

To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: <http://www.renesas.com>

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (<http://www.renesas.com>)

Send any inquiries to <http://www.renesas.com/inquiry>.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of 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. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. 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 of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may 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.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: “Standard”, “High Quality”, and “Specific”. The recommended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as “Specific” without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as “Specific” or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is “Standard” unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.
 - “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.
 - “High Quality”: Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.
 - “Specific”: Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its 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 be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, 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. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be 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, or if you have any other inquiries.

(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.

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

1. Overview of the H8 sample project	2
2. Migrating the H8 sample project to RX	3
2.1 Creating the RX project.....	3
2.2 Migrating source files for main processing	13
2.3 Building and checking H8 compatibility	16
2.4 Handling compatibility check instructions	19
2.4.1 Specifying sign for the char type.....	19
2.4.2 Specifying sign for bit field members.....	20
2.4.3 Specifying bit field member allocation.....	21
2.4.4 Specifying endian	22
2.4.5 Specifying the size of double type variables	23
2.5 Rebuilding	24
2.6 Running the simulator	24
2.7 Handling invalid execution results.....	27
2.7.1 Specifying the size of int type variables	27
Web site and support <website and support>	28

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.

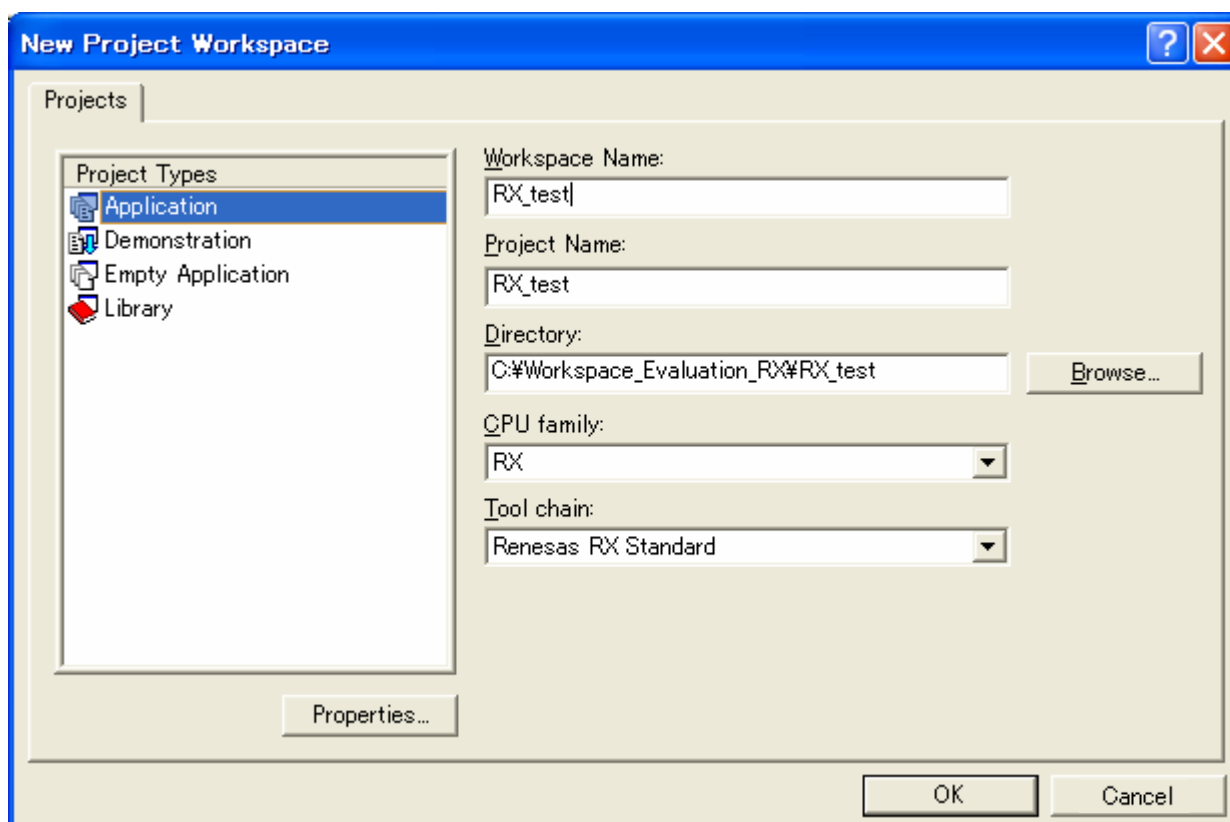


Figure 1-1

(2) Selecting a CPU

Leave this set to the default value, and proceed.

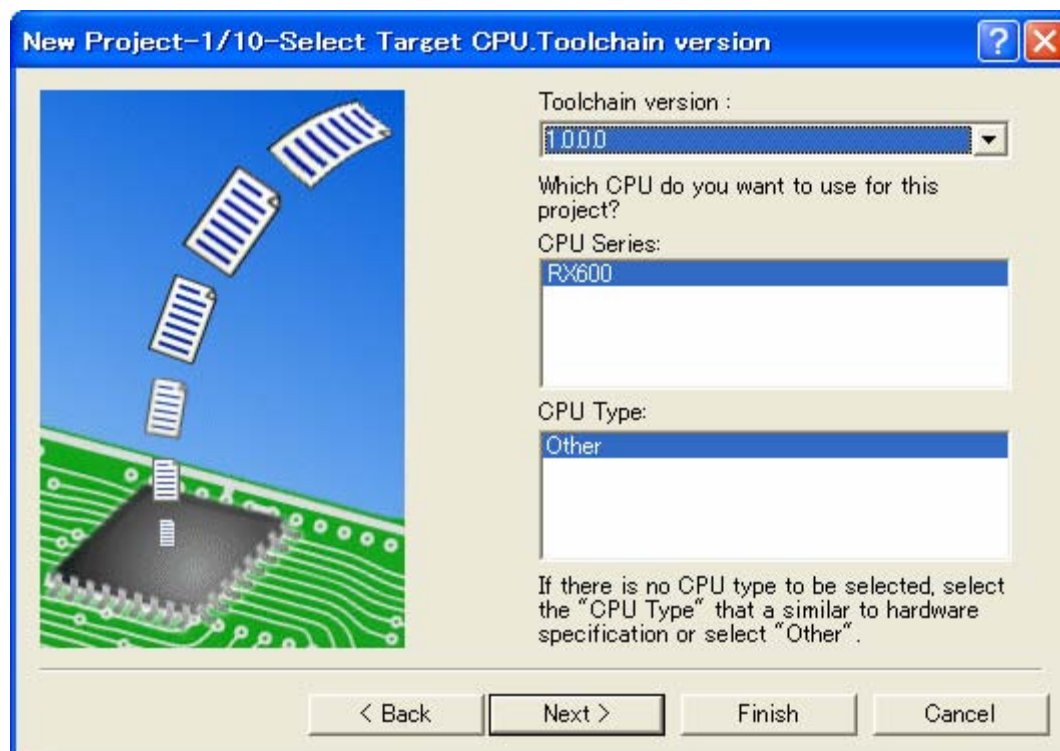


Figure 1-2

(3) Setting options

Leave this set to the default value, and proceed.

