

==== 必ずお読みください ====

M16C シリーズ,R8C ファミリ用 C コンパイラパッケージ
V.5.45 Release 01
 リリースノート
 (第 4 版)

株式会社ルネサス ソリューションズ

2010 年 12 月 15 日

概要

本資料は M16C シリーズ,R8C ファミリ用 C コンパイラパッケージ (M3T-NC30WA) V.5.45 Release 01 のご使用にあたり、C コンパイラパッケージの電子マニュアルの補足等について説明します。電子マニュアルの該当項目をご覧になる場合は、併せてこのリリースノートをご覧くださいませよう願ひ申し上げます。

| | |
|---|----|
| 1. C コンパイラパッケージのインストール..... | 3 |
| 2. 最新情報のご案内..... | 3 |
| 3. C コンパイラユーザズマニュアル更新のご案内..... | 3 |
| 4. 注意事項..... | 3 |
| 4.1. TM に関する注意事項..... | 3 |
| 4.2. リアルタイム OS MR30 の対応バージョンについて..... | 3 |
| 4.3. ファイル名に関する注意事項..... | 3 |
| 4.4. ウィルスチェックプログラムに関する注意事項..... | 4 |
| 4.5. 機種依存部に関する注意事項..... | 4 |
| 4.5.1. 割り込み制御レジスタに関する注意事項..... | 4 |
| 4.5.2. SFR 領域のアクセスに関する注意事項..... | 5 |
| 4.5.3. 割り込み優先レベルに値を設定する場合の注意事項..... | 5 |
| 4.5.4. M16C/62 4M 拡張モードに関して..... | 5 |
| 4.5.5. オンチップデバッグ選択時の FirmRam_NE セクションと SB レジスタの値に関して..... | 6 |
| 4.5.6. M16C シリーズで、FFF8h 番地～FFFDBh 番地に予約領域のある品種について..... | 6 |
| 4.6. コンパイラ、アセンブラ、リンケージエディタ及びユーティリティに関する注意事項..... | 6 |
| 4.6.1. -OGJ について..... | 6 |
| 4.6.2. -OLU, -OFFTI, インライン関数を併用する場合について..... | 7 |
| 4.6.3. -ffar_pointer(-ffp) について..... | 7 |
| 4.6.4. アセンブラ記述スタートアップファイル(ncrt0.a30,sect30.inc,nc_define.inc)の扱いについて..... | 7 |
| 4.6.5. 標準入出力関数について..... | 7 |
| 4.6.6. インクルードファイルの検索に関する注意事項..... | 7 |
| 4.6.7. インラインアセンブル機能(#pragma ASM～#pragma ENDASM、asm 関数)に関する注意事項..... | 8 |
| 4.6.8. _Bool 型を使用しているプログラムのデバッグに関する注意事項..... | 8 |
| 4.6.9. 前処理命令#define に関する注意事項..... | 8 |
| 4.6.10. マクロ定義に関する注意事項..... | 8 |
| 4.6.11. #if 指令に関する注意事項..... | 9 |
| 4.6.12. 構造体を返す関数をコールすると、System Error となる..... | 9 |
| 4.6.13. コンパイルオプション-Ostack_frame_align(-OSFA)を使用する場合の注意事項..... | 10 |
| 4.6.14. 右シフト演算に関する注意事項..... | 11 |
| 4.6.15. utl30 の注意事項..... | 11 |
| 4.6.16. MapViewer の注意事項..... | 11 |

| | | |
|---------|--|----|
| 4.6.17. | メモリ管理関数malloc()、calloc()およびrealloc()の注意事項 | 11 |
| 4.6.18. | 不完全型構造体または共用体の型定義に関する注意事項 | 12 |
| 5. | V.5.45 Release 00 からのバージョンアップ内容 | 12 |
| 5.1. | Cコンパイラ | 12 |
| 5.1.1. | Cコンパイラ不具合改修 | 12 |
| 5.2. | High-performance Embedded Workshop | 12 |
| 5.2.1. | High-performance Embedded Workshop不具合改修 | 12 |
| 6. | ソフトウェアのバージョン一覧 | 13 |
| 7. | MISRA Cルール適合に関して | 14 |
| 7.1. | 標準関数ライブラリ | 14 |
| 7.1.1. | ルール違反の要因 | 14 |
| 7.1.2. | ルール違反となった検査番号 | 14 |
| 7.1.3. | 評価環境 | 14 |
| 7.2. | HEW (High-performance Embedded Workshop) が自動生成するソースコード | 14 |
| 7.2.1. | ルール違反の要因 | 14 |
| 7.2.2. | ルール違反となった検査番号 | 15 |
| 7.2.3. | 評価環境 | 15 |
| 7.2.4. | Cスタートアップ中で使用する#pragma拡張機能 (Misra Cルール 99) | 15 |
| 8. | C言語スタートアップについて | 16 |
| 8.1. | 生成ファイル | 16 |
| 8.2. | 各生成ファイルの処理 | 17 |
| 8.3. | C言語スタートアップの生成方法 | 23 |
| 9. | マニュアルの誤記 | 29 |

1. Cコンパイラパッケージのインストール

インストールについては、[インストールガイド](#)をご覧ください。

2. 最新情報のご案内

本製品の最新情報については以下を参照してくださるようお願いします。

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

3. Cコンパイラユーザーズマニュアル更新のご案内

インストールされるユーザーズマニュアルから誤記等を修正しています。

ウェブから最新のユーザーズマニュアルをダウンロードして下さい。

http://documentation.renesas.com/jpn/products/tool/rjj10j2478_nc30_u.pdf

4. 注意事項

本製品をご使用いただく際に以下の注意事項があります。

4.1. TMに関する注意事項

V.5.45 Release 01 では、統合化開発環境 TM には対応しておりません。

そのため下記の対応ができません。

- V.5.45 Release 01 での新規プロジェクトの作成
- TM 対応時に作成したプロジェクトの移行

TM で作成したプロジェクトを High-performance Embedded Workshop 環境へ移行する方法については、[C コンパイラパッケージガイドブック](#)をご参照ください。

4.2. リアルタイムOS MR30 の対応バージョンについて

M3T-MR30 V.3.30 Release 1 以降のバージョンと組み合わせてご使用下さい。

注意) M3T-MR30 をインストールする場合は、必ずコンパイラと同じディレクトリ (bin,lib30,inc30) にインストールしてください。

4.3. ファイル名に関する注意事項

ソースプログラムファイルの名前や作業を行うディレクトリ名、ワークスペース名は、次の注意事項に従ってください。

- ASCII 以外の文字を含むディレクトリ名、ワークスペース名及びファイル名は使用できません。
- ファイル名に使用するピリオド (.) は一つのみ使用可能です。
- ネットワークパス名は使用できません。ドライブ名に割り当ててご使用ください。
- 「ショートカット」は使用できません。
- "..."表記を用いて 2 つ以上のディレクトリを指定することはできません。

※ワークスペースとは、統合化開発環境 High-performance Embedded Workshop 上でコンパイル/ビルド/

デバッグ等を行う作業ディレクトリです。

なお、上記を使用した場合、以下のような現象が発生する場合があります。

- アセンブラ指示命令 `.id`、`.ofsreg`、`.protect`、`.rvector` および `.svector` で 設定した値が正しく動作しない。
そのため、ID コードやオプション機能選択 レジスタの値を正しく設定できない等の問題が発生します。
- スタックサイズ使用量を参照する Call Walker が 正しく表示されない。
- アプソリュートモジュールファイルのマップ情報を参照する MAPViewer が正しく表示されない。
- 上記アセンブラ指示命令で設定した内容が、`.map` ファイルに表示されない。
- "Can't open file"などのコンパイルエラーが発生する。
- 「問題が発生したため、`lnxx.exe` を終了します」のようなメッセージを 出力してリンクが異常終了する。
- 可変ベクタテーブルの自動生成機能が正しく動作しない。

4.4. ウィルスチェックプログラムに関する注意事項

ウィルスチェックプログラムが常駐した状態で M3T-NC30WA を起動すると正常に起動しない場合があります。その場合は、ウィルスチェックプログラムの常駐を解除してから M3T-NC30WA を起動しなおしてください。

