CS+ RH850 Compiler CC-RH V1.07.00 Release Note

R20UT4208EJ0100
Rev.1.00
December 20, 2017

Thank you for using the CS+ integrated development environment.
This document describes the restrictions and points for caution. Read this document before using the product.

Contents

Chapter 1. Target Devices

The target devices supported by the CC-RH compiler are listed on the Website.

Please see the URL below.

CS+ Product Page:

http://www.renesas.com/cs+

Chapter 2. User’s Manuals

Please read the following user’s manuals along with this document.

Manual Name	Document Number
CC-RH Compiler User's Manual	R20UT3516EJ0104
CS+ Integrated Development Environment User’s Manual: CC-RH Build Tool Operation	R20UT3283EJ0105

Chapter 3. Keywords When Uninstalling the Product

There are two ways to uninstall this product.

- Use the integrated uninstaller from Renesas (uninstalls all CS+ components)
- Use the Windows uninstaller (only uninstalls this product)

To use the Windows uninstaller, select “CS+ CC-RH V1.07.00” from “Programs and Features” of the control panel.

Chapter 4. Changes

This chapter describes changes to the CC-RH compiler from V1.06.00 to V1.07.00.

Note that the features which are only available to users holding a registered license for the Professional edition are indicated as [Professional edition].

4.1 C99 standard

To select conformance of the language specifications with the C99 standard, the **-lang** and **-strict_std** compiler options have been added.

Note that this version of the compiler does not support variable-length arrays and complex types, and some standard library functions.

```
-lang={c|c99}
```

When the -lang=*c* option is specified, the language specifications conform with the C90 standard.
When the -lang=*c99* option is specified, the language specifications conform with the C99 standard.

```
-strict_std
```

This option selects processing of the C source program in strict accordance with the language standard (C90 or C99) specified with the -lang option. Error and warning messages are output in response to code that violates the given standard.

V1.06.00 and earlier versions have the -Xansi option to select the processing of C source programs in strict accordance with the C90 standard; however, in V1.07.00 and later versions, the -strict_std option is used. If the -Xansi option is specified in V1.07.00 and later versions, it is automatically converted to the -strict_std option for input to the compiler.

4.2 Improvements to the feature for checking source code against MISRA-C:2012 rules [Professional edition]

The following rule numbers have been added to those which can be designated as arguments of the -Xmisra2012 option for use with the C99 standard, which selects checking by the compiler of source code against the specified MISRA-C:2012 rules.

[Mandatory rules]	**17.6**
[Required rules]	**8.14, 9.4, 9.5, 13.1, 18.7, 21.11**
[Advisory rules]	**21.12**

The following are the numbers of MISRA-C:2012 rules against which each revision of compilers can check source code for compliance.

Rule classification (number of rules in the standard)	*V1.03.00*	*V1.04.00*	*V1.05.00*	*V1.06.00*	***V1.07.00***
Mandatory rules (16)	3	3	4	6	**7**
Required rules (108)	31	58	76	80	**86**
Advisory rules (32)	7	21	23	25	**26**
Total number of rules (156)	41	82	103	111	**119**

4.3 Feature for detecting illegal indirect function calls [Professional edition]

A feature for detecting indirect function calls to illegal addresses has been added.

The compiler checks the branch destination addresses of indirect calls of functions through the following steps and calls an error function if it detects a problem.

1. The compiler automatically extracts functions which may be indirectly called in the C-source program and the linker consolidates the information to generate a list of functions in the executable files.

2. The compiler inserts processing for calling the "__control_flow_integrity" checking function immediately before calls of indirect functions it finds in analyzing the C-source program. The branch destination address of the call of the indirect function is passed as an argument to this checking function.

3. Within the checking function at the time of execution, the branch destination address given as the argument is checked to see if it is included in the list of functions. If the address is not included, it is regarded as an illegal indirect function call, so the "__control_flow_chk_fail" error function will be called.

The following C-source program shows an example of an illegal indirect function call.

```
1:  extern void func1(void);
2:  extern void func2(void);
3:
4:  void (*fp)(void) = &func1;
5:
6:  void main(void) {
7:      (*fp)();            // Function func1 is indirectly called.
8:      func2();            // Function func2 is directly called.
9:  }
```

Since the address of func1 is acquired in the fourth line, the call of func1 is regarded as indirect and added to the list of functions.