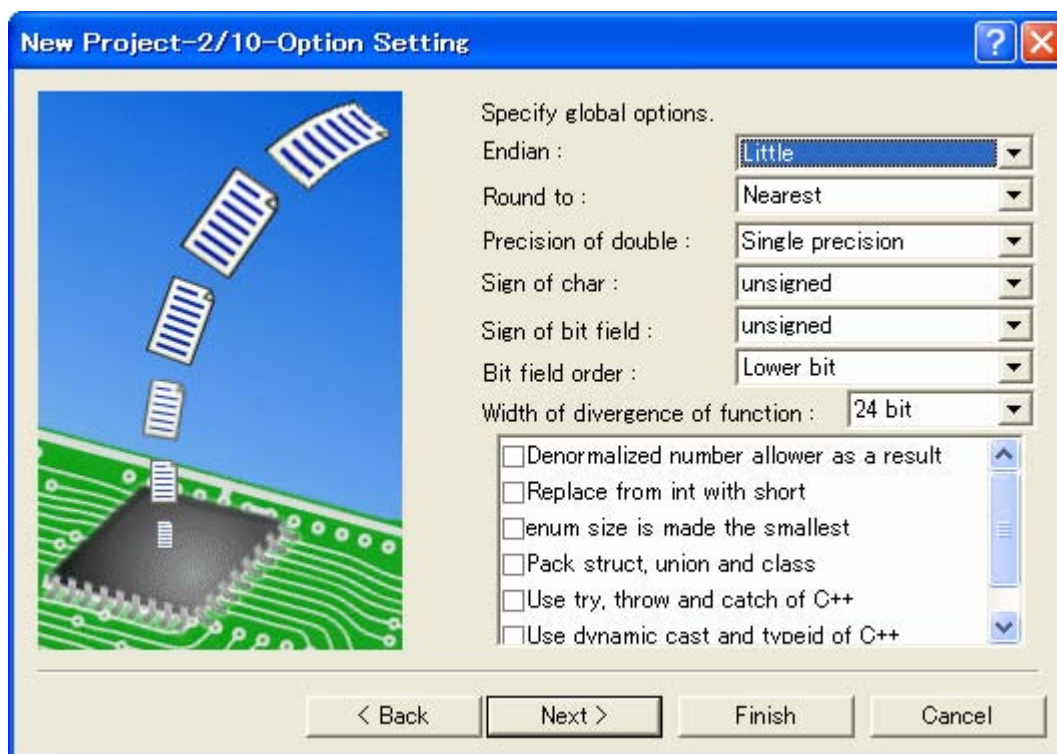


Figure 1-3

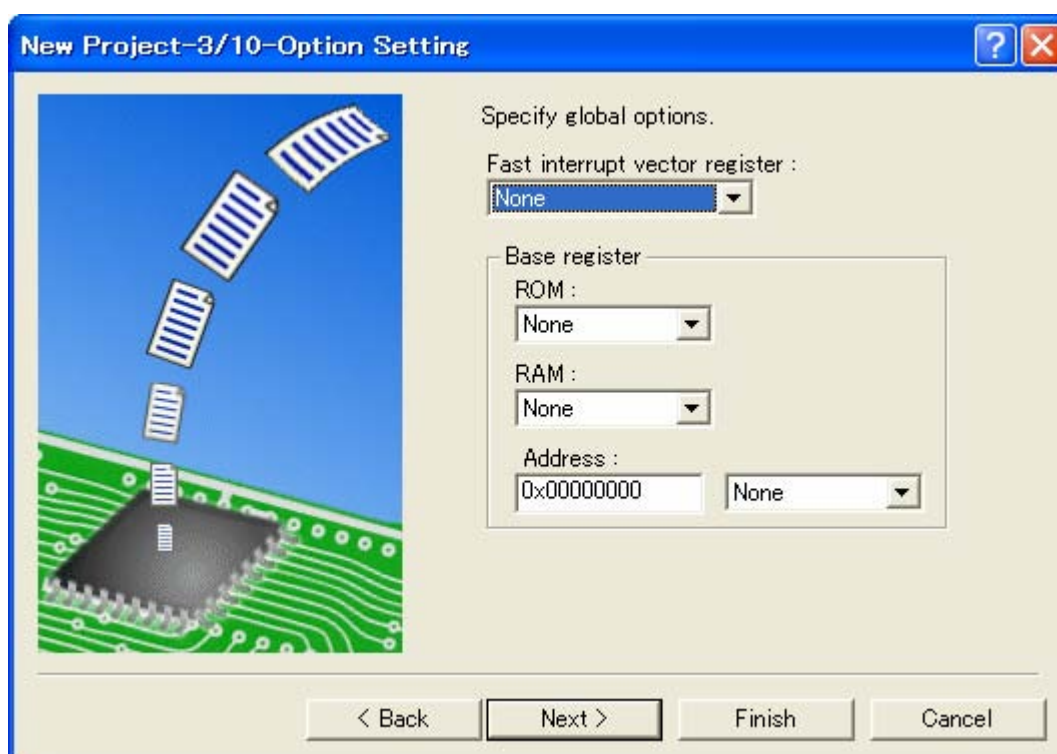


Figure 1-4

(4) Set up generated files

Select "Use I/O library".

Specify "20" for "I/O stream count".

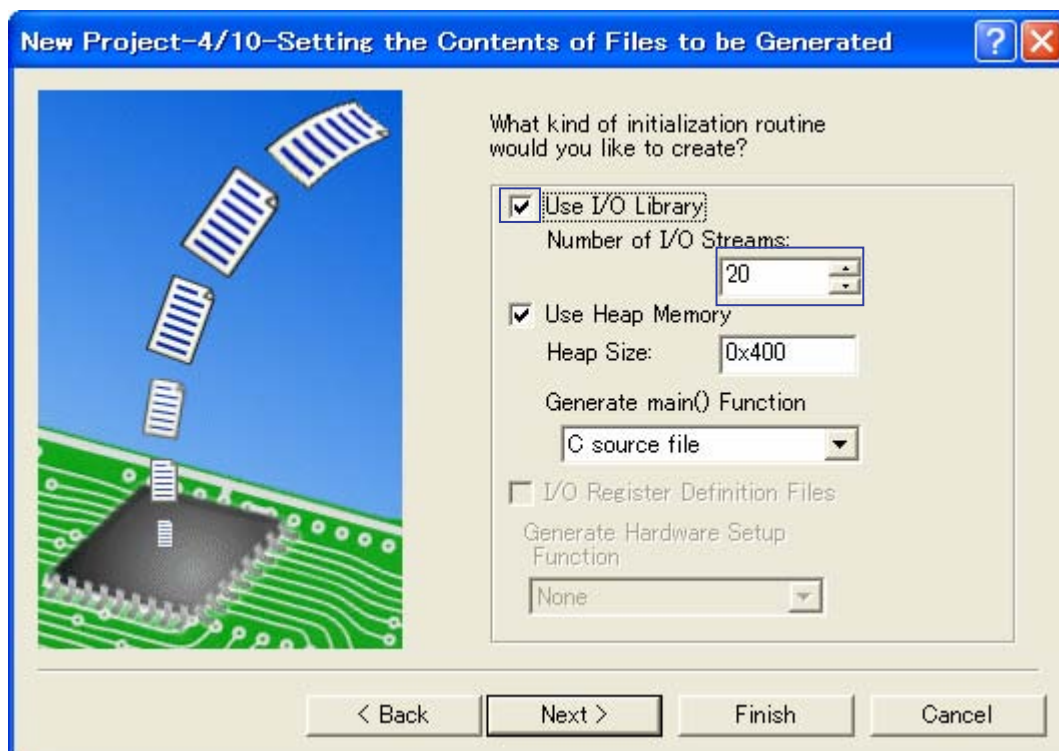


Figure 1-5

(5) Set up the standard library

Leave this set to the default value, and proceed.



