RH850 Development Environment Migration Guide

R020UT3233EJ0101
Rev.1.01
Apr. 21, 2017

Migration from V850E2 Family to RH850 Family (Compiler ed.) (CX to CC-RH)

Introduction

This document describes the points that should be especially borne in mind for migration from the V850E2 compiler (hereafter called the CX) to the RH850 family compiler (hereafter called the CC-RH).

Contents

1. Options

This section shows a table that compares the CX options with their corresponding CC-RH options. Note that the CC-RH distinguishes between uppercase and lowercase letters in the compiler options and assembler options, but does not distinguish between them in the linkage editor options.

1.1 Compiler Options

Classification	Description	CX (cx.exe)	CC-RH (ccrh.exe)
Version/help display specification	This option displays the version information.	-V	-V
	This option displays the descriptions of options.	-h	-h
Output file specification	This option specifies the output file name.	-o	-o
	This option specifies where the object module file generated during compiling is to be saved.	-Xobj_path	-Xobj_path
	This option specifies where an assembly-language source file generated during compilation is to be saved.	-Xasm_path	-Xasm_path
	This option specifies where an assemble list file is to be saved.	-Xprn_path	-Xasm_option=-Xprn_path
	This option specifies the temporary folder.	-Xtemp_path	None.
	This option saves the load module file before ROMization processing.	-Xlink_output	None. (Note 1)
Source debugging control	This option outputs information for source debugging.	-g	-g
	This option prohibits changing the memory access size.	-Xkeep_access_size	None.
Device specification	This option specifies the target device.	-C	None. (Note 2)
	This option specifies that an object module file common to the various devices is generated.	-Xcommon	-Xcommon (Note 3)
	This option specifies the folder to search for device files.	-Xdev_path	None. (Note 2)
	This option sets the start address of the programmable peripheral I/O register.	-Xprogrammable_io	None.
Processing interrupt specification	This option executes only preprocessing for the input file.	-P	-P
	This option does not execute processing after assembling.	-S	-S
	This option does not execute processing after linking.	-c	-c
Preprocessor control	This option defines preprocessor macros.	-D	-D
	This option deletes the preprocessor macro definitions by the –D option.	-U	-U
	This option specifies the folder to search for include files.	-I	-I
	This option controls outputting the result of preprocessing.	-Xpreprocess	-Xpreprocess

Note 1: The CC-RH does not execute ROMization by default. To enable it, the user should specify the "–ROm" option at linkage.

Note 2: The CC-RH does not support the use of device files.

Note 3: The specifiable parameters differ between the CX and CC-RH.

Classification	Description	CX (cx.exe)	CC-RH (ccrh.exe)
C language control	This option processes as to make C source programs comply strictly with the ANSI standard.	-Xansi	-Xansi
	This option specifies signed for char type.	-Xchar	None. (Note 4)
	This option specifies which integer type the enumeration type handles.	-Xenum_type	-Xenum_type
	This option handles tentative definitions of variables as full definitions.	-Xdef_var	None. (Note 5)
Variable/function information specification	This option specifies the use of a symbol information file.	-Xsymbol_file	None.
Japanese/Chinese character control	This option specifies the Japanese/Chinese character code.	-Xcharacter_set	-Xcharacter_set
Optimization specification	This option specifies the optimization level or the details of each optimization item.	-O	-O (Note 6)
	This option deletes unused functions.	-Xdelete_func	None.
	This option sorts external variables.	-Xsort_var	None.
	This option performs inline expansion of standard library function calls "strcpy", "strcmp", "memcpy", and "memset".	-Xinline_strcpy	-Xinline_strcpy
	This option specifies whether or not to perform prologue/epilogue processing of the function through runtime library calls.	-Xpro_epi_runtime	None.
	This option suppresses inline expansion of the library function.	-Xcall_lib	None.
	This option merges string literals.	-Xmerge_string	-Xmerge_string
Generated code control	This option performs the structure packing.	-Xpack	-Xpack (Note 7)
	This option outputs a C source program as a comment to the assembly-language source file.	-Xpass_source	-Xpass_source
	This option specifies a mode in which the code of a switch statement is to be output.	-Xswitch	-Xswitch
	This option generates a 4-byte branch table.	-Xword_case	None.
	This option specifies the register where external variables are to be allocated.	-Xr	None.
	This option specifies the register mode.	-Xreg_mode	-Xreg_mode (Note 8)
	This option specifies the maximum size of data allocated to the .sdata or .sbss section.	-Xsdata	-Xsection (note 9)
	This option specifies that constant data is allocated to the .sconst section.	-Xsconst	
	This option controls generating floating-point calculation instructions.	-Xfloat	-Xfloat
	This option controls outputting far jump.	-Xfar_jump	-Xfar_jump
	This option generates the div and divu instructions for division.	-Xdiv	-Xdiv
Assembler control specification	This option controls outputting far jump for an assembly-language source file.	-Xasm_far_jump	-Xasm_option=-Xasm_far_jump

Note 4: The CC-RH always handles unsigned char types as signed types.
Note 5: The CC-RH always handles tentative definitions of variables as full definitions.
Note 6: The specifiable parameters differ between the CX and CC-RH.
Note 7: The CC-RH does not allow a value of 8 to be specified as a parameter.
Note 8: The CC-RH does not provide the 26-register mode.
Note 9: This option was added in CC-RH V1.02.00; it collectively changes the default allocation of variables to sections.
Note 10: Both the CX and CC-RH allocate constant data to the .const section by default.

Classification	Description	CX (cx.exe)	CC-RH (ccrh.exe)
Library link control	This option specifies the library file to be used during linking.	-l	-Xlk_option=-LIBrary
	This option specifies the folder to search for library files.	-L	None. (Note 11)
	This option suppresses linking the standard library.	-Xno_stdlib	None. (Note 12)
	This option suppresses linking the startup routine.	-Xno_startup	None. (Note 13)
	This option specifies the startup routine.	-Xstarup	None. (Note 13)
ROMization control	This option suppresses ROMization processing.	-Xno_romize	None. (Note 14)
	This option specifies the ROMization area reservation code file.	-Xrompcrt	None.
	This option specifies the start address of the rompsec section.	-Xrompsec_start	None.
	This option specifies the data section included in the rompsec section.	-Xrompsec_data	None.
	This option specifies the text section included in the rompsec section.	-Xrompsec_text	None.
	This option generates the load module file that has only the rompsec section.	-Xrompsec_only	None.
	This option omits error checking under ROMization.	-Xromize_check_off	None.
Link control	This option specifies the link directive file.	-Xlink_directive	None. (Note 15)
	This option outputs the link map file.	-Xmap	-Xlk_option=-LISt
	This option outputs symbol information to the link map file.	-Xsymbol_dump	-Xlk_option=-SHow
	This option specifies the security ID.	-Xsecurity_id	None. (Note 16)
	This option sets the user option bytes.	-Xoprion_byte	None. (Note 16)
	This option specifies the entry point address.	-Xentry_address	-Xlk_option=-ENTry
	This option generates the relocatable object module file.	-Xrelinkable_object	-Xlk_option= -FOrm=Relocate
	This option outputs detailed information when different register modes are used together.	-Xregmode_info	None.

