Thank you for using the RL78 Family Flash Self-Programming Library Type01 Package Ver.4.00.

This document contains restrictions and notes regarding use of the Flash Self-Programming Library Type01 Package Ver.4.00. Please read this document before using the library.

Contents

Chapter 1 Target Product 2
Chapter 2 User's Manual 2
Chapter 3 Revisions 2
Chapter 4 Points for Caution 3
Chapter 5 Supported Tools 3
Chapter 6 Installation 4
6.1 Installation 4
6.2 Uninstallation 4
6.3 File Organization 5
Chapter 7 How to Build a Program 6
7.1 Software to be used 6
7.2 Building using CS+(former CubeSuite+) 6
7.2.1 Building a C program 7
7.2.2 Building an assembly language program 10
7.2.3 Removing the automatically generated files (only when the CC-RL compiler is used) 12
7.2.4 Building 13
7.3 Building Using e^2 studio 13
7.3.1 Creating a Project 13
7.3.2 Building a C Program 15
7.3.3 Building an Assembly-Language Program 19
7.4 Notes at Build 21
7.4.1 When the CA78K0R Compiler is Used 21
7.4.2 When the CC-RL Compiler is Used 22
Chapter 8 How to Debug a Program 23
Chapter 9 Sample Program 24
9.1 Initial Settings of the Sample Program 24
9.2 Settings of Option byte and On-Chip Debugging 25
9.3 Compilation Switch for the C-Language Sample Program 27
9.4 Defining the Internal RAM Area 28
9.4.1 When the CA78K0R Compiler is Used 28
9.4.2 When the CC-RL Compiler is Used 29
9.4.3 When the LLVM Compiler is Used 33

Chapter 1 Target Product

In the Flash Self-Programming Library Type01 Package Ver.4.00, “RL78 Family Flash Self-Programming Library Type01 for LLVM” has been newly added. The following shows the target products for this release note.

Product Name	Ver.	Installer File Name	Ver.
Flash Self-Programming Library Type01 for CA78K0R Compiler for the RL78 Family	V2.20	RENESAS_RL78_FSL_T01_4V00.exe	V4.00
Flash Self-Programming Library Type01 for CC-RL Compiler for the RL78 Family	V2.21		
Flash Self-Programming Library Type01 for LLVM Compiler for the RL78 Family	V2.21		

Chapter 2 User’s Manual

The following user’s manual covers this version of the library.

Title	Document Number
RL78 Family Flash Self-Programming Library Type01 User’s Manual	R01US0050EJ0110

Chapter 3 Revisions

The following shows the items revised in this version.

No.	Package Ver.	Target	Contents
1	V4.00	Library V2.20 for CA78K0R Compiler	There is no change in the library from the package Ver.3.00.
		Library V2.21 for CC-RL Compiler	There is no change in the library from the package Ver.3.00.
		Library V2.21 for LLVM Compiler	Newly added.
		User’s manual	Revised from Rev.1.05 to Rev.1.10. For details on the corrections to the user's manual in response to the revision, refer to the revision history of the user's manual.

Chapter 4 Points for Caution

For points for caution on using the Flash Self-Programming Library Type01, read this chapter and the user's manual described later.

No.	Description
1	• Debugging by a simulator The flash self-programming library cannot be debugged by a simulator. To perform debugging, either use the on-chip debugging function of the RL78 microcontroller or prepare the IECUBE.
2	• Restrictions regarding use of the flash self-programming library on RL78/G13 sample devices (not including mass-produced devices) Some RL78/G13 sample devices (not including mass-produced devices) have restrictions on the interrupt vector change processing through the flash self-programming library. Note that the following flash functions cannot be used on those devices. • FSL_ChangeInterruptTable • FSL_RestoreInterruptTable

Chapter 5 Supported Tools

Use the following tool version when using the Flash Self-Programming Library Type01.

Target library	Tool Name	Version
Library for CA78K0R Compiler	Integrated development environment CubeSuite+	V1.00.00 or later
	Integrated development environment CS+	V3.00.00 or later
Library for CC-RL Compiler	Integrated development environment CS+	V3.01.00 or later
	Integrated development environment e2 studio	Listed from Version: 2023-10 Note
Library for LLVM Compiler	Integrated development environment e2 studio	Version: 2023-10 or later

Note: Available for e2 studio with embedded CC-RL compiler V1.00 or later.

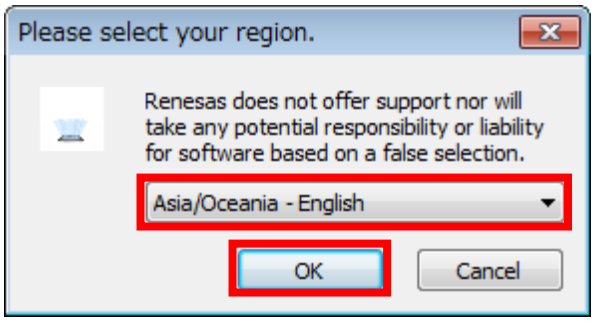
Chapter 6 Installation

This chapter describes how to install and uninstall the Flash Self-Programming Library Type01 Package Ver.4.00.

6.1 Installation

Install the Flash Self-Programming Library Type01 by using the following procedure:

(1) Start Windows.
(2) Decompress the file that contains the Flash Self-Programming Library Type01 Package and run the installer.
(3) Select "Asia/Oceania - English" from the drop-down list.
(4) Click on the "OK" button to proceed installation according to the instructions of the installer.



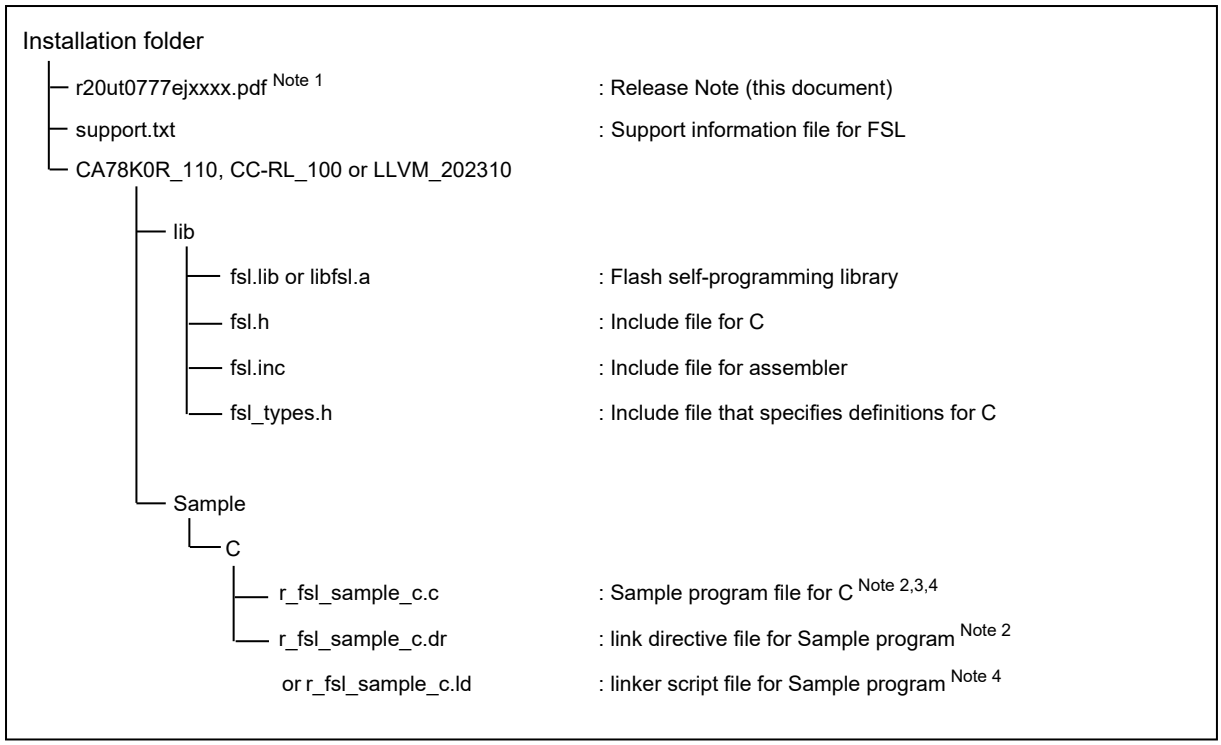
6.2 Uninstallation

Uninstall the Flash Self-Programming Library Type01 by using the following procedure:

(1) Start Windows.
(2) Delete the folder that contains the Flash Self-Programming Library Type01 files.

6.3 File Organization

The file organization after this library is installed is shown below.



Notes: 1. x indicates the omitted numerals in version or revision numbers.

2. To use the sample program for CA78K0R, the program file (*.c) and link directive file (*.dr) should be embedded together.
3. To use the sample program for CC-RL, the program file (*.c) should be embedded. The link information for the sample program for CC-RL should be specified through the link setting window on the CS+ or the e^2 studio.
4. To use the sample program for LLVM, the program file (*.c) and linker script file (*.ld) should be embedded together.

Chapter 7 How to Build a Program

This chapter describes how to build a program using the Flash Self-Programming Library Type01.

7.1 Software to be used

The following integrated development environment is necessary for building programs using the Flash Self-Programming Library Type01.

- Integrated development environment CS+ V3.00.00 or later for CA78K0R compiler
  /Integrated development environment CubeSuite+ V1.00.00 or later for CA78K0R compiler
- Integrated development environment CS+ V3.01.00 or later for CC-RL compiler
  /Integrated development environment e^2 studio Listed from Version:2023-10 or later for CC-RL compiler Note
  Note: Available for e2 studio with embedded CC-RL compiler V1.00 or later.
- Integrated development environment e^2 studio Version: 2023-10 or later for LLVM compiler

7.2 Building using CS+(former CubeSuite+)

This section describes how to include the Flash Self-Programming Library Type01 in a user-created program and build the user program by using CS+. The target compilers for CS+ are CC-RL compiler and CA78K0R compiler.

7.2.1 Building a C program

(1) Creating a project and specifying the source file

Create a project by using CS+. In the Project Tree window displayed on the left, right-click the File node, click Add, and then click Add File. The Add Existing File dialog box is displayed (as shown in Figure 7-1).
Next, click the Files of type drop-down list to display a list of the file types. Select C source file (*.c), and then register the user-created program as the source file.

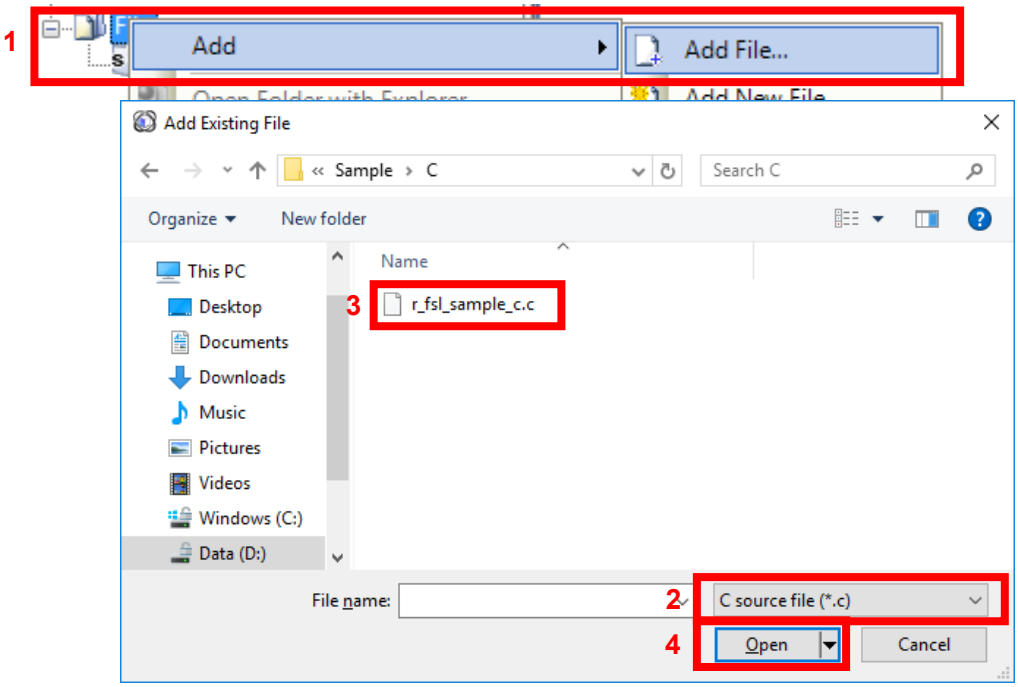


Figure 7-1. Registering the User Program File

(2) Specifying the include file

In the CS+ Project Tree window, right-click the File node, click Add, and then click Add File.
The Add Existing File dialog box is displayed (as shown in Figure 7-2).
Next, click the Files of type drop-down list to display a list of the file types. Select Header file (*.h;*.inc), and then register the header files (fsl.h, fsl_types.h) of the flash self-programming library.

