如何在 IAR 和 KEIL 中计算 CRC 值

前言

市面上越来越多的产品对其使用提出了安全要求，如何避免使用过程中对操作者带来危险，或者降低这种危险发生的概率，这都是产品安全性需要考虑的。鉴于此，相关产品需要通过相关行业的安全认证才能生产上市。针对 CLASSB 以及 SIL 认证，ST 分别提供了对应的软件库以及应用手册来帮助客户开发有安全认证需求的产品。

我们在支持客户的过程中，发现客户经常在 Flash 校验这一块碰到问题。这里整理了遇到的常见问题，并基于 IAR，KEIL 这两种 IDE 介绍如何配置 FLASH 的 CRC 计算的方法。

Flash 自检的流程

Flash 的自检一般分为启动时自检和程序运行时自检两个阶段。不管是哪种自检，其思路都是：

在程序编译完成后，计算整个程序的 CRC 值，然后将这个 CRC 值添加到可执行文件末尾。再将带有 CRC 校验值的可执行文件烧录到 MCU 中。在程序启动后，由程序中的自检代码重新再根据当前 Flash 内容（不包括预存的 CRC 校验值）计算一次 CRC 值，再与之前预先计算并烧录到 Flash 中的 CRC 校验值进行比较，如果一致就通过检测。

这两个自检阶段的区别就是：

程序启动自检是一次性对整个实际 Flash 代码范围计算出最终的 CRC 值；而运行时的自检，为了不影响其他程序模块的运行，计算 CRC 的过程是分步进行的，每次计算一部分，分多次计算出最终的 CRC 值。

围绕 Flash 的自检所发生的问题可以归为两大类，一类是预先计算 CRC 值时和上电后计算 CRC 值的 Flash 范围设置是否一致；第二类就是预先计算 CRC 时和上电后计算 CRC 采用的 CRC 算法是否一致。

如何添加 CRC 值

下面我们主要介绍如何添加 CRC 校验值到可执行文件。

基于 IAR 环境

如果你使用 IAR IDE，那么添加 CRC 值的配置相对比较简单。通过配置 IAR 的 CRC 计算参数，自动对整个 FLASH 空间进行 CRC 计算，并将计算结果放到 FLASH 的末尾。