4.5. 機種依存部に関する注意事項

4.5.1. 割り込み制御レジスタに関する注意事項

最適化オプション "-O5" を指定するとビット操作命令 (BTSTS,BTSTC) を生成する可能性があります。BTSTS、BTSTC 命令は、M16C の割り込み制御レジスタを書きかえる命令として使用できません。本オプションを指定する場合は、必ず生成されたコードに問題が無いことをご確認ください。

- 発生例

以下のプログラムで最適化オプション "-O5" を指定した場合、最適化により BTSTC 命令を生成します。このため、割り込み要求ビットの判定が正常に行われず意図しない動作を行います。

```
#pragma ADDRESS TA0IC 55H
struct {
    char ILVL:3;
    char IR :1;          /*割り込み要求ビット */
    char dmy :4;
}TA0IC;
void wait_until_IR_is_ON(void)
{
    while(TA0IC.IR ==0){          /*1 になるまで待つ */
        :
    }
    TA0IC.IR =0;          /*1 になったら 0 に戻す */
}
```

- 対策

- (1) 該当する最適化オプションに加えてオプション "-Ono_asmopt[-ONA]" を指定することにより BTSTC、BTSTS 命令を生成する最適化を抑制してください。
- (2) 以下のように "asm 関数 "を挿入することにより最適化を抑制してください。

```
while(TA0IC.IR ==0){
    asm();          /* asm 関数を挿入。TA0IC に対して処理を抑制します。
*/
}
```

- 注意

オプション "-Ono_asmopt[-ONA]" または `asm` 関数の使用による対策後は、`BTSTC`、`BTSTS` 命令が生成されていないことを必ずご確認ください。

4.5.2. SFR領域のアクセスに関する注意事項

SFR 領域のレジスタをアクセスする場合には、特定の命令を使用しなければならないことがあります。この特定の命令は機種毎に異なりますので詳しくは各機種のユーザーズマニュアルなどを参照してください。本注意事項にかかわる命令は、`asm` 関数等のインラインアセンブル機能を使用してプログラム中に命令を直接記述してください。

4.5.3. 割り込み優先レベルに値を設定する場合の注意事項

テクニカルニュース(No.M16C-14-9804)「M16C/60、M16C/61、M16C/62、M16C/63 グループ割り込み制御レジスタの注意事項」に対応するため割り込み優先レベルのセット及び変更を行う関数をサポートしています。使用方法は、以下の通りです。

- セットする場合

`SetLevel` 関数をご使用ください。この時、`intlevel.h` ファイルを必ずインクルードしてください。

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

`adr` : 割り込み制御レジスタのアドレス

`val` : セットする値

- 変更の場合

`ChgLevel` 関数をご使用ください。この時、`intlevel.h` ファイルを必ずインクルードしてください。

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

`adr` : 割り込み制御レジスタのアドレス

`val` : セットする値

【例】

```
#include <intlevel.h>
#pragma ADDRESS timerA 55H
char *timerA;
void func(void)
{
    SetLevel(timerA,2); // 割り込み優先レベルを 2 に設定
    :
    ChgLevel(timer A,4); // 割り込み優先レベルを 4 に変更
}
```

4.5.4. M16C/62 4M拡張モードに関して

プログラムは、内部 ROM に配置するようにしてください。

4.5.5. オンチップデバッグ選択時のFirmRam_NEセクションとSBレジスタの値に関して

新規プロジェクトワークスペース作成時に OnChipDebugger 選択画面でデバッガを選択した場合、FirmRam_NE セクションが 400H から配置されることがあります。SB レジスタの初期値は 400H で設定されているため、SB 相対アドレッシングモードで正しい領域をアクセスできなくなります。

リンクした結果、FirmRam_NE セクションが 400H から配置されている場合は、SB レジスタの初期値を bss_SE セクションの先頭アドレスの値に変更してください。bss_SE セクションの先頭アドレスは、マップファイルで確認してください。

次の 2 箇所の値を bss_SE セクションの先頭アドレスにしてください。

<resetprg.c>

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

<resetprg.h>

```
#define DEF_SBREGISTER      _asm( " .glob SB %n"
                             " __SB__ .equ 0400H" )
```

該当 MCU は以下の通りです。(2009年5月16日現在)

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

4.5.6. M16Cシリーズで、FFFD8h番地～FFFDh番地に予約領域のある品種について

スペシャルページベクタテーブルの機能(#pragma SPECIAL、アセンブラ指示命令.svector)を使用できません。

4.6. コンパイラ、アセンブラ、リンカージェディタ及びユーティリティに関する注意事項

4.6.1. -OGJについて

コンパイラオプション-Oglobal_jump(-OGJ)、アセンブラオプション-JOPT、及びリンクオプション-JOPT を使用した場合、リンカージェディタ ln30 のオプション-ORDER もしくは-LOC を複数個指定すると、最後に指定した-ORDER もしくは、-LOC のみが有効となり、正常にリンクできません。

その結果、

-ORDER を複数指定した場合は、リンク時にエラーとなります。

-LOC を複数指定した場合は、誤った配置になります。

-ORDER 及び-LOC は1つずつ指定するようにしてください。

4.6.2. -OLU,-OFFTI,インライン関数を併用する場合について

コンパイルオプション `-Oloop_unroll(-OLU)` と `-Ofoward_function_to_inline(-OFFTI)` を併用し、`-Oloop_unroll` の対象 (ループが展開される) となる繰り返し文中にインライン展開される関数記述すると、アセンブルエラー

ファイル名 行番号 Error (asp30): Undefined symbol exist 'シンボル名 '

が発生する場合があります。

本エラーが発生した場合は、該当する繰り返し文中にダミーの `asm()` を挿入してください。

4.6.3. -ffar_pointer(-fFP)について

`-ffar_pointer` を使用した場合、`near` 属性の変数のアドレスを取得する `&` 演算子を使用すると、16 ビットで扱われます。`&` 演算子の前に `far` ポインタでキャストするようにしてください。

また、`sizeof` でポインタサイズを取得した場合、戻り値は 2 となります。プロトタイプ宣言のない関数を呼び出すとアドレスを 2 バイトしか積みません。必ずプロトタイプ宣言をしてください。

4.6.4. アセンブラ記述スタートアップファイル(ncrt0.a30,sect30.inc,nc_define.inc)の扱いについて

ご使用のマイコン機種、お客様のシステムにあわせてスタートアップファイルは変更していただく必要があります。機種により変更を要する内容は対応機種のデータブック等を参照いただき添付のスタートアップファイルを修正ください。

4.6.5. 標準入出力関数について

`printf` 関数等の標準入出力関数は、多くの RAM を消費します。そのため [R8C ファミリー MCU 用ライブラリ \(-R8C オプション指定時\)](#) において、標準入出力関数をご使用になる場合は、`%e,%E,%f,%g,%G` の変換指定記号は使用できません。

4.6.6. インクルードファイルの検索に関する注意事項

`#include` の記述においてドライブ名付きで記述し、コンパイル対象となるファイルが存在するディレクトリとは異なったディレクトリからコンパイルした場合、インクルードファイルを検索できない場合があります。

例)

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

```
C:\user>nc30 \user\test\sample.c -silent
```

```
[Error(cpp30.21):\user\test\sample.c, line 1] include file not found 'c:\user\test\sample.h'
```

4.6.7. インラインアセンブル機能(#pragma ASM~#pragma ENDASM、asm関数)に関する注意事項

- 関数外で#pragmaASM/ENDASM、asm 関数をご使用になる場合のデバッグ情報について
#pragma ASM/ENDASM 内の記述に対して、アセンブル及びリンク時のエラーメッセージの行数、デバッグ情報の行情報等が正常に出力されない場合があります。
- #pragma ASM~#pragma ENDASM、asm 関数内の記述について
 - (1) コンパイラは、レジスタの生存区間、変数の生存区間についてプログラムフローを解析して処理を行っているため asm 関数などでフローに影響を与えるようなブランチ(条件ブランチ含む)を記述しないようにしてください。
 - (2) コンパイラはレジスタを介して渡される引数およびレジスタ変数に対して、これらの有効範囲を解析しコードを生成します。しかし、インラインアセンブル機能(#pragma ASM~#pragma ENDASM または asm 関数)を使用してレジスタ値を操作する記述を行った場合、C コンパイラはインラインアセンブル機能で記述されたプログラム部分で有効となるこれらの引数およびレジスタ変数の範囲の情報を保持することができません。したがって、インラインアセンブル機能を使用してレジスタを操作する記述を行う場合は、必ずレジスタの退避・復帰を行ってください。

