X-Cube-SBSFU 使用技巧（之四）

——从单核 STM32H7 到双核 STM32H7 的移植与集成

关键字：安全启动，安全升级，SBSFU，签名校验，加解密

1. 引言

安全启动与安全更新是嵌入式设备的基本安全要求之一：安全启动提供信任根，保证每次设备启动运行的应用程序的完整性与合法性；安全更新则在固件升级环节避免系统软件被恶意修改或替换。

为了方便客户在其嵌入式设备的设计中更加容易地集成安全启动和安全更新功能，STM32 提供了安全启动与安全更新的参考实现，支持众多 STM32 MCU 系列。其中 X-CUBE-SBSFU 是针对 Cortex V6/V7M 内核的 STM32 MCU 产品的参考方案，软件包下载地址：*https://www.st.com/x-cube-sbsfu*。

在 X-CUBE-SBSFU 使用技巧的第一篇，我们对软件包及软件架构等进行了介绍，让读者对这个软件包有个初步的认识；第二篇重点介绍了如何将 SBSFU 与应用程序集成，实现安全启动和安全升级的话题；第三篇主要介绍了如何将 SBSFU 参考实现从一个 STM32 MCU 平台移植到另外一个 STM32 MCU 型号的硬件平台的内容。这一篇我们将以 STM32H7 为例，讨论如何基于 STM32H7 单核的 SBSFU 参考实现移植到双核 STM32H7 平台上，并与 STM32H7 上的双核应用进行集成。

2. 双核安全启动方案的选择

从单核到双核的移植需要考虑两大方面因素，一方面是在第三篇中介绍过的单纯从一个硬件平台移植到另一个的过程；另一方面是关于双核的考虑。在 STM32H755 这样的双核架构的 MCU 中有 CM7 和 CM4 两个内核，分别运行不同的应用代码，这就意味着双核的安全启动至少需要考虑这样几个问题：

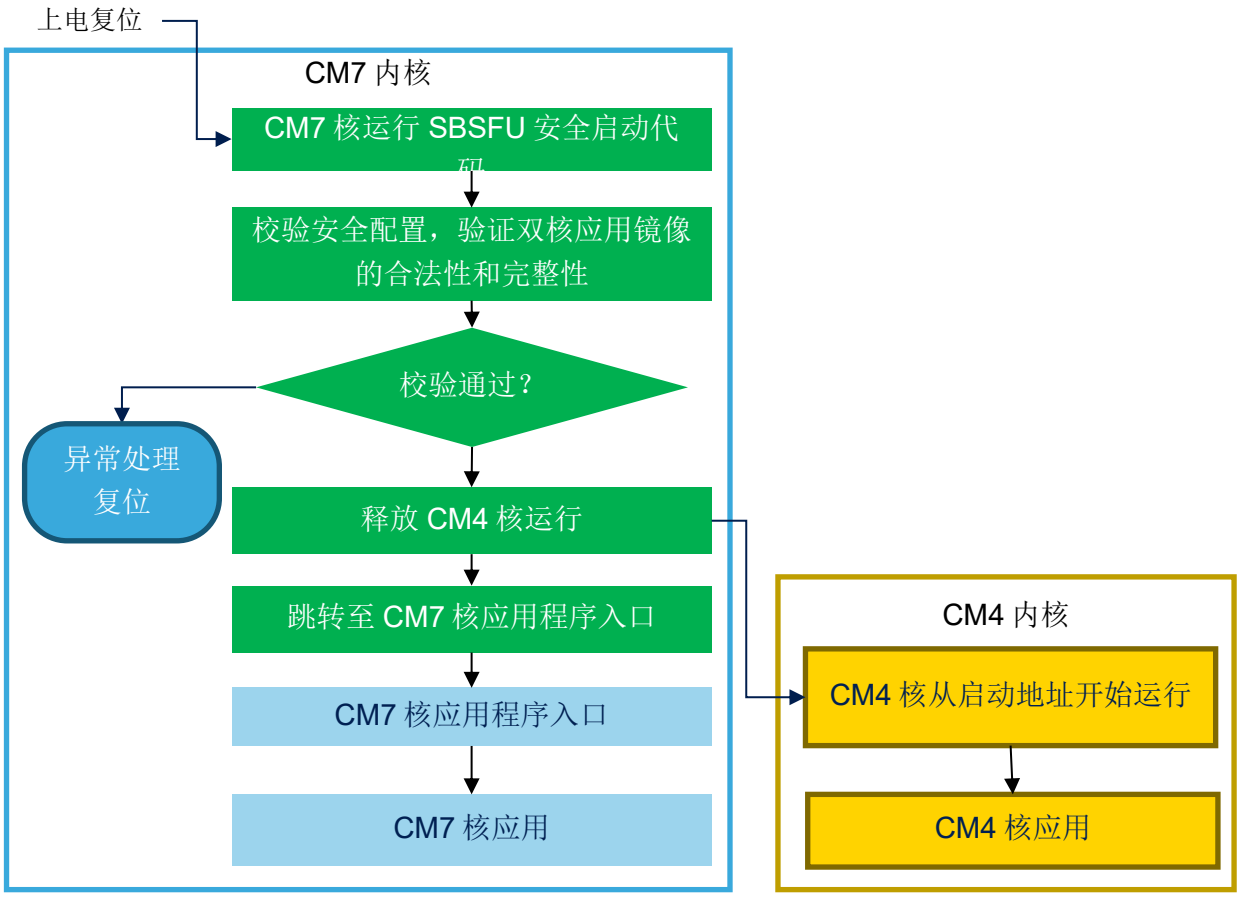
- 两个核的启动顺序，安全启动代码从哪个核开始运行？
- 运行在两个核上的应用程序 binary 的校验问题，是作为两个 image 分别校验还是合成一个 binary 进行启动校验以及升级？
- 运行在两个核上的应用代码地址如何配置？

这些问题的答案未必是唯一的，根据不同的需求可以采用不同的方式。本文的例子将采用如下的方案进行展开介绍：

- 上电后启动配置只运行 CM7 内核，即安全启动由 CM7 负责，SBSFU 的代码也只运行在 CM7 内核上；

  - 从安全角度考虑，CM7 内核的 SBSFU 需要对选项字节的配置做安全检查，例如确保 CM4 内核的启动选项被禁止等；
  - 为了保证启动入口的唯一性，在最终产品的安全配置中应当设置 RDP2，以避免 CM7/CM4 启动配置及启动地址被随意修改；
- 应用程序分为 CM7 内核和 CM4 内核两个部分，两个 binary 会被合并成一个 image 进行统一的签名校验以及加解密，固件更新的过程也会使用合并的 binary 镜像。
- 安全启动过程将对应用镜像文件进行校验，校验成功后将跳转至 CM7 内核的应用代码执行，并释放 CM4 内核。
  - 这里也可以选择由 CM7 内核上运行的用户应用程序控制 CM4 内核的释放。
- 启动流程如图 1 所示

图1. 双核安全启动方案启动流程



采用这种方案的好处是比较简单，对现有的 SBSFU 参考实现改动比较小，由于 CM7 内核启动时 CM4 内核还没有开始运行，所有的 SBSFU 代码都仅仅运行在 CM7 内核上，而且 CM7 和 CM4 内核的应用程序代码会合并成一个二进制文件，这样对于 SBSFU 来说并不需要关心被校验的代码是属于 CM7 内核还是 CM4 内核的，所以校验流程和单核的情况相比并没有变化，而 CM4 内核应用程序的启动地址也可以相对灵活选择。这样移植的过程主要还是集中在与单核芯片类似的硬件平台移植的部分。另外参考实现的 UserApp 部

分也是 H7 单核的，我们可以先保持 UserApp 为单核不变，方便验证第一步骤移植的有效性。

3. SBSFU 从单核 H753 切换到双核 H755

本节以 STM32H7 为例，具体介绍如何将 X-CUBE-SBSFU 基于单核 H7 的参考实现移植到双核 H7 平台。示例选择的目标芯片和开发板为 Nucleo-STM32H755ZIQ，根据第三篇中关于 SBSFU 移植的介绍，在选择参考实现基础的时候我们尽可能寻找和目标平台最接近的芯片系列，因此在这个示例中选择 Nucleo-STM32H753ZI 开发板平台的 SBSFU 作为移植的基础，参考工程选择 Projects\NUCLEO-H753ZI\Applications\2_Images_OSC，缺省的 Crypto 方案为 ECDSA 签名+AES CBC 加密（加密主要用于安全升级部分）。

3.1. SBSFU 硬件平台移植相关的修改

3.1.1. 工程基本配置和编译选项

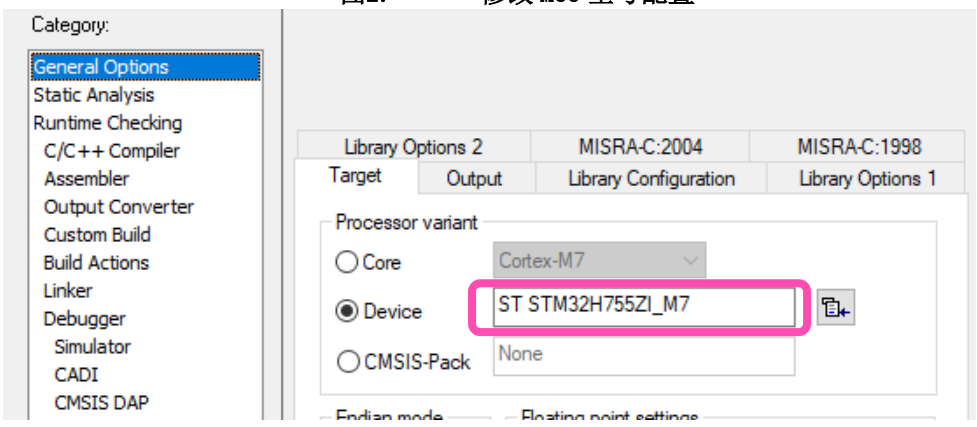
第一步，首先需要对几个工程的基本配置针对 H755ZI 进行相应修改，以下几个修改对 SECoreBIN，SBSFU，UserApp 几个工程都适用。

