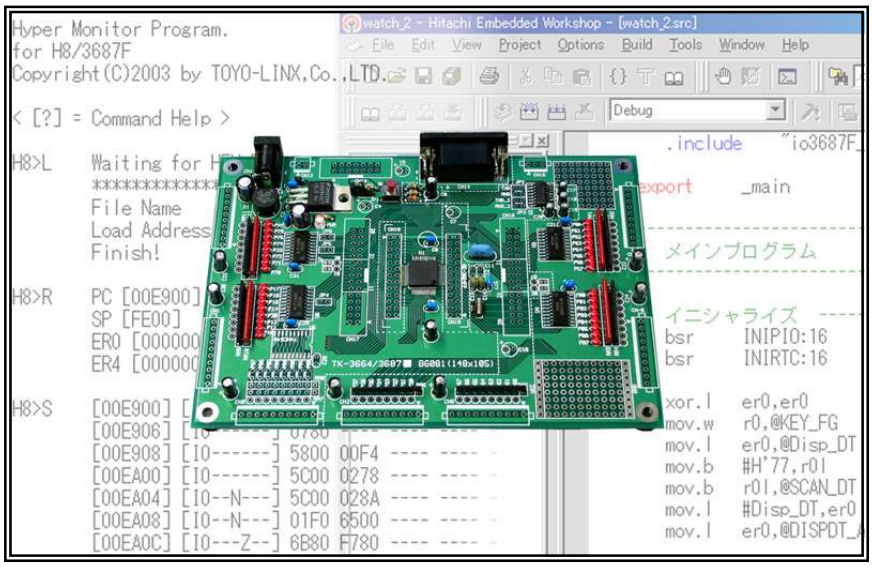
TK-3687版

マイコン事始め

～はじめてのマイコン～

Version 2.03



目次

はじめに …… P. 1
第 1 章　TK-3687 を眺めてみよう …… P. 3
第 2 章　ハイパーH8 を動かしてみよう …… P. 9
第 3 章　プログラムを動かしてみよう …… P. 16
第 4 章　プログラムを作ってみよう …… P. 19
第 5 章　開発環境を手に入れよう …… P. 27
第 6 章　アセンブラでプログラムを作ってみよう …… P. 30
第 7 章　I/O ポートの基本的な使い方をマスターしよう …… P. 48
第 8 章　シリアルポートの使い方入門 …… P. 65
第 9 章　I/O ポートのちょっと高度な応用例 …… P. 73
第 10 章　AD コンバータの使い方入門 …… P. 97
第 11 章　C 言語入門 …… P. 105
第 12 章　(応用編)アセンブラによるサーボモータの制御 …… P. 119
第 13 章　(応用編)C 言語による DC モータの制御 …… P. 130
付録(回路図, 参考資料) …… P. 136

(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-3687 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-3687 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-3687 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-3687 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。

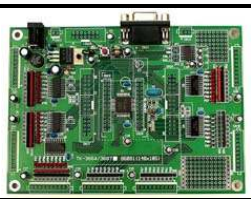
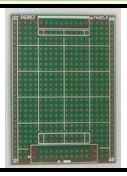
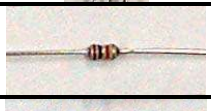
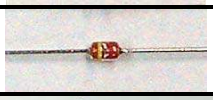
マニュアルについて

ルネサステクノロジのサイト（http://japan.renesas.com）から，マニュアルをダウンロードできます。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。

♯ ♪キットの内容を確かめて下さい。ちゃんとそろっていますか。♭ ♪

		部品名	メーカ	数	備考
1	CPU ボード	TK-3687	東洋リンクス	1	
2	ユニバーサル基板		東洋リンクス	1	
3	フレックス ジャンパケーブル （10 芯）			1	
4	丸ピンソケット （20 ピン）			1	 10 ピンずつ，半分に切って使います。
5	LED			1	
6	プッシュスイッチ	SKHHAK/AM/DC	ALPS	2	
7	サウンダ	QMX-05	STAR	1	
8	トランジスタ	2SC1881 2SD1823 etc.		1	
9	抵抗	1KΩ		3	
10	ダイオード	1S1588		1	
11	ツインモーター ギヤーボックス		タミヤ	1	
12	ラッピングワイヤ	50 cm		1	
13	モータ用ケーブル	10 cm		1	

- 部品は相当品を使用する事があります。
- 不足部品があるときは，東洋リンクスまでお問い合わせください。（巻末の連絡先参照）

TK-3687 を組み立てよう!!

TK-3687 は半完成品です。コンデンサやコネクタ等の部品についてはハンダ付けが必要です。「TK-3687 ユーザーズマニュアル」を見て，組み立ててください。

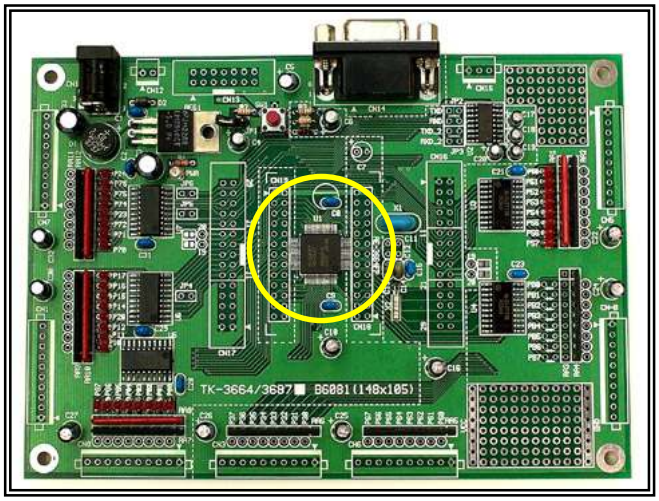
第1章 TK-3687 を眺めてみよう

1. TK-3687 の構成
2. CPU について
3. メモリについて

まずはマイコンと TK-3687 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. TK-3687 の構成

まずは TK-3687 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。

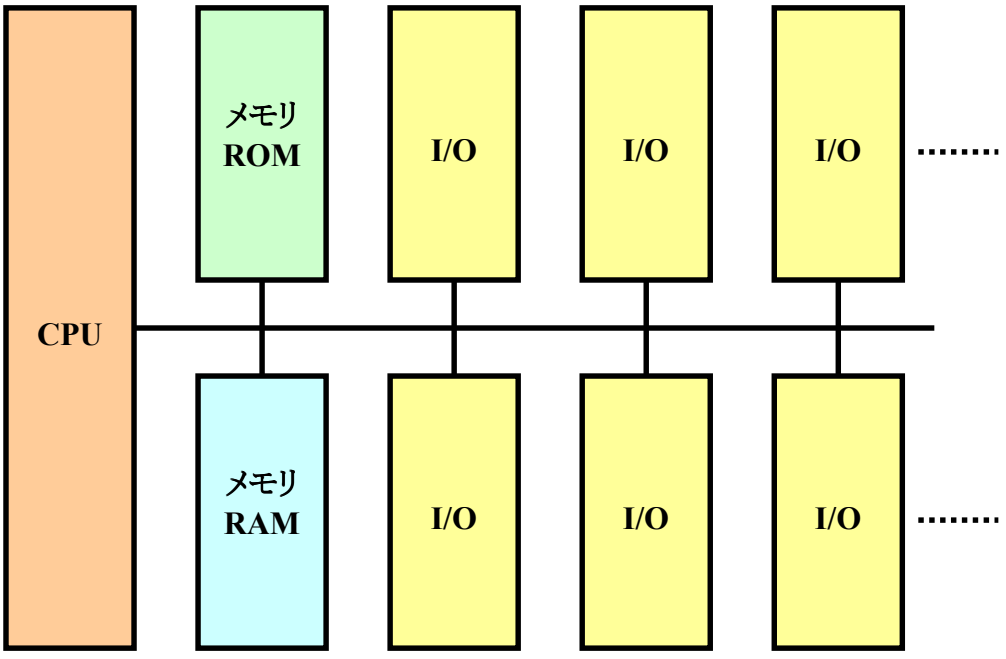


■ マイコンの 3 要素

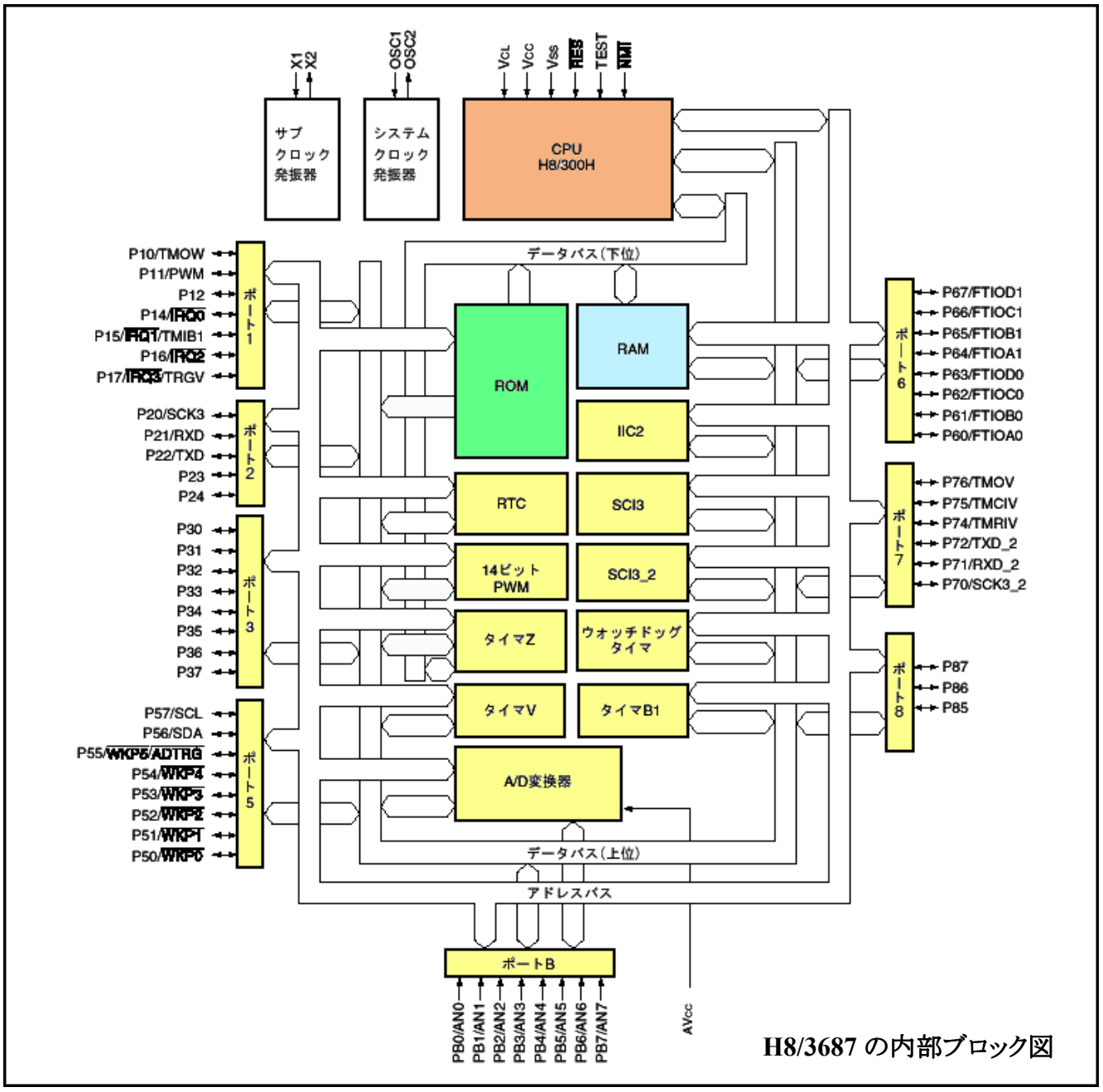
どんなマイコンでも次の基本的な 3 つの要素からできています。もちろん H8/3687 も例外ではありません。

- 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。TK-3687 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

2. CPU について

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。

■ レジスタ構成

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-3687（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

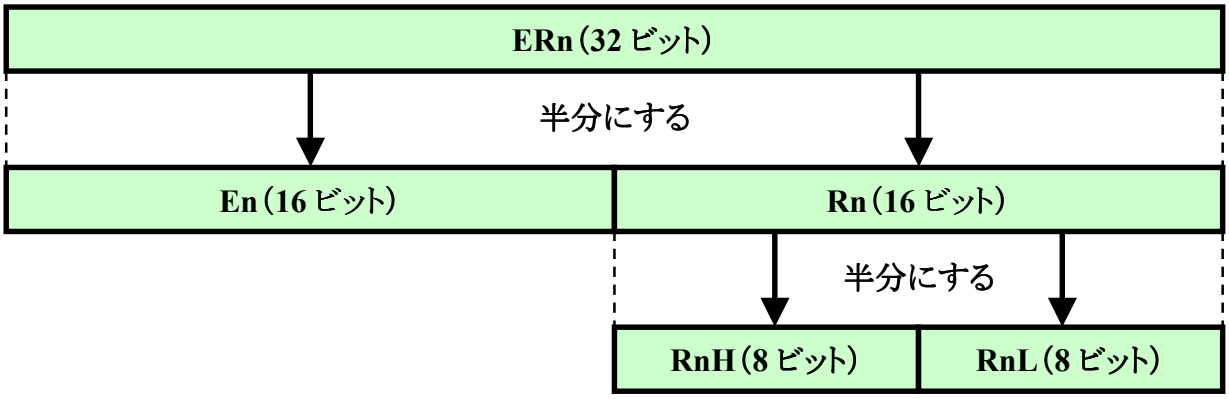
H8/300H は 32 ビット長の汎用レジスタを 8 本持っています。それぞれ，ER0～ER7 という名前がつけられています。

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…。

3. メモリについて

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。

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

◆

ここまでは話の入口です。次の章から，いよいよマイコンを動かしていきましょう。

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 とかと呼びます。

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 ハイパーH8 を動かしてみよう

1. ハイパーH8 って何?
2. TK-3687 とパソコンをつなぐ
3. ハイパーターミナルの設定
4. 電源オン!!

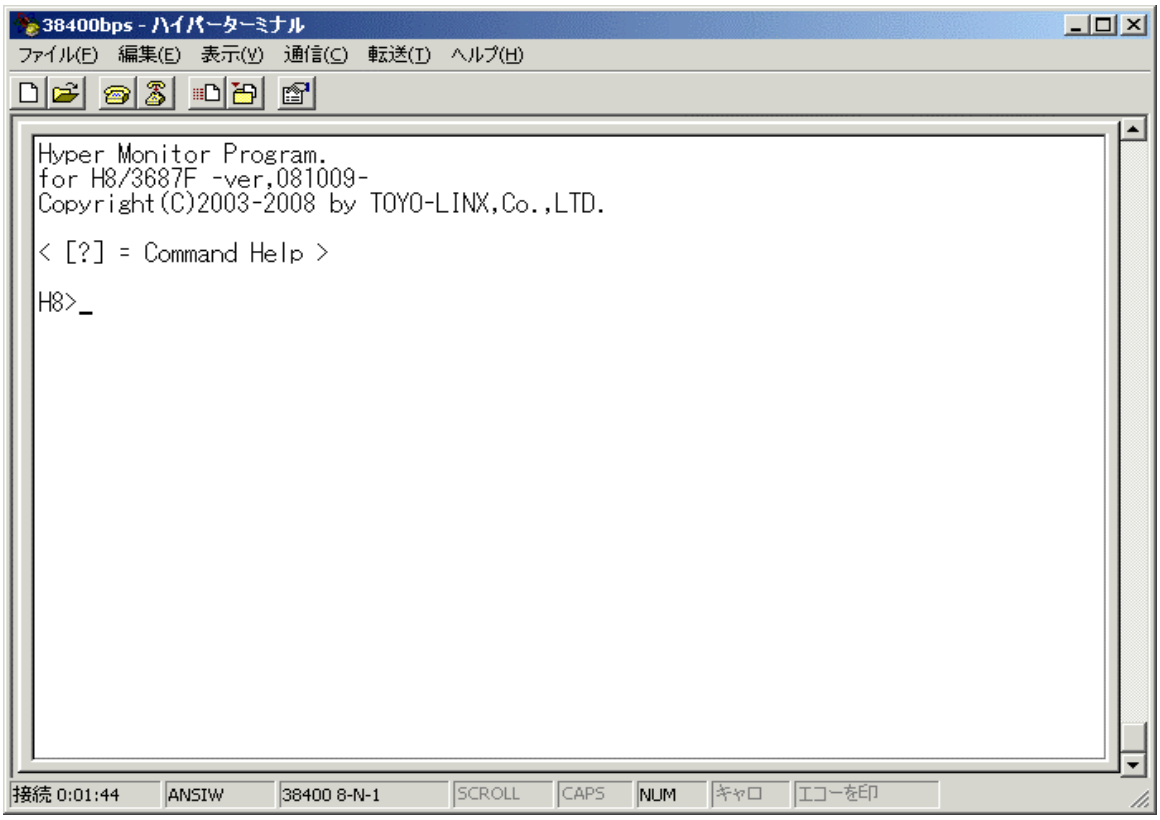
では，早速 TK-3687 を動かしてみましょう。とはいっても，TK-3687 をみると分かるように，電源をオンしてもなんだかよく分かりません。というわけで，マイコンの中身をのぞく道具を準備して，それを動かしてみましょう。その道具の名前は‘ハイパーH8’です。

1. ハイパーH8 って何?

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-3687 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。

ところでモニタとは何でしょうか。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，途中で止めることもできます。

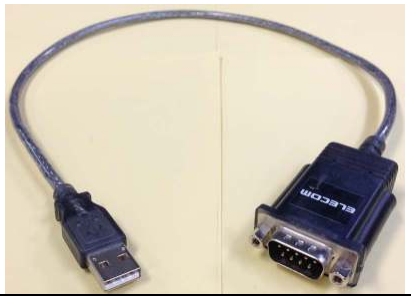
TK-3687 にはあらかじめハイパーH8 が書き込まれていて，電源オンですぐにマイコンの中身を見ることができます。



2. TK-3687 とパソコンをつなぐ

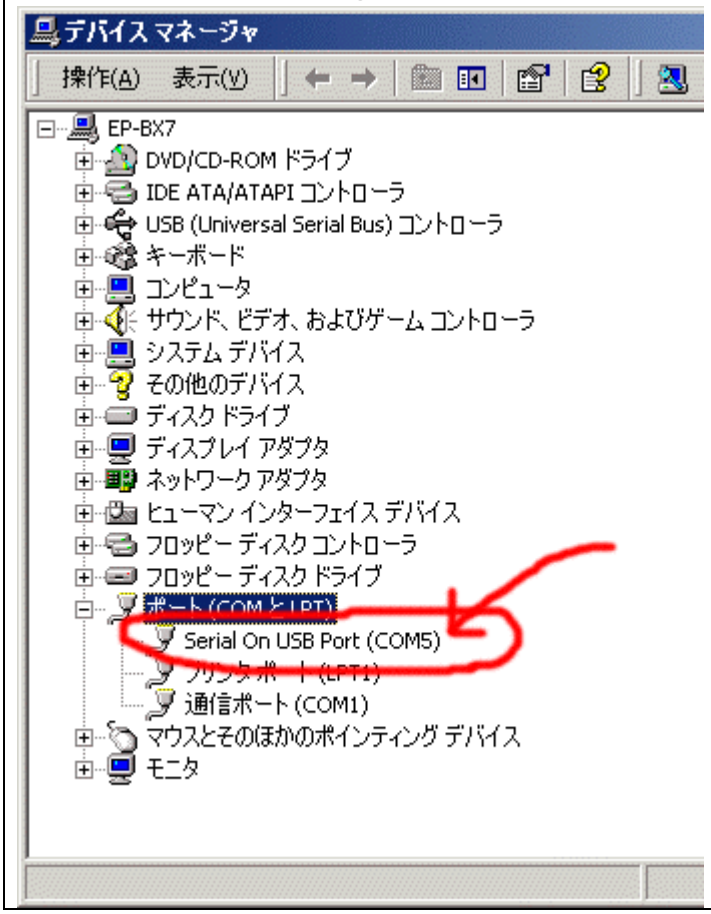
まず TK-3687 とパソコンをつなぎます。最近のパソコンは USB ポートしかないため，USB-シリアル変換ケーブルを用意します（写真①）。USB-シリアル変換ケーブルの説明書に従いドライバをインストールしてください。もちろん，パソコンに COM ポート（写真②）があればそれを使うことができます。ハイパーターミナルをつなぐときに必要になるのが COM 番号です。COM 番号はパソコンの環境に左右されるため，パソコンのデバイスマネージャを開き COM 番号を確認してください（写真③）。次に，D-Sub・9pin（オス）－9pin（メス）ストレートケーブル（写真④）で，メス側を USB-シリアル変換ケーブルの COM ポートへ，オス側を TK-3687 の CN14（写真⑤）へ接続します。なお，この時はまだ TK-3687 の電源は入れないでください。

①

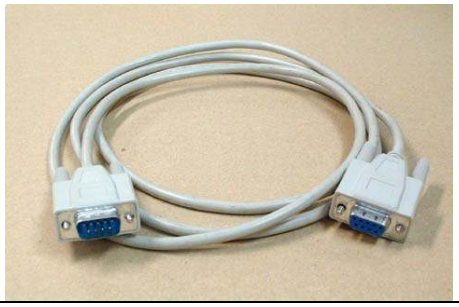


②

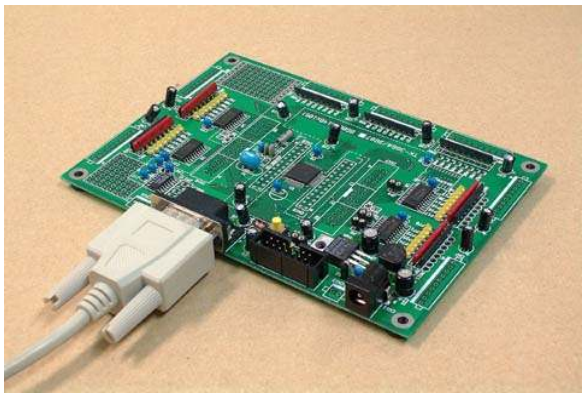
③



④



⑤



3. 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-3687 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，

から起動できます。Windows のバージョンによっては，

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，

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

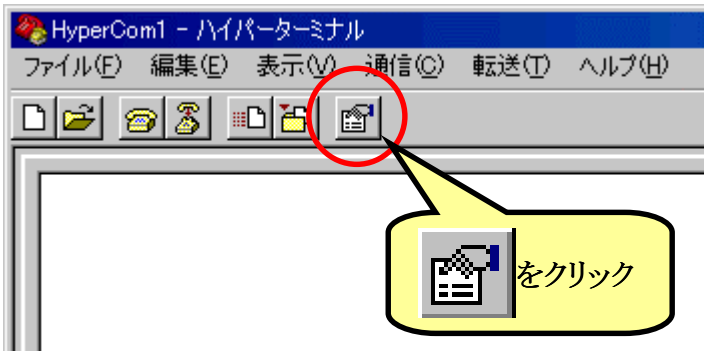
ｽﾄｯﾌﾟﾋﾞｯﾄ(S) :1

ﾌﾛｰ制御(F) :Xon/Xoff

設定し終えたら OK をクリックします。

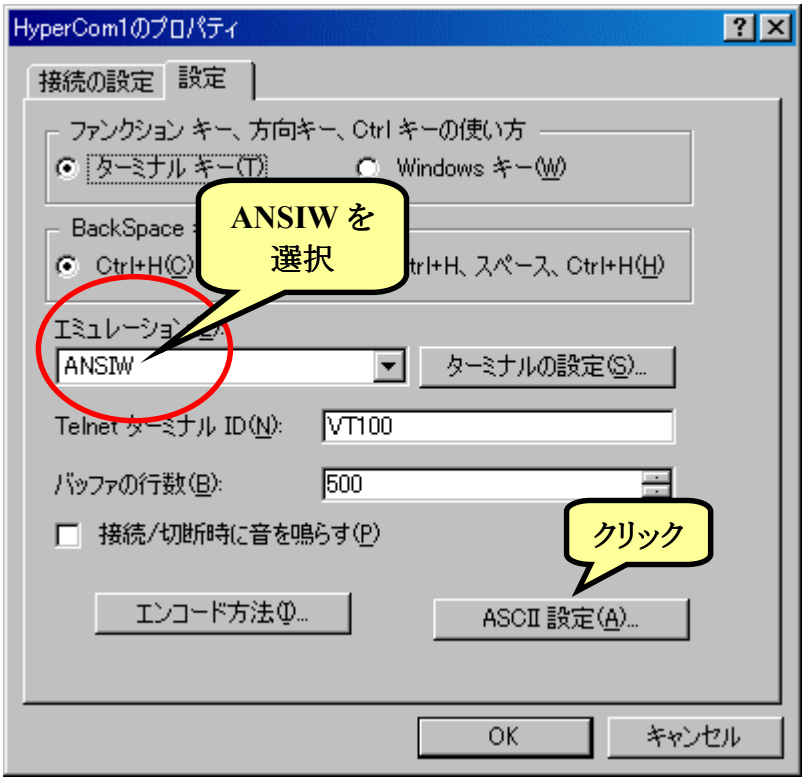
④ プロパティアイコンをクリック

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。



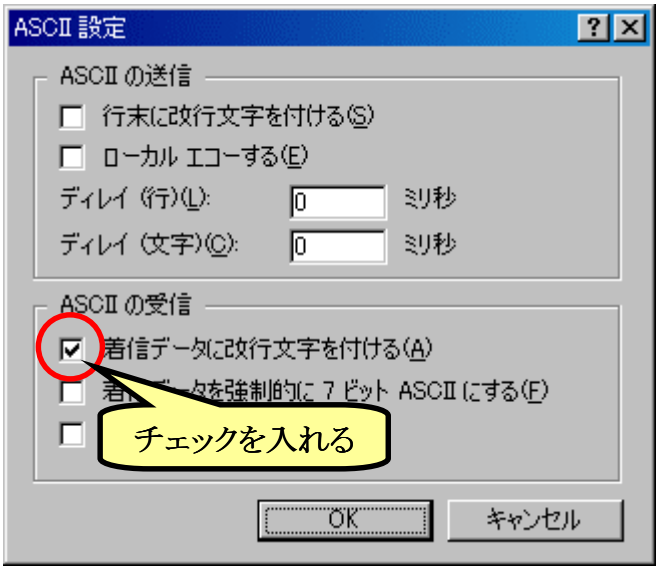
⑤ プロパティ

‘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。



◆

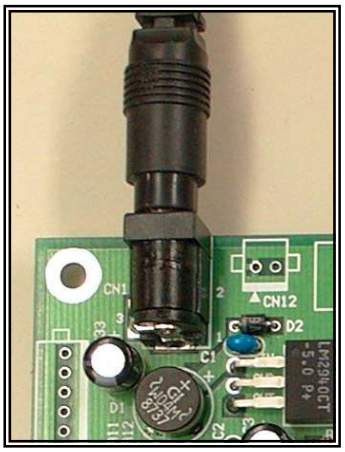
これで設定は終了です。それでは電源をオンしてみましょう。ちゃんと動くでしょうか。

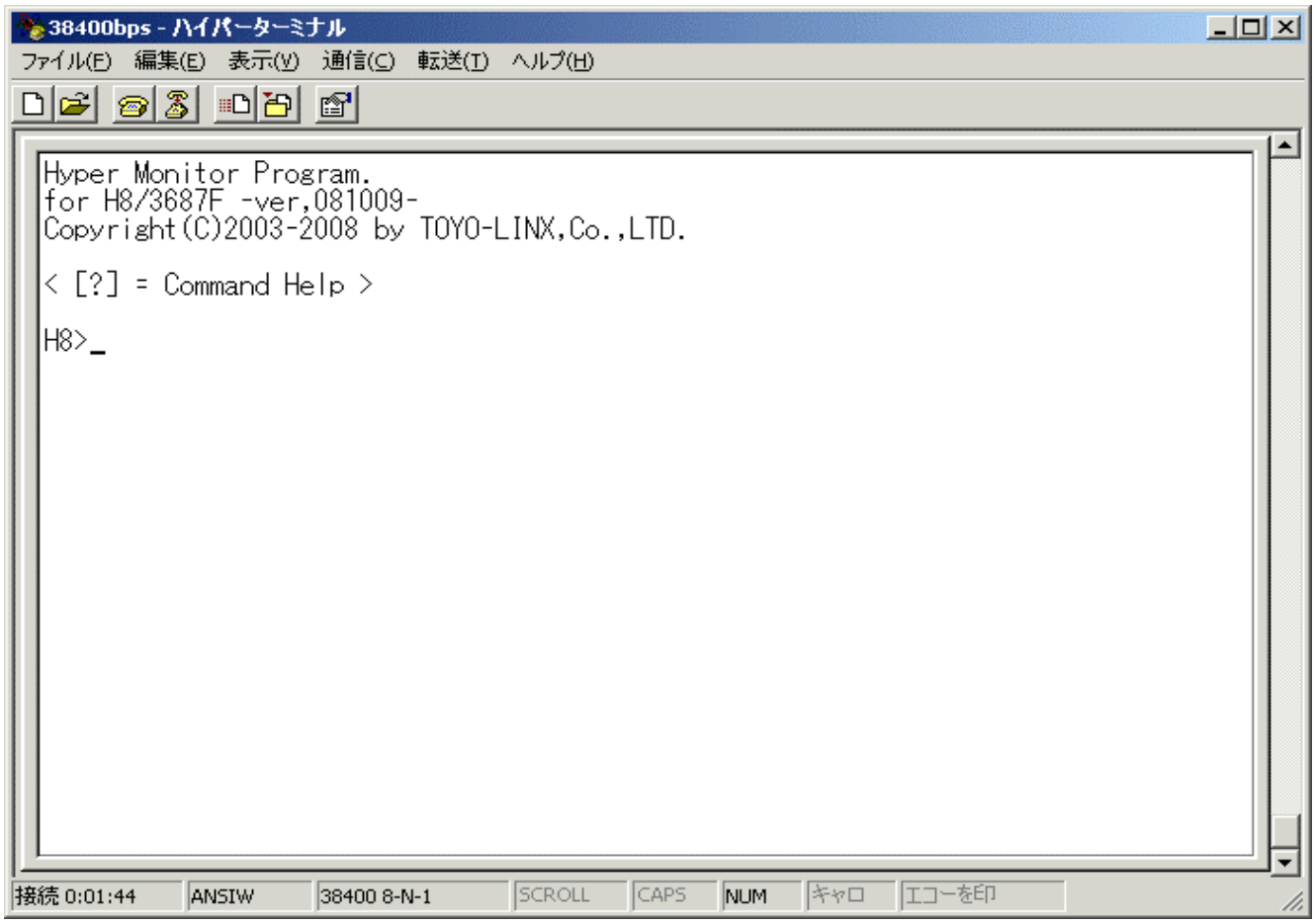
4. 電源オン!!

AC アダプタを TK-3687 の CN11 につなぎます(右写真参照)。使うことができる AC アダプタは,

6～12V, 0.3A 以上, 2.1 φ

です。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-3687 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。

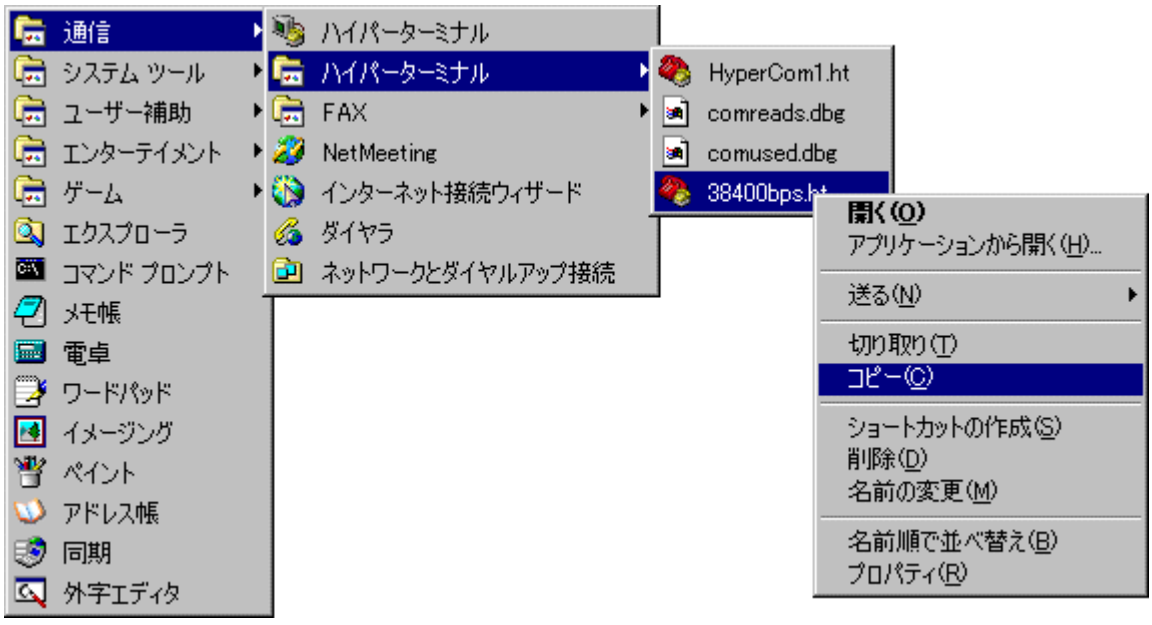
でも, その前に…(次のページを見てください)

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。

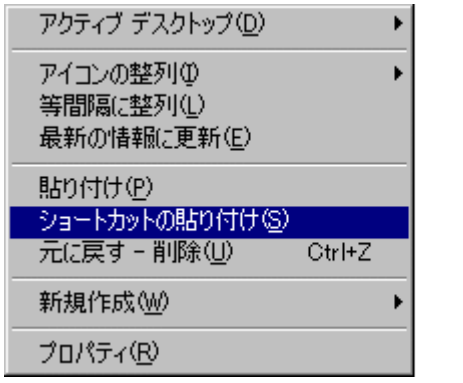
さらに，この設定のハイパーターミナルをすぐに呼び出せるように，デスクトップにショートカットを作成しましょう。スタートメニューから，

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。

第3章 プログラムを動かしてみよう

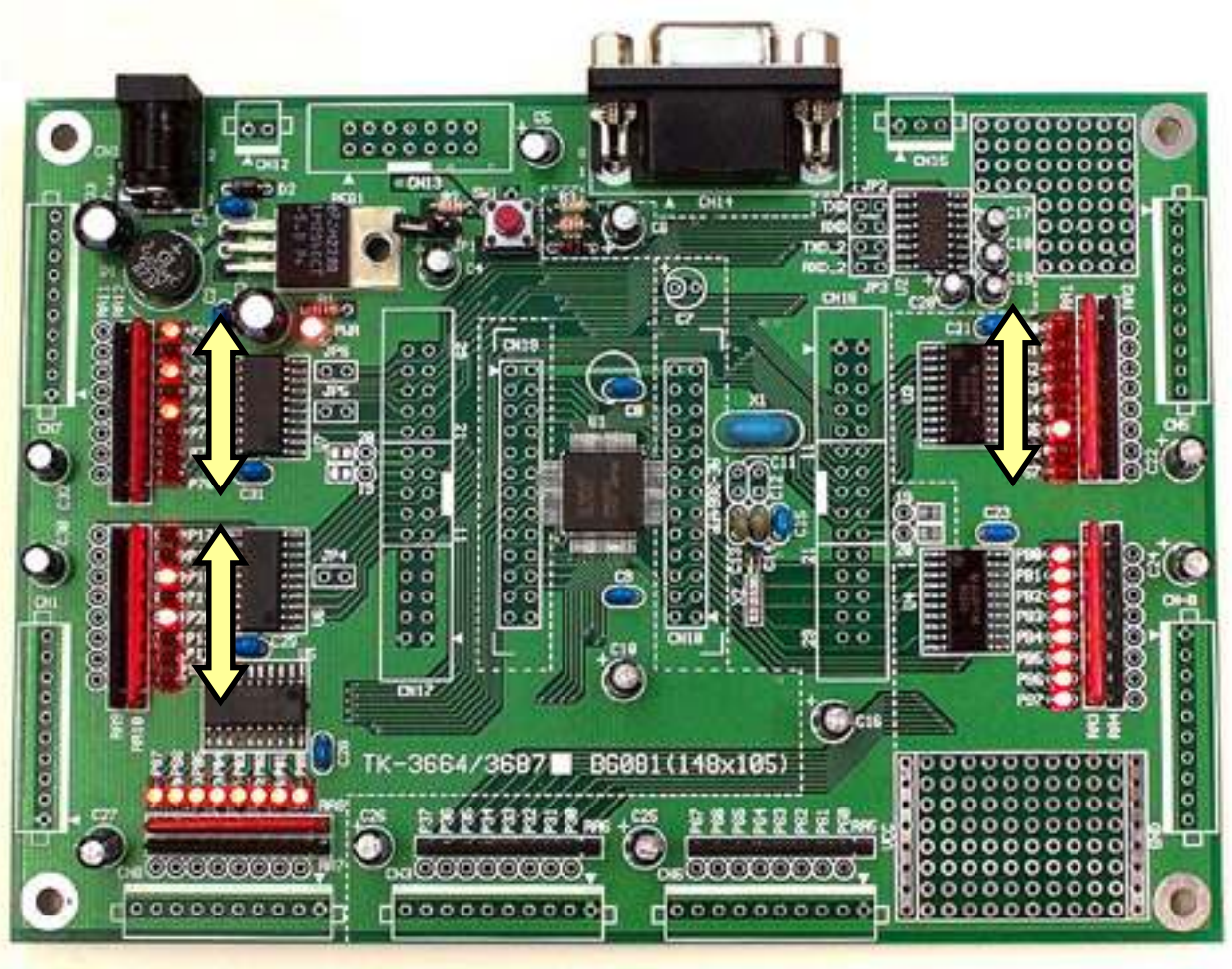
1. プログラムの実行

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-3687 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-3687 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。

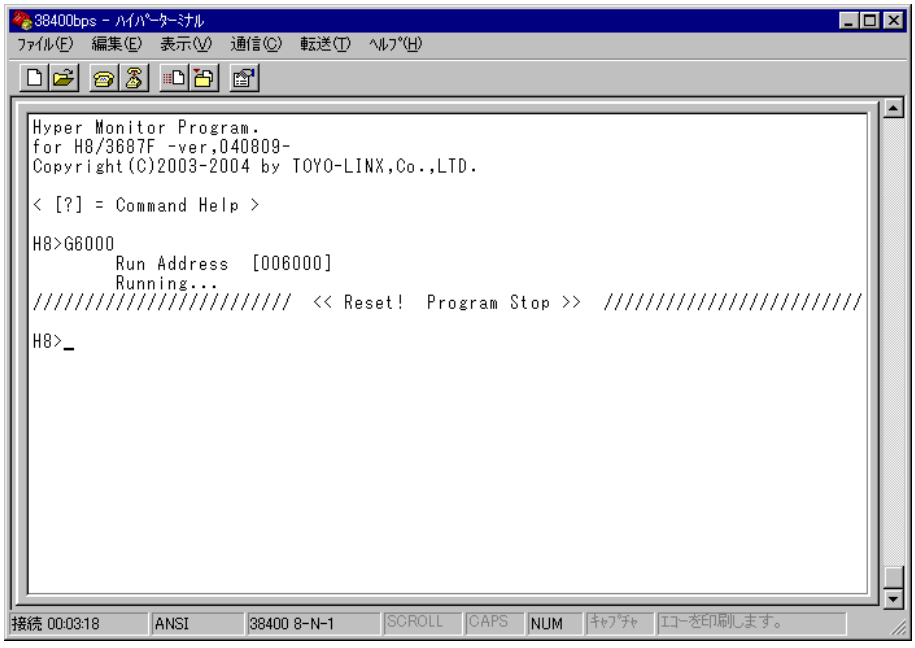
1. プログラムの実行

TK-3687 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。

P10～17, P50～57, P70～76 の LED が順番に点滅します。ただし, P20, P23, P24 は JP4, JP5, JP6 をつながないと点灯したままです。



TK-3687 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 は右図のように入力待ちの状態になります。



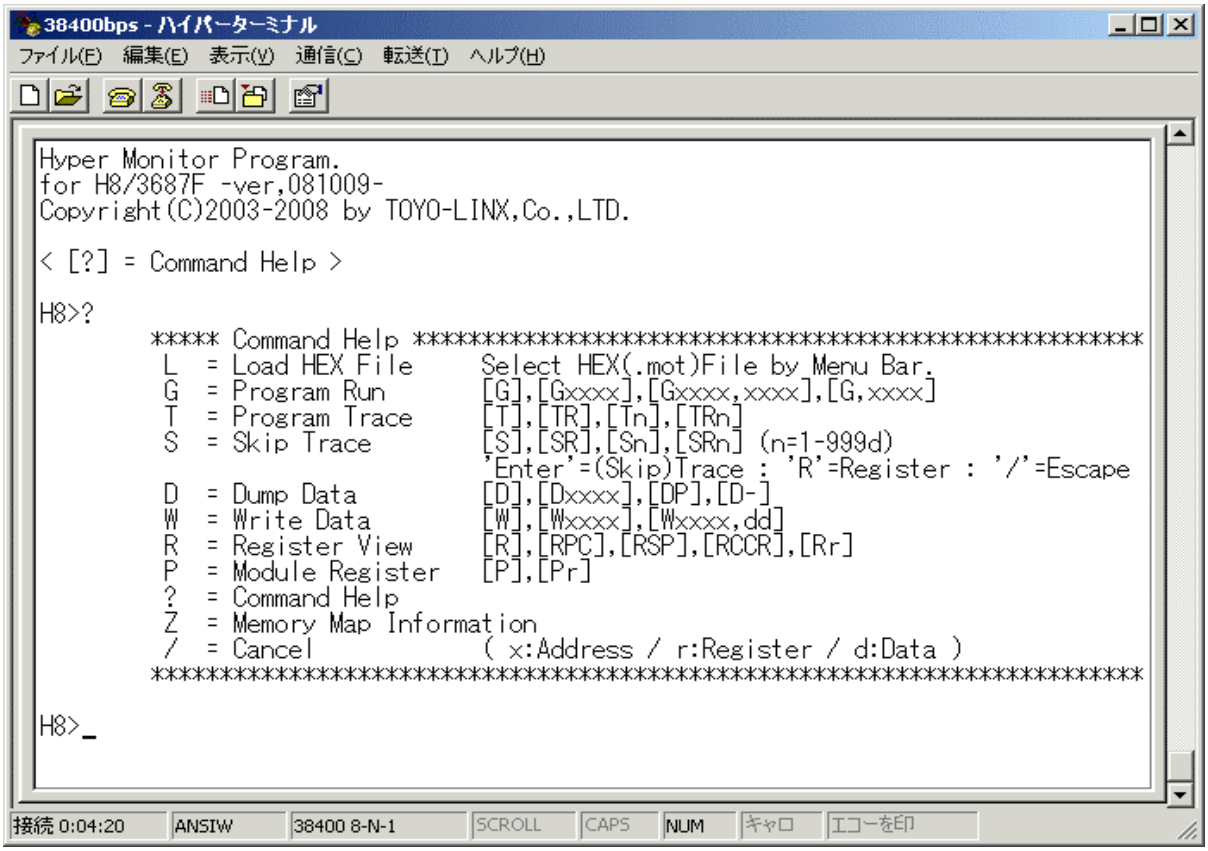
さて, 今の操作は,

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‘E7’というエミュレータを購入して使うことになります。

第4章 プログラムを作ってみよう

1. プログラムを作ろう
2. マシン語への変換
3. プログラムの入力
4. プログラムのトレース実行

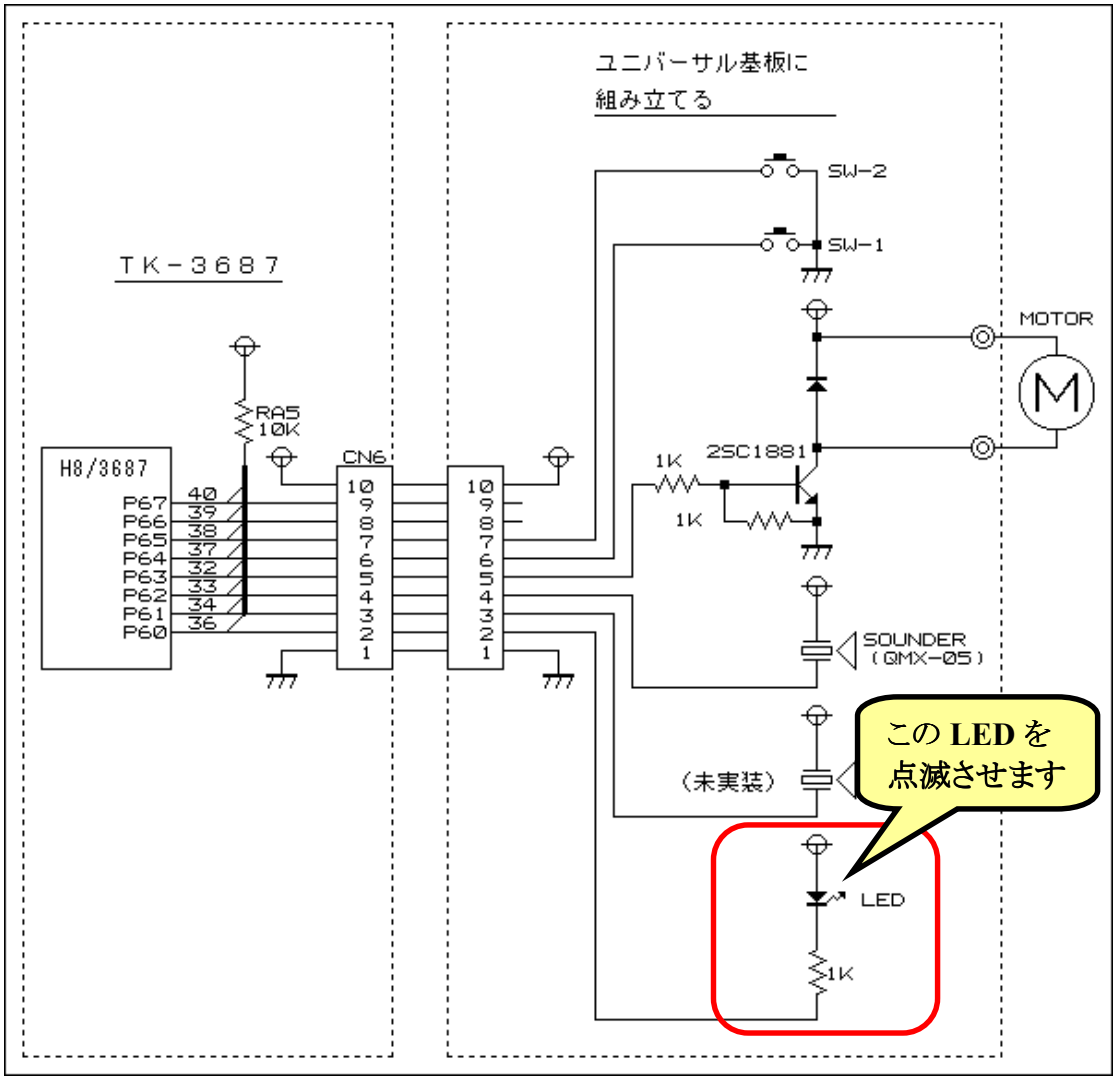
プログラムの作り方を理解するための最も早道は何でしょうか。それは，とにかくたくさん作ってみる，ということです。くりかえし作っているうちに段々プログラミングの考え方が身についてきます。

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-3687 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」をお読みください。また，このマニュアルの付録で，もっと詳しい説明をしています。興味のある方はお読みください。

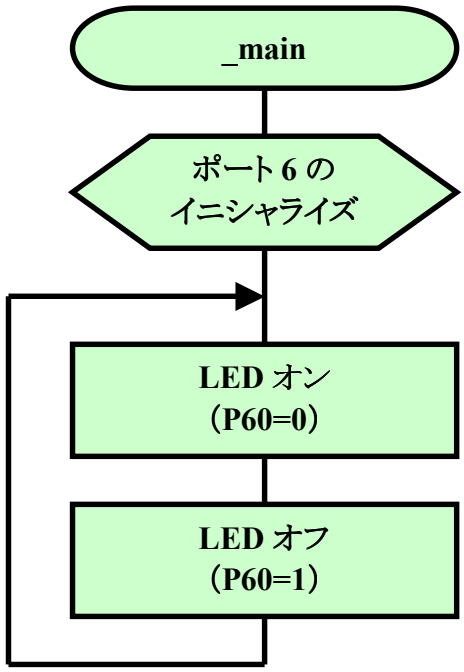
同時に，マイコンがどのようにプログラムを動かしているのか、ちょっとだけのぞいてみましょう。

1. プログラムを作ろう

ここで作るプログラムは，TK-3687 の P60 に接続した LED を点滅させるというものです。次のような回路で考えてみましょう。（くわしい組み立てについては第 7 章をご覧ください。）



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。



さて次は，今考えたフローチャートを見ながら H8/3687 の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，マイコンの言葉に直す必要があります。これについては「2．マシン語への変換」で説明します。

フローチャートからコーディングする方法に興味のある方は，付録の「コーディングの方法」をご覧下さい。

2. マシン語への変換

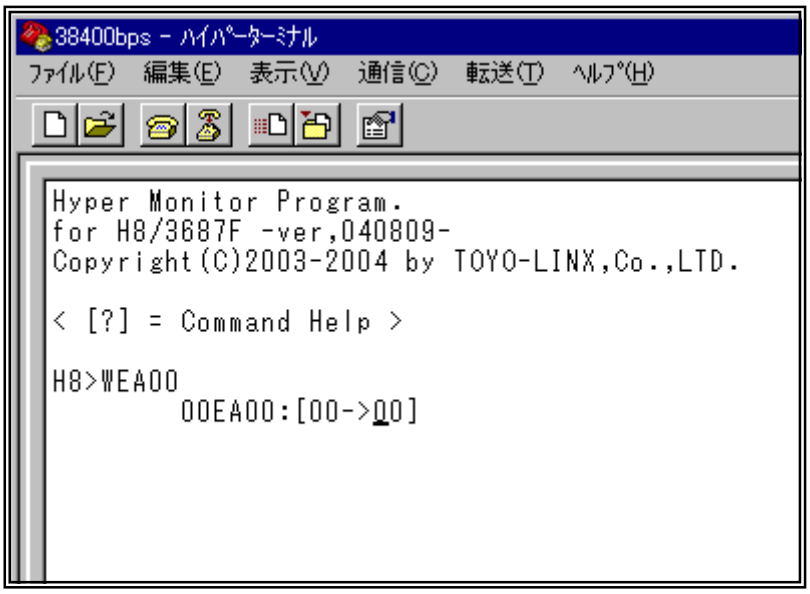
コーディングが終了したものの，このままではマイコン（H8/3687）は何をしたら良いのか理解できません。マイコンが理解できるのはマシン語と呼ばれる 16 進の数字の羅列だけです。そこで，次はマシン語への変換作業を行ないます。これをアセンブルと呼びます。マシン語に変換すると次のようになります。これをハイパーH8 で RAM に入力していきます。

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,R0L	ポート6のイニシャライズ
EA01	01				
EA02	38		MOV.B	R0L,@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					

