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。

お客様各位

資料中の「三菱電機」、「三菱XX」等名称の株式会社ルネサス テクノロジへの変更について

2003年4月1日を以って株式会社日立製作所及び三菱電機株式会社のマイコン、ロジック、アナログ、ディスクリート半導体、及びDRAMを除くメモリ(フラッシュメモリ・SRAM等)を含む半導体事業は株式会社ルネサス テクノロジに承継されました。
従いまして、本資料中には「三菱電機」、「三菱電機株式会社」、「三菱半導体」、「三菱XX」といった表記が残っておりますが、これらの表記は全て「株式会社ルネサス　テクノロジ」に変更されておりますのでご理解の程お願い致します。尚、会社商標・ロゴ・コーポレートステートメント以外の内容については一切変更しておりませんので資料としての内容更新ではありません。

注:「高周波・光素子事業、パワーデバイス事業については三菱電機にて引き続き事業運営を行います。」

2003年4月1日
株式会社ルネサス テクノロジ
カスタマサポート部

User's Manual

TW32R V.3.10
ユーザーズマニュアル

M32R ファミリ用　クロスツールキット(GNU 版)

ルネサス エレクトロニクス
www.renesas.com

Rev.1.00　2000.06

《安全設計に関するお願い》

三菱電機株式会社・三菱電機セミコンダクタシステム株式会社は品質、信頼性の向上に努めておりますが、半導体製品は故障が発生したり、誤動作する場合があります。弊社の半導体製品の故障又は誤動作によって結果として、人身事故、火災事故、社会的損害などを生じさせないような安全性を考慮した冗長設計、延焼対策設計、誤動作防止設計などの安全設計に十分ご留意ください。

《本資料ご利用に際しての留意事項》

- 本資料は、お客様が用途に応じた適切な三菱半導体製品をご購入いただくための参考資料であり、本資料中に記載の技術情報について三菱電機株式会社・三菱電機セミコンダクタシステム株式会社が所有する知的財産権その他の権利の実施、使用を許諾するものではありません。
- 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他応用回路例の使用に起因する損害、第三者所有の権利に対する侵害に関し、三菱電機株式会社・三菱電機セミコンダクタシステム株式会社は責任を負いません。
- 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他全ての情報は本資料発行時点のものであり、三菱電機株式会社・三菱電機セミコンダクタシステム株式会社は、予告なしに、本資料に記載した製品または仕様を変更することがあります。三菱半導体製品のご購入に当たりましては、事前に三菱電機株式会社・三菱電機セミコンダクタシステム株式会社または特約店へ最新の情報をご確認頂きますとともに、三菱電機半導体情報ホームページ（http://www.semicon.melco.co.jp/）および三菱開発ツールホームページ（http://www.tool-spt.mesc.co.jp/）などを通じて公開される情報に常にご注意ください。
- 本資料に記載した情報は、正確を期すため、慎重に制作したものですが万一本資料の記述誤りに起因する損害がお客様に生じた場合には、三菱電機株式会社・三菱電機セミコンダクタシステム株式会社はその責任を負いません。
- 本資料に記載の製品データ、図、表に示す技術的な内容、プログラム及びアルゴリズムを流用する場合は、技術内容、プログラム、アルゴリズム単位で評価するだけでなく、システム全体で十分に評価し、お客様の責任において適用可否を判断してください。三菱電機株式会社・三菱電機セミコンダクタシステム株式会社は、適用可否に対する責任は負いません。
- 本資料に記載された製品は、人命にかかわるような状況の下で使用される機器あるいはシステムに用いられることを目的として設計、製造されたものではありません。本資料に記載の製品を運輸、移動体用、医療用、航空宇宙用、原子力制御用、海底中継用機器あるいはシステムなど、特殊用途へのご利用をご検討の際には、三菱電機株式会社・三菱電機セミコンダクタシステム株式会社または特約店へご照会ください。
- 本資料の転載、複製については、文書による三菱電機株式会社・三菱電機セミコンダクタシステム株式会社の事前の承諾が必要です。
- 本資料に関し詳細についてのお問い合わせ、その他お気付きの点がございましたら三菱電機株式会社・三菱電機セミコンダクタシステム株式会社または特約店までご照会ください。

製品の内容及び本書についてのお問い合わせ先

電子メールの場合：インストーラが生成する以下のテキストファイルに必要事項を記入の上、開発ツールサポート窓口support@tool.mesc.co.jpまで送信ください。

Windows 98/95/Windows NT 4.0版：\SUPPORT\製品名\SUPPORT.TXT
EWS版：/support/製品名/toolinfo.txt

FAXの場合：各ユーザーズマニュアル及びガイドブックの最後に添付されている「技術サポート連絡書」に必要事項を記入の上、開発ツールサポート窓口まで送信ください。FAX送信先は「技術サポート連絡書」に記載してあります。

Short Contents

Table of Contents

はじめに

このユーザーズマニュアルをお読みになる前に、製品に添付されているリリースノートを必ずお読みください。製品構成、製品の扱い、注意事項などの重要な内容が記述されています。

■対象読者

このユーザーズマニュアルでは、次の方を対象としています。

- C 言語による組み込み用プログラム開発およびデバッグ経験のある方
- 三菱統合化開発環境を初めてご使用になる方

■参考マニュアル

三菱統合化開発環境 (TM) に関する詳しい用語説明、機能説明、操作については、次のマニュアルを参照してください。このマニュアルはオンラインでも参照することができます。

- 三菱開発サポートツール用統合化開発環境
  TM V.2.00 ユーザーズマニュアル

1 基礎知識編

1.1 TW32R のソフトウェア構成

TW32R は、次のソフトウェアから構成されています。

●統合化開発環境 (TM)

GUI(Graphical User Interface) 操作による、コンパイラ、アセンブラ、リンカなどの C コンパイラ系ツールの制御機能をサポートし、アプリケーションプログラムの開発作業を効率よく実施するための開発ツールです。また、アプリケーションプログラムを構成するソースファイルの情報も管理します。

●M32R 用 GNU C コンパイラ系ツール

M32R 用 GNU C コンパイラ系ツールを同梱しています。M32R 用 GNU C コンパイラ系ツールのコマンドは、GNU 標準のコマンド名の前に、'`m32r-elf-`'が付いたコマンド名になっています。次のコマンドより構成されています。[1]

`m32r-elf-gcc`
: C コンパイラ

`m32r-elf-as`
: アセンブラ

`m32r-elf-ld`
: リンカ

`m32r-elf-ar`
: アーカイブツール

`m32r-elf-ranlib`
: アーカイブインデックス作成ツール

`m32r-elf-objcopy`
: オブジェクトコピー、変換ツール

`m32r-elf-strip`
: シンボル削除ツール

`m32r-elf-nm`
: オブジェクトシンボル表示ツール

`m32r-elf-objdump`
: オブジェクト情報表示ツール

`m32r-elf-size`
: オブジェクトサイズ表示ツール

`m32r-elf-strings`
: 文字列表示ツール

[1] これらの GNU ツールの扱いは、GNU General Public License に従います。

●三菱製 ANSI C 標準ライブラリ

三菱製 ANSI C 準拠の C 標準ライブラリ (libc.a) と数値計算ライブラリ (libm.a) が含まれています。

1.2 ツールのドキュメント

各ツールの使用方法、説明に関する詳細は、TW32R に含まれているドキュメントを参照してください。[2] ドキュメントは、スタートメニューから選択して参照することができます。それぞれ、次のようにメニューを選択してください[3] 。

●TM のドキュメント (PDF 形式、日本語)

```
[スタート] → [プログラム] → [MITSUBISHI-TOOL] →
[TM V.2.01] → TM ユーザーズマニュアル
```

