RH850 Family C Compiler Package (CC-RH)

How to Divide Boot and Flash Areas

R20UT4672EJ0100
Rev. 1.00
Dec 18,2019

Introduction

This document describes the processing necessary to divide a program into boot and flash areas when using the C compiler for the RH850 family (CC-RH).

Versions of Tools with which Correct Operation has been Confirmed

The following tools and versions were used for the descriptions in this document.

- C compiler for the RH850 family (CC-RH): V2.01.00
- CS+ for CC integrated development environment: V8.02.00

Contents

1. Overview

1.1 Dividing the boot and flash areas

The purpose of dividing the boot and flash areas is to ensure that only the program in the flash area can be modified without reconfiguring the program in the boot area.

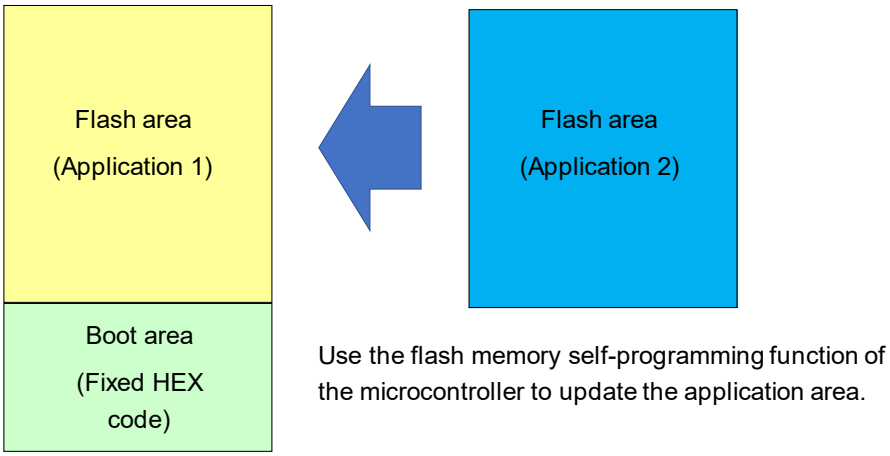

Figure 1 Divided Areas on System

Note: In this document, the boot area is defined as an area that cannot be modified following design of the system while the flash area is defined as an area that can be modified or replaced on the system.

To divide the boot and flash areas, create two projects, one to be used as the boot area project and the other to be used as the flash area project. These projects must satisfy the following conditions.

- The variables and functions in the boot area are accessible from the flash area.
  - The linker option -FSymbol should be used for the boot area project so that externally defined symbols will be output in a file.
  - The above externally defined symbol file should be specified as a target of building in the flash area project.
- The functions in the flash area can be called from the boot area through a function table.
  - When calling functions in the flash area, the boot area project should call the address of each branch instruction for a function that is specified in the function table.
  - A table of branch instructions for functions to be called from the boot area project should be created in the flash area project.

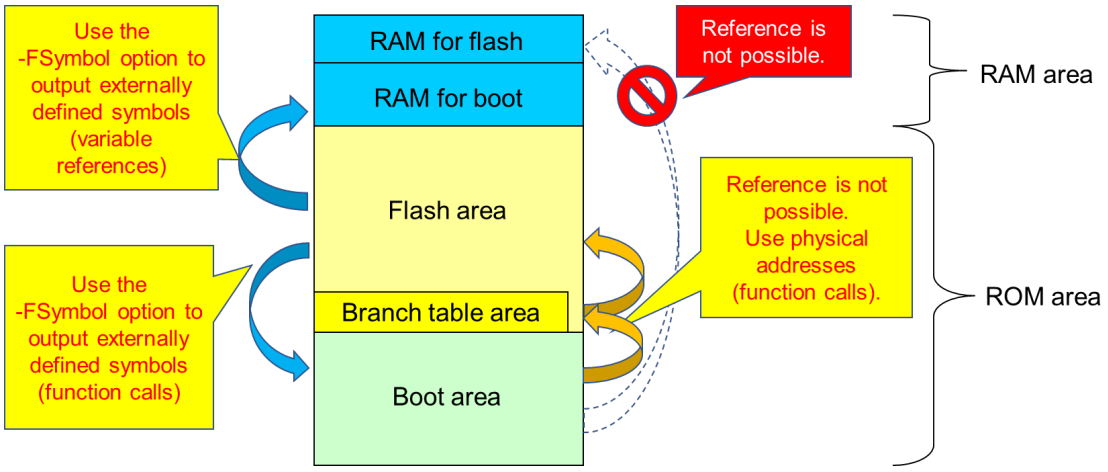

Figure 2 References to Variables and Functions between the Boot and Flash Areas

1.2 Allocating the boot and flash areas

Allocate the boot and flash areas as follows.

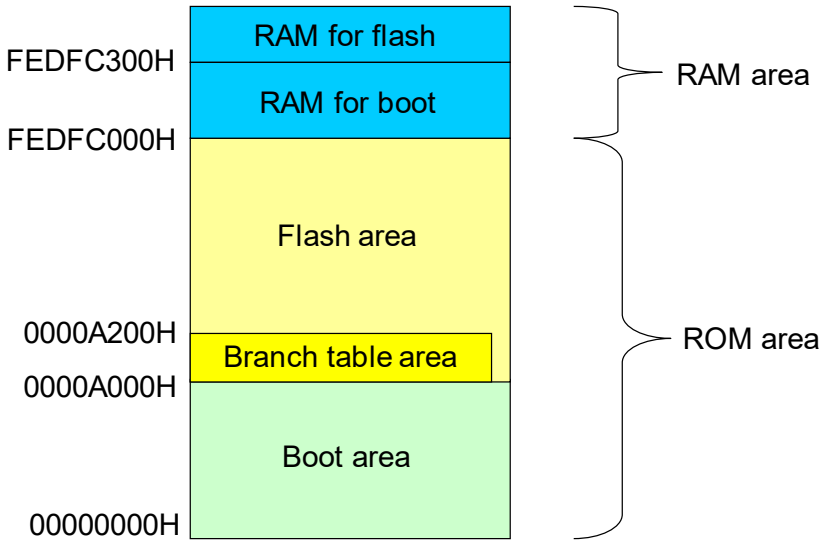

Figure 3 Example of Allocating the Boot and Flash Areas

1.3 Overview of build processing for the boot and flash areas

Figure 4 shows an overview of build processing for the boot and flash areas.

