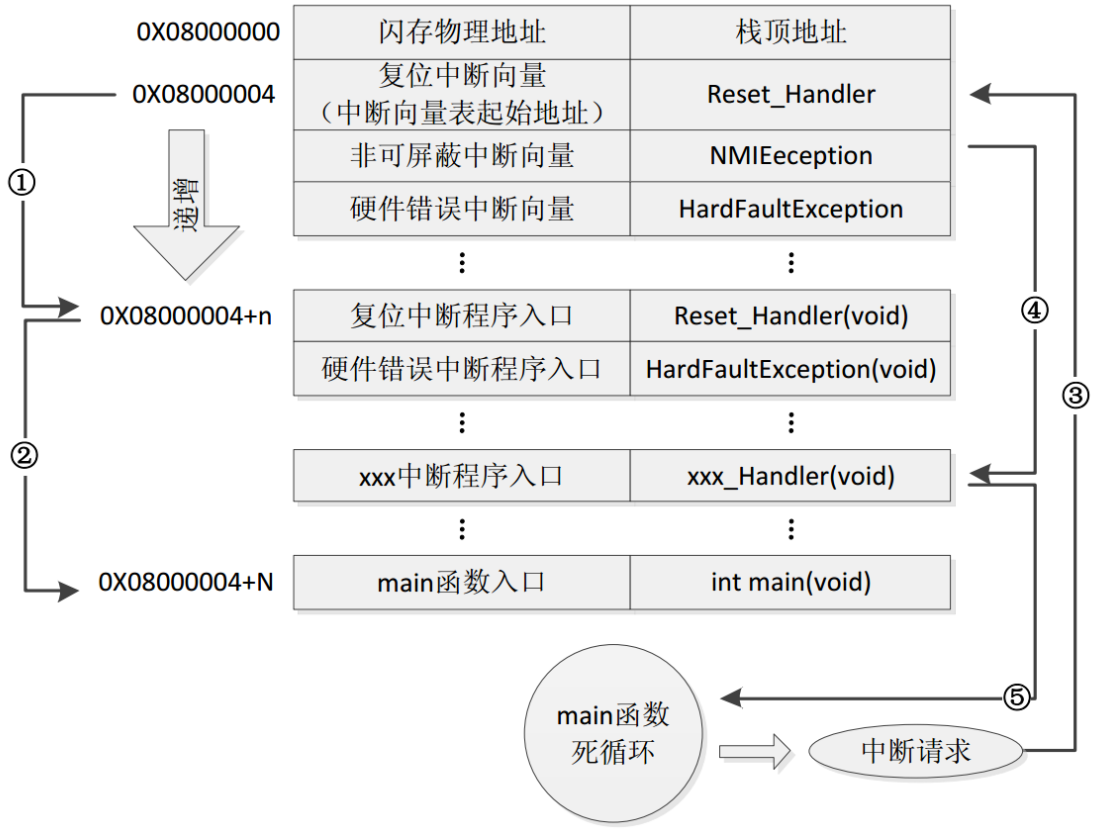

图 3-1 STM32 正常运行流程图

STM32 的内部闪存（FLASH）地址起始于 0x08000000，一般情况下，程序文件就从此地址开始写入。此外 STM32 是基于 Cortex-M3 内核的微控制器，其内部通过一张“中断向量表”来响应中断，程序启动后，将首先从“中断向量表”取出复位中断向量执行复位中断程序完成启动，而这张“中断向量表”的起始地址是 0x08000004，当中断来临，STM32 的内部硬件机制亦会自动将 PC 指针定位到“中断向量表”处，并根据中断源取出对应的中断向量执行中断服务程序。

在图 3-1 中，STM32 在复位后，先从 0X08000004 地址取出复位中断向量的地址，并跳转到复位中断服务程序，如图标号①所示；在复位中断服务程序执行完之后，会跳转到我们的 main 函数，如图标号②所示；而我们的 main 函数一般都是一个死循环，在 main 函数执行过程中，如果收到中断请求（发生重中断），此时 STM32 强制将 PC 指针指回中断向量表处，如图标号③所示；然后，根据中断源进入相应的中断服务程序，如图标号④所示；在执行完中断服务程序以后，程序再次返回 main 函数执行，如图标号⑤所示。