Note 11: The user should specify the folder to search for library files as a parameter of -Xlk_option=-LIBrary.

Note 12: The CC-RH does not link the default standard library.

Note 13: The CC-RH handles the startup routine as an ordinary source file.

Note 14: The CC-RH linkage editor does not execute ROMization by default. To enable it, the user should specify the "–ROm" option at linkage.

Note 15: The CC-RH does not support the use of link directive files.

The user should specify the start addresses of section allocation through the "–STARt" linkage editor option.

Note 16: The security ID and option bytes for the RH850 should be specified through a flash programmer or the like.

Classification	Description	CX (cx.exe)	CC-RH (ccrh.exe)
Link control	This option continues link processing when the internal ROM/RAM overflows.	-Xforce_link	None. (Note 17)
	This option outputs information that can be used as a reference value for the parameter of the -Xsdata option to the standard output.	-Xsdata_info	None.
	This option performs linking in the 2-pass mode.	-Xtwo_pass_link	None.
	This option continues link processing if an illegality is found during relocation processing when linking.	-Xignore_address_error	None. (Note 17)
	This option outputs an error message for all multi-defined external symbols.	-Xmultiple_symbol	None. (Note 17)
	This option suppresses checking when linking.	-Xlink_check_off	None. (Note 17)
	This option specifies the filling value of align holes.	-Xalign_fill	None. (Note 18)
	This option rescans the library file specified by the -l option.	-Xrescan	None.
	This option generates the load module file from which the debugging information, line number information, and global pointer table have been removed.	-Xstrip	-Xlk_option=-NODEBug
Hex output control	This option specifies the hex file name.	-Xhex	-Xlk_option=-OUtput
	This option executes only hex output.	-Xhex_only	-Xlk_option=-FOrm
	This option specifies the format of the hex file to be output.	-Xhex_format	-Xlk_option=-FOrm
	This option specifies fill processing of the hex file.	-Xhex_fill	-Xlk_option=-SPace
	This option converts the codes in the specified section to hex format and outputs them.	-Xhex_section	-Xlk_option=-OUtput
	This option specifies the maximum length of the block.	-Xhex_block_size	-Xlk_option=-BYte_count
	This option specifies the offset of the address to be output.	-Xhex_offset	None.
	This option generates as many null characters as the size of the section of data without initial values.	-Xhex_null	None.
	This option converts the symbol table and outputs it.	-Xhex_symtab	None.
	This option does not use the information of the internal ROM area when the hex file is filled.	-Xhex_rom_less	None.
	This option outputs the result of the CRC operation.	-Xcrc	None.
	This option specifies the method for the CRC operation.	-Xcrc_method	None.

Note 17: Processing can be continued by changing the type of messages from an error to a warning through the "-CHange_message" option for the CC-RH linkage editor.

Note 18: Part of the functions of this option can be implemented by using CC-RH linkage editor options such as –PADDING and -Space.

Classification	Description	CX (cx.exe)	CC-RH (ccrh.exe)
Information file output control	This option outputs the static analysis information file.	-Xcref	-Xcref
	This option suppresses the output of the static analysis information file.	-Xno_cref	None.
	This option generates a symbol information file.	-Xsfg	None.
	This option outputs the optimum allocation information.	-Xsfg_opt	None.
	This option specifies the size of .tidata section.	-Xsfg_size_tidata	None.
	This option specifies the size of .tidata.byte section.	-Xsfg_size_tidata_byte	None.
	This option specifies the size of .sidata section.	-Xsfg_size_sidata	None.
	This option specifies the size of .sedata section.	-Xsfg_size_sedata	None.
	This option specifies the size of .sdata section.	-Xsfg_size_sdata	None.
Error output control	This option outputs error messages to a file.	-Xerror_file	-Xerror_file
Warning message output control	This option outputs the specified warning message.	-Xwarning	None.
	This option suppresses outputting warning messages of the specified numbers.	-Xno_warning	-Xno_warning
Phase individual option specification	This option specifies the file to be assembled.	-Xasm_option	-Xasm_option
	This option specifies the file to be linked.	-Xlk_option	-Xlk_option
	This option specifies an option for the common optimization module.	-Xopt_option	None.
Command file specification	This option specifies a command file.	@	@

1.2 Assembler Options

Classification	Description	CX (cx.exe)	CC-RH (asrh.exe)
Version/help display specification	This option displays the version information.	-V	-V
	This option displays the descriptions of options.	-h	-h
Output file specification	This option specifies the output file name.	-o	-o
	This option specifies where the object module file generated during compilation is to be saved.	-Xobj_path	-Xobj_path
	This option specifies where an assemble list file is to be saved.	-Xprn_path	-Xprn_path
Source debugging control	This option outputs information for source debugging.	-g	-g
Device specification	This option specifies the target device.	-C	None. (Note 1)
	This option specifies that an object module file common to the various devices is generated.	-Xcommon	-Xcommon (Note 2)
	This option specifies the folder to search for device files.	-Xdev_path	None. (Note 1)
	This option sets the start address of the programmable peripheral I/O register.	-Xprogrammable_io	None.
Preprocessor control	This option defines assembly-language macros.	-D	-D
	This option deletes the assembly-language symbol definitions by the –D option.	-U	-U
	This option specifies the folder to search for include files.	-I	-I
Japanese/Chinese character control	This option specifies the Japanese/Chinese character code.	-Xcharacter_set	-Xcharacter_set
Generated code control	This option specifies the register mode.	-Xreg_mode	-Xreg_mode (Note 3)
	This option specifies the maximum size of data allocated to the .sdata or .sbss section.	-Xsdata	None. (Note 4)
Assembler control specification	This option controls outputting far jump for an assembly-language source file.	-Xasm_far_jump	-Xasm_far_jump
Error output control	This option outputs error messages to a file.	-Xerror_file	-Xerror_file
Warning message output control	This option outputs the specified warning message.	-Xwarning	None.
	This option suppresses outputting warning messages of the specified numbers.	-Xno_warning	-Xno_warning
Command file specification	This option specifies a command file.	@	@

Note 1: The CC-RH does not support the use of device files.