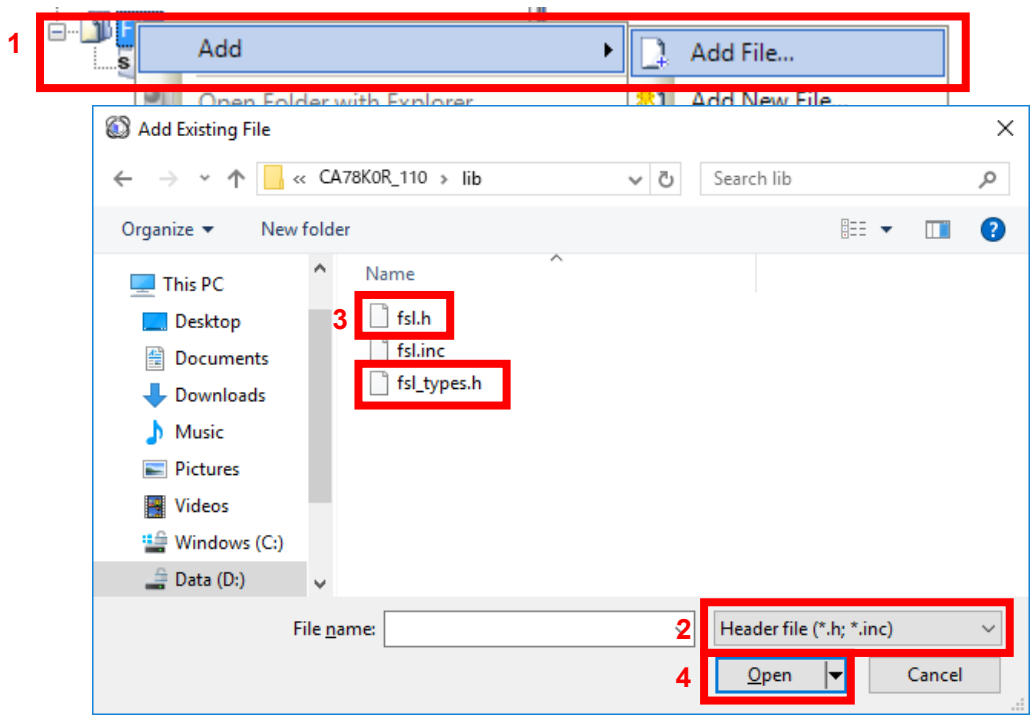


Figure 7-2. Registering the Include Files

(3) Specifying the library file

In the CS+ Project Tree window, right-click the File node, click Add, and then click Add File. The Add Existing File dialog box is displayed (as shown in Figure 7-3).
Next, click the Files of type drop-down list to display a list of the file types. Select Library file (*.lib), and then register the flash self-programming library file (fsl.lib).

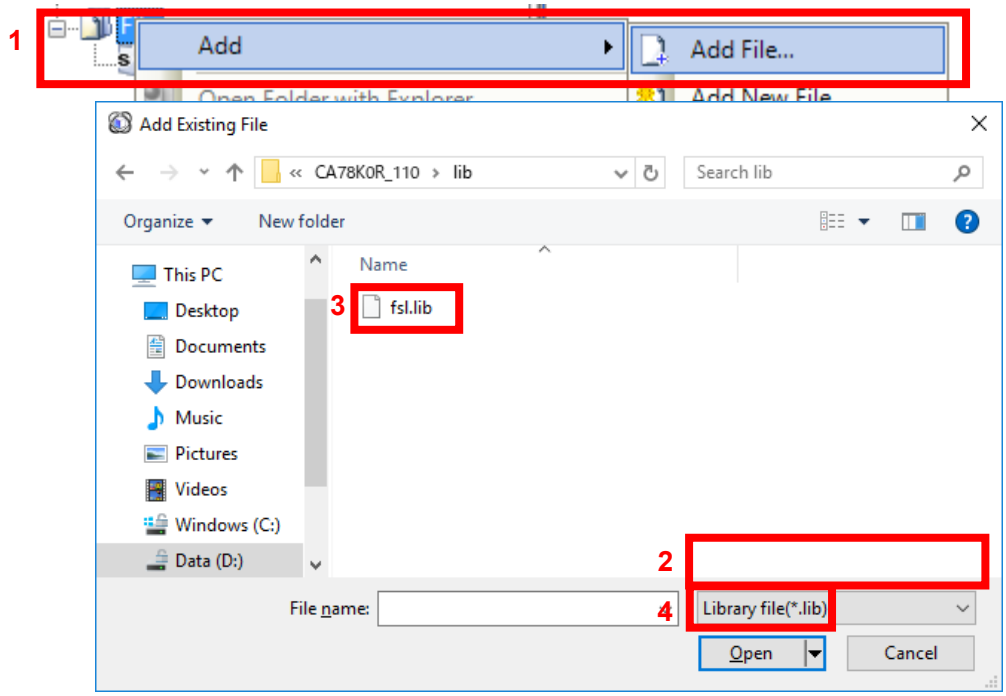


Figure 7-3. Registering the Library File

(4) Specifying the link directive file (only when the CA78K0R compiler is used)

In the CS+ Project Tree window, right-click the File node, click Add, and then click Add File. The Add Existing File dialog box is displayed (as shown in Figure 7-4).

Next, click the Files of type drop-down list to display a list of the file types. Select Link Directive File (*.dr;*.dir), and then register the link directive file that has the same name as the user-created program.

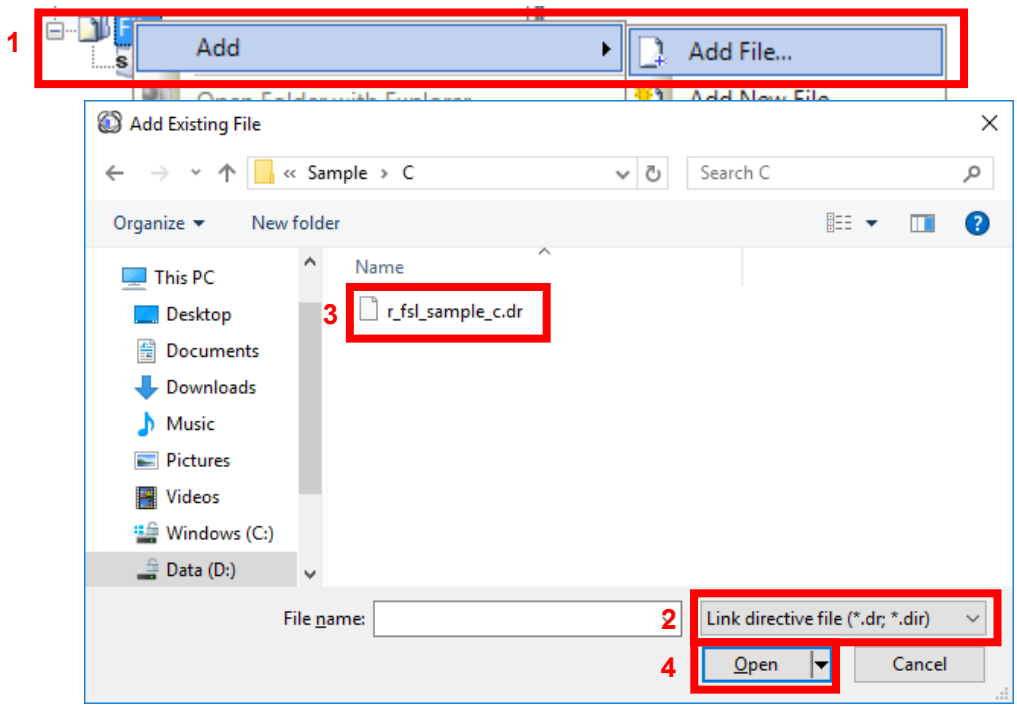


Figure 7-4. Registering the Link Directive File

7.2.2 Building an assembly language program

(1) Creating a project and specifying the source file

Create a project by using CS+. In the Project Tree window displayed on the left, right-click the File node, click Add, and then click Add File. The Add Existing File dialog box is displayed (as shown in Figure 7-5).
Next, click the Files of type drop-down list to display a list of the file types. Select Assemble file (*.asm), and then register the user-created program as the source file.

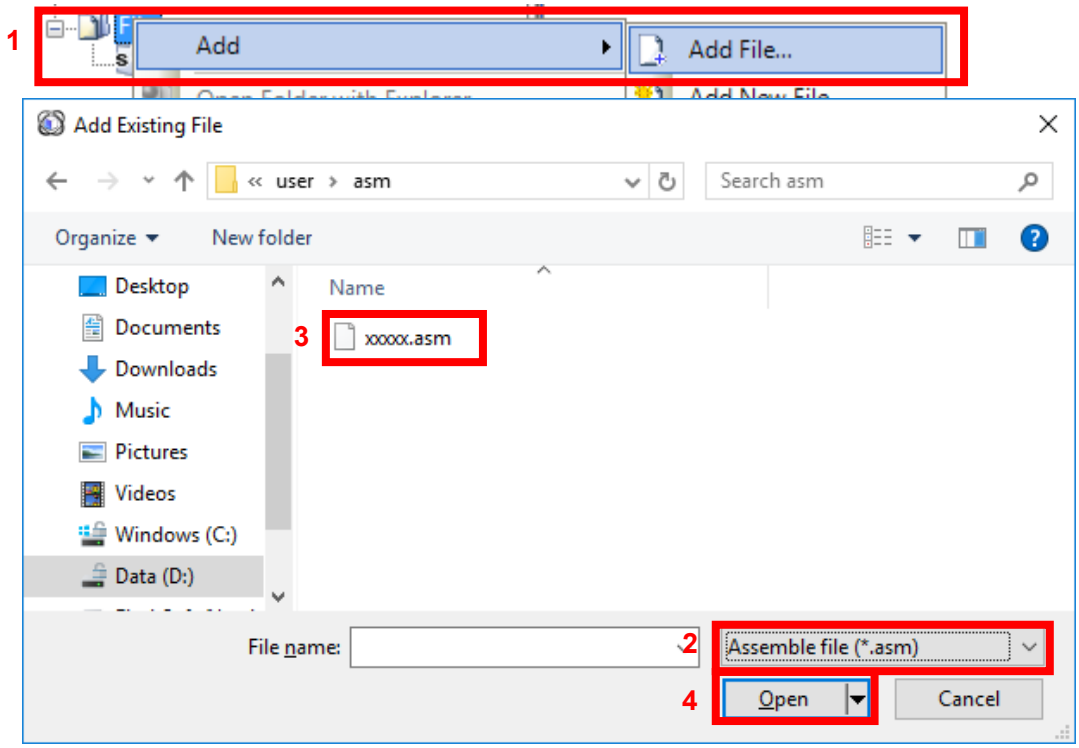


Figure 7-5. Registering the User Program File

(2) Specifying the include file

In the CS+ Project Tree window, right-click the File node, click Add, and then click Add File.
The Add Existing File dialog box is displayed (as shown in Figure 7-6).
Next, click the Files of type drop-down list to display a list of the file types. Select Header file (*.h;*.inc), and then register the header file (fsl.inc) of the flash self-programming library.

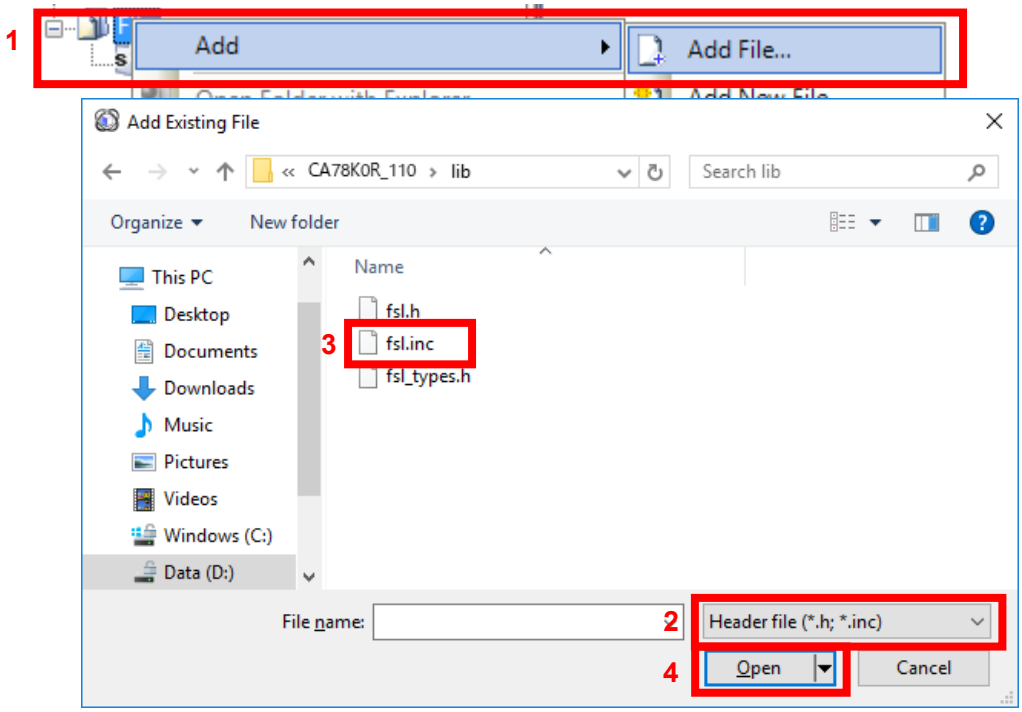


Figure 7-6. Registering the Include File

(3) Specifying the library file

In the CS+ Project Tree window, right-click the File node, click Add, and then click Add File. The Add Existing File dialog box is displayed (as shown in Figure 7-7).

