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.

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

For M16C Series,R8C Family C compiler Package

V.5.45 Release 00

Release note

(Rev.3.00)

The following information has been added to "4.1.3. Fixed Problem" on page 13 in this document.

- With calling the Run-time function.

Renesas Solutions Corporation

October 16. 2009

Abstract

Welcome to M16C Series,R8C Family C compiler Package (M3T-NC30WA) V.5.45 Release 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. About Installation of C compiler Package

For details on how to install, please refer to “Install Guide”.

2. The latest information

Please refer to the following for the latest information on this product.
http://tool-support.renesas.com/eng/toolnews/p_m16c_1.htm

3. Precautions on Product

When using the compiler, please be sure to follow the precautions and suggestions described below.

3.1. About Integrated Development Environment TM

TM does NOT support M3T-NC30WA V.5.45 Release 00.
Therefore, the following cannot be specified.
(1) Create a new project of M3T-NC30WA V.5.45 Release 00 with TM
(2) Port the projects created by TM to High-performance Embedded Workshop

Please refer to “C Compiler Package Guidebook” for the method of porting the project created by TM to High-performance Embedded Workshop.

3.2. About the version of MR30(Real-time OS)

This C compiler can be used with M3T-MR30 V.3.30 Release 1 or later versions.
Caution:
When you install M3T-MR30, please be sure to install in the same directory (bin,lib30,inc30) as this C compiler package.

3.3. Suggestions Concerning File Names

The file names ,directory names and Workspace[1] names that can be specified are subject to the following restrictions:

- The directory, file, or workspace name which comprised of ASCII character-code only can be used.
- Only one period (.) can be used in a file name.
- Network path names cannot be used. Assign the path to a drive name.
- Shortcut cannot be used.
- The "..." symbol cannot be used as a means of specifying two or more directories.

If the limitations above are violated, the following problems may occur.

- The value set by the assembler directive commands .id, .ofsreg, .protect,.rvector or .svector cannot operate correctly. As a result, the ID code and the option function select register may not be set correctly.
- Call Walker and STK Viewer to refer to the stack size are not displayed correctly.

[1] Workspace is a working directory used for processing like the compilation,build or debugging on High-performance Embedded Workshop.

- The MAP Viewer to refer to the map information in the absolute module file isn't displayed correctly.
- The setting by these assembler directive commands isn't displayed in .map file.
- A compile error like "Can't open file" arises.
- A message like "Because a problem occurred, lnxx.exe is terminated." is issued and then the linker is terminated abnormally.
- The file name length including the path should be less than 128 characters.

3.4. Precautions about virus check programs

If the virus detection program is memory-resident in your computer, M3T-NC30WA may not start up normally. In such a case, remove the virus detection program from memory before you start M3T-NC30WA.

3.5. Precautions on M16C Series,R8C Family-Dependent Code

3.5.1. Precautions regarding the M16C interrupt control register

When -O5 optimizing option is used, the compiler generates in some cases BTSTC or BTSTS bit manipulation instructions. In M16C, the BTSTC and BTSTS bit manipulation instructions are prohibited from rewriting the contents of the interrupt control registers.
When using any of the products concerned, ensure that no incorrect code is generated.

- Example

When -O5 optimizing options is used in the program shown below, a BTSTC instruction is generated at compilation, which prevents an interrupt request bit from being processed correctly, resulting in the assembled program performing improper operations.

```
#pragma ADDRESS TA0IC     0055h          /* M16C/62 MCU's Timer A0 interrupt control register */
struct {
            char        ILVL : 3;
            char        IR : 1;          /* An interrupt request bit */
            char        dmy : 4;
} TA0IC;

void        wait_until_IR_is_ON(void)
{
            while (TA0IC.IR == 0)        /* Waits for TA0IC.IR to become 1 */
            {
                        ;
            }
            TA0IC.IR = 0;                /* Returns 0 to TA0IC.IR when it becomes 1 */
}
```

- Workaround

(1) Suppress the generation of the BTSTC and BTSTS instructions resulting from using an optimizing option by selecting the -ONA (or -Ono_asmopt) option together with "-O5" optimizing option.
(2) Add an asm function to disable optimization locally, as shown in the example below.

```
while( TA0IC.IR == 0 )
{
            asm( );
}
```

- Notes

Make sure that no BTSTC and BTSTS instructions are generated after these side-steppings.

3.5.2. Precautions about access of SFR area

You may need to use specific instructions when writing to or reading registers in the SFR area.
Because the specific instruction is different for each model, see the User's Manual for the specific Machine. These instructions should be used in your program using the asm function.

3.5.3. About specifying the interrupt control register

M3T-NC30WA supports the functions that set or change the value of an interrupt priority level to conform to MESC TECHNICAL NEWS(No. M16C-14-9805).

- case of setting the value

Please use SetLevel function. In this time, please be sure to include "intlevel.h" file.

SetLevel(char *adr, char val);

adr : Address of the interrupt control register
val : value

- case of changing the value

Please use ChgLevel function. In this time, please be sure to include "intlevel.h" file.

ChgLevel(char *adr, char val) ;

adr : Address of the interrupt control register
val : value

[Example]

```
#include <intlevel.h>
#pragma ADDRESS timerA 55H
char   *timerA;
void   func(void)
{
        SetLevel(timerA,2);     // Specifying the value "2" to the interrupt priority level
                :
        ChgLevel(timerA,4);     // Changing the value "4" to the interrupt priority level
}
```

3.5.4. Regarding M16C/62 4M extended mode

Make sure the program is located in the internal ROM.

3.5.5. Section FirmRam_NE and the Value of the SB Register when the On-Chip Debugger is Selected

If you select the debugger on the dialog box selecting OnChipDebegger when creating a new project workspace, FirmRam_NE section may be allocated from 400H. Then, you can't access the correct area by using the SB relative addressing mode, because the initial value for SB register is set to 400H.
If section FirmRam_NE starts from 400H as the result of linkage, the initial value for the SB register should be changed to the value of the start address of the bss_SE section. For the start address of the bss_SE section, see the contents of the map file.

The values indicated below should be changed to the start address of the bss_SE section.

<resetprg.c>

```
void start(void)
{
        :
        _sb_    = 0x400; // 400H fixation (Do not change)

}
```

<resetprg.h>

```
#define  DEF_SBREGISTER        _asm("      .glb     __SB__¥n"¥
                                    "__SB__  .equ     0400H")
```

MCU Concerned as of May 16, 2009

M16C/26, M16C/26A, M16C/28, M16C/29,
M16C/30P,
M16C/62P,
M16C/6N4, M16C/6N5, M16C/6NK, M16C/6NL, M16C/6NM, M16C/6NN,
M16C/6S,
M16C/64,
M16C/64A,
M16C/65

3.5.6. About M16C/64,M16C/64A,M16C/65

If you select "C source startup Application" when creating a new workspace, the generated C-language startup "resetprg.c" includes the following data for the user boot function.

```
#pragma sectaddress _UB_section_FE,ROMDATA 0x13FF0
#pragma section rom _UB_section
struct      _UB_struct {
            unsigned char code[8];
            unsigned char _near *addr;
            unsigned char bit;
            unsigned char level;
            unsigned char reserved[4];
} const _far _UB_data = {
            { 0xffU,0xffU,0xffU,0xffU,0xffU,0xffU,0xffU,0xffU},        // user boot code
            (unsigned char _near *)0xffffU,                            // Port address
            0xffU,                                                     // Port bit
            0xffU,                                                     // Port Level
            {0xffU,0xffU,0xffU,0xffU}                                  // reserve
};
```

If you are not using the program ROM2 area, comment out the above like #if 0 - #endif.
If the above is enabled, when you program data of the mot file into flash memory by FDT, the message saying "The image contained no valid data - operation cancelled" is output.

