===== Be sure to read this note. =====

C/C++ Compiler Package for M16C Series and R8C Family

NC30 V.6.00 Release 00

Release note

(Rev.1.00)

Renesas Solutions Corp.

Mar 31, 2011

Abstract

Welcome to C/C++ Compiler Package for M16C Series and R8C Family (M3T-NC30WA) V.6.00 . This document contains supplementary descriptions to User's Manual. When you read certain items in the User's Manual, please read this document as well.

1. Installing the C/C++ Compiler Package

For details on how to install, see the installation guide.

2. Where to Contact for the Latest Information

For the latest information on this product, visit and look at the website given below.

http://japan.renesas.com/nc30wa

http://www.renesas.com/nc30wa

For the latest information on this product's document, visit and look at the website given below.

http://japan.renesas.com/nc30wa_document

http://www.renesas.com/nc30wa_document

For the latest information on this product,'s release note visit and look at the website given below.

http://tool-support.renesas.com/jpn/toolnews/p_m16c_1.htm

http://tool-support.renesas.com/eng/toolnews/p_m16c_1.htm

3. Precautions

There are following precautions to take when you use this product.

3.1. About the Supported Version of the Realtime OS MR30

- M3T-MR30/4

  The correct operation of the compiler is not guaranteed when used in combination with M3T-MR30/4 Ver.4.00 Release 01 or earlier.
- MR8C/4

  The correct operation of the compiler is not guaranteed when used in combination with MR8C/4 Ver.1.00 Release 00.

3.2. Precautions Concerning the File Names

For the source program file names, and for the directory and workspace names in which you perform your work, observe the precautions described below.

- The input/output files used by ieee-lmc30 cannot have any characters other than ASCII in their directory names, workspace names or file names.
- The period (.) you use in a file name is usable only once in one file name.
- Network path names cannot be used. If you want to use a network path name, assign it to a drive name.
- "Shortcuts" cannot be used.

* The "workspace" is a working directory in which you compile, build, and debug under the integrated development environment, or High-performance Embedded Workshop.

3.3. Precautions Concerning the Virus Check Program

When you launch the M16C compiler while a virus check program is resident in memory, it may not start up normally. In that case, remove the virus check program from memory and then restart the M16C compiler.

3.4. About the method of comparing the Motorola S-format in an English-language PC

To compare the Motorola S-format, use the Srecord package (srec_cmp) available from the link shown below or similar other tool.

http://srecord.sourceforge.net/

(Note, however, that Renesas will not assume responsibility for troubles or damage arising from the use of this software.)

We've prepared a sample for comparing S-format files in the folder below for your reference.

<install folder>\Tools\Renesas\nc30wa\vXXXrXX\sample\mot_compare

3.5. Precautions Concerning the MCU-Dependent Part

3.5.1. Precautions Concerning the Interrupt Control Register

When the optimization option "-O5" is specified, the compiler may generate bit manipulating instructions (BTSTS and BTSTC). The BTSTS and BTSTC instructions cannot be used as instructions to rewrite the M16C interrupt control register.
When you specify this option, be sure to check that the generated code has no problem.

● Example

If the optimization option "-O5" is specified in a program like the example shown below, the compiler generates a BTSTC instruction by optimization. For this reason, the interrupt request bit may not be determined correctly, causing an unintended behavior to occur.

```
#pragma ADDRESS TA0IC 55H
struct {
        char ILVL:3;
        char IR :1;                     /*Interrupt request bit **/
        char dmy :4;
}TA0IC;
void wait_until_IR_is_ON(void)
{
        while(TA0IC.IR ==0){            /*Waits until the bit is set to 1**/
                ;
        }
        TA0IC.IR =0;            /*Resets the bit to 0 when it is 1**/
}
```

● Solutions

(1) In addition to the relevant optimization option, specify the option "-O5OA" to suppress the optimization that generates BTSTS and BTSTC instructions.
(2) Insert an "asm function" as in the example below to suppress optimization.

```
while(TA0IC.IR ==0){
        asm();          /* Inserts asm function. Suppresses processing performed on TA0IC. */
}
```

● Notes

When you've taken a corrective measure using the option "-O5OA" or an asm function, be sure to check that BTSTS and BTSTC instructions are not generated.

3.5.2. Precautions Concerning Access to the SFR Area

To access the registers in the SFR area, you may need to use a specific instruction.
This specific instruction differs with each MCU type used. For details, see the user's manual for the MCU type you use. When the instruction mentioned here is known, write it directly in your program using the inline assemble facility of an asm function, etc.

3.5.3. Precautions to Taken when Setting Up the Interrupt Priority Level

In response to the technical news No. M16C-14-9804, "Precautions to Take when Using the M16C/60, M16C/61, M16C/62, M16C/63 Group Interrupt Control Registers," functions are now supported that let you set or change the interrupt priority level. Here is an example of how to use.

● To set the priority level

Use the SetLevel function. At this time, be sure to include the intlevel.h file.

```
SetLevel(char *adr, char val);
```

adr : Address of the interrupt control register
val : The value to set

● To change the priority level

Use the ChgLevel function. At this time, be sure to include the intlevel.h file.

ChgLevel(char *adr, char val);

Adr : Address of the interrupt control register

Val : The value to set

Example:

```
#include <intlevel.h>
#pragma ADDRESS timerA 55H
char   timerA;
void   func(void)
{
       SetLevel(&timerA,2);       // Sets the interrupt priority level to 2
       ChgLevel(&timerA,4);       // Changes the interrupt priority level to 4
}
```

3.5.4. Select "E8" When Using On-Chip Debugger E8a

When you use on-chip debugger E8a, take the following steps during the process of creating a new project in the High-performance Embedded Workshop.

(1) Select "C source startup Application" as the project type.

(2) Select "E8" from the "Use OnChip Debugging Emulator" list.

3.6. Precautions Concerning the Compiler, Assembler, Optimizing Linkage Editor, and Utility

3.6.1. About the section location by the optimizing linkage editor (optlnk)

Please be aware that the optimizing linkage editor (optlnk) locates sections in a different way than does the old linkage editor (ln30). For example, assume an assembler source written as follows:

```
.section   prg1
.org       100h
.section   prg2
.section   prg3
.org       200h
.section   prg4
.end
```

ln30 would locate sections in the order shown below unless -order is specified. However, the optimizing linkage editor locates relative sections differently.

```
100H    prg1
        prg2
200H    prg3
        prg4
```

If you want sections to be located by the optimizing linkage editor in the same way as does ln30, use the -start option as you locate sections.

```
optlnk   -start=prg1,prg2/100,prg3,prg4/200
```

If you're using .ORG, be sure that it matches the specified address in the -start option. If they do not match, a warning will result, in which case the set value of .ORG is used.

When you do not use .ORG, you can specify any desired address in the -start option.

3.6.2. About Debugger Display when Local Variables with Same Name Exist in the Same Function

If local variables with the same name are declared in different blocks within a function, the debugger may not be able to display their values correctly. To solve this problem, use different names for the variables.

3.6.3. About the handling of ABS files generated after lowering the error level at link time

If the error level is lowered by an option at link time, it may be possible, even when an error occurs, to forcibly generate load module files (ABS). But we recommend you not to use such load module files.

3.6.4. About the case where a definition of objects using incomplete enumeration type in C++ results in an error

Write a definition of enumeration type to turn it to be of complete type.

3.6.5. About the case where, when debug is specified, specification of extern or static for the template function results in an error

Although specifying an extern or static declaration while debug is specified (-g option) results in an error, this is a syntactic error in writing, so rewrite the program not to use extern or static.