Build processing for the flash area project requires symbol information on external variables and functions which are generated in building the boot area project. Accordingly, build the boot area project before the flash area project.

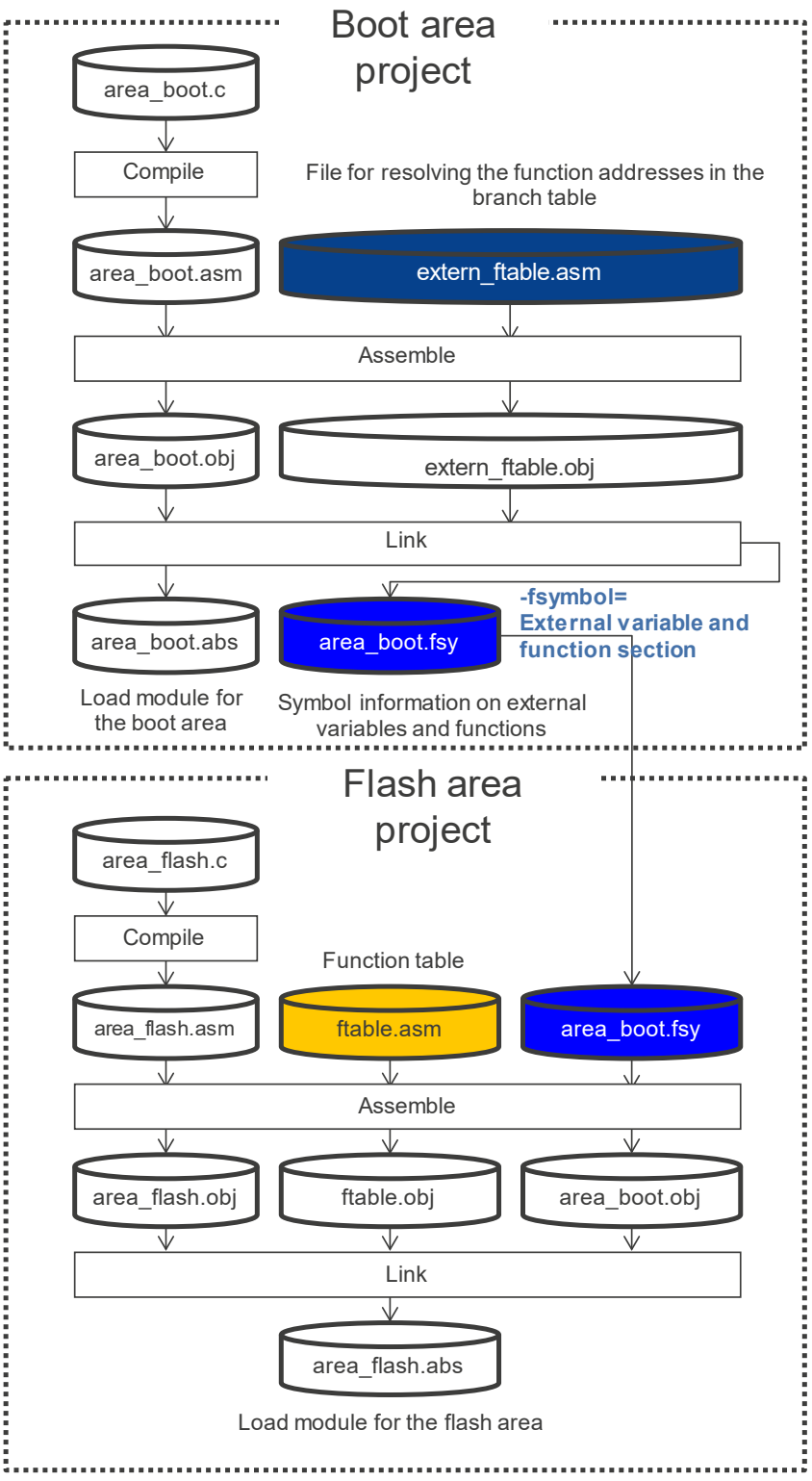

Figure 4 Build Processing for the Boot and Flash Areas

2. Common Processing for the Boot and Flash Areas

2.1 Creating projects

2.1.1 Creating the main project and the sub-project

Create the flash area project as the main project and the boot area project as a sub-project*.

Note: The build order in CS+ should be [Sub-project] -> [Main project].
The boot area program will not be modified once it has been created. Therefore, when creating the second- or a later generation flash area project, the sub-project can be deleted.

2.1.2 Excluding the automatically generated file from the targets of building

(1) Exclude the following file from the boot area project.
- main.c

(2) Exclude the following files from the flash area project.
- main.c
- boot.asm

2.1.3 Adding files as targets of building

(1) Add the following files to the boot area project.
- area_boot.c
- extern_ftable.asm
- ftable.inc*1

(2) Add the following files to the flash area project.
- area_flash.c
- ftable.asm
- ftable.inc
- area_boot.fsy*2

Notes: 1. This file is used in common as ftable.inc for the flash area project.
2. This file is added after the boot area project is built.