Note 2: The specifiable parameters differ between the CX and CC-RH.

Note 3: The CC-RH does not provide the 26-register mode.

Note 4: By default, the CX allocates variables to the .sdata or .sbss section, but the CC-RH allocates them to the .data or .bss section.

1.3 Linkage Editor Options

Classification	Description	CX (cx.exe)	CC-RH (rlink.exe)
Version/help display specification	This option displays the version information.	-V	None. (Note 1)
	This option displays the descriptions of options.	-h	None. (Note 1)
Output file specification	This option specifies the output file name.	-o	-output
	This option specifies the temporary folder.	-Xtemp_path	None.
Source debugging control	This option outputs information for source debugging.	-g	-debug
Device specification	This option specifies the target device.	-C	None. (Note 2)
	This option specifies the folder to search for device files.	-Xdev_path	None. (Note 2)
Library link control	This option specifies the library file to be used during linking.	-l	-library
	This option specifies the folder to search for library files.	-L	None.
	This option suppresses linking the standard library.	-Xno_stdlib	None. (Note 3)
Link directive file specification	This option specifies the link directive file.	-Xlink_directive	None. (Note 4)
Security ID control	This option specifies the security ID.	-Xsecurity_id	None. (Note 5)
User option byte control	This option sets the user option bytes.	-Xoprion_byte	None. (Note 5)
Force linking to continue	This option continues link processing when the internal ROM/RAM overflows.	-Xforce_link	None. (Note 6)
Entry point address specification	This option specifies the entry point address.	-Xentry_address	-entry
Link map file output specification	This option outputs the link map file.	-Xmap	-list
Symbol information output specification	This option outputs symbol information to the link map file.	-Xsymbol_dump	-show
Generating object module file control	This option generates the relocatable object module file.	-Xrelinkable_object	-form=relocate

Note 1: The CC-RH information is displayed by entering rlink[ENTER] from the command line.

Note 2: The CC-RH does not support the use of device files.

Note 3: The CC-RH does not link the default standard library.

Note 4: The CC-RH does not support the use of link directive files.

The user should specify the section allocation addresses through the "-start" option.

Note 5: The security ID and option bytes for the RH850 should be specified through a flash programmer or the like.

Note 6: Processing can be continued by changing the type of messages from an error to a warning through the "-change_message" linkage editor option.

Classification	Description	CX (cx.exe)	CC-RH (rlink.exe)
Generating object module file control	This option generates the relocatable object module file.	-Xrelinkable_object	-form=relocate
Control checking for mixing regarding register modes and device common objects	This option checks whether the specified register mode is mixed.	-Xreg_mode	None.
	This option outputs detailed information when different register modes are used together.	-Xregmode_info	None.
	This option checks for mixing of the device common object module file to be generated and the device specified by the -C option.	-Xcommon	None.
sdata/sbss information output specification	This option outputs information that can be used as a reference value for the parameter of the -Xsdata option to the standard output.	-Xsdata_info	None.
2-pass mode link specification	This option performs linking in the 2-pass mode.	-Xtwo_pass_link	None.
Relocation resolution error processing control	This option continues link processing if an illegality is found during relocation processing when linking.	-Xignore_address_error	None. (Note 6)
Symbol multiple definition error output specification	This option outputs an error message for all multi-defined external symbols.	-Xmultiple_symbol	None. (Note 6)
Link-time check suppress specification	This option suppresses checking when linking.	-Xlink_check_off	None. (Note 6)
Filling value specification	This option specifies the filling value of align holes.	-Xalign_fill	None. (Note 7)
Library file rescan specification	This option rescans the library file specified by the -l option.	-Xrescan	None.
Debugging information section output suppress specification	This option generates the load module file from which the debugging information, line number information, and global pointer table have been removed.	-Xstrip	-nodebug
Non-ROMized load module file save specification	This option saves the load module file before ROMization processing.	-Xlink_output	None. (Note 8)
Error output control	This option outputs error messages to a file.	-Xerror_file	None.
Warning message output control	This option outputs the specified warning message.	-Xwarning	None.
	This option suppresses outputting warning messages of the specified number.	-Xno_warinig	None.
Command file specification	This option specifies a command file.	@	-subcommand

Note 6: Processing can be continued by changing the type of messages from an error to a warning through the "-change_message" linkage editor option.

Note 7: Part of the functions of this option can be implemented by using CC-RH linkage editor options such as –PADDING and -Space.

Note 8: The CC-RH does not execute ROMization by default. To enable it, the user should specify the "–ROm" option at linkage.

2. Intrinsic Functions

This section shows a table that compares the CX intrinsic functions with their corresponding CC-RH intrinsic functions. Note that when an intrinsic function that is provided in the CX but not in the CC-RH is called, the CC-RH compiles it as an ordinary function. If the definition of the function is not found, an error will occur.

Instruction	Description	CX	CC-RH
di	Disables interrupts	void __DI(void);	void __DI(void);
ei	Enables interrupts	void __EI(void);	void __EI(void);
nop	No operation	void __nop(void);	void __nop(void);
halt	Stops the processor	void __halt(void);	void __halt(void);
satadd	Saturated addition	long a, b; long __satadd(a, b);	long a, b; long __satadd(a, b);
satsub	Saturated subtraction	long a, b; long __satsub(a, b);	long a, b; long __satsub(a, b);
bsh	Halfword data byte swap	long a; long __bsh(a);	long a; long __bsh(a);
bsw	Word data byte swap	long a; long __bsw(a);	long a; long __bsw(a);
hsw	Word data halfword swap	long a; long __hsw(a);	long a; long __hsw(a);
sxb	Byte data sign extension	char a; long __sxb(a);	None.
sxh	Halfword data sign extension	short a; long __sxh(a);	None.
mul	Instruction that assigns the result of 32-bit * 32-bit signed multiplication to a variable using mul instruction	long a, b; long long __mul(a, b);	None.
mulu	Instruction that assigns the result of 32-bit * 32-bit unsigned multiplication to a variable using mulu instruction	unsigned long a, b; unsigned long long __mulu(a,b);	None.
mul	Instruction that assigns the higher 32 bits of multiplication result to a variable using mul32 instruction	long a, b; long __mul32(a, b);	long a, b; long __mul32(a, b);
mulu	Instruction that assigns the higher 32 bits of unsigned multiplication result to a variable using mul32u instruction	unsigned long a, b; unsigned long __mul32u(a, b);	unsigned long a, b; unsigned long __mul32u(a, b);
sasf	Flag condition setting with logical left shift	long a; unsigned int b; long __sasf(a, b);	None.
sch0l	Bit (0) search from MSB side	long a; long __sch0l(a);	long a; long __sch0l(a);
sch0r	Bit (0) search from LSB side	long a; long __sch0r(a);	long a; long __sch0r(a);
sch1l	Bit (1) search from MSB side	long a; long __sch1l(a);	long a; long __sch1l(a);
sch1r	Bit (1) search from LSB side	long a; long __sch1r(a);	long a; long __sch1r(a);