当加入 IAP 程序之后，程序运行流程如图 3-2 所示：

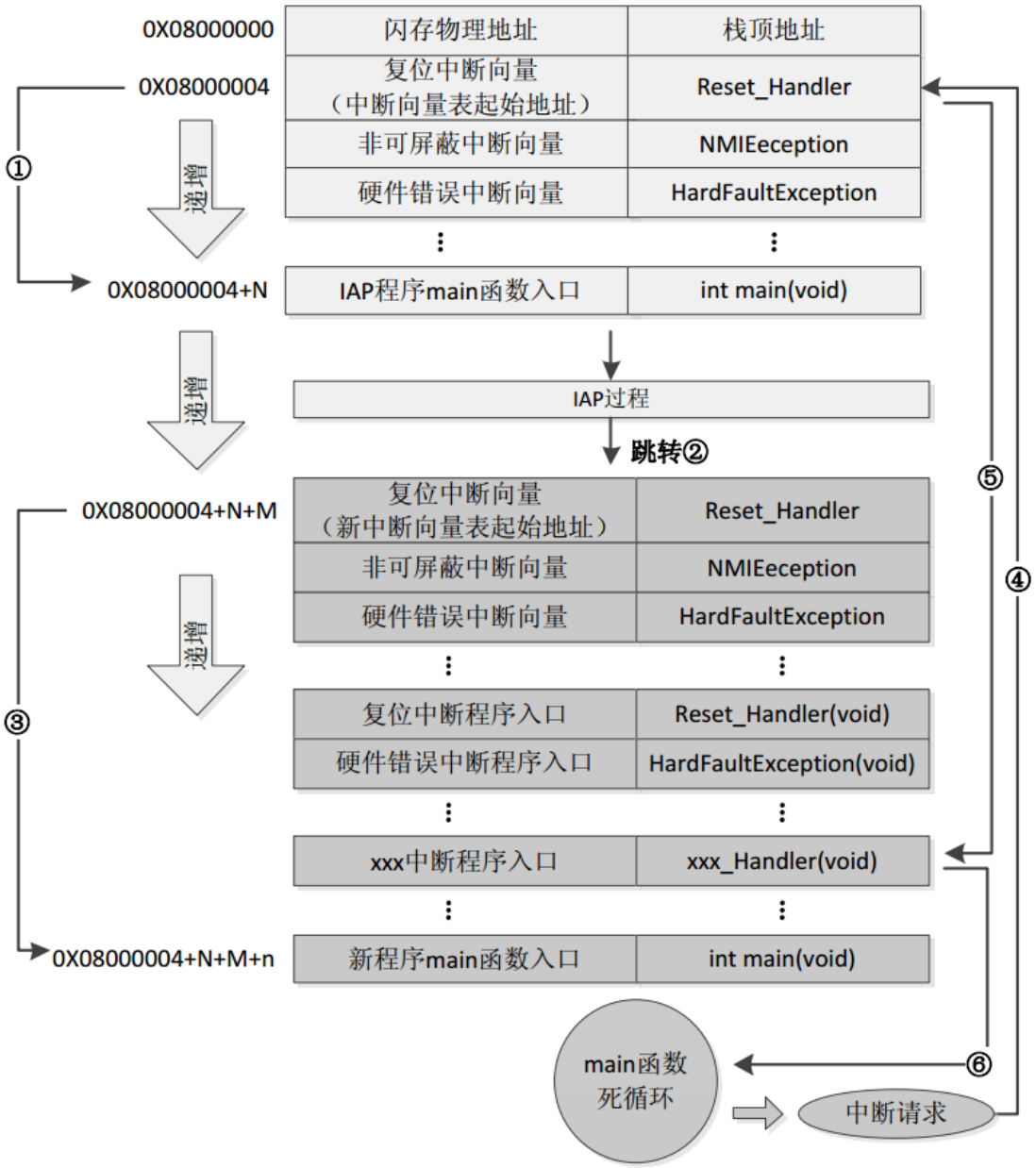

图 3-2 加入 IAP 之后程序运行流程图

在图 3-2 所示流程中，STM32 复位后，还是从 0X08000004 地址取出复位中断向量的地址，并跳转到复位中断服务程序，在运行完复位中断服务程序之后跳转到 IAP 的 main 函数，如图标号①所示，此部分同图 2-1 一样；在执行完 IAP 以后（即将新的 APP 代码写入 STM32 的 FLASH，灰底部分。新程序的复位中断向量起始地址为 0X08000004+N+M），跳转至新写入程序的复位向量表，取出新程序的复位中断向量的地址，并跳转执行新程序的复位中断服务程序，随后跳转至新程序的 main 函数，如图标号②和③所示，同样 main 函数为一个死循环，并且注意到此时 STM32 的 FLASH，在不同位置上，共有两个中断向量表。

在 main 函数执行过程中，如果 CPU 得到一个中断请求，PC 指针仍强制跳转到地址 0X08000004 中断向量表处，而不是新程序的中断向量表，如图标号④所示；程序再根据我们设置的中断向量表偏移量，跳转到对应中断源新的中断服务程序中，如图标号⑤所示；在执行完中断服务程序后，程序返回 main 函数继续运行，如图标号⑥所示。

