

お客様各位

カタログ等資料中の旧社名の扱いについて

2010年4月1日を以ってNECエレクトロニクス株式会社及び株式会社ルネサステクノロジが合併し、両社の全ての事業が当社に承継されております。従いまして、本資料中には旧社名での表記が残っておりますが、当社の資料として有効ですので、ご理解の程宜しくお願い申し上げます。

ルネサスエレクトロニクス ホームページ (<http://www.renesas.com>)

2010年4月1日

ルネサスエレクトロニクス株式会社

【発行】ルネサスエレクトロニクス株式会社 (<http://www.renesas.com>)

【問い合わせ先】 <http://japan.renesas.com/inquiry>

ご注意書き

1. 本資料に記載されている内容は本資料発行時点のものであり、予告なく変更することがあります。当社製品のご購入およびご使用にあたりましては、事前に当社営業窓口で最新の情報をご確認いただきますとともに、当社ホームページなどを通じて公開される情報に常にご注意ください。
2. 本資料に記載された当社製品および技術情報の使用に関連し発生した第三者の特許権、著作権その他の知的財産権の侵害等に関し、当社は、一切その責任を負いません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
3. 当社製品を改造、改変、複製等しないでください。
4. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器の設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因しお客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
5. 輸出に際しては、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。本資料に記載されている当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途の目的で使用しないでください。また、当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器に使用することができません。
6. 本資料に記載されている情報は、正確を期すため慎重に作成したのですが、誤りが無いことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質水準を「標準水準」、「高品質水準」および「特定水準」に分類しております。また、各品質水準は、以下に示す用途に製品が使われることを意図しておりますので、当社製品の品質水準をご確認ください。お客様は、当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途に当社製品を使用することができません。また、お客様は、当社の文書による事前の承諾を得ることなく、意図されていない用途に当社製品を使用することができません。当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途または意図されていない用途に当社製品を使用したことによりお客様または第三者に生じた損害等に関し、当社は、一切その責任を負いません。なお、当社製品のデータ・シート、データ・ブック等の資料で特に品質水準の表示がない場合は、標準水準製品であることを表します。
標準水準： コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット
高品質水準： 輸送機器（自動車、電車、船舶等）、交通用信号機器、防災・防犯装置、各種安全装置、生命維持を目的として設計されていない医療機器（厚生労働省定義の管理医療機器に相当）
特定水準： 航空機器、航空宇宙機器、海底中継機器、原子力制御システム、生命維持のための医療機器（生命維持装置、人体に埋め込み使用するもの、治療行為（患部切り出し等）を行うもの、その他直接人命に影響を与えるもの）（厚生労働省定義の高度管理医療機器に相当）またはシステム等
8. 本資料に記載された当社製品のご使用につき、特に、最大定格、動作電源電圧範囲、放熱特性、実装条件その他諸条件につきましては、当社保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めておりますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害などを生じさせないようお客様の責任において冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、機器またはシステムとしての出荷保証をお願いいたします。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、詳細につきましては製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関し、当社は、一切その責任を負いません。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを固くお断りいたします。
12. 本資料に関する詳細についてのお問い合わせその他お気付きの点等がございましたら当社営業窓口までご照会ください。

注 1. 本資料において使用されている「当社」とは、ルネサスエレクトロニクス株式会社およびルネサスエレクトロニクス株式会社とその総株主の議決権の過半数を直接または間接に保有する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

=====
必ずお読みください
=====

R32C/100 シリーズ用 C コンパイラパッケージ
V.1.02 Release 00

リリースノート

(第 2 版)

株式会社ルネサス ソリューションズ

2009 年 10 月 1 日

概要

このたびは、R32C/100 シリーズ用 C コンパイラパッケージ V.1.02 Release 00 を採用いただきまして、誠にありがとうございます。本資料は 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いただきますようお願い申し上げます。

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. 注意事項.....	3
3.1 Microsoft Windows に関する注意事項.....	3
3.1.1 動作環境に関する注意事項.....	3
3.1.2 MapViewer に関する注意事項.....	3
3.1.3 ファイル名に関する注意事項.....	3
3.1.4 ウィルスチェックプログラムに関する注意事項.....	4
3.2 機種依存に関する注意事項.....	4
3.2.1 R32C の割り込み制御レジスタに関する注意事項.....	4
3.2.2 SFR 領域のアクセスに関する注意事項.....	5
3.2.3 .OFSA、及び ID に関する注意事項.....	5
3.3 C 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.....	5
3.3.1 -OGJ について.....	5
3.3.2 インクルードファイルの検索に関する注意事項.....	5
3.3.3 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.....	5
3.3.4 前処理命令#define に関する注意事項.....	6
3.3.5 マクロ定義に関する注意事項.....	6
3.3.6 #if..#endif 文に関する注意事項.....	6
3.3.7 メモリ管理関数 malloc()、calloc()および realloc()に関する注意事項.....	7
4. V.1.01 Release 00 からのリビジョンアップ内容.....	7
4.1 機能改定.....	7
4.1.1 コンパイルオプション“-I”の制限値変更.....	7
4.1.2 64 ビット整数型のシフト処理高速化.....	7
4.1.3 オプション機能選択領域の設定機能追加.....	7
4.2 不具合修正.....	7
5. プログラムの起動または終了.....	8
5.1.1 High-performance Embedded Workshop の起動と終了.....	8
5.1.2 Manual Navigator の起動.....	8
5.1.3 MAP ビューワ (MAP Viewer) の起動.....	8