示例中工程的名字从 STM32H753ZIxxx 改为了 STM32H755ZIxxx（修改方法略过），因而编译结果所在的目录也会相应变化，这里会涉及某些 postbuild 脚本的修改。

示例以 IAR 为例，如选择其他工具链也需要做类似修改。

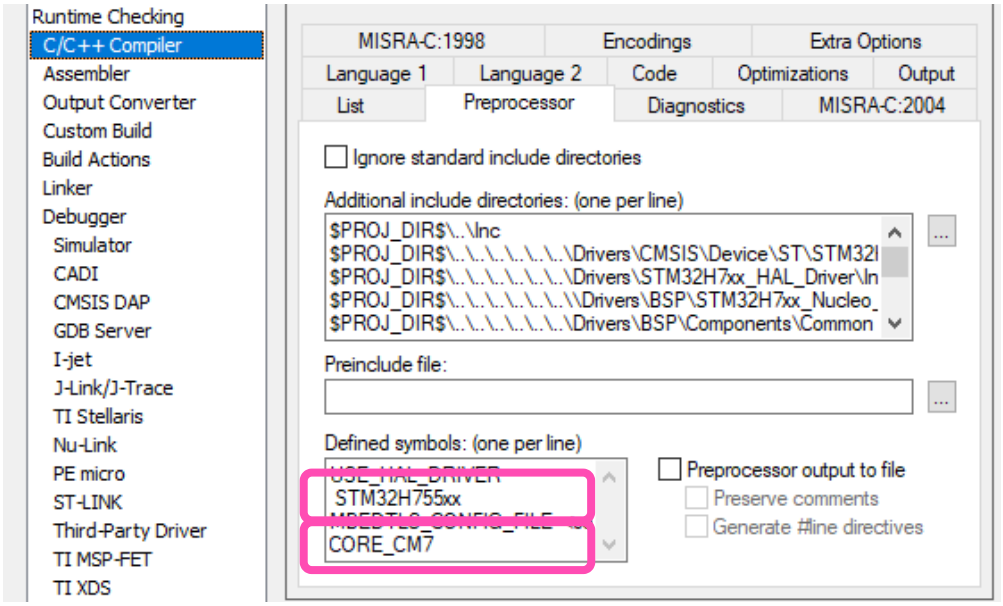
- 芯片 Device 的修改，选择 H755ZI_M7。如图 2

图2. 修改 MCU 型号配置



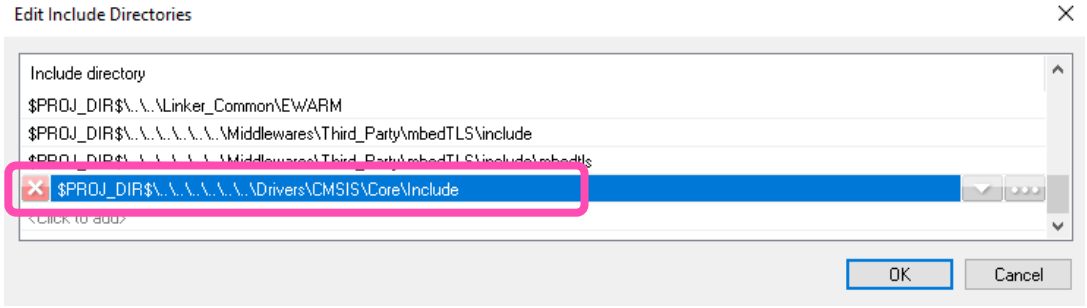
- 针对双核 H755 的内核需要定义编译选项 STM32H755xx 和 CORE_CM7，如图 3

图3. IAR 中 MCU 型号配置修改



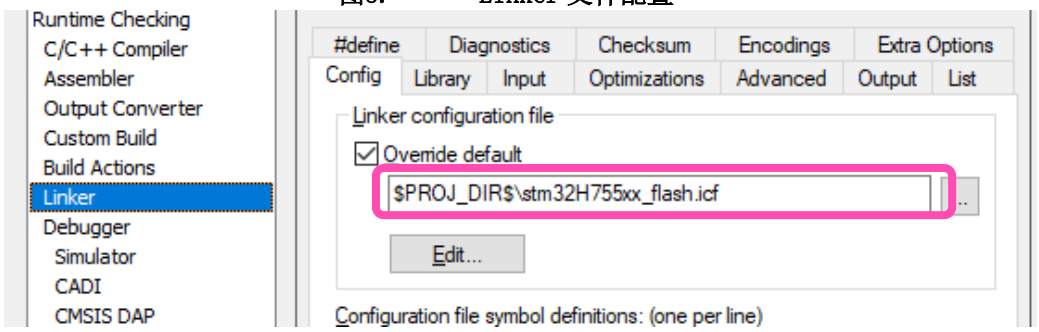
- 添加 CMSIS 目录作为头文件搜索目录之一，如图 4

图4. 添加头文件搜索路径



- 修改 linkder file 的名称以及在 linker 配置中的引用，见图 5

图5. Linker 文件配置



- 修改工程中的 startup_stm32h753xx.s 为 startup_stm32h755xx.s，可以使用 H7 CubeFW 包中 Templates 中对应的文件：
  Drivers/CMSIS/Device/ST/STM32H7xx/Source/Templates/iar(or arm or gcc)/startup_stm32h755xx.s

  此修改对 SECoreBin 工程不需要。

3.1.2. 硬件驱动配置

第二步，检查底层硬件的异同，并对相关硬件配置部分做必要的修改。硬件配置部分的修改，只要该硬件在相关工程中使用到，则需要做相应的检查和修改，例如 UART 在 SBSFU 和 UserApp 工程都用到，Flash 驱动在 SBSFU、SECoreBin 和 UserApp 工程中都用到等。

- COM UART GPIO
  - Nucleo-H753ZI 和 Nucleo-H755ZI-Q 两个开发板的 COM 缺省都使用 USART3，所以这部分配置不需要修改。
- 用户按键 GPIO
  - Nucleo-H753ZI 和 Nucleo-H755ZI-Q 两个开发板缺省都使用 PC13 作为 push button 的 IO，所以这部分配置不需要修改。
- TAMPER 和 DAP
  - 两个平台具有相同的 TAMPER 和 SWD 对应的 GPIO，所以这部分配置不需要修改。
- Flash
  - H753ZI 和 H755ZI-Q 同样都是 2MB 大小的用户 Flash，具有双 bank，每个 bank 1MB，分 8 个 sector，每个 sector 128KB，32 Byte 字大小，所以这部分不需要进行修改。
  - 注意：如果移植到其他系列 Bank 和 sector 大小有变化，或者每次写入的字长有不同时，需要修改 xxx_SECoreBin\Src\se_low_level.c 和 xxx_SBSFU\SBSFU\Target\sfu_low_level_flash_int.c 中 Flash 操作函数的实现。
- System Clock
  - 由于 Nucleo-H755ZI-Q 是带 SMPS 的，所以时钟初始化部分需要将供电配置部分由 LDO 模式改为 SMPS 模式，如图 6：

图6. 系统时钟初始化修改

```
 99 static void SystemClock_Config(void)
100 {
101   RCC_ClkInitTypeDef RCC_ClkInitStruct;
102   RCC_OscInitTypeDef RCC_OscInitStruct;
103   HAL_StatusTypeDef ret = HAL_OK;
104
105   /*!< Supply configuration update enable */
106   HAL_PWREx_ConfigSupply(PWR_LDO_SUPPLY);
107
108   /* The voltage scaling allows optimizing the
109      clocked below the maximum system frequency
110      regarding system frequency refer to produc
111   __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR
112
```

```
 99 static void SystemClock_Config(void)
100 {
101   RCC_ClkInitTypeDef RCC_ClkInitStruct;
102   RCC_OscInitTypeDef RCC_OscInitStruct;
103   HAL_StatusTypeDef ret = HAL_OK;
104
105   /*!< Supply configuration update enable */
106   HAL_PWREx_ConfigSupply(PWR_DIRECT_SMPS_SUPPLY);
107
108   /* The voltage scaling allows optimizing the power
109      clocked below the maximum system frequency, to u
110      regarding system frequency refer to product data
111   __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTA
112
```

3.2. Postbuild 脚本

SECoreBin 的工程目录下有三个 postbuild 脚本模板，里面事先写好了 sbsfuelf 的路径和文件名，其中项目名称的部分需要根据实际修改后的项目名称进行修改。例如 EWARM 目录下的.bat 中 sbsfuelf 文件的路径和名字需要做相应修改，如图 7：

