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.

RENESAS

RX Family C/C++ Compiler Package

Application Notes: Sample Project RX Migration Guide, M16C Edition

This document explains how to migrate the sample project created in M16C to RX.

Table of contents

This edition is a guide for migrating M16C sample projects for which operation can be confirmed in the simulator debugger, to RX.

1. Overview of the M16C sample project

The M16C sample project ‘M16C_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	Depends on integer promotion specifications for the char type	M16C_extended_integer.c	2.7.1
2	Depends on size of int type	M16C_int_size.c	2.7.2
3	Depends on structure member placement	M16C_struct_member.c	2.7.3
4	Depends on size of double type	M16C_double_size.c	2.4.1
5	Depends on pragma BITADDRESS	M16C_pragma_BITADDRESS.c	2.4.2
6	Depends on pragma ROM	M16C_pragma_ROM.c	2.4.3
7	Depends on pragma PARAMETER	M16C_pragma_PARAMETER.c	2.4.4
8	Depends on inline keywords	M16C_inline_keyword1.c M16C_inline_keyword2.c	2.4.5
9	Main function	M16_Sample_main.c	--

2. Migrating the M16C sample project to RX

2.1 Creating the RX project

Create a new RX project workspace for the migration destination of the M16C 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.

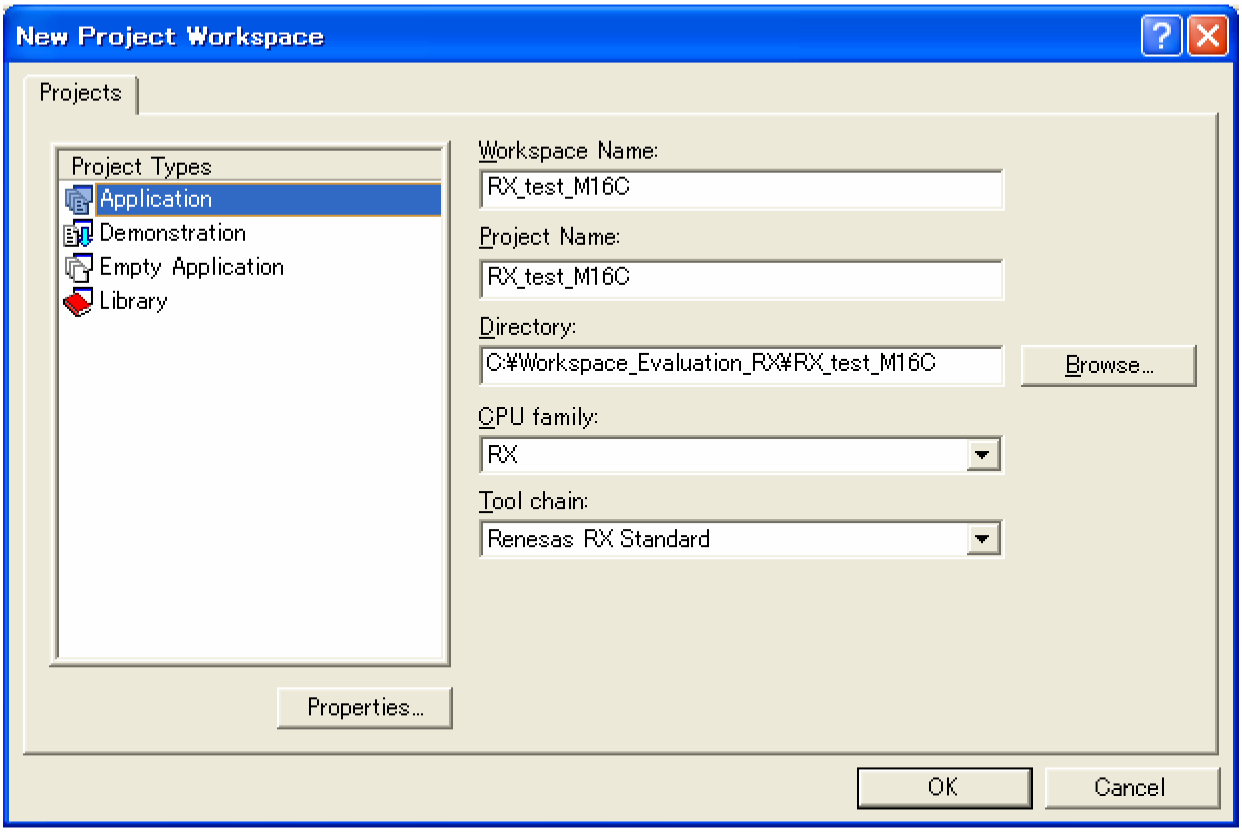


Figure 1-1

(2) Selecting a CPU

Leave this set to the default value, and proceed.

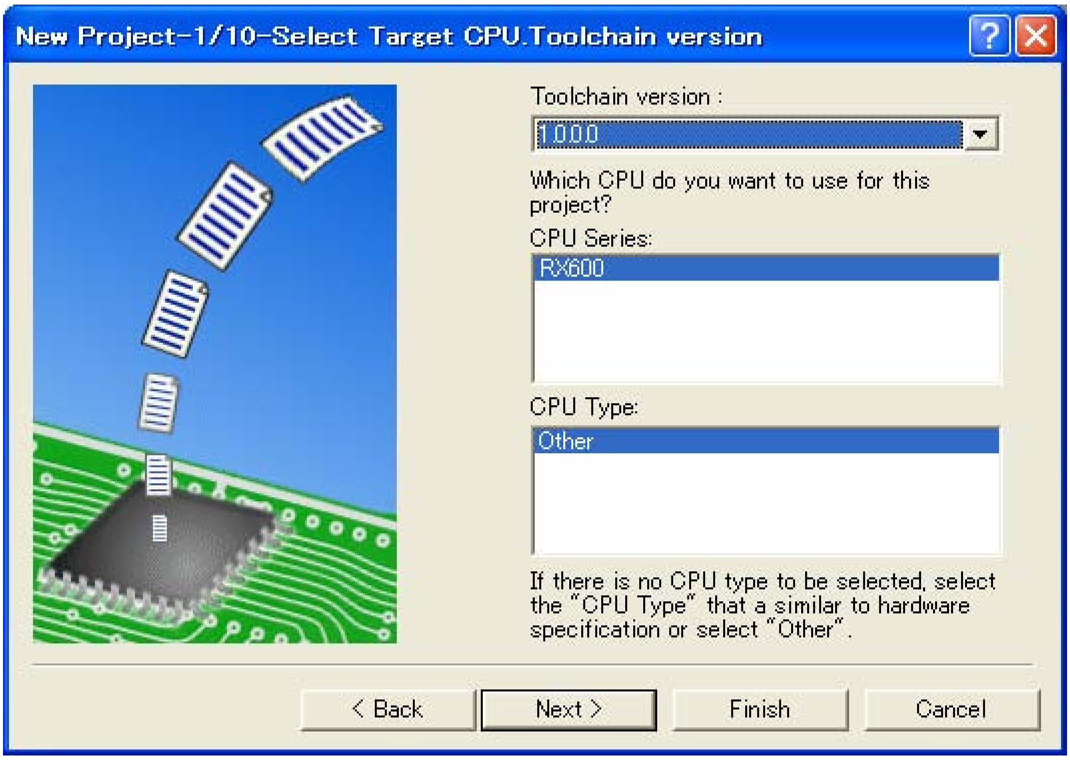


Figure 1-2

(3) Setting options

Leave this set to the default value, and proceed.

