ルネサス統合開発環境
High-performance Embedded Workshop
操作マニュアル
応用編

第 1.22 版
2009.02.09
ジャパンマイコンカーラリー実行委員会

注 意 事 項 (rev.1.3)

著作権

・本マニュアルに関する著作権はジャパンマイコンカーラリー実行委員会に帰属します。
・本マニュアルは著作権法および、国際著作権条約により保護されています。

禁止事項

ユーザーは以下の内容を行うことはできません。
・第三者に対して、本マニュアルを販売、販売を目的とした宣伝、使用、営業、複製などを行うこと
・第三者に対して、本マニュアルの使用権を譲渡または再承諾すること
・本マニュアルの一部または全部を改変、除去すること
・本マニュアルを無許可で翻訳すること
・本マニュアルの内容を使用しての、人命や人体に危害を及ぼす恐れのある用途での使用

転載、複製

本マニュアルの転載、複製については、文章によるジャパンマイコンカーラリー実行委員会の事前の承諾が必要です。

責任の制限

本マニュアルに記載した情報は、正確を期すため、慎重に制作したものですが万一本マニュアルの記述誤りに起因する損害が生じた場合でも、ジャパンマイコンカーラリー実行委員会はその責任を負いません。

その他

本マニュアルに記載の情報は本マニュアル発行時点のものであり、ジャパンマイコンカーラリー実行委員会は、予告なしに、本マニュアルに記載した情報または仕様を変更することがあります。製作に当たりましては、事前にマイコンカー公式ホームページ(http://www.mcr.gr.jp/)などを通じて公開される情報に常にご注意ください。

連絡先

ルネサステクノロジ　マイコンカーラリー事務局
〒162-0824　東京都新宿区揚場町 2-1　軽子坂MNビル
TEL (03)-3266-8510
E-mail:official@mcr.gr.jp

目 次

1. ビルド 1
1.1 オブジェクトファイルの作成 1
1.2 リンク 3
1.3 MOT ファイル、ABS ファイル 4
1.4 まとめ 5
2. ツールチェインの設定 6
2.1 コンパイラの設定 6
2.1.1 RY3048Fone ボードの場合（H8/3048F-ONE） 8
2.1.2 RY3687 ボードの場合（H8/3687F） 9
2.1.3 その他の CPU 9
2.2 アセンブラの設定 10
2.3 最適化リンカの設定 10
2.3.1 RY3048Fone ボードの場合（H8/3048F-ONE） 11
2.3.2 RY3687 ボードの場合（H8/3687F） 11
2.3.3 その他の CPU の場合 11
2.4 標準ライブラリの設定 12
2.5 CPU の設定 13
2.5.1 RY3048Fone ボードの場合（H8/300H シリーズの CPU） 13
2.5.2 RY3687 ボードの場合（H8/300H Tiny シリーズの CPU） 13
2.6 ツールチェインとプロジェクトの関係 13
3. セクション 14
3.1 セクションの種類 14
3.1.1 C ソースファイル 14
3.1.2 アセンブラリソースファイル 14
3.2 プログラムをセクションに分類 15
3.2.1 section.c をコンパイルしてセクションごとに分類 15
3.2.2 car_printf_3048.c をコンパイルしてセクションごとに分類 16
3.2.3 initsct_3048.c をコンパイルしてセクションごとに分類 16
3.2.4 sectionstart.src をアセンブルしてセクションごとに分類 17
3.2.5 まとめ 19
3.3 セクションの割り当てアドレス 20
3.3.1 セクション D について 20
3.3.2 セクション B について 21
3.4 実際のアドレス 21
3.4.1 RY3048Fone ボードの場合（H8/3048F-ONE） 22
3.4.2 RY3687 ボードの場合（H8/3687F） 24
3.5 ツールチェインでのセクション設定 26
3.6 セクション D,R,B を初期化するプログラム 29
3.7 セクションのまとめ 30
4. 実行委員会開発環境のプログラムをルネサス統合開発環境へ移植する 31
4.1 新しいワークスペースを作成する 31
4.2 ファイルを登録する前設定 34
4.3 ファイルの登録 36

4.4 ツールチェインの設定 38
4.5 ビルド 39

5. プログラム改造例 40

5.1 定数を追加したときの設定(セクション C の追加) 40
5.1.1 追加内容 40
5.1.2 設定内容 41
5.2 初期値のないグローバル変数を追加したときの設定(セクション B の追加) 42
5.2.1 追加内容 42
5.2.2 設定内容 43
5.3 初期値のあるグローバル変数を追加したときの設定(セクション D,R の追加) 47
5.3.1 追加内容 47
5.3.2 設定内容 47
5.4 算術関数(標準ライブラリ関数)の使用 48
5.4.1 追加内容 48
5.4.2 設定内容 50
5.4.3 実行 54

6. エラー対策 55

6.1 L1200 ライブラリファイルのバックアップ 55
6.2 L2310 未定義のシンボル(標準ライブラリ関数) 55
6.3 L2310 未定義のシンボル(自作の関数) 57
6.4 L1120 セクションの指定がない 59
6.5 L1100 不要なセクションがある 60
6.6 電源を切ってすぐに入れるとマイコンカー(または制御プログラムが)が暴走する 61
6.7 ワークスペースが開けない 61

7. 補足 62

7.1 ヘッダファイルの取り扱い 62
7.2 型の範囲 64
7.3 三角関数を使ったプログラムの実行速度 65
7.4 ツールチェインの設定をサイズ優先設定にする 66
7.4.1 ツールチェインの設定 66
7.4.2 サイズと実行スピード 68
7.4.3 まとめ 69
7.5 Cソースファイルがアセンブリソースファイルに変換されたリストを見る 69
7.5.1 ツールチェインの設定 70
7.5.2 リストの表示 72
7.6 プログラム表示のフォントのサイズ変更 74

8. FDT(ルネサスマイコン書き込みソフト)のインストール 75

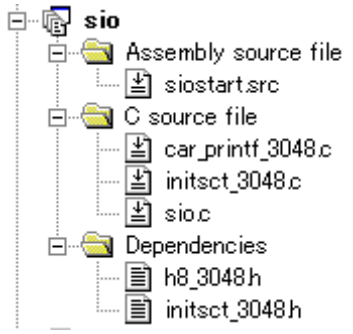
8.1 概要 75
8.2 CD から取得する 75
8.3 ホームページから取得する 76
8.4 インストールする 79
8.5 ショートカットを作成する 82
8.6 書き込み(最初に起動したとき) 82
8.7 2 回目以降の画面 85

9. 参考文献 86

1. ビルド

C 言語ソースファイルやアセンブリソースファイル（要はプログラム）を、CPU が理解できる機械語に変換することを**ビルド**といいます。ルネサス統合開発環境がどのような流れでビルド作業を行い、最終的な MOT ファイルを作るのか説明していきます。

1.1 オブジェクトファイルの作成

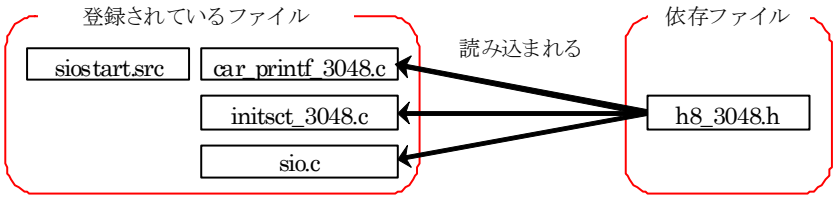


ワークスペース「h8_3048」のプロジェクト「sio」を例に、説明します。

プロジェクト「sio」を有効なプロジェクトにすると左のようなファイルリストが現れます。

・「siostart.src」というアセンブリソースファイル
・「car_printf_3048.c」というCソースファイル
・「initsct_3048.c」という C ソースファイル
・「sio.c」という C ソースファイル

があります。依存関係として「h8_3048.h」というファイルが、上記 4 ファイルのどれかから include 命令で呼ばれています。実際には、「car_printf_3048.c」と「initsct_3048.c」と「sio.c」ファイルから呼ばれています（下図）。



ビルドを実行するとまず、これらの 4 ファイルを**オブジェクトファイルと呼ばれるファイルに変換（翻訳）します。オブジェクトファイルとは、機械語の形になったファイルのことです。**

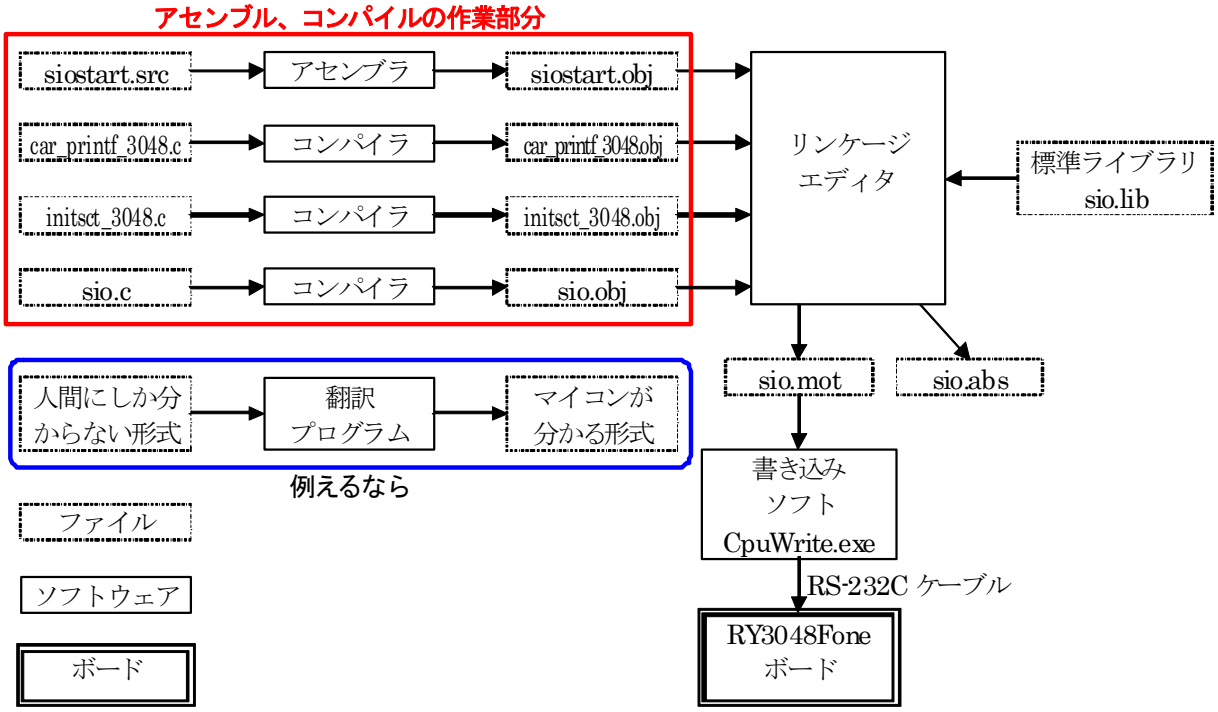
ソースファイルをオブジェクトファイルに変換する作業を**アセンブル**や**コンパイル**といいます。変換元のファイルの種類によって名称が異なります。

・アセンブリソースファイルを変換するときはアセンブルすると言います。
・Cソースファイルを変換するときはコンパイルすると言います。

表にまとめてみました。

	アセンブリソースファイル	C ソースファイル
ファイルの内容	機械語と 1 対 1 の関係にある言語が記述されたファイルです。	C 言語という高級言語で記述されたファイルです。パスカル、フォートラン、BASIC 言語も高級言語です。
オブジェクトファイルに変換する翻訳プログラム	アセンブリソースファイルを機械語ファイル（オブジェクトファイル）に変換する翻訳プログラムをアセンブラと言います。アセンブラで変換する作業をアセンブルすると言います。	高級言語（今回は C ソースファイル）を機械語ファイル（オブジェクトファイル）に変換する翻訳プログラムをコンパイラと言います。コンパイラで変換する作業をコンパイルすると言います。
機械語との対応	例えば、「mov.b #0, r01」をアセンブルすると「F8 00」という機械語に翻訳されます。アセンブリソースプログラムは、命令と対応する機械語が決まっています。	例えば、「a=0;」をコンパイルすると、機械語に変換されますが、どのような機械語に変換されるかは変換結果を見なければ分かりません。それは、「a=0」という命令に直接対応する機械語がないためです。一般的に高級言語1命令は、複数の機械語に変換されます。
移植（違うシステムに書き換えること）	例えば、Z80 のアセンブラで「mov.b #0, r01」をアセンブルするとエラーになります。アセンブリ言語はそれぞれの CPU によって違う命令です。移植できません。	例えば、Z80 の C コンパイラで「a=0;」をコンパイルすると、Z80 の機械語命令に書き換わります。これが高級言語です。

□部分がアセンブル、コンパイルの作業部分です。



・アセンブラ（翻訳プログラム）がアセンブル（行為）をして、オブジェクトファイルができます。
・コンパイラ（翻訳プログラム）がコンパイル（行為）をして、オブジェクトファイルができます。
　サッカーに例えると、
・プレーヤ（人）がプレー（行為）をして、ゴールします。
と言えます。

4 つのオブジェクトファイルができました。

※C ソースファイルを分ける理由

今回の例では C ソースファイルは、「sio.c」、「car_printf_3048.c」、「initsct_3048.c」の 3 つに分かれています。なぜ複数の C ソースファイルを登録するのでしょうか。1 つの C ソースファイルにまとめてはいけないのでしょうか。
絶対にまとめてはいけないと言うことは無いのですが、下記の理由によりファイルを分けることが一般的です。

・機能ごとに C ソースファイルを分ける

例えば「eeprom.c」を作り、EEP-ROM に関するプログラムだけを入れておきます。EEP-ROM に関するプログラムを改造したい場合は「eeprom.c」ファイルを開いて、その中からプログラムを探せば良いことになります。ファイルが1つしかなく長いプログラムだと、修正する部分を探すのが大変です。

・コンパイル時間を短くする

「ビルド」コマンドは、毎回すべてのソースファイルをアセンブルやコンパイルしている訳ではありません。ビルドを1回して再度ビルドを行うとき、ソースファイルの更新があったファイルのみアセンブルやコンパイルをしています（「すべてをビルド」コマンドは更新に関係なく全ソースファイルをアセンブルやコンパイルします）。
例えば、10,000 行の C ソースファイルが 1 ファイルあった場合、内容の一部を更新してコンパイルすると 10 秒かかりました。これを 5,000 行の C ソースファイル 2 ファイルに分割して 1 ファイルのみ更新した場合、コンパイルすると 5 秒ですみます（これは例えで実際は違います）。このように、コンパイルの時間を短くすることができます。

1.2　リンク

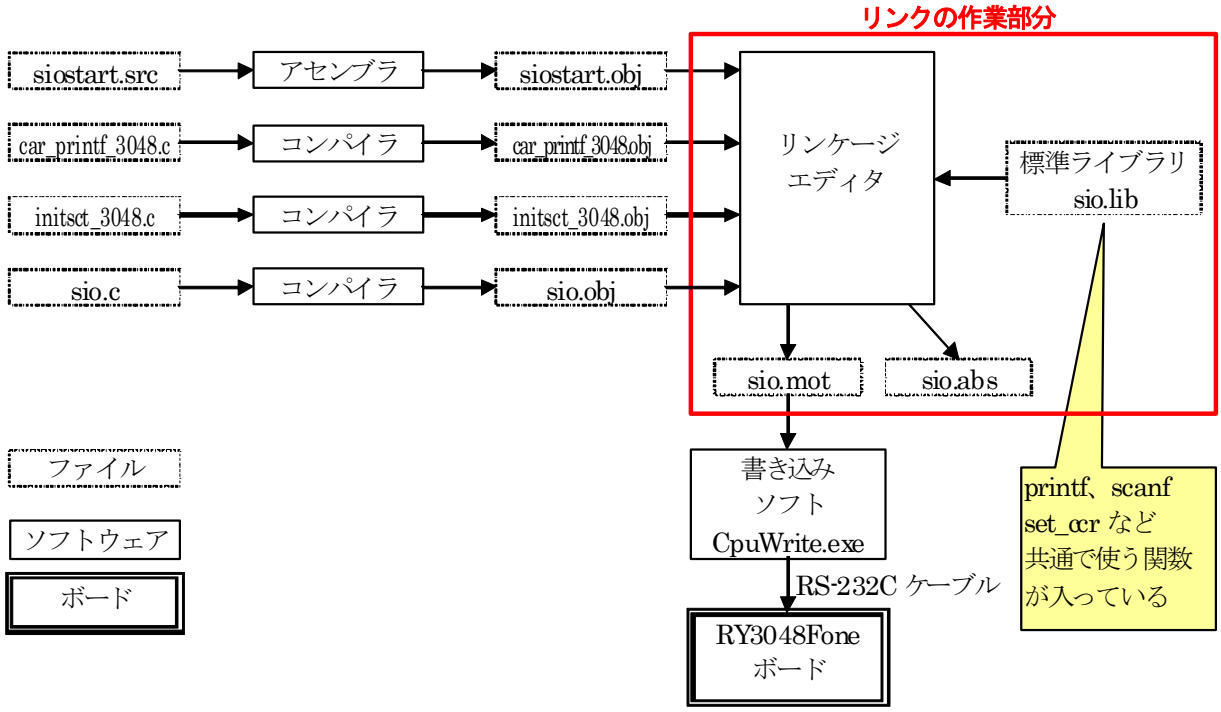
次に、**複数のオブジェクトファイル（今回は 4 ファイル）を 1 つに合体させ、マイコンに書き込む最終的な形式にします**。これを**リンク**といいます。

ただ合体させるだけかというと、他にも重要な役割があります。sio.c では「printf 文」を使用しています。オブジェクトファイル「sio.obj」には、「printf 関数へ飛びなさい」という命令が機械語に変換されています。ただし、printf 関数そのものは入っていません。そのため、リンケージエディタは printf 関数本体を組み込みます。printf 文など、基本的な動作を受け持つ関数がまとめられたファイルがあります。これを「**標準ライブラリ**」といいます。例えば sio プロジェクトの場合、「C:¥Workspace¥h8_3048¥sio¥Debug」フォルダに「sio.lib」というファイルがあります。これが標準ライブラリファイルです。ライブラリファイルが無い場合は、sio プロジェクトのビルドを行ってください。作成されます。

複数のオブジェクトファイルを合体させ、基本的な動作が入っている「標準ライブラリ」を組み込む作業をリンクするといい、その作業を行うソフトウェア（機能）がリンケージエディタなのです。

リンクすると「sio.mot」と「sio.abs」ファイルが作成されます。

下図にその様子を示します。　□部分がリンク部分です。



ここまでがビルドで行われる作業になります。

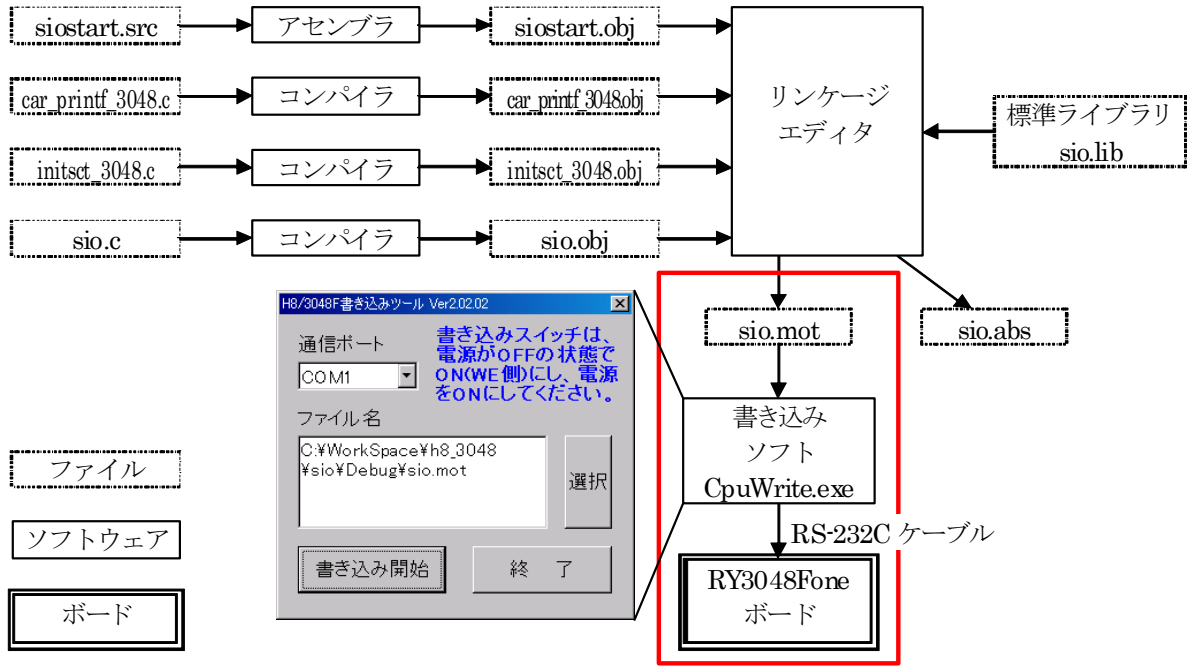
1.3　MOTファイル、ABSファイル

ビルドが正常に終わると、「sio.mot」と「sio.abs」の 2 ファイルが作成されます。これらのファイルは、下記のような仕様になっています。

名称／ ファイル名	解　説
S タイプ ファイル sio.mot	Sタイプというのは、ファイルの中身を開くと、Sという文字から始まっているのでSタイプと呼びます。モトローラ社が作成した形式なので、モトローラ形式とも呼びます。社名「Motorola」から拡張子「MOT」となります。 ファイルの中身を見てみると、例えば S1130100010O6DF6189939BA38B80D8139B97906F4 となっています。これを分解すると、 S1　13　0100　01006DF6189939BA38B80D8139B97906　F4 1　2　3　4　5 となります。 1…形式を表しています。この場合は S1 形式です。S2 や S3 などがあります。 2…データの個数を表しています。16 進数表記なので、0x13→19 個データがあるということです。13 以降のデータは 2 文字で1バイトの数値データです。 01　00　01　00　6D　F6　18　99　39　BA　38　B8　0D　81　39　B9　79　06　F4 1　2　3　4　5　6　7　8　9　10　11　12　13　14　15　16　17　18　19 確かに 19 個あります。 3…アドレスを示しています。0x100 番地からデータを書き込みなさいという意味です。 4…データです。0x0100 番地から順に割り当てます。 0x100 番地…01　0x101 番地…00 0x102 番地…6D　というように割り当たります。 5…チェック用で、各バイトの合計の1の補数です。 このデータを書き込みソフト（CpuWrite.exe など）が読み込み、マイコンの内蔵 ROM にプログラムを書き込みます。詳しくは、インターネットで「モトローラ S フォーマット」と検索してみてください。
アブソリュート ファイル sio.abs	デバッグ装置で使用する形式です。デバッグ装置を使用しない場合は使いません。右写真は、E10T というデバッグ装置です。 

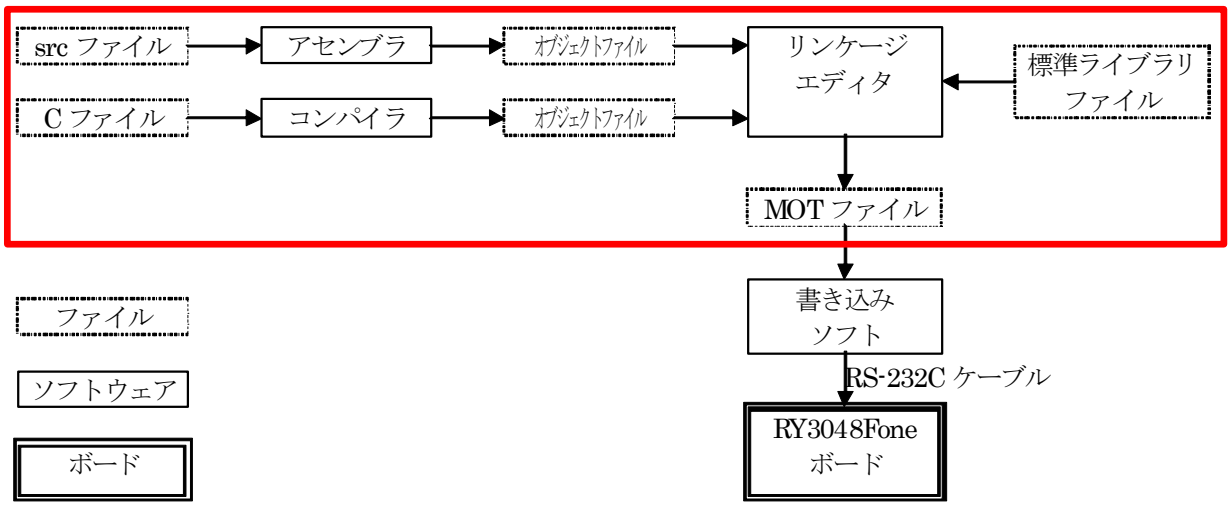
「sio.mot」ファイル、「sio.abs」ファイルの作成までが、ビルドで行う作業です。

ちなみに、書き込みソフト(CpuWrite.exe など)が「sio.mot」ファイルを読み込み、マイコンの内蔵 ROM にプログラムを書き込みます。下図にその様子を示します。 □部分が書き込み作業部分です。



1.4　まとめ

ビルドで行う作業をまとめると、下図の□部分になります。



MOT ファイル作成までの一連の作業を、ルネサス統合開発環境では「**ビルド**」という操作で行います。

ビルド＝アセンブル＋コンパイル＋リンク

ということになります。ビルドでは、書き込みは行いません。

実行委員会開発環境では、「コンパイル」というボタンがありましたが、実はコンパイルだけではなく、アセンブルやリンクも行っていました。厳密には実行委員会開発環境の「コンパイル」ボタンは「ビルド」ボタンと言えます。

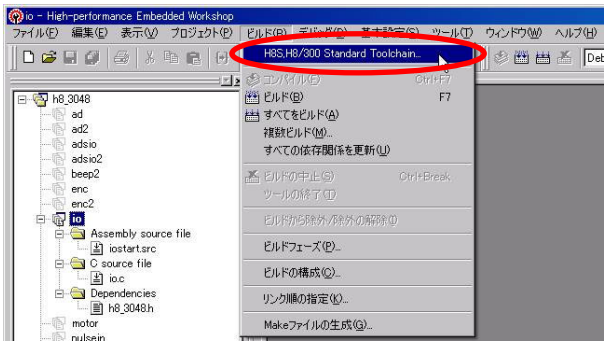
2. ツールチェインの設定

ツールチェインという言葉は馴染みがないと思います。要は、**ビルドするときの設定**のことです。実行委員会開発環境には sub ファイルと呼ばれるファイルがありました。io.sub は下記のような内容です。

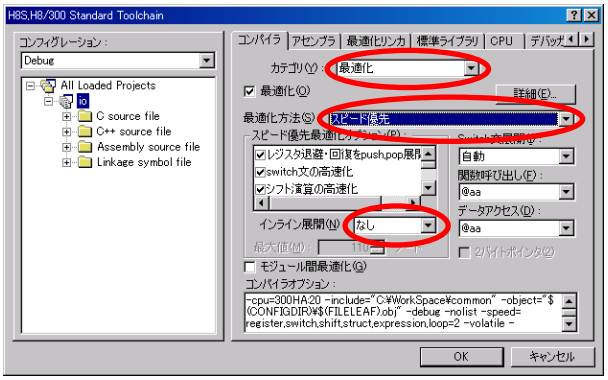
```
input iostart,io
lib c:¥h8n_win¥3048¥c¥c38hae.lib
output io
print io
start V(000000)
start P,C(000100)
start B(0fef10)
exit
```

この sub ファイルに当たる部分を、ルネサス統合開発環境ではツールチェインの設定で行います。ここでは、その設定内容を説明します。

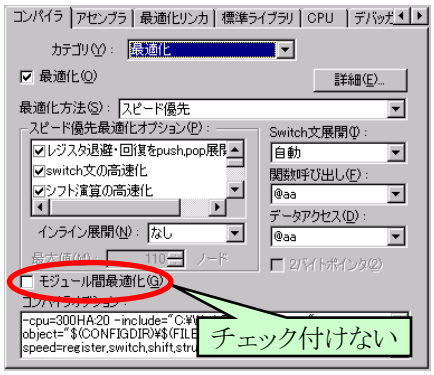
2.1 コンパイラの設定



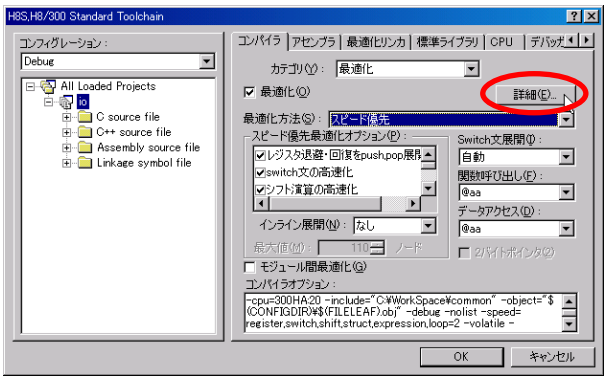
1.「ビルド→H8S,H8/300 Standard Toolchain」(ツールチェイン)を選択します。



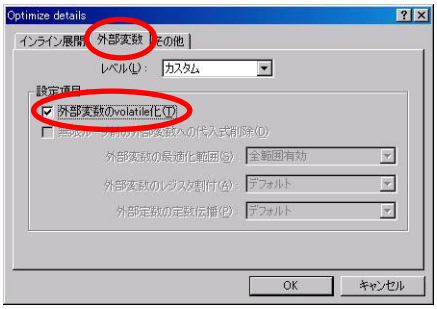
2.「カテゴリ:**最適化**」、「最適化方法:**スピード優先**」、「インライン展開:**なし**」を選択します。



3.「モジュール間最適化」のチェックは**付けません**。最適化という文字に惑わされてチェックを付けると良くなりそうですが、チェックをつけるとプログラムが暴走することがあります(チェックをつけた場合は、さらに設定が必要です)。

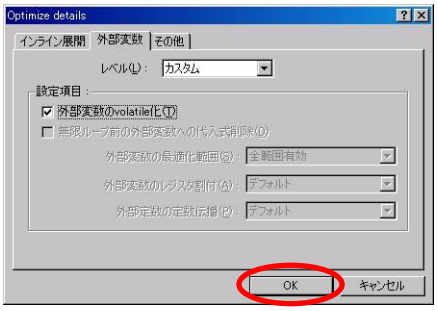


4.詳細をクリックします。



5.「外部変数の volatile 化」のチェックを付けます。

※volataile についての詳しい説明は下記を参照してください。



6.OK をクリックします。

※参考資料－volatile について

コンパイラはプログラムをコンパイルするとき最適化と呼ばれる作業を行います。これは、無駄な部分を省いてプログラムのサイズを小さく、実行速度を速くするようにします。

例えば、下記のプログラムを作ったとします。コンパイルするとどうなるでしょうか。

```
PADR = 0x55;
PADR = 0xaa;
PADR = 0x00;
```

コンパイル →

```
PADR = 0x00;
```

ポート A にデータを3回連続して書き込んでします。コンパイラは、同じ場所に連続して値を書き込んでいるので、最終的な結果は PADR に 0x00 を書き込めばいいだろう、と勝手に解釈してしまいます。本来は、ポート A に出力する信号を連続して変化させたいのですが、そうなりません。

そこで、コンパイルのオプションで「外部変数の volatile 化」にチェックを付けます。**このチェックを付けることにより最適化を行わないようにします**。

```
PADR = 0x55;
PADR = 0xaa;
PADR = 0x00;
```

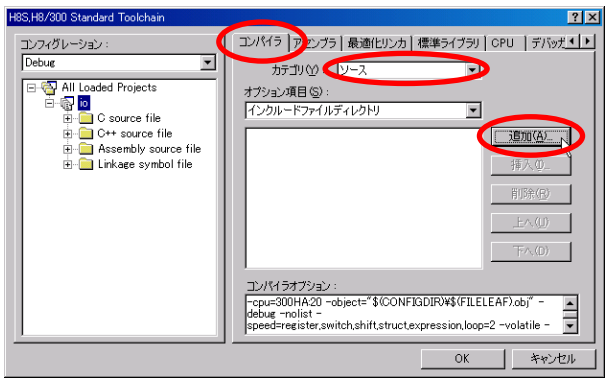
コンパイル →

```
PADR = 0x55;
PADR = 0xaa;
PADR = 0x00;
```

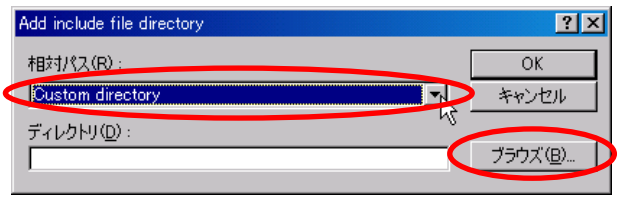
と、プログラムしたとおりの動きになります。

次に共通フォルダの設定をします。CPU の種類により、設定するフォルダが異なります。

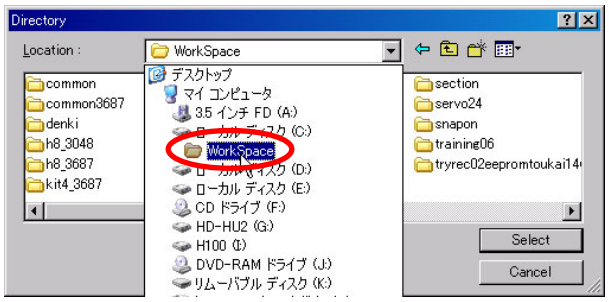
2.1.1　RY3048Foneボードの場合（H8/3048F-ONE）



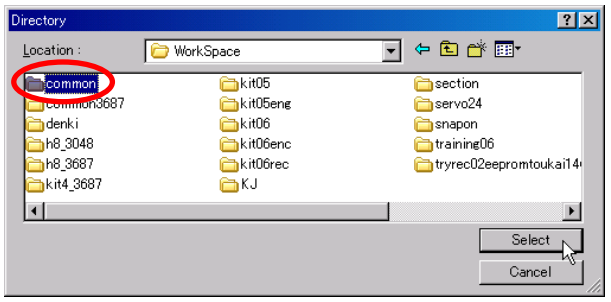
1.「コンパイラ」を選択します。「カテゴリ:**ソース**」、追加をクリックします。



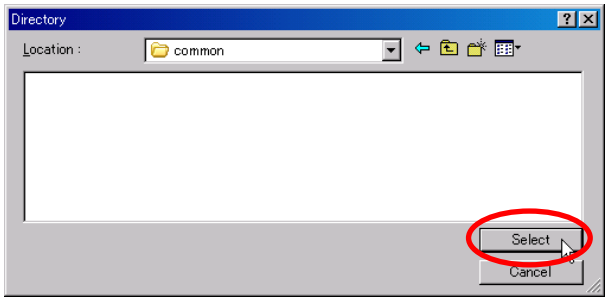
2.「相対パス:**Custom directory**」にします。続けて、ブラウズをクリックします。



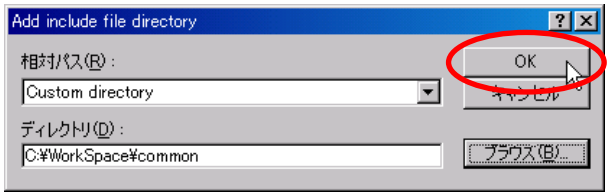
3.「C ドライブの WorkSpace」フォルダを選択します。



4.「common」フォルダをダブルクリックします。

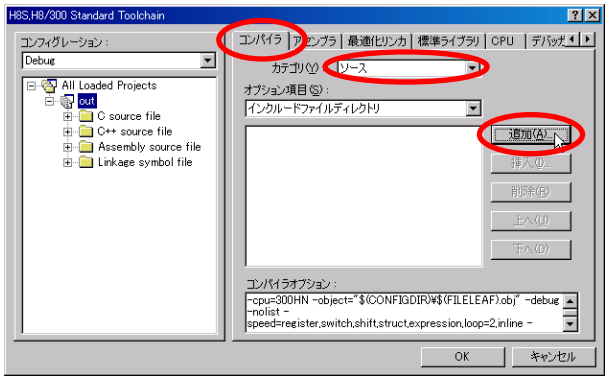


5.Select をクリックします。



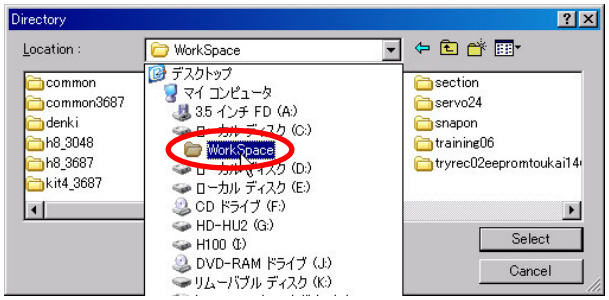
6.OK をクリックします。

2.1.2　RY3687 ボードの場合（H8/3687F）

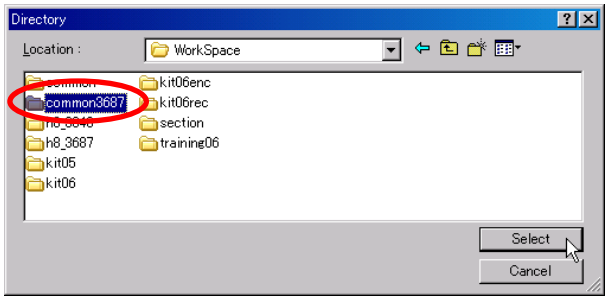


1.「コンパイラ」を選択します。「カテゴリ：**ソース**」、追加をクリックします。

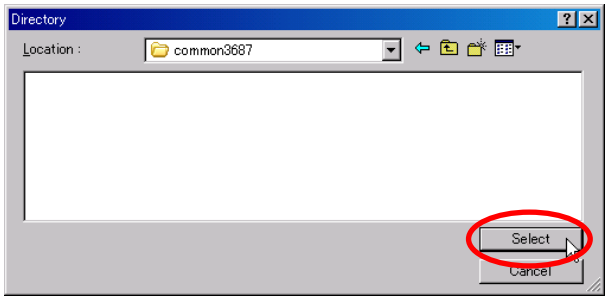
2.「相対パス：**Custom directory**」にします。続けて、ブラウズをクリックします。



