How to Divide Boot and Flash Areas

CC-RL C Compiler for RL78 Family

Microcomputer Tool Product Marketing Department,
Tool Business Division

Renesas System Design Co., Ltd.

Jun 19, 2015 Rev. 1.00

R20UT3475EJ0100

- This document describes the processing necessary to divide a program into boot and flash areas when using the CC-RL C compiler for the RL78 family.

- This document uses the following tools and versions for descriptions.
  - CC-RL C compiler for the RL78 family V.1.01.00
  - e^2 studio integrated development environment V.4.0.0.26
  - CS+ integrated development environment V.3.01.00

How to Divide Boot and Flash Areas

- Overview
- Common Processing for Boot and Flash Areas
- Boot Area
- Flash Area
- Debugging Tool
- Sample Programs

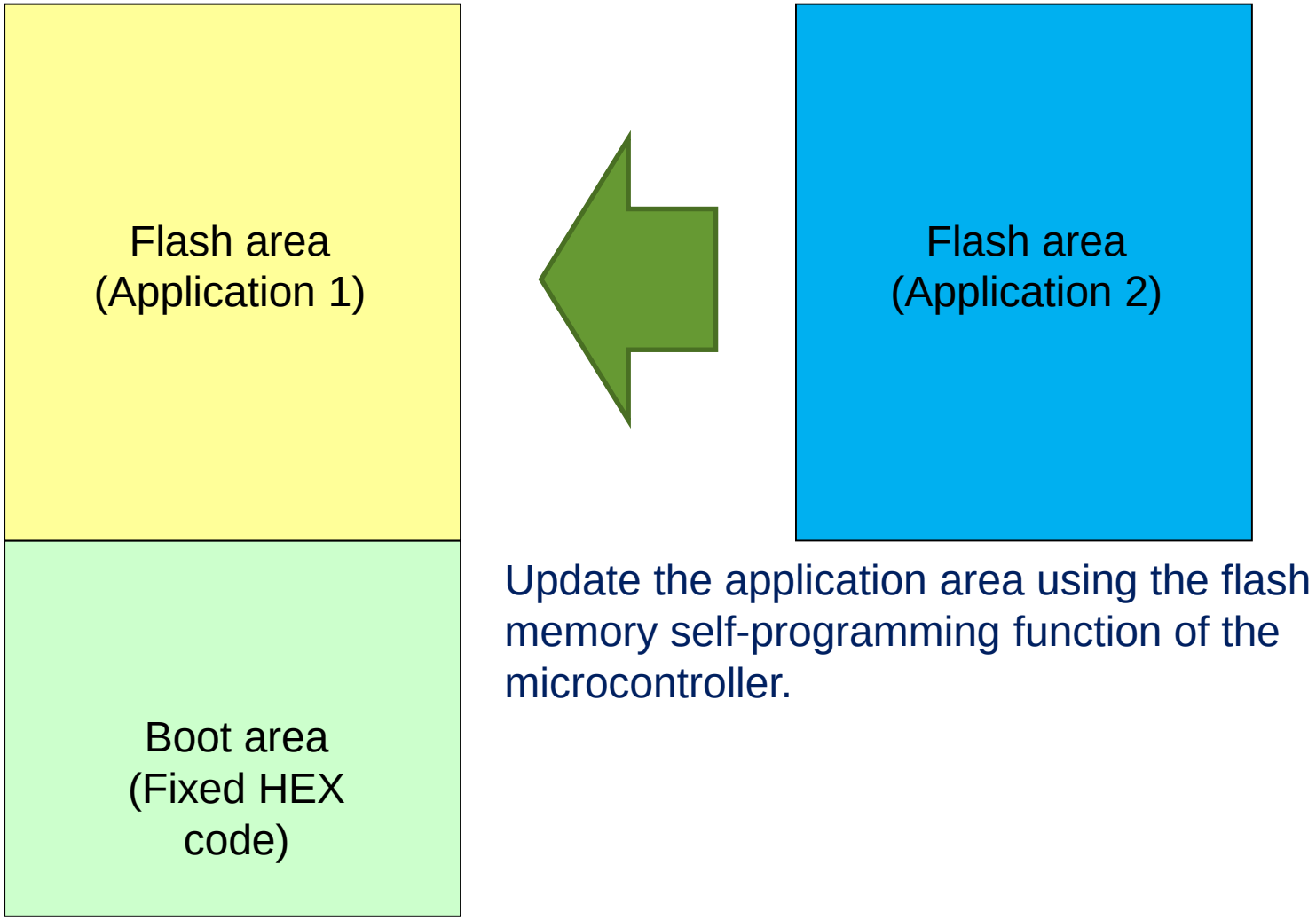
Overview

Overview

- Dividing Boot and Flash Areas
- Allocating Boot and Flash Areas
- Processing for Dividing Boot and Flash Areas
- Build Procedures for Boot and Flash Areas

Dividing Boot and Flash Areas (1/3)

- Divided areas on system



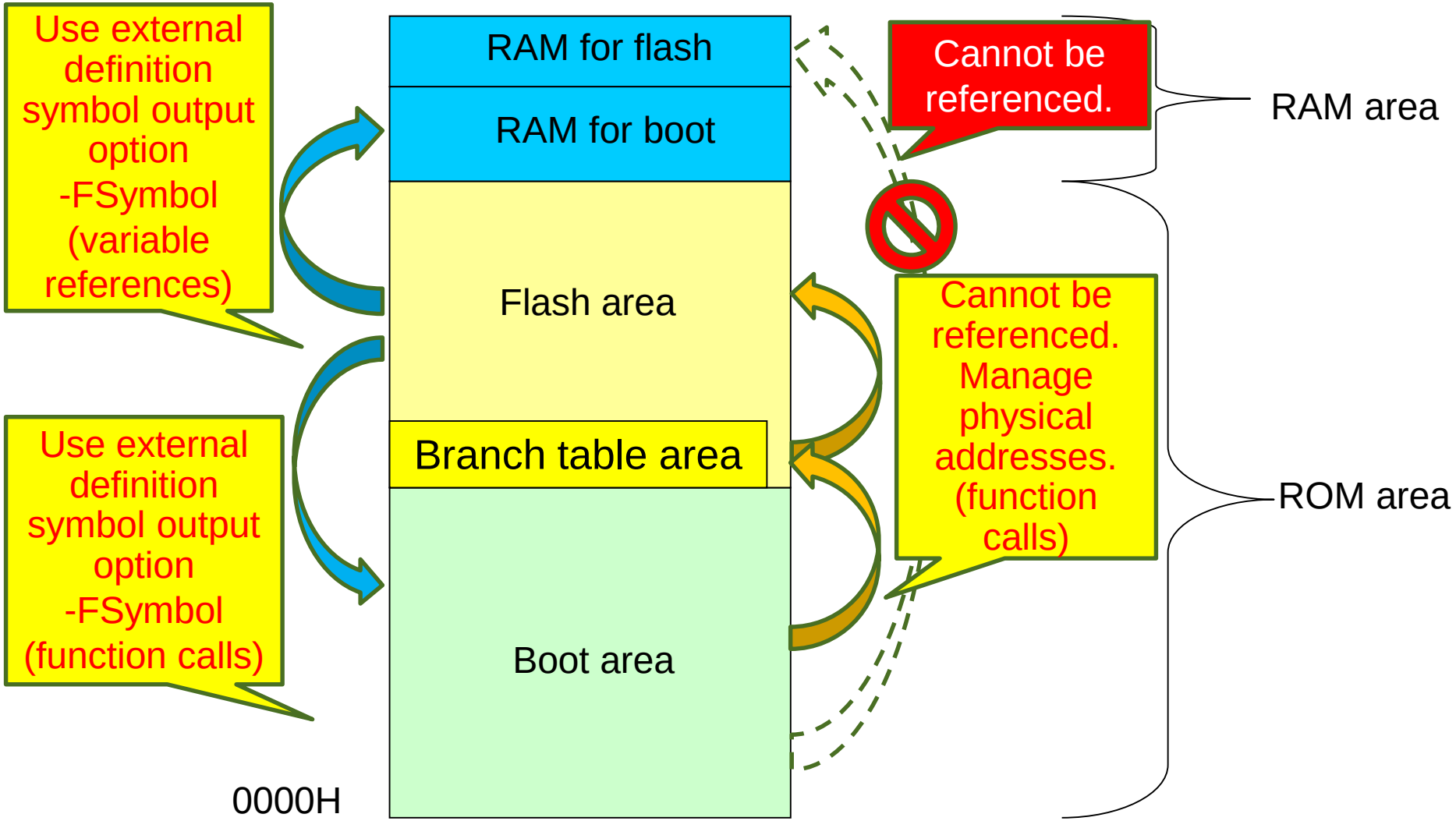
Update the application area using the flash memory self-programming function of the microcontroller.

Dividing Boot and Flash Areas (2/3)

- What are boot and flash areas?
  - Boot area: This area cannot be modified on the system.
  - Flash area: This area can be modified or replaced on the system.
- Purpose of dividing boot and flash areas
  - Only the program in the flash area can be modified without reconfiguring the program in the boot area.
- Requirements for boot and flash areas
  - The variables and functions in the boot area can be accessed from the flash area.
    - The external definition output option -FSymbol should be used in the boot area project.
    - The external definition symbol file should be specified as a target of build 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.

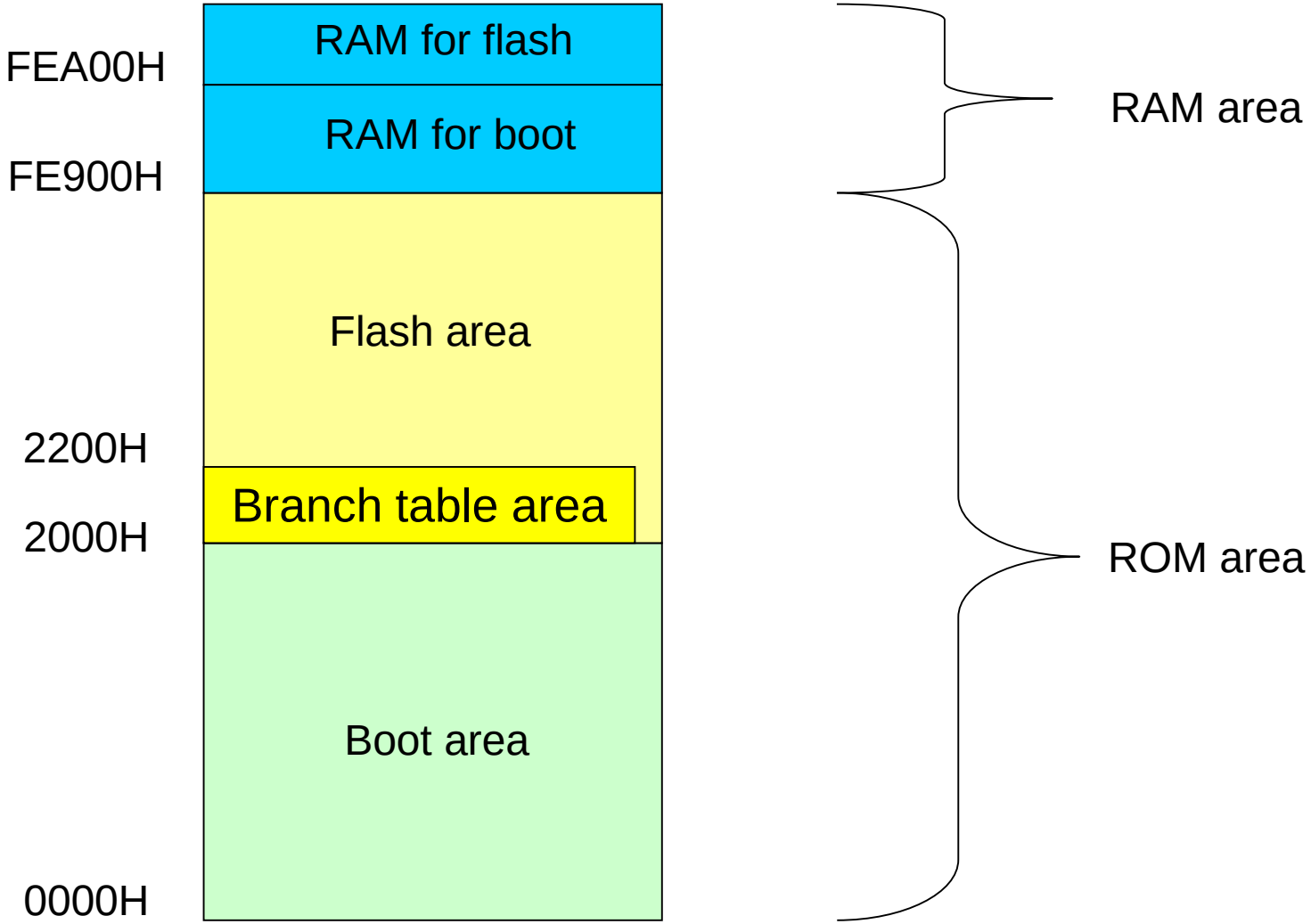
Dividing Boot and Flash Areas (3/3)

- References to variables and functions between boot and flash areas



Allocating Boot and Flash Areas

- Example: Allocate the boot and flash areas as follows.