图7. Postbuild 脚本中 sbsfuelf 路径修改

```
set "ecckey=%SBSFUBootLoader%\\2_Images_SECoreBin\\Binary\\ECCKEY%fwid%.txt"
set "sbsfuelf=%SBSFUBootLoader%\\2_Images_SBSFU\\EWARM\\STM32H755ZI-Nucleo\\Exe\\STM32H755ZI-Nucleo.out"
set "oemkey=%SBSFUBootLoader%\\2_Images_SECoreBin\\Binary\\OEM_KEY_COMPANY%fwid%_key_AES_GCM.bin"
```

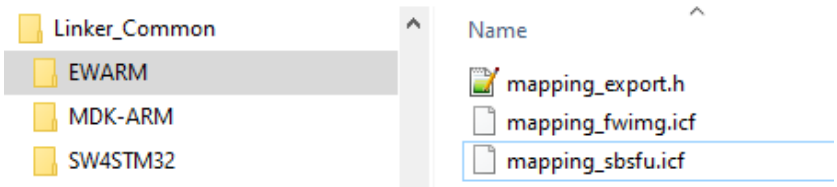
这个修改是由于前面对工程的名字做了修改，如果自己移植的时候工程名字没有改，则这个步骤不需要。

3.3. Memory map 与 linker 配置

由于 Nucleo-H753ZI 和 Nucleo-H755ZI-Q 两个硬件平台的 Flash 和 RAM 具有相同的大小和布局，如果仅仅作为平台间移植操作，不需要对 Memory map 以及 linker file 做任何修改，完成前面三个步骤之后，就可以直接将 Nucleo-H753ZI 的 SBSFU 参考实现及其示例 UserApp 在 Nucleo-H755ZI-Q 开发板上运行起来。

如果实际移植的某个目标系列的 Flash 或者 RAM 的大小与原始参考工程使用的硬件平台的大小不同，这时候就可能需要对 memory 布局进行调整，调整的方法是修改 Linker_Common 目录下的共享地址定义文件的内容，如图 8：

图8. Linker Common 中的 memory map 定义文件



- mapping_sbsfu.icf
  - 定义 SecureBoot 使用的 memory 布局，包括内部各个部分的布局，例如 Key 放在哪里，SECoreBin 的入口在哪里，Firewall 或者 HDP 保护的区域在哪里，哪部分 SRAM 是保留给 SBSFU 使用的等等。
  - 如果不对 SBSFU 做任何优化和裁剪，那么这个文件不需要修改。
  - 如果的确需要修改 SBSFU 的 memory 布局，需要注意结合某些硬件安全特性时的限制条件，例如 MPU 区域起始地址和大小要满足倍数关系等等。
- mapping_fwimg.icf
  - 文件定义应用程序运行区的起始地址和大小，下载区的起始地址和大小，swap 区的起始地址和大小。
  - 可以通过修改这个文件中的定义来调整 App 使用区的位置以及整体占用空间的大小。调整过程中也需要注意一些对起始地址和大小的限制，例如需要对齐到 Flash page/sector 等等。
  - 如果 App 需要在运行期间存储用户数据，那么存储数据用到的 NVM 存储区需要在前面提到的三个区以外的空间，不能有重叠。

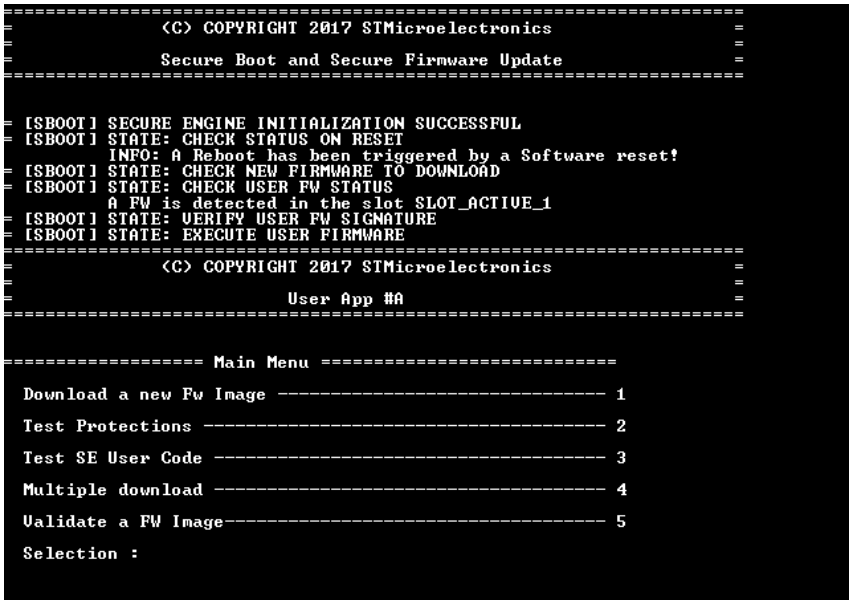
3.4. 测试 CM7 内核移植后的 SBSFU

按照前面几个小节分别对三个工程 SECoreBin，SBSFU 和 UserApp 进行修改完成后，重新按顺序编译三个工程，编译成功后，在 UserApp 工程的 Binary 目录下会生成 SBSFU_UserApp.bin 和 UserApp.sfb 文件。

接下来可以测试移植的结果，将最终生成的二进制文件 SBSFU_UserApp.bin 直接烧写入 Nucleo-H755ZI-Q 开发板，打开上位机的串口工具，则可以从串口工具终端看到类似

如图 9 的打印信息，看到这样的打印说明 SecureBoot 已经正确启动，UserApp 也校验成功并正常运行

图9. SBSFU 在 H755ZI-Nucleo 开发板上运行的打印信息

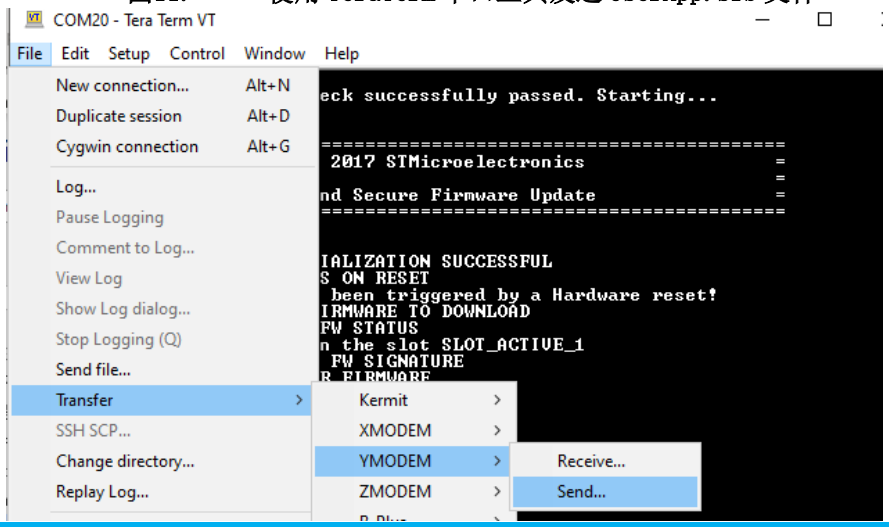


启动过程验证成功后，可以继续验证安全升级功能。升级过程可以从应用程序触发，也可以按住板子上的 User 按键然后复位，直接触发 SecureBoot 代码进入下载模式。如果在上位机串口看到类似图 10 的打印信息，说明 SBSFU 已经进入下载模式，并等待发送固件 binary。

图10. SBSFU 开始接收版本下载的打印信息

版本升级过程需要使用 TeraTerm 的 Transfer->YMODEM->Send 功能，将 UserApp.sfb 文件通过串口传送给 MCU，如图 11

图11. 使用 TeraTerm 串口工具发送 UserApp.sfb 文件



测试下载更新功能也会看到类似图 12 的串口打印信息：

图12. 安全升级串口打印信息

4. SBSFU 针对双核应用的修改

经过第 2 章的步骤，从 H753ZI 切换到 H755ZI 的 CM7 部分的修改已经完成，也就是说，CM7 内核上的安全启动已经能够正常运行并启动运行在 CM7 内核上的应用。但是到目前为止还没有考虑 CM7+CM4 双核应用的情况。接下来讨论针对双核 App 的修改。

4.1. CPU2 Boot 选项

从安全的角度考虑，由于 SecureBoot 完全由 CM7 内核掌握，我们需要确保每次上电只有 CM7 内核运行，SecureBoot 代码运行期间 CM4 内核一直保持挂起状态。因此在 SBSFU 的初始化过程中需要检查选项字节的配置，确认 CM4 内核的 Boot 选项被禁止，如果发现选项与期望配置不符，则在 DEV 模式自动设置该选项字节配置。这里我们还可以对 CPU2 的 BootAddress 做一个检查，这个示例中只检查了 CPU2 启动地址是否在 ActiveSlot 范围内，如果应用事先确定了 Boot 地址则可以改为直接比较期望的地址与 CPU2 BootAddr 是否一致。

为此我们将对 SFU_LL_SECU_CheckApplyStaticProtections(void)函数进行修改，增加一项 CPU2_BOOT 选项字节的检查。以下是几个修改涉及的文件：