●M32R 用 GNU C コンパイラ系ツール (HTML 形式および PDF 形式、英語)

```
[スタート] → [プログラム] → [MITSUBISHI-TOOL] →
[TW32R V.3.10 Release 1] → TW32R ドキュメント
```

●三菱製 ANSI C 標準ライブラリ (HTML 形式および PDF 形式、日本語)

```
[スタート] → [プログラム] → [MITSUBISHI-TOOL] →
[TW32R V.3.10 Release 1] → TW32R ドキュメント
```

1.3 TM のウインドウ構成

TM のウインドウ (Main ウインドウ) は、次の部分から構成されています。

●メニューバー

TM を操作するためには、このメニューバーから機能を選択します。メニューを選択することにより、TM の全ての機能を実行できます。

---

[2] PDF 形式のドキュメントを参照する場合、Acrobat Reader 3.0Jがインストールされている必要があります。また、HTML 形式のドキュメントを参照する場合、HTML 形式のドキュメントを表示可能なブラウザまたは同等のツールがインストールされている必要があります。

[3] スタートメニューに登録されるショートカット名は、TW32R のバージョンにより異なるため、ここに示したバージョン番号と異なることがあります。

●ツールバー

メニューバーから選択できる機能のうち、使用頻度の高いものをツールバーにボタンとして配置してあります。メニューバーから選択するのと同様に使用できます。

●ツリーウインドウ

プロジェクトに含まれるターゲットファイル、ソースファイル、ヘッダファイルをアイコンで階層的に表示します。また、関数の依存関係もビジュアルに表示します。

●ビューウインドウ

ソースファイルの内容を表示します。ツリーウインドウでファイルのアイコン(またはファイル名)をクリックすると、ファイルの内容がビューウインドウに表示されます。

●Output ウインドウ

コンパイル、アセンブル、リンクの実行フェーズや実行結果を表示します。また、外部ツールの実行結果も表示します。

●ステータスバー

ツールバーの各ボタン上にマウスカーソルを合わせた時に、そのボタン機能の簡易説明を表示したり、TM の状態を表示したりします。

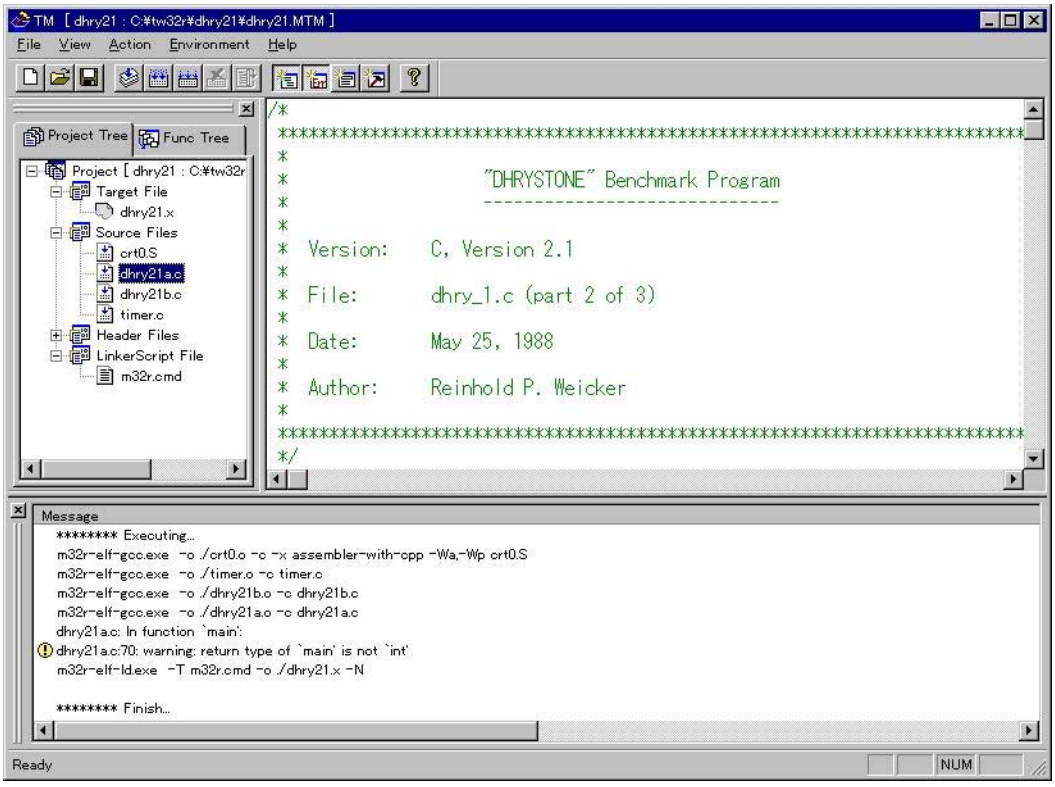


図 1. TM のウインドウ構成

1.4 TM のプロジェクト管理

TM はプロジェクトという単位で、アプリケーションの開発環境を管理します。開発環境には次の情報が含まれています。

- アプリケーションのターゲット MCU
- アプリケーションの構成ファイル (C ソースファイル、インクルードファイル)
- アプリケーションのターゲットプログラム名
- クロスツールおよびオプション
- デバッガ
- エディタ
- 環境変数

TM は、これらの情報を 'プロジェクト名.mtm'ファイルにまとめて管理し、これらの情報を参照して、ターゲットプログラムファイルであるロードモジュール (実行形式ファイル) を作成します。

1.5 TM を使ったプログラム開発手順の概要

TM を使用したプログラム開発の手順は、以下のようになります。

(1) プロジェクトの作成
　　プロジェクトを作成し、使用するエディタ、アプリケーションの構成ファイル、クロスツール、オプション、デバッガなどを登録します。

(2) ビルドの実行
　　ビルドの実行により、プロジェクトの登録内容に従ってコンパイラ、アセンブラ、リンカが起動されターゲットプログラムが生成されます。

(3) デバッグの実行
　　プロジェクトに登録されているデバッガが起動され、ターゲットプログラムのデバッグをすることができます。

2 開発工程編

2.1 TM を起動する

TM を実際に使ってみましょう。

ここでは、"Hello World!"のプログラムを作成し、モトローラ S-record ファイル[1] を作成するまでの例を示しながら説明します。

TM を起動するには、Windows のスタートメニューを利用してください。インストールが完了していれば、次のようにメニューを選択することにより起動できます[2] 。

```
[スタート] → [プログラム (P)] → [MITSUBISHI-TOOL] →
[TM V.2.01] → TM
```

2.2 プロジェクトを作成する

ターゲットプログラムを作成するためには、まず新規プロジェクトを作成します。新規プロジェクトの作成は、以下の手順で行います。

2.2.1 ターゲット CPU の選択

TM メニューから [File] → [New Project...] を選択[3] します。これにより、新規プロジェクト設定ウィザード (New Project Wizard - Page 1) がオープンします。この Page 1 で次の項目を設定してください。

- *Target Chip*
  使用するマイコンのファミリ名もしくはグループ名を選択します。
  ここでは *M32R Family* を選択してください。

- *Project Name*
  プロジェクト名を指定します。
  プロジェクト名は、プロジェクトを区別するために使用されます。また、ターゲットファイル名はプロジェクト名に拡張子 (M32R Family の場合、".x") が付けられます。
  例えば、プロジェクト名として 'hello'を指定した場合、ターゲットファイル名は 'hello.x'になります。ここでは 'hello' と入力してください。

