AN0189
Application Note

AT32A403A Security Library Application Note

Introduction

This application note introduces the security library (sLib) application principle of AT32A403 AMCUs, operation methods and example projects.

Applicable products:

Part number	AT32A403A

Contents

List of tables

List of figures

1 Overview

As more and more MCU applications require complex algorithms and middleware solutions, it has become an important issue that how to protect IP-Codes (such as core algorithms) developed by software solution providers.

The AT32A403Aseries MCUs are designed with a security library (sLib) to protect important IP Codes against being changed or read by the end user’s program.

This application note details the sLib application principle and operation methods of AT32A403A MCUs.

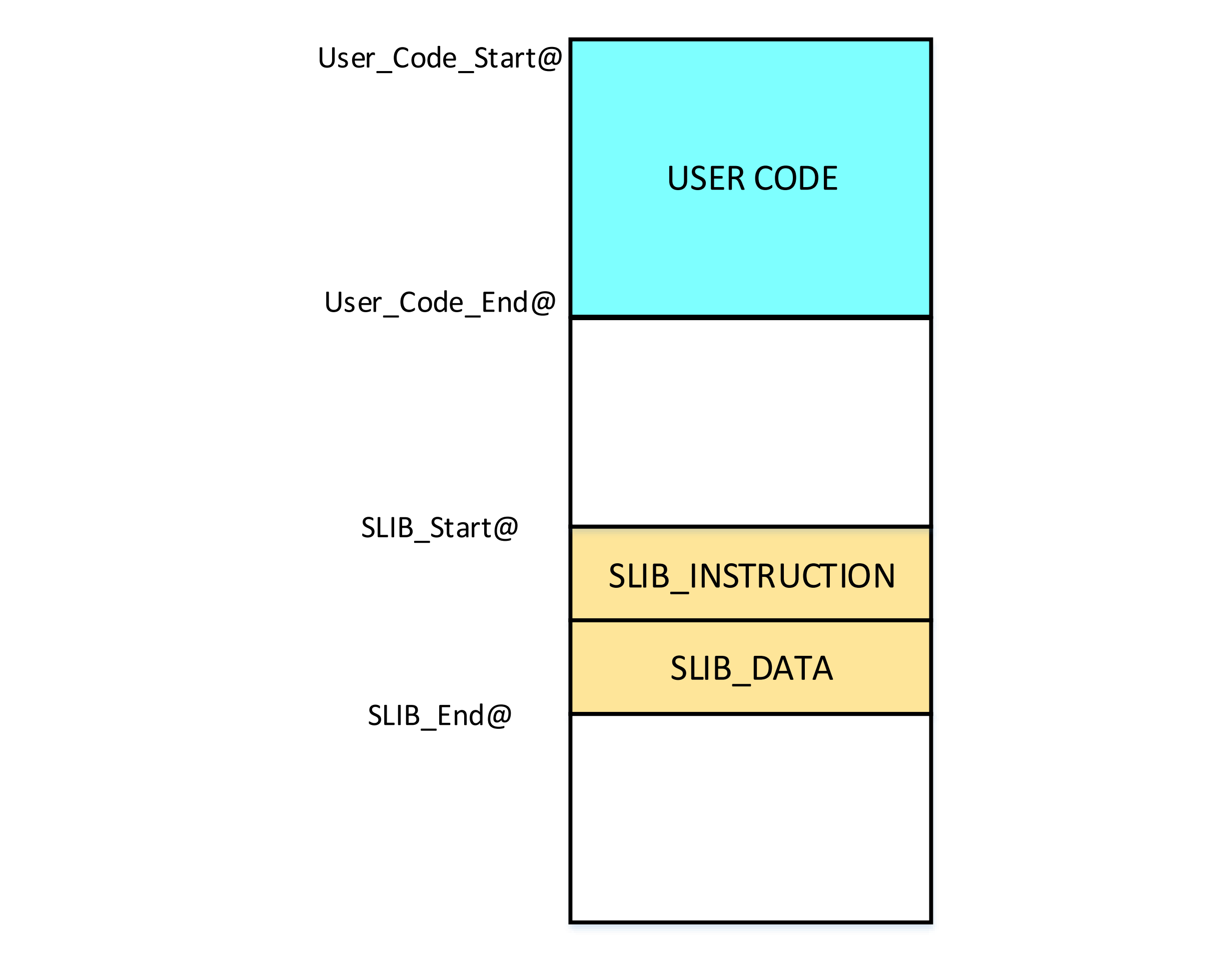
2 Application principles

2.1 Application principle of sLib

- Security library is a defined area protected by a code in the main memory, so that solution providers can program core algorithm into this area, and the rest of the area can be used for secondary development by end customers.
- Security library contains instruction security library (SLIB_INSTRUCTION) and data security library (SLIB_DATA); users can select part of or the whole security library for instruction storage, but using the whole security library for storing data is not supported.
- Program codes in the instruction security library (SLIB_INSTRUCTION) can only be fetched though I-Code bus (can only be executed) but not be read through D-Code bus (including ISP/ICP debug mode and programs that boot from internal RAM). When reading the SLIB_INSTRUCTION area, values are all read 0xFF.
- The data security library (SLIB_DATA) can only be read through D-Code bus, but data cannot be written to it.
- Program codes and data in the security library cannot be erased unless the correct code is keyed in. If a wrong code is keyed in, in an attempt of writing or erasing the security library, a warning message will be issued by EPPERR=1 in the FLASH_STS register.
- The program code and data in security library are not erased when the end users perform a mass erase on the main Flash memory.
- Users can write the programmed password in the SLIB_PWD_CLR register to disable security library protection. When the security library protection is disabled, the chip will perform a mass erase on the main Flash memory (including the contents of security library). Therefore, even if the code defined by the software solution provider is leaked, the program code will not be leaked.

The mapping of main Flash memory featured with sLib is shown in Figure 1. The program codes in security library can be easily called and executed by end users, but cannot be read directly.

Figure 1. Mapping of main Flash memory designed with sLib



The security library is set by sectors, and the size of each sector is subject to the specific MCUs. Table 1 lists the main Flash size, sector size and configurable range of AT32A403A series MCUs.

Table 1. Flash size of AT32A403A series

Model	Internal Flash size (Byte)	Sector size (Byte)	Configurable range
AT32A403AxC	256K	2K	Sector 2 ~ 63 (0x08001000 ~ 0x0801FFFF)
AT32A403AxE	512K	2K	Sector 2 ~ 63 (0x08001000 ~ 0x0801FFFF)
AT32A403AxG	1024K	2K	Sector 2 ~ 63 (0x08001000 ~ 0x0801FFFF)

2.2 How to enable sLib protection

By default, security library setting register is unreadable and write-protected. To enable write access to this register, security library should be unlocked first by writing 0xA35F6D24 to the SLIB_UNLOCK register. Then check the SLIB_ULKF bit in the SLIB_MISC_STS register to verify if it is unlocked successfully. Then, write the programmed value into the security library setting register.

The steps to enable security library are as follows:

- Check the OBF bit in the FLASH_STS register to confirm that there is no other ongoing programming operation;
- Write 0xA35F6D24 to the SLIB_UNLOCK register to unlock security library;
- Check the SLIB_ULKF bit in the SLIB_MISC_STS register to verify that it is unlocked successfully;
- Set the area to be protected in the SLIB_SET_RANGE register, including the addresses of instruction area and data area;
- Wait until the OBF bit becomes “0”;
- Set security library password in the SLIB_SET_PWD register;
- Wait until the OBF bit becomes “0”;
- Program the code to be saved in security library;
- Perform system reset, and then reload security library setting word;
- Read the SLIB_STS0/STS1 registers to verify the security library setting.

Note:

- It is allowed to set security library in the main Flash memory; refer to Table 1 for the configuration range;
- The security library code must be programmed by sectors, with its start address aligned with the main Flash memory address;
- The interrupt vector table is in data type and usually placed in the first sector (sector0, which should not be configured as security library) of the main Flash memory.
- Program codes to be protected by the security library should not be placed in the first 4 KB of the Flash memory.

For details about security library setting register, refer to *AT32A403A Series Reference Manual*.

The security library can be enabled by the *slib_enable()* function in main.c of project_l0. In addition, users can use Artery ICP or ISP tools for configuration.

2.3 How to disable sLib protection

The security library protection can be disabled by writing the previously programmed password to the SLIB_PWD_CLR register. While disabling security library protection, MCU will perform mass erase on the main Flash memory (including the contents of security library).

The steps to disable main Flash security library are as follows:

- Check the OBF bit in the FLASH_STS register to confirm that there is no other ongoing programming operation;
- Write the programmed password to the SLIB_PWD_CLR register;
- Perform a system reset, and then reload security library setting word;
- Read the SLIB_STS0 register to verify the security library setting.

2.4 Compile and execute program in sLib

As aforementioned, program codes in the instruction security library (SLIB_INSTRUCTION) can be fetched by MCU via I-Code bus but cannot be read via D-Code bus, which means that program codes in SLIB_INSTRUCTION cannot read the data saved in the same SLIB_INSTRUCTION. For example, literal pool, branch table or constant compiled from C program code in the SLIB_INSTRUCTION cannot be read via D-Code bus.

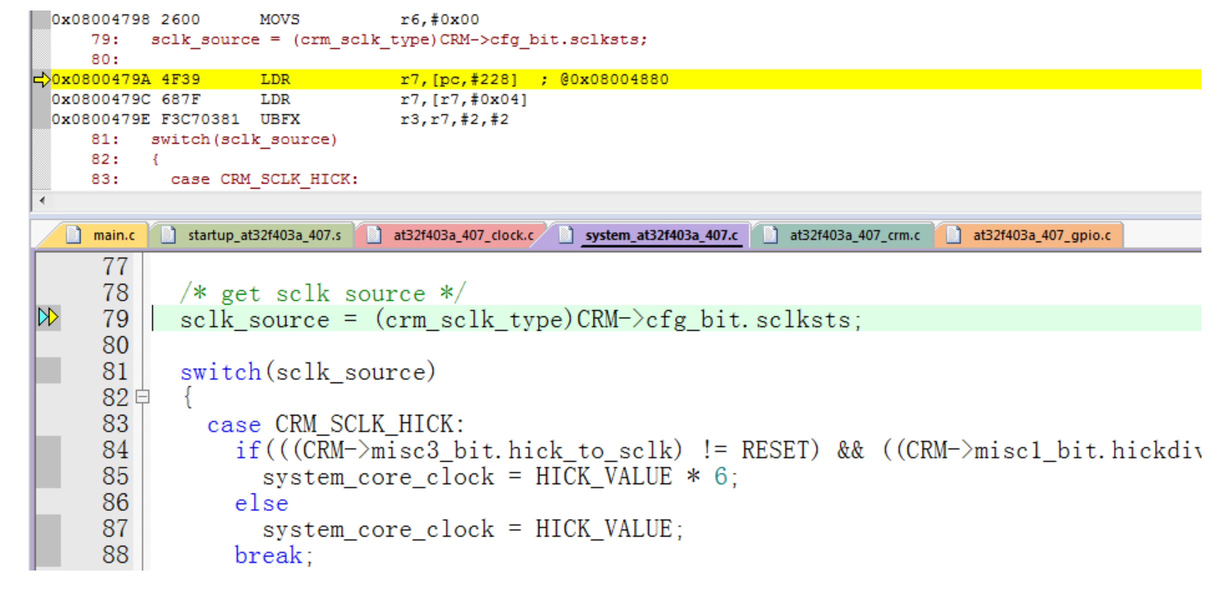
Only instructions rather than data can be placed in the instruction security library. Therefore, when compiling program codes to be placed in the instruction security library, the user must configure the compiler to generate execute-only codes to avoid generating the above mentioned data.

Figure 2 and Figure 3 show the examples of literal pool and branch table.

The “switch()” is a jump instruction in C program, and the “sclk_source” variable is used to read the CRM_CFG register. As shown in Figure 2, the compiled assembly code “LDR R7, [PC, #288]” obtains the address of the CRM_CFG register in a PC (program counter) indirect addressing manner, and the address of the CRM_CFG register is saved as a constant in the adjacent instruction area (within the instruction security library); therefore, the data is read when the “switch()” instruction is executed. An error will occur if there is such program code in the instruction security library.

The example program in Section 3 introduces how to configure compiler settings to avoid error.

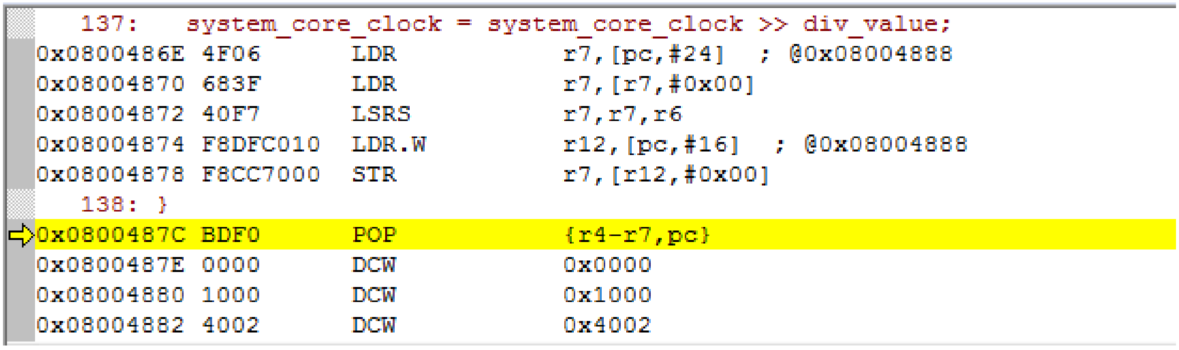
Figure 2. Literal pool example (1)



```
0x08004798 2600      MOVS          r6,#0x00
    79:    sclk_source = (crm_sclk_type)CRM->cfg_bit.sclksts;
    80:
0x0800479A 4F39      LDR           r7,[pc,#228]  ; @0x08004880
0x0800479C 687F      LDR           r7,[r7,#0x04]
0x0800479E F3C70381  UBFX          r3,r7,#2,#2
    81:    switch(sclk_source)
    82:    {
    83:      case CRM_SCLK_HICK:
```

```

  /* get sclk source */
  sclk_source = (crm_sclk_type)CRM->cfg_bit.sclksts;

  switch(sclk_source)
  {
    case CRM_SCLK_HICK:
      if(((CRM->misc3_bit.hick_to_sclk) != RESET) && ((CRM->misc1_bit.hickdi
        system_core_clock = HICK_VALUE * 6;
      else
        system_core_clock = HICK_VALUE;
      break;
```

Figure 3. Literal pool example (2)



```
   137:    system_core_clock = system_core_clock >> div_value;
0x0800486E 4F06      LDR          r7,[pc,#24]   ; @0x08004888
0x08004870 683F      LDR          r7,[r7,#0x00]
0x08004872 40F7      LSRS         r7,r7,r6
0x08004874 F8DFC010  LDR.W        r12,[pc,#16]   ; @0x08004888
0x08004878 F8CC7000  STR          r7,[r12,#0x00]
   138: }
0x0800487C BDF0      POP          {r4-r7,pc}
0x0800487E 0000      DCW          0x0000
0x08004880 1000      DCW          0x1000
0x08004882 4002      DCW          0x4002
```

2.4.1 Setting interrupt vector table as sLib area not allowed

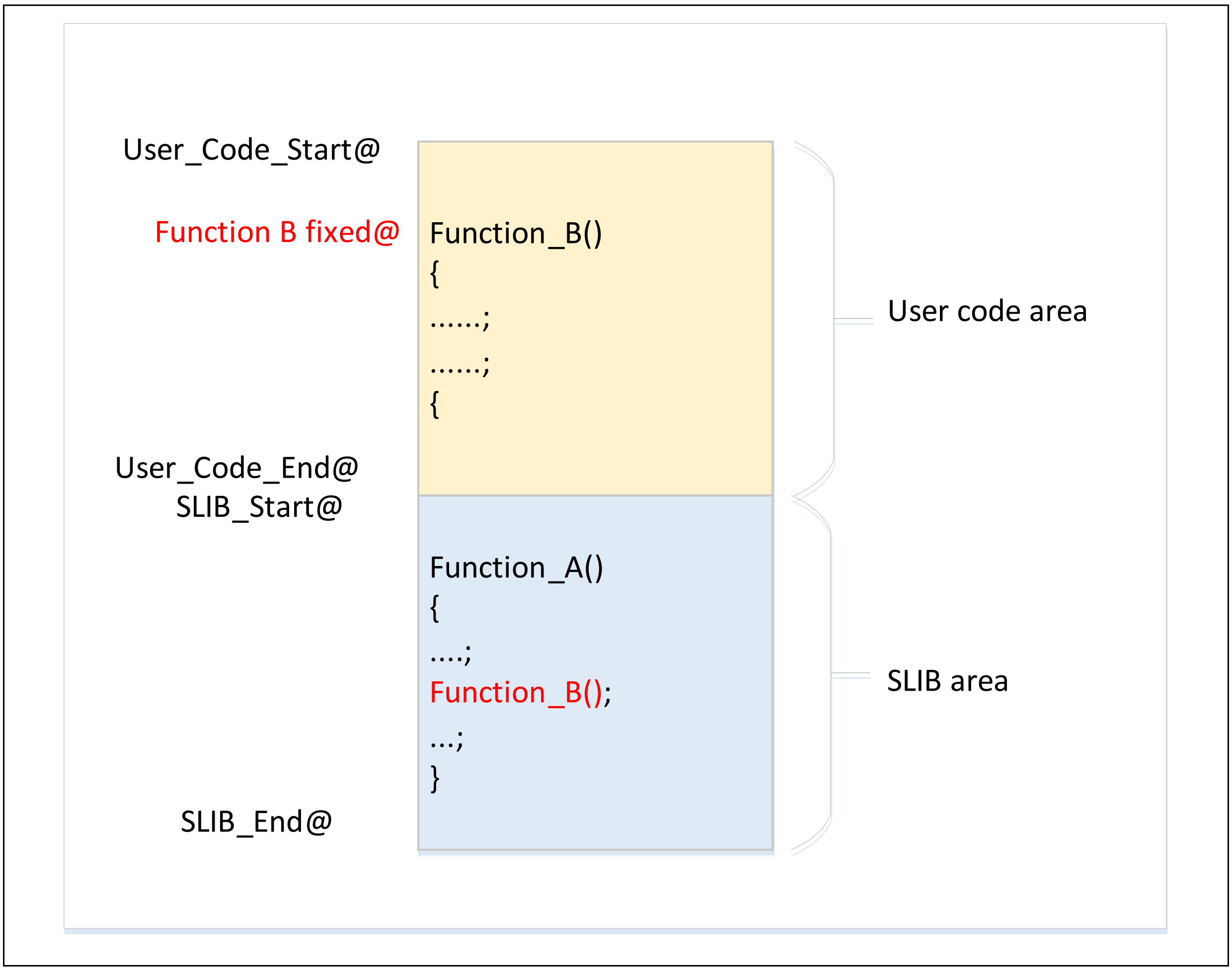
The interrupt vector table contains entry point address of each interrupt handler, which is read by MCU via D-Code bus. Generally, the interrupt vector table is located in the first sector (sector0, starting address: 0x08000000). Therefore, the following rules must be followed when setting the instruction security library:

- Do not configure the first sector of the main Flash memory as sLib area
- Program codes to be protection by security library cannot be placed in the first section of the main Flash memory.