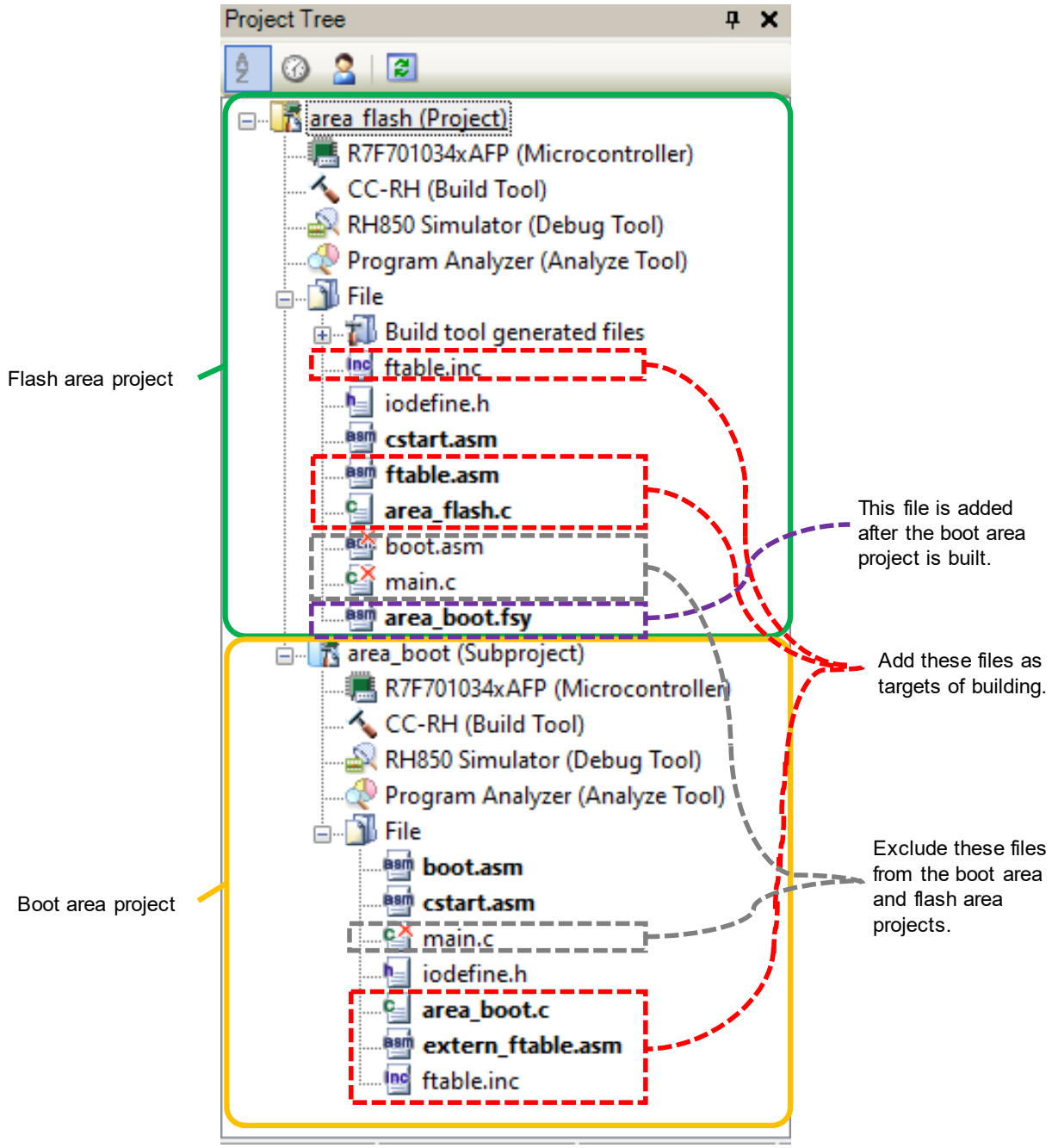

Figure 5 Example of Creating Projects

2.2 Creating a common program for the boot and flash areas

2.2.1 Address definition file for the branch table (assembly language)

- Create ftable.inc, which is the address definition file for the branch table for reference from both the boot and flash areas.
  - FLASH_TABLE: Start address of the branch table
  - INTERRUPT_OFFSET: Size of the interrupt area in the branch table

Example: ftable.inc

```
FLASH_TABLE        .EQU  0xA000
INTERRUPT_OFFSET   .EQU  0x100
```

2.3 Hex files for the boot and flash areas

File names used in this document are listed below (output procedures are described later).

- Hex file for the boot and flash areas combined: boot_flash.mot
- Hex file for the flash area: flashA000_ffff.mot
- Hex file for the boot area: boot0000_9fff.mot

Note: A load module file (*.abs) is separately generated for each of the boot and flash areas.

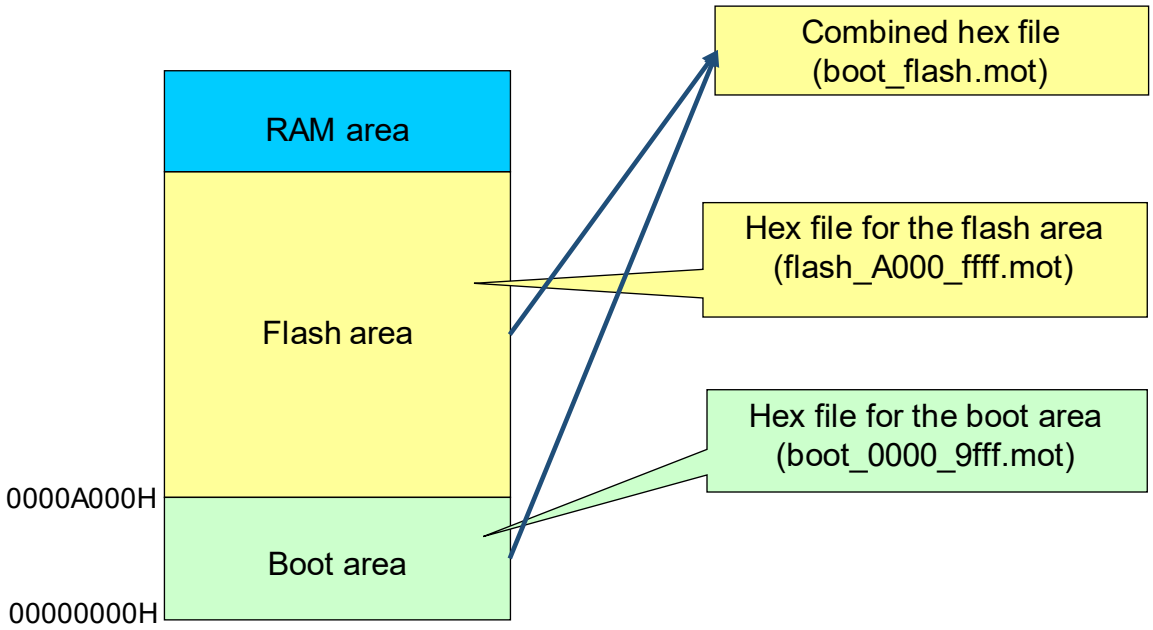

Figure 6 Hex Files for the Boot and Flash Areas

2.4 Initialization procedure

Figure 7 shows the initialization procedure.

