H8/3048F-ONE
シリアルモニタ
実習マニュアル
ルネサス統合開発環境版

第 1.01 版
2009.08.28
ジャパンマイコンカーラリー実行委員会

注 意 事 項 (rev.1.4)

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、文書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. シリアルモニタとは？

1.1 デバッグ

プログラムを作り動作させても、目的の動作をしてくれないことがあります。これは当然プログラムが間違っているためです(回路は正常だとします)。プログラムの誤りを「**バグ**」といい、プログラムが複雑になればなるほどバグを見つけるのは難しくなります。

バグを見つけるためプログラムの動作を調べたり、変数(レジスタ)を調べる装置のことを「**デバッガ**」といい、その行為を「**デバッグ**」といいます。

デバッガは多種多様な種類があり、値段と機能が違います。表 1. 1にデバッガについて示します。本マニュアルでは、「**シリアルモニタ**」を使いデバッグする方法について説明していきます。

表 1. 1　デバッガについて

装置名	フルスペック エミュレータ	コンパクト エミュレータ	オンチップデバッギングエミュレータ	**シリアルモニタ**
ルネサス製の製品	 E6000	 R0E436640CPE00	 E10T-USB	装置は使いません！
概要	デバッグする基板のマイコン部分に専用コネクタを取り付け、エミュレータが基板上のマイコンの変わりに動作してデバッグします。	フルスペックエミュレータより価格を抑え、また小型にしたにも関わらず、フルスペックエミュレータに迫る充実したデバッグ機能を備えたエミュレータです。	マイコンと通信機能で接続してデバッグします。簡単に取り付け可能ですが、機能が限られます。	マイコンと通信機能で接続してデバッグします。パソコン(ルネサス統合開発環境)とRS-232C で接続します。
できること	マイコンが動作している状態で、実行しているプログラムの確認、レジスタ、メモリの内容参照、実行時間の測定、メモリの内容が設定値になったらブレーク(動作停止)など、非常に高機能です。	フルスペックエミュレータでできる機能の一部ができません。	動作中の状態を見ることはできませんが、動作を一時停止してレジスタやメモリの書き換え、内容参照などできます。	同左
制限	特にありません。	特にありません。	デバッガがウォッチドッグタイマ(WDT)とSCI(通信)を使うため、ユーザ側で使用することができません。	オンチップデバッギングエミュレータの制限に加え、プログラムはRAMエリアに転送するため、約 3KB くらいのプログラムまでしか作成できません。
値段	数百万以上	数十万～数百万	数万程度	**0円！**

※エミュレータ…コンピュータや機械の模倣装置あるいは模倣ソフトウェアのことである。コンピュータ分野で使われることが多い用語だが、もともとは機械装置全般に使う言葉である。判りやすく言えば、機械を真似る機械である。(出典: フリー百科事典『ウィキペディア(Wikipedia)』)

1.2 シリアルモニタを使った構成

シリアルモニタは、パソコン(ルネサス統合開発環境)とマイコンをRS-232Cで接続し、このケーブルを通してプログラムやデータのやり取りを行い、デバッグすることができます(図 1. 1)。

ユーザプログラム(自分で作ったプログラム)は RAM 上に転送するため、書き込みソフト(CpuWrite や FDT など)でフラッシュ ROM に書き込む必要がありません(最初の1回だけはモニタプログラムをフラッシュ ROM に書き込まなければいけません)。フラッシュ ROM の寿命を延ばすという点でも、シリアルモニタは有効です。

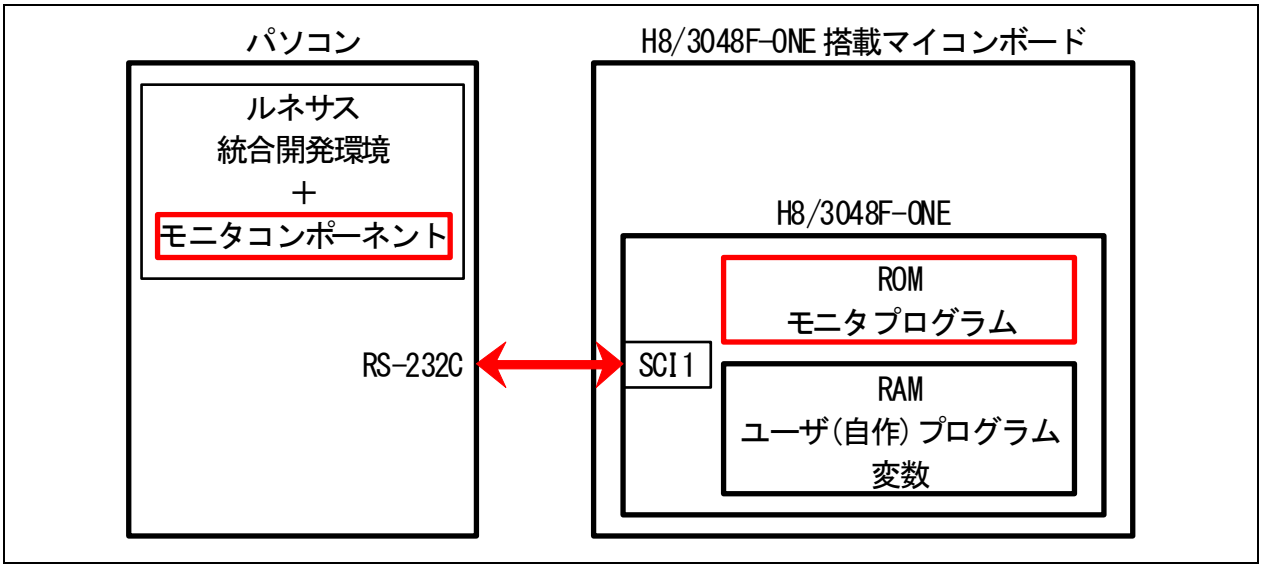


図 1. 1　構成

■ルネサス統合開発環境

ルネサス統合開発環境には、マイコンと通信を行う**モニタコンポーネント**という追加ソフトウェアをインストールします。インストールすることによって、ルネサス統合開発環境にレジスタやメモリの値を参照したり、プログラムを1行ずつ実行する機能などを追加することができます。一昔前は、テキストエディタなどでプログラムの作成、ビルド(MOT ファイルの作成)、デバッグをそれぞれのソフトで行う必要がありましたが、ルネサス統合開発環境だけでこれらのすべてを行うことができます。「統合開発環境」と言われるゆえんです。

■H8/3048F-ONE

H8/3048F-ONE には、モニタプログラムをあらかじめ書き込んでおきます。このモニタプログラムとルネサス統合開発環境が通信を行い、ユーザプログラムを RAM に転送し、実行します。プログラム停止中は、レジスタやメモリやプログラムの変数の値を参照、変更することができます。またプログラムを1行ずつ実行したり、指定した場所まで実行することができ、少しずつ動作を確認をすることができます。

1.3 シリアルモニタを使ったときの制限

モニタプログラムは、表 1. 2にあるH8/3048F-ONEの機能を使用しています。そのため、これらの機能をユーザープログラムで使用することができません。これらの機能以外は、自由にユーザプログラムで使用することができます。

表 1. 2 モニタプログラムが使用している H8/3048F-ONE の機能

モニタプログラムで使用している機能	内容
外部割り込み NMI	NMI 端子による外部割り込みが発生した場合、ユーザプログラムを強制停止するように設定されています。そのため、ユーザプログラムで NMI 割り込みを使うことはできません。
SCI1（シリアルコミュニケーションインタフェースのチャネル 1）	ルネサス統合開発環境のモニタコンポーネントとシリアルモニタは、SCI1 を使った通信により、命令のやり取りを行っています。そのため、ユーザプログラムで SCI1 は使えません。 **printf 関数や scanf 関数は SCI1 を使っていますので、これらの関数は使えません。**
ベクタ番号 4,5,6	ブレークポイントやステップイン、ステップオーバ、ステップアウトなどのシングルステップ動作を実現するために、ベクタ番号 4,5,6 のメモリ間接を利用しています。そのため、ユーザプログラムでベクタ番号 4,5,6 は使用できません。

1.4 シリアルモニタを使ったときのアドレス構成

1.4.1 H8/3048F-ONEのアドレス構成

ユーザプログラムは、ルネサス統合開発環境の設定で、RAMのアドレスになるように設定します。H8/3048F-ONEのアドレス構成は 表 1. 3のようになっています。

表 1. 3 H8/3048F-ONE のアドレス構成

内容	範囲	容量	内容
ROM	0x00000～0x1ffff	131,072 バイト（128KB）	モニタプログラムは、あらかじめ ROM に書き込んでおきます。
RAM	0xfef10～0xfff0f	4,096 バイト（4KB）	ユーザプログラムは、ここに配置します。ただし、次の用途でも RAM は使われます。 ・ユーザプログラムの変数領域 ・モニタプログラムの変数領域 ・スタックポインタ(SP)が使う領域 そのため、4KB すべて使えるわけではありません。プログラムエリアとして使えるのは、約 2.5K～3KB 程度です。
内蔵周辺機能	0xfff1c～0xfffff	228 バイト	内蔵周辺機能の領域です。例えば、PADR は、0xfffd3 番地です。

ROMの中でも、0x0000～0x00f3 番地はベクタアドレスといって、リセット後に実行するアドレスや、割り込み発生時に実行するアドレスが書かれている領域です。ユーザプログラムは、どのアドレスにあっても実行することができますのでアドレスを自由に変更することができますが、ベクタアドレスは変更することができません（図 1. 2）。ベクタアドレスを書き換えるために、書き込みソフトでフラッシュROMに書き込むのでは、シリアルモニタを使う意味がありません。

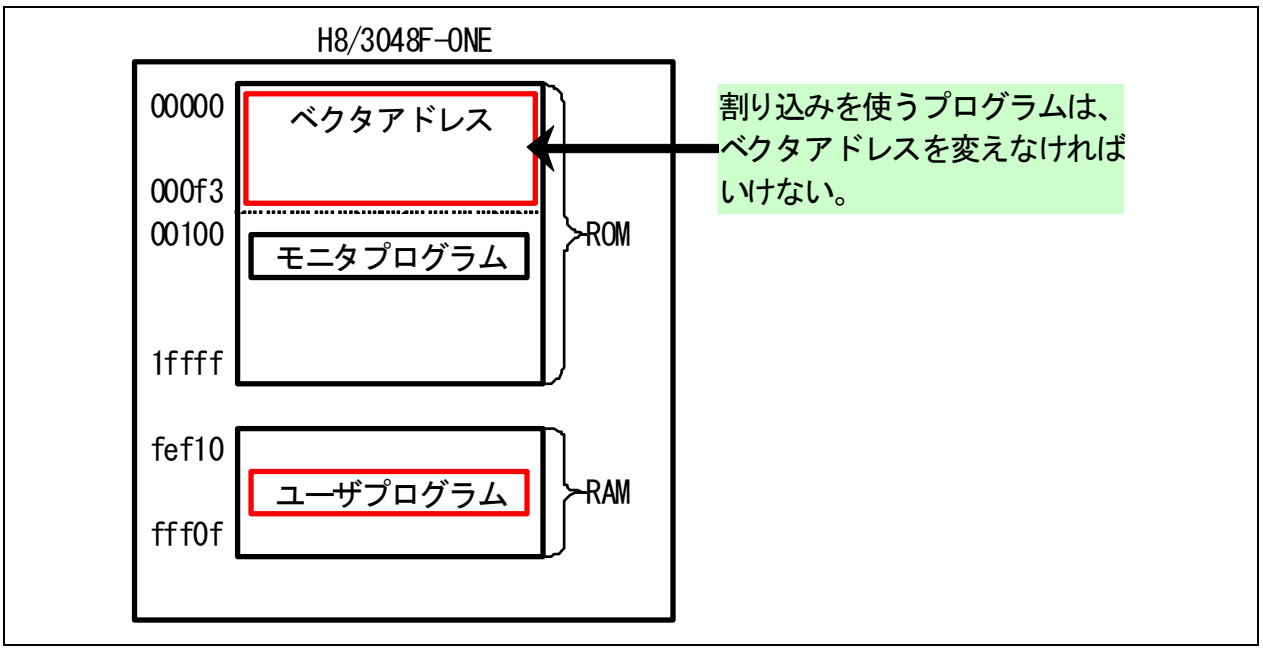


図 1. 2　ベクタアドレスとユーザプログラムのアドレス

1.4.2　RAMによるフラッシュメモリのエミュレーション機能を使う

H8/3048F-ONEには、「**RAMによるフラッシュメモリのエミュレーション**」という機能があります。具体的には、0xff000～0xff3ff番地のRAMをROM領域に重ねることができます。今回は、0x00000～0x003ff番地に重ねます。結果、ベクタアドレスもRAM上にあることになり、ベクタアドレス、プログラムをRS-232Cを通してマイコンのRAM領域におくことができ、フラッシュROMを書き換える必要がありません（図 1. 3）。

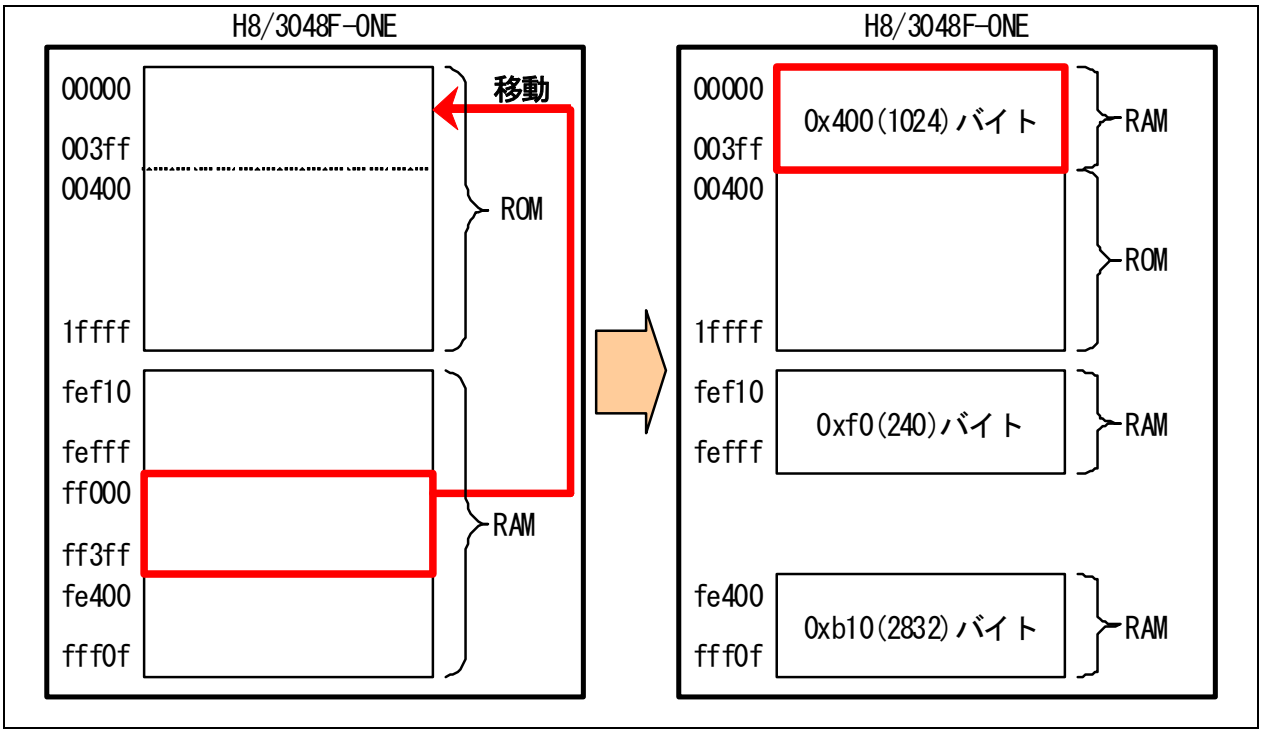


図 1. 3　RAM によるフラッシュメモリのエミュレーション機能を使ったときのアドレス構成

本当は、ベクタアドレスである 0x00000～0x000f3 番地だけ RAM にできれば、メモリを有効に活用できるのですができません。また、移動元も 0xff000～0xff3ff 番地ではなく RAM の先頭か最後にできれば、RAM が分かれずに済むのですがこちらもできません。

0x00000～0x003ff 番地の ROM は、無効になります。

1.4.3　H8/3048F-ONEの実際のアドレス構成

今回のシリアルモニタでは、モニタプログラムとユーザプログラムを 図 1. 4のように配置します。

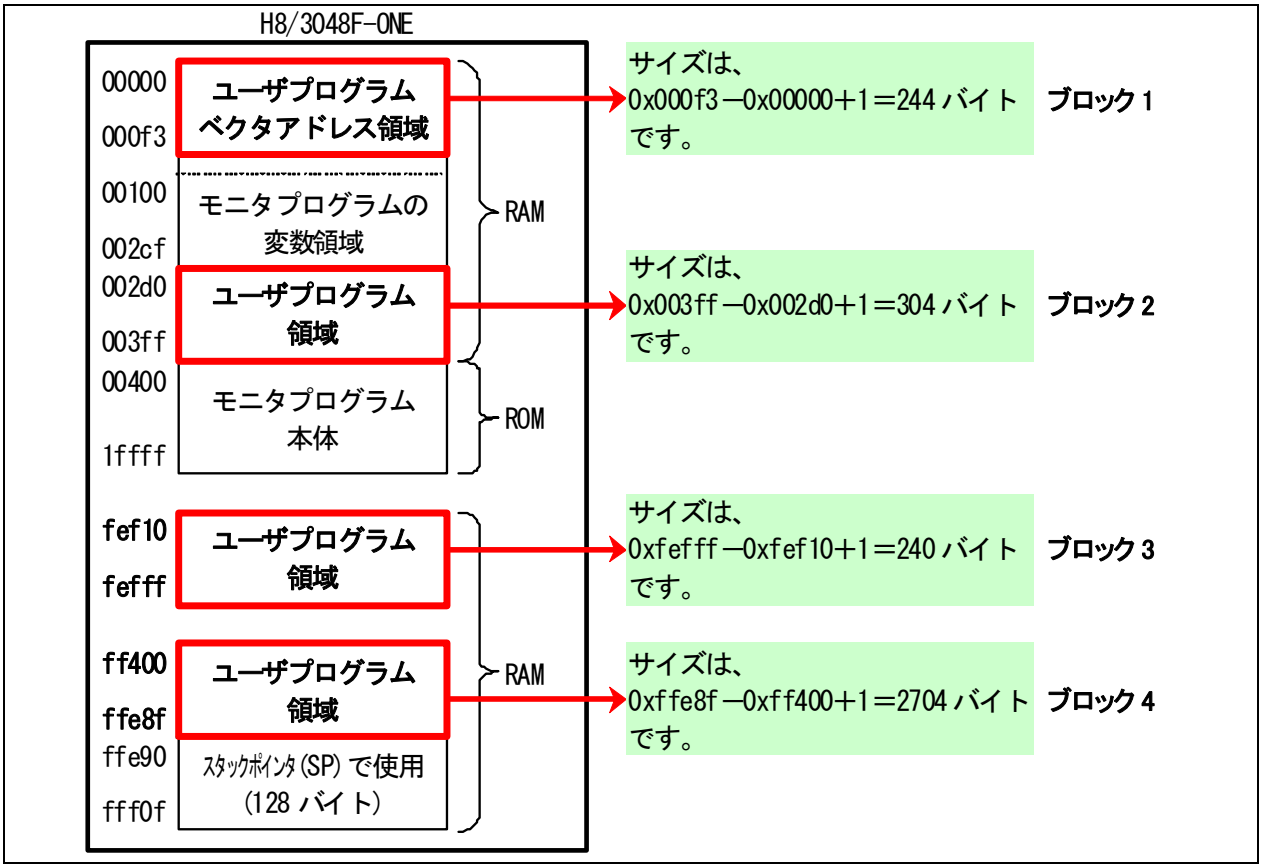


図 1. 4　ベクタアドレスとユーザプログラムのアドレス

このようにブロック 1～4 まで 4 つに分かれており、この領域にユーザプログラムを配置します。ただし、ブロック 1 (0x00000～0x000fc番地)は特別な領域で、必ずベクタアドレスを配置しなければいけません。他の 3 つは自由に使うことができます。本マニュアルでは、主に 表 1. 4のように割り当てます。

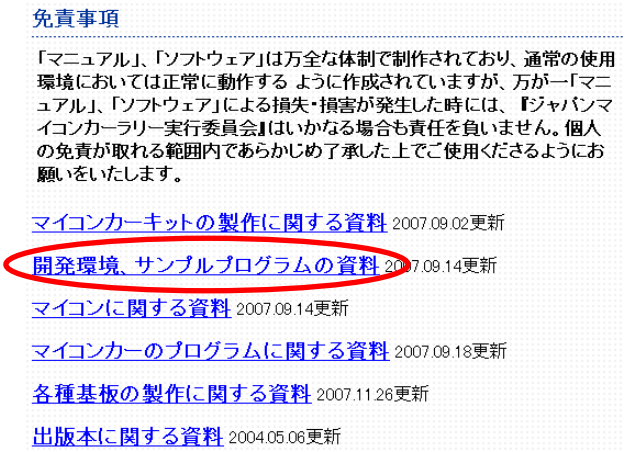
表 1. 4 ユーザプログラムで使える RAM アドレスと主な使い方

ブロック	ユーザプログラムで使える RAM アドレス	容量	今回の主な使い方
1	0x00000～0x000f3 番地	244 バイト	ユーザプログラムのベクタアドレスをこのアドレスに設定します。これは変えることはできません。
2	0x002d0～0x003ff 番地	304 バイト	基本的には使いません。
3	0xfef10～0xfefff 番地	240 バイト	主に変数領域として使います。
4	0xff400～0xffe8f 番地	2704 バイト	主にプログラム領域として使います。

ベクタアドレス以外は、自由に使うことができます。今回は、ブロック 4(0xff400～0xffe8f 番地)をプログラム領域として使いますが、変数領域として使っても構いません。プログラムサイズに応じて臨機応変に変更してください。

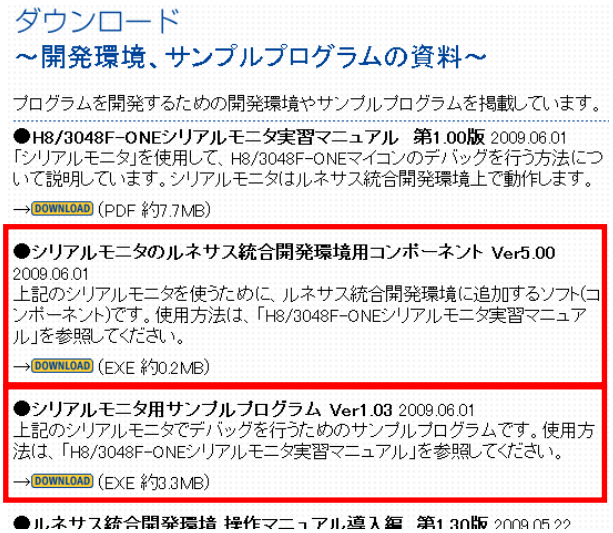
2. インストール

2.1 ファイルのダウンロード



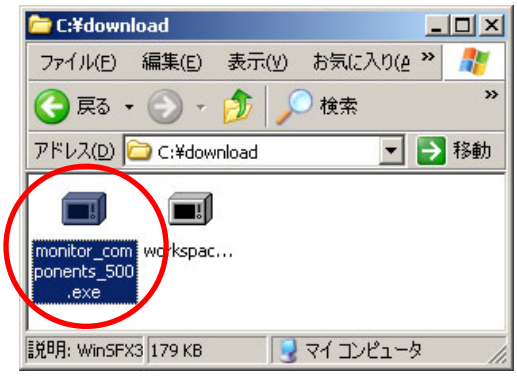
1.マイコンカーラリーサイト
「http://www.mcr.gr.jp/」の技術情報→ダウンロード内のページへ行きます。

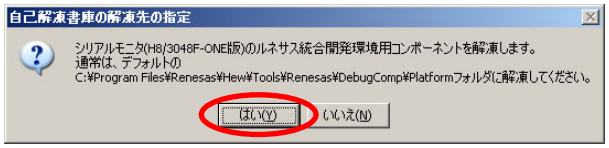
2.「開発環境、サンプルプログラムの資料」をダウンロードします。



3.「**シリアルモニタのルネサス統合開発環境用コンポーネント**」と「**シリアルモニタ用サンプルプログラム**」をダウンロードします。

2.2 ルネサス統合開発環境の設定

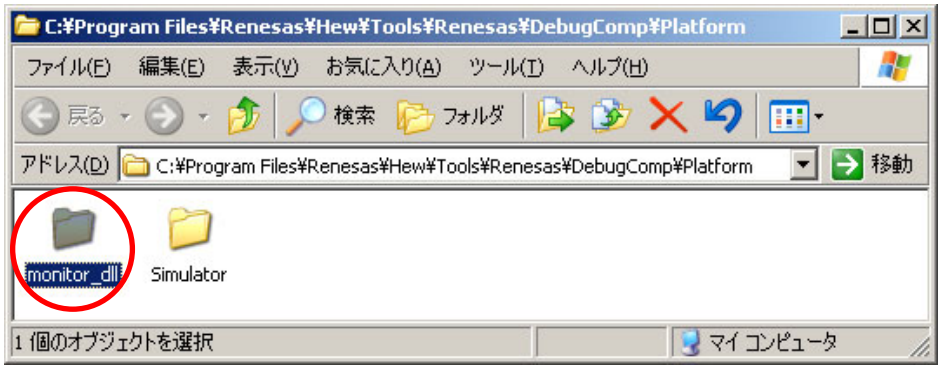




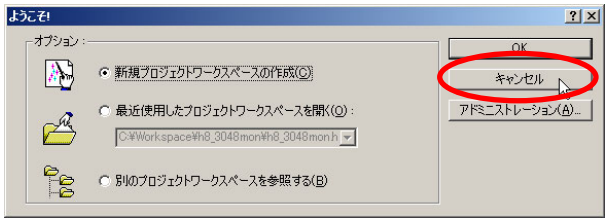
1.ダウンロードした、「monitor_components_500.exe」を実行します。数字の 500 は、バージョンにより異なります。

2.はいをクリックします。

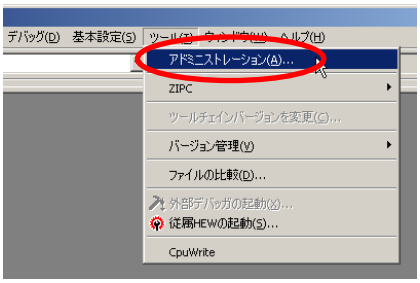
3.解凍先フォルダを指定します。ルネサス統合開発環境のインストール位置を変更していなければ、デフォルトの
「C:¥Program Files¥Renesas¥Hew¥Tools¥Renesas¥DebugComp¥Platform」
になります。インストールフォルダを変更した場合は、波線の位置をインストールしたフォルダに変更してください。OK をクリックします。



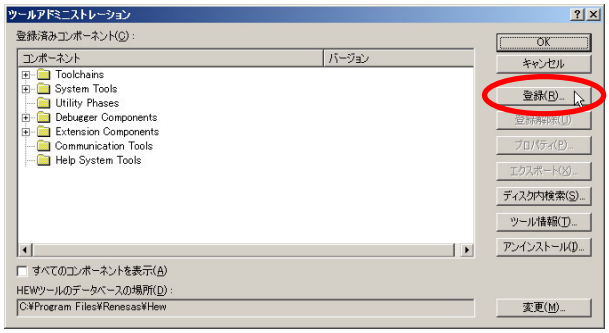
4.解凍先フォルダに「monitor_dll」というフォルダが作られていれば成功です。確認できればフォルダ画面は閉じます。



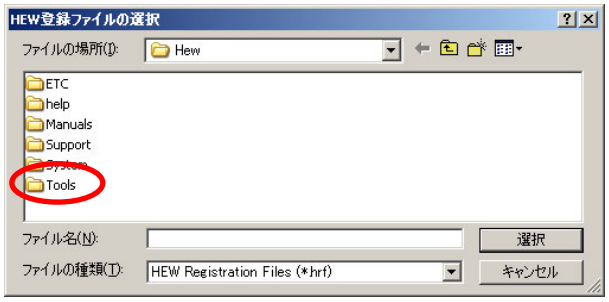
5.ルネサス統合開発環境を実行し、ようこそ画面をキャンセルします。



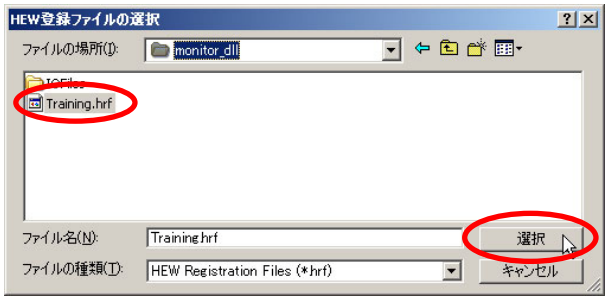
6.「ツール→アドミニストレーション」をクリックします。



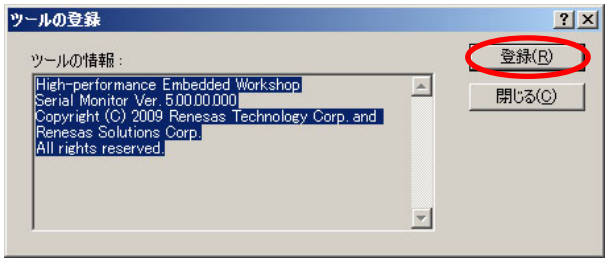
7.登録をクリックします。



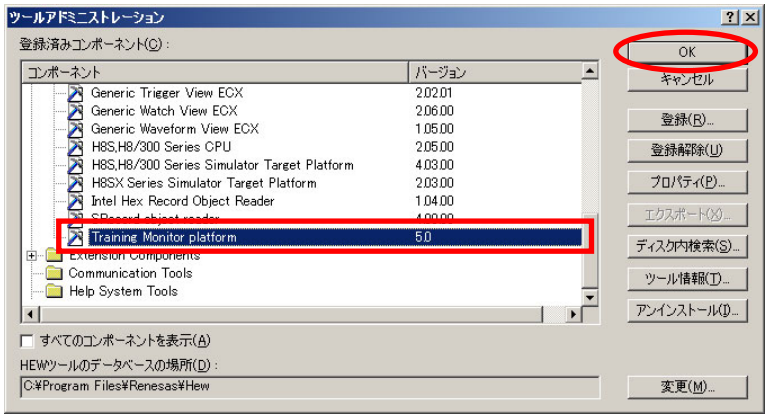
8.この画面なら、「Tools」をダブルクリックします。そうでない場合は、ルネサス統合開発環境をインストールしたフォルダを探して「Tools」をダブルクリックします。



9.続けて「Renesas」→「DebugComp」→「Platform」→「monitor_dll」と進んでいくと「Training.hrf」ファイルが表示されます。このファイルを選んで、選択をクリックします。

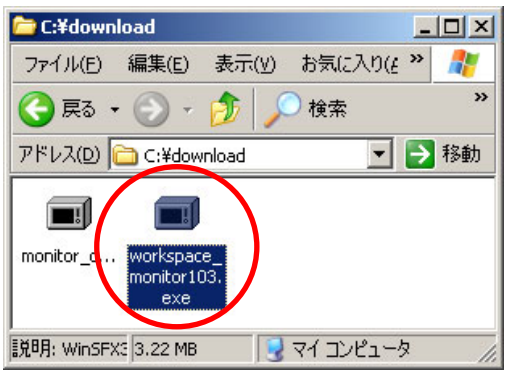


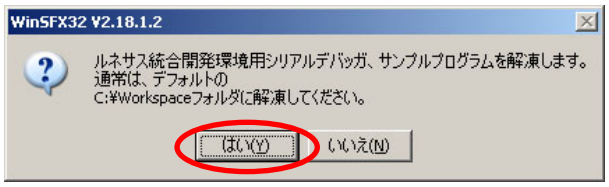
10.登録をクリックします。



11.「Debugger Components」の中に「**Training Monitor platform**」があればインストール完了です。OKをクリックします。ルネサス統合開発環境は終了します。

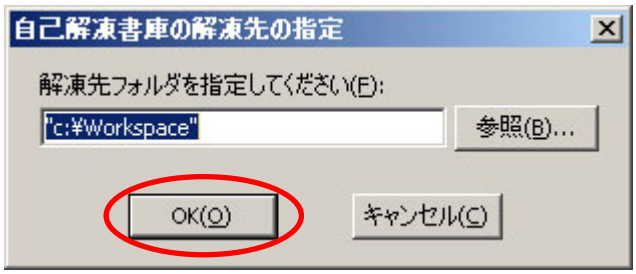
2.3 サンプルプログラムのインストール



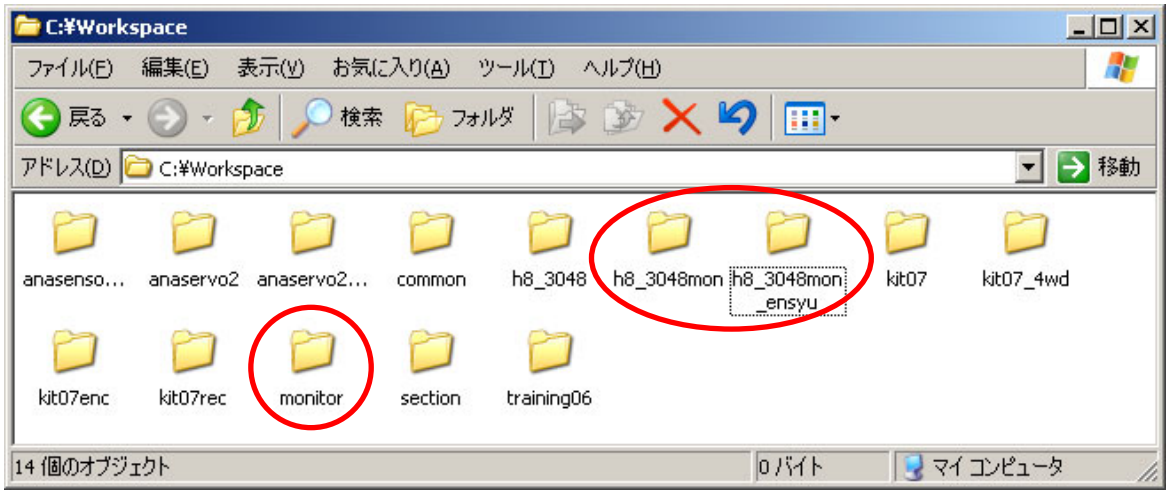


1.ダウンロードした、「workspace_monitor103.exe」を実行します。数字の 103 は、バージョンにより異なります。

2.はいをクリックします。

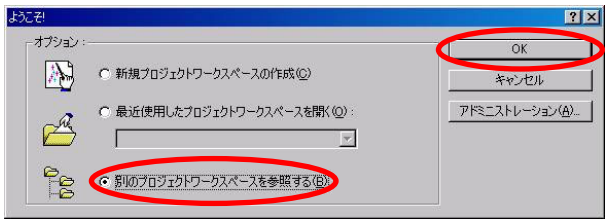


3.解凍先フォルダを指定します。標準のワークスペースフォルダの位置を変更していなければデフォルトの「c:¥Workspace」
になります。インストール位置を変更した場合は、インストールしたフォルダに変更してください。OK をクリックします。

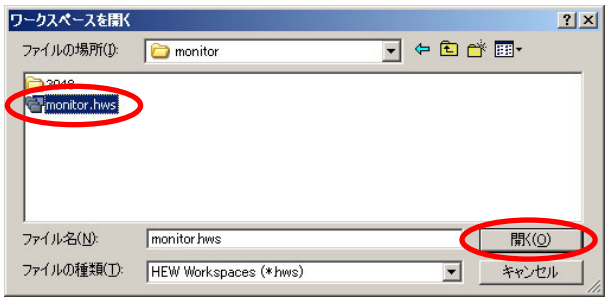


4.解凍先フォルダに「h8_3048mon」、「h8_3048mon_ensyu」、「monitor」という 3 つのフォルダが作られていれば成功です。また、「ルネサス統合開発環境 H8/3048 関連プログラム」をインストールしていない場合は、マイコンカーラリーホームページからダウンロードしてインストールをしておきます。

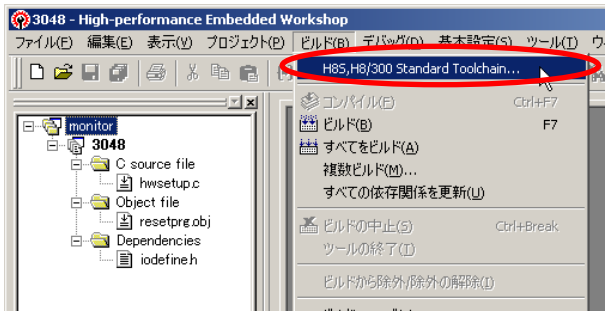
2.4 モニタプログラムの設定、書き込み



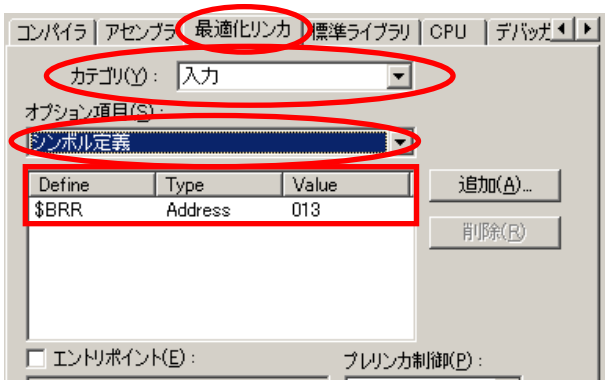
1.ルネサス統合開発環境を実行して、「別のプロジェクトワークスペースを参照する」を選択、OK をクリックします。



2.「Cドライブ→workspace→monitor」の「monitor.hws」を選択、開くをクリックします。



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



4.「最適化リンカ」を選択、「カテゴリ:**入力**」、「オプション項目:**シンボル定義**」にします。□欄の$BRR は 16 進数で「013」になっています。この値は、モニタプログラムを書き込むマイコンボードのクリスタルの値によって変わります。RY3048Fone ボード(24.576MHz)の場合は、0x13 です。詳しくは「$BRR の計算方法」を参照してください。

※$BRR の計算方法

BRR は次の式で求めます。

BRR ＝ φ／(64×2^{2n-1}×B)－1

B: ビットレート(bit/s)

φ: 動作周波数(Hz)

n: ボーレートジェネレータ入力クロック 0～3

B は、ルネサス統合開発環境とシリアルモニタが通信するスピードで本モニタは 38,400bps と決まってます。また、n も 0 と決まっています。分かっている値を入れると

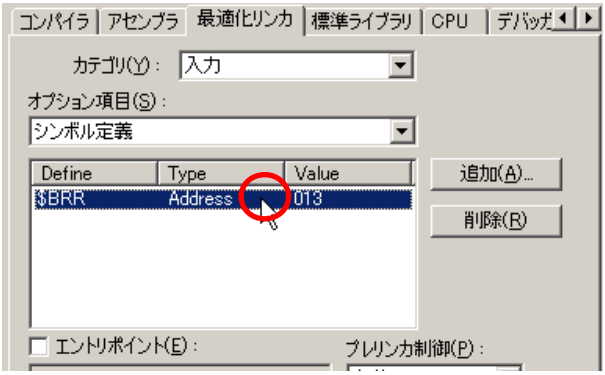
BRR ＝ φ／1,228,800－1

となります。

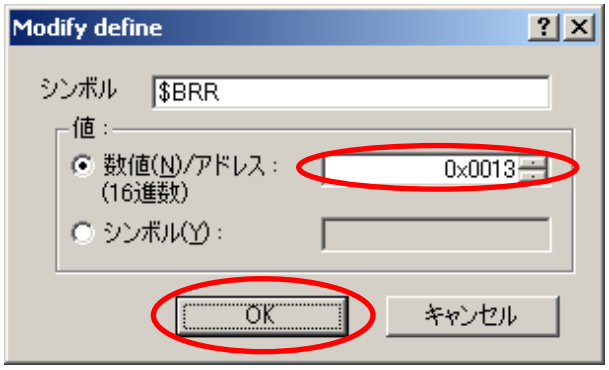
クリスタルが 24.576MHz の場合、

BRR ＝24.576×10^6／1,228,800－1＝19＝**0x13**

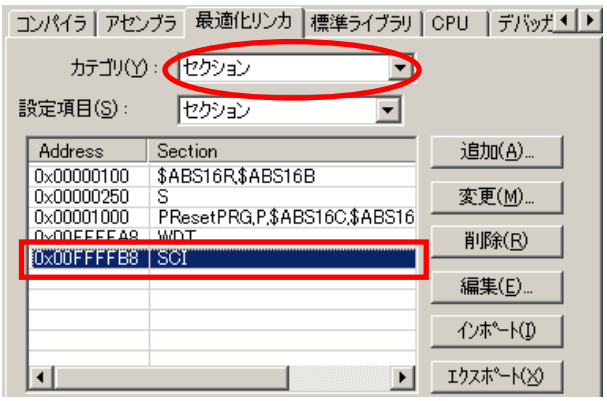
となります。計算結果に小数点が出た場合は四捨五入してください。



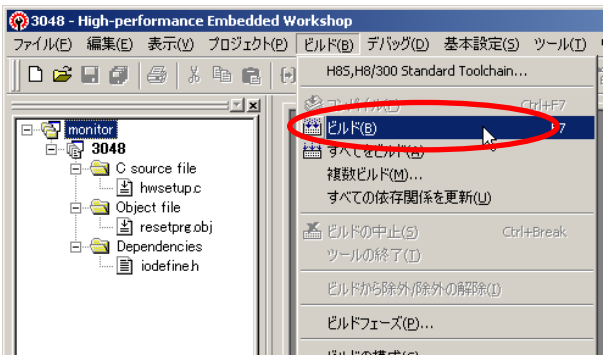
5.変更する場合は、$BRR 部分をダブルクリックします。



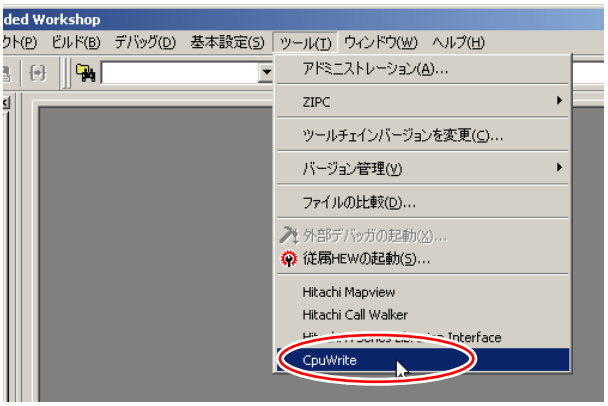
6.数値/アドレス欄に計算した値を入入力して、OK をクリックします。



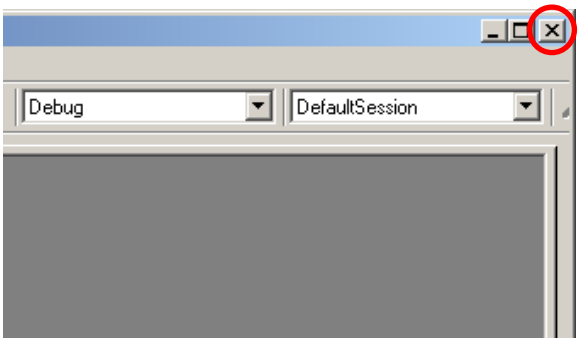
7.モニタは標準では SCI1 を使用しますが、SCI0 をモニタの通信として使用することもできます。SCI0 に変更する場合は、「カテゴリ:**セクション**」、「設定項目:**セクション**」にして、SCI の部分を「0x00FFFFB8→**0x00FFFFB0**」にします。



