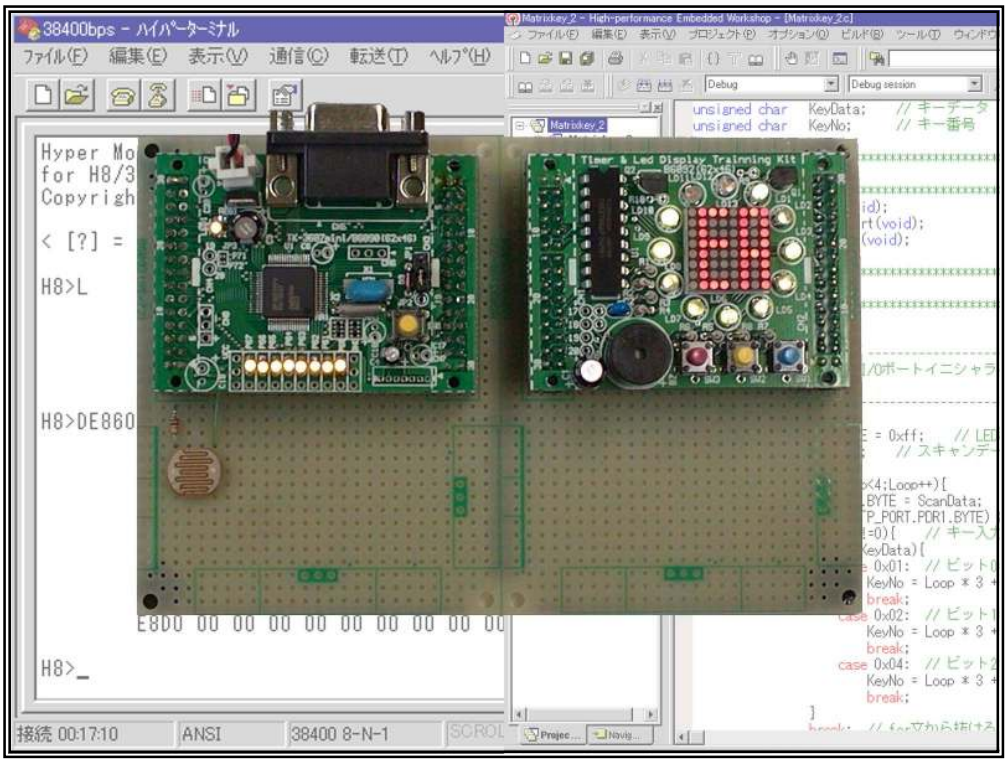
タイマ＆ディスプレイ併用による

TK-3687miniユーザーズマニュアル
C言語版
Ver2.00



(株)東洋リンクス

はじめに

コンピュータというとみなさんは何を思い浮かべますか。きっと，パソコンでしょうね。インターネットと電子メールが普通のものになった今，パソコンは一人一台（もしかしたらそれ以上）の時代になってきました。

ところで，みなさんはコンピュータをいくつ持っていますか。（パソコンではないですよ。コンピュータです。）実は一人 10 台以上持っていても不思議ではありません。というのは，マイクロコンピュータ，つまりマイコンがありとあらゆる電気製品に組み込まれているからです。テレビ，ラジオ，洗濯機，冷蔵庫，電子レンジ，炊飯器，エアコン…。あげればきりがありません。

これだけ身近なマイコンですが，多くの人にとって今なおマイコンは遠い存在です。マイコンを使っていても，その仕組みを理解している人はそれほど多くはないでしょう。

もっとも，これは当然のことかもしれません。マイコンはすでに空気のようなもので，なくてはならないものですが，普段は意識されない存在だからです。でも，空気について調べると非常に興味深い事実があるのと同じように，マイコンもその仕組みを理解すると非常に面白いものであることがわかります。

TK-3687mini は，そんなマイコンの面白さを理解したい，という人のために用意されました。マイコンを理解する早道は，とにかくプログラムを作って動かしてみる，という事につきますが，そのための道具としてきっとお役に立つことでしょう。

ここで，TK-3687mini で採用されている H8/3687 というワンチップマイコンについて少しふれておきましょう。H8/3687 は日立によって開発が始まった H8 シリーズの一員です。H8 シリーズは規模や用途に応じて多くのシリーズがありますが，H8/3687 はシステムの小型化を目指してそのほとんどをワンチップ化した‘H8/300H Tiny’シリーズに属します。‘H8/300H Tiny’シリーズの標準品は H8/3664 で多くのボードメーカによってマイコンボードが作られました。TK-3687mini で採用している H8/3687 はその機能強化版にあたります。H8 シリーズは現在，日立と三菱が共同で設立したルネサステクノロジが製造・販売しています。

このマニュアルでは，マイコンにはじめて触れる人に向けて TK-3687mini の基本的な使い方を説明しています。細かい理屈はわからなくても，このとおりやっていけばとりあえず動かすことができるようになっています。細かい理屈もちょっとだけ書いていますので興味がわいたら読んでみてください。みなさんのマイコン技術がさらにステップアップする入口になれば幸いです。

マニュアルについて

ルネサステクノロジのサイト（http://www.renesas.com/jpn/）から，マニュアルのダウンロードサイト（http://www.renesas.com/jpn/products/mpumcu/16bit/tiny/tiny_manual.html）に移動し，次のマニュアルをダウンロードして下さい。技術文書のため読みこなすのはかなりたいへんですが，欠かすことができない資料です。

「H8/3687 グループ　ハードウェアマニュアル」
「H8/300H シリーズ　プログラミングマニュアル」

あとは HEW と一緒にパソコンにコピーされるマニュアルが，アセンブラや C の言語仕様を説明しています。これも読むのはたいへんですが，やはり欠かすことができません。

目次

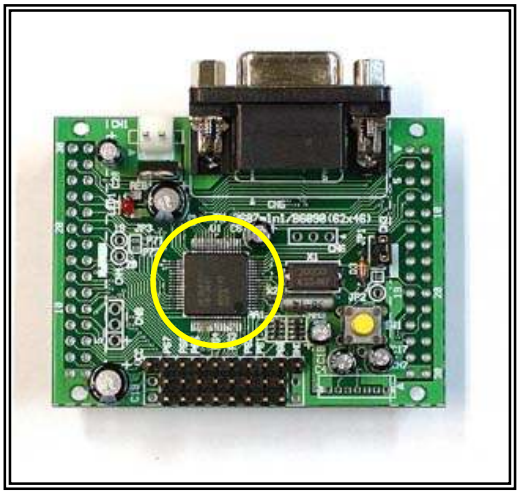
第1章 ワンチップマイコン入門

1. ワンチップマイコンとは何か
2. H8/3687 の構成

まずはマイコンとTK-3687mini について見てみましょう。この章は「マイコンって，こんなもんか」という感じで気楽に読んでもらえればよいです。あとは動かしていくうちにわかってくるでしょう。

1. ワンチップマイコンとは何か

まずは TK-3687mini を箱から出して眺めてみましょう。基板の中央にLSI（H8/3687）が 1 個のっていますね。まわりにいろいろと部品がのっていますが，これらは全部おまけみたいなもので，実はこの LSI がマイコンそのものです。この中にマイコンの機能の全てがつまっています。

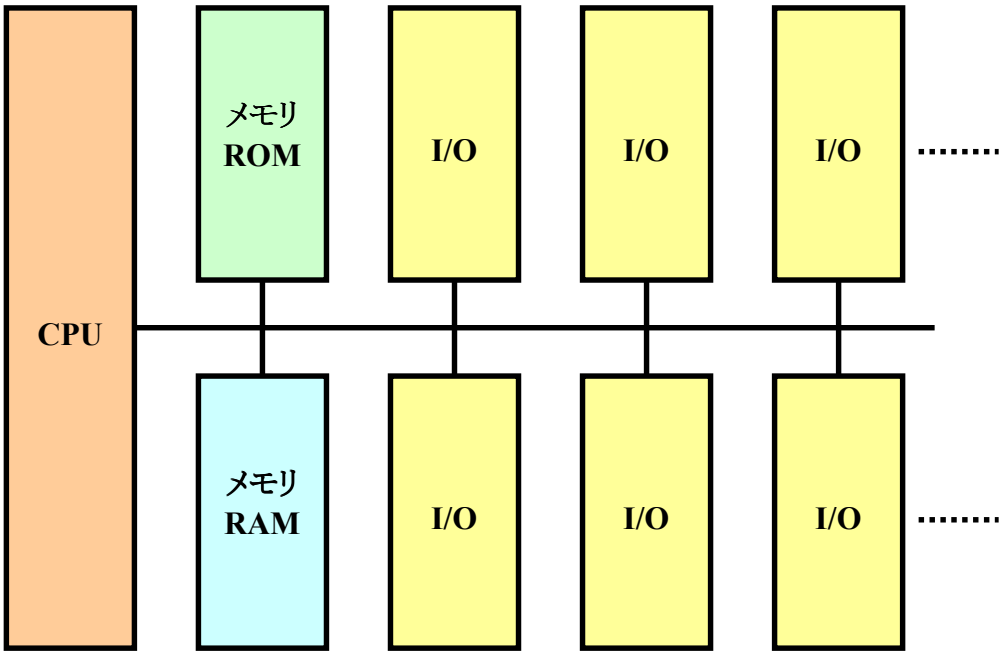


■ マイコンの 3 要素

どんなマイコンでも次の基本的な 3 つの要素からできています。もちろん H8/3687 も例外ではありません。

- CPU（Central Processing Unit：中央演算装置）
- メモリ（記憶装置）
- I/O（Input/Output：入出力装置）

CPU は演算や判断の処理を行ない，データの流れをコントロールするコンピュータの頭脳です。そして，その CPU を動作させるためのプログラムやデータを記憶するのがメモリです。外部から信号を入力したり外部機器をコントロールするのが I/O です。基本的には右のような構成になります。

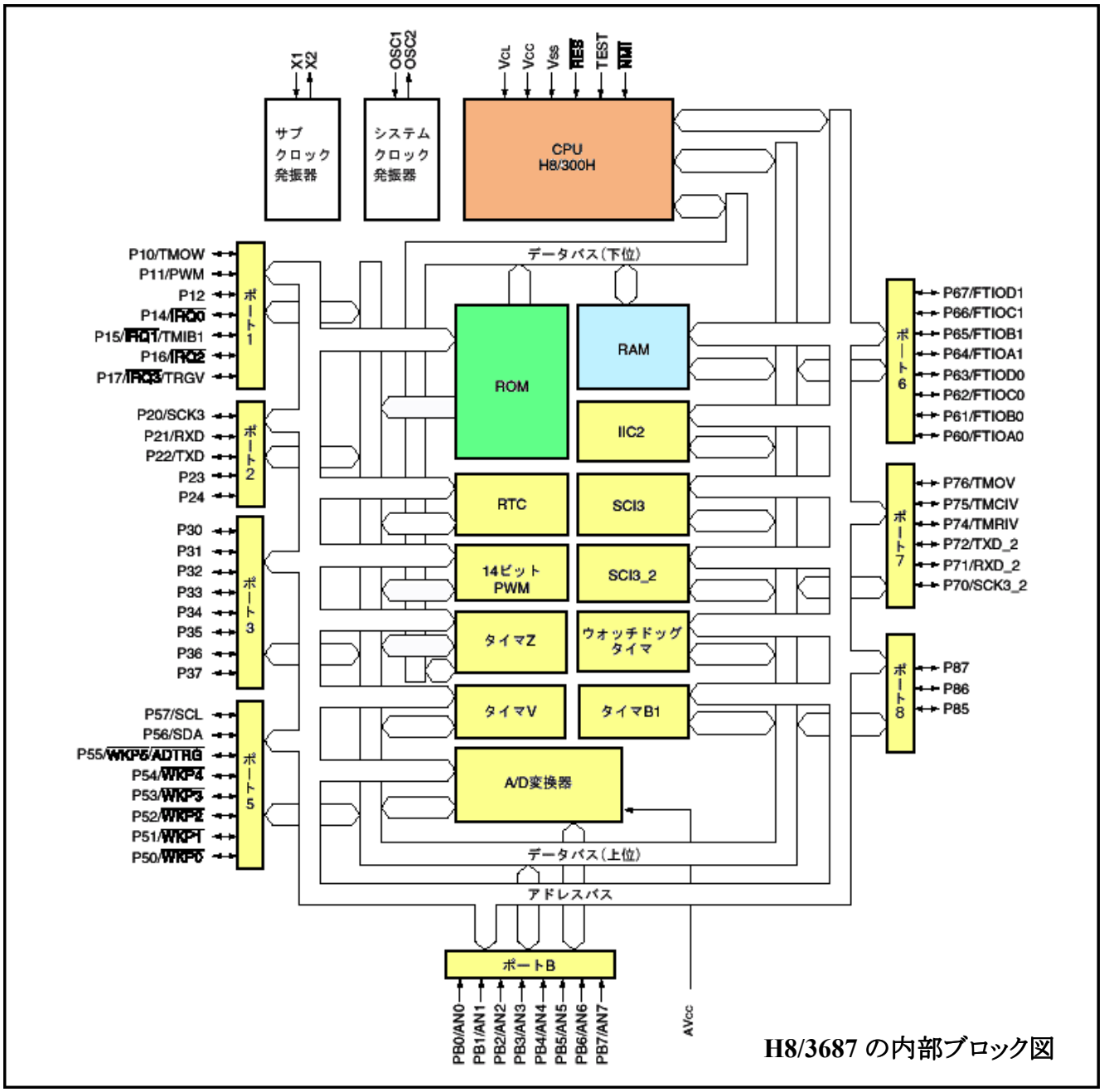


以前はこの 3 要素は別々の LSI で，それぞれを配線する必要がありました。しかし，最近はこれら全てが一つの LSI に集積されるようになりました。これをワンチップマイコンと呼んでいます。

2. H8/3687 の構成

■ H8/3687 の内部ブロック

TK-3687mini で使っている H8/3687 もワンチップマイコンです。H8/3687 に何が内蔵されているか次の図をご覧下さい。前のページの図とどのように対応するか色分けしてみました。マイコンの 3 要素の全てが入っていますね。



H8/3687 の内部ブロック図

H8/3687 には，H8/300H という CPU が内蔵されています。CPU は，メモリから順番に命令を取り出し，その命令に従って計算（演算）したり，さらにメモリに対してデータをリード/ライトしたり，I/O に対してデータをリード/ライトしたりします。

■ H8/3687 のレジスタ

H8/300H の内部には，一時的にデータをセットするために使う汎用レジスタ（ER0～ER7）と，CPU の制御のために使うコントロールレジスタ（PC と CCR）があります。レジスタはメモみたいなもので，ちょっとデータを記録しておく，というような感じで使います。これからこのマニュアルで TK-3687mini（H8/3687）について調べていきますが，レジスタはよく出てくるため，ここでまとめて取り上げます。

• 汎用レジスタ

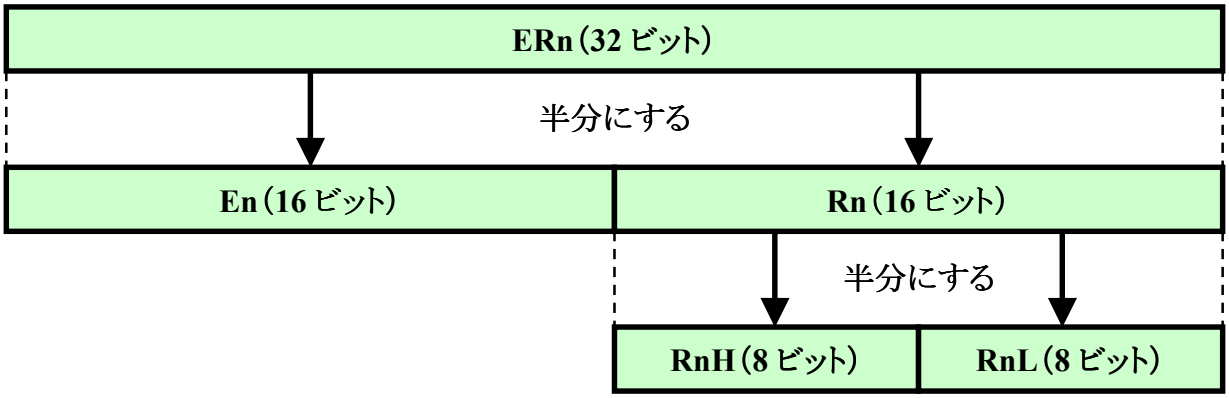
H8/300H は 32 ビット長の汎用レジスタを 8 本持っています。それぞれ，ER0～ER7 という名前がつけられています。

この 32 ビットレジスタを上位 16 ビットと下位 16 ビットにわけて，それぞれを 16 ビットレジスタとして使うことができます。E0～E7，R0～R7 という名前がつけられていて，16 ビットレジスタを最大 16 本使うことができます。

さらに，R0～R7 については上位 8 ビットと下位 8 ビットにわけて，それぞれを 8 ビットレジスタとしても使うことができます。R0H～R7H，R0L～R7L という名前がつけられていて，8 ビットレジスタを最大 16 本使うことができます。

これらの汎用レジスタは「汎用」と名付けられているとおり，全て同じ機能を持っています。つまり，ER0 でできることは ER1～ER7 でもできますし，R0L でできることは R0H～R7H，R1L～R7L でもできます。また，各レジスタは独立して 32，16，8 ビットレジスタとして使うことができます。

汎用レジスタの構成について図で示すと次のようになります。（n=0～7）



汎用レジスタは全て同じ機能を持っているのですが，ER7 だけは汎用レジスタとしての機能にプラスして，スタックポインタとしての機能も持っています。

- **コントロールレジスタ**

H8/300H には 2 つのコントロールレジスタがあります。1 つはプログラムカウンタ（PC）です。PC は 24 ビットのレジスタで，CPU が次に実行する命令のアドレスを示しています。H8/3687 は PC の下位 16 ビットを使用しています。

プログラムカウンタ（24 ビット）

もう 1 つはコンディションコードレジスタ（CCR）です。CCR は 8 ビットのレジスタで，それぞれのビットが CPU の内部状態を表しています。演算結果が 0 になったとか，マイナスになったとか，キャリやボローやオーバフローが発生したという情報がセットされます。おもに分岐命令で使われます。どんな種類があるのか下記に示します。

コンディションコードレジスタ（8 ビット）							
I	**UI**	**H**	**U**	**N**	**Z**	**V**	**C**

ビット	ビット名称	機能
ビット 7	I	**割り込みマスクビット** このビットが‘1’にセットされると割り込み要求がマスクされます。
ビット 6	UI	**ユーザビット** ユーザが自由に定義，設定できるビットです。
ビット 5	H	**ハーフキャリフラグ** 8 ビット算術演算のときは，ビット 3 にキャリが生じたとき‘1’，生じなかったとき‘0’になります。16 ビット算術演算のときは，ビット 11 にキャリが生じたとき‘1’，生じなかったとき‘0’になります。32 ビット算術演算のときは，ビット 27 にキャリが生じたとき‘1’，生じなかったとき‘0’になります。 このフラグは 10 進補正命令（DAA，DAS）のときに使用されます。
ビット 4	U	**ユーザビット** ユーザが自由に定義，設定できるビットです。
ビット 3	N	**ネガティブフラグ** データの最上位ビットを符号ビットと見なし，最上位ビットの値を格納します。
ビット 2	Z	**ゼロフラグ** データがゼロのときに‘1’，ゼロ以外のときに‘0’になります。
ビット 1	V	**オーバフローフラグ** 算術演算命令の実行によりオーバフローが発生したときに‘1’，それ以外のときは‘0’になります。
ビット 0	C	**キャリフラグ** 演算の結果，キャリが生じたときに‘1’，生じなかったときに‘0’になります。キャリには①加算結果のキャリ，②減算結果のボロー，③シフト/ローテート命令のキャリがあります。

ダウンロードした「H8/300H シリーズ プログラミングマニュアル」の各命令のページに，「●コンディションコード」という項目があります。その命令を実行した結果，CCR がどのように変化するか説明されています。

「？？？…」という感じでしょうか。でも心配には及びません。このマニュアルでも関係するところで説明しますし，プログラムを作ったり動かしたりしていくうちにだんだんわかってくると思います。「習うより慣れろ」と言いますしね。

■ H8/3687 のメモリマップ

メモリはプログラムも含めたデータを記憶する部分です。もっとも見た目は単なる数字にしか見えませんが…。CPU からの命令で以前に記憶させたデータを読んだり(リード)，新たにデータを記憶させる(ライト)ことができます。例えば，CPU がプログラムを実行する時は，メモリからデータをリードして，そのデータがどんな命令か解析して実行します。

メモリには 1 バイトごとに 0 から始まるアドレスがつけられています。アドレスというぐらいなので，考え方としては町の住所のようなものです。広い日本の特定の家に手紙を届けるために住所をきちんと指定するのと同じように，メモリをリード/ライトする時には必ずアドレスを指定しなければなりません。このとき使う表現が「メモリの～番地」というフレーズです(やっぱり住所ですね)。メモリの場合は 16 進数で表します。例えば，「EA00 番地から実行する」という感じです。

さて，H8/3687 には ROM と RAM という 2 種類のメモリが内蔵されています。ROM とは Read Only Memory の略で，電源をオフしても消えることはなく，特別な方法でしか書き換えることができないメモリです。通常はリードするだけです。H8/3687 に内蔵されている ROM はフラッシュメモリで，プログラムや変更する必要のないデータはここに書き込みます。レジスタをメモとすれば，ROM は本ですね。出荷時にはハイパーH8 というプログラムが書き込まれています。なお，フラッシュメモリを書き換えるためには‘FDT’という道具を使います(FDT については「TK-3687mini 組み立て手順書」をご覧下さい)。

RAM は Random Access Memory の略で，いつでも自由にリード/ライトすることができます。その代わり，電源をオフすると全て忘れてしまいます。というわけで，普通はプログラム中で変更するデータをここに記憶させておきます。もちろん，RAM にプログラムを書き込んでも，そのプログラムを実行することはできます(あとでででてくるハイパーH8 では RAM にプログラムをセットします)。ただ，電源をオフすると，きれいさっぱり忘れてしまい，思い出すことは不可能です。レジスタがメモ，ROM が本とすれば，RAM はノートです。作業にあわせてそのつど書いたり消したりします。ただ，電源をオフするとまるごとごみ箱に捨てて，電源をオンするたびに新しいまっさらなノートを準備する，という感じですが。

H8/3687 のメモリの広さは 64K バイト(アドレスは 0 番地から FFFF 番地まで)あります。この中に ROM や RAM，さらには I/O が割り当てられています。メモリマップは右のとおりです。

0000 番地 DFFF 番地	ROM/フラッシュメモリ (56K バイト)
E000 番地 E7FF 番地	未使用
E800 番地 EFFF 番地	RAM (2K バイト)
F000 番地 F6FF 番地	未使用
F700 番地 F77F 番地	I/O レジスタ
F780 番地 FB7F 番地	RAM (1K バイト) フラッシュメモリ書換え用 ワークエリアのため使用不可
FB80 番地 FF7F 番地	RAM (1K バイト)
FF80 番地 FFFF 番地	I/O レジスタ

10 進数と 2 進数，16 進数

私たちが日常使っているのは 10 進数です。0～9 の 10 個の数字を使って数を表します。

ところが，コンピュータの世界，特にマイコンの世界では 2 進数や 16 進数が普通に使われています。2 進数は 0 と 1 の 2 個の数字で数を表す方法，16 進数は 0～9 と A～F の 16 個の数字で数を表す方法です。

では，ちょっと比較してみましょう。

10 進数	2 進数	16 進数
0	00000000	00
1	00000001	01
2	00000010	02
3	00000011	03
4	00000100	04
5	00000101	05
6	00000110	06
7	00000111	07
8	00001000	08
9	00001001	09
10	00001010	0A
11	00001011	0B
12	00001100	0C
13	00001101	0D
14	00001110	0E
15	00001111	0F
16	00010000	10

ところで，2 進数と 16 進数を比較するとおもしろいことに気づきませんか？それは，2 進数を 4 桁づつ区切ると 16 進数の 1 桁に相当する，ということです。

（例）00001010 = 0A

実はこれがマイコンで 16 進数が使われている理由です。よく言われているようにデジタルの世界は 0 か 1 です。ご多分にもれずマイコンの世界も 0 か 1 です。なので，本当は 2 進数がぴったりなのです。でも，2 進数は桁が長すぎる，それなら 4 桁づつまとめて 16 進数で表してしまおう，ということになりました。

ちなみに 10 進数，2 進数，16 進数の表し方はいろいろですが，このマニュアルでは次のようにあらわすことにします。（10 進数の 10 をどのように表すか）

10 進数　：（例）10

2 進数　：（例）B'00001010

16 進数　：（例）H'0A，0x0A または 0Ah

ビット，バイト，ワード，ロングワード

マイコンの世界はデジタルの世界なので，‘0’か‘1’の世界です。というわけで，2 進数 1 桁が最小単位となり，これをビットと呼びます。

さて，メモリがそうですが，マイコンでは 1 データを 8 ビット単位で扱うことが多いです。そこで，8 ビットで構成される単位をバイトと呼びます。16 進数 2 桁になります。（R0H～R7H，R0L～R7L レジスタ）

さらに，2 バイト単位でまとめることもよくあります。で，これをワードという単位にします。16 進数 4 桁ですね。（E0～E7，R0～R7 レジスタ）

そして最後に，2 ワード単位（4 バイト単位）にしたものをロングワードと呼びます。16 進数 8 桁になります。（ER0～ER7 レジスタ）

まとめると，

1 ロングワード＝2 ワード＝4 バイト＝32 ビット

となります。

メモリマップとは

CPU はアクセスできるアドレスの範囲が決まっています。例えば，H8/3687 の場合は，0000～FFFF までです。この中に ROM や RAM を割付けていきます。

さて，どこに何が割付けられているか示した図をメモリマップと呼びます。前のページは H8/3687 のメモリマップになるわけです。

ところで，メモリマップといいながら I/O も割付けられていました。H8 の場合 I/O もメモリのように扱っています。データをリード/ライトするという点では，メモリも I/O もかわりないですしね。このような割付け方をメモリマップド I/O と呼びます。

対して，I/O のための専用のマップを準備する CPU もあります。この場合，メモリマップではなく I/O マップといいます。このような割り付け方を I/O マップド I/O とか，アイソレーテッド I/O と呼びます。

これはどちらが優れているというわけではありません。単に思想の違いです。

第2章 マイコンボードを動かしてみよう

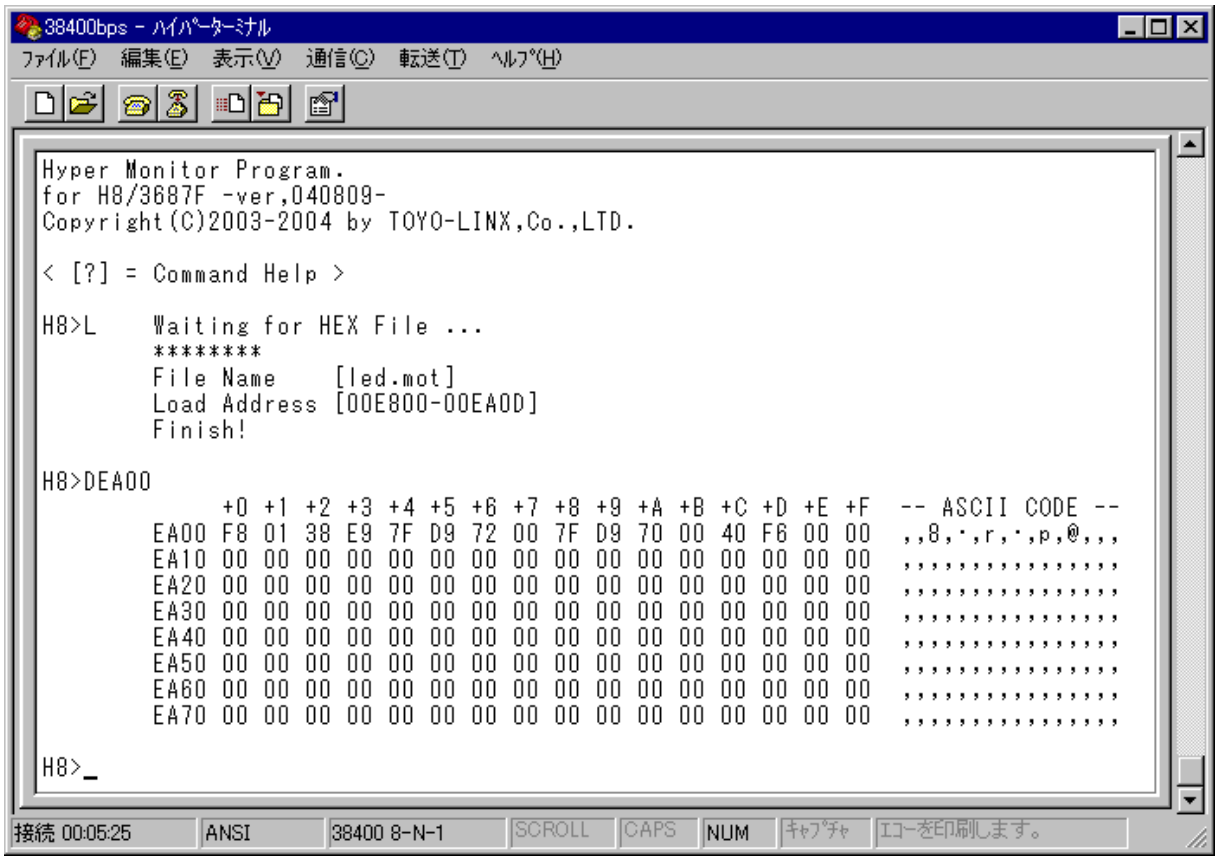
1. モニタプログラムとは何か
2. モニタプログラム「ハイパーH8」を詰まってみる
3. デモプログラムの実行

では，早速 TK-3687mini を動かしてみましょう。とはいっても，TK-3687mini をみると分かるように，電源をオンしてもなんだかよく分かりません。というわけで，マイコンの中身をのぞく道具を準備して，それを動かしてみましょう。その道具の名前は‘ハイパーH8’です。

1. モニタプログラムとは何か

モニタプログラムとは何でしょうか。モニタ（monitor）には監視する，という意味があります。マイコンでいうモニタというプログラムは，マイコンの中身を監視するプログラムです。レジスタの値はどうなっているでしょうか。ROM や RAM にどんなデータが入っているでしょうか。I/O にどんなデータが入出力されているでしょうか。モニタが搭載されていれば，このようなマイコンの中身の情報を見ることができます。また，パソコンで作ったプログラムをマイコンに送り込む（ロード）こともできます。さらに，プログラムの動作そのものも制御することができ，ロードしたプログラムを実行したり，途中で止めたりもできます。いろいろな機能を持つモニタですが，C 言語ではロード（L）と実行（G）のみを使います。

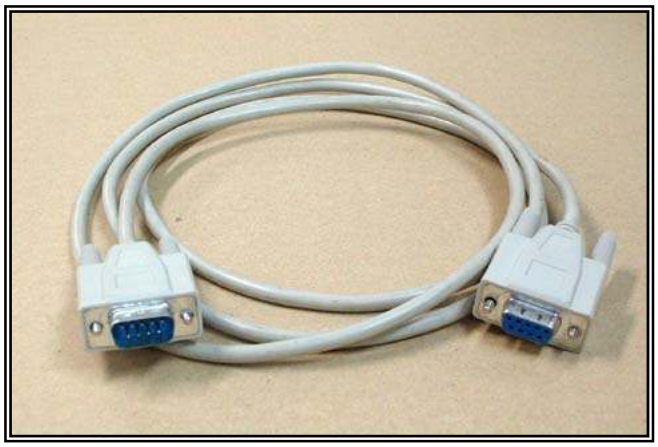
ハイパーH8 は Windows シリーズに標準で搭載されているターミナルソフト，‘ハイパーターミナル’を使用した簡易モニタです。お手持ちのパソコンと TK-3687mini のシリアルポートを RS-232C ケーブルで接続することで，簡単なモニタ環境を作ることができます。TK-3687mini にはあらかじめハイパーH8 が書き込まれていて，電源オンですぐにマイコンの中身を見ることができます。

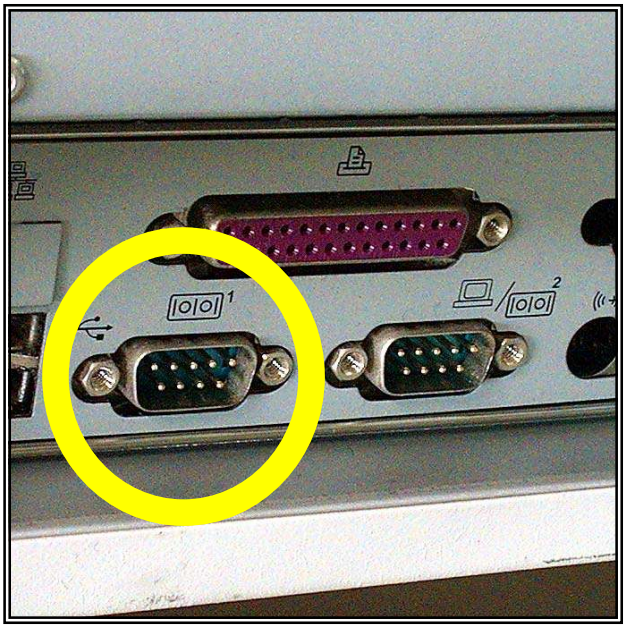


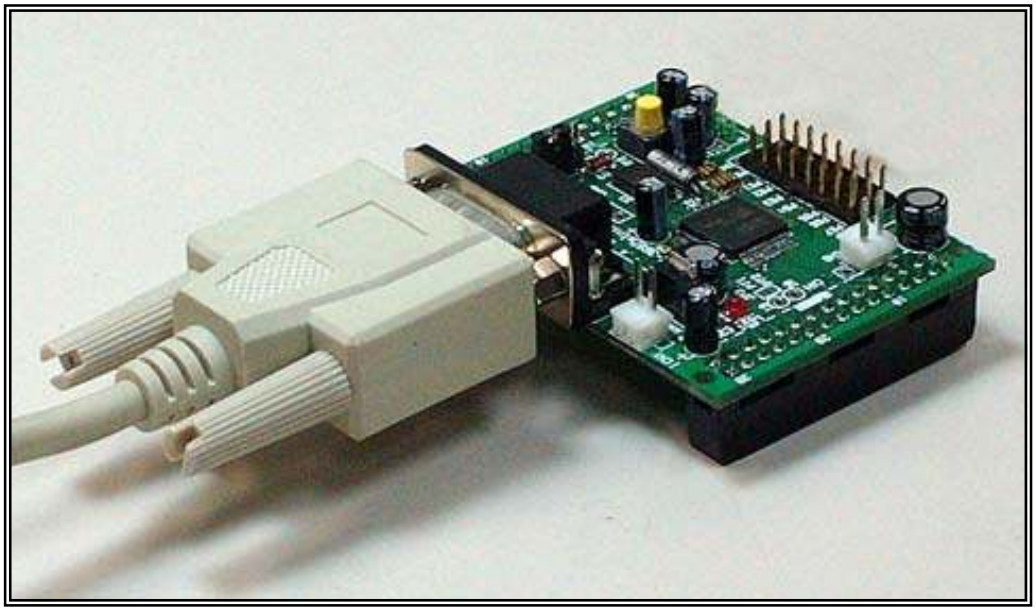
2. モニタプログラム「ハイパーH8」を使ってみる

■ パソコンと TK-3687mini をつなぐ

まず TK-3687mini とパソコンをつなぎます。D-Sub・9pin（オス）－9pin（メス）ストレートケーブル（写真上）でメス側をパソコンの COM ポート（写真中）へ，オス側を TK-3687mini の CN5（写真下）へ接続します。しっかりとさし込み，ケーブルにネジがついている場合はネジをしめて固定しましょう。写真のように COM ポートがいくつか空いている場合は1番につなげてください。なお，この時はまだ TK-3687mini の電源は入れないでください。







■　ハイパーターミナルの設定

それでは，通信ソフト‘ハイパーターミナル’を起動して，TK-3687mini と通信するためのセッティングを行いましょう。

まずハイパーターミナルを起動します。ハイパーターミナルは，

スタート → プログラム → アクセサリ → 通信 → ハイパーターミナル → ハイパーターミナル

から起動できます。Windows のバージョンによっては，

スタート → プログラム → アクセサリ → 通信 → ハイパーターミナル

から起動する場合もあります。もし，スタートメニューにない場合は，

スタート → 検索 → ファイルやフォルダ

で‘hypertrm．exe’を検索してください。ハイパーターミナルを起動したら，出てくるダイアログウィンドウにしたがって設定していきましょう。

①　接続の設定（1）

名前とアイコンを設定します。右の画面では，名前は接続速度がわかるように「38400bps」としました。名前を入力してアイコンを選択したら OK をクリックします。

② 接続の設定(2)

接続方法(N):のプルダウンメニューから，ケーブルを接続した COM ポート(右の画面では COM1)を選択して OK をクリックします。

③ COM1 のプロパティ

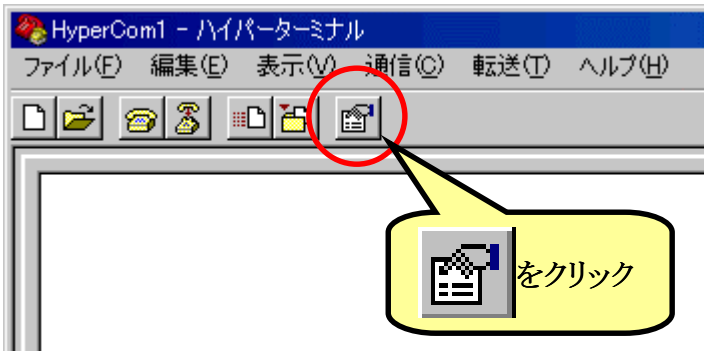
各項目を次のように設定します。

ﾋﾞｯﾄ/秒(B) :38400
ﾃﾞｰﾀﾋﾞｯﾄ(D) :8
ﾊﾟﾘﾃｨ(P) :なし
ｽﾄｯﾌﾟﾋﾞｯﾄ(S) :1
ﾌﾛｰ制御(F) :Xon/Xoff

設定し終えたら OK をクリックします。

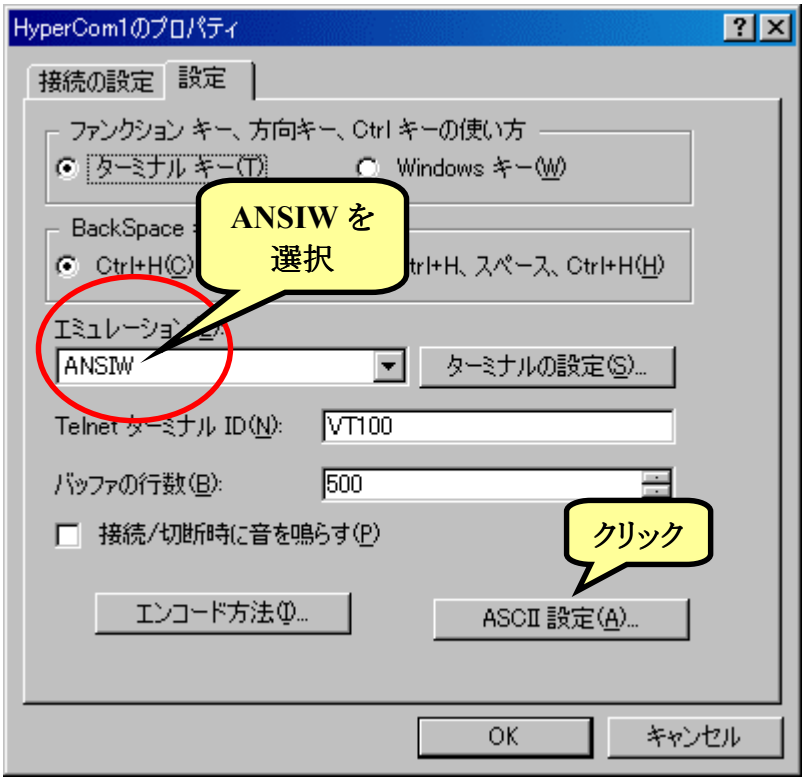
④ プロパティアイコンをクリック

ターミナル画面に切り替わりますので，ツールバーのプロパティアイコンをクリックしてプロパティダイアログを開きます。



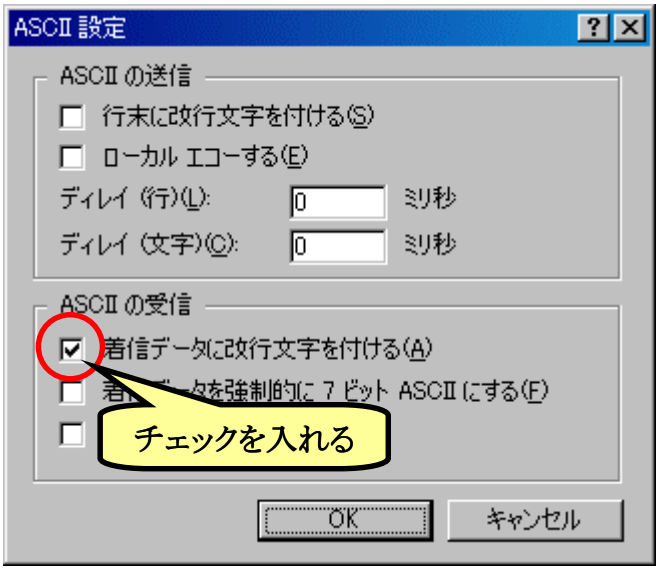
⑤ プロパティ

‘設定’タブをクリックして‘エミュレーション(E):’のプルダウンメニューから‘ANSIW’を選択し，ASCII 設定(A)...をクリックします。



⑥ ASCII 設定

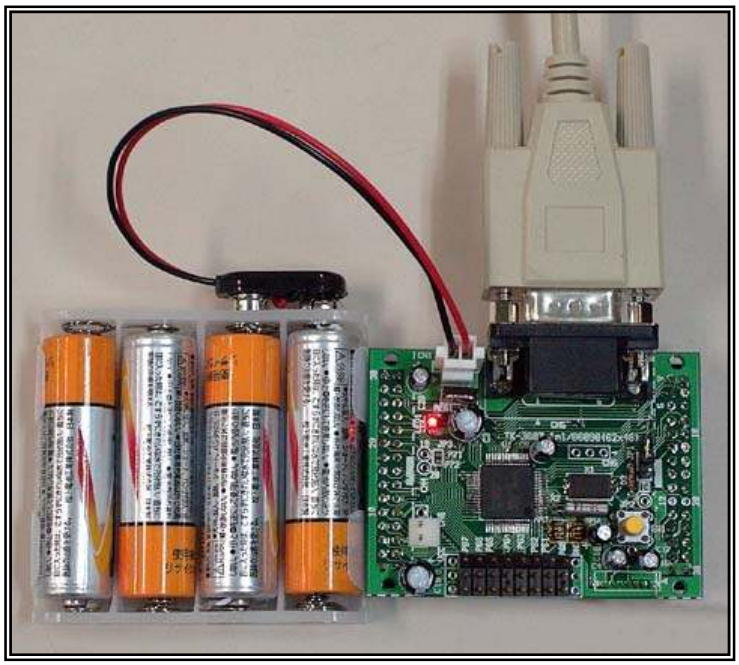
‘ASCII の受信’の中の‘着信データに改行文字を付ける(A)’のチェックを入れてOKをクリックします。するとプロパティダイアログに戻りますので，もう一度OKをクリックしてターミナル画面に戻ります。

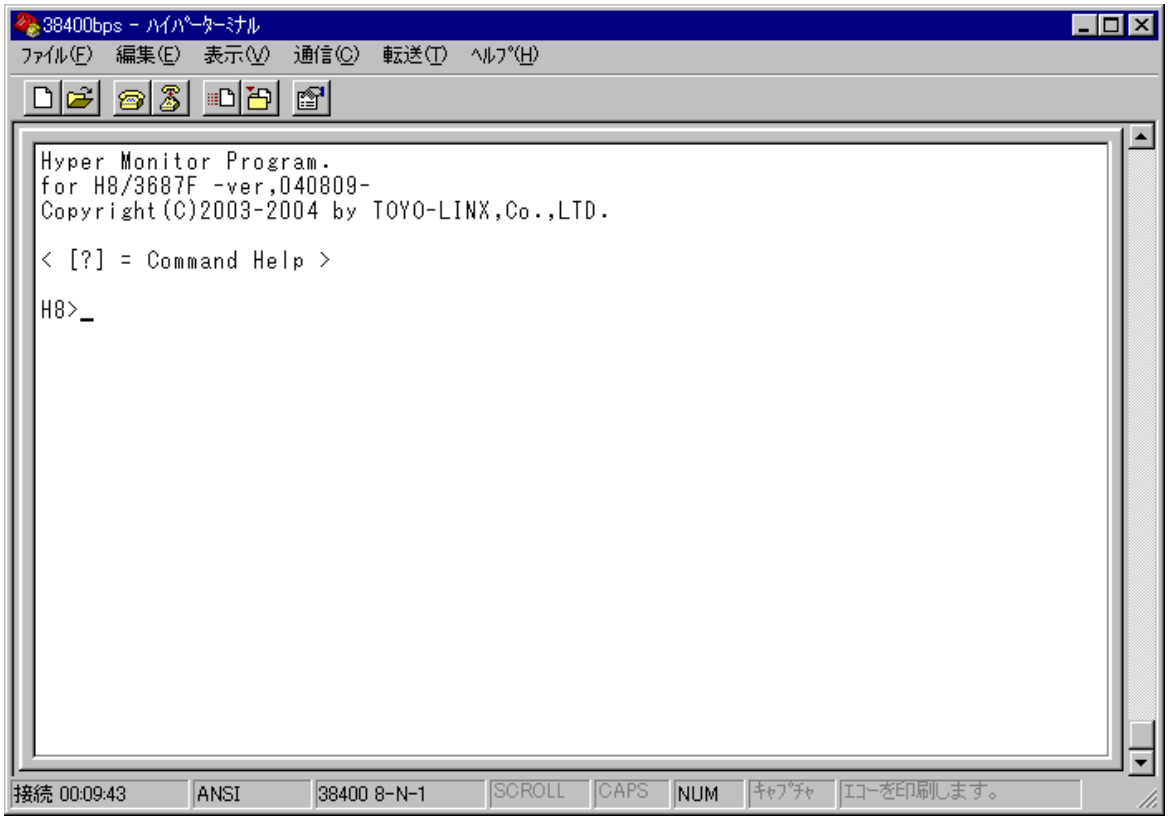


◆

これで設定は終了です。それでは電源をオンしてみましょう。ちゃんと動くでしょうか。

電源（例えば電池 4 本=6V）を TK-3687mini の CN1 につなぎます（右写真参照）。電源をオンするとハイパーターミナルの画面に次のように表示されます。





ここまでくればマイコンの中身を自由に見ることができます。次の章では手始めにあらかじめ TK-3687mini に書き込まれているデモプログラムを実行してみましょう。

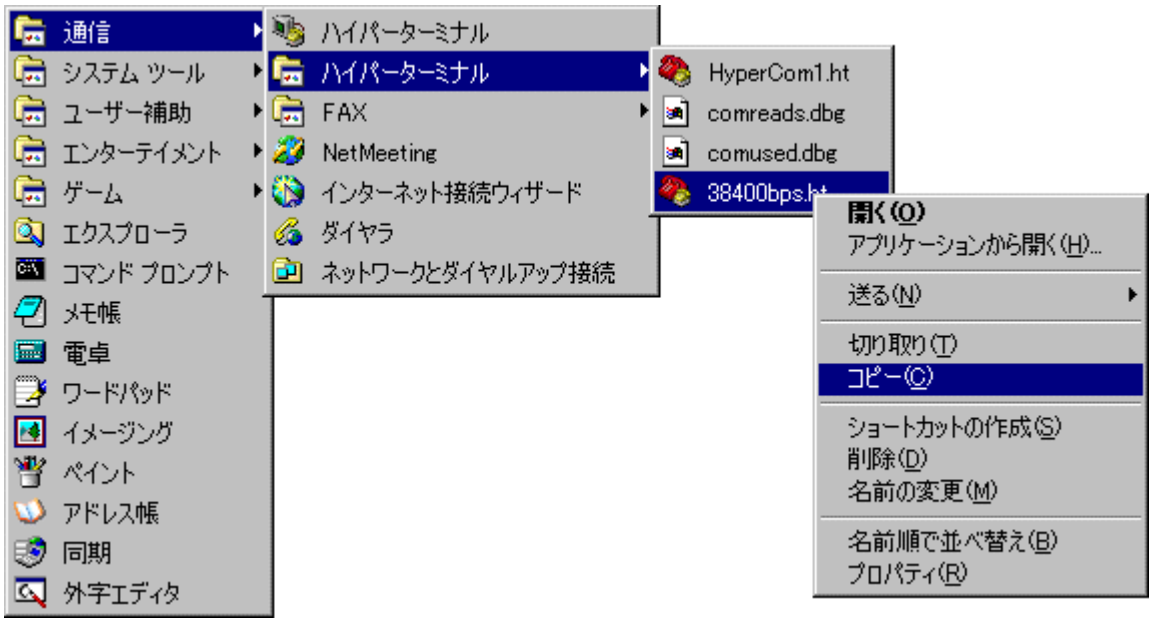
でも，その前に…（次のページを見てください）

ハイパーターミナルを起動するたびに毎回毎回設定を繰り返していたのでは面倒ですね。そこで，ハイパーターミナルの設定を保存しておきましょう。メニューバーの「ファイル(F)」→「上書き保存(S)」を選択して保存して下さい。

さらに，この設定のハイパーターミナルをすぐに呼び出せるように，デスクトップにショートカットを作成しましょう。スタートメニューから，

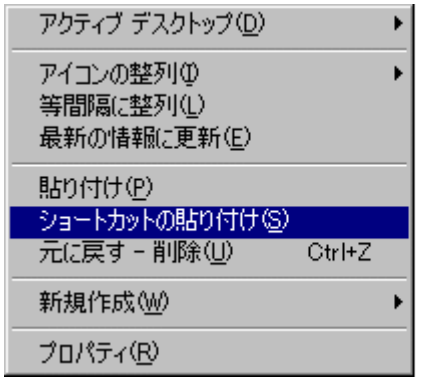
スタート → ﾌﾟﾛｸﾞﾗﾑ → ｱｸｾｻﾘ → 通信 → ﾊｲﾊﾟｰﾀｰﾐﾅﾙ → 38400bps.ht

までカーソルを進め，右クリックします。プルダウンメニューの中の「コピー(C)」を選択してください。



デスクトップで
右クリックすると…

デスクトップで再度右クリックし，「ショートカットの貼り付け(S)」を選択してショートカットを作成します。



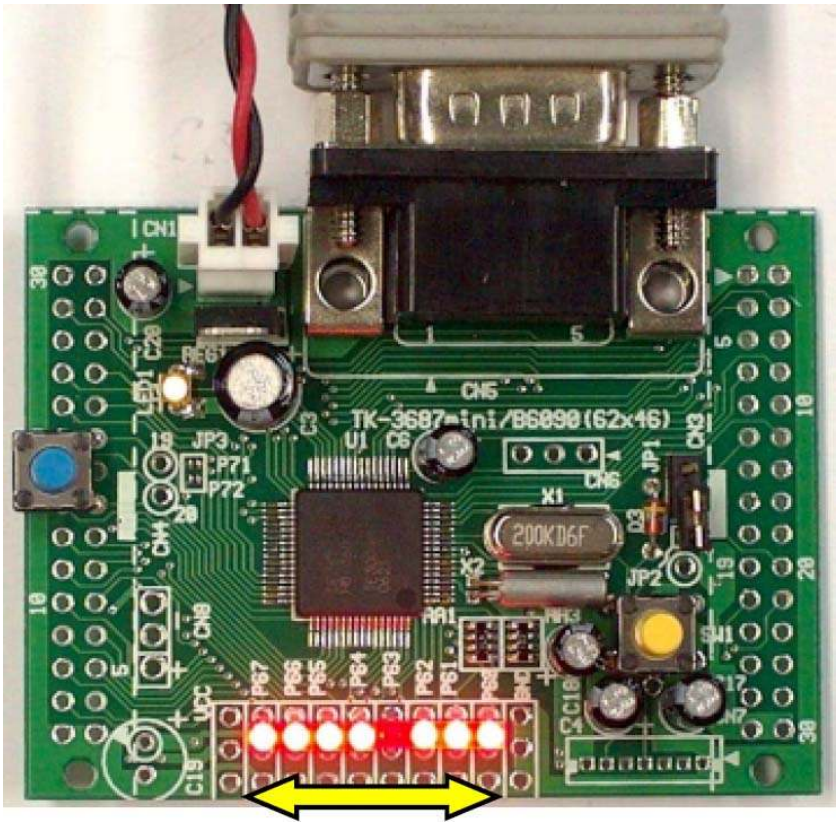
なお，ここで示した方法は Windows2000 の場合です。この方法でショートカットが作成できない場合は，エクスプローラやファイルの検索を使ってデスクトップにショートカットを作ってください。

3. デモプログラムの実行

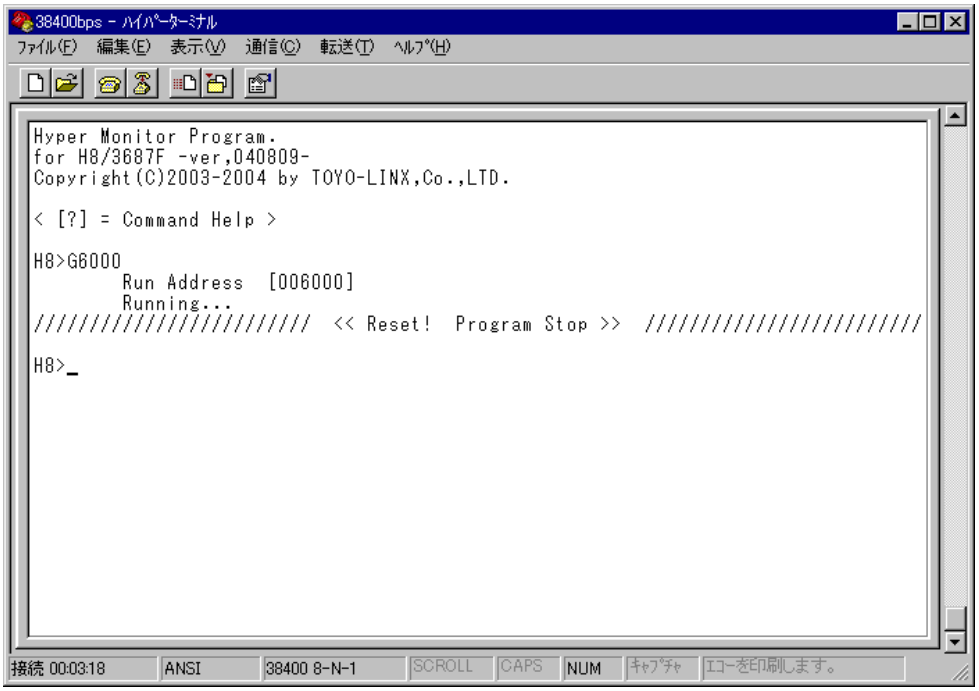
プログラムの作り方はあとで説明するとして，この章ではとにかくプログラムを動かしてみましょう。TK-3687mini とパソコンは RS-232C ケーブルでつながっていますか。前の章の最後でデスクトップに「38400bps」のショートカットを作りました。ハイパーターミナルを終了してしまった人はショートカットをダブルクリックしてハイパーターミナルを起動して下さい。TK-3687mini の電源をオンして右のとおりハイパーH8 の最初の画面が表示されたら準備 OK です。

TK-3687mini にはデモ用に，また，基板チェックのために，いくつかのプログラムが ROM にあらかじめ書き込まれています。そのうちの一つを動かしてみましょう。ハイパーターミナルから‘G6000’と入力して‘Enter’キーを押します。すると，あっけないほど簡単にプログラムが動き出します。

P60～67 の LED が順番に点滅します。



TK-3687mini のリセットスイッチ（SW1）を押すと，実行中のプログラムは停止して，ハイパー H8 は下図のように入力待ちの状態になります。



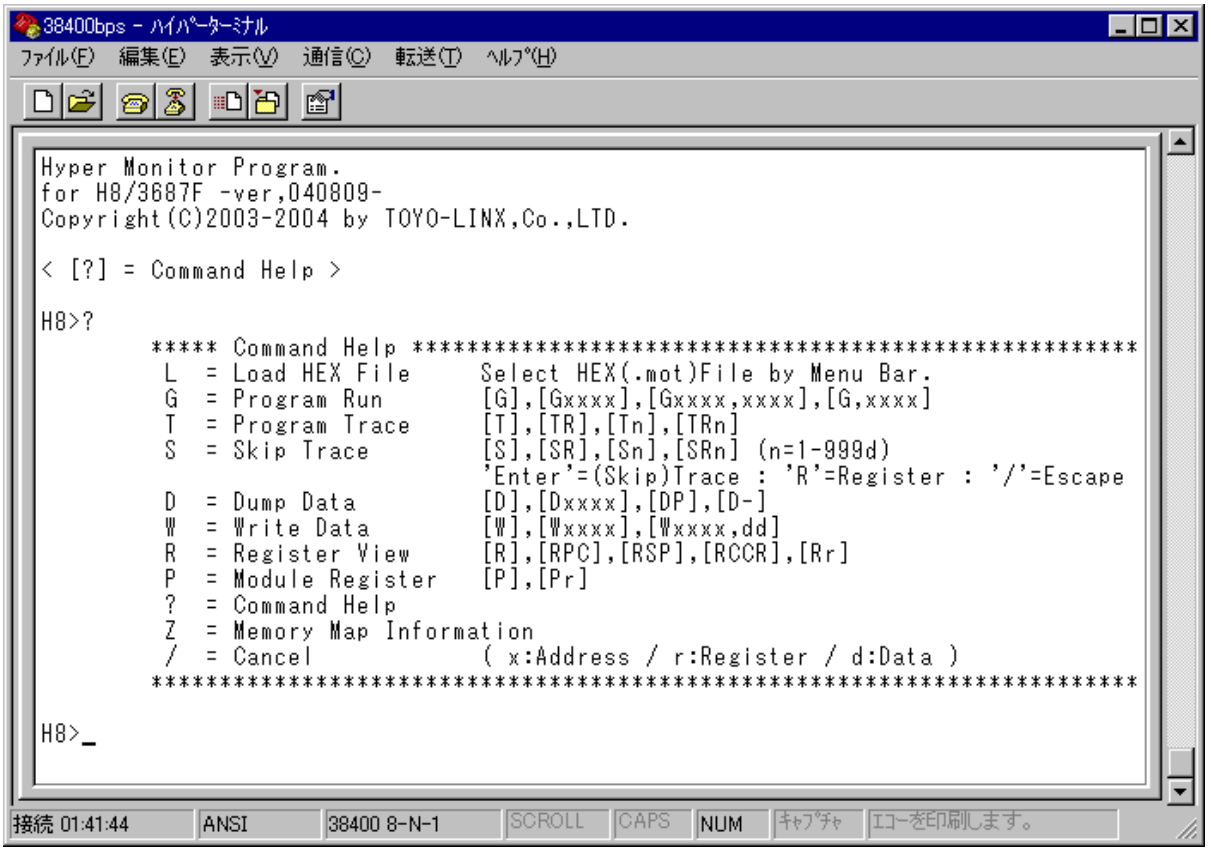
さて，今の操作は，

メモリの‘6000’番地のアドレスからのプログラムを実行する

というものでした。これで，プログラムの実行ができるようになりました。

ハイパーH8 のコマンドを調べるには…

‘G’コマンドを使いましたが，そのほかにもハイパーH8 には便利なコマンドがたくさん用意されています。詳しくはハイパーH8 のマニュアルを見ていただくとして，思い出しやすいようにコマンドヘルプがハイパーH8 には組み込まれています。キーボードから‘？’を入力して下さい。次の画面が表示されます。



