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ものですが、誤りがな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て、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が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 =====

M32C/90, M32C/80, M16C/80, M16C/70 シリーズ用 C コンパイラパッケージ
V.5.40 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

2006 年 2 月 16 日

概要

この度は M32C/90, M32C/80, M16C/80, M16C/70 シリーズ用 C コンパイラパッケージ V.5.40 Release 00 をご採用いただき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. 注意事項

製品をご使用になる際は、以下の内容に従ってくださるようお願いいたします。

1.1. C コンパイラに関する注意事項

1.1.1. inline 関数のネストに関する注意事項

- 内容
  仮引数を持つ inline 関数をネストすると、誤った実引数(実引数と異なる変数)を参照する場合があります。
- 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) inline 関数をネストしている。
  (2) 呼び出し元 inline 関数 A と呼び出し先 inline 関数 B の仮引数名が同一である。
- 発生例

```
inline B(int aaa, char ccc)                    /* 発生条件 (2) */
 {
          .....
 }
 inline A(int c, int aaa, char *ccc)           /* 発生条件(2) */
 {
 int   i;
 char c;
        B(i,c);                                /* 発生条件(1) */
 }
```

- 回避策
  次のいずれかの方法で回避してください。
  (1) 呼び出し先の関数(発生例では、inline 関数 B)の引数名を変更する。
  (2) inline 関数のネストをしない。
  (3) コンパイルオプション-Oforward_function_to_inline(-OFFTI)を使用してコンパイルする。

1.1.2. コマンドオプション-I に関する注意事項

コマンドオプション"-I"で指定できるディレクトリの個数は最大 50 個です。

1.1.3. 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。

1.1.4. 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

- #pragam 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。
- 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。
- 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。

1.1.5. _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 "-gBTC(-gbool_to_char)" をご使用ください。

