RX Family C/C++ Compiler Package

Application Notes: Sample Project RX Migration Guide, H8 Edition

This document explains how to migrate the sample project created in H8 to RX.

Table of contents

This edition is a guide for migrating H8 sample projects for which operation can be confirmed in the simulator debugger, to RX.

1. Overview of the H8 sample project

The H8 sample project 'H8_Sample' can be broadly divided into pre- and post-processing such as for initialization, and main processing to perform central processing. This edition shows how to migrate the main processing to perform central processing to an RX project, and check its operation. The following table shows the files that comprise main processing.

Table 1-1 List of main processing files

No	Processing	File name	Reference
1	Specifying sign for the char type	H8_sign_char.c	2.4.1
2	Specifying sign for bit field members	H8_sign_bit_field.c	2.4.2
3	Specifying bit field member allocation	H8_bit_order.c	2.4.3
4	Specifying endian	H8_endian.c	2.4.4
5	Specifying the size of double type variables	H8_double_size.c	2.4.5
6	Specifying the size of int type variables	H8_int_size.c	2.7.1
7	main function	H8_Sample_main.c	-

2. Migrating the H8 sample project to RX

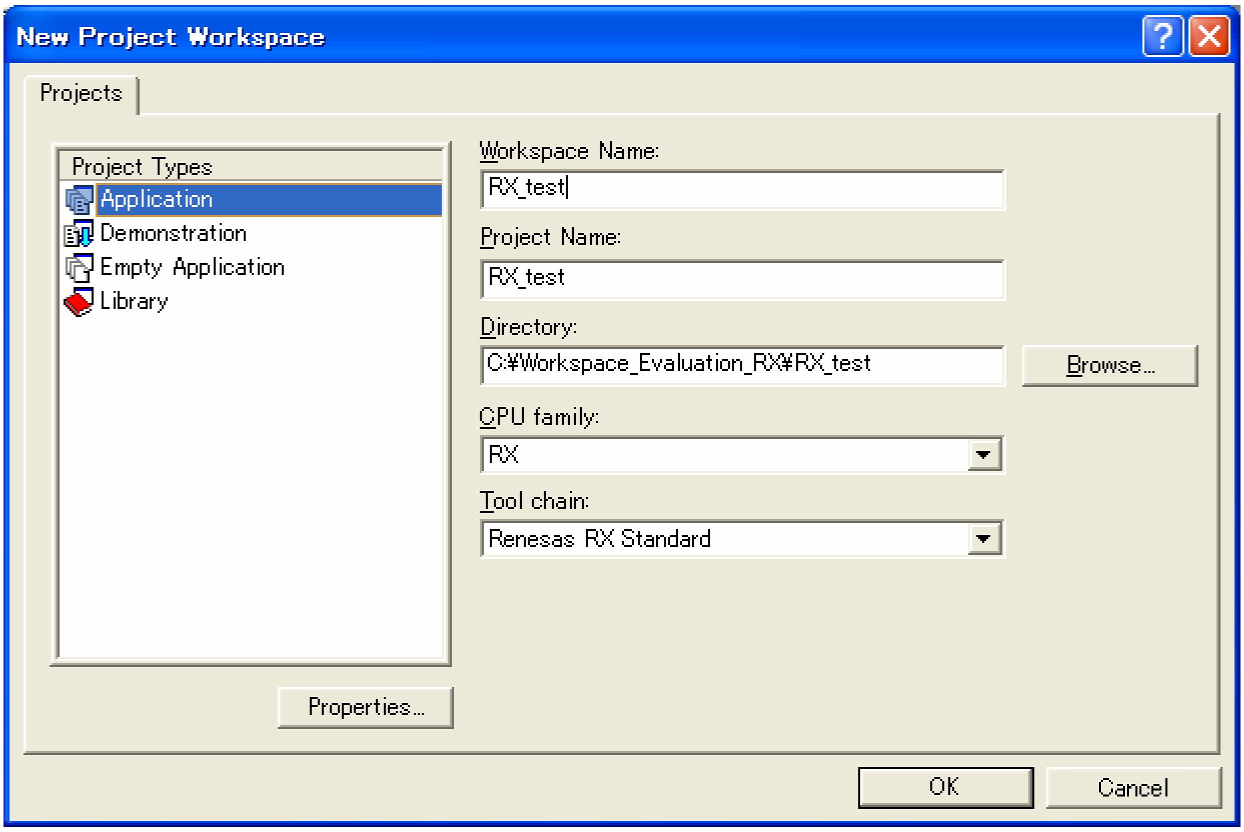
2.1 Creating the RX project

Create a new RX project workspace for the migration destination of the H8 sample project.

This section explains how to generate sample project in the project generator (launched from HEW by choosing the File menu and then New workspace), according to the following procedures.

(1) Creating a new workspace

Select the “Application” project type.



Workspace Name: RX_test

Project Name: RX_test

Directory: C:¥Workspace_Evaluation_RX¥RX_test

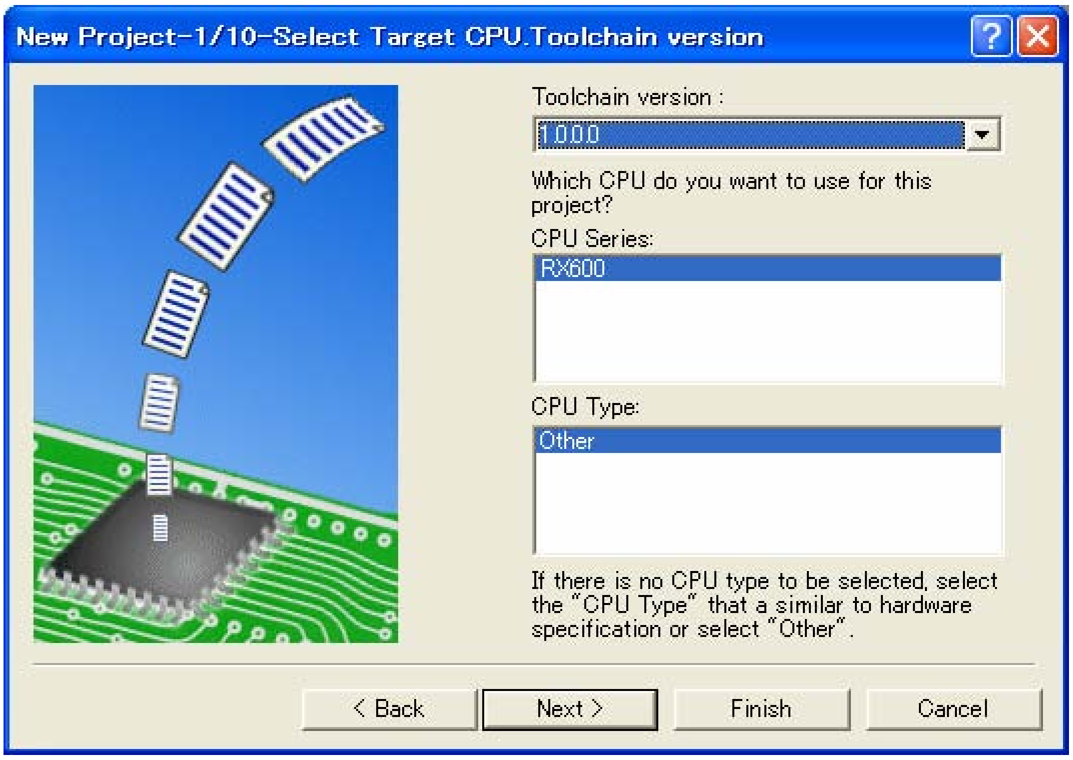
CPU family: RX

Tool chain: Renesas RX Standard

Figure 1-1

(2) Selecting a CPU

Leave this set to the default value, and proceed.



New Project-1/10-Select Target CPU.Toolchain version

Toolchain version : 1.0.0.0

Which CPU do you want to use for this project?

CPU Series: RX600

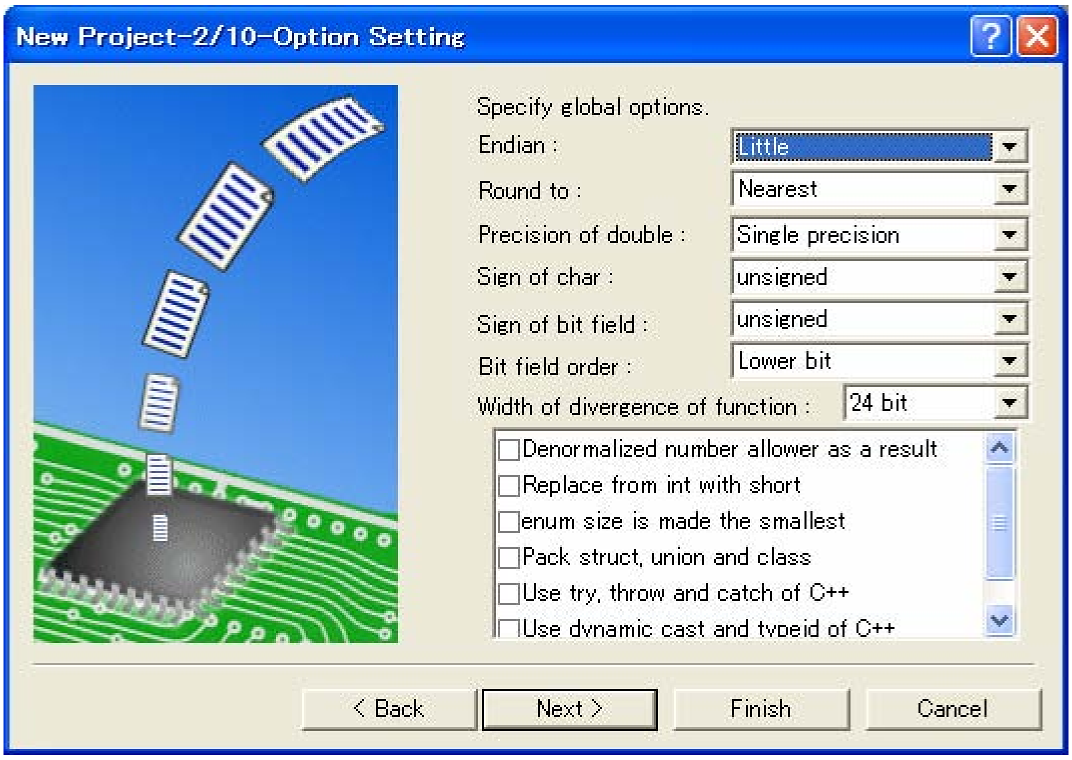
CPU Type: Other

If there is no CPU type to be selected, select the "CPU Type" that a similar to hardware specification or select "Other".

Figure 1-2

(3) Setting options

Leave this set to the default value, and proceed.



