RX Development Environment Migration Guide

REJ06J0077-0200
Rev.2.00
Nov. 30, 2016

Migration from H8C Family to RX Family (Compiler ed.)
(High-performance Embedded Workshop and H8C to CS+ and CC-RX)

Introduction

In this application notes, it explains the software migration method when C program made by H8SX, H8S, H8 family compiler (It is recorded as H8-family) is transplanted to RX-family compiler.

In Renesas RX-family compiler, the function to absorb the difference between the option and the language specification is supported inconsideration of the migration from H8-family to RX-family. As a result, the application part of the embedded software can be smoothly transplanted.

Please use this application notes when you transplanted to RX-family from H8-family.

Contents

1. Options

Some specifications differ for the default options between H8-family compilers and RX-family compilers. The following explains options that will likely require handling during migration from H8 to RX.

Table 1-1 List of options

No	Functionality	H8 option	RX option	Reference
1	Specifying sign for the char type	-	signed_char	1.1
2	Specifying sign for bit-field members	-	signed_bitfield	1.2
3	Specifying bit-field member allocation	bit_order	bit_order	1.3
4	Specifying endian	-	endian	1.4
5	Specifying the size of double type	dubole=float	dbl_size	1.5
6	Correspondence of int type size to difference	-	int_to_short	1.6

1.1 Specifying sign for the char type

H8-family compilers treat char types with no sign specified as signed char types, whereas RX-family compilers treat them as unsigned char types in default. To migrate to RX a program created in H8 based on the requirement that char types be signed char types, specify the “signed_char” option.

Format

signed_char : unsigned_char by default

unsigned_char

[How to specify this option in CS+]

Perform the following settings in the [Common Options] page of CC-RX (build tool) properties.

