AN0081
应用笔记

AT32F435/437 Security Library Application Note

前言

这篇应用笔记主要在阐述AT32F435/437系列安全库区的应用原理、软件使用方法及范例程序。

支持型号列表：

支持型号	AT32F435 系列
	AT32F437 系列

目录

表目录

图目录

1 概述

目前越来越多的微控器(MCU)应用需要使用到复杂的算法及中间件解决方案(middleware solution)，因此，如何保护软件方案商开发出来的核心算法等知识产权代码(IP-Code)，便成为微控制器应用中一项很重要的课题。

因为这一重要的需求，AT32F435/437系列提供了安全库区(SLIB)的功能，以防止重要的IP-Code被终端用户的程序做修改或读取，进而达到保护的目的。

本文档将详细阐述AT32F435/437系列安全库区的应用原理和软件使用方法。

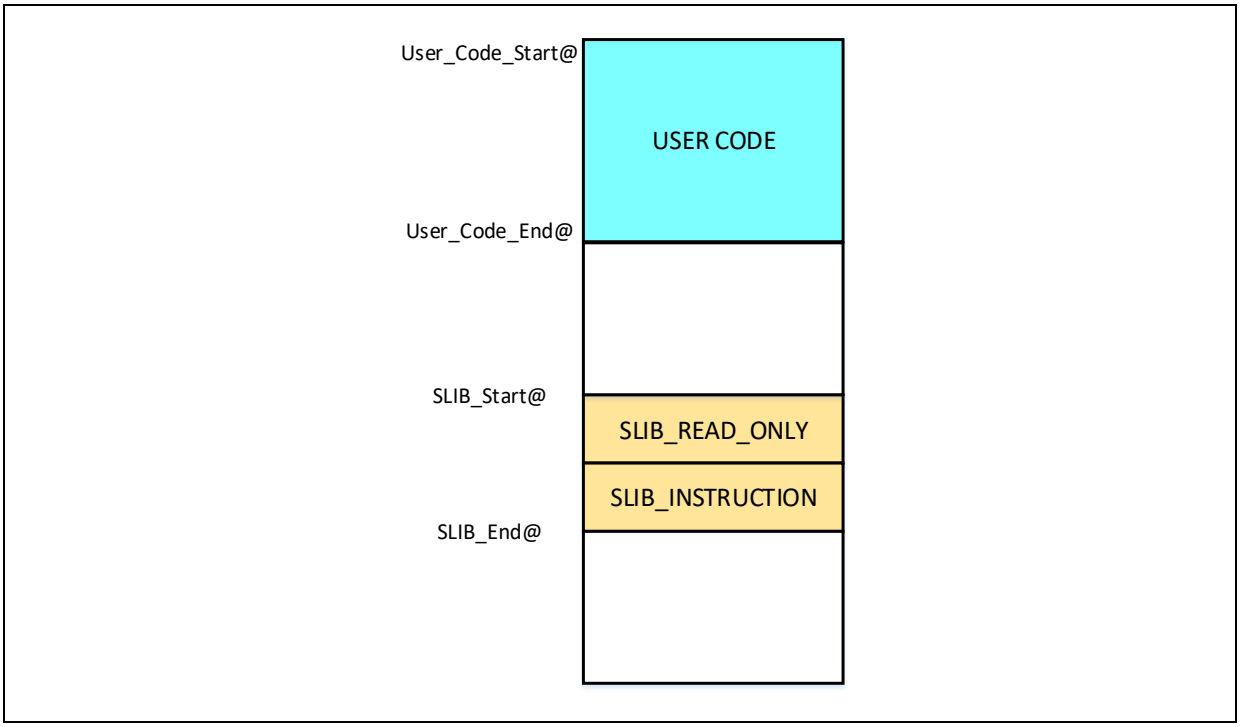
2 应用原理

2.1 安全库区的应用原理

- 设定以密码保护主闪存中指定范围的程序区(即安全库区)，软件方案商可将核心算法存放到此区域，以达到保护的功能，其余空白程序区可以提供给终端商客户进行二次开发。
- 安全库区划分为唯读区(SLIB_READ_ONLY)及指令区(SLIB_INSTRUCTION)，并可选择部分或是整个安全库区存放唯读区或者指令区。
- 唯读安全库区(SLIB_READ_ONLY)的数据能透过I-Code和D-Code总线读取，不能写入。
- 指令安全库区(SLIB_INSTRUCTION)内的程序代码仅能被MCU透过I-Code总线抓取指令(仅能被执行)，不能透过D-Code总线以读取数据的方式读取(包含ISP/ICP/调试模式以及从内部RAM启动的程序)，以读取数据的方式去访问SLIB_INSTRUCTION时，读到的数值全都是0xFF。
- 安全库区的程序代码及数据，除非输入正确的密码，否则无法被擦除。在密码不正确时，对安全库区执行写入或擦除，将会在FLASH_STS寄存器的EPPERR位置"1"提出警告。
- 终端用户执行主闪存的整片擦除时，安全库区的程序代码及数据不会被擦除。
- 当安全库区的保护功能被启动后，可以透过在SLIB_PWD_CLR寄存器写入先前设置的密码来解除保护功能。解除安全库区的保护时，芯片将会执行主闪存的整片擦除(包含安全库区的内容)。因此即使软件方案商设置的密码被泄漏，也不会有程序代码外泄的疑虑。

下图是包含安全库区的主闪存区映射示意图，安全库区的程序代码可以很容易地被终端用户调用并执行，但不能直接被读取，因而达到保护的功能。

图 1. 带有安全库区的主闪存区映射



安全库区的范围大小是以扇区(sector)为单位做设定，每一扇区的大小以实际MCU型号为准。表1是AT32F435/437系列各型号的主闪存大小、每扇区大小及可设置范围。

表 1. AT32F435/437 各型号闪存大小总表

型 号	内部闪存大小 (Byte)	每扇区大小 (Byte)	可设置范围
AT32F435xC AT32F437xC	256K	2K	Sector 0 ~ 127 (0x08000000 ~ 0x0803FFFF)
AT32F435xG AT32F437xG	1024K	2K	Sector 0 ~ 511 (0x08000000 ~ 0x080FFFFF)
AT32F435xM AT32F437xM	4032K	4K	Sector 0 ~ 1007 (0x08000000 ~ 0x083EFFFF)

2.2 如何启动安全库区保护

默认状态下，安全库区设定寄存器始终是不可读且被写保护。要想对安全库区设定寄存器进行写操作，首先要对安全库区设定寄存器解锁，对SLIB_UNLOCK寄存器写入解锁0xA35F6D24值，通过查看SLIB_MISC_STS寄存器的SLIB_ULKF位确认解锁成功，随后便允许对安全库区设定寄存器写入设定值。

启动主闪存安全库区的步骤如下：

- 检查FLASH_STS和FLASH_STS2寄存器的OBF位，以确认没有其他正在进行的闪存操作；
- 对SLIB_UNLOCK寄存器写入0xA35F6D24，以进行安全库区解锁；
- 检查SLIB_MISC_STS寄存器的SLIB_ULKF位，以确认解锁成功；
- 在SLIB_SET_RANGE0寄存器设定要保护的区域，包含SLIB的起始和结束地址；
- 在SLIB_SET_RANGE1寄存器设定要保护的区域，包含SLIB指令区的起始地址和SLIB配置使能位；
- 等待OBF位变为‘0’；
- 在SLIB_SET_PWD寄存器设定安全区域密码；
- 等待OBF位变为‘0’；
- 烧录将存入安全库区的代码；
- 进行系统复位，重装载安全库区设定字；
- 读出SLIB_STS0/STS1/STS2寄存器用于判断安全库区设定结果。

注意事项:

- 只可在主闪存中设置安全库区，实际可设置范围参见表1;
- 安全库区代码必须以扇区为单位进行烧录，且起始地址必须与主闪存地址对齐;
- 中断向量表是数据型态且通常会被放置在闪存的第一扇区(扇区0)内，请勿将闪存的第一扇区设定为安全库区的指令区;

关于安全库区设定寄存器的详细说明，请参阅AT32F435/437系列技术手册。

启动安全库区的程序可参考安全库区应用范例project_l0中位于main.c中的slib_enable()函数。亦可使用雅特力的ICP或ISP刻录工具做设定，后面章节将会有详细的说明。

2.3 如何解除安全库区保护

当安全库区的保护功能被启动后，可以透过在SLIB_PWD_CLR寄存器写入先前设置的密码来解除保护功能。解除安全库区的保护时，芯片将会执行主闪存的整片擦除(包含安全库区的内容)。

解除主闪存安全库区的步骤如下:

- 检查FLASH_STS寄存器的OBF位，以确认没有其他正在进行的编程操作;
- 在SLIB_PWD_CLR寄存器写入先前设置的安全区域密码;
- 进行系统复位，重装载安全库区设定字;
- 读出SLIB_STS0寄存器用于判断安全库区设定结果。

2.4 编排及执行安全库区的程序

如前面章节所提到，在指令安全库区(SLIB_INSTRUCTION)内的的程序代码可以被MCU经由I-Code总线抓取，但不能经由D-Code总线以读取数据的方式去读出，这样的保护是全面性的，也就是说在指令安全库区之内的程序代码，也不能读取同样被放置在指令安全库区之内的数据，例如C程序代码常被编译成的文字池(literal pool)、分支表(branch table)或常数(constant)等之类当指令被执行时会经由D-Code总线去读取的数据。

这代表指令安全库区之内只能放置指令，不能放置任何数据。因此用户在编排要放置在指令安全库区之内的程序代码时，必须配置编译程序(compiler)的设定去产生只执行(execute-only)的代码以避免上述那些型态的数据产生。

图2及图3是一般常见的文字池跟分支表的例子:

switch()是C程序中常用的跳转指令，此例子中的sclk_source变量是去读取CRM_CFG寄存器，图2可看到编译出来的汇编代码(assembly code) “LDR R7, [PC, #288]”，会用程序计数器(program counter, PC)间接寻址的方式去取得CRM_CFG寄存器的地址，而CRM_CFG的地址会被以常数的方式存放在邻近的指令区(也在指令安全库区之内)，因此执行switch()指令时就会发生数据的读取。如果指令安全库区内有这类的程序代码，在执行的时候就会产生错误。

第三章的范例程序将会说明如何设定编译程序的配置来避免这样的问题。

图 2. 文字池例子(1)

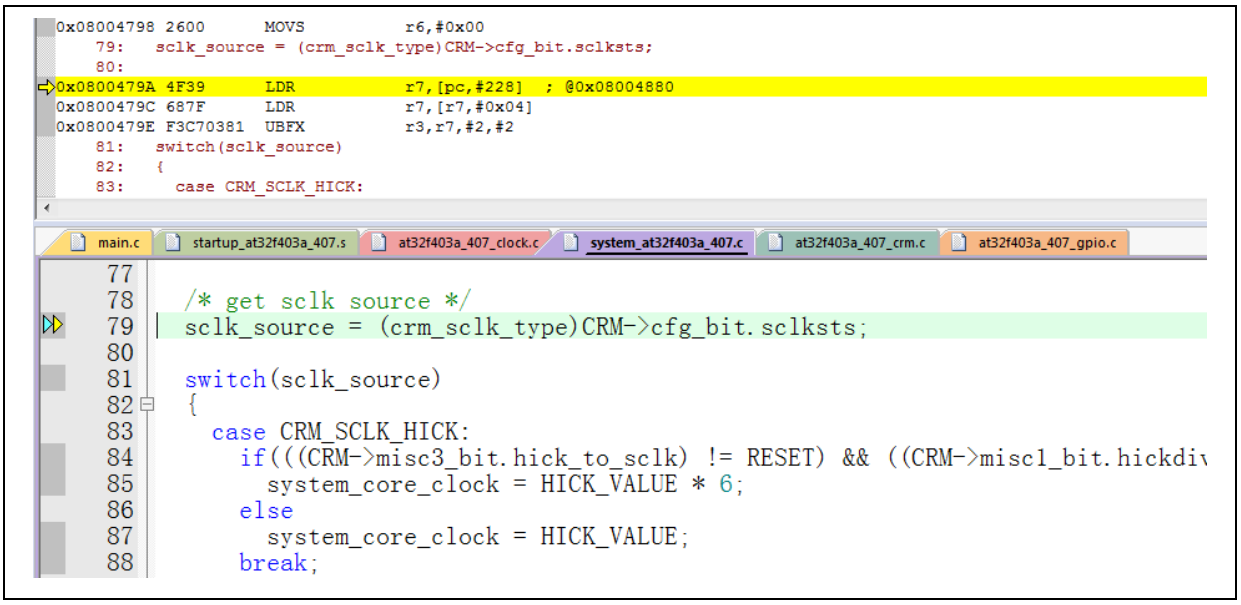
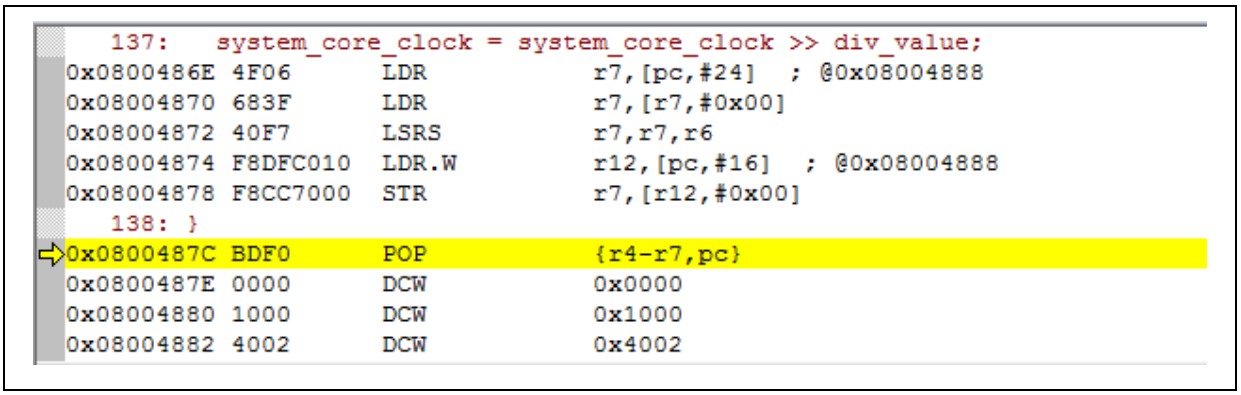


图 3. 文字池例子(2)