3.5.7. Precaution About .ID , .protect and .ofsreg

If you are using an MCU that has the option function select register, option function select area, ROM code protect control address, and ID code check function, make sure to set the value of each area by the assembler pseudo-command.

When using the R8C Family:

the option function select register : .ofsreg
ID code check function : .ID

When using the M16C Series:

ROM code protect control address : .protect
ID code check function : .ID

If the above-mentioned pseudo-command is not set, the value of each area is set to "00".

3.6. Precautions about Compiler, Assembler, Linkage Editor and Utilities

3.6.1. About -Oglobal_jmp(-OGJ)

If the compiler option -Oglobal_jmp(-OGJ) , the assembler option -JOPT and the link option -JOPT are used and the link option -ORDER or -LOC is specified more than one time, only either -ORDER or -LOC that is specified last time becomes effective and a linkage error occurs.

As a result:
If -ORDER is specified more than one time, a linkage error will occur.
If -LOC is specified more than one time, allocation will not be done properly.

Please be sure to specify -ORDER and -LOC respectively one by one.

3.6.2. About Using Inline Functions –OLU and -OFFTI at the Same Time

If the compiler option -Oloop_unroll(-OLU) and -Oforward_function_to_inline(-OFFTI) are used at the same time, and a function that is expanded inline is described in the loop statement, the following error may occur:

Error (asp30): Undefined symbol exist 'Symbol name'

If this error occurs add a dummy asm function in the statement.

3.6.3. About -ffar_pointer (-fFP)

If -ffar_pointer is used, be aware that when the & operator that acquires the address of a near attribute variable is used, it is handled in 16–bit address. Make sure that it is cast with the far pointer prior to the & operator.
Note also that if the pointer size is acquired with sizeof, the return value is 2. If any function without prototype declaration is called, only 2 bytes of address are stacked. Always be sure to declare function prototypes.

3.6.4. Precaution for Assembler start-up files (ncrt0.a30, sect30.inc,nc_define.inc)

The content of start-up files may be customized depending on the target MCU or application.
Please refer to the hardware manual or the datasheet of the target MCU when undergoing such customizations.

3.6.5. About the standard I/O function

The standard I/O functions consume much RAM. If you use the standard I/O functions in your program for R8C Family, you cannot use %f,%E,%e,%g,%G for printf.

3.6.6. Precautions about the search of an include file

If you specify a file to include with a drive name in the #include line, and attempt to compile the file from a directory different from the one in which the file to compile is present, instances may occur in which the file to

include cannot be found.

Example

```
#include "c:\user\test\sample.h"
main(){}

C:\user>nc30    \user\test2\sample.c -silent \user2\tm_test\aa.c
[Error(cpp30.21):\user2\test2\sample.c, line 1] include file not found 'c:\user\test\sample.h'
```

3.6.7. Precautions to be taken when using #pragma ASM/ENDASM and asm()

- Regarding debug information when using #pragma ASM/ENDASM and asm() outside functions, if you write #pragma ASM anywhere outside functions, no C source line information will be output. For this reason, information regarding descriptions in #pragma ASM to #pragma ENDASM, such as error message lines when assembling or linking and line information when debugging, may not be output normally.
- C compilers generate code of arguments to be passed via registers and of register variables by analyzing their scopes. However, if manipulations of register values are described using inline assemble features(such as #pragma ASM / #pragma ENDASM directives and asm function), C compilers cannot hold information on the scopes of the above-mentioned arguments and register variables. So, be sure to save and restore register contents on and from the stack when registers are loaded using inline assemble functions described above.

3.6.8. Precautions about debugging of a program using _Bool type

When you debug the program which uses the _BOOL type, please confirm whether the debugger supports the _BOOL type.
In using the debugger which does not support the _BOOL type, please use a debugging option"-gbool_to_char (-gBTC)" at the time of compile.

3.6.9. Precautions regarding the preprocessing directive #define

To define a macro which will be made the same value as the macro ULONG_MAX, always be sure to add the suffix UL.

3.6.10. Precautions on macro definition

If the name of a macro itself is used in the content of a macro definition and the defined macro is specified in an argument to other function-like macro, macro replacement cannot be executed correctly.

- Example

```
int     a = 10;
#define a       a + a                   // macro name 'a'
#define p( x,y ) x + y

void    func(void)
{
        int     i = p ( a , a );        // results in i = 80
}                                       // i = 40 is correct
```

- Workaround

Make sure the macros passed to the arguments to function-like macros are defined with a name that is not used in the macro definition.

```
int     a = 10;
#define b       a + a                   // Change to a macro name that is not 'a'
#define p( x,y ) x + y

void    func(void)
{
        int     i = p ( b , b );
}
```

3.6.11. Precautions on #if preprocessing directive

If a constant expression of #if directive is a shift whose left operand is a negative value and right operand is a value of unsigned type, the result of the shift expression cannot be worked correctly.

- Example

```
void    func( void )
{
        char    a;

#if  (-1 << 1U ) > 0              // Determined to be true
        a = 1;                    // (–1 << 1U) is –2, so that it correctly is false
#else
        a = 2;
#endif
}
```

- Workaround

If the left operand of a shift is a negative value, change the right operand of that shift to a value of signed type.

```
void    func( void )
{
        char    a;

#if  (-1 << 1 ) > 0       // Disuse of the suffix U changes the right operand of
        a = 1;            //  a shift to signed type.
#else
        a = 2;
#endif
}
```

3.6.12. Calls to functions that return a structure result in a system error.

System Error occurs when a return value of a function which returns a structure is used to initialize an auto structure variable.

Example

```
typedef struct tag{
        long abc;
}st;

st func(int);

void main(){
        st st1 = func(10);
}
```

Workaround

Make sure that structure variables of storage class auto are defined separately from the initialization of those variables.

```
void main(){
     st st1;
     st1 = func(10);
}
```

3.6.13. Precaution about - Ostack_frame_align (-OSFA)

If compile option -Ostack_frame_align (-OSFA) is used, incorrect values of stack size may be provided in inspector information and the stack size display file (with extension .stk). As a result, the values of stack size calculated by the STKViewer and CallWalker (utilizing inspector information) and the stk30 stack size calculate utility (utilizing the stack size display file) will be false.

[Tool News: http://tool-support.renesas.com/eng/toolnews/070701/tn5.htm]

Conditions

This problem occurs if the following conditions are all satisfied:

(1) Compile option -Ostack_frame_align (-OSFA) is selected.

(2) Compile option -genter is not selected.

(3) In the program exists a function that does not make any stack frame.

In order that no function can make the stack frame, the following three conditions must be met:

- The function does not take arguments passed via the stack.
- In the function exist no auto variables (except for the ones assigned to registers) or auto variables have been deleted by the optimization of the compiler.
- The compiler does not create any temporary variables.

Example

```
void   sub(unsigned int);

void   func(void)            /* Condition (3) */
{
       sub(10);
}
```

Workaround

Do Not select compile option -Ostack_frame_align (-OSFA), or use -genter along with -Ostack_frame_align (-OSFA),.

3.6.14. Precaution about performing right-shift operations

Under the condition that an optimizing option is selected at compilation, System Error may arise if the result of shifting a 32-bit data piece to the right by the number of bits within a range of 11 to 15 is directly stored in or cast to a variable of 16 bits long.

[Tool News: http://tool-support.renesas.com/eng/toolnews/070716/tn4.htm]

Conditions

This problem may occur if the following conditions are all satisfied:

(1) Any of the following optimizing options is selected to enhance the speed of program execution and minimize ROM consumption: -O, -OR, and -OR_MAX

(2) A data piece of 32 bits long is shifted right by the number of bits within a range of 11 to 15.

(3) The result in (2) above is directly stored in or cast to a variable of 16 bits long.