8.ツールチェインの設定は完了です。OK をクリックして、ツールチェイン画面を閉じて「ビルド→ビルド」で MOT ファイルを作成します。



9.「ツール→CpuWrite」で書き込んでください。



10.モニタプログラムの書き込みは完了です。ルネサス統合開発環境は閉じておきます。

3. 使い方

3.1 とりあえず接続する

ルネサス統合開発環境にコンポーネントをインストール、RY3048Fone ボードにモニタプログラムを書き込んでいる状態であれば、後は簡単です。細かい説明をする前にとりあえず接続して動作を確認してみましょう。

RY3048FoneボードとMCR2003A実習基板を 図 3. 1のように結線してください。詳しくは、H8/3048F-ONE実習マニュアルを参照してください。

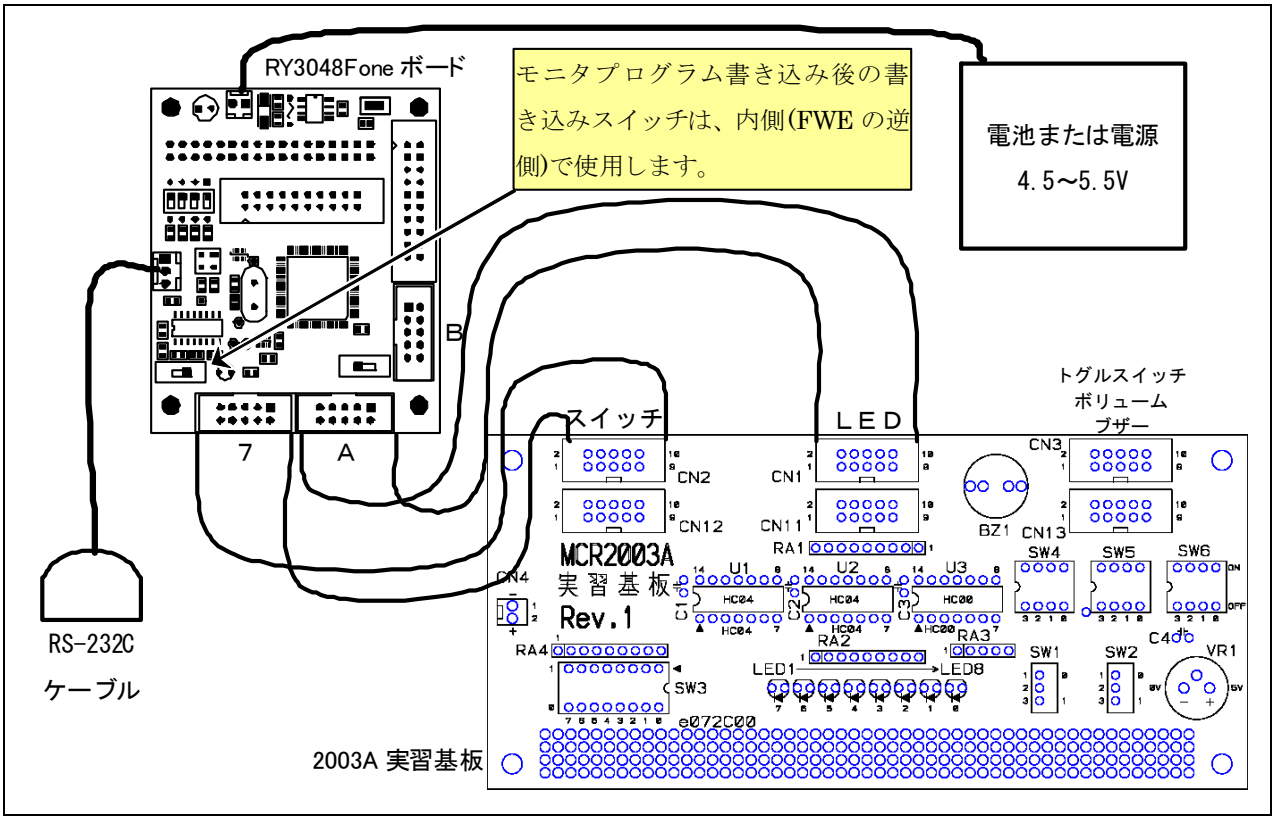
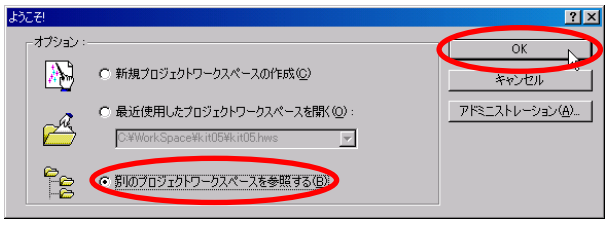
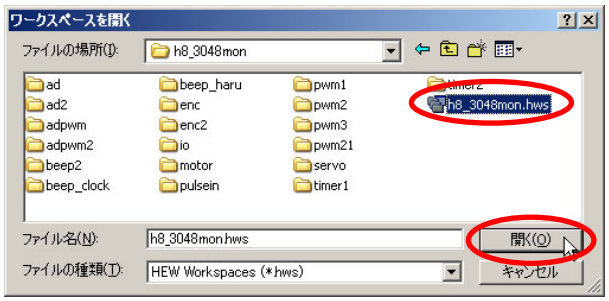


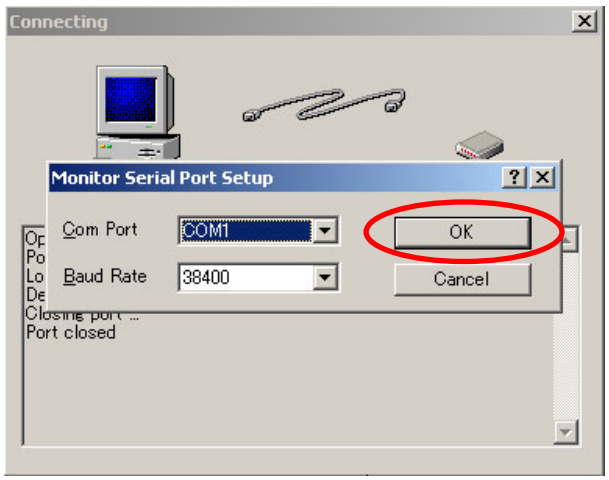
図 3. 1　結線図



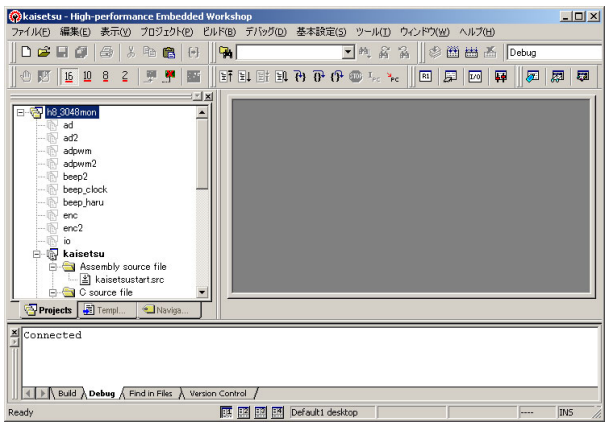
1.ルネサス統合開発環境を立ち上げます。「別のプロジェクトワークスペースを参照する」を選択して OK をクリックします。



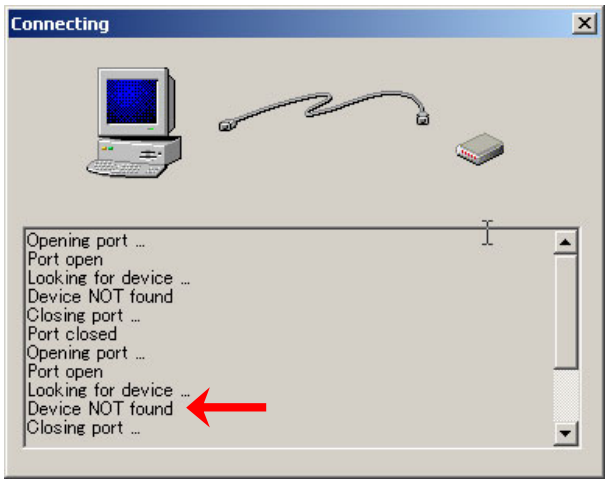
2.「C ドライブ → workspace → h8_3048mon 」の「h8_3048mon.hws」を選択します。開くをクリックします。



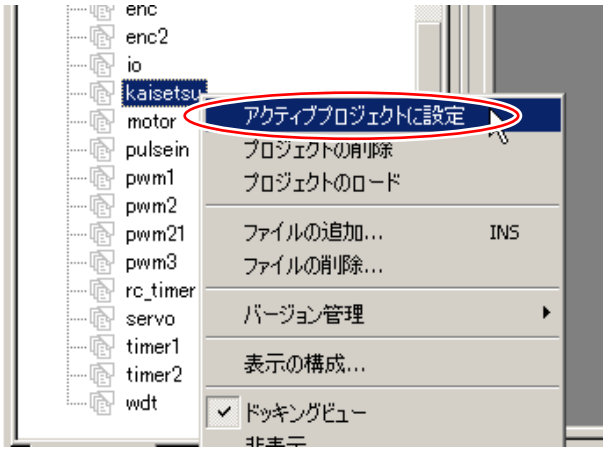
3.この画面が出てきたら、次の作業を行います。
・パソコンとマイコンボードを RS-232C ケーブルで接続する
・マイコンボードを通常状態(**書き込みでない状態**)**で電源を入れた状態にする**
その後、「Com Port」欄で COM 番号を設定、「Baud Rate」は 38400 のまま上記作業を実行して OK をクリックします。



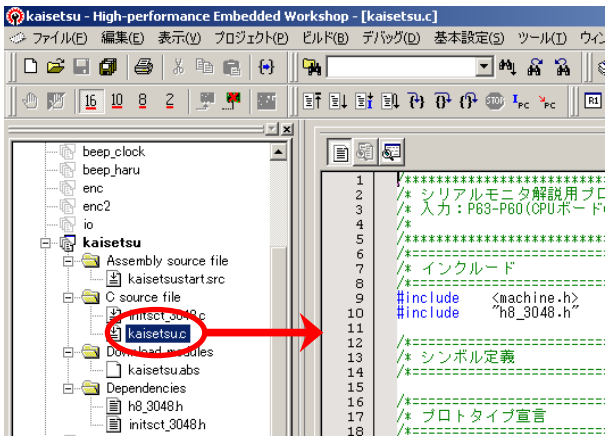
4.ルネサス統合開発環境とマイコンボード間で通信が確立すれば、ルネサス統合開発環境の画面が立ち上がります。



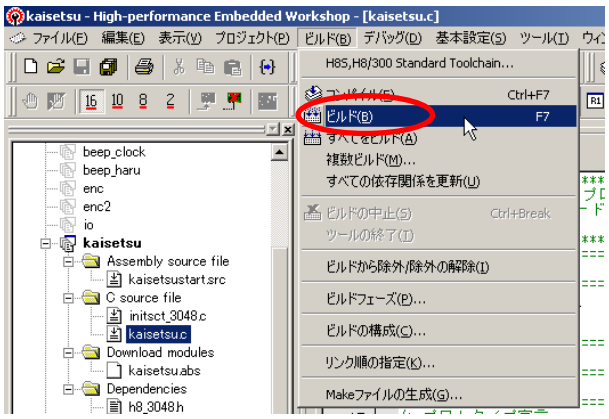
5.もし「Device NOT found」と出て、先ほどのポート番号設定画面に戻った場合は、マイコンボードと通信ができていません。通信ケーブルが接続されているか、マイコンボードの電源が入っているかなど調べて、再度 OK をクリックしてください。



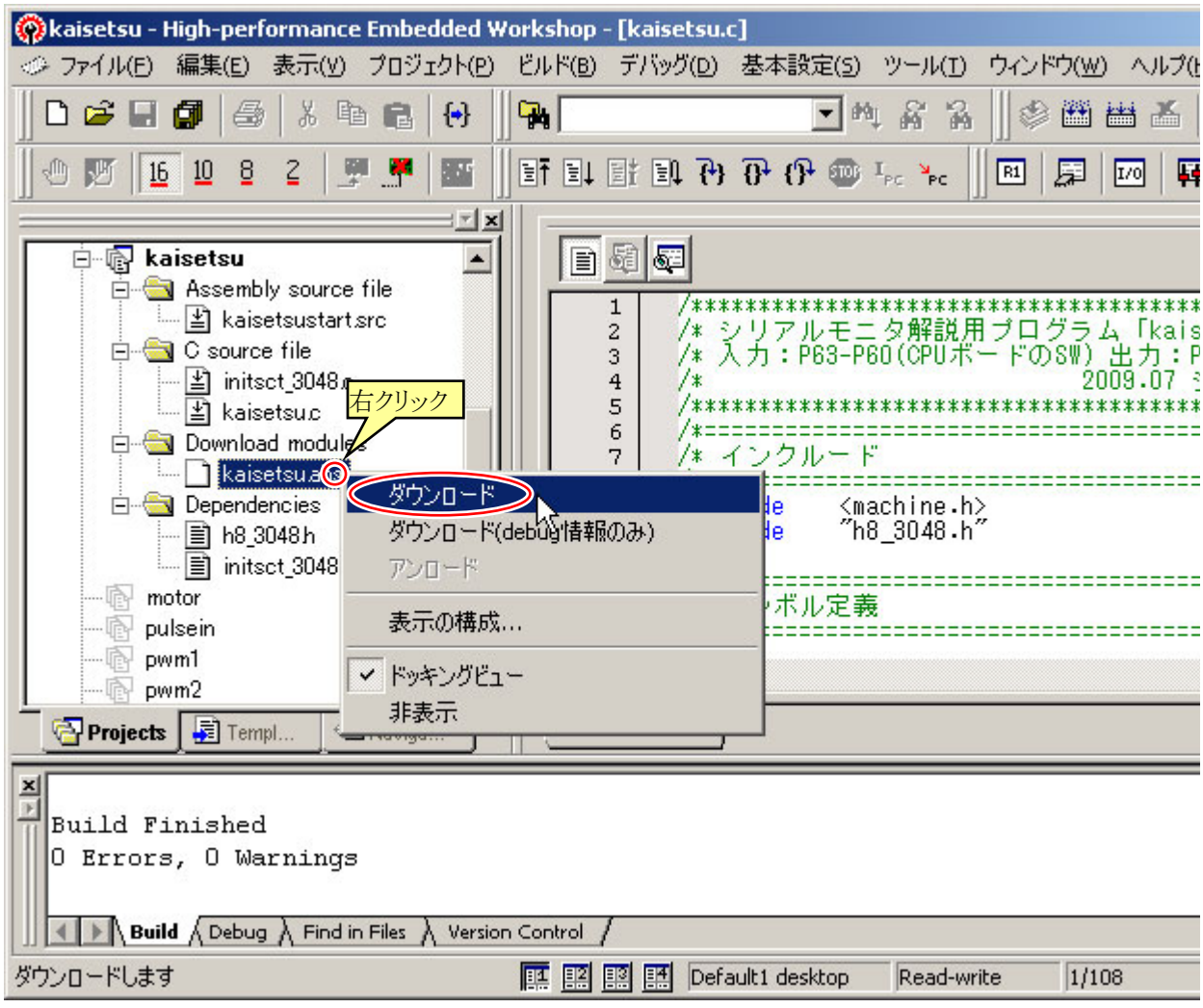
6.プロジェクト「kaisetsu」が有効(太字)か確認します。有効でない場合は、「kaisetsu」を右クリックして「アクティブプロジェクトに設定」をクリックします。



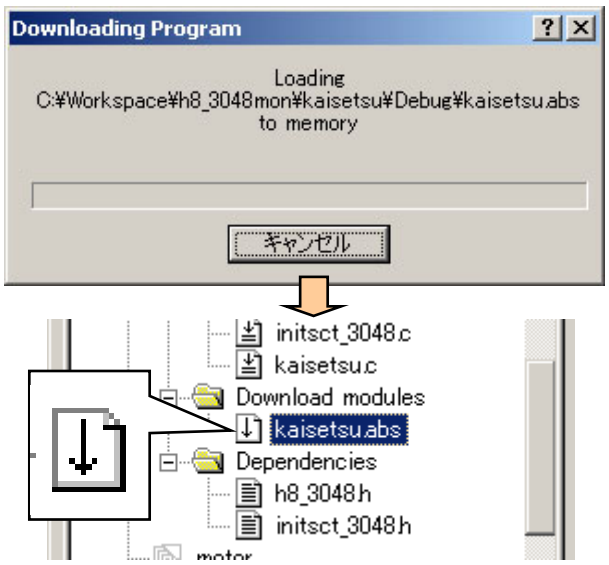
7.「kaisetsu.c」をダブルクリックしてエディタウィンドウに io.c を表示させます。



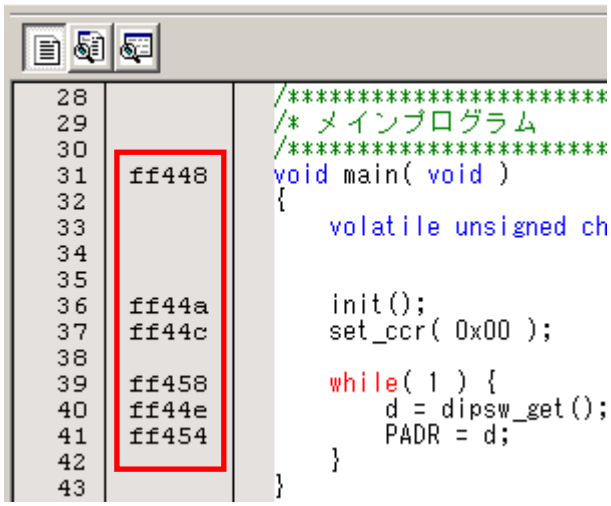
8.「ビルド→ビルド」でビルドして、エラーがないことを確認します。



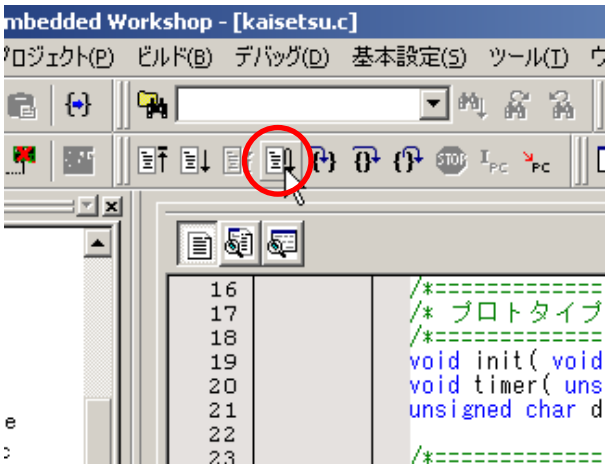
9.「kaisetsu.abs」上で右クリック、「ダウンロード」をクリックします。



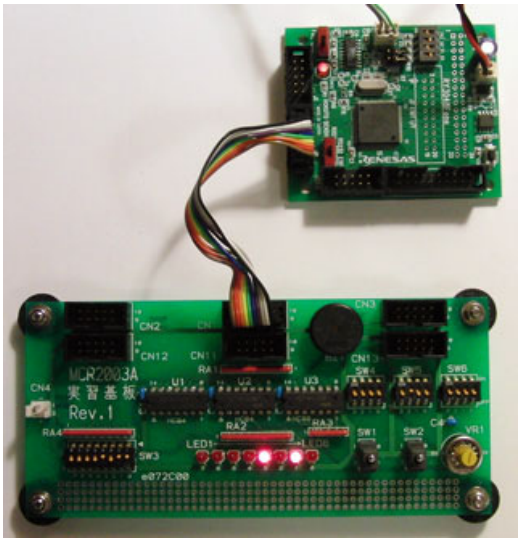
10.ルネサス統合開発環境からマイコンボードへプログラムを転送しています。今まで CpuWrite や FDT などでプログラムを書き込んでいましたが、**シリアルモニタではダウンロード操作で書き込み(RAM へ転送)を行います**。ダウンロードが完了すると、「kaisetsu.abs」の左にあるアイコンに下矢印が付きます。



11.エディタウィンドウの行番号とプログラムの間に列が追加されました。これがプログラムの番地です。init 関数を実行しているプログラムは、0xff44a 番地にあります、ということです。



12.リセット後実行ボタンをクリックします。

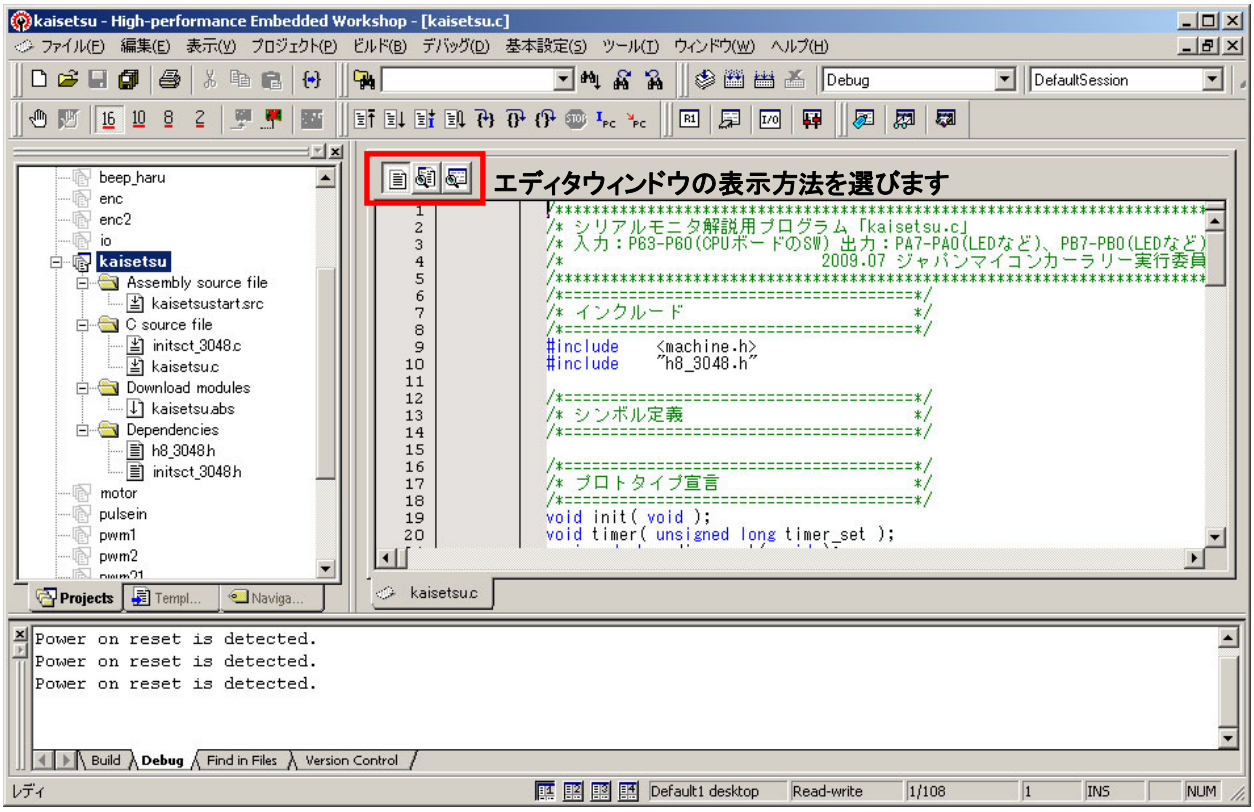


13.マイコンボードのディップスイッチの値が、ポート A に接続されている LED へ出力されるはずです。今までは、ROM に書き込んだプログラムを実行していましたが、今回は RAM に転送したプログラムを実行されています。実行を止めるには、マイコンボードのリセットスイッチを押します。

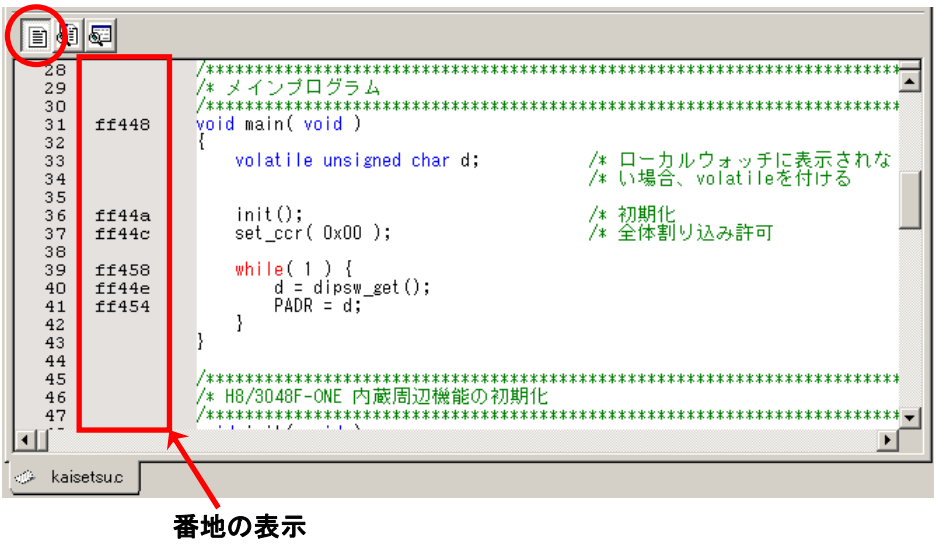
3.2　ルネサス統合開発環境の操作

3.2.1　エディタウィンドウの表示方法

□で囲った部分に 3 つのボタンがあります。このアイコンで、エディタウィンドウの表示方法を選ぶことができます。



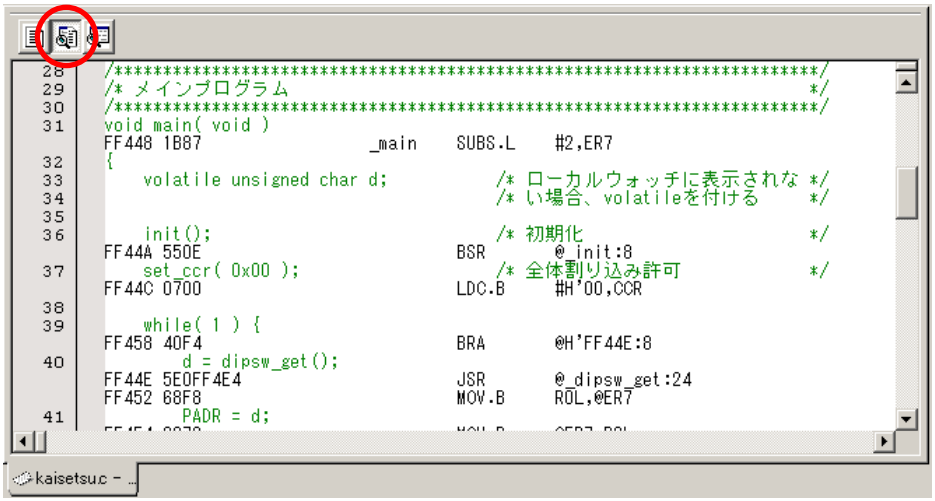
(1) ソースモードで表示



3 つの内の左側のボタンは、「ソースモードで表示」ボタンです。

左画面のように C 言語ソースが表示されます。ただし、行番号とプログラムの間に番地が表示されます。

(2) 混合モードで表示



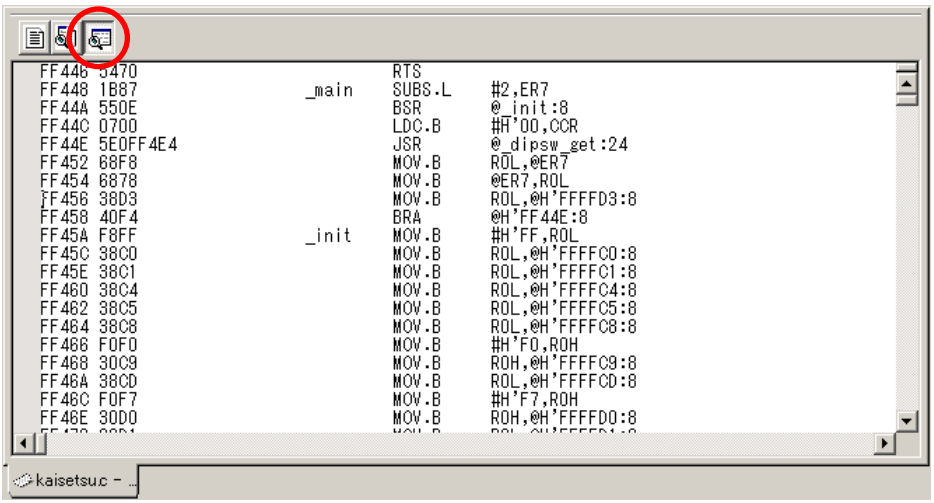
3 つの内の真ん中のボタンは、「混合モードで表示」ボタンです。

例えば、42 行目の下に「FF450 F8FF」とありますが、これは、0xff450 番地に 0xf8、0xff451 番地に 0xff のデータがありますよ、ということになります。

混合モードで表示中は、プログラムを変更することができません。

(3) 逆アセンブリモードで表示

3 つの内の右側のボタンは、「逆アセンブリモードで表示」ボタンです。

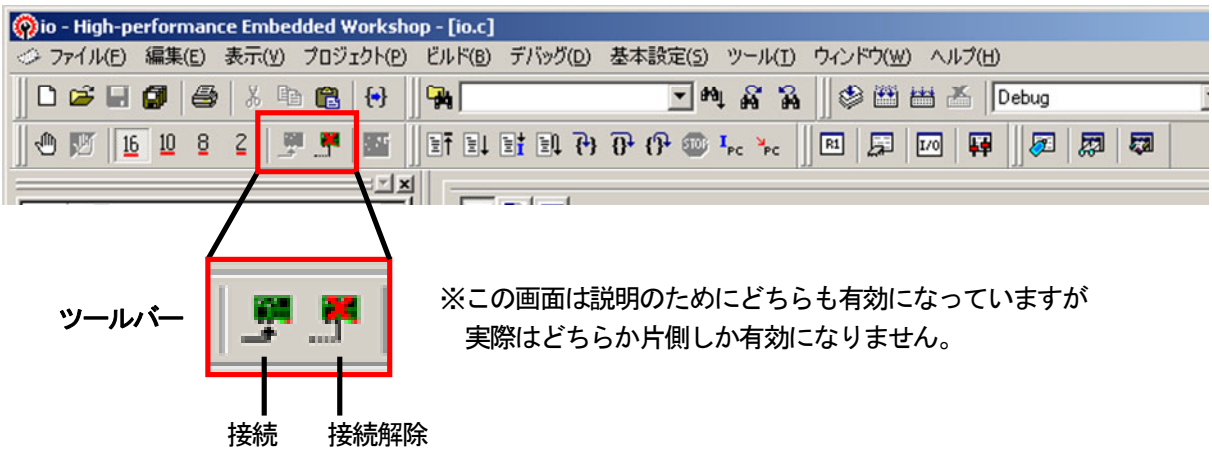


アセンブリ言語での表示となります。

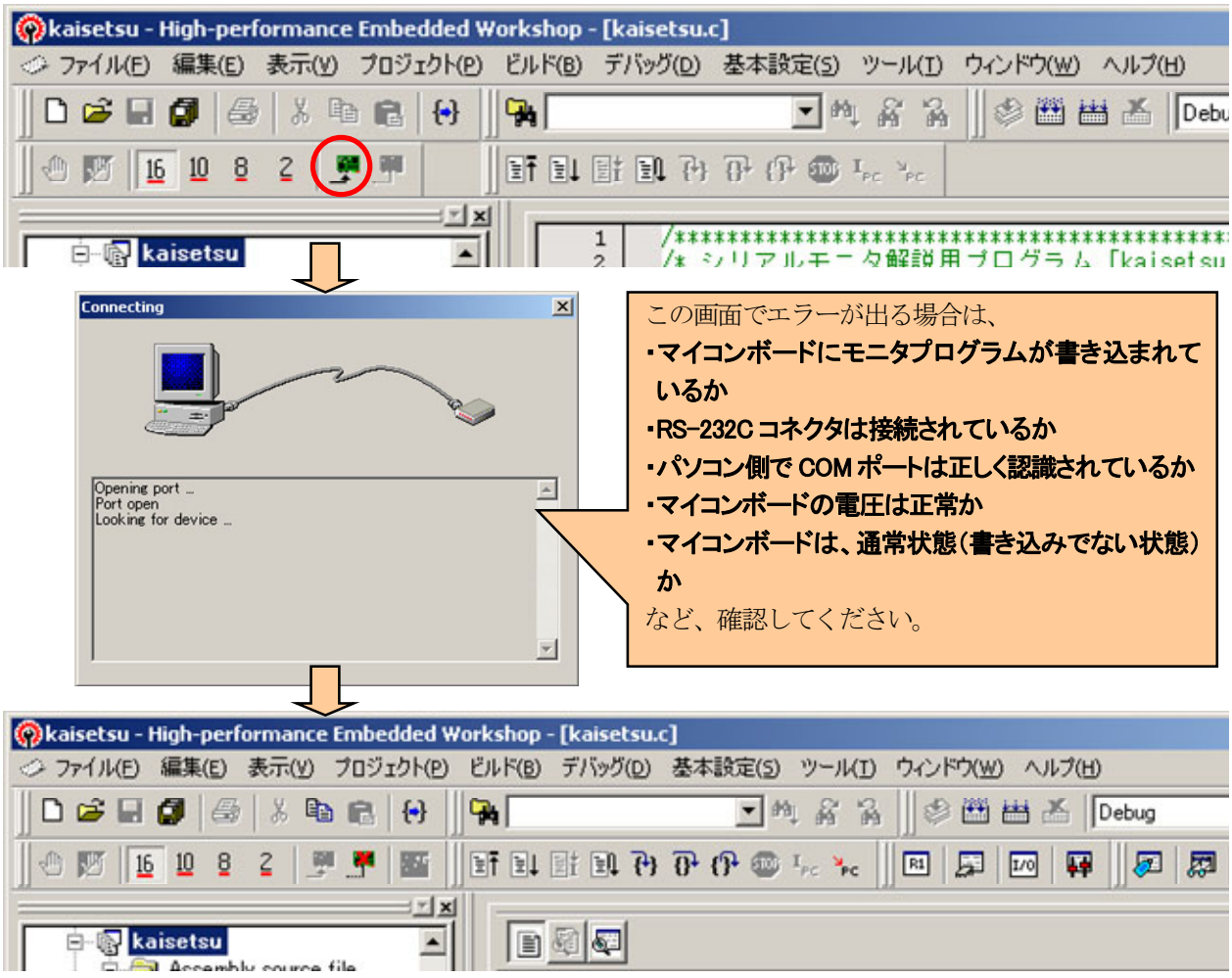
逆アセンブリモードで表示中は、プログラムを変更することができません。

3.2.2　ツールバー（接続）

□で囲ったツールバーが、ルネサス統合開発環境とマイコンボードを接続するボタンです。



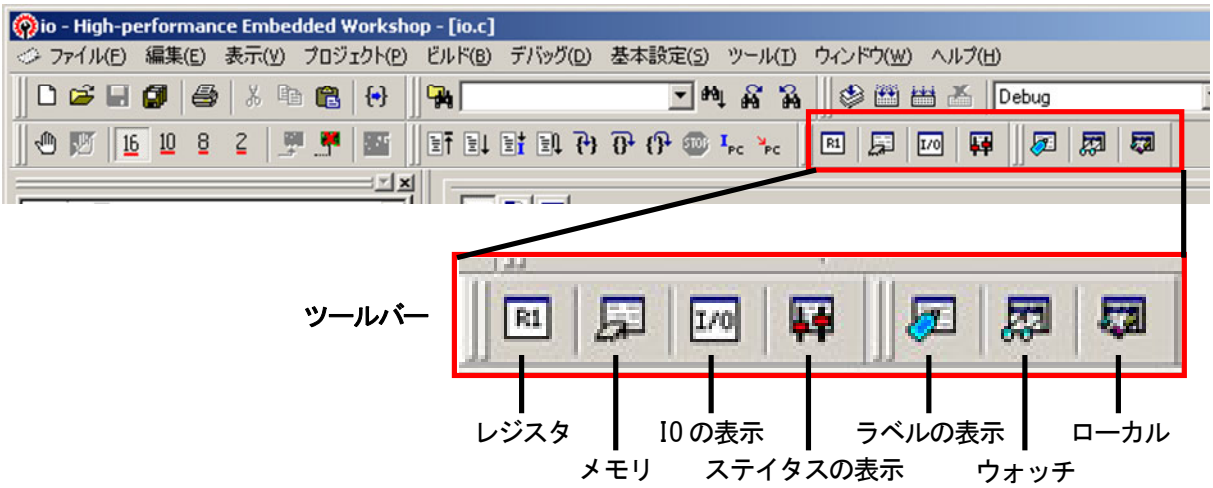
接続ボタン  で、ルネサス統合開発環境とマイコンボードを接続します。もし、接続ボタンが有効になっていなければ、マイコンボードと通信ができていませんので、接続ボタンで接続してください。



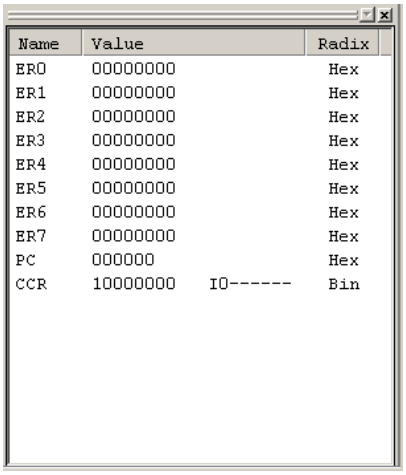
マイコンボードの電源を切る場合など、切断する場合は、切断ボタン  で切断してください。

3.2.3 ツールバー（情報表示関係）

□で囲ったツールバーが、マイコンの様々な情報を表示するボタンです。



(1) レジスタ

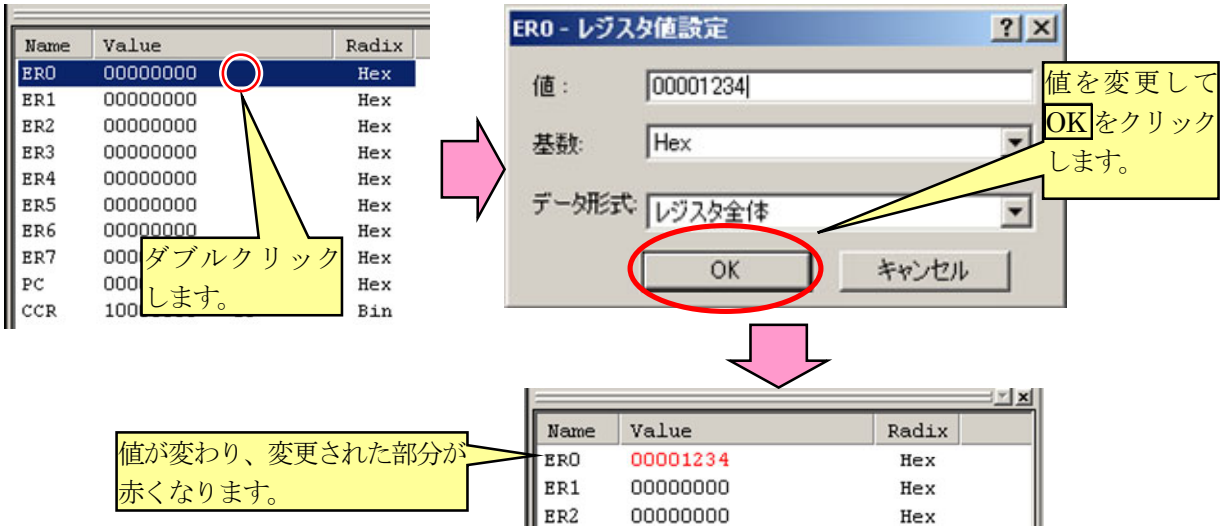


Name	Value	Radix
ER0	00000000	Hex
ER1	00000000	Hex
ER2	00000000	Hex
ER3	00000000	Hex
ER4	00000000	Hex
ER5	00000000	Hex
ER6	00000000	Hex
ER7	00000000	Hex
PC	000000	Hex
CCR	10000000 IO------	Bin