5.1.4	スタック解析ツール (Call Walker) の起動.....	9
5.2	DOS プロンプトまたはコマンドプロンプト上でC コンパイラを使用する場合の設定.....	9
5.2.1	環境変数とパス.....	10
5.2.2	バッチファイル.....	10
6.	ソフトウェアのバージョン一覧.....	10
7.	MISRA C ルール適合に関して.....	11
7.1	標準関数ライブラリ.....	11
7.1.1	ルール違反の要因.....	11
7.1.2	ルール違反となった検査番号.....	11
7.2	C 言語スタートアップ.....	12
7.2.1	ルール違反の要因.....	12
7.2.2	ルール違反となった検査番号.....	12
7.2.3	C 言語スタートアップ中で使用する #pragma 拡張機能 (Misra C ルール 99).....	12
7.3	SFR ヘッドファイル.....	13
7.3.1	ルール違反の要因.....	13
7.3.2	ルール違反となった検査番号.....	13
7.4	評価環境.....	13
8.	C 言語スタートアッププログラムについて.....	14
8.1	C 言語スタートアッププログラムのファイル構成.....	14
8.2	C 言語スタートアッププログラムの処理.....	14
8.2.1	resetprg.c.....	14
8.2.2	resetprg.h.....	16
8.2.3	initsct.c.....	16
8.2.4	initsct.h.....	17
8.2.5	heap.c.....	17
8.2.6	heapdef.h.....	17
8.2.7	fvector.c.....	18
8.2.8	intprg.c.....	18
8.2.9	firm.c.....	19
8.2.10	cregdef.h.....	19
8.2.11	stackdef.h.....	19
8.2.12	vector.h.....	20
8.2.13	typedefine.h.....	20
8.3	High-performance Embedded Workshop でC 言語スタートアッププログラムを使用する場合.....	21
8.4	High-performance Embedded Workshop でアセンブリ言語スタートアッププログラムを使用する場合.....	25

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いたします。

http://tool-support.renesas.com/jpn/toolnews/r32c_compiler.htm

3. 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。

3.1 Microsoft Windows に関する注意事項

3.1.1 動作環境に関する注意事項

C コンパイラパッケージは、Windows¹ 2000、Windows XP および Windows Vista の環境で動作します。Windows 3.1、Windows 95、Windows 98、Windows ME および Windows NT 4 以前のバージョンでは動作しません。

3.1.2 MapViewer に関する注意事項

Windows Vista 環境では MapViewer の HELP 機能は使用できません。

Windows Vista 環境では、High-performance Embedded Workshop のマップ機能を使用してください。

3.1.3 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 文字以外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ません。
- 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 および STKViewer が 正しく表示されない。
- アブ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 出力してリンカが異常終了する。
- 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。

¹ Microsoft, Windows, Windows NT および Windows Vista は、米国 Microsoft Corporation の米国およびその他の国における登録商標です。

3.1.4 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 C コンパイラパッケージ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、ウィルスチェックプログラムの常駐を解除してから C コンパイラパッケージを起動しなおしてください。

3.2 機種依存に関する注意事項

3.2.1 R32C の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(BTSTC、BTSTS)を生成する可能性があります。BTSTC、BTSTS 命令は、R32C 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。本オプションを使用する場合は、必ず生成されたコードに問題がない事をご確認ください。

- 発生例

以下のプログラムに対して最適化オプション "-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しく行われず、意図しない動作をおこします。

```
#pragma ADDRESS ta0ic_addr    006CH /* タイマ A0 割り込み制御レジスタ */

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 */
}
```

- 回避策

以下のいずれかの方法で回避してください。

- (1) "-O5"以外の最適化オプションを選択してください。または"-O5OA"と併用して選択して下さい。
- (2) 次のように "asm 関数" を挿入することにより、最適化を抑止してください。

```
#pragma ADDRESS ta0ic_addr    006CH /* タイマ A0 割り込み制御レジスタ */

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 */
    {
        asm( );
    }
    ta0ic.ir = 0;          /* 1 になったら 0 に戻す */
}
```

- 注意
コンパイルオプションの変更または asm 関数の使用による対策後は、必ず BTSTC、BTSTS 命令が生成されていない事を確認してください。

3.2.2 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、詳しくは各機種のハードウェアマニュアルなどを参照してください。この注意事項に関わる命令は、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。

3.2.3 .OFSA、及び.ID に関する注意事項

オプション機能選択レジスタ、オプション機能選択領域、および ID コードチェック機能を搭載したデバイスを使用する場合は、各領域の値をアセンブラ指示命令で必ず設定してください。

オプション機能選択レジスタ	: .OFSA
ID コードチェック機能	: .ID

- 上記指示命令が設定されていない場合、各領域には"00"が設定されます

3.3 C 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

3.3.1 -OGJ について

コンパイラオプション-Oglobal_jmp(-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 ln100 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。

3.3.2 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。

3.3.3 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

- (1) #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。
- (2) コンパイラは、レジスタや変数の有効範囲について、プログラムフローを解析して処理を行っているため、インラインアセンブル機能(#pragma ASM ~ #pragma ENDASM または asm 関数)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。
- (3) インラインアセンブル機能を使用してレジスタの値を変更する記述をする場合、有効範囲中でレジスタの値を変更した情報を得ることができません。必ずレジスタを退避・復帰してください。

3.3.4 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。

3.3.5 マクロ定義に関する注意事項

- 内容
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
                    // (i=40 が正しい)
```

- 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);
```

3.3.6 #if...#endif 文に関する注意事項

- 内容
#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。


```
int main( 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
}
```

3.3.7 メモリ管理関数 malloc()、calloc()および realloc()に関する注意事項

-fint_16(-fi16)オプション使用時において、メモリ管理関数 malloc()、calloc()および realloc()を使用する場合、一度に64KB以上の領域を確保することができません。