(4) The compiler specifies the register in which the data piece to be shifted in (2) and the 16-bit variable in (3) are stored.

Example

```
int i;
long l;

    i = (int)( l >> 15 );
```

Workaround

Do not directly store the result of right-shifting a 32-bit data piece in or cast it to a 16-bit variable, but assign it to a 32-bit variable and then store this variable in a 16-bit variable.

Modification of the above example

```
int i;
long l, ll;

    ll = l >> 15;       /* Assign the result to a 32-bit variable */
    i = (int)ll;
```

3.6.15. Precaution about utl30

C compiler user's manual "Appendix G the SBDATA declaration & SPECIAL page Function declaration utility (utl30)" on Page 357 has the statement that "Includes, during startup (sect30.inc), the SPECIAL Page vector definition file (special.inc) as a file to be included". But, this is the explanation for the version older than V.5.40. The SPECIAL Page vector definition file is unnecessary in V.5.40 or later. Therefore, please do not use it.

3.6.16. Precaution about MapViewer and StkViewer

As you cannot use Online Help of the MapViewer and StkViewer with a PC running Windows Vista(R), please use that of the EcxMap and CallWalker instead.

3.6.17. Precaution about malloc(), calloc() and realloc()

Memory management function malloc ,calloc and realloc of the NC30WA cannot secure the area of 64KB or more at a time.

3.6.18. Precaution about Call Walker

Values indicated by Call Walker are not strictly accurate so simply use them for reference when you examine the size of the stack space. Careful evaluation is needed if you have decided the actual size of the stack space according to the information indicated by Call Walker.

4. Contents of upgrade from V.5.44 Release 00

4.1. Contents of upgrade about C compiler

4.1.1. Function Improvement

- Assembler Optimizer Improved
- The length of command line Improved
  The maximum number of the letters that can be specified with command-line has been expanded from 256 to 2048.

4.1.2. Addition of C-language startup files.

The C-language startup files of the following CPU group are added.
M16C/64A

4.1.3. Fixed Problems

The following known problems have been fixed

- With Using SQMlint, the MISRA C rule checker
  [RENESAS TOOL NEWS]
  http://tool-support.renesas.com/eng/toolnews/080416/tn2.htm
- With using a volatile-qualified variable in the for or while statement
  [RENESAS TOOL NEWS]
  http://tool-support.renesas.com/eng/toolnews/080616/tn1.htm
- With comparing a bit field variable with 1 or 0
  [RENESAS TOOL NEWS]
  http://tool-support.renesas.com/eng/toolnews/080616/tn2.htm
- With using a variable volatile-qualified with the indirect member operator
  [RENESAS TOOL NEWS]

http://tool-support.renesas.com/eng/toolnews/080616/tn3.htm

- With using a volatile-qualified variable in the bit-wise and operation
  [RENESAS TOOL NEWS]
  http://tool-support.renesas.com/eng/toolnews/080616/tn4.htm
- With using the strcmp function
  [RENESAS TOOL NEWS]
  http://tool-support.renesas.com/eng/toolnews/080616/tn5.htm
- With division operations between single-precision floating-point numbers
  [RENESAS TOOL NEWS]
  http://tool-support.renesas.com/eng/toolnews/090116/tn3.htm
- On placing functions in sections
  [RENESAS TOOL NEWS]
  http://tool-support.renesas.com/eng/toolnews/n051101/tn8.htm
- On nesting inline functions
  [RENESAS TOOL NEWS]
  http://tool-support.renesas.com/eng/toolnews/n041116/tn6.htm
- With using the Scan All Dependencies function on High-performance Embedded Workshop
  [RENESAS TOOL NEWS]
  http://tool-support.renesas.com/eng/toolnews/090201/tn4.htm

- With calling the Run-time function.
  [RENESAS TOOL NEWS]
  http://tool-support.renesas.com/eng/toolnews/090516/tn1.htm

5. Conformance with MISRAC Rule

5.1. Standard Function Library

In C-Source code of standard function library M3T-NC30WA, it is found that some rules[2] are against the MISRAC Rule, but these violations do not constitute a drawback to any operation.

5.1.1. Cause of Rule Violation

In C-Source code of standard function library M3T-NC30WA, the major causes for rule violation are as follows:

- C-Compiler specifications (near/far modifier, asm () function and #pragma)
- Declaration of function based on ANSI Standard
- The evaluation sequence in the conditional statement is not described explicitly, using a parenthesis.
- Implicit type conversion

5.1.2. Inspection No. running counter to the rule

The following are Inspection Nos. that run counter to the Rule:

1	12	13	14	18	21	22	28	34	35
36	37	38	39	43	44	45	46	48	49
50	54	55	56	57	58	59	60	61	62
65	69	70	71	72	76	77	82	83	85
99	101	103	104	105	110	111	115	118	119
121	124								

5.1.3. Evaluation Environment

Compiler	M3T-NC30WA V.5.30 Release 1
Compile Option	-O -c -as30 "-DOPTI=0" -gnone -finfo -fNII -misra_all -r $*.csv
MISRA C Checker	SQMlint V.1.00 Release 1A

5.2. Conformance with MISRA C Rule in HEW Generation Source Code

In C-Source code that HEW (High-performance Embedded Workshop) generates automatically, it is found that some rules are against the MISRAC Rule, but these violations do not constitute a drawback to any operation.

5.2.1. Cause of Rule Violation

In C-Source code that HEW generates, the major causes for rule violation are as follows:

- C-Compiler specifications (#pragma etc.)
- cope of variable defined by Header-File
- Definition of type used in Bit-Field

5.2.2. Inspection No. running counter to the rule

The following are Inspection Nos. that run counter to the Rule:

13	14	22	34	36	37	43	45	46
49	54	59	69	76	82	85	99	104
110	111	115	124	126				

[2] These results were produced after inspection using MISRAC Rule Checker SQMLint.

5.2.3. Evaluation Environment

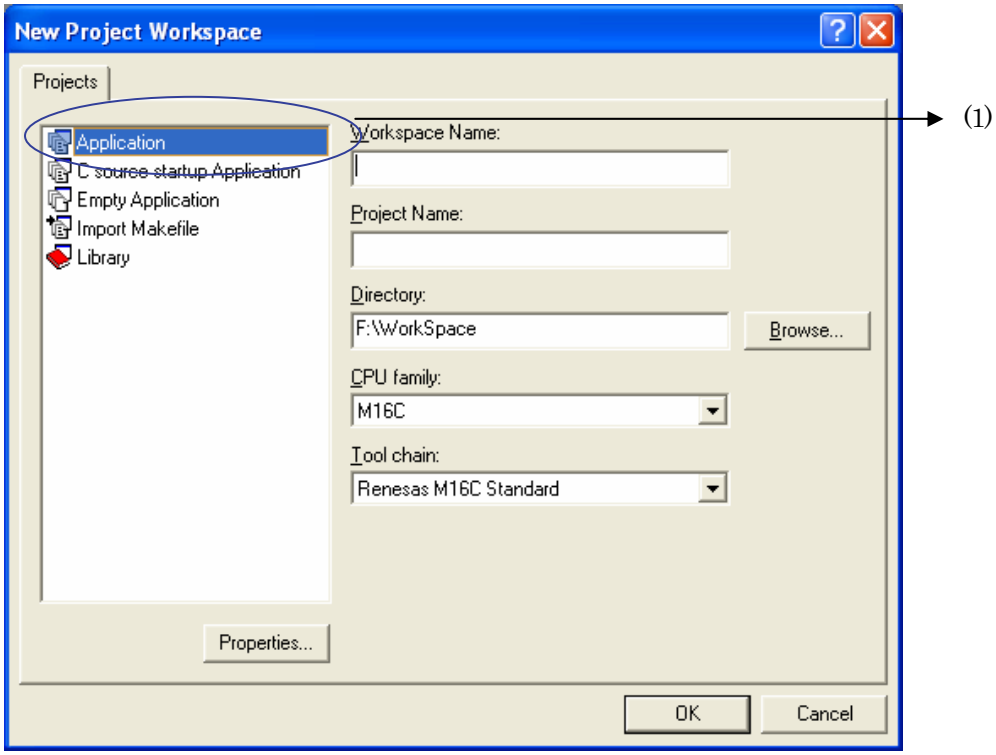
Compiler	M3T-NC30WA V.5.45 Release 00
Compile Option	-c -misra_all
MISRA C Checker	SQMlint V.1.03 Release 00

