

AN0038

Application Note

AT32F413 Security Library Application Note

Introduction

This application note mainly introduces the security library (sLib) application principle of AT32F413 MCUs, operation methods and example projects.

Applicable products:

Part number	AT32F413
-------------	----------

Contents

1	Ove	rview.		6		
2	Арр	Application principles				
	2.1	Applic	cation principle of sLib	7		
	2.2	How to	o enable sLib protection	9		
	2.3	How to	o disable sLib protection	9		
	2.4	Comp	ile and execute program in sLib	10		
		2.4.1	Setting interrupt vector table as security library not allowed	11		
		2.4.2	Correlation between sLib area and user code area	11		
3	Exa	mple a	pplications of sLib	14		
	3.1	Exam	ple application requirements			
		3.1.1	Hardware requirements	14		
		3.1.2	Software requirements	14		
	3.2	Overv	<i>r</i> iew			
	3.3	SLIB p	protected code: FIR low-pass filter	15		
	3.4	Projec	ct_L0: example for solution providers			
		3.4.1	Generate execute-only code	16		
		3.4.2	Compile security library address	18		
		3.4.3	Enable sLib protection	22		
		3.4.4	Project_L0 execution process	24		
		3.4.5	Generate header file and symbol definition file	26		
	3.5	Projec	ct_L1: example for end users			
		3.5.1	Create user application project	28		
		3.5.2	Add symbol definition file to project	29		
		3.5.3	Call functions in SLIB-protected area	30		
		3.5.4	Project_L1 execution process	30		
		3.5.5	SLIB protection in debug mode	31		
4	Inte	grate c	odes and download	35		
	4.1	Progra	am codes separately			
	4.2	Integra	ate and program codes			
5	Rev	ision h	nistory	41		



List of tables

Table 1. Flash size of AT32F413 MCUs	8
Table 2. Document revision history	41

,**:17[**7]

List of figures

Figure 1. Mapping of main Flash memory featured with sLib	8
Figure 2. Literal pool example (1)	
Figure 3. Literal pool example (2)	11
Figure 4. Function in sLib area calling the function in user code area	
Figure 5. Example of user-defined function	
Figure 6. Example application process	14
Figure 7. Example application	
Figure 8. FIR low-pass filter	
Figure 9. Enter Option interface in Keil	
Figure 10. Tick Execute-only Code in Keil	
Figure 11. Enter Option interface in IAR	
Figure 12. Set C/C++ options in IAR	
Figure 13. Main Flash memory mapping and RAM partition	
Figure 14. Set Linker option in Keil	
Figure 15. Modify scatter in Keil	20
Figure 16. Modify SLIB RAM address in KEIL code	20
Figure 17. Modify SLIB constant address in KEIL code	21
Figure 18. SLIB address definition in icf file	21
Figure 19. Address assignment in icf file	21
Figure 20. Modify IP-Code RAM in icf file	22
Figure 21. Modify SLIB constant address in IAR code	22
Figure 22. Configure ICP Programmer	23
Figure 23. Set parameters in Download Form	24
Figure 24. Project_L0 execution process	25
Figure 25. Set Misc controls in Keil	
Figure 26. Contents of modified fir_filter_symbol.txt	
Figure 27. Set Build Actions in IAR	27
Figure 28. Edit steering_file.txt content	27
Figure 29. Modified scatter file	
Figure 30. Modified icf file	
Figure 31. Add symbol definition file in Keil	
Figure 32. Modify symbol definition file type to "Object file"	
Figure 33. Add symbol definition file in IAR	
Figure 34. Project_L1 execution process	

<u>, 175-77;</u>

AT32F413 Security Library Application Note

Figure 35. Enter Show Disassembly at Address	32
Figure 36. Set Show Code at Address	32
Figure 37. View codes	32
Figure 38. View codes in Memory	33
Figure 39. View SLIB_DATA start sector in Memory	33
Figure 40. SLIB write test	33
Figure 41. Write protection error interrupt	34
Figure 42. Save SLIB codes	35
Figure 43. Generate bin file of SLIB code	36
Figure 44. Online MCU programming in ICP	36
Figure 45.Offline programming to MCU via AT-Link	37
Figure 46. End users program codes to MCU	38
Figure 47. Set offline project	39
Figure 48. Add project file	40

475<u>-</u>171²

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 AT32F413 series 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 AT32F413 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 includes 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 (can only be executed) by MCU through I-Code bus and cannot be read through D-Code (including ISP/ICP debug mode and programs that boot from internal RAM). When accessing the SLIB_INSTRUCTION in the manner of reading data, values are all read 0xFF.
- Data in the data security library (SLIB_DATA) can only be read through D-Code bus and cannot be programmed.
- The program code and data in 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 previously defined 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.

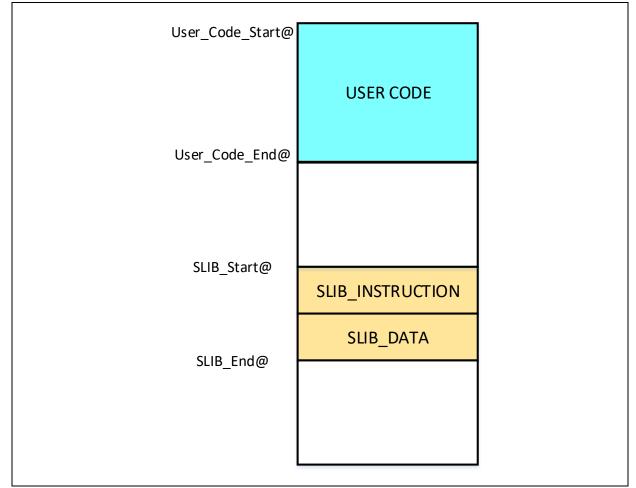


Figure 1. Mapping of main Flash memory featured with sLib

The range of sLib is set by sector, 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 AT32F413 series MCUs.

Table 1. Flash size of AT32F413 MCUs

Part number	Internal Flash size (byte)	Sector size (byte)	Configurable range
AT32F413x8	64K	1K	Sector 4 ~ 62 (0x08001000 ~ 0x0800FBFF)
AT32F413xB	128K	1K	Sector 4 ~ 62 (0x08001000 ~ 0x0800FBFF)
AT32F413xC	256K	2К	Sector 2 ~ 30 (0x08001000 ~ 0x0800F7FF)

AT32F413 Security Library Application Note

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. Write 0xA35F6D24 to the SLIB_UNLOCK register, and check the SLIB_ULKF bit in the SLIB_MISC_STS register to verify if it is unlocked successfully. Then, set values can be written 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 sectors to be protected in the SLIB_SET_RANGE register, including the address of instruction and data areas;
- Wait until the OBF bit becomes "0";
- Set a 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 register to verify the security library settings.

Notes:

- It is allowed to set security library in the main Flash memory only, and refer to <u>Table 1</u> for the configurable 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) of the main Flash memory, which should not be configured as security library;
- Codes to be protected by the security library should not be placed in the first 4 KB of main Flash memory.

For details of security library setting register, please refer to *AT32F413 Series Reference Manual*. The program to start security library can be found in the slib_enable() function in main.c file of project_I0. In addition, users can use Artery ICP/ISP tools for configuration.

2.3 How to disable sLib protection

The security library protection can be disabled by writing the previously defined password to the SLIB_PWD_CLR register. While disabling security library protection, MCU will perform mass erase operation to 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 ensure that there is no other ongoing programming operation;
- Write the previously defined password to the SLIB_PWD_CLR register;
- Perform system reset, and then reload security library setting word;
- Read the SLIB_STS0 register to check the security library settings.

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 in the manner of reading data, 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 CRM_CFG register in a PC (program counter) indirect addressing manner, and the address of 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 during execution 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.