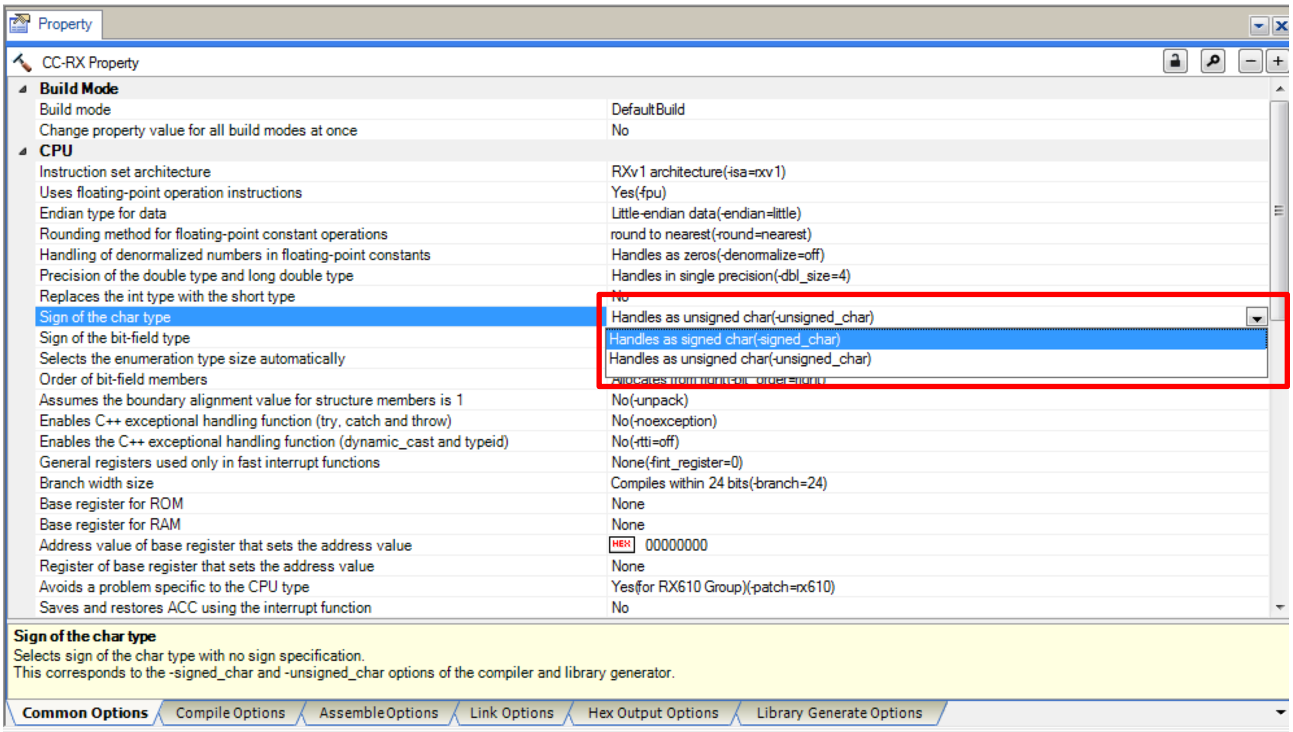

Figure 1-1

1.2 Specifying sign for bit-field members

H8-family compilers treat bit-field members with no sign specified as signed types, whereas RX-family compilers treat them as unsigned types in default.

To migrate to RX a program created in H8 based on the requirement that bit-field members with no sign specification are signed types, specify the “signed_bitfield” option.

Format

signed_bitfield : unsigned_bitfield by default

unsigned_bitfield

[How to specify this option in CS+]

Perform the following settings in the [Common Options] page of CC-RX (build tool) properties.