---

[1] モトローラ形式の S フォーマット

[2] スタートメニューに登録されるショートカット名は、TM のバージョンにより異なるため、ここに示したバージョン番号と異なることがあります。

[3] ツールバーの左端のボタンをクリックすることでも同様に、新規プロジェクト設定ウィザードをオープンできます。

プロジェクト名には、任意の名前を指定することができますが、ディレクトリ名として使用できない文字は指定できません。

- *Working Directry*

  プロジェクトの作業ディレクトリのパスを指定します。作業ディレクトリが希望のパス位置になるようにドライブレター付きでパスを指定してください。任意のパスが指定できますが、ディレクトリ名として使用できない文字は指定できません。

図 2. 新規プロジェクト作成ウインドウ 1

2.2.2 ツールの選択

[次へ>] ボタンを押し、次のウインドウ画面に移動します。これにより、New Project Wizard - Page 2 がオープンします。この Page 2 で次の項目を設定してください。

- *Compiler Package*

  使用するクロスツール名を指定します。ここでは M32R 用統合クロスツールキット TW32R を選択してください。

  チェックボックスをチェックすると、C 言語の開発に必要なファイルを使用できるようにします。通常は、チェックしておくことを推奨します。ここではチェックしておきます。

図 3. 新規プロジェクト作成ウインドウ 2

2.2.3 OS パッケージおよびデバッガパッケージの選択

[次へ>] ボタンを押し、次のウインドウ画面に移動します。

これにより、New Project Wizard - Page 3 がオープンします。この Page 3 では、リアルタイム OS を使用する場合のパッケージを設定します。今回の例では使用しないので、そのまま [次へ>] ボタンを押し、次のウインドウ画面に移動します。

また、次の New Project Wizard - Page 4 では、使用するデバッガのパッケージを選択します。今回の例では使用しないので、そのまま [次へ>] ボタンを押し、次のウインドウ画面に移動します。

2.2.4 プロジェクトの設定確認

[次へ>] ボタンを押し、次のウインドウ画面に移動します。これにより、New Project Wizard - Page 5 がオープンします。ここで設定内容を確認してください。

図 4. 新規プロジェクト作成ウインドウ 3

これらの設定が正しければ [完了] ボタンを押し、プロジェクトの作成を完了します。設定を変更したい場合、[<戻る] ボタンを押し、設定をやり直してください。

プロジェクトの作成が完了すると、File Tree ウインドウ画面にプロジェクトファイルのフォルダアイコン '`hello.mtm`'が表示されます。このフォルダ以下に、このプロジェクトを構成する Target File、Source File、Header File が階層的に管理されます。

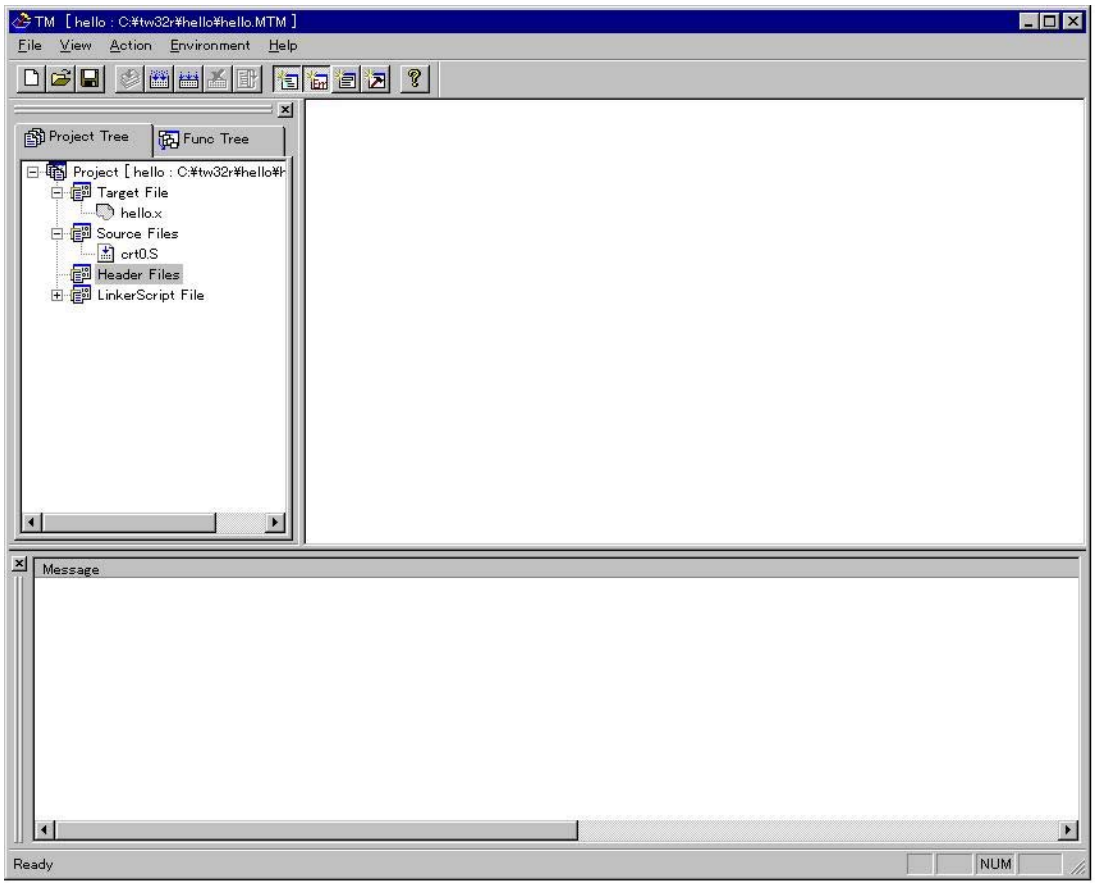


図 5. 新規プロジェクト作成ウインドウ 4

2.3 ソースファイルを登録する

プロジェクトにソースファイルを登録するには、メニューから [Environment] → [Add Source File] → [Select File...] を選択し、登録するソースファイル名を指定します。新規ファイルを作成する場合には、[New File...] を選択します。

ここでは [New File...] を選択し、ファイル名に 'hello.c'を指定して新規作成で登録してください。

TM には、デフォルトのエディタとしてメモ帳 (Notepad.exe) が登録されていますので、ここでは、メモ帳が起動され 'hello.c'の新規ファイルがオープンされます。ここで以下のプログラムを入力し、上書き保存してエディタを終了してください。

```
#include <stdio.h>
main()
{
        printf("Hello World!\n");
}
```

2.4 ビルド (コンパイル、リンク) を実行する

ツールバーの<Build>ボタンをクリックするか、メニューから [Action] → [Build] を選択してプロジェクトのビルドを実行してください。ビルドとリビルド作業の内容は次の通りです。

- ビルド　コンパイル、アセンブル、リンクを実行してターゲットプログラムを生成します。makeユーティリティの機能を利用して、ソースファイルなどに変更があった場合のみコンパイル、アセンブル、リンクを実行します。
- リビルド　ソースファイルの修正の有無に関わらず全てのソースファイルをコンパイル、アセンブル、リンクします。日付の古いソースファイルをコピーし、再コンパイル、再アセンブル、再リンクしたい場合などに使用します。