3.6.6. The keyword mutable is not supported.

3.6.7. About a problem that writing type near or far beforehand in a function-like cast "type (expression)" form results in an error

The types near and far written in casts have no effect on program behavior, so do not write them.

3.6.8. About the use of #pragma ASM in C++

Writing #pragma ASM external to a function in a C++ source results in a compile error.
Since asm() can be written external to a function, if it's necessary to write assembly instructions outside of a function, we recommend you to use asm(). In C sources, both #pragma ASM and asm() can be used outside of a function as in the past.

3.6.9. About assembler macro function specifications in C++

Assembler macro functions defined using #pragma ASM cannot be used in C++ source files.
In such a case, alter the macro definition using an asm function instead of #pragma ASM, as in the example below.

・Example

```
#pragma __ASMMACRO addition(R0,R2)
static int addition(int, int);
    _asm("_addition .macro\n"
         "          add.w    R2,R0\n"
         "          .endm");
```

3.6.10. About limitations in cases where a function itself is referenced using #pragma ASM, asm() or assembler macro function call in it

The following limitations apply.
(1) The number of times the function was referenced, or the information shown in .map, is not output correctly.
(2) Even if the function is an unreferenced symbol, it is not removed during optimization by optlnk.

・Example

```
void func(void)
 {
 #pragma asm
    .call _func,G
    jsr _func
 #pragma endasm
 }
```

3.6.11. About the registration of init.c and device.c when upgrading from Ver. 5.xx to Ver. 6.xx

When you upgrade a Ver. 5.xx project that uses standard input/output to Ver. 6.xx, you need to register init.c and device.c in the upgraded project. Get the copies of init.c and device.c generated following the method below and register them in your upgraded project.

- Launch HEW.
- Select the menu item to create a new project.
- Select the MCU type.
- Enable the checkbox "Use I/O Library."

For the M16C/24, however, the standard input/output device of Ver.5.xx was UART0, but that of Ver.6.xx is UART1. Use the compiler option -D_UART0__ as you compile device.c. That way, you can change the standard input/output device to UART0.

3.6.12. Precautions concerning C++ dynamic initialization

If const has the far attribute, the located section of a const variable declared outside of a function, one that involves C++ dynamic initialization, is bss_FE or bss_FO. By removing the const qualifier from the relevant variable declaration, you can change the located section of the variable to bss_NE or bss_NO.

3.6.13. There is a mistake in the description of the standard library function scanf in the compiler user's manual

[Wrong]
"argument must be a far type pointer to each variable"
[Correct]
"argument must be a near type pointer to each variable"

3.6.14. If the type of the argument to be converted as specified in one of printf functions is type double, the result is incorrect.

- Occurrence condition

The problem occurs when all of the following conditions are met.
(1) One of the following compiler options is used.
  (a) -fdouble_32(-fD32)
  (b) -OR_MAX(-ORM)
  (c) -OS_MAX(-OSM)

(2) One of the following library functions is called.
  (d) printf
  (e) fprintf
  (f) sprintf
  (g) vprintf
  (h) vfprintf
  (i) vsprintf

(3) The type of the argument to be converted as specified in (2) is type double.

- Occurrence example

[Command line]
nc30 -c -fD32 sample.c /* Occurrence condition (a) of (1) */

[sample.c]
```
#include <stdio.h>
void func(double x)
{
        printf("%f\n", x); /* Occurrence condition (2) and (3) */
}
```
・Solutions
Cast to type long double.
[Example]
```
#include <stdio.h>
void func(double x)
{
      printf("%f\n", (long double)x);
}
```

3.6.15. If the type of the argument to be converted as specified in one of scanf functions is a pointer to type double, the result is incorrect.

・Occurrence condition
The problem occurs when all of the following conditions are met.
(1) One of the following compiler options is used.
(a) -fdouble_32(-fD32)
(b) -OR_MAX(-ORM)
(c) -OS_MAX(-OSM)

(2) One of the following library functions is called.
(d) scanf
(e) fscanf
(f) sscanf

(3) The type of the argument to be converted as specified in (2) is a pointer to type double.

・Occurrence example
[Command line]
```
nc30 -c -fD32 sample.c
```
[sample.c]
```
#include <stdio.h>
void func(double* x)
{
      scanf("%lf\n", x);
 }
```
・Solutions
Remove the modifier 1.
[Example]
```
#include <stdio.h>
void func(double* x)
{
            scanf("%f\n", x);   /* Modifier 1 removed */
}
```

3.6.16. About assert macro specifications

Specifications of the assert macro are as follows:

Description:
Adds a diagnostic feature in a program.
Format:
#include <assert.h>
void assert(test)
Implementation:
Macro
Parameter:
test; Expression to be evaluated
Return value:
None
Explanation:
The assert macro, when test is true, finishes processing without returning a value. When test is false, it outputs diagnostic information to a standard error file in the form defined by the compiler. Then it calls the abort function.
The diagnostic information includes program text of parameters, source file names and source line numbers.
To enable the assert macro, it is necessary to call the standard input/output initialize function _init.
To disable the assert macro, define a macro NDEBUG before including assert.h.
Implementation-defined specification:
In assert(test), if test = false, a message is output.
Assertion failed: △expression, △<file name>(line△<line number>)

3.6.17. Precautions concerning the compiler option -Wlarge_to_small(-WLTS)

When you use the compiler option -Wlarge_to_small(-WLTS), pay attention to the following.
(1) When compiled as a C++ program, a warning is output only when the right side is a constant.
(2) When compiled as a C program, a warning is output when the right side consists only of a variable.

3.6.18. Method of avoiding the fixed vector when locating sections in the R8C family (ROM 64KB or more)

By setting the -cpu option at link time, it is possible to avoid the fixed vector as you locate sections.
Example: -cpu=RAM=400-11ff,ROM=4000-ffd7,ROM=10000-23fff -cpu=stride
* In the above example, mapping of ROM/RAM to the vector area (address range from 0ffd8H to 0ffffH) is suppressed.
Automatic placement in noncontiguous RAMs (e.g., R8C/2A to R8C/2D) is also possible.
To set from GUI of High-performance Embedded Workshop, follow the procedure described below.
(1) From the Build menu, select Renesas M16C Standard Toolchain.
(2) Select the Linker tab.
(3) Select Verify from the Category list.
(4) Select Verify from the Option list.
(5) Click the Add button and set the type of memory and the start and end addresses.
(6) Enable the "-cpu=stride" checkbox.

3.7. Limitations and Known Troubles

The edition is subject to the following limitations:

- An internal process may output warnings or error messages that are duplicates of those output by a different internal process.
- Use of a binary notation of underscore (_) in C++ results in an error.
- When using an EC++ class library, do not use -fdouble_32 (-fD32).
- If #pragma is used in a function that has no prototype declaration, neither warnings nor error messages are output.
- A breakpoint may not always be set at the beginning of a function that is defined at the top in a class.
- No errors result even when a frame-passed argument or a non-argument frame variable is declared by #pragma entry.

  The functions using #pragma entry are such that because the user stack pointer, interrupt stack pointer and frame base register are not set at the beginning of the function yet, arguments and auto variables cannot be used. Be sure that no arguments and auto variables are used in a function that uses #pragma entry.
- If an expression casting a negative floating-point constant to unsigned integer type is written in an argument to a function call, a C1841 warning is output and the cast value of the constant becomes zero.