2.4.1 不可将中断向量表设置为安全库区的指令区

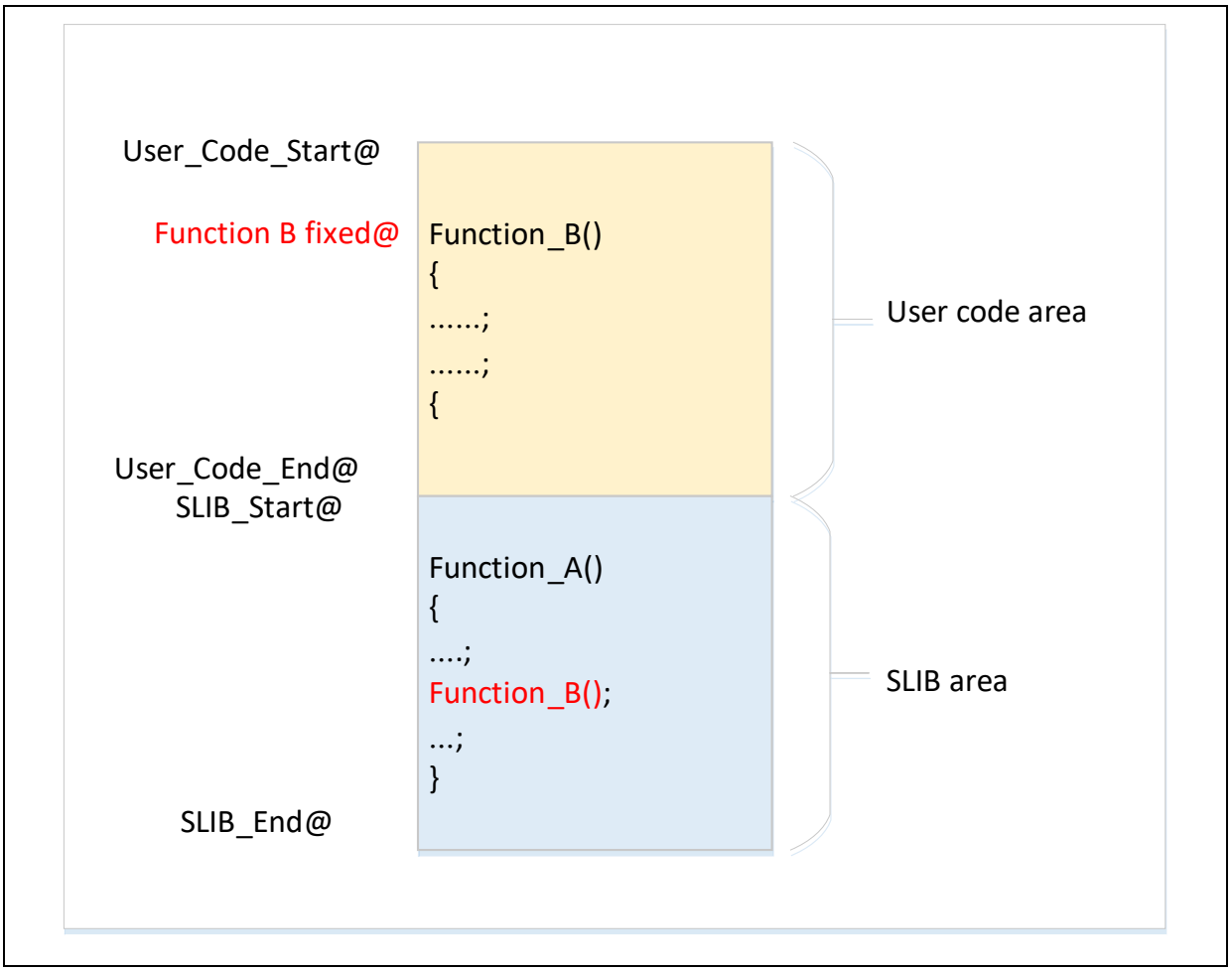
中断向量表包含每个中断处理程序的入口点地址，由MCU通过D-Code总线读取。通常，中断向量表位于主闪存第一扇区(sector 0)的起始地址0x08000000，因此在设置指令安全库区时，必须遵守以下的规则：

- 不可将主闪存的第一扇区设置为安全库区的指令区。

2.4.2 安全库区代码与用户区代码的关联性

受安全库区保护的程序代码(IP-Code)可以从位于用户代码区(安全库区之外的区域)的函数库中调用函数。在这种情形下，IP-Code将会包含这些函数的地址，允许PC（程序计数器）在执行IP-Code时跳转到这些函数。一旦安全库区被启动，这些函数的地址就不能被改变，此时，这些位于用户代码区的函数的地址就必须固定下来，否则PC将跳转到错误的地址而无法正常工作。因此在设置安全库区的时候，应该将所有与IP-Code相关联的函数都一起编排到安全库区之内以避免此情况发生。下图显示出一个被保护的函数Function_A()调用到用户区内的函数Function_B()的例子。

图 4. 安全库区的函数调用用户区函数的例子



此外，另一个最常见的情形就是使用到C语言的标准函式库，例如memset()及memcpy()这类函数。如果IP-Code跟用户区代码都有调用到这类函数，就会有上述问题的困扰。列举两种常用的解决方法：

1) 将其编译到安全库区范围内，具体如何实现可以查看keil或IAR的相关文档。
2) 避免在IP-Code内使用C的标准函式库，若非要使用，就必须将用到的函数改写为其他名称，以下是一个范例，在IP-Code 中写一个my_memset()函数取代原先的memset()。

图 5. 自定义函数范例

```
void* my_memset(void *s, int c, size_t n);

void arm_fir_init_f32(
  arm_fir_instance_f32 * S,
  uint16_t numTaps,
  float32_t * pCoeffs,
  float32_t * pState,
  uint32_t blockSize)
{
  /* Assign filter taps */
  S->numTaps = numTaps;

  /* Assign coefficient pointer */
  S->pCoeffs = pCoeffs;

  /* Clear state buffer and the size of state buffer is (blockSize + numTaps - 1) */
  my_memset(pState, 0, (numTaps + (blockSize - 1u)) * sizeof(float32_t));

  /* Assign state pointer */
  S->pState = pState;

}



void* my_memset(void *s, int c, size_t n)
{
  while (n>0)
      *( (char*)s + n-- -1 ) = (char)c;

  return (s);
}
```

3 安全库区范例程序

本章节将介绍安全库区的使用范例，并详述完成此范例程序所需的每一个步骤。

因为AT32F435的SLIB功能与AT32F437完全相同，此处就以AT32F437为范例做说明。

3.1 范例需求

3.1.1 硬件需求

- 带有AT32F437ZMT7芯片的AT-START-F437实验板
- AT-Link仿真器，用来调试范例程序

3.1.2 软件需求

- Keil® µvision IDE (本范例使用µvision V5.18.0.0) 或 IAR Embedded workbench IDE(本范例使用IAR V8.22.2)
- 雅特力ICP或ISP刻录工具，主要是用来启动或解除安全库区的设置

3.2 范例概述

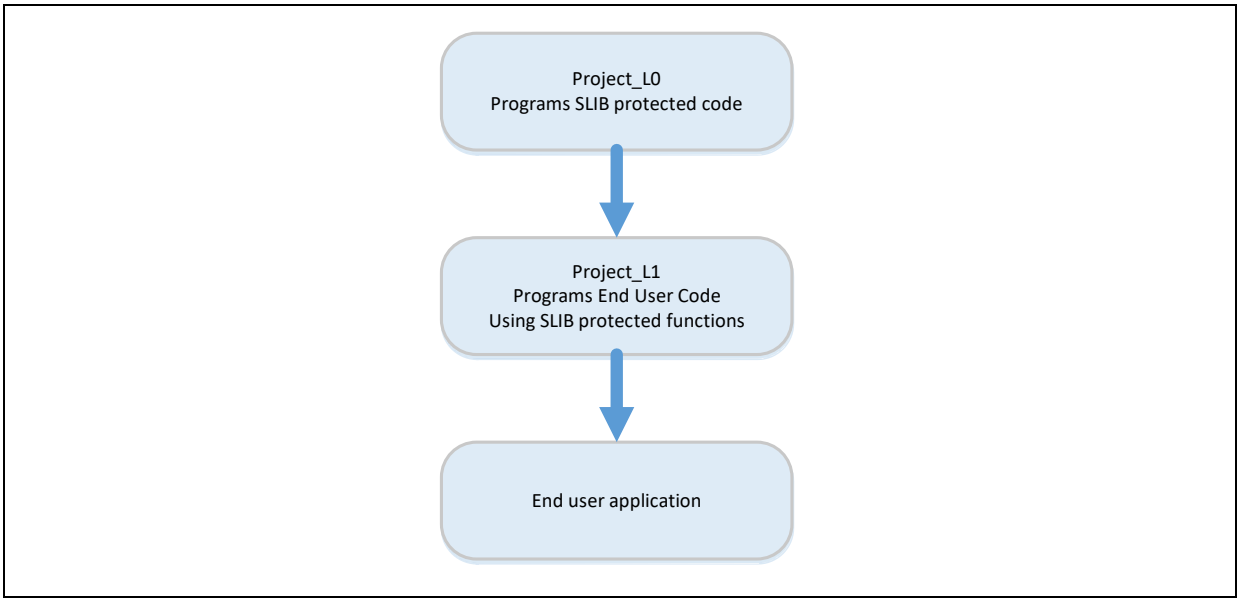
本应用指南提供了两个范例项目，展示了软件开发商开发智权代码(IP-Code)给终端用户应用的场景。其中

- Project_L0为方案商开发算法并编排到安全库区的示例
- Project_L1为终端用户应用此算法的示例

Project_L0完成的算法将预先下载刻录到AT32F437芯片并设置安全库区保护，同时提供下列各项设定讯息给终端客户应用程序使用:

- 主闪存区块的映像，说明安全库区所占用的区域以及用户可开发程序的区域
- 包含算法函数定义的头文件，让终端用户可以用来调用相关的函数
- 符号定义文件(symbol definition file)，此符号文件内含IP-Code的各个函数的实际地址，让终端用户程序可以正确的调用，下图为此范例的示意图。

图 6. 范例流程示意图



软件方案商可以参考Project_L0范例开发算法代码，并参考Project_L1提供终端用户使用，下图为应

用示意图。

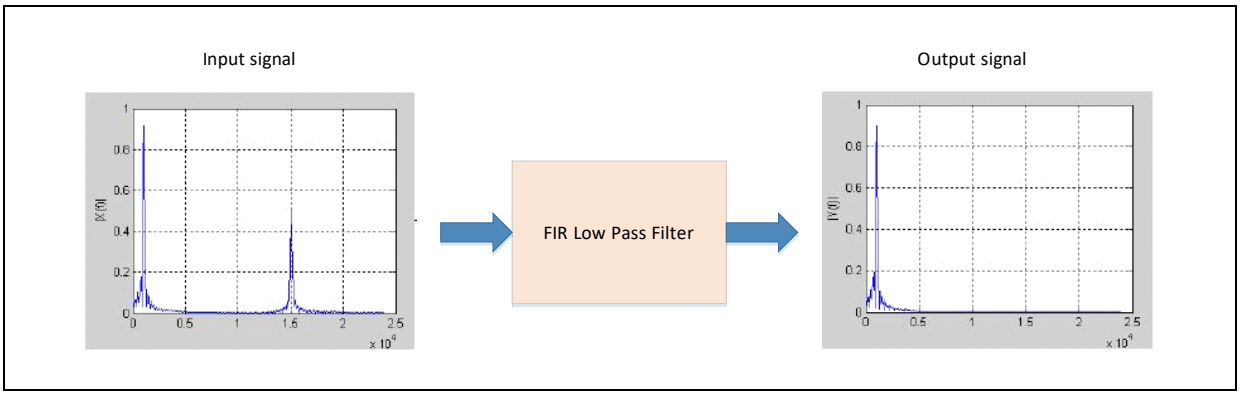
图 7. 应用示意图

3.3 安全库区保护的代码：FIR 低通滤波器

本范例使用CMSIS-DSP库提供的FIR低通滤波器(FIR lowpass filter)算法作为被安全库保护的IP-Code，关于FIR低通滤波器算法可详阅CMSIS-DSP的相关文件，这里仅着重在说明如何设置安全库以保护此算法及如何被终端用户的程序代码调用。

范例中的低通滤波器的输入信号是一个混和了频率各为1KHz及15KHz的两个正弦波的讯号，而低通滤波器的截止频率约为6KHz。经过低通滤波后，将15KHz的讯号滤除而仅剩下1KHz的正弦波输出。下图为FIR低通滤波功能的示意图。

图 8. FIR 低通滤波器



使用到的CMSIS DSP库的函数及文件包括：

- arm_fir_init_f32()

此函数的功能是做滤波器函数的初始化设定，包含在arm_fir_init_f32.c文件里

- arm_fir_f32()

此函数为滤波器算法的主要部分，包含在arm_fir_f32.c文件里

- FIR_lowpass_filter()

此函数为使用上述两个基本函数写成的FIR低通滤波器全局函数，供终端用户调用，包含在fir_filter.c文件里

- fir_coefficient.c

此C文件内含FIR滤波器函数所使用的系数(只读的常数)，在范例中会将这些系数放置到唯读安全库区

在此范例中，MCU内嵌的FPU及DSP指令会被用来做信号处理以及浮点运算，以达到准确的运算及正确的输出信号。

3.4 Project_L0：方案商范例

在此阶段的范例程序，将完成下列几个项目:

- 将算法的相关函数编译成只可执行(execute-only)的代码
- 将算法的程序代码编排放置到主闪存区的扇区2(sector 2)
- 将滤波器函数的系数编排放置到主闪存区的扇区1(sector 1)
- 在主程序中执行FIR_lowpass_filter()以验证其正确性
- 验证成功后，将扇区2(sector 2)设置为指令安全库区，并将扇区1(sector 1)设置为唯读安全库区，此部分可在范例的主程序中以调用slib_enable()函数来完成，或使用Artery ICP Programmer来完成(建议使用ICP工具完成设置)
- 产出终端用户程序调用低通滤波函数时需用到的头文件及符号定义文件

3.4.1 产生只执行(Execute-only)代码

每一种工具链(toolchain)都有自己的设定选项，可以防止编译程序生成文字池(literal pools)和分支表(branch table)这些在指令执行时会发生读取数据的指令格式，例如"LDR Rn, [PC, #offset]"这类指令。关于文字池及分支表的例子可参照章节2.4的说明。

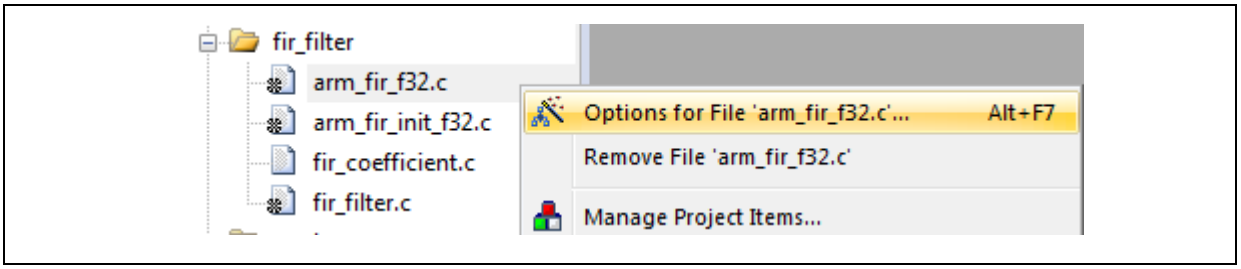
以Keil® µvision为例，Keil® µvision有Execute-only Code的选项来做设定，设定的方式如下:

Keil® µvision：使用Execute-only Code选项

设置的方式是:

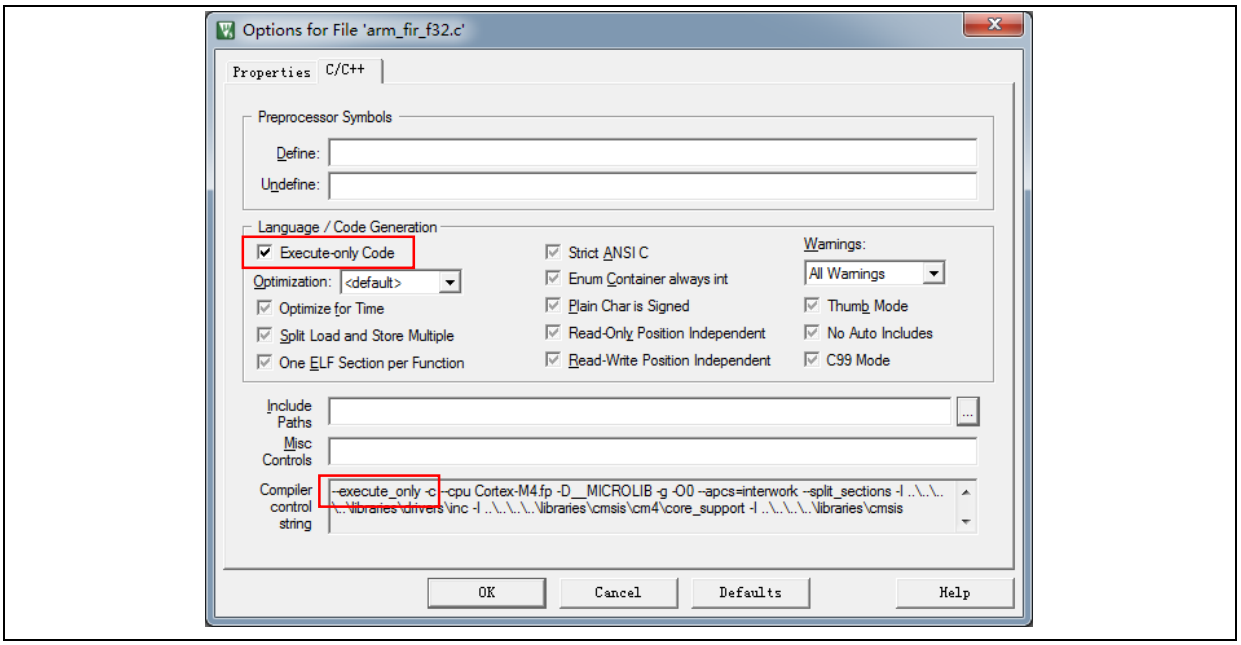
- 选择C文件群组或个别的C文件，范例中是把要保护的相关C文件都放在fir_filter群组
- 按鼠标右键然后选择对应文件，例如本例程的Option for File 'arm_fir_f32.c'，如下图

图 9. Keil 进入 Option 界面



- 勾选 C/C++ 窗口里的Execute-only Code选项，然后--execute_only命令就会被加到编译过程控制字符串里，如下图

图 10. Keil 选择 Execute-only Code



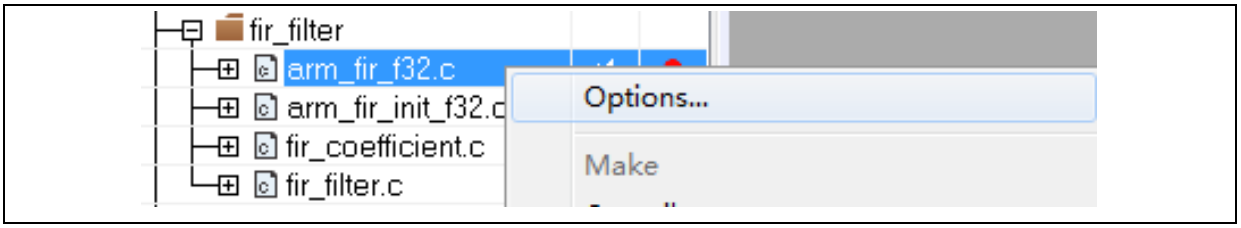
- 本例程中有三个文件位于SLIB_INSTRUCTION区，分别是arm_fir_f32.c、arm_fir_init_f32.c和fir_filter.c，这三个文件都需要配置产生为只执行代码。

IAR ：使用No data read in code memory 选项

设置的方式是:

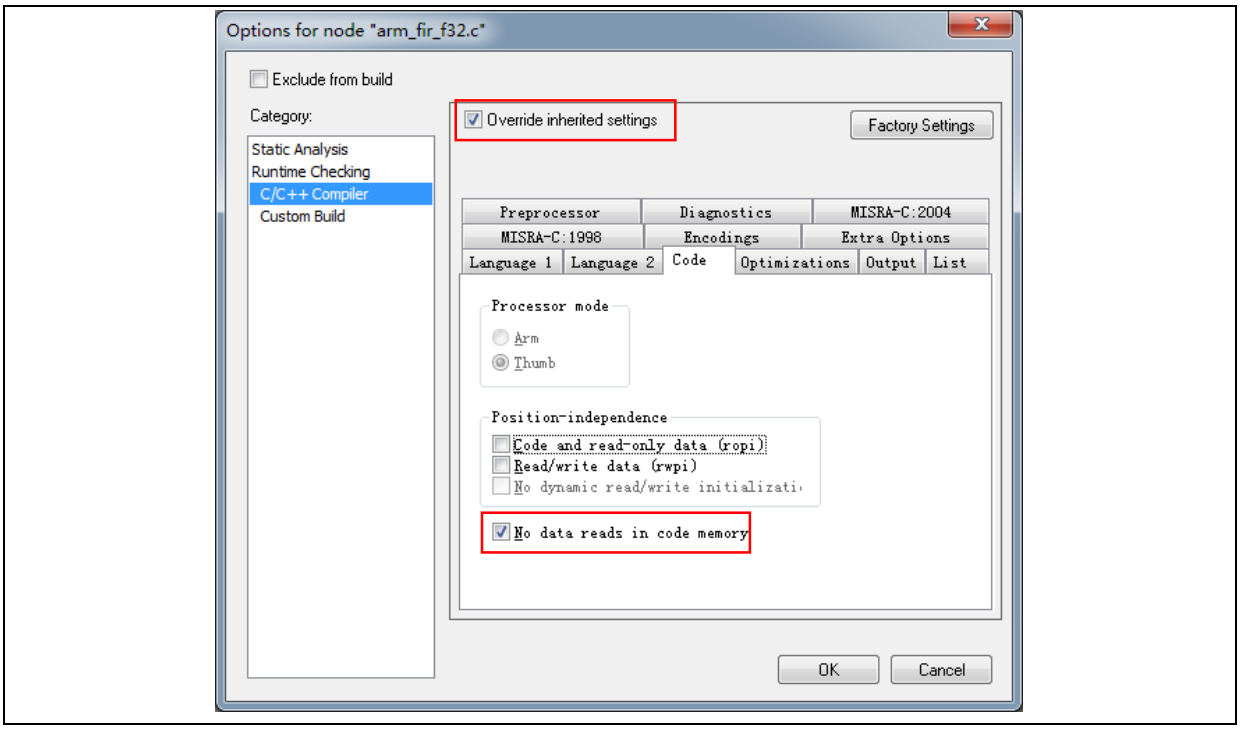
- 选择fir_filter群组里对应的文件，按鼠标右键选择Option

图 11. IAR 进入 Option 界面



- 如下图，在"C/C++"窗口内勾选Override inherited settings以及No data read in code memory

图 12. IAR 设置 C/C++窗口选项

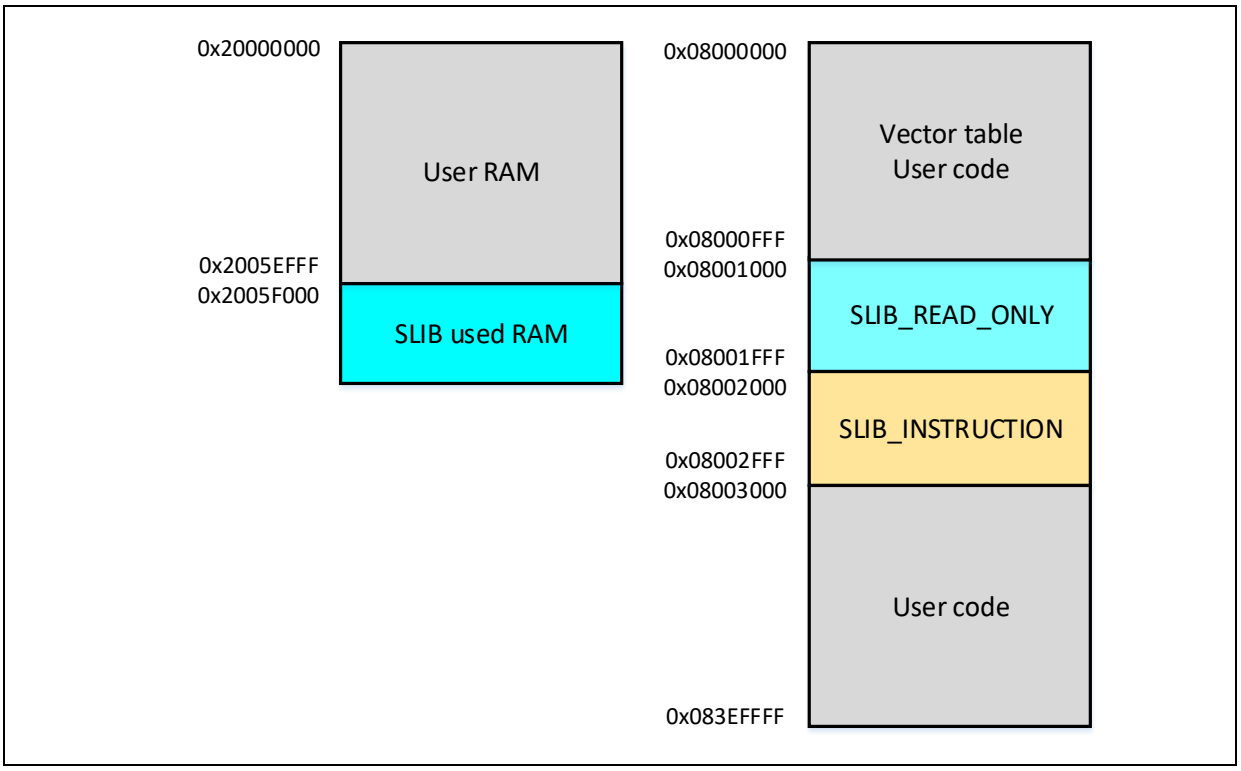


- 本例程中有三个文件位于SLIB_INSTRUCTION区，分别是arm_fir_f32.c、arm_fir_init_f32.c和fir_filter.c，这三个文件都需要配置产生为只执行代码。

3.4.2 编排安全库区的地址

如前面章节提到的，因为主闪存的第一扇区(sector 0)会被用来存放中断向量表，所以本范例选择从sector 1开始设定为安全库区，其中sector 2是指令安全库区，而sector 1是唯读安全库区。下图为主闪存的映射及RAM的使用分区。RAM的分区主要是为了避免SLIB保护区的代码与终端用户的代码用到相同的RAM而产生的冲突问题。

图 13. 范例程序的主闪存映像及 RAM 分区

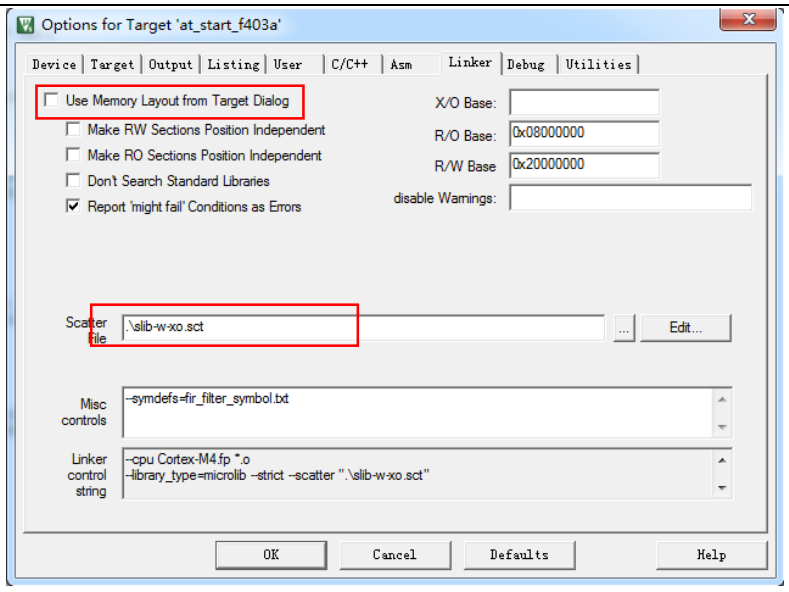


Keil® µvision的scatter file

步骤如下:

- 到 Project → Optios for Target→Linker 窗口，取消Use memory layout from Target Dialog选项，然后按Edit按键来开启slib-w-xo.sct文件做修改，如下图

图 14. Keil 设置 Linker 窗口选项



- 打开scatter file之后，将需要放到指令安全库区(SLIB_INSTRUCTION)的代码的目标文件(object file)放到名为LR_SLIB_INSTRUCTION的专用加载区，并将标示修改为execute-only (+XO)，加载区的起始位置为sector 2，大小占用一个sector，同时也要将SLIB_READ_ONLY占用的区域保留起来放到名为LR_SLIB_READ_ONLY的专用加载区，避免编译程序将其他非IP-Code的函

数编排到SLIB区内，RW_IRAM2是将0x2005F000到0x2005FFFF的区块，指定给安全库区算法的函数使用，目的是为了避免终端用户的项目也用到同样的RAM区块，而造成程序执行时发生错误，如下所示

图 15. Keil scatter 修改

```
LR_IROM1 0x08000000 0x001000  {    ; load region size_region
  ER_IROM1 0x08000000 0x001000  {  ; load address = execution address
   *.o (RESET, +First)
   *(InRoot$$Sections)
   .ANY (+RO)
  }

  RW_IRAM1 0x20000000 0x0005F000  {  ; user RW data
   .ANY (+RW +ZI)
  }

  RW_IRAM2 0x2005F000 0x00001000  {  ; RAM used for slib code
   fir_filter.o (+RW +ZI)
  }
}

LR_SLIB_READ_ONLY 0x08001000 0x00001000  {    ; sLib read-only area
  ER_SLIB_READ_ONLY 0x08001000 0x00001000  {
   fir_coefficient.o (+RO)
  }
}

LR_SLIB_INSTRUCTION 0x08002000 0x00001000  {    ; slib instruction area
  ER_SLIB_INSTRUCTION 0x08002000 0x00001000  {  ; load address = execution address
    arm_fir_init_f32.o (+XO)
    arm_fir_f32.o (+XO)
    fir_filter.o (+XO)
  }
}

LR_IROM2 0x08003000 0x003ED000  {    ; user code area
  ER_IROM2 0x08003000 0x003ED000  {  ; load address = execution address
   .ANY (+RO)
  }
}
```

- IP-Code 用到的RAM，除了上述的修改scatter file方式之外，也可以使用Keil的__attribute__((at(address)))描述元将变数放置到固定的地址0x2005F000，如下图

图 16. KEIL 代码中 SLIB 使用的 RAM 地址修改

```
#if   defined ( __ICCARM__ )
static float32_t firStateF32[BLOCK_SIZE + NUM_TAPS - 1]  @ 0x2005F000 ;
#elif defined ( __CC_ARM )
static float32_t firStateF32[BLOCK_SIZE + NUM_TAPS - 1] __attribute__((at(0x2005F000)));
#endif
static float32_t firStateF32[BLOCK_SIZE + NUM_TAPS - 1] ;
```

- 唯读安全库区的起始位置为sector 1 (0x08001000)，要将FIR低通滤波器函数使用到的常数编排到此地址，除了上述的修改scatter file方式之外，也可以使用Keil的__attribute__((at(address)))描述元将常数放置到固定的地址，如下图