- app_sfu.h：添加 CPU_BOOT2 检查的编译选项

2_Images_SBSFU\SBSFU\App\app_sfu.h

```
#if !defined(SECBOOT_DISABLE_SECURITY_IPS)

/* Uncomment the following defines when in Release mode.
    In debug mode it can be better to disable some of the following protection
    for a better Debug experience (WRP, RDP, IWDG, DAP, etc.) */
         switching to UserApp: a refresh is needed.

#define SFU_CPU2_BOOT_PROTECT_ENABLE
#define SFU_WRP_PROTECT_ENABLE
#define SFU_RDP_PROTECT_ENABLE
…
```

- sfu_boot.h：添加对应 CPU2_BOOT 检查的 FlowControl 定义

2_Images_SBSFU\SBSFU\App\sfu_boot.h

```
…
#ifdef  SFU_CPU2_BOOT_PROTECT_ENABLE
#define FLOW_STEP_CPU2BOOT 0x0000a4b5U                /*!< Step CPU2 BOOT config */
#else
#define FLOW_STEP_CPU2BOOT 0x00000000U                /*!< No effect on control flow */
#endif /* SFU_RDP_PROTECT_ENABLE */
…
```

```
/**
  * @brief  SFU_BOOT Flow Control : Control values static protections
  */
#define FLOW_CTRL_UBE (FLOW_CTRL_INIT_VALUE ^ FLOW_STEP_UBE)
#define FLOW_CTRL_WRP (FLOW_CTRL_UBE ^ FLOW_STEP_WRP)
#define FLOW_CTRL_PCROP (FLOW_CTRL_WRP ^ FLOW_STEP_PCROP)
#define FLOW_CTRL_SEC_MEM (FLOW_CTRL_PCROP ^ FLOW_STEP_SEC_MEM)
#define FLOW_CTRL_RDP (FLOW_CTRL_SEC_MEM ^ FLOW_STEP_RDP)
#define FLOW_CTRL_CPU2BOOT (FLOW_CTRL_RDP ^ FLOW_STEP_CPU2BOOT)
#define FLOW_CTRL_STATIC_PROTECT FLOW_CTRL_CPU2BOOT
```

- sfu_low_level_security.c：添加检查与设置 CPU2_BOOT 选项的代码

2_Images_SBSFU\SBSFU\Target\sfu_low_level_security.c

```
…
#ifdef SFU_CPU2_BOOT_PROTECT_ENABLE
static SFU_ErrorStatus SFU_LL_SECU_SetProtectionCPU2Boot(FLASH_OBProgramInitTypeDef
*psFlashOptionBytes,
                                                         SFU_BoolTypeDef
*pbIsProtectionToBeApplied);
#endif /*SFU_CPU2_BOOT_PROTECT_ENABLE*/
…
```

```
SFU_ErrorStatus SFU_LL_SECU_CheckApplyStaticProtections(void)
{
…
#ifdef SFU_RDP_PROTECT_ENABLE
    if (e_ret_status == SFU_SUCCESS)
    {
```

```
      e_ret_status = SFU_LL_SECU_SetProtectionRDP(&flash_option_bytes,
&is_protection_to_be_applied);
    }
#endif  /* SFU_RDP_PROTECT_ENABLE */

#ifdef SFU_CPU2_BOOT_PROTECT_ENABLE
    if (e_ret_status == SFU_SUCCESS)
    {
      e_ret_status = SFU_LL_SECU_SetProtectionCPU2Boot(&flash_option_bytes,
&is_protection_to_be_applied);
    }
#endif /*SFU_CPU2_BOOT_PROTECT_ENABLE*/
…
  return e_ret_status;

}
```

```
#if defined SFU_CPU2_BOOT_PROTECT_ENABLE
SFU_ErrorStatus SFU_LL_SECU_SetProtectionCPU2Boot(FLASH_OBProgramInitTypeDef
*psFlashOptionBytes,
                                                  SFU_BoolTypeDef *pbIsProtectionToBeApplied)
{
  SFU_ErrorStatus e_ret_status = SFU_ERROR;

  /* Check/Apply CM4 Boot config ****************************************************/
  /* Since we do Secure Boot from CM7 side only, need to make sure CM4 is not booting */
  if ((psFlashOptionBytes->USERConfig & OB_BCM4_ENABLE) == OB_BCM4_DISABLE)
  {
    /* Also need to make sure the boot address of CM4 is within Active slot 1 range */
    if ((psFlashOptionBytes->CM4BootAddr0 > SLOT_ACTIVE_1_START)
        && (psFlashOptionBytes->CM4BootAddr0 < SLOT_ACTIVE_1_END)
        && (psFlashOptionBytes->CM4BootAddr1 > SLOT_ACTIVE_1_START)
        && (psFlashOptionBytes->CM4BootAddr1 < SLOT_ACTIVE_1_END))
    {
      e_ret_status = SFU_SUCCESS; /*Protection already applied */
      /* Execution stopped if flow control failed */
      FLOW_CONTROL_STEP(uFlowProtectValue, FLOW_STEP_CPU2BOOT, FLOW_CTRL_CPU2BOOT);
    }
    else
    {
      TRACE("\r\n= [SBOOT] System Security Configuration failed: CPU2 BOOT Address is
incorrect. STOP!");
      /* Security issue : execution stopped ! */
      SFU_EXCPT_Security_Error();
    }
  }
  else
  {
#if defined(SECBOOT_OB_DEV_MODE)
    psFlashOptionBytes->OptionType      = OPTIONBYTE_USER;
    psFlashOptionBytes->USERConfig      &= (~OB_BCM4_ENABLE);
    if (HAL_FLASHEx_OBProgram(psFlashOptionBytes) == HAL_OK)
    {
      *pbIsProtectionToBeApplied |= 1U;
      e_ret_status = SFU_SUCCESS;
      /* Execution stopped if flow control failed */
      FLOW_CONTROL_STEP(uFlowProtectValue, FLOW_STEP_CPU2BOOT, FLOW_CTRL_CPU2BOOT);
    }
#else
    TRACE("\r\n= [SBOOT] System Security Configuration failed: CPU2 BOOT config is incorrect.
STOP!");
    /* Security issue : execution stopped ! */
    SFU_EXCPT_Security_Error();
#endif /* SECBOOT_OB_DEV_MODE */
  }

  return e_ret_status;
}
#endif /*SFU_CPU2_BOOT_PROTECT_ENABLE*/
```

如果希望 SBSFU 在校验成功之后直接释放 CM4 内核的运行，还可以在该文件加上以下部分代码，如果选择在 CM7 内核的应用中释放 CM4 内核的运行，则这段可省略。

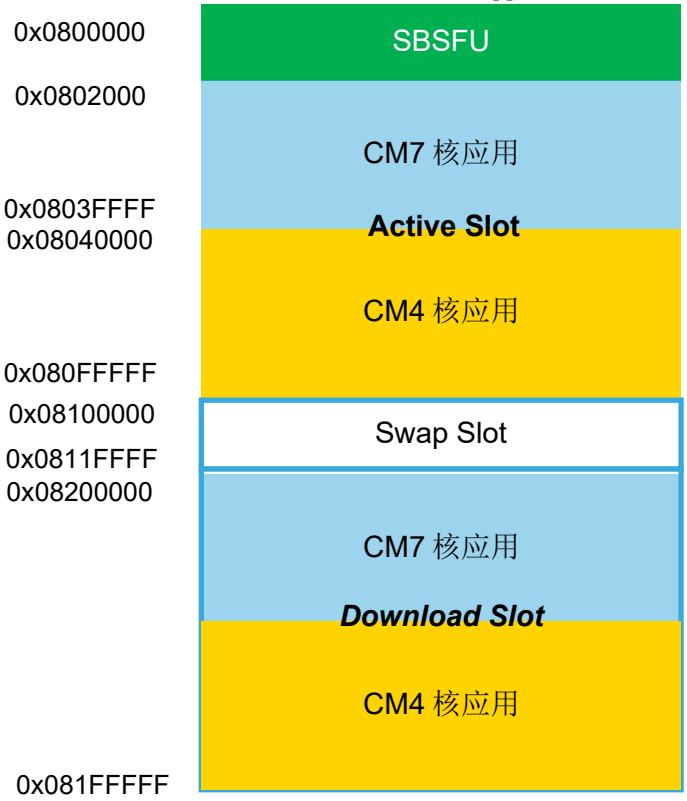
```
__RAM_FUNC void SFU_LL_SECU_ActivateSecUser(uint32_t Address)
```

```
{
…
    /* clear process stack & unprivileged bit */
    __set_CONTROL(__get_CONTROL() & ~0x3U);

#ifdef SFU_CPU2_BOOT_PROTECT_ENABLE
    /* release CPU2 for dual core H7 */
    SET_BIT(RCC->GCR, RCC_BOOT_C2) ;
#endif
    /* returns from interrupt into application */
    launch_application(Address, (uint32_t)jump_to_function);
  }

  /*we should never reach this point */
  NVIC_SystemReset();
}
```

