通过 KEIL 制作 QSPI 接口的外部 Flash 下载算法

关键字：KEIL, QSPI, 下载算法

1. 引言

随着用户的应用越来越复杂以及 GUI 等需要大存储空间的需求越来越多，很多时候我们需要将代码或数据放在外扩的 Flash 存储空间。但是这样存在一个外部 Flash 烧写的问题，尤其是在应用调试时，需要将代码或数据烧录到外部 Flash。如果调试工具不能够一键烧录，势必会给调试带来诸多的麻烦。本文以 STM32H750 芯片为例，介绍通过 KEIL 制作 QSPI 接口的外部 Flash 下载算法的方法。

2. MDK 下载算法基础知识

FLASH 编程算法是一种用于擦除应用程序或将应用程序下载到 Flash 的程序代码。MDK 本身支持的各种器件都自带下载算法，存放在 MDK 各种器件的软件包里面，以 STM32H7 为例，算法存在于 Keil\STM32H7xx_DFP\2.6.0\CMSIS\Flash（软件包版本不同，数字 2.6.0 会不同）。但是，只有 STM32 官方部分开发板提供了算法工程，大多数用户开发板都需要自己编写下载算法。不同的 QSPI-FLASH 的驱动略有差别，因此下载算法也不同。所以，掌握通用的下载算法制作步骤很重要。

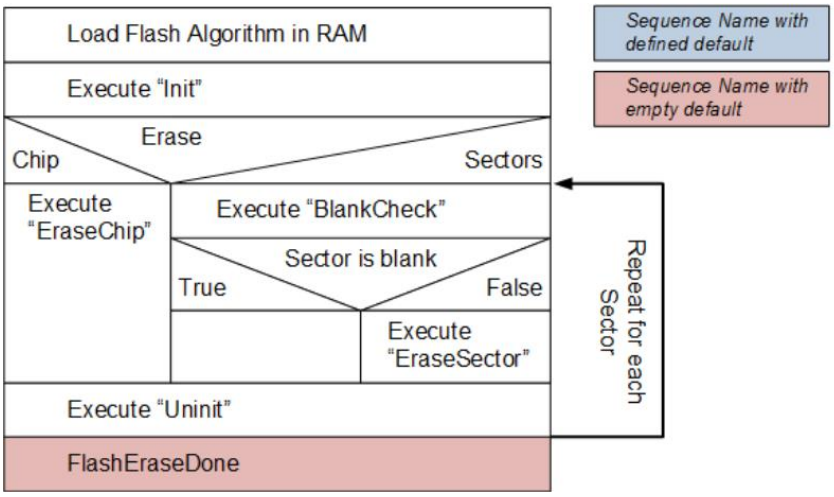
MDK 通过创建一批与地址信息无关的函数，实现的功能主要有初始化，擦除，编程，读取，校验等，然后在 MDK 调试下载阶段，会将算法文件加载到芯片的内部 RAM 里面（加载地址可以通过 MDK 设置），然后 MDK 通过与这个算法文件的交互，实现程序下载，调试阶段数据读取等操作。

3. 程序擦除操作执行流程

如图 1（Algorithm Functions (keil.com)）所示，可以看出 Flash 的擦除过程包括以下几个步骤：

a) 加载算法到 RAM（一般指片内 SRAM）
b) 执行初始化函数 Init
c) 执行擦除操作，根据用户的 MDK 配置，这里可以选择整个芯片擦除或者扇区擦除。
d) 擦除操作结束后执行 Uninit 函数