Next, click the Files of type drop-down list to display a list of the file types. Select Library file (*.lib), and then register the flash self-programming library file (fsl.lib).

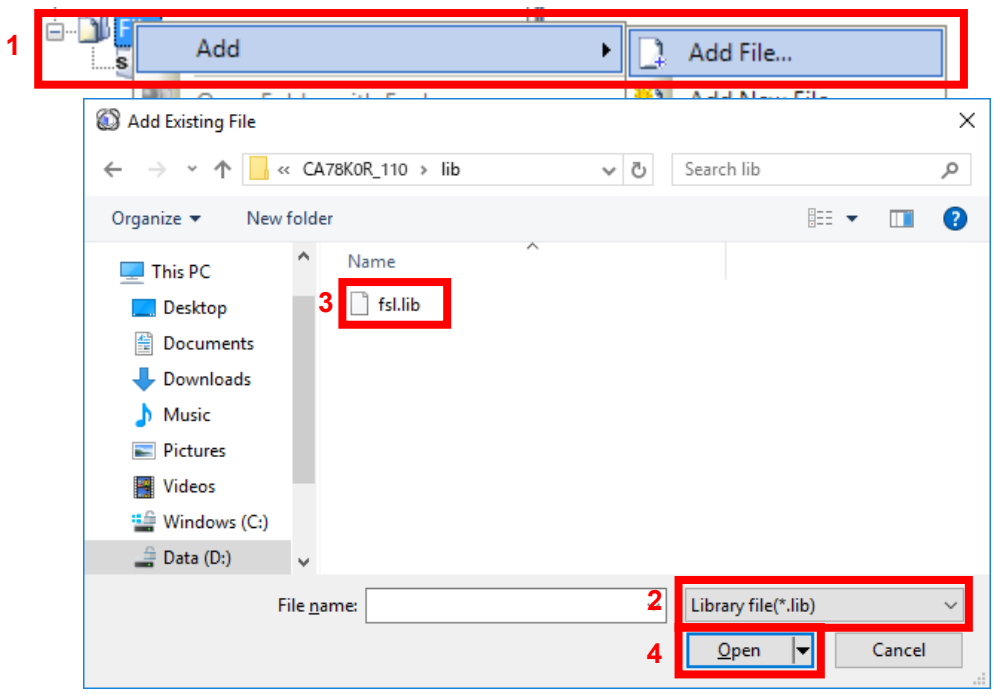


Figure 7-7. Registering the Library File

(4) Specifying the link directive file (only when the CA78K0R compiler is used)

In the CS+ Project Tree window, right-click the File node, click Add, and then click Add File. The Add Existing File dialog box is displayed (as shown in Figure 7-8).
Next, click the Files of type drop-down list to display a list of the file types. Select Link Directive File (*.dr;*.dir), and then register the link directive file that has the same name as the user-created program.

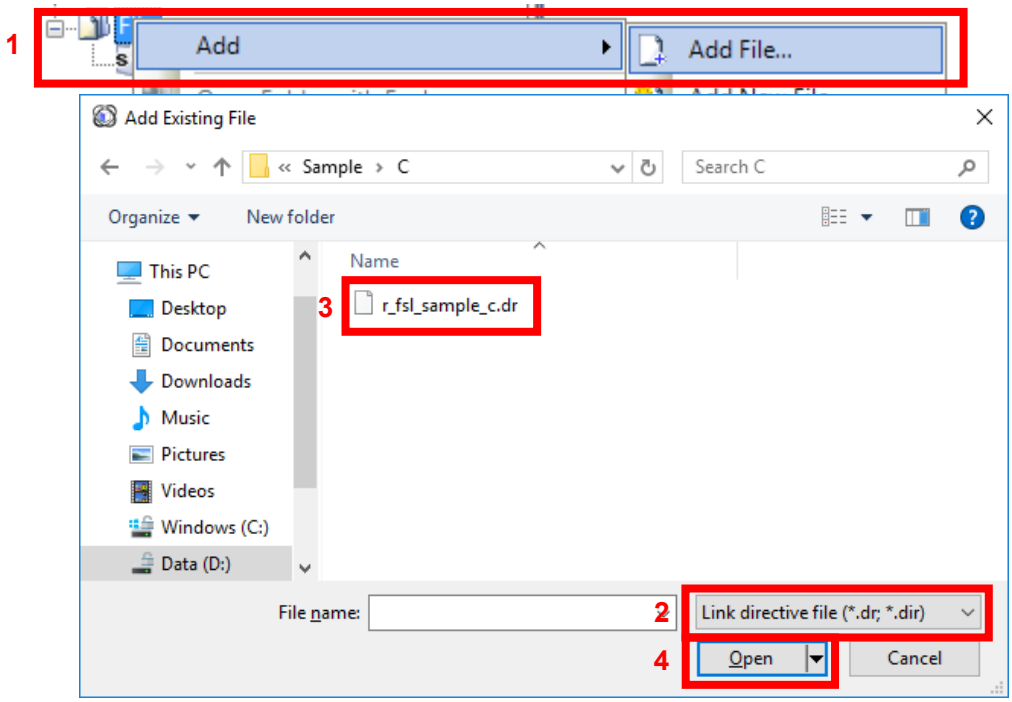


Figure 7-8. Registering the Link Directive File

7.2.3 Removing the automatically generated files (only when the CC-RL compiler is used)

CS+ for the CC-RL compiler automatically generates some files under the File node in the Project Tree window. Among these, the processing of the "main.c" and "hdwinit.asm" files is included in the flash self-programming library. Therefore, remove these two files from the target of the build process.
To use assembly language, only “main.c” is removed because the sample program is not used.

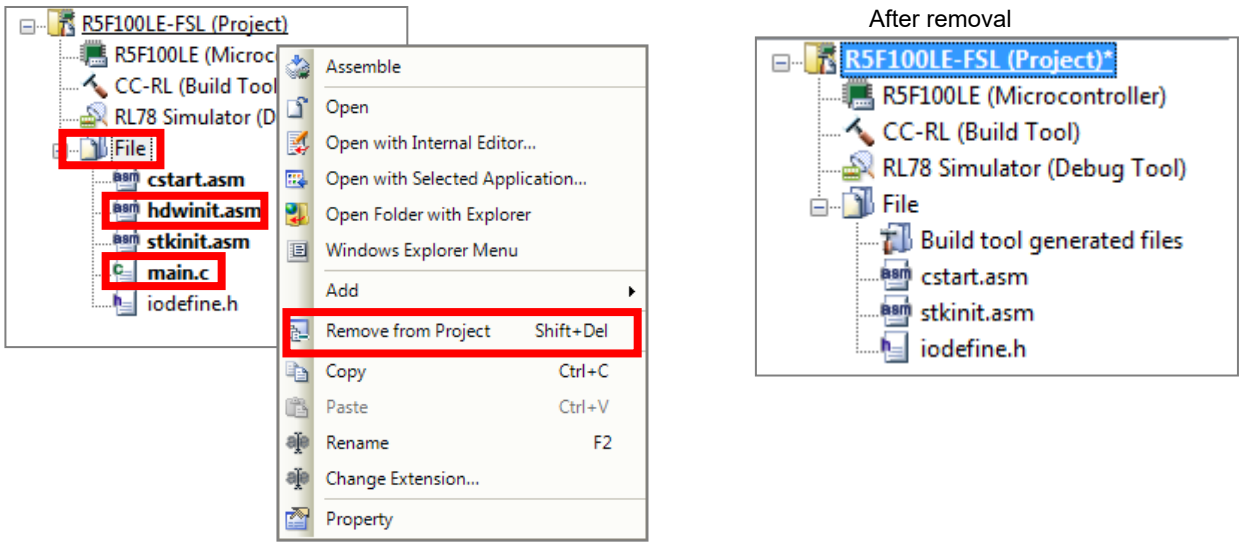


Figure 7-9. Removing the Automatically Generated Files

7.2.4 Building

On the CS+ Build menu, click Build Project to build the project.

7.3 Building Using e² studio

This section describes how to include the Flash Self Programming Library Type01 in a user-created program and build the user program by using e² studio. The target compilers for e² studio are CC-RL and LLVM.

7.3.1 Creating a Project

The e² studio starts and from the [File] menu, select [New] – [C/C++ Project], the “Templates for New C/C++ Project” window will open.

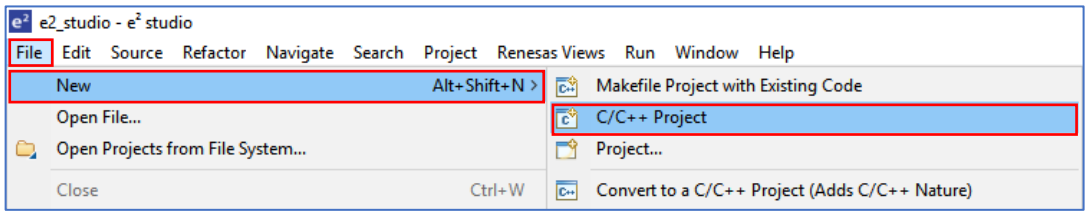


Figure 7-10. Removing the Automatically Generated Files

- When using the CC-RL compiler, select [Renesas CC-RL C/C++ Executable Project] displayed after selection in [Renesas RL78], and press “next” button.

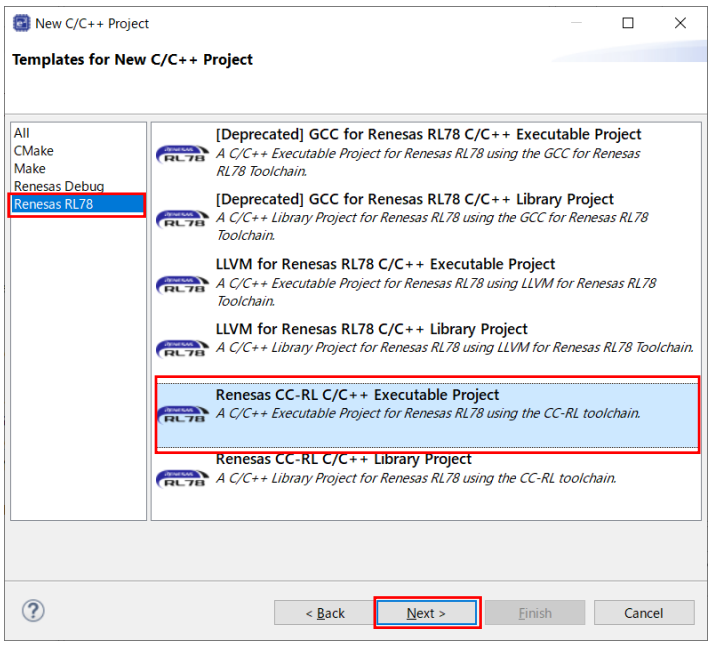


Figure 7-11. Select the CC-RL Compiler for the Tool Chain

Input “project name” on “New Renesas CC-RL Executable Project" window, and press “Next” button.

- When using the LLVM compiler, Select [LLVM for Renesas RL78 C/C++ Executable Project] displayed after selection in [Renesas RL78], and press “Next” button.

