RL78 Development Environment Migration Guide

Migration between RL78 family (IDE ed.) (CA78K0R to CC-RL)

December 28, 2016
R20UT3415EJ0102

Software Business Division
Renesas System Design Co., Ltd

Introduction

- This document describes how to manipulate projects in CS+ to migrate projects created for the CA78K0R C compiler for the RL78 family of MCUs to the CC-RL C compiler for the RL78 family of MCUs.

- This document describes the CS+ integrated development environment, the CA78K0R C compiler for the RL78 family of MCUs, and the CC-RL C compiler for the RL78 family of MCUs.
  The applicable versions are as follows.
  - CS+ V4.01.00
  - CA78K0R V1.20 and later
  - CC-RL V1.03.00

Agenda

Porting Projects to CS+ for CC-RL

Porting Projects to CS+ for CC-RL

CA78K0R projects that have been created using CS+ or CubeSuite+ can be ported to the CS+ environment for CC-RL in either of the following two ways.

Method 1 : Create a new project with CS+.
Create a new project in CS+ for RL78 and register existing source files that you have.

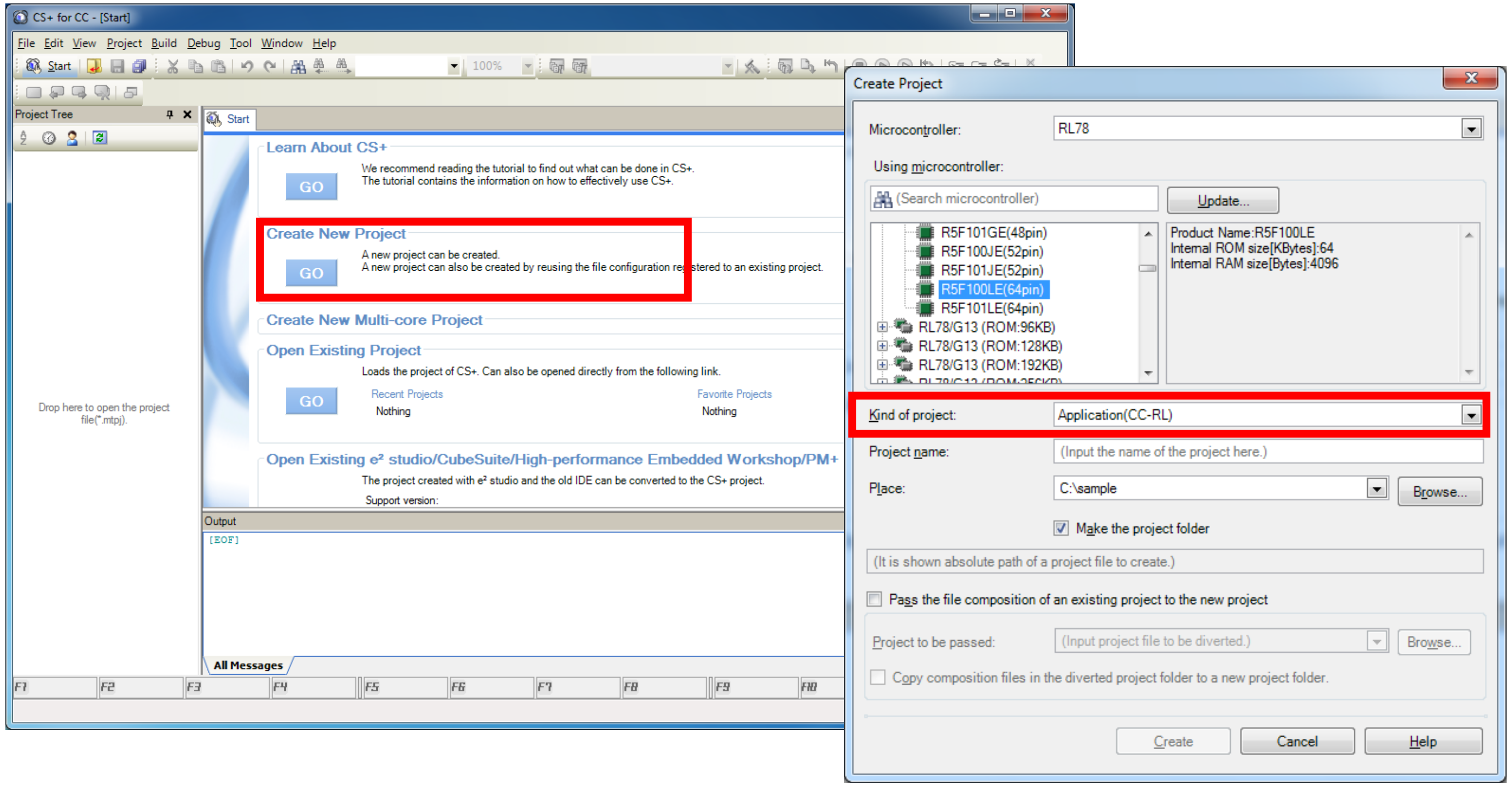
Method 2 : Utilize an existing project.
Utilize a CA78K0R CS+ or CubeSuite+ project to create a new project with CS+ for CC-RL.

Process	Method 1	Method 2
Source file registration	Manual	Automatic
Option setting	Manual	(Partially) automatic
Source file folder location	No care needs to be taken regarding the file registration location.	The folder structure should be the same as that of the existing project.(Note)
Conflict between source files and automatically generated files	Care should be taken regarding conflict at manual file registration.	Files should be modified after a project is created.

(Note): If you do not convert the source files, a build error may occur if the structures of folders differ and the paths to the folders are not specified.

Creating a New Project

After creating a new project, register and use the existing source files for CA78K0R.