2.5 ビルド (リビルド) の経過及び結果の確認

ビルド (リビルド) を実行すると、実行状況が Output ウインドウに表示されます。各フェーズの実行状況を確認してください。

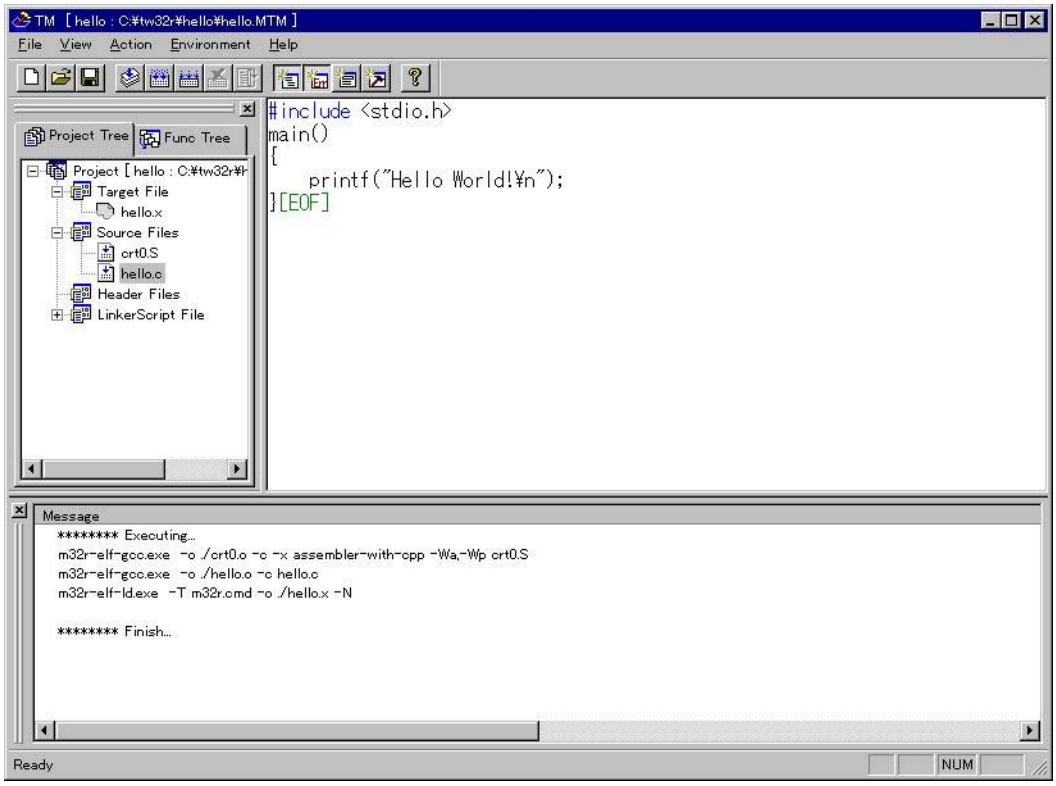


図 6. ビルドの実行

2.6 ソースファイルを編集する

ソースファイルの記述に間違いがあればファイルを編集し修正します。ファイルの編集を行うには、File Tree に表示されているソースファイルをダブルクリックします。これによりプロジェクトに登録されているエディタが起動され、ファイルがオープンされます。

ソースファイルを修正できたら、再びビルドを実行します。

2.7 モトローラ S-record ファイルを作成する

作成したターゲットプログラム 'hello.x'をモトローラ S-record ファイルに変換してみましょう。

ここでは、TM の Other Tool 機能を使用して、モトローラ S-record ファイルへの変換ツール (m32r-elf-objcopy) の起動により変換します。変換作業は以下の手順で行います。

1. メニューから [Action] → [Other] → [m32r-elf-objcopy] を選びます。
2. *Parameter* に '-O srec hello.x hello.mot'と入力します。
   '-O srec'は S-record 変換、'hello.x'は入力ファイル、'hello.mot'は出力ファイルをそれぞれ示します。
3. <OK>ボタンをクリックします。
4. これで Working Directory ディレクトリに、'hello.mot'の S-record ファイルが作成されます。

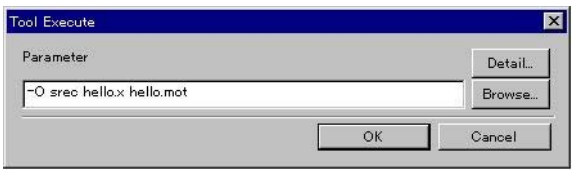


図 7. Other Tool の実行 (Tool Execute) ウインドウ

2.8 プロジェクトを保存する

ツールバーの<Save Project>ボタンをクリックするか、メニューから [File] → [Save Project] を選択し、作成したプロジェクトを保存します。プロジェクトは、プロジェクト名.MTM(拡張子が.MTM) というファイル名で保存されます。ここでは 'hello.mtm'というファイルに保存されます。

保存したプロジェクトファイルをオープンすることにより、同じ設定でプロジェクトに対する作業を再開できます。

3 基本操作編

3.1 ヘルプを参照する

メニュー [Help] → [Help Topics] から目的のファイルを選択し、ヘルプ画面を表示します。

3.2 TM からオンラインマニュアルを参照する

オンラインマニュアルを参照するには、メニュー [Help] → [Online Manual] から目的のファイルを選択し、オンラインマニュアルを表示します。次のオンラインマニュアルを参照することができます。

- TM ユーザーズマニュアル (PDF 形式)
- TW32R ドキュメント (PDF 形式および HTML 形式)

ただし、PDF 形式のオンラインマニュアルを表示するためには、`Acrobat Reader 3.0J`がインストールされている必要があります。HTML 形式のオンラインマニュアルを表示するためには、HTML ドキュメントを表示可能なブラウザまたは同等のツールがインストールされている必要があります。また、プロジェクトが読み込まれていない場合は、TW32R のオンラインマニュアルは参照できません。

3.3 コンパイラのオプション設定

コンパイラのオプション設定は、次の手順で行います。

1. メニューから [Environment] → [Build Settings] を選択します。
2. Build Settings ダイアログの左側に表示されているツールのツリーから *C Compiler* を選択します。
3. Build Settings ダイアログの右側の *Group* から、変更するオプショングループを選択します。
4. *Group* の下に表示されるオプション一覧より、*Option* のチェックボックスをクリックしてオプションを設定します。
5. *All Source* ボタンを押します。
6. <OK>ボタンをクリックし、*Build Settings* ダイアログを閉じます。

この設定は、全ての C 言語ソースに対して有効[1] になります。

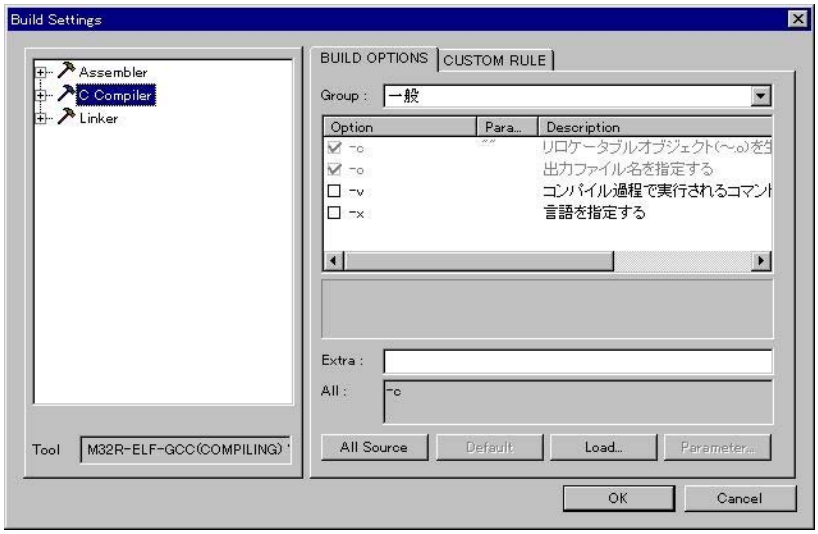


図 8. コンパイラオプション設定ウインドウ

3.4 アセンブラのオプション設定

アセンブラのオプション設定は、次の手順で行います。

1. メニューから [Environment] → [Build Settings] を選択します。
2. Build Settings ダイアログの左側に表示されているツールのツリーから *Assembler* を選択します。
3. *Group* の下に表示されるオプション一覧より、*Option* のチェックボックスをクリックしてオプションを設定します。
4. *All Source* ボタンを押します。
5. <OK>ボタンをクリックし、*Build Settings* ダイアログを閉じます。