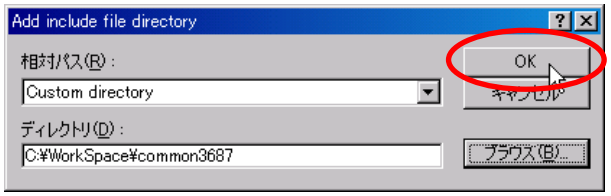
3.「C ドライブの WorkSpace」フォルダを選択します。



4.「common3687」フォルダをダブルクリックします。



5.Select をクリックします。

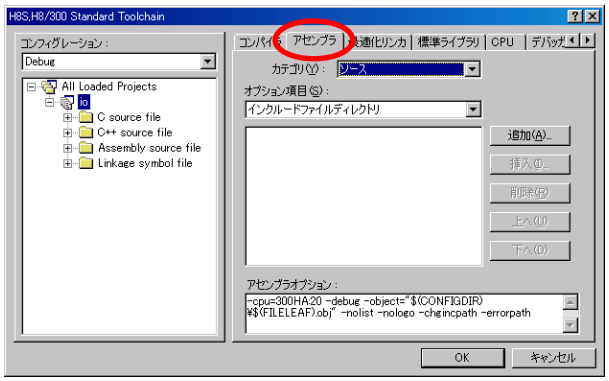


6.OK をクリックします。

2.1.3　その他のCPU

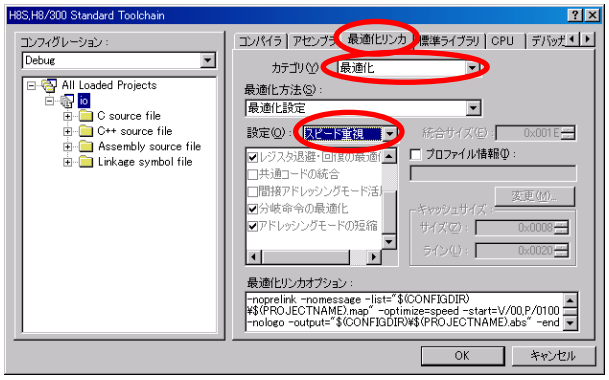
その他の CPU の場合、共通フォルダを用意していません。今回の作業は必要ありません。

2.2　アセンブラの設定

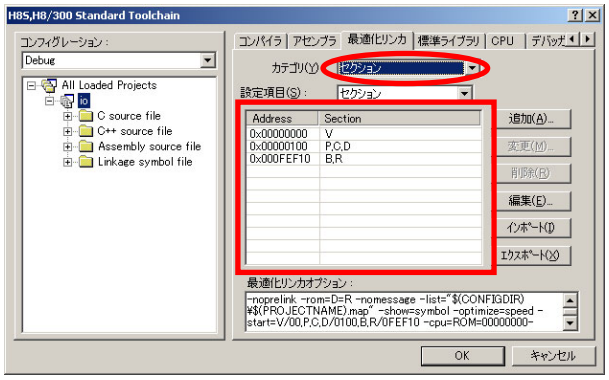


1.アセンブラで設定する項目はありません。

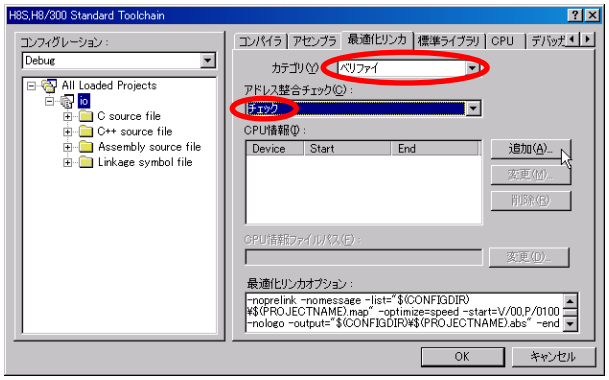
2.3　最適化リンカの設定



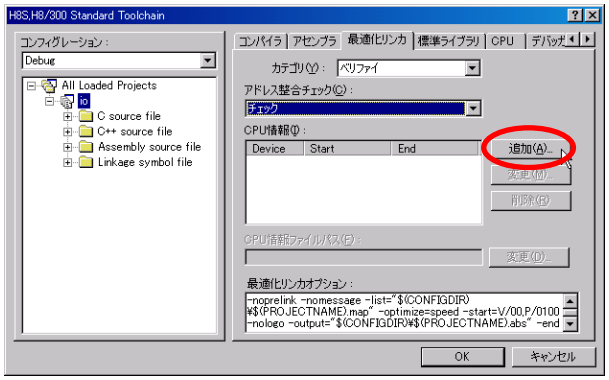
1.「最適化リンカ」を選択します。「カテゴリ：**最適化**」、「設定：**スピード重視**」にします。



2.「カテゴリ：**セクション**」にします。□欄はプログラムに合わせてセクションを設定します。設定については「**3. セクション**」を参照してください。

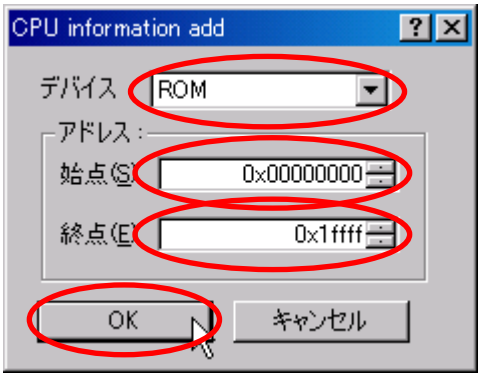


3.「カテゴリ：**ベリファイ**」、「アドレス整合チェック：**チェック**」にします。これは、CPU の無効なアドレスにプログラムを書き込んだときに、警告するための設定です。



4.［追加］をクリックします。
次に ROM の容量と RAM の容量を入力しますが、CPU の種類によってアドレスが変わります。

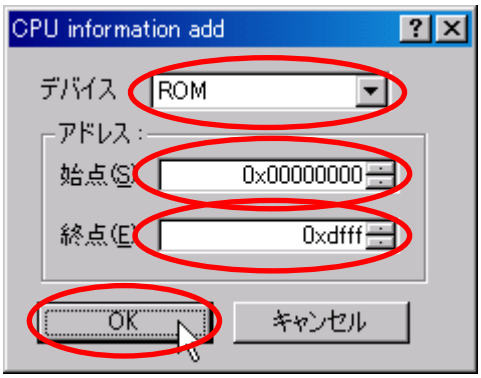
2.3.1　RY3048Foneボードの場合（H8/3048F-ONE）



1.「デバイス:**ROM**」を選択、
「始点:**0x0000**」、「終点:**0x1ffff**」
を入力して OK をクリックします。

2.再度 追加 をクリックします。
「デバイス:**RAM**」を選択、「始点:**0xfef10**」、
「終点:**0xffb0f**」を入力して OK をクリックします。
RAM は 4KB ありますが、1KB はスタックなどで使用しますので、
0xfef10+3KB＝0xffb0f　とします。

2.3.2　RY3687 ボードの場合（H8/3687F）



1.「デバイス:**ROM**」を選択、
「始点:**0x0000**」、「終点:**0xdfff**」
を入力して OK をクリックします。

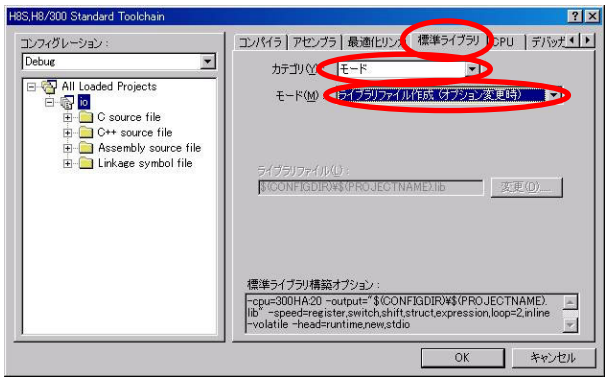
2.再度 追加 をクリックします。
「デバイス:**RAM**」を選択、「始点:**0xe800**」、
「終点:**0xefff**」を入力して OK をクリックします。
RAM は 4KB ありますが、下記のように 2 つに分かれています。
・0xe800～0xefff
・0xf780～0xff7f
そのため、前者を変数エリア、後者をスタックエリアとします。変数エリアがあふれないかチェックすれば良いため、ここでは前者の RAM 領域のみ指定しておきます。

2.3.3　その他のCPUの場合

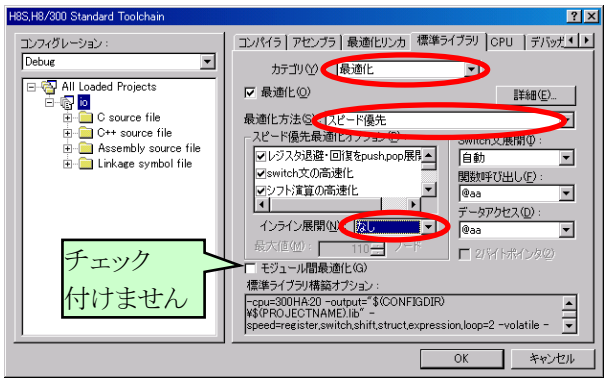
それぞれの CPU に合わせて、ROM と RAM の開始アドレスと終了アドレスを指定してください。番地についてはハードウェアマニュアルの「アドレス空間とメモリマップ」に記載されています。

もし、分からない場合は設定しなくとも良いですが、万が一間違ったアドレスにプログラムを書き込んでしまった場合、エラーメッセージが出ないため、エラーに気づきにくくなります。

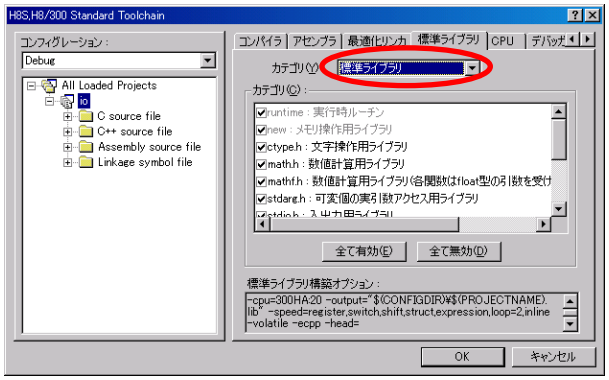
2.4　標準ライブラリの設定



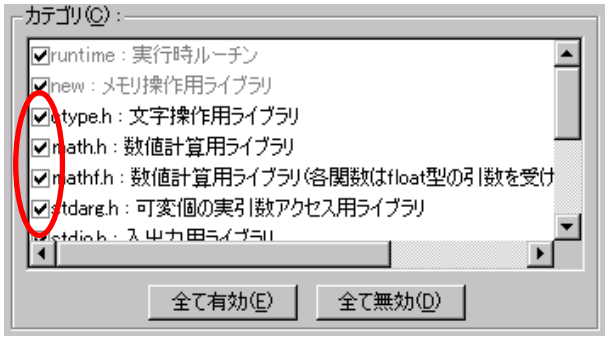
1.「標準ライブラリ」を選択します。「カテゴリ：**モード**」、「モード：**ライブラリファイル作成（オプション変更時）**」にします。



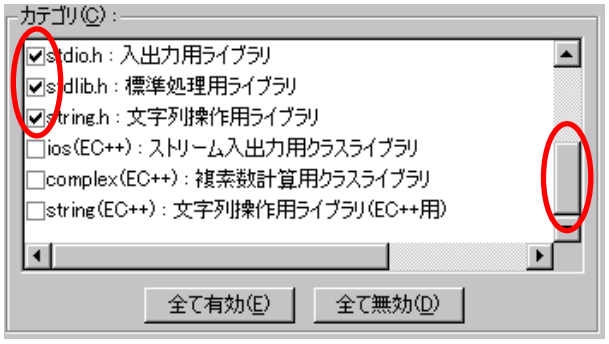
2.「カテゴリ：**最適化**」、「最適化方法：**スピード優先**」、「インライン展開：**なし**」を選択します。「モジュール間最適化」のチェックは**付けません**。



3.「カテゴリ：**標準ライブラリ**」を選択します。



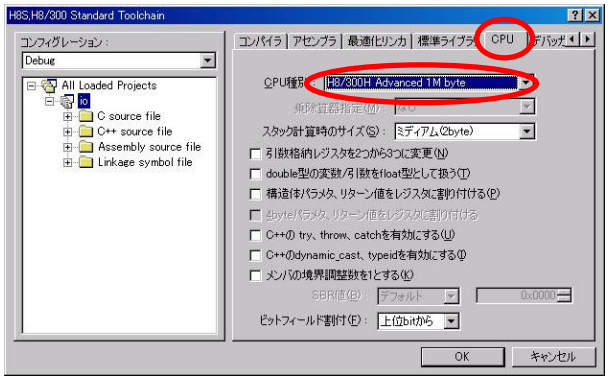
4.カテゴリ欄のファイルにチェックを付けます。



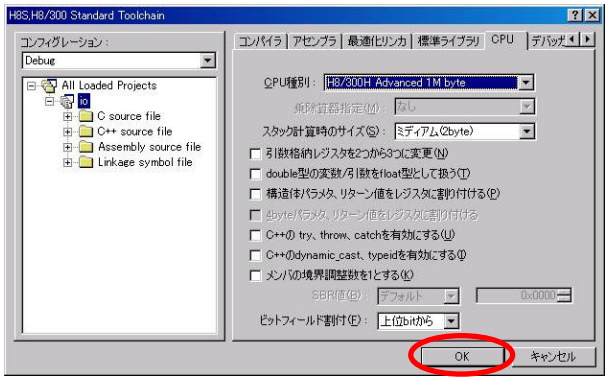
5.スクロールバーを下ろして、更にチェックを付けます。いちばん下の 3 ファイル(EC++と書かれたファイル)にはチェックを付けません。

2.5　CPUの設定

2.5.1　RY3048Foneボードの場合（H8/300Hシリーズの CPU）

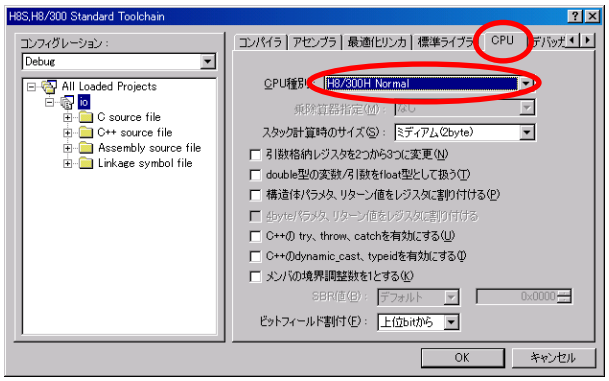


1.「CPU」を選択します。
「CPU　種別：**H8/300H　Advanced　1M　byte**」にします。

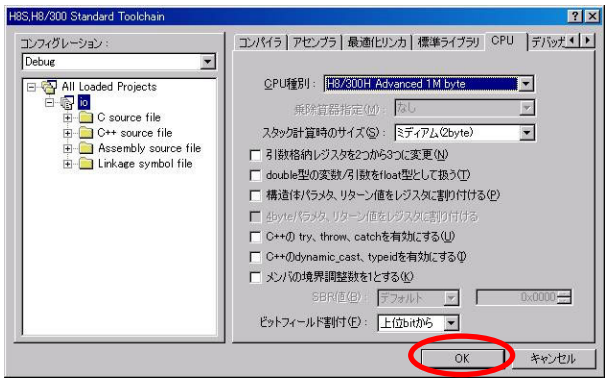


2.OK をクリックして、ツールチェインの設定を完了します。

2.5.2　RY3687 ボードの場合（H8/300H Tinyシリーズの CPU）



1.「CPU」を選択します。「CPU 種別：**H8/300H Normal**」にします。



2.OK をクリックして、ツールチェインの設定を完了します。

2.6　ツールチェインとプロジェクトの関係

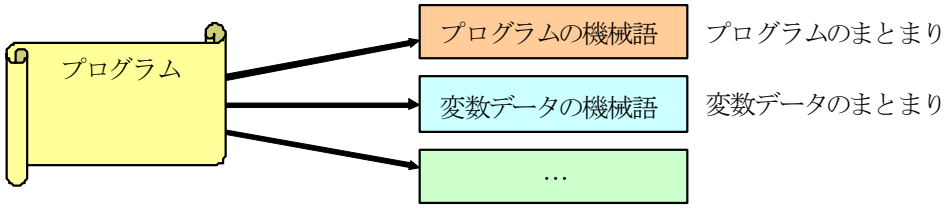
ツールチェインの設定は、プロジェクトごとに独立しています。

例えば、プロジェクト「io」でツールチェインの設定を変更しても、他のプロジェクトには反映されません。プロジェクトごとに設定する必要があります。1 つのプロジェクトの設定を変えると、ワークスペースすべての設定が変わった気になりがちですので、注意してください。

逆なことを言うと、プロジェクトごとに設定を変えることができます。

3. セクション

プログラムがオブジェクトファイル(機械語)に翻訳されると、プログラムだろうが変数データだろうが数値の羅列でしかありません。そのため、プログラムの機械語、変数データの機械語、・・・、というように**それぞれの内容に応じてグループ化します。それをセクションと呼びます。**セクションは「部分、区切り」と翻訳できますが、あまりピンと来ません。セクションをグループと言い換えて構いません。



1つ1つのまとまりをセクションといいます

3.1 セクションの種類

3.1.1 Cソースファイル

C ソースファイルは、コンパイラにより 4 つのセクションに自動で分けられます。

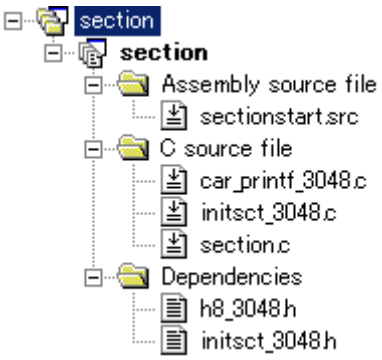
セクション名	領域名	詳細
P	プログラム領域 Program	プログラムがまとまって、セクション P という名前になります。
C	定数領域 Constant	const 型修飾子のある変数がまとまって、セクション C という名前になります。const 型修飾子とは、値の変更できない変数にすることです。 例) const int aaa = 50; /*aaa は変更できない*/
D	初期化データ領域 Data	初期値のある変数がまとまって、セクション D という名前になります。
B	未初期化データ領域 Block Started by Symbol	初期値のない変数がまとまって、セクション B という名前になります。

3.1.2 アセンブラリソースファイル

アセンブラソースファイルは、アセンブラはセクションを分けません。プログラマ自身がセクションを分類する必要があります。詳しくは後述します。

3.2　プログラムをセクションに分類

サンプルプログラムをインストールしているなら、「C:¥Workspace¥section」フォルダにワークスペース「section.hws」があります。これを開くと、



・section.c
・car_printf_3048.c
・initsct_3048.c
・sectionstart.src
の 4 ファイルが登録されています。h8_3048.h はインクルードファイルなのでアセンブルやコンパイルの対象にはなりません。

このプロジェクトを例に、説明していきます。

3.2.1　section.cをコンパイルしてセクションごとに分類

「section.c」は下記のようなプログラムになっています。

```
 1 :  /*************************************************************************/
 2 :  /* セクション説明用プログラム「section.c」                               */
 3 :  /*                              2006.04 ジャパンマイコンカーラリー実行委員会  */
 4 :  /*************************************************************************/
 5 :  #include    <machine.h>
 6 :  #include    "h8_3048.h"
 7 :
 8 :  const char C = 0xff;         /* const型修飾子のある変数なのでセクションC    */
 9 :  unsigned char d=0x55;        /* 初期値のある変数なのでセクションD           */
10 :  unsigned char b;             /* 初期値のない変数なのでセクションB           */
11 :  const char C2 = 0xf0;        /* const型修飾子のある変数なのでセクションC    */
12 :
13 :  void main( void ){           /* プログラムはセクションP                     */
14 :      PADDR = C;
15 :      PBDDR = C;
16 :      P6DDR = C2;
17 :      PBDR = b;
18 :      PADR = d;
19 :      while( 1 ); /* 無限ループで終了 */
20 :  }
```

コンパイルすると、オブジェクトファイル（機械語）に変換され、下記のようにセクションごとに分類されます。**C ソースファイルは、コンパイラが自動でセクション P, C, D, B に分類します。**

セクション名	プログラム
P	13 : void main(void){ /* プログラムはセクションP */ 14 : PADDR = C; 15 : PBDDR = C; 16 : P6DDR = C2; 17 : PBDR = b; 18 : PADR = d; 19 : while(1); /* 無限ループで終了 */ 20 : }
C	8 : const char C = 0xff; /* const 型修飾子のある変数なのでセクションC */ 11 : const char C2 = 0xf0; /* const 型修飾子のある変数なのでセクションC */
D	9 : unsigned char d=0x55; /* 初期値のある変数なのでセクションD */
B	10 : unsigned char b; /* 初期値のない変数なのでセクションB */

3.2.2 car_printf_3048.cをコンパイルしてセクションごとに分類

同様に、「car_printf_3048.c」もオブジェクトファイル（機械語）に変換され、セクションごとに分類されます。

セクション名	プログラム
P	内容は省略
C	内容は省略
D	内容は省略
B	内容は省略

3.2.3 initsct_3048.cをコンパイルしてセクションごとに分類

同様に、「initsct_3048.c」もオブジェクトファイル（機械語）に変換され、セクションごとに分類されます。

セクション名	プログラム
P	内容は省略
C	内容は省略
D	内容は省略
B	内容は省略

3.2.4 sectionstart.srcをアセンブルしてセクションごとに分類

(1) セクションを指定する命令

アセンブラソースプログラムは、プログラマ自身がセクションを指定しなければいけません。セクションを指定する命令は下記のようになります。

```
        .SECTION    セクション名
```

(2) プログラムの構成

sectionstart.src ファイルの構成は、

ベクタアドレス ＋ スタートアップルーチン(プログラム)

となります。
スタートアップルーチンはプログラムなのでセクションＰにします。下記のように記述すればセクションＰになります。

```
        .SECTION    P
        プログラム記述
        …
```

ベクタアドレスはセクションＶという新しいセクション名にします(名前はP,C,D,B,R 以外なら何でも構いません)。理由は、**ベクタアドレスは決まったアドレス(0x0000 番地)に配置しなければいけないため、専用のセクション名を新たに付けます**。

```
        .SECTION    V
        ベクタアドレスの記述
        …
```

(3) 実際のプログラム

「sectionstart.src」は下記のような内容です。

```
1 :  RESERVE: .EQU    H'FFFFFFFF      ; 未使用領域のアドレス
2 :
3 :           .IMPORT _main           ; 外部参照
4 :           .IMPORT _INITSCT
5 :
6 :           .SECTION V
7 :           .DATA.L RESET_START     ;  0 H'000000 ﾘｾｯﾄ
8 :           .DATA.L RESERVE         ;  1 H'000004 ｼｽﾃﾑ予約
9 :           .DATA.L RESERVE         ;  2 H'000008 ｼｽﾃﾑ予約
10 :          .DATA.L RESERVE         ;  3 H'00000c ｼｽﾃﾑ予約
11 :          .DATA.L RESERVE         ;  4 H'000010 ｼｽﾃﾑ予約
12 :          .DATA.L RESERVE         ;  5 H'000014 ｼｽﾃﾑ予約
13 :          .DATA.L RESERVE         ;  6 H'000018 ｼｽﾃﾑ予約
14 :          .DATA.L RESERVE         ;  7 H'00001c 外部割り込み NMI
15 :          .DATA.L RESERVE         ;  8 H'000020 ﾄﾗｯﾌﾟ命令
16 :          .DATA.L RESERVE         ;  9 H'000024 ﾄﾗｯﾌﾟ命令
17 :          .DATA.L RESERVE         ; 10 H'000028 ﾄﾗｯﾌﾟ命令
18 :          .DATA.L RESERVE         ; 11 H'00002c ﾄﾗｯﾌﾟ命令
19 :          .DATA.L RESERVE         ; 12 H'000030 外部割り込み IRQ0
20 :          .DATA.L RESERVE         ; 13 H'000034 外部割り込み IRQ1
21 :          .DATA.L RESERVE         ; 14 H'000038 外部割り込み IRQ2
22 :          .DATA.L RESERVE         ; 15 H'00003c 外部割り込み IRQ3
23 :          .DATA.L RESERVE         ; 16 H'000040 外部割り込み IRQ4
24 :          .DATA.L RESERVE         ; 17 H'000044 外部割り込み IRQ5
25 :          .DATA.L RESERVE         ; 18 H'000048 ｼｽﾃﾑ予約
26 :          .DATA.L RESERVE         ; 19 H'00004c ｼｽﾃﾑ予約
27 :          .DATA.L RESERVE         ; 20 H'000050 WDT MOVI
28 :          .DATA.L RESERVE         ; 21 H'000054 REF CMI
29 :          .DATA.L RESERVE         ; 22 H'000058 ｼｽﾃﾑ予約
30 :          .DATA.L RESERVE         ; 23 H'00005c ｼｽﾃﾑ予約
31 :          .DATA.L RESERVE         ; 24 H'000060 ITU0 IMIA0
32 :          .DATA.L RESERVE         ; 25 H'000064 ITU0 IMIB0
33 :          .DATA.L RESERVE         ; 26 H'000068 ITU0 OVI0
34 :          .DATA.L RESERVE         ; 27 H'00006c ｼｽﾃﾑ予約
35 :          .DATA.L RESERVE         ; 28 H'000070 ITU1 IMIA1
36 :          .DATA.L RESERVE         ; 29 H'000074 ITU1 IMIB1
37 :          .DATA.L RESERVE         ; 30 H'000078 ITU1 OVI1
38 :          .DATA.L RESERVE         ; 31 H'00007c ｼｽﾃﾑ予約
39 :          .DATA.L RESERVE         ; 32 H'000080 ITU2 IMIA2
40 :          .DATA.L RESERVE         ; 33 H'000084 ITU2 IMIB2
41 :          .DATA.L RESERVE         ; 34 H'000088 ITU2 OVI2
42 :          .DATA.L RESERVE         ; 35 H'00008c ｼｽﾃﾑ予約
43 :          .DATA.L RESERVE         ; 36 H'000090 ITU3 IMIA3
44 :          .DATA.L RESERVE         ; 37 H'000094 ITU3 IMIB3
45 :          .DATA.L RESERVE         ; 38 H'000098 ITU3 OVI3
46 :          .DATA.L RESERVE         ; 39 H'00009c ｼｽﾃﾑ予約
47 :          .DATA.L RESERVE         ; 40 H'0000a0 ITU4 IMIA4
48 :          .DATA.L RESERVE         ; 41 H'0000a4 ITU4 IMIB4
49 :          .DATA.L RESERVE         ; 42 H'0000a8 ITU4 OVI4
50 :          .DATA.L RESERVE         ; 43 H'0000ac ｼｽﾃﾑ予約
51 :          .DATA.L RESERVE         ; 44 H'0000b0 DMAC DEND0A
52 :          .DATA.L RESERVE         ; 45 H'0000b4 DMAC DEND0B
53 :          .DATA.L RESERVE         ; 46 H'0000b8 DMAC DEND1A
54 :          .DATA.L RESERVE         ; 47 H'0000bc DMCA DEND1B
55 :          .DATA.L RESERVE         ; 48 H'0000c0 ｼｽﾃﾑ予約
56 :          .DATA.L RESERVE         ; 49 H'0000c4 ｼｽﾃﾑ予約
57 :          .DATA.L RESERVE         ; 50 H'0000c8 ｼｽﾃﾑ予約
58 :          .DATA.L RESERVE         ; 51 H'0000cc ｼｽﾃﾑ予約
59 :          .DATA.L RESERVE         ; 52 H'0000d0 SCI0 ERI0
60 :          .DATA.L RESERVE         ; 53 H'0000d4 SCI0 RXI0
61 :          .DATA.L RESERVE         ; 54 H'0000d8 SCI0 TXI0
62 :          .DATA.L RESERVE         ; 55 H'0000dc SCI0 TEI0
63 :          .DATA.L RESERVE         ; 56 H'0000e0 SCI1 ERI1
64 :          .DATA.L RESERVE         ; 57 H'0000e4 SCI1 RXI1
65 :          .DATA.L RESERVE         ; 58 H'0000e8 SCI1 TXI1
66 :          .DATA.L RESERVE         ; 59 H'0000ec SCI1 TEI1
67 :          .DATA.L RESERVE         ; 60 H'0000f0 A/D ADI
68 :
69 :          .SECTION P
70 :  RESET_START:
71 :          MOV.L   #H'FFF10,ER7    ; スタックの設定
72 :          JSR     @_INITSCT       ; RAMエリアの初期化
73 :          JSR     @_main          ; Ｃ言語の main()関数へジャンプ
74 :  OWARI:
75 :          BRA     OWARI
76 :
77 :          .END
```

セクションVに指定
これ以降のプログラムやデータは、新たにセクションを指定するまでセクションVになります。

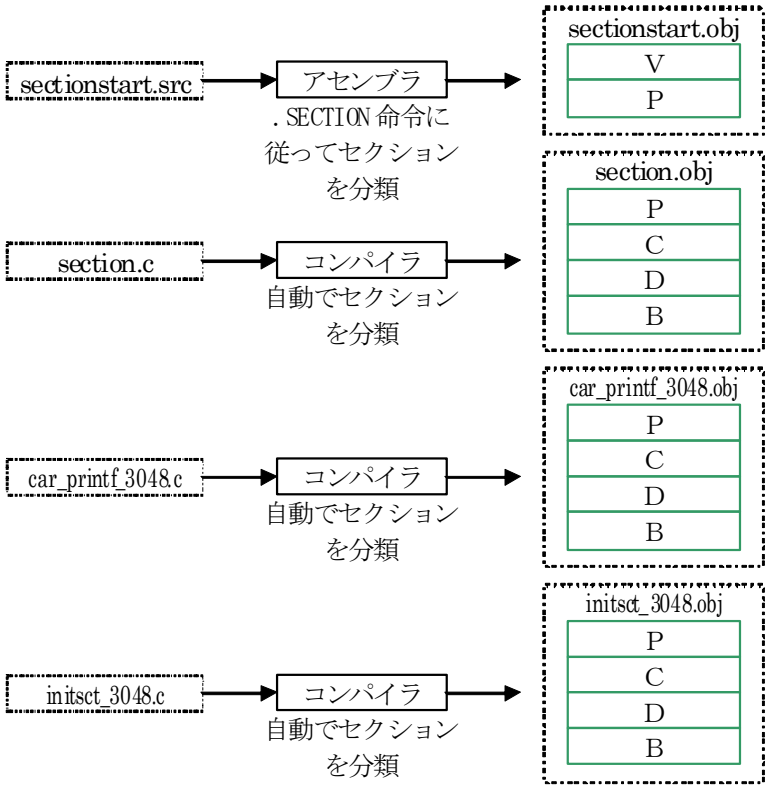
セクションPに指定
これ以降のプログラムやデータは、新たにセクションを指定するまでセクションPになります。

この内容でアセンブルすると、下記のようにセクションごとに分類されオブジェクトファイルに変換されます。繰り返しますが、アセンブルプログラムは、自分でセクションを指定します。C ソースプログラムは、コンパイラが自動でセクションを分類してくれます。

セクション名	プログラム
V	7 : .DATA.L RESET_START ; 0 H'000000 ﾘｾｯﾄ 中略 67 : .DATA.L RESERVE ; 60 H'0000f0 A/D ADI
P	70 : RESET_START: 71 : MOV.L #H'FFF10,ER7 ; スタックの設定 72 : JSR @_INITSCT ; RAM エリアの初期化 73 : JSR @_main ; C言語の main()関数へジャンプ 74 : OWARI: 75 : BRA OWARI

3.2.5 まとめ

4 ファイルは下記のようなセクションに分類します。



セクション名	領域名	詳細
V	ベクタアドレス領域	ベクタアドレス
P	プログラム領域 Program	関数(プログラム)
C	定数領域 Constant	const 型修飾子のある変数
D	初期化データ領域 Data	初期値のある変数
B	未初期化データ領域 Block Started by Symbol	初期値のない変数

3.3　セクションの割り当てアドレス

分類した後、それぞれのセクションを ROM 領域に割り当てるか、RAM 領域に割り当てるか決めます。

セクション名	領域名	**割り当て先**	説明
V	ベクタアドレス領域	**ROM（番地指定あり）**	ベクタアドレスは、H8/3048F-ONE では 0x0000 番地～0x00f3 番地に割り当てます。 ベクタアドレスについては、H8/3048 実習マニュアルの「timer2.c」のプログラム解説に詳しく載っていますのでそちらを参照してください。
P	プログラム領域	**ROM**	プログラムは変更する必要がないので ROM 領域に割り当てます。
C	定数領域	**ROM**	定数は変更する必要がないので ROM 領域に割り当てます。
D	初期化データ領域	**ROM**	以下の「※セクション D について」を参照してください。
B	未初期化データ領域	**RAM**	以下の「※セクション B について」を参照してください。

3.3.1　セクションDについて

セクション D は、プログラムを実行したときから値の入っているグローバル変数です。RAM 領域に割り当てます。しかし、RAM の値はマイコンの電源を入れたとき「不定」（どのような値になっているか分からない）です。そのため、セクション D は RAM 領域ではありますが、電源を入れたときに初期値を入れる工夫をしなければいけません。

そこで、セクション D を電源が消えてもメモリの値が消えない ROM に割り当てます。そして、あらかじめセクション D の値を RAM 領域にコピーするプログラムを作っておき、それをマイコンの電源を入れた直後(main 関数を実行する前)に実行するようにします。これで、**あたかも RAM 領域に初めから値が入っているような動作になります**。

リンクはセクション単位で行われます。そのため、コピー先にもセクション名を付けなければいけません。今回は RAM 領域にコピーするので、セクション R とします。セクション D と同じサイズのセクション R を作り、マイコンの電源を入れた直後、セクション D の内容をセクション R へコピーします。

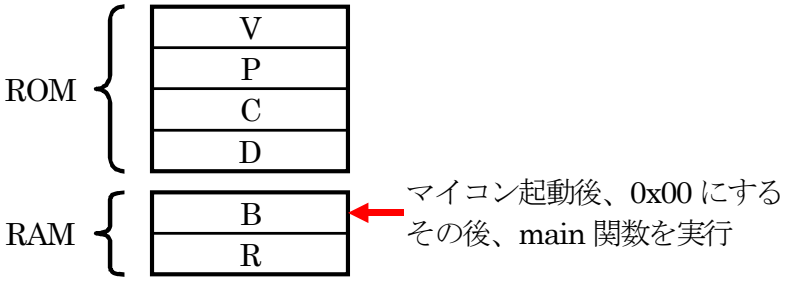
セクション名	領域名	割り当て先	詳細
R	未初期化データ領域	RAM	セクション D のコピー先です。セクション D と同じサイズのセクションです。

3.3.2　セクションBについて

セクション B は、初期値のないグローバル変数領域です。しかし、**ANSI C 規格（C言語の規格）で初期値のないグローバル変数は、初期値 0x00 でなければいけない、と決まってます**。実は初期値があるのです。

変数は RAM 領域に割り当てます。しかし、RAM の値はマイコンの電源を入れたとき「不定」（どのような値になっているか分からない）です。そのため、セクション B は RAM 領域ではありますが、電源を入れたときに 0x00 になるよう工夫をしなければいけません。

そこで、あらかじめセクション B の値を 0x00 にするプログラムを作っておき、それをマイコンの電源を入れた直後(main 関数を実行する前)に実行するようにします。これで、**あたかもセクション B の領域が初めから 0x00 であるような動作になります**。

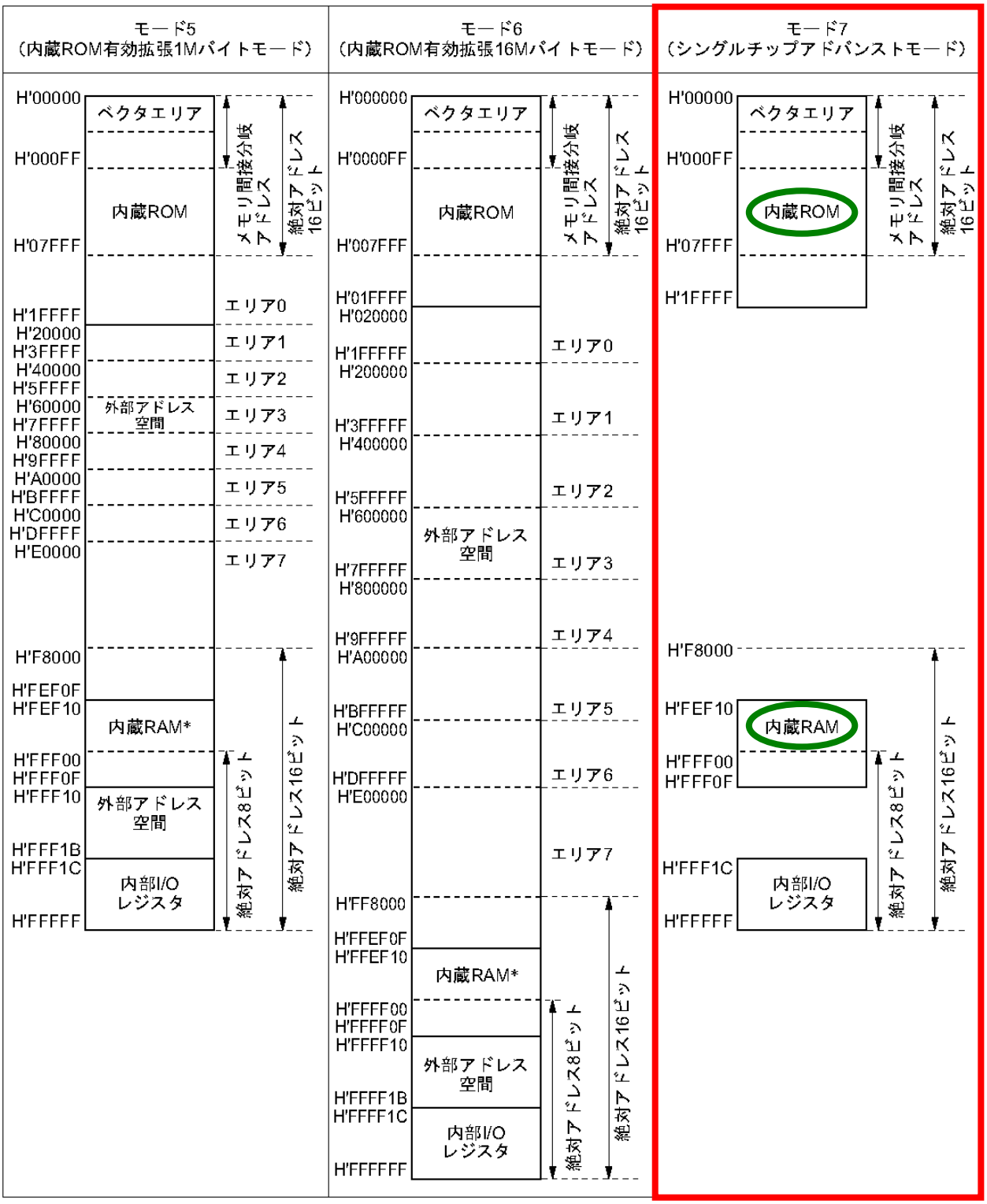


3.4　実際のアドレス

以上で、それぞれのセクションをROM領域、RAM領域のどちらに割り当てるか決まりました。では、それぞれ何番地なのでしょうか。実際のアドレスを確認します。

3.4.1　RY3048Foneボードの場合（H8/3048F-ONE）

H8/3048F-ONE のハードウェアマニュアルの「3.6　各動作モードのメモリマップ」を参照します。シングルチップモード（ROM や RAM など外付けのデバイスはない状態）で動作していますので、モード 7 のメモリマップとなります。



※H8/3048B シリーズ　ハードウェアマニュアル　第 2 版　3-6 ページより抜粋

表から、H8/3048F-ONE マイコンは、

・内蔵 ROM のアドレスは、0x00000～0x1ffff 番地 ただし、0x00000～0x000f3 番地はベクタアドレス
・内蔵 RAM のアドレスは、0xfef10～0xfff0f 番地
ということになります。