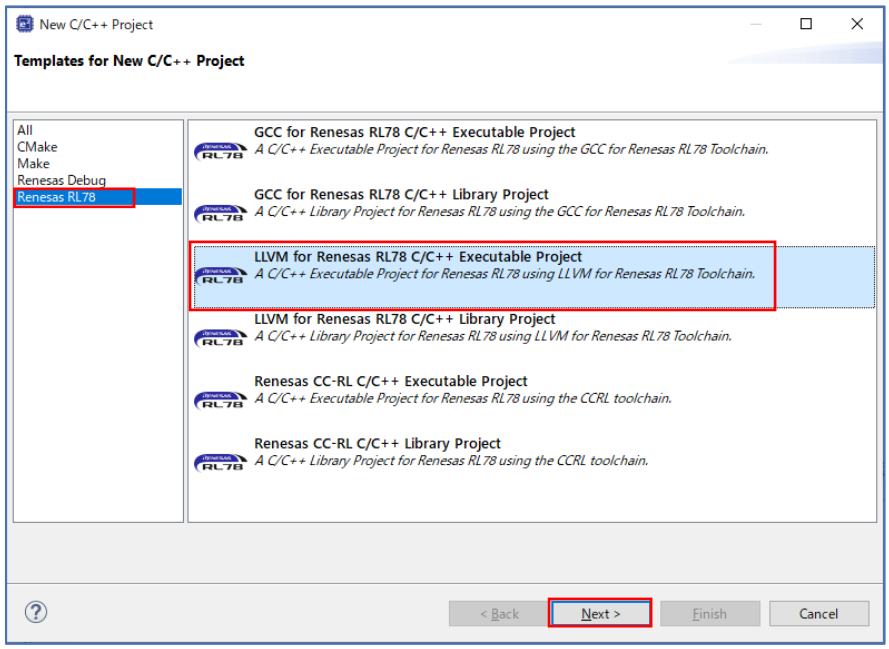


Figure 7-12. Select the LLVM Compiler for the Tool Chain

Input “Project name” on “New LLVM for Renesas RL78 Executable Project” window, and press “Next” button.

Select the [Target Device] of [Device Settings] and select “RL78 – G13” - “R5F100LE”. (When the target device is RL78/G13 [Part Number: R5F100LE].)

It is a premise that E2 Lite is selected as a debugging tool and on-chip debugging is executed. Put a check mark to “Create Hardware Debug Configuration” by [Configurations]. And select “E2 Lite(RL78)”. Press “Finish” button.

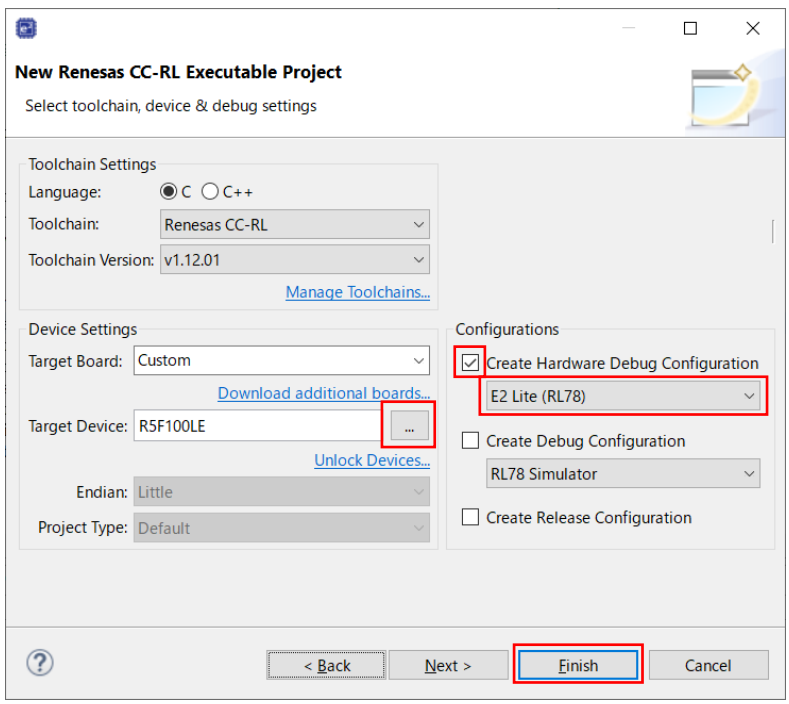


Figure 7-13. Device Selection

7.3.2 Building a C Program

(1) When the on-chip debugging function is in use

Specifying the flash self-programming library file in the created project.

- CC-RL: Register the flash self-programming library files “fsl.h”, “fsl_types.h”, “fsl.lib” and “r_fsl_sample_c.c” in the “src” folder output by e^2 studio. (Figure 7-14)

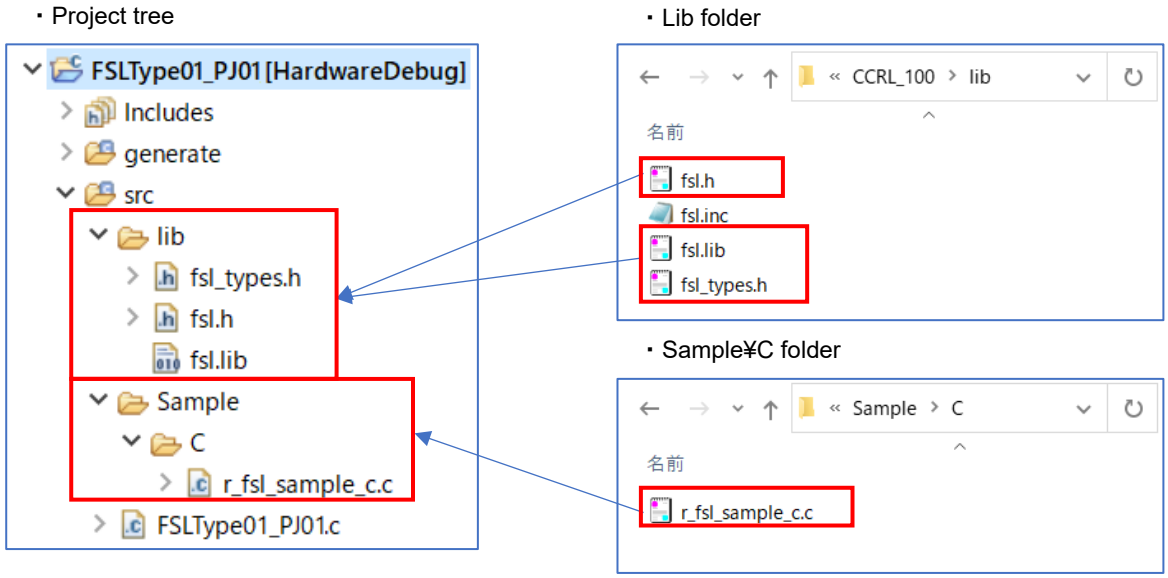


Figure 7-14. Specifying the Source and Include Files (CC-RL)

- LLVM: Register the flash self-programming library files “fsl.h”, “fsl_types.h”, “libfsl.a”, “r_fsl_sample_c.c” and “r_fsl_sample_c.ld” in the “src” folder output by e^2 studio. (Figure 7-15)

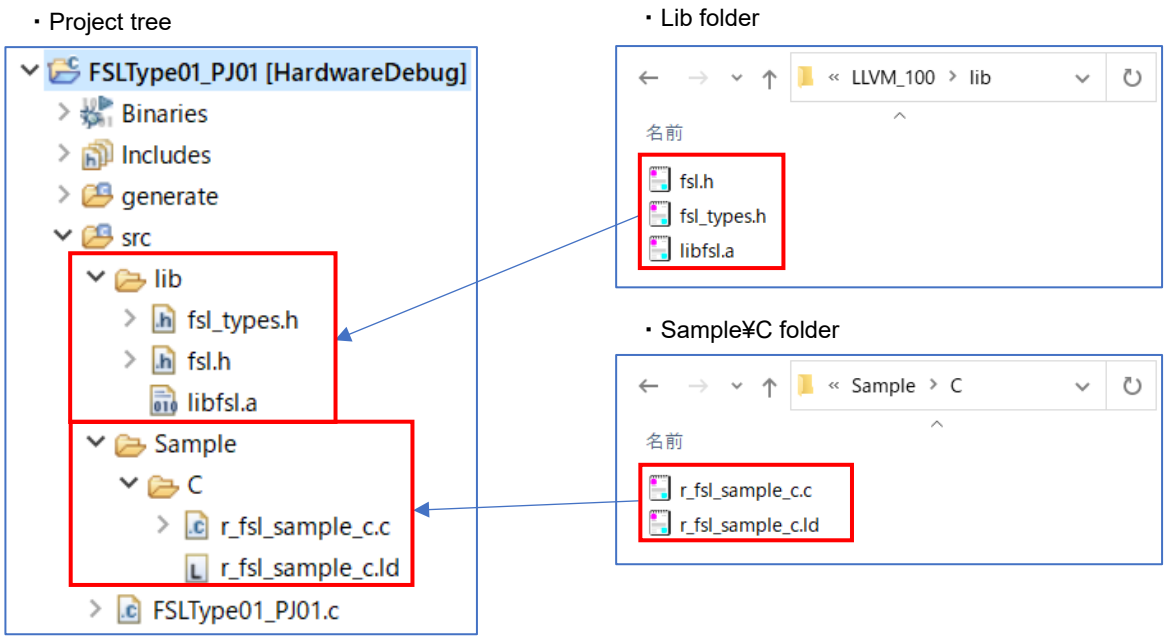


Figure 7-15. Specifying the Source and Include Files (LLVM)

Exclusion of the file automatically added by the function of e^2 studio.

There are files added automatically in the created project. The same file as these exists also in the “sample” folder of FSL Type01. Therefore, using the function of IDE, Select those files from tree, and excludes from a project. Clicks the right mouse button for the file of tree. And On the [Settings] screen displayed by the “Properties”, put a check mark to [Exclude resource from build] and exclude a target file. (Exclusion of a folder is also possible)

- CC-RL: Target files are “hdwinit.asm” in a [project name]/generate folder, and [project name] .c (“FSLType01_PJ01.c”) in a [project name]/src folder.
- LLVM: Target files are “linker_script.ld” in a [project name]/generate folder, and [project name] .c (“FSLType01_PJ01.c”) in a [project name]/src folder.

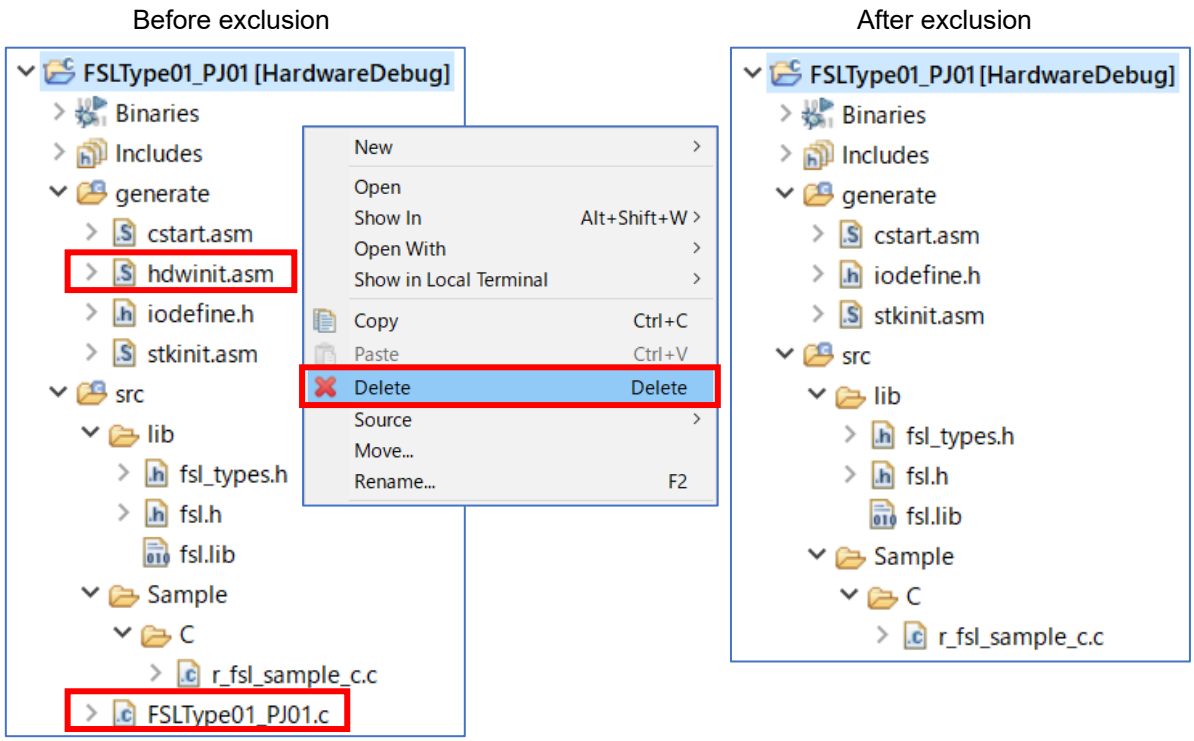


Figure 7-16. File Exclusion Example

(2) Specifying the library file

- CC-RL: Click the right mouse button for the project in a tree, and select “Properties”. In the “Add file” window that appears by clicking the “+” button to the right of “Relocatable files, object files, and library files” on the “C/C++ Build” [Settings] – “Linker” [Input] screen, change the [Format] to “library”, and register the path to the flash self-programming library file “fsl.lib”. (Figure 7-17)