[1] ソースファイル名を選択後、オプション設定を変更することにより、ファイル毎のオプションも設定することができます。

この設定は、全てのアセンブリ言語ソースに対して有効になります。[2]

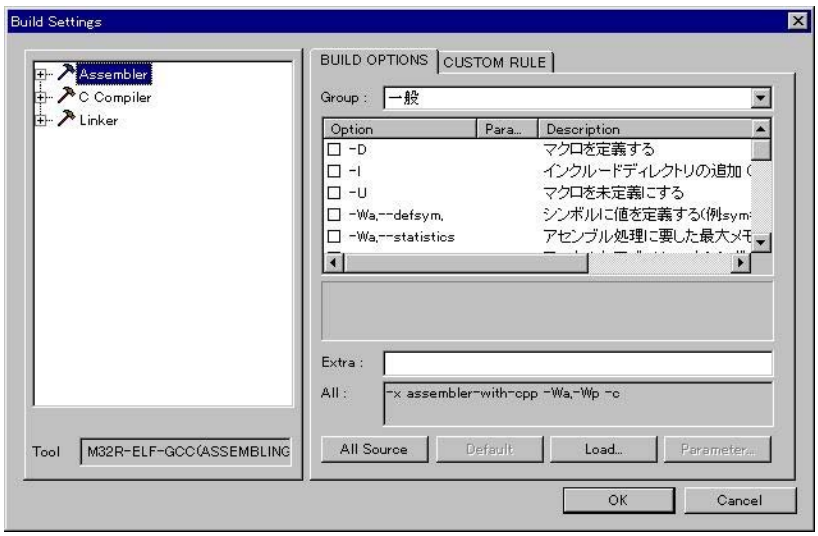


図 9. アセンブラオプション設定ウインドウ

3.5 リンカのオプション設定

リンカのオプション設定は、次の手順で行います。

1. メニューから [Environment] → [Build Settings] を選択します。
2. Build Settings ダイアログの左側に表示されているツールのツリーから *Linker* を選択します。
3. *Group* の下に表示されるオプション一覧より、*Option* のチェックボックスをクリックしてオプションを設定します。
4. *All Source* ボタンを押します。
5. <OK>ボタンをクリックし、*Build Settings* ダイアログを閉じます。

[2] ソースファイル名を選択後、オプション設定を変更することにより、ファイル毎のオプションも設定することができます。

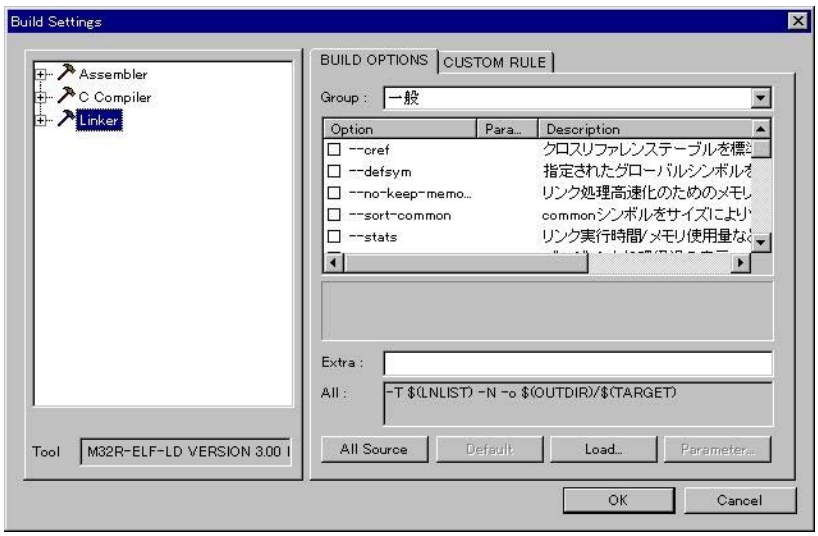


図 10. リンカオプション設定ウインドウ

3.6 メモリモデルの設定/変更

メモリモデル設定/変更は、次の 2 点を設定します。また、設定を変更した場合、リビルドする必要があります。

- C コンパイラのオプション`-mmodel=<`*mmodel-name*`>`に、対応するメモリモデルを指定します。
- リンカのオプション`-mmodel=<`*mmodel-name*`>`に、対応するメモリモデルを指定します。

C コンパイラとリンカのメモリモデルオプションの設定は、次の手順で行います。

1. メニューから [Environment] → [Build Settings] を選択します。
2. Build Settings ダイアログの左側に表示されているツールのツリーから *C Compiler* を選択します。
3. Build Settings ダイアログの右側の *Group* から、コード /データモデルのオプショングループを選択します。
4. *Group* の下に表示されるオプション一覧より、'`-mmodel=`'オプションの行を選択して *Parameter...* ボタンを押します。
5. *Select Options* ダイアログ中の *Option* から、該当するメモリモデルのチェックボックスをクリックします。クリックしたら *OK* ボタンを押し、*Select Options* ダイアログをクローズします。
6. *All Source* ボタンを押します。
7. 次に Build Settings ダイアログ左側に表示されているツールのツリーから *Linker* を選択し、*C Compiler* の場合と同様にオプションを指定します。
8. <OK>ボタンをクリックし、*Build Settings* ダイアログを閉じます。

4 基本編集編

4.1 リンカスクリプトの概要

TW32R における TM 環境では、プログラムのロードモジュール作成 (ビルドおよびリビルド作業) 時にリンカスクリプトを利用します。リンカスクリプトは、ロードモジュール作成に関する次の情報を定義するものです。

- 入力オブジェクトファイル
- デフォルトのオブジェクトおよびライブラリ検索パス
- デフォルトの入力ライブラリ
- 入出力ファイルのフォーマット
- 出力ファイルのアーキテクチャ
- プログラムのエントリアドレス
- メモリレイアウト
- セクションレイアウト (メモリ配置、実行アドレス、ロードアドレス)

開発中のターゲットプログラム (プロジェクト) に最適なように、これらの情報を編集し定義することができます。リンカスクリプトは、新規プロジェクトを作成した時に、サンプルのリンカスクリプト 'm32r.cmd'がそのプロジェクト用に Working Directory にコピーされます。TM では、コピーされたリンカスクリプトを利用してターゲットプログラムをリンクします。

4.2 リンカスクリプトの編集

リンカスクリプト 'm32r.cmd'を編集するには、メニューから [Environment] → [Section] を選択します。TM に登録されているエディタが起動され、'm32r.cmd' ファイルがオープンされます。

4.3 リンカスクリプトの詳細

サンプルのリンカスクリプトの各部について説明します。記述方法に関する詳細は、リンカ 'ld'のドキュメントを参照してください。

4.3.1 入力オブジェクトファイルの定義

入力オブジェクトファイルの定義は、TM のプロジェクト管理機能により自動管理および更新されます。この定義は、次のキーワードで囲まれた部分に対して、TM がロードモジュール作成時 (ビルドおよびリビルド時) に適切に自動編集します。

- 入力オブジェクトファイル

```
/*[MTM_INPUT_TOP]*/
            TM が入力オブジェクトファイルに関する情報を出力
/*[MTM_INPUT_END]*/
```

したがって、これらのキーワードの削除や変更、キーワード間の編集は行わないでください。また、TM の管理している入力オブジェクト以外のオブジェクトファイルを入力指定したい場合、これらのキーワードの範囲外に定義してください。