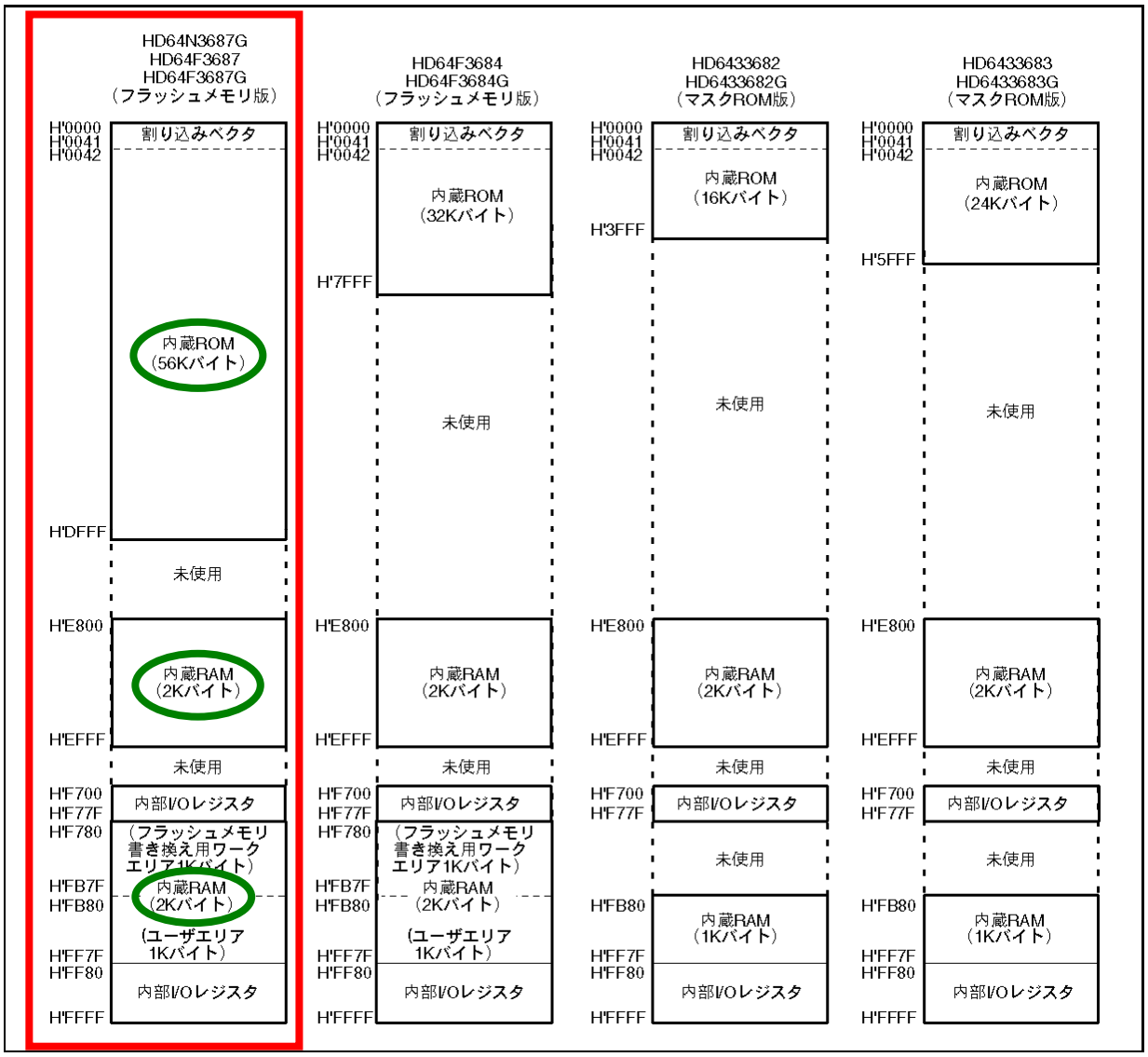
セクションの割り当て先は下表のようになります。

セクション名	領域名	割り当て先	割り当てアドレス
V	ベクタアドレス領域	ROM (番地指定あり)	ベクタアドレスは 0x0000 番地から割り当てます。 ちなみにベクタアドレスは、0x0000～0x00f3 番地までです。
P	プログラム領域	ROM	ベクタアドレスが終わった 0x00f4 番地からで良いのですが、分かりやすい 0x0100 番地からとします。
C	定数領域	ROM	セクション P の次から割り当てます。
D	初期化データ領域	ROM	セクション C の次から割り当てます。
B	未初期化データ領域	RAM	RAM の先頭である 0xfef10 番地にセクションBを割り当てます。
R	未初期化データ領域	RAM	セクション B の次から割り当てます。

この表に従って、設定します。

3.4.2　RY3687 ボードの場合（H8/3687F）

H8/3687 のハードウェアマニュアルの「2.1 アドレス空間とメモリマップ」を参照します。フラッシュメモリ版ですので、「HD64F3687G」と書かれているメモリマップとなります。



※H8/3687 シリーズ　ハードウェアマニュアル　第 1 版　2-2 ページより抜粋

表から、H8/3687F マイコンは、

・内蔵 ROM のアドレスは、0x0000～0xdfff 番地　※ただし、0x0000～0x0041 番地はベクタアドレス
・内蔵 RAM のアドレスは、0xe800～0xefff 番地、0xf780～ff7f 番地
ということになります。

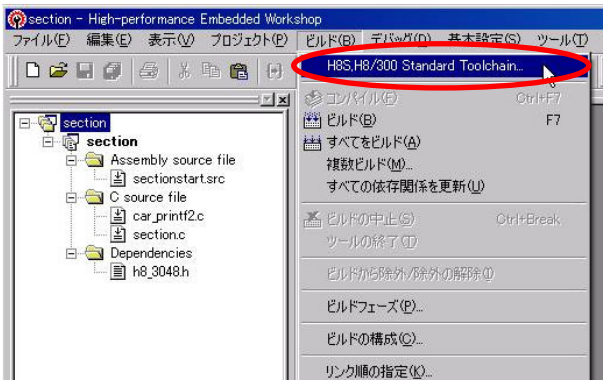
セクションの割り当て先は下表のようになります。

セクション名	領域名	割り当て先	割り当てアドレス
V	ベクタアドレス領域	ROM (番地指定あり)	ベクタアドレスは 0x0000 番地から割り当てます。 ちなみにベクタアドレスは、0x0000～0x0041 番地までです。
P	プログラム領域	ROM	ベクタアドレスが終わった 0x0042 番地からです。
C	定数領域	ROM	セクション P の次から割り当てます。
D	初期化データ領域	ROM	セクション C の次から割り当てます。
B	未初期化データ領域	RAM	RAM の先頭である 0xe800 番地にセクションBを割り当てます。
R	未初期化データ領域	RAM	セクション B の次から割り当てます。

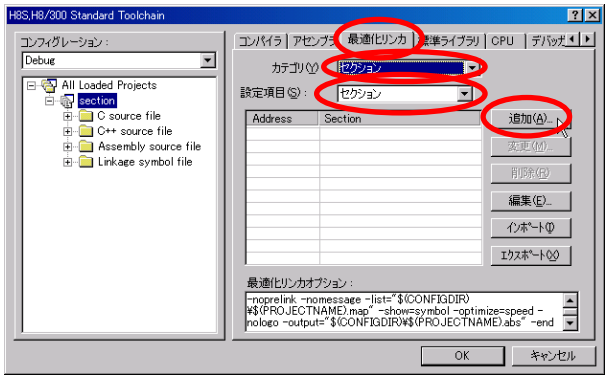
この表に従って、設定します。

3.5　ツールチェインでのセクション設定

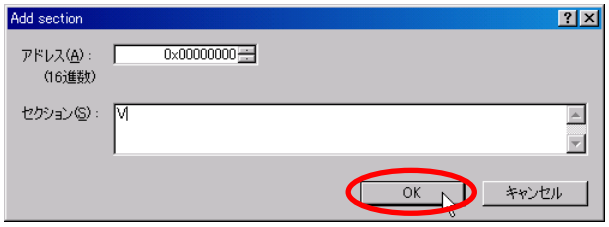
セクションの番地指定はツールチェインで行います。下記にその手順を示します。



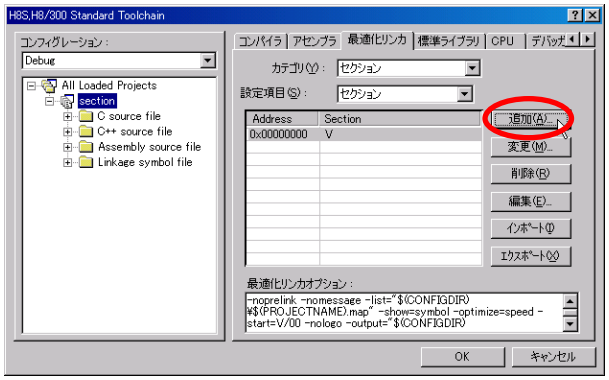
1.「ビルド→H8S,H8/300 Standard Toolchain」(ツールチェイン)を選択します。



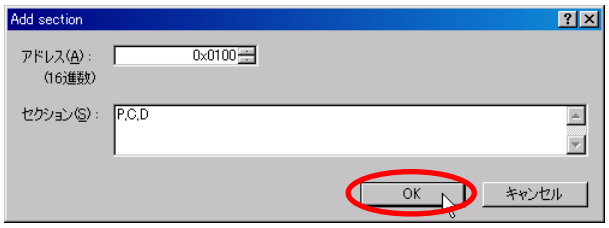
2.「最適化リンカ」を選択します。「カテゴリ:**セクション**」、「設定項目:**セクション**」を選択、追加をクリックします。



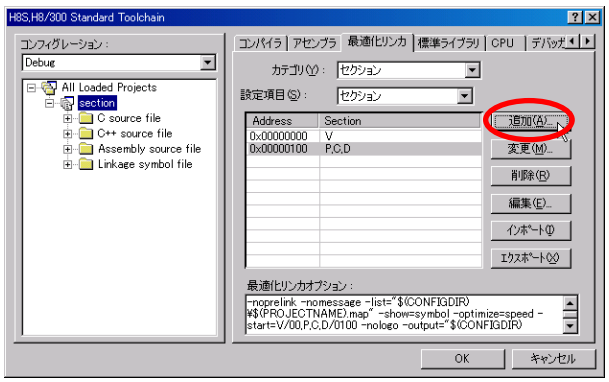
3.「アドレス:**0x0000**」(変更無し)、「セクション:**V**」と入力します。OKをクリックします。



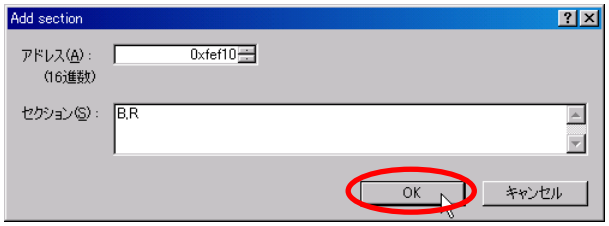
4.追加をクリックします。



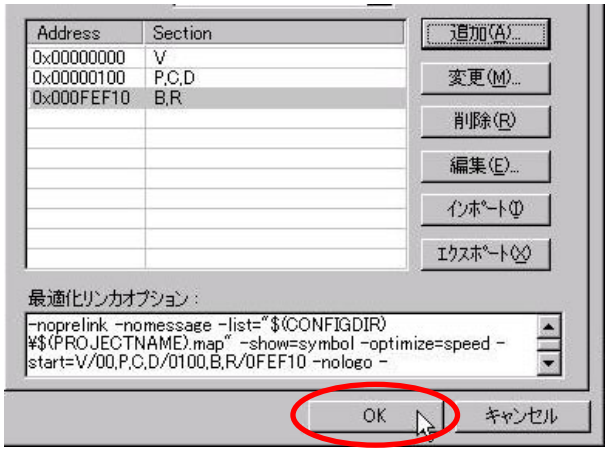
5.「アドレス:**0x0100**」、「セクション:**P, C, D**」と入力します。セクション名を続けて設定する場合は、「,」(カンマ)で区切ります。OKをクリックします。



6.追加をクリックします。

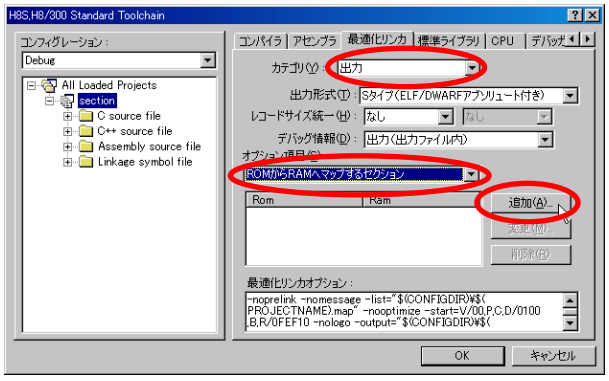


7. 「アドレス:0xfef10」、「セクション:B,R」と入力します。OK をクリックします。

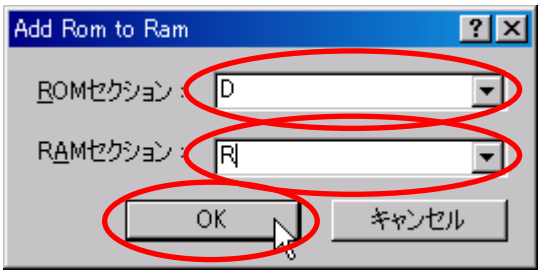


8. これで設定完了です。上画面のようになったかもう一度確かめてください。OK をクリックしてセクション設定を終わります。

次に、「セクション R はセクション D のコピー先です」という設定を行います。



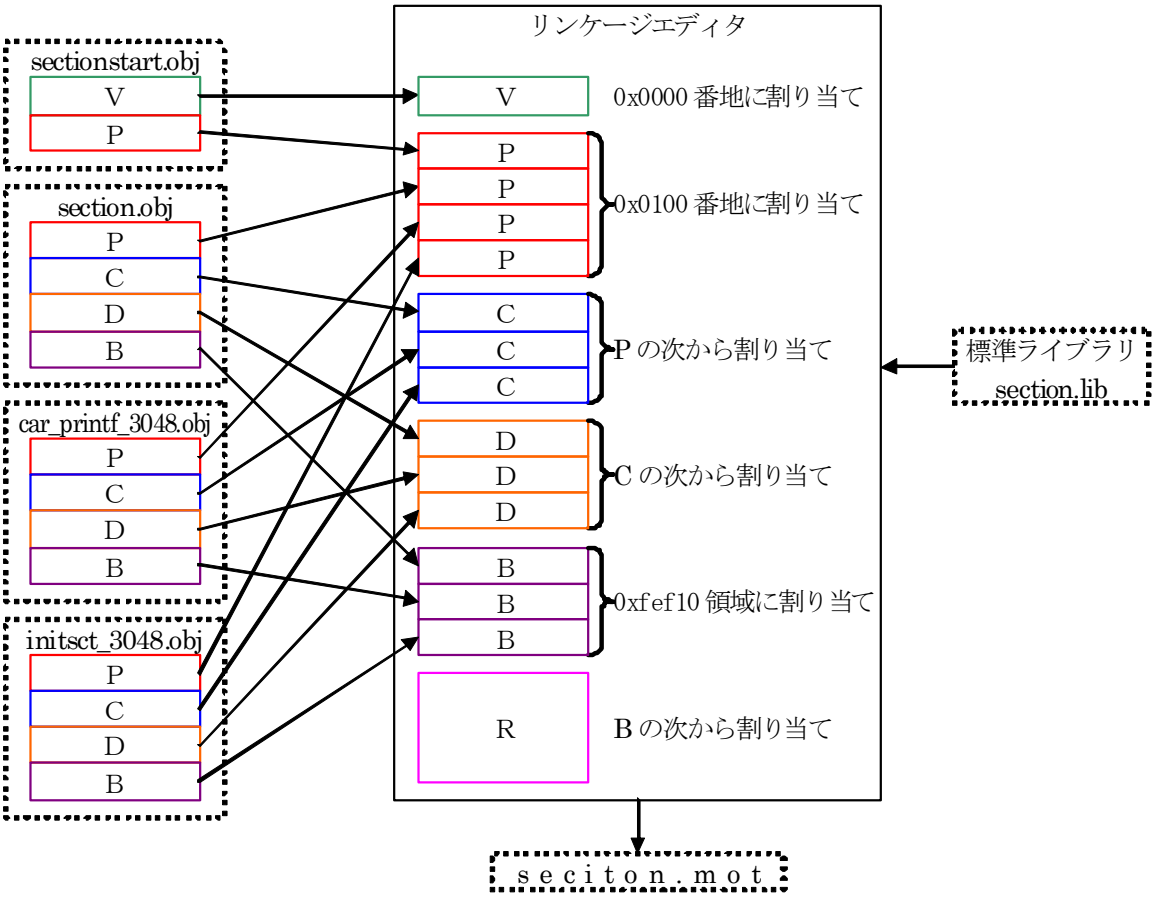
9.「カテゴリ:**出力**」、「オプション項目:**ROM から RAM へマップするセクション**」を選択、追加をクリックします。



10.「ROM セクション:**D**」、「RAM セクション:**R**」を入力、OK をクリックします。

今回の設定でセクションと実際のアドレスが関連づけされました。
リンケージエディタは、これらの情報を基に、
・各オブジェクトファイルを合体させるときセクションごとにまとめます
・標準ライブラリ内の使用する関数（printf、scanf など）だけを取り込みます
・まとめたセクションを指定された番地に割り当てます
・その上で、MOT ファイルを出力します

下図にそのイメージをまとめます（アドレスは H8/3048F-ONE の場合）。



こうして、MOT ファイルができるのです。

3.6　セクションD,R,Bを初期化するプログラム

先に説明したとおり、RAM の値はマイコンの電源を入れたとき「不定」（どのような値になっているか分からない）です。しかし、グローバル変数は電源を入れたときに、下記のような値である必要があります。

・初期値のないグローバル変数は、0x00 でなければいけません。
・初期値のあるグローバル変数は、初期値が入っていなければいけません。

これらの作業を行うプログラムを、電源を入れたときに実行します。

ツールチェインの最適化リンカのオプション項目設定で、「ROM から RAM へマップするセクション」を D から R に設定しました。これは、セクション R の領域をセクション D と同じだけ確保しなさいという設定です。**ツールチェインの設定だけでは、セクション D のデータはセクション R にコピーされません。**コピーはプログラムで行います。

「sectionstart.src」は下記のようなプログラムになっています。

```
  1 :  RESERVE: .EQU    H'FFFFFFFF      ; 未使用領域のアドレス
  2 :
  3 :          .IMPORT _main            ; 外部参照
  4 :          .IMPORT _INITSCT
  5 :
  6 :          .SECTION V
  7 :          .DATA.L RESET_START      ;  0 H'000000 ﾘｾｯﾄ
中略
 67 :          .DATA.L RESERVE          ; 60 H'0000f0 A/D ADI
 68 :
 69 :          .SECTION P
 70 :  RESET_START:
 71 :          MOV.L   #H'FFF10,ER7     ; スタックの設定
 72 :          JSR     @_INITSCT        ; RAM エリアの初期化
 73 :          JSR     @_main           ; C言語の main()関数へジャンプ
 74 :  OWARI:
 75 :          BRA     OWARI
 76 :
 77 :          .END
```

72 行目に記述のある INITSCT（イニシャライズセクションの意味、イニットセクトと呼びます）関数が下記の作業を行います。

・セクション D 領域のデータをセクション R へコピー
・セクション B をクリア

4 行目は、INITSCT 関数が「sectionstart.src」ファイル内になく、別なファイルにあることを指定しています。

initsct_3048.c ファイル内にある INITSCT 関数は、下記のような内容です。

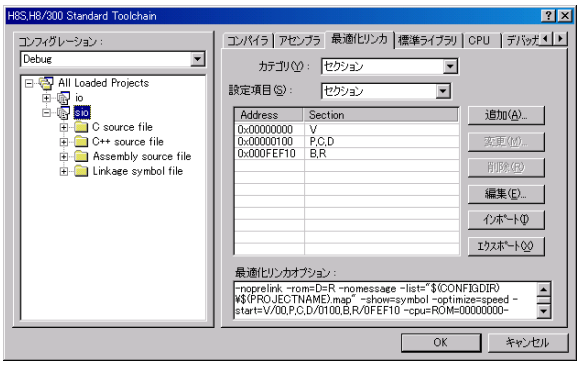
```
void INITSCT( void )
{
    int *s, *e, *r;

    r = __sectop("R");              /* R セクション(RAM)の最初   */
    s = __sectop("D");              /* D セクション(ROM)の最初   */
    e = __secend("D");              /* D セクション(ROM)の最後   */
    while(s < e) {
        *r++ = *s++;                /* R ← D コピー             */
    }

    s = __sectop("B");              /* B セクション(RAM)の最初   */
    e = __secend("B");              /* B セクション(RAM)の最後   */
    while(s < e) {
        *s++ = 0x00;                /* B ← 0x00                 */
    }
}
```

詳しく説明しませんが、「**cinitsct_3048.c にある INITSCT 関数は、セクション D から R へコピーと、セクション B のクリアを行っている。main 関数を呼ぶ前に実行する**」と覚えておいてください。

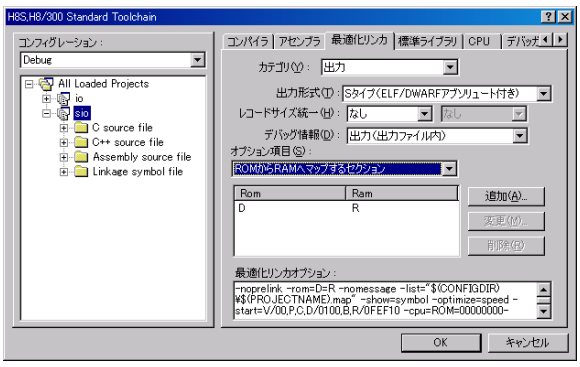
3.7　セクションのまとめ

セクションの名称と詳細をまとめます。

セクション名	名称	詳細
V	ベクタアドレス領域 Vector	ベクタアドレスの先頭アドレスに割り当てます。 H8/3048F-ONE は 0x0000～0x00f3 番地になります。
P	プログラム領域 Program	ROM エリアに割り当てます。 ベクタアドレスが終わった 0x00f4 番地からで良いのですが、キリのいい 0x0100 番地からとします。
C	定数領域 Constant	ROM エリアに割り当てます。 「Constant」は「不変の」という意味です。変えることのないデータを配置するアドレスを設定します。 例えば、 const double PI=3.14 とすると、自動的に 3.14 が C セクションに割り当たります。
D	初期化データ領域 Data	ROM エリアに割り当てます。 初期値付き変数の初期値データを保存するセクションです。
B	未初期化データ領域 Block Started by Symbol	RAM エリアに割り当てます。 初期値のない変数がこのセクションに割り当てられます。無いといっても 0x00 が初期値です。 H8/3048F-ONE は 0xfef10 番地からになります。
R	未初期化データ領域 RAM	RAM エリアに割り当てます。 初期値付き変数がセクション R に割り当てられます。初期値付き変数というぐらいですから、電源が入った瞬間から、あらかじめ設定しておいた値が入っていなければいけません。しかし、RAM は電源が切れると内容が消えてしまいます。そこで、ROM にセクション D を作っておきます。セクション D には変数の初期値を保存しておきます。電源が入った瞬間、セクション D の値をセクション R へコピーすれば、あたかも初期値付き変数に値が入っているようにすることができます。セクション D とセクション R は必ず組み合わせて使用します。



「ビルド→H8S,H8/300 Standard Toolchain」
最適化リンカタブ
カテゴリ：セクション
設定項目：セクション
の画面です。



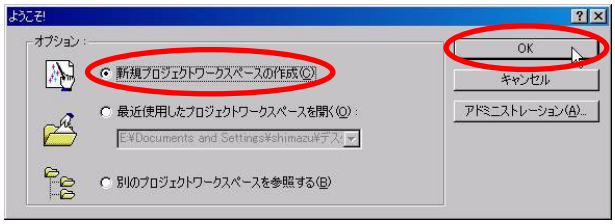
「ビルド→H8S,H8/300 Standard Toolchain」
最適化リンカタブ
カテゴリ：出力
オプション項目：ROM から RAM へ
　　マップするセクション
の画面です。

4. 実行委員会開発環境のプログラムをルネサス統合開発環境へ移植する

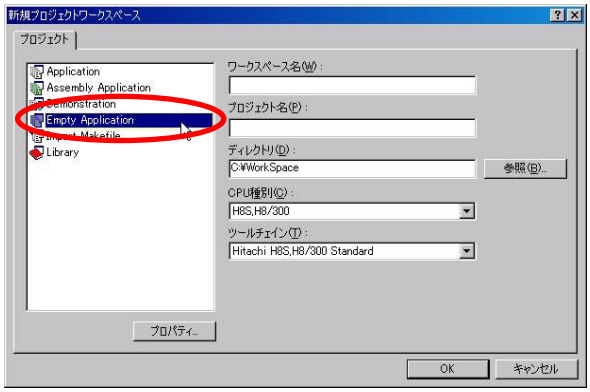
実行委員会開発環境で製作したプログラムを、ルネサス統合開発環境に移植します。
ここでは例として、kit05 関係のファイルをルネサス統合開発環境のワークスペースに登録します。
・kit05.c は、　　「C:¥h8n_win¥data¥3048¥c¥kit05.c」
・kit05start.src は、「C:¥h8n_win¥data¥3048¥c¥kit05start.src」
・kit05.sub は、　　「C:¥h8n_win¥data¥3048¥c¥kit05.sub」
に、それぞれあるものとします。

4.1 新しいワークスペースを作成する

1. ルネサス統合開発環境を実行します。



2.「新規プロジェクトワークスペースの作成」を選択します。OK をクリックします。

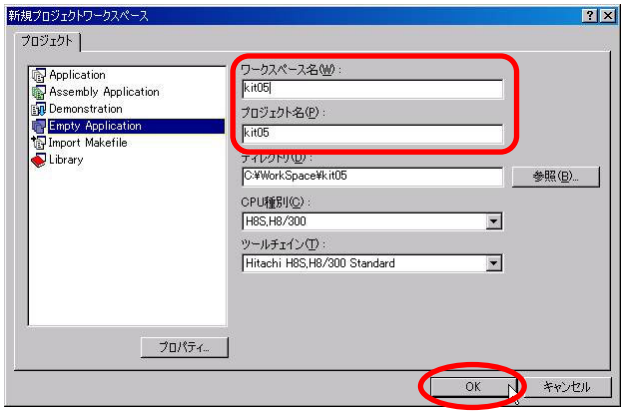


3.「プロジェクト:**Empty Application**」を選択します。

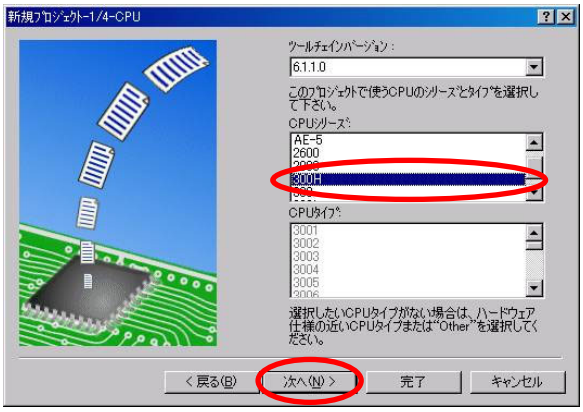
※プロジェクトについて

今回、「Empty Application」を選びました。これはどういう意味なのでしょうか。他のプロジェクトはどのような意味なのでしょうか。下記に表でまとめます。

プロジェクト名	内容
Application	C++言語で記述された、初期ルーチンファイルを含むプログラムを作るためのプロジェクトです。内蔵レジスタ名の記述の仕方が、「PA.DR.BIT.D7」など構造体を使用した場合はこちらを選択し、「PADR」のような記述を使う場合は、「Empty Application」を選んでください（ただし、定義ファイルは自分で作成する必要があります）。
Assembly Application	アセンブリ言語で記述された、初期ルーチンファイルを含むプログラムを作るためのプロジェクトです。
Demonstration	C 言語で記述された、デモンストレーション用プログラムを作成するための、プロジェクト（SH/H8 シミュレータデバッガ専用）です。
Empty Application	ツールチェイン環境の、設定のみのプロジェクトです。ファイルは作りません。**移植するときはこちらを選びます**。
Import Makefile	既存のメイクファイルを取り込んで、プログラムを作るためのプロジェクトです。HEW3.0 より、GNU make フォーマットまたは、Hmake フォーマット（HEW が出力）のメイクファイルを解析し、作成するワークスペースにファイル情報を取り込むことができます。
Library	ライブラリファイルを作るためのプロジェクトです。



4. ワークスペース名、プロジェクト名を入力します。ここでは「kit05」と入力します。プロジェクト名は自動でワークスペース名と同じ「kit05」が入ります。変更しても構いません。入力ができたら OK をクリックします。

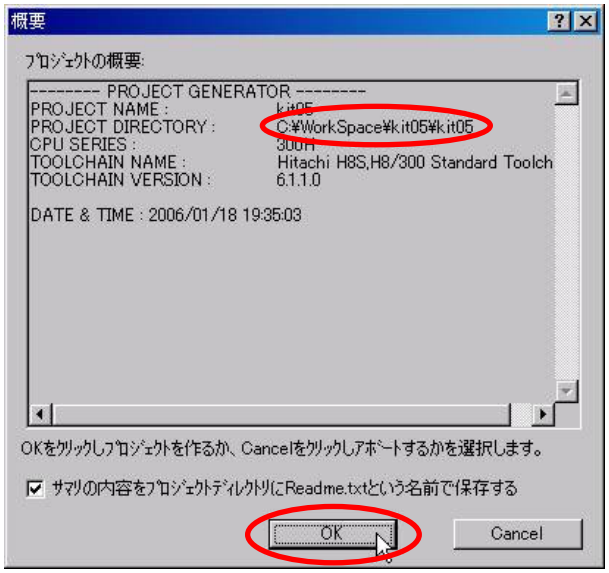


5. 「CPU シリーズ：**300H**」を選択します。次へをクリックします。

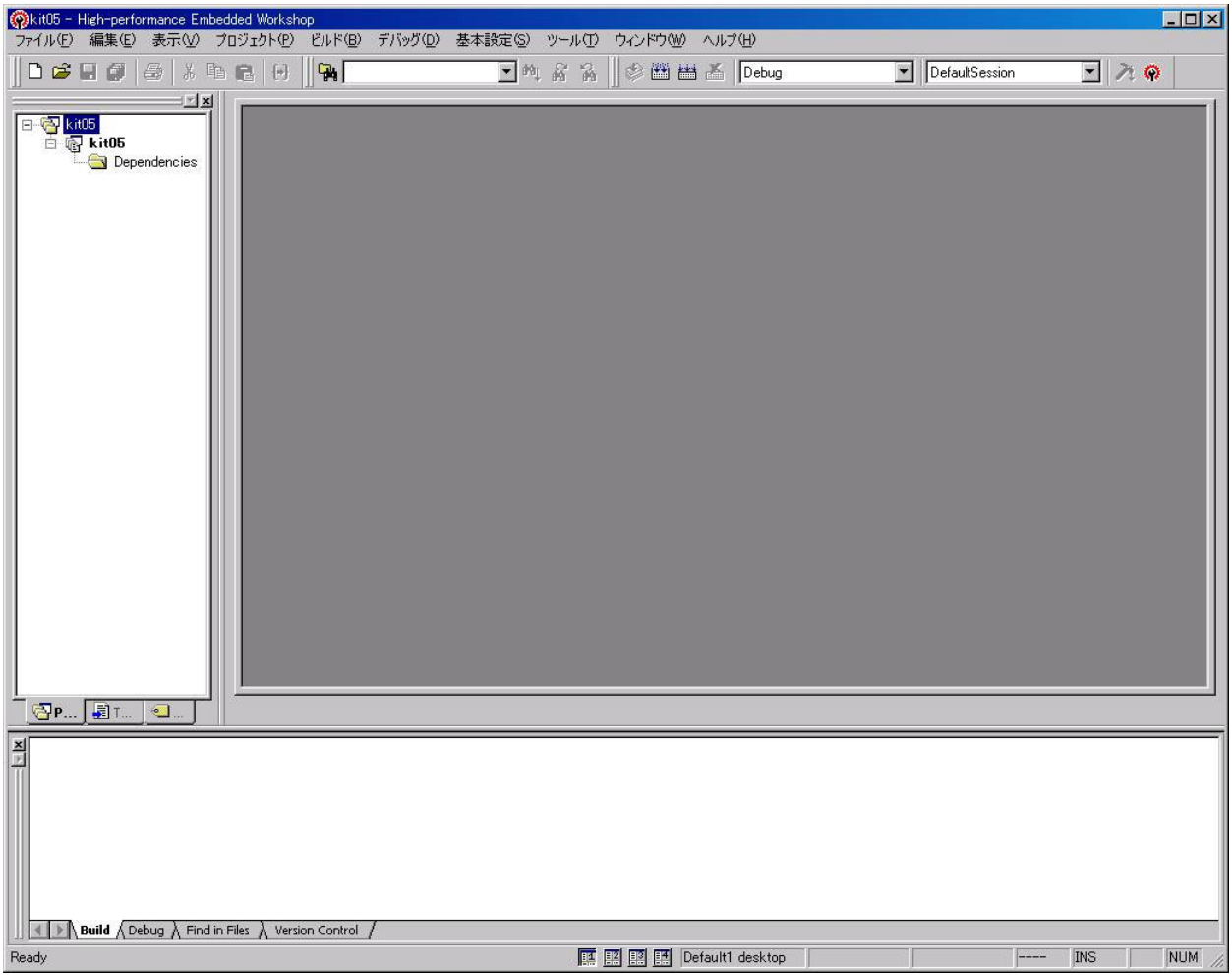
6. CPU の種類により設定が変わります。
   ●RY3048Fone ボード(H8/300H シリーズ)の場合
   「動作モード:**Advanced**」、
   「アドレス空間:**1M byte**」を選択します。
   ●RY3687 ボード(H8/300H Tiny シリーズ)の場合
   「動作モード:**Nomal**」を選択します。

   設定ができたら、次へをクリックします。

7. 何もせずに完了をクリックします。

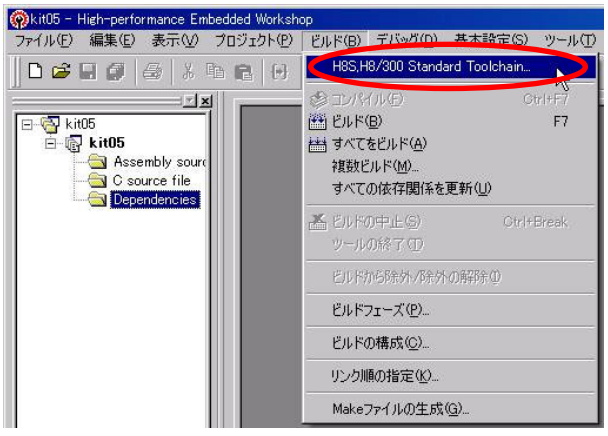


8. 設定できました。OKをクリックします。ここで覚えておくことは、「PROJECT　DIRECTORY」(プロジェクトディレクトリ)です。このフォルダが新規に作られ、後ほどこのフォルダの中にプログラムを入れることになります。

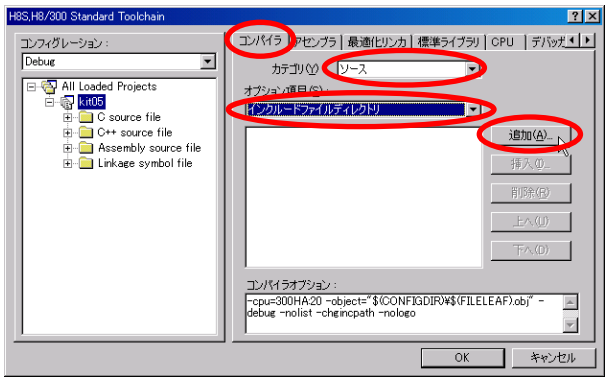


9. ルネサス統合開発環境の画面が立ち上がりました。次にファイルを登録します。

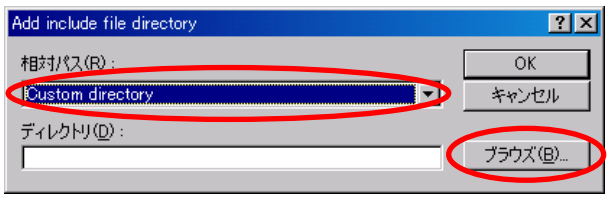
4.2　ファイルを登録する前設定

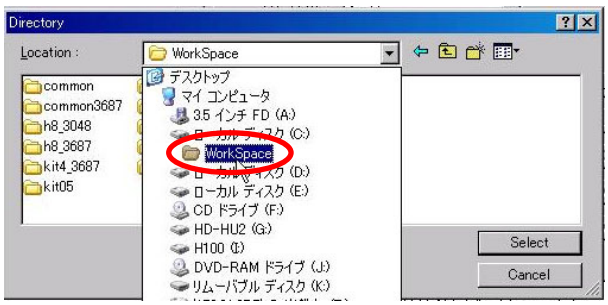


1.「ビルド→H8S,H8/300 Standard Toolchain」(ツールチェイン)を選択します。



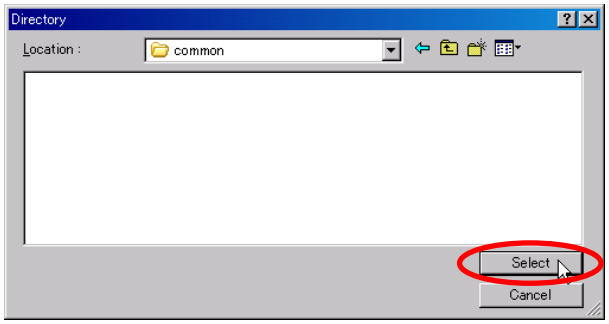
2.「コンパイラ」を選択、「カテゴリ:**ソース**」、「オプション項目:**インクルードファイルディレクトリ**」を選択します(最初はこの画面になります)。追加をクリックします。

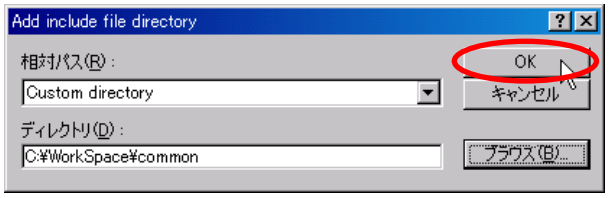




3.「相対パス:**Custom Directory**」を選択します。ブラウズをクリックします。

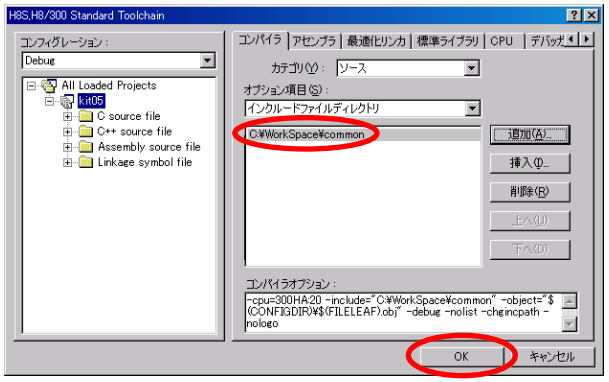
4.「c:¥workspace」フォルダを選択します。





5. ●H8/3048F-ONE の場合
「common」フォルダを選択し、Select をクリックします。
●H8/3687F の場合
「common3687」フォルダを選択し、Select をクリックします。