=> <mark>0x0</mark>	800479A	4F39 LDR		r7,[pc,#228]	; @0x08004880
	B00479C			r7,[r7,#0x04]	
0x0		F3C70381 UBF		r3,r7,#2,#2	
	81: s 82: {	witch(sclk_so	urce)		
	83:	case CRM SCL	K HICK:		
4			_		
	main.c	startup_at32f40	3a_407.s 📋 a	at32f403a_407_clock.c	system_at32f403a_407.c at32f403a_407_crm.c at32f403a_407_gpio.c
	77				
	78	/* get s	clk sour	ce */	
\triangleright	79				pe)CRM->cfg bit.sclksts;
	80		200 (0		po, chai , c18_0100 collidos,
	81	switch(s	clk sour	ce)	
	82 0	{	oin_boai	00)	
	83	case C	RM SCLK	HICK	
	84				ck to sclk) != RESET) && ((CRM->misc1 bit.hickdi
	UT				HICK VALUE * 6;
Ľ	85	0.17			
I,	85		_		
5	85 86 87	else	_	_	HICK VALUE:

Figure 2. Literal pool example (1)



		E_CIOCK - SYSC	em_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

Figure 3. Literal pool example (2)

2.4.1 Setting interrupt vector table as security library 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 as security library;
- Program codes to be protected by security library should not be placed in the first sector.

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 not 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.



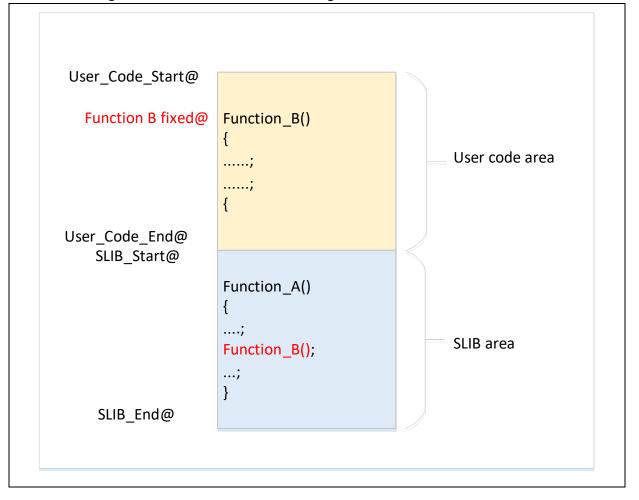


Figure 4. Function in sLib area calling the function in user code area

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.

The two solutions are recommended:

- 1) Compile into the sLib area (refer to Keil or IAR documents for details about implementation).
- 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 - lu)) * 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 Example applications of sLib

This section introduces example applications of sLib and how to complete these applications step by step.

3.1 Example application requirements

3.1.1 Hardware requirements

- AT-START-F413 demo board with AT32F413RCT7 chip
- AT-Link emulator for debugging example applications

3.1.2 Software requirements

- Keil® µvision IDE (µvision V5.18.0.0 is used in this example) or IAR Embedded workbench IDE (IAR V8.22.2 is used in this example)
- Artery ICP/ISP programming tools for enabling or disabling sLib

3.2 Overview

This application note provides two example 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: Apply algorithm for end users

The algorithm completed in Project_L0 will be pre-downloaded and pre-burned to AT32F413 chip and configured as sLib protected. In addition, the following settings are available for the end-user applications.

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

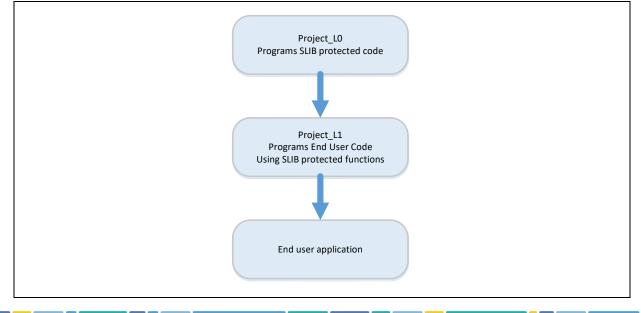
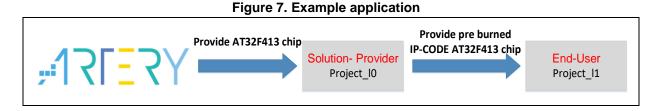


Figure 6. Example application process



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

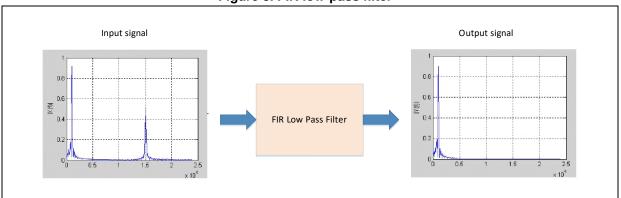


Figure 8. FIR low-pass filter

CMSIS DSP library functions and files to be used are:

arm_fir_init_f32()

It is used for initialization of filter function, which is included in "arm_fir_init_f32.c" file.

• arm_fir_f32()

It is the main part of filter algorithm, which 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 for the end user to call, which 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 data security library in the example.

In this example, FPU and DSP instructions in the MCU are used for signal processing and floating point arithmetic to realize accurate calculation and correct output signal.

3.4 **Project_L0: example for solution providers**

The following projects are completed in this example:

- Compile the algorithm-related functions to execute-only code;
- Place the algorithm program code to the main Flash memory sector 2;
- Place the coefficients of filter functions to the main Flash memory sector 4;
- Execute FIR_lowpass_filter() in the main program to verify its correctness;
- If correct, set section 2/3 as the instruction security library and section 4/5 as the data security library, which can be completed by calling the slib_enable() function in the main program or using Artery ICP Programmer (it is recommended to use ICP tools to complete configurations);
- 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

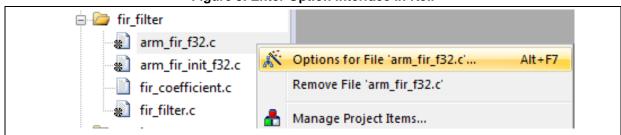
Each toolchain has specific setting options to prevent the compiler generating literal pools and branch table that can read data while executing instructions, such as "LDR Rn, [PC, #offset]". Section 2.4 lists examples of literal pools and branch table.

For Keil® μ vision, it has the Execute-only Code option, which can be set as follows:

Keil® µvision: Set Execute-only Code option

Operate as follows:

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



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

Figure 9. Enter Option interface in Keil

Options for File 'arm_fir_f32.c'		×
Properties C/C++ Preprocessor Symbols Define: Undefine:		
Language / Code Generation Iveration Iverati	I Strict ANSI C I Enum Container always int I Plain Char is Signed I Read-Only Position Independent I Read-Write Position Independent	Wamings: All Wamings ▼ ✓ Thumb Mode ✓ No Auto Includes ✓ C99 Mode
	tex-M4.fp -DMICROLIB -g -O0apcs=interwo Nibiraries\cmsis\cm4\core_support -l Cancel Defaults	\\libraries\cmsis

Figure 10. Tick Execute-only Code in Keil

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

IAR: Set No data read in code memory option

Operate as follows:

• Select the corresponding file in the fir_filter group; right click and select Option;

Figure 11. Enter Option interface in IAR

├	
⊞ <mark>arm_fir_f32.c</mark> ⊞ ⊡ arm_fir_init_f32.c	Options
⊡ i fir_coefficient.c	Maka
└─⊞ lo fir_filter.c	Make

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



Options for node "arm_fir_f	f32.c*
Exclude from build	
Category: Static Analysis	Override inherited settings Factory Settings
C/C++ Compiler Custom Build	Preprocessor Diagnostics MISRA-C:2004
	MISRA-C:1998 Encodings Extra Options Language 1 Language 2 Code Optimizations Output List
	Processor mode Arm Thumb Position-independence Code and read-only data (ropi) Read/write data (rwpi) No dynamic read/write initializati. Mo data reads in code memory
	OK Cancel

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

• The arm_fir_f32.c, arm_fir_init_f32.c and fir_filter.c files are in the SLIB_INSTRUCTION area, and these files need to be set 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 sector 2 in this example, with sector 2 and sector 3 being set as instruction security library, and sector 4 and sector 5 being set as 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.