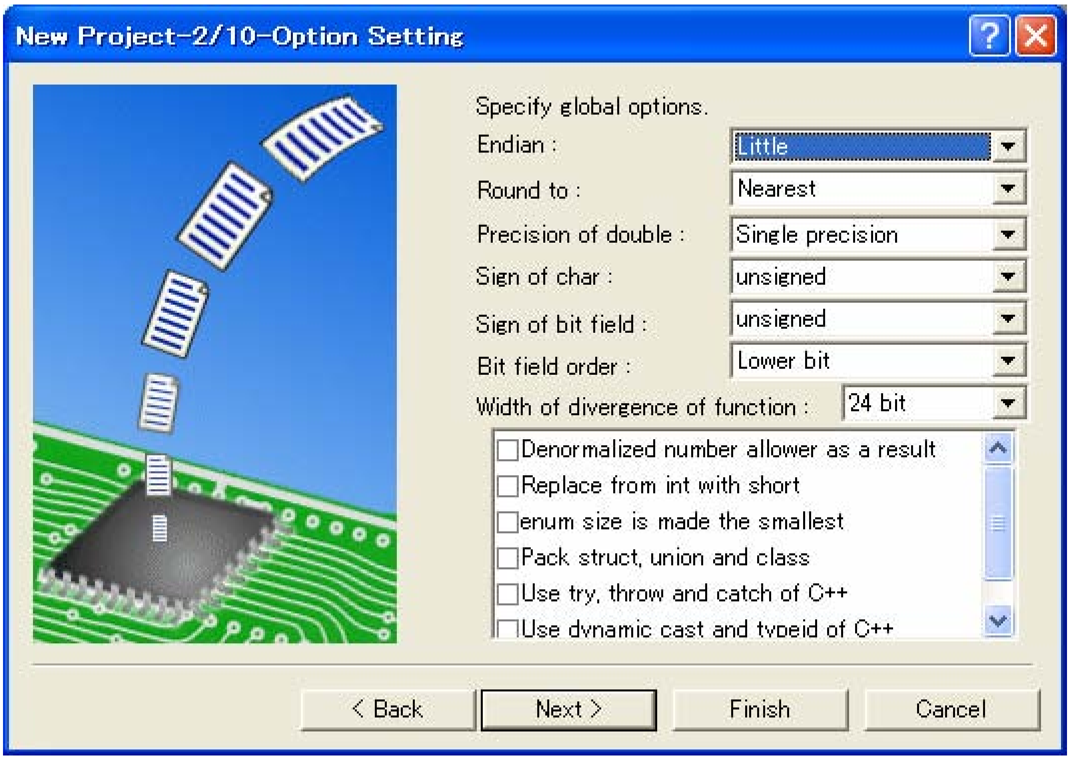


Figure 1-3

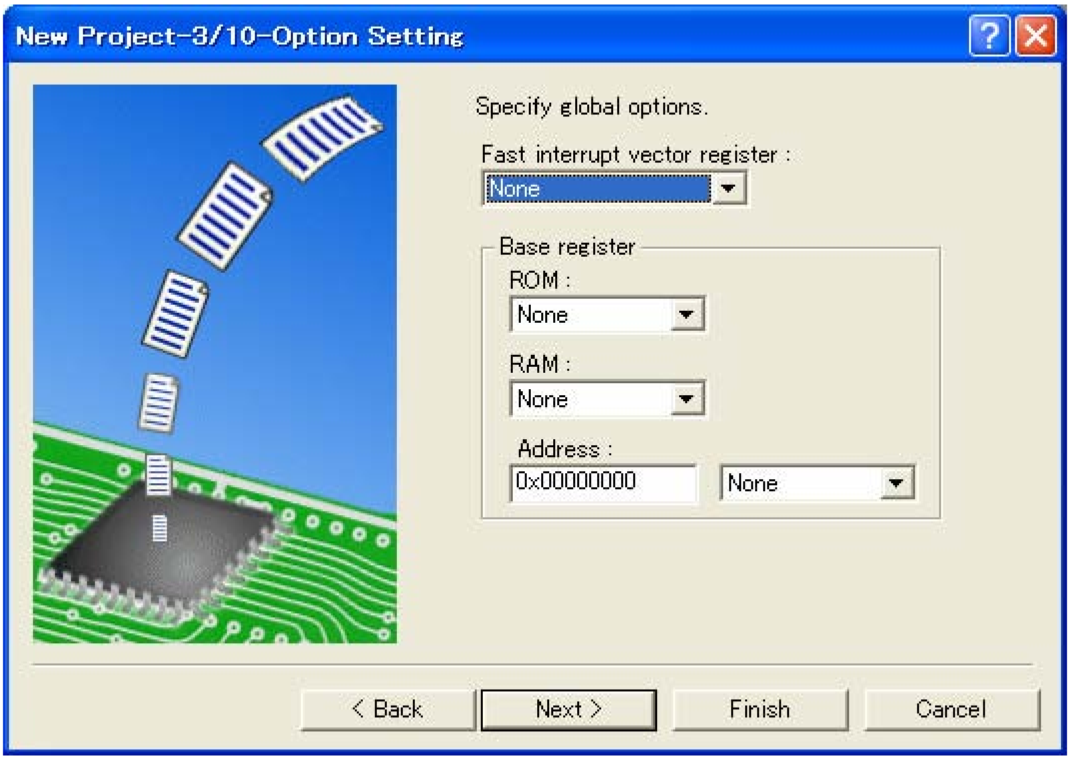


Figure 1-4

(4) Set up generated files

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

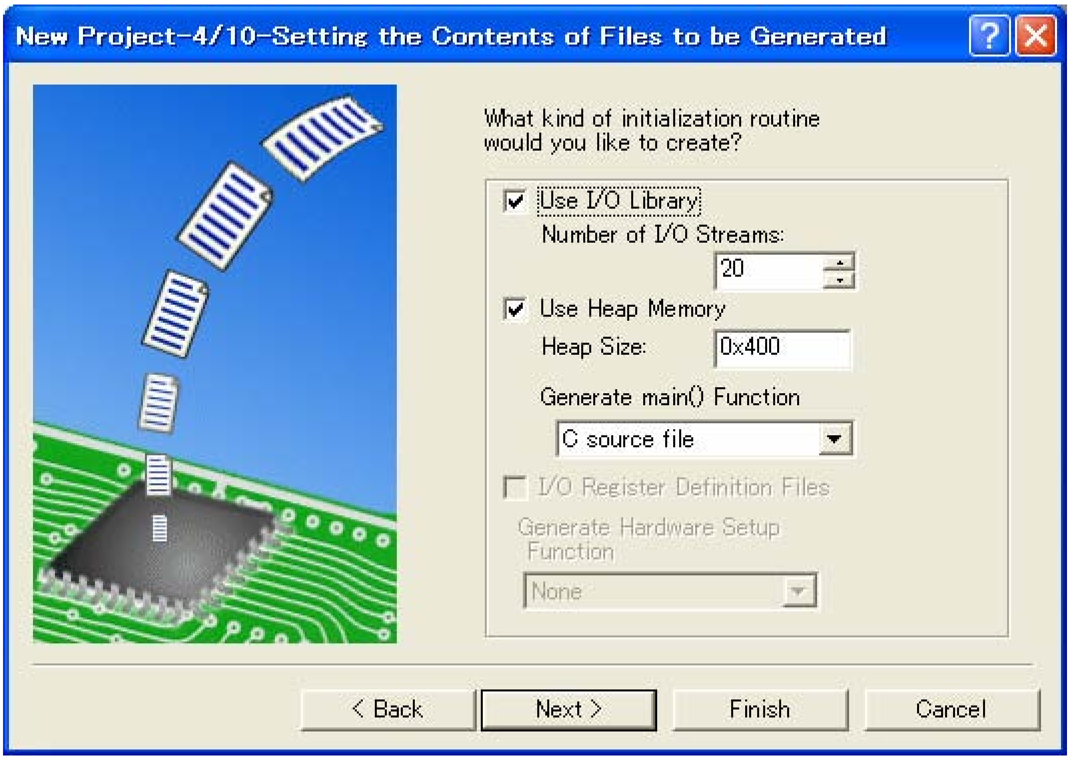


Figure 1-5

(5) Set up the standard library

Leave this set to the default value, and proceed.

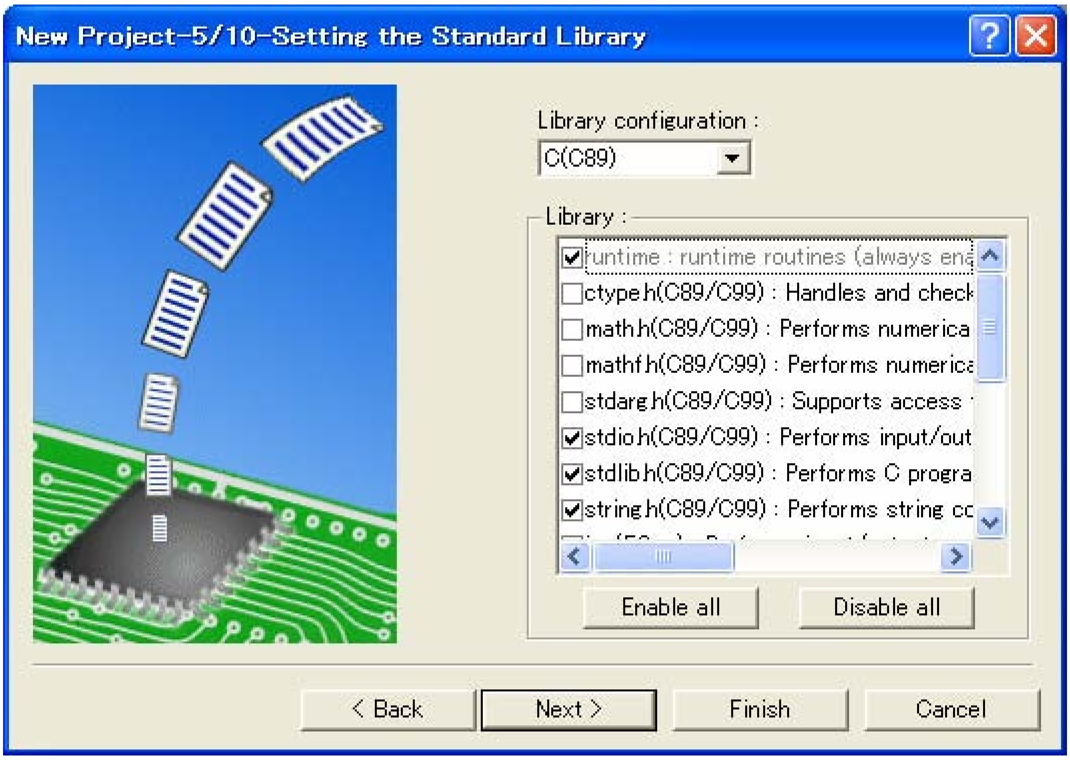


Figure 1-6

(6) Set up the stack space

Leave this set to the default value, and proceed.