レジスタボタン R1 をクリックすると、レジスタの値を表示するレジスタウィンドウが表示されます。

ウィンドウを消したいときは、×で消すことができます（以下、全てのウィンドウ）。

値部分をダブルクリックすると、入力画面が現れ、値の変更をすることができます。

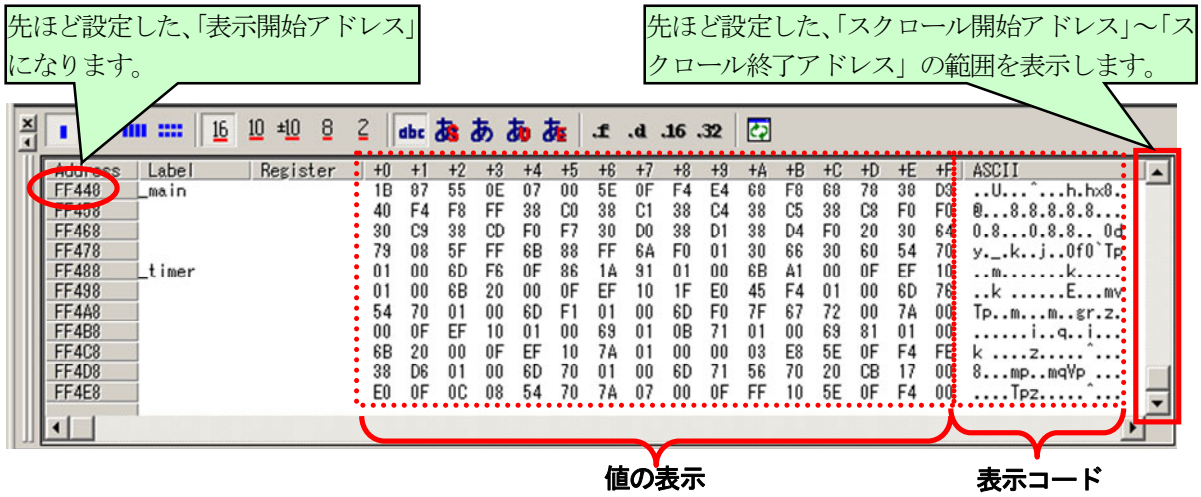


(2) メモリ

メモリボタン  をクリックすると、メモリの値を表示、変更することのできるメモリウィンドウが表示されます。

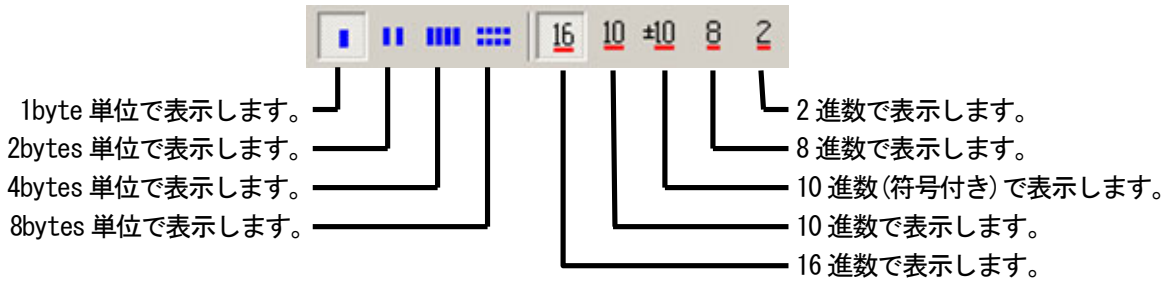
最初に、表示開始アドレスを入力します。

OKをクリックすると、下の画面のように、メモリの値が表示されます。



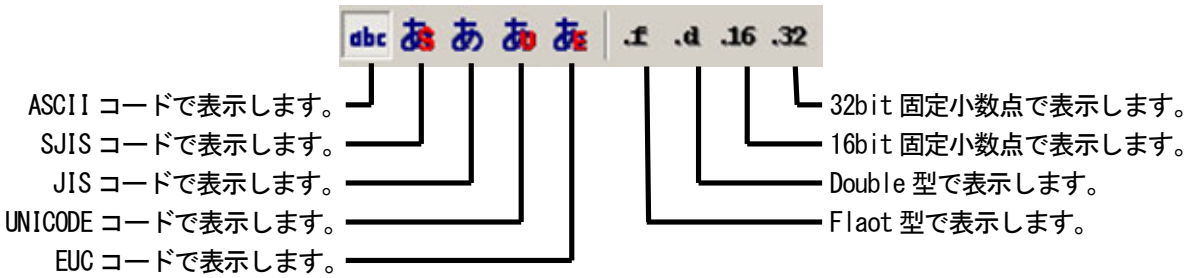
■値の表示

値の表示方法を選ぶことができます。どのように値が表示されるか、試してみてください。



■表示コード

表示コードの表示方法を選ぶことができます。どのように表示コードが表示されるか、試してみてください。

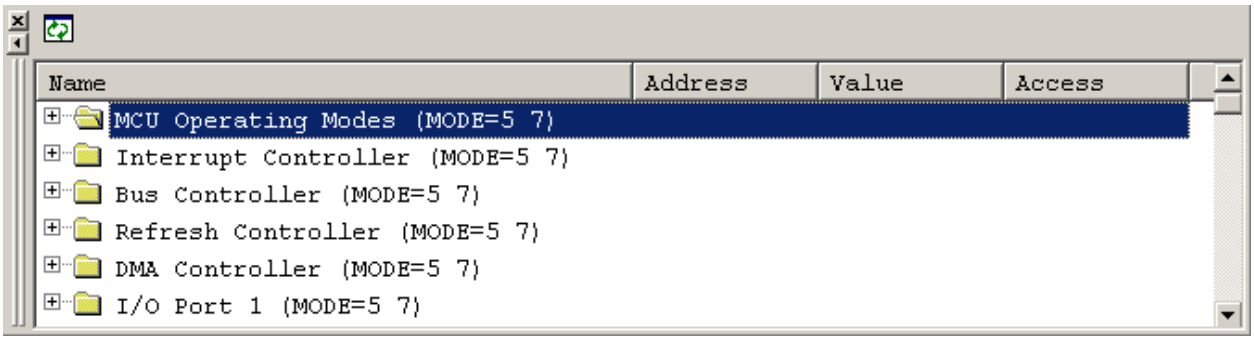


■値の更新

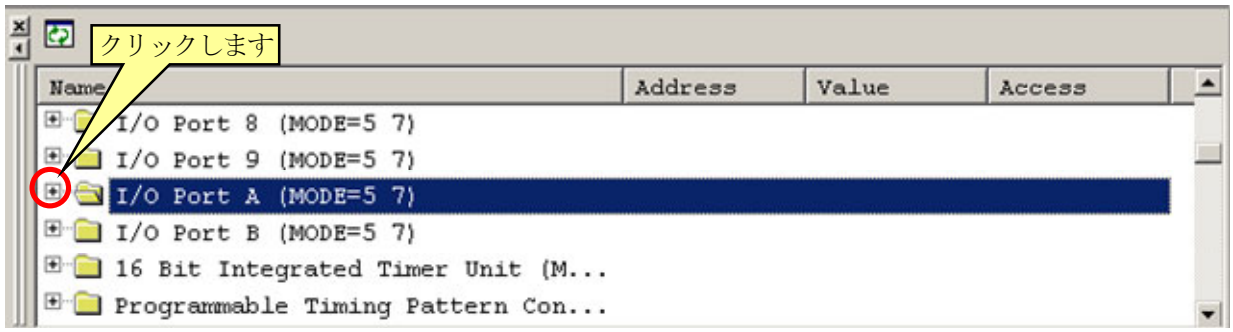
 ボタンで、値を最新の状態に更新します。

(3) IOの表示

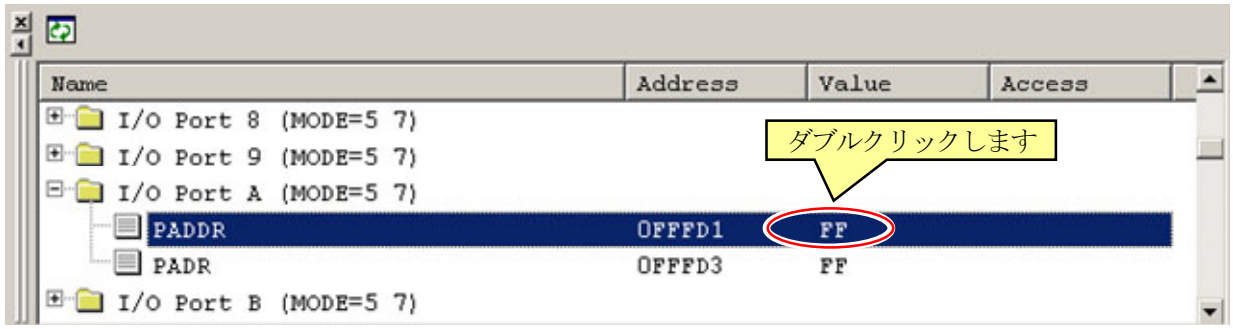
IO の表示ボタン  をクリックすると、内蔵周辺機能の I/O レジスタの値を表示、変更することのできる IO ウィンドウが表示されます。**表示や変更は、プログラムの実行が止まっている状態で行ってください。**



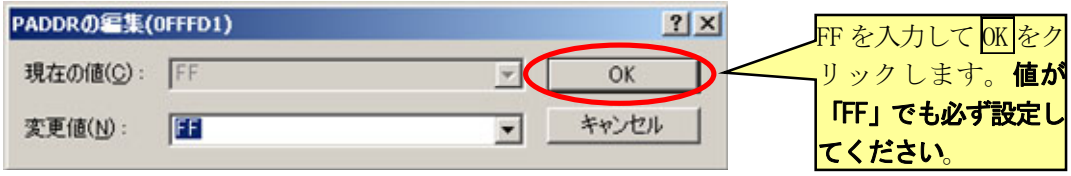
例として、ポート A に 0x55 を出力してみます。「I/O Port A (MODE=5 7)」を探して、+をクリックします。



「PADDR」をダブルクリックします。



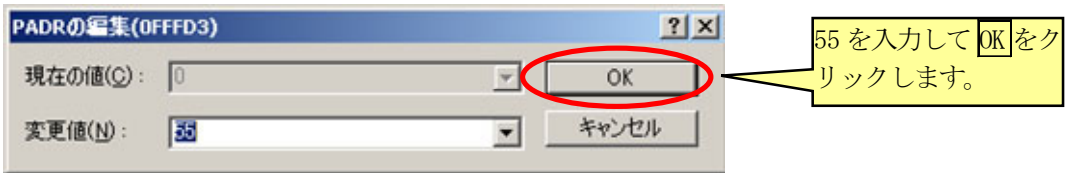
変更値を「FF」にして、OK をクリックします。**現在の値が「FF」でも必ず行ってください。**



※「P○DDR」レジスタについて

DDR（データディレクションレジスタ）は、どのような値を設定しても表示は「FF」になります。これは、H8/3048F-ONE マイコンの仕様です。「FF」が表示されているから「FF」の値が設定されているわけではありません。DDR は必ず1度は設定してください。

「PADR」の行が赤く変わりました。値が変わった場合、赤く表示されます。「PADR」をダブルクリックします。変更値に「55」を入力します。値は、16 進数になります。

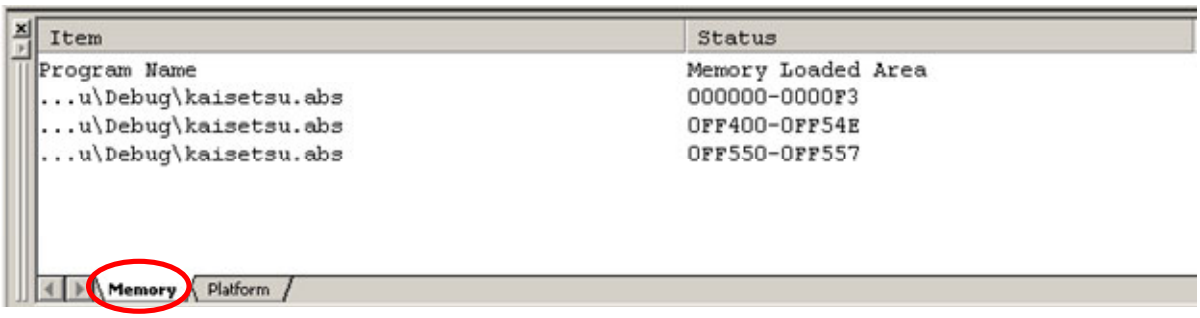


これで、ポートＡから 0x55 が出力されます。LED 基板などを接続して確認してみてください。他のレジスタも同様に変更することができます。また、○部分でレジスタの値(Value 部分)を更新することができます。

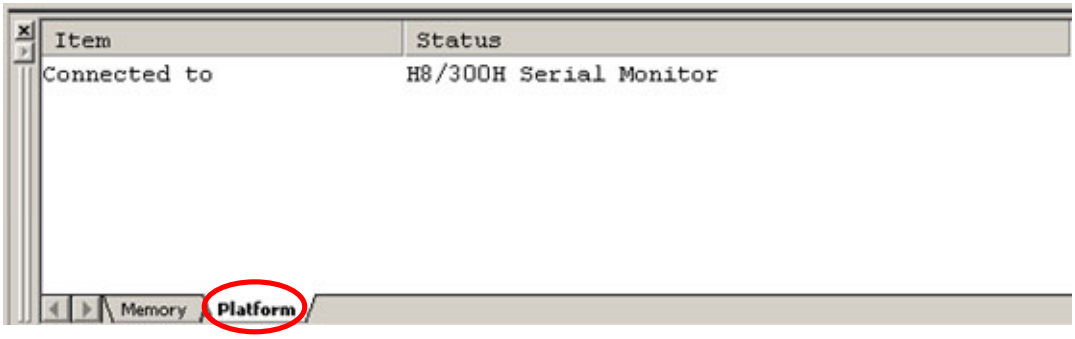
(4) ステイタスの表示

ステイタスの表示ボタン  をクリックすると、メモリマップに関する情報などのステイタスウィンドウが表示されます。

「Memory」タブは、メモリマッピングおよび現在ロードしたオブジェクトファイルが使用するメモリエリアなど、現在のメモリステータスに関する情報を表示します。

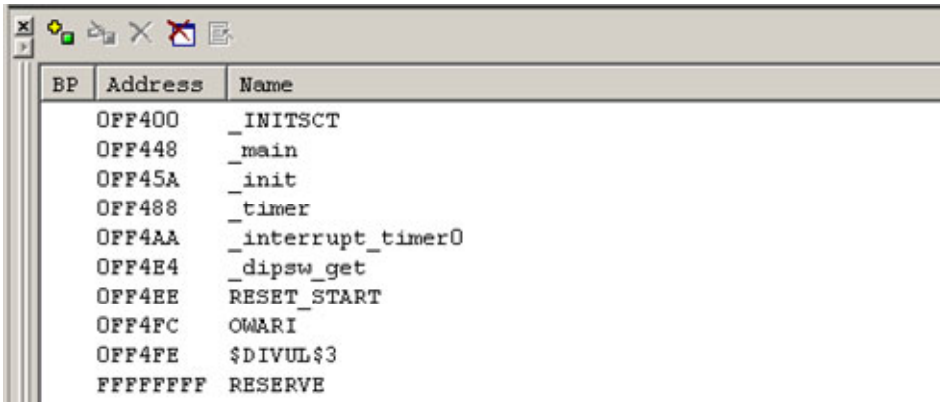


「Platform」タブは、CPU シリーズおよび動作モードなど、デバッギングプラットフォームのステータス情報、実行状態および実行統計情報を表示します。



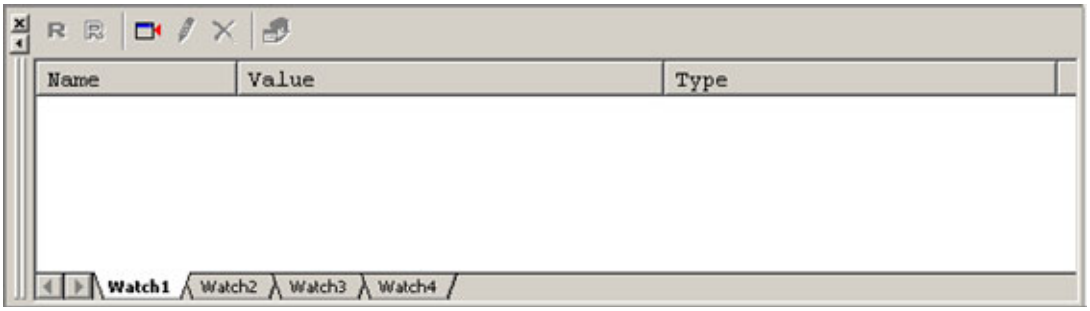
(5) ラベルの表示

ラベルの表示ボタン  をクリックすると、関数などのラベルのアドレスを表示するラベルウィンドウが表示されます。

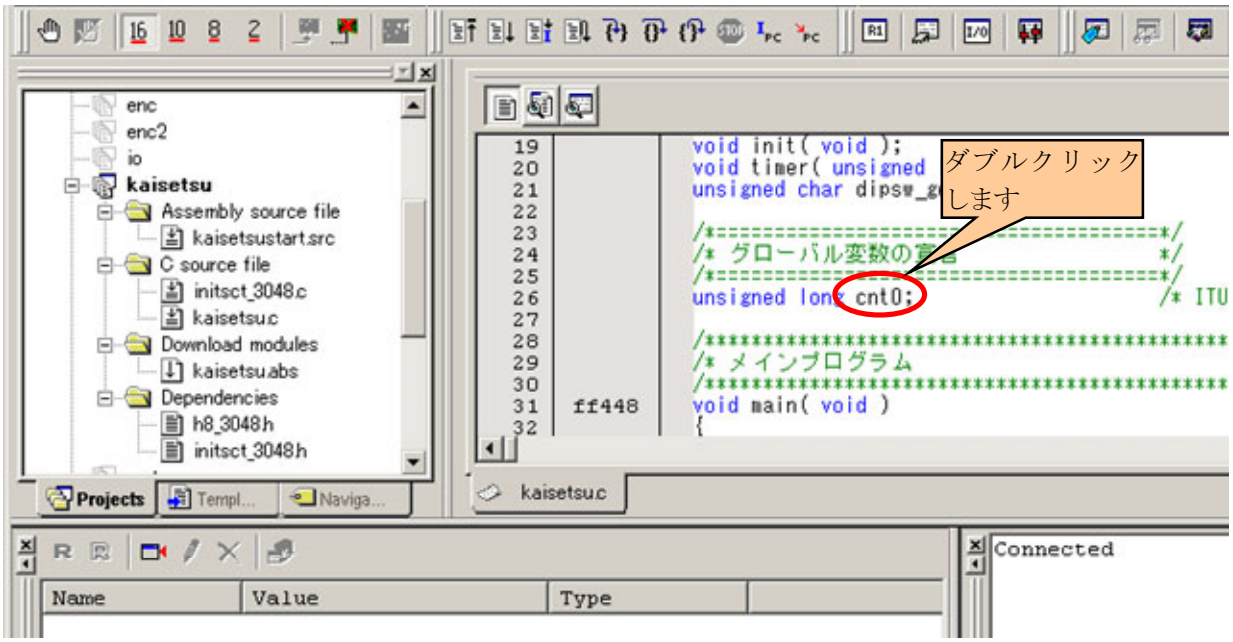


(6) ウォッチ

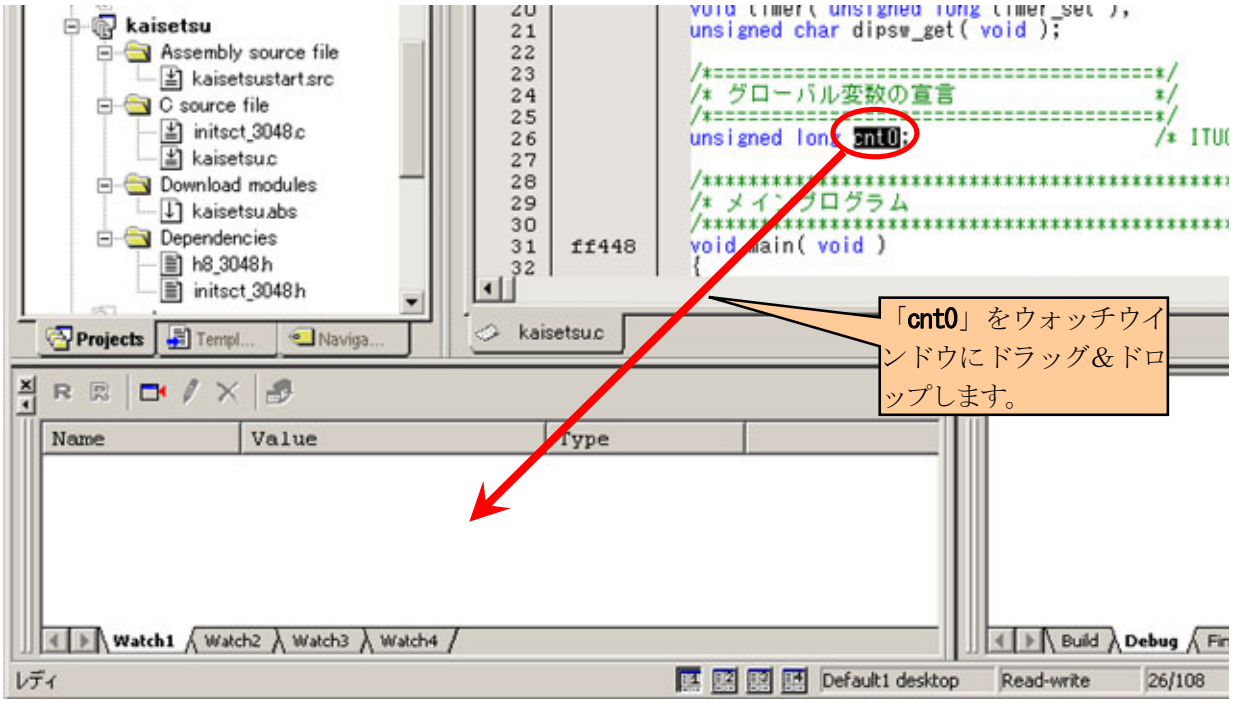
ウォッチボタン  をクリックすると、変数の値を表示、変更することのできるウォッチウィンドウが表示されます。ただし、**ここで表示できるのはグローバル変数だけです。ローカル変数は表示できません。**ローカル変数は、次のローカルウィンドウで表示することができます。



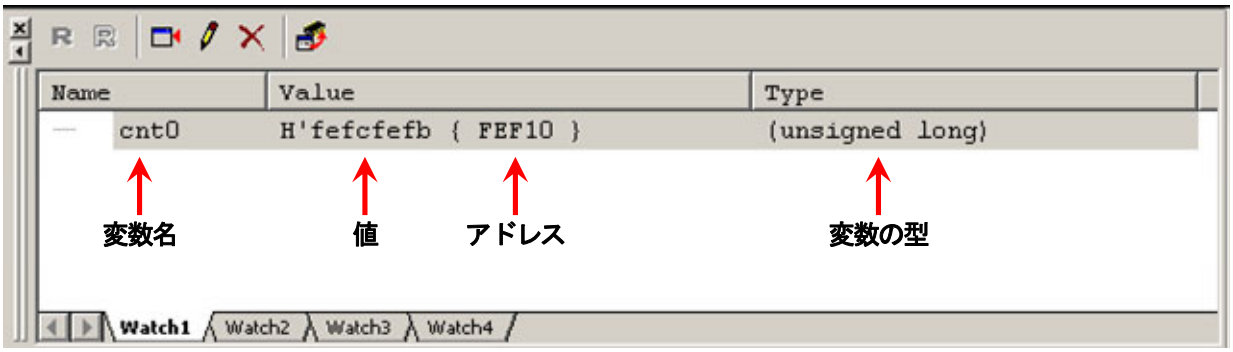
ここでは例として、プロジェクト「timer2」を有効にして「timer2.c」ファイルを開きます。プロジェクトを変えたときは、「timer2.abs 上で右クリック→ダウンロード」でファイルをダウンロード（マイコンに転送）するのを忘れないでください。
ウォッチボタンをクリックして、ウォッチウィンドウを表示させておきます。ウォッチウィンドウが狭い場合は、適宜広げておきます。
「cnt0」をダブルクリックして選択状態にします。



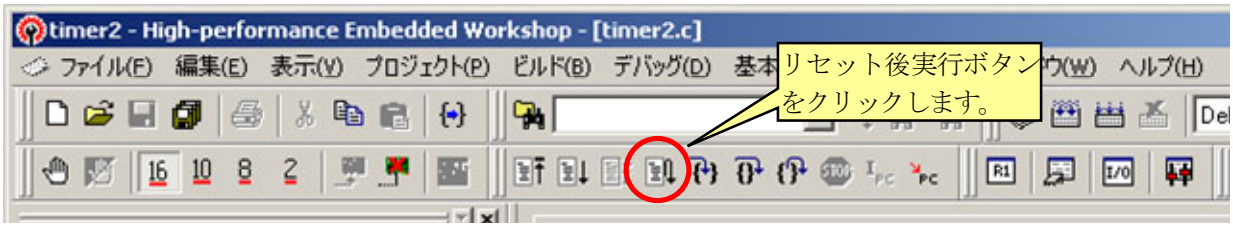
「cnt0」をウォッチウィンドウにドラッグ＆ドロップします。



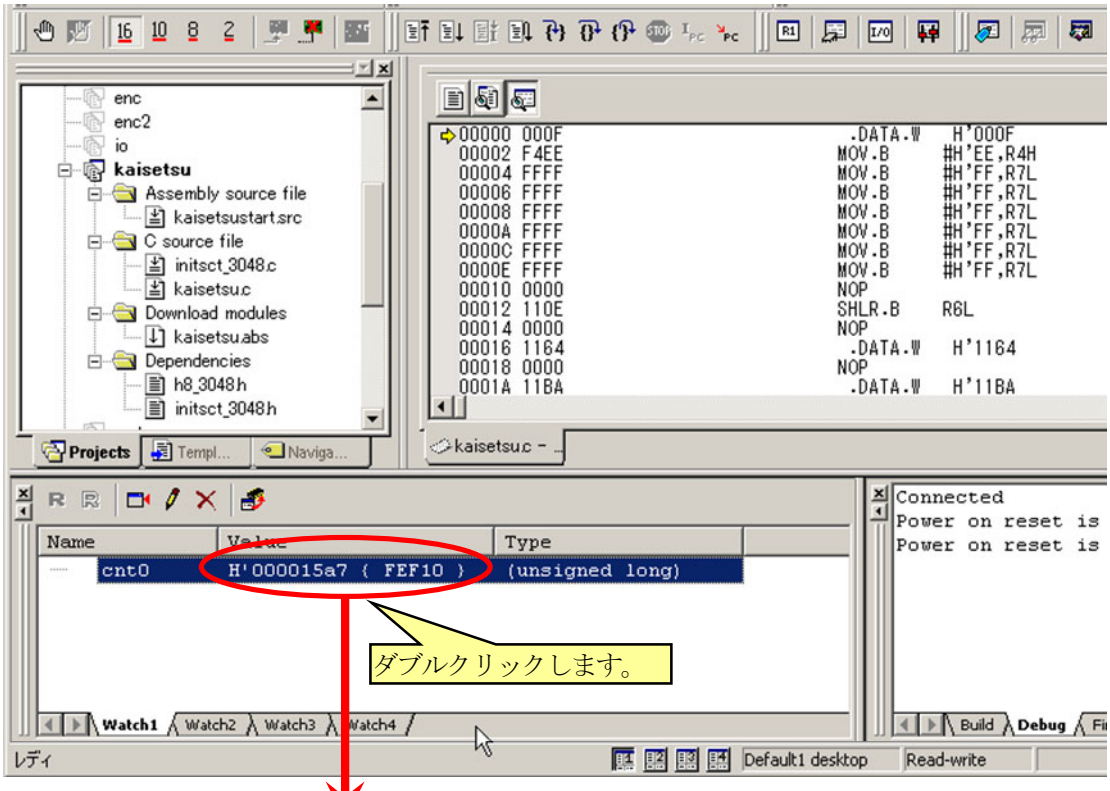
変数の値が表示されました。今回の例では今回の例ではダウンロード直後なので「fe fc fe fb」という不定の値になっています。



リセット後実行ボタンをクリックして実行します。数秒後、マイコンボードのリセットスイッチを押して停止させます。

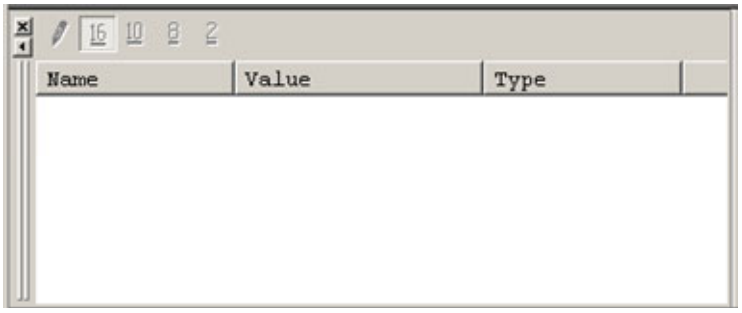


cnt0 の値が「00 00 01 76」の状態で停止しました。値部分をダブルクリックすると手動で値を変更することができます。

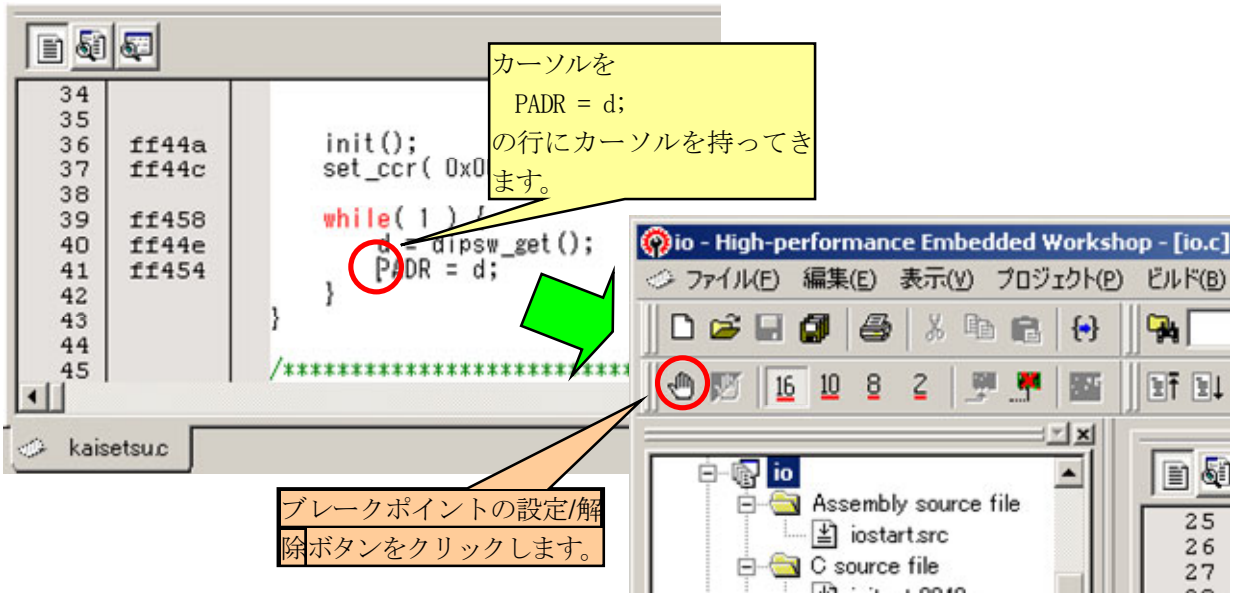


(7) ローカル

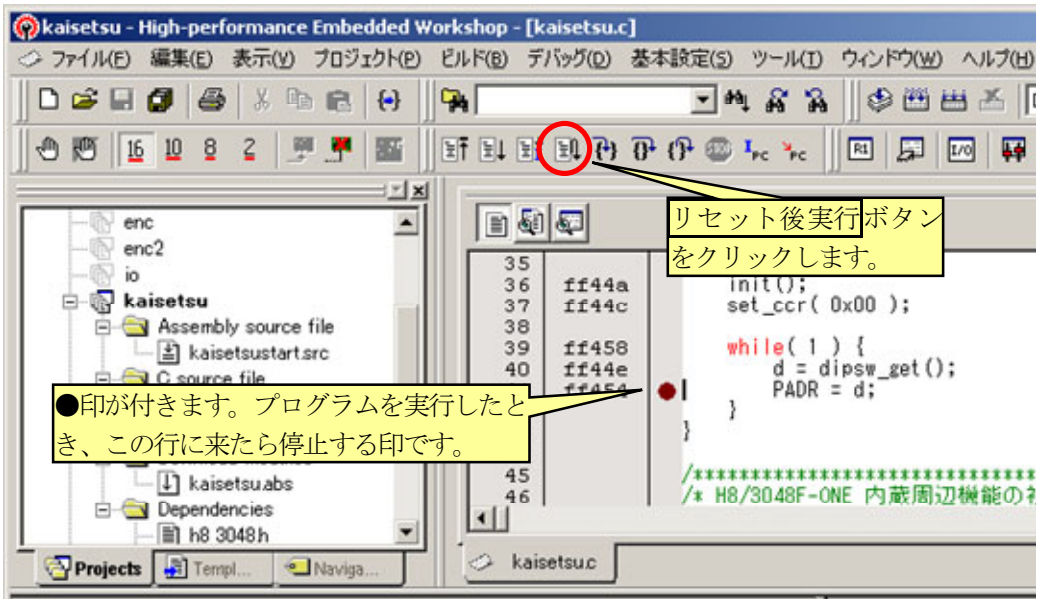
ローカルボタン  をクリックすると、ローカル変数の値を表示、変更することのできるローカルウィンドウが表示されます。ローカルウィンドウは、自動的に変数が追加、削除されます。



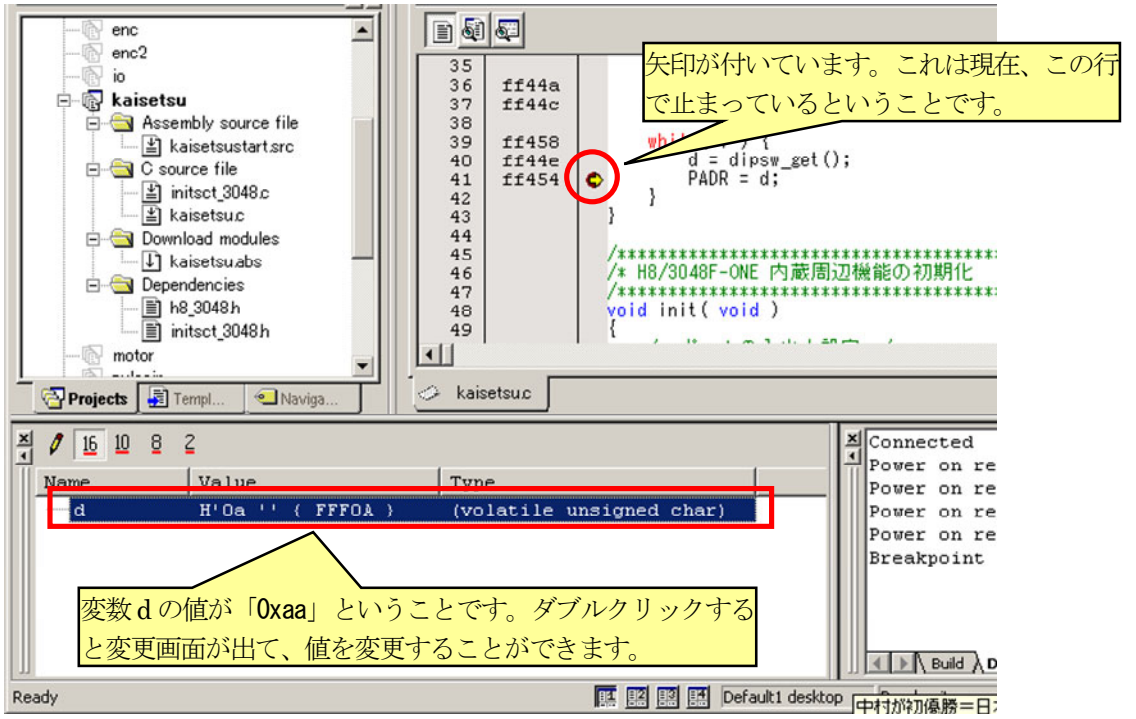
「PADR = d;」の行にカーソルを持ってきて、ブレークポイントの設定/解除ボタンをクリックします。



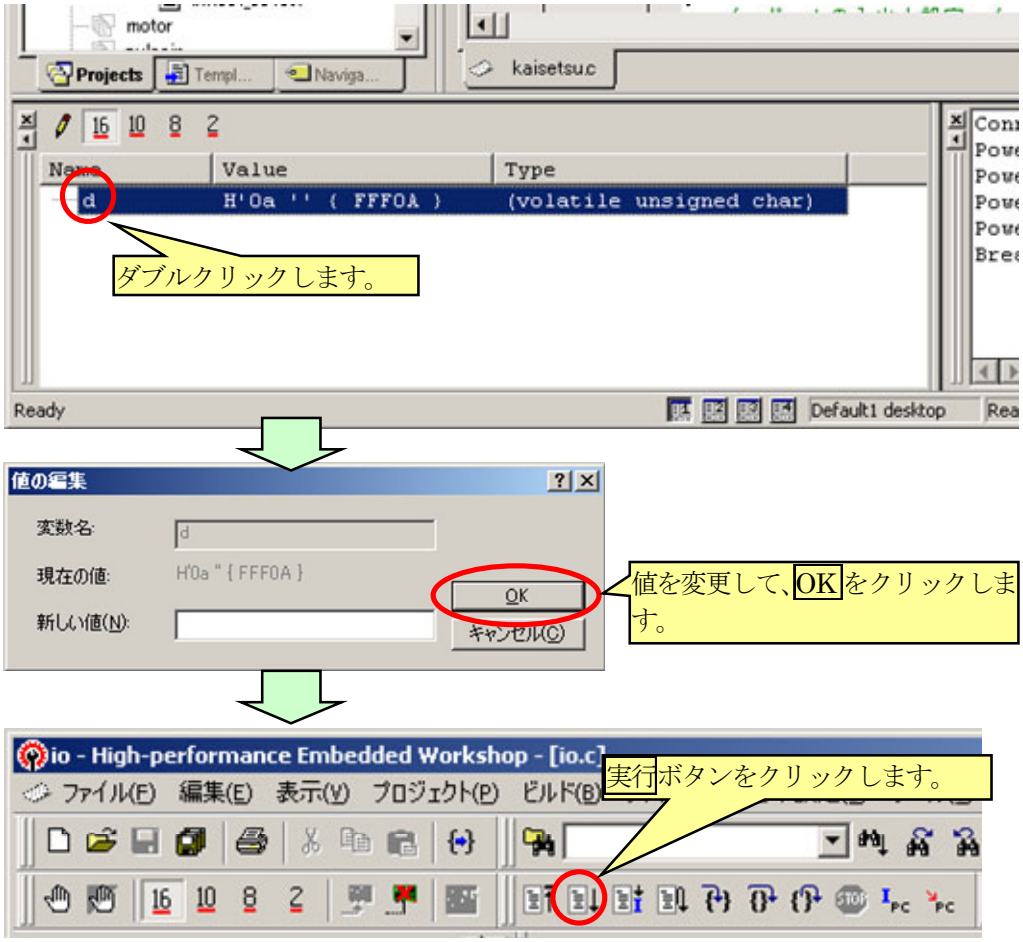
ブレークポイントの設定/解除ボタンをクリックすると、行の左に●印が付きます。これは、プログラムを実行したとき、この行に来たら停止させる印です。ブレークポイントを設定した後、リセット後実行ボタンをクリックします。
※マイコンボードのディップスイッチの状態は、「1010（0x0a）」とします。



エディタウィンドウ部分に矢印が付いています。これは矢印の 1 つ前の行までのプログラムを実行して、この行に来たということです。そのため、この行自体はまだ実行していません。このとき、IO ウィンドウには、ローカル変数 d の値が表示されています。1 行手前のプログラムは、ポート 7 の状態を読み込んで変数 d に入れるプログラムです。したがって、変数 d の値は、ポート 7 の入力状態と同じ値と言うことになります。



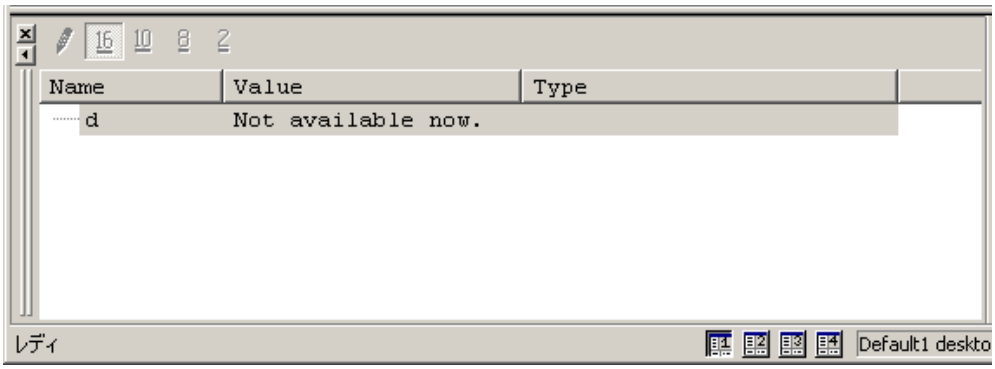
変数 d の値を変更してみます。ここでは「55」(16 進数になります)と入力します(何でも構いません)。実行ボタンをクリックして、この続きからプログラムを実行します。



プログラムどおりなら、マイコンボードのディップスイッチの値がポート A の LED に出力されるはずです。しかし、ポート A に接続されている LED には「0x55」が出力されます。これは、ポート A に出力する前にローカルウィンドウで変数 d の値を変えたためです。このように、ローカル変数の値を表示したり変更したりすることができます。

※ローカル変数が「Not available now.」と表示された場合

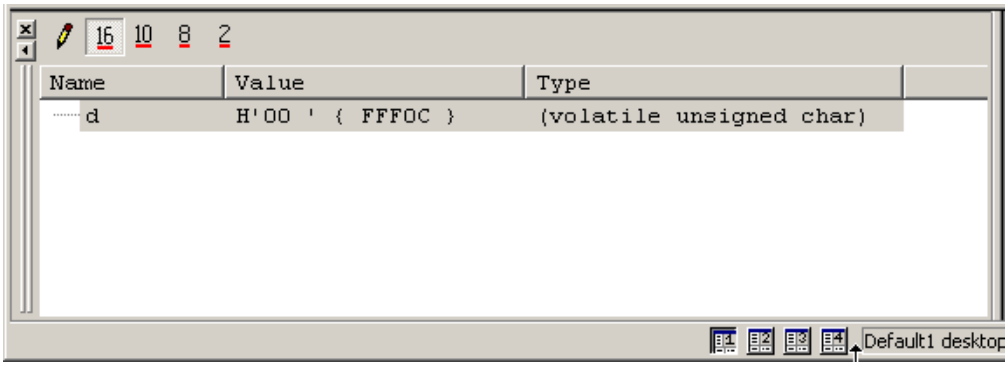
「Not available now.」は、「現在、利用可能ではありません。」ということです。ローカル変数が、RAM に割当たっている場合は常に表示できますが、最適化などによってレジスタなどに割当たった場合は、その変数のある行を実行した直後しか表示できません。