4.6.8. _Bool型を使用しているプログラムのデバッグに関する注意事項

_Bool 型を使用したプログラムをデバッグする場合、デバッガが_Bool 型に対応しているかご確認ください。
_Bool 型に対応していないデバッガをご使用になる場合は、コンパイル時にデバッグオプション "-gbool_to_char(-gBTC)" をご使用ください。

4.6.9. 前処理命令#defineに関する注意事項

マクロ ULONG_MAX と同一値になるマクロを定義する場合は、必ず接尾語 UL を付けてください。

4.6.10. マクロ定義に関する注意事項

マクロの定義内容にそのマクロ自体の名前を使用している場合、他の関数形式マクロの引数にそのマクロを指定すると、正しくマクロ置換されません。

- 発生例

```
int    a = 10;
#define a      a + a                // マクロ名 a
#define p( x,y )x + y

void   func(void)
{
    int    i = p ( a, a );          // i = 80 になる。
                                           // 正しくは i = 40
}
```

- 回避策

関数形式マクロの引数に渡すマクロは、その定義内容で使用しない名前でご定義してください。

```
int    a = 10;
#define b      a + a                // a とは異なるマクロ名に変更する
#define p( x,y )x + y

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


4.6.11. #if指令に関する注意事項

#if 指令の定数式がシフトで、そのシフトの左オペランドが負の値で、かつ右オペランドが **unsigned** 型の値である場合、シフト結果に対して正しく判定することができません。

- 発生例

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

    #if (-1 << 1U ) > 0           // 真と判断
        a = 1;                   // (-1 << 1U) は -2 のため偽が正しい
    #else
        a = 2;
    #endif
}
```

- 回避策

シフトの左オペランドが負の値の場合は、そのシフトの右オペランドを **signed** 型の値にしてください。

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

    #if (-1 << 1 ) > 0           // U接尾語を使用しないことでシフトの右オペランドを signed 型にする
        a = 1;
    #else
        a = 2;
    #endif
}
```

4.6.12. 構造体を返す関数をコールすると、System Errorとなる

auto 記憶クラスである構造体変数の初期化時に構造体を返す関数の戻り値を使用すると、**System Error** が発生します。

- 発生例

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

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

- 回避策

auto 記憶クラスである構造体変数の宣言と初期化を分けてください。

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

4.6.13. コンパイルオプション-Ostack_frame_align(-OSFA)を使用する場合の注意事項

コンパイルオプション-Ostack_frame_align(-OSFA)を使用した場合に、コンパイラが生成するインスペクタ情報やスタック使用量表示ファイル(拡張子.stk)のスタックサイズが誤った値になる場合があります。このため、インスペクタ情報を使用する STK ビューワや CallWalker、およびスタック使用量表示ファイルを使用するスタックサイズ算出ユーティリティ stk30 の算出するスタックサイズが誤った値になる場合があります。

[ツールニュース: <http://tool-support.renesas.com/jpn/toolnews/070701/tn5.htm>]

- 発生条件

以下の条件をすべて満たす場合に発生します。

- (1) コンパイルオプション -Ostack_frame_align(-OSFA)を使用している。
- (2) コンパイルオプション -genter を指定していない
- (3) スタックフレームを構築しない関数がある。

以下の条件をすべて満たす場合にスタックフレームを構築しません。

- ・関数はスタックを介して渡される引数を持たない。
- ・自動変数がない(register 指定された自動変数を除く)または、コンパイラの最適化により自動変数が削除されている。
- ・コンパイラがテンポラリ変数を生成していない。

- 発生例

```
void sub(unsigned int);
void func(void) /* 発生条件(3) */
{
    sub(10);
}
```

- 回避策

ご使用中のコンパイルオプション-Ostack_frame_align(-OSFA)を使用しない、もしくは、-genter を追加してください。

4.6.14. 右シフト演算に関する注意事項

最適化のコンパイルオプションを選択し、32 ビットデータを 11～15 ビットの範囲で右シフトをした結果を直接 16 ビット長の変数に格納、あるいはキャストした場合に System Error が発生する場合があります。
[ツールニュース: <http://tool-support.renesas.com/jpn/toolnews/070716/tn4.htm>]

- 発生条件

以下の条件すべてに該当する場合に発生することがあります。

- (1) プログラムの実行速度の高速化や ROM 容量を最小にする最適化オプション (-O, -OR, -OR_MAX)のいずれかを使用している。
- (2) 32 ビット長のデータを 11～15 ビットの範囲で右シフトをしている。
- (3) 上記(2)の実行結果を直接 16 ビット長の変数に格納、あるいはキャストしている。
- (4) 上記(2)のシフト対象および(3)の格納先がコンパイラによりレジスタに割り当てられている。

- 発生例

```
int i;
long l;

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

- 回避策

32 ビット長データの右シフト結果を、直接 16 ビット長の変数へ格納、またはキャストせずに、一旦 32 ビット長の変数に代入した後に 16 ビット長の変数へ格納してください。

例) 発生例の場合の回避方法

```
int i;
long l,ll;

ll = (int)( l >> 15 ); /* 一旦 32 ビット長の変数に代入する */
i = (int)ll;
```

4.6.15. utl30 の注意事項

C コンパイラユーザーズマニュアル「付録G SBDATA 宣言&SPECIAL ページ関数宣言ユーティリティ (utl30)」の Page363 にて、SPECIAL ページベクタ定義ファイル(special.inc)をスタートアップ(sect30.inc)中でインクルードするという記述がありますが、これは、V.5.40 よりも古いバージョンを対象とした説明です。

V.5.40 以降では、SPECIAL ページベクタ定義ファイルは不要となりましたので、使用しないようお願いいたします。

4.6.16. MapViewerの注意事項

Windows Vista 環境では MapViewer の HELP 機能はご使用いただけません。

Windows Vista 環境では、EcxMap 及び CallWalker をご使用ください。

4.6.17. メモリ管理関数malloc()、calloc()およびrealloc()の注意事項

NC30WA のメモリ管理関数”malloc()”、”calloc()”および”realloc()”は、一度に 64KB 以上の領域を確保することはできません。

4.6.18. 不完全型構造体または共用体の型定義に関する注意事項

不完全型の構造体型または共用体型(タグのみが定義されたもの)を `typedef` を用いて定義し、その後にメンバを定義する場合、`typedef` 名を用いて宣言した構造体または共用体に含まれるメンバがデバッガで表示されないことがあります。

なお、`typedef` より前にメンバが定義されている場合には、問題なくデバッガでメンバを表示できます。

- 発生例

```
typedef struct str1 str1_t; /* 不完全型の構造体を定義 */

struct str1 {                /* 後からメンバを定義 */
    int i;
    int j;
}

str1_t s = { 1, 2 };         /* typedef 名での構造体宣言 */
```

5. V.5.45 Release 00 からのバージョンアップ内容

5.1. Cコンパイラ

5.1.1. Cコンパイラ不具合改修

- リンク時のエラーに関する注意事項を修正しました。
該当ツールニュース
<http://tool-support.renesas.com/jpn/toolnews/091116/tn3.htm>
- 使用スタックサイズを計算する際の注意事項を修正しました。
該当ツールニュース
<http://tool-support.renesas.com/jpn/toolnews/091116/tn2.htm>
- 可変ベクタテーブルの自動生成機能に関する注意事項を修正しました。
該当ツールニュース
<http://tool-support.renesas.com/jpn/toolnews/091001/tn4.htm>

5.2. High-performance Embedded Workshop

5.2.1. High-performance Embedded Workshop不具合改修

- 統合開発環境 High-performance Embedded Workshop で、SQMlint(MISRA C ルール チェッカ)を使用する場合の注意事項を修正しました。
該当ツールニュース
<http://tool-support.renesas.com/jpn/toolnews/090801/tn1.htm>