  ・Occurrence example

```
int func(int x) { return x; }
void main( void )
{
    int i = (int)((unsigned int)-1.0f);
    int j;
    j = func((int)((unsigned int)-1.0f));

    printf("%d, %d\n", i, j);     /* Becomes –1 and 0 */
}
```

  ・Solutions

  Assign to a temporary variable before a function call and use that variable in the function call.

```
int func(int x) { return x; }
void main( void )
{
    int i = (int)((unsigned int)-1.0f);
    int j;
    int tmp = (int)((unsigned int)-1.0f); /* Here */
    j = func(tmp);

    printf("%d, %d\n", i, j); /* Becomes –1 and 0 */
}
```

- If a near array is pointed to by a far pointer and a negative value is used to calculate the address of the far pointer, correct elements may not always be retrieved.

  ・Occurrence condition

  The problem occurs when all of the following conditions are met.

  (1) A near array is declared.

  (2) A far pointer is declared.

  (3) The pointer in (2) points to subsequent but the first element of the array in (1).

  (4) Based on 2-dimensional or higher-order address calculation for (3) and a negative value representing the number of elements, the array elements pointed to by the pointer are referenced.

・Occurrence example

```
#include <stdio.h>
int near buf[2][1] = { { 1 }, { 2 } };
int far (*p)[1] = buf + 1;
void main( void )
{
        int i = 0;
        int j = -1;

        if( p[i][j] != 1 ) {
                printf( "NG\n" );
        } else {
                printf( "OK\n");
        }
}
```

・Solutions

Do not use a negative value for the subscript of an array reference.

- No code is generated for a function using #pragma interrupt (interrupt vector numbers available) and no function address is set in the variable interrupt vector.

・Occurrence condition

The problem occurs when all of the following conditions are met.

(1) One of the following applies.

(a) The compiler option -OS_MAX(-OSM) is used.

(b) The compiler option -Oforward_function_to_inline(-OFFTI) and the following compiler options are used.

(b1) -O[1–5]

(b2) -OR_MAX(-ORM)

(2) #pragma interrupt (interrupt vector numbers available) is used.

(3) The object of (2) is an internal linkage function.

(4) The function in (3) has no address reference.

・Occurrence example

[Command line]

```
nc30 -c -OSM sample.c
```

[sample.c]

```
#pragma interrupt func(vect=31)
static void func(void)
{
}
```

・Solutions

Define a dummy variable that refers to the address of a function that comes under the above occurrence condition.

```
#pragma interrupt func(vect=31)
static void func(void)
{
}
void (*dummy)(void) = &func;    /* Address reference */
```

- If .SECTION(.END) that terminates a CODE section and the last instruction are written in separate pieces of an assembly language source file, the program cannot be debugged correctly.

  ・Occurrence condition
  The problem occurs when all of the following conditions are met.
  (1) The assembler option "-N" is not set.
  (2) .SECTION(.END) that terminates a CODE section and the last instruction are written in separate files.

  ・Occurrence example
  <sampe.a30>
  ```
  .SECTION prg,CODE
  NOP
  NOP
  .INCLUDE    sample.inc
  .END                    ; .END that terminates a section
  ```
  <sample.inc>
  ```
  RTS                     ;; Instruction at the end of a prg section
  ```

4. Contents of version-up from V.5.45 Release 01

For details, please see Appendix K in C/C++ Compiler User's Manual for the M16C Series, R8C Family C/C++ Compiler Package V.6.00.

4.1. When using -ffar_pointer (-fFP) in C++, pointers to arrays in functions are handled as 16 bits.

In C++, pointers to arrays in functions are handled as 16 bits. When the far qualifier is explicitly written, the pointers can be handled as 32 bits.

```
int i1, i2;
void func(void)
{
 i1 = sizeof(int(*)[]);       // 4 in C and 2 in C++ when –fFP is used
 i2 = sizeof(far int(*)[]);   // Always 4
}
```

4.2. Cast Notation Regarding near/far

Although the compiler user's manual states that const_cast can be used for casting a far pointer to a near pointer, this is not correct.

[Incorrect]
If it is evident that the far pointer to be substituted points to a near area, cast the far pointer to the near pointer by a cast notation or const_cast operator in order to avoid an error.

[Correct]
If it is evident that the far pointer to be substituted points to a near area, cast the far pointer to the near pointer by a cast notation in order to avoid an error.

4.3. Note on .glb Declaration for Symbol __SB__

.glb declaration for symbol __SB__ to be generated by the compiler has been changed as follows.

● V.5.xx
The compiler outputs .glb __SB__ whether __SB__ is used or not.

● V.6.xx
The compiler only outputs .glb __SB__ when it acknowledges that __SB__ is used in a C source program.
One of the following conditions must be satisfied.

a) #pragma SBDATA is written.
b) Compiler option -fauto_over_255(-fAO2) is used and a function that will use a stack area of 255 bytes or more is defined.
c) A function using #pragma TASK (which is a pragma directive for realtime OS) is defined.

Due to this change, porting a V.5.xx project, in which a C-source startup program was created, to a V.6.xx project will lead to an error because __SB__ is detected as an undefined symbol.
To avoid this error, add the following description to the file in which symbol __SB__ is defined.

```
_asm(" .glb __SB__");
```

4.4. Linkage of standard libraries may not be possible when -ffar_pointer (-fFP) or -ffar_RAM (-fFRAM) is used.

4.5. Code hoisting optimize (which, from compound statements on both of if-else statements, puts common expressions together in front of the if statement for optimization) has been enhanced.

4.6. In NC30 V.5xx, arguments were structures and … inline function definitions were an assembly macro. From V.6.00 on, however, they are ordinary functions.

4.7. A problem that if functions and variables with the same name as those of the “functions and variables declared at file scope” are declared as extern in a function, no errors result even when they differ in type and the declaration in the function is made valid, has been corrected.

・Occurrence condition
The problem occurs when all of the following conditions are met.
(1) There is a variable or function declaration at file scope.
(2) There is an extern declaration with the same name as (1) but differing in type.
(3) Within the function definition (2), the variable or function in (2) is accessed at a position where the declaration in (2) is valid.

4.8. A trouble that when writing successively to union members, writes are sometimes performed in a wrong order, has been corrected.

・Occurrence condition
The trouble occurs when all of the following conditions are met.
(1) The optimization option -O[1–5], -OS, -OR, -OS_MAX or -OR_MAX is selected
(2) There are multiple assignment statements occurring in succession that assign constant values to different members of one union.

(3) The members to which assigned in (2) are not volatile qualified.
(4) Two or more of members to which assigned in (2) that differ in size coexist.
・Occurrence example

```
union{
    unsigned int   unim01;
    unsigned int   unim02;
    unsigned long unim05;
} uni[2];

void main(){
    uni[0].unim01 = 0x1111;
    uni[1].unim01 = 0x1111;
    uni[0].unim02 = 0x2222;
    uni[0].unim05 = 0x55555555; /* Because members have their values substituted in a wrong
                                   order, this value does not remain */

}
```

4.9. A problem that if the sizeof operator is used in an additive expression of integer and array name a wrong size is returned, has been corrected.

There was a problem that if the sizeof operator is used in an additive expression of integer and array name, a value derived by adding the array size and not the pointer size results.
・Occurrence condition
The problem occurs when all of the following conditions are met.
(1) The sizeof operator gets the size of the type of an additive expression of integer constant and array name.
(2) The additive expression in (1) consists of an integer constant in the left operand and an array name in the right operand of the additive operator.
・Occurrence example

```
short arr[30],i;
int test(void)
{
    i = sizeof(0+arr);/* The size of the array, not the size of &arr[0] address expression, is returned
                         by mistake. */
}
```

4.10. A problem that a constant expression of floating type does not become zero when its value is smaller than the minimum value of the normalized number, has been corrected.