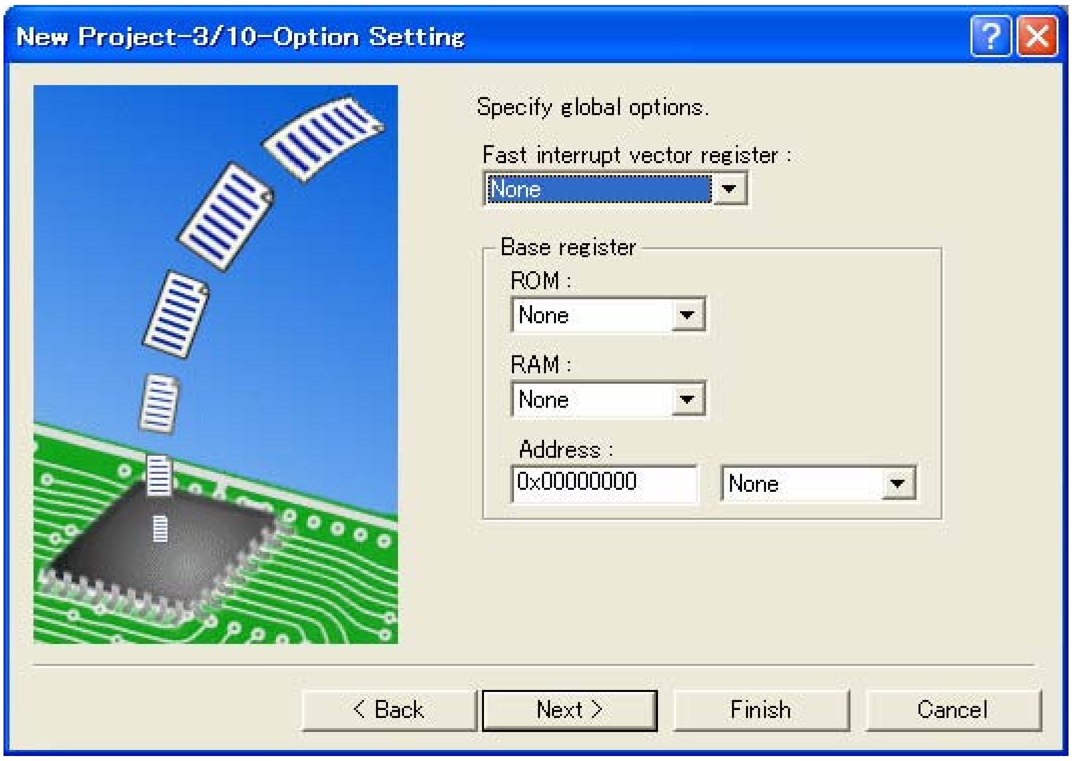
New Project-2/10-Option Setting

Specify global options.

Endian :	Little
Round to :	Nearest
Precision of double :	Single precision
Sign of char :	unsigned
Sign of bit field :	unsigned
Bit field order :	Lower bit
Width of divergence of function :	24 bit

- [ ] Denormalized number allower as a result
- [ ] Replace from int with short
- [ ] enum size is made the smallest
- [ ] Pack struct, union and class
- [ ] Use try, throw and catch of C++
- [ ] Use dynamic cast and typeid of C++

Figure 1-3



New Project-3/10-Option Setting

Specify global options.

Fast interrupt vector register : None

Base register

ROM : None

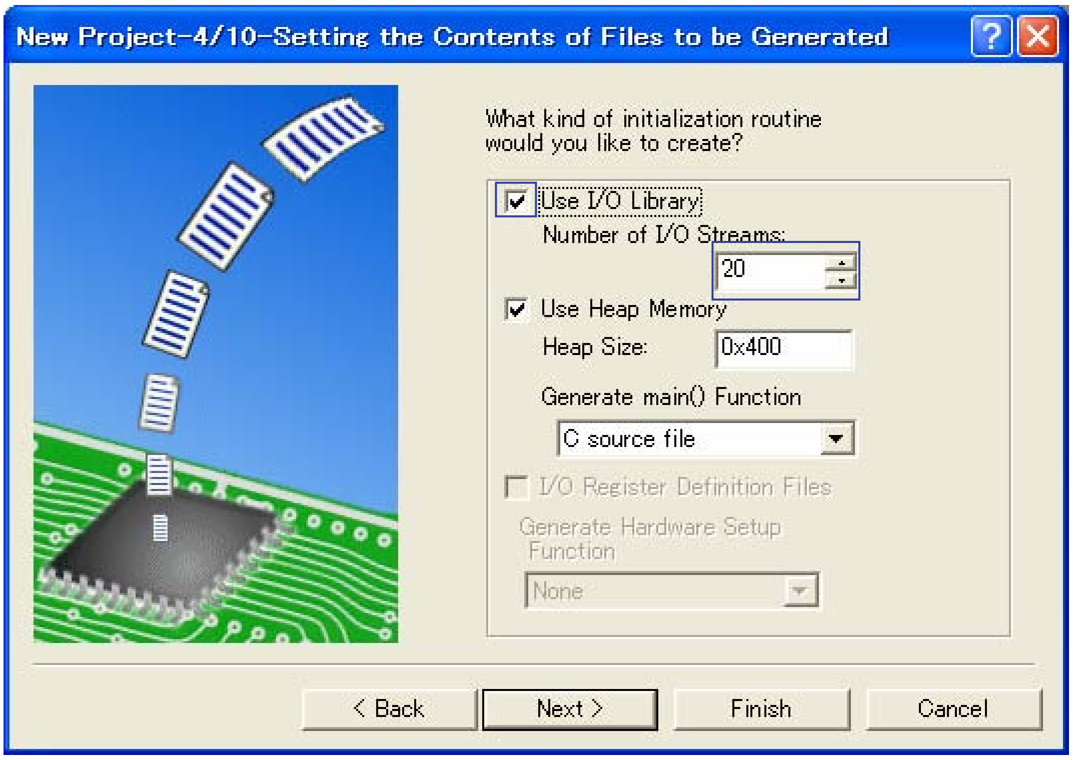
RAM : None

Address : 0x00000000 None

Figure 1-4

(4) Set up generated files

Select “Use I/O library”.
Specify “20” for “I/O stream count”.


New Project-4/10-Setting the Contents of Files to be Generated

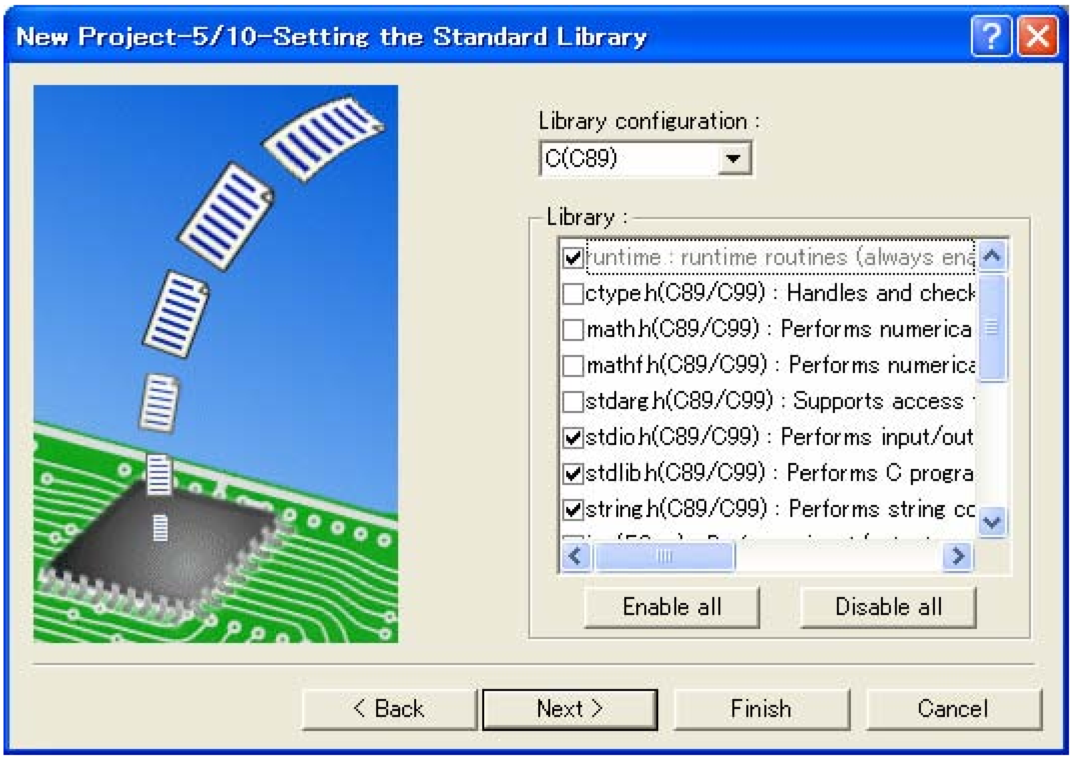
What kind of initialization routine would you like to create?

[✓] Use I/O Library
Number of I/O Streams: 20
[✓] Use Heap Memory
Heap Size: 0x400
Generate main() Function: C source file
[ ] I/O Register Definition Files
Generate Hardware Setup Function: None

Figure 1-5

(5) Set up the standard library

Leave this set to the default value, and proceed.