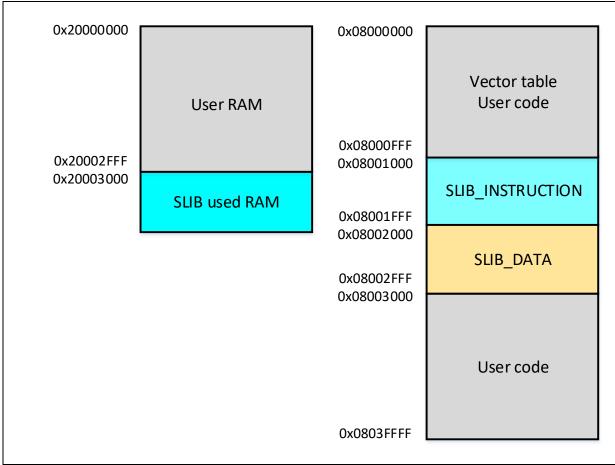


Figure 13. Main Flash memory mapping and RAM partition

Keil® µvision: scatter file

Operate 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:

Options fo	r Target 'at_start_f403a'	
Device Tar	get Output Listing Vser C/C++ Asm Linker Debug Utilities	1
☐ Mak ☐ Mak	Not Layout from Target Dialog X/O Base: PRW Sections Position Independent R/O Base: PRO Sections Position Independent R/O Base: Dx20000000 Dx20000000 Search Standard Libraries Dx20000000 Inight fail Conditions as Errors disable Warnings:	
Misc controls Linker control string	symdefs=fir_filter_symbol.bd cpu Cortex-M4.fp *.o -library_type=microlibstrictscatter ".\slib-w-xo.sct"	× •
	OK Cancel Defaults H	lelp

Open scatter file, load the object file of the code to be placed in SLIB_INSTRUCTION area to



"LR_SLIB_INSTRUCTION" (a dedicated loading area that starts from sector 2 and occupies two sectors), and modify the label to "execute-only (+XO)". In addition, place the area occupied by 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 0x20003000 to 0x20003FFF to the algorithm functions to avoid the same RAM region being used by end-user project, causing fault or error in program execution.

```
Figure 15. Modify scatter 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 0x00003000 { ; user RW data
   .ANY (+RW +ZI)
  RW_IRAM2 0x20003000 0x00001000 { ; RAM used for slib code
   fir filter.o (+RW +ZI)
LR_SLIB_INSTRUCTION 0x08001000 0x00001000 {
                                                   ; slib instruction area
  ER_SLIB_INSTRUCTION 0x08001000 0x00001000 { ; load address = execution address
    arm_fir_init_f32.o (+X0)
arm_fir_f32.o (+X0)
    fir_filter.o (+X0)
}
LR_SLIB_DATA 0x08002000 0x00001000 {
ER_SLIB_DATA 0x08002000 0x00001000 {
                                            ; sLib data area
   fir_coefficient.o (+R0)
LR_IROM2 0x08003000 0x0003D000 {
                                        ; user code area
  ER_IROM2 0x08003000 0x0003D000 { ; load address = execution address
   \overline{A}NY (+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 0x20003000, as shown in Figure 16.

Figure 16	. Modify	SLIB	RAM	address	in KEIL coo	le
		•=		uuu i 0000		

```
##if defined (__ICCARM__)
static float32_t firStateF32[BLOCK_SIZE + NUM_TAPS - 1] @ 0x20003000 ;
#elif defined (__CC_ARM )
static float32_t firStateF32[BLOCK_SIZE + NUM_TAPS - 1]
#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 the 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 code

<pre>##if defined (ICCARM) #const float32_t firCoeffs32[NUM_TAPS] @ 0x08002000 ={ #elif defined (CC ARM)</pre>
<pre>lconst float32_t firCoeffs32[NUM_TAPS]attribute((at(0x08002000))) = { #endif</pre>
-0.0018225230f, -0.0015879294f, +0.000000000f, +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 };

IAR: ICF file

Operate as follows:

 Open the icf file in \project_I0\IAR_V8.2\, and add three new SLIB loading areas as shown in Figure 18. The SLIB_RAM region reserves the RAM (0x20003000 ~ 0x20003FFF) 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__ = 0x20003FFF;
/* SLIB RAM region */
define symbol __ICFEDIT_region_SLIB_RAM_start__ = 0x20003000;
define symbol __ICFEDIT_region_SLIB_RAM_start__ = 0x20003000;
define symbol __ICFEDIT_region_SLIB_RAM_end__ = 0x20003000;
define symbol __ICFEDIT_region_SLIB_RAM_end__ = 0x20003FFF;
```

 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 reserved.

Figure 19. Address assignment in icf file

<pre>/* Reserved 0x08001000 ~ 0x08002FFF as SLIB area */ define region ROM_region = mem:[fromICFEDIT_region_ROM_start toICFEDIT_region_ROM_end_] -mem:[fromICFEDIT_region_SLIB_INSTRUCTION_start toICFEDIT_region_SLIB_INSTRUCTION_end_] -mem:[fromICFEDIT_region_SLIB_DATA_start toICFEDIT_region_SLIB_DATA_end_];</pre>
define region SLIB_INSTRUCTION_region = mem:[from _ICFEDIT_region_SLIB_INSTRUCTION_start_ to _ICFEDIT_region_SLIB_INSTRUCTION_end_];
define region SLIB_DATA_region = mem:[fromICFEDIT_region_SLIB_DATA_start_ toICFEDIT_region_SLIB_DATA_end_];
<pre>/* Reserved 0x20003000 ~ 0x20003FFF as RAM used for SLIB code */ define region RAM_region = mem:[fromICFEDIT_region_RAM_start toICFEDIT_region_RAM_end]</pre>
define region SLIB_RAM_region = mem:[fromICFEDIT_region_SLIB_RAM_start toICFEDIT_region_SLIB_RAM_end];

• For the RAM used by IP-Code, users can use the IAR @ descriptor to load variables to a fixed address 0x20003000 or modify the icf file, as shown in Figure 20.



Figure 20. Modify IP-Code RAM in icf file

 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 icf file as aforementioned, or use the IAR @ descriptor to load the constants to a fixed address, as shown in Figure 21.

Figure 21. Modify SLIB constant address in IAR code

```
#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) Use Artery ICP Programmer (recommended)

It is recommended to use the ICP Programmer as follows:

- Connect AT-Link emulator to AT-START-F413 board and 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.