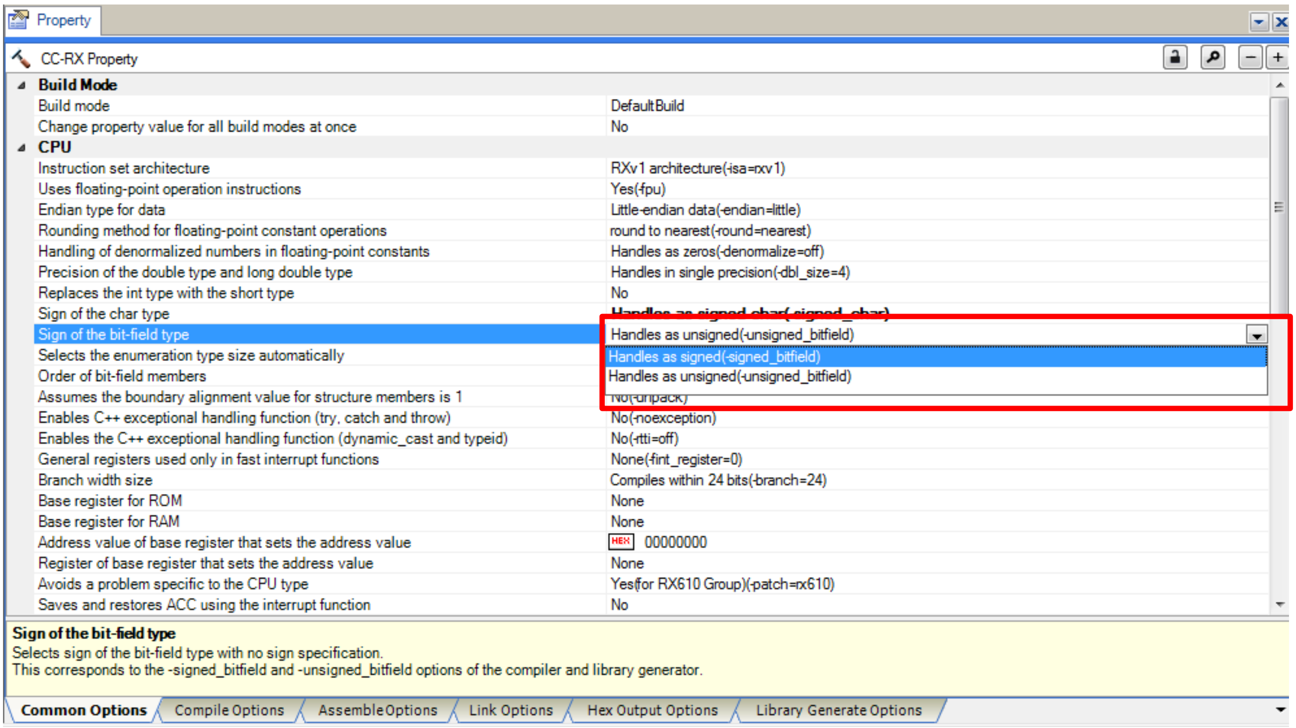

Figure 1-2

1.3 Specifying bit-field member allocation

With H8-family compilers, bit-field members are allocated from the most significant bit, whereas with RX-family compilers, they are allocated from the least significant bit in default. To migrate to RX a program created in H8 based on the requirement that bit-field members are allocated from the most significant bit, specify the “bit_order=left” option.

Format

bit_order={left|right} : right by default

[How to specify this option in CS+]

Perform the following settings in the [Common Options] page of CC-RX (build tool) properties.

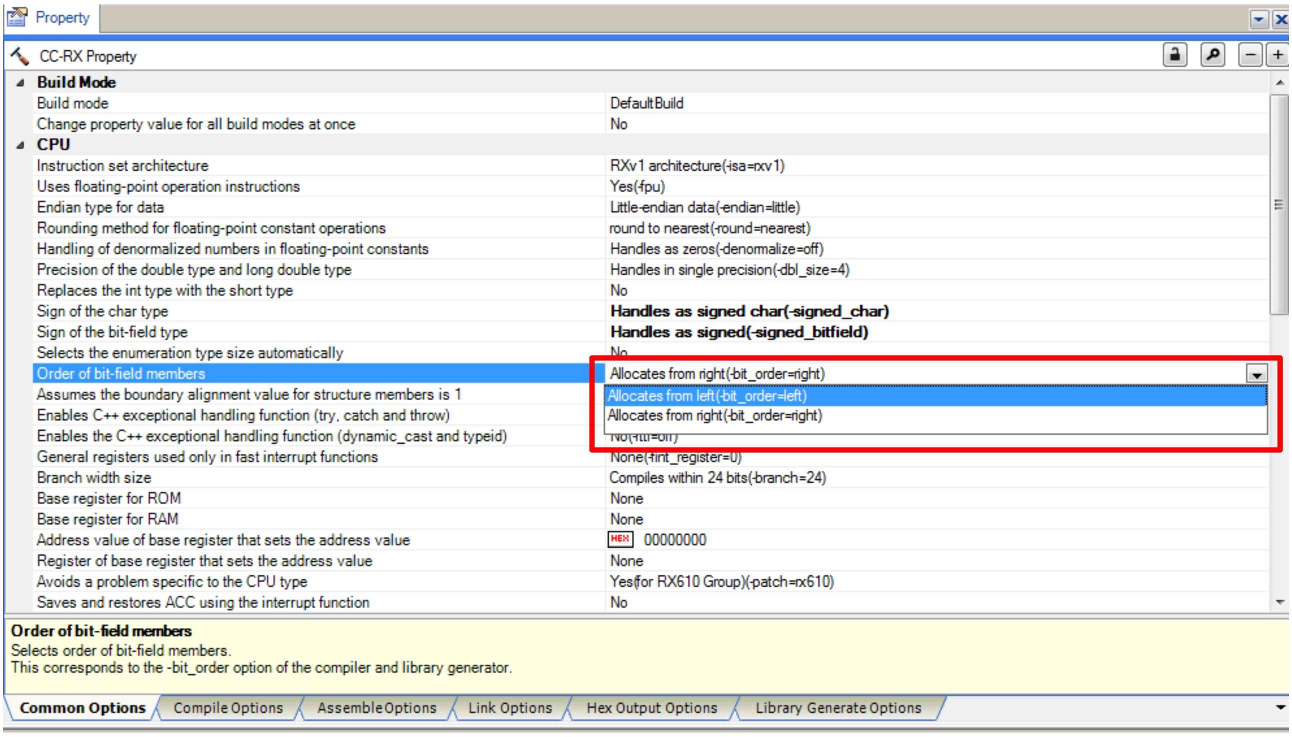

Figure 1-3

1.4 Specifying endian

With H8-family compilers, the data byte order is big-endian, whereas with RX-family compilers, it is little-endian in default.