6. OK をクリックします。



7. もう一度、「c:¥workspace¥common」、または「c:¥workspace¥common3687」になっている事を確かめて、OK をクリックします。

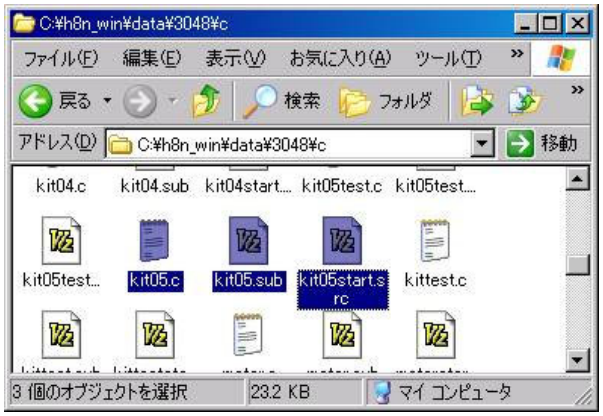
これで、前準備は完了です。次は、c ファイルや src ファイルを登録します。

4.3 ファイルの登録

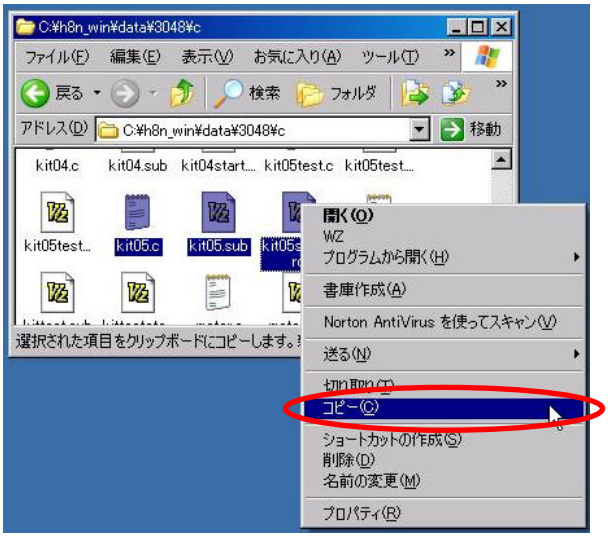
これから行う作業は、

元のファイル		プロジェクトディレクトリ
・C:¥h8n_win¥data¥3048¥c¥**kit05.c**	を	C:¥Workspace¥kit05¥kit05 へコピー
・C:¥h8n_win¥data¥3048¥c¥**kit05start.src**	を	C:¥Workspace¥kit05¥kit05 へコピー
・C:¥h8n_win¥data¥3048¥c¥**kit05.sub**	を	C:¥Workspace¥kit05¥kit05 へコピー

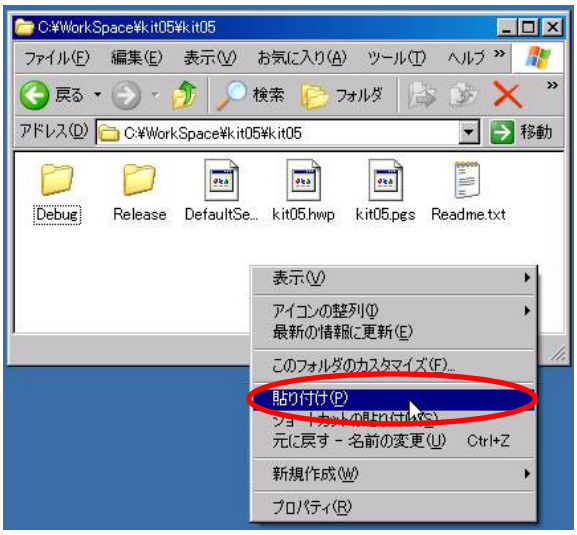
の作業を行います。kit05.sub ファイルは必要ないのですが、セクションの設定をするとき参考資料として使うのでコピーしておきます。



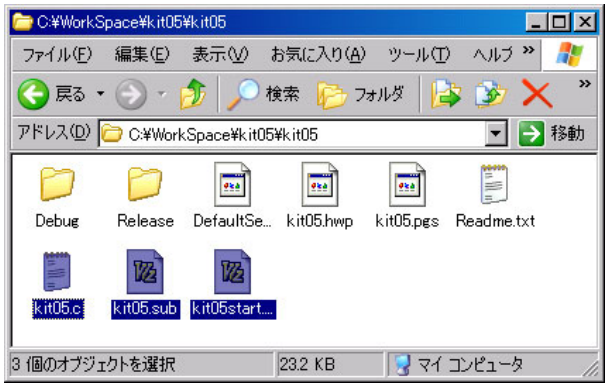
1.「C:¥h8n_win¥data¥3048¥c」フォルダ（元のファイルのあるフォルダ）を開きます。「kit05.c」、「kit05start.src」、「kit05.sub」を選択します。



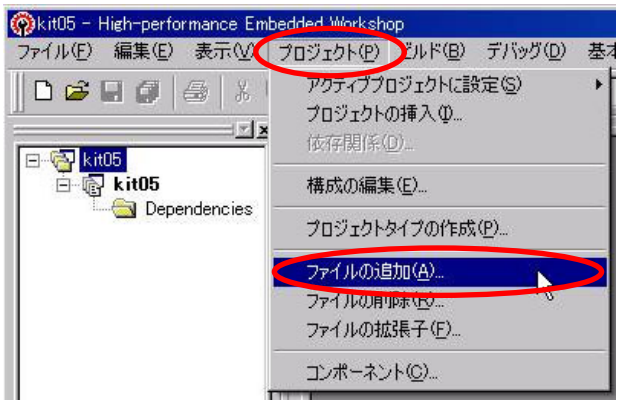
2. 右クリックして「コピー」をクリックします。



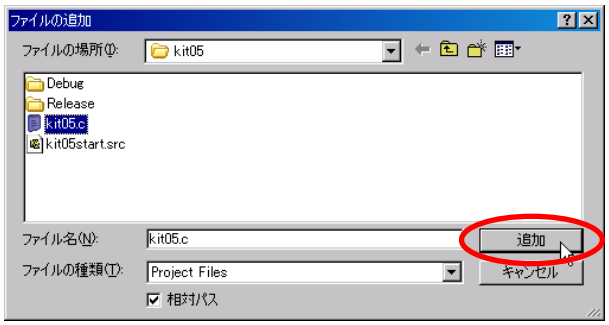
3.「C:¥Workspace¥kit05¥kit05」フォルダ（新しく作ったプロジェクトフォルダ）を開きます。右クリックして「貼り付け」をクリックします。



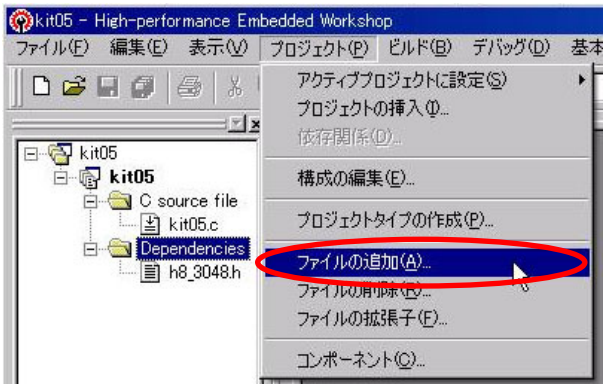
4. これで、「C:¥Workspace¥kit05¥kit05」フォルダに
   ・kit05.c
   ・kit05start.src
   ・kit05.sub
   の 3 ファイルがコピーされました。



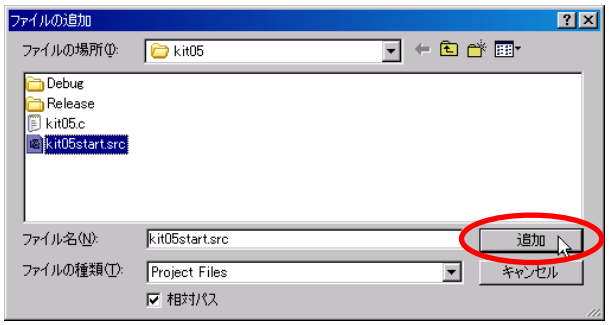
5. ルネサス統合開発環境へ戻り、「プロジェクト→ファイルの追加」を選択します。



6. 「kit05.c」を選択、追加をクリックします。kit05.c が登録されます。

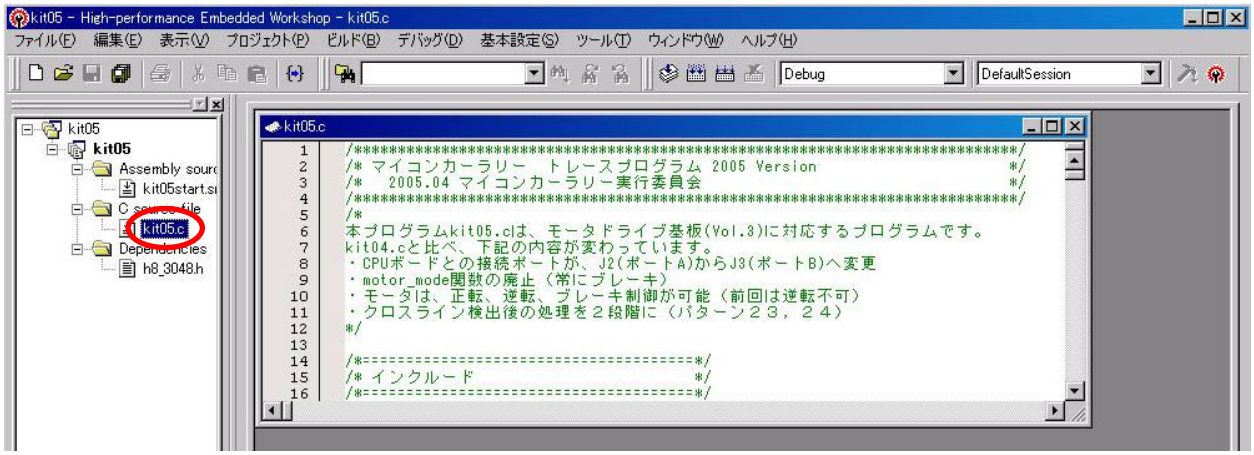


7. 再度、「プロジェクト→ファイルの追加」を選択します。



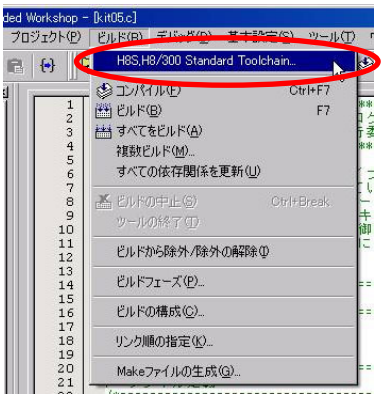
8. 「kit05start.src」を選択、追加をクリックします。

9. ファイルが追加されました。「h8_3048.h」はインクルードファイルです。**インクルードファイルは追加しません**。ルネサス統合開発環境がＣファイルや、srcファイルを解析して、依存(Dependency)ファイルとして**自動的に登録されます**。



10. kit05.c をダブルクリックすると、エディタウィンドウにプログラムが表示されます。

4.4　ツールチェインの設定

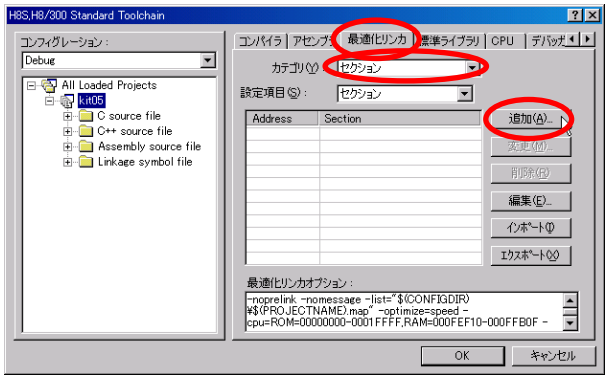


1. 次に、sub ファイルの情報を登録します。実はこれから行う作業手順は、sub ファイルが必要有りません。この作業で中身を見ることを考えて sub ファイルをコピーしました。
「ビルド→H8S,H8/300 Standard Toolchain」(ツールチェイン)を選択します。ここでコンパイラのパラメータなど、すべて設定することができます。詳しくは、「**2. ツールチェインの設定**」を参照して、設定してください。

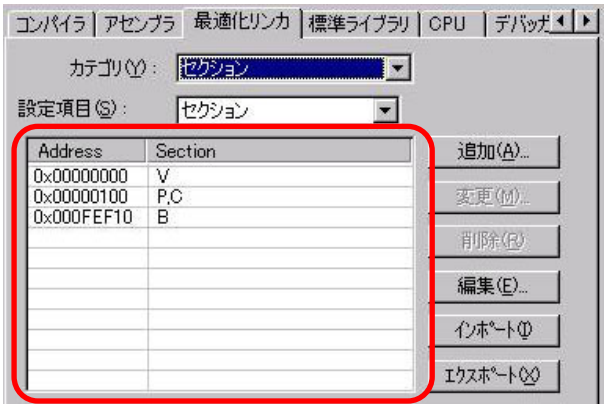
2. セクションは、プログラムの構造によってそれぞれ違います。「C:¥WorkSpace¥kit05¥kit05¥kit05.sub」をテキストエディタで開きます。

```
input kit05start,kit05
lib c:¥h8n_win¥3048¥c¥c38hae.lib
output kit05
print kit05
start V(000000)
start P,C(000100)
start B(0fef10)
exit
```

start と記述している部分が kit05 のセクション情報です。
・セクション V は、0x0000 番地
・セクション P と C は、0x0100 番地
・セクション B は、0xfef10 番地
となります。



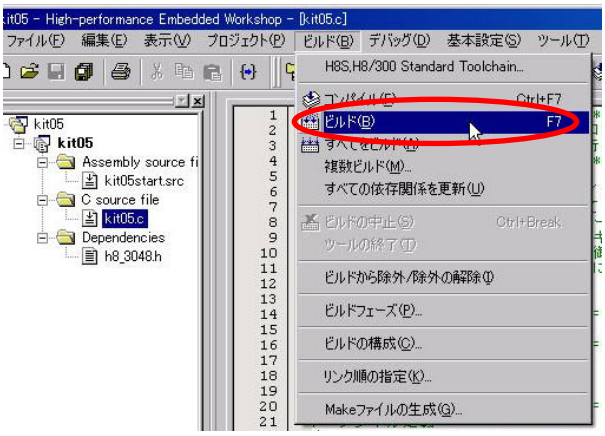
3.「最適化リンカ」を選択します。「カテゴリ：**セクション**」を選択します。追加をクリックします。



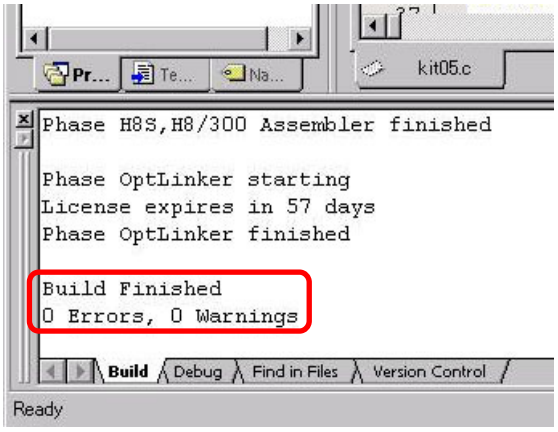
4. sub ファイルの内容と同様に設定します。

4.5 ビルド

「ビルド」とは、実行委員会開発環境の「コンパイル」と同じです。正常にビルドできると、MOT ファイルが作成されます。



1.「ビルド→ビルド」を実行します。



2. `Build Finished`
`0 Errors, 0 Warnings`
が表示されれば、完了です。

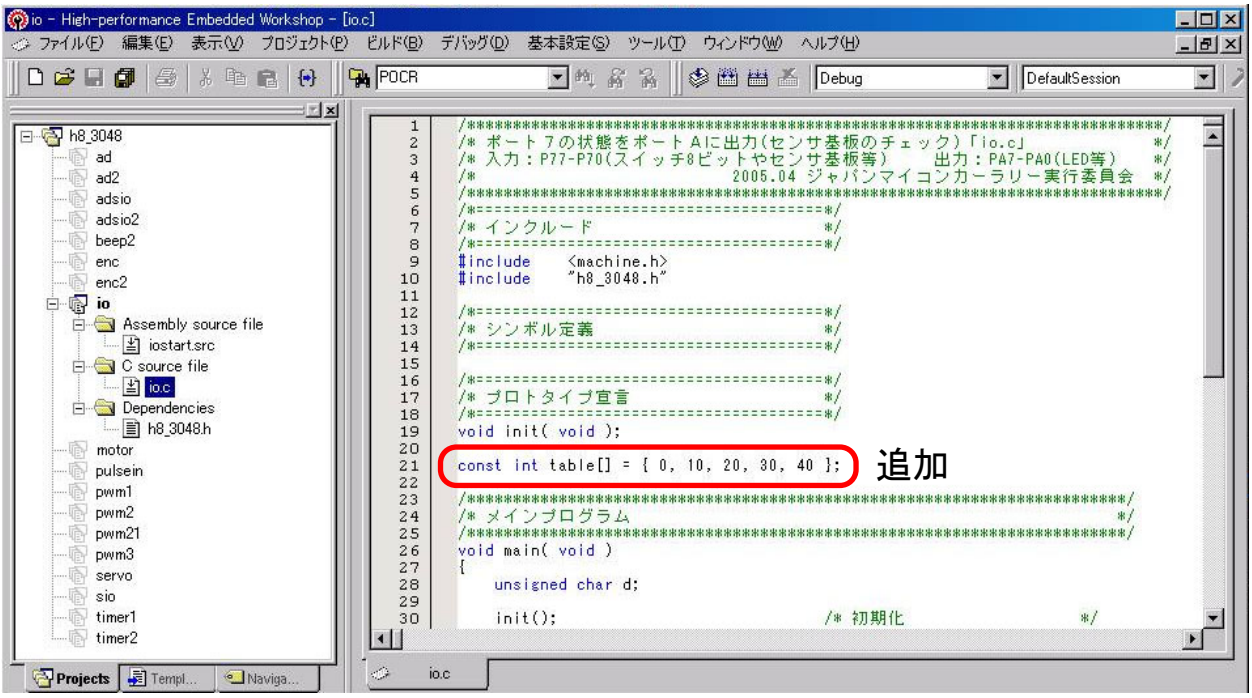
5. プログラム改造例

実際に改造が行われそうな手順を説明します。

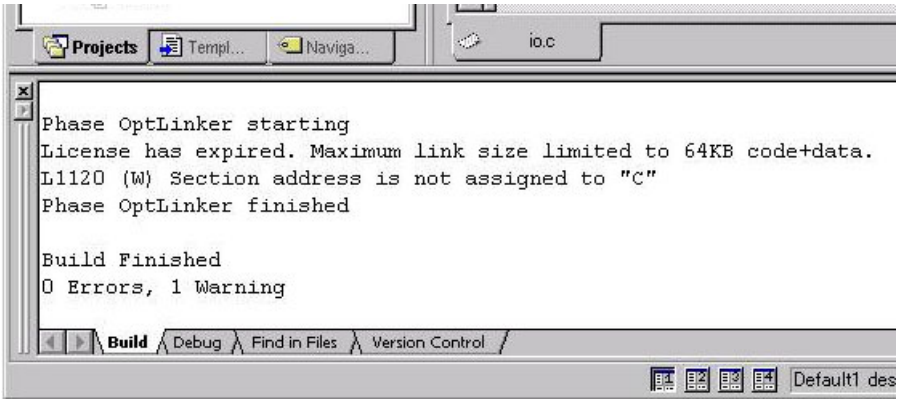
5.1 定数を追加したときの設定(セクションCの追加)

定数を追加したとき、行わなければいけない作業を説明します。ここではワークスペース「h8_3048」のプロジェクト「io」を例に説明します(実際のプロジェクト「io」は、今回の説明の設定を既にしています)。

5.1.1 追加内容

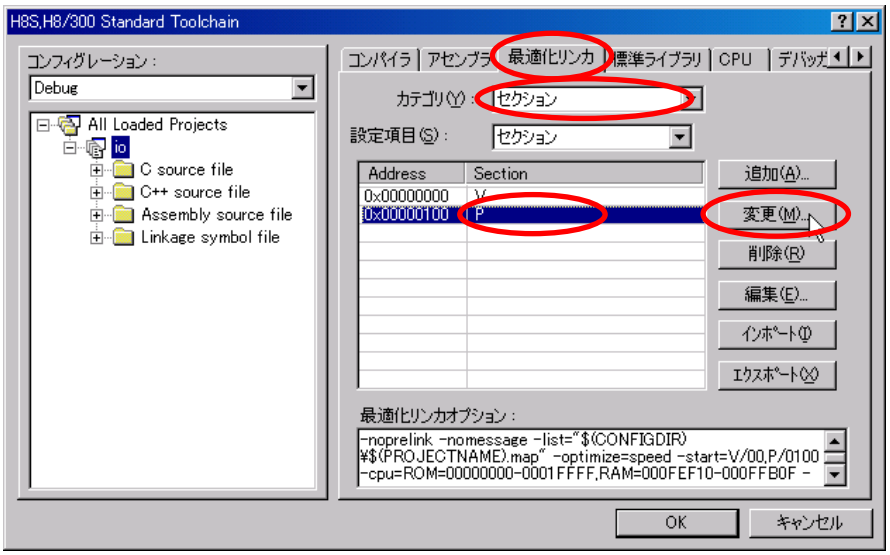


1. io.c の 21 行目に定数を追加しました。

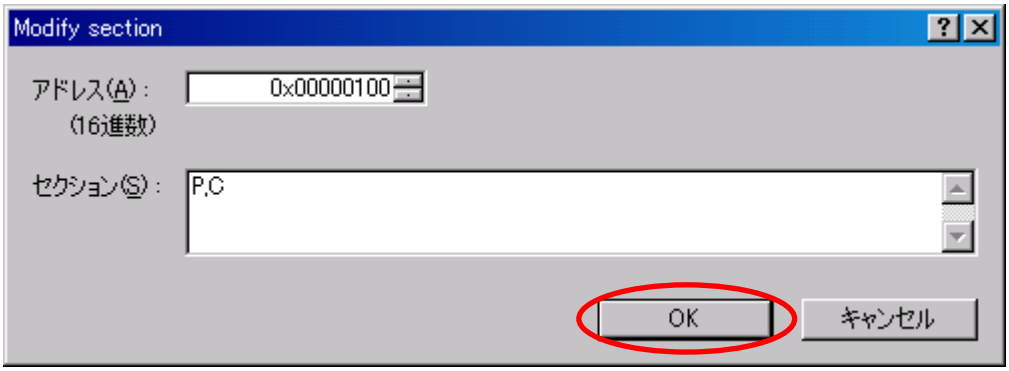


2. ビルドすると、エラーが出ました。F1 キーを押してエラーのヘルプを見ると「**"セクション"のアドレス指定がありません。"セクション"を最後尾に配置します。**」と出ました。セクション C が定義されていないのでツールチェインで設定してください、というエラーです。ツールチェインでセクション C を追加して、エラーを無くしましょう。

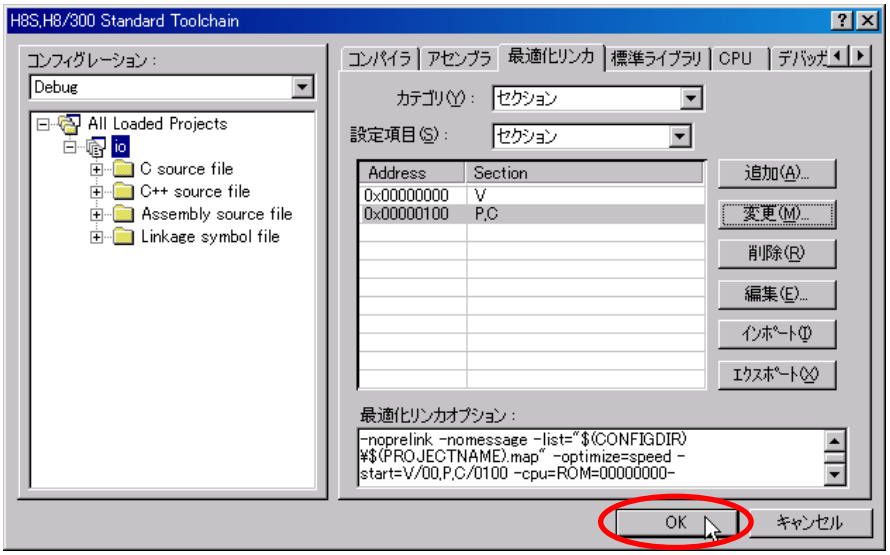
5.1.2 設定内容



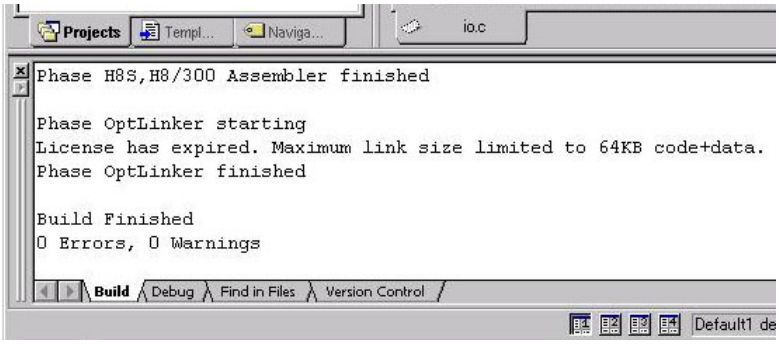
3.「ビルド→H8S,H8/300 Standard Toolchain」を選択します。「最適化リンカ」を選択、「カテゴリ：**セクション**」を選択します。一覧の「P」部分をクリックし、変更をクリックします。



4.「セクション：P」を「セクション：P, C」(P と C の間はカンマ)と変更し、OK をクリックします。



5. OK をクリックして設定完了です。

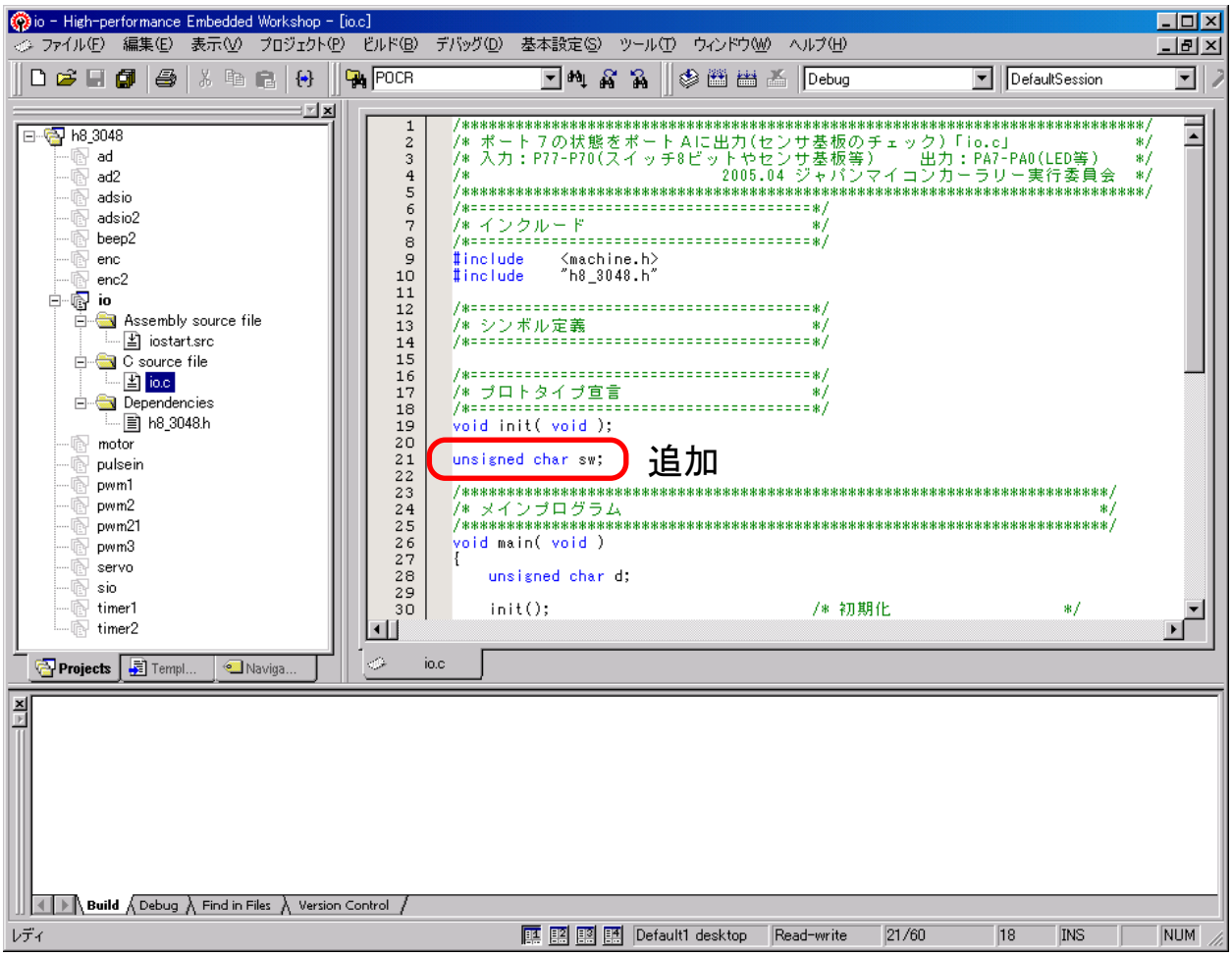


6. ビルドすると、今度はエラーがありません。これで設定完了です。

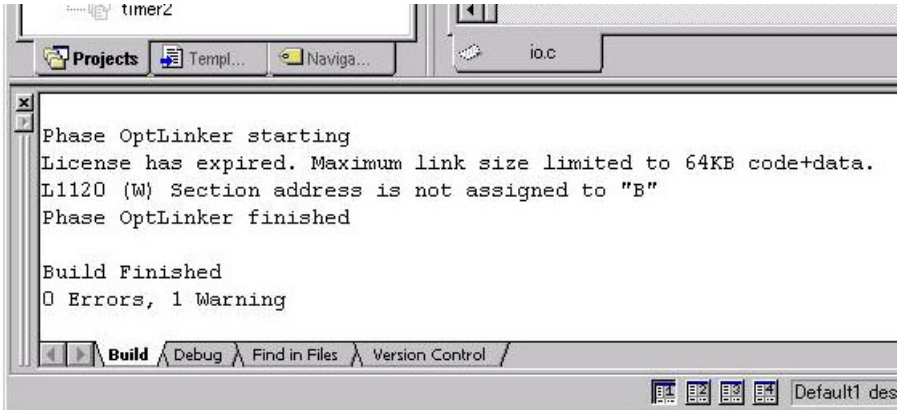
5.2　初期値のないグローバル変数を追加したときの設定(セクションBの追加)

初期値のないグローバル変数を追加したとき、行わなければいけない作業を説明します。ここではワークスペース「h8_3048」のプロジェクト「io」を例に説明します(実際のプロジェクト「io」は、今回の説明の設定を既にしています)。

5.2.1　追加内容

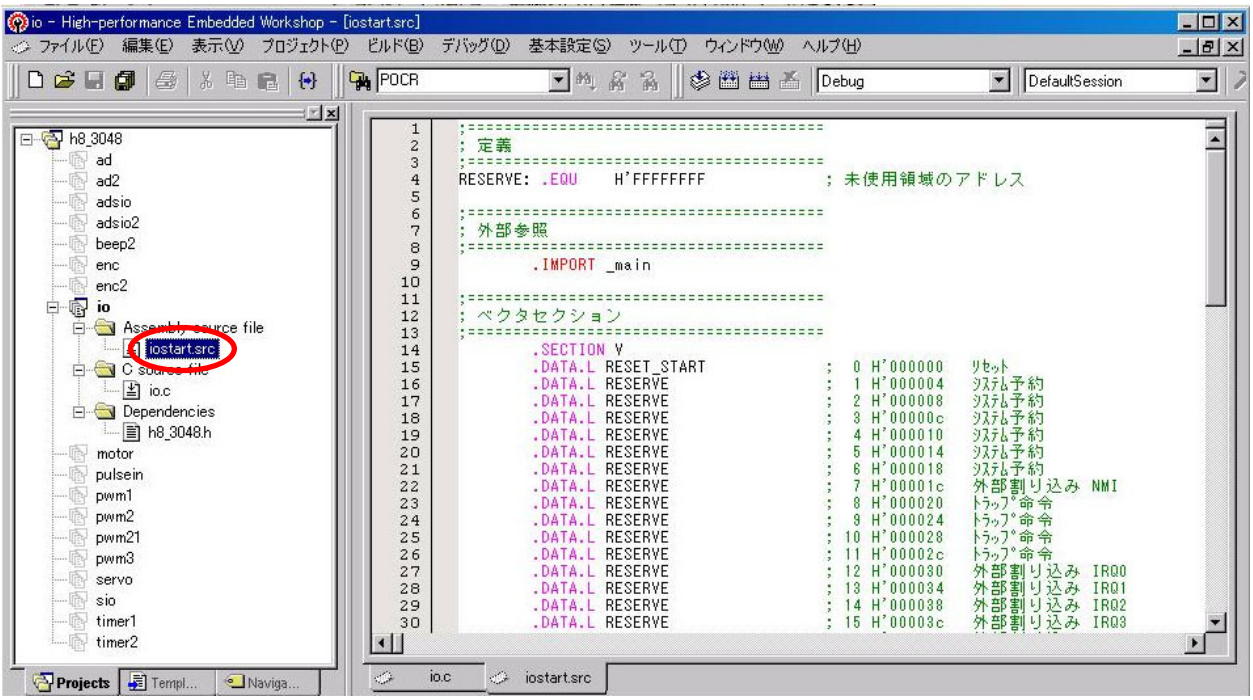


1. io.c の 21 行目に初期値のないグローバル変数 sw を追加しました。

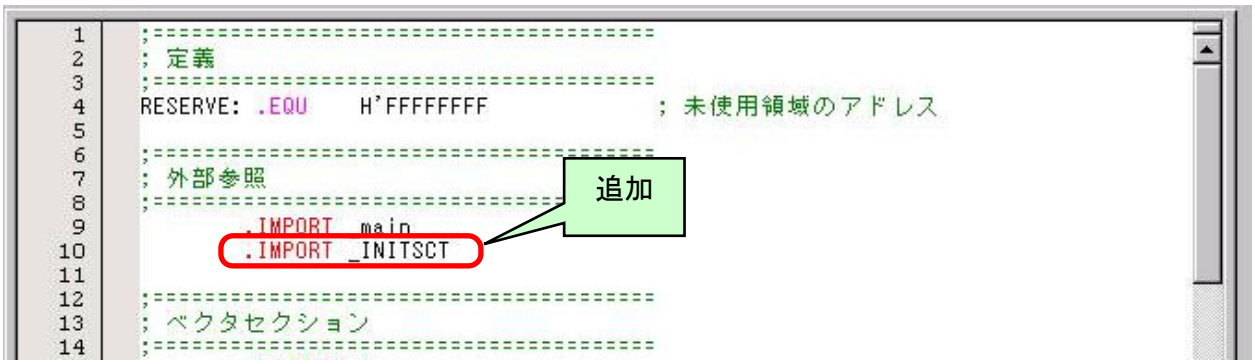


2. ビルドすると、エラーが出ました。F1 キーを押してエラーのヘルプを見ると「"セクション"のアドレス指定がありません。"セクション"を最後尾に配置します。」と出ました。セクション B が定義されていないのでツールチェインで設定してください、というエラーです。ツールチェインでセクション B を追加して、エラーを無くしましょう。

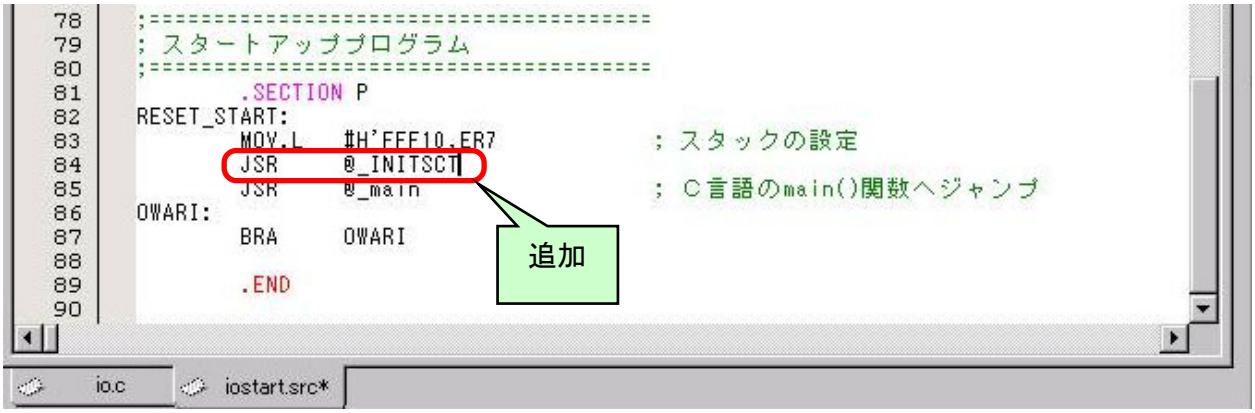
5.2.2 設定内容



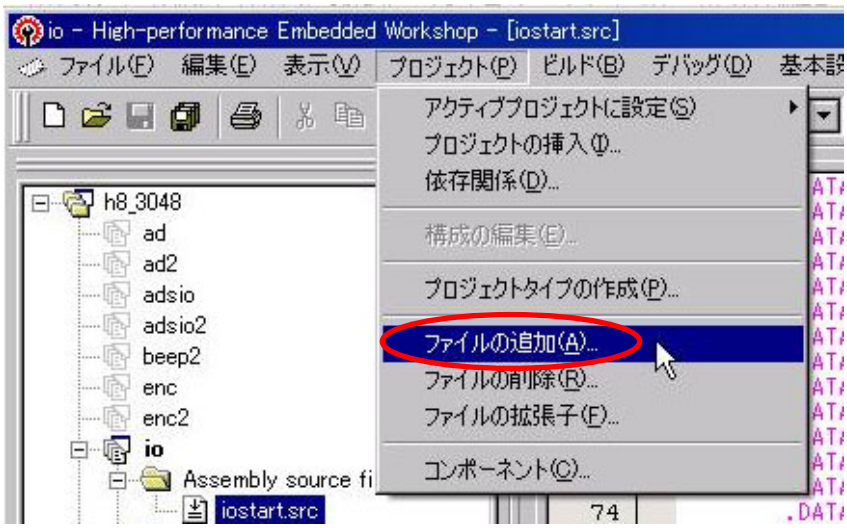
1. セクション B の RAM 領域を 0 にする関数の登録が必要です。「iostart.src」を開きます。