5.2.4. #pragma extended functions for use in C start-up (Misra C rule 99)

Extended Function	Definition File	Description	Function
#pragma STACKSIZE	resetprg.h	Defines the user stack size.	The stack section (stack) is output and the top label name of the stack is generated.
#pragma ISTACKSIZE	resetprg.h	Defines the interrupt stack size.	The interrupt stack section (istack) is output and the top label name of the interrupt stack is generated.
#pragma CREG	resetprg.h	Declares an internal register of the MCU.	A special instruction is used to generate code for access to an internal register declared by this pragma.
#pragma sectaddress	resetprg.h fvector.c	Defines a section. Its address can also be declared at the same time.	The section name declared by this pragma is used to define a section. When its address is specified at the same time, an address definition using a pseudo instruction ".org" is output.
#pragma entry	resetprg.h	Declares a function to be executed at the time of a reset.	An enter instruction to configure a stack frame for the function declared by this pragma is not output. This is because the enter instruction should not be generated before the stack pointer is initialized.
#pragma interrupt/V	fvetor.c	Generates a vector table.	Only the interrupt vector is defined for the function declared by this pragma.
#pragma inline	resetprg.h	Declares an inline function.	The function declared by this pragma is inline-expanded.
#pragma interrupt	intprg.c fvector.c	Declares an interrupt function.	Interrupt-function code is generated for the function declared by this pragma.
#pragma interrupt	heap.c resetprg.c initsct.h resetprg.c firm.c	Changes the name of a section.	The section name is changed to the one defined by this pragma.
#pragma ADDRESS	Each sfr header file	Defines the I/O address and declares a variable.	.equ is used to define the I/O address for sfr defined by this pragma.

6. C-language Startups

V.5.40 Release 00(A) or earlier version of compilers does not support startup programs written in C language.
Please note that the conventional startups written in assembler such as ncrt0.a30 , sect30.inc and nc_define.inc can be used the same way as in the past.
To use the conventional ncrt0.a30 , sect30.inc and nc_define.inc select Application indicated by (1) in the new project workspace below.



6.1. Generated Files

The C-language startup includes the following files:

(1) resetprg.c
Initializes the microcomputer.

(2) initsct.c
Initializes each section (by clearing them to 0 and transferring initial values).

(3) heap.c
Reserves storage for the heap area.

(4) fvector.c
Defines the fixed vector table.

(5) intprg.c
Declares the entry function for variable vector interrupts.

(6) firm.c/firm_ram.c
Reserves storage for the program and workspace areas used by firm of FoUSB/E8 as dummy areas when OnChipDebugger is selected. (Please do not alter the file.)

(7) cstartdef.h
Defines the sizes of stack and heap

(8) initsct.h

Contains statements for the processes (assembler macros) that initialize each section. (Please do not alter the file.)

(9) resetprg.h

Includes each header file.

(10) typedef.h

Declares each type by typedef. (Please do not alter the file.)

(11) sfrXX.h,sfrXX.inc

The sfr definition header file of CPU chosen when a project was created is registered to the work space.

6.2. Processing of Each Generated File

- resetprg.c (essential)

The content of this file varies with the selected MCU (M16C or R8C).

```
#pragma section program interrupt ..... ①

void start(void) ..... ②
{
        _isp_   = &_istack_top;     // set interrupt stack pointer ..... ③
        prcr    = 0x02;             // change protect mode register ..... ④
        pm0     = 0x00;             // set processor mode register ..... ⑤
        prcr    = 0x00;             // change protect mode register ..... ⑥
        _flg_   = __F_value__;      // set flag register ..... ⑦
#if __STACKSIZE__!=0
        _sp_    = &_stack_top;      // set user stack pointer ..... ⑧
#endif
        _sb_    = 0x400;  // 400H fixation (Do not change) ..... ⑨

        // set variable vector's address
        _asm("  ldc     #((topof vector)>>16)&0FFFFh,INTBH"); ..... ⑩
        _asm("  ldc     #(topof vector)&0FFFFh,INTBL");

        initsct();              // initialize each sections ..... ⑪
#if __STACKSIZE__!=0
        _sp_    = &_stack_top;      // set user stack pointer
#else                                                          ⑫
        _isp_   = &_istack_top;     // set interrupt stack pointer
#endif

#if __HEAPSIZE__ != 0
        heap_init();                // initialize heap ..... ⑬
#endif
#if __STANDARD_IO__ != 0
        _init();                    // initialize standard I/O ..... ⑭
#endif
        _fb_ = 0;  // initialize FB registe for debugger
        main();                     // call main routine ..... ⑮

        exit();             // call exit
}
```

(1) The startup function is located in the interrupt section.
(2) The function body of the CPU initialization function start() is defined.
(3) Initializes the interrupt stack pointer.
(4) Sets the protect register to "Write-enabled".
(5) Sets the processor mode register to "single-chip mode."
If modes need to be changed, this expression must be altered.
(6) Sets the protect register to "Write-inhibited".
(7) Sets the U flag.
If you chose "Use the user stack" in the workspace creation wizard, the user stack pointer is set
(8) Initializes the user stack pointer if you chose "Use the user stack" in the workspace creation wizard.
(9) Sets the SB register to address 0x400 (which sets the start address of RAM).
(10) Sets the variable vector address in the INTB register.

(11) Initializes each section (by clearing them to 0 and transferring initial values).

(12) Initializes the stack pointer again after the initialization of sections.
(13) Initializes the heap area.
If memory management functions are used, call to this function must be enabled.
(14) Initializes the standard input/output library
If standard input/output functions are used, call to this function must be enabled.
(15) Calls the main function.

- initsct.c (essential)
The content of this file varies with the selected MCU (M16C or R8C).

```
void initsct(void)
{
        sclear("bss_SE","data,align");          ----------(1)
        sclear("bss_SO","data,noalign");
        sclear("bss_NE","data,align");
        sclear("bss_NO","data,noalign");
#ifndef __NEAR__
        sclear_f("bss_FE","data,align");        ----------(2)
        sclear_f("bss_FO","data,noalign");
#endif
        // add new sections
        // bss_clear("new section name");

        scopy("data_SE","data,align");          ----------(3)
        scopy("data_SO","data,noalign");
        scopy("data_NE","data,align");
        scopy("data_NO","data,noalign");
#ifndef __NEAR__
        scopy_f("data_FE","data,align");        ----------(4)
        scopy_f("data_FO","data,noalign");
#endif
}
```

(1) sclear: Clears the bss section of the near area to zero.
If the bss section name is altered or a new bss section name is added using the #pragma SECTION bss feature, NE and NO must be altered or added in pairs.

sclear("section name_NE," "data,align");
sclear("section name_NO," "data,noalign");

Example: When a section is added by #pragma section bss bss2, the following must be added to initsct.c
sclear("bss2_NE," "data,align");
sclear("bss2_NO," "data,noalign");

(2) sclear_f: Clears the bss section of the far area to zero.
If an external variable without initial values is declared using the far qualifier, this macro function must be enabled. This option is invalid only when -R8C option is specified.