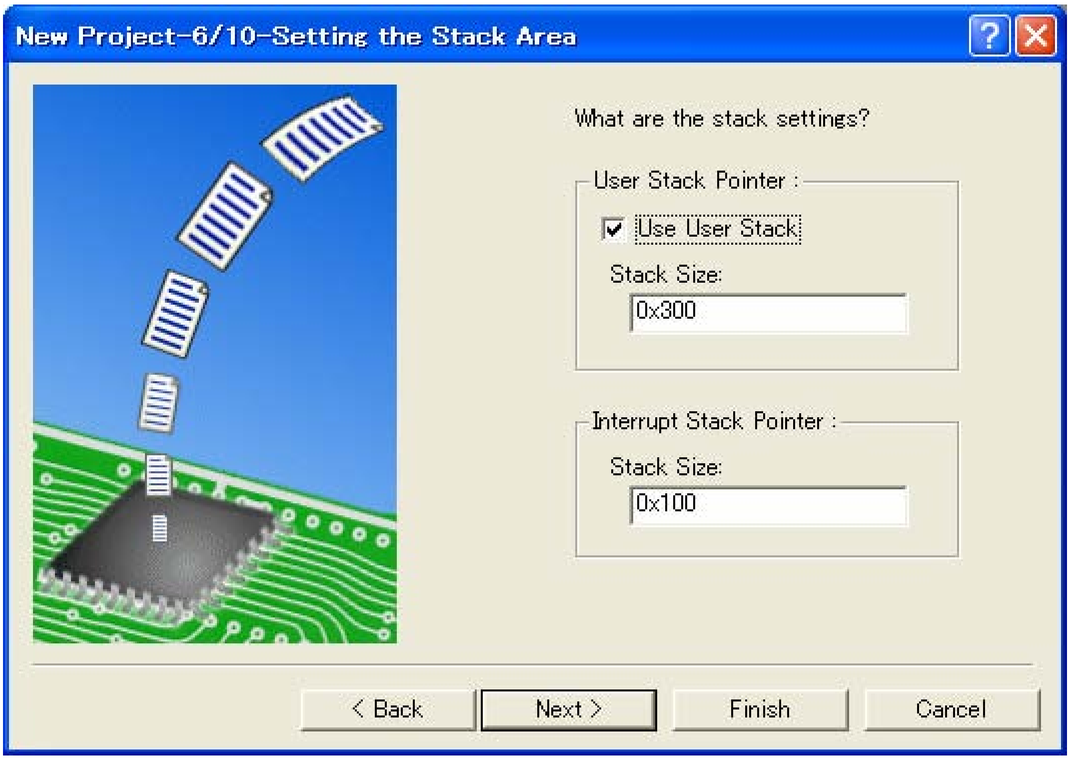


Figure 1-7

(7) Set up vectors

Leave this set to the default value, and proceed.

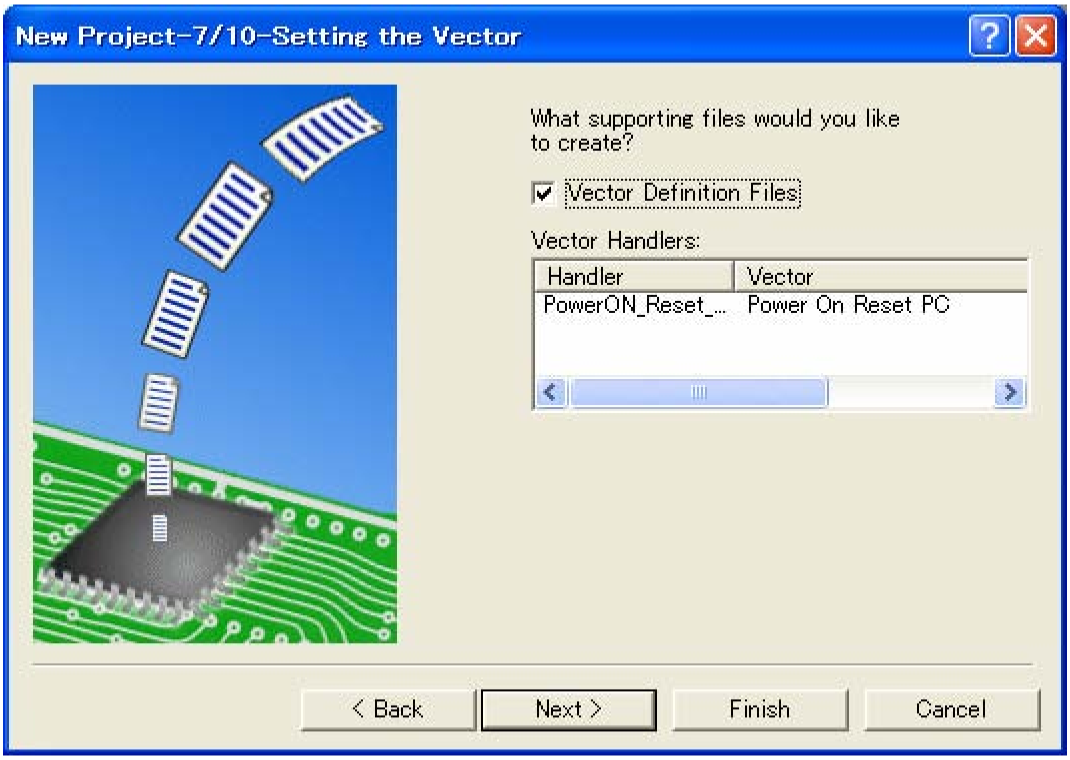


Figure 1-8

(8) Set up the debugger

Select “RX600 Simulator”.

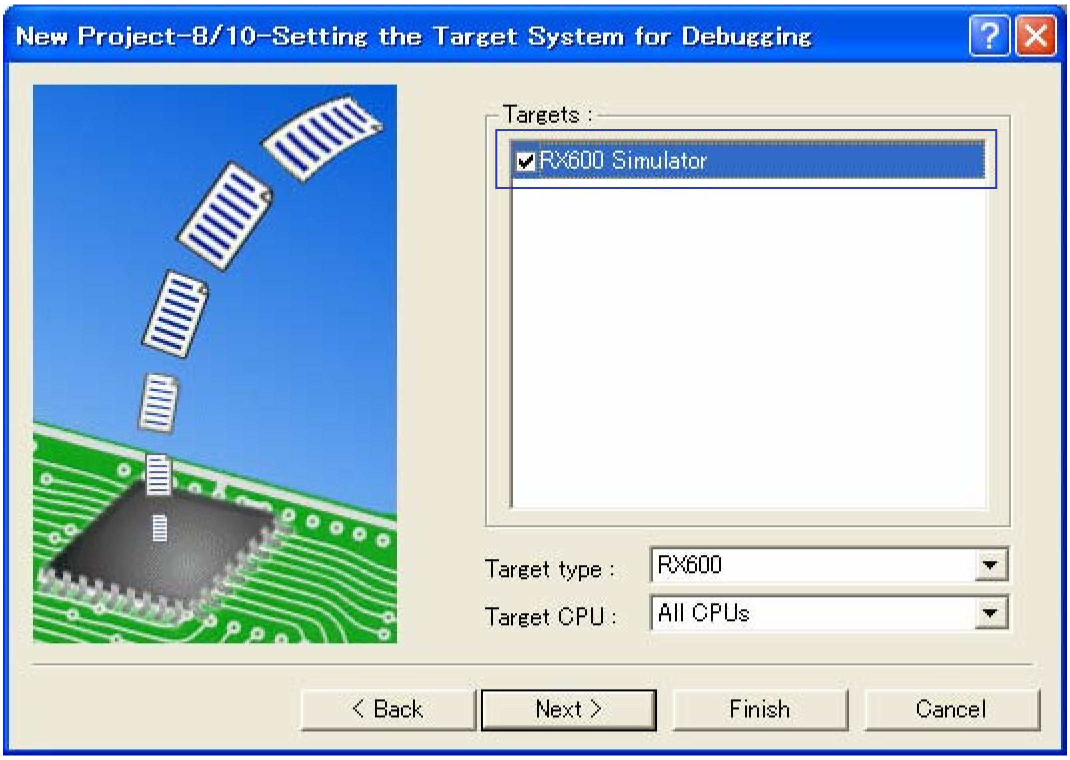


Figure 1-9

(9) Set the debugger options

Select “Initial session”.

