===== 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 01

リリースノート

(第 2 版)

誤記に関するお詫び:
本資料のP.10「6.ソフトウェアのバージョン一覧」に誤記があり、訂正いたしました。

株式会社ルネサス ソリューションズ

2010 年 6 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 01 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

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/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、Windows[1] 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 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。

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。

[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。

3.2 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。

3.3 C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

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、リンケージエディタ 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つづ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。

3.3.8 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  */
```

4. V.1.02 Release 00 からのリビジョンアップ内容

4.1 不具合修正

- 配列を連続してアクセス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/100116/tn1.htm
- リンク時のエラー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/091116/tn3.htm
- 使用スタックサイズを計算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/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改修しました。
  該当ツールニュース
  http://tool-support.renesas.com/jpn/toolnews/091001/tn4.htm
- 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改修しました。
  該当ツールニュース
  http://tool-support.renesas.com/jpn/toolnews/090801/tn1.htm

5. プログラムの起動または終了

5.1 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。

5.2 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 Reader[2]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。

5.3 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

  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¥v102r01¥bin¥MapViewer.exe （コンパイラ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］メニューに名前で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。

[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。

5.4 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 01」→「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、アブ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。

5.5 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。

5.5.1 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。

5.5.2 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 ﾂｰﾙﾁｪｲﾝ用 環境変数 *****
SET BIN100=C:¥Program Files¥Renesas¥Hew¥Tools¥Renesas¥nc100¥v102r01¥BIN
SET BIN100=C:¥Program Files¥Renesas¥Hew¥Tools¥Renesas¥nc100¥v102r01¥LIB100
SET BIN100=C:¥Program Files¥Renesas¥Hew¥Tools¥Renesas¥nc100¥v102r01¥INC100
SET BIN100=C:¥Program Files¥Renesas¥Hew¥Tools¥Renesas¥nc100¥v102r01¥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 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。

・nc100	V.2.00.06.002	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.1.02.02.001	コンパイラ本体
・aopt100	V.1.01.02.001	アセンブラオプティマイザ
~~・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.1.02.00.001	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.1.00.01.001	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

・as100 V.1.00.04.001 アセンブラドライバ

7. MISRA C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、MISRA C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] MISRA C 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ラ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ラ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ヘッダファイル

High-performance Embedded Workshop が出力する各マイコン用の SFR ヘッダファイルの 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 ヘッダファイルの 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
OnChipDedebugger 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)
{
        _isp_   = &(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();              // initlalize 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 registe 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 intprg.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

OnChipDedebugger 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          0xffffffdc          → (1)
#ifndef VECTOR_ADR
#define VECTOR_ADR         0x0fffffbdc         → (2)
#endif
```

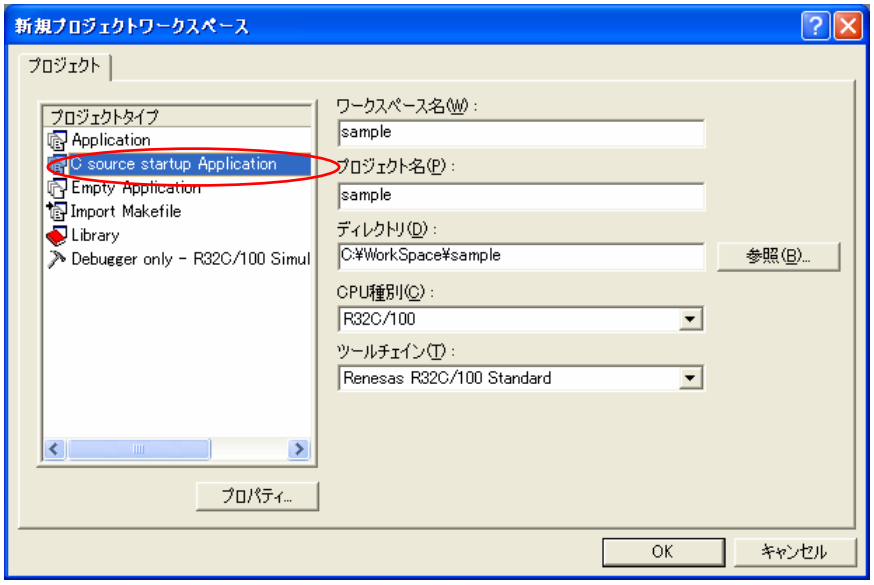
(1) 固定ベクタテーブルの先頭アドレスを設定します。
(2) 可変ベクタテーブルの先頭アドレスを設定します。
可変ベクタテーブルの先頭アドレスを変更する場合は、resetprg.c ファイルの INTB レジスタのアドレス設定も変更してください。

