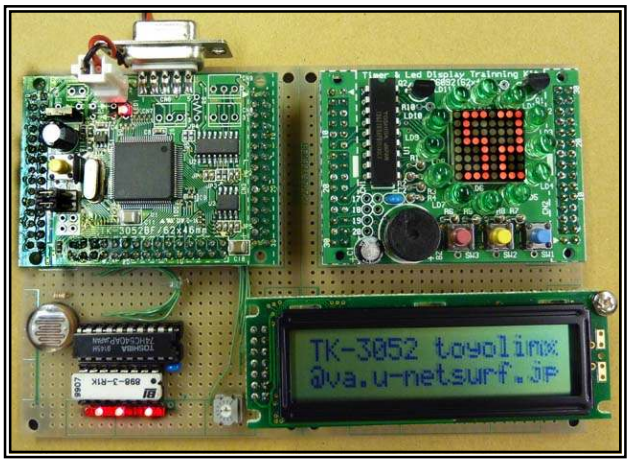
TK-3052版
マイコン事始め
～はじめてのマイコン～
Version 2.00



目次

(株)東洋リンクス

はじめに

コンピュータというと多くの方はパソコンをイメージすると思います。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（パソコンではなくコンピュータです）。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。携帯電話，メモリオーディオプレイヤー，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-3052 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-3052 で採用されている H8/3052 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。H8/3052 は日立によって開発が始まった，H8/300H CPU を核にしてシステム構成に必要な周辺機能を集積した高性能シングルチップマイクロコンピュータです。H8 シリーズは現在，日立と三菱が共同で設立し，後に NEC が加わった，ルネサスエレクトロニクスが製造・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-3052 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 ♯♭♪♯♭♪

ルネサスエレクトロニクスのサイト（http://japan.renesas.com）から，マニュアルのダウンロードができます。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。

「H8/3052B F-ZTAT 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。

♯♭♪♯♭♪ 次のページから始まる学習の前に ♯♭♪♯♭♪

組み立てキットを購入された皆さんは，ハードウェアを完成させてください。組み立て方法は巻末の付録で説明しています。

次に開発環境を準備します。「HEW」と「FDT」，「Hterm」を準備します。詳細は弊社 CD に含まれている，「TK-3052 ユーザーズマニュアル」の第 3 章と第 4 章，同じく CD に含まれている「ルネサスダウンロード. pdf」をご覧ください。「Hterm」が動作するところまで確認してください。

なお，完成品の TK-3052 の場合「Hterm」で使用する「組み込み型モニタ」はダウンロード済みです。パソコン側のプログラムを準備すれば「Hterm」が動作しまので，「Hterm」が動作するところまで確認してください。

♯♭♪♯♭♪ プログラムのソースリストについて ♯♭♪♯♭♪

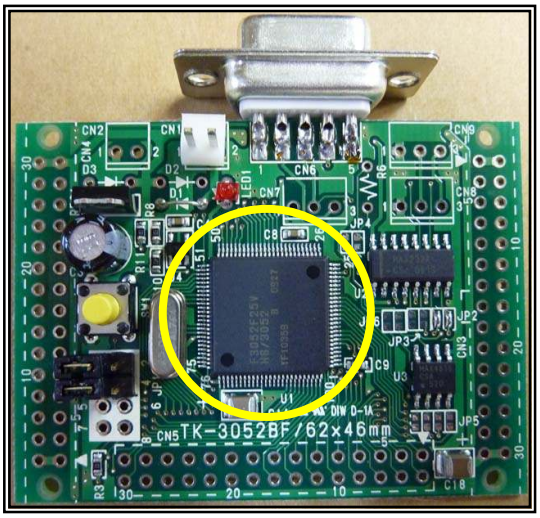
このマニュアルでは一部を除きソースリストは関係する部分のみ抜粋しています。ソースリストは CD に含まれていますので，そちらをご覧ください。（場所 CD:¥TK-3052¥事始めﾌﾟﾛｸﾞﾗﾑ¥）

第1章 TK-3052 の基礎知識

1. TK-3052 の構成
2. CPU について
3. メモリについて

1. TK-3052 の構成

まずは TK-3052 を見てみましょう(右写真参照)。基板の中央に LSI(H8/3052)が 1 個のっています。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。

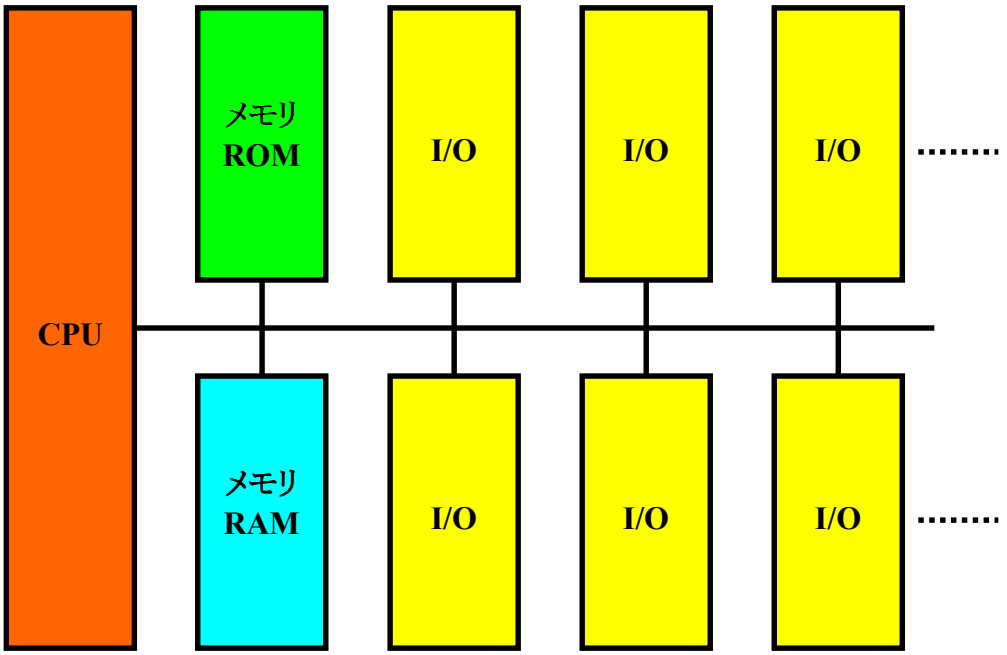


■ マイコンの 3 要素

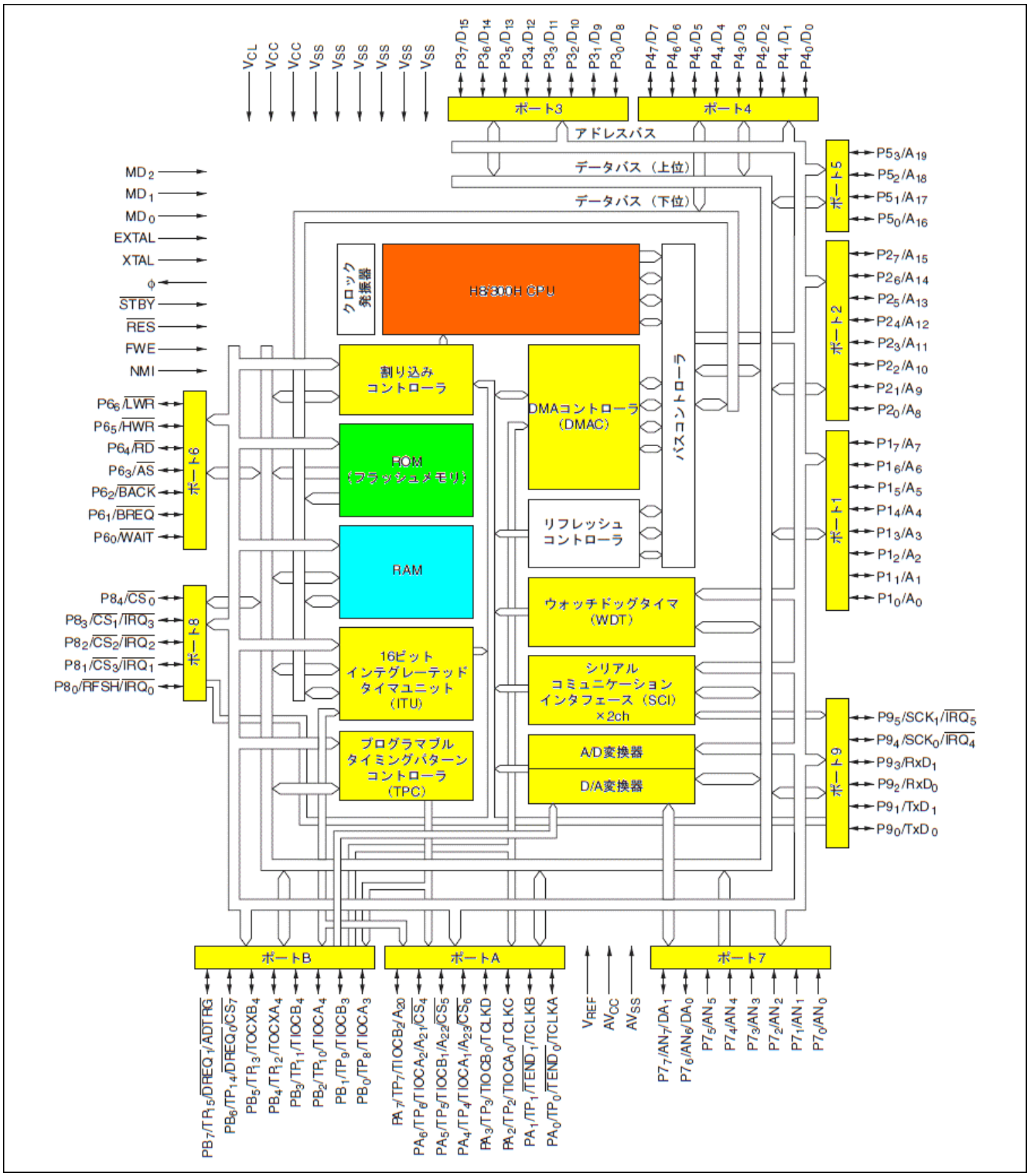
どんなマイコンでも次の基本的な 3 つの要素からできています。もちろん H8/3052 も例外ではありません。

- 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したりするのが I/O です。基本的には右のような構成になります。



昔，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，現在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-3052 で使っている H8/3052 もワンチップマイコンです。H8/3052 に何が内蔵されているか次の図をご覧下さい。マイコンの 3 要素の全てが入っていることがわかります。



2. CPU について

H8/3052 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，その命令に従って演算したり，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。このマニュアルでは C 言語がメインになっていますので，レジスタを意識することはそれほどないのが現実です。しかし，C 言語からアセンブラで記述されたサブルーチンをコールするなど，ハイレベルな使い方をするときにはレジスタを避けて通ることはできません。それで，ここでまとめて取り上げます。

■ 汎用レジスタ

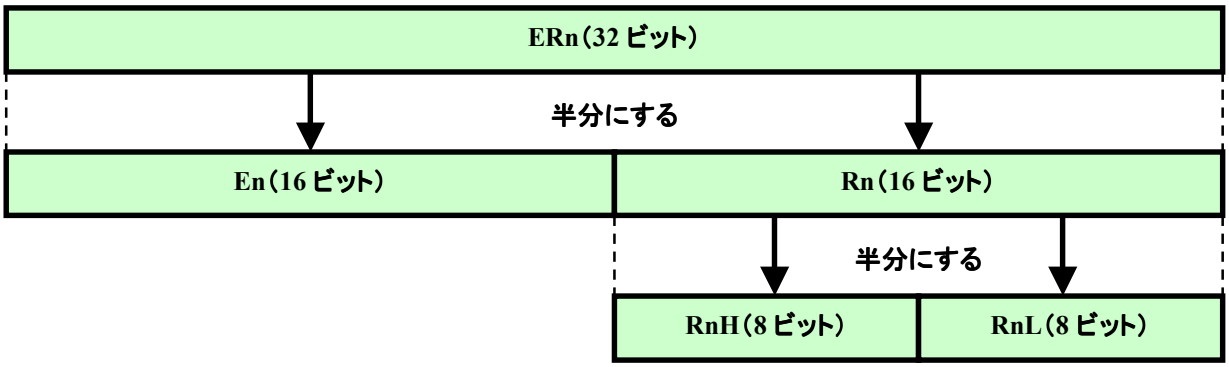
H8/300H は 32 ビット長の汎用レジスタを 8 本持っていて，ER0～ER7 という名前がつけられています。

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）

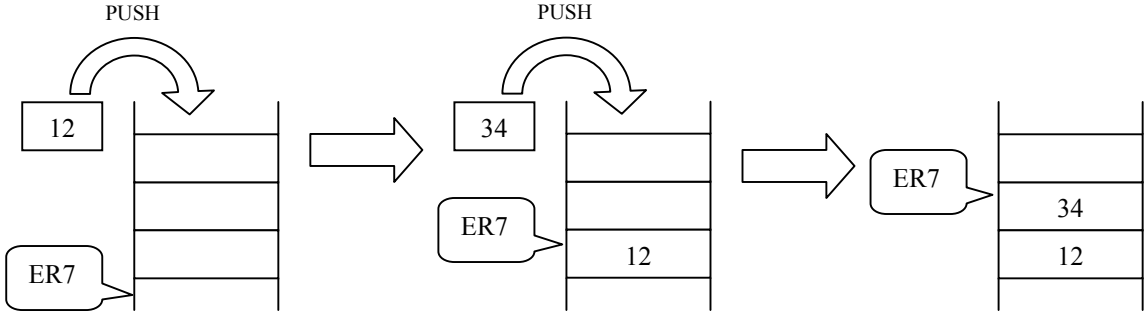


汎用レジスタは全て同じ機能を持っているのですが，ER7 だけは汎用レジスタとしての機能にプラスして，スタックポインタ（SP）としての機能も持っています。

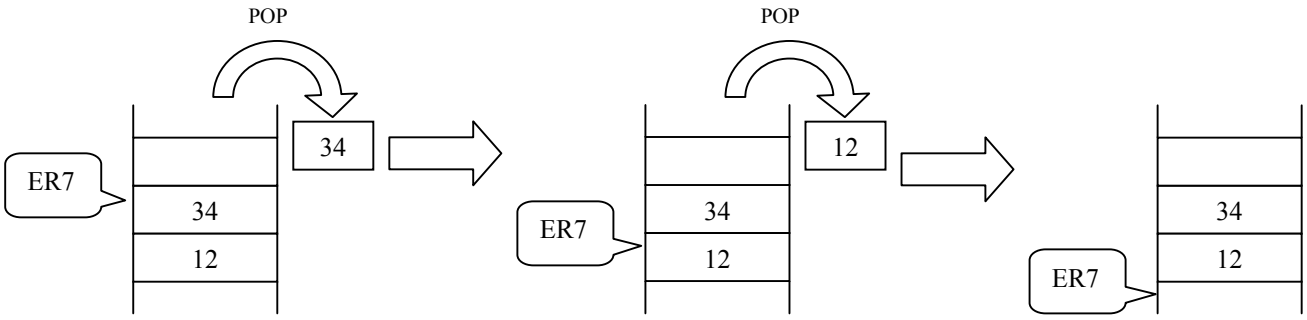
■ スタックポインタ

H8/300H にはスタックと呼ばれるデータ領域があります。スタックは FILO（First In Last Out：先入れ後出し）のデータ構造で，スタックをどこまで使用しているかをセーブするレジスタをスタックポインタと呼びます。スタックにデータを入れることを PUSH と呼び，逆にスタックからデータを取り出すことを POP と呼びます。

PUSH のときは ER7 を更新（-2 または-4）したあと ER7 が指しているアドレスにデータをセットします。



POP のときは ER7 が指しているアドレスのデータを取り出したあと ER7 を更新（＋2 または＋4）します。



関数をコールするときの戻り先のアドレスをセーブしたり，割り込み処理を行なう際にレジスタを保護したりする際に，スタックはよく使われます。

■ **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。

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。

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。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たりする(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。メモリの場合は 16 進数で表します。例えば，「FE400 番地から実行する」という感じです。

さて，H8/3052 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。H8/3052 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。なお，フラッシュメモリを書き換えるためには‘FDT’という道具を使います(FDT については「TK-3052 ユーザーズマニュアル」をご覧下さい)。

RAM は Random Access Memory の略で，いつでも自由にリード/ライトすることができます。その代わり，電源をオフすると全て忘れてしまいます。というわけで，普通はプログラム中で変更するデータをここに記憶させておきます。もちろん，RAM にプログラムを書き込んでも，そのプログラムを実行することはできます(あとでででてくる Hterm 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。電源をオンするたびに新しいまっさらなノートを準備し，電源をオフするとまるごとごみ箱に捨ててしまう，という感じです。

H8/3052 のメモリの広さは最大 16M バイト(アドレスは 0 番地から FFFFFF 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。基本的なメモリマップは次のとおりです。

000000 番地 07FFFF 番地	内蔵 ROM/フラッシュメモリ (512K バイト)
080000 番地 FFDF0F 番地	外部アドレス空間
FFDF10 番地 FFFF0F 番地	内蔵 RAM (8K バイト)
FFFF10 番地 FFFF1B 番地	外部アドレス空間
FFFF1C 番地 FFFFFF 番地	内部 I/O レジスタ

ところで，基本的なメモリマップはこのとおりなのですが，H8/3052 には幾つかの動作モードがあり(モード 1～7)，それによってメモリマップが多少かわります。内蔵 ROM を使うか使わないか，最大アドレス空間は 1M バイトか 16M バイトか，データバス幅は 8 ビットか 16 ビットか，外部メモリを使うか使わないか，各モードをハードで設定します。TK-3052 は H8/3052 の全ての端子をコネクタに接続しているので，全てのモードを使うことができます。ただし，外部メモリは実装されていませんので，ユーザが自分でハードを設計する必要があります。また，モードによっては内部 I/O のうち使うことができなくなる機能があります。

このマニュアルでは，H8/3052 をモード 7 で動かすことを前提にしています。モード 7 はシングルチップアドバンストモードで，最大アドレス空間は 1M バイト，内蔵 ROM と内蔵 RAM だけを使用し，全ての内部 I/O を使うことができます（兼用端子の関係で同時に使えない I/O はある）。モード 7 のメモリマップは右のとおりです。

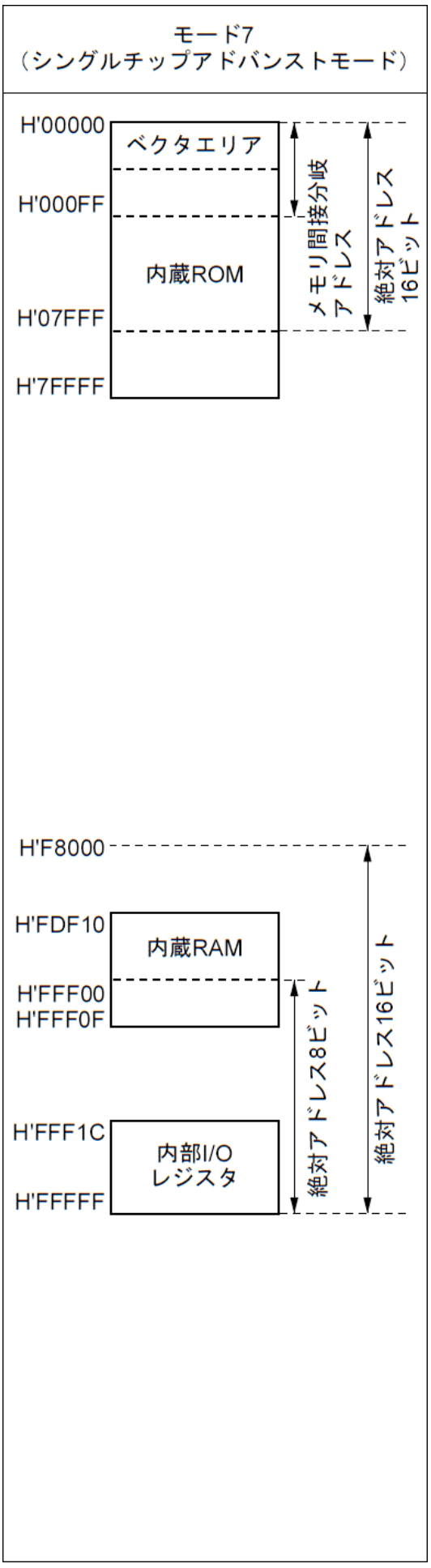
また，このマニュアルではモニタプログラムとしてルネサスエレクトロニクスの「Hterm」と「組み込み型モニタ」を使いますが，モニタが使用する領域，ならびにユーザが使用できる領域は次のとおりです。

シリアルポート	：	チャンネル 1
ボーレート	：	38400 ボー
動作モード	：	モード7（シングルチップアドバンストモード）
モニタベクタ	：	00000～000FF
モニタプログラム	：	00100～059EB
モニタが使う RAM	：	FDF10～FDFE3
ユーザベクタ	：	FE000～FE0FF
ユーザ使用可能エリア	：	FE100～FFCFF （ここがユーザのプログラムとワークエリア）
ユーザスタック初期値	：	FFF00 （スタックエリアは FFD00～FFEFF の 200h バイト）

それで，ユーザが作成するプログラムは FE100～FFCFF 番地の範囲に，プログラムとワークエリアが入るようにしなければなりません。

◆

さて，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）0000 1010 = 0A

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。マイコンの世界も 0 か 1 です。それで，本当はマイコンも 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，D'10

2 進数　：（例）B'00001010，_00001010B

16 進数：（例）H'0A，0x0A，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，H8/3052 の場合は，000000～FFFFFF 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 H8/3052 のモード 7 における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 もかわりないからです。このような割付け方をメモリマップド I/O と呼びます。

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

これはどちらが優れているというわけではありません。単に思想の違いです。

第2章 マシン語によるプログラムの作成

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

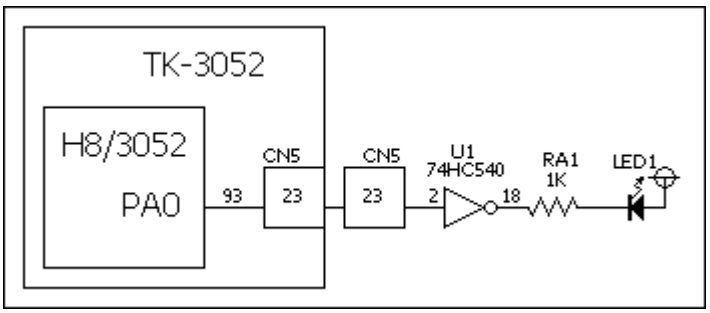
プログラムの作り方を理解するための早道は，とにかくたくさん作ってみる，ということです。くりかえし作っているうちに段々プログラミングの考え方が身についてきます。

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。そこで第 2 章と第 3 章では，何もないところからプログラムを作り始めて，TK-3052 で動かすまでの流れを説明します。ここで作るプログラムは非常に簡単なものです。命令の細かい部分はルネサスの資料，「H8/300H シリーズ プログラミングマニュアル」で，もっと詳しい説明がされています。興味のある方はお読みください。

さて，組込み系の分野では，コンピュータの仕組みやプログラムの動作の本質を知ることはとても大事なことです。そのためにはマシン語プログラムが最適です。それで，C 言語（第 3 章以降で説明）を学ぶ前にマイコンがどのようにプログラムを動かしているのか、マシン語の世界をちょっとだけのぞいてみましょう。

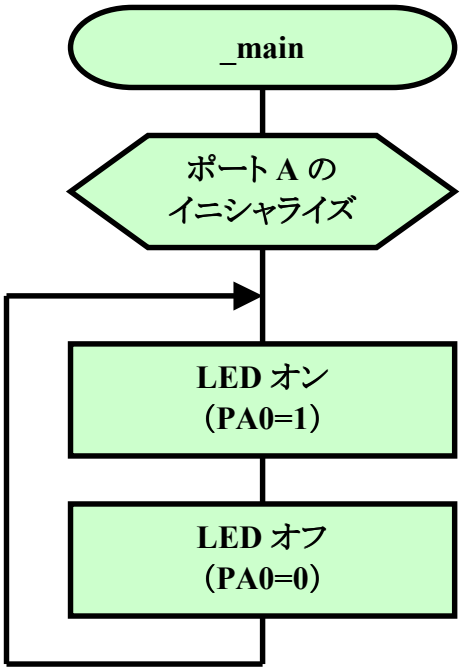
1. プログラムの作成

ここで作るプログラムは TK-3052 の PA0 に接続した LED を点滅させるというものです。その部分の回路図を抜き出すと次のようになります。



■ フローチャートの作成

まずは大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ましょう。できたフローチャートは右のとおりです。LED のオンとオフを繰り返すだけです。



■ コーディング

さて次は，今考えたフローチャートを見ながら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。今回は H8/3052 のアセンブラ命令にコーディングします。コーディングした結果は下のリストになります。

```
_main:
        MOV.B   #H'FF,R0L           ;ポートAのイニシャライズ（出力に設定）
        MOV.B   R0L,@H'FFD1
LOOP:
        BSET    #0,@H'FFD3          ;LEDオン（PA0=1）
        BCLR    #0,@H' FFD3         ;LEDオフ（PA0=0）
        BRA     LOOP
```

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

■ マシン語に変換する

コーディングが終了したものの，このままではマイコン（H8/3052）は何をしたらよいのか理解できません。マイコンが理解できるのはマシン語（機械語）と呼ばれる 16 進の数字の羅列だけです。そこで，次はマシン語への変換作業を行ないます。これをアセンブルと呼びます。マシン語に変換すると次のようになります。これをモニタ上で RAM に入力していきます。（なぜこのようなコードになるか興味のある方は付録をご覧ください。）

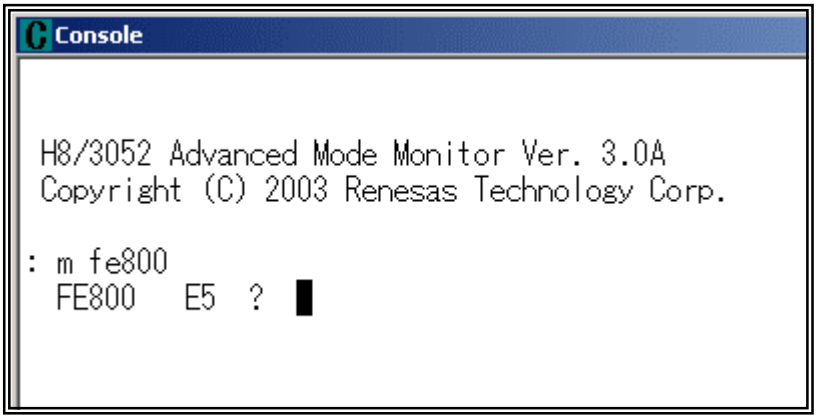
マシン語		ソースリスト			
アドレス	データ	ラベル	ニーモニック	オペランド	コメント
FE800	F8	_main:	MOV.B	#H'FF,R0L	ポートAのイニシャライズ（出力に設定）
FE801	FF				
FE802	38		MOV.B	R0L,@H'FFD1	
FE803	D1				
FE804	7F	LOOP:	BSET	#0,@H'FFD3	LEDオン（PA0=1）
FE805	D3				
FE806	70				
FE807	00				
FE808	7F		BCLR	#0,@H'FFD3	LEDオフ（PA0=0）
FE809	D3				
FE80A	72				
FE80B	00				
FE80C	40		BRA	LOOP	LOOPにジャンプ
FE80D	F6				
FE80E					
FE80F					

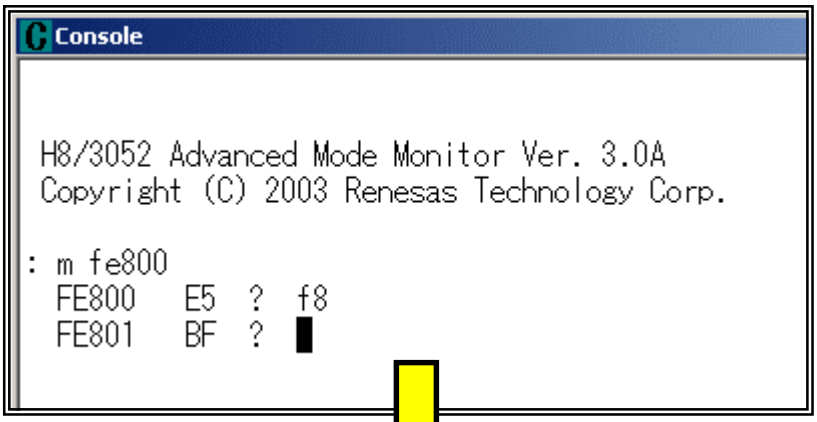
C 言語やアセンブリ言語，あるいは他のどんな言語を使ったとしても，最終的にはマシン語に変換されます。

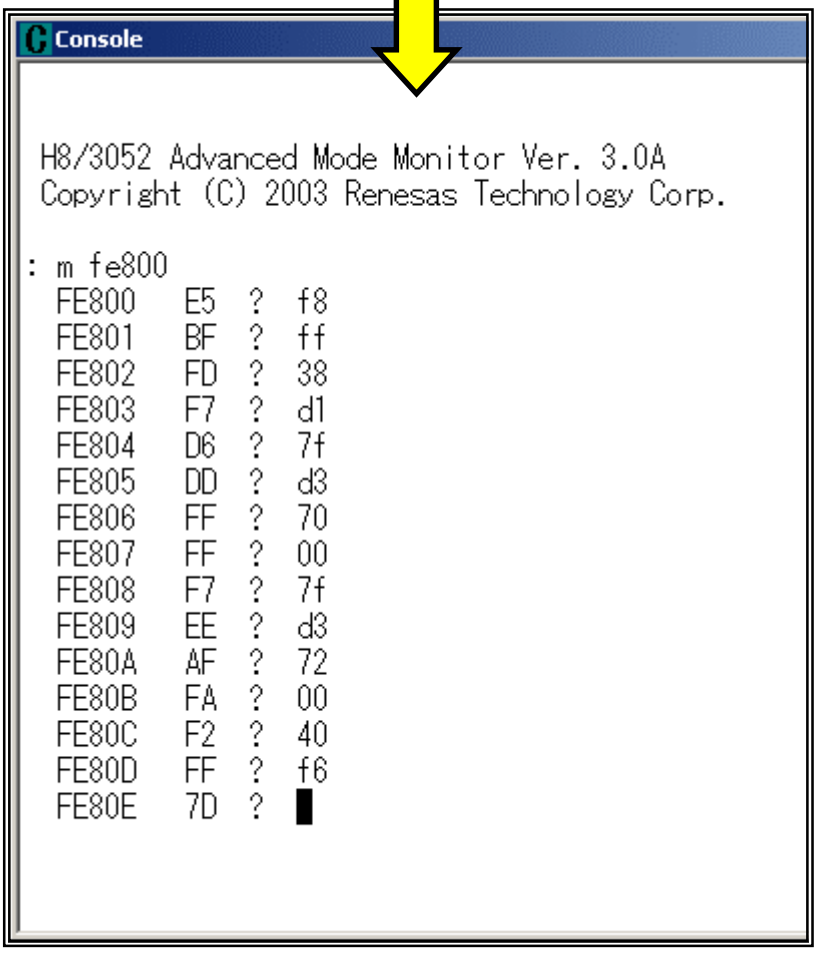
次のページから「Hterm」を利用してプログラムを入力します。しかし，Windows のバージョンによっては「Hterm」が動作しないことがあるようです。そのときは「ハイパーターミナル」や「TeraTerm」のようなターミナルソフト上でモニタプログラムを操作します。ターミナルソフト上でモニタプログラムを利用する方法は巻末の付録をご覧ください。

2. プログラムの入力

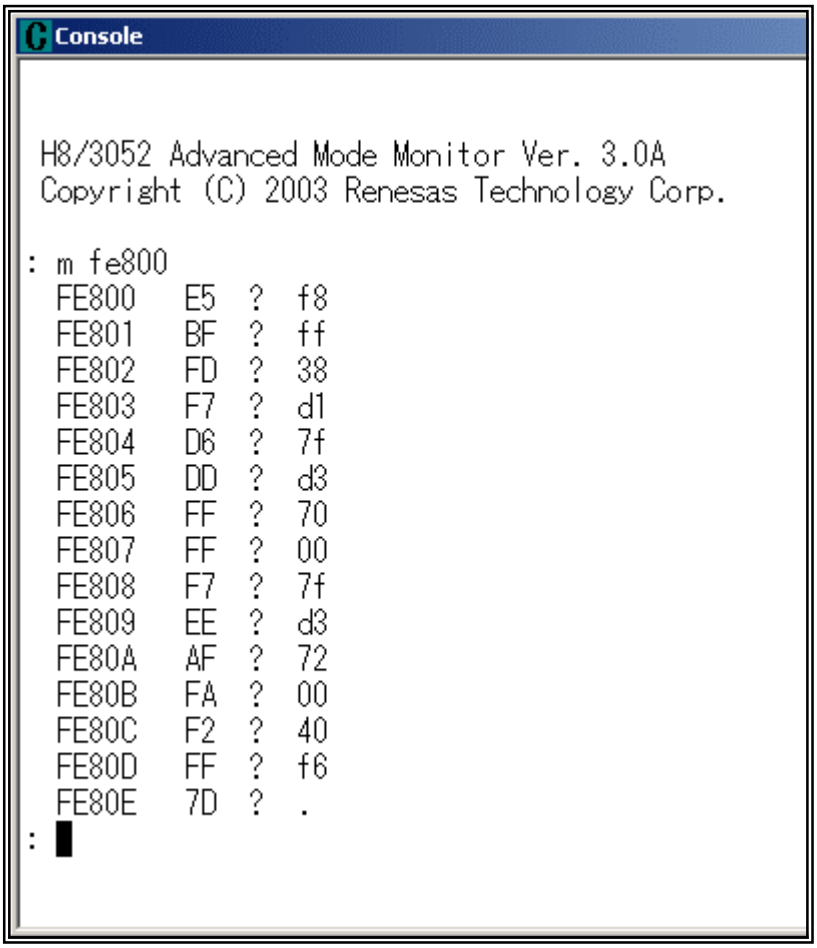
それでは TK-3052 のメモリにマシン語を入力していきましょう。Hterm を起動し，TK-3052 の電源をオンすると，Console ウィンドウが開き，コマンド入力待ちになります。メモリ内容を変更するときは‘M’コマンドを使います。‘FE800’番地から入力しますので，パソコンのキーボードから「m fe800」と入力し「Enter」キーを押します。入力状態になったら 1 バイトずつ順番にデータを入力し「Enter」キーを押します。







全てのデータを入力したらキーボードから「. 」を入力し，「Enter」キーを押してコマンド入力待ちに戻します。



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。プログラムを入力したときはメモリ内容をアセンブラで表示したほうがわかりやすいです。メモリ内容をアセンブル表示することを「逆アセンブル」と呼びます。メモリ内容を逆アセンブル表示するときは‘DA’コマンドを使います。‘FE800’番地から‘FE80D’番地まで入力したので，パソコンのキーボードから「da fe800 fe80d」と入力し「Enter」キーを押します。

```
: da fe800 fe80d
  <ADDR>   <CODE>          <MNEMONIC> <OPERAND>
  FE800    F8FF            MOV.B      #H'FF:8,R0L
  FE802    38D1            MOV.B      R0L,@H'FFFD1:8
  FE804    7FD37000        BSET       #0:3,@H'FFFD3:8
  FE808    7FD37200        BCLR       #0:3,@H'FFFD3:8
  FE80C    40F6            BRA        FE804:8
: 
```

3. プログラムをデバッグする

では，プログラムを実行してみましょう。デバッグの際にはプログラムを一命令ずつ追いかける事がよくあります。これを「トレース実行」とか「シングルステップ実行」とか呼びます。今回はこの機能を使って実行します。

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‘FE800’番地からスタートしますので，プログラムカウンタ（PC）に‘FE800’をセットします。キーボードから「. pc fe800」と入力し「Enter」キーを押します。

```
: da fe800 fe80d
  <ADDR>   <CODE>          <MNEMONIC> <OPERAND>
  FE800    F8FF            MOV.B      #H'FF:8,R0L
  FE802    38D1            MOV.B      R0L,@H'FFFD1:8
  FE804    7FD37000        BSET       #0:3,@H'FFFD3:8
  FE808    7FD37200        BCLR       #0:3,@H'FFFD3:8
  FE80C    40F6            BRA        FE804:8
: .pc fe800
: 
```

シングルステップ実行は‘S’コマンドです。キーボードから「s」と入力し「Enter」キーを押します。すると‘FE800’番地を実行しPC は次の命令のアドレスに進みます。引き続き「Enter」キーを押すと‘S’コマンドが繰り返し入力されたとみなされ，次の命令をシングルステップ実行します。

「Enter」キー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確かめてみましょう。また，LED がちゃんと点滅するかも見てください。

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

Console

```
: .pc fe800
: s
  PC=0FE802  CCR=88:I...N...  SP=00FFFF00
  ER0=000000FF  ER1=00000000  ER2=00000000  ER3=00000000
  ER4=00000000  ER5=00000000  ER6=00000000  ER7=00FFFF00
  FE800   F8FF                  MOV.B     #H'FF:8,R0L
: S
  PC=0FE804  CCR=88:I...N...  SP=00FFFF00
  ER0=000000FF  ER1=00000000  ER2=00000000  ER3=00000000
  ER4=00000000  ER5=00000000  ER6=00000000  ER7=00FFFF00
  FE802   38D1                  MOV.B     R0L,@H'FFFD1:8
: S
  PC=0FE808  CCR=88:I...N...  SP=00FFFF00
  ER0=000000FF  ER1=00000000  ER2=00000000  ER3=00000000
  ER4=00000000  ER5=00000000  ER6=00000000  ER7=00FFFF00
  FE804   7FD37000              BSET      #0:3,@H'FFFD3:8
: S
  PC=0FE80C  CCR=88:I...N...  SP=00FFFF00
  ER0=000000FF  ER1=00000000  ER2=00000000  ER3=00000000
  ER4=00000000  ER5=00000000  ER6=00000000  ER7=00FFFF00
  FE808   7FD37200              BCLR      #0:3,@H'FFFD3:8
: S
  PC=0FE804  CCR=88:I...N...  SP=00FFFF00
  ER0=000000FF  ER1=00000000  ER2=00000000  ER3=00000000
  ER4=00000000  ER5=00000000  ER6=00000000  ER7=00FFFF00
  FE80C   40F6                  BRA       FE804:8
: S
  PC=0FE808  CCR=88:I...N...  SP=00FFFF00
  ER0=000000FF  ER1=00000000  ER2=00000000  ER3=00000000
  ER4=00000000  ER5=00000000  ER6=00000000  ER7=00FFFF00
  FE804   7FD37000              BSET      #0:3,@H'FFFD3:8
: S
  PC=0FE80C  CCR=88:I...N...  SP=00FFFF00
  ER0=000000FF  ER1=00000000  ER2=00000000  ER3=00000000
  ER4=00000000  ER5=00000000  ER6=00000000  ER7=00FFFF00
  FE808   7FD37200              BCLR      #0:3,@H'FFFD3:8
: S
  PC=0FE804  CCR=88:I...N...  SP=00FFFF00
  ER0=000000FF  ER1=00000000  ER2=00000000  ER3=00000000
  ER4=00000000  ER5=00000000  ER6=00000000  ER7=00FFFF00
  FE80C   40F6                  BRA       FE804:8
: S
  PC=0FE808  CCR=88:I...N...  SP=00FFFF00
  ER0=000000FF  ER1=00000000  ER2=00000000  ER3=00000000
  ER4=00000000  ER5=00000000  ER6=00000000  ER7=00FFFF00
  FE804   7FD37000              BSET      #0:3,@H'FFFD3:8
: 
```

第3章 C 言語によるプログラムの作成

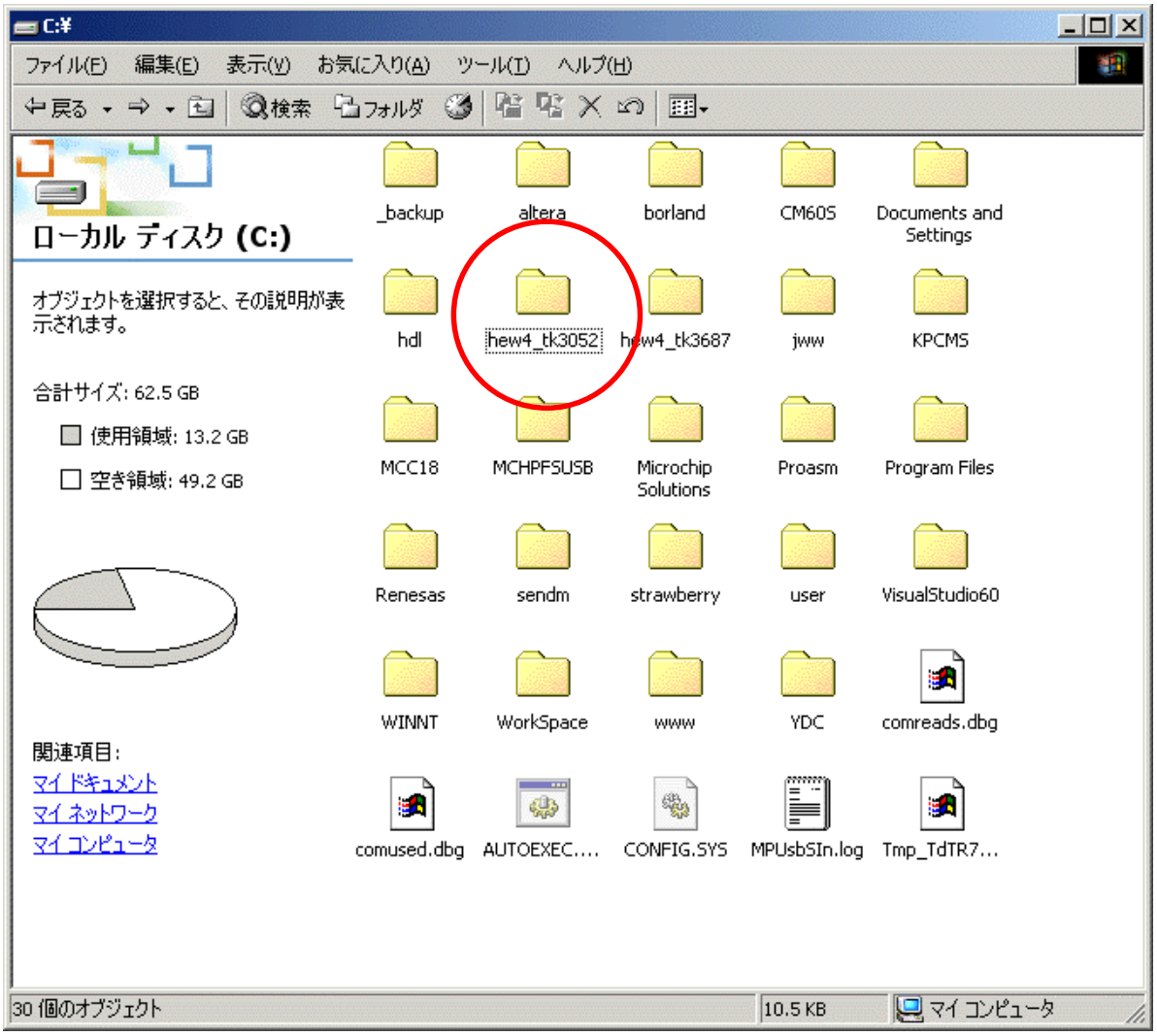
1. プログラムの作成
2. ダウンロードと実行

前の章ではアセンブラ→マシン語でプログラムを作ってみました。ハンドアセンブルは H8 に限らず，どんな CPU にも対応できるので便利(？)ですが，プログラムが長くなってくると，間違いは増え，間違いを修正するのも大変で，何よりアセンブル作業そのものに多くの時間を費やしてしまいます。しかも，アセンブラはマシン語よりわかりやすいとはいえ，人間にとってはまだまだわかりにくい言語です。というわけで，人間にとってもっとわかりやすい言葉で表現し，マシン語への変換という機械的にできる作業はパソコンにまかせてしまいましょう。現在のところ C 言語を使うのがスマートな方法です。

1. プログラムの作成

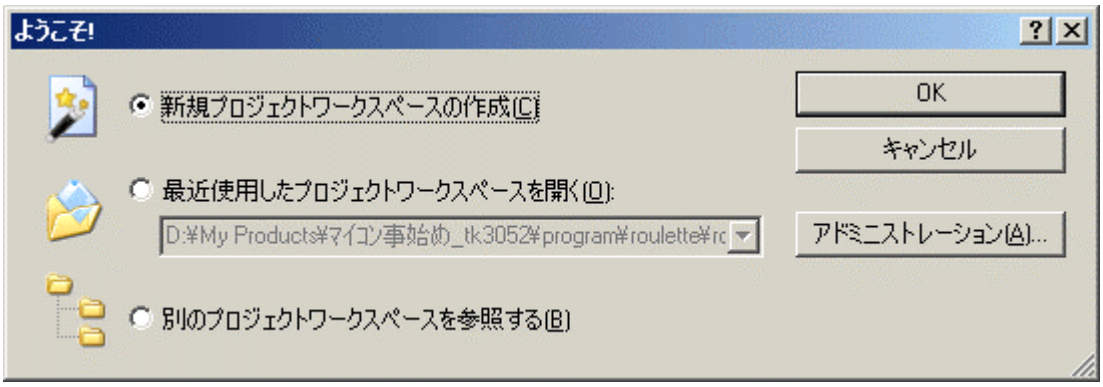
HEW ではプログラム作成作業をプロジェクトと呼び，そのプロジェクトに関連するファイルは 1 つのワークスペース内にまとめて管理されます。通常はワークスペース，プロジェクト，メインプログラムには共通の名前がつけられます。この章で作るプロジェクトは‘IoPort_led_c’と名付けます。以下に，新規プロジェクト‘IoPort_led_c’を作成する手順と動作確認の手順を説明します。

しかしその前に，HEW 専用作業フォルダを作っておきましょう。C ドライブに‘Hew4_tk3052’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。



■ **プロジェクトの作成**

HEW を起動すると，次のようなダイアログが表示されます。「新規プロジェクトワークスペースの作成」を選択し「OK」をクリックします。

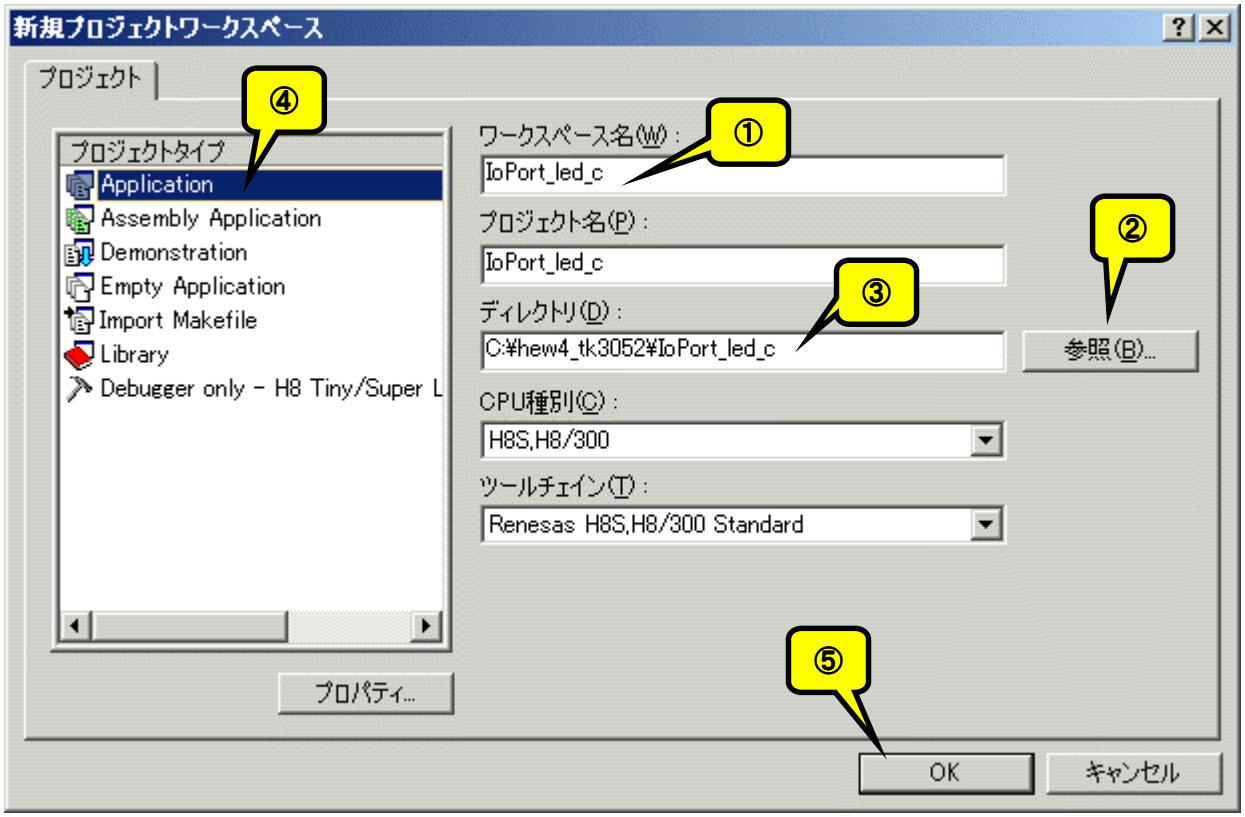


まず，①「ワークスペース名（W）」（ここでは‘IoPort_led_c’）を入力します。「プロジェクト名（P）」は自動的に同じ名前になります。

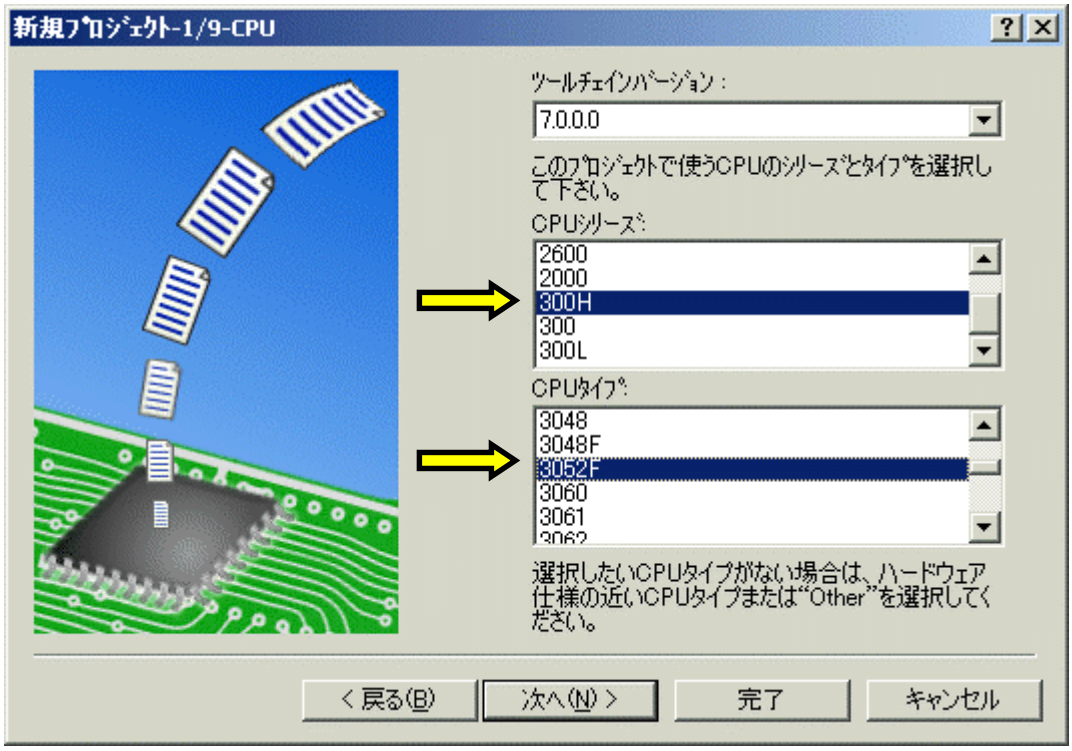
ワークスペースの場所を指定します。②右の「参照（B）…」ボタンをクリックします。そして，あらかじめ用意した HEW 専用作業フォルダ（ここでは Hew4_tk3052）を指定します。設定後，「ディレクトリ（D）」が正しいか確認して下さい。（③）

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。

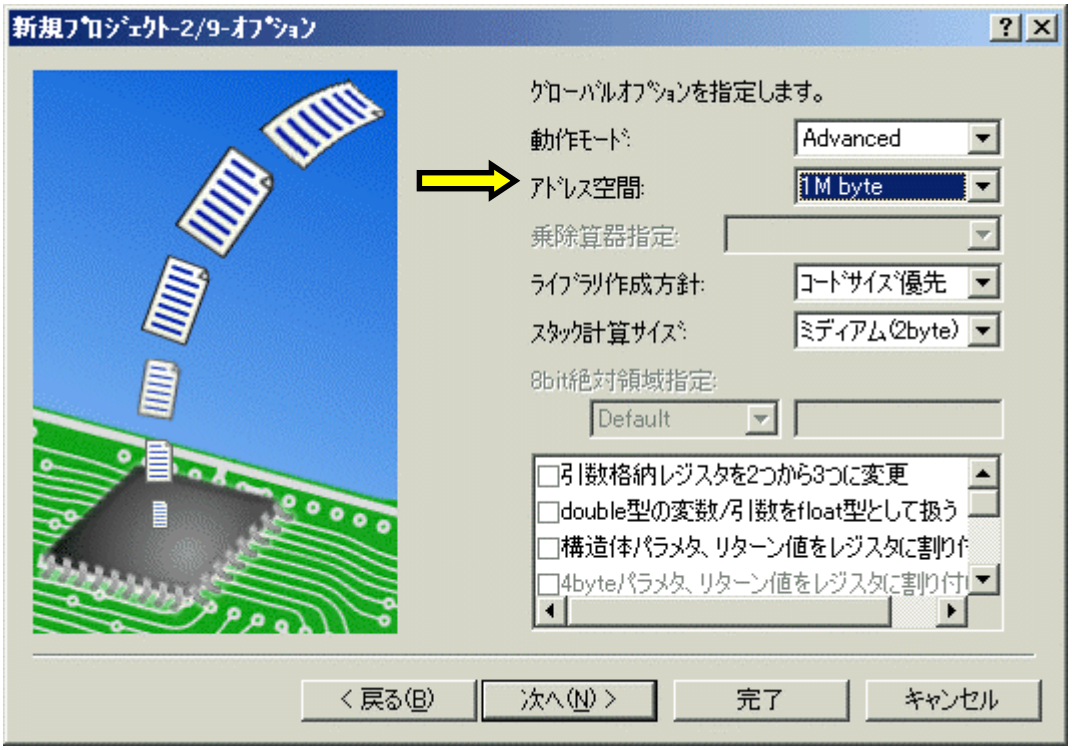
入力が終わったら⑤「OK」をクリックして下さい。



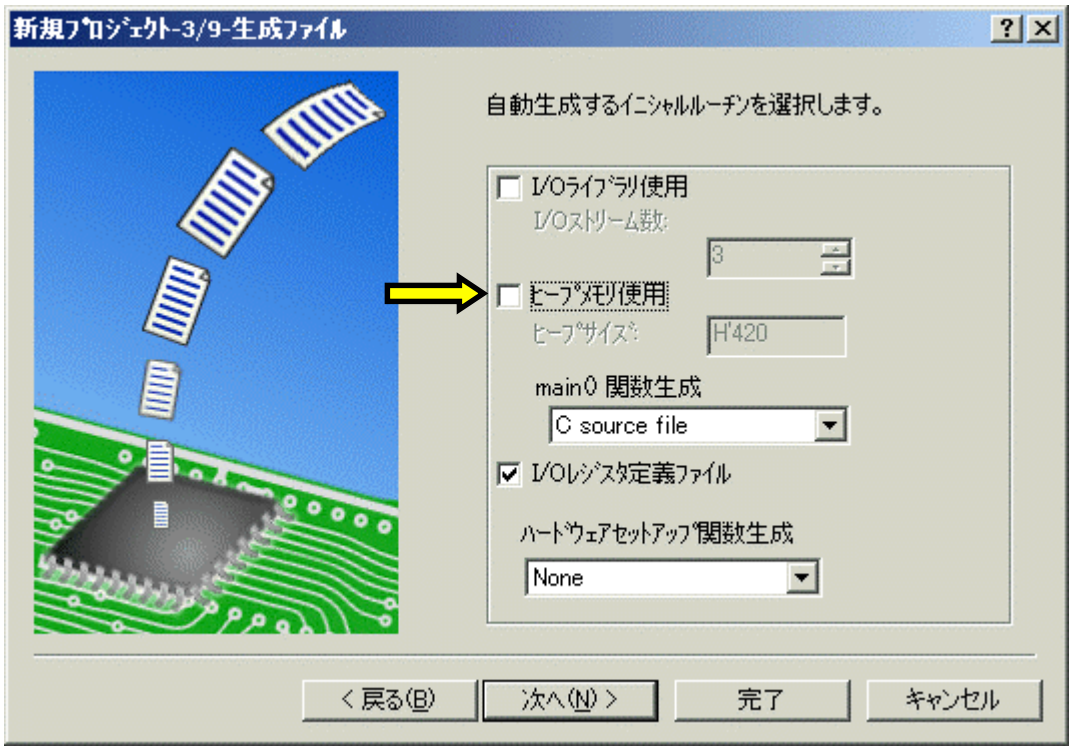
「新規プロジェクトー1／9ーCPU」で，使用する CPU シリーズ(300H)と，CPU タイプ(3052F)を設定し，「次へ(N) >」をクリックします。



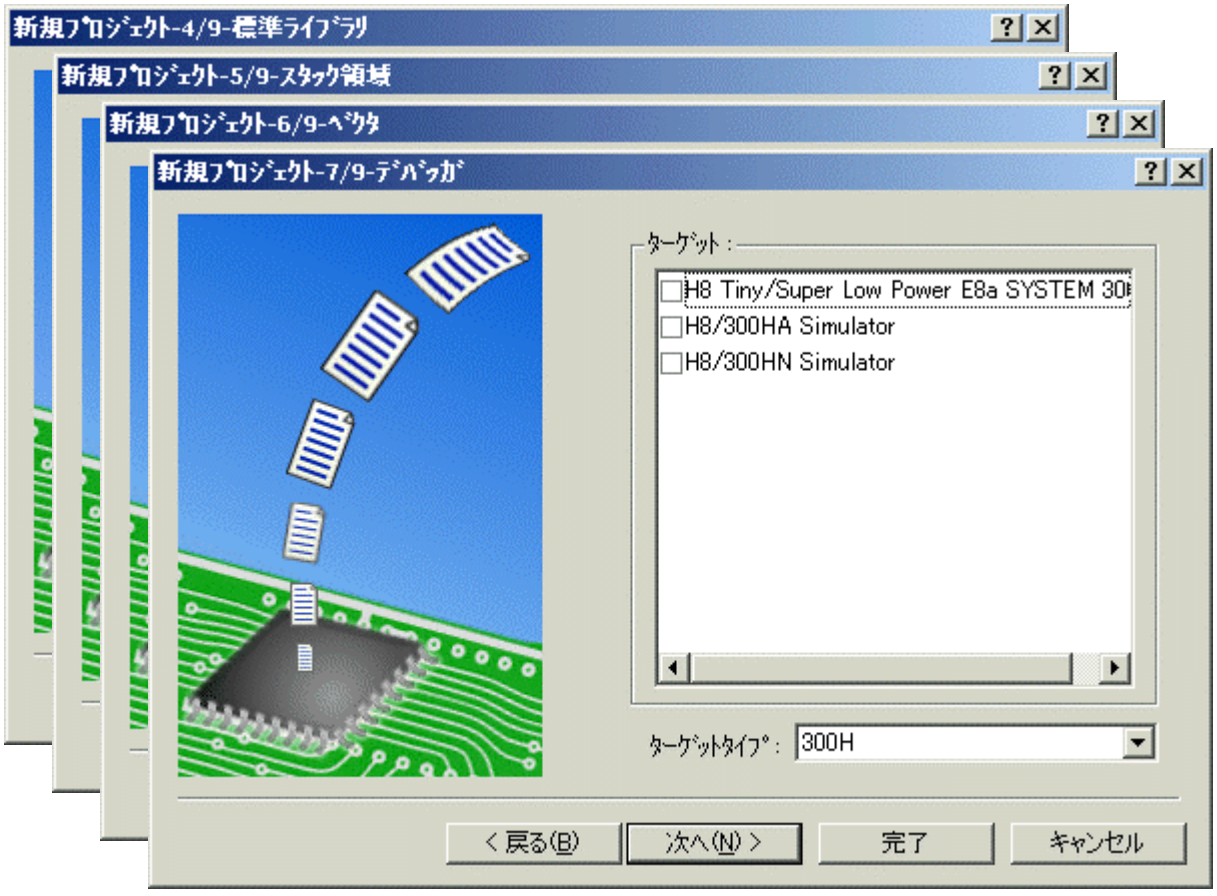
「新規プロジェクトー2／9ーオプション」で，アドレス空間を「1Mbyte」に設定し，「次へ(N) >」をクリックします。



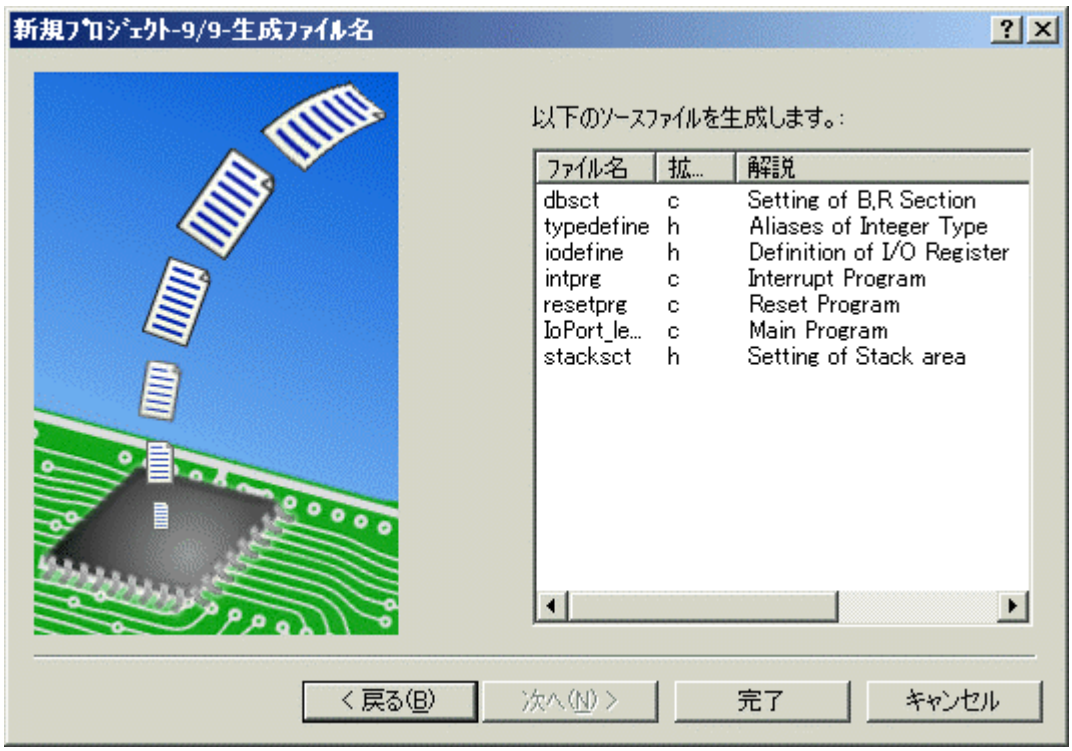
「新規プロジェクトー3／9ー生成ファイル」で，「ヒープメモリ使用」のチェックを外し，「次へ(N) >」をクリックします。(ヒープ領域を使う場合はチェックを入れたままにします。このマニュアルではヒープ領域は使いません。)



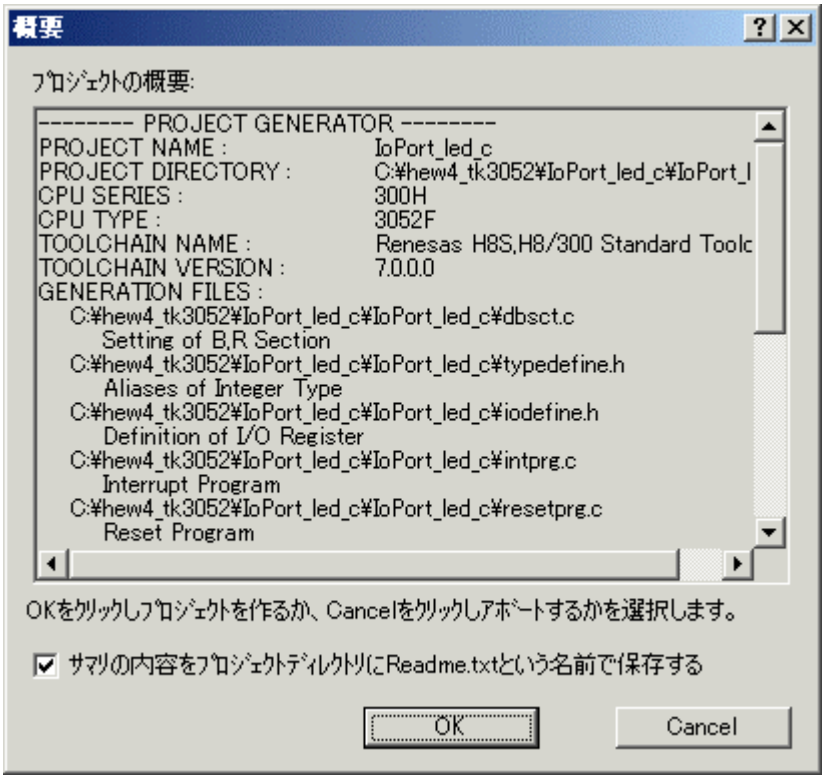
「新規プロジェクトー4／9ー標準ライブラリ」，「新規プロジェクトー5／9ースタック領域」，「新規プロジェクトー6／9ーベクタ」，「新規プロジェクトー7／9ーデバッガ」は変更しません。「次へ(N) >」をクリックして次の画面に進みます。