ハイパーH8 は便利な道具なんですが…

ハイパーH8 は便利な道具ですが，多少の制限もあります。もっとも大きな制限は「ROM にデータを書き込むことができない」ということです。

この制限のため，ハイパーH8 でプログラムを入力する時は，RAM に入力しなければなりません。また，HEW を使ってアセンブルする時も，RAM 上にプログラムができるようにSection を設定しなければなりません。（この意味は HEW を使う章でわかります。）

さらに，ROM に比べて RAM のサイズが小さいため，あまり大きなプログラムを実行することができない，という問題もおきます。

しかし，学習用と割り切って使う分には全く気にする必要はありません。なお，ROM にプログラムを書き込む場合は，専用のツール（無償版があります）を使うことになります。また，デバッグまで行なう場合は‘E8a’というエミュレータを購入して使うことになります。

第3章 マシン語でプログラムを作ってみよう

1. プログラムの作成
2. プログラムの入力
3. プログラムをデバッグする

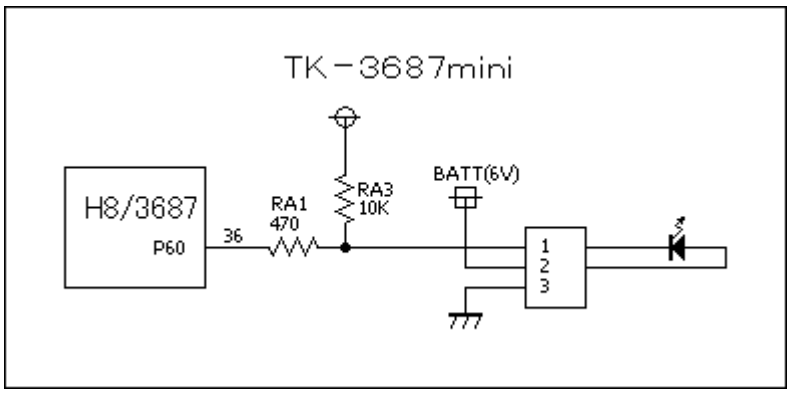
プログラムの作り方を理解するための最も早道は何でしょうか。それは，とにかくたくさん作ってみる，ということです。くりかえし作っているうちに段々プログラミングの考え方が身についてきます。

とはいっても，最初はどこから手をつけてよいかわらないでしょう。そこでこの章では，何もないところからプログラムを作り始めて，TK-3687mini で動かすまでの流れを理解しましょう。もっとも，ここで作るプログラムはものすご～く簡単なものです。命令の細かい部分はルネサスの資料，「H8/300H シリーズ　プログラミングマニュアル」で，もっと詳しい説明がされています。興味のある方はお読みください。

組込み系の分野では，コンピュータの仕組みやプログラムの本質を知ることはとても大事なことです。それで，C言語を学ぶ前にマイコンがどのようにプログラムを動かしているのか、マシン語の世界をちょっとだけのぞいてみましょう。

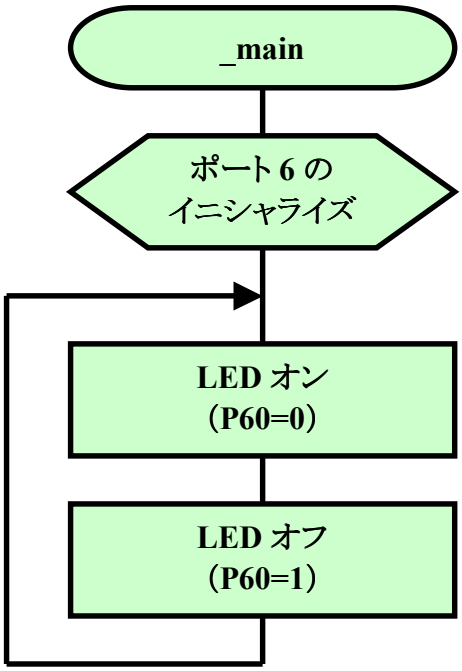
1. プログラムの作成

ここで作るプログラムは，TK-3687mini の P60 に接続した LED を点滅させるというものです。その部分の回路図を抜き出すと次のようになります。



■ フローチャートの作成

まずはおおまかなフローチャートを作ってどんなプログラムにするか考えてみましょう。できたフローチャートは下のとおりです。それほど難しくありませんね。LED のオンとオフを繰り返すだけです。



■ コーディング

さて次は，今考えたフローチャートを見ながら H8/3687 のアセンブラ命令に変換していきます。これをコーディングといいます。コーディングした結果は下のリストのとおりです。

```
_main:
        MOV.B   #H'01,ROL       ;ポート6のイニシャライズ
        MOV.B   ROL,@H'FFE9
LOOP:
        BCLR    #0,@H'FFD9      ;LEDオン (P60=0)
        BSET    #0,@H'FFD9      ;LEDオフ (P60=1)
        BRA     LOOP            ;LOOPにジャンプ
```

ところで，実はまだ人間の言葉にすぎなくて，マイコンにとっては理解できない外国語です。それで，マイコンの言葉(=マシン語，機械語)に直す必要があります。

■ マシン語に変換する

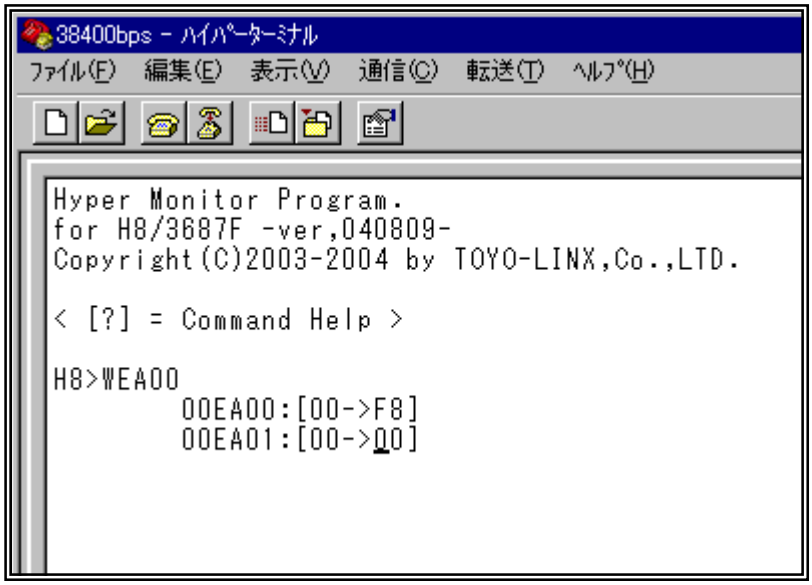
コーディングが終了したものの，このままではマイコン（H8/3687）は何をしたら良いのか理解できません。マイコンが理解できるのはマシン語（機械語）と呼ばれる 16 進の数字の羅列だけです。そこで，次はマシン語への変換作業を行ないます。これをアセンブルと呼びます。マシン語に変換すると次のようになります。これをハイパーH8 で RAM に入力していきます。

マシン語		ソースリスト			
アドレス	データ	ラベル	ニーモニック	オペランド	コメント
EA00	F8	_main:	MOV.B	#H'01,ROL	ポート6のイニシャライズ
EA01	01				
EA02	38		MOV.B	ROL,@H'FFE9	
EA03	E9				
EA04	7F	LOOP:	BCLR	#0,@H'FFD9	LEDオン（P60=0）
EA05	D9				
EA06	72				
EA07	00				
EA08	7F		BSET	#0,@H'FFD9	LEDオフ（P60=1）
EA09	D9				
EA0A	70				
EA0B	00				
EA0C	40		BRA	LOOP	LOOPにジャンプ
EA0D	F6				
EA0E					
EA0F					

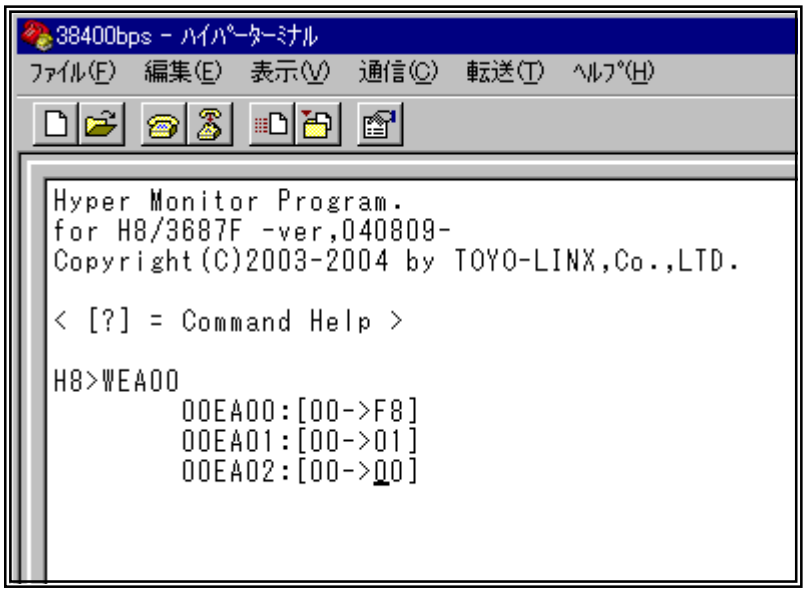
2. プログラムの入力

それでは，ハイパーH8 を使って TK-3687mini のメモリにマシン語を入力していきましょう。‘W’コマンドを使います。‘EA00’番地から入力しますので，パソコンのキーボードから‘WEA00’と入力して‘Enter’キーを押します。入力状態になったら 1 バイトづつ順番にデータを入力します。

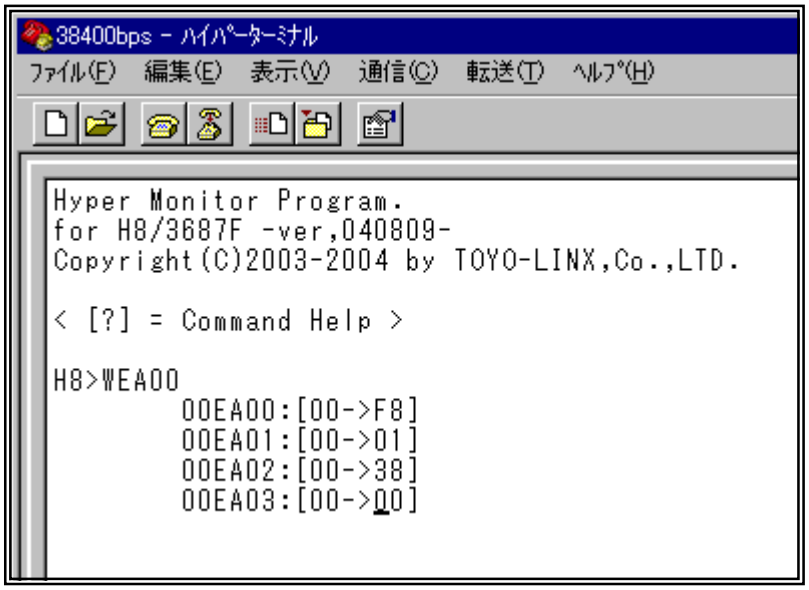
キーボード：‘F’，‘8’，‘Enter’



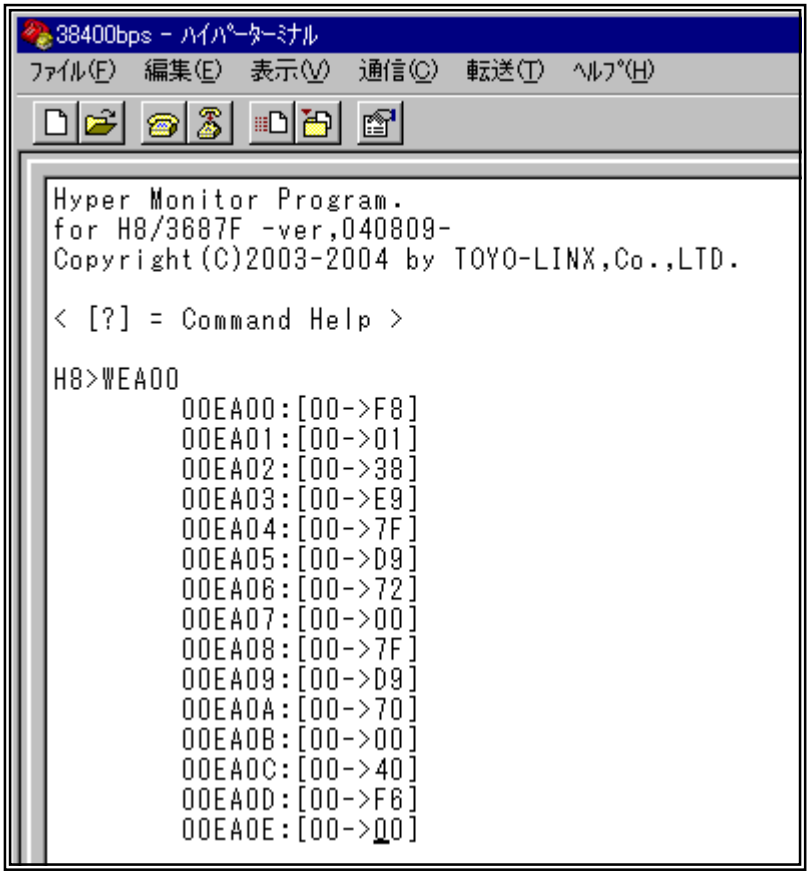
キーボード：‘0’，‘1’，‘Enter’



キーボード：‘3’，‘8’，‘Enter’

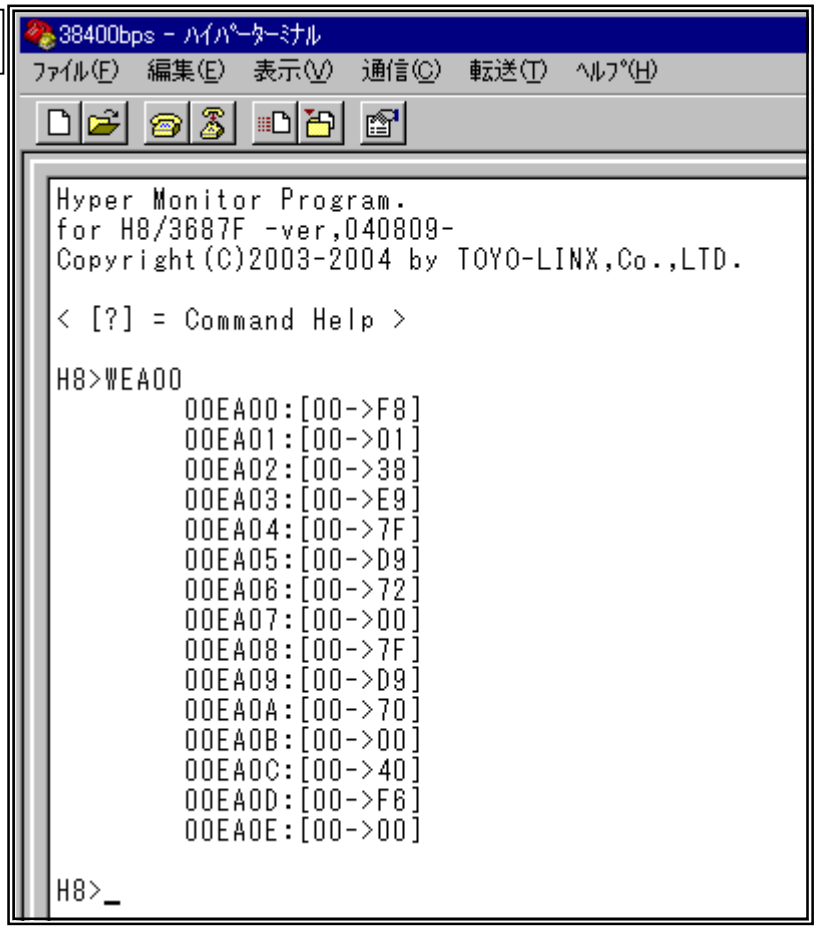


キーボード：‘F’，‘6’，‘Enter’

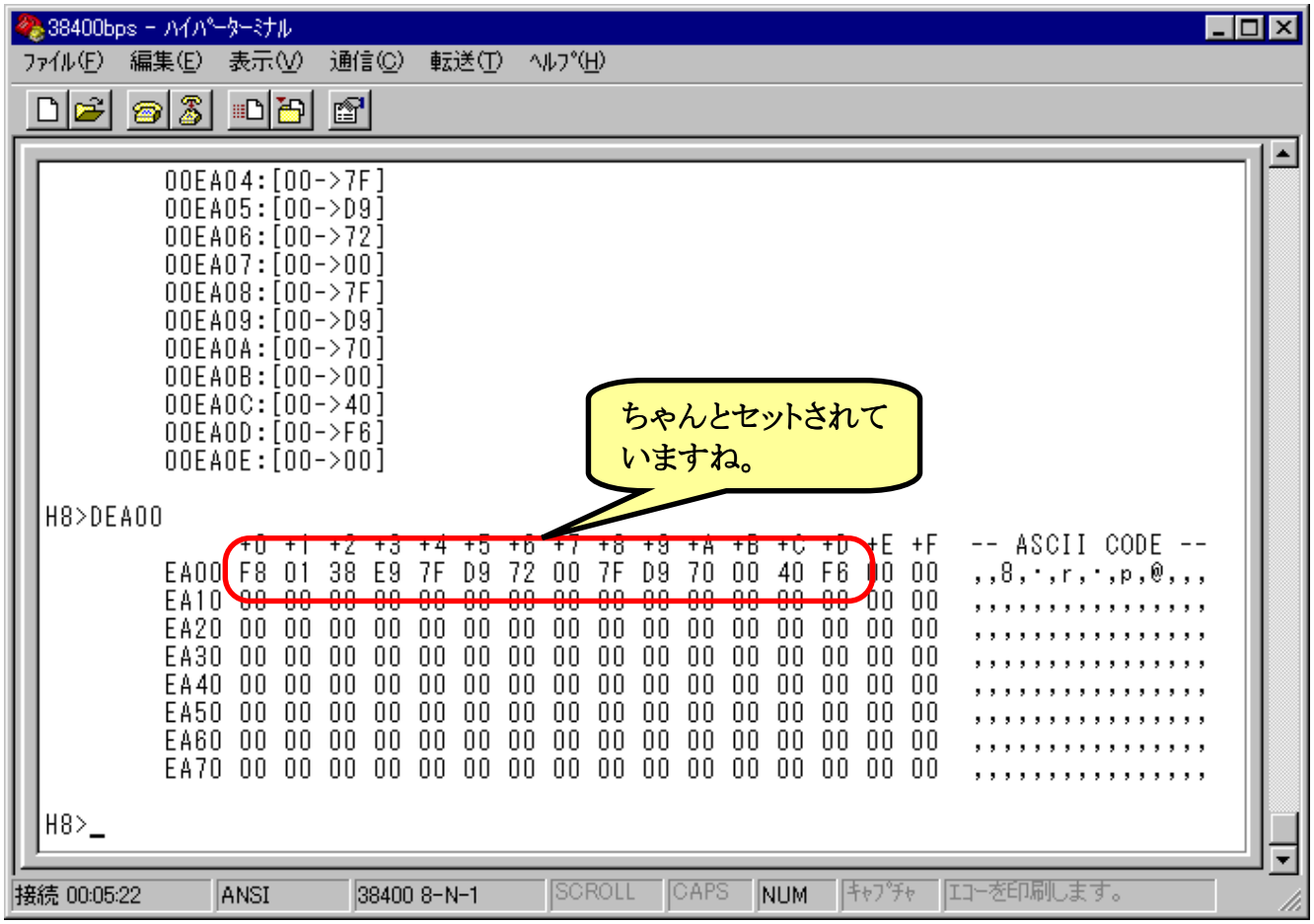


全てのデータを入力したら，キーボードから‘/’を押してコマンド入力に戻ります。

キーボード：‘/’



最後に，間違いなくデータを入力できたか確認しておきましょう。パソコンのキーボードから‘DEA00’と入力して‘Enter’キーを押します。



さてここで，この項目で使ったハイパーH8 のコマンドをまとめておきます。まずは‘W’コマンドです。‘WEA00’と入力しましたが，これは，

メモリの‘EA00’番地のアドレスからデータをセットする

という意味です。入力が終わったら‘/’キーを押すとコマンド入力に戻ります。

もう一つは‘D’コマンドです。‘DEA00’と入力しましたが，これは，

メモリの‘EA00’番地のアドレスからメモリの内容をダンプ表示する

という意味です。

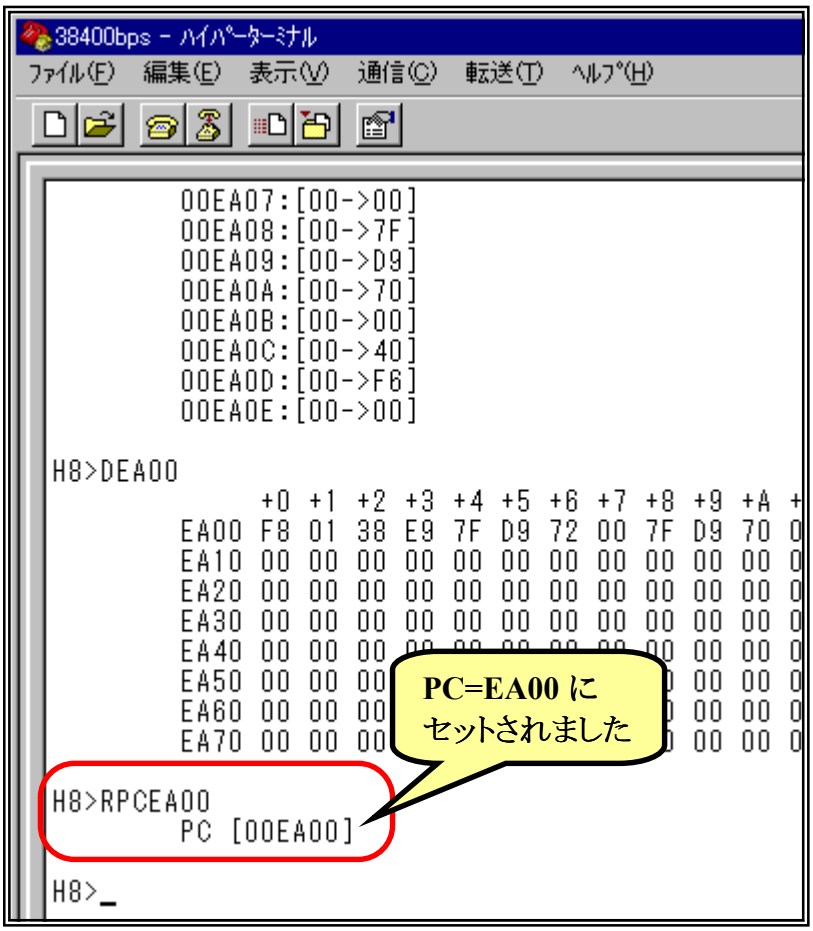
これで，メモリの中身を読み書きできるようになりました。

3. プログラムをデバッグする

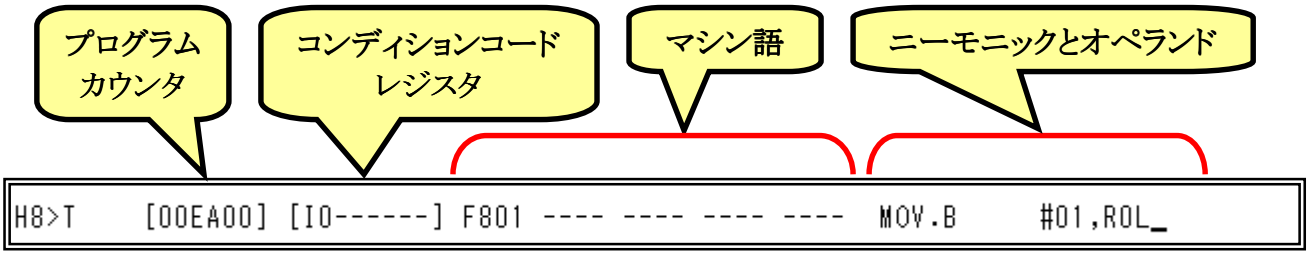
では，プログラムを実行してみましょう。今回は‘G’コマンドではなく，‘T’コマンドを使ってプログラムの動きを一命令づつ追いかけてみたいと思います。

■ トレース実行

最初にどこからプログラムをスタートするか指定します。このプログラムは EA00 番地からスタートしますので，PC に EA00 をセットします。キーボードから‘RPCEA00’と入力して‘Enter’キーを押します。そうすると，PC に EA00 がセットされたことが表示されます。



次に，キーボードから‘T’と入力して‘Enter’キーを押してください。



このとき注意したいのは，この時点ではまだこの命令は実行されていないということです。次に，‘Enter’キーを押すと，この命令が実行され，その結果が次に表示されます。

それでは，‘Enter’キーを押していってプログラムが考えたとおりに動いていくかたしかめてみましょう。また，LEDがちゃんと点滅するかもみてください。

```
H8>T    [00EA00] [I0------] F801 ---- ---- ---- ----  MOV.B     #01,R0L
        [00EA02] [I0------] 38E9 ---- ---- ---- ----  MOV.B     R0L,@FFE9
        [00EA04] [I0------] 7FD9 7200 ---- ---- ----  BCLR      #0,@FFD9
        [00EA08] [I0------] 7FD9 7000 ---- ---- ----  BSET      #0,@FFD9
        [00EA0C] [I0------] 40F6 ---- ---- ---- ----  BRA       EA04
        [00EA04] [I0------] 7FD9 7200 ---- ---- ----  BCLR      #0,@FFD9
        [00EA08] [I0------] 7FD9 7000 ---- ---- ----  BSET      #0,@FFD9
        [00EA0C] [I0------] 40F6 ---- ---- ---- ----  BRA       EA04
        [00EA04] [I0------] 7FD9 7200 ---- ---- ----  BCLR      #0,@FFD9
        [00EA08] [I0------] 7FD9 7000 ---- ---- ----  BSET      #0,@FFD9
        [00EA0C] [I0------] 40F6 ---- ---- ---- ----  BRA       EA04_
```

いかがでしょうか。ちゃんと動きましたか？うまく動作しないときはプログラムの入力ミスの可能性が大です。もう一度ちゃんと入力しているかたしかめてみましょう。

さて，このように命令を一命令づつ実行して，考えたとおりに動いていくか確認するのがデバッグ（プログラムのまちがい探し）の第一歩です。

最後に，‘/’キーを押してコマンド入力に戻りましょう。

```
H8>T    [00EA00] [I0------] F801 ---- ---- ---- ----  MOV.B     #01,R0L
        [00EA02] [I0------] 38E9 ---- ---- ---- ----  MOV.B     R0L,@FFE9
        [00EA04] [I0------] 7FD9 7200 ---- ---- ----  BCLR      #0,@FFD9
        [00EA08] [I0------] 7FD9 7000 ---- ---- ----  BSET      #0,@FFD9
        [00EA0C] [I0------] 40F6 ---- ---- ---- ----  BRA       EA04
        [00EA04] [I0------] 7FD9 7200 ---- ---- ----  BCLR      #0,@FFD9
        [00EA08] [I0------] 7FD9 7000 ---- ---- ----  BSET      #0,@FFD9
        [00EA0C] [I0------] 40F6 ---- ---- ---- ----  BRA       EA04
        [00EA04] [I0------] 7FD9 7200 ---- ---- ----  BCLR      #0,@FFD9
        [00EA08] [I0------] 7FD9 7000 ---- ---- ----  BSET      #0,@FFD9
        [00EA0C] [I0------] 40F6 ---- ---- ---- ----  BRA       EA04

H8>_
```

さてここで，この項目で使ったハイパーH8 のコマンドをまとめておきます。まずは‘R’コマンドです。‘RPCEA00’と入力しましたが，これは，

プログラムカウンタに‘EA00’をセットする

という意味です。

もう一つは‘T’コマンドです。‘T’と入力しましたが，これは，

プログラムカウンタが示すアドレスからトレース実行する

という意味です。‘Enter’キーを押すと一命令づつレース実行します。また，‘/’キーを押すとコマンド入力に戻ります。

これで，スタートアドレスを指定して，そこからトレース実行ができるようになりました。

第4章 C 言語でプログラムを作ってみよう

前の章ではアセンブラ→マシン語でプログラムを作ってみました。ハンドアセンブルは H8 に限らず，どんな CPU にも対応できるので便利（？）ですが，プログラムが長くなっていくと，間違いは増え，間違いを修正するのも大変で，何よりアセンブルするだけで，ものすご～く疲れます。しかも，アセンブラはマシン語よりわかりやすいとはいえ，人間にとってはまだまだわかりにくい言語です。というわけで，大変なことはパソコンにまかせてしまいましょう。そこで，C 言語を使うのがスマートな方法となります。

1. 統合開発環境「HEW」のインストール

ダウンロード先をデスクトップにした場合で説明します。ダウンロードした‘h8v6200_ev. exe’をダブルクリックしてください。すると，インストールが始まります。画面の指示に従ってインストールしてください。

さて，この章で入手した無償評価版コンパイラは，はじめてコンパイルした日から 60 日間は製品版と同等の機能と性能のままで試用できます。61 日目以降はリンクサイズが 64K バイトまでに制限されますが，H8/3687 はもともとアクセスできるメモリサイズが 64K までバイトなので，この制限は関係ありません。また，無償評価版コンパイラは製品開発では使用できないのですが，H8/300H Tiny シリーズ（H8/3687 も含まれる）では許可されています。

さて，無償評価版コンパイラは，無償とはいえ非常に強力な開発環境で，アセンブラにも C 言語にも対応しています。（というよりは，C 言語がメインで，アセンブラはその一部分を使用しているに過ぎないのですが…。）がんばってマスターしてください。

最新版の HEW を手に入れましょう

HEW は頻繁にバージョンアップされています。HEW はルネサステクノロジのマイコン全てに対応しているため，H8 シリーズはもとより，R8 シリーズや SH シリーズなど，対応するマイコンが増えるとそのたびにマイナーチェンジされるようです。また，その際に報告されていた不具合を一緒に修正することもあります。

それで，ルネサステクノロジのホームページは定期的にのぞいてみることをおすすめします。特にデバイスアップデータの情報は要注意です。

2. ハイパーH8 を使うときのメモリマップ

HEW を使うときのコツの一つは，メモリマップを意識する，ということです。プログラムがどのアドレスに作られて，データはどのアドレスに配置されるか，ちょっと意識するだけで，HEW を理解しやすくなります。ハイパーH8 を使うときのメモリマップは次のとおりです。

<table>
<tr><td>0000番地

DFFF番地</td><td>ROM/フラッシュメモリ
（56Kバイト）</td><td colspan="3">モニタプログラム
“ハイパーH8”</td></tr>
<tr><td>E000番地
E7FF番地</td><td>未使用</td><td colspan="3">未使用</td></tr>
<tr><td>E800番地</td><td rowspan="3">RAM
（2Kバイト）</td><td colspan="3">ハイパーH8
ユーザ割り込みベクタ</td></tr>
<tr><td>E860番地</td><td>PResetPRG
PIntPRG</td><td>リセットプログラム
割り込みプログラム</td><td rowspan="2">ユーザ
RAM
エリア</td></tr>
<tr><td>EA00番地

EFFF番地</td><td>P
C
C$DSEC

C$BSEC

D</td><td>プログラム領域
定数領域
初期化データセクションの
アドレス領域
未初期化データセクションの
アドレス領域
初期化データ領域</td></tr>
<tr><td>F000番地
F6FF番地</td><td>未使用</td><td colspan="3">未使用</td></tr>
<tr><td>F700番地
F77F番地</td><td>I/Oレジスタ</td><td colspan="3">I/Oレジスタ</td></tr>
<tr><td>F780番地

FB7F番地</td><td>RAM（1Kバイト）
フラッシュメモリ書換用
ワークエリアのため
FDTとE7,E8, E8a
使用時は
ユーザ使用不可</td><td>B
R</td><td>未初期化データ領域
初期化データ領域
（変数領域）</td><td rowspan="3">ユーザ
RAM
エリア</td></tr>
<tr><td>FB80番地

FD80番地
FDFF番地</td><td rowspan="3">RAM
（1Kバイト）</td><td>S</td><td>スタック領域</td></tr>
<tr><td>FE00番地
FF7F番地</td><td colspan="3">ハイパーH8
ワークエリア</td></tr>
<tr><td>FF80番地
FFFF番地</td><td>I/Oレジスタ</td><td colspan="3">I/Oレジスタ</td></tr>
</table>

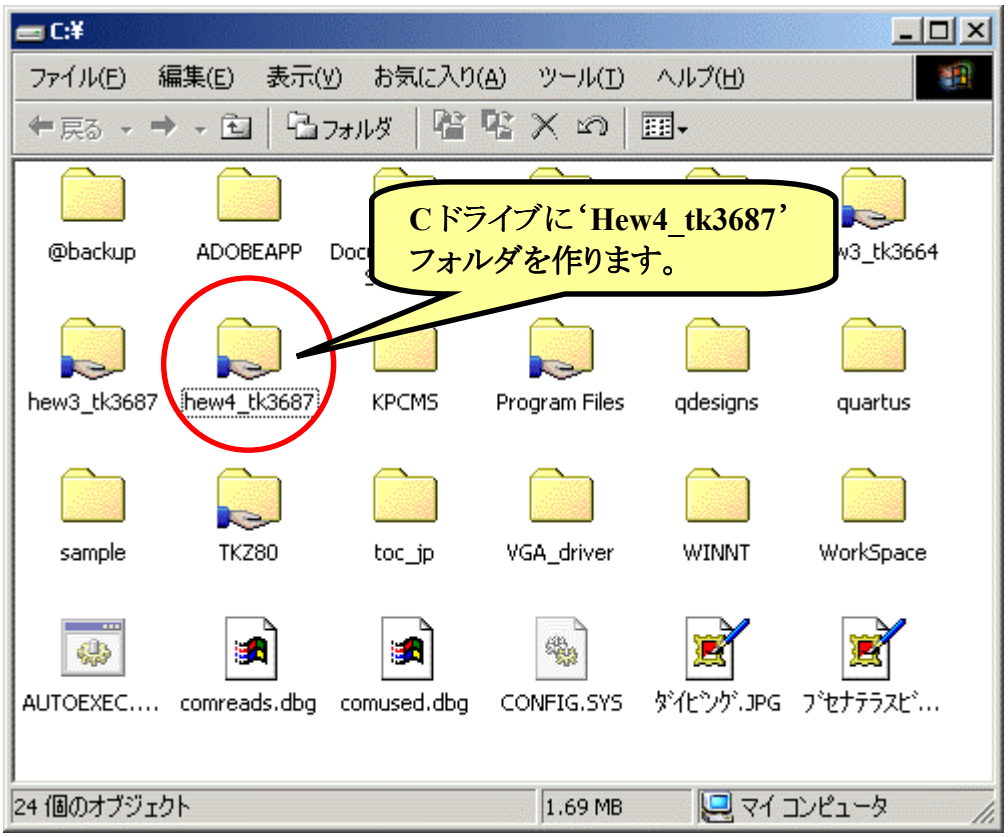
メモリマップのうち“ユーザ RAM エリア”の部分だけが自由に使用できるエリアです。

3. C 言語でプログラムを作ってみる

■ プロジェクトの作成

HEW ではプログラム作成作業をプロジェクトと呼び，そのプロジェクトに関連するファイルは 1 つのワークスペース内にまとめて管理されます。通常はワークスペース，プロジェクト，メインプログラムには共通の名前がつけられます。この章で作るプロジェクトは‘IoPort_led_c’と名付けます。以下に，新規プロジェクト‘IoPort_led_c’を作成する手順と動作確認の手順を説明します。

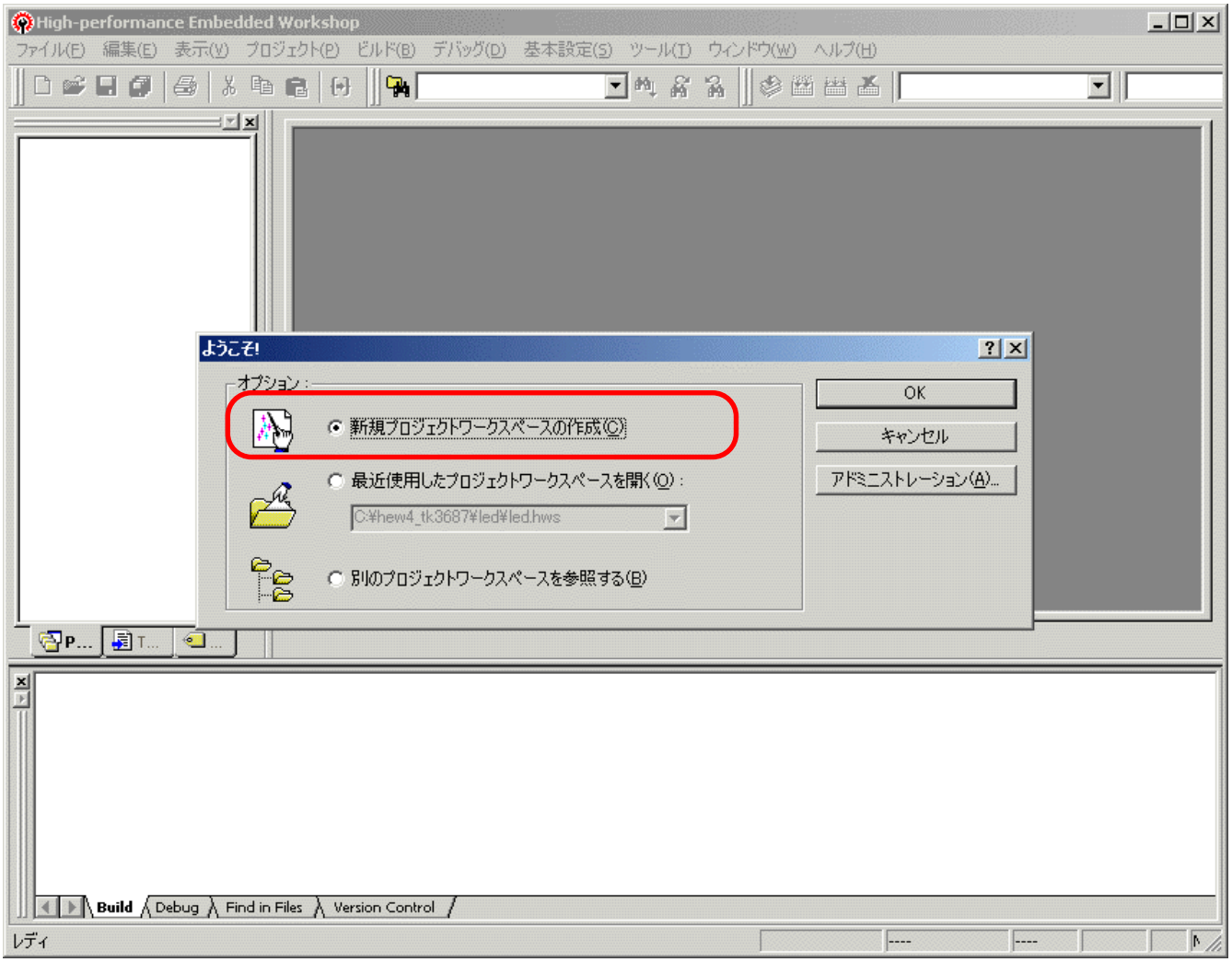
しかしその前に，HEW 専用作業フォルダを作っておきましょう。C ドライブに‘Hew4_tk3687’を作ってください。このマニュアルのプロジェクトは全てこのフォルダに作成します。



では，HEW を起動しましょう。スタートメニューから起動します。

スタート → プログラム → Renesas → High-performance Embedded Workshop → High-performance Embedded Workshop

HEW を起動すると下記の画面が現れるので，「新規プロジェクトワークスペースの作成」を選択して‘OK’をクリックします。



前に作ったプロジェクトを使うとき

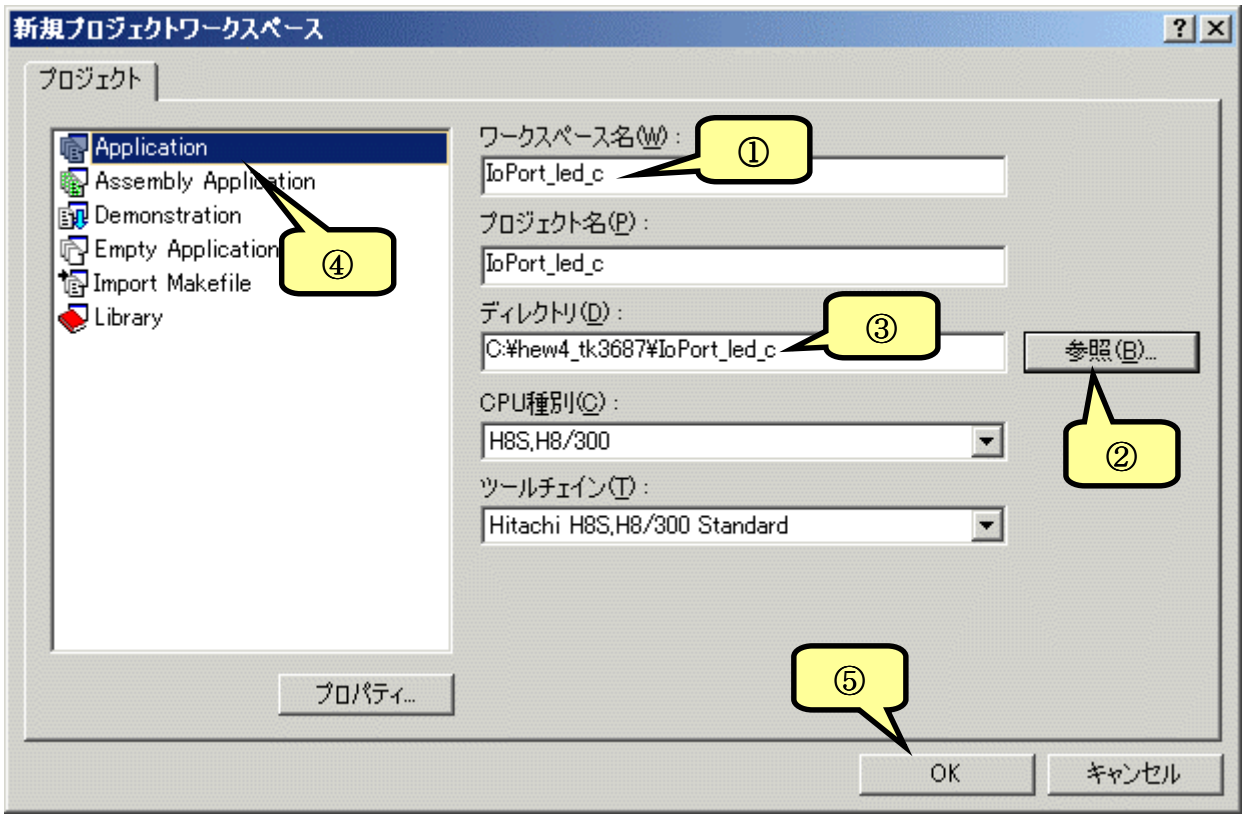
その場合は，「ようこそ!」ダイアログで「最近使用したプロジェクトワークスペースを開く」を選択して‘OK’をクリックします。そのプロジェクトの最後に保存した状態で HEW が起動します。

まず，①「ワークスペース名（W）」（ここでは‘IoPort_led_c’）を入力します。「プロジェクト名（P）」は自動的に同じ名前になります。

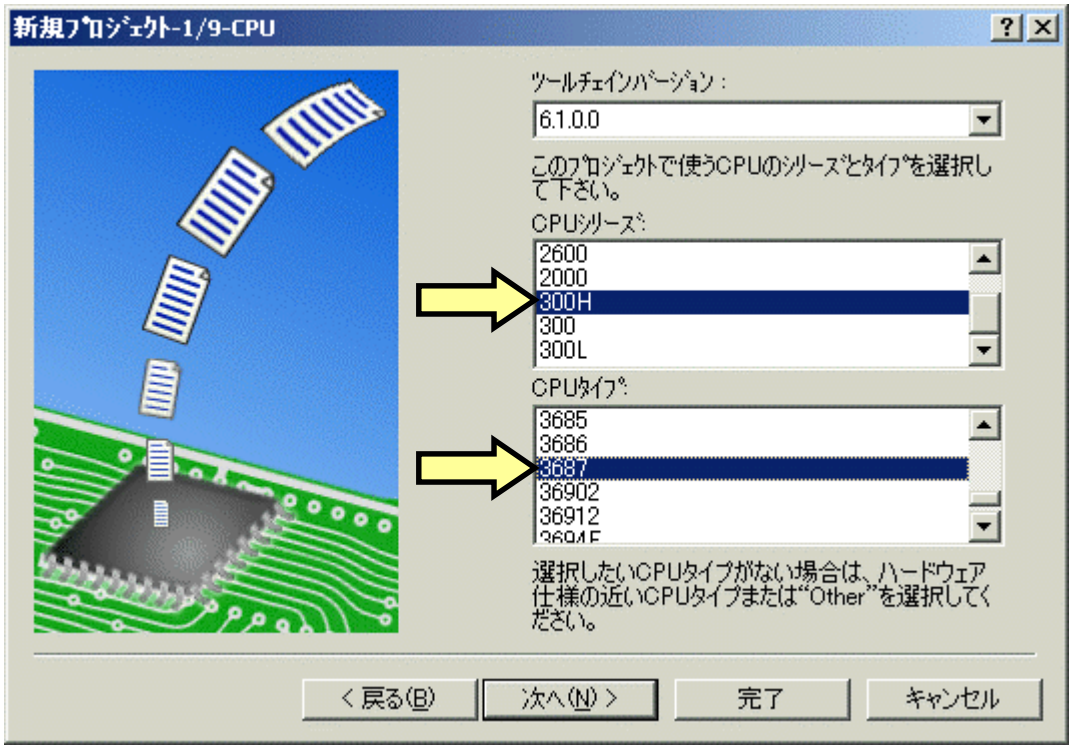
ワークスペースの場所を指定します。②右の「参照（B）…」ボタンをクリックします。そして，あらかじめ用意した HEW 専用作業フォルダ（ここでは Hew4_tk3687）を指定します。設定後，「ディレクトリ（D）」が正しいか確認して下さい。（③）

次にプロジェクトを指定します。今回は C 言語なので④「Application」を選択します。

入力が終わったら⑤「OK」をクリックして下さい。

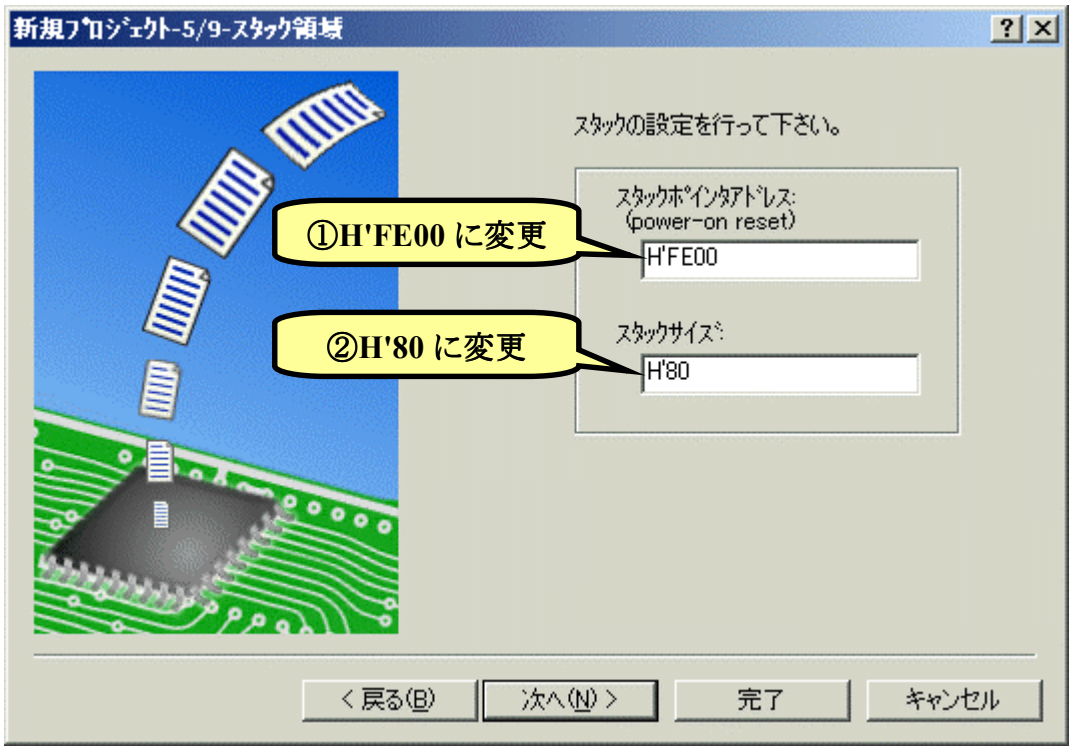


「新規プロジェクト－1／9－CPU」で，使用する CPU シリーズ（300H）と，CPU タイプ（3687）を設定し，「次へ（N）＞」をクリックします。

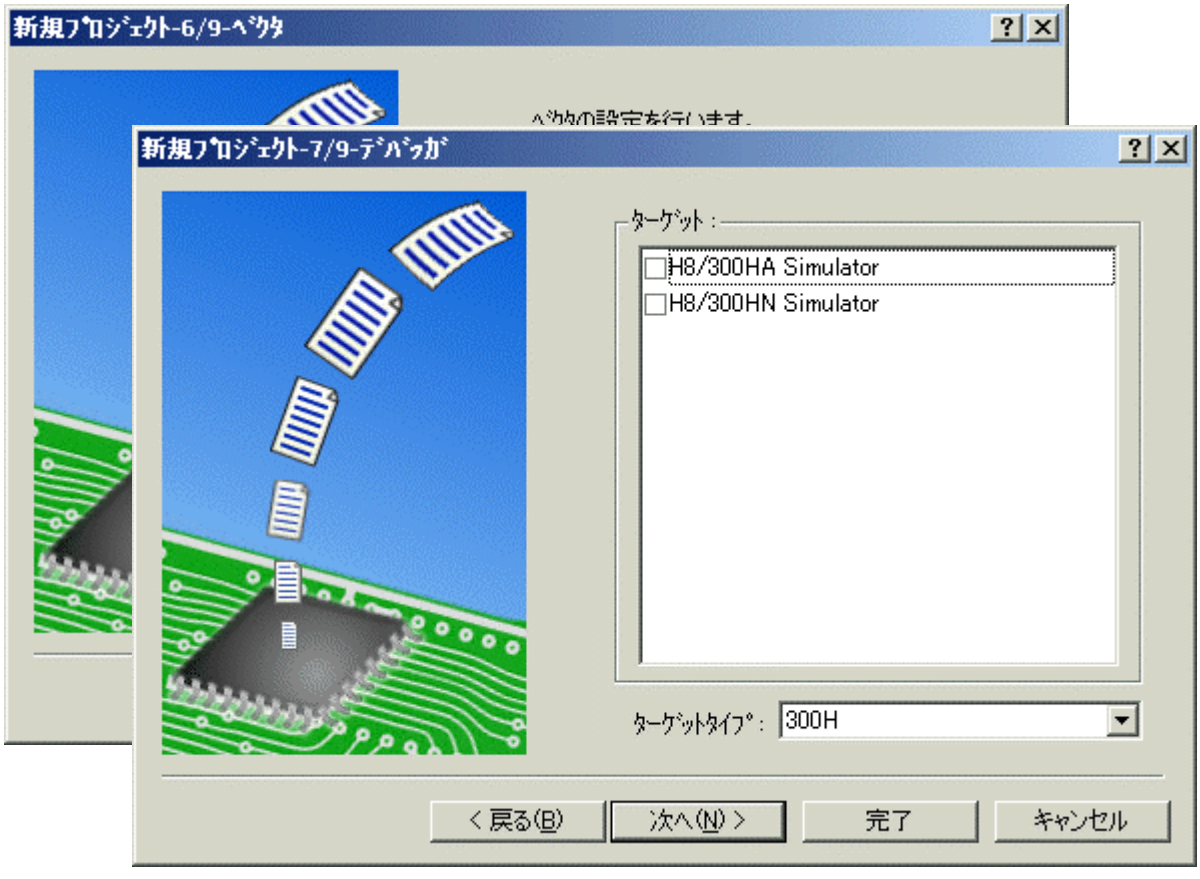


「新規プロジェクト－2／9－オプション」，「新規プロジェクト－3／9－生成ファイル」，「新規プロジェクト－4／9－標準ライブラリ」は変更しません。「次へ（N）＞」をクリックして次の画面に進みます。

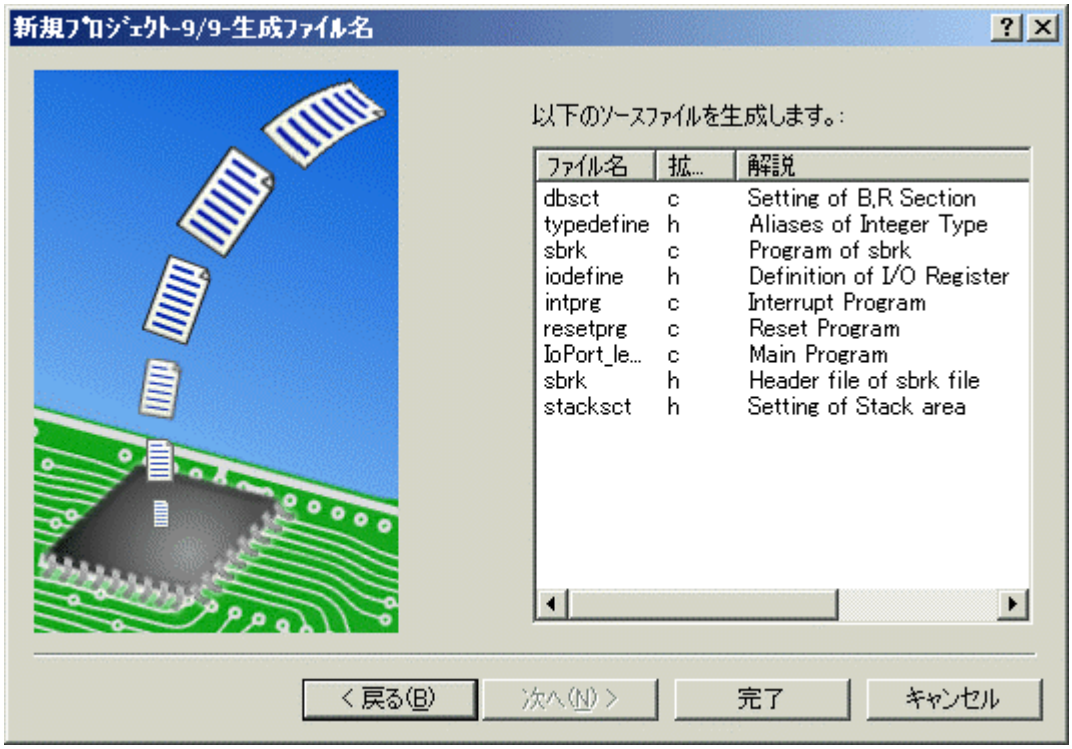
「新規プロジェクトー5／9ースタック領域」でスタックのアドレスとサイズを変更します。ハイパーH8 を使用するので，①スタックポインタを H'FE00 に，②スタックサイズを H'80 にします。設定が終わったら「次へ(N) >」をクリックします。



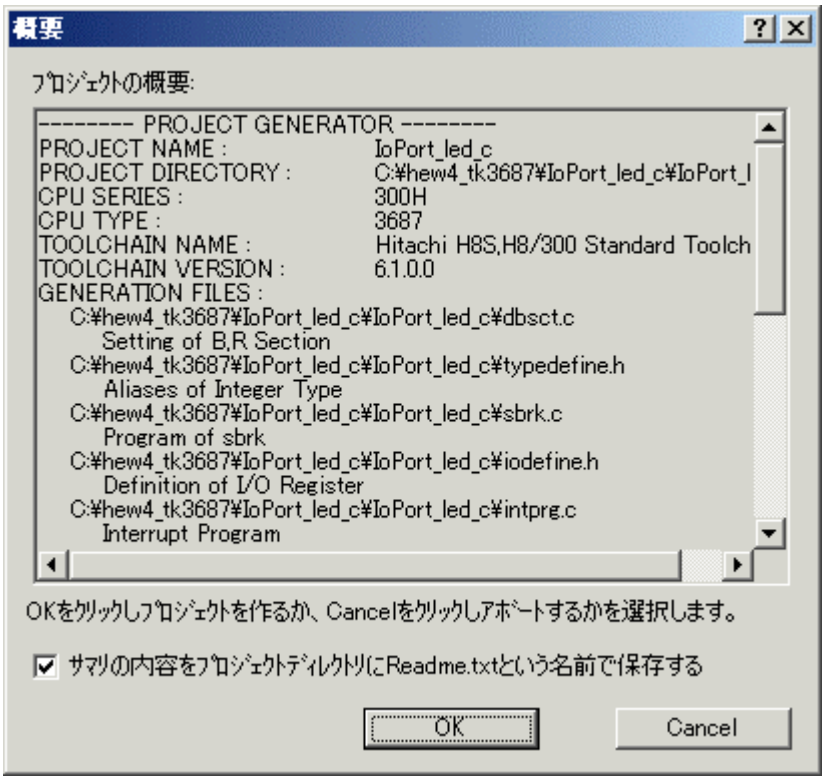
「新規プロジェクトー6／9ーベクタ」と「新規プロジェクトー7／9ーデバッガ」は変更しません。「次へ(N) >」をクリックして次の画面に進みます。



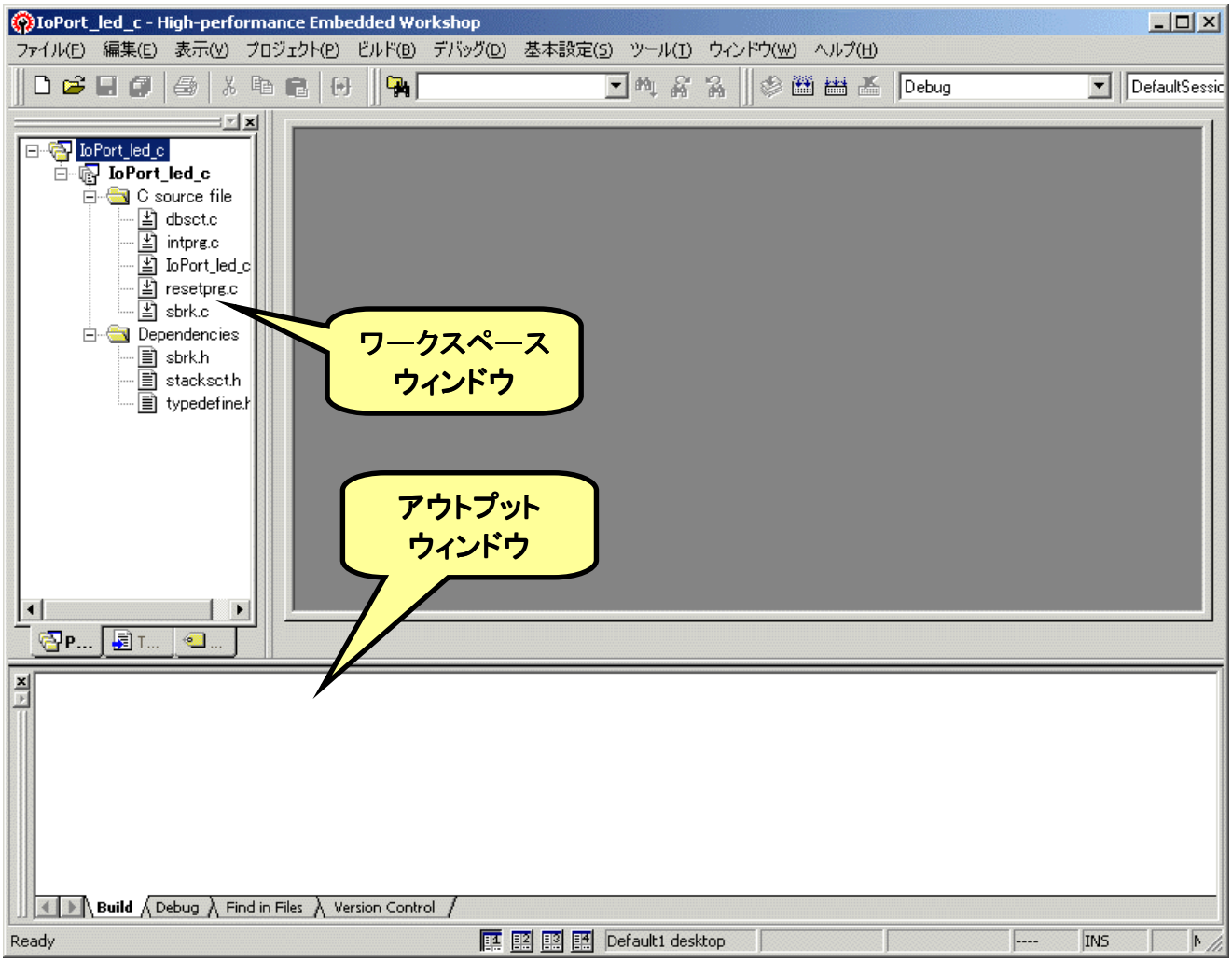
次は「新規プロジェクトー9／9ー生成ファイル名」です。ここも変更しません。「完了」をクリックします。