New Project-5/10-Setting the Standard Library

Library configuration : C(C89)

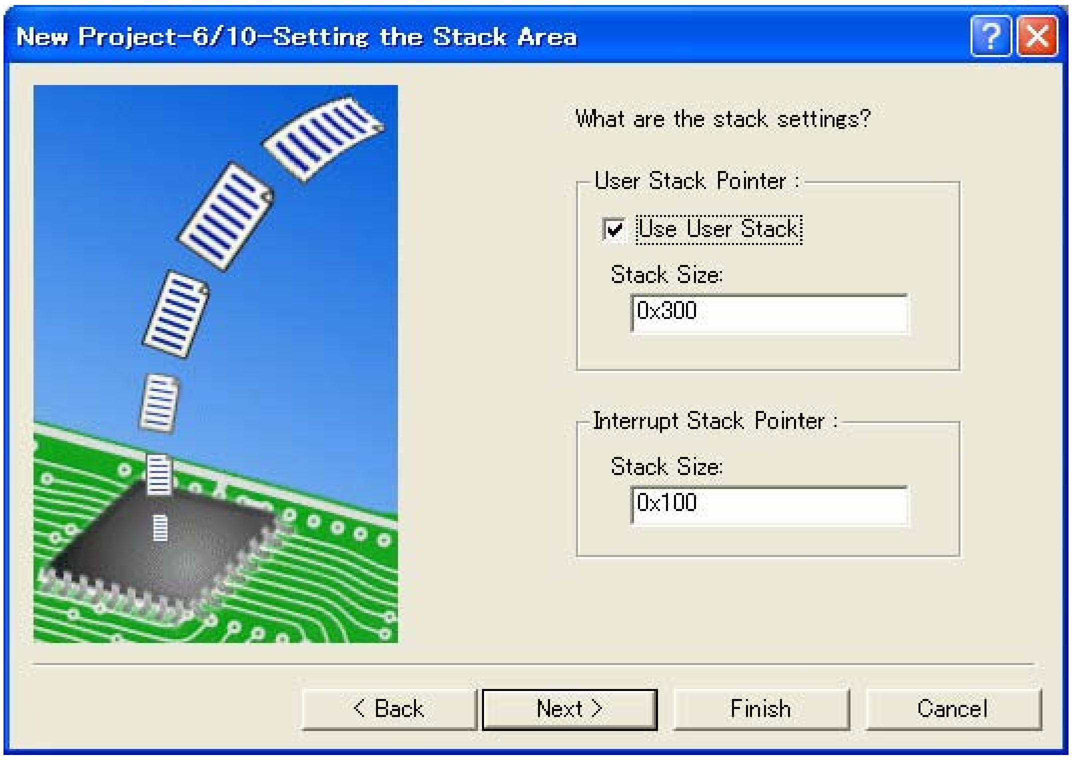
Library :

- [✓] runtime : runtime routines (always ena
- [ ] ctype.h(C89/C99) : Handles and check
- [ ] math.h(C89/C99) : Performs numerica
- [ ] mathf.h(C89/C99) : Performs numerica
- [ ] stdarg.h(C89/C99) : Supports access
- [✓] stdio.h(C89/C99) : Performs input/out
- [✓] stdlib.h(C89/C99) : Performs C progra
- [✓] string.h(C89/C99) : Performs string co

Figure 1-6

(6) Set up the stack space

Leave this set to the default value, and proceed.



New Project-6/10-Setting the Stack Area

What are the stack settings?

User Stack Pointer :

☑ Use User Stack

Stack Size: 0x300

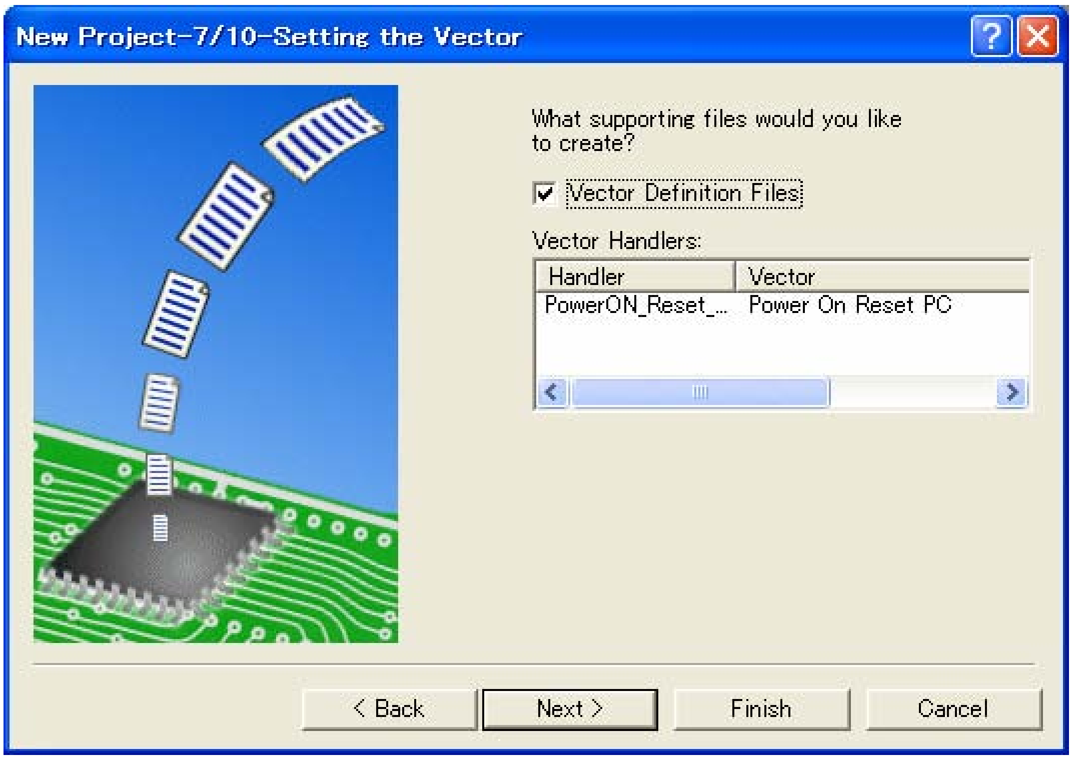
Interrupt Stack Pointer :

Stack Size: 0x100

Figure 1-7

(7) Set up vectors

Leave this set to the default value, and proceed.



New Project-7/10-Setting the Vector

What supporting files would you like to create?

[✓] Vector Definition Files

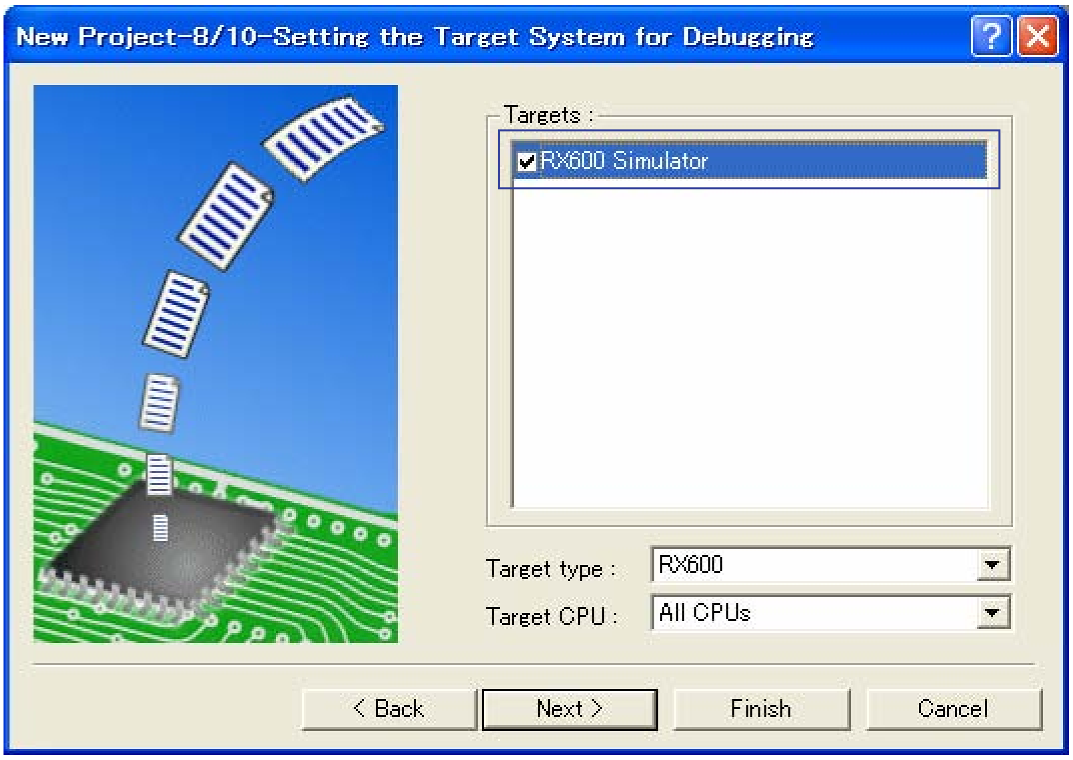
Vector Handlers:

Handler	Vector
PowerON_Reset_...	Power On Reset PC

Figure 1-8

(8) Set up the debugger

Select “RX600 Simulator”.


New Project-8/10-Setting the Target System for Debugging

Targets :

☑ RX600 Simulator

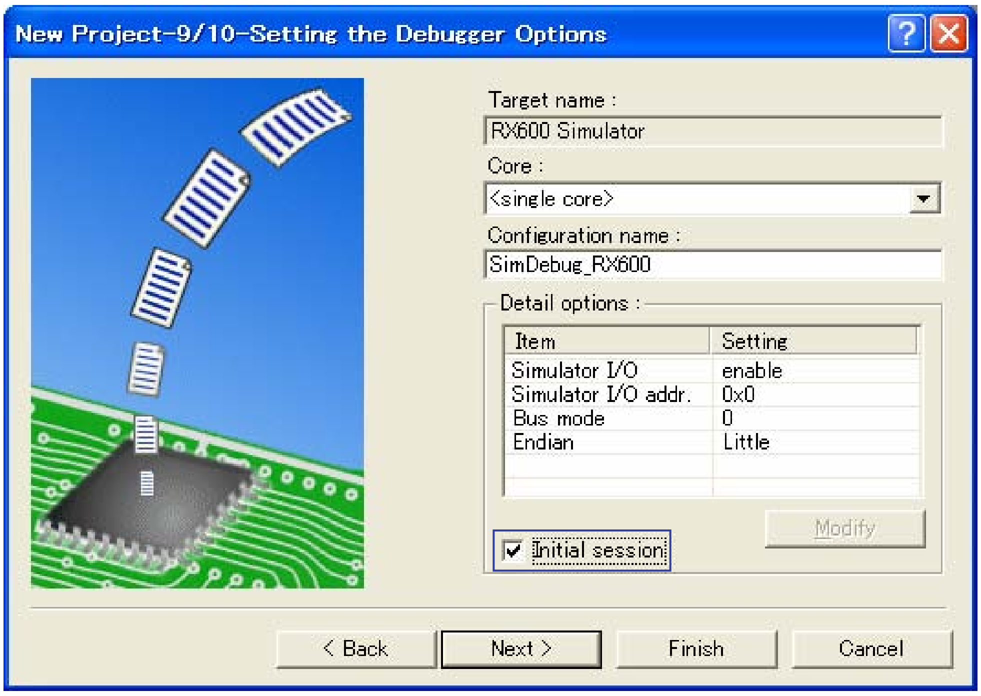
Target type : RX600

Target CPU : All CPUs

Figure 1-9

(9) Set the debugger options

Select “Initial session”.