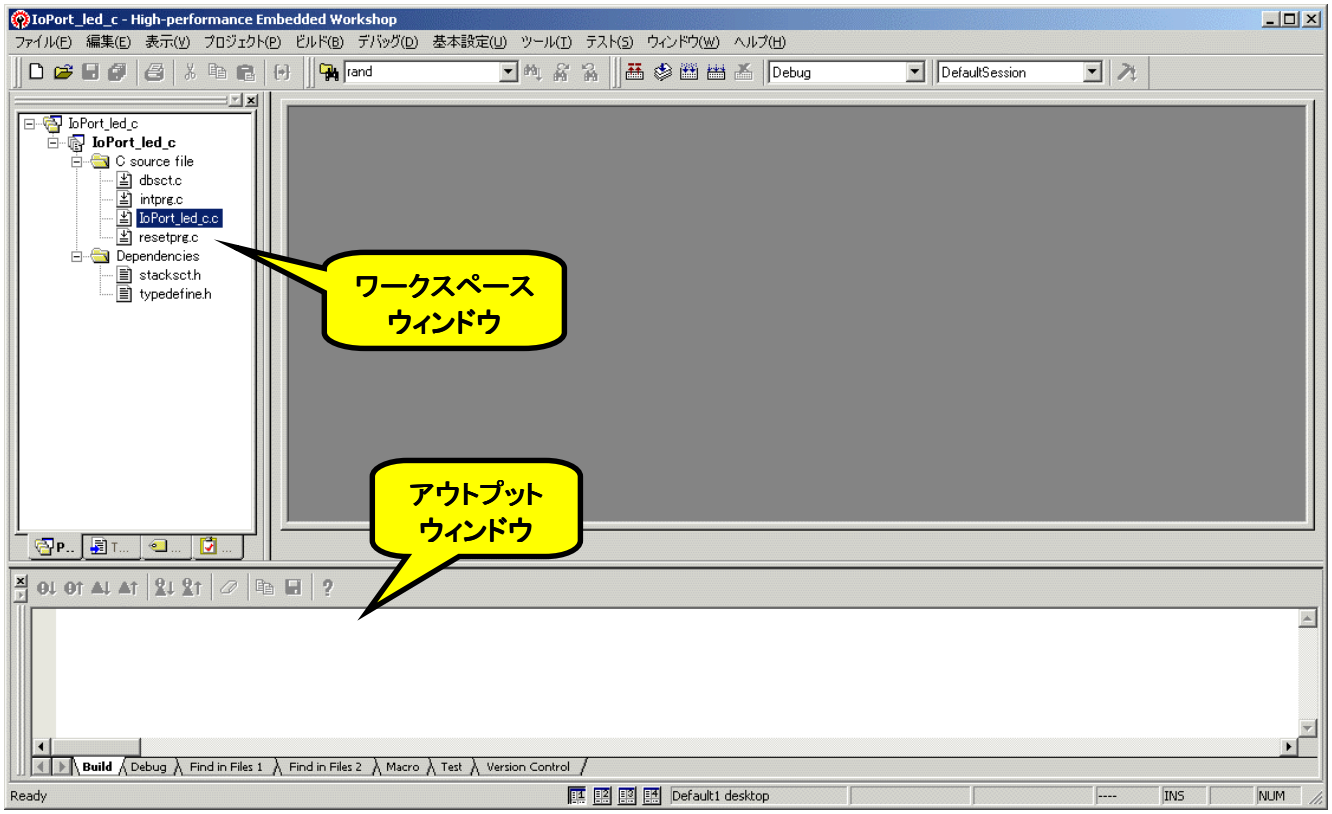
次は「新規プロジェクトー9／9ー生成ファイル名」です。「完了」をクリックします。



すると，「概要」が表示されるので「OK」をクリックします。

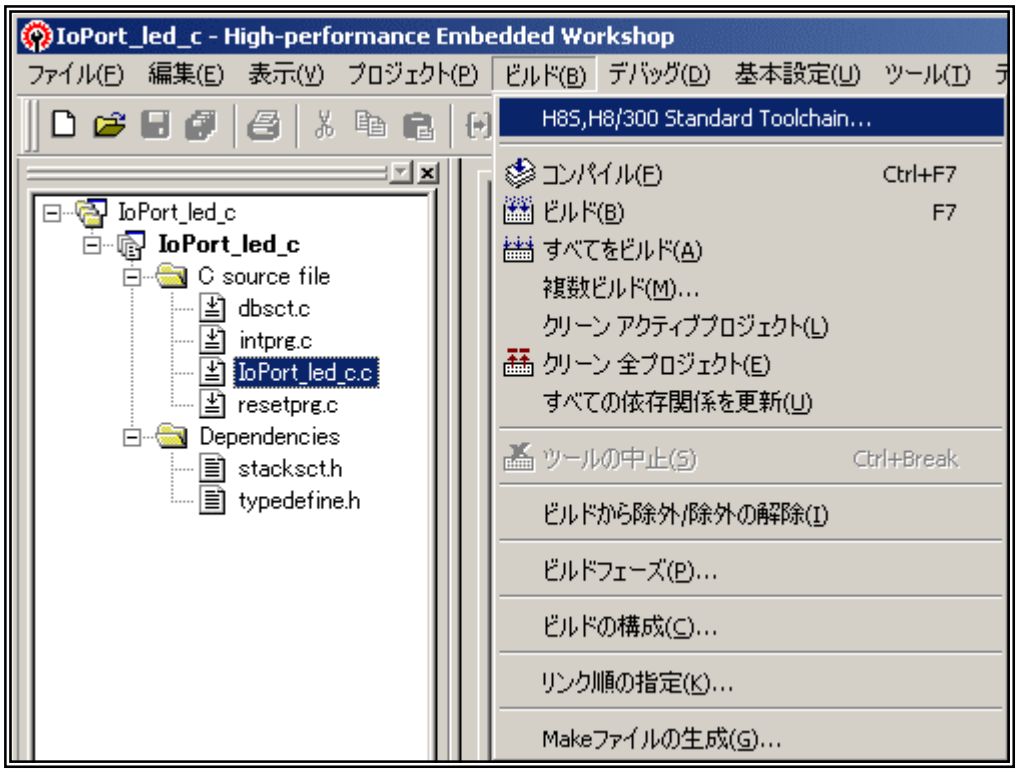


これで，プロジェクトワークスペースが完成します。HEWはプロジェクトに必要なファイルを自動生成し，それらのファイルは左端のワークスペースウィンドウに一覧表示されます。

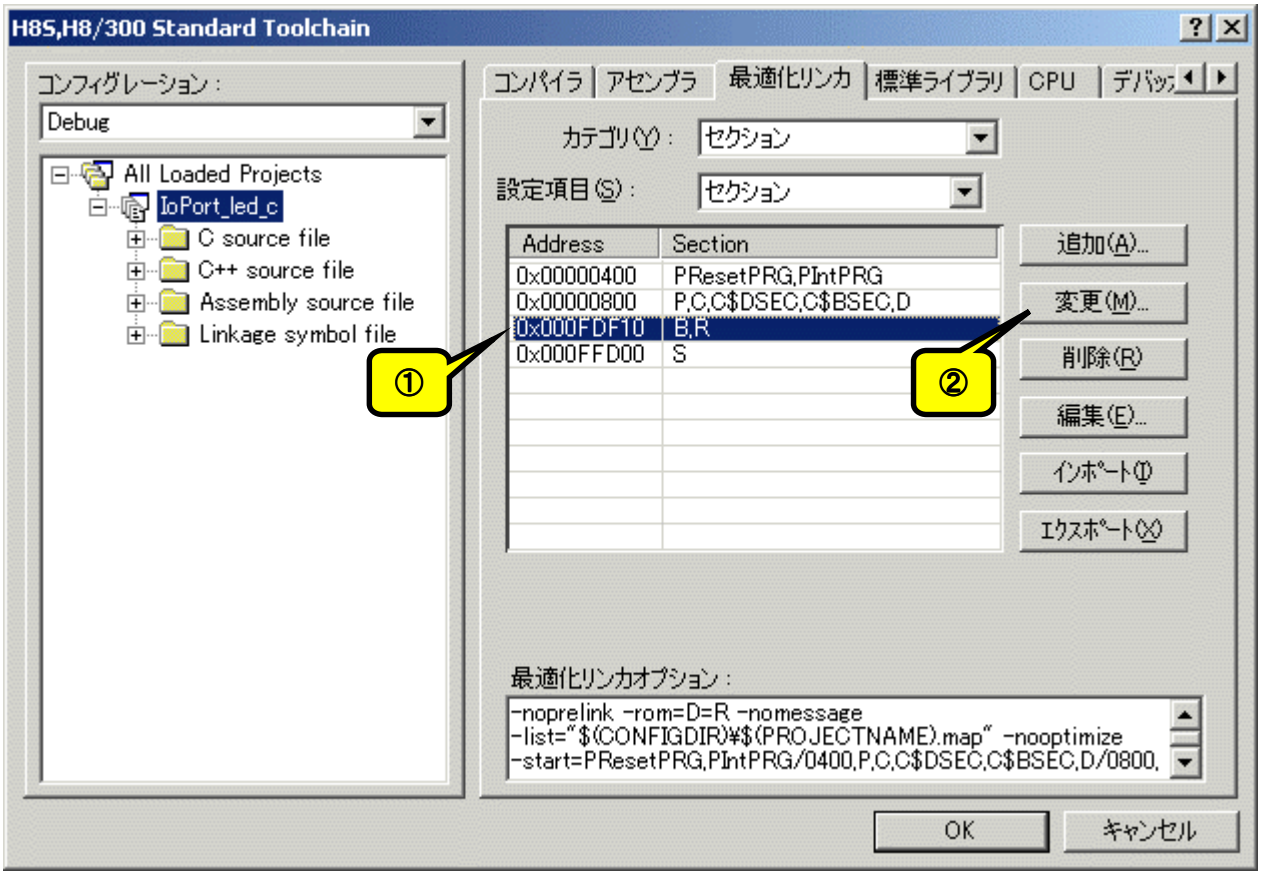


さて，これでプロジェクトは完成したのですが，Htermを使うときはセクションを変更してプログラムがRAM上にできるようにします。（当然ながら，通常はROM上で動かすので，基本的に変更する必要はなし。）

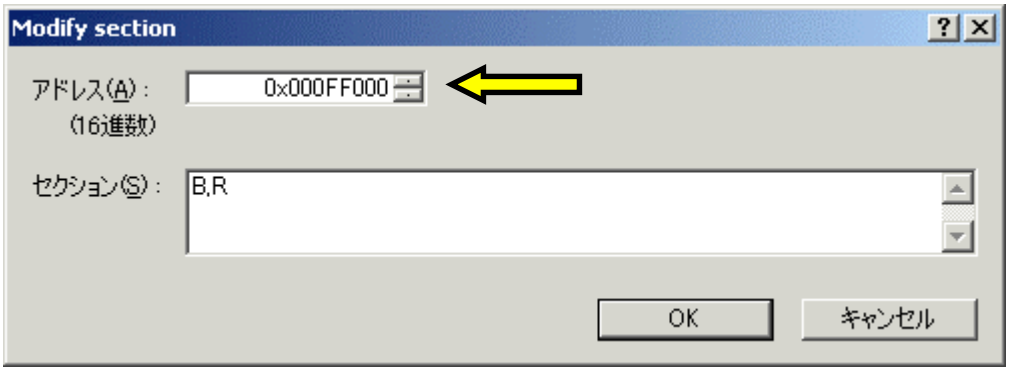
下図のように，メニューバーから「H8S,H8/300 Standard Toolchain…」を選びます。



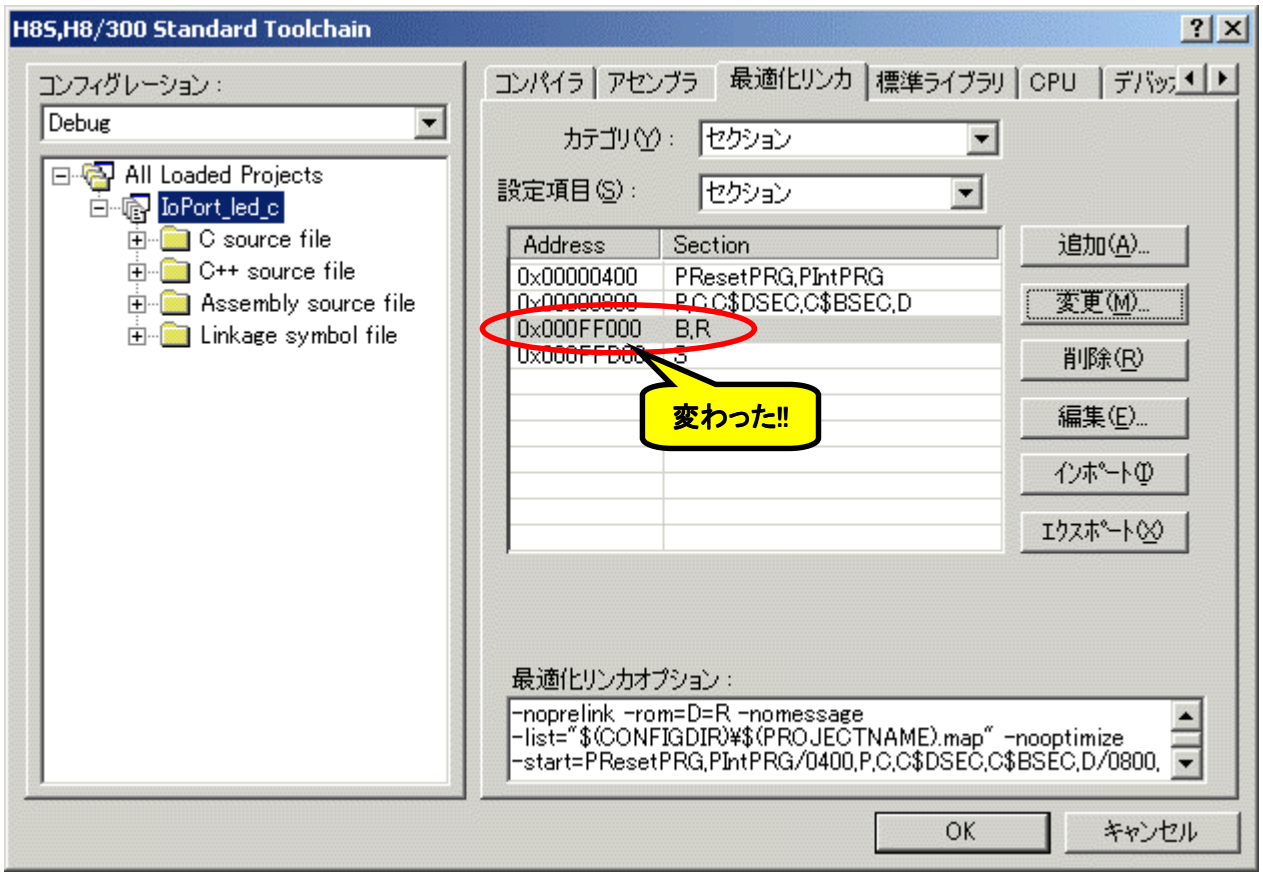
すると，「H8S，H8/300 Standard Toolchain」ウィンドウが開きます。「最適化リンカ」のタブを選び，「カテゴリ(Y)」のドロップダウンメニューの中から「セクション」を選択します。すると，下図のような各セクションの先頭アドレスを設定する画面になります。第 1 章で考えたように，ユーザが作成するプログラムは FE100～FFCFF 番地の範囲に，プログラムとワークエリアが入るようにしなければなりません。まず「B,R」セクションを変更してみましょう。①「B,R」セクションをクリックしてから，②「変更(M)」ボタンをクリックしてください。



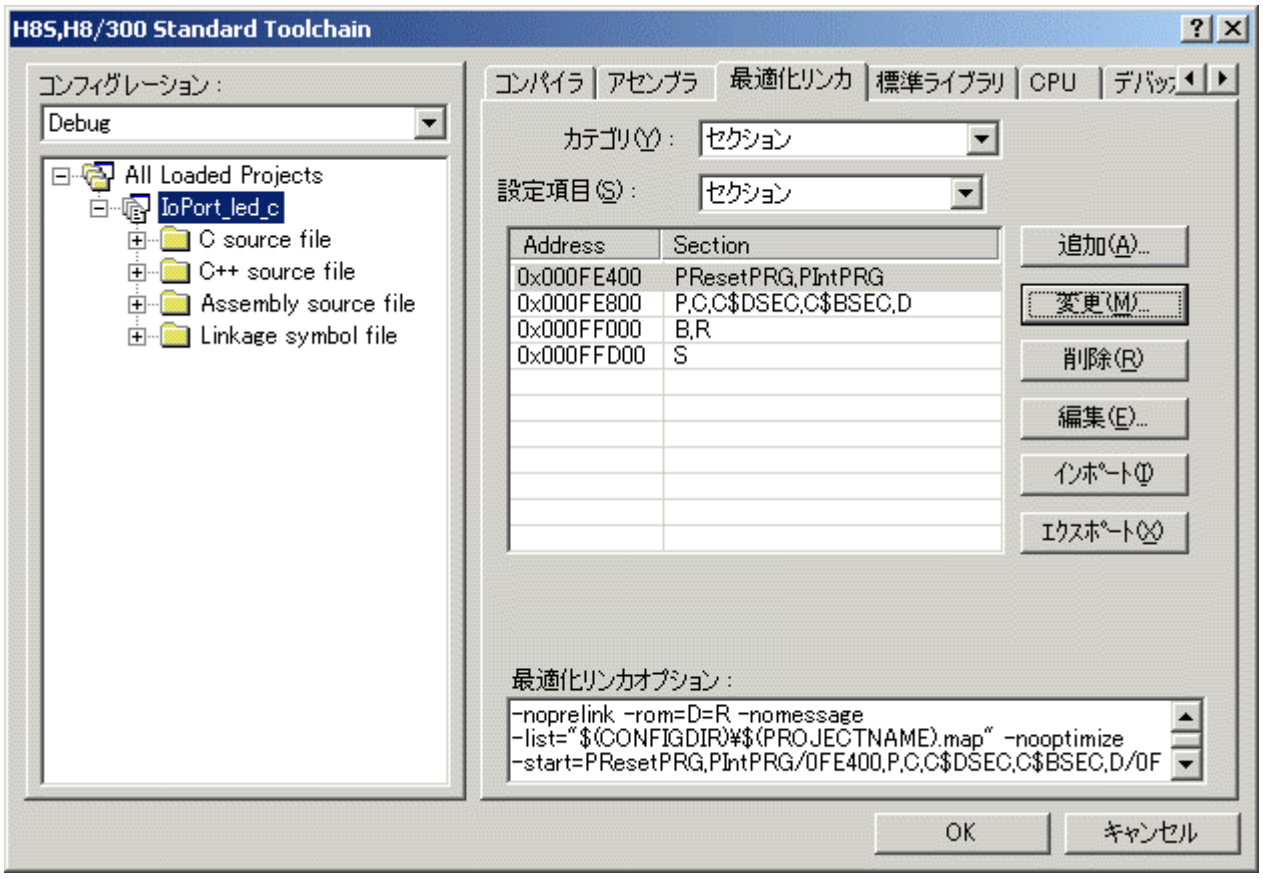
すると，次のようなダイアログが表示されます。「B,R」セクションは FF000 番地から割り当てます。それで，「アドレス」に入力して「OK」をクリックします。



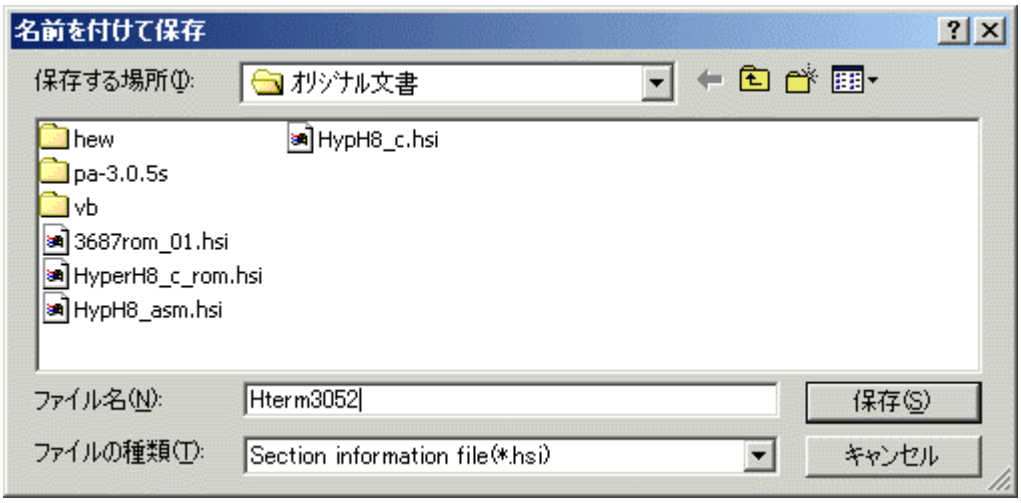
すると・・・



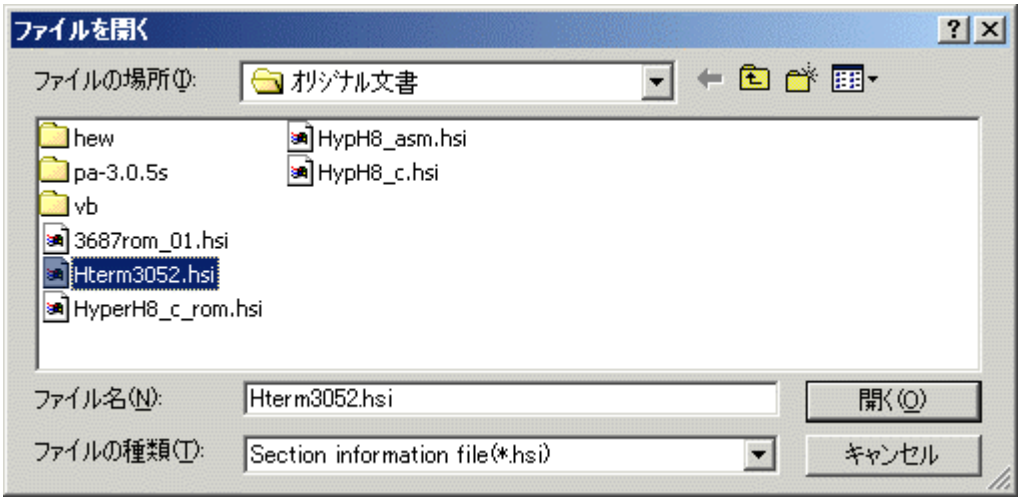
同じように他のセクションも下図のように変更します。



さて，「Hterm」を使うときのセクション設定は，今後もよく使うので保存しておきましょう。「エクスポート(X)」をクリックしてください。「名前を付けて保存」ダイアログが開きます。適当な名前（今回は「Hterm3052」にしました）をつけて保存します。



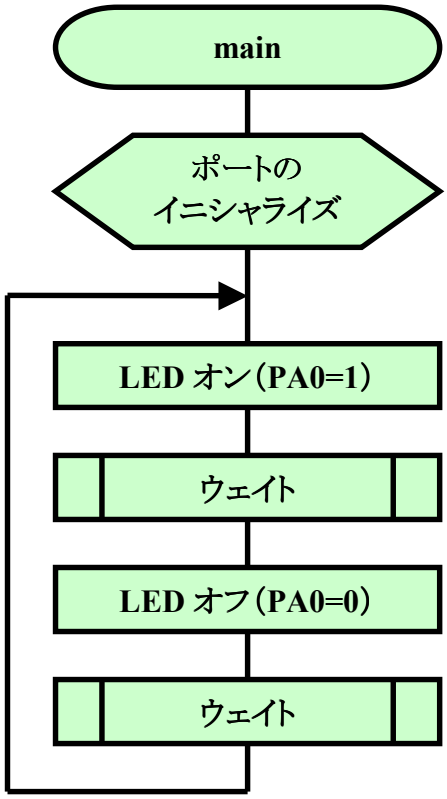
ちなみに，次回以降，保存した設定を使うときは「インポート(I)」をクリックします。すると，「ファイルを開く」ダイアログが開きます。保存した設定を選択して「開く」をクリックするとセクションが設定されます。



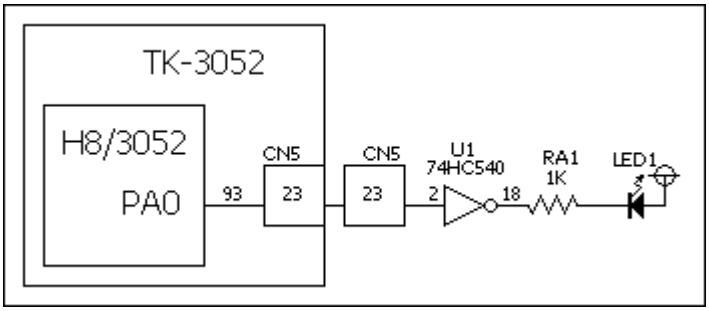
これで，セクションの指定は完了です。「OK」をクリックして「H8S，H8/300 Standard Toolchain」ウィンドウを閉じます。

■ ソースリストの入力

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



回路図は前の章と同じです。下記に再掲します。



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

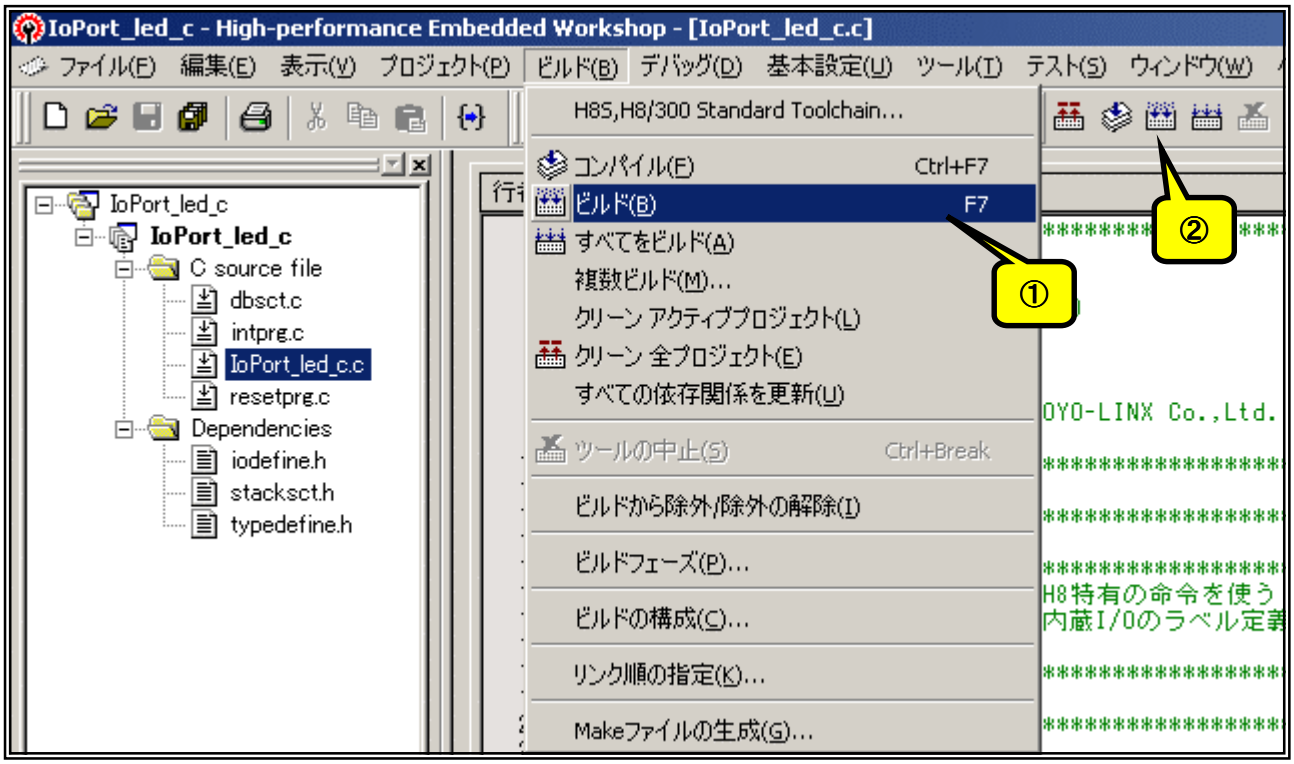
```
行番… S… ソース
   1     /************************************************************************/
   2     /*                                                                      */
   3     /*  FILE        :IoPort_led_c.c                                         */
   4     /*  DATE        :Thu, Jun 24, 2010                                      */
   5     /*  DESCRIPTION :Main Program                                           */
   6     /*  CPU TYPE    :H8/3052F                                               */
   7     /*                                                                      */
   8     /*  This file is generated by Renesas Project Generator (Ver.4.8).     */
   9     /*                                                                      */
  10     /************************************************************************/
  11
  12
  13
  14     #ifdef __cplusplus
  15     extern "C" {
  16     void abort(void);
  17     #endif
  18     void main(void);
  19     #ifdef __cplusplus
  20     }
  21     #endif
  22
  23     void main(void)
  24     {
  25
  26     }
  27
  28     #ifdef __cplusplus
  29     void abort(void)
  30     {
  31
  32     }
  33     #endif
  34
```

このファイルに追加・修正していきます。下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。

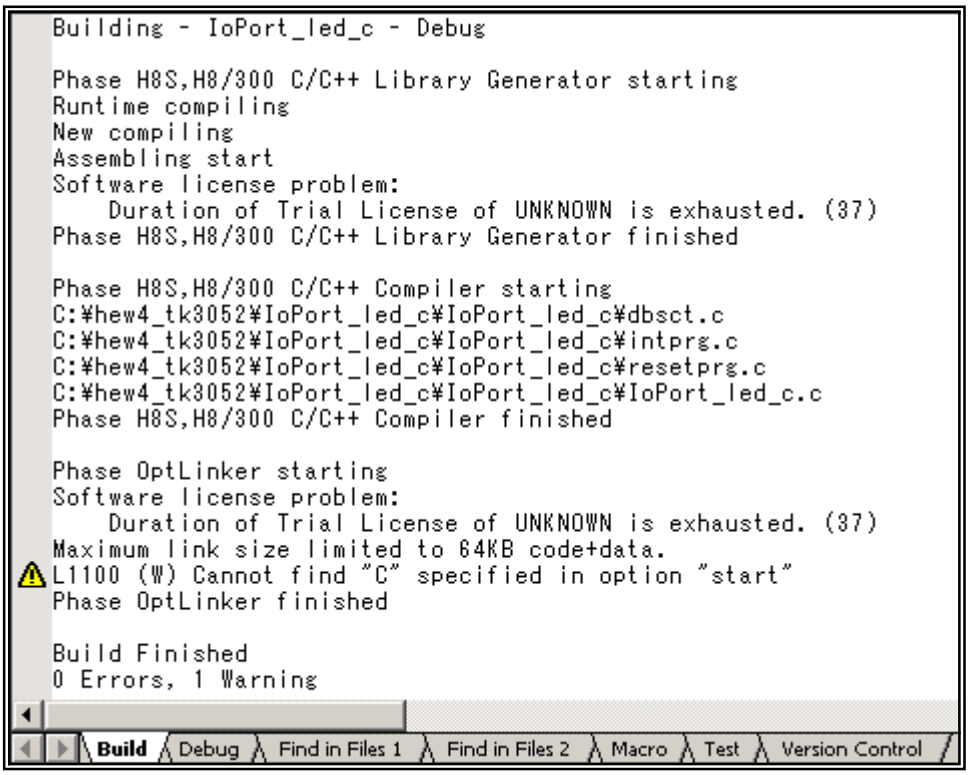
```
/************************************************************************/
/*                                                                      */
/*  FILE        :IoPort_led_c.c                                         */
/*  DATE        :Wed, Jun 02, 2010                                      */
/*  DESCRIPTION :Main Program                                           */
/*  CPU TYPE    :H8/3052F                                               */
/*                                                                      */
/*  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)
{
    PA.DDR      =   0xff;   // ポートAを出力に設定
    PA.DR.BYTE  =   0x00;   // ポートA出力クリア

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

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

    for (i=0;i<2083333;i++){}
}

```

■ 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(B)」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。まちがいなく入力しているか確認して下さい。

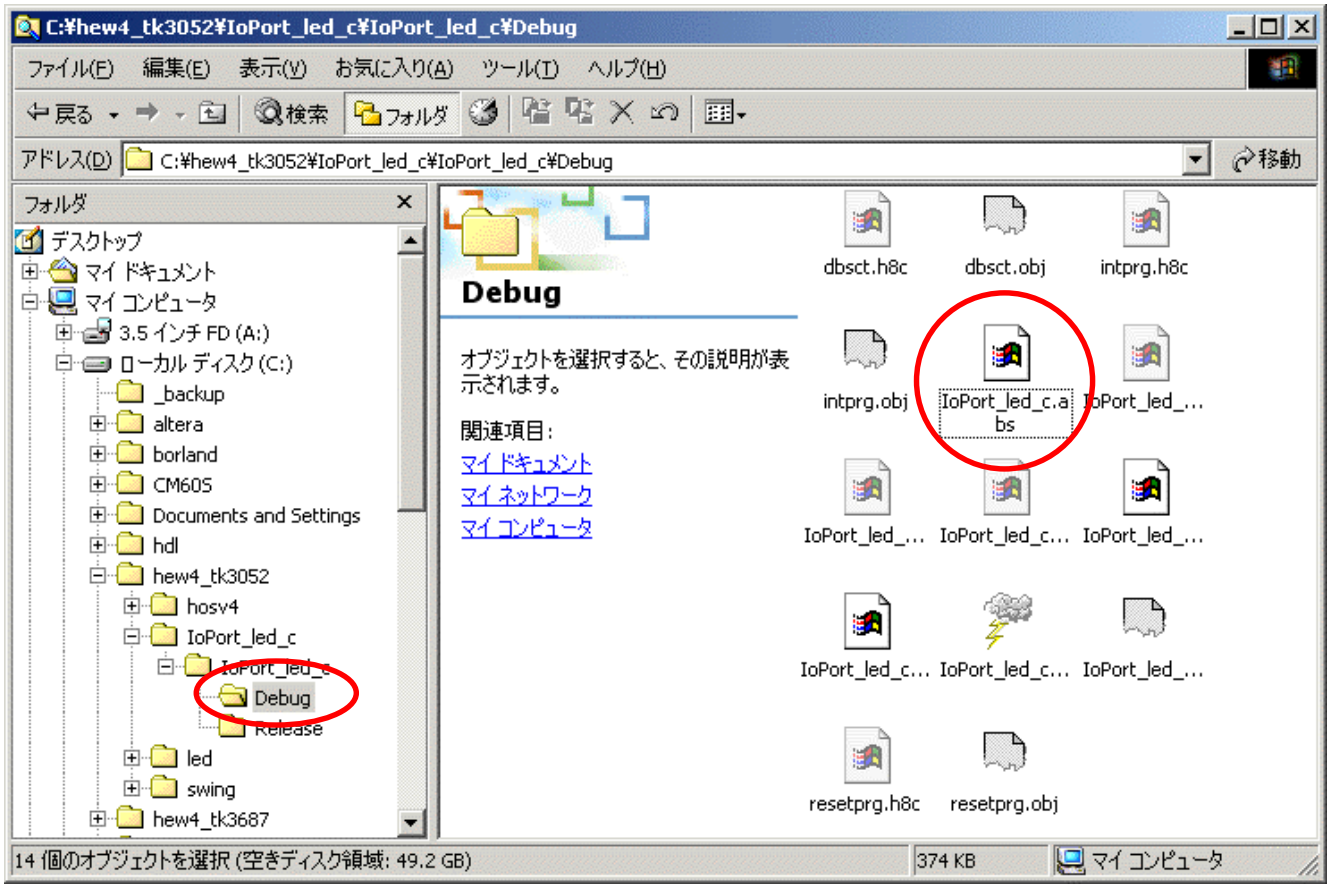
図では「1 Warning」と表示されています。これは「まちがいではないかもしれないけど，念のため確認してね」という警告表示です。例えばこの図の「L1100(W) Cannot find "C" specifind in option "start"」は，C セクションを設定したのに C セクションのデータがないとき表示されます。今回のプログラムでは C セクションは使っていませんので，この警告が出ても何も問題ありません。

もっとも，Warninng の中には動作に影響を与えるものもあります。「H8S, H8/300 シリーズ C/C++コンパイラ，アセンブラ，最適化リンケージエディタ ユーザーズマニュアル」の 741 ページからコンパイラの，853 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。

2. ダウンロードと実行

これから「Hterm」を利用してプログラムをダウンロード・実行します。しかし，Windows のバージョンによっては「Hterm」が動作しないことがあるようです。そのときは「ハイパーターミナル」や「TeraTerm」のようなターミナルソフト上でモニタプログラムを操作します。ターミナルソフト上でモニタプログラムを利用する方法は巻末の付録をご覧ください。

ビルドが成功すると C の命令がマシン語に変換され「IoPort_led_c. abs」というファイルが作られます。拡張子が「. abs」のファイルは「アブソリュートロードモジュールファイル」もしくは「ELF/DWARF ファイル」と呼ばれていて，マシン語の情報とデバッガ用の情報が含まれているファイルです。このファイルはプロジェクトフォルダの中の「Debug」フォルダ内に作られます。

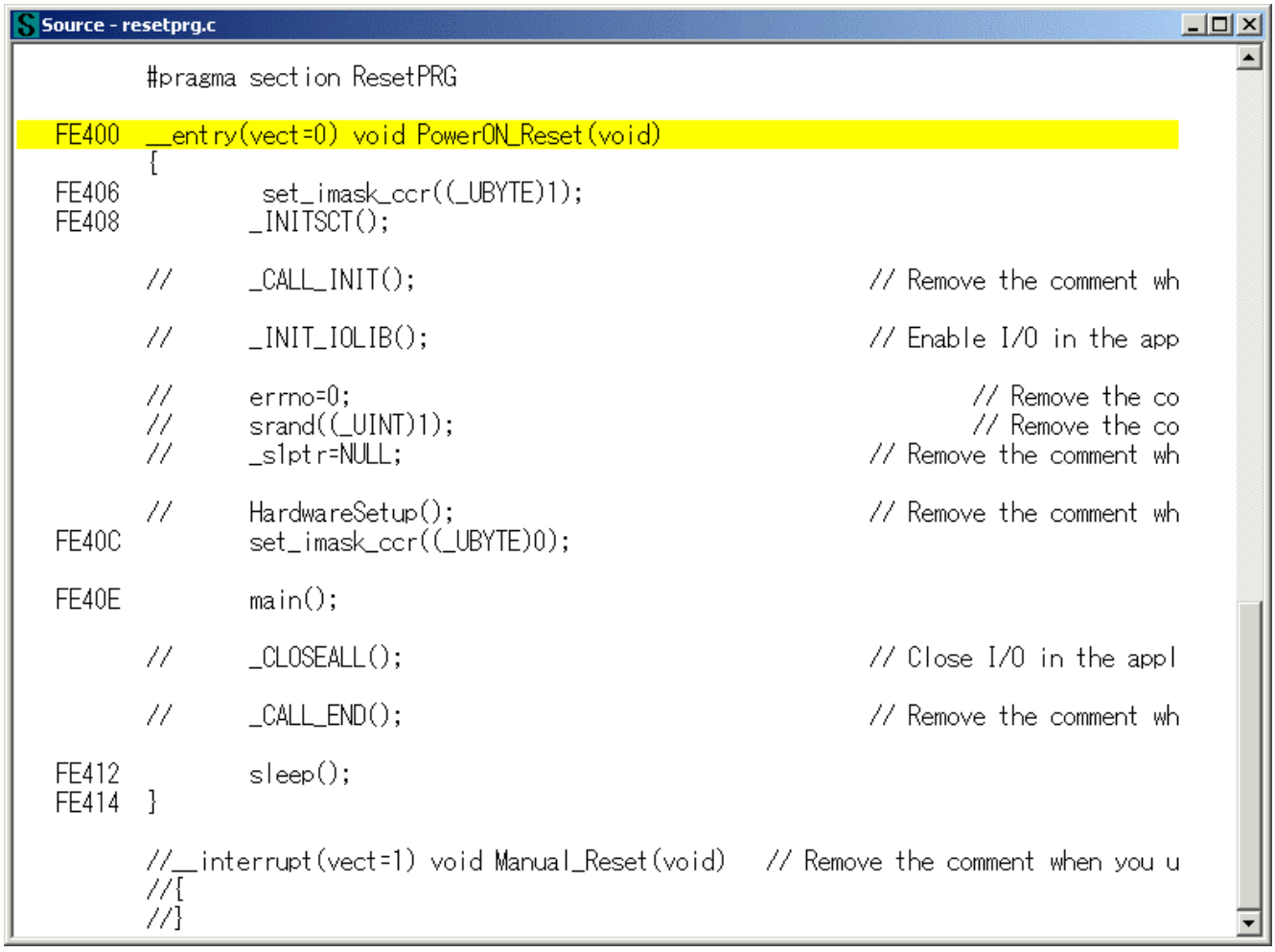


それではダウンロードしましょう。「Hterm」を起動してください。ファンクションキーの［F9］を押すか，メニューバーから「Load（L）」を選んでください。

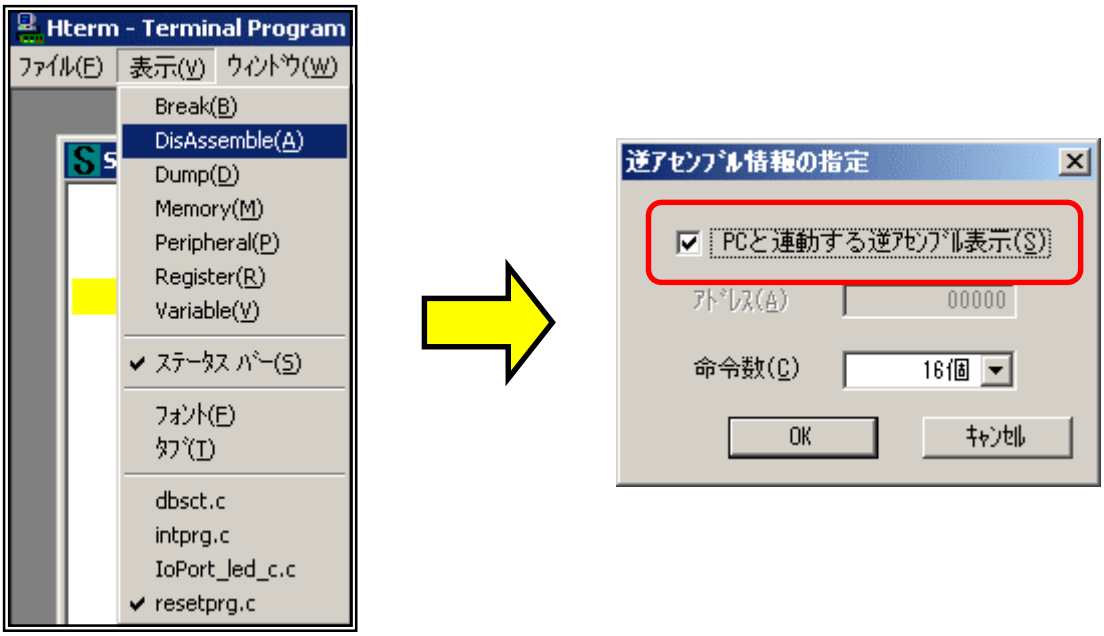
すると，「ユーザプログラムのダウンロード」ダイアログが開きます。「IoPort_led_c．abs」を選択します。

ダウンロードが終わると(今回はプログラムが短いのであっという間です)ソースプログラムを表示するか尋ねられます。「はい(Y)」をクリックします。

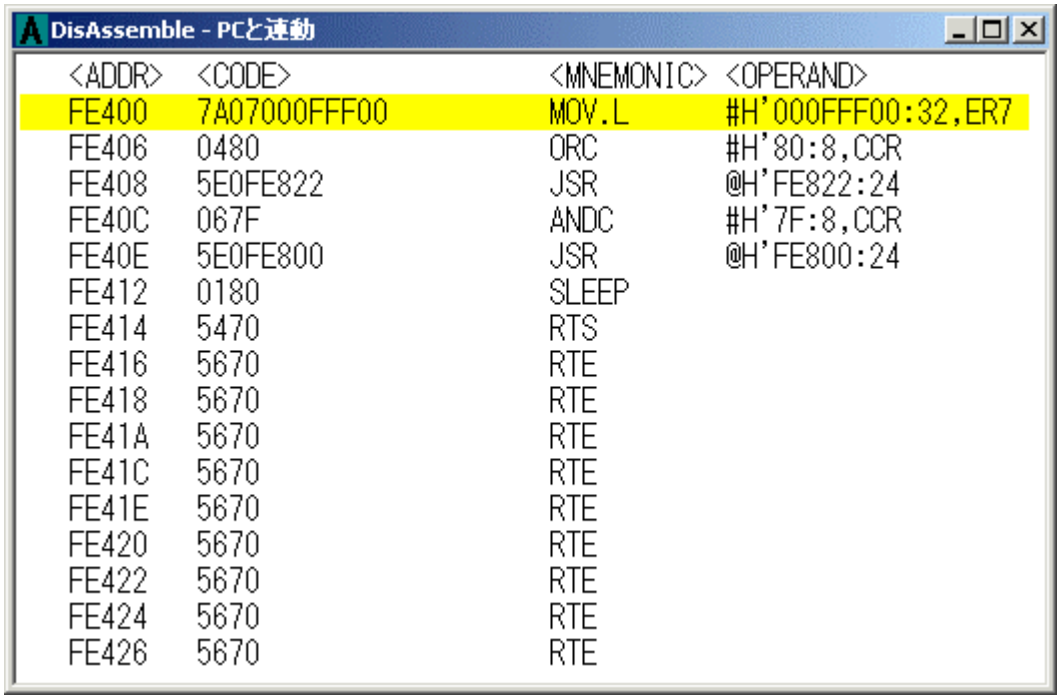
すると，ソースプログラムが表示されます(なお，ダウンロードするとプログラムの先頭アドレスがプログラムカウンタ(PC)にセットされます)。まずはプログラムの先頭の「resetprg．c」が表示されます。また，現在の PC のある行は黄色でマークされます。



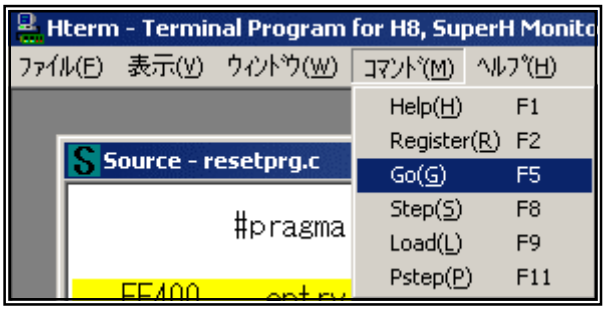
さて，デバッグのためには C 言語のソースファイルがどのようなマシン語に変換されたのか表示しておいたほうが便利です。メニューバーから「DisAssemble（A）」を選んでください。すると，「逆アセンブル情報の指定」ダイアログが開きます。「PC と連動する逆アセンブル表示（S）」にチェックを入れます。



すると，逆アセンブルウィンドウが開きます。こちらも現在の PC のある行は黄色でマークされます。

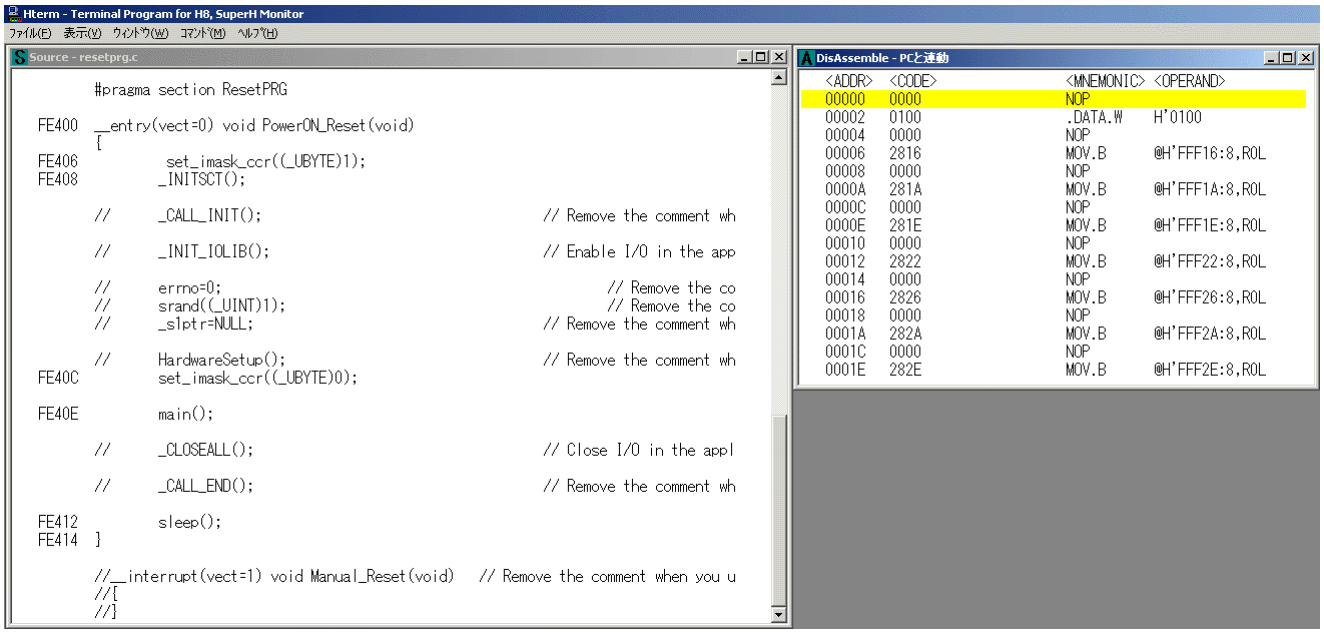


それでは実行しましょう。ファンクションキーの［F5］を押すか，メニューバーから「Go（G）」を選んでください。



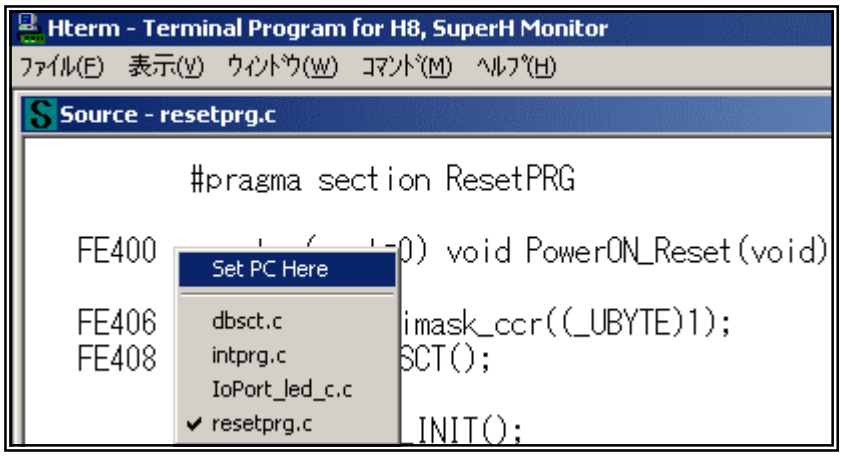
いかがでしょうか。ちゃんと LED は点滅しましたか。うまく動作しないときはプログラムの入力ミスの可能性が大です。もう一度ちゃんと入力しているか確認してみてください。

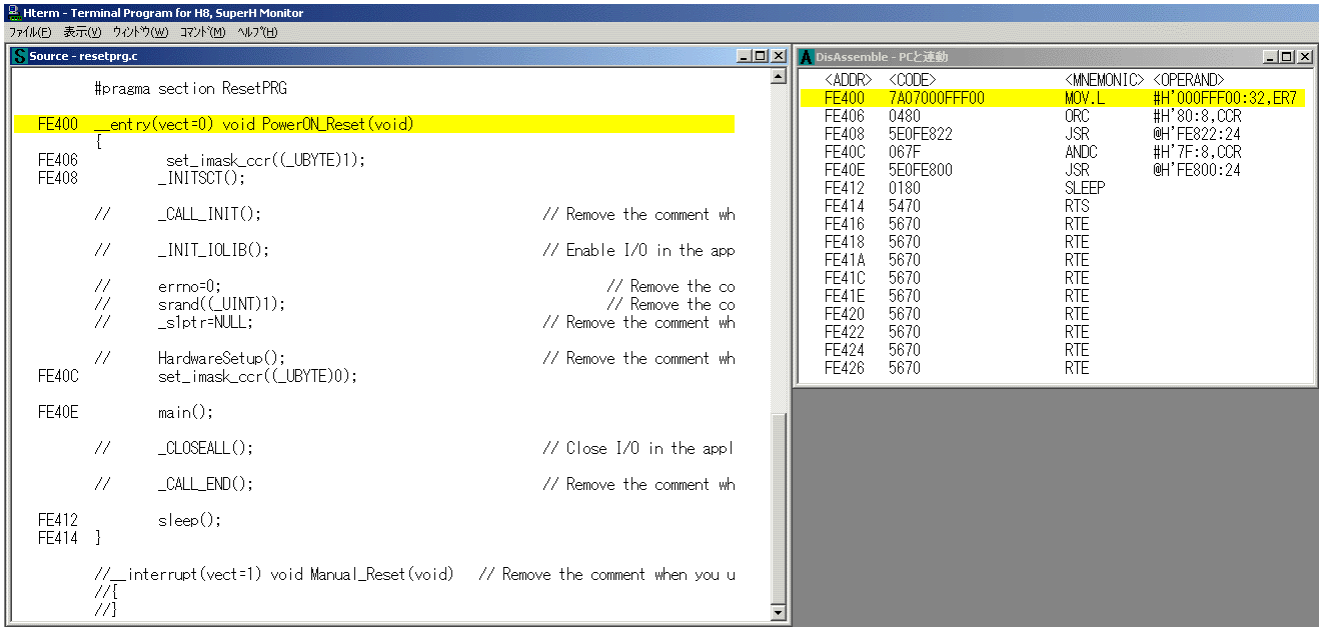
さて，実行中のプログラムをストップするときは TK-3052 のリセットスイッチ（SW1）を押します。



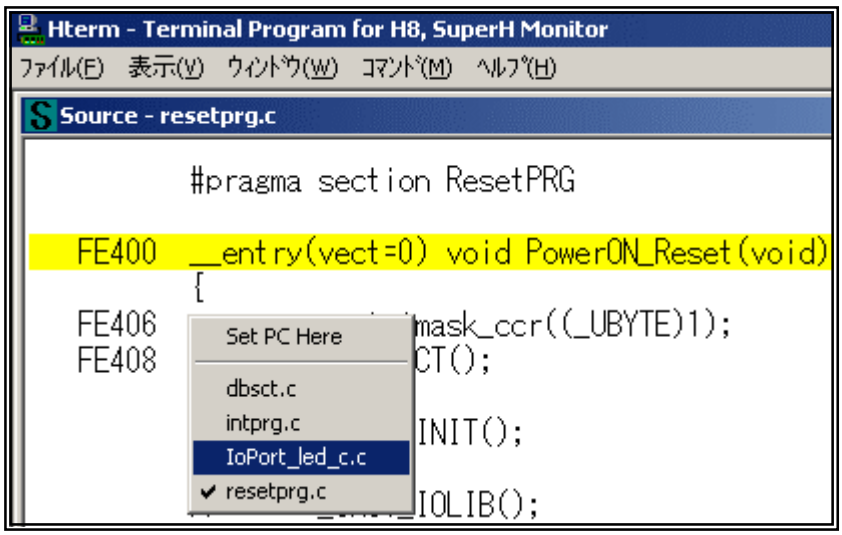
なお，リセットするとPCはプログラムの先頭アドレスになるのではなく，0にクリアされることに注意してください。上図でも逆アセンブルウィンドウの 0 番地が黄色でマークされています。

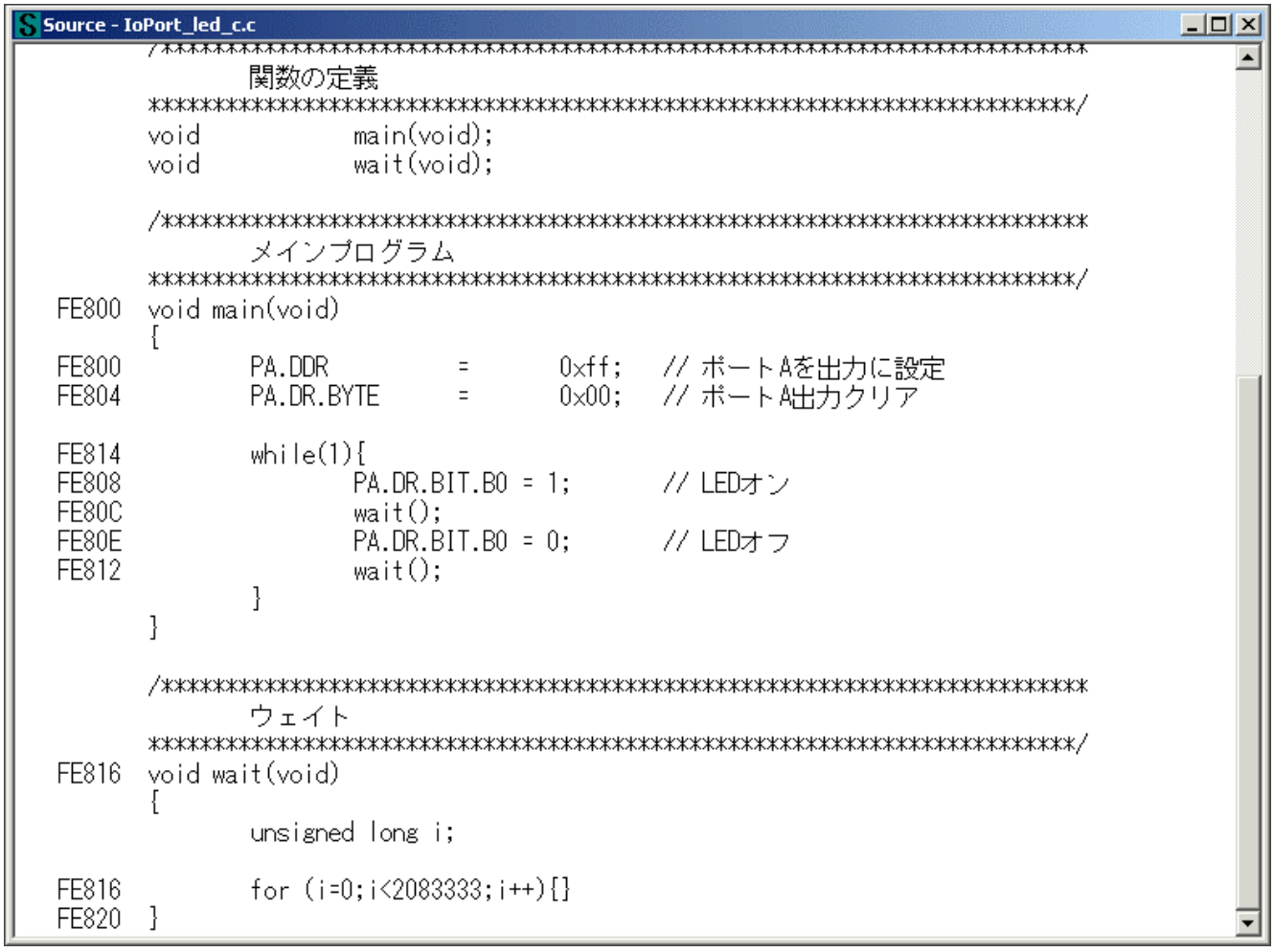
続いて，もう一度プログラムの先頭アドレスを PC にセットします。ソースウィンドウの FE400 番地にマウスカーソルをあわせて右クリックします。するとメニューが表示されますので「Set PC Hera」を選択してクリックしてください。これで FE400 が PC にセットされます。



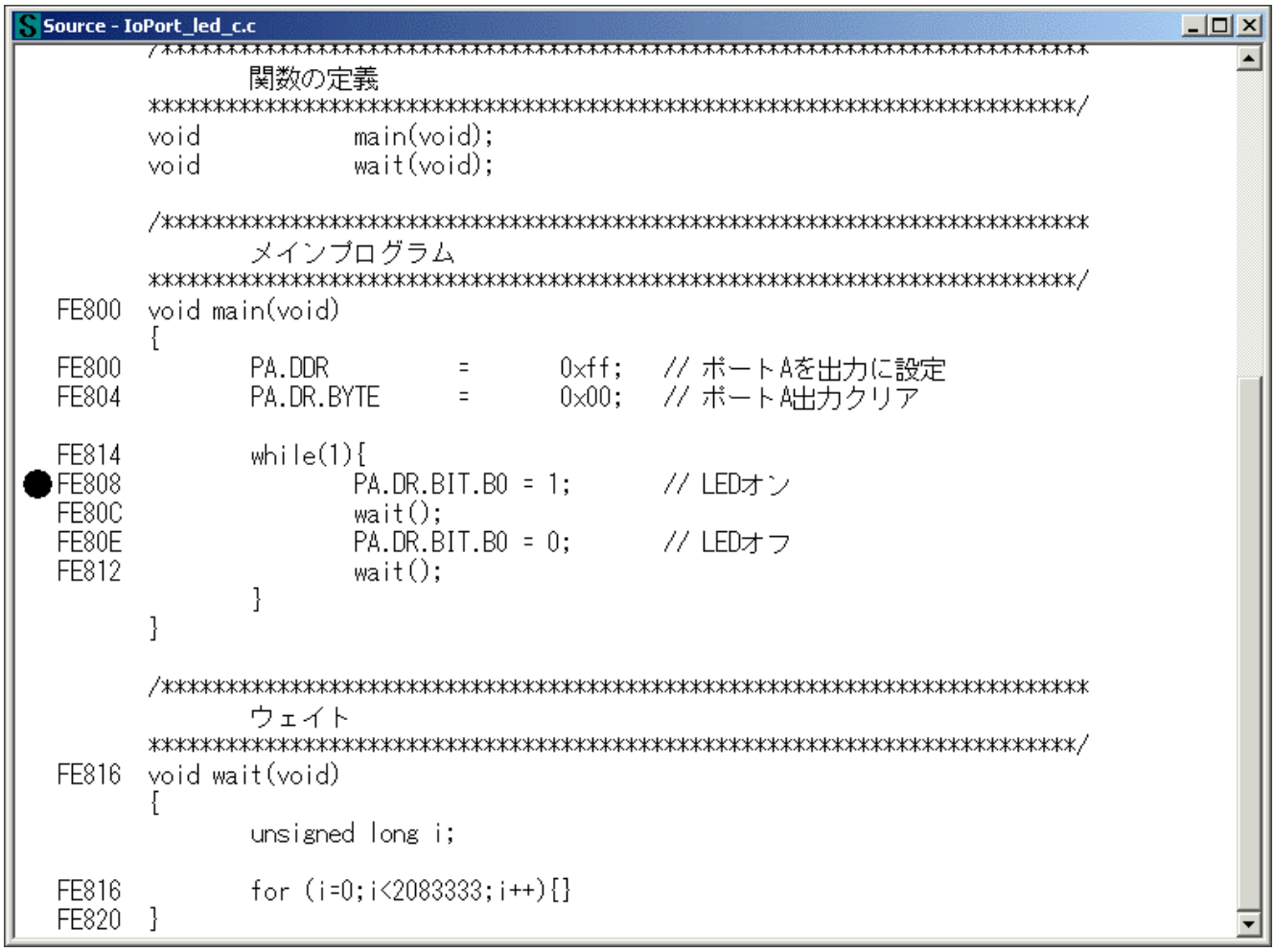


再びプログラムを実行するのですが，今度はメインループに入ったところでプログラムをストップしてみましょう。ストップするアドレスをブレークポイントと呼びます。まず，ソースウィンドウの表示をメインループのある「IoPort_led_c.c」に切り替えます。ソースウィンドウの上で右クリックしてください。メニューが表示されますので，「IoPort_led_c. c」を選択してクリックします。すると表示が切り替わります。

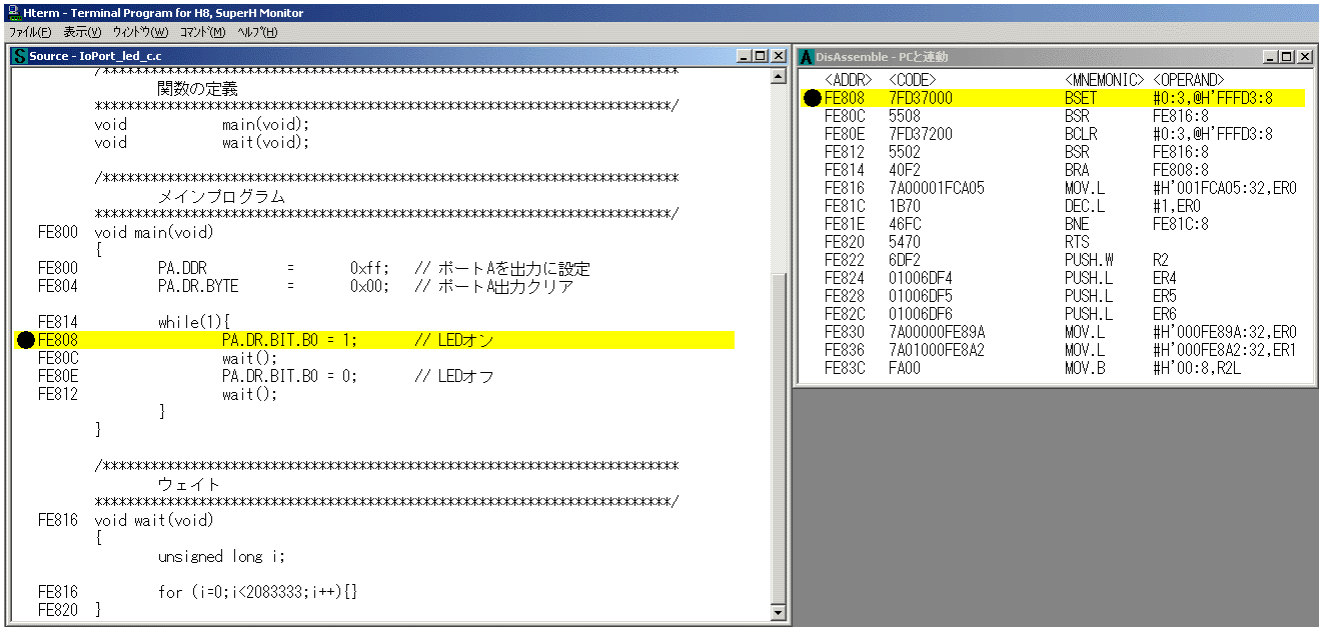




ストップしたいアドレス(今回は FE808 番地)にマウスカーソルをあわせてダブルクリックします。FE808 番地の行に「●」が表示されました。



ファンクションキーの[F5]を押すか, メニューバーから「Go(G)」を選んでください。現在の PC からスタートし, ブレークポイントでストップします。



あとは，このときのレジスタやメモリの値を確認したり，続きをステップ実行して考えたとおりに動作するか確認したりします。考えたとおりにプログラムが動くようになったらデバッグ完了です。