1. 修改 Link 文件，指定 CRC 值的存放位置

在 Link 文件中增加下面语句，指定 checksum 在 FLASH 中的存储位置。

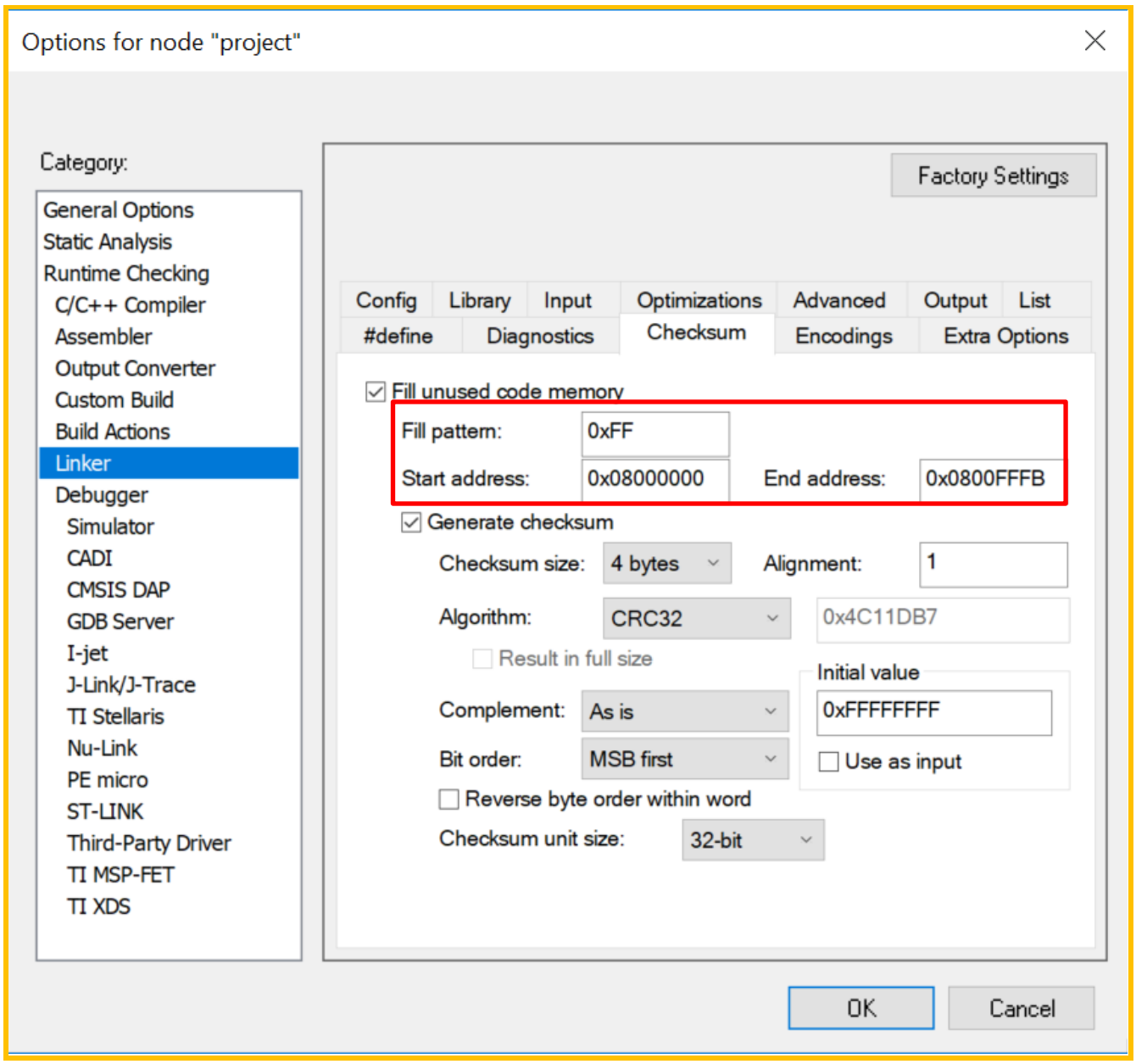
```
place at end of ROM_region { ro section .checksum };
```

该语句指定将 CRC 的值放在 FLASH 的末尾位置。是整个 FLASH 空间的末尾，不是应用程序的代码末尾。这样，CRC 值的位置就是固定的。不会随代码大小而变化。