图1. 程序擦除操作执行流程



4. 程序烧录操作执行流程

如图 2（Algorithm Functions (keil.com)）所示，程序烧录执行流程包括以下步骤：

a)对所有的 AXF 文件做 init 初始化（AXF 是 MDK 生成的可执行文件，也就是需要烧录和调试的代码）

b)查看烧写算法 FLM 文件是否存在，如果不存在，则操作结束；如果存在，则继续

c)加载算法文件到 RAM 中

d)执行初始化函数 Init

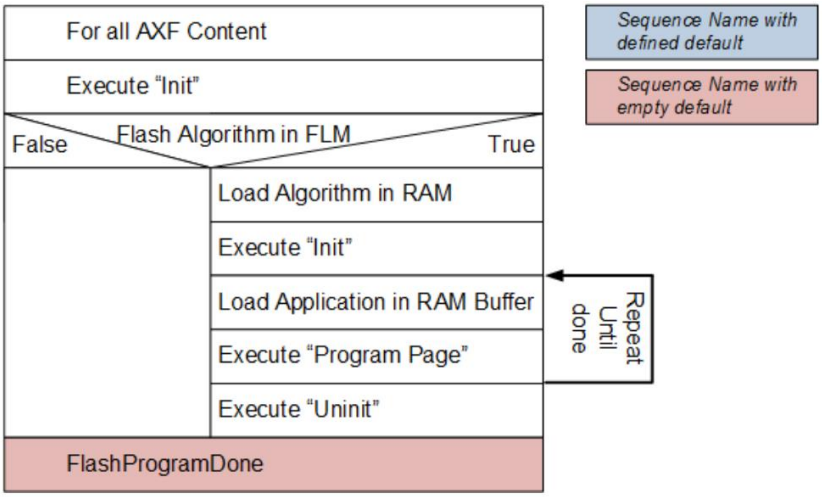
e)加载用户程序代码至 RAM 中

f)执行写入编程函数 Program Page

g)执行 Uninit 函数

f)操作结束

图2. 程序烧录操作执行流程



从以上操作过程中可以看出，我们的用户代码是先放在 RAM 中缓存的，然后 IDE 通过调用 RAM 中的 FLASH 写入函数 Program Page 将代码写入到 Flash 中，完成代码的烧录。

5. 程序校验操作执行流程

程序校验操作大致流程如图 3（Algorithm Functions (keil.com)）所示，其中校验要用到 MDK 生成的 axf 可执行文件。 校验就是 把 axf 文件中的程序和实际下载到芯片的程序读出来做比较。

a)查看烧写算法 FLM 文件是否存在，如果不存在，则操作失败；如果存在，则继续

b)加载算法到 RAM 中

c)执行初始化 Init

d)检查校验算法是否存在：

如果存在，加载应用程序到 RAM 中，然后执行校验算法函数

如果不存在，计算 CRC，将芯片中读取出来的数据和 RAM 中加载应用计算输出的 CRC 值做比较。

e)执行 Uninit 函数。

f)替换 BKPT（ BreakPoint 断点指令）为 B. 死循环指令

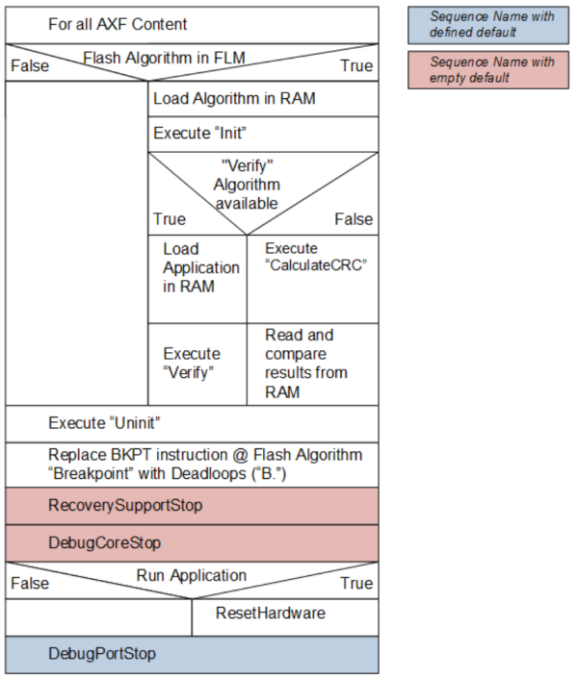
h)执行 RecoverySupportStop， 恢复支持停止。

i)执行 DebugCoreStop，调试内核停止

g)运行应用，若运行成功，则硬件复位

k)操作完成，停止调试端口

图3. 程序校验操作执行流程



6. MDK 创建下载算法过程通用步骤

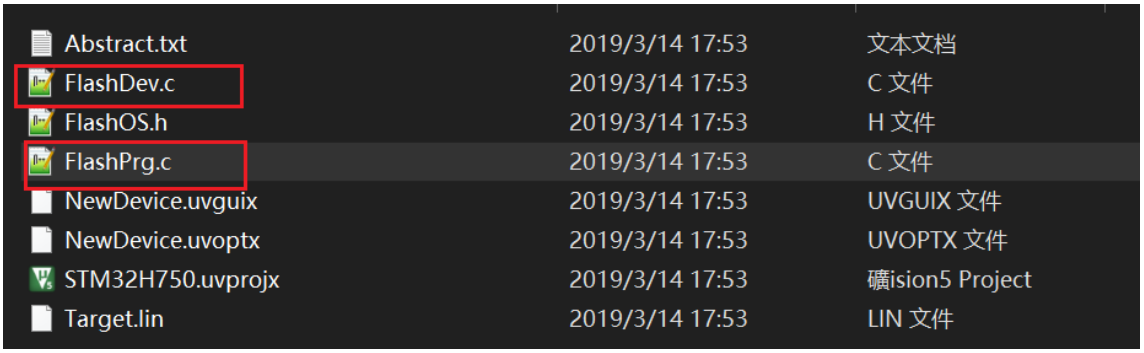
6.1 使用 MDK 提供的通用模板

模板路径：

C:\Keil_v5\ARM\Packs\ARM\CMSIS\5.6.0\Device_Template_Flash

请找到自己 MDK 的安装路径，找到后将工程拷贝出来。然后将以下两个文件的只读属性取消。FlashDev.c 是 flash 信息描述文件，需要根据自己的 flash 实际情况修改。FlashPrg.c 是具体擦除、写入、校验等接口函数实现文件，需要自己根据实际情况进行开发。