◆

「Hterm」には，ここで説明した機能のほかにもデバッグに便利な機能がたくさんあります。全ての機能を説明することはできませんので，「Hterm」の HELP を一読されることをおすすめします。

◆

ここまで HEW を使った C 言語プログラムの作成方法と，Hterm を使ったデバッグの方法を説明してきました。次の章から H8/3052 の内部 I/O の使い方を実習します。

第4章 I/O ポートの使い方

H8/3052 には様々な機能が内蔵されています。この章以降いくつかの内蔵周辺機能の使い方を説明します。なお，内蔵周辺機能の詳しい内容は「H8/3052B F-ZTAT ハードウェアマニュアル」(これからハードウェアマニュアルと呼びます)で説明されています。本書では説明されていない機能がたくさんありますので，ぜひお読みください。

1. I/O とは何か

そもそも I/O とは何でしょうか。第 1 章では，

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

と，簡単に説明しました。ここでは，もっと詳しく説明しましょう。

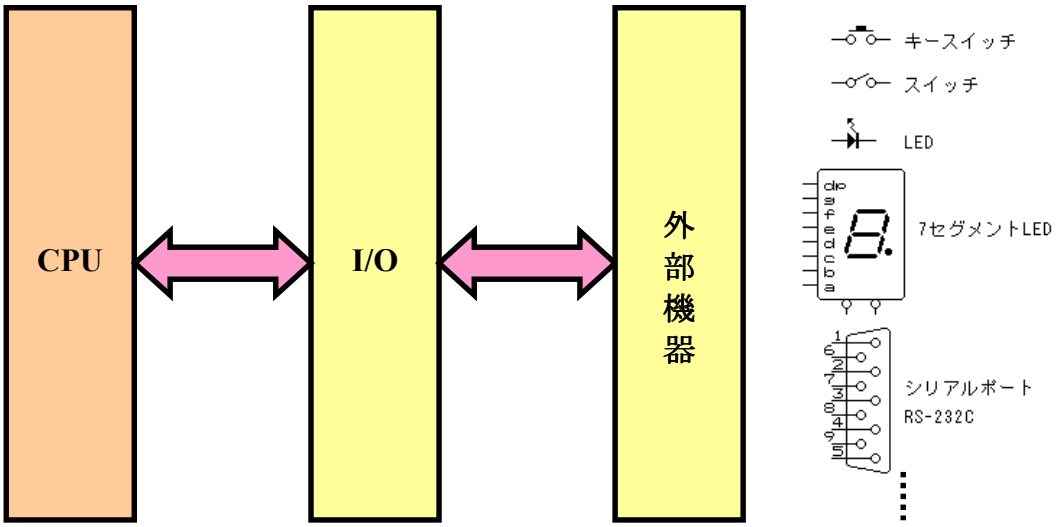
もともと CPU は「できるだけ速く」を合言葉に進歩してきました。H8/3052 は 1 つの命令を 0．08μs～0．88μs (μs:マイクロ秒は 1 秒の百万分の一)で実行できるように作られています。

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ありません。リレーなどはマイコンでオンしてから実際にオンで安定するまで数ミリ秒かかります。

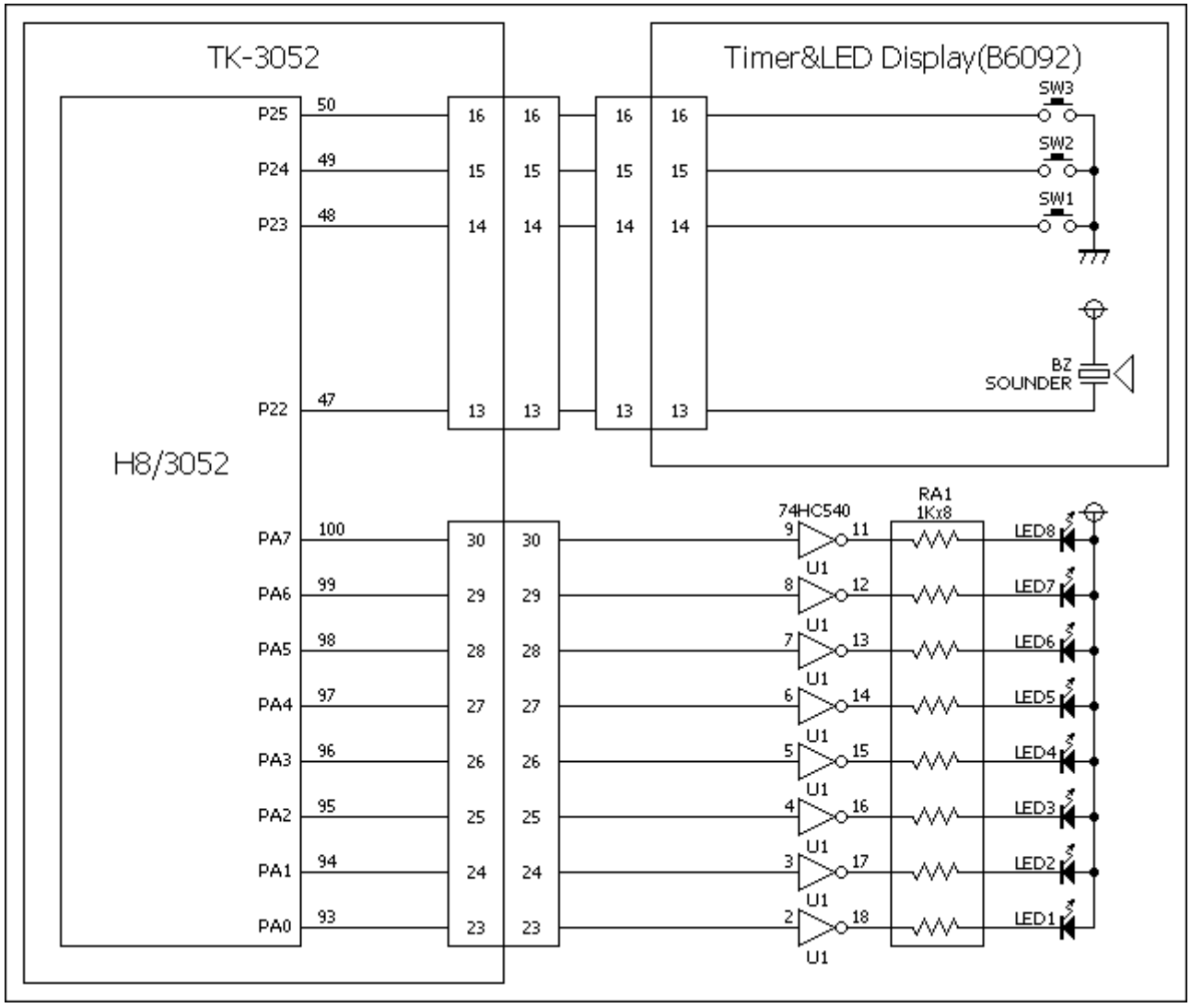
また，一般的に CPU の動作電圧は 5V で(最近は 3．3V が増えていて，さらに低くなる傾向にある)，できるだけ少ない電流で動くように発展してきましたが，外部機器の中には 12V だったり電流がたくさん必要だったりするものがあります。

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

第 1 章の「H8/3052 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，本章で I/O ポート，あるいはポートといえば，パラレルポートをさすものとします。



2. 実習回路



全体の回路図は付録をご覧ください。この章で使う回路だけ抜き出してみました。

3. 設定用レジスタ

H8/3052 の端子はいくつもの機能が兼用されています。それで，各ピンをどの機能で使うか設定する必要があります。実習回路はポート 2 にスイッチとサウンダ，ポート A に LED がつながっています。それで，ポート 2 とポート A の設定用レジスタについて説明します。なお，I/O ポートの詳細については「H8/3052B　F-ZTAT　ハードウェアマニュアル」の「9. I/O ポート」をご覧ください。

ポート 2 は，プログラムで制御可能なプルアップ MOS が内蔵されています。また，1 個の TTL 負荷と 90pF の容量負荷を駆動することができます。ダーリントントランジスタを駆動することもできます。さらに，LED を駆動（シンク電流 10mA）することができます。

ポート 2 に関係するレジスタは P2DDR，P2DR，P2PCR です。

ポート 2 データディレクションレジスタ（P2DDR）　：　アドレス=0xFFC1 番地（下位 16 ビット）

ビット	ビット名	初期値	R/W	説明
7	P27DDR	0	W	各ビットの入出力の設定。 0：入力ポート。 / 1：出力ポート。
6	P26DDR	0	W	
5	P25DDR	0	W	
4	P24DDR	0	W	
3	P23DDR	0	W	
2	P22DDR	0	W	
1	P21DDR	0	W	
0	P20DDR	0	W	

ポート 2 データレジスタ（P2DR）　：　アドレス=0xFFC3 番地（下位 16 ビット）

ビット	ビット名	初期値	R/W	説明
7	P27	0	R/W	データレジスタ。入力ポートに設定されているビットの端子の状態を読み出したり，出力ポートに設定されている端子の出力値を格納したりする。
6	P26	0	R/W	
5	P25	0	R/W	
4	P24	0	R/W	
3	P23	0	R/W	
2	P22	0	R/W	
1	P21	0	R/W	
0	P20	0	R/W	

ポート 2 入力プルアップ MOS コントロールレジスタ（P2PCR）　：　アドレス=0xFFD8 番地（下位 16 ビット）

ビット	ビット名	初期値	R/W	説明
7	P27PCR	0	R/W	各ビットのプルアップ MOS の設定。 0：プルアップしない。 / 1：プルアップする。
6	P26PCR	0	R/W	
5	P25PCR	0	R/W	
4	P24PCR	0	R/W	
3	P23PCR	0	R/W	
2	P22PCR	0	R/W	
1	P21PCR	0	R/W	
0	P20PCR	0	R/W	

ポートAは，1個のTTL負荷と30pFの容量負荷を駆動することができます。ダーリントントランジスタを駆動することもできます。また，シュミット入力になっています。

ポートAに関係するレジスタはPADDR，PADRです。

ポートA データディレクションレジスタ(PADDR) ： アドレス=0xFFD1 番地(下位 16 ビット)				
ビット	ビット名	初期値	R/W	説明
7	PA7DDR	0	W	各ビットの入出力の設定。 0:入力ポート。 / 1:出力ポート。
6	PA6DDR	0	W	
5	PA5DDR	0	W	
4	PA4DDR	0	W	
3	PA3DDR	0	W	
2	PA2DDR	0	W	
1	PA1DDR	0	W	
0	PA0DDR	0	W	

ポートA データレジスタ(PADR) ： アドレス=0xFFD3 番地(下位 16 ビット)				
ビット	ビット名	初期値	R/W	説明
7	PA7	0	R/W	データレジスタ。入力ポートに設定されているビットの端子の状態を読み出したり，出力ポートに設定されている端子の出力値を格納したりする。
6	PA6	0	R/W	
5	PA5	0	R/W	
4	PA4	0	R/W	
3	PA3	0	R/W	
2	PA2	0	R/W	
1	PA1	0	R/W	
0	PA0	0	R/W	

ところで，HEWでプロジェクトを作成すると，自動的に「iodefine. h」が生成されます。通常H8/3052のI/Oにアクセスする際，「iodefine. h」で定義されている名称を使います。しかし，「iodefine. h」は構造体や共用体を駆使して定義されているため，最初はとっつきにくく感じるかもしれません。それでも慣れてくると非常に便利です。

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

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

一例ですが「iodefine. h」でポートAは次のように定義されています。

```
                                    ∫
struct st_pa {                                  /* struct PA    */
            unsigned char      DDR;             /* PADDR        */
            char               wk;              /*              */
            union {                             /* PADR         */
                  unsigned char BYTE;           /*  Byte Access */
                  struct {                      /*  Bit  Access */
                        unsigned char B7:1;     /*    Bit 7     */
                        unsigned char B6:1;     /*    Bit 6     */
                        unsigned char B5:1;     /*    Bit 5     */
                        unsigned char B4:1;     /*    Bit 4     */
                        unsigned char B3:1;     /*    Bit 3     */
                        unsigned char B2:1;     /*    Bit 2     */
                        unsigned char B1:1;     /*    Bit 1     */
```

```
                        unsigned char B0:1;          /*    Bit 0     */
                        }      BIT;                  /*              */
                    }          DR;                   /*              */
};                                                   /*              */
                                                     ∫
#define PA      (*(volatile struct st_pa    *)0xFFFFD1) /* PA    Address*/
```

それでは，ポート A データディレクションレジスタ(PADDR)に値をセットしてみましょう。ポート A は全ビット出力で使いますので，セットする値は 0xFF です。

I/O ポート: ポート A データディレクションレジスタ									
モジュール名称	レジスタ名称	ビット名称							
		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
PA	DDR	（ビットアクセスは未定義）							

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

```
PA.DDR = 0xff;
```

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

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

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

```
PA.DR.BYTE = 0x55;
```

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

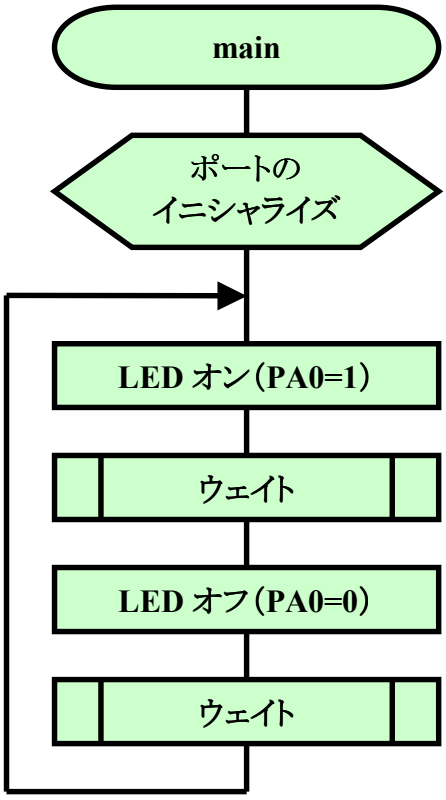
```
PA.DR.BIT.B0 = 0;
```

4. LED の点滅

それでは第 3 章で入力した PA0 につながった LED の点滅プログラムを詳しく見てみましょう。ポート A は全ビット LED につながっているので出力に設定します。それで，ポート A データディレクションレジスタに 0xFF をセットします。

メインループでは LED を点滅します。CPU の速度で点滅させると人間の目では識別できなくなるので，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。少し待つ処理は単純ループで作ります。

以上をフローチャートにすると右のようになります。



では，コーディングしてみましょう。

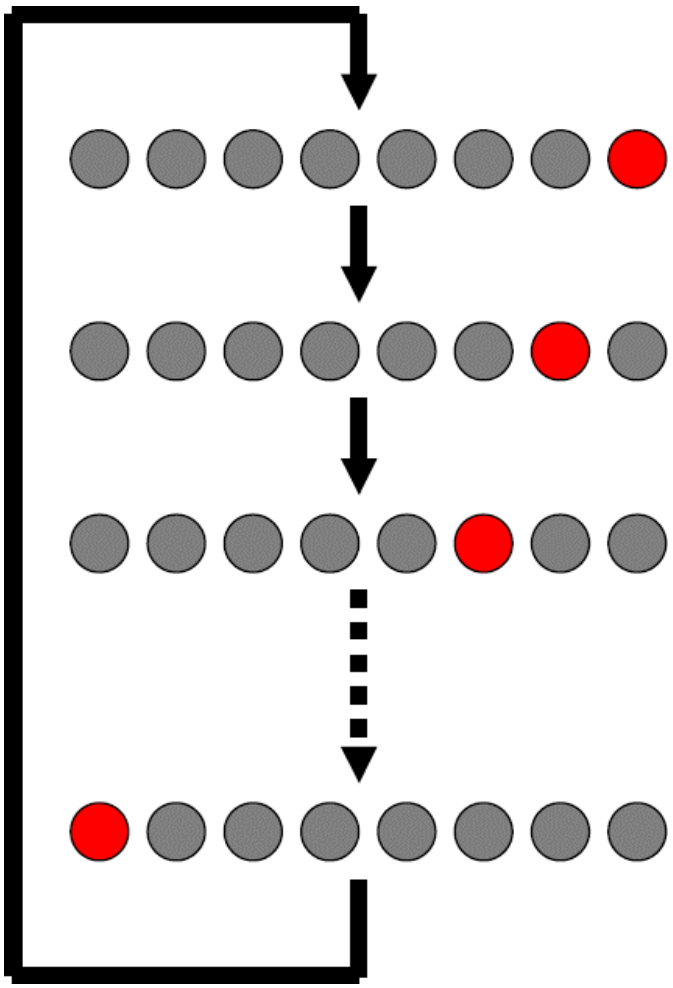
```
/************************************************************************/
/*                                                                      */
/*  FILE        :IoPort_led_c.c                                         */
/*  DATE        :Wed, Jun 02, 2010                                      */
/*  DESCRIPTION :Main Program                                           */
/*  CPU TYPE    :H8/3052F                                               */
/*                                                                      */
/*  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)
{
     PA.DDR        =    0xff;     // ポートAを出力に設定
     PA.DR.BYTE    =    0x00;     // ポートA出力クリア

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

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

     for (i=0;i<2083333;i++){}
}
```

付属の CD-ROM にはあらかじめダウンロードするファイルがおさめられています。「IoPort_led_c. abs」をダウンロードして実行してください。

■ 練習問題(1)

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

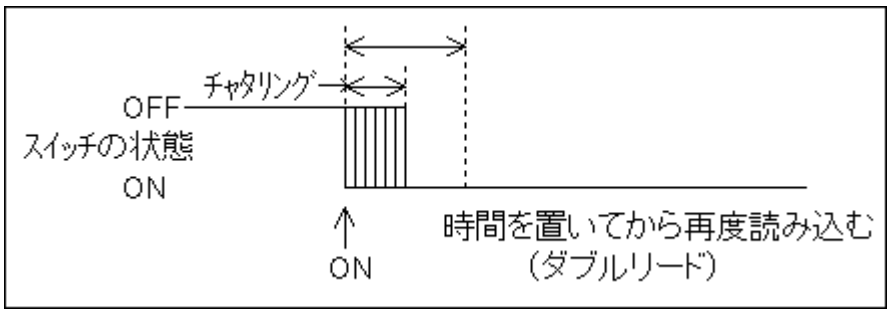


5. スイッチの入力

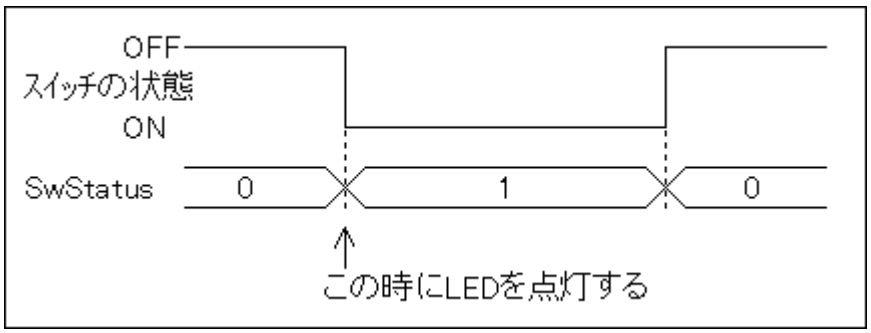
次は入力ポートの例として，プッシュスイッチの入力を考えてみましょう。スイッチが押された瞬間だけ，0．2 秒間（200ms）LED が光る（ワンショット動作），というプログラムを作ります。プッシュスイッチは P23 につながっているものを使います。LED は PA0 です。

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

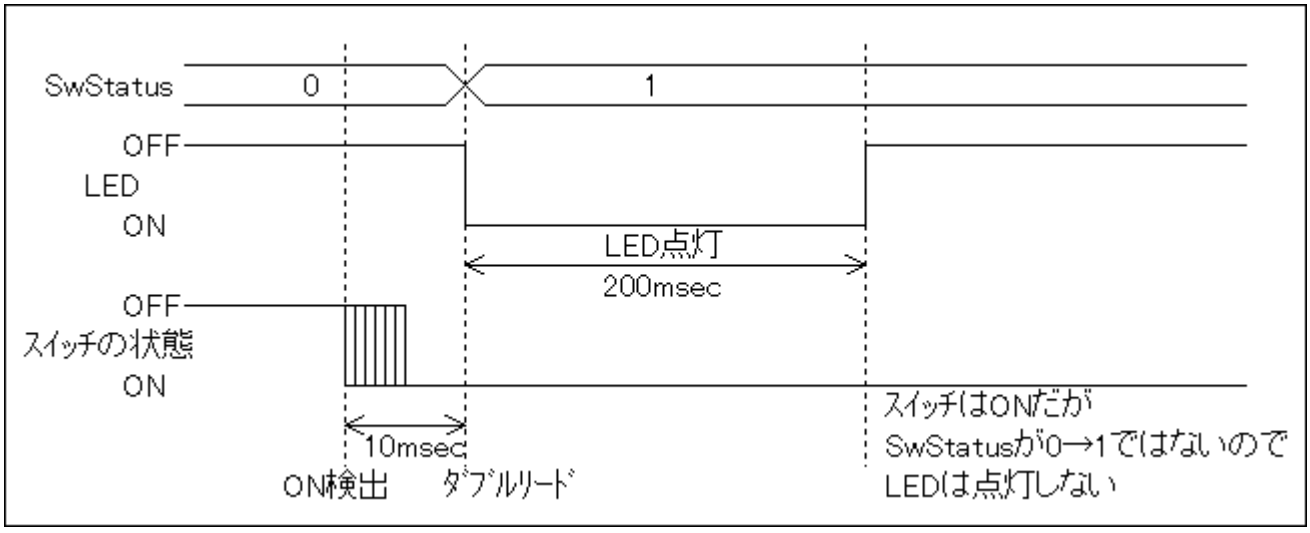
チャタリングを取り除くために，スイッチがオンしたら，しばらく待ってから（数 ms）もう一度読む（ダブルリード），ということを行ないます。2 度目に読んだときもオンだったら本当にスイッチがオンしたと見なします。



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



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



```
/************************************************************************/
/*                                                                      */
/*  FILE        :IoPort_sw_led_c.c                                      */
/*  DATE        :Wed, Jun 02, 2010                                      */
/*  DESCRIPTION :Main Program                                           */
/*  CPU TYPE    :H8/3052F                                               */
/*                                                                      */
/*  This file is programed by TOYO-LINX Co.,Ltd. / yKikuchi             */
/*                                                                      */
/************************************************************************/

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

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

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

/************************************************************************
    メインプログラム
************************************************************************/
void main(void)
{
    P2.PCR.BYTE    =    0x38;     // ポート2プルアップ抵抗の設定
    P2.DDR         =    0x00;     // ポート2を入力に設定
    PA.DDR         =    0xff;     // ポートAを出力に設定
    PA.DR.BYTE     =    0x00;     // LEDオフ

    while(1){
        if (P2.DR.BIT.B3==0){            //スイッチオン
            wait10();                    //ちょっと待つ
            if (P2.DR.BIT.B3==0){        //やっぱりスイッチオン
                if (SwStatus==0){        //今までスイッチオフだった
                    SwStatus = 1;        //スイッチオンを記憶
                    PA.DR.BIT.B0 = 1;    //LEDオン
                    wait200();           //しばらく待つ
                    PA.DR.BIT.B0 = 0;    //LEDオフ
                }
            }
            else{                        //スイッチオフだった
                SwStatus = 0;
            }
```

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

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

    for (i=0;i<41666;i++){}
}

void wait200(void)
{
    unsigned long i;

    for (i=0;i<833333;i++){}
}
```

付属の CD-ROM にはあらかじめダウンロードするファイルがおさめられています。「IoPort_sw_led_c. abs」をダウンロードして実行してください.

■ 練習問題(2)

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

■ 練習問題(3)

スイッチが押された瞬間だけ, 0. 1 秒間(100ms)サウンダが 1KHzで鳴動し, スイッチが押されている間ずっと LED が点灯するプログラムを作りなさい。プッシュスイッチは P23, サウンダは P22, LED は PA0 につながっています。(解答例は「IoPort_Sounder」)

第5章 外部割り込みの使い方

1. 割り込みとは何か

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

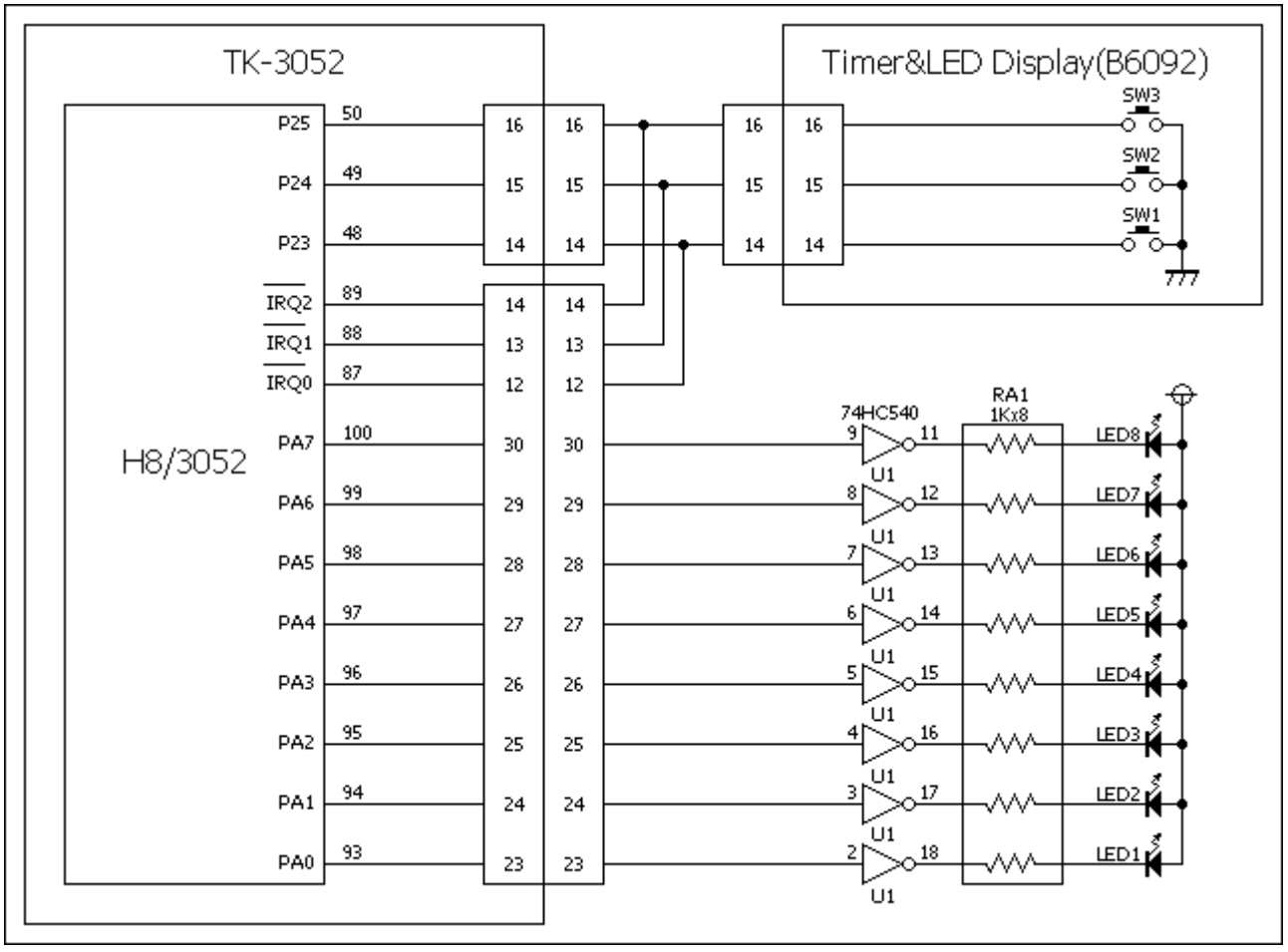
このようなときに使うのが割り込みです。H8/3052 には NMI, IRQ0, IRQ1, IRQ2, IRQ3, IRQ4, IRQ5 という 7 つの端子が用意されていて，それぞれの端子のレベルや変化によって割込みをかけたり，内蔵周辺モジュールから割り込みをかけたりして，今行なっている処理を一時中断して特定の処理を実行することができます。それで，割込みとはハードウェア的にサブルーチンを実行する方法，とも言えるでしょう。

NMI は「ノンマスカブル割り込み」で，マスク不可能な外部割り込みです。立ち上がりエッジか立ち下がりエッジのいずれかで割り込み処理を起動します（ソフトウェアで指定可能）。

IRQ0～5 は「外部割込み要求 0～5」でマスク可能な外部割り込みです。立ち下がりエッジか Low 入力のいずれかで割り込み処理を起動します（ソフトウェアで指定可能）。

なお，詳細についてはハードウェアマニュアルの「4. 例外処理」と「5. 割り込みコントローラ」をご覧ください。

2. 実習回路



3. 割り込み処理の概要

割り込み要求は H8/3052 に内蔵されている割り込みコントローラによって制御されます。外部割り込み処理の概要は次のとおりです。

割り込み要因	要因発生元	ベクタ番号	ベクタアドレス*	IPR	優先順位
NMI	外部端子	7	H'001C～H'001F	―	高
IRQ_0		12	H'0030～H'0033	IPRA7	▲
IRQ_1		13	H'0034～H'0037	IPRA6	
IRQ_2 IRQ_3		14 15	H'0038～H'003B H'003C～H'003F	IPRA5	
IRQ_4 IRQ_5		16 17	H'0040～H'0043 H'0044～H'0047	IPRA4	
リザーブ	―	18 19	H'0048～H'004B H'004C～H'004F		
WOVI（インターバルタイマ）	ウォッチドッグタイマ	20	H'0050～H'0053	IPRA3	
CMI（コンペアマッチ）	リフレッシュコントローラ	21	H'0054～H'0057		
リザーブ	―	22 23	H'0058～H'005B H'005C～H'005F		
IMIA0（コンペアマッチ／インプットキャプチャ A0） IMIB0（コンペアマッチ／インプットキャプチャ B0） OVI0（オーバフロー0）	ITU チャネル 0	24 25 26	H'0060～H'0063 H'0064～H'0067 H'0068～H'006B	IPRA2	
リザーブ	―	27	H'006C～H'006F		
IMIA1（コンペアマッチ／インプットキャプチャ A1） IMIB1（コンペアマッチ／インプットキャプチャ B1） OVI1（オーバフロー1）	ITU チャネル 1	28 29 30	H'0070～H'0073 H'0074～H'0077 H'0078～H'007B	IPRA1	
リザーブ	―	31	H'007C～H'007F		
IMIA2（コンペアマッチ／インプットキャプチャ A2） IMIB2（コンペアマッチ／インプットキャプチャ B2） OVI2（オーバフロー2）	ITU チャネル 2	32 33 34	H'0080～H'0083 H'0084～H'0087 H'0088～H'008B	IPRA0	
リザーブ	―	35	H'008C～H'008F		
IMIA3（コンペアマッチ／インプットキャプチャ A3） IMIB3（コンペアマッチ／インプットキャプチャ B3） OVI3（オーバフロー3）	ITU チャネル 3	36 37 38	H'0090～H'0093 H'0094～H'0097 H'0098～H'009B	IPRB7	
リザーブ	―	39	H'009C～H'009F		
IMIA4（コンペアマッチ／インプットキャプチャ A4） IMIB4（コンペアマッチ／インプットキャプチャ B4） OVI4（オーバフロー4）	ITU チャネル 4	40 41 42	H'00A0～H'00A3 H'00A4～H'00A7 H'00A8～H'00AB	IPRB6	
リザーブ	―	43	H'00AC～H'00AF		低
DEND0A DEND0B DEND1A DEND1B	DMAC	44 45 46 47	H'00B0～H'00B3 H'00B4～H'00B7 H'00B8～H'00BB H'00BC～H'00BF	IPRB5	▲
リザーブ	―	48 49 50 51	H'00C0～H'00C3 H'00C4～H'00C7 H'00C8～H'00CB H'00CC～H'00CF	―	
ERI0（受信エラー0） RXI0（受信データフル 0） TXI0（送信データエンプティ 0） TEI0（送信終了 0）	SCI チャネル 0	52 53 54 55	H'00D0～H'00D3 H'00D4～H'00D7 H'00D8～H'00DB H'00DC～H'00DF	IPRB3	
ERI1（受信エラー1） RXI1（受信データフル 1） TXI1（送信データエンプティ 1） TEI1（送信終了 1）	SCI チャネル 1	56 57 58 59	H'00E0～H'00E3 H'00E4～H'00E7 H'00E8～H'00EB H'00EC～H'00EF	IPRB2	
ADI（A/D エンド）	A/D	60	H'00F0～H'00F3	IPRB1	低

【注】 * アドレスの下位 16 ビットを示しています。

1. IRQ0～5 端子に立ち下がりエッジか Low レベルの信号が入力されると，割り込みコントローラに割り込み要求信号を送ります。(立ち下がりエッジか Low レベルかは個別に選択可能)
2. 割り込みコントローラは CPU に対して割り込み処理を要求します。
3. CPU は割り込みを受け付けると，実行中の命令を実行したあと，プログラムカウンタと CCR (コンディションコードレジスタ)をスタックにプッシュします。
4. CPU は CCR の I ビットを 1 にセットして他の割り込みを受け付けないようにします。
5. CPU は外部割り込み要求に対するベクタアドレス(NMI のとき 0x001C，IRQ0 のとき 0x0030，IRQ1 のとき 0x0034，IRQ2 のとき 0x0038，IRQ3 のとき 0x003C，IRQ4 のとき 0x0040，IRQ5 のとき 0x0044)を生成し，そのベクタアドレスにセットされているデータ(割り込み処理のスタートアドレス)をプログラムカウンタにセットします。
6. 割り込み処理がスタートします。

4. 設定用レジスタ

割り込みコントローラの設定用レジスタのうち，NMI や IRQ0～5 に関係するレジスタは次のとおりです。関係するビットを黄色でマークします。

システムコントロールレジスタ(SYSCR) ： アドレス=0xFFF2 番地(下位 16 ビット)				
ビット	ビット名	初期値	R/W	説明
7	SSBY	0	R/W	ソフトウェアスタンバイ。 0：SLEEP 命令実行後，スリープモードに遷移。 1：SLEEP 命令実行後，ソフトウェアスタンバイモードに以降。
6	STS2	0	R/W	スタンバイタイマセレクト 2～0 000：8192 ステート，001：16384 ステート，010：32768 ステート 011：65536 ステート，100：131072 ステート，101：1024 ステート 11－：使用禁止
5	STS1	0	R/W	
4	STS0	0	R/W	
3	UE	1	R/W	ユーザビットイネーブル。 0：CCR の UI ビットを割り込みマスクビットとして使用。 1：CCR の UI ビットをユーザビットとして使用。
2	NMIEG	0	R/W	NMI エッジセレクト。 0：NMI 入力の立ち下がりエッジで割り込み要求を発生。 1：NMI 入力の立ち上がりエッジで割り込み要求を発生。
1	－	1	－	リザーブビット。
0	RAME	1	R/W	RAM イネーブル。 0：内蔵 RAM 無効。 1：内蔵 RAM 有効。

IRQ センスコントロールレジスタ(ISCR) ： アドレス=0xFFF4 番地(下位 16 ビット)				
ビット	ビット名	初期値	R/W	説明
7	－	0	－	リザーブビット。
6	－	0	－	リザーブビット。
5	IRQ5SC	0	R/W	IRQ5 センスコントロール 0：IRQ5 入力の Low レベルで割り込み要求を発生。 1：IRQ5 入力の立ち下がりエッジで割り込み要求を発生
4	IRQ4SC	0	R/W	IRQ4 センスコントロール 0：IRQ4 入力の Low レベルで割り込み要求を発生。 1：IRQ4 入力の立ち下がりエッジで割り込み要求を発生
3	IRQ3SC	0	R/W	IRQ3 センスコントロール 0：IRQ3 入力の Low レベルで割り込み要求を発生。 1：IRQ3 入力の立ち下がりエッジで割り込み要求を発生
2	IRQ2SC	0	R/W	IRQ2 センスコントロール 0：IRQ2 入力の Low レベルで割り込み要求を発生。 1：IRQ2 入力の立ち下がりエッジで割り込み要求を発生
1	IRQ1SC	0	R/W	IRQ1 センスコントロール 0：IRQ1 入力の Low レベルで割り込み要求を発生。 1：IRQ1 入力の立ち下がりエッジで割り込み要求を発生
0	IRQ0SC	0	R/W	IRQ0 センスコントロール 0：IRQ0 入力の Low レベルで割り込み要求を発生。 1：IRQ0 入力の立ち下がりエッジで割り込み要求を発生

IRQ イネーブルレジスタ(IER) ： アドレス=0xFFF5 番地(下位 16 ビット)				
ビット	ビット名	初期値	R/W	説明
7	−	0	−	リザーブビット。
6	−	0	−	リザーブビット。
5	IRQ5E	0	R/W	IRQ5 イネーブル 0:IRQ5 割り込みを禁止。 1:IRQ5 割り込みを許可。
4	IRQ4E	0	R/W	IRQ4 イネーブル 0:IRQ4 割り込みを禁止。 1:IRQ4 割り込みを許可。
3	IRQ3E	0	R/W	IRQ3 イネーブル 0:IRQ3 割り込みを禁止。 1:IRQ3 割り込みを許可。
2	IRQ2E	0	R/W	IRQ2 イネーブル 0:IRQ2 割り込みを禁止。 1:IRQ2 割り込みを許可。
1	IRQ1E	0	R/W	IRQ1 イネーブル 0:IRQ1 割り込みを禁止。 1:IRQ1 割り込みを許可。
0	IRQ0E	0	R/W	IRQ0 イネーブル 0:IRQ0 割り込みを禁止。 1:IRQ0 割り込みを許可。

IRQ ステータスレジスタ(ISR) ： アドレス=0xFFF6 番地(下位 16 ビット)				
ビット	ビット名	初期値	R/W	説明
7	－	0	－	リザーブビット。
6	－	0	－	リザーブビット。
5	IRQ5F	0	R/W	IRQ5 フラグ [クリア条件](1)IRQ5F=1 の状態で IRQ5F フラグをリードした後，IRQ5F フラグに 0 をライトしたとき。(2)IRQ5SC=0，IRQ5 入力が High レベルの状態で割り込み例外処理を実行したとき。(3)IRQ5SC=1 の状態で IRQ5 割り込み例外処理を実行したとき。 [セット条件](1)IRQ5SC=0 の状態で IRQ5 入力が Low レベルになったとき。(2)IRQ5SC=1 の状態で IRQ5 入力に立ち下がりエッジが発生したとき。
4	IRQ4F	0	R/W	IRQ4 フラグ [クリア条件](1)IRQ4F=1 の状態で IRQ4F フラグをリードした後，IRQ4F フラグに 0 をライトしたとき。(2)IRQ4SC=0，IRQ4 入力が High レベルの状態で割り込み例外処理を実行したとき。(3)IRQ4SC=1 の状態で IRQ4 割り込み例外処理を実行したとき。 [セット条件](1)IRQ4SC=0 の状態で IRQ4 入力が Low レベルになったとき。(2)IRQ4SC=1 の状態で IRQ4 入力に立ち下がりエッジが発生したとき。
3	IRQ3F	0	R/W	IRQ3 フラグ [クリア条件](1)IRQ3F=1 の状態で IRQ3F フラグをリードした後，IRQ3F フラグに 0 をライトしたとき。(2)IRQ3SC=0，IRQ3 入力が High レベルの状態で割り込み例外処理を実行したとき。(3)IRQ3SC=1 の状態で IRQ3 割り込み例外処理を実行したとき。 [セット条件](1)IRQ3SC=0 の状態で IRQ3 入力が Low レベルになったとき。(2)IRQ3SC=1 の状態で IRQ3 入力に立ち下がりエッジが発生したとき。
2	IRQ2F	0	R/W	IRQ2 フラグ [クリア条件](1)IRQ2F=1 の状態で IRQ2F フラグをリードした後，IRQ2F フラグに 0 をライトしたとき。(2)IRQ2SC=0，IRQ2 入力が High レベルの状態で割り込み例外処理を実行したとき。(3)IRQ2SC=1 の状態で IRQ2 割り込み例外処理を実行したとき。 [セット条件](1)IRQ2SC=0 の状態で IRQ2 入力が Low レベルになったとき。(2)IRQ2SC=1 の状態で IRQ2 入力に立ち下がりエッジが発生したとき。
1	IRQ1F	0	R/W	IRQ1 フラグ [クリア条件](1)IRQ1F=1 の状態で IRQ1F フラグをリードした後，IRQ1F フラグに 0 をライトしたとき。(2)IRQ1SC=0，IRQ1 入力が High レベルの状態で割り込み例外処理を実行したとき。(3)IRQ1SC=1 の状態で IRQ1 割り込み例外処理を実行したとき。 [セット条件](1)IRQ1SC=0 の状態で IRQ1 入力が Low レベルになったとき。(2)IRQ1SC=1 の状態で IRQ1 入力に立ち下がりエッジが発生したとき。
0	IRQ0F	0	R/W	IRQ0 フラグ [クリア条件](1)IRQ0F=1 の状態で IRQ0F フラグをリードした後，IRQ0F フラグに 0 をライトしたとき。(2)IRQ0SC=0，IRQ0 入力が High レベルの状態で割り込み例外処理を実行したとき。(3)IRQ0SC=1 の状態で IRQ0 割り込み例外処理を実行したとき。 [セット条件](1)IRQ0SC=0 の状態で IRQ0 入力が Low レベルになったとき。(2)IRQ5SC=0 の状態で IRQ0 入力に立ち下がりエッジが発生したとき。

5. プログラムの作成

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

スイッチオンの検知は外部割り込みで行ないます。スイッチオンで信号は High から Low になりますから，立ち下がりエッジで割り込みをかけます。それで，割り込みコントローラのイニシャライズは次のようになります。

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

実際にスイッチが押されると，それぞれの割り込みプログラムにジャンプします。

```
/************************************************************************
    IRQ0 割込み
************************************************************************/
#pragma regsave (intprog_irq0)
void intprog_irq0(void)
{
    DispData++;                     // On → インクリメント
    INTC.ISR.BIT.IRQ0F = 0;         // 割込み要求フラグクリア
}

/************************************************************************
    IRQ1 割込み
************************************************************************/
#pragma regsave (intprog_irq1)
void intprog_irq1(void)
{
    DispData--;                     // On → デクリメント
    INTC.ISR.BIT.IRQ1F = 0;         // 割込み要求フラグクリア
}

/************************************************************************
    IRQ2 割込み
************************************************************************/
#pragma regsave (intprog_irq2)
void intprog_irq2(void)
{
    DispData = rotlc(1,DispData);   // On → 左ローテート
    INTC.ISR.BIT.IRQ2F = 0;         // 割込み要求フラグクリア
}
```

さて，割り込みを使うためには，IRQ0～2 に対応するベクタアドレスに，割り込み処理のスタートアドレスをセットする必要があります。通常は HEW が自動生成する「intprg. c」を修正して割り込み処理のスタートアドレスをベクタアドレスにセットします。

ところで，IRQ0～2 のベクタアドレスはそれぞれ 0x00030，0x00034，0x00038 番地からですが，ここはメモリマップで見たように ROM（フラッシュメモリ）の範囲になります。わたしたちがプログラムの実行のために使っている「Hterm」は，RAM にプログラムをダウンロードして実行するタイプなのですが，ベクタアドレスだけは H8/3052 を使うかぎり変更することはできません。それで，このままだと割り込みプログラムを実行することができません。

そこで，「Hterm」ではユーザプログラム用のベクタアドレスを RAM に配置し，割り込み発生時は RAM 上の仮

想ベクタアドレスを参照してユーザの割り込みプログラムを実行することができるようになっています。弊社 CD に収められているモニタプログラムは，本来 0x00000～0x000FF 番地のベクタアドレスの領域を，0xFE000～0xFE0FF にするようカスタマイズされています。そこで，IRQ0～2 の割り込みプログラムのスタートアドレスを仮想ベクタアドレスにセットします。

しかし，ここでもう一つ問題が生じます。HEW が自動生成する「intprg. c」で IRQ0 のスタートアドレスを設定している行は次のようになっています。

```
__interrupt(vect=12) void INT_IRQ0(void) {/* sleep(); */}
```

これを，

```
__interrupt(vect=12) void INT_IRQ0(void) {intprog_irq0();}
```

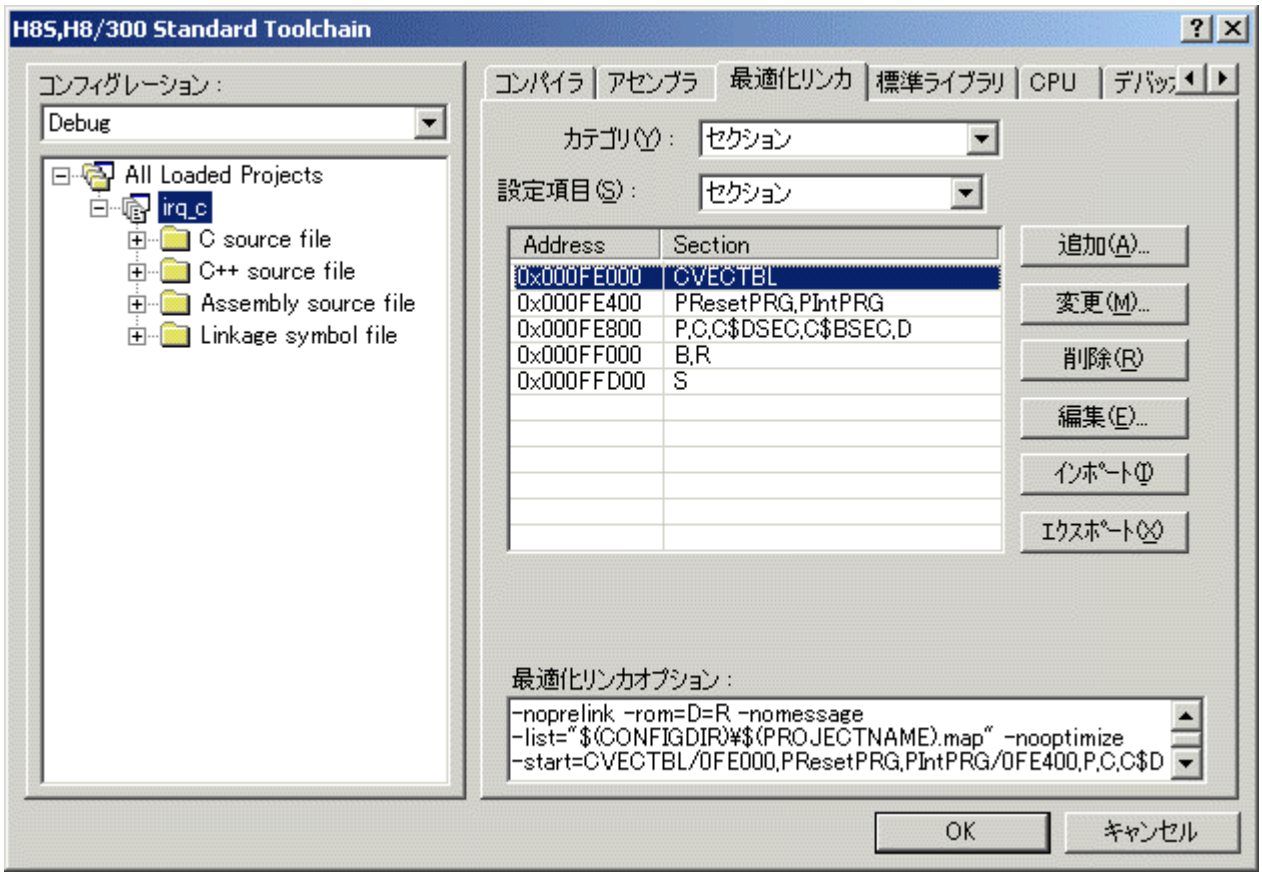
と変更するのですが，この「__interrupt(vect=12)」という命令は，「INT_IRQ0 という関数を定義し，ベクタ番号 12 のベクタアドレス(0x00030 番地)にこの関数の先頭アドレスをセットしなさい」というものです。つまり，この命令を使う限り仮想ベクタアドレスにセットすることができません。そこで，「intprg. c」を「__interrupt」命令を使わない方法で書き換えます。自動生成された「intprg. c」を，巻末の付録にある「intprg. c」に置き換え，次のように追加・変更します。

```
/************************************************************************/
/*                                                                      */
/*  FILE        :intprg.c                                               */
/*  DATE        :Wed, Jun 04, 2010                                      */
/*  DESCRIPTION :Interrupt Program                                      */
/*  CPU TYPE    :H8/3052F                                               */
/*                                                                      */
/*  This file is programed by TOYO-LINX Co.,Ltd. / yKikuchi             */
/*                                                                      */
/************************************************************************/
                ∫
extern void PowerON_Reset(void);
extern void intprog_irq0(void);
extern void intprog_irq1(void);
extern void intprog_irq2(void);
                ∫
void INT_IRQ0(void)     {intprog_irq0();}
void INT_IRQ1(void)     {intprog_irq1();}
void INT_IRQ2(void)     {intprog_irq2();}
                ∫
```

追加

変更

もう一つ，仮想ベクタアドレスのセクションを追加します。ビルドする前に下記のように 0xFE000 番地に「CVECTBL」セクションを追加してください。



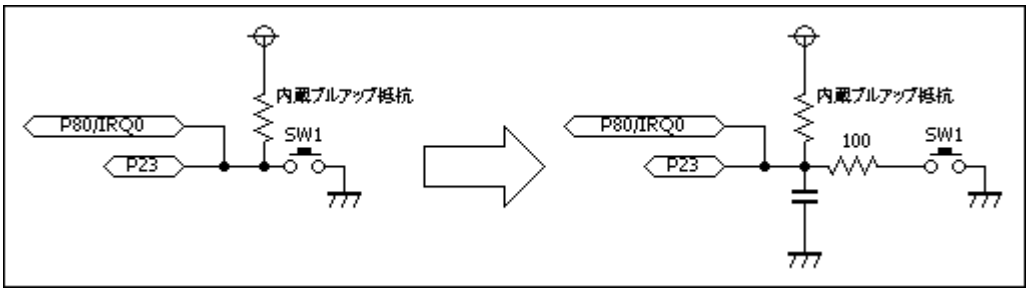
これで準備は整ったのでビルドして実行します。なお，付属の CD-ROM にはあらかじめダウンロードするファイルがおさめられています。「irq_c．abs」をダウンロードして実行してください。

■ 練習問題

おそらく，スイッチを押したときに+1 だけではなく，不規則に+2 や+3 になったりすると思います。これはスイッチのチャタリングの影響です（チャタリングについては第 4 章で説明しています）。では，チャタリングを取り除く方法を考えてみてください。（解答例は「irq_c_v2」）

♭ ♯ ♪ ♭ ♯ ♪

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



抵抗とコンデンサを1 個ずつ追加するだけです。コンデンサは使うスイッチにあわせてチャタリングが除去できる容量にします。0.1μF～1μF，チャタリングが多いスイッチなら10μFぐらいでしょうか。あまり大きいとスイッチの反応が鈍くなります。オシロスコープで波形を観察しながらカット&トライで決めます。この抵抗とコンデンサでローパスフィルタを作り，波形をなまらせてチャタリングを吸収します。プログラムは「irq_c. c」を使います。

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

第6章 16ビットインテグレーテッドタイマユニット(ITU)の使い方

マイコンの内蔵機能で I/O ポートと並んでよく使われる機能はタイマです。出力波形を作ったり，タイミングを作ったり，パルスを数えたりします。H8/3052 には 16 ビットインテグレーテッドタイマユニット(ITU)と呼ばれる高機能なタイマが内蔵されています。この章では ITU の基本的な使い方を練習してみましょう。

1. ITU とは何か

ITU は 5 チャネルの 16 ビットタイマで構成されています。各チャネルは独立しており，それぞれ CPU クロック(25MHz)の 1/1，1/2，1/4，1/8，もしくは外部クロックでカウントすることができます。ITU の機能一覧を示します。かなり多機能であることがわかりますね。ITU の詳細についてはハードウェアマニュアルの「10. 16 ビットインテグレーテッドタイマユニット(ITU)」をご覧ください。

ITU の機能一覧

項　目		チャネル 0	チャネル 1	チャネル 2	チャネル 3	チャネル 4
カウントクロック		内部クロック：φ、φ/2、φ/4、φ/8 外部クロック：TCLKA、TCLKB、TCLKC、TCLKD から独立に選択可能				
ジェネラルレジスタ(アウトプットコンペア／インプットキャプチャ兼用レジスタ)		GRA0、GRB0	GRA1、GRB1	GRA2、GRB2	GRA3、GRB3	GRA4、GRB4
バッファレジスタ		－	－	－	BRA3、BRB3	BRA4、BRB4
入出力端子		$TIOCA_0$、$TIOCB_0$	$TIOCA_1$、$TIOCB_1$	$TIOCA_2$、$TIOCB_2$	$TIOCA_3$、$TIOCB_3$	$TIOCA_4$、$TIOCB_4$
出力端子		－	－	－	－	$TOCXA_4$、$TOCXB_4$
カウンタクリア機能		GRA0/GRB0 のコンペアマッチまたはインプットキャプチャ	GRA1/GRB1 のコンペアマッチまたはインプットキャプチャ	GRA2/GRB2 のコンペアマッチまたはインプットキャプチャ	GRA3/GRB3 のコンペアマッチまたはインプットキャプチャ	GRA4/GRB4 のコンペアマッチまたはインプットキャプチャ
コンペアマッチ出力	0 出力	○	○	○	○	○
	1 出力	○	○	○	○	○
	トグル出力	○	○	－	○	○
インプットキャプチャ機能		○	○	○	○	○
同期動作		○	○	○	○	○
PWM モード		○	○	○	○	○
リセット同期 PWM モード		－	－	－	○	
相補 PWM モード		－	－	－	○	
位相計数モード		－	－	○	－	－
バッファ動作		－	－	－	○	○
DMAC の起動		GRA0 のコンペアマッチまたはインプットキャプチャ	GRA1 のコンペアマッチまたはインプットキャプチャ	GRA2 のコンペアマッチまたはインプットキャプチャ	GRA3 のコンペアマッチまたはインプットキャプチャ	－
割り込み要因		3 要因 ・コンペアマッチ／インプットキャプチャ A0 ・コンペアマッチ／インプットキャプチャ B0 ・オーバフロー	3 要因 ・コンペアマッチ／インプットキャプチャ A1 ・コンペアマッチ／インプットキャプチャ B1 ・オーバフロー	3 要因 ・コンペアマッチ／インプットキャプチャ A2 ・コンペアマッチ／インプットキャプチャ B2 ・オーバフロー	3 要因 ・コンペアマッチ／インプットキャプチャ A3 ・コンペアマッチ／インプットキャプチャ B3 ・オーバフロー	3 要因 ・コンペアマッチ／インプットキャプチャ A4 ・コンペアマッチ／インプットキャプチャ B4 ・オーバフロー

【記号説明】
○：可能
－：不可

2. 設定用レジスタ

ITU は多機能なので設定用レジスタも多いです。全て説明することはできませんが, このマニュアルで使用している部分を説明します。まずは全チャネル共通のレジスタです。

タイマスタートレジスタ(TSTR) : アドレス=0xFF60 番地(下位 16 ビット)				
ビット	ビット名	初期値	R/W	説明
7	−	1	−	リザーブビット。
6	−	1	−	リザーブビット。
5	−	1	−	リザーブビット。
4	STR4	0	R/W	TCNT4 の動作/停止。 0:停止 / 1:動作
3	STR3	0	R/W	TCNT3 の動作/停止。 0:停止 / 1:動作
2	STR2	0	R/W	TCNT2 の動作/停止。 0:停止 / 1:動作
1	STR1	0	R/W	TCNT1 の動作/停止。 0:停止 / 1:動作
0	STR0	0	R/W	TCNT0 の動作/停止。 0:停止 / 1:動作

続いて, 各チャネルに用意されているレジスタです(n=0～4, アドレスは前から順番に 0～4)。

タイマコントロールレジスタ(TCRn) : アドレス=0xFF64, 0xFF6E, 0xFF78, 0xFF82, 0xFF92 番地 (下位 16 ビット)				
ビット	ビット名	初期値	R/W	説明
7	−	1	−	リザーブビット。
6	CCLR1	0	R/W	カウンタクリア。 00:TCNT のクリア禁止。 01:GRA のコンペアマッチ/インプットキャプチャで TCNT クリア。 10:GRB のコンペアマッチ/インプットキャプチャで TCNT クリア。 11:同期クリア。
5	CCLR0	0	R/W	
4	CKEG1	0	R/W	クロックエッジ。 00:立ち上がりエッジでカウント。 01:立ち下がりエッジでカウント。 1−:立ち上がり/立ち下がりの両エッジでカウント。
3	CKEG0	0	R/W	
2	TPSC2	0	R/W	タイマプリスケーラ。 000:内部クロック, Φでカウント。 001:内部クロック, Φ/2 でカウント。 010:内部クロック, Φ/4 でカウント。 011:内部クロック, Φ/8 でカウント。 100:外部クロック, TCLKA でカウント。 101:外部クロック, TCLKB でカウント。 110:外部クロック, TCLKC でカウント。 111:外部クロック, TCLKD でカウント。
1	TPSC1	0	R/W	
0	TPSC0	0	R/W	

タイマ I/O コントロールレジスタ(TIORn) ： アドレス=0xFF65, 0xFF6F, 0xFF79, 0xFF83, 0xFF93 番地 (下位 16 ビット)				
ビット	ビット名	初期値	R/W	説明
7	−	1	−	リザーブビット。
6	IOB2	0	R/W	GRB の機能選択。 000:コンペアマッチによる端子出力禁止。 001:GRB のコンペアマッチで 0 出力。 010:GRB のコンペアマッチで 1 出力。 011:GRB のコンペアマッチでトグル出力。 100:立ち上がりエッジで GRB へインプットキャプチャ。 101:立ち下がりエッジで GRB へインプットキャプチャ。 11−:立ち上がり/立ち下がり両エッジで GRB へインプットキャプチャ。
5	IOB1	0	R/W	
4	IOB0	0	R/W	
3	−	1	−	リザーブビット。
2	IOA2	0	R/W	GRA の機能選択。 000:コンペアマッチによる端子出力禁止。 001:GRA のコンペアマッチで 0 出力。 010:GRA のコンペアマッチで 1 出力。 011:GRA のコンペアマッチでトグル出力。 100:立ち上がりエッジで GRA へインプットキャプチャ。 101:立ち下がりエッジで GRA へインプットキャプチャ。 11−:立ち上がり/立ち下がり両エッジで GRA へインプットキャプチャ。
1	IOA1	0	R/W	
0	IOA0	0	R/W	

タイマインタラプトイネーブルレジスタ(TIERn) ： アドレス=0xFF66, 0xFF70, 0xFF7A, 0xFF84, 0xFF94 番地 (下位 16 ビット)				
ビット	ビット名	初期値	R/W	説明
7	−	1	−	リザーブビット。
6	−	1	−	リザーブビット。
5	−	1	−	リザーブビット。
4	−	1	−	リザーブビット。
3	−	1	−	リザーブビット。
2	OVIE	0	R/W	オーバフローインタラプトイネーブル。 0:禁止 / 1:許可
1	IMIEB	0	R/W	インプットキャプチャ/コンペアマッチインタラプトイネーブル B。 0:禁止 / 1:許可
0	IMIEA	0	R/W	インプットキャプチャ/コンペアマッチインタラプトイネーブル A。 0:禁止 / 1:許可

タイマステータスレジスタ(TSRn) ： アドレス=0xFF67, 0xFF71, 0xFF7B, 0xFF85, 0xFF95 番地(下位 16 ビット)				
ビット	ビット名	初期値	R/W	説明
7	−	1	−	リザーブビット。
6	−	1	−	リザーブビット。
5	−	1	−	リザーブビット。
4	−	1	−	リザーブビット。
3	−	1	−	リザーブビット。
2	OVF	0	R/W	オーバフローフラグ。 [クリア条件]OVF=1 の状態で, OVF フラグをリードした後, OVF フラグに 0 をライトしたとき。 [セット条件]TCNT の値がオーバフロー(H'FFFF→H'0000), またはアンダーフロー(H'0000→H'FFFF)したとき。
1	IMFB	0	R/W	インプットキャプチャ/コンペアマッチフラグ B。 [クリア条件]IMFB=1 の状態で, IMFB フラグをリードした後, IMFB フラグに 0 をライトしたとき。 [セット条件](1)GRB がアウトプットコンペアレジスタとして機能している場合, TCNT=GRB になったとき。(2)GRB がインプットキャプチャレジスタとして機能している場合, インプットキャプチャ信号により TCNT の値が GRB に転送されたとき。
0	IMFA	0	R/W	インプットキャプチャ/コンペアマッチフラグ A。 [クリア条件]IMFA=1 の状態で, IMFA フラグをリードした後, IMFA フラグに 0 をライトしたとき。 [セット条件](1)GRA がアウトプットコンペアレジスタとして機能している場合, TCNT=GRA になったとき。(2)GRA がインプットキャプチャレジスタとして機能している場合, インプットキャプチャ信号により TCNT の値が GRA に転送されたとき。

タイマカウンタ(TCNTn) ： アドレス=0xFF68, 0xFF72, 0xFF7C, 0xFF86, 0xFF96 番地(下位 16 ビット)				
ビット	ビット名	初期値	R/W	説明
15		0	R/W	TCNT0, TCNT1 アップカウンタ。 TCNT2 位相係数モード:アップ/ダウンカウンタ。 上記以外:アップカウンタ。 TCNT3 相補 PWM モード:アップ/ダウンカウンタ。 上記以外:アップカウンタ。
14		0	R/W	
13		0	R/W	
12		0	R/W	
11		0	R/W	
10		0	R/W	
9		0	R/W	
8		0	R/W	
7		0	R/W	
6		0	R/W	
5		0	R/W	
4		0	R/W	
3		0	R/W	
2		0	R/W	
1		0	R/W	
0		0	R/W	

ジェネラルレジスタ A（GRAn） ： アドレス=0xFF6A, 0xFF74, 0xFF7E, 0xFF88, 0xFF98 番地（下位 16 ビット）				
ビット	ビット名	初期値	R/W	説明
15		0	R/W	GRA0, GRA1, GRA2 アウトプットコンペア/インプットキャプチャ兼用レジスタ。 GRA3, GRA4 アウトプットコンペア/インプットキャプチャ兼用レジスタ。 バッファレジスタ A と組み合わせることにより，バッファ動作設定可能。
14		0	R/W	
13		0	R/W	
12		0	R/W	
11		0	R/W	
10		0	R/W	
9		0	R/W	
8		0	R/W	
7		0	R/W	
6		0	R/W	
5		0	R/W	
4		0	R/W	
3		0	R/W	
2		0	R/W	
1		0	R/W	
0		0	R/W	

ジェネラルレジスタ B（GRBn） ： アドレス=0xFF6C, 0xFF76, 0xFF80, 0xFF8A, 0xFF9A 番地（下位 16 ビット）				
ビット	ビット名	初期値	R/W	説明
15		0	R/W	GRB0, GRB1, GRB2 アウトプットコンペア/インプットキャプチャ兼用レジスタ。 GRB3, GRB4 アウトプットコンペア/インプットキャプチャ兼用レジスタ。 バッファレジスタ B と組み合わせることにより，バッファ動作設定可能。
14		0	R/W	
13		0	R/W	
12		0	R/W	
11		0	R/W	
10		0	R/W	
9		0	R/W	
8		0	R/W	
7		0	R/W	
6		0	R/W	
5		0	R/W	
4		0	R/W	
3		0	R/W	
2		0	R/W	
1		0	R/W	
0		0	R/W	

◆

では次のページから，実際にプログラムを作りながら ITU の使い方を見てみましょう。

3. LED の点滅

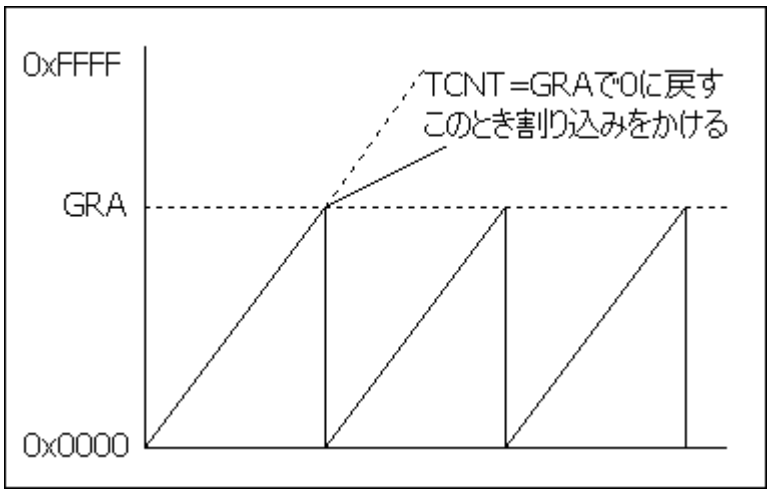
第 4 章の I/O ポートの使い方の中で作った LED の点滅間隔の 0. 5 秒はプログラムのループ回数で作りました。この方法は簡単ですが問題点もあります。ループしている間はほかに何もできない，という点です。そこで，ITU を使って時間を計測し，正確に 0. 5 秒という時間を計りながら，ほかの仕事も並行して行なう，ということを考えてみましょう。

ITU は通常，クロックが入力されるたびに各チャネルの TCNT がインクリメント(+1)します。TCNT は 16 ビットなので，0x0000～0xFFFF までカウントアップし 0x0000 に戻ります。0xFFFF から 0x0000 になるときにオーバーフローフラグがセットされ，許可されていればオーバーフローインタラプトが発生します。

さて，コンペアマッチが設定されているとき TCNT は常に GRA や GRB と比較されています。そして TCNT=GRA になるとコンペアマッチフラグ A がセットされ，許可されていればコンペアマッチインターラプト A が発生します。同じように，TCNT=GRB になるとコンペアマッチフラグ B がセットされ，許可されていればコンペアマッチインターラプト B が発生します。

TCNT は通常 0xFFFF までカウントアップするのですが，TCNT=GRA になったときや TCNT=GRB になったときに 0x0000 に戻すこともできます。

これらの機能を組み合わせることで，任意の間隔で割り込みを発生させることができます。



では，0. 5 秒の間隔を作るには GRA の値はどれくらいにすればよいでしょうか。TK-3052 のクロックは 25MHz です。ということは，25MHz から 0. 5 秒を作るわけですから，

$$GRA = \frac{0.5秒}{\frac{1}{25\mathrm{MHz}}} = 0.5秒 \times 25MHz = 0.5 \times 25 \times 10^6 = 12500000$$

となります。12500000 は 16 進数で 0xBEBC20 なので 16 ビットカウンタでは桁が足りません。ITU はクロックを 1/2，1/4，1/8 に分周できるのですが，もっとも遅い 1/8 に分周(＝3. 125MHz)したとしても GRA=1562500，つまり 16 進数で 0x17D784 となり，やはり 16 ビットカウンタでは足りません。

そこで，考え方をかえて，もう少し短い間隔で割り込みをかけ，その回数をプログラムでカウントし，0. 5 秒の間隔になったときに LED を反転することにします。今回は 1ms で割り込みをかけて，500 回割り込みがかかったら LED を反転させましょう。25MHz で 1ms をカウントするときの GRA を計算します。

$$GRA = \frac{1m秒}{\frac{1}{25\mathrm{MHz}}} = 1m秒 \times 25MHz = 1 \times 10^{-3} \times 25 \times 10^6 = 25000$$

25000 は 16 進数で 0x61A8 なので，16 ビットカウンタで十分対応できます。

では，これを踏まえてプログラムを考えてみましょう。ITU はチャネル 4 を使いました。イニシャライズは次のようになります。