Since a function pointer fp is used to indirectly call func1 in the seventh line, the compiler acquires the value of fp immediately before this call and generates code to call the "__control_flow_integrity" checking function by specifying the acquired value as an argument. Within the checking function, a check of whether the value specified by the argument (the address of func1 in the case of normal operation) is included in the list of functions is conducted and subsequent operation is as follows.

- If the list includes the value, the compiler continues to process the C-source program.
- If the list does not include the value, the "__control_flow_chk_fail" error function is called.

Illegal indirect function calls can thus be detected in the way described above.
Since the call of func2 is direct, the eighth line is not detected.

Specify the following options to enable this feature.

[Compile option]

```
-control_flow_integrity
```

This option selects the generation of code for detecting illegal indirect function calls.

[Link option]

```
-cfi
```

This option selects the generation of a list of functions for use in detecting illegal indirect function calls.

The following linker options have also been added in association with this feature.

- **-cfi_add_func**
  This option adds the symbols or addresses of functions which are specified as arguments to the list of functions.
- **-cfi_ignore_module**
  This option selects the non-addition of the addresses of functions included in a file which is specified as an argument to the list of functions.
- **-show=*cfi***
  This option selects the output of the contents of the list of functions to the list file which is output in response to specifying the -list option.

4.4 PIC/PID facilities

PIC and PID (described below) facilities have been added for the allocation of functions, constants, or variables to given addresses that differ from those they would otherwise be given at the time of linkage. In the CC-RH compiler,

- the facility for allocating functions to addresses and making them executable there is called the **PIC** (**P**osition **I**ndependent **C**ode) facility,
- the facility for allocating constants to addresses and making reference to them at those addresses possible is called the **PIROD** (**P**osition **I**ndependent **R**ead **O**nly **D**ata) facility, and
- the facility for allocating variables to addresses and making reference to them at those addresses possible is called the **PID** (**P**osition **I**ndependent **D**ata) facility.

4.4.1 Section

The following sections have been added to support the PIC and PID facilities.

Target	*Section Name*	*Access*
Function	**.pctext**	Access with a 32-bit address relative to the PC value or the __pc_data symbol
Constant	**.pcconst16**	Access with a 16-bit address relative to the __pc_data symbol
Constant	**.pcconst23**	Access with a 23-bit address relative to the __pc_data symbol
Constant	**.pcconst32**	Access with a 32-bit address relative to the __pc_data symbol
Variable with the initial value	**.sdata32**	Access with a 32-bit address relative to the value of r4 (GP)
Variable without the initial value	**.sbss32**	
Variable with the initial value	**.edata32**	Access with a 32-bit address relative to the value of r30 (EP)
Variable without the initial value	**.ebss32**	

In association with the addition of the sections above, the following options have been added.

- Attribute strings which can be specified for #pragma section:
  pctext, pcconst16, pcconst23, pcconst32, gp_disp32, ep_disp32
- Section attributes which can be specified as arguments of the -Xsection option:
  pcconst16, pcconst23
- Relocatable attributes which can be specified as operands of the .cseg, .dseg or .section directive.
  PCTEXT, PCCONST16, PCCONST23, PCCONST32,
  SDATA32, SBSS32, EDATA32, EBSS32

4.4.2 Compile options

The following compiler options have been added for enabling the PIC and PID facilities.

[To enable the PIC facility]

```
-pic
```

The default section of the functions is changed from .text to .pctext.

Execution at the given position is enabled by calling the functions that were allocated to the .pctext section in PC-relative mode.

[To enable the PIROD facility]

```
-pirod
```

The default section of the constants is changed from .const to .pcconst32.

Allocation and reference to the given positions are enabled by referring to the constants that were allocated to the .pcconst32 section in PC-relative mode.

[To enable the PID facility]

```
-pid
```

The default sections of the variables are changed from .data and .bss to .sdata32 and .sbss32.

Allocation and reference to the given positions are enabled by referring to the variables that were allocated to the .sdata32 and .sbss32 sections in GP-relative mode.

4.4.3 Assemble options

The following assembler options have been added to enable the PIC and PID facilities.

[To enable the PIC facility]

```
-pic
```

The relocation attribute that can be specified as an operand in the .cseg or .section directive is changed as follows. If a relocation attribute that is not specifiable is specified, an error will occur.

- When this option is not specified: TEXT
- When this option is specified: PCTEXT

[To enable the PIROD facility]

```
-pirod
```

The relocation attribute that can be specified as an operand in the .cseg or .section directive is changed as follows. If a relocation attribute that is not specifiable is specified, an error will occur.