Figure 1-6

(6) Set up the stack space

Leave this set to the default value, and proceed.

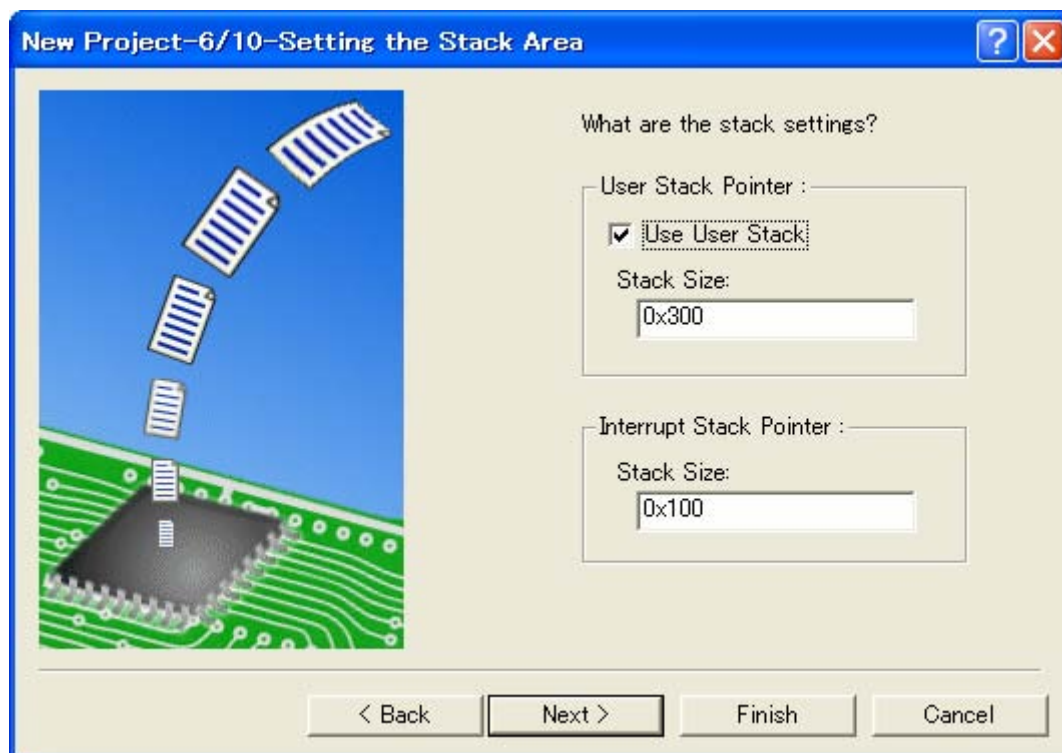


Figure 1-7

(7) Set up vectors

Leave this set to the default value, and proceed.

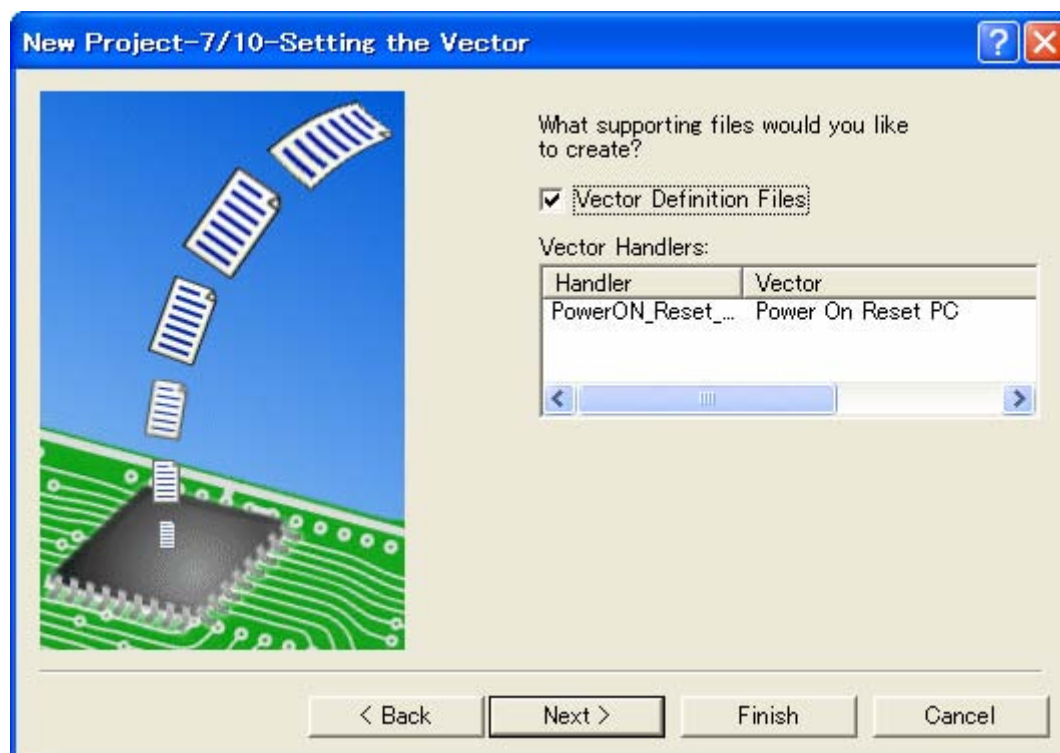


Figure 1-8

(8) Set up the debugger

Select "RX600 Simulator".

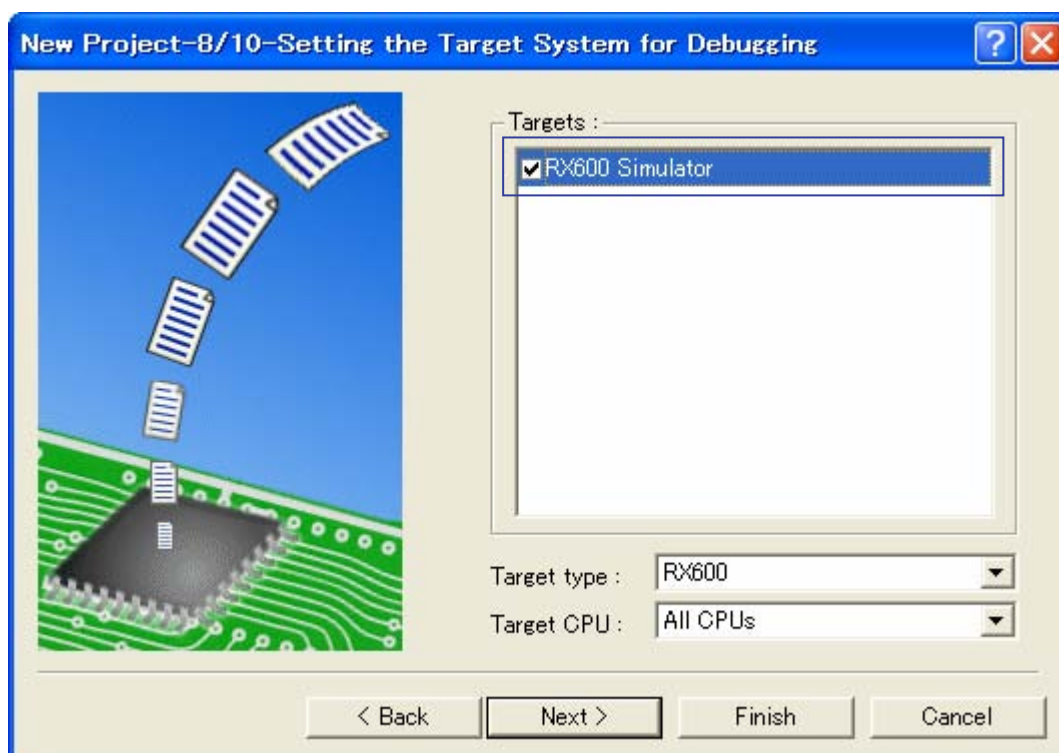


Figure 1-9

(9) Set the debugger options

Select "Initial session".