ldsr	Loads to a system register	long regID; long a; void __ldsr(regID, a);	long regID; unsigned long a; void __ldsr(regID, a);
stsr	Stores contents of a system register	long regID; unsigned long __stsr(regID);	long regID; unsigned long __stsr(regID);
ldgr	Loads to a general-purpose register	long a; void __ldgr(regID, a);	None.
stgr	Stores contents of a general-purpose register	unsigned long __stgr(regID);	None.
caxi	Compare and exchange	long *a; long b, c; void __caxi(a, b, c);	long *a; long b, c; long __caxi(a, b, c);
-	Controls interrupt level (Note 1)	int NUM; void __set_il(NUM, "*interrupt-request name*");	int NUM; void* ADDR; void __set_il_rh(NUM, ADDR);

3. Predefined Macros

This section shows a table that compares the CX predefined macros with their corresponding CC-RH predefined macros.

CX Macro Name	CX Definition	CC-RH Macro Name
__LINE__	Line number of source line at that point (decimal).	__LINE__
__FILE__	Name of assumed source file (character string constant).	__FILE__
__DATE__	Date of translating source file	__DATE__
__TIME__	Translation time of source file	__TIME__
__STDC__	Decimal constant 1. (Defined when the -Xansi option is specified)	__STDC__
__v850 __v850__ __v850e2 __v850e2__ __v850e2v3 __v850e2v3__	Decimal constant 1.	__RH850 __RH850__ __v850e3v5 __v850e3v5__
__CX __CX__	Decimal constant 1.	__CCRH __CCRH__
__CHAR_SIGNED__	Decimal constant 1. (Defined when signed is specified by the -Xchar option or when the -Xchar option is not specified).	No value specified.
__CHAR_UNSIGNED__	Decimal constant 1 (Defined when unsigned is specified by the -Xchar option).	None.
__DOUBLE_IS_64BITS__	Decimal constant 1.	No value specified.
__CPUmacro__	Macro indicating the target CPU. Decimal constant 1. A character string indicated by "product type specification" in the device file with "_ _" prefixed and "_" or "_ _"suffixed is defined.	None.
__reg32__	Decimal constant 1. (Defined when the -Xreg_mode=32 option is specified or when the -Xreg_mode option is not specified)	No value specified. (Defined when the -Xreg_mode=32 option is specified)
__reg26__	Decimal constant 1. (Defined when the -Xreg_mode=26 option is specified)	None.
__reg22__	Decimal constant 1. (Defined when the -Xreg_mode=22 option is specified)	No value specified. (Defined when the -Xreg_mode=22 option is specified)
__reg_common__	Decimal constant 1. (Defined when the -Xreg_mode=common option is specified)	No value specified. (Defined when the -Xreg_mode=common option is specified)

_MULTI_CORE__	Decimal constant 1. (Defined when the -Xmulti option is specified)	None.
__MULTI_CMN__	Decimal constant 1. (Defined when the -Xmulti=cmn option is specified)	None.
__MULTI_PEn__	Decimal constant 1. (Defined when -Xmulti=pen option is specified)	None.

4. Extended Language Specifications

This section shows a table that compares the CX extended language specifications with their corresponding CC-RH extended language specifications.

Description	CX	CC-RH
Description with assembly-language instruction (Note 1)	#pragma asm *Assembly-language instruction* #pragma endasm	#pragma inline_asm *function name*[, *function name*]
	__asm("*assembly-language instruction*");	None.
Inline expansion specification	#pragma inline *function name*[, *function name*]	#pragma inline *function name*[, *function name*]
Data memory allocation (Note 2)	#pragma section *section type* ["*section name*"] *Variable declarations and definitions* #pragma section default	#pragma section *attribute strings* ["*section name*"] *Variable declarations and definitions* #pragma section default
Program memory allocation (Note 3)	#pragma text "*section name*" *function name*[,*function name*]…	#pragma section text ["*section name*"] *Variable definitions* #pragma section default
Peripheral I/O register name validation specification (Note 4)	#pragma ioreg	None.
Interrupt/exception handler specification (Note 5)	#pragma interrupt *interrupt-request name function name* [*allocation method*] [*option*]	#pragma interrupt *function name* [*interrupt specification*]
Interrupt disable function specification	#pragma block_interrupt *function name*	#pragma block_interrupt *function name*
Task specification	#pragma rtos_task [*function name*]	None.
Structure type packing specification	#pragma pack ([1|2|4|8])	#pragma pack ([1|2|4])

Note 1: In the CX, this type of extended description is used to embed an assembly-language instruction in a function written in C language. However, the CC-RH assumes that the specified function consists only of assembly-language instructions and inline-expands the assembly-language function declared with #pragma inline_asm at the location where the function is called.

Note 2: The section names differ between the CX and CC-RH. Therefore, the specifiable character strings differ between *section type* and *attribute strings*. For details, refer to the user's manuals for coding.

Note 3: If a section name starting with a number is specified in the CC-RH, "_" is automatically added before the number.

Note 4: In the CC-RH, include the header file for the peripheral I/O registers.

Note 5: The CX automatically allocates at the interrupt handler address an instruction for branching to the specified interrupt function. In the CC-RH, however, the user should define and allocate interrupt and exception vectors. The specifiable character strings differ between *option* and *interrupt specification*.

5. Assembler Directives

This section shows a table that compares the CX assembler directives with their corresponding CC-RH assembler directives. Directives are used to give various directions necessary for the assembler to execute a series of processes.

Description	CX	CC-RH
Section definition directives	.cseg	.cseg (Note 1)
	.dseg	.dseg (Note 1)
	.org	.offset (Note 2)
	.vseg	None.
Symbol definition directive	.set	.set
Data definition and area reservation directives	.db	.db
	.db2/.dhw	.db2/.dhw
	.dshw	.dshw
	.db4/.dw	.db4/.dw
	.db8/.ddw	.db8/.ddw
	.float	.float
	.double	.double
	.ds	.ds
	.align	.align
External definition and external reference directives	.public	.public
	.extern	.extern
	.comm	None.
Macro directives	.macro	.macro
	.local	.local
	.rept	.rept
	.irp	.irp
	.exitm	.exitm
	.exitma	.exitma
	.endm	.endm

Note 1: The relocation attribute that can be specified as the operand differs from that in the CX.