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。

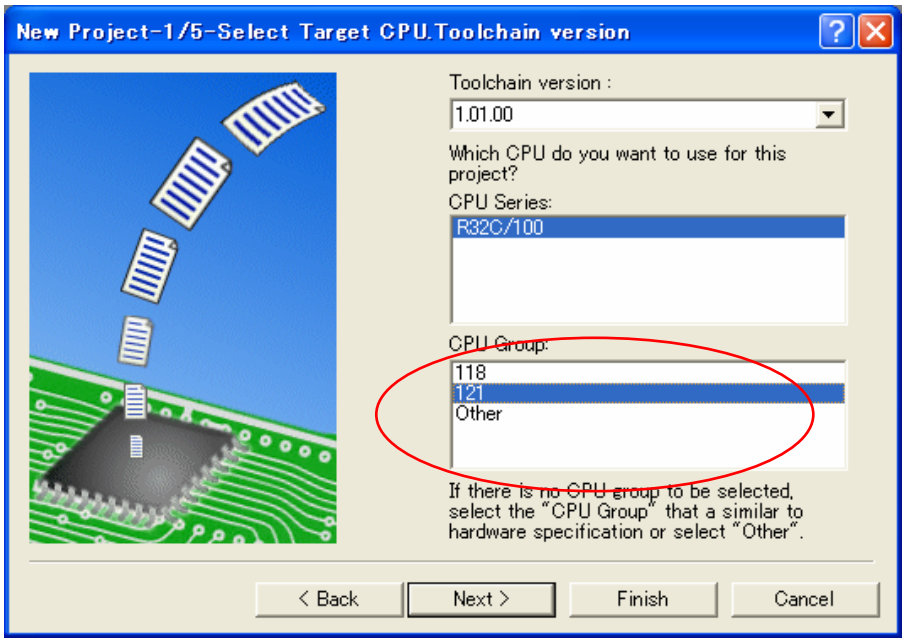
8.3 High-performance Embedded WorkshopでC言語スタートアッププログラムを使用する場合

(1) 「新規プロジェクトワークスペース」で「Csource 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 ヘッダ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) 標準入出力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) 標準入出力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
選択可能なデバッガは、「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。