常に表示させたい場合(RAM に割り当てたい場合)、変数の宣言に「volatile」を追加します。

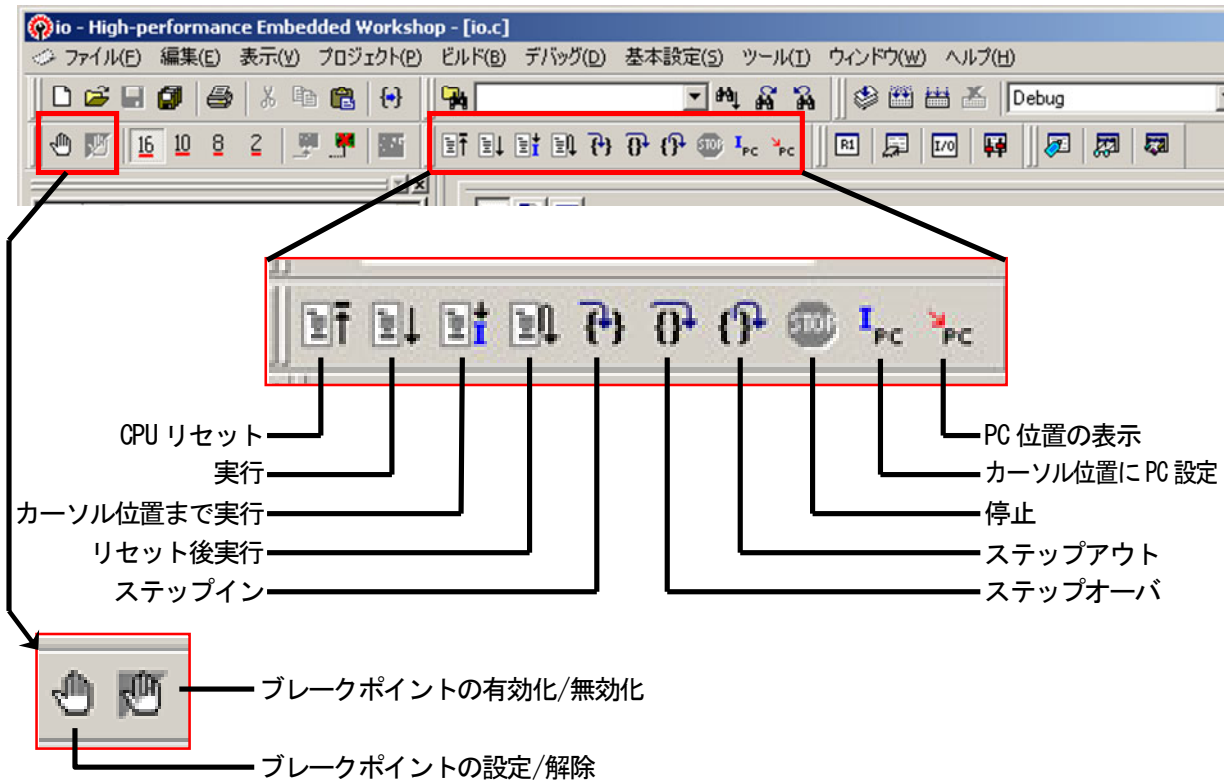
```
unsigned char d;
  ↓
volatile unsigned char d;
```

「ビルド→ビルド」を実行して、ダウンロードすると、次のようにローカル変数が表示されます。



3.2.4 ツールバー(デバッグ関係)

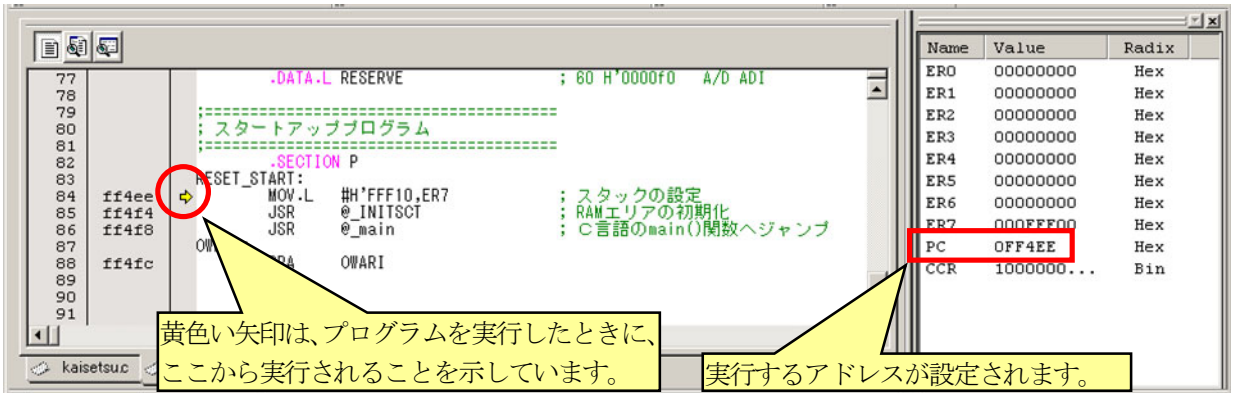
□で囲ったツールバーが、プログラムのデバッグに関するボタンです。



(1) CPUリセット

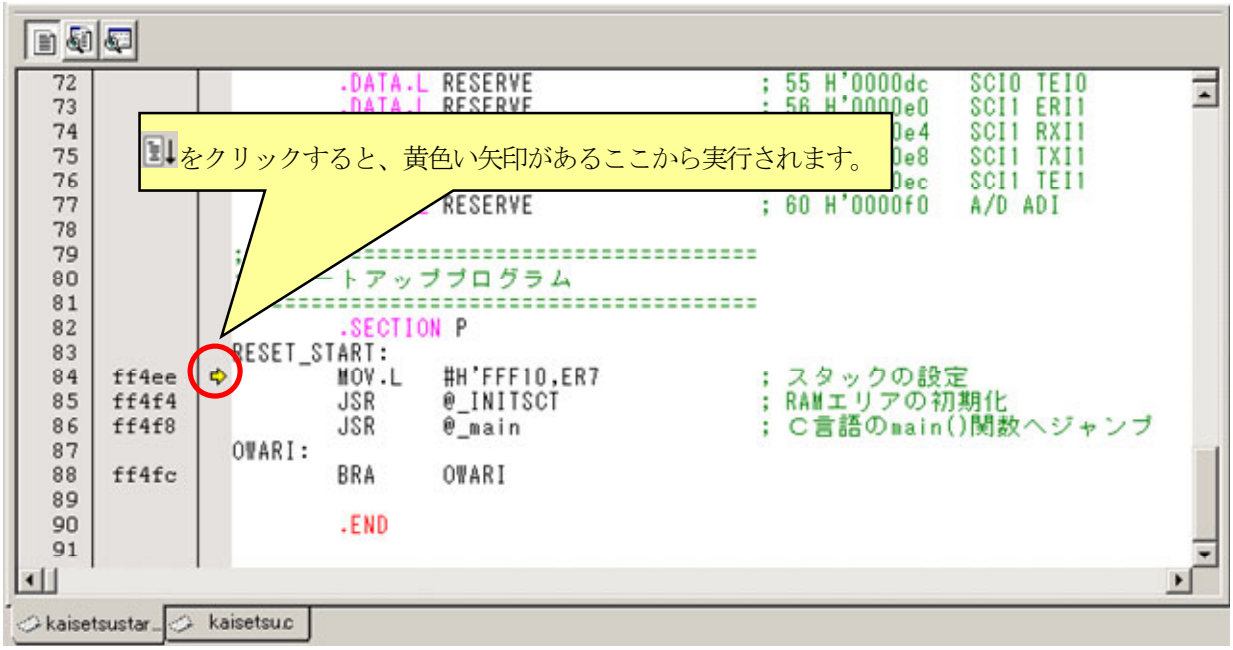
CPUリセットボタンをクリックすると、プログラムカウンタ(PC)をセットして、プログラムが実行できる状態にします。マイコンボードのリセットスイッチを押したのとは違います。

プログラムカウンタ(PC)には、実行するアドレスが入っています。一番最初、kaisetsustart.src の「RESET_START」から実行されます。その為、PC には、「0FF4ee」が入ります。エディタウィンドウの黄色い矢印は、プログラムを実行したときに、動作する行を示しています。



(2) 実行

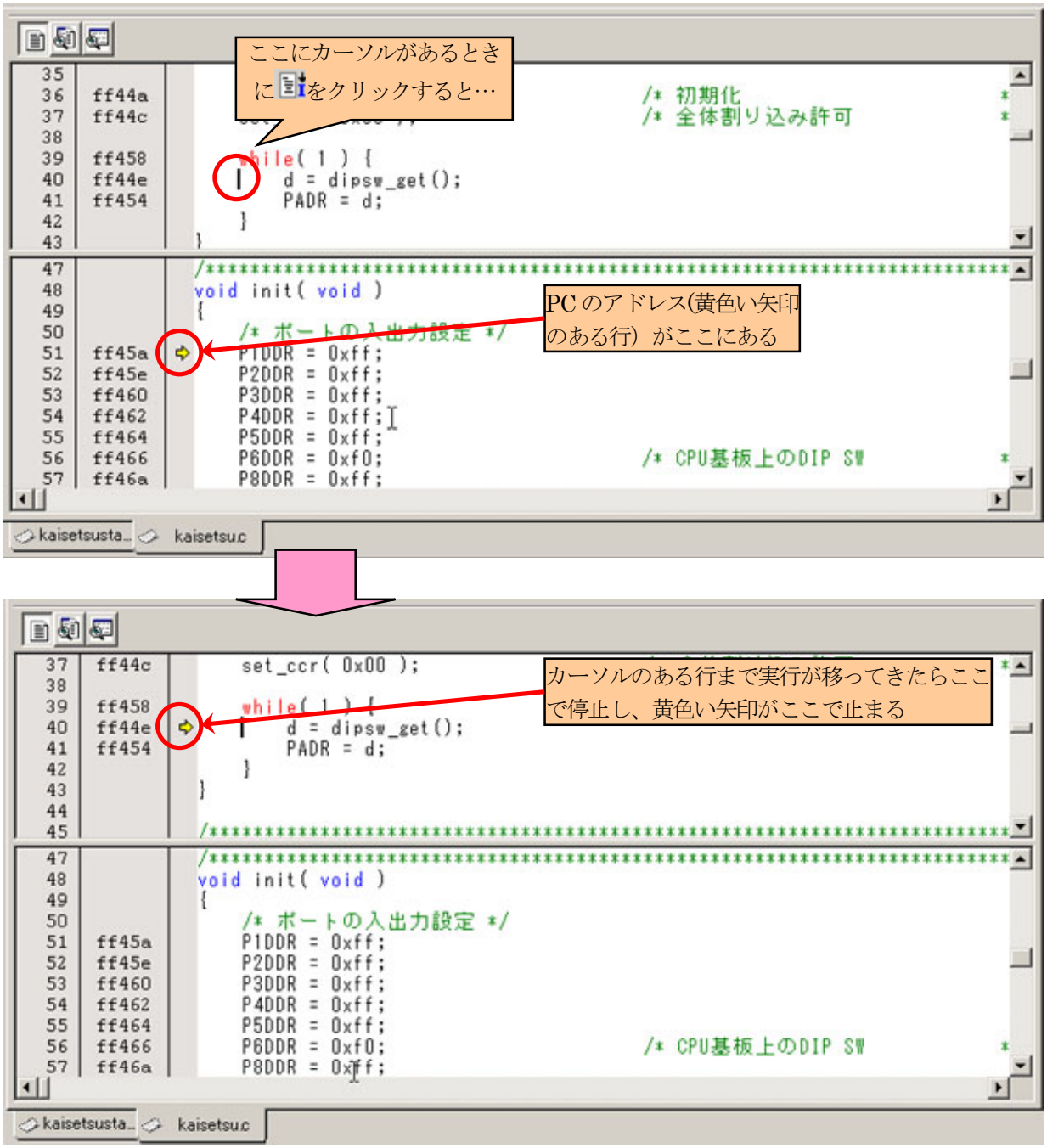
実行ボタンをクリックすると、PC(プログラムカウンタ)のアドレス(黄色い矢印のある行)からプログラムを実行します。F5 キーでも同様です。黄色い矢印は消えて、現在プログラム実行していることを示します。**プログラムを止めたいときは、マイコンボードのリセットスイッチを押します**。マイコンボードのリセットスイッチは実行ボタンでプログラムを実行したときに、止めたいときだけ使ってください。マイコンボードのリセットスイッチを押すと、黄色い矢印が 0000 番地で停止します。CPUリセットボタンとマイコンボードのリセットスイッチを押すことは、全く違う動作です。



(3) カーソル位置まで実行

カーソル位置まで実行ボタンをクリックすると、PC のアドレス(黄色い矢印のある行)からプログラムを実行し、現在カーソルのある行が実行されようとすると動作が停止します。現在カーソルのある行自体は実行されません。

現在カーソルのある行が実行されるかどうか、確認するときに使用すると便利です。停止すれば、実行されようとしていることになり、プログラムが停止しなければ、現在カーソルのある行は実行されないと言うことになります。



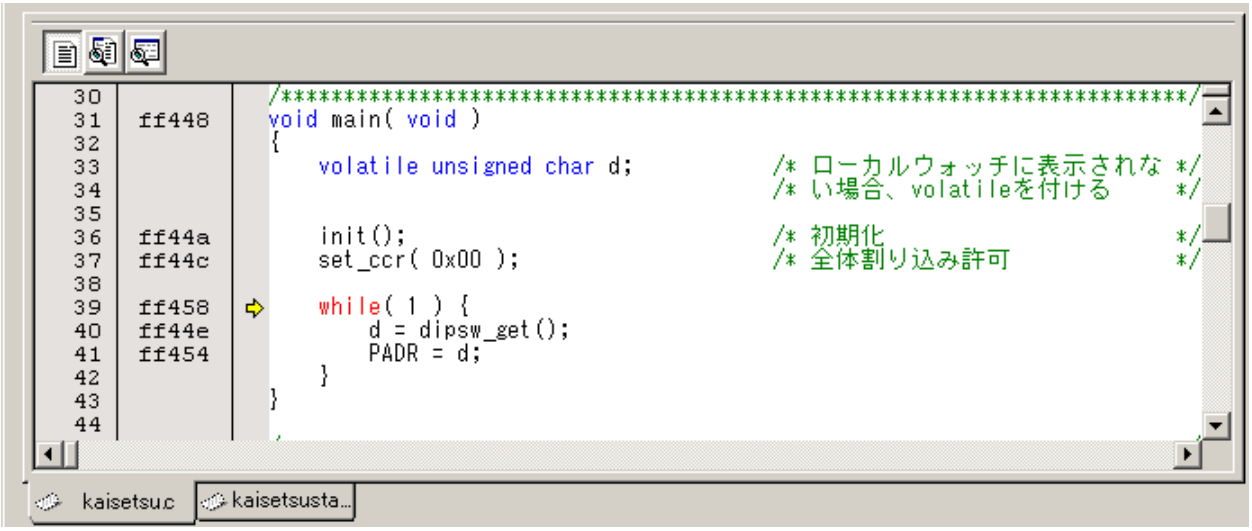
(4) リセット後実行

リセット後実行ボタンは、「CPU リセット」と「実行」を連続して行うボタンです。SHIFT+F5 キーでも同様です。プログラムの停止は、マイコンボードのリセットスイッチを押します。

(5) ステップイン

ステップインボタン をクリックすると、プログラムを1行ずつ実行することができます。F11 キーでも同様です。

次のように、プログラム実行位置が「while(1)」の行とします。



ここで、ステップインボタンを押すと、現在矢印のある行を実行して、次の行に矢印が進んで止まります。関数があった場合、リスト 3. 1のように、関数内のプログラムも含めて矢印が進んでいきます。

リスト 3. 1

```
void main( void )
{
    int ad;

    init();                          /* 初期化                    */

    while( 1 ) {
        d = dipsw_get();
        PADR = ad;
    }
}
中略
unsigned char dipsw_get( void )
{
    unsigned char sw;

    sw  = ~P6DR;                     /* ディップスイッチ読み込み  */
    sw &= 0x0f;

    return i;
}
```

dipsw_get 関数へジャンプ

ジャンプした
次の行へ戻る

(6) ステップオーバ

ステップオーバボタン  をクリックすると、プログラムを1行ずつ実行することができます。F10 キーでも同様です。ステップインとの違いは、関数があっても関数内のプログラムは含めずに矢印が進んでいきます(リスト 3. 2)。今回の場合は、ad関数内のプログラムは一気に実行されて、次の行へ進みます。

リスト 3. 2

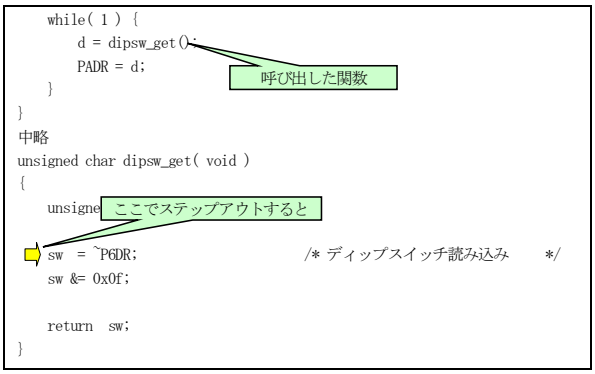
```
void main( void )
{
    int ad;

    init();                              /* 初期化                    */

    while( 1 ) {
        d = dipsw_get();
        PADR = ad;
    }
}
中略
unsigned char dipsw_get( void )
{
    unsigned char sw;

    sw  = ~P6DR;                         /* ディップスイッチ読み込み      */
    sw &= 0x0f;

    return  sw;
}
```

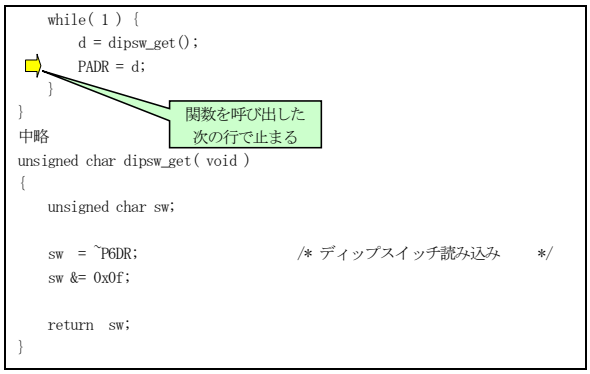
関数は関係なく、次の行に移動する

(7) ステップアウト

ここから

ステップアウトボタン  は、ステップインで関数内に矢印が進んだときにこのボタンを押すと、現在矢印のある関数を最後まで実行して、関数を呼び出した次の行で停止するボタンです。



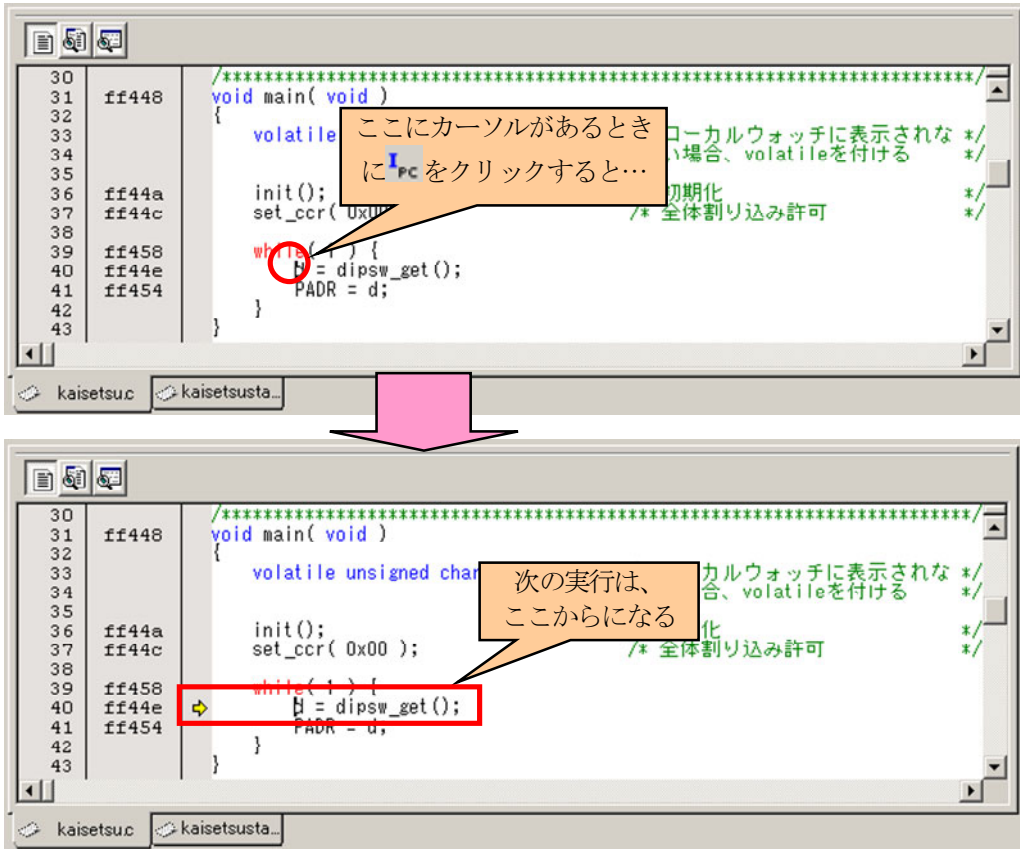


(8) 停止

停止ボタン  をクリックすると、現在実行しているプログラムを停止します。**ただし、シリアルモニタの場合は、このボタンは使えません。シリアルモニタの場合は、マイコンボードのリセットスイッチを押してください**。このボタンは、フルスペックエミュレータなどで使います。

(9) カーソル位置にPC設定

カーソル位置に PC 設定ボタン  をクリックすると、現在カーソルのある位置のアドレスが PC(プログラムカウンタ)の値になり黄色い矢印が付きます。ステップインなどを実行するときは、この行から実行することになります。

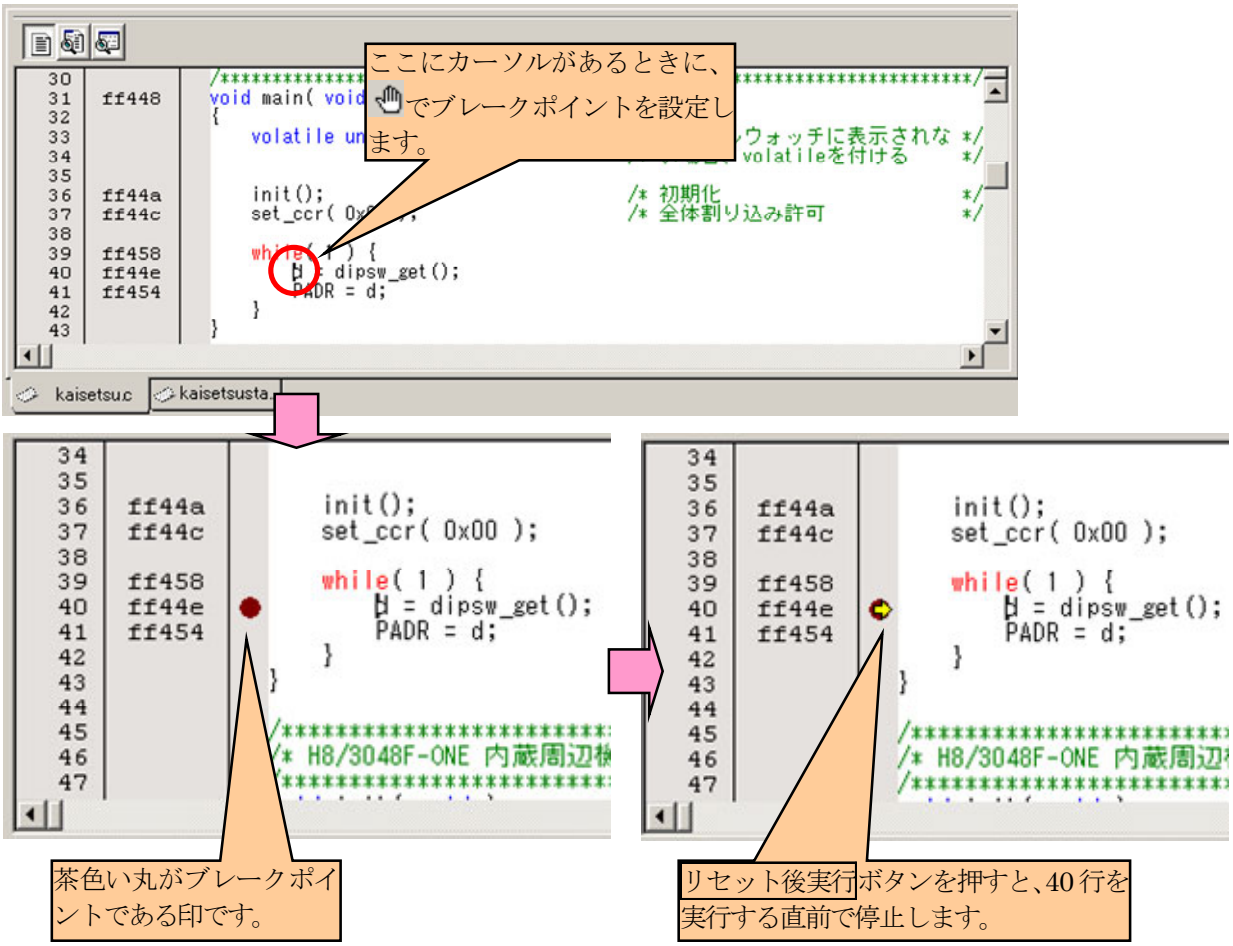


(10) PC位置の表示

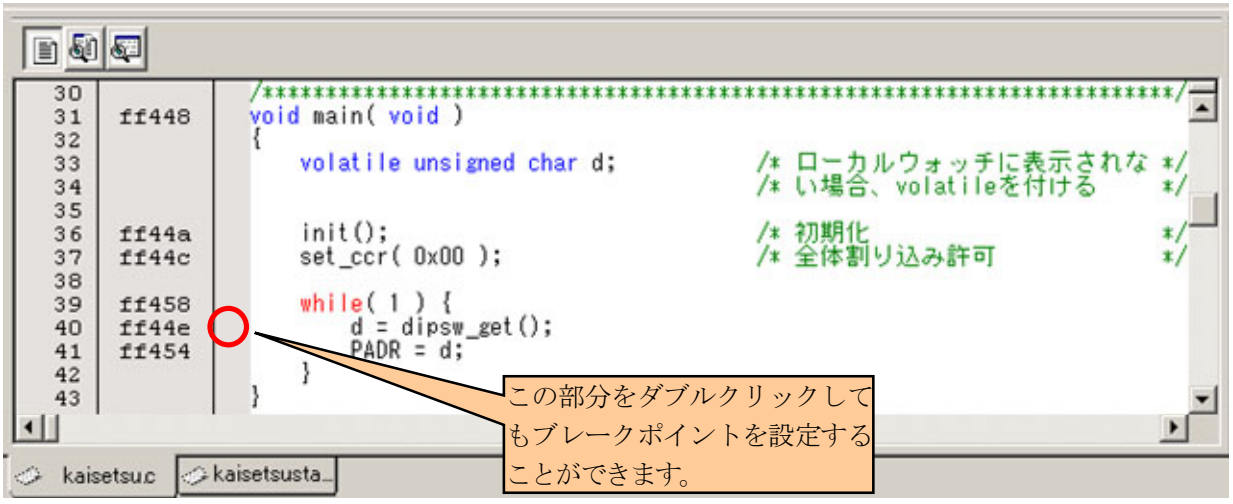
PC 位置の表示ボタン  をクリックすると、現在実行している行(黄色い矢印のある行)にカーソルがジャンプします。現在実行している行を知ることができます。

(11) ブレークポイントの設定/解除

作成したプログラムをデバッグするとき、「この行を実行したときにプログラムを一時停止させて、変数やレジスタの値を確認したい！」という場面がでてきます。このとき、ブレークポイントという「一時停止」マークを設定することにより、プログラムを一時停止させることができます。ブレークポイントの設定/解除ボタンは、一時停止位置を設定、解除するボタンです。

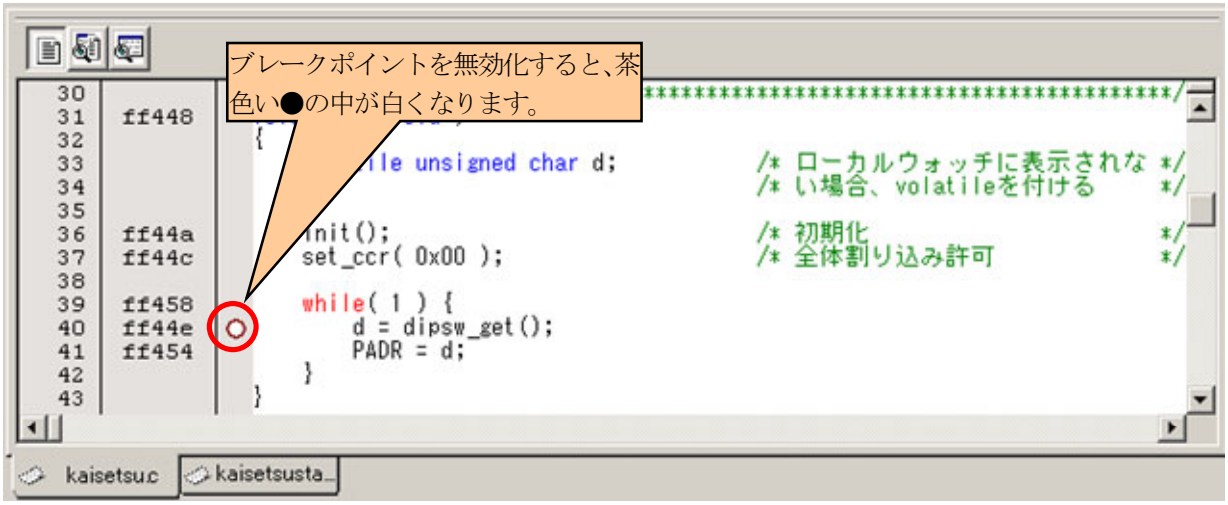


ブレークポイントの設定は、下画面のように、○部分をダブルクリックしても設定することができます。



(12) ブレークポイントの有効化/無効化

ブレークポイントの有効化/無効化ボタン  は、一時的にブレークポイントを無効にしたい場合に使用します。もう一度押すと、無効にしたブレークポイントを有効にすることができます。



3.3 ユーザプログラムのツールチェインの設定

プログラムには、表 3. 1のようなセクションがあります。セクションとは、プログラムのまとまりのことです。例えば「const型修飾子のある変数」のすべてをプログラムから抜き出して、セクションCという名称を付けてグループ化しています。

表 3. 1 セクション一覧

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、値の変更はできません。
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

例えば、ワークスペース「section」、プロジェクト「section」の「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 */

本マニュアルでは、今までのワークスペースとの互換を考えて、それぞれのセクションを 図 3. 2 各セクションの配置場所のように配置しています。

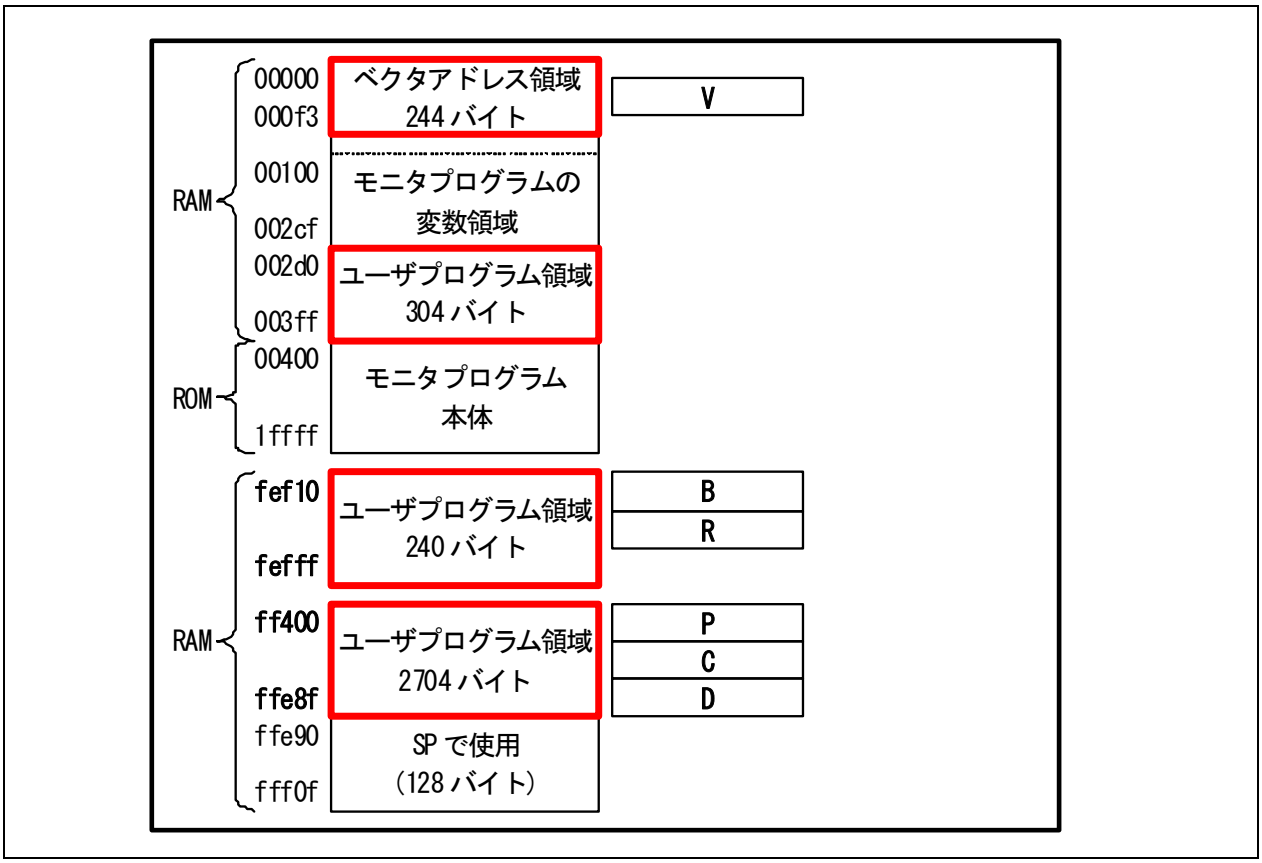
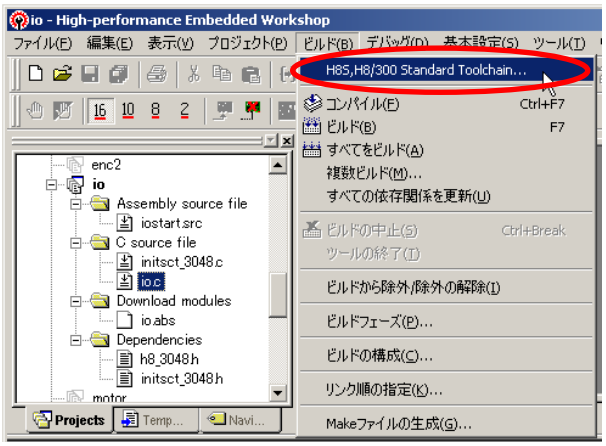
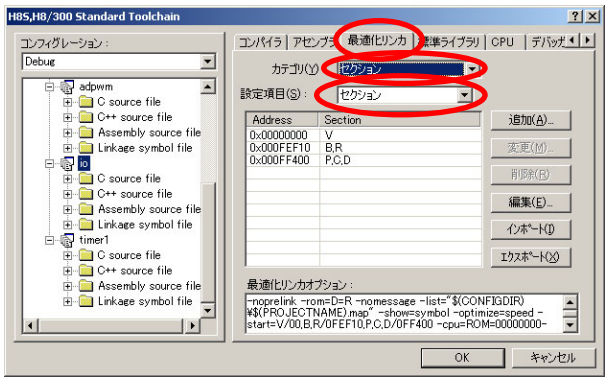


図 3. 2 各セクションの配置場所

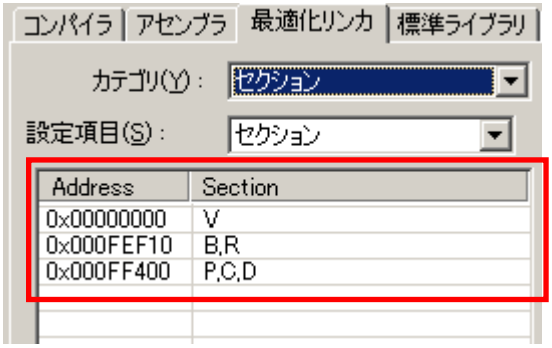
セクションを何番地に配置するかの設定は、ルネサス統合開発環境のツールチェインで行います。下記にその手順を示します。

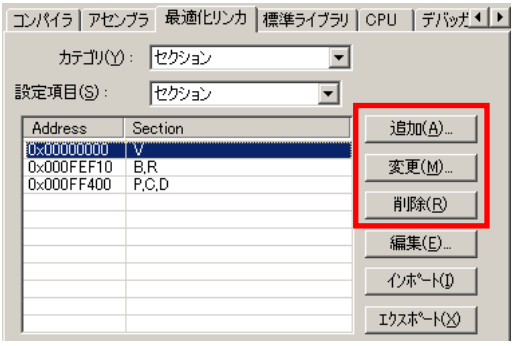


1.「ビルド→H8S,H8/300 Standard Toolchain」(ツールチェイン)を選択します。



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**」を選択します。



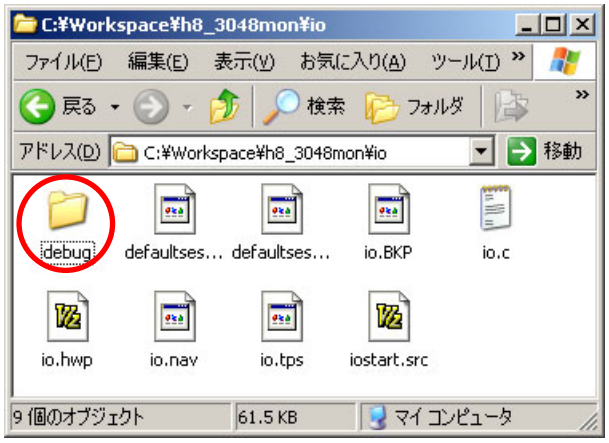


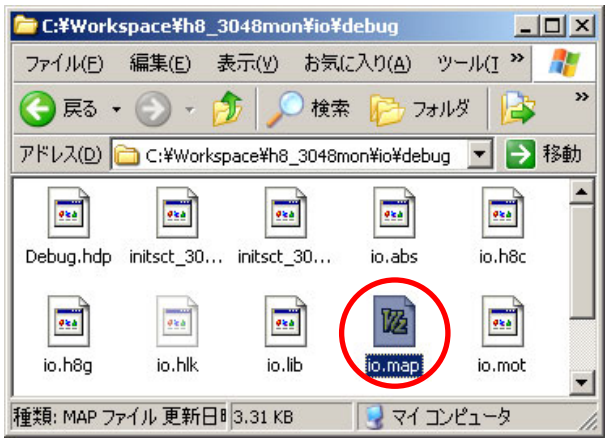
3.□部分が、現在設定されているセクションとアドレスです。

4.編集したい場合は、追加、変更、削除のそれぞれのボタンで編集してください。

3.4 セクションの容量の確認

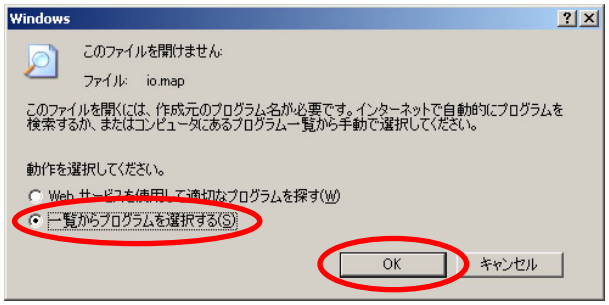
ツールチェインでは、セクションを定義して何番地に配置するか設定することができます。ただし、ツールチェインの設定画面では、セクションの容量は分かりません。ここでは、セクションの容量の確認方法を説明します。例として、プロジェクト「io」で説明します。

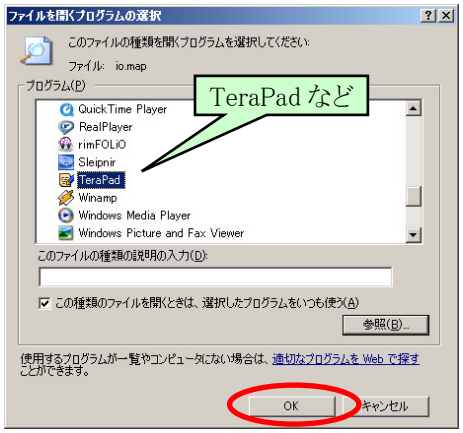




1.エクスプローラなどで、プロジェクトのあるフォルダを開きます。例えば、
「C ドライブ→Workspace→h8_3048mon→io」です。このフォルダ内にある「debug」フォルダを開きます。

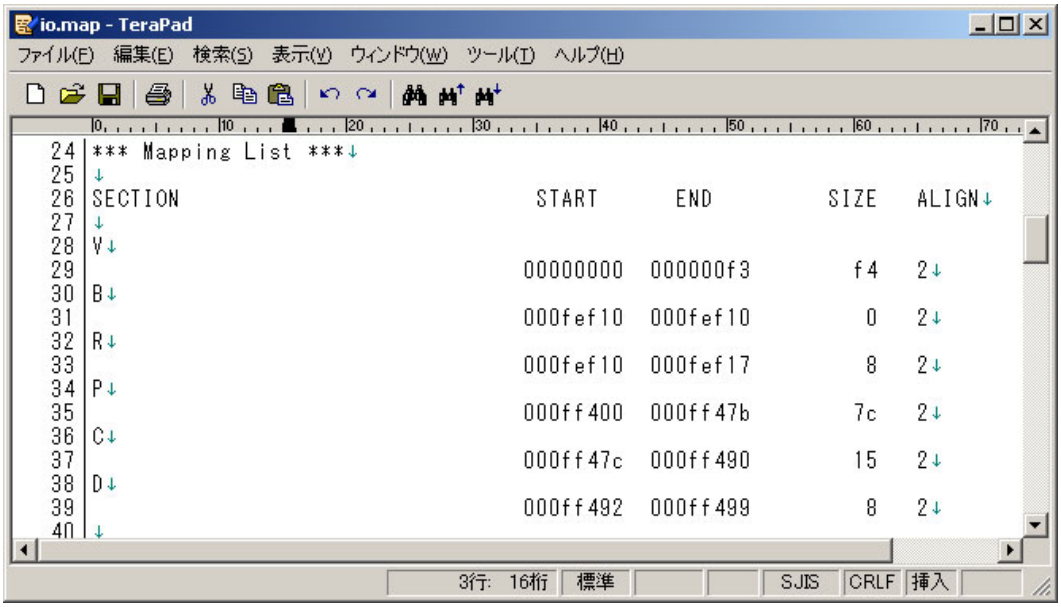
2.「io.map」ファイルをダブルクリックして開きます。





3.もし、「このファイルを開けません」メッセージが表示された場合は、「一覧からプログラムを選択する」のチェックを付けて、OKをクリックします。

4.「NotePad(メモ帳)」や「TeraPad」など、使い慣れたエディタを選び、OKをクリックします。



5.「***** Mapping List *****」と記述されている部分があります。この部分にセクションの開始アドレスと、終了アドレスが記載されています。各セクションの配置場所を 図 3. 3に示します。必ず、