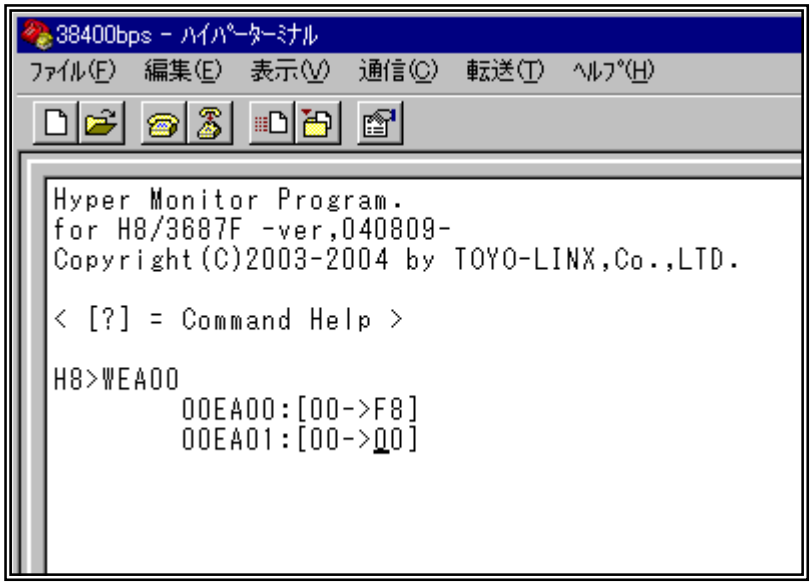
マシン語への変換に興味のある方は，付録の「ハンドアセンブルの方法」をご覧下さい。

3. プログラムの入力

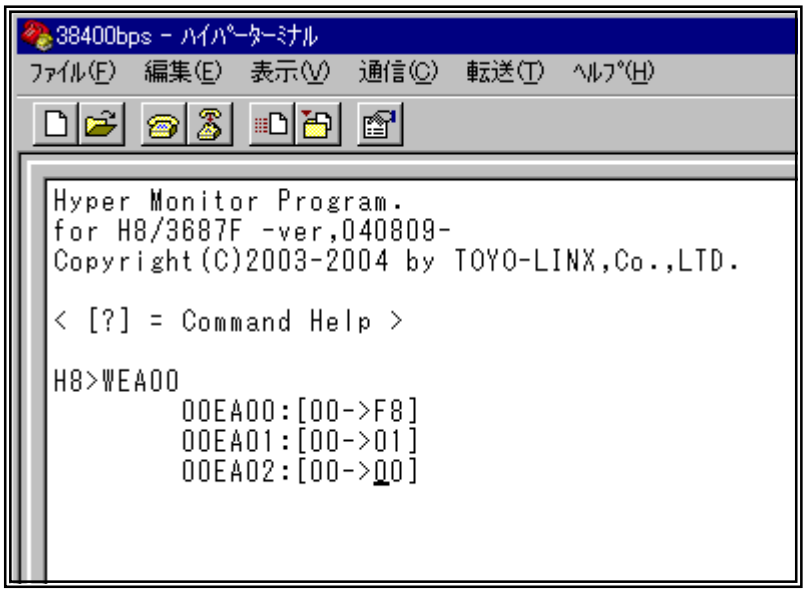
それでは，ハイパーH8 を使って TK-3687 のメモリにマシン語を入力していきましょう。‘W’コマンドを使います。‘EA00’番地から入力しますので，パソコンのキーボードから‘WEA00’と入力して‘Enter’キーを押します。入力状態になったら 1 バイトづつ順番にデータを入力します。



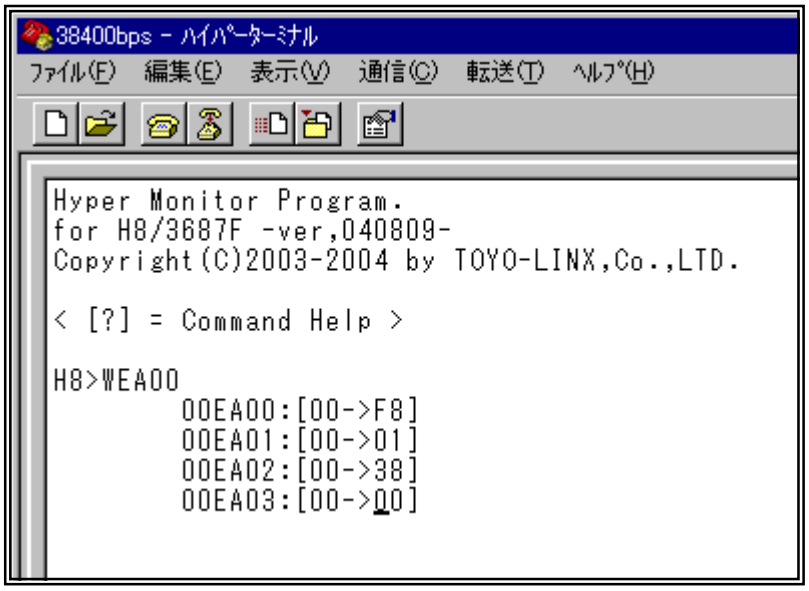
キーボード：‘F’，‘8’，‘Enter’



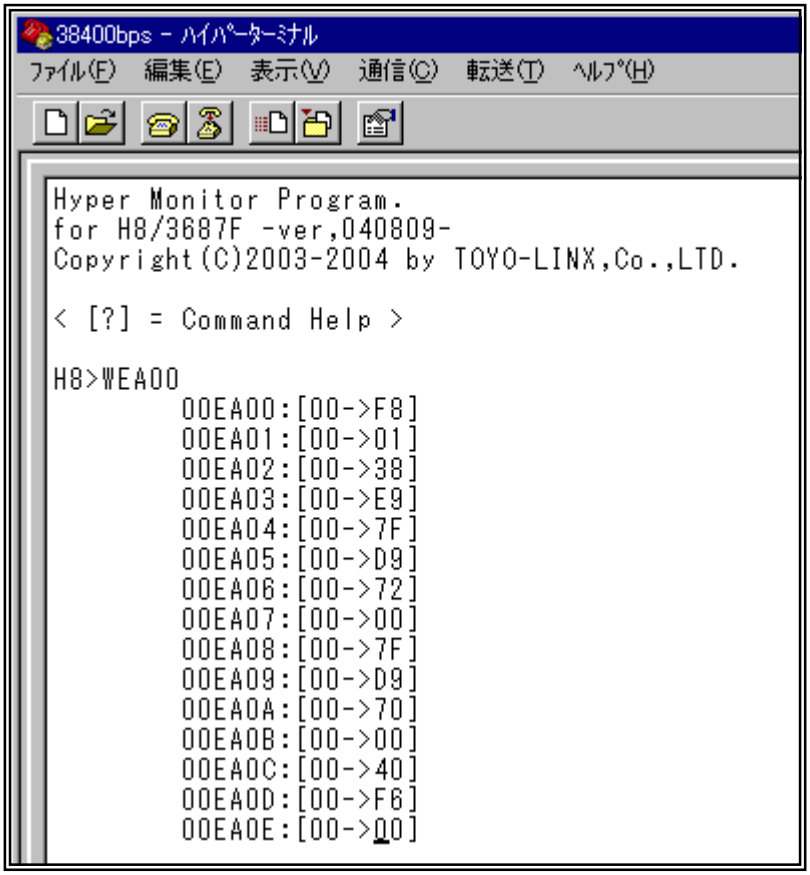
キーボード：‘0’，‘1’，‘Enter’



キーボード：‘3’，‘8’，‘Enter’

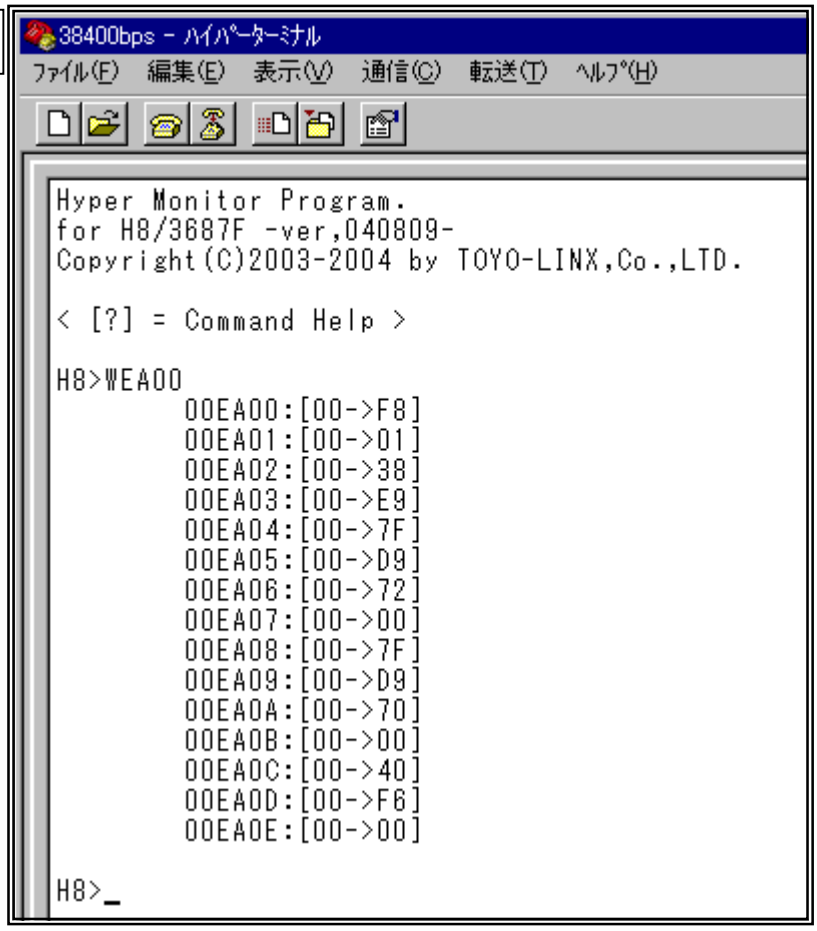


キーボード：‘F’，‘6’，‘Enter’

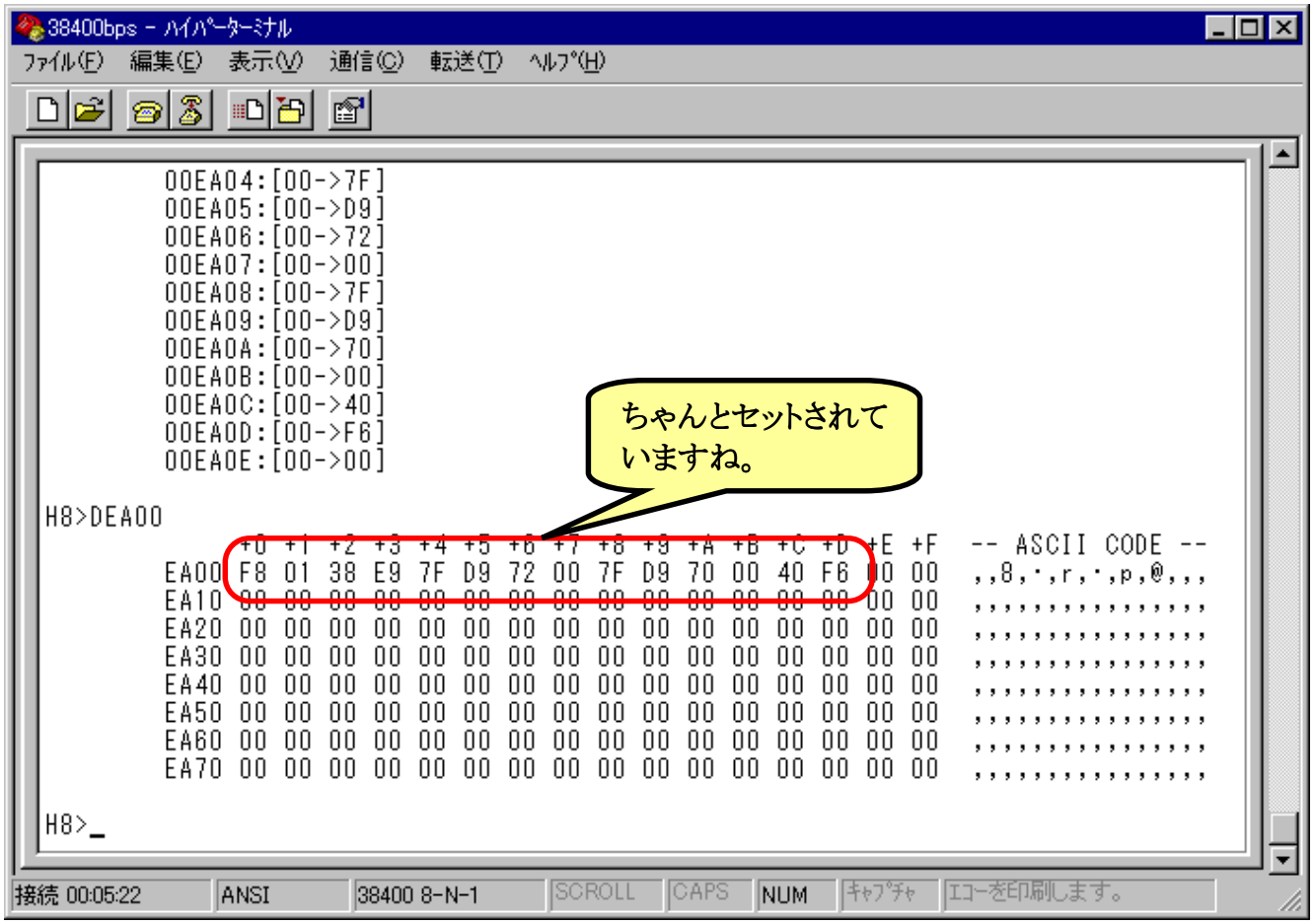


全てのデータを入力したら，キーボードから‘/’を押してコマンド入力に戻ります。

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

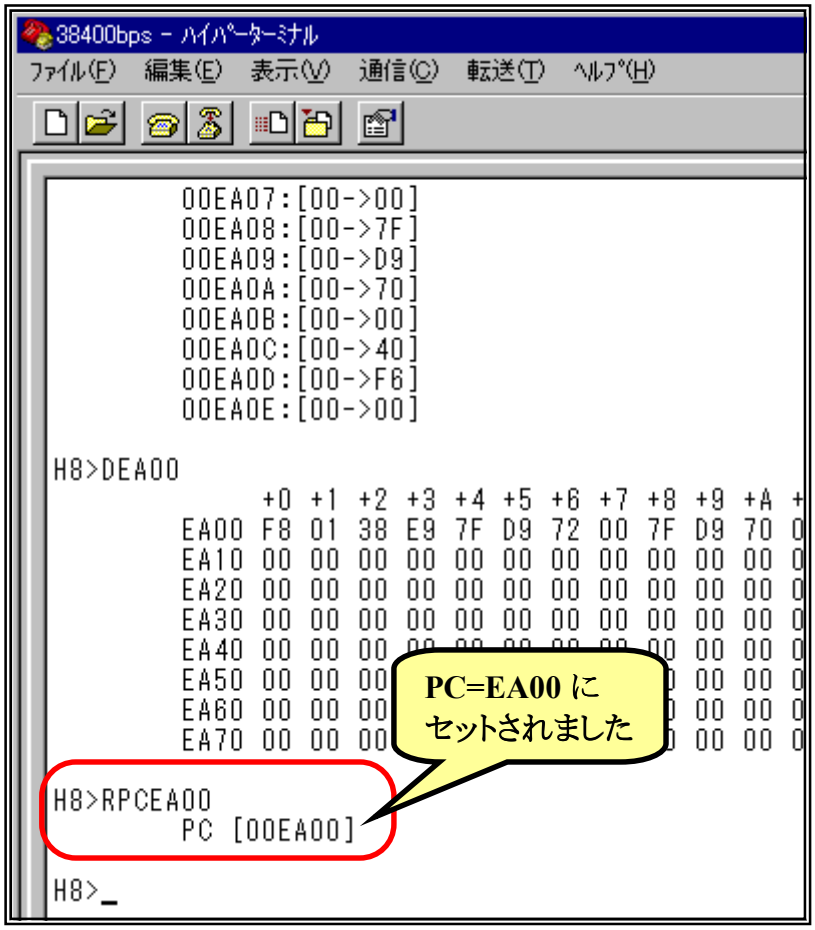
という意味です。

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。

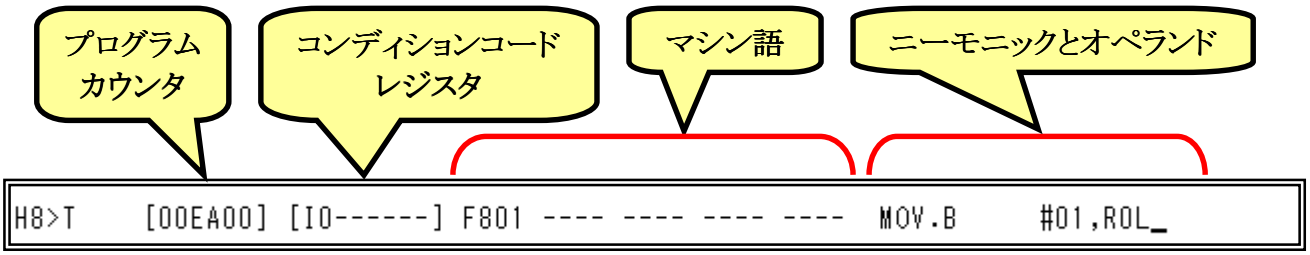
4. プログラムのトレース実行

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。

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。

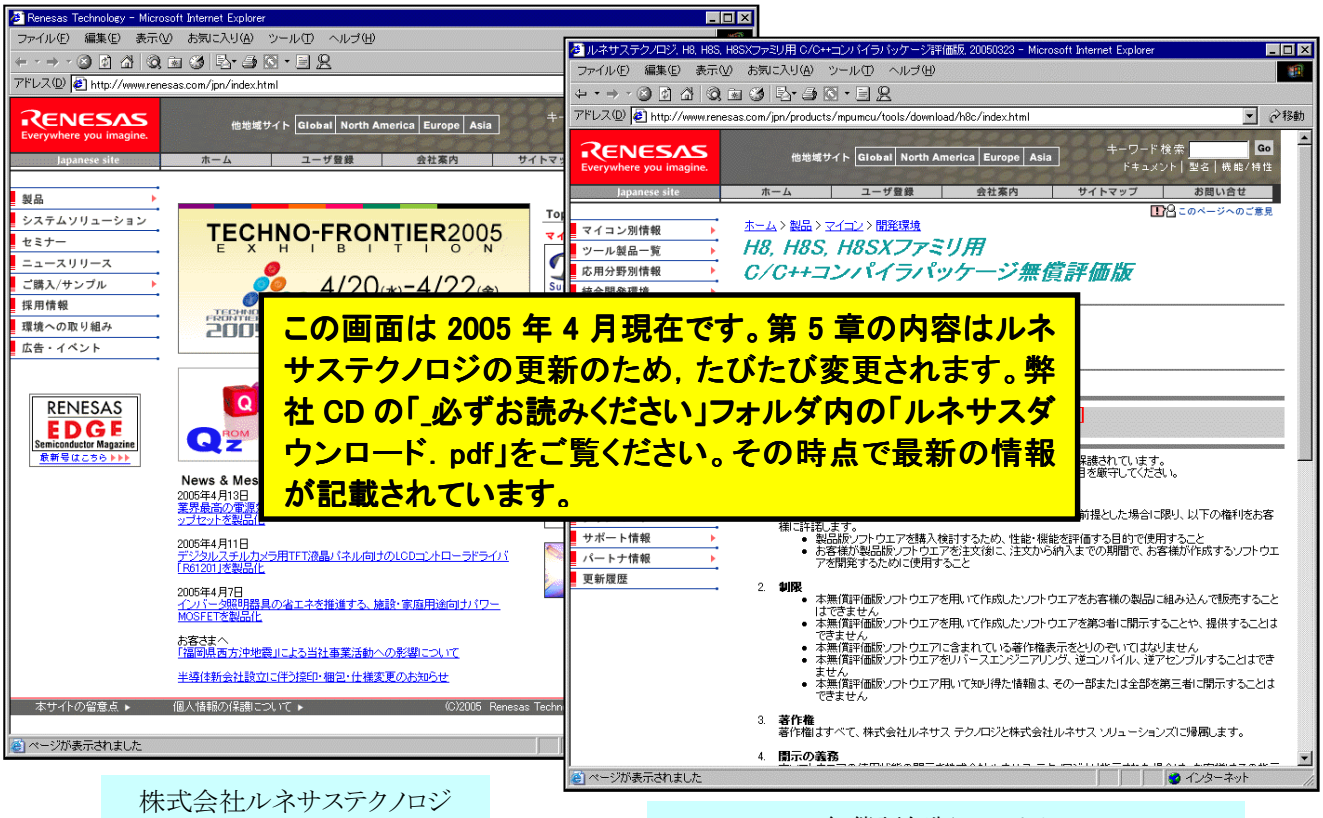
第5章 開発環境を手に入れよう

1. HEW の入手
2. HEW のインストール

前の章では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。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。命令をマシン語に変換するプログラム，アセンブラを使うのがスマートな方法です。

1. HEW の入手

TK-3687mini のプログラミングで使用するアセンブラは，High-performance Embedded Workshop V.4（HEW4）に対応した無償評価版コンパイラに含まれており，株式会社ルネサステクノロジのホームページよりダウンロードします。ダウンロードサイトの URL は以下の通りです。

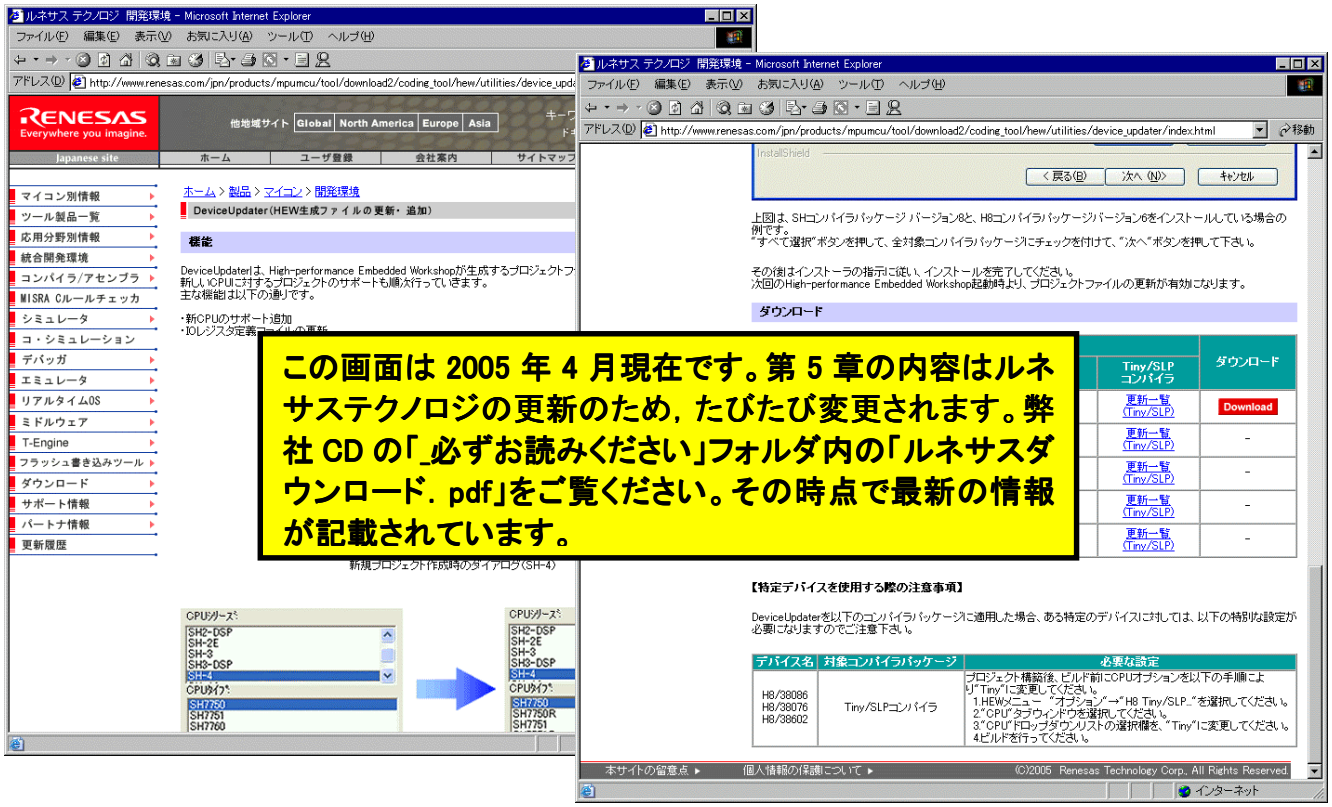


株式会社ルネサステクノロジ
http://www.renesas.com/jpn/

無償評価版コンパイラ
ダウンロードサイト
http://www.renesas.com/jpn/products/mpumcu/tools/download/h8c/index.html

ダウンロードサイトの下の方にある「ダウンロードのページへ」をクリックして下さい。次のページで必須事項を入力してダウンロードを開始します。ダウンロード先はデスクトップにすると便利です。全部で 69．4MByte になりますので，ADSL か光回線でないと，かなり大変なのが実情です。‘h8cv601r00．exe’というファイルがダウンロードされます。

ところで，ここでダウンロードした無償評価版コンパイラには不具合があることが報告されています。それで，ルネサステクノロジが公開しているデバイスアップデータを使用して不具合を修正します。デバイスアップデータは下記の URL のサイトからダウンロードできます。



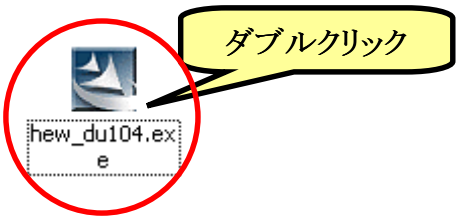
デバイスアップデータ
ダウンロードサイト
http://www.renesas.com/jpn/products/mpumcu/tool/download2/coding_tool/hew/utilities/device_updata/index.html

ページの下の方にある「Download」をクリックしてください。ダウンロード先はデスクトップにすると便利です。全部で 3．55MByte になります。‘hew_du104．exe’というファイルがダウンロードされます。

2. 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‘h8cv601r00. 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。

次に，無償評価版コンパイラをアップデートします。ダウンロードした‘hew_du104. 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，アセンブラどころか 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。

なお，「マイコン事始め」で説明している HEW はこの章で入手した無償評価版です。有償版の HEW やエミュレータ‘E7’に付属している HEW の場合はちがうバージョンの可能性もあるため画面が多少異なるかもしれません。もちろん基本的な操作方法や考え方は同じです。

最新版の HEW を手に入れましょう

HEW はときどきバージョンアップされます。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。

第6章 アセンブラでプログラムを作ってみよう

1. メモリマップの確認
2. プロジェクトの作成
3. プログラムの入力
4. ビルド!!
5. ダウンロードとトレース実行

この章では，第 4 章でハンドアセンブルしたプログラムをアセンブラで作り，HEW のアセンブラとしての使い方を覚えましょう。

1. メモリマップの確認

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 colspan="3">モニタプログラム
‘ハイパーH8’</td><td>ROM/フラッシュメモリ
（56K バイト）</td></tr>
<tr><td>E000 番地
E7FF 番地</td><td colspan="3">未使用</td><td>未使用</td></tr>
<tr><td>E800 番地</td><td>VECTTBL
INTTBL</td><td>ベクタテーブル</td><td rowspan="3">ユーザ RAM エリア</td><td rowspan="3">RAM
（2K バイト）</td></tr>
<tr><td>E860 番地</td><td>ResetPRG
IntPRG</td><td>リセットプログラム
割り込みプログラム</td></tr>
<tr><td>EA00 番地

EFFF 番地</td><td>P</td><td>プログラム領域</td></tr>
<tr><td>F000 番地
F6FF 番地</td><td colspan="3">未使用</td><td>未使用</td></tr>
<tr><td>F700 番地
F77F 番地</td><td colspan="3">I/O レジスタ</td><td>I/O レジスタ</td></tr>
<tr><td>F780 番地

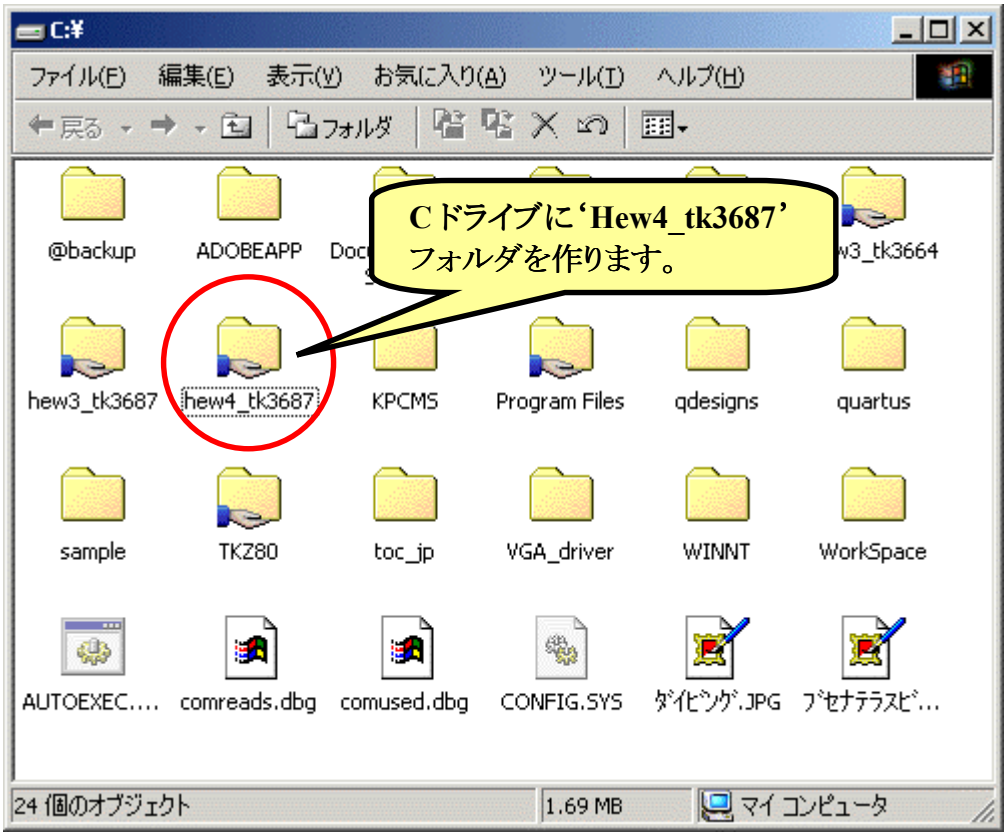
FB7F 番地</td><td rowspan="2">B</td><td rowspan="2">未初期化データ領域
（変数領域）</td><td rowspan="3">ユーザ RAM エリア</td><td>RAM（1K バイト）
フラッシュメモリ書換え用
ワークエリアのため，
FDT と E7 使用時は，
ユーザ使用不可</td></tr>
<tr><td>FB80 番地</td><td rowspan="3">RAM
（1K バイト）</td></tr>
<tr><td>FD80 番地
FDFF 番地</td><td>Stack</td><td>スタック領域</td></tr>
<tr><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 colspan="3">I/O レジスタ</td><td>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。

2. 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‘led’と名付けます。以下に，新規プロジェクト‘led’を作成する手順と動作確認の手順を説明します。

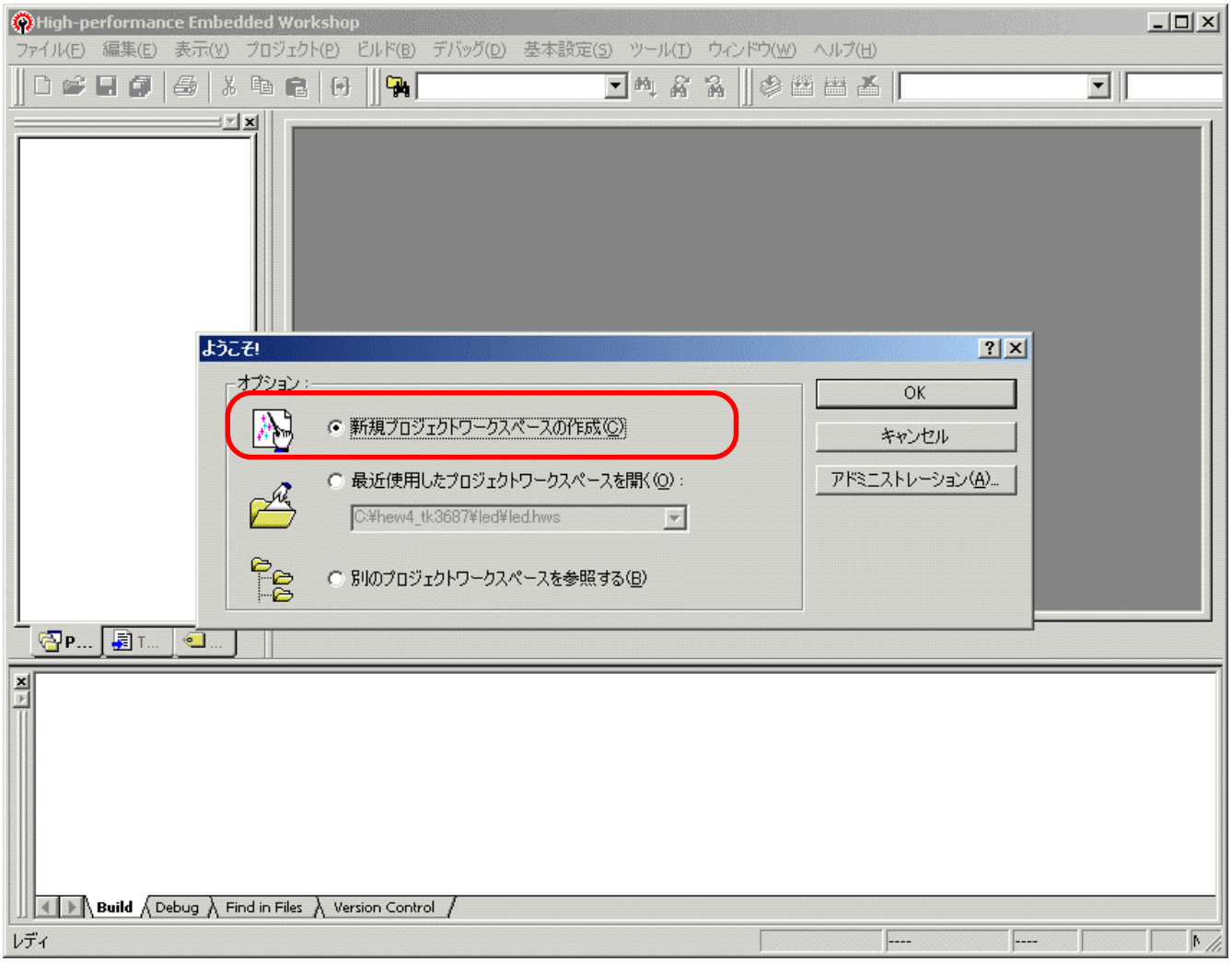
しかしその前に，HEW 専用作業フォルダを作っておきましょう。C ドライブに‘Hew4_tk3687’を作ってください。このマニュアルのプロジェクトは全てこのフォルダに作成します。



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

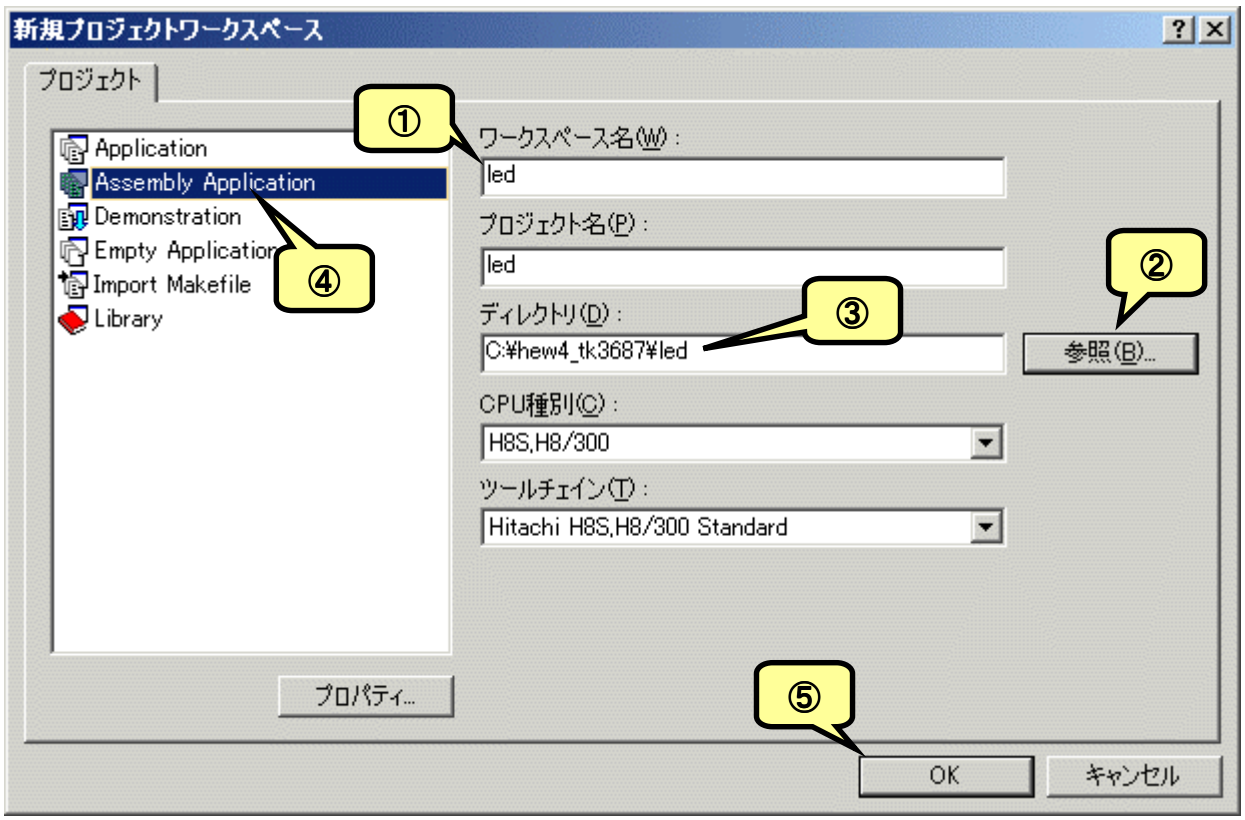
その場合は，「ようこそ!」ダイアログで「最近使用したプロジェクトワークスペースを開く」を選択して‘OK’をクリックします。そのプロジェクトの最後に保存した状態で HEW が起動します。

まず，①「ワークスペース名（W）」（ここでは‘led’）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。

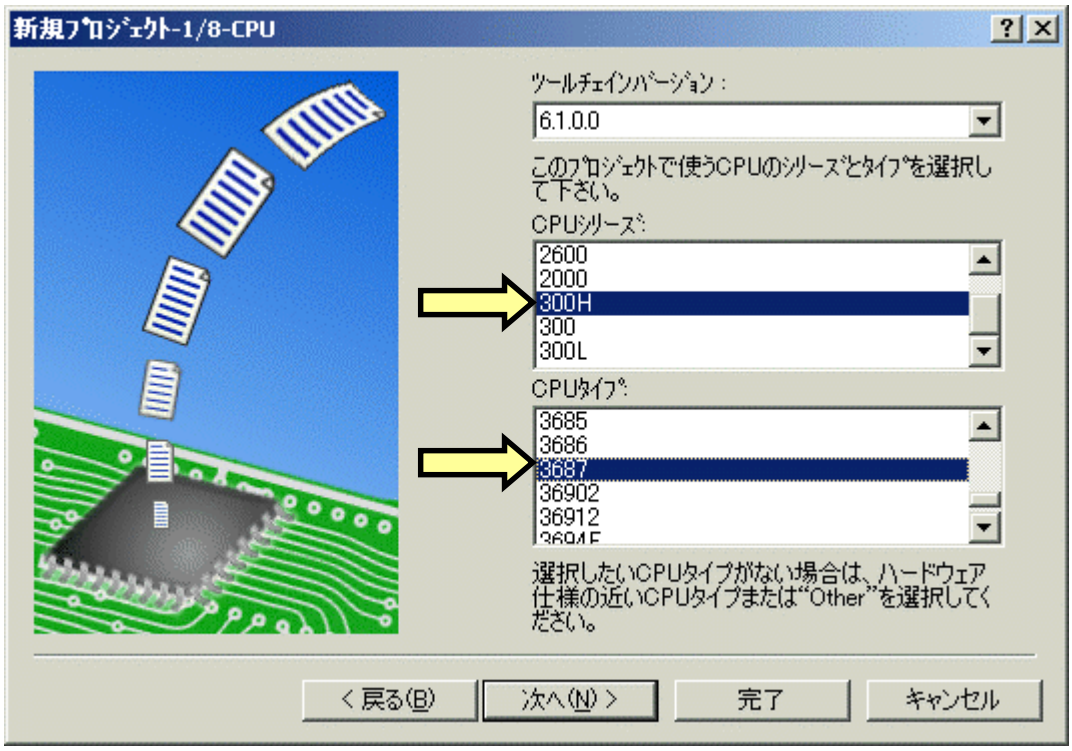
ワークスペースの場所を指定します。②右の「参照（B）…」ボタンをクリックします。そして，あらかじめ用意した HEW 専用作業フォルダ（ここでは Hew4_tk3687）を指定します。設定後，「ディレクトリ（D）」が正しいか確認して下さい。（③）

次にプロジェクトタイプを指定します。今回はアセンブラなので④「Assembly Application」を選択します。

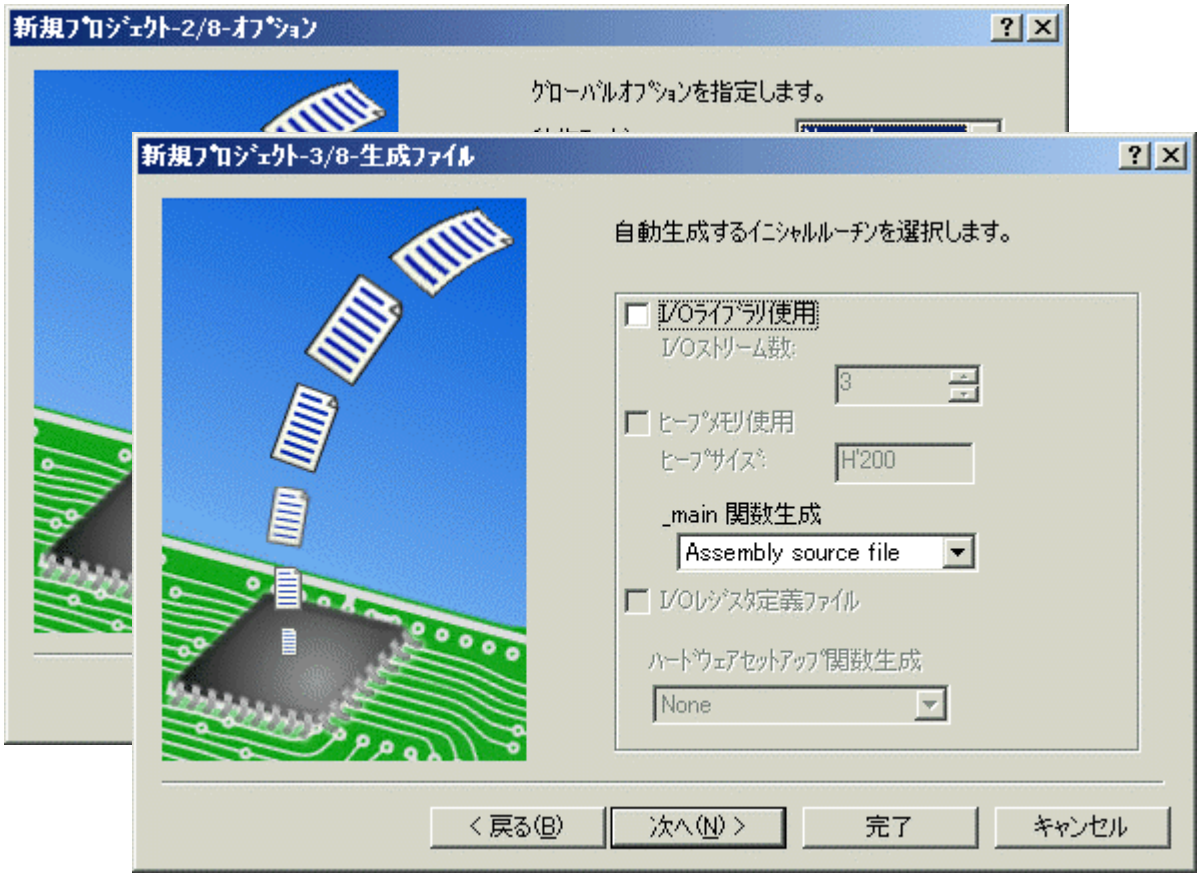
入力が終わったら⑤「OK」をクリックして下さい。



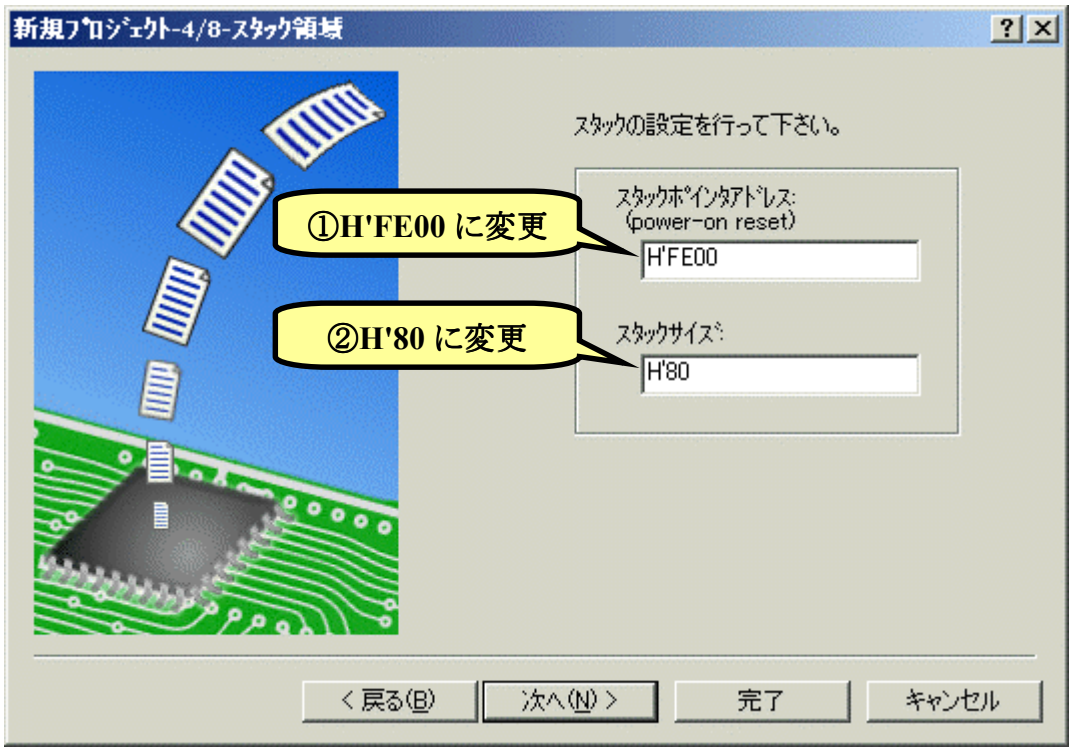
「新規プロジェクトー1／8－CPU」で，使用する CPU シリーズ（300H）と，CPU タイプ（3687）を設定し，「次へ（N）＞」をクリックします。



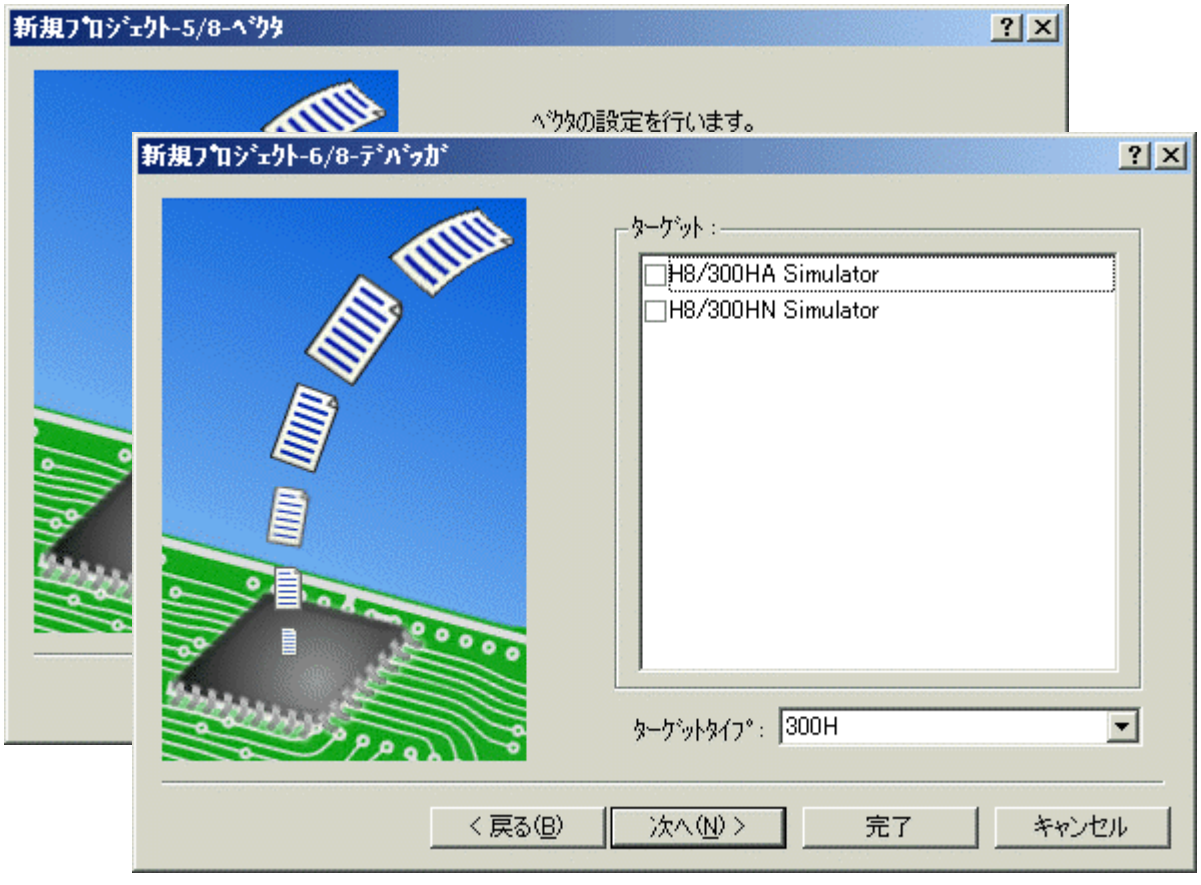
「新規プロジェクトー2／8－オプション」と「新規プロジェクトー3／8－生成ファイル」は変更しません。「次へ（N）＞」をクリックして順に次の画面に進みます。