图4. 程序烧录操作执行流程

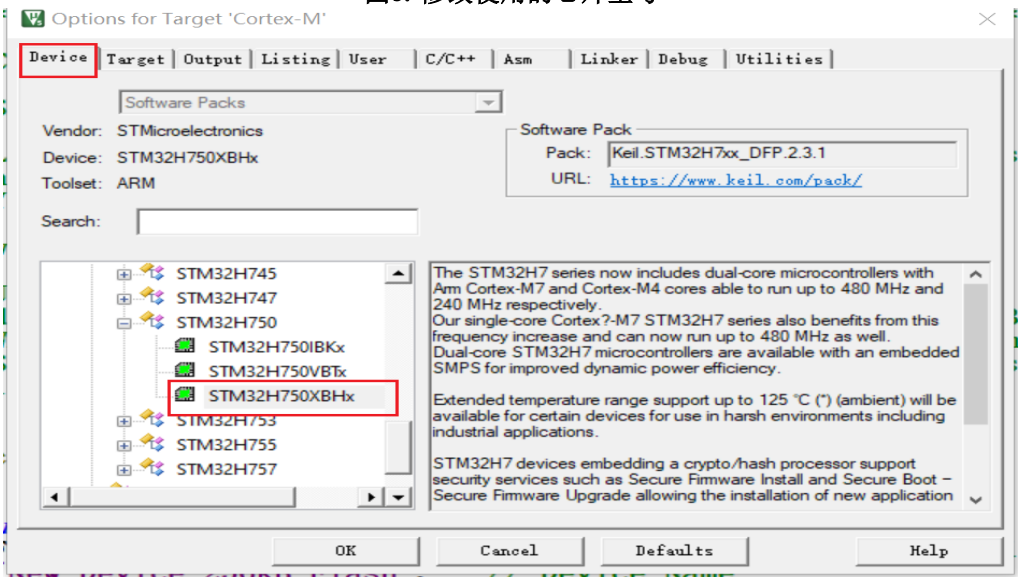


同时 MDK 提供的工程模板原始名字是 NewDevice.uvprojx， 大家可以根据自己的需要做修改。

6.2 MDK 工程设置

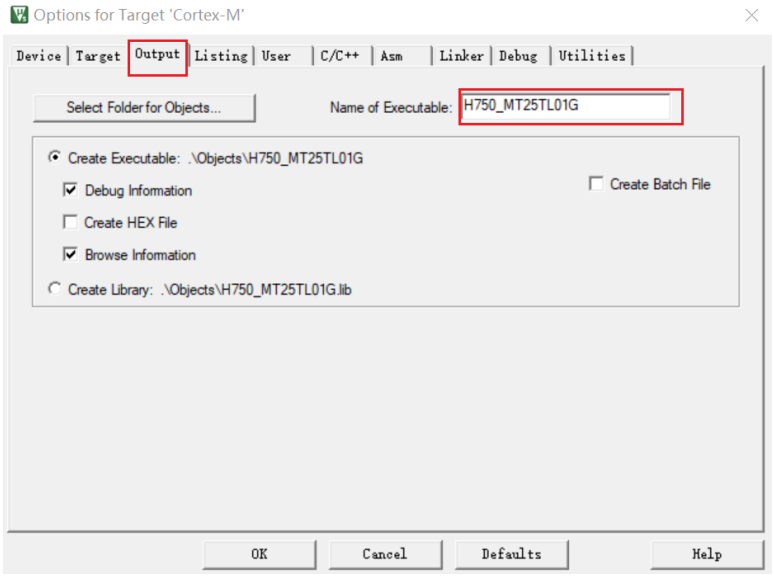
6.2.1. 修改使用的芯片型号

图5. 修改使用的芯片型号



6.2.2. 修改输出算法文件名

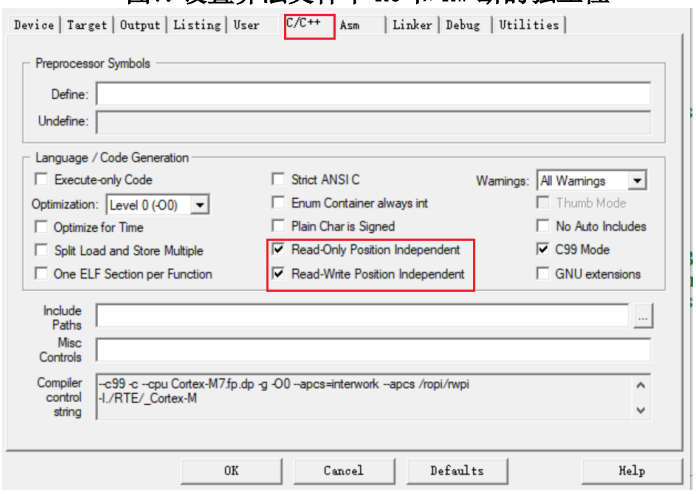
图6. 修改输出算法文件名



这个名字是方便用户查看的， 比如设置为 stm32h7，那么输出的算法文件就是 stm32h7.flm。

6.2.3. 设置算法文件中 RO 和 RW 段的独立性

图7. 设置算法文件中 RO 和 RW 断的独立性