RAM の配置先アドレス ≧ map ファイルに記載さているアドレス

にしなければいけません。今回は、すべてのセクションが範囲内になっていますので OK です。もし範囲外なら、プログラムや変数が大きすぎるので、小さくするかセクションの構成を変更します。

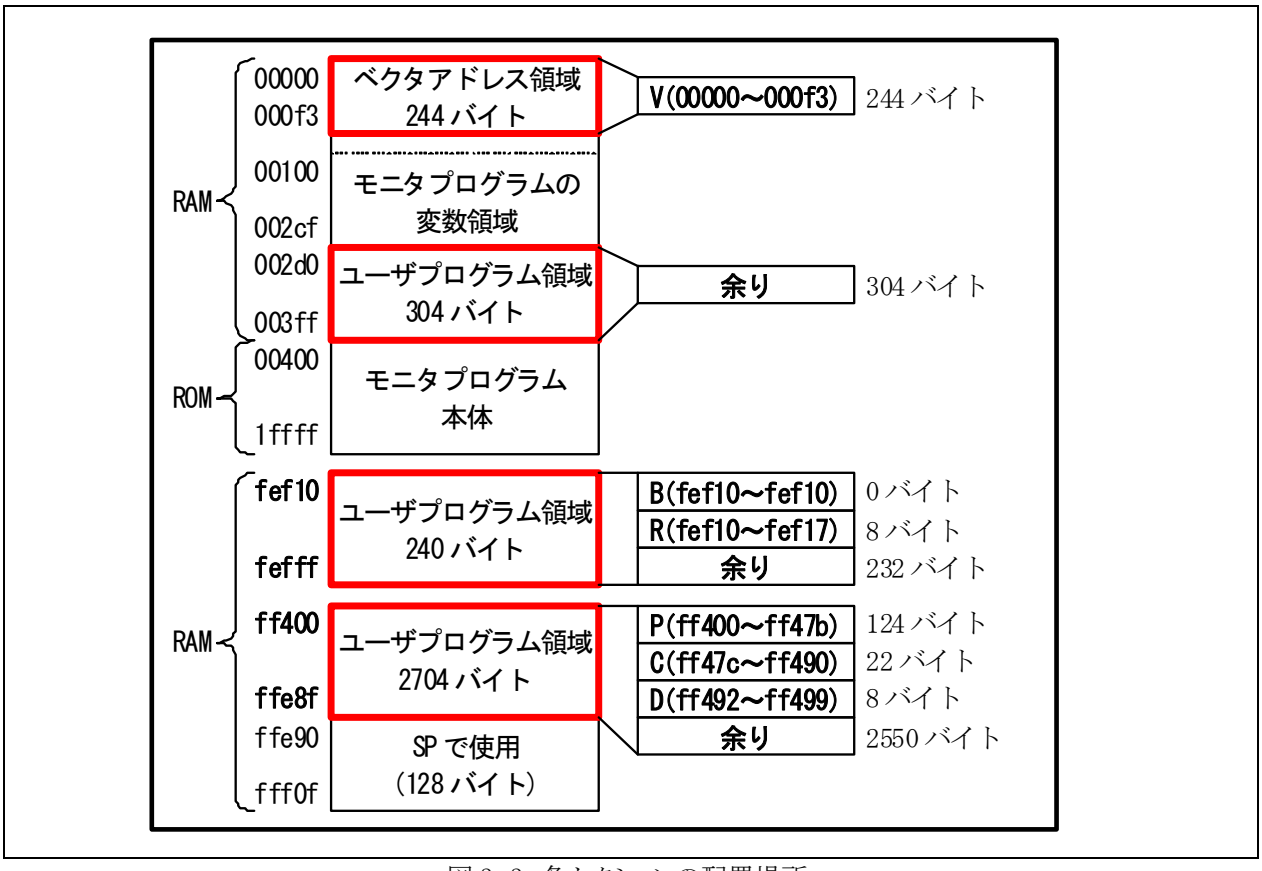


図 3. 3　各セクションの配置場所

3.5　セクションの変更

セクションには、位置を変更できるセクションとできないセクションがあります。ユーザプログラムで使えるRAMアドレスと配置できるセクションの関係を 図 3. 4に示します。

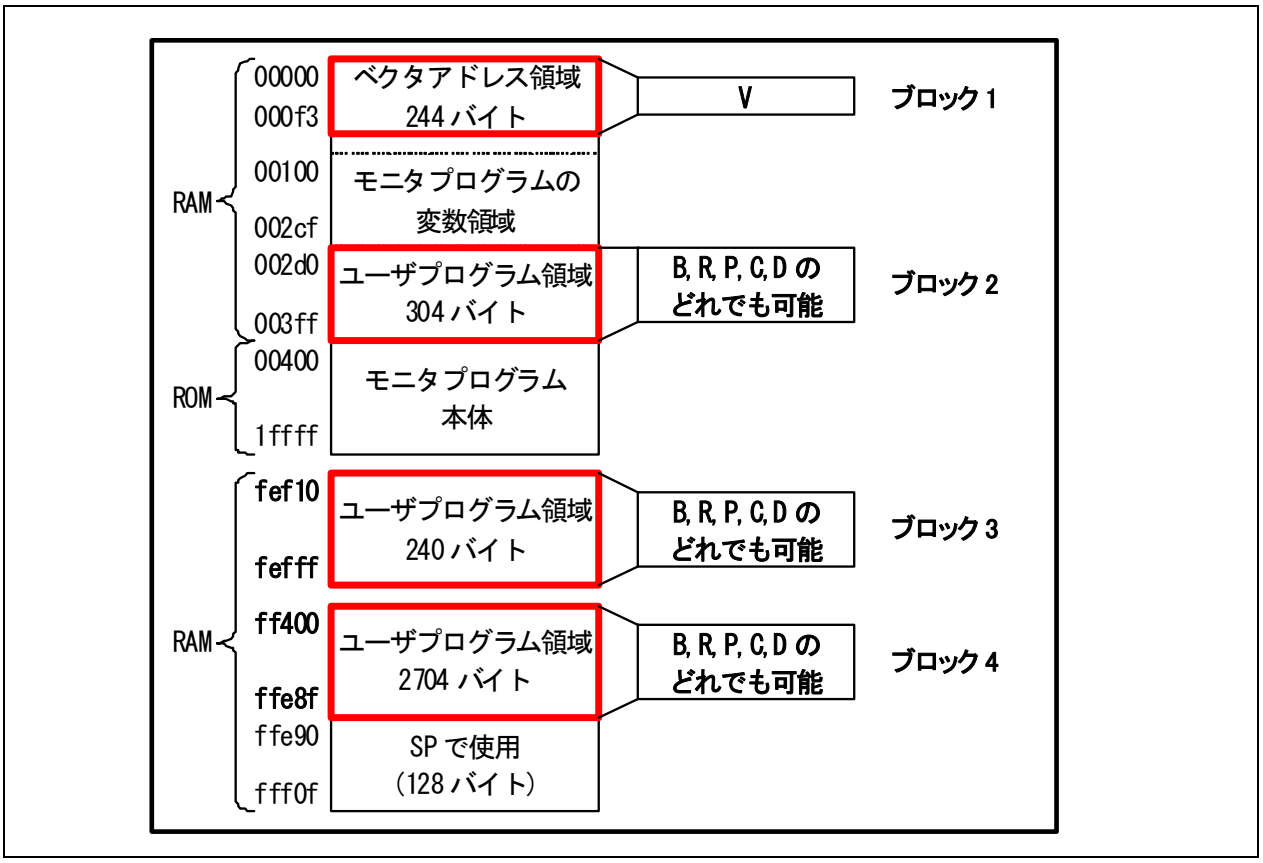
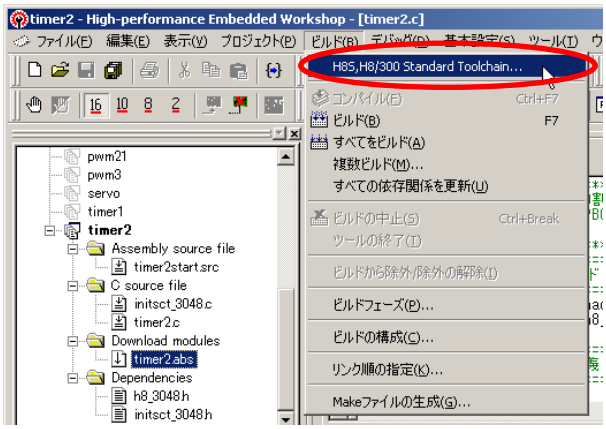


図 3. 4　ユーザプログラムで使える RAM アドレスと配置できるセクションの関係

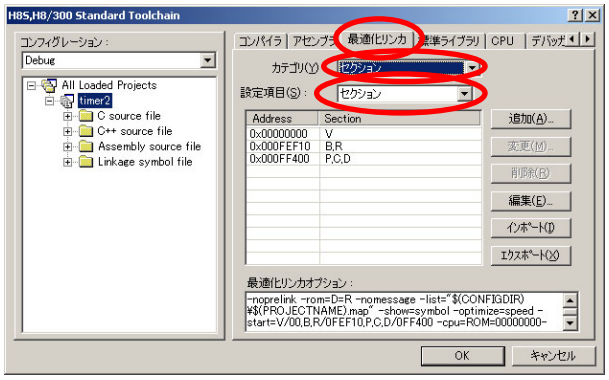
ブロック 1 には、セクション V を配置しなければいけません。ブロック 2～4 は、セクション V 以外のどれを配置しても構いません。

今回は例として、プロジェクト「timer2」のセクションを下表のように変更してみます。

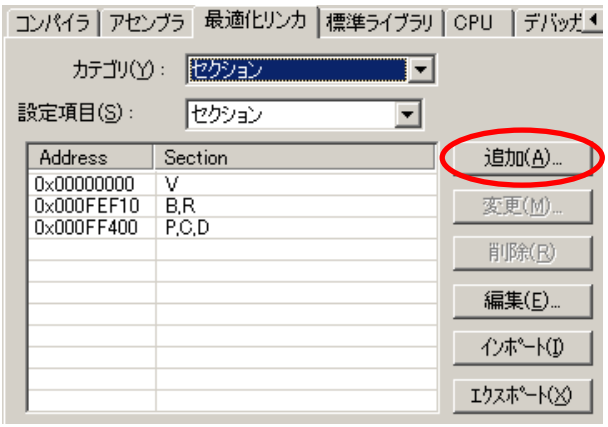
ブロック	変更前	変更後
1	V	V
2	なし	B,R,D
3	B,R	C
4	P,C,D	P



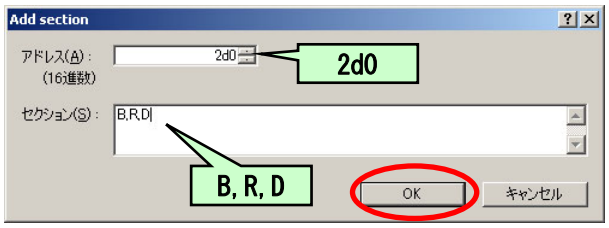
1.「ビルド→H8S,H8/300 Standard Toolchain」(ツールチェイン)を選択します。



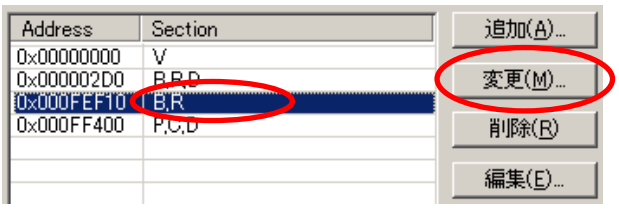
2.「最適化リンカ」を選択します。「カテゴリ:**セクション**」、「設定項目:**セクション**」を選択します。



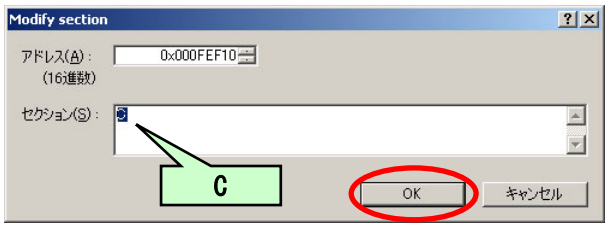
3.0x02d0 番地がないので、登録します。



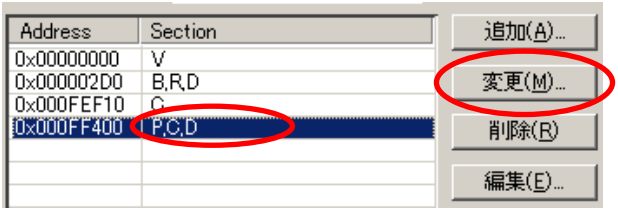
4.「アドレス:**2d0**」、「セクション:**B, R, D**」を入力し、OK をクリックします。



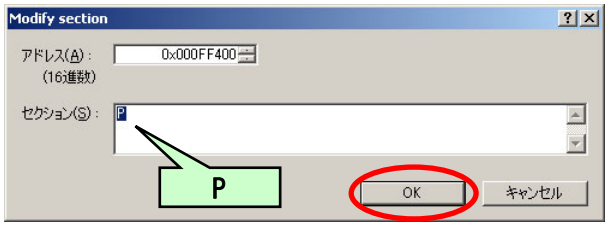
5.「B,R」の行を選択、変更をクリックします。



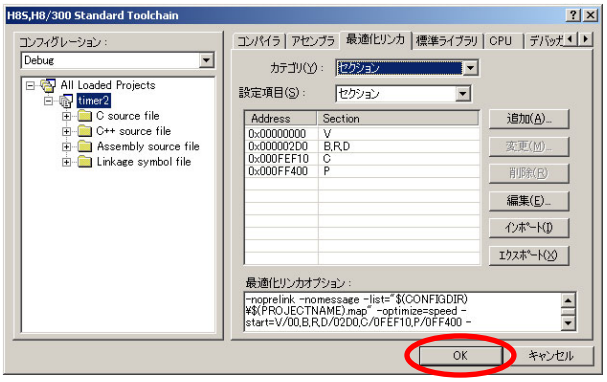
6.「セクション:**C**」に変更して、OK をクリックします。



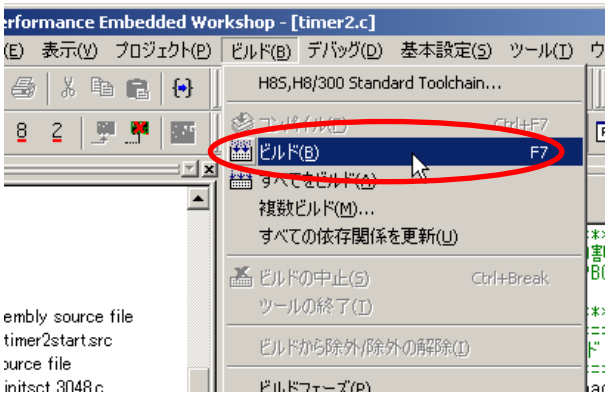
7.「P,C,D」の行を選択、変更をクリックします。



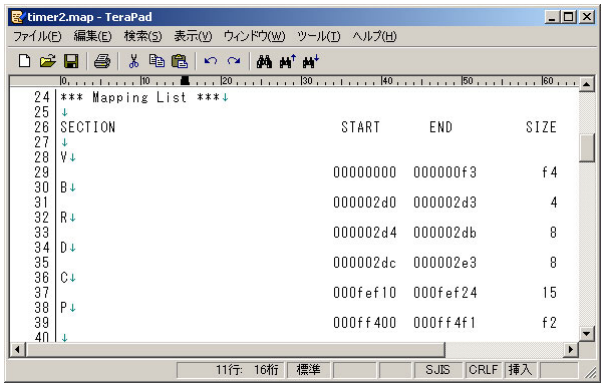
8.「セクション:**P**」に変更して、OK をクリックします。



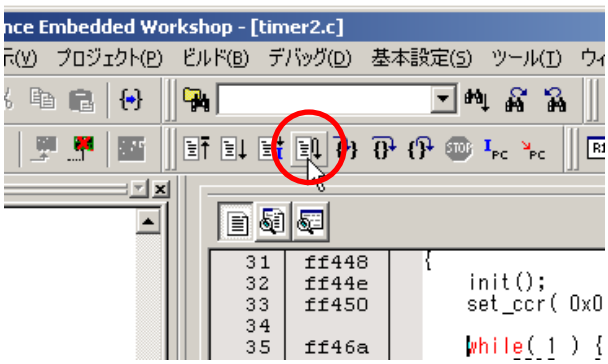
9.セクションの設定は完了です。OKをクリックします。



10.「ビルド→ビルド」でビルドします。エラーがないことを確認してください。



11.「C:¥Workspace¥h8_3048mon¥timer2¥Debug」フォルダの「timer2.map」を開いて、セクションのアドレスを確認してください。



12.リセット後実行ボタンで実行、動作を確認してください。

4. オリジナルプログラムをモニタで使用できるように改造する

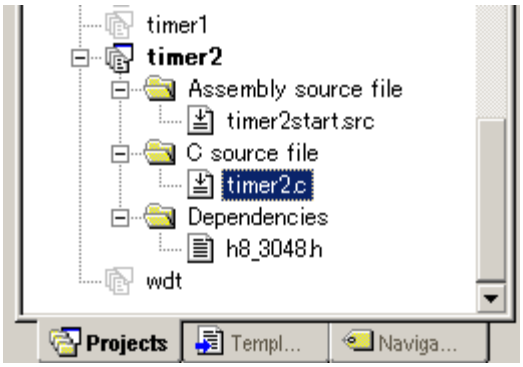
ここでは、今あるプログラムを、シリアルモニタで使用できるように改造します。
シリアルモニタの仕様で次のような場合は、シリアルモニタ用に改造することはできません。改造する前に確認してください。

・printf 関数、scanf 関数を使っている
・SCI1(シリアルコミュニケーションインターフェースのチャネル 1)を使っている
・外部割り込み NMI を使っている

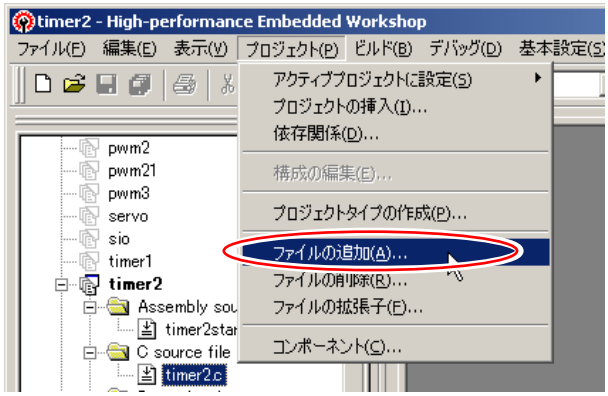
もし、これらを使用しないように修正できる場合は、シリアルモニタ用に改造可能です。

4.1 「car_printf2.c」を使用していない場合

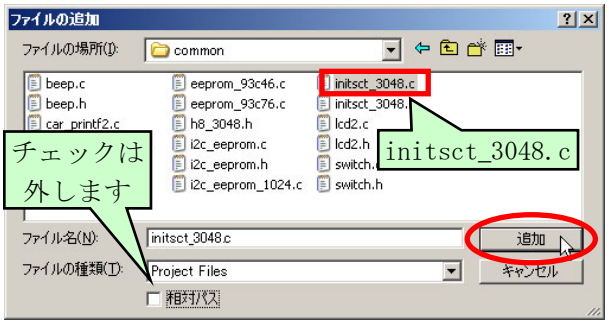
ワークスペース「h8_3048」のプロジェクト「timer2」を例に説明します（今まで説明してきた「h8_3048mon」ではありません）。



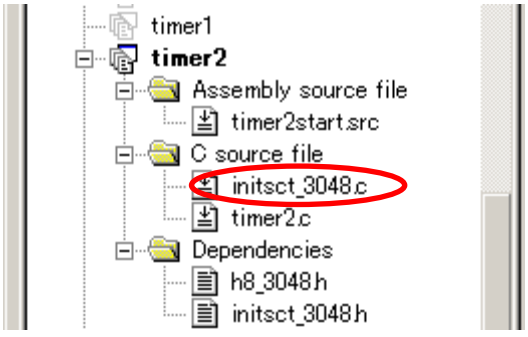
1.「C source file」欄に「initsct_3048.c」が無い場合、追加します。ある場合は 5 番に進んでください。



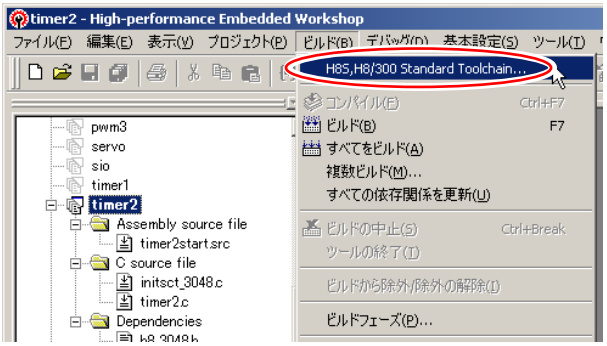
2.「プロジェクト→ファイルの追加」をクリックします。



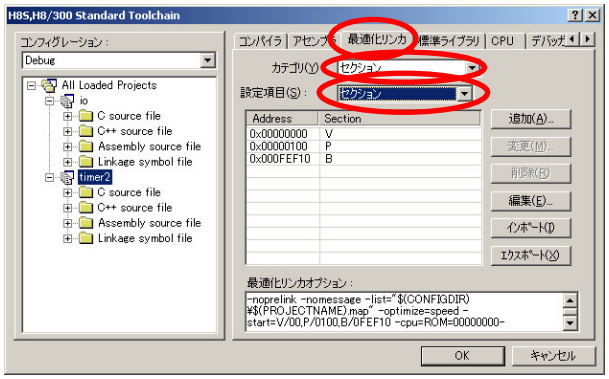
3.「c:¥workspace¥common」フォルダの「initsct_3048.c」を選択します。相対パスのチェックは外します。



4.「initsct_3048.c」が追加されました。



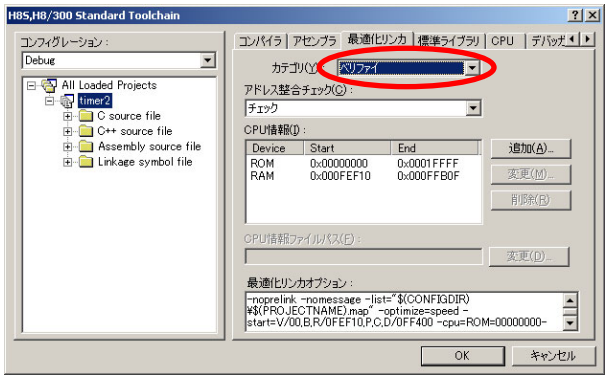
5.「ビルド→H8S,H8/300 Standard Toolchain」(ツールチェイン)を選択します。



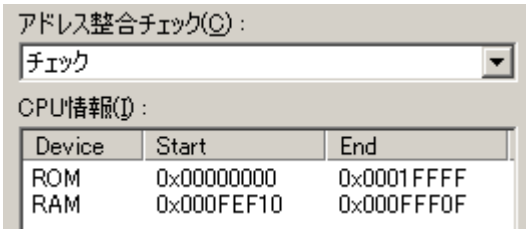
6.「最適化リンカ」を選択します。「カテゴリ:**セクション**」、「設定項目:**セクション**」を選択します。

Address	Section
0x00000000	V
0x000FEF10	B,R
0x000FF400	P,C,D

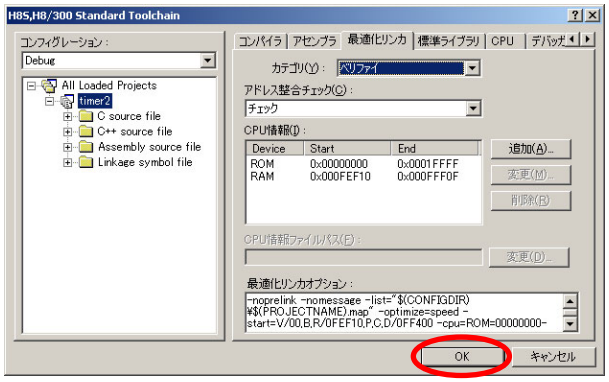
7.画面のようにセクションを設定します。



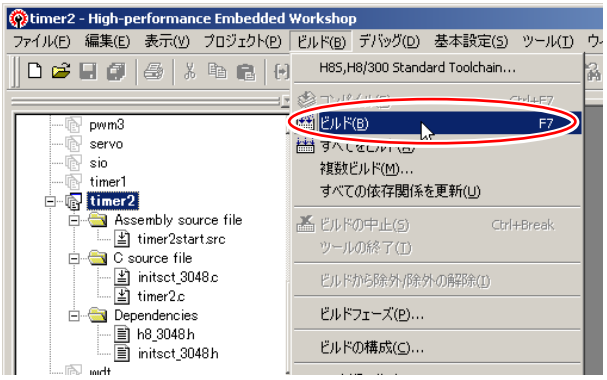
8.「カテゴリ:**ベリファイ**」を選択します。



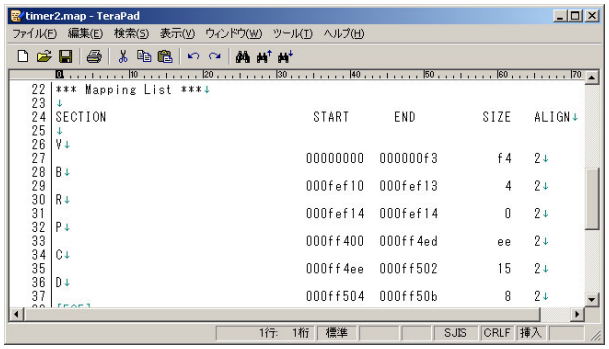
9.画面のように CPU 情報を設定します。



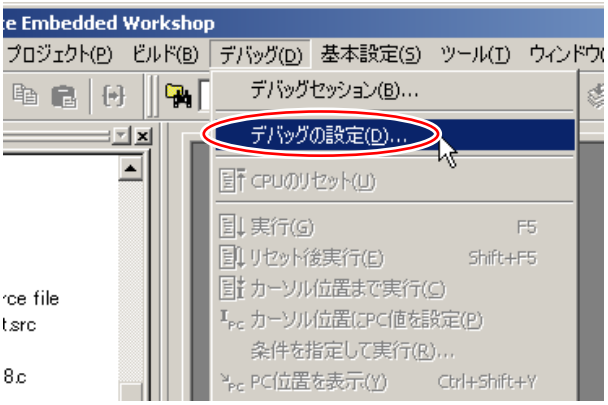
10.OK をクリックしてツールチェインの設定を完了します。



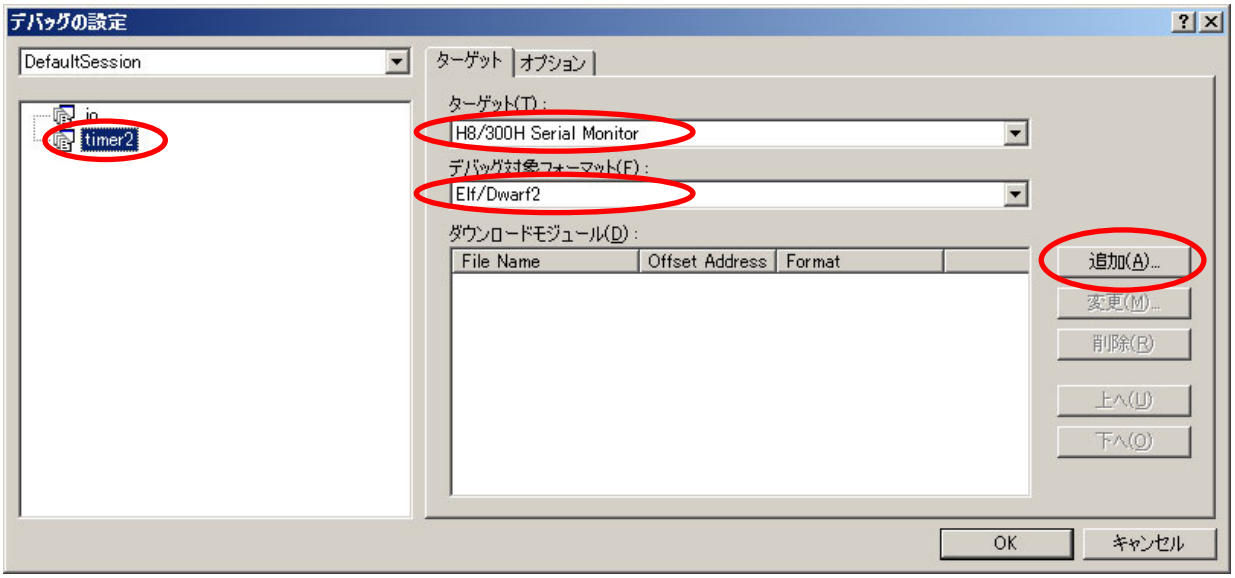
11.「ビルド→ビルド」でビルドします。
「0 Errors, 0 Warnings」か確認します。



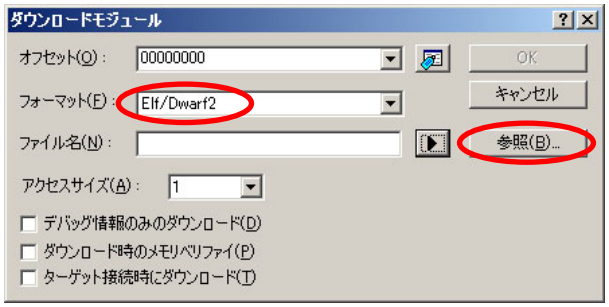
12.「C:¥Workspace¥h8_3048¥timer2¥Debug」フォルダ（または、プロジェクトフォルダ内の Debug フォルダ）を開いて「timer2.map」をエディタで開きます。ユーザプログラムで使える RAM アドレスを超えていないかチェックします。



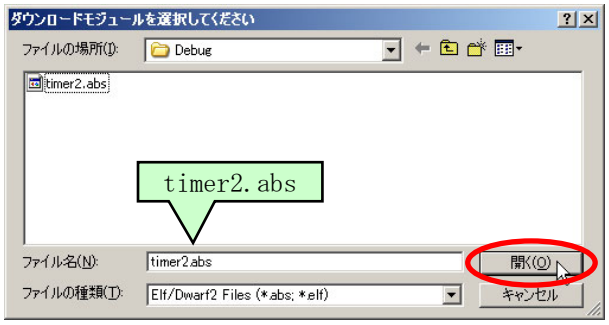
13.「デバッグ→デバッグの設定」をクリックします。



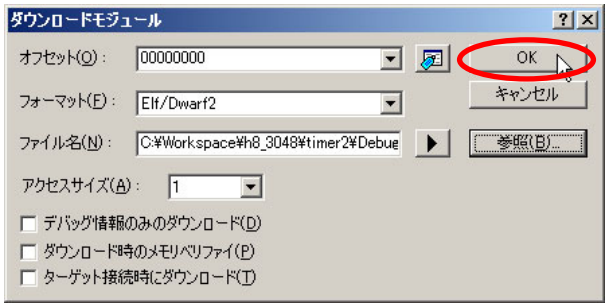
14.左側のプロジェクトを選ぶ欄で「**timer2**」を選択します。「ターゲット:**H8/300H Serial Monitor**」、「デバッグ対象フォーマット:**Elf/Dwarf2**」を選択、追加をクリックします。



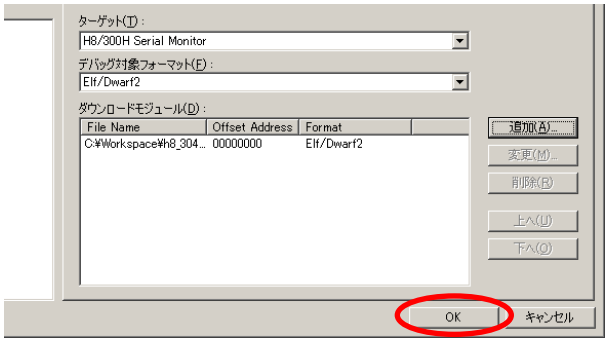
15.「フォーマット:**Elf/Dwarf2**」を選択、参照をクリックします。



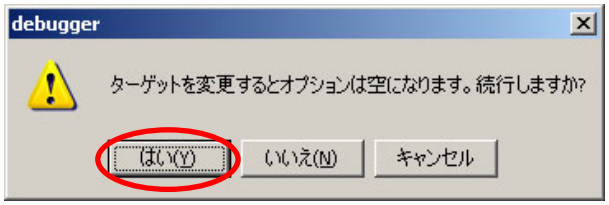
16.「C:¥Workspace¥h8_3048¥timer2¥Debug」フォルダ（または、プロジェクトフォルダ内の Debug フォルダ）を開いて「timer2.abs」を選択、開くをクリックします。



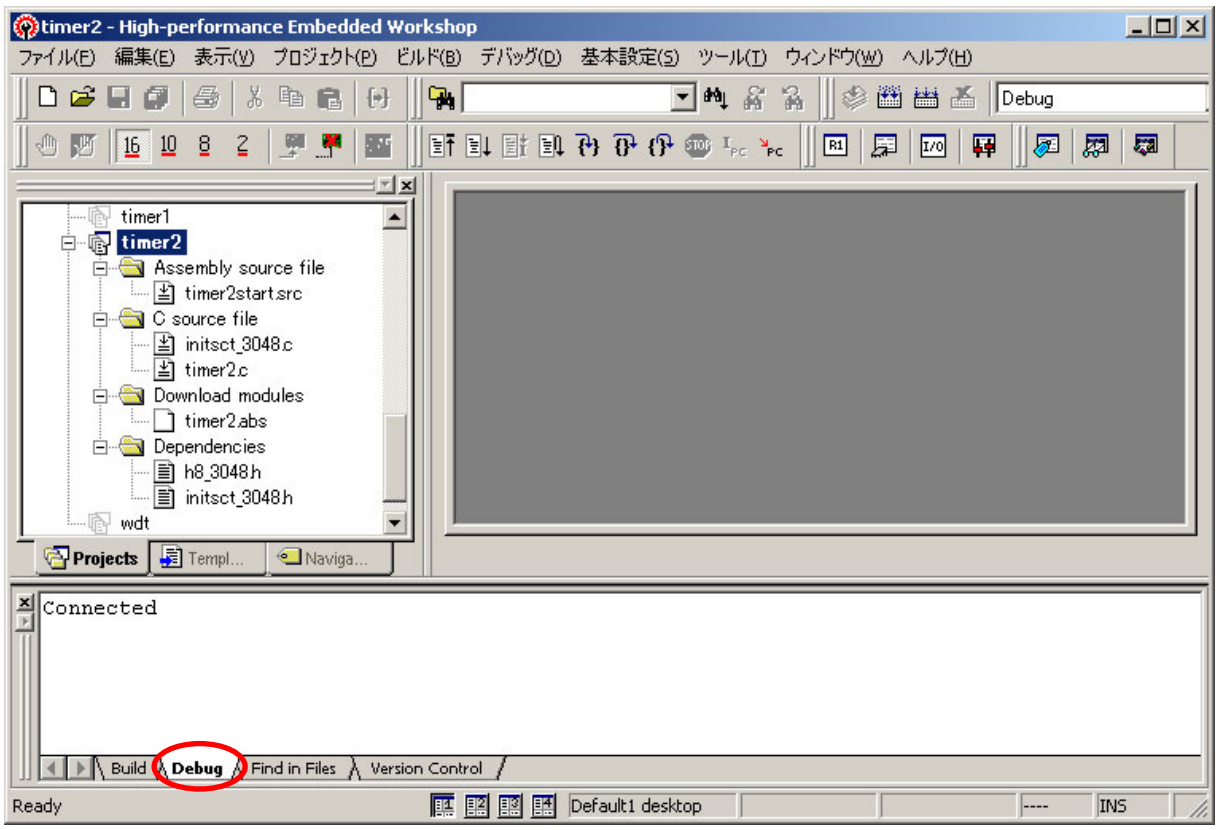
17. OK をクリックします。



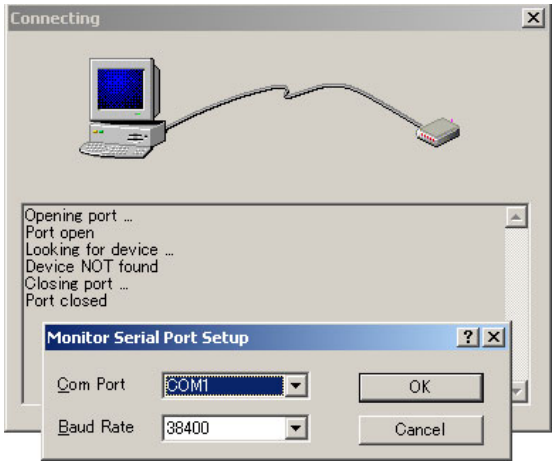
18. OK をクリックします。



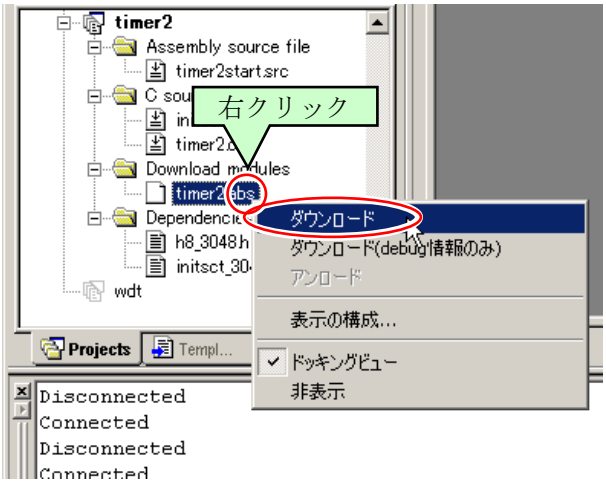
19. はい をクリックします。



20.「Debug」欄が「Connected」になり、シリアルモニタと接続しました。



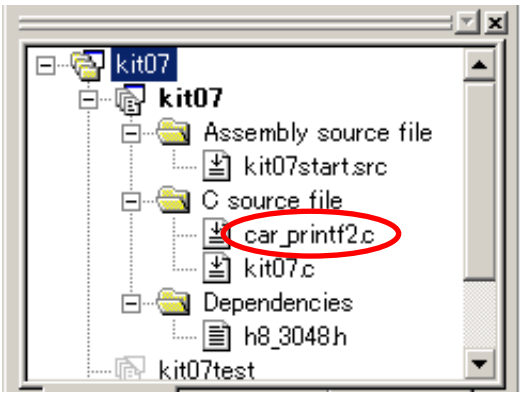
21.このような画面が出てきた場合、シリアルモニタと接続できていません。マイコンボードにモニタプログラムを書き込んでいるか、RS-232C ケーブルが接続されているかなど、確認してください。



22.「timer2.abs」上で右クリックして、「ダウンロード」をクリックします。プログラムがダウンロード（RAM に転送）されました。リセット後実行ボタンで実行し、プログラムが動くか試してみましょう!!

