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/60,30,Tiny,20,10,R8C/Tiny series C compiler Package

M3T-NC30WA V.5.43 Release 00

Release note

(Rev.1.0)

Renesas Solutions Corporation

Mar.1, 2007

Abstract

Welcome to M3T-NC30WA V.5.43 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, refer to Appendix A, “C Compiler Package Installation Guidebook.

2. Precautions on Product

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

2.1. Precaution M16C Family-Dependent Code

2.1.1. M16C Precautions regarding the M16C interrupt control register

When -O5 optimizing options 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.
However, the compiler does not recognize the type of any register, so, should BTSTC or BTSTS instructions be generated for interrupt control registers, the assembled program will be different from the one you intend to develop. For detailed information about this, see Section 5, "Precautions for Interrupts" Described in Related Documents (Excerpts).
When using any of the products concerned, ensure that no incorrect code is generated.

Note:
In the products other than those concerned, this problem does not occur because BTSTC and BTSTS instructions are not generated if any of the above-mentioned optimizing options is used.

- Workaround
  This problem will be circumvented in either of the following ways:
  (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 any of the above-mentioned optimizing options.
  (2) Add an asm function to disable optimization locally, as shown in the example below.
- Example

```
while(TA0IC.IR==0)
{
        asm();      /* An asm function added to disable optimization for TAOIC.IR */
}
```

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

- 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.

2.1.2. 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.

2.1.3. About specifying the interrupt control register

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

- case of specifying 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 : Specifying the 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 : Specifying the 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(timer A,4);    // Changing the value "4" to the interrupt priority level
}
```

2.1.4. Regarding M16C/62 4M extended mode

Make sure the program is located in the internal ROM.

2.2. Precautions about NC30

2.2.1. About conditional operator

When the comma is used for the operational expression of the conditional operator, all the left sides of the constant

type are not executed when the end of the comma type is a type of the constant alone.
Example)
(func1(), 1+2) ? func2() : func3();
Workaround:
Please divide the expression without writing the comma at the left of the conditional operator.

2.2.2. 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 bits.
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.

2.2.3. About –Ostack_frame_align(-OSFA)option

OSFA adjust addresses of stacks of each function entry to an even-numbered address.
Therefore, if a function has no auto variable declaration, because enter #00H is always added, the processing speed may be slowed down.

2.2.4. On nesting inline functions

- Description
  When an inline function that takes a parameter is nested, it may refer to an incorrect argument (a variable, not an argument).
- Conditions
  This problem occurs if the following conditions are both satisfied:
  (1) An inline function is nested in another.
  (2) Inline function A as a calling source and inline function B as the destination take the same parameter.
- Example

```
inline    B(int aaa, char ccc)          /* Condition (2) */
{
          .....
}

inline    A(int c, int aaa, char *ccc)  /* Condition (2) */
{
          int       i;
          char      c;

          B(i,c);                       /* Condition (1) */
}
```

- Workaround
  This problem can be circumvented any of the following ways:
  (1) Change the name of the parameter taken by the destination function (inline function B in the above example).
  (2) Don't nest any inline function.
  (3) Compile the program using the -Oforward_function_to_inline(-OFFTI) option.

2.2.5. Precaution for startup

For C-language programs to be written into ROM,NC30 comes with a sample startup program written in the assembly language to initial set the hardware (M16C),locate sections, and set up interrupt vector address tables,etc. This startup program needs to be modified to suit the system in which it will be installed. See the manual or data book for your machine for the address of the processor mode register and the bit settings and for your machine for the address of the interrupt vector.

2.2.6. About the standard I/O function

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

2.2.7. Precautions about the search of an include file

If you give a file to include together 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 searched.

```
#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'
```

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

- Regarding debug information when using #pragma 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 functions (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 recover register contents on and from the stack when registers are loaded using inline assemble functions described above.

2.2.9. Precautions about debugging of a program using _Bool type

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

2.2.10. 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 prefix UL.

2.2.11. 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 );
}
```

2.2.12. 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 cannot be determined to be good or not 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
}
```

2.3. Precautions about the MS-Windows version

2.3.1. Precautions about environment of operation

- The MS-Windows version operates under Windows 98, Windows NT 4.0 or later. It does not work under Windows 95 and Windows NT 3.5x or earlier.
- If in Windows NT environment the command prompt size is set to other than "80 x 25," the command prompt size will change frequently as you start the compiler. Make sure the command prompt size is set to "80 x 25."

2.3.2. Suggestions Concerning File Names

The file names that can be specified are subject to the following restrictions:

- Directory and file names that contain kanji cannot 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.
- Keyboard shortcuts cannot be used.
- The "..." symbol cannot be used as a means of specifying two or more directories.
- A file name in length of 128 characters or more including path specification cannot be used.

2.3.3. Precautions about virus check programs

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

2.3.4. Precautions when upgrading

To upgrade M3T-NC30WA, uninstall the currently installed M3T-NC30WA first before you install the new version.

- Procedure for uninstalling M3T-NC30WA
  To uninstall M3T-NC30WA, launch Add/Remove Programs in Control Panel and then execute Uninstall.

3. Contents of upgrade

3.1. Content of NC30 upgrade

- Automatic generation of SB relative addressing implemented

In addition to the conventional method in which SB relative is assigned to frequently referenced external variables by launching UTL30 after linking, a new facility has been implemented that analyzes the number of times external variables are referenced in each function before, based on which SB relative is assigned.

This facility is executed only when a newly prepared option is specified.
Option name: -fSB_auto (abbreviated: -fSBA)

* This option cannot be specified simultaneously with the optimization options -OS and -OS_MAX(-OSM).

Precaution:
If this option is specified, SB relative declared with #pragma SBDATA is ignored.
If this option is specified, -fno_even is enabled at the same time.