isconnect Part Number: AT32F413RCT7 FlashSize: 256KB	File	J-Link	settings	AT-	-Link se	ettings	s Ta	arget	La	ngua	age	He	lp					
AT-Link-EZ FW: V1.5.8 AT-Link SN: AB68855120040F15809179C02	Discon	nect	Part N	lumbe	r: AT	32F41	.3RCT	7	Flas	hSiz	e: 25	6KB			1	2	-	7 Y
SPIM FLASH_DA 0x 0 ● Remap0 (Use PA11/PA12 pins) Type Select • Remap1 (Use PB10/PB11 pins) Memory read settings address 0x 08000000 Read size No. File name File size Address range(0x) Add 1 project_l0.hex 8484 08000000-080003B7,08001000-08001117,08 Delete Flash CRC File CRC verify DownLoad ash info File:project_l0.hex						V1.5.	8								-			
SPIM FLASH_DA 0x 0 ● Remap0 (Use PA11/PA12 pins) Type Select • Remap1 (Use PB10/PB11 pins) Memory read settings address 0x 08000000 Read size No. File name File size Address range(0x) Add 1 project_l0.hex 8484 08000000-080003B7,08001000-08001117,08 Delete Flash CRC File CRC verify DownLoad ash info File:project_l0.hex	T-Lin	k 🔻	AT-Lin	k SN:	AB68	551200	040F1	58091	179C0	2					猚	1		カ
Aemory read settings Address 0x 0800000 Read size 0x 00006870 Data bits 8 bits • Read ile info Image: Read Add Add No. File name File size Address range(0x) Add 1 project_l0.hex 8484 08000000-080003B7,08001000-08001117,08 Delete Image: Read Image: Read Image: Read Image: Read Image: Read Read Image: Read Image: Read Image: Read Image: Read Image:			SPIN	и	F	LASH I	DA 0x		0	0	۵) Re	map	0 (Us					
Aemory read settings Address 0x 0800000 Read size 0x 00006870 Data bits 8 bits • Read ile info Image: Read Add Add No. File name File size Address range(0x) Add 1 project_l0.hex 8484 08000000-080003B7,08001000-08001117,08 Delete Image: Read Image: Read Image: Read Image: Read Image: Read Read Image: Read Image: Read Image: Read Image: Read Image:			Туре						elect		D Re	map	1 (Us	e PB1	LO/PB	11 pir	ns)	
Address 0x 0800000 Read size 0x 00006870 Data bits 8 bits Read ile info No. File name File size Address range(0x) 1 project_l0.hex 8484 08000000-080003B7,08001000-08001117,08 Pelete Flash CRC File CRC verify DownLoad ash info File:project_l0.hex Address range:[0x08000000 0x080003B7] Address range:[0x08001000 0x08001117] Address range:[0x08002000 0x08004BDF] checksum: 0x000C612F	Memo	rv read		15												1		
ile info No. File name File size Address range(0x) Add 1 project_l0.hex 8484 08000000-080003B7,08001000-08001117,08 Delete III Filesh CRC File CRC verify DownLoad ash info File:project_l0.hex Address range:[0x08000000 0x080003B7] Address range:[0x08001000 0x08001117] Address range:[0x08000000 0x080003B7]		· .													1		0	
No. File name File size Address range(0x) Add 1 project_l0.hex 8484 08000000-080003B7,08001000-08001117,08 Delete III IIII IIII IIII IIIII IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII	Addre	ss Ox	0800000	0	Read	size	0x 00	0068	70	Da	ata b	its	5 bits	•			кеас	
1 project_l0.hex 8484 08000000-080003B7,08001000-08001117,08 Delete III III Flash CRC File CRC verify DownLoad ash info File:project_l0.hex Address range:[0x0800000 0x080003B7] Address range:[0x08001000 0x08001117] Address range:[0x08002000 0x08004BDF] checksum: 0x000C612F	File in	fo																
Flash CRC File CRC verify DownLoad File:project_10.hex Address range:[0x0800000 0x080003B7] Address range:[0x08001000 0x08001117] Address range:[0x08002000 bx08002073] Address range:[0x08003000 0x08004BDF] checksum: 0x000C612F																		
Flash CRC File CRC verify DownLoad ash info File:project_10.hex	No.	File n	ame			File	size	Ad	dress	rang	e(0x))					A	dd
Flash CRC File CRC verify DownLoad ash info File:project_10.hex				c									0010	00-08	0011	17,08		
Flash CRC File CRC verify DownLoad ash info File:project_10.hex				¢									0010	00-08	30011	17,08		
ash info File:project_10.hex Address range:[0x08000000 0x080003B7] Address range:[0x08001000 0x08001117] Address range:[0x08002000 bx08002073] Address range:[0x08003000 0x08004BDF] checksum: 0x000C612F				¢									0010	00-08	30011	17,08		
ash info File:project_10.hex Address range:[0x08000000 0x080003B7] Address range:[0x08001000 0x08001117] Address range:[0x08002000 bx08002073] Address range:[0x08003000 0x08004BDF] checksum: 0x000C612F	1			κ									0010	00-08	30011			
Address range:[0x08000000 0x080003B7] Address range:[0x08001000 0x08001117] Address range:[0x08002000 0x080020731 Address range:[0x08003000 0x08004BDF1 checksum: 0x000C612F	1			κ					00000	0-080	00031	B7,08				•	De	lete
0x080020731 Address range: [0x08003000 0x08004BDF] checksum: 0x000C612F	1	proje	ect_I0.hex						00000	0-080	00031	B7,08				•	De	lete
uddress 0 1 2 3 4 5 6 7 8 9 A B C D E F A.	1 Flash in	proje	e:project	t_l0.hex		8484	4	080	Flash	0-080	00031	B7,08	• CRC	C verif	y [Þ Dov	De	lete
	1 < Flash in Addres	proje	e:project	t_10.hex	80003B7	8484	4 dress ra	080	Flash	0-080 CRC	0x08	B7,08	• CRC	C verif	y [Þ Dov	De	lete
	1 Iash ir Addres 0x0800 	proje	e:project :[0x08000 Address I	t_10.hex 0000 0x0 range:10	80003B7 0x080030 2 3	8484] Add 00 0x0	4 dress ra 8004BE	080 ange:[0F1 c 6	Flash 0x0800 thecks 7	0-080 0-080 0 CRC 01000 um: 0 8	0x080	File 00111	• CRC	verif ddres	y (• Dov e:[0x0	De wnLc	elete

Figure 22. Configure ICP Programmer

• 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 as the end sector; set the enable password as "0x55665566" (user-defined), tick "Enable SLIB", and then click "Start Download" to complete programming and enable SLIB, as shown in Figure 23.



sLib status: Disable Enable password 0x 55665566	Remaining usage times: 246
Disable password 0x	Start sector Sector 20x08001000
Disable sLib	DATA start sector Sector 40x08002000
	End sector Sector 50x08002800
Extra options	
Erase the sectors of file size	Disable sLib before download
Verify Custom encryption key	y for verify: 🛛 Enable sLib
	Disable FAP before download
Jump to the user program	Enable FAP after download
🔲 Write software serial number(SN)	
Write address 0x 08010000	Button free mode
Current SN 0x 00000001	
Increase step 0x 00000001	
Write user system data	
User system data file path	

Figure 23. Set parameters in Download Form

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

(2) Use slib_enable() function 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 expected (SNR larger than the preset threshold), the green LED on the board blinks; otherwise, the red LED blinks. Figure 24 shows the Project_L0 execution process.

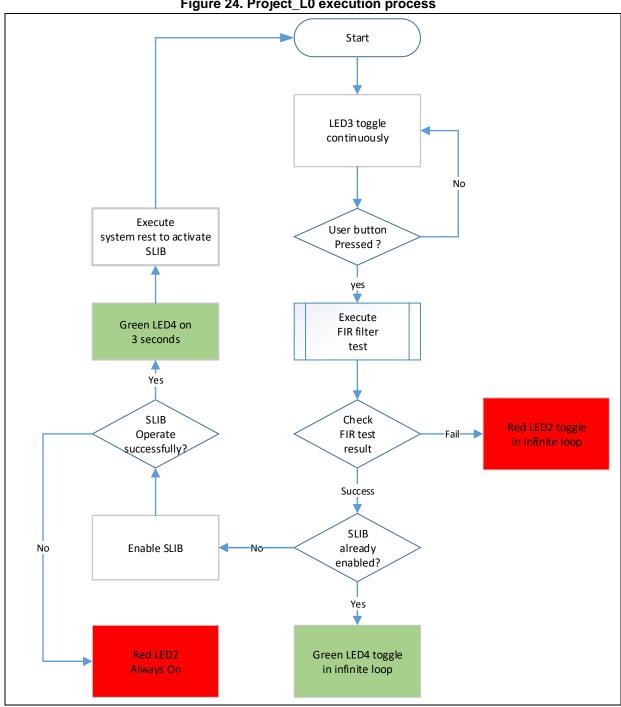


Figure 24. Project_L0 execution process

Go through the following steps to execute this example program:

- (1) Use Keil® µvision to open the Project_L0 under \utilities\at32f413_slib_demo\project_I0\mdk_v5\, and then compile;
- (2) Before downloading the code, check whether the chip on AT-START-F413 board is SLIBprotected 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 operation of low-pass filter;
- (5) Compare the computation result. If it is correct, the green LED4 keeps blinking; otherwise, the

red 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, go 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

Operate 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

Device Target Output Listing User C/C++ Use Memory Layout from Target Dialog Make RW Sections Position Independent Make RO Sections Position Independent Don't Search Standard Libraries	Asm Linker Debug Utilities ▲/O Base: ▲ B/O Base: ↓ R/W Base ↓	
Scatter File	gisable Warnings:	Edit
Miscsymdefs=fir_filter_symbol.bt controls Linkercpu Cortex-M4.fp *.o -critrollibrary_type=microlibstrictscatter ".\slib- string	www.sct"	4
	Cancel Defaults	Help

- After compiling the project, a symbol definition file named "fir_filter_symbol.txt" is generated under project_I0\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. Contents of modified fir_filter_symbol.txt

0x08001001 T FIR_lowpass_filter



Use IAR to generate symbol definition file

Operate as follows:

● Select Project→Option→Build Actions

Figure 27. Set Build Actions in IAR

Options for node "project, Category: General Options Static Analysis Runtime Checking C/C++ Compiler Assembler Output Converter Custom Build	D. Build Actions Configuration Pre-build command line:
Build Actions Linker Debugger Simulator CADI CMSIS DAP GDB Server I-jet/JTAGjet J-Link/J-Trace TI Stellaris Nu-Link PE micro	Post build command line: \$TOOLKIT_DIR\$\bin\isymexport.exeedit "\$PROJ_DIR\$\st.
ST-LINK Third-Party Driver TI MSP-FET TI XDS	OK Cancel

- 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_I0\iar_v8.2, which is used to select function symbols to be generated. Users can manually edit according to the contents called by sLib. As shown in Figure 28, the "show" is the command used to select functions.

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 AT32F413 MCU main Flash memory and 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 function;

• Develop and debug user's program.