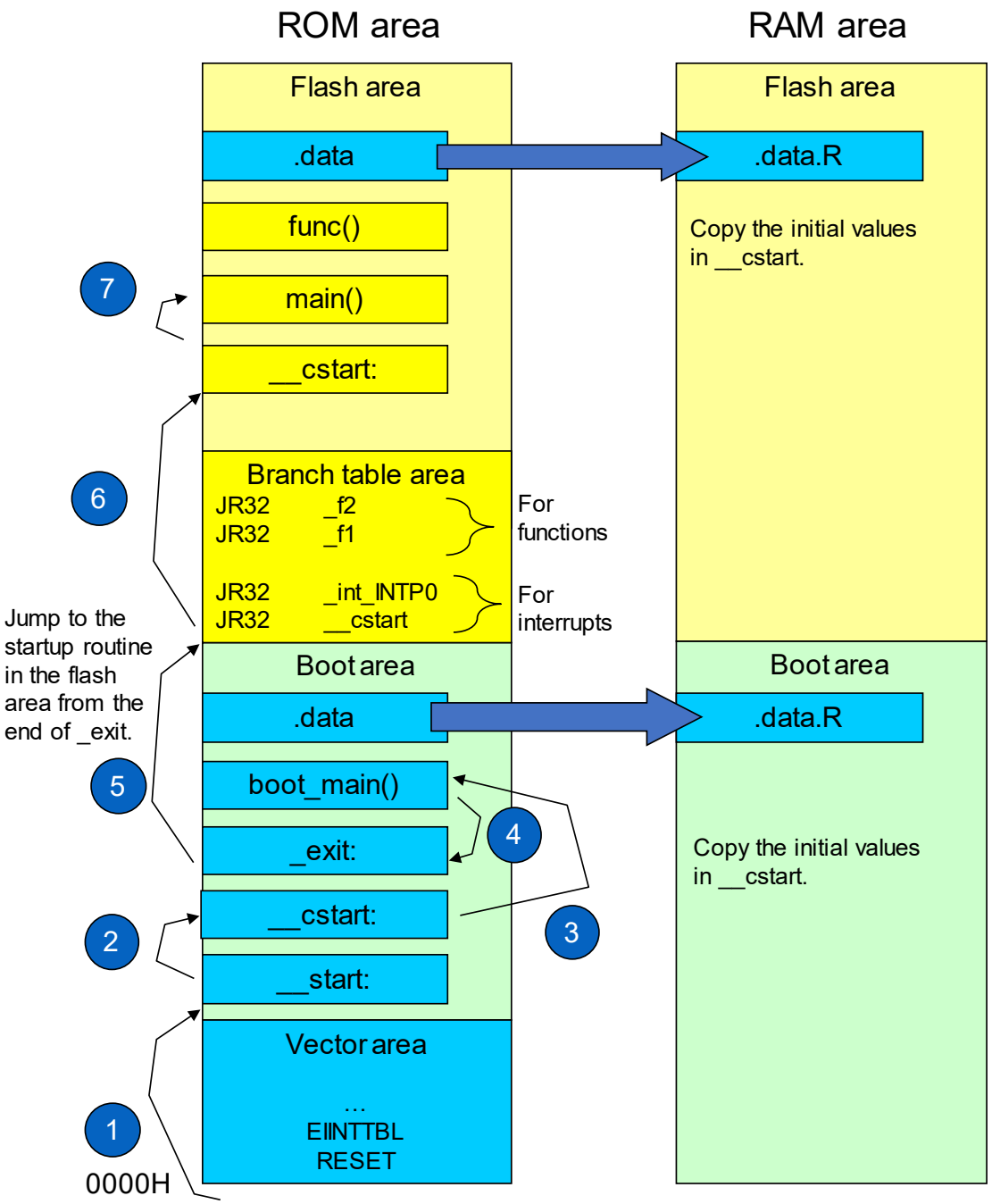

Figure 7 Initialization Procedure

3. Boot Area

3.1 Creating boot area programs

The following steps are required for boot area programs.

- Modifying the startup routine
- Modifying the interrupt and exception vector format
- Creating a file for resolving the function addresses in the branch table

3.1.1 Modifying the startup routine (cstart.asm)

The procedure for modifying the startup routine (cstart.asm) is given below.

1. Add inclusion of the address definition file for the branch table.

Example: Modifying cstart.asm (1/5)

```
$INCLUDE "ftable.inc"

;-------------------------------------------------------------------------------
;    system stack
```

2. Modify the section name to exclude it from the target of the -FSymbol option (which is used to output externally defined symbols).

Example: Modifying cstart.asm (2/5)

```
;-------------------------------------------------------------------------------
;    startup
;-------------------------------------------------------------------------------
     .section    ".btext", text
     .public     __cstart
```

3. Comment out the section where use of the FPU has been set if the FPU is not to be used from the boot area. In such a case, make the initial settings for the FPU in the startup routine for the program in the flash area.

Example: Modifying cstart.asm (3/5)

```
     ; enable FPU
$if 0 ; disable this block when not using FPU
     stsr  6, r10, 1    ; r10 <- PID
     shl   21, r10
     shr   30, r10
     bz    .L1               ; detecting FPU
     stsr  5, r10, 0         ; r10 <- PSW
     movhi 0x0001, r0, r11
     or    r11, r10
     ldsr  r10, 5, 0         ; enable FPU

     movhi 0x0002, r0, r11
     ldsr  r11, 6, 0         ; initialize FPSR
     ldsr  r0, 7, 0          ; initialize FPEPC
.L1:
$endif
```

4. Allow the generation of exceptions if interrupt functions are to be used.

Example: Modifying cstart.asm (4/5)

```
      ; set various flags to PSW via FEPSW

      stsr  5, r10, 0          ; r10 <- PSW
      xori  0x0020, r10, r10   ; enable interrupt
      ;movhi      0x4000, r0, r11
      ;or   r11, r10           ; supervisor mode -> user mode
      ldsr  r10, 3, 0          ; FEPSW <- r10
```

5. Modify the main function call to the call to the main function of the boot area program. In addition, modify the setting for the r31 register (lp) to FLASH_TABLE to cause a branch to the flash area startup routine after the main function in the boot area has been executed.

Example: Modifying cstart.asm (5/5)

```
      mov   FLASH_TABLE, lp          ; lp <- FLASH_TABLE
      mov   #_boot_main, r10
      ldsr  r10, 2, 0                ; FEPC <- #_boot_main
      ; apply PSW and PC to start user mode
      feret
_exit:
      br    _exit                    ;  end of program
```

3.1.2 Modifying the interrupt and exception vector format (boot.asm)

The procedure for modifying the interrupt and exception vector format (boot.asm) is given below.