```
int sym;

void func(void)
{
        int j;                          generate code that SB register is changed.

        j = sym;
}
```

External variables located in a 256-byte area beginning with the address that contains a frequently referenced external variable in each function are referenced using SB relative addressing.

Switchover Condition
Applied to the following storage classes:

- static variables
- Defined external variables

Following are the switchover conditions:

- SB address begins with the address of an external variable that is frequently referenced in a function, and the variables adjacent to it are covered by SB relative.
- All other variables are referenced by absolute addressing.

Switchover Processing
Primarily, following processing are performed.
(1) At the beginning of each function, SB registers are saved and addresses are set in SB registers.
(2) Information on SB symbols that are effective in each function is generated.
(3) At the exit of each function, SB registers are restored.

Added Directive Commands
The directive commands used in the complier's SB relative addressing automatic generation facility (compile option "-fSB_auto")—".SB_AUTO," ".SB_AUTO_S," ".SB_AUTO_SBVAL," ".SB_AUTO_SBSYM," ".SB_AUTO_R," and ".SB_AUTO_E"—have been added.

Precaution
The above directive commands cannot be written in user programs because they are the compiler's dedicated directives that it generates.

Note, however, that assembler code may be generated for the directive commands listed below.
.SB_AUTO_SBVAL: Instruction to save SB register (PUSHC) and the instruction to set SB register value (LDC)
.SB_AUTO_R: Instruction to restore SB register (POPC)

- #pragma section Improvement

Conventionally, for the section bss that has data without initial values located in it, section names could only be changed once per file by #pragma SECTION. Thanks to functional enhancement, section names can now be changed any number of times as for other sections (data section and program section).

```
#pragma SECTION bss bss1
int i;