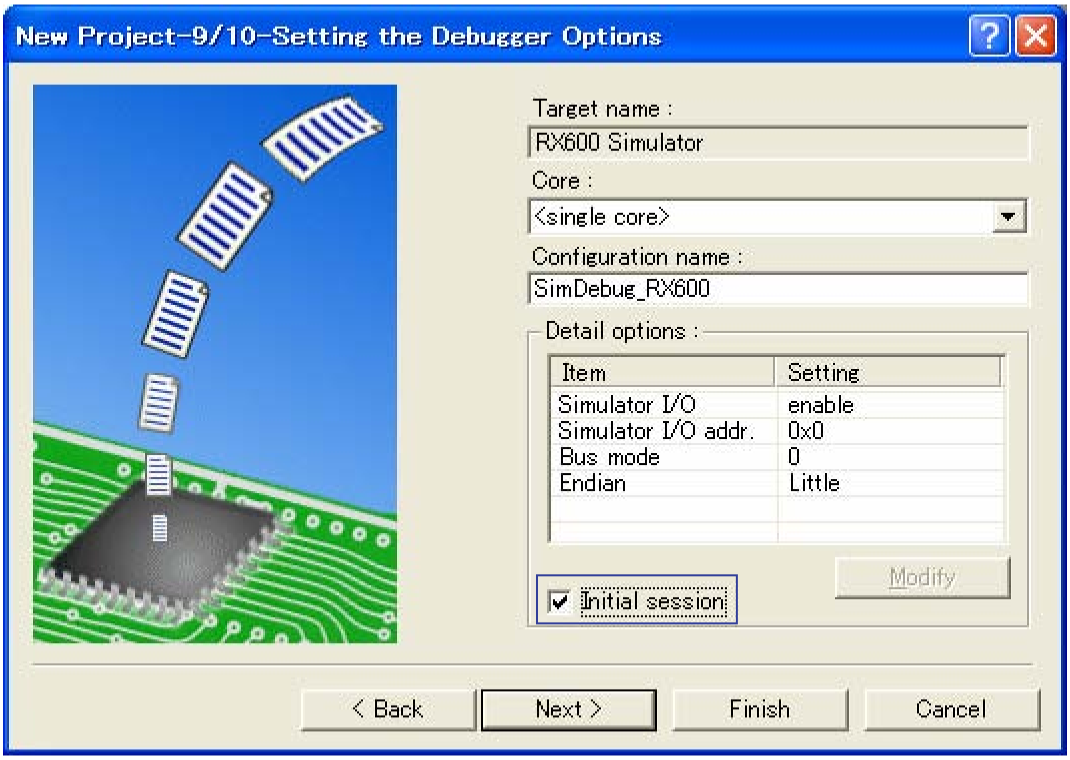


Figure 1-10

(10) Check the generated file names

Click “Finish”.

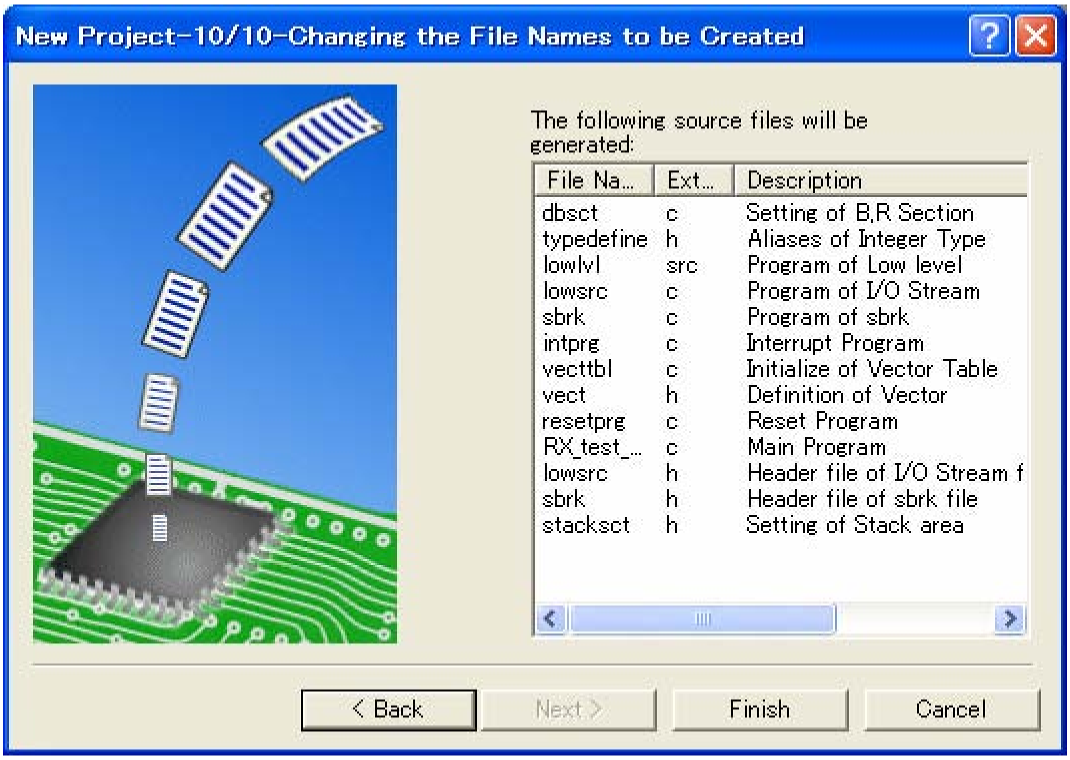


Figure 1-11

(11) Set up the simulator

Click "OK".

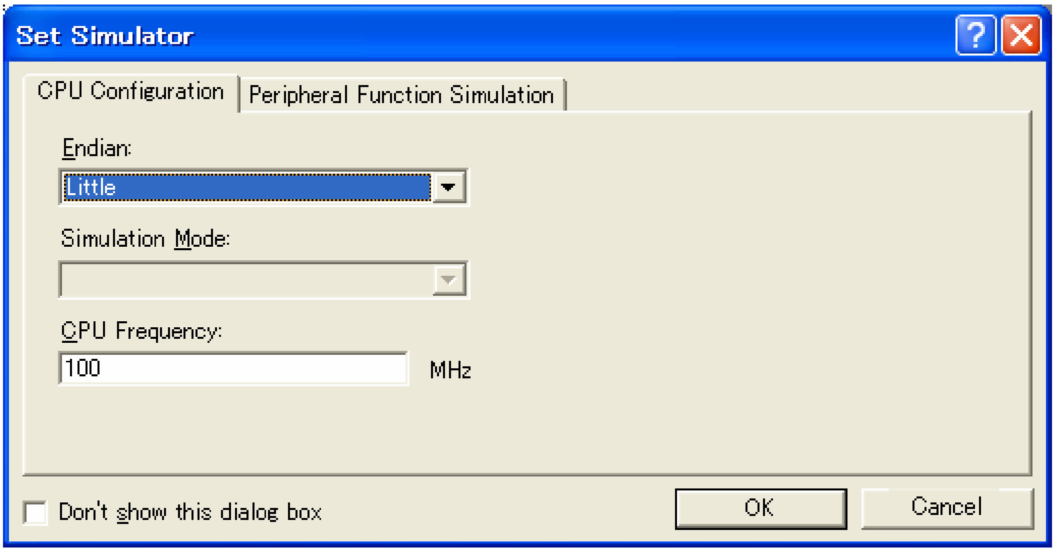


Figure 1-12

2.2 Migrating source files for main processing

Copy, and register with the created RX project, the files comprising main processing for the M16C sample project, as explained in *1. Overview of the M16C sample project.*

(1)Copy files from the M16C sample project folder

Copy to the RX project the 10 files explained in *1. Overview of the M16C 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.

