AN0027
Application Note

Setup SLIB with Eclipse and GCC

Introduction

This document describes how to debug AT32 MCU through Eclipse and how to configure security library (sLib) with an application case.

AT32F403A is used as an example in this guideline. For more information on security library (sLib) of AT32F403A, refer to the *AT32F403A Security Library Application Note*.

Applicable products:

Model	AT32 series

Contents

List of tables

List of figures

1 Overview

This document describes how to configure and debug security library (sLib) embedded in the AT32F403A series through Eclipse, ARM-GCC compiler, GNU-ARM add-on, J-Link, AT-Link and other resources. Project_L0 and Project_L1 example cases are provided for sLib solution providers and end users, respectively.

For more information on sLib of the AT32F403A, refer to the document: *AN0040_AT32F403A_407_Security_Library_Application_Note*.

Environment requirements

The installation described in this document is done based on WINDOWS 7 x64 system, and the evaluation board AT-START-F403A.

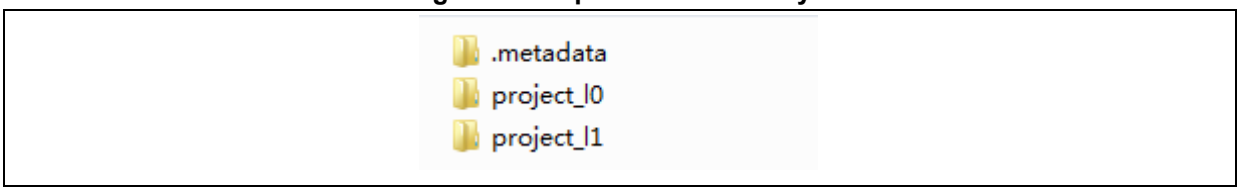
AN0033_Eclipse_with_GCC document: gives a description of how to install Eclipse debugging environment and set up Eclipse project.

All software used in this document are packed in *Setup_SLIB_with_Eclipse_and_GCC_V2.0.0.zip*. Just unzip this file, compile and run it.

The workspace of Eclipse, after unzipping above-mentioned file, can be found in *Setup_SLIB_with_Eclipse_and_GCC_V2.0.0\utilities\slib_with_eclipse_and_gcc_demo.*

It includes the following content:

Figure 1. Eclipse work directory



.metadata: workspace environment configuration

project_l0: algorithm development example for solution providers

project_l1: application case for end users

2 Project_L0 example case

Project_L0 example case includes the following operations:

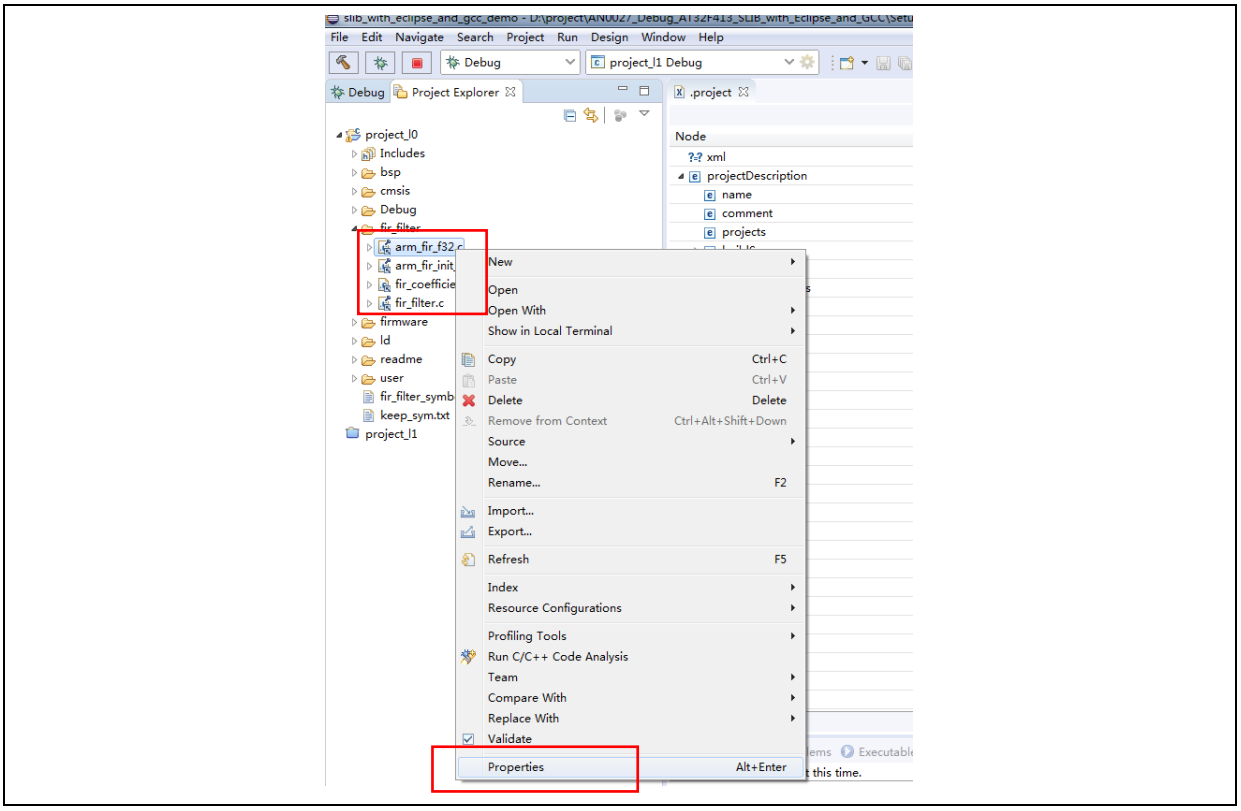
- Compile *low-pass filter* function into Execute-only code
- Place *low-pass filter* function code in Flash address 0x08004000 ~0x08004FFF(sector 8~9)
- Place *low-pass filter* coefficient in Flash address 0x08005000 ~ 0x08005FFF(sector 10~11)
- After successful verification, set sector 8~9 as instruction sLib, and sector 10~11 as data sLib, which can be done by calling *slib_enable()* function in main.c, or by using Artery ICP Programmer (ICP tool is recommended)
- Generate head files and symbol definition files that are used to call low-pass filter function by end-user applications.

2.1 How to generate Execute-only code?

To generate Execute-only code, the following procedures are recommended:

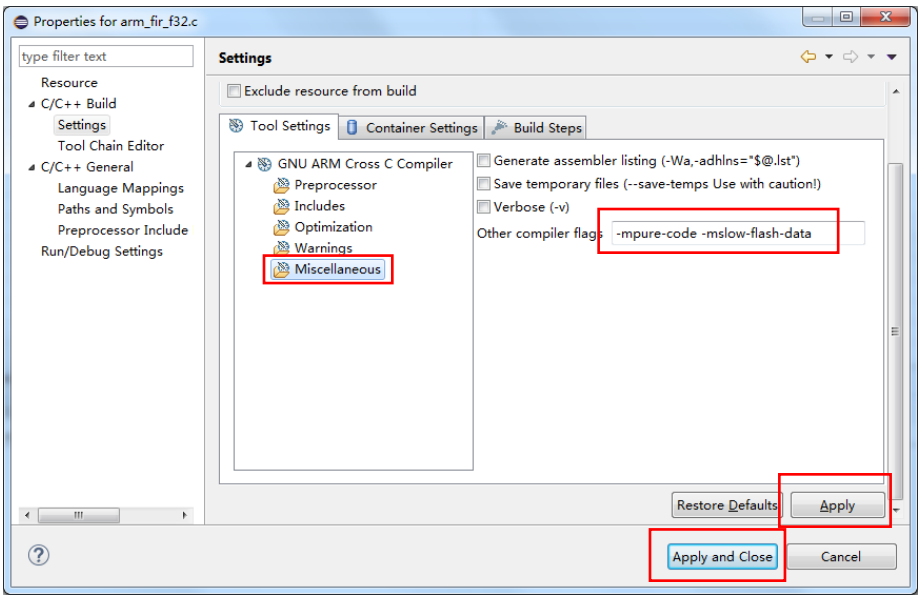
1) Select C file group or a separate C file. The example below stores the desired C files in *FIR_Filter* group. Click a file which you want to set as Execute-only from it, and right click and select "*Properties*"