#pragma SECTION bss bss2
int m;
```

Conventionally...
~~Variable 'i' is located in bss1 section.~~
~~#pragma SECTION bss bss1 has no effect, so that variable 'm' is located in bss2 section.~~

#pragma SECTION bss bss1 has no effect, so that
Variable 'i' and 'm' are located in bss2 section.

In V.5.43 Release 00 ...
Variable 'i' is located in bss1 section.
Variable 'j' is located in bss2 section by #pragma SECTION bss bss2.

3.2. Content of upgrade about Integrated Development Environment

The High-performance Embedded Workshop included with this C compiler package has been updated to V.4.02.00.

3.3. Other

- Support for Call Walker
  The C compiler now supports the utility tool named "Call Walker" that calculates the stack size used in an application program.
- Support for the Map Section Information Window of the High-performance Embedded Workshop
  The C compiler now supports the Map Section Information Window of the High-performance Embedded Workshop that shows the map information of absolute module files.

4. Installing M3T-NC30WA

4.1. Before installing M3T-NC30WA

Please confirm as follows before installing M3T-NC30WA in your computer.

- Please carefully read the "License Agreement" and "Release Note" included with your product before using M3T-NC30WA. If you've installed this product in your computer, it is assumed that you've agreed to the provisions stipulated in the License Agreement.
- In order that M3T-NC30WA operates comfortably, it requires at least 32Mbytes of memory and a hard disk having 20Mbytes or more of space.
- Use the dedicated installer to install M3T-NC30WA.
- You need to input a license ID in the middle of installation. Before you start installing M3T-NC30WA, check your license ID.

4.2. Precaution about installing this product

- When installing this product in a computer, the integrated development environment HEW (High-performance Embedded Workshop) is also installed together with a compiler.
- When the following dialog is outputted during installation, please choose "***it is all yes***."

  [Dialog message]
  The following file is already on your computer.
  C:¥WINDOWS¥TEMP¥....
  Do you wish to overwrite this file?
  :

4.3. M3T-NC30WA Installer

The installer is provided for each of the environments (supported host, supported OS and language) listed below. Check the product you've purchased to find the appropriate installer.

- Japanese environment

Supported host	Supported OS	Installer name	Directory on CD-ROM
IBM[1] PC/AT compatible	Microsoft Windows[2] 2000 Microsoft Windows XP	SETUP.EXE	¥NC30WA¥W95J

- English environment

Supported host	Supported OS	Installer name	Directory on CD-ROM
IBM PC/AT compatible	Microsoft Windows 2000 Microsoft Windows XP	SETUP.EXE	¥NC30WA¥W95E

4.4. Installing MS-Windows version

Please install the MS-Windows version in the following procedure.

(1) Go to the directory corresponding to your system, which can be found the name of the software you purchased, on the CD-ROM.
(2) Start up the installer and follow the messages displayed on the screen as you install M3T-NC30WA.

[1] IBM and AT are registered trademarks of International Business Machines Corporation.
[2] Microsoft, Windows, and Windows NT are registered trademarks of Microsoft Corporation in the U.S. and other countries.

4.5. Setting when compiler is used on DOS prompt and command prompt

The environment variable of the compiler is set to setnc30.bat that exists in the installation directory of the compiler.
Please execute setnc30.bat when you use the compiler on the DOS prompt and the command prompt.

5. Software version list of M3T-NC30WA V.5.43 Release 00

The following lists the software items their versions included with NC30WA V.5.43 Release 00.

nc30	V.6.01.20.000	Compile driver
cpp30	V.4.09.02.000	Preprocessor
ccom30	V.5.31.30.001	Compiler
aopt30	V.1.03.05.000	Assembler's optimizer
sbauto	V.1.00.00.000	SB register automatic changeover utility
utl30	V.1.00.09	SBDATA & Special page Utility
stk	V.1.00.05	stack utility
stkviewer	V.1.00.01(W95J&W95E)	STK Viewer
mapviewer	V.3.01.02(W95J&W95E)	MAP Viewer
as30	V.5.14.00.000	Assemble driver
mac30	V.3.42.00.000	Macro processor
pre30	V.1.10.12	Structure Assembler
asp30	V.5.13.00.000	Assemble Processor
ln30	V.5.12.02.000	Linkage Editor
lb30	V.1.02.00	Librarian
lmc30	V.4.01.01.000	Loadmodule Converter
xrf30	V.2.02.00.000	Cross Referencer
abs30	V.2.11.00	Absolute listor
genmap	V.1.00.00.000	for EcxMap
gensni	V.1.00.00.002	for CallWalker

6. Installing C Compiler Package

6.1. Before installing C Compiler Package

Please confirm as follows before installing C Compiler Package in your computer.

- Please carefully read the "License Agreement" and "Release Note" included with your product before using C Compiler Package. If you've installed this product in your computer, it is assumed that you've agreed to the provisions stipulated in the License Agreement.
- In order that C Compiler Package operates comfortably, it requires at least 32Mbytes of memory and a hard disk having 20Mbytes or more of space.
- You need to input a license ID in the middle of installation. Before you start installing C Compiler Package, check your license ID.

6.2. Installing C Compiler Package

Please install C Compiler Package in the following procedure.

(1) Go to the directory corresponding to your system, which can be found the name of the software you purchased, on the CD-ROM.
(2) Start up the installer and follow the messages displayed on the screen as you install C Compiler Package.

6.3. Setting when compiler is used on DOS prompt and command prompt

The environment variable of the compiler is set to setnc30.bat that exists in the installation directory of the compiler.
Please execute setnc30.bat when you use the compiler on the DOS prompt and the command prompt.

●

7. Conformance with MISRAC Rule in Standard Function Library

7.1. Standard Function Library

In C-Source code of standard function library M3T-NC30WA, it is found that 52 rules[3] are against the MISRAC Rule NOTE, but these violations do not constitute a drawback to any operation.

7.1.1. 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

7.1.2. 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								

7.1.3. 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

7.2. SFR header files (used in C startup files)

In C-Source code of sfr header files outputted by High-performance Embedded Workshop, it is found that 6 rules[4] are against the MISRAC Rule NOTE, but these violations do not constitute a drawback to any operation.

7.2.1. Cause of Rule Violation

In C-Source code of sfr header files outputted by High-performance Embedded Workshop, the major causes for rule violation are as follows:

- C-Compiler specifications (near/far modifier, asm () function and #pragma)
- Declaration of typedef
- Declaration of member of bitfield

7.2.2. Inspection No. running counter to the rule

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

13	14	22	99	110	111

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

7.2.3. Evaluation Environment

Compiler	M3T-NC30WA V.5.42 Release 1
Compile Option	-O -c -as30 "-DOPTI=0" -gnone -finfo -fNII -misra_all -r $*.csv
MISRA C Checker	SQMlint V.1.03 Release 00

7.3. C startup files

In C-Source code of C startup files outputted by High-performance Embedded Workshop, it is found that 8 rules[5] are against the MISRAC Rule NOTE, but these violations do not constitute a drawback to any operation.

7.3.1. Cause of Rule Violation

In C-Source code of C startup files outputted by High-performance Embedded Workshop, 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

7.3.2. Inspection No. running counter to the rule

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

18 22 54 59 71 99 115 126

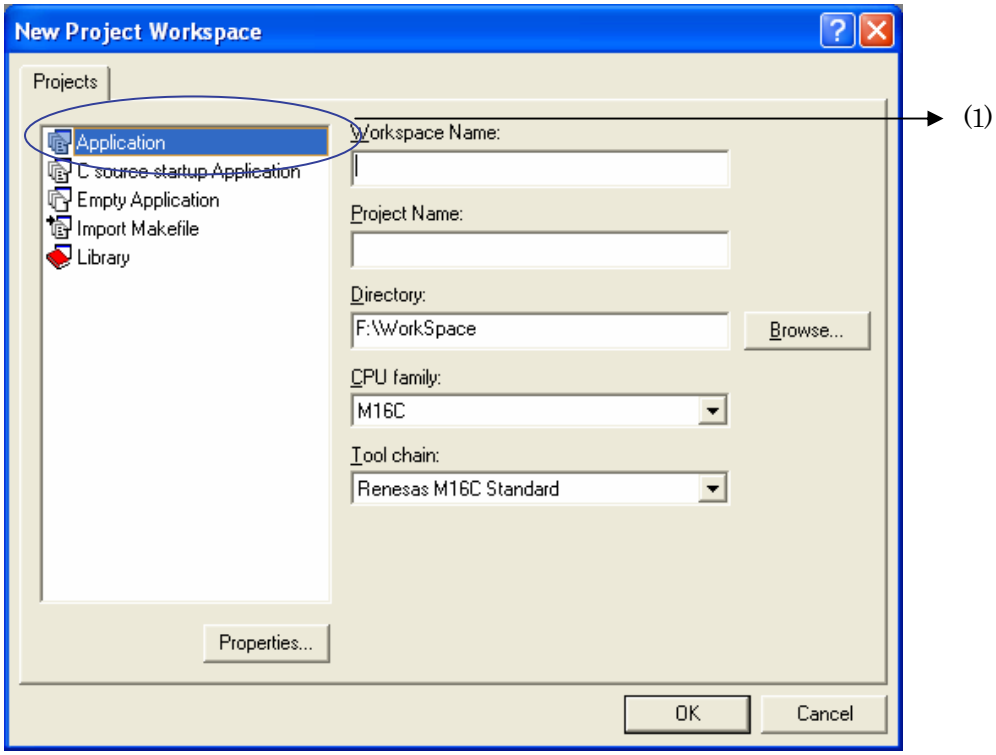
7.3.3. Evaluation Environment

Compiler	M3T-NC30WA V.5.42 Release 1
Compile Option	-O -c -as30 "-DOPTI=0" -gnone -finfo -fNII -misra_all -r $*.csv
MISRA C Checker	SQMlint V.1.03 Release 00

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

C-language Startups

Startup programs written in C language are supported beginning with the latest version described here.
This C language startup can only be used in combination with V.5.43 Release 00. It cannot be compiled in an earlier version of the compiler preceding V.5.40 Release 00(A).
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 above.



7.4. 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.
(7) cstartdef.h
set defined values. (Please do not alter the file.)
(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) vector.h
Defines the variable vector address.
(11) typedef.h
Declares each type by typedef.

7.5. Processing of Each Generated File

7.5.1. resetprg.c (essential)

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