1.1.6. 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。

1.2. 機種依存に関する注意事項

1.2.1. M16C 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、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   006ch                  // M16C/80 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   WaitUntillRisON( void )
{
            while( TA0IC.IR == 0 )               // 1 になるまで待つ
            {
                        ;
            }                                    // 1 になったら 0 に戻す
}
```

- 回避策

  (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止してください。

  (2) 以下のように "asm 関数" を挿入することにより、最適化を抑止してください。

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

- 注意

  コンパイルオプション "-Ono_asmopt[-ONA]" 、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。

1.2.2. 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、詳しくは各機種のユーザーズ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。

1.3. MS-Windows に関する注意事項

1.3.1. 動作環境に関する注意事項

(1) C コンパイラパッケージは、Windows 98、Windows NT 4.0 以降の環境で動作します。Windows 3.1 および Windows NT 3.5x 以前のバージョンでは動作しません。

(2) 日本語 Windows NT 環境でコマンドプロンプトのサイズが「80 x 25」以外に設定されている場合、製品を起動するとコマンドプロンプトのサイズが頻繁に切り替わります。コマンドプロンプトのサイズは「80 x 25」に設定してください。

1.3.2. 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は、次の注意事項に従ってください。

- 漢字を含むディレクトリ名、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。
- パス指定を含めたファイル名の長さが 128 文字以上になるものは使用できません。

1.3.3. 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。その場合は、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。

1.3.4. バージョンアップするときの注意事項

C コンパイラパッケージをバージョンアップする場合は、あらかじめ、インストールされている C コンパイラパッケージをアンインストールしてから、新しいバージョンをインストールしてください。

- C コンパイラパッケージのアンインストール手順

  C コンパイラパッケージをアンインストールするには、「コントロールパネル」- 「アプリケーションの追加と削除」を選択しアンインストールを実行してください。

2. バージョンアップ内容

2.1. C コンパイラ機能追加・改定

- M32C/90 シリーズに対応したコードを生成するためのコンパイルオプション "-M90" を追加しました。
- ROM 圧縮を強化するコンパイルオプション"-OR_MAX"と実行サイクル数を縮小するコンパイルオプション "-OS_MAX"を追加しました。
- 指定された外部変数を NSD 仕様エミュレータの RAM モニタ領域専用のセクションに配置することを宣言する #pragma MONITOR[n]を追加しました。
- 空白文字を含むディレクトリ名を使用できるようにしました。
- アセンブリ言語スタートアッププログラム(ncrt0.a30、sect308.inc)に加えて、C 言語スタートアップをサポートしました。
- 最適化を強化しました。
  (1) 共通式を関数呼び出しとして纏め上げる。
  (2) 最適化オプション選択時にループ回数が 1 回のループ式を展開する。

2.2. C コンパイラ問題点修正

ツールニュースでお知らせの以下の問題点を改修しました。

- if-else 文に関する注意事項
  http://tool-support.renesas.com/jpn/toolnews/n050401/tn13.htm
- 関数の実引数にポインタまたはアドレスを渡す場合の注意事項
  http://tool-support.renesas.com/jpn/toolnews/n050601/tn8.htm
- const 修飾された配列型の参照に関する注意事項
  http://tool-support.renesas.com/jpn/toolnews/n050601/tn9.htm
- 空関数が複数存在する場合の注意事項
  http://tool-support.renesas.com/jpn/toolnews/n050716/tn9.htm
- 最適化オプション"-OR"使用時の注意事項
  http://tool-support.renesas.com/jpn/toolnews/n050716/tn10.htm
- リアルタイム OS M3T-MR308 を使用したプログラムをコンパイルする場合
  http://tool-support.renesas.com/jpn/toolnews/n051001/tn6.htm
- do 文に関する注意事項
  http://tool-support.renesas.com/jpn/toolnews/n051116/tn7.htm
- 減算に関する注意事項
  http://tool-support.renesas.com/jpn/toolnews/n051116/tn8.htm

2.3. アセンブラシステム機能追加・改定

2.3.1. as308

- 空白文字を含むディレクトリ名を使用できるようにしました。
- C言語スタートアップに対応しました。本改定に伴い、以下の変更を行いました。
  (1) アセンブラオプション "-fMVT"、"-fMST"を削除
  (2) 指示命令 .INITSCT を追加
  (3) 指示命令 .RVECTOR、.SVECTOR の機能拡張
- M32C/90 シリーズに対応したコードを生成するためのアセンブラオプション "-M90" を追加しました。

2.3.2. ln308

- 空白文字を含むディレクトリ名を使用できるようにしました。
- C言語スタートアップに対応しました。本改定に伴い、以下の変更を行いました。
  (1) リンクオプション "-fMVT"、"-fMST"を削除

(2) リンクオプション "-ORDER" の機能拡張
(3) リンクオプション "-VECT" の機能拡張
(4) リンクオプション "-VECTN" を追加

- M32C/90 シリーズに対応したコード生成を行うリンクオプション "-M90" を追加しました。

2.3.3. lb308

空白文字を含むディレクトリ名を使用できるようにしました。

2.3.4. xrf308

空白文字を含むディレクトリ名を使用できるようにしました。

2.4. アセンブラシステム問題点修正

2.4.1. as308

ツールニュースでお知らせの以下の問題点を改修しました。

- アセンブラオプション"-mode60"および"-mode60p"に関する注意事項
  http://tool-support.renesas.com/jpn/toolnews/n050301/tn12.htm

2.4.2. ln308

ツールニュースでお知らせの以下の問題点を改修しました。

- リンクオプション"-JOPT"に関する注意事項
  http://tool-support.renesas.com/jpn/toolnews/n051201/tn7.htm

2.4.3. lmc308

ツールニュースでお知らせの以下の問題点を改修しました。

- ロードモジュールコンバータ(lmc308 および lmc30)オプション"-F"を使用して生成される機械語ファイルについて
  http://tool-support.renesas.com/jpn/toolnews/n050816/tn2.htm

3. 製品のインストール

3.1. 必要なシステム構成

ホストコンピュータ	IBM[1] PC互換機
OS	Windows[2] 98、Windows Me、Windows NT 4.0、Windows 2000 またはWindows XP
メモリ容量	最低 128MB、256MB 以上を推奨
ハードディスク容量	空き容量 150MB 以上
ディスプレイ	SVGA 以上の解像度
I/O 装置	CD-ROM ドライブ
その他	マウス等のポインティングデバイス

3.2. インストール方法

C コンパイラパッケージは、以下の手順でインストールしてください。
なお、インストールに際しては、全てのアプリケーションの実行を終了させてください。

3.2.1. インストール手順

(1) 「Renesas C Compiler Packeage for M16C family」の CD-ROM を CD-ROM ドライブに挿入します。(以下、仮に D ドライブとします)
(2) Windows スタートメニューの[ファイル名を指定して実行...]をクリックします。
(3) CD-ROM のルートディレクトリにある Autorun.exe を[ファイル名を指定して実行]ダイアログボックスで指定し(例 D:¥ Autorun.exe)、[OK]をクリックします。
(4) 画面に表示されるインストールの指示に従います。
(5) C コンパイラパッケージのインストールが完了すると AutoUpdate 設定画面が表示されるので、初期設定を行ってください。詳細は「オートアップデートツール説明添付資料」を参照してください。

3.2.2. インストールの注意事項

(1) インストールの途中でライセンス ID を入力する必要があります。インストールを始める前にライセンス ID を確認してください。
(2) 全角文字のないディレクトリパスにインストールしてください。
(3) Hitachi Integration Manager または Hitachi Embedded Workshop と同じディレクトリにインストールしないでください。
(4) インストールした直後に、「Renesas」が Windows スタートメニューの「プログラム」の中に表示されない場合は、Windows を再起動してください。
(5) インストール中にインストーラが異常終了した場合、コンピュータを再起動してから再度インストールしてください。
(6) High-performance Embedded Workshop V.4.00.03 は、High-performance Embedded Workshop Ver.1.x とは別ディレクトリにインストールしてください。
(7) High-performance Embedded Workshop Ver.1.0x で作成したプロジェクトはそのまま使えません。High-performance Embedded Workshop Ver.1.0x で作成したプロジェクトを使う場合は、High-performance Embedded Workshop 1.1x 用に変換(High-performance Embedded Workshop 1.1x でオープンしてセーブ)してから High-performance Embedded Workshop V.4.00.03 で使ってください。
(8) 本製品の「使用権許諾契約書」、「リリースノート」などをよくお読みください。製品をインストールした場合は、契約書の記載内容に同意されたものとみなします。
(9) 最新の注意事項は以下の URL を参照してください。
http://tool-support.renesas.com/jpn/toolnews/hew

[1] IBMは、International Business Machines Corporationの登録商標です。
[2] Windows®、Windows NT®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。

3.3. プログラムの削除

C コンパイラパッケージの削除は以下の手順にしたがってください。
なお、削除に際しては、全てのアプリケーションの実行を終了させてください。

3.3.1. High-performance Embedded Workshop の削除

(1) Windows スタートメニューの[コントロールパネル]をクリックします。
(2) 「プログラムの追加と削除」アイコンをダブルクリックします。
(3) 「プログラムの変更と削除」タブから「High-performance Embedded Workshop」をクリックし、[削除]ボタンをクリックします。
(4) 画面に表示される指示に従います。

3.3.2. C コンパイラの削除

(1) Windows スタートメニューの[コントロールパネル]をクリックします。
(2) 「プログラムの追加と削除」アイコンをダブルクリックします。
(3) 「プログラムの変更と削除」タブから「ルネサス M32C/90,80, M16C/80,70 シリーズ用 C コンパイラパッケージ V.5.40 Release 00」をクリックし、[削除]ボタンをクリックします。
(4) 画面に表示される指示に従います。

3.4. プログラムの起動または終了

3.4.1. 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。

3.4.2. 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。