```
/************************************************************************
    ITUの初期化
************************************************************************/
void ini_itu(void)
{
    ITU.TSTR.BYTE       =   _11100000B;   //タイマ停止

    ITU4.TCR.BYTE       =   _10100000B;   //GRAのコンペアマッチでTCNTクリア
                                          //立ち上がりエッジでカウント
                                          //内部クロックΦでカウント
    ITU4.TIOR.BYTE      =   _10001000B;   //コンペアマッチによる出力禁止
    ITU4.TCNT           =   0;            //TCNTクリア
    ITU4.GRA            =   0x61a8;       //GRA(1KHz)
    ITU4.TIER.BYTE      =   _11111001B;   //コンペアマッチA割り込みイネーブル
    ITU4.TSR.BYTE       =   _11111000B;   //タイマステータスレジスタクリア

    ITU.TSTR.BYTE       =   _11110000B;   //タイマスタート
}
```

ITU のイニシャライズで設定するレジスタが多いので難しく見えますが，実際にはそれほど複雑なことをしているわけではありません。1ms 毎にコンペアマッチ A 割り込みをかけるようにしてタイマをスタートするだけです。

ところで，このイニシャライズ関数では「_11100000B」というように 2 進数で設定しています。通常 C 言語では 2 進数の表記はできないので 16 進数が使われています。ただ，イニシャライズなどでは 2 進数で表記したほうがわかりやすいのも事実です。そこで，「#define」を使って 0x00～0xff については 2 進数表記を使えるように定義しています。インクルードファイルの中に「binary. h」がありますが，このファイル内でまとめて定義しています。HEW で自動生成されるわけではなく東洋リンクスで作りました。「binary. h」の中身を抜粋します。

```
/************************************************************************
   C言語で2進数を使うための定義
************************************************************************/
#define       _00000000B    0x00
#define       _00000001B    0x01
#define       _00000010B    0x02
                ∫
#define       _11111110B    0xfe
#define       _11111111B    0xff
```

次に割り込み関数です。ITU4 のコンペアマッチ A 割り込みがかかると，まずステータスレジスタをクリアして次の割り込みに備えます。この関数内で「LedCount」をデクリメントし，500 回数えたら LED を反転します。

```
/************************************************************************
    ITU4 コンペアマッチ/インプットキャプチャA4 割込み
************************************************************************/
#pragma regsave (intprog_imia4)
void intprog_imia4(void)
{
    ITU4.TSR.BIT.IMFA = 0;  //タイマステータスレジスタクリア
```

```
    LedCount--;
    if (LedCount==0){
        LedCount = 500;
        PA.DR.BYTE = ~PA.DR.BYTE;     //LED反転
    }
}
```

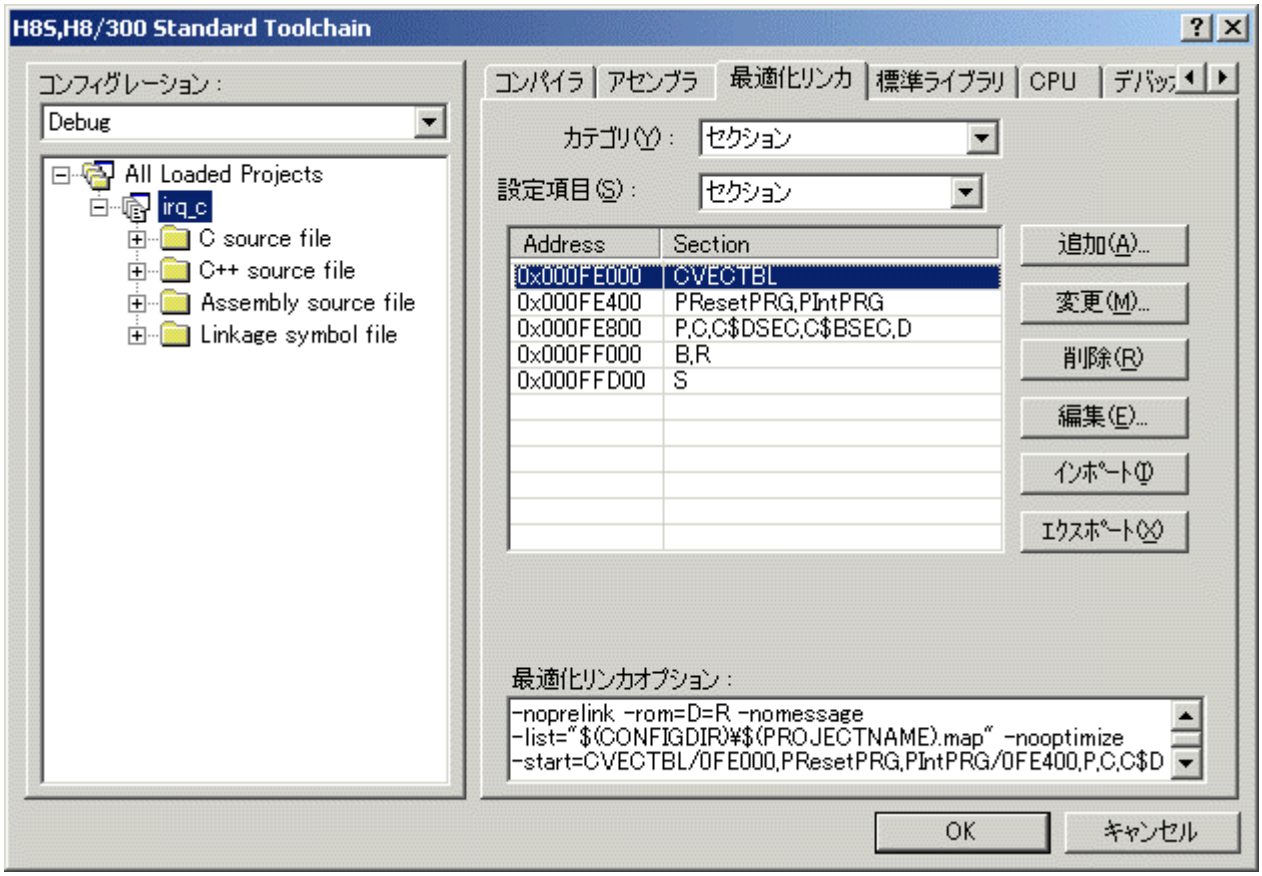
次は「intprg. c」です。第 5 章で説明したように，HEW が自動生成した「intprg. c」だと「Hterm」で実行することができません。それで，HEW が生成した「intprg. c」を巻末の付録と置き換え，内容を次のように変更します。

```
                ∫
extern void PowerON_Reset(void);
extern void intprog_imia4(void);

                ∫
void INT_IMIA4(void){intprog_imia4();}
                ∫
```

追加

変更

ビルドする前に下記のように 0xFE000 番地に「CVECTBL」セクションを追加します。



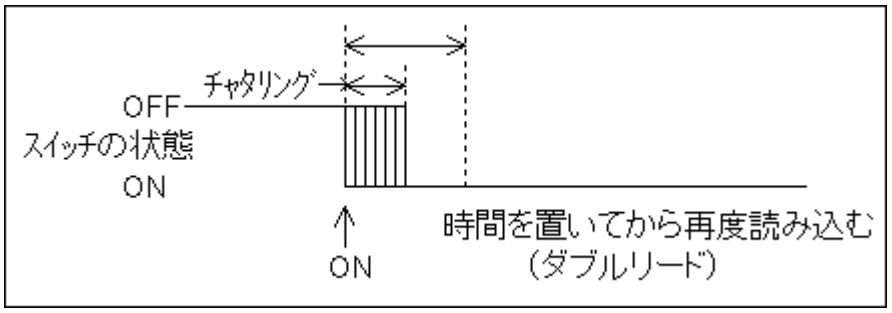
これで準備は整ったのでビルドして実行します。なお，付属の CD-ROM にはあらかじめダウンロードするファイルがおさめられています。「itu01_led. abs」をダウンロードして実行してください。

今回は点滅以外何もすることがないので，メインループでは何もしていません。実際には，ここにほかの作業を加えていくことになります。

4. スイッチの入力

第5章の最後で,「コンデンサを追加せずに, なおかつ他の処理に影響を与えずにプログラムでチャタリングを除去する方法もあります。これについては次の章をご覧ください。」と述べました。スイッチの入力にタイマを組み合わせると, 他の処理に影響を与えずにチャタリングをプログラムだけで除去することができます。ここでその方法を考えてみましょう。

第4章でチャタリング除去について考えてみました。チャタリング除去の基本的な考え方は, 一定間隔で2回入力して, 一致したら採用する, というものです。この一定間隔をタイマで作り, 待っている間ほかの作業を行ないます。



割り込み間隔は前項の LED 点滅と同じく, ITU のチャネル 4 を使って 1ms に設定します。それで, ITU のイニシャライズや「intprg. c」はそのまま使えます。ただ 1ms は少し短いので, 割り込みルーチンでカウントし, 10ms 間隔でスイッチを入力するようにします。

1 回目(SwStatus＝0)の入力データは「SwData1」にストアします。10ms 後に 2 回目(SwStatus＝1)の入力を行ないますが, このデータと「SwData1」を比較し一致したら「SwData2」にストアします。一致しないときはチャタリングと判断し「SwData2」は更新しません。それで,「SwData2」を見ればチャタリングが除去されたスイッチの状態を知ることができます。

さて,「スイッチが押されている間処理する」というのとともにスイッチ入力でよくあるのは「スイッチが押された瞬間に処理する」というものです。今回作成するプログラムは「SwData3」に前回のチャタリングが除去されたスイッチデータをストアし, チャタリングが除去された「SwData2」と比較することで, オフからオンに変化したスイッチを検知します。オフからオンに変化したスイッチのデータを「SwData4」にストアします。それで,「SwData4」を見ればスイッチがオンされた瞬間を知ることができます。なお,「SwData4」はオフからオンになったときだけデータを更新するので,「SwData4」で判断したらクリアする必要があります。スイッチ入力関数は次のようになります。

```
/*************************************************************************
    スイッチ入力
*************************************************************************/
void switch_in(void)
{
    #define   SW_CNT_CONST 10

    SwCount++;
    if (SwCount<SW_CNT_CONST)    {return;}
    else                         {SwCount = 0;}

    switch(SwStatus){
        case 0:
            SwData1 = ~P2.DR.BYTE & _00111000B;
            if (SwData1!=0)      {SwStatus = 1;}
            else                 {SwData2 = SwData3 =0;}
            break;
        case 1:
            if (SwData1==(~P2.DR.BYTE & _00111000B)){
                SwData2 = SwData1;
                SwData4 = SwData4 | (SwData2 & (~SwData3));
                SwData3 = SwData2;
            }
```

```
            SwStatus = 0;
            break;
    }
}
```

今回のプログラムは「SwData4」を利用してオフからオンに変化した瞬間を検知しています。この使い方はメインルーチンをご覧ください。

```
/************************************************************************
    メインプログラム
************************************************************************/
void main(void)
{
    ini_io();
    ini_itu();

    while(1){
        if      ((SwData4 & _00001000B)==_00001000B){   //SW1が押された
            DispData++;                                 //インクリメント
            SwData4 = 0;
        }
        else if ((SwData4 & _00010000B)==_00010000B){   //SW2が押された
            DispData--;                                 //デクリメント
            SwData4 = 0;
        }
        else if ((SwData4 & _00100000B)==_00100000B){   //SW3が押された
            DispData = rotlc(1,DispData);               //左ローテート
            SwData4 = 0;
        }

        PA.DR.BYTE = DispData;  // ポートAに表示する
    }
}
```

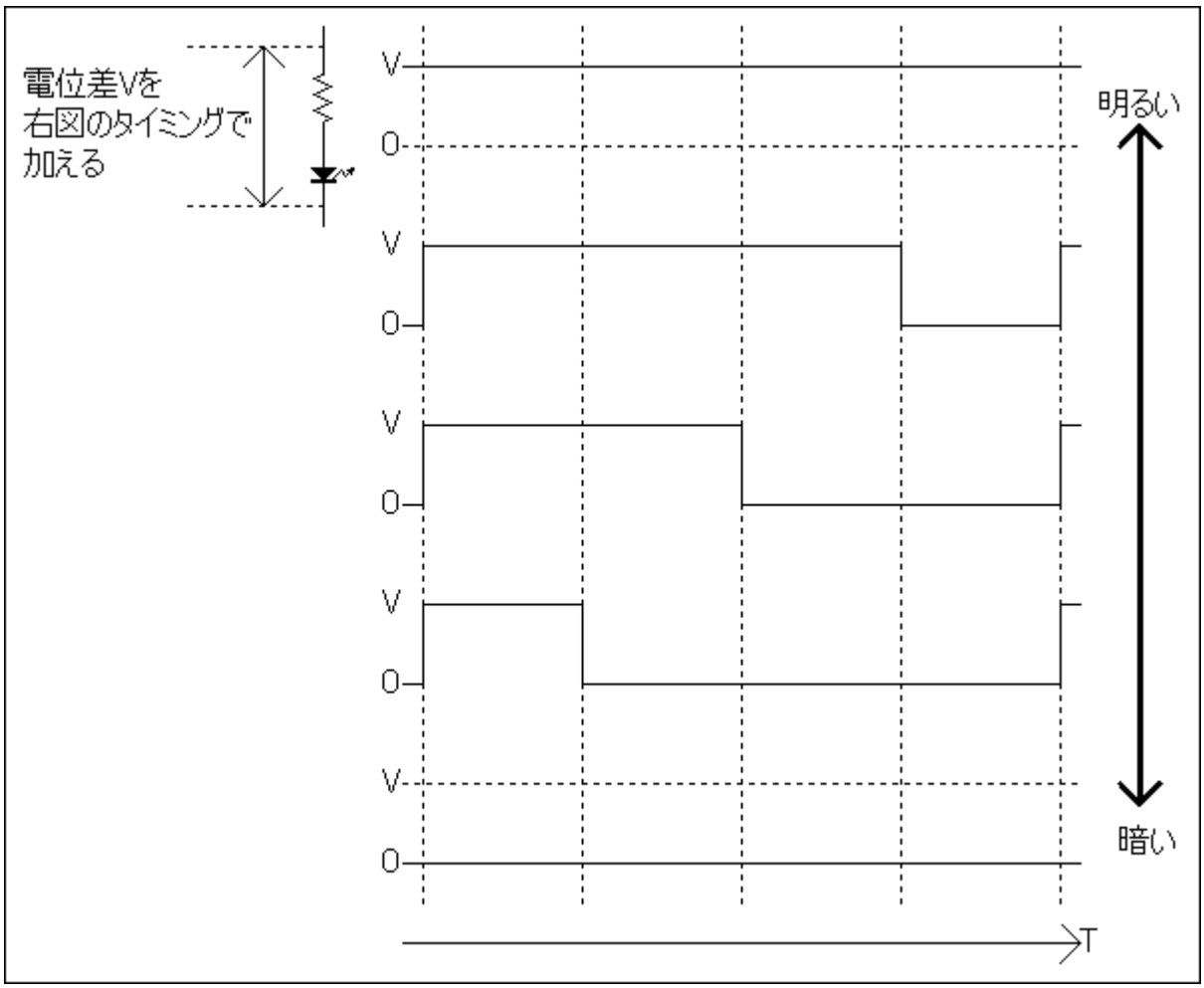
「intprg. c」は前項の LED の点滅と同じです。また, 0xFE000 番地に「CVECTBL」セクションを追加するのも同じです。

これで準備は整ったのでビルドして実行します。なお, 付属の CD-ROM にはあらかじめダウンロードするファイルがおさめられています。「itu02_sw. abs」をダウンロードして実行してください。

5. PWM

今までは LED を単純に点滅させるプログラムを作ってきました。今回は，LED の明るさを徐々に変化させて，「じわ～」っと点滅するプログラムを作ってみましょう。

LED の明るさなどパワーをデジタル回路で制御するのによく使われる方法の一つは，PWM という制御方法です。次の図をご覧ください。



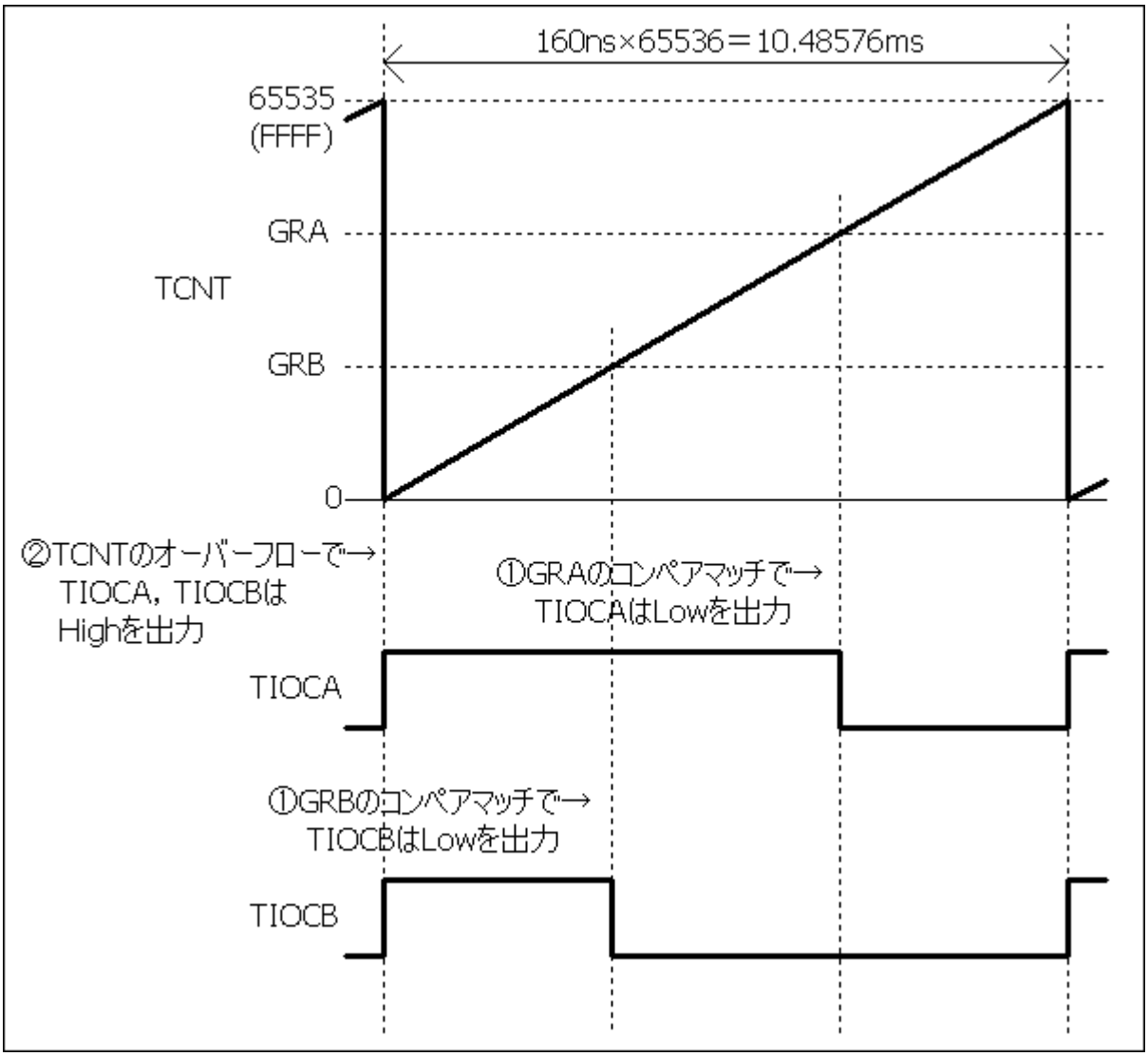
このように，一定の周期の中で電圧を加える時間の割合（デューティー）を変化させることで，平均電力を制御することができます。このような方法を PWM（Pulse Width Modulation）と呼びます。

図からわかるように，厳密に言えば LED は点灯と消灯を繰り返しています。しかし，人間の目には残像現象があるため，周期を十分早くすれば（10ms～20ms ぐらい），点滅を感じることなく LED の明るさの変化として感じるようになります。

PWM は LED の明るさの制御だけではなく，DC モータの回転制御や白熱電灯の調光制御（位相制御）など，電力制御によく使われています。

◆

では，ITU で PWM の信号を作ることを考えてみましょう。1 チャネルにつき 2 種類の値（GRA，GRB）と比較することができ（コンペアマッチ機能），その値と一致したら出力（TIOCA，TIOCB）を変化させます。次の図をご覧ください。



斜めの線がタイマによって+1 されるカウンタ（TCNT）の値，GRA，GRB が TCNT と比較する値をセットするレジスタです。TIOCA，TIOCB はコンペアマッチによる出力端子です。これらが各チャネルに 1 セット用意されています。なお，TIOCA，TIOCB は I/O ポートと兼用ピンになっています。この教材では LED のつながっているピンになっているので，そのまま LED を点灯することができます。さて，ITU は次のように動作します。

① クロックによって TCNT がどんどん+1 されていくと，そのうち GRA と一致します（コンペアマッチ）。一致したら TIOCA を Low にします。同じように，GRB と一致したら TIOCB を Low にします。

② クロックによって+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。この状態をオーバフローと呼びます。オーバフローしたら TIOCA，TIOCB を High にします。

この①②を繰り返すことで，一定の間隔でパルスを出力することができます。ITU の TCNT は CPU クロックの 1/4 で+1 するように設定します（25MHz÷4＝6.25MHz，つまり 160ns 毎に+1 する）。16 ビットカウンタということは，オーバフローするまでに 65536 回カウントするので，160ns×65536＝10.48576ms がパルス周期になります。明るさに応じた値を GRA，GRB にセットします。

ところで，①の TIOCA，TIOCB を Low にするのは ITU が自動的に行なってくれるのですが，②の TIOCA，TIOCB を High にするのは自動的に行なってくれません。しかも，TIOCA，TIOCB を直接 High や Low に制御することができません。

そこで，TCNT のオーバフローで割り込みをかけます。そして，割り込みの中で「GRA=1，GRB=1，コンペアマッチで High を出力」に設定ます。すると，すぐにコンペアマッチになり TIOCA，TIOCB は High になります。その後，あらためて ITU を再スタートします。

今回は明るさを徐々に変化させようとしているのですが，明るさは GRA，GRB の設定値で決まります。数値が大きいほど明るくなります。タイミングチャートからわかるように TCNT の周期より短い間隔で明るさを変えても意味がありません。それで，TCNT のオーバフロー割り込みの中で次の値を計算しセットします。

以上のことを踏まえた上でプログラムを考えてみましょう。まずは 1 個の LED の明るさを変化させます。ITU のチャネル 0 の TIOCA(=TIOCA0=PA2)につながっている LED です。イニシャライズ関数と割り込み関数のソースリストは次のようになります。

```
/************************************************************************
     ITUの初期化
************************************************************************/
void ini_itu(void)
{
     ITU.TSTR.BYTE      =    _11100000B;    //タイマ停止

     ITU0.TCR.BYTE      =    _10000010B;    //TCNTクリア禁止
                                            //立ち上がりエッジでカウント
                                            //内部クロックΦ/4でカウント
     ITU0.TIOR.BYTE     =    _10001001B;    //GRAのコンペアマッチで0出力
     ITU0.TCNT          =    0;             //TCNTクリア
     ITU0.GRA           =    GRA0Const;     //GRA
     ITU0.TIER.BYTE     =    _11111100B;    //オーバフロー割り込みイネーブル
     ITU0.TSR.BYTE      =    _11111000B;    //タイマステータスレジスタクリア

     ITU.TSTR.BYTE      =    _11100001B;    //タイマスタート
}

/************************************************************************
     ITU0 TCNT0オーバフロー 割込み
************************************************************************/
#pragma regsave (intprog_ovi0)
void intprog_ovi0(void)
{
     ITU.TSTR.BYTE      =    _11100000B;    //タイマ停止
     ITU0.TCR.BYTE      =    _10000000B;    //TCNTクリア禁止
                                            //立ち上がりエッジでカウント
                                            //内部クロックΦでカウント
     ITU0.TIOR.BYTE     =    _10001010B;    //GRAのコンペアマッチで1出力
     ITU0.TCNT          =    0;             //TCNTクリア
     ITU0.GRA           =    1;             //GRA
     ITU0.TIER.BYTE     =    _11111000B;    //割り込みディセーブル
     ITU0.TSR.BYTE      =    _11111000B;    //タイマステータスレジスタクリア
     ITU.TSTR.BYTE      =    _11100001B;    //タイマスタート

     while(ITU0.TSR.BIT.IMFA==0){}          //出力が1になるのを待つ

     next_GR();                             //次のGA値を計算
     ini_itu();                             //再スタート
}

/************************************************************************
     次のGR値（パルス長）を計算
************************************************************************/
void next_GR(void)
{
```

```
    if(GRA0Flag==0){
        GRA0Const = GRA0Const + 0x0100;
        if(GRA0Const>=0xff00){GRA0Flag = -1;}
    }
    else{
        GRA0Const = GRA0Const - 0x0100;
        if(GRA0Const<=0x0100){GRA0Flag = 0;}
    }
}
```

次は「intprg. c」です。HEW が生成した「intprg. c」を巻末の付録と置き換え，次のように変更します。

```
                ∫
extern void PowerON_Reset(void);
extern void intprog_ovi0(void);

                ∫
void INT_OVI0(void)     {intprog_ovi0();}
                ∫
```

追加

変更

割り込みを使うので，これまでと同じく 0xFE000 番地に「CVECTBL」セクションを追加します。

これで準備は整ったのでビルドして実行します。なお，付属の CD-ROM にはあらかじめダウンロードするファイルがおさめられています。「itu03_pwm. abs」をダウンロードして実行してください。

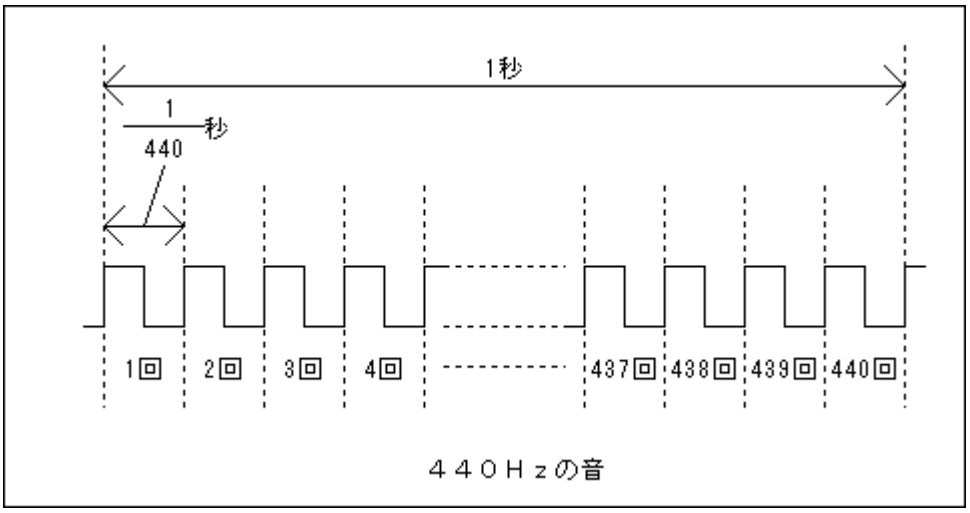
■ 練習問題

今のプログラムは1個のLEDだけ点滅させました。TIOCA0～TIOCB4全てのLED（全部で10個）を点滅させて下さい（TIOCA3，TIOCB3，TIOCA4，TIOCB4はタイマ＆LEDディスプレイキット上のLEDにつながっている，点灯の論理が逆になっていることに注意）。全てのLED を同じタイミング点滅させるのではなく，すこしずづ明るくなるタイミングをずらしてみると面白いと思います。（解答例は「itu04_pwm」）

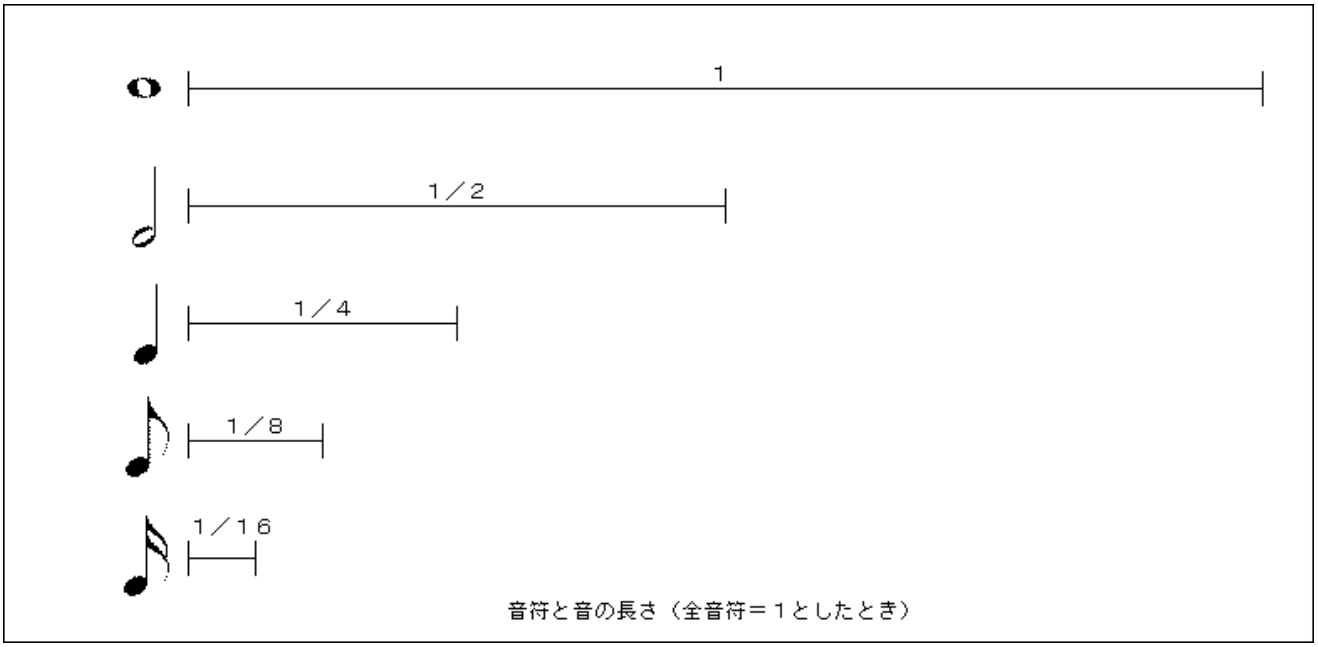
6. メロディの演奏

ITU の応用例としてメロディの演奏を考えてみましょう。

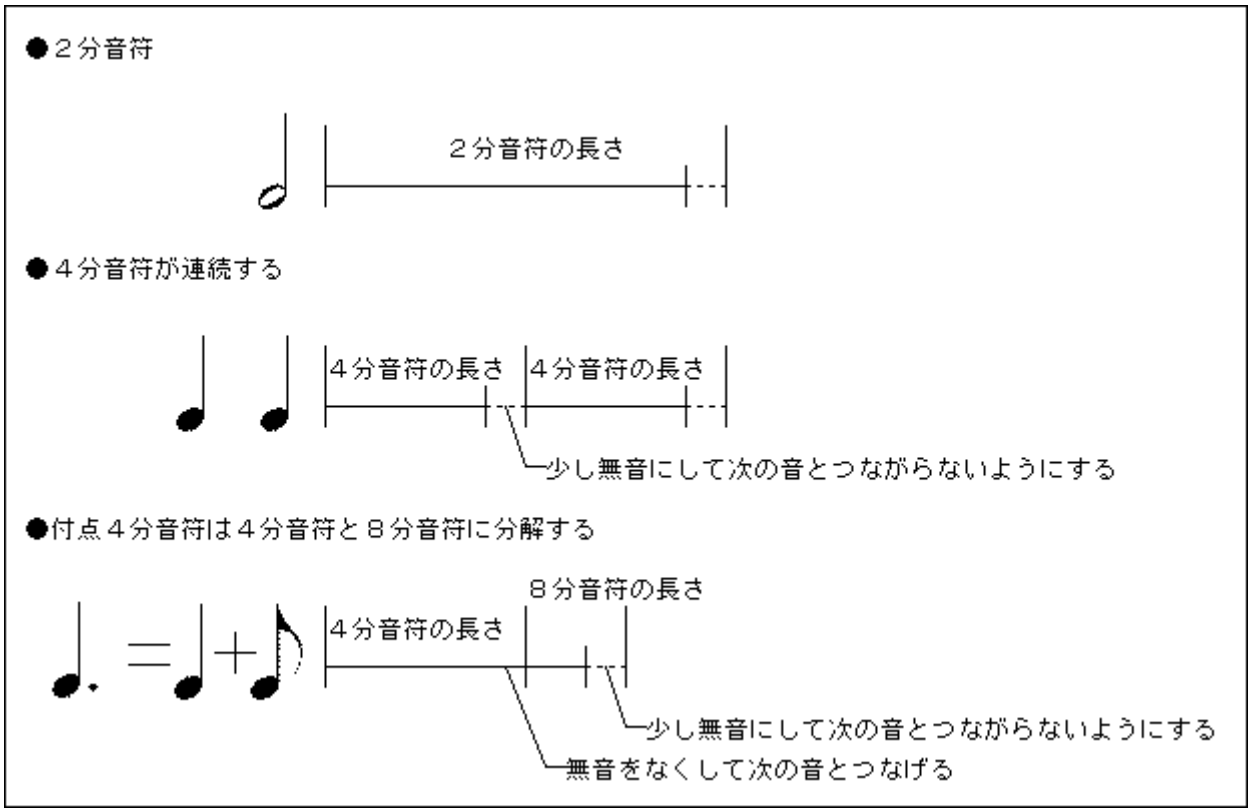
メロディにまず必要なのは，ドレミファソラシド，つまり音階です。音階は物理的には音の周波数のことで，1 秒間に何回くりかえすか，ということです（単位は Hz：ヘルツ）。このプログラムの基準音はラ（A）ですが，周波数は 440Hz になります。今回使ったサウンダという部品は，ある周波数のパルス信号を加えると，その周波数の音を出します。というわけで，出したい音の周波数のパルス信号を P22 から出力することで，特定の音階の音を出しています。あとは，周波数をいろいろ変えればメロディになっていきます。例えば，440Hz の音を出すときには右の図のように P22 からパルス信号を出力します。



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（つまり 1 個の 4 分音符の長さは 60 秒÷120=0.5 秒）。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。

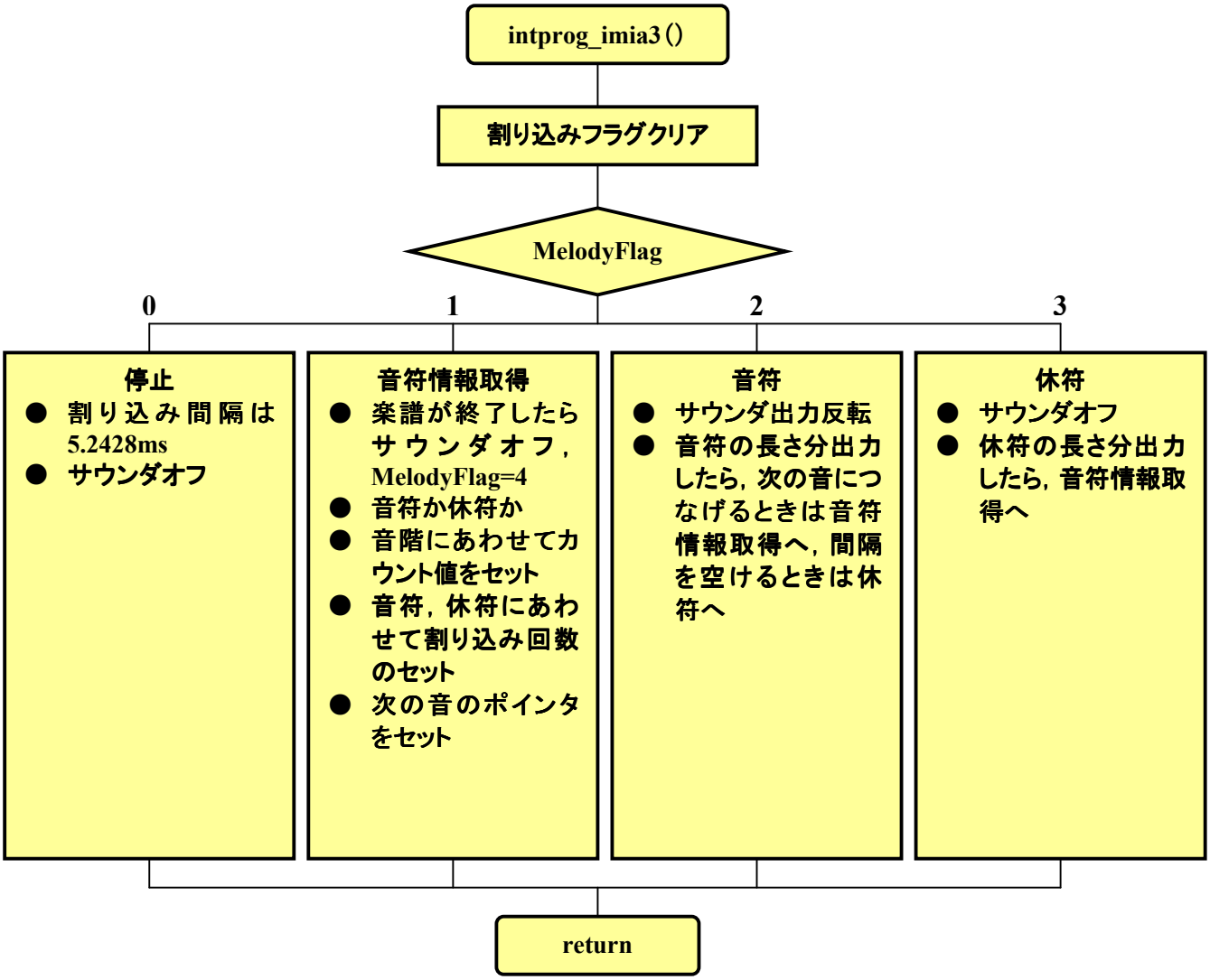


音符の長さでもう一つ重要なのは，音符の長さの全部で音を出すか，ということです。例えば，同じ音階の4分音符が2つ並んでいるのと，2分音符との違いです。トータルの音の長さ(1秒)は同じですが，4分音符が2つのときは明らかに二つの音です。音符の長さの全部で音を出してしまうと2つの音がつながってしまって一つの音のように聞こえてしまいます。それで，音符の長さのうち，16分の15音を出して，最後の16分の1は無音にします。ただ，スラーやタイのときは次の音とつなげたいので，そのときは音符の長さの全部で音を出すようにします。また，付点音符は二つの音で指定します。例えば付点4分音符は，4分音符と8分音符の二つの音として扱います。そして，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，ITU のチャネル 3，コンペアマッチ A 割込みの間隔を調整し，メロディを奏でていきます。詳しくは概略フローチャート(ITU3コンペアマッチA割り込み‘intprog_imia3()’ルーチン)とソースリストをご覧ください。(次ページ)




```
                ∫
/*****************************************************************************
    定数の定義（直接指定）
*****************************************************************************/
//メロディ ------------------------------------------------------------
#define   KIJUN_ON  0x0c //基準音が音階テーブルの配列の何番目の要素になるか

/*****************************************************************************
    グローバル変数の定義とイニシャライズ(RAM)
*****************************************************************************/
// メロディに関係した変数 ---------------------------------------------
unsigned char  MelodyFlag     =    0;   //メロディフラグ
                                        // 0:停止
                                        // 1:音楽スタート
                                        // 2:音符
                                        // 3:休符
                                        // 4:音楽終了
unsigned int   OnpuCnt;                 //音符カウンタ
unsigned int   KyufuCnt;                //休符カウンタ
unsigned int   *GakufuPnt;              //楽譜ポインタ
```

```
/*****************************************************************************
     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分音符,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,付点音符)
          下位8ビット(bit7-0) : 音階
               基準音=80hとした相対値。ただし00hは休符。
*****************************************************************************/
//「小さな世界」  ------------------------------------------------------
const unsigned int Gakufu_0[] = {
     0x037b,0x037c,0x027e,0x0287,0x0283,
     0x0385,0x0383,0x0283,0x0282,0x0282,

     0x0379,0x037b,0x027c,0x0285,0x0282,
     0x0383,0x0382,0x0280,0x027e,0x027e,

     0x037b,0x037c,0x027e,0x0383,0x0385,0x0287,
     0x0385,0x0383,0x0280,0x0385,0x0387,0x0288,
     0x0387,0x0385,0x027e,0x0288,0x0287,0x0285,0x0183,0x0200,

     0xffff    //テーブル終了マーク
};

                ∫
/*****************************************************************************
     音階テーブル
     {(ITU チャネル3のGRAにセットする値),(全音符の長さ,割込回数)}
*****************************************************************************/
const unsigned int OnkaiTbl[][2] = {
     {28409, 882},       //A  ,ラ    ,220.00Hz
     {28615, 934},       //A# ,ラ#   ,233.08Hz
     {25310, 988},       //B  ,シ    ,246.94Hz

     {23889,1048},       //C  ,ド    ,261.63Hz
     {22549,1110},       //C# ,ド#   ,277.18Hz
     {21283,1176},       //D  ,レ    ,293.66Hz
     {20088,1246},       //D# ,レ#   ,311.13Hz
     {18961,1320},       //E  ,ミ    ,329.63Hz
     {17897,1398},       //F  ,ファ  ,349.23Hz
     {16892,1480},       //F# ,ファ#,369.99Hz
     {15944,1570},       //G  ,ソ    ,392.00Hz
     {15049,1662},       //G# ,ソ#   ,415.30Hz
     {14205,1760},       //A  ,ラ    ,440.00Hz          //←基準音
     {13407,1866},       //A# ,ラ#   ,466.16Hz
     {12655,1976},       //B  ,シ    ,493.88Hz

     {11945,2094},       //C  ,ド    ,523.25Hz
     {11274,2218},       //C# ,ド#   ,554.37Hz
     {10641,2350},       //D  ,レ    ,587.33Hz
     {10044,2490},       //D# ,レ#   ,622.25Hz
     { 9480,2638},       //E  ,ミ    ,659.26Hz
```

```
    { 8948,2794},       //F  ,ファ ,698.46Hz
    { 8446,2960},       //F# ,ファ#,739.99Hz
    { 7972,3136},       //G  ,ソ   ,783.99Hz
    { 7525,3324},       //G# ,ソ#  ,830.61Hz
    { 7102,3520},       //A  ,ラ   ,880.00Hz
    { 6704,3730},       //A# ,ラ#  ,932.33Hz
    { 6327,3952},       //B  ,シ   ,987.77Hz
};

/************************************************************************
    ITUの初期化
************************************************************************/
void ini_itu(void)
{
    ITU.TSTR.BYTE       =   _11100000B;   //タイマ停止

                                          //チャネル3
    ITU3.TCR.BYTE       =   _10100001B;   //コンペアマッチAでTCNTクリア
                                          //立ち上がりエッジでカウント
                                          //内部クロックΦ/2でカウント
    ITU3.TIOR.BYTE      =   _10001000B;   //コンペアマッチによる出力禁止
    ITU3.TCNT           =   0;            //TCNTクリア
    ITU3.GRA            =   0xffff;       //GRA
    ITU3.TIER.BYTE      =   _11111001B;   //コンペアマッチAによる割り込みイネーブル
    ITU3.TSR.BYTE       =   _11111000B;   //タイマステータスレジスタクリア

    ITU.TSTR.BYTE       =   _11101000B;   //タイマスタート
}

/************************************************************************
    ITU3 コンペアマッチA 割込み
************************************************************************/
#pragma regsave (intprog_imia3)
void intprog_imia3(void)
{
    unsigned char  Onkai;
    unsigned char  Onpu;
    unsigned char  Kyufu = 0;

    //ITU3 チャネルA コンペアマッチインタラプトフラグ クリア
    ITU3.TSR.BIT.IMFA =0;

    //メロディ
    switch (MelodyFlag){
        //停止 ----------------------------------------------------------
        case 0:
            ITU3.GRA = 0xffff; //メロディなしのときは5.2428msで割込みをかける
            P2.DR.BIT.B2 = 1;  //サウンダオフ
            break;
        //音符情報取得 --------------------------------------------------
        case 1:
            if (*GakufuPnt==0xffff){     //楽譜終了
                P2.DR.BIT.B2 = 1;  //サウンダオフ
                MelodyFlag = 4;
                break;
```

```
            }

            Onkai = (unsigned char)(*GakufuPnt & 0x00ff);   //音階
            Onpu  = (unsigned char)(*GakufuPnt / 0x0100);   //音符の長さ
            if (Onkai==0) {Onkai = 0x80; Kyufu = 1;}    //休符

            //音階にあわせてカウント値をセットする,割込み間隔の調整
            ITU.TSTR.BIT.STR3 = 0;
            ITU3.GRA = OnkaiTbl[Onkai - (0x80 - KIJUN_ON)][0];
            ITU3.TCNT = 0x0000;
            ITU.TSTR.BIT.STR3 = 1;

            OnpuCnt = OnkaiTbl[Onkai - (0x80 - KIJUN_ON)][1];    //全音符の長さ
            if ((Onpu & 0x7f)!=0)   {OnpuCnt = OnpuCnt >> (Onpu & 0x7f);} //音符の長さを決定
            if (Kyufu==0){//音符
                if ((Onpu&0x80)==0){    //次の音と間隔を開ける
                    KyufuCnt = OnpuCnt / 16;
                    OnpuCnt = OnpuCnt - KyufuCnt;
                }
                else{    //次の音とつなげる
                    KyufuCnt = 0;
                }
                MelodyFlag = 2;
            }
            else{    //休符
                KyufuCnt = OnpuCnt;
                OnpuCnt = 0;
                MelodyFlag = 3;
            }

            GakufuPnt++;  //次の音符のポインタに
            break;
        //音符 ------------------------------------------------------
        case 2:
            P2.DR.BIT.B2 = ~P2.DR.BIT.B2;     //サウンダ出力反転
            OnpuCnt--;
            if (OnpuCnt==0){
                if (KyufuCnt==0)    {MelodyFlag = 1;}  //次の音につなげる
                else                {MelodyFlag = 3;}  //間隔を空ける
            }
            break;
        //休符 ------------------------------------------------------
        case 3:
            P2.DR.BIT.B2 = 1;  //サウンダオフ
            KyufuCnt--;
            if (KyufuCnt==0)    {MelodyFlag = 1;}  //終了したら次の音符に移る
            break;
    }
}
```

次は「intprg. c」です。HEW が生成した「intprg. c」を巻末の付録と置き換え，次のように変更します。

```
                ∫
extern void PowerON_Reset(void);
extern void intprog_imia3(void);

                ∫
void INT_IMIA3(void)     {intprog_imia3();}
                ∫
```

追加

変更

割り込みを使うので，これまでと同じく 0xFE000 番地に「CVECTBL」セクションを追加します。

これで準備は整ったのでビルドして実行します。なお，付属の CD-ROM にはあらかじめダウンロードするファイルがおさめられています。「itu05_melody. abs」をダウンロードして実行してください。

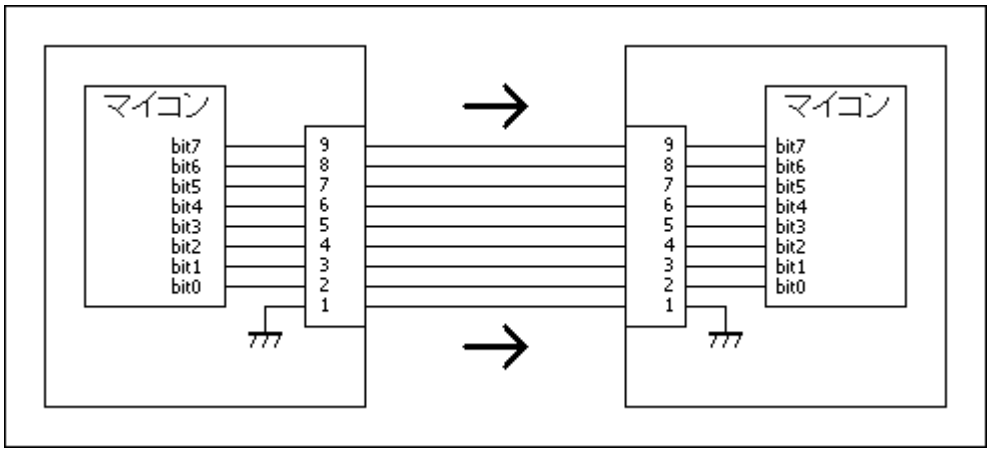
第7章 シリアルコミュニケーションインターフェース（SCI）の使い方

最近のパソコン事情に詳しい方はおわかりのように，外部機器との接続はシリアル通信方式が主流になっています。ハードディスクも ATA（パラレル）より SATA（シリアル）が主流です。当然，マイコンの世界でもシリアル通信はよく使われていますし，H8/3052 にもシリアル通信機能が内蔵されています。そこで，この章ではシリアル通信の基本的な考え方・使い方を調べてみましょう。

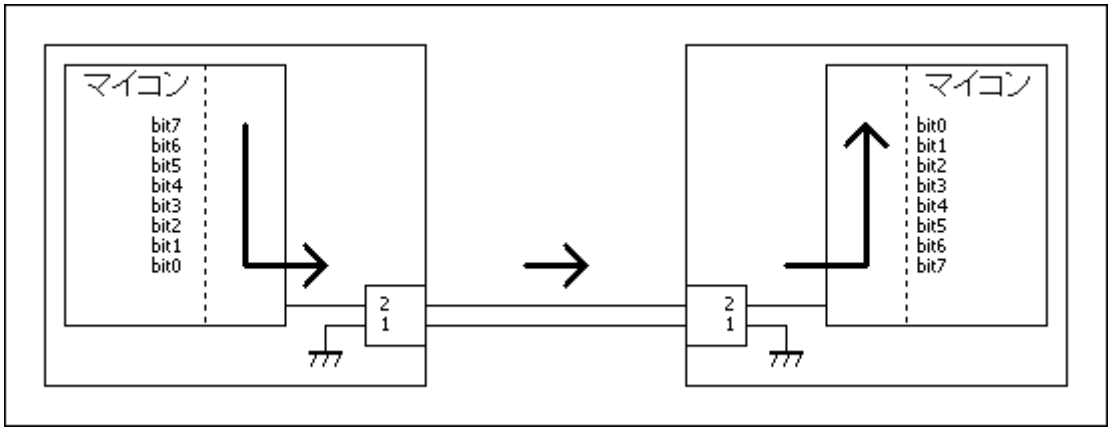
1. 調歩同期式シリアル通信

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。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。



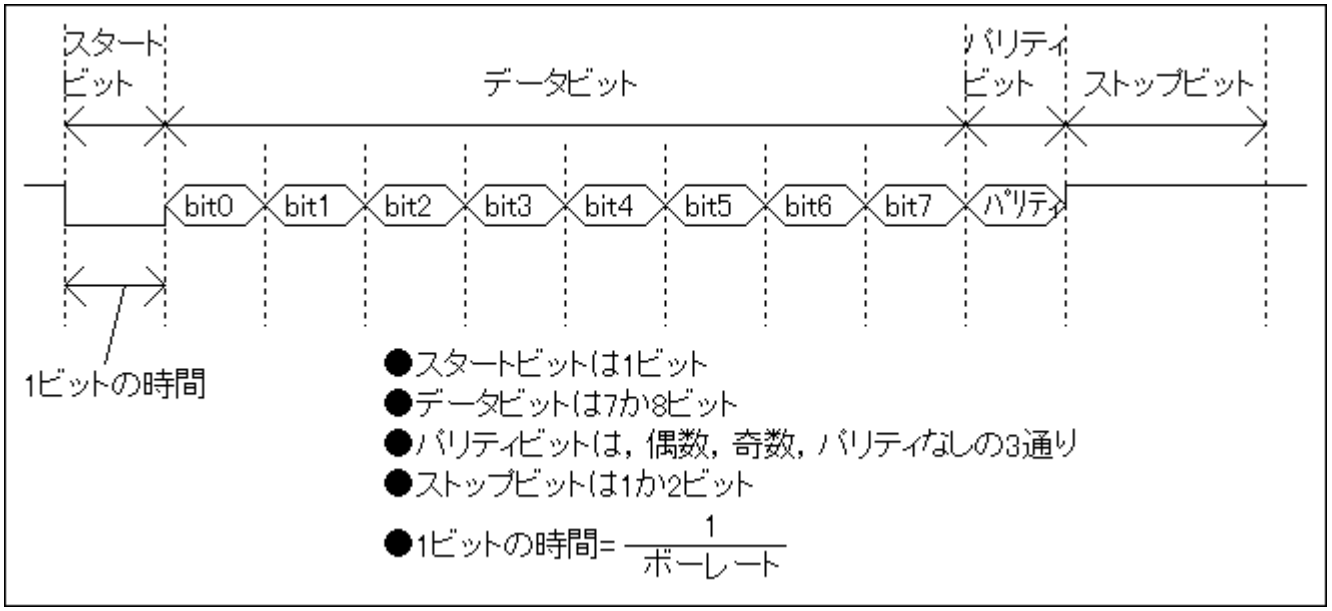
さて，これを，信号 1 本と GND1 本，合計 2 本の線だけでデータを伝えることはできないでしょうか。信号が1本しかないのですから1ビットずつ順番に送るしか方法がありません。この発想から生まれた方法がシリアル通信です。



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。

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だろうか，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。



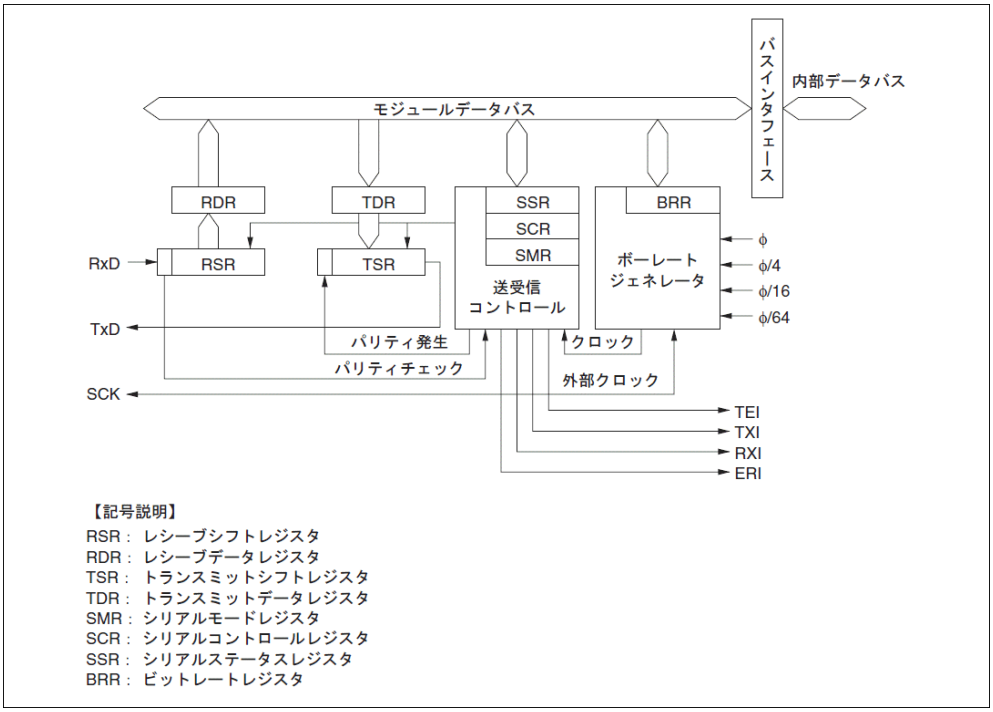
かぎとなるのは，1ビットの時間が決まっていることと，信号線は通常はHigh（5V）で，スタートビットで必ずLow（0V）になるということです。

通信速度は 1 秒あたりのビット数で表わし，これをボーレート（単位：bps，bit/s，またはボー）と呼びます。例えば，Hterm のボーレートは 38400 ボーですが，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。

2. SCI とは何か

調歩同期式シリアル通信の考え方はわかりましたが，これを I/O ポートとプログラムだけで作ろうとすると，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ないので大変です。

それで，H8/3052 にはシリアル通信用の I/O が 2 チャネル内蔵されています。「シリアルコミュニケーションインタフェース（SCI）」と呼ばれています。SCI 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，ハードウェアマニュアルの「13．シリアルコミュニケーションインターフェース（SCI）」で説明されていますのでお読みください。



3. 設定用レジスタ

SCI はチャネル 0 と 1 の 2 チャネル用意されていますが，完全に独立して動作し，設定用レジスタもチャネル毎に用意されています（アドレスは前から順番にチャネル 0，チャネル 1）。

シリアルモードレジスタ（SMR） ： アドレス=0xFFB0，0xFFB8 番地（下位 16 ビット）

ビット	ビット名	初期値	R/W	説明
7	C/A	0	R/W	コミュニケーションモード。 0：調歩同期式モード ／ 1：クロック同期式モード
6	CHR	0	R/W	キャラクタレングス。 0：8 ビットデータ ／ 1：7 ビットデータ
5	PE	0	R/W	パリティイネーブル。 0：パリティビットの付加，およびチェックを禁止。 1：パリティビットの付加，およびチェックを許可。
4	O/E	0	R/W	パリティモード。 0：偶数パリティ ／ 1：奇数パリティ
3	STOP	0	R/W	ストップビットレングス 0：1 ストップビット ／ 1：2 ストップビット
2	MP	0	R/W	マルチプロセッサモード。 0：マルチプロセッサ機能の禁止。 1：マルチプロセッサフォーマットを選択。
1	CKS1	0	R/W	クロックセレクト。 00：Φクロック 01：Φ／4 クロック 10：Φ／16 クロック 11：Φ／64 クロック
0	CKS0	0	R/W	

ビットレートレジスタ（BRR） ： アドレス=0xFFB1，0xFFB9 番地（下位 16 ビット）

ビット	ビット名	初期値	R/W	説明
7		1	R/W	SMR の CKS1，0 と組み合わせてボーレートを設定するレジスタ。 CKS1，0=11，10，01，00 のとき n=3，2，1，0 Φ=CPU クロック，B=ボーレートとすると， ［調歩同期式モード］ $BRR = \frac{\Phi}{64 \times 2^{2n-1} \times B} \times 10^6 - 1$ ［クロック同期式モード］ $BRR = \frac{\Phi}{8 \times 2^{2n-1} \times B} \times 10^6 - 1$
6		1	R/W	
5		1	R/W	
4		1	R/W	
3		1	R/W	
2		1	R/W	
1		1	R/W	
0		1	R/W	

シリアルコントロールレジスタ(SCR) : アドレス=0xFFB2, 0xFFBA 番地(下位 16 ビット)				
ビット	ビット名	初期値	R/W	説明
7	TIE	0	R/W	トランスミットインタラプトイネーブル。 0:送信データエンプティ割り込み(TXI)要求の禁止。 1:送信データエンプティ割り込み(TXI)要求の許可。
6	RIE	0	R/W	レシーブインタラプトイネーブル。 0:受信データフル割り込み(RXI)要求, および受信エラー割り込み(ERI)要求を禁止。 1:受信データフル割り込み(RXI)要求, および受信エラー割り込み(ERI)要求を許可。
5	TE	0	R/W	トランスミットイネーブル。 0:送信動作を禁止 / 1:送信動作を許可
4	RE	0	R/W	レシーブイネーブル。 0:受信動作を禁止 / 1:受信動作を許可
3	MPIE	0	R/W	マルチプロセッサインタラプトイネーブル。 0:マルチプロセッサ割り込み禁止状態。 [クリア条件](1)MPIEビットを0にクリア。(2)MPB=1 のデータを受信したとき。 1:マルチプロセッサ割り込み許可状態。 マルチプロセッサビットが 1 のデータを受け取るまで, 受信割り込み要求(RXI), 受信エラー割り込み要求(ERI), SSR の RDRF, FER, ORER のフラグセットの禁止。
2	TEIE	0	R/W	トランスミットエンドインタラプトイネーブル。 0:送信終了割り込み(TEI)を禁止。 1:送信終了割り込み(TEI)を許可。
1	CKE1	0	R/W	クロックイネーブル。
0	CKE0	0	R/W	[調歩同期式モード] 00:内部クロック/SCK 端子は入出力ポート。 01:内部クロック/SCK 端子はクロック出力。 1-:外部クロック/SCK 端子はクロック入力。 [クロック同期式モード] 0-:内部クロック/SCK 端子は同期クロック出力。 1-:外部クロック/SCK 端子は同期クロック入力。

トランスミットデータレジスタ(TDR) : アドレス=0xFFB3, 0xFFBB 番地(下位 16 ビット)				
ビット	ビット名	初期値	R/W	説明
7		1	R/W	シリアル送信データ。
6		1	R/W	
5		1	R/W	
4		1	R/W	
3		1	R/W	
2		1	R/W	
1		1	R/W	
0		1	R/W	

シリアルステータスレジスタ(SSR) ： アドレス=0xFFB4, 0xFFBC 番地(下位 16 ビット)				
ビット	ビット名	初期値	R/W	説明
7	TDRE	1	R/W	トランスミットデータレジスタエンプティ。 0:TDR に有効な送信データがライトされていることを表示。 [クリア条件](1)TDRE=1 の状態をリードした後, 0 をライトしたとき。 (2)DMAC で TDR へデータをライトしたとき。 1:TDR に有効な送信データがないことを表示。 [セット条件](1)リセット, またはスタンバイモード時。(2)SCR の TE ビットが 0 のとき。(3)TDR から TSR にデータ転送が行なわれ, TDR にデータライトが可能になったとき。
6	RDRF	0	R/W	レシーブデータレジスタフル。 0:RDR に受信データが格納されていないことを表示。 [クリア条件](1)リセット, またはスタンバイモード時。(2)RDRF=1 の状態をリードした後, 0をライトしたとき。(3)DMAC で RDR のデータをリードしたとき。 1:RDR に受信データが格納されていることを表示。 [セット条件]シリアル受信が正常終了し, RSR から RDR へ受信データが転送されたとき。
5	ORER	0	R/W	オーバランエラー。 0:受信中, または正常に受信を完了したことを表示。 [クリア条件](1)リセット, またはスタンバイモード時。(2)ORER=1 の状態をリードした後, 0 をライトしたとき。 1:受信時にオーバランエラーが発生したことを表示。 [セット条件]RDRF=1 の状態で次のシリアル受信を完了したとき。
4	FER	0	R/W	フレーミングエラー。 0:受信中, または正常に受信を完了したことを表示。 [クリア条件](1)リセット, またはスタンバイモード時。(2)FER=1 の状態をリードした後, 0 をライトしたとき。 1:受信時にフレーミングエラーが発生したことを表示。 [セット条件]SCI が受信終了時に受信データの最後尾のストップビットが 1 であるかどうかをチェックし, ストップビットが 0 であったとき。
3	PER	0	R/W	パリティエラー。 0:受信中, または正常に受信を完了したことを表示。 [クリア条件](1)リセット, またはスタンバイモード時。(2)PER=1 の状態をリードした後, 0 をライトしたとき。 1:受信時にパリティエラーが発生したことを表示。 [セット条件]受信時の受信データとパリティビットをあわせた 1 の数が, SMR の O/E ビットで指定した偶数パリティ/奇数パリティの設定と一致しなかったとき。
2	TEND	1	R/W	トランスミットエンド。 0:送信中であることを表示。 [クリア条件](1)TDRE=1 の状態をリードした後, TDRE フラグに 0 をライトしたとき。(2)DMAC で TDR へデータをライトしたとき。 1:送信を終了したことを表示。 [セット条件](1)リセット, またはスタンバイモード時。(2)SCR の TE ビットが 0 のとき。(3)1 バイトのシリアル送信キャラクタの最後尾ビットの送信時に TDRE=1 であったとき。
1	MPB	0	R/W	マルチプロセッサビット。 0:マルチプロセッサビットが 0 のデータを受信したことを表示。 0:マルチプロセッサビットが 1 のデータを受信したことを表示。
0	MPBT	0	R/W	マルチプロセッサビットトランスファ。 0:マルチプロセッサビットが 0 のデータを送信。 0:マルチプロセッサビットが 1 のデータを送信。

レシーブデータレジスタ(RDR) ： アドレス=0xFFB5, 0xFFBD 番地(下位 16 ビット)				
ビット	ビット名	初期値	R/W	説明
7		0	R	受信したシリアルデータ。
6		0	R	
5		0	R	
4		0	R	
3		0	R	
2		0	R	
1		0	R	
0		0	R	

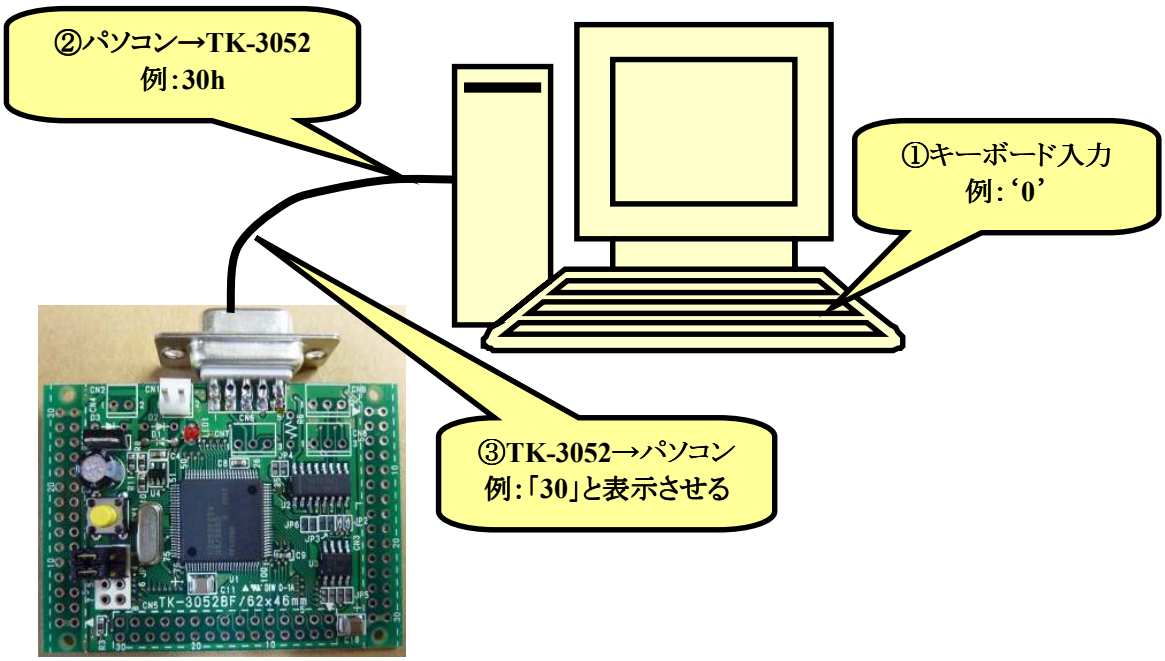
4. RS-232C のプログラム例

SCI を使ったプログラム例を考えてみましょう。

「Hterm」ではパソコンのシリアルポート(COM ポート)とTK-3052 のシリアルポートをつないで通信しています。このとき使っている通信規格は RS-232C と呼ばれる規格で昔から使われてきました。

ところで,「Hterm」で Console ウィンドウがアクティブのとき, パソコンのキーボードで入力すると, それに反応していろいろと表示されます。このとき, パソコンから TK-3052 にどんなデータが送信されているのでしょうか。

答えを言いますと, それぞれのキーに割り当てられている数値が送られています。そこで, どんな数値が送られてきたか, それを返信して Console ウィンドウに表示するプログラムを作ってみましょう。



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「Hterm」の Console ウィンドウに表示するので，「Hterm」と同じ条件になります。

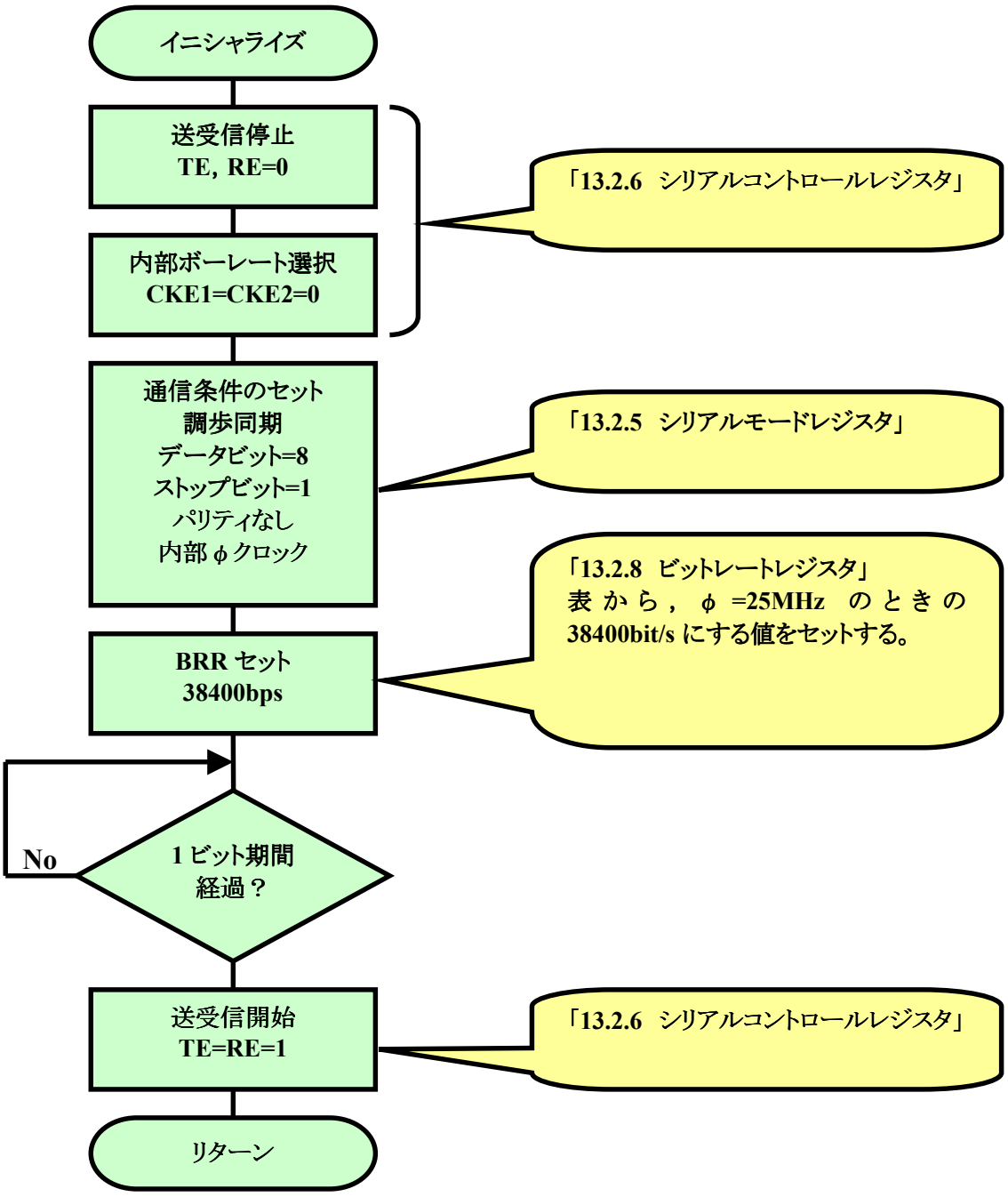
ビット/秒（ボーレート）・・・・・・・・・・・・・・・ 38400bps（38400bit/s，38400 ボー）

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 SCI にこの条件をセットします。ハードウェアマニュアルの「図 13．4　SCI の初期化フローチャートの例」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。

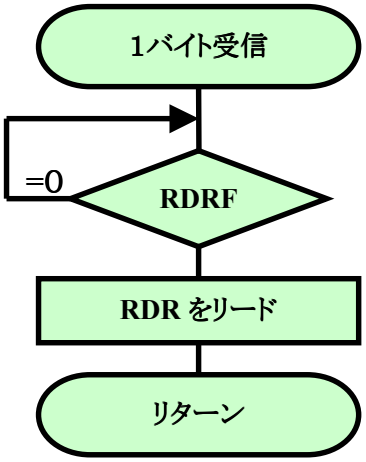


ソースリストは次のようになります。