6. ソフトウェアのバージョン一覧

C コンパイラパッケージ V.5.45 Release 01 に含まれているソフトウェアの各バージョンは以下のとおりです。

- nc30 V.6.03.22.003
- cpp30 V.4.10.00.000
- ccom30 V.5.32.04.003
- aopt30 V.1.04.04.001
- sbauto V.1.00.00.000
- as30 V.5.15.00.001
- mac30 V.3.42.00.000
- pre30 V.1.10.12
- asp30 V.5.14.00.000
- ln30 V.5.15.00.001
- lb30 V.1.02.00.000
- lmc30 V.4.02.01.000
- xrf30 V.2.02.00.000
- abs30 V.2.11.00.000
- utl30 V.1.00.09.001
- MapViewer V.3.01.02
- genmap V.1.00.01.001
- gensni V.1.00.00.002

7. MISRA Cルール適合に関して

7.1. 標準関数ライブラリ

M3T-NC30WAの標準関数ライブラリのCソースコードは、MISRA Cルールに対していくつかのルール違反¹が認められますが、これらの違反は動作に支障はありません。

7.1.1. ルール違反の要因

M3T-NC30WA の標準関数ライブラリの C ソースコードにおいて、ルール違反となった主な要因は次の通りです。

- C コンパイラの仕様 (near/far 修飾、asm()関数、#pragma)
- ANSI 規格に基づく関数の宣言
- 条件文における評価順序をカッコ()により明示的に記述していない
- 暗黙の型変換

7.1.2. ルール違反となった検査番号

ルール違反になった検査番号は次のとおりです。

| | | | | | | | | | |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 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. 評価環境

| | |
|--------------|--|
| コンパイラ | M3T-NC30WA V.5.30 Release 1 |
| コンパイルオプション | -O -c -as30 "-DOPTI=0" -gnone -finfo -fNII -misra_all -r \$*.csv |
| MISRA C チェッカ | SQMLint V.1.00 Release 1A |

7.2. HEW (High-performance Embedded Workshop) が自動生成するソースコード

HEW (High-performance Embedded Workshop) が自動生成するソースコードは、MISRA C ルールに対し、いくつかのルール違反が認められますが、これらの違反は動作に支障はありません。

7.2.1. ルール違反の要因

HEW 生成ソースコードにおいて、ルール違反となった主な要因は次の通りです。

- C コンパイラの仕様 (#pragma 等)
- ヘッダで定義された変数のスコープ
- ビットフィールドで使用する型の定義

¹ MISRA C ルールチェッカ SQMLint による検査結果値です。

7.2.2. ルール違反となった検査番号

ルール違反になった検査番号は次のとおりです。

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

7.2.3. 評価環境

コンパイラ M3T-NC30WA V.5.45 Release 00
 コンパイルオプション -c -misra_all
 MISRA C チェッカ SQMlint V.1.03 Release 00

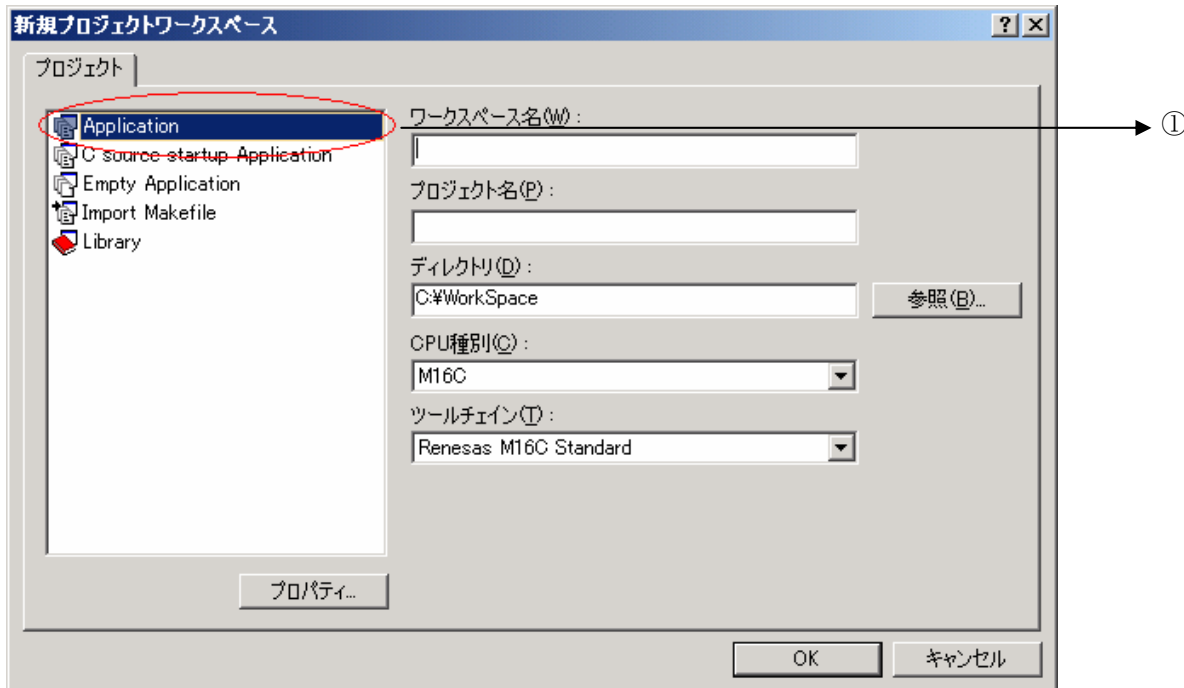
7.2.4. C スタートアップ中で使用する #pragma 拡張機能 (Misra C ルール 99)

| 拡張機能 | 宣言ファイル | 内容 | 機能 | 補足 |
|---------------------|---|---------------------------------------|---|-------------------------------|
| #pragma STACKSIZE | resetprg.h | ユーザスタックサイズを定義します。 | スタックセクション (stack) の出力及び、スタックのトップラベル名を生成します。 | コンパイラ専用の拡張機能です。 |
| #pragma ISTACKSIZE | resetprg.h | 割り込みスタックサイズを定義します。 | 割り込みスタックセクション (istack) の出力及び、割り込みスタックのトップラベル名を生成します。 | コンパイラ専用の拡張機能です。 |
| #pragma CREG | resetprg.h | MCU の内部レジスタを宣言します。 | 本 pragma で宣言された内部レジスタにアクセスする場合、専用命令を使用してアクセスするコードを生成します。 | コンパイラ専用の拡張機能です。 |
| #pragma sectaddress | resetprg.h fvector.c | セクションの定義を行います。同時に配置アドレスの宣言をすることができます。 | 本 pragma で宣言されたセクション名でセクション定義を行います。同時にアドレス指定された場合は、疑似命令 .org を使用したアドレス定義を出力します。 | コンパイラ専用の拡張機能です。 |
| #pragma entry | resetprg.h | reset 時に実行する関数を宣言します。 | 本 pragma で宣言された関数に対してスタックフレームを構築する enter 命令を出力しません。これは、スタックポインタ初期化前に enter 命令を生成しないようにするためです。 | コンパイラ専用の拡張機能です。 |
| #pragma interrupt/V | fvector.c | ベクタテーブルを生成します。 | 本 pragma で宣言された関数に対して割り込みベクタのみを定義します。 | コンパイラユーザーズマニュアルの付録 B を参照ください。 |
| #pragma interrupt | intprg.c fvector.c | 割り込み関数を宣言します。 | 本 pragma を使用して宣言された関数に対して、割り込み関数のコードを生成します。 | コンパイラユーザーズマニュアルの付録 B を参照ください。 |
| #pragma section | heap.c resetprg.c initsct.h resetprg.c firm.c | セクション名を変更します。 | 本 pragma で定義されたセクション名に変更します。 | コンパイラユーザーズマニュアルの付録 B を参照ください。 |
| #pragma ADDRESS | 各 sfr ヘッダ ファイル | I/O のアドレス定義及び変数宣言を行います。 | 本 pragma で定義された sfr に対して .equ でアドレス定義を行います。 | コンパイラユーザーズマニュアルの付録 B を参照ください。 |