4.2. Memory 布局以及 CM4 内核应用程序的启动地址

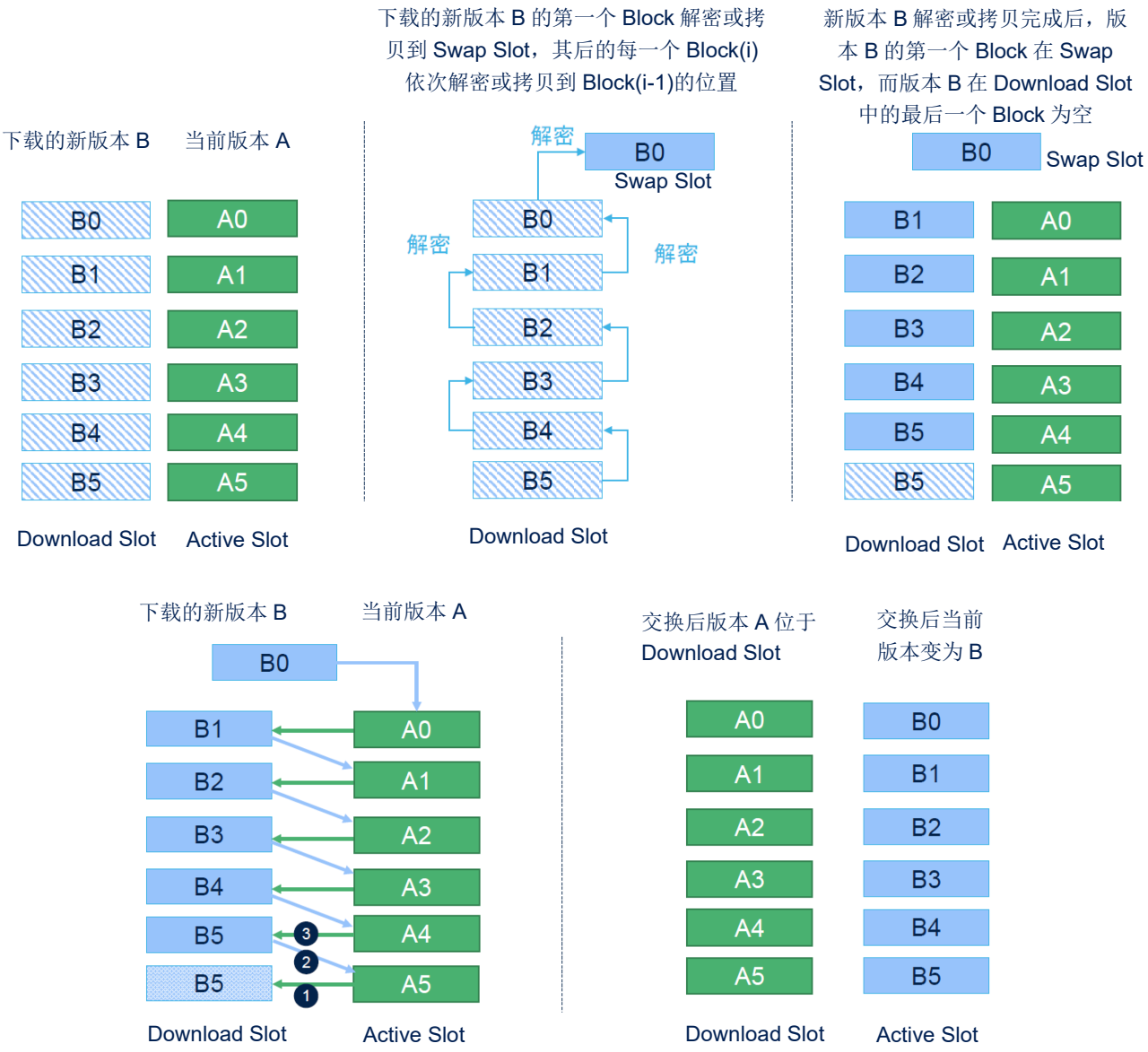
前面提到我们选择的 SBSFU 参考实现基础为 H753ZI Nucleo 板的双镜像实现，也就是说 Flash 中会有两个存放 App 的 Slot，一个 Active Slot，一个 Download Slot。那么 CM7 内核与 CM4 内核应用代码的启动地址都应当在 Active Slot 之内，同时 CM4 内核应用的起始地址需要对齐到 0x10000。在本示例中布局定义如图 13， CM7 内核应用的起始地址为 0x08020000+0x400（H7 的 Header 大小为 1024 字节），CM4 内核应用的启动地址设在 0x08040000，这个地址可以根据实际的应用中的情况和需要进行调整。

图13. 带安全启动的 H755ZI 双核 App Flash 地址布局



虽然此时 CM7 和 CM4 的应用为两个不同的部分，但是在我们选择的方案中把二者合并看做一个整体，因此，升级的过程中使用到 Swap slot 进行交换的时候，也是把两个应用合并的 binary 看成一个 binary 进行升级。通过 Swap Slot 交换的过程如图 14 所示：

图14. 下载新版本 B 到 Download Slot 并与 Active Slot 中原版本 A 交换的过程



实际交换的过程从版本 A 的最后一个 Block 开始，最后一个 Block(N)写入 Download Slot 中版本 B 空出来的最后一个 Block(N)的位置，然后再将版本 B 位于 Block(N-1)位置但实际是最后一个 Block(N)的内容写入 Active Slot 中版本 A 的最后一个 Block(N)的位置，以此类推，直到最后将 Swap Slot 中的版本 B 的第一个 Block 的内容写入 Active Slot 的 Block(0)的位置，整个交换过程完成。交换完成后 Download Slot 中存放版本 A 的内容，Active Slot 中存放版本 B 的内容。对于这里的双核 H7 的示例，由于我们将 CM7 和 CM4 的两个应用合并为一个 binary 来对待，所以这里的版本 A 或 B 实际都包含了 CM7+CM4 两个应用的内容，交换过程和以往单核应用的情况没有区别。

由于 SBSFU 本身把 Active Slot 看成一个整体，所以对于 SBSFU 来说，CM4 的启动地址不需要在 linker file 中出现，但是我们在集成 H755ZI 双核应用的时候会引用到 Linker_Common 目录下的共享 linker 文件，因此在这里作为后面集成双核应用的准备，我们还是对共享 Linker 文件做一些修改，增加 CM7 内核应用的结束地址和 CM4 内核应用的起始地址的定义，方便应用程序引用。涉及的改动如下（红色高亮的部分）：

Linker_Common\EWARM\mapping_fwimg.icf:

```
/* Swap sector in sector 0 of Bank2 : 128 kbytes */
define exported symbol  __ICFEDIT_SWAP_start__ = 0x08100000;
define exported symbol  __ICFEDIT_SWAP_end__   = 0x0811FFFF;

/* Active slot #1 in Sector 1 to 7 of Bank1; 7 x 128 kbytes */
define exported symbol  __ICFEDIT_SLOT_Active_1_start__  = 0x08020000;
define exported symbol  __ICFEDIT_SLOT_Active_1_M7_end__ = 0x0803FFFF;
define exported symbol  __ICFEDIT_SLOT_Active_1_M4_start__ = 0x08040000;
define exported symbol  __ICFEDIT_SLOT_Active_1_end__      = 0x080FFFFF;
define exported symbol  __ICFEDIT_SLOT_Active_1_header__= __ICFEDIT_SLOT_Active_1_start__;


/* Dwl slot #1 in sector 1 to 7 of Bank2 : 7 * 128 kbytes */
define exported symbol  __ICFEDIT_SLOT_Dwl_1_start__= 0x08120000;
define exported symbol  __ICFEDIT_SLOT_Dwl_1_end__  = 0x081FFFFF;

/* Slots not configured */
```

Linker_Common\MDK-ARM\mapping_fwimg.h

```
/* Swap sector in sector 0 of Bank2 : 128 kbytes */
#define SWAP_START                    0x08100000
#define SWAP_END                      0x0811FFFF

/* Active slot #1 in Sector 1 to 7 of Bank1; 7 x 128 kbytes */
#define SLOT_ACTIVE_1_START           0x08020000
#define SLOT_ACTIVE_1_M7_END            0x0803FFFF
#define SLOT_ACTIVE_1_M4_START        0x08040000
#define SLOT_ACTIVE_1_END             0x080FFFFF
#define SLOT_ACTIVE_1_HEADER          SLOT_ACTIVE_1_START

/* Dwl slot #1 in sector 1 to 7 of Bank2 : 7 * 128 kbytes */
#define SLOT_DWL_1_START              0x08120000
#define SLOT_DWL_1_END                0x081FFFFF
```

Linker_Common\SW4STM32\mapping_fwimg.ld

```
/* Swap sector in sector 0 of Bank2 : 128 kbytes */
__ICFEDIT_SWAP_start__           = 0x08100000;
__ICFEDIT_SWAP_end__             = 0x0811FFFF;

/* Active slot #1 in Sector 1 to 7 of Bank1; 7 x 128 kbytes */
__ICFEDIT_SLOT_Active_1_start__  = 0x08020000;
__ICFEDIT_SLOT_Active_1_CM7_end__   = 0x0803FFFF;
__ICFEDIT_SLOT_Active_1_CM4_start__    = 0x08040000;
__ICFEDIT_SLOT_Active_1_end__    = 0x080FFFFF;
__ICFEDIT_SLOT_Active_1_header__ = __ICFEDIT_SLOT_Active_1_start__;

/* Dwl slot #1 in sector 1 to 7 of Bank2 : 7 * 128 kbytes */
__ICFEDIT_SLOT_Dwl_1_start__     = 0x08120000;
__ICFEDIT_SLOT_Dwl_1_end__       = 0x081FFFFF;
```