```
/*****************************************************************************
    SCI1 イニシャライズ
*****************************************************************************/
void init_sci1(void)
{
    #define       CPU_CLK       25000000       // Clock=25MHz=25000000Hz
    #define       BAUD          38400          // baudrate
    #define       BITR          (CPU_CLK) / (BAUD*32) - 1
    #define       WAIT_1B       (CPU_CLK) / 6 / BAUD

    char i;

    SCI1.SCR.BYTE =    0x00;
    SCI1.SMR.BYTE =    0x00;     //調歩同期,8bit,NonParity,StopBit=1
    SCI1.BRR      =    BITR;     //38400Baud
    for  (i=0;i<WAIT_1B;i++){};  //1bit期間 wait
    SCI1.SCR.BYTE =    0x30;     //送信イネーブル,受信イネーブル,割込みディセーブル
}
```

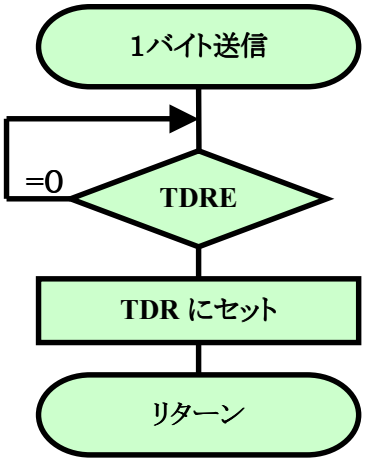
次に受信動作について考えてみましょう。当然ながら，TK-3052 はパソコンのキーがいつ押されるかわかりません。もっとも，受信動作そのものは SCI が自動的に行なってくれます。そして，データを受信したかどうかしらせるステータスが用意されています。マイコンはそのステータスを見て，受信していたらデータを読み込みます。

先ほど調べた「シリアルステータスレジスタ(SSR)」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。ソースリストは次のようなります。




```
/*****************************************************************************
    1文字受信（ポーリング）
------------------------------------------------------------------------------
    戻り値    受信データ
*****************************************************************************/
unsigned char rxone(void)
{
    unsigned char d;

    while(SCI1.SSR.BIT.RDRF==0){}          //受信するまで待つ
    d = SCI1.RDR;
    SCI1.SSR.BIT.RDRF = 0;
    return d;
}
```

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(H8/3052)は 1 つの命令を 0．080μs～0．88μs 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 SCI に送信データをどんどん書き込むと，まだ送信が終わっていないのに書き込むことになるかもしれません。それで，送信データを書き込んでよいか判断するステータスが用意されています。マイコンはそのステータスを見て，大丈夫ならデータを書き込みます。



先ほど調べた「シリアルステータスレジスタ(SSR)」をご覧下さい。ビット 7，‘TDRE’が 1 になったら送信データを書き込んでも大丈夫です。トランスミットデータレジスタ‘TDR’に送信データを書き込みます。フローチャートにしてみました。ソースリストは次のようになります。

```
/*************************************************************************
     1文字送信（ポーリング）
-------------------------------------------------------------------------
     引数 txdata         送信データ
*************************************************************************/
void txone(unsigned char txdata)
{
     while(SCI1.SSR.BIT.TDRE==0){}          //送信可能まで待つ
     SCI1.TDR = txdata;
     SCI1.SSR.BIT.TDRE = 0;
}
```

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，以上の関数を利用してメイン関数を作ります。データを受信したら，アスキーコードに変換して送信します。ソースリストは次のようになります。

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

     init_sci1();   //SCI1イニシャライズ
     txone(0x0d);   //復帰
     txone(0x0a);   //改行

     while(1){
          rd = rxone();          //1バイト受信
          hd = hex2asc(rd);      //アスキーコード変換
          txone(hd / 0x0100);    //上位バイト送信
          txone(hd & 0x00ff);    //下位バイト送信
          txone(0x0d);           //復帰
          txone(0x0a);           //改行
```

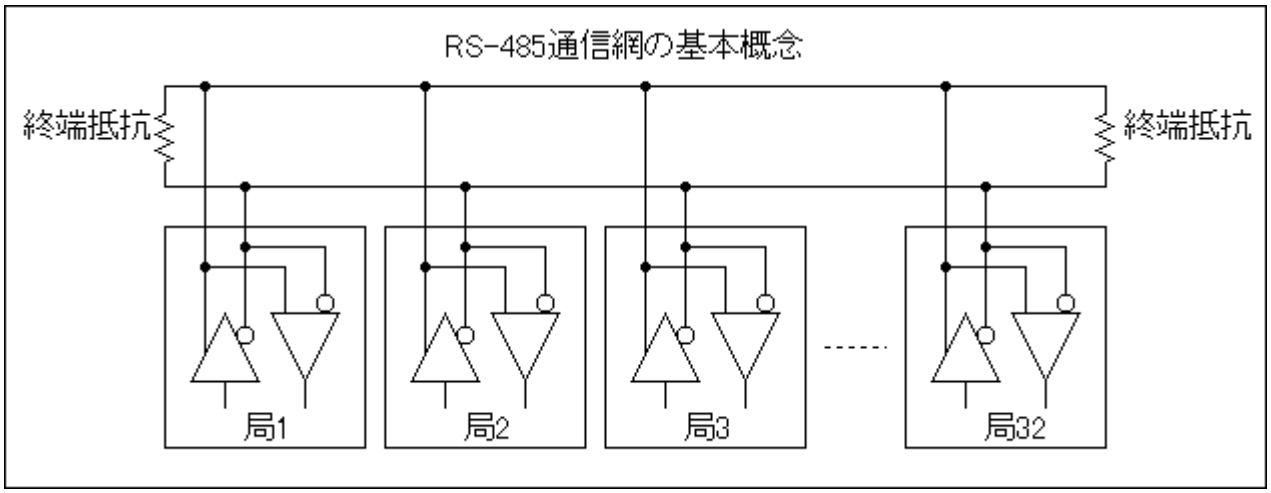
```
    }
}

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

    asc_dt = hex_dt & 0x0f;
    if (asc_dt>0x09)    {asc_dt = asc_dt + 0x37;}
    else                {asc_dt = asc_dt + 0x30;}
    hex_dt = hex_dt / 0x10;
    if (hex_dt>0x09)    {asc_dt = (hex_dt + 0x37) * 0x0100 + asc_dt;}
    else                {asc_dt = (hex_dt + 0x30) * 0x0100 + asc_dt;}

    return asc_dt;
}
```

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。なお，付属の CD-ROM にはあらかじめダウンロードするファイルがおさめられています。「rs232_01. abs」をダウンロードして実行してください。

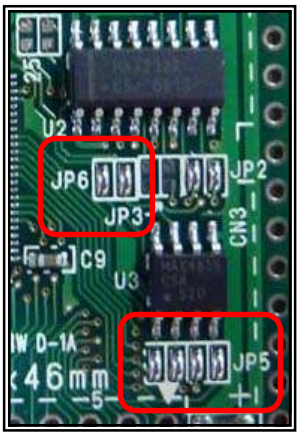
5. RS-485 とは

前項で調べたパソコンのシリアルポートはRS-232Cという規格です。1 対 1 で接続するためによく使われてきました。しかし，伝送速度が遅く，不平衡伝送のためノイズに弱いという欠点があり，最近の高速・長距離伝送に対応できなくなりました。今のパソコンで搭載されることは（一部の特殊用途を除いて）ありません。

RS-232C の欠点を改善した通信規格が幾つかあります。RS-485 はその一つです（もちろん，パソコンに標準で搭載されてはいるわけではありません）。規格上は 100K ビット/s で 1.2km まで（距離を短くすればもっと早くできる）接続できます。また，平衡伝送を採用しているためノイズにも強くなりました。さらに，バス方式に対応し 1 本のライン上に複数の端末（規格では MAX32 台）を接続できるようになりました。



TK-3052 は RS-485 のトランシーバ（ドライバとレシーバが一つになったもの，MAX485 相当品）が実装されています。本項では RS-485 の基本的な動作を実験します。その前に，TK-3052 の JP5 と JP6 をハンダでショートし RS-485 を使えるようにしましょう（右写真参照）。



RS-485 も調歩同期式モードで使うので，基本的なプログラムの考え方は何も変わりません。実験用のサンプルプログラムとして，送信データを自局で受信するセルフチェックプログラムと，H8/3052 のマルチプロセッサ通信機能を利用したセルフチェックプログラムを考えました。セルフチェックプログラムなので 1 台だけで動作します。MAX485 のドライバとレシーバを両方イネーブルにすることで,自分で送信したデータを自分で受信します。なお，セルフチェックなのでケーブルやコネクタは実装しませんが，ケーブルで外部に接続するときは CN8 か CN9 にコネクタを実装します。

■ サンプルプログラム(1)

MAX485 のドライバとレシーバを同時にイネーブルにし，送信データを自分で受信します。受信したデータは LED に表示します。なお，送信データは送信するたびにインクリメントし(0x00～0xFF)，0x00 になったときにブザーを鳴らします。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。「rs485_01. abs」をダウンロードして実行してください。

```
/************************************************************************
    メインプログラム
************************************************************************/
void main(void)
{
    unsigned char  TxData = 0x00;
    unsigned long  i;

//----- イニシャライズ ----------------------------------------------
    ini_port();
    ini_sci_0();

//----- メインループ ------------------------------------------------
    while(1){
        // 1データ送信
        while(SCI0.SSR.BIT.TDRE==0){}     //送信可?
        SCI0.TDR = TxData;                //送信
        SCI0.SSR.BIT.TDRE = 0;            //フラグクリア

        // 1データ受信,ポートAに表示する
        while(SCI0.SSR.BIT.RDRF==0){}     //受信した?
        PA.DR.BYTE = SCI0.RDR;            //受信データを表示
        SCI0.SSR.BIT.RDRF = 0;            //フラグクリア

        // 次の送信データをセット
        TxData++;

        // TxData=0x00のときブザーを鳴らす
        if (TxData==0x00){
            sounder_1shot();
        }

        // ウェイト
        for (i=0; i<62500; i++){}
    }
}

/************************************************************************
    I/O PORTの初期化
************************************************************************/
void ini_port(void)
{
    P2.PCR.BYTE    =    0x38;     //P23,24,25プルアップ抵抗の設定
    P2.DDR         =    0x07;     //P20-22 Out / P23-27 In
    P2.DR.BYTE     =    0x07;     //P20-22 初期値

    PA.DDR         =    0xff;     //PA0-7 Out
    PA.DR.BYTE     =    0x00;     //PA0-7=Low
```

```
    P9.DDR        =    0x30;     //P94,95 Out
    P9.DR.BYTE    =    0x10;     //P95(-RE)=Low,P94(DE)=High
                                 //ドライバ&レシーバ イネーブル
}

/************************************************************************
    SCIの初期化
************************************************************************/
void ini_sci_0(void)
{
    #define       MHz       25             // Clock=25MHz
    #define       BAUD      38400          // baudrate
    #define       BITR (MHz*1000000)/(BAUD*32)-1
    #define       WAIT_1B   (MHz*1000000)/6/BAUD

    unsigned  long i;

    SCI0.SCR.BYTE =    0x00;          // 動作停止
    SCI0.SMR.BYTE =    0x00;          // 非同期･8bit･P-non･CLK/1
    SCI0.BRR      =    BITR;          // ビットレート
    for  (i=0;i<WAIT_1B;i++){};       // 1bit期間 wait
    SCI0.SCR.BYTE =    0x30;          // 送受信イネーブル
    SCI0.SSR.BYTE =    0x00;          // フラグクリア
}

/************************************************************************
    サウンダを0.1秒鳴らす（1KHｚ）
************************************************************************/
void sounder_1shot(void)
{
    unsigned int i,j;

    for (j=0; j<100; j++){
        P2.DR.BIT.B2 = 0;
        for (i=0;i<2083;i++){}
        P2.DR.BIT.B2 = 1;
        for (i=0;i<2083;i++){}
    }
}
```

■ サンプルプログラム(2)

H8/3052 のマルチプロセッサ通信機能を使用したセルフチェックプログラムです。マルチプロセッサ通信機能を使用すると，マルチプロセッサビットを付加した調歩同期式シリアル通信により複数のプロセッサ間で通信回線を共有してデータの送受信を行うことができ，RS-485 のバス方式の通信網に応用することができます。

マルチプロセッサ通信では受信局に各々固有の ID コードを割付けます。そして，送信局は受信局の ID コードにマルチプロセッサビット‘1’を付加したデータを送信し，続いてその受信局に送りたいデータにマルチプロセッサビット‘0’を付加して送信します。受信局はマルチプロセッサビットが‘1’のデータを受信すると自局の ID と比較し，一致した場合は続いて送信されるデータを受信します。一致しなかった場合は再びマルチプロセッサビットが‘1’のデータを受信するまでデータを読み飛ばします。H8/3052 の SCI にはマルチプロセッサビットが‘1’のデータを受信するまで受信データを読み飛ばす機能と，マルチプロセッサビットを付加する機能が内蔵されています。自局の ID コードの管理，及び受信した ID コードとの比較，ID コードが一致したときに続いて送信されるデータの受信はソフトウェアで行ないます。

このサンプルプログラムでは，MAX485 のドライバとレシーバを同時にイネーブルにし，送信データを自分で受信します。送信データの ID コードを 0x00 から 0xFF まで順番に変化させながら送信し，自局の ID コード(0x55)を受信したときのデータを LED に表示しブザーを鳴らします。なお，送信データも順にインクリメントします。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。「rs485_02. abs」をダウンロードして実行してください。

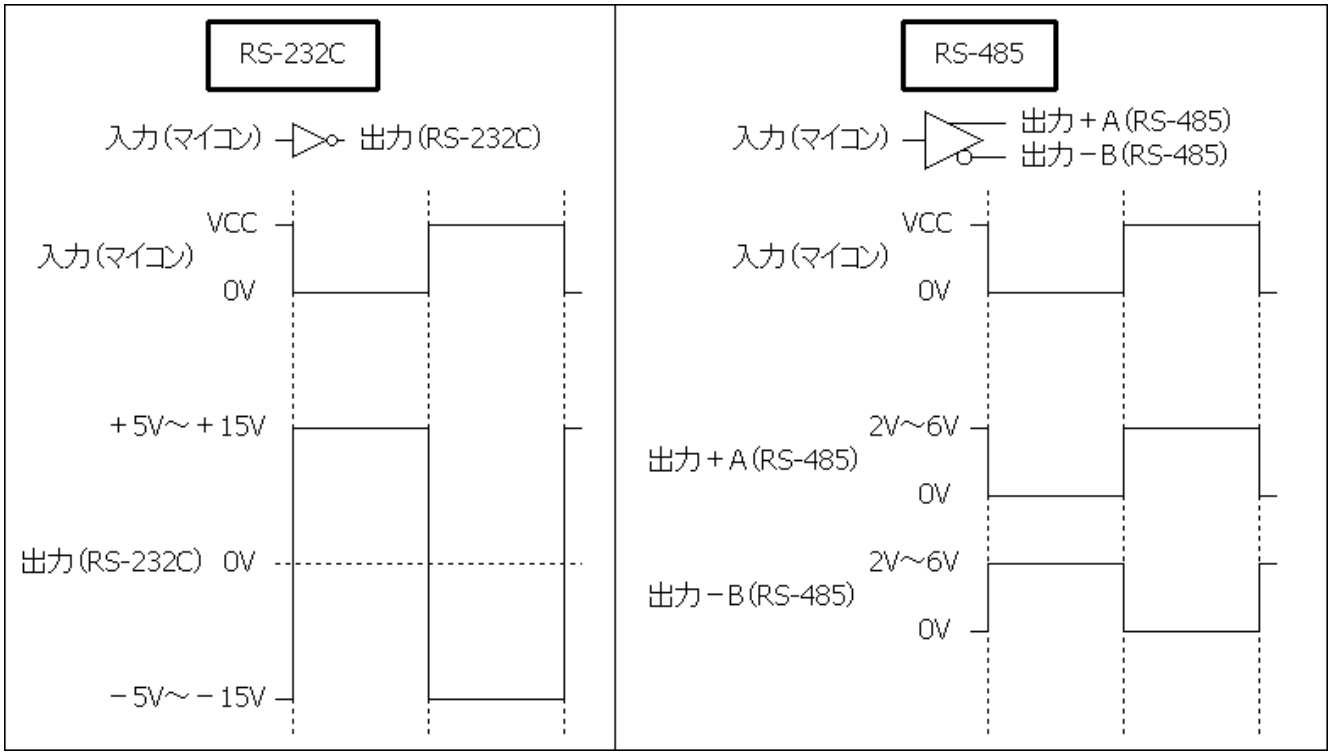
```
/************************************************************************
    変数の定義
************************************************************************/
unsigned char  TxData = 0x00;      //送信データ
unsigned char  RxData;             //受信データ
unsigned char  TxId = 0x00;        //送信IDコード
unsigned char  MyId = 0x55;        //自局IDコード
unsigned char  RxId;               //受信IDコード
unsigned char  TxStatus = 0;       //送信ステータス
                                   //  0:IDコード送信
                                   //  1:データ送信
                                   //  2-FF:次の送信待ち
unsigned char  RxStatus = 0;       //受信ステータス
                                   //  0:IDコード受信待ち
                                   //  1:自局データ受信待ち

/************************************************************************
    メインプログラム
************************************************************************/
void main(void)
{
    unsigned long  i;

//----- イニシャライズ -------------------------------------------------
    ini_port();
    ini_sci0();

//----- メインループ ---------------------------------------------------
    while(1){
        switch (TxStatus){
            case 0:        //IDコード送信
                if (SCI0.SSR.BIT.TDRE==1){
                    SCI0.SSR.BIT.MPBT = 1;
                    SCI0.TDR = TxId;
                    SCI0.SSR.BIT.TDRE = 0;
```

```
                    TxId++;
                    TxStatus = 1;
               }
               break;
          case 1:        //データ送信
               if (SCI0.SSR.BIT.TDRE==1){
                    SCI0.SSR.BIT.MPBT = 0;
                    SCI0.TDR = TxData;
                    SCI0.SSR.BIT.TDRE = 0;
                    TxStatus = 2;
               }
               break;
          default:       //次のデータ送信待ち
               TxStatus++;
               if ((TxId==0x00)&&(TxStatus==0))  {TxData++;}         //送信データの更新
     }

     switch (RxStatus){
          case 0:        //IDコード受信
               if (SCI0.SSR.BIT.RDRF==1){
                    RxId = SCI0.RDR;
                    SCI0.SSR.BIT.RDRF = 0;  //フラグクリア
                    SCI0.SSR.BIT.ORER = 0;  //フラグクリア
                    SCI0.SSR.BIT.FER = 0;   //フラグクリア
                    if (RxId==MyId){   //自局IDコード受信
                         SCI0.SCR.BIT.MPIE = 0;  //ﾏﾙﾁﾌﾟﾛｾｯｻｲﾝﾀﾗﾌﾟﾄﾃﾞｨｾｰﾌﾞﾙ
                         RxStatus = 1;
                    }
                    else{              //他局IDコード受信
                         SCI0.SCR.BIT.MPIE = 1;  //ﾏﾙﾁﾌﾟﾛｾｯｻｲﾝﾀﾗﾌﾟﾄｲﾈｰﾌﾞﾙ
                    }
               }
               break;
          case 1:        //自局データ受信
               if (SCI0.SSR.BIT.RDRF==1){
                    RxData = SCI0.RDR;
                    PA.DR.BYTE = RxData;    //受信データをPA0-7に表示
                    SCI0.SSR.BIT.RDRF = 0;  //フラグクリア
                    SCI0.SSR.BIT.ORER = 0;  //フラグクリア
                    SCI0.SSR.BIT.FER = 0;   //フラグクリア
                    SCI0.SCR.BIT.MPIE = 1;  // ﾏﾙﾁﾌﾟﾛｾｯｻｲﾝﾀﾗﾌﾟﾄｲﾈｰﾌﾞﾙ
                    RxStatus = 0;
                    sounder_1shot();   //ブザーオン
               }
               break;
     }
   }
}

/*********************************************************************
     I/O PORTの初期化
*********************************************************************/
void ini_port(void)
{
   P2.PCR.BYTE   =   0x38;     //P23,24,25プルアップ抵抗の設定
```

```
    P2.DDR          =   0x07;     //P20-22 Out / P23-27 In
    P2.DR.BYTE      =   0x07;     //P20-22 初期値

    PA.DDR          =   0xff;     //PA0-7 Out
    PA.DR.BYTE      =   0x00;     //PA0-7=Low

    P9.DDR          =   0x30;     //P94,95 Out
    P9.DR.BYTE      =   0x10;     //P95(-RE)=Low,P94(DE)=High
                                  //ドライバ&レシーバ イネーブル
}

/************************************************************************
    SCI0の初期化
************************************************************************/
void ini_sci0(void)
{
    #define       MHz       25              // Clock=25MHz
    #define       BAUD      38400           // baudrate
    #define       BITR (MHz*1000000)/(BAUD*32)-1
    #define       WAIT_1B   (MHz*1000000)/6/BAUD

    unsigned  long i;

    SCI0.SCR.BYTE =   0x00;           // 動作停止
    SCI0.SMR.BYTE =   0x04;           // 非同期･8bit･P-non･CLK/1･ﾏﾙﾁﾌﾟﾛｾｯｻﾓｰﾄﾞ
    SCI0.BRR      =   BITR;           // ビットレート
    for  (i=0;i<WAIT_1B;i++){};       // 1bit期間 wait
    SCI0.SCR.BYTE =   0x38;           // 送受信イネーブル,ﾏﾙﾁﾌﾟﾛｾｯｻｲﾝﾀﾗﾌﾟﾄｲﾈｰﾌﾞﾙ
    SCI0.SSR.BYTE =   0x00;           // フラグクリア
}

/************************************************************************
    サウンダを0.1秒鳴らす（1KHｚ）
************************************************************************/
void sounder_1shot(void)
{
    unsigned int i,j;

    for (j=0; j<100; j++){
        P2.DR.BIT.B2 = 0;
        for (i=0;i<2083;i++){}
        P2.DR.BIT.B2 = 1;
        for (i=0;i<2083;i++){}
    }
}
```

6. RS-232C と RS-485 の比較

ここまで調歩同期式シリアル通信について，また，これまでパソコンでもよく使われてきた RS-232C，その欠点を改良した RS-485 について実験してきました。ここで，RS-232C と RS-485 を比較してみましょう。RS-232C と RS-485 のドライバの入力に同じ信号を加えると次のような波形が出力されます。



RS-232C と RS-485，それぞれのレシーバは，これらの波形を受信すると元の波形に変換します。RS-232C のレシーバは，0V に対して-3V 以下で VCC を，+3V 以上で 0V を出力します。

一方，RS-485 のレシーバは 0V からの電位ではなく，(+A)と(-B)の電位差に応じて出力されます。(+A)-(-B)＝0. 2V 以上で VCC を，(+A)-(-B)＝-0. 2V 以下で 0V を出力します。

ノイズに対する強さという点で考えてみましょう。RS-232C の場合，ドライバは±5V～±15V で出力されるのに対し，レシーバは±3V 以上で OK です。ということは，仮にドライバが最低の±5V で出力していたとしても，あと±2V の余裕があることになります。つまり，±2V 以内のノイズ，もしくは電圧降下であれば，理論上は問題は発生しません。

RS-485 の場合は単純にノイズ電圧値だけでは決まりません。(+A)と(-B)のケーブルが別々の場所に敷設されるということは考えられません。通常はツイストペアケーブルだったり，同じシールド線の中の 2 本が使われたりします。ということは，ケーブル周辺で発生するノイズは(+A)と(-B)の両方のケーブルに同じように乗ると考えられます。仮に+1V のノイズが発生したとしましょう。両方のケーブルに同じように+1V のノイズが乗るので電位差を計算すると，

$$((+A)+1) - ((-B)+1) = (+A) + 1 - (-B) - 1 = (+A) - (-B)$$

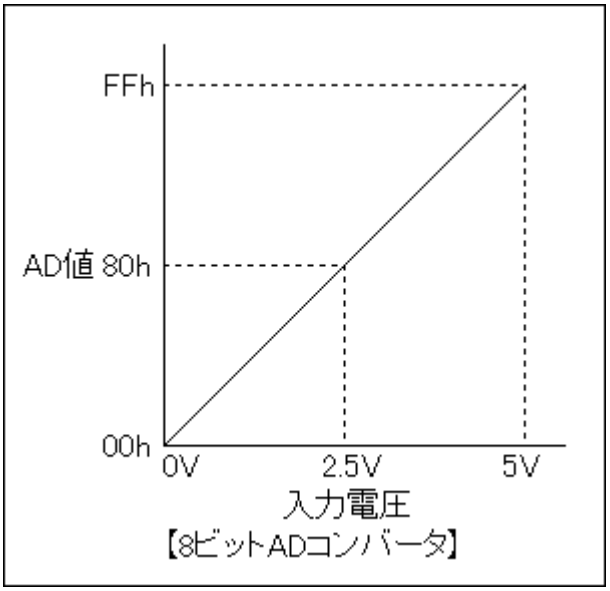
となり，理論的にはノイズがきれいに除去されます。このような伝送方式を平衡伝送といい，高速・長距離の通信によく使われます。

第8章 A/D 変換器の使い方

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 A/D 変換器（A/D 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 A/D 変換器の基本的な考え方と使い方を調べてみましょう。

1. A/D 変換器とは何か

A/D 変換器は，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 A/D 変換器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。



A/D 変換器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など，明るさを測るには CDS などを使います。

2. H8/3052 の A/D 変換器

H8/3052 には A/D 変換器が内蔵されています。詳しくは，ハードウェアマニュアルの「15. A/D 変換器」で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 VREF までです。TK-3052 は VREF 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'0000000000，5V 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，5V のときは FFC0h になります。変換結果をプログラムで 6 ビット右シフトして 0000h～03FFh として扱うこともあります。もちろん，どうするかはプログラマしだいです。なお，これから作成するプログラムでは平均後の上位 8 ビットを使います。

入力チャンネル：8 チャンネル

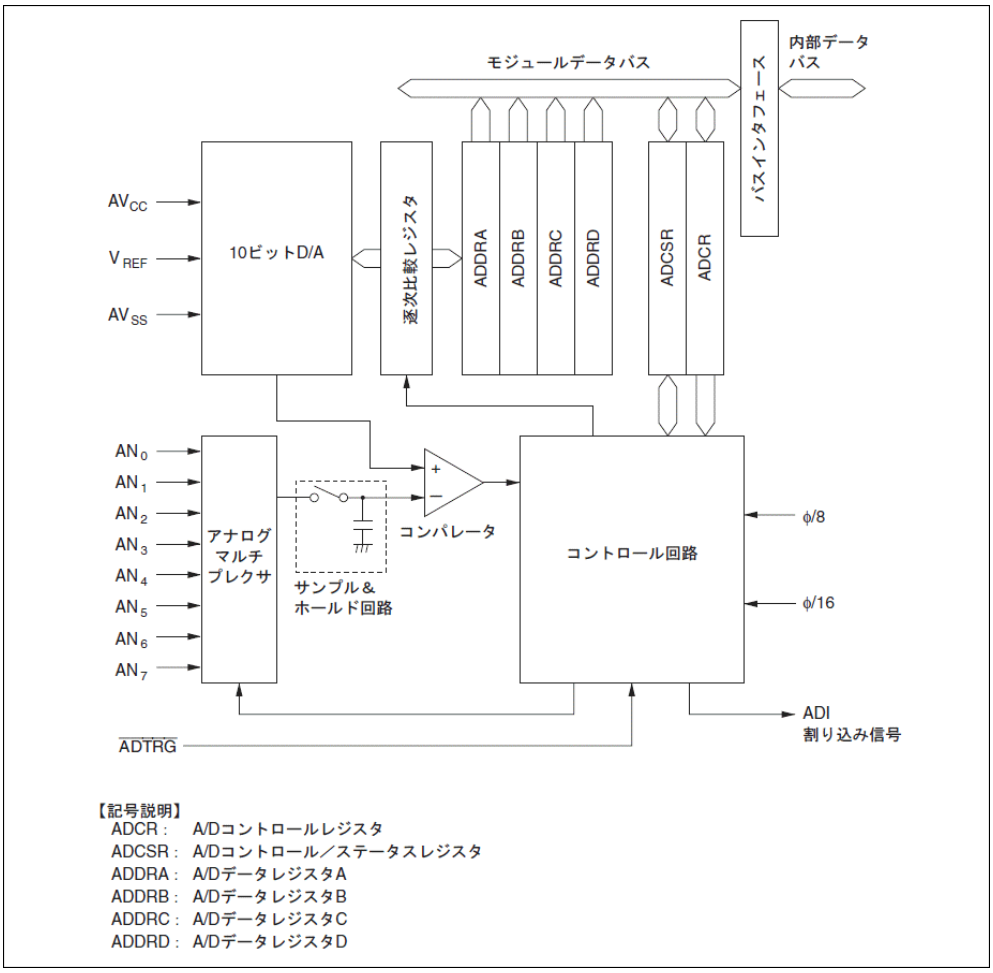
A/D 変換器自体は 1 個だけなのですが，アナログマルチプレクサ回路が内蔵されているので，8 種類の電圧を入力することができます。そのため，同時に 8 チャンネルの A/D 変換ができるわけでなく，順番に 1 チャンネルずつ A/D 変換します。

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 A/D 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 A/D 変換します。

変換速度

TK-3052 は CPU クロックが 25MHz なので，1 チャンネルあたり最短 5．4μs で変換できます。



3. 設定用レジスタ

A/D データレジスタ A～D(ADDRn) : アドレス=0xFFE0, 0xFFE2, 0xFFE4, 0xFFE6 番地(下位 16 ビット)

ビット	ビット名	初期値	R/W	説明
15	AD9	0	R	A/D 変換データ。 上位 10 ビットに格納される。
14	AD8	0	R	
13	AD7	0	R	
12	AD6	0	R	
11	AD5	0	R	
10	AD4	0	R	
9	AD3	0	R	
8	AD2	0	R	
7	AD1	0	R	
6	AD0	0	R	
5	−	0	R	
4	−	0	R	
3	−	0	R	
2	−	0	R	
1	−	0	R	
0	−	0	R	

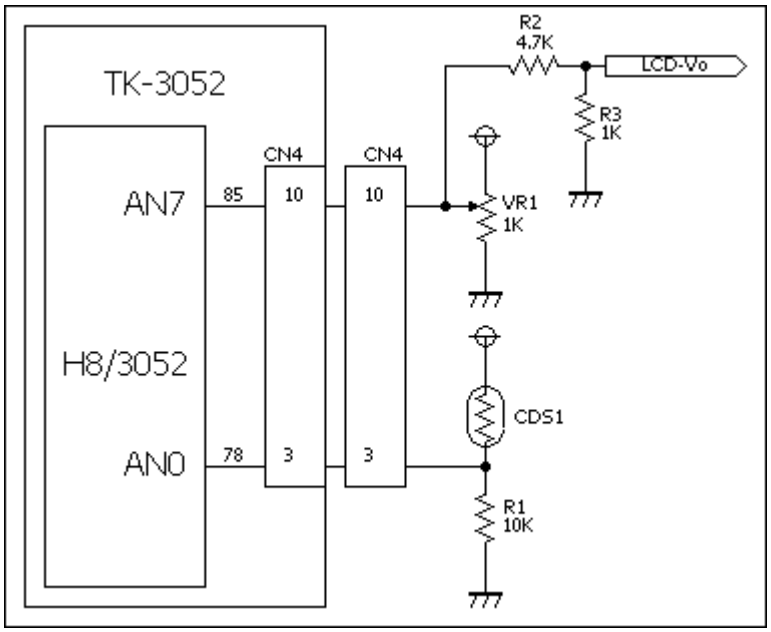
アナログ入力チャンネル		A/D データレジスタ
グループ 0	グループ 1	
AN0	AN4	ADDRA
AN1	AN5	ADDRB
AN2	AN6	ADDRC
AN3	AN7	ADDRD

A/D コントロール/ステータスレジスタ(ADCSR) : アドレス=0xFFE8 番地(下位 16 ビット)

ビット	ビット名	初期値	R/W	説明
7	ADF	0	R/W	A/D エンドフラグ。 0:[クリア条件]ADF=1 の状態で, ADF フラグをリードした後, ADF フラグに 0 をライトしたとき。 1:[セット条件](1)単一モード:A/D 変換が終了したとき。(2)スキャンモード:設定された全てのチャネルの A/D 変換が終了したとき。
6	ADIE	0	R/W	A/D インタラプトイネーブル。 0:A/D 変換終了による割り込み(ADI)要求を禁止。 1:A/D 変換終了による割り込み(ADI)要求を許可。
5	ADST	0	R/W	A/D スタート。 0:A/D 変換を停止。 1:(1)単一モード:A/D 変換を開始し, 変換が終了すると自動的に 0 にクリア。(2)スキャンモード:A/D 変換を開始し, ソフトウェア, リセット, またはスタンバイモードによって 0 にクリアされるまで選択されたチャネルを順次連続変換。
4	SCAN	0	R/W	スキャンモード。 0:単一モード。 / 1:スキャンモード。
3	CKS	0	R/W	クロックセレクト。 0:変換時間＝266 ステート(MAX)。 1:変換時間＝134 ステート(MAX)。
2	CH2	0	R/W	チャネルセレクト。 [単一モード] 000:AN0 001:AN1 010:AN2 011:AN3 100:AN4 101:AN5 110:AN6 111:AN7 [スキャンモード] 000:AN0 001:AN0～1 010:AN0～2 011:AN0～3 100:AN4 101:AN4～5 110:AN4～6 111:AN4～7
1	CH1	0	R/W	
0	CH0	0	R/W	

A/D コントロールレジスタ(ADCR) ： アドレス=0xFFE9 番地(下位 16 ビット)				
ビット	ビット名	初期値	R/W	説明
7	TRGE	0	R/W	トリガイネーブル。 0:外部トリガ入力による A/D 変換の開始を禁止。 1:外部トリガ端子(ADTRG)の立ち下がりエッジで A/D 変換を開始。
6	−	1	−	リザーブビット。
5	−	1	−	リザーブビット。
4	−	1	−	リザーブビット。
3	−	1	−	リザーブビット。
2	−	1	−	リザーブビット。
1	−	1	−	リザーブビット。
0	−	0	−	リザーブビット。

4. 実習回路



全体の回路図は付録をご覧ください。この章で使う回路だけ抜き出してみました。

5. 明るさを表示する

では，A/D 変換器で明るさを表示するプログラムを作ってみましょう。ここでは，CDS というセンサを使って明るさをマイコンに取り込んでみます。CDS は光のエネルギーで抵抗値が変化する素子です。CDS は個体差が大きな素子ですが，一般的に，明るいところでは 100Ω以下だったものが，暗くなると数 KΩになります。前ページの回路で明るさを電圧に変換して A/D 変換器のチャネル 0(AN0)に入力します。

A/D 変換器で変換すること自体はそれほど難しくありません。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 にセンサがつながっているので，AD 変換が終了したら ADDRA からデータを入力します。

この AD 値をそのまま使えばよいかというと，そういうわけにはいきません。入力電圧が安定していてノイズがまったくなければよいのですが，現実にはそういう信号はなく，ノイズなどの影響で AD 値はふらふらします。そこで，何回か入力してその平均値を求めることでノイズの影響をなくします。今回は 65536 回(16 進で 0x10000 回)加算して平均しました。得られた平均値の上位 8 ビットを LED に表示します。

さて，以上のことを踏まえてコーディングしましょう。メイン関数のソースリストは次のようになりました。

```
/************************************************************************
    メインプログラム
************************************************************************/
void main(void)
{
    PA.DDR        =   0xff;     // ポートAを出力に設定
    PA.DR.BYTE    =   0x00;     // ポートA出力クリア

    AD.ADCSR.BYTE = _00001000B;  //ADCイニシャライズ
                                 // AD終了割り込みディセーブル
                                 // AD変換停止,単一モード,変換時間=134ステート
                                 // AN0選択

    while(1){
        AdSum = 0;     //加算バッファクリア
        AdCount = 0;   //加算カウンタ初期化

        for (AdCount=0; AdCount<0x00010000; AdCount++){
            AD.ADCSR.BIT.ADST = 1;            //AD変換スタート
            while(AD.ADCSR.BIT.ADF==0){}      //AD変換終了待ち
            AD.ADCSR.BIT.ADF = 0;             //AD変換終了フラグクリア
            AdSum = AdSum + (unsigned long)AD.ADDRA;    //AD値を取得,バッファに加算
        }

        PA.DR.BYTE = (unsigned char)(AdSum / 0x01000000);    //加算バッファの上位8ビットを表示
    }
}
```

CDSを手で覆ったりして明るさを変えるとLEDに表示される値(2進数)が変化します。なお，付属のCD-ROMにはあらかじめダウンロードするファイルがおさめられています。「adc01．abs」をダウンロードして実行してください。

■ 練習問題

今のプログラムと同じようにして，今度はAN7につながっているボリュームの位置に応じた値をLEDに表示して下さい。(解答例は「adc02．abs」)

第9章 その他の内蔵 I/O について

H8/3052 には今回説明したもの以外にも便利な内蔵 I/O が用意されています。そのうち幾つかについて，どのようなものか，ごく簡単に説明します。詳細について，またここに掲載されていないものについてはハードウェアマニュアルをご覧ください。

1. DMA コントローラ

通常，データ転送は必ず CPU を介して行ないます。

例 1:メモリアドレス A のデータをメモリアドレス B にコピーするとき，アドレス A のデータを **CPU が読み込み**，読み込んだデータをアドレス B に書き込む。

例 2:内蔵 I/O からデータを入力するとき，内蔵 I/O からデータを **CPU が読み込み**，読み込んだデータをメモリに書き込む。

この一連の動作について，CPU は命令を読み込み，解析して，実行する，という手順を踏みます。そのため，どうしてもある程度の時間はかかってしまいます。

DMA とは「Direct Memory Access」の略で，CPU を介さず直接(ダイレクトに)データを転送する方法です。それを制御するのが DMA コントローラです。データ転送に特化したものなので(逆にいうと，それ以外はなにもできない)，高速にデータ転送を行なえます。特に，一連かつ大量のデータを転送するときに効果を発揮します(逆にデータ数が少ないと，DMA コントローラの初期設定など必要な追加プログラムのためメリットがなくなります)。また，速度面のメリットだけでなく，CPU を介さずハードウェアによるタイミングで順次データを転送する目的にも使えます。

2. プログラマブルタイミングパターンコントローラ(TPC)

ITU をタイムベースとし，コンペアマッチ信号をトリガとしてパルス出力を行ないます。また，コンペアマッチ信号で DMA を起動することで，CPU の介在なしに順次パルスを出力することもできます。

3. ウォッチドッグタイマ

システムの監視を行なうタイマです。8 ビットカウンタが常にインクリメントされていて，オーバフローすると H8/3052 を強制的にリセットします。

プログラムでは定期的にカウンタを 0 にクリアしオーバフローしないようにしておきます。何かのトラブルでプログラムが暴走してカウンタのクリアがなされないと，やがてオーバフローしてリセットされるという仕掛けです。

4. スマートカードインターフェース

SCI のチャネル 0 は拡張機能として，ISO/IEC7816-3(Indentification Card)に準拠した IC カード(スマートカード)インターフェースをサポートしています。

5. D/A 変換器

D/A 変換器は A/D 変換器の逆で，デジタル値に応じた電圧を出力します。H8/3052 には 8 ビット分解能の D/A 変換器が 2 チャネル内蔵されています。出力電圧は 0V～VREF までで，変換時間は最大 10μs です。

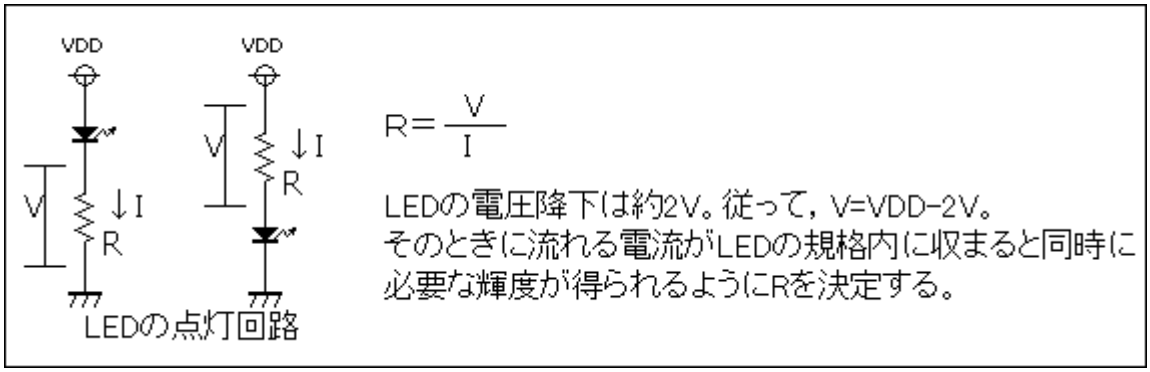
第10章 タイマ&LED ディスプレイキットを使ったダイナミックスキャンの学習

1. LED の基本的な点灯方法
2. LED のダイナミック点灯
3. ルーレットの製作

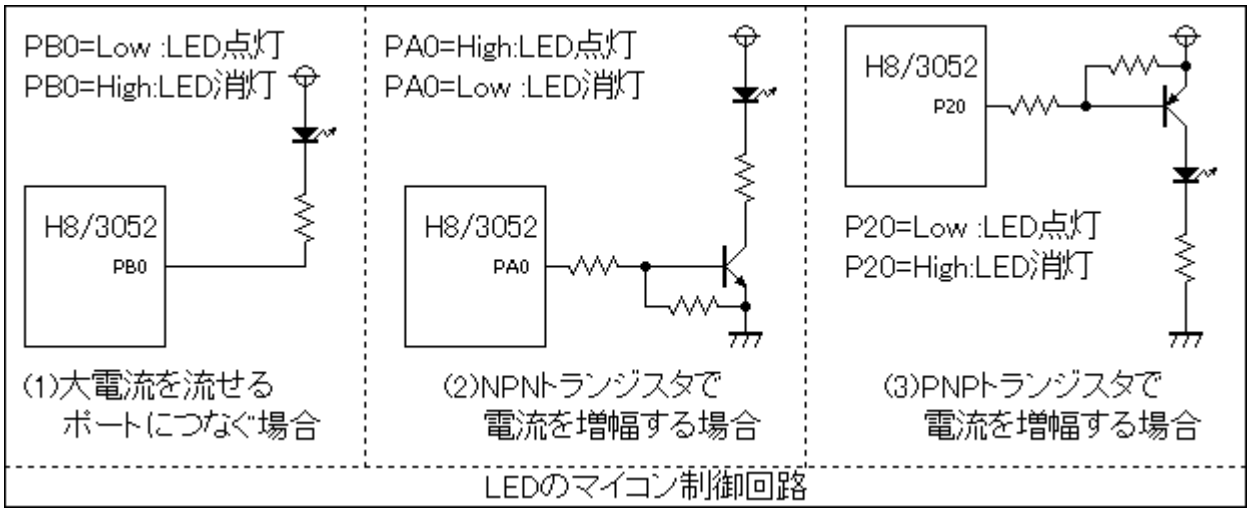
駅のホームや電車の中に LED による電光掲示板が置かれるようになり，いろいろな情報をわかりやすく伝えることができるようになりました。このような，たくさんの LED を制御することには，人間の目の特徴を巧みに利用した，定番ともいえる一つのテクニックが関係しています。この章では「タイマ&LED ディスプレイキット」を使い，そのテクニックを説明します。

1. LEDの基本的な点灯方法

まずはおさらいです。LED は次のように電源と抵抗をつなぐと点灯します。なお，抵抗は LED に流れる電流を制限する役目があり，必ず入れなければなりません。



ただ，これだとマイコンで制御できないので，次のようにして点灯/消灯を制御できるようにします。



(1)の場合，PB0 を Low にすると LED 点灯，High にすると LED 消灯になります。TK-3052 で使用しているワンチップマイコン H8/3052 の PB0～B7 は，他のポートと異なり Low レベル出力のとき 20mA まで流すことができますので，LED の点灯など比較的大きな電流を流す必要があるときに使うことができます。大きな電流を流せないポートを使うときや 20mA を超える電流が必要なときは(2)や(3)のようにトランジスタを使って電流を増幅するか，ドライバ IC で電流を増幅します。

ところで，この方法で LED を制御することはできますが，数個程度ならともかく数十個となると LED を制御するだけで全てのポートを使い果たしてしまいます(場合によっては足りないかも･･･)。回路図を見るとわかりますが，今回使用している LED ディスプレイは LED が 64 個入っています。「タイマ&LED ディスプレイキット」はそのほかに 12 個の LED を実装していますから，全部で 76 個の LED を制御しなければなりません。

そこで，ダイナミック点灯という方法を使って少ないポートでたくさんの LED を制御します。

2. LED のダイナミック点灯

ダイナミック点灯とは，一言でいえば目の錯覚を利用したごまかしです。ちょっと実験してみましょう。次のプログラムを入力して実行してみてください。付属の CD-ROM にもおさめられています。「b6092_01_jikken」です。

```
/************************************************************************/
/*                                                                      */
/*  FILE        :b6092_01_jikken.c                                      */
/*  DATE        :Fri, Jun 11, 2010                                      */
/*  DESCRIPTION :Main Program                                           */
/*  CPU TYPE    :H8/3052F                                               */
/*                                                                      */
/*  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
#include "binary.h"     //Cで2進数を使うための定義

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

/************************************************************************
    メインプログラム
************************************************************************/
void main(void)
{
    ini_io();

    while(1){
        wait();
        PA.DR.BYTE = rotlc(1,PA.DR.BYTE); //左ローテート
    }
}

/************************************************************************
    I/Oポートの初期化
************************************************************************/
void ini_io(void)
{
    PA.DDR        =    _11111111B;    //PortAを出力に設定
    PA.DR.BYTE    =    _00000001B;    //出力初期値
}

/************************************************************************
    ウェイト
************************************************************************/
```

```
void wait(void)
{
    unsigned long i;

    for (i=0;i<4166666/1;i++){}  //←'1'を増やしてウェイト時間を短くする
}
```

順番に LED を点灯していくプログラムです。さて，黄色でマークしている行の「/1」を「/2」に変更して実行してみてください。光り方はどうなりますか？きっと，隣の LED に移るのが速くなったことでしょう。

さらに「/4」にしたらどうでしょうか？「/8」，「/16」・・・。どんどん速くなっていきます。やがて点滅というより光が流れているように見えてくることでしょう。さらに速くすると，暗くはなりましたが，全部点灯しているように見えてこないでしょうか。目のいい人で「なんとなく，ちらついているかな？」という感じになります。さらに速くすると，ちらつきを感じることなく全部点灯しているように見えるはずです。

もちろん，プログラム自体は変更していませんから，瞬間瞬間では 1 個の LED しか点灯していません。しかし，人間の目には残像現象という性質があります。そのため LED を消してもすぐにはわかりません。わからないうちに同じ LED を点灯すると，その LED は消えたと感じないわけです。

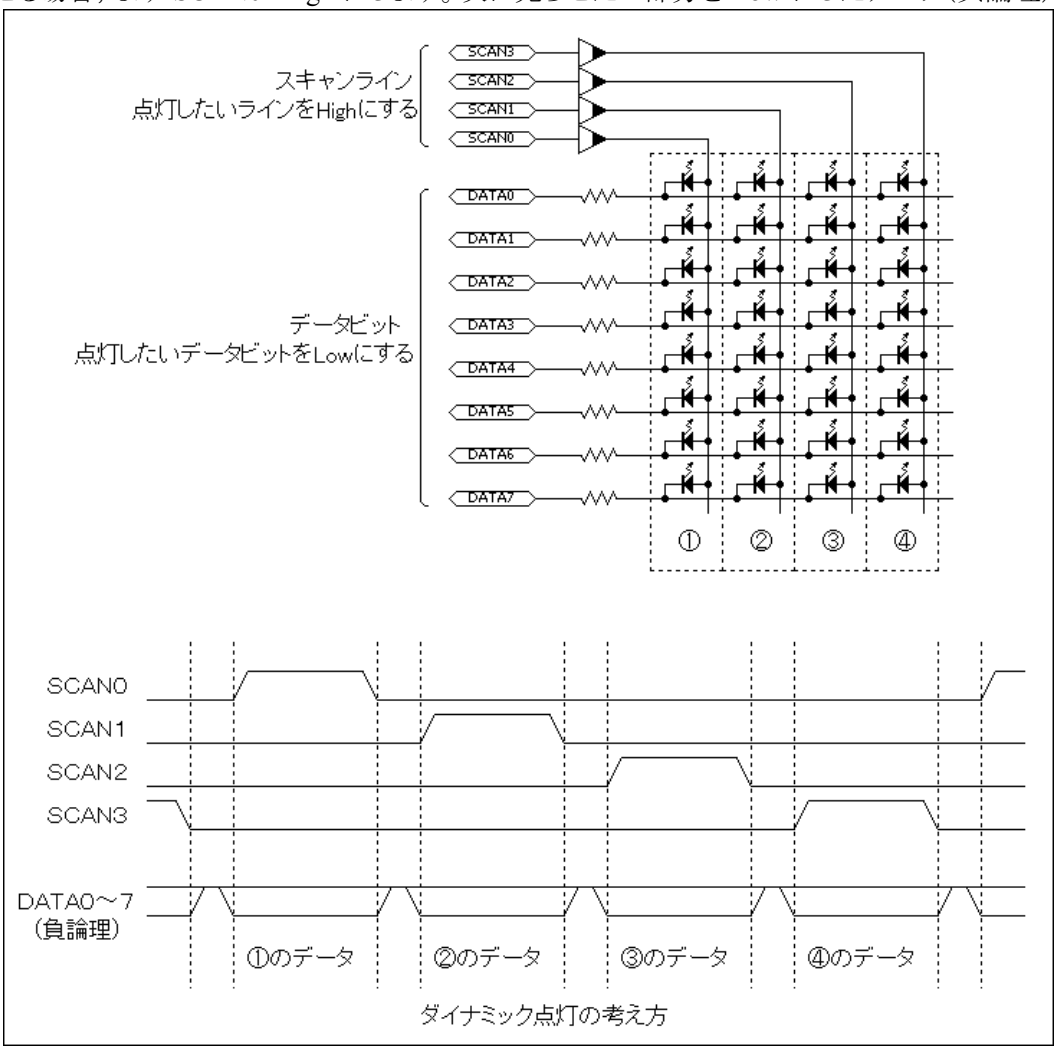
ただし，つきっぱなしと比べると 8 分の 1 の時間しか LED は点灯していないため，暗く感じてしまいます。そこで，たくさん電流を流して（つまり抵抗値を小さくして）明るくしてあげます。

では，この考え方を応用して，たくさんの LED の点滅を制御する方法を考えてみましょう。少し省略して 32 個の LED を制御する回路とタイミングチャートは次のようになります。

①の LED を光らせる場合，まず SCAN0=High にします。次に光らせたい部分を Low にしたデータ（負論理）を DATA0～7 にセットします。同じようにして②，③，④の LED を光らせます。あとはこれを繰り返します。

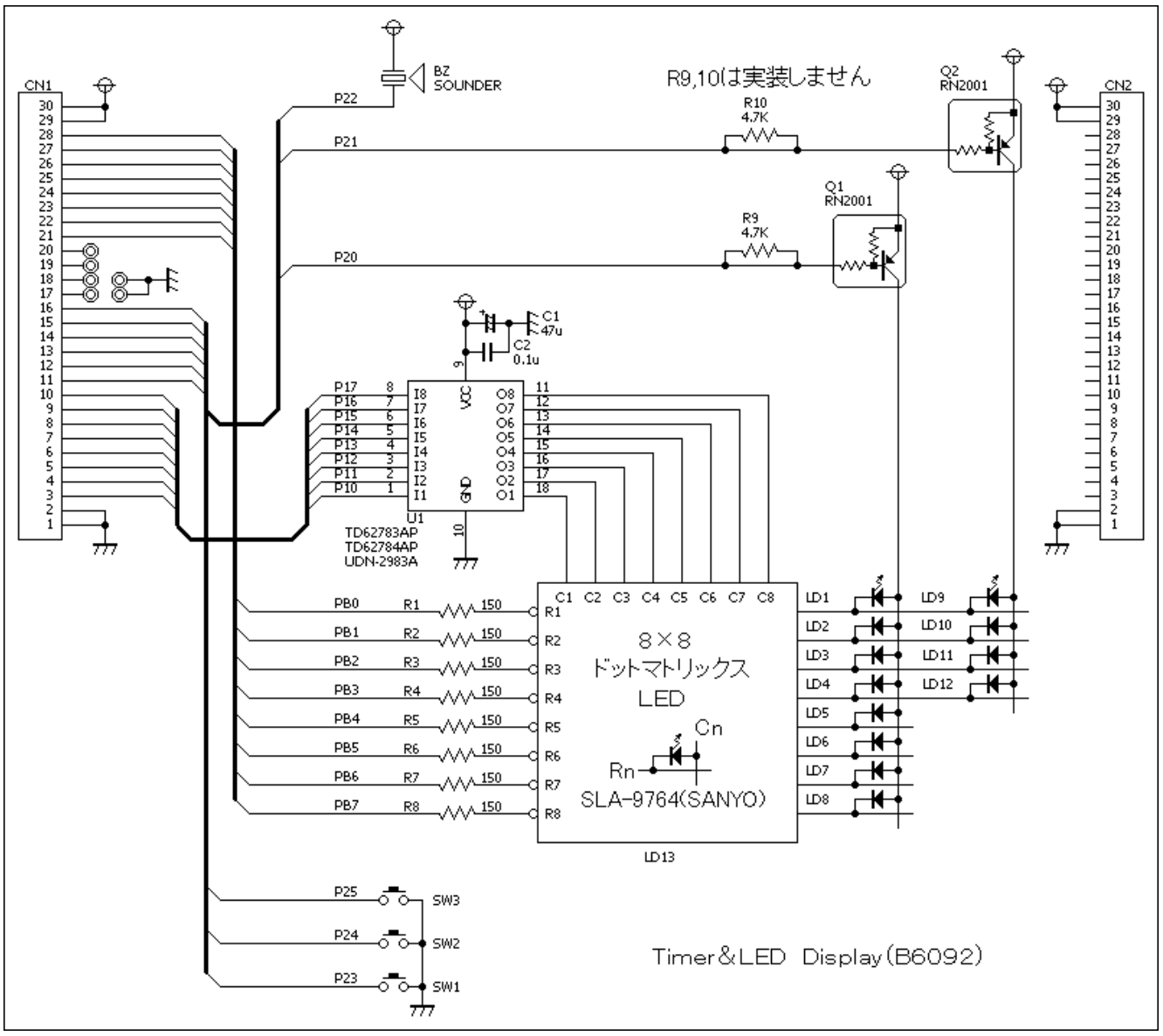
先の実験と同じように，瞬間々を見れば最大で 8 個の LED しか点灯していないのですが，人間の目の残像現象という性質のため，LED が消えてもすぐにはわかりません。わからないうちにもう一度同じ LED を点灯すると，その LED は消えたと感じないわけです。

①→②→③→④→①・・・という切り替えを人間の目で分からないくらいの速さで行なえば，全ての LED が同時に点灯しているように見せかけることができます。



ダイナミック点灯の考え方

ここで「タイマ＆LED ディスプレイキット」の回路図をご覧ください。



P10～P17, P20, P21 の 10 本のスキャンラインを使って 72 個の LED を制御しています。このうち, P10～P17 は High にするとスキャンラインに電流が流れる一方, P20 と P21 は Low にするとスキャンラインに電流が流れます。

◆

このことを踏まえた上で, 最初なので P20 と P21 につながっている 2 本のスキャンラインだけ使って, LD1～12 の 12 個の LED を制御してみましょう。

このプログラムでは ITU のチャネル 4 を使って 1ms の間隔で割り込みをかけ, その中でダイナミックスキャンのタイミングを作っています。P20 が Low のとき Q1 がオンになり LD1～8 のアノード側が High になります。その状態で LD1～8 の LED の点灯パターンを PB0～PB7 に負論理で出力します。同じように P21 が Low のとき Q2 がオンになり LD9～12 のアノード側が High になります。その状態で LD9～12 の点灯パターンを PB0～PB7 に負論理で出力します。割り込みのたびに P20=Low/P21=High と P20=High/P21=Low を切り替えます。

このプログラムの点灯パターンは LD1～12 をぐるぐる 1 個づつルーレットのように光らせていきます。表示データは‘DisplayData’にセットされていて, wait() 関数の間隔でローテートします。