通过以上两个过程的分析，我们知道 IAP 程序必须满足两个要求：

1) 新程序必须在 IAP 程序之后的某个偏移量为 x 的地址开始；
2) 必须将新程序的中断向量表相应的移动，移动的偏移量为 x；

3.1 IAP 在 STM32F207 中实现

IAP 流程图如图 3-3 所示。

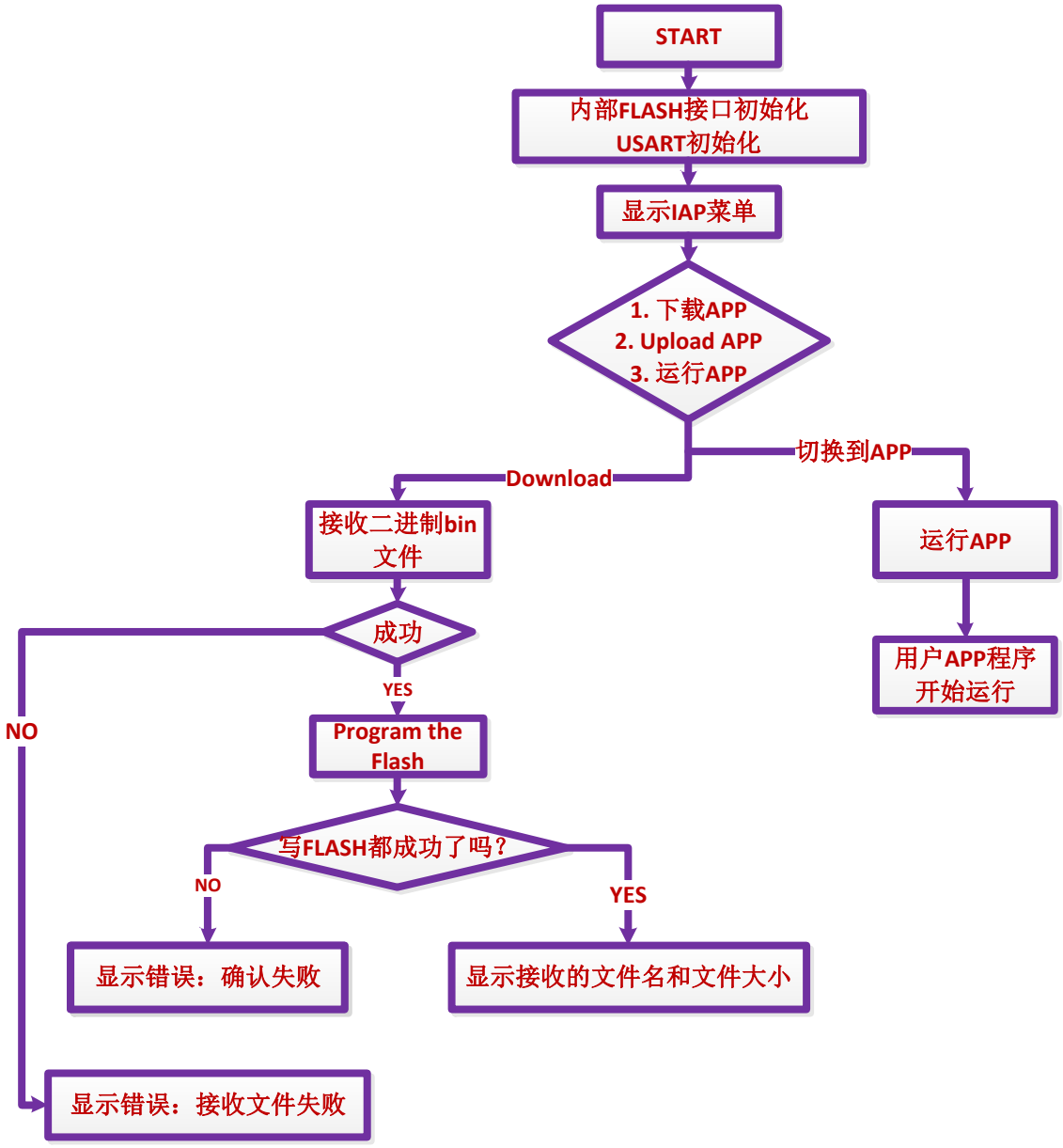

图 3-3 IAP 程序流程

IAP 文件下载协议为 Ymodem，使用超级终端即可发送。网上也有一个很经典的串口库，里面有 Ymodem 协议的接口，以后有需要可以很容易实现自己的下载程序。具体的可以参考文档《STM32 直流电机控制说明书.pdf》。