Concerning the result of an operation performed on a constant or two constants of floating type, it was observed that its absolute value, when smaller than the minimum value of the normalized number, is not rounded to zero.

・Occurrence condition
The problem occurs when all of the following conditions are met.
(1) A constant of floating type or a constant expression of floating type is used.
(2) The absolute value of (1) is smaller than the minimum value of the normalized number (FLT_MIN for float type, or DBL_MIN for double type).

・Occurrence example

```
float test()
{
    float f = 1.1760000000e-38F - 0.0020000000e-41F;
    return f;  /* The subtraction result is smaller than FLT_MIN, where 0 should be returned, but
                  actually a denormal number is returned */
}
```

4.11. A problem that for a conditional expression that compares 0.0 and NaN, the operands are erroneously evaluated to compare as equal, has been corrected.

・Occurrence condition

The problem occurs when all of the following conditions are met.
(1) The compile option -O[1–5], -OR, -OS, -OR_MAX(-ORM) or -OS_MAX(-OSM) is selected.
(2) There is a controlling expression to be compared in an equality or inequality operation.
(3) One side of the controlling expression in (2) is a variable that has the value of NaN.

・Occurrence example

```
int test(void)
{
    float f  = 3.402823466e+38 + 0.001e+38;
    float f1 = 3.402823466e+38 + 0.001e+38;
    f /= f1;
    if (f == 0.0f) {
        return 0;
    }
    return 1;
}
```

4.12. A problem that when the compiler option -OS_MAX is selected, a forward-referenced inline function results in an error, has been corrected.

A problem was observed that when compiler option -OS_MAX is selected, the -Oforward_function_to_inline(-OFFTI) option is not enabled and a forward-referenced inline function results in an error.

・Occurrence condition

The problem occurs when all of the following conditions are met.
(1) The compile option -OS_MAX(-OSM) is specified.
(2) An inline qualified function is defined.
(3) The inline qualified function in (2) is called before its definition.

・Occurrence example

```
inline int add(int, int);

int func(void)
{
    int rc = add(1, 2);
    return rc;
}
```

```
inline int add(int a, int b)
{
    return a + b;
}
```

4.13. A problem that an argument that has double-precision floating type in function declaration and has single-precision floating type in its definition is erroneously read, has been corrected.

・Occurrence condition

The problem occurs when all of the following conditions are met.

(1) One of the following compiler options is used.

(2) A function that has an argument of double-precision floating type is declared.

(3) The argument in (2) is defined in an old form.

(4) The argument in (2) is defined to be single-precision floating type in its definition.

(5) The first access to the argument in (4) is a read.

・Occurrence example

```
nc30 -c -O1 xxxx.c

#include <stdio.h>
float func(double);
float func(f)
    float f;
{
    return f;
}
void main(void)
{
    float x = func(1.0);
    if (x == 1.0) {
        printf("OK\n");
    } else {
        printf("NG\n");
    }
}
```

4.14. A problem that an argument defined to be _Bool type in an old form function definition is erroneously read, has been corrected.

・Occurrence condition

The problem occurs when all of the following conditions are met.

(1) A function is defined in an old form.

(2) The argument to (1) is _Bool type.

(3) A value that the argument in (2) had before it was altered is read.

(4) The value of the argument in (2) is an even number equal to or greater than 2.

・Occurrence example

```
#include <stdio.h>
_Bool func(x)
    _Bool x;
{
    return x;
}
void main(void)
{
    _Bool x = func(2);
    if (x == 1) {
        printf("OK\n");
    } else {
        printf("NG\n");
    }
}
```

4.15. A problem that if the type of a switch statement controlling expression is signed char, control does not branch to the correct case label, has been corrected.

・Occurrence condition

The problem occurs when all of the following conditions are met.

(1) The type of a switch statement controlling expression is signed char.

(2) The switch statement in (1) satisfies one of the following:

(a) The minimum value of a case value is –127 and the maximum value is 127, wherein the case value is contiguous from –127 to 127 without a break.

(b) The minimum value of a case value is –128 and the maximum value is 126, wherein the case value is contiguous from –128 to 126 without a break

(c) The minimum value of a case value is –128 and the maximum value is 127, wherein a total of 135 or more instances of the case statement (not including default) exists.

(3) The value of the controlling expression in (1) is smaller than zero.

・Occurrence example

```
#include <stdio.h>
signed char d = -1;
void main( void )
{
    switch( d ) {
        case -127 : printf( "NG...[-127]--->[%d]\n", d ) ; break ;                    …
        case   -1 : printf( "OK\n", d ) ; break ;
        …….
        case   127 : printf( "NG...[127]--->[%d]\n", d ) ; break ;          }
}
```

4.16. A problem that if, when initializing the structure of auto variables, a function is called that returns a structure, System Error is generated, has been corrected.

・Occurrence condition

The problem occurs when all of the following conditions are met.

(1) A function that returns a structure or union is declared.

(2) The 1st argument to the function in (1) is passed via the stack.

(3) The function in (1) is called to initialize auto variables.

・Occurrence example

```
struct S {
    int i;
    int j;
};

struct S func(long);

int test(void)
{
    struct S s = func(0x1234);
    return 1;
}
```

4.17. A problem that if, while the compile option -Oloop_unroll(-OLU) is specified, an inline function call in a for statement is attempted, an assemble error results, has been corrected.

A problem that if, while using the compile options -Oloop_unroll(-OLU) and -Oforward_function_to_inline(-OFFTI) in combination, a function call to be expanded in-line is written in an iteration statement whose loop is to be unrolled, an assemble error is generated, has been corrected.

・Occurrence condition

The problem occurs when all of the following conditions are met.

(1) Both of the following compiler options are selected.

(2) In the compiler option -Oloop_unroll(-OLU), a function call to be expanded in-line is written in a loop to be unrolled.

(Example1) A case where an inline-declared function is called in a loop

(Example2) A case where the compiler option -Ostatic_to-inline(-OSTI) is selected and a static-declared function is called in a loop

・Occurrence example

```
int gi;
inline void inline_func(void)
{
    ++gi;
}

void func(void)
{
    char i = 0;
    for( i = 0; i <= 4; i++ ){
        inline_func();
    }
}
```

4.18. A problem that constants larger than LONG_MAX written in a constant expression following #if or #elif were not correctly interpreted has been corrected.

Example:

```
#include <stdio.h>
void main()
{
#if   2147483648 > 0
      printf("2147483648 > 0\n");     // Ver.6.xx
#else
      printf("2147483648 <= 0\n"); // Ver.5.xx
#endif
}
```

4.19. Precautions concerning the standard input/output functions have been corrected.

Where to contact: FAQ
http://japan.renesas.com/support/faqs/faq_results/Q1000000-Q9999999/tools/coding_tools/compiler_100706a.jsp
http://www.renesas.com/support/faqs/faq_results/Q1000000-Q9999999/tools/coding_tools/compiler_100706a.jsp

4.20. Precautions concerning the assembler directives .id and .ofsreg have been corrected.

Where to contact: FAQ
http://japan.renesas.com/support/faqs/faq_results/Q104601-Q104700/tool_faq_2005020701.jsp
http://www.renesas.com/support/faqs/faq_results/Q107401-Q107500/compiler_107459_en_GL.jsp

4.21. Precautions concerning the right shift operation have been corrected.

Where to contact: Tool news
http://tool-support.renesas.com/jpn/toolnews/070716/tn4.htm
http://tool-support.renesas.com/eng/toolnews/070716/tn4.htm

4.22. Precautions concerning the case where structure members are initialized by an expression that includes the sizeof operator have been corrected.