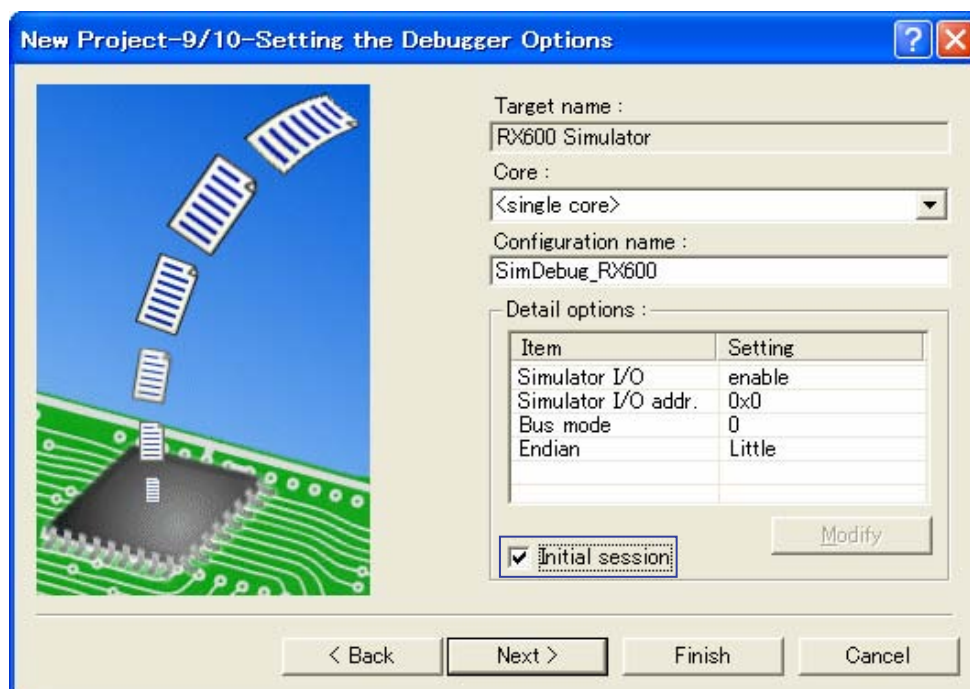


Figure 1-10

(10) Check the generated file names

Click "Finish".

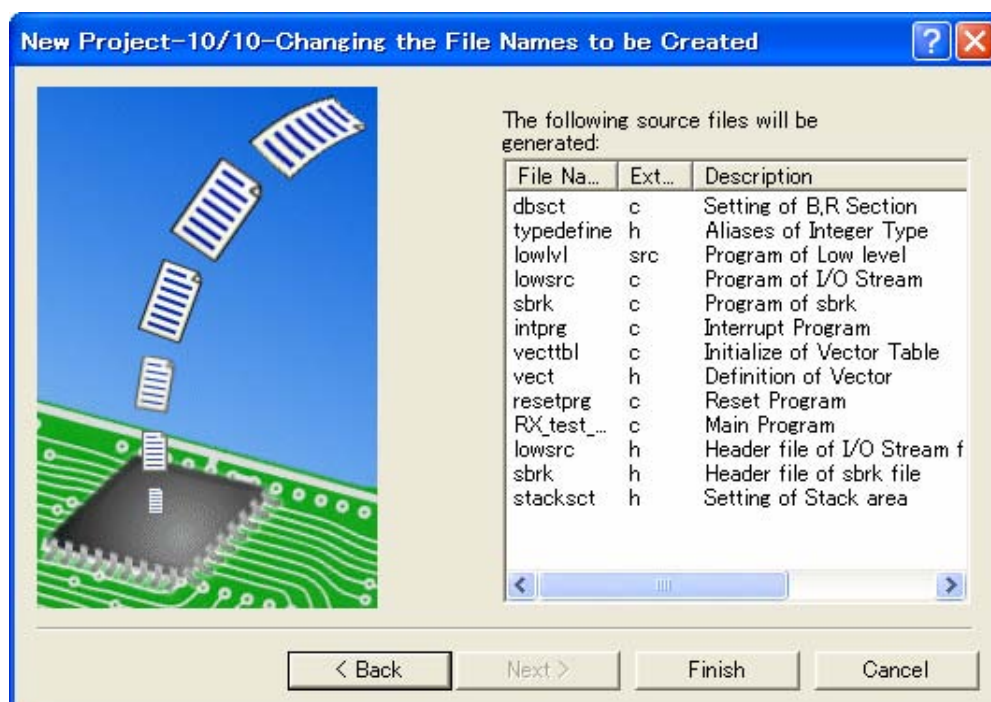


Figure 1-11

(11) Set up the simulator

Click "OK".

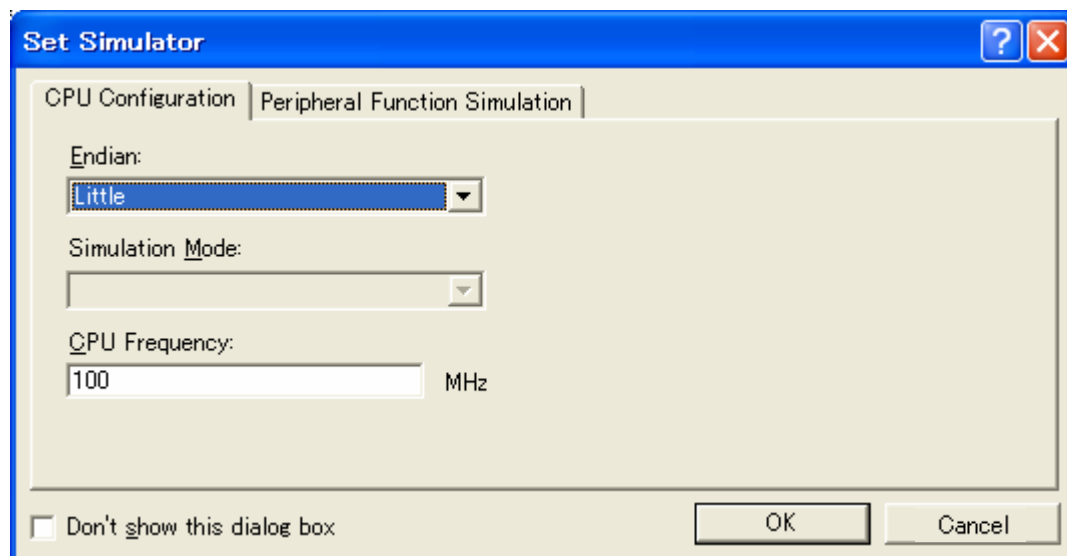


Figure 1-12

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.