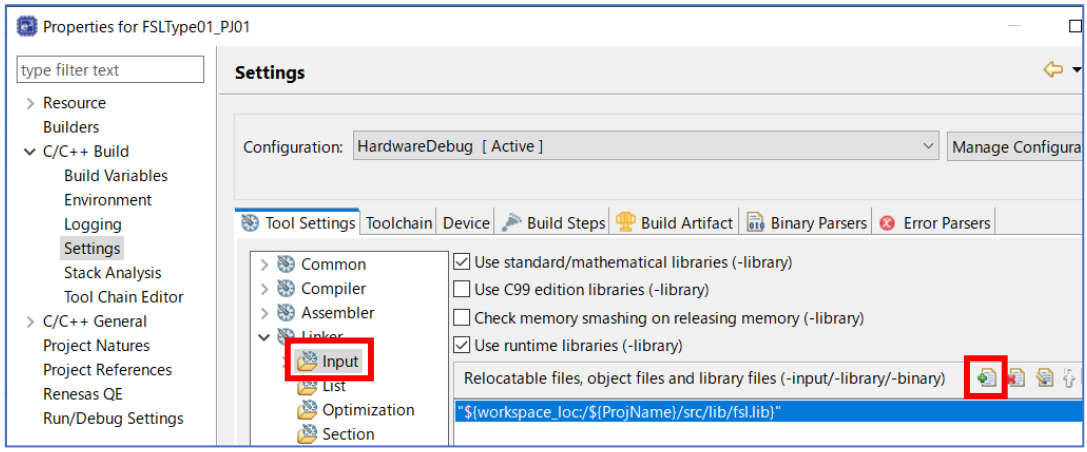


Figure 7-17 (a). Specifying the Library File (CC-RL)

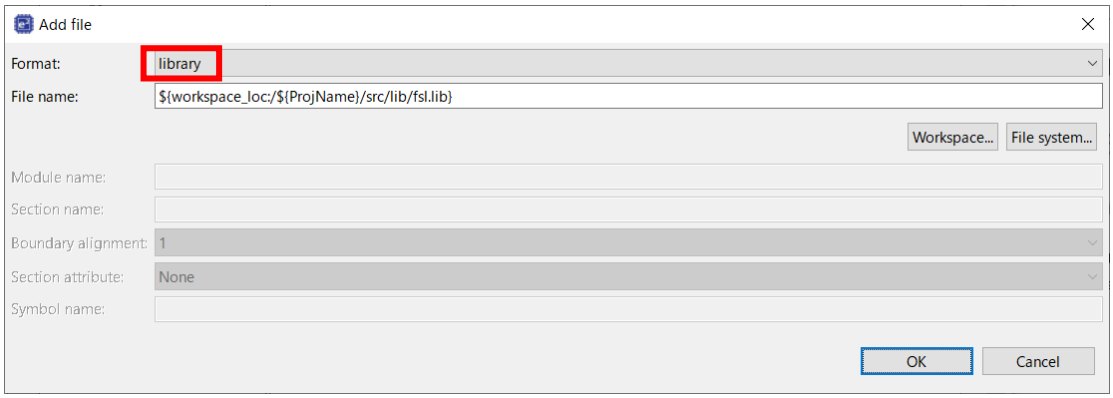


Figure 7-17 (b). Specifying the Library File (CC-RL)

- LLVM: Click the right mouse button for the project in a tree, and select “Properties”. Register the file path of the flash self-programming library file “libfsl.a” in the “Additional input files” field on the screen displayed in “C/C++ Build” [Settings] – “Linker” [Source].

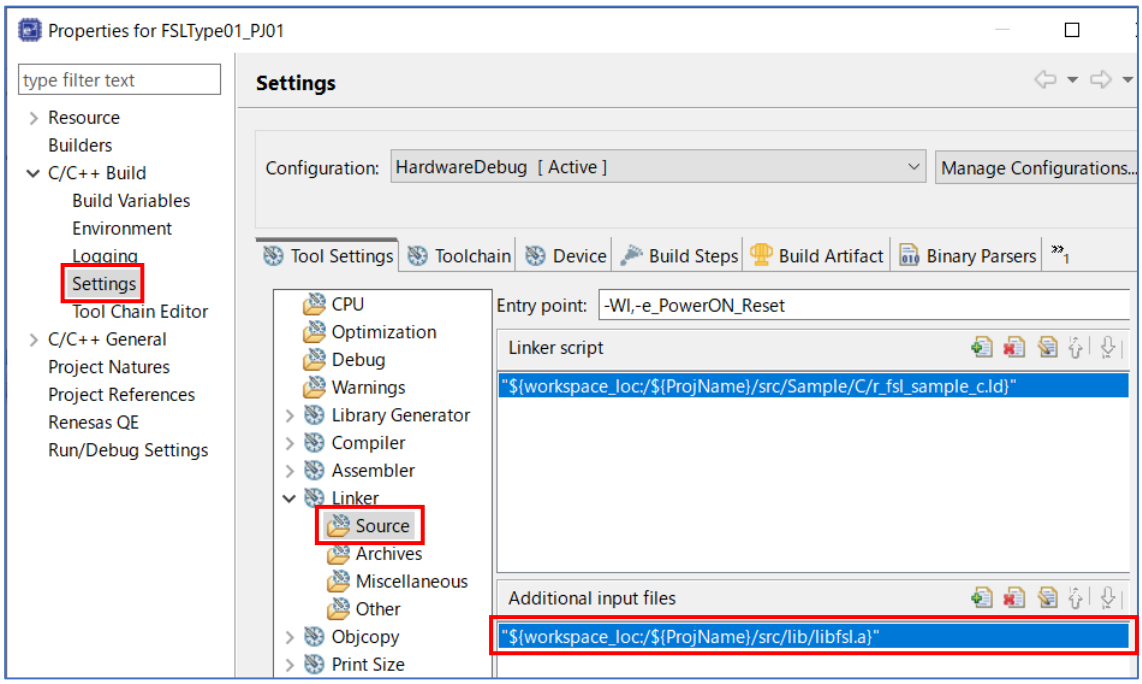


Figure 7-18. Specifying the Library File (LLVM)

(3) Specifying the linker script file (only when the LLVM compiler is used)

Click the right mouse button for the project in a tree, and select “Properties”. Register the file path of the linker script file “.ld” in the “Linker script” field on the screen displayed in “C/C++ Build” [Settings] – “Linker” [Source].

Here, select the file path of “r_fsl_sample_c.ld” prepared for the FSL Type01.

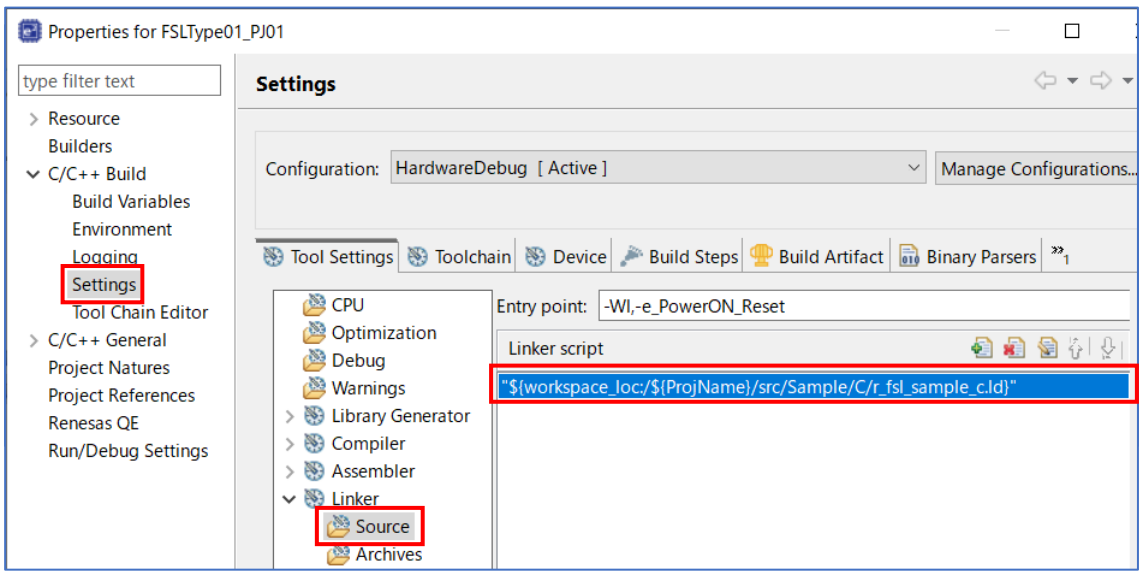


Figure 7-19. Specifying the Linker Script File

Note: Refer to each reference manual of LLVM about the descriptive content of linker script file, and the details of the description method.

(4) Building

Right-click on the [Project] in the e^2 studio project tree and select “Build Project” to build the project.

7.3.3 Building an Assembly-Language Program

(1) Specifying the source and include files

Specifying the flash self-programming library file in the created project.

- CC-RL: Register user program file (“xxxxxx.asm”), the flash self-programming library files “fsl.inc” and “fsl.lib” in the “src” folder output by e^2 studio. (Figure 7-20)

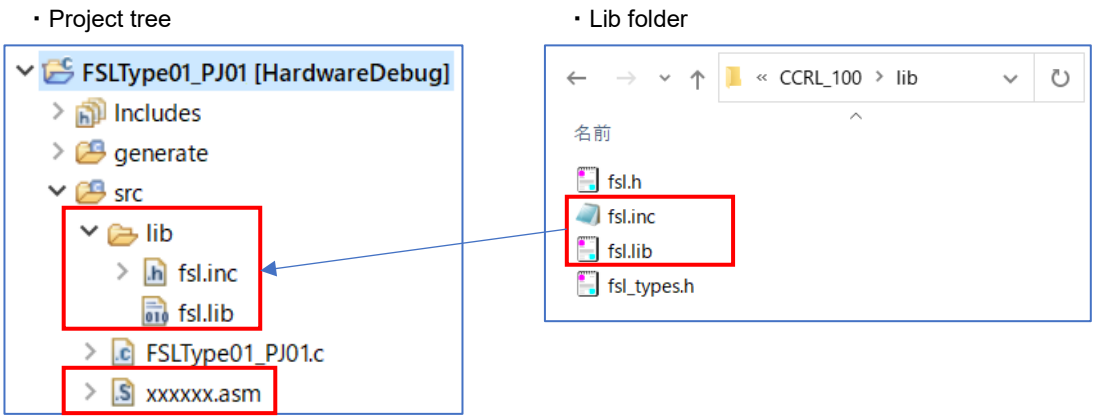


Figure 7-20. Specifying the Source and Include Files (CC-RL)

Exclusion of the file automatically added by the function of IDE.

There are files added automatically in the created project. The same file as these exists also in the “sample” folder of FSL Type01. Therefore, using the function of IDE, Select those files from tree, and excludes from a project. Clicks the right mouse button for the file of tree. And On the [Settings] screen displayed by the “Properties”, put a check mark to [Exclude resource from build] and exclude a target file (target folder).

(Exclusion of a folder is also possible)

- CC-RL: Target file is [project name] .c (“FSLType01_PJ01.c”) in a [project name]/src folder.