Where to contact: Tool news
http://tool-support.renesas.com/jpn/toolnews/080716/tn2.htm
http://tool-support.renesas.com/eng/toolnews/080716/tn2.htm

4.23. Precautions concerning the case where the compile option -Ostack_frame_align(-OSFA) is used have been corrected.

Where to contact: Tool news
http://tool-support.renesas.com/jpn/toolnews/070701/tn5.htm
http://tool-support.renesas.com/eng/toolnews/070701/tn5.htm

4.24. Precautions concerning the type definition of an incomplete-type structure or union have been corrected.

Where to contact: Release note
http://tool-support.renesas.com/jpn/toolnews/100401/tn3.htm
http://tool-support.renesas.com/eng/toolnews/100401/tn3.htm

5. Project conversion in the integrated development environment (High-performance Embedded Workshop, or HEW)

5.1. Settings when you're using an assembler startup

The contents written in K.1.2, "HEW Project Conversion," in C/C++ Compiler User's Manual Appendix K, "Contents of Upgrade and Migration Method," apply to the setup method in cases when you're using the M16C series. Therefore, if you're using an assembler startup of the R8C family, you need to set the content given below in the optlnk option "-start" that specifies the start address of a section.

[Less than ROM 64K]
-start=data_SE,bss_SE,data_SO,bss_SO,data_NE,bss_NE,data_NO,bss_NO,istack,stack,heap/0400,rom_NE,rom_NO,data_SEI,data_SOI,data_NEI,data_NOI,switch_table,program,interrupt/0E000,vector/0FED8

[Over ROM 64K]
-start=data_SE,bss_SE,data_SO,bss_SO,data_NE,bss_NE,data_NO,bss_NO,istack,stack,heap/0400,rom_NE,rom_NO,data_SEI,data_SOI,data_NEI,data_NOI,switch_table,program,interrupt/04000,vector/0FED8,rom_FE,rom_FO/010000

* The start address of each section needs to be changed to what you've set.
* If you've added any section, they need to be added to the above setting.

How to set the option –start

- From the Build menu of HEW, select Renesas M16C Standard Toolchain → Linker → Category: Section.
- Select Add and in the ensuing "Add Sections," locate each section at the respective addresses.

Address	Section
0x400	data_SE,bss_SE,data_SO,bss_SO,data_NE,bss_NE,data_NO,bss_NO,istack,stack,heap
0x0E000	rom_NE,rom_NO,data_SEI,data_SOI,data_NEI,data_NOI,program,interrupt
0x0FED8	Vector

- Mapping of each section is finished.

[Precautions]
It is possible that when you execute a build after you've finished mapping each section, an optlnk warning "L1323 (W) Section attribute mismatch: section name" will be output.
To avoid this warning, alter your source as described below.
・Comment out the address declaration (.ORG) for the section name in which the warning was generated. (In the above example, this applies to the section names "data_SE," "rom_NE" and "vector" in sect30.inc.)
・If the code written in sect30.inc below causes an assemble error, change "data_SE_top" written in the operand of the assembler directive ".EQU" to the start address value of "data_SE."

```
; SBDATA area definition
                .glb        __SB__
__SB__          .equ        data_SE_top
```

6. List of Software Versions

The software versions included with the C/C++ compiler package V.6.00 are listed below.

- nc30.exe V.7.00.01.000
- rcfrt.exe V.3.10.1
- ccom30.exe V.6.00.02.000
- aopt30.exe V.1.05.02.000
- sbauto.exe V.1.00.00.000
- as30.exe V.6.00.01.000
- mac30.exe V.3.44.01.000
- pre30.exe V.1.12.01.000
- asp30.exe V.6.00.04.000
- optlnk.exe V.10.01.00.000
- lb30.exe V.1.02.00.000
- ieee-lmc30.exe V.4.03.00.000
- utl30.exe V.1.01.02
- lbg30.exe V.1.00.000
- conv30.exe V.1.00.00.000
- elvconv.exe V.1.00.00.000
- prelnk.exe V.1.3.0.0

7. A Guide to Porting Projects Created with TM to High-performance Embedded Workshop Ver.4

This document explains how to port projects created with TM Ver.2.xx or Ver.3.xx into High-performance Embedded Workshop Ver.4.
For how to port projects from TM to High-performance Embedded Workshop (NC30WA Ver. 6.xx), see the FAQ section of the Renesas' development tools page.

7.1. Summary

To port projects created using TM Ver.2.xx or Ver.3.xx into High-performance Embedded Workshop Ver.4, the Import Makefile function of High-performance Embedded Workshop is used. This function can create projects from such items of information as source files and build options described in the specified makefile files.
In TM, project files are created in the makefile format executable in GNU make format. When project files created with TM are selected as makefile files using High-performance Embedded Workshop Import Makefile function, they are converted to files that can run in High-performance Embedded Workshop. In addition to TM project files, the Import Makefile function can also convert files in the makefile formats for hmake, nmake, and gmake to High-performance Embedded Workshop projects.

7.2. Porting Procedure

To port projects created using TM into High-performance Embedded Workshop, perform the following steps:

1. Open the File menu and select the New Workspace command.
2. The New Project Workspace dialog box opens.

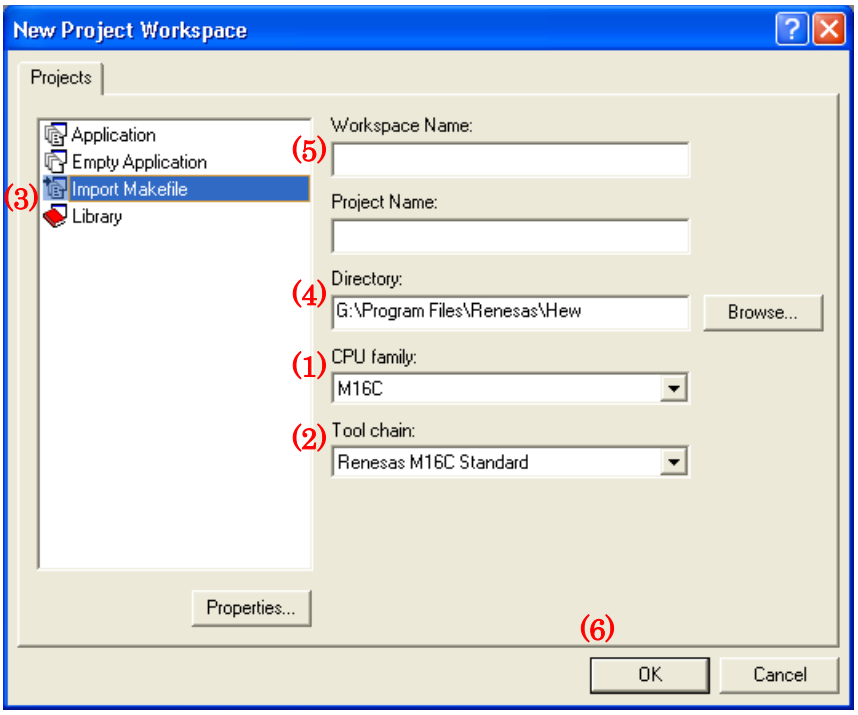

Figure 1 New Project Workspace Dialog Box

- Select the type of CPU used in the TM project from the Type of CPU drop-down list.
- Select the tool chain (cross tool) used for the TM project from the Toolchain drop-down list. The names of tool chains and corresponding cross tools are shown in Table 1.