```
agma 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();              // initlalize 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) Declares the function body of the CPU initialization function start().
(3) Initializes the interrupt stack pointer.
(4) Write enables the protect mode register.
(5) Sets the processor mode register to "single-chip mode."
If modes need to be changed, this expression must be altered.
(6) Write protects the protect mode register.
(7) Sets the U flag.
If you chose to "Use the user stack" in the workspace creation wizard, the user stack pointer is set
(8) Initializes the user stack pointer if you chose to "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) Calls the CPU initialization function.
(12) Initializes each section (by clearing them to 0 and transferring initial values).
(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.

7.5.2. 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");
#if 0
        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");
#if 0
        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 init.sct.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.

(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.

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

```
#pragma SECTIONbss          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__.

7.5.4. 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).
* This pragma is used exclusively for startup and cannot normally be used.
(3) Defines the entry function.
The function to be executed upon reset is registered in a fixed vector.

7.5.5. intprg.c

(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.

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

The content of this file is altered depending on the microcomputer type and selected FoUSB/E8.

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

#pragma section bss FirmArea                                       ----------(2)
_far _UBYTE _firmarea[0x800];          // dummy for monitor        ----------(3)


#else       // for FoUSB

#pragma section bss FirmArea                                       ----------(4)
_far   _UBYTE _firmarea[0x600];        // dummy for monitor        ----------(5)
#endif
```

(1) Enables E8 when it is to be used
(2) Locates the firmware program of E8 in the FirmArea section.
(3) Specifies the size of the firmware program.
(4) Locates the firmware program of FoUSB in the FirmArea section.
(5) Specifies the size of the firmware program.

```
#ifdef __E8__                                   ----------(6)

#pragma section bss FirmRam                     ----------(7)
_UBYTE _workram[0x100];                         ----------(8)

#endif
```

(6) Enables E8 when it is to be used
(7) Allocates the work ram area to be used by the E8 firmware in the FirmRam_NE section.
(8) Reserves 0x100 bytes of storage for the work ram area. (It depends on the corresponding microcomputer type)

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

7.5.8. initsct.h (essential)

Please do not alter the content of this file.

7.5.9. resetprg.h (essential)

Please do not alter the content of this file.

7.5.10. resetprg.h (essential)

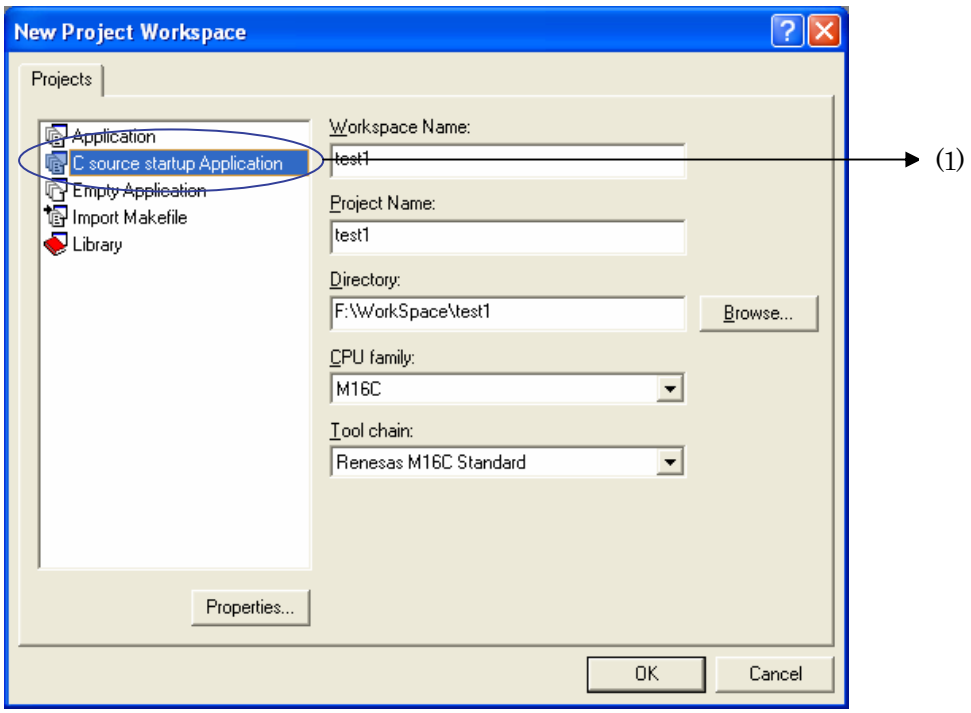
Please do not alter the content of this file.

7.5.11. typedef.h (essential)

Please do not alter the content of this file.