图 17. KEIL 代码中 SLIB 使用的常量地址修改

```
#if   defined ( __ICCARM__ )
const float32_t firCoeffs32[NUM_TAPS] @ 0x08001000 ={
#elif defined ( __CC_ARM )
const float32_t firCoeffs32[NUM_TAPS] __attribute__((at(0x08001000))) = {
#endif
  -0.0018225230f, -0.0015879294f, +0.0000000000f, +0.0036977508f, +0.0080754303f, +0.0085302217f, -0.00(
  -0.0341458607f, -0.0333591565f, +0.0000000000f, +0.0676308395f, +0.1522061835f, +0.2229246956f, +0.25(
  +0.1522061835f, +0.0676308395f, +0.0000000000f, -0.0333591565f, -0.0341458607f, -0.0173976984f, -0.00(
  +0.0080754303f, +0.0036977508f, +0.0000000000f, -0.0015879294f, -0.0018225230f
};
```

IAR的ICF file

步骤如下:

- 开启\project_I0\IAR_V8.2\目录下的icf文件，添加三个新的加载区，如下所示，其中SLIB_RAM区块将0x2005F000到0x2005FFFF的RAM保留给算法的函数使用

图 18. icf 文件中 SLIB 地址定义

```
/* SLIB read-only area */
define symbol __ICFEDIT_region_SLIB_READ_ONLY_start__ = 0x08001000;
define symbol __ICFEDIT_region_SLIB_READ_ONLY_end__   = 0x08001FFF;

/* SLIB instruction area */
define symbol __ICFEDIT_region_SLIB_INST_start__ = 0x08002000;
define symbol __ICFEDIT_region_SLIB_INST_end__   = 0x08002FFF;

define symbol __ICFEDIT_region_RAM_start__ = 0x20000000;
define symbol __ICFEDIT_region_RAM_end__   = 0x2005EFFF;

/* SLIB RAM region */
define symbol __ICFEDIT_region_SLIB_RAM_start__ = 0x2005F000;
define symbol __ICFEDIT_region_SLIB_RAM_end__   = 0x2005FFFF;
```

- 在ICF文件中，也要将SLIB占用的区域保留起来，避免编译程序将其他非IP-Code的函数编排到SLIB区内，同时将IP-Code使用的RAM区域保留起来

图 19. icf 文件中地址分配

```
/* Reserved 0x08001000 ~ 0x08002FFF as SLIB area */
define region ROM_region   = mem:[from __ICFEDIT_region_ROM_start__   to __ICFEDIT_region_ROM_end__]
                           -mem:[from __ICFEDIT_region_SLIB_READ_ONLY_start__  to __ICFEDIT_region_SLIB_READ_ONLY_end__]
                           -mem:[from __ICFEDIT_region_SLIB_INST_start__  to __ICFEDIT_region_SLIB_INST_end__] ;

define region SLIB_READ_ONLY_region   = mem:[from __ICFEDIT_region_SLIB_READ_ONLY_start__   to __ICFEDIT_region_SLIB_READ_ONLY_end__];

define region SLIB_INST_region   = mem:[from __ICFEDIT_region_SLIB_INST_start__   to __ICFEDIT_region_SLIB_INST_end__];

/* Reserved 0x2005F000 ~ 0x2005FFFF as RAM used for SLIB code */
define region RAM_region   = mem:[from __ICFEDIT_region_RAM_start__   to __ICFEDIT_region_RAM_end__]
                           - mem:[from __ICFEDIT_region_SLIB_RAM_start__   to __ICFEDIT_region_SLIB_RAM_end__];

define region SLIB_RAM_region = mem:[from __ICFEDIT_region_SLIB_RAM_start__   to __ICFEDIT_region_SLIB_RAM_end__];
```

- IP-Code 用到的RAM，代码中可以使用IAR的@描述元将变数放置到固定的地址0x2005F000，或者修改icf文件，如下图

图 20. icf 文件中 SLIB 使用的 RAM 修改

```
/* Place IP Code in instruction area which will be SLIB protected */
place in SLIB_INST_region    { ro object arm_fir_f32.o,
                               ro object arm_fir_init_f32.o,
                               ro object fir_filter.o};

/* Place SLIB DATA(or CODE) in read-only area  */
place in SLIB_READ_ONLY_region    { ro object fir_coefficient.o };


place in RAM_region    { readwrite,
                         block CSTACK, block HEAP };

/* Place slib used sram */
place in SLIB_RAM_region    { readwrite object fir_filter.o };
```

- 唯读安全库区的起始位置为sector 1 (0x08001000)，要将FIR低通滤波器函数使用到的常数编排到此地址，除了上述的ICF文件方式之外，也可以使用IAR的@描述元将常数放置到固定的地址，如下图

图 21. IAR 代码中 SLIB 使用的常量地址修改

```
#if   defined ( __ICCARM__ )
static float32_t firStateF32[BLOCK_SIZE + NUM_TAPS - 1] @ 0x2005F000 ;
#elif defined ( __CC_ARM )
static float32_t firStateF32[BLOCK_SIZE + NUM_TAPS - 1] __attribute__((at(0x2005F000)));
#endif
static float32_t firStateF32[BLOCK_SIZE + NUM_TAPS - 1] ;
```

3.4.3 启用安全库区保护

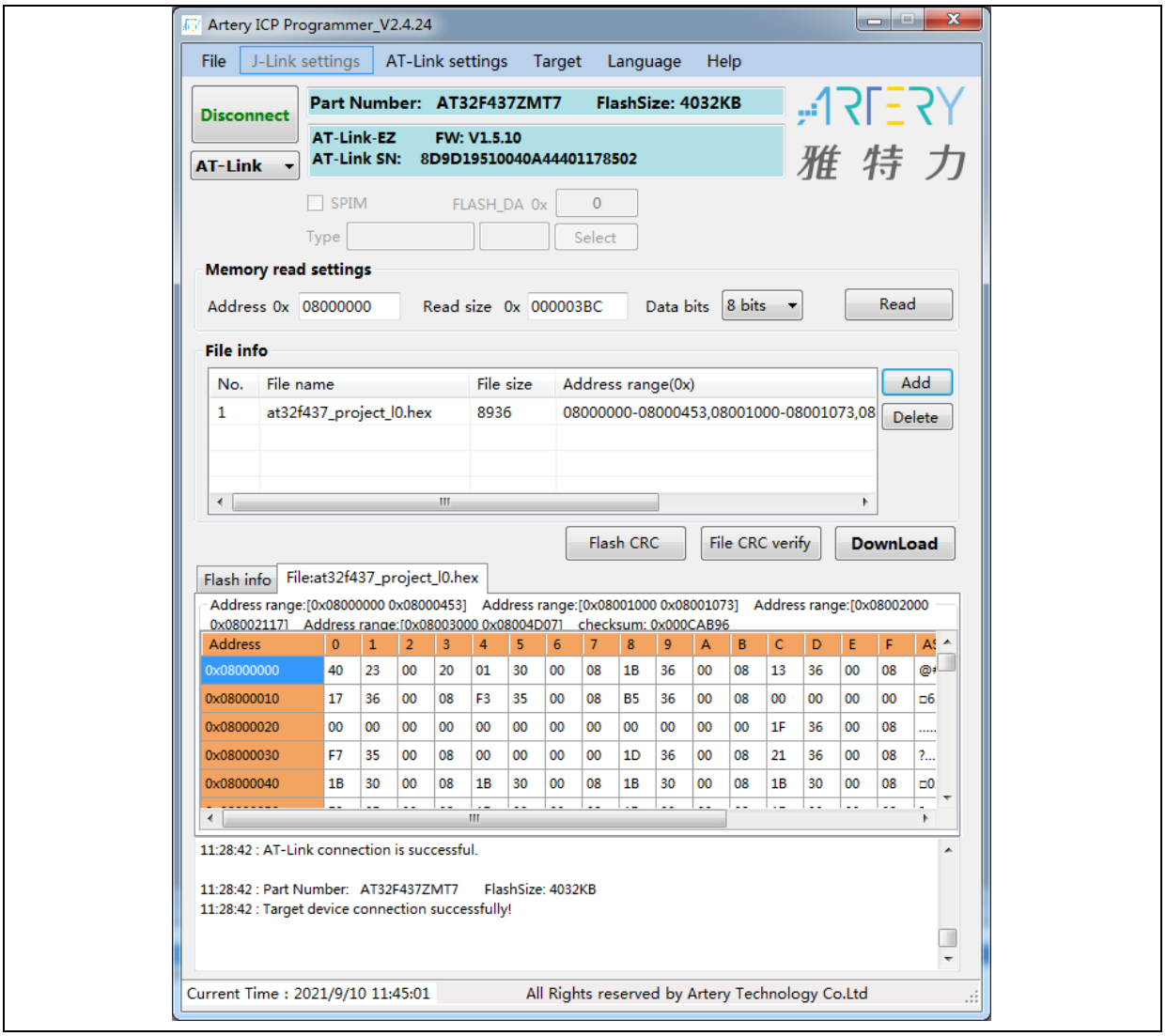
要启用安全库区的保护功能，有以下两种方式:

(1) 使用ICP刻录工具 Artery ICP Programmer(建议用此方式)

要使用ICP Programmer，请参照以下步骤:

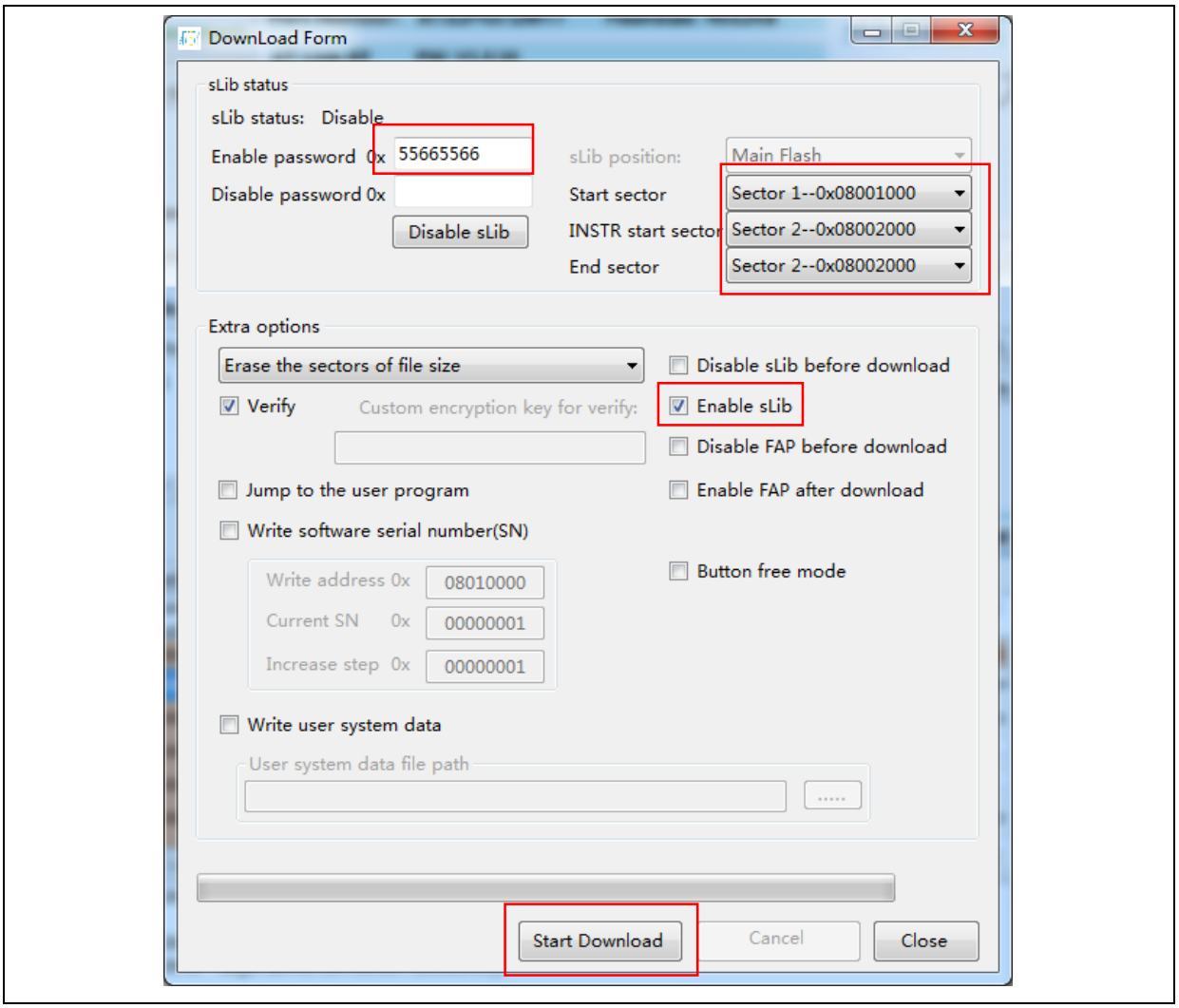
- 连接AT-Link到AT-START-F437板子上并上电
- 开启ICP Programmer，选择用AT-Link连接，然后添加Project_L0范例编译后产出的HEX或BIN文件，如下图

图 22. 配置 ICP Programmer



- 按下载按键，会出现下载选项的页面，此页面会显示SLIB的状态及相关的参数，设定sector 1为开始扇区、sector 2为指令开始扇区和SLIB结束扇区，设定启用密码0x55665566 (可自定义)并勾选启用SLIB，然后按开始下载，即可完成程序的烧录并启用SLIB，如下图