Figure 2. Configure C files



2) Click *C/C++ Build->Settings->GNU ARM Cross C Complier->Miscellaneous*, and go to *Other compiler flags* where you need to input two key words: -mpure-code, -mslow-flash-data, and then click on *Apply*.

Figure 3. Miscellaneous settings

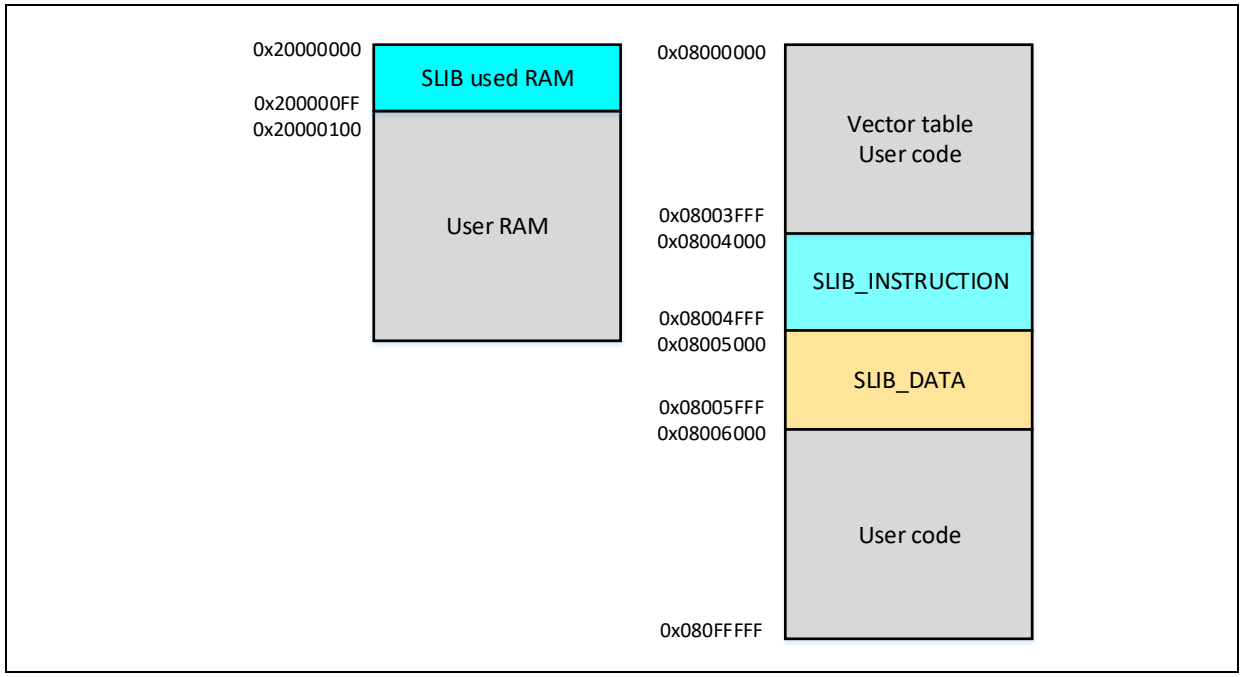


2.2 Security library address arrangement

Figure 4 presents the main Flash mapping and RAM area division used in Project_L0 example case.

The RAM area division is designed to prevent the same RAM area from being used by both sLib code and end user code.

Figure 4. Main Flash mapping and RAM area division



The filter function code needs to be placed in the main Flash address 0x08004000 ~ 0x08004FFF (sector 8 ~ 9).