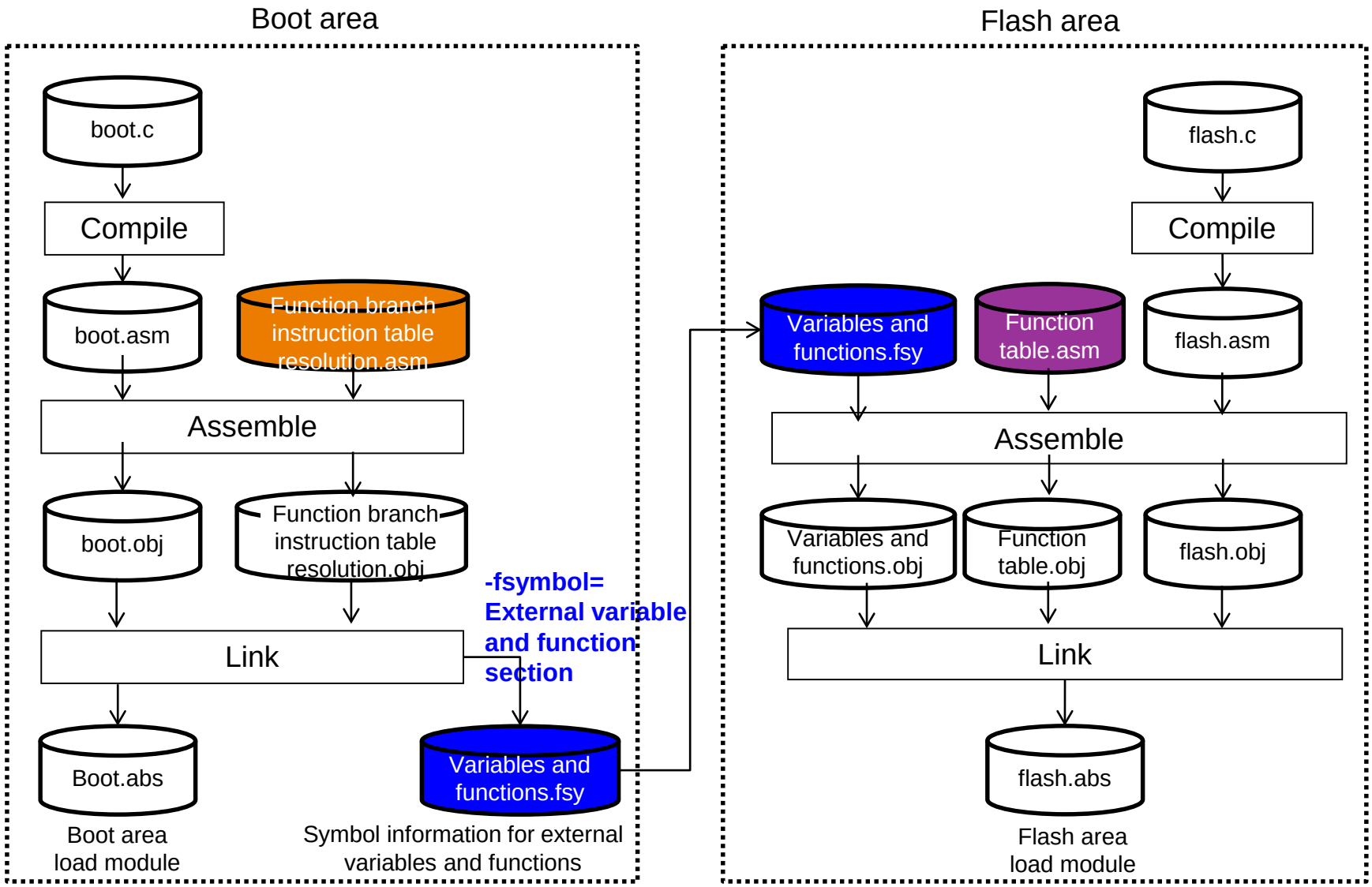


Processing for Dividing Boot and Flash Areas

- Creating the boot area project
  - Create boot area programs in the source file.
  - Specify necessary linker options.
  - Build the boot area project before the flash area project because the boot area project is necessary when building the flash area project.

- Creating the flash area project
  - Create flash area programs in the source file.
  - Specify necessary linker options.

Overview of Build Processing for Boot and Flash Areas



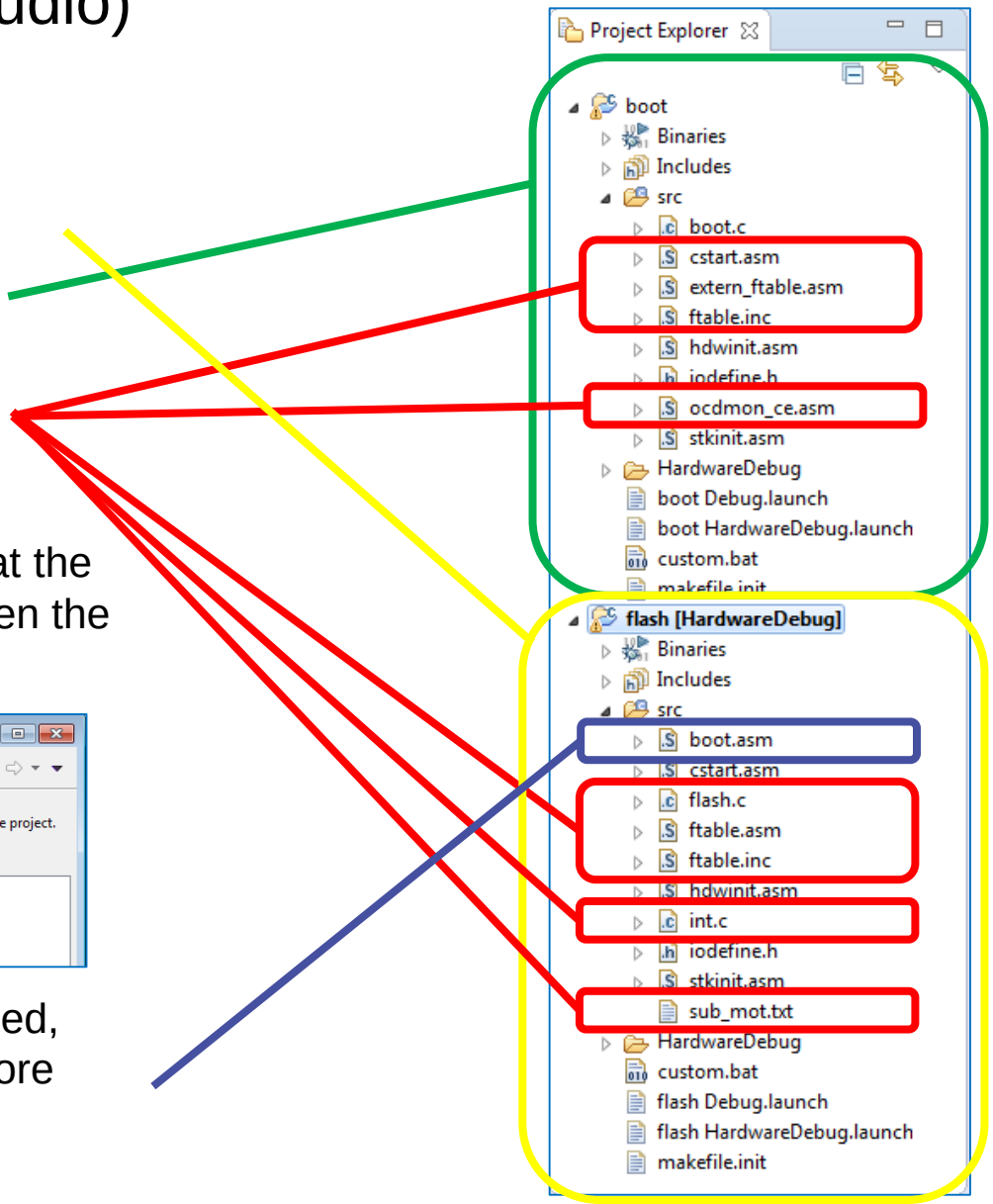
Common Processing for Boot and Flash Areas

Common Processing for Boot and Flash Areas

- Creating projects
  - e^2 studio
  - CS+
- Creating a common program for the boot and flash areas
  - Address definition file for the branch table (assembly language)
- Hex files for the boot and flash areas
- Initialization procedure

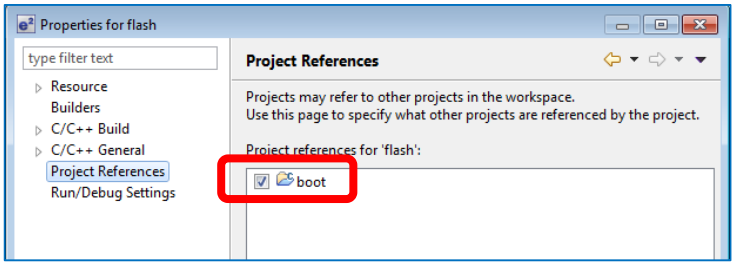
Creating Projects (e² studio)

- Create projects. *
  - Flash area project
  - Boot area project
- Add target files for build.



* Remarks

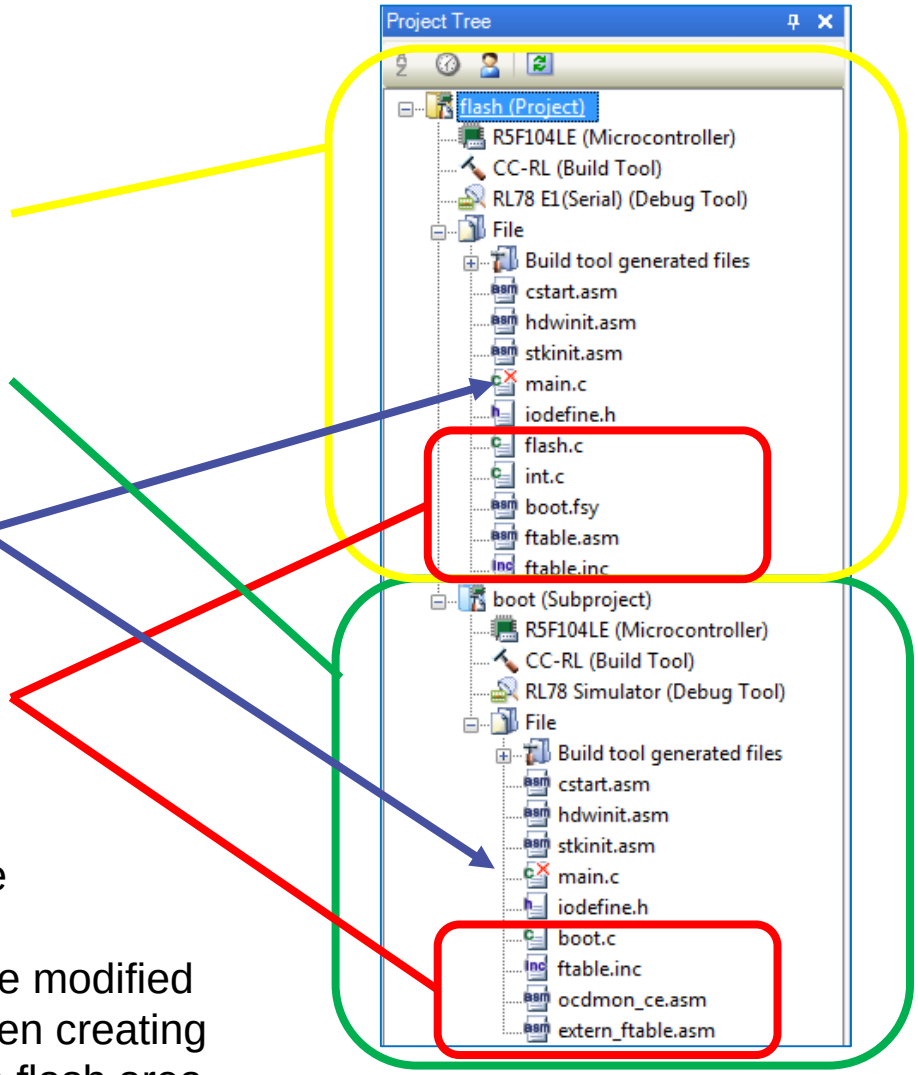
(1) Set up the flash area project so that the boot area project is referenced when the flash area project is built.