さて，以上のことを踏まえてコーディングしましょう。ソースリストの一部は次のとおりです。

```
/************************************************************************
    グローバル変数の定義とイニシャライズ(RAM)
************************************************************************/
unsigned int  DispData =   0x0001;  //LED表示データ

/************************************************************************
    メインプログラム
************************************************************************/
void main(void)
{
    ini_io();
    ini_itu();

    while(1){
        wait();
        DispData = DispData * 2;
        if (DispData>=0x1000) {DispData = 0x0001;}
    }
}

/************************************************************************
    ITUの初期化
************************************************************************/
void ini_itu(void)
{
    ITU.TSTR.BYTE      =   _11100000B;   //タイマ停止

    ITU4.TCR.BYTE      =   _10100000B;   //GRAのコンペアマッチでTCNTクリア
                                         //立ち上がりエッジでカウント
                                         //内部クロックΦでカウント
    ITU4.TIOR.BYTE     =   _10001000B;   //コンペアマッチによる出力禁止
    ITU4.TCNT          =   0;            //TCNTクリア
    ITU4.GRA           =   0x61a8;       //GRA
    ITU4.TIER.BYTE     =   _11111001B;   //コンペアマッチA割り込みイネーブル
    ITU4.TSR.BYTE      =   _11111000B;   //タイマステータスレジスタクリア

    ITU.TSTR.BYTE      =   _11110000B;   //タイマスタート
}

/************************************************************************
    ITU4 コンペアマッチ/インプットキャプチャA4 割込み (1ms)
************************************************************************/
#pragma regsave (intprog_imia4)
void intprog_imia4(void)
{
    //タイマステータスレジスタクリア
    ITU4.TSR.BIT.IMFA = 0;

    //LEDデータセット
    if(P2.DR.BIT.B1==0){
        P2.DR.BIT.B1 = 1;
        PB.DR.BYTE = ~((unsigned char)(DispData & 0x00ff));
        P2.DR.BIT.B0 = 0;
```

```
    }
    else{
        P2.DR.BIT.B0 = 1;
        PB.DR.BYTE = ~((unsigned char)(DispData / 0x0100));
        P2.DR.BIT.B1 = 0;
    }
}

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

    for (i=0;i<4166666/10;i++){}
}
```

HEW が生成した「intprg. c」を巻末の付録に置き換え，次のように変更します。

```
                ∫
extern void PowerON_Reset(void);
extern void intprog_imia4(void);
                ∫
void INT_IMIA4(void)    {intprog_imia4();}
                ∫
```

追加

変更

割り込みを使うので，これまでと同じく 0xFE000 番地に「CVECTBL」セクションを追加します。

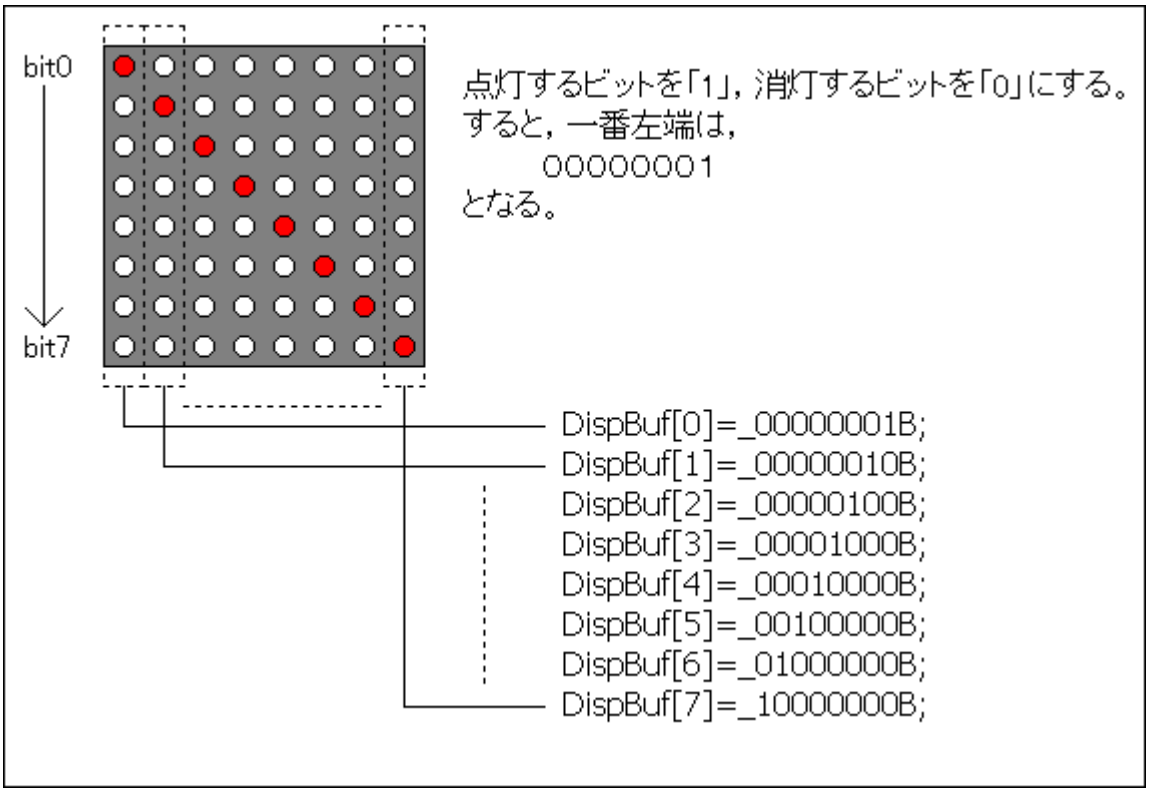
これで準備は整ったのでビルドして「Hterm」で実行します。なお，付属の CD-ROM にはあらかじめダウンロードするファイルがおさめられています。「b6092_02_kaiten. abs」をダウンロードして実行してください。

◆

次は P10～P17 につながっている 8 本のスキャンラインを使って，中央の 8×8 ドットマトリックス LED ディスプレイを制御してみましょう。

P10 が High のとき LED ディスプレイの C1 が High になります。その状態で LED ディスプレイ左端の LED の点灯パターンを PB0～PB7 に負論理で出力します。同じように P11 が High のとき C2 が High になります。その状態で次の列の点灯パターンを PB0～PB7 に負論理で出力します。あとはこれを P17(C8)まで繰り返せば LED ディスプレイを表示させることができます。

今回作るプログラムは‘DispBuf’という配列にセットしたデータを LED ディスプレイに表示します。配列の要素と表示の関係は次のとおりです。



以上のことを踏まえてコーディングしましょう。ITU のイニシャライズ関数はまったく同じです。ソースリストの一部は次のとおりです。

```
/************************************************************************
     定数エリアの定義(ROM)
************************************************************************/
//スキャンデータ
const unsigned int ScanData[10]   =    {0xff01,0xff02,0xff04,0xff08
                                       ,0xff10,0xff20,0xff40,0xff80
                                       ,0xfe00,0xfd00};

/************************************************************************
     グローバル変数の定義とイニシャライズ(RAM)
************************************************************************/
// LED表示に関係した変数 ----------------------------------------------
unsigned char  ScanCnt        =    0;   //スキャンカウンタ
unsigned char  DispBuf[10]    =    {0x00,0x00,0x00,0x00,0x00
                                   ,0x00,0x00,0x00,0x00,0x00};  //表示バッファ

/************************************************************************
     メインプログラム
************************************************************************/
void main(void)
{
     ini_io();
     ini_itu();

     DispBuf[0] = _00000001B;
     DispBuf[1] = _00000010B;
     DispBuf[2] = _00000100B;
     DispBuf[3] = _00001000B;
```

ここを変更して，いろいろなパターンを表示してみましょう。

```
    DispBuf[4] = _00010000B;
    DispBuf[5] = _00100000B;
    DispBuf[6] = _01000000B;
    DispBuf[7] = _10000000B;

    while(1){}
}

/************************************************************************
    ITU4 コンペアマッチ/インプットキャプチャA4 割込み (1ms)
************************************************************************/
#pragma regsave (intprog_imia4)
void intprog_imia4(void)
{
    //タイマステータスレジスタクリア
    ITU4.TSR.BIT.IMFA = 0;

    //LEDデータセット
    led_scan();
}

/************************************************************************
    LEDスキャン
************************************************************************/
void led_scan(void)
{
    //表示を消す
    P2.DR.BYTE = P2.DR.BYTE | _00000011B;
    P1.DR.BYTE = _00000000B;
    PB.DR.BYTE = _11111111B;

    //表示データセット
    PB.DR.BYTE = ~DispBuf[ScanCnt];

    //スキャン信号出力
    P1.DR.BYTE = (unsigned char)(ScanData[ScanCnt] & 0x00ff);
    P2.DR.BYTE = P2.DR.BYTE & (unsigned char)(ScanData[ScanCnt] / 0x0100);

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

「intprg. c」の内容は先ほどと同じです。セクション指定も変更します。なお，付属の CD-ROM にはあらかじめダウンロードするファイルがおさめられています。「b6092_03_line. abs」をダウンロードして実行してください。

◆

次はLEDディスプレイに00～FFをカウントアップして表示します。このプログラムのポイントは，数字の表示パターン(キャラクタデータ)を配列「LedDispData」にあらかじめ定義しておいて，カウント値に応じてそのデータをDispBufにセットするところです。

```
/*************************************************************************
     定数エリアの定義(ROM)
*************************************************************************/
//スキャンデータ
const unsigned int ScanData[10]   =    {0xff01,0xff02,0xff04,0xff08
                                       ,0xff10,0xff20,0xff40,0xff80
                                       ,0xfe00,0xfd00};
//キャラクタデータ(4×8)
const unsigned char LEDDispData[][4] = {
     {0x00,0xff,0x81,0xff},   // 0
     {0x00,0x02,0xff,0x00},   // 1
     {0x00,0xf1,0x91,0x9f},   // 2
     {0x00,0x89,0x89,0xff},   // 3
     {0x00,0x1f,0x10,0xff},   // 4
     {0x00,0x8f,0x89,0xf9},   // 5
     {0x00,0xff,0x89,0xf9},   // 6
     {0x00,0x0f,0x01,0xff},   // 7
     {0x00,0xff,0x89,0xff},   // 8
     {0x00,0x9f,0x91,0xff},   // 9
     {0x00,0xff,0x11,0xff},   // A
     {0x00,0xff,0x90,0xf0},   // B
     {0x00,0xff,0x81,0x81},   // C
     {0x00,0xf0,0x90,0xff},   // D
     {0x00,0xff,0x89,0x89},   // E
     {0x00,0xff,0x09,0x09},   // F
};
/*************************************************************************
     メインプログラム
*************************************************************************/
void main(void)
{
     ini_io();
     ini_itu();

     while(1){
          PA.DR.BYTE = Count;

          DispBuf[0] = LEDDispData[Count / 0x10][0];
          DispBuf[1] = LEDDispData[Count / 0x10][1];
          DispBuf[2] = LEDDispData[Count / 0x10][2];
          DispBuf[3] = LEDDispData[Count / 0x10][3];
          DispBuf[4] = LEDDispData[Count & 0x0f][0];
          DispBuf[5] = LEDDispData[Count & 0x0f][1];
          DispBuf[6] = LEDDispData[Count & 0x0f][2];
          DispBuf[7] = LEDDispData[Count & 0x0f][3];

          Count++;
          wait();
     }
}
```

Count の値に応じて，表示パターンをセットする。

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。「b6092_04_count．abs」をダウンロードして実行してください。

■ 練習問題

今のプログラムの応用です。SW1が押されたら「Count」をインクリメント(+1)，SW2が押されたら「Count」をデクリメント(-1)，SW3が押されたら「Count」を左ローテートして，その値を中央のドットマトリックスLEDに16進数で表示してください。(解答例は「b6092_05_sw_count」)

3. ルーレットの製作

ダイナミック点灯の考え方がわかったところで，応用プログラムを考えてみましょう。この項ではルーレットを作ってみます。

■ 仕様

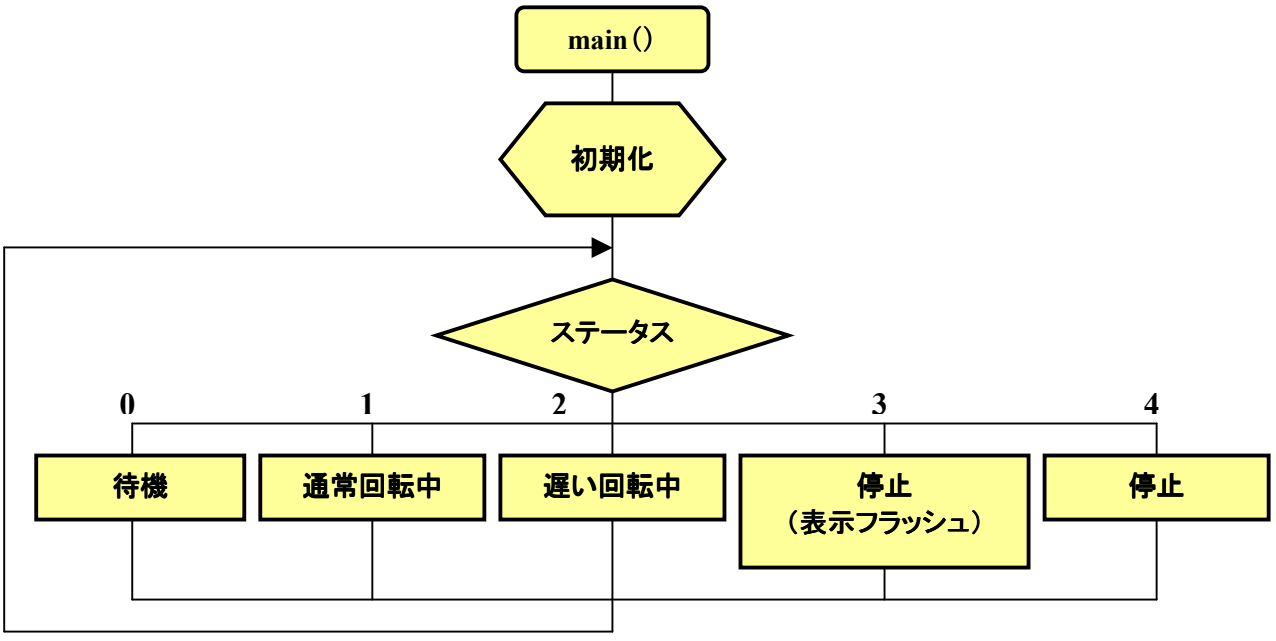
周囲に 12 個の LED がありますので，何かスイッチが押されたら時計回りに 1 個づつ点灯させます。さらに何かスイッチが押されたらゆっくり回り始め，ある程度の時間が経過したら停止，数字を決定します。LED ディスプレイには周囲の LED に対応した数字を表示します。また，スイッチが押されてゆっくり回り始めてから停止するまでの時間はランダムに変化するものとします。

なお，最初にスイッチが押されるまでの間は，スタンバイ表示として周囲の LED を交互に点滅させます。

■ プログラム

メインルーチンでルーレットの制御を行ないます。ステータス‘RouletteStatus’に応じて待機状態から通常回転…停止まで順番に移行させます。移行するタイミングはスイッチ入力だったり，時間だったりします。(スイッチを押したら回転を始める，ある程度の時間が経過したら停止する，とか)

LED の回転速度は ITU チャネル 2 割込みを利用したソフトウェアタイマで設定します。割込みは 10ms ごとにかけます。この中でソフトウェアタイマ処理を行ないます。



ゲーム本体は roulette 関数にまとめました。

```
/*************************************************************************
     ルーレット
*************************************************************************/
void roulette(void)
{
     switch(RouletteStatus){
          //待機 ----------------------------------------------------------
          case 0:
               RouletteStopCnt++; if (RouletteStopCnt>3) {RouletteStopCnt = 0;}
               if (TimT1.Status==0) {TimT1.Status = 1;}
               if (TimT1.Status==3){
                    TimT1.Status = 1;
                    if ((DispBuf[8]&0x01)==0){
                         DispBuf[8] = 0x55; DispBuf[9] = 0x05;
                    }
                    else{
                         DispBuf[8] = 0xaa; DispBuf[9] = 0x0a;
                    }
               }
               if ((SwData4 & 0x38)!=0){ //何かスイッチが押されたら次のステージへ
                    RouletteStatus = 1;
                    TimT1.Status = 0;
                    TimT0.Status = 1;
                    SwData4 = 0;
               }
               break;
          //通常回転中 ----------------------------------------------------
          case 1:
               RouletteStopCnt++; if (RouletteStopCnt>3) {RouletteStopCnt = 0;}
               if (TimT0.Status==3){
                    TimT0.Status = 1;
                    set_disp_data(RouletteData);
                    DispBuf[8] = (unsigned char)(RouletteDisp & 0x00ff);
                    DispBuf[9] = (unsigned char)(RouletteDisp / 0x0100);
                    RouletteDisp = RouletteDisp << 1;
                    if ((RouletteDisp & 0x1000)!=0) {RouletteDisp = (RouletteDisp | 0x0001) & 0x0fff;}
                    RouletteData++;
                    if (RouletteData>11){
                         RouletteDisp = 0x0800;
                         RouletteData = 0;
                    }
               }
               if ((SwData4 & 0x38)!=0){ //何かスイッチが押されたら次のステージへ
                    RouletteStatus = 2;
                    TimT0.Status = 0;
                    TimT1.Status = 1;
                    RouletteStopCnt = RouletteStopCnt + 3;     //あといくつ進んだら停止か
                    SwData4 = 0;
               }
               break;
          //遅い回転中 ----------------------------------------------------
          case 2:
               if (TimT1.Status==3){
                    RouletteStopCnt--;
                    if (RouletteStopCnt!=0){   //次の表示に進む
                         TimT1.Status = 1;
                         set_disp_data(RouletteData);
                         DispBuf[8] = (unsigned char)(RouletteDisp & 0x00ff);
                         DispBuf[9] = (unsigned char)(RouletteDisp / 0x0100);
```

```
                        RouletteDisp = RouletteDisp << 1;
                        if ((RouletteDisp & 0x1000)!=0) {(RouletteDisp = RouletteDisp | 0x0001) & 0x0fff;}
                        RouletteData++;
                        if (RouletteData>11){
                             RouletteDisp = 0x0800;
                             RouletteData = 0;
                        }
                   }
                   else{//停止
                        TimT1.Status = 0;
                        set_disp_data(RouletteData);
                        DispBuf[8] = (unsigned char)(RouletteDisp & 0x00ff);
                        DispBuf[9] = (unsigned char)(RouletteDisp / 0x0100);
                        RouletteFlash = 20;
                        RouletteStatus = 3;
                   }
              }
              break;
         //停止(表示フラッシュ) ----------------------------------------
         case 3:
              if (TimT0.Status==0) {TimT0.Status=1;}
              if (TimT0.Status==3){
                   RouletteFlash--;
                   if (RouletteFlash==0){     //次のステージへ
                        TimT0.Status = 0; TimT2.Status = 1; RouletteStatus = 4; SwData4 = 0;
                   }
                   else{
                        TimT0.Status = 1;
                   }

                   set_disp_data(RouletteData);
                   DispBuf[8] = (unsigned char)(RouletteDisp & 0x00ff);
                   DispBuf[9] = (unsigned char)(RouletteDisp / 0x0100);
                   if ((RouletteFlash&0x01)==0)     {DispFlag = 1;} //通常表示
                   else                             {DispFlag = 2;} //反転表示
              }
              break;
         //停止 ----------------------------------------------------------
         case 4:
              if (TimT2.Status==3){//時間が来たら次のステージへ
                   TimT2.Status = 0;
                   RouletteStatus = 0;
              }
              if ((SwData4 & 0x38)!=0){  //何かスイッチが押されたら回転スタート
                   RouletteStatus = 1;
                   TimT2.Status = 0;
                   TimT0.Status = 1;
                   SwData4 = 0;
              }
              break;
    }
}
```

このプログラムは，ソフトウェアタイマに ITU2 のコンペアマッチ A 割り込みを，LED スキャンとスイッチ入力に ITU4 のコンペアマッチ A 割り込みを使っています。ITU は複数のチャネルを使ってもイニシャライズの考え方はまったくかわりません。イニシャライズ関数は次のとおりです。

```
/************************************************************************
      ITUの初期化
************************************************************************/
void ini_itu(void)
{
      ITU.TSTR.BYTE         =     _11100000B;     //タイマ停止

                                                  //チャネル2
      ITU2.TCR.BYTE         =     _10100010B;     //コンペアマッチAでTCNTクリア
                                                  //立ち上がりエッジでカウント
                                                  //内部クロックΦ/4でカウント
      ITU2.TIOR.BYTE        =     _10001000B;     //コンペアマッチによる出力禁止
      ITU2.TCNT             =     0;              //TCNTクリア
      ITU2.GRA              =     0xf424;         //GRA
      ITU2.TIER.BYTE        =     _11111001B;     //コンペアマッチA割り込みイネーブル
      ITU2.TSR.BYTE         =     _11111000B;     //タイマステータスレジスタクリア

                                                  //チャネル4
      ITU4.TCR.BYTE         =     _10100000B;     //GRAのコンペアマッチでTCNTクリア
                                                  //立ち上がりエッジでカウント
                                                  //内部クロックΦでカウント
      ITU4.TIOR.BYTE        =     _10001000B;     //コンペアマッチによる出力禁止
      ITU4.TCNT             =     0;              //TCNTクリア
      ITU4.GRA              =     0x61a8;         //GRA
      ITU4.TIER.BYTE        =     _11111001B;     //コンペアマッチA割り込みイネーブル
      ITU4.TSR.BYTE         =     _11111000B;     //タイマステータスレジスタクリア

      ITU.TSTR.BYTE         =     _11110100B;     //タイマスタート
}
```

HEW が生成した「intprg. c」の内容を巻末の付録と置き換え，次のように変更します。

```
extern void PowerON_Reset(void);
extern void intprog_imia2(void);
extern void intprog_imia4(void);

void INT_IMIA2(void) {intprog_imia2();}

void INT_IMIA4(void) {intprog_imia4();}
```

追加

変更

変更

割り込みを使うので，これまでと同じく 0xFE000 番地に「CVECTBL」セクションを追加します。

これで準備は整ったのでビルドして「Hterm」で実行します。なお，付属の CD-ROM にはあらかじめダウンロードするファイルがおさめられています。「roulette. abs」をダウンロードして実行してください。

第11章 LCD に表示する

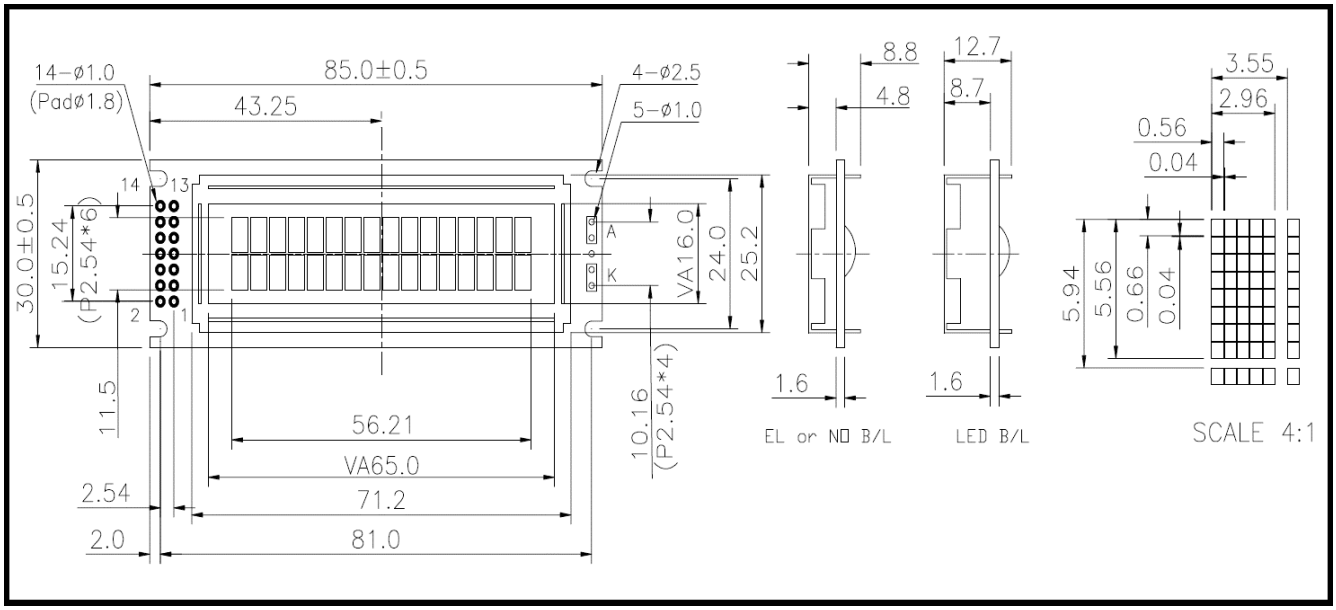
前の章では表示デバイスとしてドットマトリックス LED を使いましたが，文字を表示するとなると必要なパターンを自分で作らないといけないので手間がかかります。また，ダイナミックスキャンで表示するので，常に表示のためにマイコンのパワーが一部使われます。この章では，比較的文字の種類が豊富で表示桁数の大きな，キャラクタタイプの LCD モジュールを TK-3052 につないでみます。

1. LCD モジュールについて

今回使用するのは，秋月電子で入手できる SUNLIKE 製 16 文字 2 行キャラクタタイプの LCD モジュール「SC1602B」です。キャラクタタイプの LCD モジュールには LCD 制御用のコントロール IC が実装されています。そのため，イニシャライズさえすれば，後はマイコンから文字データをセットするだけで簡単に表示されます。キャラクタタイプの LCD モジュールに使われている LCD 制御用のコントロール IC は比較的互換性があり，他社の LCD モジュールでもプログラムを変更することなく動作することが多いです。

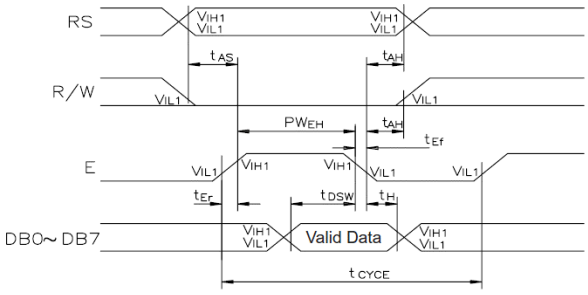
SC1602B の仕様書の抜粋(ピンアサインと外寸図，AC 特性，タイミングチャート)を下記に示します。なお，詳細資料は SUNLIKE 社のホームページ(http://www.lcd-modules.com.tw/)から入手してください。

NO.	SYMBOL	FUNCTION	NO.	SYMBOL	FUNCTION
1	VDD	Supply Voltage	9	DB2	Data Bit 2
2	Vss	Supply Ground	10	DB3	Data Bit 3
3	Vo	Contrast Adj.	11	DB4	Data Bit 4
4	RS	Register Select	12	DB5	Data Bit 5
5	R/W	Read/Write	13	DB6	Data Bit 6
6	E	Enable Signal	14	DB7	Data Bit 7
7	DB0	Data Bit 0			
8	DB1	Data Bit 1			



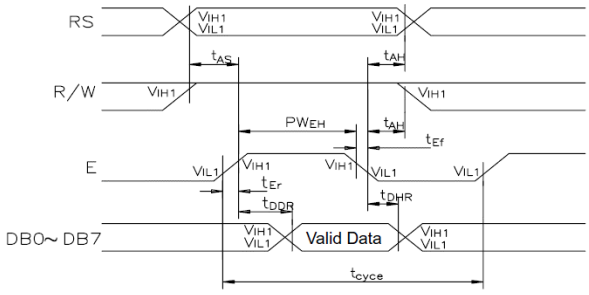
ITEM		SYMBOL	MIN	MAX	UNIT
Enable Cycle Time		t_{CYCE}	500	-	ns
Enable Pulse Width	"High Level"	P_{WEH}	230	-	ns
Enable Rise/Fall Time		t_{ER}, t_{Ef}	-	20	ns
Address Set-up Time	RS,R/$\overline{W}$ to E	t_{AS}	40	-	ns
Address Hold Time		t_{AH}	10	-	ns
Data Set-up Time		t_{DSW}	80	-	ns
Data delay Time		t_{DDR}	-	160	ns
Data Hold Time (Writing)		t_H	10	-	ns
Data Hold Time (Reading)		t_{DHR}	5	-	ns
Clock Oscillation Frequency		f_{OSC}	270 (TYP.)		KHz

FIG.1 WRITE OPERATION



(Write Data from MPU to MODULE)

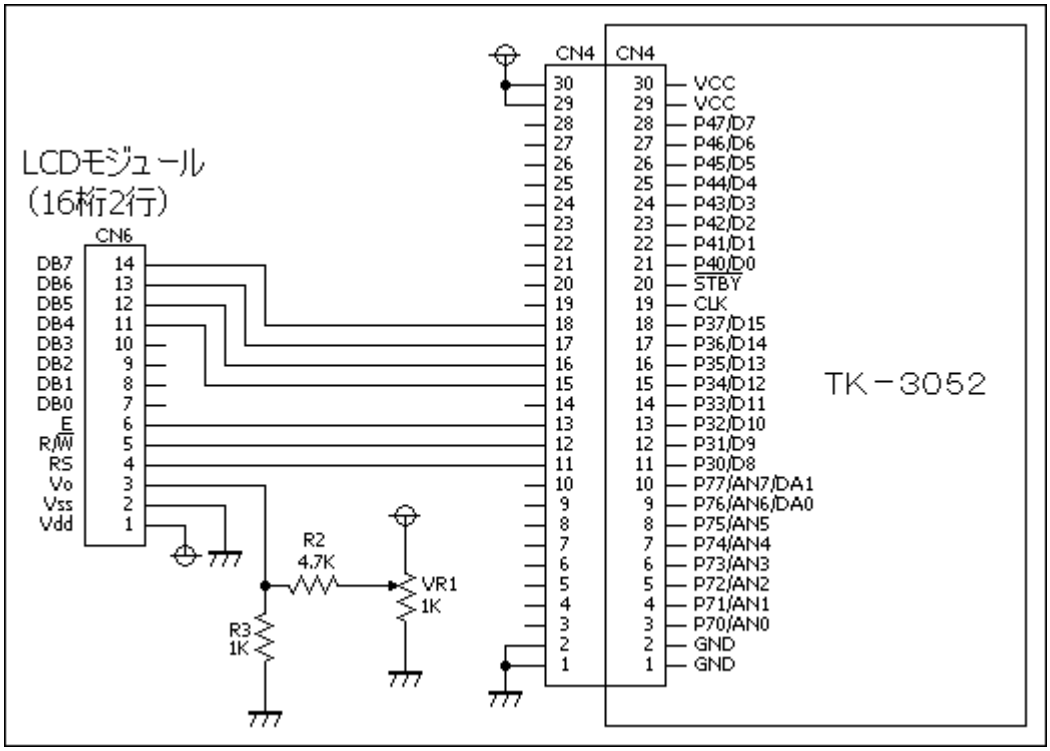
FIG.2 READ OPERATION



(Read Data from MODULE to MPU)

2. TK-3052 との接続

前ページのピンアサインを見ると，SC1602B の制御のための信号は「RS，R/W，E，DB0～DB7」の 11 本です。ただし，SC1602B はデータ転送幅が 2 種類あり，8 ビット転送のときは DB0～DB7 の 8 本を使いますが，4 ビット転送のときは DB4～DB7 の 4 本で接続することもできます。そのかわり，上位 4 ビット，下位 4 ビットの順にデータを二回に分けて転送する必要があります。ワンチップマイコンで使う場合，できるだけ I/O を節約して使いたいので，今回は 4 ビット転送を使います。それで，必要な I/O は 7 本です。次のような回路を考えてみました。



3. インストラクション

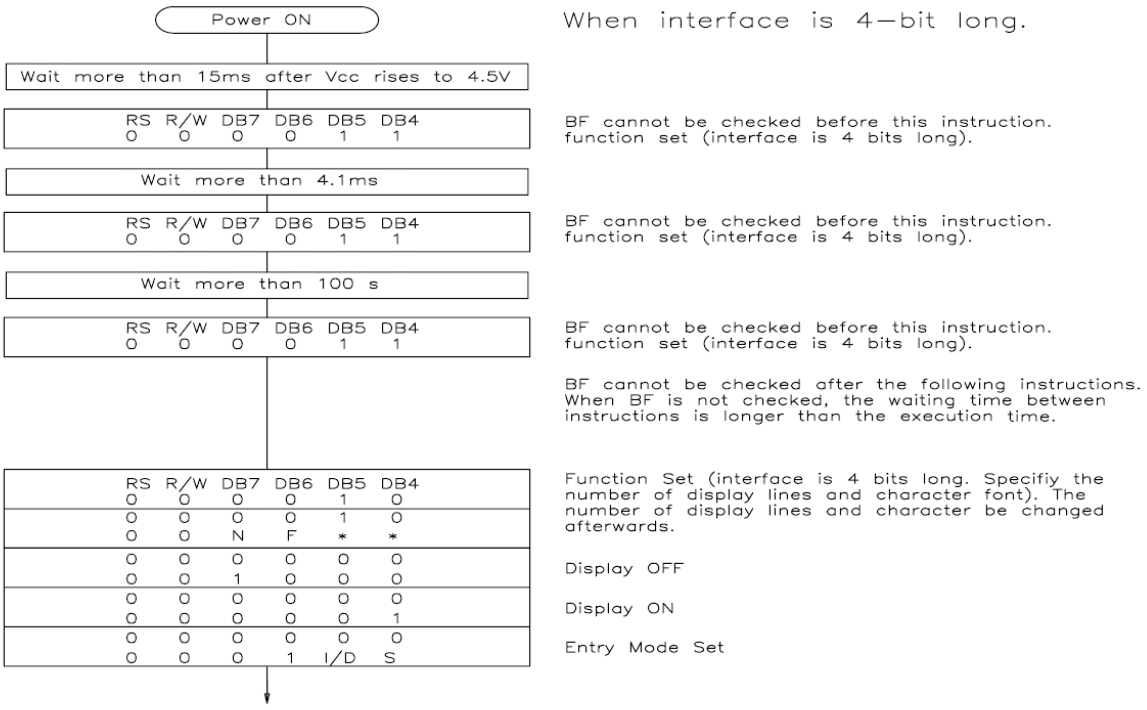
インストラクション		コード										機能
		RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
1	表示クリア	0	0	0	0	0	0	0	0	0	1	全表示クリア後，カーソルをホーム位置に戻す。
2	カーソルホーム	0	0	0	0	0	0	0	0	1	*	カーソルをホーム位置に戻す。シフトしていた表示も元に戻る。DDRAM の内容は変化しない。
3	エントリーモードセット	0	0	0	0	0	0	0	1	I/D	S	カーソルの進む方向，表示をシフトさせるか指定。 I/D=1:インクリメント，カーソル右移動 I/D=0:デクリメント，カーソル左移動 S=1:シフトさせる S=0:シフトしない
4	表示オン/オフコントロール	0	0	0	0	0	0	1	D	C	B	表示のオン/オフ，カーソルのオン/オフ，カーソル位置のブリンクのオン/オフの指定。 D=1/0:表示オン/オフ C=1/0:カーソルオン/オフ B=1/0:ブリンクオン/オフ
5	カーソル/表示シフト	0	0	0	0	0	1	S/C	R/L	*	*	DDRAM の内容を変えずに，カーソルの移動と表示シフト。 S/C=1:表示シフト S/C=0:カーソル移動 R/L=1:右シフト R/L=0:左シフト
6	ファンクションセット	0	0	0	0	1	DL	N	F	*	*	インターフェースデータ長，デューティ，文字フォントの設定。 DL=1/0:8 ビット/4 ビット N=1:1/16 デューティ N=0:1/8，1/11 デューティ F=1:5×10 ドット F=0:5×7 ドット
7	CGRAM アドレスセット	0	0	0	1	ACG						CGRAM アドレスセット。以後送受するデータは CGRAM のデータ。 ACG:CGRAM アドレス
8	DDRAMアドレスセット	0	0	1	ADD							DDRAM アドレスセット。以後送受するデータは DDRAM のデータ。 ADD:DDRAM アドレス
9	BF/アドレスセット	0	1	BF	AC							内部動作中，アドレスカウンタの値。 BF=1/0:内部動作中/インストラクション受付可 AC:アドレス値
10	CGRAM,DDRAM へのデータ書き込み	1	0	書き込みデータ								データ書き込み。
11	CGRAM,DDRAM からのデータ読み出し	1	1	読み出しデータ								データ読み出し。

実行時間 : 1&2 → 1.52ms / 3～11 → 37μs
＊ → 無効ビット(0 でも 1 でも可)

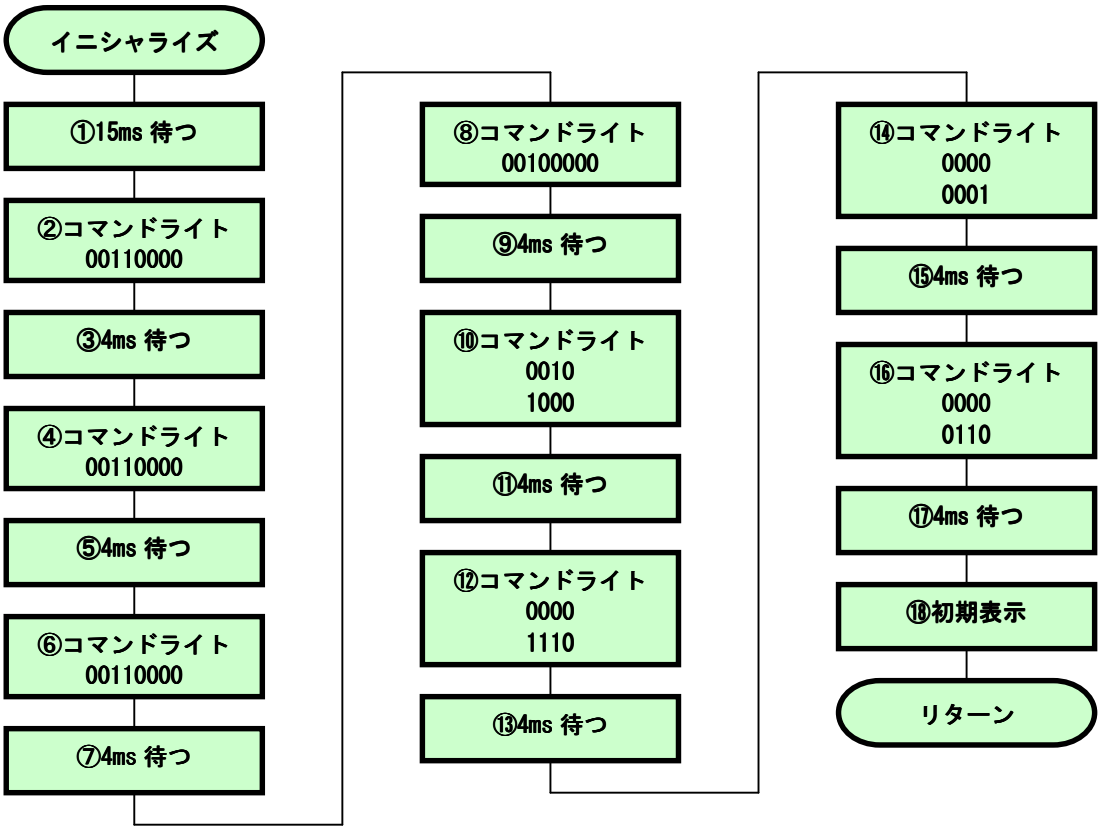
4. イニシャライズ

SC1602B は電源オン時に自動的に初期化されることになっています。しかし，初期化に失敗したり，電源オンの状態で何度もイニシャライズしたりする場合を考えると，プログラムできちんとイニシャライズする方が安全です。仕様書には 4 ビットインターフェースのときのイニシャライズ例が次のように記載されています。

(1) When interface is 4 bits



これを参考に，次のようなフローチャートを考えてみました。



最初に 15ms 待った後，コマンド「00110000」を 3 回ライトしています。これは 8 ビットインターフェース長にセットするコマンドですが，なぜこのようなことをしているのでしょうか。

もし電源オン時の自動的な初期化に失敗していたとすると，この時点で LCD モジュールがどんな状態かは不定です。考えられる状態は(1)8 ビットインターフェースのデータ待ち，(2)4 ビットインターフェースの上位 4 ビットデータ待ち，(3)4 ビットインターフェースの下位 4 ビットデータ待ち，の 3 種類です。

(1)の状態だったときコマンド「00110000」を 3 回ライトしても状態は変わらず，⑥が終わった時点で 8 ビットインターフェース長にセットされています。(2)の状態だったときはコマンド「00110000」を 2 回ライトした時点(④が終わった時点)で 8 ビットインターフェースに切り替わり，⑥が終わった時点で 8 ビットインターフェース長にセットされています。(3)の状態だったときはコマンド「00110000」を 1 回ライトした時点(②が終わった時点)で 4 ビットインターフェースの上位 4 ビットデータ待ちになり，コマンド「00110000」をあと 2 回ライトした時点(⑥が終わった時点)で 8 ビットインターフェース長にセットされます。つまり，電源オンでどのような状態になっていたとしても，⑥が終わった時点で 8 ビットインターフェース長にセットされた状態になります。ここで，コマンド「00100000」をライトすると 4 ビットインターフェース長に切り替わり(⑧が終わった時点)，ここから本来の SC1602B のイニシャライズを行ないます。

⑩はファンクションセットです。4 ビットインターフェース長，1/16 デューティ，文字フォントは 5×7 ドットにセットします(デューティと文字フォントは SC1602B のときはこの値にセットする)。ファンクションセットは他のコマンドより優先して実行しないといけません。

⑫は表示オン/オフコントロールです。表示オン，カーソルオン，ブリンクオフにセットします。

⑭は表示クリアです。全表示クリア後，カーソルをホーム位置に戻します。

⑯はエントリーモードセットです。カーソルの進む方向や表示をシフトするか指定します。カーソル右移動，表示のシフトはしないにセットします。

⑱で初期表示データをセットします。

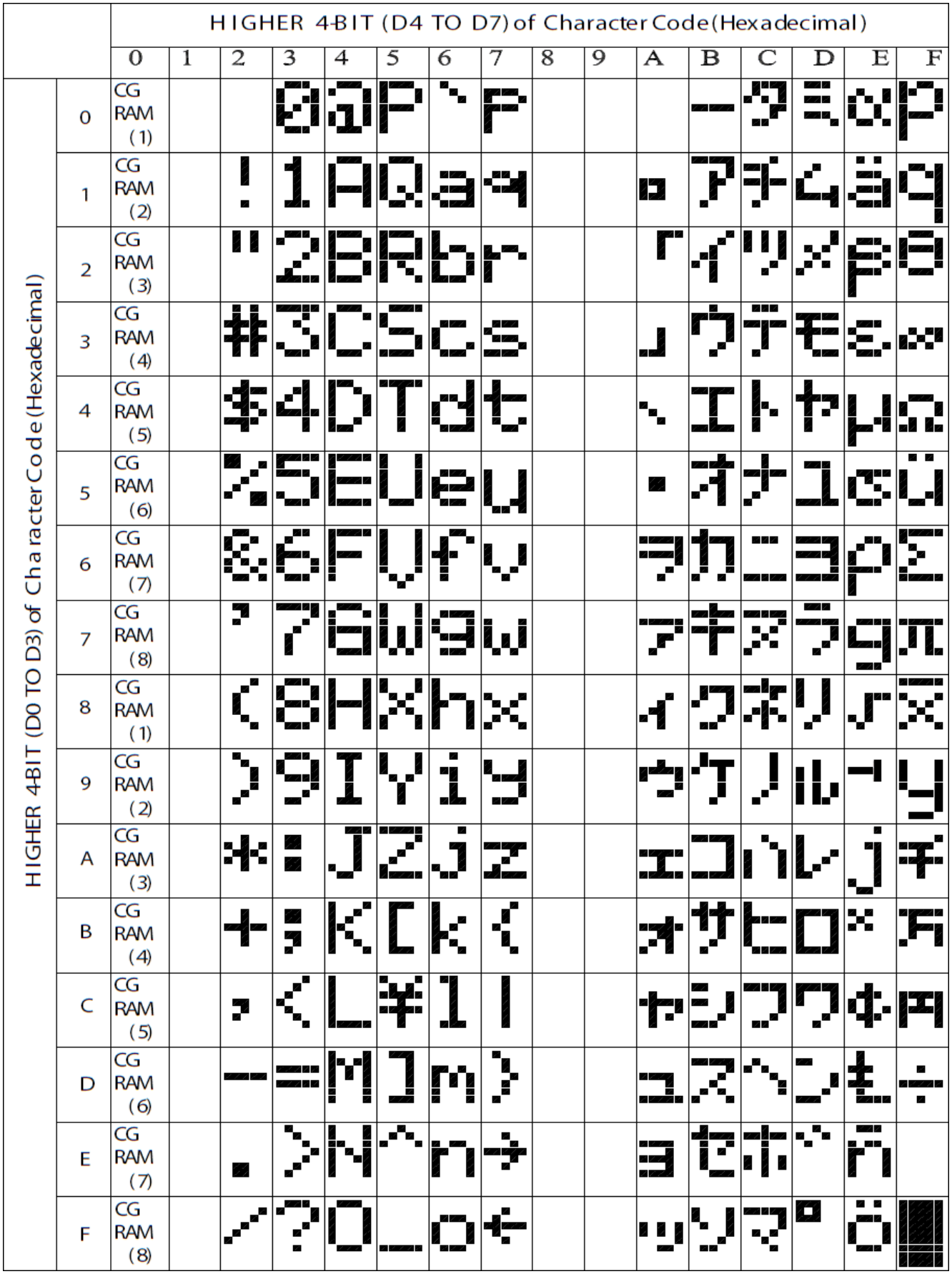
5. テキストの表示

SC1602B は 16 桁 2 行ですが，それぞれの文字位置にアドレスが割り付けられています。そのアドレスにデータをセットするとテキストが表示されます。アドレスの割り付けは次のようになっています(数字は 16 進数)。

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

気をつける点は 2 行目が 40h 番地から始まっていることです。プログラムではメモリ上に LCD の表示バッファを定義し(LCDBuff[0]～[15]が 00～0F 番地，LCDBuff[16]～[31]が 40～4F 番地に相当)，ここに表示データをセットした後 SC1602B にまとめて転送します。次のようなフローチャートになります。

なお，セットする表示データと，表示される文字の関係は次の表のとおりです。基本的にアスキーコードに準じます。



6. サンプルプログラム

これまでのことを踏まえた上で，SC1602B をイニシャライズして，文字を表示するプログラムを考えてみましょう。ソースリストは次のようになりました。

```
/************************************************************************/
/*                                                                      */
/*  FILE        :lcd01.c                                                */
/*  DATE        :Thu, Jul 22, 2010                                      */
/*  DESCRIPTION :Main Program                                           */
/*  CPU TYPE    :H8/3052F                                               */
/*                                                                      */
/*  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
#include "binary.h"     //Cで2進数を使うための定義

/************************************************************************
    定数の定義（直接指定）
************************************************************************/
// LCD ------------------------------------------------------------------
#define       LCD_PORT      P3.DR.BYTE
#define       LCD_RS        P3.DR.BIT.B0
#define       LCD_RW        P3.DR.BIT.B1
#define       LCD_E         P3.DR.BIT.B2

#define       FUNC0         0x30 // Reset Function
#define       FUNC1         0x20 // Function Set
#define       FUNC2         0x28 // Function
#define       D_CLR         0x01 // Display clear
#define       D_ON          0x0e // Display ON
#define       EM_SET        0x06 // Entry Mode Set
#define       C_HOME        0x02 // Cursor home
#define       C_2L          0xc0 // Cursor to 2'nd Line

/************************************************************************
    定数エリアの定義(ROM)
************************************************************************/
// LCD ------------------------------------------------------------------
// イニシャライズコマンド
const unsigned charCMD[ ] = {FUNC1,FUNC2,D_ON,D_CLR,EM_SET};
// 初期表示
const unsigned charMOJI[ ] = "TK-3052 toyolinx@va.u-netsurf.jp";

/************************************************************************
    グローバル変数の定義とイニシャライズ(RAM)
************************************************************************/
// LCD ------------------------------------------------------------------
unsigned char  LCDBuff[33];       // 表示用文字バッファ
```

```
/************************************************************************
    関数の定義
************************************************************************/
void     ini_io(void);
void     main(void);

// LCD ------------------------------------------------------------------
void     ini_lcd(void);
void     lcd_dsp(unsigned char *);
void     lcd_out4(unsigned char);
void     lcd_out8(unsigned char);
void     delay(short);

/************************************************************************
    メインプログラム
************************************************************************/
void main(void)
{
    ini_io();
    ini_lcd();

    while(1){
        PA.DR.BYTE = 0x88;
        delay(1000);
        PA.DR.BYTE = 0x44;
        delay(1000);
        PA.DR.BYTE = 0x22;
        delay(1000);
        PA.DR.BYTE = 0x11;
        delay(1000);
    }
}

/************************************************************************
    I/Oポートの初期化
************************************************************************/
void ini_io(void)
{
    P3.DDR        =    _11111111B;    //Port3を出力に設定
    P3.DR.BYTE    =    _00000000B;    //初期出力値

    PA.DDR        =    _11111111B;    //PortAを出力に設定
    PA.DR.BYTE    =    _00000000B;    //初期出力値
}

/************************************************************************
    LCDのイニシャライズ
************************************************************************/
void ini_lcd(void)
{
    short i;
    unsigned char *cp;

    delay (150);                    // 15000us=15ms待つ
```

```
    for (i=0; i<3; i++){              // FUNC0を3回ライト
        LCD_RS =0;                    // RSをOFF インストラクションコード
        lcd_out8(FUNC0);              // ライト
        delay(40);                    // 4000us=4ms待つ
    }
    cp = CMD;                         // ポインタ設定
    LCD_RS = 0;                       // RS OFF
    lcd_out8(*cp++);                  // FUNC1をライト
    delay(40);                        // 4000us=4ms待つ
    for (i=0; i<4; i++){              // FUNC2,D_ON,EM_SET,D_CLRをライト
        LCD_RS = 0;                   // RS信号OFF
        lcd_out4(*cp++);              // ライト
        delay(40);                    // 4000us=4ms待つ
    }

    for (i=0; i<33; i++){             // 表示文字初期設定
        LCDBuff[i] = MOJI[i];         // LCDBuffにセット
    }
    lcd_dsp(LCDBuff);                 // LCD表示
}

/*****************************************************************************
    LCD表示
*****************************************************************************/
void lcd_dsp(unsigned char *buff)
{
    int i;

    LCD_RS = 0;                       // RS OFF
    lcd_out4(C_HOME);                 // カーソル・ホーム
    delay(40);                        // 4000us=4ms待つ
    for (i=0; i<16; i++){             // 1行  16文字
        LCD_RS = 1;                   // RS信号ON データ
        lcd_out4(*buff++);            // ライト
        LCD_RS = 0;                   // RS 信号 OFF
    }
    lcd_out4(C_2L);                   // カーソルを 2 行目に移動
    delay(40);                        // 4000us=4ms待つ
    for (i=0; i<16; i++){             // 1行  16文字
        LCD_RS = 1;                   // RS信号ON
        lcd_out4(*buff++);            // ライト
        LCD_RS = 0;                   // RS信号OFF
    }
}

/*****************************************************************************
    LCDデータライト（4ビットインターフェース）
*****************************************************************************/
void lcd_out4(unsigned char c)
{
    unsigned char uc;

    LCD_E = 1;                        // E信号ON
    uc = LCD_PORT & 0x0f;             // LCD_PORTの現在値
    LCD_PORT = (c & 0xf0) | uc;       // 上位4ビット出力
```

```
    delay(4);                       // 400us待つ
    LCD_E = 0;                      // E信号OFF
    delay(4);                       // 400us待つ

    LCD_E = 1;                      // E信号ON
    uc = LCD_PORT & 0x0f;           // LCD_PORTの現在値
    LCD_PORT = (c * 0x10) | uc;     // 下位4ビット出力
    delay(4);                       // 400us待つ
    LCD_E = 0;                      // E信号OFF
    delay(4);                       // 400us待つ
}

/*************************************************************************
    LCDデータライト（4ビットインターフェース時の8ビットライト）
*************************************************************************/
void lcd_out8(unsigned char c)
{
    unsigned char uc;

   LCD_E = 1;                       // E信号ON
   delay(4);                        // 400us待つ
    uc = LCD_PORT & 0x0f;           // LCD_PORTの現在値
    LCD_PORT = (c & 0xf0) | uc;     // 上位4ビット出力
   delay(4);                        // 400us待つ
   LCD_E = 0;                       // E 信号OFF
   delay(4);                        // 400us待つ
}

/*************************************************************************
    ディレー（i×100us）
*************************************************************************/
void delay(short i)
{
    unsigned int j,k;

    for (j=0; j<i; j++){
        for (k=0; k<410; k++){
        }
   }
}
```

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。「lcd01.abs」をダウンロードして実行してください。LCD に下記のように表示されます。

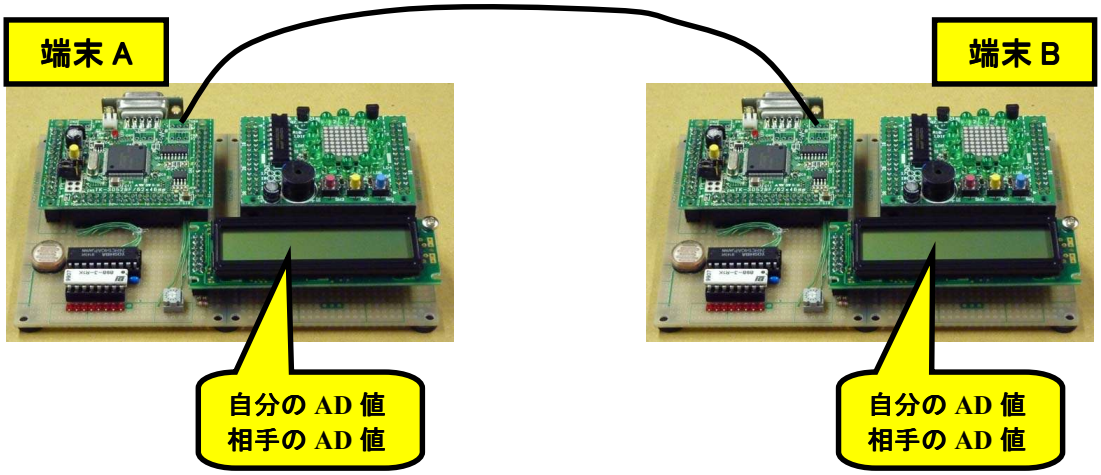
第12章 RS-485を使ったネットワークの実験(1)

1. 実験システムの構成
2. プロトコルの検討
3. 通信プログラム

SCI の使い方の章で RS-485 について調べました。この章では，さらに一歩進んで，RS-485 を利用したマイコン同士の通信を実験します。

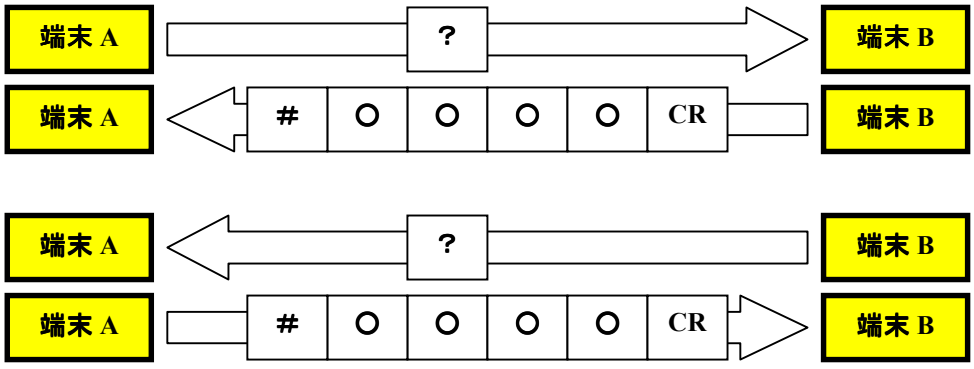
1. 実験システムの構成

TK-3052 版マイコン事始めキットを 2 台使って RS-485 で接続します。CDS の電圧を AD 変換し，自分の LCD に表示するとともに，相手の LCD にも表示します。今回は，複数台の接続ではなく，1 対 1 の接続で考えます。TK-3052 版マイコン事始めキットを 2 台ご用意ください。CN9 どうしを接続します。



2. プロトコルの検討

1 対 1 の通信で，かつ，データは AD 値のみなので，プロトコルはなるべくシンプルにします。知りたい情報は相手の AD 値です。そこで，相手の AD 値を知りたいときは「？」を送信することにし，返信として AD 値が送られてくるものとします。また，「？」を受信したら自分の AD 値を返信することにします。なお，AD 値は上位桁から順番に「0」～「F」をアスキーコードで送信します（下図の「○」）。また，AD 値の先頭に「＃」を，最後に CR（0Dh）を付加します。それぞれの端末は 1 秒間隔で「？」を送信し AD 値を要求します。



さて，RS-485 の場合，送信も受信も同じ信号線を使っているので，お互いの送信データが衝突する可能性を考える必要があります。衝突した場合，「？」に対する正しい返信がいつまでも受信できないことになります。それで，「？」を送信してから 0．1 秒待っても AD 値を受信できなかった場合，次の「？」の送信までの時間をランダムに変更することにします。1 秒のうち AD 値の送信にかかる時間は約 1．56ms なので，各端末の「？」の送信のタイミングをずらせば次に送信データが衝突する確率が下がると考えられます。

3. 通信プログラム

■ 送信プログラム

送信プログラムは第 7 章とほぼ同じ考え方です。異なるのは，第 7 章では送受信を両方イネーブルにしていましたが，今回は，通常は「送信ディセーブル・受信イネーブル」にし，送信時のみ「受信ディセーブル・送信イネーブル」にします。そのため，どのタイミングでトランシーバ（MAX485 相当）を切り替えるかという問題が生じます。

「受信ディセーブル・送信イネーブル」への切り替えは送信するときとします。「受信中は切り替えない」という処理も必要に思えますが，通信失敗時はタイミングをかえて「？」を再送してきますので大きな問題になりません。

問題は「送信ディセーブル・受信イネーブル」への切り替えのタイミングです。SCI に送信データをセットしてすぐに切り替えると相手に届きません。1 ビットずつ送信するわけなので，送信が終わるまで SCI にセットしてから最短でも 260μs 必要だからです。その前に切り替えると，送信途中で送信がディセーブルになってしまします。

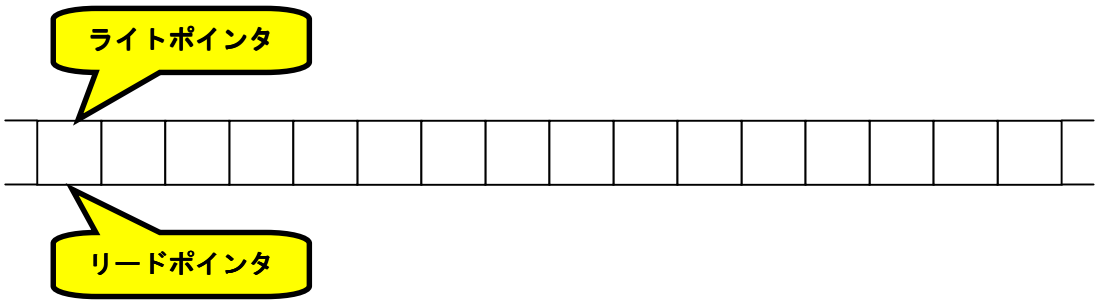
「シリアルステータスレジスタ（SSR）」をご覧下さい。ビット 2，‘TEND’が 1 になったらデータ送信が終了しています。それで，最後の送信データを SCI にセットした後 TEND が 1 になるのを待ってから「送信ディセーブル・受信イネーブル」に切り替えます。AD 値を送信する部分のソースリストは次のとおりです。

```
/*****************************************************************************
     AD値の送信
*****************************************************************************/
void send_ad(void)
{
     P9.DR.BIT.B5 = 1;     //MAX485 受信ディセーブル
     P9.DR.BIT.B4 = 1;     //MAX485 送信イネーブル

     txone('#');
     txone(hex2asc((unsigned char)((AdData & 0xf000) / 0x1000)));
     txone(hex2asc((unsigned char)((AdData & 0x0f00) / 0x0100)));
     txone(hex2asc((unsigned char)((AdData & 0x00f0) / 0x0010)));
     txone(hex2asc((unsigned char)( AdData & 0x000f          )));
     txone(CR);

     chk_tend();           //送信完了まで待つ

     P9.DR.BIT.B5 = 0;     //MAX485 受信イネーブル
     P9.DR.BIT.B4 = 0;     //MAX485 送信ディセーブル
}

/*****************************************************************************
     1文字送信（ポーリング）
------------------------------------------------------------------------------
     引数 txdata           送信データ
*****************************************************************************/
void txone(unsigned char txdata)
{
     while(SCI0.SSR.BIT.TDRE==0){}         //送信可能まで待つ
     SCI0.TDR = txdata;
     SCI0.SSR.BIT.TDRE = 0;
}

/*****************************************************************************
     送信完了の確認
*****************************************************************************/
void chk_tend(void)
{
     while(SCI0.SSR.BIT.TEND==0){}         //送信終了まで待つ
}
```

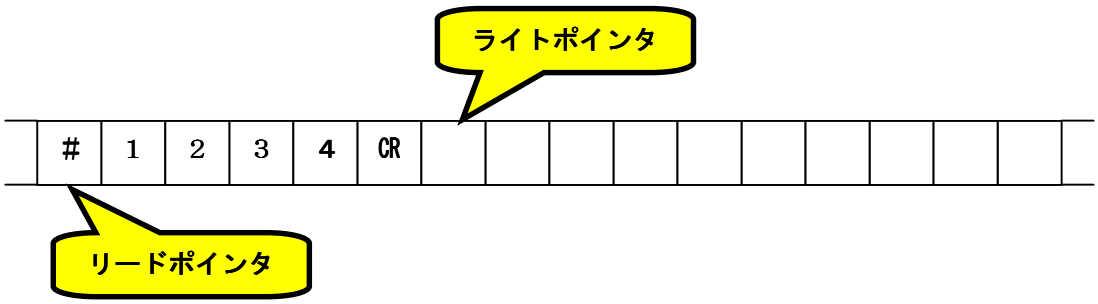
■ 受信プログラム

基本的に受信データはいつやってくるかわかりません。また，受信したデータを SCI から読み込まないうちに次のデータがきてしまうとオーバーランエラーになります。それで，受信したら割り込みをかけてデータを読み込むことを考えます。読み込んだデータはメモリにとりあえずストアし，あとから取り出して処理の続きを行ないます。このようなときに使うデータ構造を「キュー」と呼びます。

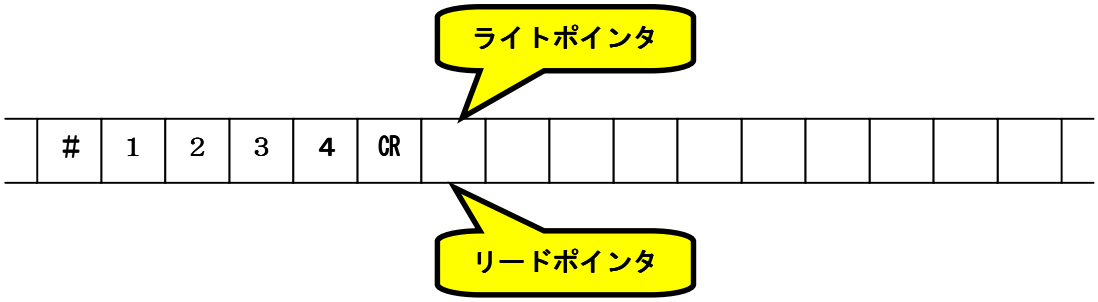
「キュー」は FIFO（First In First Out：先入れ先出し）のデータ構造です。キューを構成するために必要な要素はキュー本体と，データが格納されている先頭の位置を示すポインタ（＝次にキューから読み出す場所，リードポインタ）と，最後を示すポインタ（＝次にキューに書き込む場所，ライトポインタ）です。キューに何も入っていないときは，リードポインタとライトポインタは同じ場所を指しています。



ここで，「＃」「1」「2」「3」「4」「CR」というデータを受信しキューに格納すると次のようにライトポインタが動きます。



リードポインタとライトポインタが異なるときはキューにデータが格納されていることを示しています。そこで，キューからデータを順番に取り出します。するとリードポインタは次のように動きます。



あとはこれを繰り返していけば，何か処理をしている最中にデータを受信しても，とりあえずキューにデータを格納しておき，あとでキューからデータを取り出すことで，データを取りこぼすことなく処理を続けることができます。

さて，キュー本体は1次元配列で表現しますが，配列のサイズは決まっているので，単純にポインタを＋1すると，やがてライトポインタもリードポインタも配列の最後になってしまい，データをこれ以上格納することができなくなります。しかし，キューから取り出してしまえば，そこに格納されていたデータは必要なくなります。そこで，ポインタが配列の最後まで到達したら，ポインタを配列の先頭に戻すようにしておけば，必要なくなったエリアを再利用できます。このような配列の使い方をリングバッファと呼びます。扱うデータに比べて配列のサイズが十分大きければ，問題が生じることなく通信を続けることができます。

ではソースリストを見ていきましょう。まずはキューの定義です。配列とポインタで定義しています。また，初期値を設定する関数も必要です。(キュー:RxBuf, ライトポインタ:RxBufWrPnt, リードポインタ:RxBufRdPnt)

```
/***********************************************************************
     定数の定義（直接指定）
***********************************************************************/
//通信 ----------------------------------------------------------------
#define         RXBUF_SIZE      256   //RxBufのサイズ

/***********************************************************************
     グローバル変数の定義とイニシャライズ(RAM)
***********************************************************************/
// 通信 ---------------------------------------------------------------
unsigned char   RxBuf[RXBUF_SIZE];    //受信バッファ
unsigned char   *RxBufWrPnt;          //受信バッファライトポインタ
unsigned char   *RxBufRdPnt;          //受信バッファリードポインタ
unsigned char   *RxBufMin;            //受信バッファの最初
unsigned char   *RxBufMax;            //受信バッファの最後

/***********************************************************************
     RxBuf の初期化
***********************************************************************/
void init_rxbuf(void)
{
     RxBufRdPnt =    RxBuf;                //受信バッファリードポインタセット
     RxBufWrPnt =    RxBufRdPnt;           //受信バッファライトポインタセット
     RxBufMin   =    RxBuf;                //受信バッファの最初をセット
     RxBufMax   =    RxBuf+RXBUF_SIZE-1;   //受信バッファの最後をセット
}
```

次に受信割り込みです。データを受信すると「intprog_rxi0」関数がコールされます。この関数の中で受信データを読み込み，受信バッファ「RxBuf」にストアします。