Target name : RX600 Simulator

Core : <single core>

Configuration name : SimDebug_RX600

Detail options :

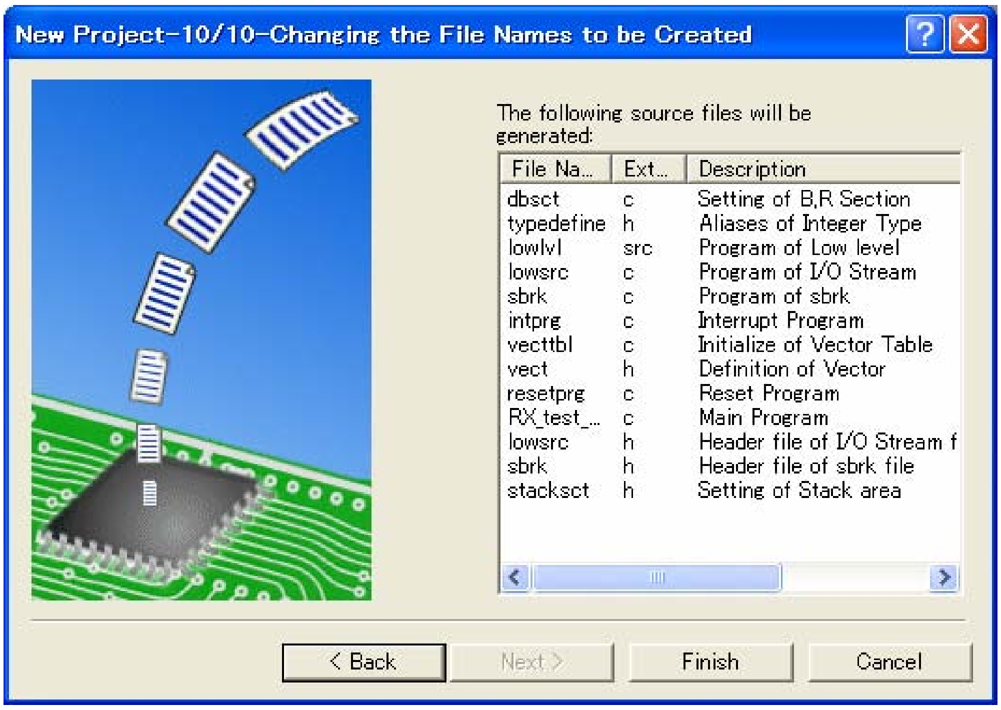
Item	Setting
Simulator I/O	enable
Simulator I/O addr.	0x0
Bus mode	0
Endian	Little

☑ Initial session

Figure 1-10

(10) Check the generated file names

Click “Finish”.



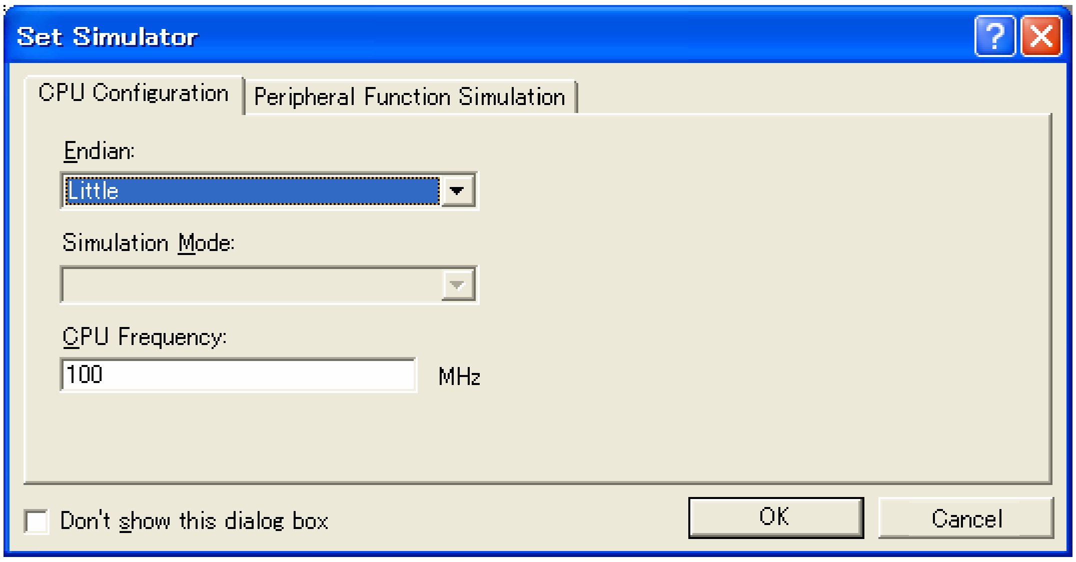
The following source files will be generated:

File Na...	Ext...	Description
dbsct	c	Setting of B,R Section
typedefine	h	Aliases of Integer Type
lowlvl	src	Program of Low level
lowsrc	c	Program of I/O Stream
sbrk	c	Program of sbrk
intprg	c	Interrupt Program
vecttbl	c	Initialize of Vector Table
vect	h	Definition of Vector
resetprg	c	Reset Program
RX_test_...	c	Main Program
lowsrc	h	Header file of I/O Stream f
sbrk	h	Header file of sbrk file
stacksct	h	Setting of Stack area

Figure 1-11

(11) Set up the simulator

Click “OK”.


Set Simulator

CPU Configuration | Peripheral Function Simulation

Endian: Little

Simulation Mode:

CPU Frequency: 100 MHz

☐ Don't show this dialog box

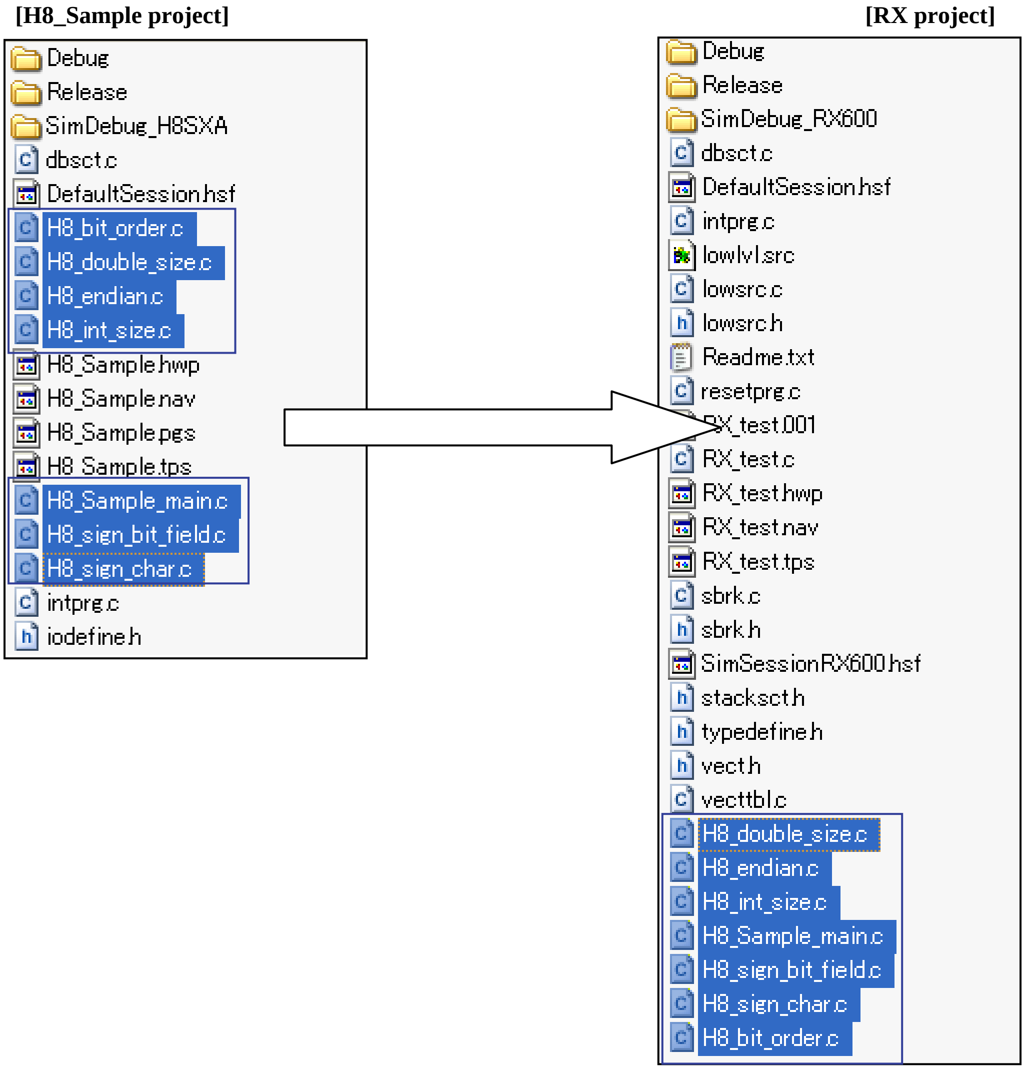
Figure 1-12

2.2 Migrating source files for main processing

Copy, and register with the created RX project, the files comprising main processing for the H8 sample project, as explained in 1. Overview of the H8 sample project.

(1)Copy files from the H8 sample project folder

Copy to the RX project the 10 files explained in *1. Overview of the H8 sample project.*



```
[H8_Sample project]          [RX project]
Debug                        Debug
Release                      Release
SimDebug_H8SXA               SimDebug_RX600
dbsct.c                      dbsct.c
DefaultSession.hsf           DefaultSession.hsf
H8_bit_order.c               intprg.c
H8_double_size.c             lowlvl.src
H8_endian.c                  lowsrc.c
H8_int_size.c                lowsrc.h
H8_Sample.hwp                Readme.txt
H8_Sample.nav                resetprg.c
H8_Sample.pgs                RX_test.001
H8_Sample.tps                RX_test.c
H8_Sample_main.c             RX_test.hwp
H8_sign_bit_field.c          RX_test.nav
H8_sign_char.c               RX_test.tps
intprg.c                     sbrk.c
iodefine.h                   sbrk.h
                             SimSessionRX600.hsf
                             stacksct.h
                             typedefine.h
                             vect.h
                             vecttbl.c
                             H8_double_size.c
                             H8_endian.c
                             H8_int_size.c
                             H8_Sample_main.c
                             H8_sign_bit_field.c
                             H8_sign_char.c
                             H8_bit_order.c
```

Figure 1-13

(2)Register the copied files with the project

Register the copied files with the created RX project.
In HEW, choose Project and then Add files, and then select the following in the displayed dialog box.