2.4.2 Correlation between sLib area and user code area

Program code (IP-code) protected by sLib area can call functions from the function library located in user code area (outside the sLib area). In this case, these function addresses are contained in the IP-Code, allowing PC (program counter) to jump to these functions when IP-Code is executed. Once the sLib area is enabled, function address cannot be changed. At this point, addresses of functions in the user code area must be fixed; otherwise, PC will jump to a wrong address and cannot work properly. Therefore, when configuring the sLib area, all functions related to IP-Code should be compiled into the sLib area. Figure 4 gives an example of the protected *Function_A()* being called to *Function_B()* in the user code area.

Figure 4. Example of function in sLib area calling the function in user code area



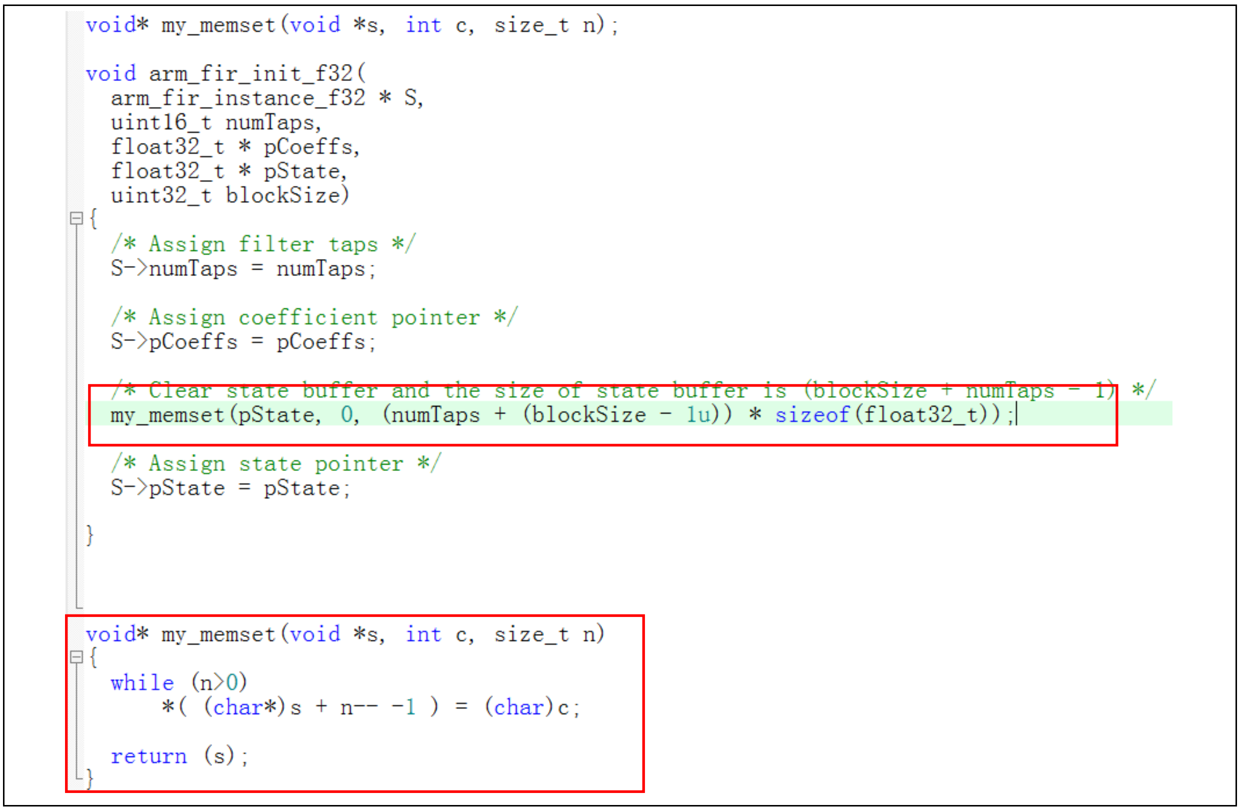
In addition, the standard function library of C programming language is commonly used, such as *memset()* and *memcpy()* functions. If both IP-Code and user area code call such functions, the above mentioned error may occur.

Recommended solutions:

1) Compile into the sLib area (refer to Keil or IAR documents for details).

2) Do not use the standard function library of C programming language in IP-Code. If it is necessary to use in IP-Code, functions to be used must be renamed. Figure 5 shows an example of writing the *my_memset()* function to replace the original *memset()* in IP-Code.

Figure 5. Example of user-defined function



```
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 Examples of sLib application

This section introduces examples of sLib application and related configurations.

3.1 Requirements

3.1.1 Hardware

- AT-START-A403A demo board with AT32A403AVGT7 chip
- AT-Link emulator for debugging

3.1.2 Software

- Keil® µvision IDE (µvision V5.36.0.0 is used in this example) or IAR Embedded workbench IDE (IAR V8.22.2 is used in this example)
- Artery ICP or ISP programming tool for enabling and disabling sLib

3.2 Overview

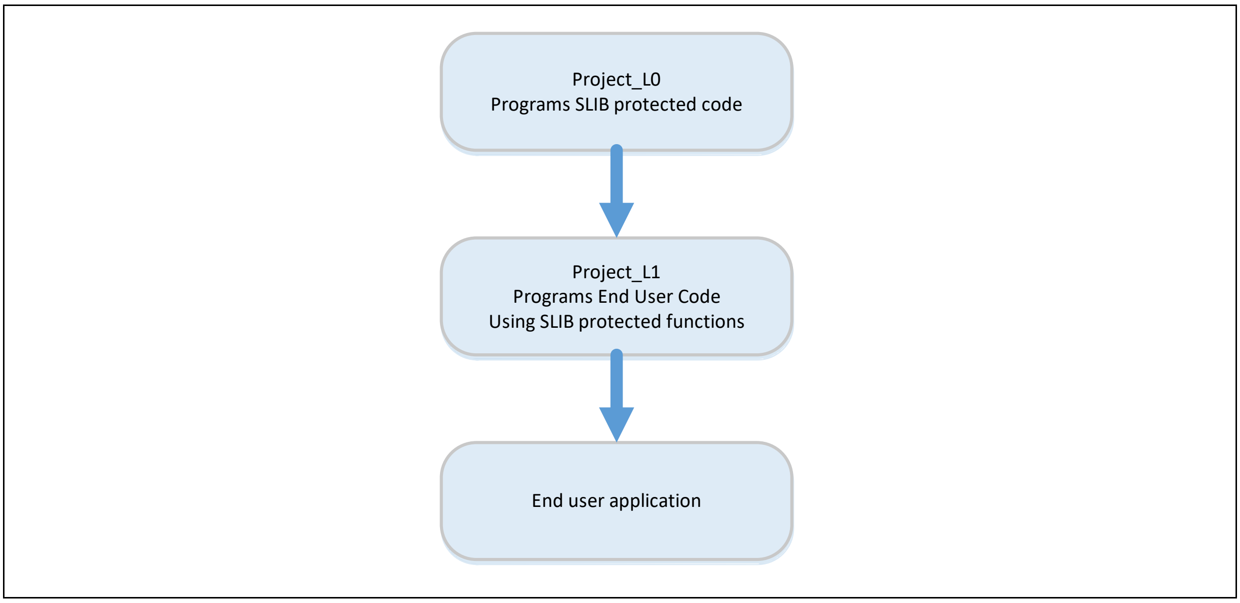
This application note provides two sample projects to demonstrate that software developers develop IP-Code for end-user applications.

- Project_L0: Solution provider develops algorithm and compiles to sLib
- Project_L1: End users apply algorithm

The algorithm completed in Project_L0 will be pre-downloaded and pre-burned to AT32A403A chip and then configured as sLib-protected. In addition, the following settings are available for the end-user applications.

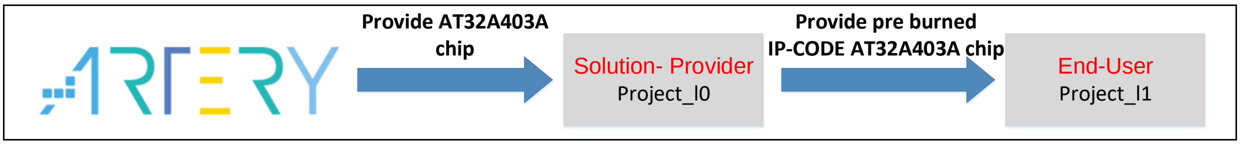
- Main Flash memory mapping, indicating the area occupied by sLib and the area where users can develop programs;
- Header file that contains algorithm function definitions, and functions that can be called by end users;
- Symbol definition file, which contains the actual address of each IP-Code function, so that these functions can be called properly by the end-user application.

Figure 6. Example of application process



Software solution providers can refer to the Project_L0 to develop algorithm code and refer to Project_L1 for end-user application.

Figure 7. Example of application

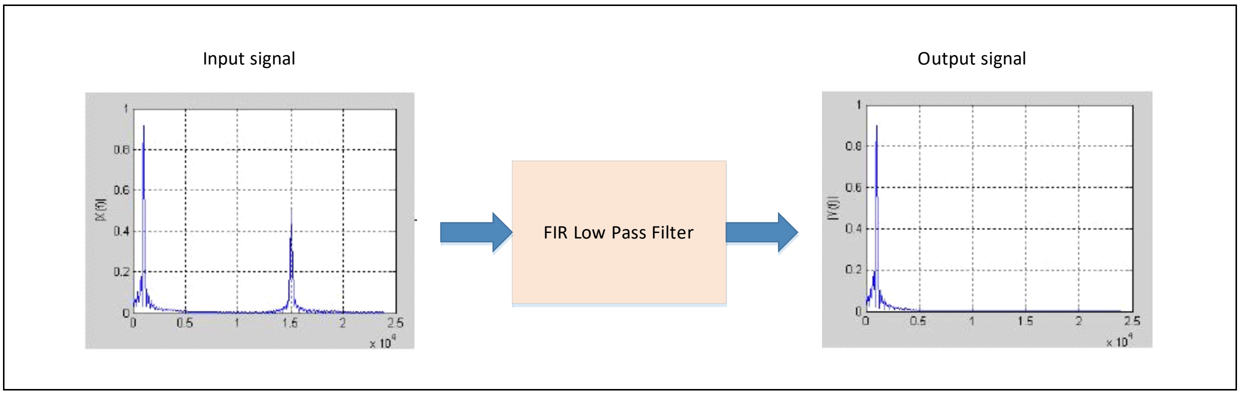


3.3 SLIB-protected code: FIR low-pass filter

This example uses FIR low-pass filter algorithm provided by CMSIS-DSP library as the sLib protected IP-Code. For details about FIR low-pass filter algorithm, refer to CMSIS-DSP relevant documents. This application note mainly introduces how to configure sLib to protect this algorithm and how it is called by the end-user program code.

The low-pass filter input signal in this example is a combination of two sine waves at frequencies of 1 KHz and 15 KHz, while the low-pass filter cut-off frequency is about 6 KHz. A 15 KHz signal is filtered through the low-pass filter and outputs 1 KHz sine wave. Figure 8 shows the FIR low-pass filter functions.

Figure 8. FIR low-pass filter



CMSIS DSP library functions and files to be used are:

- arm_fir_init_f32()

This function is used for initialization of filter, and it is included in “arm_fir_init_f32.c” file.

- arm_fir_f32()

=This function is the main part of filter algorithm, and it is included in “arm_fir_f32.c” file.

- FIR_lowpass_filter()

It is a FIR low-pass filter global function written by using the above two functions, and is included in “fir_filter.c” file.

- fir_coefficient.c

This C file contains coefficients (read-only constants) used by FIR filter functions, and these coefficients are placed in the data security library in the example.

In this example, embedded FPU and DSP are used for signal processing and floating-point operations to realize accurate computation and output correct signals.

3.4 Project_L0: Example for solution providers

The following projects are completed in this level:

- Compile the algorithm-related functions to execute-only code;
- Place the algorithm code to the main Flash memory sector2;
- Place the filter function coefficients to the main Flash memory sector 4;
- Execute FIR_lowpass_filter() in the main program to verify its correctness;
- If it is verified correct, configure sector 2/3 as the instruction security library and sector 4/5 as the data security library, by calling the slib_enable() function in the main program or by using Artery ICP Programmer (recommended);
- Generate the header file and symbol definition files that are used by end-user program to call low-pass filter functions.

3.4.1 Generate execute-only code

Each toolchain has specific setting options to prevent the compiler generating literal pools and branch table that can read data while executing instructions, such as “LDR Rn, [PC, #offset]”. Section 2.4 lists examples of literal pool and branch table.

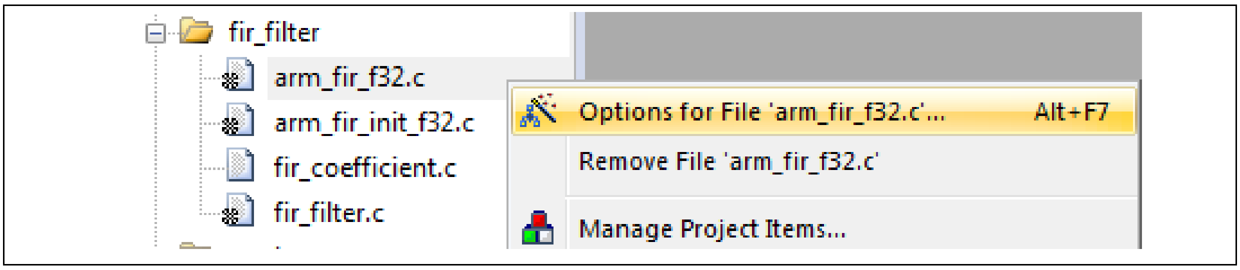
For example, Keil® µvision has Execute-only Code option, which can be set as follows:

Keil® µvision: Execute-only Code option

Configure as follows:

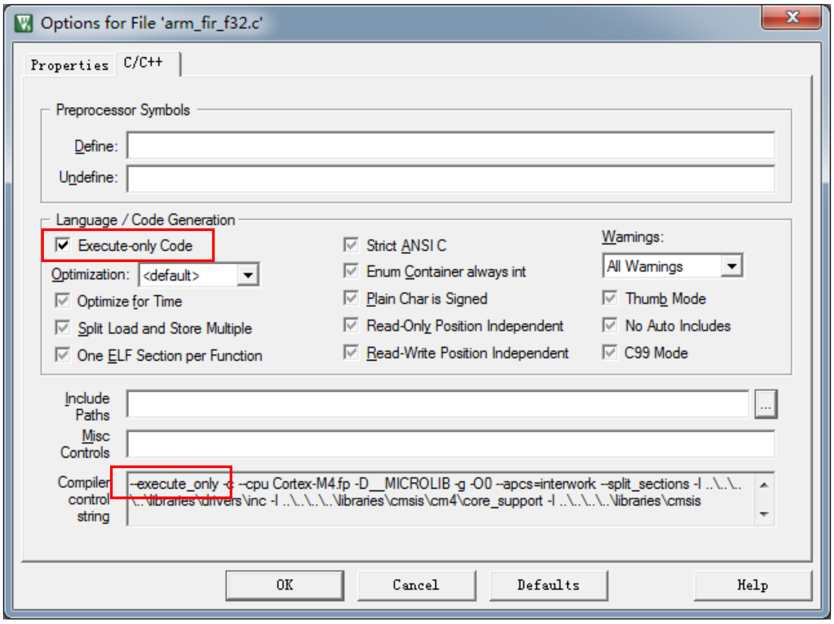
- Select C file group or individual C file (in this example, C files to be protected are placed “infir_filter”).
- Right click and select the corresponding files (for example, the *Option for File ‘arm_fir_f32.c’*), as shown in Figure 9.

Figure 9. Enter Option interface in Keil



- Tick “Execute-only Code” in the C/C++ and the “--execute_only” instruction is added to the compiler control string, as shown in Figure 10.

Figure 10. Tick Execute-only Code in Keil



Preprocessor Symbols

Define:

Undefine:

Language / Code Generation

☑ Execute-only Code

Optimization: <default>

☑ Optimize for Time

☑ Split Load and Store Multiple

☑ One ELF Section per Function

☑ Strict ANSI C

☑ Enum Container always int

☑ Plain Char is Signed

☑ Read-Only Position Independent

☑ Read-Write Position Independent

Warnings: All Warnings

☑ Thumb Mode

☑ No Auto Includes

☑ C99 Mode

Include Paths

Misc Controls

Compiler control string