7.6. How to Generate a C-language Startup

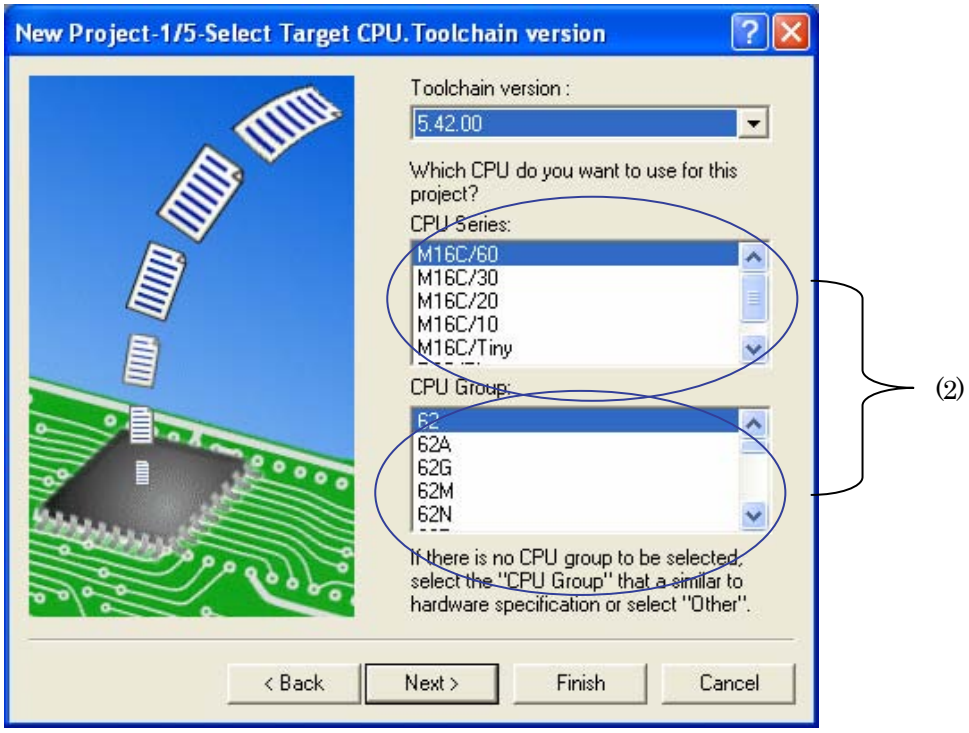
7.6.1. Selecting the project that uses a C-language startup.



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

* If while multiple compilers are installed in your computer you select another microcomputer for the CPU type after selecting C source startup Application, the focus for C source startup Application will move to Application, with the result that the selected C source startup has no effect. In such a case, therefore, select C source startup Application again.

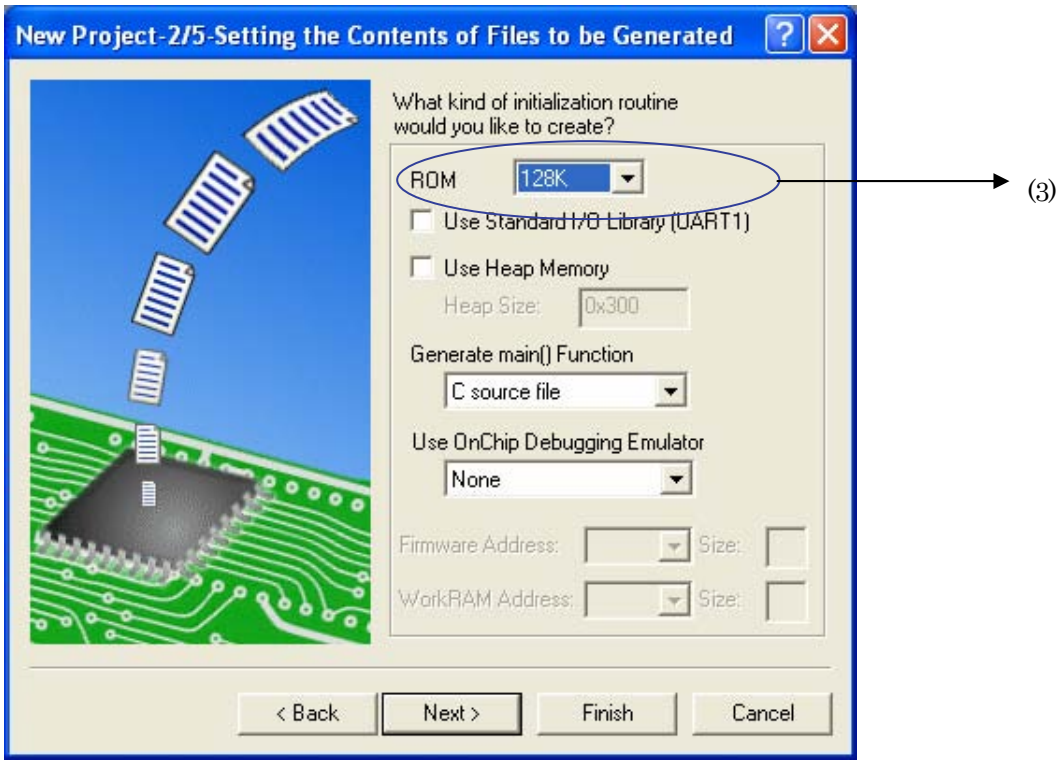
7.6.2. Selecting the type of microcomputer



(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.