5. SBSFU 与 H7 双核应用集成

5.1. 准备双核应用作为 UserApp

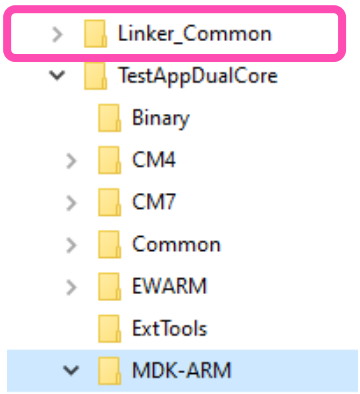
这里的双核 App 我们以 H7 的 CubeFW 包中的 Projects\NUCLEO-H745ZI-Q\Examples\BSP 为基础，当然也可以选择 Projects\NUCLEO-H745ZI-Q\Templates\目录下的双核工程模板进行修改。在这个例子工程中，CM7 内核的应用中将使用 USART3 用于串口打印，CM4 内核应用将使用 LED Red GPIO 用于闪灯，这样当两个内核的应用都运行起来时，我们将看到串口打印和红色 LED 的闪烁。如果 SBSFU 实现中没有在校验完

成后释放 CM4 内核，那么这里的应用程序需要确保 CM7 内核的应用会释放 CM4 内核的运行。

5.1.1. Flash 地址空间分配与 linker 文件配置

Flash 的地址空间分配在 4.2 章节已经提到，这里不再赘述。App 将引用 Linker_Common 文件的内容来指定 ROM 和 RAM 的地址。这里我们假设 Linker_Common 目录放在了工程目录的上两级目录位置，例如图 15 所示。

图15. Linker_Common 目录与 App 目录的相对位置



为了 Linker 能够找到引用的 Linker 文件，应用工程需要添加类似图 16 和 17 的配置

图16. IAR 中配置 Linker Common 共享链接文件的查找位置

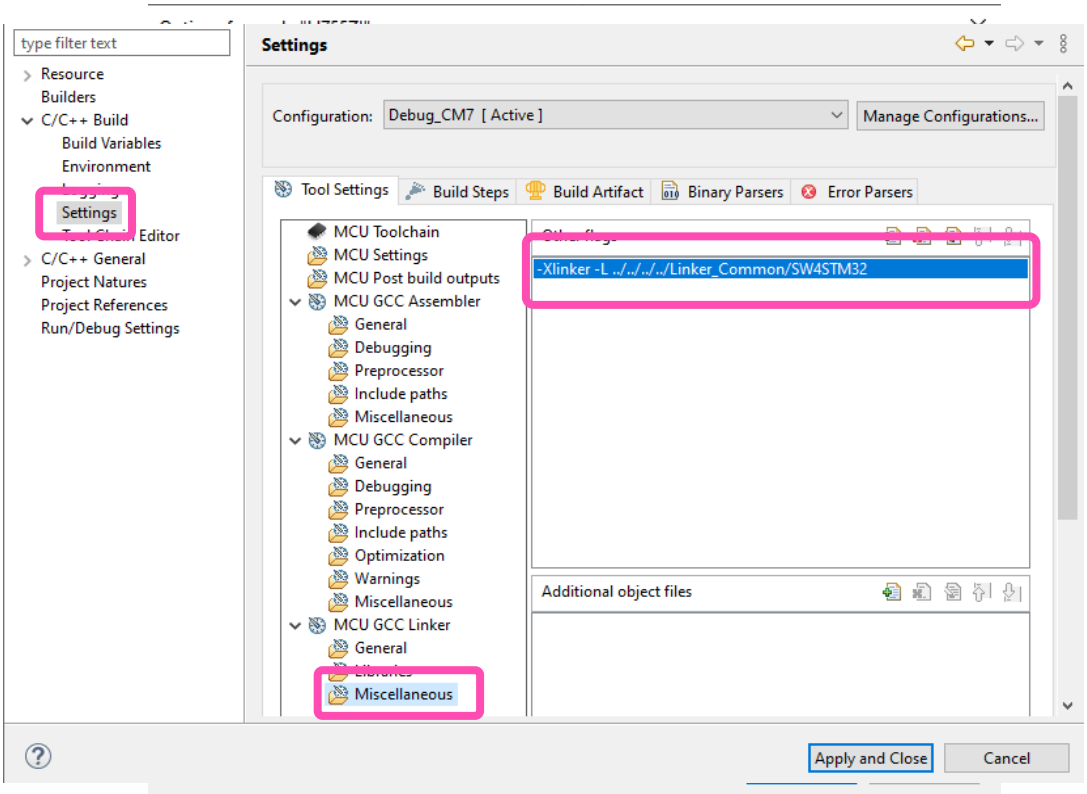


图17. STM32CubeIDE 中配置 Linker Common 共享链接文件的查找位置

```
;#! armcc -E -I.\
; *******************************************************************
; *** Scatter-Loading Description File generated by uVision ***
; *******************************************************************
#include "..\..\Linker_Common\MDK-ARM\mapping_sbsfu.h"
#include "..\..\Linker_Common\MDK-ARM\mapping_fwimg.h"

LR_IROM1 SLOT_ACTIVE_1_M4_START   {    ; load region size_region
  vector_start (SLOT_ACTIVE_1_M4_START) FIXED VECTOR_SIZE {
  *.o (RESET, +First)
  }
  ROM_region +0 {
  *(InRoot$$Sections)
  .ANY (+RO)
  .ANY (+XO)
  }
  SB_RAM_region (0x10000000) {
  .ANY (STACK)
  .ANY (HEAP)
  .ANY (+RW +ZI)
  }
}

; extra ROM region to make sure the binary size is a multiple
; of the AES block size (16 bytes) and H7 flash writing unit (32 bytes)
LR_ROM1(+0) ALIGN(32) {
  ForAlignment +0 {
    startup_stm32h745xx.o (ALIGNTOAESBLOCK,+Last)
  }
}
```

如果使用 KEIL 工具链，那么缺省的例子中是没有.sct 文件的。这里我们可以定义两个.sct 文件作为 CM7 内核与 CM4 内核的 Linker 文件，那么 Linker 引用文件的查找目录直接写在.sct 文件中即可。同样的的，.sct 文件会引用 Linker Common 目录下的共享 linker 文件。KEIL 工程 CM4 内核的.sct 文件内容如下：

KEIL 工程 CM7 内核的.sct 文件内容：

```
#! armcc -E -I.\
; *************************************************************
; *** Scatter-Loading Description File generated by uVision ***
; *************************************************************
#include "..\..\Linker_Common\MDK-ARM\mapping_sbsfu.h"
#include "..\..\Linker_Common\MDK-ARM\mapping_fwimg.h"


LR_IROM1 SLOT_ACTIVE_1_START + 0x400   {    ; load region size_region
  vector_start (SLOT_ACTIVE_1_START + 0x400 ) FIXED VECTOR_SIZE {
  *.o (RESET, +First)
  }
  ROM_region +0 {
  *(InRoot$$Sections)
  .ANY (+RO)
  .ANY (+XO)
  }
  SB_RAM_region (SE_REGION_RAM_END + 1) {
  .ANY (STACK)
  .ANY (HEAP)
  .ANY (+RW +ZI)
  }
}

; extra ROM region to make sure the binary size is a multiple of
; the AES block size (16 bytes) and H7 flash writing unit (32 bytes)
LR_ROM1(+0) ALIGN(32) {
  ForAlignment +0 {
    startup_stm32h745xx.o (ALIGNTOAESBLOCK,+Last)
  }
}
```

类似 IAR 的 linker 文件，这里.sct 文件也会在最后插入填充内容，确保生成的 image 遵守 16 字节对齐。不同的是，这里会用到一个 ALIGNTOAESBLOCK 的定义，这个定义需要增加在 startup_stm32h745xx.s 文件最后。

```
;*******************************************************************************
; Element for aligning the Firmware binary size on the AES block size (16 bytes)
; and H7 flash writing unit (32 bytes)
;*******************************************************************************
  AREA ALIGNTOAESBLOCK, DATA, READONLY, MERGE=1, STRINGS
                                DCB    "0123456789ABCDEF0123456789ABCDE",0

                END
;************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE*****
```

IAR 工程 CM4 内核应用的.icf 文件内容如下，其中高亮的部分是针对 SBSFU 集成的修改。这里的 aes_block_padding 部分主要针对 Crypto 方案中带有 AES 加密的情况，如果不使用任何加密，只是做签名校验，那么这个部分不是必须的。

```
/*###ICF### Section handled by ICF editor, don't touch! ****/
define memory mem with size = 4G;
/*-Editor annotation file-*/
/* IcfEditorFile="$TOOLKIT_DIR$\config\ide\IcfEditor\cortex_v1_0.xml" */
/*-Specials-*/

include "mapping_sbsfu.icf";
include "mapping_fwimg.icf";

define exported symbol __ICFEDIT_intvec_start__ = __ICFEDIT_SLOT_Active_1_M4_start__;

/*-Memory Regions-*/
define symbol __ICFEDIT_region_ROM_start__      = __ICFEDIT_intvec_start__; /* The intvec size is 0x400 on H7 */
define symbol __ICFEDIT_region_ROM_end__        = __ICFEDIT_SLOT_Active_1_end__ ;

define symbol __ICFEDIT_region_RAM_start__ = 0x10000000;
define symbol __ICFEDIT_region_RAM_end__   = 0x1003FFFF;
/*-Sizes-*/
define symbol __ICFEDIT_size_cstack__ = 0x800;
define symbol __ICFEDIT_size_heap__   = 0x400;
/**** End of ICF editor section. ###ICF###*/



define region ROM_region      = mem:[from __ICFEDIT_region_ROM_start__   to __ICFEDIT_region_ROM_end__];
define region RAM_region      = mem:[from __ICFEDIT_region_RAM_start__   to __ICFEDIT_region_RAM_end__];

define root section aes_block_padding with alignment=32
{
udata8 "Force Alignment";
pad_to 32;
};

define block CSTACK    with alignment = 8, size = __ICFEDIT_size_cstack__   { };
define block HEAP      with alignment = 8, size = __ICFEDIT_size_heap__     { };

initialize by copy { readwrite };
do not initialize  { section .noinit };

place at address mem:__ICFEDIT_intvec_start__ { readonly section .intvec };

place in ROM_region   { readonly, last section aes_block_padding };
place in RAM_region   { readwrite,
                        block CSTACK, block HEAP };
```