```
--execute_only -c --cpu Cortex-M4.fp -D__MICROLIB -g -O0 --apcs=interwork --split_sections -I ..\..\..
\..\libraries\drivers\inc -I ..\..\..\..\libraries\cmsis\cm4\core_support -I ..\..\..\..\libraries\cmsis
```

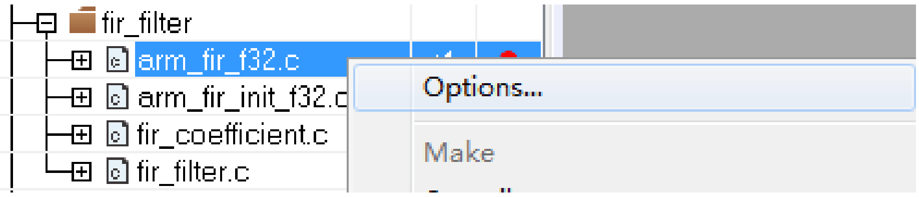
- The *rm_fir_f32.c*, *arm_fir_init_f32.c* and *fir_filter.c* are in the SLIB_INSTRUCTION area, and these files need to be configured as generating execute-only code.

IA: No data read in code memory

Configure as follows:

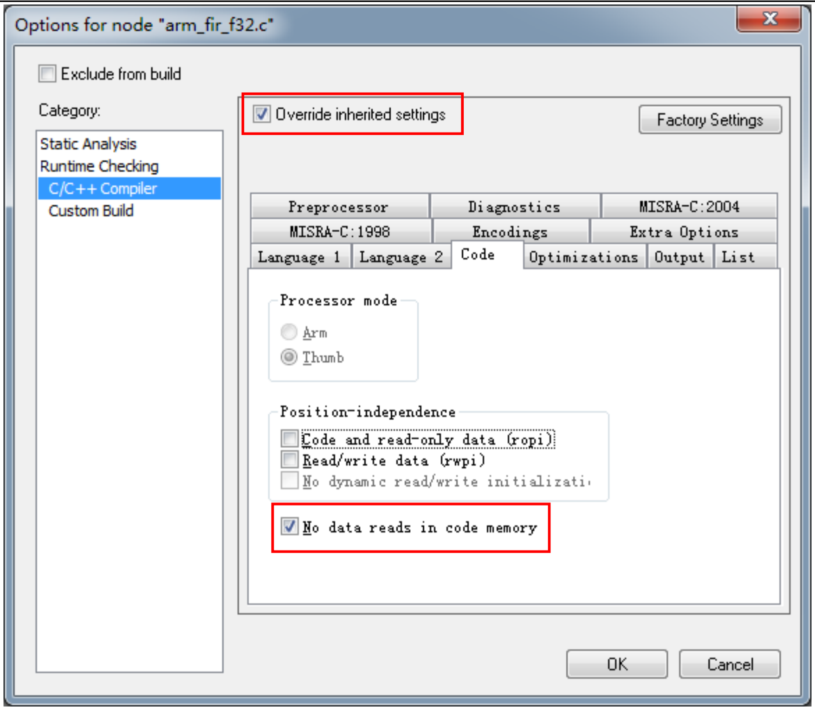
- Select the corresponding file in the *fir_filter* group, and right click to select Option.

Figure 11. Enter Option interface in IAR



- Enter "C/C++" interface and tick “Override inherited settings” and “No data read in code memory”, as shown in Figure 12.

Figure 12. Set C/C++ Compiler in IAR



Options for node "arm_fir_f32.c"

□ Exclude from build

Category:

Static Analysis
Runtime Checking
C/C++ Compiler
Custom Build

☑ Override inherited settings

Processor mode
○ Arm
● Thumb

Position-independence
□ Code and read-only data (ropi)
□ Read/write data (rwpi)
□ No dynamic read/write initializati

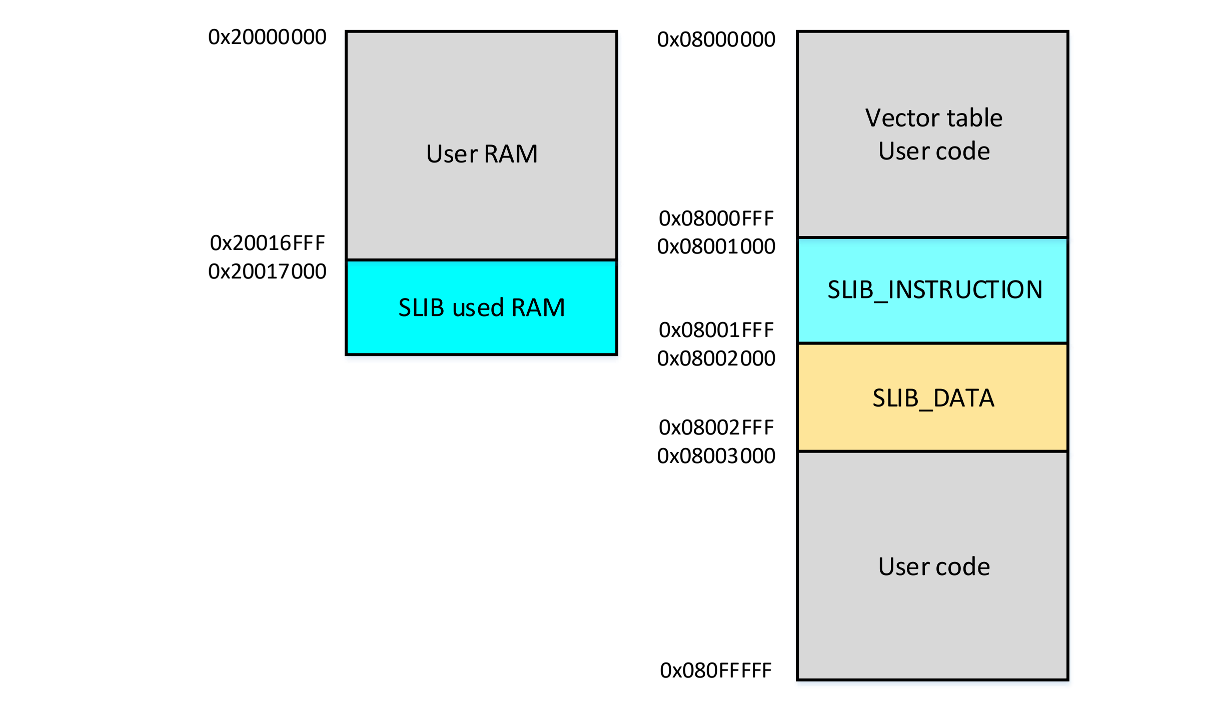
☑ No data reads in code memory

- The *arm_fir_f32.c*, *arm_fir_init_f32.c* and *fir_filter.c* are in the SLIB_INSTRUCTION area, and these files need to be configured as generating execute-only code.

3.4.2 Compile security library address

As aforementioned, the first sector (sector0) of the main Flash memory is used to store interrupt vector table. Therefore, the security library is set from sector2 in this example, with sector 2/3 as the instruction security library and sector 4/5 as the data security library. Figure 13 shows the main Flash memory mapping and RAM partition. The main purpose of RAM partitioning is to avoid the same RAM being used by sLib-protected code and end user code.

Figure 13. Main Flash memory mapping and RAM partition

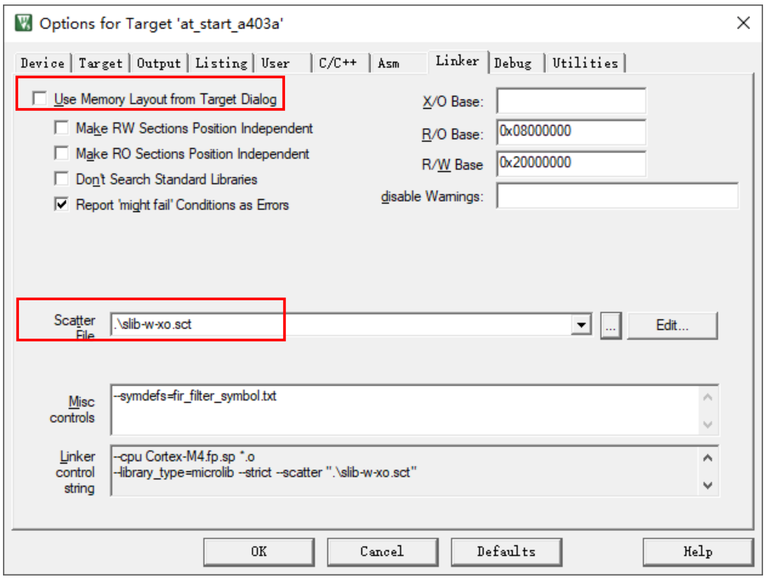


Keil® µvision: scatter file

Configure as follows:

- Click Project → Options for Target→Linker, untick "Use memory layout from Target Dialog" and click "Edit" to open and modify *slib-w-xo.sct* file, as shown below.

Figure 14. Set Linker in Keil



☐ Use Memory Layout from Target Dialog

☐ Make RW Sections Position Independent

☐ Make RO Sections Position Independent

☐ Don't Search Standard Libraries

☑ Report 'might fail' Conditions as Errors

X/O Base:

R/O Base: 0x08000000

R/W Base 0x20000000

disable Warnings:

Scatter File: .\slib-w-xo.sct

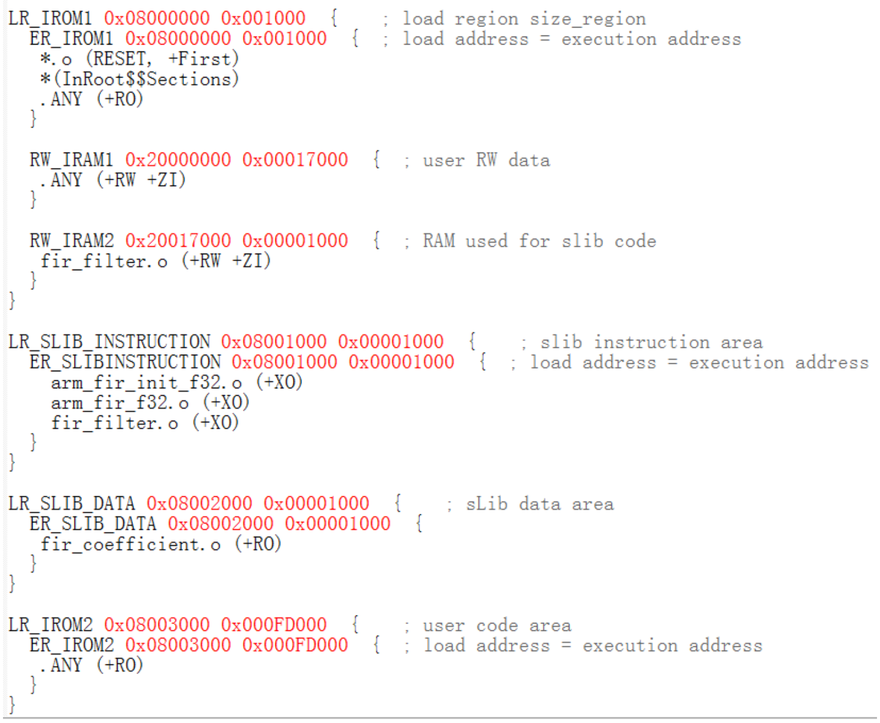
Misc controls: --symdefs=fir_filter_symbol.txt

Linker control string:

```
--cpu Cortex-M4.fp.sp *.o
--library_type=microlib --strict --scatter ".\slib-w-xo.sct"
```

- Open scatter file, load the object file of the code to be placed in SLIB_INSTRUCTION area to the "LR_SLIB_INSTRUCTION (a dedicated loading area that starts from sector2 and occupies two sectors) and modify the label to "execute-only (+XO)". Place the area occupied by the SLIB_Data to a dedicated loading area named "LR_SLIB_DATA" to avoid the compiler compiling other non-IP-Code functions to the SLIB area. The RW_IRAM2 assigns the region from 0x20017000 to 0x20017FFF to algorithm functions to avoid the same RAM region being used by end-user project, causing fault or error in program execution process.

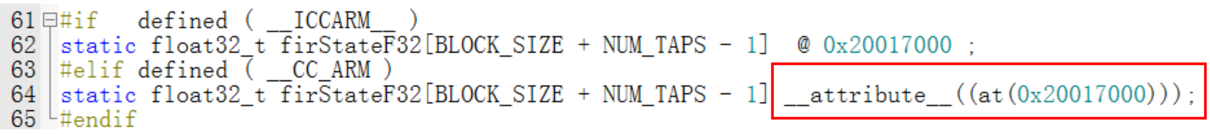
Figure 15. Modify scatter file in Keil



```
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 0x00017000  {  ; user RW data
   .ANY (+RW +ZI)
  }

  RW_IRAM2 0x20017000 0x00001000  {  ; RAM used for slib code
   fir_filter.o (+RW +ZI)
  }
}

LR_SLIB_INSTRUCTION 0x08001000 0x00001000  {    ; slib instruction area
  ER_SLIBINSTRUCTION 0x08001000 0x00001000  {  ; load address = execution address
    arm_fir_init_f32.o (+XO)
    arm_fir_f32.o (+XO)
    fir_filter.o (+XO)
  }
}

LR_SLIB_DATA 0x08002000 0x00001000  {    ; sLib data area
  ER_SLIB_DATA 0x08002000 0x00001000  {
   fir_coefficient.o (+RO)
  }
}

LR_IROM2 0x08003000 0x000FD000  {    ; user code area
  ER_IROM2 0x08003000 0x000FD000  {  ; load address = execution address
   .ANY (+RO)
  }
}
```

- In addition to modifying the scatter file, for the RAM used by IP-Code, users can also use the Keil “__attribute__((at(address)))” descriptor to load variables to 0x20017000, as shown in Figure 16.

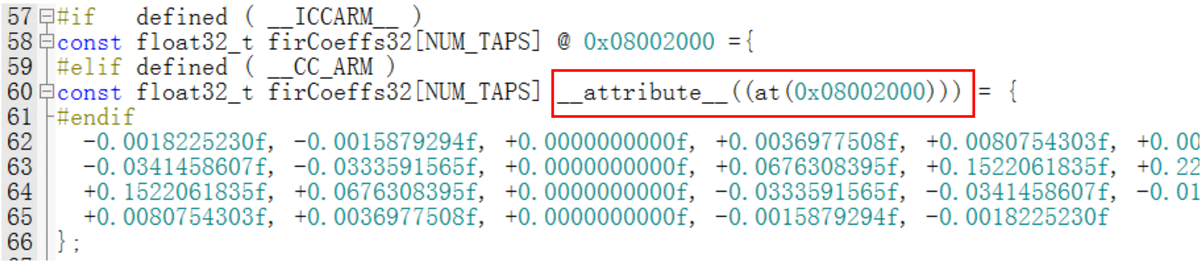
Figure 16. Modify SLIB RAM address in Keil



```
#if   defined ( __ICCARM__ )
static float32_t firStateF32[BLOCK_SIZE + NUM_TAPS - 1]  @ 0x20017000 ;
#elif defined ( __CC_ARM )
static float32_t firStateF32[BLOCK_SIZE + NUM_TAPS - 1] __attribute__((at(0x20017000)));
#endif
```

- The start address of data security library is sector 4 (0x08002000). To compile the constants used by FIR low-pass filter functions to this address, users can modify the scatter file as aforementioned, or use Keil “__attribute__((at(address)))” descriptor to load the constants to a fixed address, as shown in Figure 17.

Figure 17. Modify SLIB constant address in KEIL



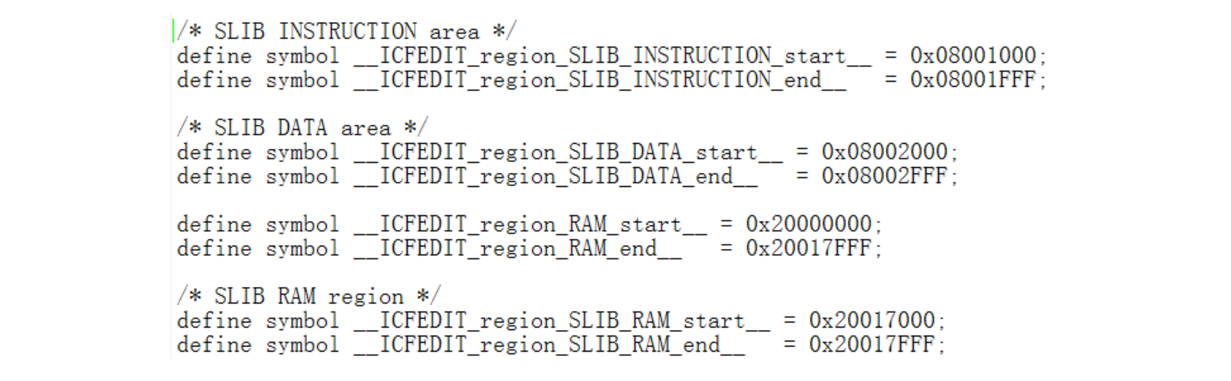
```
#if   defined ( __ICCARM__ )
const float32_t firCoeffs32[NUM_TAPS] @ 0x08002000 ={
#elif defined ( __CC_ARM )
const float32_t firCoeffs32[NUM_TAPS] __attribute__((at(0x08002000))) = {
#endif
  -0.0018225230f, -0.0015879294f, +0.0000000000f, +0.0036977508f, +0.0080754303f, +0.00
  -0.0341458607f, -0.0333591565f, +0.0000000000f, +0.0676308395f, +0.1522061835f, +0.22
  +0.1522061835f, +0.0676308395f, +0.0000000000f, -0.0333591565f, -0.0341458607f, -0.01
  +0.0080754303f, +0.0036977508f, +0.0000000000f, -0.0015879294f, -0.0018225230f
};
```

IAR: ICF file

Configure as follows:

- Open the *icf* file in "\project_l0\IAR_V8.2\", and add three SLIB loading areas as shown in Figure 18. The SLIB_RAM region reserves the corresponding RAM (0x20017000 to 0x20017FFF) for the algorithm functions.

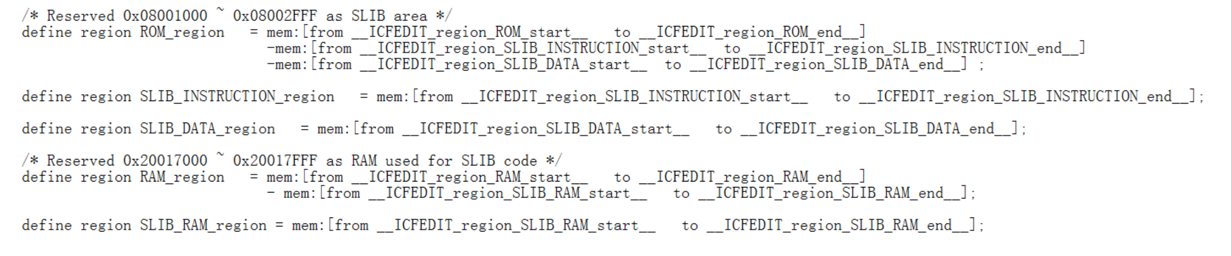
Figure 18. SLIB address definition in icf file