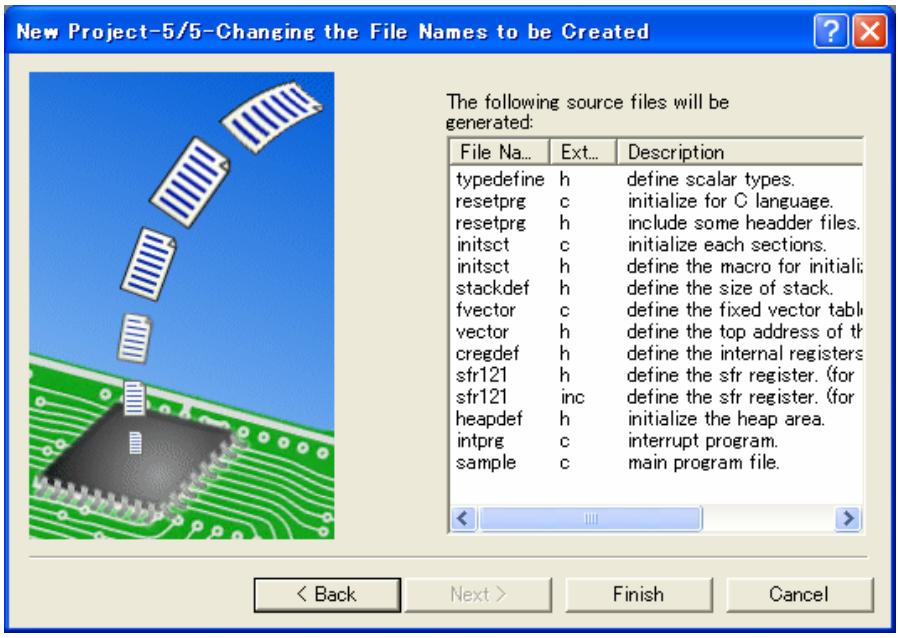
(4)　スタックサイズを設定します。

(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
```

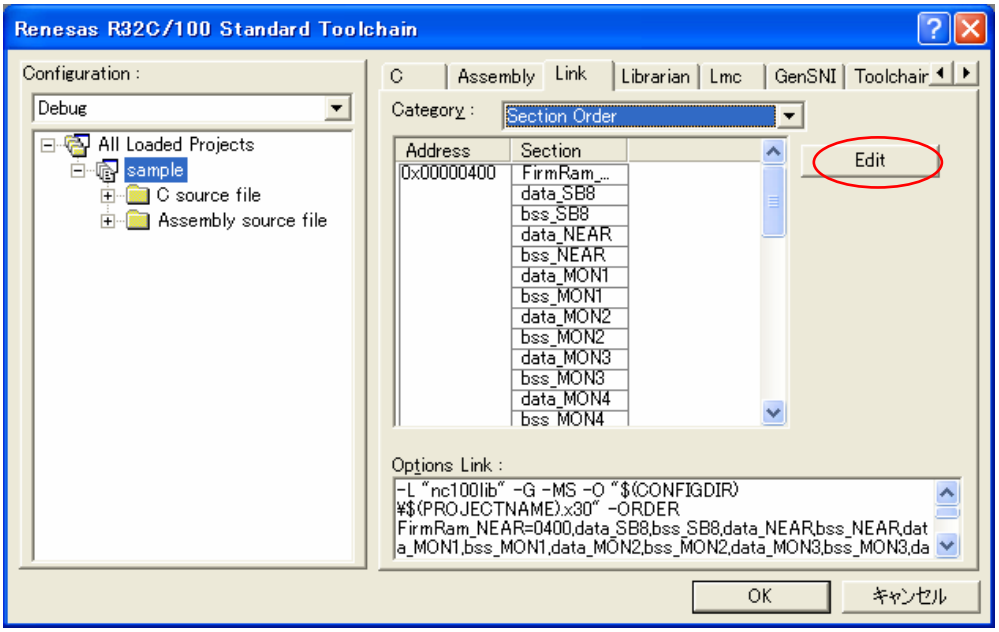
(5)　登録ファイルを確認します。



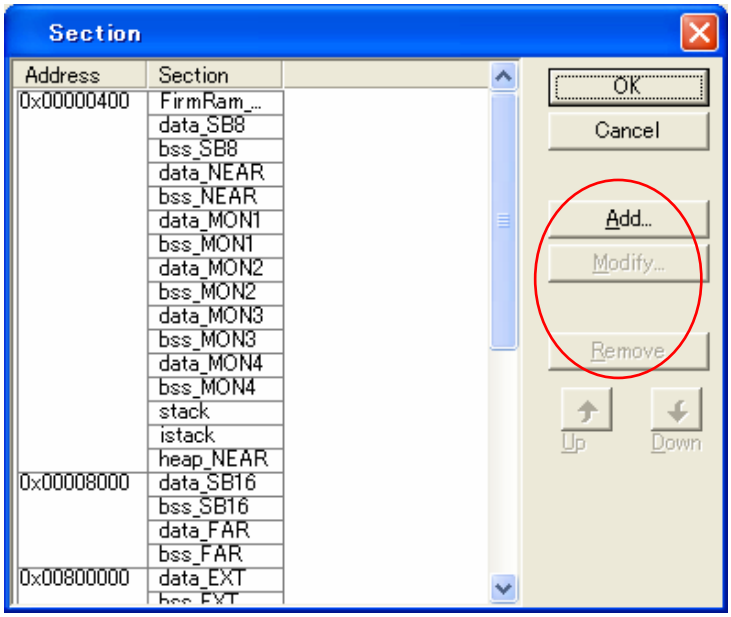
ここで、登録されるファイルを一覧で確認できます。

(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。