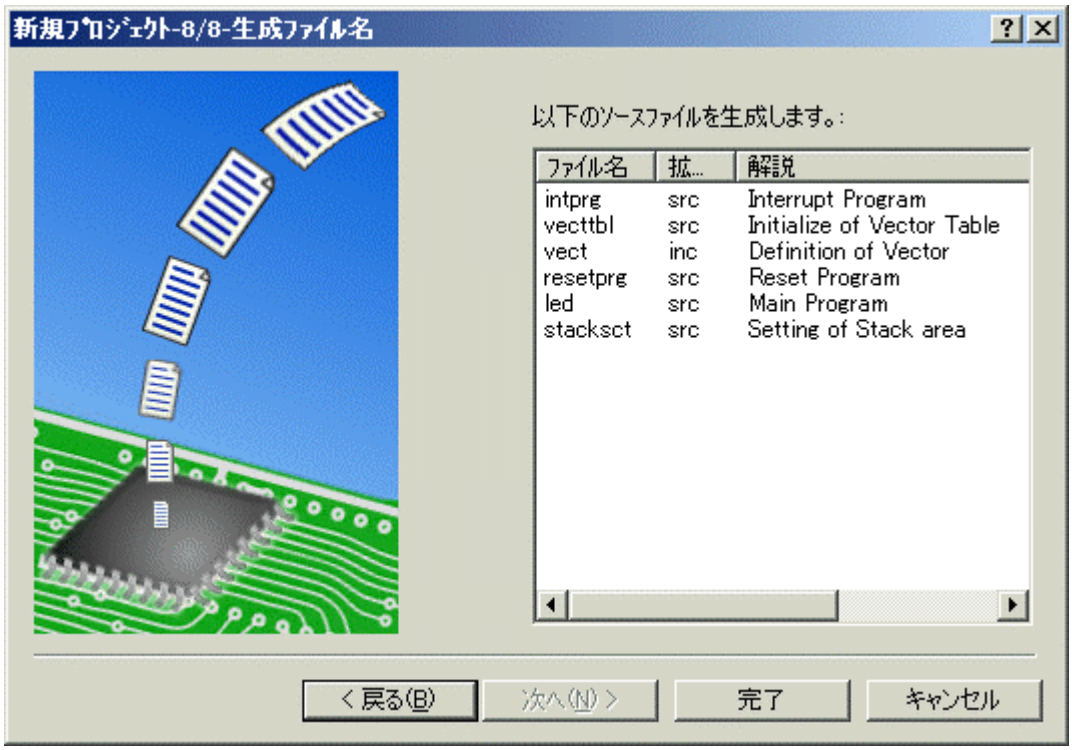
「新規プロジェクトの作成－4／8－スタック領域」でスタックのアドレスとサイズを変更します。ハイパーH8 を使用するので，①スタックポインタを H'FE00 に，②スタックサイズを H'80 にします。設定が終わったら「次へ(N) >」をクリックします。（ハイパーH8 を使わないときは変更の必要はありません。）



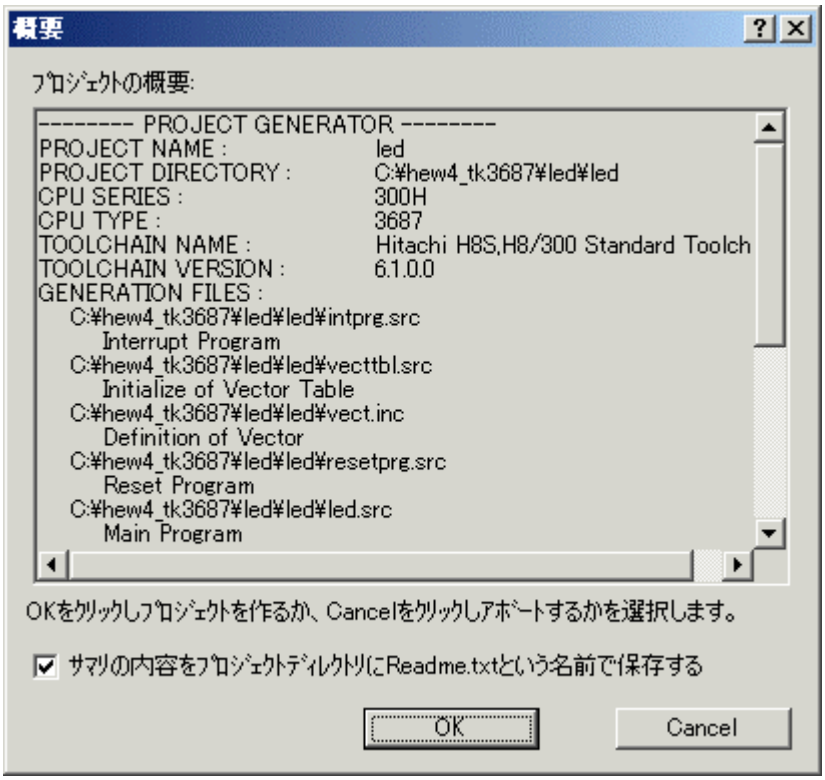
「新規プロジェクト－5／8－ベクタ」と「新規プロジェクト－6／8－デバッガ」は変更しません。「次へ(N) >」をクリックして順に次の画面に進みます。



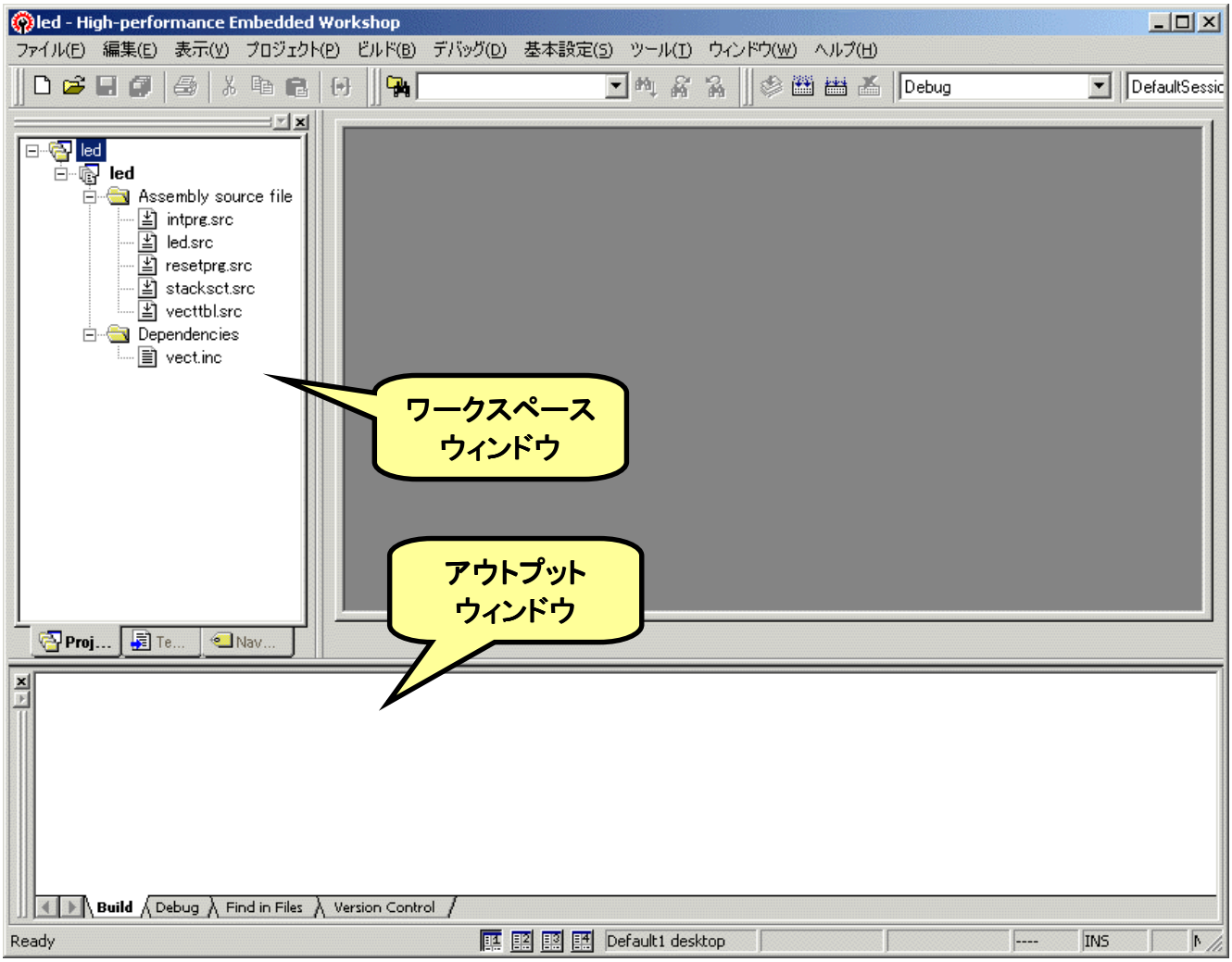
次は「新規プロジェクトー8／8ー生成ファイル名」です。ここも変更しません。「完了」をクリックします。



すると,「概要」が表示されるので「OK」をクリックします。

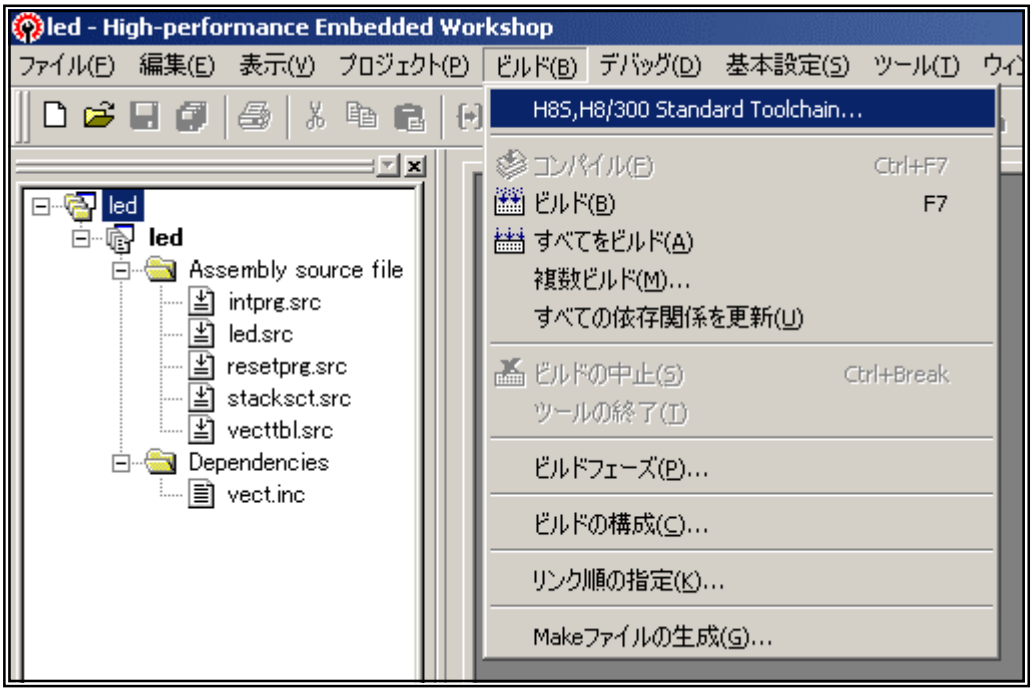


これで，プロジェクトワークスペースが完成します。HEW はプロジェクトに必要なファイルを自動生成し，それらのファイルは左端のワークスペースウィンドウに一覧表示されます。

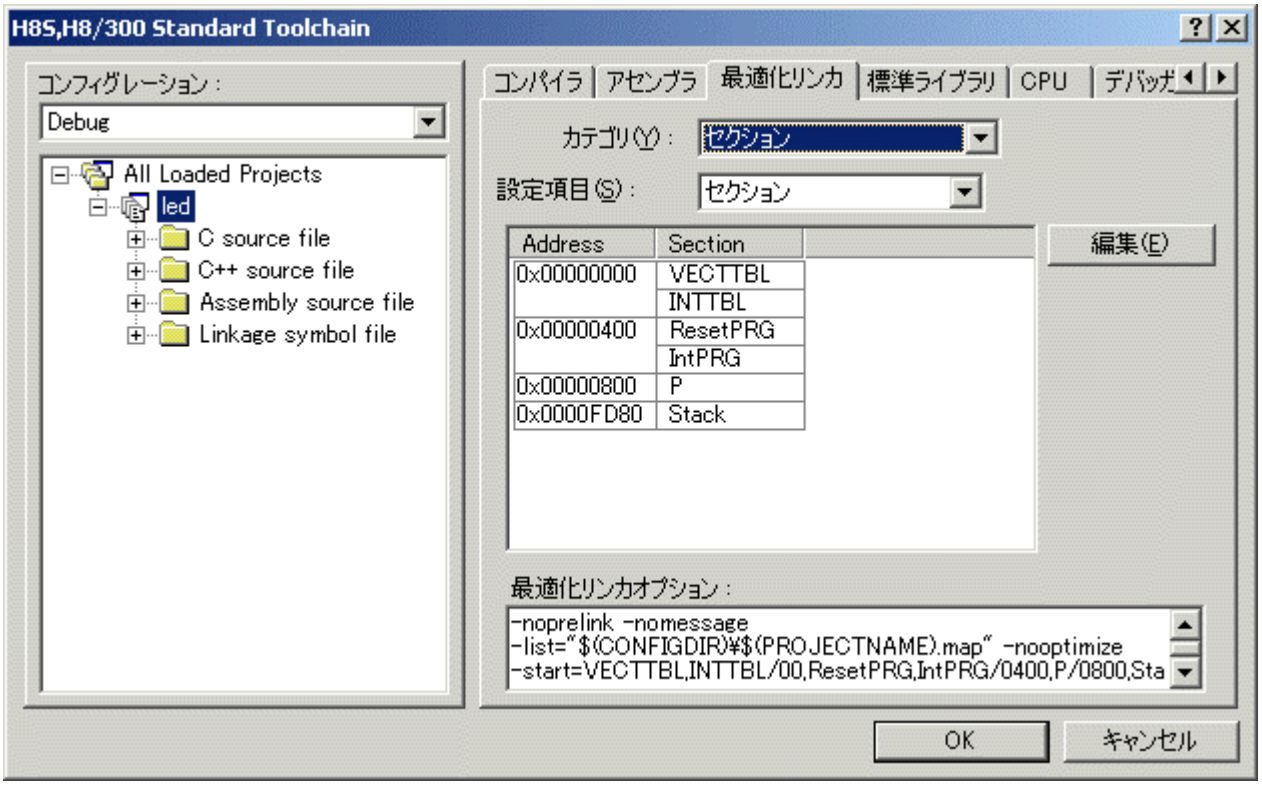


さて，これでプロジェクトは完成したのですが，ハイパーH8 を使うためにセクションを変更してプログラムが RAM 上にできるようにします。（当然ながら，ハイパーH8 を使わないときは変更する必要はなく，そのままで OK。）

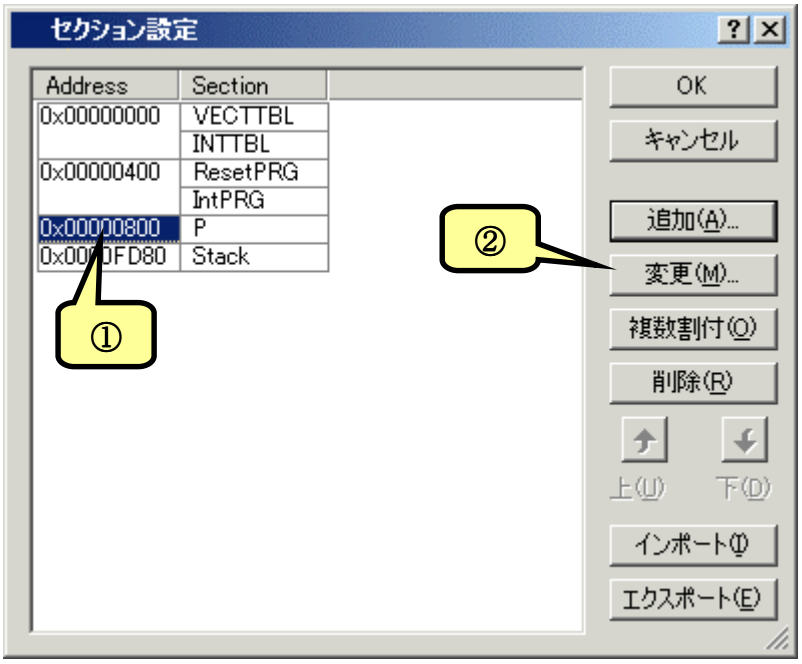
下図のように，メニューバーから「H8S, H8/300 Standard Toolchain…」を選びます。



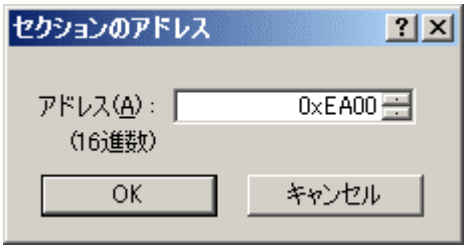
すると，「H8S, H8/300 Standard Toolchain」ウィンドウが開きます。「最適化リンカ」のタブを選び，「カテゴリ(Y)」のドロップダウンメニューの中から「セクション」を選択します。すると，下図のような各セクションの先頭アドレスを設定する画面になります。「編集(E)」ボタンをクリックしてください。



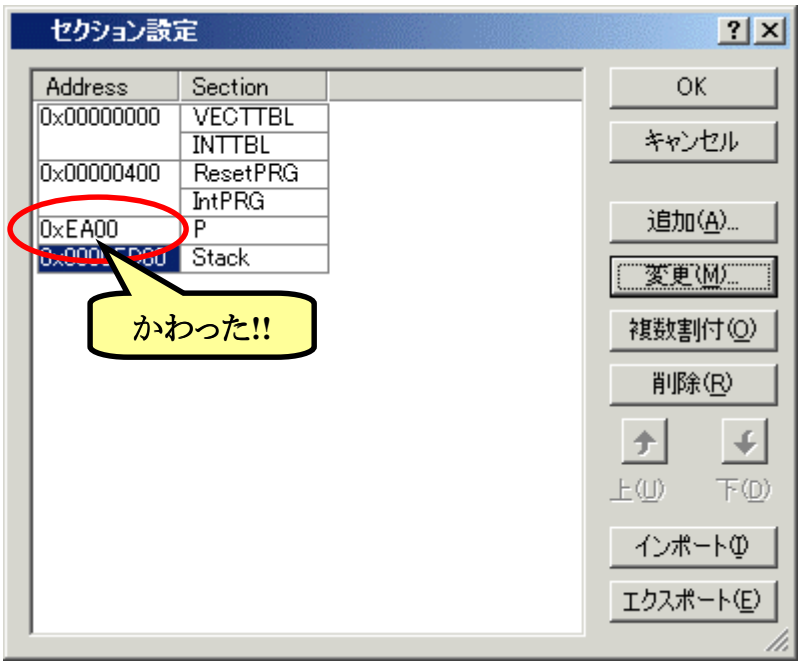
「セクション設定」ダイアログが開きます。それでは，「1. メモリマップの確認」で調べたメモリマップにあわせて設定していきましょう。最初に‘P’Section のアドレスを変更します。デフォルトでは 800 番地になっていますね。①‘0x00000800’というところをクリックして下さい。それから，②「変更(M)…」をクリックします。



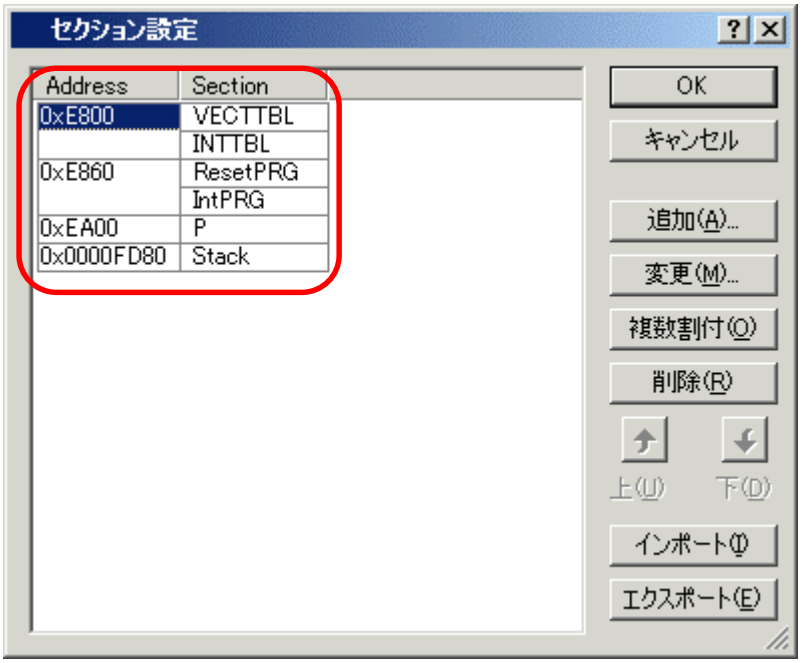
そうすると，「セクションのアドレス」ダイアログが開きます。‘P’Section は EA00 番地から始まりますので，右のように入力して‘OK’をクリックします。



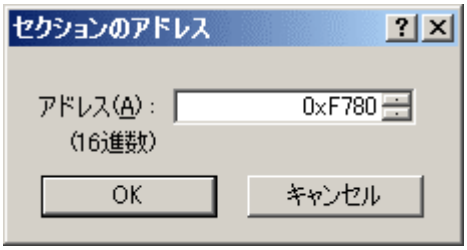
すると…



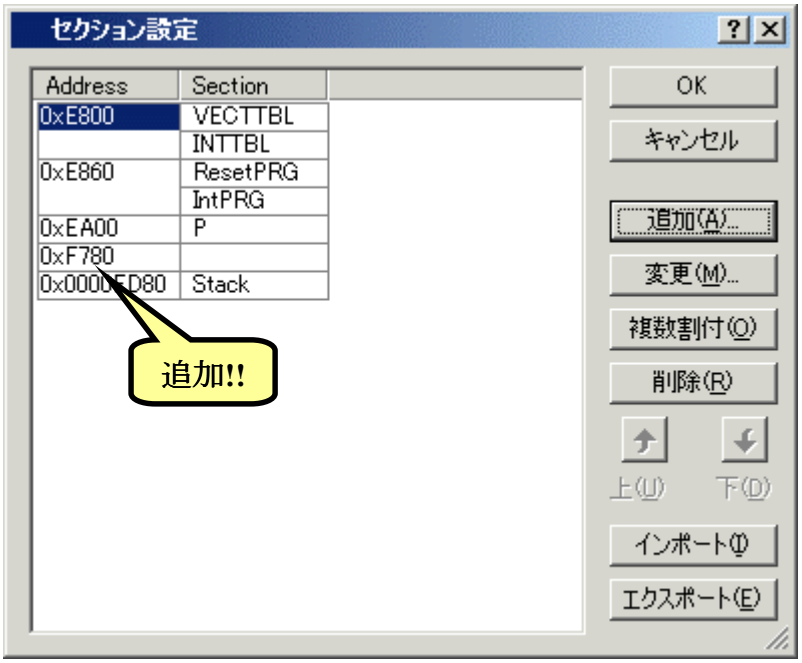
同じように，他のセクションも変更しましょう。



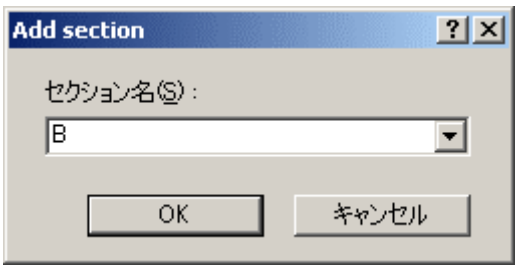
次は‘B’Section を追加します。どこでもかまわないのでアドレスをクリックして，「追加(A)…」をクリックして下さい。そうすると，「セクションのアドレス」ダイアログが開きます。‘B’Section は F780 番地から始まりますので，右のように入力して「OK」をクリックします。



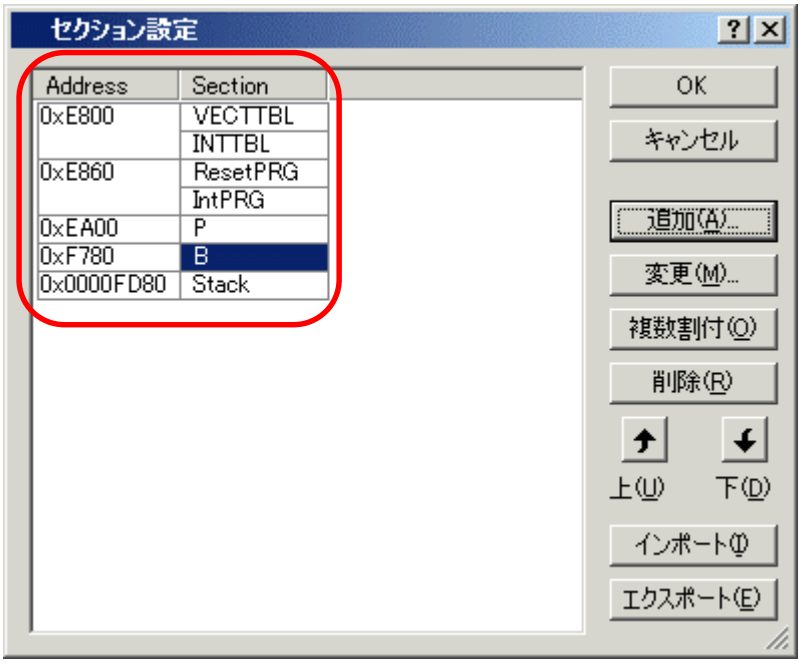
Address に 0xF780 が追加されました。



0xF780 番地の Section をクリックして，さらに‘Add…’をクリックして下さい。‘Add Section’ダイアログが開きます。‘Section name’のドロップダウンメニューの中から‘B’を選択し，「OK」をクリックします。

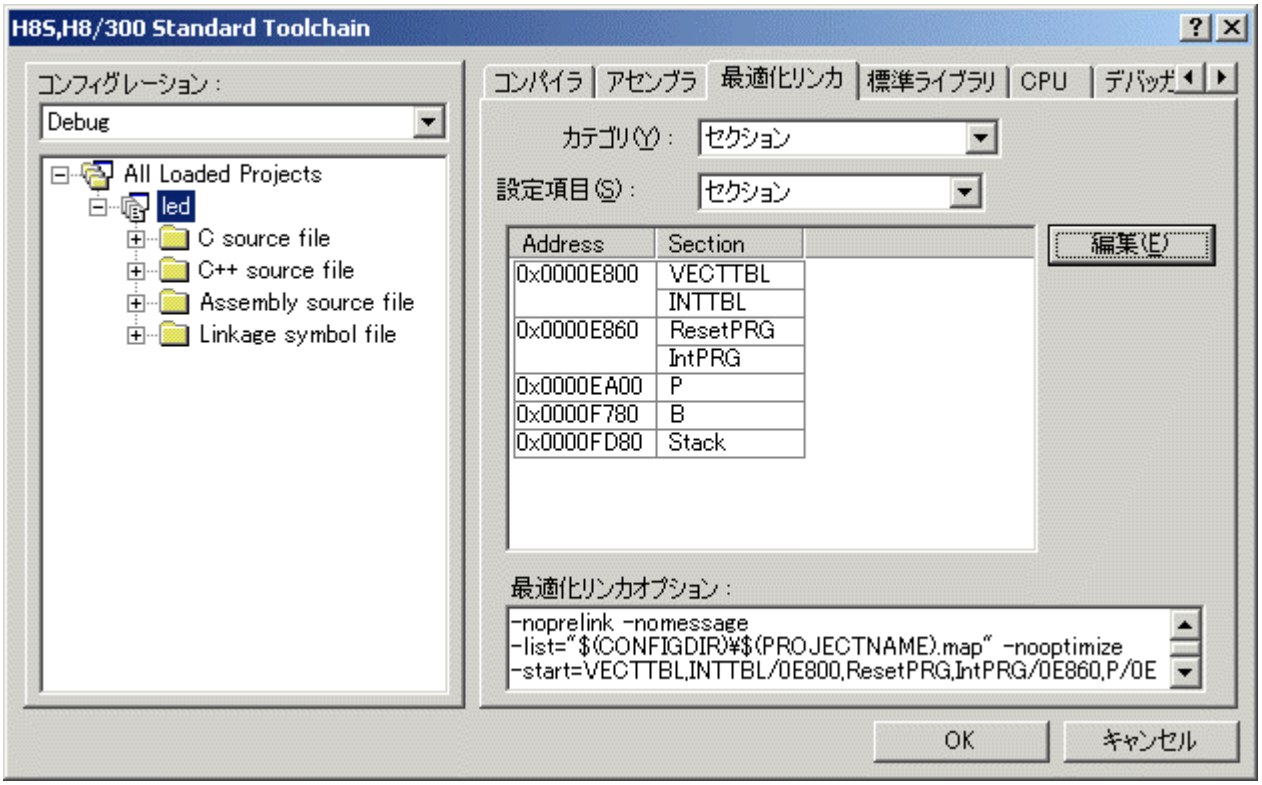


メモリマップと同じように Section が指定されていることを確認します。ちゃんと設定されていたら「OK」をクリックします。



セクション設定の保存

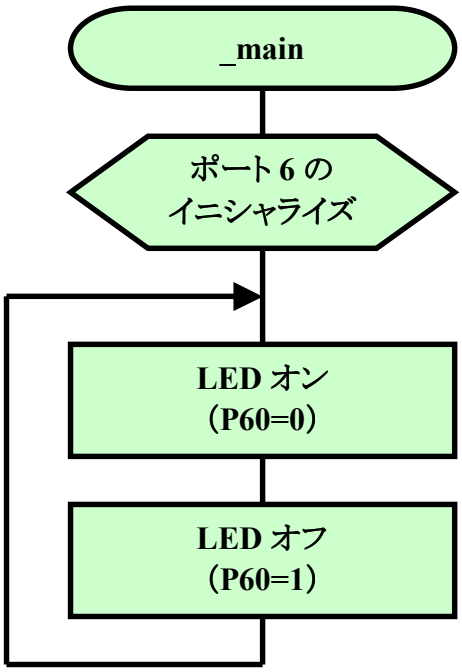
次回のために今修正したセクション情報を保存することができます。下段の「エクスポート(E)」ボタンをクリックしてください。保存用のダイアログが開きますので好きな名前を付けて保存します。次回は「インポート(I)」ボタンをクリックすると保存したセクション設定を呼び出すダイアログが開きます。(おすすめ!!)



もう一度確認してから「OK」をクリックして‘H8S，H8/300 Standard Toolchain’ウィンドウを閉じます。

3. プログラムの入力

だいぶ間隔が開きましたので，ここでもう一度，第4章で作ったプログラムのフローチャートとコーディングしたソースリストも思い出しておきましょう。




```
_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
```

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

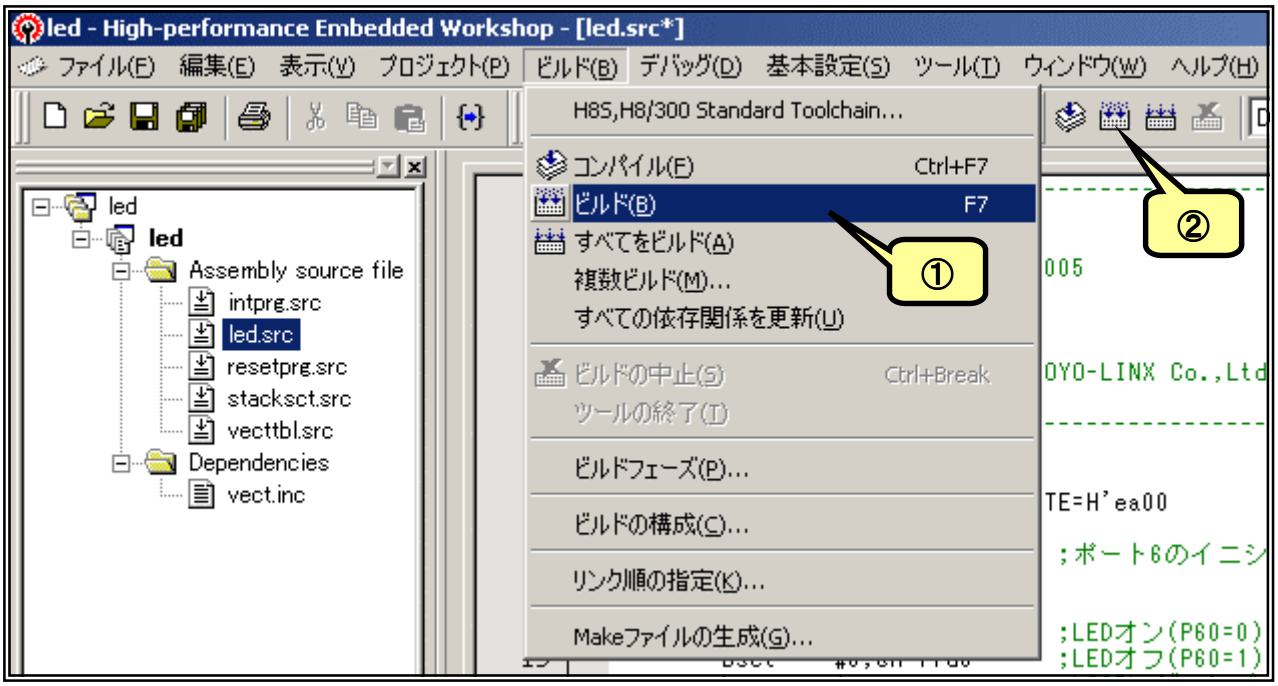
```
;------------------------------------------------------------------------
;                                                                       |
;   FILE        :led.src                                                |
;   DATE        :Wed, Apr 20, 2005                                      |
;   DESCRIPTION :Main Program                                           |
;   CPU TYPE    :H8/3687                                                |
;                                                                       |
;   This file is generated by Renesas Project Generator (Ver.4.0).      |
;                                                                       |
;------------------------------------------------------------------------



        .export     _main
_main:
        rts
        .end
```

このファイルに追加・修正していきます。コーディングしたソースリストを参考に，HEW のアセンブラの文法で追加・修正したあとのリストは次のとおりです。このとおり入力してみてください。なお，アセンブラの文法については，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。

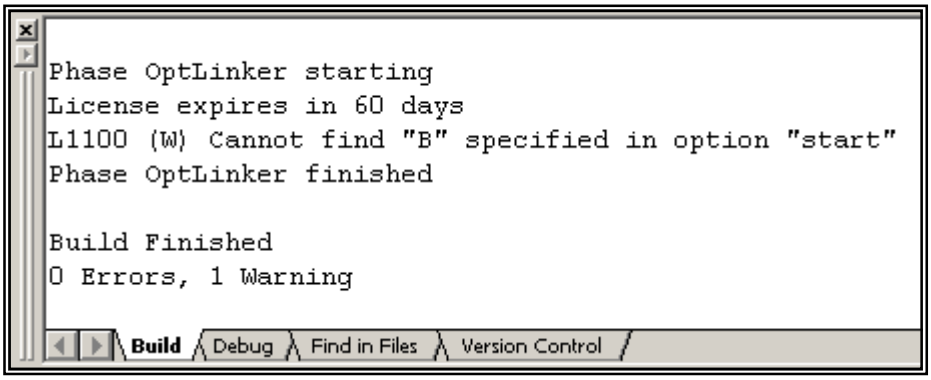
```
;------------------------------------------------------------------------
;                                                                       |
;   FILE        :led.src                                                |
;   DATE        :Wed, Apr 20, 2005                                      |
;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

        .export     _main
        .section    P,CODE,LOCATE=H'ea00
_main:
        mov.b   #H'01,r0l       ;ポート6のイニシャライズ
        mov.b   r0l,@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

        .end
```

4. ビルド!!

では，アセンブルしてみましょう。HEW ではこの作業をビルドと呼んでいます。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。



アセンブル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。



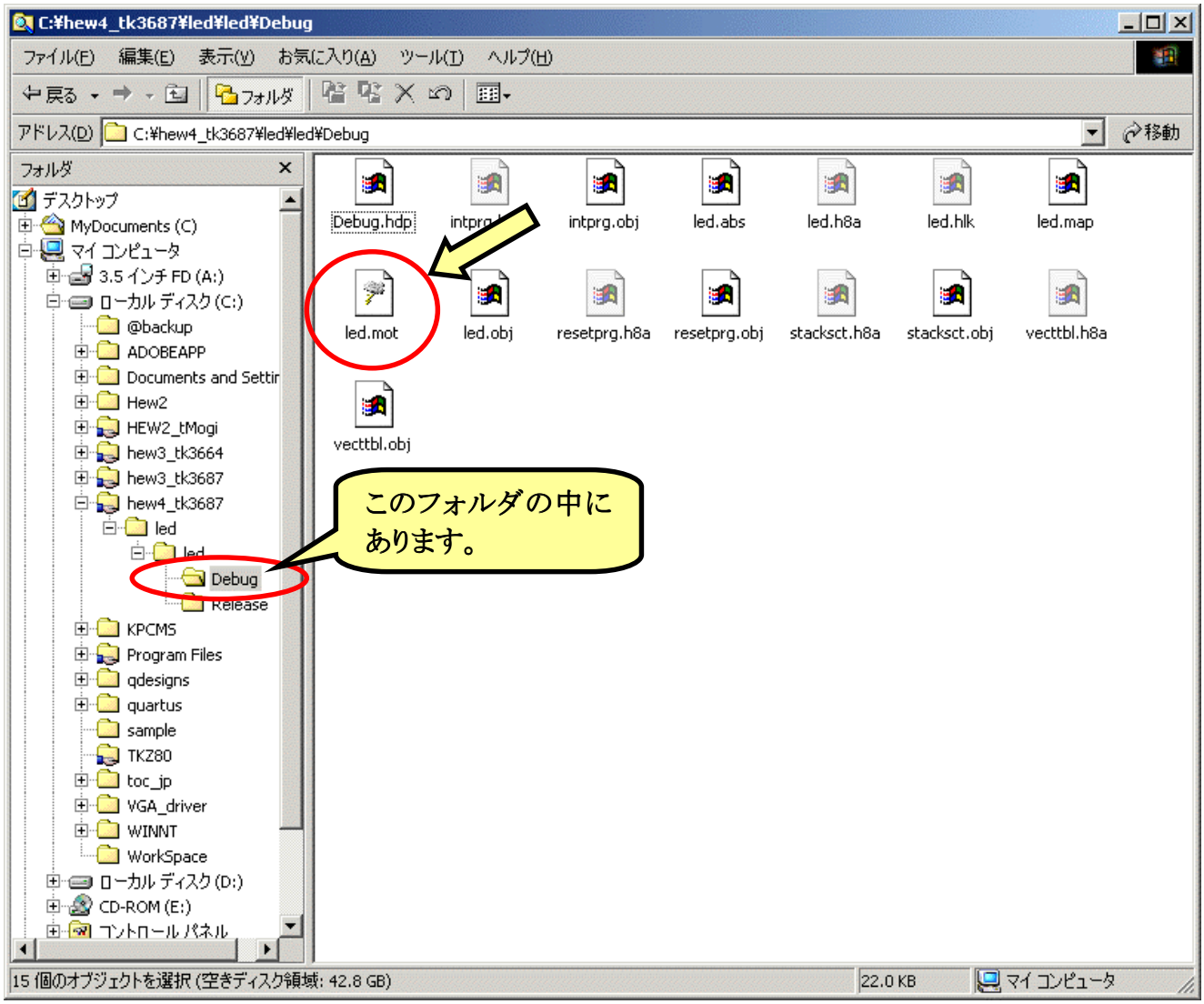
エラーがある場合はソースファイルを修正します。アウトプットウィンドウのエラー項目にマウスカーソルをあててダブルクリックすると，エラー行に飛んでいきます(このあたりの機能が統合化環境の良いところですね。)ソースファイルと前のページのリストを比べてまちがいなく入力しているかもう一度確認して下さい。

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 "B" specifind in option "start"」は，B セクションを設定したのに B 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 B 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 607 ページからアセンブ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。

5. ダウンロードとトレース実行

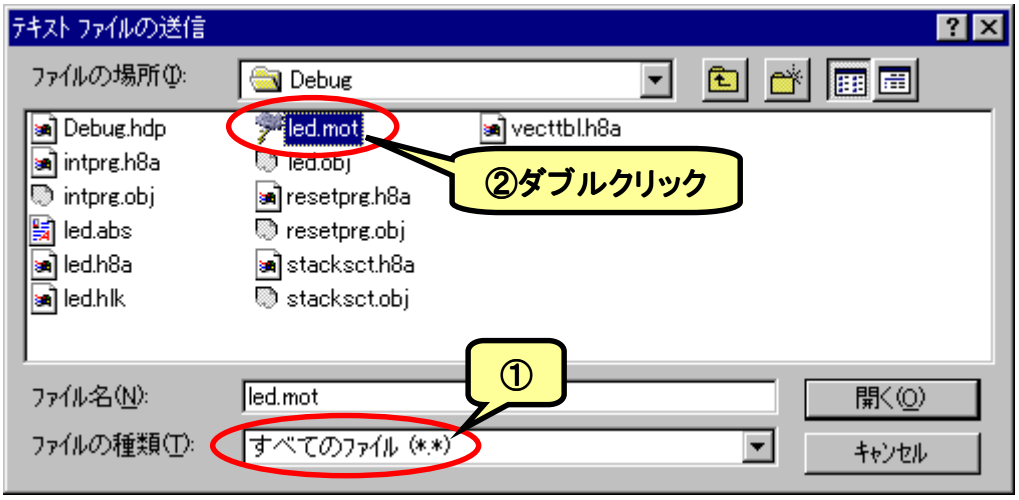
アセンブルすると‘led. mot’というファイルが作られます。拡張子が‘. mot’のファイルは「S タイプファイル」と呼ばれていて，マシン語の情報が含まれているファイルです。このファイルは次のフォルダ内に作られます。



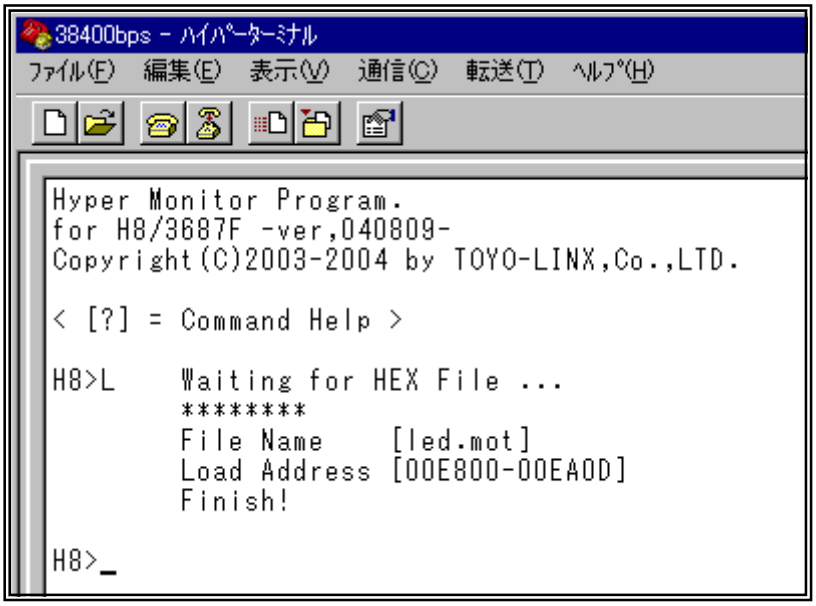
それでは，ハイパーH8 を起動して下さい。‘L’コマンドを使います。パソコンのキーボードから‘L’と入力して‘Enter’キーを押します。

メニューから「テキストファイルの送信(T)…」を選択します。

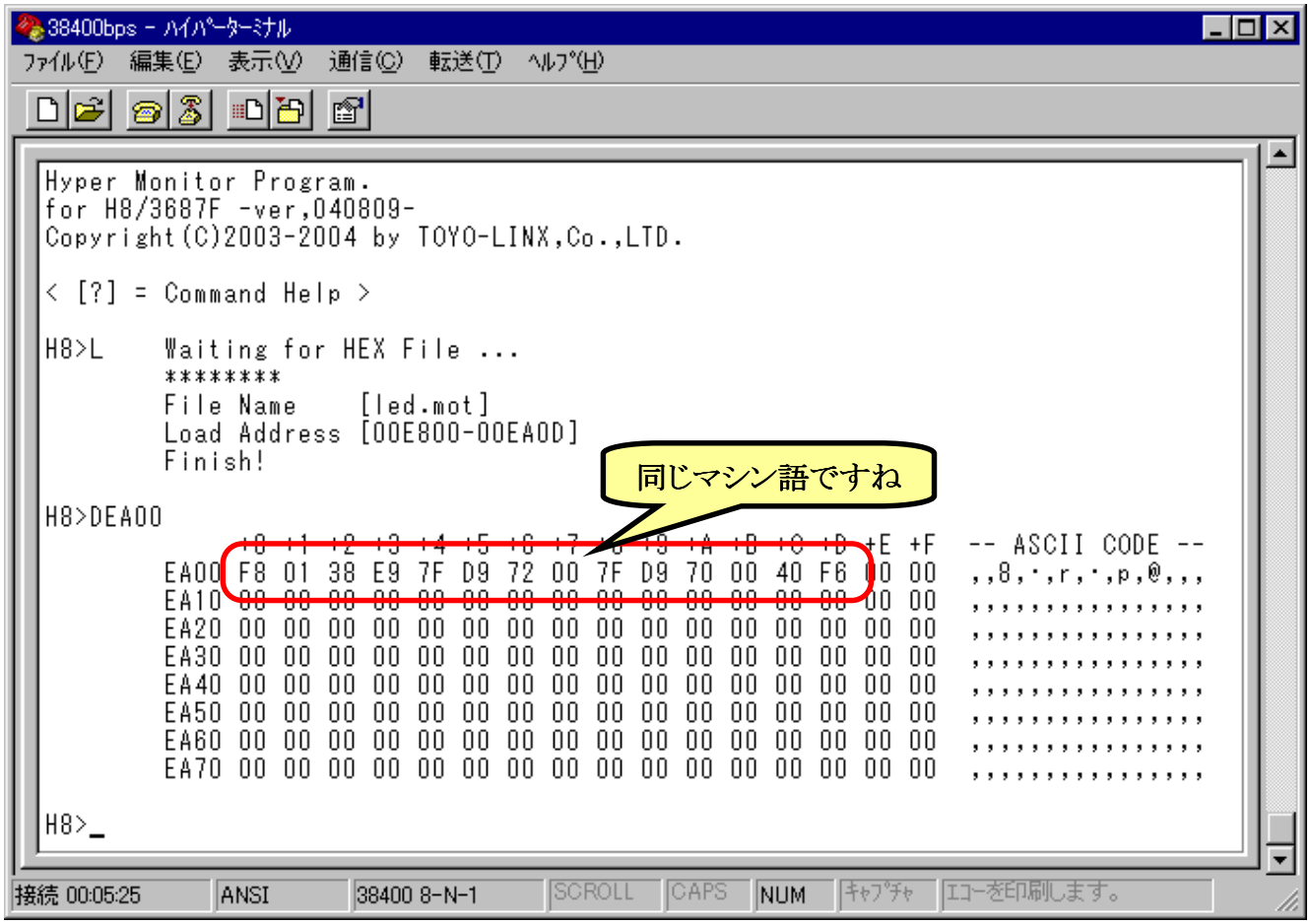
'テキストファイルの送信'ウィンドウが開きます。①ファイルの種類を'すべてのファイル'にして下さい。②'led. mot'をダブルクリックします。



ダウンロードが始まります。終了すると右のように表示されます。



ちゃんとダウンロード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。第 4 章と同じマシン語になっているでしょうか。



あとは第 4 章と同じ方法でトレース実行してみてください。ちゃんと動作するでしょうか。

◆

ところで，‘L’コマンドでダウンロードが終了するとロードアドレスが表示されます。よ～く見てみるとちょっと変ですよね。プログラムは EA00 番地からのはずなのに…

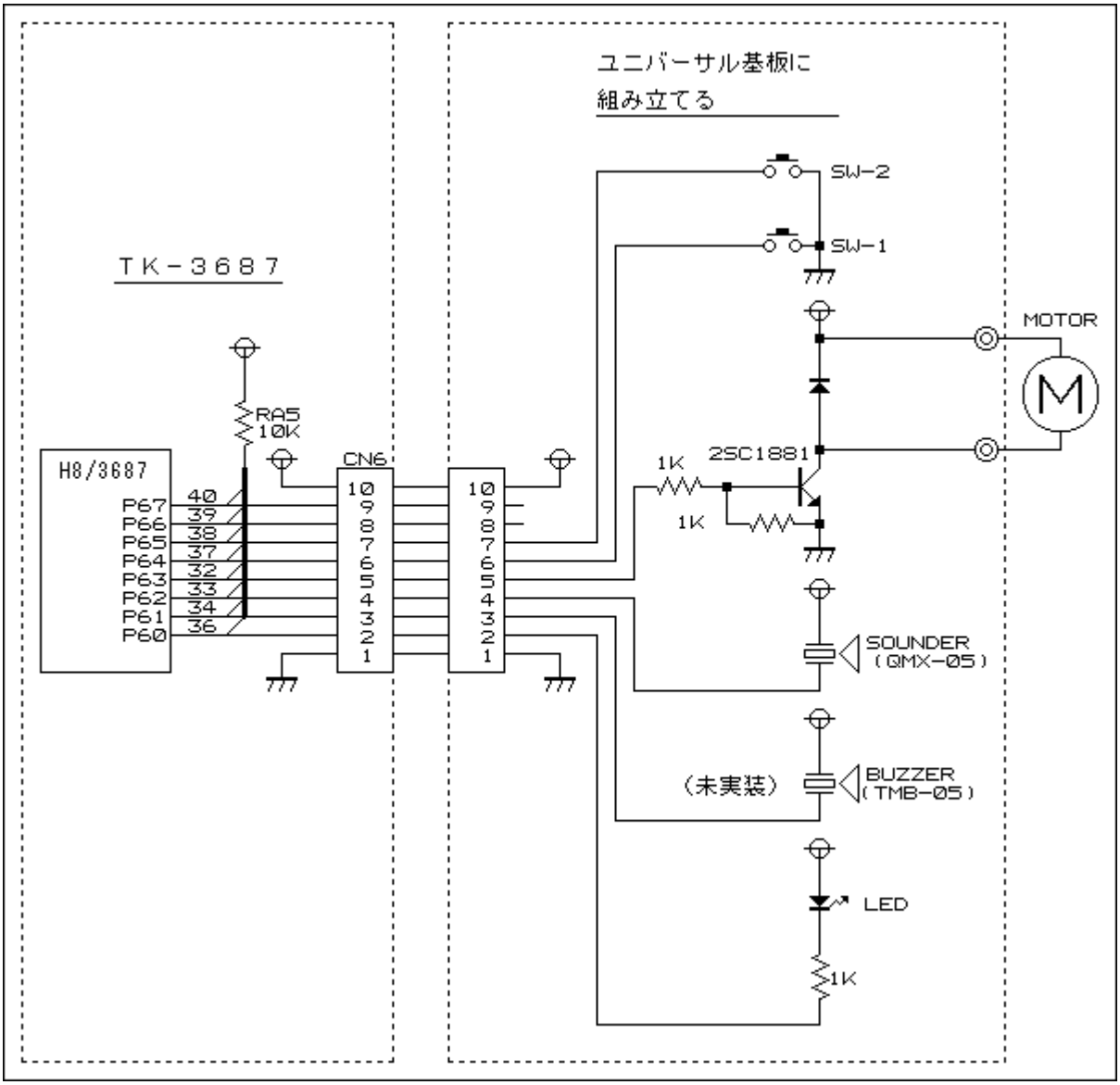
「H8/3687 シリーズ ハードウェアマニュアル」の「3. 例外処理」と，「H8S, H8/300 シリーズ C/C++コンパイラ，アセンブラ，最適化リンケージエディタ ユーザーズマニュアル」を読みこなすと理由がわかってきます。ぜひ，チャレンジしてみてください。