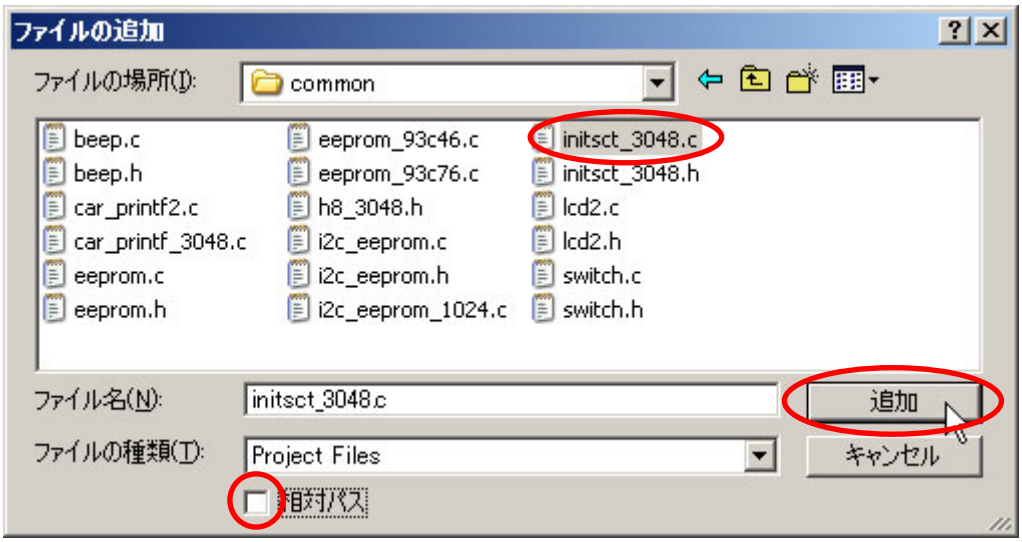
2.「.IMPORT _main」のある次の行に「.IMPORT _INITSCT」を追加します。



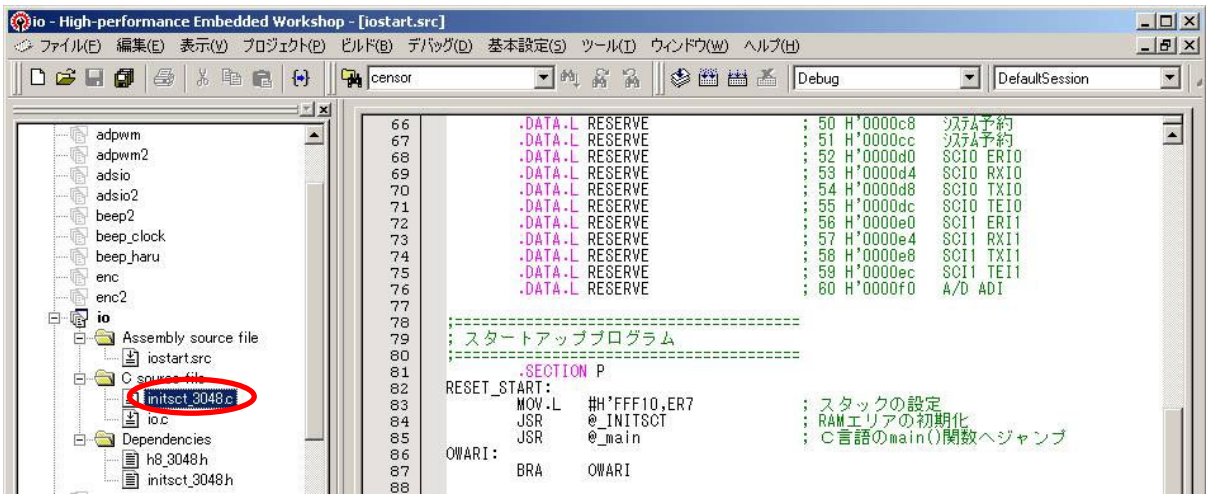
3.「MOV.L #H'FFF10,ER7」のある次の行に「JSR @_INITSCT」を追加します。



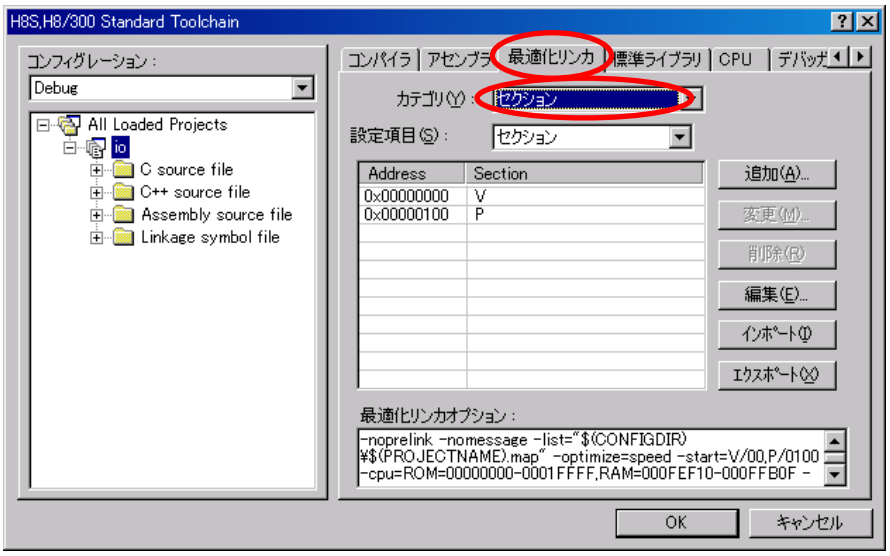
4.「プロジェクト→ファイルの追加」を選択します。



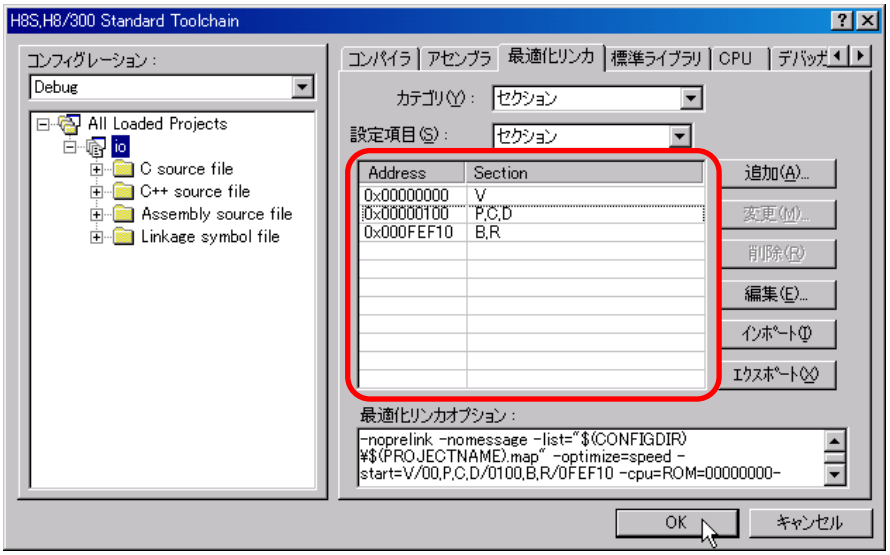
5.「c:¥workspace¥common¥initsct_3048.c」を選択、「相対パス」のチェックを外します。追加をクリックします。



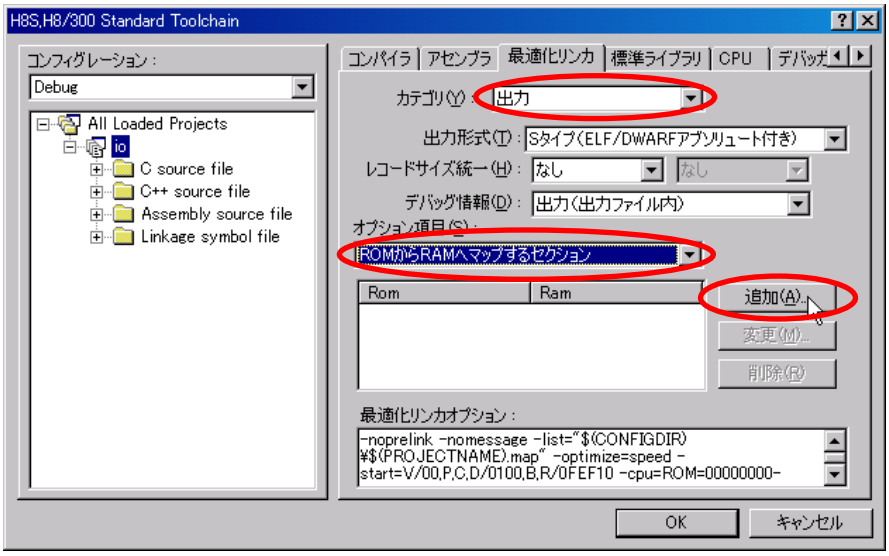
6. ファイルリストに「initsct_3048.c」が追加されました。



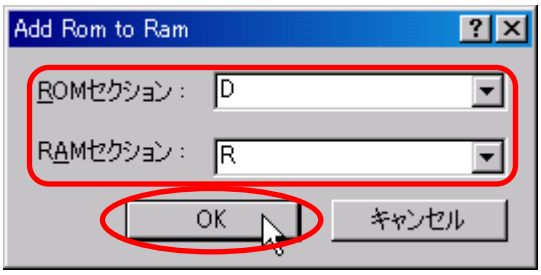
7. 「ビルド→H8S,H8/300 Standard Toolchain」を選択します。「最適化リンカ」を選択、「カテゴリ：**セクション**」を選択し、セクション一覧を表示させます。



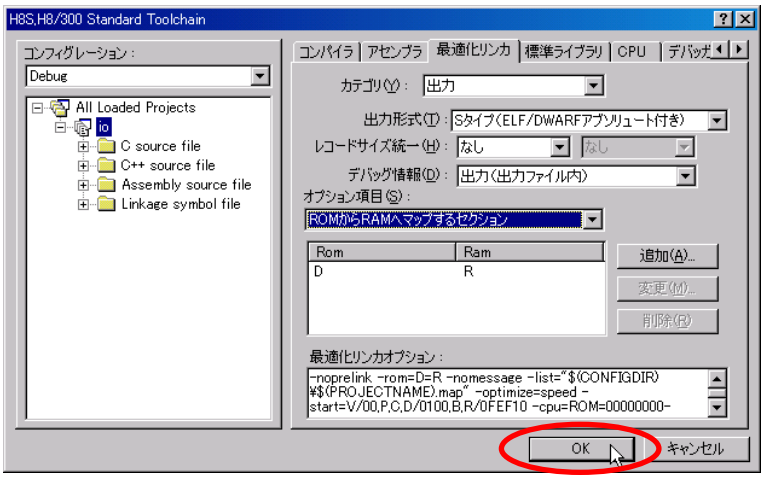
8. ここでセクション C,D,B,R を追加します。□のように設定します。



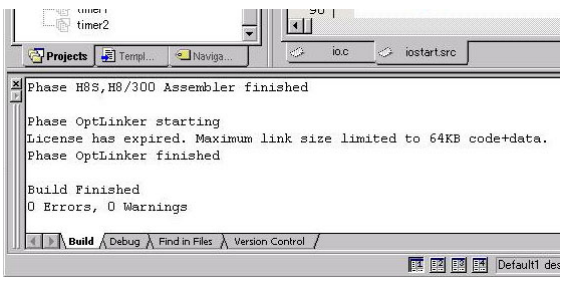
9.「カテゴリ:**出力**」、「オプション項目:**ROM から RAM へマップするセクション**」を選択します。追加をクリックして完了です。



10.「ROM セクション:**D**」、「RAM セクション:**R**」を選択します。OK をクリックします。



11. OK をクリックして、ツールチェインの設定を完了します。

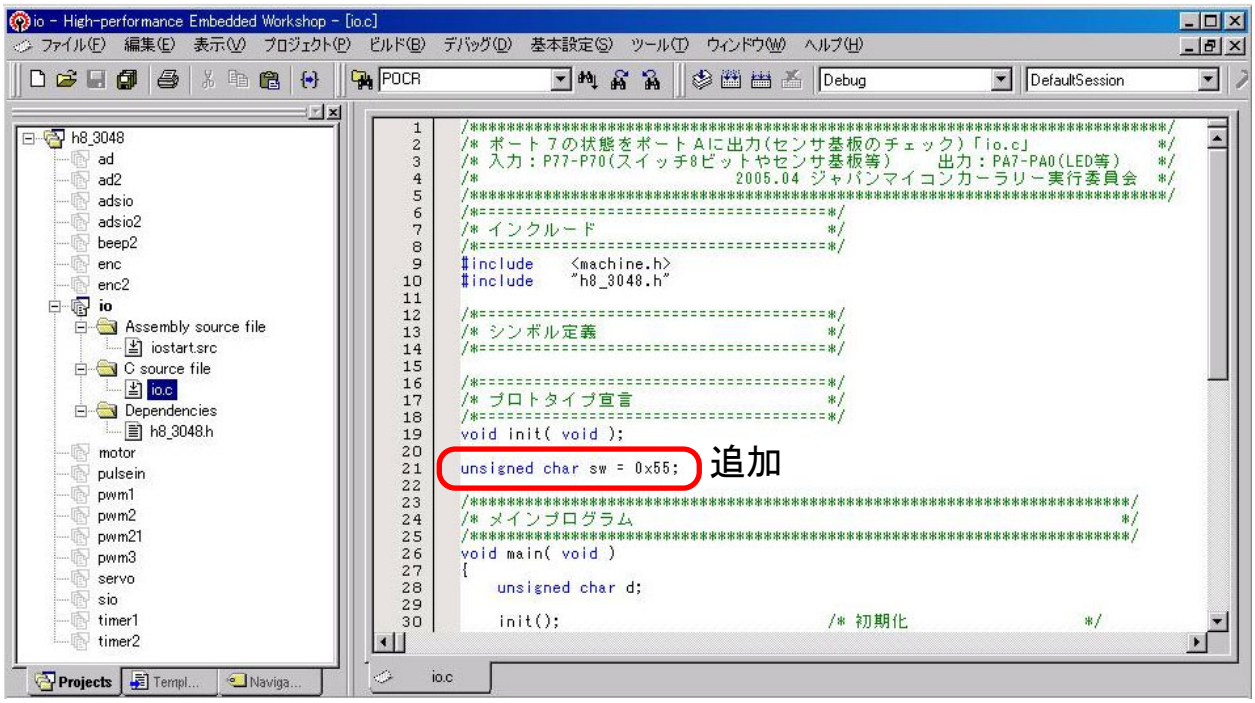


12. ビルドを実行します。エラーが無くなりました。

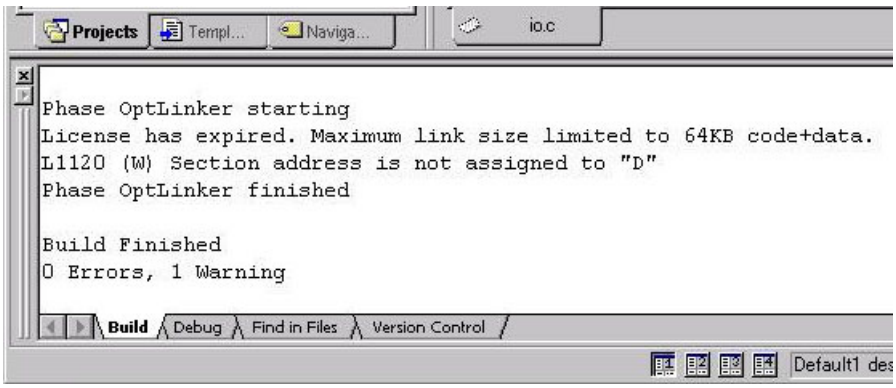
5.3　初期値のあるグローバル変数を追加したときの設定(セクションD,Rの追加)

ワークスペース「h8_3048」のプロジェクト「io」に初期値のあるグローバル変数を追加したとき、行わなければいけない作業を説明します。

5.3.1　追加内容



1. io.c の 21 行目に初期値のあるグローバル変数 sw を追加しました。



2. ビルドすると、エラーが出ました。F1 キーを押してエラーのヘルプを見ると「"セクション"のアドレス指定がありません。"セクション"を最後尾に配置します。」と出ました。セクション D が定義されていないのでツールチェインで設定してください、というエラーです。ツールチェインでセクション D を追加して、エラーを無くしましょう。

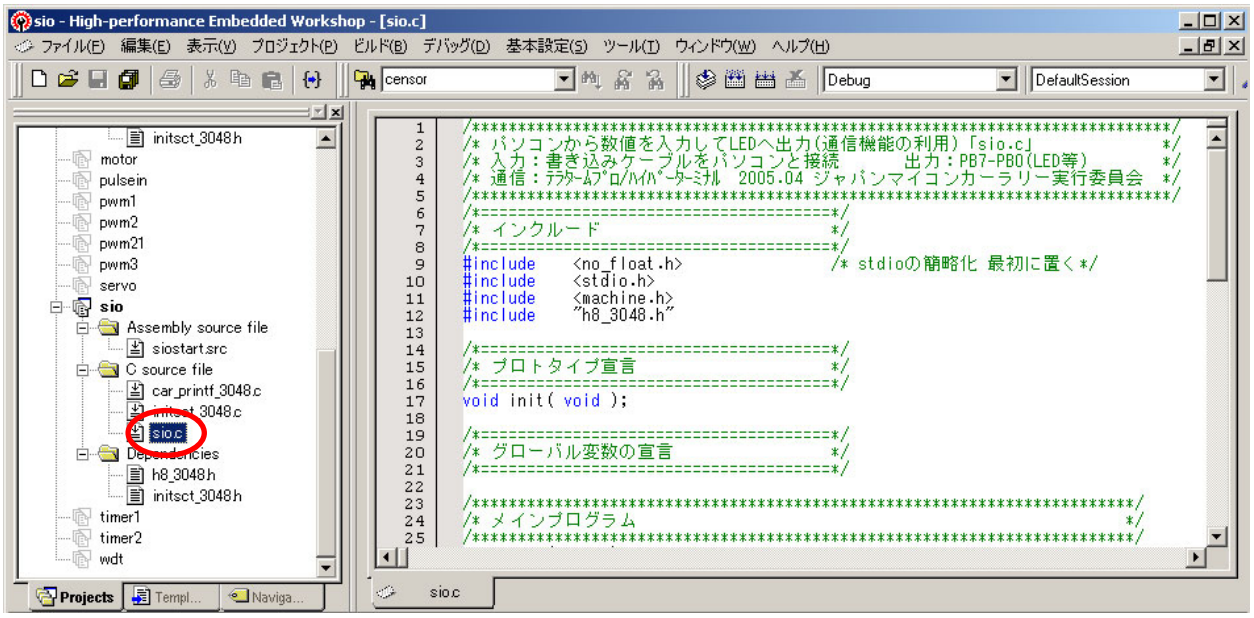
5.3.2　設定内容

「**5.2.2　設定内容**」と作業内容は同じです。

5.4　算術関数（標準ライブラリ関数）の使用

実行委員会開発環境で「sin、cos、tan は使えないの？」という質問が良くありました。ルネサス統合開発環境では使えるようになりました。ここでは、ワークスペース「h8_3048」のプロジェクト「sio」に cos 関数を追加する方法を説明していきます。

5.4.1　追加内容



1. ワークスペース「h8_3048」のプロジェクト「sio」をアクティブプロジェクトにします。「sio.c」をダブルクリックして、エディタウィンドウに「sio.c」を開きます。

```
/************************************************************************/
/* パソコンから数値を入力して LED へ出力(通信機能の利用)「sio.c」          */
/* 入力：書き込みケーブルをパソコンと接続        出力：PB7-PB0(LED など)    */
/* 通信：ﾃﾗﾀｰﾑﾌﾟﾛ/ﾊｲﾊﾟｰﾀｰﾐﾅﾙ  2005.04 ジャパンマイコンカーラリー実行委員会  */
/************************************************************************/
/*======================================*/
/* インクルード                         */
/*======================================*/
/*#include    <no_float.h>*/                /* stdio の簡略化 最初に置く */
#include    <stdio.h>
#include    <math.h>                        /* 算術関数を使用するためのヘッダ */
#include    <machine.h>
#include    "h8_3048.h"

/*======================================*/
/* プロトタイプ宣言                     */
/*======================================*/
void init( void );

/*======================================*/
/* グローバル変数の宣言                 */
/*======================================*/

/************************************************************************/
/* メインプログラム                                                     */
/************************************************************************/
void main( void )
{
    int     i, ret;
    float   f;

    /* マイコン機能の初期化 */
    init();                             /* 初期化                   */
    init_sci1( 0x00, 79 );              /* SCI1 初期化              */
    set_ccr( 0x00 );                    /* 全体割り込み許可         */

    printf( "Hello World!¥n" );
    while( 1 ) {
        printf( "Input data : " );
        ret = scanf( "%d", &i );
        if( ret == 1 ) {
            printf( "Get data : %d¥n", i );
            PBDR = i;
            f = cos( i * 3.14159 / 180 );
            printf( "cos値 : %f¥n", f );
        } else {
            printf( "Data Error!!¥n" );
            scanf( "%*[^¥n]" );
        }
    }
}
以下略
```

2. プログラムリストの、太字で□で囲った部分を追加します。「no_float.h」ファイルを無効にしています。詳細は下記を参照してください。

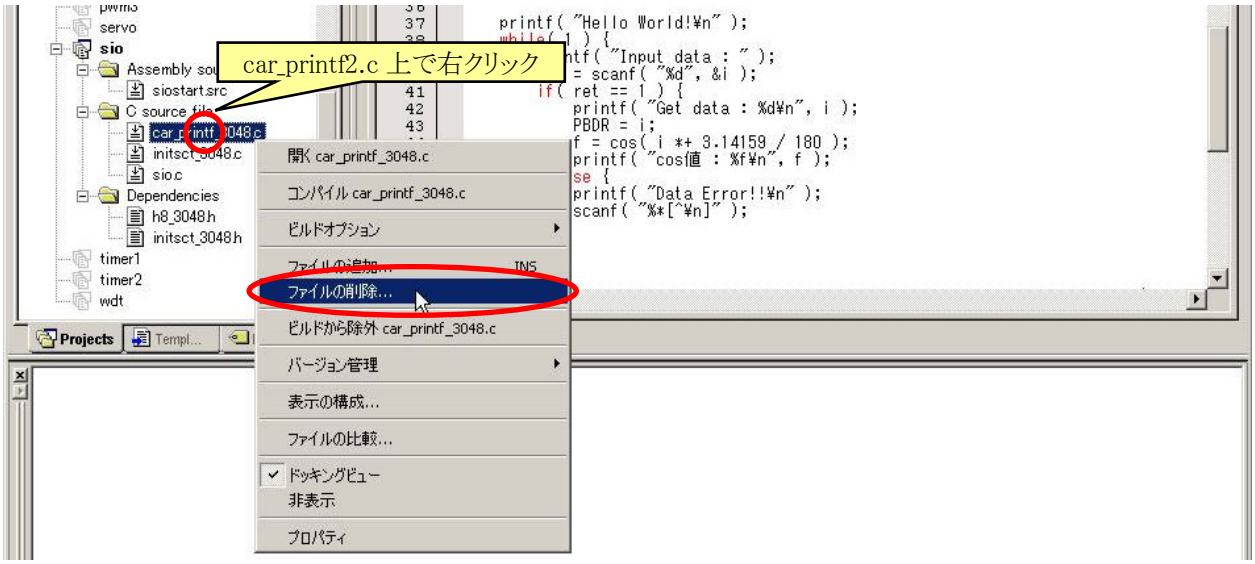
※参考資料ーヘッダファイル「no_float.h」について

```
/*#include      <no_float.h>*/                    /* stdio の簡略化  最初に置く*/
```

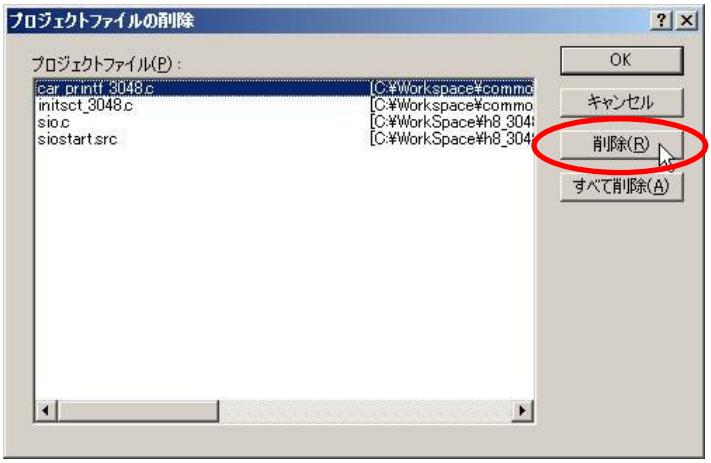
「sio.c」ファイルは、上記のように、「#include <no_float.h>」の前後に「/*　*/」を付けました。これは、
#include <no_float.h>
が無効になるということです。
このファイルは、ルネサス統合開発環境専用のヘッダファイルで、printf,scanf 文で float 型や double 型を使わなければ、stdio.h をインクルードする前に no_float.h をインクルードすることにより、MOT ファイルサイズを小さくすることができます。もし、**float 型や double 型を使用するのであれば、no_float.h を入れてはいけません**。
今回、float 型を使用するので、削除します。ただ、まるっきり削除してしまうと後でまた復活させたいときに、ヘッダファイル名を思い出すのが大変なので、コメント化しています。

「car_printf2.c」ファイルも「no_float.h」ファイルをインクルードしています。**プロジェクト内の全ファイルは、「no_float.h」ファイルを使用するかしないか統一しなければいけません**。そのため、このファイル内の記述もコメント化します。ただし、「car_printf_3048.c」ファイルは共通ファイルですので、他のプロジェクトからも参照されています。したがって、「car_printf_3048.c」ファイルの複製を作り、複製のファイルを編集するようにします。

5.4.2 設定内容

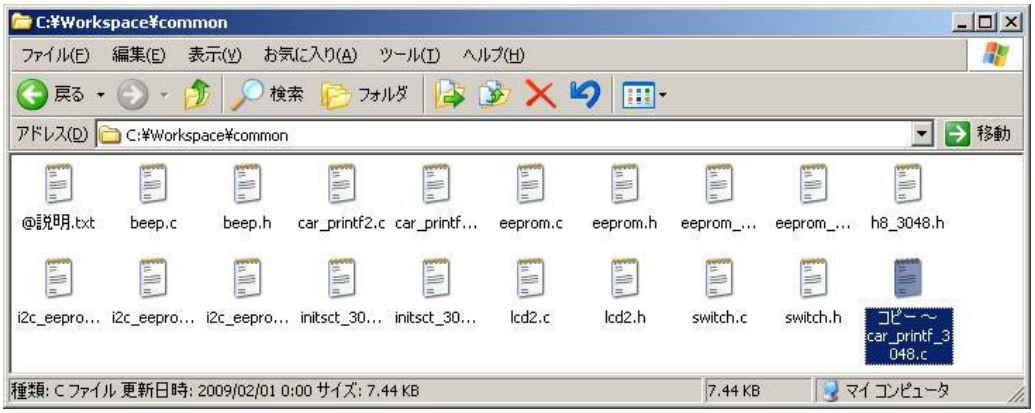


1.「car_printf_3048.c」上で右クリック、「ファイルの削除」を選択します。

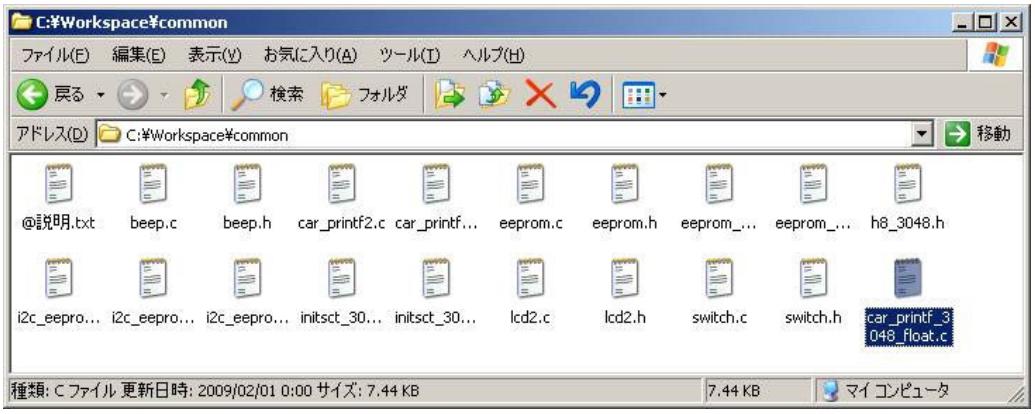


2.「car_printf_3048.c」を選択、削除をクリックします。OK をクリックして終了です。

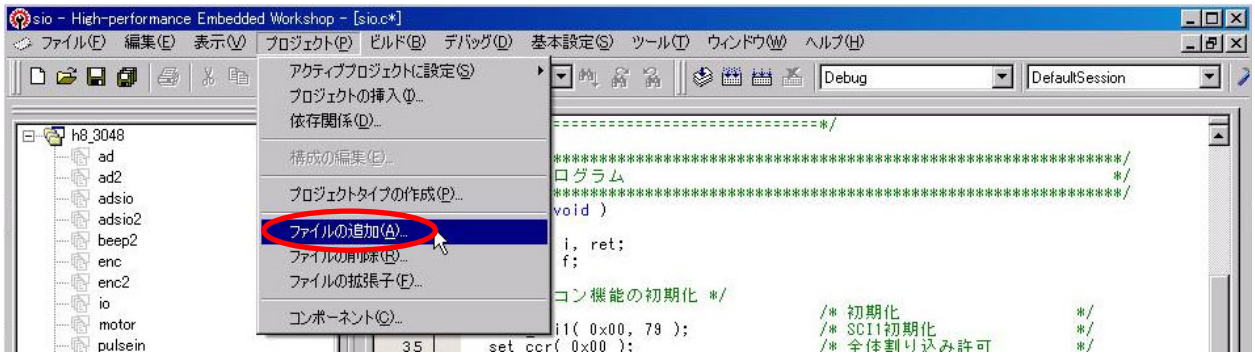
※実際にあるファイルの削除ではありません。リストから削除されるだけです。



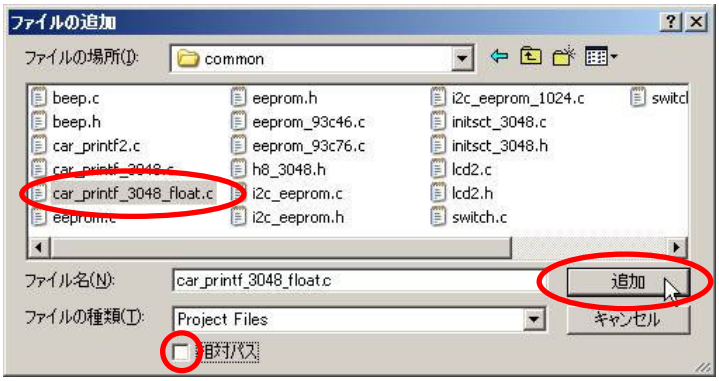
3.「C:¥WorkSpace¥common」フォルダを開きます。「car_printf_3048.c」をコピー、その場で貼り付けて「car_printf_3048.c」の複製を作ります。



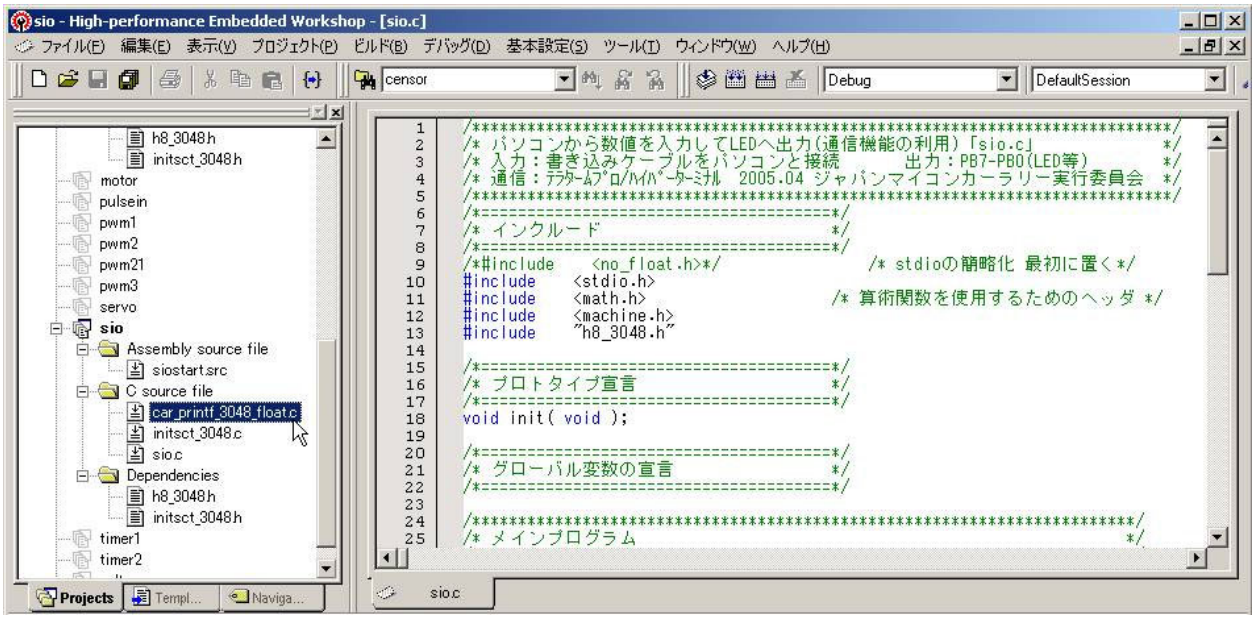
4. ファイル名を自分が分かるように変えます。ここでは、「**car_printf_3048_float.c**」とします。



5.「プロジェクト→ファイルの追加」を選択します。



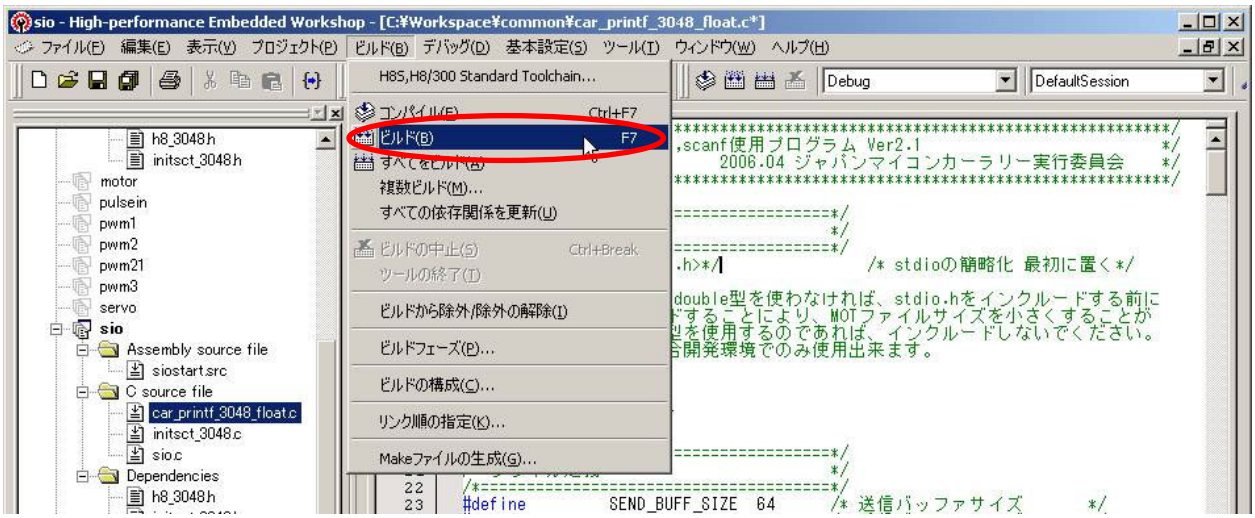
6.「C:¥WorkSpace¥common」フォルダへ移動します。「car_printf_3048_float.c」を選択、「相対パス」オプションを外して、追加をクリックします。



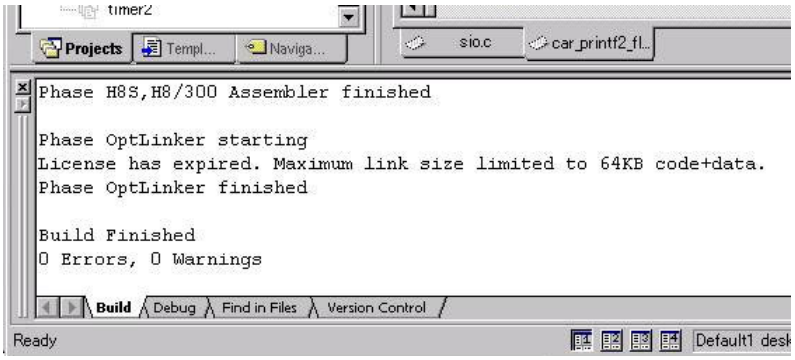
7.「car_printf_3048_float.c」をダブルクリックして、プログラムを開きます。

```
/*****************************************************************************/
/* マイコンカー用printf,scanf使用プログラム Ver2                               */
/*                          2006.04 ジャパンマイコンカーラリー実行委員会       */
/*****************************************************************************/

/*=====================================*/
/* インクルード                        */
/*=====================================*/
/*#include    <no_float.h>*/                  /* stdioの簡略化 最初に置く*/
/*
printf,scanf文でfloatやdouble型を使わなければ、stdio.hをインクルードする前に
no_float.hをインクルードすることにより、MOTファイルサイズを小さくすることが
出来ます。もし、double型を使用するのであれば、インクルードしないでください。
no_float.hはルネサス統合開発環境でのみ使用出来ます。
*/
#include    <stdio.h>
#include    <machine.h>
#include    "h8_3048.h"

以下略
```

8. 上記のように、「no_float.h」をコメント化します。これでプログラムの変更が完了です。



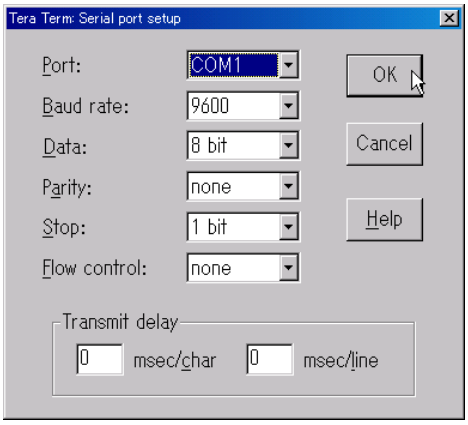
9.「ビルド→ビルド」でビルドします。



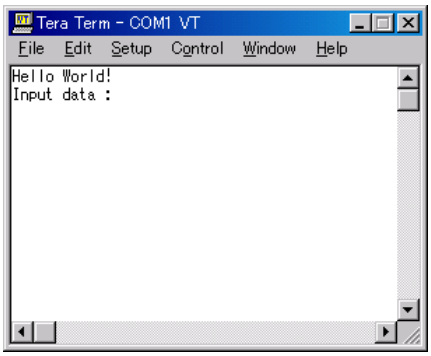
10. エラー0 です。作業完了です。

5.4.3 実行

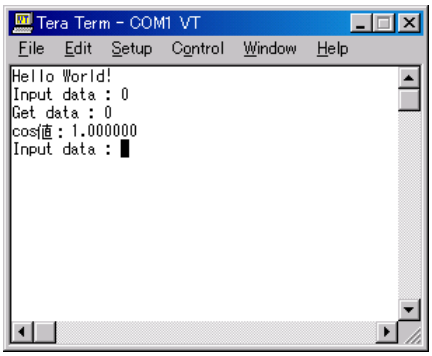
1. 実行してみます。「ツール→CpuWrite」で sio.mot を CPU ボードへ書き込みます。書き込み後、CPU ボードの電源は切っておきます。