CM7 内核应用的.icf 文件如下：

```
/*###ICF### Section handled by ICF editor, don't touch! ****/

define memory mem with size = 4G;

/*-Editor annotation file-*/
/* IcfEditorFile="$TOOLKIT_DIR$\config\ide\IcfEditor\cortex_v1_0.xml" */
/*-Specials-*/

include "mapping_sbsfu.icf";
include "mapping_fwimg.icf";

define exported symbol __ICFEDIT_intvec_start__ = __ICFEDIT_SLOT_Active_1_start__ + 1024;
/* Cortex-M7 : must be a multiple of 1024 */

/*-Memory Regions-*/
define symbol __ICFEDIT_region_ROM_start__      = __ICFEDIT_intvec_start__; /* The intvec size is 0x400 on H7 */
define symbol __ICFEDIT_region_ROM_end__        = __ICFEDIT_SLOT_Active_1_M7_end__ ;
define symbol __ICFEDIT_region_RAM_start__      = __ICFEDIT_SE_region_RAM_end__ + 1;

define symbol __ICFEDIT_region_RAM_end__       = 0x2001FFFF;
define symbol __ICFEDIT_region_ITCMRAM_start__ = 0x00000000;
define symbol __ICFEDIT_region_ITCMRAM_end__   = 0x0000FFFF;
/*-Sizes-*/
define symbol __ICFEDIT_size_cstack__ = 0x400;
define symbol __ICFEDIT_size_heap__ = 0x200;
/**** End of ICF editor section. ###ICF###*/


define region ROM_region      = mem:[from __ICFEDIT_region_ROM_start__   to __ICFEDIT_region_ROM_end__];
define region RAM_region      = mem:[from __ICFEDIT_region_RAM_start__   to __ICFEDIT_region_RAM_end__];
define region ITCMRAM_region  = mem:[from __ICFEDIT_region_ITCMRAM_start__ to __ICFEDIT_region_ITCMRAM_end__];

define root section aes_block_padding with alignment=32
{
udata8 "Force Alignment";
pad_to 32;
};

define block CSTACK    with alignment = 8, size = __ICFEDIT_size_cstack__   { };
define block HEAP      with alignment = 8, size = __ICFEDIT_size_heap__     { };

initialize by copy { readwrite };
do not initialize  { section .noinit };

place at address mem:__ICFEDIT_intvec_start__ { readonly section .intvec };

place in ROM_region   { readonly, last section aes_block_padding };
place in RAM_region   { readwrite,
                        block CSTACK, block HEAP };
```

M7 的 linker 文件中的 aes_block_padding 不是必须的，因为按照本示例中的 memory 布局定义，CM7 内核的应用位于 M4 之前，而 M4 的起始地址对齐到 0x10000，而且最终两个核的 binary 将会合并为一个 image 进行签名校验，因此只需要 CM4 内核的 linker 文件中有填充的部分就可以了。如果 CM7 内核的应用将作为一个单独的 image 进行签名加密，那么这段填充也需要。

如果使用的工具链是 STM32CubeIDE，那么 linker file 也需要做相应修改，使用 Linker Common 共享链接文件中定义的新的起始地址和大小，具体修改这里不详述。

另外，无论使用哪个工具链，只要 Linker 文件中的 initvector 地址修改了，system_stm32h7xx.c 文件中的 VTOR 也需要相应修改。细节这里不赘述。

5.2. 双核应用的签名加密打包

在本示例中，我们采用的方案是把 CM7 与 CM4 内核的应用合并后做签名加密，比较简单的方式是各自工程中生成 hex 输出，通过工具合并后生成二进制文件，然后进行签名加密。各个编译器工具链生成 hex 输出的方法不同，这里不赘述。

SBSFU 参考实现包中的 postbuild 脚本缺省需要使用 App 的 elf 进行打包，这种方式在同时有 CM7 和 CM4 两个应用 elf 的情况下不适用，所以这里不直接使用原有的 postbuild 脚本进行 App 的签名加密打包，而基于已有脚本进行修改，这个脚本也不必区分应用的 binary 是从那种工具链编译获得的。

脚本需要的输入参数：

- 参数 1：App 工程目录。脚本运行的结果会生成一些中间文件和最终输出文件，例如双核 App 合并的.bin 二进制文件、用于升级的.sfb 二进制文件以及 SBSFU 与 App 合并的用于直接烧写入 Flash 的最终的 hex 文件。这个参数的目的是获得输出文件存放目录的位置。
- 参数 2 和 3：CM7 内核 App 与 CM4 内核 App 的 hex 文件。
- 参数 4：fwid，通常为 1。
- 参数 5：version，App 的版本。
- 参数 6：SBSFU 二进制文件。用于与签名加密后的双核 App 合并成一个可以直接烧写的文件，安全启动代码二进制文件从 SecureBoot 参考实现的 xxx_SBSFU 工程中编译得到。
- 参数 7：SBSFU 参考实现工程所在的目录。脚本从这个目录中获取签名加密使用的密钥等数据，脚本也会引用这个目录中的签名加密打包工具 prepareimage 脚本或.exe。

脚本的主要操作步骤包括：

1. 使用 srec_cat 工具合并 CM7 内核 App 与 CM4 内核 App 的两个单独的 hex，并输出为一个合并的 binary 文件（这里的地址 offset 等是按照前文假设的 Memory 地址布局来写的，也就是假设 M7 的启动地址在 0x08020400）

```
%SREC_CAT% %hex_CM7% -intel %hex_CM4% -intel -o %userAppBinary%\tmp.hex -intel
%SREC_CAT% %userAppBinary%\tmp.hex -intel -offset -0x08020400 -o %app_bin% -binary
```

   这个步骤完成后，我们将得到合并后的双核 App 的二进制文件%app_bin%

2. 使用 prepareimage 工具对 App 二进制文件进行加密签名打包。（这里假设 Crypto 方案选择的是缺省的 ECDSA 签名+AES CBC 加密，如果选择其他模式，具体的步骤可能不同，可以参考 SECoreBin 目录下对应 Crypto 方案的脚本模板。）

```
::对合并后的 App 二进制文件加密
set "command=%python%%prepareimage% enc -k %oemkey% -i %iv% %app_bin%  %sfu%
> %projectdir%\output.txt 2>&1"
%command%
...
::生成合并后的 App 二进制文件的 SHA256 HASH
set "command=%python%%prepareimage% sha256 %app_bin% %sign%
>> %projectdir%\output.txt 2>&1"
%command%
...
::对合并后的 App 二进制文件签名并生成用于升级的.sfb 文件
set "command=%python%%prepareimage% pack -m %magic% -k %ecckey%  -r 28 -
v %version% -i %iv% -f %sfu% -t %sign% %sfb% -o %offset%
>> %projectdir%\output.txt 2>&1"
%command%
...
::对合并后的 App 二进制文件签名并生成固件头 header，用于大 binary 的合并
set "command=%python%%prepareimage% header -m %magic% -k  %ecckey% -r 28 -
v %version%  -i %iv% -f %sfu% -t %sign% -o %offset% %headerbin%
>> %projectdir%\output.txt 2>&1"
%command%
```

这个几个步骤完成后，我们将得到可以用于安全升级的.sfb 文件%sfb%，加密后的 App 二进制文件%app_bin%，以及 App 的固件头二进制文件%headerbin%用于最后与加密后的 App 及 SBSFU 合并为一个完整 image 文件。

3. 使用 srec_cat 工具将 SBSFU，固件头以及合并后的 App 合成一个最终的二进制文件，用于直接烧入 Flash 启动运行（这里的地址 offset 等是按照前文假设的 Memory 地址布局来写的，也就是假设 ActiveSlot 的起始地址为 0x08020000，这里放固件头，CM7 的启动地址在 0x08020400，从这里开始放加密后的固件二进制文件）

```
%SREC_CAT% %sbsfubin% -Binary -offset 0x08000000 -fill 0xFF 0x08000000
0x08020000 %headerbin% -Binary -offset 0x08020000 %app_bin% -Binary -offset
0x08020400 -o %merged_hex%
```

运行脚本之后会在所给定的第一个参数的目录下的 Binary 目录中生成以下几个文件

5.3. 验证带安全启动的双核应用

5.3.1. 验证安全启动校验过程

将 SBSFU_App.hex 通过 STM32CubeProgrammer 烧写入 Nucleo-H755ZI-Q 开发板，重新上电启动后，则会在上位机的串口工具上看到类似图 18 的打印信息，同时会看到红色 LED 闪烁，说明安全启动验证成功，并且 H7 上的两个核的应用程序都成功运行。