图 3-4 为 IAP 程序运行截图。

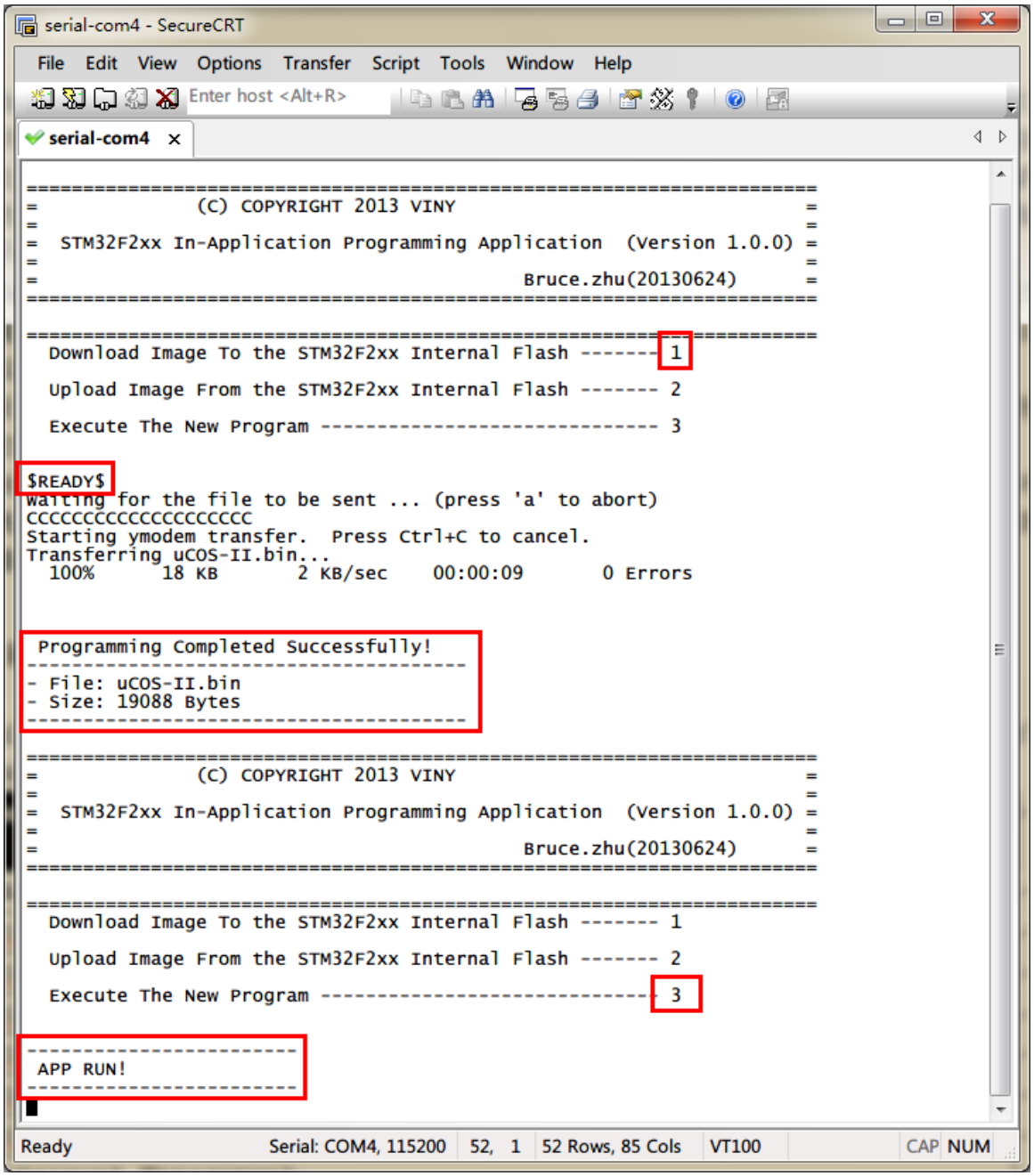

图 3-4 程序运行效果

3.2 APP 程序实现

打开我们的工程，点击 Options for Target 选项卡，如图 3-5 所示：

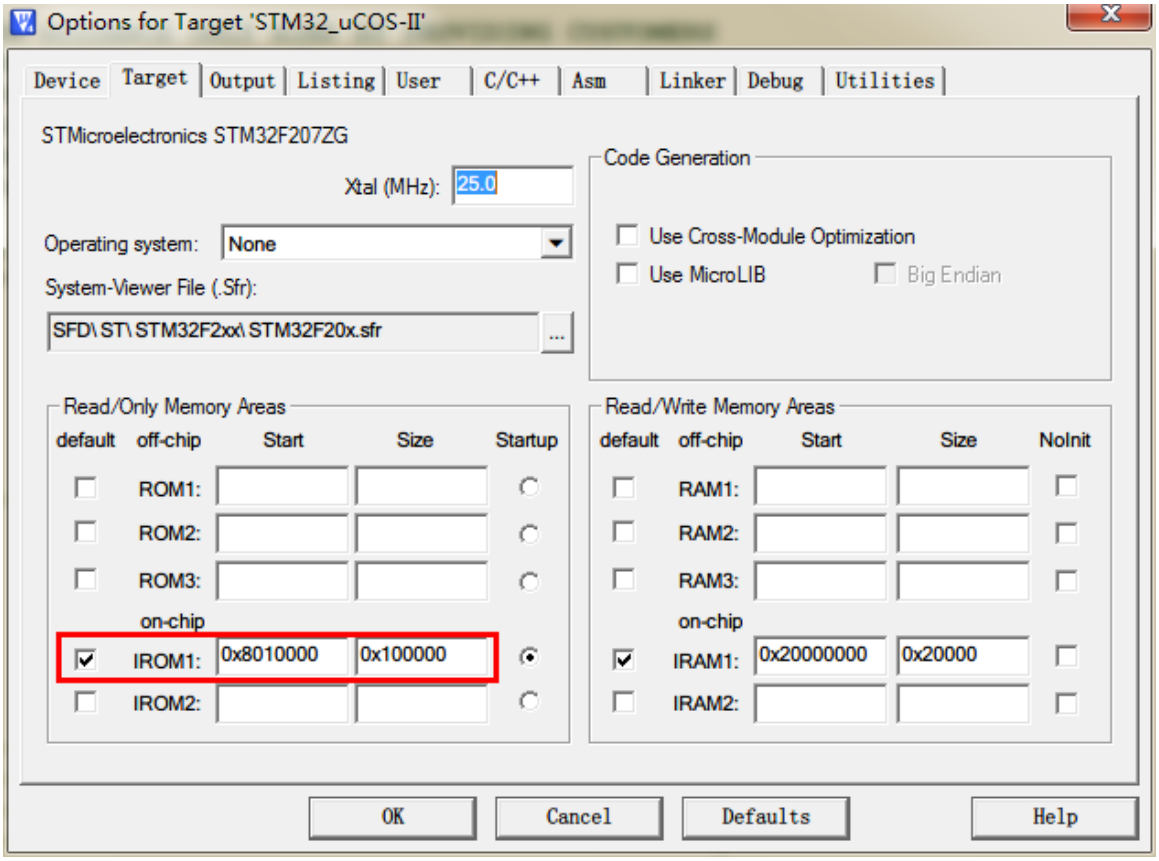

图 3-5 FLASH Target 选项卡设置

默认的条件下，图中 IROM1 的起始地址（Start）一般为 0x08000000，大小（Size）为 0x100000，即从 0x08000000 开始的 1MB 空间为我们的程序存储（因为我们的 STM32F207 的 FLASH 大小是 1MB）。而图中，我们设置起始地址（Start）为 0x08010000，即偏移量为 0x10000 （64K 字节），因而，留给 APP 用的 FLASH 空间（Size）只有 0x100000- 0x10000=0xF0000（960K 字节）大小了。设置好 Start 和 Szie，就完成 APP 程序的起始地址设置。

在系统启动的时候，会首先调用 systemInit 函数（在文件 system_stm32f207xx.c）初始化时钟系统，同时 systemInit 还完成了中断向量表的设置，我们可以打开 systemInit 函数，看看函数体的结尾处有这样几行代码：

```
00233:   /* Configure the System clock source, PLL Multiplier and Divider factors,