Notes:

Project_L1 must use the same toolchain and the same version of the compiler as that 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 V5.18.0.0; 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, as shown in Figure 13. In the main Flash memory, sector 2~ sector 5 are occupied by the security library, and end users need to isolate this region by using linker control file to avoid code being compiled to this region.

Keil® µvision: scatter file

Refer to the *end_user_code.sct* file 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 after 0x20003000 needs to be reserved for the RAM area, 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 0x00003000 { ; RW data
   . ANY (+RW +ZI)
  0x20003000 \sim 0x20003FFF RAM reserved for SLIB code
; 0x08001000 \sim 0x08002FFF is SLIB area
LR IROM2 0x08003000 0x0003D000
                               {
                                     ; load region size_region
  ER IROM2 0x08003000 0x0003D000 { ; load address = execution address
   . ANY (+RO)
}
```

IAR: ICF file

Refer to the *enduser.icf* file under project_I1\iar_V8.2\, as shown in Figure 30.

Figure 30. Modified icf file

<pre>/* Reserved SLIB area */ define region ROM_region</pre>	<pre>= mem:[fromICFEDIT_region_ROM_start toICFEDIT_region_ROM_end_] -mem:[fromICFEDIT_region_SLIB_start_ toICFEDIT_region_SLIB_end_];</pre>
define region RAM_region	<pre>= mem:[fromICFEDIT_region_RAM_start toICFEDIT_region_RAM_end] - mem:[fromICFEDIT_region_SLIB_RAM_start toICFEDIT_region_SLIB_RAM_end];</pre>

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.



Project_I1	F
😑 ᇶ at_start_f407	
🕀 💼 user	
🕀 🧰 bsp	
🕀 🧰 firmware	
🕀 🧰 cmsis	
🛱 🔂 🔂 🖓	
fir_filter_symbol.txt	
🗈 🛄 readme	

After adding this file to fir_filter, modify its file type from "text" to "Object", as shown in Figure 32.

Options for File 'fir_filter_symbol.txt'	X
Properties	
	✓ Include in Target Build ✓ Always Build
last change: Fri May 21 11:14:16 2021	Generate Assembler SRC File
	Assemble SRC File Image File Compression
Custom Arguments:	
Memory Assignment: Code / Const: Zero Initialized Data: Other Data: Code fault> V	
OK Cancel Def	faults Help

Figure 32. Modify symbol definition file type to "Object file"

Add symbol definition file in IAR

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



riies	¥	•
🗆 🌒 project_l1 - at_start_f	~	
⊨⊞ 🛋 bsp		•
⊨ 🖬 🛋 cmsis		•
┝──── ── 「 fir_filter └────		
├ 🗹 📹 firmware		•
⊣⊕ 🛋 readme		
⊨ 🖬 🛋 user		٠
└─⊞ 📹 Output		

Figure 33. Add symbol definition file in IAR

3.5.3 Call functions in SLIB-protected area

When 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 containing input signal data table;
- *outputF32*: pointer storing 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 will keep blinking;
- Press the USER button on AT-START board, and the FIR_lowpass_filter() starts operation;
- If the result is correct, the green LED4 will keep blinking; otherwise, the red LED2 will keep blinking.

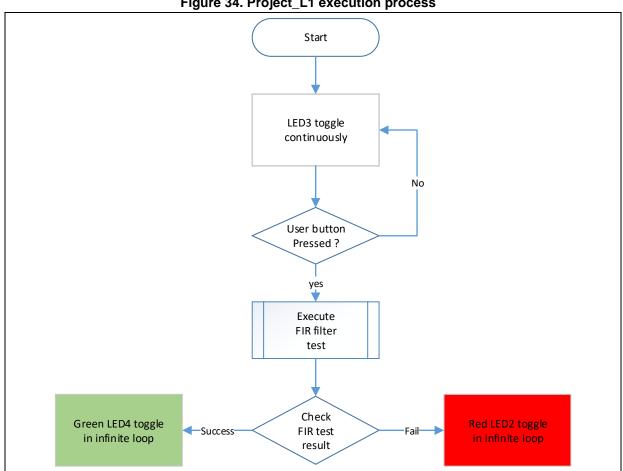


Figure 34. Project_L1 execution process

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 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.



] 🔯 🖬 📰 🅃 Disassembly		• 📑 • 🔜 • 💷						д 🔀
	0x08003E52	4770 AT32_Board	BX L_Init();	lr					÷ 🛆
	0x08003E54	2804	e Flash to ge CMP	r0,#C		Mixed Mode Assembly Mode		error occur */	
	0x08003E56 97: 1 98:	D106 Enable_Fla	BNE sh_INT();	0x080		Address Range	•		
	99:					Show Disassembly at Address			
	100: , 0x08003E58 0x08003E5A	490A	LDR LDR	to be p r1,[p r1,[r	-0	Set Program Counter Run to Cursor line	Ctrl+F10		- 1
	101: 102: 0x08003E5C	(_BUTTON_State			Insert/Remove Breakpoint Enable/Disable Breakpoint	Ctrl+F9		
	0x08003E5C 0x08003E60 0x08003E62 104:	4108	LDR STR	r1,r1 r2,[p r1,[r		Insert Tracepoint at '0x08003E54'. Enable/Disable Tracepoint			
	105: 106: 107:) –	Off LED3 */			Inline Assembly			
	0x08003E64 0x08003E66	E005 4907	B LDR	0x080 r1,[p		Load Hex or Object file Instruction Trace	•		
	<	2000	וחחז	•1 F•		Execution Profiling	+		>
1	main.c				1	Insert/Remove Bookmark	Ctrl+F2		▼ ×
			д 🛛	Call Stac		Conv	Ctrl+C		д 🛛

Figure 35. Enter Show Disassembly at Address

Enter the address "0x08001000" of SLIB_INSTRUCTION start sector (sector 2);

Figure 36. Set Show Code at Address

Show Code at Address	×
Address: 0x08001000	Go To

• As shown in Figure 37, codes from 0x08001000 are all 0xFFFFFFF;

Figure 37. View codes

Disassembly	д
76: {	
77: uint32 t i;	
0x08001000 FFFFFFFF DCD	OxFFFFFFF
0x08001004 FFFFFFFF DCD	OxFFFFFFF
0x08001008 FFFFFFFF DCD	OxFFFFFFFF
79: uint32_t num	Blocks = testlengthsamples/BLOCK_SIZE;
80: arm_fir_instance_f	32 5;
81:	
82:	
	unction to initialize the instance structure. */
	OxFFFFFFF
84: arm_fir_init_f32(&	S, NUM_TAPS, (float32_t *)&firCoeffs32[0], &firStateF32[0], blockSize
85: /*	
	ocess function for every blockSize samples */
0x08001010 FFFFFFF DCD	
0x08001014 FFFFFFF DCD	
0x08001018 FFFFFFF DCD	OXFFFFFFF
	OXFFFFFFF
0x08001020 FFFFFFFF DCD	OxFFFFFFFF
0x08001024 FFFFFFFF DCD	OxFFFFFFFF
0x08001028 FFFFFFFF DCD	OxFFFFFFF
0x0800102C FFFFFFFF DCD	OxFFFFFFF
0x08001030 FFFFFFFF DCD	OxFFFFFFFF
0x08001034 FFFFFFFF DCD	0×FFFFFFF

 Similarly, enter address 0x08001000 in "Memory" window, and codes are all 0xFF, as shown in Figure 38;



Figure 38. View codes in Memory

Memory 1																				д	x
Address:	0x080	0100)																		^
0x0800	1000:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	
0x0800	1013:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	
0x0800	1026:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	
0x0800	1039:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	$\mathbf{F}\mathbf{F}$	FF	FF	
0x0800	104C:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	
0x0800	105F:	FF	FF	FF	$\mathbf{F}\mathbf{F}$	FF	FF	$\mathbf{F}\mathbf{F}$	$\mathbf{F}\mathbf{F}$	FF	$\mathbf{F}\mathbf{F}$	FF	FF	FF	FF	$\mathbf{F}\mathbf{F}$	FF	$\mathbf{F}\mathbf{F}$	$\mathbf{F}\mathbf{F}$	FF	
0x0800	1072:	FF	FF	FF	FF	FF	FF	FF	FF	FF	$\mathbf{F}\mathbf{F}$	FF	FF	FF	FF	FF	FF	FF	$\mathbf{F}\mathbf{F}$	FF	
0x0800	1085:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	$\mathbf{F}\mathbf{F}$	FF	FF	
0x0800	1098:	F F	শ শ_		ዋዋ	न न	FF	ፑፑ	नन	न न	FF	ዋ ዋ	F F	FF	ፑፑ	नन	न न	FF	FF	FF	~
Call St	tack + L	ocals		Mei	mory	1	M	emo	ry 2												
			_			_															

In the "Memory" window, enter the address 0x08002000 of SLIB_DATA starting sector (sector 4); this region is allowed to be read through D-Code bus, so original values can be found, as shown in Figure 39;

Memory 1					д	x
Address: 0x08002000						^
0x08002000: B9 E	1 EE BA 1	12 22 DO	BA 00 00 00	00 F7 55 72	3B CF 4E 04	
0x08002013: 3C 5	8 C2 OB 3	3C 00 00	00 80 9E 85	8E BC 88 DC	OB BD 9C A3	
0x08002026: 08 E	D 00 00 0	AO 00 00	82 8A 3D FO	DB 1B 3E 5F	46 64 3E 06	
0x08002039: 41 8	O 3E 5F 4	46 64 3E	FO DB 1B 3E	OA 82 8A 3D	00 00 00 00	
0x0800204C: 9C A	3 08 BD 8	88 DC OB	BD 9E 85 8E	BC 00 00 00	80 58 C2 OB	
0x0800205F: 3C C	F 4E 04 3	3C F7 55	72 3B 00 00	00 00 12 22	DO BA B9 E1	