4. V.1.01 Release 00 からのリビジョンアップ内容

4.1 機能改定

4.1.1 コンパイルオプション“-I”の制限値変更

コンパイルオプション“-I”で設定できるインクルードディレクトリの数の上限を50個から256個へ変更しました。

4.1.2 64ビット整数型のシフト処理高速化

64ビット整数(signed long long型およびunsigned long long型)のシフトで生成されるコードを改善し、実行速度を向上しました。

4.1.3 オプション機能選択領域の設定機能追加

オプション機能選択領域に値を設定するアセンブラ指示命令“.OFSA”を追加しました。
オプション機能の詳細については、各機種のハードウェアマニュアルを参照してください。

4.2 不具合修正

- R32C/118 および/121 グループ以外のマイコン用プロジェクト作成時の注意事項を改修しました。
該当ツールニュース
<http://tool-support.renesas.com/jpn/toolnews/071216/tn6.htm>
- プリプロセスコマンド#error を使用している場合の注意事項を改修しました。
該当ツールニュース
<http://tool-support.renesas.com/jpn/toolnews/080301/tn2.htm>
- C コンパイラの拡張機能 #pragma DMAC を使用する場合の注意事項を改修しました。
該当ツールニュース
<http://tool-support.renesas.com/jpn/toolnews/080416/tn1.htm>
- for または while 文で volatile 修飾した変数を使用する際の注意事項を改修しました。
該当ツールニュース
<http://tool-support.renesas.com/jpn/toolnews/080616/tn1.htm>
- ビットフィールド比較をする際の注意事項を改修しました。
該当ツールニュース
<http://tool-support.renesas.com/jpn/toolnews/080616/tn2.htm>

-
- メンバ選択演算子で volatile 修飾した変数を使用する際の注意事項を改修しました。
該当ツールニュース
<http://tool-support.renesas.com/jpn/toolnews/080616/tn3.htm>
 - ビット単位の and 演算子で volatile 修飾した変数を使用する際の注意事項を改修しました。
該当ツールニュース
<http://tool-support.renesas.com/jpn/toolnews/080616/tn4.htm>
 - ロードモジュールコンバータのオプション設定の注意事項を改修しました。
該当ツールニュース
<http://tool-support.renesas.com/jpn/toolnews/080701/tn2.htm>
 - 構造体のメンバを、sizeof 演算子を含む式で初期化する場合の注意事項を改修しました。
該当ツールニュース
<http://tool-support.renesas.com/jpn/toolnews/080716/tn2.htm>
 - コンパイルオプション -Wno_used_static_function(-WNUSF)に関する注意事項を改修しました。
該当ツールニュース
<http://tool-support.renesas.com/jpn/toolnews/081101/tn1.htm>

5. プログラムの起動または終了

5.1.1 High-performance Embedded Workshop の起動と終了

- 起動
Windows スタートメニューの「プログラム」の中にある「Renesas」メニューの「High-performance Embedded Workshop」メニュー内の「High-performance Embedded Workshop」をクリックします。
- 終了
「ファイル」メニューの「アプリケーションの終了」をクリックします。

5.1.2 Manual Navigator の起動

- 起動
Windows スタートメニューの「プログラム」の中にある「Renesas」メニューの「High-performance Embedded Workshop」メニュー内の「Manual Navigator」をクリックします。
オンラインマニュアルおよび添付資料の参照ができます。
- 終了
「ファイル」メニューの「アプリケーションの終了」をクリックします。
- 注意
 - (1) Manual Navigator でマニュアルを表示するためには、Adobe Reader²が必要です。
 - (2) Manual Navigator にマニュアルを登録した後、マニュアルのフォルダを移すと、マニュアルを表示できなくなります。

5.1.3 MAP ビューワ (MAP Viewer) の起動

- 起動
次の2種類の方法により、MAP Viewer を起動することができます。
 - (1) High-performance Embedded Workshop から MAP Viewer を起動する場合

² Adobe および Reader はアドビシステムズ社の商標または登録商標です。

High-performance Embedded Workshop の基本設定メニューより「カスタマイズ」をクリックします。表示されるカスタマイズダイアログボックスのメニュータブから、追加ボタンをクリックし、ツールの追加ダイアログボックスを表示してください。

次の項目を指定して OK ボタンをクリックします。

名前	MAPViewer (任意の名称)
コマンド	C:\Program Files\Renesas\Hew\Tools\Renesas ¥nc100¥v102r00¥bin¥MapViewer.exe (コンパイルインストールディレクトリにある mapviewer.exe を指定ください)
引数	\$(CONFIGDIR)\\$(PROJECTNAME).x30
初期ディレクトリ	\$(CONFIGDIR)

[ツール]メニューに名前指定した名称が追加されます。
上記名前をクリックすると MAPViewer が起動します。

- (2) Windows スタートメニューから MAP Viewer を起動する場合
Windows スタートメニューの「プログラム」「Renesas」「R32C-100 Series C Compiler V.1.02 Release 00」「MAP Viewer」をクリックします。

- 終了
MAP Viewer の「File」メニューの「Exit」をクリックします。

5.1.4 スタック解析ツール (Call Walker) の起動

- 起動
次の2種類の方法により、Call Walker を起動することができます。
 - (1) High-performance Embedded Workshop から Call Walker を起動する場合
High-performance Embedded Workshop のツールメニューより「Renesas Call Walker」をクリックします。
 - (2) Windows スタートメニューから Call Walker を起動する場合
Windows スタートメニューの「プログラム」「Renesas」「R32C-100 Series C Compiler V.1.02 Release 00」「Call Walker」をクリックします。
- 終了
Call Walker の「File」メニューの「Exit」をクリックします。
- 入力ファイル (スタック情報ファイル) の作成方法
Call Walker の入力ファイルは、.sni ファイル作成ツール gensni を使用して作成します。
Call Walker の入力ファイルの作成は、アプソリュートモジュールファイル(x30)のビルド方法により異なります。
 - (1) High-performance Embedded Workshop 上でビルドする場合
ビルド時に gensni が自動的に実行されます。
 - (2) コマンドプロンプト(またはDOS プロンプト)上でコンパイル、アセンブル、リンクする場合
gensni をコマンドプロンプト(またはDOS プロンプト)上で実行してください。