4.3.2 デフォルトのオブジェクトおよびライブラリ検索パスの定義

デフォルトのオブジェクトおよびライブラリ検索パスの定義は、TM のプロジェクト管理機能により自動管理および更新されます。この定義は、次のキーワードで囲まれた部分に対して、TM がロードモジュール作成時 (ビルド及びリビルド時) に適切に自動編集します。

- デフォルトのオブジェクトおよびライブラリ検索パス

```
/*[MTM_SEARCH_DIR_TOP]*/
  TM がデフォルトのオブジェクトおよびライブラリ検索パスを出力
/*[MTM_SEARCH_DIR_END]*/
```

したがって、これらのキーワードの削除や変更、キーワード間の編集は行わないでください。また、TM の管理している検索パス以外のパスを優先して検索パス指定したい場合、MTM_SEARCH_DIR_TOPキーワードより前に定義してください。

4.3.3 デフォルトの入力ライブラリの定義

デフォルトの入力ライブラリの定義は、TM のプロジェクト管理機能により自動管理および更新されます。この定義は、次のキーワードで囲まれた部分に対して、TM がロードモジュール作成時 (ビルド及びリビルド時) に適切に自動編集します。

- デフォルトの入力ライブラリ

```
/*[MTM_GROUP_TOP]*/
  TM がデフォルトの入力ライブラリを出力
/*[MTM_GROUP_END]*/
```

したがって、これらのキーワードの削除や変更、キーワード間の編集は行わないでください。また、TM の管理しているデフォルトの入力ライブラリ以外のライブラリを優先して指定したい場合、MTM_GROUP_TOP キーワードより前に定義してください。

4.3.4 入出力ファイルフォーマットの定義

入力オブジェクトフォーマットおよび出力オブジェクトフォーマットを指定します。この定義の変更や削除は行わないでください。

```
OUTPUT_FORMAT("elf32-m32r", "elf32-m32r", "elf32-m32r")
```

4.3.5 出力ファイルのアーキテクチャの定義

出力ファイルのアーキテクチャを指定します。この定義の変更や削除は行わないでください。

```
OUTPUT_ARCH(m32r)
```

4.3.6 エントリアドレスの定義

プログラムのエントリアドレス (シンボル) を定義します。ユーザープログラムのエントリアドレスを指定してください。

サンプルのリンカスクリプトでは、サンプルのスタートアッププログラム 'crt0.S'用のエントリアドレスとして__rom_startが指定されています。

```
ENTRY(__rom_start)
```

4.3.7 メモリレイアウトの定義

メモリレイアウトを定義し、各セクションがここで定義したメモリのどこに配置されるかを指定することにより、セクションの配置指定とオーバーフローチェックを行うことができます。サンプルのリンカスクリプトでは、メモリレイアウトの定義のみ行っており、配置指定には使用していません。

```
MEMORY
{
    in_ram1   : org = 0, len = 1M
    in_ram2   : org = 0, len = 2M
    in_ram4   : org = 0, len = 4M
    out_rom1  : org = 0x7fe00000, len = 1M
}
```

セクション配置とオーバーフローを正しくチェックする場合、これらのメモリレイアウト定義をターゲットのメモリ構成に合わせて適切に編集し、各セクションをメモリのどこに配置するか指定してください。

全てのオブジェクトの.textセクションが、*in_ram1* で定義したメモリに配置されるように指定するには、SECTIONS 定義内で次のように記述します。

```
SECTIONS {
    .text : { *(.text)  } > in_ram1
}
```

4.3.8 セクションの定義

リンクするセクションの指定および配置指定を定義します。サンプルのリンカスクリプトでは、セクションの定義は基本的に次のように定義しています。

```
SECTIONS {
  /* 実行時開始アドレス定義のシンボル */
  START_OF_RAM = 0x100;

  /* ロード時開始アドレス定義のシンボル */
  START_OF_ROM = 0x100;

  /* リセットベクタアドレス定義のシンボル */
  RESET = 0x7ffffff0;

  /* ロケーションカウンタの設定 */
  . = START_OF_RAM;

  /* 実行時アドレスとロード時アドレスを指定したセクション */
  出力セクション名（実行時アドレス）: AT(ロード時アドレス)
  {
    /* 実行時のセクション開始アドレス */
    __入力セクション名_start = .;

    /* 全てのセクション配置指定 */
    *(入力セクション名)

    /* 実行時のセクション終了アドレス */
    __入力セクション名_end = .;
  }
  /* ロード時のセクション開始アドレス */
  __rom_入力セクション名 = LOADADDR(出力セクション名);
  :
  :
  /* 実行時アドレスの指定のみ */
  出力セクション名  :
```

```
    {
      /* 実行時のセクション開始アドレス */
      __入力セクション名_start = .;

      /* 全てのセクション配置指定 */
      *(入力セクション名)

      /* 実行時のセクション終了アドレス */
      __入力セクション名_end = .;
    }
    .
    .
    .
}
```

この定義中のシンボルや用語は、それぞれ次のことを示しています。

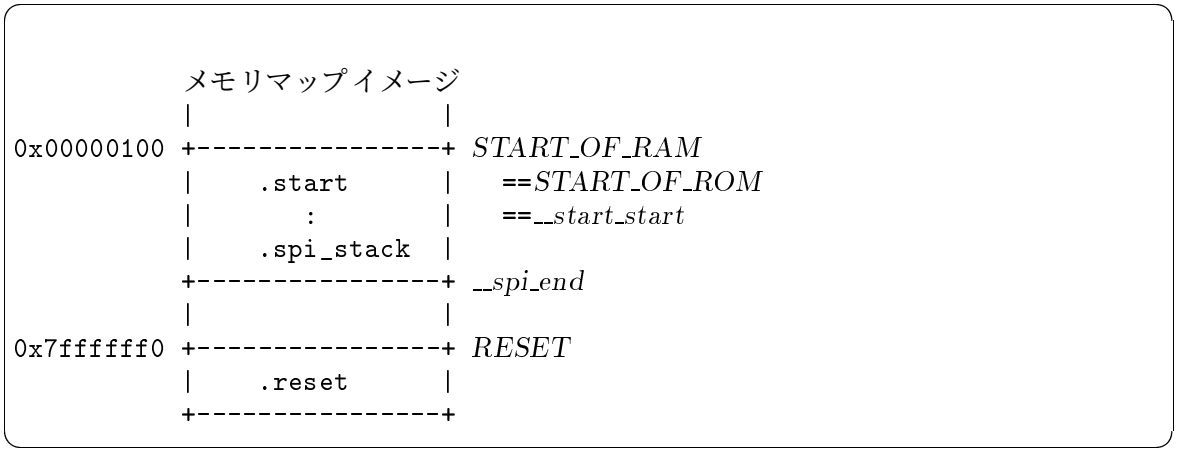
- 入力セクション名
  オブジェクト内に含まれるセクション名を示しています。
- 出力セクション名
  ロードモジュールに出力されるセクション名を示しています。
- 実行時開始アドレス
  プログラムやデータが実行される時に配置される開始アドレスを示しています。
- ロード時開始アドレス
  プログラムの実体がロード時に配置されるアドレスを示しています。
- __入力セクション名_*start* シンボル
- __入力セクション名_*end* シンボル
- __*rom*_入力セクション名シンボル
  サンプルのスタートアッププログラム 'crt0.S'内で各セクションの転送 (ROM から RAM への転送、あるいは、低速 RAM から高速 RAM への転送) 時に参照するために定義してあるグローバルシンボルであり、スタートアッププログラムとの整合をとる必要があります。