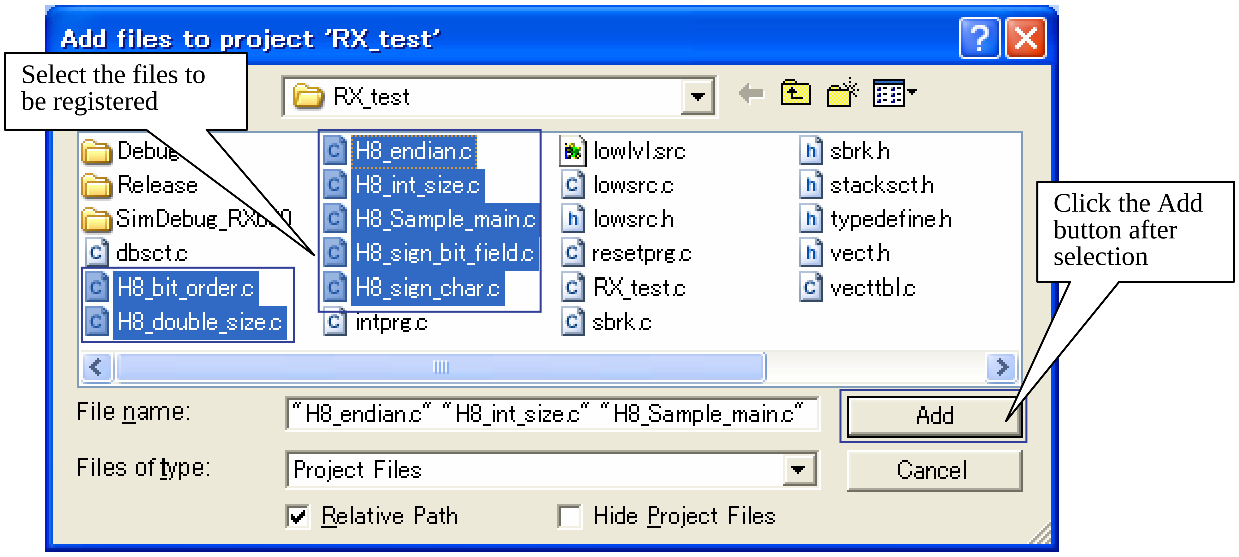
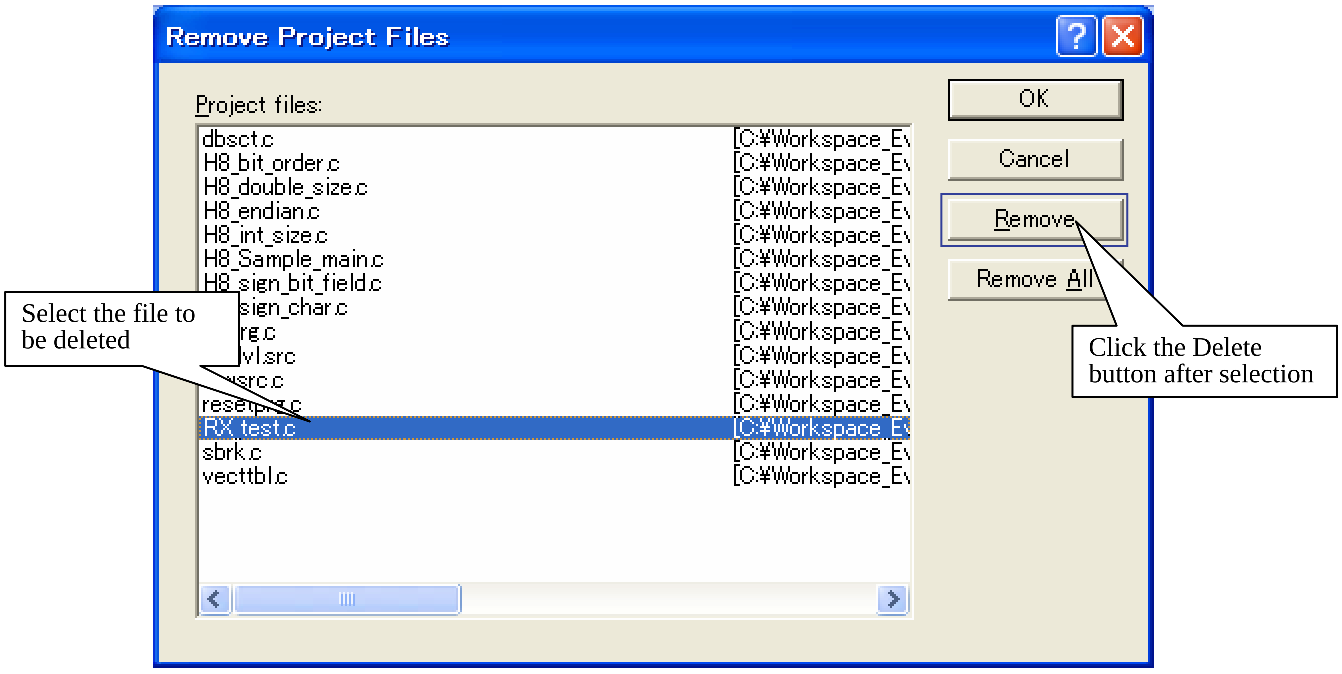


Figure 1-14

(3)Unregister unnecessary files

Delete the 'RX_test.c' file generated by the project generator for the main function. It is no longer necessary, because the main function file was copied from the H8 sample project.

In HEW, choose Project and then Delete files, and then select the following in the displayed dialog box.


Remove Project Files
Project files:
OK
Cancel
Remove
Remove All
Select the file to be deleted
Click the Delete button after selection

```
dbsct.c             [C:¥Workspace_E\
H8_bit_order.c      [C:¥Workspace_E\
H8_double_size.c    [C:¥Workspace_E\
H8_endian.c         [C:¥Workspace_E\
H8_int_size.c       [C:¥Workspace_E\
H8_Sample_main.c    [C:¥Workspace_E\
H8_sign_bit_field.c [C:¥Workspace_E\
  sign_char.c       [C:¥Workspace_E\
  rg.c              [C:¥Workspace_E\
  vl.src            [C:¥Workspace_E\
  src.c             [C:¥Workspace_E\
rese    c           [C:¥Workspace_E\
RX_test.c           [C:¥Workspace_E\
sbrk.c              [C:¥Workspace_E\
vecttbl.c           [C:¥Workspace_E\
```

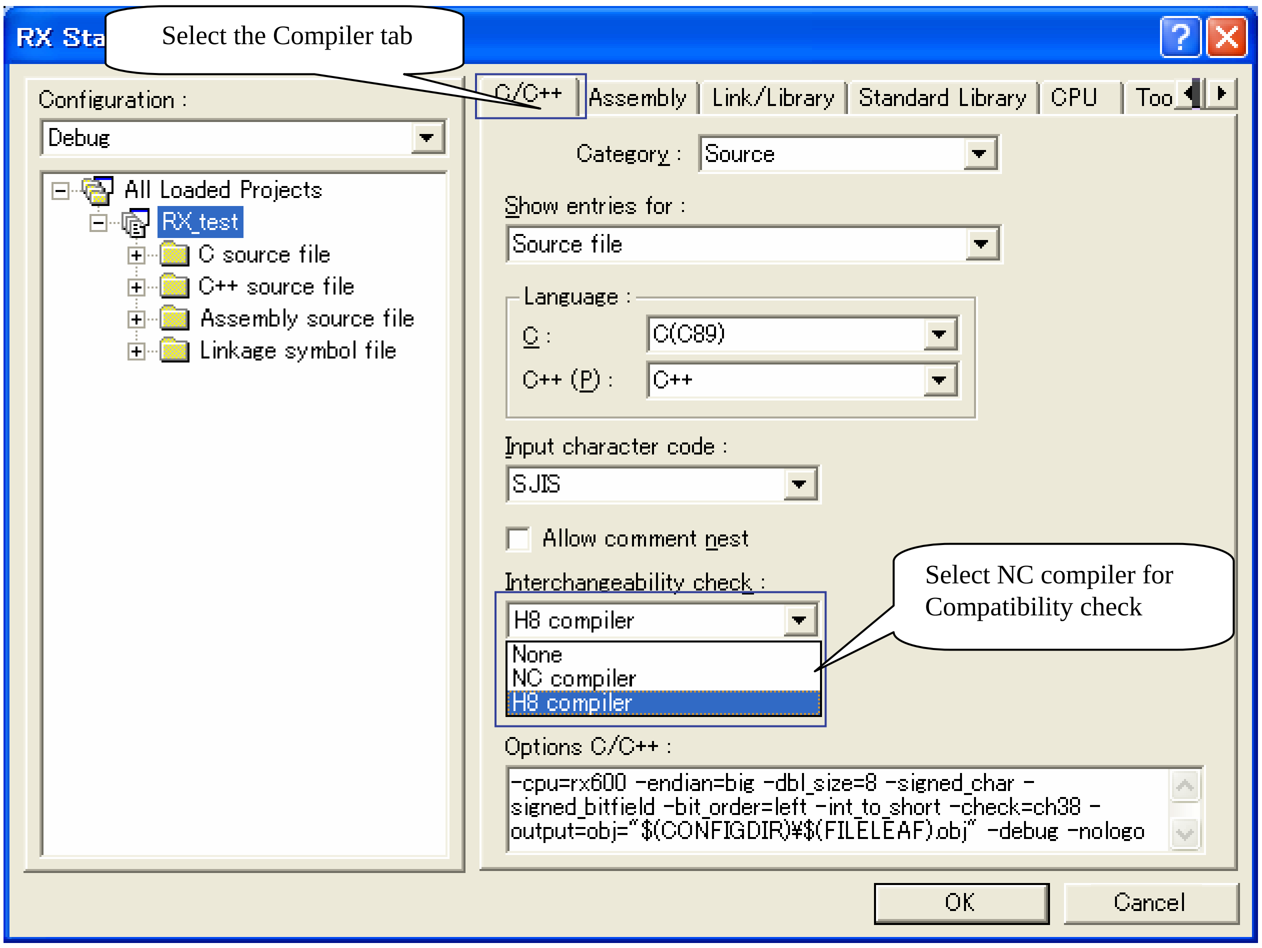
Figure 1-15

2.3 Building and checking H8 compatibility

Build the RX project for which the main processing file was copied and registered. When a build is performed in HEW, since the H8 compatibility check functionality is enabled, option specifications and source code that may impact compatibility can be checked. This section explains how to enable H8 compatibility check functionality and perform a build, and then check the displayed compatibility confirmation messages.

(1) Set up H8 compatibility check functionality

In HEW, choose Build and then RX Standard Toolchain, and then select the following in the displayed dialog box.



Category : Source

Show entries for : Source file

Language :

C : C(C89)

C++ (P) : C++

Input character code : SJIS

Allow comment nest

Interchangeability check : H8 compiler

Options C/C++ :

```
-cpu=rx600 -endian=big -dbl_size=8 -signed_char -
signed_bitfield -bit_order=left -int_to_short -check=ch38 -
output=obj="$(CONFIGDIR)¥$(FILELEAF).obj" -debug -nologo
```

Figure 1-16

(2) Building

In HEW, choose Build and then Build to start the build. A message is displayed during the build, in the output window.