- 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。

3.4.3. オンラインマニュアルおよび添付資料の参照

Manual Navigator の Navigator ウィンドウでマニュアルを選んで表示してください。

- 注意
  (1) Manual Navigatorでマニュアルを表示するためには、Adobe Reader[3]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。

3.5. C コンパイラを DOS プロンプトおよびコマンドプロンプト上で使用する場合の設定

C コンパイラの環境変数は、C コンパイラのインストールディレクトリにある setnc308.bat に設定されています。
C コンパイラを DOS プロンプトおよびコマンドプロンプト上で使用する場合は、setnc308.bat を実行してください。

[3] 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。

4. 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.5.40 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。

- nc308 V.1.08.00.000
- cpp308 V.4.08.00.000
- ccom308 V.5.04.00.000
- aopt308 V.1.04.00.000
- as308 V. 4.02.02.000
- mac308 V. 2.22.00.000
- pre30 V.1.10.12
- asp308 V. 4.02.05.000
- ln308 V. 4.03.03.000
- lb308 V. 2.02.01.000
- lmc308 V. 3.00.02.000
- xrf308 VV. 2.02.00.000
- abs308 V. 1.03.00.000
- stk V.1.00.04
- utl308 V.1.00.10
- Stk Viewer V.1.00.01
- MapViewer V.3.00.00

5. 標準関数ライブラリの MISRA C ルール適合に関して

M32C/90, M32C/80, M16C/80, M16C/70 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[4]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。

5.1. 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。

- 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

5.2. 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								

5.3. 評価環境

コンパイラ	M32C/80, M16C/80, M16C/70 シリーズ用 C コンパイラパッケージ V.5.20 Release 1
コンパイルオプション	-O -c -as308 "-DOPTI=0" -gnone -finfo -fNII -misra_all -r $*.csv
MISRA C チェッカ	SQMlint V.1.00 Release 1A

6. M32C/90, /80, M16C/80, /70 シリーズ用リアルタイム OS の対応バージョン

C コンパイラパッケージは、M32C/90, /80, M16C/80, /70 シリーズ用リアルタイム OS V.1.10 Release 1A およびに対応しています。本製品と組合せてリアルタイム OS をご使用になる場合には、上記バージョンをご使用ください。
なお、C コンパイラパッケージとリアルタイム OS は同一のディレクトリにインストールしてください。

[4] MISRA C ルールチェッカSQMLintによる検査結果値です。

7. C 言語スタートアッププログラムについて

C コンパイラパッケージ V.5.40 Release 00 から、C 言語スタートアッププログラムをサポートしました。C 言語スタートアッププログラムは、V.5.20 Release 02 以前のコンパイラでは使用することができません。
なお、C 言語スタートアッププログラムの代わりに、従来のアセンブリ言語スタートアッププログラムも使用できます。

7.1. 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 / firm_ram.c
OnChipDedebugger 使用時の 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) 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。

(8) heapdef.h
ヒープ領域を初期化します。

(9) 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。

(10) 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。

(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。

7.2. C 言語スタートアッププログラムの処理

7.2.1. resetprg.c

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。
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
#pragma ADDRESS   protect    0AH
#pragma ADDRESS   pmode0   04H
#pragma ADDRESS   _SB__      0400H
_UBYTE protect,pmode0;
_UBYTE _SB__;

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

#pragma section program   interrupt                                    → (1)
#pragma inline set_cpu()
void set_cpu(void)                                                     → (2)
{
        _isp_       = &_istack_top;       // set interrupt stack pointer       → (3)
        protect = 0x02;                   // change protect mode register      → (4)
        pmode0   = 0x00;                  // set processor mode register       → (5)
        protect = 0x00;                   // change protect mode register      → (6)
        _flg_       = 0x0080;             // set flag register                 → (7)
        _sp_        = &_stack_top;        // set user stack pointer            → (8)
        _sb_        = (char _far *)0x400; // 400H fixation (Do not change)     → (9)
        _asm("      fset        b");
        _sb_        = (char _far *)0x400;
        _asm("      fclr        b");
        _intb_ = (char _far *)VECTOR_ADR; // set variable vector's address     → (10)

}

void start(void)
{
        set_cpu();                        // initialize mcu                    → (11)
        initsct();                        // initlalize each sections          → (12)
#ifdef __HEAP__
        heap_init();                      // initialize heap                   → (13)
#endif
#ifdef __STANDARD_IO__
        _init();                          // initialize standard I/O           → (14)
#endif
        _fb_ = 0;                         // initialize FB registe for debugger
        main();                           // call main routine                 → (15)

        exit(0);                          // infinite loop
}

void exit(int rc)
{
        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) プロテクトモードレジスタを”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) プロテクトモードレジスタを”書き込み禁止”に設定します。
(7) 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。

(8) 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 レジスタに設定します。
(11) マイコン初期化関数を呼び出します。
(12) 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。
(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。
(15) main 関数を呼び出します。

7.2.2. 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。

7.2.3. initsct.c

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。
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_SE","data,align");                  → (1)
        sclear("bss_SO","data,noalign");
        sclear("bss_NE","data,align");
        sclear("bss_NO","data,noalign");
        sclear("bss_FE","data,align");                  → (2)
        sclear("bss_FO","data,noalign");

        /* clear bss for NSD */
        sclear("bss_MON1_E","data,align");
        sclear("bss_MON2_E","data,align");
        sclear("bss_MON3_E","data,align");
        sclear("bss_MON4_E","data,align");
        sclear("bss_MON1_O","data,noalign");
        sclear("bss_MON2_O","data,noalign");
        sclear("bss_MON3_O","data,noalign");
        sclear("bss_MON4_O","data,noalign");

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

        scopy("data_SE","data,align");                  → (3)
        scopy("data_SO","data,noalign");
        scopy("data_NE","data,align");
        scopy("data_NO","data,noalign");

        /* copy data section for NSD */
        scopy("data_MON1_E","data,align");
        scopy("data_MON2_E","data,align");
        scopy("data_MON3_E","data,align");
        scopy("data_MON4_E","data,align");
        scopy("data_MON1_O","data,noalign");
        scopy("data_MON2_O","data,noalign");
        scopy("data_MON3_O","data,noalign");
        scopy("data_MON4_O","data,noalign");
        scopy("data_FE","data,align");                  → (4)
        scopy("data_FO","data,noalign");
}
```