【gensni 操作例】

```
c:\> gensni -o sample.sni sample.x30
```

- 入力ファイルの選択方法
Call Walker 起動後は、[File]メニューの[Import Stack File...]から入力ファイルとしてスタック情報ファイル(拡張子.sni)を指定してください。

5.2 DOS プロンプトまたはコマンドプロンプト上で C コンパイラを使用する場合の設定

DOS プロンプトまたはコマンドプロンプト上で C コンパイラを使用する場合は、C コンパイラが使用する環境変数の

設定が必要です。

5.2.1 環境変数とパス

環境変数	用途
BIN100	C コンパイラの実行ファイル(*.exe など)を格納したディレクトリを指定します。
INC100	C コンパイラの標準インクルードファイルを格納したディレクトリを指定します。
LIB100	C コンパイラの標準ライブラリファイルを格納したディレクトリを指定します。
TMP100	C コンパイラが一時的に生成するテンポラリファイルを格納するディレクトリを指定します。
path	C コンパイラの実行ファイル(*.exe など)を格納したディレクトリを指定します。 アクセス権のあるディレクトリを指定してください。

5.2.2 バッチファイル

C コンパイラをインストールしたディレクトリにバッチファイル setnc100.bat が生成されます。setnc100.bat には C コンパイラが使用する環境変数を明記しています。

C コンパイラを DOS プロンプトまたはコマンドプロンプト上で使用する場合は、setnc100.bat を実行してください。

【バッチファイルの記述内容】

```
REM ***** R32C ツールチェーン用 環境変数 *****
SET BIN100=C:\Program Files\Renesas\Hew\Tools\Renesas\nc100\v102r00\BIN
SET BIN100=C:\Program Files\Renesas\Hew\Tools\Renesas\nc100\v102r00\LIB100
SET BIN100=C:\Program Files\Renesas\Hew\Tools\Renesas\nc100\v102r00\INC100
SET BIN100=C:\Program Files\Renesas\Hew\Tools\Renesas\nc100\v102r00\TMP
SET PATH=%BIN100%;%PATH%
```

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.1.02 Release 00 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。

・ nc100	V.2.00.06.000	コンパイルドライバ
・ igen100	V.1.00.00.000	インラインジェネレータ
・ cpp100	V.1.01.00.000	プリプロセッサ
・ ccom100	V.1.02.01.000	コンパイラ本体
・ aopt100	V.1.01.01.000	アセンブラオブティマイザ
・ as100	V.1.00.04.000	アセンブラドライバ
・ mac100	V.1.00.01.000	マクロプロセッサ
・ asp100	V.1.01.00.000	アセンブラプロセッサ
・ psfp100	V.1.00.01.000	プログラマブル音場プロセッサ
・ ln100	V.1.01.01.000	リンケージエディタ
・ lb100	V.1.00.01.000	ライブラリアン
・ lmc100	V.1.01.00.000	ロードモジュールコンバータ
・ abs100	V.1.00.01.000	アブソリュートリスタ
・ gensni	V.1.00.00.002	スタック情報解析ユーティリティ
・ genmap	V.1.00.00.000	マップ情報解析ユーティリティ
・ mapviewer	V.3.01.02	マップビューワ

7. MISRAC ルール適合に関して

7.1 標準関数ライブラリ

R32C/100 シリーズ用 C コンパイラパッケージの標準関数ライブラリの C ソースコードは、MISRAC ルールに対して 52 のルール違反³が認められますが、これらの違反は動作に支障がありません。

7.1.1 ルール違反の要因

C コンパイラパッケージ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。

- (1) C コンパイラの仕様 (near/far 修飾、asm()関数、#pragma)
- (2) ANSI 規格に基づく関数の宣言
- (3) 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
- (4) 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通りです。

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								

³ MISRAC ルールチェッカ SQMLint による検査結果値です。

7.2 C 言語スタートアップ

High-performance Embedded Workshop が出力する C 言語スタートアップの C ソースコードは、MISRA C ルールに対して 3 つ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がありません。

7.2.1 ルール違反の要因

High-performance Embedded Workshop が出力する各マイコン用の C 言語スタートアップの 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。

- (1) C コンパイラの仕様 (asm()関数、#pragma)
- (2) ANSI 規格に基づく関数の宣言

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通りです。

22 45 99

7.2.3 C 言語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
#pragma STACKSIZE	stackdef.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、およびスタックのトップレベル名を生成します。
#pragma ISTACKSIZE	stackdef.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、および割り込みスタックのトップレベル名を生成します。
#pragma CREG	cregdef.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。
#pragma sectaddress	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、指示命令“.org”を使用したアドレス定義を出力します。
#pragma entry	resetprg.c	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。
#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。
#pragma inline	heapdef.h resetprg.c	inline 関数を宣言します。	本 pragma で宣言された関数に対してインライン展開します。
#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。
#pragma section	heap.c resetprg.c initsct.h 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。
#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。

7.3 SFR ヘッドファイル

High-performance Embedded Workshop が出力する各マイコン用の SFR ヘッドファイルの C ソースコードは、MISRA C ルールに対して 5 つ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がありません。

7.3.1 ルール違反の要因

High-performance Embedded Workshop が出力する各マイコン用の SFR ヘッドファイルの 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。

- (1) C コンパイラの仕様 (#pragma)
- (2) typedef を使用した宣言
- (3) bitfield のメンバ宣言

7.3.2 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通りです。

13 14 99 110 111

7.4 評価環境

コンパイラ	R32C/100 シリーズ用 C コンパイラパッケージ V.1.01 Release 00
コンパイルオプション	-O -c -as100 "-DOPTI=0" -gnone -finfo -fNII -misra_all -misra_report \$*.csv
MISRA C チェッカ	SQMLint V.1.03 Release 00