```
/* SLIB INSTRUCTION area */
define symbol __ICFEDIT_region_SLIB_INSTRUCTION_start__ = 0x08001000;
define symbol __ICFEDIT_region_SLIB_INSTRUCTION_end__   = 0x08001FFF;

/* SLIB DATA area */
define symbol __ICFEDIT_region_SLIB_DATA_start__ = 0x08002000;
define symbol __ICFEDIT_region_SLIB_DATA_end__   = 0x08002FFF;

define symbol __ICFEDIT_region_RAM_start__ = 0x20000000;
define symbol __ICFEDIT_region_RAM_end__   = 0x20017FFF;

/* SLIB RAM region */
define symbol __ICFEDIT_region_SLIB_RAM_start__ = 0x20017000;
define symbol __ICFEDIT_region_SLIB_RAM_end__   = 0x20017FFF;
```

- In the *icf* file, the area occupied by SLIB is reserved to avoid the compiler compiling other non-IP-Code functions to the SLIB area, and the RAM region used by IP-Code is also reserved.

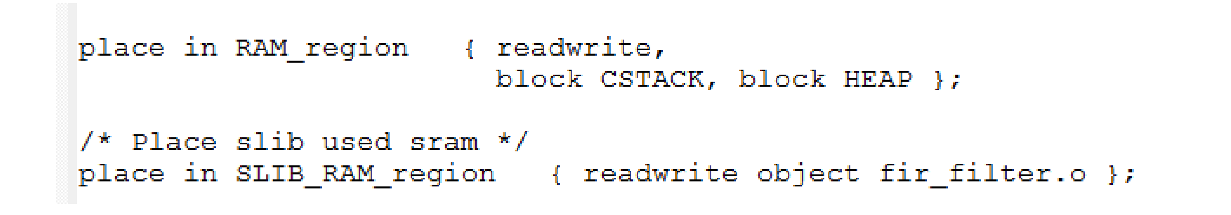
Figure 19. Address assignment in icf file



```
/* 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_INSTRUCTION_start__   to __ICFEDIT_region_SLIB_INSTRUCTION_end__]
                            -mem:[from __ICFEDIT_region_SLIB_DATA_start__  to __ICFEDIT_region_SLIB_DATA_end__] ;

define region SLIB_INSTRUCTION_region   = mem:[from __ICFEDIT_region_SLIB_INSTRUCTION_start__   to __ICFEDIT_region_SLIB_INSTRUCTION_end__];

define region SLIB_DATA_region   = mem:[from __ICFEDIT_region_SLIB_DATA_start__   to __ICFEDIT_region_SLIB_DATA_end__];

/* Reserved 0x20017000 ~ 0x20017FFF 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__];
```

- For the RAM used by IP-Code, users can use the IAR @descriptor to load variables to a fixed (0x20017000) or modify the *icf* file, as shown in Figure 20.

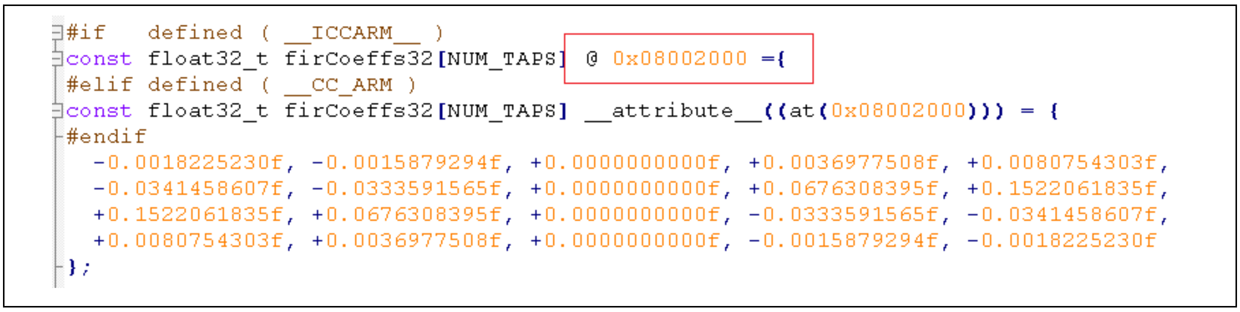
Figure 20. Modify IP-Code RAM in icf file



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

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

- The start address of data security library is sector 4 (0x08002000). To compile constants used by FIR low-pass filter functions to this address, users can modify the *icf* file as mentioned above or use the IAR @ descriptor to place these constants to a fixed address, as shown in Figure 21.

Figure 21. Modify SLIB used constant address in IAR



```
#if   defined ( __ICCARM__ )
const float32_t firCoeffs32[NUM_TAPS] @ 0x08002000 ={
#elif defined ( __CC_ARM )
const float32_t firCoeffs32[NUM_TAPS] __attribute__((at(0x08002000))) = {
#endif
  -0.0018225230f, -0.0015879294f, +0.0000000000f, +0.0036977508f, +0.0080754303f,
  -0.0341458607f, -0.0333591565f, +0.0000000000f, +0.0676308395f, +0.1522061835f,
  +0.1522061835f, +0.0676308395f, +0.0000000000f, -0.0333591565f, -0.0341458607f,
  +0.0080754303f, +0.0036977508f, +0.0000000000f, -0.0015879294f, -0.0018225230f
};
```

3.4.3 Enable sLib protection

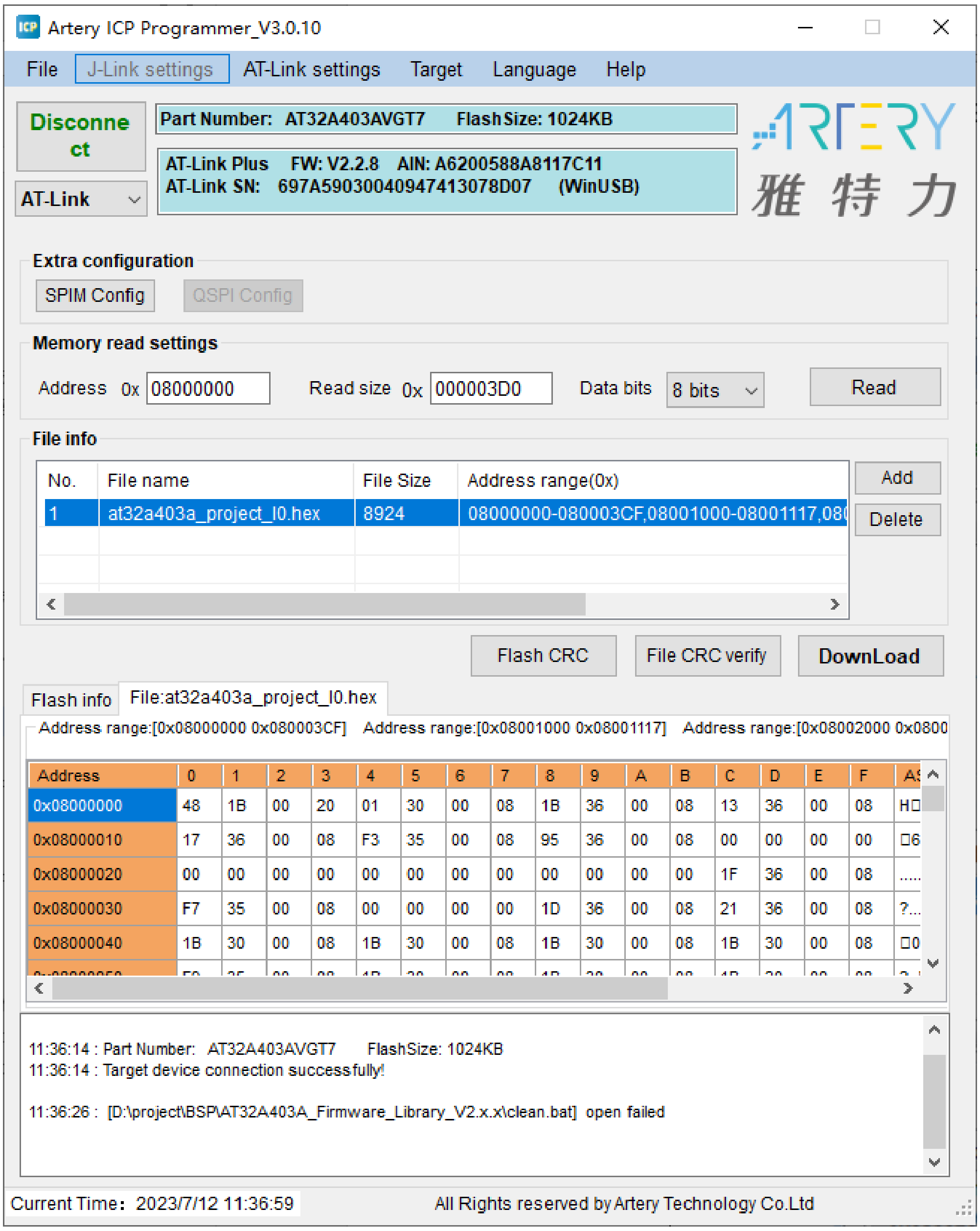
There are two methods to enable sLib protection:

(1) User Artery ICP Programmer (recommended)

It is recommended to use Artery ICP Programmer as follows:

- Connect AT-Link emulator to AT-START-A403A board and then power on;
- Open ICP Programmer, select AT-Link for connection, and add the HEX or BIN file generated by Project_L0, as shown in Figure 22.

Figure 22. Configure ICP Programmer



Part Number: AT32A403AVGT7 FlashSize: 1024KB

AT-Link Plus FW: V2.2.8 AIN: A6200588A8117C11
AT-Link SN: 697A59030040947413078D07 (WinUSB)

Address 0x 08000000 Read size 0x 000003D0 Data bits 8 bits

No.	File name	File Size	Address range(0x)
1	at32a403a_project_l0.hex	8924	08000000-080003CF,08001000-08001117,08

File:at32a403a_project_l0.hex

Address range:[0x08000000 0x080003CF] Address range:[0x08001000 0x08001117] Address range:[0x08002000 0x0800

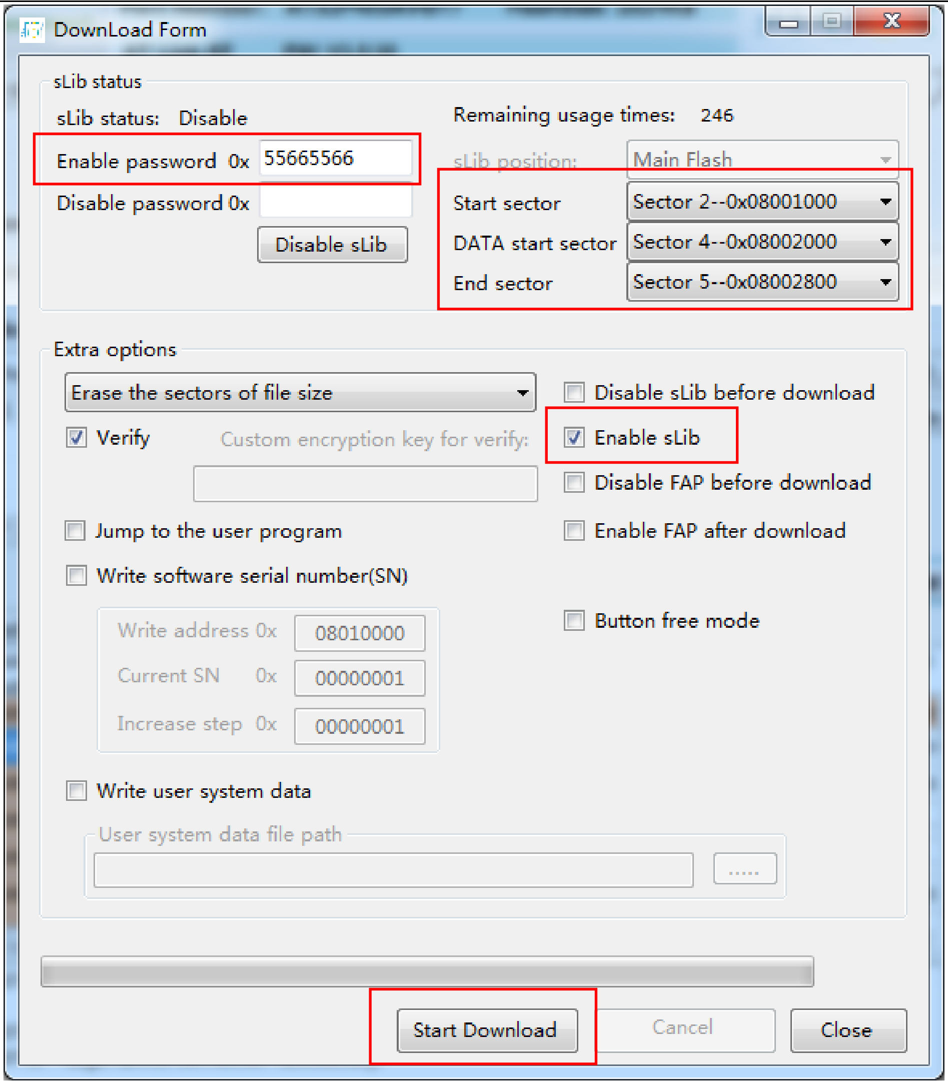
Address	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	AS
0x08000000	48	1B	00	20	01	30	00	08	1B	36	00	08	13	36	00	08	H□
0x08000010	17	36	00	08	F3	35	00	08	95	36	00	08	00	00	00	00	□6
0x08000020	00	00	00	00	00	00	00	00	00	00	00	00	1F	36	00	08	.....
0x08000030	F7	35	00	08	00	00	00	00	1D	36	00	08	21	36	00	08	?...
0x08000040	1B	30	00	08	1B	30	00	08	1B	30	00	08	1B	30	00	08	□0

```
11:36:14 : Part Number:  AT32A403AVGT7      FlashSize: 1024KB
11:36:14 : Target device connection successfully!