(1) sclear : near 領域の 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_NE” , “data.align” );
sclear( “セクション名_NO” , “data,noalign” );
```

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_NE” , “data.align” );
sclear( “bss2_NO” , “data,noalign” );
```

(2) sclear_f : far 領域の bss セクションをゼロクリアします。
far 修飾子を用いて初期値の無い外部変数を宣言した場合は、このマクロ関数を有効にする必要があります。

(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 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。

```
sclear( “セクション名_NE” , “data.align” );
sclear( “セクション名_NO” , “data,noalign” );
```

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。

```
sclear( “data2_NE” , “data.align” );
sclear( “data2_NO” , “data,noalign” );
```

(4) scopy_f : far 領域の data セクションに初期値を転送します。
far 修飾子を用いて初期値の無い外部変数を宣言した場合は、このマクロ関数を有効にする必要があります。

7.2.4. initsct.h

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。

7.2.5. heap.c

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_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 セクションに変更します。

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

7.2.6. heapdef.h

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。

7.2.7. fvector.c

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 _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)


#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。

7.2.8. intprg.c

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
// vector 2 reserved
// vector 3 reserved
// vector 4 reserved
// vector 5 reserved
// vector 6 reserved

// A/D1 (software int 7)
#pragma interrupt    _ad1(vect=7)
void _ad1(void){}

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

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

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

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

// TIMER A0 (software int 12)
#pragma interrupt    _timer_a0(vect=12)
void _timer_a0(void){}

// TIMER A1 (software int 13)
#pragma interrupt    _timer_a1(vect=13)
void _timer_a1(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。