```
/***********************************************************************
     SCI0 受信割込み
***********************************************************************/
#pragma regsave (intprog_rxi0)
void intprog_rxi0(void)
{
     put_rxbuf(SCI0.RDR); //データをRxBufにストア
     SCI0.SSR.BIT.RDRF = 0;
}

/***********************************************************************
     RxBuf にデータを格納する
-----------------------------------------------------------------------
     引数       data       RxBufに格納するデータ
-----------------------------------------------------------------------
     戻り値     OK              格納できた
                NG              エラー,バッファからあふれた
***********************************************************************/
int put_rxbuf(unsigned char data)
{
     int ret_code;   //OK or NG

     if ((RxBufRdPnt==RxBufMin && RxBufWrPnt==RxBufMax) || RxBufWrPnt==RxBufRdPnt-1)
          ret_code = NG;        //バッファサイズを越えた
     else{
          *RxBufWrPnt = data;
```

```
          RxBufWrPnt++;
          if (RxBufWrPnt>RxBufMax)
               RxBufWrPnt=RxBufMin; //ライトポインタを先頭に戻す
          ret_code = OK;
     }
     return ret_code;
}
```

メインループでは「analyze_rxcmd」関数で受信バッファにデータがストアされているかチェックします。ストアされているならデータをチェックし,「？」を受信していたら AD 値を送信(「send_ad」関数),「CR」を受信していたらコマンドを取り出し(「get_command」関数),内容をさらにチェックして受信した AD 値をセーブします(「rcv_ad」関数)。

```
/*****************************************************************************
     受信コマンドの解析
*****************************************************************************/
void analyze_rxcmd(void)
{
     unsigned char data;        //RxBufから取り出したデータ

     if (get_rxbuf(&data)==NG) return;     //受信データがない

     //正規コマンドを受信したかチェック
     switch (data){
          case '?':  //?を受信した
               send_ad();
               break;
          case CR:   //CRを受信した
               get_command(6);
               if (rcv_ad()==OK){
                    TimT1.Status = 0;
                    RcvTimOutCnt = 0;
               }
               break;
          default:   //制御コマンド以外のデータを受信した
               return;
     }
}

/*****************************************************************************
     RxBuf からデータを取り出す
-----------------------------------------------------------------------------
     引数       *pd             取り出したデータをセットするポインタ
-----------------------------------------------------------------------------
     戻り値     OK              取り出せた
                NG              エラー,バッファに何も入っていない
*****************************************************************************/
int get_rxbuf(unsigned char *pd)
{
     int ret_code;

     if (RxBufWrPnt==RxBufRdPnt)
          ret_code = NG;        //バッファに何も入っていない
     else{
          *pd = *RxBufRdPnt;
          RxBufRdPnt++;
          if (RxBufRdPnt>RxBufMax)
               RxBufRdPnt=RxBufMin; //リードポインタを先頭に戻す
          ret_code = OK;
     }
```

```
    return ret_code;
}
```

さて，コマンドを取り出す「get_command」関数ですが，「？」に対する応答は 6 バイトと決まっています。そこで，CR を受信したら，そこから 6 バイトさかのぼって受信バッファから取り出し，コマンドバッファ「RxCmd」にストアします。なお，6 バイト以外にも対応できるように，関数の引数でコマンド長を指定するようにしています。

```
/*****************************************************************************
    RxBuf からコマンドを取り出す
*****************************************************************************/
void get_command(unsigned char len)
{
    unsigned char *pa;          //RxBufのポインタ
    unsigned char *pb;          //RxCmdのポインタ
    int i;

    pa = RxBufRdPnt;
    pb = RxCmd+len-1;

    //RxBufの最後に蓄えたデータから(len)バイト RxCmdに移す
    for(i=len;i!=0;i--){
        pa--;if (pa<RxBufMin) pa=RxBufMax;
        *pb = *pa;
        pb--;
    }
}
```

AD 値をセーブする「rcv_ad」関数では，取り出されたコマンド「RxCmd」の内容を確認しています。

```
/*****************************************************************************
    AD値の受信
*****************************************************************************/
char rcv_ad(void)
{
    unsigned char i;

    //受信データチェック
    if (RxCmd[0]!='#')    {return NG;}
    for (i=1; i<5; i++){
        if          (RxCmd[i]<'0')                          {return NG;}
        else if     ((RxCmd[i]>'9') & (RxCmd[i]<'A'))       {return NG;}
        else if     (RxCmd[i]>'F')                          {return NG;}
    }

    //受信データのセーブ
    for (i=0; i<4; i++){
        RcvAD[i] = RxCmd[i+1];
    }

    return OK;
}
```

◆

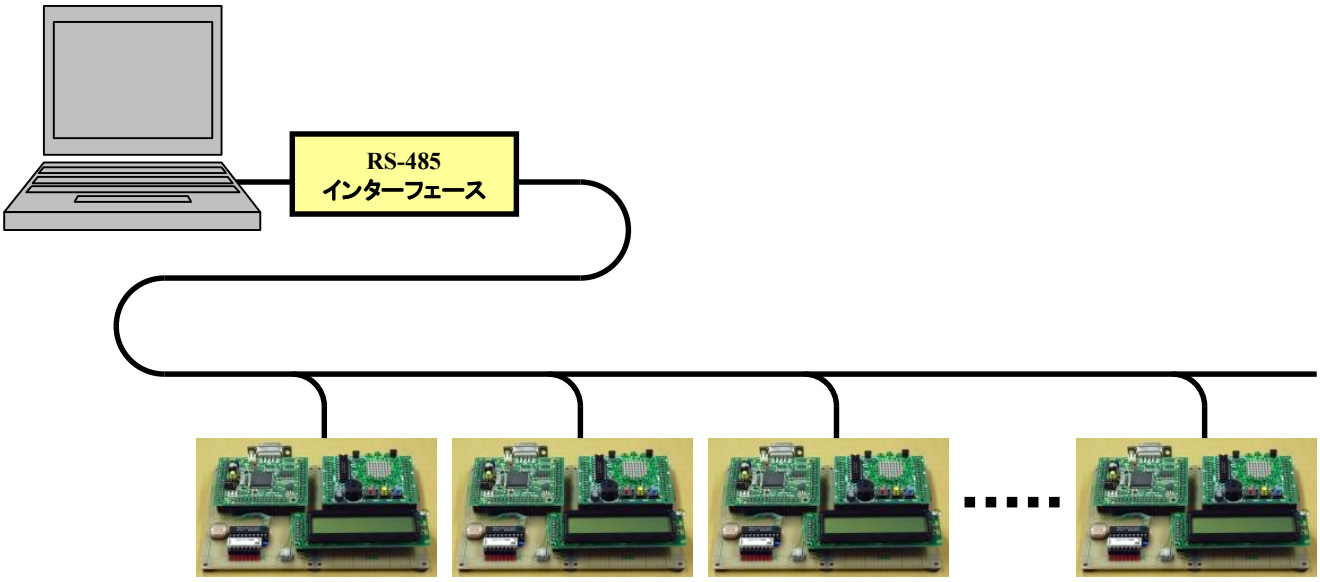
付属の CD-ROM にはあらかじめダウンロードするファイルがおさめられています。「rs485_03.abs」をダウンロードして実行してください。

第13章 RS-485を使ったネットワークの実験(2)

前の章でRS-485を使った1対1のネットワークを実験しました。この章ではさらに一歩進んで，複数の端末をRS-485のネットワークにつなぐ実験をします。

1. 実験システムの構成

パソコンに RS-485 のインターフェースを追加し親機にします。子機は TK-3052(マイコン事始め)です。子機にはそれぞれ独立したアドレス(00～FF)が割り付けられています。パソコンから，RS-485 ネットワークに複数台つながっている子機の LED(8ビット)のオン/オフを個別に制御します。



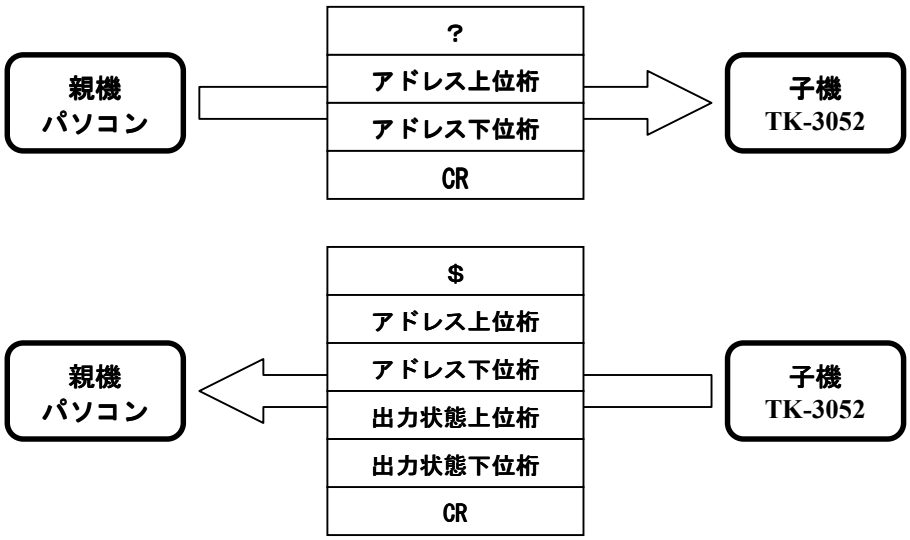
ところで，普通パソコンにRS-485インターフェースはついていません。また，追加したくても一般的に入手できるものでもありません。それで，実験ということもあり，できるだけ簡単な方法で RS-485 インターフェースを作ることにしました。株式会社秋月電子通商(http://akizukidenshi.com)の USB-シリアル変換モジュール「AE-UM232R(通販コード K-01977)」と，MAXIM の「MAX485」(相当品可，秋月電子通商でリニアテクノロジーの LTC1485(通販コード I-01869)，LTC1785(通販コード I-02791)，LTC485(通販コード I-02792)が入手可能)を組み合わせて，USB-RS485 変換モジュールを作ります。パソコンにドライバをインストールする必要はありますが，パソコンから見ると通常のシリアルポート(COM ポート)として RS-485 インターフェースで通信することができます。なお，ドライバはFTDI社のサイト(http://www.ftdichip.com)からロイヤリティフリーで入手，使用することができます。

巻末の付録にUSB-RS485変換モジュールの回路図，部品表，実装図，写真を掲載しています。

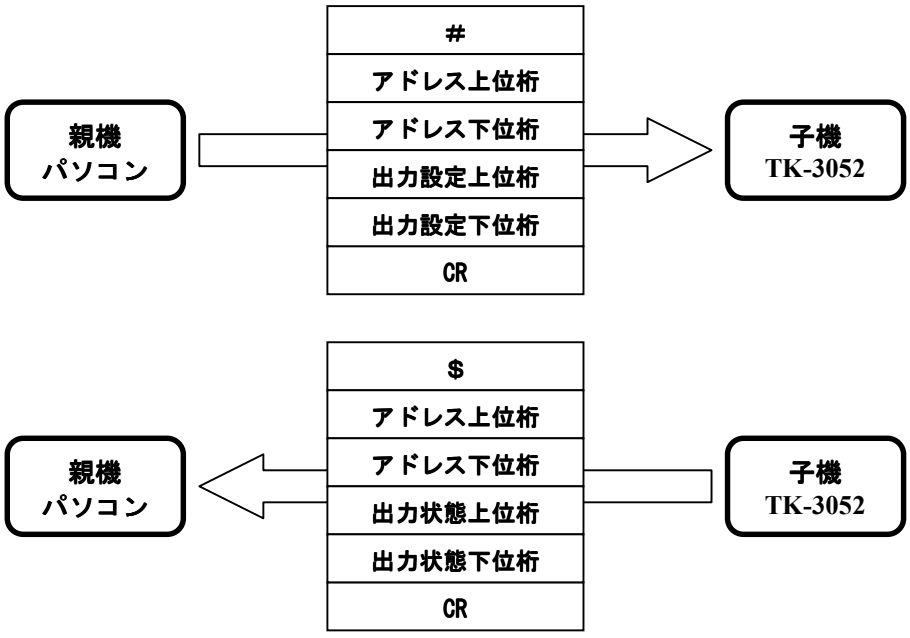
2. プロトコルの検討

親機が知りたい情報は子機の現在の出力状態であり，子機に送りたい情報は設定したい出力状態です。子機は親機からの命令に従ってデータを返信することとし，子機から親機に勝手にデータを送信することは禁止します。また，同じ回線に子機を複数台つなぐので，それぞれのデータに子機のアドレス番号を付加する必要もあります。それで，プロトコルは次のようにします。

■ **出力状態の問い合わせ（チャンネル番号と出力の数値はアスキーコード）**



■ **出力設定（チャンネル番号と出力の数値はアスキーコード）**



■ 受信プログラム

前の章と同じように受信データはキュー(リングバッファ)にセーブします。このあたりのプログラムは前の章のプログラムをそのまま使います。異なるのは，キューからコマンドを取り出したあと，正規のコマンドか判断するとともに，自分宛のコマンドか判断し，自分宛のときだけ返信します。

自分のアドレスは次のように定義します。

```
/*****************************************************************************
      グローバル変数の定義とイニシャライズ(RAM)
*****************************************************************************/
// 端末器に関係した変数 ------------------------------------------------
unsigned char   MyAddr = 0x55;        //アドレス
```

今回のプログラムではソースリスト上で自分のアドレスを定義しています。アドレスを変更したいときはこの値をかえてビルドします。

「analyze_cmd」関数でキューからコマンドを取り出し判別します。受信するコマンド長は6バイトの場合と4バイトの場合があるので，CR を受信したら 6 バイトと 4 バイトのそれぞれのケースで正しいデータか判断します。

```
/*****************************************************************************
      受信コマンドの解析
*****************************************************************************/
void analyze_rxcmd(void)
{
      unsigned char data;         //RxBufから取り出したデータ

      if (get_rxbuf(&data)==NG) return;     //受信データがない

      //正規コマンドを受信したかチェック
      switch (data){
            case CR:   //CRを受信した
                  get_command(6);
                  if (chk_cmd('#')==OK){
                        send_responce();
                  }
                  get_command(4);
                  if (chk_cmd('?')==OK){
                        send_responce();
                  }
                  break;
            default:   //制御コマンド以外のデータを受信した
                  return;
      }
}
```

正しいコマンドか判断するのは「chk_cmd」関数です。この中で，自分宛のコマンドかどうかも判断しています。なお，「＃」コマンドのときの出力ポートの変更もここで処理しています。

```
/*****************************************************************************
      コマンドのチェック
*****************************************************************************/
char chk_cmd(unsigned char cmd)
{
      unsigned char i,addr;
```

```
    switch (cmd){
        //コマンド(#) ------------------------------------------------
        case '#':
            //データチェック
            if (RxCmd[0]!='#')    {return NG;}
            for (i=1; i<5; i++){
                if        (RxCmd[i]<'0')                        {return NG;}
                else if   ((RxCmd[i]>'9') & (RxCmd[i]<'A'))     {return NG;}
                else if   (RxCmd[i]>'F')                        {return NG;}
            }

            //自分宛かアドレスのチェック
            addr = asc2hex(RxCmd[1]) * 0x10 + asc2hex(RxCmd[2]);
            if (addr != MyAddr)   {return NG;}

            //出力データをストア
            OutData = asc2hex(RxCmd[3]) * 0x10 + asc2hex(RxCmd[4]);
            PA.DR.BYTE = OutData;

            break;

        //コマンド(?) ------------------------------------------------
        case '?':
            //データチェック
            if (RxCmd[0]!='?')    {return NG;}
            for (i=1; i<3; i++){
                if        (RxCmd[i]<'0')                        {return NG;}
                else if   ((RxCmd[i]>'9') & (RxCmd[i]<'A'))     {return NG;}
                else if   (RxCmd[i]>'F')                        {return NG;}
            }

            //自分宛かアドレスのチェック
            addr = asc2hex(RxCmd[1]) * 0x10 + asc2hex(RxCmd[2]);
            if (addr != MyAddr)   {return NG;}

            break;
    }

    return OK;
}
```

■ 送信プログラム

送信プログラムの考え方は前の章と同じです。「＃」コマンドも「？」コマンドも返信データは同じなので「send_responce」関数にまとめました。出力の状態とあわせて自分のアドレスも付加しています。

```
/*****************************************************************************
     返事の送信
*****************************************************************************/
void send_responce(void)
{
     P9.DR.BIT.B5 = 1;     //MAX485 受信ディセーブル
     P9.DR.BIT.B4 = 1;     //MAX485 送信イネーブル

     txone('$');
     txone(hex2asc((unsigned char)((MyAddr & 0xf0) / 0x10)));
     txone(hex2asc((unsigned char)( MyAddr & 0x0f          )));
     txone(hex2asc((unsigned char)((PA.DR.BYTE & 0xf0) / 0x10)));
     txone(hex2asc((unsigned char)( PA.DR.BYTE & 0x0f          )));
     txone(CR);

     chk_tend();           //送信完了まで待つ

     P9.DR.BIT.B5 = 0;     //MAX485 受信イネーブル
     P9.DR.BIT.B4 = 0;     //MAX485 送信ディセーブル
}
```

◆

その他の部分のプログラムはソースリストをご覧ください。LCD に自分のアドレスと出力データを表示するようにプログラムしています。なお，付属の CD-ROM にはあらかじめダウンロードするファイルがおさめられています。「onoff_01．abs」をダウンロードして実行してください。

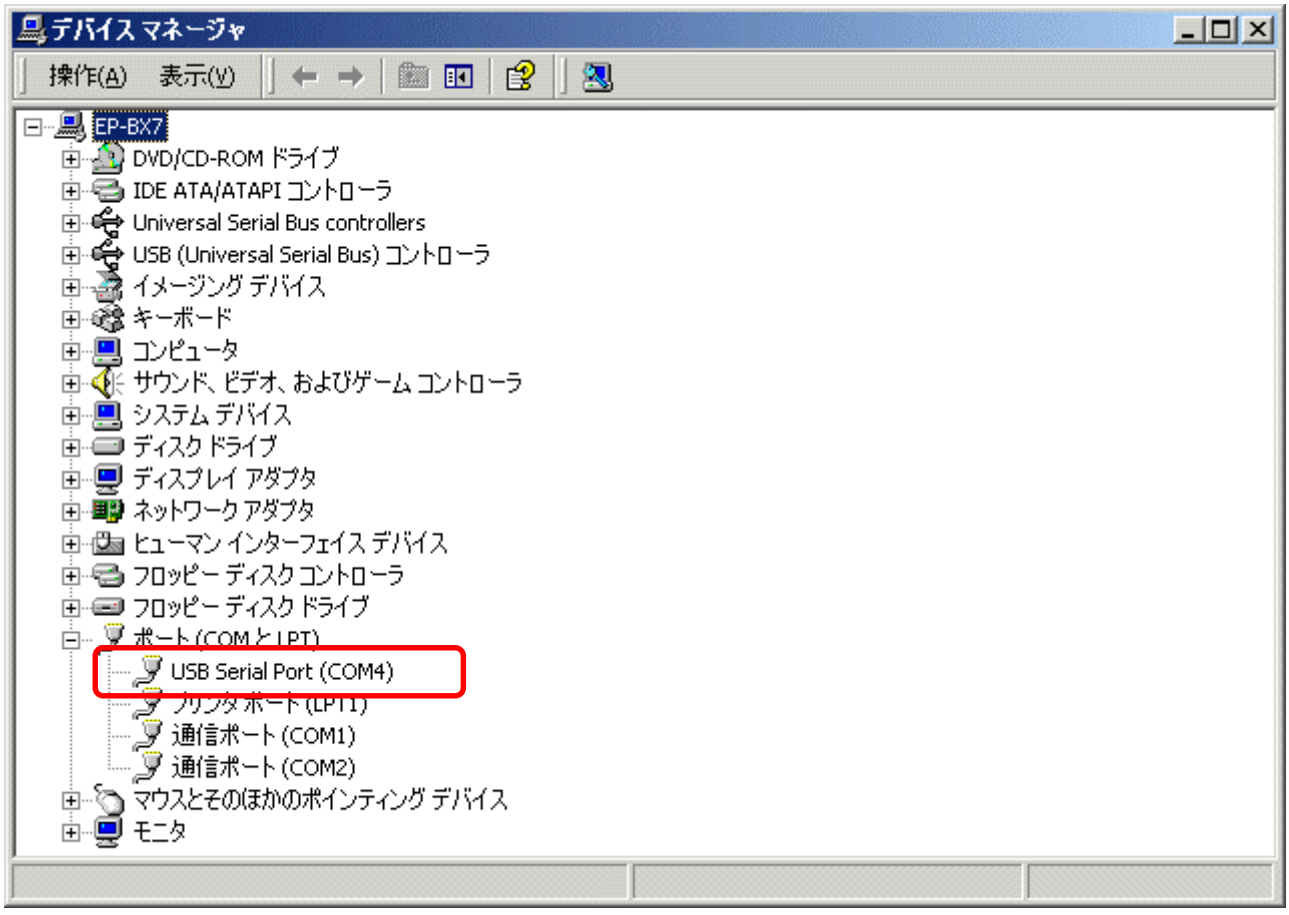
4. プログラムのデバッグ

親機（パソコン）のプログラムを作る前に，TK-3052 のプログラムをデバッグします。そのためには RS-485 でテストデータを TK-3052 に送信しないといけません。

インターフェースは最初に説明した自作の「USB-RS485 変換モジュール」を使い，テスト用のパソコンプログラムにはターミナルソフトを使います。OS が WindowsXP までならば「ハイパーターミナル」が標準で付属しています。WindowsVista 以降の場合は付属していないので自分で用意します。インターネット上にフリーのターミナルソフトがいくつも公開されていますので，気にいったものを一つ入手してください（例：TeraTerm，弊社 CD の「_始めにお読みください」フォルダにある「ターミナルソフトについて．PDF」をご覧ください）。

■ ポート番号の確認

「USB-RS485 変換モジュール」はシリアルポートとして認識されます。OS によってポート番号が割り振られますので，「システムのプロパティ」から「デバイスマネージャ」を開いてポート番号を確認します。（下記の例では COM4 になる）



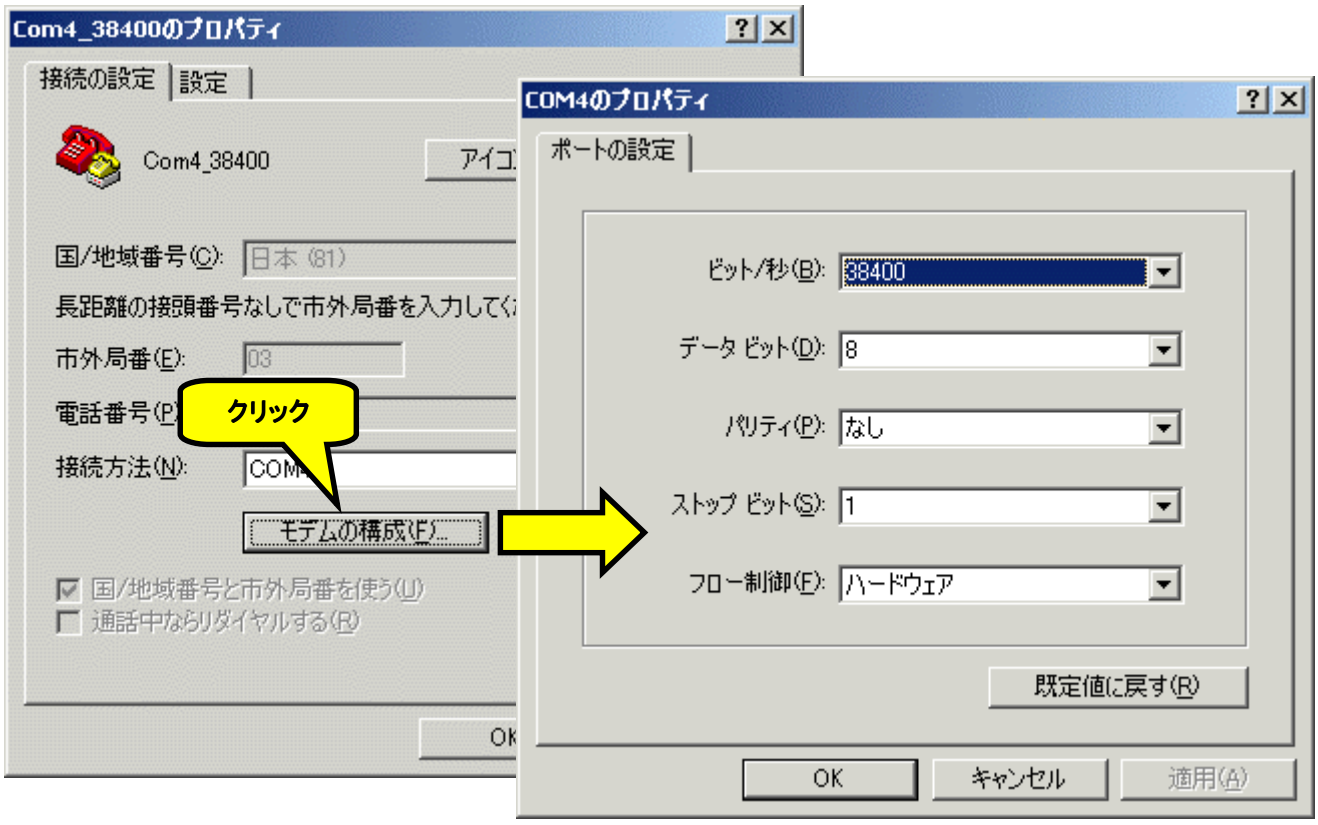
■ 通信条件

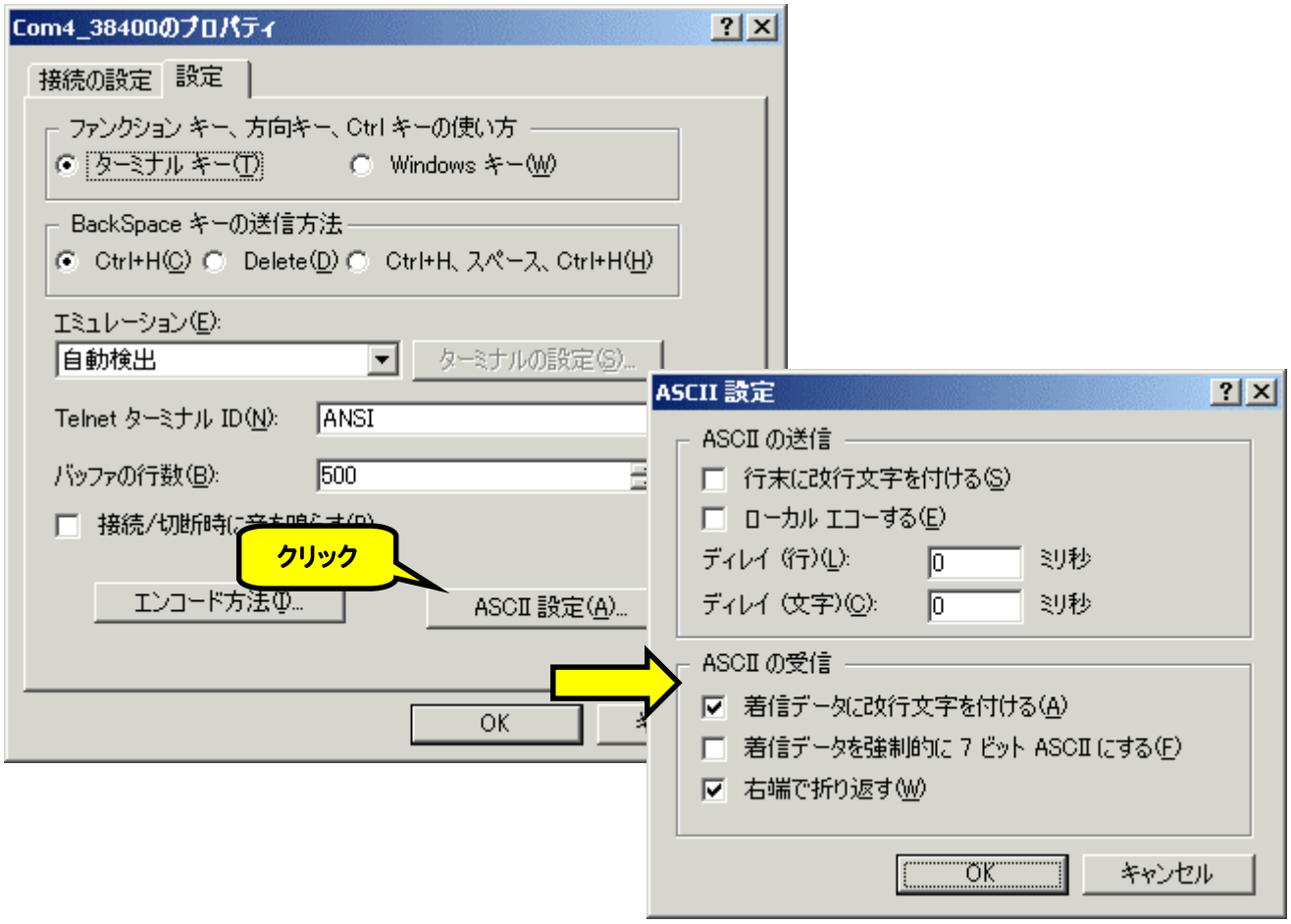
ターミナルを次のように設定します。（ターミナルソフト起動時に設定することもあります）

ビット/秒	:	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
フロー制御	:	ハードウェア

■ **ハイパーターミナルの場合**

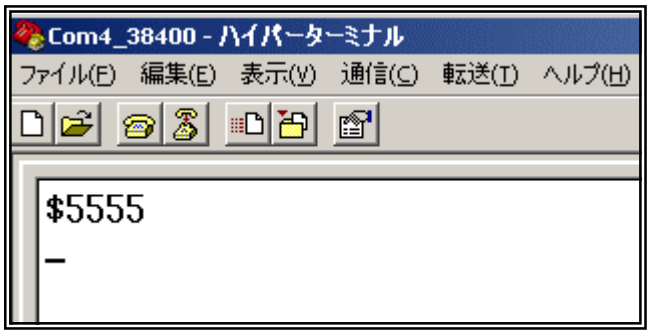
通信条件の設定を「プロパティ」で確認します。





Hterm で TK-3052 のプログラムを実行します。ハイパーターミナルから「#5555」と入力し「Enter」キーを押します。右のように返信されれば OK です。

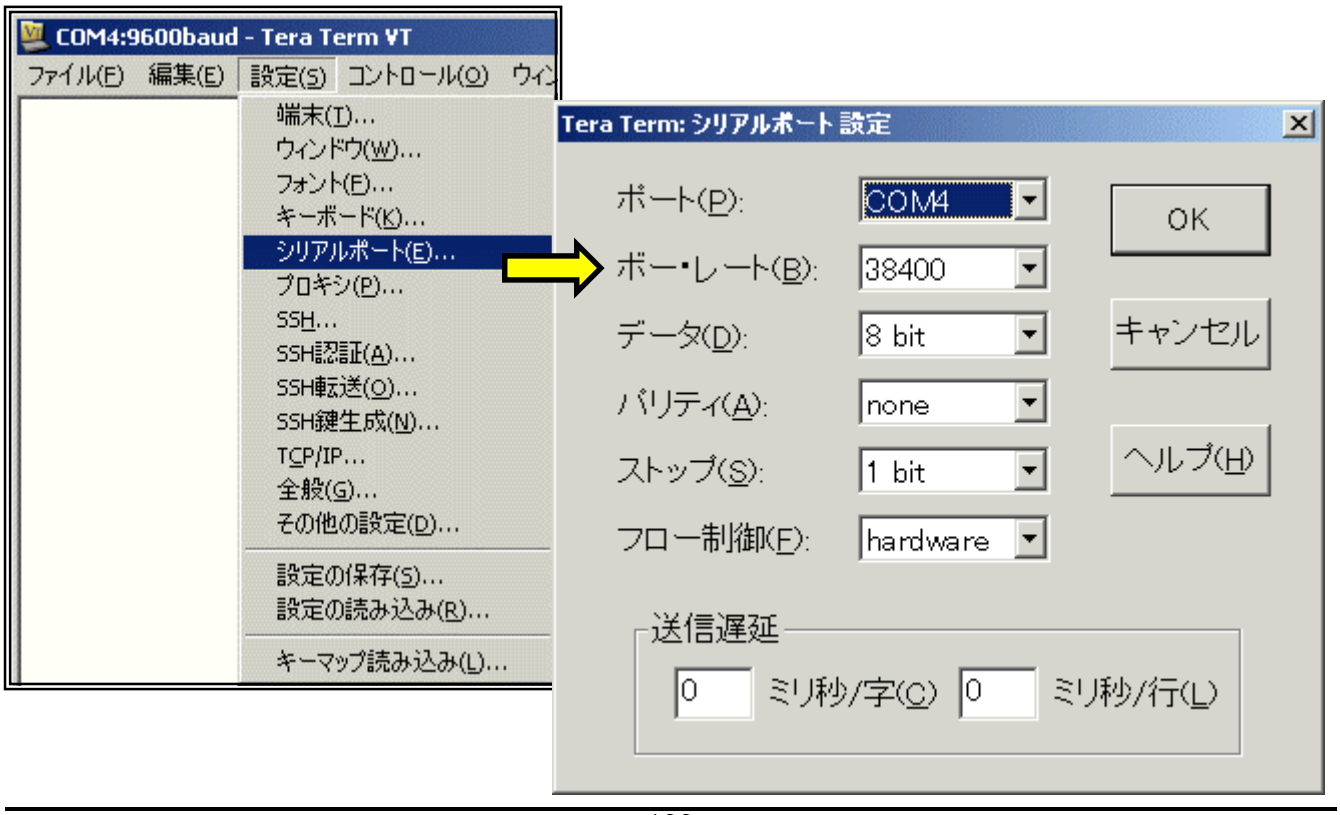
また，この状態でハイパーターミナルから「?55」と入力し「Enter」キーを押します。上と同じ返信があれば OK です。



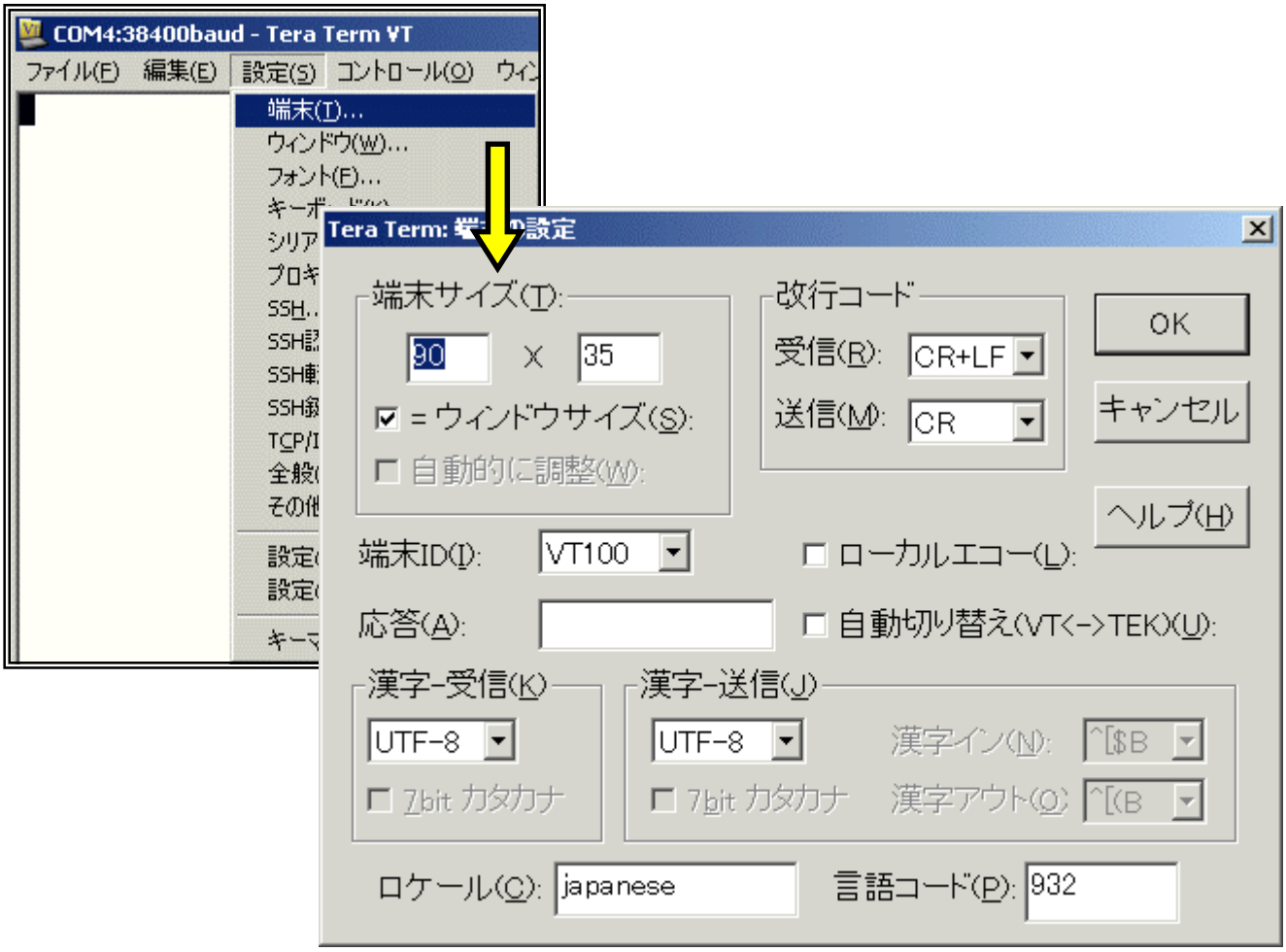
■ **TeraTerm の場合**

TeraTerm を起動すると「新しい接続」ダイアログが開きます。シリアルポートを指定し「OK」をクリックします。

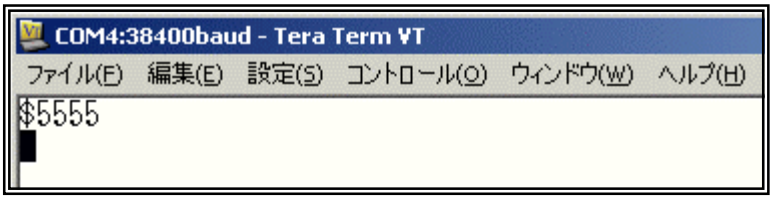
続いて，シリアルポートの通信条件を設定します。



次に端末の設定をします。



Hterm で TK-3052 のプログラムを実行します。TeraTerm から「#5555」と入力し「Enter」キーを押します。右のように返信されれば OK です。



また，この状態で TeraTerm から「?55」と入力し「Enter」キーを押します。上と同じ返信があれば OK です。

5. VB を使ったプログラム

パソコン(親機)のプログラムは VisualBasic(以降 VB と表記)で作ります。VB をお持ちの方はプロジェクト(onoff_01. vbp)を開くと詳細を見ることができます。プログラムの詳細はソースリストをご覧ください。

実行ファイル(onoff_01. exe)も作成済みです。VB をお持ちでない方も, 実行ファイルを起動すれば子機(TK-3052)を制御することができます。

なお, この VB プログラムを実行するためには標準ライブラリのほかに「MSCOMM32. OCX」が必要です。VB をインストールしてあれば問題ありませんが, そうでないときは別途入手する必要があります。詳しくは弊社 CD の「_始めにお読みください」フォルダ内の「VB プログラムについて. pdf」をご覧ください。

「onoff_01. exe」を起動すると次のような画面が表示されます。

まず, ポート番号(COM1～COM16)を指定します。USB-RS485 変換モジュールが使用するポート番号を選択してください。この部分のソースリストは次のようになっています。コンボボックス「cboCommPort」をクリックすると生じるイベントです。設定できないポートを使おうとしたり, すでに開いているポートを使おうとすると, エラーが生じます。エラーが発生したときは, エラー番号に応じてメッセージボックスを表示するようにします。

```
'*************************************************************************
'   ポート番号の切り替え
'*************************************************************************
Private Sub cboCommPort_Click()
    ans = CommPort_Open
End Sub

'*************************************************************************
'   通信ポートのオープン
'*************************************************************************
Public Function CommPort_Open() As Boolean
    On Error GoTo ErrorHandler

    'COMポートオープン
    With MSComm1
        If .PortOpen = True Then .PortOpen = False
        .CommPort = cboCommPort.ListIndex + 1
        .Settings = "38400" + "," + "N" + "," + "8" + "," + "1"
        .RTSEnable = True
        .InBufferCount = 0
        .OutBufferCount = 0
        .PortOpen = True
    End With

    CommPort_Open = True
    Exit Function

ErrorHandler:
    If Err.Number = 8002 Then       '設定できないポートを使おうとした
        prompt = "使用できないポートを開こうとしました。" & Chr(13)
        prompt = prompt & "ポート番号を確認してください。"
        buttons = vbOKOnly + vbExclamation
```

```
        Title = “ポート番号の確認”
        ans = MsgBox(prompt, buttons, Title)
        CommPort_Open = False
        Exit Function
    ElseIf Err.Number = 8005 Then   ‘開いているポートを使おうとした
        prompt = “使用中のポートを開こうとしました。” & Chr(13)
        prompt = prompt & “ポート番号を確認してください。”
        buttons = vbOKOnly + vbExclamation
        Title = “ポート番号の確認”
        ans = MsgBox(prompt, buttons, Title)
        CommPort_Open = False
        Exit Function
    End If
    CommPort_Open = False
End Function
```

次に子機のアドレス(00～FF)を指定します。TK-3052 のソースリストで設定したアドレスを指定します(出荷時のリストは 55 になっています)。ポート番号を指定しないとアドレスを指定することはできません(警告メッセージを表示する)。アドレスを指定すると子機に対して「出力状態の問い合わせ」コマンドを送信します。同時にタイマーを起動します。

```
‘*************************************************************************
‘   アドレス切り替え
‘*************************************************************************
Private Sub cboAddress_Click()
    If MSComm1.PortOpen = False Then
        prompt = “ポート番号が指定されていません。” & Chr(13)
        prompt = prompt & “ポート番号を指定してください。”
        buttons = vbOKOnly + vbExclamation
        Title = “ポート番号の確認”
        ans = MsgBox(prompt, buttons, Title)
        cboAddress.Text = “”
        Exit Sub
    End If

    strSendData = “?” + cboAddress.Text + Chr$(&HD)
    Command_Send

    Timer1.Interval = 1000
    Timer1.Enabled = True
End Sub

’*************************************************************************
’   コマンド送信
’*************************************************************************
Public Sub Command_Send()
    If MSComm1.PortOpen = False Then
        prompt = "ポート番号が指定されていません。" & Chr(13)
        prompt = prompt & "ポート番号を指定してください。"
        buttons = vbOKOnly + vbExclamation
        Title = "ポート番号の確認"
        ans = MsgBox(prompt, buttons, Title)
        Exit Sub
    End If

    ’送信
    blnHeadderRcv = False
    MSComm1.Output = strSendData
End Sub
```

子機からの返信を受信するとタイマーを停止しますが，もしタイマーイベントが発生したときは，そのアドレスの子機がつながっていないか，ケーブルが断線している可能性があります。それで，エラーメッセージを表示します。

```
'*************************************************************************
'   タイムオーバー
'*************************************************************************
Private Sub Timer1_Timer()
    Timer1.Enabled = False

    prompt = "端末から反応がありません。" & Chr(13)
    prompt = prompt & "端末，ならびにケーブルを確認してください。"
    buttons = vbOKOnly + vbExclamation
    Title = "端末からの反応なし"
    ans = MsgBox(prompt, buttons, Title)
End Sub
```

子機との接続が完了したら，出力の指定を行ないます。「B0」～「B7」のボタンや「全部オン」，「全部オフ」，「反転」のボタンをクリックします。すると，「出力設定」コマンドを送信します。いずれも，子機のアドレスが指定される前にクリックされたときは警告メッセージを表示します。

```
'*************************************************************************
'   全部オフ
'*************************************************************************
Private Sub cmdAllOff_Click()
    If cboAddress.Text = "" Then
        prompt = "アドレスが指定されていません。" & Chr(13)
        prompt = prompt & "アドレスを指定してください。"
        buttons = vbOKOnly + vbExclamation
        Title = "アドレスの確認"
        ans = MsgBox(prompt, buttons, Title)
        Exit Sub
    End If

    For I = 0 To 7
        cmdBit(i).Tag = 0
        cmdBit(i).BackColor = &HFFFFFF
        OnOff(Val("&H" + cboAddress.Text), I) = 0
    Next I

    a = 0
    For I = 0 To 7
        a = OnOff(Val("&H" + cboAddress.Text), I) * (2 ^ I) + a
    Next I
    strSendData = "#" + cboAddress.Text + Right$("0" + Hex$(a), 2) + Chr$(&HD)
    Command_Send

End Sub

'*************************************************************************
'   全部オン
'*************************************************************************
Private Sub cmdAllOn_Click()
    If cboAddress.Text = "" Then
        prompt = "アドレスが指定されていません。" & Chr(13)
        prompt = prompt & "アドレスを指定してください。"
        buttons = vbOKOnly + vbExclamation
        Title = "アドレスの確認"
        ans = MsgBox(prompt, buttons, Title)
        Exit Sub
```

```
    End If

    For I = 0 To 7
        cmdBit(i).Tag = 1
        cmdBit(i).BackColor = &HFF
        OnOff(Val("&H" + cboAddress.Text), I) = 1
    Next I

    a = 0
    For I = 0 To 7
        a = OnOff(Val("&H" + cboAddress.Text), I) * (2 ^ I) + a
    Next I
    strSendData = "#" + cboAddress.Text + Right$("0" + Hex$(a), 2) + Chr$(&HD)
    Command_Send
End Sub

'*************************************************************************
'   On/Off
'*************************************************************************
Private Sub cmdBit_Click(Index As Integer)
    If cboAddress.Text = "" Then
        prompt = "アドレスが指定されていません。" & Chr(13)
        prompt = prompt & "アドレスを指定してください。"
        buttons = vbOKOnly + vbExclamation
        Title = "アドレスの確認"
        ans = MsgBox(prompt, buttons, Title)
        Exit Sub
    End If

    If cmdBit(Index).Tag = 1 Then
        cmdBit(Index).Tag = 0
        cmdBit(Index).BackColor = &HFFFFFF
    Else
        cmdBit(Index).Tag = 1
        cmdBit(Index).BackColor = &HFF
    End If
    OnOff(Val("&H" + cboAddress.Text), Index) = cmdBit(Index).Tag

    a = 0
    For I = 0 To 7
        a = OnOff(Val("&H" + cboAddress.Text), I) * (2 ^ I) + a
    Next I
    strSendData = "#" + cboAddress.Text + Right$("0" + Hex$(a), 2) + Chr$(&HD)
    Command_Send
End Sub

'*************************************************************************
'   反転
'*************************************************************************
Private Sub cmdToggle_Click()
    If cboAddress.Text = "" Then
        prompt = "アドレスが指定されていません。" & Chr(13)
        prompt = prompt & "アドレスを指定してください。"
        buttons = vbOKOnly + vbExclamation
        Title = "アドレスの確認"
        ans = MsgBox(prompt, buttons, Title)
        Exit Sub
    End If

    For I = 0 To 7
        If OnOff(Val("&H" + cboAddress.Text), I) = 1 Then
            cmdBit(i).Tag = 0
```

```
            cmdBit(i).BackColor = &HFFFFFF
            OnOff(Val("&H" + cboAddress.Text), I) = 0
        Else
            cmdBit(i).Tag = 1
            cmdBit(i).BackColor = &HFF
            OnOff(Val("&H" + cboAddress.Text), I) = 1
        End If
    Next I

    a = 0
    For I = 0 To 7
        a = OnOff(Val("&H" + cboAddress.Text), I) * (2 ^ I) + a
    Next I
    strSendData = "#" + cboAddress.Text + Right$("0" + Hex$(a), 2) + Chr$(&HD)
    Command_Send
End Sub
```

あとは受信ルーチンです。何かデータを受信するとMSComm1イベントが発生します。「出力状態の問い合わせ」コマンドに対する返信の場合は，現在の出力の状態を表示し，メモリにもストアします。「出力設定」コマンドに対する返信の場合は，送信した設定値と比較し異なる場合は警告メッセージを表示します。

```
'*************************************************************************
'   MSComm1イベント
'*************************************************************************
Private Sub MSComm1_OnComm()
    Select Case MSComm1.CommEvent
        Case comEvReceive
            Call Data_Receive
    End Select
End Sub

'*************************************************************************
'   データ受信
'*************************************************************************
Public Sub Data_Receive()
    Dim rcv As String

    rcv = MSComm1.Input     '受信データの取得

    If blnHeadderRcv = False Then   'ヘッダー未受信
        If rcv = "$" Then
            blnHeadderRcv = True    'ヘッダー受信
            blnDelimiterRcv = False 'デリミタ未受信
            strRcvData = rcv
        End If
    Else                            'ヘッダー受信済み
        If Right(rcv, 1) = Chr(13) Then     'デリミタ受信？
            strRcvData = strRcvData + rcv
            blnHeadderRcv = False
            intAddressRcv = Val("&h" + Mid$(strRcvData, 2, 2))
            intOnOffRcv = Val("&h" + Mid$(strRcvData, 4, 2))
            Message.Text = "Address=" + "&h" + Mid$(strRcvData, 2, 2)
            Message.Text = Message.Text + "  Data=" + "&h" + Mid$(strRcvData, 4, 2)
            If Left$(strSendData, 1) = "?" Then
                For i = 0 To 7
                    If (intOnOffRcv And (2 ^ i)) = 0 Then
                        OnOff(intAddressRcv, i) = 0
                        cmdBit(i).Tag = 0
                        cmdBit(i).BackColor = &HFFFFFF
                    Else
```

```
                    OnOff(intAddressRcv, i) = 1
                    cmdBit(i).Tag = 1
                    cmdBit(i).BackColor = &HFF
                End If
            Next i
        ElseIf Left$(strSendData, 1) = "#" Then
            a = 0
            For i = 0 To 7
                a = OnOff(intAddressRcv, i) * (2 ^ i) + a
            Next i
            If a <> intOnOffRcv Then
                prompt = "端末の出力状態が指定と異なります。" & Chr(13)
                prompt = prompt & "端末を確認してください。"
                buttons = vbOKOnly + vbExclamation
                Title = "端末の確認"
                ans = MsgBox(prompt, buttons, Title)
            End If
        End If
        Timer1.Enabled = False
    Else
        strRcvData = strRcvData + rcv   '受信文字列追加
    End If
End If
End Sub
```

6. プログラムを改造する

今回サンプルとして用意した VB プログラムは必要最小限の機能に絞っています。これをベースに機能を追加し，もっと使いやすくすることができます。

■ 複数アドレスのオン/オフ一覧表示

一つのアドレスのみ表示していますが，RS-485 ネットワークに複数の子機を接続したときは一度に全部見ることができたほうが使いやすいでしょう。

■ スケジュール管理

ボタンをクリックすると出力がオン/オフしますが，パソコンの時計に同期して自動的にオン/オフする機能を追加すれば，スケジュールに基づく自動運転も可能になります。

そのほかにもいろいろ考えられると思います。ぜひ挑戦してみてください。

第14章 RS-485 を使ったネットワークの実験(3)

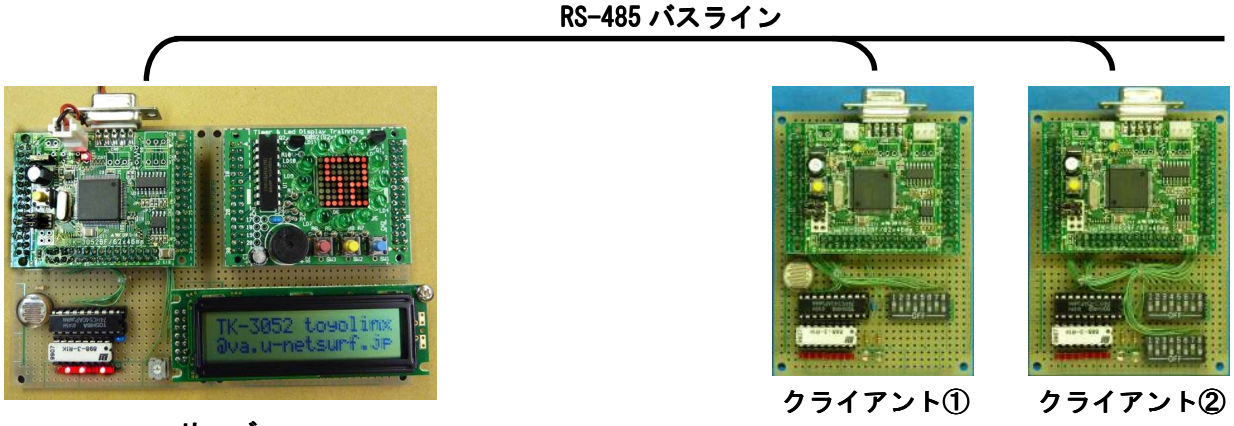
前の章では複数の端末を RS-485 のネットワークにつなぐ実験をしましたが，親機（パソコン）から子機（TK-3052）に現在の状態を要求したり，出力を送信したりして，親機がネットワークに流れるデータ全体を管理する方法を取りました。しかし，別の方法として，子機がデータを親機にアップデートしたり，子機が必要なときに親機に要求してデータをダウンロードしたりする，ということも考えられます。この場合，親機はデータを蓄えるサーバ，子機はデータを要求するクライアントになります。この章では，この「クライアント・サーバ方式」で複数の端末を RS-485 のネットワークにつなぐ実験をします。

1. 実験システムの構成

マイコン事始め（TK-3052）をサーバにし，加えて新たに TK-3052 を使用したクライアントを作成します。クライアントにはそれぞれ独立したアドレス（00～FF）をディップスイッチで割り付けます。

クライアントは 2 種類作成します。一つは CDS を搭載し，明るさを AD コンバータで取得して，AD 値をサーバにアップロードします。モニタ用に AD 値を LED にローカルで表示します。

もう一つのクライアントは，表示したいクライアントのアドレスをディップスイッチで指定し，得られた AD 値を LED に表示します。

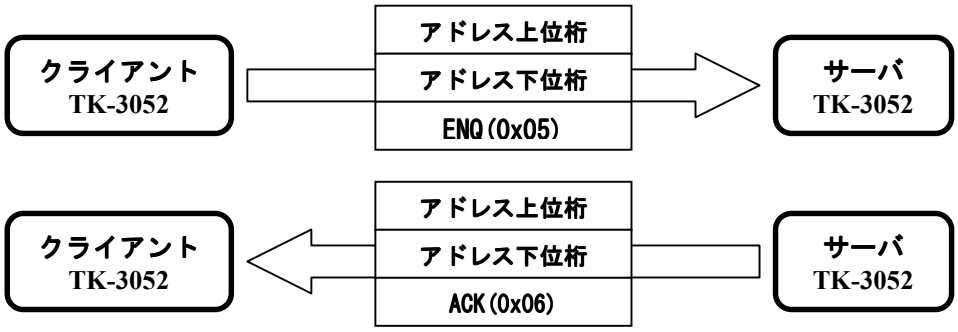


クライアント①と②は，TK-3052 を使用して新たに作成します。巻末の付録に回路図と部品表，実装図，写真が掲載されています。これらを参考に組み立ててください。

2. プロトコルの検討

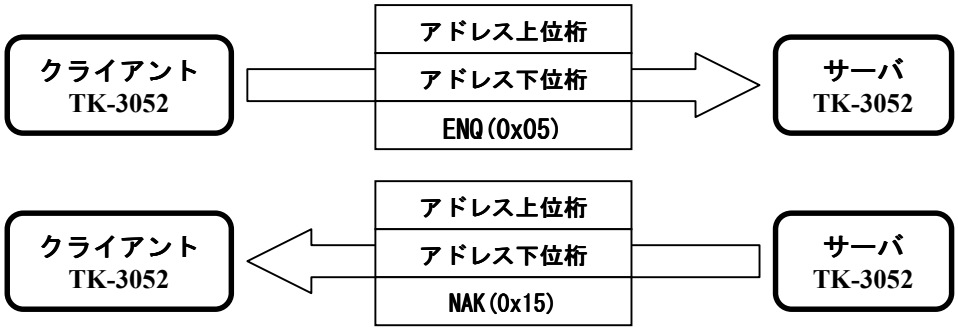
同じ回線に複数台のクライアントをつなぎ，かつ，クライアントが必要に応じてサーバと通信しようとするので，データが衝突しないように考慮する必要があります。それで，データを通信する前に回線を使用してよいか尋ね，許可されてからデータ通信を開始します。データ通信が終了したら回線使用を解除します。それで，プロトコルは次のようにします。

■ **通信要求，サーバは通信を許可する（アドレス値はアスキーコード）**



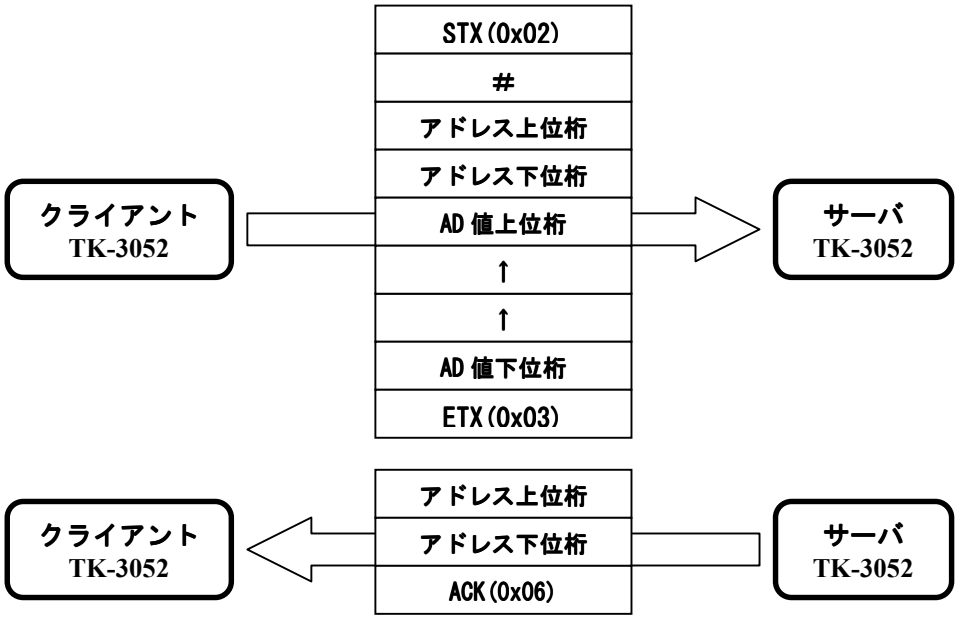
クライアントはサーバに回線の使用を要求します。許可されたならデータ通信を続けます。これ以降，他のクライアントはサーバとの通信が許可されません。

■ **通信要求，サーバは通信を許可しない（アドレス値はアスキーコード）**

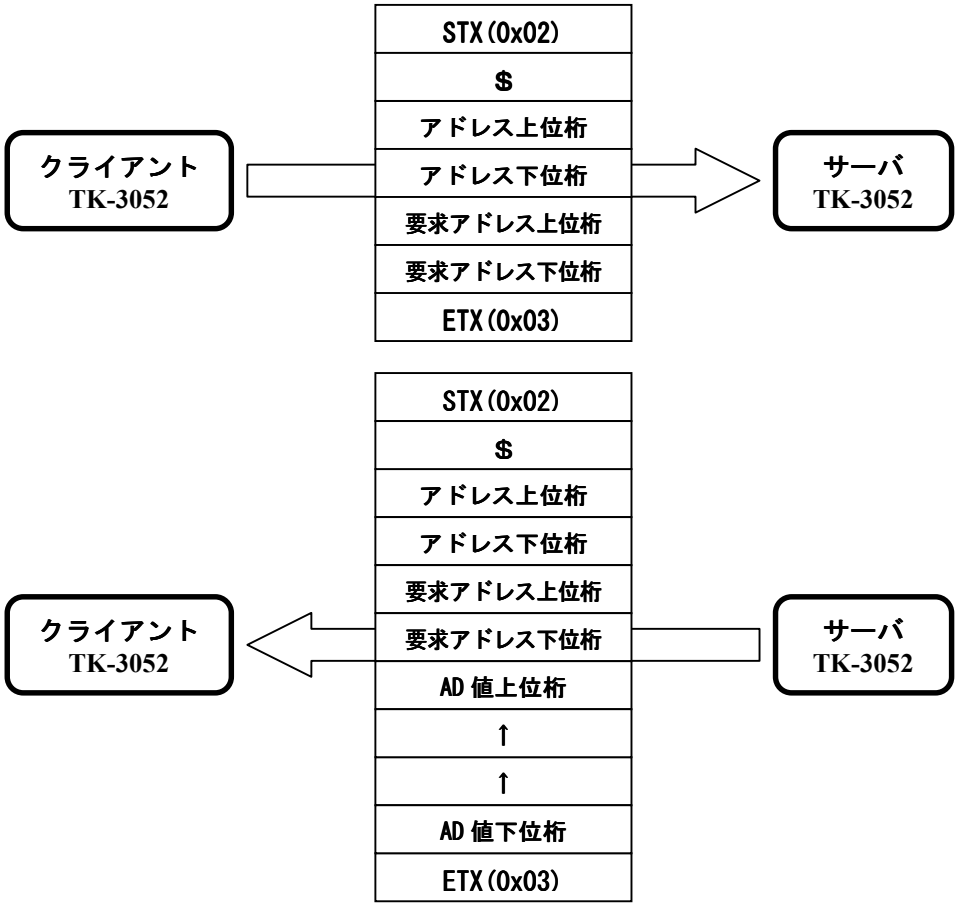


クライアントはサーバに回線の使用を要求します。許可されないときは，しばらくしてから再び通信要求をサーバに送信します。

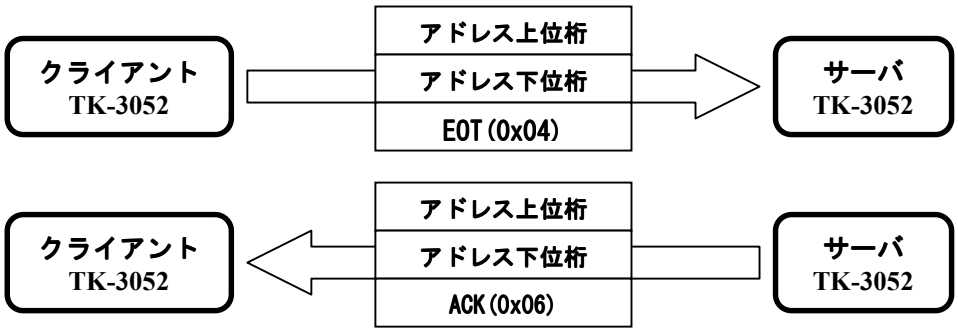
■ **AD 値のアップロード(アドレス値, AD 値はアスキーコード)**



■ **AD 値のダウンロード(アドレス値, AD 値はアスキーコード)**



■ **通信の開放(アドレス値はアスキーコード)**



これ以降, 他のクライアントの通信も許可されます。

3. サーバのプログラム

これまでと同じように受信データはキュー(リングバッファ)にセーブします。これまで使用してきたプログラムを流用します。

「analyze_cmd」関数でキューからコマンドを取り出し判別します。プロトコルからわかるように, ENQとEOTを受信した場合, および, ETX を受信したときは 9 バイトの場合と 7 バイトの場合のそれぞれのケースで正しいデータか判断します。

```
/*****************************************************************************
     受信コマンドの解析
*****************************************************************************/
void analyze_rxcmd(void)
{
     unsigned char data;        //RxBufから取り出したデータ

     if (get_rxbuf(&data)==NG) return;     //受信データがない

     //正規コマンドを受信したかチェック
     switch (data){
          case ENQ:  //ENQを受信した
               get_command(ENQ_LEN);
               chk_enq_cmd();
               break;
          case ETX:  //ETXを受信した
               get_command(ETX1_LEN);
               if (chk_etx1_cmd()==OK) {break;}
               get_command(ETX2_LEN);
               chk_etx2_cmd();
               break;
          case EOT:  //EOTを受信した
               get_command(EOT_LEN);
               chk_eot_cmd();
               break;
          default:   //制御コマンド以外のデータを受信した
               return;
     }
}
```

ENQ, ETX, EOT, それぞれのコマンドが正しいかどうか判断し, 正しい場合はそれぞれのコマンドの処理を実行します。

```
/*****************************************************************************
     ENQコマンドチェック
*****************************************************************************/
char chk_enq_cmd(void)
{
     unsigned char i;

     //データチェック
     for (i=0; i<2; i++){
          if        (RxCmd[i]<'0')                          {return NG;}
          else if   ((RxCmd[i]>'9') & (RxCmd[i]<'A'))       {return NG;}
          else if   (RxCmd[i]>'F')                          {return NG;}
     }
     if (RxCmd[2]!=ENQ) {return NG;}

     //返信
     if (ConnectFlag==0){ //回線未使用
          ConnectFlag = 1;
          ConnectAddr = asc2hex(RxCmd[0]) * 0x10 + asc2hex(RxCmd[1]);
          send_ack(ConnectAddr);
          TimT0.Status = 1;     //タイマスタート
     }
     else{                      //回線使用中
          if (ConnectAddr == (asc2hex(RxCmd[0]) * 0x10 + asc2hex(RxCmd[1]))){
               send_ack(ConnectAddr);
          }
          else{
               send_nak(asc2hex(RxCmd[0]) * 0x10 + asc2hex(RxCmd[1]));
          }
     }

     return OK;
}

/*****************************************************************************
     ETX(AD値のアップロード)コマンドチェック
*****************************************************************************/
char chk_etx1_cmd(void)
{
     unsigned char i;

     //データチェック
     if (RxCmd[0]!=STX) {return NG;}
     if (RxCmd[1]!='#') {return NG;}
     for (i=2; i<8; i++){
          if        (RxCmd[i]<'0')                          {return NG;}
          else if   ((RxCmd[i]>'9') & (RxCmd[i]<'A'))       {return NG;}
          else if   (RxCmd[i]>'F')                          {return NG;}
     }
     if (RxCmd[8]!=ETX) {return NG;}