(3) scopy: Transfers initial values to the data section of the near area.
If the data section name is altered or a new dada section name is added using the #pragma SECTION data feature, NE and NO must be altered or added in pairs.
scopy("section name_NE," "data,align");
scopy("section name_NO," "data,noalign");

Example: When a section is added by #pragma section data data2, the following must be added to initsct.c
scopy("data2_NE," "data,align");
scopy("data2_NO," "data,noalign");

(4) scopy_f: Transfers initial values to the data section of the far area.
If an external variable with initial values is declared using the far qualifier, this macro function must be enabled. This option is invalid only when -R8C option is specified.

- heap.c (only when memory management functions such as malloc are used)

```
#pragma SECTION  bss        heap          ----------(1)

_UBYTE heap_area[__HEAPSIZE__];    ----------(2)
```

(1) Locates the heap area in the heap_NE section.
* If the heap size consists of an odd number of bytes, the heap_NO section is assumed by default.
(2) Reserves storage for the heap area by an amount equal to the size defined in __HEAPSIZE__.

- fvector.c (essential)

```
#pragma sectaddress             fvector,ROMDATA   Fvectaddr       ----------(1)

////////////////////////////////////////////////////////////////

#pragma interrupt/v _dummy_int        //udi                      ----------(2)
#pragma interrupt/v _dummy_int        //over_flow
#pragma interrupt/v _dummy_int        //brki
#pragma interrupt/v _dummy_int        //address_match
#pragma interrupt/v _dummy_int        //single_step
#pragma interrupt/v _dummy_int        //wdt
#pragma interrupt/v _dummy_int        //dbc
#pragma interrupt/v _dummy_int        //nmi
#pragma interrupt/v start                                        ----------(3)
```

(1) Outputs the section and address of a fixed vector table.
* This pragma is used exclusively for startup and cannot normally be used.

(2) Fills fixed vectors other than reset with a dummy function (_dummy_int).
* #pragma interrupt/v Function_Name
The Function_Name is registered in the vector table. When the function is defined, this definition needs #pragma interrupt.

(3) Defines the entry function.
The function to be executed upon reset is registered in a fixed vector.

- intprg.c (This file may be needed depends on the target MCU)

```
// DMA0 (software int 8)
#pragma interrupt  _dma0(vect=8)      ----------①
void _dma0(void){}

// DMA1 (software int 9)
#pragma interrupt  _dma1(vect=9)
void _dma1(void){}

// DMA2 (software int 10)
#pragma interrupt  _dma2(vect=10)
void _dma2(void){}

// DMA3 (software int 11)
#pragma interrupt  _dma3(vect=11)
void _dma3(void){}

        (Skipped)
```

(1) Declares the variable vector interrupt function.
The functions corresponding to each variable vector interrupt function are declared. A variable vector table is

generated at the same time.

(2) Defines the variable vector interrupt function.
Please write the content of processing in the functions corresponding to the interrupt vector numbers used.

Example: To use interrupt vector number 9 (DMA1)

```
#pragma interrupt _dma1(vect=9)
void _dma1(void)
{
        // Omission
}
```

(3) If intprg.c is unnecessary
Please remove it from file registration to exclude it from the target to be linked.

- firm.c/firm_ram.c (Only when on chip debugger is selected)

DO NOT change the content of this file directly.

The content is altered automatically depending on the target MCU and selected FoUSB/E8

```
#ifdef __E8__          // for E8                                    ----------(1)

#pragma section bss FirmArea                                       ----------(2)

#ifndef __WORK_RAM__
#define __WORK_RAM__ 0x80
#endif

_UBYTE _workram[__WORK_RAM__];                                     ----------(3)

#pragma section bss FirmArea                                       ----------(4)
_far _UBYTE _firmarea[0x800];          // dummy for monitor        ----------(5)

#else       // for FoUSB

#pragma section bss FirmRam                                        ----------(6)
_UBYTE _workram[0x80];    // for Firmware's workram                ----------(7)

#pragma section bss FirmArea                                       ----------(8)
_far   _UBYTE _firmarea[0x600];        // dummy for monitor        ----------(9)
#endif
```

(1) Enables E8 when it is to be used
(2) Allocates the work ram area to be used by the E8 firmware in the FirmRam_NE section.
(3) Reserves the area of work ram for the size defined as __WORK__RAM__.
(4) Locates the firmware program of E8 in the FirmArea section.
(5) Specifies the size of the firmware program.
(6) Allocates the work ram area to be used by the FoUSB firmware in the FirmRam_NE section.
(7) Reserves 0x80 bytes of storage for the work ram area. (It depends on the corresponding microcomputer type)
(8) Locates the firmware program of FoUSB in the FirmArea section.

(9) Specifies the size of the firmware program.

- cstartdef.c (essential)

```
#define __STACKSIZE__          0x80 ----------①
#define __ISTACKSIZE__         0x80 ----------②
#define __HEAPSIZE__           0x80 ----------③
#define __STANDARD_IO__        0    ----------④
#define __WATCH_DOG__          0    ----------⑤
```

(1) Varies according to the stack size that you entered in the creating workspace wizard.
(2) Varies according to the interrupt stack size that you entered in the creating workspace wizard.
(3) Varies according to the heap size that you entered in the creating workspace wizard.
(4) Set to 1 if you chose to "Use the standard input/output function" in the creating workspace wizard.
(5) Should be set to 1 if the WATCHDOG feature needs to be enabled immediately after reset.
(R8C Family only)
If you want to change the above again after you've created a new workspace, be sure to change this file directly

- initsct.h (essential)

  Please do not alter the content of this file.

- resetprg.h (essential)

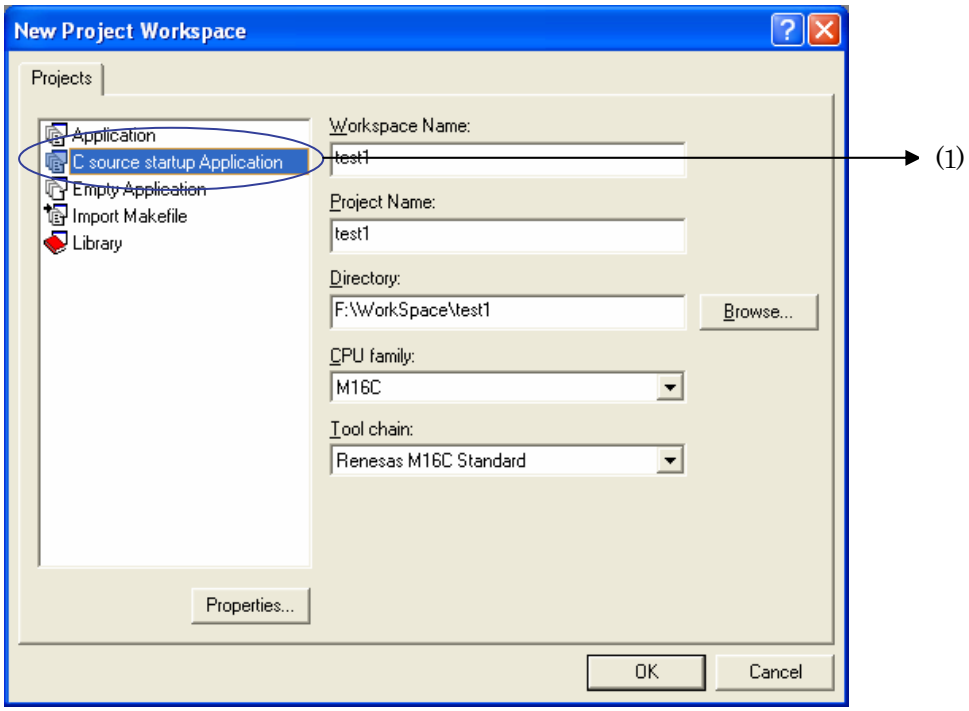
  If using On-Chip Debugger, please refer to the section "3.3.5. Section FirmRam_NE and the Value of the SB Register when the On-Chip Debugger is Selected" in this release note.