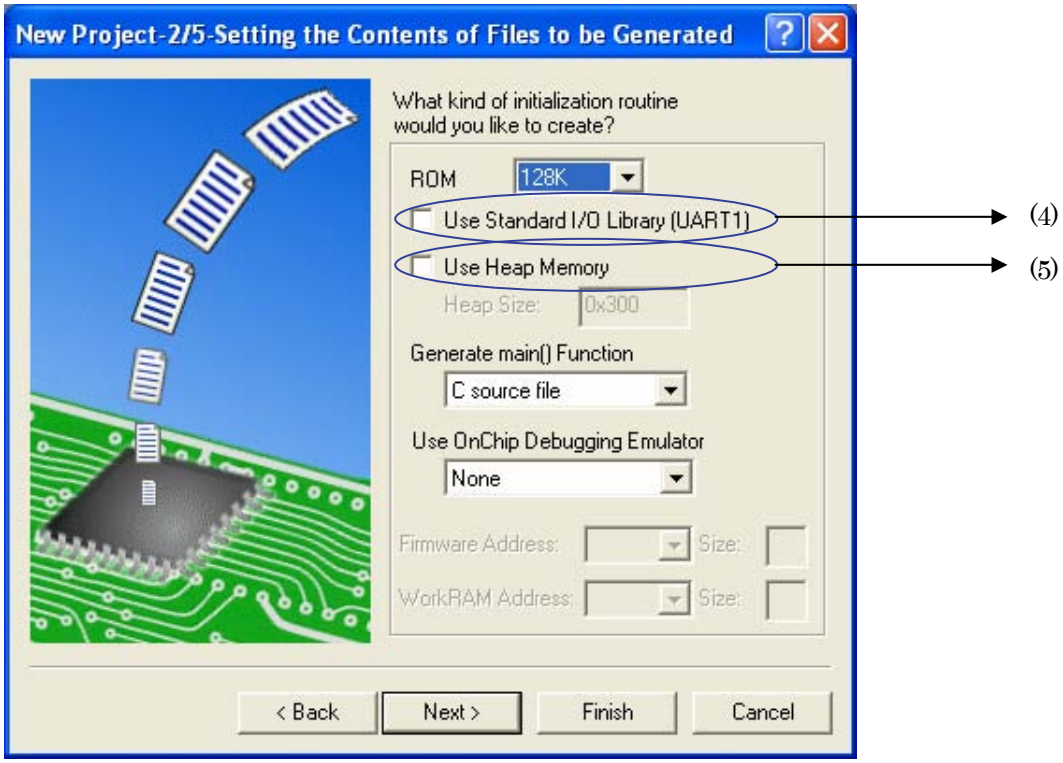
With this selection, the corresponding sfr header file is registered in the workspace.
Furthermore, the variable vector entry function (intprg.c) is registered.
Although V.5.40 Release 00(A) showed ROM sizes in parentheses in CPU Group selection, note that beginning with this version, ROM size selection is moved to the wizard that is displayed when you click the Next> button.

7.6.3. 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.

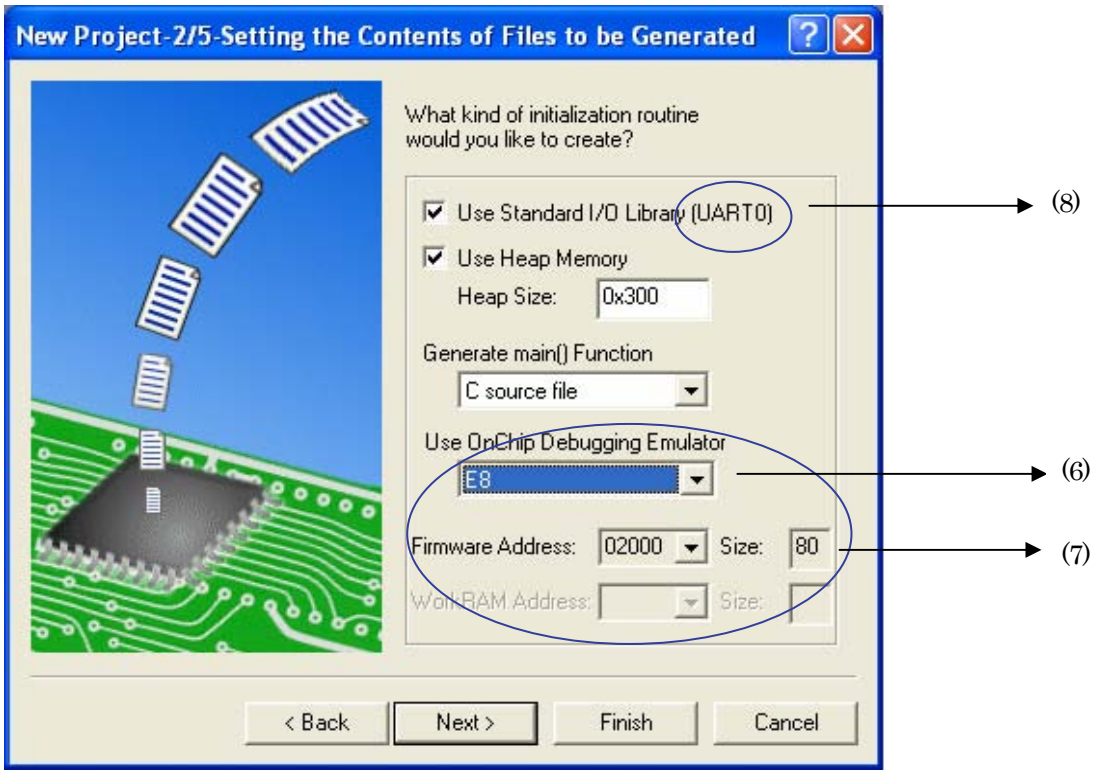
7.6.4. Settings for the case where the standard function and memory management function libraries are used



(4) Select this check box when you use the standard function library.
When this check box is selected (flagged with a check mark), 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 (flagged with a check mark), function calls to heap_init() in resetprg.c are enabled.
Furthermore, heapdef.h and heap.c are registered to the project.