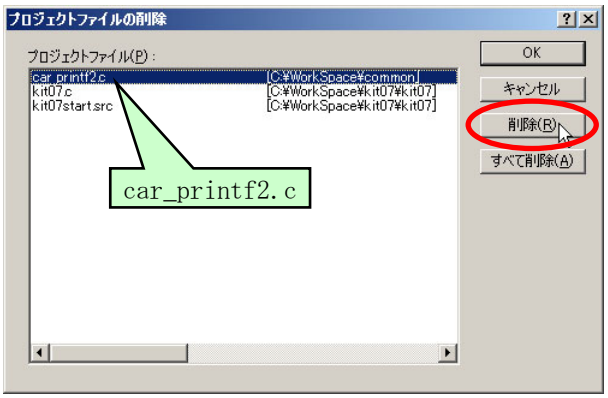
4.2 「car_printf2.c」を使用している場合

ワークスペース「kit07」のプロジェクト「kit07」を例に説明します。ちなみにプロジェクト「sioservo」と「sioservo2」はSCI1 を使用しているため、シリアルモニタで使うことはできません。

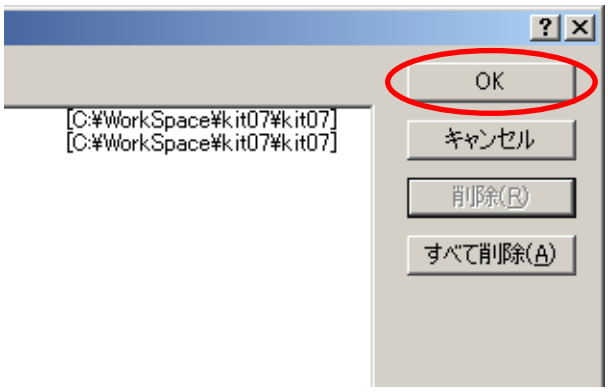


1.「C source file」欄に「car_printf2.c」がある場合、削除します。

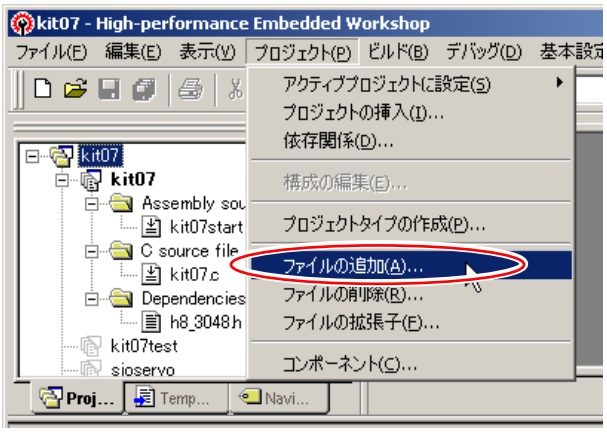
2.「プロジェクト→ファイルの削除」をクリックします。



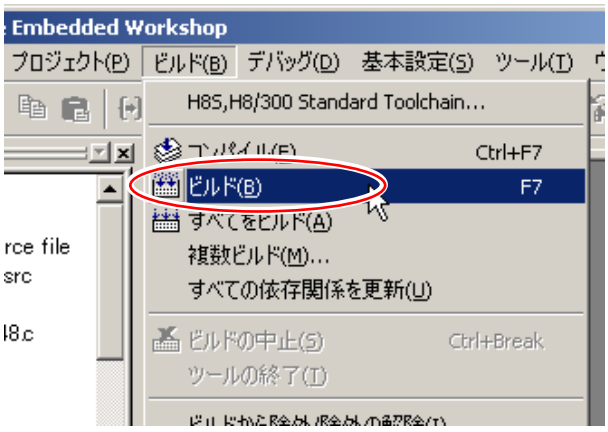
3.「car_printf2.c」を選択、削除をクリックします。



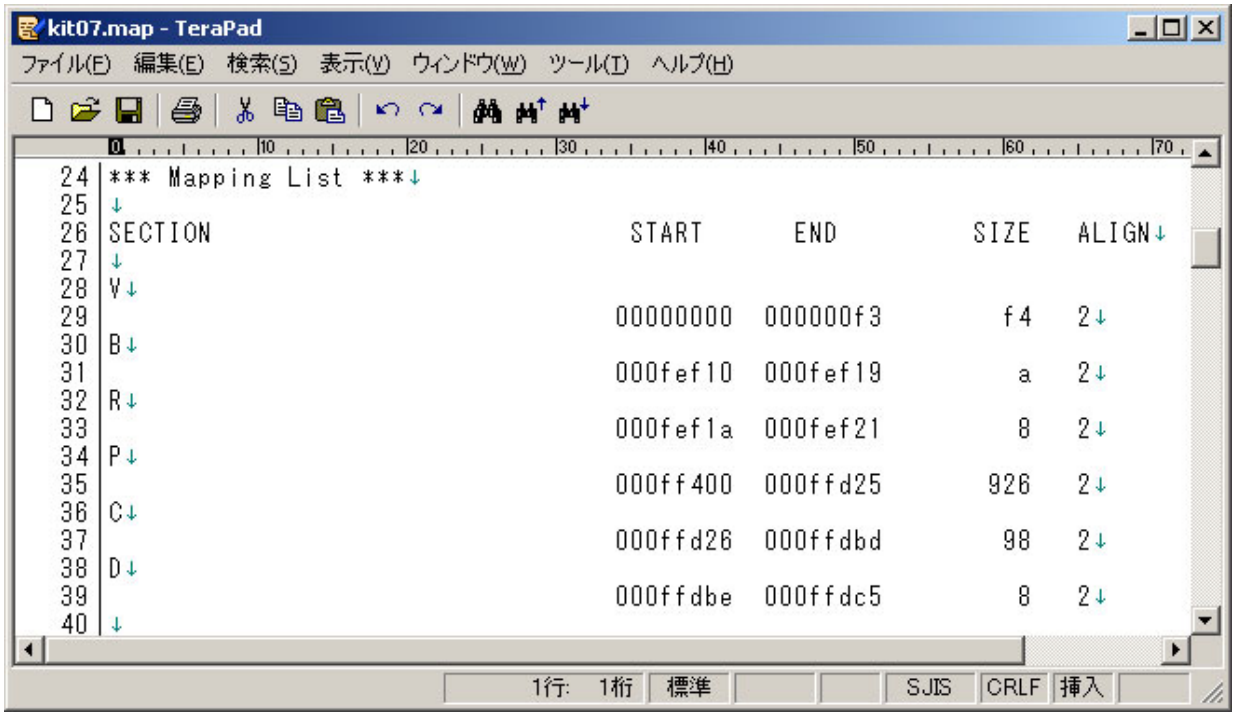
4.OK をクリックします。**プログラムで、printf 文、scanf 文を使用している場合は、削除します。**



5.続きは「**4.1 「car_printf2.c」を使用していない場合**」の「initsct_3048.c」の追加から続けてください。



6.「ビルド→ビルド」でビルドします。



7.「C:¥Workspace¥kit07¥kit07¥debug」フォルダ（または、プロジェクトフォルダ内のDebugフォルダ）を開いて「kit07.map」をエディタで開きます。図 3.5にアドレスをまとめます。

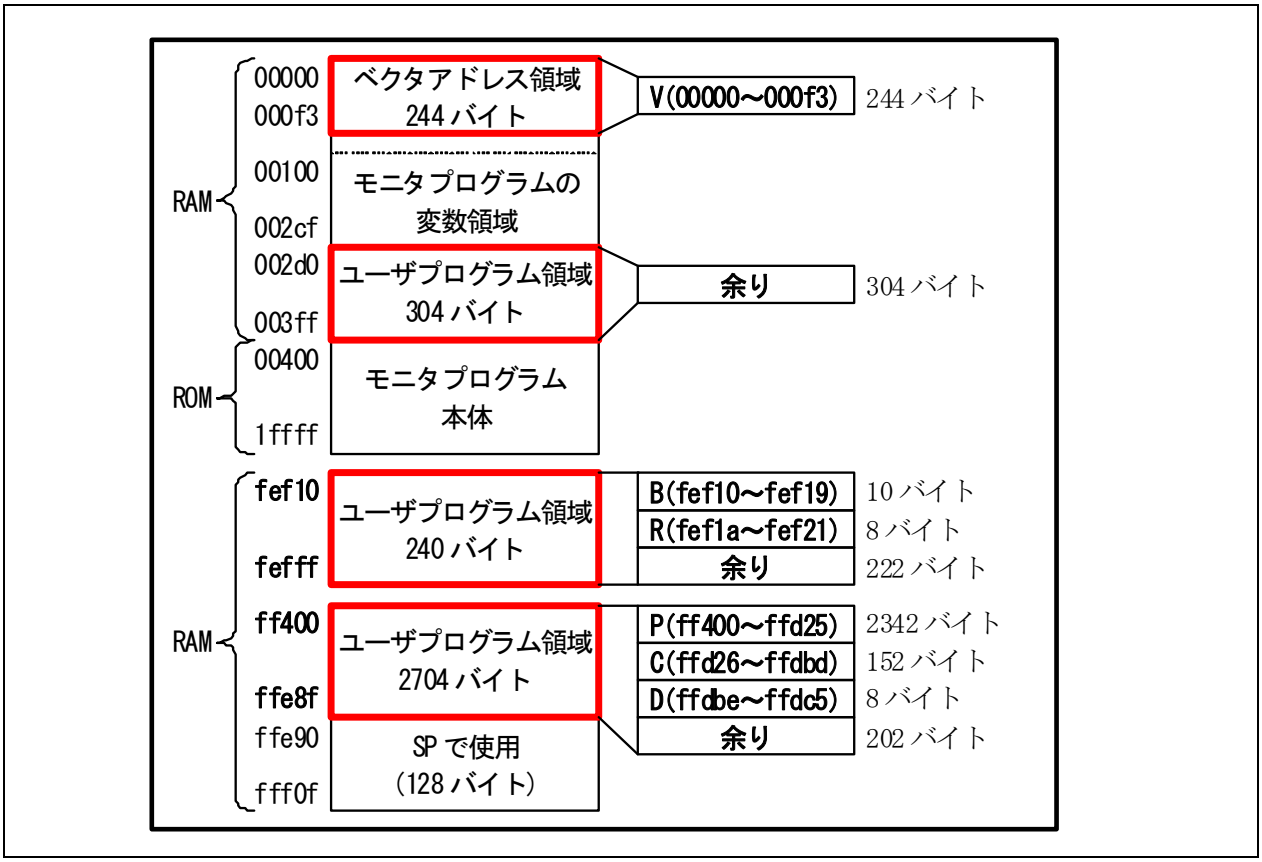


図 3.5　プロジェクト「kit07」のセクション

セクションP（プログラムのセクション）があるRAM領域は、余りが 202 バイトしかありません。プログラムを追加していくと容量を超えてしまうかもしれません。そこで、セクションC,Dを移動させて、セクションPの領域を増やしてみましょう。今回、図 3. 6のように考えました。

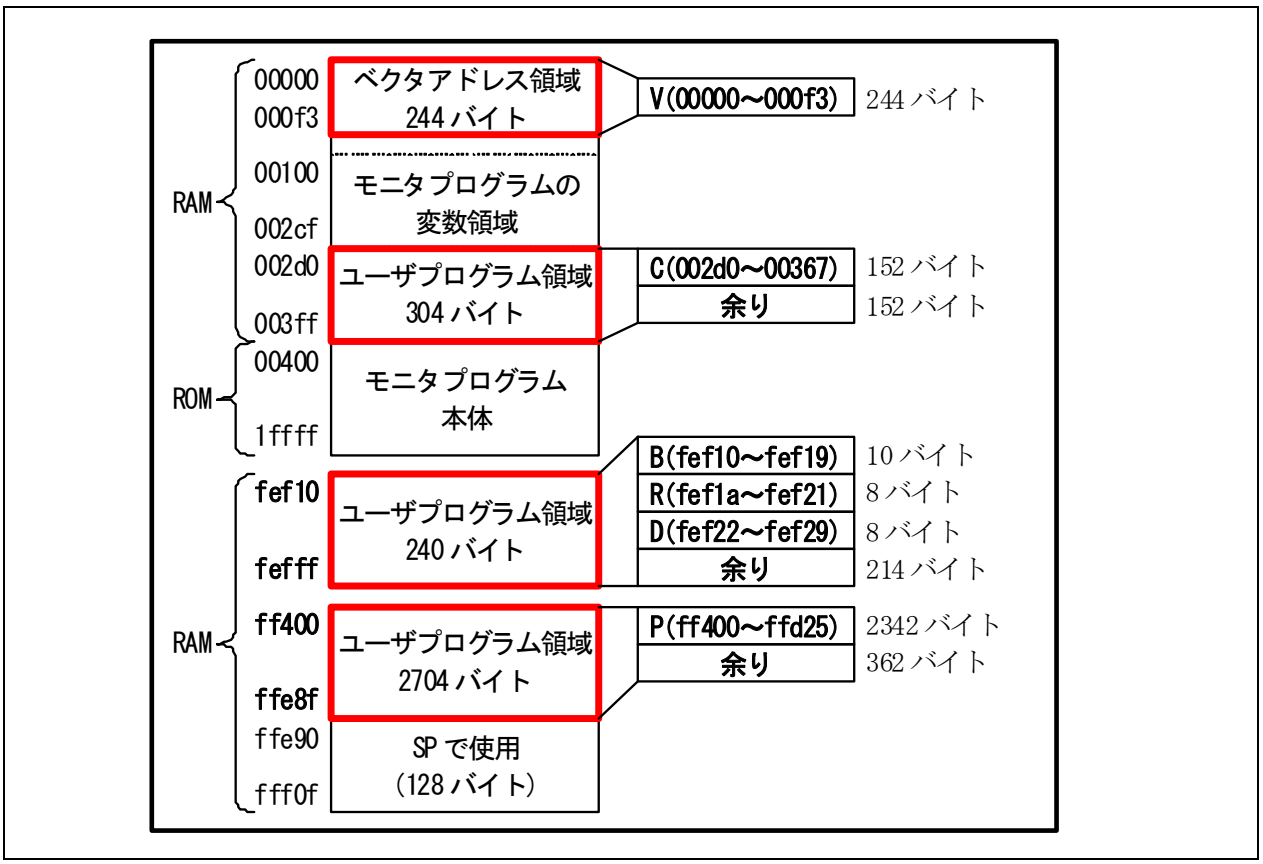
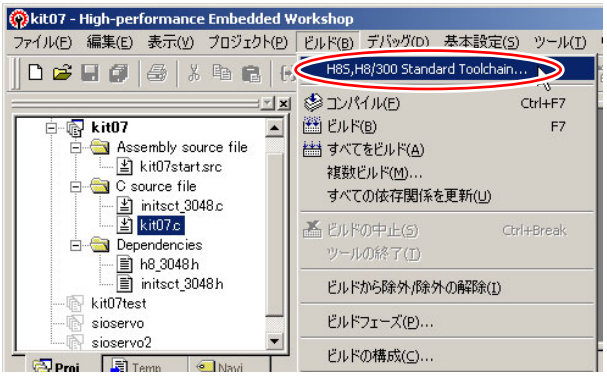
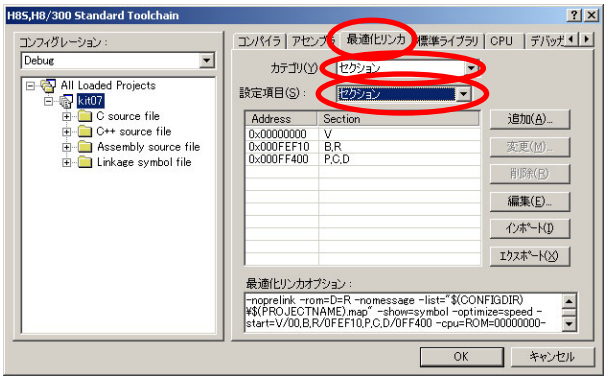


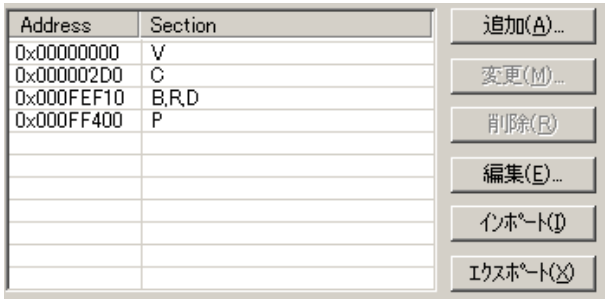
図 3. 6　プロジェクト「kit07」の変更後のセクション

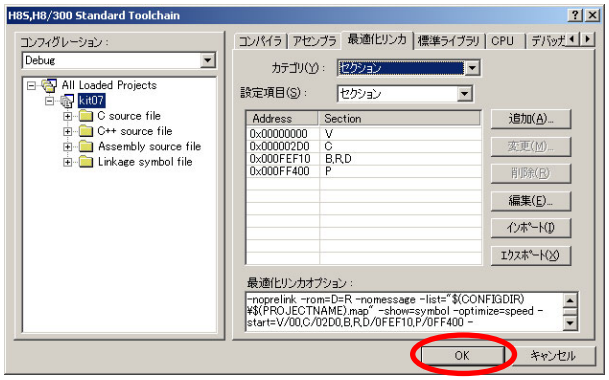


1.「ビルド→H8S,H8/300 Standard Toolchain」（ツールチェイン）を選択します。



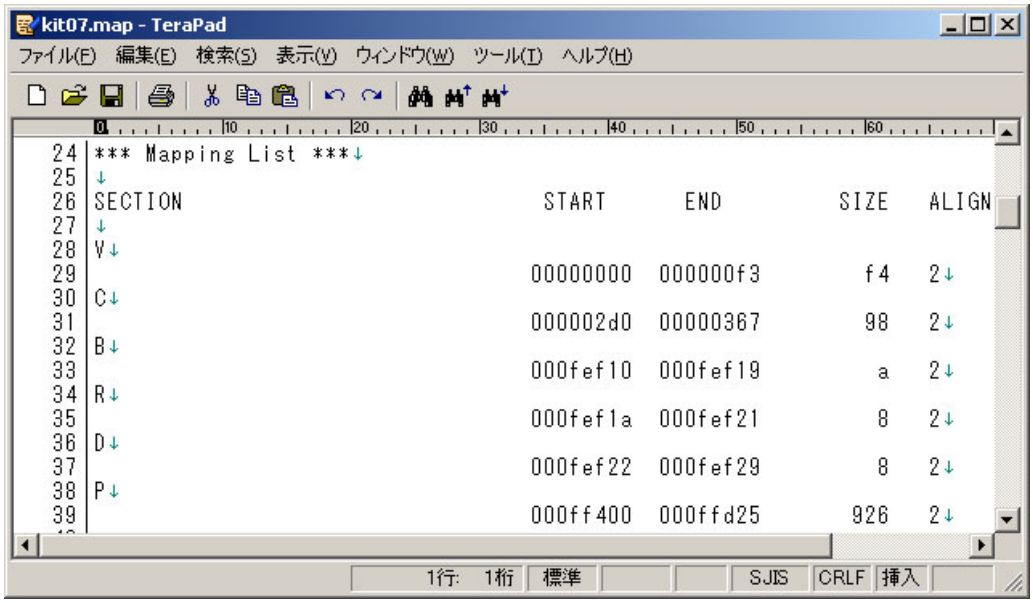
2.「最適化リンカ」を選択します。「カテゴリ：**セクション**」、「設定項目：**セクション**」を選択します。





3.上画面のようにセクションを設定します。

4.OK をクリックして完了です。「ビルド→ビルド」でビルドします。



5.「C:¥Workspace¥kit07¥kit07¥debug」フォルダ（または、プロジェクトフォルダ内の Debug フォルダ）を開いて「kit07.map」をエディタで開きます。期待したとおりのセクションになりました。

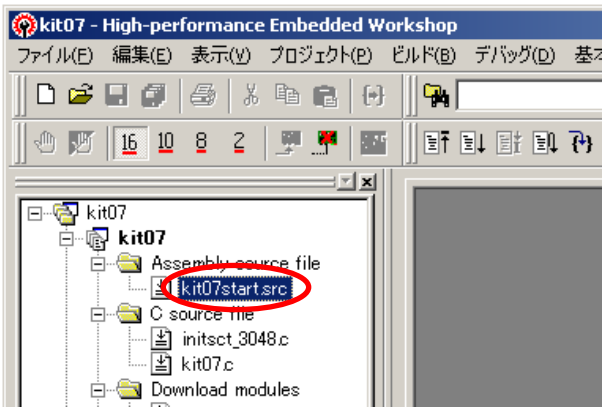
4.3 SP（スタックポインタ）領域の移動

RAM で連続して一番大きいのは、0xff400～0xffe8f 番地の 2704 バイトです。一番大きいのでこのアドレスにはプログラム(セクション P)を配置するのが一般的です。では、プログラムが大きくなって、この領域以上になったらどうすればよいでしょう。SP（スタックポインタ領域）は、シリアルモニタを使っていないプログラムとの互換性を考えて、0xffe90～0xfff0f 番地にしていますが、移動することができます。

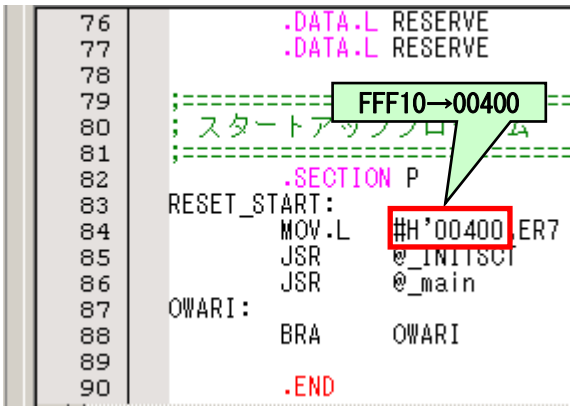
ここでは、SP 領域を 0x002d0～0x003ff 番地に移動させて、セクション P の領域を増やしてみます。

ちなみに、セクションは 0x002d0→0x003ff 番地というように配置しますが、SP は 0x003ff→0x002d0 番地というように逆の進み方をします。そのため、SP の設定は、設定したい領域の最後のアドレス＋1 にします。

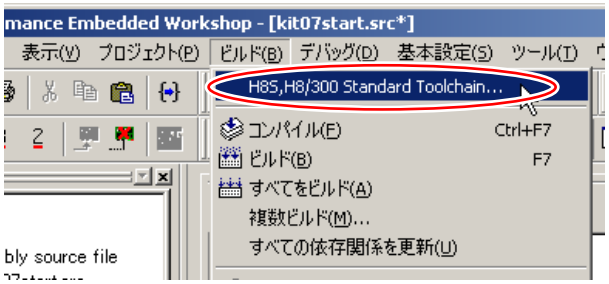
ここでは、「4.2 「car_printf2.c」を使用している場合」の最後の状態から進めていきます（図 3. 6の状態）。0x002d0～0x003ff番地内にSPを設定することにします。



1.「kit07start.src」をダブルクリックして開きます。



2.「MOV.L #H'FFF10,ER7」の「FFF10」を、「**00400**」に変更します。



3.「ビルド→ビルド」でエラーがないことを確認します。

SPの領域が、0xffe90～0xfff0f番地から 0x002d0～0x003ff番地に移動しました(図 4. 1)。セクションPの範囲が、0xff400～0xfff0f番地までの 2832 バイトに増やすことができました。

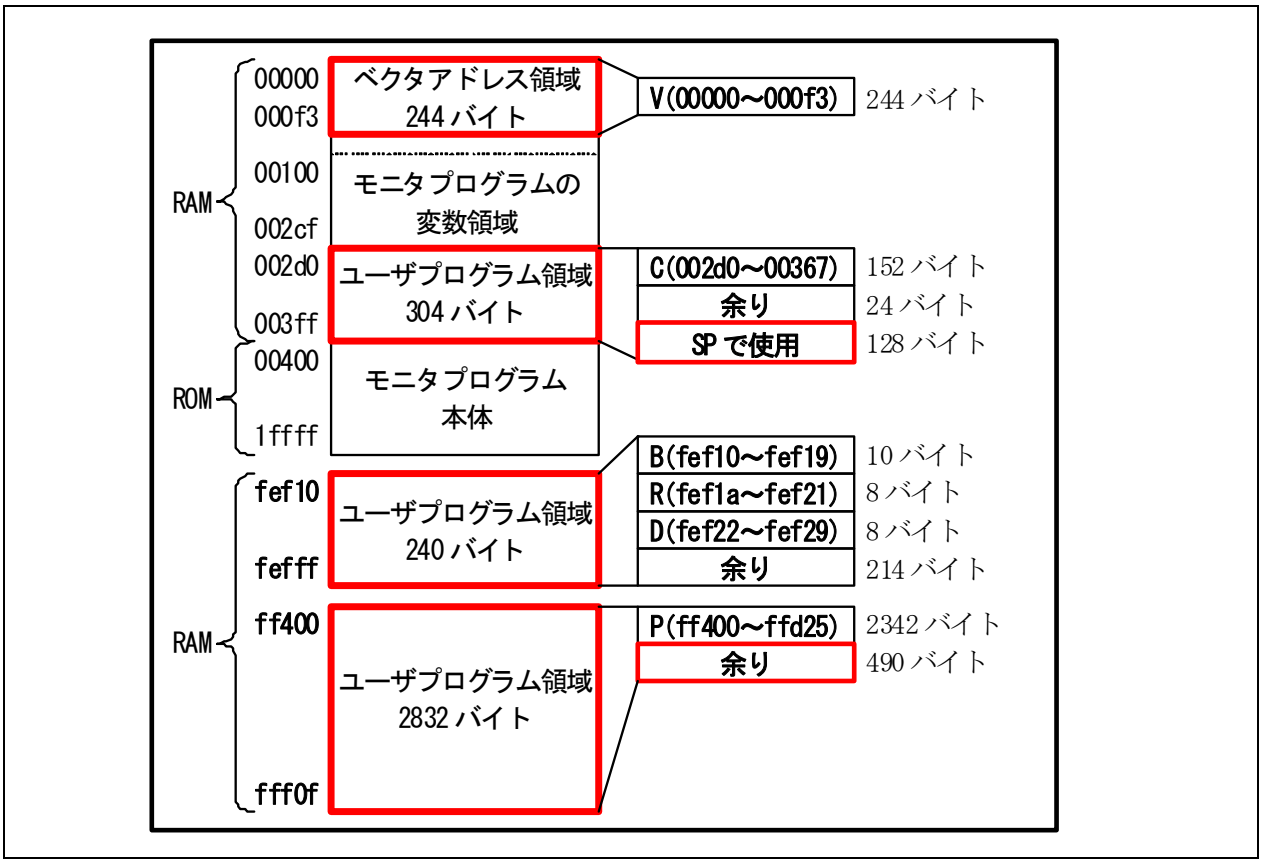


図 4. 1 SP 領域の移動

5. 補足

5.1 シリアルモニタの動作

ここでは、シリアルモニタの電源を入れてからの動作を説明します。

(1) マイコンボードの電源を入れると、ベクタアドレスのベクタ番号 0 番(0x0000 番地)に書いてある 4 つの数値を読みます。この番地には、「00001000」と書かれており、その書かれている番地である 0x01000 番地から実行を開始します(図 5. 1の(1))。
(2) 0x1000 番地にはモニタプログラムがあります。モニタプログラムは、「**RAMによるフラッシュメモリのエミュレーション**」を実行、0xff000～0xff3ff番地のRAMを 0x00000～0x003ff番地に重ねます(図 5. 1の(2))。
(3) モニタプログラムは、ルネサス統合開発環境から通信で送られてきたユーザプログラムのデータをRAMに保存します(図 5. 1の(3))。
(4) モニタプログラムは、ルネサス統合開発環境から通信で送られてきた「実行」指令により、RAMに保存したユーザプログラムへジャンプ、実行します(図 5. 1の(4))。

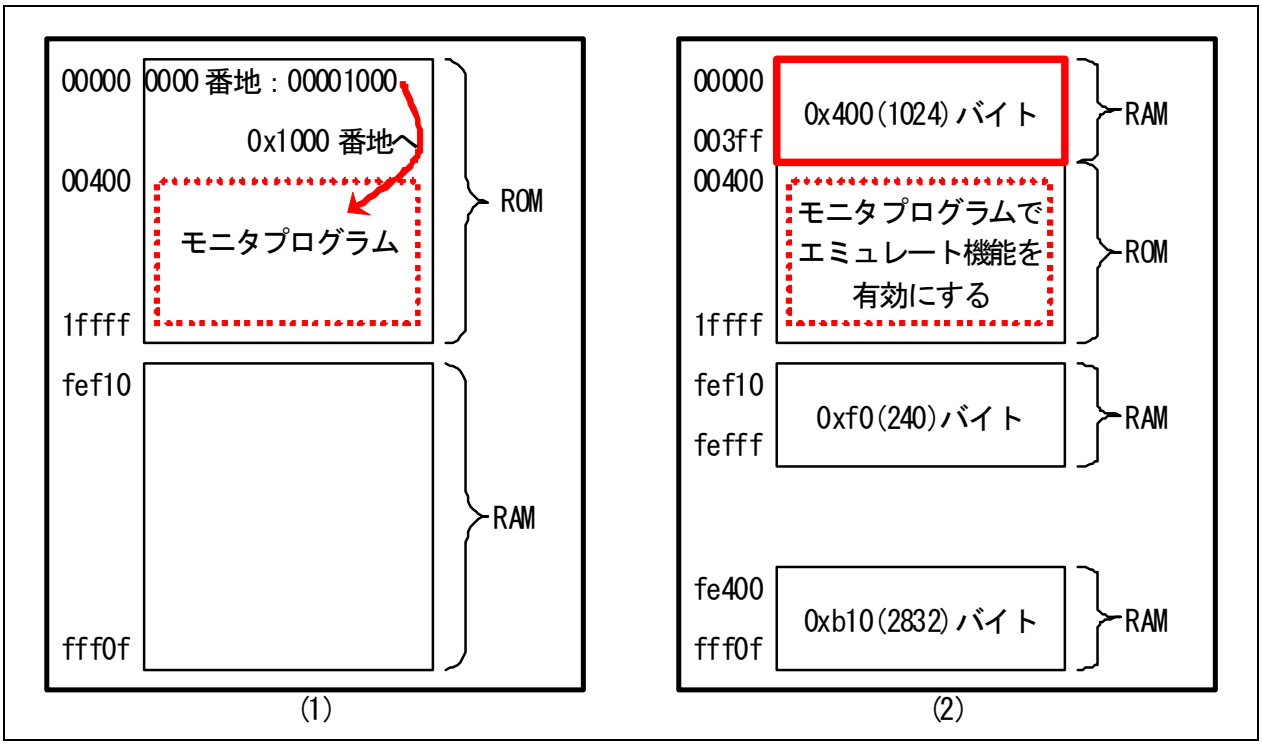


図 5. 1 シリアルモニタの動作(1) (2)

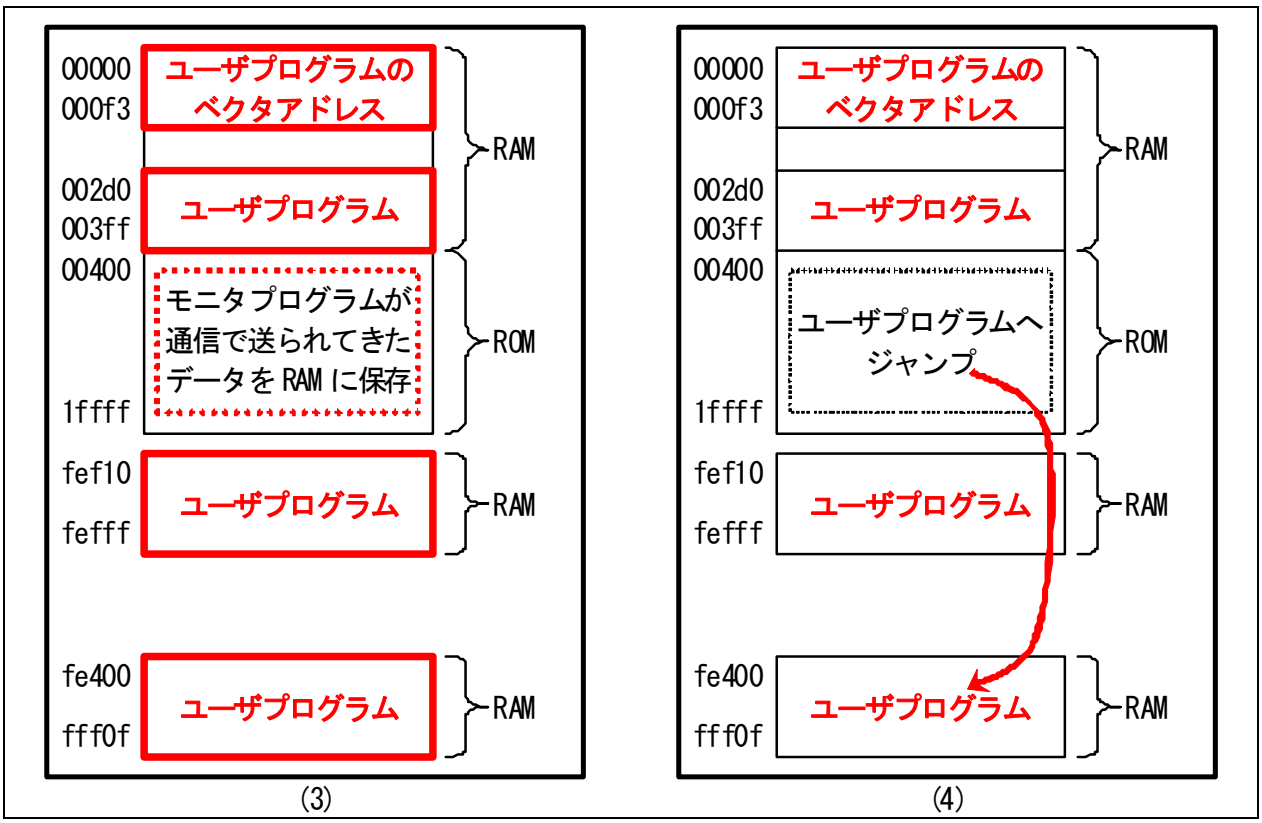


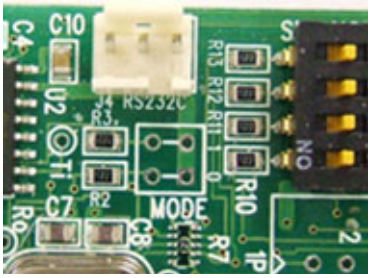
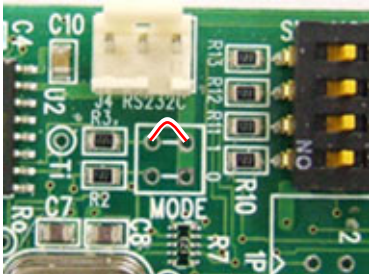
図 5. 1 シリアルモニタの動作(3) (4)

5.2 外付けのRAMを使用する

セクションや SP(スタックポインタ)領域をどの RAM 領域に配置するか工夫しても、大きいプログラム開発には限界があります。そこで、外付けの RAM を追加して、RAM 領域を増設したいと思います。

外付けRAMを接続するときは、H8/3048F-ONEを「内蔵ROM有効拡張 1Mバイトモード」と呼んでいるモード 5 で動作させます(表 5. 1)。

表 5. 1 シングルチップモードと内蔵 ROM 有効拡張 1M バイトモード

項目	シングルチップモード(モード 7)	内蔵 ROM 有効拡張 1M バイトモード(モード 5)
モードの 切り替え方法	 MODE 端子は開放します(特に何もしません)。	 MODE部分のランドの赤線部分をショートさせます。
外付けのメモリ との接続	モード 7 は外付けのメモリとは接続せず、内蔵の ROM、RAM のみを使用します。	外付けのメモリと接続することができます。H8/3048F-ONE と外付けメモリと接続は、RY3048Fone ボードの 34 ピンコネクタと 20 ピンコネクタの端子を使用します。そのため、これらの端子を I/O ポートとして使用することができません。

RY3048FoneボードとRAMの接続例を 図 5. 2に示します。

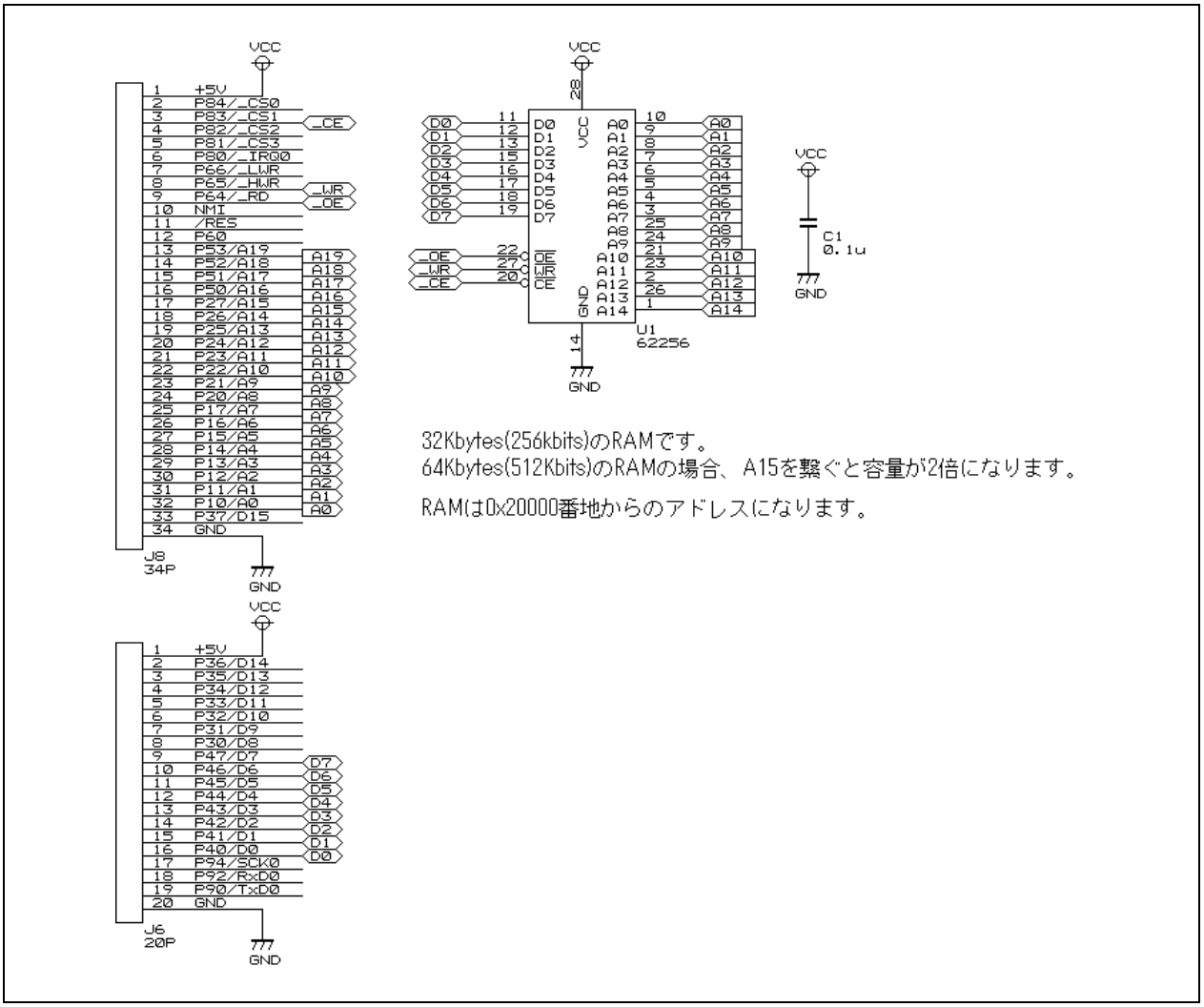
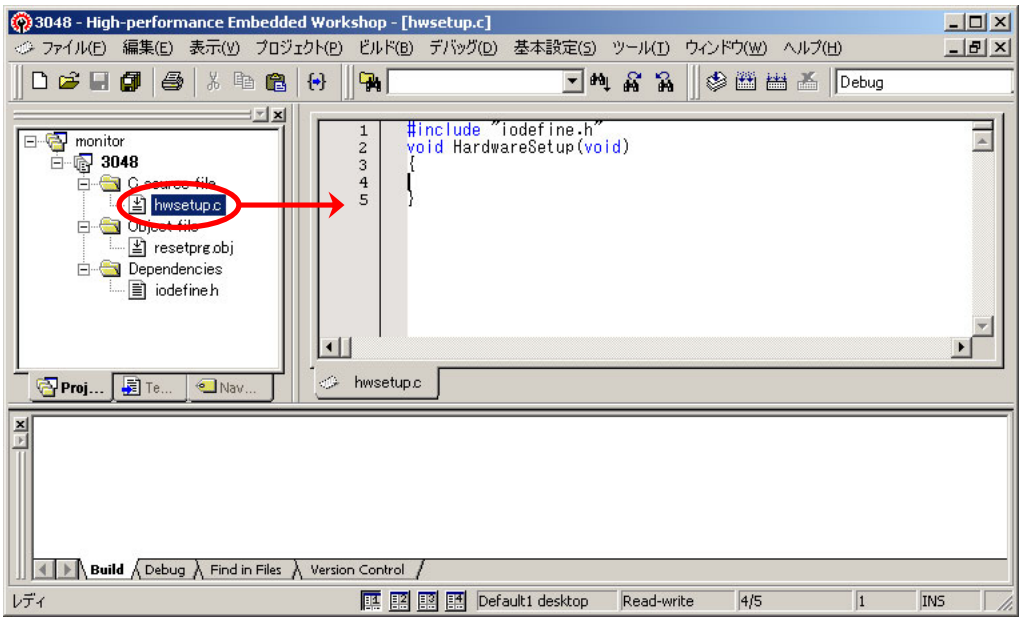


図 5. 2 内蔵 ROM 有効拡張 1M バイトモードで外付け RAM に接続する回路例

ワークスペース「monitor」のプロジェクト「3048」を開きます。「hwsetup.c」を開きます。

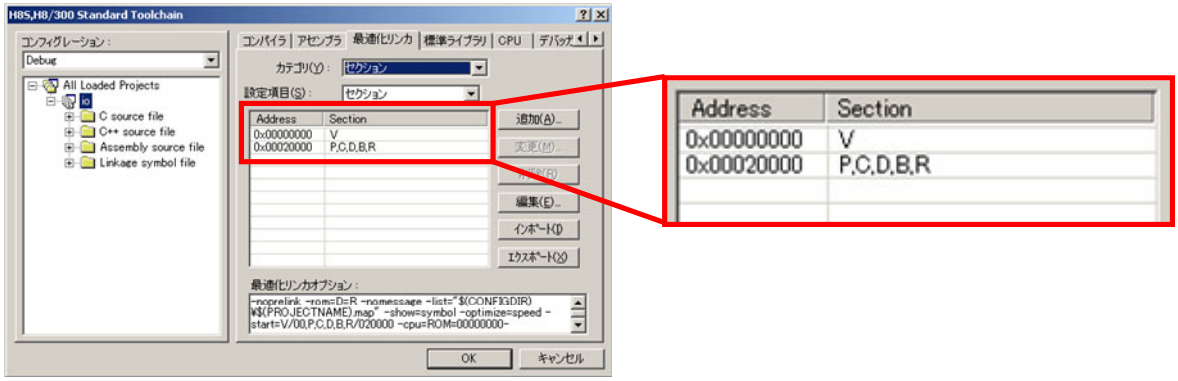


プログラムをリスト 5. 1のように入力します。このプログラムで、外付けRAMを有効にします。外付けRAMは、0x20000 番地から、メモリ容量分までのアドレスになります。例えば 32KBのRAMなら、0x20000～0x2ffff番地になります。ビルドして、エラーがないことを確認して、できあがったMOTファイルを書き込んでください。

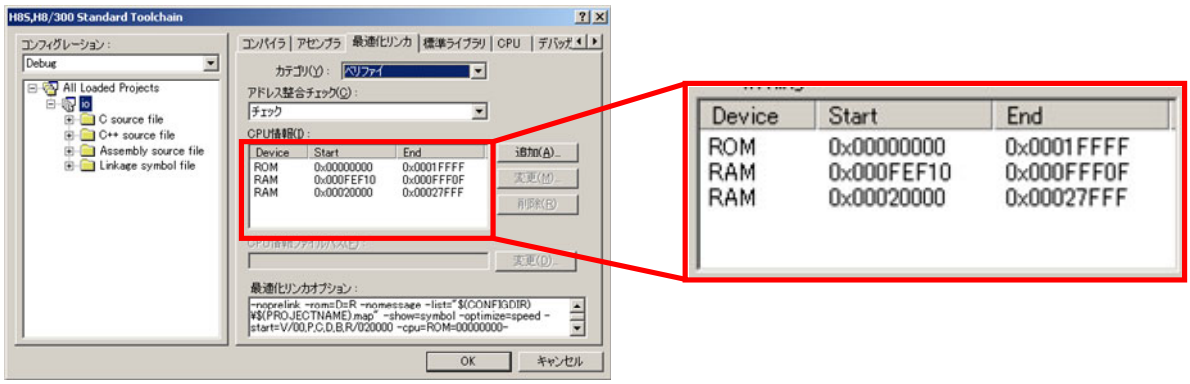
リスト 5. 1

```
#include "iodefine.h"
void HardwareSetup(void)
{
    BSC.ABWCR.BIT.ABW1 = 1;          // CS1 空間は８ビット幅
    BSC.ASTCR.BIT.AST1 = 1;          // CS1 空間は３ステート
    BSC.WCR.BYTE = 0xF0;             // プログラマブルウェイトモード、挿入なし
    BSC.WCER.BIT.WCE1 = 1;           // CS1 空間の WSC は有効
    P1.DDR = 0xFF;                   // A0-A7 端子は有効
    P2.DDR = 0xFF;                   // A8-A15 端子は有効
    P5.DDR = 0x07;                   // A16-A18 端子は有効
    P8.DDR = 0x08;                   // CS1 端子は有効
}
```

次に、ユーザプログラムのセクションを設定します。ツールチェインの「最適化リンカ」、「カテゴリ:**セクション**」、「設定項目:**セクション**」を選択します。ベクタアドレス(セクション V)は、0x00000 番地で変更しません。その他のセクションはすべて 0x20000 番地に設定します。容量が大きいので、セクション V 以外のすべてを外付け RAM に配置することができます。



次に、「カテゴリ:**ベリファイ**」を選択します。CPU 情報を外付け RAM の領域も有効になるよう設定します。



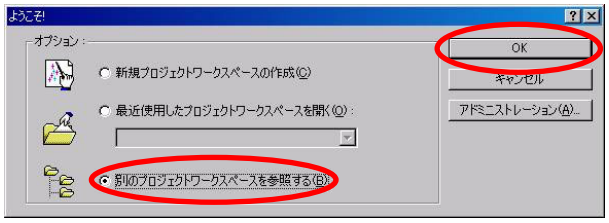
その他は、今までと同様です。これでメモリ容量をほとんど気にせずにプログラムを作成することができます。いろいろなプログラムを作成しましょう！