Table 1 Tool Chains and Corresponding Cross Tools

Tool Chain	Cross Tool
Renesas M16C Standard	NC30WA
Renesas R8C Standard	NC8C
Renesas M32C Standard	NC308WA
Renesas M32R Standard	CC32R

- Select Import Makefile from the Project list.
- Type the directory path in the Directory text box.
- Type the workspace name in the Workspace Name text box. The same name will be automatically entered as the project name in the Project Name text box.
- Click **OK**.

3. You should now be able to see the New Project-1/4-Import Makefile wizard.

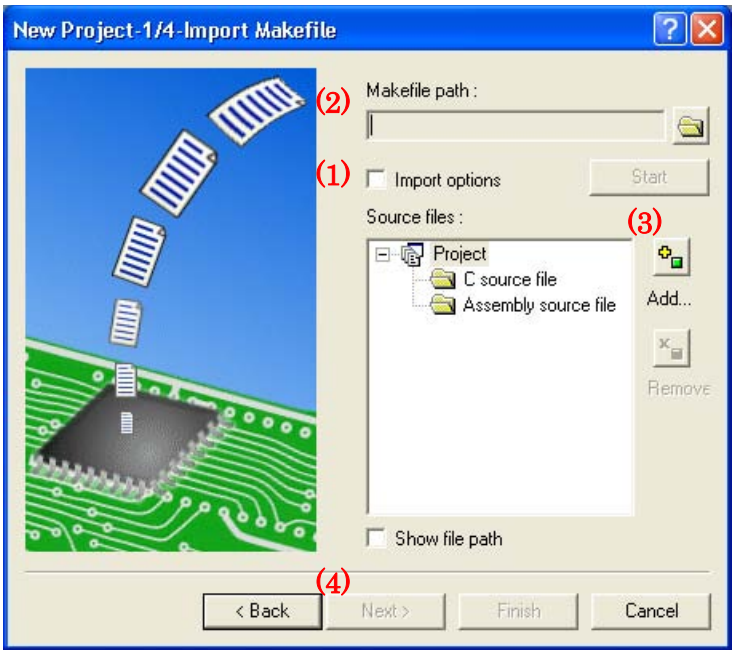

Figure 2 New Project-1/4-Import Makefile Wizard

- Select the Import options check box; this will enable information on build options (compiling and assembling options etc.) to be used to create High-performance Embedded Workshop projects. If you clear the Import options check box, the above information is neglected and not used in High-performance Embedded Workshop.
- Type the name of the TM project file (with extension .tmk) in the Makefile path text box. As soon the name is input, the specified file is analyzed, and upon analysis completion, the analyzed source files are displayed in a tree structure in the Source files box. Click the Start button to analyze the specified file again.
- If there are any errors in the analysis results (tree structure in the Source files box), rectify the tree structure with the Add and Remove buttons.
- Click **Next**.

4. Follow the instructions according to the Wizard as it continues in the procedure.

7.3. Usage Notices

7.3.1. TM-to-High-performance Embedded Workshop Portable and Non-Portable Information

When you port a project created using TM into High-performance Embedded Workshop, not all the components of the project can be ported.

Portable information is as follows:

- Paths of assembler source files
- Paths of C-language source files
- Assembling options
- C-compiling options
- Linking options (except linkage order)

Non-Portable Information:

- Linkage order
- Tool configurations, dependencies, and options other than Assembler, C Compiler, Linker

To transfer these items, edit the High-performance Embedded Workshop project as described in Section 3.4 and further after processing the Import Makefile.

7.3.2. Cross Tools

Import Makefile cannot enable all cross tool versions for use in High-performance Embedded Workshop projects regardless of whether they are used with TM or not; only the following cross tools versions are valid for High-performance Embedded Workshop projects:

NC30WA : V.5.20 Release1 … V.5.45 Release 01

7.3.3. High-performance Embedded Workshop Versions

When TM projects are ported into High-performance Embedded Workshop, information portable to High-performance Embedded Workshop varies according to the High-performance Embedded Workshop version. The information that can be ported from each cross tool to various High-performance Embedded Workshop versions are shown in Table 2.

Table 2 Portable Information and Corresponding High-performance Embedded Workshop Versions

		High-performance Embedded Workshop				
		~V.3.01.02	V.3.01.04	V.3.01.05	V.3.01.06	V.4.00
NC30WA	V.5.20 Release1	B	B	B	B	A
	V.5.30 Release1	B	B	B	B	A
	V.5.30 Release02	-	-	-	-	A

A: All the items of information listed in Section 3.1 are portable.
B: Only the paths of assembler and C-language source files are portable.

7.3.4. Load Module Converter

Import Makefile cannot port the information contained in any load module converter (for example, information on options, command executions, or dependencies) into the High-performance Embedded Workshop project. If using a load module converter to create projects in TM, change the settings of the load module converter as follows after completing the Makefile processing:

1. Open the Build menu and select the Build Phases command.
2. The Build Phases dialog box will open.

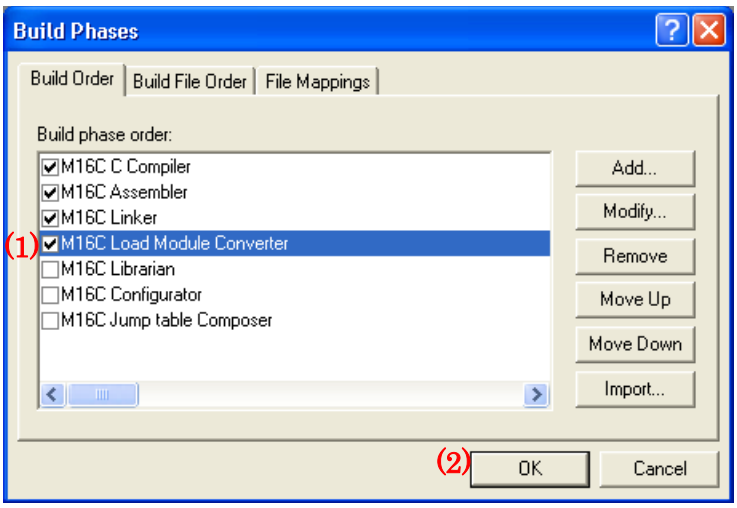

Figure 3 Build Phases Dialog Box

- Select the Mxxx Load Module Converter check box from the Order of Build Phases list.
- Click **OK**.

3. Open the Build menu and select Renesas Mxxx Standard Toolchain.
4. The Renesas Mxxx Standard Toolchain dialog box appears.

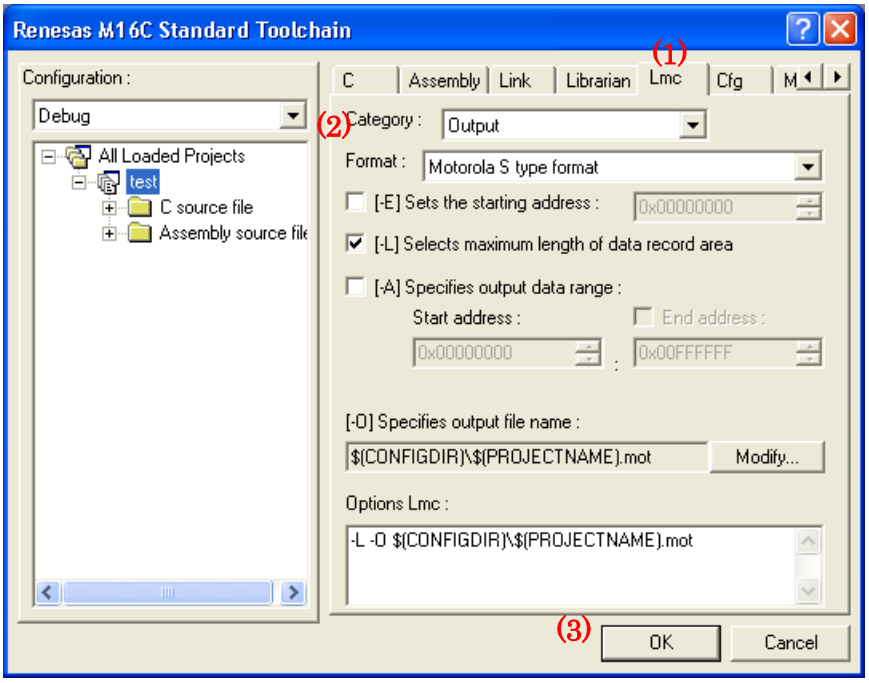

Figure 4 Renesas M16C Standard Toolchain Dialog Box

- Click the Lmc tab.
- Select the Category type from the Category drop-down list.
- Click **OK**.