サンプルのリンカスクリプトでは、以下のセクションを定義しています。

.start	スタートアッププログラムのセクション
.init	C および C++用初期化処理プログラムのセクション
.text	コードのセクション
.fini	C および C++用終了処理プログラムのセクション
.rodata	初期値あり読み出し専用データのセクション
.data	初期値あり読み書きデータのセクション
.ctors	C++用グローバルコンストラクタテーブルのセクション
.dtors	C++用グローバルデストラクタテーブルのセクション
.sdata	初期値あり *small data area* のセクション
.sbss	初期値なし *small data area* のセクション

.bss	初期値なしデータのセクション
.heap	標準ライブラリで使用するヒープのセクション
.spu_stack	ユーザースタックのセクション
.spi_stack	割り込みスタックのセクション
.reset	リセットベクタのセクション

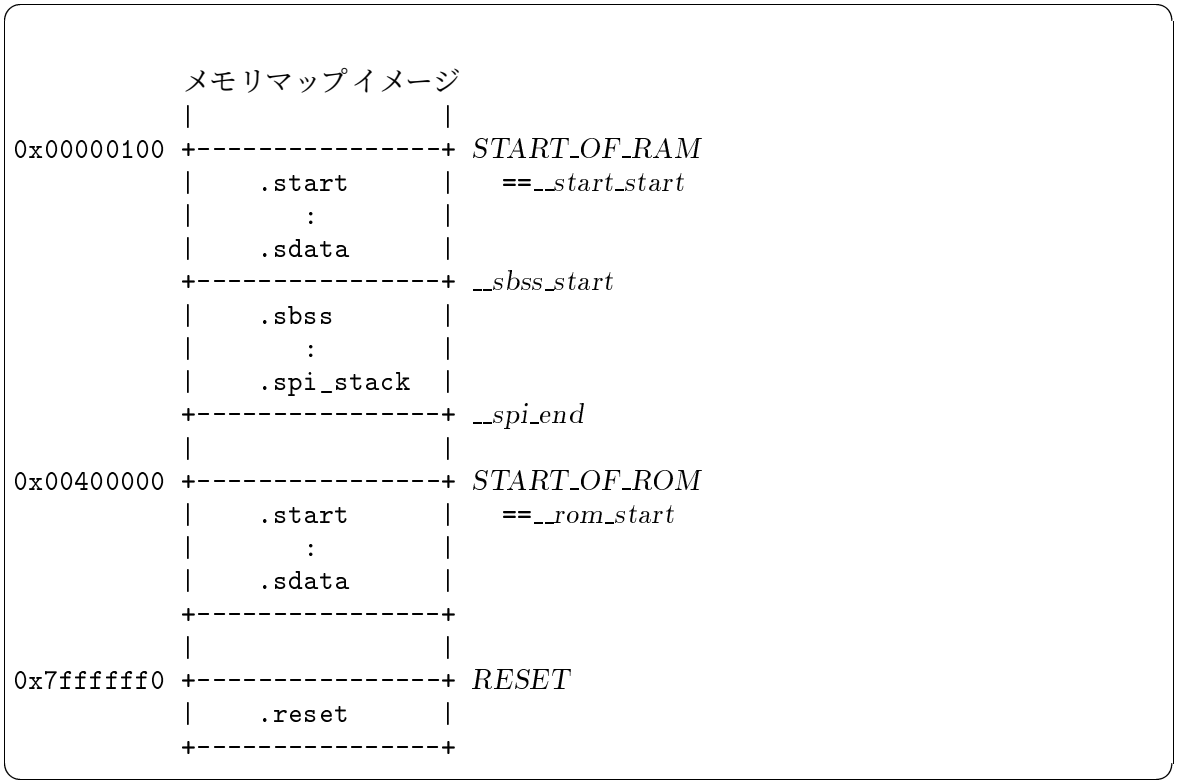
サンプルのリンカスクリプトでは、.startから.spi_stack までのセクションは、0x100番地からこの順番に連続に配置されるように定義しています。また、これらのセクションは、*START_OF_RAM* シンボル値 (実行時の開始アドレス) と *START_OF_ROM* シンボル値 (ロード時の開始アドレス) が同一となるよう定義しています。すなわち、プログラムコード及びデータの実体は、プログラム実行時のメモリに配置されます。



START_OF_RAM シンボルと *START_OF_ROM* シンボルを別々のアドレスに定義[1] することにより、実行時のアドレスとロード時のアドレスをそれぞれ指定することができます。

例えば、*START_OF_RAM* を RAM 領域、*START_OF_ROM* を ROM 領域に定義することにより、ROM 化可能なロードモジュールを作成することができます。この場合、サンプルのリンカスクリプトでは.sbssセクションから.spi_stack セクションまでは、実行時のアドレスにそのまま配置されます。以下に *START_OF_RAM* に 0x100 番地を指定し、*START_OF_ROM* に 0x400000 番地を指定した場合のメモリマップイメージを示します。

[1] 実行時のアドレス範囲とロード時のアドレス範囲がオーバーラップしないように注意してください。



.resetセクションは、リセットベクタ用のセクションであり、*RESET* シンボル値で定義したアドレスにプログラムコードを配置するための定義です。

次に、実際のセクション定義の一部について説明します。

```
:
:
.start (START_OF_RAM) :
       AT(START_OF_ROM)
{
  __start_start = .;
  *(.start)
  __start_end = .;
}
__rom_start = LOADADDR(.start);
.init (ADDR(.start)+SIZEOF(.start)) :
      AT( LOADADDR(.start) + SIZEOF(.start) )

{
  __init_start = .;
  *(.init)
  __init_end = .;
} =0
:
:
```

この定義は、.startセクションと.initセクションを定義している部分です。

.startセクションでは、実行時アドレスが *START_OF_RAM* シンボルで定義されたアドレス、ロード時アドレスが *START_OF_ROM* シンボルで定義されたアドレスに配置することを定義しています。

次に、この.startセクションの実行時の開始シンボルとして*__start_start* を定義し、この時のロケーションカウンタの値 (.) を設定しています。

*(.start)は、全ての入力ファイル (*) の.startというセクションを配置することを定義しています。

そして、.startセクションの実行時の終了シンボルとして*__start_end* シンボルを定義し、この時のロケーションカウンタの値 (.) を指定しています。

__rom_start=LOADADDR(.start) は、*__rom_start* シンボルを定義し、このシンボルに.startセクションのロード時開始アドレスを設定しています。

.initセクションは、実行アドレスが、.startセクションの実行時アドレスと.startセクションのサイズを加算したアドレス位置から開始することを定義し、ロードアドレスが、.start セクションのロード時アドレスと.startセクションのサイズを加算したアドレス位置から開始することを定義しています。

__init_start=.と*__init_end*=.の定義は、.startセクションの定義の場合と同じく、.initセクションの実行時の開始アドレスと終了アドレスを定義しています。

最後の}=0は、セクションがアライメント調整のためにパディングされた場合、データとして 0 を埋め込むことを定義しています。

4.4 スタートアッププログラム処理概要

サンプルのスタートアッププログラム (crt0.S) では、次の処理行っています。

- PSW初期値の設定
  サンプルでは、初期値として 0x80(割り込み禁止、ユーザー用スタックポインタを使用) を設定しています。
- 割り込みスタック (SPI) ポインタの初期値の設定
  'crt0.S'で確保した領域を初期値として設定しています。
- ユーザースタック (SPU) ポインタの初期値の設定
  'crt0.S'で確保した領域を初期値として設定しています。
- 戻り番地保持レジスタの初期値の設定
  R14レジスタに初期値として 0 を設定しています。
- 初期値なしデータ領域の 0 クリア
  .sbssセクションと.bssセクションを 0 クリアします。.bssセクションのクリア例を以下に示します。

```
; (.bss)
        seth    r3, #high(__bss_start)
        or3     r3, r3, #low(__bss_start) ; r3: start of
                                          ;     .bss section
        seth    r4, #high(__bss_end)
        or3     r4, r4, #low(__bss_end)   ; r4: end of .bss
                                          ;      section + 1
        sub     r4, r3                    ; r4: .sbss section
                                          ;       size(bytes)
        blez    r4, .Lendloop_bss         ; if size <= 0
                                          ;       break
        ldi     r2, #0
.Lloop_bss:
        stb     r2, 3
        addi    r3, #1
        addi    r4, #-1
        bgtz    r4, .Lloop_bss
.Lendloop_bss:
```