7.2.9. firm.c / firm_ram.c

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[0x4];          // for Firmware's workram              → (2)
```

(1) E8 エミュレータ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。
(2) Work ram 領域を__WORK_RAM__で定義されたサイズ分確保します。

7.2.10. cregdef.h

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。

7.2.11. stackdef.h

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。

7.2.12. vector.h

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          0x0fffd00                       → (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。

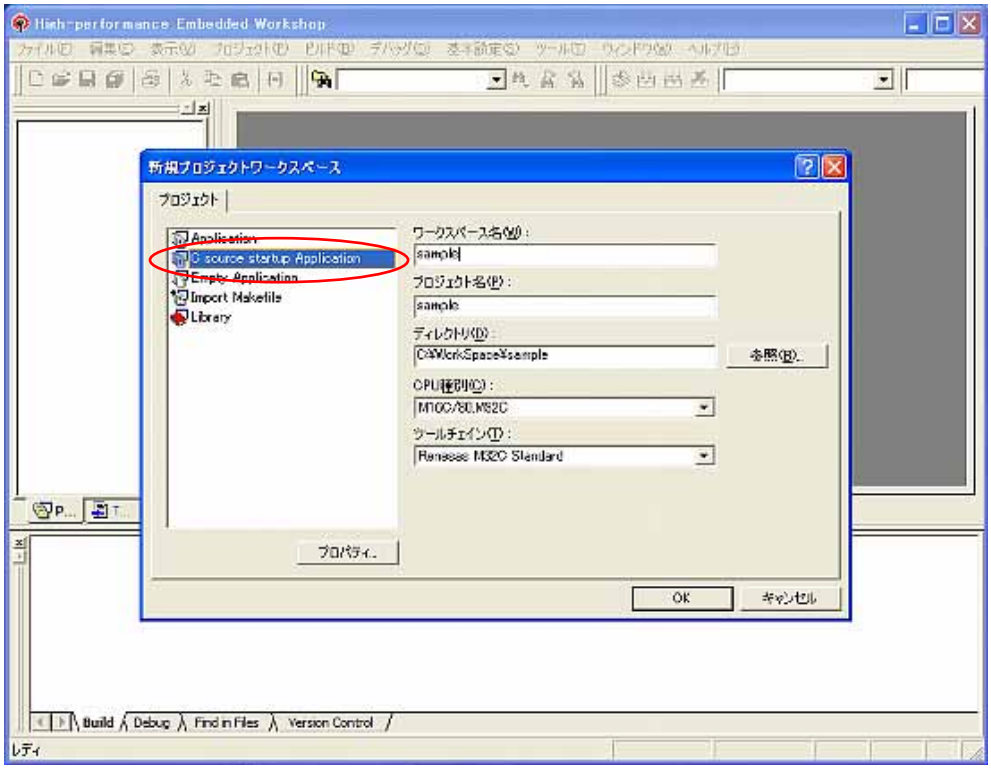
(2) 可変ベクタテーブルの先頭アドレスを設定します。
可変ベクタテーブルの先頭アドレスを変更する場合は、resetprg.c ファイルの INTB レジスタのアドレス設定も変更してください。

7.2.13. typedefine.h

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。

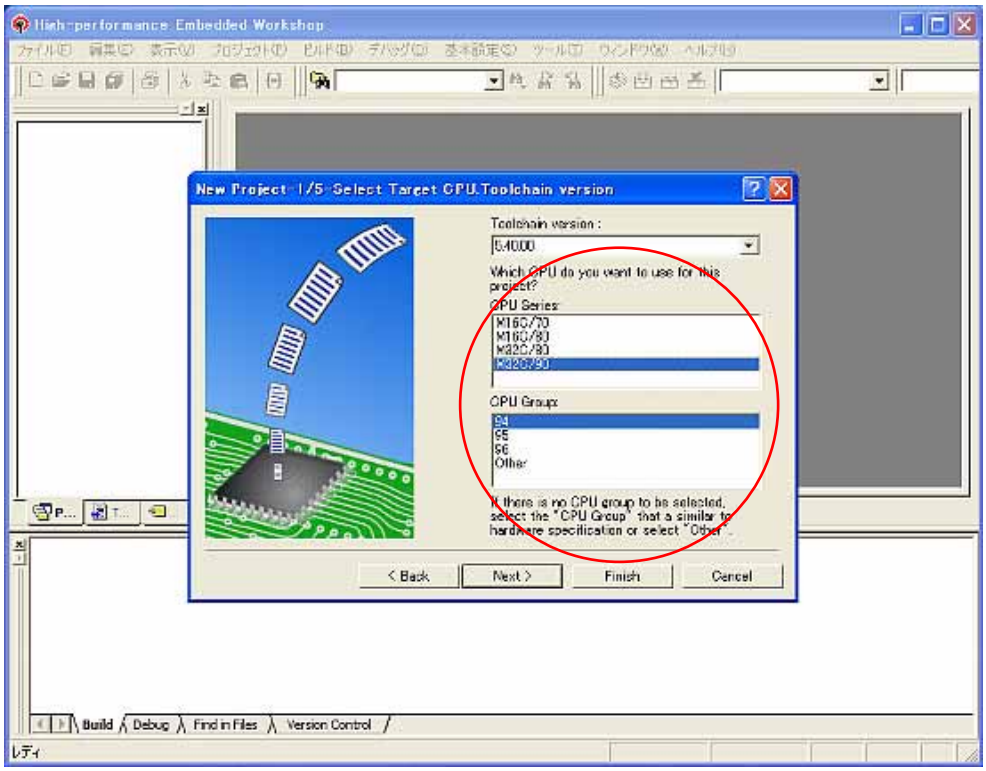
7.3. High-performance Embedded Workshop で C 言語スタートアッププログラムを使用する場合