To migrate to RX a program created in H8 based on the requirement that the data byte order is big-endian, specify the “endian=big” option.

Format

endian={big|little} : little by default

[How to specify this option in CS+]

Perform the following settings in the [Common Options] page of CC-RX (build tool) properties.

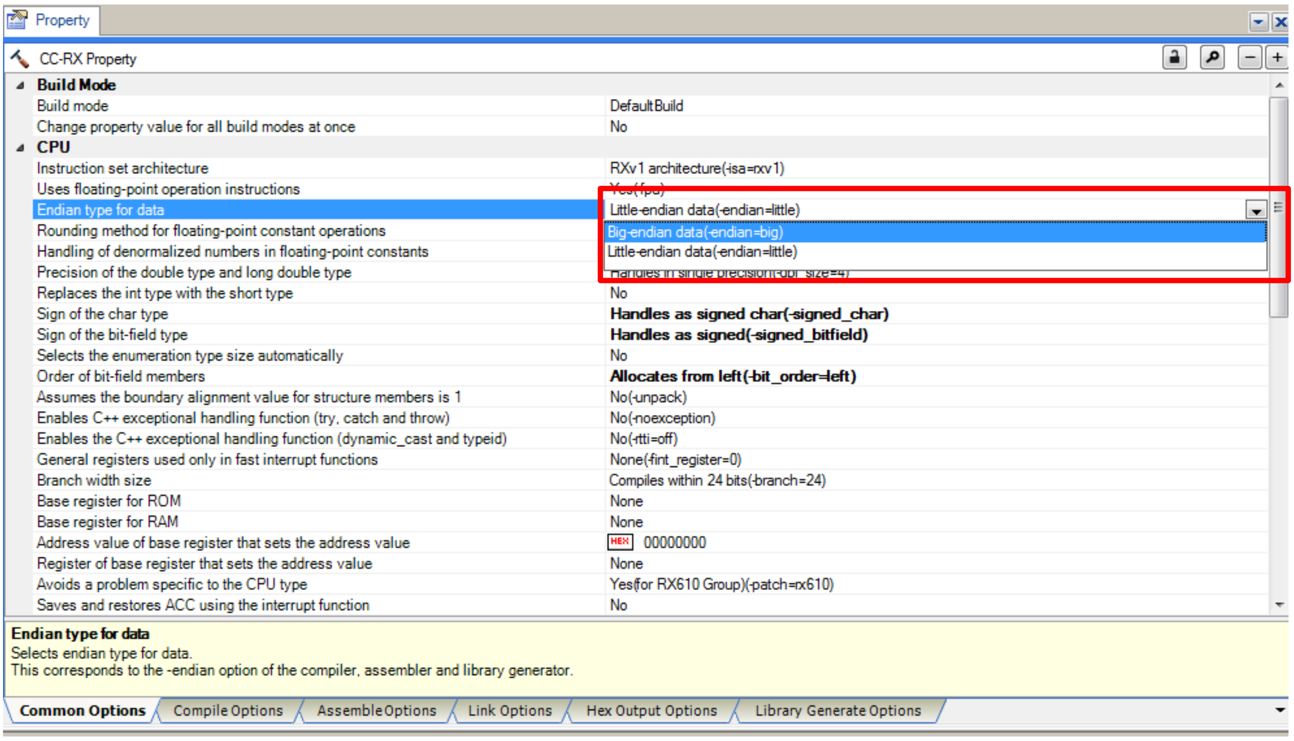

Figure 1-4

1.5 Specifying the size of double type

With H8-family compilers, the size of the double type is 8 bytes, whereas with RX-family compilers, the size of the double type is four bytes in default. To migrate to RX a program created in H8 based on the requirement that the size of the double type is 8 bytes, specify the “dbl_size=8” option.

Format

dbl_size ={4|8} : 4 by default

[How to specify this option in CS+]

Perform the following settings in the [Common Options] page of CC-RX (build tool) properties.