CS+ for CC - [Start]
File Edit View Project Build Debug Tool Window Help
Start
Project Tree
Start
Drop here to open the project file(*.mtpj).
Output
All Messages
F1
F2
F3
F4
F5
F6
F7
F8
F9
F10

Learn About CS+

We recommend reading the tutorial to find out what can be done in CS+.
The tutorial contains the information on how to effectively use CS+.

GO

Create New Project

A new project can be created.
A new project can also be created by reusing the file configuration registered to an existing project.

GO

Create New Multi-core Project

Open Existing Project

Loads the project of CS+. Can also be opened directly from the following link.

GO

Recent Projects: Nothing
Favorite Projects: Nothing

Open Existing e² studio/CubeSuite/High-performance Embedded Workshop/PM+

The project created with e² studio and the old IDE can be converted to the CS+ project.

Support version:

```
[EOF]
```

Create Project

Microcontroller:	RL78
Using microcontroller:	(Search microcontroller) Update...
	R5F101GE(48pin)
	R5F100JE(52pin)
	R5F101JE(52pin)
	R5F100LE(64pin)
	R5F101LE(64pin)
	RL78/G13 (ROM:96KB)
	RL78/G13 (ROM:128KB)
	RL78/G13 (ROM:192KB)

Product Name:R5F100LE
Internal ROM size[KBytes]:64
Internal RAM size[Bytes]:4096

Kind of project:	Application(CC-RL)
Project name:	(Input the name of the project here.)
Place:	C:\sample

[✓] Make the project folder

(It is shown absolute path of a project file to create.)

[ ] Pass the file composition of an existing project to the new project

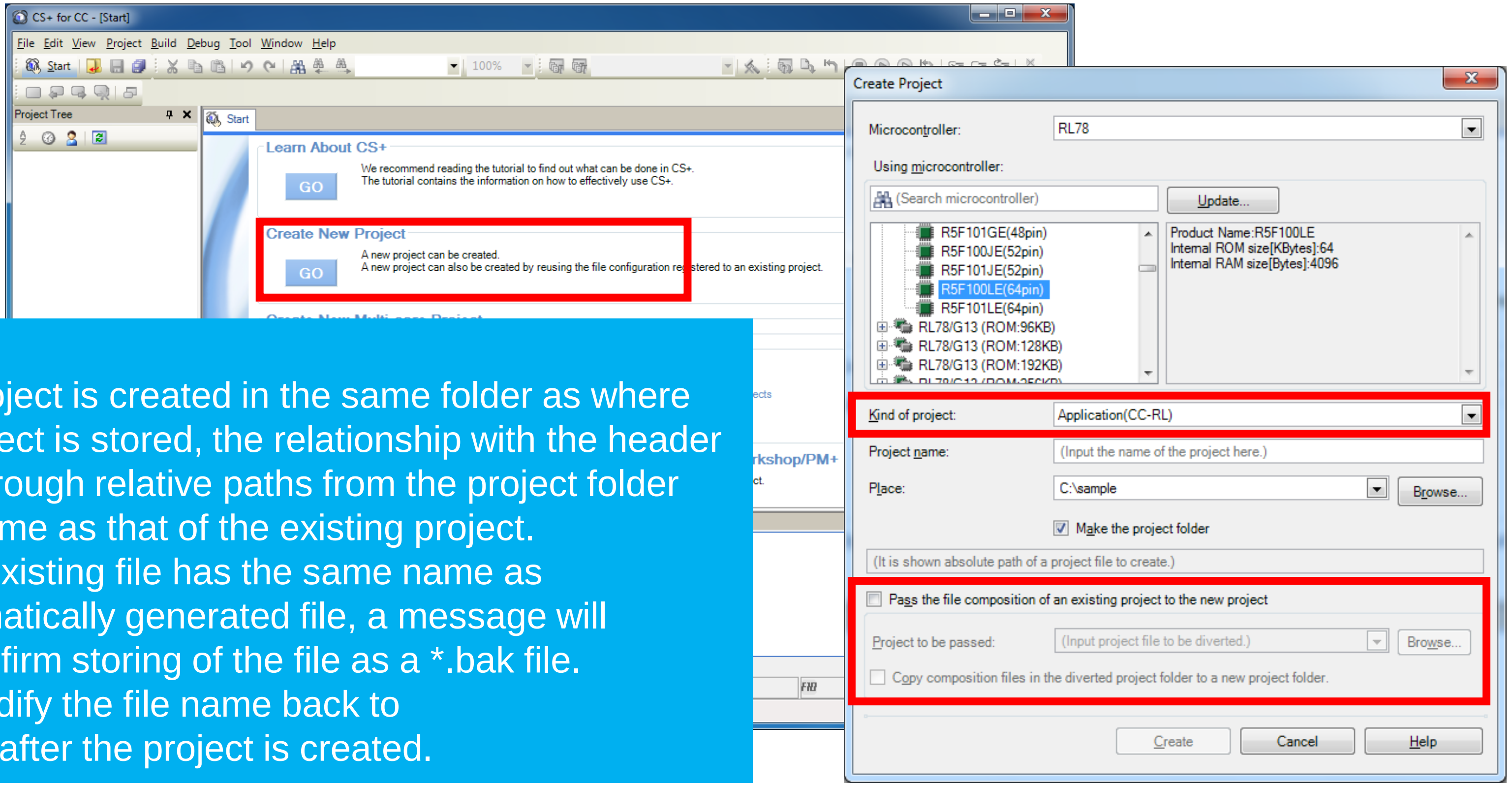
Project to be passed: (Input project file to be diverted.) Browse...

[ ] Copy composition files in the diverted project folder to a new project folder.

Create | Cancel | Help

Utilizing an Existing Project (1/2)

During the process of creating a new project, select [Pass the file composition of an existing project to the new project]. Then select a project that was created using CA78K0R.



Learn About CS+

We recommend reading the tutorial to find out what can be done in CS+.
The tutorial contains the information on how to effectively use CS+.