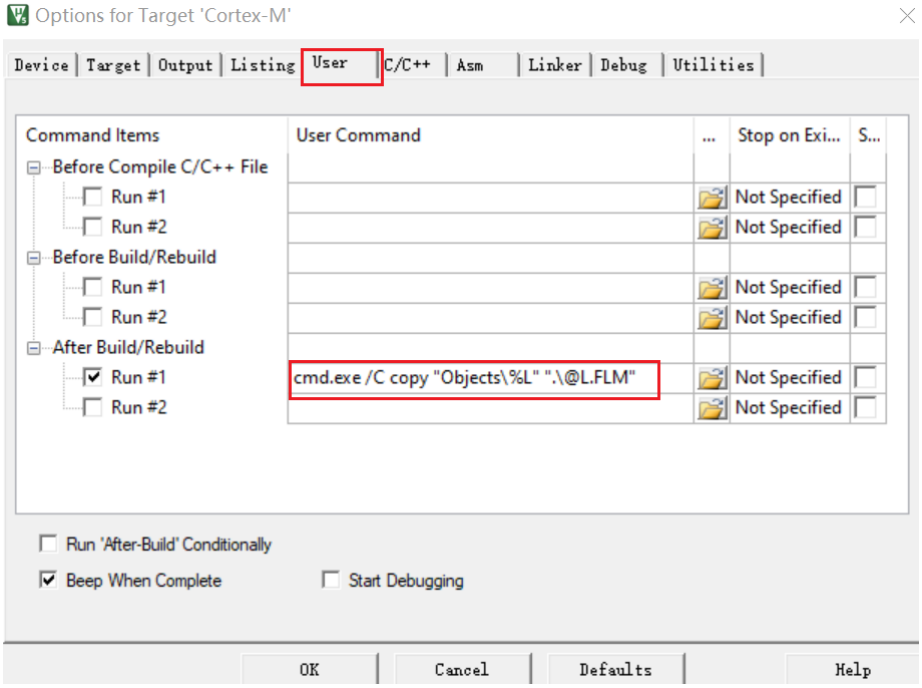
选择“ ROPI”以及 “RWPI”选项， 可以避免用户不得不将代码加载到内存中的特定位置。

6.2.4. 将程序可执行文件 axf 修改为 FLM

通过以下命令即可在编译后生成 FLM 文件。

```
cmd.exe /C copy "Objects\%L" ".\@L.FLM"
```

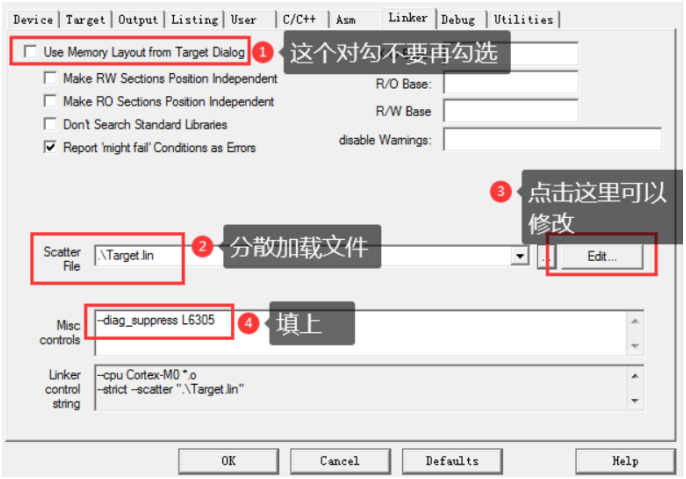
图8. 将程序可执行文件 axf 修改未 FLM 格式



6.2.5. 分散加载设置

分散加载文件 Target.lin 模板中有提供，--diag_suppress L6305 用于屏蔽 L6503 类型警告信息，设置了分散加载后， 此处的配置就不再起作用了。