在自检代码中，可以通过__checksum 读取 Flash 中保存的 CRC 校验值来与重新计算的 CRC 值进行比较。

2. 配置 Checksum 页面的参数

在 link 文件中指定了 checksum 的存储位置后，还要在工程配置菜单中，配置计算 CRC 值的范围和参数。见下图：



IAR 的 checksum 页面分为两个部分。

第一部分，也就是红线圈出的部分。定义了 FLASH 中需要计算 CRC 的范围和空闲字节填充值。这里注意要留出 flash 末尾存储 CRC 值的位置。

剩下的部分，就是对 checksum 计算参数的设定部分。

Checksum size ：选择 checksum 的大小（字节数）

Alignment：指定 checksum 的对齐方式。一般，处理器不支持非对齐访问时有用，不填的话默认 1 字节对齐。

Algorithm：选择 checksum 的算法

Complement：是否需要进行补码计算。选择“As is”就是不进行补码计算。

Bit order：位输出的顺序。MSB first，每个字节的高位在前。LSB first，每个字节的低位在前。

Reverse byte order within word： 对于输入数据，在一个字内反转各个字节的顺序。

Initial value： checksum 计算的初始化值

Checksum unit size ：选择进行迭代的单元大小，按 8-bit，16-bit 还是 32-bit 进行迭代。

3. STM32CRC 外设的配置

与上图 IARchecksum 的配置对应，STM32 CRC 外设可以按以下配置：

POLY= 0x4C11DB7（CRC32）

Initial_Crc = 0Xffffffff

输入/输出数据不反转

输入数据：根据实际 Flash 范围设定，留出 CRC 校验值的位置

CRC 外设初始化及计算代码：

```
/*##-1- Configure the CRC peripheral #####################################*/
  CrcHandle.Instance = CRC;

  /* The default polynomial is used */
  CrcHandle.Init.DefaultPolynomialUse    = DEFAULT_POLYNOMIAL_ENABLE;

  /* The default init value is used */
  CrcHandle.Init.DefaultInitValueUse     = DEFAULT_INIT_VALUE_ENABLE;

  /* The input data are not inverted */
  CrcHandle.Init.InputDataInversionMode  = CRC_INPUTDATA_INVERSION_NONE;

  /* The output data are not inverted */
  CrcHandle.Init.OutputDataInversionMode = CRC_OUTPUTDATA_INVERSION_DISABLED;

  /* The input data are 32 bits lenght */
  CrcHandle.InputDataFormat              = CRC_INPUTDATA_FORMAT_WORDS;

  if (HAL_CRC_Init(&CrcHandle) != HAL_OK)
  {
    /* Initialization Error */
    Error_Handler();
  }

  pdata = (uint32_t*)ROM_START;

  /*##-2- Compute the CRC of "aDataBuffer" ##################################*/
  uwCRCValue = HAL_CRC_Calculate(&CrcHandle, pdata, ROM_SIZEinWORDS);
```

基于 ARM keil MDK 环境

KEIL 没有提供直接生成 CRC 值的功能，所以需要借助外部的工具计算 CRC 值，然后添加到可执行文件的末尾。在 X-CUBE-CLASSB 软件中提供了 bat 文件，它会利用外部工具 Srecord 来生成整个 Flash 的 CRC 校验码并放在文件末尾。这个工具同样也可以和标准外设库的 ClassB 库一起用。下面我们就来看看如何在 KEIL 工程中利用 Srecord 工具来添加 CRC 值。

1. 安装 Srecord 工具

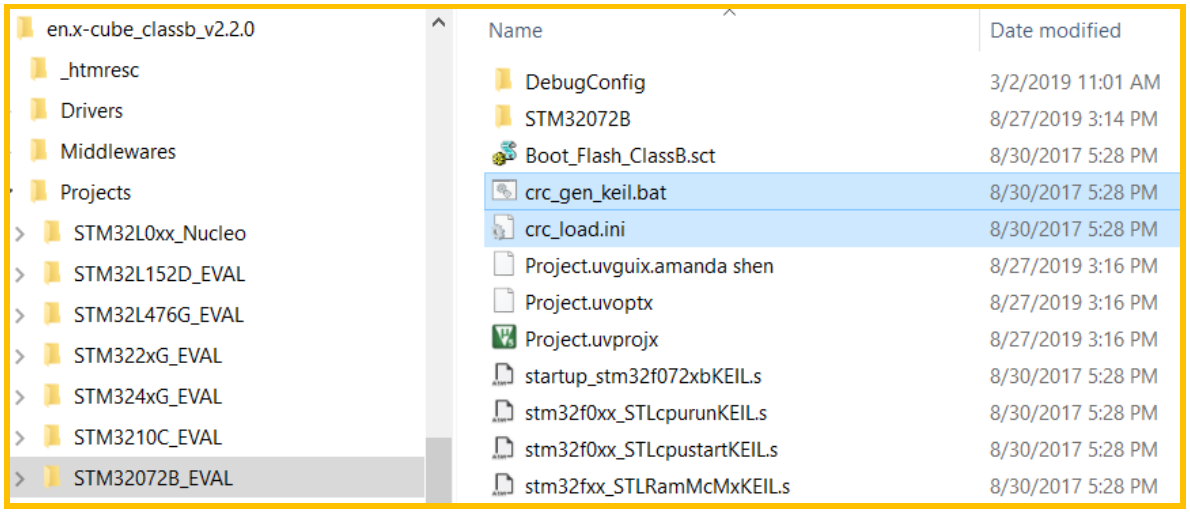
下载 Srecord 工具（http://srecord.sourceforge.net ）。将 srec_cat.exe，srec_cmp.exe，srec_info.exe 拷贝到 C:\SREC（自己新建）目录下。