すると,「概要」が表示されるので「OK」をクリックします。

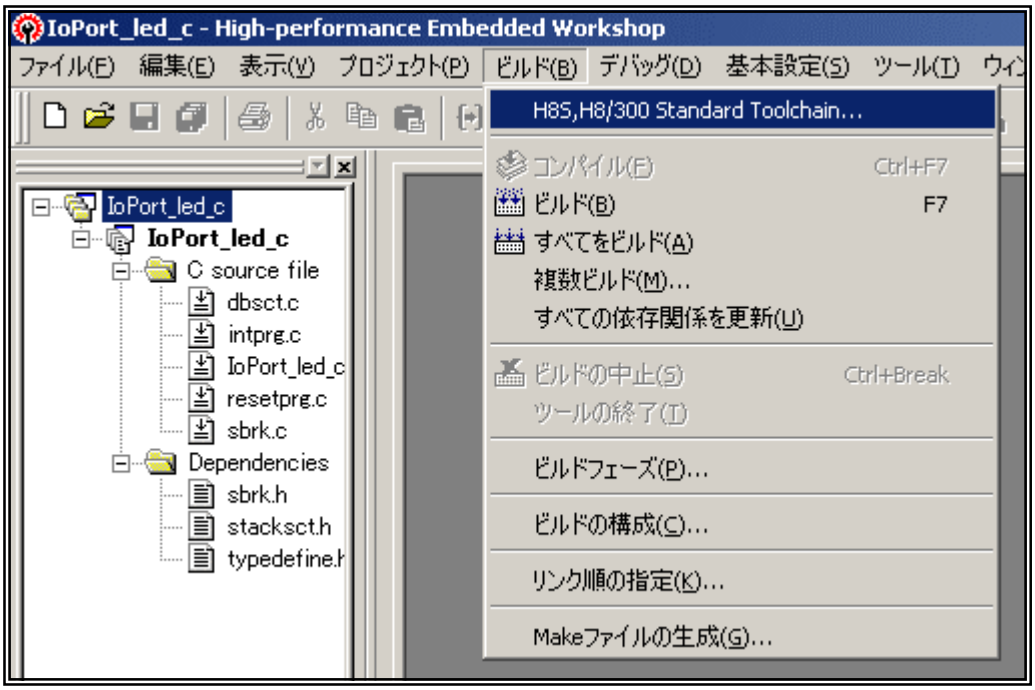


これで，プロジェクトワークスペースが完成します。HEW はプロジェクトに必要なファイルを自動生成し，それらのファイルは左端のワークスペースウィンドウに一覧表示されます。

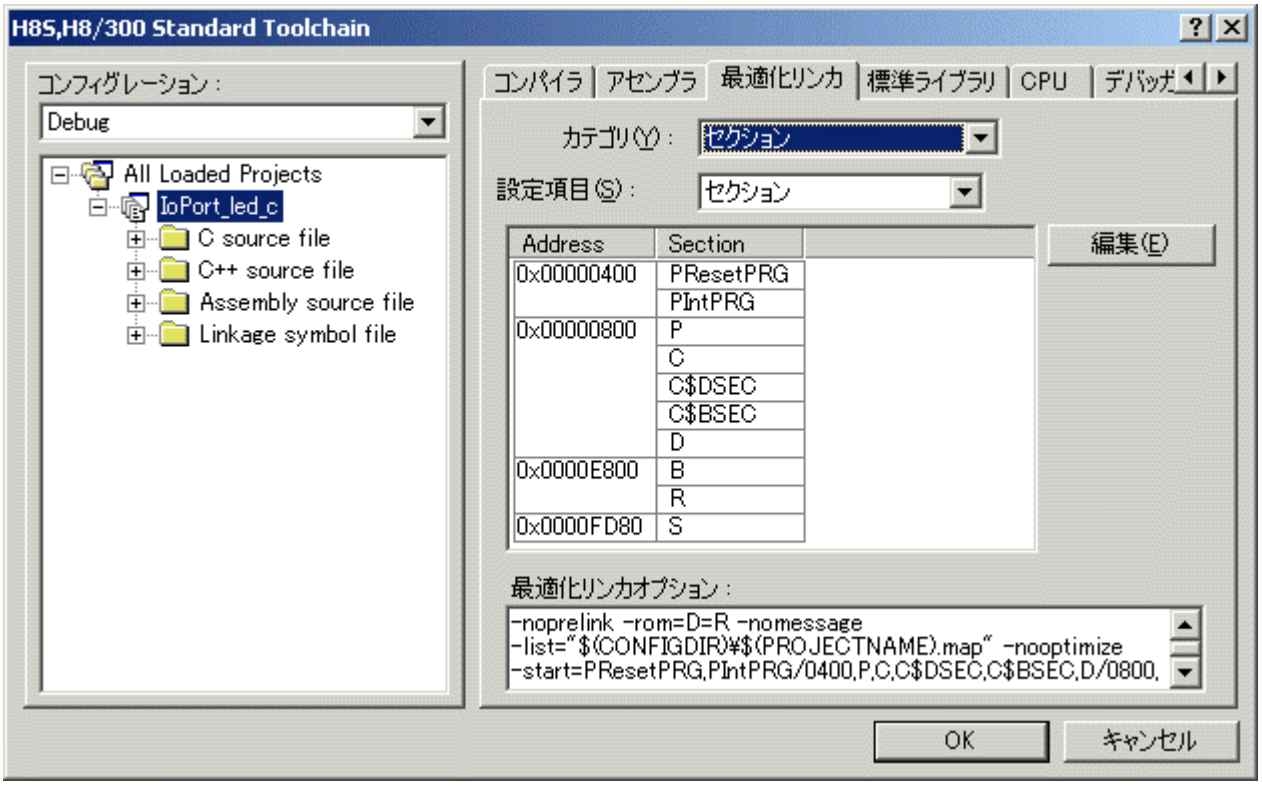


さて，これでプロジェクトは完成したのですが，ハイパーH8 を使うときはセクションを変更してプログラムが RAM 上にできるようにします。（当然ながら，ハイパーH8 を使わないときは変更する必要はなく，そのままで OK。）

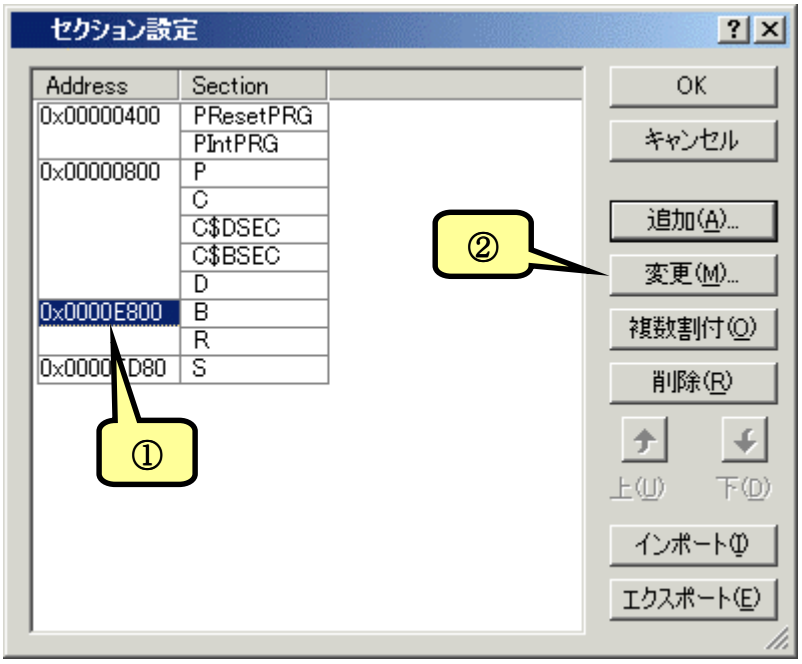
下図のように，メニューバーから「H8S,H8/300 Standard Toolchain…」を選びます。



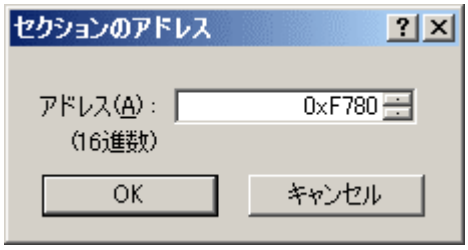
すると，「H8S，H8/300 Standard Toolchain」ウィンドウが開きます。「最適化リンカ」のタブを選び，「カテゴリ(Y)」のドロップダウンメニューの中から「セクション」を選択します。すると，下図のような各セクションの先頭アドレスを設定する画面になります。「編集(E)」ボタンをクリックしてください。



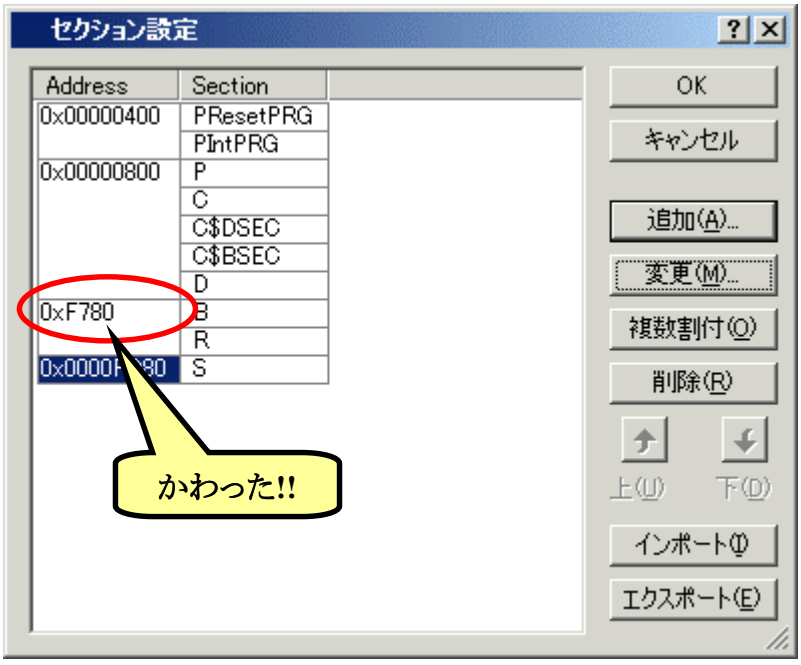
「セクション設定」ダイアログが開きます。それでは，「2. ハイパーH8 を使うときのメモリマップ」で調べたメモリマップにあわせて設定していきましょう。最初に‘B’Section のアドレスを変更します。デフォルトでは E800 番地になっていますね。①‘0x0000E800’というところをクリックして下さい。それから，②「変更(M)…」をクリックします。



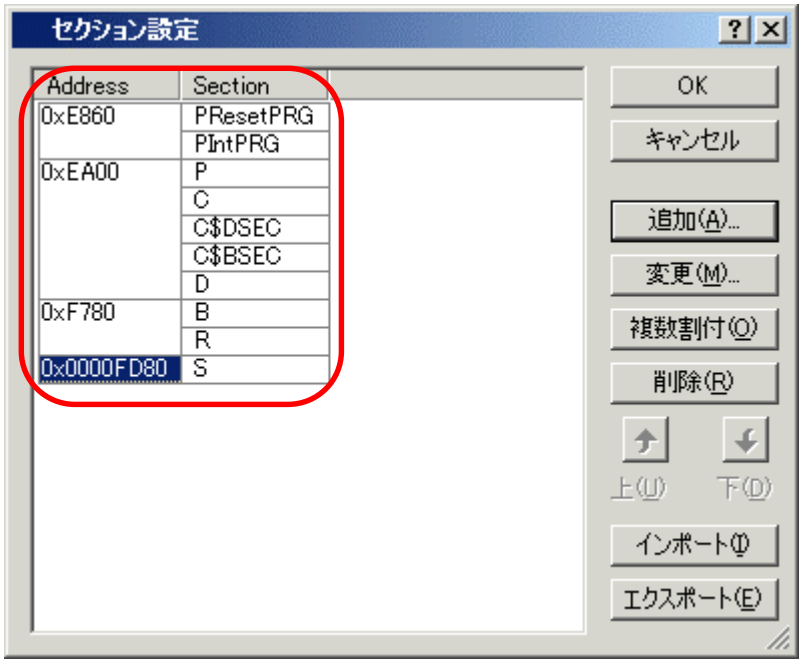
そうすると，「セクションのアドレス」ダイアログが開きます。‘B’Section は F780 番地から始まりますので，右のように入力して‘OK’をクリックします。



すると…

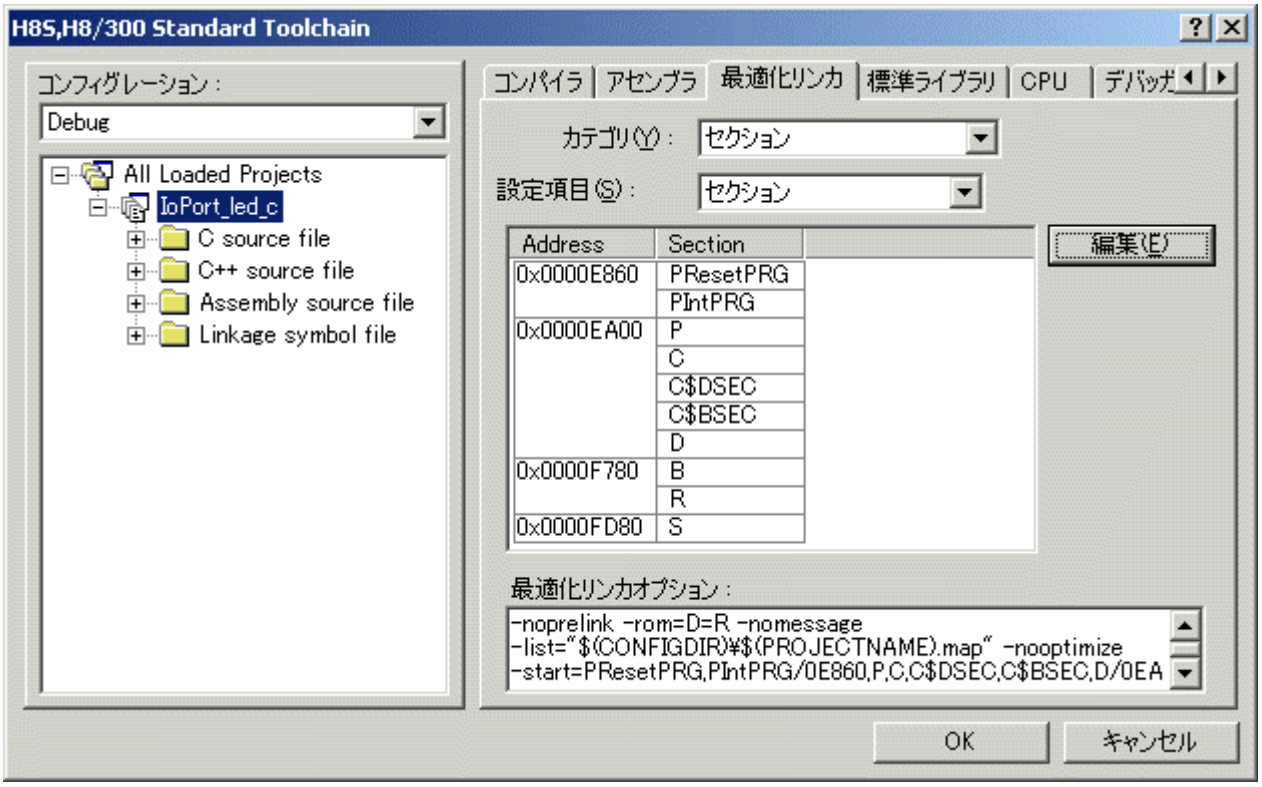


同じように，他のセクションも変更しましょう。メモリマップと同じように Section が指定されていることを確認します。ちゃんと設定されていたら「OK」をクリックします。



セクション設定の保存

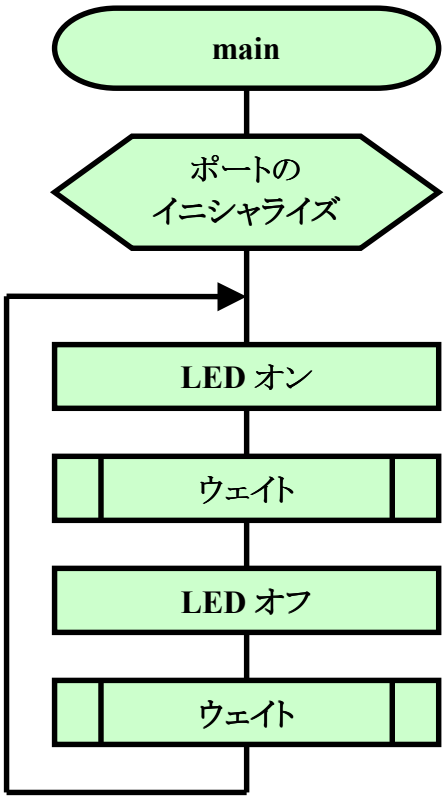
次回のために今修正したセクション情報を保存することができます。下段の「エクスポート(E)」ボタンをクリックしてください。保存用のダイアログが開きますので好きな名前を付けて保存します。次回は「インポート(I)」ボタンをクリックすると保存したセクション設定を呼び出すダイアログが開きます。(おすすめ!!)



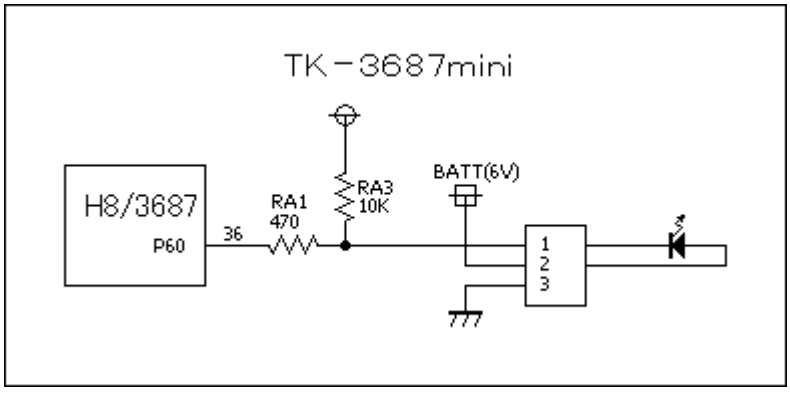
もう一度確認しましょう。ちゃんと設定されていたら「OK」をクリックして‘H8S，H8/300 Standard Toolchain’ウィンドウを閉じます。

■ ソースリストの入力

いよいよソースリストの入力です。3 章で作った LED の点滅プログラムの考え方をもとにして，G コマンドで実行しても点滅がわかるようにウェイトを入れてみます。フローチャートは次のようになります。詳しくは次の章で説明しますので，リストどおり入力してみてください。



回路図は前回と同じです。下記に再掲します。



HEW のワークスペースウィンドウの‘IoPort_led_c．c’をダブルクリックしてください。すると，自動生成された‘IoPort_led_c．c’ファイルが開きます。

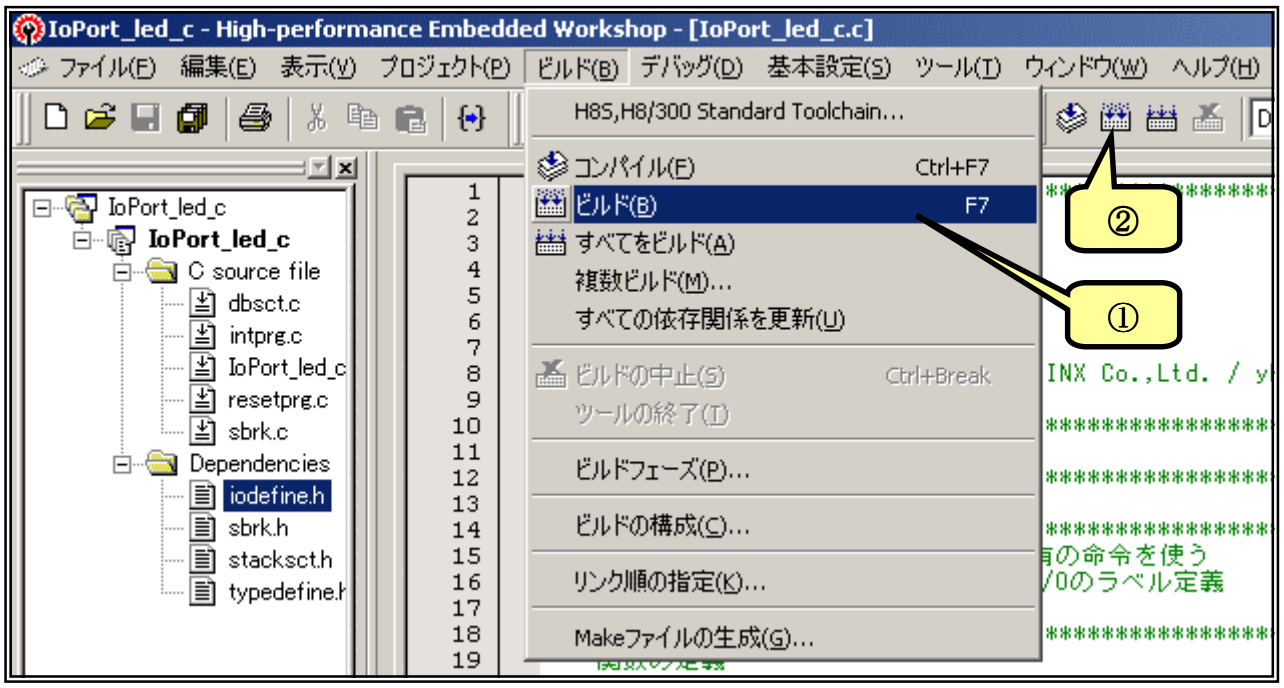
```
/***********************************************************************/
/*                                                                     */
/*  FILE        :IoPort_led_c.c                                        */
/*  DATE        :Wed, Apr 20, 2005                                     */
/*  DESCRIPTION :Main Program                                          */
/*  CPU TYPE    :H8/3687                                               */
/*                                                                     */
/*  This file is generated by Renesas Project Generator (Ver.4.0).     */
/*                                                                     */
/***********************************************************************/



#ifdef __cplusplus
extern "C" {
void abort(void);
#endif
void main(void);
#ifdef __cplusplus
}
#endif

void main(void)
{

}

#ifdef __cplusplus
void abort(void)
{

}
#endif
```

```
/***********************************************************************/
/*                                                                     */
/*  FILE        :IoPort_led_c.c                                        */
/*  DATE        :Wed, Apr 20, 2005                                     */
/*  DESCRIPTION :Main Program                                          */
/*  CPU TYPE    :H8/3687                                               */
/*                                                                     */
/*  This file is programed by TOYO-LINX Co.,Ltd. / yKikuchi            */
/*                                                                     */
/***********************************************************************/

/************************************************************************
    インクルードファイル
************************************************************************/
#include    <machine.h>     // H8特有の命令を使う
#include    "iodefine.h"    // 内蔵I/Oのラベル定義

/************************************************************************
    関数の定義
************************************************************************/
void        main(void);
void        wait(void);

/************************************************************************
    メインプログラム
************************************************************************/
void main(void)
{
    IO.PCR6 = 0x01;         // ポート6のbit0(P60)を出力に設定

    while(1){
        IO.PDR6.BIT.B0 = 0; // LEDオン
        wait();
        IO.PDR6.BIT.B0 = 1; // LEDオフ
        wait();
    }
}

/************************************************************************
    ウェイト
************************************************************************/
void wait(void)
{
    unsigned long i;

    for (i=0;i<1666666;i++){}
}
```

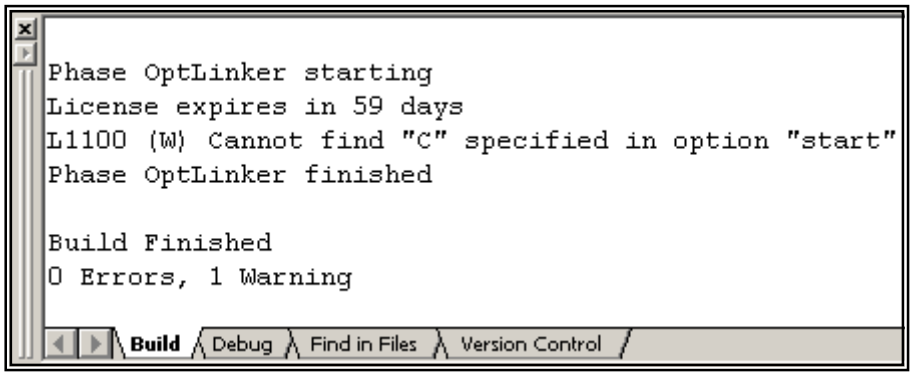
このファイルに追加・修正していきます。左のリストのとおり入力してみてください。なお，C 言語の文法については，HEW をインストールしたときに一緒にコピーされる「H8S，H8/300 シリーズ C/C++コンパイラ，アセンブラ，最適化リンケージエディタユーザーズマニュアル」の中で説明されています。

■ ビルド

では，ビルドしてみましょう。ファンクションキーの[F7]を押すか，図のように①メニューバーから‘ビルド’を選ぶか，②ツールバーのビルドのアイコンをクリックして下さい。



ビルドが終了するとアウトプットウィンドウに結果が表示されます。文法上のまちがいがないかチェックされ，なければ「0　Errors」と表示されます。



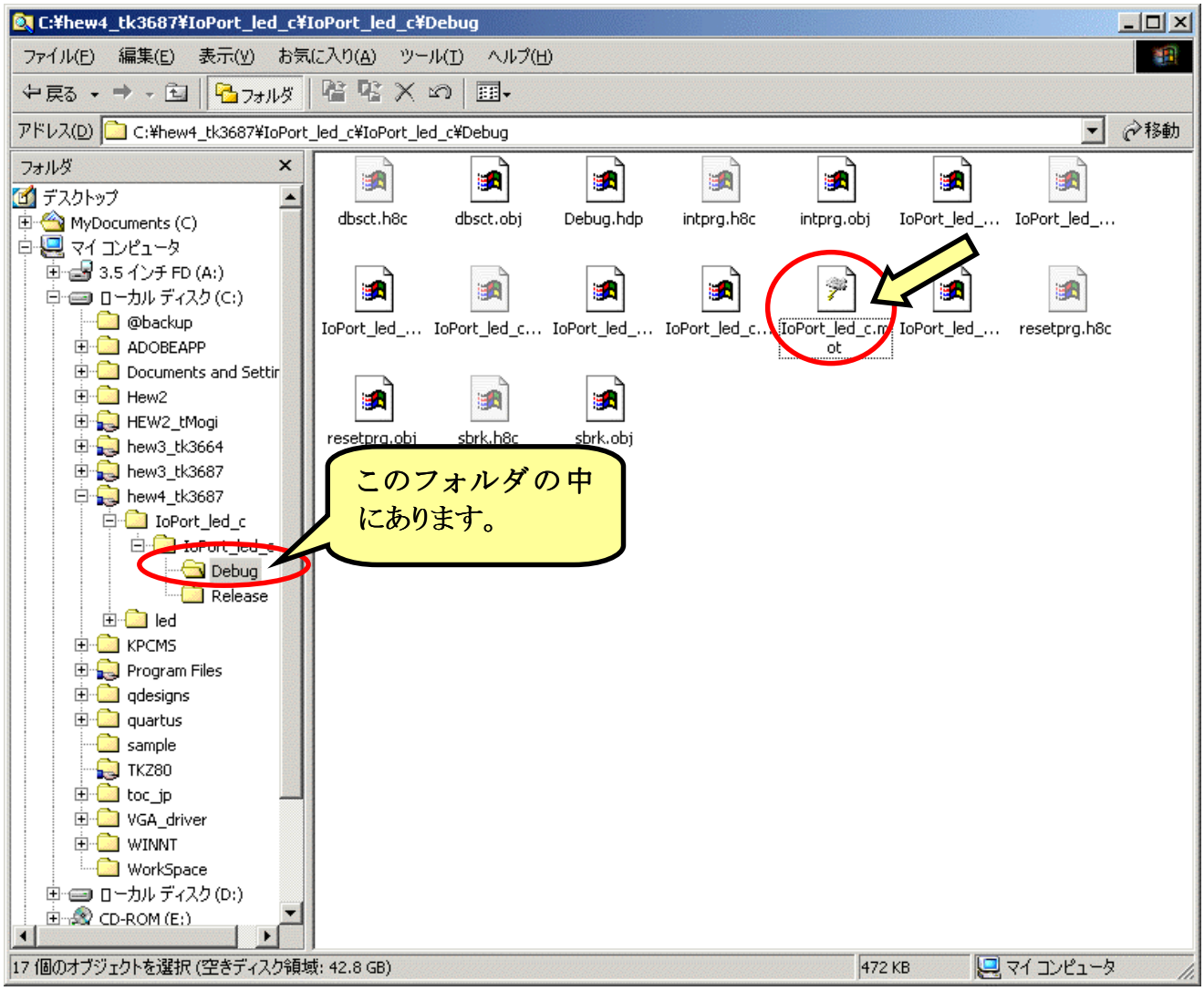
エラーがある場合はソースファイルを修正します。アウトプットウィンドウのエラー項目にマウスカーソルをあててダブルクリックすると，エラー行に飛んでいきます(このあたりの機能が統合化環境の良いところですね。)ソースファイルと前のページのリストを比べてまちがいなく入力しているかもう一度確認して下さい。

さて，図では「1　Warning」と表示されています。これは「まちがいではないかもしれないけど，念のため確認してね」という警告表示です。例えばこの図の「L1100(W) Cannot find "C" specifind in option "start"」は，C セクションを設定したのに C セクションのデータがないとき表示されます。今回のプログラムでは C セクションは使っていませんので，この警告が出ても何も問題ありません。

もっとも，Warninng の中には動作に影響を与えるものもあります。「H8S，H8/300 シリーズ C/C++コンパイラ，アセンブラ，最適化リンケージエディタ ユーザーズマニュアル」の 539 ページからコンパイラのエラーメッセージが，621 ページから最適化リンケージエディタのエラーメッセージが載せられていますので，問題ないか必ず確認して下さい。

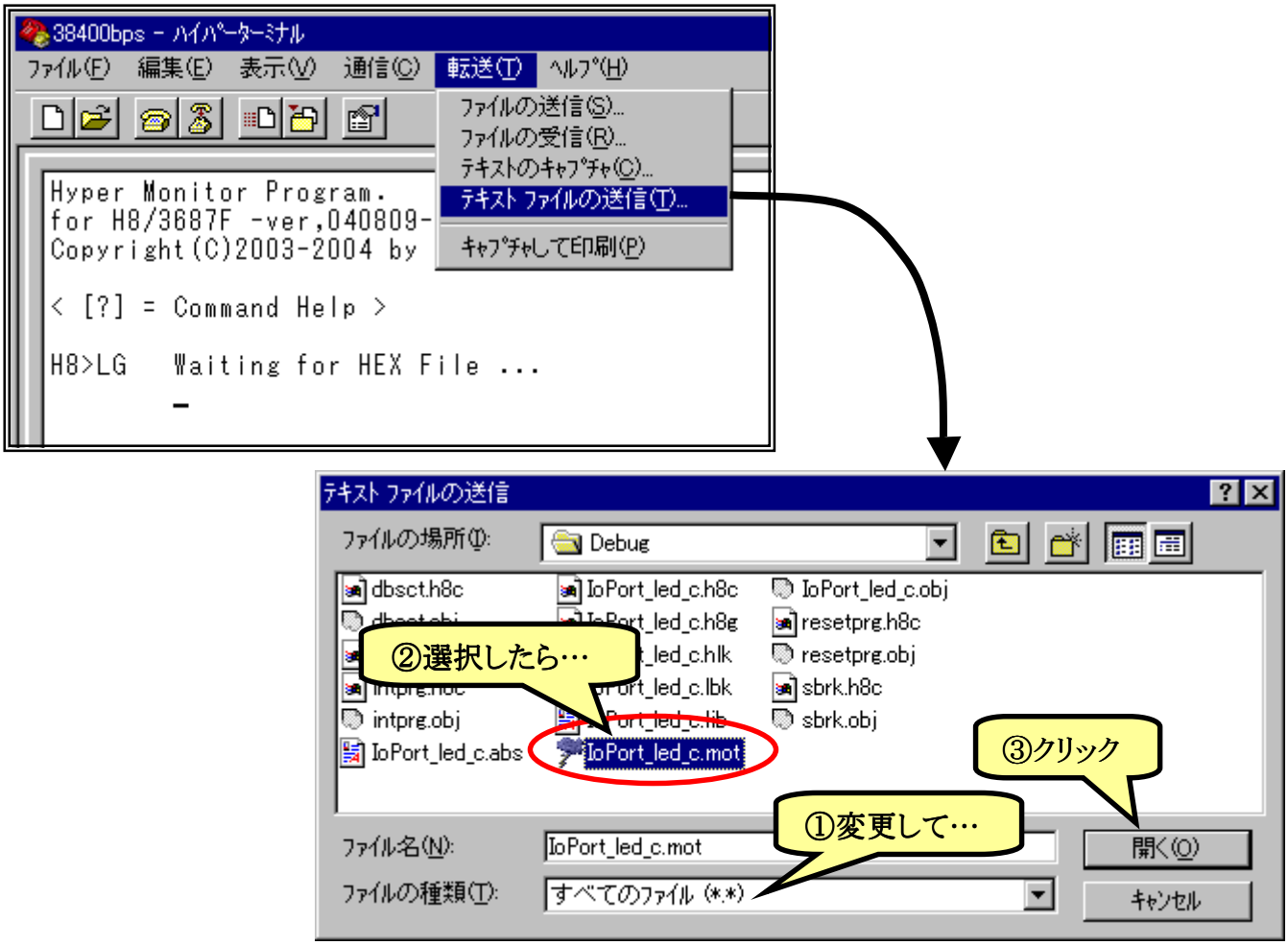
4. プログラムのダウンロードと実行

ビルドが成功すると C の命令がマシン語に変換され‘IoPort_led_c. mot’というファイルが作られます。拡張子が‘. mot’のファイルは「S タイプファイル」とか「mot ファイル」と呼ばれていて，マシン語の情報が含まれているファイルです。このファイルはプロジェクトフォルダの中の「Debug」フォルダ内に作られます。

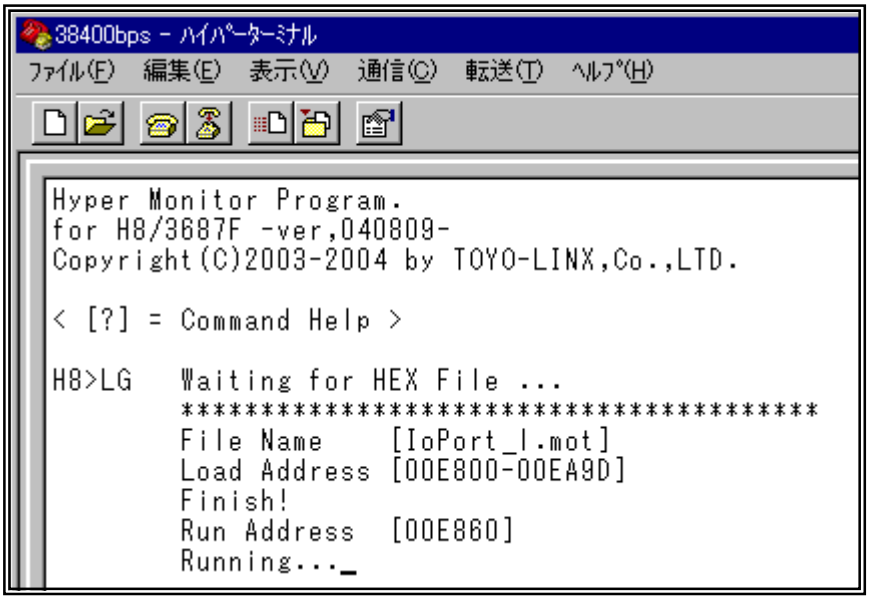


それでは実行してみましょう。ハイパーH8を起動して下さい。‘L’コマンドと‘G’コマンドを使います。ハイパーH8 はコマンドを続けて入力すると，その順番に実行することができます。今回はこれを使ってみましょう。パソコンのキーボードから‘LG’と入力して‘Enter’キーを押します。

次に，メニューの‘転送(T)’から‘テキストファイルの送信(T)’を選び，「テキストファイルの送信」ウィンドウを開きます。ファイルの種類を‘すべてのファイル’にして，‘IoPort_led_c．mot’を選びます。



ダウンロードが終了すると(プログラムが短いのであっという間です)，続いてロードしたプログラムを実行します。



いかがでしょうか。ちゃんと LED は点滅しましたか。うまく動作しないときはプログラムの入力ミスの可能性が大です。もう一度ちゃんと入力しているか確認してみてください。

第5章 内蔵周辺機能を使う

H8/3687 には様々な機能が内蔵されています。この章ではいくつかの内蔵周辺機能の使い方をマスターしましょう。なお，内蔵周辺機能の詳しい内容は「H8/3687 グループ ハードウェアマニュアル」（これからハードウェアマニュアルと呼びます）で説明されています。本書では説明されていない機能がたくさんありますので，ぜひお読みください。

1. I/O ポート

そもそも I/O とは何でしょうか。第 1 章では，

「外部から信号を入力したり外部機器をコントロールしたりするのが I/O です。」

と，簡単に説明しました。ここでは，もっと詳しく説明しましょう。

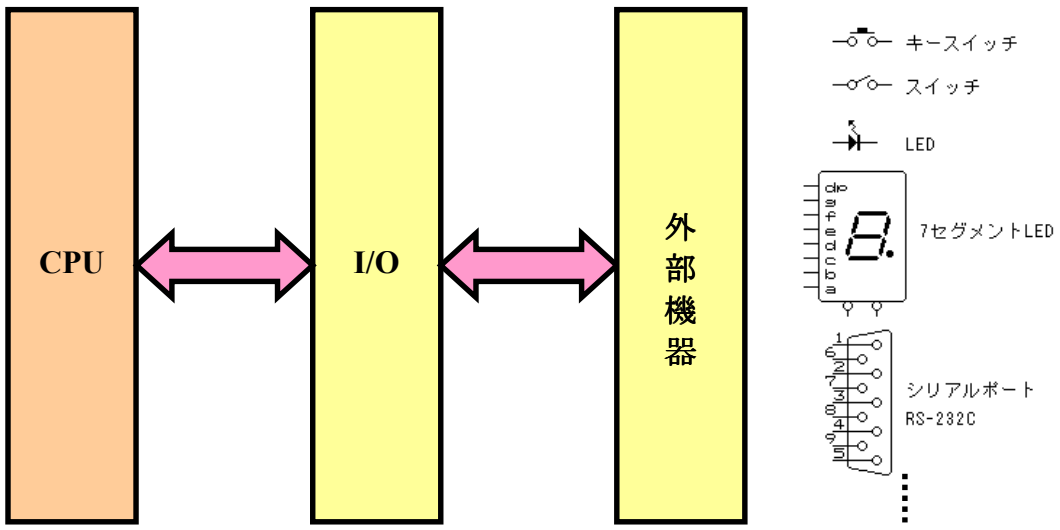
もともと CPU は「できるだけ速く」を合言葉に進歩してきました。H8/3687 は 1 つの命令を 0．1μs～1．2μs（μs:マイクロ秒は 1 秒の百万分の一）で実行できるように作られています。

一方，外部機器は速いものももちろんありますが，大抵はもっとのんびりしています。例えば LED の表示なんかはマイクロ秒単位で点滅しても人間の目にはわかりませんし，スイッチをマイクロ秒単位で入力しようとしても人間の指のスピードはそんなに速くなりません。

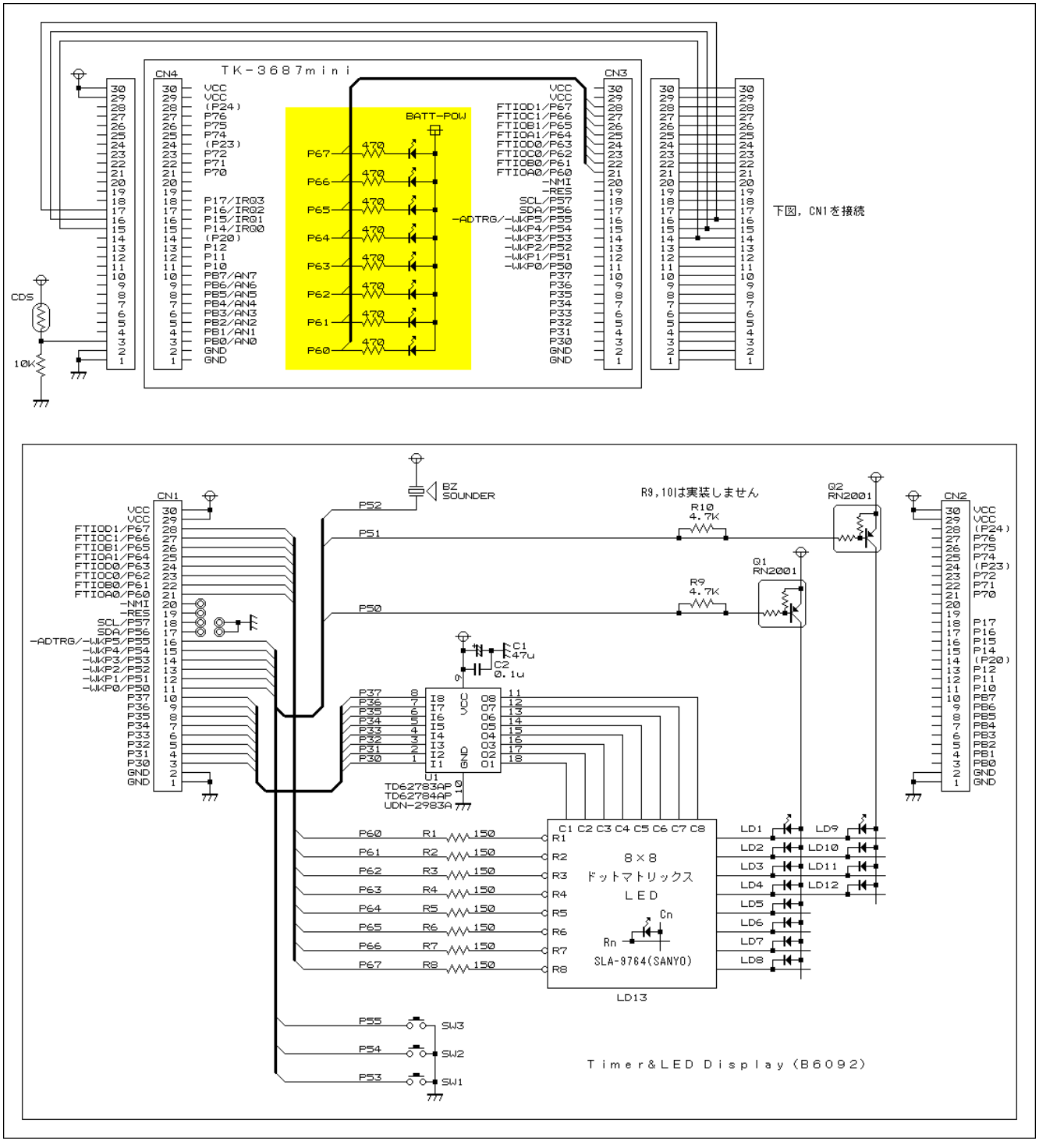
また，一般的に CPU は電圧が 5V で動いていますし，できるだけ少ない電流で動くように発展してきましたが，外部機器の中には 12V だったり電流がたくさん必要だったりするものがあります。

というわけで，CPU が，性格の異なる外部から信号を入力したり，外部機器をコントロールしたりするには，間に立ってデータを受け渡す役目が必要になります。この役目を果たすのが I/O になります。

第 1 章の「H8/3687 の内部ブロック図」からわかるように I/O にはいくつも種類がありますが，この節で取り上げている I/O ポートはパラレルポートと呼ばれるものです。以後，本節で I/O ポート，あるいはポートといえば，パラレルポートをさすものとします。



実習には，TK-3687mini，Timer＆LED　Display（B6092）と，この二つを接続するユニバーサル基板に実装されている CDS を利用します。回路図は次のようになります。



■ 設定用レジスタの説明

H8/3687 の端子はいくつもの機能が兼用されています。それで，各ピンをどの機能で使うか設定する必要があります。実習用 TK-3687mini はポート 5 にスイッチ，ポート 6 に LED がつながっています。それで，ポート 5 とポート 6 の設定用レジスタについて説明します。

ポート 5 に関係するレジスタは，PMR5，PCR5，PDR5，PUCR5 の 4 つです。

ポートモードレジスタ 5（PMR5） ： アドレス=0xFFE1 番地

ビット	ビット名	初期値	R/W	説明
7	POF57	0	R/W	P57 の出力形式の選択。 0：CMOS 出力。 / 1：NMOS オープンドレイン出力。
6	POF56	0	R/W	P56 の出力形式の選択。 0：CMOS 出力。 / 1：NMOS オープンドレイン出力。
5	WKP5	0	R/W	P55，WKP5，ADTRG 端子の機能の選択。 0：汎用入出力ポート。 / 1：WKP5，および，ADTRG 入力端子。
4	WKP4	0	R/W	P54，WKP4 端子の機能の選択。 0：汎用入出力ポート。 / 1：WKP4 入力端子。
3	WKP3	0	R/W	P53，WKP3 端子の機能の選択。 0：汎用入出力ポート。 / 1：WKP3 入力端子。
2	WKP2	0	R/W	P52，WKP2 端子の機能の選択。 0：汎用入出力ポート。 / 1：WKP2 入力端子。
1	WKP1	0	R/W	P51，WKP1 端子の機能の選択。 0：汎用入出力ポート。 / 1：WKP1 入力端子。
0	WKP0	0	R/W	P50，WKP0 端子の機能の選択。 0：汎用入出力ポート。 / 1：WKP0 入力端子。

ポートコントロールレジスタ 5（PCR5） ： アドレス=0xFFE8 番地

ビット	ビット名	初期値	R/W	説明
7	PCR57	0	W	各ビットの入出力の設定。 0：入力ポート。 / 1：出力ポート。
6	PCR56	0	W	
5	PCR55	0	W	
4	PCR54	0	W	
3	PCR53	0	W	
2	PCR52	0	W	
1	PVR51	0	W	
0	PCR50	0	W	

ポートデータレジスタ 5（PDR5） ： アドレス=0xFFD8 番地

ビット	ビット名	初期値	R/W	説明
7	P57	0	R/W	データレジスタ。入力ポートに設定されているビットの端子の状態を読み出したり，出力ポートに設定されている端子の出力値を格納する。
6	P56	0	R/W	
5	P55	0	R/W	
4	P54	0	R/W	
3	P53	0	R/W	
2	P52	0	R/W	
1	P51	0	R/W	
0	P50	0	R/W	

ポートプルアップコントロールレジスタ 5（PUCR5） ： アドレス=0xFFD1 番地				
ビット	ビット名	初期値	R/W	説明
7	－	0	－	各ビットのプルアップ MOS の設定。 0:プルアップしない。 / 1:プルアップする。
6	－	0	－	
5	PUCR55	0	R/W	
4	PUCR54	0	R/W	
3	PUCR53	0	R/W	
2	PUCR52	0	R/W	
1	PUCR51	0	R/W	
0	PUCR50	0	R/W	

ポート 6 に関係するレジスタは，PCR6，PDR6 の 2 つです。

ポートコントロールレジスタ 6（PCR6） ： アドレス=0xFFE9 番地				
ビット	ビット名	初期値	R/W	説明
7	PCR67	0	W	各ビットの入出力の設定。 0:入力ポート。 / 1:出力ポート。
6	PCR66	0	W	
5	PCR65	0	W	
4	PCR64	0	W	
3	PCR63	0	W	
2	PCR62	0	W	
1	PCR61	0	W	
0	PCR60	0	W	

ポートデータレジスタ 6（PDR6） ： アドレス=0xFFD9 番地				
ビット	ビット名	初期値	R/W	説明
7	P67	0	R/W	データレジスタ。入力ポートに設定されているビットの端子の状態を読み出したり，出力ポートに設定されている端子の出力値を格納する。
6	P66	0	R/W	
5	P65	0	R/W	
4	P64	0	R/W	
3	P63	0	R/W	
2	P62	0	R/W	
1	P61	0	R/W	
0	P60	0	R/W	

ところで，HEW でプロジェクトを作成すると，自動的に“iodefine．h”が生成されます。通常 H8/3687 の I/O にアクセスする際，“iodefine．h”で定義されている名称を使います。しかし，“iodefine．h”は構造体や共用体を駆使して定義されているため，最初はとっつきにくく感じるかもしれません。それでも慣れてくると非常に便利です。

“iodefine．h”で定義されている名称を使う場合，まずこのファイルをインクルードします。

```
#include "iodefine.h"   // 内蔵I/Oのラベル定義
```

一例として“iodefine．h”でポート 6 は次のように定義されています。(黄色で塗りつぶしている行はあとの説明で使う部分)

```
struct st_io {                                  /* struct IO    */
                                                ∫
            union {                             /* PDR6         */
                  unsigned char BYTE;           /*  Byte Access */
                  struct {                      /*  Bit  Access */
                        unsigned char B7:1;     /*    Bit 7     */
                        unsigned char B6:1;     /*    Bit 6     */
                        unsigned char B5:1;     /*    Bit 5     */
                        unsigned char B4:1;     /*    Bit 4     */
                        unsigned char B3:1;     /*    Bit 3     */
                        unsigned char B2:1;     /*    Bit 2     */
                        unsigned char B1:1;     /*    Bit 1     */
                        unsigned char B0:1;     /*    Bit 0     */
                        }      BIT;             /*              */
                  }            PDR6;            /*              */
                                                ∫
            unsigned char      PCR6;            /* PCR6         */
                                                ∫
};                                              /*              */
                                                ∫
#define IO      (*(volatile struct st_io    *)0xFFD0)   /* IO    Address*/
                                                ∫
```

それでは，ポートコントロールレジスタ 6(PCR6)に値をセットしてみましょう。ポート 6 は全ビット出力で使いますので，セットする値は 0xFF です。

I/O ポート: ポートコントロールレジスタ 6									
モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
IO	PCR6	(ビットアクセスは未定義)							

ソースリストから分かるように PCR6 は共用体を使ってビットアクセスするようには定義されていません。それで常に 1 バイト単位でアクセスします。ここでは PCR6 に 0xFF をセットするので次のように

記述します。

```
IO.PCR6 = 0xff;
```

次にポートデータレジスタ 6 に値をセットしましょう。セットする値は 55h としましょう。

I/O ポート: ポートデータレジスタ 6									
モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
IO	PDR6	B7	B6	B5	B4	B3	B2	B1	B0

ソースリストから分かるように PDR6 は共用体を使ってバイトアクセスとビットアクセスができるように定義しています。ここでは PDR6 に 55h をセットするのでバイトアクセスです。次のように記述します。

```
IO.PDR6.BYTE = 0x55;
```

さて，B0 だけを一時的に 0 にする場合はどうでしょうか。今度は特定のビットだけを書き替えるのでビットアクセスです。次のように記述します。

```
IO.PDR6.BIT.B0 = 0;
```

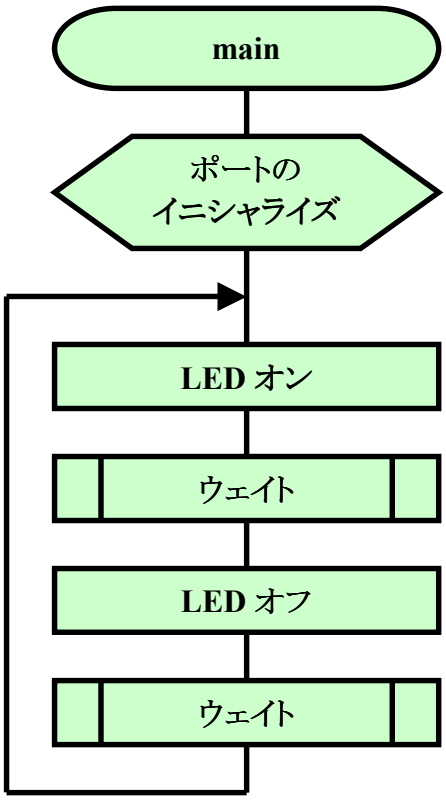
■ LED の点滅

4 章で実行した P60 につながっている LED の点滅について詳しく見てみましょう。まず，ポートの入出力を「ポートコントロールレジスタ 6（PCR6）」で設定します。P60 を出力にします。P61～67 は LED がつながっていますが今回は使いません。どちらでもよいのですが，とりあえず入力にしておきましょう。

	b7	b6	b5	b4	b3	b2	b1	b0
PCR6	0	0	0	0	0	0	0	1

あとは自由に LED を光らせることができます。ポート 6 の入出力は「ポートデータレジスタ 6（PDR6）」で行ないます。

では，プログラムを作ってみましょう。LED を点滅させるという，しごく簡単なプログラムです。ただ，今回はハイパーH8 の‘G’コマンドで動かしますので，CPU の速度で点滅させると人間の目には判別不能になります。それで，オンしたら少し待つ，オフしたら少し待つ，というのを繰り返すことにします。では，フローチャートを考えてみましょう。



ウェイトは単純ループで作ります。

では，コーディングしてみましょう。

```
/************************************************************************/
/*                                                                      */
/*  FILE        :IoPort_led_c.c                                         */
/*  DATE        :Wed, Apr 20, 2005                                      */
/*  DESCRIPTION :Main Program                                           */
/*  CPU TYPE    :H8/3687                                                */
/*                                                                      */
/*  This file is programed by TOYO-LINX Co.,Ltd. / yKikuchi             */
/*                                                                      */
/************************************************************************/

/************************************************************************
    インクルードファイル
************************************************************************/
#include <machine.h>    // H8特有の命令を使う
#include "iodefine.h"   // 内蔵I/Oのラベル定義

/************************************************************************
    関数の定義
************************************************************************/
void     main(void);
void     wait(void);

/************************************************************************
    メインプログラム
************************************************************************/
void main(void)
{
    IO.PCR6 = 0x01;             // ポート6のbit0(P60)を出力に設定

    while(1){
        IO.PDR6.BIT.B0 = 0;     // LEDオン
        wait();
        IO.PDR6.BIT.B0 = 1;     // LEDオフ
        wait();
    }
}

/************************************************************************
    ウェイト
************************************************************************/
void wait(void)
{
    unsigned long i;

    for (i=0;i<1666666;i++){}
}
```

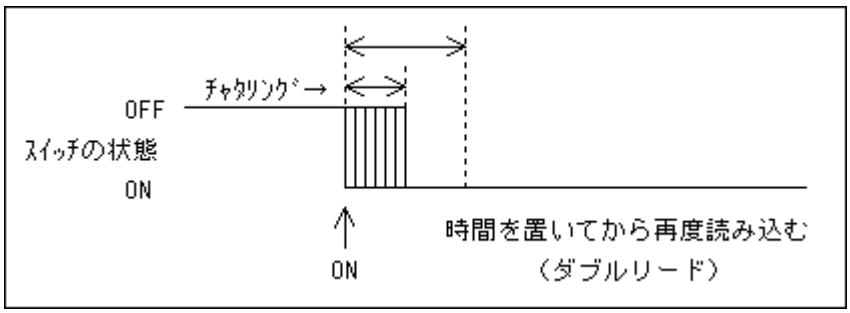
このプログラムは 4 章でお試しで作ったものと全く同じです。どのような考え方で作られたのかわかると興味が湧いてきませんか。

■ スイッチの入力

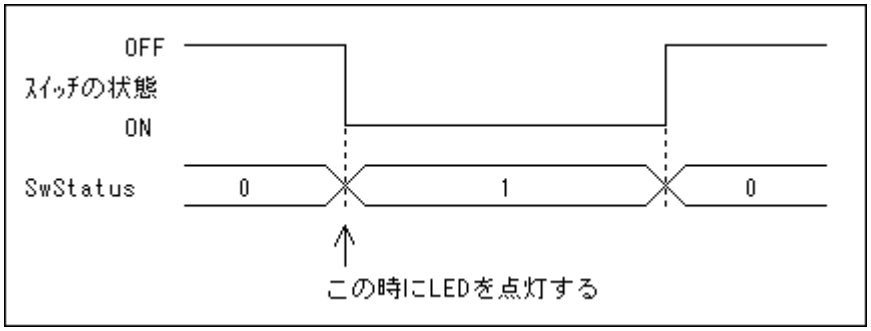
次は入力ポートの例として，プッシュスイッチの入力を考えてみましょう。スイッチが押された瞬間だけ，0．2 秒間（200ms）LED が光る（ワンショット動作），というプログラムを作ります。プッシュスイッチは P53 につながっているものを使います。LED は P60 です。

さて，プッシュスイッチを入力するとき最初に考えなければならないのはチャタリングの除去です。スイッチをオンにするというのは，おおざっぱに言えば金属と金属をぶつけることです。そのため，押した瞬間，金属の接点がバウンドしてオンとオフが繰り返されます。これをチャタリングと呼びます。数 ms の間だけなのですが，マイコンにしてみれば十分長い時間です。そのため，単純に入力すると，このオンとオフをすべて読んでしまって，何度もスイッチが押されたとかんちがいしてしまいます。

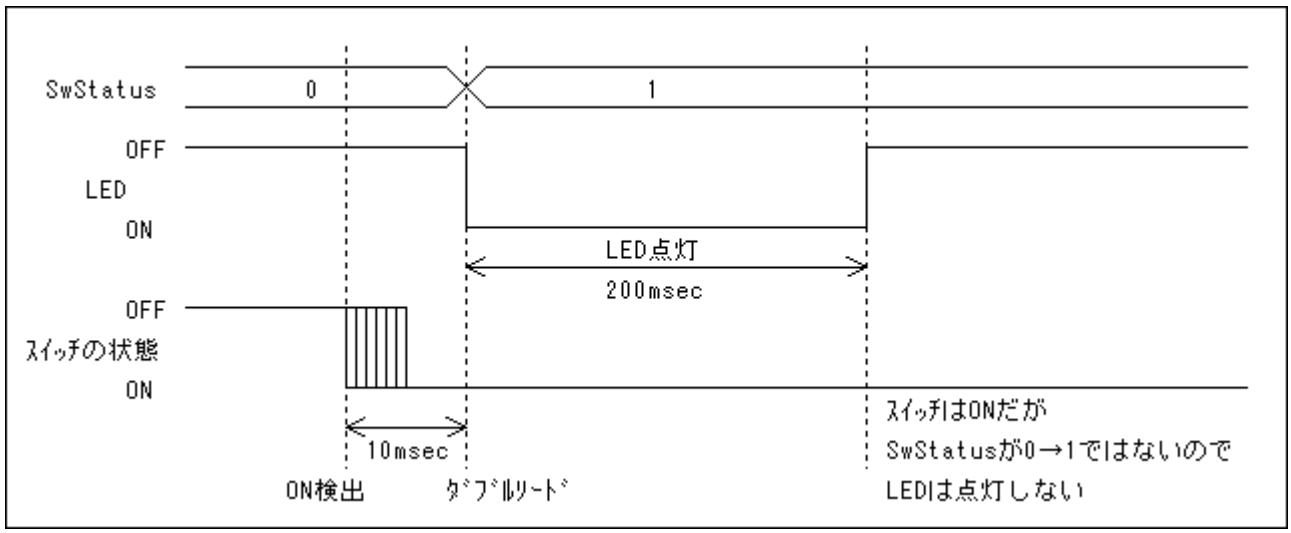
チャタリングを取り除くために，スイッチがオンしたら，しばらく待ってから（10ms ぐらい）もう一度読む（ダブルリード），ということを行ないます。2 度目に読んだときもオンだったら本当にスイッチがオンしたと見なします。



次はワンショット動作について考えてみましょう。スイッチがオンになった瞬間だけを検出するためにスイッチの状態をおぼえておくことにします。変数として‘SwStatus’を B セクションに用意し，SwStatus=0 のときはスイッチが押されていない，SwStatus=1 のときはスイッチが押されている，ということにします。スイッチが押された瞬間を検出するので，SwStatus が 0 から 1 に変化したときに LED を点灯します。



以上のことを考えてタイミングチャートをかいてみましょう。これを見ながらコーディングしていきます。



```
/************************************************************************/
/*                                                                      */
/*  FILE        :IoPort_sw_led_c.c                                      */
/*  DATE        :Wed, Jan 09, 2008                                      */
/*  DESCRIPTION :Main Program                                           */
/*  CPU TYPE    :H8/3687                                                */
/*                                                                      */
/*  This file is programed by TOYO-LINX Co.,Ltd. / yKikuchi             */
/*                                                                      */
/************************************************************************/

/************************************************************************
    インクルードファイル
************************************************************************/
#include <machine.h>    // H8特有の命令を使う
#include "iodefine.h"   // 内蔵I/Oのラベル定義

/************************************************************************
    グローバル変数の定義とイニシャライズ(RAM)
************************************************************************/
unsigned char  SwStatus  =    0;   //スイッチの状態

/************************************************************************
    関数の定義
************************************************************************/
void      main(void);
void      wait10(void);
void      wait200(void);

/************************************************************************
    メインプログラム
************************************************************************/
void main(void)
{
    IO.PUCR5.BYTE = 0x08;    // ポート5プルアップ抵抗の設定
    IO.PCR5 = 0x00;          // ポート5を入力に設定
    IO.PCR6 = 0x01;          // ポート6のbit0(P60)を出力に設定
    IO.PDR6.BIT.B0 = 1;      // LEDオフ

    while(1){
        if (IO.PDR5.BIT.B3==0){              //スイッチオン
            wait10();                        //ちょっと待つ
            if (IO.PDR5.BIT.B3==0){          //やっぱりスイッチオン
                if (SwStatus==0){            //今までスイッチオフだった
                    SwStatus = 1;            //スイッチオンを記憶
                    IO.PDR6.BIT.B0 = 0;      //LEDオン
                    wait200();               //しばらく待つ
                    IO.PDR6.BIT.B0 = 1;      //LEDオフ
                }
            }
            else{                            //スイッチオフだった
                SwStatus = 0;
            }
```

```
        }
        else{                                   //スイッチオフ
            SwStatus = 0;
        }
    }
}

/*****************************************************************************
    ウェイト
*****************************************************************************/
void wait10(void)
{
    unsigned long I;

    for (I=0;I<33333;I++){}
}

void wait200(void)
{
    unsigned long I;

    for (I=0;I<666666;I++){}
}
```

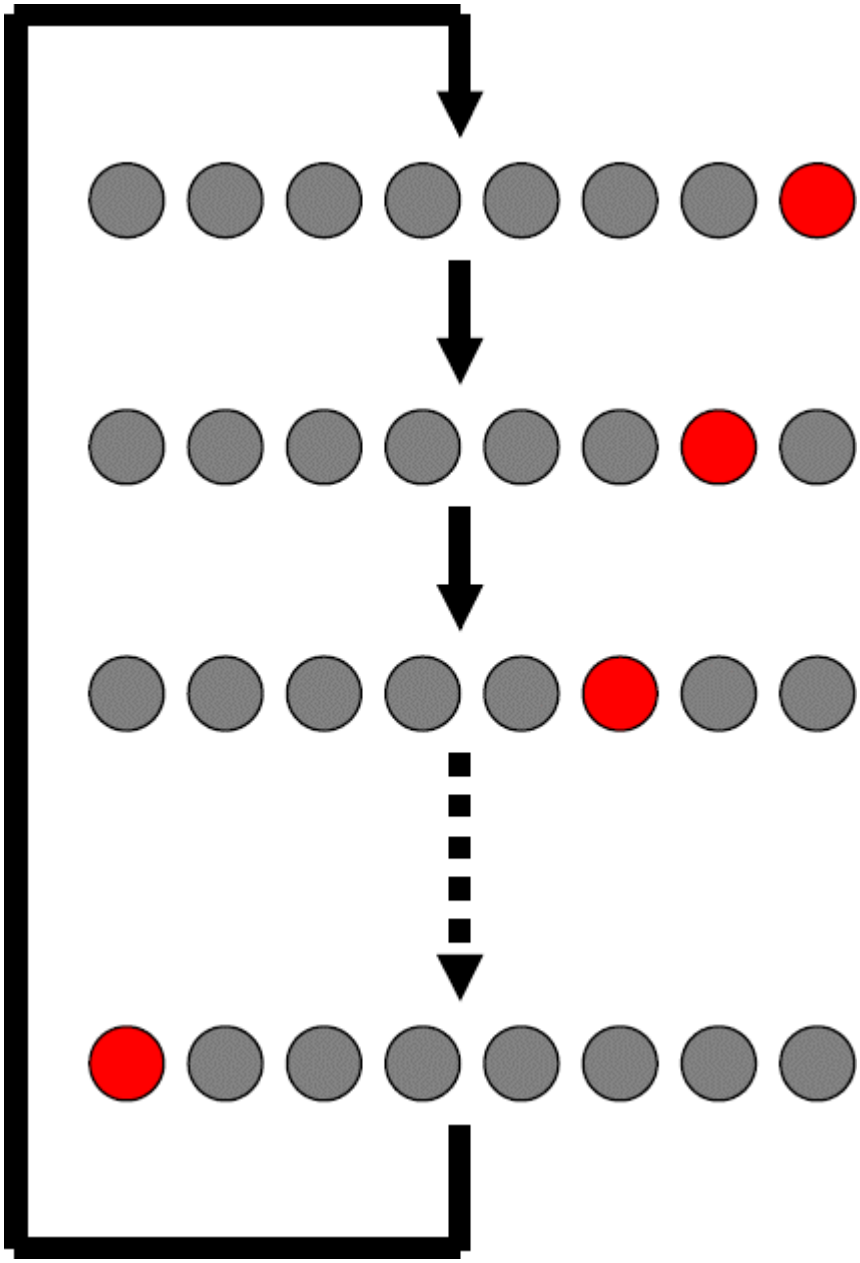
付属の CD-ROM にはあらかじめダウンロードするファイルがおさめられています。

‘Io_Port_sw_led_c. mot’

をダウンロードして実行してください.