8. C 言語スタートアッププログラムについて

8.1 C 言語スタートアッププログラムのファイル構成

C 言語スタートアップは次の 13 個の C 言語ファイルで構成しています。

- (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
OnChipDedebgger 使用時の NSD の firm が使用するワークスペース領域をダミーとして確保します。
- (7) resetprg.h
C 言語スタートアップで使用する各ヘッダファイルをインクルードしています。
- (8) initsct.h
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。
このファイルの内容は変更しないでください。
- (9) heapdef.h
ヒープ領域を初期化します。
- (10) cregdef.h
マイコン内部レジスタを宣言しています。
このファイルの内容は変更しないでください。
- (11) stackdef.h
スタックサイズを定義しています。
- (12) vector.h
可変ベクタのアドレスを定義しています。
- (13) typedef.h
データ型を typedef 宣言しています。

8.2 C 言語スタートアッププログラムの処理

8.2.1 resetprg.c

マイコンの初期設定を行います。

このファイルは、C 言語スタートアップで必須のファイルです。


```

#include "resetprg.h"
////////////////////////////////////
// declare sfr register
DEF_SBREGISTER;

#pragma entry start
void start(void);
extern void initsct(void);
extern void _init(void);
void exit(void);
void main(void);

#pragma section program interrupt → (1)
#pragma inline set_cpu()
void set_cpu(void) → (2)
{
    _jsp_ = &(unsigned long)_istack_top; // set interrupt stack pointer → (3)
    _flg_ = 0x0080; // set flag register → (4)
    _sp_ = &(unsigned long)_stack_top; // set user stack pointer → (5)
    _sb_ = (unsigned long *)0x400; // 400H fixation (Do not change) → (6)
    _asm(" fset b");
    _sb_ = (unsigned long *)0x400;
    _asm(" fclr b");
    _intb_ = (unsigned long *)VECTOR_ADR; // set variable vector's address → (7)
}

void start(void)
{
    set_cpu(); // initialize mcu → (8)
    initsct(); // initialize each sections → (9)
#ifdef __HEAP__
    heap_init(); // initialize heap → (10)
#endif
#ifdef __STANDARD_IO__
    _init(); // initialize standard I/O → (11)
#endif
    _fb_ = 0; // initialize FB register for debugger
    main(); // call main routine → (12)

    exit(); // infinite loop
}

void exit(void)
{
    while(1);
}

```

- (1) マイコンリセット後に実行されるスタート関数 start()は interrupt セクションに配置します。
- (2) マイコン初期化関数 set_cpu()本体を定義します。
- (3) 割り込みスタックポインタを初期化します。
- (4) U フラグを 1 に設定(スタックポインタをユーザスタックに設定)します。
- (5) ユーザスタックポインタを初期化します
- (6) 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。
- (7) 可変ベクタアドレスを INTB レジスタに設定します。
- (8) マイコン初期化関数を呼び出します。
- (9) 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。
- (10) 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。
- (11) 標準入出力関数の初期化を行います。標準入出力関数を使用する場合は、この関数の呼び出しを有効にする必要があります。

(12) main 関数を呼び出します。

8.2.2 resetprg.h

C 言語スタートアップで使用する各ヘッダファイルをインクルードします。

このファイルは、C 言語スタートアップで必須のファイルです。

8.2.3 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。

このファイルは、C 言語スタートアップで必須のファイルです。

```
#include "initsct.h"
void initsct(void);

void initsct(void)
{
    sclear("bss_SB8", "data,align");
    sclear("bss_NEAR", "data,align");           → (1)
    sclear("bss_FAR", "data,align");
    sclear("bss_EXT", "data,align");

    /* clear bss for NSD */
    sclear("bss_MON1", "data,align");
    sclear("bss_MON2", "data,align");
    sclear("bss_MON3", "data,align");
    sclear("bss_MON4", "data,align");

    // when add new sections
    // bss_clear("new section's name");

    scopy("data_SB8", "data,align");
    scopy("data_NEAR", "data,align");           → (2)
    scopy("data_FAR", "data,align");
    scopy("data_EXT", "data,align");

    /* copy data section for NSD */
    scopy("data_MON1", "data,align");
    scopy("data_MON2", "data,align");
    scopy("data_MON3", "data,align");
    scopy("data_MON4", "data,align");
}
```

(1) sclear: bss セクションをゼロクリアします。

#pragma 拡張機能 #pragma SECTION を用いて bss セクションの名称を変更した場合は、変更後のセクションを追加する必要があります。

```
sclear( "セクション名_NEAR", "data,align" );
```

例えば、#pragma SECTION bss bss2 で bss2 セクションを追加した場合は、次の行を initsct.c ファイルに追記します。

```
sclear( "bss2_NEAR", "data,align" );
```

(2) scopy: data セクションに対して初期値を転送します。

#pragma 拡張機能 #pragma SECTION を用いて data セクション名の名称を変更した場合は、変更後のセクションを追加する必要があります。

```
scopy( "セクション名_NEAR", "data,align" );
```

例えば、#pragma SECTION data data2 で data2 セクションを追加した場合は、次の行を initsct.c ファイルに追記します。

```
scopy("data2_NEAR", "data.align");
```

補足事項：

使用していないセクションの初期化 (sclear、scopy) をコメントアウトすることで ROM サイズの節約およびスタートアップ処理を高速化することができます。

- (A) #pragma SBDATA を使用しない場合
セクションベースの属性が "SB8" の初期化を削除します。
- (B) #pragma EXTMEM を使用しない場合
セクションベースの属性が "EXT" の初期化を削除します。
- (C) #pragma MONITOR[n] を使用しない場合
セクションベースの属性が "MON[n]" の初期化を削除します。