2. 添加 crc_gen_keil.bat 及 crc_load.ini 文件到 KEIL 工程同级目录下

打开 X-CUBE-CLASSB 软件包中的任意 KEIL 工程目录，将其中 crc_gen_keil.bat 及 crc_load.ini 文件拷贝到自己的 KEIL 工程目录下。

crc_gen_keil.bat：利用外部工具 Srecord 来生成整个 Flash 的 CRC 校验码并放在文件末尾。

crc_load.ini：这个文件调试时有用，用来配置调试时导入带 CRC 校验码的 HEX，避免对 FLASH 检测失败导致程序无法正常运行。



这两个文件中的内容也需要根据新工程路径进行修改：

- 将 crc_gen_keil.bat 中的 TARGET_NAME 和 TARGET_PATH 改成跟新工程一致。否则不能成功的自动生成带 CRC 校验值的 HEX 文件。

```
@echo off
ECHO Computing CRC
ECHO -------------------------------------
REM Batch script for generating CRC in KEIL project
REM Must be placed at MDK-ARM folder (project folder)

REM Path configuration
SET SREC_PATH=C:\SREC
SET TARGET_NAME=STM32072B_EVAL
SET TARGET_PATH=STM32072B
SET BYTE_SWAP=1
SET COMPARE_HEX=1
SET CRC_ADDR_FROM_MAP=1
REM Not used when CRC_ADDR_FROM_MAP=1
SET CRC_ADDR=0x08007ce0

REM Derived configuration
SET MAP_FILE=%TARGET_PATH%\%TARGET_NAME%.map
SET INPUT_HEX=%TARGET_PATH%\%TARGET_NAME%.hex
SET OUTPUT_HEX=%TARGET_PATH%\%TARGET_NAME%_CRC.hex
SET TMP_FILE=crc_tmp_file.txt
```