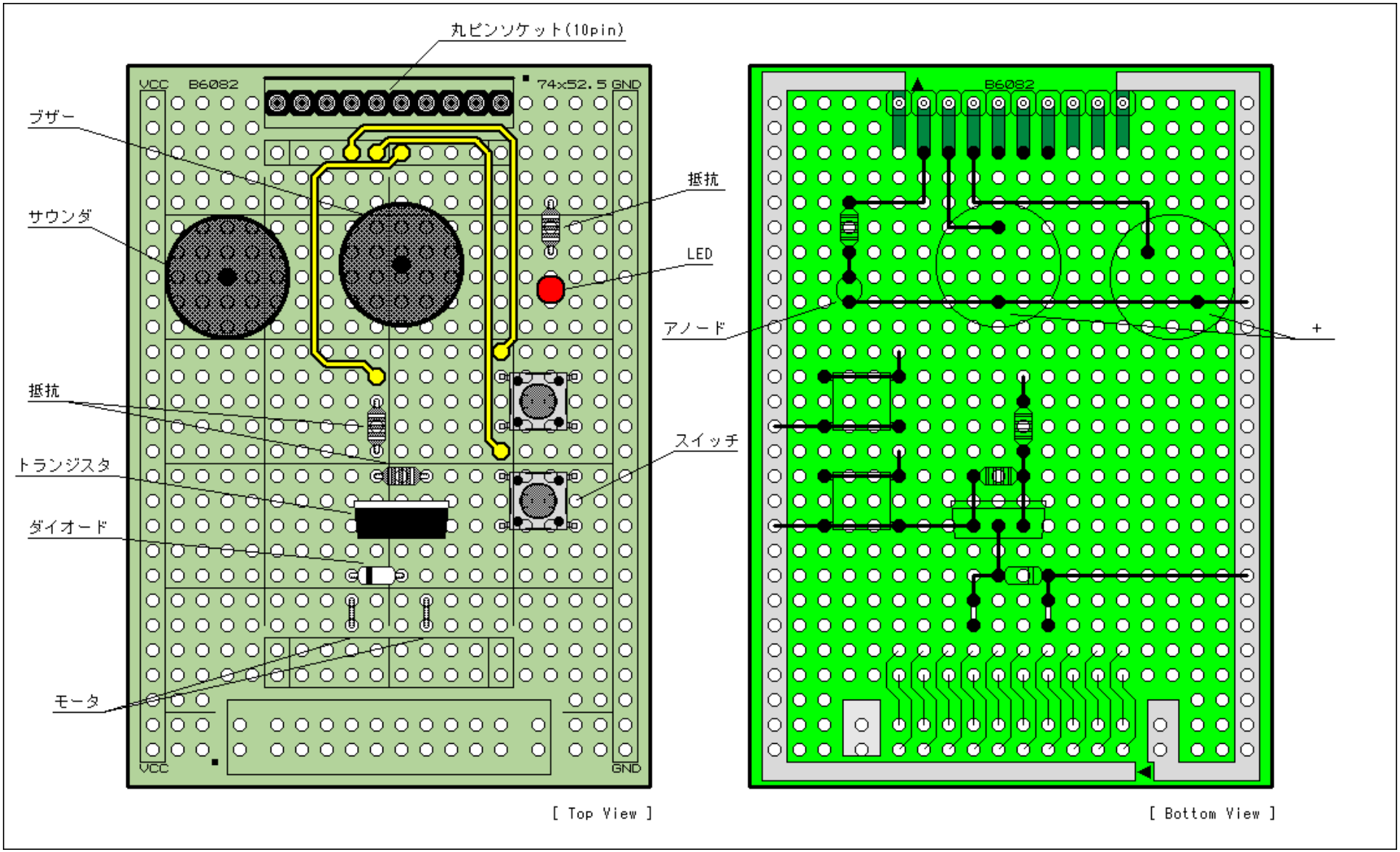
第7章 I/O ポートの使い方をマスターしよう

1. I/O って何だろう？
2. LED を光らせよう
3. ブザーを鳴らそう
4. スイッチ入力

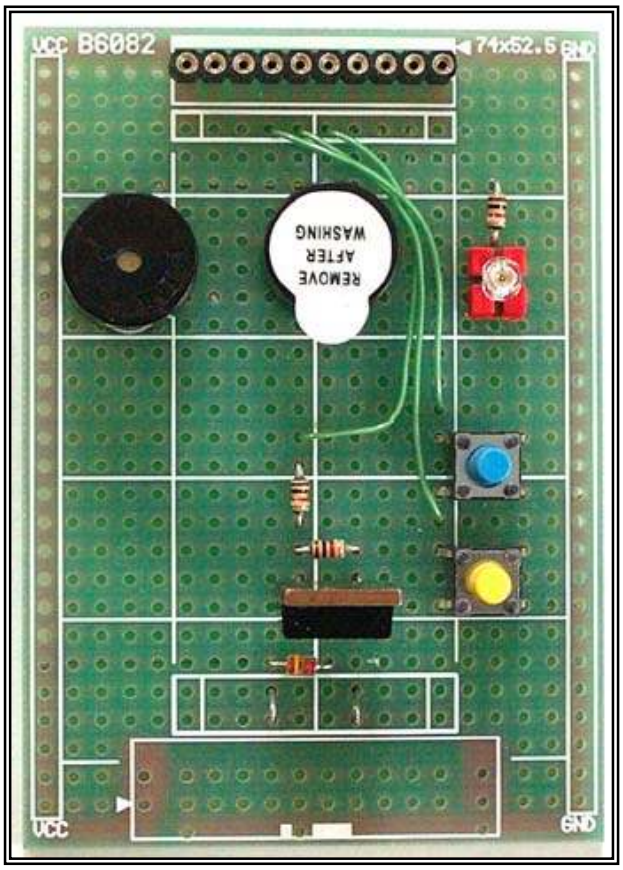
第 4 章と第 6 章で作ったプログラムはポート 6 を使いました。H8/3687 にはこのような I/O ポートが，入出力ポート 45 本，入力専用ポート 8 本，用意されています。I/O ポートを使うと，LED を光らせたり，スイッチを読んだりすることができます。この章では I/O ポートの使い方をマスターして，I/O ポートの考え方，基本的な入出力のプログラムを理解しましょう。なお，I/O ポートの詳しい内容は「H8/3687 グループ ハードウェアマニュアル」(これからハードウェアマニュアルと呼びます)の 9-1 ページから説明されていますので，ぜひお読みください。

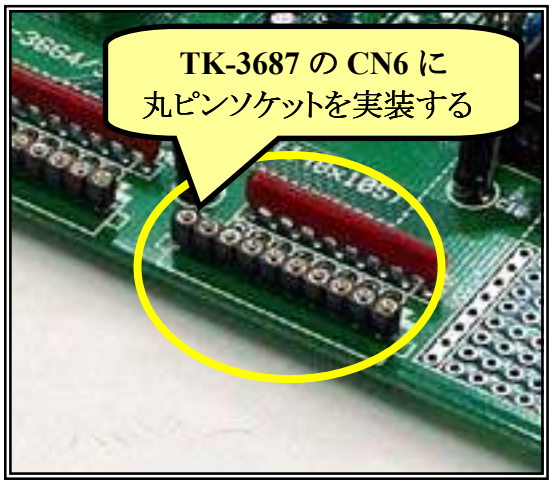
なお，第 7 章と第 9 章の回路はユニバーサル基板に組み立てます。I/O ポートの使い方を調べる前にまず回路の組立てを行ないましょう。TK-3687 の CN6 にも丸ピンソケットを実装してください。組み立てた基板はフレックスジャンパーケーブルで TK-3687 の CN6 に接続します。回路図は下記のとおりです。また実装図は次のページに，また組み立て例と TK-3687 に接続した様子はその次のページの写真をご覧ください。

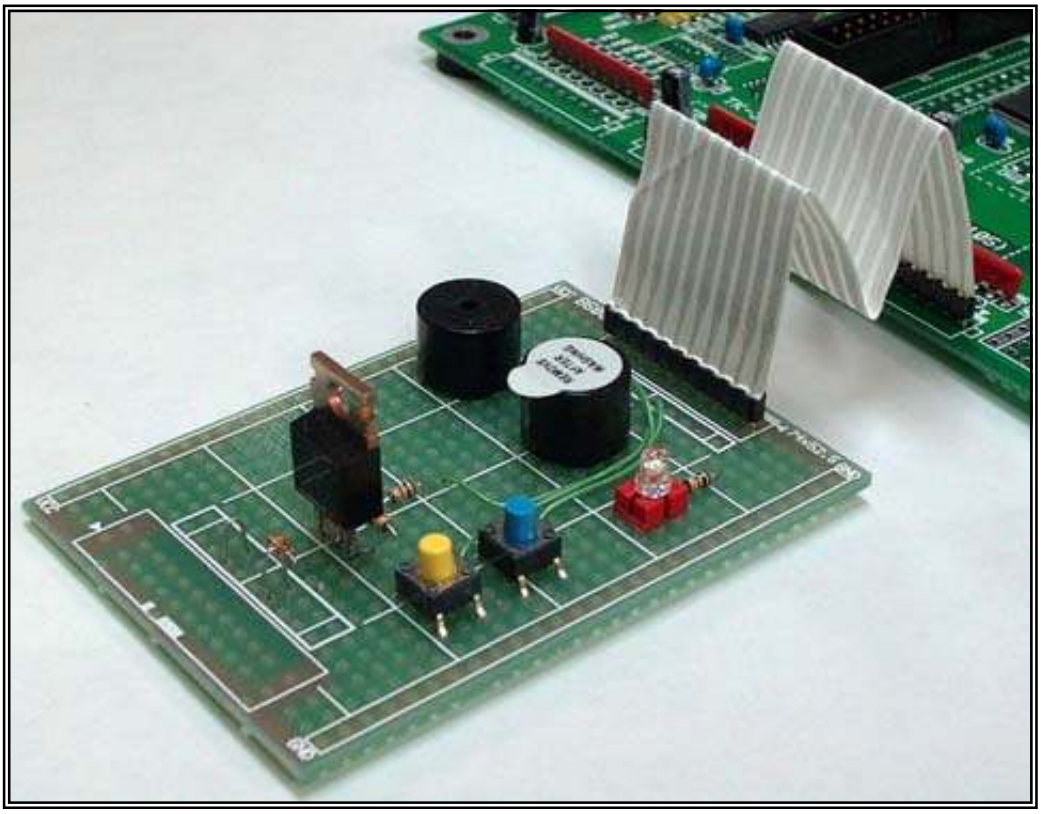




この図ではブザーを実装するようになっていますが，ブザーは付属していません。実装する場合の参考とお考えください。







写真ではブザーを実装するようになっていますが，ブザーは付属していません。実装する場合の参考とお考えください。

1. I/O って何だろう？

そもそも I/O とは何でしょうか。

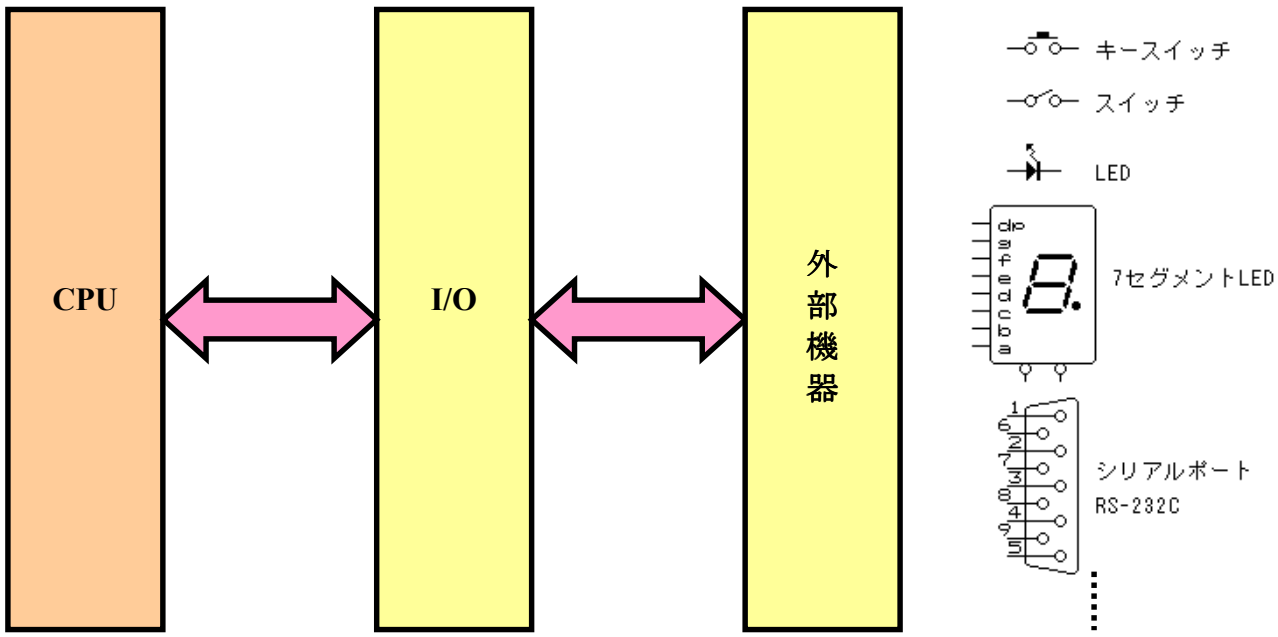
もともと CPU は「できるだけ速く」を合言葉に進歩してきました。H8/3687 は 1 つの命令を 0．1μs～1．2μs（μs:マイクロ秒は 1 秒の百万分の一）で実行できるように作られています。

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，この章で取り上げている I/O ポートはパラレルポートと呼ばれるものです。以後，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。



2. LED を光らせよう

組み立てた回路を例にして，I/O ポートの使い方を考えてみましょう。

P60 に LED がつながっていますのでポート 6 を使います。ハードウェアマニュアルの 9-17 ページからポート 6 について説明されています。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

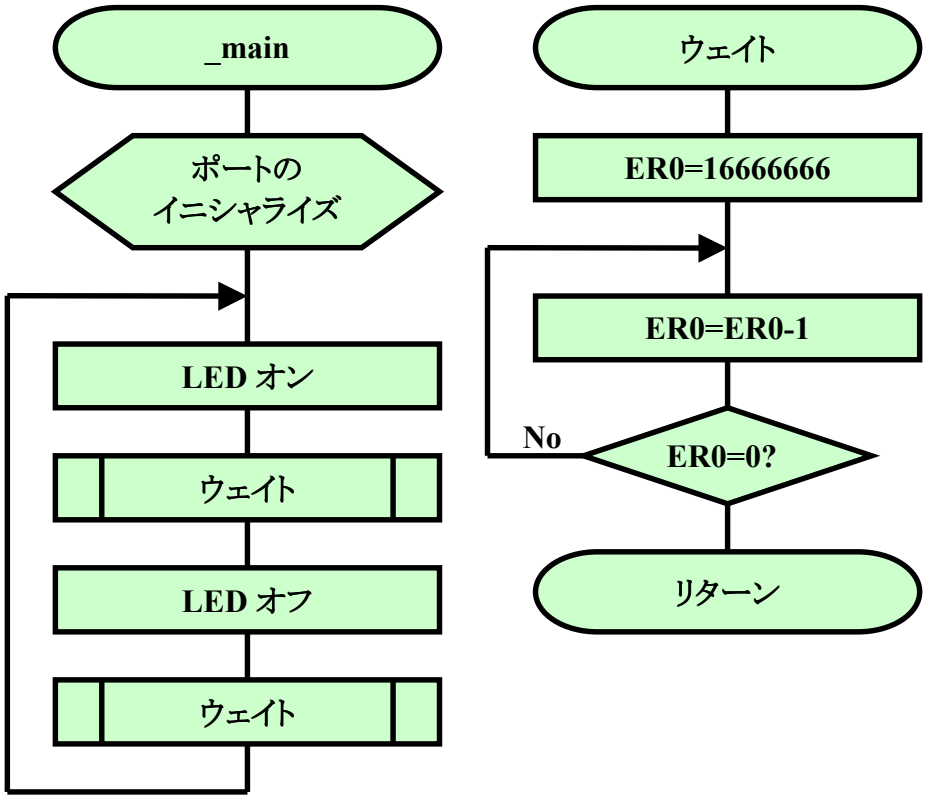
あとは自由に LED を光らせることができます。ポート 6 の入出力は「ポートデータレジスタ 6（PDR6）」で行ないます。

これまでででてきた「なんとかレジスタ」にはすべてアドレスが割り当てられています。プログラムはそのアドレスに対してデータを書いたり読んだりしていきます。アドレスとレジスタの対応を表にして見ましょう。アドレスの順番に並べています。

アドレス	略称	レジスタ名称
FFD9	PDR6	ポートデータレジスタ 6
FFE9	PCR6	ポートコントロールレジスタ 6

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。だいたい 0．5 秒くらい待つようにしてみましょう。

まずはフローチャートを考えます。



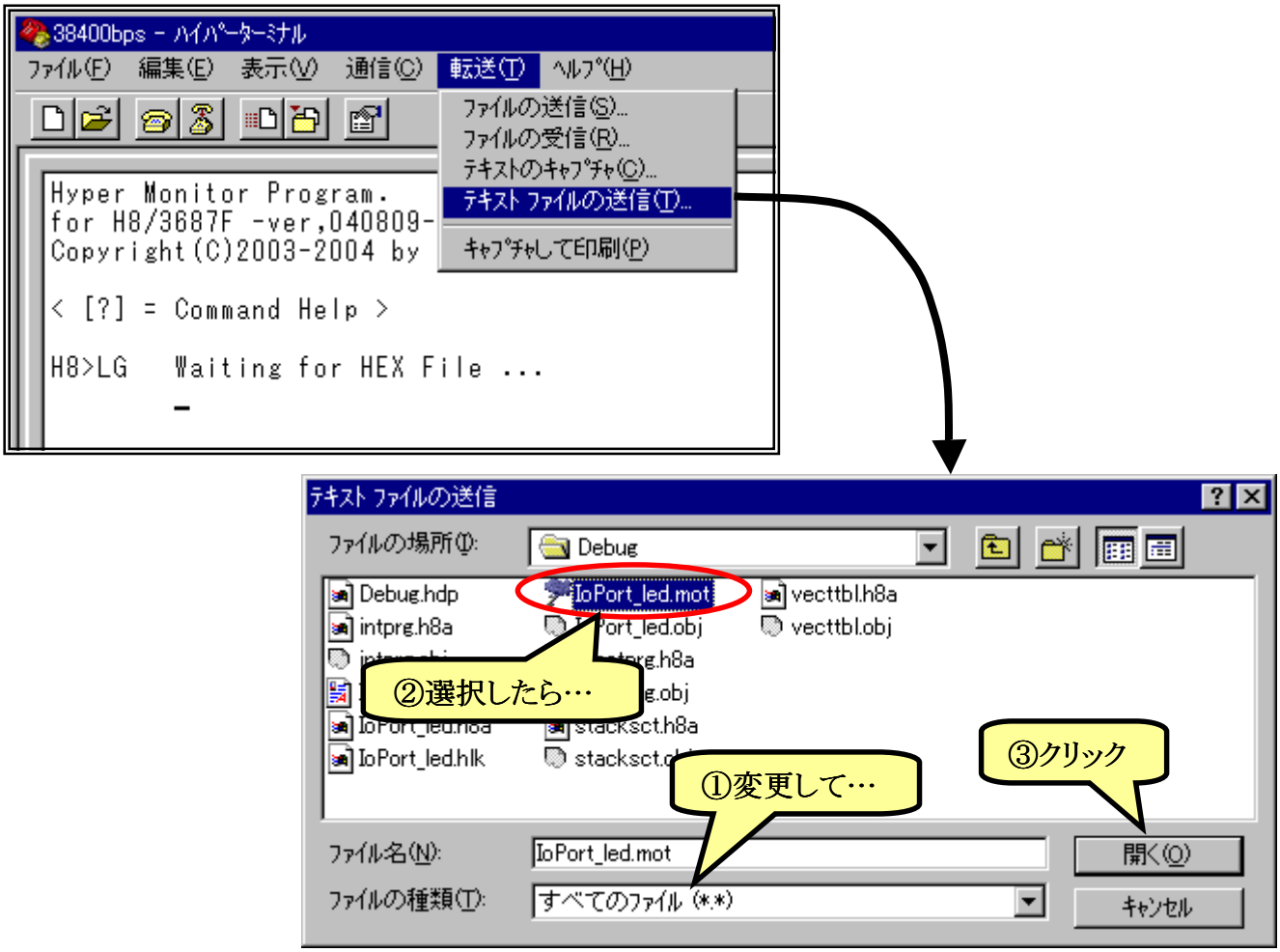
ではコーディングしてみましょう。

```
;---------------------------------------------------------------------
;                                                                     |
;   FILE        :IoPort_led.src                                       |
;   DATE        :Mon, Dec 27, 2004                                    |
;   DESCRIPTION :Main Program                                         |
;   CPU TYPE    :H8/3687                                              |
;                                                                     |
;   This file is programed by TOYO-LINX Co.,Ltd. / yKikuchi           |
;                                                                     |
;---------------------------------------------------------------------

        .export     _main

;*********************************************************************
;    定数定義
;*********************************************************************
PDR6    .equ    h'FFD9    ;ﾎﾟｰﾄﾃﾞｰﾀﾚｼﾞｽﾀ 6
PCR6    .equ    h'FFE9    ;ﾎﾟｰﾄｺﾝﾄﾛｰﾙﾚｼﾞｽﾀ 6

;*********************************************************************
;    メイン
;*********************************************************************
        .section P,CODE,LOCATE=H'ea00
_main:
        mov.b   #B'00000001,r0l   ;ポートのイニシャライズ
        mov.b   r0l,@PCR6
_main_01:
        bclr    #0,@PDR6          ;LEDオン(P60)
        bsr     wait
        bset    #0,@PDR6          ;LEDオフ(P60)
        bsr     wait
        bra     _main_01

;*********************************************************************
;    ウェイト
;*********************************************************************
wait:
        mov.l   #1666666,er0      ;0.5秒
wait_01:
        dec.l   #1,er0
        bne     wait_01
        rts

;---------------------------------------------------------------------
        .end
```

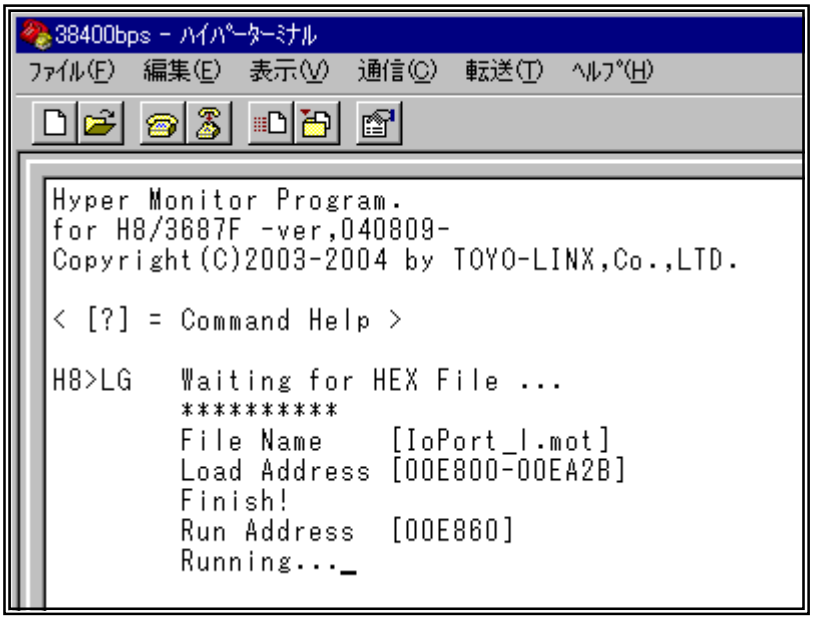
入力が終わったらビルドしてください。

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。それで，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．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。

◆

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。

‘IoPort_led. mot’

をダウンロードして実行して下さい。

ハイパーH8 の‘L’コマンド

第 4 章ではプログラムの実行の前に，‘RPCEA00’と入力してスタートアドレスを指定しました。今回は省略していますが「大丈夫なの？」と思った人もいるかもしれません。

‘L’コマンドはプログラムのダウンロードコマンドですが，ダウンロードするファイルの中にはスタートアドレスがどこかも含まれています。それで，‘L’コマンドでダウンロードしたときには自動的にスタートアドレスをセットするようにハイパーH8 はできています。

ご安心下さい。

3. ブザーを鳴らそう

ブザーは付属しておりません。下記プログラムを試したいときはパーツショップでブザーをご購入ください。

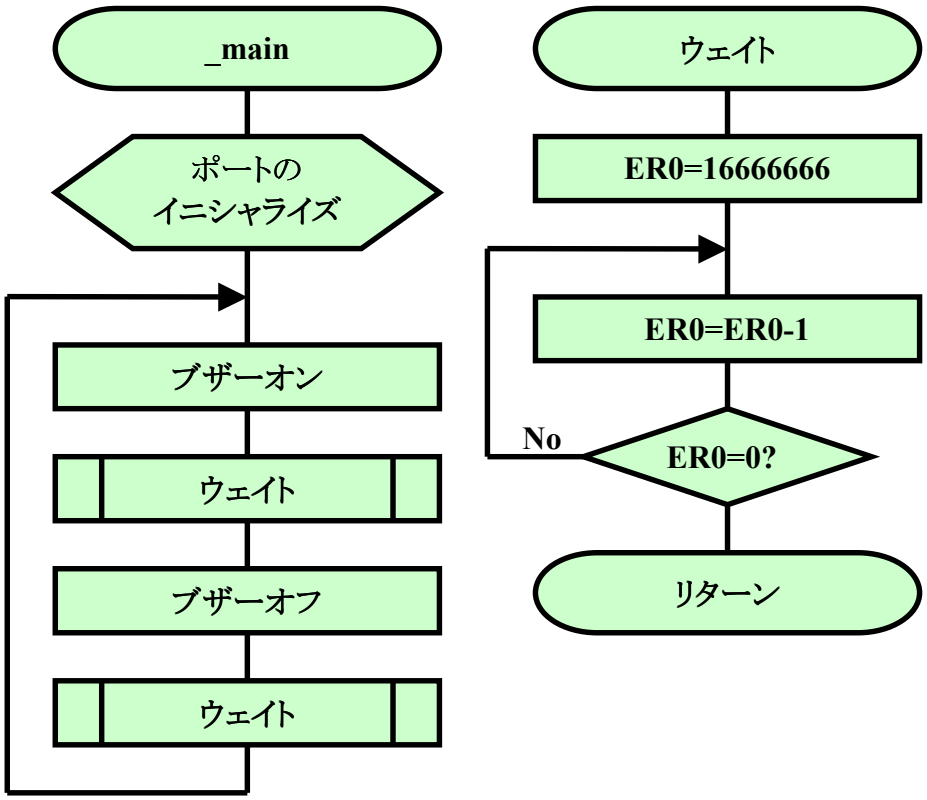
出力ポートの別の例として，ブザーを鳴らしてみましょう。CD-ROM から

‘IoPort_buzzer. mot’

をダウンロードして実行して下さい。この回路図でブザーは P61 を 0 にすると鳴ります。LED のときと同じように，しばらく鳴ったら，しばらく止まる，というのをくりかえします。

◆

さて，フローチャートは LED のときとほとんど同じです。

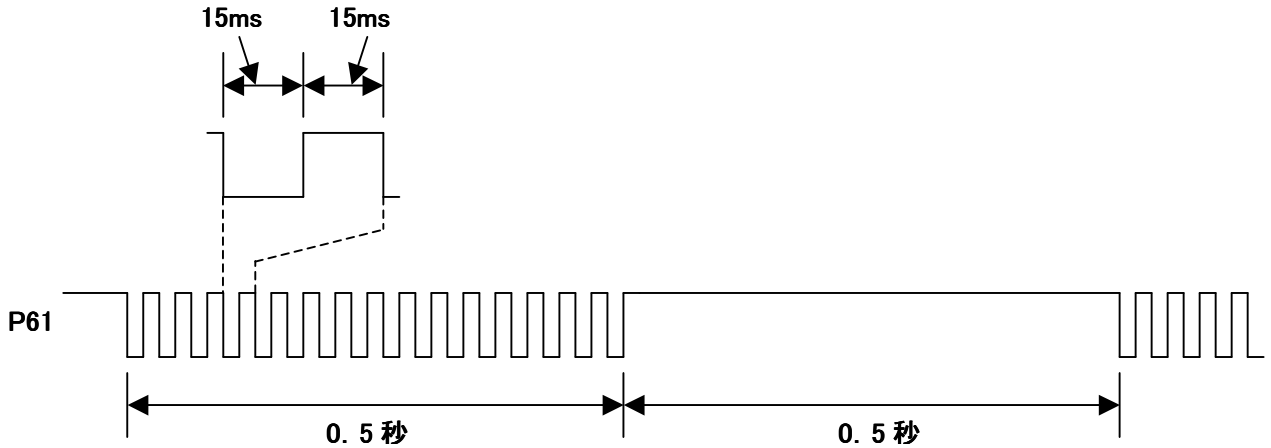


では，コーディングしてみましょう。

```
;---------------------------------------------------------------------
;                                                                     |
;   FILE        :IoPort_buzzer.src                                    |
;   DATE        :Mon, Dec 27, 2004                                    |
;   DESCRIPTION :Main Program                                         |
;   CPU TYPE    :H8/3687                                              |
;                                                                     |
;   This file is programed by TOYO-LINX Co.,Ltd. / yKikuchi           |
;                                                                     |
;---------------------------------------------------------------------

        .export     _main
```

```
;***************************************************************************
;    定数定義
;***************************************************************************
PDR6     .equ     h'FFD9    ;ﾎﾟｰﾄﾃﾞｰﾀﾚｼﾞｽﾀ 6
PCR6     .equ     h'FFE9    ;ﾎﾟｰﾄｺﾝﾄﾛｰﾙﾚｼﾞｽﾀ 6

;***************************************************************************
;    メイン
;***************************************************************************
         .section P,CODE,LOCATE=H'ea00
_main:
         mov.b    #B'00000010,r0l    ;ポートのイニシャライズ
         mov.b    r0l,@PCR6
_main_01:
         bclr     #1,@PDR6           ;ブザーオン(P61)
         bsr      wait
         bset     #1,@PDR6           ;ブザーオフ(P61)
         bsr      wait
         bra      _main_01

;***************************************************************************
;    ウェイト
;***************************************************************************
wait:
         mov.l    #1666666,er0       ;0.5秒
wait_01:
         dec.l    #1,er0
         bne      wait_01
         rts

;---------------------------------------------------------------------------
         .end
```

ところで，今のブザーの音は‘ピー，ピー，…’という感じですね。ちょっと機械的な音なので，‘リリリ…，リリリ…，リリリ…，’という鈴虫のような音色にしてみましょう。ブザーを鳴らすときに，P61 を単純に 0 にするのではなく，15ms ごとに 0 と 1 を繰り返すようにします。文章ではわかりにくいのでタイミングチャートをかいてみましょう。

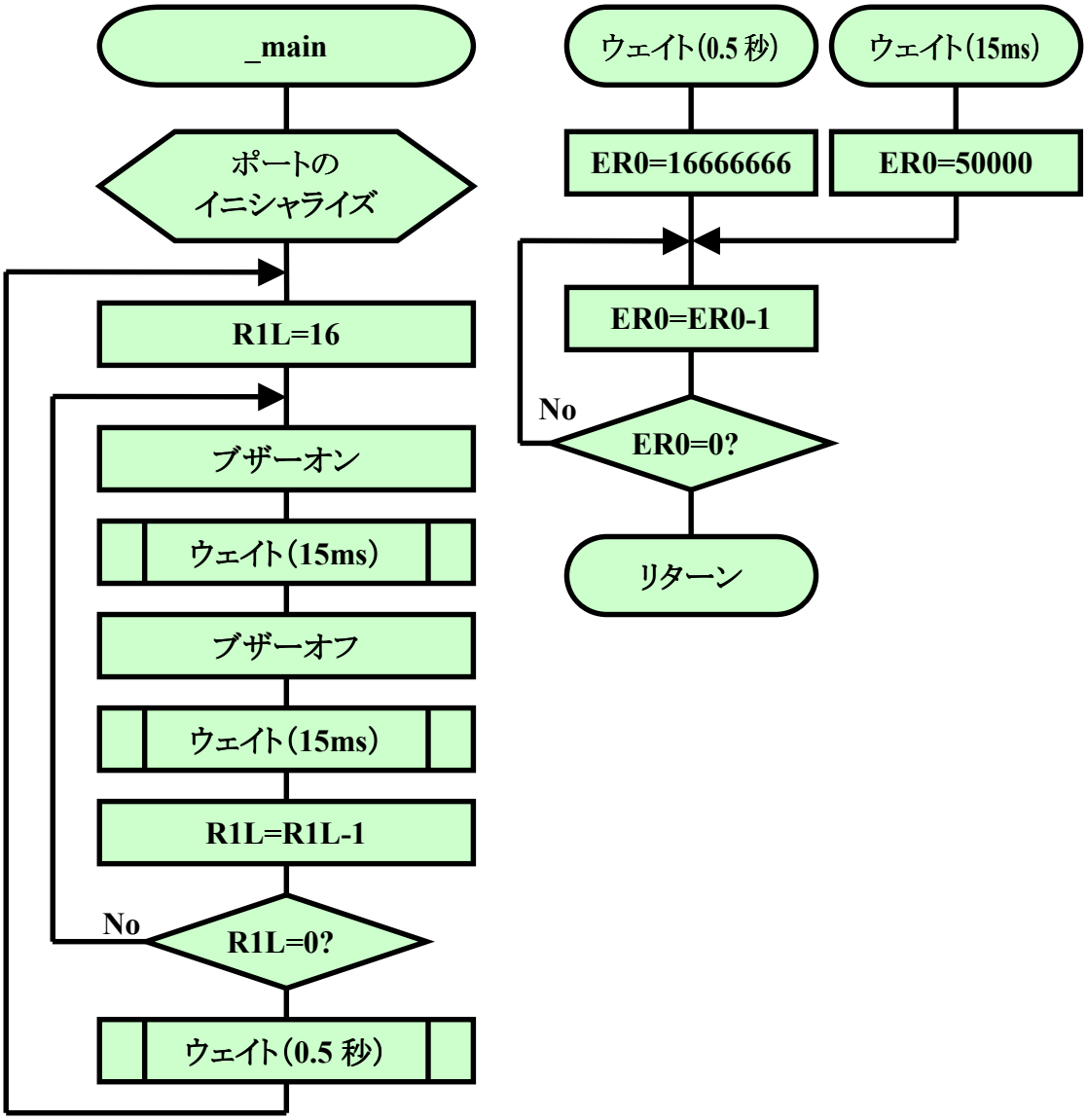


これだけで音色がかなり変わります。不思議ですね。CD-ROM から

‘IoPort_buzzer2. mot’

をダウンロードして実行して下さい。

では，いつものようにフローチャートとソースリストです。



```
;------------------------------------------------------------------------
;                                                                       |
;   FILE        :IoPort_buzzer2.src                                     |
;   DATE        :Mon, Dec 27, 2004                                      |
;   DESCRIPTION :Main Program                                           |
;   CPU TYPE    :H8/3687                                                |
;                                                                       |
;   This file is generated by Hitachi Project Generator (Ver.2.1).      |
;                                                                       |
;------------------------------------------------------------------------

        .export     _main

;************************************************************************
;    定数定義
;************************************************************************
PDR6    .equ    h'FFD9    ;ﾎﾟｰﾄﾃﾞｰﾀﾚｼﾞｽﾀ 6
PCR6    .equ    h'FFE9    ;ﾎﾟｰﾄｺﾝﾄﾛｰﾙﾚｼﾞｽﾀ 6

;************************************************************************
;    メイン
;************************************************************************
        .section  P,CODE,LOCATE=H'ea00
_main:
        mov.b   #B'00000010,r0l     ;ポートのイニシャライズ
        mov.b   r0l,@PCR6
_main_01:
        mov.b   #16,r1l
_main_02:
        bclr    #1,@PDR6            ;ブザーオン(P61)
        bsr     wait_15
        bset    #1,@PDR6            ;ブザーオフ(P61)
        bsr     wait_15
        dec.b   r1l
        bne     _main_02
        bsr     wait_500
        bra     _main_01

;************************************************************************
;    ウェイト
;************************************************************************
wait_500:
        mov.l   #1666666,er0        ;0.5秒
        bra     wait_01
wait_15:
        mov.l   #50000,er0          ;15ms
wait_01:
        dec.l   #1,er0
        bne     wait_01
        rts

;------------------------------------------------------------------------
        .end
```

4. スイッチ入力

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。

CD-ROM から

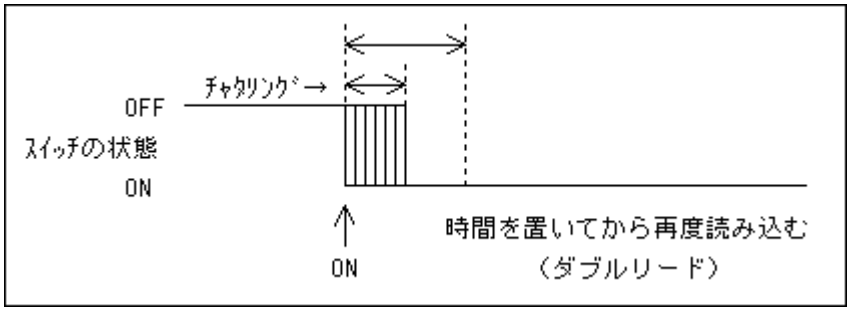
‘IoPort_sw_led．mot’

をダウンロードして実行して下さい。スイッチが押された瞬間だけ LED が光りますか。

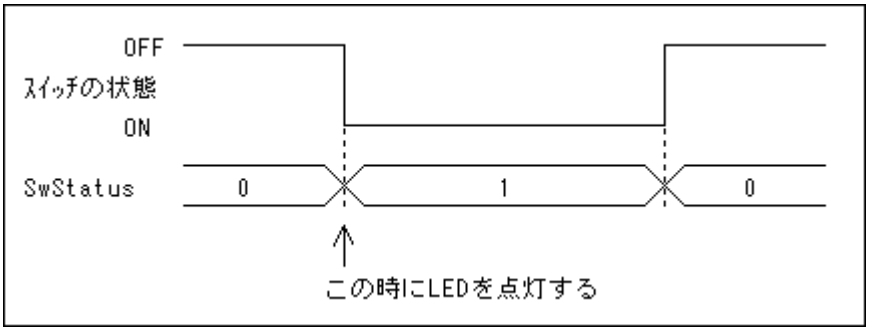
◆

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。

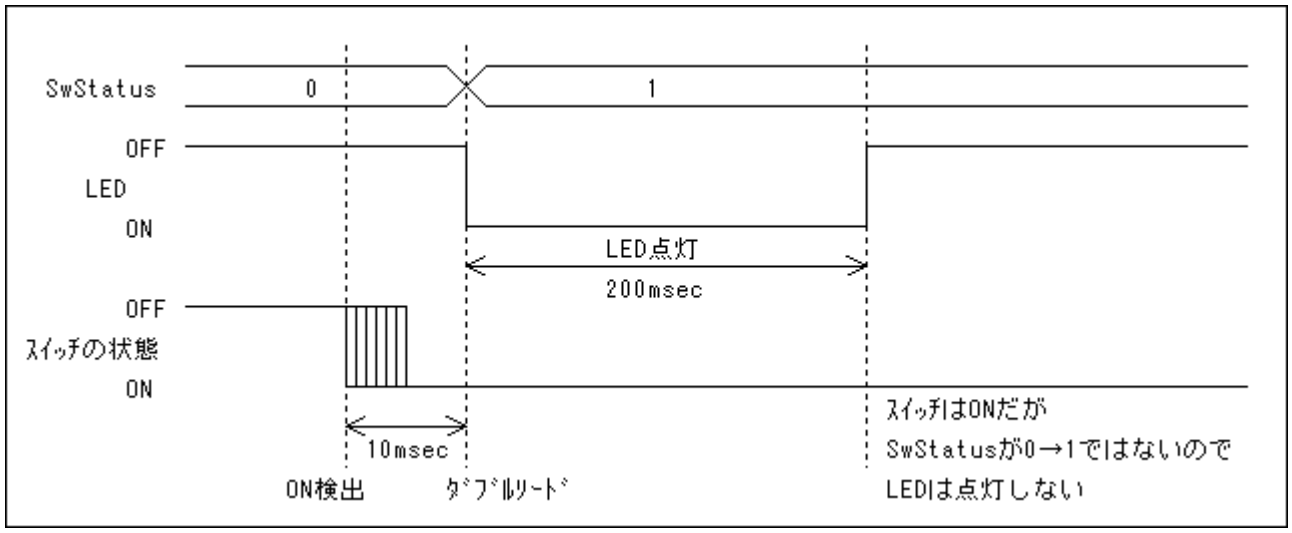
チャタリングを取り除くために，スイッチがオンしたら，しばらく待ってから（10ms ぐらい）もう一度読む（ダブルリード），ということを行ないます。2 度目に読んだときもオンだったら本当にスイッチがオンしたと見なします。

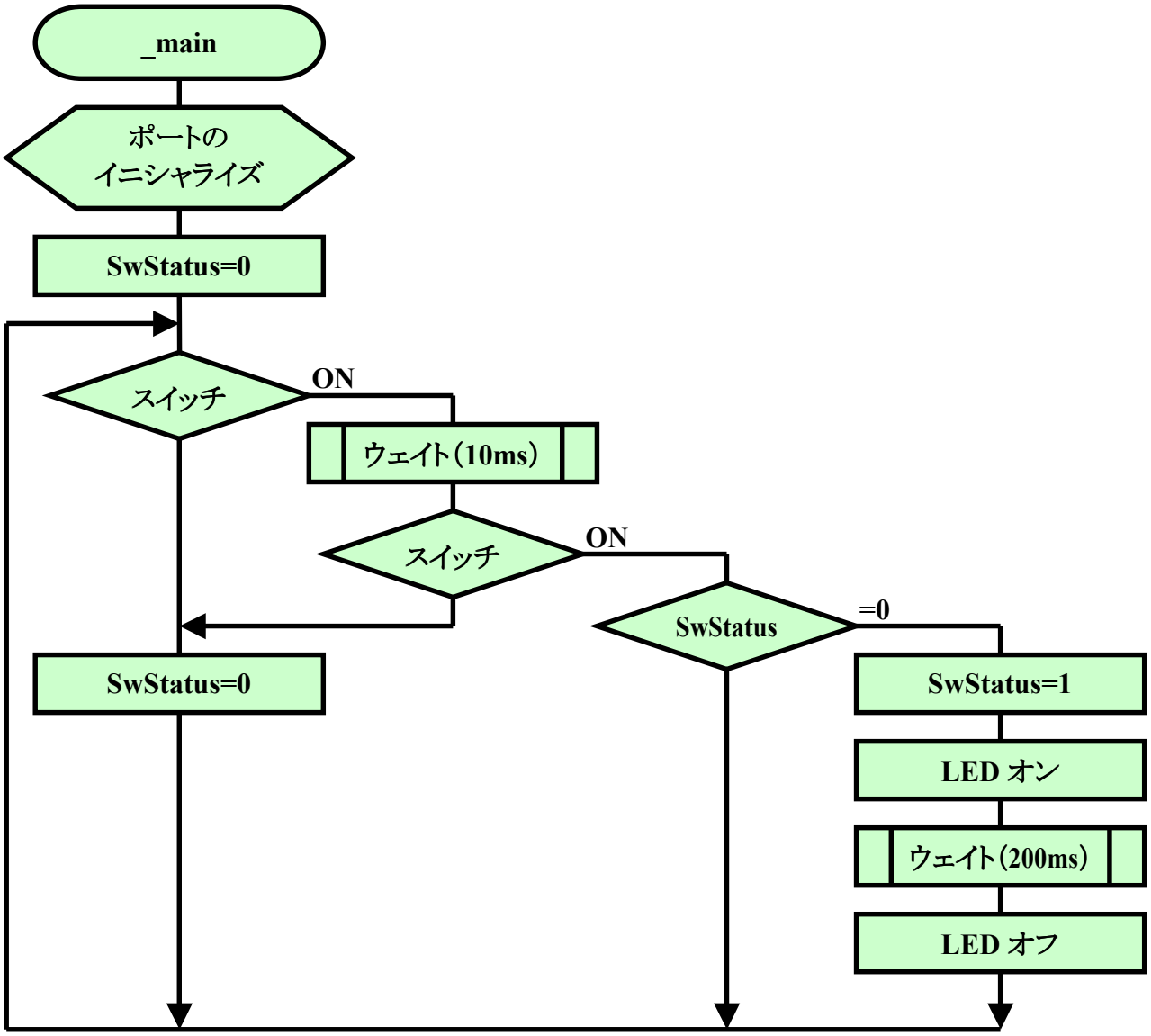


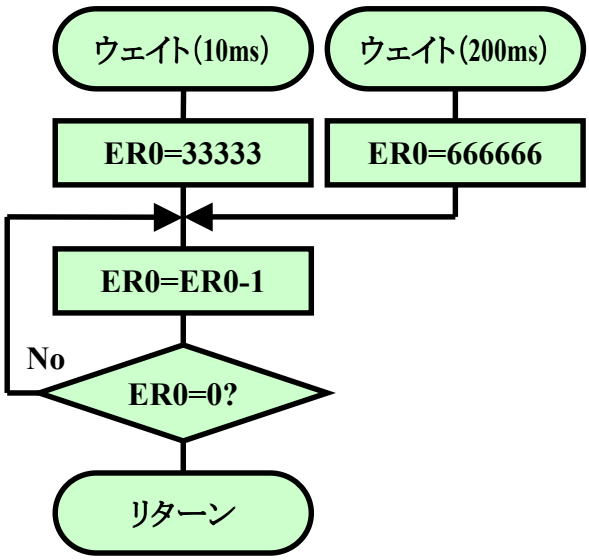
次はワンショット動作について考えてみましょう。スイッチがオンになった瞬間だけを検出するためにスイッチの状態をおぼえておくことにします。変数として‘SwStatus’を B セクションに用意し，SwStatus=0 のときはスイッチが押されていない，SwStatus=1 のときはスイッチが押されている，ということにします。スイッチが押された瞬間を検出するので，SwStatus が 0 から 1 に変化したときに LED を点灯します。



以上のことを考えてタイミングチャートとフロー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。




_main
ポートの
イニシャライズ
SwStatus=0
スイッチ
ON
ウェイト(10ms)
スイッチ
ON
SwStatus
=0
SwStatus=0
SwStatus=1
LED オン
ウェイト(200ms)
LED オフ


ウェイト(10ms)
ウェイト(200ms)
ER0=33333
ER0=666666
ER0=ER0-1
No
ER0=0?
リターン

```
;---------------------------------------------------------------------
;                                                                     |
;   FILE        :IoPort_sw_led.src                                    |
;   DATE        :Fri, Jan 14, 2005                                    |
;   DESCRIPTION :Main Program                                         |
;   CPU TYPE    :H8/3687                                              |
;                                                                     |
;   This file is programed by TOYO-LINX Co.,Ltd / yKikuchi            |
;                                                                     |
;---------------------------------------------------------------------

        .export     _main

;*********************************************************************
;   定数定義
;*********************************************************************
PDR6    .equ    h'FFD9   ;ﾎﾟｰﾄﾃﾞｰﾀﾚｼﾞｽﾀ 6
PCR6    .equ    h'FFE9   ;ﾎﾟｰﾄｺﾝﾄﾛｰﾙﾚｼﾞｽﾀ 6

;*********************************************************************
;   メイン
;*********************************************************************
        .section P,CODE,LOCATE=H'ea00
_main:
;----- イニシャライズ ----------------------------------------------
        mov.b   #B'00000001,r0l   ;ポートのイニシャライズ
        mov.b   r0l,@PDR6         ; あらかじめLEDをオフにする
        mov.b   r0l,@PCR6         ; bit1-7 入力 : bit0 出力

        mov.b   #0,r0l            ;スイッチの状態クリア
        mov.b   r0l,@SwStatus

;----- メインループ ------------------------------------------------
_main_01:
        btst    #4,@PDR6          ;スイッチリード
        beq     _main_02          ; オン
        bra     _main_03          ; オフ
_main_02:
        bsr     wait_10           ;チャタリング除去
        btst    #4,@PDR6          ;スイッチリード（ダブルリード）
        beq     _main_10          ; オン
_main_03:
        mov.b   #0,r0l            ;スイッチの状態, オフにする
        mov.b   r0l,@SwStatus
        bra     _main_01

_main_10:
        mov.b   @SwStatus,r0l     ;今までのスイッチ状態
        beq     _main_11          ; オフ
        bra     _main_01          ; オン
```

```
_main_11:
        mov.b   #1,r0l          ;スイッチの状態，オンにする
        mov.b   r0l,@SwStatus
        bclr    #0,@PDR6        ;LEDオン
        bsr     wait_200        ;200msウェイト
        bset    #0,@PDR6        ;LEDオフ
        bra     _main_01

;*************************************************************************
;   ウェイト
;*************************************************************************
wait_10:
        mov.l   #33333,er0      ;10ms
        bra     wait_01
wait_200:
        mov.l   #666666,er0     ;200ms
wait_01:
        dec.l   #1,er0
        bne     wait_01
        rts

;*************************************************************************
;   ワークエリア
;*************************************************************************
        .section B,DATA,LOCATE=H'f780
SwStatus .res.b  1              ;スイッチの状態
                                ; 0-オフ
                                ; 1-オン

;-------------------------------------------------------------------------
        .end
```

第8章 シリアルポートの使い方入門

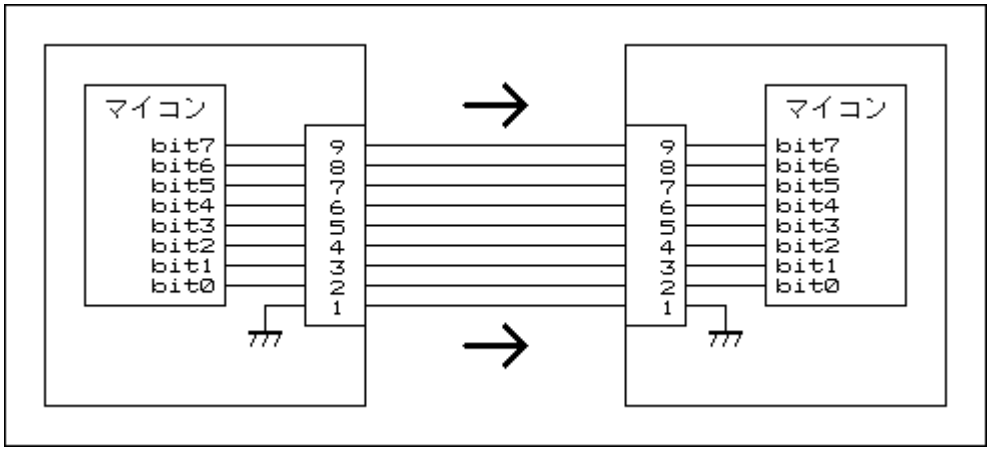
1. シリアル通信の基本的な考え方　　4. ハイパーターミナルから送られてくるデータを見てみよう
2. 調歩同期式シリアル通信
3. シリアルコミュニケーションインタフェース3

前の章では I/O のうちパラレルポートについて調べてきました。この章では，ちょっと趣向をかえてシリアルポートを調べてみましょう。いまや時代はシリアル通信が中心になりつつあります（USB，IEEE1394，LAN，ハードディスクなど）。