7.6.5. Select OnChipDebugger

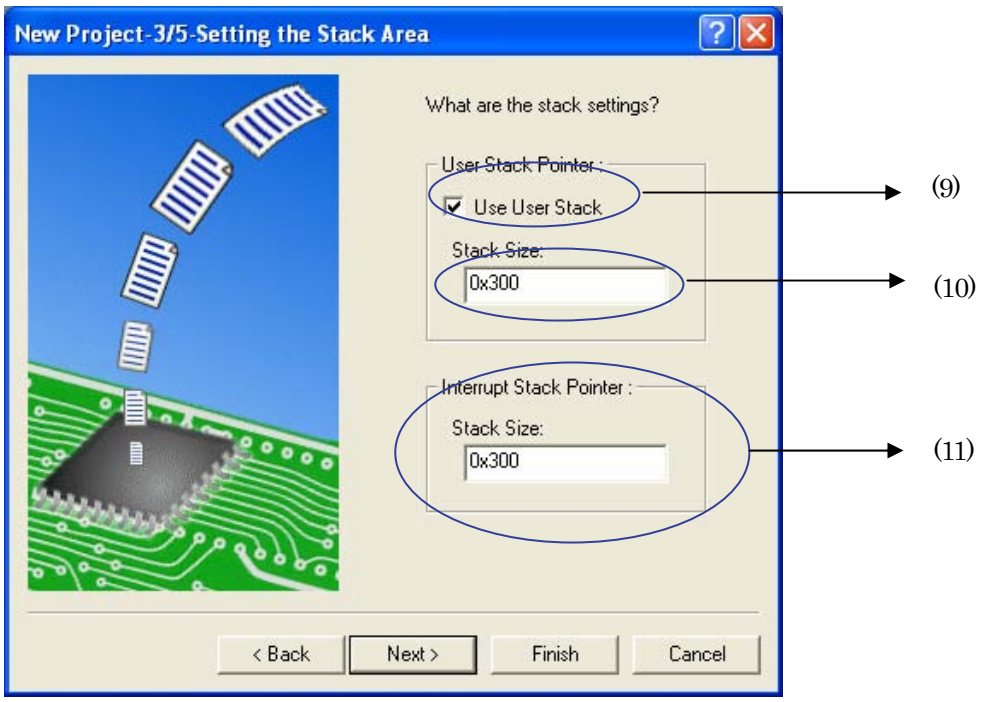


(6) Select the appropriate debugger when you use OnChipDebugger.
The selectable debuggers are FoUSB and E8.
Note, however, that you cannot select either one of the two or both depending on the selected type of microcomputer.
When this selection is made, firm.c is registered.

(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. However, if either input is 'grayed out,' you cannot alter it.
You can alter the set value only when input is enabled.
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.

7.6.6. Selecting the stack size

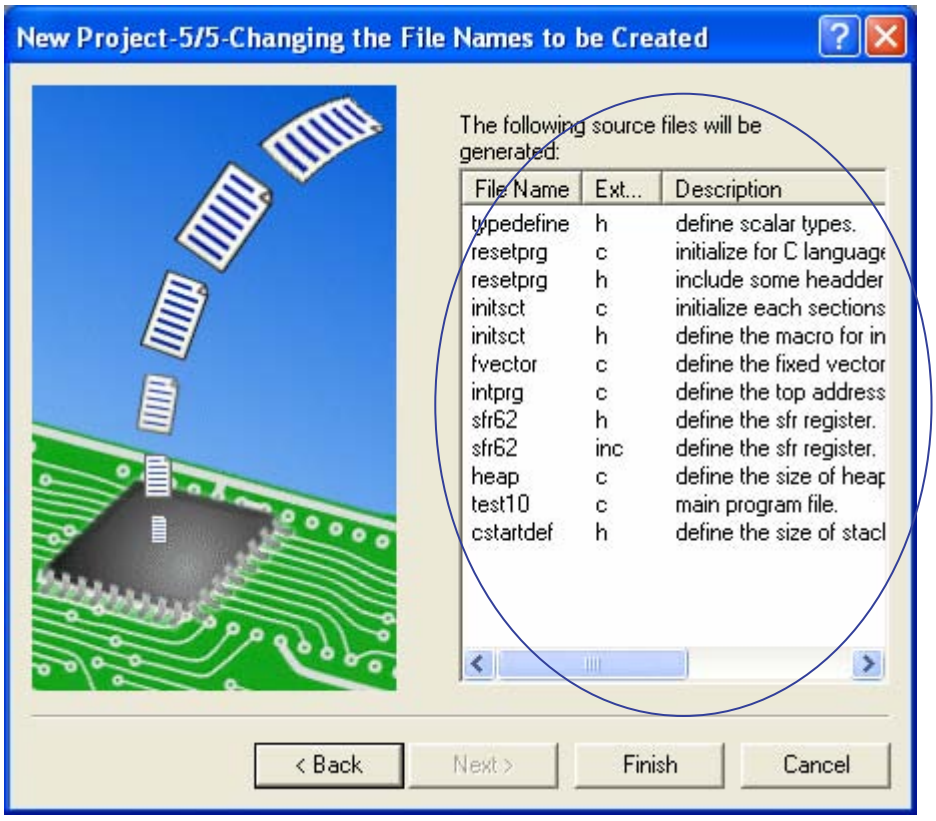


(9) Choose to use or not use the user stack.
If this check box is unselected, settings are changed not to use the user stack in the start function.
(10) Set the user stack size.
The define value in cstartdef.h is changed.
(12) Set the interrupt stack size.
The define value in cstartdef.h is changed.

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.