■ **練習問題(1)**

次のように LED が点灯するプログラムを作りなさい。(解答例は「IoPort_led_rotate_c」)



■ **練習問題(2)**

スイッチが押されるたびに，練習問題(1)の LED の点灯方向が逆になるプログラムを作りなさい。(解答例は「IoPort_sw_led_rotate_c」)

2. 外部割込み

基本的にマイコンはプログラムに従って一連の作業を順番に実行していきます。しかし，マイコンにセンサやスイッチをつないで，「センサから入力があったらこの処理を行なう」とか，「スイッチが押されたらあの処理を行なう」というようにプログラムする場合，いつ入力があるかわからないため，センサやスイッチの状態をいつも監視していなければなりません。マイコンが普段ひまだったり，多少反応が遅くなってもよかったりするのであれば，そういう方法でもよいのですが，いろいろな処理を行ないながら，いざ入力があったときは優先して処理しなければならないとなると，別の方法を考えなえればなりません。

このようなときに使うのが外部割込みです。H8/3687 には IRQ0，IRQ1，IRQ2，IRQ3 という 4 つの端子が用意されていて，それぞれの端子の信号の変化によって割込みをかけ，特定の処理を実行することができます。割込みとはハードウェア的にサブルーチンを実行する方法，とも言えるでしょう。

■ 割込み処理の概要

割込み要求は H8/3687 に内蔵されている割込みコントローラによって制御されます。外部割込みの割込み動作は下記のとおりです。

表 3.1　例外処理要因とベクタアドレス

発生元	例外処理要因	ベクタ番号	ベクタアドレス	優先度
RES 端子 ウォッチドッグタイマ	リセット	0	H'0000～H'0001	高
—	システム予約	1～6	H'0002～H'000D	
外部割り込み端子	NMI	7	H'000E～H'000F	
CPU	トラップ命令 #0	8	H'0010～H'0011	
	トラップ命令 #1	9	H'0012～H'0013	
	トラップ命令 #2	10	H'0014～H'0015	
	トラップ命令 #3	11	H'0016～H'0017	
アドレスブレーク	ブレーク条件成立	12	H'0018～H'0019	
CPU	スリープ命令の実行による直接遷移	13	H'001A～H'001B	
外部割り込み端子	IRQ0 低電圧検出割り込み*	14	H'001C～H'001D	
	IRQ1	15	H'001E～H'001F	
	IRQ2	16	H'0020～H'0021	
	IRQ3	17	H'0022～H'0023	
	WKP	18	H'0024～H'0025	
RTC	オーバフロー	19	H'0026～H'0027	
—	システム予約	20	H'0028～H'0029	
タイマ V	コンペアマッチ A コンペアマッチ B オーバフロー	22	H'002C～H'002D	
SCI3	受信データフル 送信データエンプティ 送信終了 受信エラー	23	H'002E～H'002F	
IIC2	送信データエンプティ、送信終了、受信データフル、 アービトレーションロスト／オーバランエラー NACK 検出、停止条件検出	24	H'0030～H'0031	
A/D 変換器	A/D 変換終了	25	H'0032～H'0033	
タイマ Z	コンペアマッチ／インプットキャプチャ A0～D0 オーバフロー	26	H'0034～H'0035	
	コンペアマッチ／インプットキャプチャ A1～D1 オーバフロー、アンダフロー	27	H'0036～H'0037	
タイマ B1	オーバフロー	29	H'003A～H'003B	
SCI3_2	受信データフル 送信データエンプティ 送信終了 受信エラー	32	H'0040～H'0041	低

【注】　*　低電圧検出割り込みは、パワーオンリセット＆低電圧検出回路内蔵版のみ有効です。

1. IRQ0～3 端子に入力エッジがあると割込みコントローラに割込み要求信号を送ります。(エッジの方向は個別に選択可能)
2. 割込みコントローラは CPU に対して割込み処理を要求します。
3. CPU は割込みを受け付けると，実行中の命令を実行したあと，プログラムカウンタと CCR (コンディションコードレジスタ) をスタックにプッシュします。
4. CPU は CCR の I ビットを 1 にセットして他の割込みを受け付けないようにします。
5. CPU は外部割込み要求に対するベクタアドレス (IRQ0 のとき 0x001C，IRQ1 のとき 0x001E，IRQ2 のとき 0x0020，IRQ3 のとき 0x0022) を生成し，そのベクタアドレスにセットされているデータ (割込み処理のスタートアドレス) をプログラムカウンタにセットします。
6. 割込み処理がスタートします。

■ 外部割込みに使用するコントロールレジスタ

割込みを制御するレジスタの内，外部割込み IRQ0～3 に関係するレジスタは次のとおりです。なお，外部割込みに関係するビットは黄色でマークしています。

割込みエッジセレクトレジスタ 1（IEGR1） ： アドレス=0xFFF2 番地				
ビット	ビット名	初期値	R/W	説明
7	NMIEG	0	R/W	NMI エッジセレクト。 0：NMI 入力端子の立ち下がりエッジを検出。 1：NMI 入力端子の立ち上がりエッジを検出。
6	－	1	－	リザーブビット。リードすると常に 1 が読み出される。
5	－	1	－	リザーブビット。リードすると常に 1 が読み出される。
4	－	1	－	リザーブビット。リードすると常に 1 が読み出される。
3	IEG3	0	R/W	IRQ3 エッジセレクト。 0：IRQ3 入力端子の立ち下がりエッジを検出。 1：IRQ3 入力端子の立ち上がりエッジを検出。
2	IEG2	0	R/W	IRQ2 エッジセレクト。 0：IRQ2 入力端子の立ち下がりエッジを検出。 1：IRQ2 入力端子の立ち上がりエッジを検出。
1	IEG1	0	R/W	IRQ1 エッジセレクト。 0：IRQ1 入力端子の立ち下がりエッジを検出。 1：IRQ1 入力端子の立ち上がりエッジを検出。
0	IEG0	0	R/W	IRQ0 エッジセレクト。 0：IRQ0 入力端子の立ち下がりエッジを検出。 1：IRQ0 入力端子の立ち上がりエッジを検出。

割込みイネーブルレジスタ 1（IENR1） ： アドレス=0xFFF4 番地				
ビット	ビット名	初期値	R/W	説明
7	IENDT	0	R/W	直接遷移割込み要求イネーブル。 0：ディセーブル。 / 1：イネーブル。
6	IENTA	0	R/W	RTC 割込み要求イネーブル。 0：ディセーブル。 / 1：イネーブル。
5	IENWP	0	R/W	ウェイクアップ割込み要求イネーブル。（WKP0～5 共通） 0：ディセーブル。 / 1：イネーブル。
4	－	1	－	リザーブビット。リードすると常に 1 が読み出される。
3	IEN3	0	R/W	IRQ3 割込み要求イネーブル。 0：ディセーブル。 / 1：イネーブル。
2	IEN2	0	R/W	IRQ2 割込み要求イネーブル。 0：ディセーブル。 / 1：イネーブル。
1	IEN1	0	R/W	IRQ1 割込み要求イネーブル。 0：ディセーブル。 / 1：イネーブル。
0	IEN0	0	R/W	IRQ0 割込み要求イネーブル。 0：ディセーブル。 / 1：イネーブル。

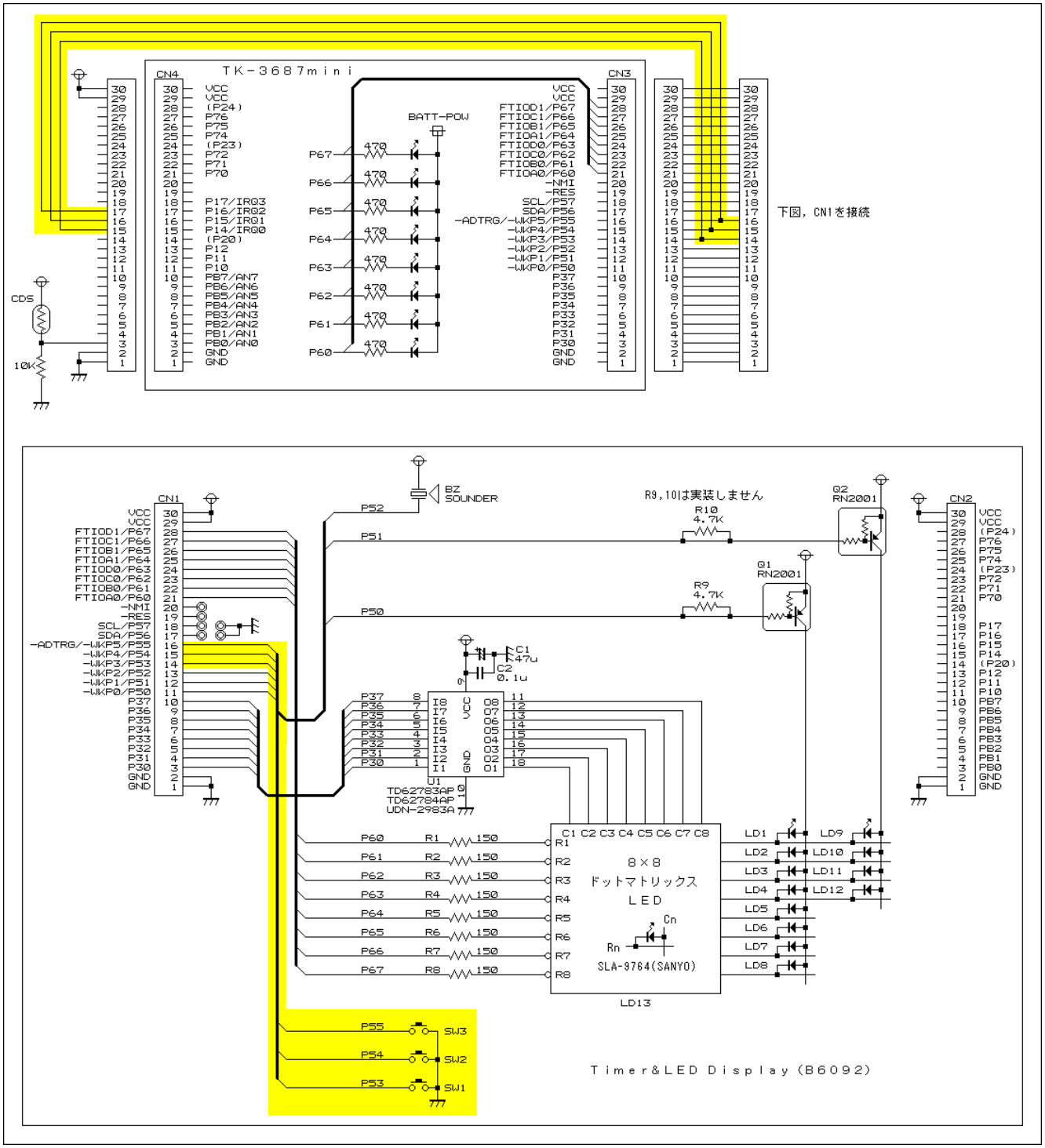
割込みフラグレジスタ 1（IRR1） ： アドレス=0xFFF6 番地				
ビット	ビット名	初期値	R/W	説明
7	IRRDT	0	R/W	直接遷移割込み要求フラグ。 ［セット条件］SYSCR2 の DTON に 1 をセットした状態でスリープ命令を実行し直接遷移したとき。 ［クリア条件］0 をライトしたとき。
6	IRRTA	0	R/W	RTC 割込み要求フラグ。 ［セット条件］RTC がオーバーフローしたとき。 ［クリア条件］0 をライトしたとき。
5	－	1	－	リザーブビット。リードすると常に 1 が読み出される。
4	－	1	－	リザーブビット。リードすると常に 1 が読み出される。
3	IRRI3	0	R/W	IRQ3 割込み要求フラグ。 ［セット条件］IRQ3 が割込み入力に設定され，指定されたエッジを検出したとき。 ［クリア条件］0 をライトしたとき。
2	IRRI2	0	R/W	IRQ2 割込み要求フラグ。 ［セット条件］IRQ2 が割込み入力に設定され，指定されたエッジを検出したとき。 ［クリア条件］0 をライトしたとき。
1	IRRI1	0	R/W	IRQ1 割込み要求フラグ。 ［セット条件］IRQ1 が割込み入力に設定され，指定されたエッジを検出したとき。 ［クリア条件］0 をライトしたとき。
0	IRRI0	0	R/W	IRQ0 割込み要求フラグ。 ［セット条件］IRQ0 が割込み入力に設定され，指定されたエッジを検出したとき。 ［クリア条件］0 をライトしたとき。

IRQ0～3 はポート 1 の P14～17 と兼用ピンになっています。それで，IRQ0～3 として使うように指定しなければなりません。次のレジスタで指定します。

ポートモードレジスタ 1（PMR1） ： アドレス=0xFFE0 番地				
ビット	ビット名	初期値	R/W	説明
7	IRQ3	0	R/W	P17，IRQ3，TRGV 端子の機能の選択。 0：汎用入出力ポート。 / 1：IRQ3，および，TRGV 入力端子。
6	IRQ2	0	R/W	P16，IRQ2 端子の機能の選択。 0：汎用入出力ポート。 / 1：IRQ2 入力端子。
5	IRQ1	0	R/W	P15，IRQ1，TMIB1 端子の機能の選択。 0：汎用入出力ポート。 / 1：IRQ1，および，TMIB 入力端子。
4	IRQ0	0	R/W	P14，IRQ0 端子の機能の選択。 0：汎用入出力ポート。 / 1：IRQ0 入力端子。
3	TXD2	0	R/W	P72，TXD_2 端子の機能の選択。 0：汎用入出力ポート。 / 1：TXD_2 出力端子。
2	PWM	0	R/W	P11，PWM 端子の機能の選択。 0：汎用入出力ポート。 / 1：PWM 出力端子。
1	TXD	0	R/W	P22，TXD 端子の機能の選択。 0：汎用入出力ポート。 / 1：TXD 出力端子。
0	TMOW	0	R/W	P10，TMOW 端子の機能の選択。 0：汎用入出力ポート。 / 1：TMOW 出力端子。

■ 回路図

Timer＆LED Display（B6092）の SW1（右のスイッチ）は H8/3687 の P53 につながっていますが，ユニバーサル基板で P14 にもつながっています。P14 は IRQ0 と兼用になっています。同じように SW2 は P54・P15・IRQ1 に，SW3 は P55・P16・IRQ2 につながっています。今回は IRQ0，1，2 割込みを使ってみましょう。回路図を再掲します。



■ プログラムの作成

作成するプログラムは「SW1 が押されるたびにポート 6 の表示をインクリメント, SW2 が押されるたびにポート 6 の表示をデクリメント, SW3 が押されるたびに表示をローテートする」というものです。

スイッチオンの検知は外部割込みで行ないます。スイッチオンで信号は High から Low になりますから, 立ち下がりエッジで割込みをかければよいわけです。ソースリストは次のようになります。

```
/************************************************************************/
/*                                                                      */
/*  FILE        :irq_c.c                                                */
/*  DATE        :Thu, Jan 10, 2008                                      */
/*  DESCRIPTION :Main Program                                           */
/*  CPU TYPE    :H8/3687                                                */
/*                                                                      */
/*  This file is programed by TOYO-LINX Co.,Ltd. / yKikuchi             */
/*                                                                      */
/************************************************************************/

/************************************************************************
    インクルードファイル
************************************************************************/
#include <machine.h>    // H8特有の命令を使う
#include "iodefine.h"   // 内蔵I/Oのラベル定義

/************************************************************************
    グローバル変数の定義とイニシャライズ(RAM)
************************************************************************/
unsigned char  DispData = 0x00;    //表示データ

/************************************************************************
    関数の定義
************************************************************************/
void      intprog_irq0(void);
void      intprog_irq1(void);
void      intprog_irq2(void);
void      main(void);

/************************************************************************
    メインプログラム
************************************************************************/
void main(void)
{
    // イニシャライズ ------------------------------------------------
    IO.PMR1.BYTE = 0x72;     // P14-16をIRQ0,1,2に接続
    IO.PCR1 = 0x00;          // ポート1を入力に設定

    IO.PUCR5.BYTE = 0x38;    // ポート5プルアップ抵抗の設定
    IO.PCR5 = 0x00;          // ポート5を入力に設定

    IO.PCR6 = 0xff;          // ポート6を出力に設定

    IEGR1.BYTE = 0x00;       // IRQ0-2 立ち下がりエッジ
    IRR1.BYTE = 0x00;        // IRQ0-2 割込み要求フラグクリア
    IENR1.BYTE = 0x07;       // IRQ0-2 割込みイネーブル
```

```
    // メインループ ----------------------------------------------------
    while(1){
        IO.PDR6.BYTE = ~DispData;    // ポート6に表示する
    }
}

/************************************************************************
    IRQ0 割込み
************************************************************************/
#pragma regsave (intprog_irq0)
void intprog_irq0(void)
{
    DispData++;                      // On → インクリメント
    IRR1.BIT.IRRI0 = 0;              // 割込み要求フラグクリア
}

/************************************************************************
    IRQ1 割込み
************************************************************************/
#pragma regsave (intprog_irq1)
void intprog_irq1(void)
{
    DispData--;                      // On → デクリメント
    IRR1.BIT.IRRI1 = 0;              // 割込み要求フラグクリア
}

/************************************************************************
    IRQ2 割込み
************************************************************************/
#pragma regsave (intprog_irq2)
void intprog_irq2(void)
{
    DispData = rotlc(1,DispData);    // On → 左ローテート
    IRR1.BIT.IRRI2 = 0;              // 割込み要求フラグクリア
}
```

割込みを使うためにはソースファイルだけではなく, HEW が自動生成する‘intprg. c’を修正する必要があります。下記のリストをご覧ください。

```
/************************************************************************/
/*                                                                      */
/*  FILE        :intprg.c                                               */
/*  DATE        :Thu, Jan 10, 2008                                      */
/*  DESCRIPTION :Interrupt Program                                      */
/*  CPU TYPE    :H8/3687                                                */
/*                                                                      */
/*  This file is generated by Renesas Project Generator (Ver.4.9).     */
/*                                                                      */
/************************************************************************/



#include <machine.h>
```

```
extern void intprog_irq0(void);
extern void intprog_irq1(void);
extern void intprog_irq2(void);

#pragma section IntPRG
//  vector 1 Reserved

//  vector 2 Reserved

//  vector 3 Reserved

//  vector 4 Reserved

//  vector 5 Reserved

//  vector 6 Reserved

//  vector 7 NMI
__interrupt(vect=7) void INT_NMI(void) {/* sleep(); */}
//  vector 8 TRAP #0
__interrupt(vect=8) void INT_TRAP0(void) {/* sleep(); */}
//  vector 9 TRAP #1
__interrupt(vect=9) void INT_TRAP1(void) {/* sleep(); */}
//  vector 10 TRAP #2
__interrupt(vect=10) void INT_TRAP2(void) {/* sleep(); */}
//  vector 11 TRAP #3
__interrupt(vect=11) void INT_TRAP3(void) {/* sleep(); */}
//  vector 12 Address break
__interrupt(vect=12) void INT_ABRK(void) {/* sleep(); */}
//  vector 13 SLEEP
__interrupt(vect=13) void INT_SLEEP(void) {/* sleep(); */}
//  vector 14 IRQ0
__interrupt(vect=14) void INT_IRQ0(void) {intprog_irq0();}
//  vector 15 IRQ1
__interrupt(vect=15) void INT_IRQ1(void) {intprog_irq1();}
//  vector 16 IRQ2
__interrupt(vect=16) void INT_IRQ2(void) {intprog_irq2();}
//  vector 17 IRQ3
__interrupt(vect=17) void INT_IRQ3(void) {/* sleep(); */}
//  vector 18 WKP
__interrupt(vect=18) void INT_WKP(void) {/* sleep(); */}
//  vector 19 RTC
__interrupt(vect=19) void INT_RTC(void) {/* sleep(); */}
//  vector 20 Reserved

//  vector 21 Reserved

//  vector 22 Timer V
__interrupt(vect=22) void INT_TimerV(void) {/* sleep(); */}
//  vector 23 SCI3
__interrupt(vect=23) void INT_SCI3(void) {/* sleep(); */}
//  vector 24 IIC2
__interrupt(vect=24) void INT_IIC2(void) {/* sleep(); */}
//  vector 25 ADI
```

追加

修正

修正

修正

```
__interrupt(vect=25) void INT_ADI(void) {/* sleep(); */}
//  vector 26 Timer Z0
__interrupt(vect=26) void INT_TimerZ0(void) {/* sleep(); */}
//  vector 27 Timer Z1
__interrupt(vect=27) void INT_TimerZ1(void) {/* sleep(); */}
//  vector 28 Reserved

//  vector 29 Timer B1
__interrupt(vect=29) void INT_TimerB1(void) {/* sleep(); */}
//  vector 30 Reserved

//  vector 31 Reserved

//  vector 32 SCI3_2
__interrupt(vect=32) void INT_SCI3_2(void) {/* sleep(); */}
```

では，ビルドして実行してみましょう。期待通り動作するでしょうか。なお，付属の CD-ROM にはあらかじめダウンロードするファイルがおさめられています。

‘irq_c．mot’

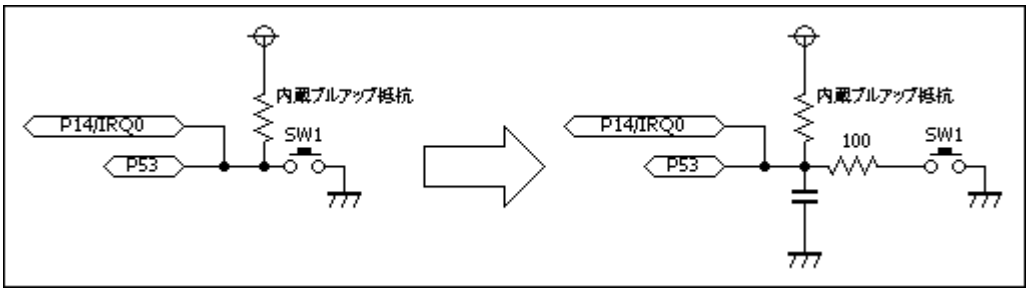
をダウンロードして実行してください。

■ 練習問題

おそらく，スイッチを押したときに+1 だけではなく，不規則に+2 や+3 になったりすると思います。これはスイッチのチャタリングの影響です。では，チャタリングを取り除く方法を考えてみてください。（解答例は「irq_c_v2」）

♭ ♯ ♪ ♭ ♯ ♪

ところで，解答例「irq_c_v2．c」はプログラムだけでチャタリングを除去する方法です。ただ，割込みプログラムの中にチャタリング除去のためのウェイトが入っているため，他の処理が行なえなくなってしまいます。なので，ハードウェアでチャタリングを除去することを考えてもよいでしょう。回路は次のようになります。



抵抗とコンデンサを1 個ずつ追加するだけです。コンデンサは使うスイッチにあわせてチャタリングが除去できる容量にします。この抵抗とコンデンサでローパスフィルタを作り，波形をなまらせてチャタリングを吸収します。プログラムは「irq_c．c」を使います。

もちろん，コンデンサを追加せずに，なおかつ他の処理に影響を与えずにプログラムでチャタリングを除去する方法もあります。これについては「6．ADコンバータ」のプログラムをご覧ください。

3. タイマ V

マイコンの内蔵機能で I/O ポートと並んでよく使われる機能はタイマです。出力波形を作ったり，タイミングを作ったりします。H8/3687 には 3 つの汎用タイマが内蔵されています。ここでは，そのうちの一つ，タイマ V を使ってみましょう。

■ タイマ V の概要

タイマ V は 8 ビットカウンタをベースにした 8 ビットタイマです。タイマ V に内蔵されているタイマカウンタ V（TCNTV）は CPU クロック（TK-3687 の場合は 20MHz）を分周したクロック（1/128，1/64，1/32，1/16，1/8，1/4 のいずれか選択可能）によって常に+1 されます。TCNTV はタイムコンスタントレジスタ A（TCORA）とタイムコンスタントレジスタ B（TCORB）と比較されており，一致すると CPU に割り込みをかけることができます。その際，TCNTV を 0 にクリアするよう設定することもできます。

なお，タイマ V の詳細については「H8/3687 シリーズハードウェアマニュアル」の「12. タイマ V」をご覧下さい。

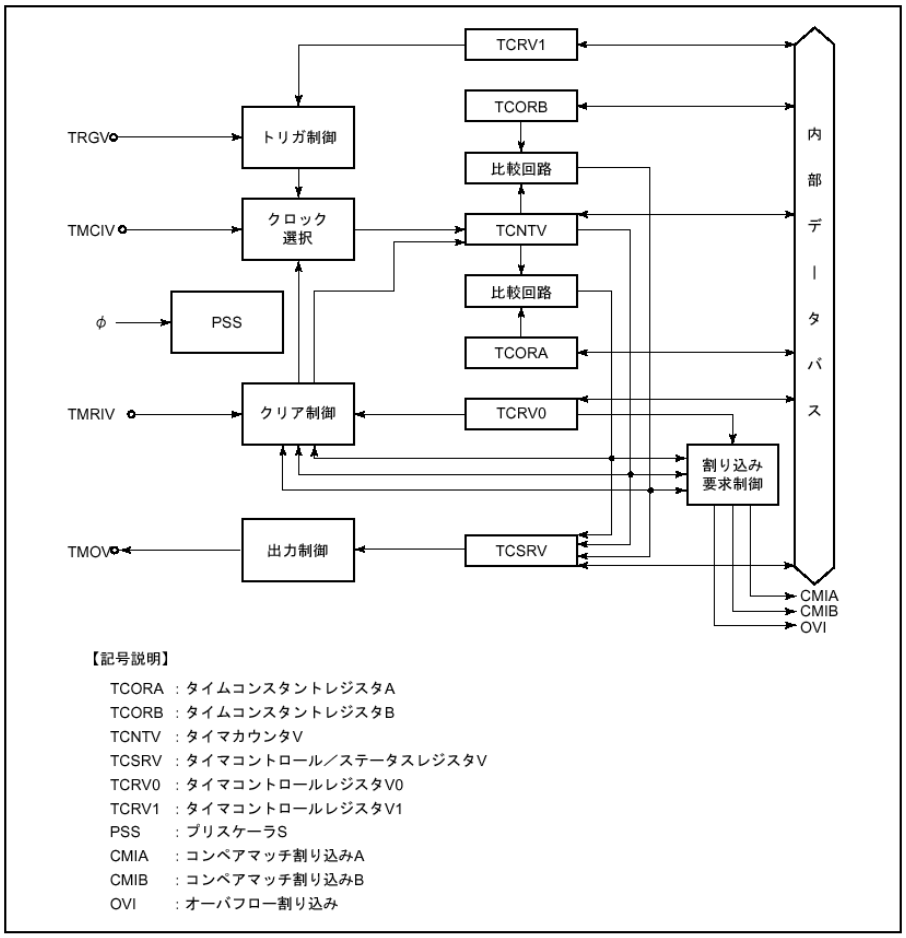


図 12.1　タイマ V のブロック図

■ タイマ V の設定用レジスタの説明

タイマ V の設定に関係するレジスタは次のとおりです。

タイマコントロールレジスタ V0(TCRV0) ： アドレス=0xFFA0 番地				
ビット	ビット名	初期値	R/W	説明
7	CMIEB	0	R/W	コンペアマッチインタラプトイネーブル B。 1 のとき TCSRV の CMFB による割込み要求がイネーブルになる。
6	CMIEA	0	R/W	コンペアマッチインタラプトイネーブル A。 1 のとき TCSRV の CMFA による割込み要求がイネーブルになる。
5	OVIE	0	R/W	タイマオーバフローインタラプトイネーブル。 1 のとき TCSRV の OVF による割込み要求がイネーブルになる。
4	CCLR1	0	R/W	カウンタクリア。TCNTV のクリア条件 00:クリアされない。 01:コンペアマッチ A でクリア。 10:コンペアマッチ B でクリア。 11:TMRIV 端子の立上がりエッジでクリア。
3	CCLR0	0	R/W	
2	CLK2	0	R/W	クロックセレクト。 TCRV1 の ICHS0 との組み合わせで，TCNTV に入力するクロックとカウント条件を選択する。下表を参照。
1	CLK1	0	R/W	
0	CLK0	0	R/W	

TCRV0			TCEV1	説明
CKS2	CKS1	CKS0	ICKS0	
0	0	0	−	クロック入力禁止。
0	0	1	0	内部クロックΦ/4 立ち下がりエッジでカウント。
0	0	1	1	内部クロックΦ/8 立ち下がりエッジでカウント。
0	1	0	0	内部クロックΦ/16 立ち下がりエッジでカウント。
0	1	0	1	内部クロックΦ/32 立ち下がりエッジでカウント。
0	1	1	0	内部クロックΦ/64 立ち下がりエッジでカウント。
0	1	1	1	内部クロックΦ/128 立ち下がりエッジでカウント。(＊)
1	0	0	−	クロック禁止。
1	0	1	−	外部クロックの立ち上がりエッジでカウント。
1	1	0	−	外部クロックの立ち下がりエッジでカウント。
1	1	1	−	外部クロックの立ち上がり/立ち下がり両エッジでカウント。

(＊)この組み合わせを次に作成するプログラムで設定する

タイマコントロールステータスレジスタ V（TCSRV） ： アドレス=0xFFA1 番地				
ビット	ビット名	初期値	R/W	説明
7	CMFB	0	R/W	コンペアマッチフラグ B。 ［セット条件］TCNTV の値と TCORB の値が一致したとき。 ［クリア条件］CMFB=1 で，CMFB をリードした後，CMFB に 0 をライト。
6	CMFA	0	R/W	コンペアマッチフラグ A。 ［セット条件］TCNTV の値と TCORA の値が一致したとき。 ［クリア条件］CMFA=1 で，CMFA をリードした後，CMFA に 0 をライト。
5	OVF	0	R/W	タイマオーバフローフラグ。 ［セット条件］TCNTV の値が FF から 00 にオーバーフローしたとき。 ［クリア条件］OVF=1 で，OVF をリードした後，OVF に 0 をライト。
4	－	1	－	リザーブビット。
3	OS3	0	R/W	アウトプットセレクト 3～2。TCORB と TCNTV のコンペアマッチによる TMOV 端子の出力方法。 00：変化しない。01：0 出力。10：1 出力。11：トグル出力。
2	OS2	0	R/W	
1	OS1	0	R/W	アウトプットセレクト 1～0。TCORA と TCNTV のコンペアマッチによる TMOV 端子の出力方法。 00：変化しない。01：0 出力。10：1 出力。11：トグル出力。
0	OS0	0	R/W	

タイマコントロールレジスタ V1（TCRV1） ： アドレス=0xFFA5 番地				
ビット	ビット名	初期値	R/W	説明
7	－		－	リザーブビット。
6	－		－	リザーブビット。
5	－		－	リザーブビット。
4	TVEG1	0	R/W	TRGV 入力エッジセレクト。 00：TRGV からのトリガ禁止。 01：立ち上がりエッジ。 10：立ち下がりエッジ。 11：立ち上がり/立ち下がり両エッジ。
3	TVEG0	0	R/W	
2	TRGE	0	R/W	TVEG1，TVEG0 で選択されたエッジの入力により，TCNTV カウントアップ開始。 0：TRGV 端子入力による TCNTV カウントアップの開始とコンペアマッチによる TCNTV クリア時の TCNTV カウントアップの停止を禁止。 1：TRGV 端子入力による TCNTV カウントアップの開始とコンペアマッチによる TCNTV クリア時の TCNTV カウントアップの停止を許可。
1	－	1	－	リザーブビット。
0	ICKS0	0	R/W	インターナルクロックセレクト。 TCRV0 の CKS2～0 との組み合わせで，TCNTV に入力するクロックとカウント条件を選択する。

■ タイマ V と割込みを使って LED を点滅させる

まずは基準となるタイミングをタイマ V で作ります。今回は CPU クロック（20MHz）の 1/128（=156.25KHz）で TCNTV を+1 します。そして，TCORA に 156 をセットし，一致したら割込みをかけると同時に TCNTV を 0 にクリアします。TCNTV は常に+1 されるので，割込みは約 1ms 毎（1KHz）にかかることになります。もちろん 1ms で LED を点滅させたのでは人間の目に感じられませんので，0．5 秒ぐらいで点滅するよう割込み処理の中で細工します。

割込みは外部割込みとほとんど同じ考え方です。異なるのは，CPU が生成するタイマ V の割り込み要求に対するベクタアドレスが 002Ch であることです。そのベクタアドレスにセットされているデータ（割込み処理のスタートアドレス）をプログラムカウンタにセットします。

表 3.1　例外処理要因とベクタアドレス

発生元	例外処理要因	ベクタ番号	ベクタアドレス	優先度
RES 端子 ウォッチドッグタイマ	リセット	0	H'0000～H'0001	高
—	システム予約	1～6	H'0002～H'000D	
外部割り込み端子	NMI	7	H'000E～H'000F	
CPU	トラップ命令 #0	8	H'0010～H'0011	
	トラップ命令 #1	9	H'0012～H'0013	
	トラップ命令 #2	10	H'0014～H'0015	
	トラップ命令 #3	11	H'0016～H'0017	
アドレスブレーク	ブレーク条件成立	12	H'0018～H'0019	
CPU	スリープ命令の実行による直接遷移	13	H'001A～H'001B	
外部割り込み端子	IRQ0 低電圧検出割り込み*	14	H'001C～H'001D	
	IRQ1	15	H'001E～H'001F	
	IRQ2	16	H'0020～H'0021	
	IRQ3	17	H'0022～H'0023	
	WKP	18	H'0024～H'0025	
RTC	オーバフロー	19	H'0026～H'0027	
—	システム予約	20	H'0028～H'0029	
タイマ V	コンペアマッチ A コンペアマッチ B オーバフロー	22	H'002C～H'002D	
SCI3	受信データフル 送信データエンプティ 送信終了 受信エラー	23	H'002E～H'002F	
IIC2	送信データエンプティ、送信終了、受信データフル、 アービトレーションロスト／オーバランエラー NACK 検出、停止条件検出	24	H'0030～H'0031	
A/D 変換器	A/D 変換終了	25	H'0032～H'0033	
タイマ Z	コンペアマッチ／インプットキャプチャ A0～D0 オーバフロー	26	H'0034～H'0035	
	コンペアマッチ／インプットキャプチャ A1～D1 オーバフロー、アンダフロー	27	H'0036～H'0037	
タイマ B1	オーバフロー	29	H'003A～H'003B	
SCI3_2	受信データフル 送信データエンプティ 送信終了 受信エラー	32	H'0040～H'0041	低

【注】　*　低電圧検出割り込みは、パワーオンリセット＆低電圧検出回路内蔵版のみ有効です。

では，ソースリストを書いてみましょう。

```
/************************************************************************/
/*                                                                      */
/*  FILE        :tmv_led_c.c                                            */
/*  DATE        :Thu, Jan 10, 2008                                      */
/*  DESCRIPTION :Main Program                                           */
/*  CPU TYPE    :H8/3687                                                */
/*                                                                      */
/*  This file is programed by TOYO-LINX Co.,Ltd. / yKikuchi             */
/*                                                                      */
/************************************************************************/

/************************************************************************
    インクルードファイル
************************************************************************/
#include <machine.h>    // H8特有の命令を使う
#include "iodefine.h"   // 内蔵I/Oのラベル定義

/************************************************************************
    グローバル変数の定義とイニシャライズ(RAM)
************************************************************************/
unsigned int   LedCnt    =    0;   //点滅タイミングカウンタ

/************************************************************************
    関数の定義
************************************************************************/
void      intprog_tmv(void);
void      main(void);

/************************************************************************
    メインプログラム
************************************************************************/
void main(void)
{
    IO.PCR6 = 0x01;             // ポート6のbit0(P60)を出力に設定

    TV.TCSRV.BYTE  =    0x00;     //TOMV端子は使わない
    TV.TCORA       =    156;      //周期=1ms(1kHz)
    TV.TCRV1.BYTE  =    0x01;     //TRGVトリガ入力禁止,
    TV.TCRV0.BYTE  =    0x4b;     //コンペアマッチA 割込みイネーブル
                                  //コンペアマッチA でTCNTVクリア
                                  //内部クロックφ/128(=156.25kHz)

    while(1){}
}

/************************************************************************
    タイマV  割込み
************************************************************************/
#pragma regsave (intprog_tmv)
void intprog_tmv(void)
{
    //コンペアマッチフラグA クリア
    TV.TCSRV.BIT.CMFA = 0;
```

```
    //点滅
    LedCnt = LedCnt + 1;
    if ((LedCnt&0x0200)==0x0200){
        IO.PDR6.BIT.B0 = 0;
    }
    else{
        IO.PDR6.BIT.B0 = 1;
    }
}
```

割込みを使うためにはソースファイルだけではなく，HEW が自動生成する‘intprg. c’を修正する必要があります。下記のリストをご覧ください。

```
/************************************************************************/
/*                                                                      */
/*  FILE        :intprg.c                                               */
/*  DATE        :Thu, Jan 10, 2008                                      */
/*  DESCRIPTION :Interrupt Program                                      */
/*  CPU TYPE    :H8/3687                                                */
/*                                                                      */
/*  This file is generated by Renesas Project Generator (Ver.4.9).     */
/*                                                                      */
/************************************************************************/



#include <machine.h>

extern void intprog_tmv(void);

#pragma section IntPRG
//  vector 1 Reserved

//  vector 2 Reserved

//  vector 3 Reserved

//  vector 4 Reserved

//  vector 5 Reserved

//  vector 6 Reserved

//  vector 7 NMI
__interrupt(vect=7) void INT_NMI(void) {/* sleep(); */}
//  vector 8 TRAP #0
__interrupt(vect=8) void INT_TRAP0(void) {/* sleep(); */}
//  vector 9 TRAP #1
__interrupt(vect=9) void INT_TRAP1(void) {/* sleep(); */}
//  vector 10 TRAP #2
__interrupt(vect=10) void INT_TRAP2(void) {/* sleep(); */}
//  vector 11 TRAP #3
__interrupt(vect=11) void INT_TRAP3(void) {/* sleep(); */}
```

```
//  vector 12 Address break
__interrupt(vect=12) void INT_ABRK(void) {/* sleep(); */}
//  vector 13 SLEEP
__interrupt(vect=13) void INT_SLEEP(void) {/* sleep(); */}
//  vector 14 IRQ0
__interrupt(vect=14) void INT_IRQ0(void) {/* sleep(); */}
//  vector 15 IRQ1
__interrupt(vect=15) void INT_IRQ1(void) {/* sleep(); */}
//  vector 16 IRQ2
__interrupt(vect=16) void INT_IRQ2(void) {/* sleep(); */}
//  vector 17 IRQ3
__interrupt(vect=17) void INT_IRQ3(void) {/* sleep(); */}
//  vector 18 WKP
__interrupt(vect=18) void INT_WKP(void) {/* sleep(); */}
//  vector 19 RTC
__interrupt(vect=19) void INT_RTC(void) {/* sleep(); */}
//  vector 20 Reserved

//  vector 21 Reserved

//  vector 22 Timer V
__interrupt(vect=22) void INT_TimerV(void) {intprog_tmv();}
//  vector 23 SCI3
__interrupt(vect=23) void INT_SCI3(void) {/* sleep(); */}
//  vector 24 IIC2
__interrupt(vect=24) void INT_IIC2(void) {/* sleep(); */}
//  vector 25 ADI
__interrupt(vect=25) void INT_ADI(void) {/* sleep(); */}
//  vector 26 Timer Z0
__interrupt(vect=26) void INT_TimerZ0(void) {/* sleep(); */}
//  vector 27 Timer Z1
__interrupt(vect=27) void INT_TimerZ1(void) {/* sleep(); */}
//  vector 28 Reserved

//  vector 29 Timer B1
__interrupt(vect=29) void INT_TimerB1(void) {/* sleep(); */}
//  vector 30 Reserved

//  vector 31 Reserved

//  vector 32 SCI3_2
__interrupt(vect=32) void INT_SCI3_2(void) {/* sleep(); */}
```

では，ビルドして実行してみましょう。期待通り動作するでしょうか。なお，付属の CD-ROM にはあらかじめダウンロードするファイルがおさめられています。

‘tmv_led_c. mot’

をダウンロードして実行してください。

■ 練習問題

今度はポート 6 にインクリメントデータを表示してください。(解答例は「tmv_led_inc_c」)

4. タイマ Z

H8/3687 に内蔵されている別のタイマ，タイマ Z について調べてみましょう。前節のタイマ V と比べ，さらに多機能なタイマになっています。

■ タイマ Z の概要

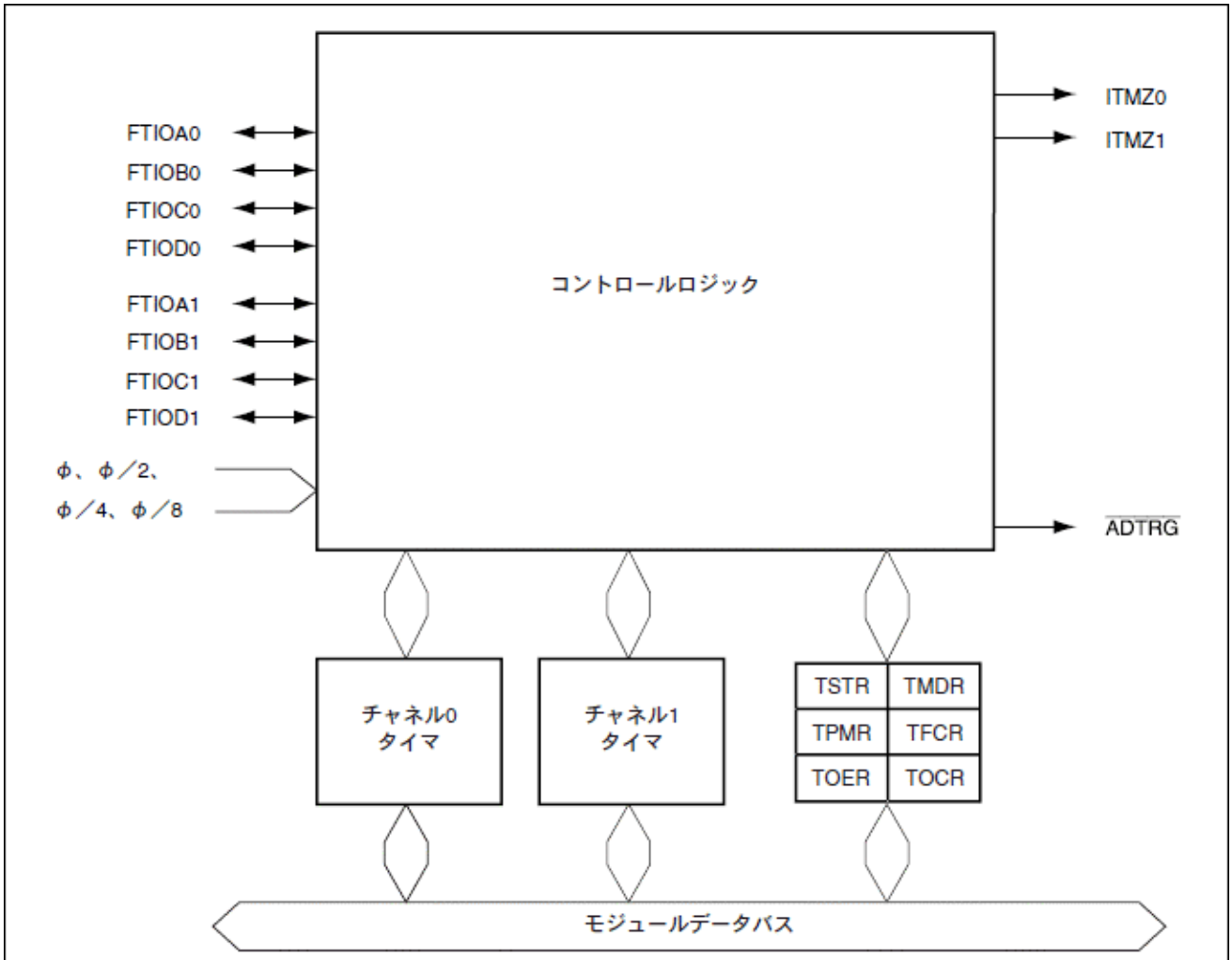
タイマ Z は 16 ビットカウンタをベースにした 16 ビットタイマで，ほとんど同じ機能の 16 ビットタイマを 2 チャンネル内蔵しています。

基本的な使い方の場合（コンペアマッチによる出力），タイマ Z に内蔵されているタイマカウンタ（TCNT_0 と TCNT_1）は CPU クロック（TK-3687mini の場合は 20MHz）を分周したクロック（1/8，1/4，1/2，1/1 のいずれか選択可能）によって常に+1 されます。TCNT_0 と TCNT_1 は各チャンネルに 4 本，合計 8 本あるジェネラルレジスタ（GR）と比較されており，一致すると CPU に割り込みをかけたり，ポートの出力を変化させることができます。その際，TCNT_0 と TCNT_1 を 0 にクリアするよう設定することができます。

タイマ Z はその他にも多くの機能を内蔵しています。タイマ Z の詳細については「H8/3687 シリーズ ハードウェアマニュアル」の「13. タイマ Z」をご覧下さい。

表 13.1 タイマ Z の機能一覧

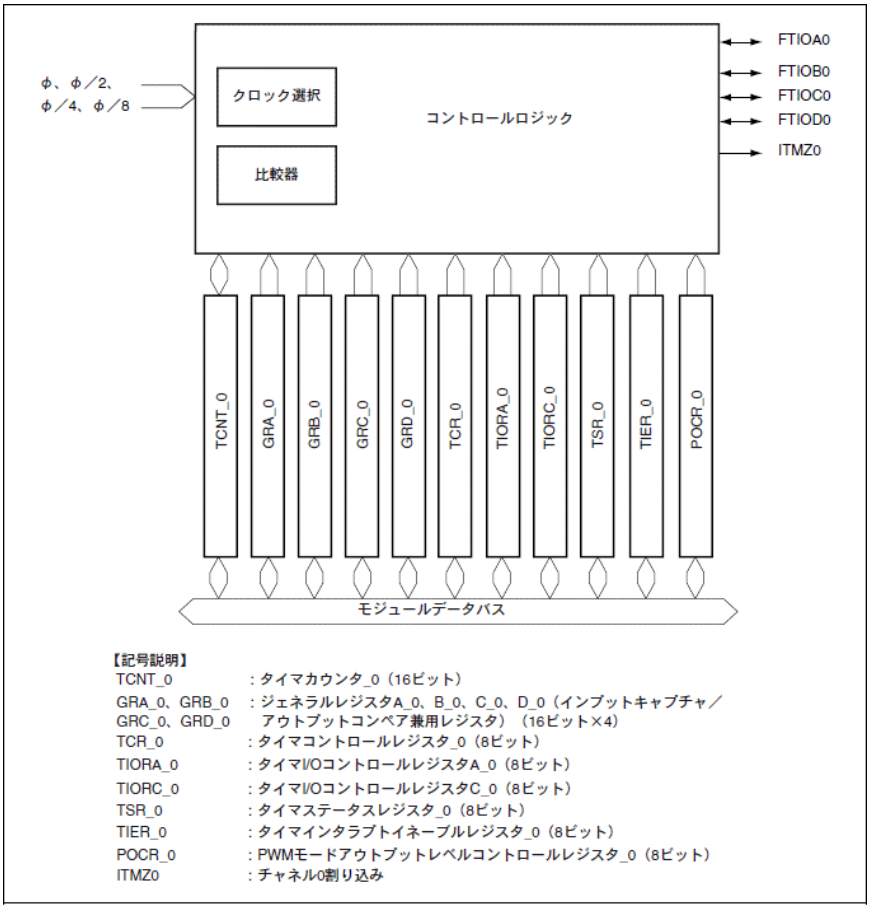
項目		チャネル 0	チャネル 1
カウントクロック		内部クロック：φ、φ／2、φ／4、φ／8 外部クロック：FTIOA0（TCLK）	
ジェネラルレジスタ（アウトプットコンペア／インプットキャプチャ兼用レジスタ）		GRA_0、GRB_0、GRC_0、GRD_0	GRA_1、GRB_1、GRC_1、GRD_1
バッファレジスタ		GRC_0、GRD_0	GRC_1、GRD_1
入出力端子		FTIOA0、FTIOB0、FTIOC0、FTIOD0	FTIOA1、FTIOB1、FTIOC1、FTIOD1
カウンタクリア機能		GRA_0／GRB_0／GRC_0／GRD_0 のコンペアマッチまたはインプットキャプチャ	GRA_1／GRB_1／GRC_1／GRD_1 のコンペアマッチまたはインプットキャプチャ
コンペアマッチ出力	0 出力	○	○
	1 出力	○	○
	トグル出力	○	○
インプットキャプチャ機能		○	○
同期動作		○	○
PWM モード		○	○
リセット同期 PWM モード		○	○
相補 PWM モード		○	○
バッファ動作		○	○
割り込み要因		コンペアマッチ／インプットキャプチャ A0～D0 オーバフロー	コンペアマッチ／インプットキャプチャ A1～D1 オーバフロー アンダフロー



【記号説明】

TSTR	：タイマスタートレジスタ（8ビット）
TMDR	：タイマモードレジスタ（8ビット）
TPMR	：タイマPWMモードレジスタ（8ビット）
TFCR	：タイマファンクションコントロールレジスタ（8ビット）
TOER	：タイマアウトプットマスタイネーブルレジスタ（8ビット）
TOCR	：タイマアウトプットコントロールレジスタ（8ビット）
$\overline{\text{ADTRG}}$	：A／D変換開始トリガ出力信号
ITMZ0	：チャネル0割り込み
ITMZ1	：チャネル1割り込み

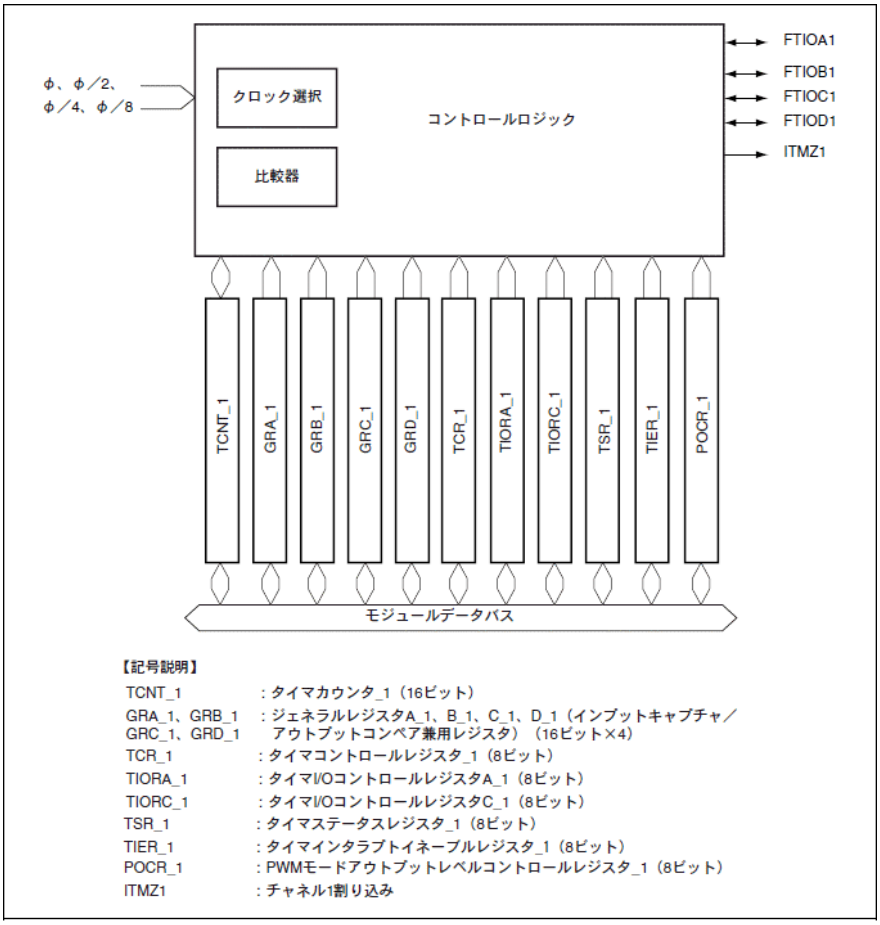
図 13.1　タイマ Z のブロック図



【記号説明】

TCNT_0	：タイマカウンタ_0（16ビット）
GRA_0、GRB_0 GRC_0、GRD_0	：ジェネラルレジスタA_0、B_0、C_0、D_0（インプットキャプチャ／アウトプットコンペア兼用レジスタ）（16ビット×4）
TCR_0	：タイマコントロールレジスタ_0（8ビット）
TIORA_0	：タイマI/Oコントロールレジスタ A_0（8ビット）
TIORC_0	：タイマI/Oコントロールレジスタ C_0（8ビット）
TSR_0	：タイマステータスレジスタ_0（8ビット）
TIER_0	：タイマインタラプトイネーブルレジスタ_0（8ビット）
POCR_0	：PWMモードアウトプットレベルコントロールレジスタ_0（8ビット）
ITMZ0	：チャネル0割り込み