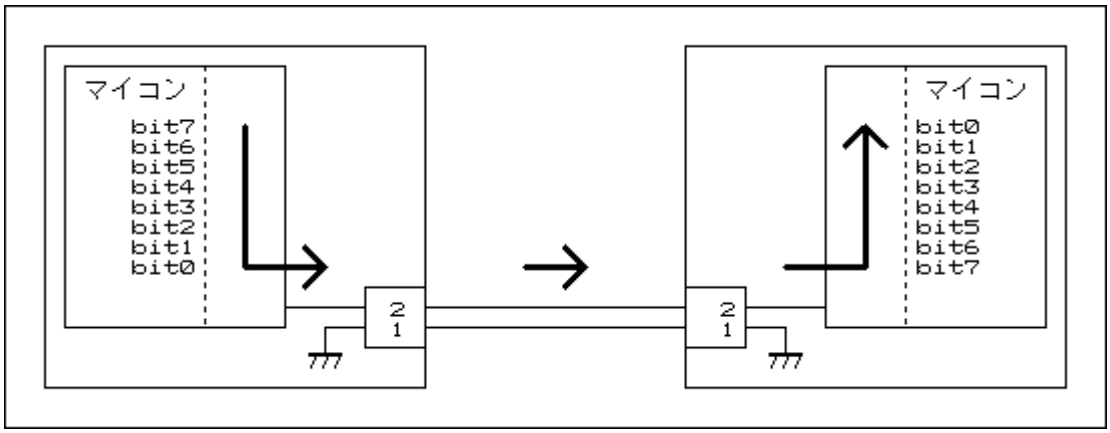
1. シリアル通信の基本的な考え方

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。

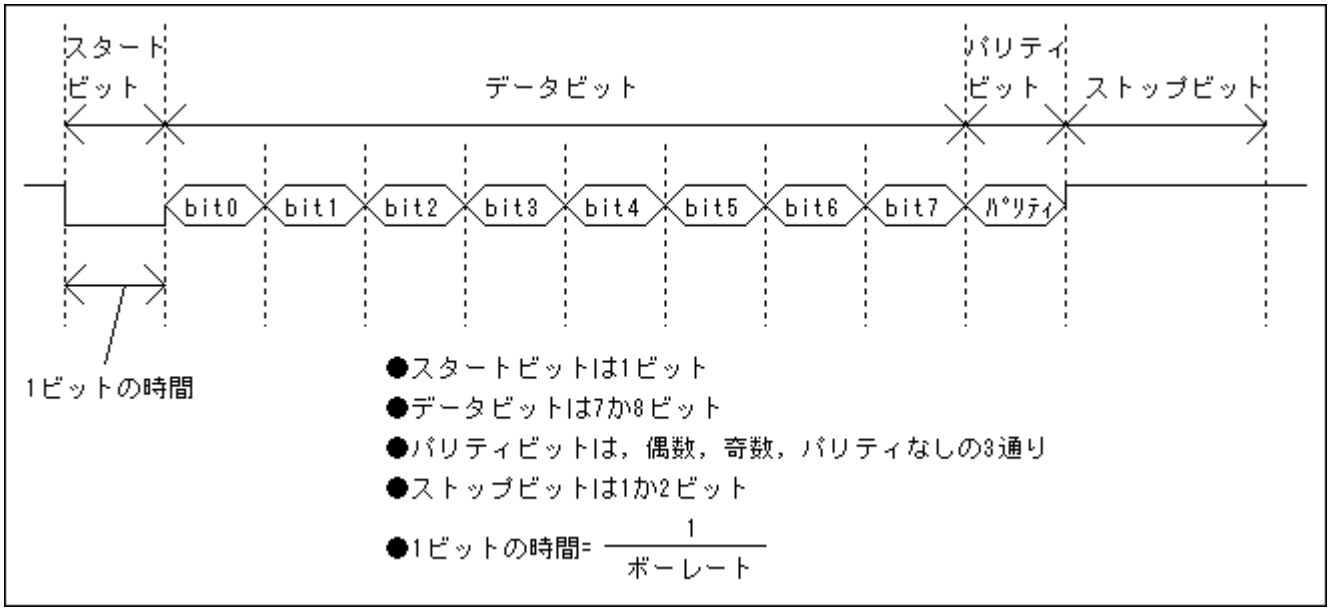


上の図では bit0 から順番に送り出します。受ける方は bit0 から受けていきます。8 ビット受け取ったら，1 バイトデータとして使います。

2. 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。うまくできていると思いませんか。

3. シリアルコミュニケーションインタフェース 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。

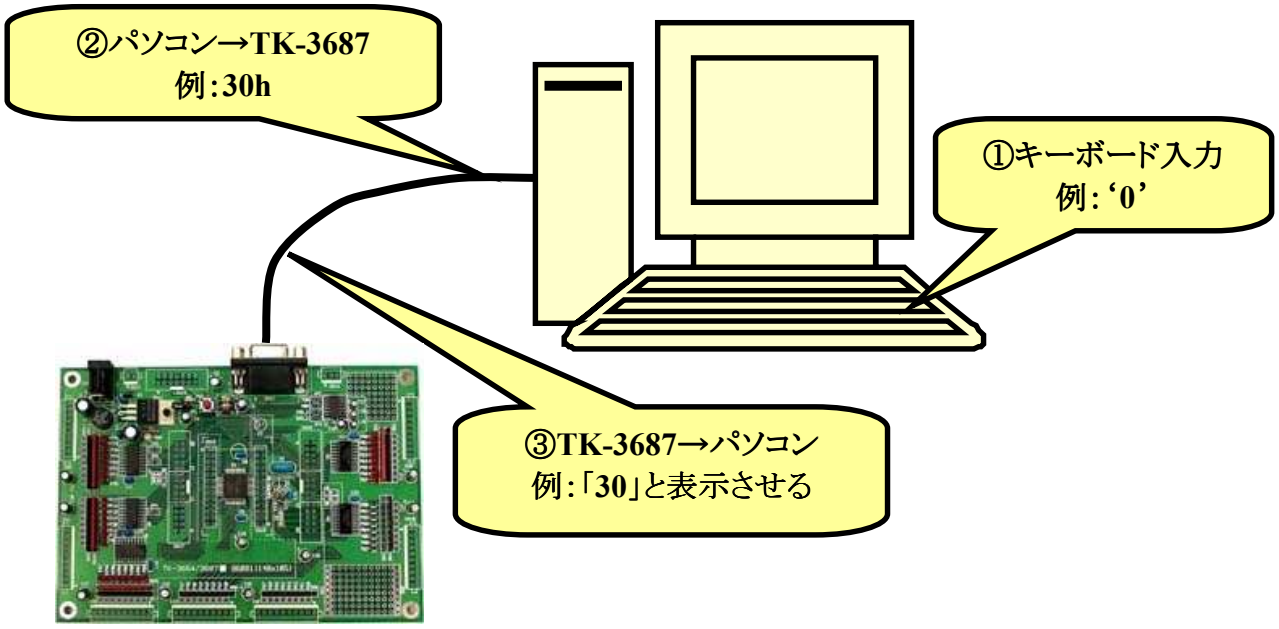
大変なことは専用のパーツにおまかせしましょう，というのがスマートな方法です。H8/3687 にはシリアル通信用の I/O が内蔵されています。「シリアルコミュニケーションインタフェース 3（SCI3）」と呼ばれています。SCI3 は調歩同期式シリアル通信以外にも対応できるように作られています。詳しくは，「H8/3687 グループ ハードウェアマニュアル」（以降ハードウェアマニュアル）の 16-1 ページから説明されていますので，ぜひお読みください。I/O ポートにくらべると SCI3 の使い方は最初は難しく感じるのですが，わかってしまうとそれほどでもありません。しかも I/O の使い方の基本が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

4. 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-3687 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. 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。

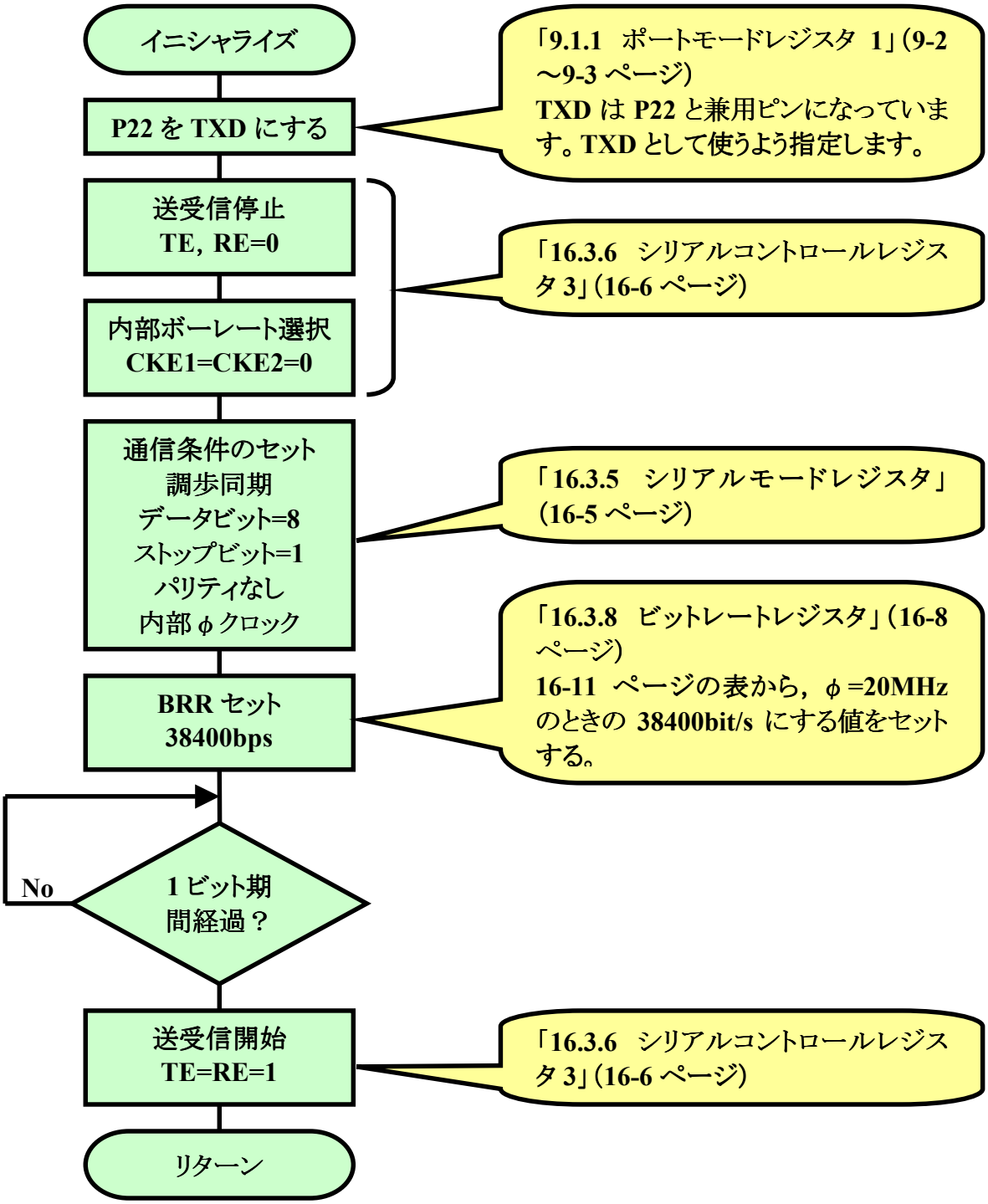
ビット/秒（ボーレート）・・・・・・・・・・・ 38400bps（38400bit/s，38400 ボー）

データビット・・・・・・・・・・・・・・・・・・・ 8 ビット

パリティ・・・・・・・・・・・・・・・・・・・・・・・ なし

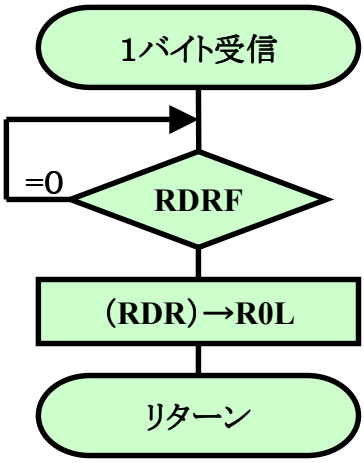
ストップビット・・・・・・・・・・・・・・・・・・ 1 ビット

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。



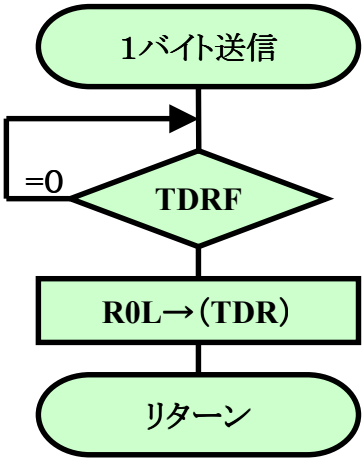
次に受信動作について考えてみましょう。当然ながら，TK-3687 はパソコンのキーがいつ押されるかわかりません。もっとも，受信動作そのものは SCI3 が自動的に行なってくれます。そして，データを受信したかどうかしらせるステータスが用意されています。というわけで，マイコンはそのステータスを見て，受信していたらデータを読み込みます。

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しらせ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，‘TDRF’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.src                                    |
;   DATE        :Fri, Jan 28, 2005                                    |
;   DESCRIPTION :Main Program                                         |
;   CPU TYPE    :H8/3687                                              |
;                                                                     |
;   This file is programed by TOYO-LINX Co.,Ltd / yKikuchi            |
;                                                                     |
;---------------------------------------------------------------------

        .export     _main

;*********************************************************************
;    定数定義
;*********************************************************************
SMR     .equ    h'FFA8    ;ｼﾘｱﾙﾓｰﾄﾞﾚｼﾞｽﾀ
BRR     .equ    h'FFA9    ;ﾋﾞｯﾄﾚｰﾄﾚｼﾞｽﾀ
SCR3    .equ    h'FFAA    ;ｼﾘｱﾙｺﾝﾄﾛｰﾙﾚｼﾞｽﾀ 3
TDR     .equ    h'FFAB    ;ﾄﾗﾝｽﾐｯﾄﾃﾞｰﾀﾚｼﾞｽﾀ
SSR     .equ    h'FFAC    ;ｼﾘｱﾙｽﾃｰﾀｽﾚｼﾞｽﾀ
RDR     .equ    h'FFAD    ;ﾚｼｰﾌﾞﾃﾞｰﾀﾚｼﾞｽﾀ

PMR1    .equ    h'FFE0    ;ﾎﾟｰﾄﾓｰﾄﾞﾚｼﾞｽﾀ 1

;*********************************************************************
;    メイン
;*********************************************************************
        .section P,CODE,LOCATE=H'ea00
_main:
;----- イニシャライズ ------------------------------------------------
        bsr     init_sci3       ;SCI3 ｲﾆｼｬﾗｲｽﾞ
        mov.b   #H'0d,r0l       ;改行
        bsr     txone

;----- メインループ --------------------------------------------------
_main_01:
        bsr     rxone           ;1ﾊﾞｲﾄ受信,受信ﾃﾞｰﾀ=R0L

        bsr     hex2asc         ;ｱｽｷｰｺｰﾄﾞ変換,R0L→R1

        mov.b   r1h,r0l         ;上位ﾊﾞｲﾄ送信
        bsr     txone
        mov.b   r1l,r0l         ;下位ﾊﾞｲﾄ送信
        bsr     txone
        mov.b   #H'0d,r0l       ;改行
        bsr     txone

        bra     _main_01        ;ﾒｲﾝﾙｰﾌﾟの先頭に戻る
```

```
;*******************************************************************************
;    SCI3 イニシャライズ
;*******************************************************************************
MHz       .equ D'20             ;X=20MHz
BAUD      .equ 38400            ;9600,19200,38400
BITR      .equ (MHz*D'1000000)/(BAUD*D'32)-1
WAIT_1B   .equ (MHz*D'1000000)/D'6/BAUD

init_sci3:
          bset      #1,@PMR1            ;P22はTXDとして使う
          xor.b     r0l,r0l
          mov.b     r0l,@SCR3           ;TE,REを0にｸﾘｱ/CKE1=CKE0=0,内部ﾎﾞｰﾚｰﾄ
          mov.b     r0l,@SMR            ;調歩同期,8ﾋﾞｯﾄ長,ｽﾄｯﾌﾟﾋﾞｯﾄ=1,ﾊﾟﾘﾃｨなし
          mov.b     #BITR,r0l
          mov.b     r0l,@BRR            ;通信ﾚｰﾄに対応する値をﾗｲﾄ
          mov.w     #WAIT_1B,r0         ;1ﾋﾞｯﾄ期間待つ
init_sci3_00:
          dec.w     #1,r0
          bne       init_sci3_00
          mov.b     #H'30,r0l           ;送受信ｲﾈｰﾌﾞﾙ,割込みﾃﾞｨｾｰﾌﾞﾙ
          mov.b     r0l,@SCR3
          rts

;*******************************************************************************
;    1バイト受信
;         R0L = 受信データ
;*******************************************************************************
rxone:
          btst      #6,@SSR             ;受信ﾃﾞｰﾀあり?
          beq       rxone               ; No→ｼﾞｬﾝﾌﾟ
          mov.b     @RDR,r0l            ;受信ﾃﾞｰﾀをﾘｰﾄﾞ
          rts

;*******************************************************************************
;    1バイト送信
;         R0L = 送信データ
;*******************************************************************************
txone:
          btst      #7,@SSR             ;送信可能?
          beq       txone               ; No→ｼﾞｬﾝﾌﾟ
          mov.b     r0l,@TDR            ;送信ﾃﾞｰﾀをﾗｲﾄ
          rts

;*******************************************************************************
;    アスキーコード変換
;         R0L = HEX → R1 = ASCII
;*******************************************************************************
hex2asc:
          mov.b     r0l,r1h
          shlr.b    r1h
          shlr.b    r1h
          shlr.b    r1h
          shlr.b    r1h
```

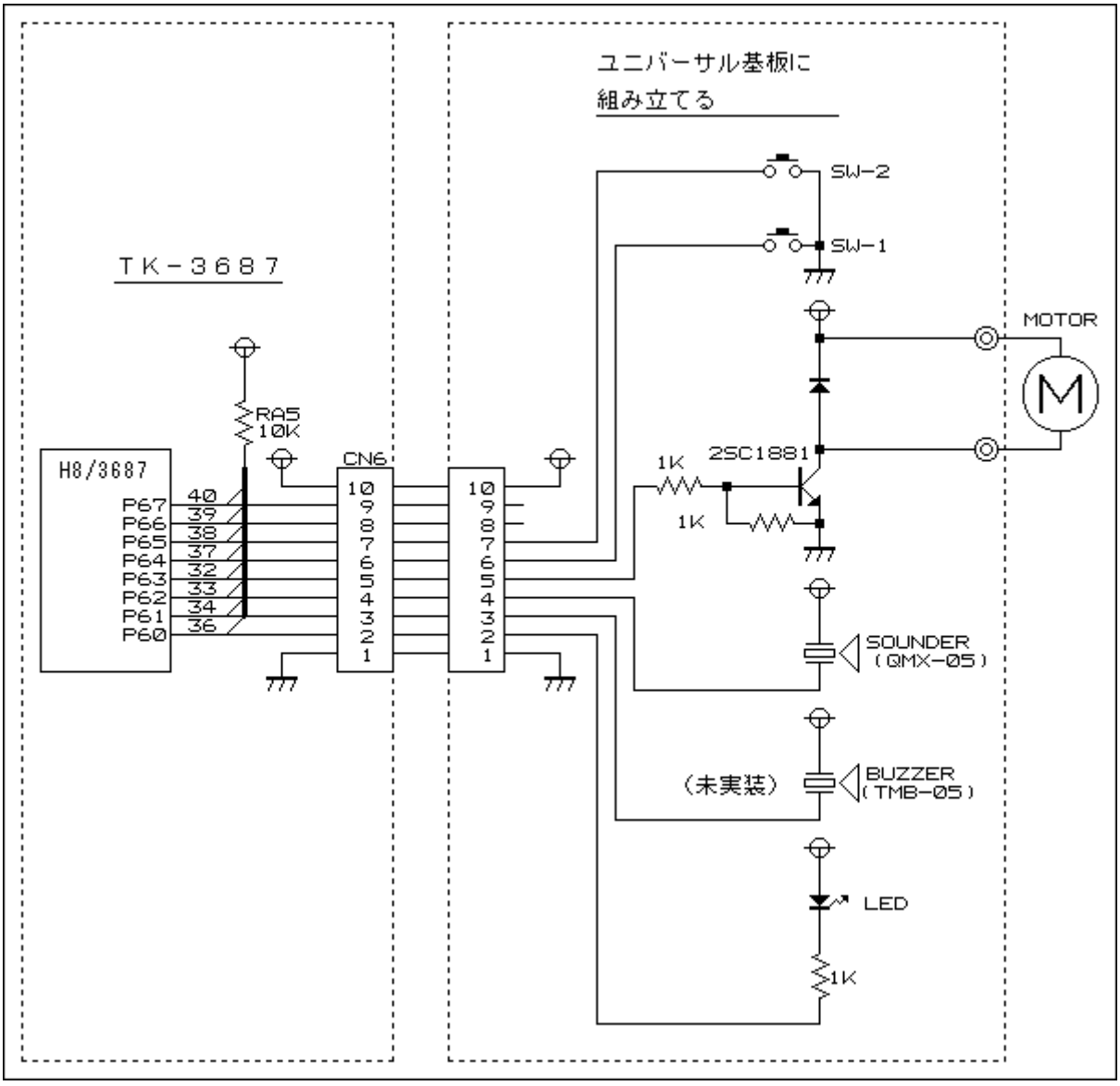
```
            add.b    #H'30,r1h
            cmp.b    #H'3a,r1h
            blo      hex2asc_01
            add.b    #H'07,r1h
hex2asc_01:
            mov.b    r0l,r1l
            and.b    #H'0f,r1l
            add.b    #H'30,r1l
            cmp.b    #H'3a,r1l
            blo      hex2asc_02
            add.b    #H'07,r1l
hex2asc_02:
            rts

;************************************************************************
;    ワークエリア
;************************************************************************
            .section B,DATA,LOCATE=H'f780

;------------------------------------------------------------------------
            .end
```

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。

第9章 I/O ポートのちょっと高度な応用例

1. メロディをかなでてみよう
2. DC モータの回転数制御

第7章で作った回路でI/Oポートを使った応用例を考えてみましょう。もう一度回路図を思い出しておきましょうか。



1. メロディをかなでてみよう

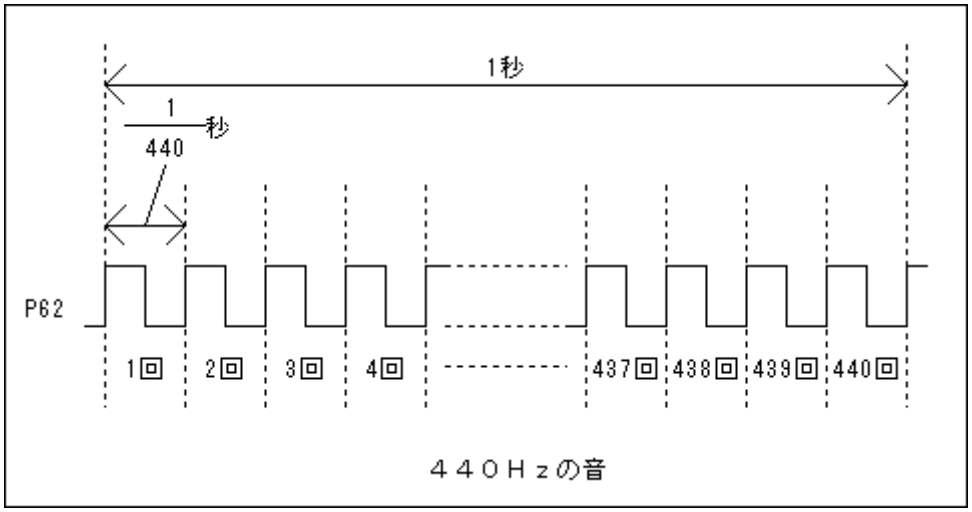
サウンダでメロディをかなでてみましょう。CD-ROM から

‘IoPort_sounder. mot’

をダウンロードして実行して下さい。簡単な回路のわりには，ちゃんとしたメロディが流れてくると思いませんか。

どのようにすればメロディを流すことができるでしょうか。考えてみましょう。

メロディにまず必要なのは，ドレミファソラシド，つまり音階です。音階は物理的には音の周波数のことです。で，周波数とは何かといえば，1 秒間に何回くりかえすか，ということです（単位は Hz：ヘルツ）。このプログラムの基準音はラ（A）ですが，周波数は 440Hz になります。今回使ったサウンダという部品は，ある周波数のパルス信号を加えると，その周波数の音を出します。というわけで，出したい音の周波数のパルス信号を P62 から出力することで，特定の音階の音を出しています。あとは，周波数をいろいろ変えればメロディになっていきます。例えば，440Hz の音を出すときには次の図のように P62 からパルス信号を出力します。



もう一つ，メロディの重要な要素は音符の長さです。音楽の授業を思い出してください。楽譜を見ればわかるように同じ音階でも，全音符，2 分音符，4 分音符，8 分音符…とだんだん音の長さが短くなっていきます。どれくらい短くなるかといいますと，半分ずつになっていきます（例：4 分音符二つで 2 分音符一つの長さ）。普通は曲の速さによって基準となる長さを変えていきます。楽譜の左上に「♩=120」という記号があるのを見たことがあるでしょうか。これは 1 分間に 4 分音符が 120 個になる速さで演奏する，という意味です。このプログラムはこれを採用しました。というわけで，全音符が 2 秒，2 分音符が 1 秒，4 分音符が 0.5 秒，8 分音符が 0.25 秒…の長さになります。

音符の長さでもう一つ重要なのは，音符の長さの全部で音を出すか，ということです。例えば，同じ音階の 4 分音符が 2 つ並んでいるのと，2 分音符との違いです。トータルの音の長さ（1 秒）は同じですが，4 分音符が 2 つのときは明らかに二つの音です。音符の長さの全部で音を出してしまうと 2 つの音がつながってしまって一つの音のように聞こえてしまいます。それで，音符の長さのうち，16 分の 15 音を出して，最後の 16 分の 1 は無音にします。ただ，スラーやタイのときは次の音とつなげたいので，そのときは音符の長さの全部で音を出すようにします。また，付点音符は二つの音で指定します。例えば付点 4 部音符は，4 分音符と 8 分音符の二つの音として扱います。で，この 4 分音符は音符の長さの全部で音を出すよう指定して，次の 8 分音符と音をつないで一つの音にします。

メロディの重要な部分の最後は休止です。これは無音状態をどれくらい続けるかで指定します。無音の長さは音符の長さと同じ方法で指定します。

プログラム中では全音符の音階をテーブルとして持たせています。また，楽譜もテーブルで持たせています。この二つのテーブルを使って，メロディをかなでていきます。詳しくはリストをご覧ください。

```
;-------------------------------------------------------------------------
;                                                                        |
;   FILE        :IoPort_sounder.src                                      |
;   DATE        :Tue, Jan 11, 2005                                       |
;   DESCRIPTION :Main Program                                            |
;   CPU TYPE    :H8/3687                                                 |
;                                                                        |
;   This file is programed by TOYO-LINX Co.,Ltd / yKikuchi               |
;                                                                        |
;-------------------------------------------------------------------------

        .export     _main

;************************************************************************
;   定数定義
;************************************************************************
PDR6    .equ    h'FFD9    ;ﾎﾟｰﾄﾃﾞｰﾀﾚｼﾞｽﾀ 6
PCR6    .equ    h'FFE9    ;ﾎﾟｰﾄｺﾝﾄﾛｰﾙﾚｼﾞｽﾀ 6

;************************************************************************
;   メイン
;************************************************************************
        .section P,CODE,LOCATE=H'ea00
_main:
;----- イニシャライズ ----------------------------------------------------
        mov.b   #B'00000000,r0l   ;ポートのイニシャライズ
        mov.b   r0l,@PDR6         ; あらかじめブザーをオフにする
        mov.b   #B'00000100,r0l   ;
        mov.b   r0l,@PCR6         ; bit0-1,3-7 入力 : bit2 出力

;----- メインループ ------------------------------------------------------
_main_00:
        mov.l   #GakufuTbl,er6
_main_01:
        mov.w   @er6,r0           ;音符データを取得
        adds    #2,er6
        cmp.w   #H'ffff,r0        ;終了?
        beq     _main_02          ; Yes
        bsr     sound             ;音を出す
        bra     _main_01
_main_02:
        bra     _main_00

;************************************************************************
;   音
;       R0H : 音の長さ bit6-0  00h-全音符,01h-2部音符,02h-4分音符
;                              03h-8分音符,04h-16分音符
;                      bit7    0-次の音符と区別する(通常)
;                              1-次の音符とつなげる(スラー,タイ,符点音符)
;       R0L : 音階     音階テーブル参照,( )内の数字で表す
```

```
;                       ただし，00hは休止
;*******************************************************************************
sound:
        mov.l   #0,er2
        mov.b   r0l,r2l
        beq     sound_01            ;休止→ジャンプ
        bra     sound_02
sound_01:
        mov.b   #H'80,r2l           ;休止は基準音(A,440Hz)の長さで設定する
sound_02:
        mov.l   #H'80,er3
        sub.l   er3,er2             ;er2=er2-er3
        shal.l  er2
        shal.l  er2
        add.l   #0nkaiTbl,er2
        mov.l   @er2,er1            ;e1=全音符の長さ,r1=音階の半周期の長さ

sound_10:
        mov.b   r0h,r2h             ;e1に音符の長さをセットする
        and.b   #H'7f,r2h
        beq     sound_12
sound_11:
        shlr.w  e1
        dec.b   r2h
        bne     sound_11
sound_12:

        mov.w   e1,e2               ;e2にe1の1/16の長さをセットする
        shlr.w  e2
        shlr.w  e2
        shlr.w  e2
        shlr.w  e2

        mov.b   r0l,r0l             ;休止?
        beq     sound_30            ; Yes

;----- 発音 ----------------------------------------------------------
sound_20:
        btst    #7,r0h              ;次の音符につなげる?
        bne     sound_23            ; Yes
        sub.w   e2,e1               ;e1=e1-e2(e1:実際の音の長さ)
sound_21:
        bset    #2,@PDR6            ;音符の長さの15/16 音を出す
        mov.w   r1,r2
        bsr     wait_sound
        bclr    #2,@PDR6
        mov.w   r1,r2
        bsr     wait_sound
        dec.w   #1,e1
        bne     sound_21
sound_22:
        bclr    #2,@PDR6            ;音符の長さの1/16を無音にする
        mov.w   r1,r2
```

```
          bsr       wait_sound
          bclr      #2,@PDR6
          mov.w     r1,r2
          bsr       wait_sound
          dec.w     #1,e2
          bne       sound_22
          rts
sound_23:
          bset      #2,@PDR6          ;音符の長さ全部 音を出す
          mov.w     r1,r2
          bsr       wait_sound
          bclr      #2,@PDR6
          mov.w     r1,r2
          bsr       wait_sound
          dec.w     #1,e1
          bne       sound_23
          rts

;----- 休止 ----------------------------------------------------------
sound_30:
          bclr      #2,@PDR6          ;無音
          mov.w     r1,r2
          bsr       wait_sound
          bclr      #2,@PDR6
          mov.w     r1,r2
          bsr       wait_sound
          dec.w     #1,e1
          bne       sound_30
          rts

;----- ウェイト ------------------------------------------------------
wait_sound:
          dec.w     #1,r2             ;2クロック 合計6クロック=0.3μ秒
          bne       wait_sound        ;4クロック
          rts

;----- 音階テーブル --------------------------------------------------
;                   全音符の長さ / 半周期の長さ
;         (何周期くりかえすか) / ('wait_sound'内で何回くりかえすか)
;                              ↓    ↓
              .data.w       523,6370 ;(77h) C  ,ﾄﾞ ,261.63Hz
              .data.w       554,6013 ;(78h) C# ,ﾄﾞ#,277.18Hz
              .data.w       587,5675 ;(79h) D  ,レ  ,293.66Hz
              .data.w       622,5357 ;(7Ah) D# ,レ# ,311.13Hz
              .data.w       659,5056 ;(7Bh) E  ,ミ  ,329.63Hz
              .data.w       698,4772 ;(7Ch) F  ,ﾌｧ ,349.23Hz
              .data.w       740,4505 ;(7Dh) F# ,ﾌｧ#,369.99Hz
              .data.w       784,4252 ;(7Eh) G  ,ソ  ,392.00Hz
              .data.w       831,4013 ;(7Fh) G# ,ソ# ,415.30Hz
OnkaiTbl:     .data.w       880,3788 ;(80h) A  ,ﾗ  ,440.00Hz    ←基準
              .data.w       932,3575 ;(81h) A# ,ﾗ# ,466.16Hz
              .data.w       988,3375 ;(82h) B  ,ｼ  ,493.88Hz
              .data.w       1047,3185;(83h) C  ,ﾄﾞ ,523.25Hz
```

```
                .data.w         1109,3006;(84h) C# ,ﾄﾞ#,554.37Hz
                .data.w         1175,2838;(85h) D  ,レ  ,587.33Hz
                .data.w         1245,2678;(86h) D# ,レ# ,622.25Hz
                .data.w         1319,2528;(87h) E  ,ミ  ,659.26Hz

;*****************************************************************************
;     楽譜テーブル
;          上位8ビット(bit15-8):音の長さ
;               bit14-8 00h-全音符,01h-2部音符,02h-4分音符
;                       03h-8分音符,04h-16分音符
;               bit15   0-次の音符と区別する(通常)
;                       1-次の音符とつなげる(スラー,タイ,符点音符)
;          下位8ビット(bit7 -0):音階
;               音階テーブル参照,( )内の数字で表す。ただし, 00hは休止。
;*****************************************************************************
GakufuTbl:
;-----  「春の小川」   作曲：岡野貞一 ------------------------------------
                .data.w         H'827b,H'027e,H'0280,H'027e
                .data.w         H'027b,H'027e,H'0283,H'0283
                .data.w         H'0280,H'0280,H'027e,H'027b
                .data.w         H'0277,H'0279,H'027b,H'0200

                .data.w         H'827b,H'027e,H'0280,H'027e
                .data.w         H'027b,H'027e,H'0283,H'0283
                .data.w         H'0280,H'0280,H'027e,H'027b
                .data.w         H'0279,H'027b,H'0277,H'0200

                .data.w         H'8279,H'027b,H'0279,H'027e
                .data.w         H'0280,H'0280,H'027e,H'0280
                .data.w         H'0283,H'0283,H'0282,H'0280
                .data.w         H'027e,H'027e,H'027b,H'0200

                .data.w         H'827b,H'027e,H'0280,H'027e
                .data.w         H'027b,H'027e,H'0283,H'0283
                .data.w         H'0280,H'0280,H'027e,H'027b
                .data.w         H'0279,H'027b,H'0277,H'0200
;-----------------------------------------------------------------------------
                .data.w         H'0000    ;曲間(全音符の長さの休止)

;-----  「朧月夜」   作曲：岡野貞一 ------------------------------------
                .data.w         H'037d,H'037d
                .data.w         H'8279,H'0379,H'037b,H'037d,H'0380
                .data.w         H'0380,H'0382,H'0280,H'027b
                .data.w         H'827d,H'037d,H'0379,H'037b,H'0380
                .data.w         H'017d,H'0380,H'0380

                .data.w         H'827d,H'037d,H'037e,H'0380,H'0385
                .data.w         H'0385,H'0387,H'0285,H'0280
                .data.w         H'8282,H'0382,H'037d,H'037b,H'037b
                .data.w         H'0179,H'0380,H'0380

                .data.w         H'8285,H'0385,H'0385,H'0385,H'0387
                .data.w         H'0385,H'0382,H'0280,H'0380,H'037d
```

```
                .data.w         H'8280,H'0380,H'0382,H'037d,H'037d
                .data.w         H'017b,H'8379,H'037b

                .data.w         H'827d,H'037d,H'0379,H'037d,H'037e
                .data.w         H'0380,H'0385,H'0282,H'0280
                .data.w         H'8282,H'0382,H'037d,H'037b,H'037b
                .data.w         H'0179
;-------------------------------------------------------------------------
                .data.w         H'0000    ;曲間(全音符の長さの休止)

;-------------------------------------------------------------------------
                .data.w         H'ffff    ;テーブル終了マーク

;*************************************************************************
;    ワークエリア
;*************************************************************************
        .section B,DATA,LOCATE=H'f780

;-------------------------------------------------------------------------
        .end
```

このプログラムに，スイッチが押されたら曲を変更する，という機能を追加してみましょう。CD-ROM から

‘IoPort_sounder2. mot’

をダウンロードして実行して下さい。

このプログラムには全部で 6 曲入っています。P64 につながっているスイッチ-1 を押すと 1 曲目の音楽がスタートします。さらにスイッチ-1 を押すと次の曲に移ります。また，P65 につながっているスイッチ-2 を押すと前の曲がスタートします。なお，1 曲演奏が終了すると音楽はそこでストップします。スイッチを押すと次の曲，もしくは前の曲がスタートします。

プログラムリストは次のとおりです。

```
;-------------------------------------------------------------------------
;                                                                        |
;   FILE        :IoPort_sounder2.src                                     |
;   DATE        :Wed, Jan 19, 2005                                       |
;   DESCRIPTION :Main Program                                            |
;   CPU TYPE    :H8/3687                                                 |
;                                                                        |
;   This file is programed by TOYO-LINX Co.,Ltd / yKikuchi               |
;                                                                        |
;-------------------------------------------------------------------------

        .export     _main

;*************************************************************************
;    定数定義
;*************************************************************************
PDR6    .equ    h'FFD9    ;ﾎﾟｰﾄﾃﾞｰﾀﾚｼﾞｽﾀ 6
PCR6    .equ    h'FFE9    ;ﾎﾟｰﾄｺﾝﾄﾛｰﾙﾚｼﾞｽﾀ 6

;*************************************************************************
;    メイン
;*************************************************************************
        .section P,CODE,LOCATE=H'ea00
_main:
;----- イニシャライズ ---------------------------------------------------
        mov.b   #B'00000001,r0l   ;ポートのイニシャライズ
        mov.b   r0l,@PDR6         ; あらかじめサウンダとLEDをオフにする
        mov.b   #B'00000101,r0l   ;
        mov.b   r0l,@PCR6         ; bit1,3-7 入力 : bit0,2 出力

        mov.b   #0,r0l            ;メモリのイニシャライズ
        mov.b   r0l,@Music
        mov.b   r0l,@SwData
        mov.b   r0l,@FirstSw

;----- メインループ -----------------------------------------------------
_MainLoop:
```

```
;..... スイッチ入力 ..................................................
swin:
        mov.b   @PDR6,r0l           ;スイッチリード
        not.b   r0l
        and.b   #B'00110000,r0l
        mov.b   r0l,@FirstSw
        bne     swin_01             ;いずれかオン
        bra     swin_02             ;すべてオフ
swin_01:
        bsr     wait_10:16          ;チャタリング除去(10msウェイト)
        mov.b   @PDR6,r0l           ;スイッチリード（ダブルリード）
        not.b   r0l
        and.b   #B'00110000,r0l
        mov.b   @FirstSw,r0h
        cmp.b   r0h,r0l             ;最初のスイッチと同じ?
        beq     swin_03             ; Yes
swin_02:
        mov.b   #0,r0l              ;スイッチの状態, オフにする
        mov.b   r0l,@SwData
        bra     swin_06
swin_03:
        mov.b   @SwData,r0h         ;r0h=前回のスイッチの状態
        mov.b   r0l,@SwData         ;r0l=今のスイッチの状態
        xor.b   r0h,r0l             ;r0l=変化があったスイッチ
        not.b   r0h
        and.b   #B'00110000,r0h
        and.b   r0h,r0l             ;r0l=0→1の変化があったスイッチ
        btst    #4,r0l              ;P64のスイッチオン?
        bne     swin_04             ; Yes
        btst    #5,r0l              ;P65のスイッチオン?
        bne     swin_05             ; Yes
        bra     swin_06
swin_04:
        bsr     next_music          ;次の曲へ
        bra     swin_06
swin_05:
        bsr     before_music        ;前の曲へ
swin_06:

;..... 演奏 ..........................................................
play:
        mov.b   @Music,r0l
        btst #7,r0l
        beq     play_02             ;演奏しないときはジャンプ

        mov.l   @GakufuPnt,er1      ;音符データを取得
        mov.w   @er1,r0
        adds    #2,er1
        mov.l   er1,@GakufuPnt
        cmp.w   #H'ffff,r0          ;曲終了?
        beq     play_01             ; Yes
        bsr     sound:16            ;音を出す
        bra     play_02
```

```
play_01:
        mov.b   @Music,r0l          ;曲の停止
        bclr    #7,r0l
        mov.b   r0l,@Music
play_02:

;..... メインループ先頭にジャンプ .....................................
        bra     _MainLoop

;*******************************************************************************
;   次の曲へ
;*******************************************************************************
next_music:
        mov.b   @Music,r0l
        and.b   #B'01111111,r0l
        cmp.b   #H'07,r0l
        beq     next_music_01
        inc.b   r0l                 ;次の曲をセット
        and.b   #B'00000111,r0l
        bset    #7,r0l              ;演奏開始セット
        mov.b   r0l,@Music
        bsr     choice_music        ;楽譜の選択
next_music_01:
        rts

;*******************************************************************************
;   前の曲へ
;*******************************************************************************
before_music:
        mov.b   @Music,r0l
        and.b   #B'01111111,r0l
        beq     before_music_01
        dec.b   r0l                 ;前の曲をセット
        and.b   #B'00000111,r0l
        bset    #7,r0l              ;演奏開始セット
        mov.b   r0l,@Music
        bsr     choice_music        ;楽譜の選択
before_music_01:
        rts

;*******************************************************************************
;   楽譜の選択
;*******************************************************************************
choice_music:
        mov.b   @Music,r0l
        and.b   #B'00000111,r0l
        cmp.b   #H'07,r0l
        beq     choice_music_07
        cmp.b   #H'06,r0l
        beq     choice_music_06
        cmp.b   #H'05,r0l
        beq     choice_music_05
        cmp.b   #H'04,r0l
```

```
        beq     choice_music_04
        cmp.b   #H'03,r0l
        beq     choice_music_03
        cmp.b   #H'02,r0l
        beq     choice_music_02
        cmp.b   #H'01,r0l
        beq     choice_music_01
choice_music_00:
        mov.l   #GakufuTbl00,er0
        bra     choice_music_ff
choice_music_01:
        mov.l   #GakufuTbl01,er0
        bra     choice_music_ff
choice_music_02:
        mov.l   #GakufuTbl02,er0
        bra     choice_music_ff
choice_music_03:
        mov.l   #GakufuTbl03,er0
        bra     choice_music_ff
choice_music_04:
        mov.l   #GakufuTbl04,er0
        bra     choice_music_ff
choice_music_05:
        mov.l   #GakufuTbl05,er0
        bra     choice_music_ff
choice_music_06:
        mov.l   #GakufuTbl06,er0
        bra     choice_music_ff
choice_music_07:
        mov.l   #GakufuTbl07,er0
choice_music_ff:
        mov.l   er0,@GakufuPnt
        rts

;*******************************************************************************
;   ウェイト
;*******************************************************************************
wait_10:
        mov.l   #33333,er0          ;10ms
wait_01:
        dec.l   #1,er0
        bne     wait_01
        rts

;*******************************************************************************
;   音
;       R0H : 音の長さ bit6-0  00h-全音符,01h-2部音符,02h-4分音符
;                              03h-8分音符,04h-16分音符
;                      bit7    0-次の音符と区別する(通常)
;                              1-次の音符とつなげる(スラー,タイ,付点音符)
;       R0L : 音階     音階テーブル参照,( )内の数字で表す
;                      ただし, 00hは休止
;*******************************************************************************
```

```
sound:
        mov.l   #0,er2
        mov.b   r0l,r2l
        beq     sound_01            ;休止→ジャンプ
        bra     sound_02
sound_01:
        mov.b   #H'80,r2l           ;休止は基準音(A,440Hz)の長さで設定する
sound_02:
        mov.l   #H'80,er3
        sub.l   er3,er2             ;er2=er2-er3
        shal.l  er2
        shal.l  er2
        add.l   #OnkaiTbl,er2
        mov.l   @er2,er1            ;e1=全音符の長さ,r1=音階の半周期の長さ

sound_10:
        mov.b   r0h,r2h             ;e1に音符の長さをセットする
        and.b   #H'7f,r2h
        beq     sound_12
sound_11:
        shlr.w  e1
        dec.b   r2h
        bne     sound_11
sound_12:

        mov.w   e1,e2               ;e2にe1の1/16の長さをセットする
        shlr.w  e2
        shlr.w  e2
        shlr.w  e2
        shlr.w  e2

        mov.b   r0l,r0l             ;休止?
        beq     sound_30            ; Yes

;----- 発音 ----------------------------------------------------------
sound_20:
        btst    #7,r0h              ;次の音符につなげる?
        bne     sound_23            ; Yes
        sub.w   e2,e1               ;e1=e1-e2(e1:実際の音の長さ)
sound_21:
        bset    #2,@PDR6            ;音符の長さの15/16 音を出す
        bclr    #0,@PDR6            ;LEDオン
        mov.w   r1,r2
        bsr     wait_sound
        bclr    #2,@PDR6
        bset    #0,@PDR6            ;LEDオフ
        mov.w   r1,r2
        bsr     wait_sound
        dec.w   #1,e1
        bne     sound_21
sound_22:
        bclr    #2,@PDR6            ;音符の長さの1/16を無音にする
        bset    #0,@PDR6            ;LEDオフ
```

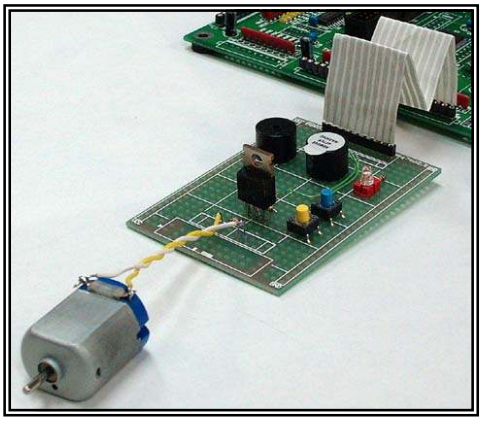
```
        mov.w   r1,r2
        bsr     wait_sound
        bclr    #2,@PDR6
        bset    #0,@PDR6        ;LEDオフ
        mov.w   r1,r2
        bsr     wait_sound
        dec.w   #1,e2
        bne     sound_22
        rts
sound_23:
        bset    #2,@PDR6        ;音符の長さ全部 音を出す
        bclr    #0,@PDR6        ;LEDオン
        mov.w   r1,r2
        bsr     wait_sound
        bclr    #2,@PDR6
        bset    #0,@PDR6        ;LEDオフ
        mov.w   r1,r2
        bsr     wait_sound
        dec.w   #1,e1
        bne     sound_23
        rts

;----- 休止 ----------------------------------------------------------
sound_30:
        bclr    #2,@PDR6        ;無音
        bset    #0,@PDR6        ;LEDオフ
        mov.w   r1,r2
        bsr     wait_sound
        bclr    #2,@PDR6
        bset    #0,@PDR6        ;LEDオフ
        mov.w   r1,r2
        bsr     wait_sound
        dec.w   #1,e1
        bne     sound_30
        rts

;----- ウェイト ------------------------------------------------------
wait_sound:
        dec.w   #1,r2           ;2クロック 合計6クロック=0.3μ秒
        bne     wait_sound      ;4クロック
        rts

;----- 音階テーブル --------------------------------------------------
;               全音符の長さ / 半周期の長さ
;       (何周期くりかえすか) / ('wait_sound'内で何回くりかえすか)
;                           ↓   ↓
            .data.w     494,6749 ;(76h) B  ,シ  ,246.94Hz
            .data.w     523,6370 ;(77h) C  ,ド  ,261.63Hz
            .data.w     554,6013 ;(78h) C# ,ド#,277.18Hz
            .data.w     587,5675 ;(79h) D  ,レ  ,293.66Hz
            .data.w     622,5357 ;(7Ah) D# ,レ# ,311.13Hz
            .data.w     659,5056 ;(7Bh) E  ,ミ  ,329.63Hz
            .data.w     698,4772 ;(7Ch) F  ,ファ ,349.23Hz
```

```
                .data.w         740,4505 ;(7Dh) F# ,ﾌｧ#,369.99Hz
                .data.w         784,4252 ;(7Eh) G  ,ｿ  ,392.00Hz
                .data.w         831,4013 ;(7Fh) G# ,ｿ# ,415.30Hz
OnkaiTbl:       .data.w         880,3788 ;(80h) A  ,ﾗ  ,440.00Hz    ←基準
                .data.w         932,3575 ;(81h) A# ,ﾗ# ,466.16Hz
                .data.w         988,3375 ;(82h) B  ,ｼ  ,493.88Hz
                .data.w         1047,3185;(83h) C  ,ﾄﾞ ,523.25Hz
                .data.w         1109,3006;(84h) C# ,ﾄﾞ#,554.37Hz
                .data.w         1175,2838;(85h) D  ,ﾚ  ,587.33Hz
                .data.w         1245,2678;(86h) D# ,ﾚ# ,622.25Hz
                .data.w         1319,2528;(87h) E  ,ﾐ  ,659.26Hz

;*******************************************************************************
;    楽譜テーブル
;         上位8ビット(bit15-8):音の長さ
;             bit14-8 00h-全音符,01h-2部音符,02h-4分音符
;                     03h-8分音符,04h-16分音符
;             bit15   0-次の音符と区別する(通常)
;                     1-次の音符とつなげる(スラー,タイ,符点音符)
;         下位8ビット(bit7 -0):音階
;             音階テーブル参照,( )内の数字で表す。ただし, 00hは休止。
;*******************************************************************************

;-------------------------------------------------------------------------
GakufuTbl00:
                .data.w         H'ffff    ;テーブル終了マーク

;-------------------------------------------------------------------------
GakufuTbl01:
                .data.w         H'0377,H'0379,H'037b,H'037c
                .data.w         H'037e,H'0380,H'0382,H'0383

                .data.w         H'ffff    ;テーブル終了マーク

;-------------------------------------------------------------------------
GakufuTbl02:
;「春の小川」  作曲：岡野貞一
                .data.w         H'827b,H'027e,H'0280,H'027e
                .data.w         H'027b,H'027e,H'0283,H'0283
                .data.w         H'0280,H'0280,H'027e,H'027b
                .data.w         H'0277,H'0279,H'027b,H'0200

                .data.w         H'827b,H'027e,H'0280,H'027e
                .data.w         H'027b,H'027e,H'0283,H'0283
                .data.w         H'0280,H'0280,H'027e,H'027b
                .data.w         H'0279,H'027b,H'0277,H'0200

                .data.w         H'8279,H'027b,H'0279,H'027e
                .data.w         H'0280,H'0280,H'027e,H'0280
                .data.w         H'0283,H'0283,H'0282,H'0280
                .data.w         H'027e,H'027e,H'027b,H'0200

                .data.w         H'827b,H'027e,H'0280,H'027e
```

```
                .data.w         H'027b,H'027e,H'0283,H'0283
                .data.w         H'0280,H'0280,H'027e,H'027b
                .data.w         H'0279,H'027b,H'0277,H'0200

                .data.w         H'ffff    ;テーブル終了マーク

;-----------------------------------------------------------------------
GakufuTbl03:
                .data.w         H'0383,H'0382,H'0380,H'037e
                .data.w         H'037c,H'037b,H'0379,H'0377

                .data.w         H'ffff    ;テーブル終了マーク

;-----------------------------------------------------------------------
GakufuTbl04:
;「朧月夜」   作曲：岡野貞一
                .data.w         H'037d,H'037d
                .data.w         H'8279,H'0379,H'037b,H'037d,H'0380
                .data.w         H'0380,H'0382,H'0280,H'027b
                .data.w         H'827d,H'037d,H'0379,H'037b,H'0380
                .data.w         H'017d,H'0380,H'0380

                .data.w         H'827d,H'037d,H'037e,H'0380,H'0385
                .data.w         H'0385,H'0387,H'0285,H'0280
                .data.w         H'8282,H'0382,H'037d,H'037b,H'037b
                .data.w         H'0179,H'0380,H'0380

                .data.w         H'8285,H'0385,H'0385,H'0385,H'0387
                .data.w         H'0385,H'0382,H'0280,H'0380,H'037d
                .data.w         H'8280,H'0380,H'0382,H'037d,H'037d
                .data.w         H'017b,H'8379,H'037b

                .data.w         H'827d,H'037d,H'0379,H'037d,H'037e
                .data.w         H'0380,H'0385,H'0282,H'0280
                .data.w         H'8282,H'0382,H'037d,H'037b,H'037b
                .data.w         H'0179

                .data.w         H'ffff    ;テーブル終了マーク

;-----------------------------------------------------------------------
GakufuTbl05:
                .data.w         H'0477,H'0479,H'047b,H'047c
                .data.w         H'047e,H'0480,H'0482,H'0483
                .data.w         H'0483,H'0482,H'0480,H'047e
                .data.w         H'047c,H'047b,H'0479,H'0477