(2) As the *.fsy file cannot be assembled, change the extension to *.asm before registration.

Creating Projects (CS+)

- Create projects. *
  - Main project
    - – Flash area project
  - Sub-project
    - – Boot area project
- Exclude the automatically generated files from the target of build.
- Add target files for build.



* Remarks
(1) The build order in CS+ should be [Sub-project] → [Main project].
(2) The boot area program will not be modified once it is created. Therefore, when creating the second- or a later generation flash area project, the sub-project can be deleted.

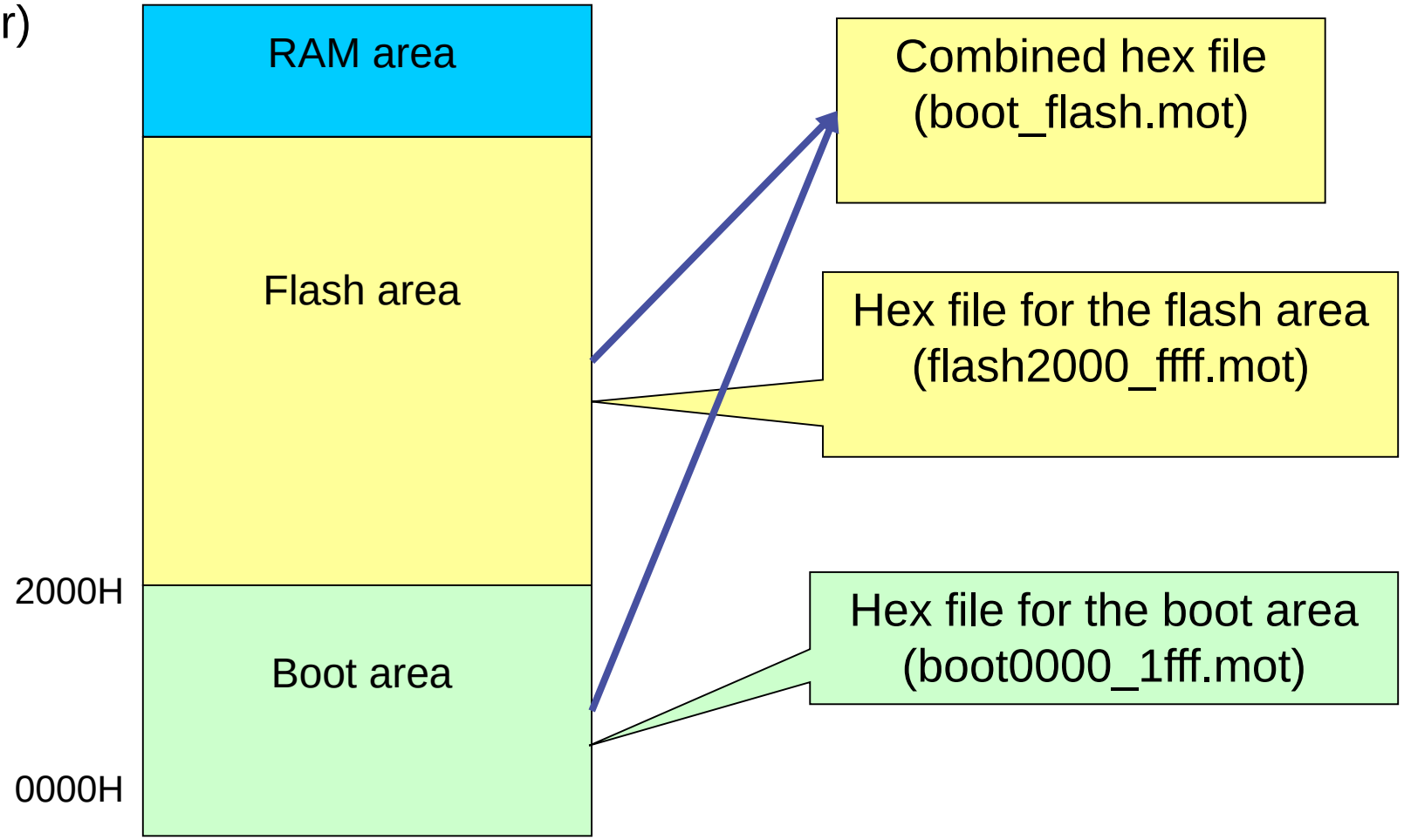
Creating a Common Program for Boot and Flash Areas

- Address definition file for the branch table (assembly language)
  - Include the file in the assembly source files for 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    0x2000
INTERRUPT_OFFSET        .EQU    0x100
```

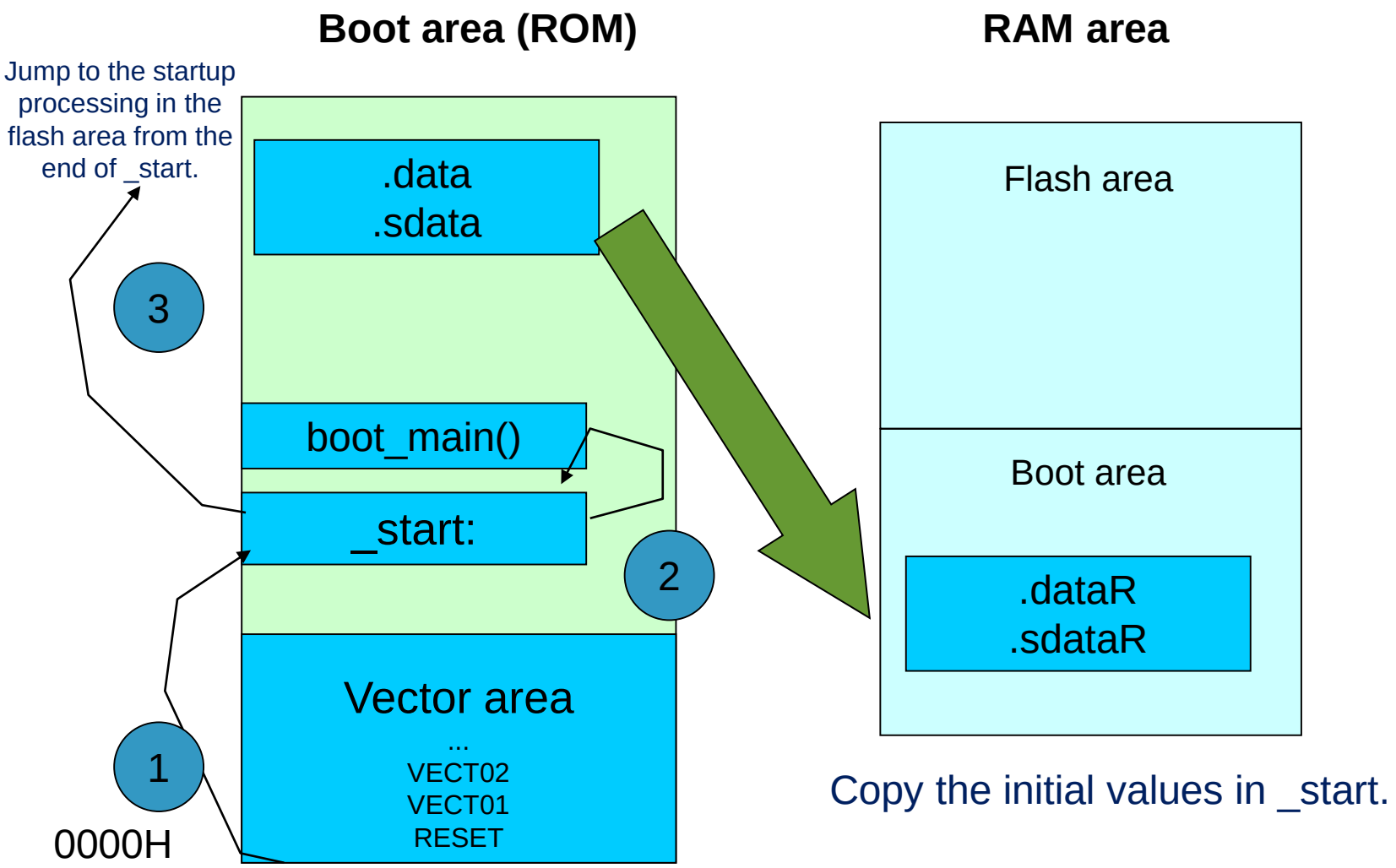
Hex Files for Boot and Flash Areas

- File names used in this document (output procedures are described later)

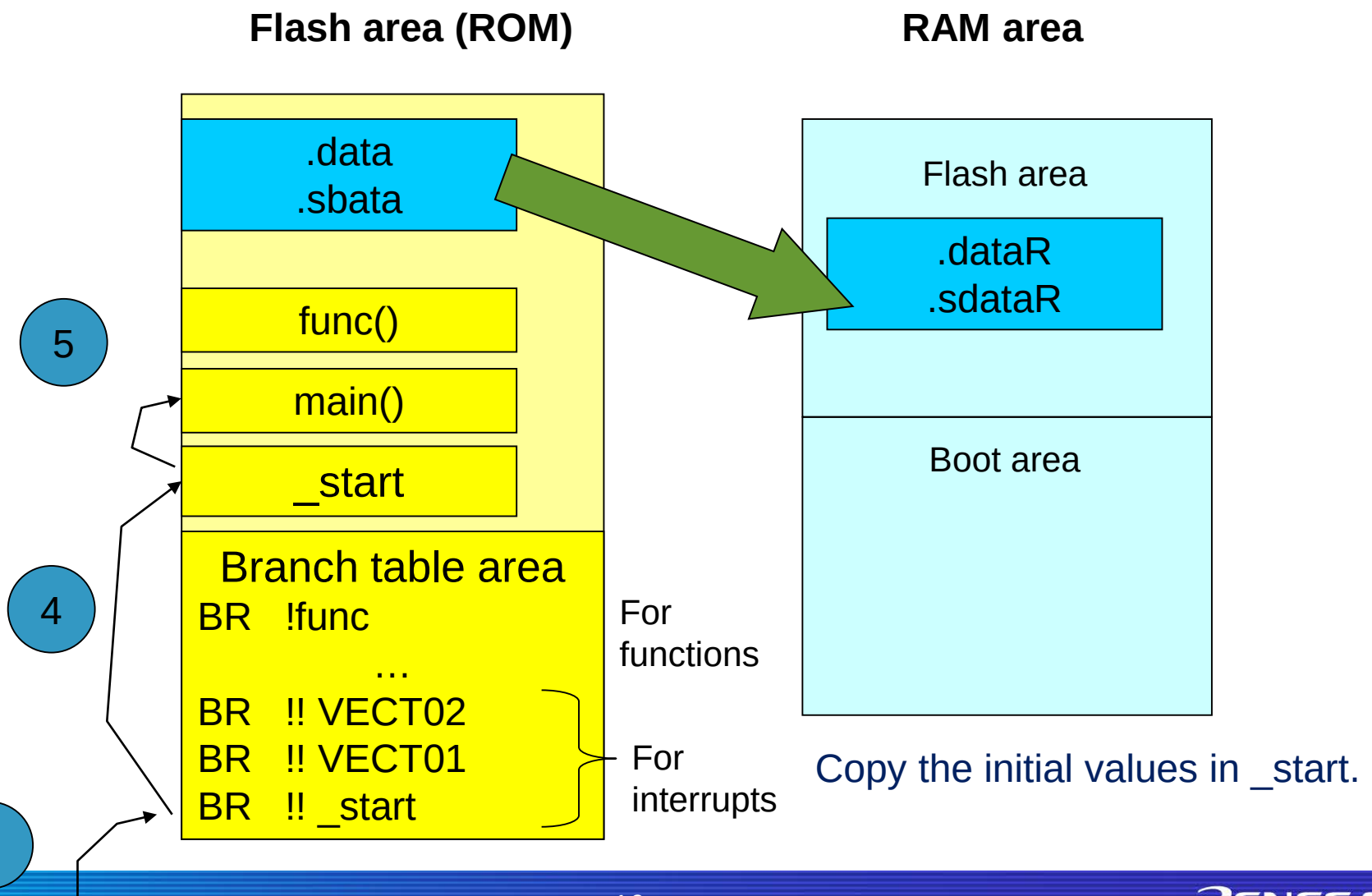


Note: A load module file (*.abs) is separately generated for each of the boot and flash areas.

Initialization Procedure (1/2)



Initialization Procedure (2/2)



Boot Area

Boot Area

- Creating boot area programs
  - Modifying the startup routine (cstart.asm)
  - Modifying hdwinit.asm and stkinit.asm
  - Creating a program for allocating the on-chip debug area
  - Creating a file for solving the function addresses in the branch table
- Specifying boot area options
  - Outputting a file for the external definition symbols
  - Specifying section allocation
  - Specifying a vector for branching to the interrupt function in the flash area
  - Making necessary settings for the on-chip debug function
  - Specifying hex file output only to the boot area addresses

Creating Boot Area Programs (1/9)

- Modifying the startup routine (cstart.asm) (1/6)
  - Add inclusion of the address definition file for the branch table.
  - Example:

```
$IFNDEF __RENESAS_VERSION__
__RENESAS_VERSION__   .EQU     0x01000000
$ENDIF

$INCLUDE "ftable.inc"
```

Creating Boot Area Programs (2/9)

- Modifying the startup routine (cstart.asm) (2/6)
  - Explicitly allocate the stack area.
    - Comment out the conditional assembly control instructions to make the definition of the .stack_bss section valid.
  - Example:

```
;$IF (__RENESAS_VERSION__ < 0x01010000)       ; for CC-RL V1.00
;-----------------------------------------------------------------
;         stack area
;-----------------------------------------------------------------
; !!! [CAUTION] !!!
; Set up stack size suitable for a project.
.SECTION .stack_bss, BSS
_stackend:
        .DS     0x200
_stacktop:
;$ENDIF
```

Creating Boot Area Programs (3/9)

- Modifying the startup routine (cstart.asm) (3/6)
  - Modify the section name.
    - Modify the section name to exclude it from the target of the external definition symbol output option -FSymbol.
  - Example:

```
;------------------------------------------------------------------------
;       startup
;------------------------------------------------------------------------
.SECTION .btext, TEXT
_start:
```

Creating Boot Area Programs (4/9)

- Modifying the startup routine (cstart.asm) (4/6)
  - Specify the stack pointer.
    - Comment out the conditional assembly control instructions to specify the explicitly allocated .stack_bss section as the stack pointer.
  - Example:

```
;$IF (__RENESAS_VERSION__ >= 0x01010000)
;         MOVW    SP,#LOWW(__STACK_ADDR_START)
;$ELSE   ; for CC-RL V1.00
          MOVW    SP,#LOWW(_stacktop)
;$ENDIF
```

Creating Boot Area Programs (5/9)

- Modifying the startup routine (cstart.asm) (5/6)
  - Modify the main function call to the call to the main function for the boot area, and add a branch instruction to the flash area startup routine.
  - Example:

```
;-----------------------------------------------
; call main function
;-----------------------------------------------
CALL    !!_boot_main                ; main();
BR      !!FLASH_TABLE
```

Creating Boot Area Programs (6/9)

- Modifying the startup routine (cstart.asm) (6/6)
  - Comment out the definition of the .const section when no mirror source area is included in the boot area.
  - Example:

```
;------------------------------------------------------------------------------
;         section
;------------------------------------------------------------------------------
$IF (__RENESAS_VERSION__ >= 0x01010000)
.SECTION .RLIB, TEXTF
.L_section_RLIB:
.SECTION .SLIB, TEXTF
.L_section_SLIB:
$ENDIF
.SECTION .textf, TEXTF
.L_section_textf:
;.SECTION .const, CONST
;.L_section_const:
```

Creating Boot Area Programs (7/9)

- Modifying hdwinit.asm and stkinit.asm
  - Modify the section name.
    - Modify the section name to exclude it from the target of the external definition symbol output option -FSymbol.
  - Example:

```
.btextf    .CSEG    TEXTF
```

Creating Boot Area Programs (8/9)

- Creating a program for allocating the on-chip debug area
  - To use the on-chip debug function, the following memory areas should be emptied (filled with 0xff).
    - Addresses 0x0002 to 0x0003
      - Specify 0xffff with the linker option -VECTN.
    - Addresses 0x00ce to 0x00d7
      - Make definitions in the assembly source (see the following program).
    - Last 512 bytes of ROM
      - Allocate this area through the flash area project.
  - Example: ocdrom_ce.asm

```
MON_CE.CSEG AT 0x00ce
        .DB8 0xffffffffffffffff
        .DB2 0xffff
```

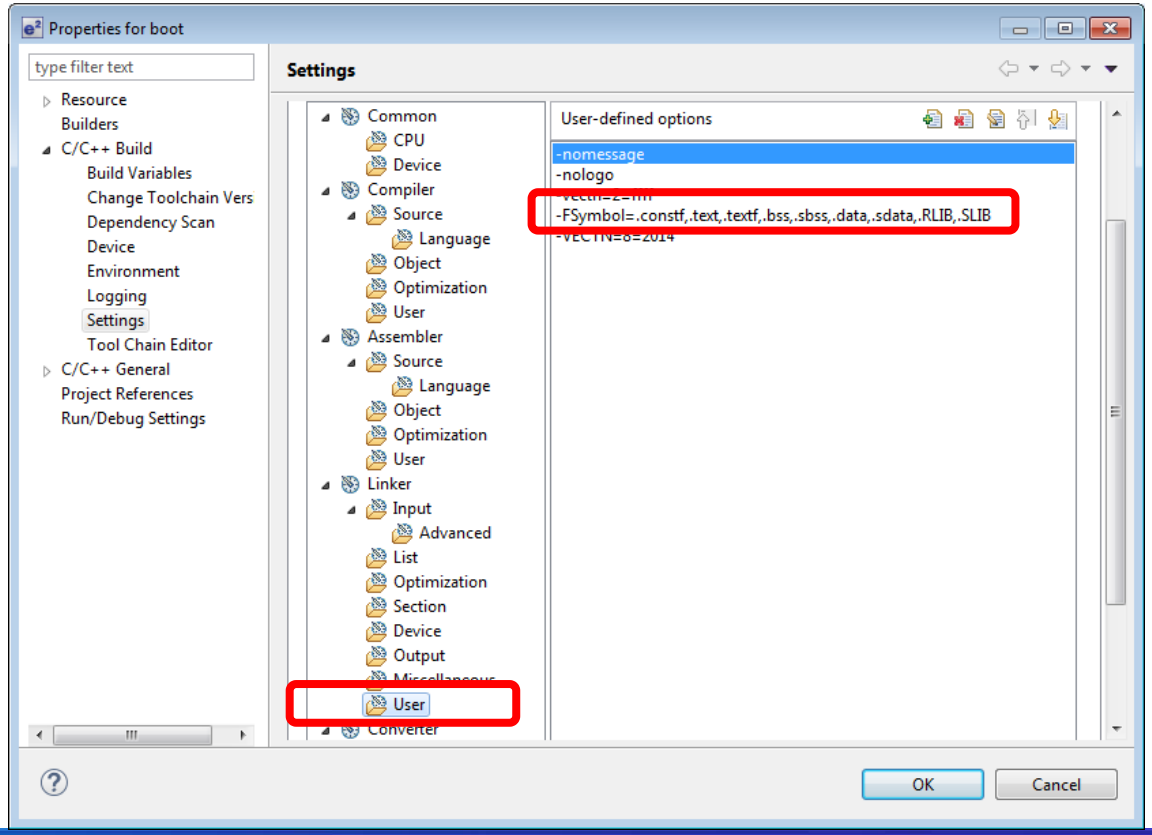
Creating Boot Area Programs (9/9)

- Creating a file for solving the function addresses in the branch table (assembler)
  - Define symbols for solving the addresses for the branch table to be used to call functions in the flash area from the C source.
  - Register this file in the project.
  - Example: extern_ftable.asm