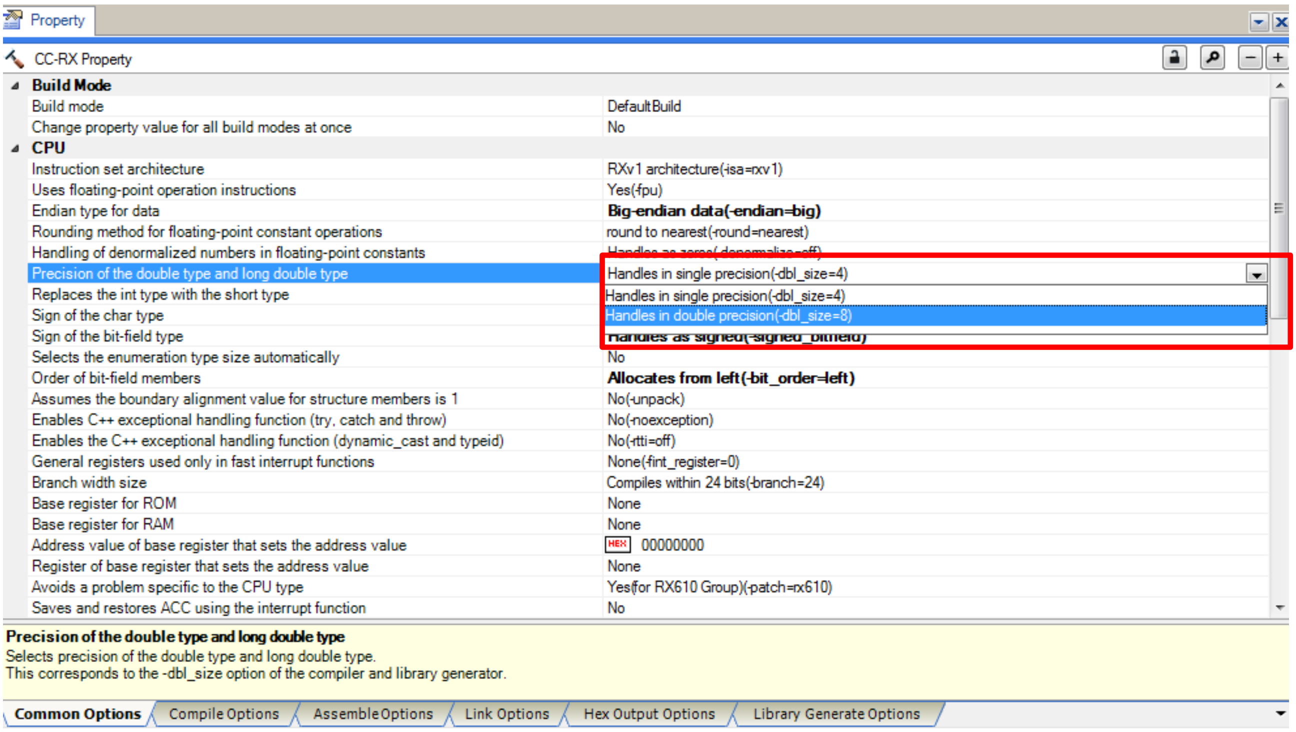

Figure 1-5

Precaution:

In optional double=float of the compiler for the H8 family, the size of the float type and the double type is four bytes. And the long long type is eight bytes. In optional dbl_size=4 of the compiler for the RX family, it is four bytes as for the size of the float type, the double type, and the long double type.

The result of conversion/library related to the floating point might be different from the result of the compiler for the H8 family when optional dbl_size=4 is effective.

1.6 Correspondence of int type size to difference

With H8-family compilers, the size of the int type is 2 bytes, whereas with RX-family compilers, the size of the int type is 4 bytes in default. To migrate to RX a program created in H8 based on the requirement that the size of the int type is 2 bytes, specify the “int_to_short” option.

Format

int_to_short

[How to specify this option in CS+]

Perform the following settings in the [Common Options] page of CC-RX (build tool) properties.