11:36:26 :  [D:\project\BSP\AT32A403A_Firmware_Library_V2.x.x\clean.bat]  open failed
```

Current Time: 2023/7/12 11:36:59 All Rights reserved by Artery Technology Co.Ltd

- Click “Download” and the “Download Form” pops up, which shows sLib status and relevant parameters. Set sector 2 as the start sector, sector 4 as the data start sector, and sector 5 as the end sector. Set the enable password as “0x55665566” (user-defined) and tick “Enable sLib”; then click “Start Download” to complete programming and enable sLib successfully, as shown in Figure 23.

Figure 23. Set parameters in Download Form



DownLoad Form

sLib status

sLib status: Disable

Remaining usage times: 246

Enable password 0x 55665566

Disable password 0x

sLib position: Main Flash

Start sector: Sector 2--0x08001000

DATA start sector: Sector 4--0x08002000

End sector: Sector 5--0x08002800

Extra options

Erase the sectors of file size

☑ Verify

Custom encryption key for verify:

□ Jump to the user program

□ Write software serial number(SN)

Write address 0x 08010000

Current SN 0x 00000001

Increase step 0x 00000001

□ Write user system data

User system data file path

□ Disable sLib before download

☑ Enable sLib

□ Disable FAP before download

□ Enable FAP after download

□ Button free mode

For details about ICP Programmer, refer to ICP Programmer User Manual.

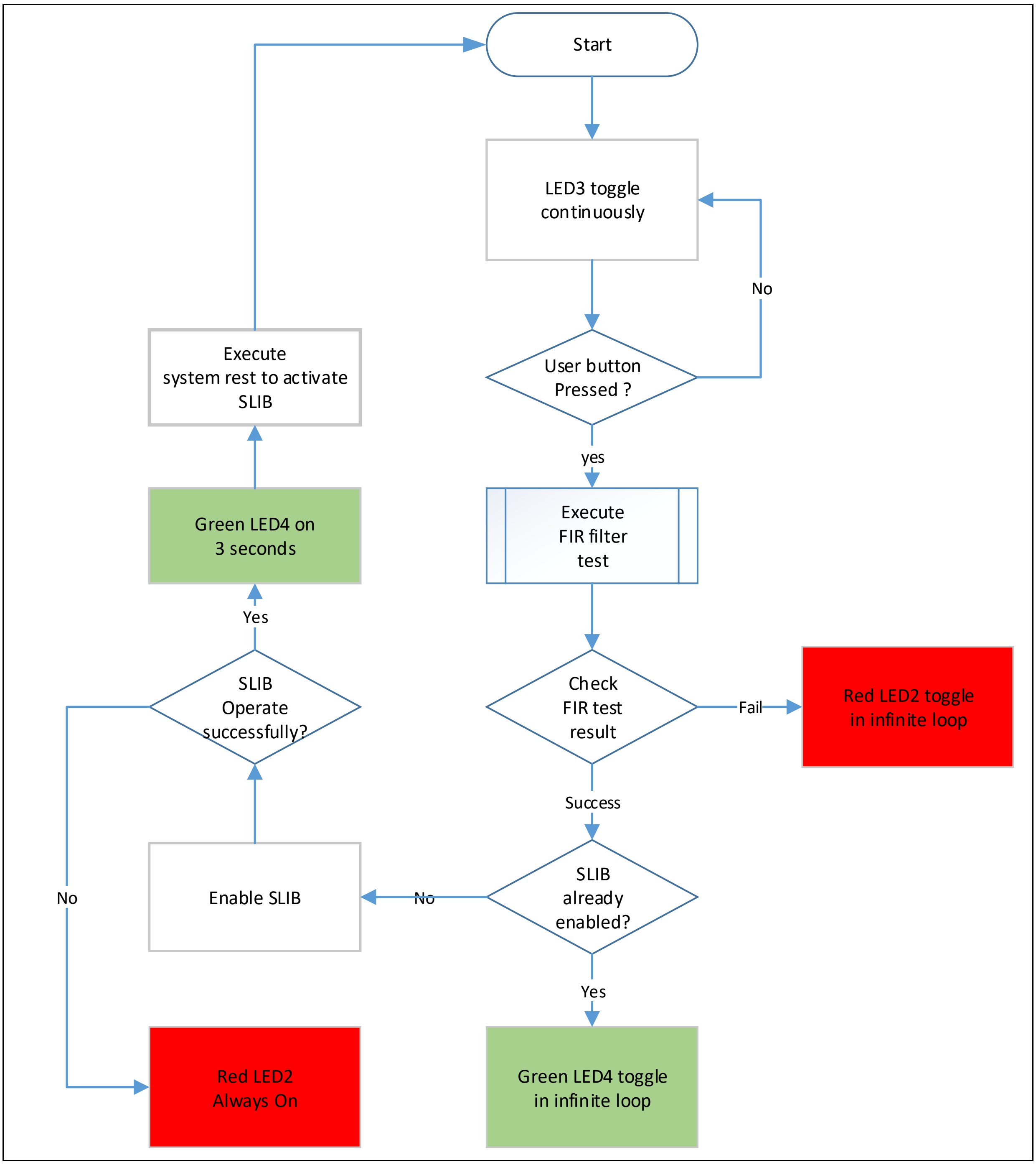
(2) Use *slib_enable()* in main.c

After the slib_enable() function is verified correct by low-pass filter function and then executed, the sLib protection can be enabled. To execute this function, enable the “#define USE_SLIB_FUNCTION” in main.c.

3.4.4 Project_L0 execution process

In this example, FIR low-pass filter calculates the input signal (testInput_f32_1kHz_15kHz) mixed with 1 KHz and 15 KHz sine waves, and the output 1 KHz sine wave data is saved in testOutput, which will be compared with the data calculated by MATLAB and saved in refOutput. If the error value is smaller than the expected (SNR larger than the preset threshold), the on-board green LED blinks; otherwise, the on-board red LED blinks. Figure 24 shows the Project_L0 execution process.

Figure 24. Project_L0 execution process



Go through the following steps to execute this example program:

(1) Use Keil® µvision to open the Project_L0 under "\utilities\at32a403a_slib_demo\project_l0\mdk_v5\", and then compile.

(2) Before downloading the code, check whether the chip on AT-START-A403A board is sLib-protected or write/read-protected (FAP/EPP). If it is protected, use ICP programmer to disable protection and then download the code.

(3) After successful download, start to execute the code, and the on-board LED3 keeps blinking rapidly.

(4) Press the on-board USER button to perform low-pass filter computation.

(5) Compare the computation result. If it is correct, the green LED4 keeps blinking; otherwise, the

read LED2 keeps blinking.

(6) After obtaining the correct result, if the USE_SLIB_FUNCTION in main.c is defined and the SLIB is not enabled, the *slib_enable()* function will be executed to set SLIB. If SLIB setting fails, the red LED2 will be always ON; if SLIB setting succeeds, the green LED4 will be ON for about three seconds and then perform system reset to enable SLIB; then, the program jumps to step (3).

3.4.5 Generate header file and symbol definition file

The header file and symbol definition file are used when the Project_L1 calls FIR low-pass filter functions, which is the *fir_filter.h* file in main.c in this example.

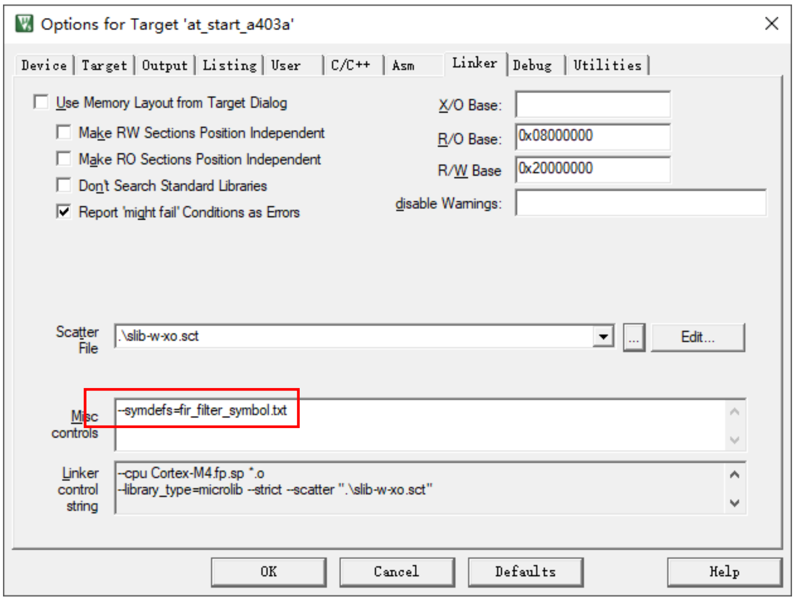
The generation of symbol definition file is related to the specific toolchain being used.

Use Keil® μvision to generate symbol definition file

Configure as follows:

- Enter Options for Target → Linker interface.
- Add "--symdefs=fir_filter_symbol.txt" command in the "Misc controls", as shown in Figure 25.

Figure 25. Set Misc controls in Keil



Options for Target 'at_start_a403a'

□ Use Memory Layout from Target Dialog

□ Make RW Sections Position Independent

□ Make RO Sections Position Independent

□ Don't Search Standard Libraries

☑ Report 'might fail' Conditions as Errors

X/O Base:

R/O Base: 0x08000000

R/W Base 0x20000000

disable Warnings:

Scatter File: .\slib-w-xo.sct

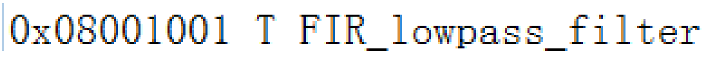
Misc controls: --symdefs=fir_filter_symbol.txt

Linker control string:

```
--cpu Cortex-M4.fp.sp *.o
--library_type=microlib --strict --scatter ".\slib-w-xo.sct"
```

- Compile the whole project, and then a "fir_filter_symbol.txt" symbol definition file is generated under "project_l0\mdk_v5\Objects".
- This symbol definition file contains all symbol definitions of the project, and it needs to be modified to only remain the definitions of low-pass filter functions to be called by end users. The modified *fir_filter_symbol.txt* is shown in Figure 26.

Figure 26. Modified fir_filter_symbol.txt



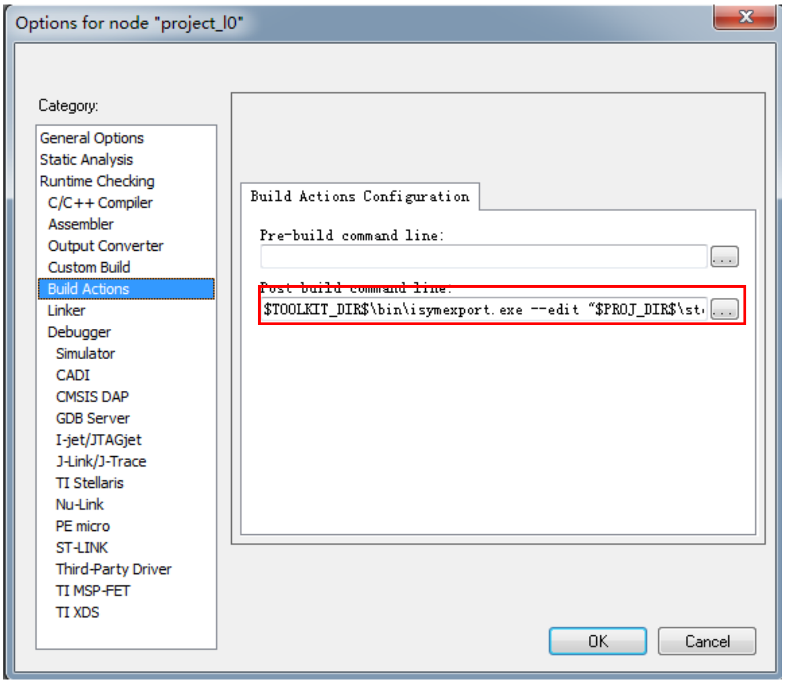
```
0x08001001 T FIR_lowpass_filter
```

Use IAR to generate symbol definition file

Configure as follows:

- Select Project→Option→Build Actions

Figure 27. Set Build Actions in IAR



Pre-build command line:

Post-build command line:

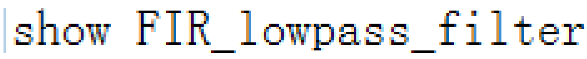
```
$TOOLKIT_DIR$\bin\isymexport.exe --edit "$PROJ_DIR$\st
```

- Input the following commands to the Post-build command line:

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

- The *fir_filter_symbol.o* is the symbol definition file to be generated, and the *steering_file.txt* is saved under "project_l0\iar_v8.2", which is used to select function symbols to be generated. Users can manually edit the contents called by sLib. As shown in Figure 28, the "show" is the command for function selection.

Figure 28. Edit steering_file.txt content



```
show FIR_lowpass_filter
```

3.5 Project_L1: example for end users

Project_L1 uses the FIR low-pass filter function that is debugged in Project_L0, programmed to AT32A403A MCU main Flash memory and configured as SLIB-protected. According to the header file, symbol definition file and the main Flash memory mapping of Project_L0, end users can complete the followings for Project_L1:

- Create an application project;
- Add the header file and symbol definition file provided by Project_L0 to the project;
- Call the FIR low-pass filter functions;

- Develop and debug user’s program.

Note:

Project_L1 must use the same toolchain and the same version of the compiler as those of Project_L0; otherwise, incompatibility problem may occur and the code provided by Project_L0 cannot be used properly. For example, Project_L0 uses Keil® µvision V5.36.0.0; therefore, Project_L1 need to use the same version

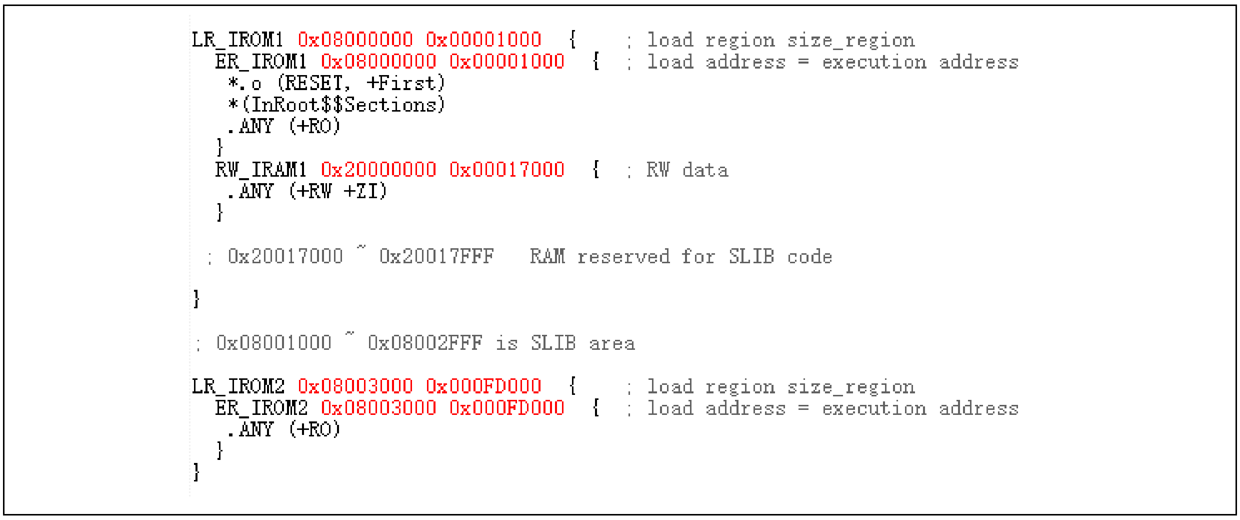
3.5.1 Create user application project

The security library enabled in Project_L0 occupies some specific main Flash memory sectors; therefore, the address for Project_L1 code storage should be compiled according to the main Flash memory mapping of Project_L0. In the main Flash memory, sector 2 to sector 5 are occupied by security library, which should be isolated by using the linker control file to avoid code being compiled to this region.

Keil®µvision: scatter file

Refer to the *end_user_code.sct* under “project_l1\mdk_v5\”, and divide the main Flash memory into two regions, and the middle part is the sLib-protected area. In addition, the area behind 0x20017000 in RAM should be reserved, as shown in Figure 29.

Figure 29. Modified scatter file

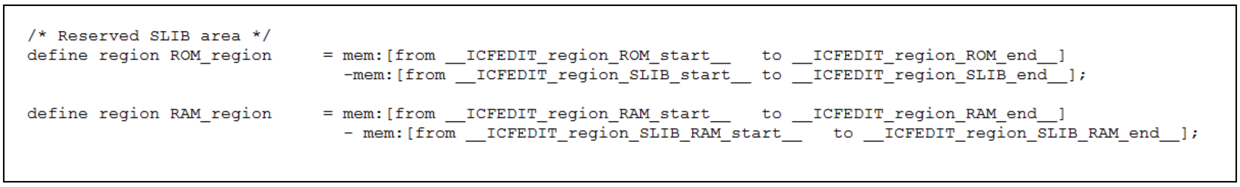


```
LR_IROM1 0x08000000 0x00001000  {    ; load region size_region
  ER_IROM1 0x08000000 0x00001000  {  ; load address = execution address
   *.o (RESET, +First)
   *(InRoot$$Sections)
   .ANY (+RO)
  }
  RW_IRAM1 0x20000000 0x00017000  {  ; RW data
   .ANY (+RW +ZI)
  }

 ; 0x20017000 ~ 0x20017FFF   RAM reserved for SLIB code

}

; 0x08001000 ~ 0x08002FFF is SLIB area

LR_IROM2 0x08003000 0x000FD000  {    ; load region size_region
  ER_IROM2 0x08003000 0x000FD000  {  ; load address = execution address
   .ANY (+RO)
  }
}
```

IAR: ICF file

Refer to the *enduser.icf* file under “project_l1\iar_V8.2\”, as shown in Figure 30.

Figure 30. Modified icf file