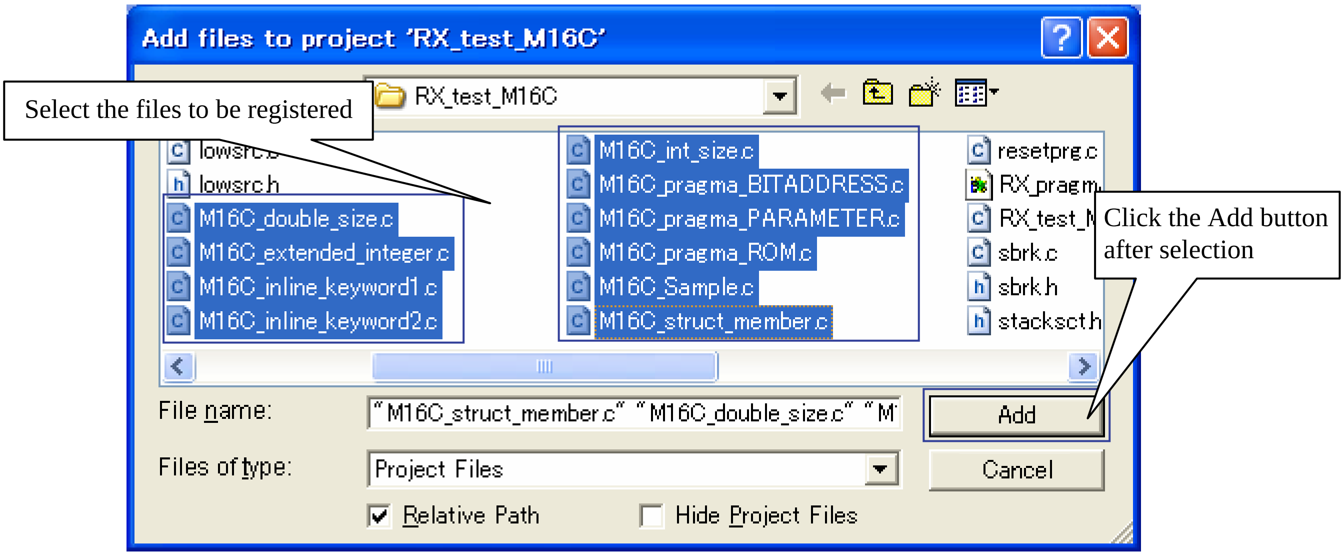


Figure 1-14

(3)Unregistering unnecessary files

Delete the 'RX_test_M16C.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 M16C sample project.

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

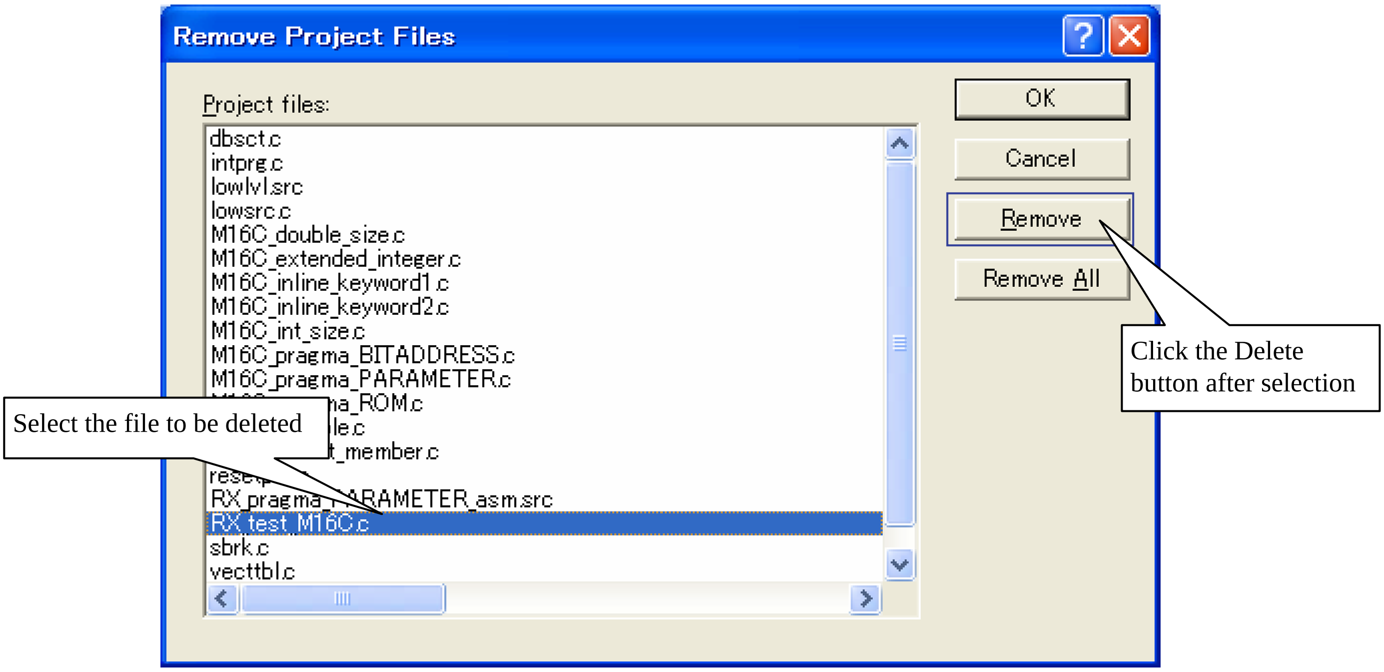


Figure 1-15

2.3 Building and checking M16C compatibility

Build the RX project for which the main processing file was copied and registered. When a build is performed in HEW, since the M16C compatibility check functionality is enabled, option specifications and source code that may impact compatibility can be checked. This section explains how to enable M16C compatibility check functionality and perform a build, and then check the displayed compatibility confirmation messages.