  ただし、0 クリアのために参照しているグローバルシンボルは、サンプルのリンカスクリプト内で定義されたシンボルを使用しています。従って、この処理はリンカスクリプトの定義と整合をとる必要があります。
- 各セクションの転送処理
  ロード時のアドレスから実行時のアドレスへ各セクションの内容を転送しています。一般には、ROM から RAM への転送あるいは低速 RAM から高速 RAM へ転送するために、

この処理を行います。
ここでは、以下のセクションの転送処理を行っています。

.start	スタートアッププログラムのセクション
.init	C および C++用初期化処理プログラムのセクション
.text	コードのセクション
.fini	C および C++用終了処理プログラムのセクション
.rodata	初期値あり読み出し専用データのセクション
.data	初期値あり読み書きデータのセクション
.ctors	C++用グローバルコンストラクタテーブルのセクション
.dtors	C++用グローバルデストラクタテーブルのセクション
.sdata	初期値あり *small data area* データのセクション

.textセクションの転送処理の例を以下に示します。

```
        ; copy .text section to RAM
        seth    r2, #high(__rom_text)
        or3     r2, r2, #low(__rom_text)        ; r2: src address
        seth    r3, #high(__text_start)
        or3     r3, r3, #low(__text_start)      ; r3: dest address
        beq     r2, r3, .Lendloop_text
        seth    r4, #high(__text_end)
        or3     r4, r4, #low(__text_end)        ; r4: end of text + 1
        sub     r4, r3                  ; r4: text size(bytes)
        blez    r4, .Lendloop_text      ; if size <= 0 break
.Lloop_text:
        ldb     r1, 2
        stb     r1, 3
        addi    r2, #1
        addi    r3, #1
        addi    r4, #-1
        bgtz    r4, .Lloop_text
.Lendloop_text:
```

ただし、転送のために参照しているグローバルシンボルは、サンプルのリンカスクリプト内で定義されたシンボルを使用しています。従って、この処理はリンカスクリプトの定義と整合をとる必要があります。

- main関数へ引数 *argv* の設定
  main関数への引数 *argv* の値として 0 を設定しています。
- main関数または_c_main関数の呼び出し
  三菱製 ANSI C 標準ライブラリを使用する場合、_c_main関数 (標準ライブラリの初期化関数) を呼び出します。_c_main 関数を呼び出した場合、main関数は、_c_main関数内から呼び出されます。
  標準ライブラリを使用しない場合、main関数を直接呼び出します。

- exit関数の呼び出し
  exit関数を呼び出しプログラムを終了します。

4.5 スタートアッププログラムのマクロ定義

サンプルのスタートアッププログラムでは、次のマクロを用意しています。

TEXT_IN_RAM
: プログラムコードの実行時アドレスとロードアドレスが異なる場合、このマクロを定義します。
このマクロが定義されている場合、プログラムコードをロードアドレスから実行時アドレスへ転送後、main関数(あるいは_c_main関数)を絶対アドレスで呼び出します。
定義されていない場合、プログラムコードは転送されません。また、main関数(あるいは_c_main関数)を相対アドレスで呼び出します。

NOUSE_LIB
: 三菱標準ライブラリを使用しない場合、このマクロを定義します。
このマクロが定義されている場合、直接main関数が呼び出されます。定義されていない場合、標準ライブラリの初期化関数_c_main関数が呼び出されます。

USE_RESET_VECTOR
: リセットベクタのコードを使用する場合、このマクロを定義します。デフォルトではリセットベクタを使用するように、このマクロは有効になっています。

4.6 スタートアッププログラムでのデータ領域の確保

サンプルのスタートアッププログラム内では、以下のデータ領域を確保します。

割り込みスタック領域
: 割り込みスタック領域を確保します。
必要な割り込みスタックのサイズを *spi_stack_size* シンボル値にバイトで定義します。スタックサイズ値には 4 の整数倍を指定してください。デフォルトでは、8K バイトを確保しています。

```
.section    .spi_stack, "aw", @nobits
.set        spi_stack_size, 0x2000 /* 8Kbytes */
.balign 4
.space      spi_stack_size
```

ユーザースタック領域
: ユーザースタック領域を確保します。
必要なユーザースタックのサイズを *spu_stack_size* シンボル値にバイトで定義します。スタックサイズ値には 4 の整数倍を指定してください。デフォルトでは、8K バイトを確保しています。

```
    .section    .spu_stack, "aw", @nobits
    .set        spu_stack_size, 0x2000 /* 8Kbytes */
    .balign 4
    .space      spu_stack_size
```

ヒープ領域

三菱製 ANSI C 標準ライブラリで使用するヒープ領域を確保します。

必要なヒープ領域のサイズを *heap_size* シンボル値に定義します。ヒープサイズ値には 4 の整数倍を指定してください。また、必要なヒープサイズは、ユーザープログラムで入出力関数 (printf など) 使用時にバッファを使用するかどうか (バッファリングモード時)、メモリ操作関数を使用するかどうかに依存します。
入出力関数でバッファを使用する場合、1 つの FILE 構造体につき 1036 バイト使用します。したがって、使用する FILE 構造体数 (ユーザープログラムで同時オープンするファイルの最大数 + 標準入力 + 標準出力 + 標準エラー出力) × 1036 バイト以上が必要になります。デフォルトでは、100K バイトを確保しています。

```
    .section    .heap, "aw", @nobits
    .set        heap_size, 0x19000 /* 100Kbytes */
    .balign 4
    .space      heap_size
```

main 関数への引数 *argv*

main関数への引数 *argv* を設定します。デフォルトでは、*argv* として 0 を設定しています。

```
environ:
.long 0
```

技術サポート連絡書

年　月　日（合計　　枚）

三菱電機セミコンダクタシステム株式会社
マイコンソフトツール部
開発ツールサポート窓口行

［電子メール］ support@tool.mesc.co.jp
［大阪地区］ FAX： 06-6398-6191
［東京地区］ FAX： 03-5783-7339
［中部地区］ FAX： 052-221-7318
［九州地区］ FAX： 092-452-1427

※インストーラが生成する以下のテキストファイルもサポート連絡書としてご利用できます。
Windows 98/95/Windows NT 4.0版の場合：\SUPPORT\製品名\SUPPORT.TXT
EWS版の場合：/support/製品名/toolinfo.txt

ご連絡先	製品情報
会社名：	ソフトウェア：
部署名：	バージョン番号：V.
担当者名：	ライセンスID：
電話番号：	– – – –
FAX番号：	ホストマシン：
電子メール：	OS： V.
通信欄：	

MEMO

TW32R V.3.10ユーザーズマニュアル

第1版：2000年6月1日発行
資料番号：MSD-TW32R-U-000601

三菱電機株式会社
三菱電機セミコンダクタシステム株式会社

TW32R V.3.10 ユーザーズマニュアル
M32R ファミリ用　クロスツールキット (GNU 版)

ルネサス エレクトロニクス株式会社
神奈川県川崎市中原区下沼部1753 〒211-8668