The filter coefficient needs to be placed in the main Flash address 0x08005000 ~ 0x08005FFF (sector 10 ~ 11).

The 256-byte RAM address ranging from 0x20000000 to 0x200000FF is to be reserved for sLib code.

To make this happen, follow the procedures below:

1) Create an ld file according to the “AT32F403AxC_FLASH.ld” linker descriptor”. For example, there is “slib.ld” to be created underproject_l0\eclipse_gcc\ld.

2) In “slib.ld”, define the main Flash memory and RAM area

```
/* Specify the memory areas */
MEMORY
{
FLASH_1 (rx)      : ORIGIN = 0x08000000, LENGTH = 16K
SLIB_INST (x)     : ORIGIN = 0x08004000, LENGTH = 4K
SLIB_DATA (r)     : ORIGIN = 0x08005000, LENGTH = 4K
FLASH_2 (rx)      : ORIGIN = 0x08006000, LENGTH = 1000K
SLIB_RAM (xrw)    : ORIGIN = 0x20000000, LENGTH = 0x100  /* used for SLIB code */
RAM (xrw)         : ORIGIN = 0x20000100, LENGTH = 96K - 0x100
}
```

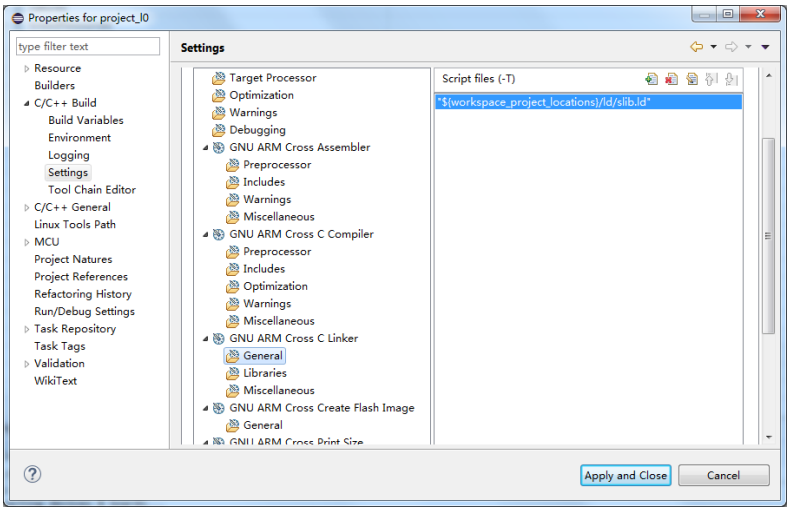
3) Place algorithm code into the *.slib_inst section*, and low-pass filter coefficient into the *.slib_data section*, and global variables for algorithm into the *.slib_ram section*, as shown below.

Figure 5. Configure code, data, ram section

```
/* Define output sections */
SECTIONS
{
  /* The startup code goes first into FLASH */
  .isr_vector :
  {
    . = ALIGN(4);
    KEEP(*(.isr_vector)) /* Startup code */
    . = ALIGN(4);
  } >FLASH_1

  .slib_inst :
  {
    . = ALIGN(4);
    *fir_filter.o (.text .text*);
    *arm_fir_f32.o (.text .text*);
    *arm_fir_init_f32.o (.text .text*);
    . = ALIGN(4);
  } > SLIB_INST

  .slib_data :        /* SLIB_DATA arae */
  {
    . = ALIGN(4);
    *fir_coefficient.o (.rodata .rodata*);
    . = ALIGN(4);
  } > SLIB_DATA

  .slib_ram :          /* Used for SLIB */
  {
    . = ALIGN(4);
    *fir_filter.o (.data .data*);
    *fir_filter.o (.bss .bss*);
    . = ALIGN(4);
  } > SLIB_RAM
```

4) Go to *Project->Properties->C/C++ Build->Setting->GNU ARM Cross C Linker->General*, and add “slib.ld” in it.