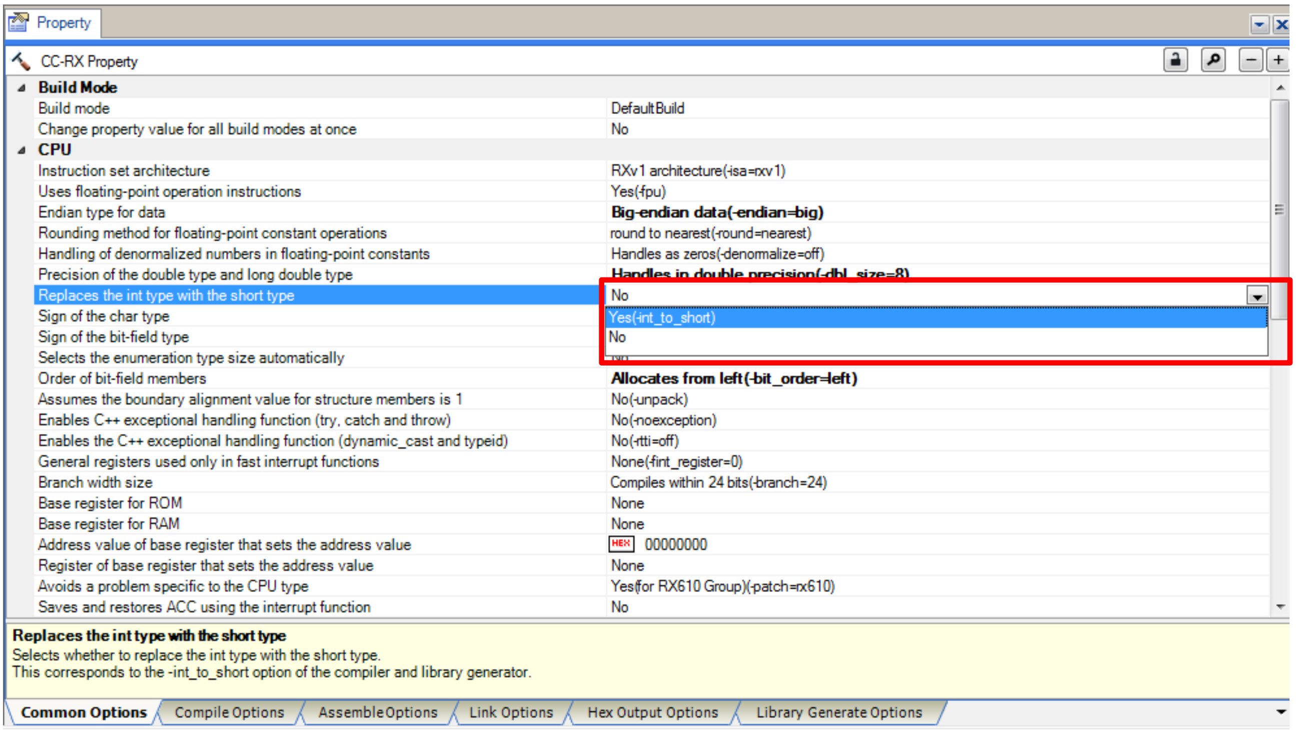

Figure 1-6

Precaution:

Before compilation, the int type is replaced with the short type in the source file, but the value of INT_MAX, INT_MIN, and UINT_MAX in limits.h are not converted.

This option cannot be specified by combining with lang=cpp or lang=ecpp. Optional int_to_short becomes invalid when combining.

2. Language specification

This chapter explains the language specifications that need to be changed during RX migration.

Table 2-1 List of language specifications

No	Functionality	Reference
1	Signs for char types	2.1
2	Specifying sign for bit-field members	2.2
3	Endian	2.3
4	Size of double type	2.4
5	Size of int type	2.5
6	asm blocks	2.6

2.1 Signs for char types

With H8-family compilers, char types without a specified sign are treated as signed char types, whereas with RX-family compilers, they are treated as unsigned char types in default. When programs created in H8 based on the requirement that char types are signed char types are migrated to RX, they may not operate properly.