(1) Set up M16C compatibility check functionality

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

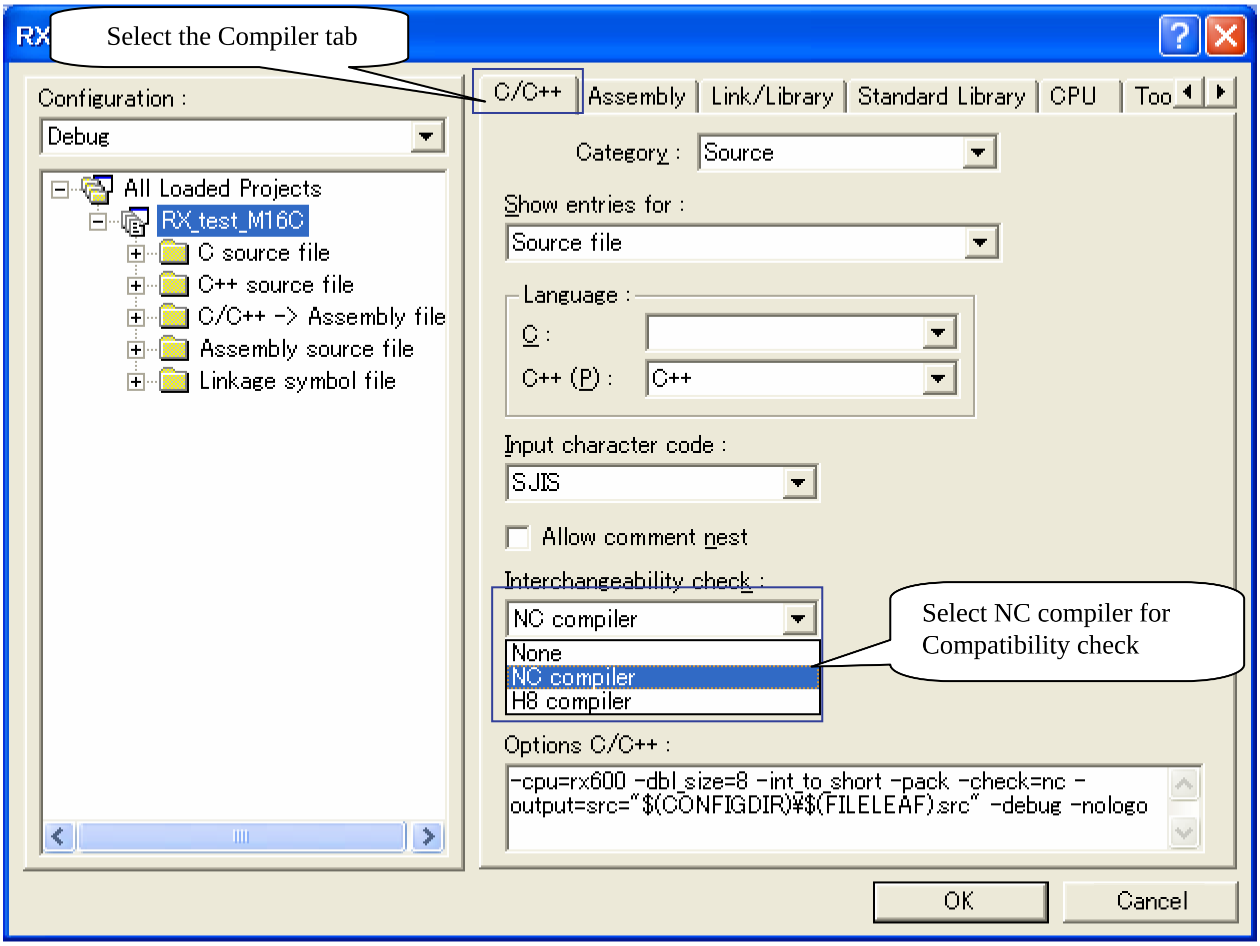


Figure 1-16

(2) Building

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

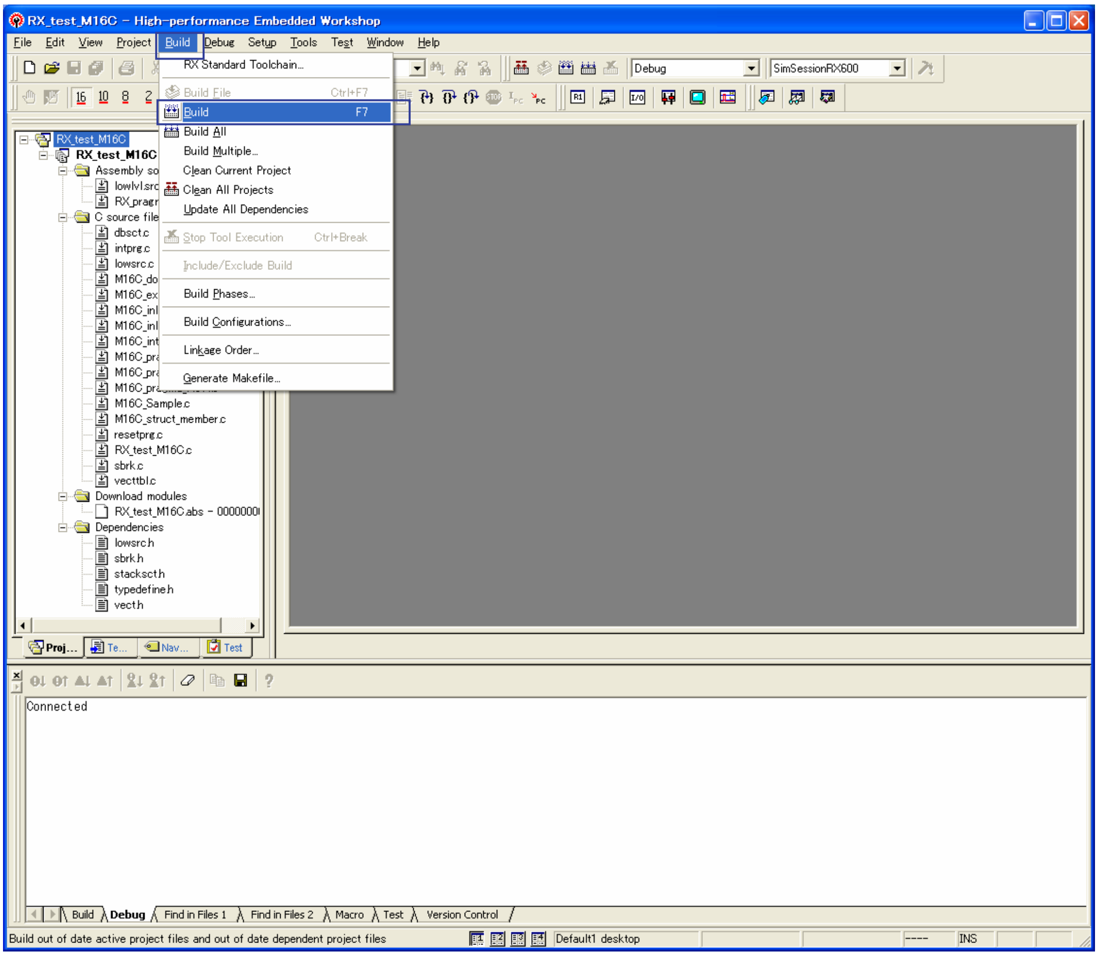


Figure 1-17

Figure 1-18

Output window

(3) Checking compatibility check messages

The C1801 warning is displayed in the messages output to the output window, indicating that this area poses a compatibility problem with M16C.

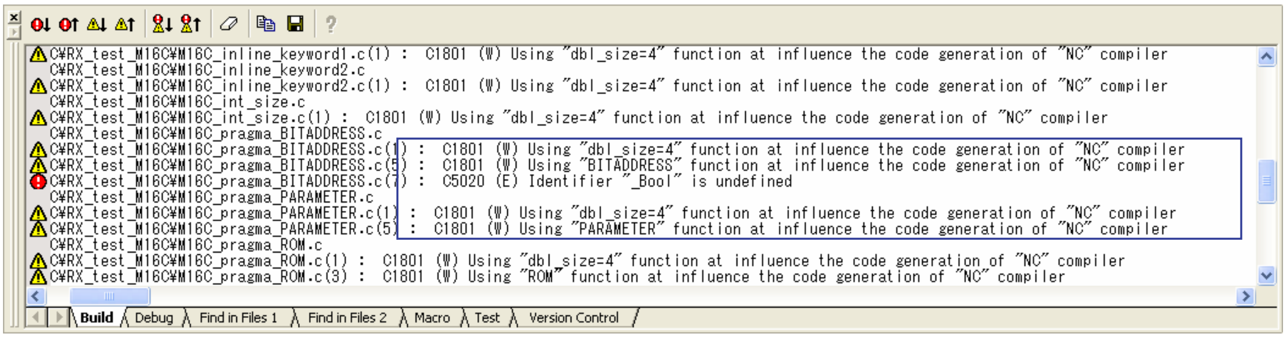


Figure 1-19

2.4 Handling compatibility check instructions

The following explains how to check and deal with C1801 messages affecting compatibility, as given in *2.3 Building and checking M16C compatibility*. The target messages are as follows.

Table 1-2 List of C1801 messages

No	Messages affecting compatibility	Reference
1	Using "dbl_size=4" function at influence the code generation of "NC" compiler	2.4.1
2	Using "BITADDRESS" function at influence the code generation of "NC" compiler	2.4.2
3	Using "PARAMETER" function at influence the code generation of "NC" compiler	2.4.3
4	Using "ROM" function at influence the code generation of "NC" compiler	2.4.4
5	Using "inline" function at influence the code generation of "NC" compiler	2.4.5

2.4.1 Specifying the size of double type variables

The message 'Using "dbl_size=4" function at influence the code generation of "NC" compiler' indicates that a compatibility problem exists with the specified "dbl_size=4" option. With M16C-family compilers, the size of the double type is 8 bytes, whereas with RX-family compilers, the size of the double type is 4-byte because dbl_size=4 is specified in default. Since the size of the double type is required to be 8 bytes based on how "M16C_double_size.c" was coded in the sample program, if the "dbl_size=4" option is specified, the operation results will differ from M16C.

"M16C_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");
    }
}
```

When migrating to RX a program created with the requirement that the size of the double type is 8 bytes, specify the "dbl_size=8" option. For details about specifying this option, see *compiler users manual*. Also, change the options specified in the created RX project.

2.4.2 Specifying #pragma BITADDRESS

The message 'Using "BITADDRESS" function at influence the code generation of "NC" compiler' indicates that a compatibility problem exists with #pragma BITADDRESS in the source. M16C-family compiler support #pragma BITADDRESS, but RX-family compilers do not.

Since "pragma_BITADDRESS.c" in the sample program contains code with #pragma BITADDRESS, it will not operate as expected on RX. Programs using #pragma BITADDRESS can be migrated to RX replacing bit access processing with structure bit fields. The following shows source code using #pragma BITADDRESS, and the same source code migrated to RX.
(Note that since the memory map differs between M16C and RX, absolutely specified addresses need to be changed as appropriate. Use the following as an example.)

"pragma_BITADDRESS.c" sample program

M16C source code

```
#pragma ADDRESS bit_data 410H

int bit_data;

#pragma BITADDRESS bit 0,410H

Bool bit;

void pragma_bitaddress(void)
{
    printf("(6) pragma BITADDRESS : ");
    bit_data = 1;

    if (bit == 1) {
        printf("OK¥n");
    } else {
        printf("NG¥n");
    }
}
```

RX source code

```
#pragma address bit_data 0x2000

int bit_data;

struct bit_address {
    unsigned char b0:1;
    unsigned char b1:1;
    unsigned char b2:1;
    unsigned char b3:1;
    unsigned char b4:1;
    unsigned char b5:1;
    unsigned char b6:1;
    unsigned char b7:1;
};