Note 2: .org in the CC-RH is a directive that specifies the start of an absolute-addressing section.

6. Peripheral I/O Registers

This section describes how the CX and CC-RH handle peripheral I/O registers.

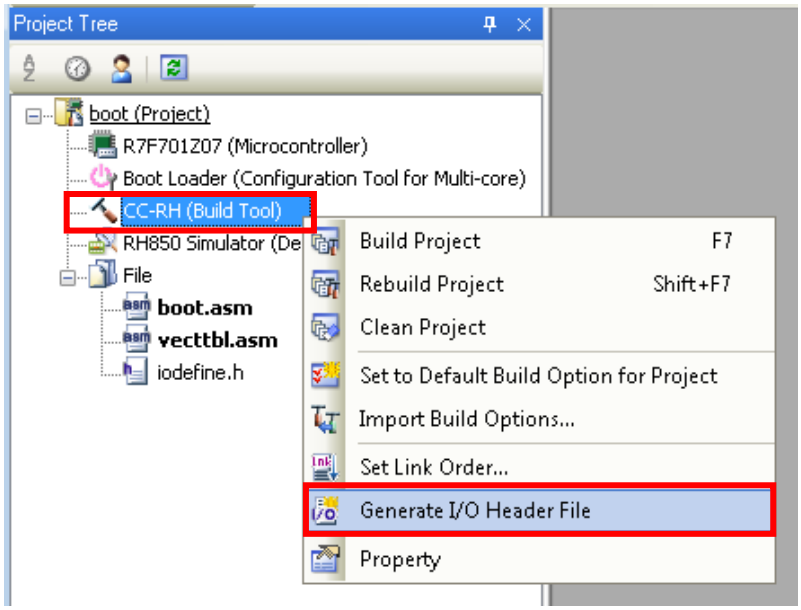
6.1 Peripheral I/O Registers in CX

In the CX, register names can be used to access peripheral I/O registers in C language when the #pragma directive is added. A list of register names and their corresponding addresses is specified in the device file and the register names are translated into their addresses at assembly. Refer to the user's manual for the register names specified in the device file.

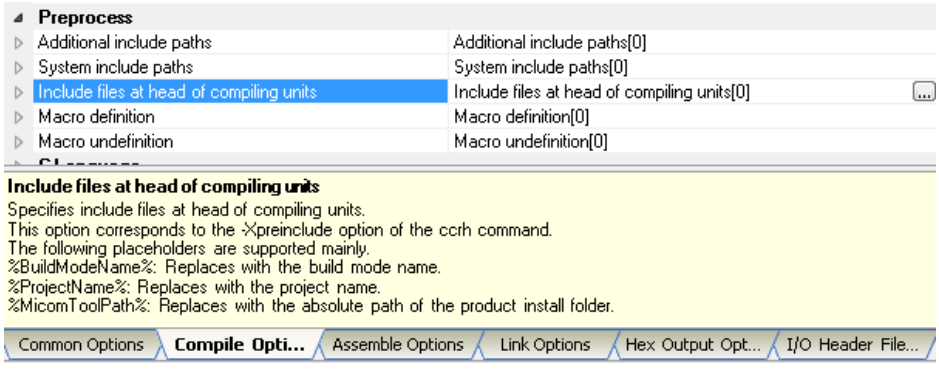
6.2 Peripheral I/O Registers in CC-RH

The CC-RH does not support the use of device files and the user should prepare a file including a list of peripheral I/O register names and their corresponding addresses.

When a new project is created in the CS+, the CS+ generates an I/O header file "iodefine.h" for the target MCU specified in the project and registers it as a source file in the project. The I/O header file defines the names of the registers provided in the MCU and their addresses. The header file can also be generated by right-clicking the [CC-RH (Build Tool)] node in the CS+ project tree and then clicking [Generate I/O Header File].



When accessing a register in a C-language program, include the I/O header file. By specifying the header file as a parameter for the –Xpreinclude option, the #include specification can be omitted from the source file. The –Xpreinclude option can be specified by selecting the [Compile Options] tab => [Preprocess] category => [Include files at head of compiling units]. In this property setting, specify the I/O header file for the target MCU.



7. Interrupts and Exceptions

This section describes the interrupt/exception handlers in the CX and CC-RH.

7.1 Interrupts and Exceptions in CX

When a #pragma interrupt directive is specified, the CX embeds an instruction for branching to the function specified in "*function name*" at the handler address corresponding to the specified "*interrupt-request name*". The CX compiles the function specified in "*function name*" as an interrupt function.

```
#pragma interrupt interrupt-request name function name [allocation method] [option]
```

For "interrupt-request name", specify an interrupt-request name registered in the device file. Refer to the user's manual for the target MCU for the interrupt-request names registered in the device file.

For example, the handler address for the interrupt-request name "INTP0" in the V850E2/FJ4 is 0x110. In this case, the "jr _func" instruction is embedded at address 0x110 according to the #pragma interrupt directive shown below. In addition, the "func" function is compiled as an interrupt function and the register saving and restoring processing is output as an interrupt/exception handler.

```
#pragma interrupt INTP0 func
void func(void) {
  …;
}
```

7.2 Interrupts and Exceptions in CC-RH

When a #pragma interrupt directive is specified, the CC-RH compiles the function specified in "*function name*" according to the specification in "*interrupt specification*".

```
#pragma interrupt function-name [interrupt specification]
```

For example, the "func" function is compiled as an interrupt function according to the #pragma interrupt directive shown below. In addition, the processing for saving and restoring the ctpc, ctpsw, fpepc, and fpsr and the ei and di instructions are output according to the interrupt specifications.

```
#pragma interrupt func (enable=true, callt=true, fpu=true)
void func (unsigned long eiic)
{
  …;
}
```

Note that the user should define and allocate interrupt and exception vectors in the CC-RH. When a new project file is created in the CS+, the "boot.asm" file is registered as a source file and it defines the format for interrupt/exception vectors. Customize the file as necessary and allocate vectors to appropriate addresses in accordance with the target MCU. The following describes the interrupt/exception vectors in "boot.asm".

a. RESET

The following definition embeds the "jr32 __start" instruction at the head of the RESET section.

```
.section "RESET", text
.align   512
jr32     __start ; RESET
```

For example, when a new project is created for the RH850/F1L by the CS+, the "-start" linkage editor option specifies the allocation of the RESET section at address %ResetVectorPE1%. %ResetVectorPE1% is specified in the [Microcontroller] node in the Project Tree Panel => [Microcontroller Information]tab => [Microcontroller Information]category => [Reset vector address].The "jr32 __start" instruction is embedded at address 0x00 by default.

b. Interrupts and exceptions in direct vector method

The base location for handler addresses is obtained by adding the base address indicated by the RBASE or EBASE register and the offset specific to the exception source. Either the RBASE or EBASE register is selected through the PSW.EBV bit. The following definition assumes RBASE as the base address and allocates interrupt/exception handlers immediately after RESET.

```
.section "RESET", text
.align   512
jr32     __start ; RESET

.align   16
jr32     _Dummy ; SYSERR

.align   16
jr32     _Dummy ;

.align   16
jr32     _Dummy ; FETRAP
...
```

In the "boot.asm" file, an instruction for branching to the dummy function "_Dummy" is specified at the offset locations corresponding to SYSERR, FETRAP, etc. The "_Dummy" function is a routine that repeats branches to itself. Customize it as necessary.