```
/* Reserved SLIB area */
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__];
```

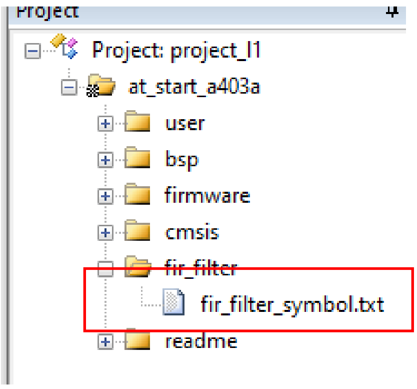
3.5.2 Add symbol definition file to project

The symbol definition file *fir_filter_symbol.txt* generated in Project_L0 must be added to Project_L1, so that it can be correctly compiled and linked to the sLib-protected area code.

Add symbol definition file in Keil® µvision

Add *fir_filter_symbol.txt* to the project, as shown in Figure 31.

Figure 31. Add symbol definition file Keil

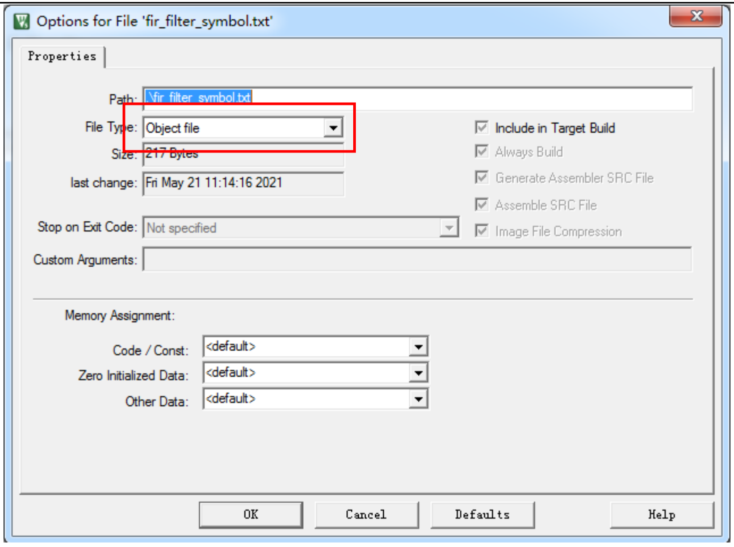


Project

- Project: project_l1
  - at_start_a403a
    - user
    - bsp
    - firmware
    - cmsis
    - fir_filter
      - fir_filter_symbol.txt
    - readme

Add this file to fir_filter, and then modify its file type from “text” to “Object”, as shown in Figure 32.

Figure 32. Modify symbol definition file as “Object file”



Options for File 'fir_filter_symbol.txt'

Properties

Path: .\fir_filter_symbol.txt

File Type: Object file

Size: 217 Bytes

last change: Fri May 21 11:14:16 2021

Stop on Exit Code: Not specified

Custom Arguments:

☑ Include in Target Build

☑ Always Build

☑ Generate Assembler SRC File

☑ Assemble SRC File

☑ Image File Compression

Memory Assignment:

Code / Const: <default>

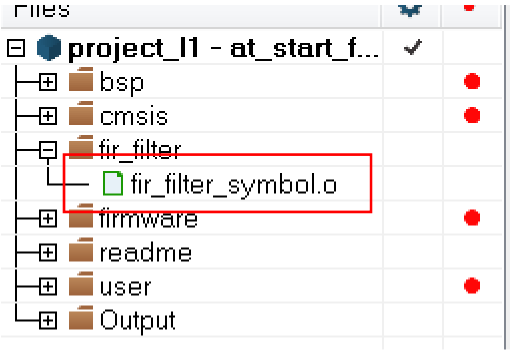
Zero Initialized Data: <default>

Other Data: <default>

Add symbol definition file in IAR

Add the fir_filter_symbol.o (Object) to fir_filter, as shown in Figure 33.

Figure 33. Add symbol definition file in IAR



3.5.3 Call functions in sLib-protected area

After the *filter.h* header file is referred in main.c and the symbol definition file is added to the project, the low-pass filter function in the protection area can be called, as shown below:

FIR_lowpass_filter(inputF32, outputF32, TEST_LENGTH_SAMPLES);

Where:

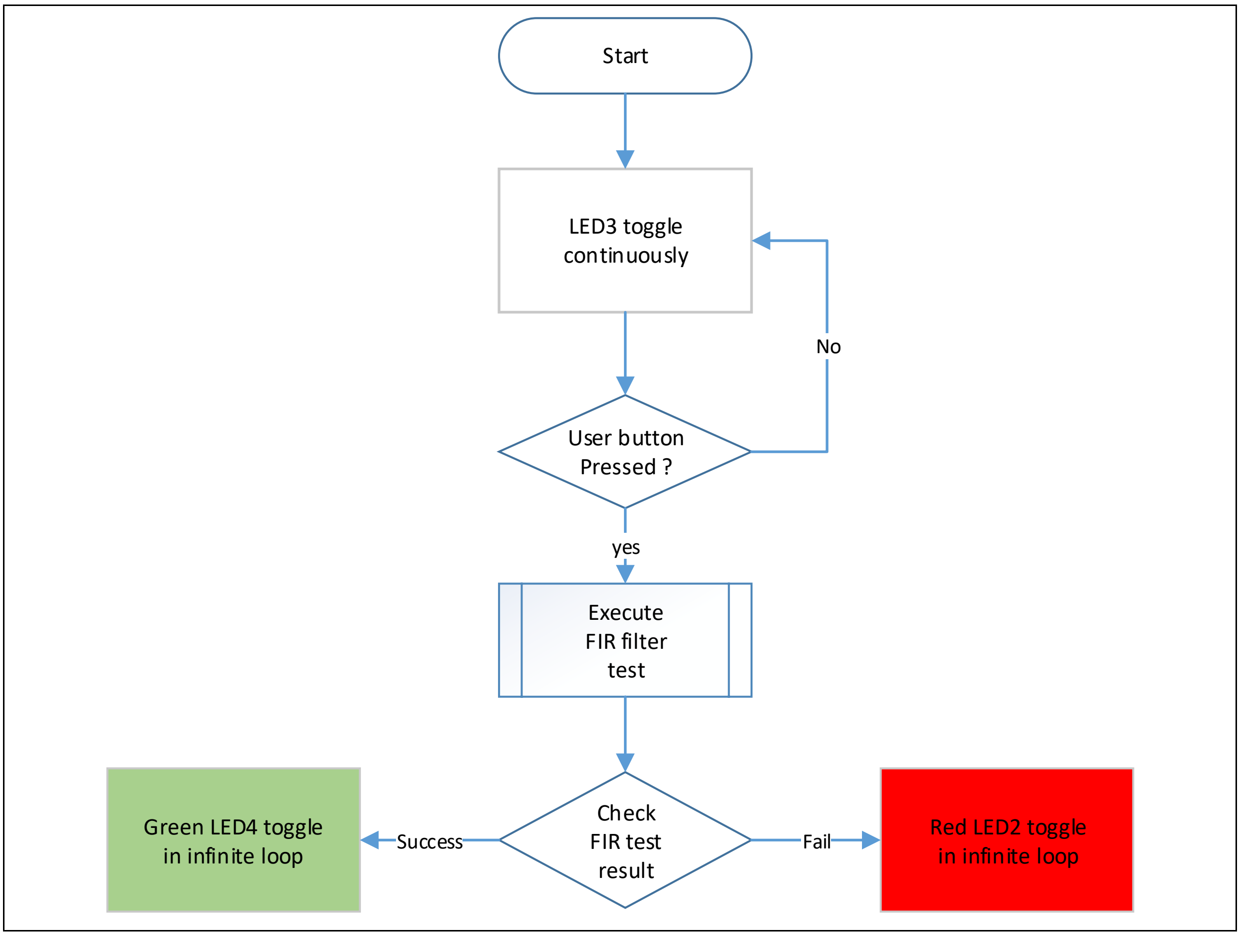
- *inputF32*: pointer to input signal data table
- *outputF32*: pointer to output signal data table
- *TEST_LENGTH_SAMPLES*: the number of signal samples to be processed

3.5.4 Project_L1 execution process

Figure 34 shows the execution process of Project_L1:

- Start execution and LED3 keeps blinking;
- Press the USER button on AT-START board, and the FIR_lowpass_filter() starts computing;
- If the result is correct, the green LED4 will keep blinking; otherwise, the red LED2 will keep blinking.

Figure 34. Project_L1 execution process

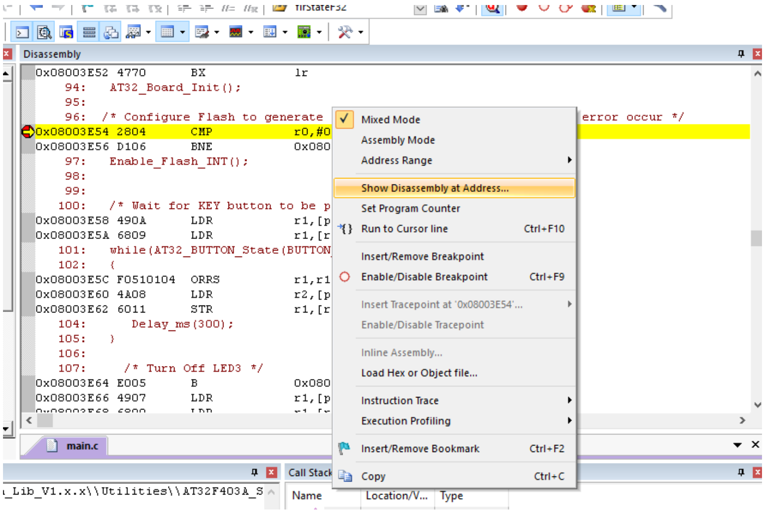


3.5.5 SLIB protection in debug mode

Development tools are used by end users to debug codes when developing applications. This section takes Keil® µvision as an example to introduce how to protect codes in the SLIB-protected area from being read as data in debug mode.

- Open Project_L1 project and compile;
- Click "Start/Stop Debug Session" to enter debug mode;
- Right click in the "Disassembly" interface and select "Show Disassembly at Address", as shown in Figure 35.

Figure 35. Set Show Disassembly at Address

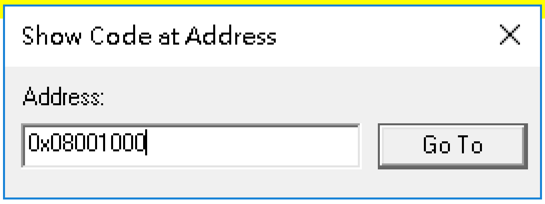


```
Disassembly
0x08003E52 4770      BX        lr
    94:   AT32_Board_Init();
    95:
    96:  /* Configure Flash to generate                     error occur */
0x08003E54 2804      CMP       r0,#0
0x08003E56 D106      BNE       0x080
    97:   Enable_Flash_INT();
    98:
    99:
   100:   /* Wait for KEY button to be p
0x08003E58 490A      LDR       r1,[p
0x08003E5A 6809      LDR       r1,[r
   101:   while(AT32_BUTTON_State(BUTTON
   102:   {
0x08003E5C F0510104  ORRS      r1,r1
0x08003E60 4A08      LDR       r2,[p
0x08003E62 6011      STR       r1,[r
   104:      Delay_ms(300);
   105:   }
   106:
   107:     /* Turn Off LED3 */
0x08003E64 E005      B         0x080
0x08003E66 4907      LDR       r1,[p
```

main.c
Call Stack
Name
Location/V...
Type
_Lib_V1.x.x\\Utilities\\AT32F403A_S
Mixed Mode
Assembly Mode
Address Range
Show Disassembly at Address...
Set Program Counter
Run to Cursor line Ctrl+F10
Insert/Remove Breakpoint
Enable/Disable Breakpoint Ctrl+F9
Insert Tracepoint at '0x08003E54'...
Enable/Disable Tracepoint
Inline Assembly...
Load Hex or Object file...
Instruction Trace
Execution Profiling
Insert/Remove Bookmark Ctrl+F2
Copy Ctrl+C

- Enter the address “0x08001000” of SLIB_INSTRUCTION start sector 2.

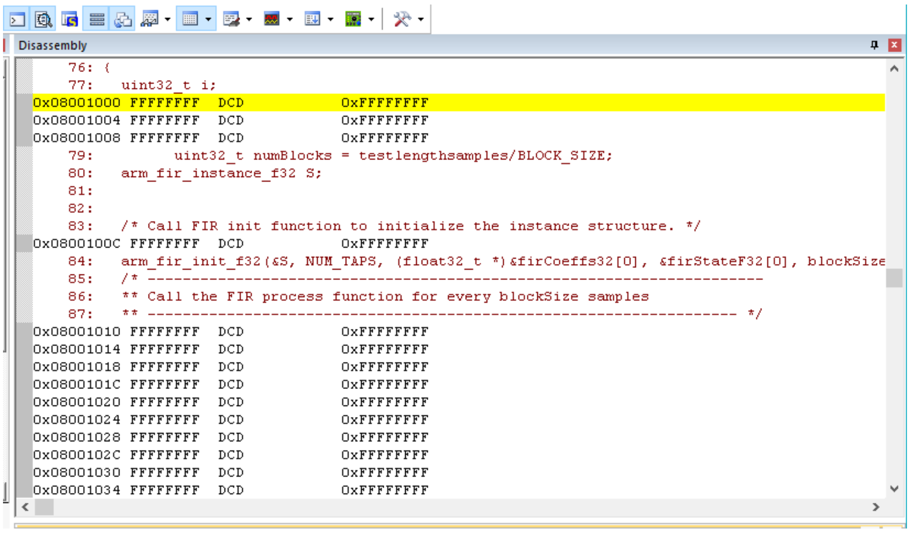
Figure 36. Set Show Code at Address



Address: 0x08001000

- As shown in Figure 37, codes from 0x08001000 are all 0xFFFFFFFF.

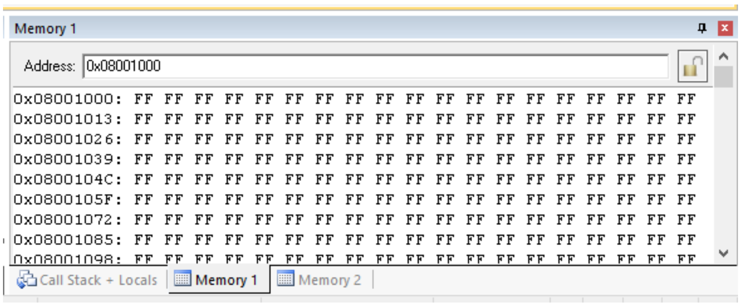
Figure 37. View codes



```
Disassembly
    76: {
    77:   uint32_t i;
0x08001000 FFFFFFFF  DCD          0xFFFFFFFF
0x08001004 FFFFFFFF  DCD          0xFFFFFFFF
0x08001008 FFFFFFFF  DCD          0xFFFFFFFF
    79:         uint32_t numBlocks = testlengthsamples/BLOCK_SIZE;
    80:   arm_fir_instance_f32 S;
    81:
    82:
    83:   /* Call FIR init function to initialize the instance structure. */
0x0800100C FFFFFFFF  DCD          0xFFFFFFFF
    84:   arm_fir_init_f32(&S, NUM_TAPS, (float32_t *)&firCoeffs32[0], &firStateF32[0], blockSize
    85:   /* ------------------------------------------------------------------
    86:   ** Call the FIR process function for every blockSize samples
    87:   ** ----------------------------------------------------------------- */
0x08001010 FFFFFFFF  DCD          0xFFFFFFFF
0x08001014 FFFFFFFF  DCD          0xFFFFFFFF
0x08001018 FFFFFFFF  DCD          0xFFFFFFFF
0x0800101C FFFFFFFF  DCD          0xFFFFFFFF
0x08001020 FFFFFFFF  DCD          0xFFFFFFFF
0x08001024 FFFFFFFF  DCD          0xFFFFFFFF
0x08001028 FFFFFFFF  DCD          0xFFFFFFFF
0x0800102C FFFFFFFF  DCD          0xFFFFFFFF
0x08001030 FFFFFFFF  DCD          0xFFFFFFFF
0x08001034 FFFFFFFF  DCD          0xFFFFFFFF
```

- Similarly, enter address “0x08001000” in “Memory”, and codes are all 0xFF, as shown in Figure 38.

Figure 38. View codes in Memory

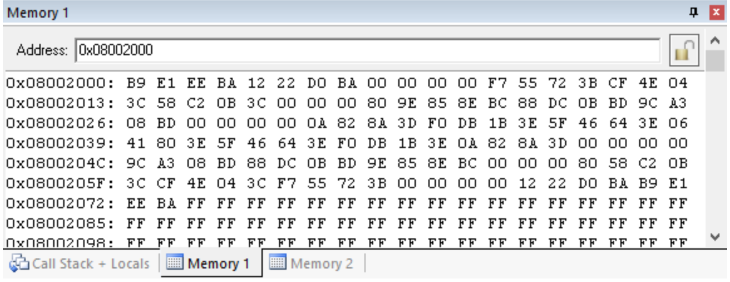


Address: 0x08001000

```
0x08001000: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x08001013: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x08001026: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x08001039: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x0800104C: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x0800105F: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x08001072: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x08001085: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x08001098: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

- In the "Memory" window, enter the address "0x08002000" of the SLIB_DATA start sector 4. This region is allowed to be read through D-Code bus, so that original values can be found, as shown in Figure 39.