8.2.4 initsct.h

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。

このファイルは、C 言語スタートアップで必須のファイルです。

ファイルの内容は変更しないでください。

8.2.5 heap.c

ヒープ領域を確保します。

このファイルは、malloc 関数などのメモリ管理関数を使用する場合に必要です。

```
#include "typedefine.h"
#include "heapdef.h"
#pragma SECTION bss heap → (1)
_UBYTE heap_area[_HEAPSIZE_]; → (2)
```

(1) heap 領域を heap_NEAR セクションに配置します。

(2) ヒープ領域を _HEAPSIZE_ で定義されたサイズ分確保します。

8.2.6 heapdef.h

ヒープ領域を初期化します。

このファイルは、malloc 関数などのメモリ管理関数を使用する場合に必要です。

```

extern  _UBYTE _far * _mnext;
extern  _UDWORD _msize;
//////////
// It's size of heap
// When you want to change size of heap,
// please change this line.
// When you change this line,
// you must modify the value using hex character.

#ifndef __HEAPSIZE__
#define __HEAPSIZE__ 0x300
#endif
extern _UBYTE heap_area[__HEAPSIZE__];

#pragma inline heap_init()
void heap_init(void)
{
    _mnext = &heap_area[0];           → (1)
    _msize = __HEAPSIZE__;           → (2)
}

```

- (1) ヒープ管理領域を初期化します。
- (2) ヒープサイズを初期化します。

8.2.7 fvector.c

固定ベクタテーブルの定義を行います。

このファイルは、C言語スタートアップで必須のファイルです。

```

#include "vector.h"
#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 0xffffffff
#pragma interrupt/v 0xffffffff
#pragma interrupt/v _dummy_int      //wdt
#pragma interrupt/v _dummy_int
#pragma interrupt/v _dummy_int      //nmi
#pragma interrupt/v start           → (3)

#pragma interrupt _dummy_int()
void _dummy_int(void){}

```

- (1) 固定ベクタテーブルのセクションとアドレスを設定します。
この#pragma 拡張機能はC言語スタートアップ専用です。
- (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。
この#pragma 拡張機能はC言語スタートアップ専用です。
- (3) リセット関数を定義します。
マイコンリセット時に実行する関数を固定ベクタに登録します。

8.2.8 intrpg.c

可変ベクタ割り込みのエントリ関数を宣言します。

このファイルの内容は、ご使用のマイコンにより異なります。

```

// BRK (software int 0)
#pragma interrupt _brk(vect=0)
void _brk(void){}

// vector 1 reserved

// uart5 trance/NACK (software int 2) → (1)
#pragma interrupt _uart5_trance(vect=2)
void _uart5_trance(void){}

// uart5 receive/ACK (software int 3)
#pragma interrupt _uart5_receive(vect=3)
void _uart5_receive(void){}

// uart6 trance/NACK (software int 4)
#pragma interrupt _uart6_trance(vect=4)
void _uart6_trance(void){}

// uart6 receive/ACK (software int 5)
#pragma interrupt _uart6_receive(vect=5)
void _uart6_receive(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。ここで宣言された関数は、リンク時に可変ベクタテーブルに反映されます。

8.2.9 firm.c

OnChipDedebgger 使用時の NSD の firm が使用するワークスペース領域をダミーとして確保します。
このファイルは、OnChipDebugger を使用する場合に使用します。

```

#include "typedefine.h"
#pragma section bss FirmRam → (1)
__UBYTE__ workram[0x8]; // for Firmware's workram → (2)

```

(1) NSD のファームウェアが使用する work ram 領域を FirmRam_NEAR セクションに確保します。

(2) Work ram 領域を __WORK_RAM__ で定義されたサイズ分確保します。

8.2.10 cregdef.h

マイコン内部レジスタを宣言しています。

このファイルは、C 言語スタートアップで必須のファイルです。
ファイルの内容は変更しないでください。

8.2.11 stackdef.h

スタックサイズを定義しています。

このファイルは、C 言語スタートアップで必須のファイルです。

```

#ifndef __STACKSIZE__
#pragma STACKSIZE 0x300          → (1)
#else
#pragma STACKSIZE __STACKSIZE__ → (2)
#endif
#ifndef ISTACKSIZE
#pragma ISTACKSIZE 0x300        → (3)
#else
#pragma ISTACKSIZE __ISTACKSIZE__ → (4)
#endif
extern _UINT _stack_top, _istack_top;

```

- (1) リンク時にスタックサイズを指定していない場合に使用するユーザスタックサイズです。この#pragma 拡張機能により、ユーザスタックのセクション設定とスタックの領域を確保します。
- (2) リンク時にスタックサイズを指定している場合に使用するユーザスタックサイズです。この#pragma 拡張機能により、ユーザスタックのセクション設定とスタックの領域を確保します。
この#pragma 拡張機能はC言語スタートアップ専用です。
- (3) リンク時にスタックサイズを指定していない場合に使用する割り込みスタックサイズです。この#pragma 拡張機能により、割り込みスタックのセクション設定とスタックの領域を確保します。
- (4) リンク時にスタックサイズを指定している場合に使用する割り込みスタックサイズです。この#pragma 拡張機能により、割り込みスタックのセクション設定とスタックの領域を確保します。
この#pragma 拡張機能はC言語スタートアップ専用です。

8.2.12 vector.h

可変ベクタのアドレスを定義しています。

このファイルは、C言語スタートアップで必須のファイルです。