- Crc_load.ini 文件中的路径和文件也要和实际的一致

crc_load.ini

```
1   LOAD "STM32072B\\STM32072B_EVAL_CRC.hex"
```

3. 添加定义 CRC 校验码存储区域

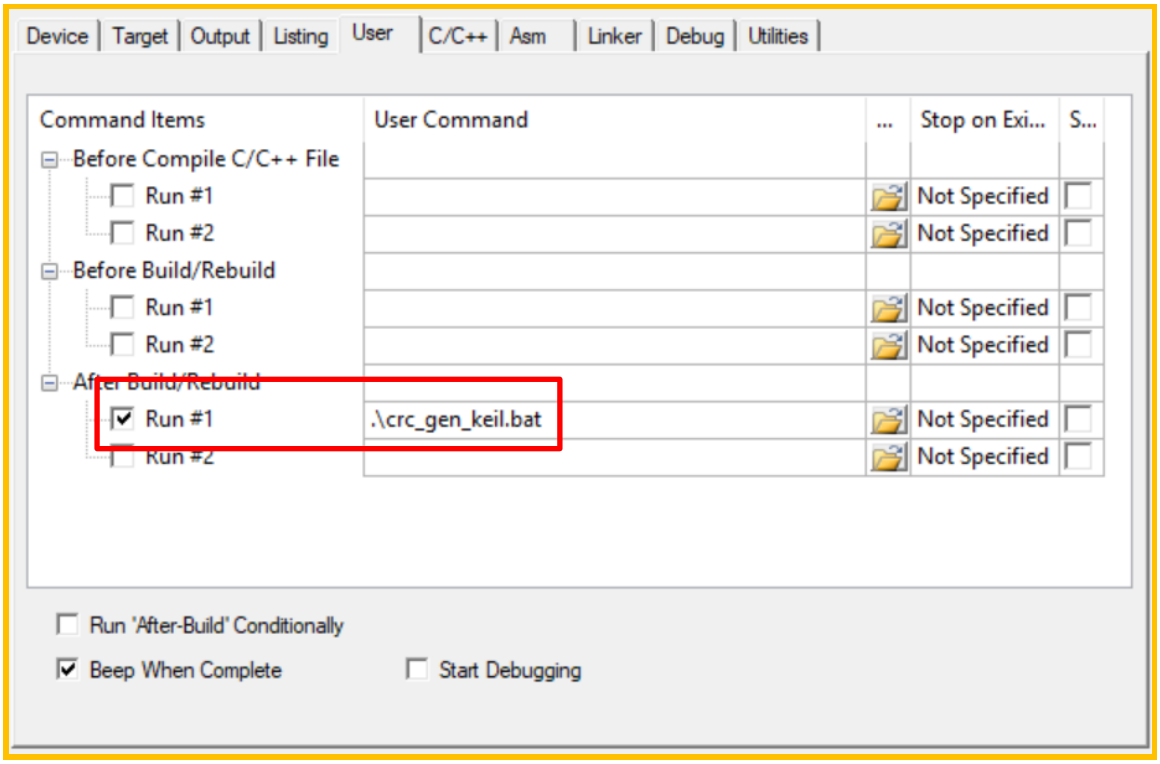
```
;*******************************************************************************
; User Checksum - must be placed at the end of memory
;*******************************************************************************
                AREA    CHECKSUM, DATA, READONLY, ALIGN=6
                EXPORT  __Check_Sum

; Alignement here must correspond to the size of tested block at FLASH run time test
                ALIGN

__Check_Sum     DCD     0x3D334398;                 ; Check sum computed externaly
```

这个值可先随意写一个，后面会通过工具直接修改 HEX 文件中的对应值

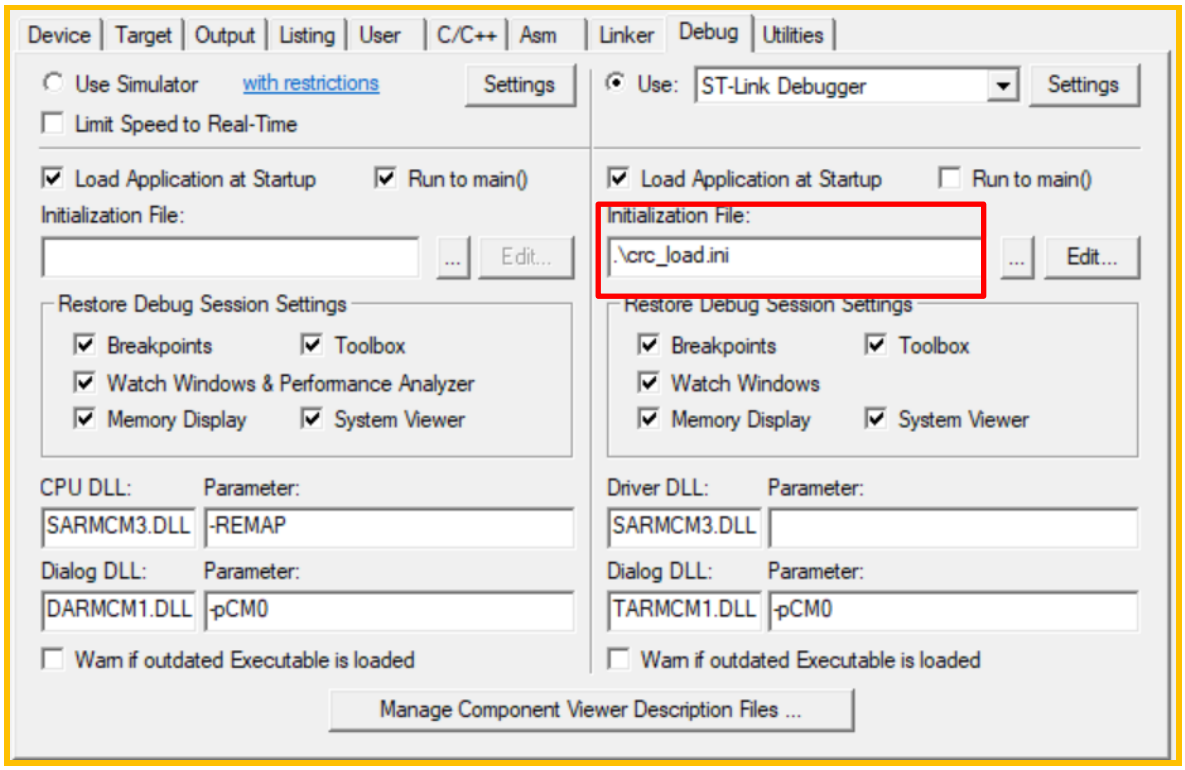
在分散加载文件中将 CHECKSUM 指定在代码的末尾。和 IAR 不同的是，通过在分散加载文件中+last 指定 checksum 的位置，它不是将其固定放在整个 flash 地址的末尾，而是放在实际代码的末尾。