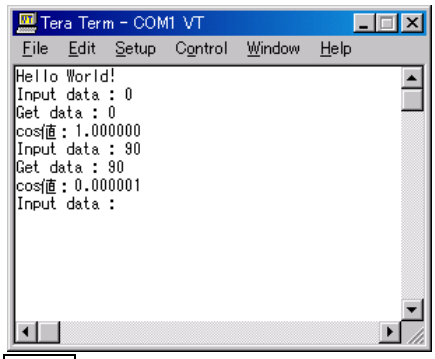
2. Tera Term Pro などの通信ソフトを立ち上げます。通信設定は、上画面のようにします。通信ポートのみ、それぞれの接続状態に合わせてください。



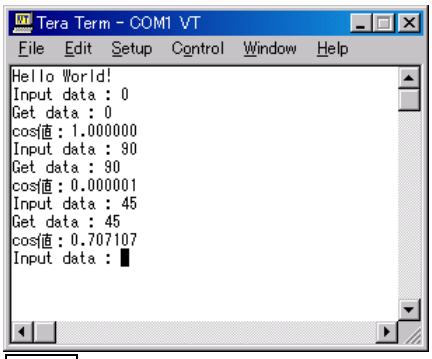
3. CPU ボードの電源を入れると、CPU ボードから送られてきた文字が表示さます。表示されない場合は、通信線の接続などを調べてください。



4. 0 エンタ を押します。cos0° は 1 ですので合っています。



5. 9 0 エンタ を押します。cos90° は 0 です。結果 0.000001≒0 なので合っています。



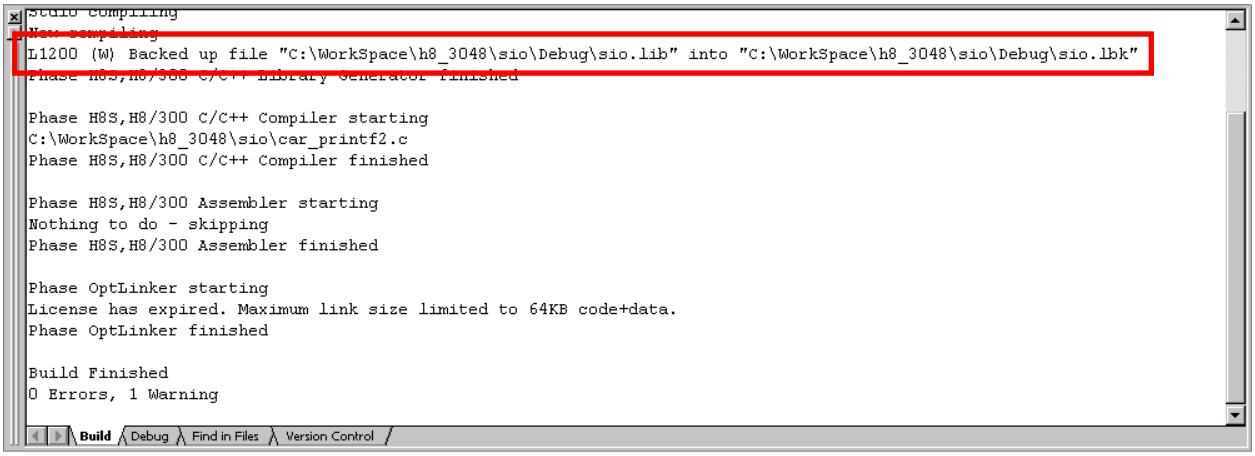
6. 4 5 エンタ を押します。cos45° は約 0.71 です。合っています。

このように、cos 関数を H8 マイコンで実行することができました。sin、tan などももちろん実行できますのでいろいろ試してみてください。

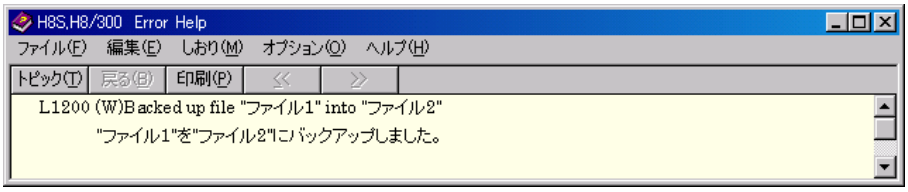
6. エラー対策

6.1 L1200 ライブラリファイルのバックアップ

ツールチェインのライブラリに関する設定を変えてビルドを実行すると、ライブラリファイル（拡張子が lib のファイル）を作り直します。このとき下記のようなワーニングが出力されます。



ヘルプで確認すると、下記のようなエラーです。

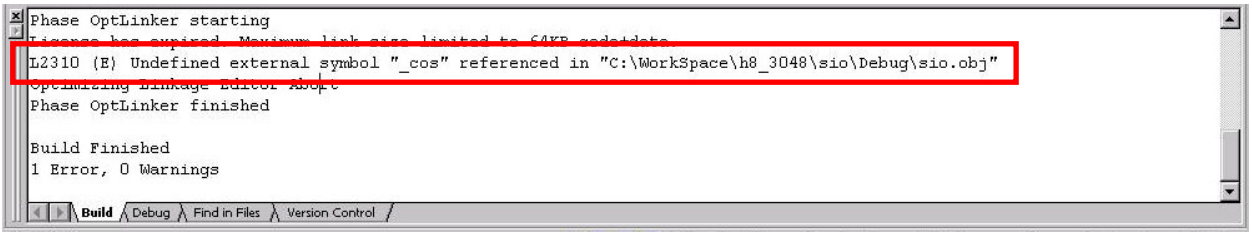


ライブラリファイルを作り直すとき、前のライブラリファイルの名前を変えたと言うことをワーニング出力するのです。このワーニングのみ、無視して構いません。

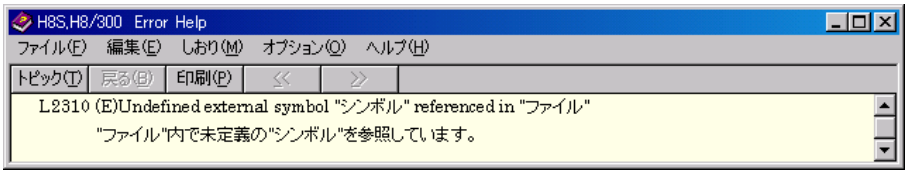
今回のエラーは、今ある「C:¥WorkSpace¥h8_3048¥sio¥Debug¥sio.lib」ファイルを「C:¥WorkSpace¥h8_3048¥sio¥Debug¥sio.lbk」という名前に変えて新しく sio.lib を作ったという内容です。

6.2 L2310 未定義のシンボル（標準ライブラリ関数）

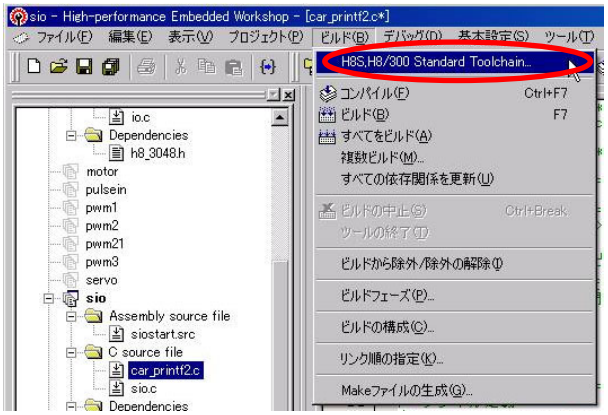
プログラムの中で記述のある関数が実際は無い場合、下記エラーが出力されます。



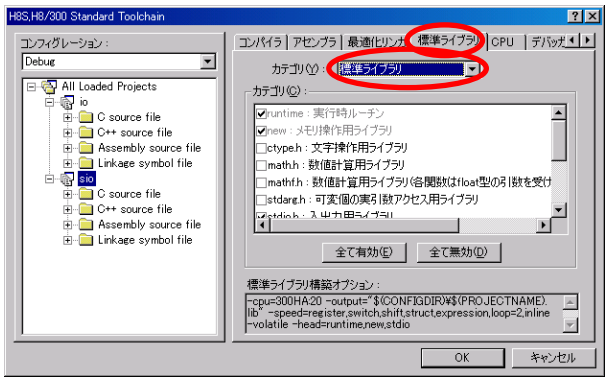
ヘルプで確認すると、下記のようなエラーです。



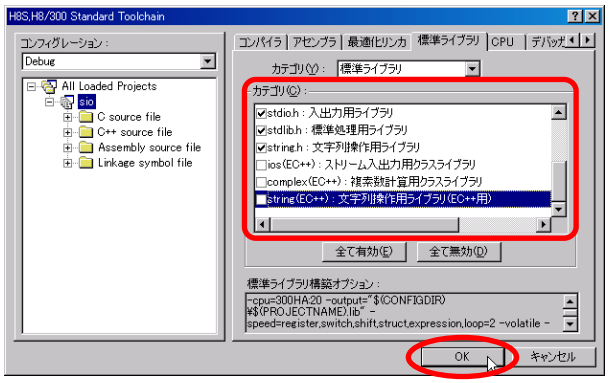
要は、「cos 関数が無い」ということです。**標準ライブラリ関数(標準のヘッダファイル)を使うときは、プログラムでヘッダファイルを追加すると共に、ツールチェインの設定も必要です。忘れがちなので気をつけます。**
設定しましょう。



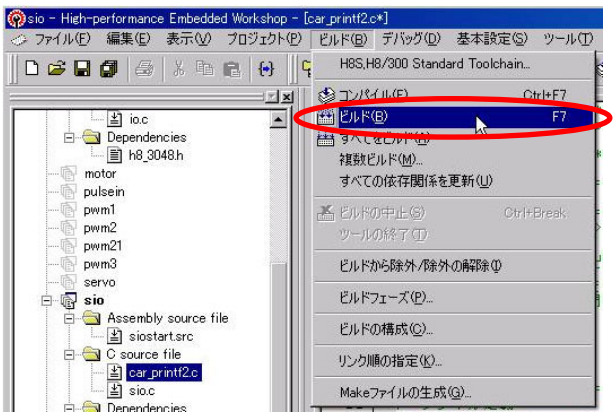
1.「ビルド→H8S,H8/300 Standard Toolchain」を選択します。



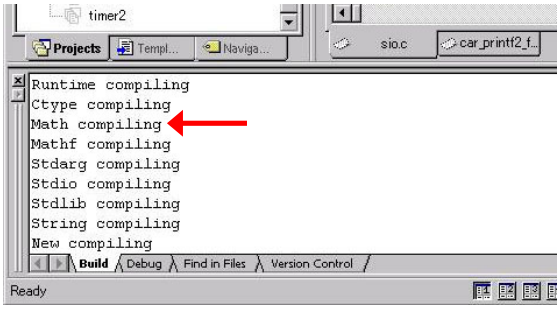
2.「標準ライブラリ」、「カテゴリ:**標準ライブラリ**」を選択します。



3. カテゴリにあるファイルにチェックを付けます。いちばん下の 3 ファイル(EC++とあるファイル)以外、すべてにチェックを付けます。OK をクリックして完了です。

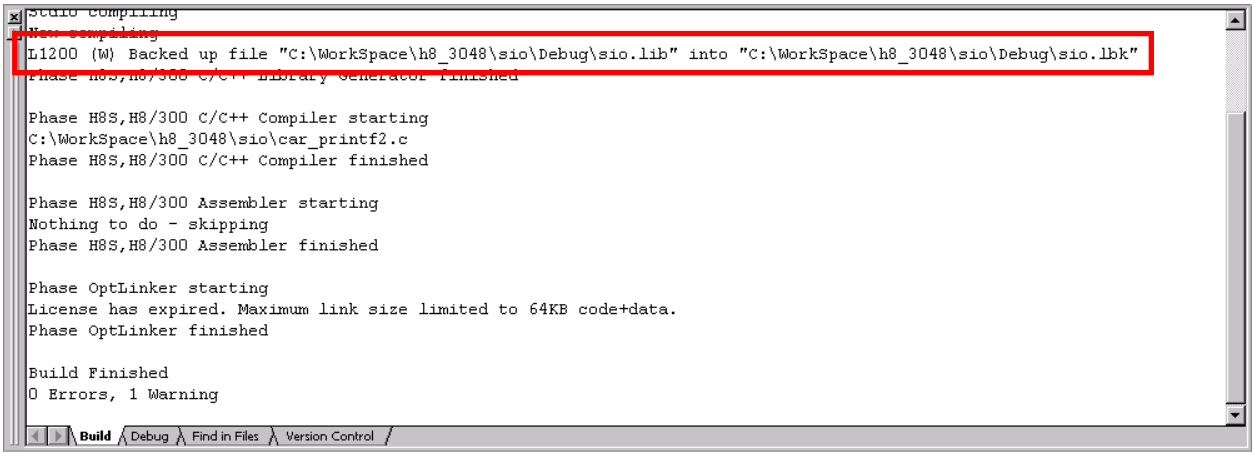


4.「ビルド→ビルド」を実行します。



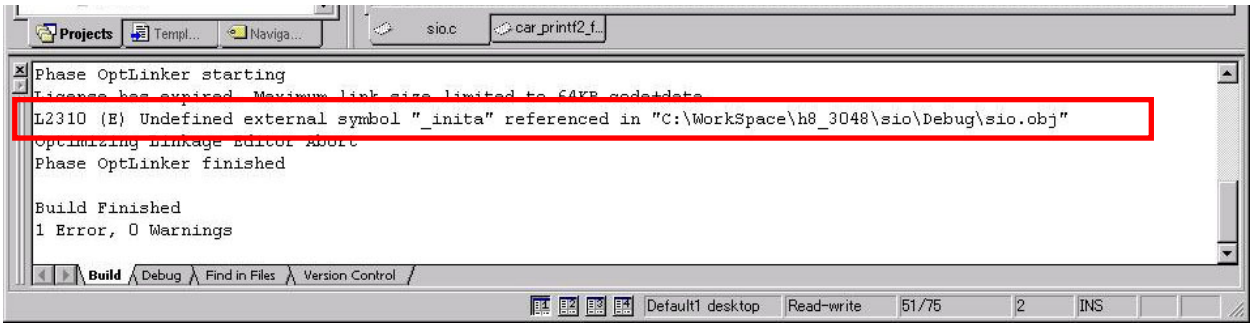
5. ビルド途中の画面です。「math.h」が変換されています。

6. ワーニングが1つ出てしまいました。

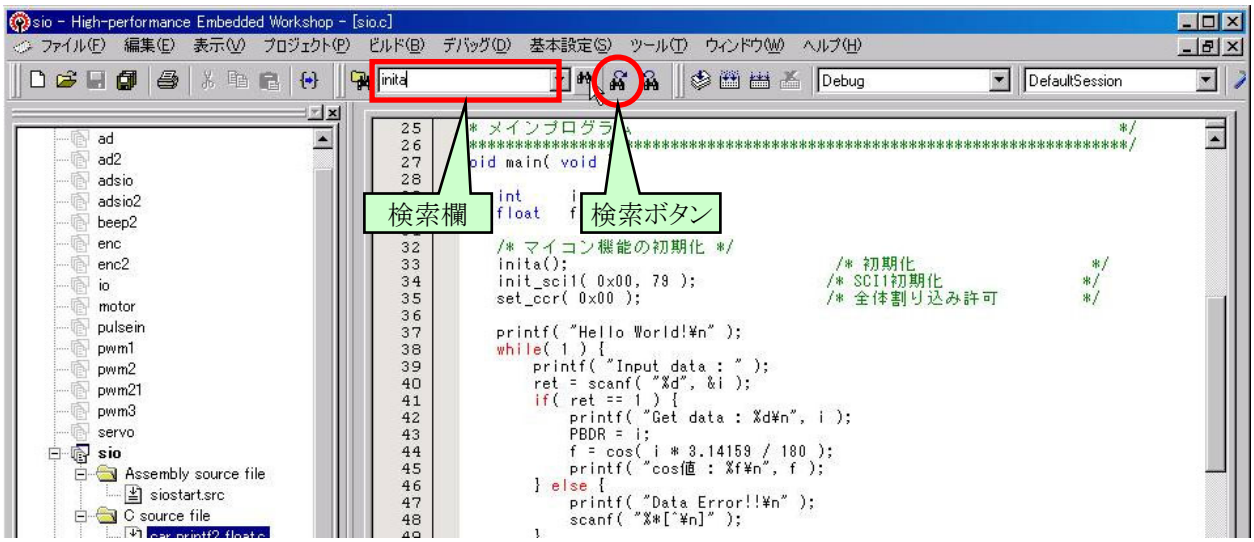


7.「6.1　L1200　ライブラリファイルのバックアップ」のワーニングです。このワーニングのみ、無視できます。

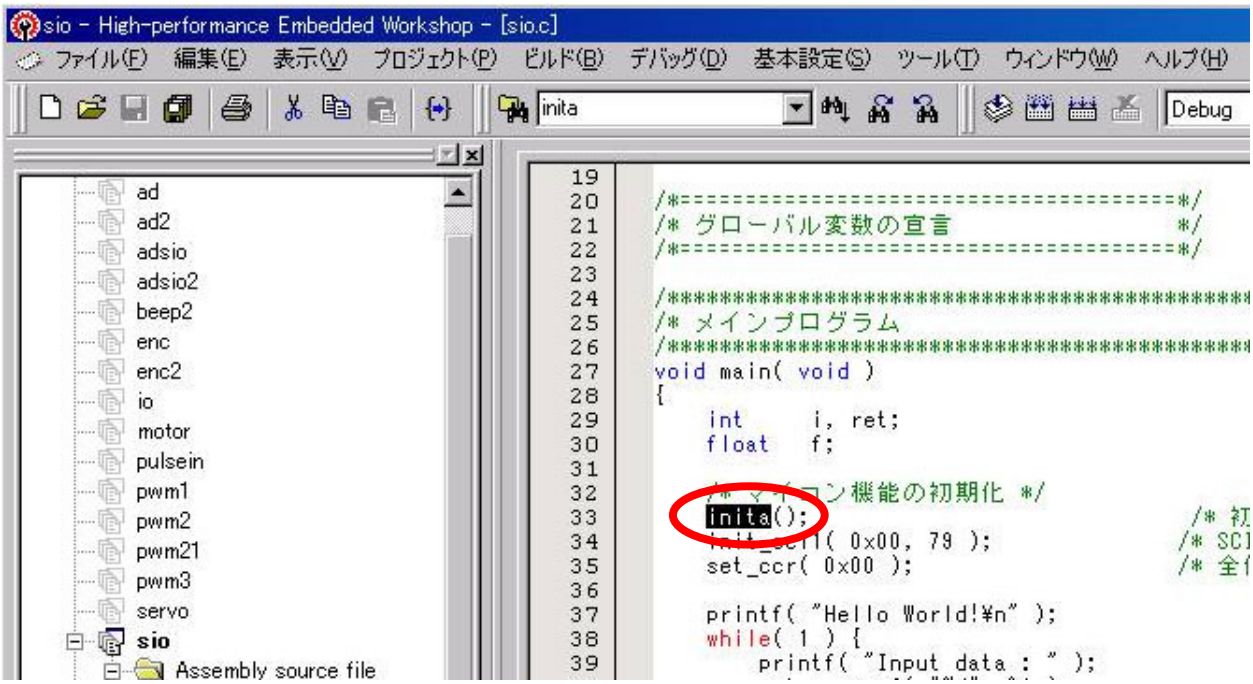
6.3　L2310　未定義のシンボル（自作の関数）



「inita 関数がない」というエラーです。この関数は標準ライブラリの関数ではありません。自作の関数です。



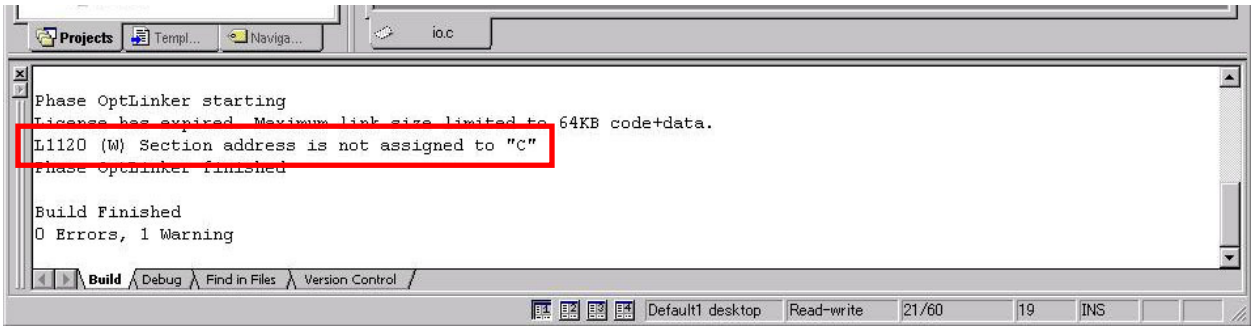
検索欄に関数名を入れ、検索ボタンをクリックします。



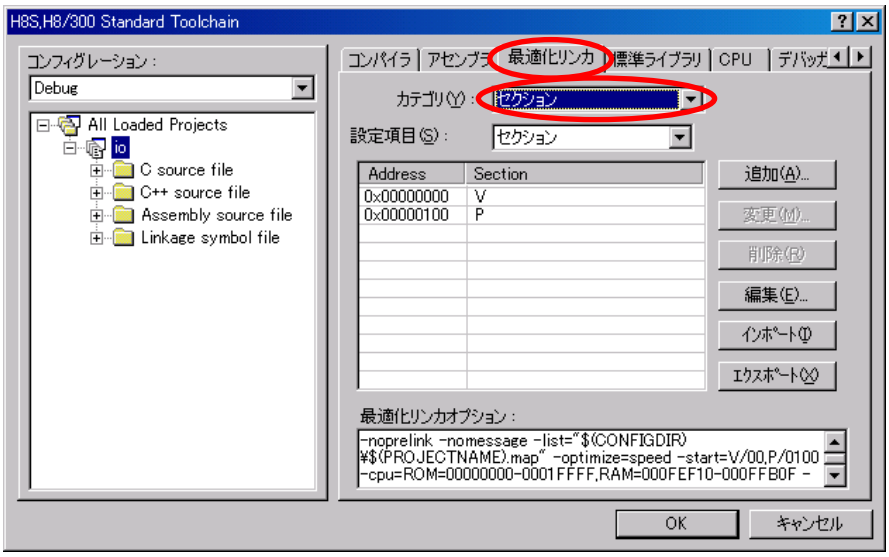
「inita」が検索されました。「init」の間違えでした。直してビルドします。

6.4 L1120 セクションの指定がない

ツールチェインでセクションの指定をしていないのに、プログラムでそのセクションを使っている場合、L1120 エラーが出力されます。

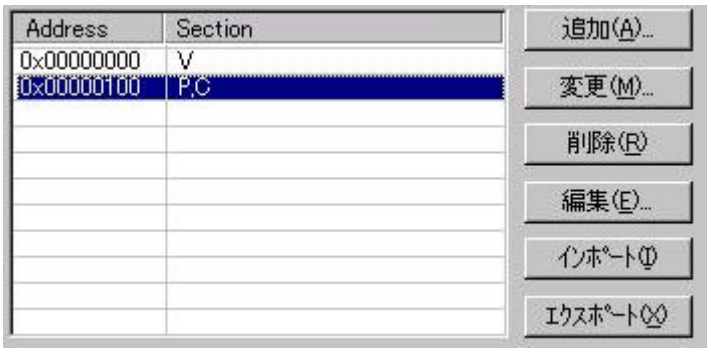


セクション C がツールチェインの設定にないというエラーです。ツールチェインでセクションを設定します。



「ビルド→H8S,H8/300 Standard Toolchain」を選択します。「最適化リンカ」を選択、「カテゴリ:**セクション**」を選択します。エラーの出たセクションを追加してください。

(1) セクションCがない場合



0x100 番地を「**P**」から「**P, C**」に変更してください。

(2) セクションBがない場合

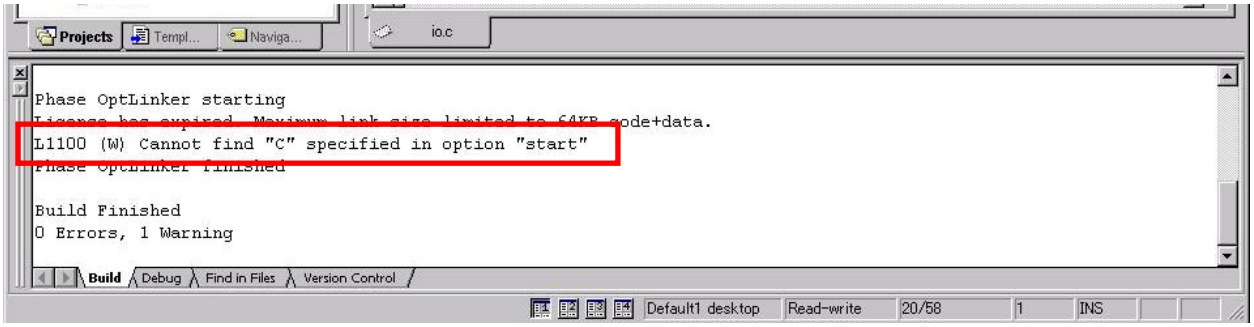
「5.2.2　設定内容」を参照してください。説明中の「iostart.src」は、現在編集中のプロジェクトの src ファイルと置き換えて設定を変更してください。

(3) セクションDがない場合

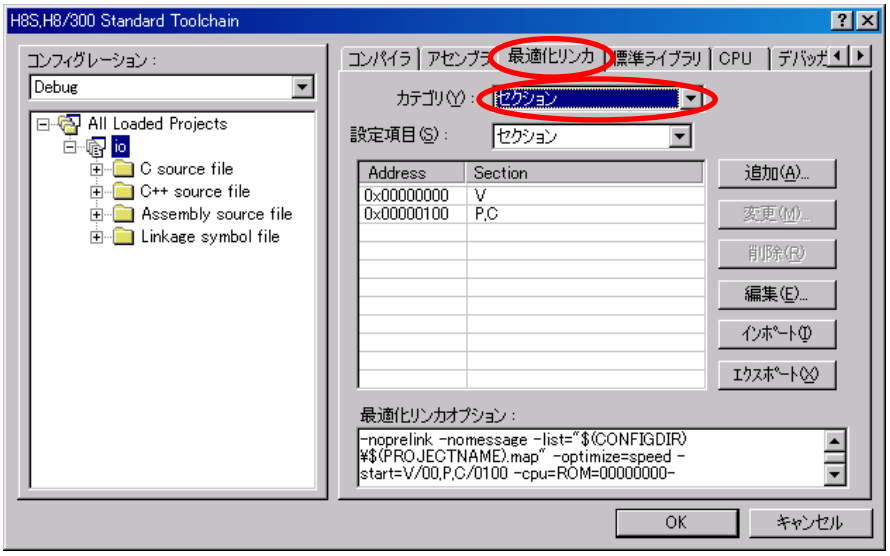
「5.2.2　設定内容」を参照してください。説明中の「iostart.src」は、現在編集中のプロジェクトの src ファイルと置き換えて設定を変更してください。

6.5　L1100　不要なセクションがある

ツールチェインでセクションの指定をしているのに、プログラムでそのセクションを使っていない場合、L1100 エラーが出力されます。先ほどの「**6.4　L1120　セクションの指定がない**」の逆バージョンです。

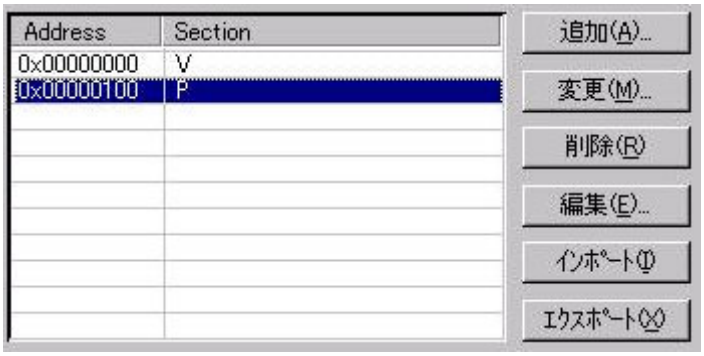


セクション C がツールチェインで設定されているのに、プログラムではセクション C がないというエラーです。ツールチェインでセクションを削除します。



「ビルド→H8S,H8/300 Standard Toolchain」を選択します。「最適化リンカ」を選択、「カテゴリ：**セクション**」を選択します。エラーの出たセクションを削除してください。

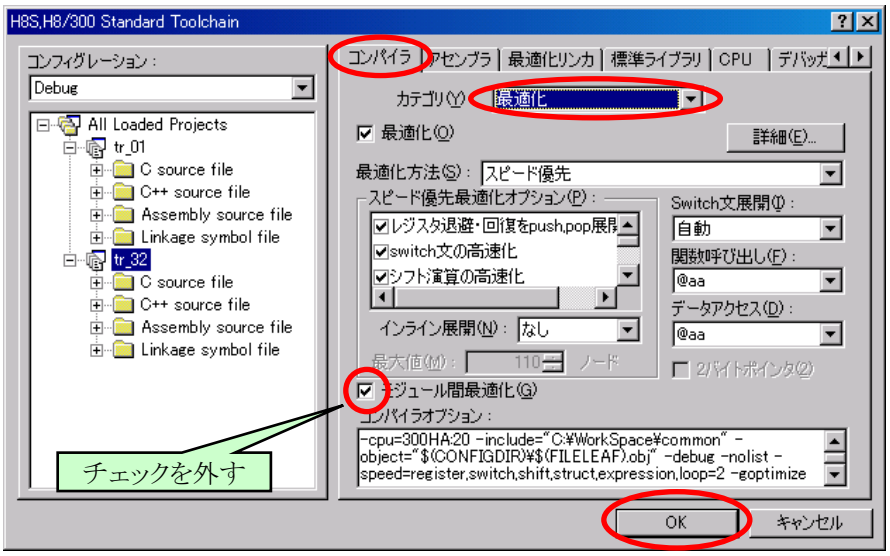
(1) ツールチェインのセクション設定からセクションCを削除する場合



0x100 番地を「**P, C**」から「**P**」に変更してください。

6.6 電源を切ってすぐに入れるとマイコンカー(または制御プログラムが)が暴走する

電源を切ってすぐに入れるとマイコンカーまたは制御プログラムが暴走することがあります。ツールチェインの設定を間違えると、この現象がおこる場合があります。



「ビルド→H8S,H8/300 Standard Toolchain」を選択します。「コンパイラ」を選択、「カテゴリ:**最適化**」を選択します。「モジュール間最適化」のチェックを確認します。レ が付いている場合は**外します**。OK をクリックして完了です。

6.7 ワークスペースが開けない

ルネサス統合開発環境で、ワークスペースを開こうとしたとき、下記のようなメッセージがでました。

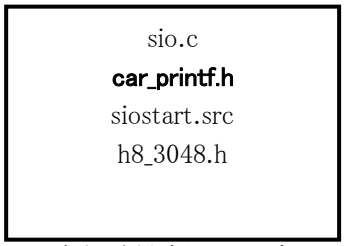
```
Tool Chain 'Hitachi H8S,H8/300 Standard Toolchain' version '6.1.2.0' is missing
from the following project
```

このメッセージは、ルネサス統合開発環境のバージョンが古い場合、出てきます。最新のルネサス統合開発環境をダウンロードして、再インストールしてください。

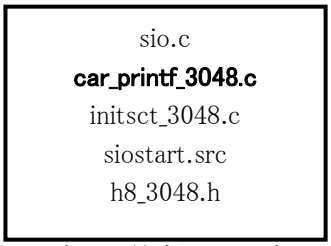
7. 補足

7.1 ヘッダファイルの取り扱い

実行委員会開発環境には「car_printf.h」というファイルがありました。今回のワークスペース「h8_3048」のプロジェクト「sio」では「car_printf2.c」というファイルがありますが、「car_printf.h」はありません。

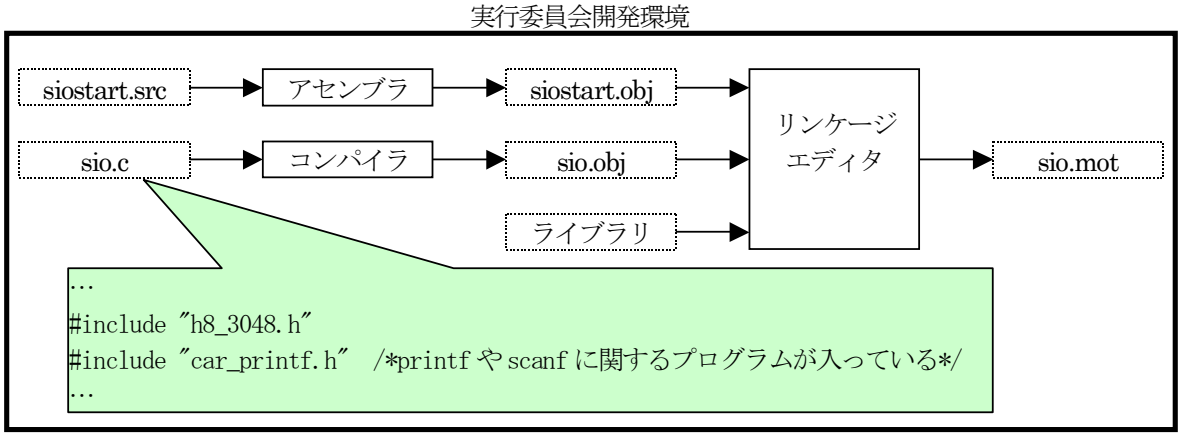


実行委員会開発環境



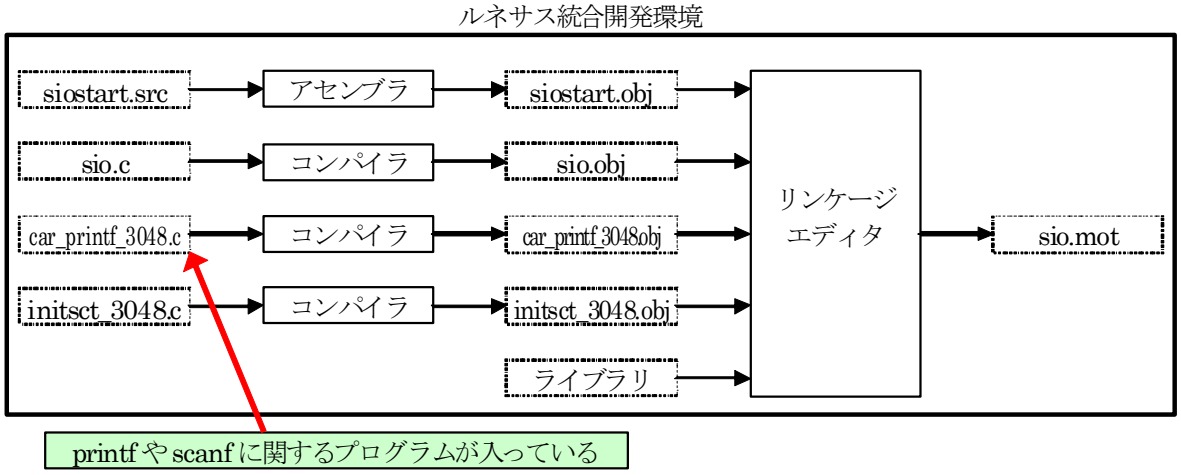
ルネサス統合開発環境

この違いはどうなっているのでしょうか。プロジェクト「sio」を例にします。
実行委員会開発環境は、下図のように MOT ファイルを作ります。



実行委員会開発環境では、「car_printf.h」ファイルを sio.c 内でインクルードして使用していました。
実はヘッダファイルの一般的な使われ方は、**「プログラムで使用する関数や変数やデータ型等の定義を行うファイル」という位置づけです。ヘッダファイルにプログラムを入れるのは一般的な方法ではありません。**
ちなみに、h8_3048.h は define 定義をしている内容なので本来の使い方です。なぜ実行委員会開発環境では、一般的ではない方法を使ったのでしょうか。実行委員会開発環境のファイルの構成は、
・C ファイル1つ
・src ファイル1つ
・sub ファイル 1 つ
の 3 ファイル構成しかできず、C ファイルを追加することができません。これは実行委員会開発環境の構造上の問題です。そのため、何とかできないかと考えた末、プログラムをヘッダファイルとして用意し、「sio.c」でインクルードすれば C ファイルは1つですみます。このようにして、printf、scanf 文を実行できるようにしています。

ルネサス統合開発環境では、このような制約がありません。printf や scanf に関するプログラムを記述した C ソースファイル「car_printf_3048.c」を作り、ルネサス統合開発環境に登録、ビルドします。



7.2 型の範囲

char 型や int 型や long 型は、値の範囲がどれくらいか分かっている方も多いと思います。では、float 型や double 型はどれくらの値の範囲なのでしょうか。
C 言語の標準の規格である、ANSI C では下記のように決められています。

・char は 1 バイト
・short int <= int <= long
・float <= double <= long double

そのため、char 以外の型のサイズは、コンパイラによって違います。H8 マイコンのコンパイラは int 型は 2 バイトですが、SH マイコンのコンパイラでは int 型は 4 バイトです。下記の値の範囲は、H8 マイコンのコンパイラのみ対応です。他の種類のマイコンは違う可能性が高いので必ず確認してください。

・整数型（H8マイコンのコンパイラの場合）

型	値の範囲	データサイズ
char （signed char 型として扱われる）	-128 ～ 127	1バイト
signed char	-128 ～ 127	1バイト
unsigned char	0 ～ 255	1バイト
short	-32768 ～ 32767	2バイト
unsigned short	0 ～ 65535	2バイト
int	-32768 ～ 32767	2バイト
unsigned int	0 ～ 65535	2バイト
long	-2147483648 ～ 2147483647	4バイト
unsigned long	0 ～ 4294967295	4バイト

・実数型（H8マイコンのコンパイラの場合）

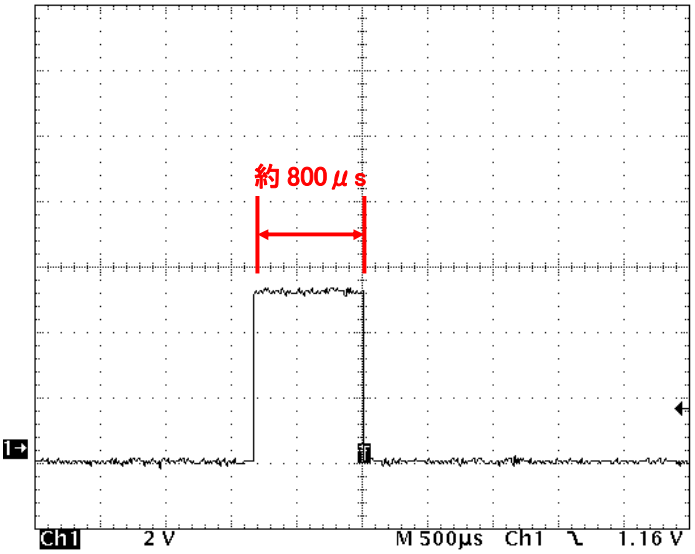
型	データサイズ	限界値	
		最大値	正の最小値
float	4バイト	3.4028235677973364e+38f (0x7F7FFFFF)	7.0064923216240862e-46f (0x00000001)
double long double	8バイト	1.7976931348623158e+308 (0x7FEFFFFFFFFFFFFF)	4.9406564584124655e-324 (0x0000000000000001)