0x08002072: EE E	A FF FF H	FF FF FF	FF FF FF FF	FF FF FF FF	FF FF FF FF	
0x08002085: FF F	FFFFF	FF FF FF	FF FF FF FF	FF FF FF FF	FF FF FF FF	
<u>0x08002098: FF F</u>	न नन नन न	नन नन नग	नन नन नन नन	नन नन नन नन	नन नन नन नन	~
Call Stack + Locals	Memory 1	III Memor	ry 2			

Figure 39. View SLIB_DATA start sector in Memory

 In the "Memory" window, double click to modify the value of 0x08002000, and a warning pops up by setting EPPERR=1 in the FLASH_STS register, indicating the protection is enabled;

Propert	у	Value	
PSR		0x0000030	
	.OCK	0	
	_UNLOCK	0	
⊟ STS		0x00000010	
4	ODF		
	EPPERR	v	
	PRGMERR		
	OBF		

Figure 40. SLIB write test

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



	rigure 41. Write protection error interrupt
	115 void FLASH_IRQHandler(void) 116 ₽ {
≫	<pre>117 if(flash_flag_get(FLASH_EPPERR_FLAG)) 118 □ {</pre>
	119 flash_flag_clear(FLASH_EPPERR_FLAG); 120 delay_ms(500);
	121 - } 122 }

Figure 41. Write protection error interrupt



4 Integrate codes and download

After codes of the solution provider and end user are well designed, 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, and then click "File-Save Flash data as" to save the codes as BIN or HEX file. In this example, it is named "slib.bin", as shown in Figure 42.

Save flash data as W: V1.5.8 Make encryption file ## 特 力 Exit FLASH_DA 0x 0 @ Remap0 (Use PA11/PA12 pins) Type Select Remap1 (Use PB10/PB11 pins) Memory read settings Address 0x 08001000 Read size 0x 2000 Data bits 8 bits Read File info	Save file a	IS	XT	32F41	3RC1	F7	FlashSi	ze: 2	56KB			2	ΈζΥ
Exit FLASH_DA 0x 0 Remap0 (Use PA11/PA12 pins) Type Select Remap1 (Use PB10/PB11 pins) Memory read settings Address 0x 08001000 Read size 0x 2000 Data bits 8 bits Read File info Image: Select_lo.hex File size Address range(0x) Add 1 project_l0.hex 8484 08000000-080003B7,08001000-08001117,08 Delete Image: File info Image: Select_lo.hex Image: Select_lo.hex Flash CRC File CRC verify DownLoad Flash info File:project_l0.hex Address range: Select_lo.hex Select_	Save flash	data as	w	: V1.5.	8								
Type Select Remap1 (Use PB10/PB11 pins) Memory read settings Address 0x 08001000 Read size 0x 2000 Data bits 8 bits • Read File info Image: Contract of the size in the size i	Make enc	ryption file	68	65120	040F1	.58 091 7	79C02				躮	ÉΊ	侍 力
Memory read settings Address 0x 08001000 Read size 0x 2000 Data bits 8 bits Read File info Image: Setting Se	Exit		F	LASH_	DA 0	c 🗌	0	@ R	emap	0 (Use	PA11/P	A12 pi	ins)
Address 0x 08001000 Read size 0x 2000 Data bits 8 bits Read File info No. File name File size Address range(0x) 1 project_I0.hex 8484 08000000-080003B7,08001000-08001117,08 Delete (III) Flash CRC File CRC verify DownLoad Flash info File:project_I0.hex Address range:[0x08000000 0x080003B7] Address range:[0x08000000 0x0800117] Address range:[0x08000000 0x08003B7] Address range:[0x08000000 0x08003B7] Address range:[0x08000000 0x08003B7]		Гуре				Se	lect	© R	emap	1 (Use	PB10/P	B11 pi	ins)
File info No. File name File size Address range(0x) Add 1 project_l0.hex 8484 08000000-080003B7,08001000-08001117,08 Delete	Memory read	settings											
File info No. File name File size Address range(0x) Add 1 project_l0.hex 8484 08000000-080003B7,08001000-08001117,08 Delete	-		Read	size	0x 20	000		Data k	oits 🖡	8 bits	•		Read
No. File name File size Address range(0x) Add 1 project_l0.hex 8484 08000000-080003B7,08001000-08001117,08 Delete	Address VA C		Read	5120	0 2				1113	0 0110			
I project_l0.hex 8484 0800000-080003B7,08001000-08001117,08 Delete < III + Flash CRC File CRC verify DownLoad Flash info File:project_l0.hex Address range:[0x08000000 0x080003B7] Address range:[0x08001000 0x08001117] Address range:[0x08002000 0x080020731 Address range:[0x08003000 0x08004BDF1 checksum: 0x0000612F	File info												_
Flash info File:project_l0.hex Address range:[0x08000000 0x080003B7] Address range:[0x08001000 0x08001117] Address range:[0x08000000 0x080003B7] Address range:[0x08001000 0x08001117]	No. File na	ime		File	size	Add	ress rar	nge(0x)				Add
Flash CRC File CRC verify DownLoad Flash info File:project_l0.hex Address range:[0x08000000 0x080003B7] Address range:[0x08001000 0x08001117] Address range:[0x08002000 0x080020731 Address range:[0x08003000 0x08004BDF1 checksum: 0x000C612F Checksum: 0x000C612F						0000		00000	07.00	00100/		447.00	
Flash CRC File CRC verify DownLoad Flash info File:project_l0.hex Address range:[0x08000000 0x080003B7] Address range:[0x08001000 0x08001117] Address range:[0x08002000 0x080020731 Address range:[0x08003000 0x08004BDF1 checksum: 0x000C612F Checksum: 0x000C612F	1 projec	t_l0.hex		848	4	0800	0-0000	80003	87,08	001000	-08001	117,08	^B Delete
Flash CRC File CRC verify DownLoad Flash info File:project_l0.hex Address range:[0x08000000 0x080003B7] Address range:[0x08001000 0x08001117] Address range:[0x08002000 0x080020731 Address range:[0x08003000 0x08004BDF1 checksum: 0x000C612F Checksum: 0x000C612F	1 projec	t_l0.hex		848	4	0800	00000-0	80003	87,08	001000	-08001	117,08	⁸ Delete
Flash info File:project_l0.hex Address range:[0x08000000 0x080003B7] Address range:[0x08001000 0x08001117] 0x080020731 Address range:[0x08003000 0x08004BDF1	1 projec	ct_10.hex		848	4	0800	0000-0	80003	87,08	001000	-08001	117,08	³ Delete
Flash info File:project_l0.hex Address range:[0x08000000 0x080003B7] Address range:[0x08001000 0x08001117] 0x080020731 Address range:[0x08003000 0x08004BDF1 checksum: 0x0000C612F		ct_I0.hex		848	4	0800	0000-0	80003	87,08	001000	-08001	117,08	Delete
Address range:[0x08000000 0x080003B7] Address range:[0x08001000 0x08001117] Address range:[0x08002000 0x08002073] Address range:[0x08003000 0x08004BDF] checksum: 0x000C612F		ct_l0.hex	m	848	4							4	
0x080020731 Address range: [0x08003000 0x08004BDF1 checksum: 0x000C612F				848	4							4	
Address 0 1 2 3 4 5 6 7 8 9 A B C D E F A	Flash info File	:project_l0.ł	hex				Flash CR	C	File	e CRC v	erify	Do	ownLoad
	Flash info File	:project_10.1	hex 0x080003B	7] Ad	dress r	ange:[0:	Flash CR x0800100	.C	File	e CRC v	erify	Do	ownLoad
	Flash info File Address range: 0x080020731	:project_l0.ł (0x08000000 Address rano	hex 0x080003B e:[0x08003(7] Ad	dress r 8004B	Fange:[0: DF1_ch	Flash CR x0800100 necksum:	C	File 000111 C612F	e CRC v 7] Add	erify Iress ran	Do	wnLoad

Figure 42. Save SLIB codes

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), respectively, as shown in Figure 43.

Figure 43. G	Generate	bin file	of SL	B code
--------------	----------	----------	-------	--------

Command Items	User Command	Stop on Exi S	i
Before Compile C/C++ File			
- 🗌 Run #1		对 Not Specified	
🗌 Run #2		📔 Not Specified 🗌	
Before Build/Rebuild			ER_IROM1
- 🗌 Run #1		对 Not Specified	
🗌 Run #2		对 Not Specified	ER_IROM2
After Build/Rebuild			
🔽 Run #1	fromelf.exebinoutput .\Listings\@L.bin !L	对 Not Specified	ER_SLIB_DATA
Run #2		对 Not Specified 🗖	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.

Disconnect	Part Numb	er: AT	32F41	3RCT7	Flash	Size: 2	56KB		.47	56=	YΣ	Lib status: Disable Remaining usage times: 256