Create New Project

A new project can be created.
A new project can also be created by reusing the file configuration registered to an existing project.

Create Project

Microcontroller: RL78

Using microcontroller:

(Search microcontroller) Update...

R5F101GE(48pin)
R5F100JE(52pin)
R5F101JE(52pin)
R5F100LE(64pin)
R5F101LE(64pin)
RL78/G13 (ROM:96KB)
RL78/G13 (ROM:128KB)
RL78/G13 (ROM:192KB)

Product Name:R5F100LE
Internal ROM size[KBytes]:64
Internal RAM size[Bytes]:4096

Kind of project: Application(CC-RL)

Project name: (Input the name of the project here.)

Place: C:\sample Browse...

☑ Make the project folder

(It is shown absolute path of a project file to create.)

☐ Pass the file composition of an existing project to the new project

Project to be passed: (Input project file to be diverted.) Browse...

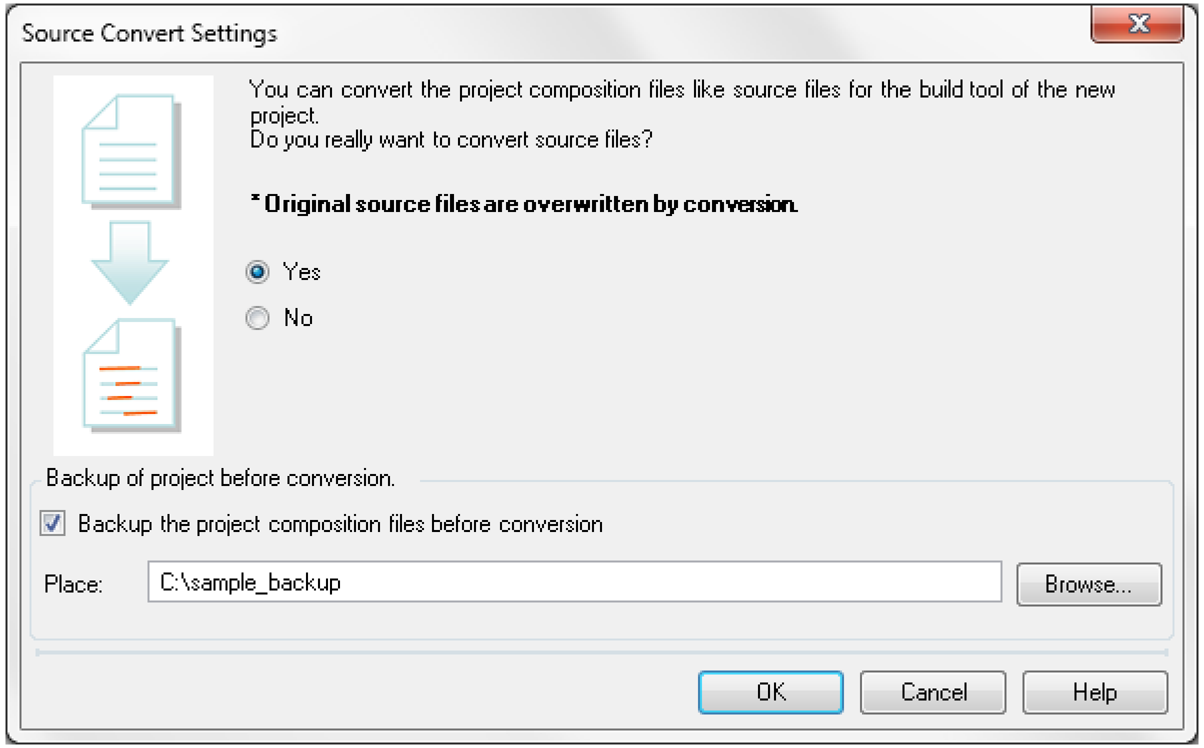
☐ Copy composition files in the diverted project folder to a new project folder.

Create Cancel Help

Remark:
When a new project is created in the same folder as where the existing project is stored, the relationship with the header files included through relative paths from the project folder becomes the same as that of the existing project.
However, if an existing file has the same name as that of an automatically generated file, a message will be output to confirm storing of the file as a *.bak file.
In this case, modify the file name back to the original one after the project is created.

Utilizing an Existing Project (2/2)

When you create a new project, you can convert the existing source files of CA78K0R compiler to the source files of CC-RL.


Source Convert Settings

You can convert the project composition files like source files for the build tool of the new project.
Do you really want to convert source files?

*Original source files are overwritten by conversion.

[✓ Yes] No

Backup of project before conversion.

☑ Backup the project composition files before conversion

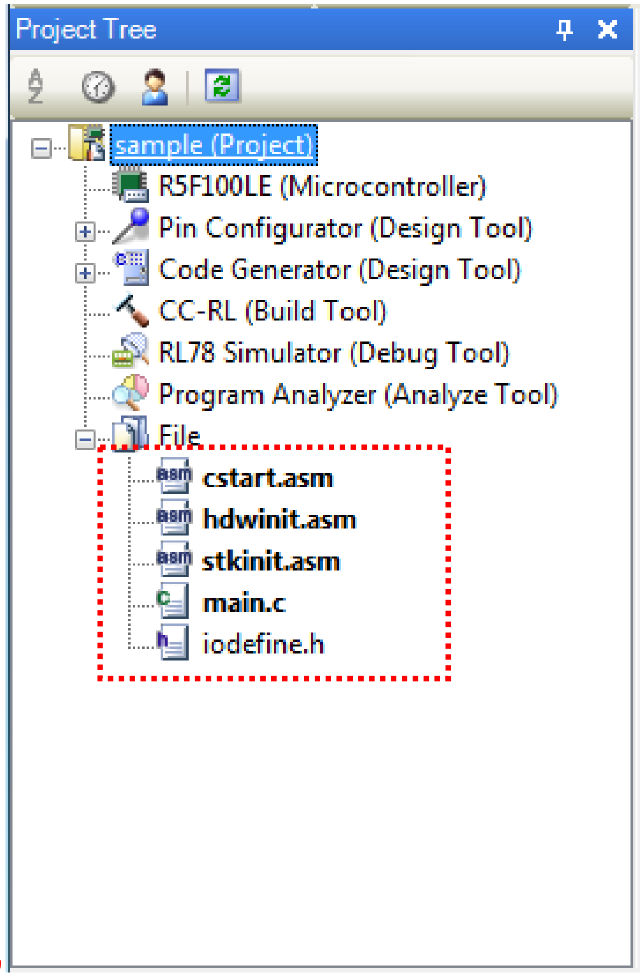
Place: C:\sample_backup

Differences from CA78K0R Projects

Generated Files

When a new project is created for CC-RL, the following files necessary for development will be generated.