图18.　　安全启动运行 H7 双核应用的串口打印

5.3.2. 验证安全更新

按住开发板的蓝色用户按键并复位，此时进入 Bootloader 的固件下载模式，如图 19。

图19.　　进入 Bootloader 的下载模式

通过 TeraTerm 串口工具的 Ymodem 协议，发送 App.sfb 文件，执行固件安装过程，则会看到如图 20 的下载过程，下载完成后，SBSFU 会自动重启并校验新版本，校验通过后会用下载的版本代替 ActiveSlot 的内容完成固件安装过程，再次重启后本次下载的 App.sfb 中的 App 版本将运行，如图 21 所示。

图20.　　Bootloader 下载 App.sfb

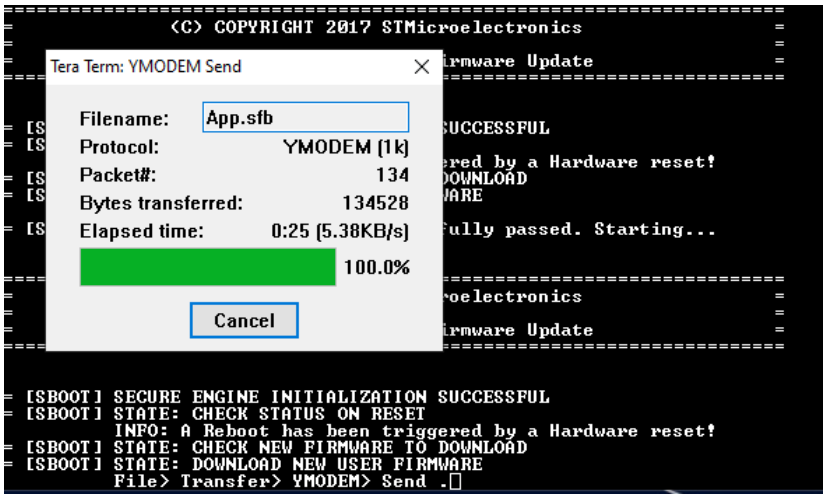
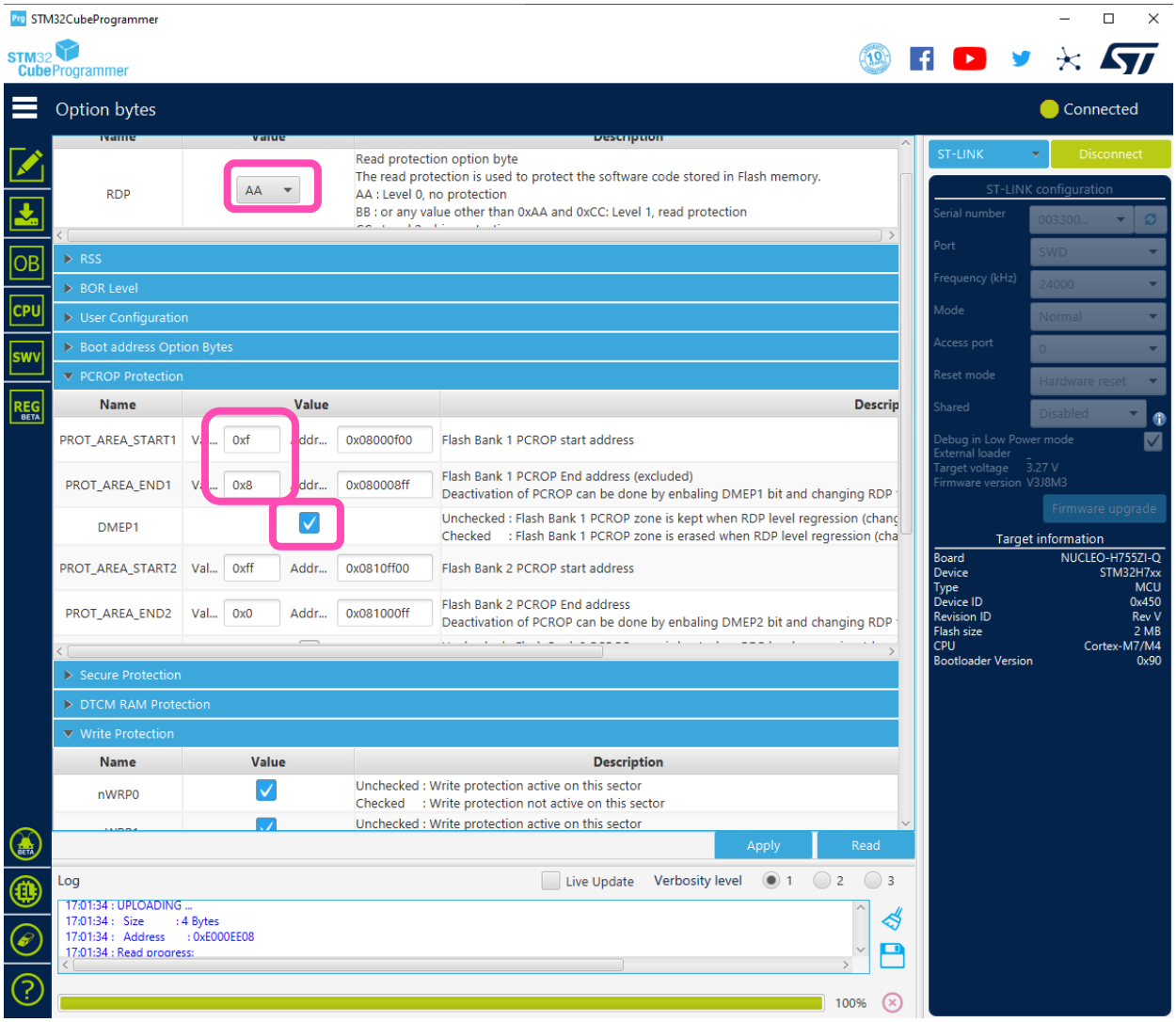


图21.　　SBSFU 完成固件更新过程并启动安装的版本

这里需要注意的一点是，如果使用缺省的 SBSFU 开发阶段安全配置，SBSFU 启动时会检查选项字节中的安全配置，并自动设置 RDP，PCROP 和 WRP 等选项字节，此时如果需要重新烧写内部 Flash，需要先回退这些配置选项，否则内部 Flash 无法通过调试端口访问。回退 RDP 和 PCROP 时，需要在 RDP 级别 1 的状态下，将 RDP 设置为 0xAA，同时将 PCROP 区的起始地址设置为大于结束地址的值，保持 DMEPx 的勾选状态，然后再 Apply，如图 22 示例。

图22. 使用 STM32CubeProgrammer 同时回退 RDP 和 PCROP



6. 小结

X-CUBE-SBSFU 是基于 STM32 MCU 的安全启动和安全更新参考实现，支持众多 STM32 系列，本文总结了一些有关如何将 SBSFU 参考实现移植到另外一个 STM32 平台以及如何与客户自己的应用程序进行集成的技巧和注意事项。X-CUBE-SBSFU 参考实现包含完整的参考代码以及配套的工具，并以源码形式提供给客户，具备非常大的灵活性，用户既可以直接使用该软件包与应用程序进行集成，为其系统快速添加安全启动和安全更新功能；也可以仅以此作为参考，重新编写自己的安全启动和安全更新实现。

本文以 STM32H7 平台为例，介绍了从单核 STM32H7 的 SBSFU 参考实现移植到双核 STM32H7 开发板，并与应用程序集成的过程，涉及到双核安全启动的方案选择，硬件平台的移植过程和双核应用的集成与验证等。用户既可以参考这个示例，也可以自行根据应用的实际需求选择不同的启动方案及实现。

参考文献

【如有， 请注明；否则，请注明：无】

文件编号	文件标题	版本号	发布日期
UM2262	Getting started with the X-CUBE-SBSFU STM32Cube Expansion Package	Rev9	June 2021
AN5056	Integration guide for the X-CUBE-SBSFU STM32Cube Expansion Package	Rev7	July 2021

文档中所用到的工具及版本

STM32CubeProgrammer v2.8.0

版本历史

日期	版本	变更记录
2021 年 12 月	1.0	首版发布

重要通知 – 请仔细阅读

意法半导体公司及其子公司 （“ST”）保留随时对 ST 产品和 / 或本文档进行变更的权利，恕不另行通知。买方在订货之前应获取关于 ST 产品的最新信息。 ST 产品的销售依照订单确认时的相关 ST 销售条款。

买方自行负责对 ST 产品的选择和使用， ST 概不承担与应用协助或买方产品设计相关的任何责任。

ST 不对任何知识产权进行任何明示或默示的授权或许可。

转售的 ST 产品如有不同于此处提供的信息的规定，将导致 ST 针对该产品授予的任何保证失效。

ST 和 ST 徽标是 ST 的商标。若需 ST 商标的更多信息，请参考 *www.st.com/trademarks*。所有其他产品或服务名称均为其 各自所有者的财产。

本文档是 ST 中国本地团队的技术性文章，旨在交流与分享，并期望借此给予客户产品应用上足够的帮助或提醒。若文中内容存有局限或与 ST 官网资料不一致，请以实际应用验证结果和 ST 官网最新发布的内容为准。您拥有完全自主权是否采纳本文档（包括代码，电路图等）信息，我们也不承担因使用或采纳本文档内容而导致的任何风险。

本文档中的信息取代本文档所有早期版本中提供的信息。