8. C言語スタートアップについて

V.5.40 Release 00(A)以前のコンパイラでは、コンパイルする事はできません。

なお、アセンブラで記述された `ncrt0.a30,sect30.inc,nc_define.inc` は今まで通りご使用いただけます。
`ncrt0.a30,sect30.inc,nc_define.inc` を使用する場合は、



新規プロジェクトワークスペースで、①で示す「Application」を選択してください。

8.1. 生成ファイル

C言語スタートアップには、以下のファイルがあります。

- (1) `resetprg.c`
マイコンの初期設定を行います。
- (2) `initsct.c`
各セクションの初期化（ゼロクリア、初期値転送）を行います。
- (3) `heap.c`
ヒープ領域を確保します。
- (4) `fvector.c`
固定ベクタテーブルの定義を行います。
- (5) `intprg.c`
可変ベクタ割り込みのエントリ関数を宣言します。
- (6) `firm.c/firm_ram.c` (変更の必要はありません)
OnChipDebugger 選択時の FoUSB/E8 の `firm` が使用するプログラム領域及びワークスペース領域をダミーとして確保します。
- (7) `cstartdef.h`
スタックサイズ、ヒープサイズ等の各 `define` 値を定義しています。
- (8) `initsct.h` (変更の必要はありません)

各セクションを初期化する処理（アセンブラマクロ）を記述しています。

(9) `resetprg.h`

各ヘッダファイルをインクルードしています。

(10) `typedefine.h`

(変更の必要はありません)

各型に対して `typedef` 宣言をしています。

(11) `sfrXX.h,sfrXX.inc`

プロジェクト作成時に選択した CPU に対応して `sfr` 定義ヘッダファイルをワークスペースへ登録します。

8.2. 各生成ファイルの処理

- `resetprg.c` (必須)

本ファイルは、選択した MCU (M16C もしくは、R8C) によって内容が変わります。

```

#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(); // initlalize each sections ..... ⑪
#if __STACKSIZE__!=0
    _sp_ = &_stack_top; // set user stack pointer ..... ⑫
#else
    _isp_ = &_istack_top; // set interrupt stack pointer ..... ⑫
#endif
#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 ..... ⑮
    main(); // call main routine ..... ⑮

    exit(); // call exit
}

```

- ① スタート関数は `interrupt` セクションに配置します。
- ② CPU 初期化関数 `start()` 本体を宣言します。
- ③ 割り込みスタックポインタを初期化します。
- ④ プロテクトレジスタを“書き込み許可”に設定します。
- ⑤ プロセッサモードレジスタを“シングルチップモード”に設定します。
モードを変更する場合は、この式を変更する必要があります。
- ⑥ プロテクトレジスタを“書き込み禁止”に設定します。
- ⑦ U フラグを設定します。
ワークスペース作成ウィザードで“ユーザスタックを使用する”を選択した場合は、ユーザスタックポインタを設定します。
- ⑧ ワークスペース作成ウィザードで“ユーザスタックを使用する”を選択した場合に、ユーザスタックポインタを初期化します。
- ⑨ SB レジスタを `0x400` 番地に設定 (RAM の先頭アドレスを設定) します。
- ⑩ 可変ベクタアドレスを INTB レジスタへの設定します。
- ⑪ 各セクションの初期化 (ゼロクリア、初期値転送) を行います。
- ⑫ セクション初期化後にスタックポインタの再初期化を行います。
- ⑬ ヒープ領域の初期化を行います。
メモリ管理関数を使用する場合は、本関数の呼び出しを有効にする必要があります。
- ⑭ 標準入出力関数用を初期化を行います。
標準入出力関数を使用する場合は、本関数の呼び出しを有効にする必要があります。
- ⑮ `main` 関数を呼び出します。

- `initsct.c` (必須)

本ファイルは、選択した MCU(M16C/R8C)により内容は変わります。

```

void initsct(void)
{
    sclear("bss_SE","data,align");          -----①
    sclear("bss_SO","data,noalign");
    sclear("bss_NE","data,align");
    sclear("bss_NO","data,noalign");
#ifdef __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");
#ifdef __NEAR__
    scopy_f("data_FE","data,align");        -----④
    scopy_f("data_FO","data,noalign");
#endif
}

```

- ① `sclear` : near 領域の `bss` セクションをゼロクリアします。
`#pragma SECTION bss` 機能を用いて `bss` セクション名を変更、追加した場合は、`NE/NO` をセットで変更、追加が必要
`sclear("セクション名_NE", "data,align");`
`sclear("セクション名_NO", "data,noalign");`

例) `#pragma section bss bss2` でセクション追加した場合

```
sclear("bss2_NE", "data,align");
sclear("bss2_NO", "data,noalign");
```

を `initsct.c` へ追加します。

- ② `sclear_f` : far 領域の `bss` セクションをゼロクリアします。
 -R8C オプション指定時以外は有効になります。
- ③ `scopy` : near 領域の `data` セクションに対して初期値を転送します。
`#pragma SECTION data` 機能を用いて `data` セクション名を変更、追加した場合は、`NE/NO` をセットで変更、追加が必要
`scopy("セクション名_NE", "data,align");`
`scopy("セクション名_NO", "data,noalign");`

例) `#pragma section data data2` でセクション追加した場合

```
scopy("data2_NE", "data,align");
scopy("data2_NO", "data,noalign");
```

を `initsct.c` へ追加します。

- ④ `scopy_f` : far 領域の `data` セクションに初期値を転送します。
 -R8C オプション指定時以外は有効になります。

- `heap.c` (malloc などのメモリ管理関数を使用する場合のみ必要)

```
#pragma SECTION      bss      heap      -----①
_UBYTE heap_area[_HEAPSIZE_];      -----②
```

- ① `heap` 領域を `heap_NE` セクションに配置します。
 ※ヒープサイズを奇数バイトにした場合は、`heap_NO` セクションになります。
- ② ヒープ領域を `_HEAPSIZE_` で定義されたサイズ分確保します。

● fvector.c (必須)

```

#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 固定ベクタテーブルのセクションとアドレスを出力します。
 ※本 **pragma** は、スタートアップ用ですので、通常は使用できません。
- ② リセット以外の固定ベクタをダミー関数 (`_dummy_int`) で埋めます。
#pragma interrupt/v 関数名
 は、関数名をベクタに登録します。関数の本体を記述する場合は、本宣言とは別に **#pragma interrupt** を用いて関数を定義してください。
- ③ エントリ関数を定義します。
 リセット時の実行関数を固定ベクタへ登録します。

● intprg.c (マイコン品種毎、必要に応じて)

```

// DMA0 (software int 8)
#pragma interrupt _dma0(vect=8) -----①
void _dma0(void){ -----②

// 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){

(省略)

```

- ① 可変ベクタ割り込み関数を宣言します。
 各可変ベクタ割り込み関数に対応した関数を宣言します。同時に可変ベクタテーブルを生成します。

- ② 可変ベクタ割り込み関数を定義します。
使用する割り込みベクタ番号に対応する関数に処理を記述してください。

例) 割り込みベクタ番号 9 番 (DMA1) を使用する場合