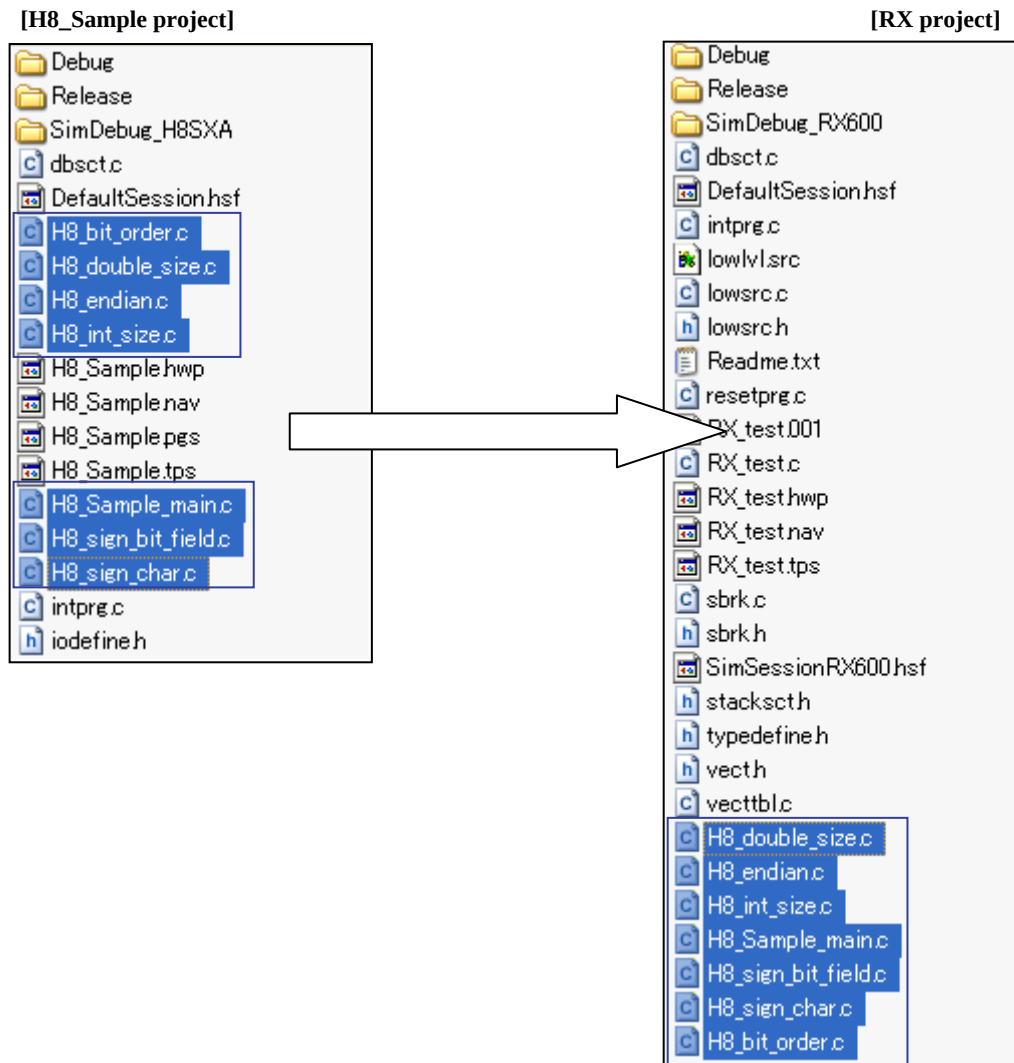


Figure 1-13

(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.

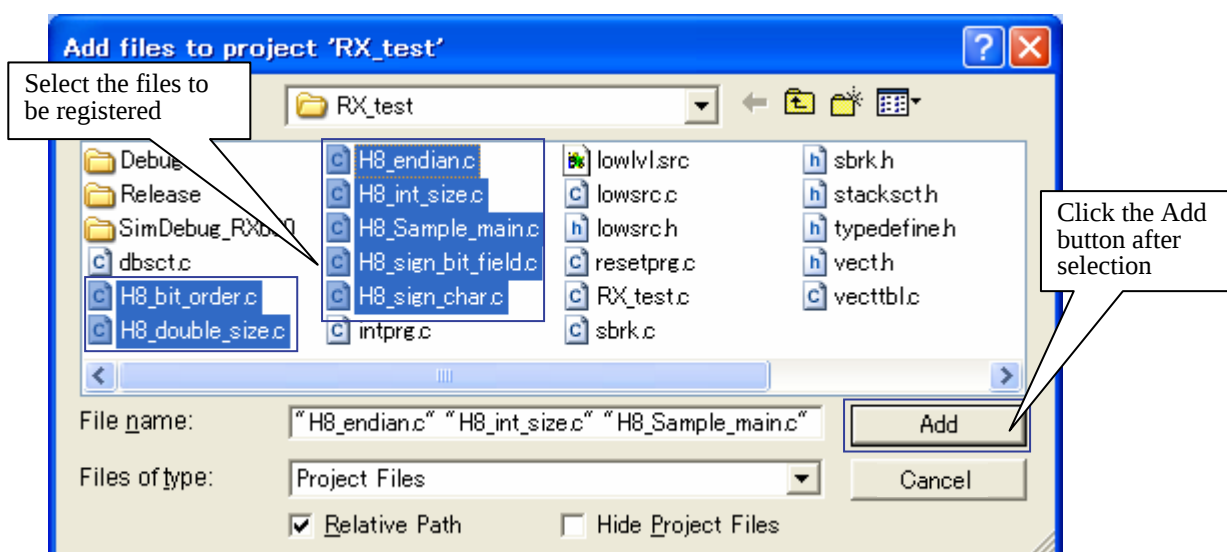


Figure 1-14

(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.