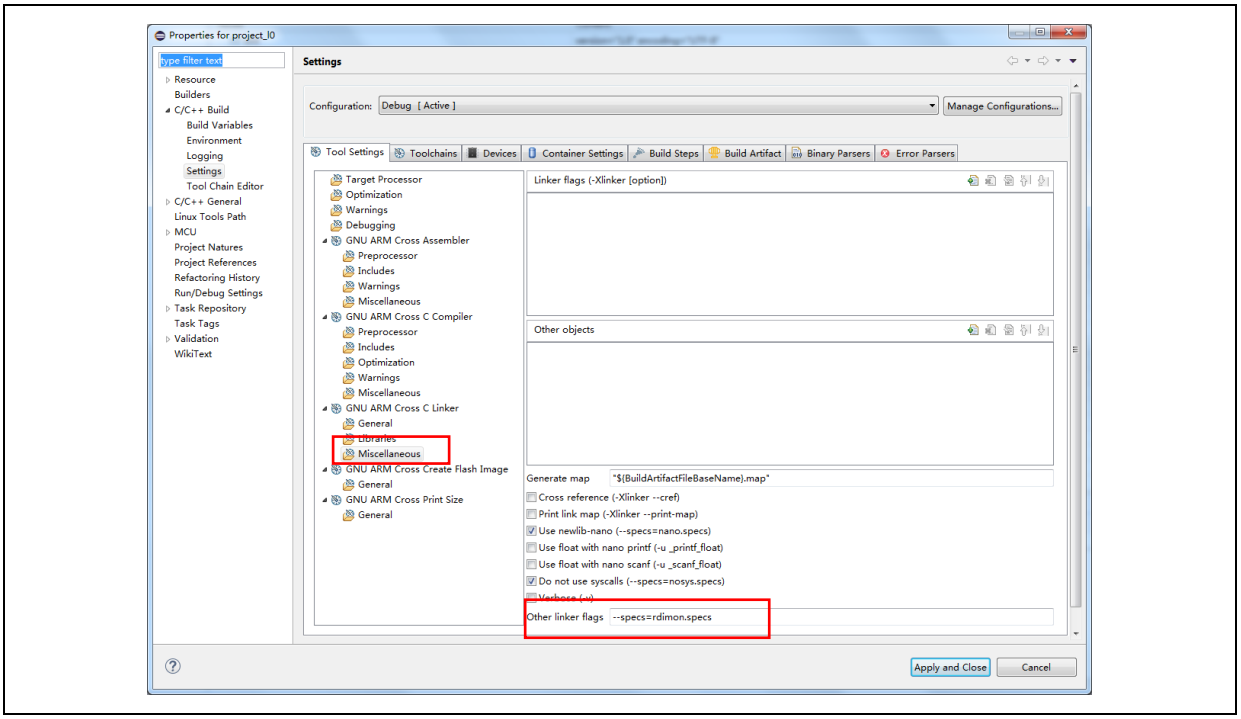
Figure 6. Set script files



5) The libm.a (GCC mathematical function) is used in this example case.

Go to *Properties->GNU ARM Cross C Linker->Miscellaneous*, and then input *"--specs=rdimon.specs"* in the *Other linker flags* dialogue box. This is to avoid linker error.

Figure 7. Set "Other linker flags"



For more information about the syntax of id file, refer to related GNU linker documents.