- ●Startup file (cstart.asm)
- ●hdwinit initial-setting function file (hdwinit.asm)
- ●stkinit stack initial-setting function (stkinit.asm)
  (This is not output for an MCU with the RL78-S1 core.)
- ●main function file (main.c)
- ●SFR file (iodefine.h)

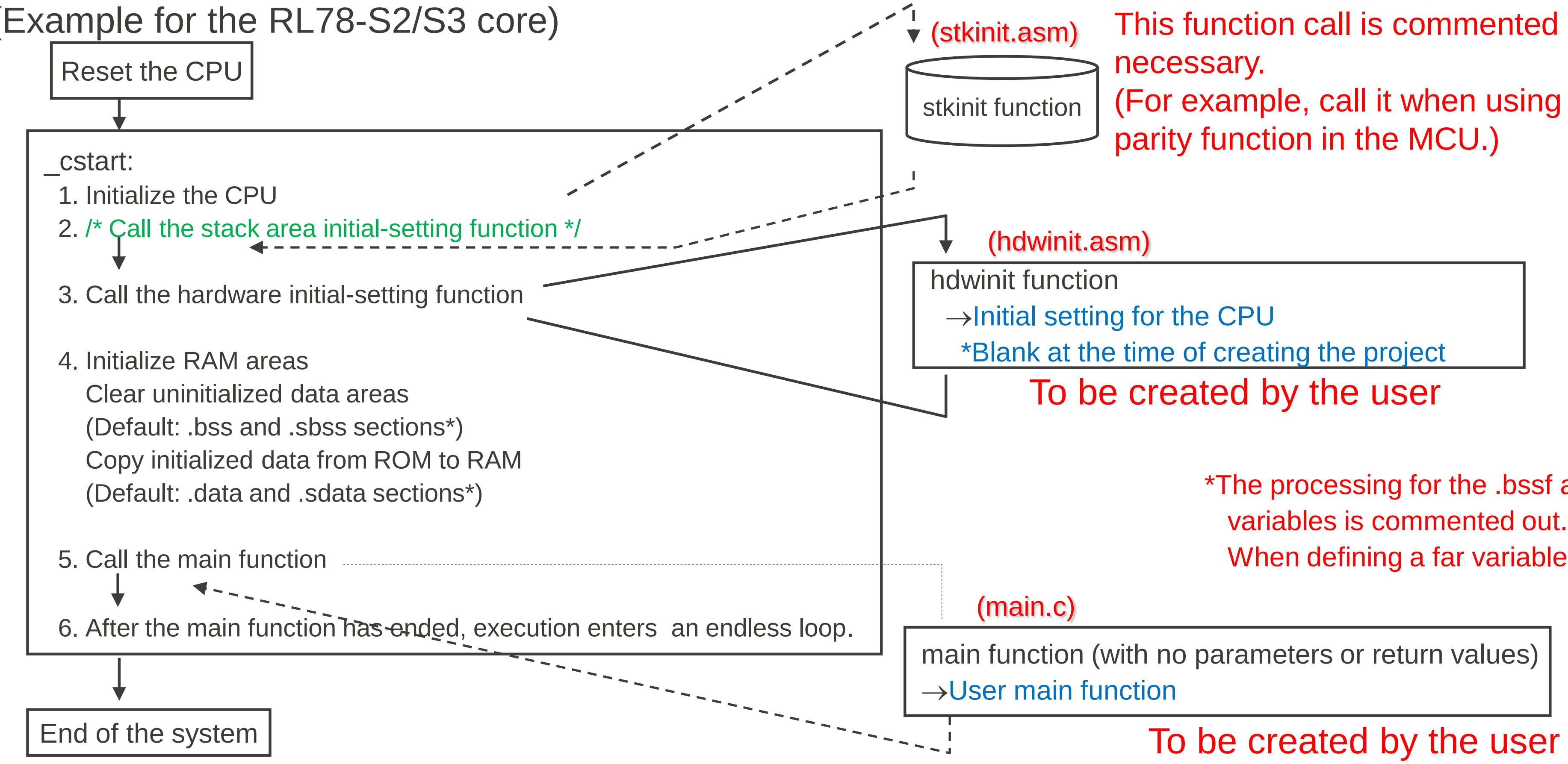

Project Tree
sample (Project)
R5F100LE (Microcontroller)
Pin Configurator (Design Tool)
Code Generator (Design Tool)
CC-RL (Build Tool)
RL78 Simulator (Debug Tool)
Program Analyzer (Analyze Tool)
File
cstart.asm
hdwinit.asm
stkinit.asm
main.c
iodefine.h

Note:
When a project is created for CA78K0R, these files for CC-RL will not be generated.

Startup File (1/2)

The following shows the contents of the startup file registered in the project tree.