7.3.5. Other Tools

Import Makefile cannot port any information (options, command executions, dependencies) contained in tools other than the assembler, C compiler, and linker. If any tools other than the assembler, C compiler, linker, and load module converter are used to create projects in TM, custom build phases must be created in High-performance Embedded Workshop. Custom build phases are specifically for operating other tools before, after, or during standard builds (in the assembler, C compiler, and linker).

For more details, see Section 3.2 "Creating Custom Build Phases" in the High-performance Embedded Workshop 4 User's Manual. The following is provided as an example of how to register the cross-reference generation tool xrf30 with High-performance Embedded Workshop.

1. Open the Build menu and select the Build Phases command.
2. The Build Phases dialog box appears; click **Add**.

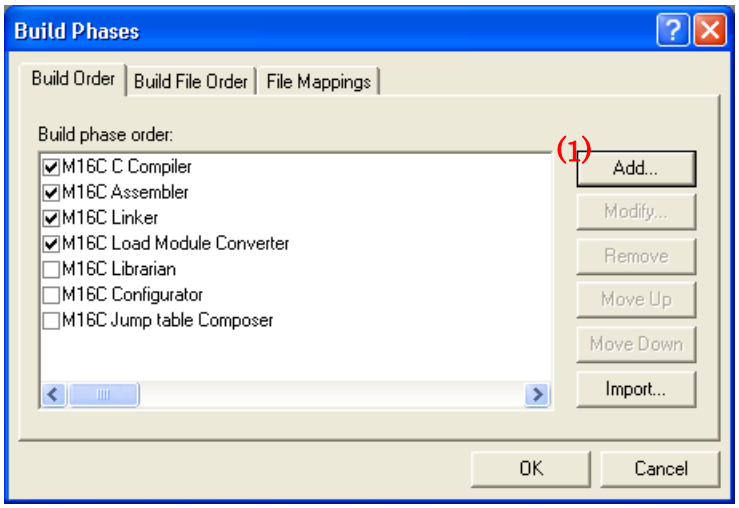

Figure 5 Build Phases Dialog Box

3. The New Build Phase- Step 1/4 wizard opens. Follow the instructions to register the tool as follows:

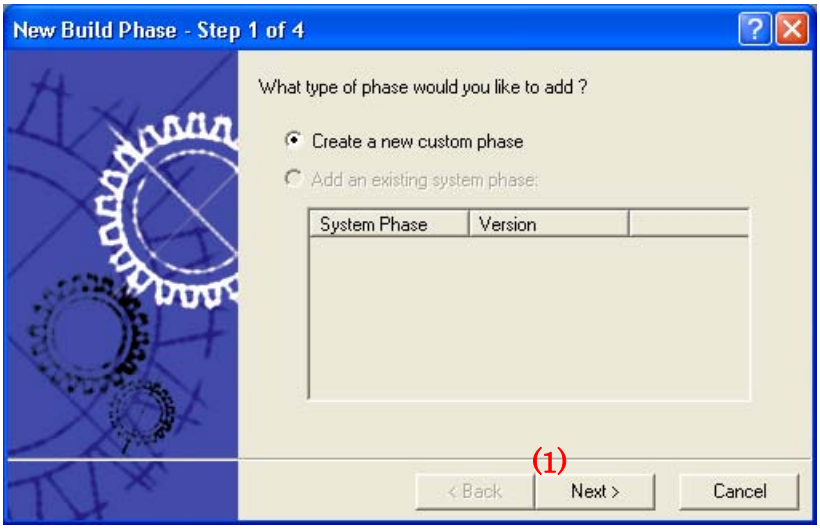

Figure 6 New Build Phase- Step 1/4 Wizard

- Click **Next** (the Create a New Custom Phase check box is selected by default); the New Build Phase-2/4 Step wizard opens.

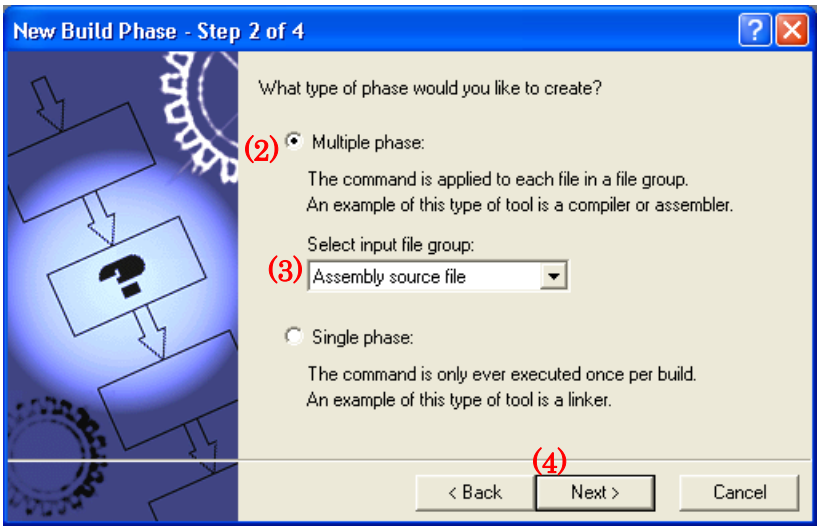

Figure 7 New Build Phase- Step 2/4 Wizard

- In this wizard, select the Multiple Phase check box.
- Select Assembly Source file from the Select input file group.
- Click **Next**; the New Build Phase- Step 3/4 wizard opens.

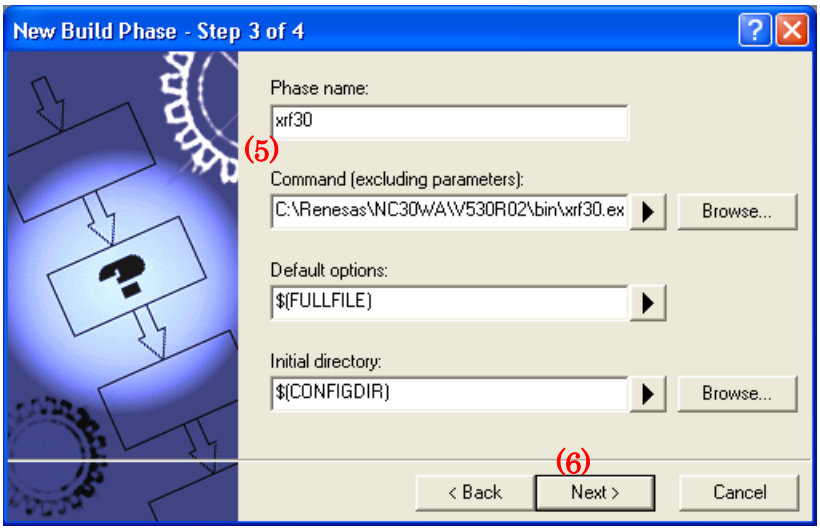

Figure 8 New Build Phase- Step 3/4 Wizard

- Type xrf30 and its fullpath name in the Phase Name and the Command text box.
- Click **Next**; the New Build Phase- Step 4/4 wizard opens.