- typedef.h (essential)

  Please do not alter the content of this file.

6.3. How to Generate a C-language Startup

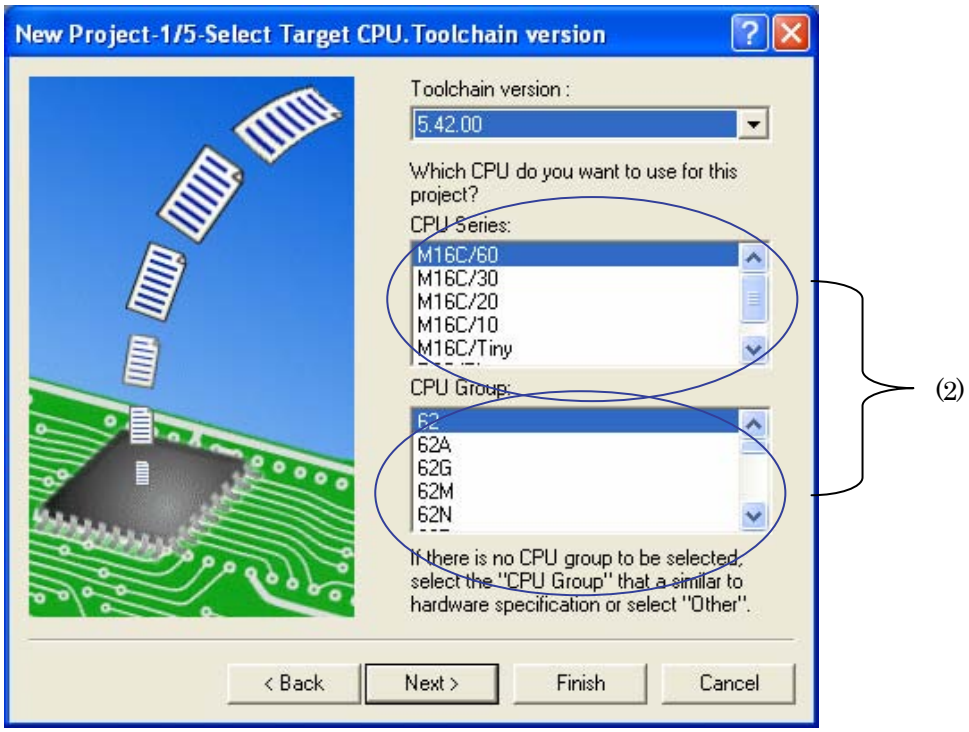
- Selecting the project that uses a C-language startup.



(1) Select C source startup Application in the left-side window.

After selecting "C source startup Application", "Application" will be automatically selected if you change the default setting of "CPU family". Please select "C source startup Application" again.

- Selecting the type of microcomputer

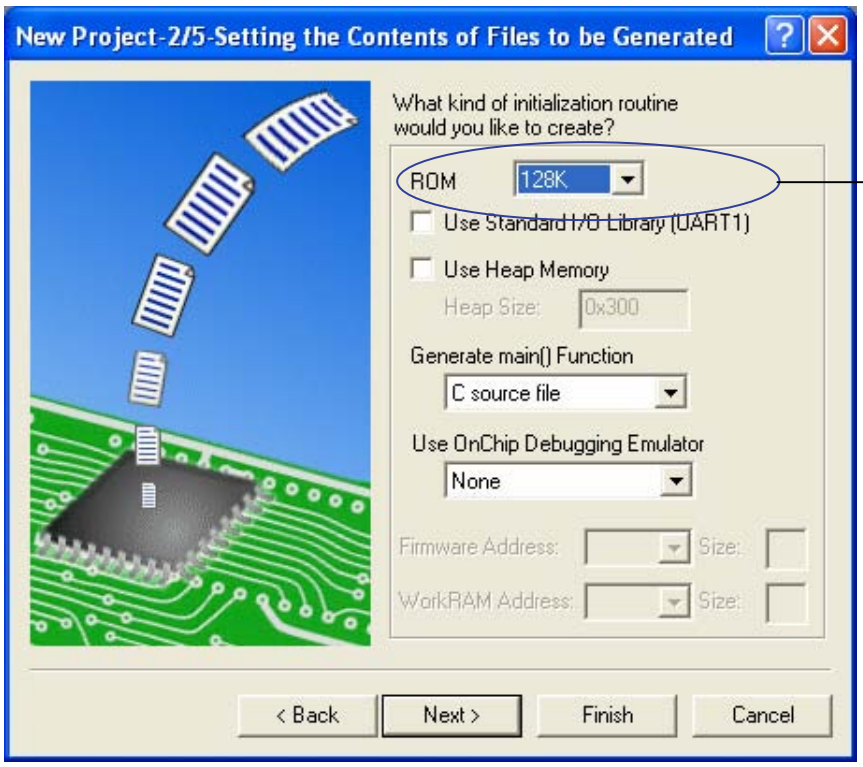


(2) Select the type of microcomputer from CPU Series and CPU Group.

When a type of microcomputer is selected, its corresponding sfr header file is copied to the workspace. Furthermore, a variable vector table (intprg.c) is registered.

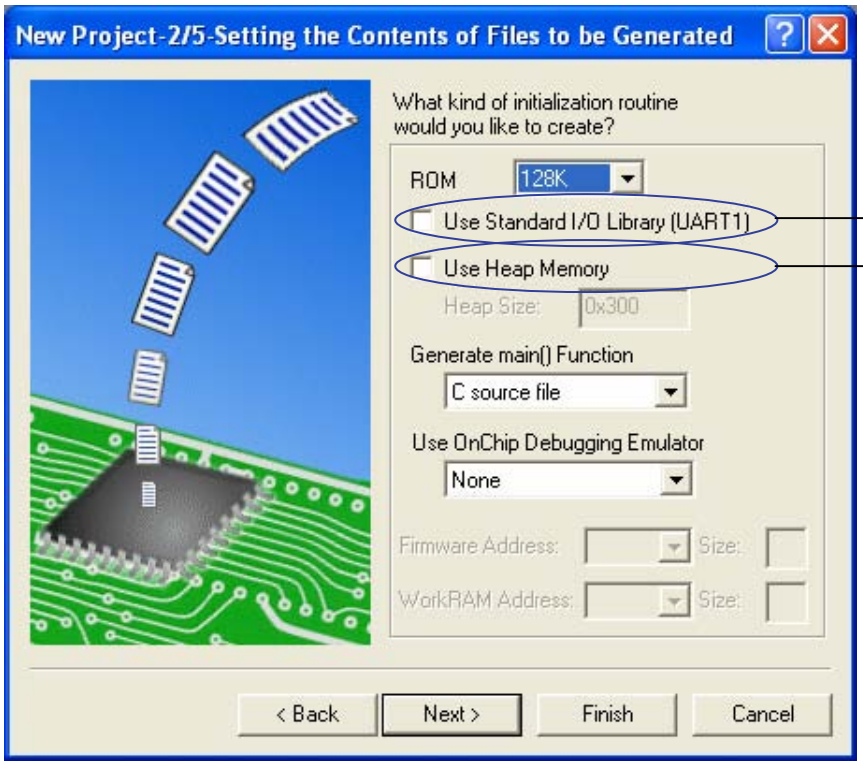
Although V.5.40 Release 00(A) showed ROM sizes in parentheses in CPU Group selection, note that beginning with this version, ROM size selection is moved to the wizard that is displayed when you click the Next> button.

- Selecting the size of ROM



The ROM size that you select in (3), in addition to settings in V.5.40 Release 00(A) where the on-chip debugger is selected, ensures that the ROM attribute sections are located appropriately when linked according to the ROM size.