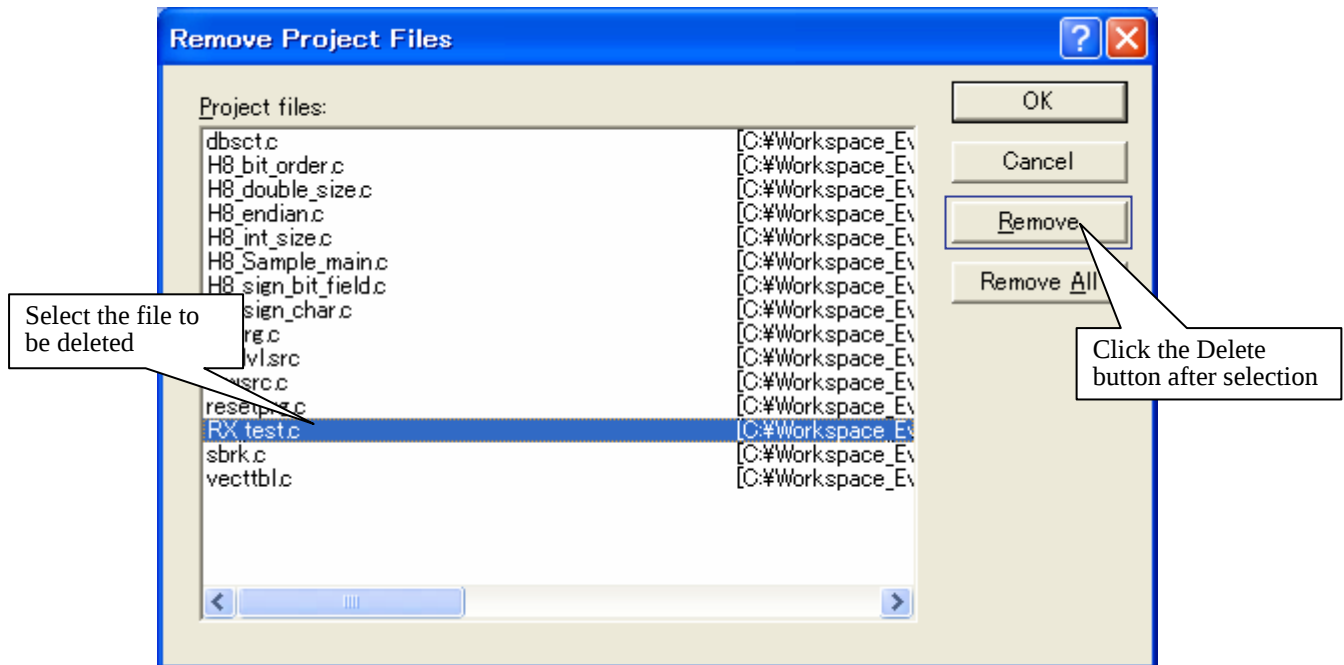


Figure 1-15

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.