                .data.w         H'ffff    ;テーブル終了マーク

;-----------------------------------------------------------------------
GakufuTbl06:
;「荒城の月」   作曲：滝廉太郎
                .data.w         H'037d,H'037d,H'0382,H'0384,H'0385,H'0384,H'0282
                .data.w         H'037e,H'037e,H'037d,H'037c,H'027d,H'0200
```

```
                .data.w         H'037d,H'037d,H'0382,H'0384,H'0385,H'0384,H'0282
                .data.w         H'037e,H'037B,H'837d,H'047d,H'047d,H'0276,H'0200

                .data.w         H'0379,H'0379,H'0378,H'0376,H'837e,H'047e,H'047e,H'027d
                .data.w         H'037b,H'037d,H'837e,H'047e,H'047e,H'027d,H'0200

                .data.w         H'037d,H'037d,H'0382,H'0384,H'0385,H'0384,H'0282
                .data.w         H'037e,H'037B,H'837d,H'047d,H'047d,H'0276,H'0200

                .data.w         H'ffff    ;テーブル終了マーク

;-----------------------------------------------------------------------
GakufuTbl07:
                .data.w         H'ffff    ;テーブル終了マーク

;***********************************************************************
;    ワークエリア
;***********************************************************************
        .section B,DATA,LOCATE=H'f780
GakufuPnt       .res.l    1             ;楽譜のポインタアドレス
Music           .res.b    1             ;音楽
                                        ; bit0-6 : 曲目
                                        ; bit7   : 0-停止/1-演奏

SwData          .res.b    1             ;現在のスイッチデータ
FirstSw         .res.b    1             ;スイッチのファーストリード

;-----------------------------------------------------------------------
        .end
```

2. DC モータの回転数制御



次は DC モータを回してみましょう。単純に回すだけならマイコンは必要ありませんが，回転数を自在に変化させるとなるとマイコンの出番です。今回はスイッチ-1 を押すたびに，

停止→低速→中速→高速

スイッチ-2 を押すたびに

高速→中速→低速→停止

と回転速度が変化するプログラムを考えてみましょう。

CD-ROM から

‘IoPort_motor. mot’

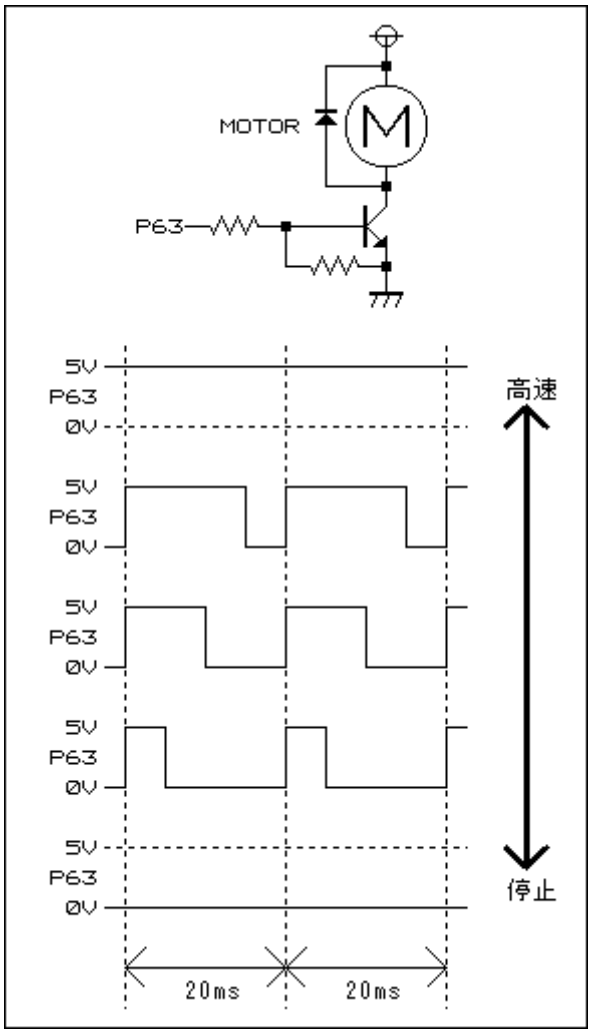
をダウンロードして実行して下さい。アップスイッチを押すたび速くなり，ダウンスイッチを押すたびに遅くなって停止すれば OK です。

さて，問題はどうやって DC モータの回転数を変化させるかです。今回は PWM という方法を使います。

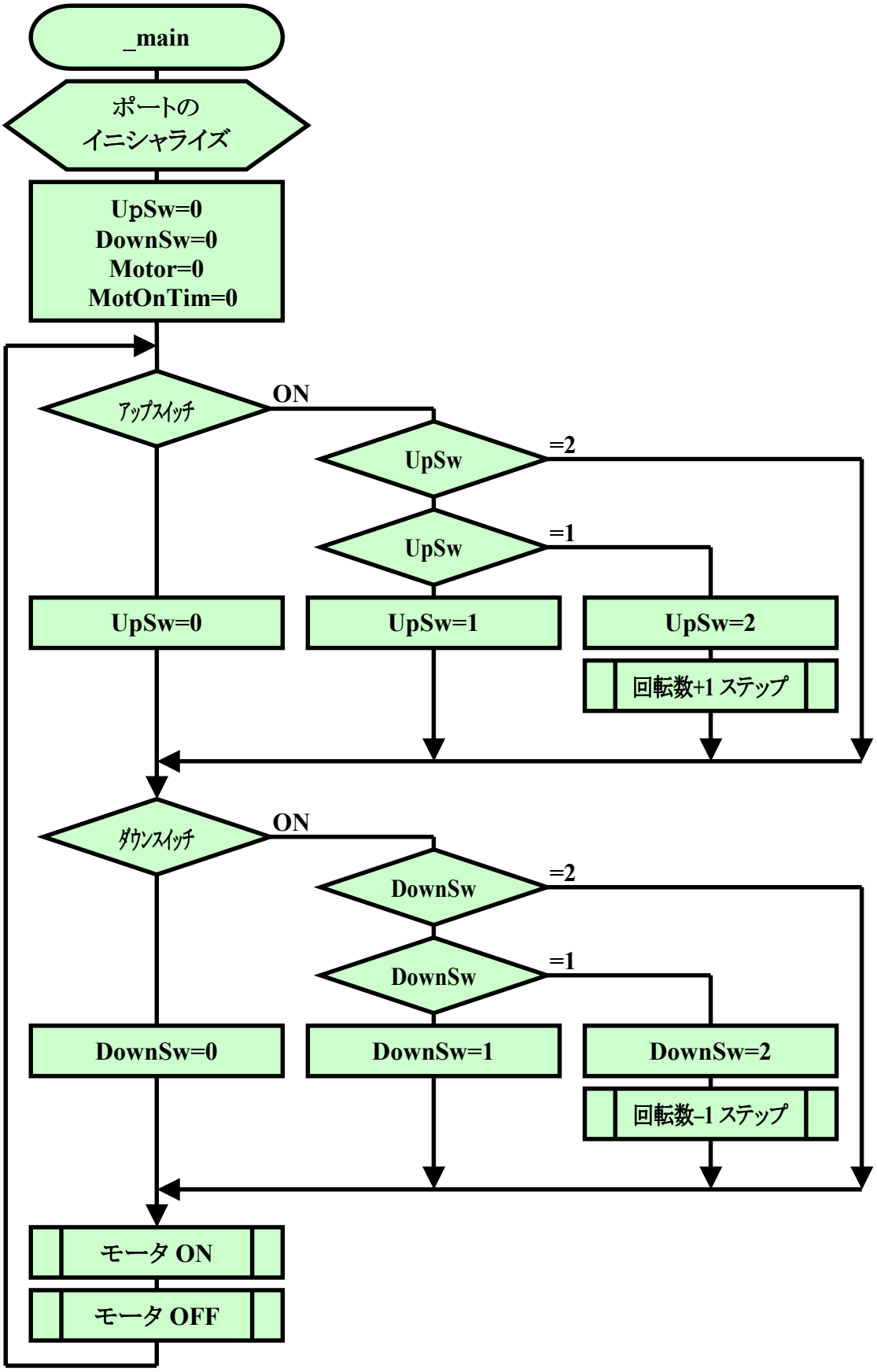
DC モータは電圧をかけると回転し，電圧をかけないと止まります。しかし，これでは回転するかしないかの 2 種類で，モータの回転数を変化させることはできません。

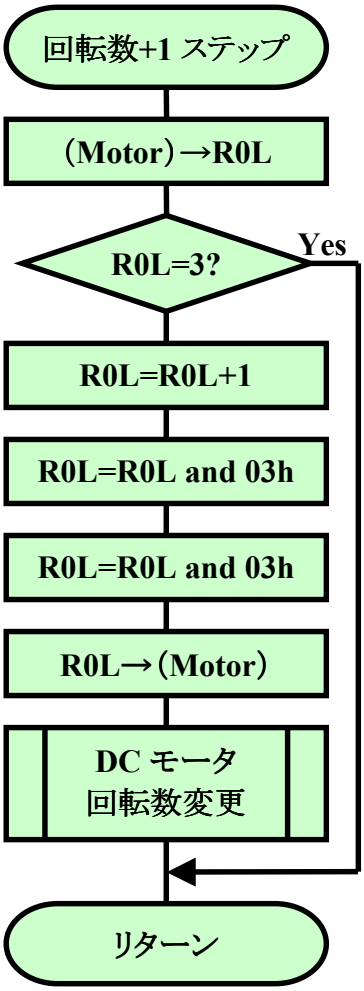
実のところ DC モータは電圧をかけたからといってその瞬間すぐに 100％の速さで回転するわけではなく，また，電圧をかけるのをやめたからといってすぐに止まるものでもありません。短時間ではありますが，徐々に回転が変化してやがて 100％の速さで回転したり止まったりします。では，電圧のオン・オフをすばやく繰り返したらどうなるでしょうか。電圧が加わると，徐々にモータが回りだします。100％に向かって回転数が上がっていきますが，すぐに電圧がオフになります。すると今度は徐々にモータが止まろうとします。止まる前に再び電圧が加わるので，また回りだします。これを繰り返せば一定の回転数でモータを回すことができます。また，オン・オフの時間を変化させれば任意の回転数でまわすこともできます。

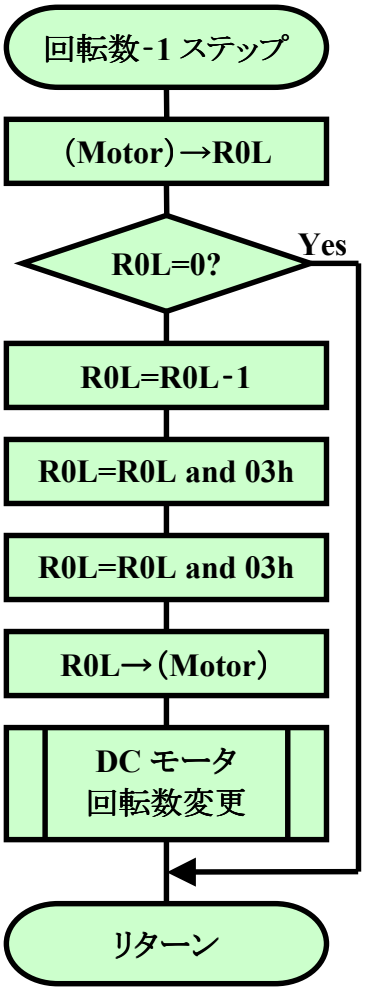
通常，オンとオフを足した時間を一定にし，オンとオフの比（デューティ）を変化させます。このような制御方法を PWM（Pulse Width Modulation：パルス幅変調）といいます。右のタイミングチャートで見てみましょう。イメージはつかめたでしょうか。

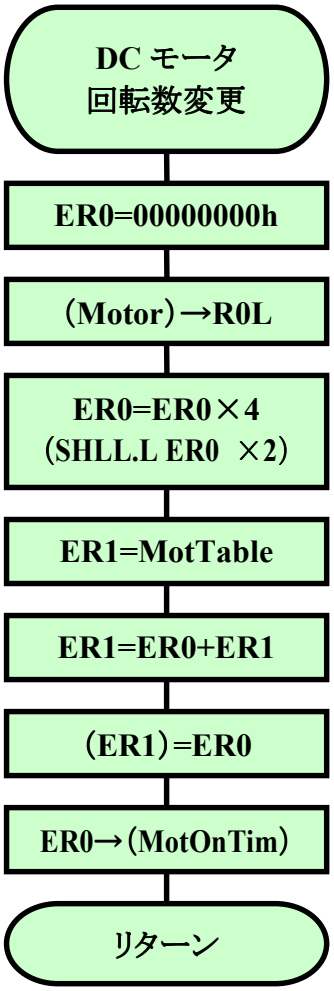


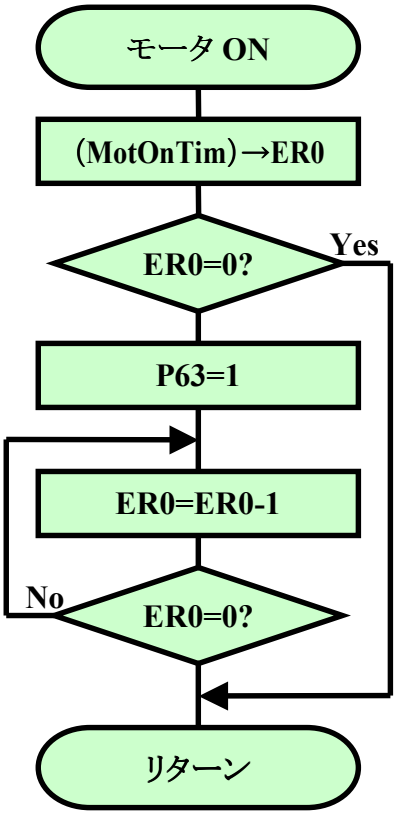
それではフローチャートを考えてみましょう。だんだん複雑になってきました。

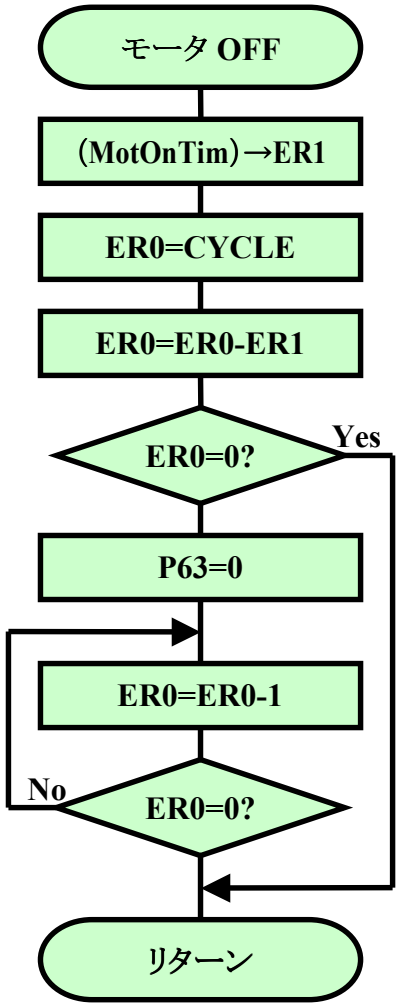



回転数+1 ステップ
(Motor)→R0L
R0L=3?
Yes
R0L=R0L+1
R0L=R0L and 03h
R0L=R0L and 03h
R0L→(Motor)
DC モータ
回転数変更
リターン


回転数-1 ステップ
(Motor)→R0L
R0L=0?
Yes
R0L=R0L-1
R0L=R0L and 03h
R0L=R0L and 03h
R0L→(Motor)
DC モータ
回転数変更
リターン


DC モータ
回転数変更
ER0=00000000h
(Motor)→R0L
ER0=ER0×4
(SHLL.L ER0 ×2)
ER1=MotTable
ER1=ER0+ER1
(ER1)=ER0
ER0→(MotOnTim)
リターン


モータ ON
(MotOnTim)→ER0
ER0=0?
Yes
P63=1
ER0=ER0-1
No
ER0=0?
リターン


モータ OFF
(MotOnTim)→ER1
ER0=CYCLE
ER0=ER0-ER1
ER0=0?
Yes
P63=0
ER0=ER0-1
No
ER0=0?
リターン

では，コーディングです。ビルドが終わったらダウンロードして実行してみてください。スイッチを押すたびにモータの回転数が変化するでしょうか。

```
;---------------------------------------------------------------------------
;                                                                           |
;   FILE        :IoPort_motor.src                                           |
;   DATE        :Wed, Jan 05, 2005                                          |
;   DESCRIPTION :Main Program                                               |
;   CPU TYPE    :H8/3687                                                    |
;                                                                           |
;   This file is programed by TOYO-LINX Co.,Ltd / yKikuchi                  |
;                                                                           |
;---------------------------------------------------------------------------

        .export     _main

;***************************************************************************
;   定数定義
;***************************************************************************
PDR6    .equ    h'FFD9  ;ﾎﾟｰﾄﾃﾞｰﾀﾚｼﾞｽﾀ 6
PCR6    .equ    h'FFE9  ;ﾎﾟｰﾄｺﾝﾄﾛｰﾙﾚｼﾞｽﾀ 6

CYCLE   .equ    66666   ;PWMの周期(20ms)

;***************************************************************************
;   メイン
;***************************************************************************
        .section P,CODE,LOCATE=H'ea00
_main:
;----- イニシャライズ ----------------------------------------------------
        mov.b   #B'00000000,r0l ;ポートのイニシャライズ
        mov.b   r0l,@PDR6       ; あらかじめモータをオフにする
        mov.b   #B'00001000,r0l
        mov.b   r0l,@PCR6       ; bit0-2,4-7 入力 : bit3 出力

        mov.l   #0,er0
        mov.b   r0l,@UpSw       ;スイッチの状態クリア
        mov.b   r0l,@DownSw     ;スイッチの状態クリア
        mov.b   r0l,@Motor      ;モータの状態クリア
        mov.l   er0,@MotOnTim   ;モータオン時間クリア

;----- メインループ ------------------------------------------------------
MainLoop:

;..... アップスイッチの入力 .....................................
upswin:
        btst    #4,@PDR6        ;アップスイッチリード(P64)
        beq     upswin_01       ; オン
        mov.b   #0,r0l          ; オフ
        mov.b   r0l,@UpSw
        bra     upswin_03
```

```
upswin_01:
        mov.b   @UpSw,r0l           ;アップスイッチの状態
        cmp.b   #2,r0l              ;すでにオンになっている
        beq     upswin_03
        cmp.b   #1,r0l              ;ダブルリード, オフからオンになった
        beq     upswin_02
        mov.b   #1,r0l              ;オン検知, もう一度読む
        mov.b   r0l,@UpSw
        bra     upswin_03
upswin_02:
        mov.b   #2,r0l              ;オン
        mov.b   r0l,@UpSw
        bsr     motor_plus          ;回転数 +1ステップ
upswin_03:

;..... ダウンスイッチの入力 ........................................
downswin:
        btst    #5,@PDR6            ;ダウンスイッチリード(P65)
        beq     downswin_01         ; オン
        mov.b   #0,r0l              ; オフ
        mov.b   r0l,@DownSw
        bra     downswin_03
downswin_01:
        mov.b   @DownSw,r0l         ;ダウンスイッチの状態
        cmp.b   #2,r0l              ;すでにオンになっている
        beq     downswin_03
        cmp.b   #1,r0l              ;ダブルリード, オフからオンになった
        beq     downswin_02
        mov.b   #1,r0l              ;オン検知, もう一度読む
        mov.b   r0l,@DownSw
        bra     downswin_03
downswin_02:
        mov.b   #2,r0l              ;オン
        mov.b   r0l,@DownSw
        bsr     motor_minus         ;回転数 -1ステップ
downswin_03:

;..... PWM ..........................................................
        bsr     motor_on            ;モータオン
        bsr     motor_off           ;モータオフ

;..... メインループの先頭にジャンプ ................................
        bra     MainLoop

;*****************************************************************************
;    回転数 +1ステップ
;*****************************************************************************
motor_plus:
        mov.b   @Motor,r0l
        cmp.b   #3,r0l              ;高速?
        beq     motor_plus_01       ; Yes
        inc.b   r0l
        and.b   #H'03,r0l
```

```
        mov.b   r0l,@Motor
        bsr     motor_change
motor_plus_01:
        rts

;*********************************************************************
;   回転数 -1ステップ
;*********************************************************************
motor_minus:
        mov.b   @Motor,r0l          ;停止?
        beq     motor_minus_01      ; Yes
        dec.b   r0l
        and.b   #H'03,r0l
        mov.b   r0l,@Motor
        bsr     motor_change
motor_minus_01:
        rts

;*********************************************************************
;   DCモータの回転数変更
;*********************************************************************
motor_change:
        mov.l   #H'00000000,er0
        mov.b   @Motor,r0l          ;モータの状態変更
        shll.l  er0
        shll.l  er0
        mov.l   #MotTable,er1       ;モータオン時間をテーブルから取得する
        add.l   er0,er1
        mov.l   @er1,er0
        mov.l   er0,@MotOnTim
        rts

MotTable:
        .data.l 0                   ;Motor=0のときのモータオン時間(停止)
        .data.l 33333               ;Motor=1のときのモータオン時間(50%)
        .data.l 50000               ;Motor=2のときのモータオン時間(75%)
        .data.l 66666               ;Motor=3のときのモータオン時間(100%)

;*********************************************************************
;   モータオン
;*********************************************************************
motor_on:
        mov.l   @MotOnTim,er0
        beq     motor_on_02         ;モータオン時間=0→ジャンプ
        bset    #3,@PDR6            ;モータオン
motor_on_01:
        dec.l   #1,er0              ;ウェイト
        bne     motor_on_01
motor_on_02:
        rts

;*********************************************************************
;   モータオフ
```

```
;*********************************************************************
motor_off:
        mov.l   @MotOnTim,er1
        mov.l   #CYCLE,er0
        sub.l   er1,er0             ;er0=er0-er1
        beq     motor_off_02        ;モータオフ時間=0→ジャンプ
        bclr    #3,@PDR6            ;モータオフ
motor_off_01:
        dec.l   #1,er0              ;ウェイト
        bne     motor_off_01
motor_off_02:
        rts

;*********************************************************************
;    ワークエリア
;*********************************************************************
        .section B,DATA,LOCATE=H'f780
UpSw        .res.b  1               ;アップスイッチ(P64)の状態
                                    ; 0-オフ
                                    ; 1-オン検知, もう一度読む
                                    ; 2-オン, ダブルリード完了
DownSw      .res.b  1               ;ダウンスイッチ(P65)の状態
                                    ; 0-オフ
                                    ; 1-オン検知, もう一度読む
                                    ; 2-オン, ダブルリード完了
Motor       .res.b  1               ;モータの状態
                                    ; 0-停止
                                    ; 1-低速(50%)
                                    ; 2-中速(75%)
                                    ; 3-高速(100%)
            .align  2
MotOnTim    .res.l  1               ;モータオン時間

;---------------------------------------------------------------------
        .end
```

◆

モータによっては停止状態から低速(50%)にしたとき，きちんと回らないかもしれません。回り始めるときには大きな力が必要なためです。どうすれば，いつもきちんと回すことができるか考えてみましょう。少しずつ改良していくのもプログラムの楽しさの一つですよ。(ヒント:一瞬 100%で回してみるとか…)

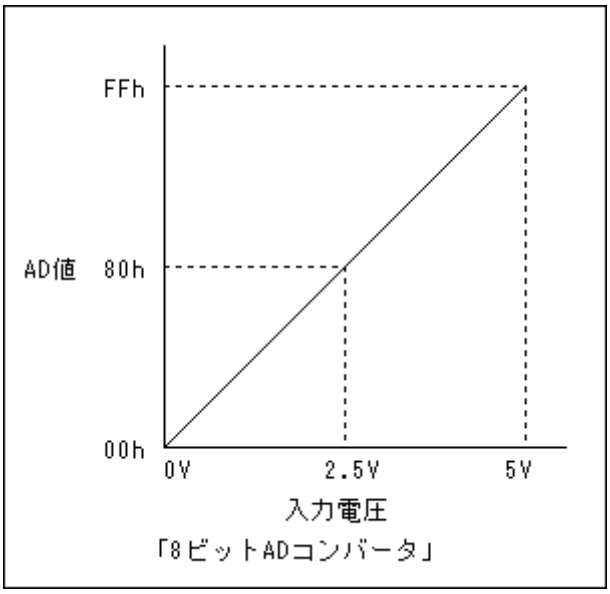
第10章 ADコンバータの使い方入門

1. ADコンバータとは
2. H8/3687のADコンバータ
3. 明るさを数字で表してみよう

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。

1. 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。

2. 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。たいていは, 変換結果をプログラムで 6 ビット右シフトして 0000h～03FFh として扱うことが多いです。もちろん, どうするかはプログラマしだいですが…。

■ 入力チャンネル:8 チャンネル

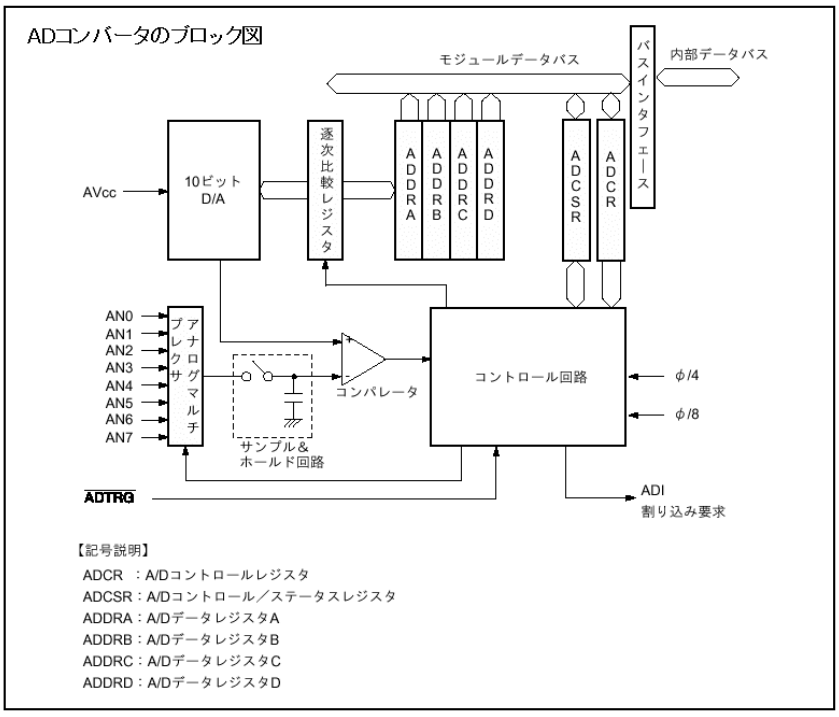
AD コンバータ自体は 1 個だけなのですが, アナログマルチプレクサ回路が内蔵されているので, 8 種類の電圧を入力することができます。そのため, 同時に 8 チャンネルの AD 変換ができるわけでなく, 順番に 1 チャンネルずつ AD 変換します。

■ 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。

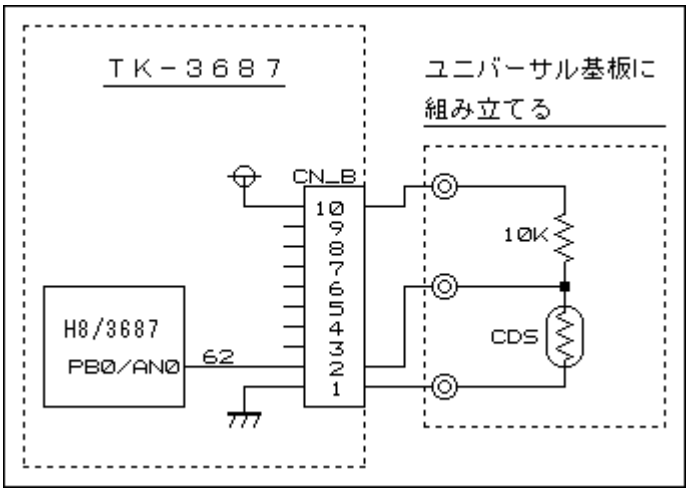
■ 変換速度

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 μs で変換できます。

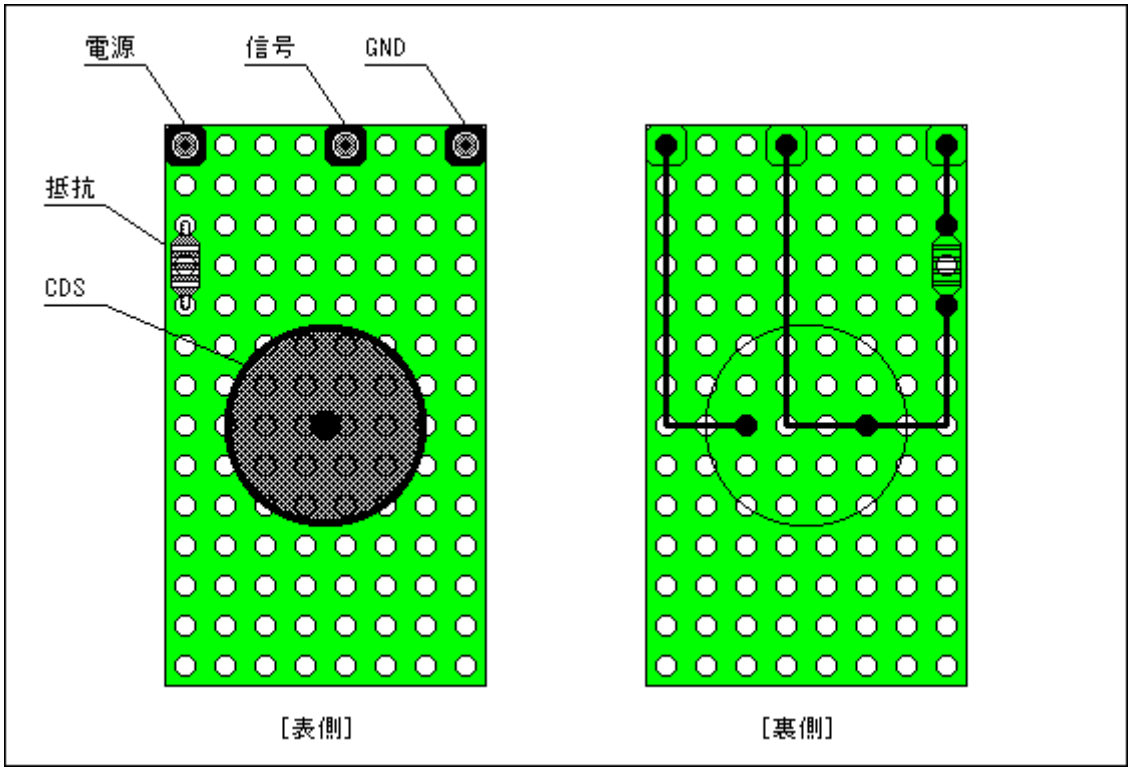


3. 明るさを数字で表してみよう

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以上になるものもあります。右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。下の実装図を参考にして組み立ててください。



注意：ここで使う部品はキットには含まれていません。パーツショップ等でご購入ください。

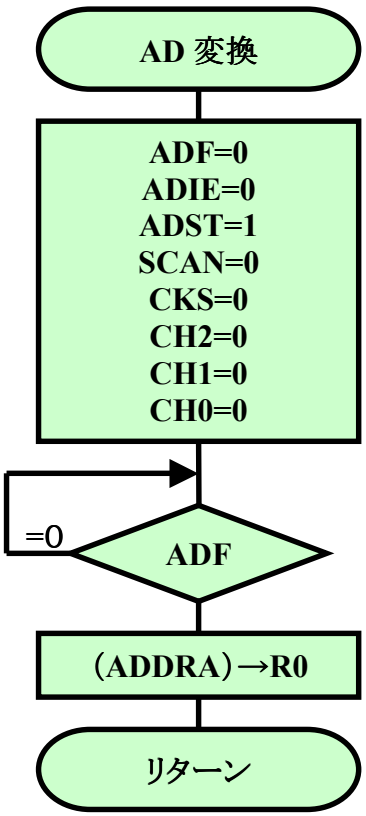


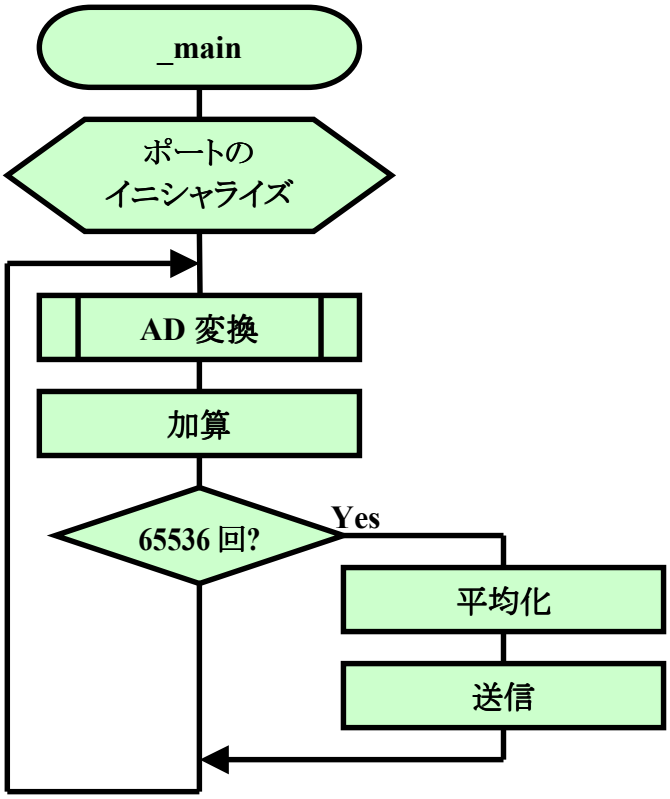
次に，CD-ROM から

‘Adconv. mot’

をダウンロードして実行して下さい。ハイパーターミナルの画面に AD 値が表示されます。だいたい 0.5 秒ごとに表示されます。Cds にあたる光の強さをかえると AD 値もかわるはずです。

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 値をそのまま表示すればよいかというと, 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 65536 回加算して平均しました。得られた平均値をシリアルポートから送信します。左図のようなフローチャートになります。ソースリストは次のページをご覧ください。

```
;---------------------------------------------------------------------------
;                                                                           |
;   FILE        :Adconv.src                                                 |
;   DATE        :Wed, Feb 23, 2005                                          |
;   DESCRIPTION :Main Program                                               |
;   CPU TYPE    :H8/3687                                                    |
;                                                                           |
;   This file is programed by TOYO-LINX Co.,Ltd / yKikuchi                  |
;                                                                           |
;---------------------------------------------------------------------------

        .export     _main

;***************************************************************************
;    定数定義
;***************************************************************************
ADDRA   .equ    h'FFB0  ;A/Dﾃﾞｰﾀﾚｼﾞｽﾀ A
ADDRB   .equ    h'FFB2  ;A/Dﾃﾞｰﾀﾚｼﾞｽﾀ B
ADDRC   .equ    h'FFB4  ;A/Dﾃﾞｰﾀﾚｼﾞｽﾀ C
ADDRD   .equ    h'FFB6  ;A/Dﾃﾞｰﾀﾚｼﾞｽﾀ D
ADCSR   .equ    h'FFB8  ;A/Dｺﾝﾄﾛｰﾙ/ｽﾃｰﾀｽﾚｼﾞｽﾀ
ADCR    .equ    h'FFB9  ;A/Dｺﾝﾄﾛｰﾙﾚｼﾞｽﾀ

SMR     .equ    h'FFA8  ;ｼﾘｱﾙﾓｰﾄﾞﾚｼﾞｽﾀ
BRR     .equ    h'FFA9  ;ﾋﾞｯﾄﾚｰﾄﾚｼﾞｽﾀ
SCR3    .equ    h'FFAA  ;ｼﾘｱﾙｺﾝﾄﾛｰﾙﾚｼﾞｽﾀ 3
TDR     .equ    h'FFAB  ;ﾄﾗﾝｽﾐｯﾄﾃﾞｰﾀﾚｼﾞｽﾀ
SSR     .equ    h'FFAC  ;ｼﾘｱﾙｽﾃｰﾀｽﾚｼﾞｽﾀ
RDR     .equ    h'FFAD  ;ﾚｼｰﾌﾞﾃﾞｰﾀﾚｼﾞｽﾀ

PDRB    .equ    h'FFDD  ;ﾎﾟｰﾄﾃﾞｰﾀﾚｼﾞｽﾀ B
PMR1    .equ    h'FFE0  ;ﾎﾟｰﾄﾓｰﾄﾞﾚｼﾞｽﾀ 1

;***************************************************************************
;    メイン
;***************************************************************************
        .section P,CODE,LOCATE=H'ea00
_main:
;----- イニシャライズ ----------------------------------------------------
        bsr     init_ad             ;ADｺﾝﾊﾞｰﾀ ｲﾆｼｬﾗｲｽﾞ
        mov.l   #0,er0              ;ADｺﾝﾊﾞｰﾀ用ﾜｰｸｴﾘｱ ｸﾘｱ
        mov.l   er0,@ADBuffer
        mov.w   r0,@ADCounter
        mov.w   r0,@ADData

        bsr     init_sci3           ;SCI3 ｲﾆｼｬﾗｲｽﾞ
        mov.b   #H'0d,r0l           ;改行
        bsr     txone:16

;----- メインループ ------------------------------------------------------
```

```
_main_01:
        bsr     adc             ;AD値入力
        mov.w   #0,e0           ;加算
        mov.l   @ADBuffer,er1
        add.l   er0,er1
        mov.l   er1,@ADBuffer
        mov.w   @ADCounter,r0   ;加算ｶｳﾝﾀｰ1
        dec.w   #1,r0
        mov.w   r0,@ADCounter
        beq     _main_02        ;65536回加算なら平均,送信へｼﾞｬﾝﾌﾟ
        bra     _main_01        ;ﾒｲﾝﾙｰﾌﾟ先頭にｼﾞｬﾝﾌﾟ
_main_02:
        mov.l   @ADBuffer,er0   ;平均(1/65536),E0を取り出す
        shlr.w  e0              ;6bit右ｼﾌﾄして左詰を右詰に変更
        shlr.w  e0
        shlr.w  e0
        shlr.w  e0
        shlr.w  e0
        shlr.w  e0
        mov.w   e0,@ADData      ;AD値をｽﾄｱ

        mov.w   e0,r0
        mov.b   r0h,r0l
        bsr     hex2asc         ;ｱｽｷｰｺｰﾄﾞ変換,R0L→R1
        mov.b   r1l,r0l         ;送信
        bsr     txone
        mov.w   e0,r0
        bsr     hex2asc         ;ｱｽｷｰｺｰﾄﾞ変換,R0L→R1
        mov.b   r1h,r0l         ;送信
        bsr     txone
        mov.b   r1l,r0l         ;送信
        bsr     txone
        mov.b   #H'0d,r0l       ;改行
        bsr     txone

        mov.l   #0,er0          ;ADｺﾝﾊﾞｰﾀ用ﾜｰｸｴﾘｱ ｸﾘｱ
        mov.l   er0,@ADBuffer

        bra     _main_01        ;ﾒｲﾝﾙｰﾌﾟ先頭にｼﾞｬﾝﾌﾟ

;*********************************************************************************
;    ADコンバータ イニシャライズ
;*********************************************************************************
init_ad:
        mov.b   #B'00000000,r0l ;単一ﾓｰﾄﾞ,134ｽﾃｰﾄ,AN0
        mov.b   r0l,@ADCSR
        rts

;*********************************************************************************
;    AD変換
;       R0 = AD値
;*********************************************************************************
adc:
```

```
        mov.b   #B'00100000,r0l     ;AD変換ｽﾀｰﾄ
        mov.b   r0l,@ADCSR
adc_01:
        btst    #7,@ADCSR           ;AD変換終了?
        beq     adc_01              ;No

        mov.w   @ADDRA,r0           ;AD値ｹﾞｯﾄ

        rts

;*****************************************************************************
;    SCI3 イニシャライズ
;*****************************************************************************
MHz         .equ D'20           ;X=20MHz
BAUD        .equ 38400          ;9600,19200,38400
BITR        .equ (MHz*D'1000000)/(BAUD*D'32)-1
WAIT_1B     .equ (MHz*D'1000000)/D'6/BAUD

init_sci3:
        bset    #1,@PMR1            ;P22はTXDとして使う
        xor.b   r0l,r0l
        mov.b   r0l,@SCR3           ;TE,REを0にｸﾘｱ/CKE1=CKE0=0,内部ﾎﾞｰﾚｰﾄ
        mov.b   r0l,@SMR            ;調歩同期,8ﾋﾞｯﾄ長,ｽﾄｯﾌﾟﾋﾞｯﾄ=1,ﾊﾟﾘﾃｨなし
        mov.b   #BITR,r0l
        mov.b   r0l,@BRR            ;通信ﾚｰﾄに対応する値をﾗｲﾄ
        mov.w   #WAIT_1B,r0         ;1ﾋﾞｯﾄ期間待つ
init_sci3_00:
        dec.w   #1,r0
        bne     init_sci3_00
        mov.b   #H'30,r0l           ;送受信ｲﾈｰﾌﾞﾙ,割込みﾃﾞｨｾｰﾌﾞﾙ
        mov.b   r0l,@SCR3
        rts

;*****************************************************************************
;    1バイト受信
;        R0L = 受信データ
;*****************************************************************************
rxone:
        btst    #6,@SSR             ;受信ﾃﾞｰﾀあり?
        beq     rxone               ; No→ｼﾞｬﾝﾌﾟ
        mov.b   @RDR,r0l            ;受信ﾃﾞｰﾀをﾘｰﾄﾞ
        rts

;*****************************************************************************
;    1バイト送信
;        R0L = 送信データ
;*****************************************************************************
txone:
        btst    #7,@SSR             ;送信可能?
        beq     txone               ; No→ｼﾞｬﾝﾌﾟ
        mov.b   r0l,@TDR            ;送信ﾃﾞｰﾀをﾗｲﾄ
        rts
```

```
;*************************************************************************
;    アスキーコード変換
;        R0L = HEX → R1 = ASCII
;*************************************************************************
hex2asc:
        mov.b    r0l,r1h
        shlr.b   r1h
        shlr.b   r1h
        shlr.b   r1h
        shlr.b   r1h
        add.b    #H'30,r1h
        cmp.b    #H'3a,r1h
        blo      hex2asc_01
        add.b    #H'07,r1h
hex2asc_01:
        mov.b    r0l,r1l
        and.b    #H'0f,r1l
        add.b    #H'30,r1l
        cmp.b    #H'3a,r1l
        blo      hex2asc_02
        add.b    #H'07,r1l
hex2asc_02:
        rts

;*************************************************************************
;    ワークエリア
;*************************************************************************
        .section B,DATA,LOCATE=H'f780
ADBuffer      .res.l   1             ;AD値加算ﾊﾞｯﾌｧ
ADCounter     .res.w   1             ;AD値加算ｶｳﾝﾀ
ADData        .res.w   1             ;平均AD値