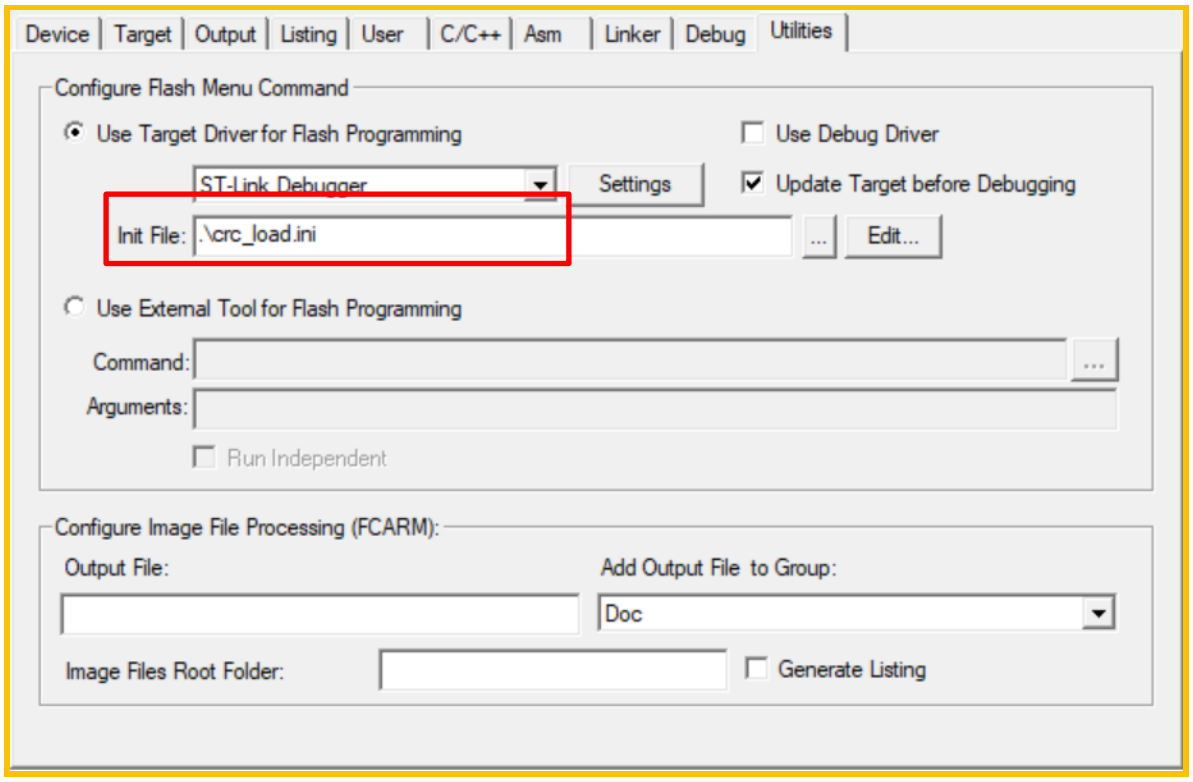
```
LR_IROM1 0x08000000 0x00010000  {    ; load region size_region
  ER_IROM1 0x08000000 0x00010000  {  ; load address = execution address
    *.o (RESET, +First)
    *(InRoot$$Sections)
    .ANY (+RO)
    *.o (CHECKSUM, +Last)
  }
```