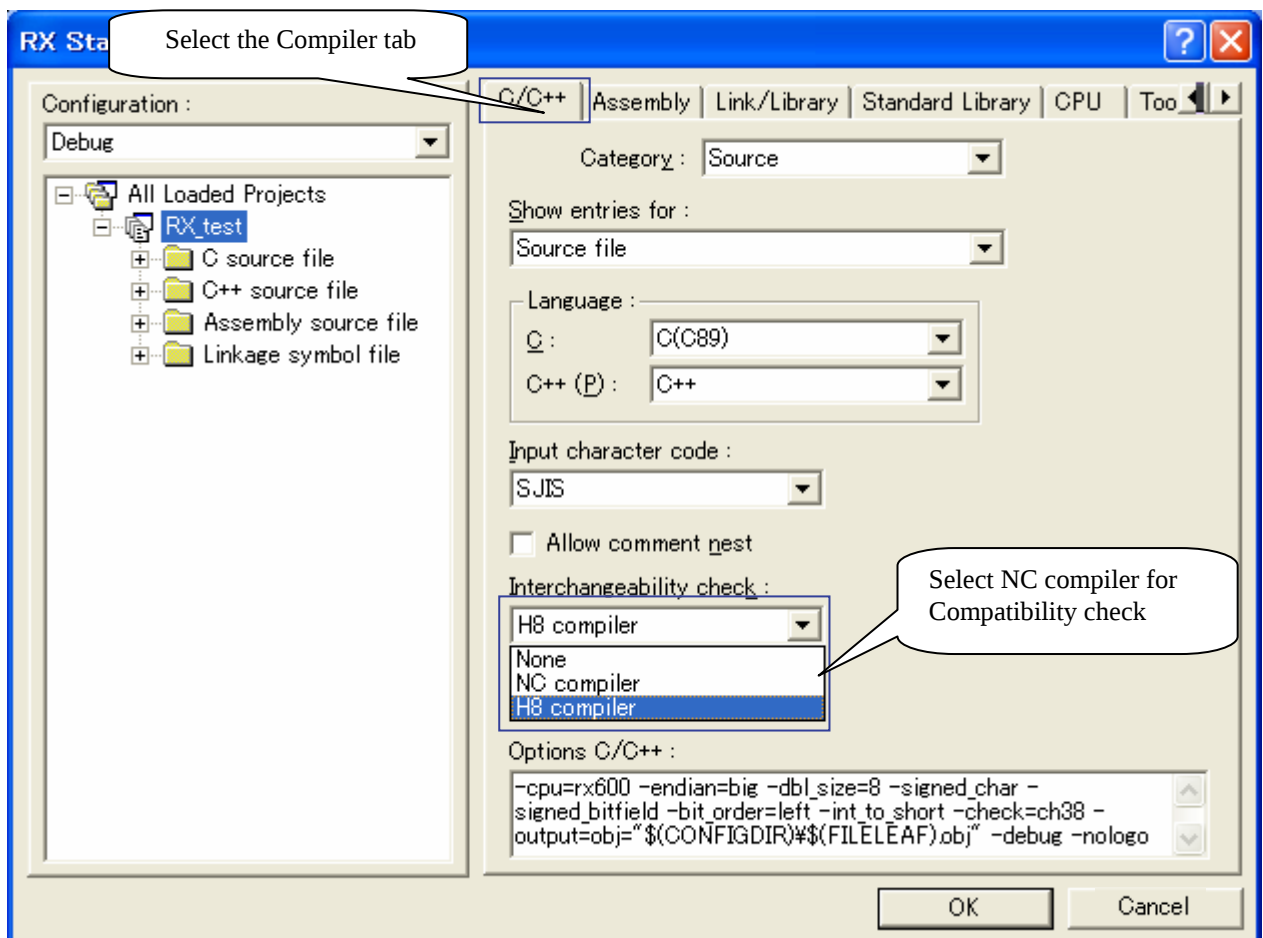


Figure 1-16

(2) Building

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

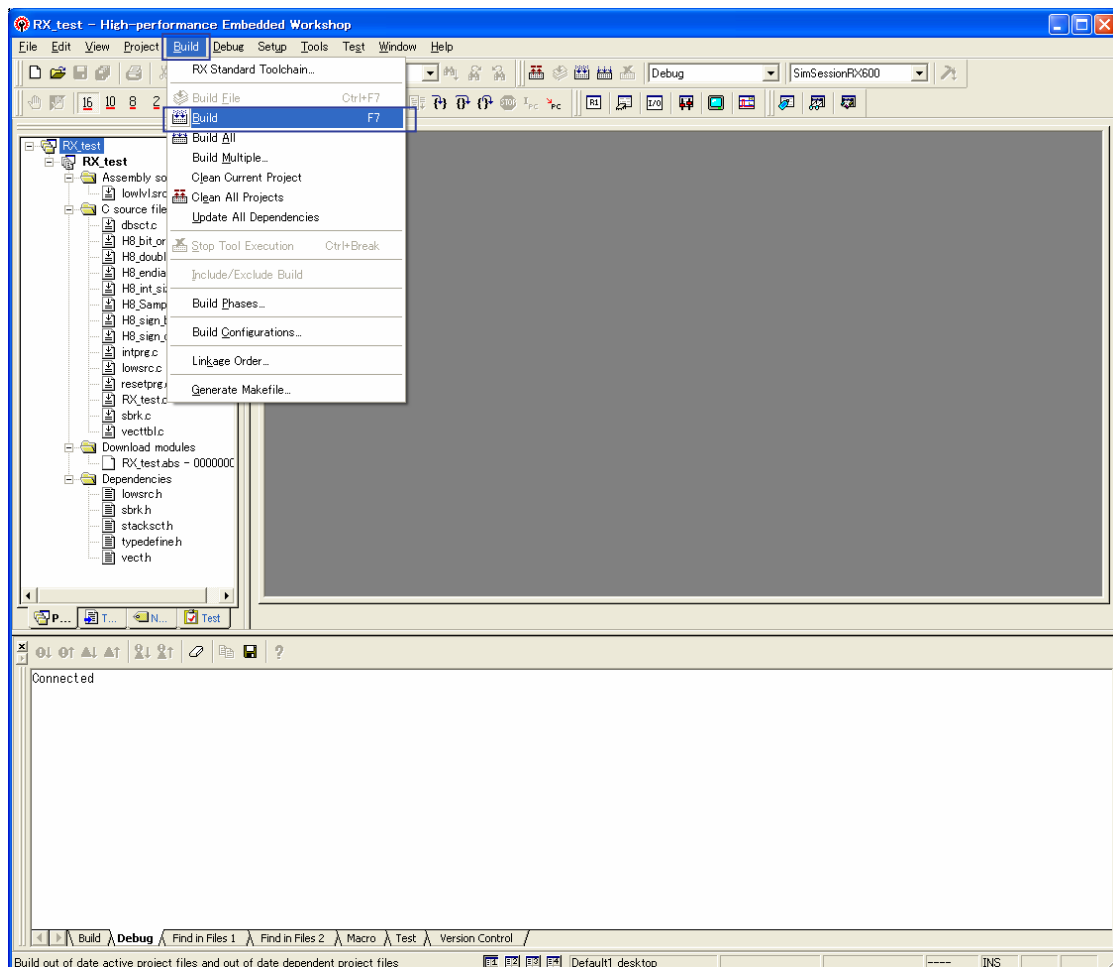


Figure 1-17

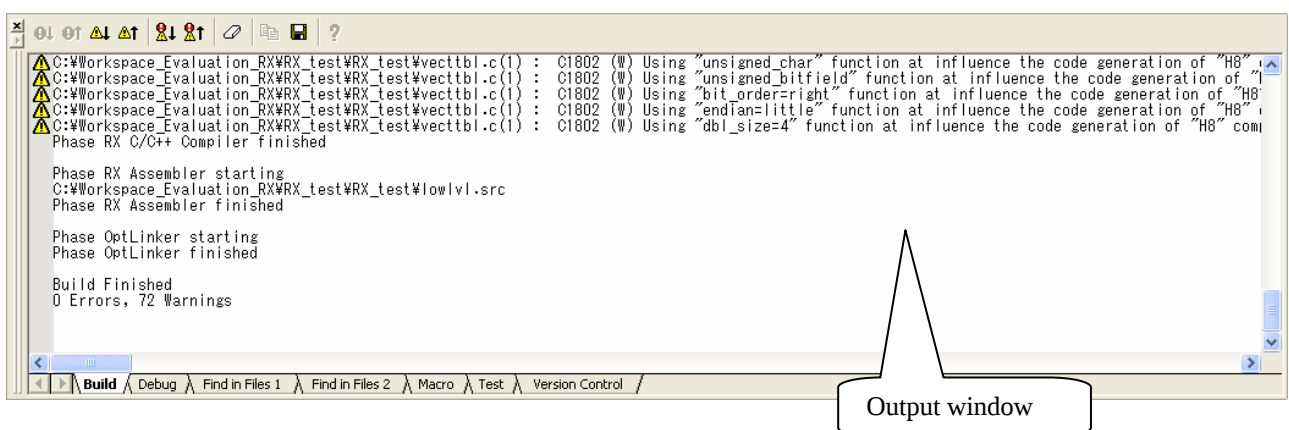


Figure 1-18

(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.

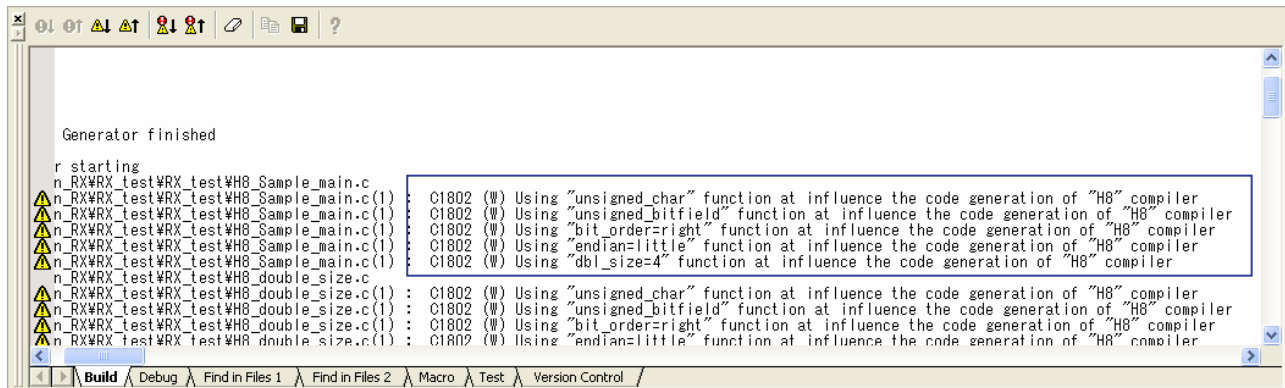


Figure 1-19

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 *compiler 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 : ");