00234:     AHB/APBx prescalers and Flash settings ----------------------------------*/
00235:   SetSysClock();
00236:
00237:   /* Configure the Vector Table location add offset address ------------------*/
00238: #ifdef VECT_TAB_SRAM
00239:   SCB->VTOR = SRAM_BASE | VECT_TAB_OFFSET; /* Vector Table Relocation in Internal SRAM */
00240: #else
00241:   SCB->VTOR = FLASH_BASE | VECT_TAB_OFFSET; /* Vector Table Relocation in Internal FLASH */
00242: #endif
00243: } ? end SystemInit ?
00244:
```

原先 VECT_TEB_OFFSET 宏定义为 0，这里我们需要改成 0x10000，即 APP FLASH 的偏移量。如图 3-6 所示。

```
00143: /*!< Uncomment the following line if you need to relocate your vector Table in
00144:      Internal SRAM. */
00145: /* #define VECT_TAB_SRAM */
00146: #define VECT_TAB_OFFSET  0x10000 /*!< Vector Table base offset field.
00147:                                      This value must be a multiple of 0x200. */
00148:
```

图 3-6 重新定义宏 VECT_TAB_OFFSET

system_stm32f207xx.c 为系统文件，一般情况下我们最好不要修改。因为这个文件是可以通过“STM32f2xx_Clock_Configuration_V1.0.0.xls”这个程序生成的。可以这样实现，在 c 语言中的 main 函数第一件事设置偏移量：

```
SCB->VTOR = FLASH_BASE | 0x10000;
```

这样，我们就完成了中断向量表偏移量的设置。

通过以上两个步骤的设置，我们就可以生成 APP 程序了，只要 APP 程序的 FLASH 大小不超过我们的设置即可。

不过 MDK 默认生成的文件是.hex 文件，并不方便我们用作 IAP 更新，我们希望生成的文件是.bin 文件，这样可以方便进行 IAP 升级（至于为什么，请大家自行百度 HEX 和 BIN 文件的区别）。