図 13.2　タイマ Z（チャネル 0）のブロック図



【記号説明】

TCNT_1	：タイマカウンタ_1（16ビット）
GRA_1、GRB_1 GRC_1、GRD_1	：ジェネラルレジスタA_1、B_1、C_1、D_1（インプットキャプチャ／アウトプットコンペア兼用レジスタ）（16ビット×4）
TCR_1	：タイマコントロールレジスタ_1（8ビット）
TIORA_1	：タイマI/Oコントロールレジスタ A_1（8ビット）
TIORC_1	：タイマI/Oコントロールレジスタ C_1（8ビット）
TSR_1	：タイマステータスレジスタ_1（8ビット）
TIER_1	：タイマインタラプトイネーブルレジスタ_1（8ビット）
POCR_1	：PWMモードアウトプットレベルコントロールレジスタ_1（8ビット）
ITMZ1	：チャネル1割り込み

図 13.3　タイマ Z（チャネル 1）のブロック図

■ タイマ Z を使ってサウンダを鳴らしてみよう

タイマ Z を使ってサウンダを鳴らしてみましょう。今回は，タイマ Z チャネル 0 のコンペアマッチ A で割り込みます。割込みがかかるたびにポート 5 の P52 をトグル出力すれば，与えられる周波数の音が出ます。そして，割込みがかかるたびに GRA の値を+1 していけば，周波数が徐々に変化していきます。どんな音になるでしょうか。なお，GRA の上位 8 ビットをポート 6 に出力して LED に表示します。

ソースリストは次のとおりです。

```
/************************************************************************/
/*                                                                      */
/*  FILE        :sounder_c.c                                            */
/*  DATE        :Fri, Jan 11, 2008                                      */
/*  DESCRIPTION :Main Program                                           */
/*  CPU TYPE    :H8/3687                                                */
/*                                                                      */
/*  This file is programed by TOYO-LINX Co.,Ltd. / yKikuchi             */
/*                                                                      */
/************************************************************************/

/************************************************************************
    インクルードファイル
************************************************************************/
#include <machine.h>    // H8特有の命令を使う
#include "iodefine.h"   // 内蔵I/Oのラベル定義

/************************************************************************
    関数の定義
************************************************************************/
void      intprog_tmz0(void);
void      main(void);

/************************************************************************
    メインプログラム
************************************************************************/
void main(void)
{
    IO.PMR5.BYTE  =   0x00;     //ポート5,汎用入出力ポート
    IO.PUCR5.BYTE =   0x38;     //ポート5,P53-55内蔵プルアップオン
    IO.PCR5       =   0x04;     //ポート5のbit2(P52)を出力に設定
    IO.PDR5.BYTE  =   0x04;     //ポート5,初期出力

    IO.PCR6       =   0xff;     //ポート6を出力に設定
    IO.PDR6.BYTE  =   0xff;     //ポート6,初期出力

    TZ.TSTR.BYTE  =   0x00;     //TCNT0,1 停止
    TZ0.TCR.BYTE  =   0x20;     //GRAのコンペアマッチでTCNT=0,φ/1
    TZ.TOER.BYTE  =   0xff;     //FTIO端子ディセーブル
    TZ0.TIORA.BYTE=   0x88;     //GRAはアウトプットコンペアレジスタ
                                //コンペアマッチによる出力禁止
    TZ0.TSR.BYTE  =   0x00;     //割込みフラグクリア
    TZ0.TIER.BYTE =   0x01;     //コンペアマッチインターラプトイネーブルA
    TZ0.GRA       =   0x0001;   //カウント初期値
    TZ0.TCNT      =   0x0000;   //TCNT0=0
    TZ.TSTR.BYTE  =   0x01;     //TCNT0 カウントスタート
```

```
    while(1){}
}

/************************************************************************
    タイマZ0　割込み
************************************************************************/
#pragma regsave (intprog_tmz0)
void intprog_tmz0(void)
{
    unsigned int d;

    //タイマZ0 コンペアマッチインタラプトフラグ クリア
    TZ0.TSR.BIT.IMFA =0;

    //サウンダ出力反転
    IO.PDR5.BIT.B2 = ~IO.PDR5.BIT.B2;

    //次のGRAを計算してセット
    d = TZ0.GRA;
    d = d + 1;
    TZ0.GRA = d;
    IO.PDR6.BYTE = ~(d / 0x0100);
}
```

割込みを使うためにはソースファイルだけではなく，HEW が自動生成する‘intprg．c’を修正する必要があります。下記のリストをご覧ください。

```
/************************************************************************/
/*                                                                      */
/*  FILE        :intprg.c                                               */
/*  DATE        :Fri, Jan 11, 2008                                      */
/*  DESCRIPTION :Interrupt Program                                      */
/*  CPU TYPE    :H8/3687                                                */
/*                                                                      */
/*  This file is generated by Renesas Project Generator (Ver.4.9).      */
/*                                                                      */
/************************************************************************/



#include <machine.h>

extern void intprog_tmz0(void);

#pragma section IntPRG
//  vector 1 Reserved

//  vector 2 Reserved

//  vector 3 Reserved

//  vector 4 Reserved
```

```
//  vector 5 Reserved

//  vector 6 Reserved

//  vector 7 NMI
__interrupt(vect=7) void INT_NMI(void) {/* sleep(); */}
//  vector 8 TRAP #0
__interrupt(vect=8) void INT_TRAP0(void) {/* sleep(); */}
//  vector 9 TRAP #1
__interrupt(vect=9) void INT_TRAP1(void) {/* sleep(); */}
//  vector 10 TRAP #2
__interrupt(vect=10) void INT_TRAP2(void) {/* sleep(); */}
//  vector 11 TRAP #3
__interrupt(vect=11) void INT_TRAP3(void) {/* sleep(); */}
//  vector 12 Address break
__interrupt(vect=12) void INT_ABRK(void) {/* sleep(); */}
//  vector 13 SLEEP
__interrupt(vect=13) void INT_SLEEP(void) {/* sleep(); */}
//  vector 14 IRQ0
__interrupt(vect=14) void INT_IRQ0(void) {/* sleep(); */}
//  vector 15 IRQ1
__interrupt(vect=15) void INT_IRQ1(void) {/* sleep(); */}
//  vector 16 IRQ2
__interrupt(vect=16) void INT_IRQ2(void) {/* sleep(); */}
//  vector 17 IRQ3
__interrupt(vect=17) void INT_IRQ3(void) {/* sleep(); */}
//  vector 18 WKP
__interrupt(vect=18) void INT_WKP(void) {/* sleep(); */}
//  vector 19 RTC
__interrupt(vect=19) void INT_RTC(void) {/* sleep(); */}
//  vector 20 Reserved

//  vector 21 Reserved

//  vector 22 Timer V
__interrupt(vect=22) void INT_TimerV(void) {/* sleep(); */}
//  vector 23 SCI3
__interrupt(vect=23) void INT_SCI3(void) {/* sleep(); */}
//  vector 24 IIC2
__interrupt(vect=24) void INT_IIC2(void) {/* sleep(); */}
//  vector 25 ADI
__interrupt(vect=25) void INT_ADI(void) {/* sleep(); */}
//  vector 26 Timer Z0
__interrupt(vect=26) void INT_TimerZ0(void) {intprog_tmz0();}
//  vector 27 Timer Z1
__interrupt(vect=27) void INT_TimerZ1(void) {/* sleep(); */}
//  vector 28 Reserved

//  vector 29 Timer B1
__interrupt(vect=29) void INT_TimerB1(void) {/* sleep(); */}
//  vector 30 Reserved

//  vector 31 Reserved

//  vector 32 SCI3_2
```

```
__interrupt(vect=32) void INT_SCI3_2(void) {/* sleep(); */}
```

では，ビルドして実行してみましょう。期待通り動作するでしょうか。なお，付属の CD-ROM にはあらかじめダウンロードするファイルがおさめられています。

‘sounder_c. mot’

をダウンロードして実行してください。

■ 練習問題

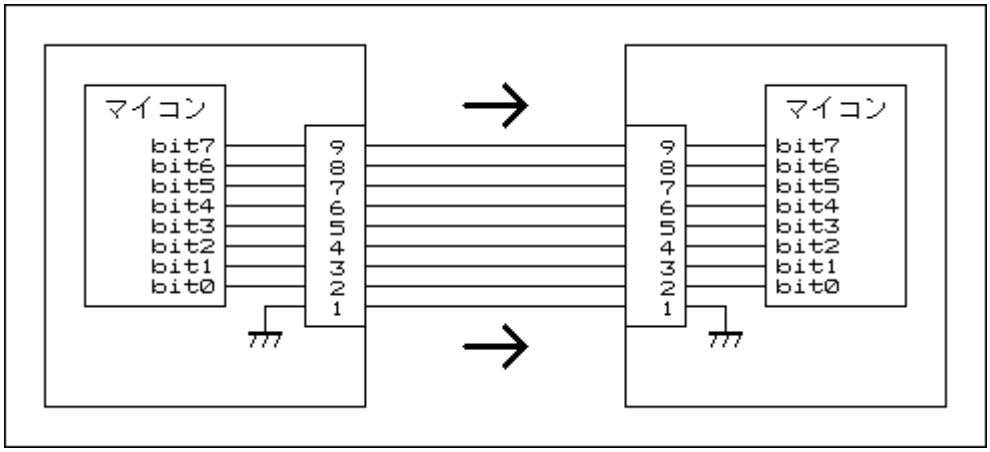
この考え方でいろいろな音が作れると思います。いろいろ考えてみてください。（解答例「sounder_c_v2」）

5. シリアルコミュニケーションインターフェース

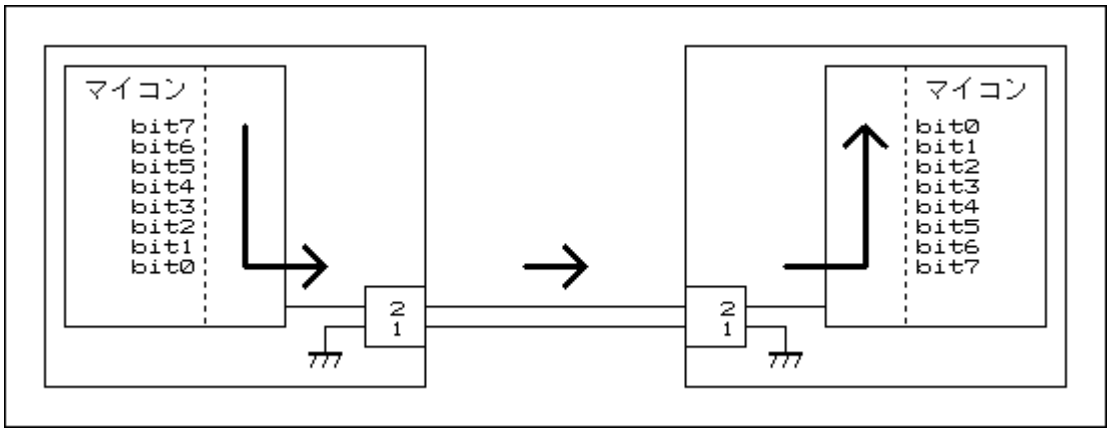
■ シリアル通信の基本的な考え方

1 バイト(=8 ビット)のデータを伝えることを考えてみましょう。線は何本必要でしょうか。

パラレルポートの考え方ですと, 1 ビットにつき 1 本なので, 8 本必要になりますね。もちろん, これだけでは当然駄目で, 信号の基準になる GND 用に 1 本は必要なので 9 本以上になります。



さて, これをなんとか, 信号 1 本と GND1 本, 合計 2 本の線だけでデータを伝えることはできないでしょうか。信号が1本しかないのですから1ビットずつ順番に送るしか方法がありません。この発想から生まれた方法がシリアル通信です。

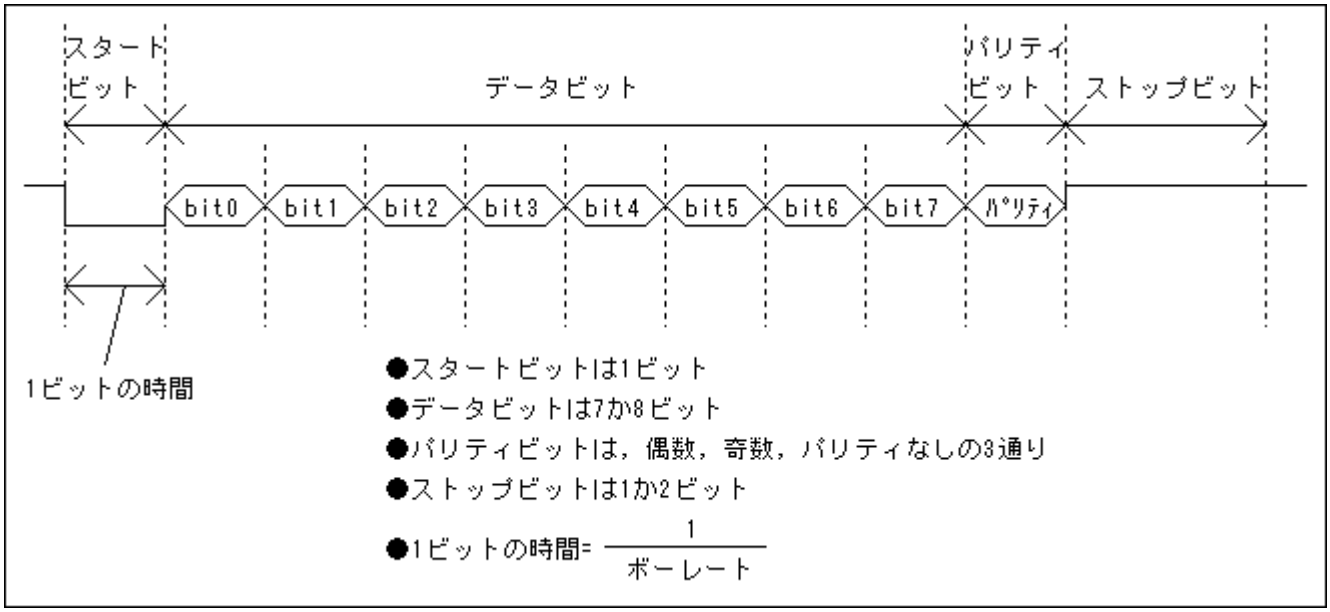


上の図では bit0 から順番に送り出します。受ける方は bit0 から受けていきます。8 ビット受け取ったら, 1 バイトデータとして使います。

■ 調歩同期式シリアル通信

1 ビットずつ送受信するわけですが，問題になるのは今受信しているデータが何ビット目なんだろう，ということです。これが伝わらないとまったくちがうデータになってしまいます。いろいろな方法が考えられているのですが，その中でもっとも基本的な調歩同期式という方法を調べてみましょう。

次の図をご覧下さい。これが調歩同期式シリアル通信のフォーマットになります。



かぎとなるのは，1 ビットの時間が決まっていることと，信号線は通常は High（5V）で，スタートビットで必ず Low（0V）になるということです。ハイパーH8 の設定をしたとき COM1 のプロパティを設定しました。ちょっと思い出してみましょう。（右図参照）

‘ビット/秒’というのがありますが，これは別の言葉でボーレート（単位：bps，bit/s，またはボー）といいます。上の式に当てはめると，1 ビットの時間は約 26μ 秒となります。シリアルポートをずっと見ていて，High から Low になったらスタートビットが始まったと判断します。そこから 26μ 秒たったら bit0 が始まります。あとはその繰り返しですべてのビットを受け取ることができます。ストップビットは必ず High なので，次のデータのスタートビットを見つける準備ができています。うまくできている思いませんか。

■ シリアルコミュニケーションインターフェース 3

調歩同期式シリアル通信の考え方はわかったと思いますが，これを I/O ポートとプログラムだけで作るのは結構たいへんです。ちょっとでもタイミングがずれると，ちゃんとデータを受け取ることができません。

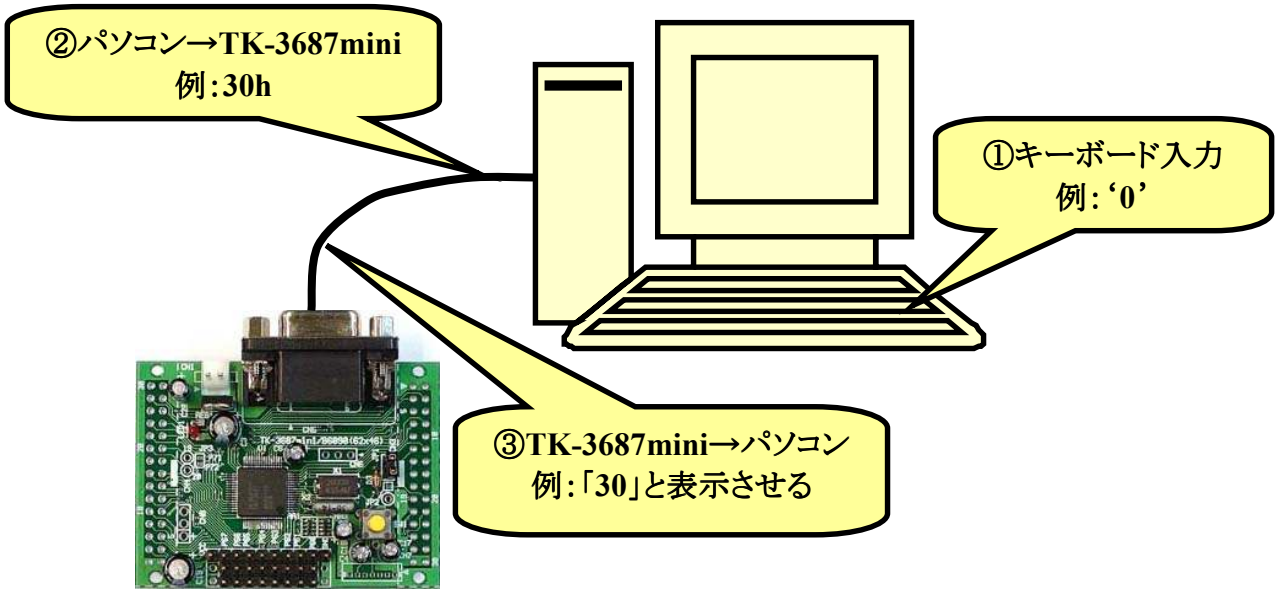
大変なことは専用のパーツにおまかせしましょう，というのがスマートな方法です。H8/3687 にはシリアル通信用の I/O が内蔵されています。「シリアルコミュニケーションインタフェース 3（SCI3）」と呼ばれています。SCI3 は調歩同期式シリアル通信以外にも対応できるように作られています。詳しくは，「H8/3687 グループ ハードウェアマニュアル」（以降ハードウェアマニュアル）の 16-1 ページから説明されていますので，ぜひお読みください。I/O ポートにくらべると SCI3 の使い方は最初は難しく感じるのですが，わかってしまうとそれほどでもありません。しかも I/O の使い方の基本が含まれているので，SCI3 の使い方がわかると他の I/O の使い方，例えば I^2C バスインタフェース 2（IIC2）を理解するのもそれほどたいへんではなくなります。ここは一つがんばってみてください。

■ ハイパーターミナルから送られてくるデータを見てみよう

SCI3 を使ったプログラム例を考えてみましょう。

ハイパーH8 はハイパーターミナルを使っていますね。パソコンのキーボードからキーを入力すると，いろいろと表示されます。よく考えると不思議ですよね。

パソコンのキーボードで入力すると，TK-3687mini にどんなデータが送られているのでしょうか。答えを言うようですが，それぞれのキーに割付けられた数字が送られてきます。というわけで，どんな数字が送られてきたか，それを返信してハイパーターミナルに表示するプログラムを作ってみましょう。



とりあえず動かしてみたい方は，CD-ROM から

‘sci3_key_code_c. mot’

をダウンロードして実行して下さい。パソコンのキーを押すとハイパーターミナルに数字が表示されるはずです。

シリアルポートのプログラムで最初に考えるのは通信条件です。今回はハイパーH8 を動かしていたハイパーターミナルに表示するので，ハイパーターミナルと同じ条件になります。

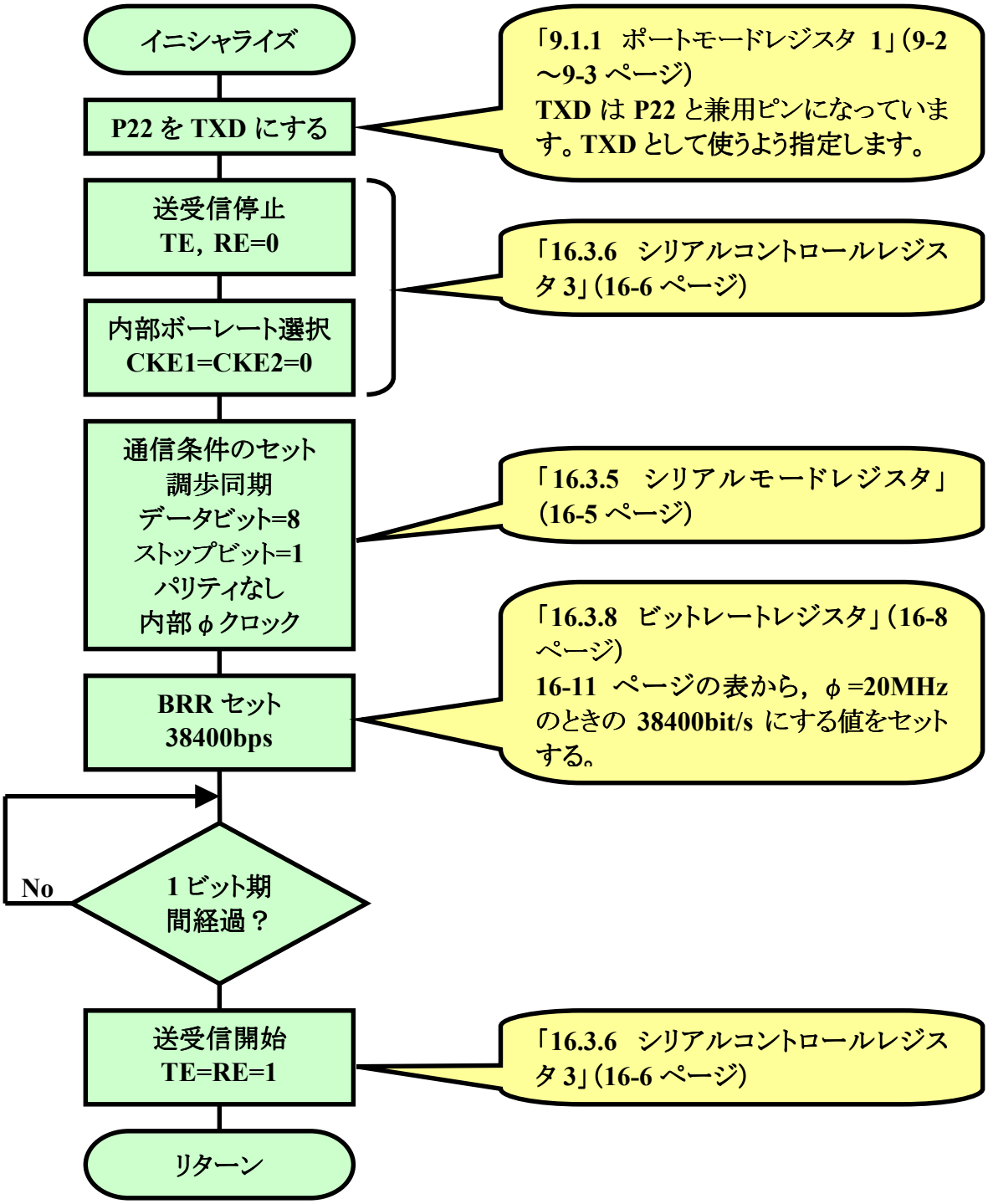
ビット/秒（ボーレート）・・・・・・・・・・・38400bps（38400bit/s，38400 ボー）

データビット・・・・・・・・・・・・・・・・・・・8 ビット

パリティ・・・・・・・・・・・・・・・・・・・・・・なし

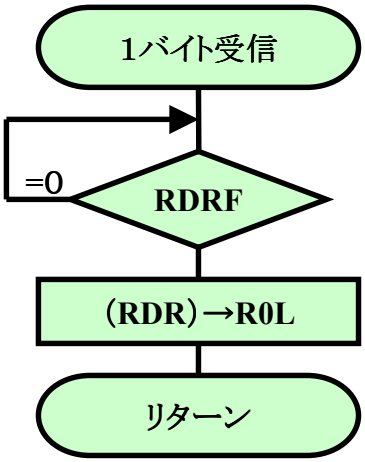
ストップビット・・・・・・・・・・・・・・・・・・1 ビット

まずは SCI3 にこの条件をセットします。ハードウェアマニュアルの 16-4 ページ，「16.4.2 SCI3 の初期化」を参考にイニシャライズのフローチャートを作ってみました。



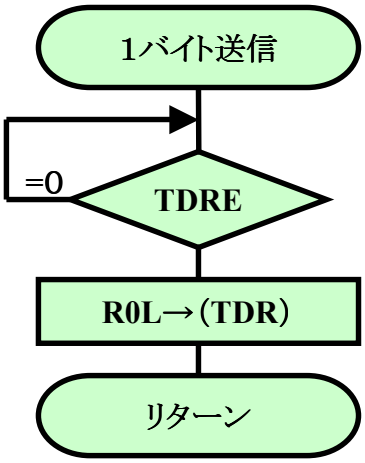
次に受信動作について考えてみましょう。当然ながら，TK-3687mini はパソコンのキーがいつ押されるかわかりません。もっとも，受信動作そのものは SCI3 が自動的に行なってくれます。そして，データを受信したかどうかしらせるステータスが用意されています。というわけで，マイコンはそのステータスを見て，受信していたらデータを読み込みます。

ハードウェアマニュアルの 16-7 ページ，「16.3.7　シリアルステータスレジスタ」をご覧下さい。ビット 6，‘RDRF’が 1 になったらデータを受信しています。受信していたらレシーブデータレジスタ‘RDR’からデータを読み込みます。フローチャートにしてみました。



あとは送信動作です。38400bps で 1 データ送信するのにどれくらい時間がかかるでしょうか。今作っているプログラムの条件だと約 260μs(μs:マイクロ秒は 1 秒の百万分の一)かかります。速いように思うかもしれませんが，マイコン(H8/3687)は 1 つの命令を 0．1μs～1．2μs で実行できることを考えると，ものすご～く遅いということがわかります。もし，何も考えずに SCI3 に送信データをどんどん書き込むと，まだ送信が終わっていないのに書き込むことになるかもしれません。それで，送信データを書き込んでよいか判断するステータスが用意されています。マイコンはそのステータスを見て，大丈夫ならデータを書き込みます。

ハードウェアマニュアルの 16-7 ページ，「16.3.7　シリアルステータスレジスタ」をご覧下さい。ビット 7，‘TDRE’が 1 になったら送信データを書き込んでも大丈夫です。トランスミットデータレジスタ‘TDR’に送信データを書き込みます。フローチャートにしてみました。



ここで出てきた，ステータスを見ながらデータを読み込んだり書き込んだりする考え方は，I/O を使うときの基本的な考え方です。ぜひおぼえておいてください。

さて，プログラムリストは次のようになりました。

```
/************************************************************************/
/*                                                                      */
/*  FILE        :sci3_key_code_c.c                                      */
/*  DATE        :Fri, Jan 11, 2008                                      */
/*  DESCRIPTION :Main Program                                           */
/*  CPU TYPE    :H8/3687                                                */
/*                                                                      */
/*  This file is programed by TOYO-LINX Co.,Ltd. / yKikuchi             */
/*                                                                      */
/************************************************************************/

/************************************************************************
    インクルードファイル
************************************************************************/
#include <machine.h>   // H8特有の命令を使う
#include "iodefine.h" // 内蔵I/Oのラベル定義


/************************************************************************
    関数の定義
************************************************************************/
unsigned int  hex2asc(unsigned char);
void          init_sci3(void);
void          main(void);
unsigned char rxone(void);
void          txone(unsigned char);

/************************************************************************
    メインプログラム
************************************************************************/
void main(void)
{
    unsigned char rd;
    unsigned int hd;

    init_sci3();  //SCI3イニシャライズ
    txone(0x0d);  //改行

    while(1){
        rd = rxone();           //1バイト受信
        hd = hex2asc(rd);       //アスキーコード変換
        txone(hd / 0x0100);     //上位バイト送信
        txone(hd & 0x00ff);     //下位バイト送信
        txone(0x0d);            //改行
    }
}

/************************************************************************
    SCI3 イニシャライズ
************************************************************************/
```

```
void init_sci3(void)
{
    #define       CPU_CLK  20000000      // Clock=20MHz=20000000Hz
    #define       BAUD     38400         // baudrate
    #define       BITR     (CPU_CLK) / (BAUD*32) - 1
    #define       WAIT_1B  (CPU_CLK) / 6 / BAUD

    char i;

    IO.PMR1.BYTE = IO.PMR1.BYTE | 0x02; //P22はTXDとして使う

    SCI3.SCR3.BYTE    =    0x00;
    SCI3.SMR.BYTE     =    0x00;     //調歩同期,8bit,NonParity,StopBit=1
    SCI3.BRR          =    BITR;     //38400Baud
    for  (i=0;i<WAIT_1B;i++){};      //1bit期間 wait
    SCI3.SCR3.BYTE    =    0x30;     //送信イネーブル,受信イネーブル,割込みディセーブル
}

/*****************************************************************************
    1文字送信（ポーリング）
------------------------------------------------------------------------------
    引数 txdata        送信データ
*****************************************************************************/
void txone(unsigned char txdata)
{
    while(SCI3.SSR.BIT.TDRE==0){}        //送信可能まで待つ
    SCI3.TDR = txdata;
}

/*****************************************************************************
    1文字受信（ポーリング）
------------------------------------------------------------------------------
    戻り値    受信データ
*****************************************************************************/
unsigned char rxone(void)
{
    while(SCI3.SSR.BIT.RDRF==0){}        //受信するまで待つ
    return SCI3.RDR;
}

/*****************************************************************************
    アスキーコード変換
------------------------------------------------------------------------------
    引数      16進データ
    戻り値    アスキーコード
*****************************************************************************/
unsigned int hex2asc(unsigned char hex_dt)
{
    unsigned int asc_dt;

    asc_dt = hex_dt & 0x0f;
    if (asc_dt>0x09)   {asc_dt = asc_dt + 0x37;}
    else               {asc_dt = asc_dt + 0x30;}
```

```
    hex_dt = hex_dt / 0x10;
    if (hex_dt>0x09)   {asc_1dt = (hex_dt + 0x37) * 0x0100 + asc_dt;}
    else               {asc_dt = (hex_dt + 0x30) * 0x0100 + asc_dt;}

    return asc_dt;
}
```

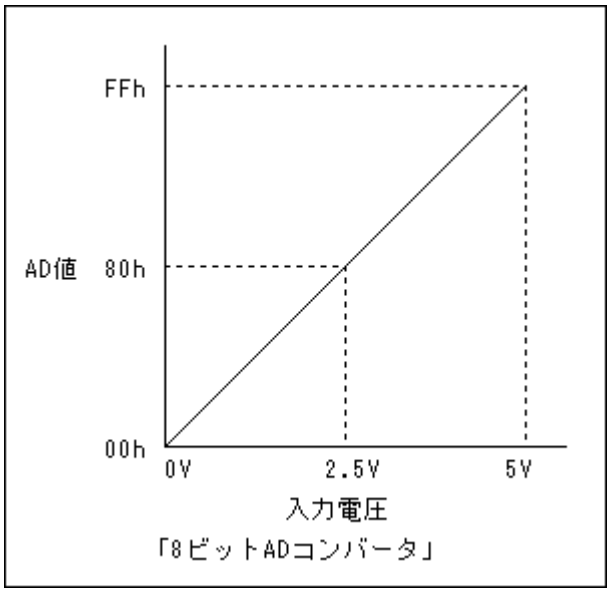
ビルドしてプログラムを実行してみましょう。いろいろキーボードから入力してみてください。どんなデータが送られてきているでしょうか。

6. AD コンバータ

自然界の物理量，例えば，温度，湿度，重さ，明るさ，音などは全てアナログ量です。一方，これまで調べたことからお分かりのように，マイコンはデジタル値しか扱うことができません。ということは，マイコンでこういったものを扱うときは何らかの方法でアナログ値をデジタル値に変換する必要があります。このような働きをする I/O が AD コンバータです。温度制御をしたい，重さを量りたい，というように，ちょっと応用範囲を広げようとすると必ずアナログ値を扱わないといけなくなります。この章では AD コンバータの基本的な考え方と使い方を調べてみましょう。

■ AD コンバータとは

AD コンバータは，入力電圧に比例したデジタル値に変換する I/O です。通常 AD コンバータには最小入力電圧と最大入力電圧，そして変換ビット数が決まっています。例えば，0V から 5V まで入力できて，変換ビット数が 8 ビットのときは，0V のときは B'00000000（16 進数で 00h）に，5V のときは B'11111111（16 進数で FFh）に変換します。その間は比例するので，例えば 2.5V を入力すると 80h に変換します。



AD コンバータに入力できるのは電圧だけなので，温度や重さといった物理量を AD 変換するには，まず電圧に変換する必要があります。このようなデバイスをセンサと呼びます。一例ですが，温度を測るにはサーミスタや熱伝対，明るさを測るには Cds などを使います。

■ H8/3687 の AD コンバータ

H8/3687 には AD コンバータが内蔵されています。詳しくは,「H8/3687 グループ　ハードウェアマニュアル」(以降ハードウェアマニュアル)の 18-1 ページから説明されていますので, ぜひお読みください。いくつか特徴をあげておきましょう。

入力電圧

0V から AVcc までです。TK-3687mini は AVcc に 5V をつないでいますので, 最大入力電圧は 5V です。

分解能:10 ビット

0V のときに B'0000000000, AVcc(5V)のときに B'1111111111 になります。ただし, 変換結果は 16 ビットデータのうち上位 10 ビットにセットされ下位 6 ビットは 0 になります。というわけで, 0V のときは 0000h, AVcc(5V)のときは FFC0h になります。変換結果をプログラムで 6 ビット右シフトして 0000h～03FFh として扱うこともあります。もちろん, どうするかはプログラマしだいです。なお, これから作成するプログラムでは平均後の上位 8 ビットを使います。

入力チャンネル:8 チャンネル

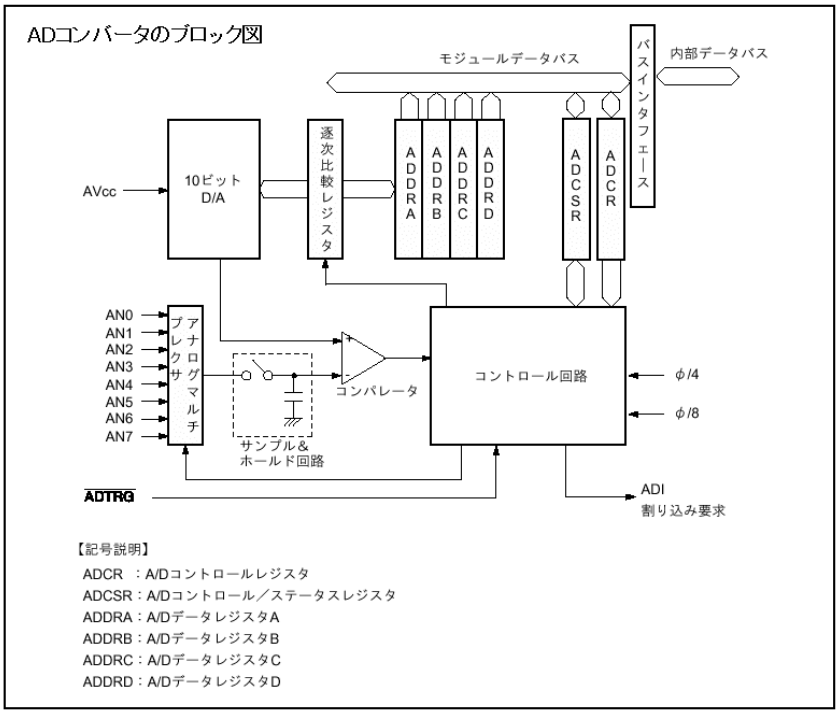
AD コンバータ自体は 1 個だけなのですが, アナログマルチプレクサ回路が内蔵されているので, 8 種類の電圧を入力することができます。そのため, 同時に 8 チャンネルの AD 変換ができるわけでなく, 順番に 1 チャンネルずつ AD 変換します。

動作モード

単一モードとスキャンモードの 2 種類があります。単一モードは指定された 1 チャンネルのアナログ入力を AD 変換します。一方, スキャンモードは指定された最大 4 チャンネルのアナログ入力を自動的に順番に AD 変換します。

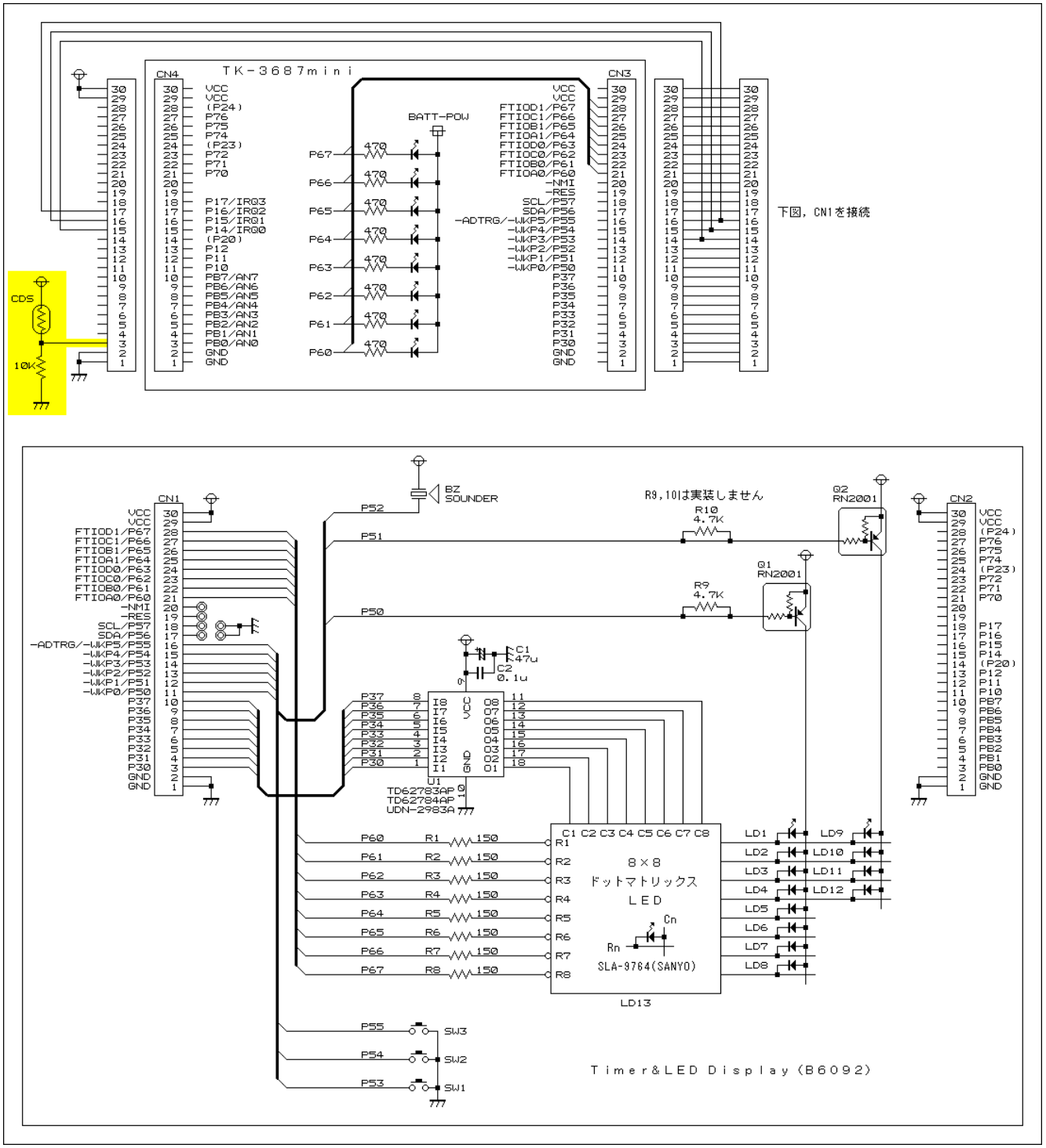
変換速度

TK-3687mini は CPU クロックが 20MHz なので, 1 チャンネルあたり最短 3.5 μ s で変換できます。



■ サンプルプログラム

では，AD コンバータで明るさを表示するプログラムを作ってみましょう。ここでは，CDS というセンサを使って明るさをマイコンに取り込んでみます。CDS は光のエネルギーで抵抗値が変化する素子です。明るいところでは 100Ω以下だったものが，暗くなると 10MΩ以上になるものがあります。回路図を再掲します。この回路で明るさを電圧に変換して AD コンバータに入力します。



AD コンバータで変換すること自体はそれほど難しくありません。ハードウェアマニュアルの 18-4 ページ，「18.3.2　AD コントロール/ステータスレジスタ」をご覧ください。ADST を 1 にすると AD 変換がスタートします。AD 変換が終了すると ADF が 1 になります。今回は AN0 の電圧を AD 変換します。それで，AD 変換が終了したら ADDRA からデータを入力します。

この AD 値をそのまま使えばよいかというと，そういうわけにはいきません。入力電圧が安定していてノイズがまったくなければよいのですが，現実にはそういう信号はなく，ノイズなどの影響で AD 値はふらふらします。そこで，何回か入力してその平均値を求めることでノイズの影響をなくします。今回はタイマ B1 で 1ms ごとに割込みをかけ，その都度 AD 変換し，256 回加算して平均しました。得られた平均値を表示します。

なお，表示はドットマトリックス LED に表示しました。SW1 を押すと 16 進数で，SW2 を押すと棒グラフ表示，SW3 を押すと折れ線グラフ表示します。今までの総決算です。(ちょっと大変かも・・・)

```
/************************************************************************/
/*                                                                      */
/*  FILE        :demo_adc.c                                             */
/*  DATE        :Wed, Jan 09, 2008                                      */
/*  DESCRIPTION :Main Program                                           */
/*  CPU TYPE    :H8/3687                                                */
/*                                                                      */
/*  This file is generated by Renesas Project Generator (Ver.4.9).      */
/*                                                                      */
/* -------------------------------------------------------------------- */
/*                                                                      */
/*  This file is programed by TOYO-LINX Co.,Ltd. / yKikuchi             */
/*                                                                      */
/************************************************************************/

/************************************************************************
    インクルードファイル
************************************************************************/
#include <machine.h>    //H8特有の命令を使う
#include "iodefine.h"   //内蔵I/Oのラベル定義

/************************************************************************
    定数の定義（直接指定）
************************************************************************/
//AD変換に関係した定数 ------------------------------------------------
#define       ADC_TIME  256       //加算回数(Max=65536)

//LED表示 -------------------------------------------------------------
#define       DRV_LOGIC 0x300     //ドライバの入力論理
                                  //負論理入力のビットを‘1’にする

/************************************************************************
    定数エリアの定義(ROM)
************************************************************************/
//スキャンデータ
const unsigned int ScanData[10]   =    {0x001,0x002,0x004,0x008
                                       ,0x010,0x020,0x040,0x080
                                       ,0x100,0x200};

//キャラクタデータ(4×8)
const unsigned char LEDDispData[][4] = {
     {0x00,0xff,0x81,0xff},  // 0
     {0x00,0x02,0xff,0x00},  // 1
     {0x00,0xf1,0x91,0x9f},  // 2
     {0x00,0x89,0x89,0xff},  // 3
```

```
    {0x00,0x1f,0x10,0xff},  // 4
    {0x00,0x8f,0x89,0xf9},  // 5
    {0x00,0xff,0x89,0xf9},  // 6
    {0x00,0x0f,0x01,0xff},  // 7
    {0x00,0xff,0x89,0xff},  // 8
    {0x00,0x9f,0x91,0xff},  // 9
    {0x00,0xff,0x11,0xff},  // A
    {0x00,0xff,0x88,0xf8},  // B
    {0x00,0xff,0x81,0x81},  // C
    {0x00,0xf8,0x88,0xff},  // D
    {0x00,0xff,0x89,0x89},  // E
    {0x00,0xff,0x09,0x09},  // F
};

/*************************************************************************
    グローバル変数の定義とイニシャライズ(RAM)
*************************************************************************/
// AD値に関係した変数 ---------------------------------------------
unsigned int  AdcData       =   0;        //平均値
unsigned long AdcBuf        =   0;        //加算バッファ
unsigned long AdcCnt        =   ADC_TIME; //加算カウンタ
unsigned int  AdcDataBuf[8] =   {0x0000,0x0000
                                ,0x0000,0x0000
                                ,0x0000,0x0000
                                ,0x0000,0x0000};    //過去データバッファ
unsigned char DispForm =    0;  //表示方式
                                // 0:16進数
                                // 1:バー
                                // 2:折れ線

// LED表示に関係した変数 ------------------------------------------
unsigned char ScanCnt       =   0;  //スキャンカウンタ
unsigned char DispFlag      =   1;  //表示フラグ
                                    // 0:消去
                                    // 1:通常表示
                                    // 2:反転表示
unsigned char DispBuf[10]   =   {0x00,0x00,0x00,0x00,0x00
                                ,0x00,0x00,0x00,0x00,0x00};  //表示バッファ

// スイッチ入力に関係した変数 -------------------------------------
unsigned char SwData1       =   0;  //ファーストリード
unsigned char SwData2       =   0;  //ダブルリードにより決定したデータ
unsigned char SwData3       =   0;  //前回のダブルリードで決定したデータ
unsigned char SwData4       =   0;  //0→1に変化したデータ
unsigned char SwStatus      =   0;  //スイッチ入力ステータス
                                    // 0:ファーストリード
                                    // 1:ダブルリード

/*************************************************************************
    関数の定義
*************************************************************************/
void          display_change(void);
void          display_set(void);
void          init_ad(void);
void          init_io(void);
```

```
void          init_tmb1(void);
void          init_tmv(void);
void          intprog_ad(void);
void          intprog_tmb1(void);
void          intprog_tmv(void);
void          main(void);
void          switch_in(void);

/************************************************************************
    メインプログラム
************************************************************************/
void main(void)
{
    // イニシャライズ ------------------------------------------------
    init_io();
    init_ad();
    init_tmb1();
    init_tmv();

    // メインループ --------------------------------------------------
    while(1){
        display_change();  //表示方式変更
        display_set();     //表示データセット
    }
}

/************************************************************************
    表示方式変更
************************************************************************/
void display_change(void)
{
    if      ((SwData4 & 0x08)==0x08){     //SW1が押された
        DispForm = 0;
        SwData4 = 0;
    }
    else if ((SwData4 & 0x10)==0x10){     //SW2が押された
        DispForm = 1;
        SwData4 = 0;
    }
    else if ((SwData4 & 0x20)==0x20){     //SW3が押された
        DispForm = 2;
        SwData4 = 0;
    }
}

/************************************************************************
    表示データセット
************************************************************************/
void display_set(void)
{
    unsigned char i,j;
    unsigned int d;

    // ドットマトリックスLED
    switch (DispForm){
```

```
        case 0:
            DispBuf[0] = LEDDispData[AdcData / 0x1000][0];
            DispBuf[1] = LEDDispData[AdcData / 0x1000][1];
            DispBuf[2] = LEDDispData[AdcData / 0x1000][2];
            DispBuf[3] = LEDDispData[AdcData / 0x1000][3];
            DispBuf[4] = LEDDispData[(AdcData / 0x0100) & 0x0f][0];
            DispBuf[5] = LEDDispData[(AdcData / 0x0100) & 0x0f][1];
            DispBuf[6] = LEDDispData[(AdcData / 0x0100) & 0x0f][2];
            DispBuf[7] = LEDDispData[(AdcData / 0x0100) & 0x0f][3];
            break;
        case 1:
            for (j=0; j<8; j++){
                d = 0x00ff;
                for (i=0; i<8-AdcDataBuf[j]/0x1ffe; i++){
                    d = d * 2;
                }
                DispBuf[j] = d & 0x00ff;
            }
            break;
        case 2:
            for (j=0; j<8; j++){
                d = 0x0001;
                for (i=0; i<8-AdcDataBuf[j]/0x1ffe; i++){
                    d = d * 2;
                }
                DispBuf[j] = d & 0x00ff;
            }
            break;
        default:
            for (i=0; i<8; i++){
                DispBuf[i] = 0x00;
            }
    }

    // 周囲のLED
    d = 0x0fff;
    for (i=0; i<12-AdcData/0x1554; i++){
        d = d / 2;
    }
    DispBuf[8] = d & 0x00ff;
    DispBuf[9] = d / 0x0100;
}

/************************************************************************
    I/Oポート イニシャライズ
************************************************************************/
void init_io(void)
{
    IO.PCR3       =   0xff;     //ポート3,P30-37出力
    IO.PDR3.BYTE  =   0x00 ^ DRV_LOGIC;

    IO.PMR5.BYTE  =   0x00;     //ポート5,汎用入出力ポート
    IO.PUCR5.BYTE =   0x38;     //ポート5,P53-55内蔵プルアップオン
    IO.PCR5       =   0x07;     //ポート5,P50-52出力,P53-P57入力
    IO.PDR5.BYTE  =   0x04 ^ (DRV_LOGIC / 0x100);
```

```
    IO.PCR6       =   0xff;     //ポート6,P60-67出力
    IO.PDR6.BYTE  =   0xff;
}

/************************************************************************
    A/D変換器イニシャライズ
************************************************************************/
void init_ad(void)
{
    AdcBuf        =   0;            //バッファクリア
    AdcCnt        =   ADC_TIME;     //カウンタセット
    AD.ADCSR.BYTE =   0x40;         //割り込みイネーブル,単一モード,134ステート,CH0
}

/************************************************************************
    ADC 割込み (A/D変換  &  平均化)
************************************************************************/
#pragma regsave (intprog_ad)
void intprog_ad(void)
{
    unsigned char i;

    AD.ADCSR.BIT.ADF = 0;             //ADエンドフラグクリア
    AdcBuf = AdcBuf + AD.ADDRA;       //加算
    AdcCnt = AdcCnt - 1;              //カウンタ-1
    if (AdcCnt==0){
        AdcData = (unsigned int)(AdcBuf / ADC_TIME);    //平均
        for (i=0; i<7; i++){
            AdcDataBuf[i] = AdcDataBuf[i+1];
        }
        AdcDataBuf[7] = AdcData;
        AdcBuf = 0;                   //バッファクリア
        AdcCnt = ADC_TIME;            //カウンタセット
    }
}

/************************************************************************
    タイマB1 イニシャライズ
************************************************************************/
void init_tmb1(void)
{
    TB1.TMB1.BYTE     =   0xfb;     //オートリロード, 内部クロックφ/256
    TB1.TLB1          =   0-78;     //周期=1ms(1000Hz=1KHz)
    IRR2.BIT.IRRTB1   =   0;        //タイマB1割込み要求フラグ クリア
    IENR2.BIT.IENTB1  =   1;        //タイマB1割込み要求イネーブル
}

/************************************************************************
    タイマB1 割込み(1ms)
************************************************************************/
#pragma regsave (intprog_tmb1)
void intprog_tmb1(void)
{
    //タイマB1割込み要求フラグ クリア
```

```
    IRR2.BIT.IRRTB1 = 0;

    //AD変換スタート
    AD.ADCSR.BIT.ADST = 1;

    //スイッチ入力
    switch_in();
}

/************************************************************************
    スイッチ入力
************************************************************************/
void switch_in(void)
{
    switch(SwStatus){
        case 0:
            SwData1 = ~IO.PDR5.BYTE & 0x38;
            if (SwData1!=0)     {SwStatus = 1;}
            else            {SwData2 = SwData3 =0;}
            break;
        case 1:
            if (SwData1==(~IO.PDR5.BYTE & 0x38)){
                SwData2 = SwData1;
                SwData4 = SwData4 | (SwData2 & (~SwData3));
                SwData3 = SwData2;
            }
            SwStatus = 0;
            break;
    }
}

/************************************************************************
    タイマV イニシャライズ
************************************************************************/
void init_tmv(void)
{
    TV.TCSRV.BYTE  =    0x00;     //TOMV端子は使わない
    TV.TCORA       =    156;      //周期=1ms(1kHz)
    TV.TCRV1.BYTE  =    0x01;     //TRGVトリガ入力禁止,
    TV.TCRV0.BYTE  =    0x4b;     //コンペアマッチA 割込みイネーブル
                                  //コンペアマッチA でTCNTVクリア
                                  //内部クロックφ/128(=156.25kHz)
}

/************************************************************************
    タイマV 割込み(1ms)
************************************************************************/
#pragma regsave (intprog_tmv)
void intprog_tmv(void)
{
    //コンペアマッチフラグA クリア
    TV.TCSRV.BIT.CMFA = 0;

    //表示を消す
    IO.PDR5.BYTE = (IO.PDR5.BYTE & 0xfc) | (0x00 ^ (DRV_LOGIC/0x100));
```

```
    IO.PDR3.BYTE = 0x00 ^ DRV_LOGIC;
    IO.PDR6.BYTE = 0xff;

    //データ出力
    if      (DispFlag==0)   {IO.PDR6.BYTE =  0xff;}
    else if (DispFlag==1)   {IO.PDR6.BYTE = ~DispBuf[ScanCnt];}
    else                    {IO.PDR6.BYTE =  DispBuf[ScanCnt];}

    //スキャン信号出力
    IO.PDR5.BYTE = ((unsigned char)((ScanData[ScanCnt] ^ DRV_LOGIC) / 0x100)) | (IO.PDR5.BYTE & 0xfc);
    IO.PDR3.BYTE =  (unsigned char)((ScanData[ScanCnt] ^ DRV_LOGIC) & 0x0ff);

    //次のスキャンのセット
    ScanCnt++; if (ScanCnt>=10) {ScanCnt = 0;}
}
```

割込みを使うためにはソースファイルだけではなく, HEW が自動生成する‘intprg. c’を修正する必要があります。下記のリストをご覧ください。

```
/************************************************************************/
/*                                                                      */
/*  FILE        :intprg.c                                               */
/*  DATE        :Wed, Jan 09, 2008                                      */
/*  DESCRIPTION :Interrupt Program                                      */
/*  CPU TYPE    :H8/3687                                                */
/*                                                                      */
/*  This file is generated by Renesas Project Generator (Ver.4.9).      */
/*                                                                      */
/************************************************************************/



#include <machine.h>

extern void intprog_ad(void);
extern void intprog_tmb1(void);
extern void intprog_tmv(void);

#pragma section IntPRG
//  vector 1 Reserved

//  vector 2 Reserved

//  vector 3 Reserved

//  vector 4 Reserved

//  vector 5 Reserved

//  vector 6 Reserved

//  vector 7 NMI
__interrupt(vect=7) void INT_NMI(void) {/* sleep(); */}
//  vector 8 TRAP #0
```

```
__interrupt(vect=8) void INT_TRAP0(void) {/* sleep(); */}
//  vector 9 TRAP #1
__interrupt(vect=9) void INT_TRAP1(void) {/* sleep(); */}
//  vector 10 TRAP #2
__interrupt(vect=10) void INT_TRAP2(void) {/* sleep(); */}
//  vector 11 TRAP #3
__interrupt(vect=11) void INT_TRAP3(void) {/* sleep(); */}
//  vector 12 Address break
__interrupt(vect=12) void INT_ABRK(void) {/* sleep(); */}
//  vector 13 SLEEP
__interrupt(vect=13) void INT_SLEEP(void) {/* sleep(); */}
//  vector 14 IRQ0
__interrupt(vect=14) void INT_IRQ0(void) {/* sleep(); */}
//  vector 15 IRQ1
__interrupt(vect=15) void INT_IRQ1(void) {/* sleep(); */}
//  vector 16 IRQ2
__interrupt(vect=16) void INT_IRQ2(void) {/* sleep(); */}
//  vector 17 IRQ3
__interrupt(vect=17) void INT_IRQ3(void) {/* sleep(); */}
//  vector 18 WKP
__interrupt(vect=18) void INT_WKP(void) {/* sleep(); */}
//  vector 19 RTC
__interrupt(vect=19) void INT_RTC(void) {/* sleep(); */}
//  vector 20 Reserved

//  vector 21 Reserved

//  vector 22 Timer V
__interrupt(vect=22) void INT_TimerV(void) {intprog_tmv();}
//  vector 23 SCI3
__interrupt(vect=23) void INT_SCI3(void) {/* sleep(); */}
//  vector 24 IIC2
__interrupt(vect=24) void INT_IIC2(void) {/* sleep(); */}
//  vector 25 ADI
__interrupt(vect=25) void INT_ADI(void) {intprog_ad();}
//  vector 26 Timer Z0
__interrupt(vect=26) void INT_TimerZ0(void) {/* sleep(); */}
//  vector 27 Timer Z1
__interrupt(vect=27) void INT_TimerZ1(void) {/* sleep(); */}
//  vector 28 Reserved

//  vector 29 Timer B1
__interrupt(vect=29) void INT_TimerB1(void) {intprog_tmb1();}
//  vector 30 Reserved

//  vector 31 Reserved

//  vector 32 SCI3_2
__interrupt(vect=32) void INT_SCI3_2(void) {/* sleep(); */}
```

では，ビルドして実行してみましょう。なお，付属の CD-ROM にはあらかじめダウンロードするファイルがおさめられています。‘demo_adc．mot’をダウンロードして実行してください。

第6章 μITRONを実装しよう

パソコンのプログラムはWindowsやLinuxなどOSの上で実行されます。一方，組み込みシステムのプログラムにおいてはOSを実装せずにプログラムすることが多かったように感じます。しかし，近年の組み込みシステムの高機能化や大規模化，ネットワーク対応やGUIの搭載などにより，システムを効率よく制御するためOSの必要性が高まってきました。ITRONは社団法人トロン協会が推進し，日本の半導体メーカ，ソフト開発メーカが参画したトロンプロジェクトの中で作成された組み込みシステム向け汎用リアルタイムOSの標準仕様です。

ところで，μITRONを手に入れようとしても，μITRONという名前のOSはありません。トロンプロジェクトでは，ITRONの仕様書は配布していますが，実際に動くコードは配布していないからです。企業や団体，個人が，配布されている仕様書を元にコードを書き，「μITRON仕様準拠OS」として配布，販売しています。このマニュアルで利用する「HOS」もプロジェクトHOSが開発したμITRON仕様のフリーのOSです。（「μITRON仕様に準拠している」と言うためにμITRONの全ての仕様を持たせる必要はありません。必要な最低限の機能も仕様書に規定されています。）

「HOS」は「Hyper Operating System」の略です。（ちなみに，90年代半ばに連載され，OVAやテレビアニメ，映画にもなったコミック「起動警察パトレイバー」に登場するレイバーと呼ばれる作業用ロボットに搭載されているOSの名称が「HOS」で，これにちなんで名付けられたそうです。）「HOS」は高い移植性を持っていて，H8だけではなく，SHやARM，IA32にも移植されています。ライセンス条項に従う限り，商用，非商用に関係なく，自由に利用し再配布することができます。（ライセンス条項はプロジェクトHOSの配布ファイル「licence. txt」を参照，元の著作権情報を削除しない，著作者は利用した損害の責任はとらない，著作者はサポートの義務は負わない，など）