图9. 分散加载设置 1



7. 添加 QSPI-FLASH 驱动及其有关库函数

使用 CubeMX 生成一个工程模板，然后将其中的 Drivers 文件夹拷贝到工程文件夹下，同时准备调试好可用的 Flash 驱动（QSPI 方式驱动），在 MDK 中新建如下分组，并添加对应的文件。

图10. 复制相关文件到工程文件夹

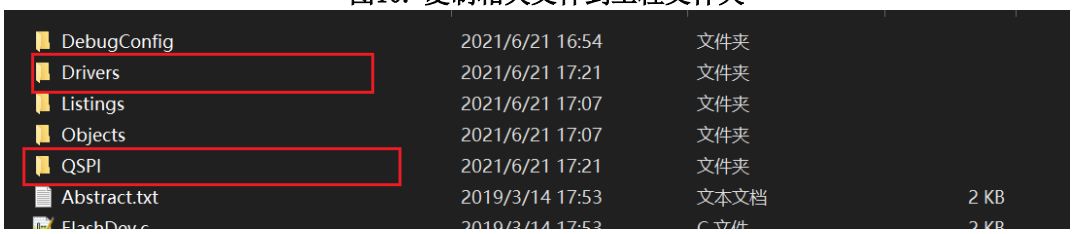
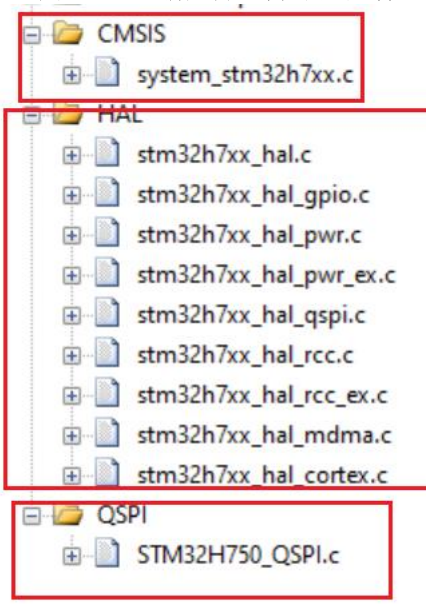


图11. MDK 新建分组并添加文件

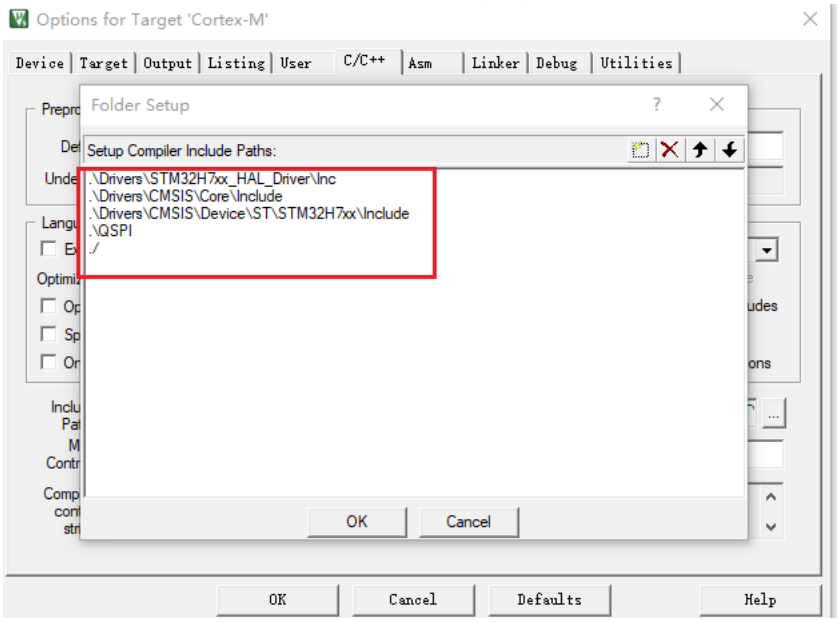


- CMSIS 分组中的 system_stm32h7xx.c 在 Drivers\CMSIS\Device\ST\STM32H7xx\Source\Templates 中
- HAL 分组中添加必要的 HAL 库文件
- QSPI 分组中添加 QSPI Flash 驱动

这里注意 stm32h7xx_hal_conf.h 这个头文件在 CubeMX 生成的生成工程\Core\Inc 文件夹中可以找到，同时要把使用的 HAL 库文件使能，而且要修改该头文件中 HSE_VALUE 的值，实际晶振多大，这里就改成多少。