1. Add inclusion of the address definition file for the branch table.

Example: Modifying boot.asm (1/4)

```
$INCLUDE "ftable.inc"

      ; if using eiint as table reference method,
```

2. Change the method of jumping in response to interrupts to table reference.

Example: Modifying boot.asm (2/4)

```
      ; if using eiint as table reference method,
      ; enable next line's macro.

      USE_TABLE_REFERENCE_METHOD .set 1
```

3. Set the addresses where the table for branching to the interrupt functions in the flash and boot areas are allocated. The following example shows code for the int_INTP0 interrupt function in the flash area to be executed in response to channel 0 “EIINT0” and the int_INTP2 interrupt function in the boot area to be executed in response to channel 2 “EIINT2”.

Example: Modifying boot.asm (3/4)

```
      .section "EIINTTBL", const
      .align      512
      .dw         FLASH_TABLE + 0x10      ; INT0
      .dw         #_Dummy_EI              ; INT1
      .dw         #_int_INTP2             ; INT2
      .rept       512 - 3
      .dw         #_Dummy_EI              ; INTn
      .endm
```

4. Modify the section name to exclude it from the target of the -FSymbol option (which is used to output externally defined symbols).

Example: Modifying boot.asm (4/4)

```
;-------------------------------------------------------------------------------
;     startup
;-------------------------------------------------------------------------------
      .section    ".btext", text
      .align      2
__start:
```

3.1.3 Creating a file for resolving the function addresses in the branch table (extern_ftable.asm)

- Creating a file for resolving the function addresses in the branch table (assembly language)
  - Define symbols for resolving the addresses for the branch table to be used to call functions in the flash area from the boot area.
  - Register this file in the project.

Example: Creating extern_ftable.asm

```
$INCLUDE "ftable.inc"
        .public      _f1
_f1     .equ   (FLASH_TABLE + INTERRUPT_OFFSET + (0 * 0x10))
        .public      _f2
_f2     .equ   (FLASH_TABLE + INTERRUPT_OFFSET + (1 * 0x10))
```

3.2 Specifying boot area options

Make the following option settings for the boot area.

- Output of a file for the externally defined symbols
- Specifying the section allocation
- Specifying hex file output only to the boot area address range

3.2.1 Output of a file for the externally defined symbols

The externally defined symbols need to be output to a file so that the flash area project has access to the variables and functions in the boot area.

Register all target sections with the -FSymbol option.

Example:

[CC-RH (Build Tool)]→[Link Options] tabbed page

→[Section]→[Section that outputs external defined symbols to the file]

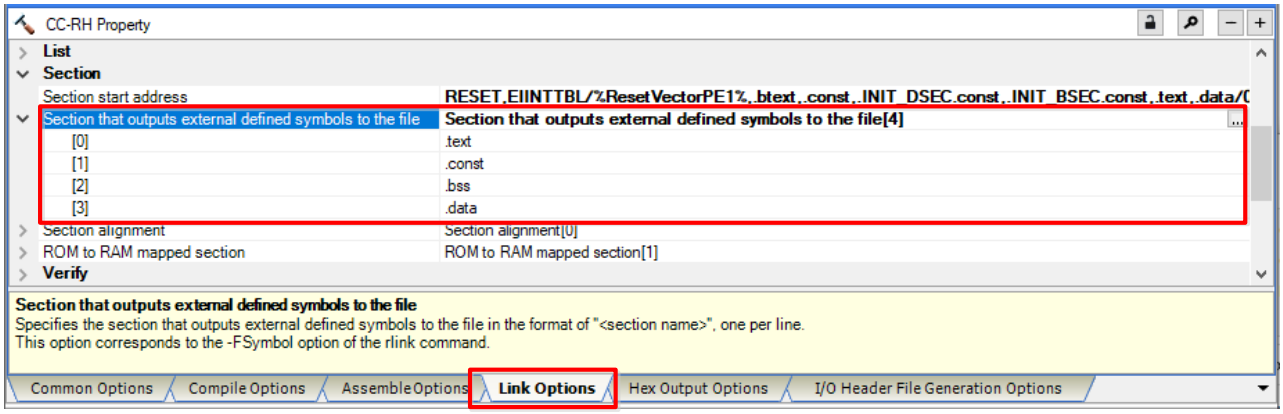

Figure 8 Example of Option Setting

3.2.2 Specifying the section allocation

Specify the section allocation in the boot area with the linker option -start. Make sure that the sections do not overlap those in the flash area.

In addition, specify the stack area section.

Example:

[CC-RH (Build Tool)]→[Link Options] tabbed page

→[Section]→[Section start address]

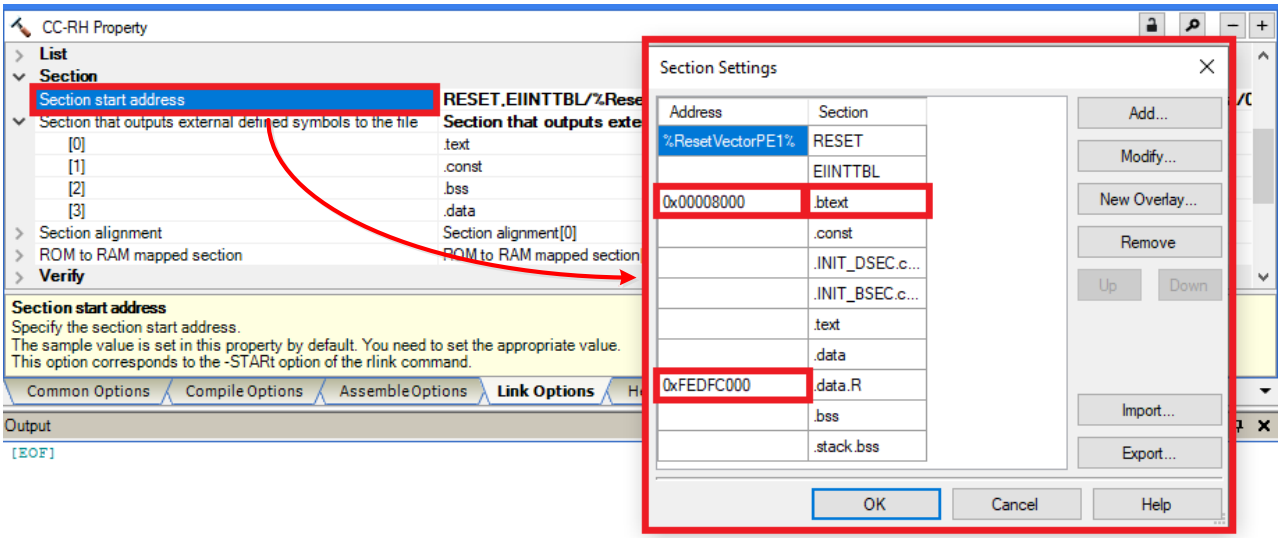

Figure 9 Example of Option Setting

3.2.3 Specifying hex file output only to the boot area address range

Specify the output file name and output addresses.