(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。



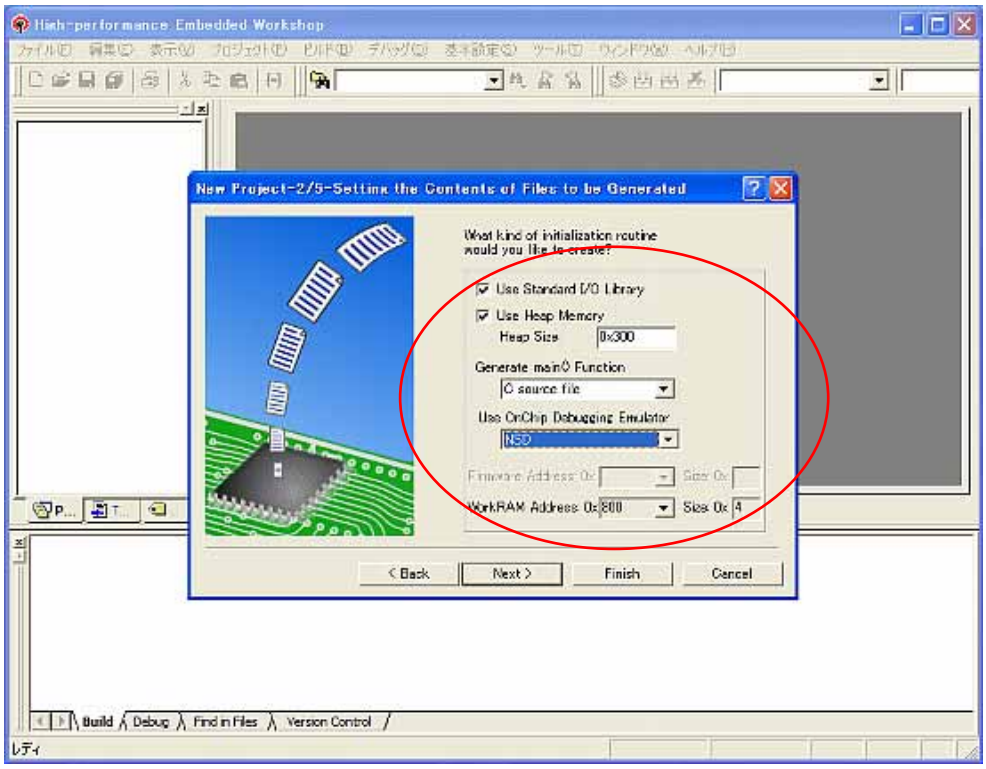
複数のコンパイラをインストールしている場合、「C source startup Application」選択後、「CPU 種別」で他マイコンを選択した場合、「C source startup Application」へのフォーカスが「Application」に移動して C ソーススタートアップの選択が無効になります。
この場合は、再度「C source startup Application」を選択してください。

(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。

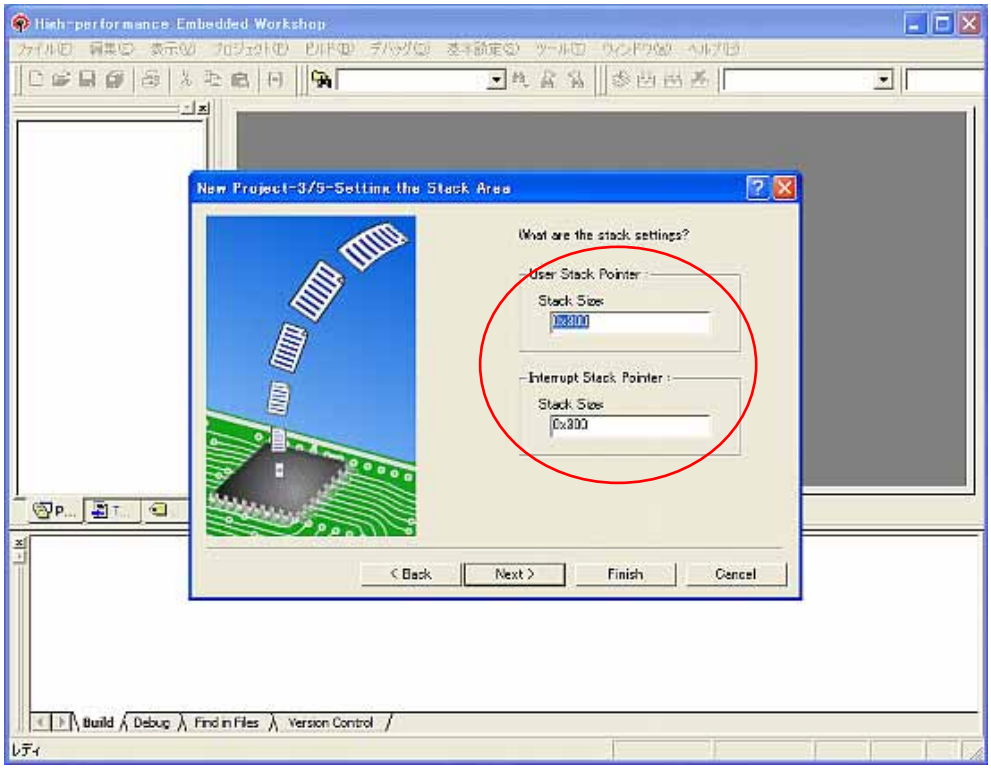


(1) 標準関数ライブラリを使用する場合
チェックすることにより、resetprg.c 中の_init( )呼び出しが有効になります。
また、ファイル device.c と init.c がプロジェクトに登録されます。

(2) 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。

(3) OnChip Debugging Emulator を使用する場合
選択可能なデバッガは、「FoUSB」と「NSD」です。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(UART0)に変わります。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。



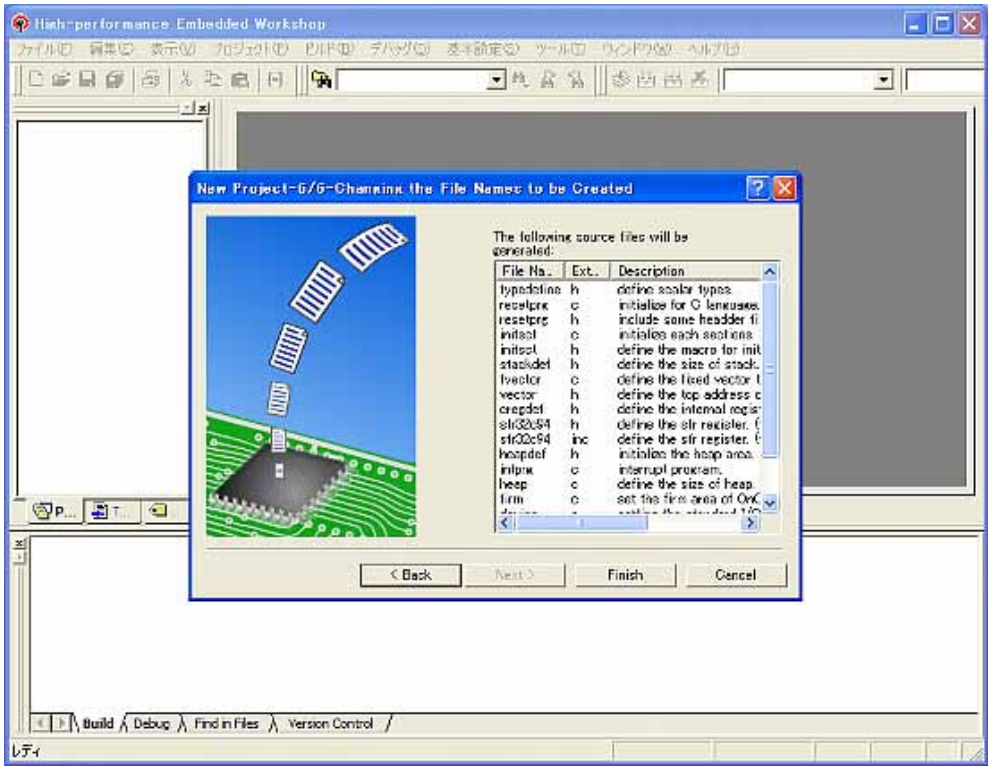
(1) ユーザスタックサイズの設定
stackdef.h が登録されます

(2) ユーザ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
```