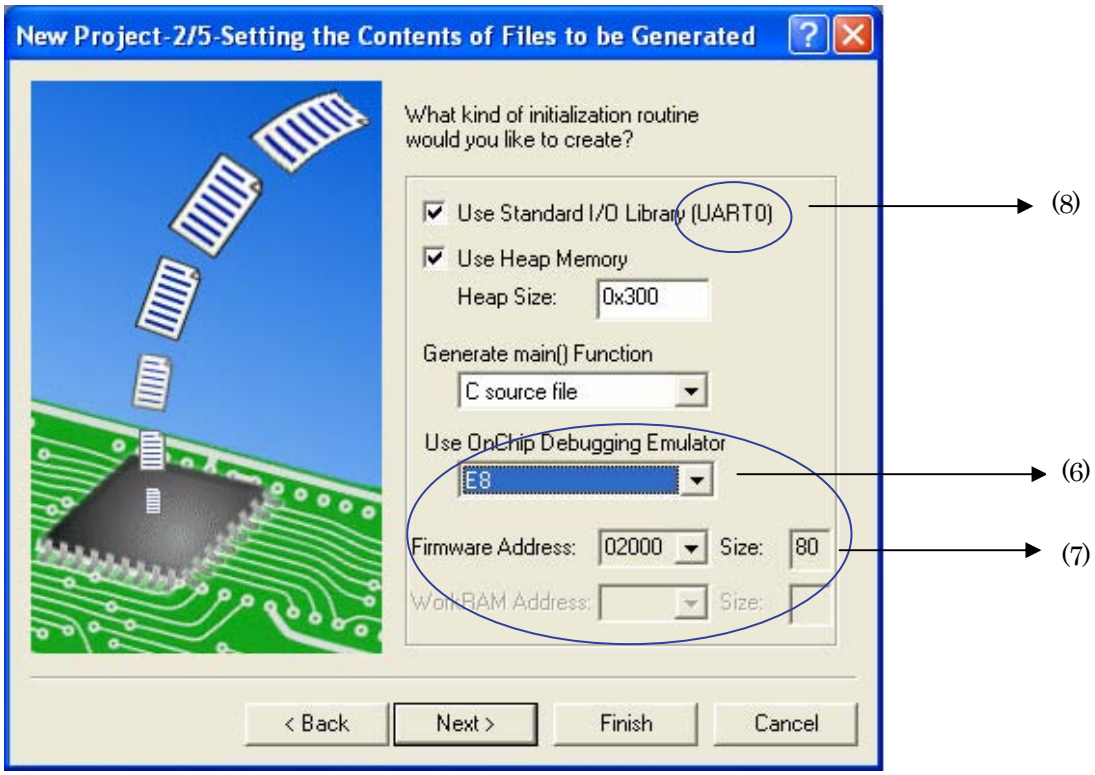
- Settings for the case where the standard function and memory management function libraries are used



(4) Select this check box when you use the standard function library.
When this check box is selected, function calls to _init() in resetprg.c are enabled.
Furthermore, device.c and init.c are registered to the project.

(5) Select this check box when you use the memory management function library.
When this check box is selected, function calls to heap_init() in resetprg.c are enabled.
Furthermore, heapdef.h and heap.c are registered to the project.

- Select OnChipDebugger



(6) Select the appropriate debugger when you use OnChipDebugger.

You can select either FoUSB or E8 as debugger.

However, there may be a case when you can't select one of or both of the debuggers depending on the target MCU.

When this selection is made, firm.c is registered and the area for the debugger displayed at (7) is saved as the variable area. Duplication of the user's program and the area for the debugger is avoided.

(7) Set Firmware Address and workRam Address.

Here, you set the program area for Firmware and the RAM area for work to be used by FoUSB/E8.

You can alter the default values only when the debugger allows you to do so.

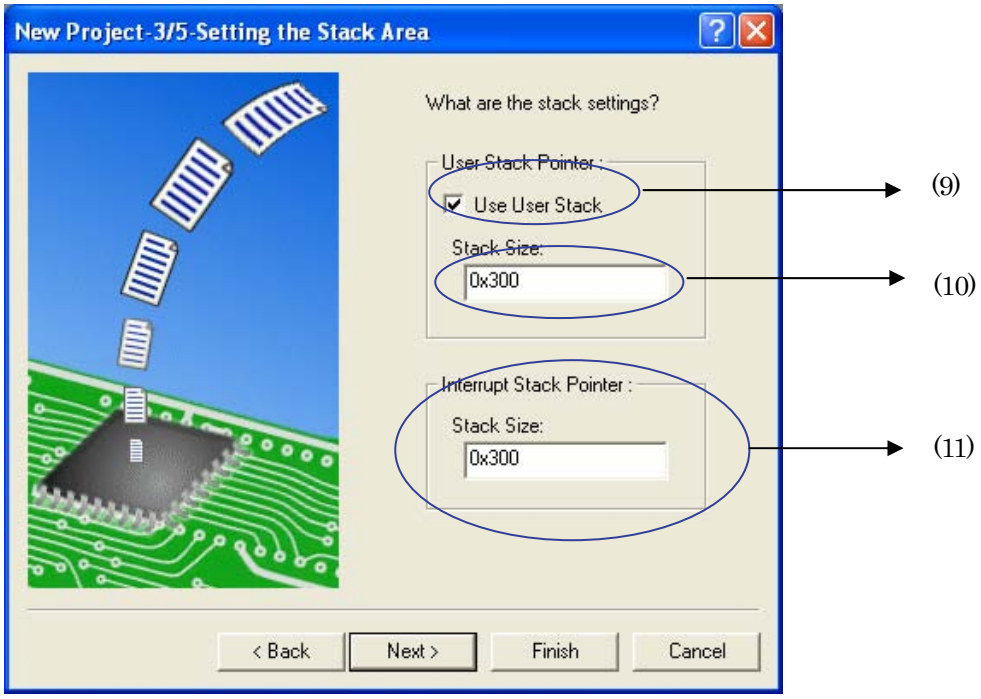
If you alter these addresses when using the debugger, alter them to suit the setup of the debugger.

For details about the address and size to be altered for each, consult the user's manual of your debugger.

(8) If you select OnChipDebugger while the standard input/output function library is selected, "UART1" indicated here changes to "UART0."

This means that the standard input/output device is changed to UART0 because the standard input/output functions and OnChipDebugger both use UART1.

- Selecting the stack size



(9) Choose to use or not use the user stack.

If this check box is unselected, settings are changed so that the user stack will not be used in the start function.