图 23. 设置下载选项参数



关于ICP Programmer的详细说明，请参阅ICP Programmer用户手册。

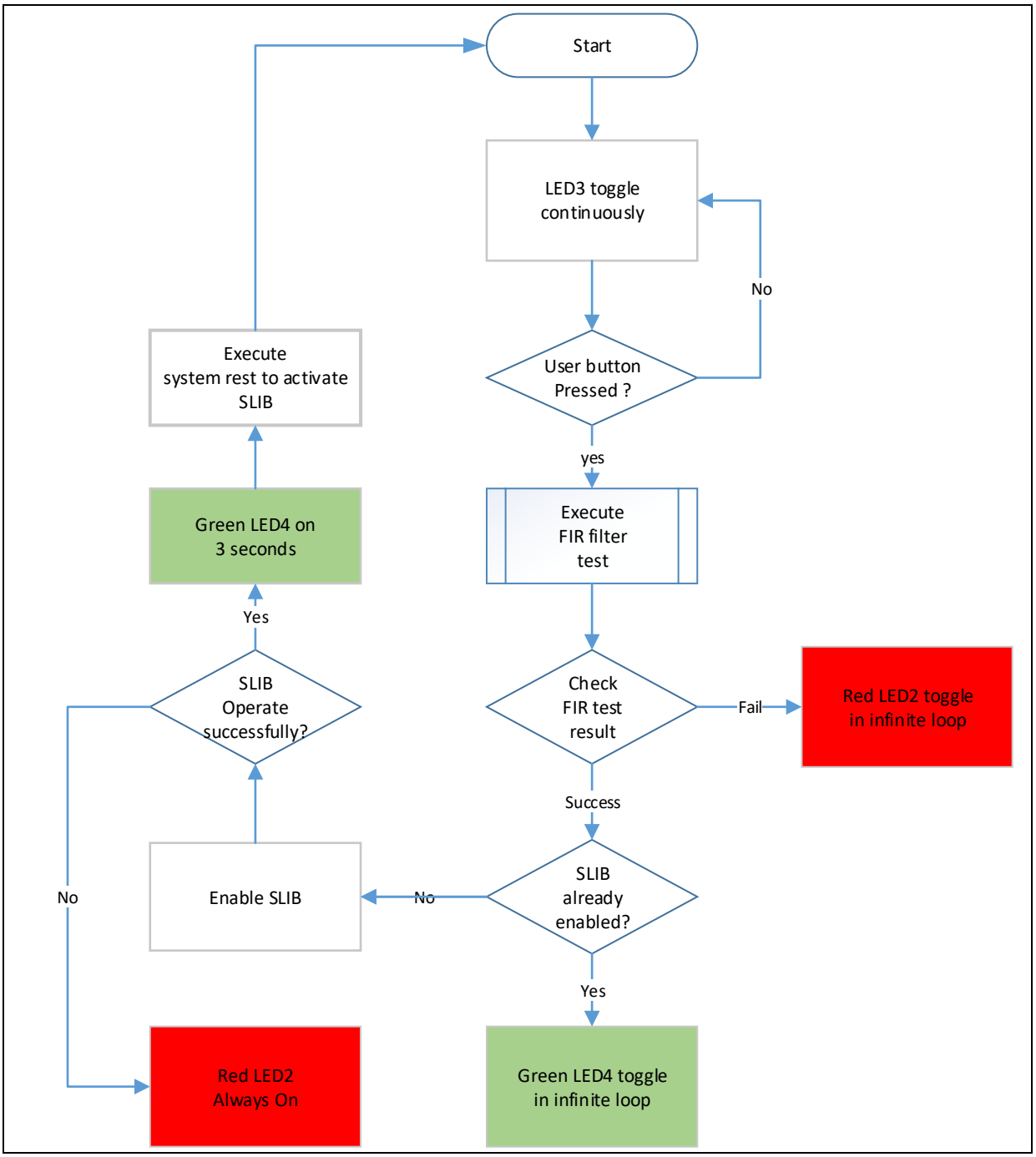
(2) 使用范例程序main.c之中的slib_enable()函数

在低通滤波函数测试正确后执行过一次此函数，就可以启用安全库区的保护功能。要执行此函数，只要在main.c中使能 #define USE_SLIB_FUNCTION 即可。

3.4.4 Project_L0 执行流程

在此范例中，FIR低通滤波器会针对混和1KHz及15KHz正弦波的输入信号testInput_f32_1kHz_15kHz 做计算，计算后输出的1KHz正弦波数据存放到testOutput，然后会跟预先用MATLAB软件计算好且存放在refOutput中的数据做比对，如果误差值小于预期值(讯噪比SNR大于预设的门坎)，板子上绿色的LED灯会一值闪烁，反之则是红色的LED灯一值闪烁，下图是Project_L0的整个流程

图 24. Project_L0 执行流程



要执行此范例程序，请按照下列步骤:

(1) 使用Keil® µvision开启\utilities\AT32F435_437_slib_demo\project_l0\mdk_v5\目录下的Project_L0项目，并重新编译。

(2) 在下载代码之前，先检查AT-START-F437板子上的芯片是否已经有SLIB或读写保护(FAP/EPP)，如果有，就请先用ICP刻录工具将这些保护都解除，然后再下载代码。

(3) 下载成功后并开始值执行后，会看到板子上的LED3灯持续快速闪烁。

(4) 按下板子上的USER按键，就会执行低通滤波器的运算。

(5) 比对运算结果，若结果正确，绿色LED4灯会持续闪烁。反之，则是红色LED2灯持续闪烁。

(6) 在比对结果正确的条件下，如果main.c中的USE_SLIB_FUNCTION有被定义且芯片未启用过

SLIB的话，就会执行slib_enable()函数去设置SLIB，若设置失败，红色LED2灯会一直亮着。若设置成功，绿色LED4灯会点亮约3秒钟然后执行系统重置(system reset)来启动SLIB。然后程序又回到步骤(3)。

3.4.5 产生头文件及符号定义文件

头文件(header file)跟符号定义文件(symbol definition file)是终端客户应用范例Project_L1在调用FIR低通滤波函数时需要用到。在范例中，就是main.c中包含的fir_filter.h 文件。

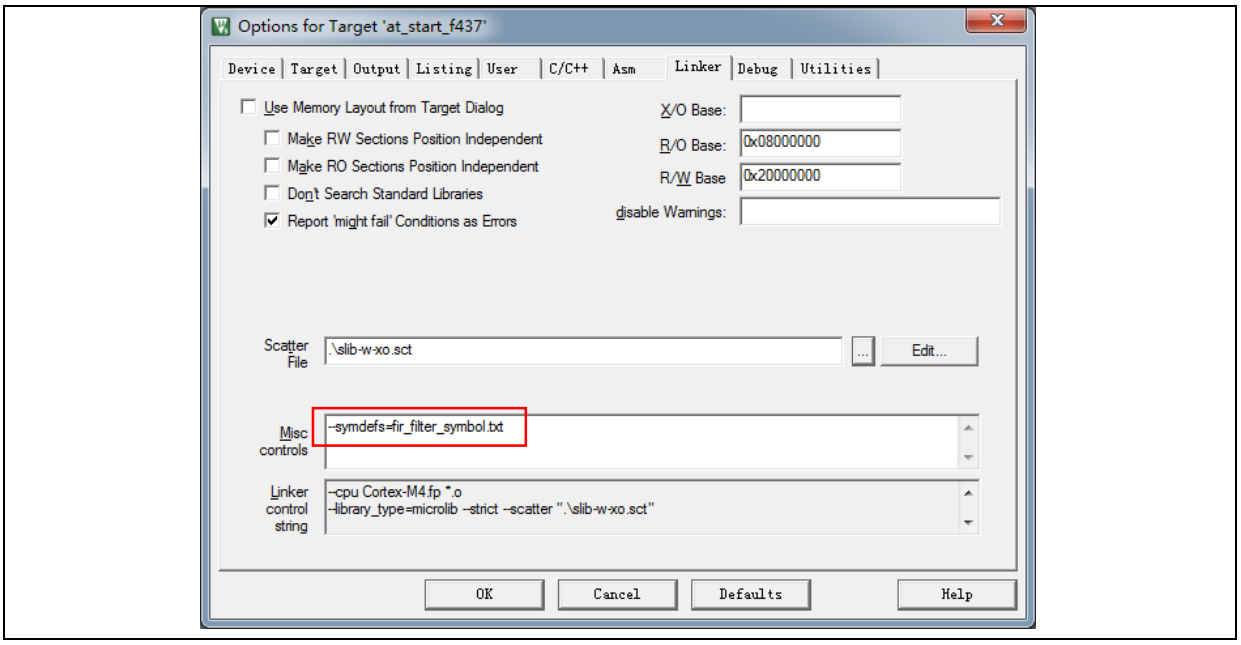
符号定义文件的产出方法跟使用的工具链(toolchain)相关。

使用Keil® µvision 产生符号定义文件

方法如下:

- 进入 Ottions for Target → Linker 设定画面
- 在 Misc controls 这一栏，添加--symdefs=fir_filter_symbol.txt 命令，如下图

图 25. 设置 Keil Misc controls 选项



- 重新编译整个项目后，在project_l0\mdk_v5\Objects 目录下就会产生一个名为fir_filter_symbol.txt的符号定义文件
- 这个符号定义文件包含了整个项目全部的符号定义，所以需要修改，只保留终端用户会调用的低通滤波函数的定义，删减后的fir_filter_symbol.txt显示如下

图 26. 修改后的 fir_filter_symbol.txt 内容

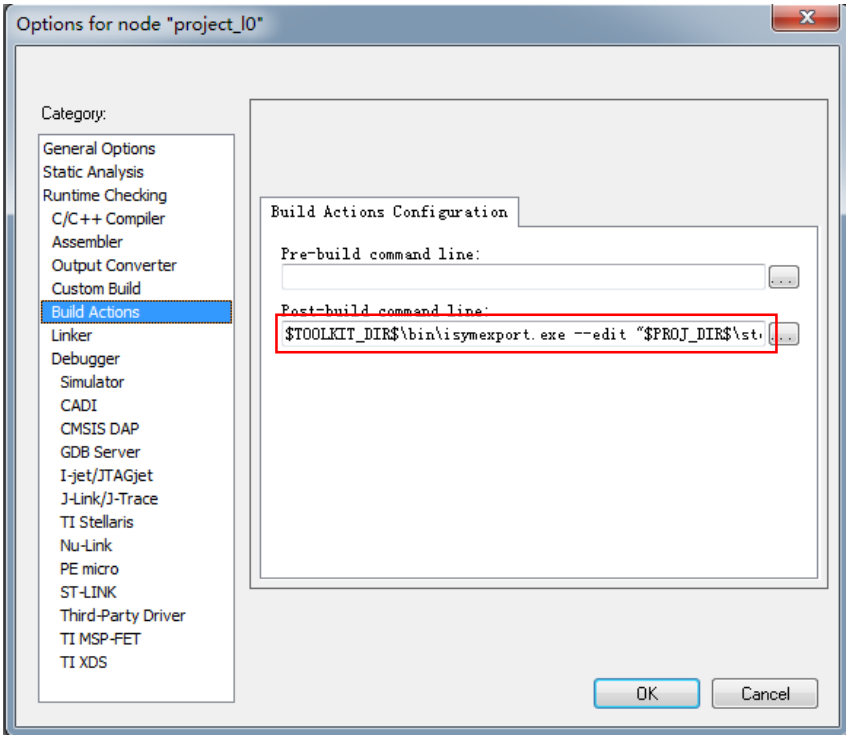
```
0x08002001 T FIR_lowpass_filter
```

使用IAR产生符号定义文件

方法如下:

- 选择 Project→Option→Build Actions

图 27. 设置 IAR Build Actions 选项



- 然后在 Post-build 命令行中输入以下命令

  $TOOLKIT_DIR$\bin\isymexport.exe --edit "$PROJ_DIR$\steering_file.txt" "$TARGET_PATH$" "$PROJ_DIR$\fir_filter_symbol.o"

- 此处 fir_filter_symbol.o 是要产出的符号定义文件，steering_file.txt 放在project_l0\iar_v8.2目录下，是用来选择要产生哪些函数的符号，需根据安全库区调用的内容进行手动编辑，内容如下，其中"show"是用来选择函数的命令

图 28. 编辑的 steering_file.txt 内容

```
show FIR_lowpass_filter
```

3.5 Project_L1：终端用户范例

Project_L1范例会使用到在Project_L0中调试好，并已经被刻录到AT32F437芯片的主闪存中且被SLIB保护的FIR低通滤波器函数。根据 Project_L0提供的头文件、符号定义文件以及主闪存区块映像，终端用户就可以参照Project_L1做到

- 建立一个应用项目
- 引用Project_L0提供的头文件及符号定义文件到项目里
- 调用FIR低通滤波器函数
- 开发并调试用户自己的应用程序

注意事项:

Project_L1必须使用跟Project_L0开发时一样的工具链及相同版本的编译程序，不然有可能会因为版本差异的兼容性问题，而无法使用Project_L0提供的代码。例如本范例中 Project_L0使用的是Keil®

µvision V5.18.0.0 ，那Project_L1也要使用同样的这个版本。

3.5.1 建立用户的应用项目

因为Project_L0启用的安全库区已经占用了一些特定的主闪存扇区，Project_L1的代码必须参照Project_L0提供的主闪存区块映像来编排放置的地址。图13为此范例的主闪存区块映射，其中sector 1至sector 2为安全库区所占用，终端用户需使用linker control file将这个区域隔离起来，避免代码在编译时被编排到这个区域内，方式如下：

Keil® µvision 的scatter file

可参照 project_l1\mdk_v5\ 目录的end_user_code.sct文件，将主闪存空间切成两个区块，中间空出来的区域就是SLIB保护区。此外，RAM的区域也要保留0x2005F000之后的区域。如下图

图 29. 修改后的 scatter 文件

```
LR_IROM1 0x08000000 0x00001000  {    ; load region size_region
  ER_IROM1 0x08000000 0x00001000  {  ; load address = execution address
   *.o (RESET, +First)
   *(InRoot$$Sections)
   .ANY (+RO)
  }
  RW_IRAM1 0x20000000 0x0005F000  {  ; RW data
   .ANY (+RW +ZI)
  }

 ; 0x2005F000 ~ 0x2005FFFF   RAM reserved for SLIB code

}

; 0x08001000 ~ 0x08002FFF is SLIB area

LR_IROM2 0x08003000 0x003ED000  {    ; load region size_region
  ER_IROM2 0x08003000 0x003ED000  {  ; load address = execution address
   .ANY (+RO)
  }
}
```

IAR 的ICF file

可参照 project_l1\iar_V8.2\ 目录下enduser.icf文件中如下图的部分

图 30. 修改后的 icf 文件

```
define region ROM_region      = mem:[from __ICFEDIT_region_ROM_start__   to __ICFEDIT_region_ROM_end__]
                              -mem:[from __ICFEDIT_region_SLIB_start__ to __ICFEDIT_region_SLIB_end__];

define region RAM_region      = mem:[from __ICFEDIT_region_RAM_start__   to __ICFEDIT_region_RAM_end__]
                              - mem:[from __ICFEDIT_region_SLIB_RAM_start__   to __ICFEDIT_region_SLIB_RAM_end__];
```

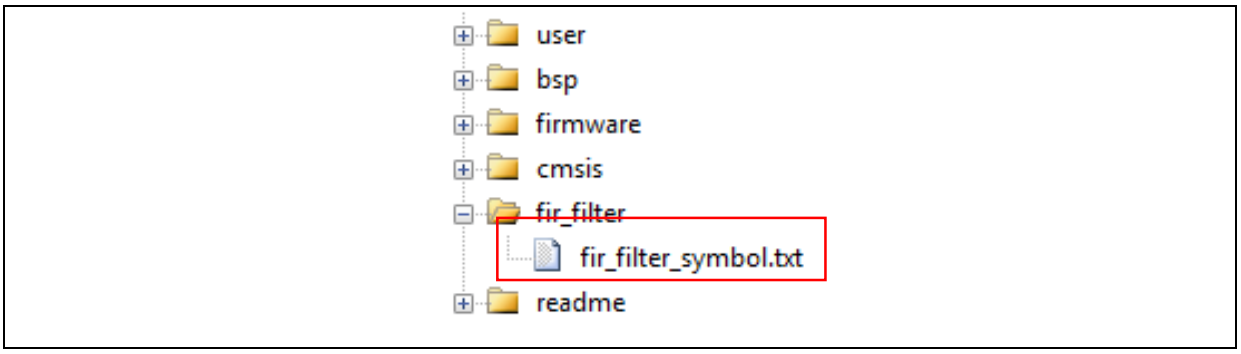
3.5.2 在项目中加入符号定义文件

Project_L0所产生的符号定义文件fir_filter_symbol.txt必须被添加到Project_L1项目中，才能被正确的编译并链结到SLIB保护区的代码。