- When this option is not specified: CONST, ZCONST, or ZCONST23
- When this option is specified: PCCONST16, PCCONST23, or PCCONST32

[To enable the PID facility]

```
-pid
```

The relocation attribute that can be specified as an operand in the .cseg or .section directive is changed as follows. If a relocation attribute that is not specifiable is specified, an error will occur.

- When this option is not specified: DATA, ZDATA, ZDATA23, BSS, ZBSS, or ZBSS23
- When this option is specified: SDATA32, SBSS32, EDATA32, or EBSS32

4.4.4 -r4 option

The **-r4** option, which causes the code generated during compilation to not use the r4 register (GP), has been added.

```
-r4={fix|none}
```

When the -r4=*fix* option is specified, the value of the r4 register is fixed for the entire project. Specify this parameter when GP-relative sections are in use due to the PID facility or for some other reason.

When the -r4=*none* option is specified, object code that can be used for both programs that include or do not include the use of PID is generated.

4.5 Enhanced optimization

The performance of generated code have been improved mainly by implementing the following optimization.

4.5.1 Handling of bitwise operations

Handling of bitwise operations for data having a narrow bit width has been enhanced.

<Example of source code>

```
void func(unsigned char *t, unsigned char i, unsigned char j, unsigned char v) {
        unsigned char *p = &t[i & 0xff];
        unsigned char m = j >> (i & 0xf);
        *p = v ? m : ~m;
}
```

In this example, a bit-mask operation for the value to be assigned to *p is deleted. Replacing bitwise operations having unnecessary processing in conditional operations with the logical instruction *not* becomes possible.

<Code generated by V1.06.00>

```
_func:
        .stack _func = 0
        andi 0x0000000F, r7, r2
        shr r2, r8
        andi 0x000000FF, r8, r2 ; Bit-mask operation
        add r7, r6
        cmp 0x00000000, r9
        bnz9 .BB.LABEL.1_2
.BB.LABEL.1_1:   ; bb20
        movea 0xFFFFFF00, r0, r2
        or r2, r8
        xori 0x000000FF, r8, r2
.BB.LABEL.1_2:   ; bb23
        st.b r2, 0x00000000[r6]
        jmp [r31]
```

<Code generated by V1.07.00>

```
_func:
        .stack _func = 0
        andi 0x0000000F, r7, r2
        shr r2, r8
        add r7, r6
        cmp 0x00000000, r9
        bnz9 .BB.LABEL.1_2
.BB.LABEL.1_1:   ; bb20
        not r8, r8    ; Logical instruction
.BB.LABEL.1_2:   ; bb23
        st.b r8, 0x00000000[r6]
        jmp [r31]
```

4.5.2 Alias analysis

Optimization by alias analysis has been enhanced. Alias analysis was implemented in V1.06.00 and is enabled by specifying the -Xalias=ansi option.

In V1.06.00, alias analysis is disabled when the -Xmerge_files option is specified. However, in V1.07.00, it is enabled even when the -Xmerge_files option is specified.

When optimization by alias analysis is enabled, the effect is the same as in V1.06.00.

<Example of source code>

```
struct tag1 {
      char member1;
      int   member2;
      long long member3;
} StructArray[2];

struct tag2 {
      short index0;
      short index1;
      short index2;
};

void func(struct tag2 *p) {
      StructArray[p->index1].member1 = 1;
      StructArray[p->index1].member2 = 2;
      StructArray[p->index1].member3 = 3;
}
```

In the example above, although the address of StructArray[p->index1] would be calculated three times without alias analysis, it is only calculated once with alias analysis.

<Without alias analysis>

```
_func:

        .stack _func = 0
        ld.h 0x00000002[r6], r2
        shl 0x00000004, r2
        mov #_StructArray, r5
        add r5, r2
        mov 0x00000001, r7
        st.b r7, 0x00000000[r2]
        ld.h 0x00000002[r6], r2
        shl 0x00000004, r2
        add r5, r2
        mov 0x00000002, r7
        st.w r7, 0x00000004[r2]
        ld.h 0x00000002[r6], r2
        shl 0x00000004, r2
        add r2, r5
        mov 0x00000003, r2
        st.w r2, 0x00000008[r5]
        st.w r0, 0x0000000C[r5]
        jmp [r31]
```

<With alias analysis>

```
_func:

        .stack _func = 0
        ld.h 0x00000002[r6], r2
        shl 0x00000004, r2
        mov #_StructArray, r5
        add r2, r5
        mov 0x00000001, r2
        st.b r2, 0x00000000[r5]
        mov 0x00000002, r2
        st.w r2, 0x00000004[r5]
        mov 0x00000003, r2
        st.w r2, 0x00000008[r5]
        st.w r0, 0x0000000C[r5]
        jmp [r31]
```

4.6 Upper limits on usable amounts of memory

The amounts of memory on the host computer that are usable by the CC-RH compiler have been expanded.