(Example for the RL78-S2/S3 core)



Reset the CPU
_cstart:
1. Initialize the CPU
2. /* Call the stack area initial-setting function */
3. Call the hardware initial-setting function
4. Initialize RAM areas
Clear uninitialized data areas
(Default: .bss and .sbss sections*)
Copy initialized data from ROM to RAM
(Default: .data and .sdata sections*)
5. Call the main function
6. After the main function has ended, execution enters an endless loop.
End of the system
(stkinit.asm)
stkinit function
(hdwinit.asm)
hdwinit function
→Initial setting for the CPU
*Blank at the time of creating the project
To be created by the user
(main.c)
main function (with no parameters or return values)
→User main function
To be created by the user

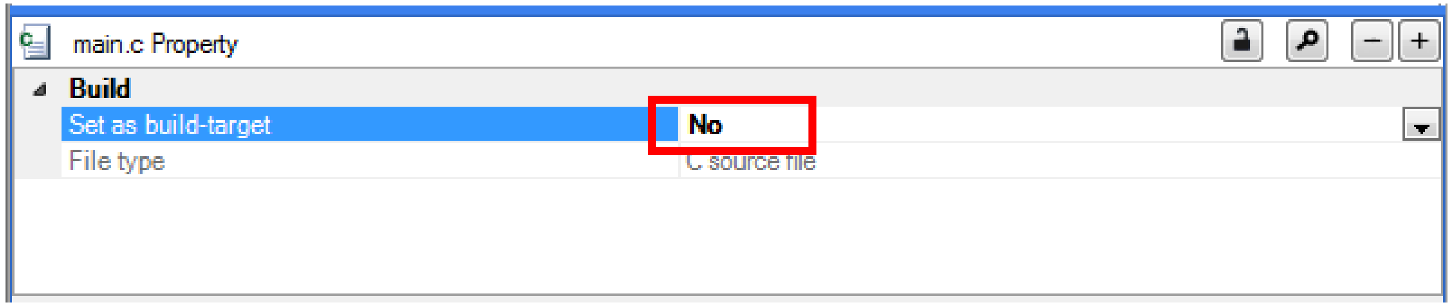
This function call is commented out; call it as necessary.
(For example, call it when using the RAM parity function in the MCU.)

*The processing for the .bssf and .dataf sections for far variables is commented out.
When defining a far variable, enable this processing.

Startup File (2/2)

When the main function and hdwinit function are registered in the existing project, use either of the following two ways to exclude the files that are automatically generated during project creation from the target of build.

- Delete the files from the project tree.
- Select [No] for "Set as build-target" in the property of the main.c and hdwinit.asm files registered in the project tree.



main.c Property

Build

Set as build-target	No
File type	C source file

iodefine.h (1/2)

The declarations in this file can be used in a C source file to access SFRs in the RL78.

```
<iodefine.h>
• • •
typedef struct
{
    unsigned char no0:1;
    unsigned char no1:1;
    unsigned char no2:1;
    unsigned char no3:1;
    unsigned char no4:1;
    unsigned char no5:1;
    unsigned char no6:1;
    unsigned char no7:1;
} __bitf_T;
• • •
#define ADM2      (*(volatile __near unsigned char  *)0x10)
#define ADM2_bit (*(volatile __near __bitf_T *)0x10)
#define P0          (*(volatile __near unsigned char  *)0xFF00)
#define P0_bit      (*(volatile __near __bitf_T *)0xFF00)
#define P1          (*(volatile __near unsigned char  *)0xFF01)
#define P1_bit      (*(volatile __near __bitf_T *)0xFF01)
 • • •
#define INTP0                      0x0008
#define INTP1                      0x000A
#define INTP2                      0x000C
#define INTP3                      0x000E
 • • •
```

```
<Registers are accessed from this file>
#include "iodefine.h"
• • •
void main(void)
{
        • • •
        ADM2 = 0x12;
        ADTYP = 1;
        P0_bit.no2 = 1;
        • • •
}
• • •
#pragma interrupt inter (vect=INTP0)
void __near inter ( void ) {
      /*Interrupt processing*/
}
• • •
```

<How to Access>

- Use the descriptions in the iodefine.h file to access SFRs and their bits.
- For the bits that are not registered as reserved words, use the names with suffix "_bit" to access them.
- Specify #pragma interrupt to use interrupt request names.

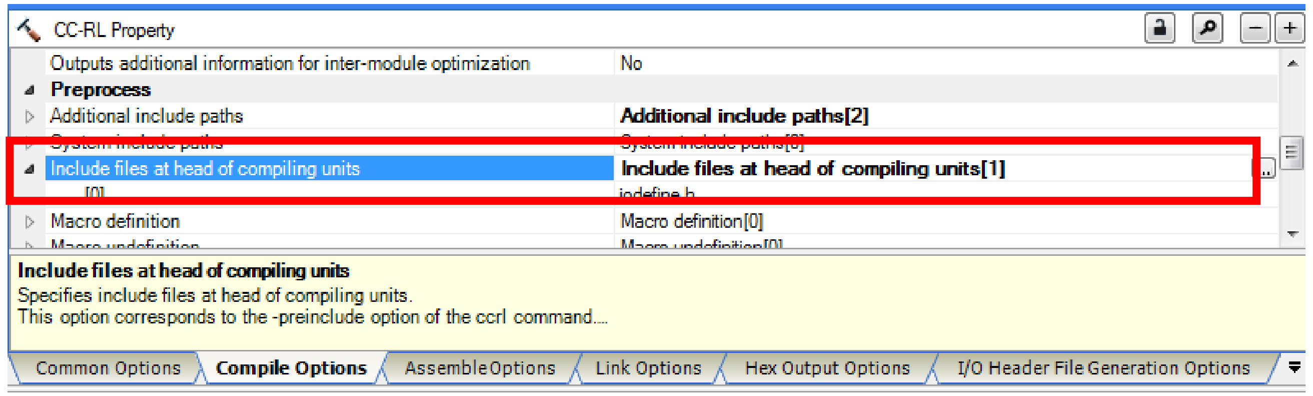
RENESAS

iodefine.h (2/2)

It’s possible by the next one of ways for inclusion iodefine.h to source files.