[CC-RH (Build Tool)]→[Hex Output Options] tabbed page

→[Output File]→Specify the output file name and output addresses in [Division output file].

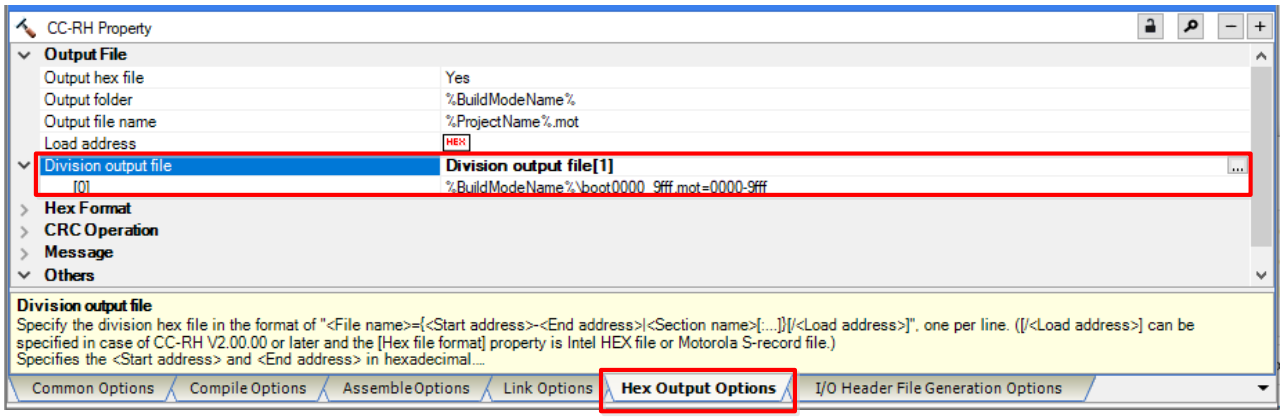

Figure 10 Example of Option Setting

4. Flash Area

4.1 Creating flash area programs

The following steps are required for flash area programs.

- Modifying the startup routine
- Creating a branch table program
- Defining an interrupt function

4.1.1 Modifying the startup routine (cstart.asm)

1. Comment out the base register settings. The base register specified in the boot area startup routine should be used; a base register must not be specified again in the flash area.*

Note: Changing the GP and EP registers from the flash area also changes addresses for reference from the boot area. We recommend standardizing the values of the GP and EP registers for use in the programs in the flash and boot areas as a whole.

Example: cstart.asm (1/2)

```
__cstart:
;       mov    #_stacktop, sp          ;  set sp register
;       mov    #__gp_data, gp          ;  set gp register
;       mov    #__ep_data, gp          ;  set ep register
```

2. Add instructions for branching to the main function in the flash area. When exception processing is to be defined, specify the user mode.

Example: cstart.asm (2/2)

```
        ; set various flags to PSW via FEPSW

        stsr   5, r10, 0          ; r10 <- PSW
        ;xori 0x0020, r10, r10   ; enable interrupt
        movhi 0x4000, r0, r11
        or     r11, r10           ; supervisor mode -> user mode
        ldsr   r10, 3, 0          ; FEPSW <- r10
        ;mov   #_exit, lp         ; lp <- #_exit
        ;mov   #_main, r10
        ;ldsr r10, 2, 0           ; FEPC <- #_main

        ; apply PSW and PC to start user mode
        ;feret
        jarl   _main, lp

_exit:
        br     _exit              ;  end of program
```

4.1.2 Creating a branch table program (ftable.asm)

At the addresses called from the boot area, write instructions for branching to the function addresses in the flash area where symbols are defined by extern_ftable.asm that is registered with the project in the boot area.

Example: ftable.asm

```
$INCLUDE "ftable.inc"

      .EXTERN     __cstart
      .EXTERN     _f1
      .EXTERN     _f2

.jtext      .CSEG text
      .ORG  FLASH_TABLE
      jr32        __cstart    ; RESET
      .align 16
      jr32        _int_INTP0  ; INTP0
.jtext2     .CSEG text
      .ORG  FLASH_TABLE+INTERRUPT_OFFSET
      jr32        _f1
      .align 16
      jr32        _f2
```

For interrupts

For functions

4.1.3 Defining an interrupt function

Example: area_flash.c (excerpt)

```
#pragma interrupt int_INTP0(channel=0)

- Omitted -

volatile char f;

- Omitted -

void int_INTP0(void)
{
      f = 1;
}
```

4.2 Specifying flash area options

Make the following option settings for the flash area.

- Registering the externally defined symbol file with the project
- Specifying the section allocation
- Specifying hex file output only to the flash area address range
- Combining the hex files for the boot and flash areas

4.2.1 Registering the externally defined symbol file with the project

Register the externally defined symbol file created in the boot area with the project to allow access to the variables and functions in the boot area.