最后，添加头文件路径，否则编译会报大量错误。头文件添加如图所示：

图12. 添加头文件路径

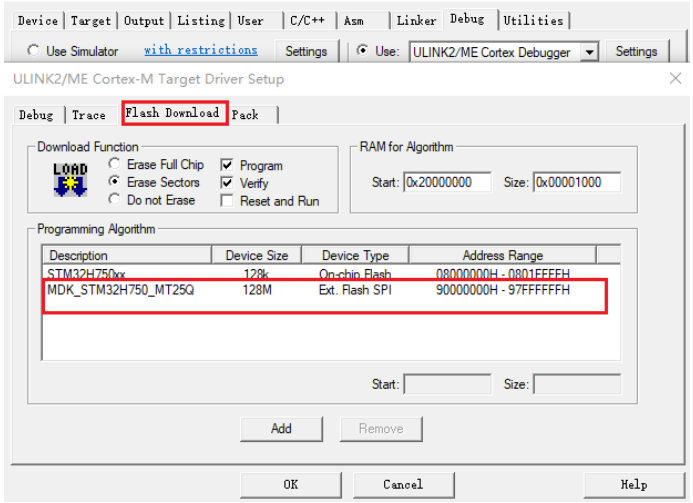


7.1. 修改 FlashDev.c

图13. 修改 FlashDev.c

```
struct FlashDevice const FlashDevice  =  {
    FLASH_DRV_VERS,                 /* 驱动版本，勿修改，这个是 MDK 定的 */
    "MT25TL01G_STM32H750DK",        /* 算法名，添加算法到 MDK 安装目录会显示此名字 */
    EXTSPI,                         /* 设备类型 */
    0x90000000,                     /* Flash 起始地址 */
    0x08000000,                     /* Flash 大小， 128MB */
    0x00001000,                      /* 编程页大小 4096 Bytes*/
    0x00,                           /* 保留，必须为 0 */
    0xFF,                           /* 擦除后的数值 */
    10000,                          /* 页编程等待时间 */
    6000,                           /* 扇区擦除等待时间 */

  // Specify Size and Address of Sectors
    0x20000, 0x000000,              // 扇区大小  128kB, 扇区数量 : 1024
    SECTOR_END
};
```

这里需要根据自己板子上 Flash 实际情况来修改，具体参数含义自行查阅相关 Flash 芯片手册，注意这里的 Flash 起始地址是内存映射模式下 Flash 的映射地址。算法名会反馈到下图这个地方。



7.2. 修改 FlashPrg.c

该文件是我们整个算法的核心文件，我们至少需要实现初始化、擦除、写入等函数接口。

7.2.1. 实现 Init 函数

```
int Init (unsigned long adr, unsigned long clk, unsigned long fnc) {

 if(Init_QSPI ()!=0)
   return 0;
 else
   return 1;
}
```

```
 int Init_QSPI()
{
      /* Zero Init structs */
  memset(&QSPIHandle, 0, sizeof(QSPIHandle));
  memset(&s_command, 0, sizeof(s_command));
  memset(&s_mem_mapped_cfg, 0, sizeof(s_mem_mapped_cfg));
  memset(&s_config, 0, sizeof(s_config));
  memset(&QspiInfo, 0, sizeof(QspiInfo));

  SystemInit();


  /* Configure the system clock to 80 MHz */
  SystemClock_Config();
/*Initialaize QSPI*/
  if(BSP_QSPI_Init() !=0)
    return 0;
    /*Configure the QSPI in memory-mapped mode*/
  if(BSP_QSPI_EnableMemoryMappedMode()!=0)
    return 3;


   return 1;
}
```

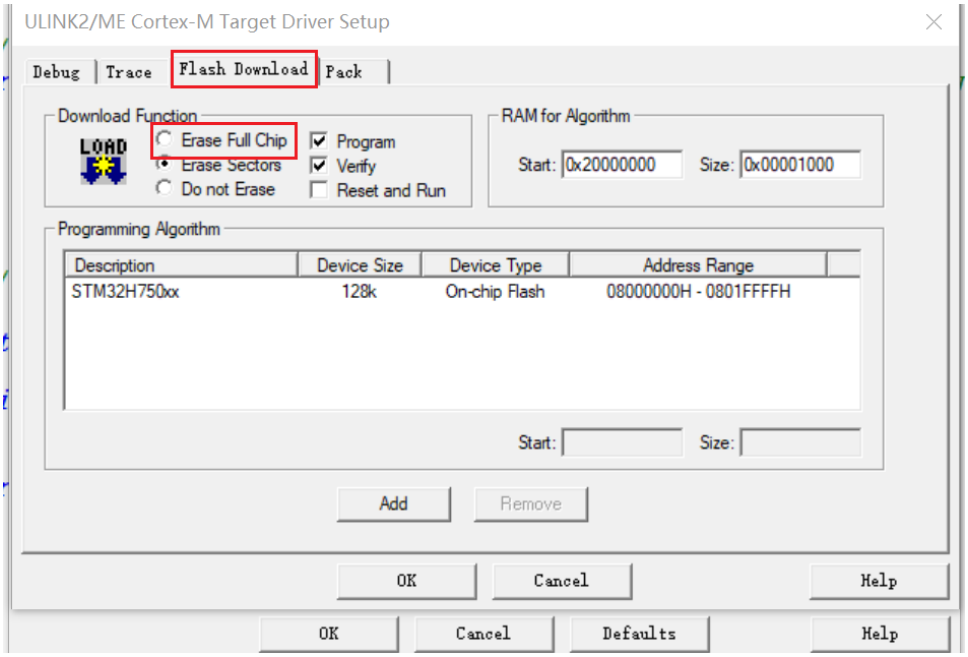
初始化完毕后要将其设置为内存映射模式，SystemClock_Config()可以从 CubeMX 生成的空白工程中拷贝过来，这是对系统外设时钟初始化函数，必须添加。