- ●Write #include "iodefine.h" in each source file.
  - - Description is needed every each source file.
  - - It‘s necessary to do inclusion before an interrupt request name of vector table designation(#pragma interrupt) and a description of SFR access.
- ●Specify -preinclude=iodefine.h by a compilation option.
  - －It's applied to all source files.
  - - When SFR name and an interrupt request name are used by the different use, #define is replaced by a definition in iodefine.h.

```
<example of C source file>
#include “iodefine.h”
• • •
void main(void)
{
        ADM2 = 0x12;
        P0_bit.no2 = 1;
}
#pragma interrupt inter (vect=INTP0)
void __near inter ( void ) {
      /*Interrupt processing*/
}
```



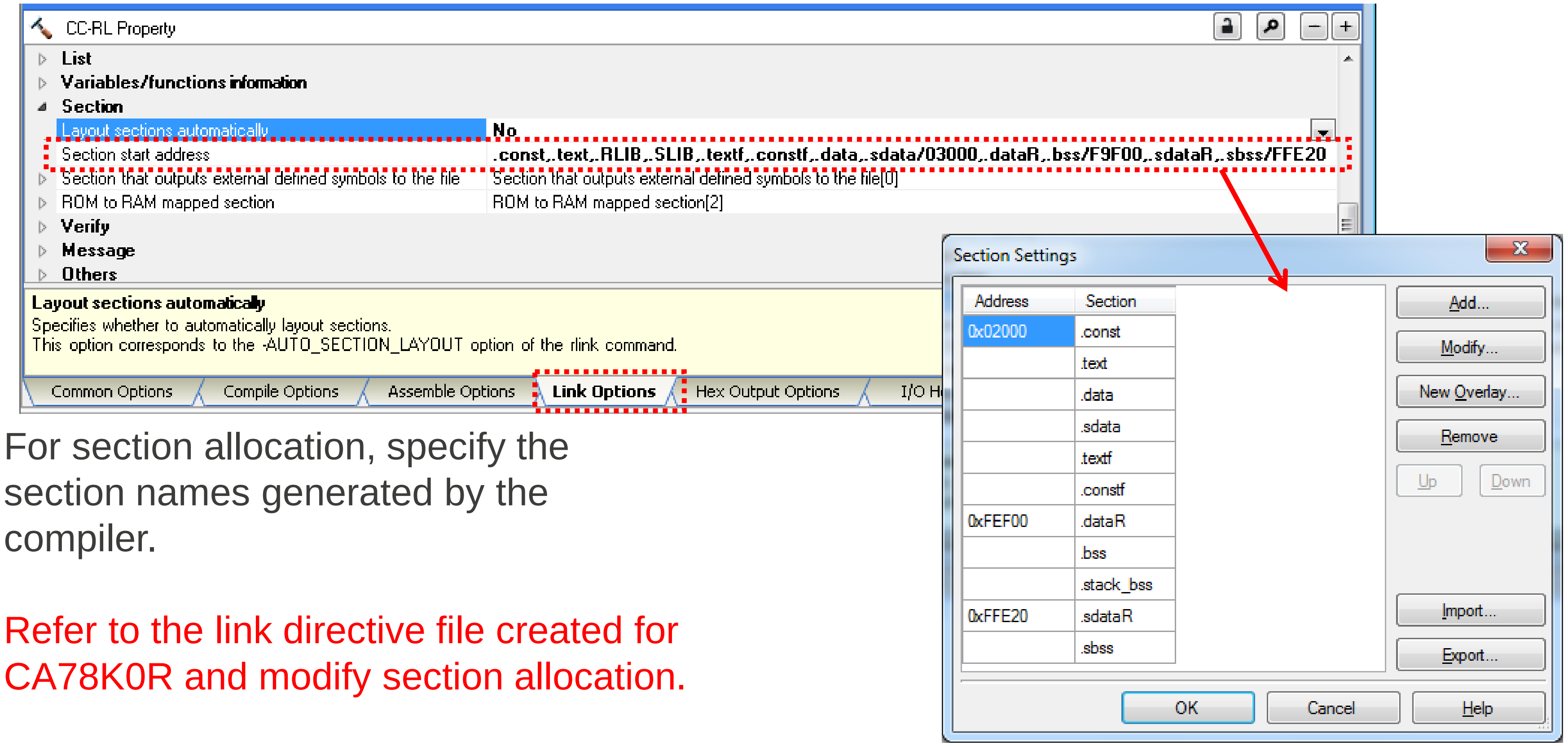
CC-RL Property

Outputs additional information for inter-module optimization	No
Preprocess	
Additional include paths	**Additional include paths[2]**
System include paths	System include paths[0]
Include files at head of compiling units	**Include files at head of compiling units[1]**
[0]	iodefine.h
Macro definition	Macro definition[0]
Macro undefinition	Macro undefinition[0]

Include files at head of compiling units
Specifies include files at head of compiling units.
This option corresponds to the -preinclude option of the ccrl command....

Section Allocation (1/3)

Specify allocation of program and data sections on the Link Options tabbed page of the Property panel.



CC-RL Property

- List
- Variables/functions information
- Section

Layout sections automatically	No
Section start address	.const,.text,.RLIB,.SLIB,.textf,.constf,.data,.sdata/03000,.dataR,.bss/F9F00,.sdataR,.sbss/FFE20
Section that outputs external defined symbols to the file	Section that outputs external defined symbols to the file[0]
ROM to RAM mapped section	ROM to RAM mapped section[2]

- Verify
- Message
- Others

Layout sections automatically
Specifies whether to automatically layout sections.
This option corresponds to the -AUTO_SECTION_LAYOUT option of the rlink command.