Example: Code for which operation is different due to absence/presence of sign for the char type

```
Source code

char a = -1;

void main(void)
{
  if (a < 0) {
    // char types are signed and 'a' is interpreted as negative,
    // so the expression is satisfied (H8)
  } else {
    // char types are unsigned and 'a' is interpreted as positive,
    // so the expression is not satisfied (RX)
  }
}
```

To migrate to RX a program created in H8 based on the requirement that a signed char type is used for the char type, specify the “signed_char” option. For details about how to specify this option, see 1.1 Specifying sign for the char type.

2.2 Specifying sign for bit-field members

With H8-family compilers, bit-field members with no sign specified as treated as signed types, whereas with RX-family compilers, they are treated as unsigned types in default.

When programs created in H8 based on the requirement that bit-field members with no sign specified are signed types are migrated to RX, they may not operate properly.

Example: Code for which operation is different due to absence/presence of sign for bit-field members

```
Source code

struct S {
  int a : 15;
} s = { -1 };

void main(void)
{
  if (s.a < 0) {
    // bit-field members are signed and 's.a' is interpreted as negative,
    // so the expression is satisfied (H8)
  } else {
    // bit-field members are unsigned and 's.a' is interpreted as positive,
    // so the expression is not satisfied (RX)
  }
}
```

To migrate to RX a program created in H8 based on the requirement that a bit-field member with no sign specified is a signed type, specify the “signed_bitfield” option. For details about how to specify this option, see 1.2 Specifying sign for bit-field members.

2.3 Endian

With H8-family compilers, the data byte order is big-endian, whereas with RX-family compilers, it is little-endian in default.

When programs created in H8 based on the requirement that the data byte order is big-endian are migrated to RX, they may not operate properly.

Example: Code for which operation is different due to variance for endian

```
Source code

typedef union{
  short data1;
  struct {
    unsigned char upper;
    unsigned char lower;
  } data2;
} UN;

UN u = { 0x7f6f };

void main(void)
{
  if (u.data2.upper == 0x7f && u.data2.lower == 0x6f) {
    // When the data byte order is big-endian (H8)
  } else {
    // When the data byte order is little-endian (RX)
  }
}
```

To migrate to RX a program created based on the requirement that data byte order is big-endian, specify the “endian=big” option. For details about how to specify this option, see 1.4 Specifying endian.

2.4 Size of double type

With H8-family compilers, the size of the double type is 8 bytes, whereas with RX-family compilers, the size of the double type is 4 bytes in default. When H8 programs created based on the requirement that the size of the double type is 8 bytes are migrated to RX, they may not operate properly.

Example: Code for which operation is different due to variance in double type size

```
Source code

double d1 = 1E30;
double d2 = 1E20;

void main(void)
{
  d1 = d1 * d1; // When the size of the double type is 4 bytes, d1 * d1 overflows
  d2 = d2 * d2; // When the size of the double type is 4 bytes, d2 * d2 overflows

  if (d1 > d2) {
    // When the size of the double type is 8 bytes,
    // normal size comparison is performed (H8)
  } else {
    // When the size of the double type is 4 bytes,
    // size comparison fails because both d1 and d2 overflow (RX)
  }
}
```

When migrating programs created based on the requirement that the size of the double type is 8 bytes to RX, specify the "dbl_size=8" option. For details about how to specify this option, see 1.5 Specifying the size of double type.

2.5 Size of int type

On H8-family compilers, the size of the int type is 2 bytes, whereas on RX-family compilers the size of the int type is 4 bytes in default. When H8 programs created based on the requirement that the size of the int type is 2 bytes are migrated to RX, they may not operate properly.

Example: Code for which operation is different due to variance in int type size

```
Source code

typedef union{
  long data;
  struct {
    int dataH;
    int dataL;
  } s;
} UN;

void main(void)
{
  UN u;
  u.s.dataH = 0;
  u.s.dataL = 1;

  if (u.data == 0) {
    // When the size of the int type is 4 bytes (RX)
  } else {
    // When the size of the int type is 2 bytes (H8)
  }
}
```

To migrate to RX a program created based on the requirement that the size of the int type is 2 bytes, specify the "int_to_short" option. For details about how to specify this option, see 1.6 Correspondence of int type size to difference.