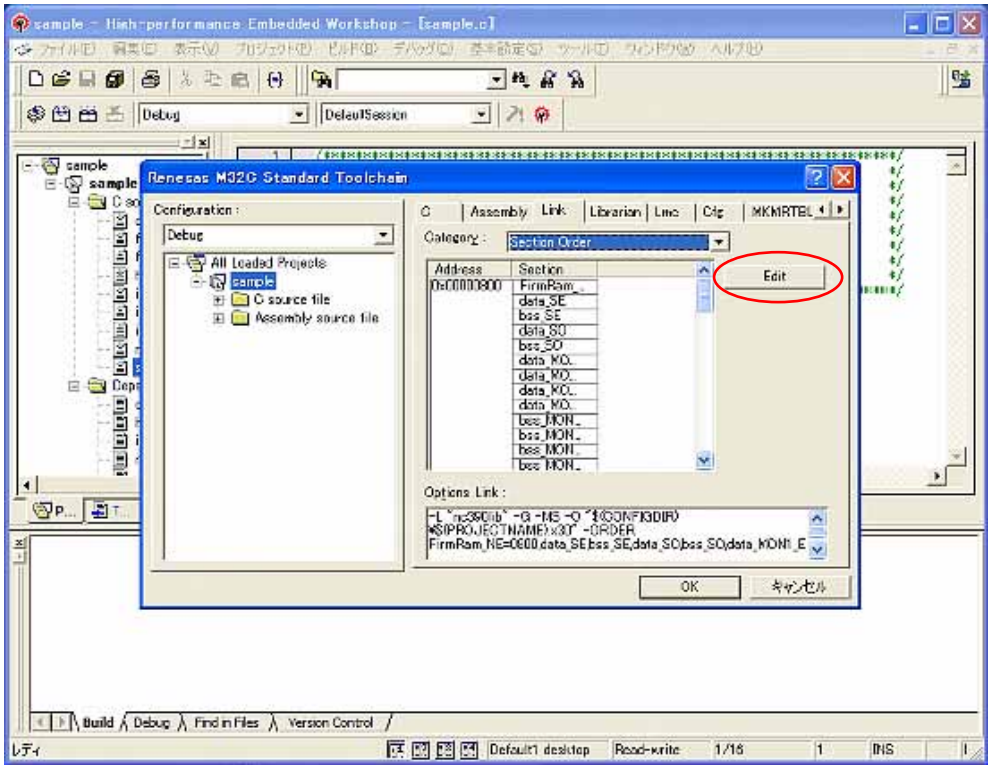
(5) 登録ファイルを確認します。



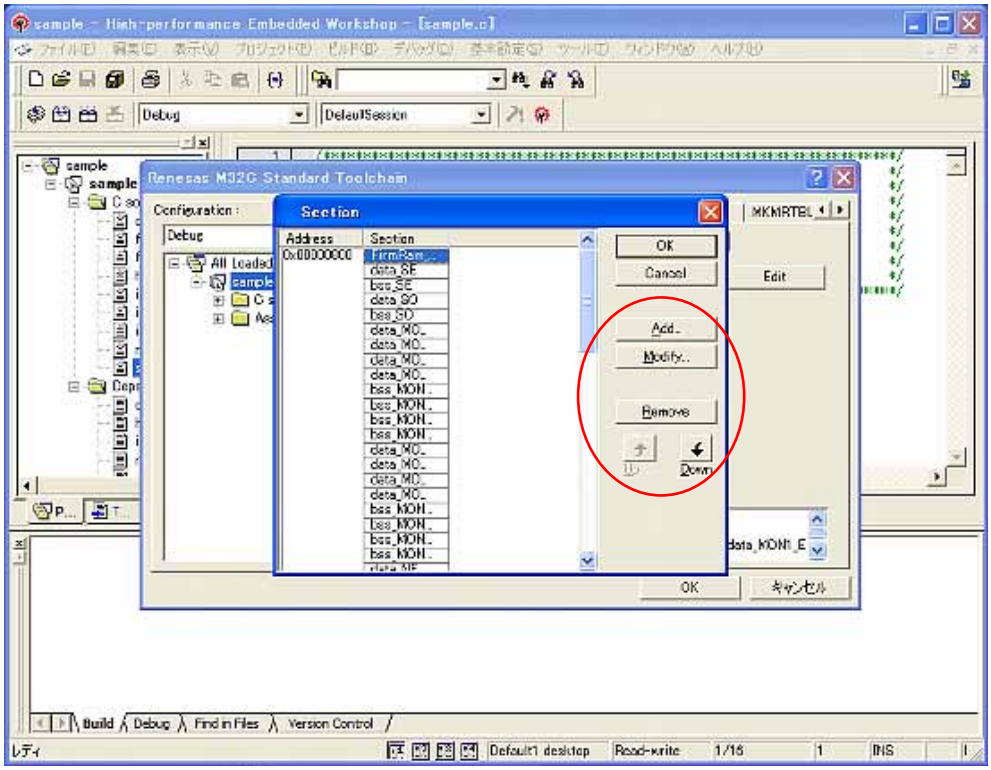
ここで、登録されるファイルを一覧で確認できます。
ただし、マイコン品種ごとに登録される sfr ヘッダ(C 言語用ヘッダ、アセンブラ用ヘッダ)は、ワークスペースへコピーのみです。