Section Settings

Address	Section
0x02000	.const
	.text
	.data
	.sdata
	.textf
	.constf
0xFEF00	.dataR
	.bss
	.stack_bss
0xFFE20	.sdataR
	.sbss

For section allocation, specify the section names generated by the compiler.

Refer to the link directive file created for CA78K0R and modify section allocation.

Any section can be allocated to a desired address.

Section Allocation (2/3)

CC-RL generates sections with default section names.

Default Section Name	Relocation Attribute	Description
.callt0	CALLT0	Section for the callt function call table
.text	TEXT	Section for code (allocated to near area)
.textf	TEXTF	Section for code (allocated to far area)
.textf_unit64kp	TEXTF_UNIT64KP	Section for code (the section is allocated so that the start address is an even number and the section does not extend over a 64-KB - 1 boundary)
.const	CONST	ROM data (allocated to near area) (in mirror area)
.constf	CONSTF	ROM data (allocated to far area)
.data	DATA	Section for initialized data (with initial values, allocated to near area)
.dataf	DATAF	Section for initialized data (with initial values, allocated to far area)
.sdata	SDATA	Section for initialized data (with initial values, variables allocated to saddr)

Section Allocation (3/3)

(Note) : The section name cannot be modified through #pragma section.

Default Section Name	Relocation Attribute	Description
.sbss_bit	SBSS_BIT	Section for bit data area (uninitialized, variables allocated to saddr)
.bss_bit	BSS_BIT	Section for bit data area (uninitialized, allocated to near area)
.bss	BSS	Section for data area (uninitialized, allocated to near area)
.bssf	BSSF	Section for data area (uninitialized, allocated to far area)
.sbss	SBSS	Section for data area (uninitialized, variables allocated to saddr)
.option_byte	OPT_BYTE	Section dedicated for user option byte and on-chip debug settings
.security_id	SECUR_ID	Section dedicated for security ID setting
.vect(Note)	AT	interrupt vector table
.dataR	DATA	Section for initialized data RAM (initialized, allocated to near area) Defined in the startup file.
.sdataR	DATA	Section for initialized data RAM (initialized, allocated to saddr area) Defined in the startup file.
.RLIB(Note)	TEXTF	Section for code of runtime libraries.
.SLIB(Note)	TEXTF	Section for code of standard libraries.

Optimization Options (1/4)

- The optimization techniques of the Renesas compilers and linkage editors have been enhanced to a higher level that matches the RL78 MCUs (optimum register assignment, optimum instruction selection, instruction scheduling, etc.) to generate compact codes.

● Optimization by the compiler

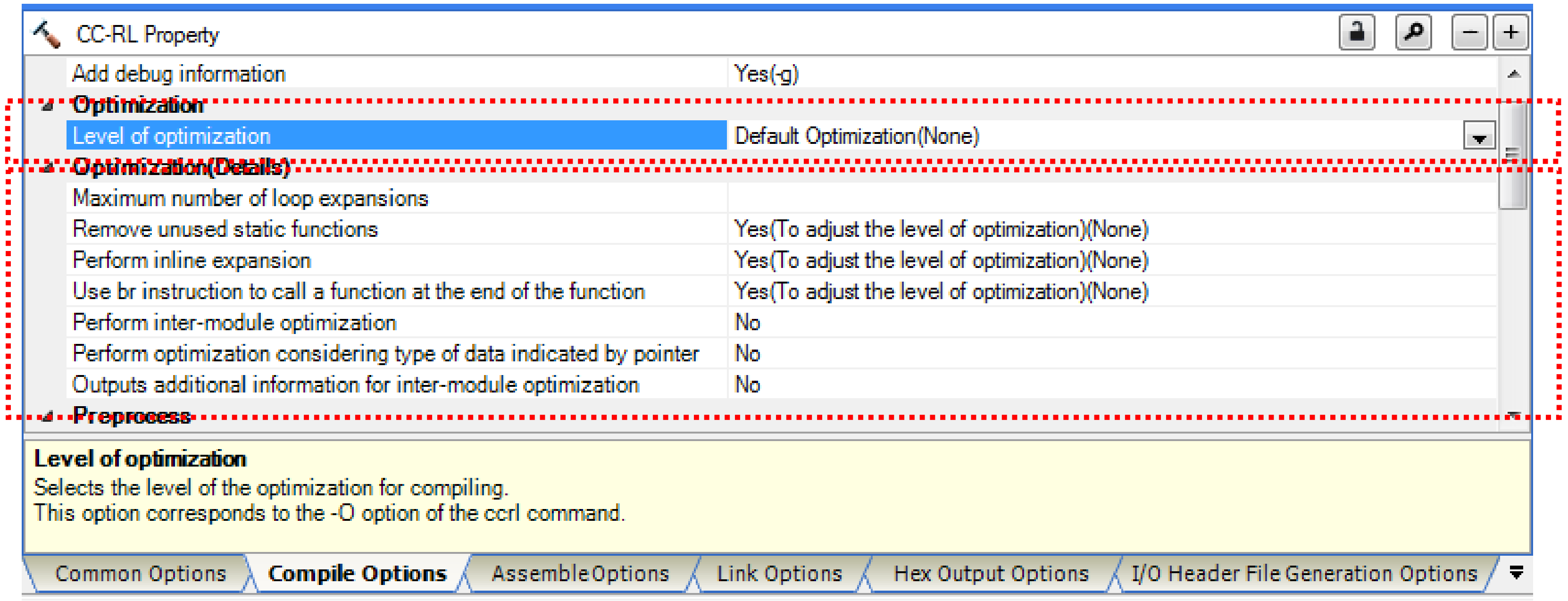
- Easy selection of optimization mode
  – Selection of size or speed precedence
- Wide-range optimization at compilation
  – Inline expansion of functions in multiple files
- Detailed optimization settings
  – Loop expansion, inline function expansion, replacement of a function call at the end of a function with a br instruction, etc.