7.2.2. 实现 EraseChip 函数

```
int EraseChip (void) {

  if (MassErase()!=0)
       return 0;

  return 1;
}
```

```
int MassErase (){
    /*Initialaize QSPI*/
  BSP_QSPI_DeInit();
  BSP_QSPI_Init();
    /* Erases the entire QSPI memory*/
  BSP_QSPI_Erase_Chip();
    /*Reads current status of the QSPI memory*/
    while (BSP_QSPI_GetStatus()!=QSPI_OK){};

  return 1;
}
```

这是对整个芯片进行擦除的函数接口，如果勾选了如下的选项就会调用该函数，实际应用中不建议勾选，因为全片擦除比较耗时。

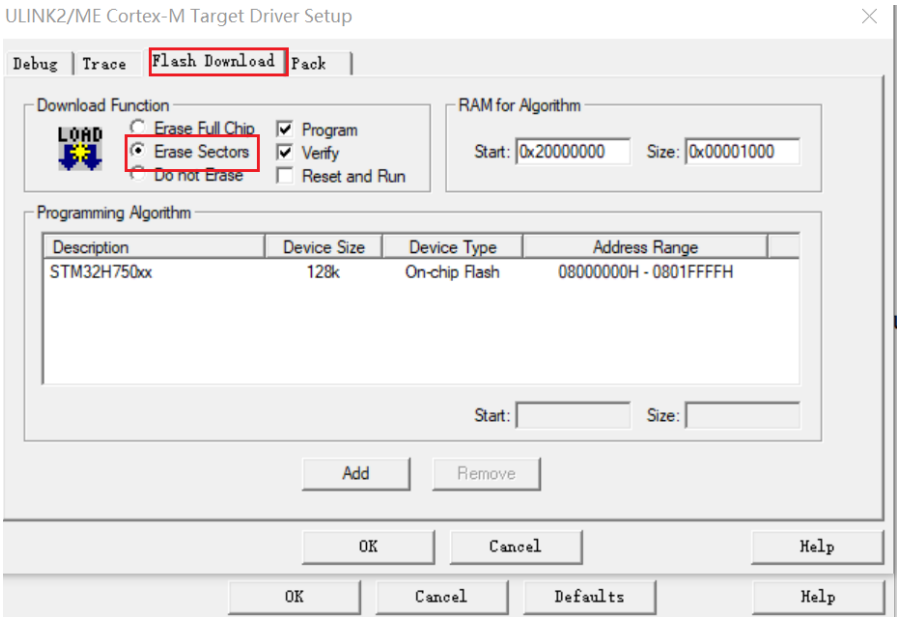


7.2.3. 实现 EraseSector 函数

```
int EraseSector (unsigned long adr) {

 int result = 0;
    uint32_t block_start=adr;
    uint32_t block_size;

 result = SectorErase ((uint32_t) block_start & 0xffffffff ,((uint32_t) block_start & 0xffffffff) + block_size);
 if (result == 1)
   return 0;
 else
   return result;
}
```

```
int SectorErase (uint32_t EraseStartAddress ,uint32_t EraseEndAddress)
{
  uint32_t BlockAddr;
  EraseStartAddress&=0x0FFFFFFF;
  EraseEndAddress &=0x0FFFFFFF;
  EraseStartAddress = EraseStartAddress -  EraseStartAddress % 0x20000;
    /*Initialaize QSPI*/
  BSP_QSPI_DeInit();
  BSP_QSPI_Init();
  while (EraseEndAddress>=EraseStartAddress)
  {
    BlockAddr = EraseStartAddress;
   /*Erases the specified block of the QSPI memory*/
        BSP_QSPI_Erase_Block(BlockAddr);
    /*Reads current status of the QSPI memory*/
    while (BSP_QSPI_GetStatus()!=QSPI_OK){};
      EraseStartAddress+=0x20000;
  }
 /*Configure the QSPI in memory-mapped mode*/
  BSP_QSPI_EnableMemoryMappedMode();
  return 1;
}
```