実際のところ，H8程度の規模のマイコンであればOSを実装しなくてもほとんどのプログラムはできてしまいます。プログラムの開発効率もそれほどかわりません。それでも，リアルタイムOSを使うと，これまでとは違った感覚でプログラムすることができます。最初の目標は，プロジェクトHOSが配布している H8/3664（16MHz）用のサンプルプログラムを TK-3687/TK-3687mini（つまり H8/3687，20MHz）で動かすことです。

μITRON4.0 仕様に準拠したソフトウェアの製品マニュアルに入れるよう強く推奨されている文言があります。それに従い，ここで入れておきます。

- TRONは“The Real-time Operating system Nucleus”の略称です。
- ITRONは“Industrial TRON”の略称です。
- μITRONは“Micro Industrial TRON”の略称です。
- TRON，ITRON，およびμITRONは，特定の商品ないしは商品群を指す名称ではありません。

μITRON4.0 仕様に準拠したソフトウェアの製品マニュアルに入れるよう推奨されている文言があります。それに従い，ここで入れておきます。

- μITRON4.0 仕様は，トロン協会が定めたオープンなリアルタイムカーネル仕様です。μITRON4.0 仕様の仕様書は，トロン協会のホームページ（http://www.assoc.tron.org/）から入手することができます。

1. 開発に必要なものを用意する

開発環境 ： HEW

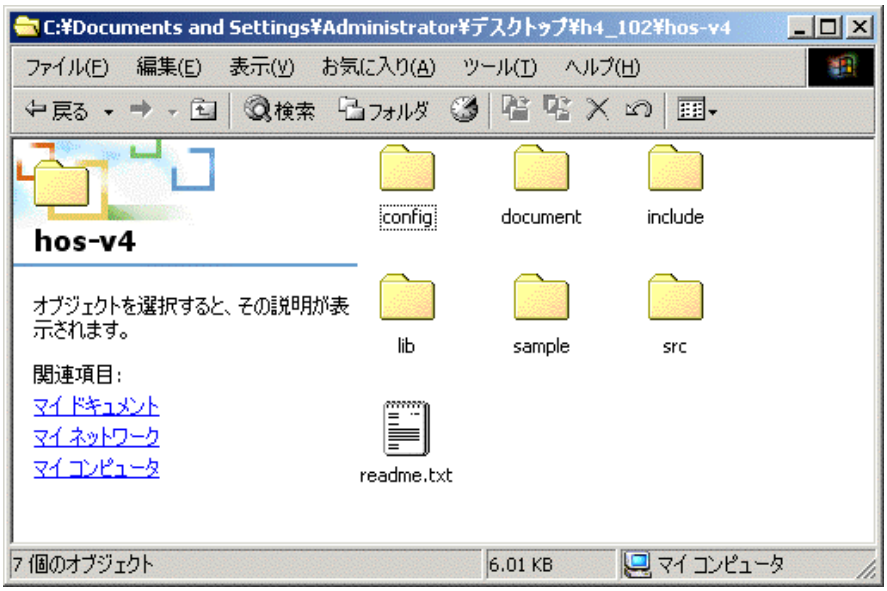
ルネサステクノロジのサイト(**http://japan.renesas.com/**)からダウンロードします。

HOS のソースコード

プロジェクトHOSのサイト(**https://sourceforge.jp/projects/hos/**)から「hos-v4」をダウンロードしてください。HOS-V4 の中にも圧縮形式の異なるファイルがあるのですが，Windows の場合は「HOS-V4<バージョンナンバー>LZH」をダウンロードします。このマニュアル執筆時点のファイル名は「h4_102.lzh」でした。

解凍ソフト

WindowsXP あたりからは ZIP 形式の圧縮ファイル(ファイル拡張子 ZIP)であれば標準で解凍することができるのですが，HOS は LHA 形式(ファイル拡張子 LZH)で圧縮されているため解凍ソフトが必要です。もっとも，LHA 形式は日本で DOS の時代から使われていて，日本ではデファクトスタンダードともいえる圧縮形式なので，日本で作られたものであればフリーソフトをはじめほとんどの解凍ソフトで対応できるはずです。気に入ったものをご用意ください。「h4_102.lzh」を解凍すると「hos_v4」というフォルダが表われ，その中に次のようなファイルがあります。



書き込みツール ： FDT

HOS はアプリケーションといっしょにフラッシュメモリに書き込みます。ルネサステクノロジのサイトからダウンロードしてください。

エミュレータ ： E8a

HOS はアプリケーションといっしょにフラッシュメモリに書き込む関係上，TK-3687/TK-3687mini に搭載されているハイパーH8 ではデバッグできません。購入することをおすすめします。

ターミナルソフト

サンプルプログラムの動作確認に利用します。

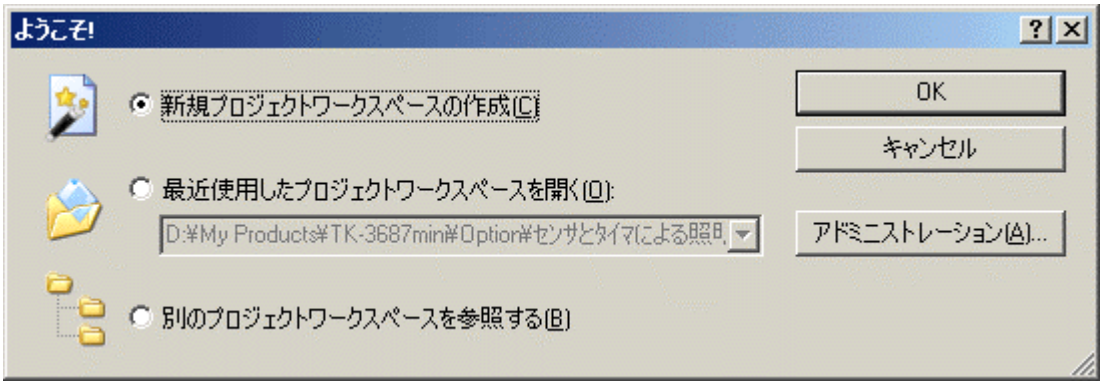
μITRON の仕様書

トロンプロジェクトのサイト(**http://www.assoc.tron.org/itron/home-j.html**)から「μITRON4.0 仕様」というドキュメントをダウンロードしてください。

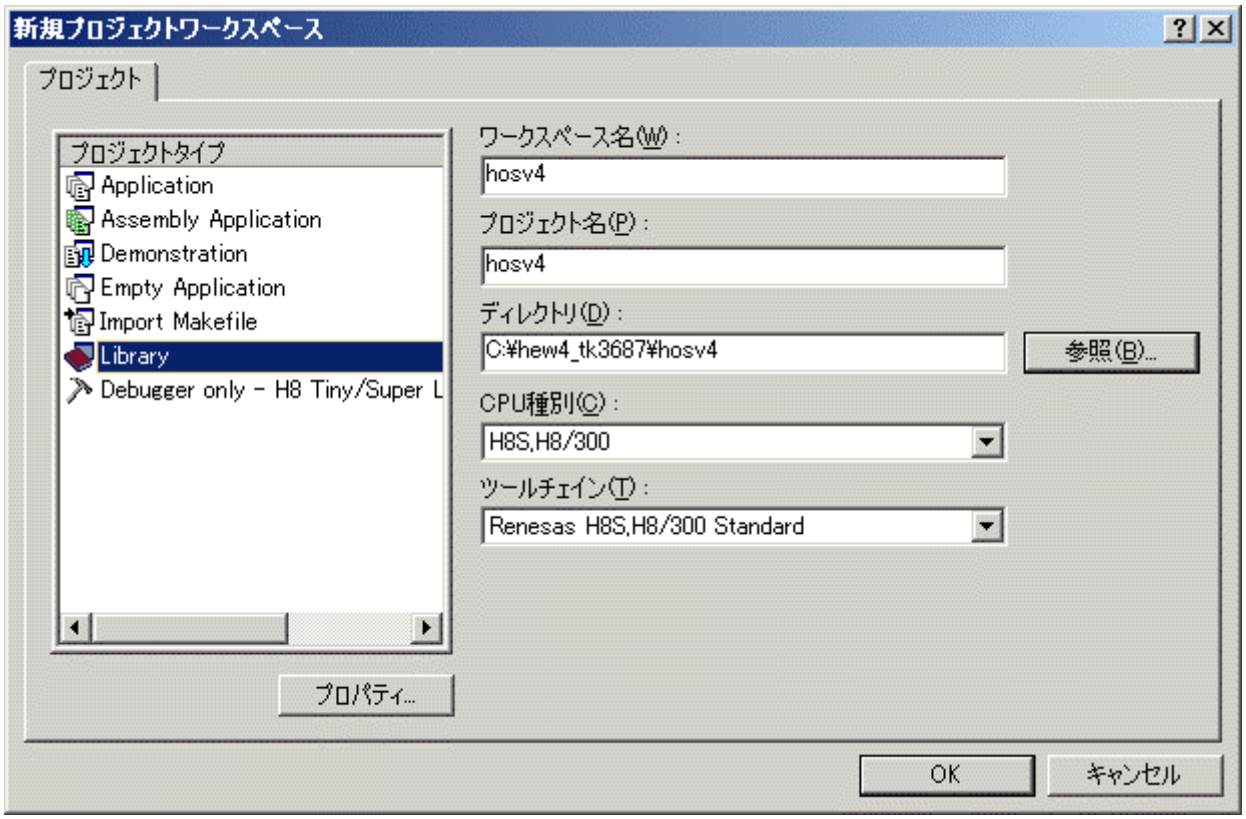
2. カーネルライブラリの構築

HOS のアプリケーションの開発では，HOS のカーネルをライブラリとして生成しておき，このライブラリをリンカから呼び出す方法を取ります。それで，まずカーネルライブラリを構築しましょう。

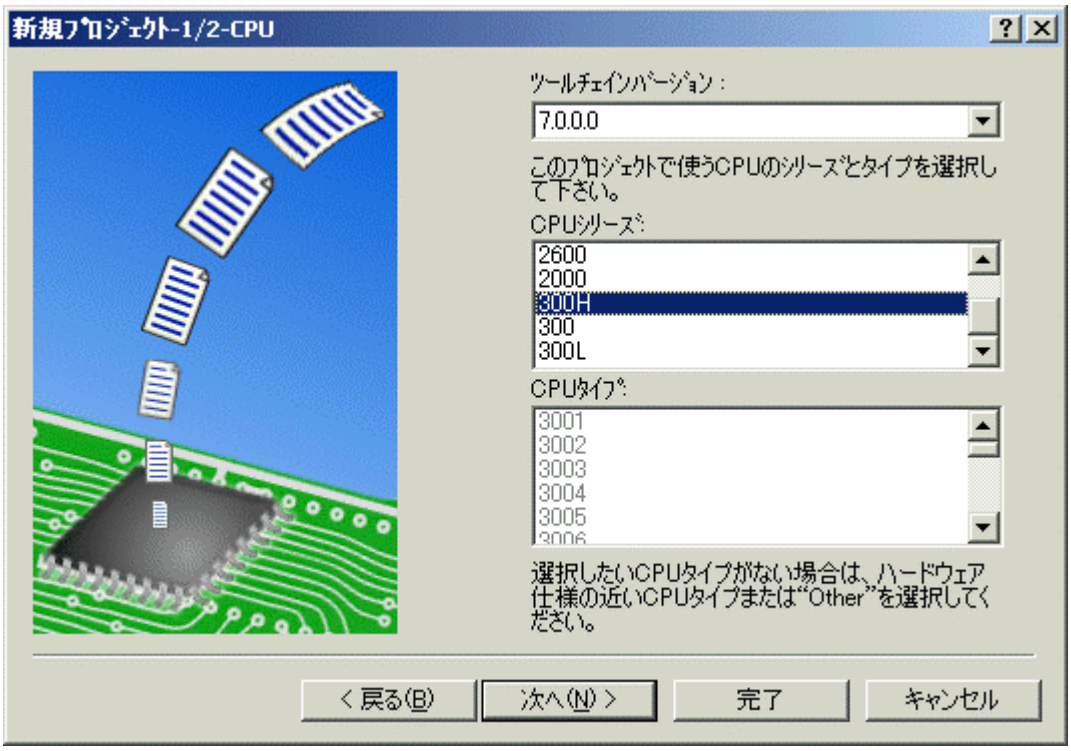
HEW を起動すると「ようこそ！」というダイアログが表示されるので，「新規プロジェクトワークスペースの作成」を選択して「OK」をクリックします。



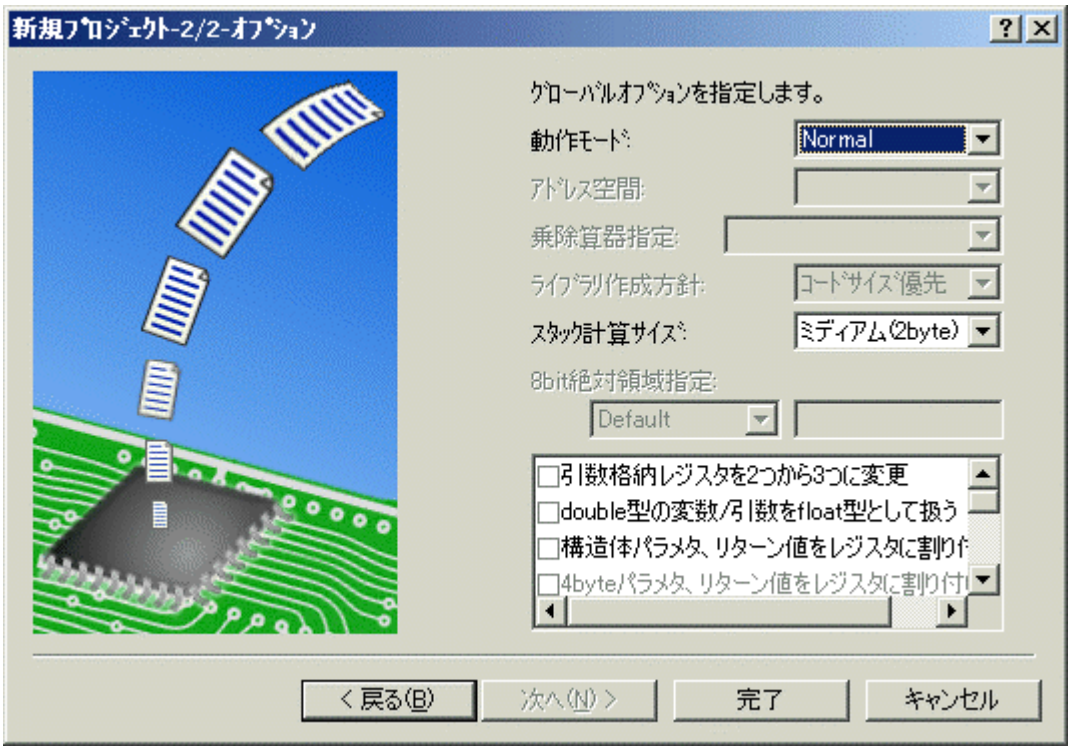
プロジェクトタイプは「Library」を指定します。ワークスペース名とプロジェクト名は「hosv4」，ディレクトリは「C:¥hew_tk3687¥hosv4」を入力します。また，CPU 種別は「H8S,H8/300」，ツールチェインは「Reneas H8S,H8/300 Standard」を選択します。入力が終わったら「OK」をクリックします。



「新規プロジェクト-1/2-CPU」で，CPU シリーズとして「300H」を選択して「次へ」をクリックします。

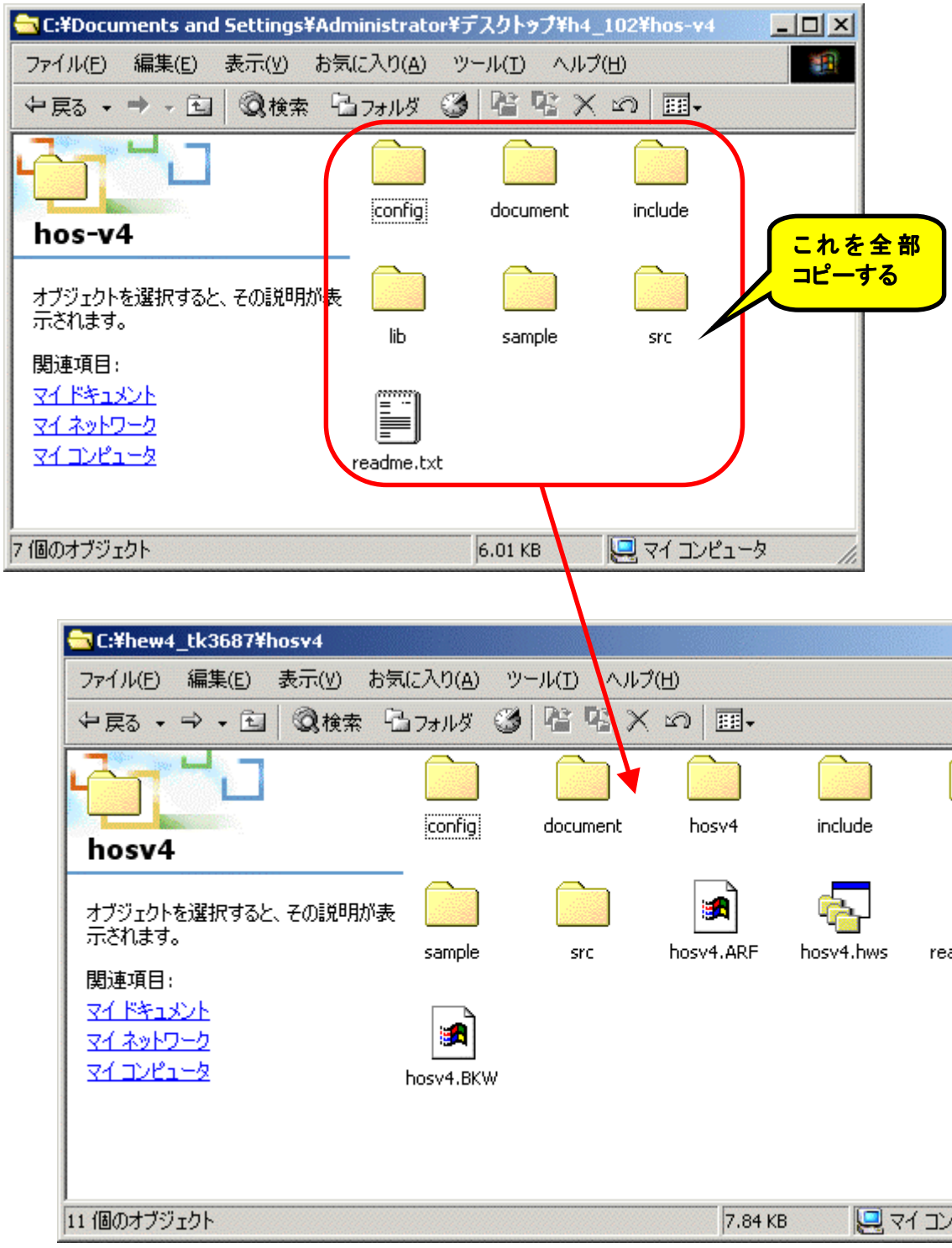


「新規プロジェクト-2/2-オプション」で，動作モードは「Normal」を選択して「完了」をクリックします。

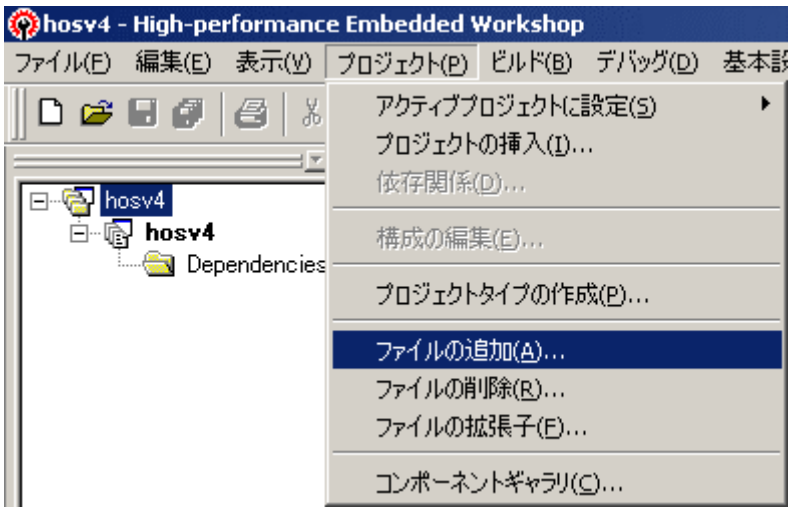


「概要」ダイアログが表示されますので「OK」をクリックします。プロジェクトが起動します。

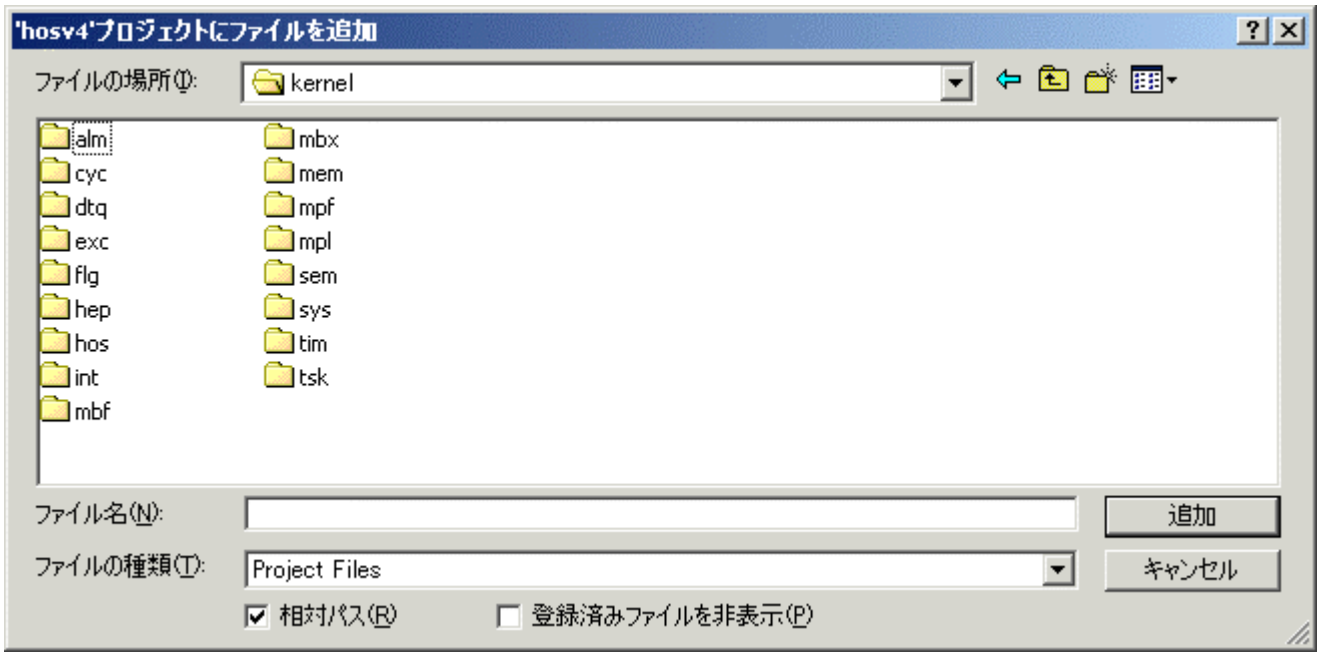
次に，HOS のソースファイルをプロジェクトに追加します。まず，解凍した「hos-v4」フォルダの中身を全て，今回作成した「hosv4」フォルダにコピーします。

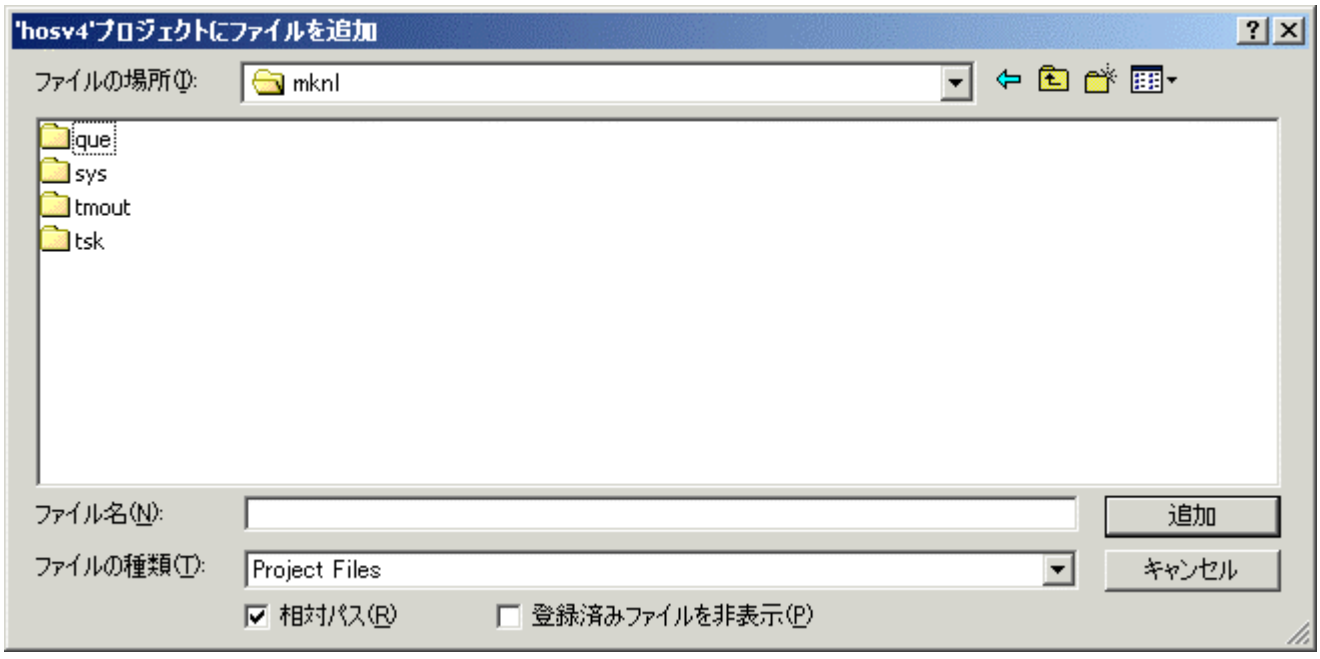


今コピーしたソースファイルをプロジェクトに登録します。登録するのは「C:¥hew4_tk3687¥hosv4¥src¥kernel¥」以下のフォルダにある全てのファイルと，「 C:¥hew4_tk3687¥hosv4¥src¥mknl¥ 」以下のフォルダにある全てのファイルです。メニューから「プロジェクト」→「ファイルの追加」を選択します。

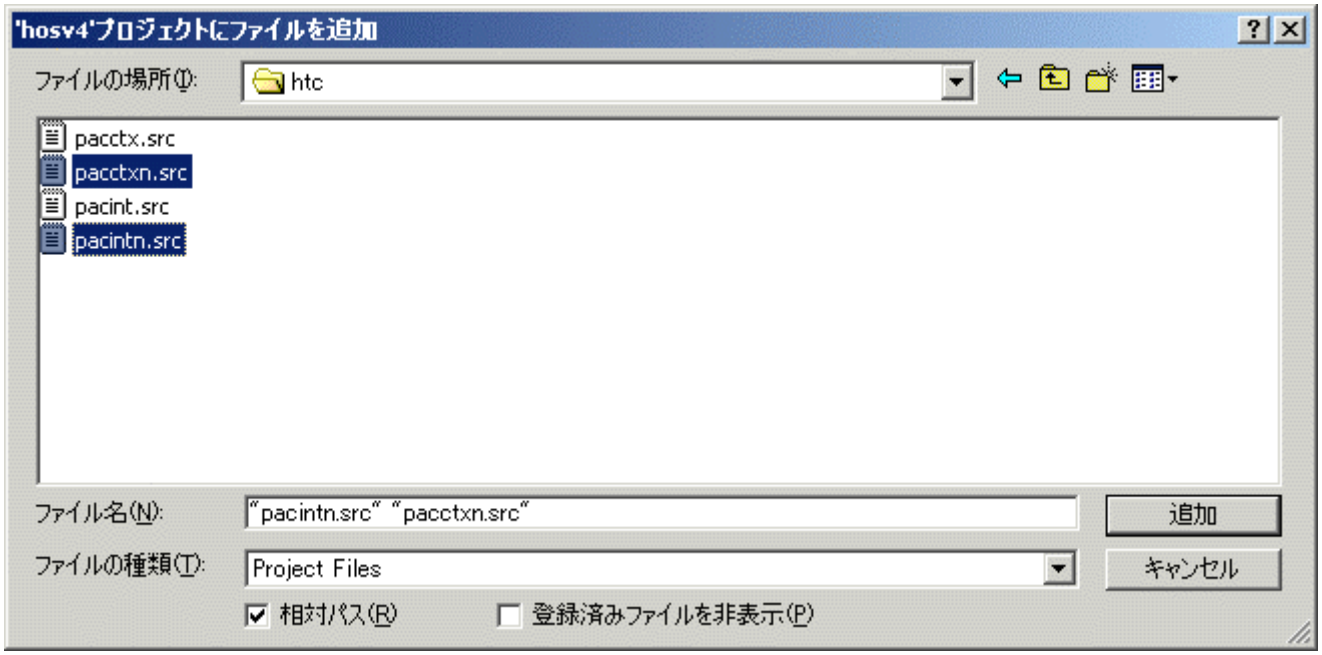


すると，「プロジェクトにファイルを追加」ダイアログが表示されますので，ファイルを選択して「追加」をクリックします。次の図は「kernel」フォルダと「mknl」フォルダを表示していますが，実際はさらにこの先のフォルダ（例えば「alm」フォルダ）を開いて，そこにある全てのファイルを追加します。「相対パス」にチェックを忘れずに入れてください。

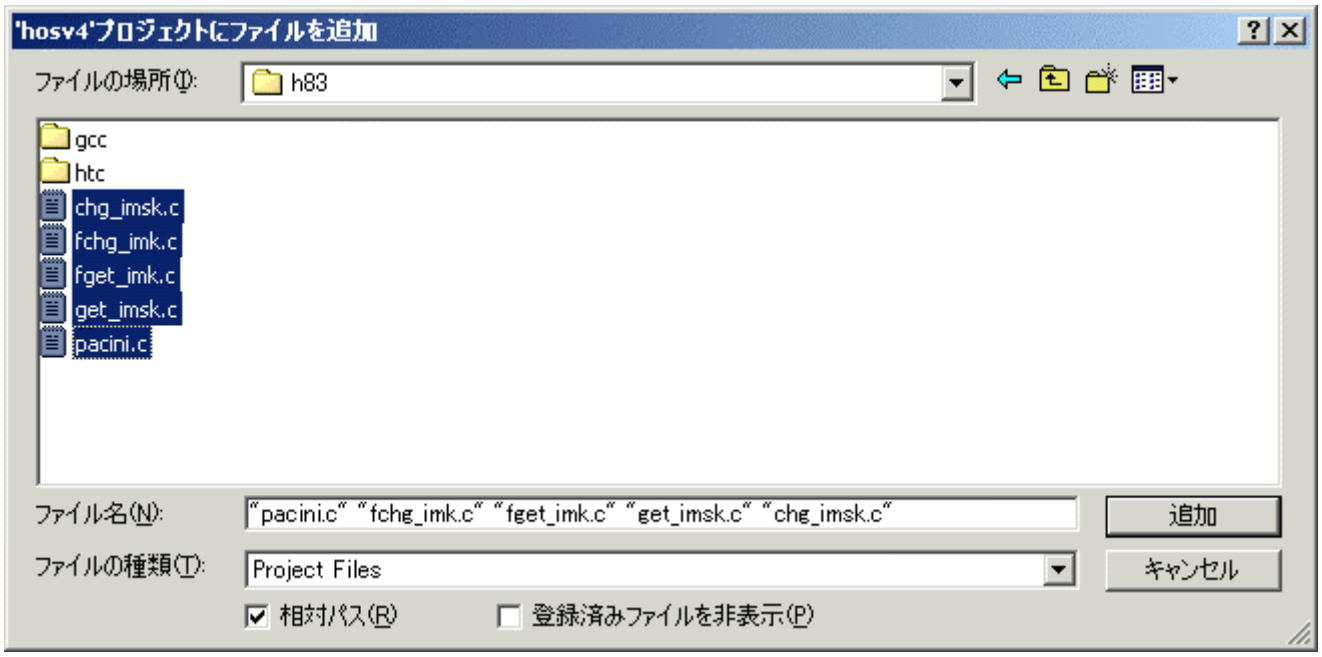




次に，CPU や開発環境に依存するプログラムをプロジェクトに追加します。「C:¥hew4_tk3687¥hosv4¥src¥h83¥htc¥」フォルダの「pacctxn.src」と「pacintn.src」です。

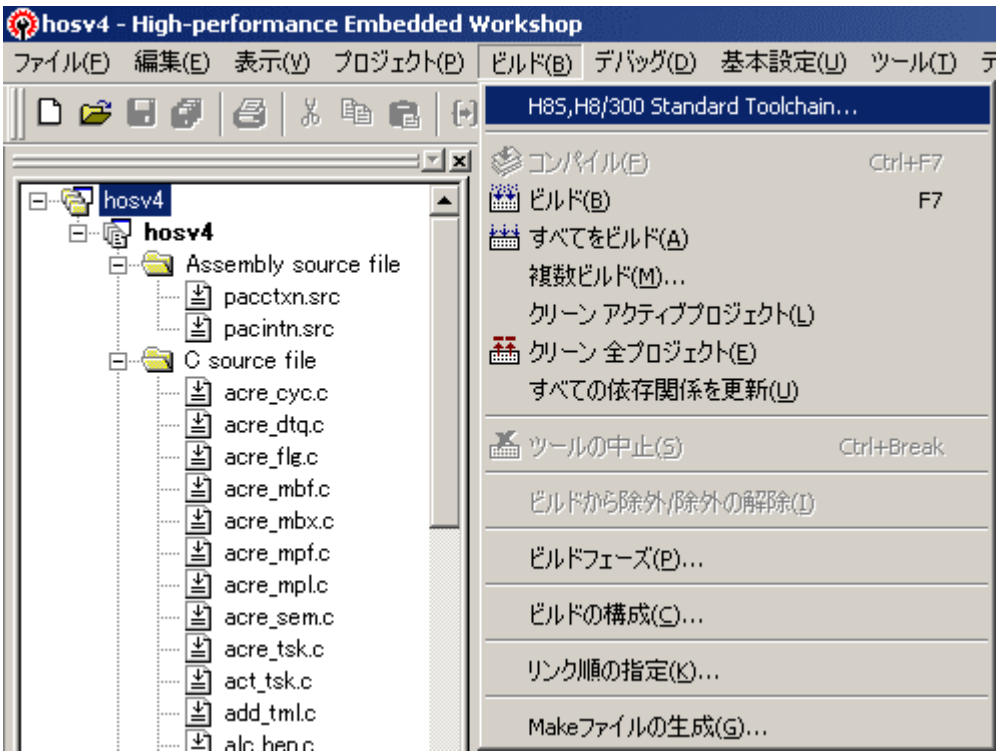


さらに，「C:¥hew4_tk3687¥hosv4¥src¥h83¥」フォルダのソースファイルもすべて追加します。

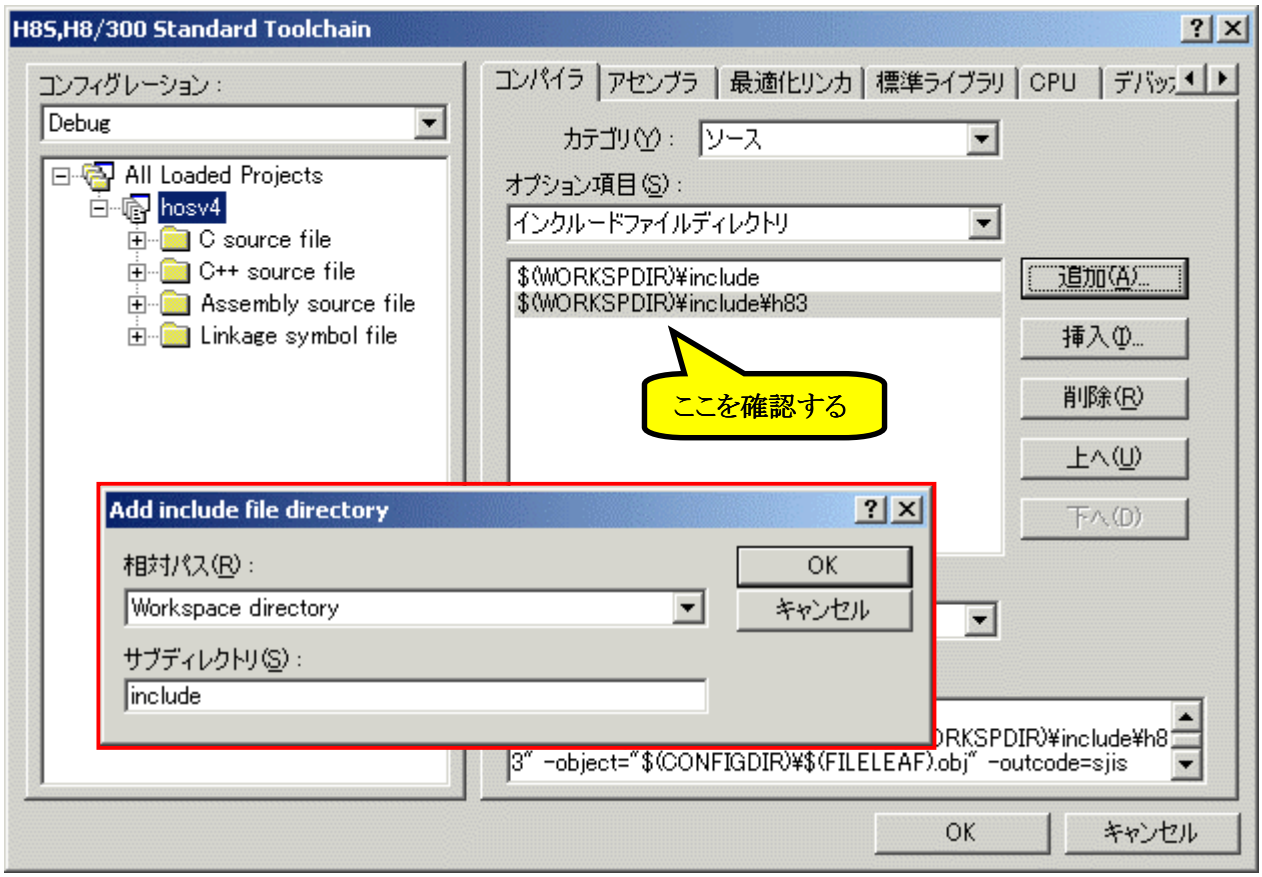


追加するファイルが多いので，追加し忘れないように注意してください。特に「kernel」フォルダと「mknl」フォルダは，その先のフォルダを一つ一つ開いてその中の全てのファイルを指定して追加するため忘れがちです。

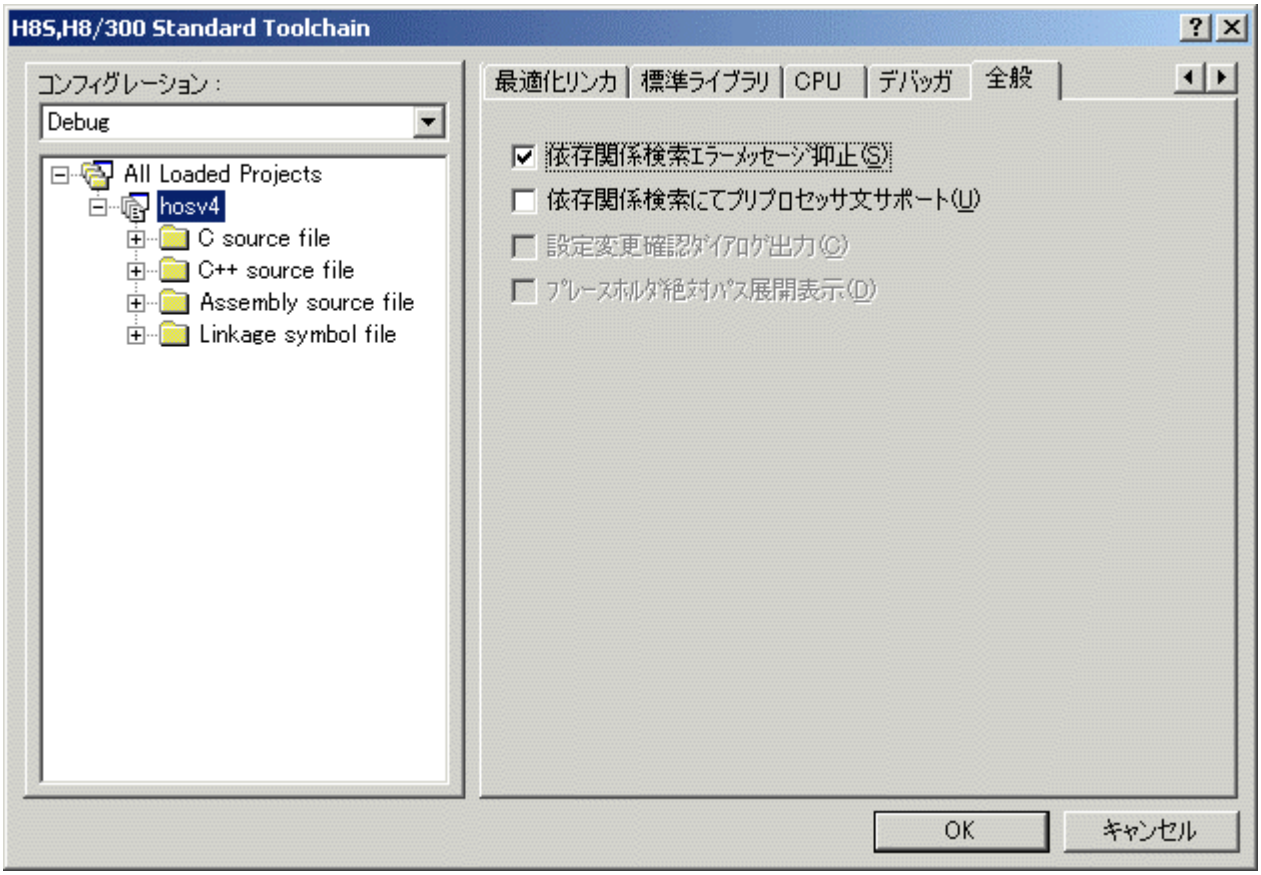
次に，インクルードファイルのパスを指定します。メニューから「ビルド」→「H8S,H8/300 Standard Toolchain...」を選択します。



「コンパイラ」タブを開き，「ソース」カテゴリのオプション項目「インクルードファイルディレクトリ」を選択して「追加」をクリックします。ダイアログが開きますので，「相対パス」は「Workspace directory」を選択し，「サブディレクトリ」は「include」を指定して「OK」をクリックします。もう一度「追加」をクリックしてダイアログを開き，「相対パス」は「Workspace directory」を選択し，「サブディレクトリ」は「include¥h83」を指定して「OK」をクリックします。



ところで，このままビルドすると大量の「Warning」が発生します。メニューから「ビルド」→「H8S,H8/300 Standard Toolchain...」を選択してください。「全般」タブの「依存関係検索エラーメッセージ抑止」にチェックを入れます。



さて，ここでH8/3687用にカスタマイズするためにソースファイルを一部修正します。H8/3664の割り込みベクタ番号は0～25ですが，H8/3687の割り込みベクタ番号は0～32です。それで，割り込み処理を行なっている「pacintn.src」を次のように修正します。(リストは一部を抜粋しています)

```
        .EXPORT   _hos_vector007
        .EXPORT   _hos_vector008
        .EXPORT   _hos_vector009
        .EXPORT   _hos_vector010
        .EXPORT   _hos_vector011
        .EXPORT   _hos_vector012
        .EXPORT   _hos_vector013
        .EXPORT   _hos_vector014
        .EXPORT   _hos_vector015
        .EXPORT   _hos_vector016
        .EXPORT   _hos_vector017
        .EXPORT   _hos_vector018
        .EXPORT   _hos_vector019
        .EXPORT   _hos_vector021
        .EXPORT   _hos_vector022
        .EXPORT   _hos_vector023
        .EXPORT   _hos_vector024
        .EXPORT   _hos_vector025
        .EXPORT   _hos_vector026
        .EXPORT   _hos_vector027
```

追加

```
        .EXPORT   _hos_vector028
        .EXPORT   _hos_vector029
        .EXPORT   _hos_vector030
        .EXPORT   _hos_vector031
        .EXPORT   _hos_vector032


;---------------------------------------------
; 割り込みハンドラ
;---------------------------------------------
_hos_vector007:     push.w    r0
        mov.b     #7, r0l
        bra  int_handler:16

_hos_vector008:     push.w    r0
        mov.b     #8, r0l
        bra  int_handler:16

_hos_vector009:     push.w    r0
        mov.b     #9, r0l
        bra  int_handler:16

_hos_vector010:     push.w    r0
        mov.b     #10, r0l
        bra  int_handler

_hos_vector011:     push.w    r0
        mov.b     #11, r0l
        bra  int_handler

_hos_vector012:     push.w    r0
        mov.b     #12, r0l
        bra  int_handler

_hos_vector013:     push.w    r0
        mov.b     #13, r0l
        bra  int_handler

_hos_vector014:     push.w    r0
        mov.b     #14, r0l
        bra  int_handler

_hos_vector015:     push.w    r0
        mov.b     #15, r0l
        bra  int_handler

_hos_vector016:     push.w    r0
        mov.b     #16, r0l
        bra  int_handler

_hos_vector017:     push.w    r0
        mov.b     #17, r0l
        bra  int_handler

_hos_vector018:     push.w    r0
```

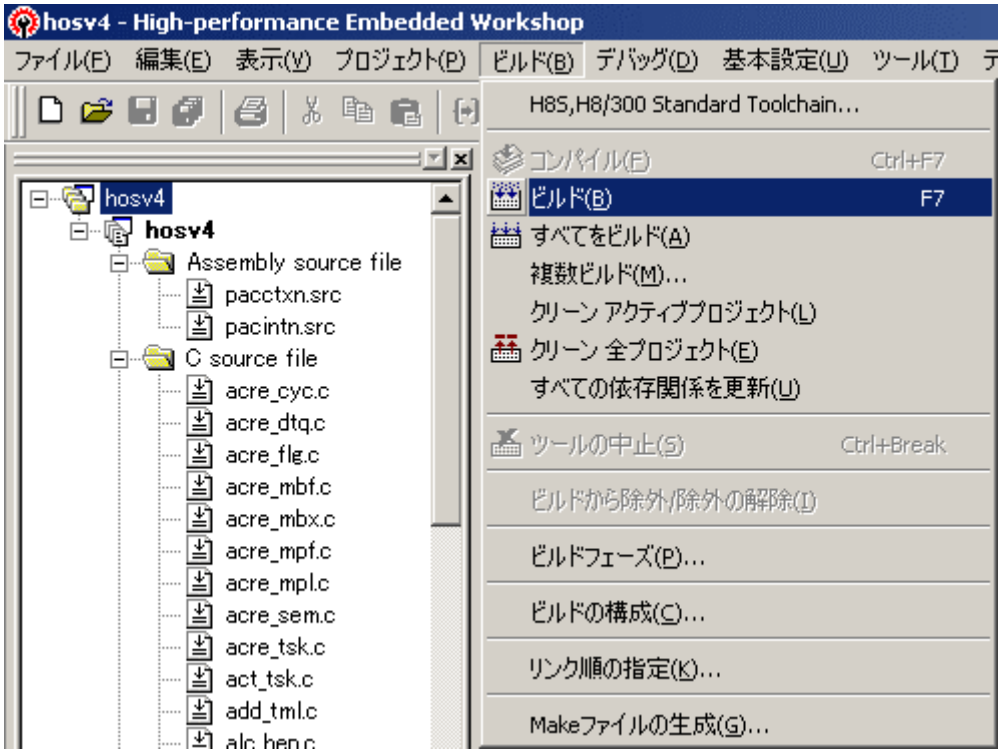
修正:コード追加に伴いジャンプ先が遠くなったため

修正:コード追加に伴いジャンプ先が遠くなったため

修正:コード追加に伴いジャンプ先が遠くなったため

```
        mov.b   #18, r0l
        bra  int_handler

_hos_vector019:   push.w   r0
        mov.b   #19, r0l
        bra  int_handler

_hos_vector021:   push.w   r0
        mov.b   #21, r0l
        bra  int_handler

_hos_vector022:   push.w   r0
        mov.b   #22, r0l
        bra  int_handler

_hos_vector023:   push.w   r0
        mov.b   #23, r0l
        bra  int_handler

_hos_vector024:   push.w   r0
        mov.b   #24, r0l
        bra  int_handler

_hos_vector025:   push.w   r0
        mov.b   #25, r0l
        bra  int_handler

_hos_vector026:   push.w   r0
        mov.b   #26, r0l
        bra  int_handler

_hos_vector027:   push.w   r0
        mov.b   #27, r0l
        bra  int_handler

_hos_vector028:   push.w   r0
        mov.b   #28, r0l
        bra  int_handler

_hos_vector029:   push.w   r0
        mov.b   #29, r0l
        bra  int_handler

_hos_vector030:   push.w   r0
        mov.b   #30, r0l
        bra  int_handler

_hos_vector031:   push.w   r0
        mov.b   #31, r0l
        bra  int_handler

_hos_vector032:   push.w   r0
        mov.b   #32, r0l
        bra  int_handler
```

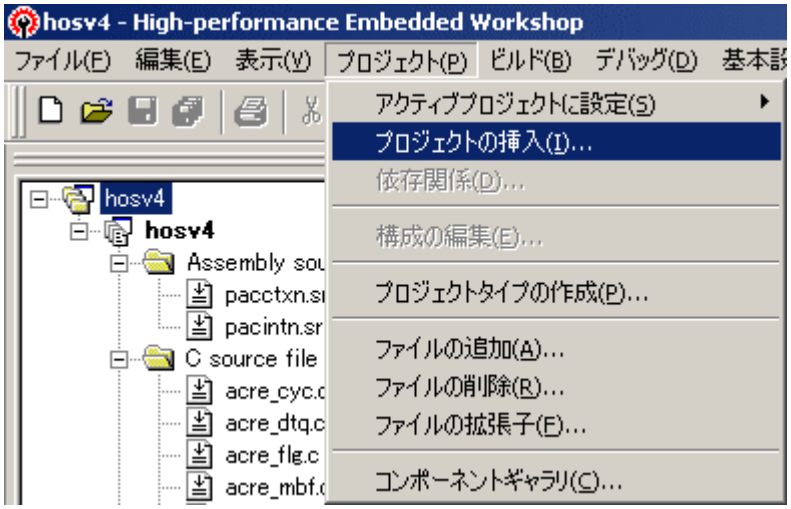
では，ビルドしてライブラリを構築しましょう。



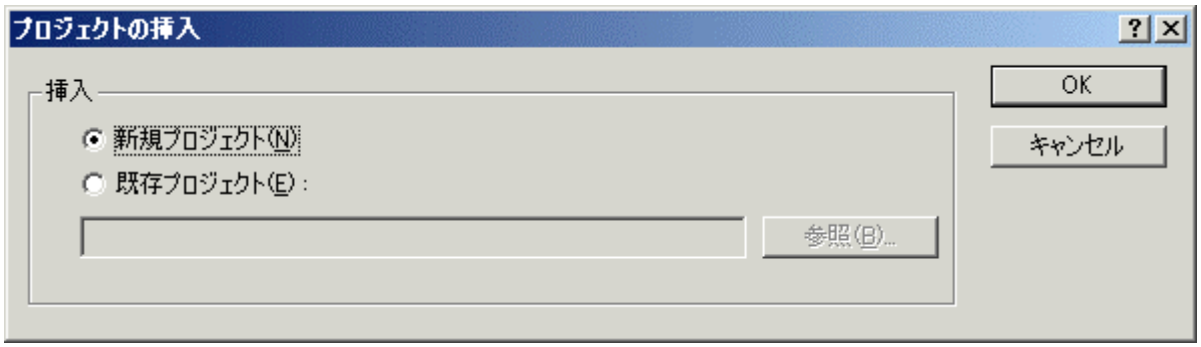
最終的にいくつか「Warning」が出ますが無視して構いません。「C:¥hew4_tk3687¥hosv4¥hosv4¥debug¥」フォルダに「hosv4.lib」というファイルができているはずです。これをアプリケーションから呼び出します。

3. プロジェクトの作成

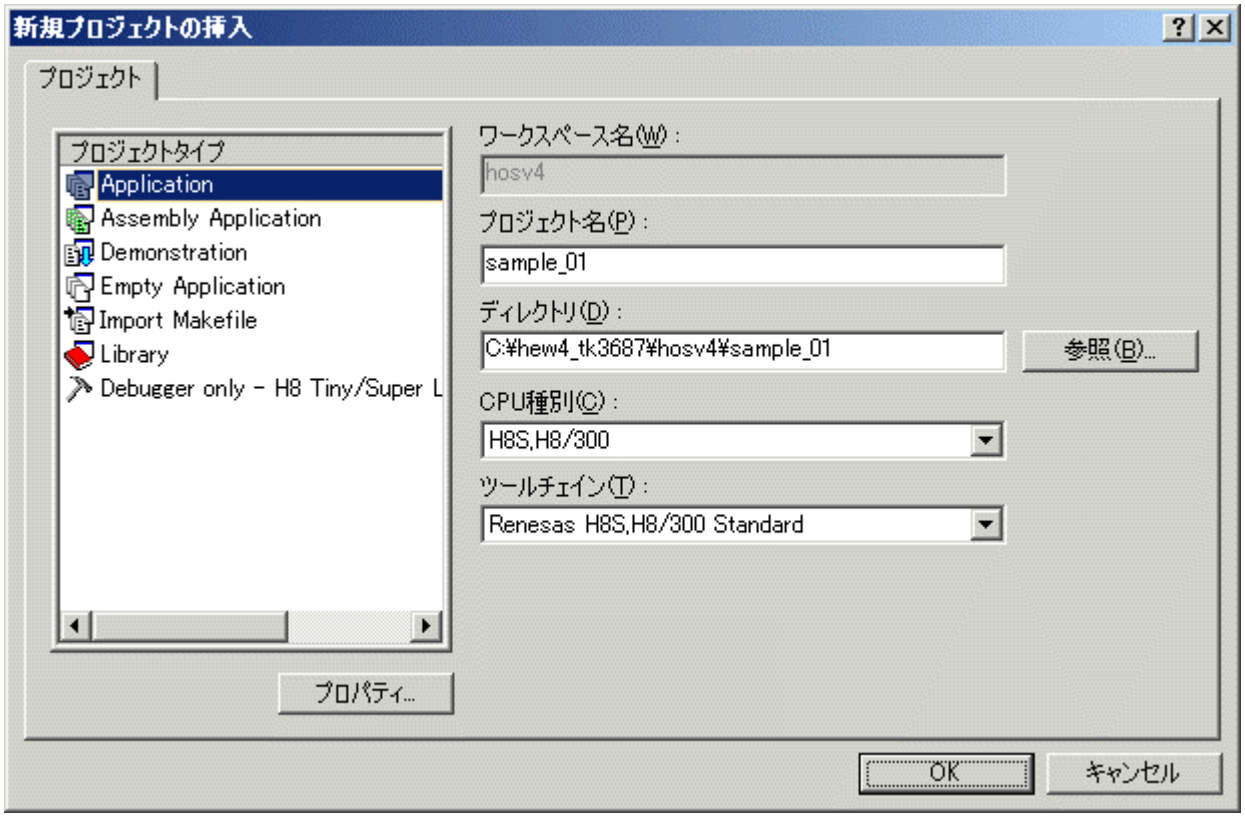
HOS のカーネルライブラリを構築したワークスペースに，アプリケーションプロジェクトを追加します。もしワークスペースを終了しているなら，「hosv4.hws」をダブルクリックしてワークスペースを起動してください。ライブラリを構築した状態になります。ここから，アプリケーションプロジェクトを挿入しましょう。メニューから「プロジェクト」→「プロジェクトの挿入」を選択します。



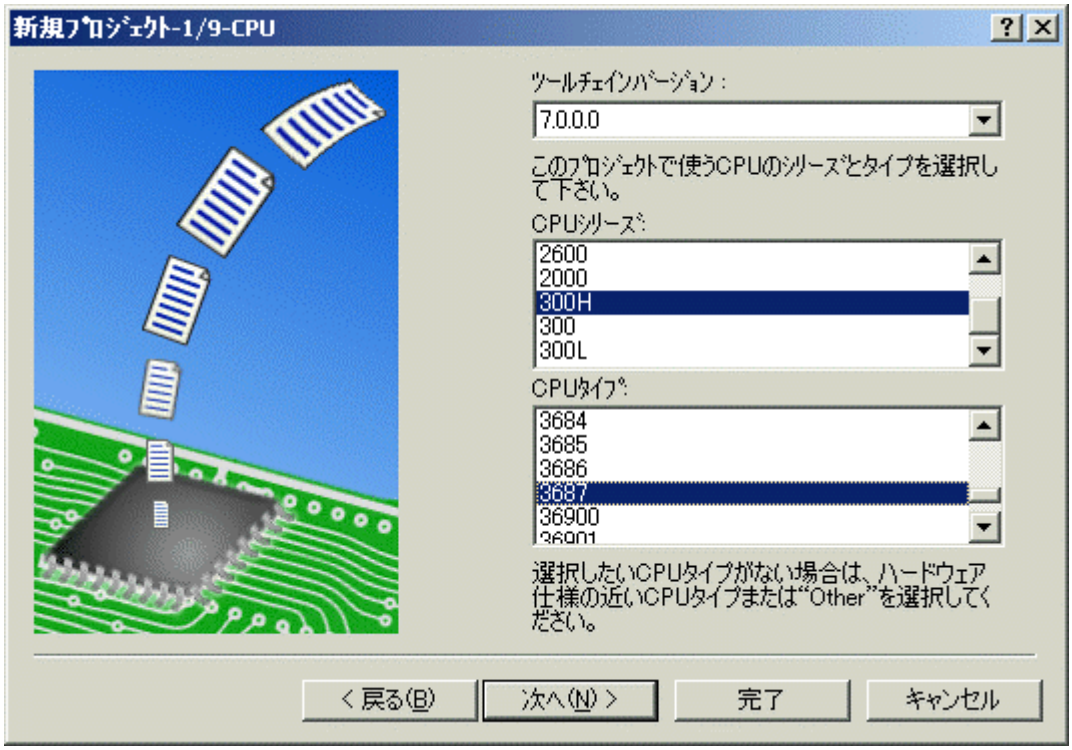
「プロジェクトの挿入」ダイアログが表示されますので，「新規プロジェクト」にチェックを入れて「OK」をクリックします。