```

#define Fvectaddr 0xffffdc          → (1)
#ifndef VECTOR_ADR
#define VECTOR_ADR 0x0ffffbdc     → (2)
#endif

```

- (1) 固定ベクタテーブルの先頭アドレスを設定します。
- (2) 可変ベクタテーブルの先頭アドレスを設定します。
可変ベクタテーブルの先頭アドレスを変更する場合は、resetprg.c ファイルの INTB レジスタのアドレス設定も変更してください。

8.2.13 typedefine.h

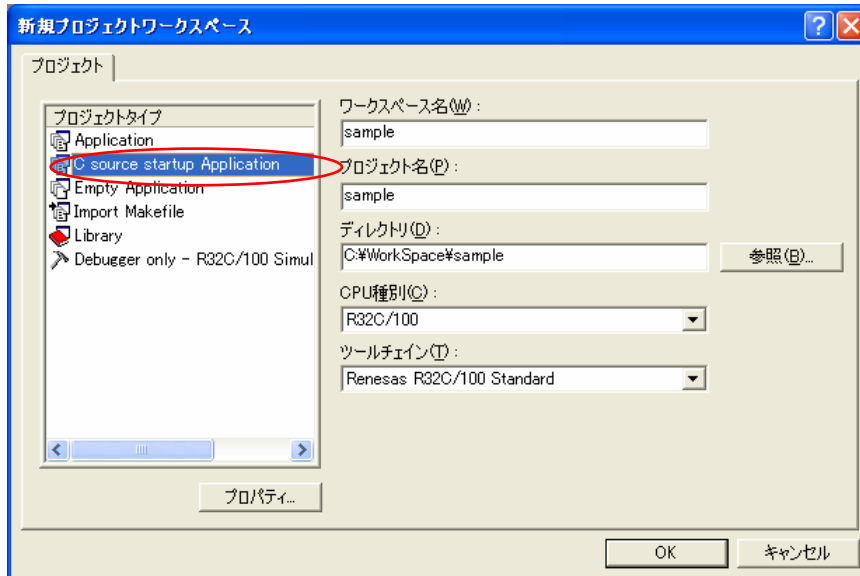
データ型を typedef 宣言しています。

このファイルは、C言語スタートアップで必須のファイルです。

ファイルの内容は変更しないでください。

8.3 High-performance Embedded Workshop で C 言語スタートアッププログラムを使用する場合

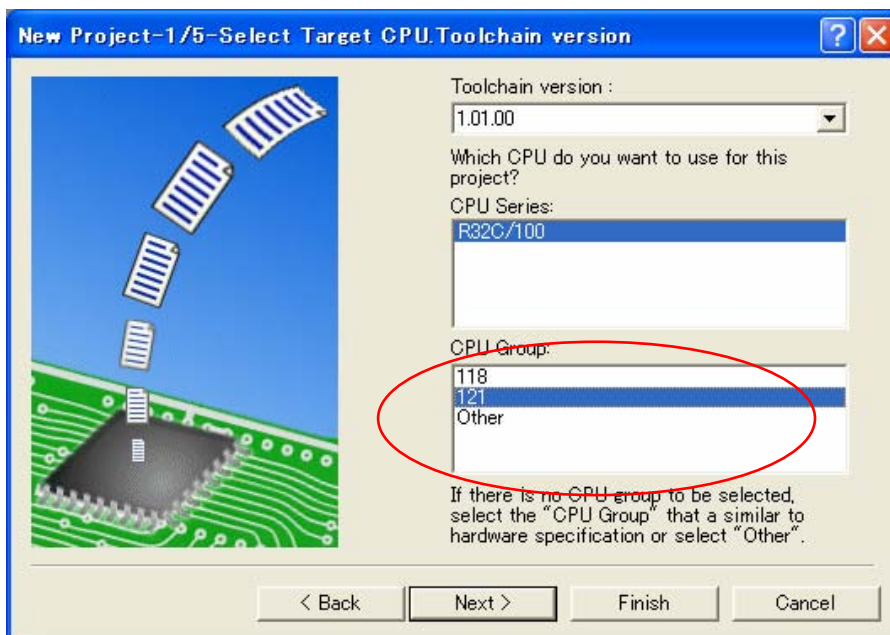
- (1) 「新規プロジェクトワークスペース」で「C source startup Application」を選択し、ワークスペースを作成します。



複数のコンパイラをインストールしている場合、「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」に移動して C ソーススタートアップの選択が無効になります。

この場合は、再度「C source startup Application」を選択してください。

- (2) マイコン品種を「CPU Series」と「CPU Group」から選択します。



この選択により、対応する sfr ヘッドファイルがワークスペースへコピーされます。また、可変ベクタテーブル (intprg.c) が登録されます

- (3) 標準関数ライブラリとメモリ管理関数ライブラリを使用する場合は、「Use Standard I/O Library(UART1)」および「Use Heap Memory」を選択します。「OnChip Debugging Emulator」を使用する場合は、「New

Project-2/5-Setting the Contents of File to be Generated」を選択します。



- (A) 標準関数ライブラリを使用する場合
チェックすることにより、resetprg.c 中の _init() 呼び出しが有効になります。
また、ファイル device.c と init.c がプロジェクトに登録されます。
- (B) メモリ管理関数を使用する場合に、チェックします。
チェックすることにより、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。
- (C) OnChip Debugging Emulator を使用する場合
選択可能なデバuggは、「NSD」です。
この選択により、firm.c が登録されます。
標準入出力関数ライブラリを選択した状態で「OnChip Debugging Emulator」を選択した場合、(UART1) の表示が削除されます。これは、標準入出力関数および OnChip Debugging Emulator 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。

- (4) スタックサイズを設定します。

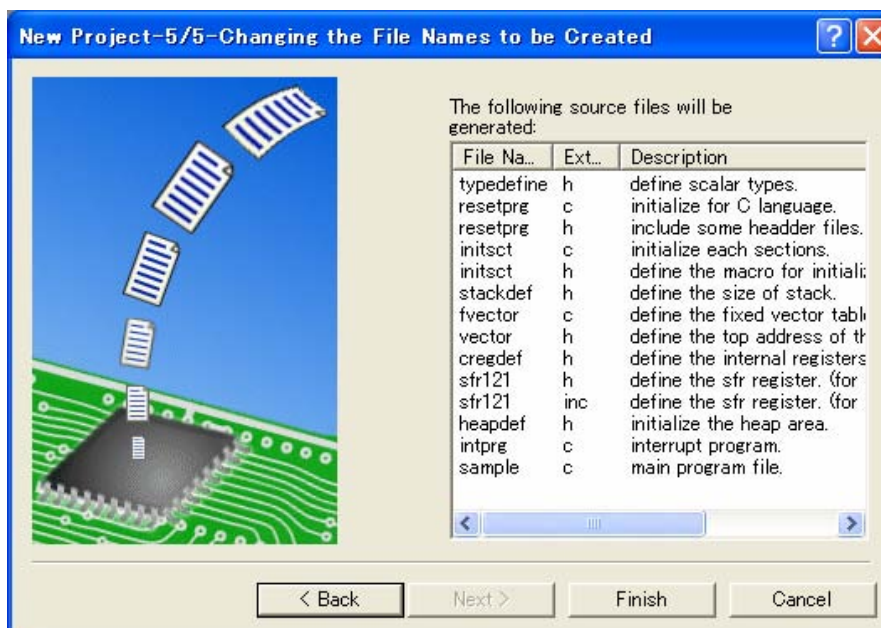


- (A) ユーザスタックサイズの設定
stackdef.h が登録されます
- (B) 割り込みスタックサイズの設定
stackdef.h が登録されます

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、コンパイルオプションで以下の項目を変更してください。