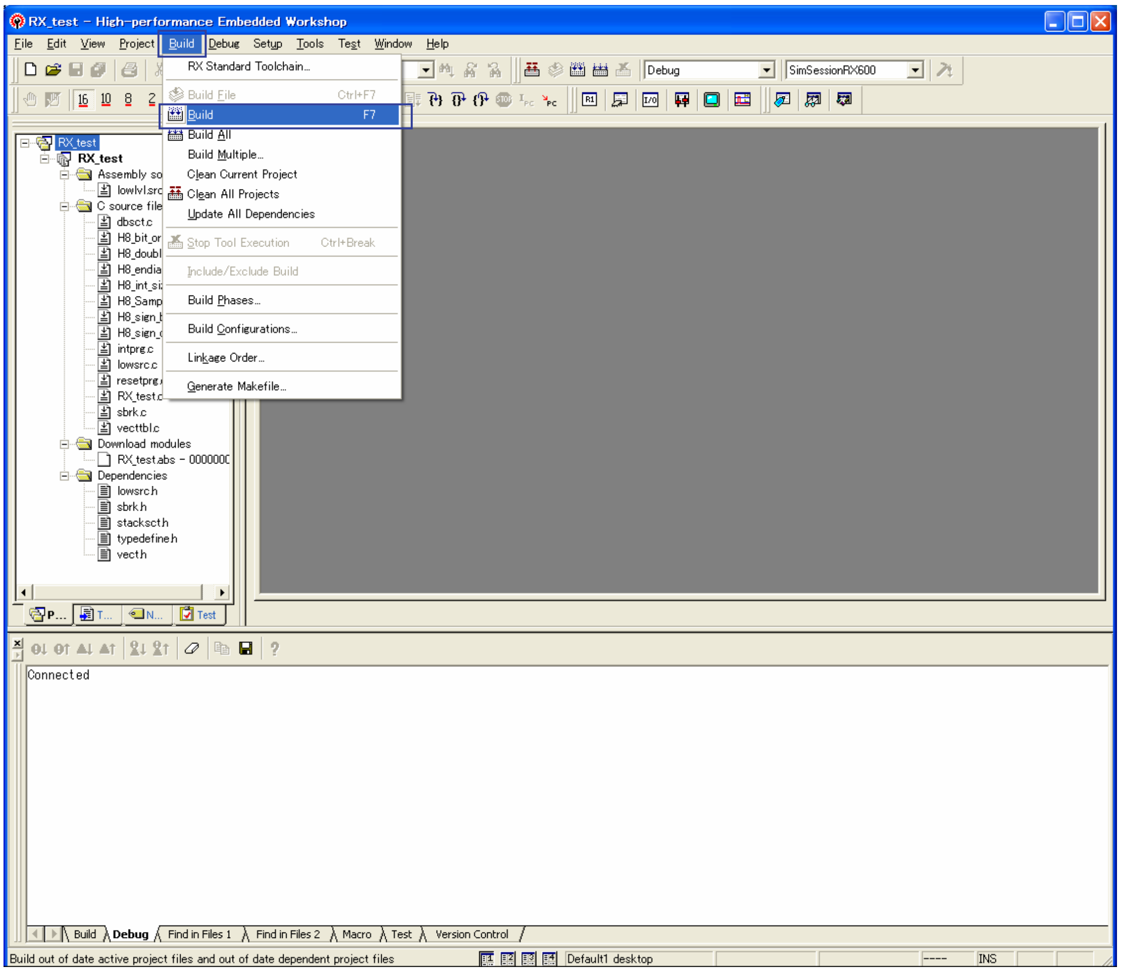

RX_test - High-performance Embedded Workshop
File Edit View Project Build Debug Setup Tools Test Window Help
RX Standard Toolchain...
Build File Ctrl+F7
Build F7
Build All
Build Multiple...
Clean Current Project
Clean All Projects
Update All Dependencies
Stop Tool Execution Ctrl+Break
Include/Exclude Build
Build Phases...
Build Configurations...
Linkage Order...
Generate Makefile...
Debug
SimSessionRX600
Connected
Build Debug Find in Files 1 Find in Files 2 Macro Test Version Control
Build out of date active project files and out of date dependent project files
Default1 desktop
INS

Figure 1-17

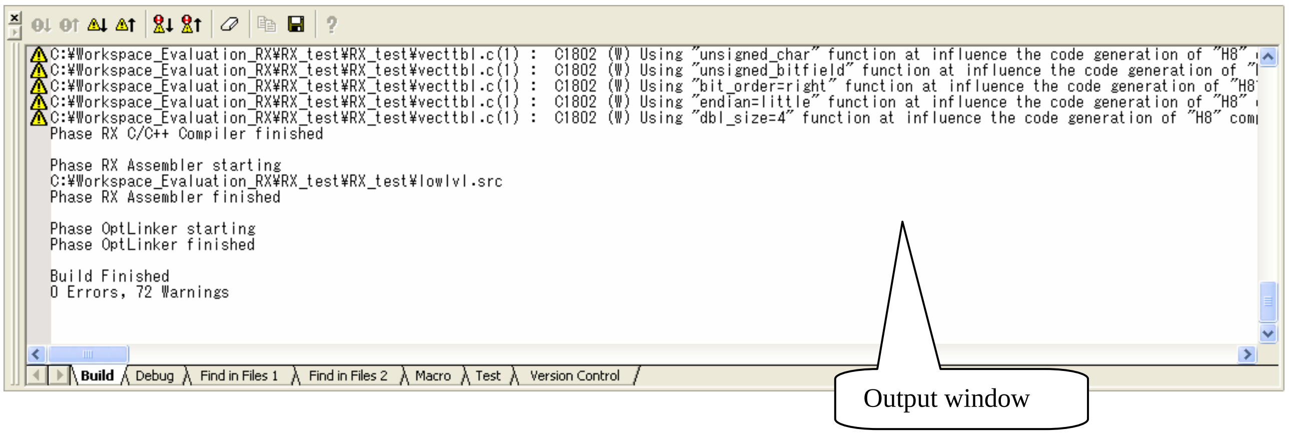
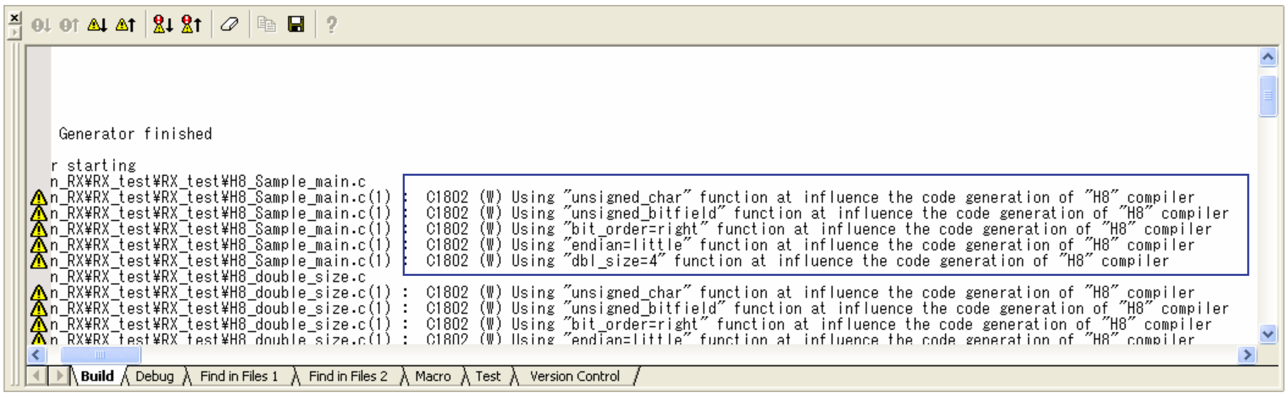

```
C:¥Workspace_Evaluation_RX¥RX_test¥RX_test¥vecttbl.c(1) :  C1802 (W) Using "unsigned_char" function at influence the code generation of "H8"
C:¥Workspace_Evaluation_RX¥RX_test¥RX_test¥vecttbl.c(1) :  C1802 (W) Using "unsigned_bitfield" function at influence the code generation of "
C:¥Workspace_Evaluation_RX¥RX_test¥RX_test¥vecttbl.c(1) :  C1802 (W) Using "bit_order=right" function at influence the code generation of "H8
C:¥Workspace_Evaluation_RX¥RX_test¥RX_test¥vecttbl.c(1) :  C1802 (W) Using "endian=little" function at influence the code generation of "H8"
C:¥Workspace_Evaluation_RX¥RX_test¥RX_test¥vecttbl.c(1) :  C1802 (W) Using "dbl_size=4" function at influence the code generation of "H8" com
Phase RX C/C++ Compiler finished

Phase RX Assembler starting
C:¥Workspace_Evaluation_RX¥RX_test¥RX_test¥lowlvl.src
Phase RX Assembler finished

Phase OptLinker starting
Phase OptLinker finished

Build Finished
0 Errors, 72 Warnings
```
Build Debug Find in Files 1 Find in Files 2 Macro Test Version Control
Output window

Figure 1-18

(3) Checking compatibility check messages

The C1802 warning is displayed in the messages output to the output window, indicating that this area poses a compatibility problem with H8.



```
 Generator finished

r starting
n_RX¥RX_test¥RX_test¥H8_Sample_main.c
n_RX¥RX_test¥RX_test¥H8_Sample_main.c(1) :  C1802 (W) Using "unsigned_char" function at influence the code generation of "H8" compiler
n_RX¥RX_test¥RX_test¥H8_Sample_main.c(1) :  C1802 (W) Using "unsigned_bitfield" function at influence the code generation of "H8" compiler
n_RX¥RX_test¥RX_test¥H8_Sample_main.c(1) :  C1802 (W) Using "bit_order=right" function at influence the code generation of "H8" compiler
n_RX¥RX_test¥RX_test¥H8_Sample_main.c(1) :  C1802 (W) Using "endian=little" function at influence the code generation of "H8" compiler
n_RX¥RX_test¥RX_test¥H8_Sample_main.c(1) :  C1802 (W) Using "dbl_size=4" function at influence the code generation of "H8" compiler
n_RX¥RX_test¥RX_test¥H8_double_size.c
n_RX¥RX_test¥RX_test¥H8_double_size.c(1) :  C1802 (W) Using "unsigned_char" function at influence the code generation of "H8" compiler
n_RX¥RX_test¥RX_test¥H8_double_size.c(1) :  C1802 (W) Using "unsigned_bitfield" function at influence the code generation of "H8" compiler
n_RX¥RX_test¥RX_test¥H8_double_size.c(1) :  C1802 (W) Using "bit_order=right" function at influence the code generation of "H8" compiler
n_RX¥RX_test¥RX_test¥H8_double_size.c(1) :  C1802 (W) Using "endian=little" function at influence the code generation of "H8" compiler
```

Figure 1-19

2.4 Handling compatibility check instructions

The following explains how to check and deal with C1802 messages affecting compatibility, as given in *2.3 Building and checking H8 compatibility*. The target messages are as follows.