(2) Specifying the library file

- CC-RL: Click the right mouse button for the project in a tree, and select “Properties”. In the “Add file” window that appears by clicking the “+” button to the right of “Relocatable files, object files, and library files” on the “C/C++ Build” [Settings] – “Linker” [Input] screen, change the [Format] to “library”, and register the path to the flashself-programming library file “fsl.lib”. (Figure 7-21)

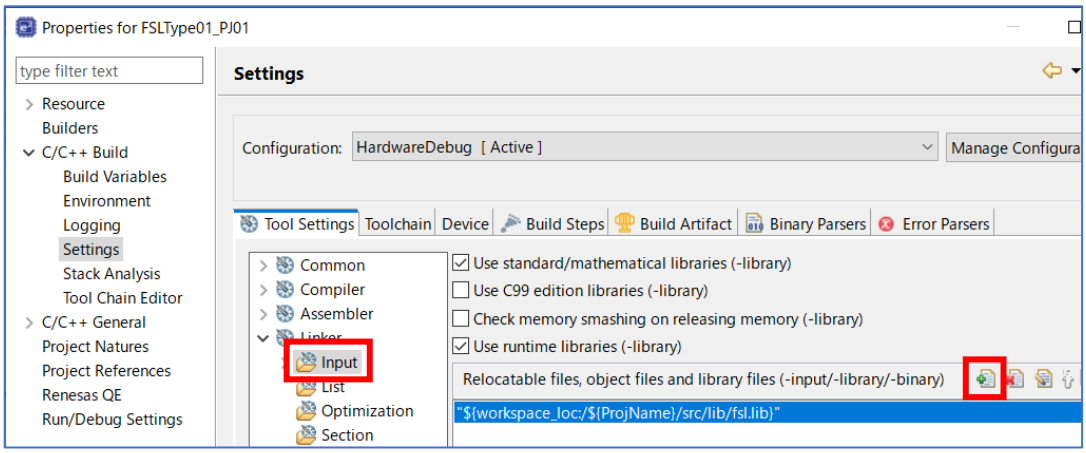


Figure 7-21 (a). Specifying the Library File (CC-RL)

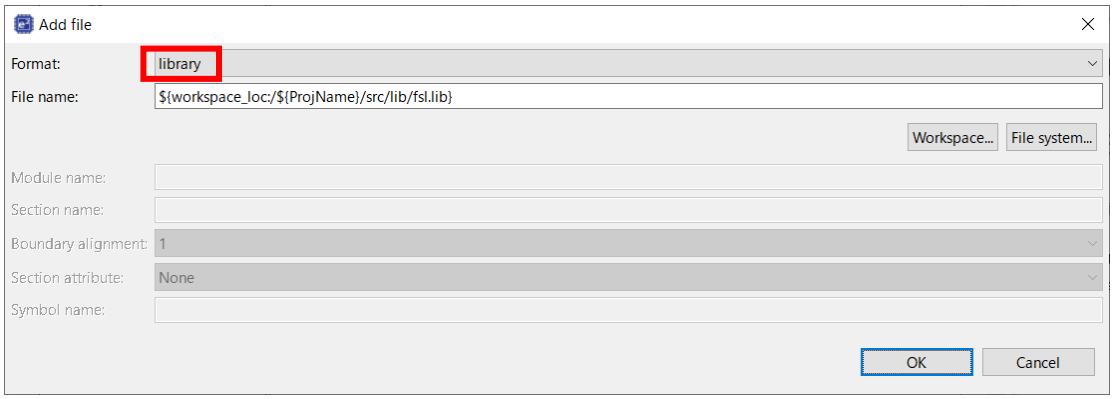


Figure 7-21 (b). Specifying the Library File (CC-RL)

(3) Building

Right-click on the [Project] in the e^2 studio project tree and select “Build Project” to build the project.

7.4 Notes at Build

7.4.1 When the CA78K0R Compiler is Used

(1) When the on-chip debugging function is in use

After the on-chip debugging function is enabled in the CS+, building a program may generate the following type of error.

```
RA78K0R error E3212: Default segment can't allocate to memory - ignored
                     Segment '??OCDROM' at xxxxxH-200H
```

This error occurs when the segment for the monitor area (OCDROM) used by the on-chip debugging function cannot be allocated. Therefore, to avoid this error, add the following code to the link directive file (*.dr) embedded in the project and prepare a separate area for allocating the segment.

```
MEMORY OCD_ROM : ( 0xxxxxH, 00200H )
```

Remarks: 1. xxxxx indicates the start address of the location where the error occurred.
2. The area name OCD_ROM is an example of the notation.

(2) When the relink function is in use (on the flash area side)

The error shown below may occur when a program is built after a file including a declaration that specifies the section name is registered in the project on the flash area side with the use of the relink function of CS+.

```
CC78K0R error E0842: Unrecognized pragma SECTION '@@xxxxx'
```

This error occurs because the section name on the flash area side differs from the normal case when the relink function is used. To avoid the error, change the specified section name from "@@xxxxx" to "@Exxxxx" as shown below to conform to the rules for the section name of the flash area side.

```
#pragma section @Exxxxx CNST_DAT
```

Remarks: 1. xxxxx indicates the string of the desired section name.
2. The changed section name CNST_DAT is an example of the notation.

7.4.2 When the CC-RL Compiler is Used

(1) When the on-chip debugging function is in use

After the on-chip debugging function is enabled in the CS+, building a program may generate the following type of error.

E0562321:Section ".monitor2" overlaps section "xxxxx"

This error occurs when the section for the monitor area (OCDROM) used by the on-chip debugging function cannot be allocated. Therefore, to avoid this error, right click the CC-RL (Build Tool) node (1) in the CS+ Project Tree window, select Property to open the CC-RL Property panel (2), and select the Link Options tab (3). In the Section category (4), modify the setting for Section start address (5) so that no other areas overlap the area where the section for the on-chip debugger monitor is allocated (monitor2: the initial address range is 0xFE00 to 0xFFFF in R5F100LE). (See Figure 7-22.)

For details of the section settings, refer to the CC-RL Compiler User's Manual.

Remark 1. xxxxx: Indicates the section name.

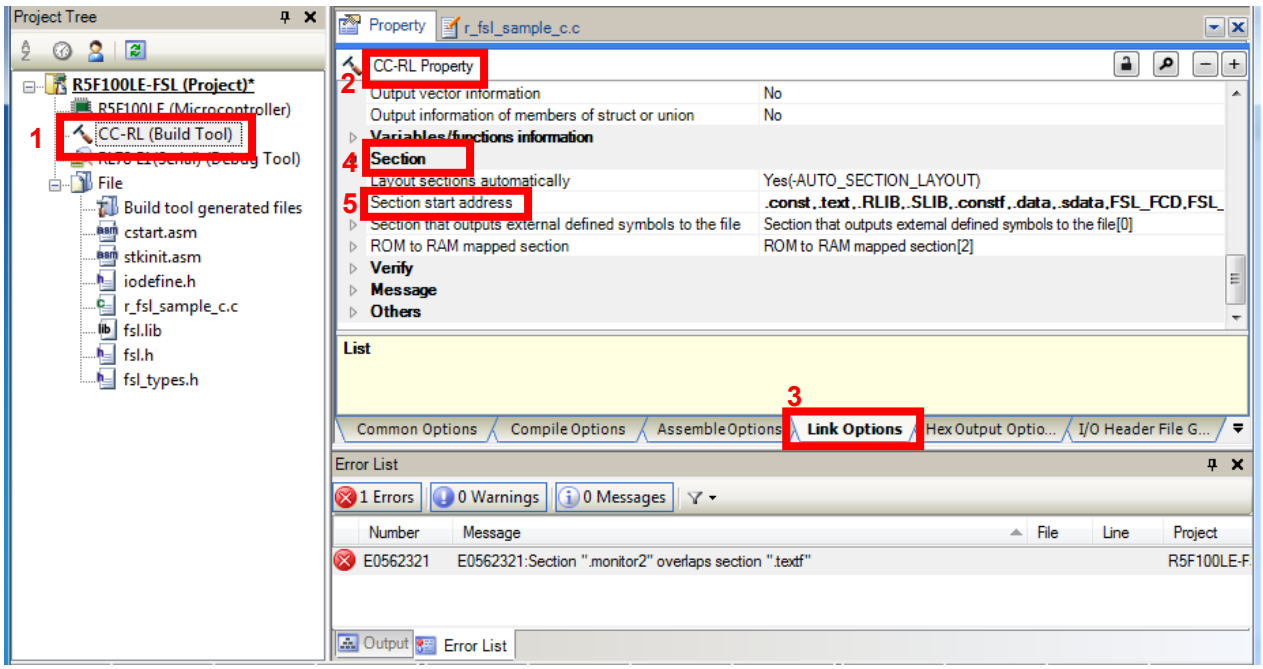


Figure 7-22. Modifying the Section Allocation

Chapter 8 How to Debug a Program

For details on how to perform debugging by using IECUBE or the on-chip debug emulator E1, E2, E2 emulator Lite or E20, see the following document:

Title
CubeSuite+ Integrated Development Environment User's Manual: RL78 Debug[CS+ for CA,CX]Note
CS+ Integrated Development Environment User's Manual: RL78 Debug Tool[CS+ for CC] Note
e² studio Integrated Development Environment User's Manual: Getting Started Guide

Note: You can download this document from the “CS+ Integrated Development Environment” page or “e² studio Integrated Development Environment” page of the Renesas Electronics website.

Chapter 9 Sample Program

The attached sample program (r_fsl_sample_c.c) is provided to enable the usage method of the Flash Self-Programming Library Type01 to be easily confirmed on the QB-R5F100LE-TB boards with R5F100LEA (RL78/G13) as the target microcontrollers. The sample program is just a reference example and the user program does not have to be created to match the sample program. The sample program should be used as a simple program to confirm operation.

- The link directive file (r_fsl_sample_c.dr) for the sample program for the CA78K0R compiler has a purpose to specify that a stack or data buffer used by the sample program is not allocated to an area where allocation is prohibited Note1. When using the sample program, this file should also be embedded with the sample program.Note2, 3
- The sample program for the CC-RL compiler, should be allocated appropriately in the section category on the “Link Options” tabbed page in the CS+ window, or on the “Linker” [Section] page in the e² studio, so that a stack or data buffer used by the sample program is not allocated to an area where allocation is prohibited Note1,3.
- The linker script file (r_fsl_sample_c.ld) for the sample program for the LLVM compiler has a purpose to specify that a stack or data buffer used by the sample program is not allocated to an area where allocation is prohibited Note1. When using the sample program, this file should also be embedded with the sample program.Note3

Notes: 1. For details, refer to chapter "2.2 Software Environment” in the user’s manual.
2. In the supplied link directive file, the RAM area size is set to 512 bytes. Even when the target microcontroller has 2 Kbytes or larger RAM, the sample program (r_fsl_sample_c.c) can be used for building without modifying the defined area setting.
3. The data in usage may be placed at an unintended area depending on how the environment in use or the program is changed. After an execution module is generated, the map file and allocation state of programs or data must be confirmed. For the definition method and allocation conditions of each code or data, refer to the user’s manual of the compiler used.

9.1 Initial Settings of the Sample Program

The sample program operates with the following initial settings. When these settings need to be changed, modify the sample program.

- CPU operating frequency: High-speed on-chip oscillator 32 MHz
- Voltage mode: High-speed mode

9.2 Settings of Option byte and On-Chip Debugging

(1) When using CA78K0R or CC-RL compiler with the CS+

When performing on-chip debug, set “Set enable/disable on-chip debug by link option” to “Yes” and specify “84” for “Option byte values for OCD”. For the CC-RL compiler, set “Set debug monitor area” to “Yes”.

The sample program normally operates by setting the high-speed on-chip oscillator at 32 MHz.

After setting “Set user option byte” to “Yes” on “Link Options” tabbed page, specify “xxxxE8” for “User option byte value” and set the high-speed on-chip oscillator at 32 MHz.

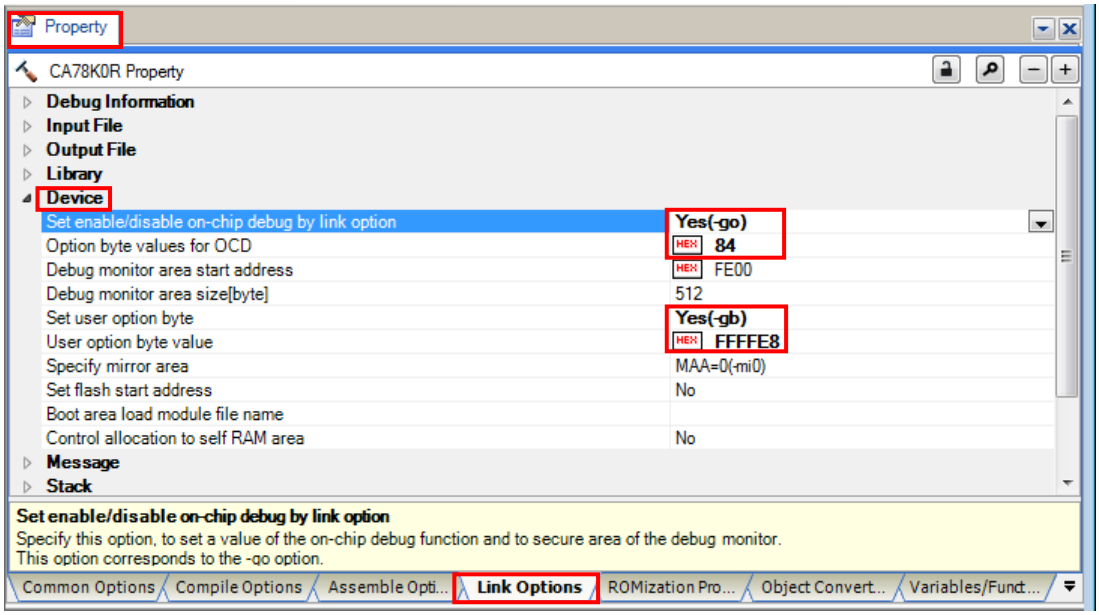


Figure 9-1 (a) Setting of Option Byte when Using the CS+ (CA78K0R Compiler)