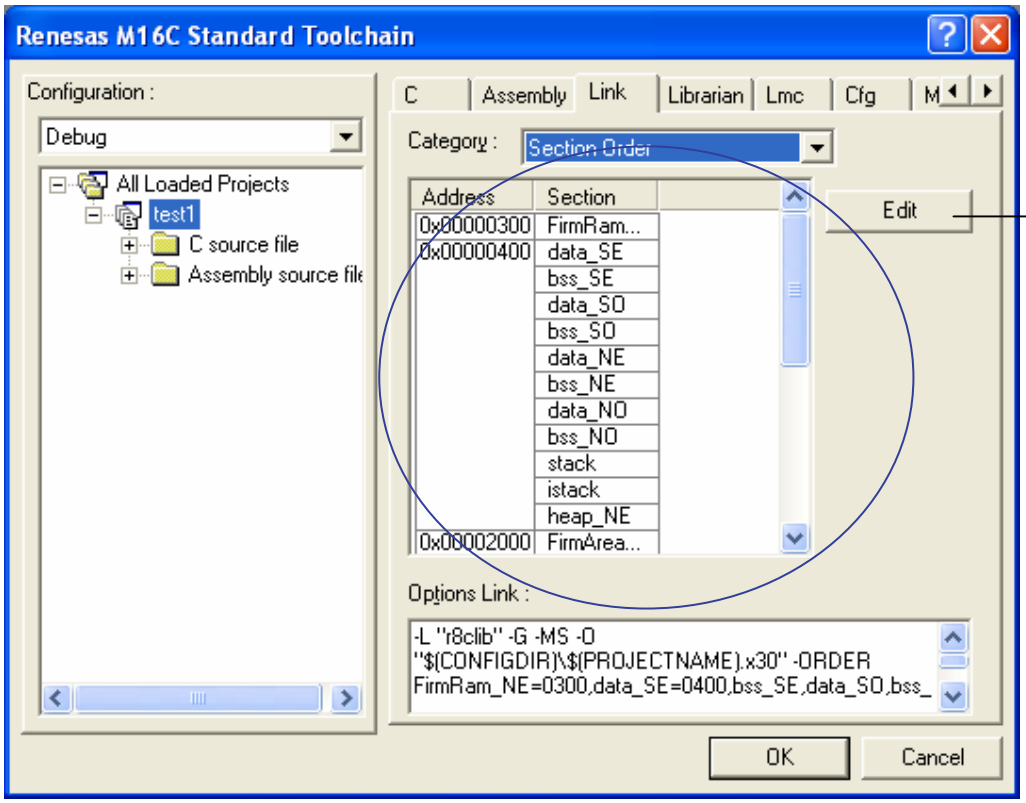
```
#define __STACKSIZE__          0x80
#define __ISTACKSIZE__         0x80
#define __HEAPSIZE__           0x80
```

7.6.7. List of registered files

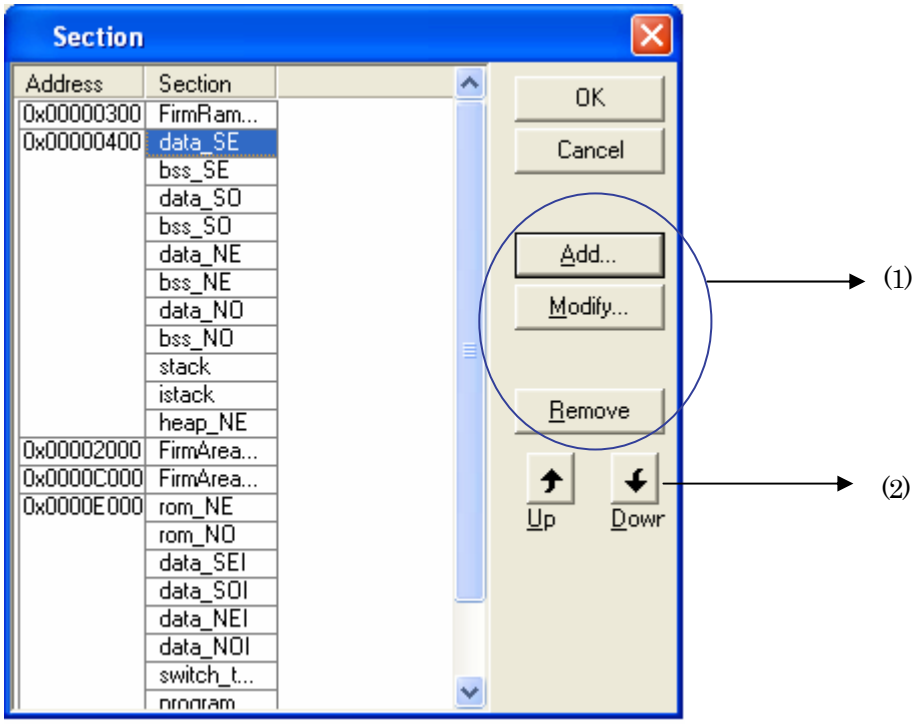


Here, you can check the list of files to be registered.

7.6.8. section order

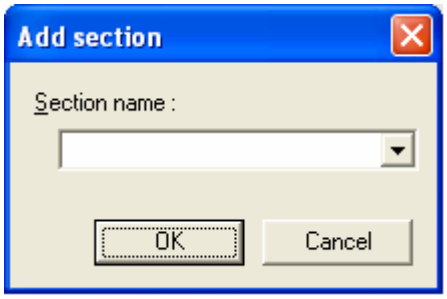


To confirm the order in which sections are linked and the addresses to which they are linked, take a look at Category: Section Order in [Renesas M16C Standard Toolchain] → [Link].



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

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



The Add New window will be displayed, 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).