Modify "_Dummy" to "*_interrupt-function name*" at the offset locations corresponding to the exceptions and interrupts that should be customized. In addition, define the interrupt functions through the #pragma interrupt directive. The following shows an example for executing the interrupt function "func" when an exception "SYSERR" occurs.

```
        .section "RESET", text
        .align   512
        jr32     __start ; RESET

                                  <-- Modify "_Dummy" to "_interrupt-function name".
        .align   16
        jr32     _func ; SYSERR


        .align   16
        jr32     _Dummy ; HVTRAP
        ...
```

```
#pragma interrupt func (priority=SYSERR, callt=true, fpu=true)
void func (unsigned long feic)
{
  ...;
}
```

c. Interrupts and exceptions in table lookup method

Interrupts can be specified in the table lookup method, which is an extended specification for interrupts. In the direct vector method, only one handler address is assigned to each priority level of EI-level interrupts; for all interrupt channels having the same priority level, execution therefore branches to the same interrupt handler address. However, there will be cases where the application requires a separate code area to be used for each interrupt handler. To implement this, the CC-RH provides the table lookup method.

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

In the "boot.asm" file, an interrupt/exception table for the table lookup method is defined in the EIINTTBL section. When a new project file is created for the RH850/F1L by the CS+, the "-start" linkage editor option specifies allocation of the table immediately after the RESET section.

The addresses where the dummy function "_Dummy_EI" is stored are specified in areas offset from the head of the EIINTTBL section by an address of a multiple of four. Thus, execution branches to _Dummy_EI when an

exception/interrupt at interrupt priority level *n* (*n* is within the range of 0 to 512) in the table lookup method occurs. The "_Dummy_EI" function is a routine that repeats branches to itself. Customize it as necessary.

Modify "#_Dummy_EI" to "#*_interrupt-function name*" at the offset locations corresponding to the channels that should be customized. In addition, when defining interrupt functions in C source file, define the interrupt functions through the #pragma interrupt directive. The following shows an example for executing the interrupt function "func" when a channel-9 interrupt "EIINT9" occurs.

```
        .section "EIINTTBL", const
        .align   512
        .dw      #_Dummy_EI ; INT0
        .dw      #_Dummy_EI ; INT1
        .dw      #_Dummy_EI ; INT2
        .dw      #_Dummy_EI ; INT3
        .dw      #_Dummy_EI ; INT4
        .dw      #_Dummy_EI ; INT5
        .dw      #_Dummy_EI ; INT6
        .dw      #_Dummy_EI ; INT7
        .dw      #_Dummy_EI ; INT8
        .dw      #_func     ; INT9
        .rept    512 - 10
        .dw      #_Dummy_EI ; INTn
```

Modify "#_Dummy_EI" to "*#_interrupt-function name*".

```
#pragma interrupt func (channel=9 enable=true, callt=true, fpu=true)

void func (unsigned long eiic)
{
  …;
}
```

Note that the direct vector method is the default exception/interrupt method in the RH850; to switch to the table lookup method, modify the interrupt control register value.

8. ROMization

The data for variables with initial values should be stored in ROM and then copied to RAM before such variables are accessed after the MCU is reset. This sequence is called ROMization. The ROMization processing differs between the CX and CC-RH. This section describes ROMization processing in the CX and CC-RH.

8.1 ROMization Processing in CX

In the CX, the sections where the variables with initial values are stored (.sdata and .data sections) are the target of ROMization by default. As the initial value data is stored in ROM, the "_rcopy" function should be used to copy the data from ROM to RAM. The destination addresses for copying (.sdata and .data section addresses) should be specified through a link directive file (*.dir). The following shows an example of an _rcopy function call in the startup routine "cstart.asm".

```
        mov32#__S_romp, r6          ;  copy romized data
        mov    -1, r7
        jarl      __rcopy, lp
```

"__S_romp" is a symbol defined in the "rompcrt.obj" file, which stores the ROMization area reservation code. "#__S_romp" is the start address of the initial value data stored in ROM. These values are automatically determined by the linkage editor.

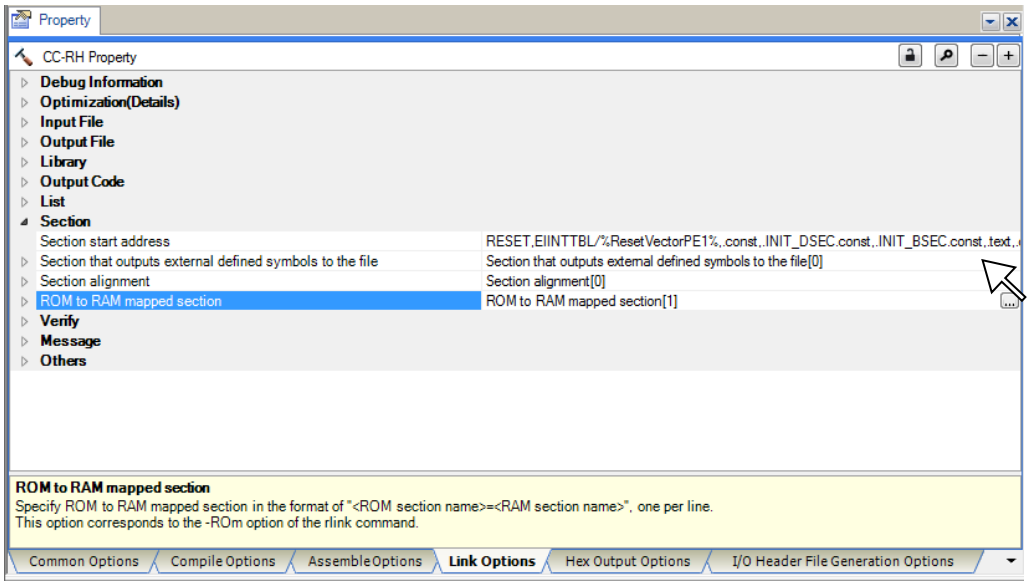
8.2 ROMization Processing in CC-RH

a. Specifying ROMization

In the CC-RH, the target sections for ROMization should be specified through the "–rom" linkage editor option. <*ROM-section name*> is a target section for ROMization. Use the "–start" linkage editor option to allocate the sections specified as <*ROM-section name*> to ROM and those specified as <*RAM-section name*> to RAM.

```
-rom=<ROM-section name>=<RAM-section name>
```