在Keil® µvision中加入符号定义文件

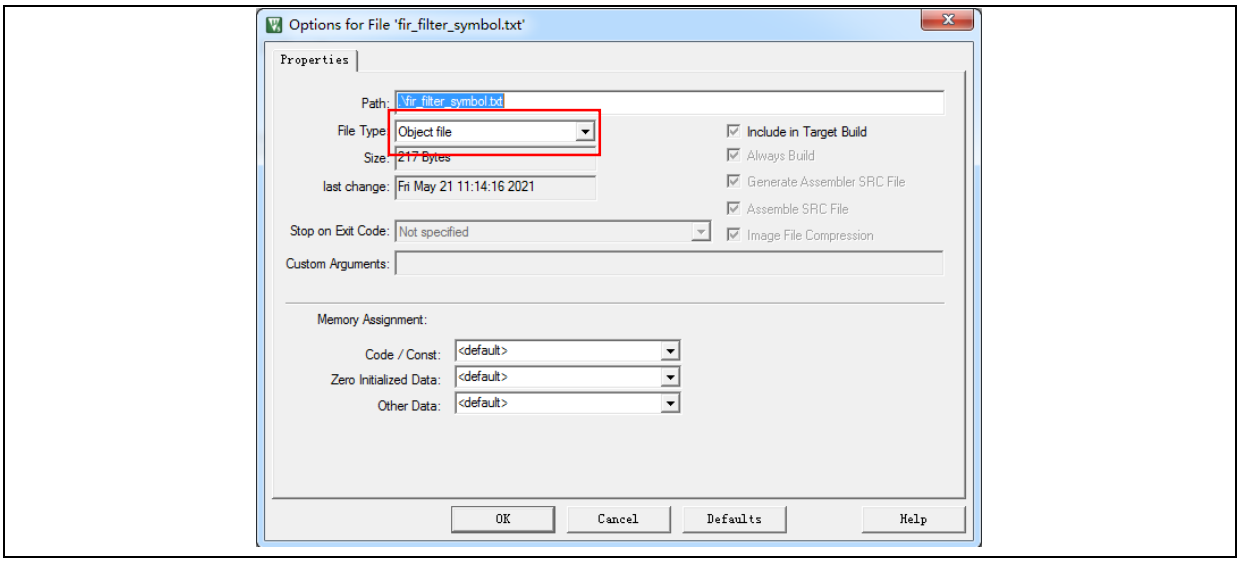
工程中添加fir_filter_symbol.txt这个符号定义文件，如下图:

图 31. 在 Keil 加入 symbol definition file



将文件加入fir_filter群组后，必须将它的文件类型更改为Object文件，而不是原来的文本(text)文件，修改方式如下

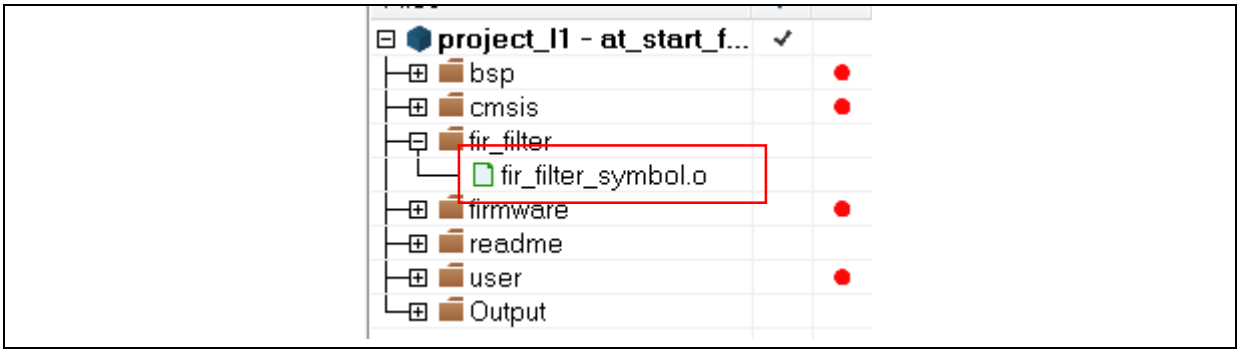
图 32. 修改符号定义文件的类型为 Object file



在IAR中加入符号定义文件

将fir_filter_symbol.o 这个Object文件加到加到fir_filter群组即可，如下图

图 33. 在 IAR 中加入符号定义文件



3.5.3 调用 SLIB 保护区的函数

当filter.h头文件被main.c引用且符号定义文件也正确地加入项目之后，保护区的低通波器函数就可以被调用。方式如下:

```
FIR_lowpass_filter(inputF32, outputF32, TEST_LENGTH_SAMPLES);
```

其中：

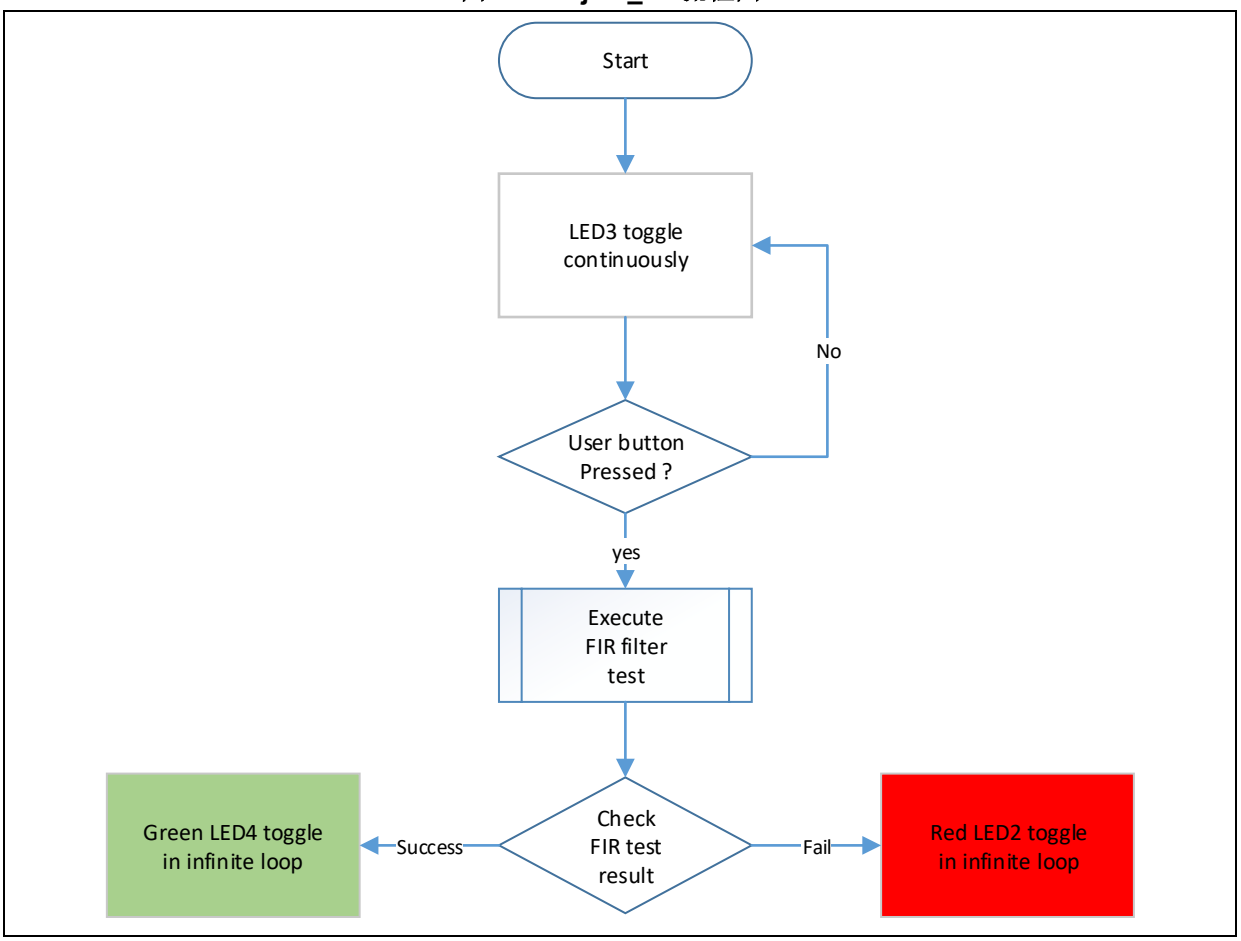
- *inputF32*：指向包含输入信号数据表的指针
- *outputF32*：指向存放输出信号数据表的指针
- *TEST_LENGTH_SAMPLES*：要被处理的信号样本数

3.5.4 Project_L1 执行流程

Project_L1的执行流程如下图，说明如下:

- 开始执行后LED3灯会持续闪烁
- 按下AT-START板子上的USER 按键，FIR_lowpass_filter()开始做运算
- 如运算结果正确，绿色LED4灯持续闪烁，如运算结果错误则红色LED2灯持续闪烁

图 34. Project_L1 流程图

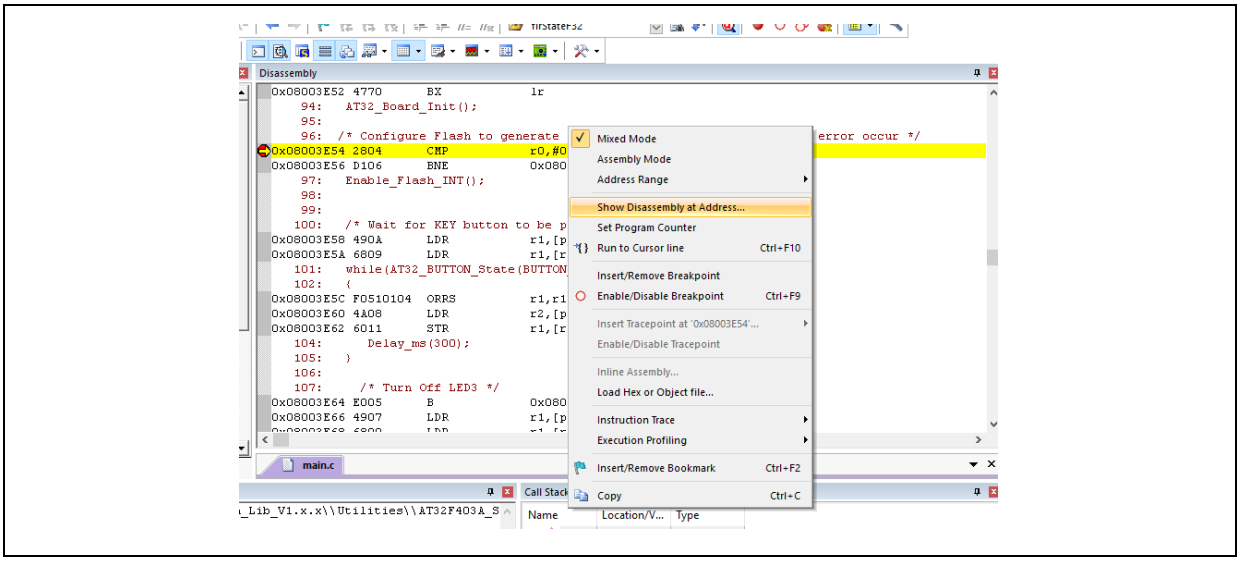


3.5.5 调试模式下的 SLIB 保护

当终端用户在开发应用程序时，会用到开发工具调试代码，以下将以Keil® µvision为例， 说明在调适模式下，SLIB如何防止保护区内的代码被以数据的方式读取

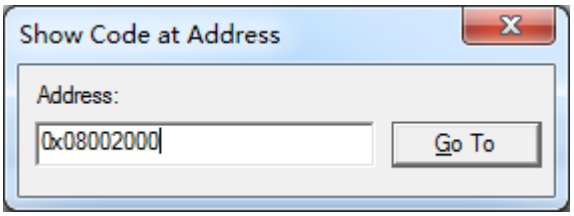
- 开启Project_L1项目并重新编译
- 点击”Start/Stop Debug Session”进入调试模式
- 在”Disassembly”窗口点击数标右键，然后选择”Show Disassembly at Address”，如下图

图 35. 进入 Show Disassembly at Address



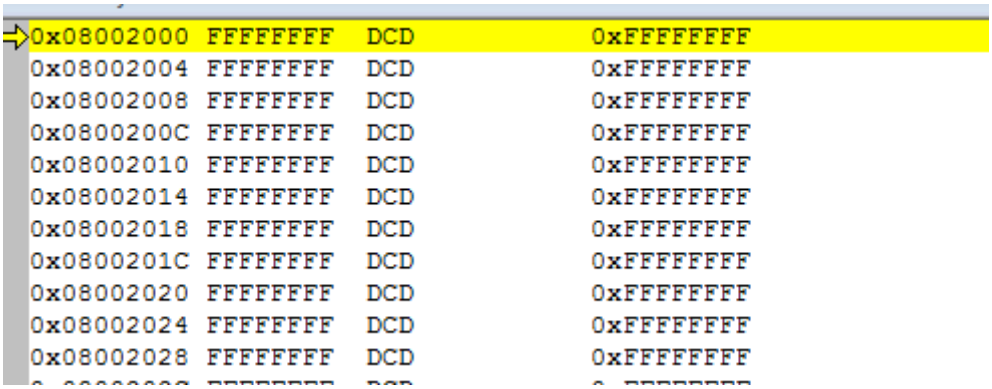
- 输入SLIB_INSTRUCTION起始扇区sector 2的地址0x08002000

图 36. 设置 Show Code at Address



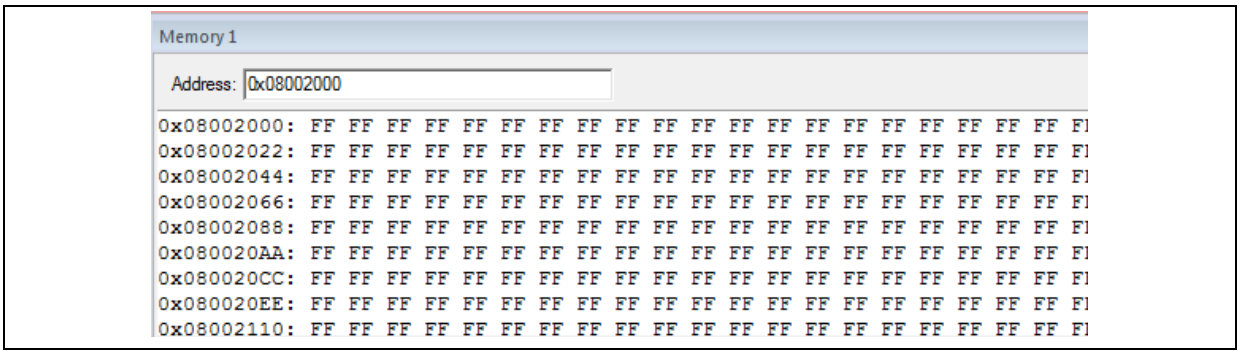
可以看到地址0x08002000这里开始，看到的代码都是0xFFFFFFFF

图 37. 代码查看



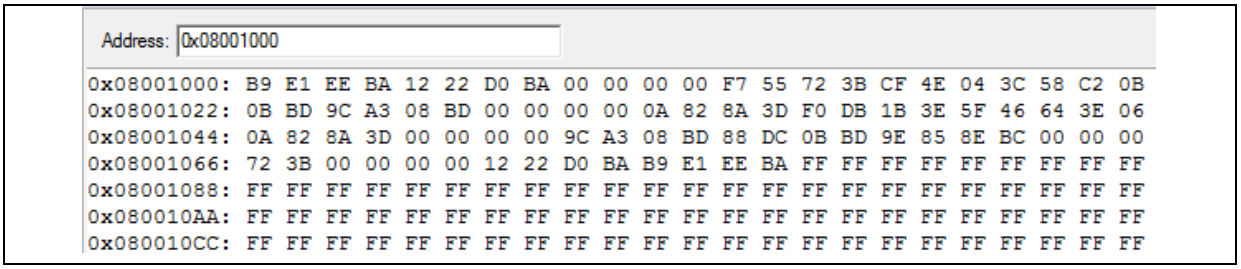
- 同样地在Memory窗口输入0x08002000的地址，也会看到全部是 0xFF