```
#pragma interrupt      _dma1(vect=9)
void _dma1(void)
{
    //処理を記述
}
```

- ③ `intprg.c` が不要な場合
ファイルの登録から削除してリンク対象からはずしてください。

- `firm.c/firm_ram.c` (OnChipDebugger FoUSB/E8 選択時)
本ファイルの内容は変更しないでください。
本ファイルの内容は、マイコン及び FOUSB/E8 選択により変更されます。

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

#pragma section bss FirmRam                          -----②

#ifdef __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 E8 を使用する場合に有効にします。
② E8 のファームウェアが使用する work ram 領域を FirmRam_NE セクションに確保します。
③ work ram 領域を __WORK_RAM__ で定義されたサイズ分確保します。
④ E8 のファームウェアプログラムを FirmArea セクションに配置します。
⑤ ファームウェアプログラムのサイズを指定します。
⑥ FoUSB のファームウェアが使用する work ram 領域を FirmRam_NE セクションに確保します。
⑦ work ram 領域を 0x80 バイト確保します。(対応するマイコンの品種により異なります。)
⑧ FoUSB のファームウェアプログラムを FirmArea セクションに配置します。
⑨ ファームウェアプログラムのサイズを指定します。

- cstartdef.h (必須)

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

- ①ワークスペース作成ウィザードで入力したスタックサイズに応じて変化します。
- ②ワークスペース作成ウィザードで入力した割り込みスタックサイズに応じて変化します。
- ③ワークスペース作成ウィザードで入力したヒープサイズに応じて変化します。
- ④ワークスペース作成ウィザードで“標準入出力関数を使用する”を選択した場合に、1 が設定されます。
- ⑤リセット直後に WATCH DOG 機能を有効にする場合は、1 を設定します。(R8C ファミリのみ)

上記を新規ワークスペース作成後に再度変更を行う場合は、本ファイルを直接変更してください。

- initsct.h (必須)

本ファイルの内容は変更しないでください。

- resetprg.h (必須)

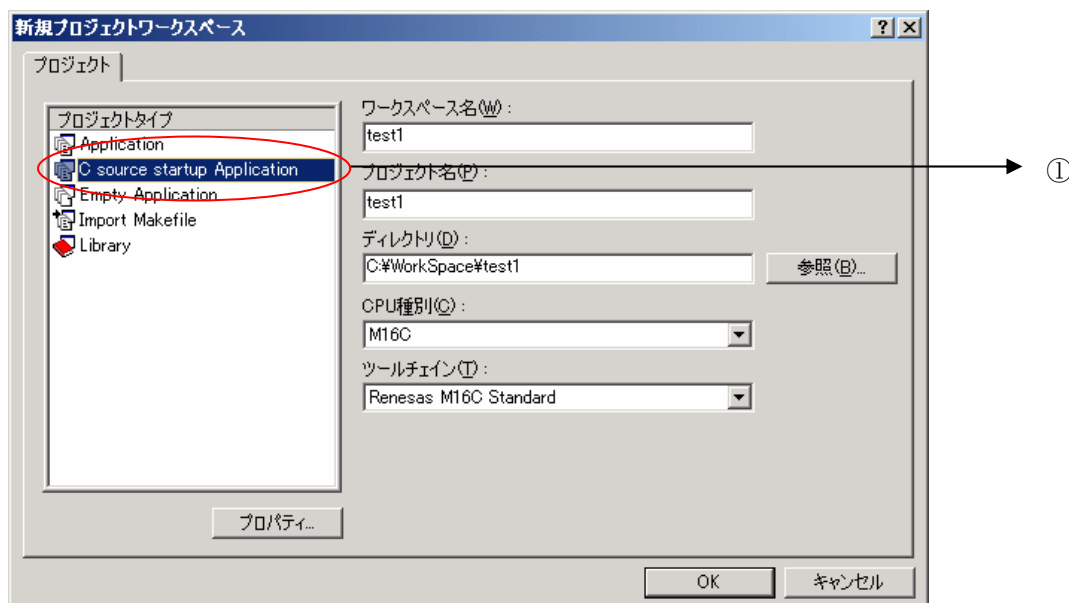
オンチップデバッガをご使用の際は、本リリースノート『3.3.5. オンチップデバッガ選択時の FirmRam_NE セクションと SB レジスタの値に関して』をご参照ください。

- typedefine.h (必須)

本ファイルの内容は変更しないでください。

8.3. C言語スタートアップの生成方法

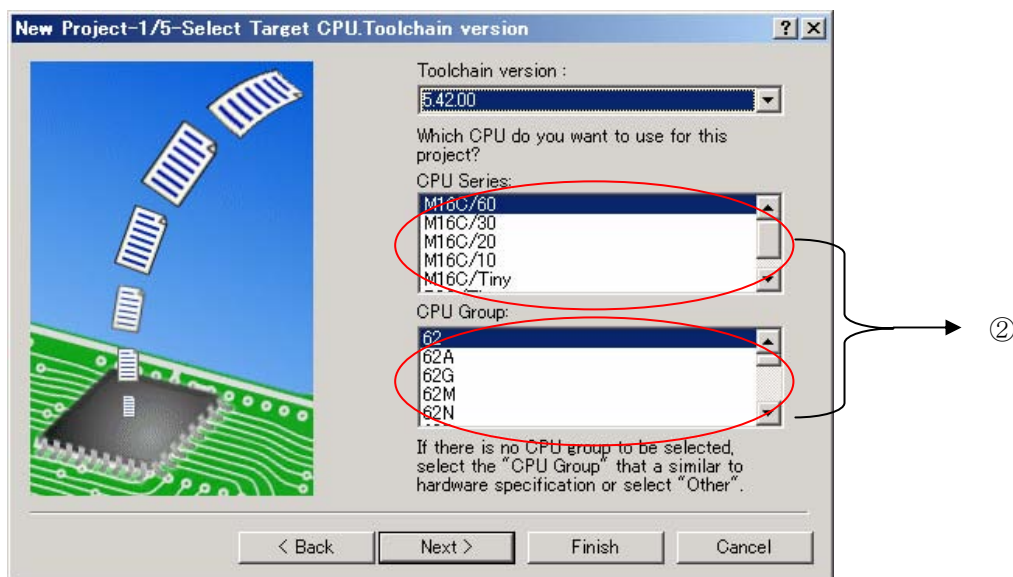