7.3　三角関数を使ったプログラムの実行速度

ルネサス統合開発環境では、算術関数など、パソコンの C 言語で使用できる関数が使用できるようになりました。例えば、三角関数である sin、cos、tan を多用したプログラムをマイコンカーに組み込むとどうなるでしょう…
cos 関数の実行時間を測ってみます。

```
void main( void )
{
    int     i, ret;
    float   f;

    /* マイコン機能の初期化 */
    init();                             /* 初期化                   */
    init_sci1( 0x01, 19 );              /* SCI1初期化               */
    set_ccr( 0x00 );                    /* 全体割り込み許可         */

    printf( "Hello World!¥n" );
    while( 1 ) {
        printf( "Input data : " );
        ret = scanf( "%d", &i );
        if( ret == 1 ) {
            printf( "Get data : %d¥n", i );
            PBDR = i;
            PADR = 0xff;                    /* 端子を"1"に */
            f = cos( i * 3.14159 / 180 );
            PADR = 0x00;                    /* 端子を"0"に */
            printf( "cos値 : %f¥n", f );
        } else {
            printf( "Data Error!!¥n" );
            scanf( "%*[^¥n]" );
        }
    }
}
```

このように、cos 関数の実行前にポートAの端子を"1"に、実行後にポートAの端子を"0"にして、オシロスコープで端子の状態を測定してみました。



測定の結果、cos 関数1つ実行するのに約 800μs かかりました。

実際のマイコンカー走行プログラムでは、算術関数を多用すると思います。実行時間の長い関数が多くなり、プログラムが一巡(1 ループ)するまで時間がかなりかかってしまいます。
例えば、
・プログラムが一巡するのに 10ms
・マイコンカーが秒速 5m/s
とします。
10ms 間にマイコンカーが進む距離は、
5×0.01＝0.05m＝5cm
プログラムが一巡する 10ms の時間で 5cm も進んでしまいます。5cm 間隔でしかセンサを見ないということです。
これでは、きめ細かな制御ができません。

そのため、マイコンカーのようにきめ細かな制御を行うとき(リアルタイムに制御を行うとき)は、**マイコンでは浮動小数点演算は使わないというのが定石です**。もし使いたい場合は、テーブル処理といってあらかじめ計算しておいて、それを参照するなどします。

どうしても使わなければいけないというときは、SH マイコンなど浮動小数点演算機能のある高性能マイコンを使います。そうすれば数μ～数十μs で計算できます。

ちなみに、「kit05.c」では、プログラムが一巡する時間は 100μs 以下です(割り込みがかかっていないとき)。実際に計測してみるのも面白いでしょう。

7.4 ツールチェインの設定をサイズ優先設定にする

ツールチェインの設定で「スピード優先」、「サイズ優先」を選択することができます。それぞれ下記のような特徴があります。

内容	スピード優先	サイズ優先
プログラムの実行速度	速い	遅い （といっても通常は問題にならない速さです）
プログラムサイズ	大きい	小さい

今までの設定は「スピード優先」でした。なぜそうしていたのでしょうか。これは、

・H8/3048F-ONE マイコンの ROM は 128KB あるので、プログラムサイズが大きくなったとしてもこのサイズを超えることはほとんど考えられない
・実行速度が速いに越したことはない

といった理由です。このマニュアルでは、ルネサス統合開発環境**無償評価版**を使用しています。無償評価版の制限は、

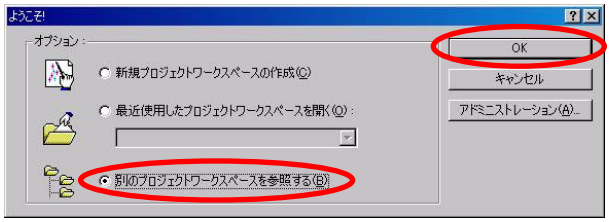
・インストール後、60 日たつと 64KB 以下のサイズしかリンクできない
・サポートがない

という内容です。大きいプログラムを作成した場合は、64KB を超えてしまうかもしれません。そこで、「サイズ優先」の設定方法を紹介します。

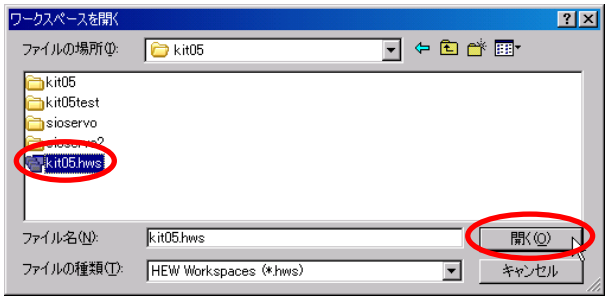
7.4.1 ツールチェインの設定

ここでは、マイコンカー標準プログラム「kit05.c」を使用したいと思います。

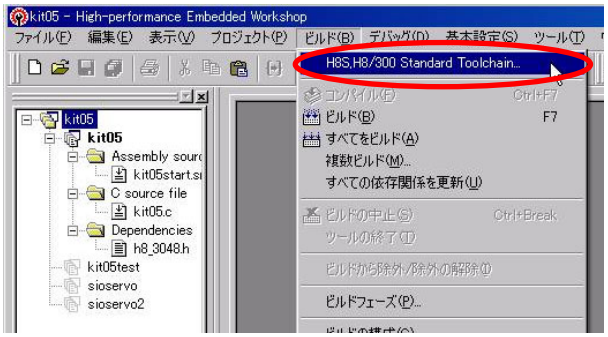
1. ルネサス統合開発環境を実行します。



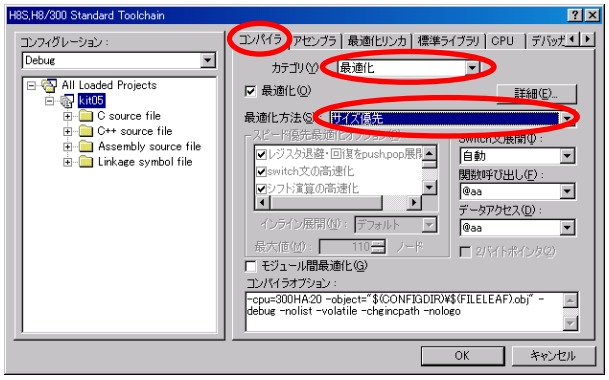
2.「別のプロジェクトワークスペースを参照する」を選択します。OK をクリックします。



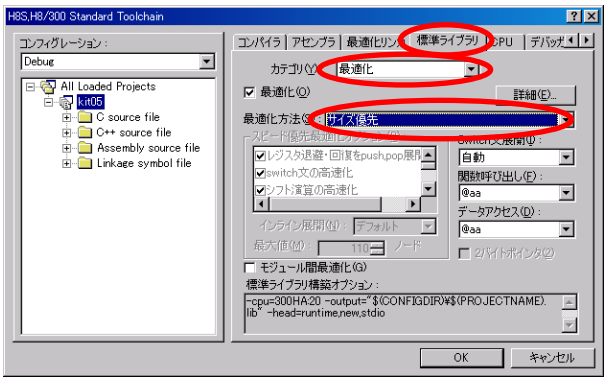
3. Cドライブ→Workspace→kit05 の「kit05.hws」を選択します。開くをクリックします。



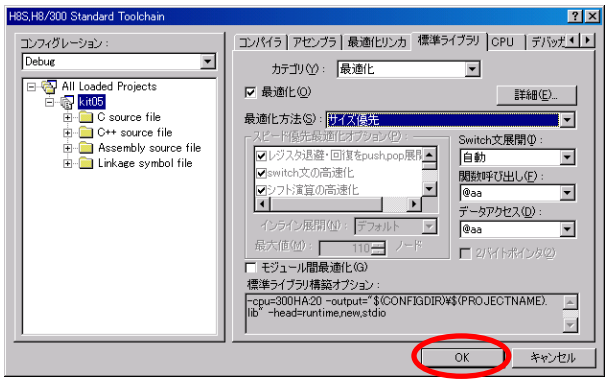
4.「ビルド→H8S,H8/300 Standard Toolchain」を選択します。



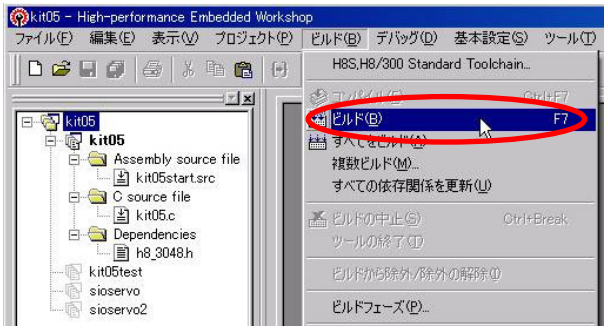
5.「コンパイラ」、「カテゴリ:**最適化**」、「最適化方法:**サイズ優先**」を選択します。



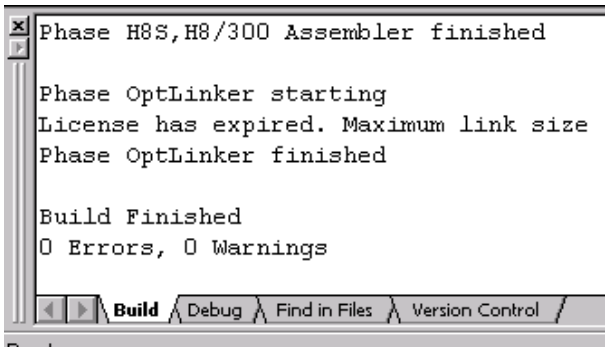
6.「標準ライブラリ」、「カテゴリ:**最適化**」、「最適化方法:**サイズ優先**」を選択します。



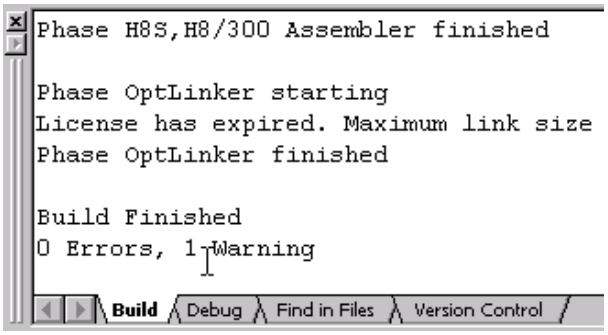
7. OK をクリックして完了です。



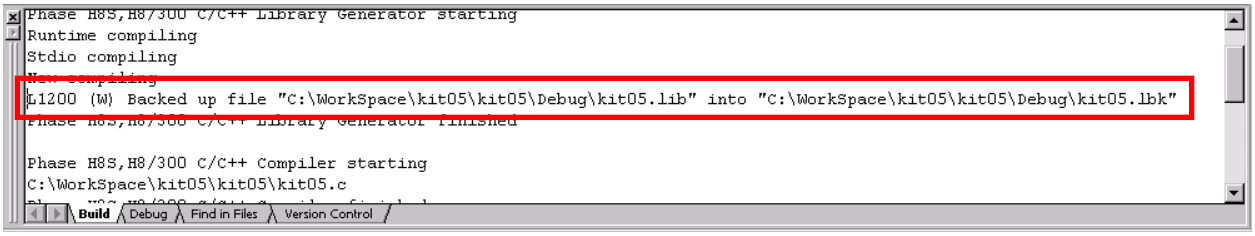
8.「ビルド→ビルド」でビルドします。



9. ビルドできました。



10. もしくはワーニングが 1 つ出ることがあります。



11. ビルド画面の上の方を見ると、「既に kit05.lib ファイルがあったのでそのファイルを kit05.lbk として、新しく kit05.lib を作りました」というメッセージが出力されました。通常、ワーニングが出た場合はその原因を追及して直さなければいけませんが、このメッセージのみ無視できます。

7.4.2　サイズと実行スピード

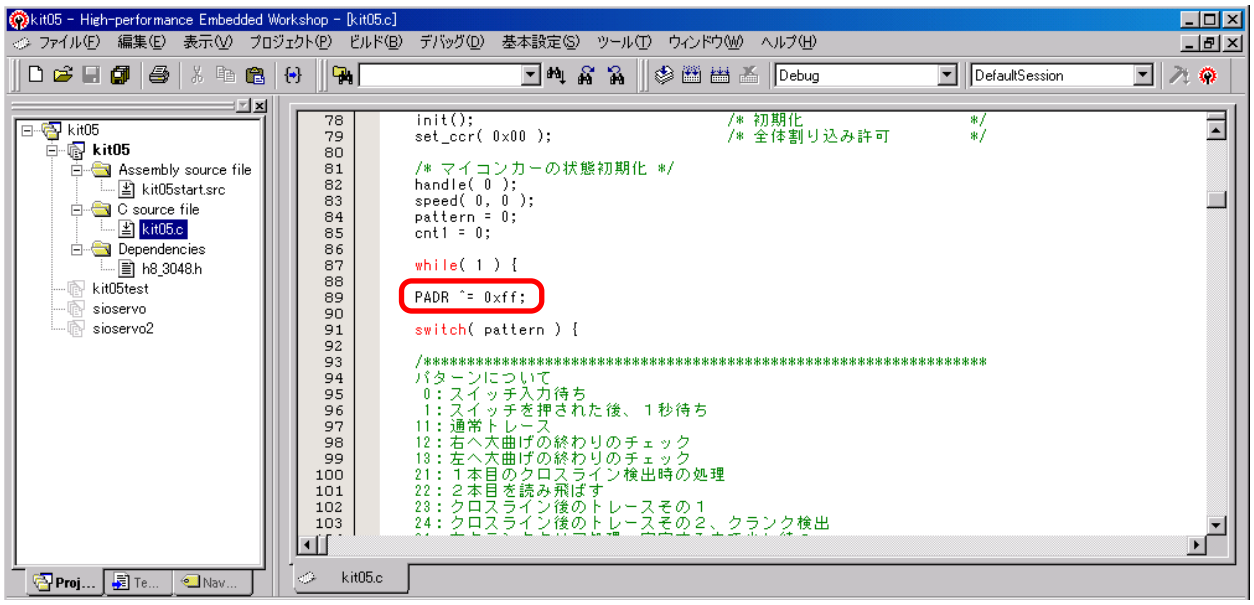
これで「kit05.mot」ができました。MOT ファイルを調べた結果、プログラムサイズは下記のようになりました。

内容	スピード優先	サイズ優先
プログラムサイズ	0x0a7b =2683 バイト	0x079d =1949 バイト

サイズ優先／スピード優先＝1949／2683＝0.73＝73%

サイズ優先の方が、スピード優先より 73%サイズが小さくなっています。

では実行スピードはどうでしょうか。下記のようにプログラムを追加します。



1 ループごとにポート A の端子を反転させます。その信号をオシロスコープで測定すれば、実行時間を測定することができます。
ちなみに、「^=」は xor 演算子で排他的論理和と言います。排他的論理和については、インターネットなどで検索すればたくさん出てきますので調べてみてください。

測定結果は下記のようになりました。「kit05.c」は、パターン番号によって実行されるプログラムが違います。そこで、代表的な、

・パターン 0: 待機状態
・パターン 11: 通常走行状態

を測定しました。

プログラムの実行速度	スピード優先	サイズ優先
パターン 0 のとき	約 5μs	約 7.5μs
パターン 11 のとき	約 20μs	約 60μs

※ただし割り込みが実行されていないときの1ループ時間

パターン 0 の時: サイズ優先／スピード優先＝7.5／5＝1.5＝150%
パターン 11 の時: サイズ優先／スピード優先＝60／20＝3.0＝300%

パターン 0 では 1.5 倍、パターン 11 ではなんと 3 倍も実行スピードが速くなっています。

7.4.3 まとめ

サイズ優先、スピード優先設定のどちらを選ぶかは、プログラマの自由ですが、プログラムサイズに問題がなければ、スピード優先が良いでしょう。

7.5 Cソースファイルがアセンブリソースファイルに変換されたリストを見る

アセンブリソースファイルは、アセンブラという翻訳プログラムによってオブジェクトファイル（機械語ファイル）に変換されます。アセンブリソースファイルは、プログラムと機械語が1対1です。要は、

```
mov.b #0,r0l
```

とプログラムすると、

```
F8 00
```

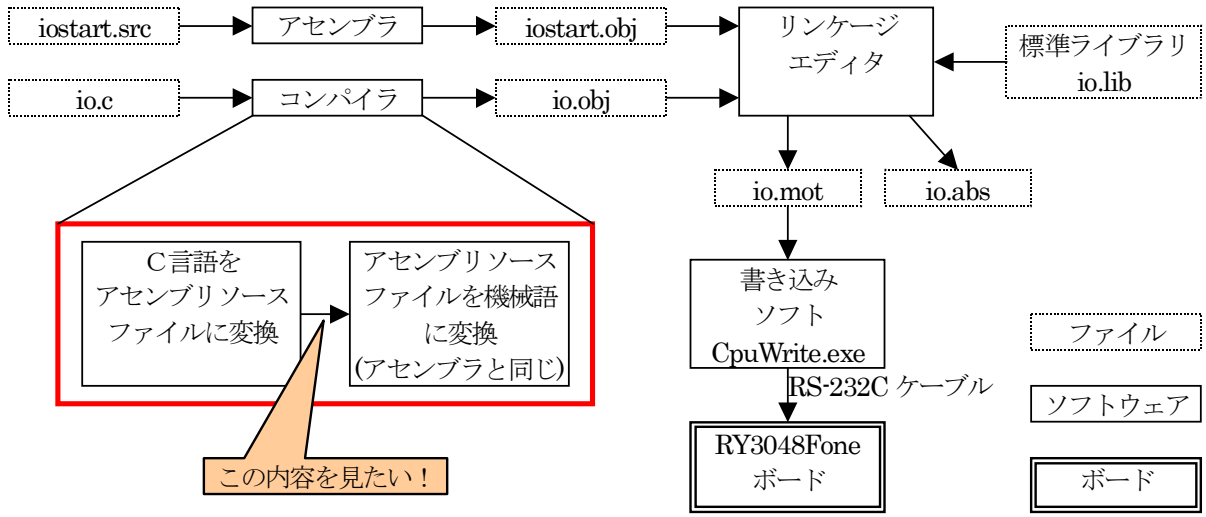
と機械語に変換されます。

Cソースファイルは、コンパイラという翻訳プログラムによってオブジェクトファイル（機械語ファイル）に変換されます。しかし、どう変換されるかは分かりません。C言語は高級言語と呼ばれる言語で、人間に分かりやすい変わりに、C言語1行で、たくさんの機械語に変換されるためです。例えば、

```
a = 0;
```

という命令があっても、「a=0」という命令に直接対応する機械語がありません。

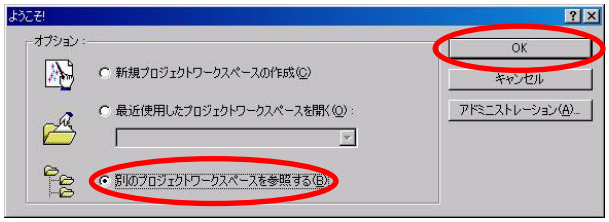
といっても、コンパイラはC言語のプログラムを直接機械語にするわけではありません。実はコンパイラは、内部でいったんアセンブルソースプログラムに直してから、機械語に変換しているのです。



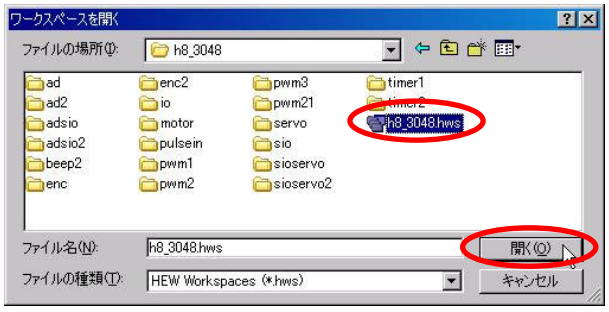
実行委員会開発環境は、コンパイラがアセンブルソースファイルに変換したファイルを見ることができませんでしたが、ルネサス統合開発環境では見ることができます。

7.5.1 ツールチェインの設定

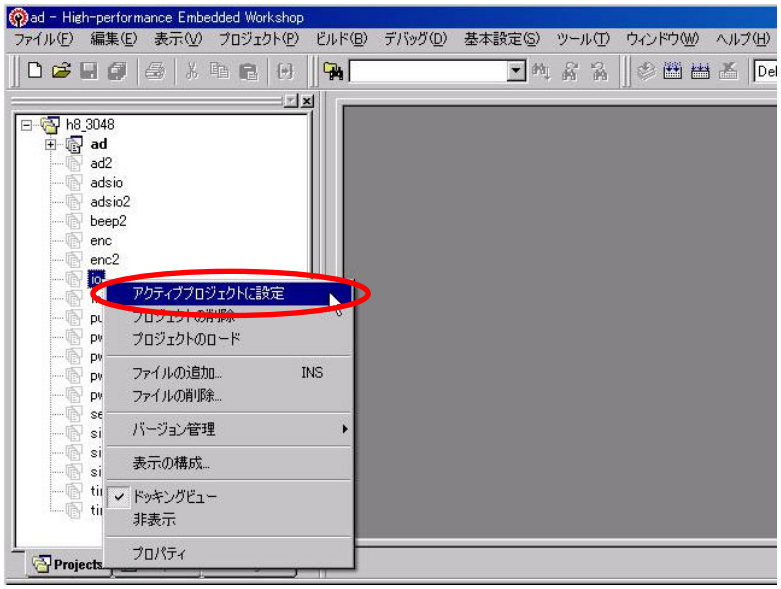
1. ルネサス統合開発環境を実行します。



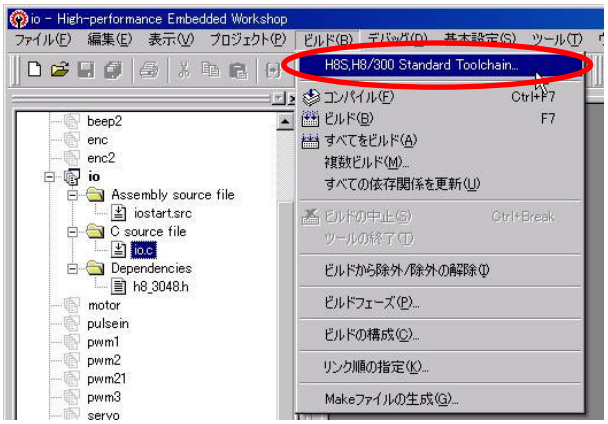
2.「別のプロジェクトワークスペースを参照する」を選択します。OK をクリックします。



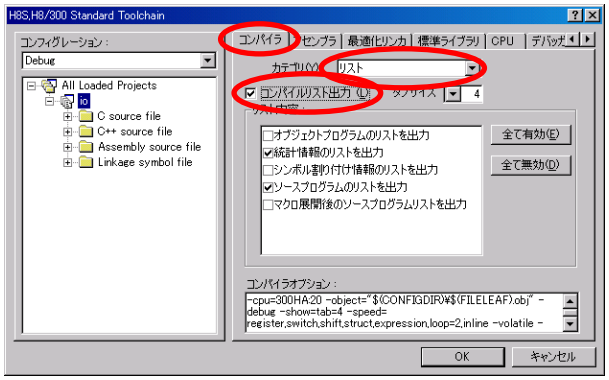
3. C ドライブ→Workspace→h8_3048 の「h8_3048.hws」を選択します。開く をクリックします。



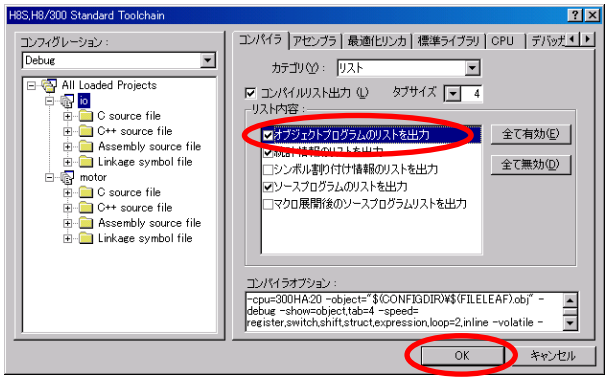
4. ワークスペース「h8_3048」を開きました。プロジェクト「io」をアクティブプロジェクトにします。「io」上で右クリック、「アクティブプロジェクトに設定」を選択します。すでにアクティブプロジェクトになっているなら設定の必要はありません。



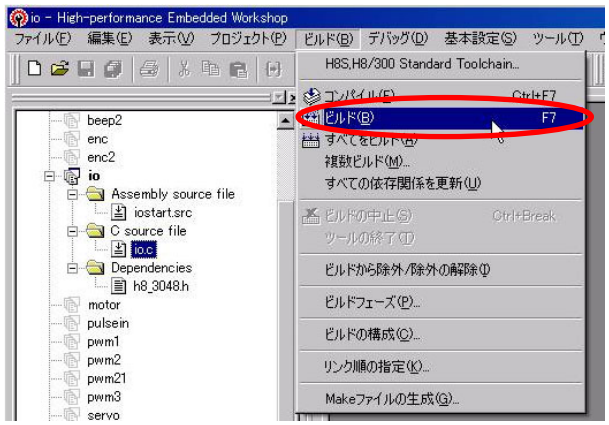
5. 「ビルド→H8S,H8/300 Standard Toolchain」を選択します。



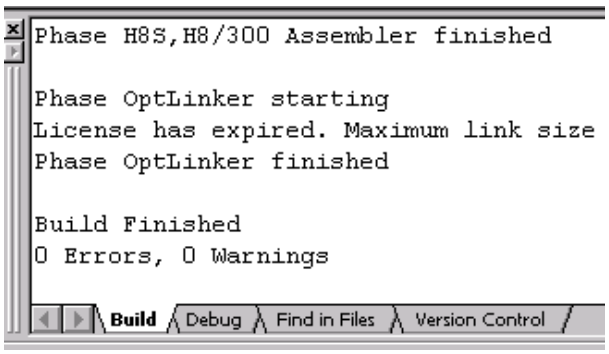
6. 「コンパイラ」、「カテゴリ:**リスト**」を選択します。「コンパイルリスト出力」にチェックを付けます。



7. 「オブジェクトプログラムのリストを出力」にチェックを付けます。OK をクリックして完了です。

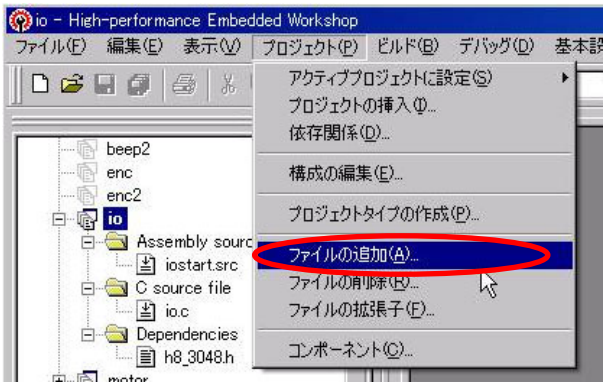


8. 「ビルド→ビルド」でビルドします。

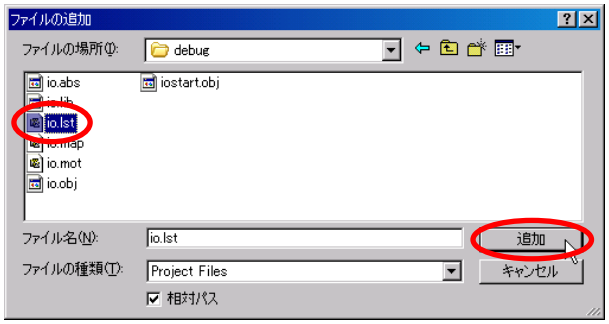


9. エラー0 で、ビルド完了です。

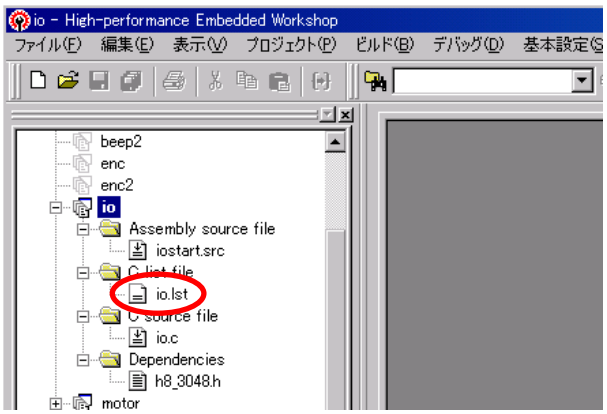
7.5.2 リストの表示



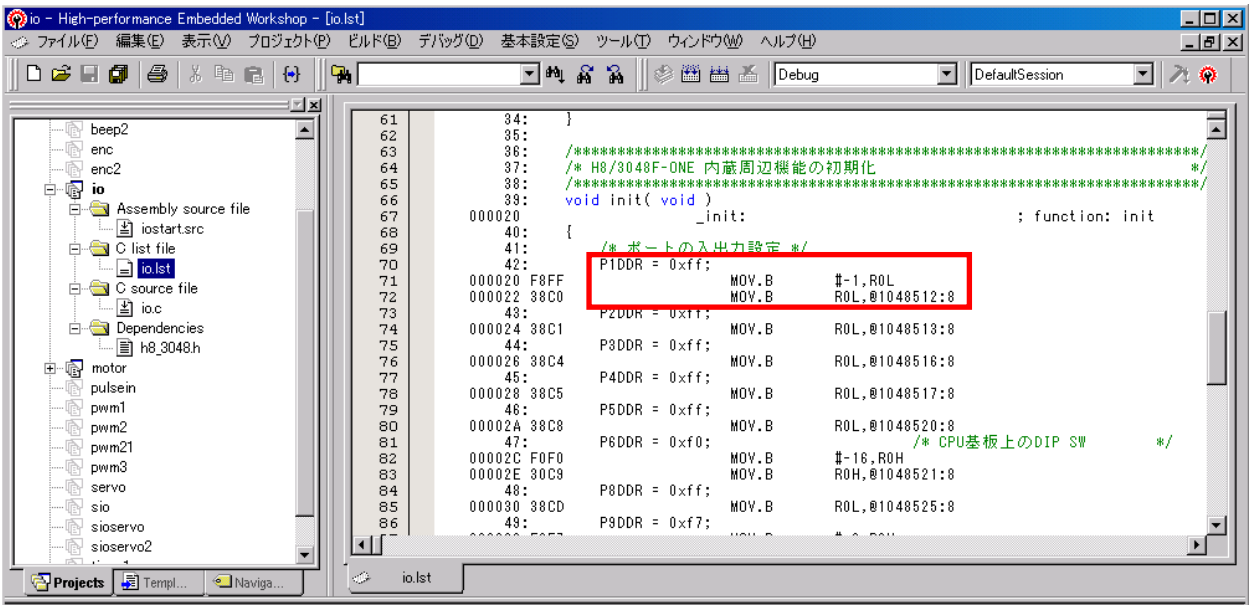
1. ファイルを見てみましょう。「プロジェクト→ファイルの追加」を選択します。



2. Cドライブ→Workspace→h8_3048→io→debug フォルダにある、「io.lst」を選択、追加をクリックします。



3. リストウィンドウに「io.lst」が追加されました。ダブルクリックします。



4. 表示されました。□部を詳しく見てみます。

```
    42:         P1DDR = 0xff;
000020 F8FF                     MOV.B       #-1,R0L
000022 38C0                     MOV.B       R0L,@1048512:8
```

となっています。これは、

```
    42:         P1DDR = 0xff;
```

という、C言語プログラムの1行が、

```
                MOV.B       #-1,R0L
                MOV.B       R0L,@1048512:8
```

という、アセンブリソースプログラム2行に変換されて、

```
            F8 FF
            38 C0
```

というオブジェクトファイル（機械語ファイル）に変換されたと言うことです。

要は、下記のように変換されたことになります。

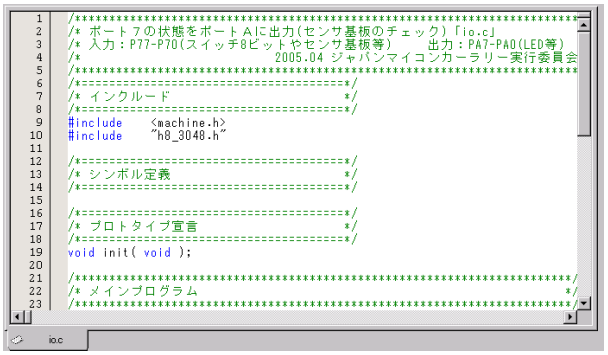
```
        P1DDR = 0xff;
              ↓
        F8 FF 38 C0
```

下記プログラムは、実行結果は同じですが、機械語サイズや実行スピードは違います。サイズが小さく実行速度が速いプログラムを目指しましょう！答えは書きませんので、上記の方法で変換されたアセンブリソースファイルを表示し、確かめてみてください。

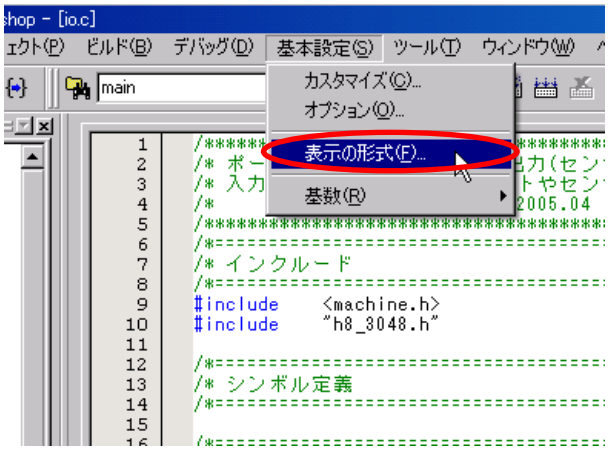
```
while( 1 ) {
    d = P7DR;
    d = d + d + d;
    PADR = d;
}
```

```
while( 1 ) {
    d = P7DR;
    d *= 3;
    PADR = d;
}
```

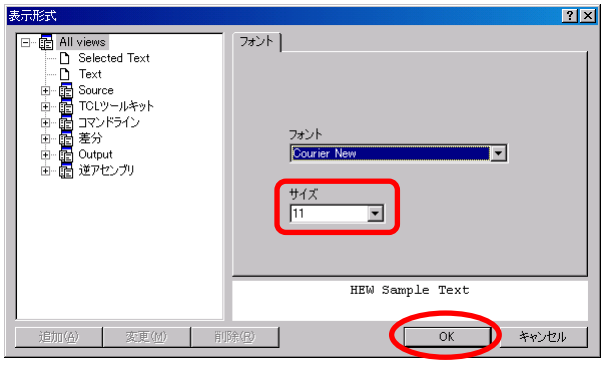
7.6 プログラム表示のフォントのサイズ変更



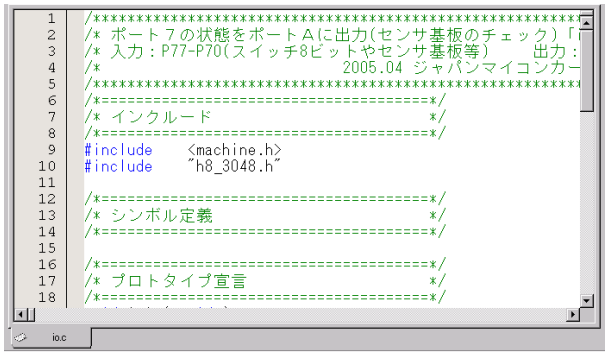
1. プログラムリストが小さくて見づらい場合、フォントのサイズを変更することができます。



2.「基本設定→表示の形式」を選択します。



3. サイズを変更します。OK をクリックして完了です。



4. 見やすくなりました。

8. FDT（ルネサスマイコン書き込みソフト）のインストール

8.1 概要

RY3048Fone ボード（H8/3048F-ONE）の場合、CpuWrite ソフトでプログラムの書き込みができました。他の CPU にプログラムを書き込みたい場合はどうしたらよいのでしょうか。

ルネサス テクノロジ製のほとんどのマイコンにプログラムを書き込むことのできる「**Flash Development Toolkit：フラッシュ開発ツール（以下 FDT と記述します）**」というソフトがあります。FDT を使って書き込みをしてみましょう。

8.2 CDから取得する

CD ドライブの「201 ルネサス統合開発環境」フォルダの「fdtv307r02.exe」を実行します（バージョンにより 307r02 部分が異なります）。

次は、「8.4 **インストールする**」へ進んで、FDTをインストールします。

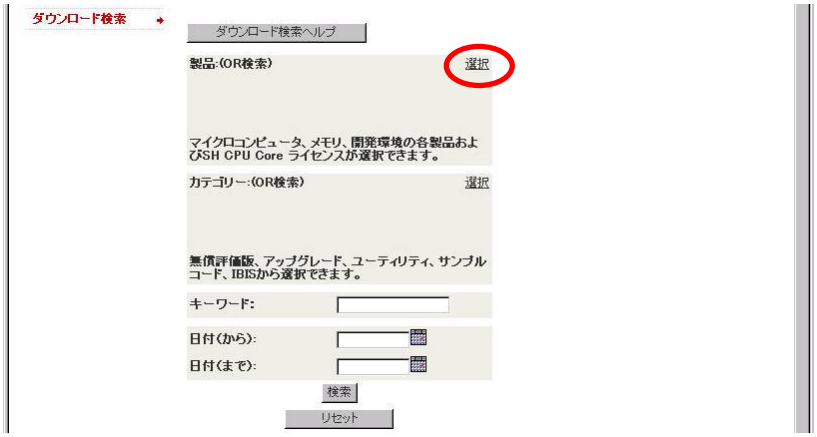
※ルネサス テクノロジのホームページには、常に最新の「FDT」があります。
100MB 程度のファイルがダウンロードできる環境なら、ホームページよりダウンロードすることをお勧めします。

8.3 ホームページから取得する

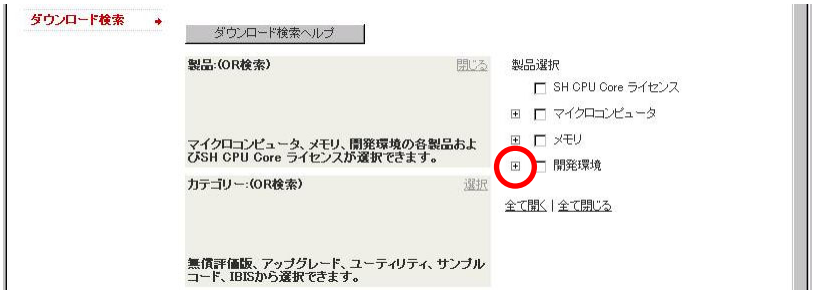
「MY　RENESAS」に登録していない方は、最初に登録してください。登録方法は「ルネサス統合開発環境　操作マニュアル　導入編」をご覧ください。