当勾选了如下图所示的选项时，则调用该函数



7.2.4. 实现 ProgramPage 函数

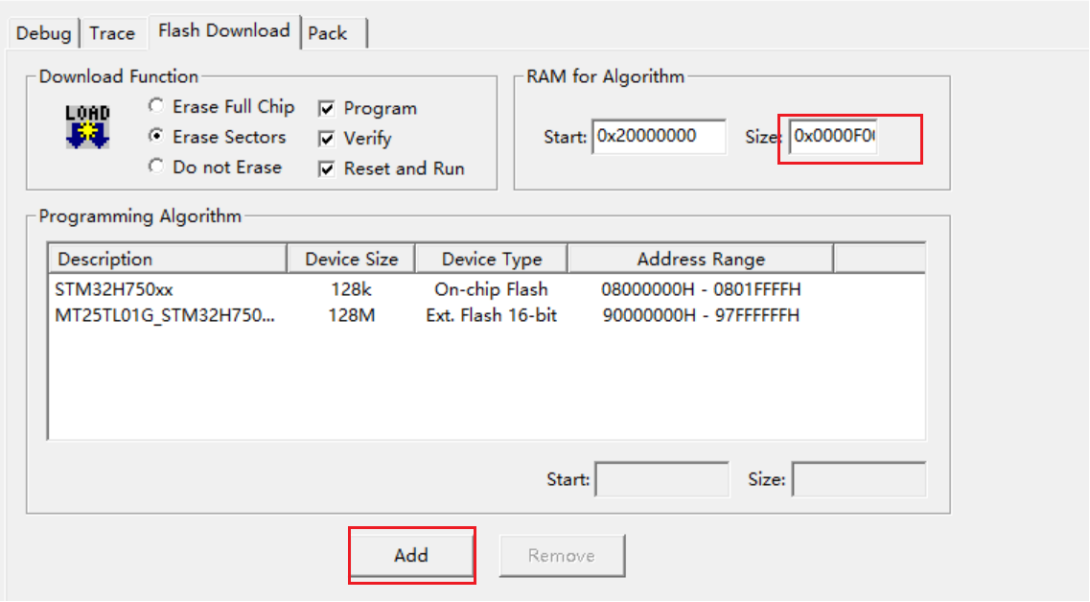
```
int ProgramPage (unsigned long block_start, unsigned long size, unsigned char *buffer) {

  if(Write(block_start ,size, buffer)!=0)
  return 0;
     else
  return 1;
}

              int Write (uint32_t Address, uint32_t Size, uint8_t* buffer)
             {

                 Address = Address & 0x0fffffff;
                   /*Initialaize QSPI*/
                 BSP_QSPI_DeInit();
                 BSP_QSPI_Init();
                  /*Writes data to the QSPI memory*/
                 BSP_QSPI_Write(buffer,Address, Size);

                return 1;
             }
```

7.3. 读取和检验函数

我们程序中未做读取函数，那么 MDK 会以总线方式进行读取，这是为什么要设置成内存映射模式的原因。如果程序中未做校验函数，那么 MDK 会读取数据做 CRC 校验。校验函数，实际上也可以不写。

8. 算法使用方法

算法生成后，将对应的算法 FLM 文件拷贝到 MDK 安装路径下 C:\Keil_v5\ARM\Flash



然后打开上图所示的设置界面，点击 ADD 按钮即可找到下载算法，然后添加即可。注意右上角 RAM for Algorithm 一定要设置大一点，否则可能算法运行失败。

9. 小结

本文介绍了基于 MDK 通用模板和已有工程制作 MDK 下载算法的过程。制作过程很容易造成失败，除了要注意工程中的相关设置以外，建议多参考 ST 官方固件包提供的 DEMO，有时可以直接找到对应型号 MCU 的烧写算法 DEMO，如果没有可以参考功能外设相似的 MCU 工程 DEMO，也可以在其 DEMO 上直接修改，主要是替换 QSPI 的驱动（引脚、命令等），还有就是注意自己板子 MCU 的时钟配置，建议直接拷贝可以成功运行工程中的时钟配置。

参考文献

文件编号	文件标题	版本号	发布日期
1	https://www.keil.com/pack/doc/CMSIS_Dev/Pack/html/algorithmFunc.html	1.6.1	2019 年 10 月 25 日

文档中所用到的工具及版本

Keil V5.36

版本历史

日期	版本	变更
2022 年 10 月 17 日	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。