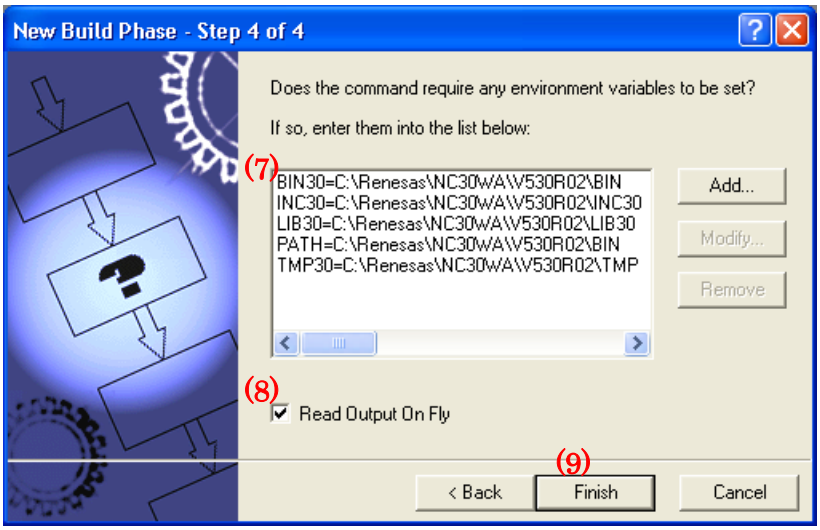

Figure 9 New Build Phase- Step 4/4 Wizard

- In this wizard, enter the necessary environment variables in the list.
- Select "Read Output On Fly" check box.
- Click **Finish**.

4. You return to the Build Phases dialog box at this point, where you can see that xrf30 has been registered as a build phase at the end of the Order of Build phase order.

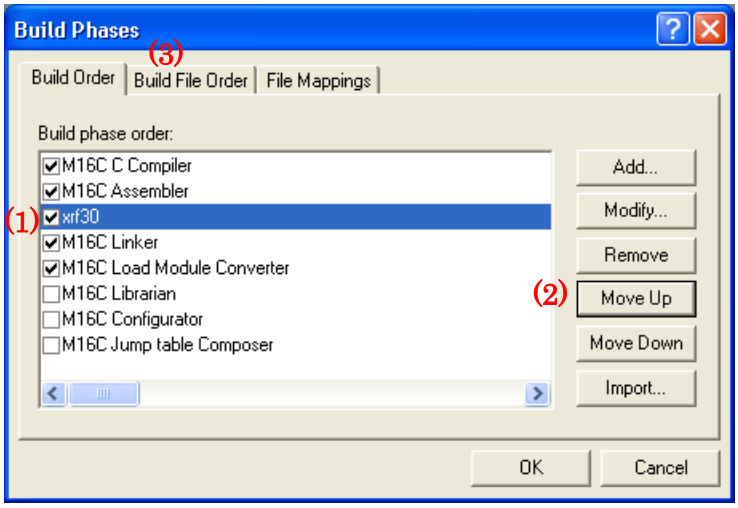

Figure 10 Build Phases Dialog Box (Build Order Tab)

- Select xrf30 from the Order of Build phase order.
- Click **Move Up** to move xrf30 next to the assembler name (see Figure 10).
- Click the Build File Order tab.

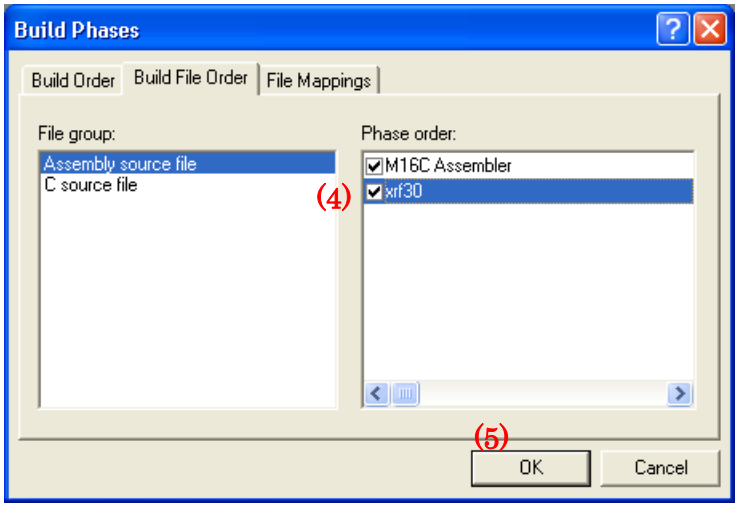

Figure 11 Build Phases Dialog Box (Build File Order Tab)

- Select the xrf30 check box in the Order of Phase order.
- Click **OK**.

5. Open the Options menu and select the xrf30 command.

6. The xrf30 Options dialog box appears; select options as necessary. This setting executes xrf30 for all assembler source files after assemble is completed at a build (before linking files).

7.3.6. Linkage order

Import Makefile cannot port the linking order information to High-performance Embedded Workshop. High-performance Embedded Workshop arranges the linking order alphabetically. To change this order, go through the following steps:

1. Open the Build menu and select the Linkage Order command.
2. The Linkage Order dialog box opens.

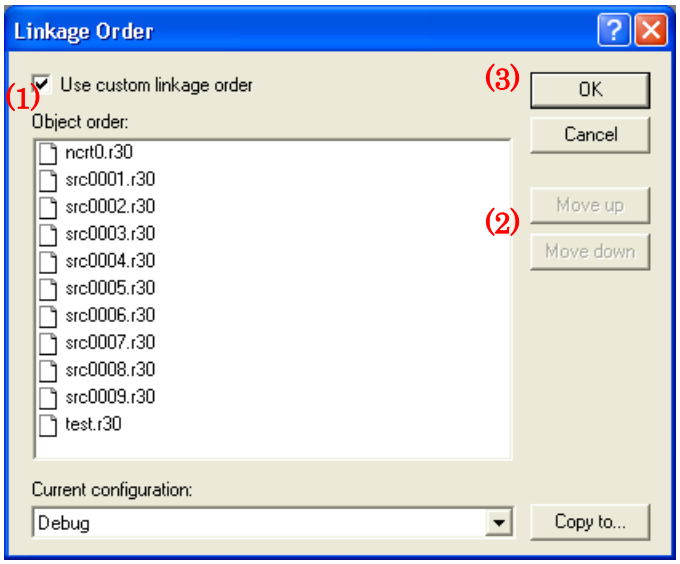

Figure 12 Linkage Order Dialog Box

- Select "Use custom linkage order" check box.
- Select a file from the Object order list, and click **Move up** or **Move down** to move the file. Repeat this step for all files that need to be rearranged.
- Click **OK**.

7.3.7. Placing the Start Up program at the top of Linkage Order

As the Import Makefile cannot port linking order information to High-performance Embedded Workshop, and links are order alphabetically, the start up program may not be placed at the top of the linking order. To place it at the top, follow the steps described previously in Section C.3.7 "Linkage Order."