;-------------------------------------------------------------------------
        .end
```

第11章 C言語入門

1. メモリマップの確認
2. プロジェクトの作成
3. プログラムの入力
4. ビルド!!
5. ダウンロードと実行

これまではアセンブラでプログラムを書いてきました。しかし，マイコンのプログラムもこれからはC 言語が主流になります(というか，すでに C で書くことが多いのですが…)。というわけで，この章ではHEWを使ったC言語によるプログラミングを体験してみましょう。実は，アセンブラはC言語コンパイラの機能の一部を使っていたのですよ。

1. メモリマップの確認

アセンブラのときにも説明しましたが，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。C 言語とハイパー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 colspan="3">モニタプログラム
‘ハイパーH8’</td><td>ROM/フラッシュメモリ
(56K バイト)</td></tr>
<tr><td>E000 番地
E7FF 番地</td><td colspan="3">未使用</td><td>未使用</td></tr>
<tr><td>E800 番地</td><td colspan="3">ハイパーH8
ユーザ割り込みベクタ</td><td rowspan="3">RAM
(2K バイト)</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 colspan="3">未使用</td><td>未使用</td></tr>
<tr><td>F700 番地
F77F 番地</td><td colspan="3">I/O レジスタ</td><td>I/O レジスタ</td></tr>
<tr><td>F780 番地

FB7F 番地</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><td>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，
FDT と E7 使用時は，
ユーザ使用不可</td></tr>
<tr><td>FB80 番地

FD80 番地
FDFF 番地</td><td>S</td><td>スタック領域</td><td rowspan="3">RAM
(1K バイト)</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 colspan="3">I/O レジスタ</td><td>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。

2. プロジェクトの作成

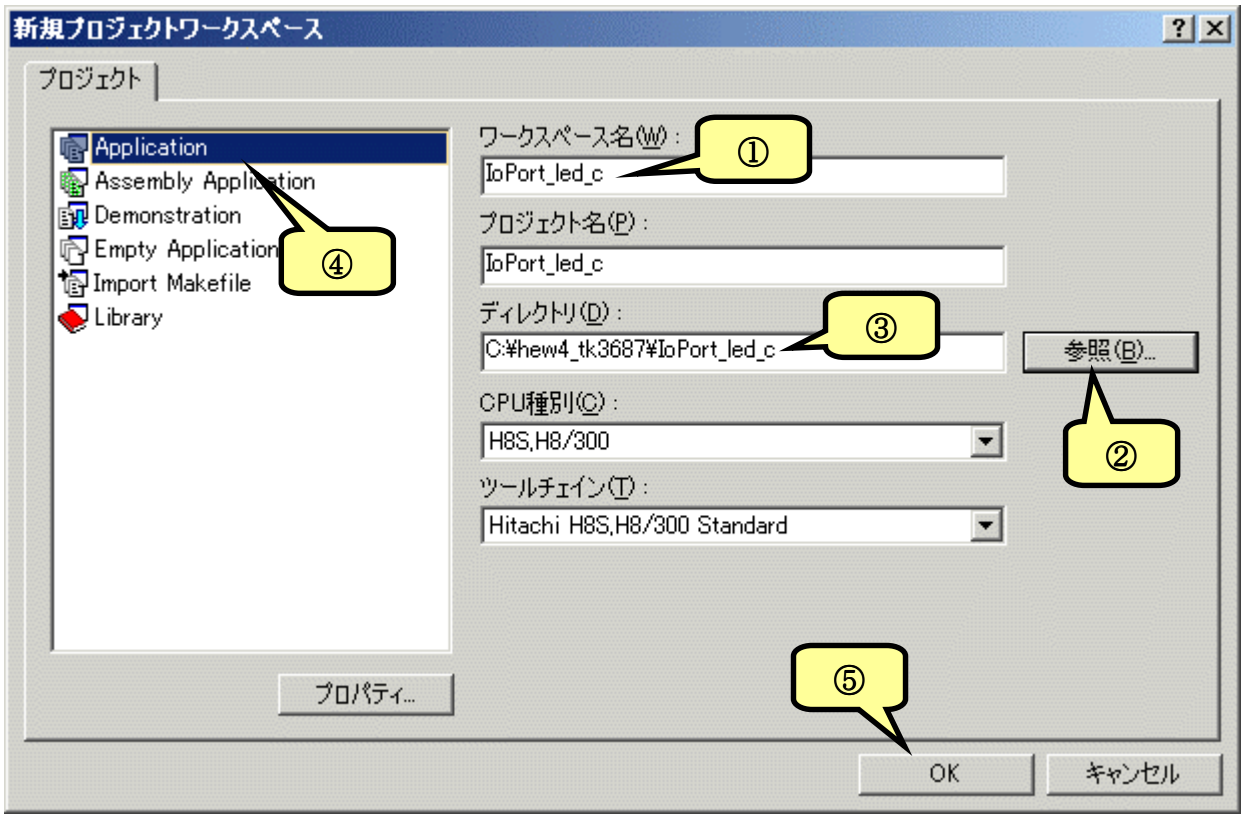
では，HEW を起動しましょう。「新規プロジェクトワークスペースの作成」を選択して「OK」をクリックします（ここまではアセンブラと同じですね）。すると，次のダイアログウィンドウが開きます。

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。

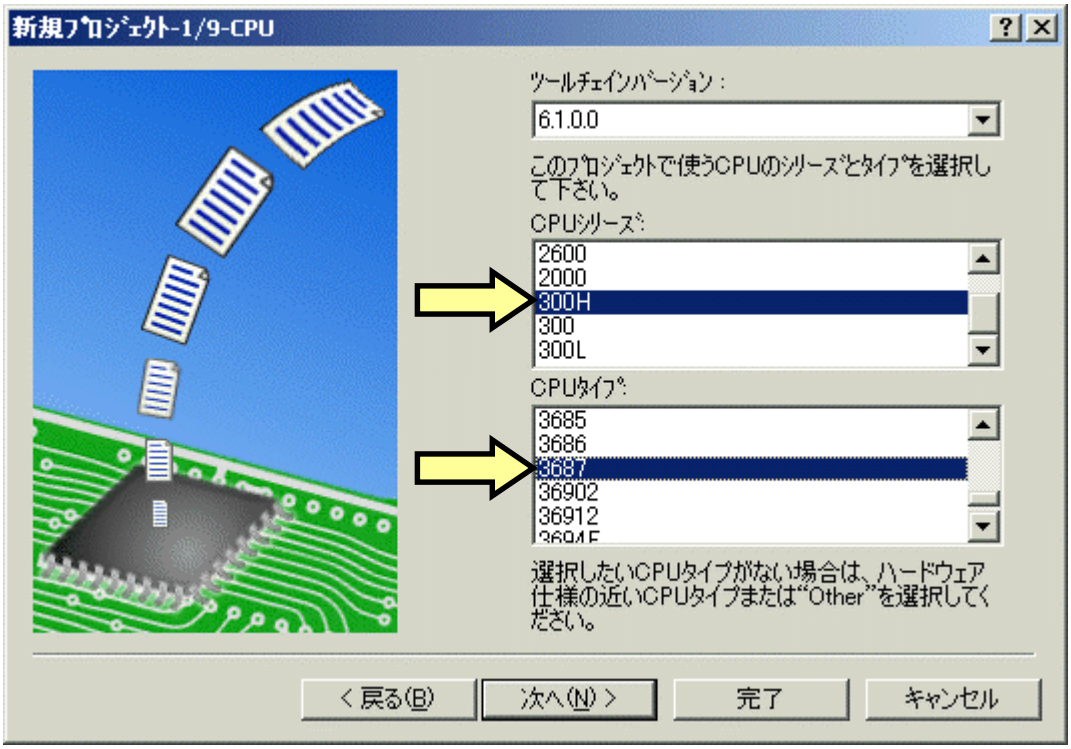
ワークスペースの場所を指定します。②右の「参照（B）…」ボタンをクリックします。そして，あらかじめ用意した HEW 専用作業フォルダ（ここでは Hew4_tk3687）を指定します。設定後，「ディレクトリ（D）」が正しいか確認して下さい。（③）

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。

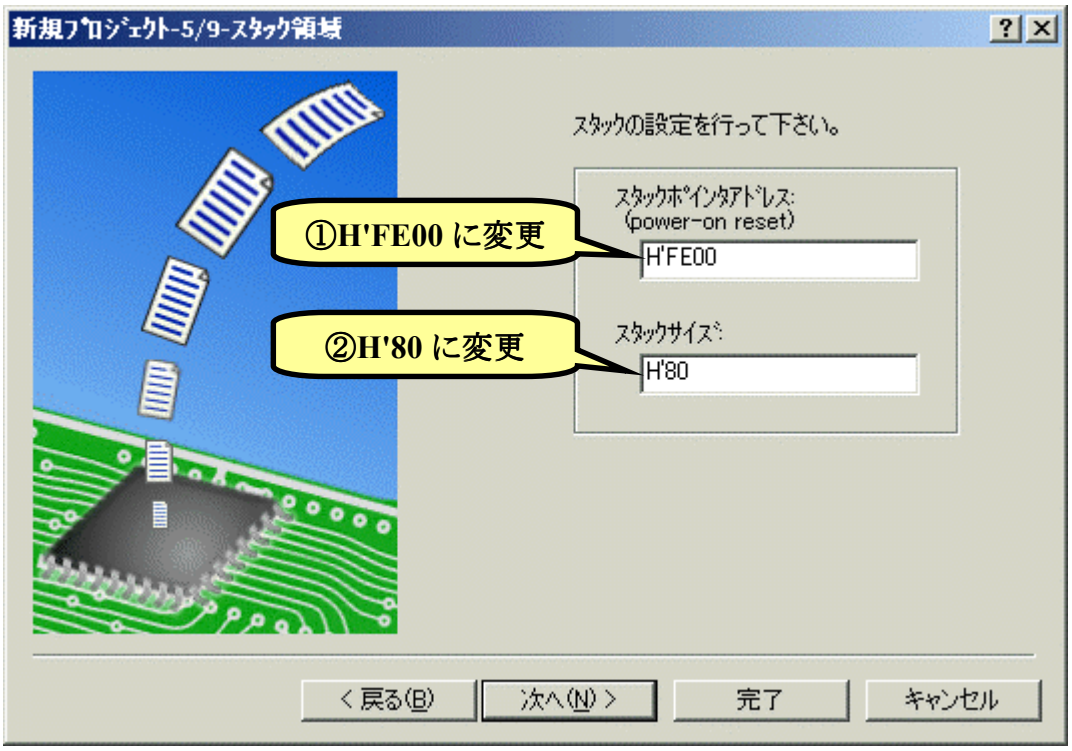


「新規プロジェクトー1／9－CPU」で，使用する CPU シリーズ（300H）と，CPU タイプ（3687）を設定し，「次へ（N）＞」をクリックします。

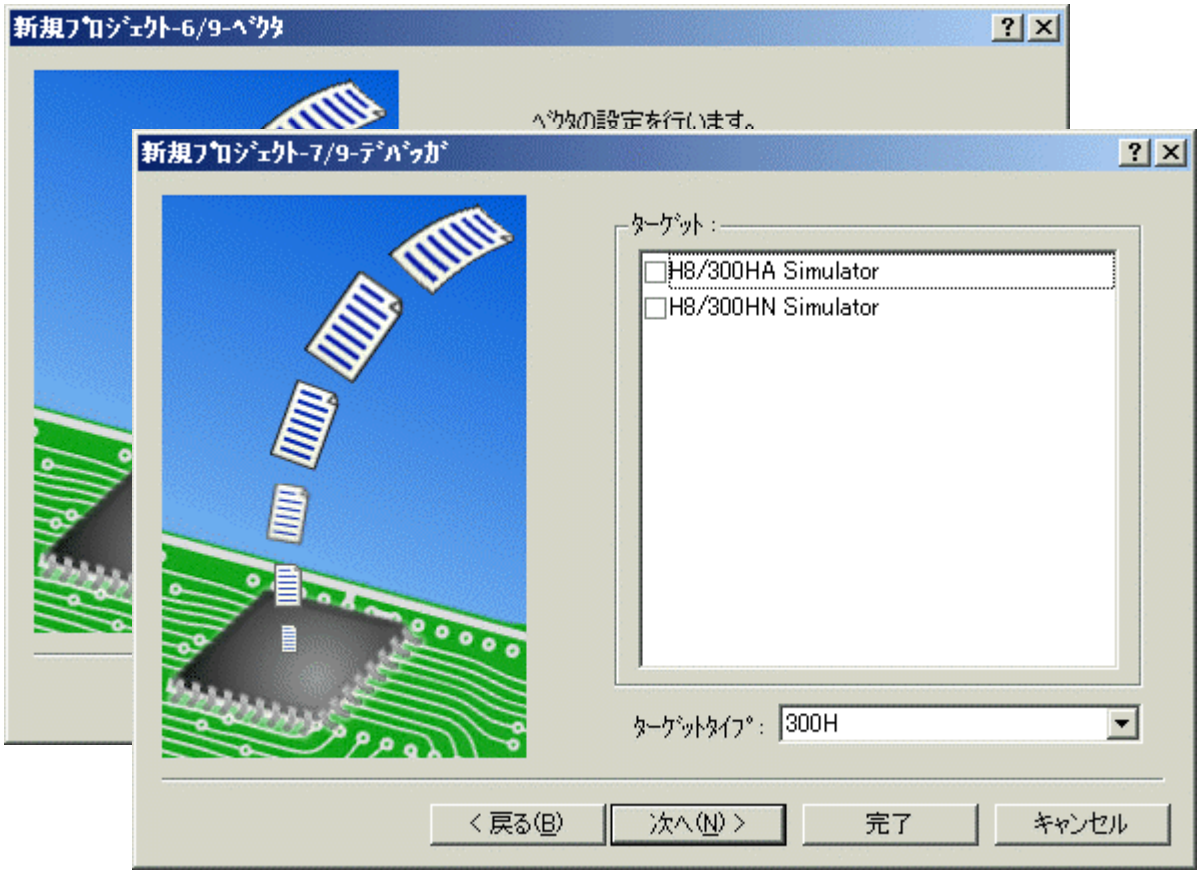


「新規プロジェクトー2／9－オプション」，「新規プロジェクトー3／9－生成ファイル」，「新規プロジェクトー4／9－標準ライブラリ」は変更しません。「次へ（N）＞」をクリックして次の画面に進みます。

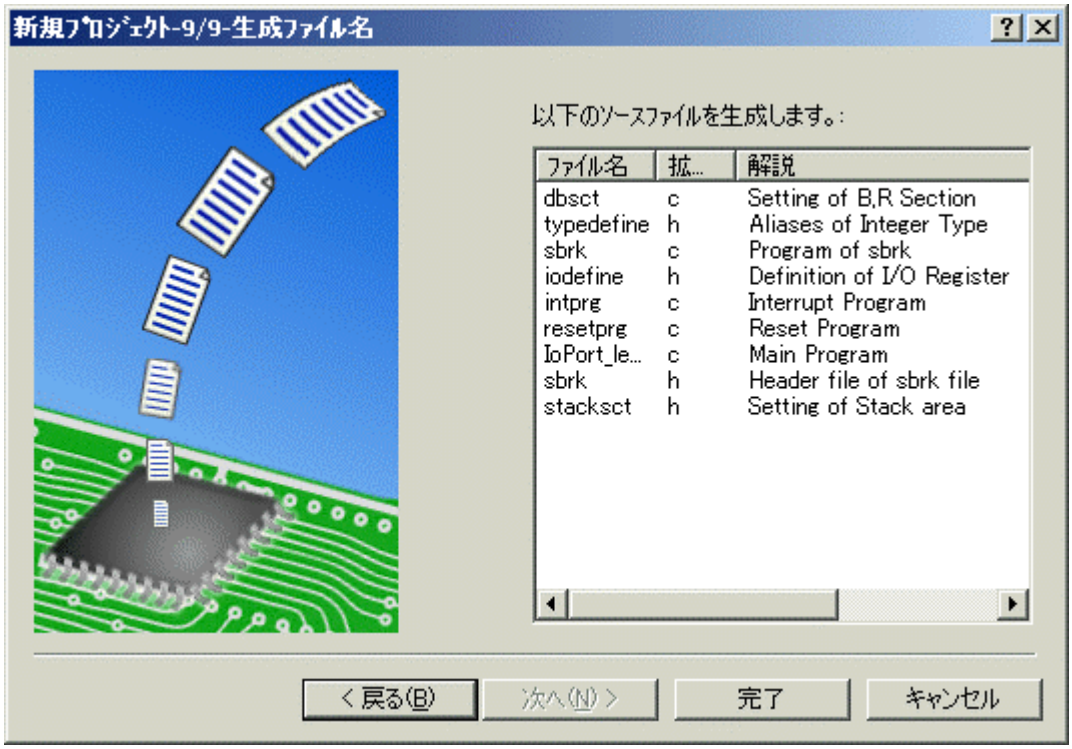
「新規プロジェクトー5／9ースタック領域」でスタックのアドレスとサイズを変更します。ハイパーH8 を使用するので，①スタックポインタを H'FE00 に，②スタックサイズを H'80 にします。設定が終わったら「次へ(N) >」をクリックします。（ハイパーH8 を使用しないときは変更の必要はありません。）



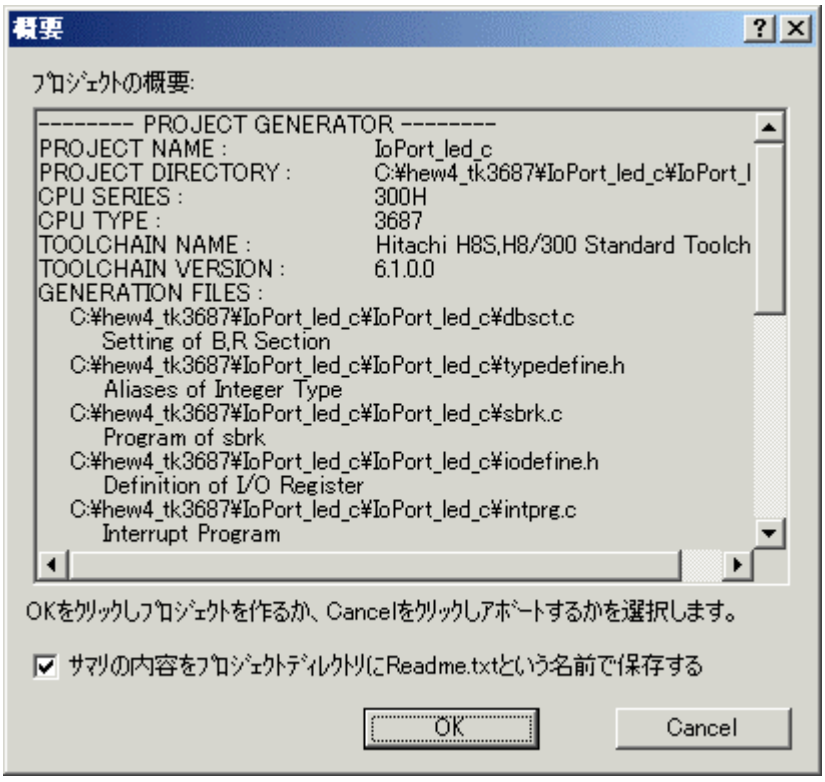
「新規プロジェクトー6／9ーベクタ」，「新規プロジェクトー7／9ーデバッガ」は変更しません。「次へ(N) >」をクリックして順に次の画面に進みます。



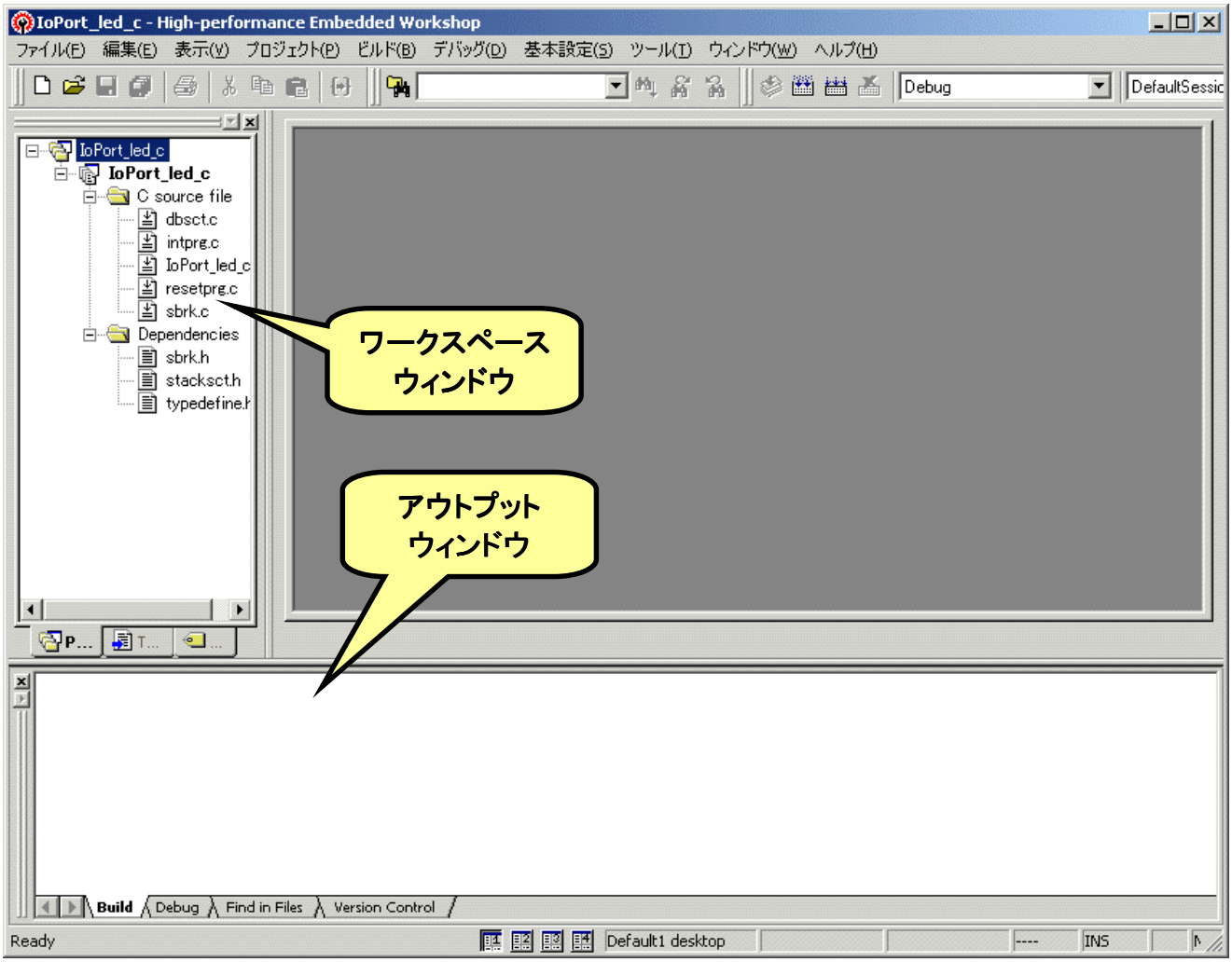
次は「新規プロジェクトー9／9ー生成ファイル名」です。ここも変更しません。「完了」をクリックします。



すると,「概要」が表示されるので「OK」をクリックします。

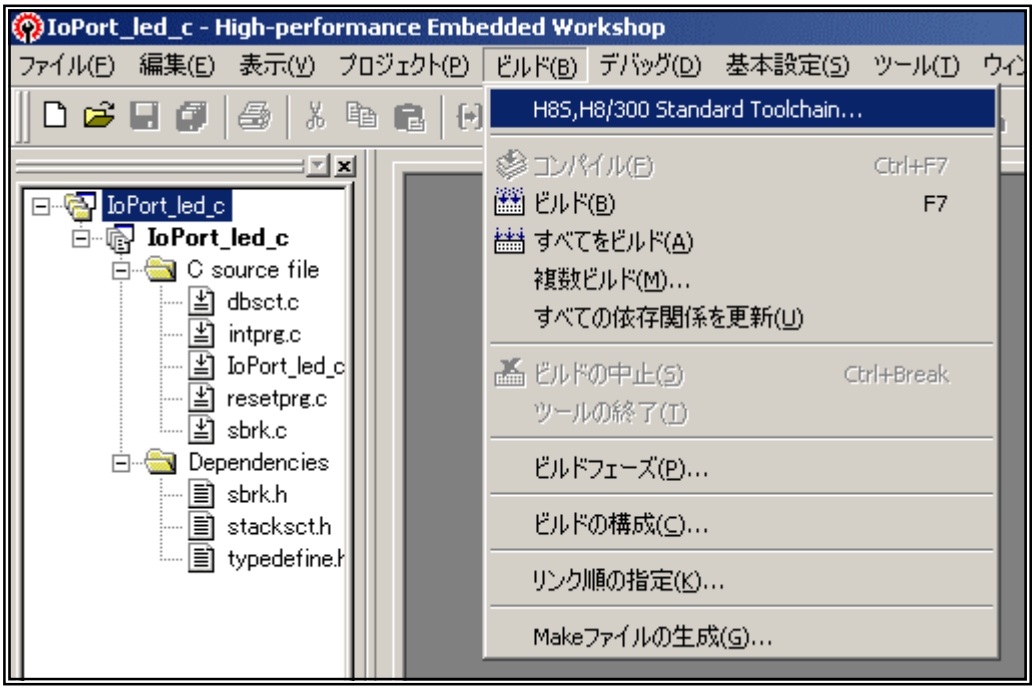


これで，プロジェクトワークスペースが完成します。HEW はプロジェクトに必要なファイルを自動生成し，それらのファイルは左端のワークスペースウィンドウに一覧表示されます。

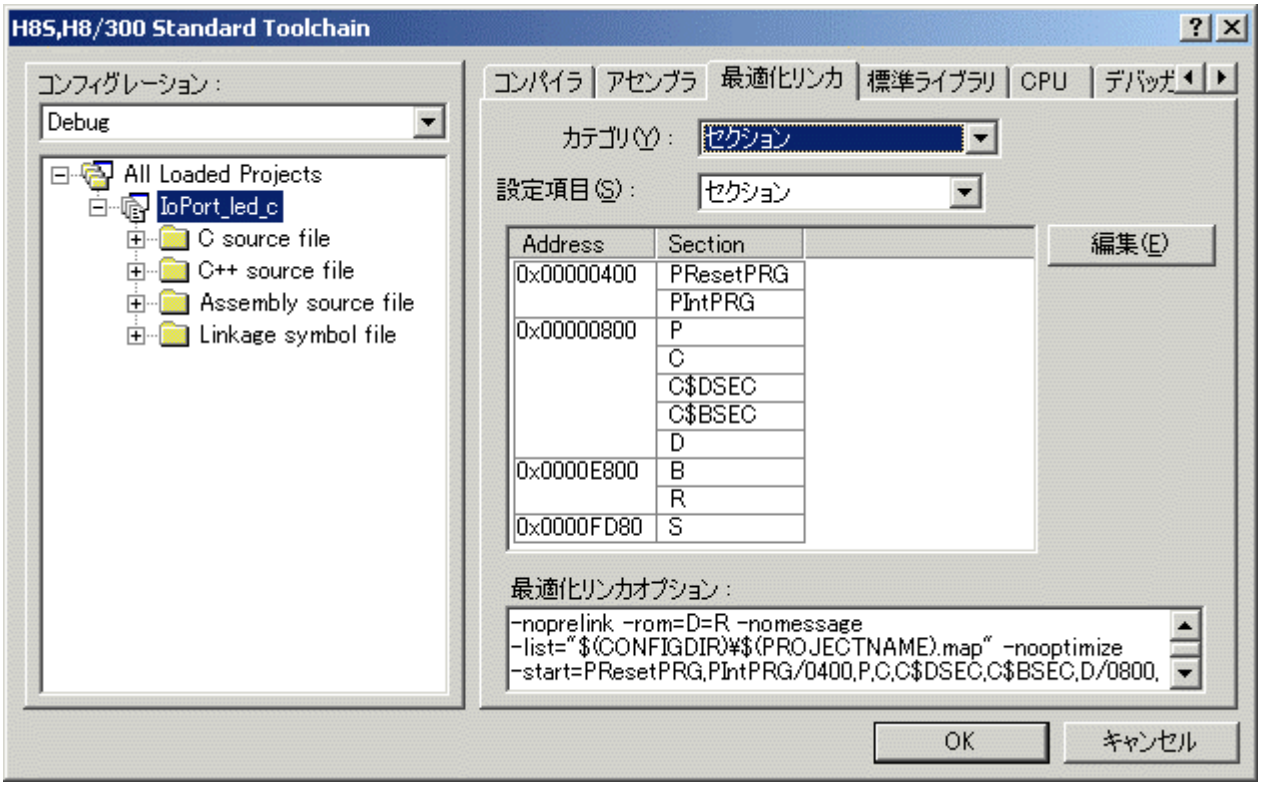


さて，これでプロジェクトは完成したのですが，ハイパーH8 を使うためにセクションを変更してプログラムが RAM 上にできるようにします。（当然ながら，ハイパーH8 を使わないときは変更する必要はなく，そのままで OK。）

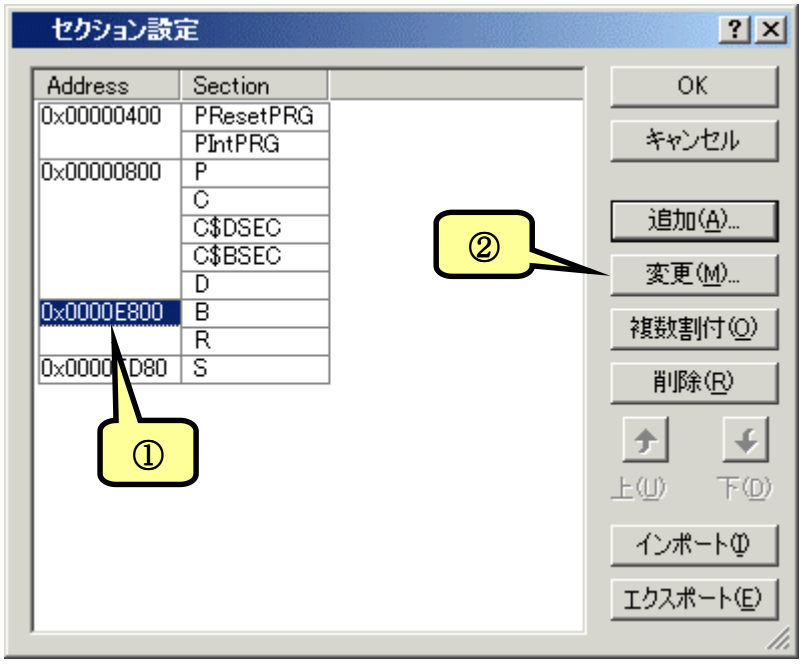
下図のように，メニューバーから「ビルド（B）」＞「H8S,H8/300 Standard Toolchain…」を選びます。



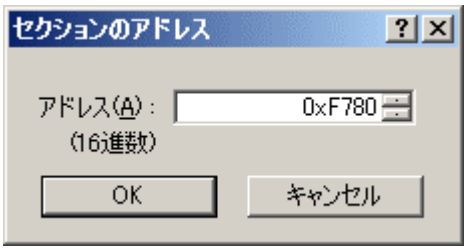
すると，「H8S，H8/300 Standard Toolchain」ウィンドウが開きます。「最適化リンカ」のタブを選び，「カテゴリ（Y）」のドロップダウンメニューの中から「セクション」を選択します。すると，下図のような各セクションの先頭アドレスを設定する画面になります。「編集（E）」ボタンをクリックしてください。



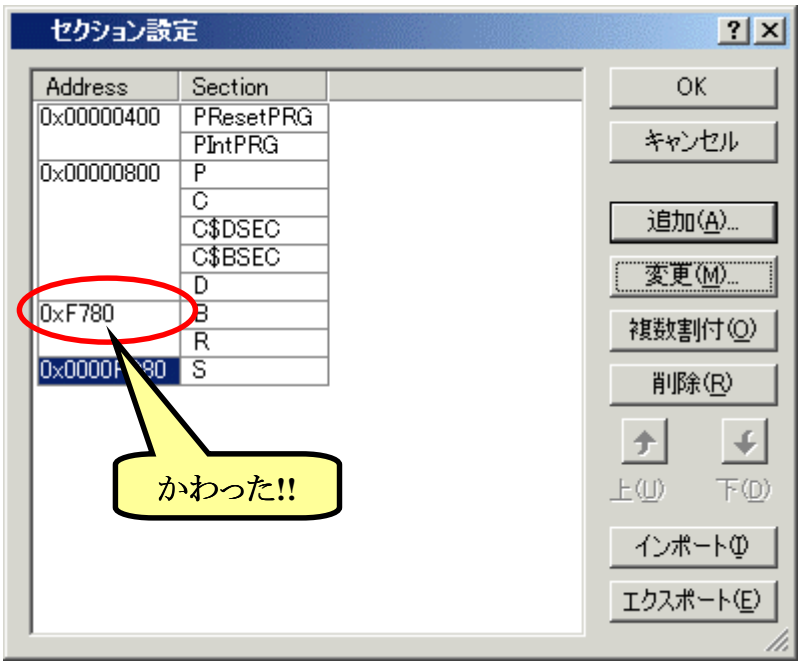
「セクション設定」ダイアログが開きます。それでは，「1. メモリマップの確認」で調べたメモリマップにあわせて設定していきましょう。最初に‘B’Section のアドレスを変更します。デフォルトでは E800 番地になっていますね。①‘0x0000E800’というところをクリックして下さい。それから，②「変更(M)…」をクリックします。



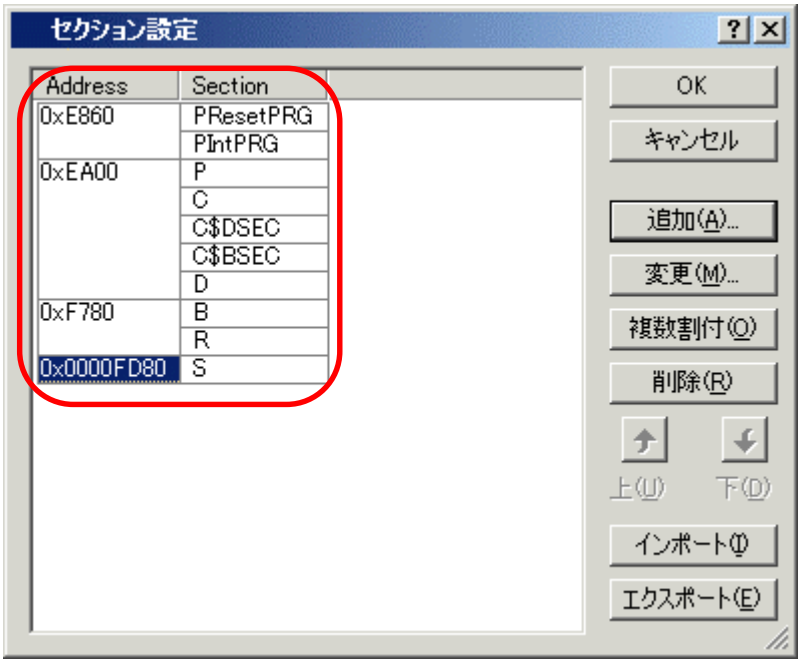
そうすると，「セクションのアドレス」ダイアログが開きます。‘B’Section は F780 番地から始まりますので，右のように入力して‘OK’をクリックします。



すると…

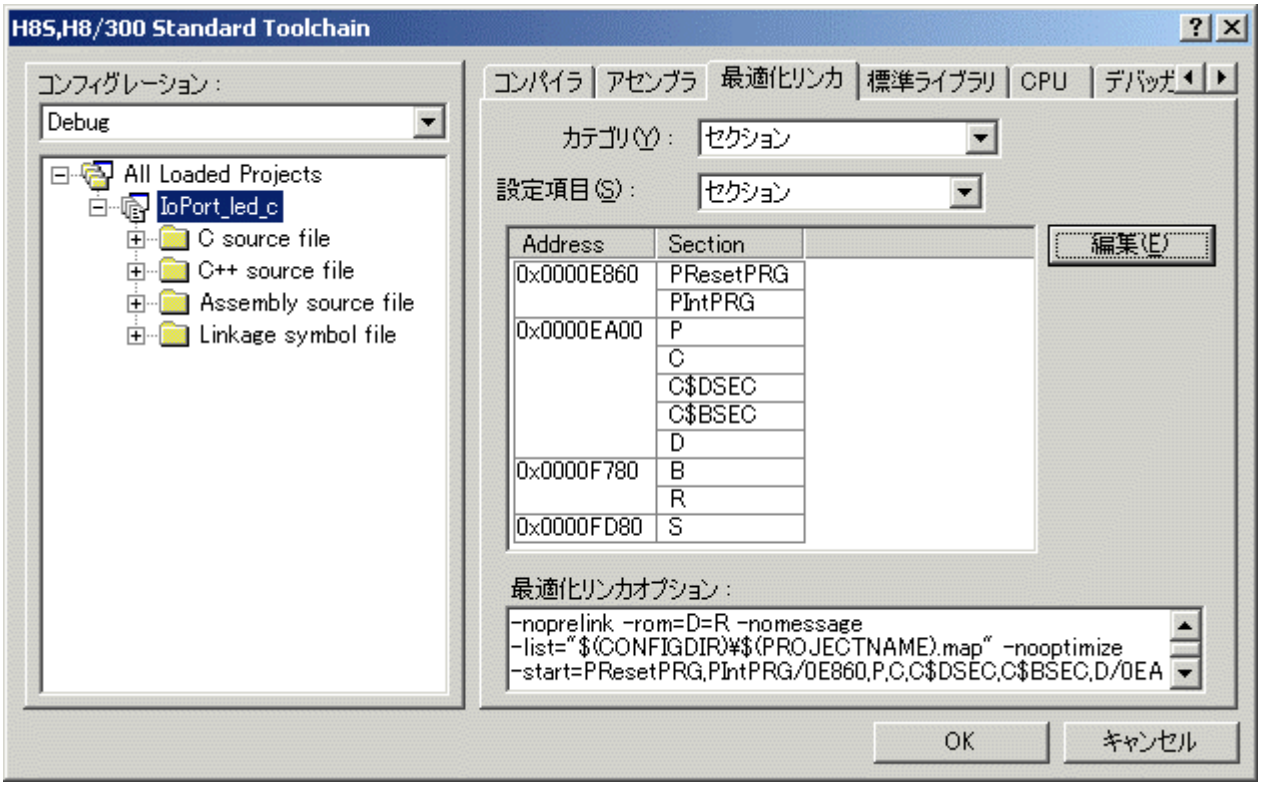


同じように，他のセクションも変更しましょう。メモリマップと同じようにSection が指定されていることを確認します。ちゃんと設定されていたら「OK」をクリックします。



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!!)



もう一度確認してから「OK」をクリックして‘H8S，H8/300 Standard Toolchain’ウィンドウを閉じます。

3. プログラムの入力

第 7 章の「2. LED を光らせよう」を C 言語に置き換えます。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。

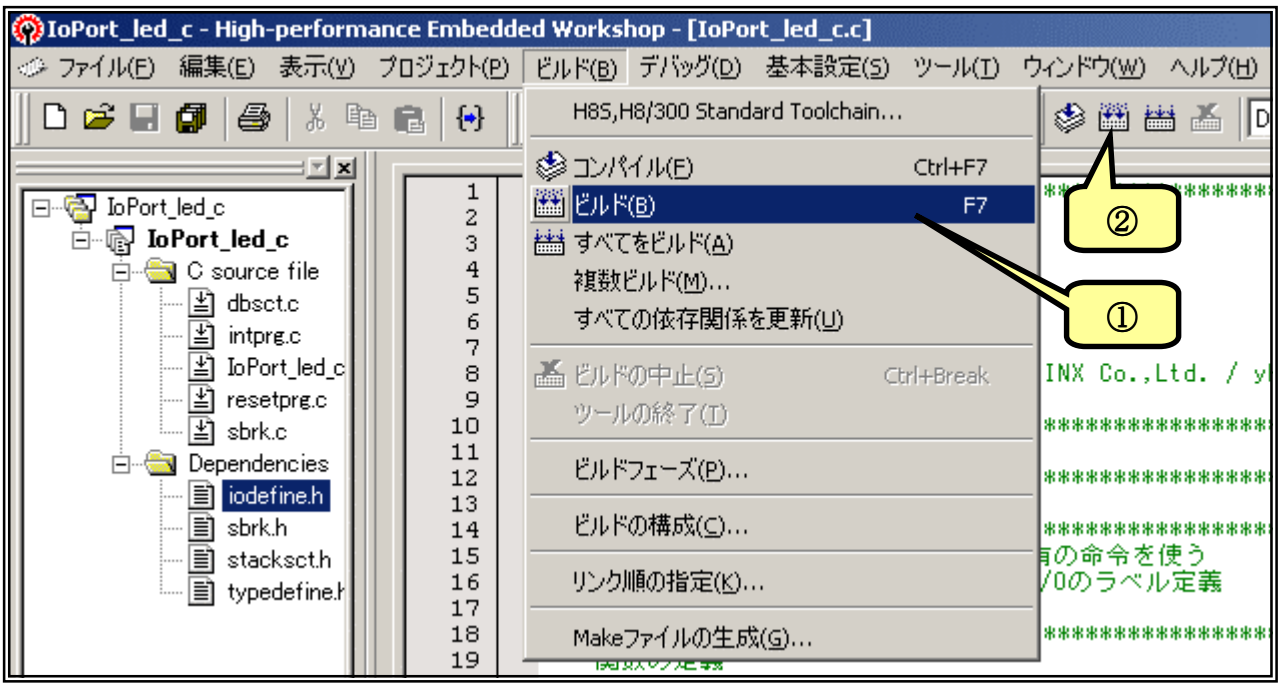
```
/************************************************************************/
/*                                                                      */
/*  FILE        :IoPort_led_c.c                                         */
/*  DATE        :Wed, Apr 20, 2005                                      */
/*  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
```

このファイルに追加・修正していきます。下記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。

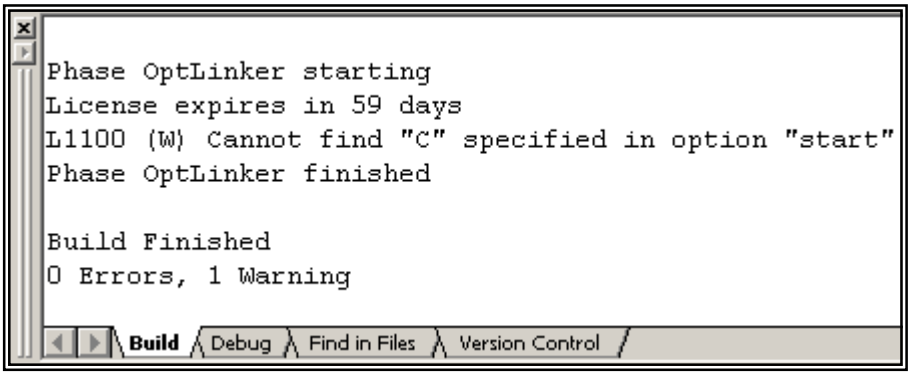
```
/************************************************************************/
/*                                                                      */
/*  FILE        :IoPort_led_c.c                                         */
/*  DATE        :Wed, Apr 20, 2005                                      */
/*  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++){}
}
```

4. 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。

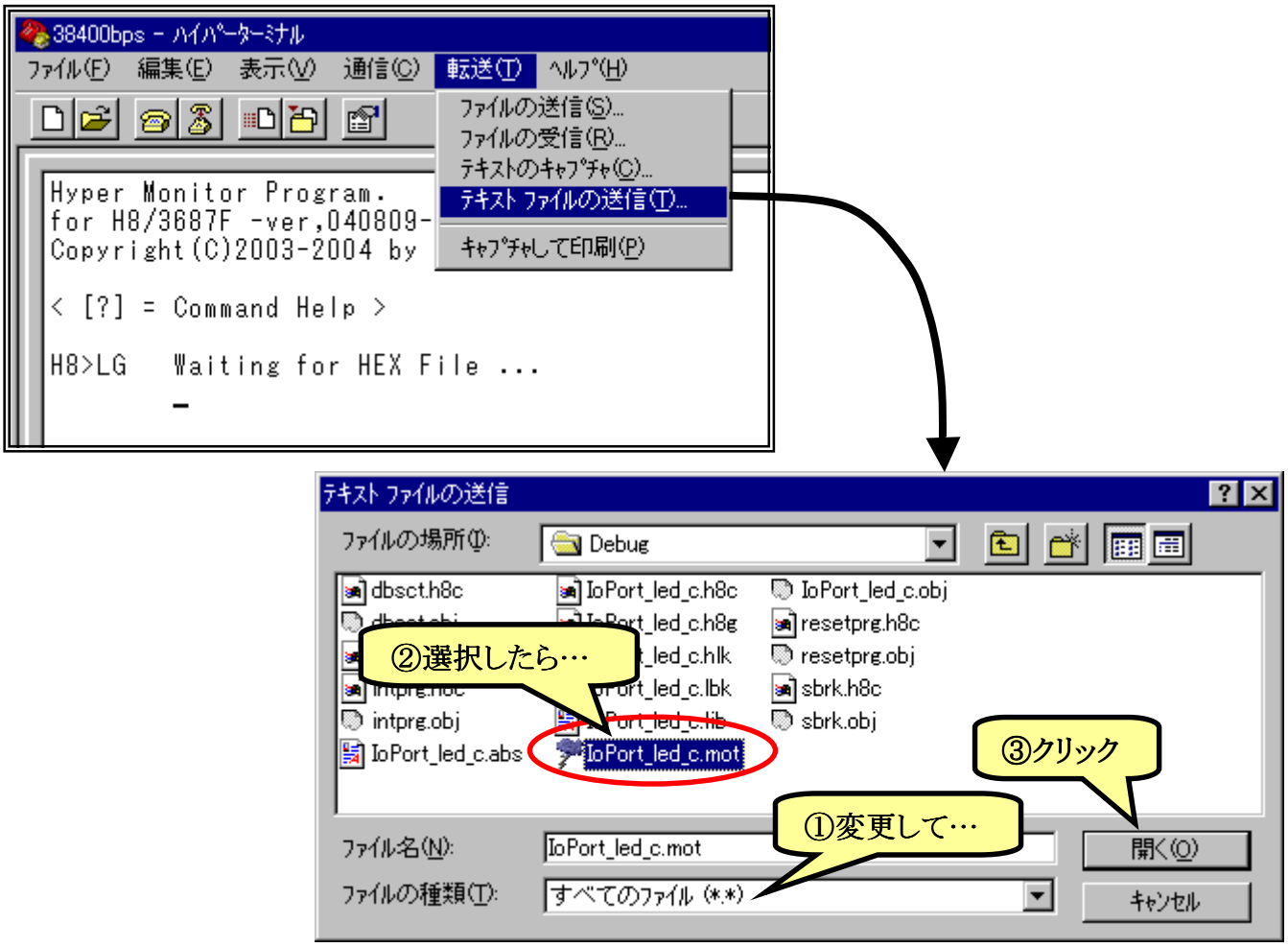
さて，図では「1 Warning」と表示されています。これは「まちがいではないかもしれないけど，念のため確認してね」という警告表示です。例えばこの図の「L1100(W) Cannot find "C" specifind in option "start"」は，C セクションを設定したのに C セクションのデータがないとき表示されます。今回は C セクションを使っていませんので，この警告が出ても何も問題ありません。（どうしても気になる方は，「H8S，H8/300 Standard Toolchain」のセクション設定でCセクションを削除してください。）

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。

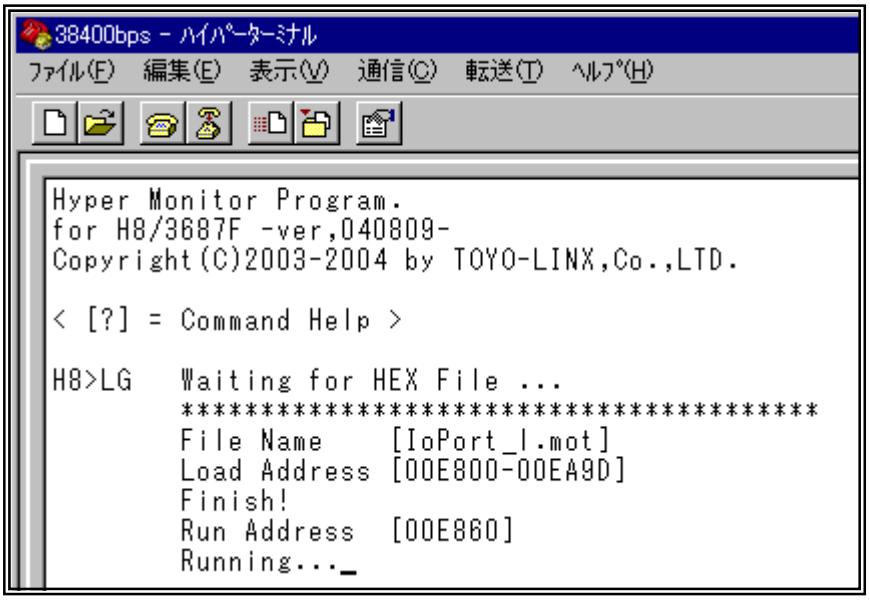
5. ダウンロードと実行

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。実行の方法はアセンブラと同じです。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12章 （応用編）アセンブラによるサーボモータの制御

1. サーボモータとは
2. サーボモータの制御方法
3. パルスを作り出す内蔵機能
4. サーボモータの接続
5. 制御プログラム例

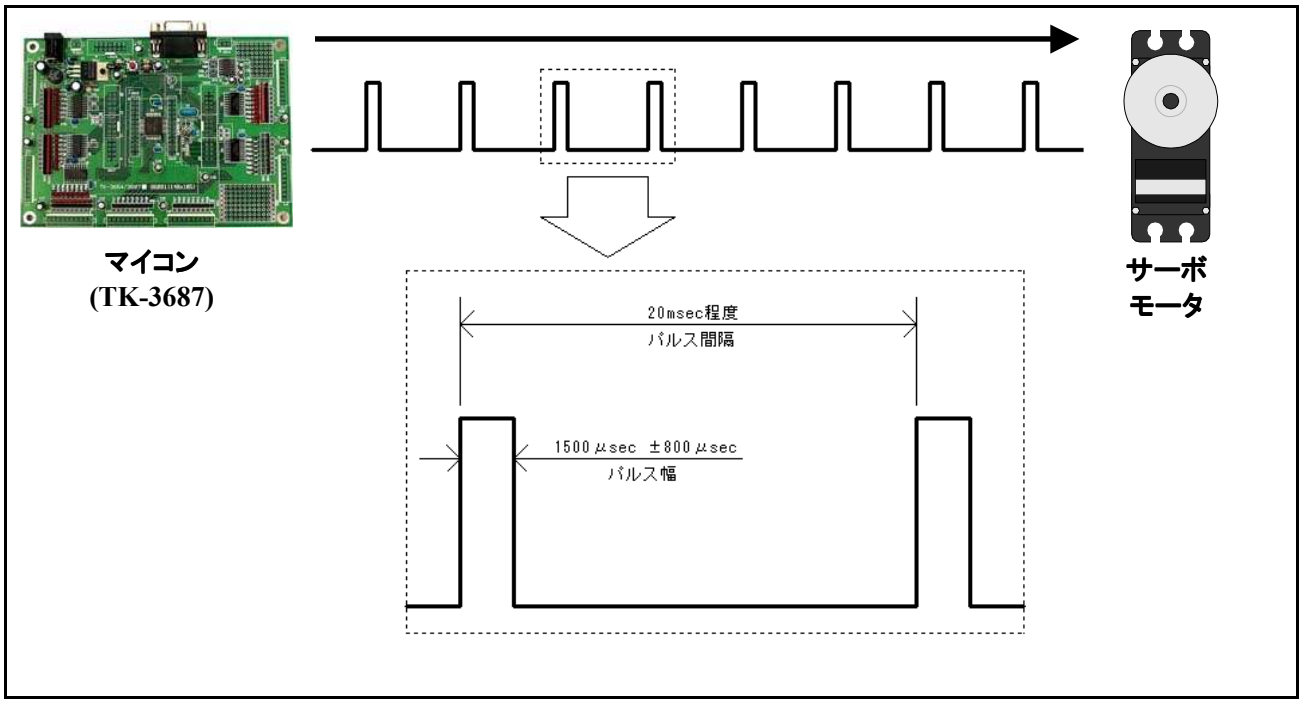
1. サーボモータとは

TK-3687 でサーボモータをコントロールしてみましょう。

ところでサーボモータがどのようなものかご存知でしょうか？テレビや雑誌等で二足歩行のロボットが格闘しているのをご覧になった事はないでしょうか？そういったロボットの関節やラジコンにサーボモータは使用されています。そのサーボモータは DC モータ、ポテンションメータ、ギヤ、制御回路をひとまとめにした物で、入力された信号（パルス）の幅に応じた角度を保持します。動作角度は 180° 程度です。

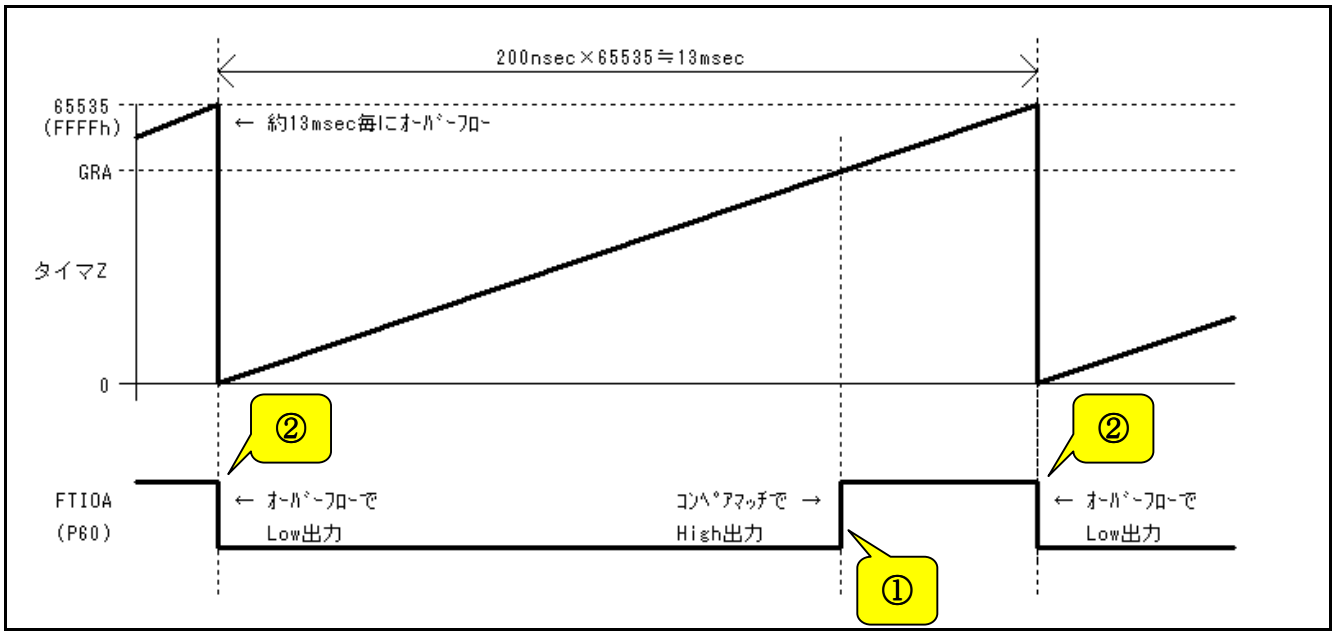
2. サーボモータの制御方法

ここではサーボモータの制御方法を説明します。サーボモータを制御するには 20msec 程度の間隔で角度に応じたパルスをサーボモータに入力します。パルスの幅は 1500μsecを中心に±800μsec です。動作角度が 180° ですと、パルス幅が 700μsec で 0° 、1500μsec で 90° 、2300μsec で 180° の角度となります。パルス幅は角度に関係するので正確に出力しなければなりませんが、パルスの間隔は大体で大丈夫です。後に紹介するプログラムでは約 13msec の間隔で出力しています。しかしパルスの間隔が長すぎたり無くなったりすると、サーボモータは角度を保持せずにフリーの状態になってしまいます。



3. パルスを作り出す内蔵機能“タイマZ”

さて、サーボモータを制御する為にどのような信号が必要かが分かりました。次はその信号をどのように作るか考えてみましょう。パルスの幅もパルスの間隔も全てプログラムで管理する事もできますが、ここでは CPU の内蔵機能を用いてパルスを作り出す事にします。その内蔵機能は“タイマZのコンペアマッチによる波形出力機能”というものです。タイマというのは一定の間隔でレジスタの値を＋1していく機能で、ソフトではなく CPU の中のハードによって動作しています。タイマには幾つかありますが、ここで使用するタイマは‘Z’というタイマです。コンペアマッチとはタイマによって＋1されている値と指定した値とが一致しているか比較する機能をさしています。つまり、タイマの値と指定した値が一致していたら波形を出力する、という機能です。文章だけでは分かりにくいと思うので、図を見ながら理解していきましょう。



タイマZの斜めの線がタイマによって＋1されているレジスタの値、GRAと書かれた点線が比較する為の値をセットするレジスタ、その下にかかれている FTIOA が出力される波形を表します。FTIOA は I/O ポート P60 と兼用ピンになっています。ですから、サーボモータへ出力する波形は P60 に出てきます。

① タイマZによって値がどんどん＋1されていくと、その内比較する値GRAと一致します。一致したら出力FTIOAをHigh出力します。

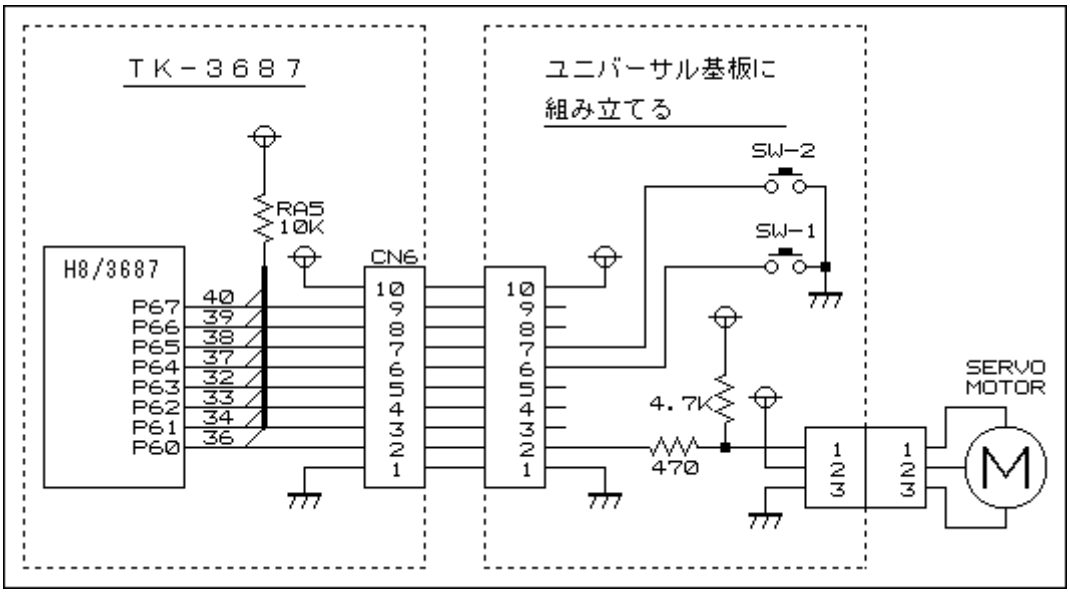
② タイマZによって＋1されていく値には上限があって、その上限になると0に戻ります。この状態をオーバーフローと言いますが、オーバーフローしたら出力 FTIOA をLowに戻します。

この①、②を繰り返す事で一定の間隔でパルスを出力する事が可能になります。タイマZによって＋1される時間は 200nsec 毎で、オーバーフローするまでに 65535 回カウントするので 200nsec×65535 回＝13.107msec、つまりパルスの間隔となります。比較する値 GRA はカウント数で指定するので、例えば 1500μsec のパルスを出力したいなら、(13.107msec－1500μsec) ÷200nsec＝58035 となります。サーボモータの角度を変えるには、比較する値 GRA を書き換えます。

尚、タイマ Z についての詳しくはルネサス“H8/3687 シリーズ　ハードウェアマニュアル”の 13 章‘タイマ Z’を参照して下さい。

4. TK-3687 とサーボモータの接続

次項で示すサンプルプログラムを動作させる為に、まずはサーボモータを TK-3687 に接続しましょう。サーボモータは付属していませんので模型店等で別途購入して下さい(メーカ:GWS 型番:S03T など)。ユニバーサル基板にサーボモータ接続用の回路を組みます。また、サンプルプログラムではスイッチも使用しますので、ユニバーサル基板にスイッチを 2 個実装します。回路図は下記の通りです。(「事始めキット」にはサーボモータのための部品は含まれていません。回路図を見ながら自分でそろえてみてください。趣味の工作は部品をそろえることも楽しむものです。)



サーボモータのコネクタを 3 ピンのコネクタに差し込みます。一般的に GND のケーブルは黒や茶など暗い色をしていますが、サーボモータの説明書等で確認してから差し込みます。間違えないように注意して下さい。

5. サーボモータの制御プログラム例

サーボモータの制御プログラム例“servo_asm”を示します。このプログラムではある一定の間隔で比較する値GRAを書き換えてサーボモータを連続的に動かしています。ただ一定の間隔で動かしだけでは面白味が無いのでスイッチで書き換える間隔を変えられるようにしました。P64のスイッチを押すとサーボモータの動作が速く、P65のスイッチを押すと逆に遅くなります。今まではプログラムの流れをフローチャートで表していましたが、ここではリストのみを示します。プログラムの流れよりも、サーボモータはどうすれば動かすことができるのか、それがこの章の主テーマだからです。もしこのプログラムを詳しく知りたいのなら、逆にリストからフローチャートを書いてみてはどうでしょうか？人が作ったプログラムを読む事はプログラムを理解する力を大きく伸ばしてくれます。リスト中のコメントと併せてこのプログラムを理解してみてください。