- 2 Gbytes with the 32-bit and 64-bit OSs [V1.06.00 and earlier versions]
- **3 Gbytes** with the 32-bit OS and **4 Gbytes** with the 64-bit OS [V1.07.00 and later versions]

4.7 Initialization of automatic variables by using immediate values

The **-Oinline_init** compiler option, which causes the initialization of structure-type or array-type automatic variables by using immediate values, has been added. This may accelerate the execution of programs.

<Example of source code>

```
void main() {
  int array[4] = {1,2,3,4};
}
```

Initialization of automatic variables is usually by reference to a table in which initializers have been defined (e.g. .STR.1 for code generated by V1.06.00). When the -Oinline_init option is specified, automatic variables are initialized by placing initializers in instructions.

<Code generated by V1.06.00>

```
_main:
        .stack _main = 16
        add 0xFFFFFFF0, r3
        movea 0x00000010, r0, r2
        mov #.STR.1, r5
        mov r3, r6
        add r6, r2
        br9 .BB.LABEL.1_2
.BB.LABEL.1_1:   ; entry
        ld23.dw 0x00000000[r5], r8
        st23.dw r8, 0x00000000[r6]
        add 0x00000008, r5
        add 0x00000008, r6
.BB.LABEL.1_2:   ; entry
        cmp r6, r2
        bnz9 .BB.LABEL.1_1
.BB.LABEL.1_3:   ; entry
        dispose 0x00000010, 0x00000000, [r31]
        .section .const, const
        .align 4
.STR.1:
        .dw 0x0001,0x0002,0x0003,0x0004
```

<Code generated by V1.07.00>
[When -Oinline_init is specified]

```
_main:
        .stack _main = 16
        add 0xFFFFFFF0, r3
        mov 0x00000001, r2
        st.w r2, 0x00000000[r3]
        mov 0x00000002, r2
        st.w r2, 0x00000004[r3]
        mov 0x00000003, r2
        st.w r2, 0x00000008[r3]
        mov 0x00000004, r2
        st.w r2, 0x0000000C[r3]
        dispose 0x000010, 0x000000, [r31]
```

4.8 Control of messages

The **-change_message** compiler option, which is used to change warning messages to error messages, has been added to avoid oversights in the form of warning messages not being noticed.

In addition, the -Xno_warning compiler option that controls the output of warning messages can now specify messages with numbers from 0510000.

- W0520000 to W0559999 can be controlled. [V1.06.00 and earlier versions]
- **W0510000 to W0559999** can be controlled. [V1.07.00 and later versions]

4.9 Fixing of the record length of the Intel HEX file

The **-fix_record_length_and_align** option, which causes the output addresses of Intel HEX files (.hex) and Motorola S-record files (.mot) to have a specified alignment and be output with a fixed record length, has been added. Since with this option a HEX file is always output with a fixed record length, it can improve the efficiency of work such as comparing HEX files.

The -byte_count option has also been extended to allow its specification along with the -form=stype option.

4.10 Addition of a message at linkage

In V1.06.00 and earlier versions, the warning code W0561322 was output if sections with different alignment conditions but the same names were linked. In V1.07.00, warning code **W0561331** is output when sections with the same names but different alignment conditions, with the condition for one not being a multiple of that of the other, are linked.

W0561322: Section alignment mismatch : "*section*"

W0561331: Section alignment is not adjusted : "*section*"

In both cases, specification of the greater value of the alignment condition is enabled and the sections are linked.

W0561322 can be ignored since it does not indicate a problem with operation. However, since W0561331 indicates a possible problem with operation, the alignment conditions must be reviewed.

4.11 Rectified points for caution

The points for caution on the following four items no longer apply. For details, refer to Tool News.

- The pow function returning incorrect values (No.9)
- Using a goto statement to move to a label in a switch statement (No.16)
- Math library functions that contain FPU instructions (No.17)
- Loop statements with loop-control variables in which constants are used as the condition for ending the loop (No.18)

4.12 Other changes and improvements

The generation of an internal error in response to building has been corrected.

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

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