- C 言語スタートアップを使用したプロジェクトの選択



①左窓の **C source startup Application** を選択します。

※ 複数コンパイラをインストールしている場合で、C source startup Application 選択後 CPU 種別で、他マイコンを選択した場合、C source startup Application へのフォーカスが Application へ移動して、C ソーススタートアップの選択が無効になりますので、再度 C source startup Application を選択してください。

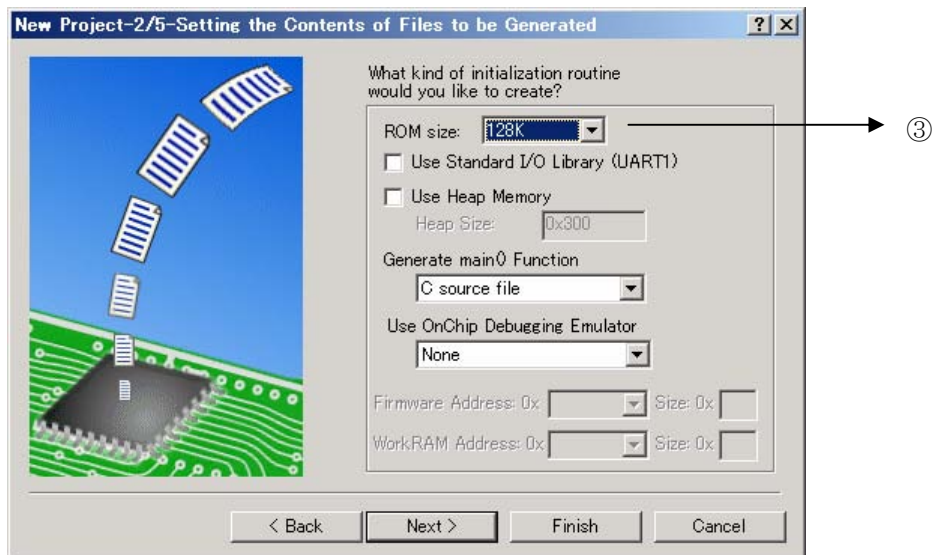
- マイコン品種の選択



②CPU Series と CPU Group からマイコン品種を選択します。

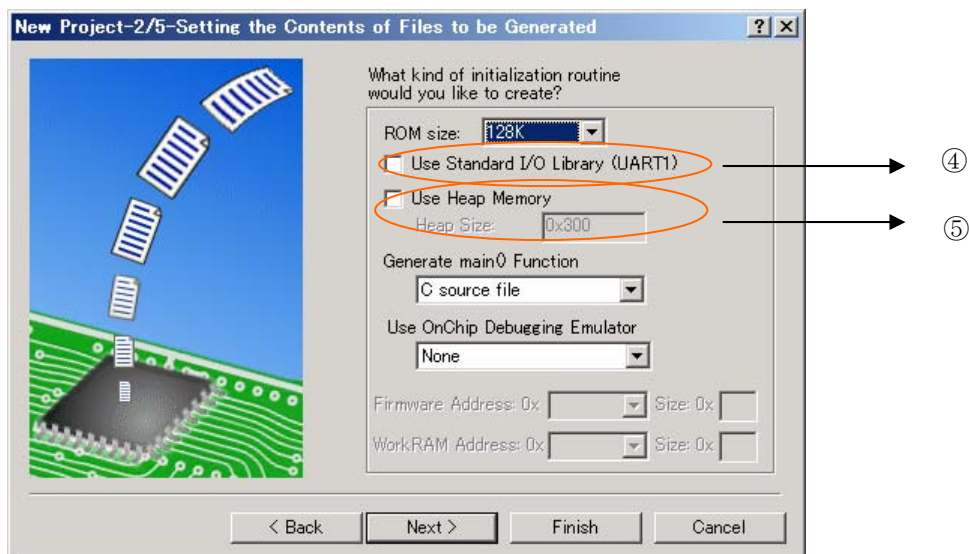
この選択により、対応する sfr ヘッダファイルがワークスペースへ登録されます。
また、可変ベクタエントリ関数 (intprg.c) が登録されます。

- ROM サイズの選択



③で選択する ROM サイズは、オンチップデバッガ選択時の設定に加えて、リンク時の ROM 属性のセクションを ROM サイズに応じて適切に配置するようにします。

- 標準入出力関数ライブラリとメモリ管理関数ライブラリ使用時の設定



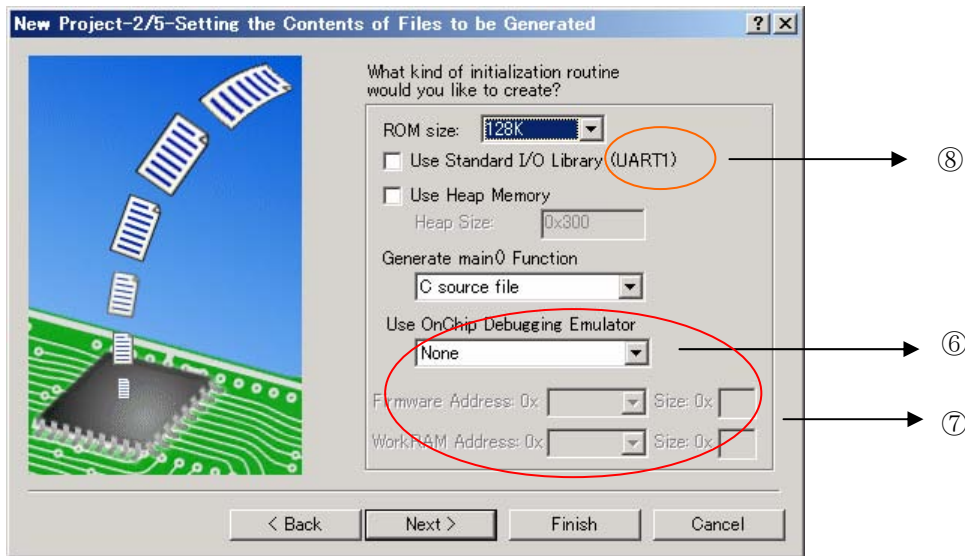
④標準入出力関数ライブラリを使用する場合に、チェックします。

チェックが行われる事により、resetprg.c 中の _init() 呼び出しが有効になります。
また、device.c と init.c がプロジェクトに登録されます。

⑤メモリ管理関数を使用する場合に、チェックします。

チェックが行われる事により、resetprg.c 中の heap_init() 呼び出しが有効になります。
また、heapdef.h ,heap.c がプロジェクトに登録されます。

- OnChipDebugger の選択



⑥ OnChipDebugger を使用する場合に選択します。

選択可能なデバッガは、FoUSB と、E8 になります。

ただし、選択するマイコン品種により、いずれか一方、もしくは両方選択できない場合があります。

この選択により、firm.c が登録され、⑦に表示のデバッガが占有する領域を変数領域として確保することにより、ユーザプログラムとの重複を回避します。

⑦ Firmware Address と workRamAddress の設定を行います。

FoUSB/E8 が占有する、Firmware 用のプログラム領域と work 用の RAM 領域の

設定を行います。使用するデバッガでアドレス変更が可能な場合のみ設定変更できます。

デバッガ使用時にこれらのアドレスを変更した場合は、デバッガ使用時の設定に合わせて変更してください。

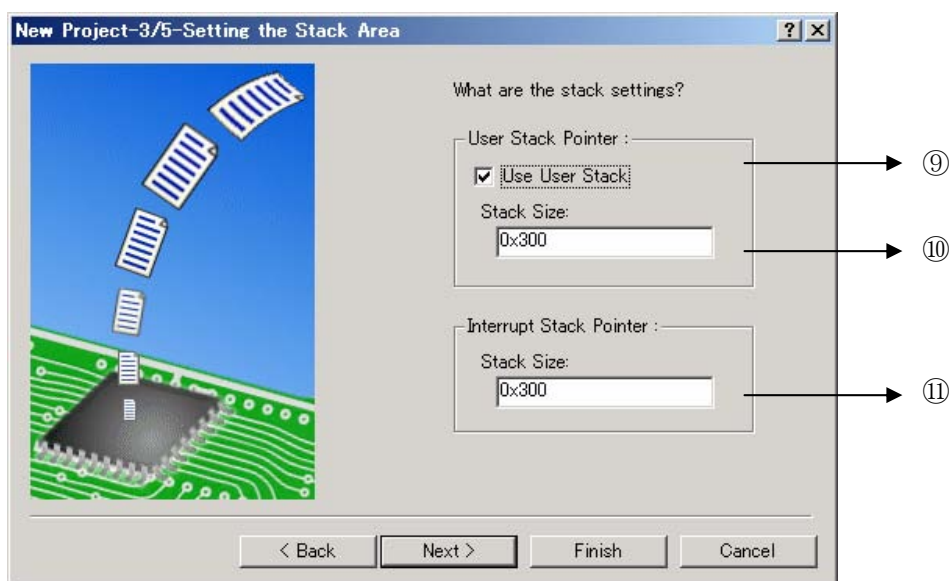
変更する際の各アドレス、サイズについては、使用するデバッガのマニュアルを参照ください。

⑧ 標準入出力関数ライブラリを選択した状態で OnChipDebugger を選択すると、