Example:

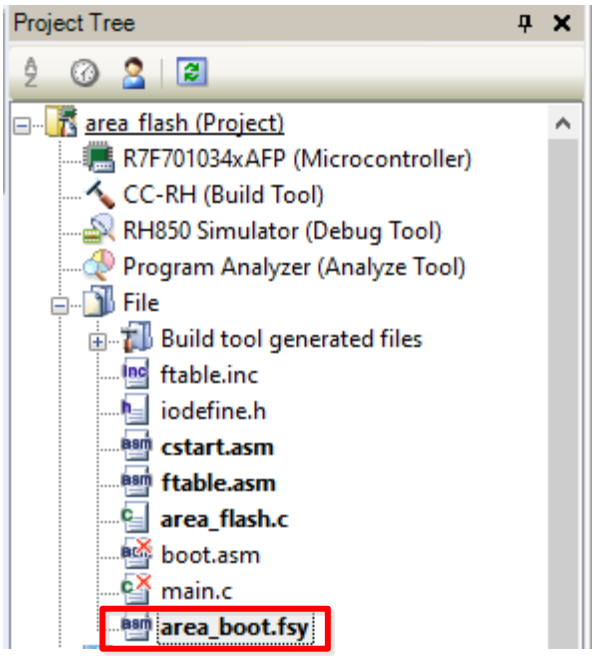

Figure 11 Example of Option Setting

4.2.2 Specifying the section allocation

Specify the section allocation in the flash area with the linker option -start.

- Make sure that the sections do not overlap those in the boot area.
- Do not allocate anything to the branch table area.
- RESET and EINTTBL sections are not required.

Example:

[CC-RH (Build Tool)]→[Link Options] tabbed page

→[Section]→[Section start address]

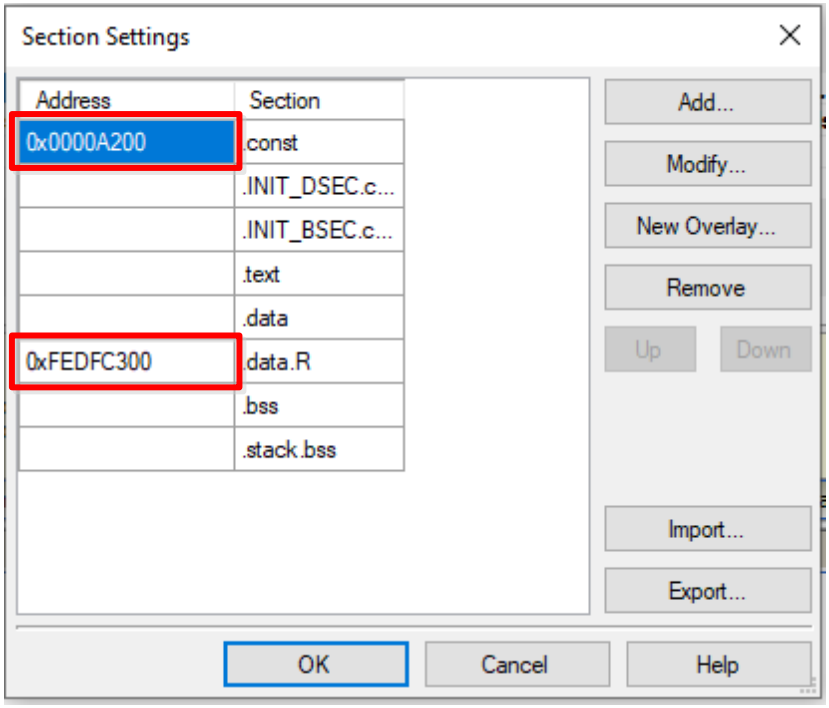

Figure 12 Example of Option Setting

4.2.3 Specifying hex file output only to the flash area address range

Specify the output file name and output addresses.

Example:

[CC-RH (Build Tool)]→[Hex Output Options] tabbed page

→[Output File]→Specify the output file name and output addresses in [Division output file].

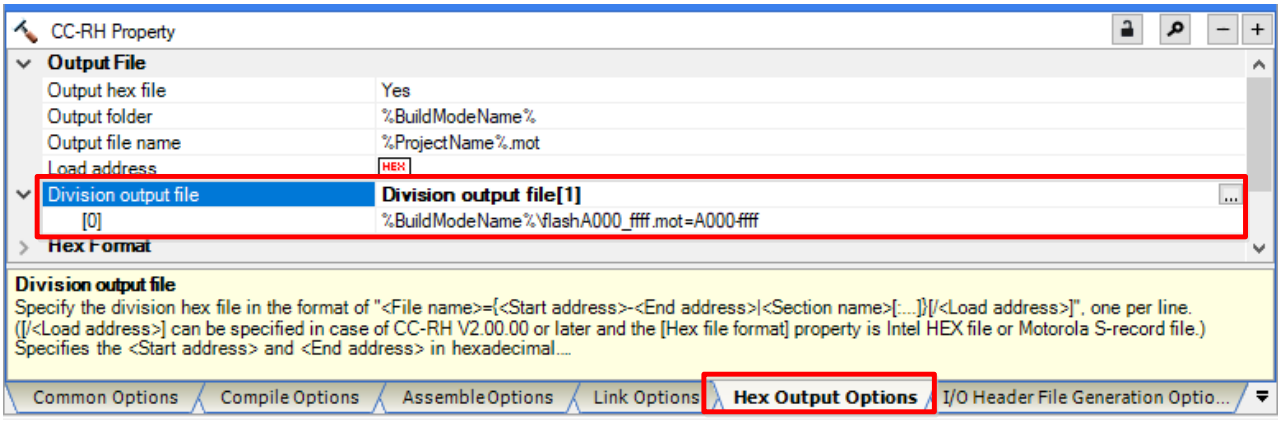

Figure 13 Example of Option Setting

4.2.4 Combining the hex files for the boot and flash areas

To combine the hex files for the boot and flash areas into one file, add the linker execution step after the build processing.

Example:

[CC-RH (Build Tool)]→[Common Options] tabbed page→[Others]
→Add the command to execute the linker ("%MicomToolPath%¥CC-RH¥V2.01.00¥bin¥rlink.exe" -subcommand=sub_mot.txt) to [Commands executed after build processing].