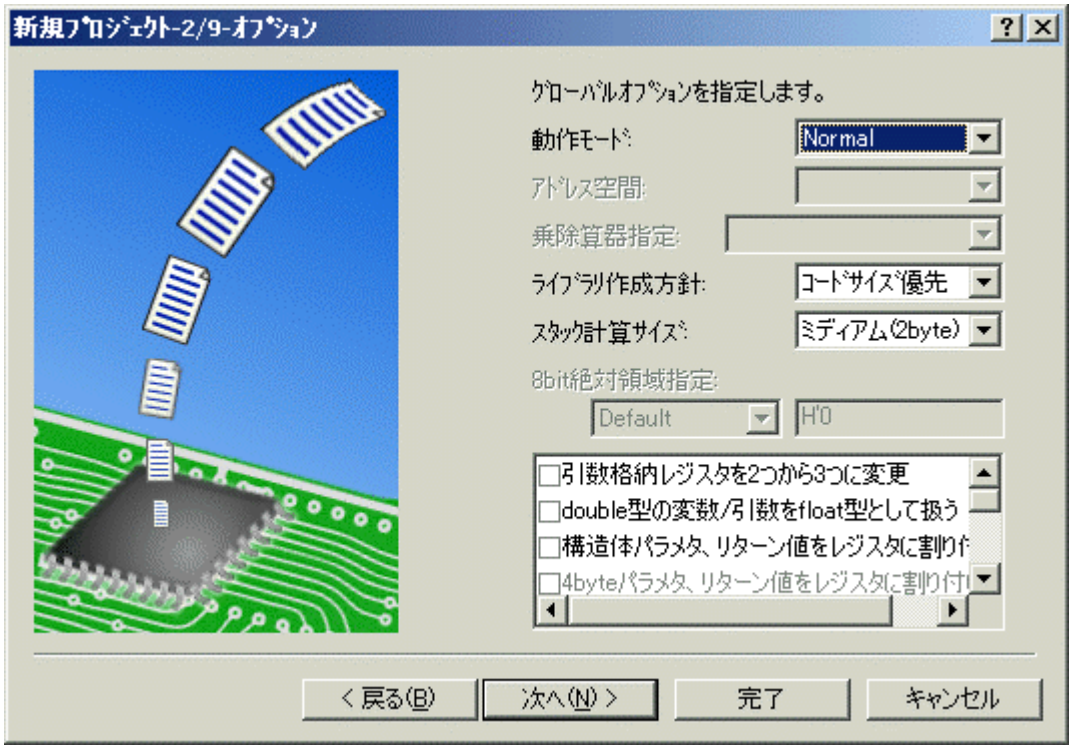
プロジェクトタイプは「Application」を指定します。プロジェクト名は「sample_01」，ディレクトリは「C:¥hew_tk3687¥hosv4¥sample_01」を入力します。また，CPU 種別は「H8S,H8/300」，ツールチェインは「Reneas H8S,H8/300 Standard」を選択します。入力が終わったら「OK」をクリックします。



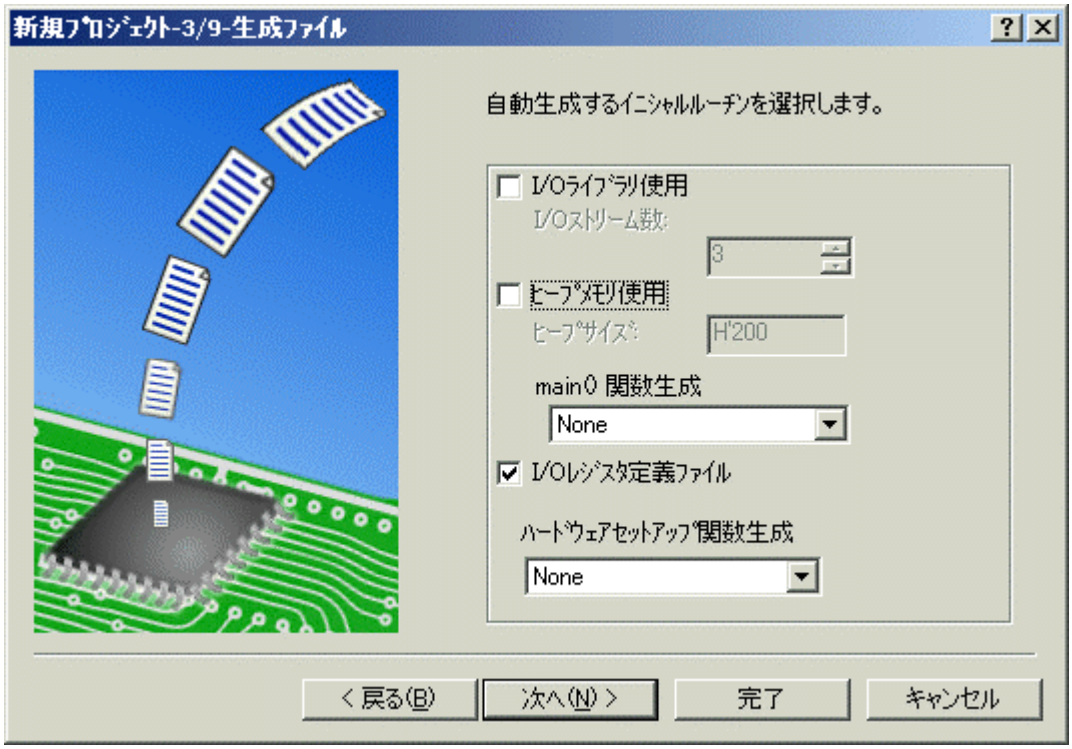
「新規プロジェクト-1/9-CPU」で，CPU シリーズは「300H」，CPU タイプは「3687」を選択して「次へ」をクリックします。



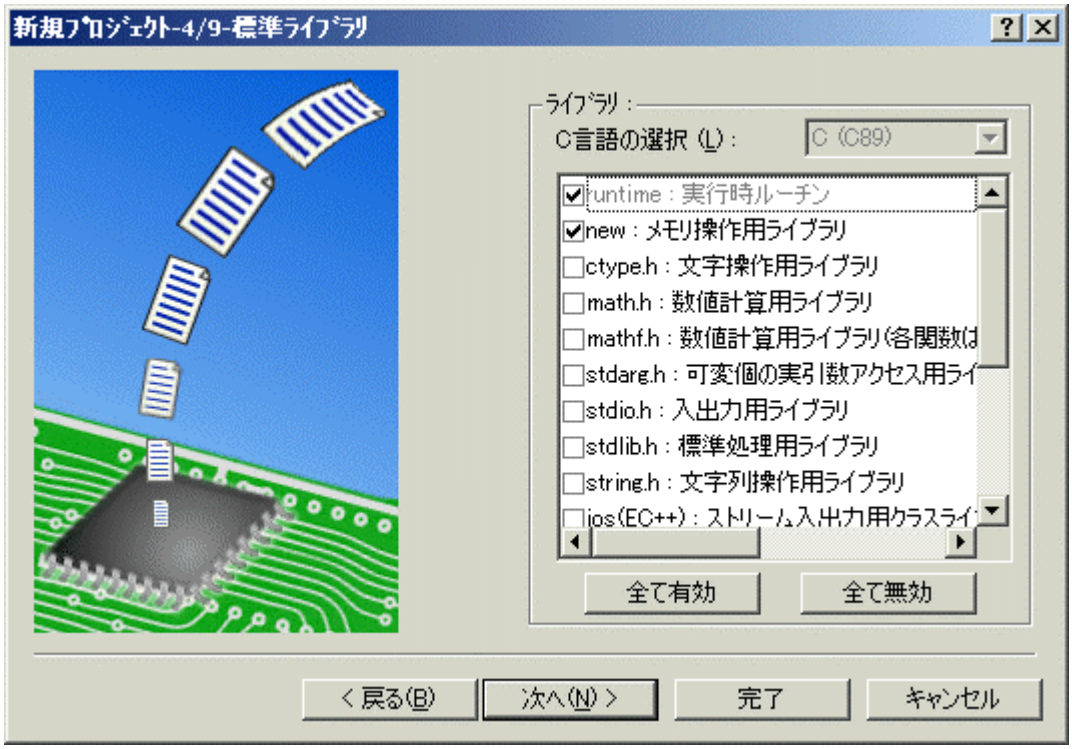
「新規プロジェクト-2/9-オプション」で，動作モードは「Normal」を選択して「次へ」をクリックします。



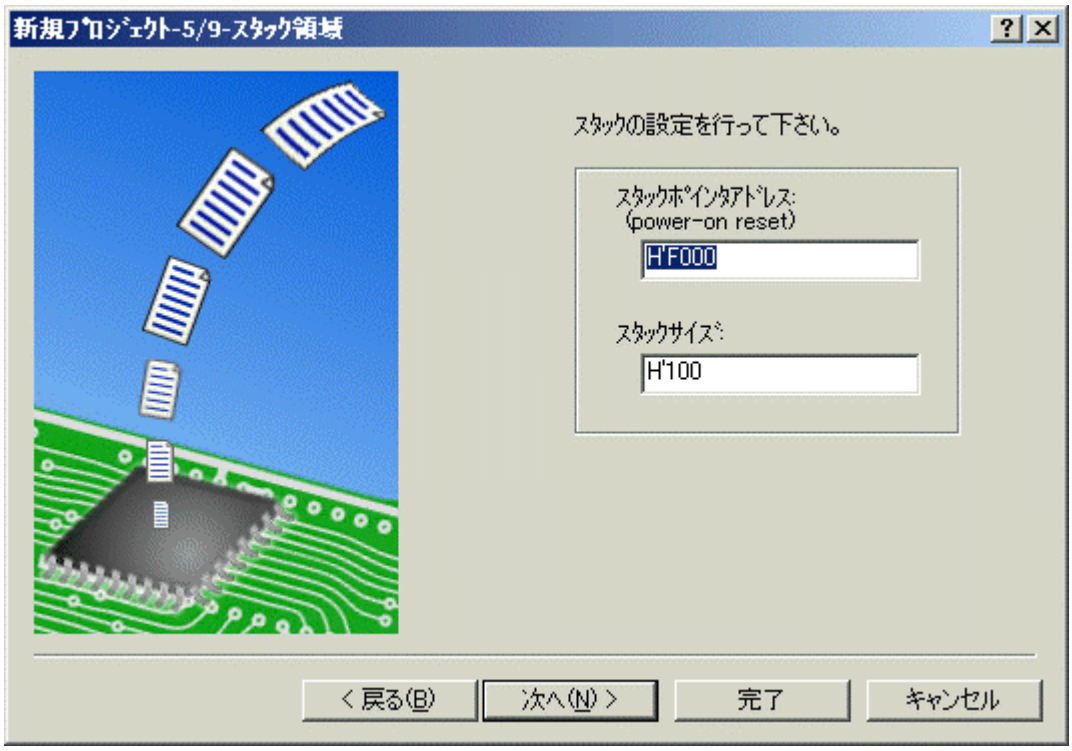
「新規プロジェクト-3/9-生成ファイル」で，「I/O レジスタ定義ファイル」のみチェックし，「I/O ライブラリ使用」と「ヒープメモリ使用」はチェックしません。「main()関数生成」は「None」を選択します。「次へ」をクリックします。



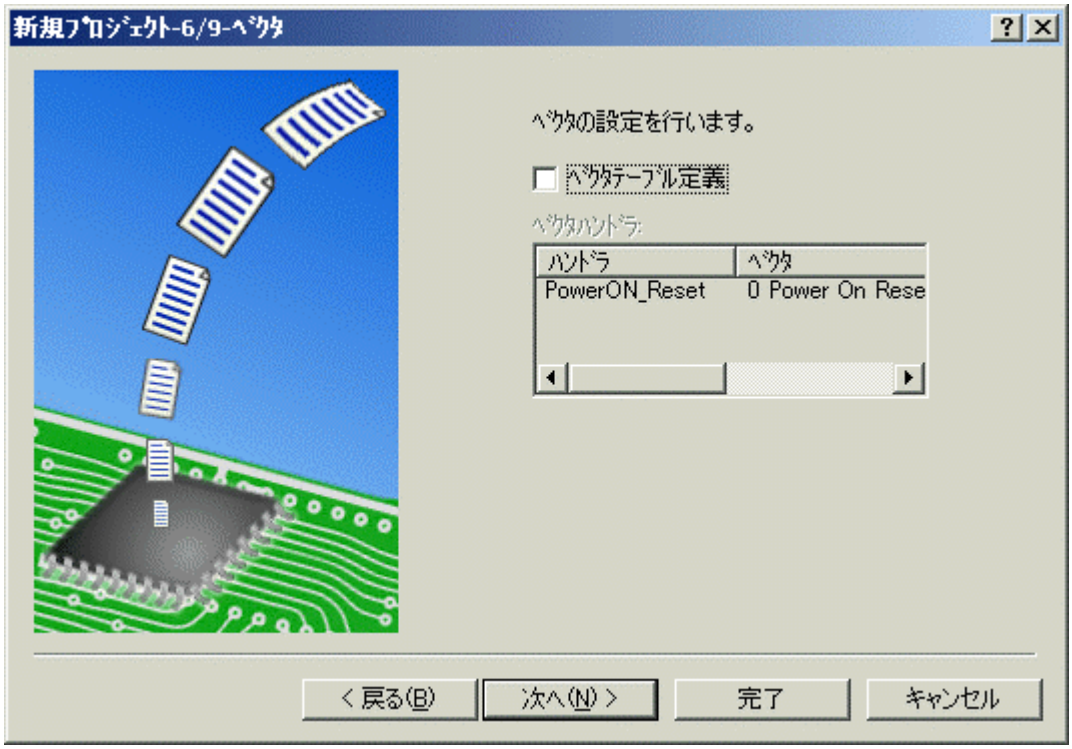
「新規プロジェクト-4/9-標準ライブラリ」は，必要なライブラリにチェックします。変更しなくても OK です。「次へ」をクリックします。



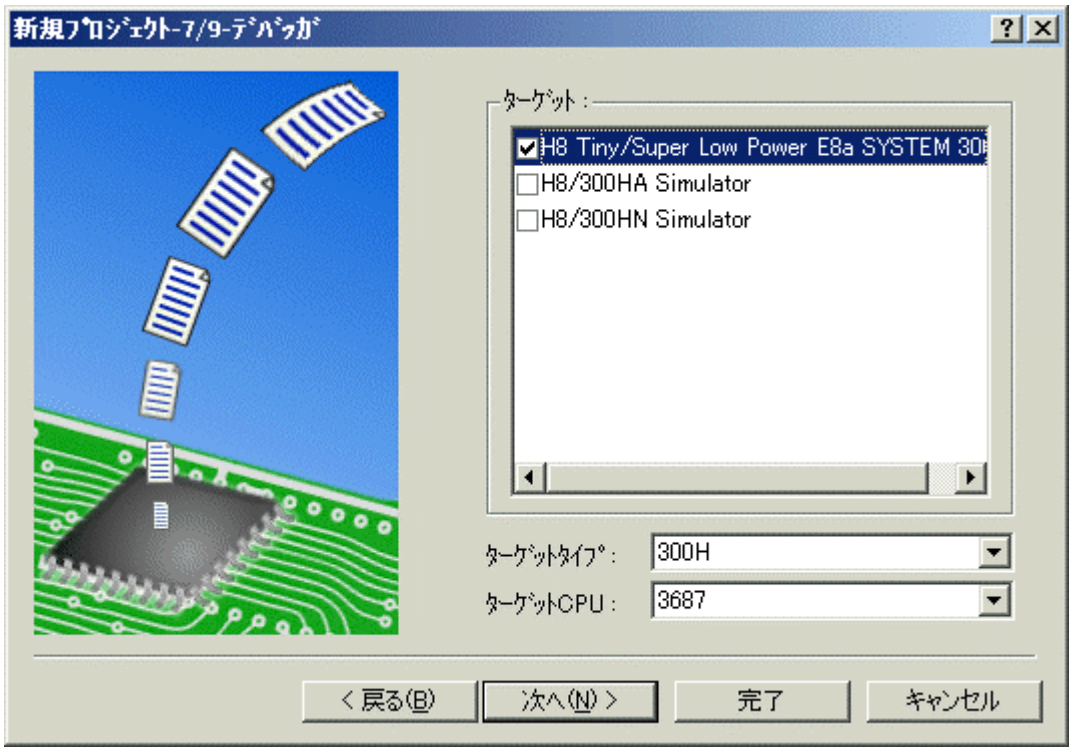
「新規プロジェクト-5/9-スタック領域」は変更しません。スタックの管理は HOS 側で行なうからです。「次へ」をクリックします。



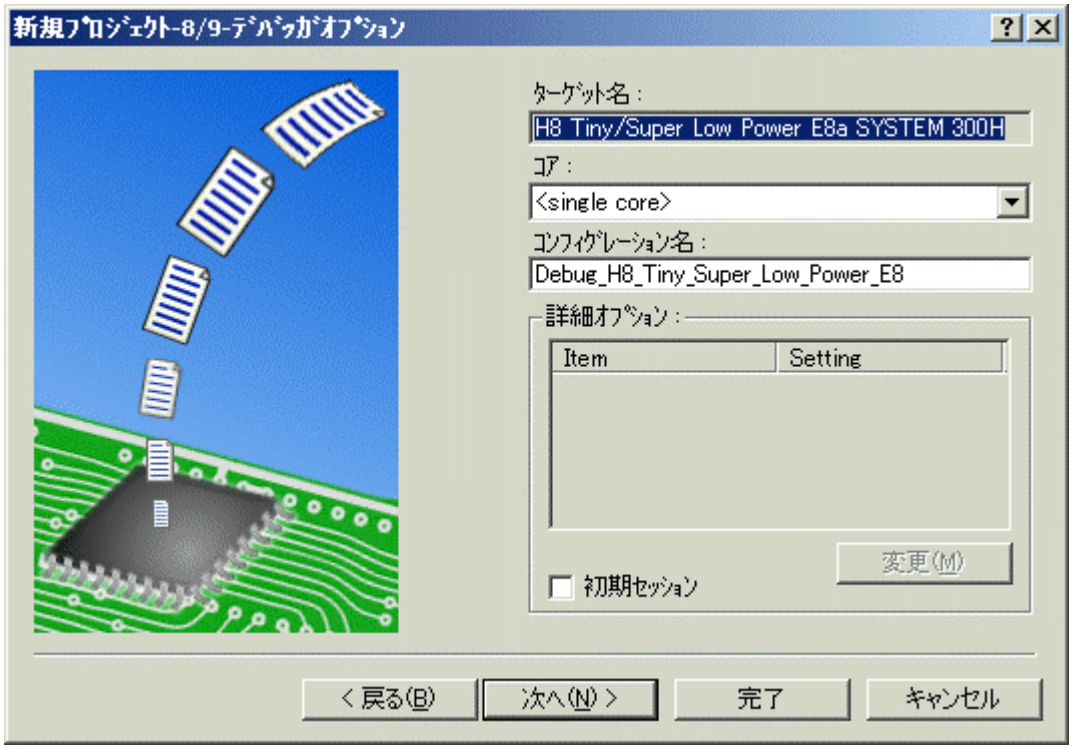
「新規プロジェクト-6/9-ベクタ」で「ベクタテーブル定義」のチェックを外します。これも HOS 側で行なうからです。「次へ」をクリックします。



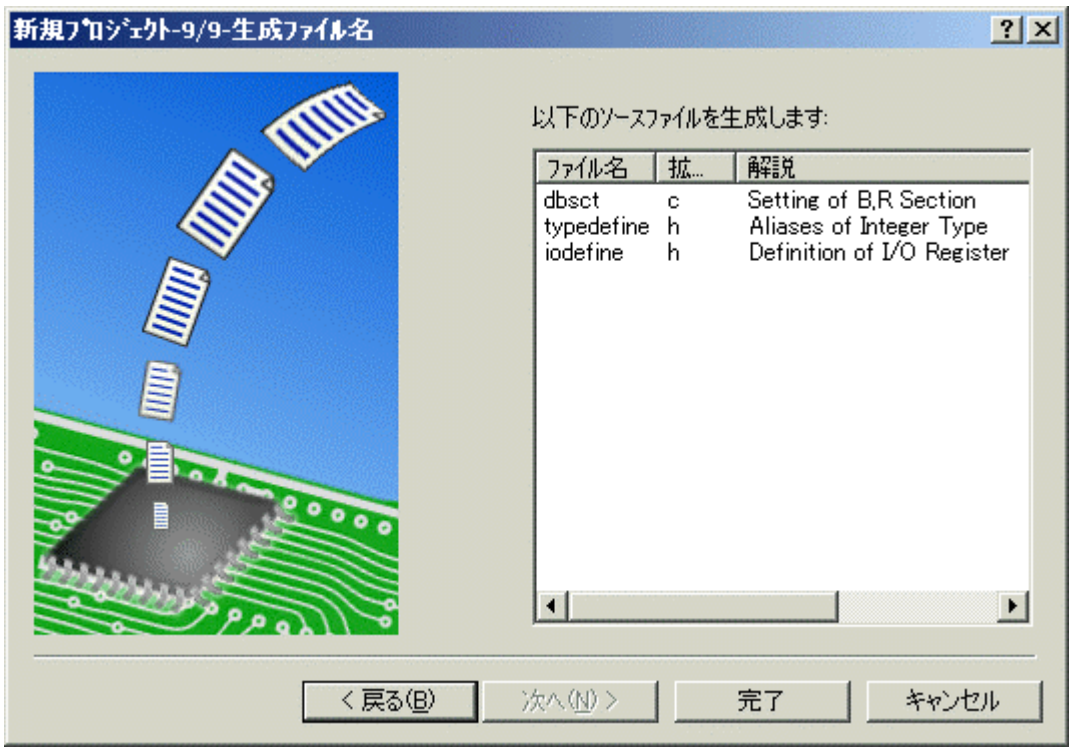
「新規プロジェクト-7/9-デバッガ」で，E8a を使うときは，ターゲットの「H8 Tiny/Super Low Power E8a SYSTEM 300」にチェックを入れます（E8a をインストールしていないと，この選択肢は表示されません）。「次へ」をクリックします。



「新規プロジェクト-8/9-デバッガオプション」は変更しません。「次へ」をクリックします。

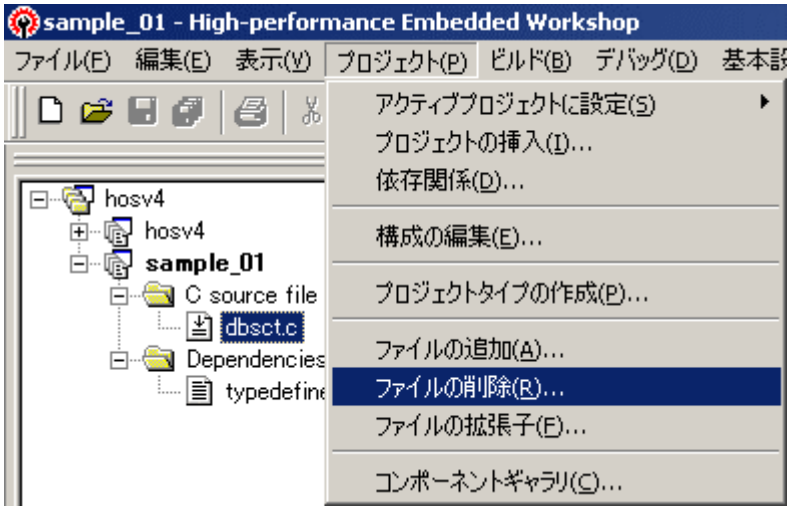


「新規プロジェクト-9/9-生成ファイル名」も変更しません。「完了」をクリックします。

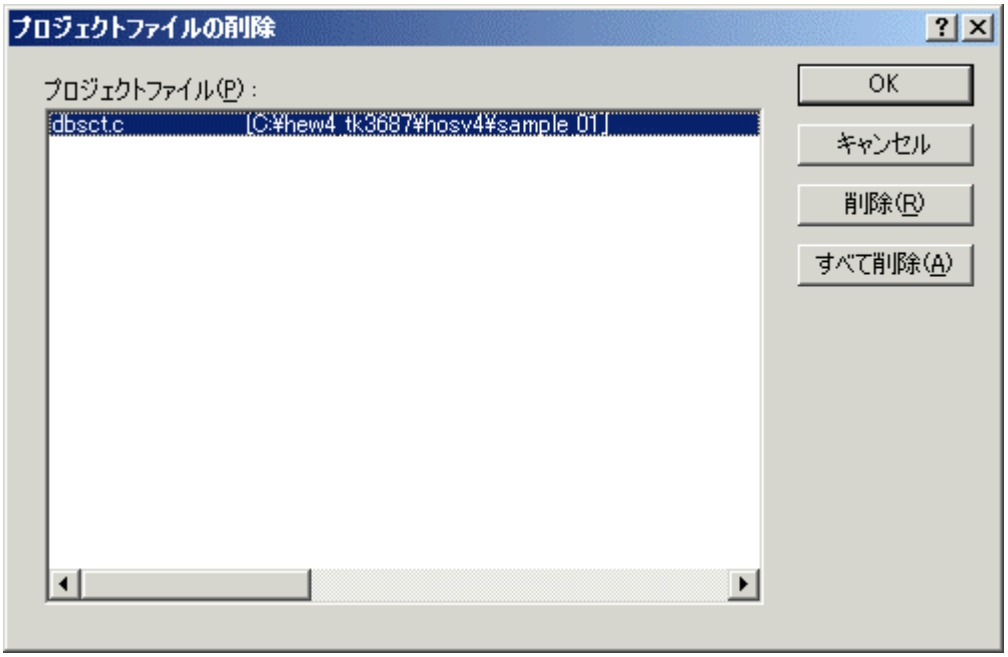


最後に「概要」ダイアログが開きます。「OK」をクリックしてください。これで，プロジェクトができました。

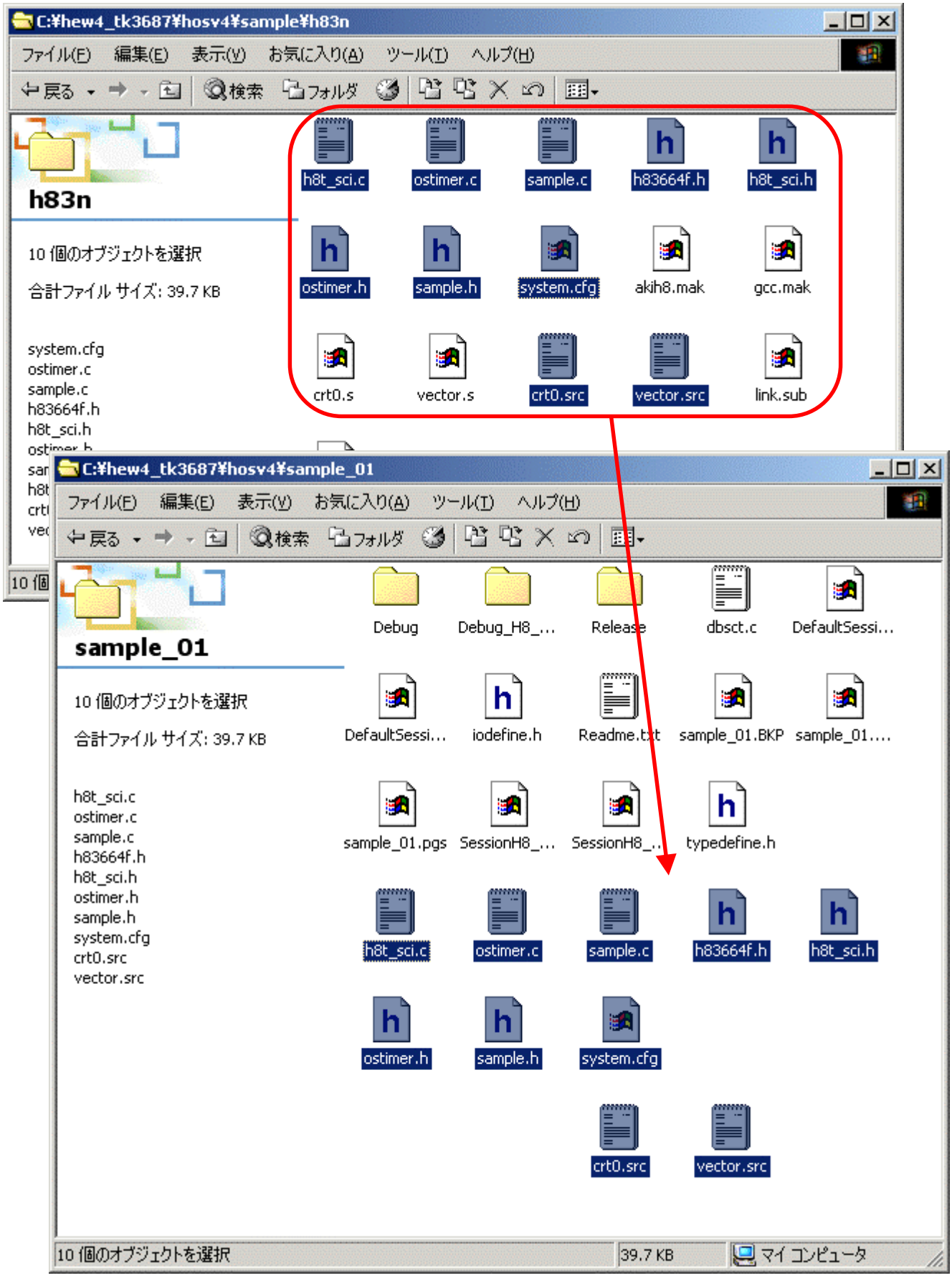
この時点で「iodefine.h」と「dbsct.c」というファイルが生成されます。このうち「dbsct.c」はセクションの初期化をするためのファイルですが，セクションの初期化は HOS の中でも行なっているため，このファイルをプロジェクトから削除します。メニューから「プロジェクト」→「ファイルの削除」を選択します。



「プロジェクトファイルの削除」ダイアログが開きます。「dbsct.c」を選択して「削除」をクリックしてください。削除されたら「OK」をクリックします。



続いて，プロジェクトに必要なファイルを登録します。まず，H8/Tiny 用のサンプルプログラムをコピーします。「C:¥hew4_tk3687¥hosv4¥sample¥h83n¥」フォルダから次の 10 個のファイルを「C:¥hew4_tk3687¥hosv4¥sample_01¥」フォルダにコピーします。

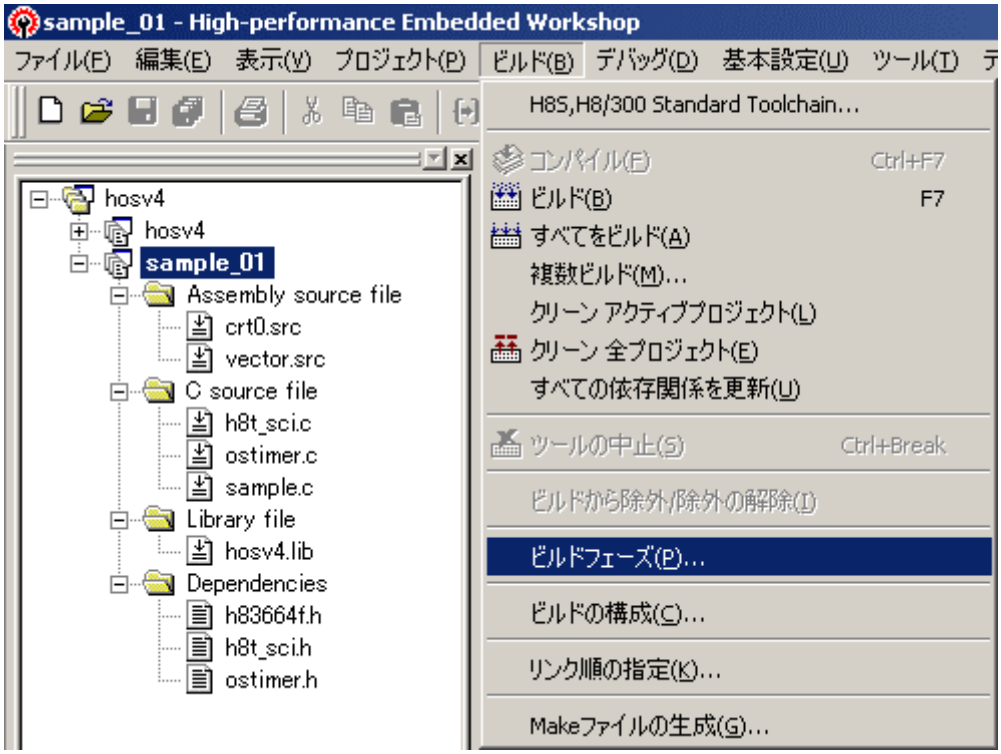


メニューから「プロジェクト」→「ファイルの追加」を選択します。すると，「プロジェクトにファイルを追加」ダイアログが表示されますので，今コピーしたファイルのうち，「crt0.src」，「h8t_sci.c」，「ostimer.c」，「sample.c」，「vector.src」の 5 つのファイルを選択して「追加」をクリックします。「相対パス」にチェックを忘れずに入れてください。

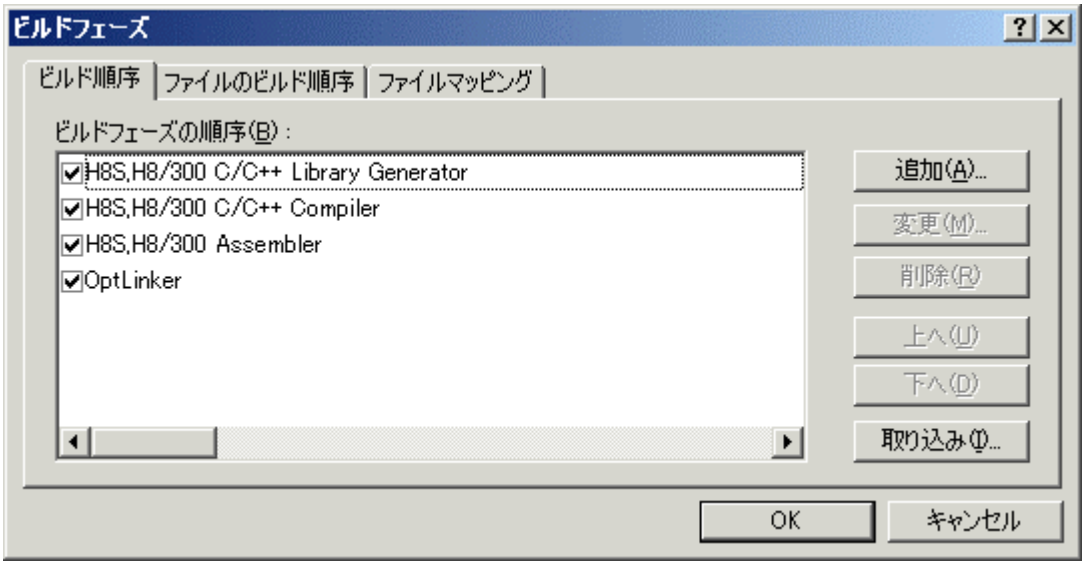
続いて，構築したカーネルライブラリを追加します。メニューから「プロジェクト」→「ファイルの追加」を選択します。すると，「プロジェクトにファイルを追加」ダイアログが表示されますので，「C:¥hew4_tk3687¥hosv4¥hosv4¥Debug¥hosv4.lib」を選択して「追加」をクリックします。「相対パス」にチェックを忘れずに入れてください。

さて，アプリケーションの構築を進めてきましたが，ここでもう一つ登録が必要なファイルがあります。それはシステムコンフィギュレーションファイル「system.cfg」に基づき生成される，カーネルの構成や初期化に必要なファイル「kernel_cfg.c」です。ここで使うツールをコンフィギュレータと呼びます。コンフィギュレータの詳細については μITRON4.0 仕様書の「2.1.10 システムコンフィギュレーションファイル」をご覧ください。ここでは，とにかく動かして生成し先に進みましょう。

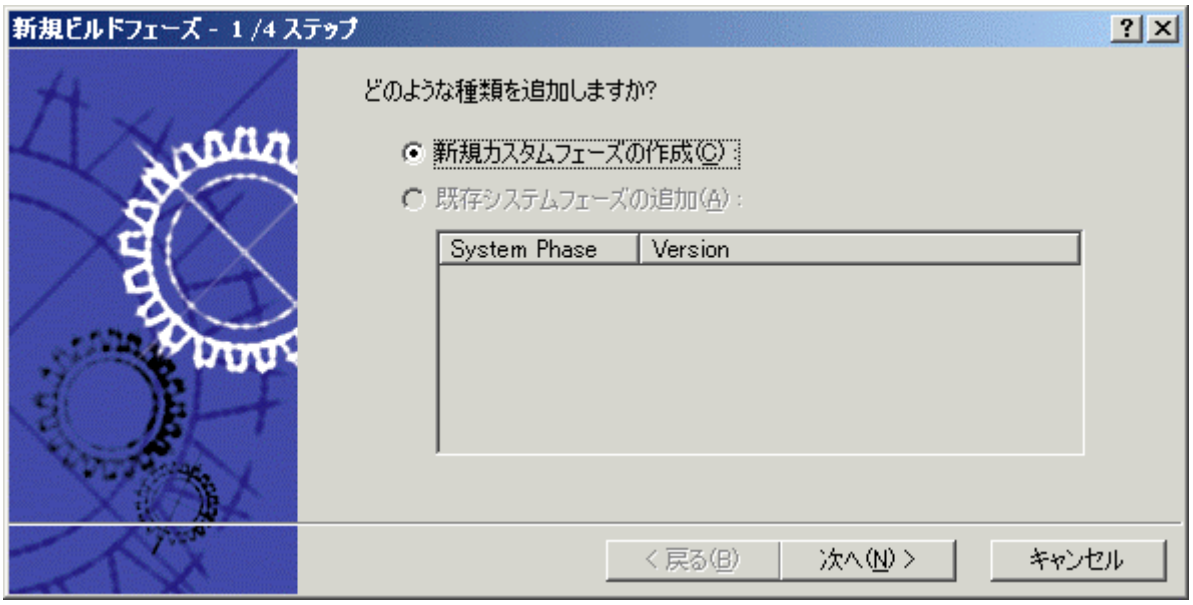
まずはコンフィギュレーションファイル「system.cfg」を HEW の H8 用コンパイラ ch38.exe のプリプロセス機能を使って整形します。そしてこの処理をビルドに組み込んで自動的に行なうようにします。メニューから「ビルド」→「ビルドフェーズ」を選択します。



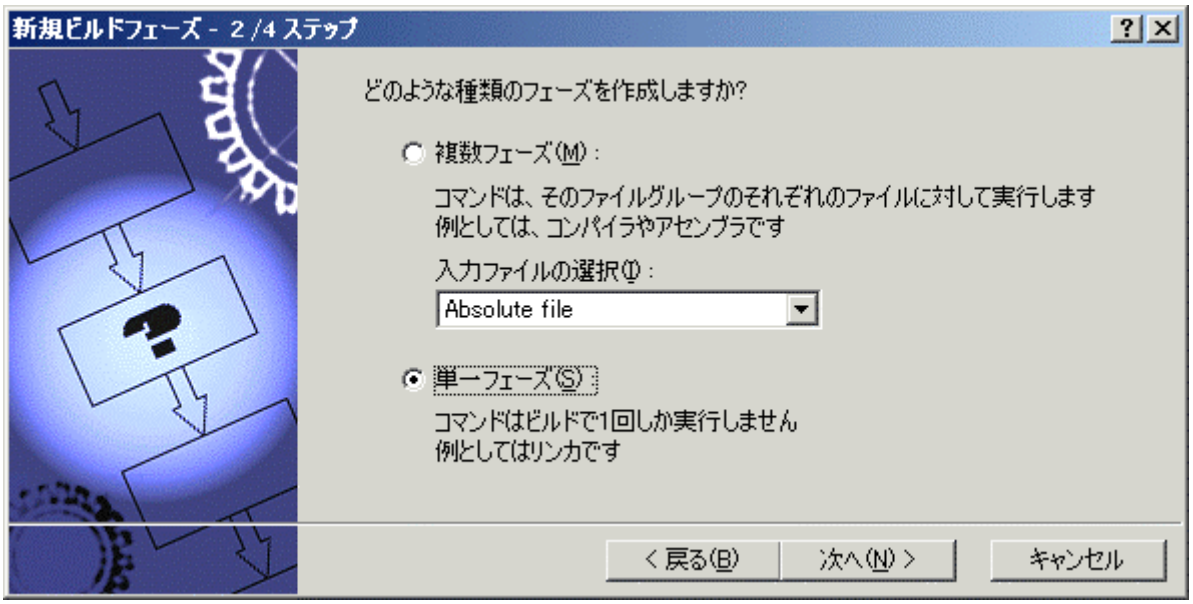
すると，ビルドフェーズダイアログが開きますので，「追加」をクリックします。



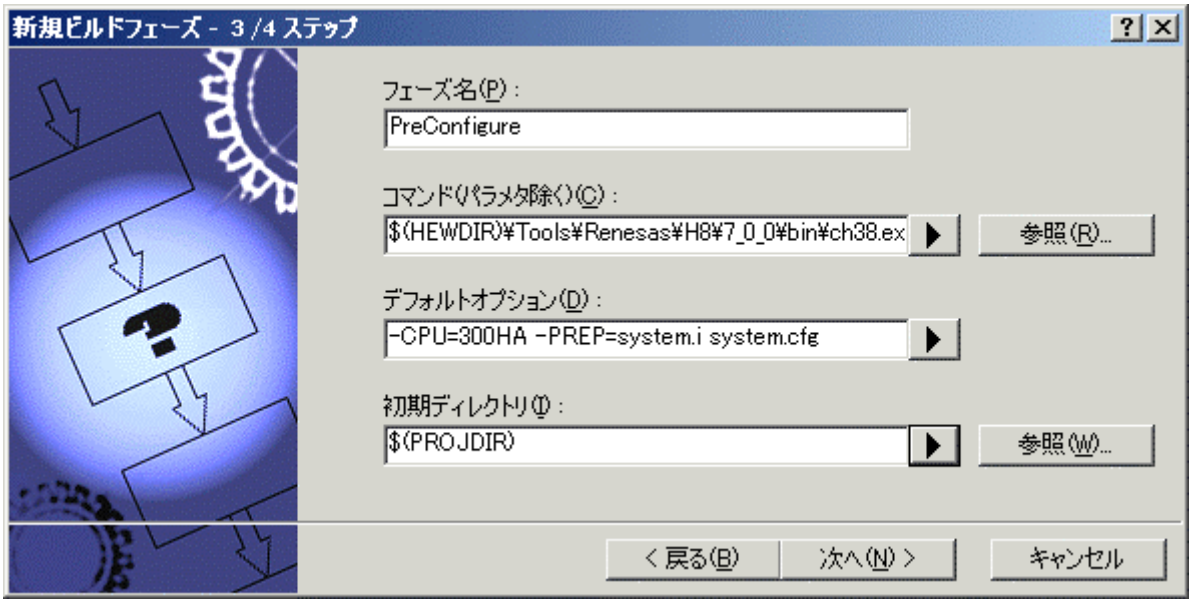
「新規ビルドフェーズ － 1/4 ステップ」で「新規カスタムフェーズの作成」をチェックします。「次へ」をクリックします。



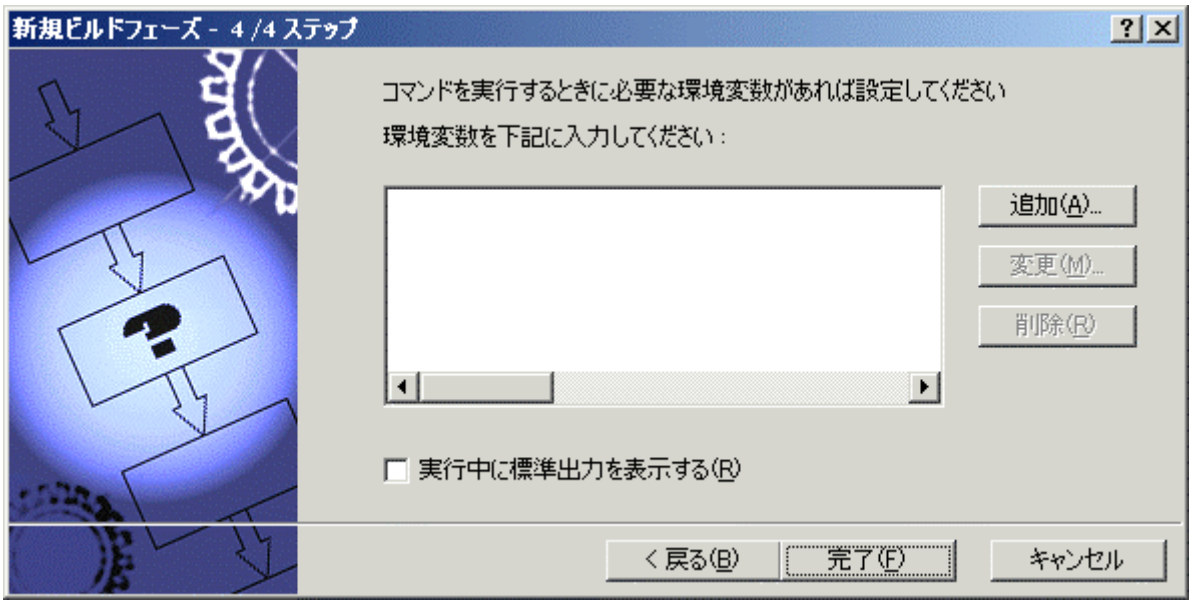
「新規ビルドフェーズ － 2/4 ステップ」で「単一フェーズ」をチェックします。「次へ」をクリックします。



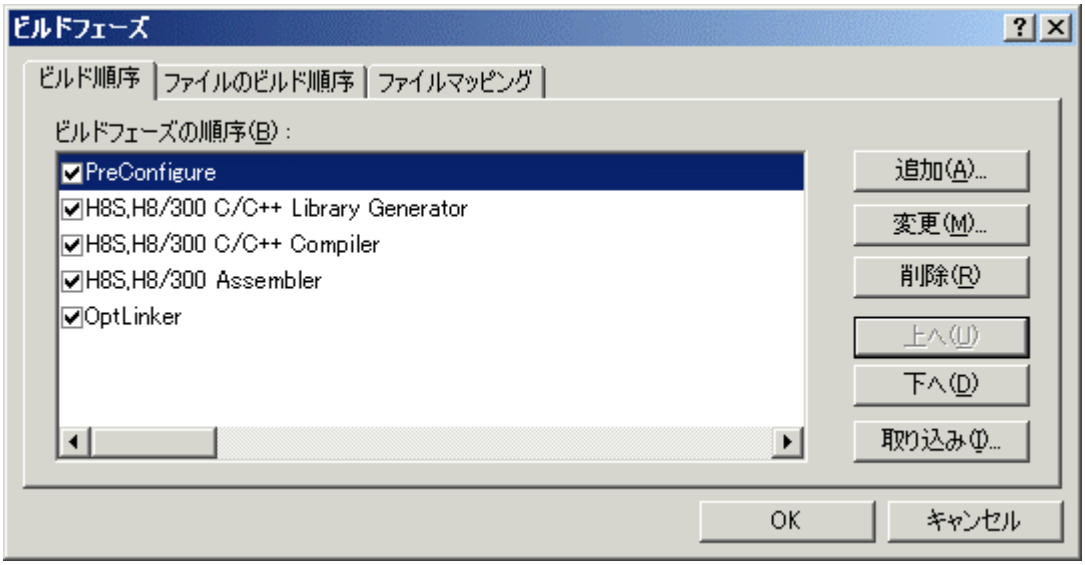
「新規ビルドフェーズ － 3/4 ステップ」でフェーズ名は「PreConfigure」，コマンドパラメータは「$(HEWDIR)¥Tools¥Renesas¥H8¥7_0_0¥bin¥ch38.exe」，デフォルトオプションは「-CPU=300HA -PREP=system.i system.cfg」，初期ディレクトリは「$(PROJDIR)」を入力します。「次へ」をクリックします。



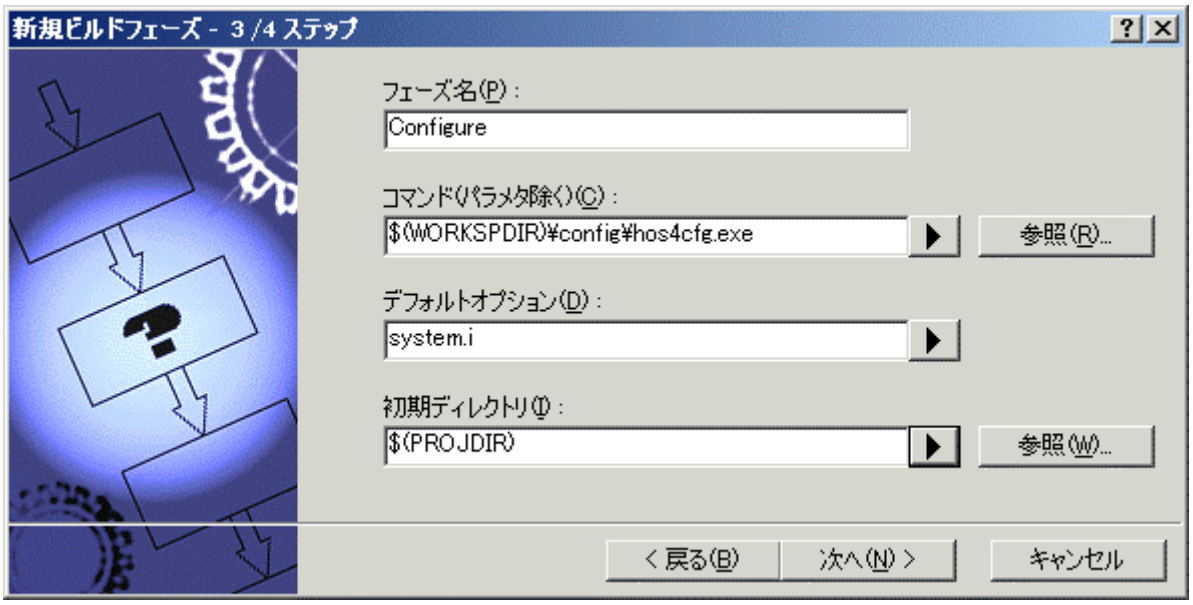
「新規ビルドフェーズ － 4/4 ステップ」の変更はありません。「完了」をクリックします。



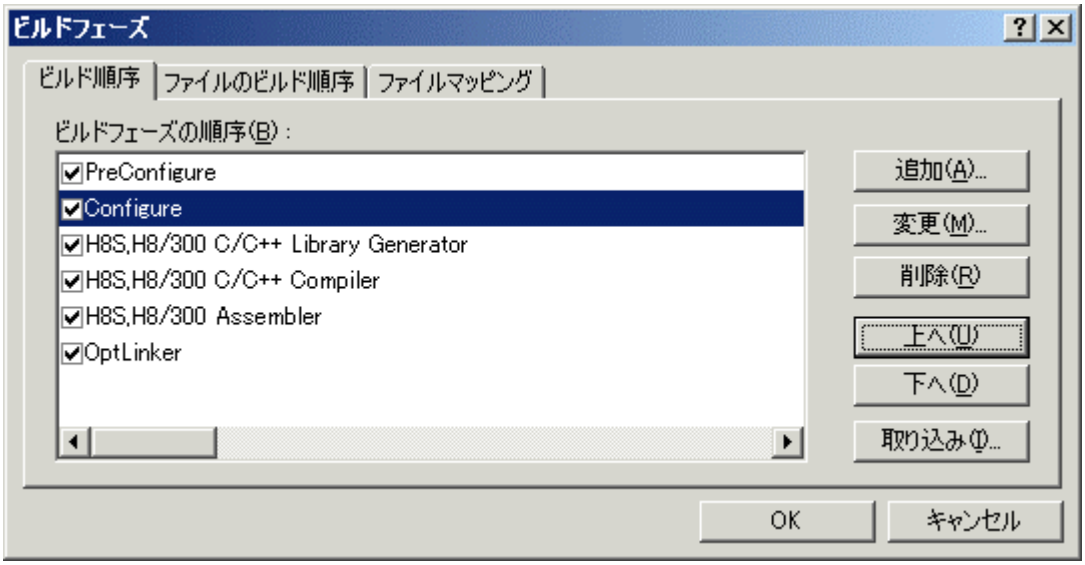
ビルドフェーズダイアログに戻ります。今作った「PreConfigure」を選択し，「上へ」をクリックして先頭に移動します。



次に，整形した中間ファイルを，HOS に付属しているコンフィギュレータ hos4cfg.exe を使ってオブジェクトファイル「kernel_cfg.c」と，ID 番号などの定義ファイル「kernel_id.h」を生成します。やはりこの処理をビルドに組み込んで自動的に行なうようにします。ビルドフェーズダイアログの「追加」をクリックします。「新規ビルドフェーズ － 1/4 ステップ」と「新規ビルドフェーズ － 2/4 ステップ」は同じです。「新規ビルドフェーズ － 3/4 ステップ」でフェーズ名は「Configure」，コマンドパラメータは「$(WORKDIR)¥config¥hos4cfg.exe」，デフォルトオプションは「system.i」，初期ディレクトリは「$(PROJDIR)」を入力します。「次へ」をクリックします。

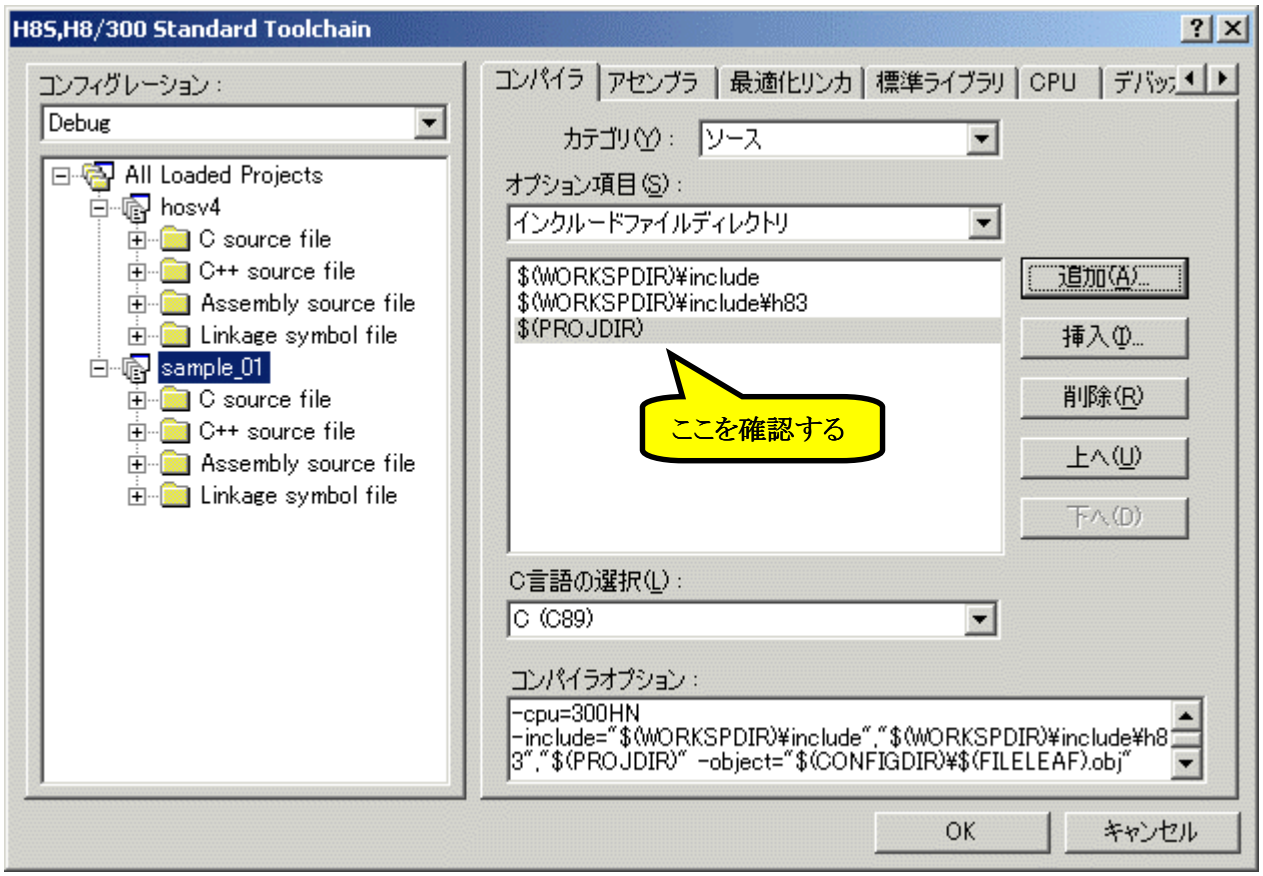


「新規ビルドフェーズ － 4/4 ステップ」の変更はありません。「完了」をクリックします。すると，ビルドフェーズダイアログに戻ります。今作った「Configure」を選択し，「PreConfigure」の次に移動します。

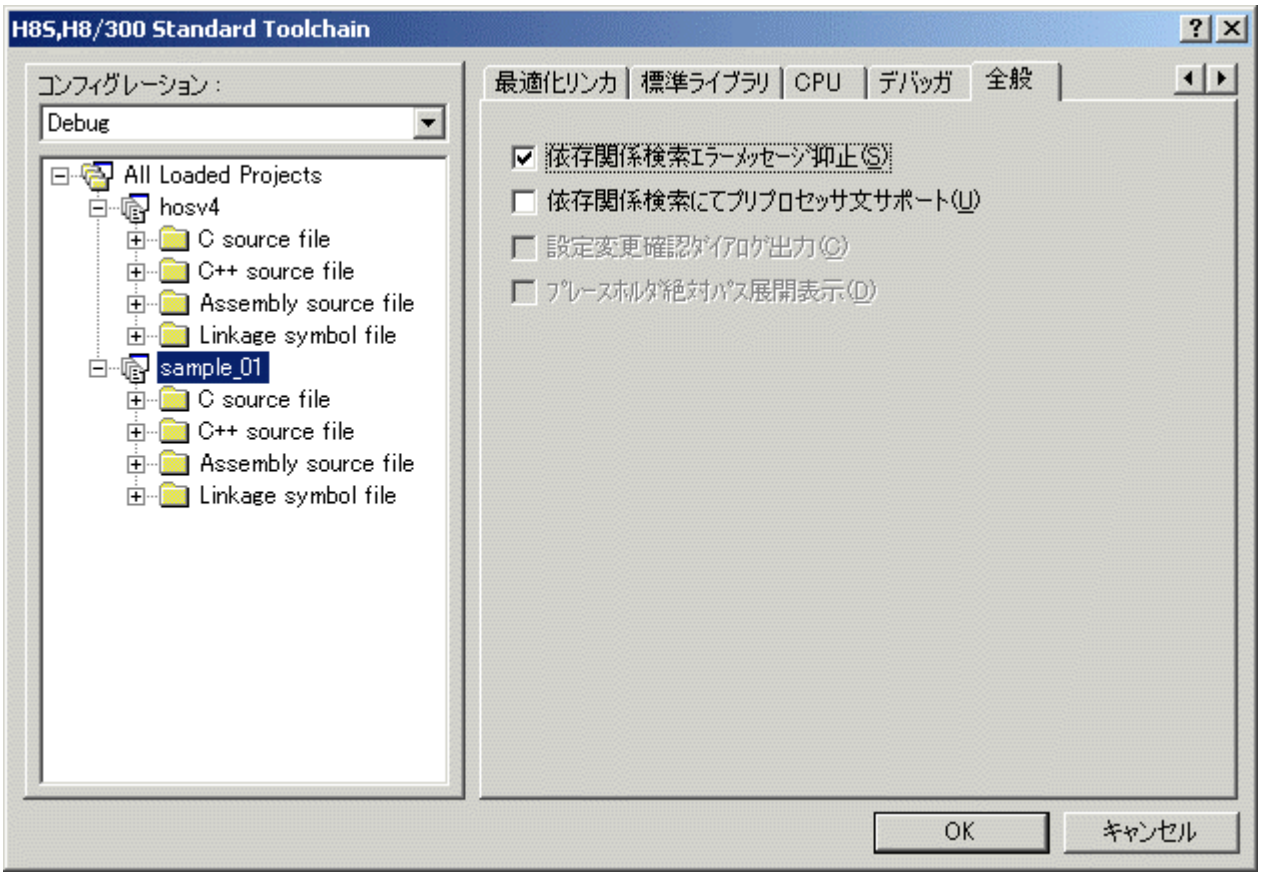


この段階で一度ビルドしてみましょう。Warning ができますが気にしないで大丈夫です。「kernel_cfg.c」と「kernel_id.h」が生成されます。このうち，「kernel_cfg.c」をプロジェクトに登録します。メニューから「プロジェクト」→「ファイルの追加」を選択します。すると，「プロジェクトにファイルを追加」ダイアログが表示されますので，「kernel_cfg.c」を選択して「追加」をクリックします。「相対パス」にチェックを忘れずに入れてください。