```
$INCLUDE "ftable.inc"
        .public  _f1
_f1     .equ     (FLASH_TABLE + INTERRUPT_OFFSET + (0 * 4))
        .public  _f2
_f2     .equ     (FLASH_TABLE + INTERRUPT_OFFSET + (1 * 4))
```

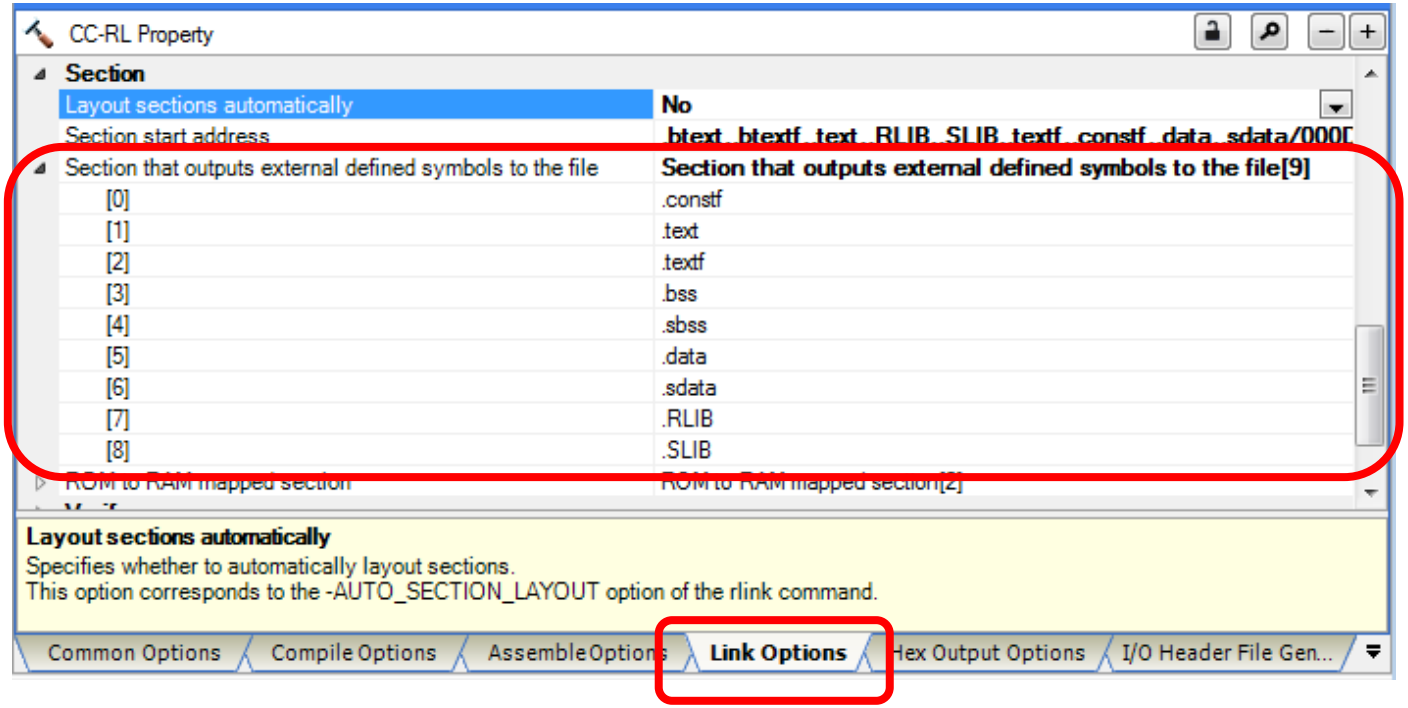
Specifying Boot Area Options (1/9)

- Outputting the external definition symbols to a file so that the flash area project can access the variables and functions in the boot area.
  - Register all target sections with the -FSymbol option.
  - Example: e^2 studio



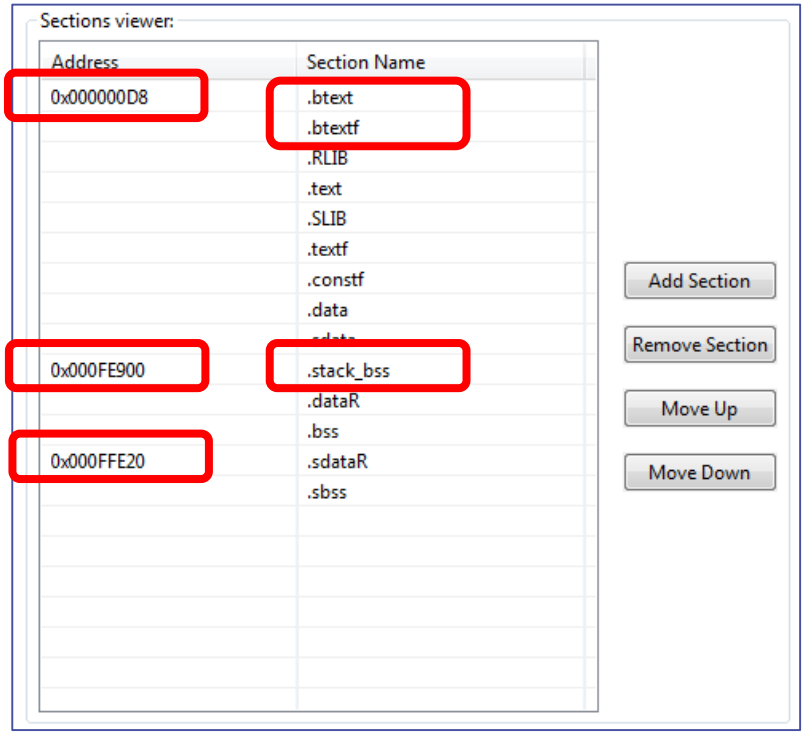
Specifying Boot Area Options (2/9)

- Example: CS+

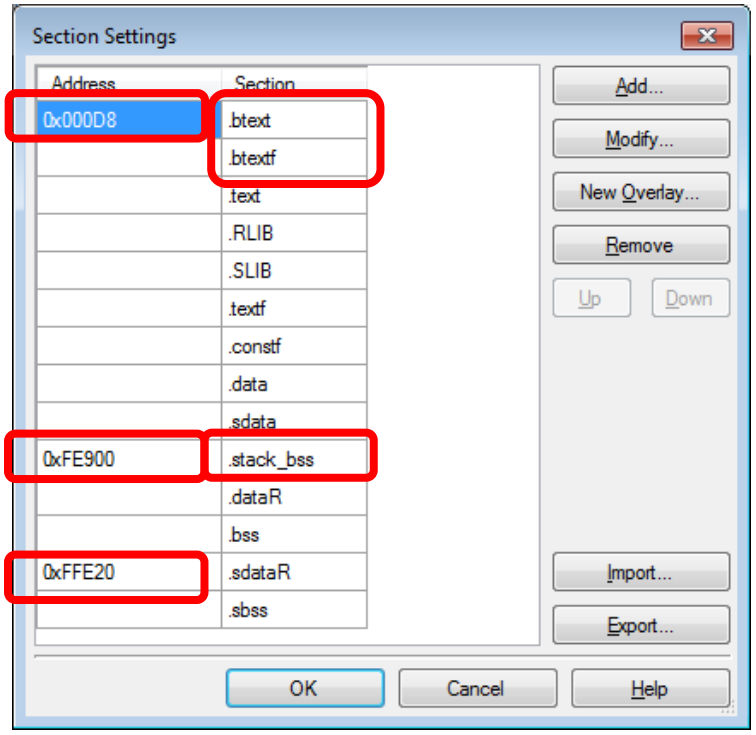


Specifying Boot Area Options (3/9)

- Specifying section allocation
  - Specify 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: e^2 studio

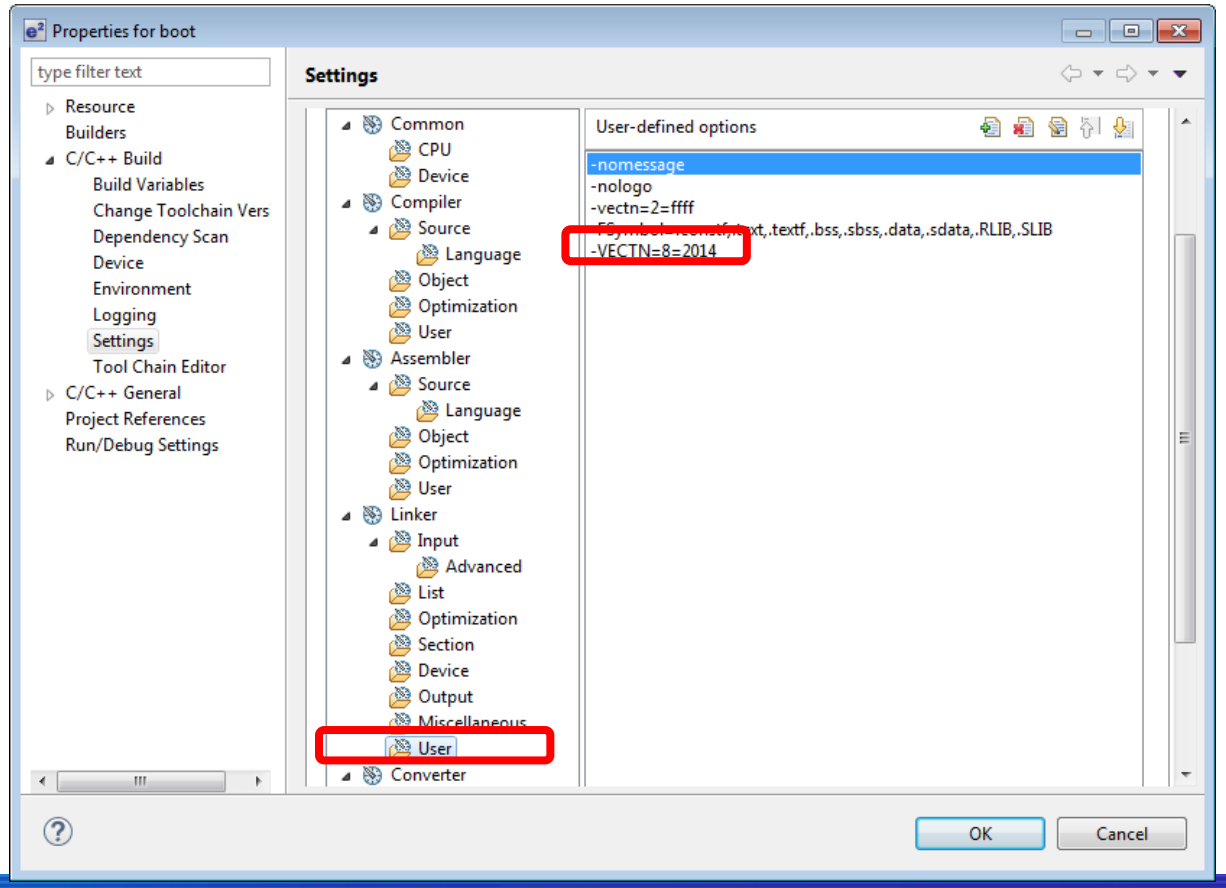


Example: CS+



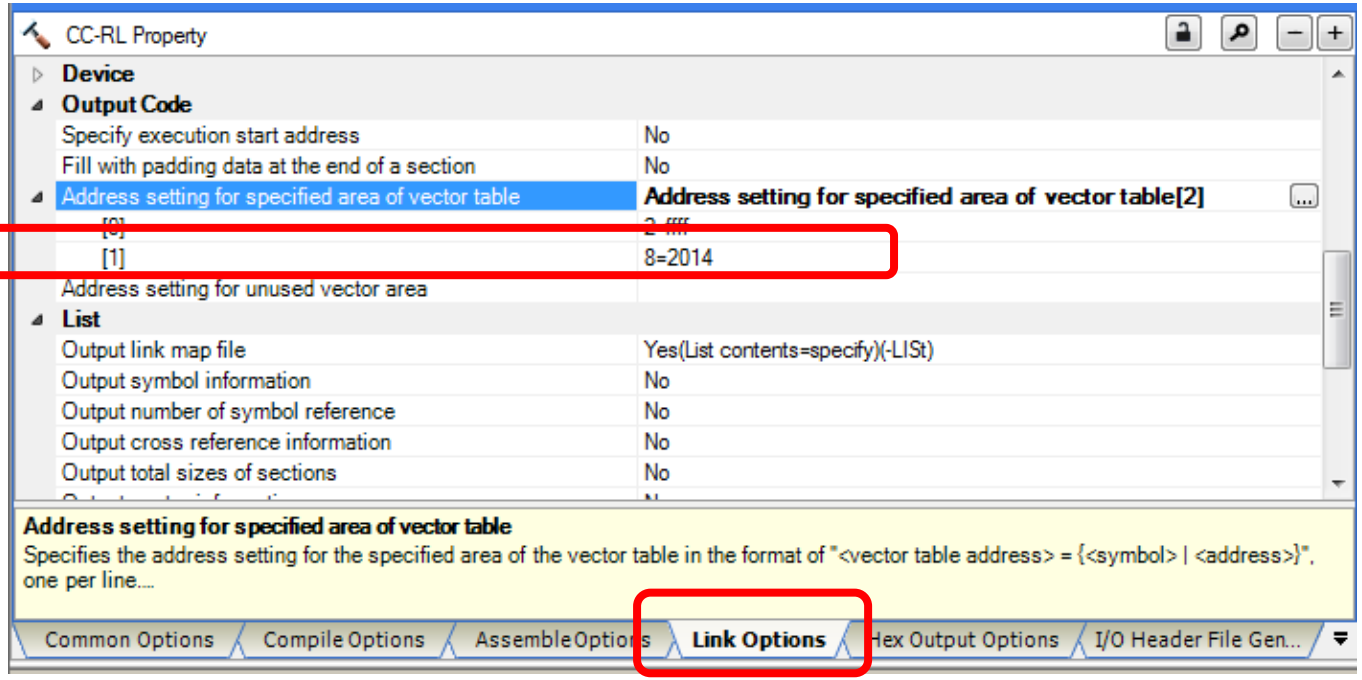
Specifying Boot Area Options (4/9)

- Specifying a vector for branching to the interrupt function in the flash area
  - Specify the address in the branch table with the linker option -VECTN.
  - Example: e^2 studio　To specify 0x2014 for address 8.



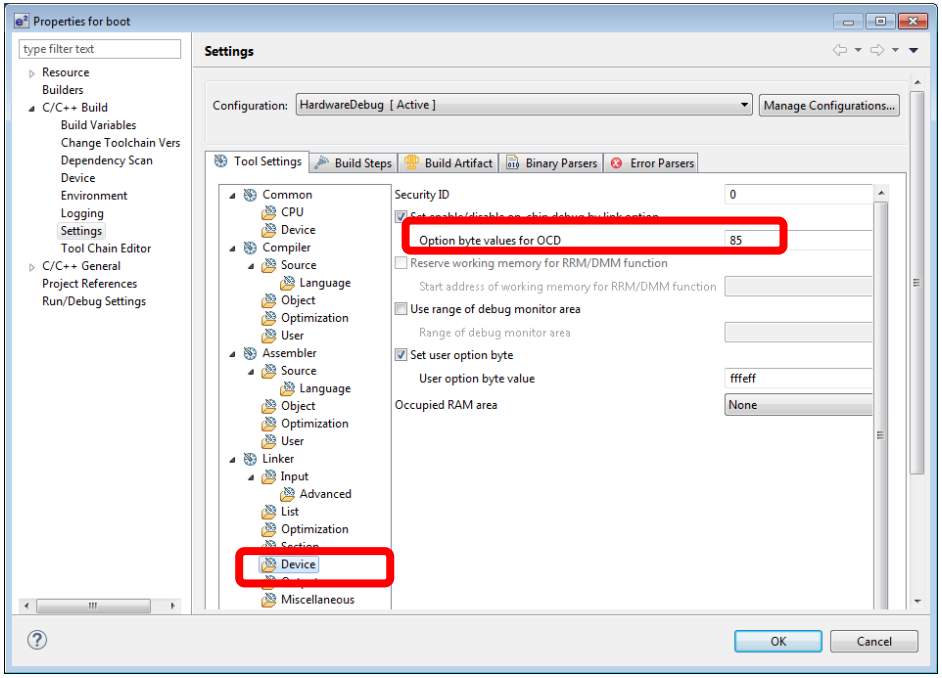
Specifying Boot Area Options (5/9)

- Example: CS+ To specify 0x2014 for address 8.



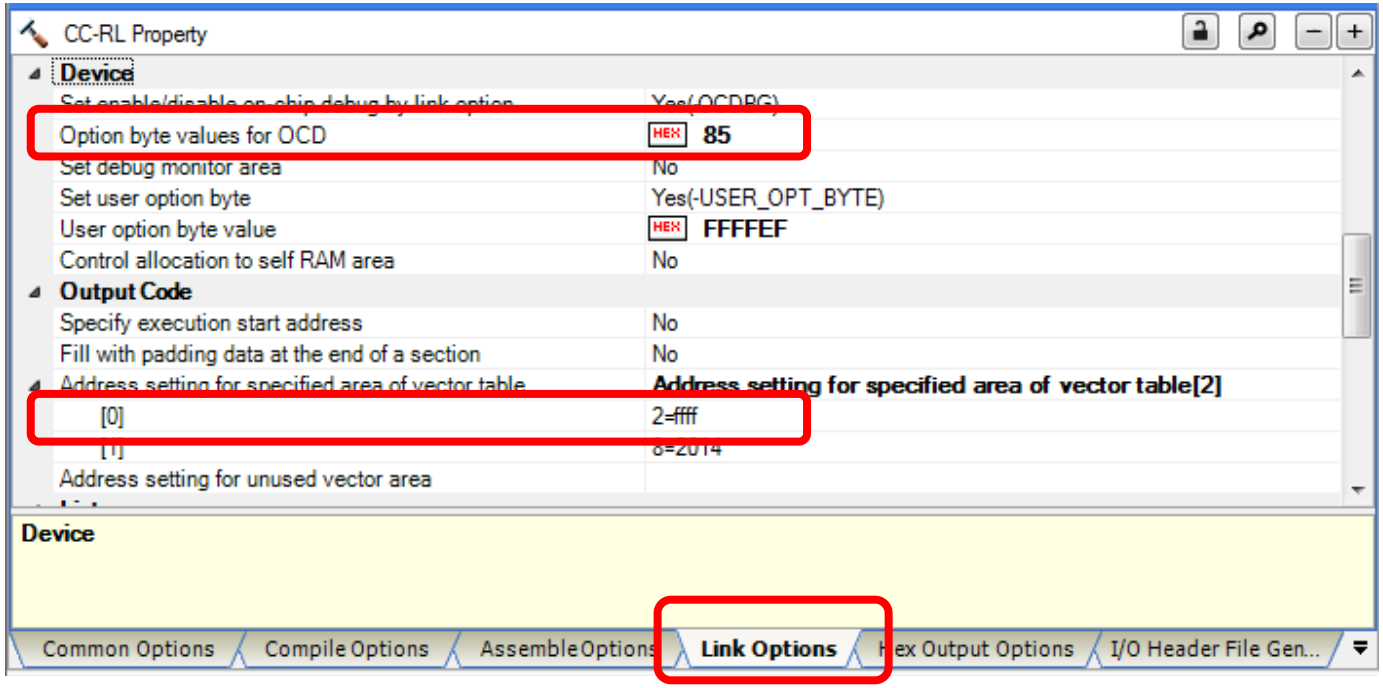
Specifying Boot Area Options (6/9)

- Making necessary settings for the on-chip debug function
  - Allocate the area of addresses 0x0002 and 0x0003 with the linker option -VECTN (in e^2 studio, this area is automatically allocated).
  - Set the linker option –OCDBG to be enabled and specify the value for the on-chip debug option byte.
  - Example: e^2 studio



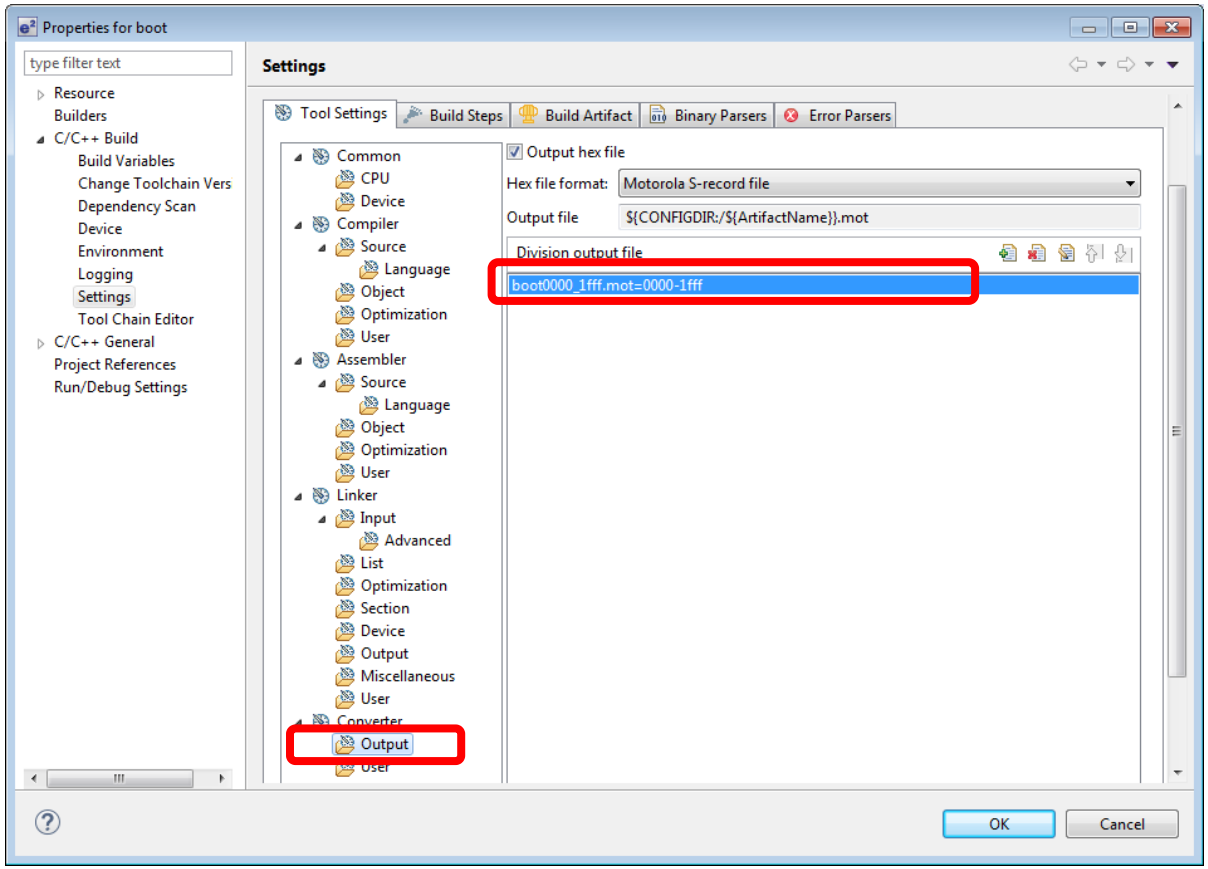
Specifying Boot Area Options (7/9)

- Example: CS+



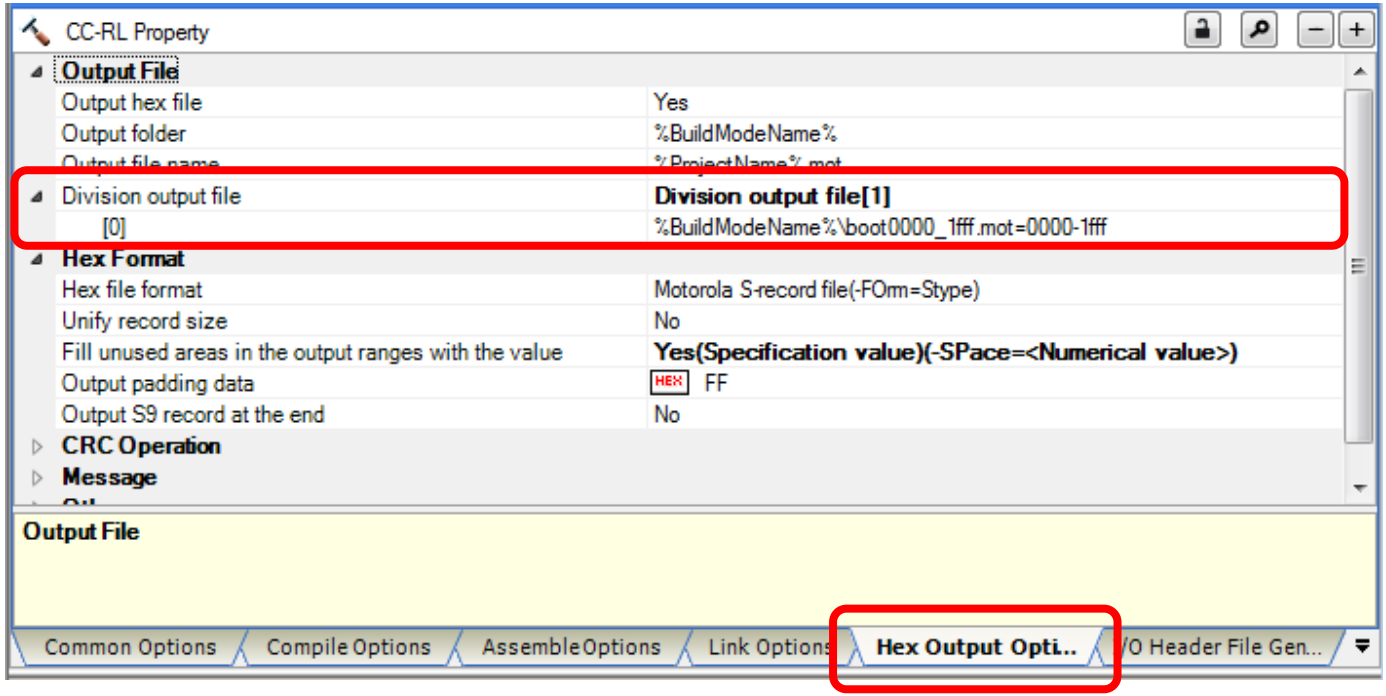
Specifying Boot Area Options (8/9)

- Specifying hex file output only to the boot area addresses
  - Specify the output file name and output addresses.
  - Example: e^2 studio



Specifying Boot Area Options (9/9)

- Example: CS+



Flash Area

Flash Area

- Creating flash area programs
  - Modifying the startup routine (cstart.asm)
  - Creating a branch table program
  - Defining an interrupt function
- Specifying flash area options
  - Registering the external definition symbol file to the project
  - Specifying section allocation
  - Specifying hex file output only to the flash area addresses
  - Combining the hex files for the boot and flash areas

Creating Flash Area Programs (1/3)

- Modifying the startup routine (cstart.asm)
  - Comment out the stack pointer settings.
    - The stack pointer specified in the boot area startup routine should be used; a stack pointer must not be specified again in the flash area.
  - Example:

```
        ;-----------------------------------------------
        ; setting the stack pointer
        ;-----------------------------------------------
;$IF (__RENESAS_VERSION__ >= 0x01010000)
;       MOVW    SP,#LOWW(__STACK_ADDR_START)
;$ELSE  ; for CC-RL V1.00
;       MOVW    SP,#LOWW(_stacktop)
;$ENDIF
```

Creating Flash Area Programs (2/3)

- Creating a branch table program
  - At the addresses called from the boot area, write instructions for branching to the function addresses in the flash area.
  - Example: ftable.asm

```
$INCLUDE "ftable.inc"
          .EXTERN  _start
          .EXTERN  _f1
          .EXTERN  _f2
.jtext    .CSEG AT FLASH_TABLE
          br       !!_start    ; RESET
          .DB4     0xffffffff  ;
          .DB4     0xffffffff  ; INTWDTI
          .DB4     0xffffffff  ; INTLVI
          .DB4     0xffffffff  ; INTP0
          br       !!_int_INTP0        ; INTP1
          .DB4     0xffffffff  ; INTP2
                   ~ Omitted ~
.jtext2   .CSEG AT FLASH_TABLE+INTERRUPT_OFFSET
          br       !!_f1
          br       !!_f2
```

For interrupts

For functions

Creating Flash Area Programs (3/3)

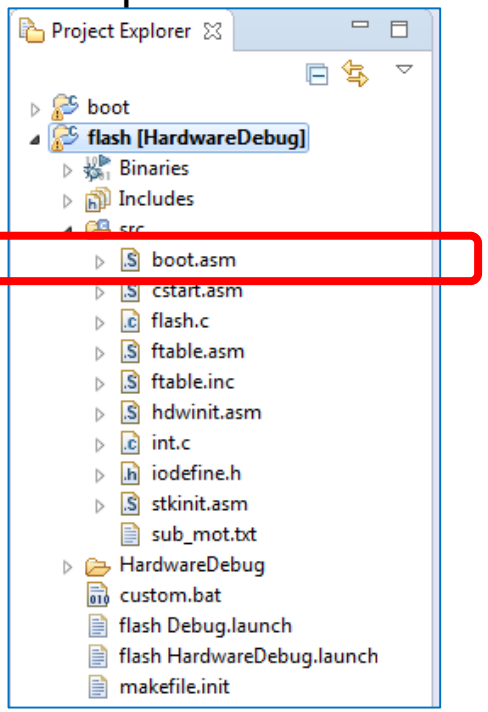
- Defining an interrupt function
  - The interrupt vector should be defined in the boot area project.
  - Do not specify the vector address (vect) with the #pragma interrupt directive in the flash area.
  - Example:

```
#include "iodefine.h"
#pragma interrupt int_INTP0
volatile char f;
void int_INTP0(void)
{
        f = 1;
}
```

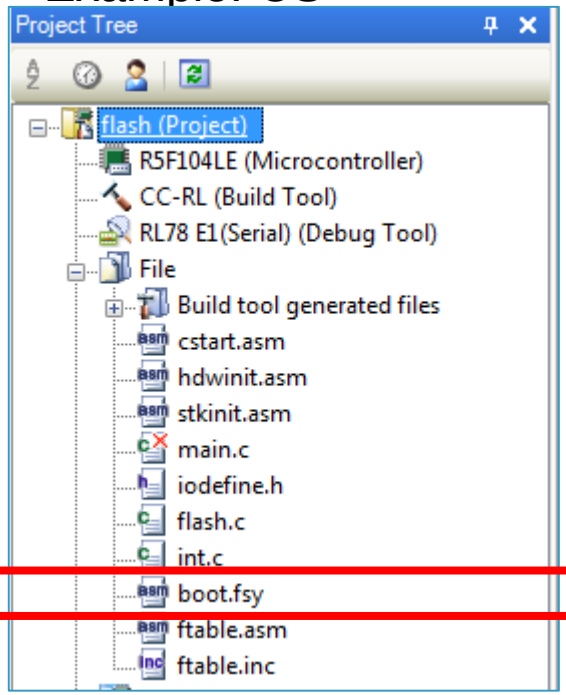
Specifying Flash Area Options (1/7)

- Registering the external definition symbol file to the project
  - Register the external definition symbol file created in the boot area to the project so that the variables and functions in the boot area can be accessed.
  -

Example: e² studio



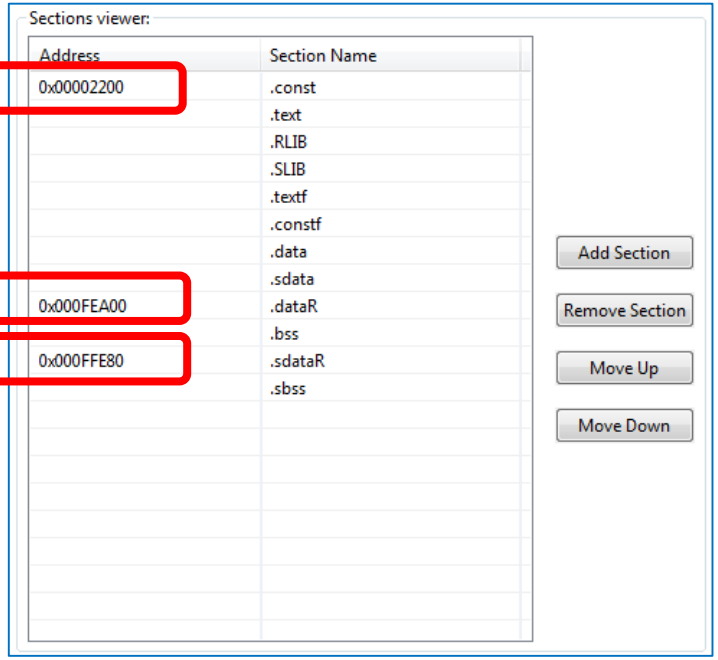
Example: CS+



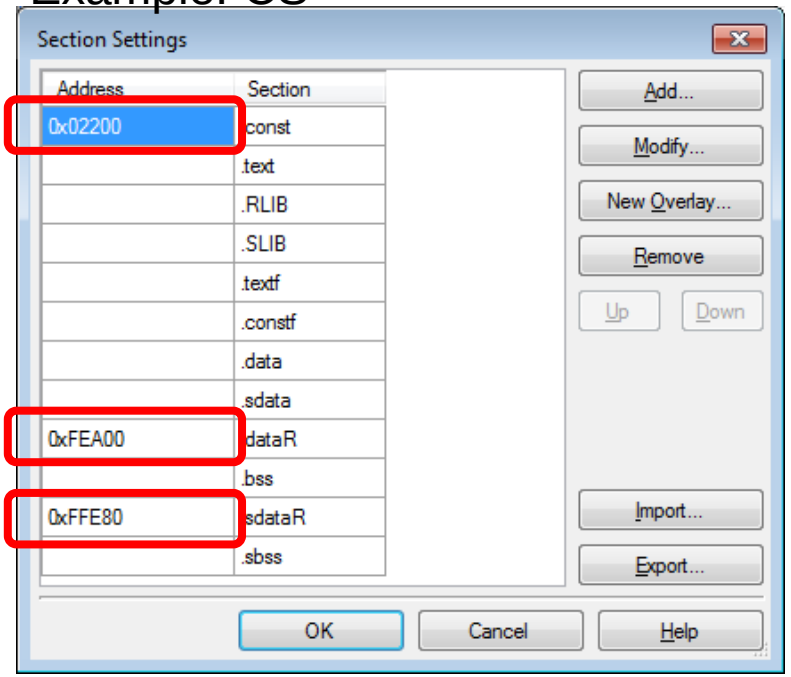
*Remarks: e² studio
As the *.fsy file cannot be assembled, change the extension to *.asm before registration

Specifying Flash Area Options (2/7)

- Specifying section allocation
  - Specify 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.
  - Example: e^2 studio

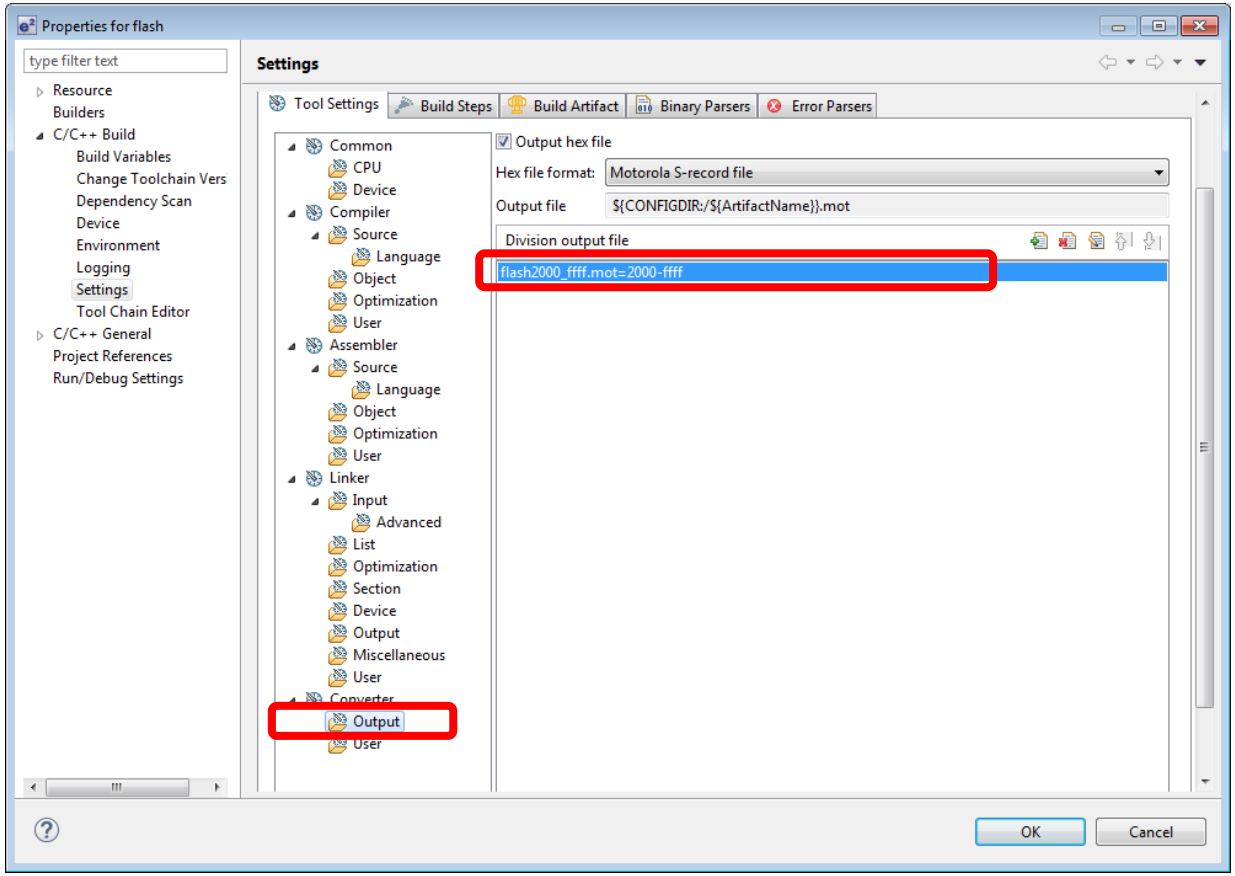


Example: CS+



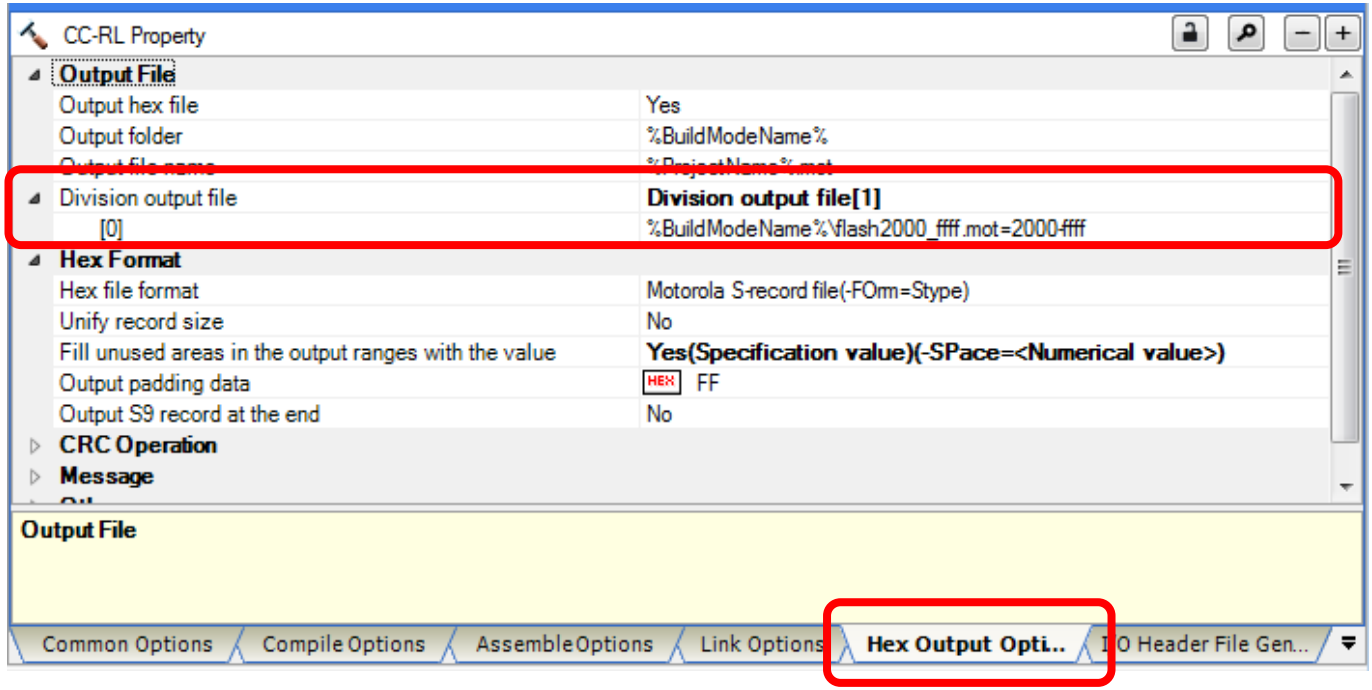
Specifying Flash Area Options (3/7)

- Specifying hex file output only to the flash area addresses
  - Specify the output file name and output addresses.
  - Example: e^2 studio



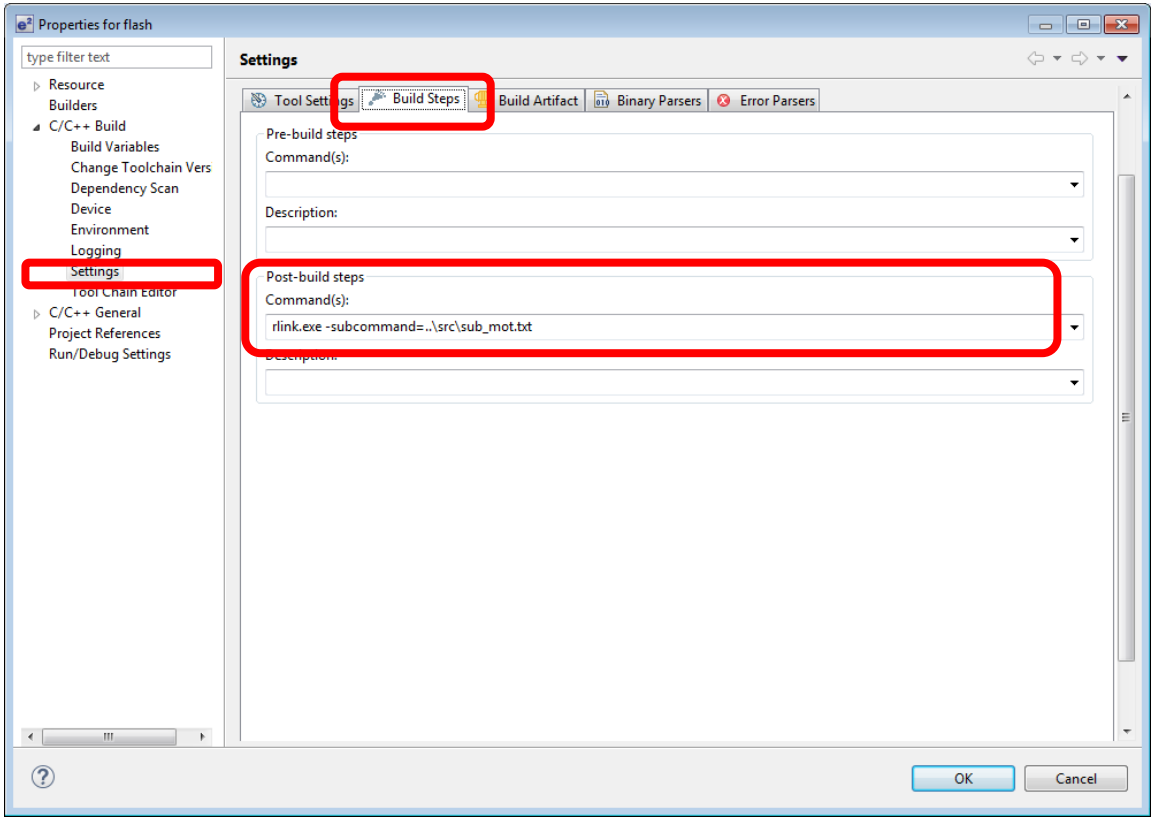
Specifying Flash Area Options (4/7)

- Example: CS+



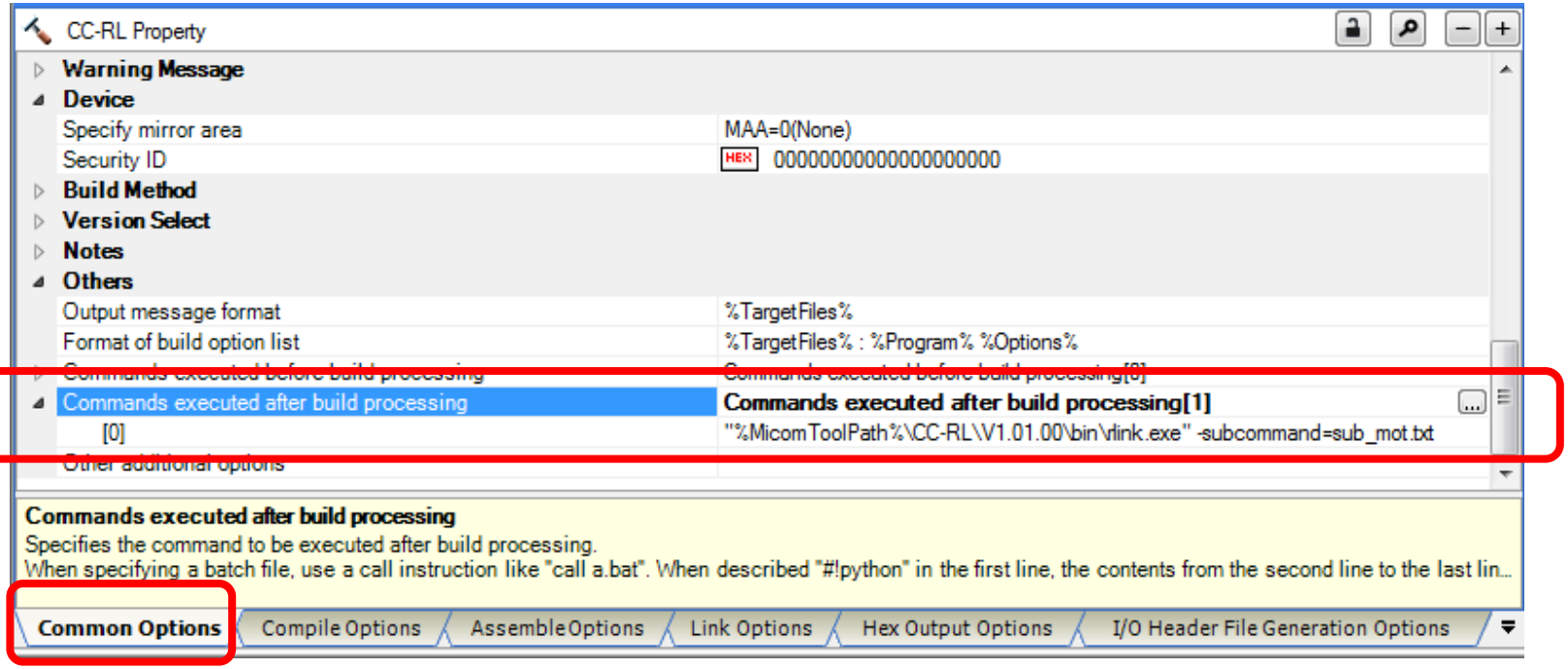
Specifying Flash Area Options (5/7)

- 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: e^2 studio



Specifying Flash Area Options (6/7)

- Example: CS+



Specifying Flash Area Options (7/7)

- Combining the hex files for the boot and flash areas
  - Specify the input hex files, their format, and output file name in the subcommand file to be input to the linker.
  - Example: sub_mot.txt (e^2 studio)