#define bit (((struct bit_address*)0x2000)->b0)

void pragma_bitaddress(void)
{
    printf("(6) pragma BITADDRESS : ");
    bit_data = 1;

    if (bit == 1) {
        printf("OK¥n");
    } else {
        printf("NG¥n");
    }
}
```

Notes

In addition to #pragma BITADDRESS, "pragma_BITADDRESS.c" also uses the "_Bool" type, which poses a problem for ANSI-standard C89. To use the "_Bool" type in RX, specify the "lang=c99" option to use ANSI-standard C99. For details about specifying this option, see *compiler users manual*.

2.4.3 Specifying #pragma PARAMETER

The message ‘Using "PARAMETER" function at influence the code generation of "NC" compiler’ indicates that a compatibility problem exists with #pragma PARAMETER in the source code. M16C-family compilers support #pragma PARAMETER, but RX-family compilers do not.

Since “M16C_pragma_PARAMETER.c” in the sample program contains code with #pragma PARAMETER, it will not operate as expected on RX. Programs using #pragma PARAMETER can be migrated to RX by changing the argument interface of the assembler function specified by #pragma PARAMETER to conform to C/C++ generation rules.

For details about function interfaces, see *compiler users manual*.

Since #pragma PARAMETER is disregarded by RX compilers, “M16C_pragma_PARAMETER.c” does not need to be changed.

Sample assembler source code

M16C source code “M16C_pragma_PARAMETER_asm.a30”

Assembler source code

```
       .FB      0
       .glb     _asm_func
       .section program
_asm_func:
       enter    #02H
       mov.w    R1,-2[FB]
       add.w    -2[FB],R0
       exitd
       .end
```

Code requiring passage with ‘i’ to be stored in R1 and ’j’ to be stored in R0, by #pragma PARAMETER

RX source code “RX_pragma_PARAMETER_asm.src”

Assembler source code

```
       .GLB      _asm_func
       .SECTION   P,CODE
_asm_func:
       ADD      R2,R1
       RTS
       .END
```

Specify code requiring passage with ‘i’ stored in R1 and ’j’ stored in R2

Since “M16C_pragma_PARAMETER_asm.a30” in the assembler source code of the M16C sample project cannot be migrated as is to the RX project, a new assembler source file needs to be created, and assembler code for RX needs to be added and registered. Since the M16C sample project contains the “RX_pragma_PARAMETER_asm.src” file with assembler code for RX already coded, register it with the RX project.

2.4.4 Specifying #pragma ROM

The message 'Using "ROM" function at influence the code generation of "NC" compiler' indicates that a compatibility problem exists with #pragma ROM in the source code. M16C-family compilers support #pragma ROM, but RX-family compilers do not.

Since "M16C_pragma_ROM.c" in the sample program contains code with #pragma ROM, it will not operate as expected on RX. To migrate this to RX, specify const keywords for variable declarations for which #pragma ROM is specified.

"M16C_pragma_ROM.c" sample program

M16C source code

```
#pragma  ROM  rrr
unsigned short rrr = 100;

void pragma_rom(void)
{
    printf("(7) pragma ROM : ");

    if (rrr == 100) {
        printf("OK¥n");
    } else {
        printf("NG¥n");
    }
}
```

Assembler source code

```
           .SECTION   rom_FE,ROMDATA,align
           .glb       _rrr
_rrr:
           .word      0064H
```

RX source code

```
const unsigned short rrr = 100;

void pragma_rom(void)
{
    printf("(7) pragma ROM : ");

    if (rrr == 100) {
        printf("OK¥n");
    } else {
        printf("NG¥n");
    }
}
```

Assembler source code

```

          .SECTION    C_2,ROMDATA,ALIGN=2
          .glb       _rrr
_rrr:                               ; static: rrr
          .word       0064H
```

2.4.5 Specifying inline keywords

The message ‘Using "inline" function at influence the code generation of "NC" compiler’ indicates that a compatibility problem exists with inline keywords in the source code. M16C-family compilers support inline keywords, but RX-family compilers is specified lang=c by default, and, the way things are going, cannot recognize the inline key word. do not.

Since “M16C_inline_keyword2.c” in the sample program contains code with inline keywords, if it is built as is on the ANSI-standard C89 for RX, a compiler error will occur. There are two ways to migrate programs with inline keywords to RX:

- When performing builds with ANSI-standard C89, change inline code to #pragma inline.
- Perform builds with ANSI-standard C99.

The following gives an example of code using #pragma inline. Change the RX sample project as follows.

“M16C_inline_keyword2.c” sample program

M16C source code	RX source code

<pre>inline int inline_func(void)
{
    return 4;
}
 	
int get_value()
{
    return inline_func();
}</pre>
	<pre>#pragma inline inline_func
	int inline_func(void)
	{
	    return 4;
	}
	
	int get_value()
	{
	    return inline_func();
	}</pre>

Notes

To use inline keywords without modifying the source, specify the “lang=c99” option to enable ANSI-standard C99. For details about how to specify this option, see *compiler users manual*..

Note that C99 inline keywords require precaution due to the following two traits:

- Inline expansions are not always performed, such as when the “noinline” option is specified, or conditions are not met for the “inline” option.
- The function specified by an inline keyword is internally linked, and therefore subject to deletion.

Since “M16C_inline_keyword2.c” in the sample program satisfies the above two conditions, when compilation is performed with the “lang=c99” option, the definition of the inline_func function is deleted regardless of whether inline expansion is performed. As such, a link error will occur because the inline_func function is undefined. Perform one of the following as a workaround.

- Replace inline keywords with #pragma inline, and perform unconditional inline expansion.
- Add inline functions to the extern specification, to be linked externally.

Example of changing inline functions to external linkage

RX source code

```
extern inline int inline_func(void)    // The inline_func function is linked externally,
                                       // and not deleted
{
    return 4;
}