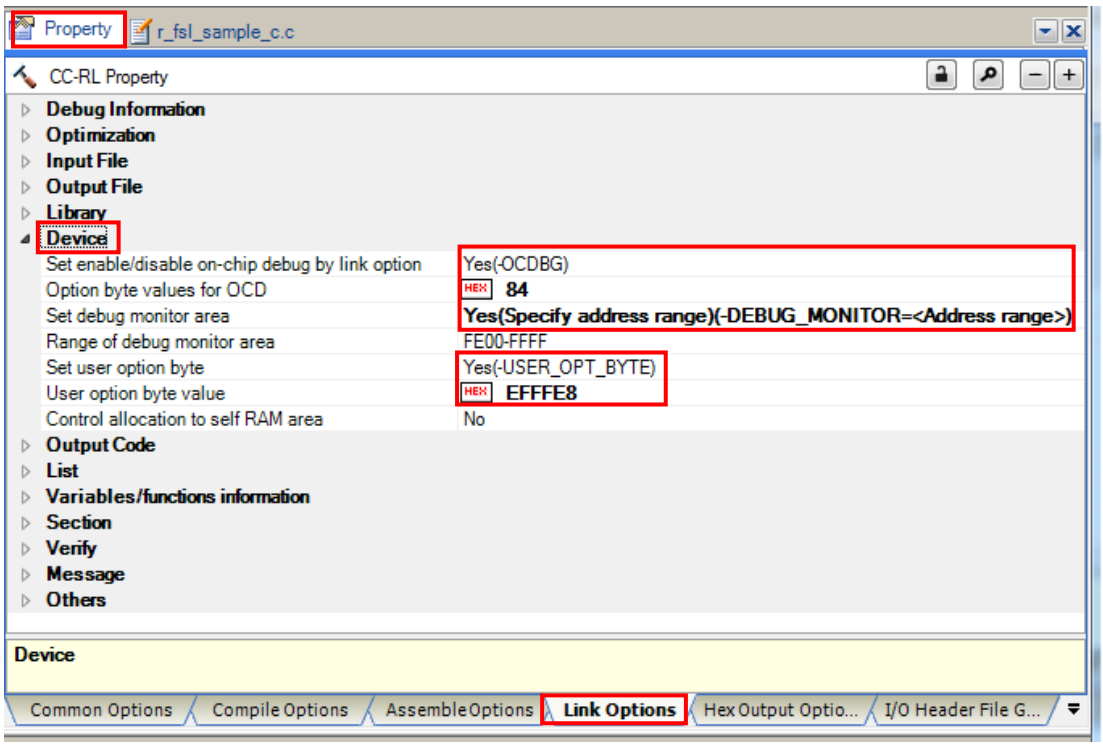


Figure 9-1 (b) Setting of Option Byte when Using the CS+ (CC-RL Compiler)

(2) When using the CC-RL compiler with the e^2 studio

Select “C/C++ Build” [Settings] - “Linker” [Device]. And set device items on the displayed screen.

When performing on-chip debug, put a check mark to “Set enable/disable on-chip debug by link option” and specify “84” for “On-chip debug control value”. Put a check mark to “Secure memory area of OCD monitor”.

The sample program normally operates by setting the high-speed on-chip oscillator at 32 MHz. Put a check mark to “Set user option byte” on the “Tool Settings” tabbed page, specify “xxxxE8” for “User option byte value” and set the high-speed on-chip oscillator at 32 MHz.

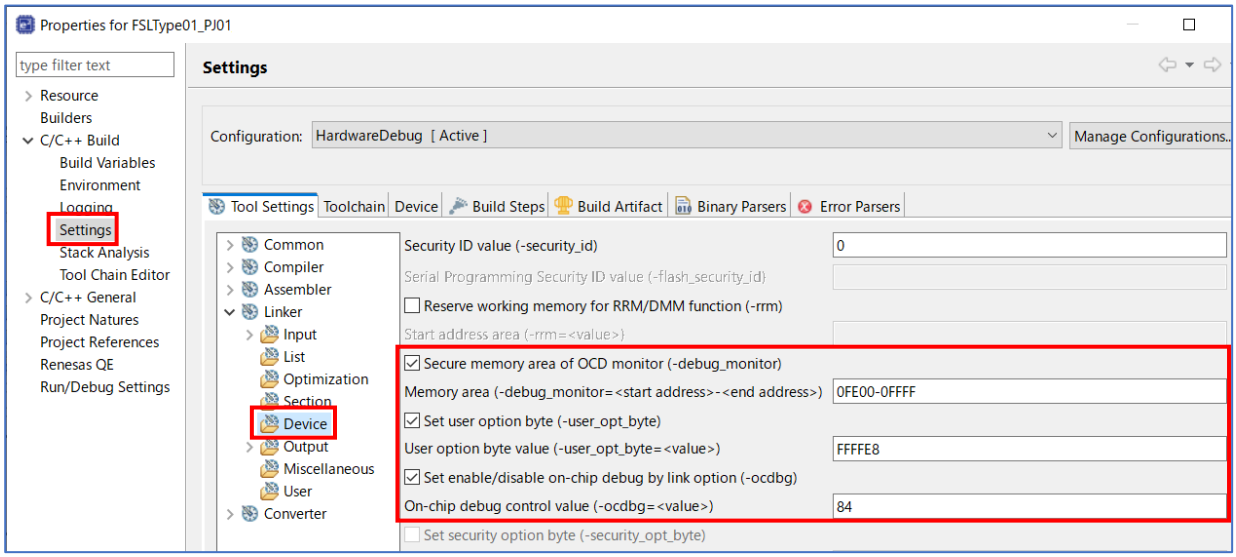


Figure 9-2. Setting of Option Byte when Using the e^2 studio (CC-RL Compiler)

(3) When using LLVM compiler with the e^2 studio

The device item settings are set in the “vects.c” file. In the “vects.c” file provided in the sample program, the option byte value and user option byte value are set in “Option_Bytes” as follows.

The sample program normally operates by setting the high-speed on-chip oscillator at 32 MHz. Therefore, set the user option byte value “xxxxe8” and the on-chip debug option byte value in the “Option_Bytes” of the “vects.c” file as follows:

[The example for RL78/G13]

“0xff, 0xff, 0xe8, 0x84” (WDT Enable, LVD reset mode, HS mode /32MHz, Enable on-chip debug operation)

```
const unsigned char Option_Bytes[]   __attribute__ ((section (".option_bytes"))) = {
        0xff, 0xff, 0xe8, 0x84
```

Note: Be sure to confirm the contents of “User option byte” of the chapter of “Option Bytes” and “On-chip debug option byte” by the user’s manual of a target device. And describe the set value used with user application.

9.3 Compilation Switch for the C-Language Sample Program

The sample program has a compilation switch as shown below. This compilation switch is used to turn on the LED to confirm operation on the QB-R5F100LE-TB board. To use this, modify "#if 0" to "#if 1" so that the #define declaration for the target CPU board becomes valid.

```
/************************************************/
/* Sample Program – Program switch symbol */
/************************************************/
/* Can be enabled when a single QB-R5F100LE-TB board is used */
#if 0
#define __QB_R5F100LE_TB__

/* Other boards */
#else
#define __NON_TARGET__
#endif
```

Can be modified to #if 1 when QB-R5F100LE-TB is used

9.4 Defining the Internal RAM Area

9.4.1 When the CA78K0R Compiler is Used

When the CA78K0R compiler is used, the entire internal RAM area is automatically defined as an area with the name "RAM" in the initial state. Unless otherwise stated in the link directive file, the stack and data buffers are to be allocated to this area Note. However, in this case, the stack and data buffers would be allocated by default to an area (FFE20H to FFEFFH in self-RAM) for which use by the flash self-programming library is prohibited, so the program may not run correctly.

In the attached link directive file for the sample program, as a solution, re-define the area with the name "RAM" so that it does not include the above area, ensuring that stack and so on are not allocated to the area for which usage is prohibited.

```
MEMORY RAM          :(0FF300H, 000B20H)
```

The above statement redefines the area with the name "RAM" to be the B20H bytes area starting from the address FF300H (FF300H to FFE1FH) Note. This prevents attempted use of the area which the flash self-programming library is prohibited to use by excluding the prohibited portion from the area with the name "RAM".

However, if this is the only change setting that is explicitly made, the area from FFE20H to FFEFFH is also unusable for any other purpose. Accordingly, separately add the following definition. No particular restrictions apply to the name of this area.

```
MEMORY SADDR_RAM:(0FFE20H, 0000E0H)
```

If there is a self-RAM area, automatic allocation of variables to this area can be restricted by defining its range as an area with the name "SELFRAM".

```
MEMORY SELFRAM    :(0FEF00H, 000400H)
```

An example of the settings for an RL78/G13 (the product with 4 Kbytes of RAM and 64 Kbytes of ROM) is given below.

```
; ----------------------------------------------------------
; Define new memory entry for Self-RAM
; ----------------------------------------------------------
MEMORY SELFRAM     : ( 0FEF00H, 000400H )   <-- Definition of the self-RAM area

; ----------------------------------------------------------
; Redefined default data segment RAM
; ----------------------------------------------------------
MEMORY RAM         : ( 0FF300H, 000B20H )   <-- Definition of the RAM area to be used normally

; ----------------------------------------------------------
; Define new memory entry for saddr area
; ----------------------------------------------------------
MEMORY RAM_SADDR : ( 0FFE20H, 0000E0H )     <-- Definition of the area from FFE20H to FFEFFH
```

Note: The CA78K0R linker allocates data with a non-specified destination for allocation (segment types DSEG and BSEG) to the internal RAM area according to the re-allocation attribute of the data. Accordingly, specific data may not be allocated to the area with the name "RAM" in some situations.
For details on the methods of defining and allocating the individual categories of data, refer to the user's manual for CS+.
Reference to the map file (*.map) generated at the time of building is required to confirm the state of allocation.

9.4.2 When the CC-RL Compiler is Used

(1) Adding the include path

In CS+ for the CC-RL compiler, no include path is specified in the initial state; the include paths for the header files used by the flash self-programming library need to be added. The flash self-programming library uses header files "fsl.h", "fsl_types.h", and "iodefine.h" (this file is automatically generated by CS+ and e^2 studio).

- In CS+, add the include path where each file resides in [Compile Options] – [Preprocessing] – [Additional Include Path].
- In e^2 studio, in the “Properties” window, add the include path where each file exists in the “Include file directories(-I)” field on the screen displayed by “C/C++ Build” [Settings] – “Compiler” [Source].

(2) Defining sections

When CS+ and e^2 studio for the CC-RL compiler is used, the sections used for the ROM and RAM areas need to be defined.

- Sections can be defined in the Section category on the Link Options tabbed page in the CS+ window. When the Layout sections automatically property is set to No, select the Section start address property to open the Section Settings dialog box and add the sections necessary for the flash self-programming library to the ROM area (Figure 9-3). (In this example, the FSL_FCD, FSL_RCD, FSL_BCD, and FSL_BECD sections that are necessary for operation of the sample program are added.)