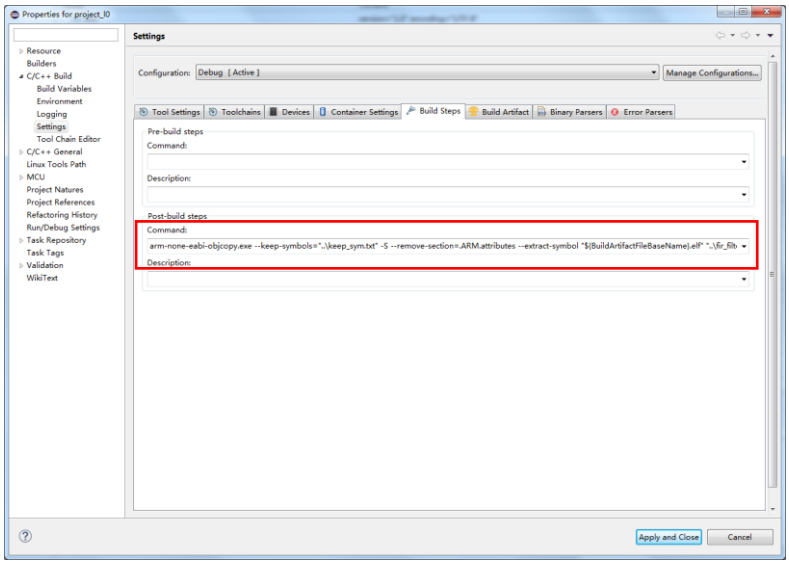
2.3 Head file and symbol definition file

In Project_L1 user application case, the header file and symbol definition file is used to call FIR low-pass filter function.
In this example, the header file refers to the “fir_filter.h” file in the main.c, while the symbol definition file defines the actual address of low-pass filter function.

To generate a symbol definition file, the following procedures are recommended:

1) Go to *Project->Properties->C/C++ Build->Setting*, and then click on *Build Steps*

Figure 8. Set “Build Steps”



In “Post-build steps”, input the command below:
arm-none-eabi-objcopy.exe --keep-symbols="..\keep_sym.txt" -S --remove-section=.ARM.attributes --extract-symbol "${BuildArtifactFileBaseName}.elf" "..\fir_filter_symbol.sym"

2) Where, “fir_filter_symbol.sym” indicates symbol definition file that needs to be generated. “keep_sym.txt” is located in the project_l0\eclipse_gcc and used to select the desired function symbol, as shown below:

```
FIR_lowpass_filter
```

3) After compiling the whole project, there is a symbol definition file called “fir_filter_symbol.sym” under *project_l0\eclipse_gcc*.

2.4 Enable security library

The following two options can be used to enable security library.

1) Artery ICP Programmer (recommended)

The details on enabling security library through ICP Programmer can be found in the *AT32F403A Security Library Application Note*.

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

After successful test of low-pass filter function, executing the *slib_enable()* function once will enable security library. To run this function, just need to enable *#define USE_SLIB_FUNCTION* in the main.c.

3 Project_L1 user case

Project_L1 example case needs to use the FIR low-pass filter function which is already debugged and programmed into main Flash memory of the AT32F403A and is sLib-enabled (security library) in Project_L0 example case.

The end user can follow Project_L1 example case to do the following operations, based on the header file, symbol definition file and main Flash mapping provided in Project_L0.

- Create a project
- Import header file and symbol definition file provided in Project_L0 to this project
- Call FIR low-pass filter function
- Develop and debug user applications

Note: Project_L1 must use the same tool chain and compiler edition as that of Project_L0 in order to avoid incompatibility between editions which may make it impossible to use Project_L0 code

3.1 Create user project

As mentioned before, the security library in Project_L0 occupies partial Flash pages, and thus the code address in Project_L1 must be allocated according to the main Flash memory mapping provided in Project_L0. The sector 8~11 of Project_L0 is owned by security library, so that such area has to be isolated using id file, in order to avoid other code being put into it.

To make this happen:
Divide the main Flash space into two blocks, FLASH_1 and FLASH_2, by using *end_user_code.ld* file under *project_l1\eclipse_gcc\ld.* The area left between FLASH_1 and FLASH_2 is the so-called security library area.
Besides, the RAM area ranging from 0x20000000 to 0x200000FF must also be reserved, as shown below:

Figure 9. end-user-code.ld configuration

```
/* Specify the memory areas */
MEMORY
{
FLASH_1 (rx)     : ORIGIN = 0x08000000, LENGTH = 16K
SLIB_INST (x)    : ORIGIN = 0x08004000, LENGTH = 4K
SLIB_DATA (r)    : ORIGIN = 0x08005000, LENGTH = 4K
FLASH_2 (rx)     : ORIGIN = 0x08006000, LENGTH = 1000K
SLIB_RAM (xrw)   : ORIGIN = 0x20000000, LENGTH = 0x100  /* used for SLIB code */
RAM (xrw)        : ORIGIN = 0x20000100, LENGTH = 96K - 0x100
}

/* Define output sections */
SECTIONS
{
  /* The startup code goes first into FLASH */
  .isr_vector :
  {
    . = ALIGN(4);
    KEEP(*(.isr_vector)) /* Startup code */
    . = ALIGN(4);
  } >FLASH_1

  .slib_inst (NOLOAD):
  {
    *(.slib_code)
  } > SLIB_INST

  .slib_data (NOLOAD) :
  {
    KEEP(*(.slib_data))
  } > SLIB_DATA

  .slib_ram :
  {
    KEEP(*(.slib_ram))
  } > SLIB_RAM
```

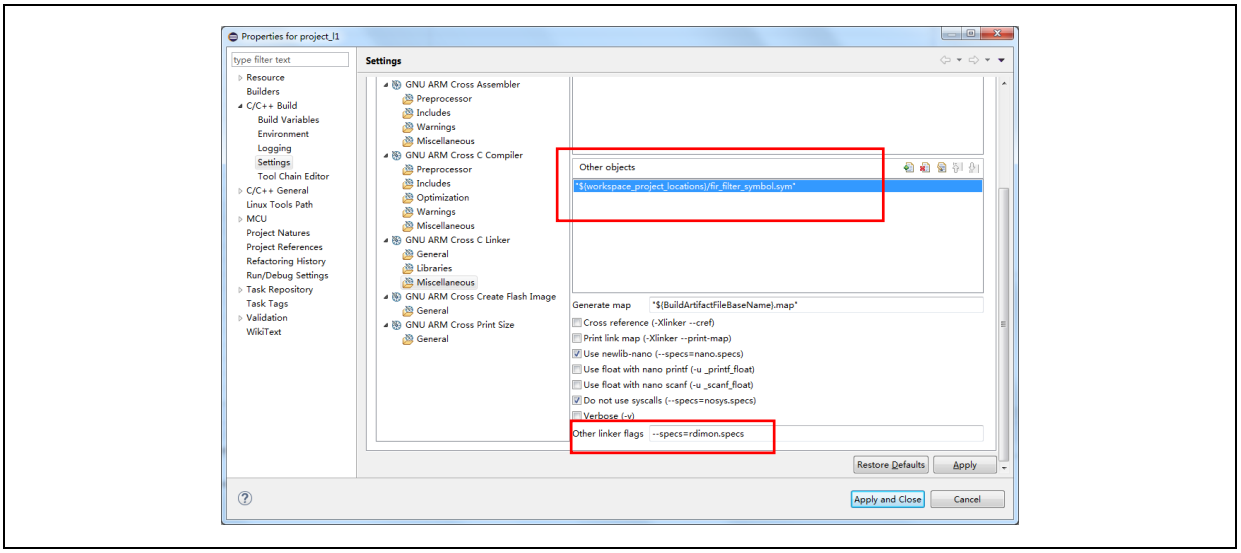
In figure 9, the SLIB_CODE and SLIB_DATA have been programmed with code in advance by solution providers, and thus they are marked as NOLOAD to avoid being downloaded again while downloading Project_L1 code into main Flash memory.

3.2 Add symbol definition file in project

The *fir_filter_symbol.sym* (symbol definition file) generated in Project_L0 must be added into Project_L1 so that it can be compiled and linked to SLIB code, as configured below:

1) Add *fir_filter_symbol.sym* into *FIR_Filter* group
2) Go to *Project->Properties->C/C++ Build->Settings->Tool Setting->GNU ARM Cross C Linker->Miscellaneous*, and add the symbol definition file into *Other objects*. In this way, this file can be found and linked while compiling project.

Figure 10. Set "Other objects"



4 Revision history

Table 1. Document revision history

Date	Revision	Changes
2022.01.06	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.