Table 1-2 List of C1802 messages

No	Messages affecting compatibility	Reference
1	Using "unsigned_char" function at influence the code generation of "H8" compiler	2.4.1
2	Using "unsigned_bitfield" function at influence the code generation of "H8" compiler	2.4.2
3	Using "bit_order=right" function at influence the code generation of "H8" compiler	2.4.3
4	Using "endian=little" function at influence the code generation of "H8" compiler	2.4.4
5	Using "dbl_size=4" function at influence the code generation of "H8" compiler	2.4.5

2.4.1 Specifying sign for the char type

The message 'Using "unsigned_char" function at influence the code generation of "H8" compiler' indicates that a compatibility problem exists with the specified “unsigned_char” option. H8-family compilers treat char types with no sign specified as signed char types, whereas RX-family compilers treat them as unsigned char types.

Since “H8_sign_char.c” in the sample program contains code based on the requirement that char types without a sign are signed char types, if the “unsigned_char” option is specified, the operation results will differ from H8.

“H8_sign_char.c” in the sample program

```
Source code

struct S {
   char a;
} s = { -1 };

void sign_char(void)
{
   printf("(1) sign char : ");

   if (s.a < 0) {
      printf("OK¥n");
   } else {
      printf("NG¥n");
   }
}
```

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. For details about how to specify this option, see c*ompiler users manual*. Also, change the options specified in the created RX project.

2.4.2 Specifying sign for bit field members

The message ‘Using "unsigned_bitfield" function at influence the code generation of "H8" compiler’ indicates that a compatibility problem exists with the specified “ unsigned_bitfield” option. H8-family compilers treat bit field members with no sign specified as signed types, whereas RX-family compilers treat them as unsigned types.

Since “H8_sign_bit_field.c” in the sample program contains code based on the requirement that bit field members with no sign specified are signed, if the “ unsigned_bitfield” option is specified, the operation results will differ from H8.

“H8_sign_bit_field.c” in the sample program

```
Source code

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

void sign_bit_field(void)
{
   printf("(2) sign bit field : ");

   if (bit.a < 0) {
   printf("OK¥n");
   } else {
      printf("NG¥n");
   }
}
```

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. For details about how to specify this option, see *compiler users manual*. Also, change the options specified in the created RX project.

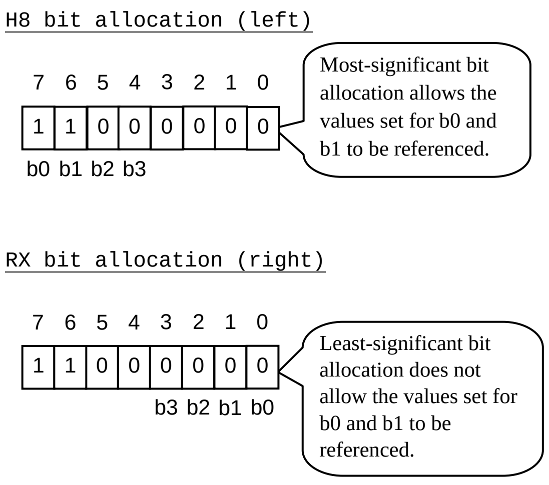
2.4.3 Specifying bit field member allocation

The message 'Using "bit_order=right" function at influence the code generation of "H8" compiler' indicates that a compatibility problem exists with the specified "bit_order=right" option. 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. Since "H8_bit_order.c" in the sample program contains code based on the requirement that bit field members are allocated from the most significant bit, if the "bit_order=right" option is specified, the operation results will differ from H8.

"H8_bit_order.c" in the sample program

Source code

```
union {
    unsigned char c1;
    struct {
        unsigned char b0 : 1;
        unsigned char b1 : 1;
        unsigned char b2 : 1;
        unsigned char b3 : 1;
    } b;
} un;

void bit_order(void)
{
    printf("(3) bit field order : ");

    un.c1 = 0xc0;
    if ((un.b.b0 == 1) && (un.b.b1 == 1) &&
      (un.b.b2 == 0) && (un.b.b3 == 0)) {
        printf("OK¥n");
    } else {
        printf("NG¥n");
    }
}
```


H8 bit allocation (left)
7 6 5 4 3 2 1 0
1 1 0 0 0 0 0 0
b0 b1 b2 b3
Most-significant bit allocation allows the values set for b0 and b1 to be referenced.
RX bit allocation (right)
7 6 5 4 3 2 1 0
1 1 0 0 0 0 0 0
b3 b2 b1 b0
Least-significant bit allocation does not allow the values set for b0 and b1 to be referenced.

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. For details about how to specify this option, see *compiler users manual*. Also, change the options specified in the created RX project.

2.4.4 Specifying endian

The message 'Using "endian=little" function at influence the code generation of "H8" compiler' indicates that a compatibility problem exists with the specified "endian=little" option. With H8-family compilers, the data byte order is big-endian, whereas with RX-family compilers, it is little-endian. Since "H8_endian.c" in the sample program contains code based on the requirement that the data byte order is big-endian, if the "endian=little" option is specified, the operation results will differ from H8.

"H8_endian.c" in the sample program

```
Source code

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

UN u = { 0x7f6f };

void endian(void)
{
    printf("(3) endian : ");

    if (u.data2.upper == 0x7f && u.data2.lower == 0x6f) {
        printf("OK¥n");
    } else {
        printf("NG¥n");
    }
}
```

To migrate to RX a program created in H8 based on the requirement that the data byte order is big-endian, specify the "endian=little" option. For details about how to specify this option, see *compiler users manual*. Also, change the options specified in the created RX project.

2.4.5 Specifying the size of double type variables

The message ‘Using "dbl_size=4" function at influence the code generation of "H8" compiler’ indicates that a compatibility problem exists with the specified “dbl_size=4” option. 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. Since “H8_double_size.c” in the sample program contains code based on the requirement that the size of the double type is 8 bytes, if the “ dbl_size=4” option is specified, the operation results will differ from H8.

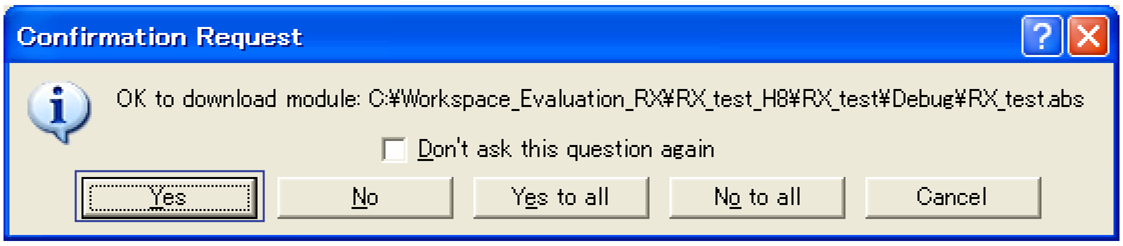
“H8_double_size.c” in the sample program

```
Source code

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

void double_size(void)
{
    d1 = d1 * d1;
    d2 = d2 * d2;

    printf("(5) double type size : ");

    if (d1 > d2) {
        printf("OK¥n");
    } else {
        printf("NG¥n");
    }
}
```

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. For details about how to specify this option, see *compiler users manual*. Also, change the options specified in the created RX project.

2.5 Rebuilding

Once the specified options and source code causing compatibility problems have been changed as shown in *2.4 Handling compatibility check instructions*, rebuild the project as shown in *2.3(2) Building*. When the following dialog box is displayed for a successful build, click Yes, and then download the load module.


Confirmation Request

OK to download module: C:¥Workspace_Evaluation_RX¥RX_test_H8¥RX_test¥Debug¥RX_test.abs

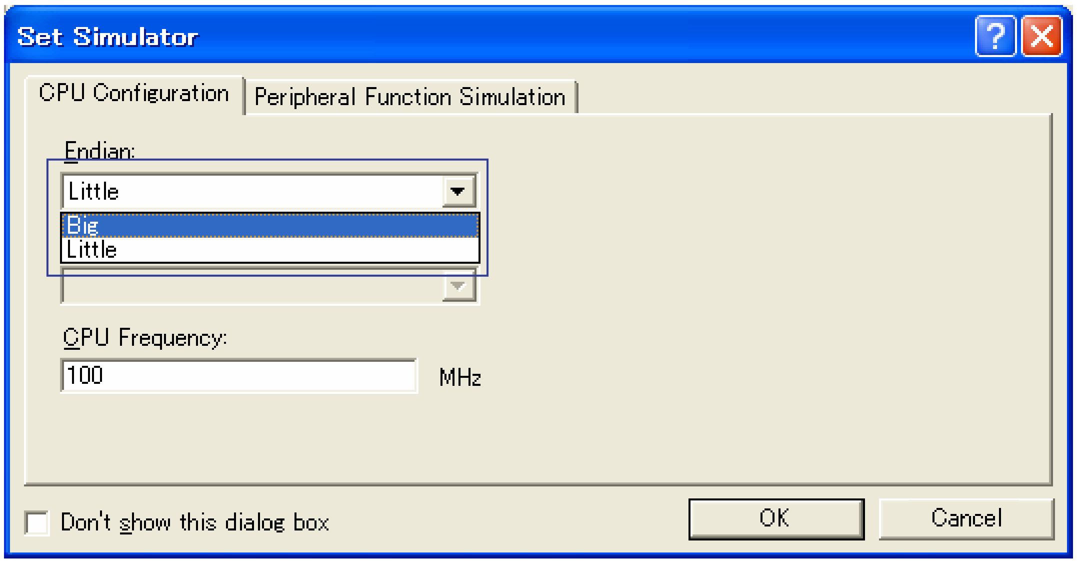
Figure 1-20

2.6 Running the simulator

Execute the rebuilt load module in the simulator.

(1) Setting session refresh

Since the endian option is used to change the endian from little to big, the endian also needs to be changed to big for the simulator. In HEW, choose File and then Session, and then change the endian to big in the displayed dialog box for simulator settings.