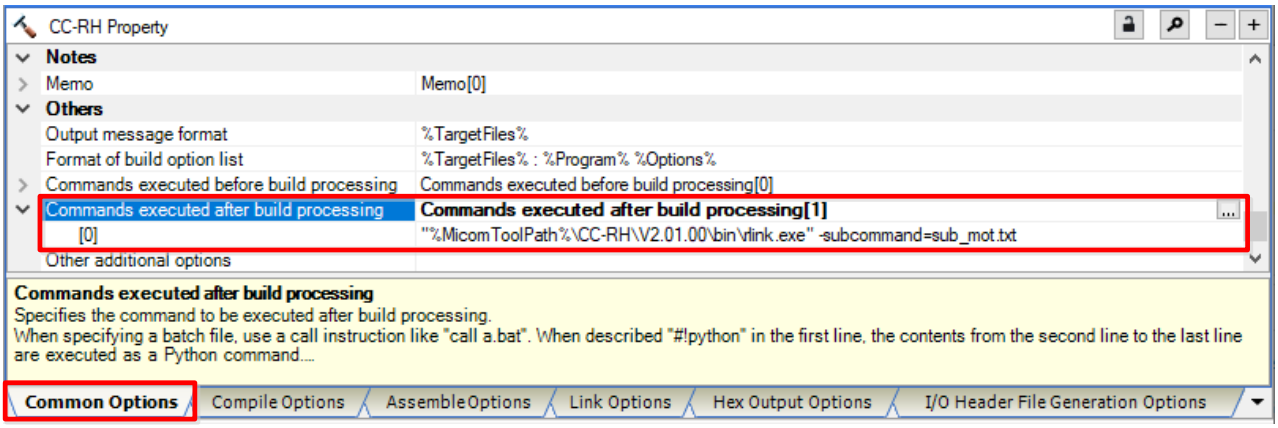

Figure 14 Example of Option Setting

Specify the input hex files, their format, and the output file name in the subcommand file for input to the linker.

Example: sub_mot.txt

```
-input=.¥area_boot¥DefaultBuild¥boot0000_9fff.mot
-input=.¥DefaultBuild¥flashA000_ffff.mot
-form=stype
-output=.¥DefaultBuild¥boot_flash.mot
```

5. Debugging Tool

5.1 Downloading to debugging tool

Two load module files (*.abs) are generated; one for each of the boot and flash areas. Download both of the load module files to the debugging tool.

Example:

[RH850 Simulator (Debug Tool)]→[Download File Settings] tabbed page

→[Download]→[Download files]

Add the load module file for the boot area to the project for the flash area.

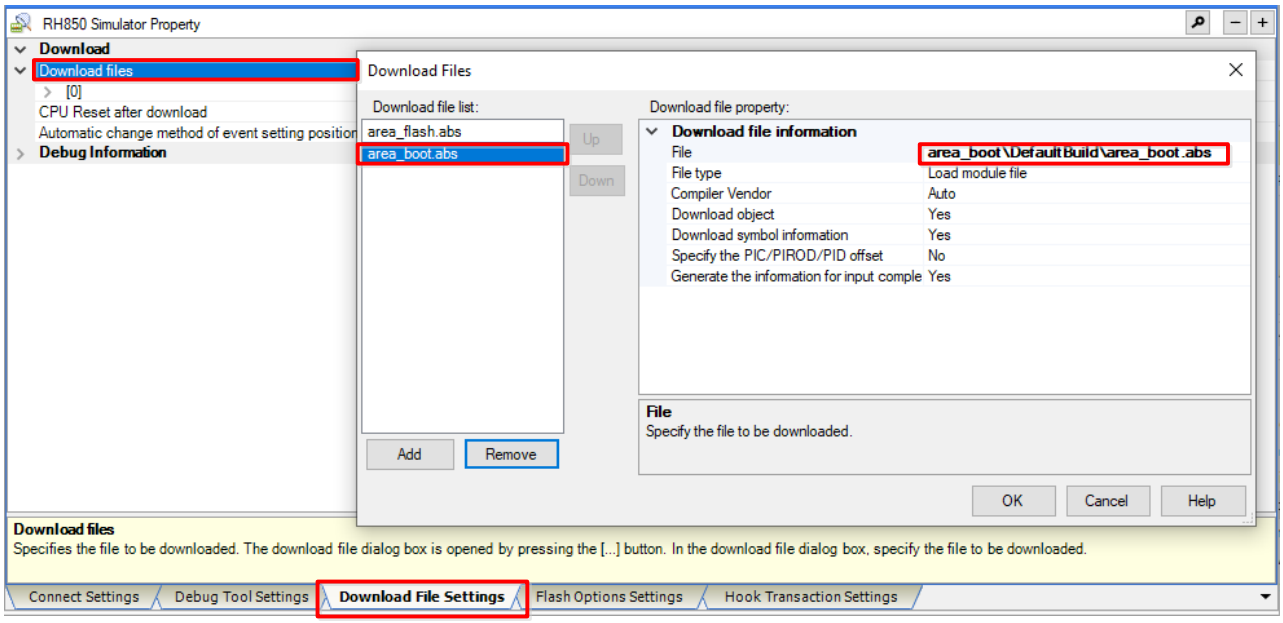

Figure 15 Example of Option Setting

6. Sample Programs

The following pages show examples of boot and flash area programs that were created through the procedures described in earlier sections.

6.1 Sample program for the boot area (area_boot.c)

```
#include "iodefine.h"

#pragma interrupt int_INTP2 (channel=2)    /* Interrupt definition in the boot
area */

int boot_a = 0x12;
int boot_b = 0x34;
extern int f1(int);      /* Prototype declaration of a function in the flash
area */
extern int f2(int);      /* Prototype declaration of a function in the flash
area */
void boot_main(void)     /* Main function in the boot area */
{
      /* Main processing in the boot area */
}
void boot_func(void)
{
  boot_a = f1(boot_a);   /* Call of a function in the flash area */
  boot_b = f2(boot_b);   /* Call of a function in the flash area */
}
void int_INTP2(void)     /* Interrupt processing in the boot area */
{
  boot_a = 1;
}
```

6.2 Sample program for the flash area (area_flash.c)

```
#include "iodefine.h"
#pragma interrupt int_INTP0(channel=0)
volatile char f;
int flash_a, b;
extern int boot_a, boot_b;    /* Variables defined in the boot area */
extern void boot_func(void);  /* Function defined in the boot area */
int f1(int a)
{
      return (++a);
}
int f2(int b)
{
      return (--b);
}
void main(void)   /* Main function in the flash area */
{
      boot_a++;   /* Access to a variable in the boot area */
      boot_b++;   /* Access to a variable in the boot area */
      boot_func();/* Access to a function in the boot area */
}
void int_INTP0(void)
{
      f = 1;
}
```

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Dec.18.19	-	New release

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 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.
5. 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.
6. 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.
7. 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.
8. 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.
9. 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.
10. 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.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. 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.4.0-1 November 2017)

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/.