Figure 39. View SLIB_DATA start sector in Memory

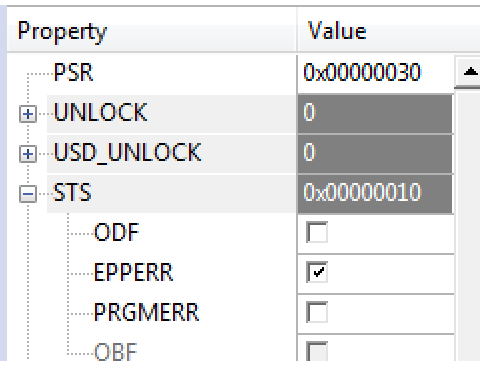


Address: 0x08002000

```
0x08002000: B9 E1 EE BA 12 22 D0 BA 00 00 00 00 F7 55 72 3B CF 4E 04
0x08002013: 3C 58 C2 0B 3C 00 00 00 80 9E 85 8E BC 88 DC 0B BD 9C A3
0x08002026: 08 BD 00 00 00 00 0A 82 8A 3D F0 DB 1B 3E 5F 46 64 3E 06
0x08002039: 41 80 3E 5F 46 64 3E F0 DB 1B 3E 0A 82 8A 3D 00 00 00 00
0x0800204C: 9C A3 08 BD 88 DC 0B BD 9E 85 8E BC 00 00 00 80 58 C2 0B
0x0800205F: 3C CF 4E 04 3C F7 55 72 3B 00 00 00 00 12 22 D0 BA B9 E1
0x08002072: EE BA FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x08002085: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x08002098: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

- Double click to modify the value of 0x08002000 in the code, and a warning message will be issued by setting EPPERR=1 in the FLASH_STS register, indicating the protection is enabled.

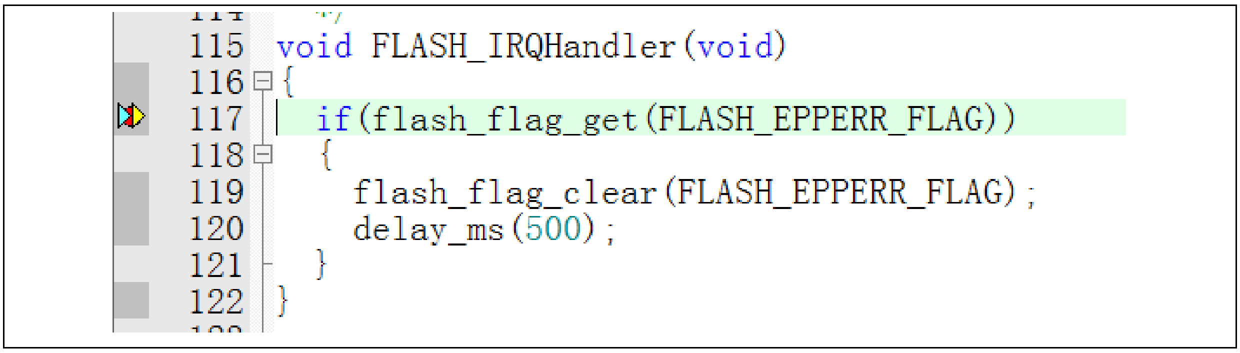
Figure 40. SLIB write test



Property	Value
PSR	0x00000030
⊞ UNLOCK	0
⊞ USD_UNLOCK	0
⊟ STS	0x00000010
ODF	□
EPPERR	☑
PRGMERR	□
OBF	□

- In case of enable erase/program protection error interrupt, continuing execution will enter the interrupt routine.

Figure 41. Write protection error interrupt



```
void FLASH_IRQHandler(void)
{
  if(flash_flag_get(FLASH_EPPERR_FLAG))
  {
    flash_flag_clear(FLASH_EPPERR_FLAG);
    delay_ms(500);
  }
}
```

4 Integrate codes and download

After codes of the solution provider and end user are configured, download to the same MCU on the premise of guaranteeing code security. Project_L0 and Project_L1 are used to introduce two downloading methods for reference.

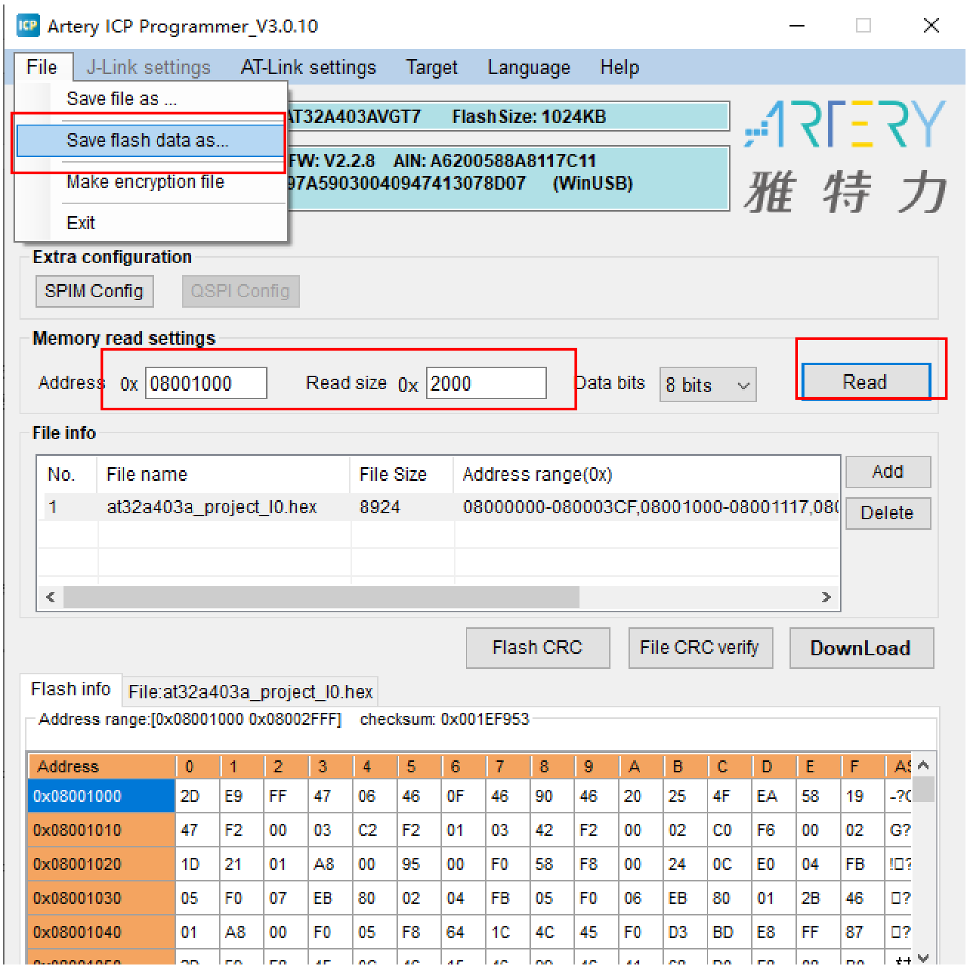
This operation involves offline downloading mode of AT-Link. For details, refer to operation manuals of ICP and AT-Link.

4.1 Program codes separately

Firstly, the solution provider programs SLIB codes to MCU; then, the end user programs application codes to MCU. The process is as follows:

(1) **Method A:** The solution provider uses ICP tool to save the SLIB code in the compiled project as BIN or HEX file: download the complete project to MCU (do not configure SLIB and FAP), read the corresponding SLIB codes (0x08001000~0x08002FFF) by using the memory access function; then click “File-Save Flash data as” to save codes as BIN or HEX file. In this example, it is named “slib.bin”, as shown in Figure 42.

Figure 42. Save SLIB codes



AT32A403AVGT7 Flash Size: 1024KB

FW: V2.2.8 AIN: A6200588A8117C11
97A59030040947413078D07 (WinUSB)

Address 0x 08001000 Read size 0x 2000 Data bits 8 bits

No.	File name	File Size	Address range(0x)
1	at32a403a_project_l0.hex	8924	08000000-080003CF,08001000-08001117,08(

File:at32a403a_project_l0.hex

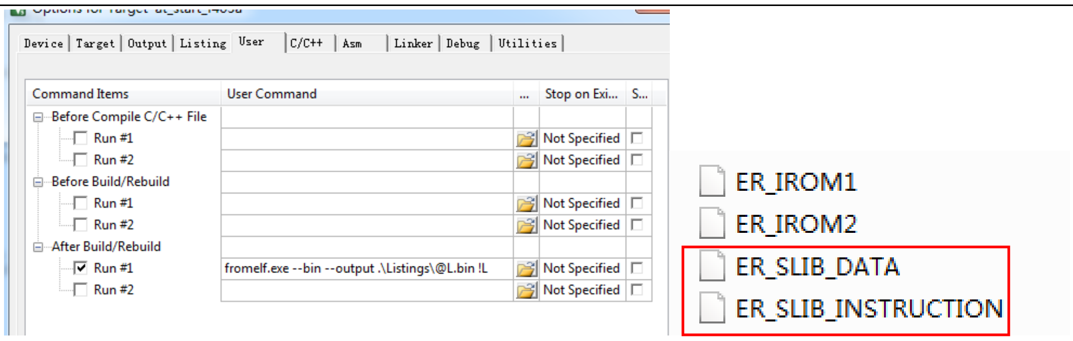
Address range:[0x08001000 0x08002FFF] checksum: 0x001EF953

Address	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	A!
0x08001000	2D	E9	FF	47	06	46	0F	46	90	46	20	25	4F	EA	58	19	-?(
0x08001010	47	F2	00	03	C2	F2	01	03	42	F2	00	02	C0	F6	00	02	G?
0x08001020	1D	21	01	A8	00	95	00	F0	58	F8	00	24	0C	E0	04	FB	!□?
0x08001030	05	F0	07	EB	80	02	04	FB	05	F0	06	EB	80	01	2B	46	□?
0x08001040	01	A8	00	F0	05	F8	64	1C	4C	45	F0	D3	BD	E8	FF	87	□?

Method B: The solution provider uses the compiled project to generate a .bin file directly, and take the corresponding section in the SLIB area. For example, in the KEIL project, add “fromelf.exe

--bin --output .\Listings\@L.bin !L” in the “user” option to generate a .bin file of the corresponding firmware, and add a suffix “.bin” to the SLIB area file. In this example, they are “ER_SLIB.bin” and “ER_SLIB_DATA.bin”, corresponding to the SLIB-INSTRUCTION file (0x08001000) and SLIB-DATA file (0x08002000), as shown in Figure 43.

Figure 43. Generate .bin file of SLIB code



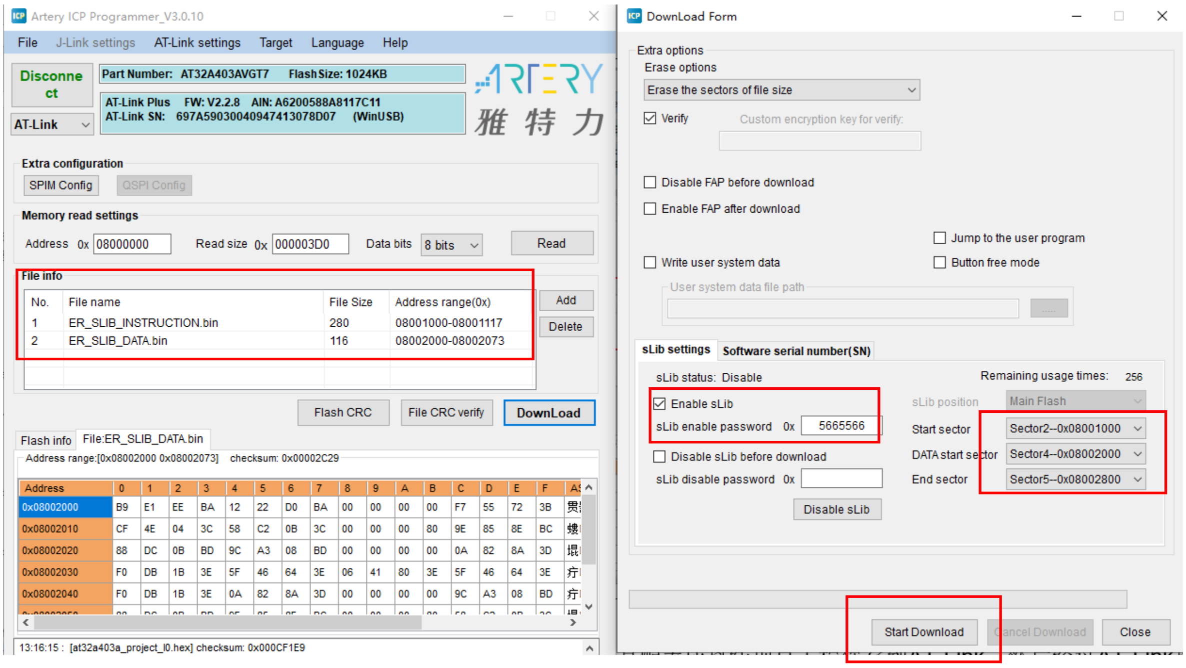
Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities

Command Items	User Command	...	Stop on Exi...	S...
Before Compile C/C++ File				
□ Run #1			Not Specified	□
□ Run #2			Not Specified	□
Before Build/Rebuild				
□ Run #1			Not Specified	□
□ Run #2			Not Specified	□
After Build/Rebuild				
☑ Run #1	fromelf.exe --bin --output .\Listings\@L.bin !L		Not Specified	□
□ Run #2			Not Specified	□

ER_IROM1
ER_IROM2
ER_SLIB_DATA
ER_SLIB_INSTRUCTION

(2) Use ICP Programmer to program the ER_SLIB_INSTRUCTION.bin and ER_SLIB_DATA.bin to MCU, as shown in Figure 44.

Figure 44.Online programming to MCU via ICP



Part Number: AT32A403AVGT7 Flash Size: 1024KB

AT-Link Plus FW: V2.2.8 AIN: A6200588A8117C11
AT-Link SN: 697A59030040947413078D07 (WinUSB)

Memory read settings: Address 0x 08000000 Read size 0x 000003D0 Data bits 8 bits

File info

No.	File name	File Size	Address range(0x)
1	ER_SLIB_INSTRUCTION.bin	280	08001000-08001117
2	ER_SLIB_DATA.bin	116	08002000-08002073

File:ER_SLIB_DATA.bin

Address range:[0x08002000 0x08002073] checksum: 0x00002C29

Address	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	AS
0x08002000	B9	E1	EE	BA	12	22	D0	BA	00	00	00	00	F7	55	72	3B	贯
0x08002010	CF	4E	04	3C	58	C2	0B	3C	00	00	00	80	9E	85	8E	BC	蝼
0x08002020	88	DC	0B	BD	9C	A3	08	BD	00	00	00	00	0A	82	8A	3D	焜
0x08002030	F0	DB	1B	3E	5F	46	64	3E	06	41	80	3E	5F	46	64	3E	疔
0x08002040	F0	DB	1B	3E	0A	82	8A	3D	00	00	00	00	9C	A3	08	BD	疔

```
13:16:15 :  [at32a403a_project_l0.hex] checksum: 0x000CF1E9
```

Erase options: Erase the sectors of file size

☑ Verify Custom encryption key for verify:

□ Disable FAP before download

□ Enable FAP after download

□ Jump to the user program

□ Write user system data

□ Button free mode

User system data file path

sLib status: Disable

Remaining usage times: 256

☑ Enable sLib

sLib enable password 0x 5665566

□ Disable sLib before download

sLib disable password 0x

sLib position: Main Flash

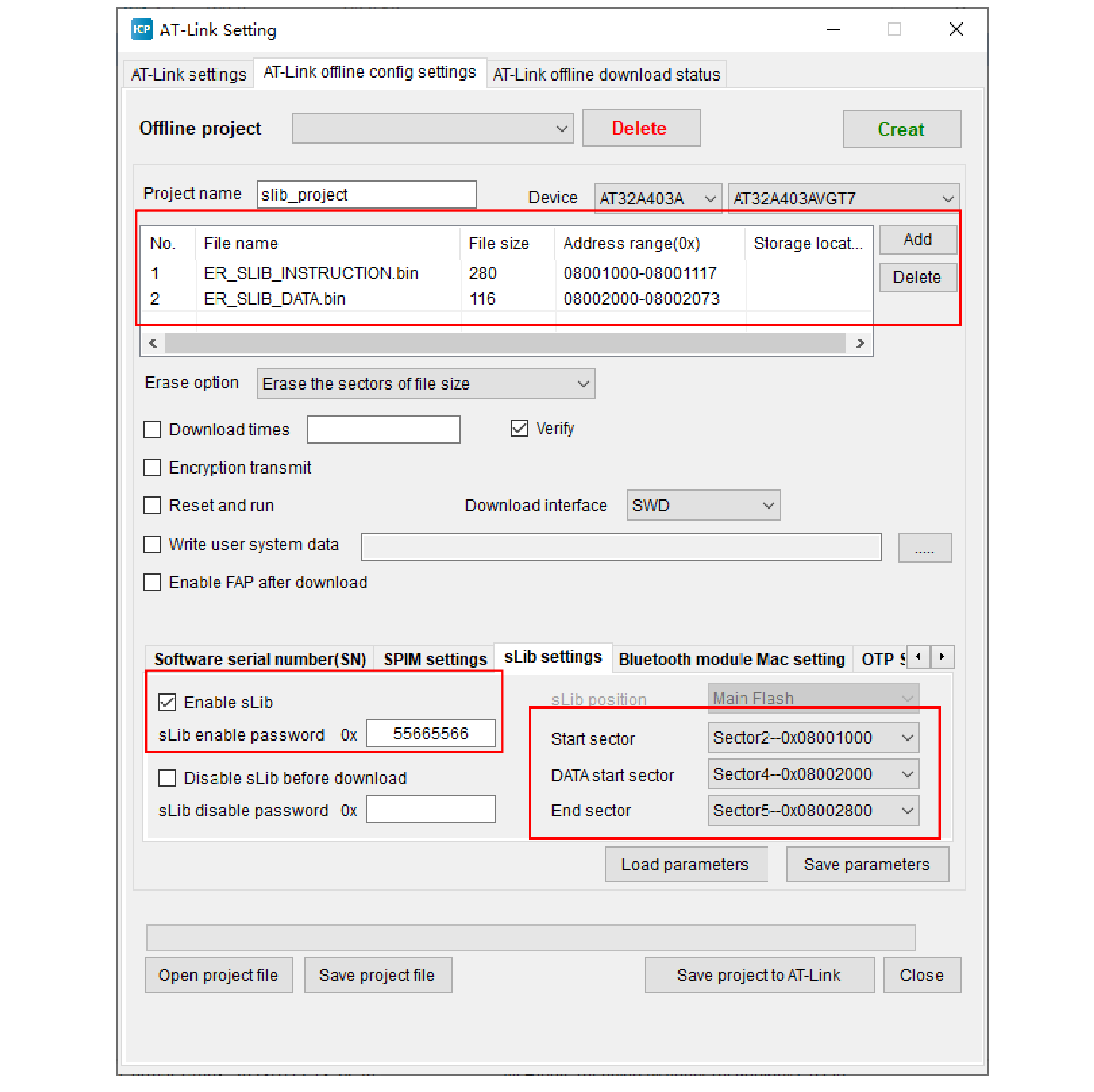
Start sector: Sector2--0x08001000

DATA start sector: Sector4--0x08002000

End sector: Sector5--0x08002800

(3) End users also can use ICP Programmer to set an offline project and save it to AT-Link, and then complete offline programming to MCU through AT-Link, as shown in Figure 45.

Figure 45. Offline programming to MCU via AT-Link



Offline project

Project name: slib_project

Device: AT32A403A, AT32A403AVGT7

No.	File name	File size	Address range(0x)	Storage locat...
1	ER_SLIB_INSTRUCTION.bin	280	08001000-08001117	
2	ER_SLIB_DATA.bin	116	08002000-08002073	

Erase option: Erase the sectors of file size

☐ Download times

☑ Verify

☐ Encryption transmit

☐ Reset and run

Download interface: SWD

☐ Write user system data

☐ Enable FAP after download

☑ Enable sLib

sLib enable password 0x 55665566

☐ Disable sLib before download

sLib disable password 0x

sLib position: Main Flash

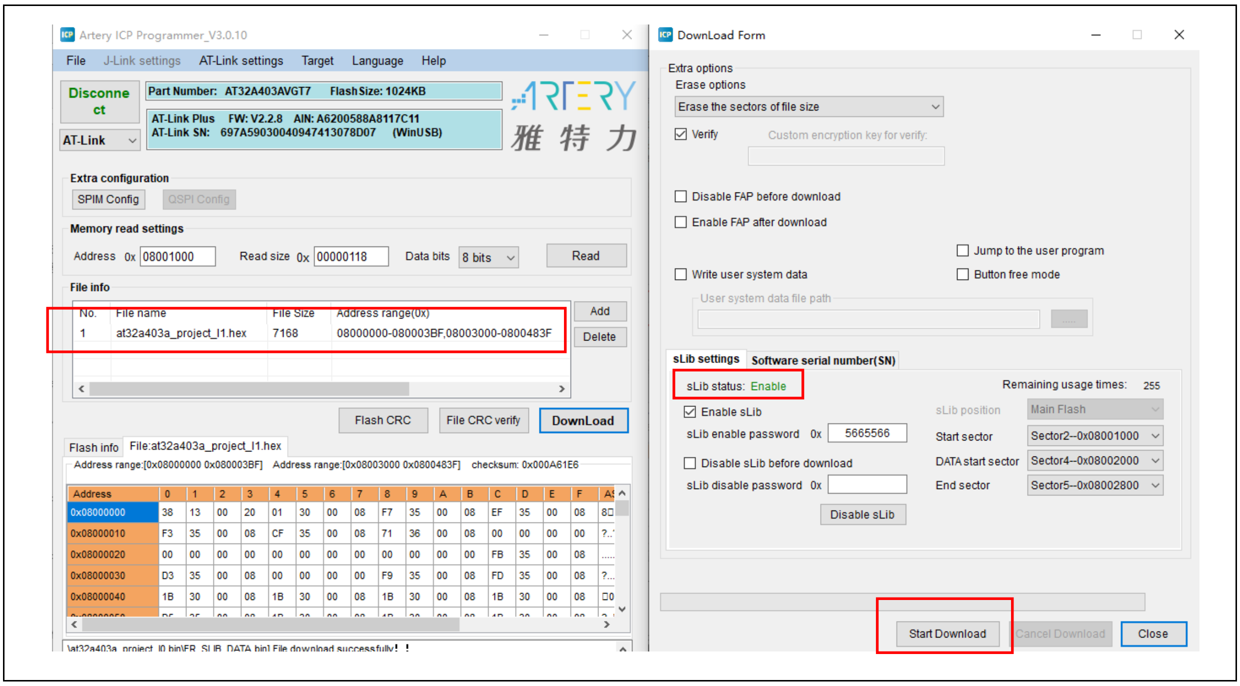
Start sector: Sector2--0x08001000

DATA start sector: Sector4--0x08002000

End sector: Sector5--0x08002800

(4) After completing step 2/3, end users can get the MCU with programmed SLIB area (SLIB status: enabled), and program the application code to MCU through online or offline programming, as shown in Figure 46.

Figure 46. End users program codes to MCU



Part Number: AT32A403AVGT7 Flash Size: 1024KB

AT-Link Plus FW: V2.2.8 AIN: A6200588A8117C11
AT-Link SN: 697A590300409474l3078D07 (WinUSB)

Extra configuration

Memory read settings

Address 0x 08001000 Read size 0x 00000118 Data bits 8 bits

File info

No.	File name	File Size	Address range(0x)
1	at32a403a_project_l1.hex	7168	08000000-080003BF,08003000-0800483F

Flash info File:at32a403a_project_l1.hex

Address range:[0x08000000 0x080003BF] Address range:[0x08003000 0x0800483F] checksum: 0x000A61E6