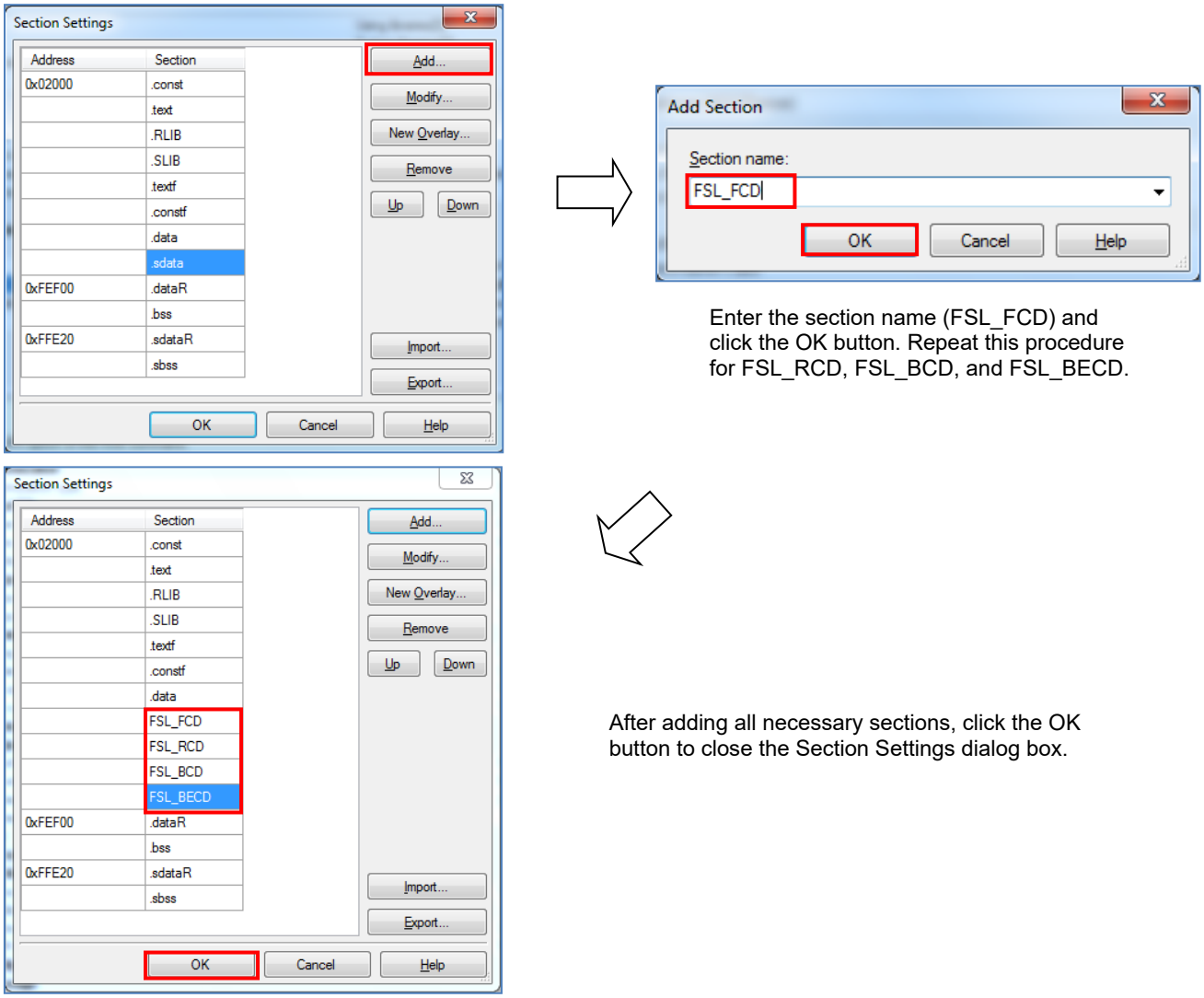


Figure 9-3. Example of Section Settings for the Flash Self-Programming Library when Use CS+(ROM Area)

- Setting of the section items on e^2 studio inputs in the “Properties” window. Select “C/C++ Build” [Setting] - “Linker” [Section]. And set section items on the displayed screen. Remove a check mark to [Layout sections automatically(-auto_section_layout)]. Press the “ ... ” button of the right-hand side which sections are displaying, and a “Section Viewer” screen is displayed and add the sections necessary for the flash self-programming library to the ROM area (Figure 9-4). (In this example, the FSL_FCD, FSL_RCD, FSL_BCD, and FSL_BECD sections that are necessary for operation of the sample program are added.)

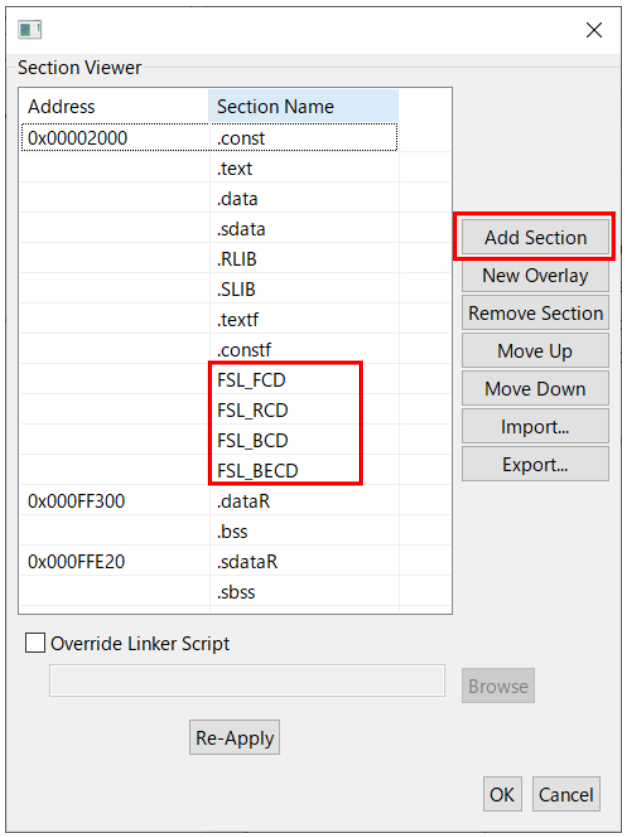


Figure 9-4. Example of Section Settings for the Flash Self-Programming Library when Use e^2 studio (ROM Area)

(3) Allocating the Self-RAM Area

In the initial state of the section settings in CS+ for the CC-RL compiler, the user RAM area is allocated at the beginning of the internal RAM area (from address FEF00H for R5F100LEA, which is the target microcontroller of the sample program). However, in R5F100LEA, the flash self-programming library uses the address range from 0xFEF00 to 0xFF2FF as the self-RAM area. Therefore, the user RAM area must be allocated outside this area. In this example, the user data start address 0xFEF00 is changed to 0xFF300.

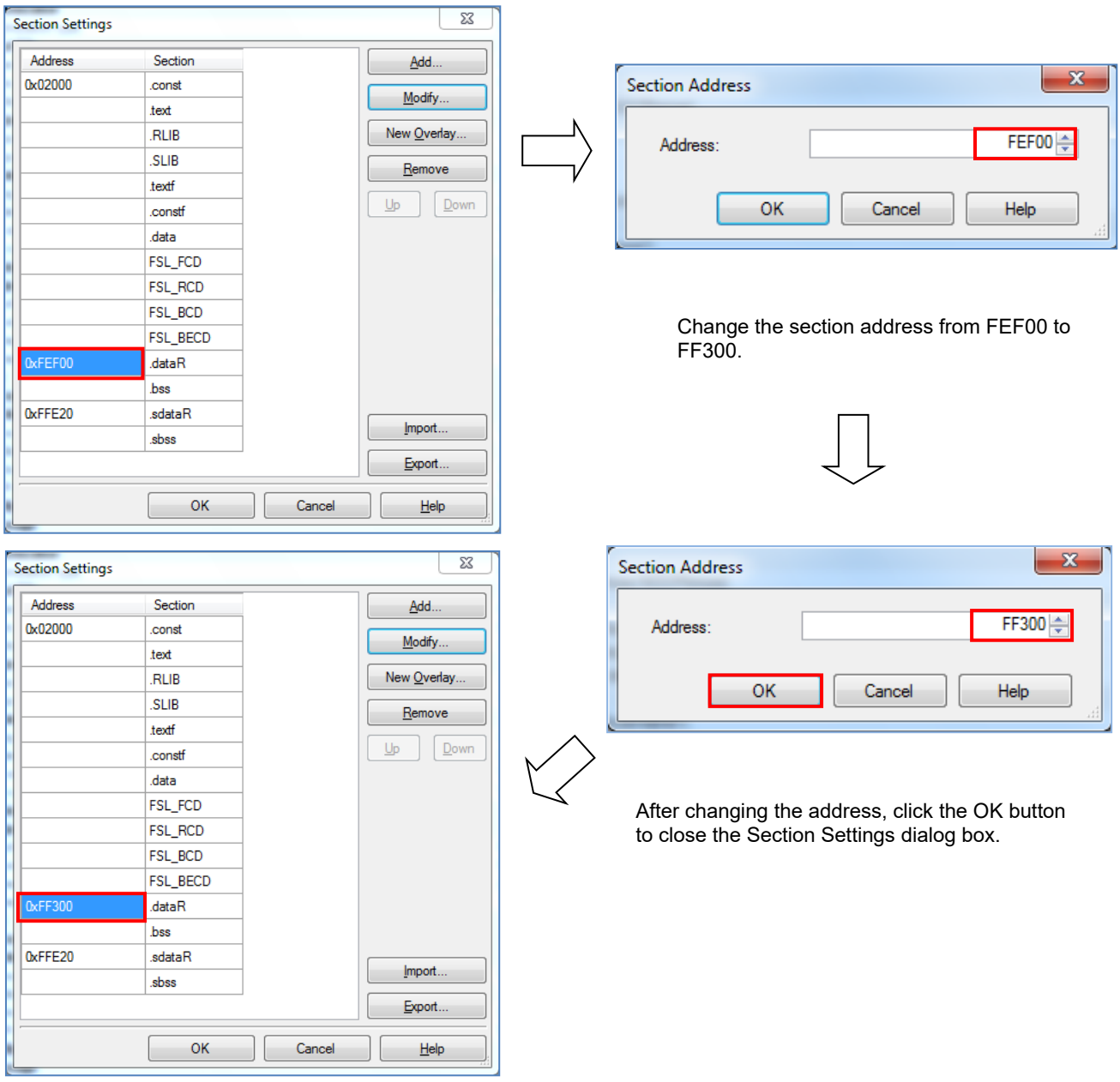


Figure 9-5. Example of Changing the User RAM Area Allocation When Use CS+ (RAM Area)

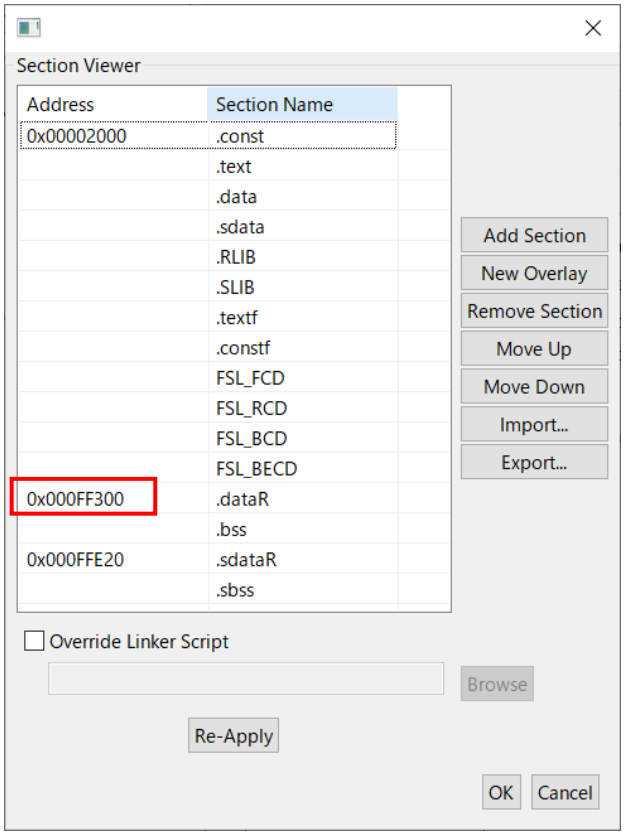


Figure 9-6. Example of Changing the User RAM Area Allocation when Use e² studio (RAM Area)

Note: The sections including the user-specified sections are automatically re-allocated when the Layout sections automatically property is temporarily set to No, the user RAM allocation is changed, and then the property is again set to Yes. In this case, sections may be allocated to areas that are not specified by the user; that is, data may be placed in unintended areas. Be sure to refer to the map file to check if the software resources (especially RAM data) used by the flash self-programming library are placed in relocatable areas.

9.4.3 When the LLVM Compiler is Used

(1) Adding the include path

In e² studio, no include path is specified in the initial state: The include path for the header files used by the flash self-programming library need to be added. The flash self-programming library uses header files “fsl.h”, “fsl_types.h”, and “iodefine.h” and “iodefine_ext.h” (this file is automatically generated by e² studio).

In e² studio, in the “Properties” window, add the include path where each file exists in the “Include file directories(-I)” field on the screen displayed by “C/C++ Build” [Settings] – “Compiler” [Includes].

(2) Allocating the Self-RAM area

The LLVM compiler describes the link settings to be performed in the build in a linker script file (*.ld).
In the linker script file (linker_script.ld) output from e^2 studio, the built-in RAM area is defined as “RAM” section. In addition, the software resources used by the flash self-programming library are defined as an area called “SELFRAM” section. (Only for devices that require “Self-RAM” area)
In the linker script file “r_fsl_sample_c.ld” included with the sample program, the “RAM” section and “SELFRAM” section are defined so that they do not overlap.

Note: The “r_fsl_sample_c.ld” provided in the sample program is prepared on the assumption that R5F100LE will be used. When using other devices, please check the Self-RAM list and modify it according to the device.
Refer to each reference manual of LLVM about the descriptive content of linker script file (*.ld), and the details of the description method.

All trademarks and registered trademarks are the property of their respective owners.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan

www.renesas.com

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/.

© 2023 Renesas Electronics Corporation. All rights reserved.