```
-input=..¥..¥boot¥HardwareDebug¥boot0000_1fff.mot
-input=flash2000_ffff.mot
-form=stype
-output=boot_flash.mot
```

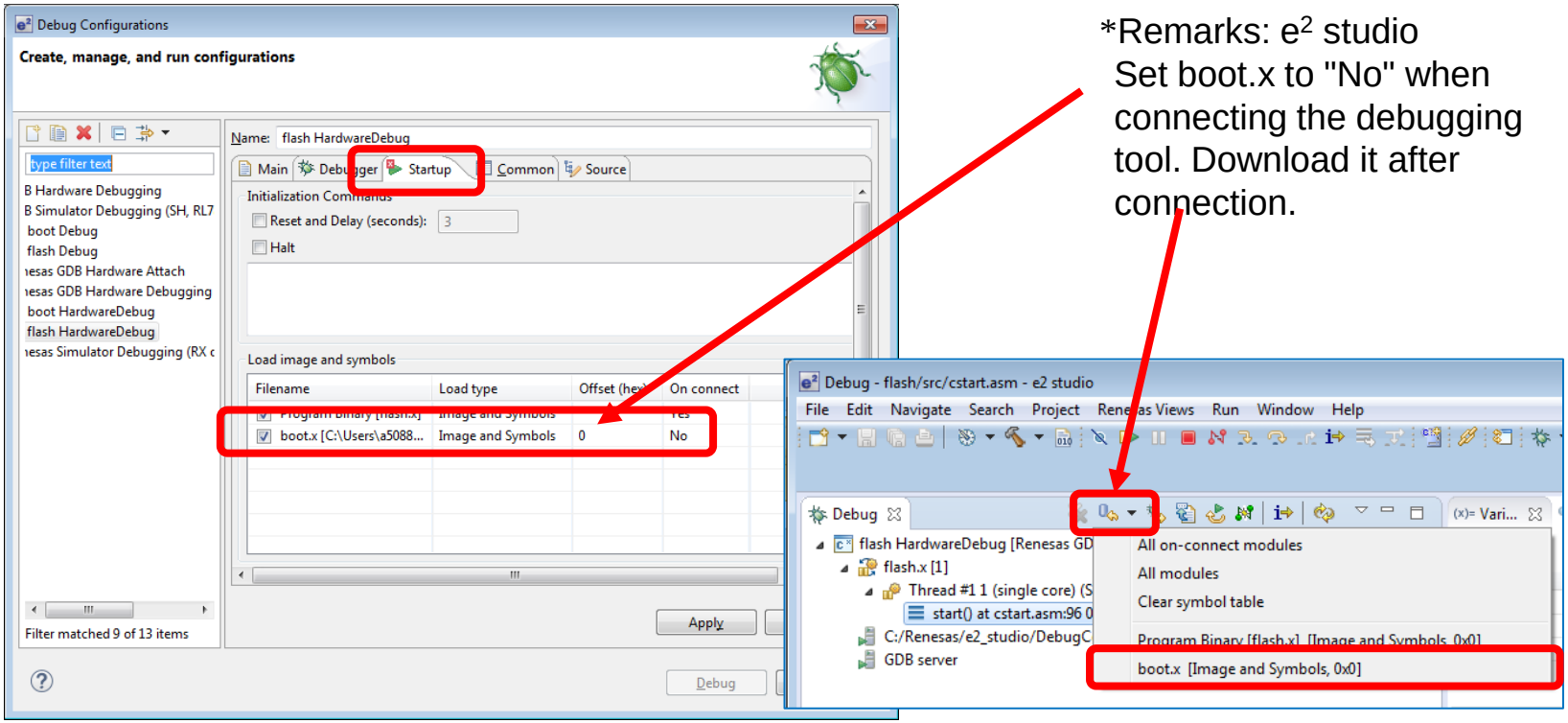
  - Example: sub_mot.txt (CS+)

```
-input=..¥boot¥DefaultBuild¥boot0000_1fff.mot
-input=.¥DefaultBuild¥flash2000_ffff.mot
-form=stype
-output=.¥DefaultBuild¥boot_flash.mot
```

Debugging Tool

Downloading to Debugging Tool (1/2)

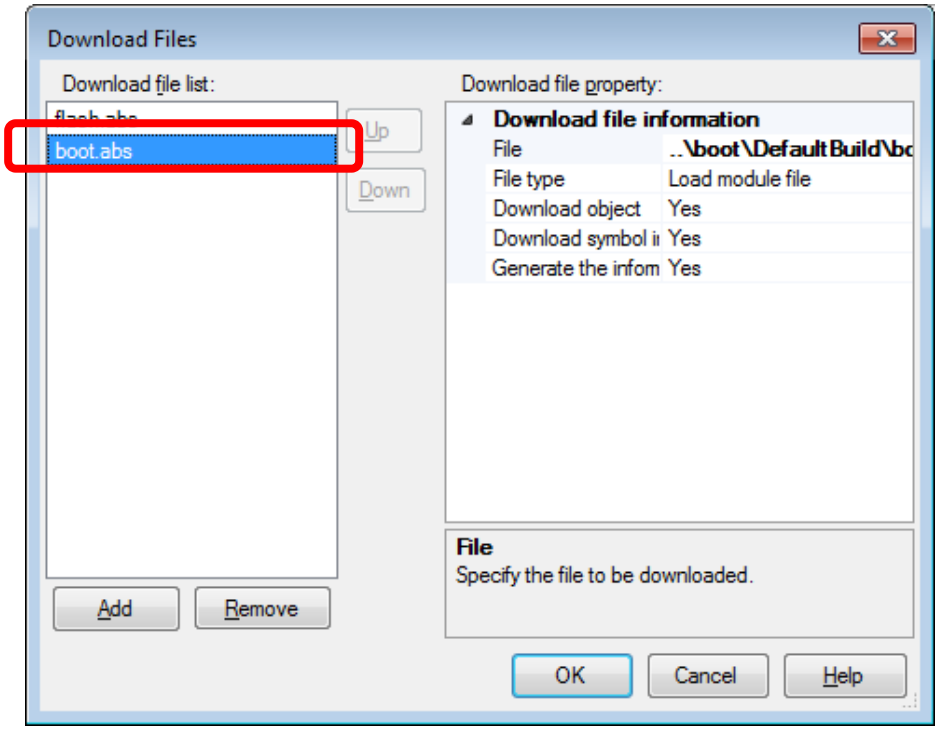
- Two load module files (*.abs) are generated; one for each of the boot and flash areas. Download both load module files.
  - Example: To add the load module file for the boot area to the flash area project (e² studio)



*Remarks: e² studio
Set boot.x to "No" when connecting the debugging tool. Download it after connection.

Downloading to Debugging Tool (2/2)

- Example: To add the load module file for the boot area to the flash area project (CS+)



Sample Programs

Sample Programs

- The following pages show examples of boot and flash area programs that use the programs created through the procedures described before.

Sample Program for Boot Area

```
#include “iodefine.h“              /* SFR definition file */
#pragma interrupt int_INTP1 (vect=INTP1)   /* 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_INTP1 (void)               /* Interrupt processing in the boot area */
{
  boot_a = 1;
}
```

Sample Program for Flash Area (1/2)

```
#include "iodefine.h"              /* SFR definition file */
int flash_a, b;
extern int boot_a, boot_b;         /* Function 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 */
}
```

Sample Program for Flash Area (2/2)

```
#include "iodefine.h"              /* SFR definition file */
#pragma interrupt int_INTP0  /* Interrupt definition in the flash area */
volatile char f;
void int_INTP0(void)               /* Interrupt processing in the flash area */
{
        f = 1;
}
```

Renesas System Design Co., Ltd.