● Optimization by the optimizing linkage editor

- Inter-module optimization
  – Branch instruction optimization
- Detailed settings for disabling optimization

Optimization Options (2/4)

Specify options on the Compile Options tabbed page in the CC-RL (build tool) Property panel.



CC-RL Property

Add debug information	Yes(-g)
Optimization	
Level of optimization	Default Optimization(None)
Optimization(Details)	
Maximum number of loop expansions	
Remove unused static functions	Yes(To adjust the level of optimization)(None)
Perform inline expansion	Yes(To adjust the level of optimization)(None)
Use br instruction to call a function at the end of the function	Yes(To adjust the level of optimization)(None)
Perform inter-module optimization	No
Perform optimization considering type of data indicated by pointer	No
Outputs additional information for inter-module optimization	No
Preprocess	

Level of optimization
Selects the level of the optimization for compiling.
This option corresponds to the -O option of the ccrl command.

Optimization Options (3/4)

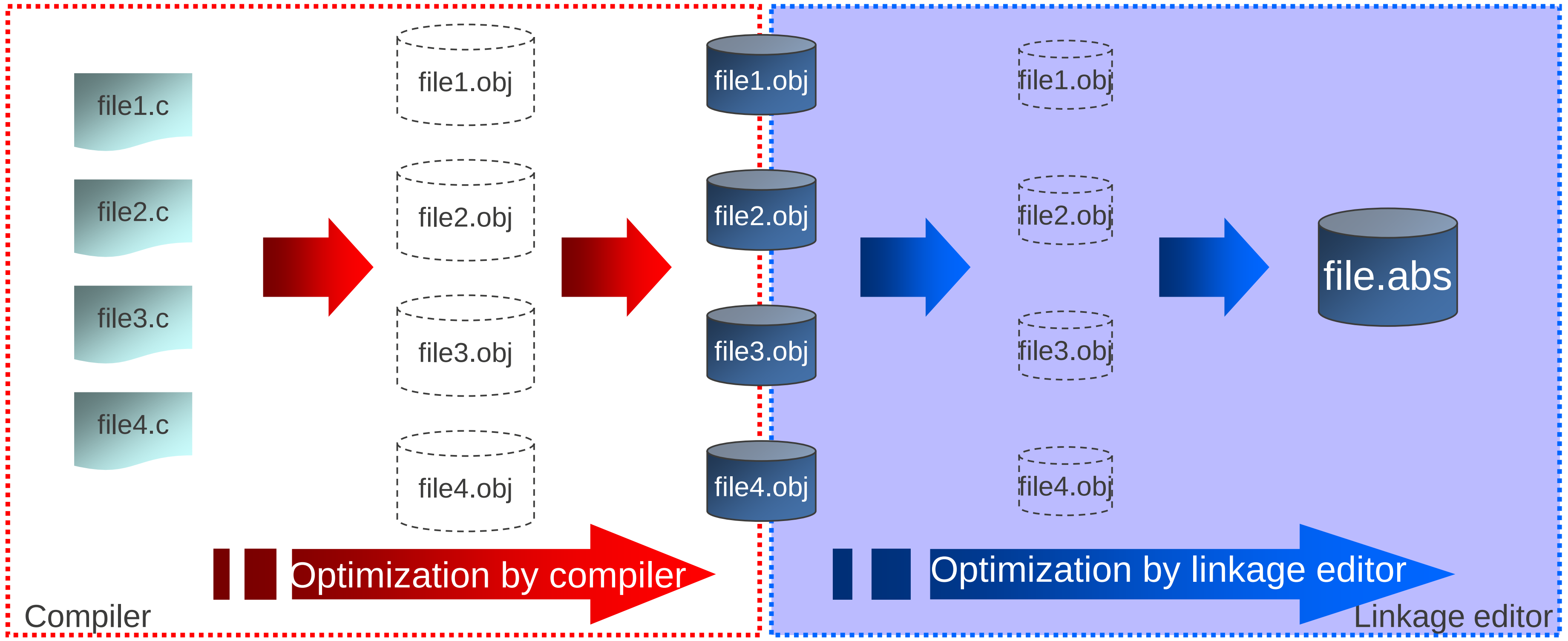
Optimization options can be specified as follows through the selection of ROM size precedence or execution speed precedence. In addition, optimization can be fine-tuned through detailed setting items.

Optimization Item	Description	Optimization Level			
		-Osize	-Ospeed	-Odefault	-Onothing
unroll	Loop expansion (the maximum rate of increase in code size after loop expansion)	1	2	1	1
delete_static_func	Deletion of unused static functions	on	on	on	off
inline_level	Inline expansion of functions (level of expansion)*1	3	2	3	-
inline_size	Size of inline expansion *2	0	100	0	-
tail_call	Replacement of a function call at the end of a function with a br instruction	On	On	On	Off

*1 Level of expansion
0: Suppresses all inline expansion including the function for which #pragma inline is specified.
1: Performs inline expansion for only a function for which #pragma inline is specified.
2: Distinguishes a function that is the target of expansion automatically and expands it.
3: Distinguishes the function that is the target of expansion automatically and expands it, while minimizing the increase in code size.
However, even if 1 to 3 is specified, the function specified by #pragma inline may not be expanded depending on the contents of the function and the status of compilation.
*2 Size of inline expansion : This specifies the maximum increasing rate (%) of the code size up to which inline expansion is performed.

Optimization Options (4/4)

The RL78 build environment provides optimization by the linkage editor in addition to optimization by the compiler. The information, such as allocation addresses, obtained at linkage is used for optimization to generate more efficient codes.



Revision History

Revision	Description	Page
Rev.1.00	First revision	-
Rev.1.01	Modification of version number of CS+	P2
	Addition of method including iodefine.h file	P12
Rev.1.02	Modification of version number of CS+ and CC-RL	-

Renesas System Design Co., Ltd.