4. 添加外部命令让 KEIL 在编译结束后，自动生成一个带 CRC 校验值的 HEX 文件



定义 debug 和 flash download 使用的 HEX 文件路径，使用带 CRC 校验值的 HEX 文件。





5. STM32CRC 外设的配置

这里需要注意, 从 X-CUBE-CLASSB 的软件包里拷贝出的 crc_gen_keil.bat 文件，里面的 BYTE_SWAP 设为 1， 也就是它在计算 CRC 值的时候，输入的数据，一个字内按字节颠倒顺序。

```
@echo off
ECHO Computing CRC
ECHO -------------------------------------
REM Batch script for generating CRC in KEIL project
REM Must be placed at MDK-ARM folder (project folder)

REM Path configuration
SET SREC_PATH=C:\SREC
SET TARGET_NAME=STM32072B_EVAL
SET TARGET_PATH=STM32072B
SET BYTE_SWAP=1
SET COMPARE_HEX=1
SET CRC_ADDR_FROM_MAP=1
REM Not used when CRC_ADDR_FROM_MAP=1
SET CRC_ADDR=0x08007ce0

REM Derived configuration
SET MAP_FILE=%TARGET_PATH%\%TARGET_NAME%.map
SET INPUT_HEX=%TARGET_PATH%\%TARGET_NAME%.hex
SET OUTPUT_HEX=%TARGET_PATH%\%TARGET_NAME%_CRC.hex
SET TMP_FILE=crc_tmp_file.txt
```

所以直接用 HAL_CRC_Calculate 函数进行计算结果是不对的。可以参考下面的代码来初始化及进行计算：

```
__CRC_CLK_ENABLE();

  CrcHandle.Instance = CRC;
  CrcHandle.Init.DefaultPolynomialUse    = DEFAULT_POLYNOMIAL_ENABLE;
  CrcHandle.Init.DefaultInitValueUse     = DEFAULT_INIT_VALUE_ENABLE;
  CrcHandle.Init.InputDataInversionMode  = CRC_INPUTDATA_INVERSION_NONE;
  CrcHandle.Init.OutputDataInversionMode = CRC_OUTPUTDATA_INVERSION_DISABLED;
  CrcHandle.InputDataFormat              = CRC_INPUTDATA_FORMAT_WORDS;
  HAL_CRC_Init(&CrcHandle);

for(index = 0; index < (uint32_t)ROM_SIZEinWORDS; index++)
{
  CRC->DR = __REV(*((uint32_t *)ROM_START + index));
}
crc_result = CRC->DR;
```

或者，将 crc_gen_keil.bat 文件，里面的 BYTE_SWAP 改为 0， 就可以直接调用 HAL_CRC_Calculate 函数进行计算了。

总结

本文介绍了基于 IAR 及 ARM KEIL 中如何添加 CRC 校验值的过程。在 X-CUBE-CLASSB 软件包中，也都可以找到对应的例程。如果在调试中，遇到 FLASH CRC 校验出错，也不用急。可以从计算 CRC 值的范围设置是否一致和采用的 CRC 算法是否一致这两个方面去找原因。一定要调试看看代码实际执行的情况，比如要测试的地址范围和实际代码执行时计算的地址范围是否一样，防止因为 coding 错误造成检测不通过。

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、更正、增强、修改和改进的权利，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。所有其他产品或服务名称均为其各自所有者的财产。

本文档中的信息取代本文档所有早期版本中提供的信息。