图 38. Memory 窗口查看代码



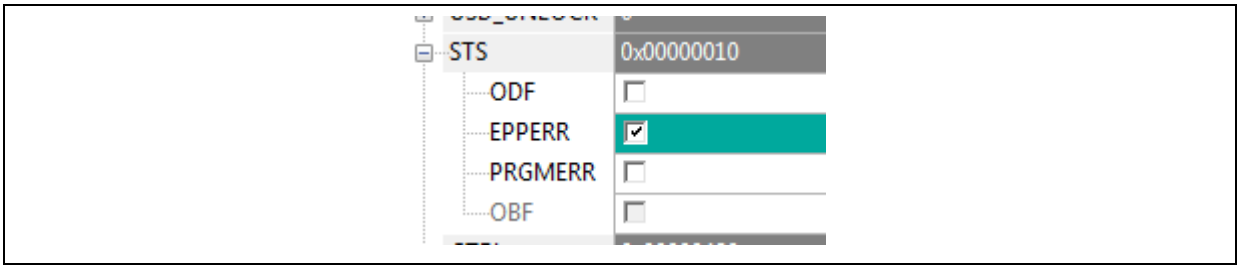
- 在Memory窗口，输入SLIB_READ_ONLY起始扇区sector 1的地址0x08001000，因为这个区块允许被D-Code数据总线读取，所以可以看到原来的数值

图 39. Memory 窗口查看 SLIB_READ_ONLY 起始页面



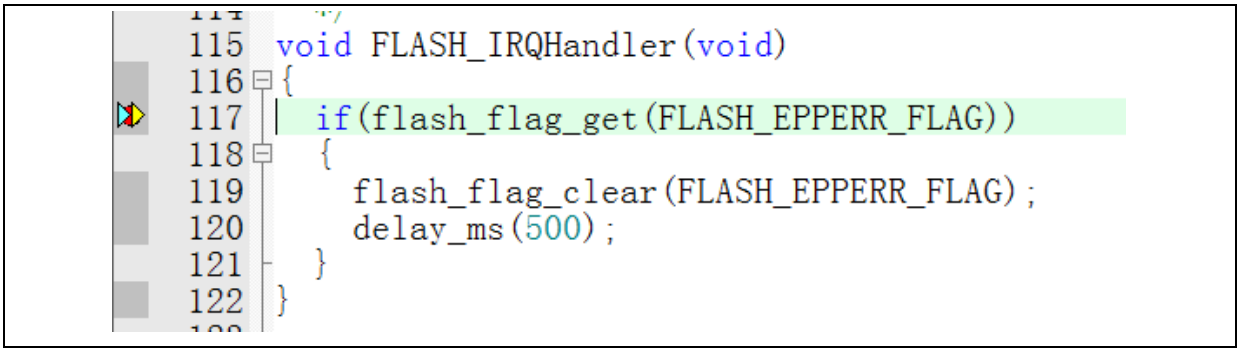
在Memory窗口用鼠标双击0x08002000的数据尝试做修改，FLASH_STS寄存器的EPPERR位置”1”提出警告，显示写保护发生作用

图 40. SLIB 写测试



如果有使能写保护错误中断，继续执行程序就会进入中断程序里面

图 41. 写保护错误中断



```
115  void FLASH_IRQHandler(void)
116  {
117    if(flash_flag_get(FLASH_EPPERR_FLAG))
118    {
119      flash_flag_clear(FLASH_EPPERR_FLAG);
120      delay_ms(500);
121    }
122  }
```

4 方案商和终端用户代码整合及下载操作流程

方案商和终端用户的代码设计完成后，需要下载到同一个MCU中，这就涉及到各自代码的安全性问题。以下列举两种常用下载操作流程供用户参考，仍然是以上面Project_L0和Project_L1为例。

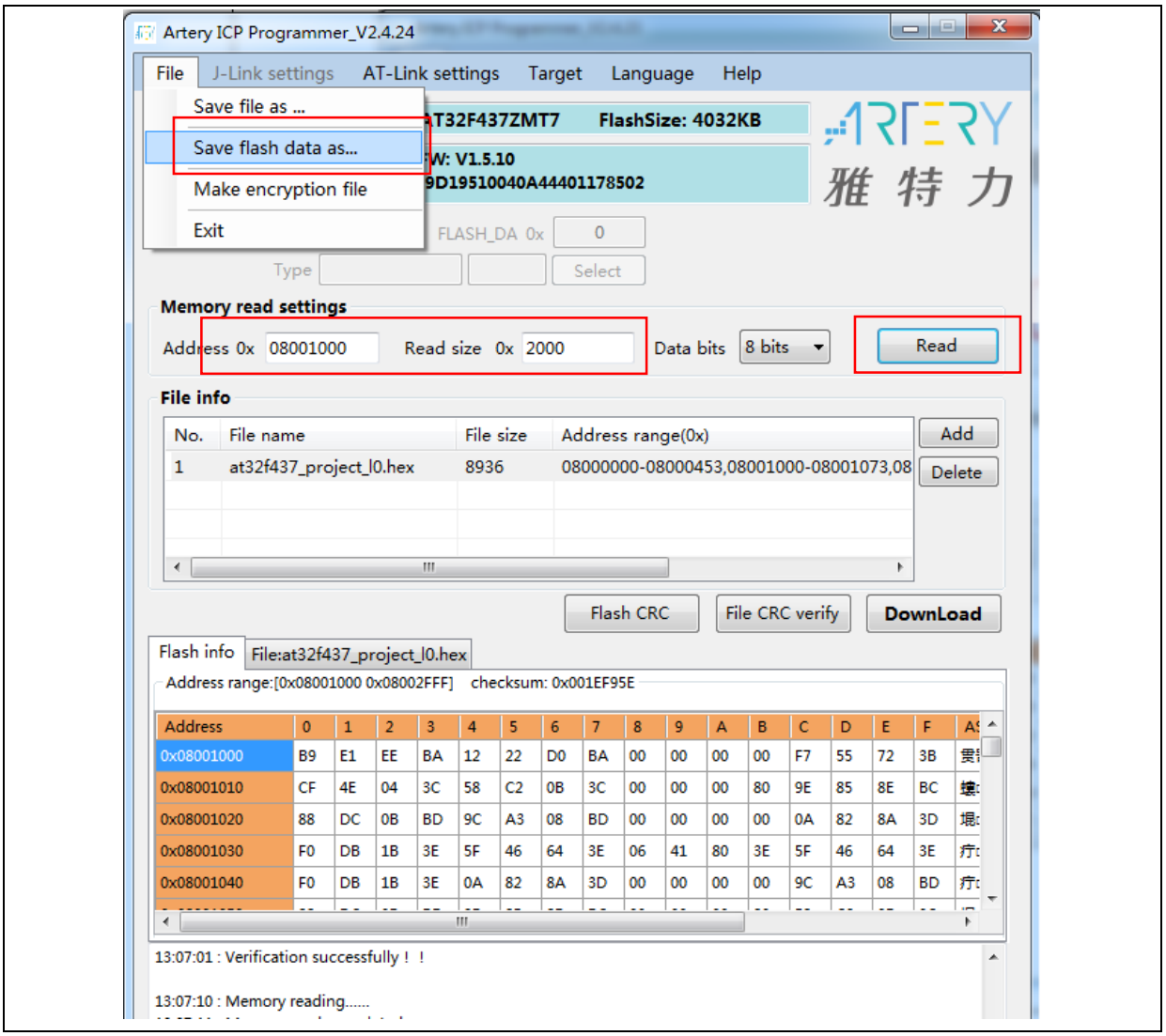
操作中涉及到AT-Link的离线下载模式，详细描述可以参考ICP使用文档及AT-Link使用文档。

4.1 方案商和终端用户代码分别烧录

方案商先烧录SLIB代码到MCU，然后终端用户再烧录应用代码到MCU，步骤如下：

(1) **方法A：**方案商将编译完成的工程中SLIB部分的代码通过ICP截取保存成BIN或者HEX档：先将整个工程下载到MCU(此时不配置SLIB及FAP等信息)，然后通过存储器读取功能读取对应SLIB部分代码(0x08001000~0x08002FFF)，再通过ICP软件操作文件-存储器数据另存为将其保存为BIN或者HEX，例程中BIN档命名为slib.bin，如下图。

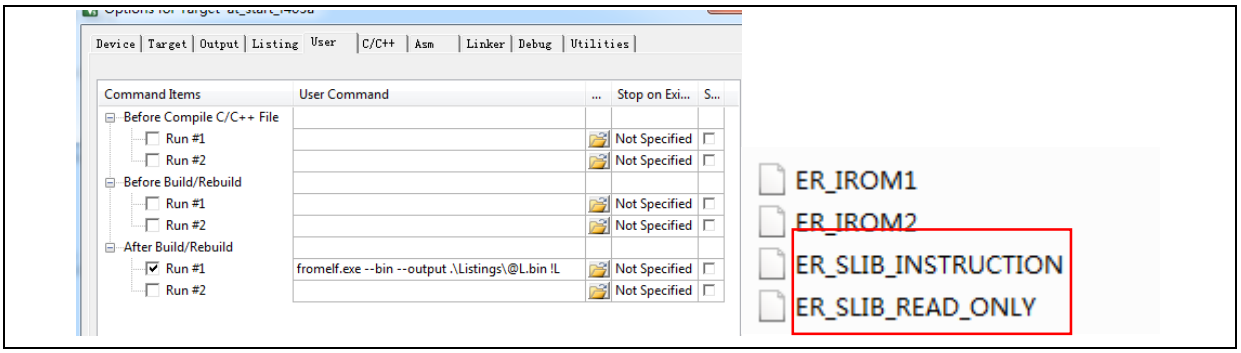
图 42. 保存 SLIB 代码



方法B：方案商将编译完成的工程直接产生bin格式的文件，取其中SLIB区域对应的一段，例如在KEIL工程中，user选项中添加fromelf.exe --bin --output .\Listings\@L.bin !L，生成对应固件的bin档，将对应的SLIB段文件添加后缀名.bin格式，本例中改为ER_SLIB_INSTRUCTION.bin和ER_SLIB_READ_ONLY.bin，对应的就是0x08002000地址段的SLIB-INSTRUCTION文件和