(UART1) の表示が (UART0) に変わります。

これは、標準入出力関数及び OnChipDebugger が共に UART1 を使用するため標準入出力側を UART0 へ変更することを意味しています。

- スタックサイズを選択



⑨ユーザスタックの使用有無を選択します。

チェックをしなかった場合は、start 関数内でユーザスタックを使用しない設定に変更します。

⑩ユーザスタックサイズを設定します。

cstartdef.h 内の define 値を変更します。

⑪ユーザスタックサイズを設定します。

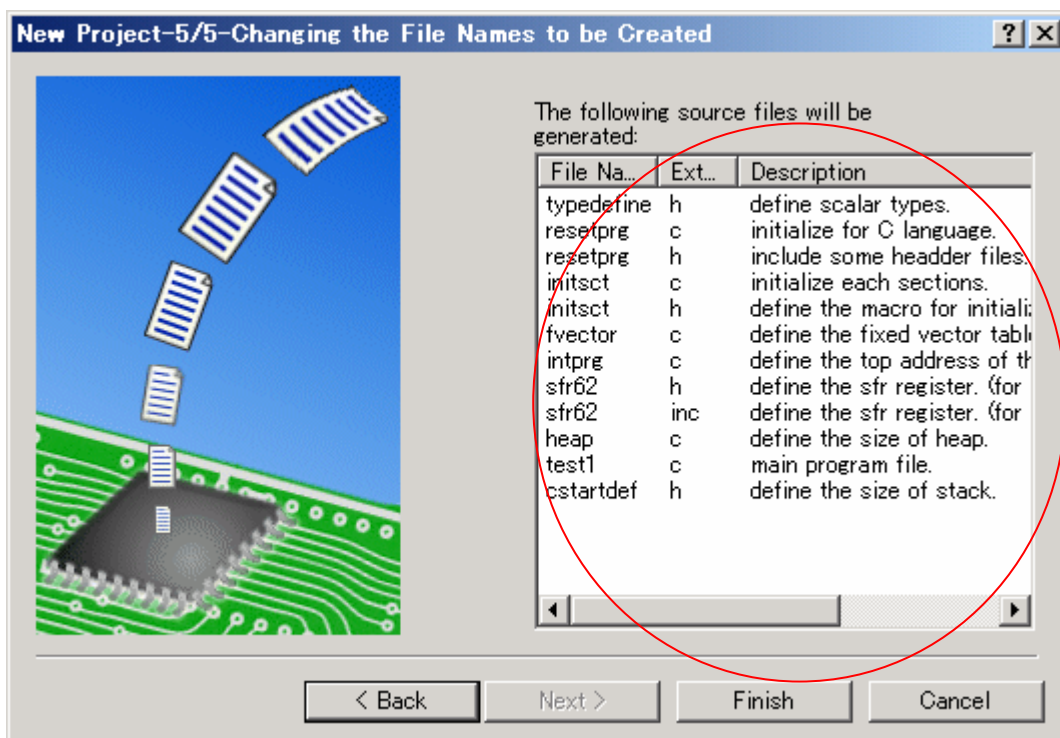
cstartdef.h 内の define 値を変更します。

プロジェクト作成後、スタックサイズ及び HEAP サイズを変更する場合は、cstartdef.h 内の設定でそれぞれ

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

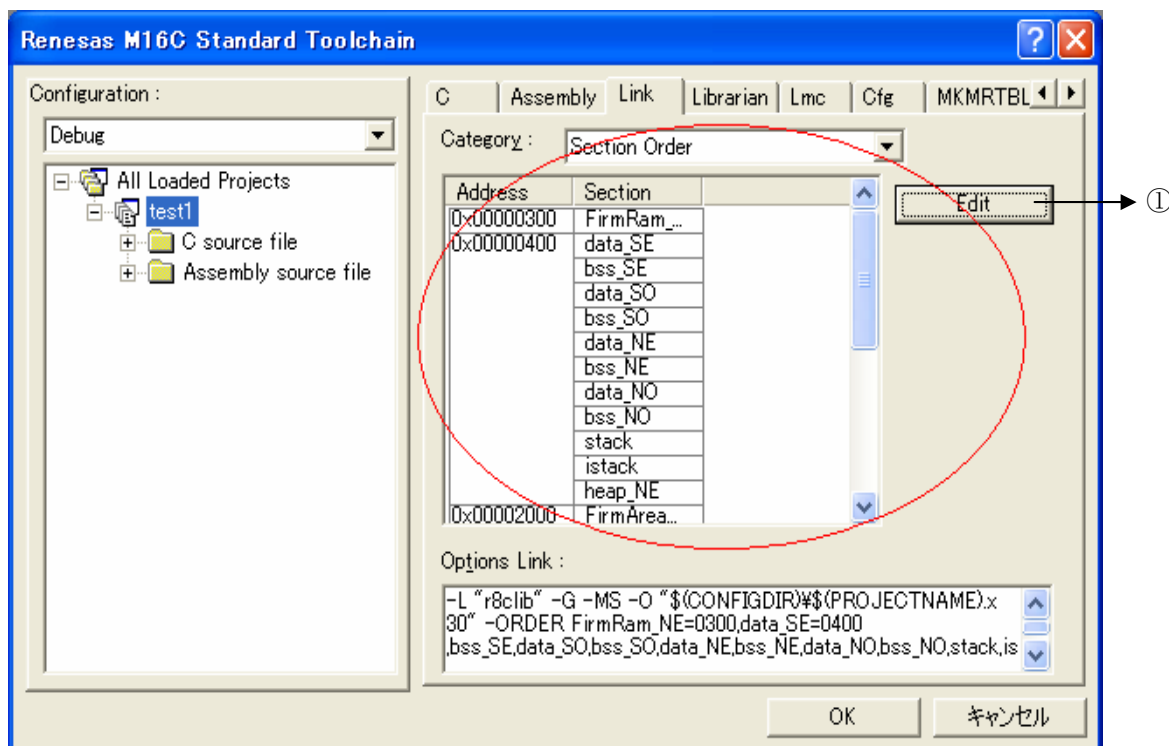
の値を変更してください。

- 登録ファイル一覧



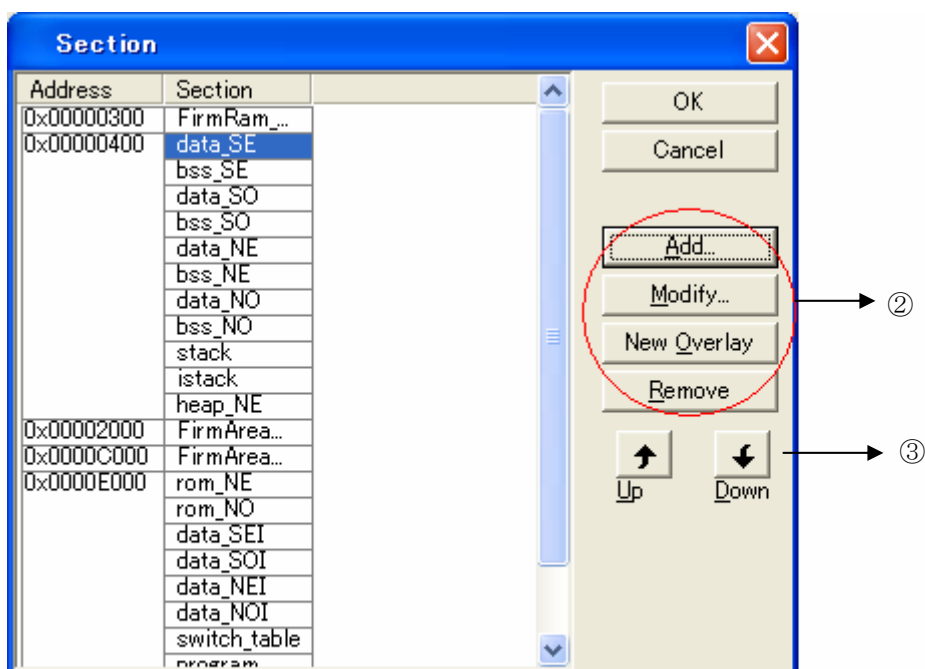
ここで、登録されるファイル一覧が確認できます。

- セクションオーダー



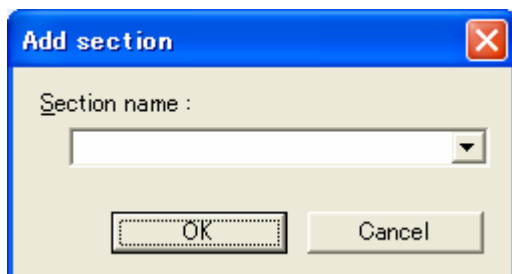
各セクションのリンク順及びリンクアドレスは、[Renesas M16C Standard Toolchain]→[Link]

の Category: Section Order で確認することができます。



#pragma SECTION で新たにセクションを追加した場合などは、①の[Edit]ボタンを選択して、Section Window をオープンしてください。

フォーカスを Section にした状態で②の[Add]ボタンを選択してください。



Add section Window がオープンしますので、新しいセクション名を入力してください。

入力されたセクションが登録されますので、配置するエリアにそのセクションを③の UP/DOWN ボタンを使用して移動させてください。

9. マニュアルの誤記

M16C シリーズ、R8C ファミリ用 C コンパイラパッケージ V.5.45 C コンパイラユーザーズマニュアル (第 3 版)に以下の誤記があります。読み替えをお願いいたします。

E.2.2 標準関数ライブラリ機能別一覧 表 E.8 数学関数

| 関数名 | リエントラント性 (誤) | リエントラント性 (正) |
|--------|--------------|--------------|
| acos | ○ | × |
| asin | ○ | × |
| atan2 | ○ | × |
| log | ○ | × |
| log10 | ○ | × |
| pow | ○ | × |
| sqrt | ○ | × |
| acosf | ○ | × |
| asinf | ○ | × |
| atan2f | ○ | × |
| log10f | ○ | × |
| sinf | ○ | × |