6. 演習

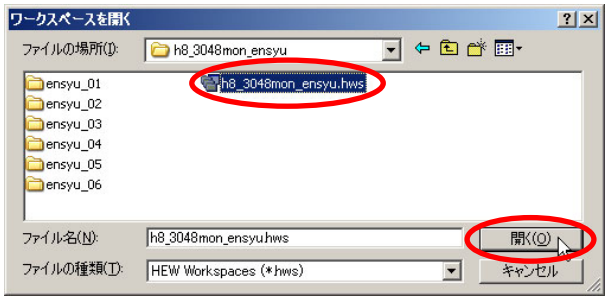
※RY3048Fone ボードにシリアルモニタ(ワークスペース「monitor」の monitor.mot)を書き込んでいない場合は、書き込んでください。

6.1 ワーススペース「h8_3048mon_ensyu」を開く

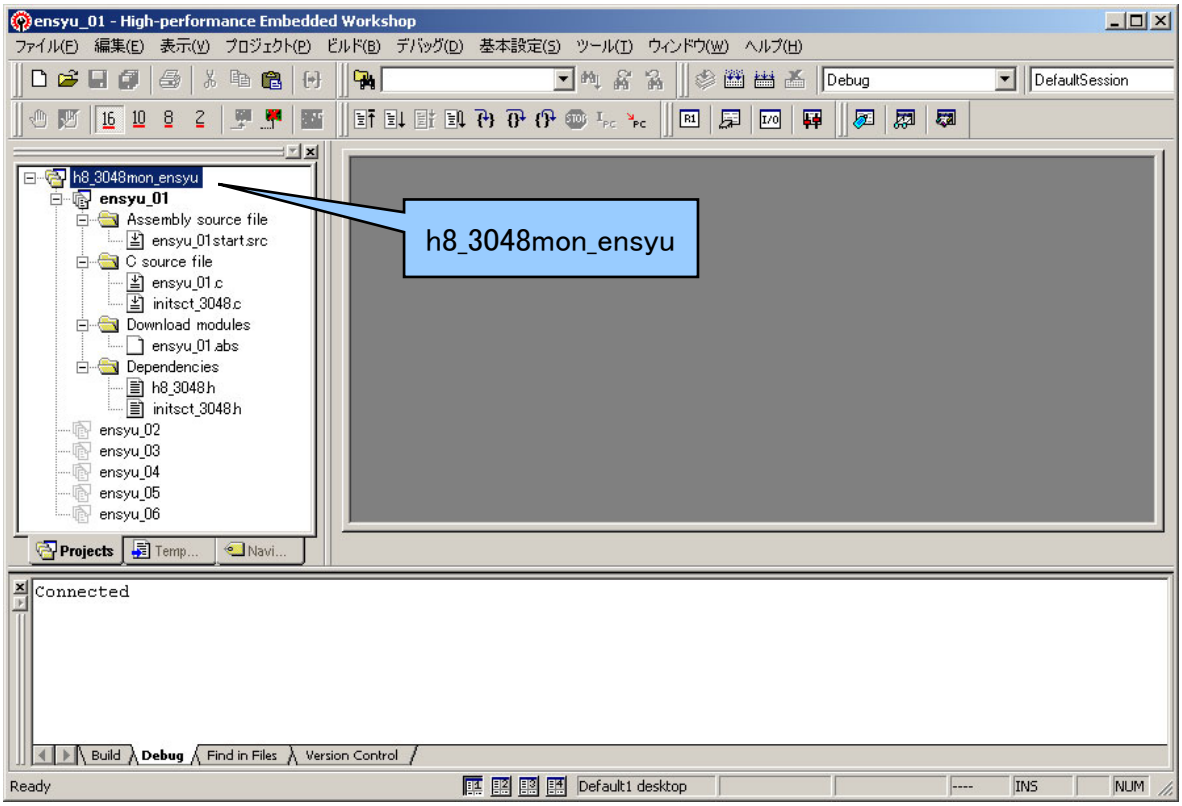
1. ルネサス統合開発環境を実行します。



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

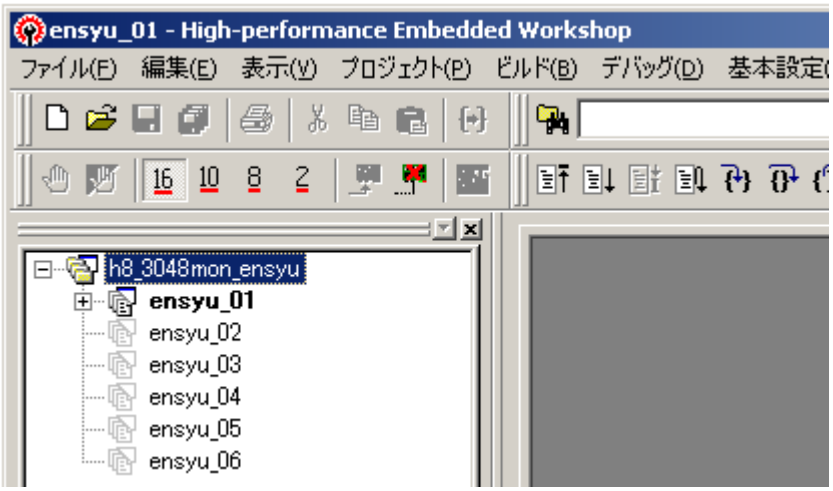


3. Cドライブ→Workspace→h8_3048mon_ensyu の「h8_3048mon_ensyu.hws」を選択、開くをクリックします。



4.「h8_3048mon_ensyu」というワークスペースが開かれます。

6.2 プロジェクト



ワークスペース「h8_3048mon_ensyu」には、6 つのプロジェクトが登録されています。ensyu_05 以外のプロジェクトは、不完全なプログラムが登録されています。シリアルモニタを使って動作を確認、正しいプログラムに書き換えてください。ensyu_05 は、どのような動作になるか確認してください。

プロジェクト名	内容
ensyu_01	if 文を使った演習です。
ensyu_02	変数の型についての演習です。
ensyu_03	変数の型の範囲についての演習です。
ensyu_04	優先順位についての演習です。
ensyu_05	演算子についての演習です。今回は、「！」(ビックリマーク、否定の論理演算子)を例として使用します。
ensyu_06	データをソートするプログラムを作る演習です。

6.3 プロジェクト「ensyu_01」 if文を使った演習

6.3.1 概要

ポート 7（ディップスイッチ）の値が、0x01 かどうかチェックします。0x01 なら、ポート A(LED)に 0x55 を出力、違うなら 0x00 を出力するプログラムを作りました。
しかし、正しく動作しない様です。シリアルモニタでプログラムの動作を確認しながら、バグを見つけましょう。

6.3.2 接続

図 6. 1　結線図の様に接続します。

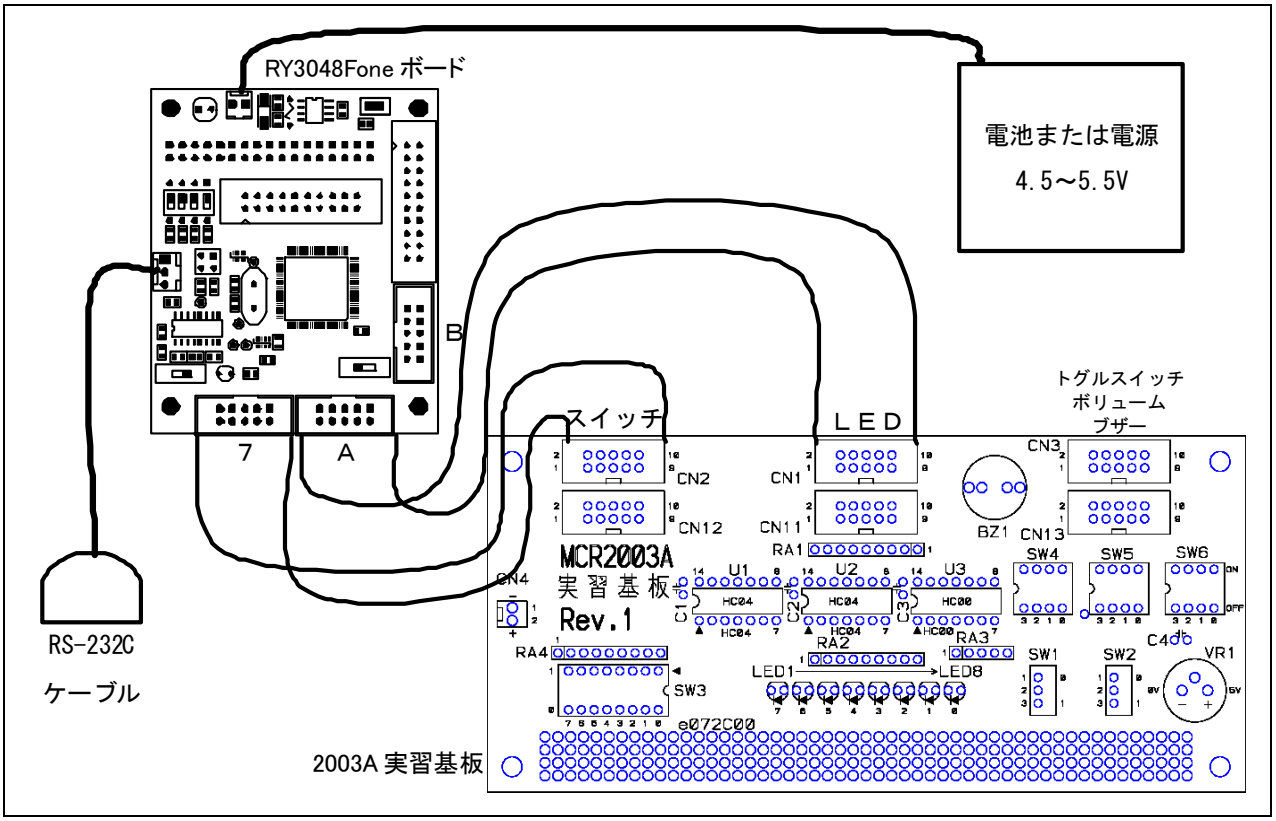
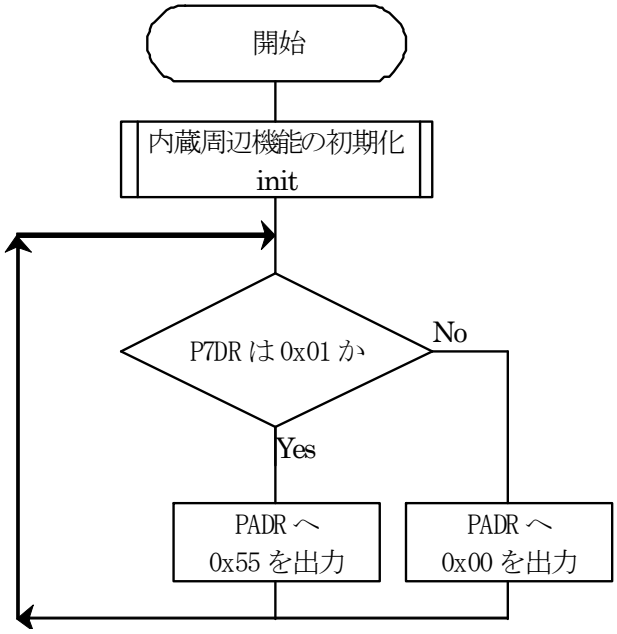


図 6. 1　結線図

・RY3048one ボードのポート 7 と 2003A 実習基板のスイッチ部分(CN2)をフラットケーブルで接続
・RY3048one ボードのポート A と 2003A 実習基板の LED 部分(CN1)をフラットケーブルで接続
・RY3048one ボードに 4.5～5.5V を供給

6.3.3 プログラムのフローチャート



6.3.4 プログラム「ensyu_01.c」

main 関数を抜粋します。下記プログラムは、フローチャートどおりの動作をしません。どの部分に間違いがあるか、シリアルモニタを使って調べてみましょう。

```
23 :  void main( void )
24 :  {
25 :      unsigned char d;
26 :
27 :      init();                              /* 初期化                    */
28 :
29 :      while( 1 ) {
30 :          d = P7DR;
31 :          if( d = 0x01 ) {
32 :              PADR = 0x55;
33 :          } else {
34 :              PADR = 0x00;
35 :          }
36 :      }
37 :  }
```

6.4 プロジェクト「ensyu_02」 変数の型についての演習

6.4.1 概要

ポート 7(ディップスイッチ)の状態を反転したとき、0x01 かどうかチェックします。0x01 なら、ポート A(LED)に 0x55 を出力、違うなら 0x00 を出力するプログラムを作りました。
しかし、正しく動作しない様です。シリアルモニタでプログラムの動作を確認しながら、バグを見つけましょう。

6.4.2 接続

図 6. 2　結線図の様に接続します。プロジェクト「ensyu_01」と同じです。

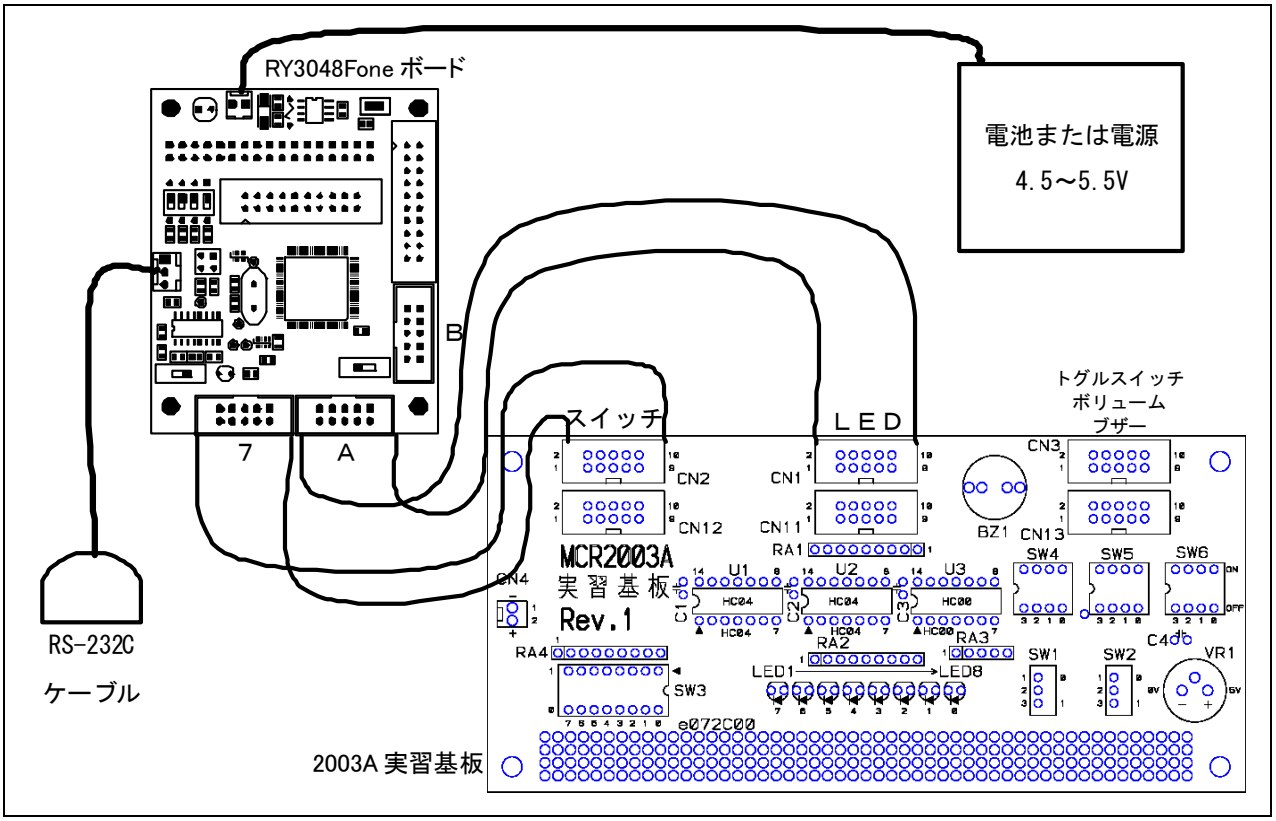
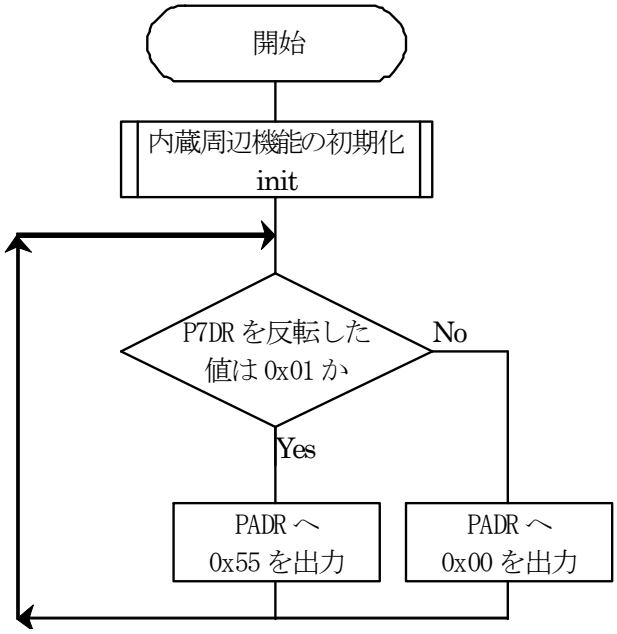


図 6. 2　結線図

・RY3048one ボードのポート 7 と 2003A 実習基板のスイッチ部分(CN2)をフラットケーブルで接続
・RY3048one ボードのポート A と 2003A 実習基板の LED 部分(CN1)をフラットケーブルで接続
・RY3048one ボードに 4.5～5.5V を供給

6.4.3 プログラムのフローチャート



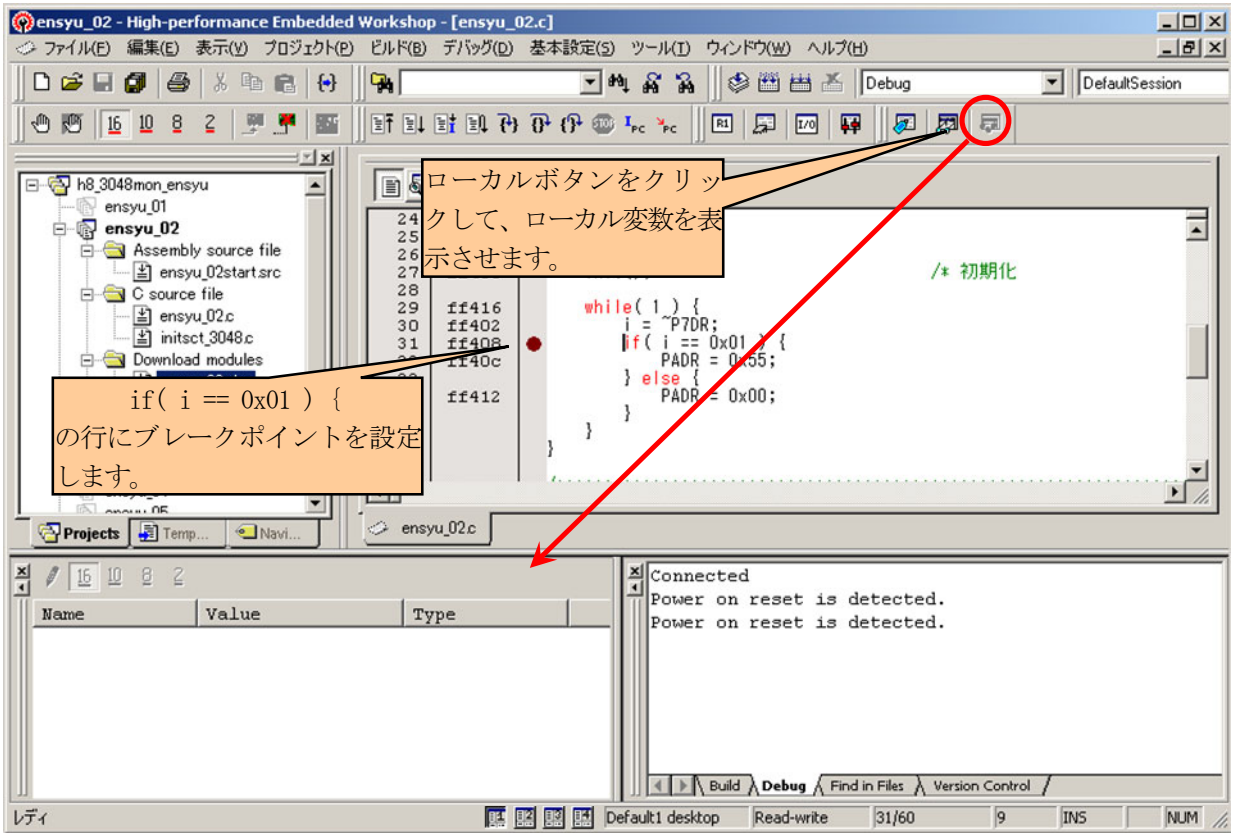
6.4.4 プログラム「ensyu_02.c」

main 関数を抜粋します。下記プログラムは、フローチャートどおりの動作をしません。どの部分に間違いがあるか、シリアルモニタを使って調べてみましょう。

```
 23 :  void main( void )
 24 :  {
 25 :      int i;
 26 :
 27 :      init();                              /* 初期化                    */
 28 :
 29 :      while( 1 ) {
 30 :          i = ~P7DR;
 31 :          if( i == 0x01 ) {
 32 :              PADR = 0x55;
 33 :          } else {
 34 :              PADR = 0x00;
 35 :          }
 36 :      }
 37 :  }
```

6.4.5 シリアルモニタの操作ポイント

1. ローカルボタンをクリックして、ローカル変数を表示させます。
2. 31 行目にブレークポイントを設定します。これでプログラム実行時、30 行目を実行し終わって、31 行目に来た瞬間、プログラムが停止します。



リセット後実行ボタン(SHIFT+F5 キー)でプログラムを実行します。
31 行目で停止したとき、ローカル変数に変数 i の値が表示されます。この値と、ディップスイッチの反転した値が一致するか、調べてみましょう。もし一致しない場合、なぜ一致しないのか検討してみてください。

6.5 プロジェクト「ensyu_03」 変数の型の範囲についての演習

6.5.1 概要

ポート 7 の bit0 をアナログ入力端子にして、0～5V を入力します。入力した電圧を、A/D 変換器で 0～1023 の値に変換、さらにプログラムで 0～1023 の値を 0～100 の値に変換して、ポート A(LED)に出力します。
しかし、正しく動作しない様です。シリアルモニタでプログラムの動作を確認しながら、バグを見つけましょう。

6.5.2 接続

図 6. 3　結線図の様に接続します。

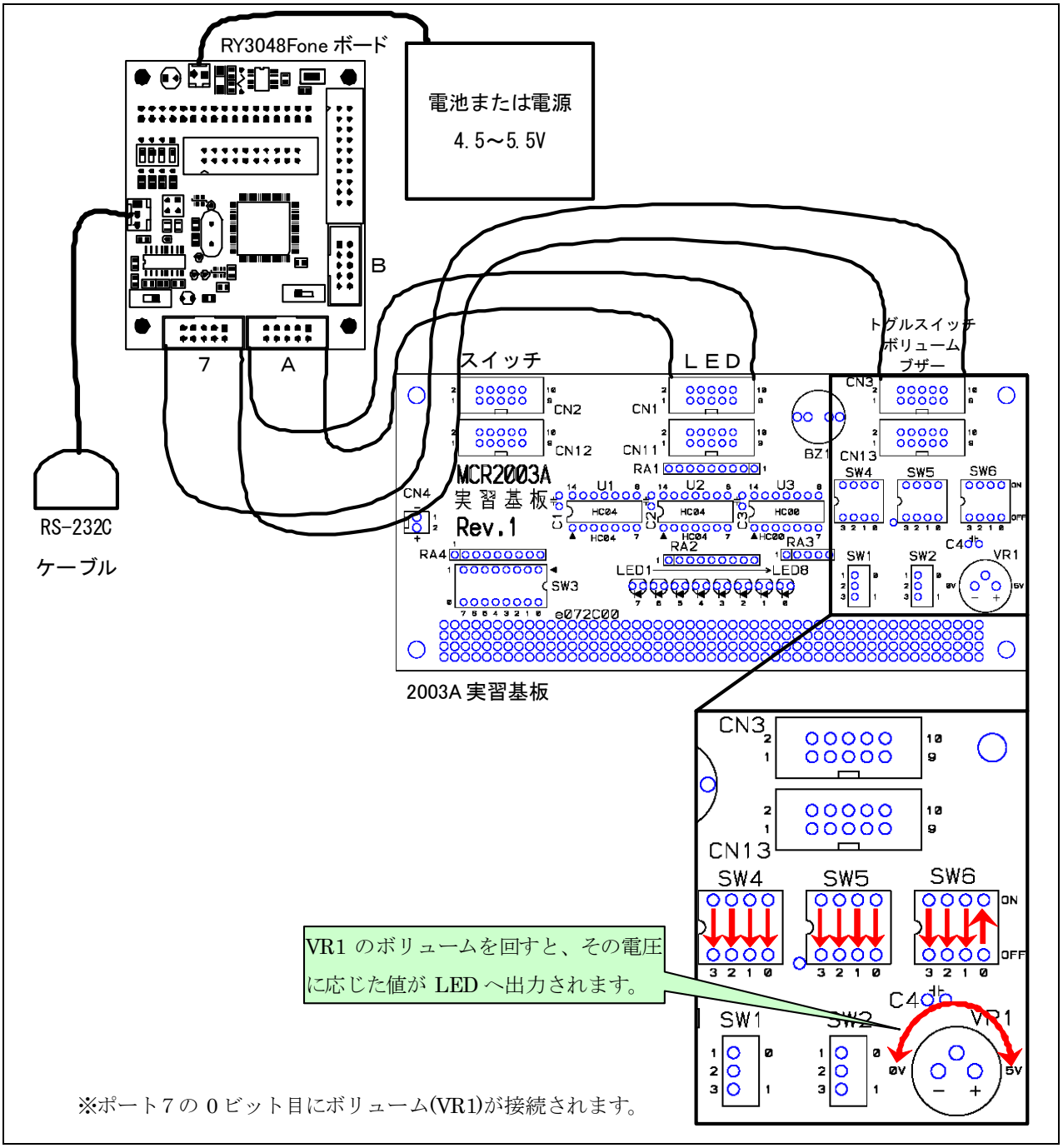
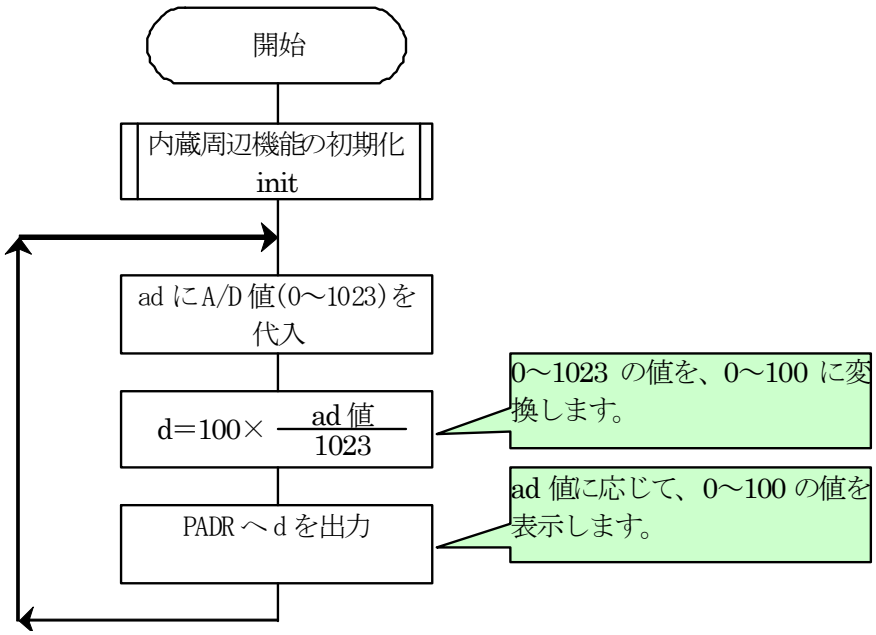


図 6. 3　結線図

・RY3048one ボードのポート 7 と 2003A 実習基板のトグルスイッチ・ボリューム・ブザー部分(CN3)をフラットケーブルで接続
・RY3048one ボードのポート A と 2003A 実習基板の LED 部分(CN1)をフラットケーブルで接続
・RY3048one ボードに 4.5～5.5V を供給

6.5.3 プログラムのフローチャート



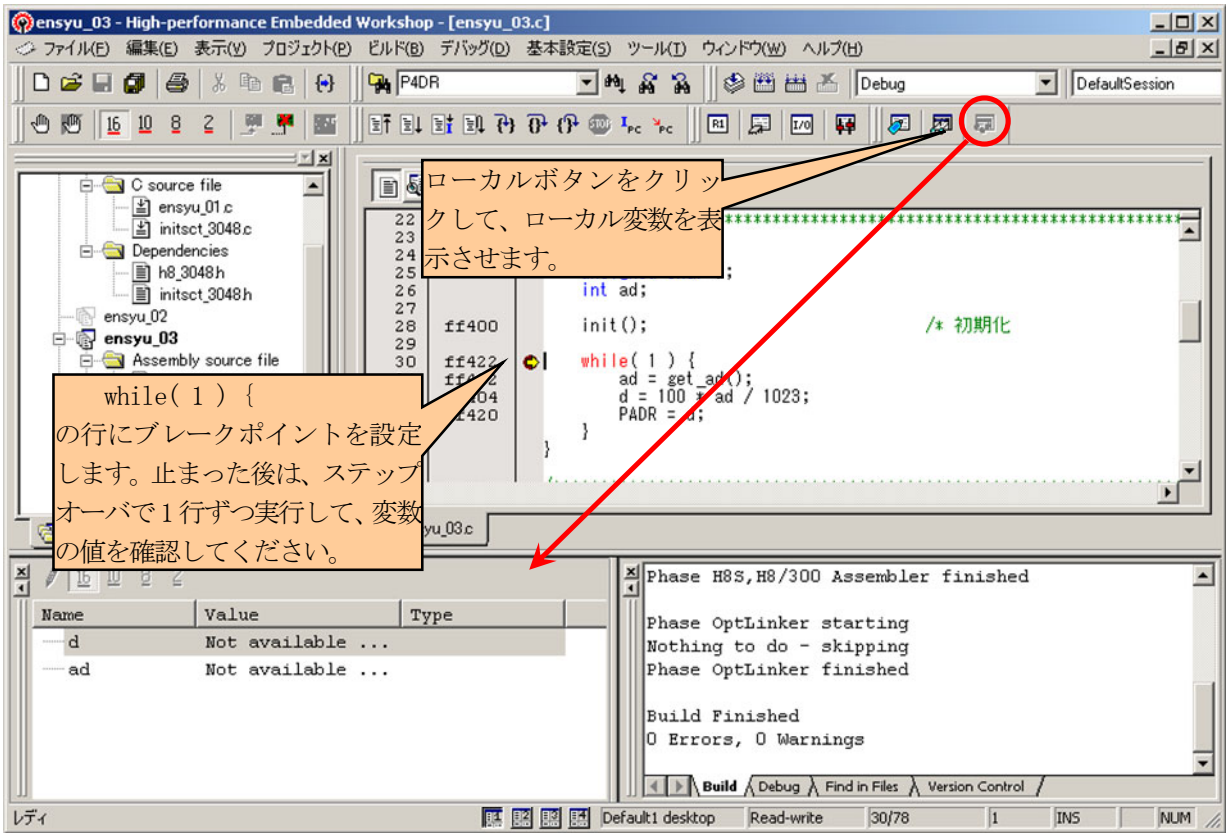
6.5.4 プログラム「ensyu_03.c」

main 関数を抜粋します。下記プログラムは、フローチャートどおりの動作をしません。どの部分に間違いがあるか、シリアルモニタを使って調べてみましょう。

```
23 :  void main( void )
24 :  {
25 :      unsigned char d;
26 :      int ad;
27 :
28 :      init();                              /* 初期化                    */
29 :
30 :      while( 1 ) {
31 :          ad = get_ad();
32 :          d = 100 * ad / 1023;
33 :          PADR = d;
34 :      }
35 :  }
```

6.5.5 シリアルモニタの操作ポイント

1. ローカルボタンをクリックして、ローカル変数を表示させます。
2. 30 行目にブレークポイントを設定します。



リセット後実行ボタン(SHIFT+F5 キー)でプログラムを実行します。

プログラムを実行すると、30 行目に来た瞬間、プログラムが停止します。この後は、ステップオーバ(F10 キー)で 1 行ずつ実行して、変数 ad や d の値を確認してみてください。もし正しい計算結果になっていない場合、なぜ正しく計算されないのか検討してみてください。

6.6 プロジェクト「ensyu_04」 優先順位についての演習

6.6.1 概要

ポート 7(ディップスイッチ)の上位 4 ビットをマスクします。下位 4 ビットの値が、0x01 かどうかチェックします。0x01 なら、ポート A(LED)に 0x55 を出力、違うなら 0x00 を出力するプログラムを作りました。
しかし、正しく動作しない様です。シリアルモニタでプログラムの動作を確認しながら、バグを見つけましょう。

6.6.2 接続

図 6. 4 結線図の様に接続します。プロジェクト「ensyu_01」と同じです。

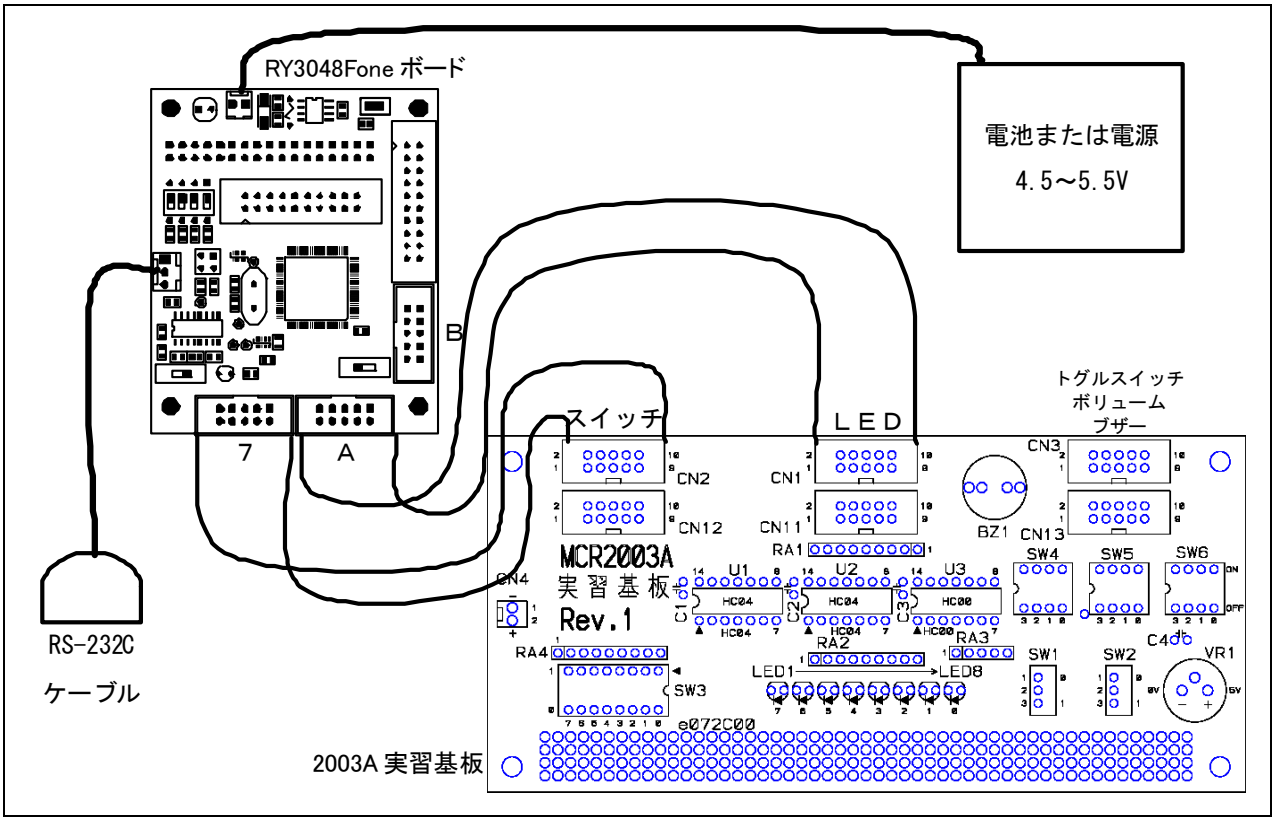
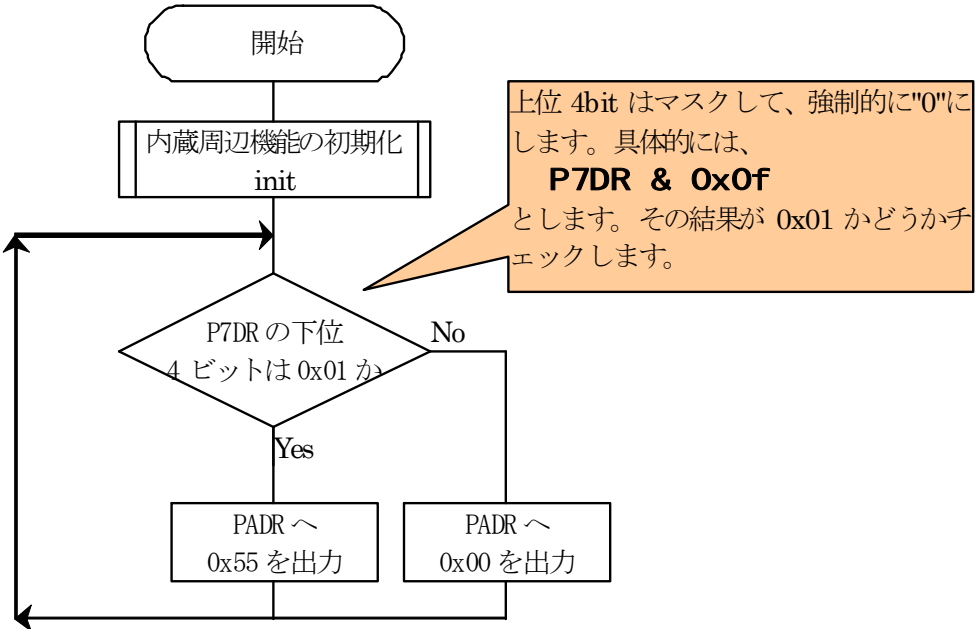


図 6. 4 結線図

・RY3048one ボードのポート 7 と 2003A 実習基板のスイッチ部分(CN2)をフラットケーブルで接続
・RY3048one ボードのポート A と 2003A 実習基板の LED 部分(CN1)をフラットケーブルで接続
・RY3048one ボードに 4.5～5.5V を供給

6.6.3 プログラムのフローチャート



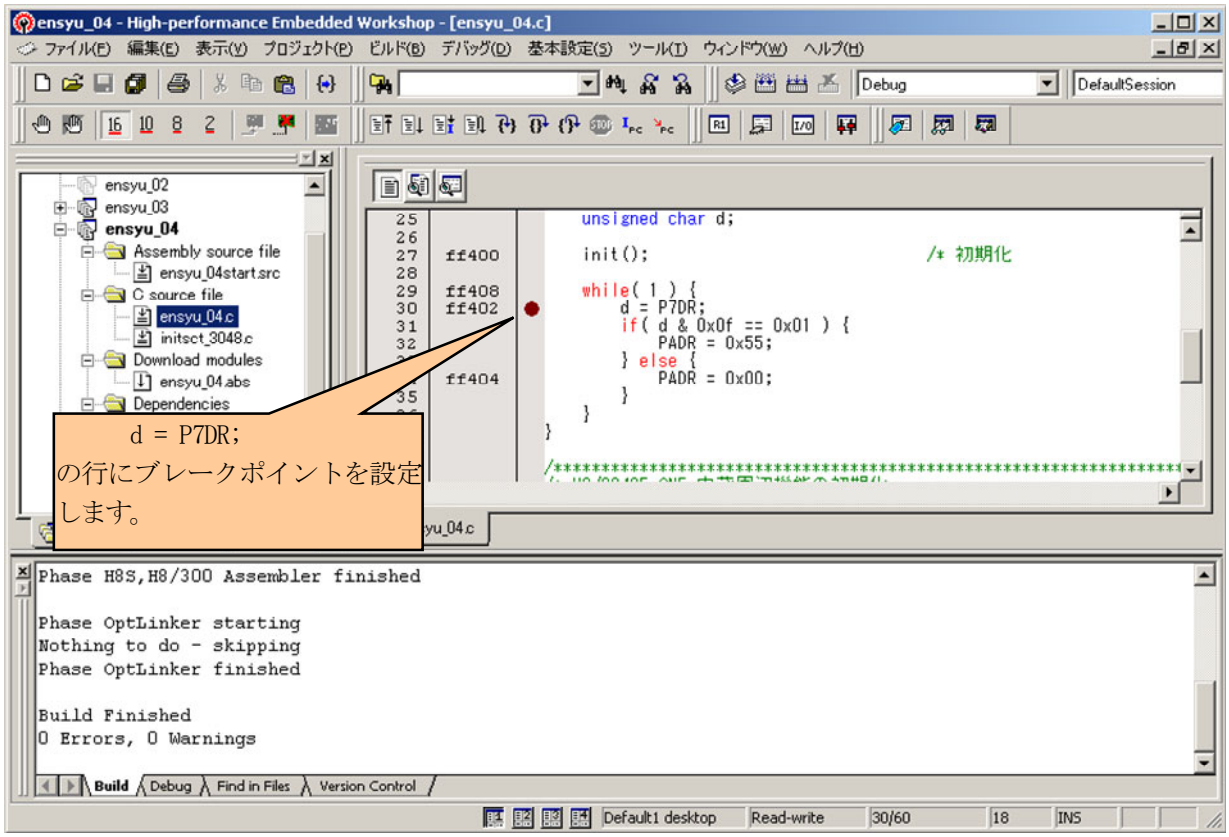
6.6.4 プログラム「ensyu_04.c」

main 関数を抜粋します。下記プログラムは、フローチャートどおりの動作をしません。どの部分に間違いがあるか、シリアルモニタを使って調べてみましょう。

```
23 :  void main( void )
24 :  {
25 :      unsigned char d;
26 :
27 :      init();                              /* 初期化                    */
28 :
29 :      while( 1 ) {
30 :          d = P7DR;
31 :          if( d & 0x0f == 0x01 ) {
32 :              PADR = 0x55;
33 :          } else {
34 :              PADR = 0x00;
35 :          }
36 :      }
37 :  }
```