Disconnect	AT-Link-EZ	FW	: V1.5.8	8								inable password 0x 55665566 sLib position: Main Flash
AT-Link 🔻	AT-Link SN:	AB686	551200	40F1580	9179C02				雅	特	力	Disable password 0x Start sector Sector20x08001000
	SPIM	F	LASH_D	X0 AC	0	0	Remap0 (Use PA1	1/PA1	2 pins)		Disable sLib DATA start sector Sector40x08002000
	Туре				Select		Remap1					End sector Sector50x08002800
Memory rea												Extra options
Address 0x	-	Read	size (0000 x	0074	Data	bits 8 l	oits 🔻	ſ	Rea	ad	Mass erase for Main Flash
	00001000	neau	3120 0		5074	Data			l			Verify Custom encryption key for verify: Verify
ile info												Castom encryption key for verify. Endbe set
	name				File	size	Addres	s range	0x)		Add	
-	SLIB_INSTRUCT	ION.bin			280)		00-0800			Delete	Jump to the user program Enable FAP after download
2 ER_S	SLIB_DATA.bin				110	5	080020	00-0800	2073			Write software serial number(SN)
												Button free mode
												Write address 0x 08010000
												Write address 0x 08010000
					Flash	CRC	File C	RC verif	y] [Down	Load	Write address UX 08010000 Current SN 0x 00000001
Flach info Fi	ie:FR SI IB DAT	A.bin		[Flash	CRC	File C	RC verif	y [Down	Load	Write address 0x 08010000
	ile:ER_SLIB_DA1 e:[0x08002000 0>] chec	cksum: 0x		CRC	File C	RC verif	y [Down	Load	Write address UX 0801000 Current SN 0x Increase step 0x 00000001
Address range	e:[0x08002000 0>	<08002073	-		00002C29							Write address UX 08010000 Current SN 0X Increase step 0X 00000001 Increase step 0X 00000001
Address range	e:[0x08002000 0x	x08002073	4	5 6	00002C29	8 9	AE	С	D	E F	A:	Write address UX 08010000 Current SN 0x 00000001 Increase step 0x 00000001 Write user system data User system data file path
Address range Address 0x08002000	e:[0x08002000 0x 0 1 B9 E1	x08002073	4		00002C29 7 BA C		AE	C F7	D 55 7	E F 72 3B	A: _	Write address UX 08010000 Current SN 0X Increase step 0X 00000001 Increase step 0X 00000001
Address range Address 0x08002000 0x08002010	e:[0x08002000 0x 0 1 B9 E1 CF 4E	08002073 2 3 EE BA	4 12 58	5 6 22 D0	00002C29	8 9 0 00	A E	C F7 9E	D 55 7 85 8	E F 72 3B	A: ^ 贯i 螺: ==	Write address UX 08010000 Current SN 0x 00000001 Increase step 0x 00000001 Write user system data User system data file path
Address range	e:[0x08002000 0> 0 1 B9 E1 CF 4E 88 DC	x08002073 2 3 EE BA 04 3C	4 12 58 9C	5 6 22 D0 C2 0B	000002C29 7 3 8A 0 3C 0 8D 0	8 9 0 00 0 00	A E 00 00	C F7 9 9E 0 0A	D 55 7 85 8 82 8	E F 72 3B 8E BC	A:	Write address UX 08010000 Current SN 0x 00000001 Increase step 0x 00000001 Write user system data User system data file path

Figure 44. Online MCU programming in ICP

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



T-Link settings AT-Link offline config se				
Offline project		▼ Delete		Creat
Project name slib_download	C	Device AT32F413	▼ AT32F413RCT7	•
No. File name	File size	Address range(0x) Storage loca	. Add
1 ER_SLIB_INSTRUCTION.bin	280	08001000-080011	17	Delete
2 ER_SLIB_DATA.bin	116	08002000-080020	73	
•			Þ	,
Erase option Erase the sectors of file	e size	•		
Download times				_
		Download interf	CA CWD	-
	Marile.	Download interfa		•
Encryption transmit	Verify	Download interfa		•
	Verify			•
Encryption transmit	Verify			•
Encryption transmit Write user system data	Verify			•
 Encryption transmit Write user system data Enable FAP after download 		Reset and run		•
Encryption transmit Krite user system data Enable FAP after download				•
 Encryption transmit Write user system data Enable FAP after download 		Reset and run		•
 Encryption transmit Write user system data Enable FAP after download Software serial number(SN) SPIM series 	settings s	Reset and run	Main Flash	
 Encryption transmit Write user system data Enable FAP after download Software serial number(SN) SPIM s Enable sLib sLib enable password 0x 556655 	settings s	Elib settings Start sector	Main Flash Sector20x08001000	-
 Encryption transmit Write user system data Enable FAP after download Software serial number(SN) SPIM s Enable sLib sLib enable password 0x 556655 Disable sLib before download 	settings s	Reset and run ELib settings Start sector DATA start sector	Main Flash Sector20x08001000 Sector40x08002000	*
 Encryption transmit Write user system data Enable FAP after download Software serial number(SN) SPIM s Enable sLib sLib enable password 0x 556655 	settings s	Elib settings Start sector	Main Flash Sector20x08001000	*
 Encryption transmit Write user system data Enable FAP after download Software serial number(SN) SPIM s Enable sLib sLib enable password 0x 556655 Disable sLib before download 	settings s	Reset and run ELib settings Start sector DATA start sector	Main Flash Sector20x08001000 Sector40x08002000	*
 Encryption transmit Write user system data Enable FAP after download Software serial number(SN) SPIM s Enable sLib sLib enable password 0x 556655 Disable sLib before download 	settings s	Reset and run	Main Flash Sector20x08001000 Sector40x08002000	
 Encryption transmit Write user system data Enable FAP after download Software serial number(SN) SPIM s Enable sLib sLib enable password 0x 556655 Disable sLib before download 	settings s	Reset and run	Main Flash Sector20x08001000 Sector40x08002000	· · ·
 Encryption transmit Write user system data Enable FAP after download Software serial number(SN) SPIM s Enable sLib sLib enable password 0x 556655 Disable sLib before download 	settings s	Reset and run	Main Flash Sector20x08001000 Sector40x08002000	· · ·
 Encryption transmit Write user system data Enable FAP after download Software serial number(SN) SPIM s Enable sLib sLib enable password 0x 556655 Disable sLib before download 	settings s	Reset and run	Main Flash Sector20x08001000 Sector40x08002000	· · ·

Figure 45.Offline programming to MCU via AT-Link

(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.



ile J-Link settings AT-Link settings Target Language Help isconnect Part Number: AT32F413RCT7 FlashSize: 256KB																		sLib status										
Disconnect	Part I	lum	ber:	АТЗ	2F43	L3RC	17	Fla	shSia	e: 2	56KB			1	2	Γ-	2	sLib status: Enable Remaining usage times: 255										
, sconneet	AT-Link-EZ FW: V1.5.8								Enable password 0x 55665566 sLib posit					ion:	Main Fla		×											
T-Link 👻	AT-Li	nk SN	I: A	B686	5120	040F1	15809	1790	02					焩	È :	特	ノ	Disable	passwore	d 0x			Star	rt secto	or	Sector2-	-0x0800100	0 •
	SPI	м		FL	ASH	DA 0	×	0		@ R	emap		e PA	11/P/	– A12 r						Disable	sLib	DAT	'A star	t sector		-0x0800200	-
	Туре							Select			emap												End	secto	r	Sector5-	-0x0800280	• 0
Memory rea		as																Extra op	tions									
Address 0x						0x 0	0000	110			oits 🛛	0 6.4				Rea	d		erase for	Main	Flach			_	🖂 Die	able stib	before dov	mload
Address 0x	080010	00		lead	size	UX U	5000.	110		ata t	ins [o Dite	, ,			ittee		Veri								able sLib		moau
File info																		veri	ту	Custo	m encry	ption ke	ey tor ve	erity:				
No. File	name				File	size	A	ddres	s ran	ge(0x)						Add								Dis	sable FAP I	before dow	nload
1 proj	ect_l1.he	×			700	8	08	30000	00-08	0003	A7,08	0030	00-0	8004	7B7	D	elete	📃 Jum	p to the	user p	rogram				📃 Ena	able FAP a	fter downlo	ad