```
-D__STACKSIZE__=xxxx
-D__ISTACKSIZE__=xxxx
-D__HEAPSIZE__=xxxx
```

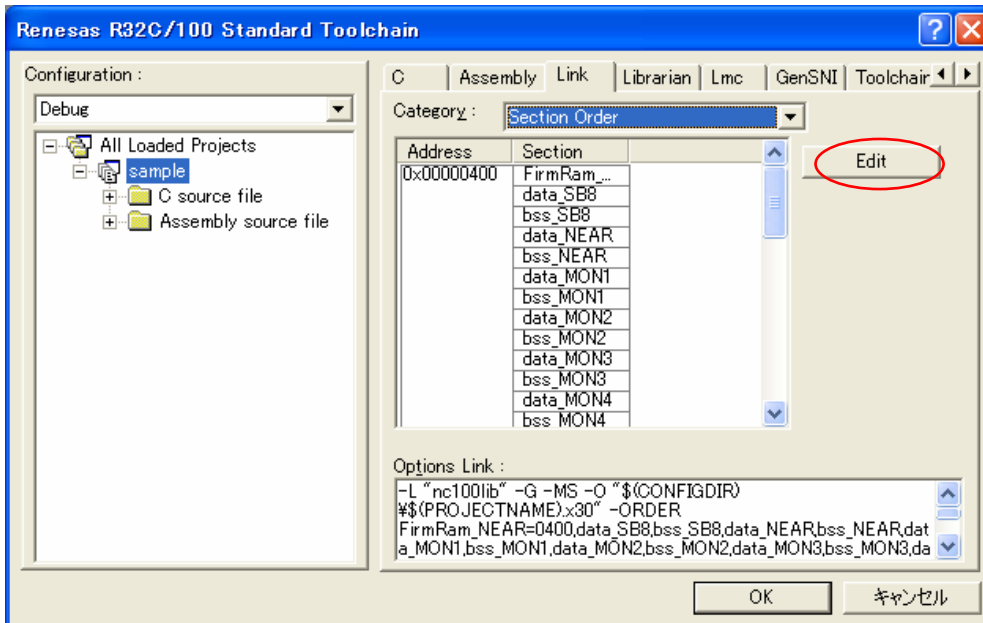
- (5) 登録ファイルを確認します。



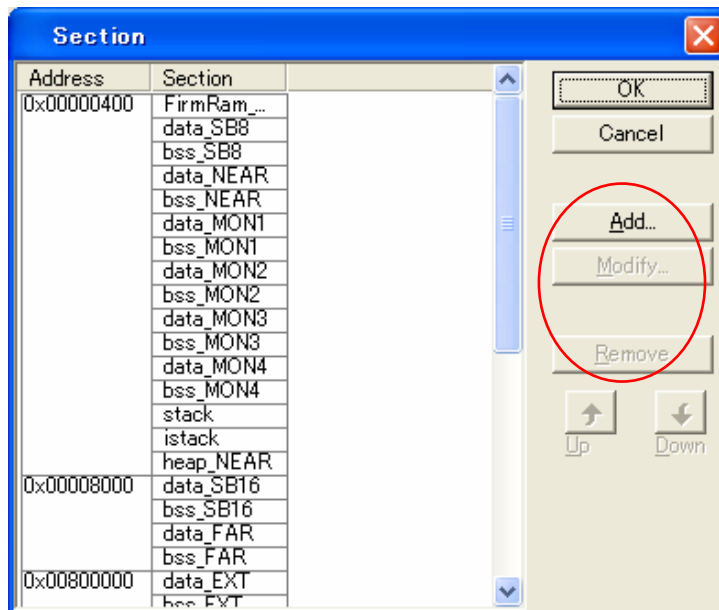
ここで、登録されるファイルを一覧で確認できます。

(6) セクションオーダー

「Renesas R32C/100 Standard Toolchain」 「Link」の「Category」で各セクションの配置およびアドレスを確認することができます。



#pragma SECTION により新規にセクションを追加した等の場合は、「Edit」を選択して、「Section Window」をオープンし、セクションの配置等を変更します。



8.4 High-performance Embedded Workshop でアセンブリ言語スタートアッププログラムを使用する場合

「新規プロジェクトワークスペース」で「Application」を選択し、ワークスペースを作成します。