2.6 asm blocks

H8-family compilers allow asm blocks to be used to code assembly language programs in C source programs. Since RX-family compilers lack the corresponding functionality, programs using asm blocks need special handling when migrated to RX.

RX-family compilers have assembly code functions to code assembly language in C source programs. The contents coded in the asm block can sometimes be handled by being coded in the assembly code function.

For details about assembly code functions, see Compiler User's Manual.

Example:

Program using the H8 asm blocks and program using the RX assembly code function

Source code using an H8 asm block	Source code using the RX assembly code function

C source code
`void func(void)`
`{`
`  __asm {`
`    NOP`
`  }`
`}`
 	
Assembler source expansion code
`_func:`
`  NOP`
`  rts`
	C source code
	`#pragma inline_asm asm_nop`
	`static void asm_nop(void)`
	`{`
	`  NOP`
	`}`
	
	`void func(void)`
	`{`
	`  asm_nop();`
	`}`
	
	Assembler source expansion code
	`_func:`
	`  NOP`
	`  RTS`

Precautions

- H8 allows variables to be coded in the assembler, but RX does not.

3. Optimization option setting for migration from H8-family

There is a difference in an optional setting method for optimization in the compiler of H8-family, and RX-family.

Please refer to the following optimization option setting when embedded software transplant from H8-family to RX-family and the performance is evaluated.

Optimization option setting of each compiler and comparison of ROM size

(The sample program for the measurement is described to the next page.)

H8SX	Optimize OFF	Object Size Precedence	Execution Speed Precedence
	opt=0	**opt=1**	**opt=1 speed**
main()	0xB8	0x96	0x96
sort()	0xA2	0x5E	0x6E

* By H8-family compiler V.7.00 Release 00

RX	Optimize OFF	Object Size Precedence			Execution Speed Precedence		
	optimize=0	**optimize=1**	**optimize=2**	**optimize =max**	**optimize=1 speed**	**optimize=2 speed**	**optimize =max speed**
main()	0xAC	0x80	0x76	0x76	0x80	0x76	0x76
sort()	0x92	0x47	0x51	0x53	0x4A	0x5D	0x5F

* By RX-family compiler V2.05.00

Please refer to the compiler user's manual for details of the optimization level of the RX-family compiler.

Exsample: Sample source

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

void main(void);
void sort(long *a);
void change(long *a);

void main(void)
{
    long a[10];
    long j;
    int i;

    printf("### Data Input ###¥n");

    for( i=0; i<10; i++ ){
        j = rand();
        if(j < 0){
            j = -j;
        }
        a[i] = j;
        printf("a[%d]=%ld¥n",i,a[i]);
    }
    sort(a);
    printf("*** Sorting results ***¥n");
    for( i=0; i<10; i++ ){
        printf("a[%d]=%ld¥n",i,a[i]);
    }
    change(a);
}
```

```
void sort(long *a)
{
    long t;
    int i, j, k, gap;

    gap = 5;
    while( gap > 0 ){
        for( k=0; k<gap; k++){
            for( i=k+gap; i<10; i=i+gap ){
                for(j=i-gap; j>=k; j=j-gap){
                    if(a[j]>a[j+gap]){
                        t = a[j];
                        a[j] = a[j+gap];
                        a[j+gap] = t;
                    }else{
                        break;
                    }
                }
            }
        }
        gap = gap/2;
    }
}
```

```
void change(long *a)
{
    long tmp[10];
    int i;
    for(i=0; i<10; i++){
        tmp[i] = a[i];
    }
    for(i=0; i<10; i++){
        a[i] = tmp[9 - i];
    }
}
```

Website and Support

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

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

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Oct. 1, 2009	--	Initial edition
2.00	Nov. 30, 2016	--	Revised the destination to CS+ and CC-RX

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.

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, HALII 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 5.0