Set Simulator

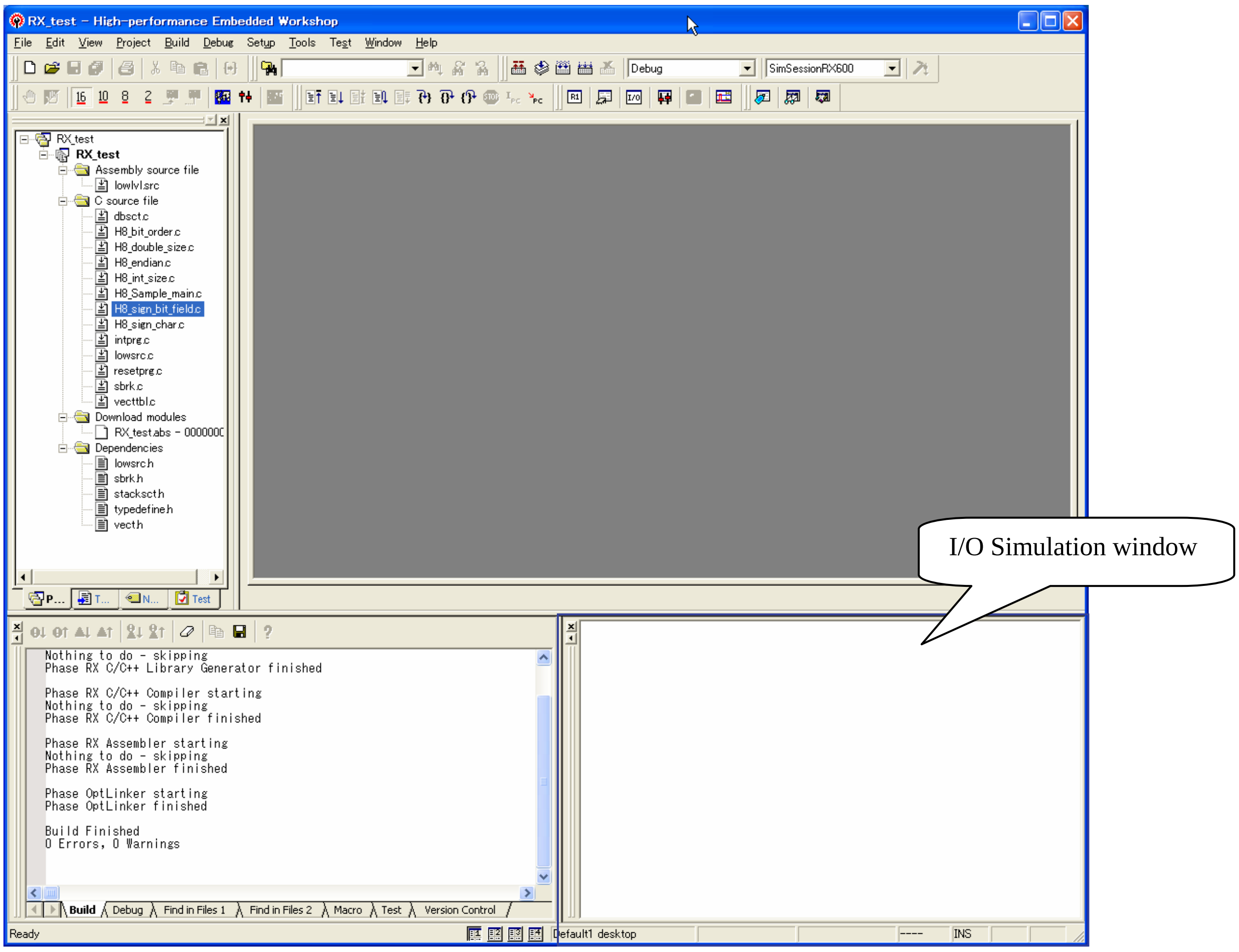
Endian: Little

CPU Frequency: 100 MHz

Figure 1-21

(2) Setting up I/O simulation

The program outputs the execution results to the standard output. The I/O Simulation window needs to be enabled to display the standard output. From HEW, choose View, then CPU, and then I/O simulation to display the I/O Simulation window.

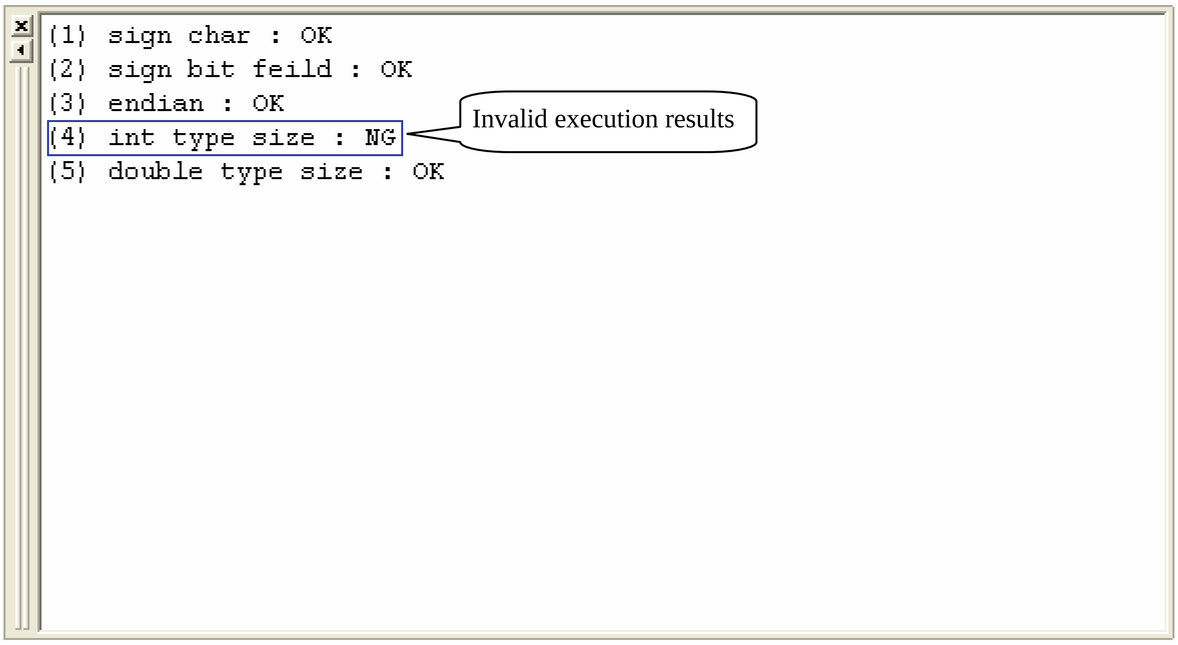


```
Nothing to do - skipping
Phase RX C/C++ Library Generator finished

Phase RX C/C++ Compiler starting
Nothing to do - skipping
Phase RX C/C++ Compiler finished

Phase RX Assembler starting
Nothing to do - skipping
Phase RX Assembler finished

Phase OptLinker starting
Phase OptLinker finished

Build Finished
0 Errors, 0 Warnings
```

Figure 1-22

(3) Running the simulator

From HEW, choose Debug and then Run after reset to run the program in the simulator, and display the program standard output in the I/O Simulation window. Once the results are displayed, (4) can be checked to see whether the value is invalid.



```
(1) sign char : OK
(2) sign bit feild : OK
(3) endian : OK
(4) int type size : NG
(5) double type size : OK
```

2.7 Handling invalid execution results

2.7.1 Specifying the size of int type variables

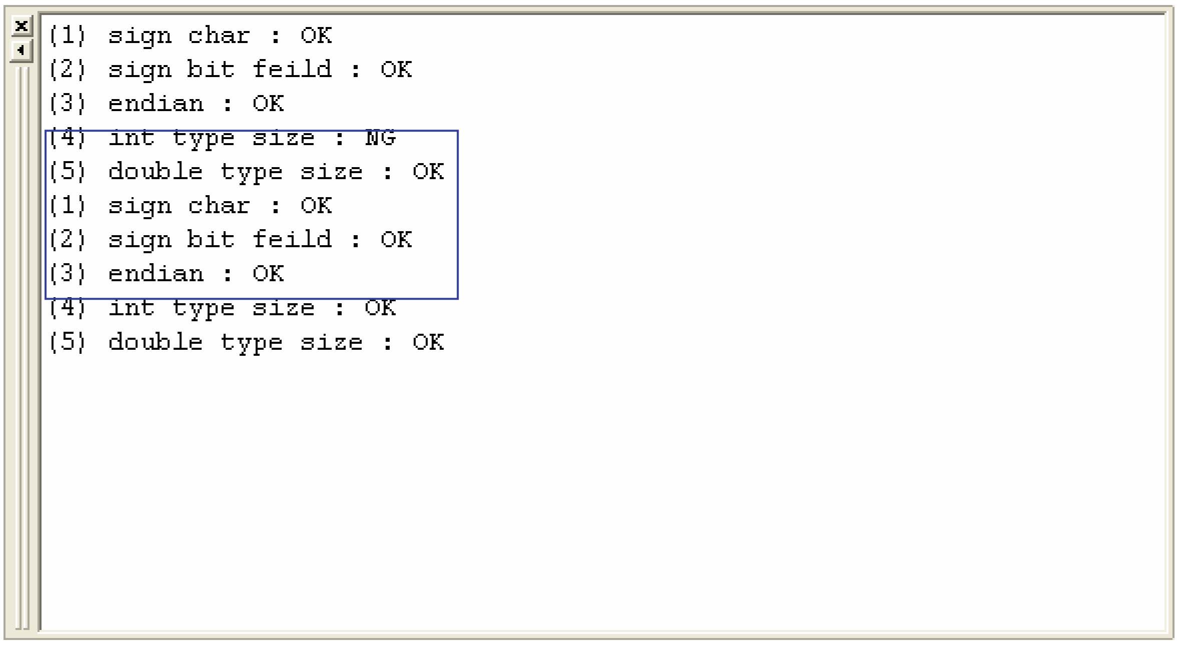
With H8-family compilers, the size of the int type is 2 bytes, whereas with RX-family compilers, the size of the int type is 4 bytes. Since “H8_int_size.c” in the sample program contains code based on the requirement that the size of the int type is 2 bytes, the results operation of operation will differ from H8.

“H8_int_size.c” in the sample program

```
Source code

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

void int_size(void)
{
    UN u;
    u.data = 0x7f6f5f4f;

    printf("(4) int type size : ");

    if (u.s.dataH == 0x7f6f && u.s.dataL == 0x5f4f) {
        printf("OK¥n");
    } else {
        printf("NG¥n");
    }
}
```

To migrate to RX programs 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 *compiler users manual*. Also, change the options specified in the created RX project.

After changing the options and code, perform rebuild as shown in *2.3(2) Building*, and run the simulator as shown in *2.6 Running the simulator* to get the following execution results, and complete migration to the RX project.



```
(1) sign char : OK
(2) sign bit feild : OK
(3) endian : OK
(4) int type size : NG
(5) double type size : OK
(1) sign char : OK
(2) sign bit feild : OK
(3) endian : OK
(4) int type size : OK
(5) double type size : OK
```

Web site and support <website and support>

Web site for Renesas Technology

http://japan.renesas.com/

Contact information

http://japan.renesas.com/inquiry

csc@renesas.com

Revision history<revision history,rh>

Rev.	Date issued	Contents changed	
		Page	Details
1.00	2009.10.1	--	Initial edition