					_		_											🔲 Writ	te softwa	re ser	al numb	per(SN)						
																		W	/rite addr	ress 0:	080	010000			📃 But	tton free n	node	
•																·			urrent SN	J 01	000	000001						
							ſ	Flas	h CR	:	File	e CRO	C veri	ify	D	ownL	.oad											
Flash info Fi	le:proje	:t_ 1.h	iex												_				icrease st	tep U	000	000001						
Address range	:[0x0800	0000 0	0x0800	003A7] Ad	dress	range	:[0x08	00300	0 0x08	0047E	7] c	heck	sum: (0x000	A2A8:	1	🔲 Writ	te user sy	ystem	data							
	0	1	2	3	4	5	6	7	8	9	A	в	С	D	E	F	A	Use	r system	data	ile path							
Address	38	13	00	20	01	30	00	08	F7	35	00	08	EF	35	00	08	80											
	ES	35	00	08	CF	35	00	08	75	36	00	08	00	00	00	00	2.4											
0x08000000	Fo	00	00	00	00	00	00	00	00	00	00	00	FB	35	00	08												
0x08000000 0x08000010	00			08	00	00	00	00	F9	35	00	08	FD	35	00	08	?					_		_				
Address 0x08000000 0x08000010 0x08000020 0x08000030		35	00	00			-																					

Figure 46. End users program codes to MCU

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 to an encrypted slib.benc 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.



AT-Link settings AT-Link offline config	settings AT-Link offline download status	
Offline project	Delete Creat	
Project name slib_project No. File name 1 ER_SLIB_INSTRUCTION.bin 2 ER_SLIB_DATA.bin Erase option Erase the sectors of f Download times Encryption transmit Write user system data Enable FAP after download Software serial number(SN)	Download interface SWD Verify Reset and run	
Enable sLib sLib enable password 0x 55665	sLib position Main Flash 🔹	The AT-Link project file settings

Figure 47. Set offline project

(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 other change settings when adding code file to the offline project, which requires the solution provider to configure the final settings in advance.



ffline project		▼ Delete	Crea	at
Project name slib_project	D	evice AT32F413	▼ AT32F413RCT7	-
No. File name	File size	Address range(0x) Storage loca ^ A	Add
2 ER_SLIB_DATA.bin	116	08002000-080020		elete
3 project_l1.hex	936	0800000-080003		
1 project II boy	III		▶	
Erase option Erase the sector	s of file size			
Download times 0		Download interfa	ace SWD -	
	✓ Verify	Reset and rur	1	
Encryption transmit Write user system data	✓ Verify	Reset and rur	, 	
	√ Verify	Reset and rur		
Write user system data	✓ Verify	☐ Reset and rur	1	
 Write user system data Enable FAP after download 				
Write user system data		Reset and rur	1	
Write user system data		ib settings	Main Flash	
Write user system data Enable FAP after download Software serial number(SN)		ib settings	Main Flash	••••
Write user system data Enable FAP after download Software serial number(SN) ✓ Enable sLib sLib enable password 0x	SPIM settings SL	ib settings sLib position Start sector	Main Flash	•
Write user system data Enable FAP after download Software serial number(SN) ✓ Enable sLib sLib enable password 0x Disable sLib before download	SPIM settings SL	ib settings	Main Flash	
Write user system data Enable FAP after download Software serial number(SN) Enable sLib sLib enable password 0x	SPIM settings SL	ib settings sLib position Start sector	Main Flash	
Write user system data Enable FAP after download Software serial number(SN) ✓ Enable sLib sLib enable password 0x Disable sLib before download	SPIM settings SL	ib settings sLib position Start sector DATA start sector End sector	Main Flash Sector20x08001000 Sector40x08002000 Sector50x08002800	
 Write user system data Enable FAP after download Software serial number(SN) Enable sLib sLib enable password 0x Disable sLib before download 	SPIM settings SL	ib settings sLib position Start sector DATA start sector	Main Flash Sector20x08001000 Sector40x08002000 Sector50x08002800	••••••••••••••••••••••••••••••••••••••
 Write user system data Enable FAP after download Software serial number(SN) Enable sLib sLib enable password 0x Disable sLib before download 	SPIM settings SL	ib settings sLib position Start sector DATA start sector End sector	Main Flash Sector20x08001000 Sector40x08002000 Sector50x08002800	••••••
Write user system data Enable FAP after download Software serial number(SN) ✓ Enable sLib sLib enable password 0x Disable sLib before download	SPIM settings SL	ib settings sLib position Start sector DATA start sector End sector	Main Flash Sector20x08001000 Sector40x08002000 Sector50x08002800	·····
Write user system data Enable FAP after download Software serial number(SN) ✓ Enable sLib sLib enable password 0x Disable sLib before download	SPIM settings Sl	ib settings sLib position Start sector DATA start sector End sector Load parar	Main Flash Sector20x08001000 Sector40x08002000 Sector50x08002800 meters Save paramet	·····

Figure 48. Add project file



5 Revision history

Table 2. Document revision history

Date	Version	Revision note
2021.11.16	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, and ARTERY assumes no liability whatsoever relating to the choice, selection or use of the ARTERY products and services described herein.

No license, express or implied, to any intellectual property rights is granted under this document. If any part of this document deals with any third party products or services, it shall not be deemed a license grant by ARTERY for the use of such third party products or services, or any intellectual property contained therein, or considered as a warranty regarding the use in any manner whatsoever of such third party products or services or services or any intellectual property contained therein.

Unless otherwise specified in ARTERY's terms and conditions of sale, ARTERY provides no warranties, express or implied, regarding the use and/or sale of ARTERY products, including but not limited to any implied warranties of merchantability, fitness for a particular purpose (and their equivalents under the laws of any jurisdiction), or infringement of any patent, copyright or other intellectual property right.

Purchasers hereby agrees that ARTERY's products are not designed or authorized for use in: (A) any application with special requirements of safety such as life support and active implantable device, or system with functional safety requirements; (B) any air craft application; (C) any automotive application or environment; (D) any space application or environment, and/or (E) any weapon application. Purchasers' unauthorized use of them in the aforementioned applications, even if with a written notice, is solely at purchasers' risk, and is solely responsible for meeting all legal and regulatory requirement in such use.

Resale of ARTERY products with provisions different from the statements and/or technical features stated in this document shall immediately void any warranty grant by ARTERY for ARTERY products or services described herein and shall not create or expand in any manner whatsoever, any liability of ARTERY.

© 2021 Artery Technology -All rights reserved