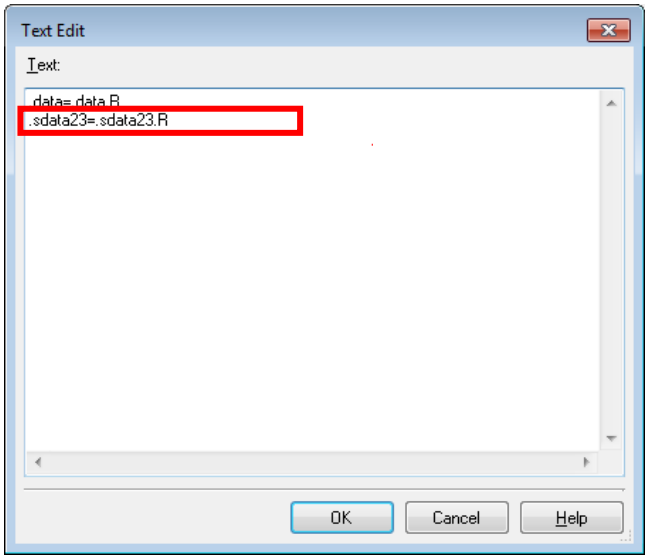
In the CS+, select the [Link Options] tab => [Section] category, click the [...] button at the right end of the [ROM to RAM mapped section] row, and specify the sections to be copied from ROM to RAM in the format of <*ROM-section name*>=<*RAM-section name*> with one section per line.



When a new project is created by the CS+, the following options are specified by default.

```
-rom=.data=.data.R
```

When an additional section other than the .data section is specified to store variables with initial values, use the "–rom" option to add this section to the target sections for ROMization. The following shows an example when the .sdata23 section is added as a target of ROMization (the RAM section name is .sdata.R).



b. Defining the initialization table

In the CC-RH, the ROMized data should be copied from ROM to RAM by using the "_INITSCT_RH" function. When a new project file is created by the CS+, the startup routine "cstart.asm" is registered as a source file and it defines the initialization table to be used to copy the data of variables with initial values as follows.

```
;-----------------------------------------------------------------------
;       section initialize table
;-----------------------------------------------------------------------
        .section ".INIT_DSEC.const", const
        .align  4
        .dw     #__s.data,      #__e.data,      #__s.data.R
```

The initialization table is allocated to the .INIT_DSEC.const section, and a 4-byte area is allocated to each of the .data section start address, .data section end address, and .data.R section start address in that order.

Prefixing a section name with "__s" generates a reserved symbol that has the start address of the section as its value. Likewise, prefixing a section name with "__e" generates a reserved symbol that has the end address of the section as its value. Using these reserved symbols is recommended for additional specifications to the initialization table.

When an additional section is specified to store variables with initial values, add the start and end addresses of the section in this initialization table.

```
;-------------------------------------------------------------------------------
;       section initialize table
;-------------------------------------------------------------------------------
        .section ".INIT_DSEC.const", const
        .align  4
        .dw     #__s.data,      #__e.data,      #__s.data.R
        .dw     #__s.sdata23,   #__e.sdata23,   #__s.sdata23.R
```

In the CC-RH, the "_INITSCT_RH" function can also be used to initialize with zero the sections where variables without initial values are to be stored. The startup routine "cstart.asm" defines the zero-initialization table.

```
        .section ".INIT_BSEC.const", const
        .align  4
        .dw     #__s.bss,       #__e.bss
```

The zero-initialization table is allocated to the .INIT_BSEC.const section, and a 4-byte area is allocated to each of the .bss section start address and .bss section end address in that order. When an additional section other than the .bss section is specified to store variables without initial values, add the addresses of the section in the same format as the existing settings.

c. Calling the copy function

The "_INITSCT_RH" function is called from the startup routine "cstartm.asm". This processing initializes the sections defined in each table.

```
        mov     #__s.INIT_DSEC.const, r6
        mov     #__e.INIT_DSEC.const, r7
        mov     #__s.INIT_BSEC.const, r8
        mov     #__e.INIT_BSEC.const, r9
        jarl32  __INITSCT_RH, lp                ; initialize RAM area
```

9. Section Allocation

This section describes the section allocation processing in the CX and CC-RH.

9.1 Section Allocation in CX

In the CX, section allocation addresses should be specified in the link directive file (*.dir) and this file should be input to the linkage editor by specifying it through the -Xlink_directive option. The following shows the format for specifying section allocation addresses in the link directive file for the CX.

```
Segment name: !segment type  ?segment attribute  Vaddress {

        Output-section name=$section type  ?section attribute  input-section name;
        Output-section name=$section type  ?section attribute  input-section name;
        …
} ;
```

In the following specifications, the .const section allocation begins from address 0x1000. Allocation of the .pro_epi_runtime and .text sections begins after the end of the .const section and proceeds toward higher addresses in that order. Allocation of the .data, .sdata, .sbss, and .bss sections begins from address 0xfedf6000 and proceeds toward higher addresses in that order.

```
CONST:!LOAD ?R V0x1000 {
        .const = $PROGBITS ?A .const ;
};

TEXT:!LOAD ?RX {
        .pro_epi_runtime = $PROGBITS ?AX .pro_epi_runtime ;
        .text = $PROGBITS ?AX .text ;
};

DATA:!LOAD ?RW V0xfedf6000 {
        .data = $PROGBITS ?AW .data ;
        .sdata = $PROGBITS ?AWG .sdata ;
        .sbss = $NOBITS ?AWG .sbss ;
        .bss = $NOBITS ?AW .bss ;
};
```