```
Address     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  AS
0x08000000  38 13 00 20 01 30 00 08 F7 35 00 08 EF 35 00 08 8□
0x08000010  F3 35 00 08 CF 35 00 08 71 36 00 08 00 00 00 00 ?..
0x08000020  00 00 00 00 00 00 00 00 00 00 00 00 FB 35 00 08 .....
0x08000030  D3 35 00 08 00 00 00 00 F9 35 00 08 FD 35 00 08 ?...
0x08000040  1B 30 00 08 1B 30 00 08 1B 30 00 08 1B 30 00 08 □0
```

\at32a403a_project_l0.bin\FR_SLIB_DATA.bin] File download successfully! !

Extra options

Erase options: Erase the sectors of file size

☑ Verify Custom encryption key for verify:

□ Disable FAP before download

□ Enable FAP after download

□ Jump to the user program

□ Write user system data

□ Button free mode

User system data file path

sLib settings | Software serial number(SN)

sLib status: Enable

Remaining usage times: 255

☑ Enable sLib

sLib enable password 0x 5665566

sLib position: Main Flash

Start sector: Sector2--0x08001000

□ Disable sLib before download

DATA start sector: Sector4--0x08002000

sLib disable password 0x

End sector: Sector5--0x08002800

4.2 Integrate and program codes

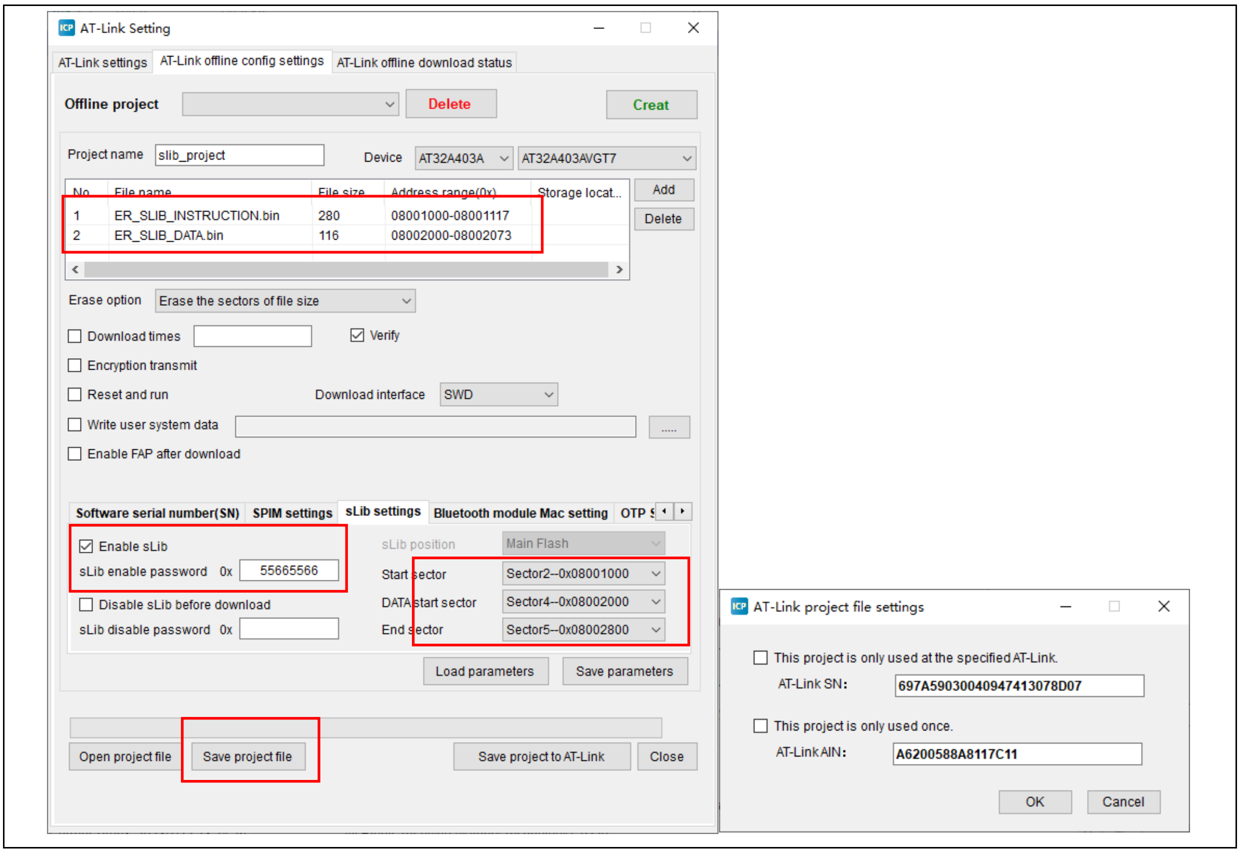
Integrate the SLIB code of solution provider and the end user application code to an offline project, and then download the integrated code to MCU through AT-Link offline programming. The process is as follows:

(1) The solution provider handles the compiled project as aforementioned to get a slib.bin file;

(2) The solution provider uses ICP Programmer to generate an offline project and save it to PC. Parameters (such as number of download, project files binding to AT-Link and enable FAP after download) can be configured as needed. Save the offline project as follows

Note: The offline project is encrypted. To enhance security, the solution provider also can set the slib.bin file as an encrypted slib.benc file and then add it to the offline project. In this case, the offline project can only be used on the AT-Link with the corresponding encryption key

Figure 47. Set offline project



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: AT32A403A AT32A403AVGT7

No	File name	File size	Address range(0x)	Storage locat...
1	ER_SLIB_INSTRUCTION.bin	280	08001000-08001117	
2	ER_SLIB_DATA.bin	116	08002000-08002073	

Add | Delete

Erase option: Erase the sectors of file size

□ Download times   ☑ Verify

□ Encryption transmit

□ Reset and run   Download interface: SWD

□ Write user system data

□ Enable FAP after download

Software serial number(SN) | SPIM settings | sLib settings | Bluetooth module Mac setting | OTP

☑ Enable sLib

sLib enable password 0x 55665566

□ Disable sLib before download

sLib disable password 0x

sLib position: Main Flash

Start sector: Sector2--0x08001000

DATA start sector: Sector4--0x08002000

End sector: Sector5--0x08002800

Load parameters | Save parameters

Open project file | Save project file | Save project to AT-Link | Close

AT-Link project file settings

□ This project is only used at the specified AT-Link.

AT-Link SN: 697A59030040947413078D07

□ This project is only used once.

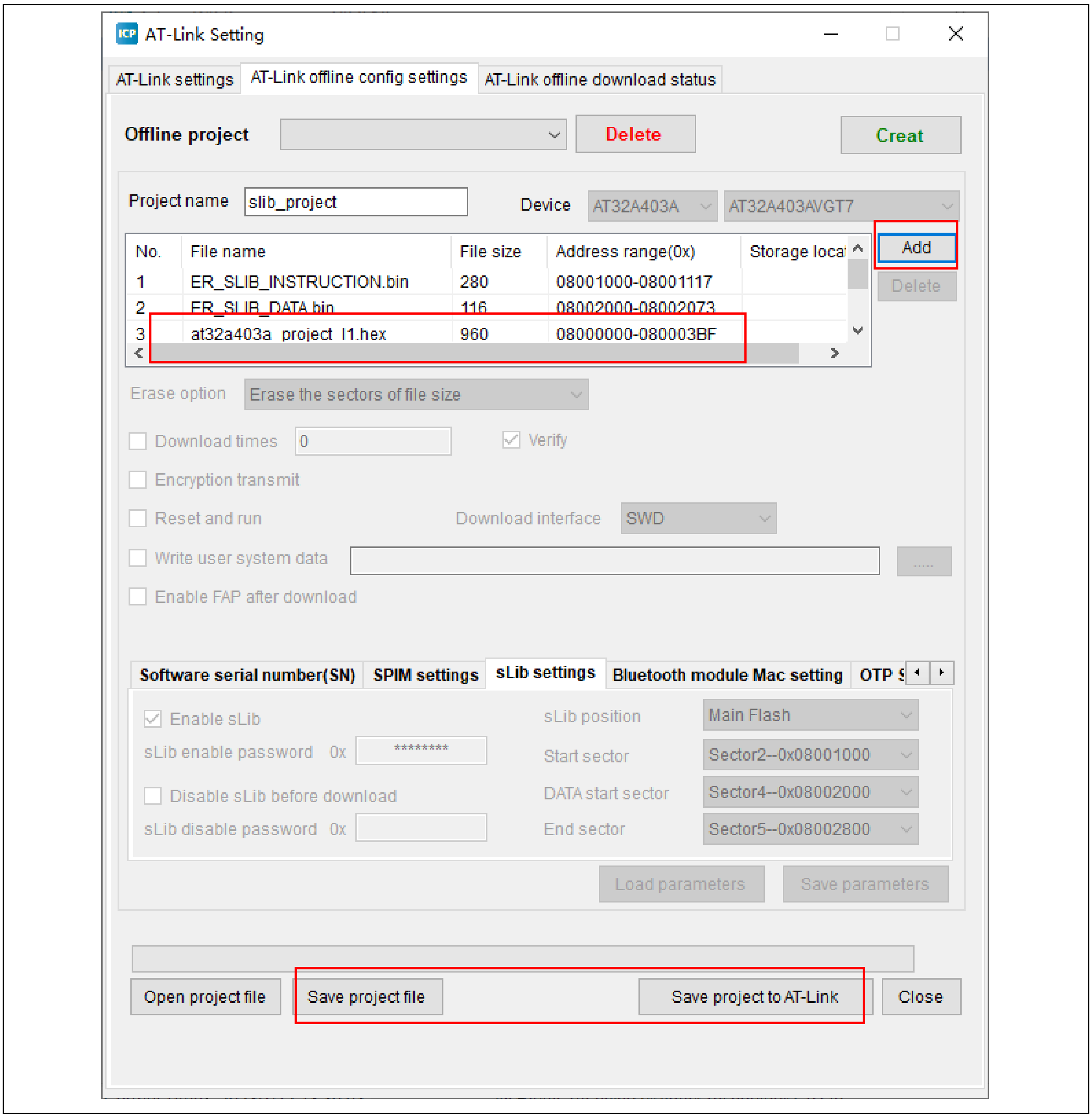
AT-Link AIN: A6200588A8117C11

OK | Cancel

(3) After obtaining the offline project, the end user should use ICP Programmer to open the project file and add the application codes to the offline project; then save to PC or AT-Link, and perform offline download. Figure 48 shows how to add the project file.

Note: To protect codes from being leaked or decoded, do not change other settings when adding code file to the offline project, which requires the solution provider to configure the final settings in advance.

Figure 48. Add project file



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: AT32A403A | AT32A403AVGT7

No.	File name	File size	Address range(0x)	Storage loca
1	ER_SLIB_INSTRUCTION.bin	280	08001000-08001117	
2	ER_SLIB_DATA.bin	116	08002000-08002073	
3	at32a403a_project_l1.hex	960	08000000-080003BF	

Erase option: Erase the sectors of file size

□ Download times: 0

☑ Verify

□ Encryption transmit

□ Reset and run

Download interface: SWD

□ Write user system data

□ Enable FAP after download

Software serial number(SN) | SPIM settings | sLib settings | Bluetooth module Mac setting | OTP S

☑ Enable sLib

sLib enable password 0x: ********

□ Disable sLib before download

sLib disable password 0x:

sLib position: Main Flash

Start sector: Sector2--0x08001000

DATA start sector: Sector4--0x08002000

End sector: Sector5--0x08002800

5 Revision history

Table 2. Document revision history

Date	Version	Revision note
2023.07.11	2.0.0	Initial release.

IMPORTANT NOTICE – PLEASE READ CAREFULLY

Purchasers are solely responsible for the selection and use of ARTERY's products and services; ARTERY assumes no liability for purchasers' selection or use of the products and the relevant services.

No license, express or implied, to any intellectual property right is granted by ARTERY herein regardless of the existence of any previous representation in any forms. If any part of this document involves third party's products or services, it does NOT imply that ARTERY authorizes the use of the third party's products or services, or permits any of the intellectual property, or guarantees any uses of the third party's products or services or intellectual property in any way.

Except as provided in ARTERY's terms and conditions of sale for such products, ARTERY disclaims any express or implied warranty, relating to use and/or sale of the products, including but not restricted to liability or warranties relating to merchantability, fitness for a particular purpose (based on the corresponding legal situation in any unjudicial districts), or infringement of any patent, copyright, or other intellectual property right.

ARTERY's products are not designed for the following purposes, and thus not intended for the following uses: (A) Applications that have specific requirements on safety, for example: life-support applications, active implant devices, or systems that have specific requirements on product function safety; (B) Aviation applications; (C) Aerospace applications or environment; (D) Weapons, and/or (E) Other applications that may cause injuries, deaths or property damages. Since ARTERY products are not intended for the above-mentioned purposes, if purchasers apply ARTERY products to these purposes, purchasers are solely responsible for any consequences or risks caused, even if any written notice is sent to ARTERY by purchasers; in addition, purchasers are solely responsible for the compliance with all statutory and regulatory requirements regarding these uses.

Any inconsistency of the sold ARTERY products with the statement and/or technical features specification described in this document will immediately cause the invalidity of any warranty granted by ARTERY products or services stated in this document by ARTERY, and ARTERY disclaims any responsibility in any form.