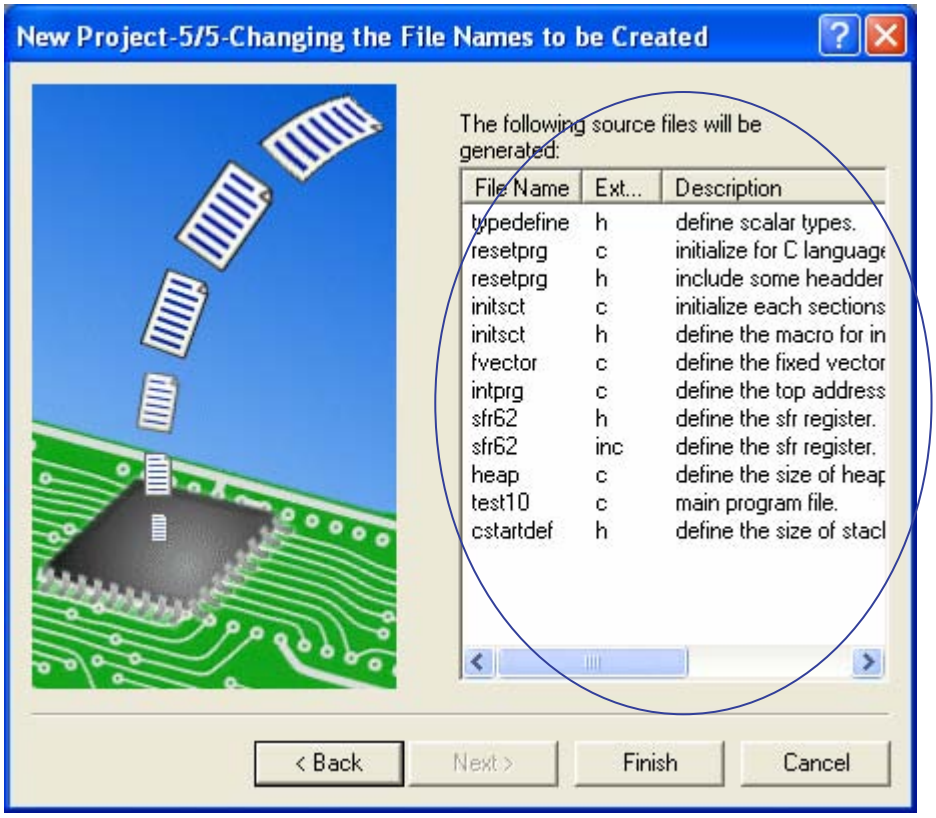
(10) Set the user stack size.
The define value in cstartdef.h is changed.

(11) Set the interrupt stack size.
The define value in cstartdef.h is changed.

To change the stack size or HEAP size after you created a project, change the respective values that are set in cstartdef.h as shown below.

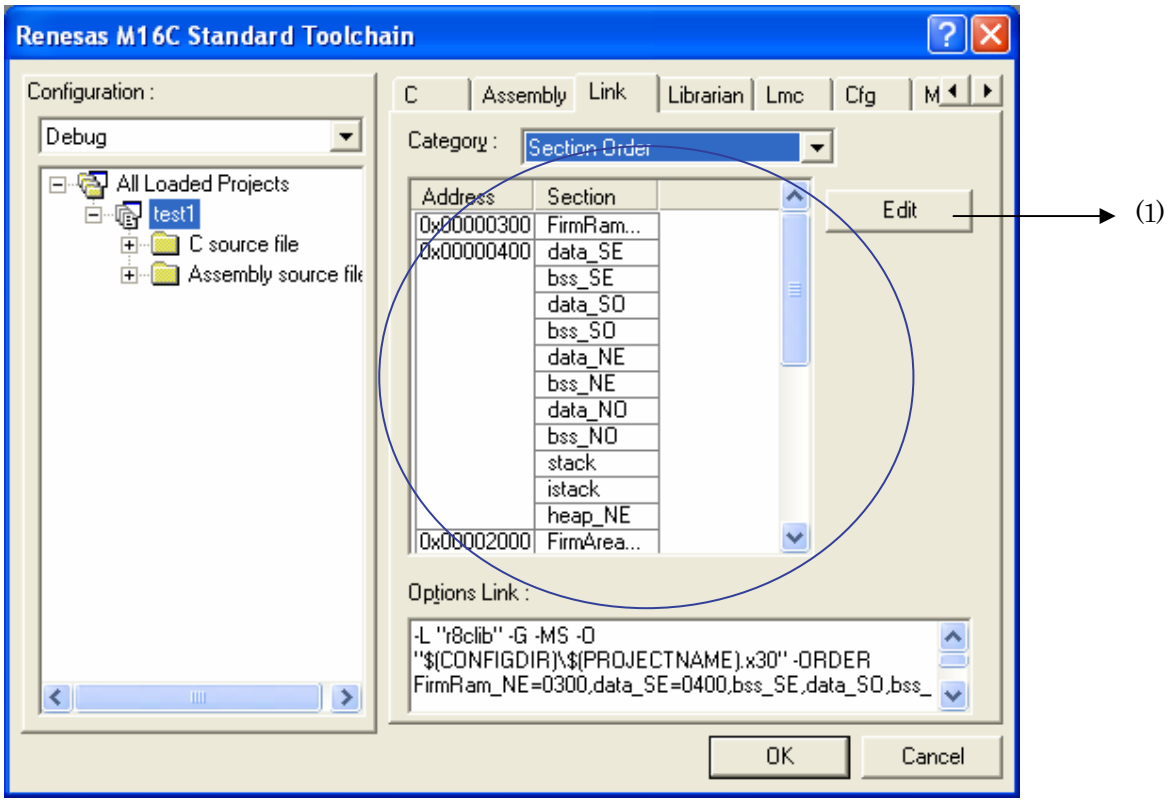
```
#define __STACKSIZE__          0x80
#define __ISTACKSIZE__         0x80
#define __HEAPSIZE__           0x80
```

- List of registered files

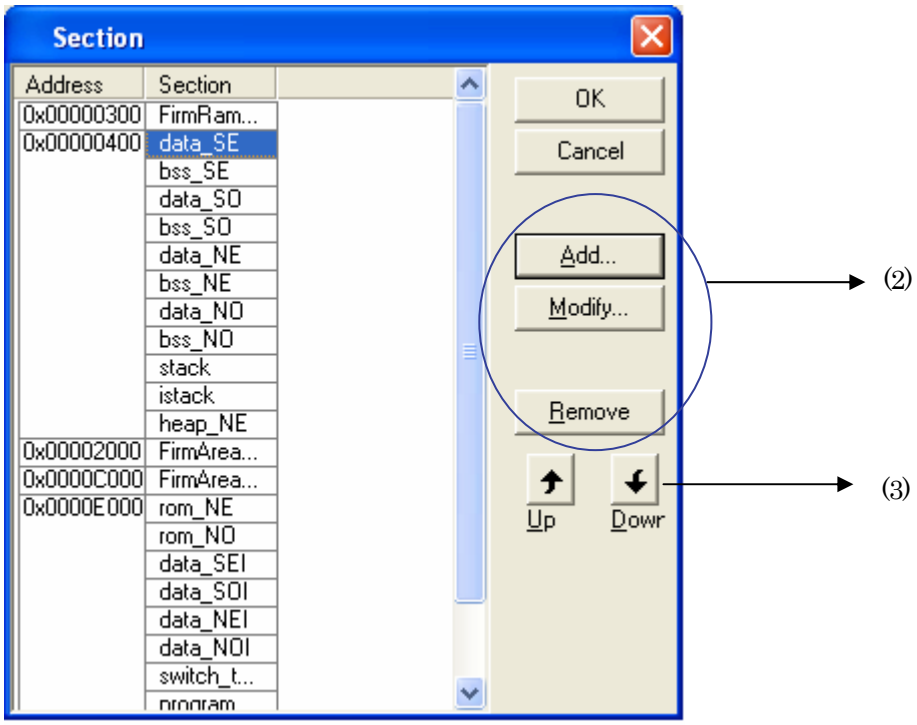


Here, you can check the list of files to be registered.

- Section order

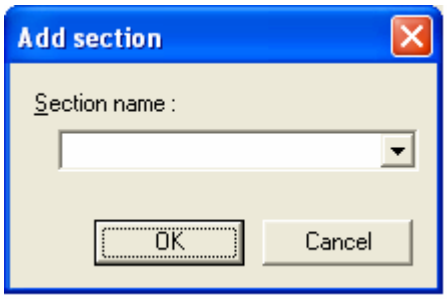


To confirm the order in which sections are linked and the addresses to which they are linked, take a look at Category: Section Order in [Renesas M16C Standard Toolchain] → [Link].



If you added a new section with #pragma SECTION, click the [Edit] button in (1) to open the Section window.

While the Sectiondialog has the focus, click the [Add] button in (2).



The Add section dialog will be launched, so enter the name of the new section that you want.
The section you've entered will be registered, so move the section to the area in which you want it to be located by using the Up or Down arrow in (3).