(6) セクションオーダー

「Renesas M32C 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。

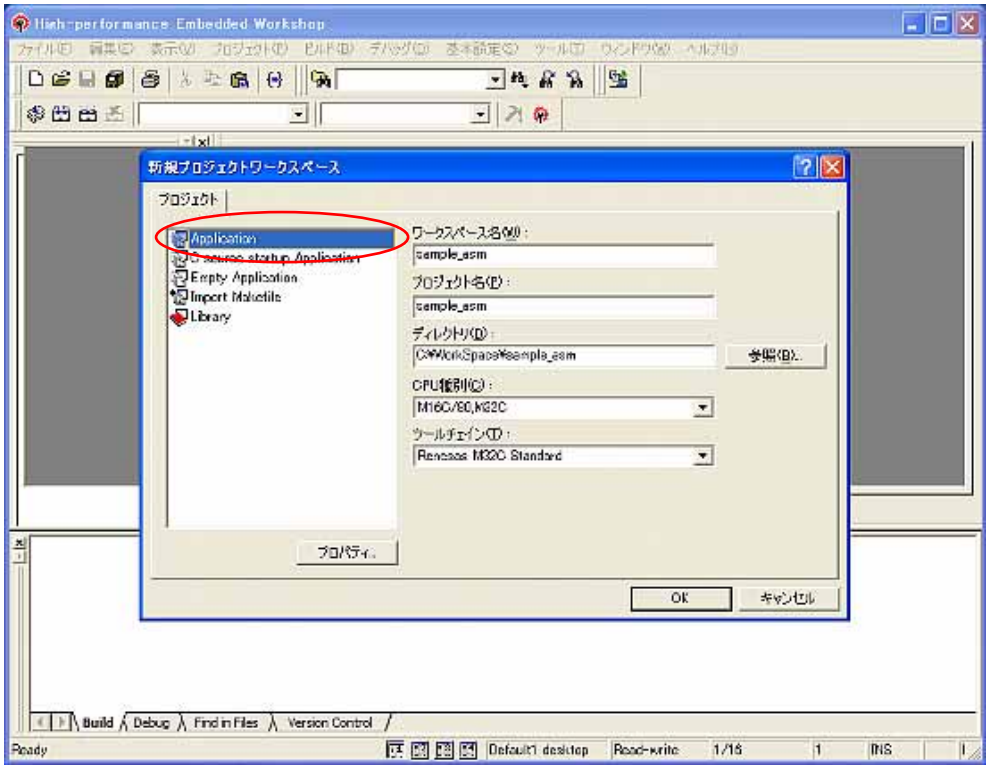


#pragma SECTION により新規にセクションを追加した等の場合は、「Edit」を選択して、「Section Window」をオープンし、セクションの配置等を変更します。



7.4. 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。



8. 付録

M32C/90,80,M16C/80,70 シリーズ用 C コンパイラパッケージ V.5.40 の C コンパイラユーザーズマニュアル、およびアセンブラユーザーズマニュアルに記載していない事項を記載しています。

8.1. C コンパイラユーザーズマニュアル

8.1.1. コード生成変更オプション

-M90

生成コードの変更

機 能 (1) M32C/90 シリーズに対応したコードを生成します。
(2) M32C90 をプリデファインドします。

注意事項 コンパイル・アセンブル時にオプション-M90 を選択した場合は、リンク時に標準関数ライブラリ nc390lib.lib を使用してください。

-fsizet_16 -fS16

型定義のビットサイズ変更

機 能 型定義 size_t を unsigned long 型から unsinged int 型に変更します。

注意事項 本オプションを選択した場合は、リンク時に以下の標準関数ライブラリを使用してください。

- M32C/90 シリーズ
  nc390_16.lib
- M32C/80 シリーズ
  nc382_16.lib
- M16C/80, /70 シリーズ
  nc308_16.lib

-f ptrdifft_16 -fP16

型定義のビットサイズ変更

機 能 型定義 ptrdiff_t を signed long 型から singed int 型に変更します。

注意事項 本オプションを選択した場合は、リンク時に以下の標準関数ライブラリを使用してください。

- M32C/90 シリーズ
  nc390_16.lib
- M32C/80 シリーズ
  nc382_16.lib
- M16C/80, /70 シリーズ
  nc308_16.lib

8.1.2. RAM モニタ領域の配置指定機能

#pragma MONITOR[n](n は 1～4)

RAM モニタ領域の配置指定機能

機 能 指定された外部変数を RAM モニタ領域専用のセクションに配置することを宣言します。

書 式 #pragma MONITOR[n]△外部変数名
(n は、1～4)

規 定

1. 指定できる変数は外部変数および外部 static 変数のみです。
2. #pragma MONITOR[n]で宣言された外部変数の領域は、下記のセクションに割り当てられます。

   - data_MON[n]_E (初期値を持つ偶数サイズの外部変数を配置)
   - data_MON[n]_O (初期値を持つ奇数サイズの外部変数を配置)
   - bss_MON[n]_E (初期値を持たない偶数サイズの外部変数を配置)
   - bss_MON[n]_O (初期値を持たない奇数サイズの外部変数を配置)
   - data_MON[n]_EI (初期値を持つ偶数サイズの外部変数の初期値を配置)
   - data_MON[n]_OI (初期値を持つ奇数サイズの外部変数の初期値を配置)

3. #pragma MONITOR[n]の宣言は外部変数の定義よりも前に行う必要があります。
4. #pragma MONITOR[n]で宣言した外部変数は他の#pragma 拡張機能と併用することはできません。ただし、#pragma SBDATA と#pragma MONITOR[n]が同時に指定された場合は#pragma SBDATA が優先されます。この際、ワーニングは出力されません。

注 意

1. #pragma MONITOR[n]はコンパイラが生成するオペコードに影響を与えません。変数の near/far 属性に注意してください。
2. RAM モニタ領域専用のセクションに near/far 属性が異なる外部変数が混在してもエラー・警告にはなりません。変数の near/far 属性に注意してください。
3. RAM モニタ領域専用のセクションにサイズ制限はありません。
4. #pragma MONITOR[n]により割り当てられるセクションの配置アドレス、および外部変数の初期値設定処理はスタートアッププログラムで記述してください。
5. 同一外部変数に対して#pragma MONITOR[n]を複数回指定した場合は、最初に指定された#pragma MONITOR[n]が有効になります。
6. コンパイルオプション-fno_even[-fNE]を指定している場合は、奇数サイズ属性のセクション(例：data_MON1_O)に配置されます。
7. #pragma SECTION の影響は受けません。
8. #pragma MONITOR[n]の n が 1～4 以外の場合は無効になります。コンパイルオプション-Wunknown_pragma[-WUP]または-Wall を指定した場合はワーニングを出力します。
9. ROM 属性を持つ外部変数は#pragma MONITOR[n]の対象外になります。ただし、コンパイルオプション-fconst_not_ROM[-fCNR]を指定している場合は#pragma MONITOR[n]対象になります。

```
#pramga   MONITOR1        i
const     int             i;   <==== 無効
```

10. #pragma MONITOR[n]はコンパイルオプション-M82 および-M90 指定の影響を受けません。

使用例

```
#pragma MONITOR1        i
#pragma MONITOR2        c

int       i;
char      c;
```