ルネサス テクノロジのサイト
http://japan.renesas.com/
にアクセスします。

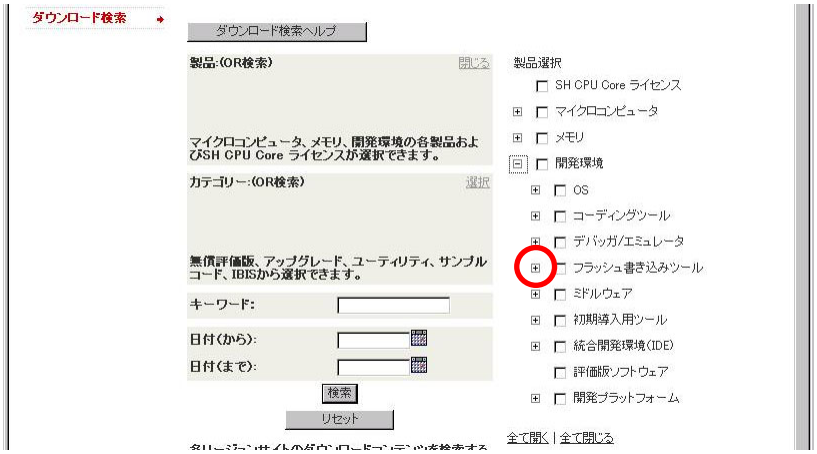
設計サポートの「ダウンロード検索」をクリックします。



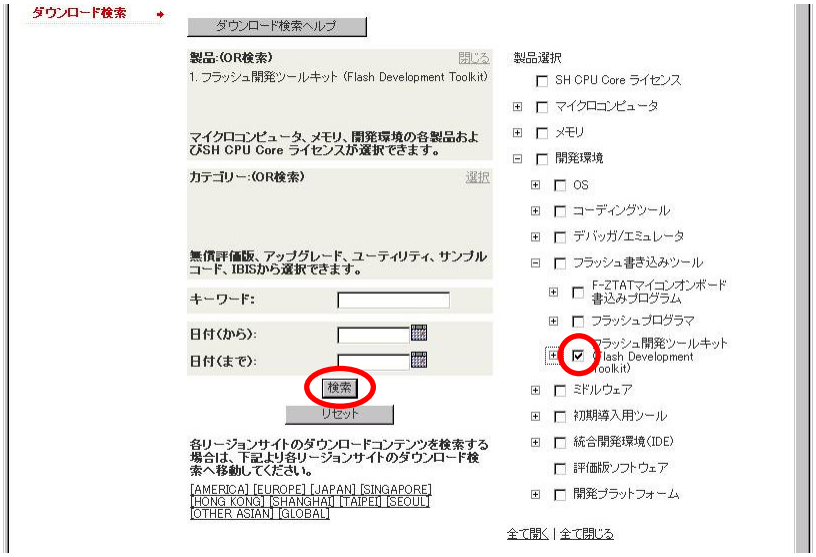
製品の「選択」をクリックします。



製品選択の「開発環境」の左にある＋をクリックします。



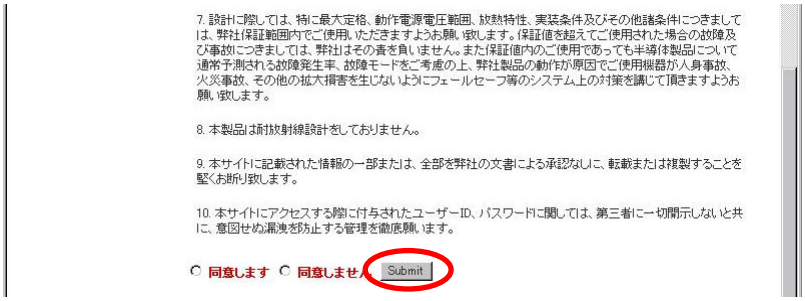
「フラッシュ書き込みツール」の左にある＋をクリックします。



「フラッシュ開発ツールキット (Flash Development Toolkit)」にチェックを付けます。検索をクリックします。

製品名「**【無償評価版】フラッシュ開発ツールキット V.3.07 Release 02**」をクリックします。
※「V.3.07 Release 02」部分は異なる場合があります。最新のバージョンを選択してください。

ログイン ID とパスワードを入力し、送信をクリックします。



ダウンロードに対する諸注意が表示されます。
同意する場合は「○同意します」のチェックを付けて、Submit(同意する)をクリックします。

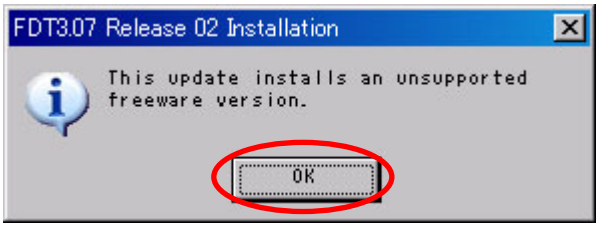
Download をクリックして、ファイルのダウンロードを開始します。ファイルサイズが約 35MB ありますので、ADSL や LAN など、ブロードバンド環境でダウンロードしてください。

ファイルがダウンロードされました。

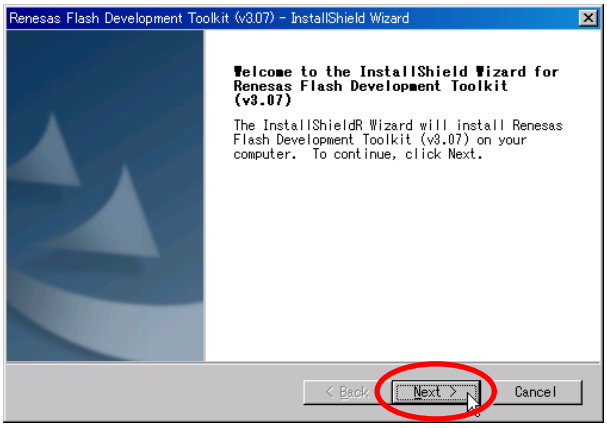
8.4　インストールする

CD もしくは、ダウンロードした「fdtv307r02.exe」(バージョンにより「307r02」の部分が異なります)を実行します。

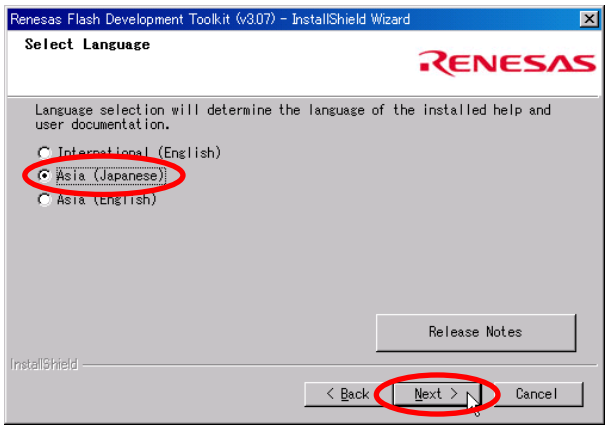
※インストールするには、ハードディスクの容量が 60MB 程度必要です。



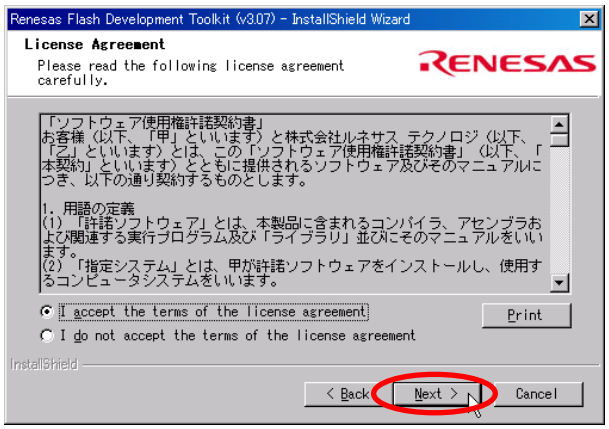
1. OK をクリックします。次の画面まで少し時間がかかります。



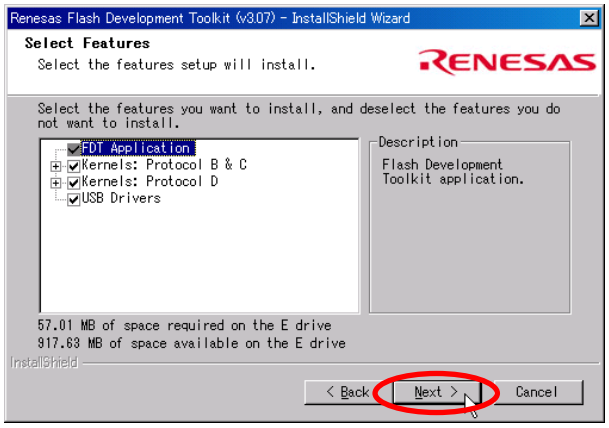
2. Next > をクリックします。



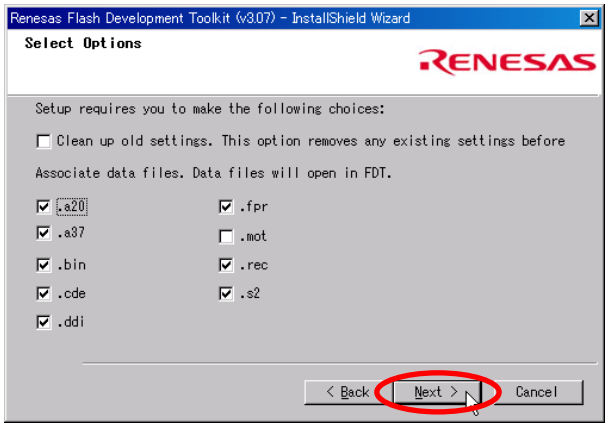
3. 言語の選択です。「Asia(Japanese)」を選択、Next > をクリックします。



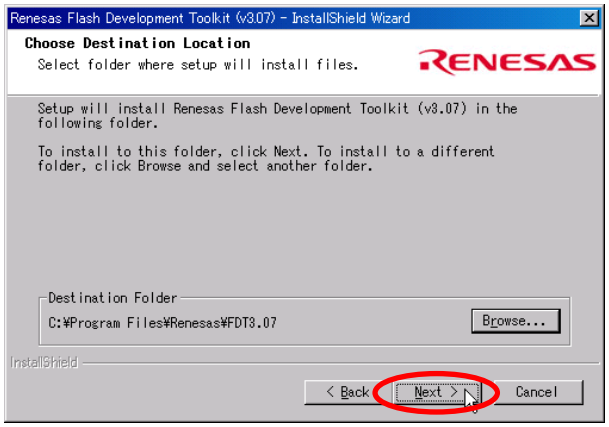
4. このソフトの使用条件を、承諾するかしないかの確認です。承諾するなら「I accept the…」を選択、Next > をクリックします。



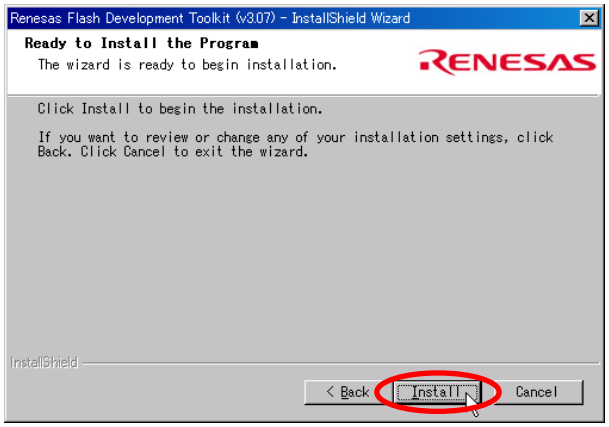
5. Next >をクリックします。



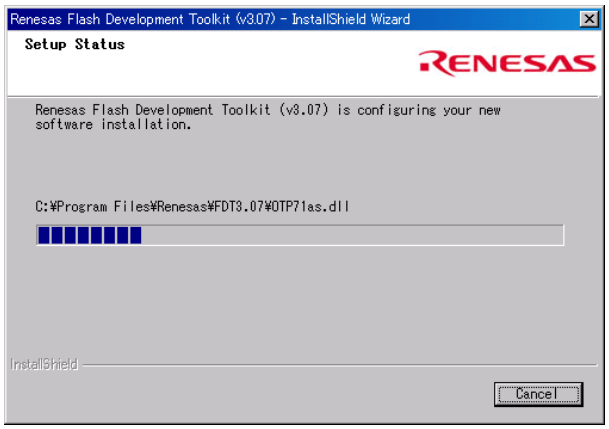
6. Next >をクリックします。



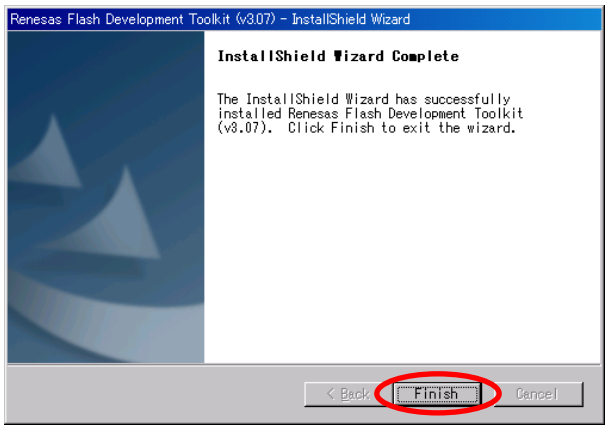
7. インストール先の選択です。変えたい場合はここで変えてください。Next >をクリックします。



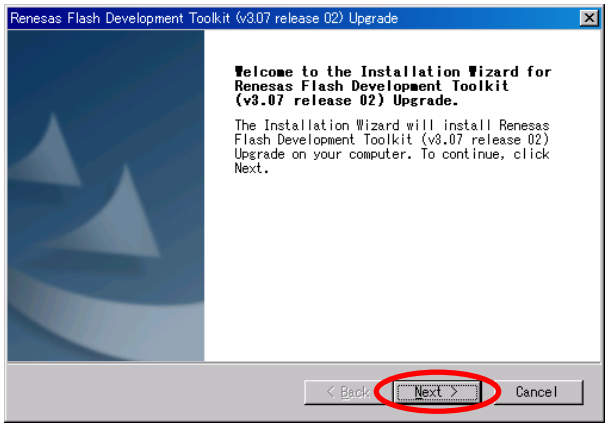
8. Installをクリックします。



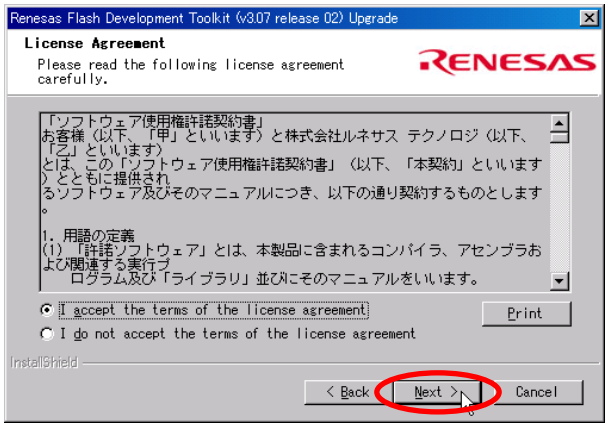
9. インストール中です。



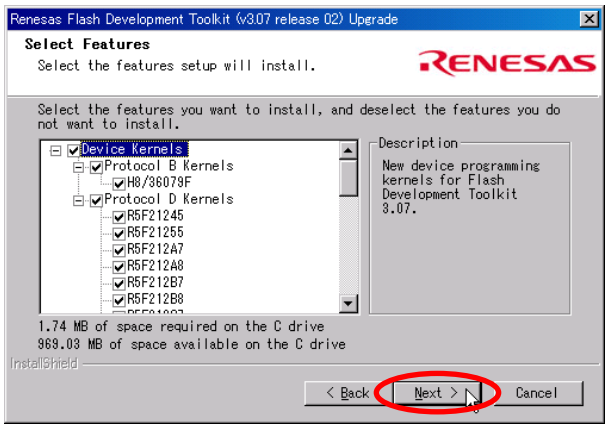
10. Finishをクリックします。



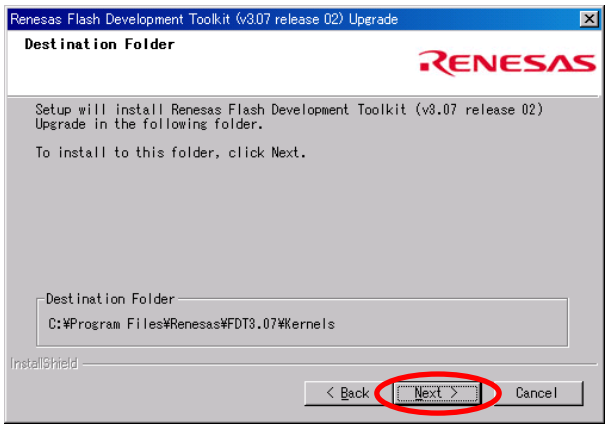
11. 自動的に 2 回目のインストールに入ります。
Next >をクリックします。



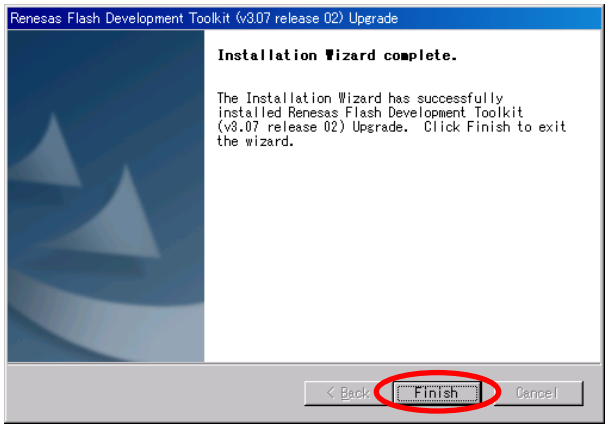
12. このソフトの使用条件を、承諾するかしないかの確認です。承諾するなら「I accept the…」を選択、
Next >をクリックします。



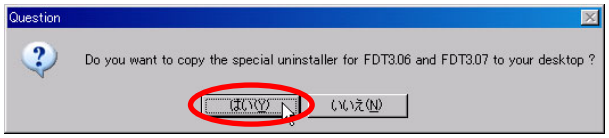
13. Next >をクリックします。



14. Next >をクリックします。



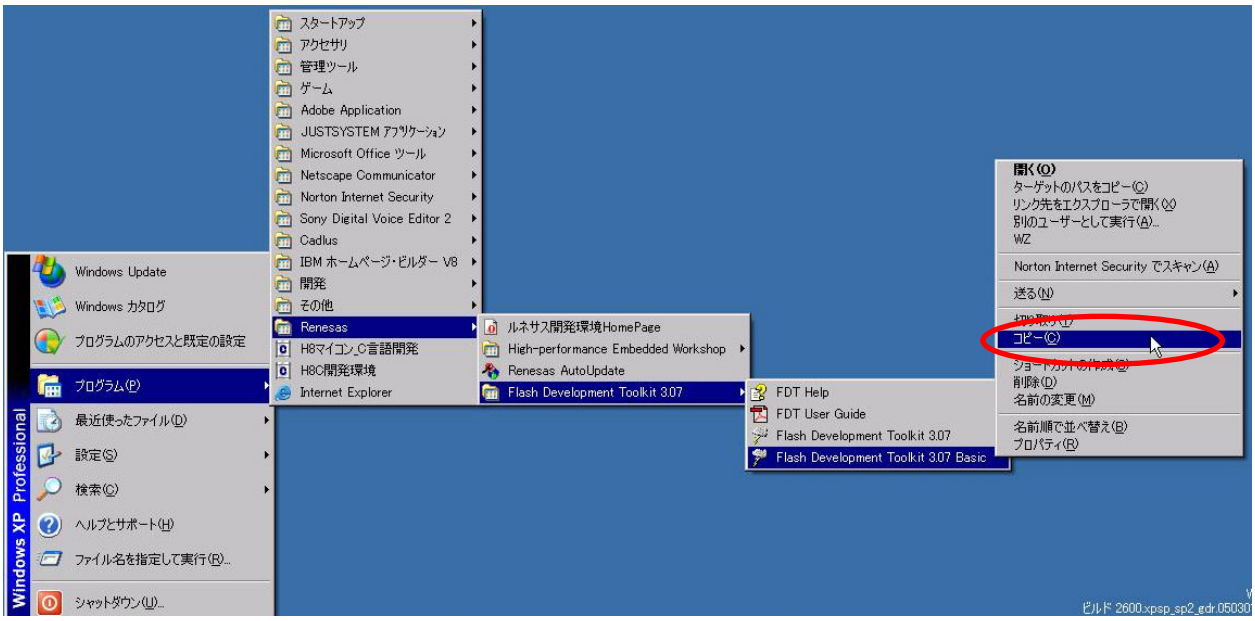
15. Finishをクリックして完了です。



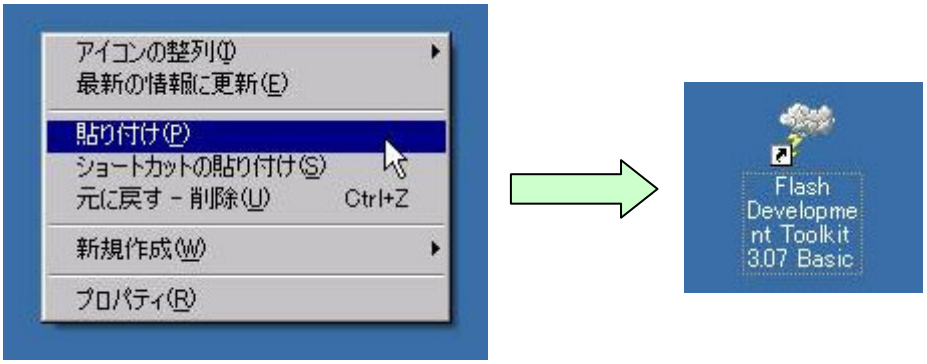
16. アンインストールプログラムをディスクトップに作りますか？と聞いてきます。はいをクリックして作っておきます。
※FDT をアンインストールするときは、このアンインストールプログラムを使ってください。

8.5　ショートカットを作成する

何度も FDT を起動させる場合は、ショートカットを作成しておきます。



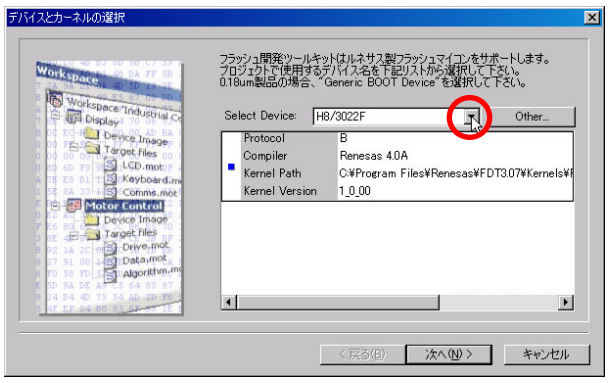
「スタート→すべてのプログラム（プログラム）→Renesas→Flash Development Toolkit 3.07→Flash Development Toolkit 3.07 Basic」を右クリックします。「コピー」をクリックします。



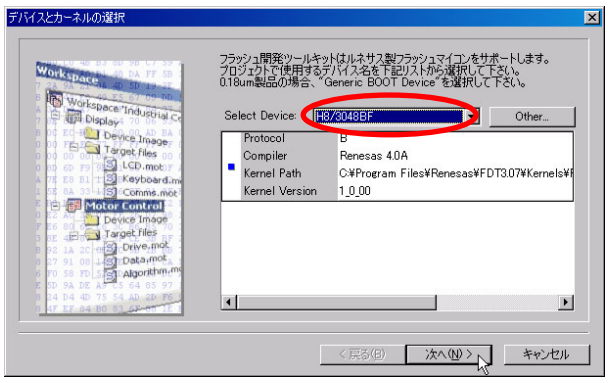
デスクトップ上で右クリックして、「貼り付け」でショートカット差作成されます。以降はこのショートカットから、FDT を実行することができます。

8.6　書き込み（最初に起動したとき）

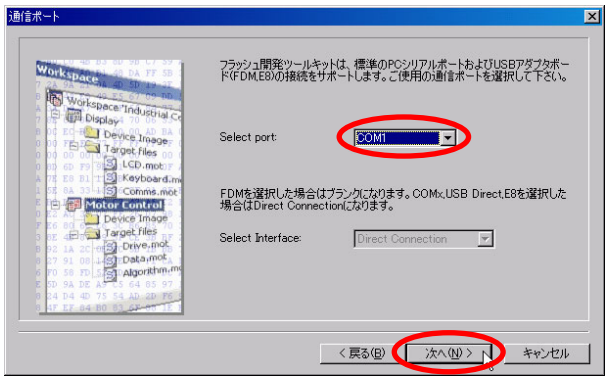
1. FDT を起動します。



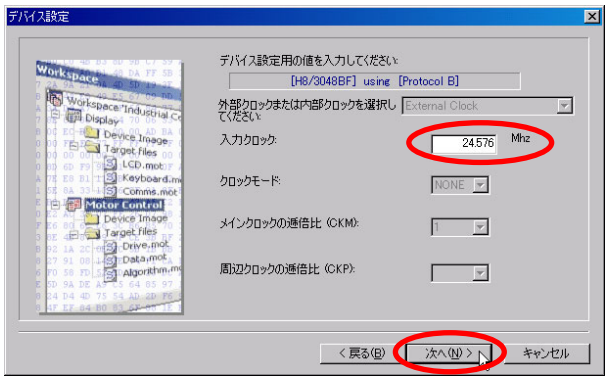
2. CPU を選択します。H8/3048F-ONE の場合を例に進めていきます。▼をクリックします。



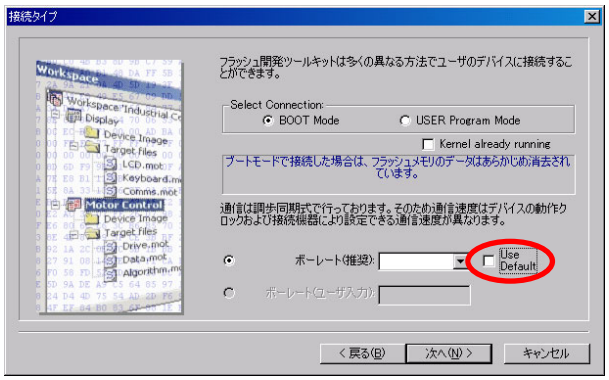
3.「H8/3048BF」を選択、次へをクリックします。
※H8/3048BF が H8/3048F-ONE の選択になります。



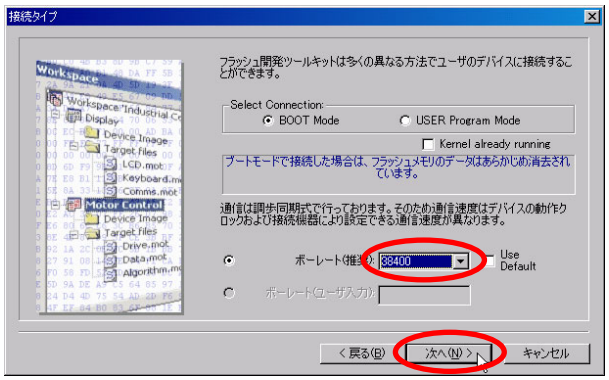
4. 通信ポートの番号を選択します。選択後、次へをクリックします。



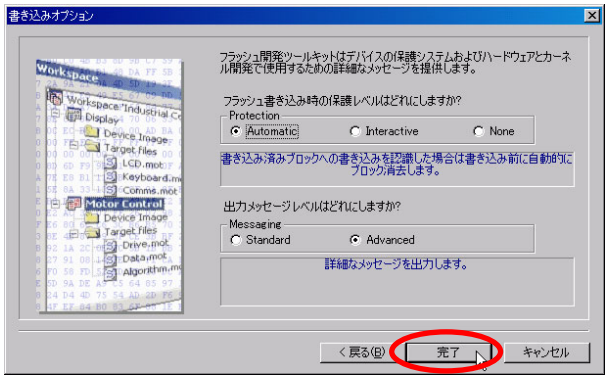
5. 入力クロックを入力します。入力クロックとは CPU ボードのクリスタル値のことです。
●RY3048Fone ボードの場合
「**24.576**」と入力します。次へをクリックします。
●RY3687 ボードの場合
「**14.7456**」と入力します。次へをクリックします。
●その他のボードの場合
クリスタルの値に合わせて入力し、次へをクリックします。



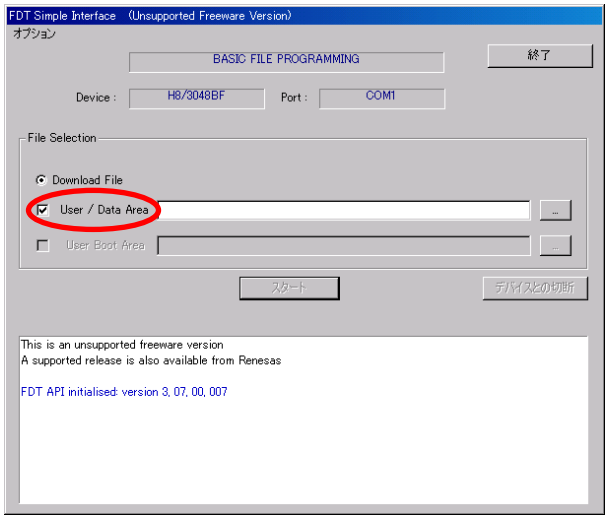
6. パソコンと CPU ボードの通信スピードを選択します。「Use Default」のチェックを外します。



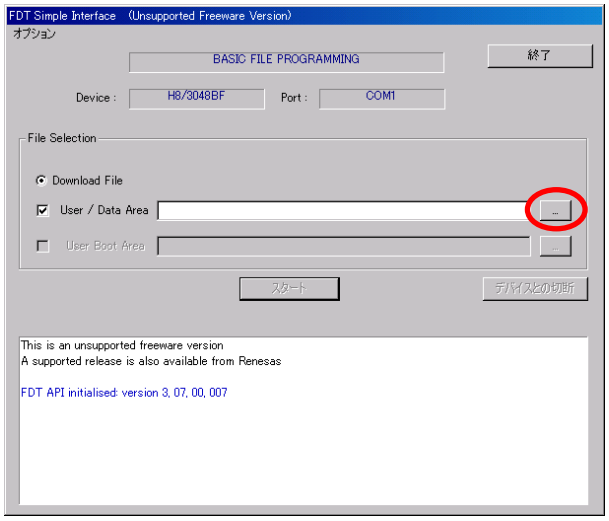
7. スピードを選択します。数値が大きい方がスピードが速いですが、通信エラーが発生しやすくなります。「**38400**」が無難な数値です。次へをクリックします。



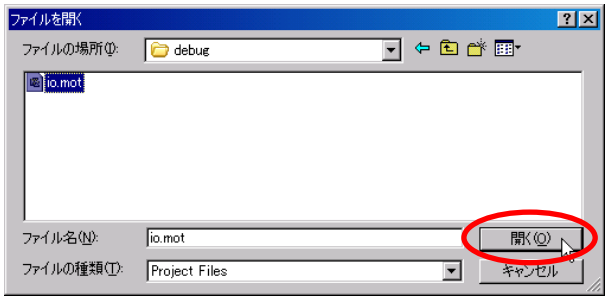
8. 完了をクリックして、完了です。



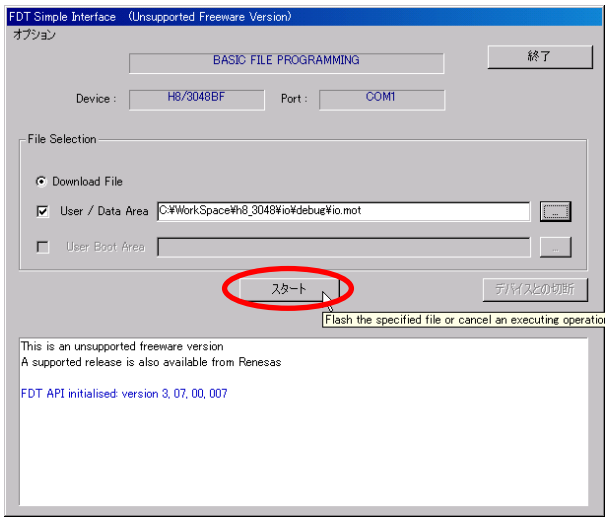
9.「User / Data Area」のチェックを付けます。



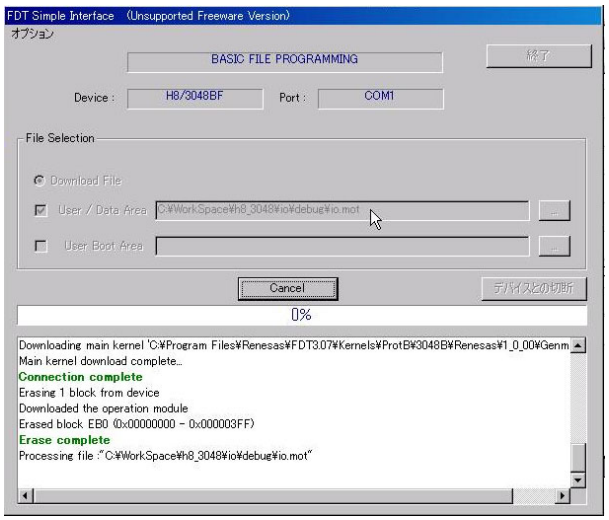
10. . . .をクリックします。



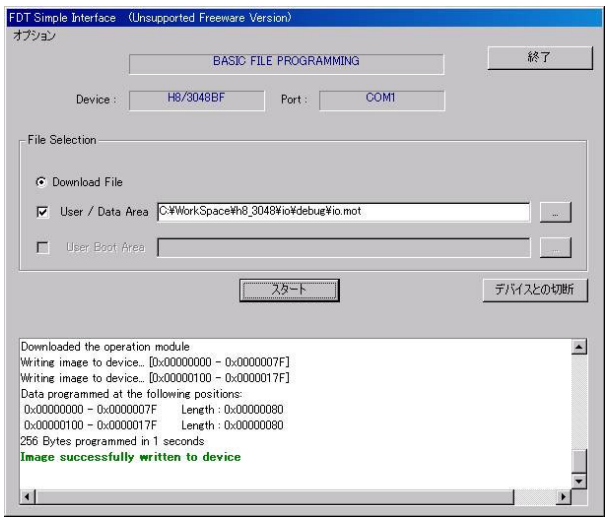
11. 書き込みたい MOT ファイルを選択して、開くをクリックします。



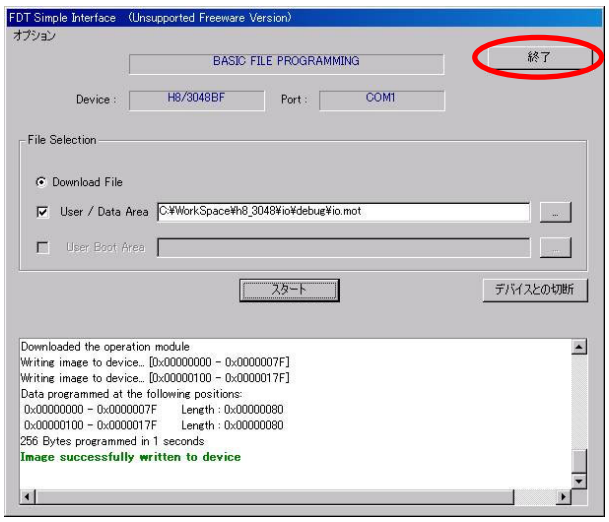
12. スタートをクリックして書き込みを開始します。



13. 書き込み中です。

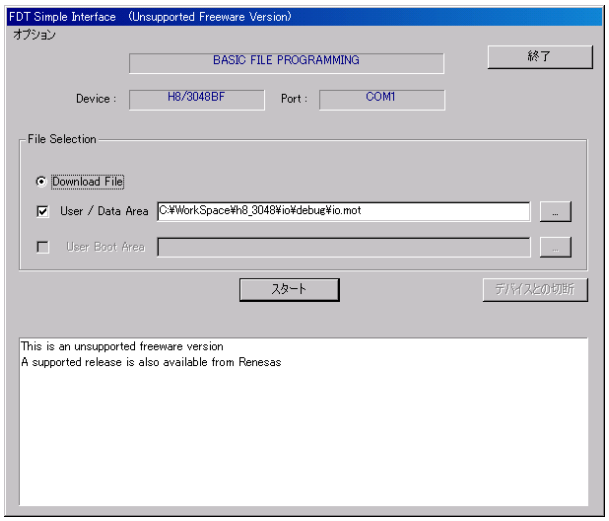


14. 正常に書き込みができれば
Image successfully written to device
と表示されて完了です。

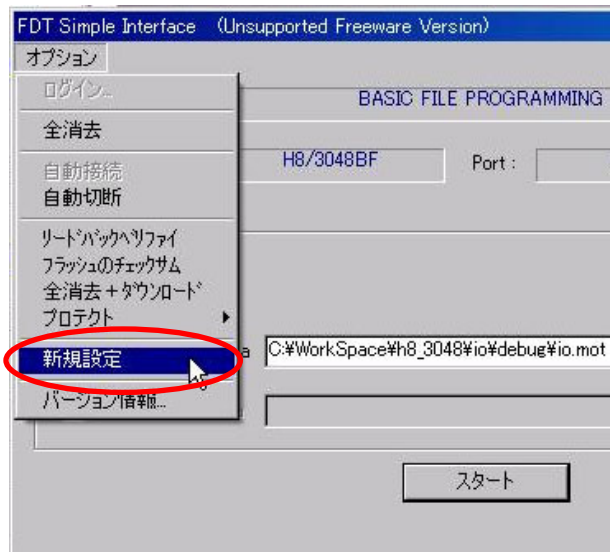


15. 書き込みが終われば、終了をクリックして FDT を終了します。

8.7 2 回目以降の画面



1. FDT を 2 回目以降立ち上げると、前の設定が引き継がれて立ち上がります。



2. CPU を変えたい場合、「オプション→新規設定」で新たに設定し直します。

9. 参考文献

・(株)ルネサス テクノロジ
H8/3048 シリーズ、H8/3048F-ZTAT™ (H8/3048F、H8/3048F-ONE)ハードウェアマニュアル 第 7 版
・(株)ルネサス テクノロジ
H8/3687 シリーズ ハードウェアマニュアル 第 3 版
・(株)ルネサス テクノロジ
High-performance Embedded Workshop V.4.00 ユーザーズマニュアル Rev.3.00
・(株)ルネサス テクノロジ 半導体トレーニングセンター C言語入門コーステキスト 第 1 版
・(株)オーム社 H8 マイコン完全マニュアル 藤澤幸穂著 第1版
・電波新聞社 マイコン入門講座 大須賀威彦著 第 1 版
・電波新聞社 C言語でH8マイコンを使いこなす 鹿取祐二著 第 1 版
・ソフトバンク(株) 新C言語入門シニア編 林晴比古著 初版
・共立出版(株) プログラマのための ANSI C 全書 L.Ammeraal 著
吉田敬一・竹内淑子・吉田恵美子訳 初版

マイコンカーラリーについての詳しい情報は、マイコンカーラリー公式ホームページをご覧ください。

http://www.mcr.gr.jp/

H8 マイコンについての詳しい情報は、(株)ルネサス テクノロジのホームページをご覧ください。

http://japan.renesas.com/

の「マイコン」→「H8 ファミリ」、または「マイコン」→「Tiny」でご覧頂けます

※リンクは、2009 年 2 月現在の情報です。