注意：因为我们加密系统的原因，MDK 生成的.hex 文件也被加密了。所以，我们必须对.hex 进行解密才能使用下面的方法进行转换。

在 ST 提供的 IAP 官方例程中有转换工具，具体见图 3-7。

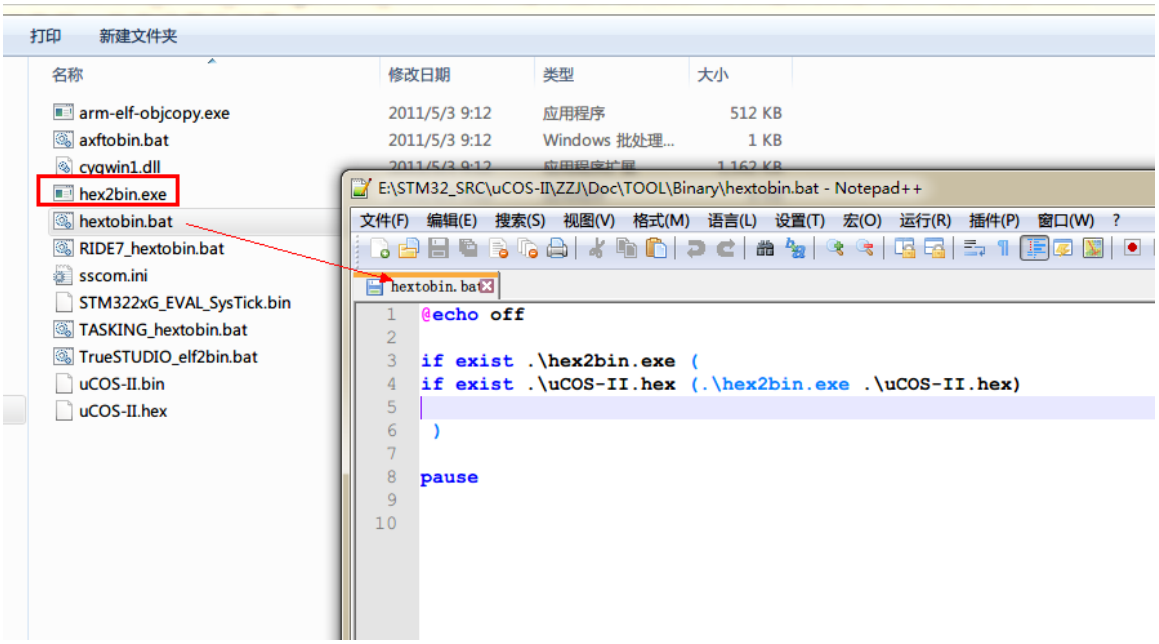

图 3-7 hex 文件转 bin 文件工具

文件夹里的工具很多，我们只需要使用 **hex2bin.exe** 就可以了。为了方便使用，我写了一个批处理文件，只要将 MDK 生成的 hex 文件命名为 uCOS-II.hex，然后将此文件放在这个目录下即可直接点击批处理文件生成我们需要的 bin 文件。当然，大家可以按照自己的需求自行修改批处理文件。

4 补充说明

经过本次测试 IAP 在我们 STM32F207 上是可行的，使用在我们以后的项目中能方便我们进行固件升级。

随着项目的进行，此版本 IAP 还有一些地方需要改进：

1）通信协议的完善，本测试版本的协议比较简单，实现的功能也有限。根据项目的需求，在后面进行增加和修改；

2）由于 MDK 不支持像 Makefile 那样的工程管理，我们可能需要将 IAP 工程和 APP 工程分开，因为随着我们工程功能的增多，一些不必要的代码会增加 IAP 的体积。现在编译出的 IAP 程序差不多有 20 几 KB 了，这里我们预留了 64KB 空间给 IAP。现在空间是够的，但是不能保证以后一直够；

3）Upload 的功能还没实现，但具体代码已经有了。

最后希望大家多提建议。