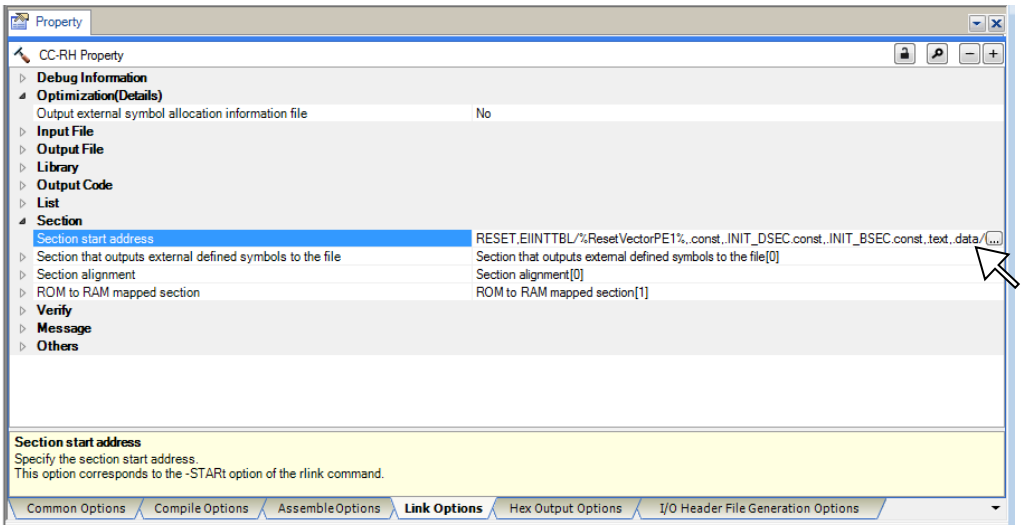
9.2 ROMization Processing in CC-RH

In the CC-RH, section allocation addresses should be specified through the "-start" linkage editor option; the link directive file in the CX or the like is not used. The following shows an example for specifying the "–start" option in the CC-RH. For details, refer to the user's manual for the build process.

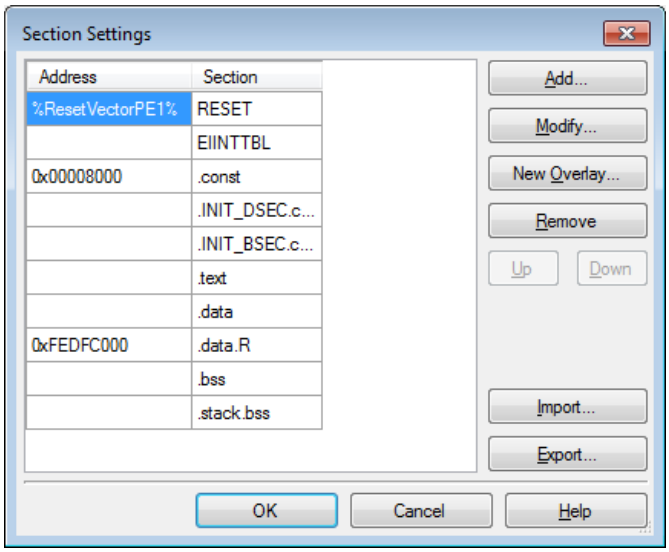
```
-start=RESET,EIINTTBL /%ResetVectorPE1%,.const,.INIT_DSEC.const,.INIT_BSEC.const,.text,
.data/00008000,.data.R,.bss,.stack.bss/FEDEFC00
```

Through the above option settings, allocation of the RESET section begins from address %ResetVectorPE1%. Allocation of the EIINTTBL, .const, .INIT_DSEC.const, .INIT_BSEC.const, .text, and .data sections begins after the end of the RESET section and proceeds toward higher addresses in that order. Allocation of the .data.R, .bss, and .stack.bss sections begins from address 0xFEDE0000 and proceeds toward higher addresses in that order.

In the CS+, section allocation can be specified through the GUI; select the [Link Options] tab => [Section] category, and click the [...] button at the right end of the [Section start address] row.



The Section Settings dialog box will open; addresses and sections can be added and modified through manipulation in this dialog box.



10. Program Compatibility

This section shows an example of program description that is successfully compiled in the CX but generates an error in the CC-RH, and also shows a workaround to avoid this error.

1. Binary notation of constants

[Example]

```
int a = 0b00000001;
```

The extended binary notation supported in the CX is not allowed in the CC-RH. Modify the notation to hexadecimal.

[Workaround]

```
int a = 0x01;
```

Website and Support <website and support,ws>

Renesas Electronics Website
http://www.renesas.com/

Inquiries
http://www.renesas.com/contact/

Revision History

		Description	
Rev.	**Date**	**Page**	**Summary**
1.00	2014.10.17	-	Publication of the first edition
1.01	2017.04.21	-	Change CC-RH compiler version to V1.05.00

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Handling of Unused Pins

   Handle unused pins in accordance with the directions given under Handling of Unused Pins in the manual.

   — The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on

   The state of the product is undefined at the moment when power is supplied.

   — The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.
   In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.
   In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

   — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals

   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.

   — When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.

5. Differences between Products

   Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.

   — The characteristics of Microprocessing unit or Microcontroller unit products in the same group but having a different part number may differ in terms of the internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

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 disputes 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, drawing, chart, program, algorithm, 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 otherwise misappropriate 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, copy or otherwise misappropriation of Renesas Electronics products.
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; and 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.

   Renesas Electronics products are neither intended nor 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 damages (space and 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 third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.
6. When using the 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 radiation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions or failure or accident arising out of the use of Renesas Electronics products beyond 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. Further, Renesas Electronics products are not subject to radiation resistance design. Please ensure to implement safety measures to guard them against the possibility of bodily injury, injury or damage caused by fire, and social damage in the event of 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 by your own responsibility as warranty for your products/system. Because the evaluation of microcomputer software alone is very difficult and not practical, please evaluate 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. Please investigate applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive carefully and sufficiently and use 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 not use Renesas Electronics products or technologies for (1) any purpose relating to the development, design, manufacture, use, stockpiling, etc., of weapons of mass destruction, such as nuclear weapons, chemical weapons, or biological weapons, or missiles (including unmanned aerial vehicles (UAVs)) for delivering such weapons, (2) any purpose relating to the development, design, manufacture, or use of conventional weapons, or (3) any other purpose of disturbing international peace and security, and you shall not sell, export, lease, transfer, or release Renesas Electronics products or technologies to any third party whether directly or indirectly with knowledge or reason to know that the third party or any other party will engage in the activities described above. When exporting, selling, transferring, etc., Renesas Electronics products or technologies, you shall comply with any applicable export control laws and regulations promulgated and administered by the governments of the countries asserting jurisdiction over the parties or transactions.
10. Please acknowledge and agree that you shall bear all the losses and damages which are incurred from the misuse or violation of the terms and conditions described in this document, including this notice, and hold Renesas Electronics harmless, if such misuse or violation results from your resale or making Renesas Electronics products available any third party.
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.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

(Rev.3.0-1 November 2016)

RENESAS

Renesas Electronics Corporation

http://www.renesas.com

SALES OFFICES

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100191, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, P. R. China 200333
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HAL II Stage, Indiranagar, Bangalore, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Gu, Seoul, 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 6.0