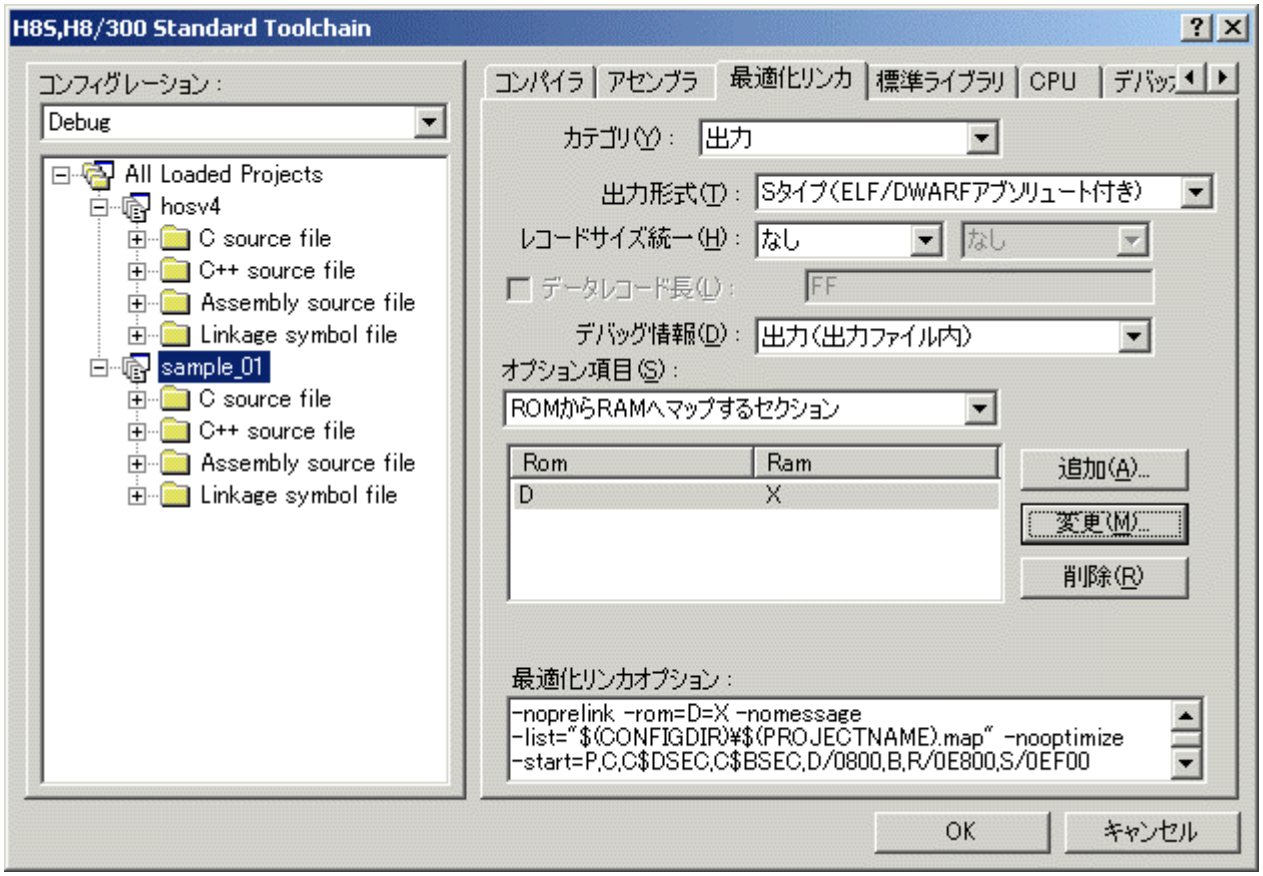
次に，インクルードファイルのパスを指定します。メニューから「ビルド」→「H8S,H8/300 Standard Toolchain...」を選択します。ダイアログが開きますので，「コンパイラ」タブを開き，「ソース」カテゴリのオプション項目「インクルードファイルディレクトリ」を選択して「追加」をクリックします。ダイアログが開きますので，「相対パス」は「Workspace directory」を選択し，「サブディレクトリ」は「include」を指定して「OK」をクリックします。もう一度「追加」をクリックしてダイアログを開き，「相対パス」は「Workspace directory」を選択し，「サブディレクトリ」は「include¥h83」を指定して「OK」をクリックします。さらに「追加」をクリックしてダイアログを開き，「相対パス」は「Project directory」を選択して「OK」をクリックします。



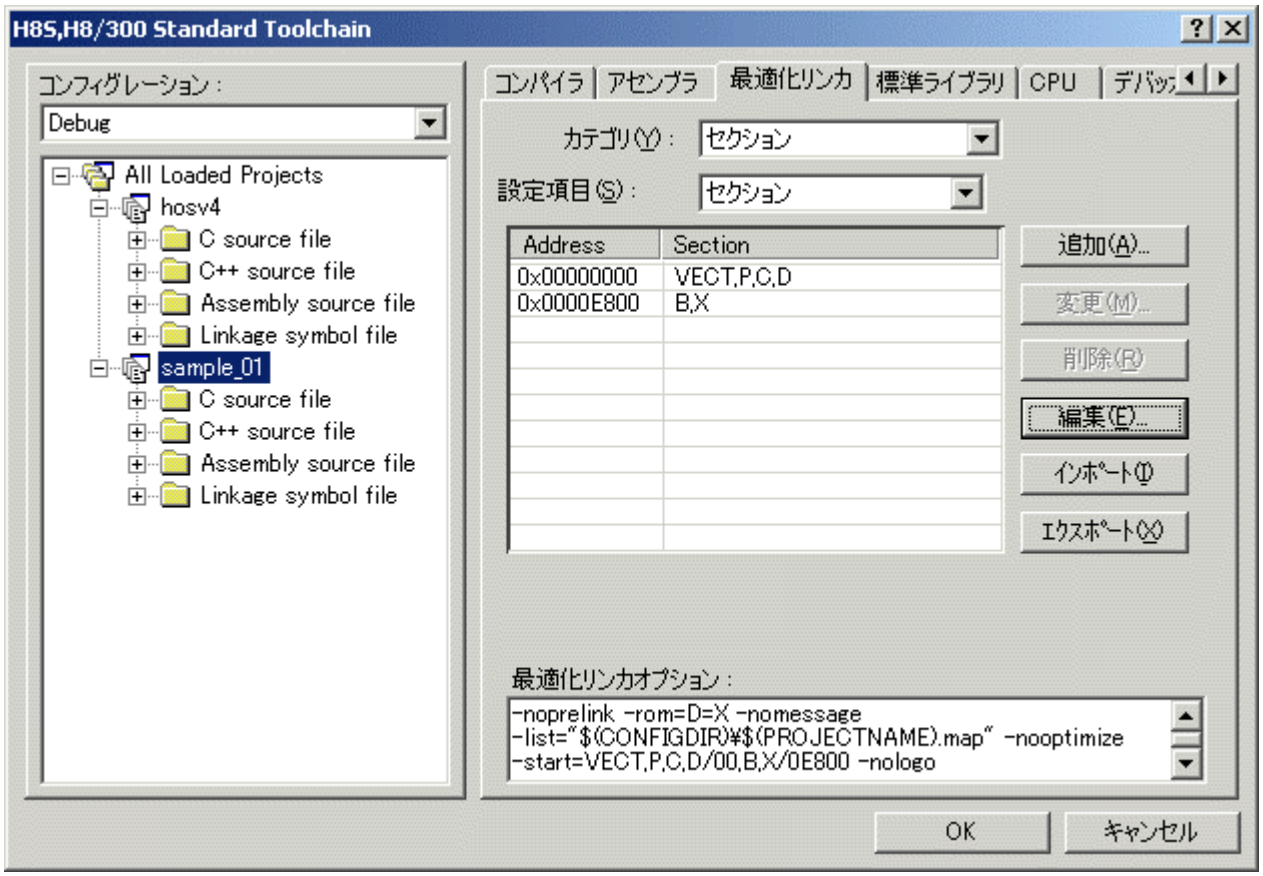
以前と同じように，このままビルドすると大量の「Warning」が発生します。メニューから「ビルド」→「H8S,H8/300 Standard Toolchain...」を選択してください。「全般」タブの「依存関係検索エラーメッセージ抑止」にチェックを入れます。



標準では ROM から RAM へマップするセクションは D セクションと R セクションですが, HOS の場合は D セクションと X セクションです。この設定を修正します。メニューから「ビルド」→「H8S,H8/300 Standard Toolchain...」を選択してください。「最適化リンカ」タブの「出力」カテゴリを選び, オプション項目のリストから「ROM から RAM へマップするセクション」を選択します。「変更」をクリックして R を X に変更してください。

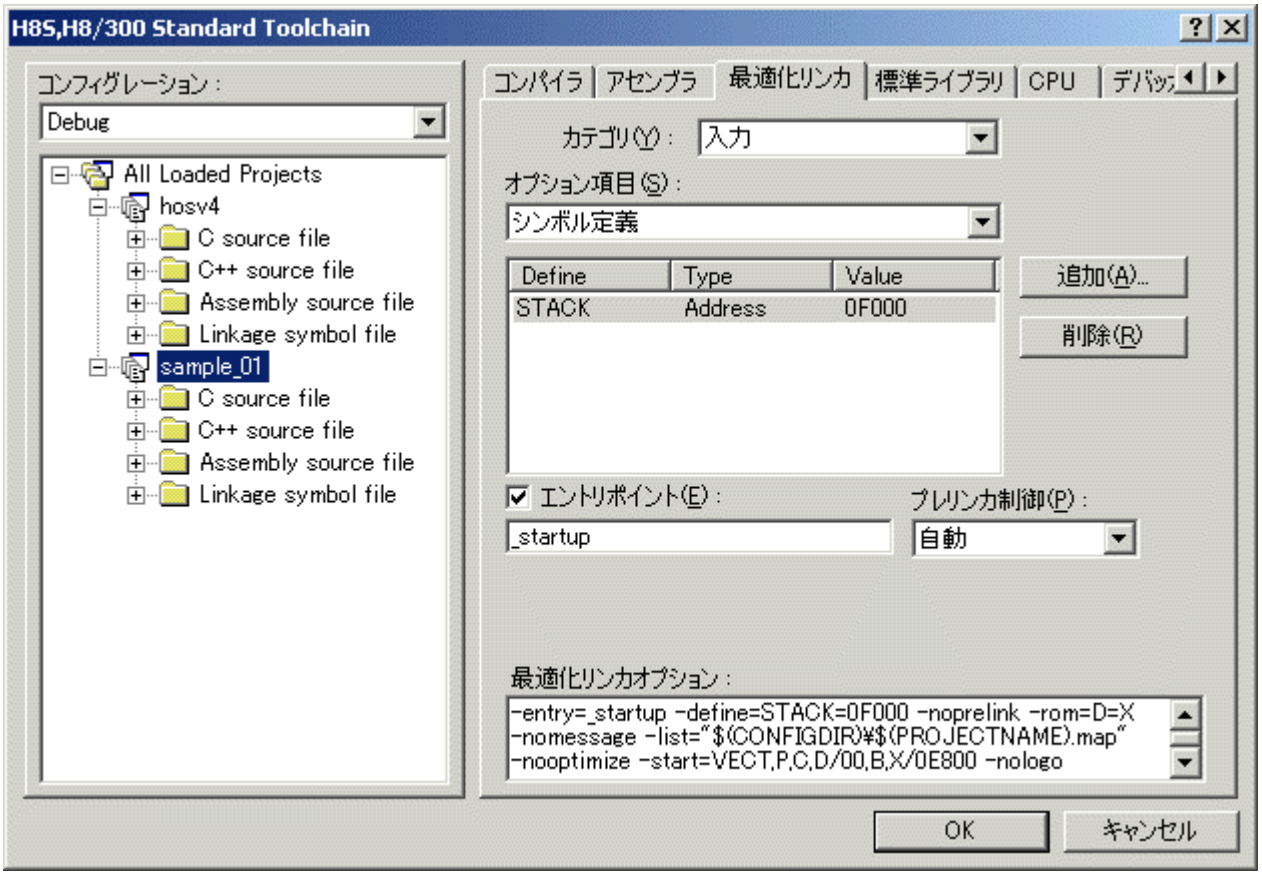


続いてセクションを指定します。メニューから「ビルド」→「H8S,H8/300 Standard Toolchain...」を選択してください。「最適化リンカ」タブの「セクション」カテゴリを選び，設定項目のリストから「セクション」を選択します。「編集」をクリックして次のように設定します。



STACK を設定します。メニューから「ビルド」→「H8S,H8/300 Standard Toolchain...」を選択してください。「最適化リンカ」タブの「入力」カテゴリを選び，オプション項目のリストから「シンボル定義」を選択します。「追加」をクリックして「STACK」を設定してください。

さらに，プログラムの開始アドレスは「_startup」なので，「エントリポイント」にチェックを入れて入力します。



これで，環境が整いました。ビルドしてください。「C:¥hew4_tk3687¥hosv4¥sample_01¥Debug¥」フォルダに「sample_01.mot」ができていれば，まずは OK です。

4. TK-3687 版にカスタマイズする

ビルドも終わりmotファイルが生成されましたが，あくまでこれはプロジェクトHOSが提供しているH8/3664 用のサンプルプログラムです。これを TK-3687/TK-3687mini 版(H8/3687)にカスタマイズします。主な変更点は，CPU:H8/3664→H8/3687，クロック:16MHz→20MHz，の 2 点です。

まず OS タイマをカスタマイズしましょう。オリジナルは H8/3664 のタイマ A を使っていますが，H8/3687 はタイマ A のかわりに RTC が搭載されているので，RTC をフリーランカウンタモードで使用するよう変更します。「ostimer.c」を次のように修正します。

```
/* ------------------------------------------------------------------------ */
/*  Hyper Operating System V4  サンプルプログラム                           */
/*    H8/3664 用 OSタイマ (Timer Aを利用)                                   */
/*                                                                          */
/*                                  Copyright (C) 1998-2002 by Project HOS  */
/*                                  http://sourceforge.jp/projects/hos/     */
/* ------------------------------------------------------------------------ */
/*    H8/3687 用 OSタイマ (RTCを利用)                                       */
/*                                                  by TOYO-LINX Co.,Ltd.*/
/*                                                  ne.jp/~toyolinx    */
/* ------------------------------------------------------------------------ */
#include "kernel.h"
#include "kernel_id.h"
//#include "h83664f.h"
#include "iodefine.h"
#include "ostimer.h"

#include "binary.h"          //Cで2進数表示を使う(00000000-11111111)



/* OSタイマの初期化 */
void OsTimer_Initialize(VP_INT exinf)
{
    /* TMA初期化
       16MHzで8.192msecのインターバルタイマとして設定 */
//  IO.PMR1.BYTE = 0x02;
//  IO.PCR1 = 0xff;
//  TA.TMA.BYTE =  0x03;    /* 00000011 */
//  IENR1.BIT.IENTA = 1;    /* タイマ割り込み許可 */

    /* RTC初期化
       20MHzで6.5536msecのインターバルタイマとして設定 */
    IO.PMR1.BYTE      =   _00000010B;   // TMOWディセーブル,TxDイネーブル
    RTC.RTCCR1.BYTE   =   _00010000B;   // STOP,RESET
    RTC.RTCCR1.BYTE   =   _00000000B;   // STOP,RESET解除
    RTC.RTCCSR.BYTE   =   _00000100B;   // CLK/512(フリーランカウンタとして使用)
    RTC.RTCCR2.BYTE   =   _00100000B;   // オーバーフロー割り込みイネーブル
    IENR1.BIT.IENTA   =   1;            // RTC割り込みイネーブル
    RTC.RTCCR1.BYTE   =   _10000000B;   // RUN
}

/* タイマ用割り込みハンドラ */
void OsTimer_TimerHandler(VP_INT exinf)
```

I/O 定義ファイルを，HEW が生成したものに変更する。

C で 2 進数表現を使えるように，東洋リンクス作成の定義ファイルをインクルードする。

H8/3664 用のプログラムをコメント化する。

H8/3687 用プログラムの追加。

```
{
    /* 割り込み要因クリア */
    IRR1.BYTE &= 0xbf; /* 10111111 */

    /* タイムティック供給 */
    isig_tim();
}

/* ------------------------------------------------------------------------ */
/*  Copyright (C) 1998-2002 by Project HOS                                  */
/*  Copyright (C) 2009      by TOYO-LINX Co.,Ltd.                           */
/* ------------------------------------------------------------------------ */
```

これで，6.5536ms 周期で割り込みが発生します。あとは HOS に対して何 ms 毎に割り込みが発生するか定義します。その定義が「system.cfg」にあります。次のように変更します。なお，HEW のエディタでは開けないので，メモ帳などのエディタソフトで修正してください。

```
//HOS_TIM_TIC(1024, 125);       /* タイムティックの設定(省略時 1/1 ) */
HOS_TIM_TIC(4096, 625);         /* タイムティックの設定(省略時 1/1 ) */
```

この定義の意味は「割り込みのたびに 4096/625(=6.5536)を積算する」というものです。

続いてシリアルポートの定義を調整します。H8/3664 と H8/3687 は同じ SCI を搭載しているのでプログラム自体はそのままでかまわないのですが，TK-3687/TK-3687mini 実装のクロックが異なるのでボーレート用の定義を変更します。ファイルは「h8t_sci.h」です。次のように修正してください。

```
/* SCI通信速度定義 (16MHzのとき) */
//#define SCI_2400    207
//#define SCI_4800    103
//#define SCI_9600     51
//#define SCI_19200    25
//#define SCI_31250    15
//#define SCI_38400    12
//#define SCI_57600     8          /* 誤差 -3.5% ぎりぎりいけるかも (^^;; */

/* SCI通信速度定義 (20MHzのとき) */
#define SCI_4800    129
#define SCI_9600     64
#define SCI_19200    32
#define SCI_31250    19
#define SCI_38400    15
#define SCI_57600    10
```

16MHz 用の定義をコメント化。

20MHz 用の定義を追加する。

続いて割り込みベクタに関する修正です。一部，カーネルライブラリ構築の際に修正していますが，残りの修正を行います。。H8/3664 はベクタ番号が 0～25 までですが，H8/3687 は 0～32 までです。それに伴い，「vector.src」を次のように修正します。

```
; ------------------------------------------------------------------------
;  Hyper Operating System V4  サンプルプログラム
;    割り込みベクターテーブル
;
```

```
;                                   Copyright (C) 1998-2002 by Project HOS
;                                   http://sourceforge.jp/projects/hos/
; ------------------------------------------------------------------------
;                                   Copyright (C) 2009 by TOYO-LINX Co.,Ltd.
;                                   http://www2.u-netsurf.ne.jp/~toyolinx
; ------------------------------------------------------------------------

                                .CPU            300HN

                        .IMPORT         _startup
                        .IMPORT         _hos_vector007
                        .IMPORT         _hos_vector008
                        .IMPORT         _hos_vector009
                        .IMPORT         _hos_vector010
                        .IMPORT         _hos_vector011
                        .IMPORT         _hos_vector012
                        .IMPORT         _hos_vector013
                        .IMPORT         _hos_vector014
                        .IMPORT         _hos_vector015
                        .IMPORT         _hos_vector016
                        .IMPORT         _hos_vector017
                        .IMPORT         _hos_vector018
                        .IMPORT         _hos_vector019
                        .IMPORT         _hos_vector021
                        .IMPORT         _hos_vector022
                        .IMPORT         _hos_vector023
                        .IMPORT         _hos_vector024
                        .IMPORT         _hos_vector025
                        .IMPORT         _hos_vector026
                        .IMPORT         _hos_vector027
                        .IMPORT         _hos_vector028
                        .IMPORT         _hos_vector029
                        .IMPORT         _hos_vector030
                        .IMPORT         _hos_vector031
                        .IMPORT         _hos_vector032

                        .SECTION  VECT,DATA,ALIGN=2

; ------------------------------------------
;           割り込みベクタテーブル
; ------------------------------------------
                        .DATA.W   _startup
                                  .DATA.W h'ffff
                                  .DATA.W h'ffff
                                  .DATA.W h'ffff
                                  .DATA.W h'ffff
                                  .DATA.W h'ffff
                                  .DATA.W h'ffff
                        .DATA.W   _hos_vector007
                        .DATA.W   _hos_vector008
                        .DATA.W   _hos_vector009
                        .DATA.W   _hos_vector010
                        .DATA.W   _hos_vector011
                        .DATA.W   _hos_vector012
                        .DATA.W   _hos_vector013
```

追加。

```
                .DATA.W   _hos_vector014
                .DATA.W   _hos_vector015
                .DATA.W   _hos_vector016
                .DATA.W   _hos_vector017
                .DATA.W   _hos_vector018
                .DATA.W   _hos_vector019
                          .DATA.W h'ffff
                .DATA.W   _hos_vector021
                .DATA.W   _hos_vector022
                .DATA.W   _hos_vector023
                .DATA.W   _hos_vector024
                .DATA.W   _hos_vector025
                .DATA.W   _hos_vector026
                .DATA.W   _hos_vector027
                .DATA.W   _hos_vector028
                .DATA.W   _hos_vector029
                .DATA.W   _hos_vector030
                .DATA.W   _hos_vector031
                .DATA.W   _hos_vector032

                .END

; ------------------------------------------------------------------------
;  Copyright (C) 1998-2002 by Project HOS
;  Copyright (C) 2009      by TOYO-LINX Co.,Ltd.
; ------------------------------------------------------------------------
```

追加。

HOS に対して割り込みベクタ番号の範囲を指定します。H8/3687 の場合，14～32 が NMI 以外の外部割り込みになるので，「system.cfg」を次のように修正します。（内部割込みを使うときは修正します。）

```
//HOS_MIN_INTNO(19);        /* 使用する割り込み番号の最小値(省略時 0) */
//HOS_MAX_INTNO(23);        /* 使用する割り込み番号の最大値(省略時 0) */
HOS_MIN_INTNO(14);          /* 使用する割り込み番号の最小値(省略時 0) */
HOS_MAX_INTNO(32);          /* 使用する割り込み番号の最大値(省略時 0) */
```

最後に「sample.c」を次のように修正します。

```
/* ------------------------------------------------------------------------ */
/*  Hyper Operating System V4  サンプルプログラム                           */
/*                                                                          */
/*                                Copyright (C) 1998-2002 by Project HOS    */
/*                                http://sourceforge.jp/projects/hos/       */
/* ------------------------------------------------------------------------ */
/*                                Copyright (C) 2009 by TOYO-LINX Co.,Ltd.  */
/*                                                urf.ne.jp/~toyolinx       */
/* ------------------------------------------------------------------------ */

#include "kernel.h"
#include "kernel_id.h"
//#include "h83664f.h"
#include "iodefine.h"
#include "h8t_sci.h"
```

I/O 定義ファイルを，HEW が生成したものに変更する。

```
#include "binary.h"            //Cで2進数表示を使う(00000000-11111111)

/* メイン関数 */
int main()
{
    /* SCIの初期化 */
//  Sci_Initialize(SCI_19200);
    Sci_Initialize(SCI_38400);

    /* 開始メッセージ */
    Sci_PutChar('H');
    Sci_PutChar('O');
    Sci_PutChar('S');
    Sci_PutChar('¥r');
    Sci_PutChar('¥n');

    sta_hos();

    return 0;
}

/* 初期化ハンドラ */
void Initialize(VP_INT exinf)
{
/*      act_tsk(TSKID_SAMPLE1);*/
}

/* サンプルタスク */
void Task1(VP_INT exinf)
{
    SYSTIM st;

    for ( ; ; )
    {
        /* タイマ値取得 */
        get_tim(&st);

        /* タイマ値出力 */
        Sci_PutChar('0' + (st.ltime / 10000) % 10);
        Sci_PutChar('0' + (st.ltime / 1000) % 10);
        Sci_PutChar('0' + (st.ltime / 100) % 10);
        Sci_PutChar('0' + (st.ltime / 10) % 10);
        Sci_PutChar('0' + (st.ltime / 1) % 10);
        Sci_PutChar(':');

        /* タスクメッセージ */
        Sci_PutChar('T');
        Sci_PutChar('a');
        Sci_PutChar('s');
        Sci_PutChar('k');
        Sci_PutChar('1');
        Sci_PutChar('¥r');
        Sci_PutChar('¥n');
```

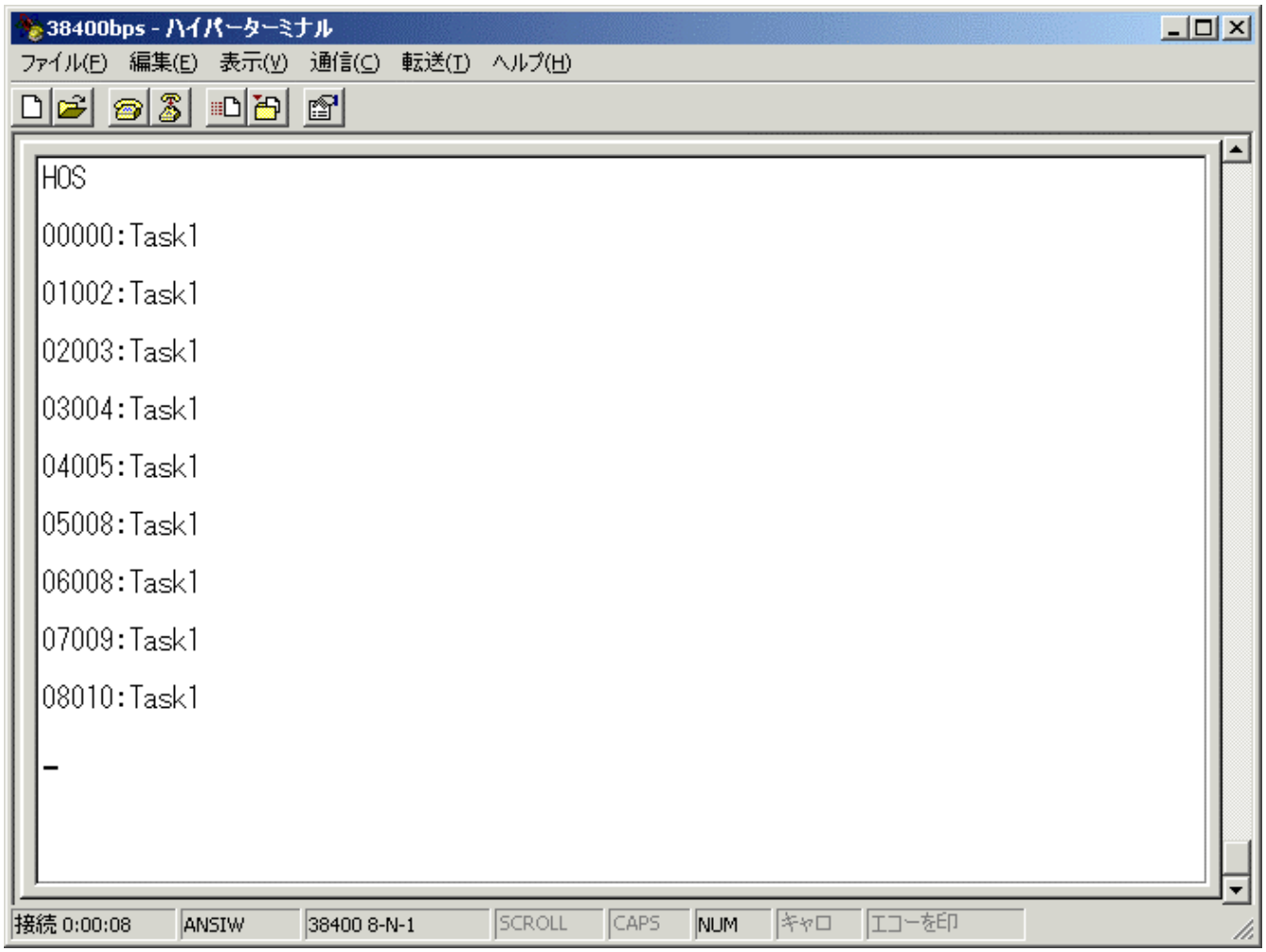
C で 2 進数表現を使えるように，東洋リンクス作成の定義ファイルをインクルードする。

ボーレートを 38400 ボーに変更する。

```
        /* 1秒待つ */
        dly_tsk(1000);
    }
}

/* ------------------------------------------------------------------------ */
/*  Copyright (C) 1998-2003 by Project HOS                                  */
/*  Copyright (C) 2009      by TOYO-LINX Co.,Ltd.                           */
/* ------------------------------------------------------------------------ */
```

ここで一つファイルをコピーしておきます。付属の CD の中に「binary.h」がありますので，「C:¥hew4_tk3687¥hosv4¥sample_01¥」フォルダにコピーします。

お疲れ様でした。これで修正作業は完了です。ビルドしてみましょう。エラーがでたときは入力ミスがないかもう一度確認してください。OK なら「C:¥hew4_tk3687¥hosv4¥sample_01¥Debug¥」フォルダの「sample_01.mot」を FDT で TK-3687/TK-3687mini にダウンロードして実行しましょう。E8a をお持ちの方は，E8a でダウンロード・実行してください。実行の様子はシリアルポートを介してターミナルに表示されます。ターミナルの設定はハイパーモニタと同じです。次のように表示されれば完成です。



5. マルチタスクを体験しよう

サンプルプログラムは 1 本のタスクを実行しました。しかし，HOS はマルチタスク OS なので，複数のタスクを実行するときに真価を発揮します。それで，このサンプルプログラムを改造して 2 本のタスクを実行するプログラムを作り，マルチタスクを体験してみましょう。プログラムの内容は次のようなものです。

タスク 1 ： 1 秒おきにシリアルポートにシステムタイマの値を送信する。
タスク 2 ： 0.1 秒おきにポート 5 とポート 6 の出力値を+1 する。

では，今作成した「sample_01」を立ち上げてください。これをそのまま改造してもよいのですが，このマニュアル（および付属 CD）では，このワークスペースに新たに「sample_02」というアプリケーションプロジェクトを追加します。これまで説明してきた手順（3 項と 4 項）で sample_01 とまったく同じプログラムを作ってください。

では改造していきましょう。まず修正するのはコンフィギュレーションファイル「system.cfg」です。このファイルにはタスクや資源などの生成情報を記述します。ここに「Task2」というタスクを追加します。次のように修正・追加してください。

```
/* サンプルプログラム */
ATT_INI({TA_HLNG, 0, Initialize});
CRE_TSK(TSKID_SAMPLE1, {TA_HLNG|TA_ACT, 1, Task1, 1, 256, NULL});
CRE_TSK(TSKID_SAMPLE2, {TA_HLNG|TA_ACT, 2, Task2, 1, 256, NULL});
```

「CRE_TSK」は「Create Task」の略でタスクを生成する静的 API です。

次の「TSKID_SAMPLE1」はタスク ID で，コンフィギュレータによってほかのタスクと重ならないようにこのラベルに番号が割り付けられます。このプログラムでは使っていませんが，アプリケーションがタスクを終了したり，起床したり，待ち状態に遷移させたりするときは，タスク ID でどのタスクかを指定します。

{ }内の「TA_HLNG|TA_ACT」は属性です。「TA_HLNG」はタスクが高級言語（C 言語）で記述されていることを示し，「TA_ACT」は OS の起動時にこのタスクを実行状態，もしくは実行可能状態にすることを示しています。ここでは「｜」（オア）でつながっているので，「このタスクは高級言語で記述され，かつ，OS 起動時に実行状態にする」ということを指定しています。

次の数字「1」は拡張情報で，この数値がタスク起動時に引数として渡されます（サンプルプログラムでは特に使っていない）。

「Task1」はタスクの開始アドレスです。サンプルプログラムのタスクの関数名を指定します。

次の数字「1」はタスクの優先順位を示しています。1から始まる数字で小さいほど優先順位が高くなります。サンプルプログラムでは二つのタスクの優先順位を同じにしています。

次の数字「256」はスタックのサイズを示し，その次の引数で指定したアドレスにスタック領域が確保されます。「NULL」を指定するとコンフィギュレータが自動的に領域を確保します。

さて，続いてサンプルプログラム「sample.c」を次のように改造します。

```
/* ------------------------------------------------------------------------ */
/*  Hyper Operating System V4  サンプルプログラム                           */
/*                                                                          */
```

```
/*                                  Copyright (C) 1998-2002 by Project HOS  */
/*                                  http://sourceforge.jp/projects/hos/     */
/* ------------------------------------------------------------------------ */
/*                                  Copyright (C) 2009 by TOYO-LINX Co.,Ltd.*/
/*                                  http://www2.u-netsurf.ne.jp/~toyolinx   */
/* ------------------------------------------------------------------------ */

#include "kernel.h"
#include "kernel_id.h"
//#include "h83664f.h"
#include "iodefine.h"
#include "h8t_sci.h"

#include "binary.h"          //Cで2進数表示を使う(00000000-11111111)

/* メイン関数 */
int main()
{
    /* SCIの初期化 */
//  Sci_Initialize(SCI_19200);
    Sci_Initialize(SCI_38400);

    /* 開始メッセージ */
    Sci_PutChar('H');
    Sci_PutChar('O');
    Sci_PutChar('S');
    Sci_PutChar('¥r');
    Sci_PutChar('¥n');

    sta_hos();

    return 0;
}

/* 初期化ハンドラ */
void Initialize(VP_INT exinf)
{
/*      act_tsk(TSKID_SAMPLE1);*/

    // I/Oポートのイニシャライズ
    IO.PCR5 = _11111111B;   //P50-57出力
    IO.PCR6 = _11111111B;   //P60-67出力
}

/* サンプルタスク */
void Task1(VP_INT exinf)
{
    SYSTIM st;

    for ( ; ; )
    {
        /* タイマ値取得 */
        get_tim(&st);

        /* タイマ値出力 */
```

ポート5とポート6のイニシャライズ。
アプリケーションのイニシャライズは初期化ハンドラに記述する。

```
        Sci_PutChar('0' + (st.ltime / 10000) % 10);
        Sci_PutChar('0' + (st.ltime / 1000) % 10);
        Sci_PutChar('0' + (st.ltime / 100) % 10);
        Sci_PutChar('0' + (st.ltime / 10) % 10);
        Sci_PutChar('0' + (st.ltime / 1) % 10);
        Sci_PutChar(':');

        /* タスクメッセージ */
        Sci_PutChar('T');
        Sci_PutChar('a');
        Sci_PutChar('s');
        Sci_PutChar('k');
        Sci_PutChar('1');
        Sci_PutChar('¥r');
        Sci_PutChar('¥n');

        /* 1秒待つ */
        dly_tsk(1000);
    }
}
```

追加するタスク。

```
void Task2(VP_INT exinf)
{
    while(1){
        // LEDをカウントアップ表示
        IO.PDR5.BYTE = IO.PDR5.BYTE + 1;
        IO.PDR6.BYTE = IO.PDR6.BYTE + 1;

        // 0.1s待つ
        dly_tsk(100);
    }
}

/* ------------------------------------------------------------------------ */
/*  Copyright (C) 1998-2003 by Project HOS                                  */
/*  Copyright (C) 2009      by TOYO-LINX Co.,Ltd.                           */
/* ------------------------------------------------------------------------ */
```

1s 待ったり，0.1s待ったりするときに「dly_tsk」を使っています。「dly_tsk」は指定された時間，タスクを待ち状態にするサービスコールです。

もう一つ，「sample.h」も修正します。サンプルプログラムのプロトタイプ宣言が記されています。関数 Task2 の定義を記述します。

```
/* ------------------------------------------------------------------------ */
/*  Hyper Operating System V4  サンプルプログラム                           */
/*                                                                          */
/*                                Copyright (C) 1998-2002 by Project HOS    */
/*                                http://sourceforge.jp/projects/hos/       */
/* ------------------------------------------------------------------------ */
/*                                Copyright (C) 2009 by TOYO-LINX Co.,Ltd.  */
/*                                http://www2.u-netsurf.ne.jp/~toyolinx     */
/* ------------------------------------------------------------------------ */
```

```
#ifndef __PROJECT_HOS__sample_h__
#define __PROJECT_HOS__sample_h__



void Initialize(VP_INT exinf);
void Task1(VP_INT exinf);
void Task2(VP_INT exinf);


#endif    /* __PROJECT_HOS__sample_h__ */



/* ------------------------------------------------------------------------ */
/*  Copyright (C) 1998-2002 by Project HOS                                  */
/*  Copyright (C) 2009      by TOYO-LINX Co.,Ltd.                           */
/* ------------------------------------------------------------------------ */
```

追加

では，ビルドしてみましょう。エラーがでたときは入力ミスがないかもう一度確認してください。OK なら「 C:¥hew4_tk3687¥hosv4¥sample_02¥Debug¥ 」フォルダの「 sample_02.mot 」を FDT で TK-3687/TK-3687mini にダウンロードして実行しましょう。E8a をお持ちの方は，E8a でダウンロード・実行してください。前回と同じようにターミナルに表示するとともに，ポート 5 とポート 6 の状態が+1 されますので，LED をつなぐとカウントアップしていく様子を見ることができます。

意外と簡単に感じたと思います。基本的には，それぞれのタスクの動作をそれぞれプログラムしておいて，並行して動かすための調整は HOS にお任せする，という感じです。これが OS を採用するメリットの一つでしょう。

6. 割り込みを使ってみよう

これまでのサンプルプログラムは「dly_tsk」を使ってタスクを待ち状態にし，その間にほかに実行できるタスクがあれば，その処理を行なっていました。逆に，何か条件がそろったら（外部入力が変化した，受信した，など）すぐに実行したい処理もあります。このようなとき OS 無しのプログラムでは割り込みを使うことが多いですが，HOS でも割り込み処理で対応することができます。今回はシリアルポートの受信処理を例に，割り込みプログラムを HOS に組み込んでみましょう。プログラムの内容は次のようなものです。

タスク 1 ： シリアルポートから受信したら，受信データをそのまま送信する。
タスク 2 ： 0.1 秒おきにポート 5 とポート 6 の出力値を+1 する。

今回は「sample_02」を改造します。ただし，このマニュアル（および付属 CD）では，このワークスペースに新たに「sample_03」というアプリケーションプロジェクトを追加します。これまで説明してきた手順（3 項と 4 項）で sample_02 とまったく同じプログラムを作ってください。

では改造していきましょう。まず修正するのはコンフィギュレーションファイル「system.cfg」です。次のように修正・追加してください。

```
/* SCI3 */
ATT_ISR({TA_HLNG, 0, 23, SCI3_RxiHandler});

/* サンプルプログラム */
ATT_INI({TA_HLNG, 0, Initialize});
CRE_TSK(TSKID_SAMPLE1, {TA_HLNG|TA_ACT, 1, Task1, 2, 256, NULL});
CRE_TSK(TSKID_SAMPLE2, {TA_HLNG|TA_ACT, 2, Task2, 1, 256, NULL});
```

追加

変更

「ATT_ISR」は割り込みサービスルーチンを定義する静的 API です。

{ }内の「TA_HLNG」は属性です。「TA_HLNG」は割り込みサービスルーチンが高級言語（C 言語）で記述されていることを示しています。

次の数字「0」は拡張情報で，この数値が割り込みサービスルーチンに引数として渡されます（サンプルプログラムでは特に使っていない）。

次の数字「23」は割り込みベクタ番号です。

次の「SCI3_RxiHandler」は割り込みサービスルーチンの先頭アドレスです。関数名を指定します。

「CRE_TSK」も変更します。タスク 1 の優先順位を「2」にします。タスク 1 は受信したかどうかを常に監視することになりますが，もしタスク 2 と優先順位が同じか，タスク 1 のほうの優先順位が高いと，タスク 1 ばかりが実行されてタスク 2 の処理が行なわれません。タスク 2 の処理を優先的に行ない，0.1 秒の待ち時間の間にタスク 1 の処理を行なうようにします。

続いてシリアル制御用プログラムライブラリ「h8_sci.c」を次のように改造します（黄色でマークしている行）。受信割り込み「Sci3_RxiHandler」でリングバッファにストアし，サブルーチン「Sci3_GetChar」でリングバッファから 1 文字ずつ取り出します。

```
/* ------------------------------------------------------------------------ */
/*  H8/3664用 SCI3制御ライブラリ                                            */
/*                                                                          */
```

```
/*                                      Copyright (C) 1998-2006 by Project HOS  */
/*                                      http://sourceforge.jp/projects/hos/     */
/* ------------------------------------------------------------------------ */
/*                                      Copyright (C) 2009 by TOYO-LINX Co.,Ltd.*/
/*                                      http://www2.u-netsurf.ne.jp/~toyolinx   */
/* ------------------------------------------------------------------------ */
#include "kernel.h"
//#include "h83664f.h"
#include "iodefine.h"
#include "h8t_sci.h"

#include "binary.h"          //Cで2進数表示を使う(00000000-11111111)

#define RECV_BUFSIZE    32        /* 受信バッファのサイズ */

static unsigned char recv_buf[RECV_BUFSIZE];
static int head;
static int tail;

/* SCI3初期化 */
void SCI3_Initialize(unsigned char rate)
{
    volatile int i;

    /* SCI3初期化 */
    SCI3.SCR3.BYTE = 0x00;
    SCI3.SMR.BYTE = 0x00;
    SCI3.BRR = rate;
    for ( i = 0; i < 280; i++ )
        ;
//  SCI3.SCR3.BYTE = 0x20; /* 送信可 */
    SCI3.SCR3.BYTE = 0x70; /* 送信可|受信可 */
    IO.PMR1.BIT.TXD = 1;
}

/* １文字出力 */
void SCI3_PutChar(char c)
{
    while ( !(SCI3.SSR.BYTE & 0x80) )
        ;

    SCI3.TDR  = c;
/*      SCI3.SSR.BYTE &= 0x7f;  */
}

/* 1文字入力 */
int SCI3_GetChar(void)
{
    unsigned char c;

    if (head==tail){
        return -1;
    }

    c = recv_buf[head++];
```

```
    if (head>=RECV_BUFSIZE){
        head = 0;
    }

    return c;
}

/* SCI受信割り込み */
void SCI3_RxiHandler(VP_INT exinf)
{
    unsigned char c;
    int next;

    /* 1文字受信 */
    c = SCI3.RDR;

    /* 次の末尾位置を計算 */
    next = tail + 1;
    if ( next == RECV_BUFSIZE )
    {
        next = 0;
    }

    /* オーバーフローチェック */
    if ( next == head )
    {
        return;
    }

    /* 受信バッファに格納 */
    recv_buf[tail] = c;
    tail = next;
}

/* ------------------------------------------------------------------------ */
/*                                  Copyright (C) 1998-2006 by Project HOS  */
/*                                  http://sourceforge.jp/projects/hos/     */
/* ------------------------------------------------------------------------ */
/*                                  Copyright (C) 2009 by TOYO-LINX Co.,Ltd.*/
/*                                  http://www2.u-netsurf.ne.jp/~toyolinx   */
/* ------------------------------------------------------------------------ */
```

シリアル制御用プログラムのプロトタイプ宣言と定義ファイル「h8_sci.h」を次のように改造します（一部分抜粋，黄色でマークしている行が追加部分）。

```
/* SCI3 */
void SCI3_Initialize(unsigned char rate);  /* SCI3初期化 */
void SCI3_PutChar(char c);                 /* １文字出力 */
int SCI3_GetChar(void);                    /* １文字入力 */
void SCI3_RxiHandler(VP_INT exinf);        /* 受信割り込みハンドラ */

/* assign SCI3 for SCI*/
#define Sci_Initialize SCI3_Initialize
```

```
#define Sci_PutChar SCI3_PutChar
#define Sci_GetChar SCI3_GetChar
#define Sci_RxiHandler SCI3_RxiHandler
```

続いてサンプルプログラム「sample.c」を次のように改造します。

```
/* ------------------------------------------------------------------------ */
/*  Hyper Operating System V4  サンプルプログラム                           */
/*                                                                          */
/*                                  Copyright (C) 1998-2002 by Project HOS  */
/*                                  http://sourceforge.jp/projects/hos/     */
/* ------------------------------------------------------------------------ */
/*                                  Copyright (C) 2009 by TOYO-LINX Co.,Ltd.*/
/*                                  http://www2.u-netsurf.ne.jp/~toyolinx   */
/* ------------------------------------------------------------------------ */

#include "kernel.h"
#include "kernel_id.h"
//#include "h83664f.h"
#include "iodefine.h"
#include "h8t_sci.h"

#include "binary.h"          //Cで2進数表示を使う(00000000-11111111)

/* メイン関数 */
int main()
{
    /* SCIの初期化 */
//  Sci_Initialize(SCI_19200);
    Sci_Initialize(SCI_38400);

    /* 開始メッセージ */
    Sci_PutChar('H');
    Sci_PutChar('O');
    Sci_PutChar('S');
    Sci_PutChar('¥r');
    Sci_PutChar('¥n');

    sta_hos();

    return 0;
}

/* 初期化ハンドラ */
void Initialize(VP_INT exinf)
{
/*      act_tsk(TSKID_SAMPLE1);*/

    // I/Oポートのイニシャライズ
    IO.PCR5 = _11111111B;   //P50-57出力
    IO.PCR6 = _11111111B;   //P60-67出力
}

/* サンプルタスク */
```

```
void Task1(VP_INT exinf)
{
    int c;

    while(1){
        c = Sci_GetChar(); //1文字入力

        if (c!=-1){
            Sci_PutChar((char)(c & 0x00ff)); //受信データを送信
        }
    }
}

void Task2(VP_INT exinf)
{
    while(1){
        // LEDをカウントアップ表示
        IO.PDR5.BYTE = IO.PDR5.BYTE + 1;
        IO.PDR6.BYTE = IO.PDR6.BYTE + 1;

        // 0.1s待つ
        dly_tsk(100);
    }
}

/* ------------------------------------------------------------------------ */
/*  Copyright (C) 1998-2003 by Project HOS                                  */
/*  Copyright (C) 2009      by TOYO-LINX Co.,Ltd.                           */
/* ------------------------------------------------------------------------ */
```

では，ビルドしてみましょう。エラーがでたときは入力ミスがないかもう一度確認してください。OK なら「 C:¥hew4_tk3687¥hosv4¥sample_03¥Debug¥ 」フォルダの「 sample_03.mot 」を FDT で TK-3687/TK-3687mini にダウンロードして実行しましょう。E8a をお持ちの方は，E8a でダウンロード・実行してください。ポート 5 とポート 6 の状態が+1 されますので，LED をつなぐとカウントアップしていくのと同時に，パソコンのキーボードから入力すると同じ文字が表示されます。

7. タスク付属同期機能

これまでのプログラムは各タスクがそれぞれ個別に動作していました。しかし，複数のタスクが協調しながら動作するアプリケーションも考えられます。例えば，通常動作しているタスク内である条件がそろったときに，より優先順位の高いタスクを実行する，というものです。「dly_tsk」を使ってタスクを待ち状態にし，その間にほかに実行できるタスクがあれば，その処理を行なう，という方法はすでに説明しましたが，ここではより積極的に特定のタスクを実行することを考えてみましょう。

このようなときに使う機能が「タスク付属同期機能」です。μITRON4.0 仕様書では「タスクの状態を直接的に操作することによって同期を行なうための機能である。」と説明されています（実を言うと「dly_tsk」もタスク付属同期機能の一部です）。では，タスク付属同期機能を使って次のプログラムを作ってみましょう。

タスク１ ： シリアルポートから受信したら，受信データをポート５とポート６に出力するとともに，受信データが‘0’～‘9’のときはタスク２を起床する。

タスク２ ： シリアルポートに「Receive Number」と送信する。

今回は「sample_03」を改造します。ただし，このマニュアル（および付属 CD）では，このワークスペースに新たに「sample_04」というアプリケーションプロジェクトを追加します。これまで説明してきた手順（3 項と 4 項）で sample_03 とまったく同じプログラムを作ってください。

では改造していきましょう。修正するのは「sample.c」だけです。次の黄色でマークしている部分を修正・追加してください。

```
/* ------------------------------------------------------------------------ */
/*  Hyper Operating System V4  サンプルプログラム                           */
/*                                                                          */
/*                                  Copyright (C) 1998-2002 by Project HOS  */
/*                                  http://sourceforge.jp/projects/hos/     */
/* ------------------------------------------------------------------------ */
/*                                  Copyright (C) 2009 by TOYO-LINX Co.,Ltd.*/
/*                                  http://www2.u-netsurf.ne.jp/~toyolinx   */
/* ------------------------------------------------------------------------ */

#include "kernel.h"
#include "kernel_id.h"
//#include "h83664f.h"
#include "iodefine.h"
#include "h8t_sci.h"

#include "binary.h"          //Cで2進数表示を使う(00000000-11111111)

/* メイン関数 */
int main()
{
    /* SCIの初期化 */
//  Sci_Initialize(SCI_19200);
    Sci_Initialize(SCI_38400);

    /* 開始メッセージ */
    Sci_PutChar('H');
    Sci_PutChar('O');
    Sci_PutChar('S');
```

```
    Sci_PutChar('¥r');
    Sci_PutChar('¥n');

    sta_hos();

    return 0;
}

/* 初期化ハンドラ */
void Initialize(VP_INT exinf)
{
/*      act_tsk(TSKID_SAMPLE1);*/

    // I/Oポートのイニシャライズ
    IO.PCR5 = _11111111B;   //P50-57出力
    IO.PCR6 = _11111111B;   //P60-67出力
}

/* サンプルタスク */
void Task1(VP_INT exinf)
{
    int c;

    while(1){
        c = Sci_GetChar(); //1文字入力

        if ((c>='0')&(c<='9')){
            wup_tsk(TSKID_SAMPLE2); //タスク2を起床,実行可能状態か実行状態に遷移
        }

        if (c!=-1){
            IO.PDR5.BYTE = c;  //受信データをポート5に出力
            IO.PDR6.BYTE = c;  //受信データをポート6に出力
        }
    }
}

void Task2(VP_INT exinf)
{
    while(1){
        slp_tsk();    //自タスクを待ち状態に遷移

        Sci_PutChar('R');  //メッセージを送信
        Sci_PutChar('e');
        Sci_PutChar('c');
        Sci_PutChar('e');
        Sci_PutChar('i');
        Sci_PutChar('v');
        Sci_PutChar('e');
        Sci_PutChar(' ');
        Sci_PutChar('N');
        Sci_PutChar('u');
        Sci_PutChar('m');
        Sci_PutChar('b');
        Sci_PutChar('e');
```

```
            Sci_PutChar('r');
            Sci_PutChar('¥r');
            Sci_PutChar('¥n');
        }
}

/* ------------------------------------------------------------------------ */
/*  Copyright (C) 1998-2003 by Project HOS                                  */
/*  Copyright (C) 2009      by TOYO-LINX Co.,Ltd.                           */
/* ------------------------------------------------------------------------ */
```

Task1 の中の「wup_tsk(TSKID_SAMPLE2)」が，TSKID_SAMPLE2 で指定するタスクを起床するサービスコールです。wup_tsk が発行されると，指定されたタスクは実行可能状態，もしくは実行状態に遷移します。

Task2 の中の「slp_tsk()」は自タスクを待ち状態に遷移するサービスコールです。

さて，「system.cfg」で Task2 の優先順位が Task1 よりも高くなっています。このことを踏まえてプログラムの動きを考えてみましょう。

プログラムが動き出すと，「CRE_TSK」により Task1 と Task2 はどちらも実行状態，もしくは実行可能状態になります。ただし，Task2 の優先順位が Task1 より高いので，Task2 が実行状態，Task1 が実行可能状態になります。

Task2 は「slp_tsk」により，すぐに待ち状態になります。その結果，Task1 が実行状態になります。

Task1 はシリアルポートから受信するとデータの範囲を判定し，‘0’～‘9’のときに「wup_tsk」を発行してタスク 2 を実行可能状態にします。すると，タスク 2 の優先順位がタスク 1 より高いので，タスク 1 は実行可能状態に遷移し，タスク 2 が実行状態になりメッセージの送信処理を実行します。

Task2 はメッセージを送信したあと「slp_tsk」により，すぐに待ち状態になります。その結果，Task1 が実行状態になり再び受信を待ちます。

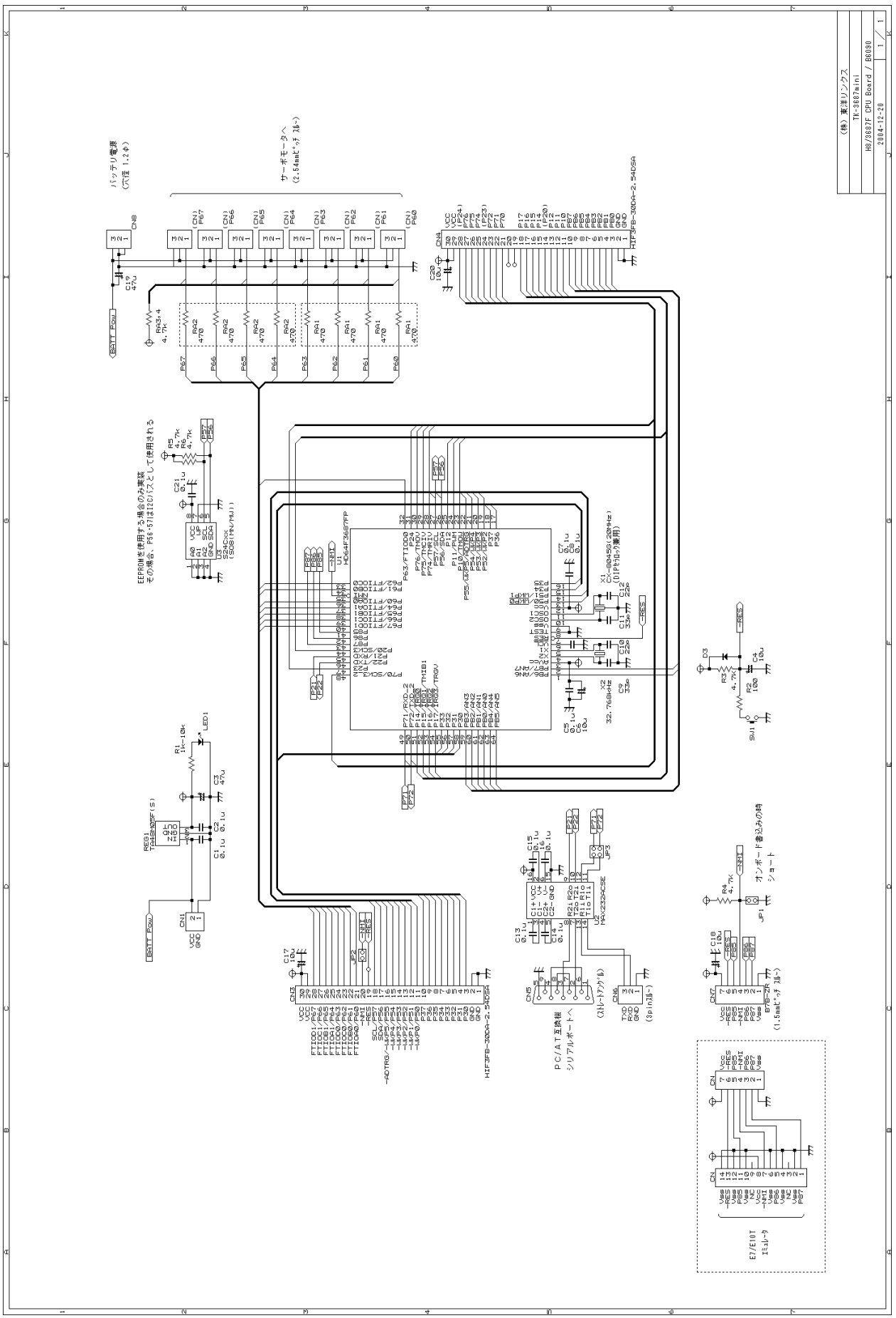
このように，タスク 1 とタスク 2 が協調しながら動作していくことになります。

◆

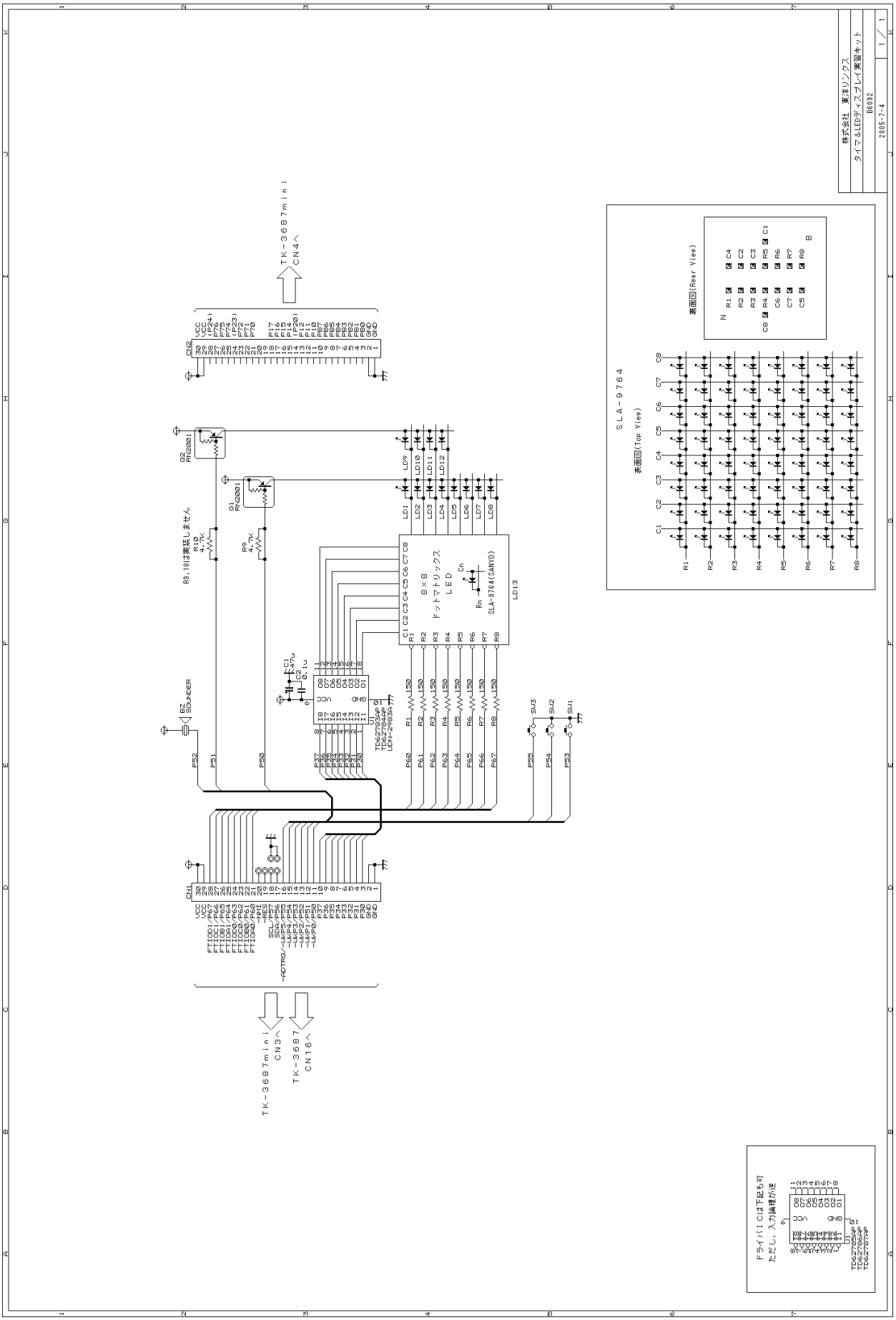
いかがだったでしょうか。HOS を H8/3687 に実装するまでは手順も多く複雑に思えたかもしれません。それでも，マルチタスクでアプリケーションタスクを実装すること自体は意外と簡単に感じたことでしょう。もちろん，μITRON の機能はこのマニュアルで説明したことだけではなく(というより説明していないことが大部分)，まだまだ入り口をくぐったに過ぎません。これをきっかけに組み込み OS についても興味を広げていただければ幸いです。

付録

回路図:TK-3687mini



回路図:タイマ&LED ディスプレイ



お問い合わせ先

株式会社東洋リンクス

※ご質問はメール，または FAX で…
ユーザーサポート係（月～金 10:00～17:00，土日祝は除く）
〒102-0093　東京都千代田区平河町 1-2-2　朝日ビル
TEL:03-3234-0559
FAX:03-3234-0549
E-mail:toyolinx@va.u-netsurf.jp
URL:http://www2.u-netsurf.ne.jp/~toyolinx

20091222