int get_value()
{
    return inline_func();
}
```

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.

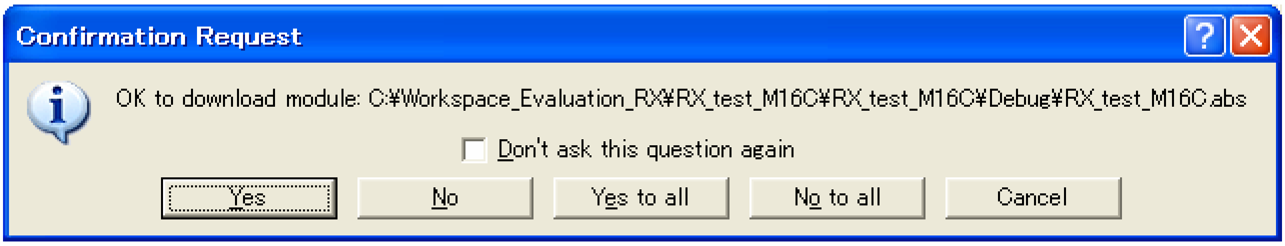


Figure 1-20

2.6 Running the simulator

Execute the rebuilt load module in the simulator.

(1) Setting 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.

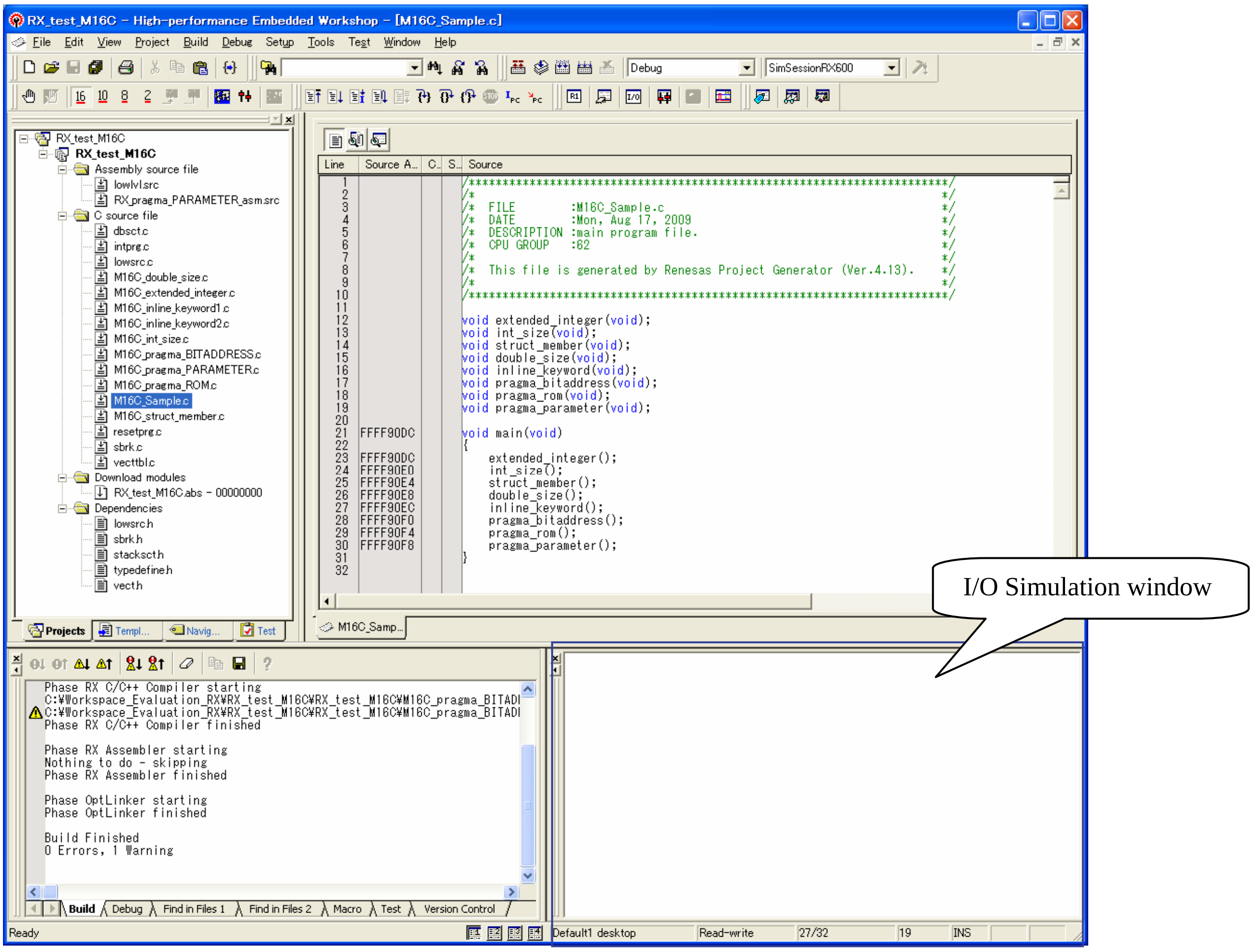
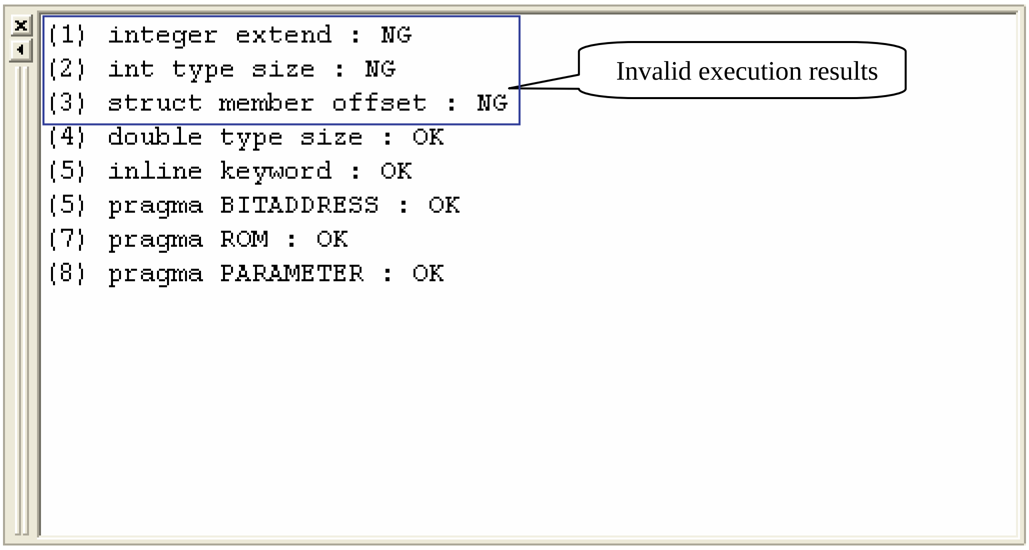


Figure 1-21

(2) 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, (1), (2), and (3) can be checked to see whether the values are invalid.



2.7 Handling invalid execution results

Run the migrated sample project, check the contents of any invalid results, and troubleshoot any compatibility problems for invalid results as shown in the following table.

No	Invalid result	Compatibility problem	Reference
1	integer extend	Integer promotion specification	2.7.1
2	int type size	Size of the int type	2.7.2
3	struct member offset	Placing structure members	2.7.3

2.7.1 Integer promotion specification

Under the ANSI standard, when char type data (such as signed char, unsigned char types) are evaluated, they are always promoted to the int type. The RX specification conforms to the ANSI-standard, but to improve ROM efficiency, M16C does not promote char type data to the int type when evaluating it. This may cause different results, such as code that would cause an overflow while calculating char type data in M16C no longer doing so due to int type promotion after migration to RX. Code that relies on an overflow occurring during calculation of char type data in M16C needs to be checked during migration to RX.

"M16C_extended_integer.c" sample program

M16C source code

```
void extended_integer(void)
{
    char  c1;
    char  c2 = 200;
    char  c3 = 200;

    c1 = (c2+c3)/2;

    printf("(1) integer extend : ");
    if (c1 != 200) {
        printf("OK¥n");
    } else {
        printf("NG¥n");
    }
}
```

M16C overflows the char type size for c2+c3, so that the result of addition is not 400.

RX promotes c2+c3 to the int type, for an addition result of 400 without an overflow

RX source code

```
void extended_integer(void)
{
    char  c1;
    char  c2 = 200;
    char  c3 = 200;

    c1 = ((char)(c2+c3))/2;

    printf("(1) integer extend : ");
    if (c1 != 200) {
        printf("OK¥n");
    } else {
        printf("NG¥n");
    }
}
```

To match M16C, a char type cast is added to the addition results.

Notes

M16C provides the following two options to promote char type data to the int type during evaluation. If either of these options is specified, the difference in integer promotion specifications discussed here will not occur.

- -fansi
- -fextend_to_int

2.7.2 Size of the int type

With M16C-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 “M16C_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 M16C.

“M16C_int_size.c” 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("(2) int type size : ");

    if (u.s.dataH == 0x5f4f && u.s.dataL == 0x7f6f) {
        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 specifying this option, see *1.3 Specifying the size of int type variables* in *C/C++ Compiler Package for the RX Family Application Notes: RX Migration Guide, M16C Edition.* Also, change the options specified for the created RX project.

2.7.3 Placing structure members

If 1-byte, 2-byte, and 4-byte members are mixed within a structure (as with shared structures and classes), free space may occur between the placement of each member, according to each alignment count. M16C-family compilers place structure members using alignment count 1, whereas RX-family compilers place structure members using the maximum alignment count. This means that programs created based on M16C structure placement may not operate properly when migrated to RX.

Since “M16C_struct_member.c” in the sample program contains code that requires a structure alignment count of 1, it will not operate properly as-is on RX. Perform one of the following to migrate the sample program to RX.

- Specify the “pack” option.
- Specify #pragma pack for structures.

For details about how to specify this option, see compiler users manual.

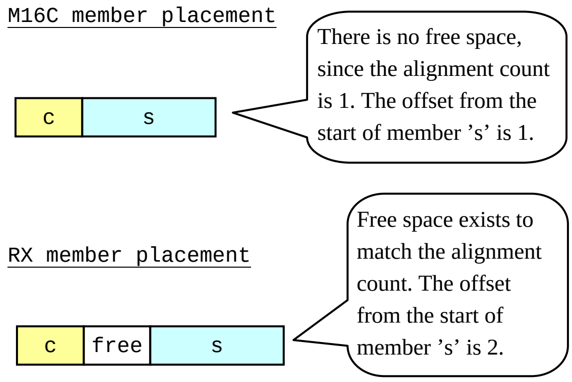
“M16C_struct_member.c” sample program

Source code

```
#include <stdio.h>
#include <stddef.h>

void struct_member(void)
{
   struct s {
      char c;
      short s;
   } ss;

   printf("(3) struct member offset : ");
   if (offsetof(struct s, s) == 1) {
      printf("OK¥n");
   } else {
      printf("NG¥n");
   }
}
```



Specify the “pack” option for 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.

```
(3) struct member offset : NG
(4) double type size : OK
(5) inline keyword : OK
(5) pragma BITADDRESS : OK
(7) pragma ROM : OK
(8) pragma PARAMETER : OK
(1) integer extend : OK
(2) int type size : OK
(3) struct member offset : OK
(4) double type size : OK
(5) inline keyword : OK
(5) pragma BITADDRESS : OK
(7) pragma ROM : OK
(8) pragma PARAMETER : 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