```
;---------------------------------------------------------------------------
;                                                                           |
;   FILE        :servo_asm.src                                              |
;   DATE        :Wed, Jan 12, 2005                                          |
;   DESCRIPTION :Main Program                                               |
;   CPU TYPE    :H8/3687                                                    |
;                                                                           |
;   This file is generated by Hitachi Project Generator (Ver.2.1).          |
;               Programed by Y.Furukawa / Toyo-linx,Co.,Ltd.                |
;                                                                           |
;---------------------------------------------------------------------------


    .include "io3687F_equ.inc"

;***************************************************************************
;    メインプログラム
;***************************************************************************
    .section P,code,locate=H'EA00

    .export      _main
_main:
;----- イニシャライズ ----------------------------------------------------
        bsr      INITPIO:16     ;PIO ｲﾆｼｬﾗｲｽﾞ
        bsr      INITTZ:16      ;ﾀｲﾏZ ｲﾆｼｬﾗｲｽﾞ

        mov.w    #0,r0
        mov.w    r0,@SQNO       ;ｼｰｹﾝｽ番号=0 ｾｯﾄ
        mov.b    r0l,@SW_FLG    ;ｽｲｯﾁ ﾌﾗｸﾞ=0

        mov.b    #D'10,r0l      ;ｼｰｹﾝｽ更新間隔初期値ｾｯﾄ
        mov.b    r0l,@WAIT_CNST

        andc #H'7F,ccr          ;割り込み許可

;----- メインループ ------------------------------------------------------
loop_00:
        bsr      SWREAD:16      ;ｽｲｯﾁ(=P64) ﾘｰﾄﾞ
        bne      loop_10        ;押されていたらloop_10へ
        bra      loop_00        ;押されるまで待つ

loop_10:
        mov.b    #H'01,r0l      ;SW_FLGをｾｯﾄ
        mov.b    r0l,@SW_FLG
        bsr      STARTTZ:16     ;ﾀｲﾏZ ｽﾀｰﾄ
loop_12:
        bsr      WAITANDSW:16   ;
        bsr      GETSQDT:16     ;次のｼｰｹﾝｽﾃﾞｰﾀ get
        bsr      SETGR:16       ;ｼﾞｪﾈﾗﾙﾚｼﾞｽﾀ ｾｯﾄ
        bra      loop_12        ;繰り返す
```

```
;***************************************************************************
;    割り込み処理
;***************************************************************************
;---------------------------------------------------------------------------
;    タイマＺ０割り込み
;---------------------------------------------------------------------------
    .export       INTRTZ0
INTRTZ0:                          ;ｵｰﾊﾞｰﾌﾛｰ割り込み
        push.l    er0

        mov.b     #H'00000000,r0l     ;FTIO出力=Low
        mov.b     r0l,@TOCR

        mov.b     @TSR_0,r0l          ;割り込みﾌﾗｸﾞ ｸﾘｱ
        and.b     #B'11001111,r0l
        mov.b     r0l,@TSR_0

        pop.l     er0
        rte


;***************************************************************************
;    サブルーチン
;***************************************************************************
;---------------------------------------------------------------------------
;        ウェイトとスイッチリード
;---------------------------------------------------------------------------
WAITANDSW:
        mov.b     @WAIT_CNST,r0l      ;ｼｰｹﾝｽ間隔をｾｯﾄ(ﾙｰﾌﾟ回数のｾｯﾄ)
        mov.b     r0l,@PDR5      ;******* ﾃﾞﾊﾞｯｸﾞ用 *******
        mov.b     #D'10,r1l
        mulxu.b   r1l,r0
        mov.b     r0l,@WAIT_CNT

WAITANDSW_00:
        bsr       SWREAD:16           ;ｽｲｯﾁ ﾘｰﾄﾞ
        bne       WAITANDSW_10        ; 押されていた(NE)らﾀﾞﾌﾞﾙﾘｰﾄﾞする
WAITANDSW_01:
        bsr       WAIT10m:16          ;10msec待ち
WAITANDSW_02:
        xor.b     r0l,r0l             ;ｽｲｯﾁが押されていなければSW_FLGはｸﾘｱ
        mov.b     r0l,@SW_FLG
WAITANDSW_04:
        mov.b     @WAIT_CNT,r0l       ;待ち時間の減算(ﾙｰﾌﾟ回数-1)
        dec.b     r0l
        mov.b     r0l,@WAIT_CNT
        bne       WAITANDSW_00        ;0になるまで繰り返す
        rts

WAITANDSW_10:                         ;ｽｲｯﾁのﾀﾞﾌﾞﾙﾘｰﾄﾞ
        mov.b     r0h,@SW_1ST
        bsr       WAIT10m:16          ;待ち(ﾁｬﾀﾘﾝｸﾞ除去も兼ねている)
```

```
        bsr     SWREAD:16           ;ｽｲｯﾁ ﾘｰﾄﾞ
        beq     WAITANDSW_02        ; 押されていなければｼｰｹﾝｽ間隔は更新しない
        mov.b   @SW_1ST,r0l
        cmp.b   r0l,r0h
        bne     WAITANDSW_04
        mov.b   @SW_FLG,r0l         ;押した時のみ検出する(ｴｯｼﾞ検出)
        bne     WAITANDSW_04        ; SW_FLG=0の時のみｼｰｹﾝｽ間隔を更新
        mov.b   #H'01,r0l           ;SW_FLGをｾｯﾄ
        mov.b   r0l,@SW_FLG

        mov.b   @WAIT_CNST,r0l      ;押されたｽｲｯﾁでｼｰｹﾝｽ間隔を加減算
        btst #4,r0h                 ;P64が押された?
        beq     WAITANDSW_12
        btst #5,r0h                 ;P65が押された?
        beq     WAITANDSW_14
        bra     WAITANDSW_04

WAITANDSW_11:
        bsr     WAIT100m:16         ;100msec待ち
        bra     WAITANDSW_04

WAITANDSW_12:                       ;P64が押された：ｼｰｹﾝｽ間隔＋１
        inc.b   r0l                 ;ｼｰｹﾝｽ間隔定数＋１
        cmp.b   #D'20,r0l           ;最高19dまで
        bcs     WAITANDSW_16
        mov.b   #D'19,r0l
        bra     WAITANDSW_16

WAITANDSW_14:                       ;P65が押された：ｼｰｹﾝｽ間隔－1
        dec.b   r0l                 ;最低01dまで
        bne     WAITANDSW_16
        mov.b   #H'01,r0l
WAITANDSW_16:
        mov.b   r0l,@WAIT_CNST
        mov.b   r0l,@PDR5     ;******* ﾃﾞﾊﾞｯｸﾞ用 *******
        bra     WAITANDSW_04

;-------------------------------------------------------------------------
;       スイッチ判定
;-------------------------------------------------------------------------
SWREAD:
        mov.b   @PDR6,r0l           ;ｽｲｯﾁ ﾘｰﾄﾞ(P64,65)
        mov.b   r0l,r0h             ; R0Hにﾎﾟｰﾄﾃﾞｰﾀ退避
        or.b #B'00110000,r0l        ; SWのbitに1をｾｯﾄ
        xor.b   r0h,r0l             ; SW判定 Z=押されていない / NZ=押された
        rts

;-------------------------------------------------------------------------
;       タイマＺ スタート
;-------------------------------------------------------------------------
STARTTZ:
        mov.b   #B'00000000,r0l     ;FTIO Low出力
        mov.b   r0l,@TOCR
```

```
        mov.b   #B'11111101,r0l    ;ﾀｲﾏZ0 ｽﾀｰﾄ
        mov.b   r0l,@TSTR
        rts

;-------------------------------------------------------------------------
;       次のシーケンスデータをＧｅｔ
;-------------------------------------------------------------------------
GETSQDT:
        mov.w   @SQNO,r0           ;ｼｰｹﾝｽNo,を進める
        inc.w   #1,r0
        cmp.w   #SQ_CNST,r0        ;全ｼｰｹﾝｽﾃﾞｰﾀ終了?
        bcs     GETSQDT_00
        xor.w   r0,r0              ;全ｼｰｹﾝｽﾃﾞｰﾀ終了後は始めに戻る
GETSQDT_00:
        mov.w   r0,@SQNO

        mov.w   #SQDT_TBL,r3       ;ｼｰｹﾝｽﾃﾞｰﾀの取得
        add.w   r0,r0              ;ﾜｰﾄﾞﾃﾞｰﾀなのでｼｰｹﾝｽ番号×２
        add.w   r0,r3              ;ﾃｰﾌﾞﾙｻｰﾁ
        mov.w   @r3,r0             ;ｼｰｹﾝｽﾃﾞｰﾀ取得
        rts

;-------------------------------------------------------------------------
;       ジェネラルレジスタへセット
;-------------------------------------------------------------------------
SETGR:
        mov.w   #5,r1
        mulxu.w r1,er0
        neg.w   R0
        mov.w   r0,@GRA_0
        rts

;-------------------------------------------------------------------------
;       １Ｏmsec Ｗａｉｔ
;-------------------------------------------------------------------------
WAIT10m:
        push.l  ER6
        mov.l   #20*10000/6,ER6
        bra     WAIT:16

;-------------------------------------------------------------------------
;       １ＯＯmsec Ｗａｉｔ
;-------------------------------------------------------------------------
WAIT100m:
        push.l  ER6
        mov.l   #20*100000/6,ER6
        bra     WAIT:16

;-------------------------------------------------------------------------
;       １sec Ｗａｉｔ
;-------------------------------------------------------------------------
WAIT1SEC:
        push.l  ER6
```

```
        mov.l   #20*1000000/6,ER6
        bra     WAIT:16

;-------------------------------------------------------------------------
;       Ｗａｉｔ
;-------------------------------------------------------------------------
WAIT:
        dec.l   #1,ER6
        bne     WAIT
        pop.l   ER6
        rts

;-------------------------------------------------------------------------
;       ＰＩＯ イニシャライズ
;-------------------------------------------------------------------------
INITPIO:
        mov.b   #B'00000000,r0l
        mov.b   r0l,@PCR6

    mov.b   #B'11111111,r0l         ;******* ﾃﾞﾊﾞｯｸﾞ用 *******
    mov.b   r0l,@PCR5               ;******* ﾃﾞﾊﾞｯｸﾞ用 *******
                                    ;******* ﾃﾞﾊﾞｯｸﾞ用 *******
        rts

;-------------------------------------------------------------------------
;       タイマーＺ イニシャライズ
;-------------------------------------------------------------------------
INITTZ:
        mov.b   #B'00000010,r0l     ;ﾌﾘｰﾗﾝﾆﾝｸﾞ,ｶｳﾝﾄ=φ/4
        mov.b   r0l,@TCR_0

        mov.b   #B'11111110,r0l     ;FTIOA0 = TZ0出力
        mov.b   r0l,@TOER

        mov.b   #B'00000000,r0l     ;ｺﾝﾍﾟｱﾏｯﾁまでLow出力
        mov.b   r0l,@TOCR

        mov.b   #B'10101010,r0l     ;ｺﾝﾍﾟｱﾏｯﾁでHigh出力
        mov.b   r0l,@TIORA_0

        mov.b   #B'11110000,r0l     ;ｵｰﾊﾞｰﾌﾛｰ割り込み許可
        mov.b   r0l,@TIER_0

        rts


;*************************************************************************
;       シーケンスデータテーブル・エリア
;*************************************************************************
;               0700=-90°: 1100=-45°: 1500=0°: 1900=+45°: 2300=+90°
SQDT_TBL:
    .data.w     D'0700,D'2300,D'0700
    .data.w     D'1100,D'1500,D'1900,D'2300,D'1900,D'1500,D'1100,D'0700
```

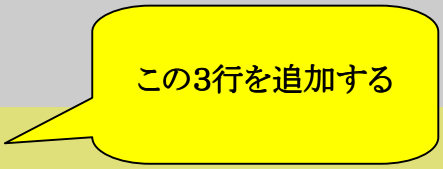
```
    .data.w       D'1500,D'0700,D'2300,D'1500,D'2300
SQDT_TBLE:
SQ_CNST: .equ (SQDT_TBLE-SQDT_TBL)/2


;*********************************************************************
;    データ・エリア
;*********************************************************************
        .section D,data,locate=H'F780

WORK_AREA:
SQNO:             .res.w   1          ;ｼｰｹﾝｽ番号
WAIT_CNST:        .res.b   1          ;ｼｰｹﾝｽ更新間隔定数
WAIT_CNT:         .res.b   1          ;ｼｰｹﾝｽ更新間隔ｶｳﾝﾀ
SW_1ST:           .res.b   1          ;ｽｲｯﾁ 初回ﾃﾞｰﾀ
SW_FLG:           .res.b   1          ;ｽｲｯﾁ 押された時のみ(ｴｯｼﾞ)検出用ﾌﾗｸﾞ
WORK_AREAE:

;====================================================================
        .end
```

あともう一息です。次のページをご覧下さい。

ソースの入力のほかに‘intprg.src’ファイルを修正します。何故修正するかというとタイマ Z のオーバーフローの検出に‘割り込み’と言う処理を使用しているからです。割り込みとは簡単に説明すると「今まで行なっていた処理を一旦中断して、別の処理を行なう」事です。このプログラムではオーバーフローで割り込みが発生し、FTIOA をLow にする事が割り込み処理になります。割り込みについては少々高度な処理なのでここでの説明はこれ位にして省きます。プログラムを動作させる為に必要な変更ですのでリストにならって追加作業を行なって下さい。

```
;------------------------------------------------------------------------
;                                                                       |
;   FILE        :intprg.src                                             |
;   DATE        :Wed, Jan 12, 2005                                      |
;   DESCRIPTION :Interrupt Program                                      |
;   CPU TYPE    :H8/3687                                                |
;                                                                       |
;   This file is generated by Hitachi Project Generator (Ver.2.1).      |
;                                                                       |
;------------------------------------------------------------------------



    .include    "vect.inc"
    .section    IntPRG, code
;1 Reserved
_INT_Reserved1
;2 Reserved
_INT_Reserved2
;3 Reserved
_INT_Reserved3
;4 Reserved
_INT_Reserved4
;5 Reserved
_INT_Reserved5
;6 Reserved
_INT_Reserved6
;7 NMI
_INT_NMI
;8 TRAP #0
_INT_TRAP0
;9 TRAP #1
_INT_TRAP1
;10 TRAP #2
_INT_TRAP2
;11 TRAP #3
_INT_TRAP3
;12 Address break
_INT_ABRK
;13 SLEEP
_INT_SLEEP
;14 IRQ0
_INT_IRQ0
;15 IRQ1
```

```
_INT_IRQ1
;16 IRQ2
_INT_IRQ2
;17 IRQ3
_INT_IRQ3
;18 WKP
_INT_WKP
;19 RTC
_INT_RTC
;20 Reserved
_INT_Reserved20
;21 Reserved
_INT_Reserved21
;22 Timer V
_INT_TimerV
;23 SCI3
_INT_SCI3
;24 IIC2
_INT_IIC2
;25 ADI
_INT_ADI

;26 Timer Z0
    .import     INTRTZ0
_INT_TimerZ0:
        jmp     @INTRTZ0

;27 Timer Z1
_INT_TimerZ1
;28 Reserved
_INT_Reserved28
;29 Timer B1
_INT_TimerB1
;30 Reserved
_INT_Reserved30
;31 Reserved
_INT_Reserved31
;32 SCI3_2
_INT_SCI3_2
    sleep
    nop
    .end
```



第13章 (応用編)C 言語による DC モータの制御

1. DC モータの回転方向制御
2. 回路を考えよう
3. プログラムを考えよう

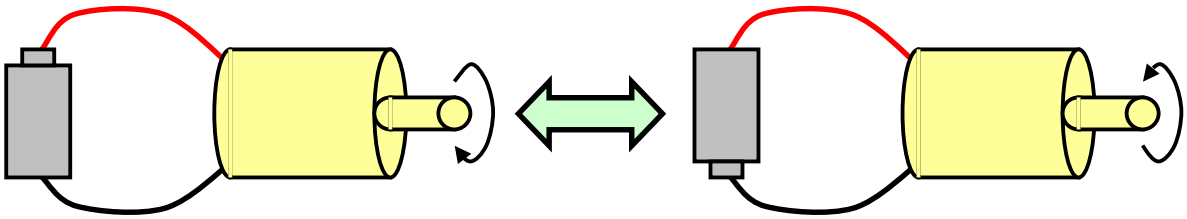
「事始めキット」にはタミヤ模型の「ツインモーターギヤーボックス」が入っていますが，いよいよ使うときがやってきました。題して，「リモコンカーを作ろう」です。

注意:「事始めキット」にはすべての部品は入っていません。部品表を見てそろえてください。

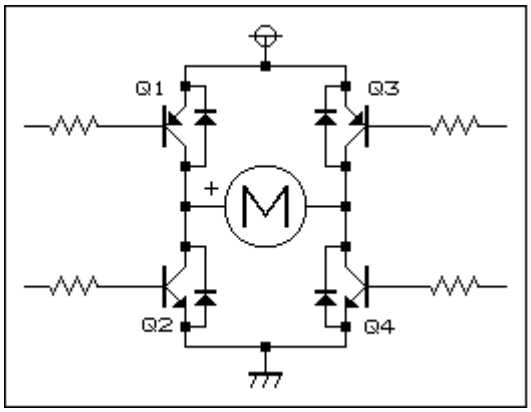
1. DC モータの回転方向制御

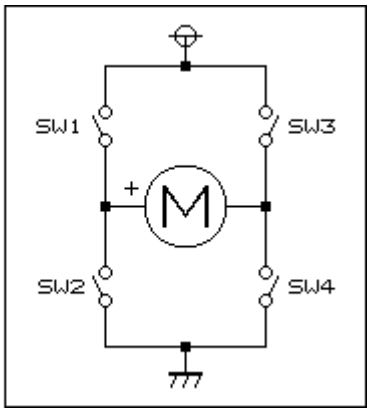
やはり車ですから，前にまっすぐ進むだけではなくて，カーブしたりバックしたりできないとおもしろくありません。そのためには，モータの回転方向を変えられないといけません。どうすればよいでしょうか。

答えは簡単で，モータの電源のプラスとマイナスをいれかえます。

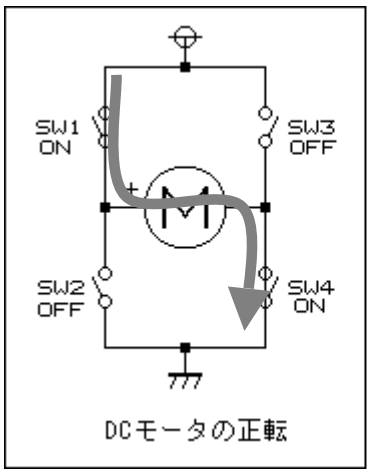


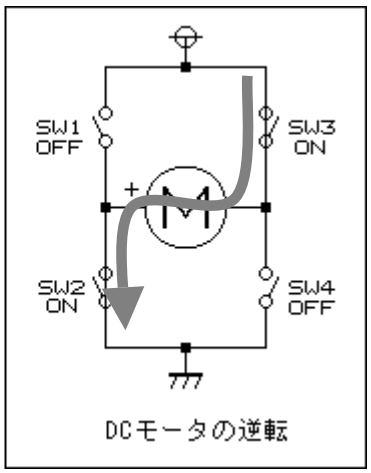
ただ，マイコンで制御する以上，上の絵のように電池をいれかえるわけにはいきません。そこで，トランジスタでスイッチ回路を作り，スイッチのオン/オフの組み合わせでモータの電源のプラスとマイナスをいれかえることにします。右の図をご覧下さい。左上の図はモータの回転方向をトランジスタで変えられるようにした回路です。このような回路を H ブリッジ回路といいます。右上の図は左上の回路の考え方を普通のスイッチでかいてみたものです。





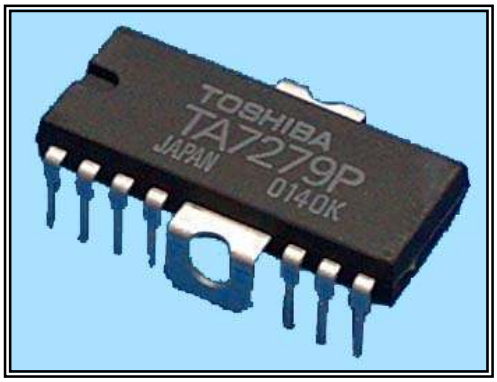
左下図のように SW1 と SW4 をオン，SW2 と SW3 をオフにすると，モータが正回転します。また，右下図のように SW1 と SW4 をオフ，SW2 と SW3 をオンにすると，モータが逆回転します。

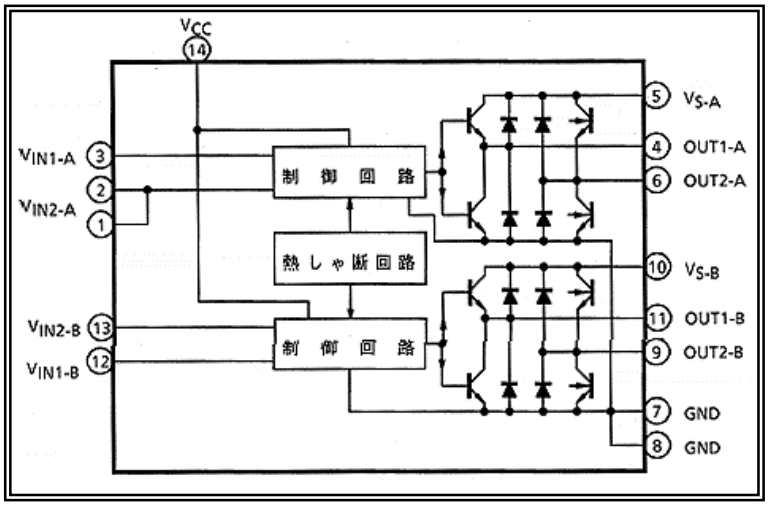




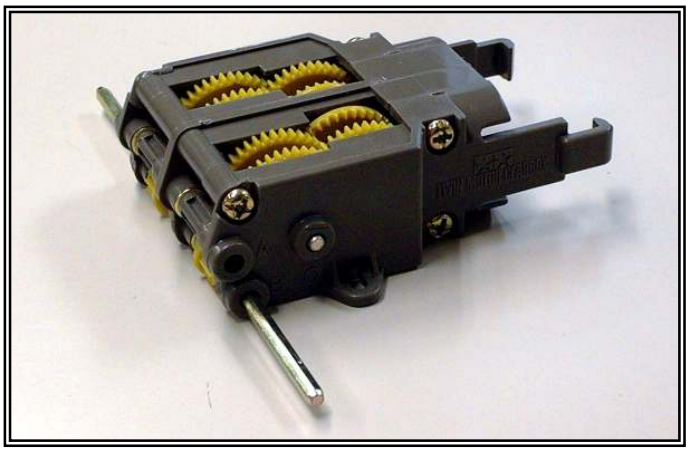
2. 回路を考えよう

ツインモーターギヤーボックスなので，モータは 2 個使います。ということは H ブリッジ回路が 2 つになるので，トランジスタが 8 個必要になります。抵抗やらなにやら配線することを考えると，やる気がうせてきますね。もっとも考えることはみんな同じで，H ブリッジ回路 2 つを一つの IC にしたものが世の中には存在します。今回はこれを使って楽をしましょう。東芝の TA7279P を使います。写真とブロック図を見てください。H ブリッジ回路が 2 個入っているのがわかりますね。





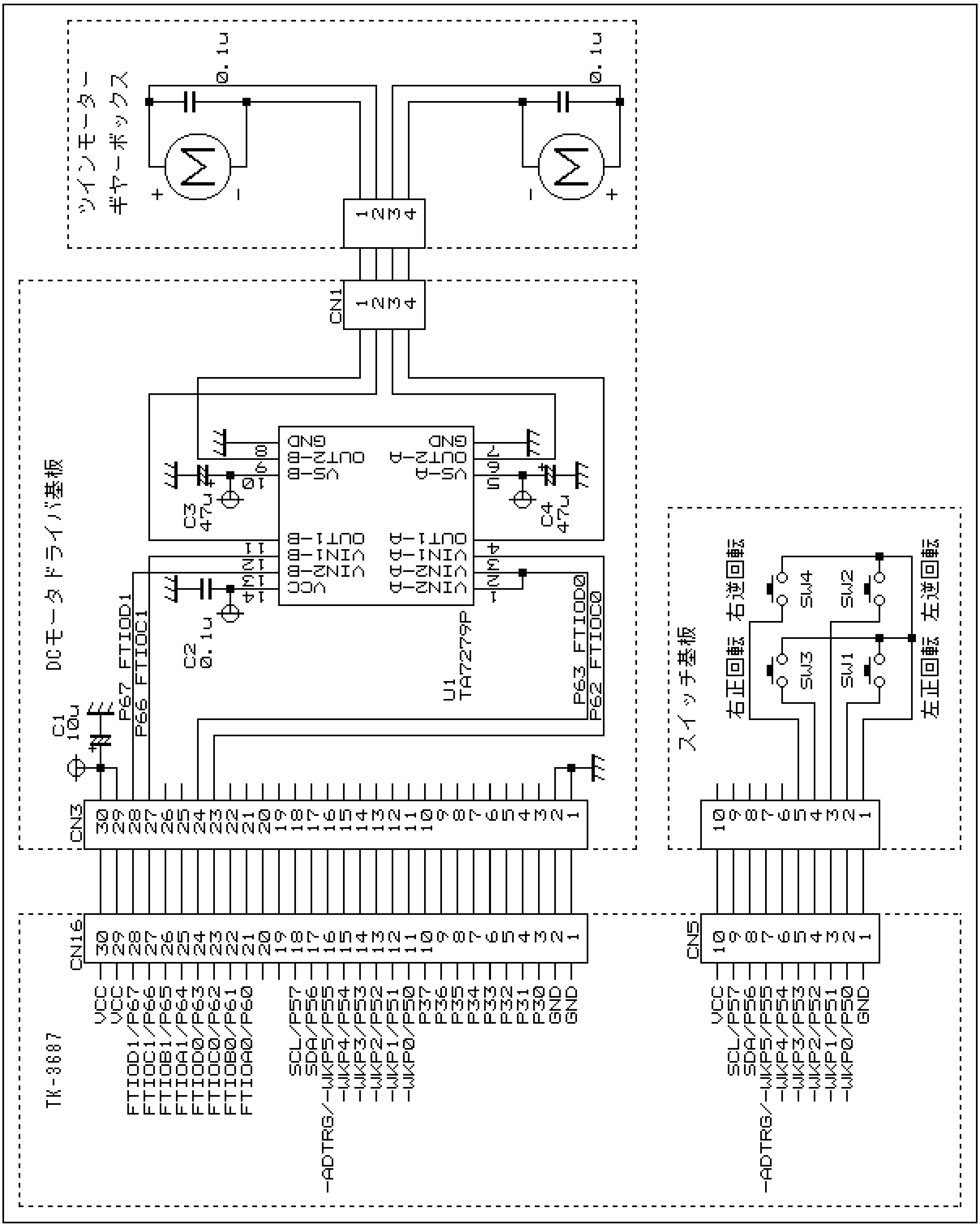
ツインモータギヤーボックスの写真からわかるように，DC モータは左右のタイヤを別々に動かします。カーブは左右のモータの回転差で対応します。



TK-3687 の CN16 と 30 芯フラットケーブルで DC モータドライバ基板と接続します。

操作スイッチは 4 つ使います。それぞれのモータの正回転と逆回転に割り当てます。スイッチ基板は別に作ってリモコン操作にしましょう。TK-3687 の CN5 につないで，P50~53 からスイッチを入力します。

というわけで，回路図は次のようになりました。


ツインモーター
ギヤーボックス
DCモータドライバ基板
U1
TA7279P
CN1
CN3
CN16
CN5
TK-3687
スイッチ基板
右正回転
右逆回転
左正回転
左逆回転
SW1
SW2
SW3
SW4
C1
10u
C2
0.1u
C3
47u
C4
47u
P67 FTIOD1
P66 FTIOC1
P63 FTIOD0
P62 FTIOC0

部品表は次のとおりです。

		部品名	メーカ	数	備考
1	●DC モータドライバ基板				
2	IC	TA7279P	東芝	1	
3	コンデンサ	0．1μF(積セラ)		1	
4		10μF/16V(電解)		2	
5		47μF/16V(電解)		2	
6	コネクタ	HIF3FC-30P-2.54DSA	HRS	2	
7		B4P-SHF-1AA	JST	1	
8	ユニバーサル基板			1	
9					
10					
11					
12	●スイッチ基板				
13	スイッチ	SKHHAK/AM/DC	ALPS	4	
14	丸ピンソケット	10 ピン		1	
15	ユニバーサル基板			1	
16					
17					
18					
19	●TK-3687				
20	コネクタ	HIF3FC-30P-2.54DSA	HRS	1	
21					
22	●モータ				
23	コンデンサ	0．1μF(セラミック)		2	モータの端子にハンダ付けする
24					
25	●ケーブル				
26	モータ用			1	4 芯
27	スイッチ基板用			1	10 芯
28	ドライバ基板用			1	30 芯
29					
30					

上記部品表は参考例とお考え下さい。相当品も使用可能です。回路図を見ながら手に入る部品で何とかならないか考えてみるのもいい勉強になりますよ。

◆

回路図を見ながら組み立ててください。自分なりに工夫してみるのも良いでしょう。

3. プログラムを考えよう

プログラムの中心はTA7279Pの制御です。次の表はTA7279Pのデータシートに載っているファンクションリストです。

ファンクション

IN1	IN2	OUT1	OUT2	MODE
1	1	L	L	BRAKE
0	1	L	H	CW/CCW
1	0	H	L	CCW/CW
0	0	ハイインピーダンス		STOP

この表と回路図をもとに，それぞれのスイッチが押されたときにポートに何を出力すればよいか考えます。IN1=0，IN2=1 で正回転，IN1=1，IN2=0 で逆回転になるように制御します。次のようになります。

左のタイヤ				
SW1	SW2	P62 （IN1）	P63 （IN2）	DC モータの動作
オフ	オフ	High	High	ブレーキ
オン	オフ	Low	High	正回転
オフ	オン	High	Low	逆回転
オン	オン	Low	Low	ストップ
右のタイヤ				
SW3	SW4	P66 （IN1）	P67 （IN2）	DC モータの動作
オフ	オフ	High	High	ブレーキ
オン	オフ	Low	High	正回転
オフ	オン	High	Low	逆回転
オン	オン	Low	Low	ストップ

この表から，SW1 がオンのとき P62=Low，SW2 がオンのとき P63=Low，SW3 がオンのとき P66=Low，SW4 がオンのとき P67=Low にすればいいわけですね。

さて，今回は C 言語でプログラムを書いてみました。CD-ROM から

‘RemoconCar. mot’

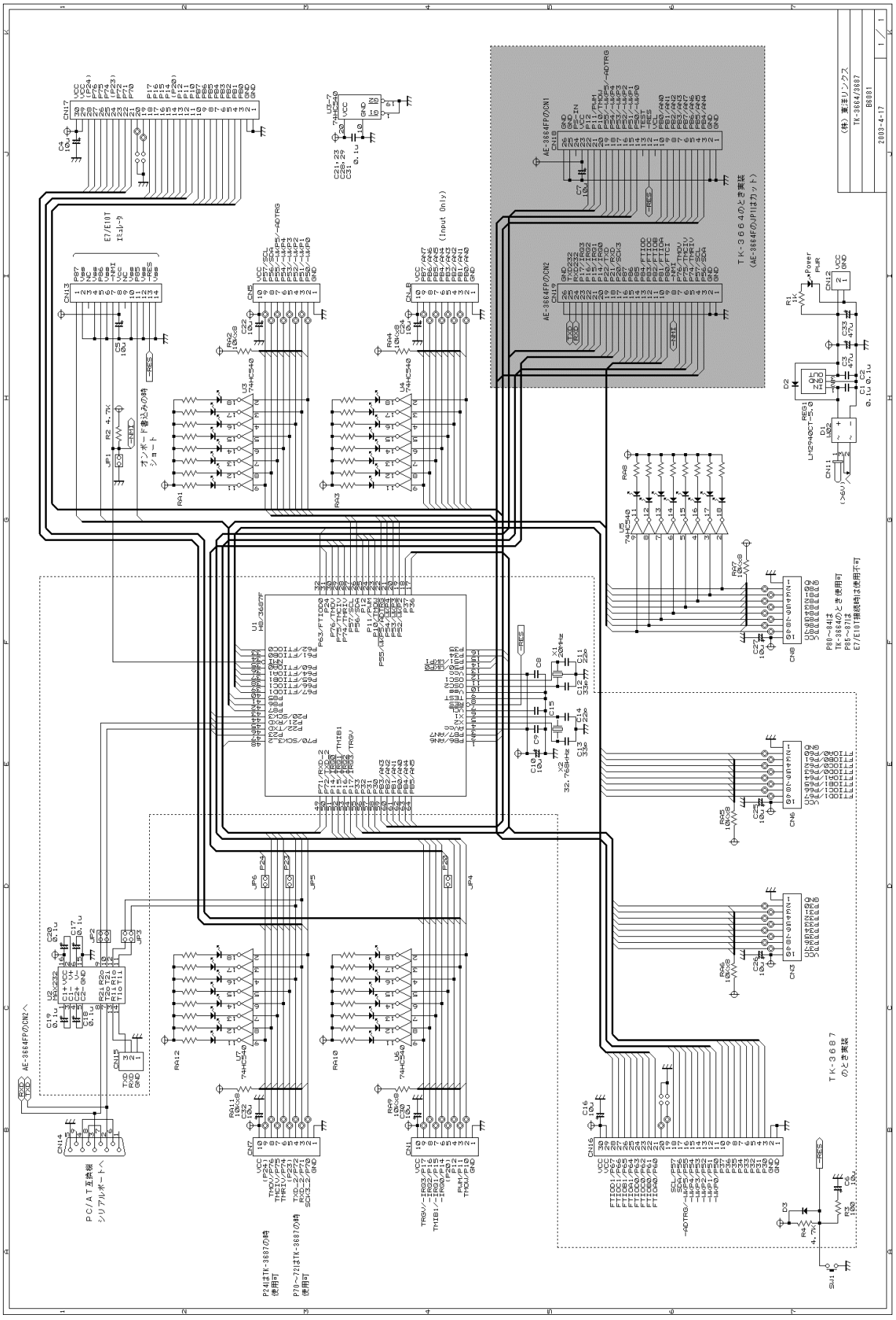
をダウンロードして実行して下さい。スイッチにあわせてちゃんと動きますか。もし動作が逆になっているときは，モータの配線が逆になっている可能性が大です。スイッチどおりに動くよう調整してください。

リストは次のとおりです。

```
/************************************************************************/
/*                                                                      */
/*  FILE        :RemoconCar.c                                           */
/*  DATE        :Thu, Feb 03, 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);

/************************************************************************
    メインプログラム
************************************************************************/
void main(void)
{
    IO.PCR5       =   0x00;     // ポート5を入力に設定
    IO.PUCR5.BYTE =   0x0f;     // ポート5のP50-53のプルアップをオン
    IO.PCR6       =   0xcc;     // ポート6のP62,63,66,67を出力に設定

    while(1){
        // SW1がオンのときP62をLowにする
        if (IO.PDR5.BIT.B0==0) {IO.PDR6.BIT.B2 = 0;}
        else                   {IO.PDR6.BIT.B2 = 1;}
        // SW2がオンのときP63をLowにする
        if (IO.PDR5.BIT.B1==0) {IO.PDR6.BIT.B3 = 0;}
        else                   {IO.PDR6.BIT.B3 = 1;}
        // SW3がオンのときP66をLowにする
        if (IO.PDR5.BIT.B2==0) {IO.PDR6.BIT.B6 = 0;}
        else                   {IO.PDR6.BIT.B6 = 1;}
        // SW4がオンのときP67をLowにする
        if (IO.PDR5.BIT.B3==0) {IO.PDR6.BIT.B7 = 0;}
        else                   {IO.PDR6.BIT.B7 = 1;}
    }
}
```

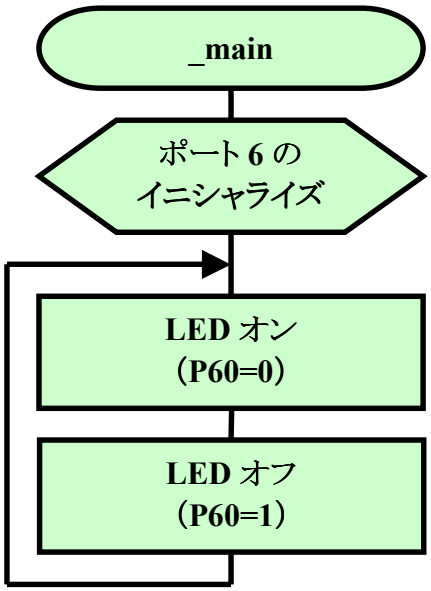
付録

回路図：TK-3687



コーディングの方法

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」を用意してください。このマニュアルにはH8/3687で使うことができる命令が全部説明されています。第4章で考えた右のフローチャートの項目ごとにどの命令を使うか考えてみましょう。



■ ポート6のイニシャライズ

I/O ポートの使い方については第 7 章で詳しく取り上げています。ここでは単純に，‘FFE9’番地に‘01’をセットすると P60 が出力ポートになる，とだけ覚えてください。さて，ある番地にデータをセットするには‘MOV’命令を使います。

ニーモニック	オペランド	命令の動作
MOV.B	Rs,@aa:8	8 ビットレジスタ Rs の内容を aa 番地に転送する。 ただし，FF00≦aa≦FFFF。

そうすると当然 8 ビットレジスタ R にデータをセットする命令も必要になります。（Rs の s はソースの意味です。転送元を表しています。）これも‘MOV’命令を使います。

ニーモニック	オペランド	命令の動作
MOV.B	#xx:8,Rd	イミディエットデータ xx を 8 ビットレジスタ Rd にセットする。 ただし，0≦xx≦FF。

同じ‘MOV’命令なのに違う働きをしているの？と思われたでしょうか。実は，‘MOV’命令の機能はデータを転送する，というのが正確な表現です。で，何と何の間でデータ転送をするのか表しているのがオペランドになります。（Rd の d はディストネーションの意味です。転送先を表しています。）

また，‘MOV’のあとに‘. B’がついていますが，これは何でしょうか？この‘. B’は BYTE の略で 8 ビットデータを表しています。つまり‘MOV. B’とは 8 ビットデータのデータ転送という意味です。この他に 16 ビットデータの場合は‘. W’（WORD），32 ビットデータの場合は‘. L’（LONG-WORD）がつきます。

というわけで，この部分の命令は次のようになります。なお，レジスタは R0L レジスタを使います。

```
MOV.B   #H'01,R0L
MOV.B   R0L,@H'FFE9
```

もうちょっとだけ説明を。まず，‘#’ですが，これはイミディエットデータを表します。イミディエットデータとは数値そのものです。また，‘@’はそのあとの数値がアドレスであることを表しています。

■ **LED オン(P60=0)**

I/O ポートの使い方については第 7 章で詳しく取り上げます。ここでは単純に，‘FFD9’番地にデータをセットするとポート 6 に出力される，とだけ覚えてください。あるアドレスの特定のビットをオフするときは‘BCLR’命令を使います。

ニーモニック	オペランド	命令の動作
BCLR	#xx:3,@aa:8	aa 番地のビット xx を 0 にクリアする。 ただし，0≦xx≦7，FF00≦aa≦FFFF。

それで，この部分の命令は次のようになります。

```
        BCLR    #0,@H'FFD9
```

■ **LED オフ(P60=1)**

あるアドレスの特定のビットをオンするときは‘BSET’命令を使います。

ニーモニック	オペランド	命令の動作
BSET	#xx:3,@aa:8	aa 番地のビット xx を 1 にセットする。 ただし，0≦xx≦7，FF00≦aa≦FFFF。

それで，この部分の命令は次のようになります。

```
        BSET    #0,@H'FFD9
```

■ **ジャンプ**

最後にループの先頭にジャンプさせます。‘BRA’命令を使います。

ニーモニック	オペランド	命令の動作
BRA	d:8	指定されたアドレスに無条件に分岐(ジャンプ)する。 ただし，アドレスはこの命令の次の命令のアドレスに d を加えたアドレス。分岐できる範囲はこの命令に対して-126～+128 バイト。

この命令だと近くにしかジャンプできない，と心配する方もいるかもしれませんね。でも，心配にはおよびません。ちゃんと次のような命令が準備されています。

ニーモニック	オペランド	命令の動作
BRA	d:16	指定されたアドレスに無条件に分岐(ジャンプ)する。 ただし，アドレスはこの命令の次の命令のアドレスに d を加えたアドレス。分岐できる範囲はこの命令に対して-32766～+32768 バイト。

ところで，分岐先のアドレスの説明がまわりくどいですね。単純にいうと現在のプログラムカウンタの値に d を加える，ということですが，この「現在のプログラムカウンタの値」というのがくせものなんですね。この辺は付録の次のページ「ハンドアセンブルの方法」で説明します。

さて，分岐先のラベルを‘LOOP’とすると，この部分の命令は次のようになります。

```
        BRA     LOOP
```

これで，コーディングは完了です。すべてをまとめると次のようなリストになります。

```
_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
```

ハンドアセンブルの方法

アセンブルにはパソコンのソフト(アセンブラ)で自動的に変換する方法と，変換表を見ながら人手で変換する方法があります(ハンドアセンブル)。ここでは第4章のプログラムを例にハンドアセンブルに挑戦します。マシン語に変換した結果は次のようなものでした。なぜこうなるのか考えてみましょう。

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					

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」を用意してください。各命令のページに「●オペランド形式と実行ステート数」という項目があります。その中に「インストラクションフォーマット」という部分がありますが，これがマシン語になります。では，やってみましょう。

プログラムの最初の命令は，

```
MOV.B   #H'01,ROL        ;ポート6のイニシャライズ
```

です。この命令についてはプログラミングマニュアルの91ページと92ページに記されています。「●オペランド形式と実行ステート数」という項目に表が載せられていて，いくつかのアドレッシングモードがあることがわかりますが，今回はイミディエットですね。この部分だけ抜き出してみましょう。

アドレッシングモード	ニーモニック	オペランド形式	インストラクションフォーマット								実行ステート数
			第1バイト	第2バイト	第3バイト	第4バイト	第5バイト	第6バイト	第7バイト	第8バイト	
イミディエット	MOV.B	#xx:8,Rd	F rd	IMM							2

まず第 1 バイトです。この表から上位 4 ビットは‘F’になることがわかりますが，‘rd’にはどんな値が入るのでしょうか。ここでプログラミングマニュアルの 28 ページを見てください。レジスタフィールドと汎用レジスタの対応表が載せられていますね。この表も抜き出してみましょう。

アドレスレジスタ 32 ビットレジスタ		アドレスレジスタ 16 ビットレジスタ		アドレスレジスタ 8 ビットレジスタ	
レジスタフィールド	汎用レジスタ	レジスタフィールド	汎用レジスタ	レジスタフィールド	汎用レジスタ
000	ER0	0000	R0	0000	R0H
001	ER1	0001	R1	0001	R1H
⋮	⋮	⋮	⋮	⋮	⋮
111	ER7	0111	R7	0111	R7H
		1000	E0	1000	R0L
		1001	E1	1001	R1L
		⋮	⋮	⋮	⋮
		1111	E7	1111	R7L

今回使うのは‘R0L’レジスタですから，この表からレジスタフィールドには 2 進数で‘1000’，つまり‘8’をセットすればよい，とわかります。というわけで，第 1 バイトは‘F8’になります。

次は第 2 バイトですが，‘IMM’でした。これはイミディエットデータそのもの，つまり xx を表しています。というわけで，第 2 バイトは‘01’です。

それで，‘MOV.B #H'01,R0L’をマシン語に変換すると‘F8’‘01’になります。

では，続けて 2 番目の命令をマシン語に変換してみましょう。

```
MOV.B   R0L,@H'FFE9
```

この命令はプログラミングマニュアルの 97 ページと 98 ページに記されています。例によって「●オペランド形式と実行ステート数」という項目にある表から関係ある部分だけ抜き出してみましょう。

アドレッシングモード	ニーモニック	オペランド形式	インストラクションフォーマット								実行ステート数
			第 1 バイト	第 2 バイト	第 3 バイト	第 4 バイト	第 5 バイト	第 6 バイト	第 7 バイト	第 8 バイト	
絶対アドレス	MOV.B	Rs,@aa:8	3 rs	abs							4

第 1 バイトは楽勝ですよね。‘rs’はレジスタフィールドですが，今回使うレジスタも‘R0L’ですから，レジスタフィールドには 2 進数で‘1000’，つまり‘8’をセットすればよい，とわかります。というわけで，第 1 バイトは‘38’です。

第 2 バイトは‘abs’です。これは絶対アドレスの値，つまり aa の下位 8 ビットを表しています。というわけで，第 2 バイトは‘E9’です。

それで，‘MOV.B R0L,@H'FFE9’をマシン語に変換すると‘38’‘E9’になります。

それでは 3 番目の命令です。

```
BCLR    #0,@H'FFD9          ;LEDオン (P60=0)
```

この命令はプログラミングマニュアルの 41 ページに記されています。「●オペランド形式と実行ステート数」という項目にある表を抜き出してみましょう。

アドレッシングモード	ニーモニック	オペランド形式	インストラクションフォーマット								実行ステート数
			第 1 バイト		第 2 バイト	第 3 バイト		第 4 バイト			
絶対アドレス	BCLR	#xx:3, @aa:8	7	F	abs	7	2	0	IMM	0	8

第 1 バイトはそのまま‘7F’です。

第 2 バイトは‘abs’です。これは絶対アドレスの値，つまり aa の下位 8 ビットを表しています。というわけで，第 2 バイトは‘D9’です。

第 3 バイトはそのまま‘70’です。

第 4 バイトですが，‘IMM’を含んでいます。これはイミディエットデータそのもの，つまり xx を表しています。というわけで，第 4 バイトは‘00’です。

それで，‘BCLR #0,@H'FFD9’をマシン語に変換すると‘7F’‘D9’‘72’‘00’になります。

4 番目の命令です。

```
BSET    #0,@H'FFD9          ;LEDオフ (P60=1)
```

この命令はプログラミングマニュアルの 50 ページに記されています。「●オペランド形式と実行ステート数」という項目にある表を抜き出してみましょう。

アドレッシングモード	ニーモニック	オペランド形式	インストラクションフォーマット								実行ステート数
			第 1 バイト		第 2 バイト	第 3 バイト		第 4 バイト			
絶対アドレス	BSET	#xx:3, @aa:8	7	F	abs	7	0	0	IMM	0	8

第 1 バイトはそのまま‘7F’です。

第 2 バイトは‘abs’です。これは絶対アドレスの値，つまり aa の下位 8 ビットを表しています。というわけで，第 2 バイトは‘D9’です。

第 3 バイトはそのまま‘70’です。

第 4 バイトですが，‘IMM’を含んでいます。これはイミディエットデータそのもの，つまり xx を表しています。というわけで，第 4 バイトは‘00’です。

それで，‘BSET #0,@H'FFD9’をマシン語に変換すると‘7F’‘D9’‘70’‘00’になります。

最後の命令です。

```
BRA     LOOP            ;LOOPにジャンプ
```

この命令はプログラミングマニュアルの 39 ページと 40 ページに記されています。例によって「●オペランド形式と実行ステート数」という項目にある表から関係ある部分を抜き出してみましょう。

アドレッシングモード	ニーモニック	オペランド形式	インストラクションフォーマット				実行ステート数
			第 1 バイト	第 2 バイト	第 3 バイト	第 4 バイト	
プログラムカウンタ相対	BRA(BT)	d:8	4 0	disp			4

第 1 バイトはそのまま‘40’です。問題は第 2 バイトですが，‘disp’はディスプレースメントと呼ばれていて，このときのプログラムカウンタに加える値をセットします。今回セットする値は‘F6’ですが，なぜそうなるのでしょうか。

CPU にはプログラムカウンタ（以降 PC）と呼ばれるレジスタがあり，CPU が次に実行する命令のアドレスがセットされています。

CPU は，まず PC の示すアドレスからデータを取り出してどんな命令か解析します。例えば今回のプログラムで，PC=EA00 だったとすると，‘F8’，次に‘01’を取り出します。この時点で CPU は，「次に実行するのは‘MOV.B #H'01,R0L’という命令だ」と判断し実行します。

さて，ここで注意したいのは，PC はデータを取り出すたびに+1 されるので，‘01’を取り出し終わったときには PC=EA02 になっている，という点です。つまり，‘MOV.B #H'01,R0L’を CPU が実行する時には PC=EA02 になっています。

このことを頭において，PC=EA0C のときを考えてみましょう。CPU は‘40’，‘F6’と取り出し，この時点で，「次に実行するのは‘BRA LOOP’という命令だ」と判断します。そして，この命令を実行するときには PC=EA0E になっています。

‘BRA’命令を簡単にいうと，現在の PC にディスプレースメントを加える，というものです。この「現在の PC」が「命令を実行する時の PC」というところが鍵で，命令がセットされている EA0C ではなく EA0E になります。それで，EA0E にディスプレースメント F6（これは 2 の補数なので 10 進数で-10 を表している）を加えて EA04 番地にジャンプすることになります。

実際にマシン語にする時の手順としては，①‘BRA’命令を実行する時の PC は EA0E，②ジャンプ先‘LOOP:’は EA04 番地なので，③EA04-EA0E=-10（10 進数）がディスプレースメント，④-10 を 2 の補数で表した F6 をセットする，ということになります。

◆

こうやって考えると，マイコンの数字の羅列にもちゃんと意味があることがよくわかりますよね。

2 の補数

2 進数でマイナスの数を表す方法の一つで，コンピュータの世界でよく使われています。

8 ビットデータのとき:		
10 進	2 進	16 進
−128	10000000	80
−127	10000001	81
＼	＼	＼
−2	11111110	FE
−1	11111111	FF
0	00000000	00
1	00000001	01
＼	＼	＼
126	01111110	7E
127	01111111	7F

16 ビットデータのとき:		
10 進	2 進	16 進
−32768	1000000000000000	8000
−32767	1000000000000001	8001
＼	＼	＼
−2	1111111111111110	FFFE
−1	1111111111111111	FFFF
0	0000000000000000	0000
1	0000000000000001	0001
＼	＼	＼
32766	0111111111111110	7FFE
32767	0111111111111111	7FFF

2 の補数は，ビット反転して，+1 することで，機械的に作ることができます。

例:-10 の場合(8 ビットデータ)
①10 は 2 進数で‘00001010’　②ビット反転すると‘11110101’
③+1 すると‘11110110’　④16 進数にすると‘F6’

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

20130603