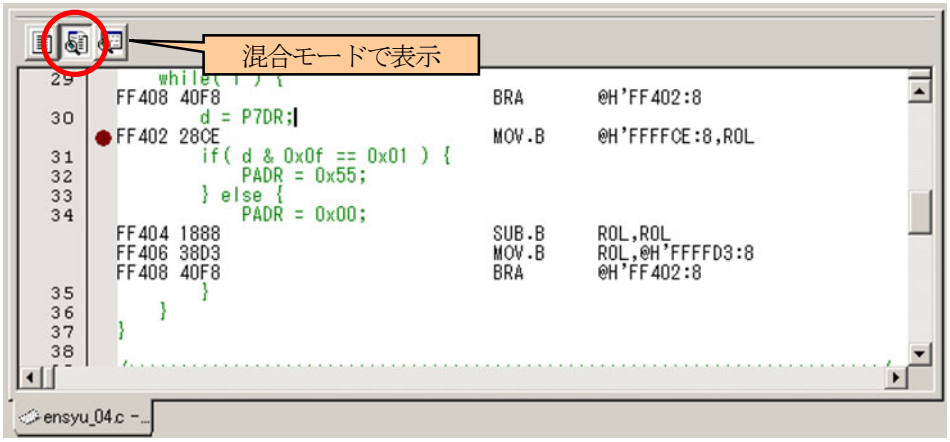
6.6.5 シリアルモニタの操作ポイント

30 行目にブレークポイントを設定します。これでプログラム実行時、ブレークポイントを設定した行に来た瞬間、プログラムが停止します。



リセット後実行ボタン(SHIFT+F5 キー)でプログラムを実行します。
プログラムの実行が、30 行目に来たとき停止します。ステップオーバ(F10 キー)でプログラムがどう実行されるか 1 行ずつ確認していきましょう。予期せぬ動きをする場合、なぜそのような動きをするのか検討してみてください。

アセンブラ言語がある程度分かるなら、C 言語とアセンブラ言語を混在して表示させる「混合モードで表示」にすると、コンパイラがどのようなアセンブリ言語に変換したか分かり、プログラムの解析がしやすくなります。



6.7 プロジェクト「ensyu_05」 演算子についての演習

6.7.1 概要

ポート 7(ディップスイッチ)の値を、変数 d に代入します。その値を「!演算子」を使って計算、PADR(LED)へ出力するプログラムを作りました。
どのように動作するか、シリアルモニタで調べてみましょう。

6.7.2 接続

図 6. 5の様に接続します。プロジェクト「ensyu_01」と同じです。

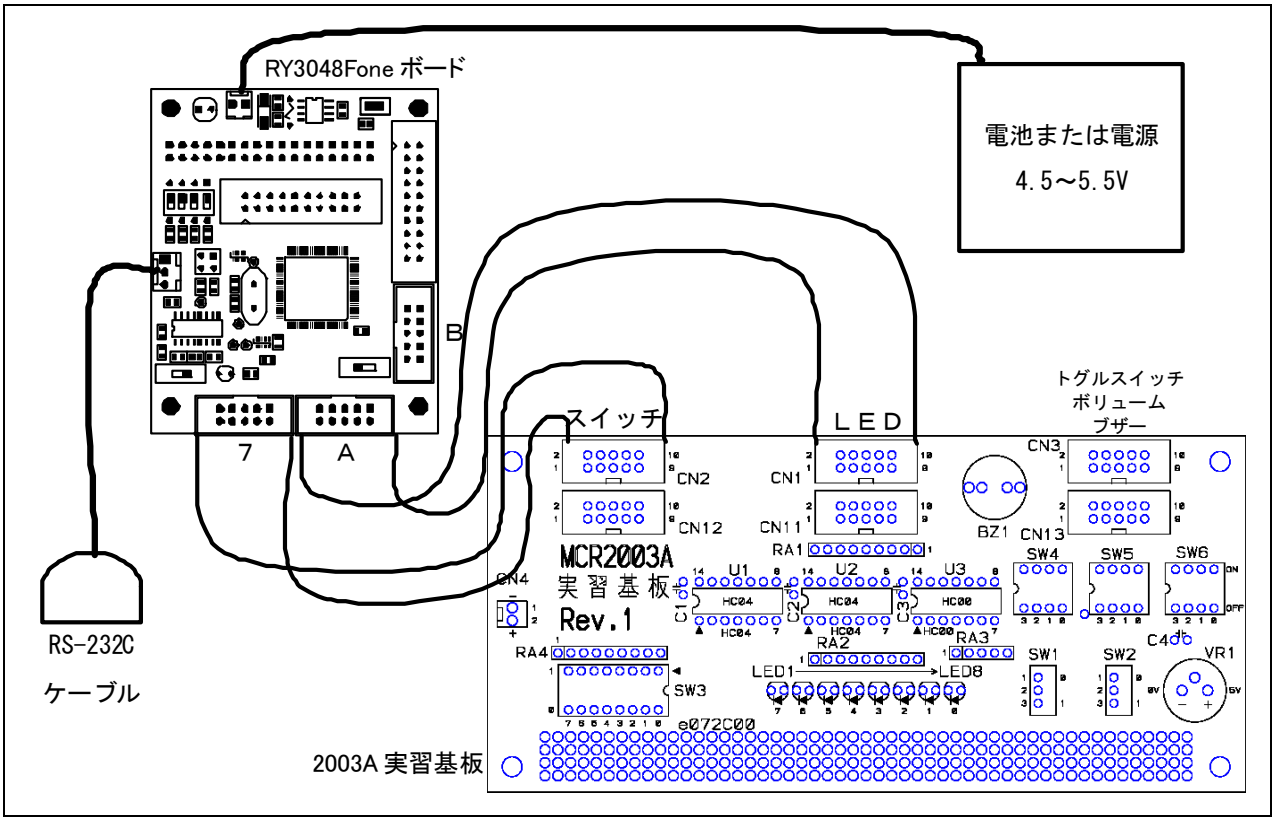
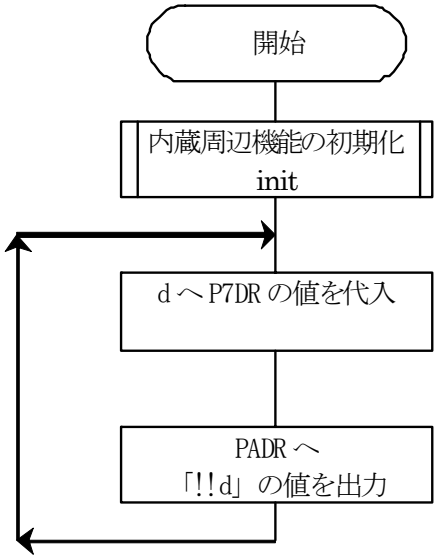


図 6. 5　結線図

・RY3048one ボードのポート 7 と 2003A 実習基板のスイッチ部分(CN2)をフラットケーブルで接続
・RY3048one ボードのポート A と 2003A 実習基板の LED 部分(CN1)をフラットケーブルで接続
・RY3048one ボードに 4.5～5.5V を供給

6.7.3 プログラムのフローチャート



6.7.4 プログラム「ensyu_05.c」

main 関数を抜粋します。ポート A にはどのような値が出力されるか、シリアルモニタを使って調べてみましょう。

```
23 :  void main( void )
24 :  {
25 :      unsigned char d;
26 :
27 :      init();                              /* 初期化                    */
28 :
29 :      while( 1 ) {
30 :          d = P7DR;
31 :          PADR = !!d;
32 :      }
33 :  }
```

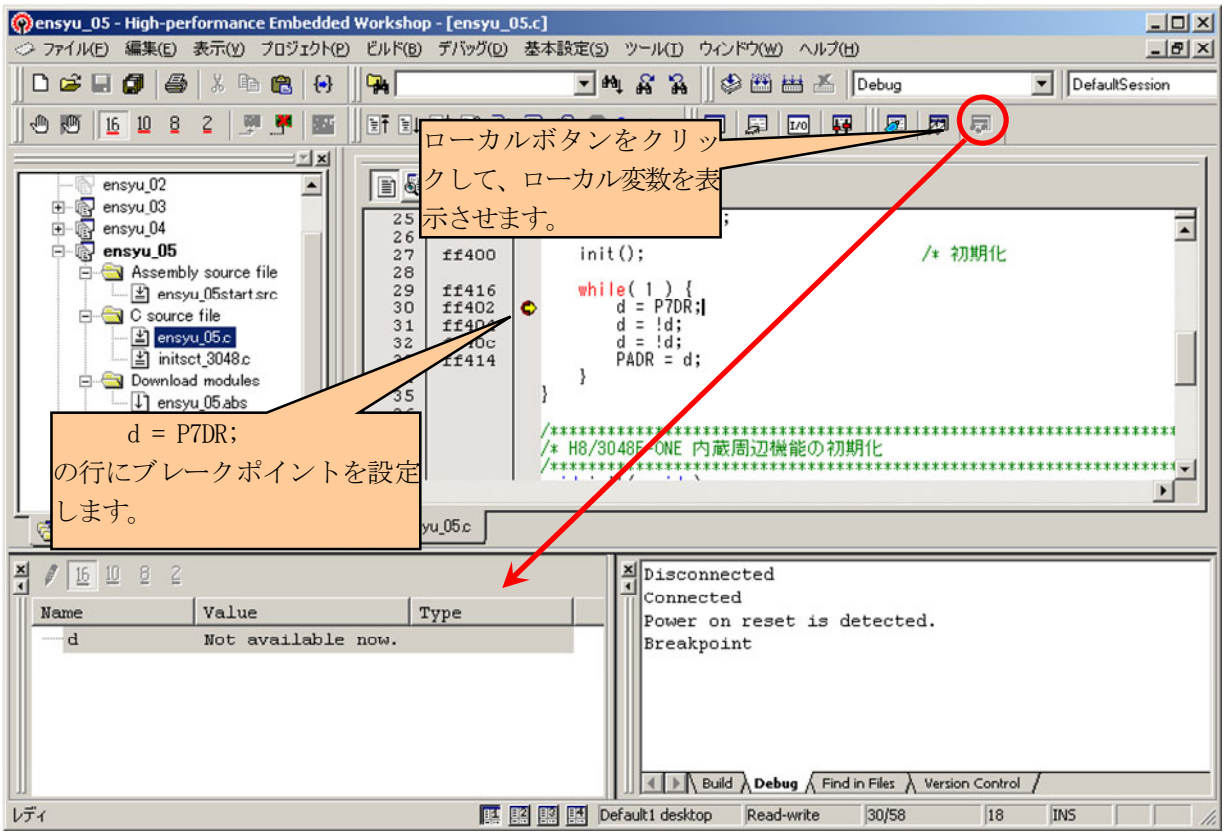
6.7.5 シリアルモニタの操作ポイント

「!!d」は分解すると、「!d」をさらに「!」するということです。1 行で処理せずに、複数の行に分解して、1 行ずつ値を確認しましょう。

1. プログラムを次のように変更します。

PADR = !!d;	d = !d; d = !d; PADR = d;

2. ローカルボタンをクリックして、ローカル変数を表示させます。
3. 30 行目にブレークポイントを設定します。30 行目に来た瞬間、プログラムが停止します。



リセット後実行ボタン(SHIFT+F5 キー)でプログラムを実行します。
プログラムの実行が、30 行目に来たとき停止します。ステップオーバ(F10 キー)でプログラムがどう実行されるか 1 行ずつ確認していきましょう。なぜそのような動きをするのか検討してみてください。

アセンブラ言語がある程度分かるなら、C 言語とアセンブラ言語を混在して表示させる「混合モードで表示」にすると、コンパイラがどのようなアセンブリ言語に変換したか分かり、プログラムが追いやすくなります。

6.8 プロジェクト「ensyu_06」 データをソートするプログラムについての演習

6.8.1 概要

10 個のデータが格納できる配列 data に 0～9 までの値が一つずつ格納されています。これらの値をソートするプログラムを作ってみましょう。うまく動作しない場合、シリアルモニタで調べながらプログラムを検証しましょう。

6.8.2 接続

図 6.6 結線図の様に接続します。プロジェクト「ensyu_01」と同じです。

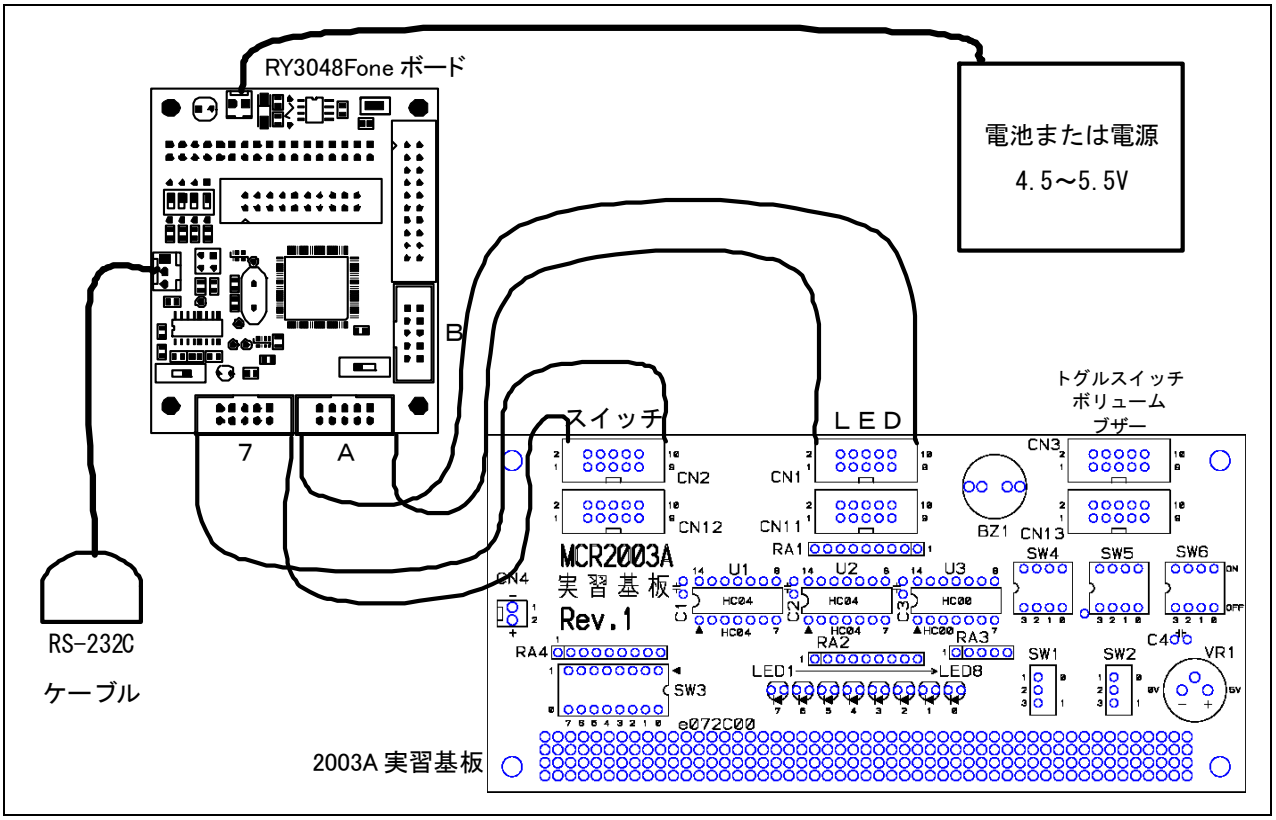
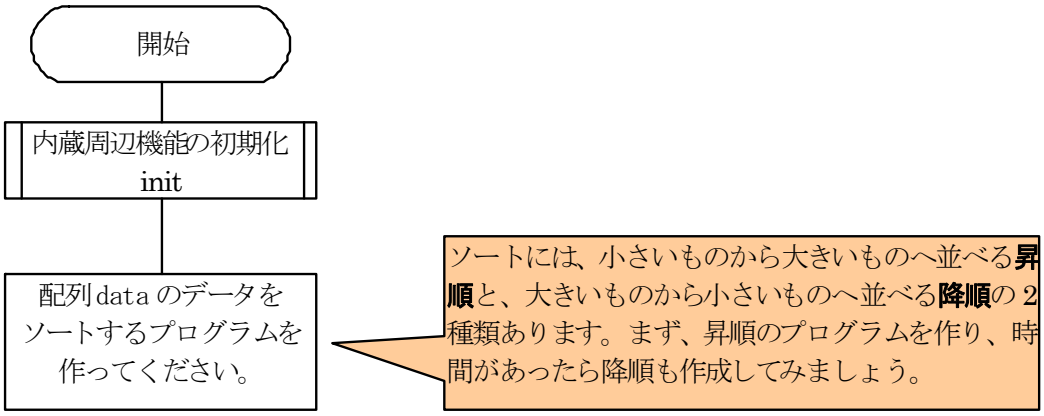


図 6.6 結線図

・RY3048one ボードのポート 7 と 2003A 実習基板のスイッチ部分(CN2)をフラットケーブルで接続
・RY3048one ボードのポート A と 2003A 実習基板の LED 部分(CN1)をフラットケーブルで接続
・RY3048one ボードに 4.5～5.5V を供給

6.8.3 プログラムのフローチャート



6.8.4 プログラム「ensyu_06.c」

プログラムを抜粋します。36 行目以降に、data 配列の値を昇順でソートするプログラムを作ってみましょう。

```
 14 :  #define SORT_SIZE   10                  /* ソートするサイズ              */
中略
 24 :  char data[SORT_SIZE] = { 5, 9, 1, 7, 6, 3, 8, 4, 0, 2 };
 25 :
中略
 29 :  void main( void )
 30 :  {
 31 :      int i, j, temp;
 32 :
 33 :      init();                             /* 初期化                        */
 34 :
 35 :      /* 配列 data をソートするプログラムを作ってみましょう */
 36 :
 37 :
 38 :
 39 :  }
```

6.8.5 ソートプログラムについて

ソート方法については、色々な方法(アルゴリズム)があります。インターネットで「C 言語 ソート」などで検索するとたくさんのページが見つかりますので、そちらで調べてみてください。
今回は、単純なバブルソート(単純交換法)を紹介します。

最初、data[0]の値とその他の配列の値を比較、data[0]の値が大きければ交換します(図 6.7)。

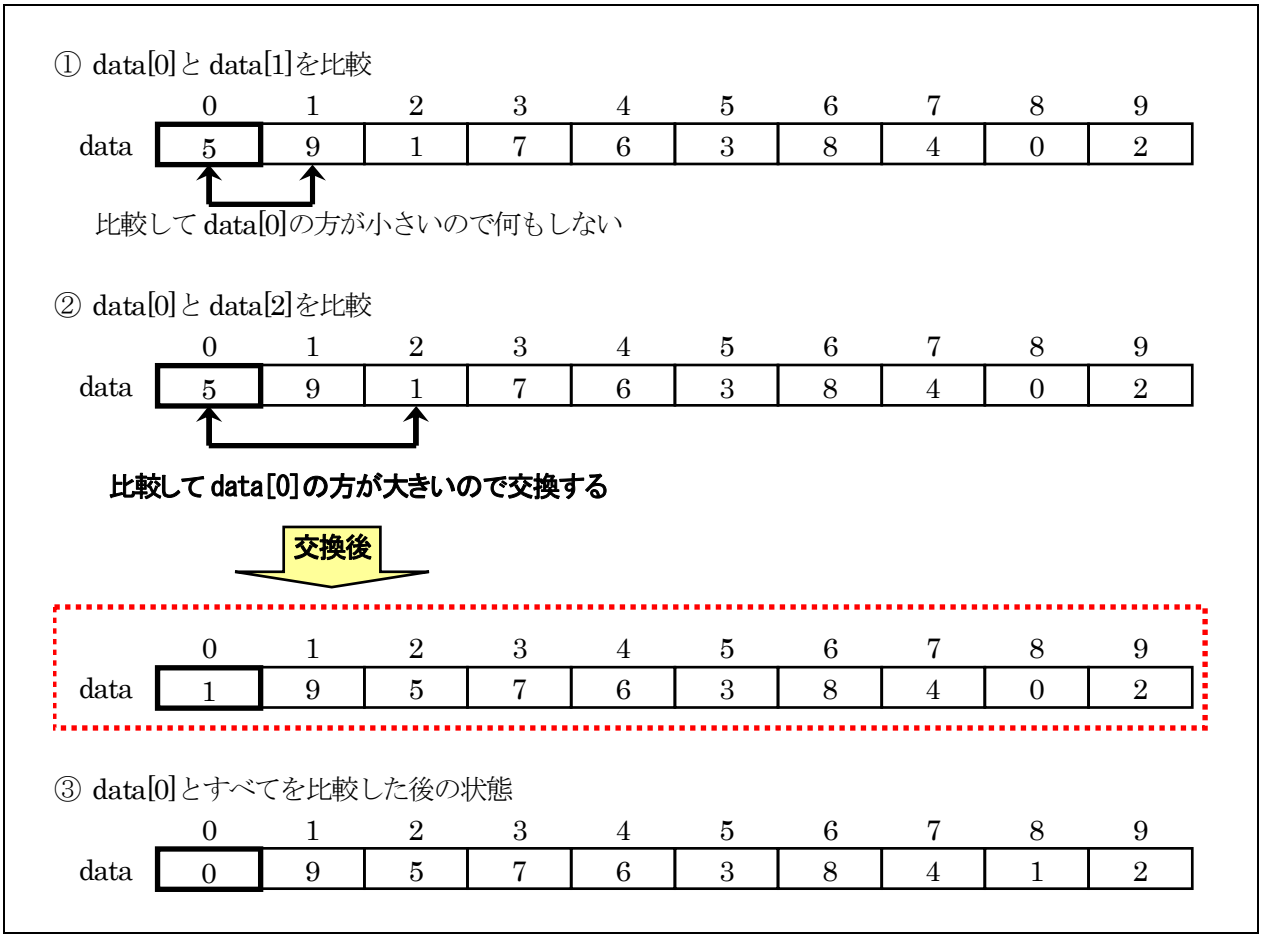


図 6.7 data[0]の比較の様子

次に、data[1]の値とdata[2]以降の配列の値を比較、data[1]の値が大きければ交換します（図 6. 8）。

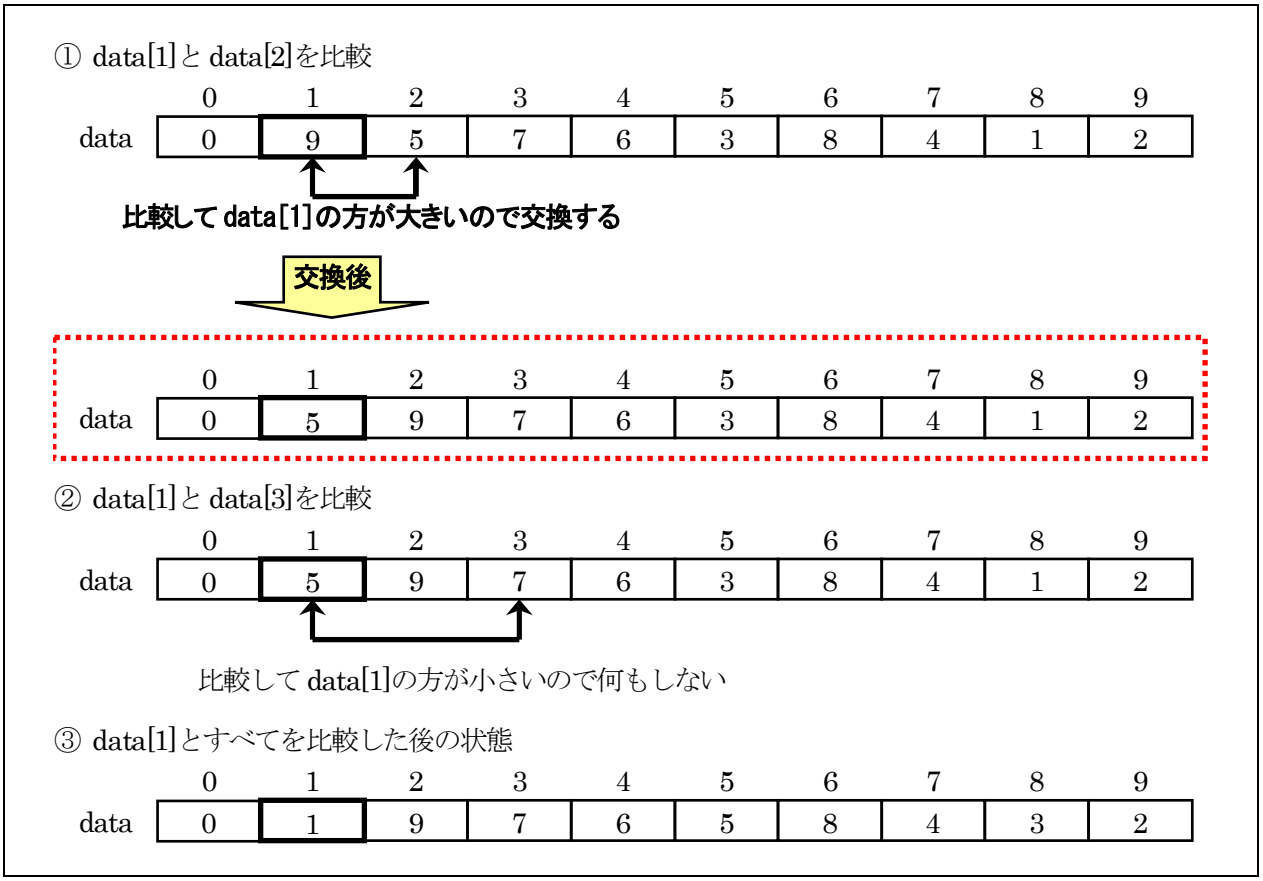


図 6. 8　data[1]の比較の様子

このように、data[3]、data[4]・・・、と比較を続けていきます。最後はdata[8]とdata[9]を比較することにより、全ての比較を行いソートが完了します。最終的には、0～9 まで昇順にソートされることになります（図 6. 9）。

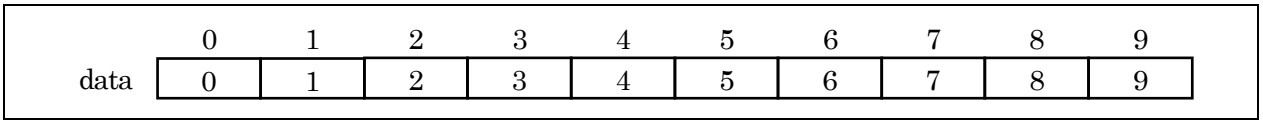
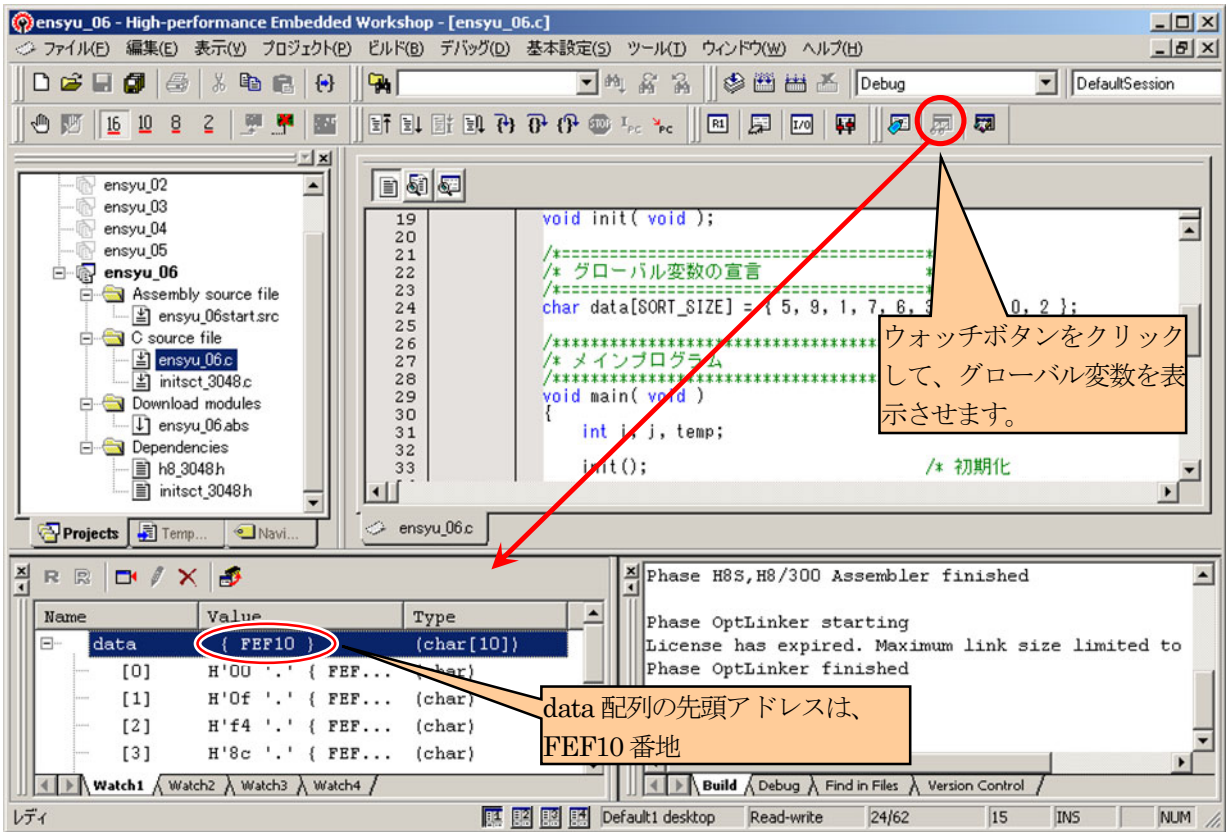


図 6. 9　最終的な data 配列の値

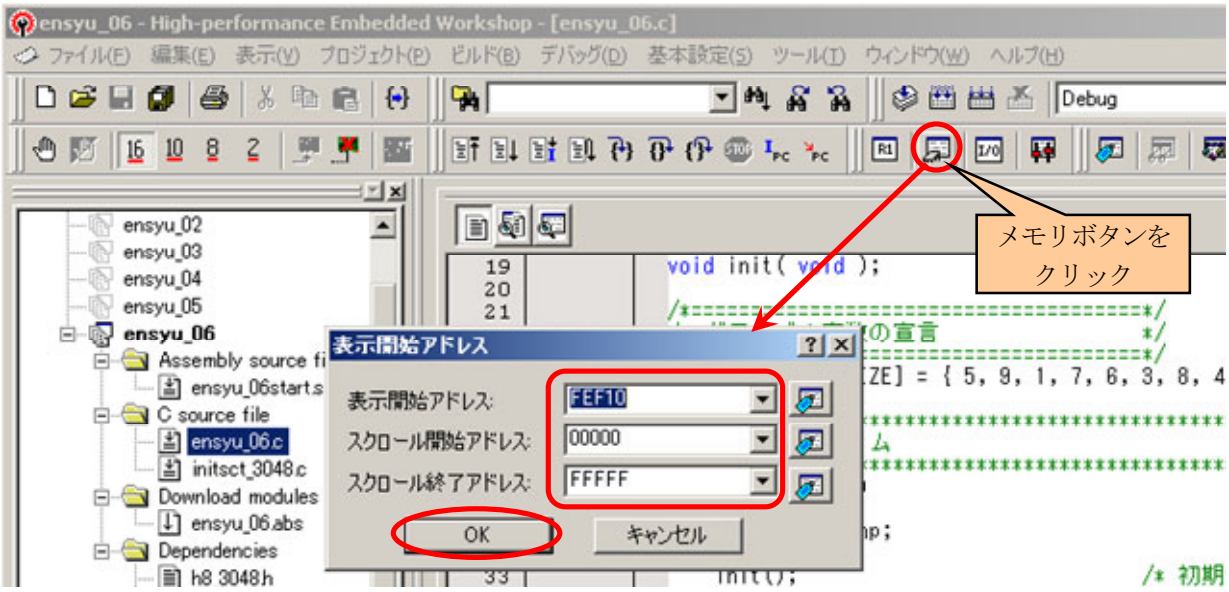
6.8.6 シリアルモニタの操作ポイント

プログラム完成後、data 配列の値がソートされるか、data 配列の値を見ながらステップ実行して確認しましょう。data 配列はグローバル変数なので、ウォッチボタンをクリックして、グローバル変数を表示させます。

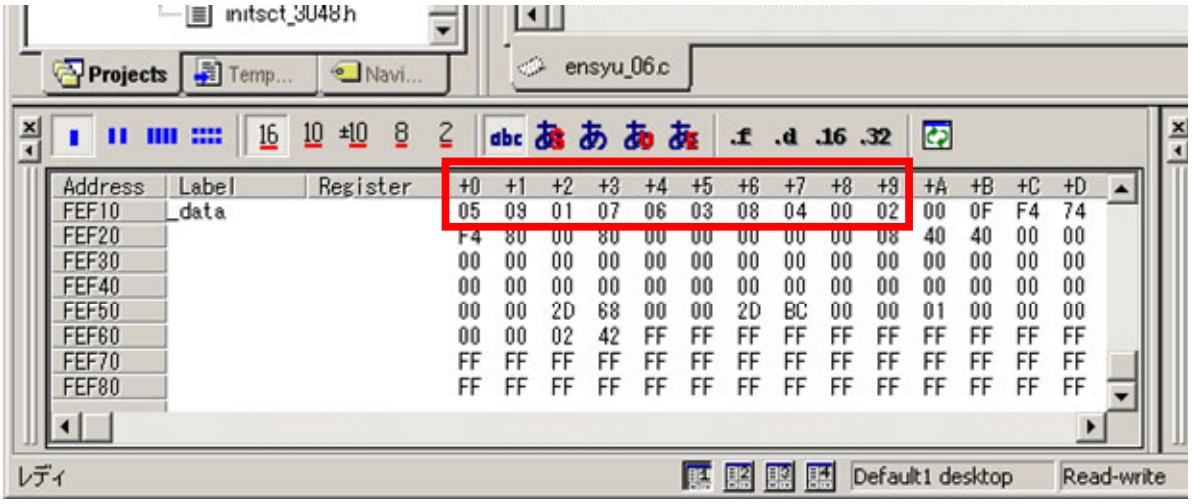


ただ、今回の場合は、ウォッチウィンドウだと見づらいので、メモリウィンドウで見てみましょう。ウォッチウィンドウ内の data 配列の「Value」を見てみます。この値が data 配列のアドレスで、今回は、「FEF10」です。

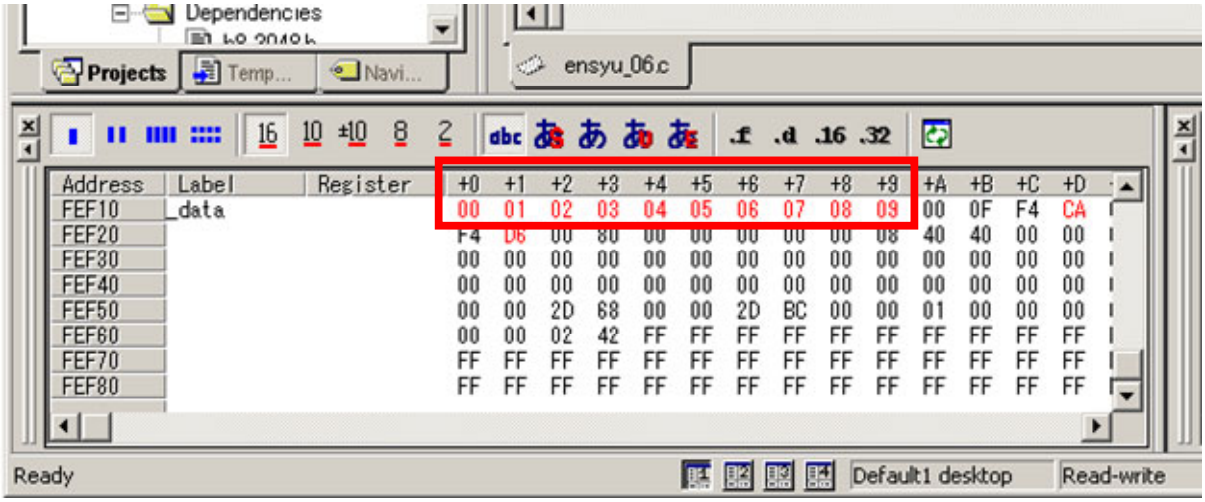
メモリボタンをクリック、表示開始アドレスを「FEF10」にします。スクロール開始アドレス、スクロール終了アドレスは変更しません。



□で囲った部分が、ソート実行前の data 配列の 10 バイト分です。



プログラム実行後、次のようになればプログラム完成です。



6.9 解答例、解説

6.9.1 「ensyu_01.c」

同じ値かどうか比較するのは、C 言語では「= =」(イコールが二つ)です。if 文内であっても、「=」は代入となります。

```
 23 :  void main( void )
 24 :  {
 25 :      unsigned char d;
 26 :
 27 :      init();                             /* 初期化                    */
 28 :
 29 :      while( 1 ) {
 30 :          d = P7DR;
 31 :          if( d == 0x01 ) {    イコール二つ
 32 :              PADR = 0x55;
 33 :          } else {
 34 :              PADR = 0x00;
 35 :          }
 36 :      }
 37 :  }
```

6.9.2 「ensyu_02.c」

ポートは 8bit 幅なので、ポートの値を代入する変数は、符号なし 8bit 幅の「**unsigned char**」型にします。

```
 23 :  void main( void )
 24 :  {
 25 :      unsigned char i;
 26 :
 27 :      init();                             /* 初期化                    */
 28 :
 29 :      while( 1 ) {
 30 :          i = ~P7DR;
 31 :          if( i == 0x01 ) {
 32 :              PADR = 0x55;
 33 :          } else {
 34 :              PADR = 0x00;
 35 :          }
 36 :      }
 37 :  }
```

6.9.3 「ensyu_03.c」

32 行目の値や変数は、次のような型になります。

```
32 :        d = 100 * ad / 1023;
                  ↑   ↑     ↑
              int 型  int 型  int 型
```

計算は、すべて乗除算です。乗除算は、左から順番に計算されます。ad 変数の値が最大の 1023 とします。
まず、

```
        100×1023(ad の値)
```

が計算されます。結果は、102,300 で、それを入れる変数の型は int 型です。int 型の範囲は、-32768～32767 です。計算結果がこの範囲外になったらどうなるのでしょうか。結果は、**不定**です。不定というのは、値がどうなるか分からないということで、正しい値にはなりません。そのため、結果が入る変数 d は、正しい値になりません。
そこで、型変換を行います。値や変数の前に「**カッコ、型、カッコ閉じ**」を追加すると、強制的にその型に変換されます。100 の前に「(long)」を入れます。

```
32 :        d = (long)100 * ad / 1023;
                    ↑     ↑     ↑
                long 型  int 型  int 型
```

long 型と int 型が混ざりました。C 言語では、型が混合した場合、下記のような決まりで演算されます。

・char と unsigned char と short は int
・unsigned short は unsigned int
・float は double

} に変換され、

long double＞double＞unsigned long＞long＞unsigned int＞int

の優先度で型の高い方に変換されて演算されます（ただし、char＜short＝int とする）。

今回は、次のような型に変換され計算されます。

```
32 :        d = (long)100 * ad / 1023;
                    ↑     ↑     ↑
                long 型  long 型  long 型
```

long 型なので、100×1023 は正しく計算されます。次の 102,300÷1023 も正しく計算されます。この結果が変数d に入ります。ad が 1023 の場合、d は 100 となります。

6.9.4 「ensyu_04.c」

31 行は次の順序で評価されます。

```
31 :          if( d & 0x0f == 0x01 ) {
                          ①
                  ②
```

①「0x0f == 0x01」は常に成り立ちません。値は「0」となります。
②「d & 0」も常に成り立ちません。
よって if 文は、d がどのような値でも成り立つことはありません。
本来は、「d を 0x0f でマスクする」、「その後、0x01 かどうか比較する」という順番ですので、カッコを付けて意図する順番にします。

```
23 :  void main( void )
24 :  {
25 :      unsigned char d;
26 :
27 :      init();                          /* 初期化                    */
28 :
29 :      while( 1 ) {
30 :          d = P7DR;
31 :          if( (d & 0x0f) == 0x01 ) {
32 :              PADR = 0x55;
33 :          } else {
34 :              PADR = 0x00;
35 :          }
36 :      }
37 :  }
```

6.9.5 「ensyu_05.c」

今回の演習は、動作を解析しなさいということでした。31 行目がポイントです。

```
31 :          PADR = !!d;
```

「!!」とあります。これは、「!!」という演算子ではなく「!」が二つあるだけです。「!」は論理演算子の否定です。「!」は右から左に評価されます。分かりやすくカッコを付けるなら、次のようになります。

```
31 :          PADR = !(!d);
```

「!値」は次のような計算結果になります。
・ !0＝1　　→ 0 なら結果は 1
・ !0 以外＝0 → 0 以外なら結果は 0
です。要は、0 なら戻り値は 1、0 以外なら戻り値は 0 となります。今回はさらに「!」するので、
d が 0 なら、 !d ＝ !0 ＝ 1、さらに!して、!1 ＝ 0
dが 0 以外なら(1 とします)、!d ＝ !1 ＝ 0、さらに!して、!0 ＝ 1

となります。
すなわち、「!!d」は、
・ d の値が 0 なら、戻り値は 0
・ d の値が 0 以外なら、戻りとは 1
となります。
if 文を使うと次のようになります。

```
if( d != 0 ) d = 1;
PADR = d;
```

これが、「!!」を使うことで、

```
PADR = !!d;
```

と、1 行で済むようになります。

6.9.6 「ensyu_06.c」

昇順にソートプログラム例です。39 行目の「>」を「<」に替えれば、降順ソートになります。
すぐできた場合、40～42 行目の交換部分を関数化したり、配列の参照をポインタにしてプログラムを作ってみましょう。

```
 29 :  void main( void )
 30 :  {
 31 :      int i, j, n, temp;
 32 :
 33 :      init();                              /* 初期化                          */
 34 :
 35 :      /* 配列 data をソートするプログラムを作ってみましょう */
 36 :
 37 :      for(i=0; i<SORT_SIZE-1; i++ ) {
 38 :          for( j=i+1; j<SORT_SIZE; j++ ) {
 39 :              if( data[i] > data[j] ) {   /* 大きければ交換              */
 40 :                  temp    = data[i];
 41 :                  data[i] = data[j];
 42 :                  data[j] = temp;
 43 :              }
 44 :          }
 45 :          /* ソートの途中経過を表示 */
 46 :          PADR = data[i];
 47 :      }
 48 :  }
```

7. 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
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
・(株)オーム社 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 8 月現在の情報です。