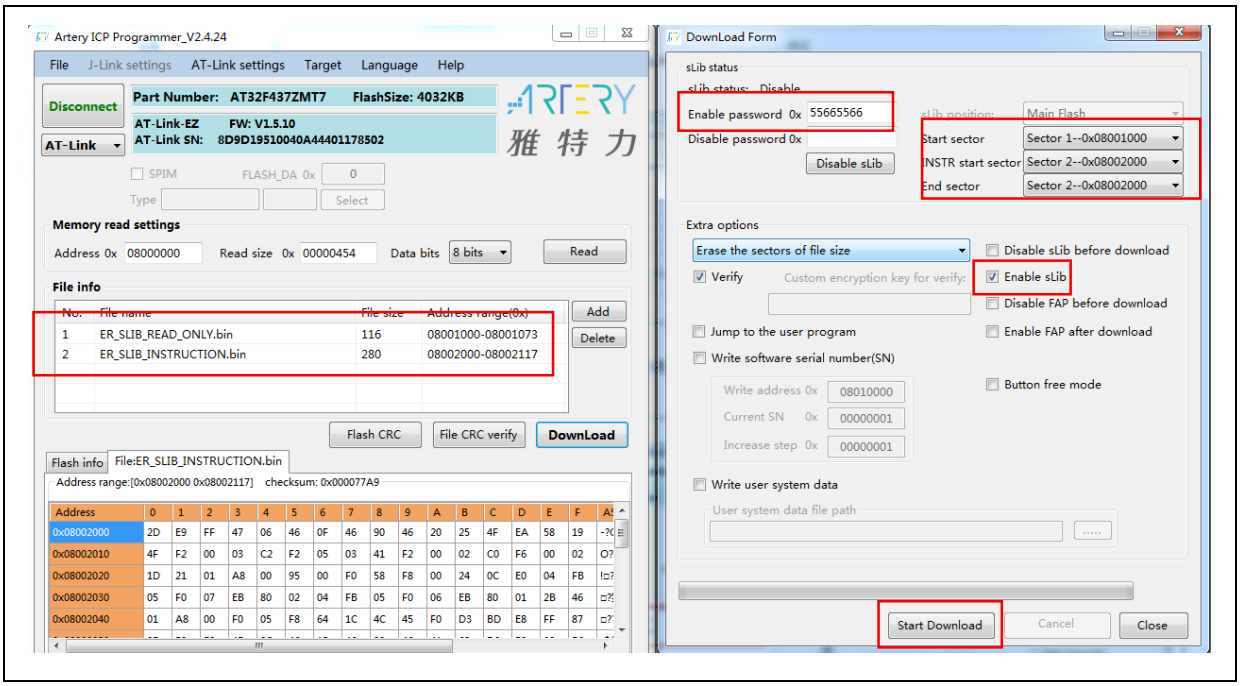
0x08001000地址段的SLIB-READ-ONLY文件，如下图

图 43. 生成 SLIB 代码部分 bin 文件



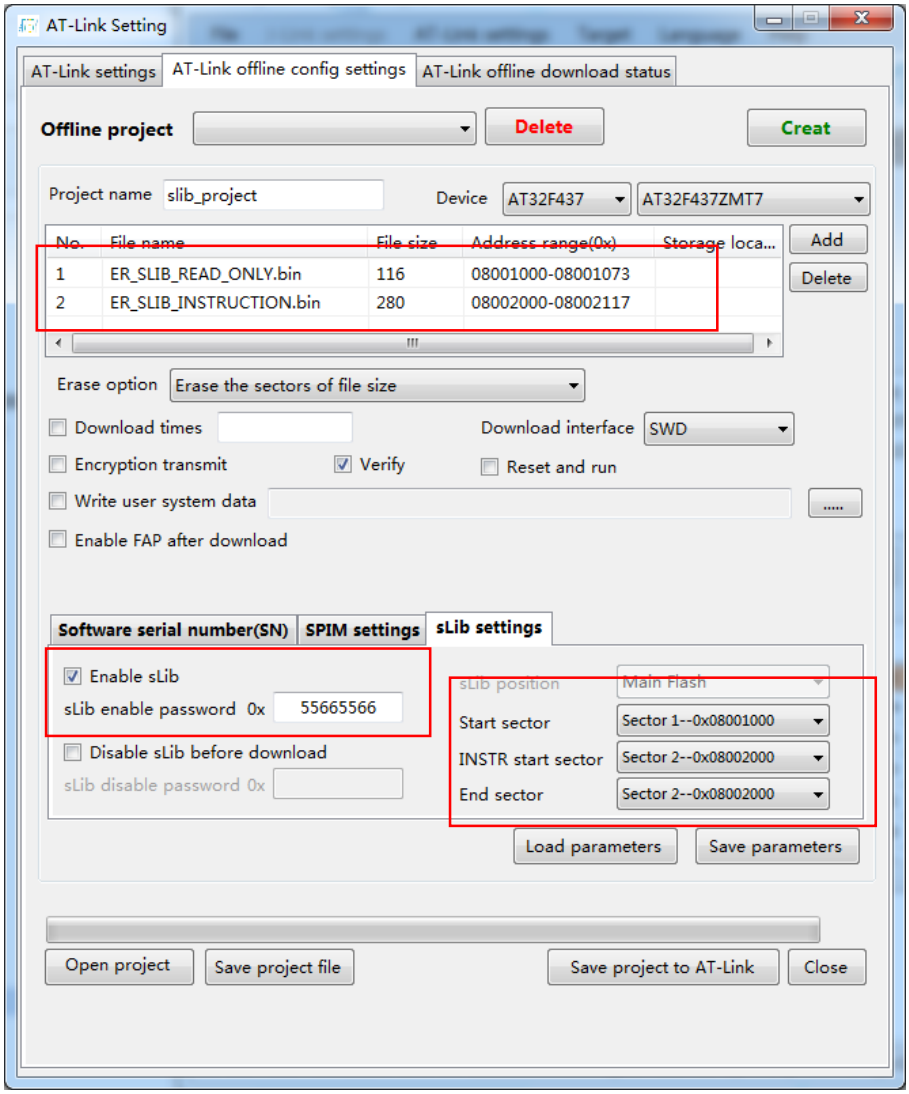
(2) 将bin通过ICP工具，在线烧录到MCU，如下图

图 44. ICP 在线烧录 MCU



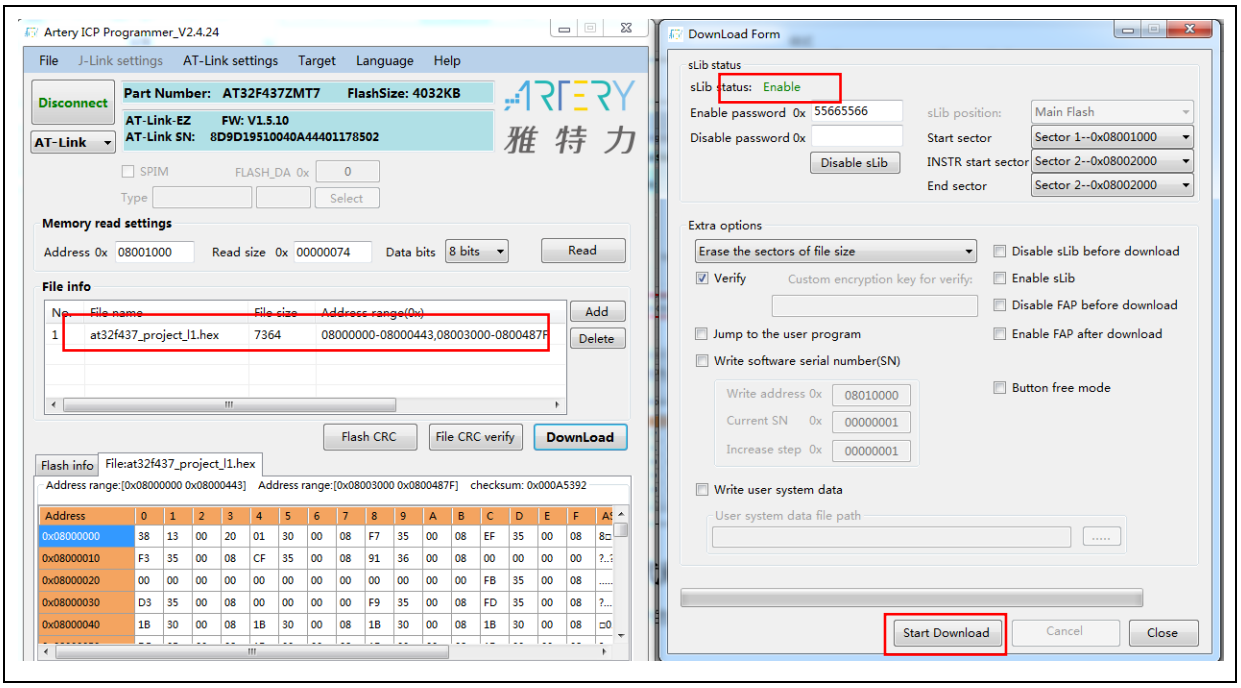
(3) 或者通过ICP工具配置成离线项目工程保存到AT-Link，然后经过AT-Link离线烧录到MCU，保存离线项目工程如下图。

图 45. AT-Link 离线烧录到 MCU



(4) 经过步骤2或者步骤3，终端用户拿到烧录好SLIB部分的MCU，此时SLIB状态会显示为已启用，终端用户通过在线烧录或者离线烧录应用代码到MCU完成整个过程，在线烧录如下图。

图 46. 终端用户烧录代码到 MCU



4.2 方案商和终端用户代码合并烧录

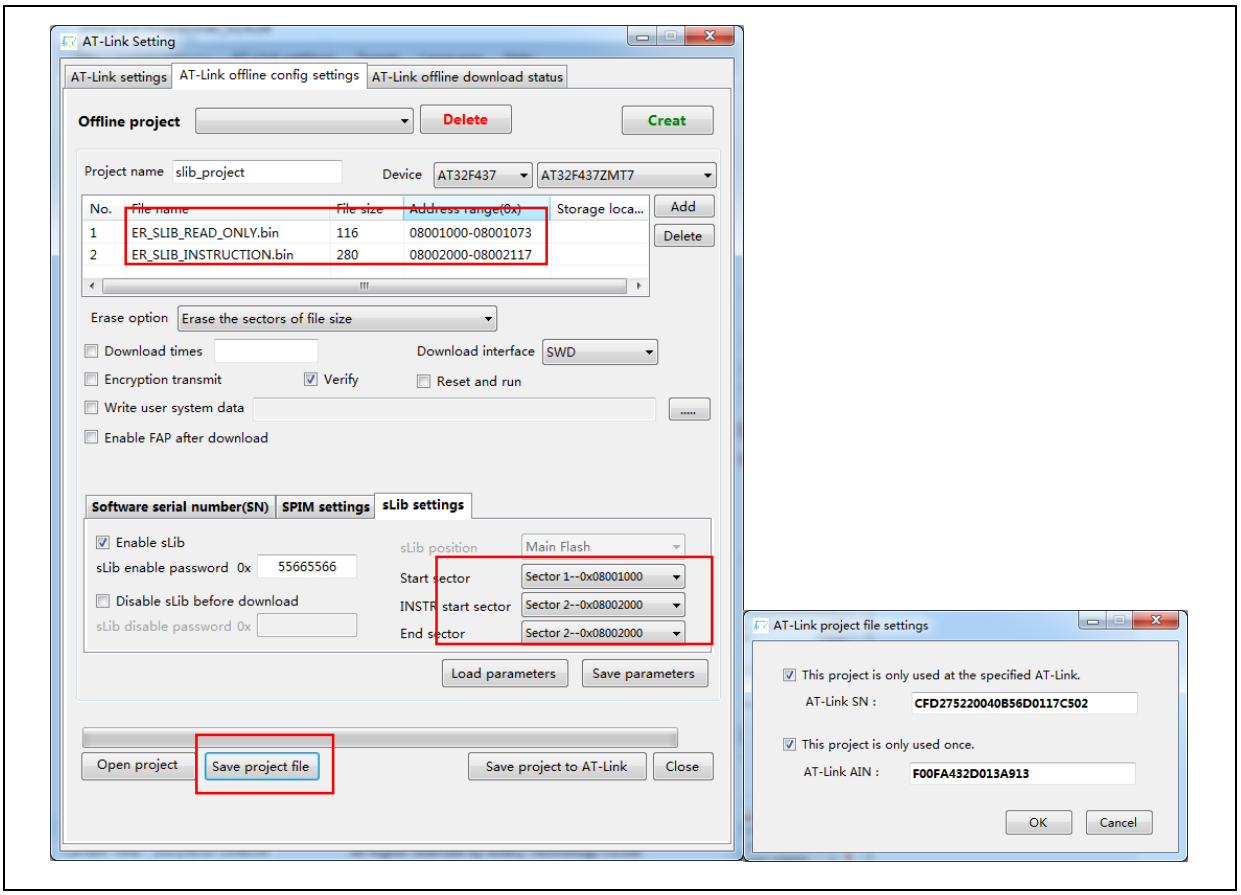
方案商的SLIB代码和终端用户的应用代码整合到一个离线项目工程中，通过AT-Link离线烧录一次下载到MCU，步骤如下：

(1) 方案商将编译完成的工程按照上一章节所述方法处理，得到SLIB部分的BIN档。

(2) 方案商通过ICP制作离线项目工程并保存到PC，可以根据最终需求配置各种参数，比如限制下载次数、项目文件绑定AT-Link、下载完成后开启FAP等，保存离线项目工程如下图

注意：离线项目工程本身已经经过加密，为进一步提升安全性，方案商还可以将slib.bin制作成加密的slib.benc文件再添加到离线项目工程中，但此时的离线项目工程只能在对应匹配加密秘钥的AT-Link上才能使用

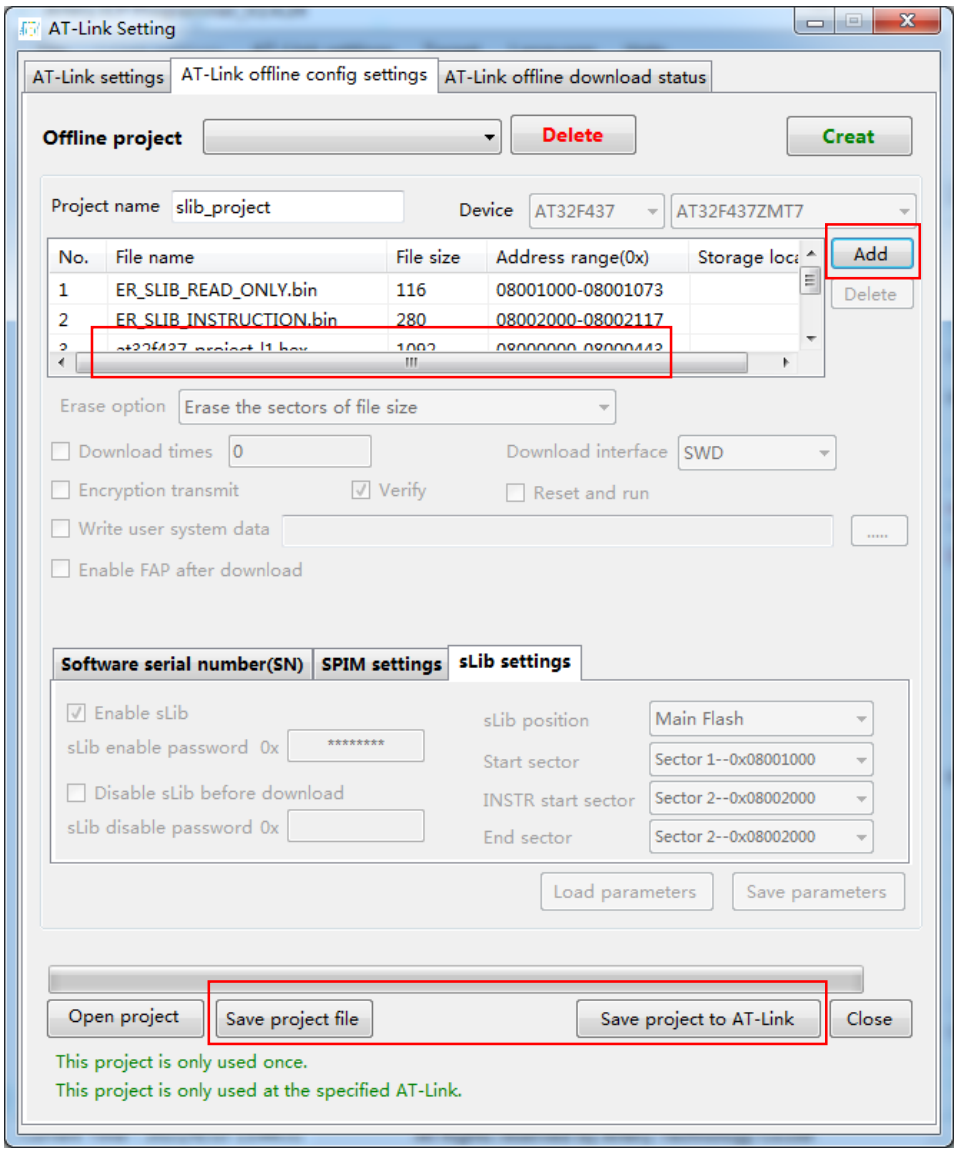
图 47. 制作离线项目工程



(3) 终端用户拿到该离线项目工程，用ICP打开项目文件，通过添加文件功能，可以添加应用代码部分到该离线项目工程，然后再保存到PC或者直接存储到AT-Link，通过执行离线下载完成操作，项目文件添加方法如下图。

注意：为防止代码泄露被破解等风险，离线项目工程添加代码文件时其余配置都不可更改，所以需要方案商预先将最终配置设置好。

图 48. 添加项目文件


AT-Link Setting
AT-Link settings
AT-Link offline config settings
AT-Link offline download status
Offline project
Delete
Creat
Project name
slib_project
Device
AT32F437
AT32F437ZMT7
No.
File name
File size
Address range(0x)
Storage loca
Add
Delete
1
ER_SLIB_READ_ONLY.bin
116
08001000-08001073
2
ER_SLIB_INSTRUCTION.bin
280
08002000-08002117
Erase option
Erase the sectors of file size
Download times
0
Download interface
SWD
Encryption transmit
Verify
Reset and run
Write user system data
Enable FAP after download
Software serial number(SN)
SPIM settings
sLib settings
Enable sLib
sLib enable password 0x
****
Disable sLib before download
sLib disable password 0x
sLib position
Main Flash
Start sector
Sector 1--0x08001000
INSTR start sector
Sector 2--0x08002000
End sector
Sector 2--0x08002000
Load parameters
Save parameters
Open project
Save project file
Save project to AT-Link
Close
This project is only used once.
This project is only used at the specified AT-Link.

5 版本历史

表 2. 文档版本历史

日期	版本	变更
2021.9.8	2.0.0	初始版本

重要通知 - 请仔细阅读

买方自行负责对本文所述雅特力产品和服务的选择和使用，雅特力概不承担与选择或使用本文所述雅特力产品和服务相关的任何责任。

无论之前是否有过任何形式的表示，本文档不以任何方式对任何知识产权进行任何明示或默示的授权或许可。如果本文档任何部分涉及任何第三方产品或服务，不应被视为雅特力授权使用此类第三方产品或服务，或许可其中的任何知识产权，或者被视为涉及以任何方式使用任何此类第三方产品或服务或其中任何知识产权的保证。

除非在雅特力的销售条款中另有说明，否则，雅特力对雅特力产品的使用和/或销售不做任何明示或默示的保证，包括但不限于有关适销性、适合特定用途（及其依据任何司法管辖区的法律的对应情况），或侵犯任何专利、版权或其他知识产权的默示保证。

雅特力产品并非设计或专门用于下列用途的产品：（A）对安全性有特别要求的应用，例如：生命支持、主动植入设备或对产品功能安全有要求的系统；（B）航空应用；（C）航天应用或航天环境；（D）武器，且/或（E）其他可能导致人身伤害、死亡及财产损害的应用。如果采购商擅自将其用于前述应用，即使采购商向雅特力发出了书面通知，风险及法律责任仍将由采购商单独承担，且采购商应独力负责在前述应用中满足所有法律和法规要求。

经销的雅特力产品如有不同于本文档中提出的声明和/或技术特点的规定，将立即导致雅特力针对本文所述雅特力产品或服务授予的任何保证失效，并且不应以任何形式造成或扩大雅特力的任何责任。