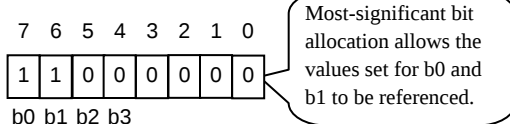
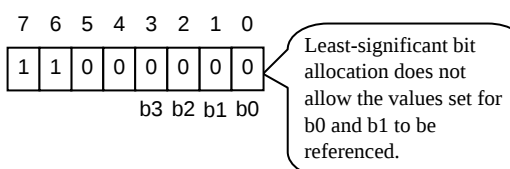
    if (bit.a < 0) {
        printf("OK\n");
    } else {
        printf("NG\n");
    }
}
```

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	H8 bit allocation (left)
<pre> 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"); } } </pre>	H8 bit allocation (left)  Most-significant bit allocation allows the values set for b0 and b1 to be referenced. RX bit allocation (right)  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 to 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 to 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.

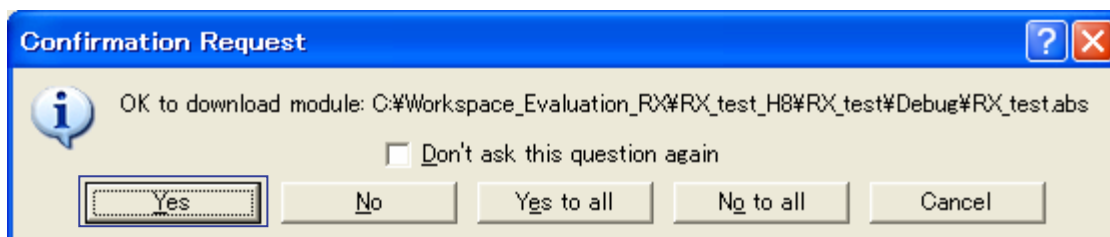


Figure 1-20

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.

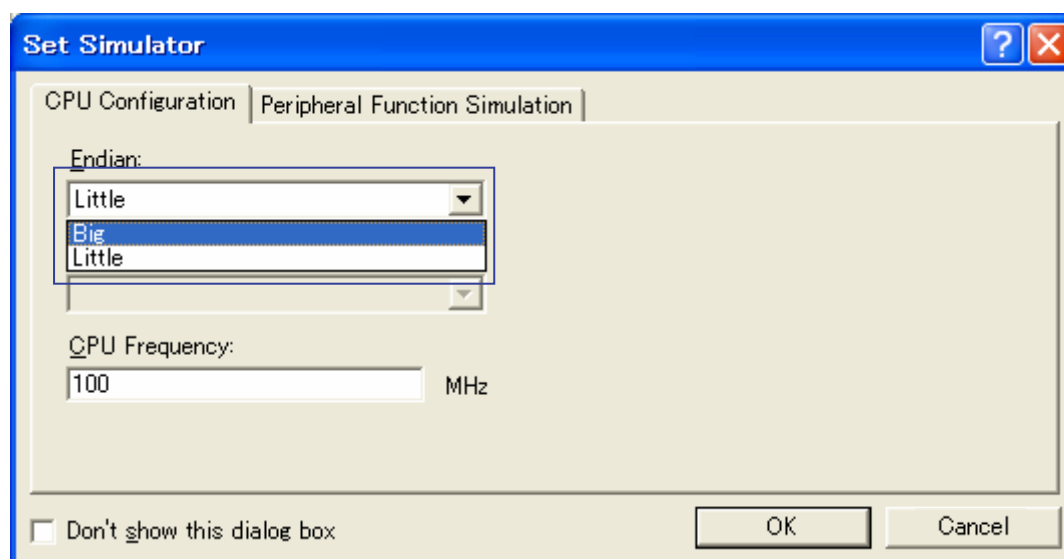


Figure 1-21

(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.

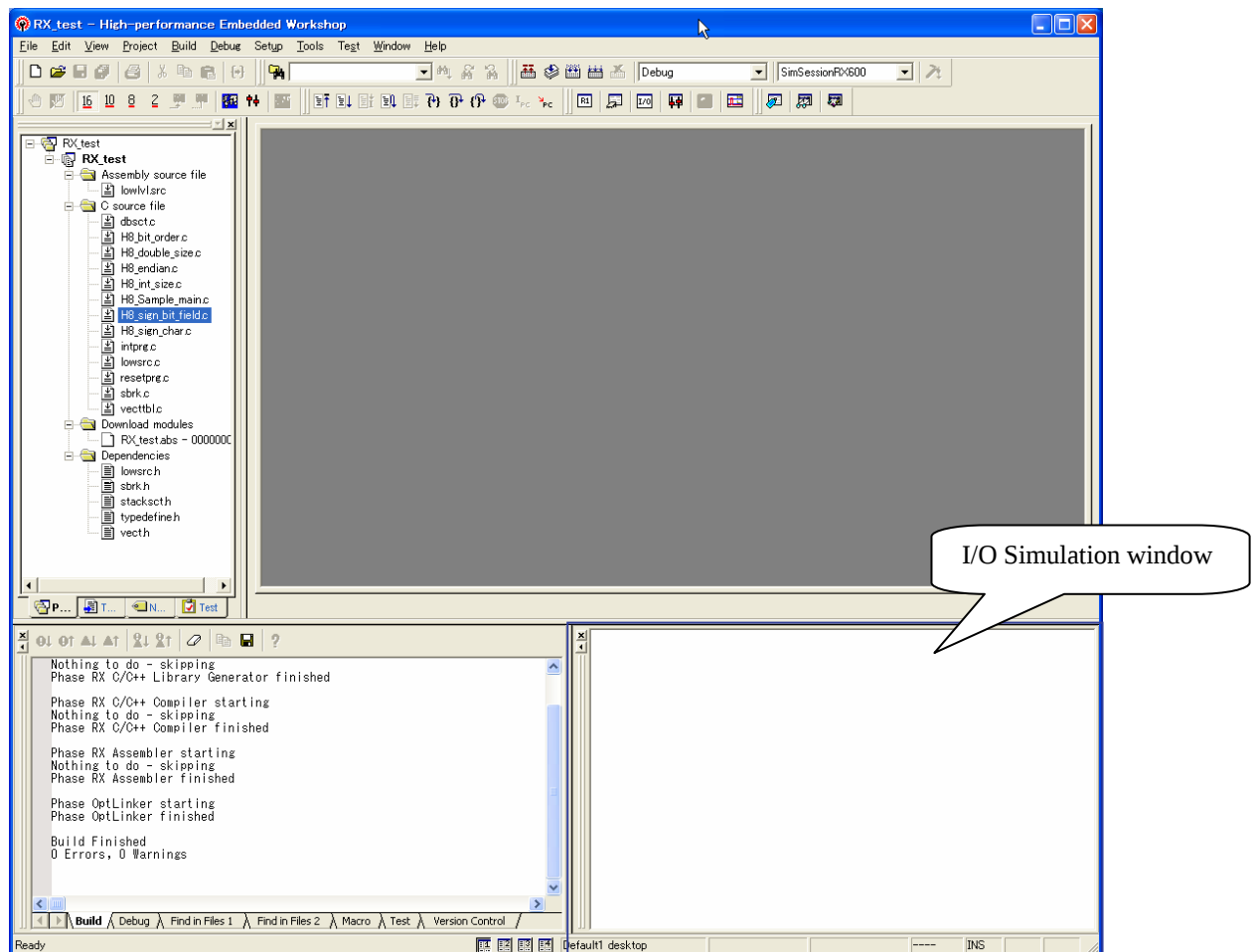
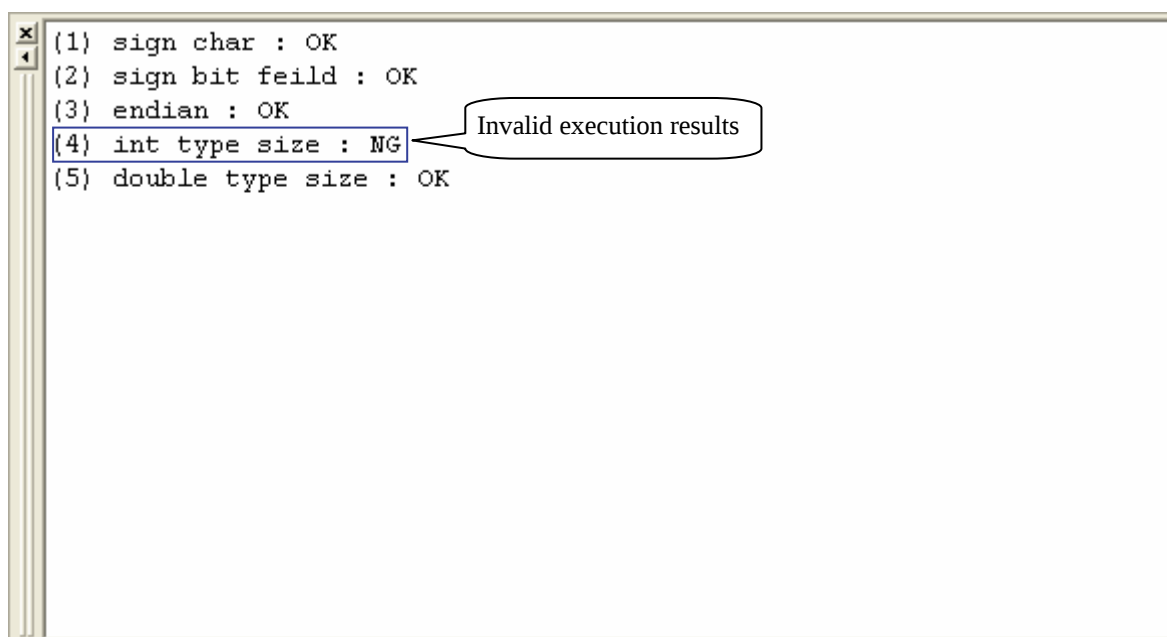


Figure 1-22

(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
  
```

Invalid execution results

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