     //アドレスチェック
     if (ConnectAddr!=(asc2hex(RxCmd[2]) * 0x10 + asc2hex(RxCmd[3]))){
          return NG;
     }

     //AD値のストア
     ServerData[ConnectAddr]    =      (asc2hex(RxCmd[4]) * 0x1000
```

```
                                    + asc2hex(RxCmd[5]) * 0x0100
                                    + asc2hex(RxCmd[6]) * 0x0010
                                    + asc2hex(RxCmd[7]));
    //返信
    send_ack(ConnectAddr);

    return OK;
}

/*****************************************************************************
    ETX(AD値のダウンロード)コマンドチェック
*****************************************************************************/
char chk_etx2_cmd(void)
{
    unsigned char i;
    unsigned int addr;

    //データチェック
    if (RxCmd[0]!=STX) {return NG;}
    if (RxCmd[1]!='$') {return NG;}
    for (i=2; i<6; i++){
        if      (RxCmd[i]<'0')                      {return NG;}
        else if ((RxCmd[i]>'9') & (RxCmd[i]<'A'))   {return NG;}
        else if (RxCmd[i]>'F')                      {return NG;}
    }
    if (RxCmd[6]!=ETX) {return NG;}

    //アドレスチェック
    if (ConnectAddr!=(asc2hex(RxCmd[2]) * 0x10 + asc2hex(RxCmd[3]))){
        return NG;
    }

    //返信
    addr = asc2hex(RxCmd[4]) * 0x10 + asc2hex(RxCmd[5]);
    P9.DR.BIT.B5 = 1;                   //MAX485 受信ディセーブル
    P9.DR.BIT.B4 = 1;                   //MAX485 送信イネーブル
    txone(STX);
    txone('$');
    txone(RxCmd[2]);
    txone(RxCmd[3]);
    txone(RxCmd[4]);
    txone(RxCmd[5]);
    txone(hex2asc((ServerData[addr] & 0xf000) / 0x1000));
    txone(hex2asc((ServerData[addr] & 0x0f00) / 0x0100));
    txone(hex2asc((ServerData[addr] & 0x00f0) / 0x0010));
    txone(hex2asc( ServerData[addr] & 0x000f));
    txone(ETX);
    chk_tend();                         //送信完了まで待つ
    P9.DR.BIT.B5 = 0;                   //MAX485 受信イネーブル
    P9.DR.BIT.B4 = 0;                   //MAX485 送信ディセーブル

    return OK;
}

/*****************************************************************************
    EOTコマンドチェック
*****************************************************************************/
char chk_eot_cmd(void)
{
    unsigned char i;

    //データチェック
```

```
    for (i=0; i<2; i++){
        if        (RxCmd[i]<'0')                        {return NG;}
        else if   ((RxCmd[i]>'9') & (RxCmd[i]<'A'))     {return NG;}
        else if   (RxCmd[i]>'F')                        {return NG;}
    }
    if (RxCmd[2]!=EOT) {return NG;}

    //返信
    if ((ConnectFlag==1) && (ConnectAddr==(asc2hex(RxCmd[0]) * 0x10 + asc2hex(RxCmd[1])))){
        ConnectFlag = 0;
        ConnectAddr = asc2hex(RxCmd[0]) * 0x10 + asc2hex(RxCmd[1]);
        send_ack(ConnectAddr);
        TimT0.Status= 0;     //タイマストップ
    }

    return OK;
}
```

その他の部分のプログラムはソースリストをご覧ください。なお，付属の CD-ROM にはあらかじめダウンロードするファイルがおさめられています。「server_01. abs」をダウンロードして実行してください。

4. クライアント①のプログラム

クライアントのプログラムは，一定時間ごとに通信要求のコマンド(ENQ)を送信し，あとは返信内容に応じて処理を進めていきます。処理を進める中で，今どの処理を行なっているかを示す値が「StageNo」にストアされていて，メインプログラムではこの「StageNo」で処理を分岐していきます。メインルーチンのソースリストは次のとおりです。

```
/*************************************************************************
    メインプログラム
*************************************************************************/
void main(void)
{
// イニシャライズ ------------------------------------------------------
    ini_io();
    ini_sci_0();
    ini_itu();
    ini_ad();

    init_rxbuf();
    init_soft_timer();

// メインループ --------------------------------------------------------
    while(1){
        //チャンネル番号の取得
        MyAddr = ~P4.DR.BYTE;

        //AD変換
        if(AD.ADCSR.BIT.ADF==1){   //AD変換終了?
            AD.ADCSR.BIT.ADF = 0;//AD変換終了フラグクリア
            AdSum = AdSum + (unsigned long)AD.ADDRA;   //AD値を取得,バッファに加算
            AdCount++;
            if(AdCount>=AD_CONST){     //加算終了?
                AdData = (unsigned int)(AdSum / AD_CONST); //平均化
                AdSum = 0;             //加算バッファクリア
                AdCount = 0;           //加算カウンタクリア
            }
            AD.ADCSR.BIT.ADST = 1;     //AD変換スタート
        }
```

```
        //ステージナンバーごとに分岐
        switch(StageNo){
            case 0:    //ENQコマンドの送信
                if ((TimT0.Status==0)||(TimT0.Status==3)){
                    TimT0.Status = 1;     //タイマステータスクリア
                    send_enq(MyAddr);
                    StageNo = 1;
                }
                break;
            case 1:    //ENQコマンドの返信待ち
                break;
            case 2:    //AD値の送信
                send_ad(MyAddr);
                StageNo = 3;
                break;
            case 3:    //AD値送信の返信待ち
                break;
            case 4:    //EOTコマンドの送信
                send_eot(MyAddr);
                StageNo = 5;
                break;
            case 5:    //EOTコマンドの返信待ち
                break;
            default:
                break;
        }

        //受信コマンドの解析
        analyze_rxcmd();

        //応答タイムアウト
        if (TimT1.Status==3){//コマンドを送信してからタイムアウトした
            TimT1.Status = 0;     //タイマステータスクリア
            StageNo = 0;          //ステージナンバークリア
            TimT0.Status = 1;     //タイマスタート
        }

        //表示
        PA.DR.BYTE = (unsigned char)(AdData / 0x100);
    }
}
```

その他の部分のプログラムはソースリストをご覧ください。なお，付属の CD-ROM にはあらかじめダウンロードするファイルがおさめられています。「client1_01. abs」をダウンロードして実行してください。

5. クライアント②のプログラム

基本的にはクライアント①と同じ考え方です。メインルーチンのソースリストは次のとおりです。

```
/************************************************************************
    メインプログラム
************************************************************************/
void main(void)
{
// イニシャライズ ------------------------------------------------
    ini_io();
    ini_sci_0();
    ini_itu();
```

```
     init_rxbuf();
     init_soft_timer();

// メインループ ----------------------------------------------------
     while(1){
          //チャンネル番号の取得
          MyAddr     =     ~P4.DR.BYTE;
          ReqAddr    =     ~P2.DR.BYTE;

          //ステージナンバーごとに分岐
          switch(StageNo){
               case 0:    //ENQコマンドの送信
                    if ((TimT0.Status==0)||(TimT0.Status==3)){
                         TimT0.Status = 0;     //タイマステータスクリア
                         send_enq(MyAddr);
                         StageNo = 1;
                    }
                    break;
               case 1:    //ENQコマンドの返信待ち
                    break;
               case 2:    //AD値の要求
                    send_ad_req(MyAddr,ReqAddr);
                    StageNo = 3;
                    break;
               case 3:    //AD値要求の返信待ち
                    break;
               case 4:    //EOTコマンドの送信
                    send_eot(MyAddr);
                    StageNo = 5;
                    break;
               case 5:    //EOTコマンドの返信待ち
                    break;
               default:
                    break;
          }

          //受信コマンドの解析
          analyze_rxcmd();

          //応答タイムアウト
          if (TimT1.Status==3){//コマンドを送信してからタイムアウトした
               TimT1.Status = 0;     //タイマステータスクリア
               StageNo = 0;          //ステージナンバークリア
               TimT0.Status = 1;     //タイマスタート
          }

          //表示
          PA.DR.BYTE = (unsigned char)(AdData / 0x100);
     }
}
```

その他の部分のプログラムはソースリストをご覧ください。なお，付属の CD-ROM にはあらかじめダウンロードするファイルがおさめられています。「client2_01. abs」をダウンロードして実行してください。

第15章 テトリスの作成

一つのアプリケーションを設計しまとめあげる方法は参考書を読めば身につくというものではありません。プログラムの文法は知っていても，大きなプログラムを作ろうとするとどこから手をつけたらよいかわからない，ということはよくあります。この章ではタイマ&LED ディスプレイでテトリスを作ることで，プログラムの設計方法や考え方を学習します。

1. テトリスの仕様

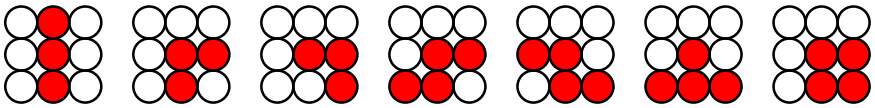
テトリスは，いわゆる「落ちゲー」と呼ばれる種類のミニゲームです。ルールは単純ですが，なぜかはまってしまうようです。有名なゲームなのでご存知の方が多いと思います。もしどんなゲームか知らないので調べてみたいと思う方は，インターネットで「テトリス」を検索してみてください。パソコン上で動くプログラムがいくつも見つかるはずです。中にはインストールしなくてもブラウザ上で動くテトリスもあります。実際に動かしてみるとゲームのイメージがつかめるでしょう。

さて，プログラムを作るにあたり最初に行なう作業は，どんなプログラムにするか仕様を決める，ということです。これがないと先に進むことができません。タイマ&LED ディスプレイ版テトリスの仕様を考えてみましょう。

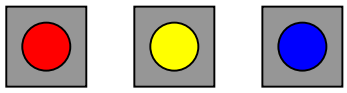
ゲーム画面の広さは LED ディスプレイで決まってしまいます。というわけで 8×8 ドットにします。上からブロックが出現し，一定時間毎に下に落ちます。下に落ちることができなくなったらそこで固定され次のブロックが再び上から現れます。

ブロックが固定されたときに横一行ブロックがそろっているとその行が削除され，消えた行の上にブロックがあるときは消えた行数分だけ下がります。ブロックを消すことができずに一番上まで到達したらゲームオーバーです。

落ちてくるブロックの形は次の 7 種類で，どのブロックにするかは乱数を使ってランダムに決定します。



ゲームユーザが操作できるのは今落ちているブロックです。ブロックを左右に移動させたり，ブロックを回転して向きを変えたりできます。タイマ&LED ディスプレイのスイッチ SW1～3 で操作します。



SW3 回転　SW2 左　SW1 右

ゲームなので点数も付けましょう。1 行削除すると 1 点とします。ただし，1 行づつ削除するよりも 2 行や 3 行まとめて削除するほうが難易度が高いので，2 行まとめて削除したときは 4 点，3 行まとめて削除したときは 9 点にします。（点数はまとめて削除した行数×行数で計算します）

ゲームが進むにつれて難易度を上げます。0～9 点までは 1 秒毎にブロックを下に落とします。そして，10～19 点のときは 0.95 秒毎，20～29 点のときは 0.9 秒毎，というように 10 点毎に 0.05 秒づつ短くしていきます。

周囲の LED は現在の点数を表します。1～9 点で 1 個，10～19 点で 2 個･･･と点灯する LED が増えていきます。点灯していない LED はブロックが落ちていくタイミングにあわせて点滅します。

ゲームオーバーで点数を表示しメロディを演奏します。なお，9 点以下と 10 点以上のときで演奏するメロディを変えます。

基本的な仕様は以上ですが，もう少しだけ。プログラムスタートでまずはゲームスタート待ちにし，周囲の LED を点滅させます。いずれかのスイッチを押したらゲームスタートです。何も押さずに 3 秒が経過するとデモ表示を行ないます。デモ表示中でもスイッチを押すとゲームスタートします。ゲームオーバーのときはメロディの演奏が終わったらゲームスタート待ちに戻ります。

◆

この仕様書からどんなプログラムが出来上がるのか，最初にイメージできていると次のページからの説明が理解しやすいと思います。ここで，プログラムを実際に動かしてみましょう。

このプログラムはサイズの関係で「Hterm」で RAM にダウンロードすることはできません。それで，FDT を使って H8/3052 のフラッシュメモリにダウンロードし電源オンですぐに動くようにします。

フラッシュメモリにダウンロードするプログラムは，付属 CD 内の「tetris. mot」です。FDT の使い方については CD 内のマニュアル，「TK-3052 ユーザーズマニュアル（TK3052 ﾏﾆｭｱﾙ. pdf）」の 9 ページからと，「ルネサスダウンロード. pdf」の説明を参考にして下さい。

なお，フラッシュメモリの内容を書き換えてしまうので「Hterm」は使えなくなります。再び「Hterm」を使うときは，CD 内の「monitor. mot」を FDT でダウンロードしてください（FDT の使い方は前述の資料を参照）。

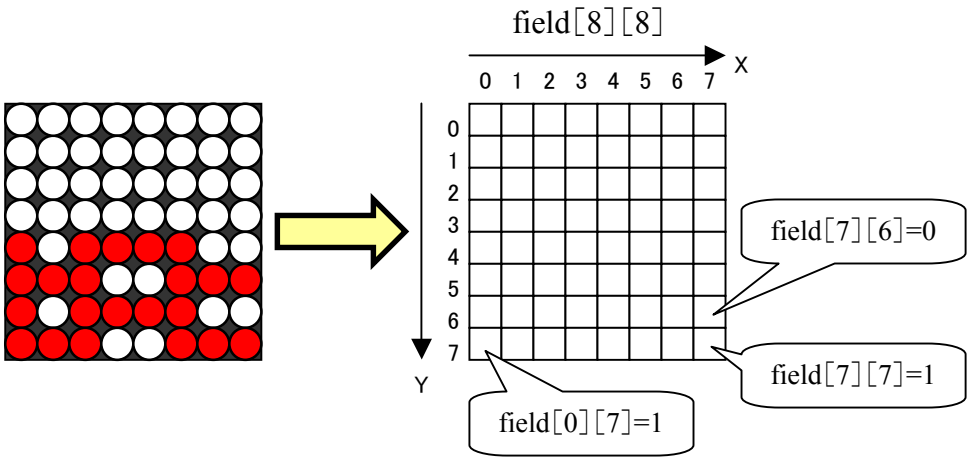
◆

このプログラムでは標準処理用ライブラリ「stdlib. h」を使用します。自分でプロジェクトから作成するときは，「新規プロジェクト-4／9-標準ライブラリ」で次のように指定してください。

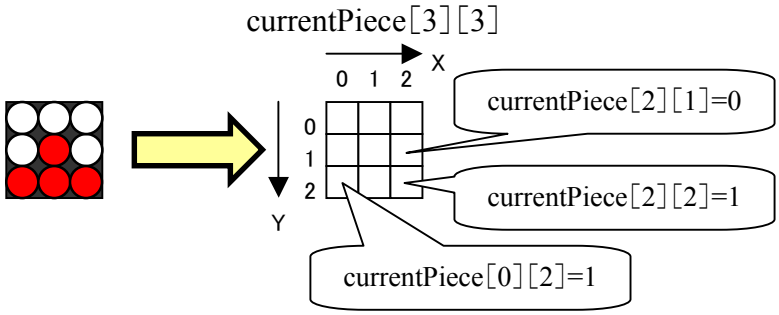
2. データの設計

仕様が決まったら，仕様どおりのプログラムを作るために，どのようなデータ構造にすればよいか検討します。ここをきちんと設計するかどうかが，すっきりしたプログラムになるか，ごちゃごちゃしたプログラムになるかの分かれ道になります。

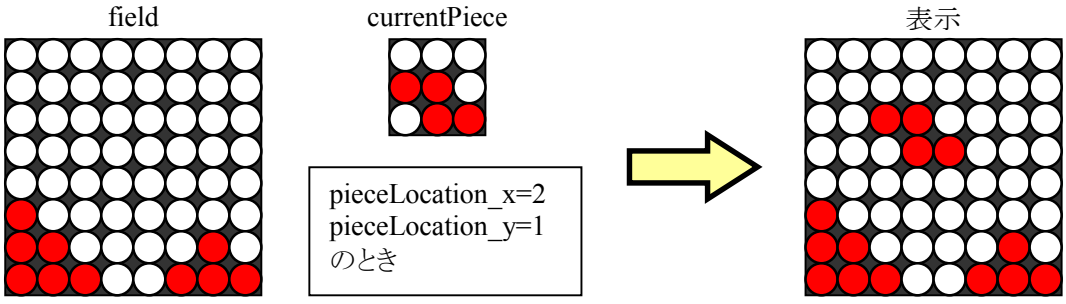
仕様書によるとゲーム画面は 8×8 ドットでした。そして，1 ドットづつ「ブロックがある/ブロックがない」という情報を持っていなければなりません。とすると，要素の数が 8×8 の二次元配列（field）を用意し，1 でブロックがある，0 でブロックがない，というように表現すればよさそうです。



次は落ちてくるブロックをどのように表現するかです。一つの案は，ゲーム画面を表す field の中に含めてしまう，という方法です。しかし，落下ブロックの移動や回転の際に，移動先のドットにすでにブロックがあるかどうかを判別していく必要があるため，複雑になりそうな予感がします。そこで，二つ目の案として，落下ブロックを表す二次元配列（currentPiece）を別に用意する方法を採用します。



currentPiece[0][0]が field のどこに位置するかを，pieceLocation_x と pieceLocation_y にセットすることにします。画面に表示するときは pieceLocation_x と pieceLocation_y をもとに field と currentPiece を重ねあわせて表示します。

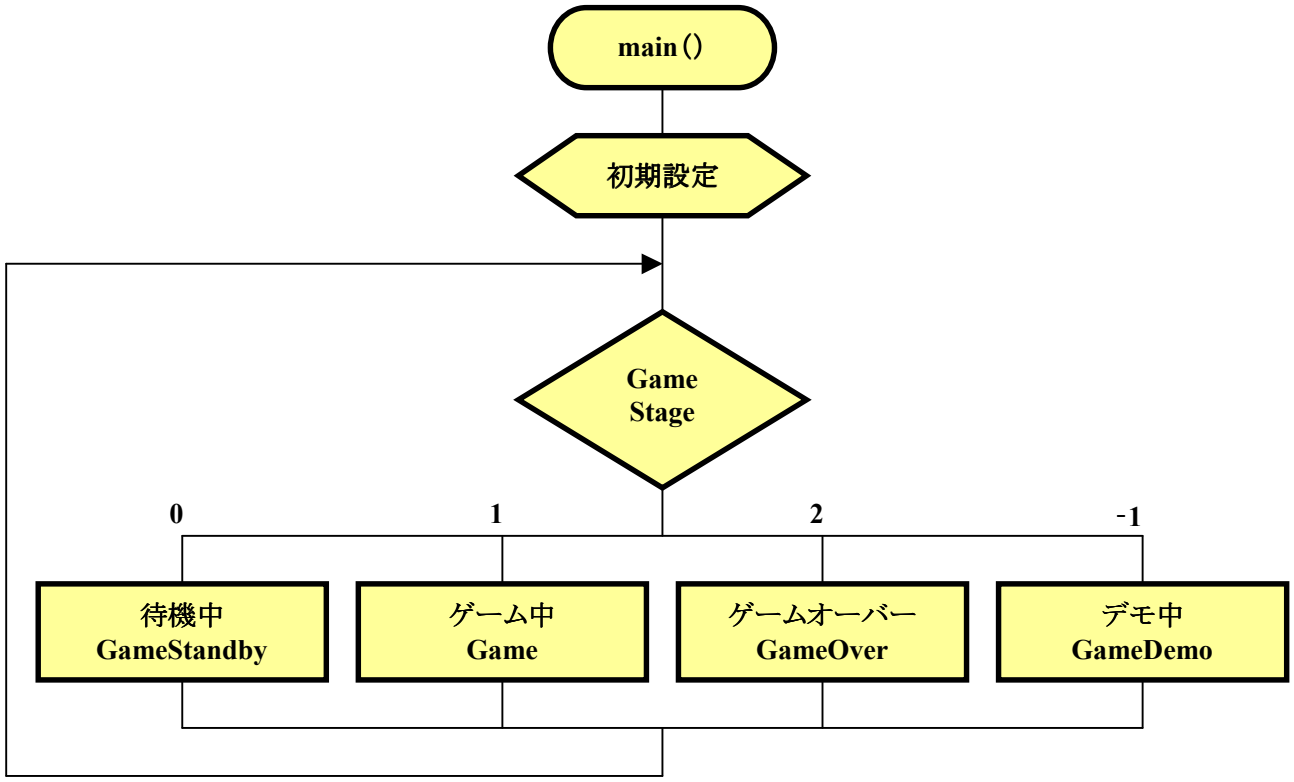


今までの説明をソースリストにまとめると次のようになります。

```
/*************************************************************************
    定数の定義（直接指定）
*************************************************************************/
#define       OK            0         //戻り値
#define       NG            -1        //戻り値
#define       TRUE          0         //戻り値
#define       FALSE         -1        //戻り値

// テトリス ------------------------------------------------------------
#define       FIELD_WIDTH       8     //ゲームフィールド(横)
#define       FIELD_HEIGHT      8     //                (縦)
#define       PIECE_LEFT        2     //ブロック左移動
#define       PIECE_RIGHT       4     //ブロック右移動
#define       PIECE_DOWN        8     //ブロック下移動
#define       MOVE_TIM_1        100   //ブロック移動間隔(100×10ms=1.00s)
#define       MOVE_TIM_2        95    //ブロック移動間隔( 95×10ms=0.95s)
#define       MOVE_TIM_3        90    //ブロック移動間隔( 90×10ms=0.90s)
#define       MOVE_TIM_4        85    //ブロック移動間隔( 85×10ms=0.85s)
#define       MOVE_TIM_5        80    //ブロック移動間隔( 80×10ms=0.80s)
#define       MOVE_TIM_6        75    //ブロック移動間隔( 75×10ms=0.75s)
#define       MOVE_TIM_7        70    //ブロック移動間隔( 70×10ms=0.70s)
#define       MOVE_TIM_8        65    //ブロック移動間隔( 65×10ms=0.65s)
#define       MOVE_TIM_9        60    //ブロック移動間隔( 60×10ms=0.60s)
#define       MOVE_TIM_10       55    //ブロック移動間隔( 55×10ms=0.55s)
#define       MOVE_TIM_11       50    //ブロック移動間隔( 50×10ms=0.50s)
#define       MOVE_TIM_12       45    //ブロック移動間隔( 45×10ms=0.45s)

/*************************************************************************
    グローバル変数の定義とイニシャライズ(RAM)
*************************************************************************/
// テトリスに関する変数 ------------------------------------------------
char          GameStage =    0;   //ゲームステージ
                                  // 0:待機中
                                  // 1:ゲーム中
                                  // 2:ゲームオーバー
                                  //-1:デモ画面
int           Point;               //点数
char          field[FIELD_WIDTH][FIELD_HEIGHT]; //ゲームフィールド
char          currentPiece[3][3];//現在移動中のブロック
int           pieceLocation_x;    //ブロックの位置(X座標)
int           pieceLocation_y;    //ブロックの位置(Y座標)
unsigned int  MoveTimCount;       //ブロック移動間隔カウンタ
unsigned int  MoveTimHalfCount;   //ブロック移動間隔カウンタの1/2
```

3. メインプログラム

メインプログラムでは I/O やワークエリアの初期設定のあと, GameStage によって「待機中」,「ゲーム中」,「ゲームオーバー」,「デモ中」に振り分けます。




```
/*****************************************************************************
    メインプログラム
*****************************************************************************/
void main(void)
{
    // イニシャライズ ------------------------------------------------
    ini_io();
    ini_itu();

    init_soft_timer();
    init_tetris();

    // メインループ --------------------------------------------------
    while(1){
        switch(GameStage){
            // 待機中 ----------------------------------------------
            case 0:
                GameStandby();
                break;

            // ゲーム中 --------------------------------------------
            case 1:
                Game();
                break;

            // ゲームオーバー --------------------------------------
```

```
            case 2:
                GameOver();
                break;

            // デモ画面 ----------------------------------------------
            case -1:
                GameDemo();
                break;
        }
    }
}

/*****************************************************************************
    ゲーム
*****************************************************************************/
void Game(void)
{
    if      ((SwData4 & 0x08)==0x08){      //SW1が押された
        MovePiece(PIECE_RIGHT);
        SwData4 = 0;
    }
    else if ((SwData4 & 0x10)==0x10){      //SW2が押された
        MovePiece(PIECE_LEFT);
        SwData4 = 0;
    }
    else if ((SwData4 & 0x20)==0x20){      //SW3が押された
        TurnPiece();
        SwData4 = 0;
    }

    Paint(); //ゲームフィールドの表示
    setLED(); //周囲のLEDの表示
}
```

次の章から「ゲーム中」のプログラムの内容を説明します。「待機中」,「ゲームオーバー」,「デモ中」のプログラムについてはソースリストをご覧下さい。また, LED 表示, スイッチ入力, メロディの演奏は, これまで作成してきたプログラムを応用しています。

4. ゲームアクションのプログラム

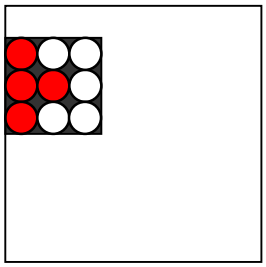
ゲームプログラムの中心はアクション部分です。「テトリス」の場合，一定時間毎にブロックを落下させたり，スイッチにあわせてブロックを左右に移動させたり回転させたり，一行そろったら削除したり，というアクションがあります。前のページのフローチャートで言えば，「ゲーム中」に行なう動作です。では，この部分を考えてみましょう。(ソースリストも併せてご覧下さい)

■ ブロックの移動(MovePiece)

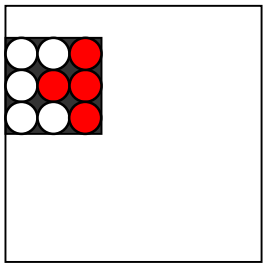
ブロックを左右に動かすことができるかどうか，ブロックを落下させることができるかどうかは，どのように判定すればよいでしょうか。ここでは左に動かすことを例に考えてみましょう。

まず，画面の大きさは決まっているので，それを越えて動くことはできません。それで，ブロックを左に動かそうとしているとき，すでに左端になっているなら動かさないようにします。ということは，pieceLocation_x=0 のときに左に動かさないということでよいでしょうか？

ところが，そう単純ではありません。下の図をご覧下さい。どちらも pieceLocation_x=0 です。左図は動かすことができません。しかし，右図はもう一列動かすことができます。つまり，右図の場合は pieceLocation_x=－1 まで動かすことができます。それで，currentPiece の最左列にブロックがあるかどうかで，判定値を使い分ける必要があります。



pieceLocation_x=0
まで動かせる



pieceLocation_x=－1
まで動かせる

もう一つ，移動できるかどうかを判別する要素は，移動先に別のブロックがあるかどうかです。もちろん，移動先にすでにブロックがあるなら移動できません。この二つの要素を判定して移動可能なときに，pieceLocation_x をマイナス 1 します。

ここでは左移動について考えてきましたが，右移動，下移動も考え方は一緒です。プログラムではブロックの移動は‘MovePiece’関数で行なっています。移動方向は関数をコールするときに引数で指定します。それで，SW1 が押されたときは右移動の引数(PIECE_LEFT)をセット，SW2 が押されたときは左移動の引数(PIECE_RIGHT)をセット，一定時間毎に下移動の引数(PIECE_DOWN)をセットして‘MovePiece’関数をコールします。

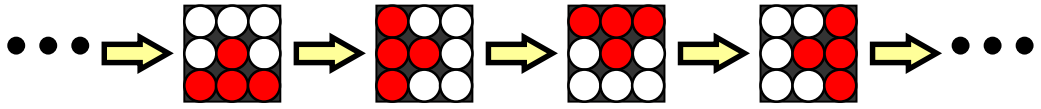
```
/*************************************************************************
      ブロックの移動判定
*************************************************************************/
int MovePiece(int move)
{
      int left,right,bottom;
      int x,y,count;

      // 左移動 ------------------------------------------------------------
      if (move==PIECE_LEFT) {
            left = GetPieceLeft();      //ブロック左側の位置情報

            if (pieceLocation_x > -(left-1)) {    //左側に移動できる
                  //移動先のブロックと重なるか?
                  count = 0;
                  for (y=0; y<3; y++) {
                        for (x=0; x<3; x++) {
                              if (((pieceLocation_x+x-1)>=0)&& ((pieceLocation_y+y)>=0)){
                                    if (currentPiece[x][y] && field[pieceLocation_x + x - 1][pieceLocation_y + y]) {
```

```
                                    count++;
                               }
                          }
                     }
                }
                if (!count) {
                     pieceLocation_x--;   //移動
                }
                else {
                     return FALSE;
                }
           }
           else {
                return FALSE;
           }
     }
     // 右移動 ------------------------------------------------------------
     else if (move==PIECE_RIGHT) {
           right = GetPieceRight();  //ブロック右側の位置情報

           if (pieceLocation_x < (FIELD_WIDTH-right)) {    //右側に移動できる
                //移動先のブロックと重なるか?
                count = 0;
                for (y=0; y<3; y++) {
                     for (x=0; x<3; x++) {
                          if (((pieceLocation_x+x+1)>=0)&& ((pieceLocation_y+y)>=0)){
                               if (currentPiece[x][y] && field[pieceLocation_x + x + 1][pieceLocation_y + y]) {
                                    count++;
                               }
                          }
                     }
                }
                if (!count) {
                     pieceLocation_x++;   //移動
                }
                else {
                     return FALSE;
                }
           }
           else {
                return FALSE;
           }
     }
     // 下移動 ------------------------------------------------------------
     else if (move==PIECE_DOWN) {
           bottom = GetPieceBottom(); //ブロック下側の位置情報

           if (pieceLocation_y < (FIELD_HEIGHT-bottom)) {  //下側に移動できる
                //移動先のブロックと重なるか?
                count = 0;
                for (y=0; y<3 ; y++) {
                     for (x=0; x<3; x++) {
                          if (((pieceLocation_x+x)>=0)&& ((pieceLocation_y+y+1)>=0)){
                               if (currentPiece[x][y] && field[pieceLocation_x + x][pieceLocation_y + y + 1]) {
                                    count++;
                               }
                          }
                     }
                }
                if (!count) {
                     pieceLocation_y++;   //移動
                }
```

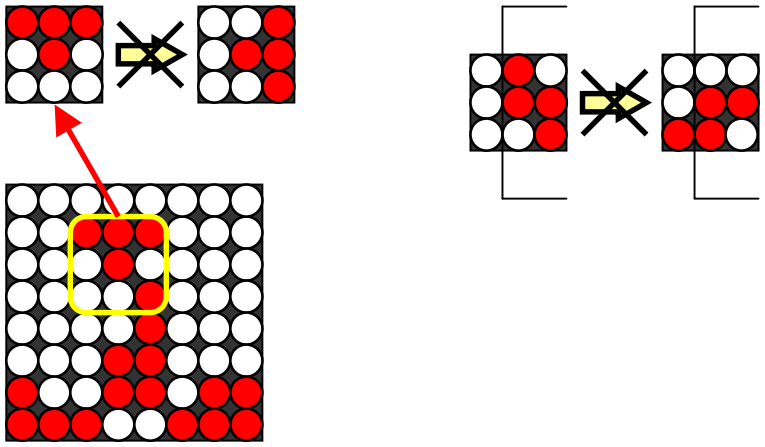
```
                else {
                    return FALSE;
                }
            }
            else {
                return FALSE;
            }
        }

        return TRUE;
}
```

■ **ブロックの回転(TurnPiece)**

currentPiece の回転は‘TurnPiece’関数で行ないます。SW3 が押されるとコールしますが，一回コールすると時計回りに 90 度回転します。



ブロックの回転にもできる場合とできない場合があります。回転した結果，別のブロックと重なるようなら回転できません。また，回転した結果，ゲーム画面をはみ出すときも回転できません。



```
/*****************************************************************************
    ブロックを回転させる
*****************************************************************************/
int TurnPiece(void)
{
    int x,y,offsetX;
    int copy[3][3];

    //回転したブロックを生成する
    for (y=0; y<3; y++){
        for (x=0; x<3; x++){
            copy[2-y][x] = currentPiece[x][y];
        }
    }
    //回転可能かどうかを調べる
    for (y=0; y<3; y++){
        for (x=0; x<3; x++){
            if (copy[x][y]){
                offsetX = pieceLocation_x + x;
```

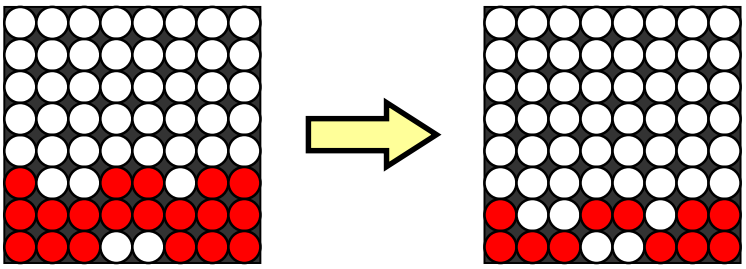
```
                    if ((offsetX < 0) || (offsetX >= FIELD_WIDTH) || field[offsetX][pieceLocation_y + y])
                         return FALSE;
               }
          }
     }

     //copyをCurrentPieceにコピーする
     for (y=0; y<3; y++){
          for (x=0; x<3; x++){
               currentPiece[x][y] = copy[x][y];
          }
     }
     return TRUE;
}
```

■ 一定時間毎のブロック判定(TimerProc)

一定時間毎に下移動の引数(PIECE_DOWN)をセットして‘MovePiece’関数をコールします。ここで，落下ブロックが下に移動できない場合は下まで到達したと判定します。

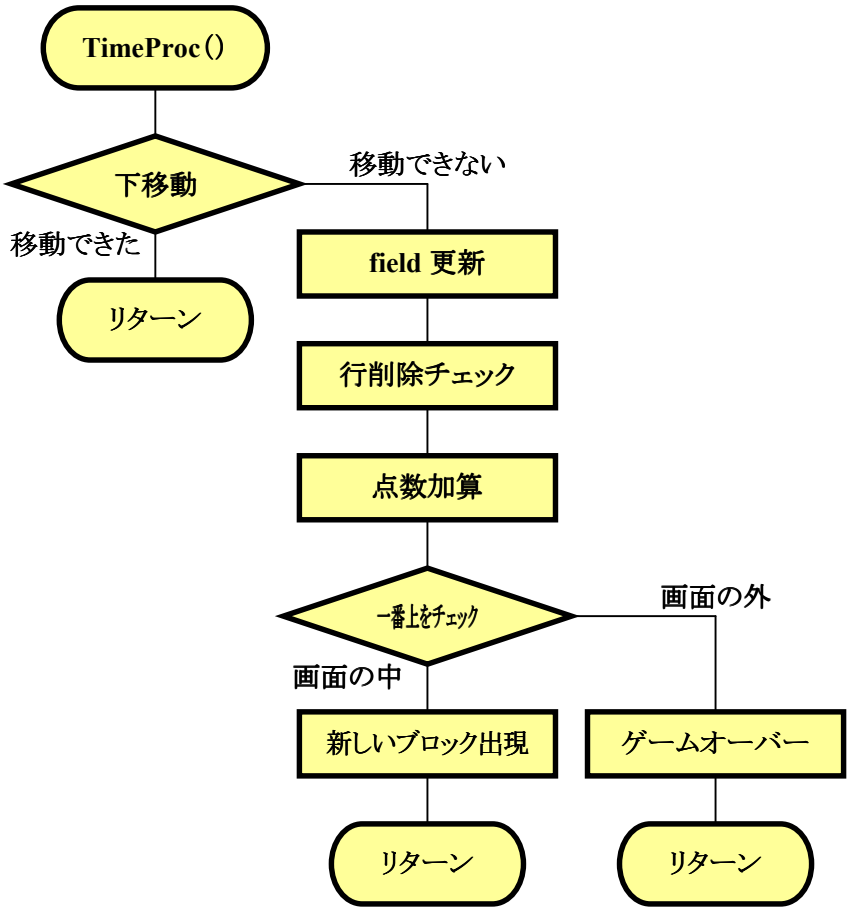
落下ブロックが下まで到達したら currentPiece を field にコピーします。

次に，一行そろっている行がないかチェックします。そろっている行は削除し，その行より上のブロックを一行下にずらします。



ここで，点数を加算します。削除した行数×行数で点数を計算します。

最後にブロックの一番上がゲーム画面の最上部を越えていないかチェックします。越えていたらゲームオーバー，まだ大丈夫なら新しいブロックを出現させます。




```
/*****************************************************************************
     1秒ごとのブロック判定
*****************************************************************************/
void TimerProc(void)
{
     int line, top;

     if (MovePiece(PIECE_DOWN)==FALSE){    //これより下に移動できない
          PieceToField();
          line = AdjustLine(); //行チェック, 1行埋まっていたらその行は削除
          Point = Point + line * line;     //削除した行数の2乗が点数になる
          top = GetPieceTop();
          if ((pieceLocation_y + line + top - 1) < 0){
               GameStage = 2;  //ゲームオーバー
          }
          else{
               NextPiece();     //次のブロック
          }
     }
}
```

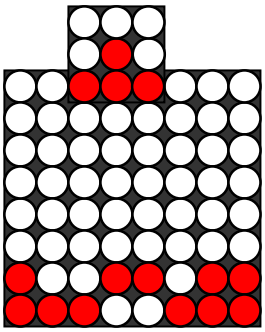
■ **新しいブロックの出現（NextPiece）**

まずは 7 種類のブロックの中から，どのブロックにするかランダムに選択します。そのために乱数を利用します。

乱数は‘rand()’関数として HEW に用意されています。‘rand()’関数は戻り値として 0～RAND_MAX の整数値を返します。次の式を使えば 0 以上 7 未満（6.9999…）の範囲の乱数を作ることができます。このうち整数部分だけ取り出して 0～6 の乱数を得ます。

$$\frac{rand()}{RAND\_MAX + 1} \times 7$$

currentPiece の最初の位置は，pieceLocation_x=2，pieceLocation_y=－2 にします。つまり，次のような状態で新しいブロックが出現することになります。



```
/*********************************************************************
     次のブロックへ
*********************************************************************/
void NextPiece(void)
{
     int x,y,num;

     for (y=0; y<3; y++){
          for (x=0; x<3; x++){
               currentPiece[x][y] = 0;
          }
     }

     num = (int)(((double)rand() / ((double)RAND_MAX + 1)) * 7);      //0～6の乱数を作る

     //ブロックの選択
     switch(num){
          case 0:
               currentPiece[1][0] = 1;
               currentPiece[1][1] = 1;
               currentPiece[1][2] = 1;
               break;
          case 1:
               currentPiece[1][1] = 1;
               currentPiece[2][1] = 1;
               currentPiece[1][2] = 1;
               break;
          case 2:
               currentPiece[1][1] = 1;
               currentPiece[2][1] = 1;
               currentPiece[2][2] = 1;
               break;
          case 3:
               currentPiece[0][2] = 1;
               currentPiece[1][1] = 1;
```

```
                currentPiece[1][2] = 1;
                currentPiece[2][1] = 1;
                break;
          case 4:
                currentPiece[0][1] = 1;
                currentPiece[1][1] = 1;
                currentPiece[1][2] = 1;
                currentPiece[2][2] = 1;
                break;
          case 5:
                currentPiece[0][2] = 1;
                currentPiece[1][1] = 1;
                currentPiece[1][2] = 1;
                currentPiece[2][2] = 1;
                break;
          case 6:
                currentPiece[1][1] = 1;
                currentPiece[1][2] = 1;
                currentPiece[2][1] = 1;
                currentPiece[2][2] = 1;
                break;
     }

     //位置の初期設定
     pieceLocation_x =  2;
     pieceLocation_y = -2;
}
```

■ 表示データのセット(Paint)

ゲーム中は field と currentPiece を重ねあわせたデータを DispBuf にセットします。

```
/*****************************************************************************
     表示データセット
*****************************************************************************/
void Paint(void)
{
     int x,y,n;
     char d,copy_field[8][8];

     //fieldをコピー
     for (y=0; y<8 ; y++){
          for (x=0; x<8; x++){
                copy_field[x][y] = field[x][y];
          }
     }

     //currentPieceを重ねる
     for (y=0; y<3 ; y++){
          for (x=0; x<3; x++){
                if (currentPiece[x][y]){
                     if (((pieceLocation_x+x)>=0)
                      && ((pieceLocation_x+x)<FIELD_WIDTH)
                      && ((pieceLocation_y+y)>=0)
                      && ((pieceLocation_y+y)<FIELD_HEIGHT)){
                           copy_field[pieceLocation_x + x][pieceLocation_y + y] = currentPiece[x][y];
                     }
                }
          }
```

```
    }

    //表示データに変換
    for (x=0; x<8; x++){
        d = 0;
        n = 1;
        for (y=0; y<8; y++){
            d = d + (unsigned char)(copy_field[x][y] * n);
            n = n * 2;
        }
        DispBuf[x] = d;
    }
}
```

5. プログラムを改造する

これまでのところで「テトリス」の基本的な動作はできるようになりました。十分遊べるレベルになっていると思います。それでも，ゲームをより楽しめるように工夫するところはたくさんあると思います。一例をご紹介します。

■ 落下ブロックの追加や変更

このプログラムでは 7 種類のブロックからランダムに選ぶようになっています。さらに種類を増やすことができます。ブロックの形によってはゲームの難易度がかなり高くなると思います。

■ 落下ブロックの出現箇所の変更

新しいブロックの出現箇所は固定になっています。もしこれがランダムに変更されるとしたら，先を読むことができないのでかなり難しくなるでしょうね。pieceLocation_xを 0～5 の範囲でランダムに設定すれば実現できます。最初からだと難易度が高すぎるので，ある程度点数を取ってからこのモードに入るようにしたらどうでしょうか。

■ 音の追加

ゲームに音は不可欠です。現在はゲームオーバーのときだけメロディを演奏していますが，ゲーム中 BGM を流してみてはどうでしょうか。また，行を削除するときに効果音を付けても面白いかもしれません。

■ ルールの変更

現在はひたすら行を削除して，点数を競うようになっています。これをクリア制にしてはどうでしょうか。周囲の LED は点数を加算する毎に点灯していきますが，全て点灯したらステージクリアとします。ステージ毎に落下するブロックの形の難易度を上げたり，落下スピードを速くしたりして，段々難しくすることができるかもしれません。

◆

自分でプログラムするということは，自分の考えた世界をマイコン上に実現できるということです。ゲームプログラムの場合，特にこの世界観というものがかなり強く反映されると思います。いろいろ工夫して自分なりの「テトリス」を作ってみてください。

第16章 デジタルストレージオシロスコープの作成

マイコンとパソコンを組み合わせると，さらに応用が広がります。この章では，デジタルストレージオシロスコープ（DSO）の作成を通して，マイコンとパソコンを組み合わせる方法を考えてみましょう。

1. DSO の仕様

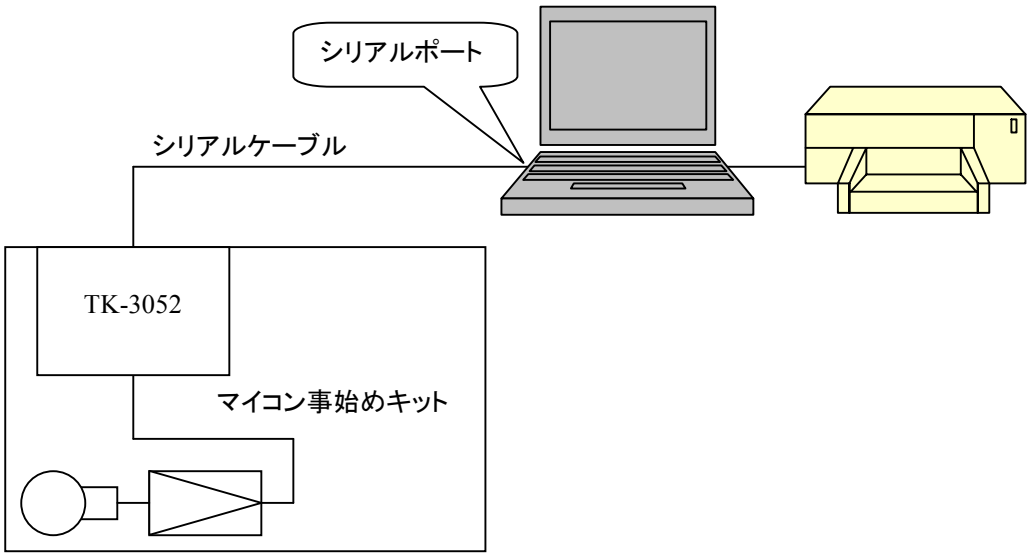
今回作成する DSO の機能は次のとおりです。

4 チャンネル入力
データ 8 ビット
サンプリング数 256 個/1 チャンネル，サンプリング周波数はそれにあわせて設定
トリガはオートトリガとエッジトリガをサポート
連続掃引とシングル掃引をサポート
波形表示
印刷機能搭載

このうち，操作や表示といったマン・マシーンのインターフェースの部分はパソコンで行ないます。また，印刷機能もパソコンにつながっているプリンタで行なったほうが楽です。アナログ信号をデジタル値で取り込む部分は TK-3052 で行ないます。

なお，この章の実習では，ジョイント基板にコンデンサマイクとマイクアンプを実装してチャンネルの一つに入力し，音声波形をパソコンに表示することを目標にします。

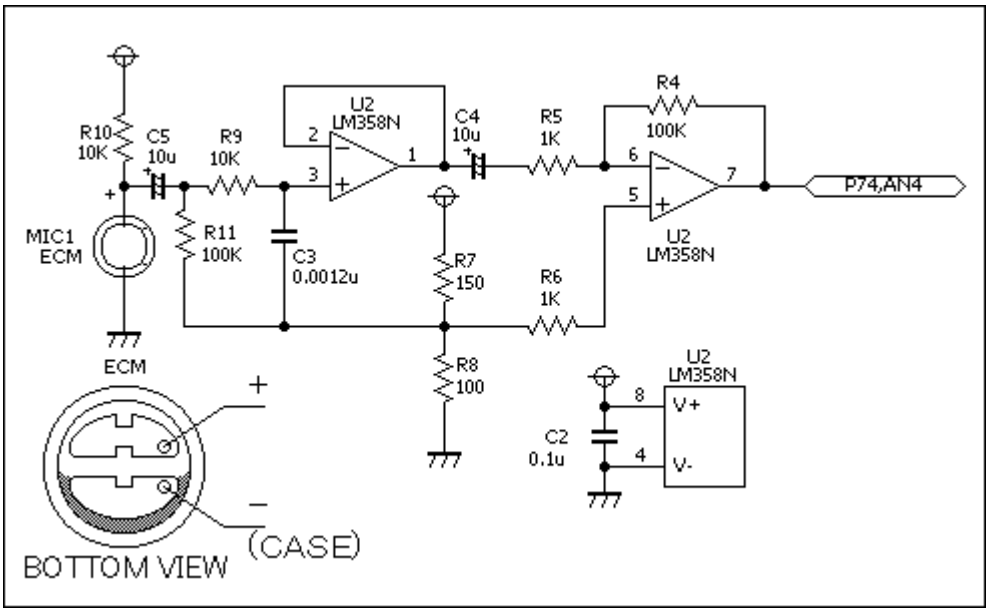
全体の構成は次のとおりです。



ところで，最近のパソコンはシリアルポートがありません。その場合はUSB-シリアル変換ケーブルを用意します。シリアルポートのポート番号はパソコンによって変化しますので，パソコンプログラムでポート番号を指定できるようにします。

2. ハードウェア

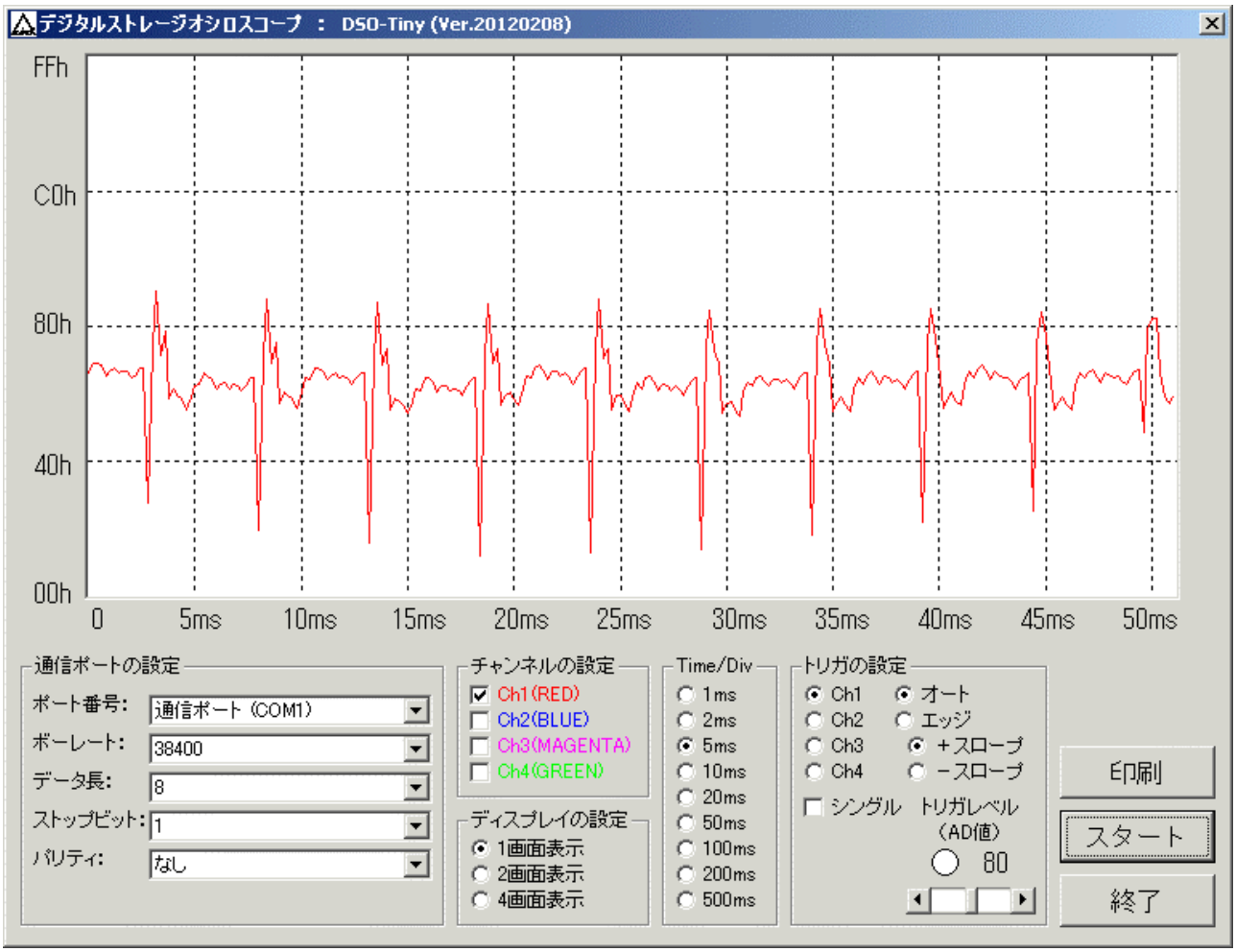
マイコン事始めキットの回路は，AN0 に CDS が接続済みです。それで，マイク回路は AN4 につないで，AN4～AN7 の 4 チャンネルを DSO のチャンネル 1～4 入力にします。あとで説明しますが，H8/3052 に搭載されている AD 変換器は AN0～AN3 をグループ 1，AN4～AN7 をグループ 2 としているからです。



マイク回路は右のとおりです。なお，巻末の付録にマイク回路を含めた「デジタルストレージオシロスコープ追加版　ジョイント基板の部品表，回路図，実装例」が掲載されています。参考にして組み立ててください。（マイク回路はオプション，別売，入手方法は弊社 WEB ページをご覧ください。）

3. 操作画面の設計

操作画面は次のとおりです。（DSO の機能全体もわかるかと思います）



通信ポートの設定

ポート番号は TK-3052 を接続するシリアルポートを指定してください。他はこの図のとおりです。

チャンネルの設定

波形を表示したいチャンネルにチェックを入れます。マイク入力（AN4）は Ch1 です。（AN5＝Ch2，AN6＝Ch3，AN7＝Ch4）

ディスプレイの設定

1 画面表示　：　Ch1～4 を一つの画面に重ねて表示します。
2 画面表示　：　二つの画面に分割し，Ch1 と 3，Ch2 と 4 を重ねて表示します。
4 画面表示　：　Ch1～4 を個別の画面に表示します。

Time/Div

一目盛の時間を選択します。

トリガの設定

Ch1～4　：　トリガをかけるチャンネルを選択します。
シングル　：　一回だけトリガをかけるときはチェックします。（トリガがかかるとストップします）
オート　：　トリガ条件に関係なく信号を取り込み，波形を表示します。
エッジ　：　トリガ条件（スロープとトリガレベル）で信号を取り込み，波形を表示します。
＋スロープ　：　トリガレベルを上回ったら，信号を取り込みます。
－スロープ　：　トリガレベルを下回ったら，信号を取り込みます。
トリガレベル　：　AD 値で指定します。スライダで設定してください。

スタート

設定が終わったらこのボタンをクリックして測定を開始します。止めるときも，このボタンをクリックします。

印刷

現在表示されている波形をパソコンに接続されているプリンタに印刷します。

終了

プログラムを終了し，ウィンドウを閉じます。（ウィンドウ右上の×と同じ）

4. 動かしてみよう

どのような動作になるのか最初にイメージできていると次のページからの説明が理解しやすいと思います。ここで，プログラムを実際に動かしてみましょう。

このプログラムはワークエリアのサイズの関係で「Hterm」で RAM にダウンロードすることはできません。それで，FDT を使って H8/3052 のフラッシュメモリにダウンロードし電源オンですぐに動くようにします。

フラッシュメモリにダウンロードするプログラムは，付属 CD 内の「Dso3052．mot」です。FDT の使い方については CD 内のマニュアル，「TK-3052 ユーザーズマニュアル（TK3052 マニュアル．pdf）」の 9 ページからと，「ルネサスダウンロード．pdf」の説明を参考にして下さい。

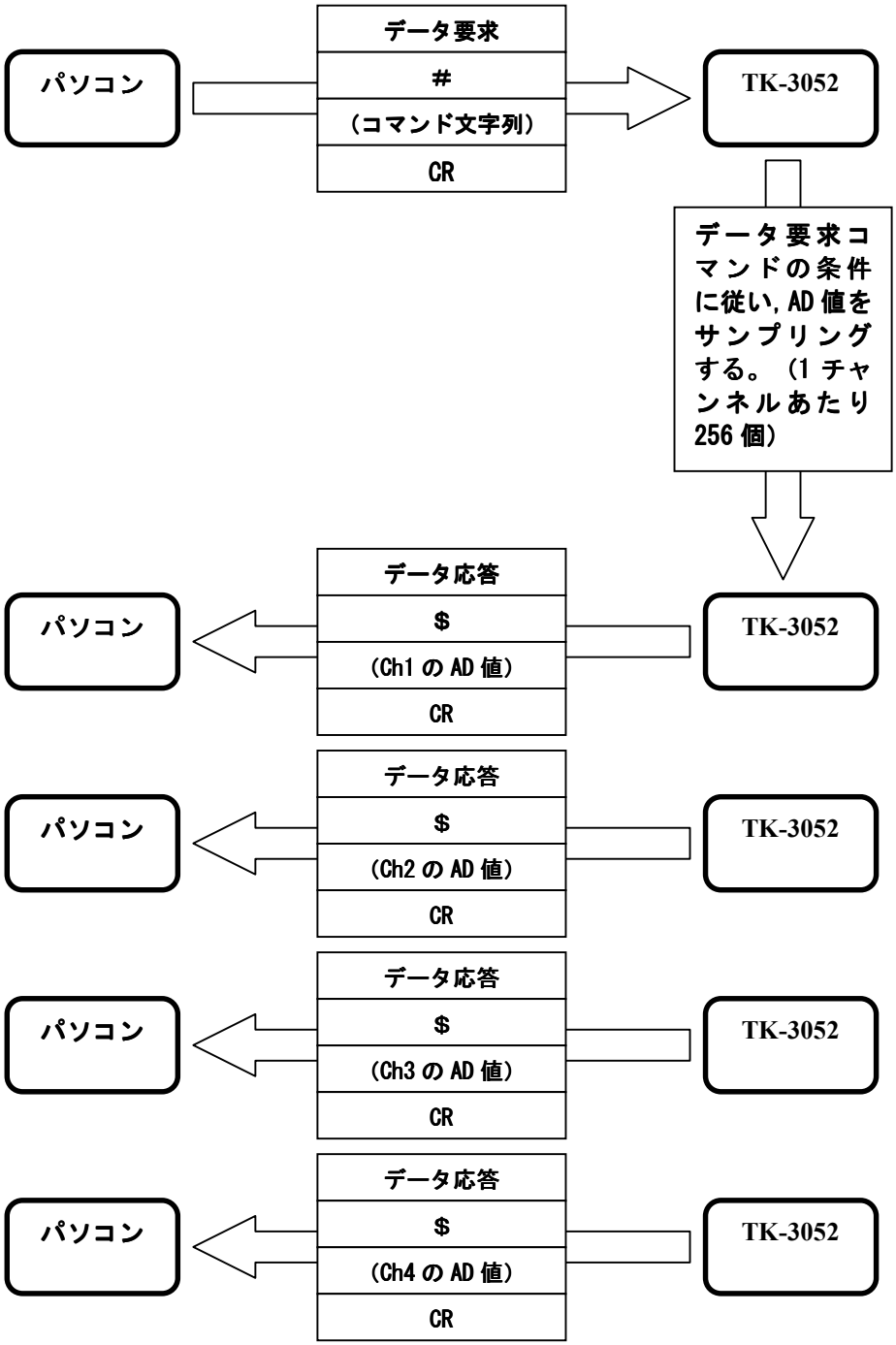
なお，フラッシュメモリの内容を書き換えてしまうので「Hterm」は使えなくなります。再び「Hterm」を使うときは，CD 内の「monitor．mot」を FDT でダウンロードしてください（FDT の使い方は前述の資料を参照）。

続いてパソコンのプログラムを起動します。付属 CD 内の「DsoTiny．exe」をダブルクリックします。なお，このプログラムを実行するためには標準ライブラリのほかに「MSCOMM32．OCX」が必要です。VB6 をインストールしてあれば問題ありませんが，そうでないときは別途入手する必要があります。詳しくは弊社 CD の「_始めにお読みください」フォルダ内の「VB プログラムについて．pdf」をご覧ください。

最初は「オート」になっているので「スタート」をクリックするとすぐに AD 値を取り込み波形を表示します。マイクに音を入力したり，画面をいろいろ操作したりしてみてください。

5. 通信プロトコル

最初にパソコンはデータ要求を送信します。TK-3052 はパソコンからデータ要求を受信すると，指定された条件で AD 値を 4 チャンネル分（1 チャンネルあたり 256 個）サンプリングし，全てのサンプリングが終了するとパソコンに取得したデータを送信します。トリガに「シングル」が指定されているときはこれで終了ですが，指定されていないときはパソコンは再びデータ要求を送信します。（下図参照）



●「データ応答」は「データ要求」で指定されたチャンネルのみ返信する。
●TK-3052 は“／”（2Fh）を受信すると，サンプリング中であっても途中で中止し，パソコンからのコマンド待ち状態になる。（キャンセル機能）

「データ要求」と「データ応答」の詳細は次のページで説明します。

■ **パソコン→TK-3052 ： データ要求**

		例
ヘッダー	#(23h)	23h
チャンネル 1	0(30h)＝ディセーブル / 1(31h)＝イネーブル	31h
チャンネル 2	0(30h)＝ディセーブル / 1(31h)＝イネーブル	30h
チャンネル 3	0(30h)＝ディセーブル / 1(31h)＝イネーブル	30h
チャンネル 4	0(30h)＝ディセーブル / 1(31h)＝イネーブル	30h
区切り	,(2Ch)	2Ch
トリガチャンネル	1(31h)＝Ch1,2(32h)＝Ch2,3(33h)＝Ch3,4(34h)＝Ch4	31h
トリガモード	0(30h)＝オートトリガ / 1(31h)＝エッジトリガ	30h
トリガスロープ	0(30h)＝プラス / 1(31h)＝マイナス	30h
トリガレベル（上位桁）	0000～FF00, 上位二桁有効, アスキーコードで指定	38h
↑		30h
↑		30h
トリガレベル（下位桁）		30h
区切り	,(2Ch)	2Ch
Time/Div（上位桁）	00-1ms/div,01-2ms/div,02-5ms/div,03-10ms/div,04-20ms/div,05-50ms/div	30h
Time/Div（下位桁）	06-100ms/div,07-200ms/div,08-500ms/div, アスキーコードで指定	32h
デリミタ	CR(0Dh)	0Dh

「例」は 165 ページの操作画面のときのデータ要求コマンド。

■ **TK-3052→パソコン ： データ応答**

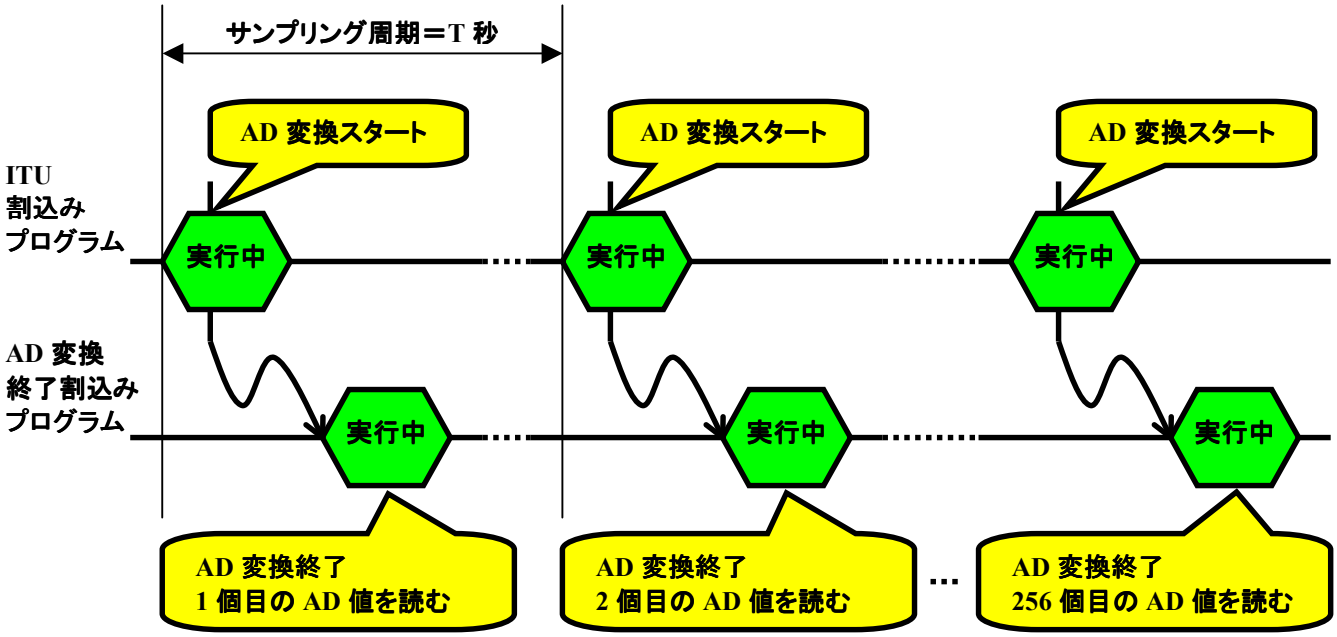
		例
ヘッダー	$(24h)	24h
チャンネル	1(31h)＝Ch1,2(32h)＝Ch2,3(33h)＝Ch3,4(34h)＝Ch4	31h
区切り	,(2Ch)	2Ch
AD 値-0（上位桁）	00～FF, アスキーコード	38h
AD 値-0（下位桁）		30h
AD 値-1（上位桁）	00～FF, アスキーコード	38h
AD 値-1（下位桁）		46h
AD 値-2（上位桁）	00～FF, アスキーコード	41h
AD 値-2（下位桁）		38h
・ ・ ・	・ ・ ・	・ ・ ・
AD 値-254（上位桁）	00～FF, アスキーコード	32h
AD 値-254（下位桁）		37h
AD 値-255（上位桁）	00～FF, アスキーコード	33h
AD 値-255（下位桁）		30h
デリミタ	CR(0Dh)	0Dh

「例」はチャンネル 1 のデータ応答。

6. TK-3052 のプログラム

プログラムの詳細は付属 CD 内のソースファイル「Dso3052. c」をご覧ください。ここでは，AD 変換を中心に解説します。

DSO で大切なのは，等間隔でサンプリングする，ということです。できるだけ正確に行ないたいので，ITU でタイミングを作ります。そして，ITU の割込みプログラムで AD 変換をスタートすれば，等間隔でサンプリングできます。ITU 割込みプログラムと AD 変換終了割込みプログラムのタイミングチャートは次のようになります。



まず，ITU 割込みプログラムを見てみましょう。

```
/*************************************************************************
      ITU チャネル0 割込み
*************************************************************************/
#pragma regsave (intprog_itu0)
void intprog_itu0(void)
{
      ITU0.TSR.BIT.IMFA = 0;      //タイマステータスレジスタクリア

      AD.ADCSR.BYTE = _01111011B | AD_CH;         //4チャンネル,スキャンモード
      PA.DR.BIT.B6 = 1;      //AD変換スタートからAD変換割込みまでの時間

      if (SamplingCount==1){
            ITU.TSTR.BIT.STR0     =     0;              //TCNT0 停止
            ITU0.TIER.BYTE        =     _11111000B;     //割込みディセーブル
            ITU0.TCNT             =     0x0000;         //TCNT0=0
      }
}
```

上のリストの黄色マーキング行が AD 変換スタートを指示している部分です。AD 変換器の設定用レジスタ「ADCSR」の構造は次のとおりです。

A/D コントロール/ステータスレジスタ(ADCSR) ： アドレス=0xFFE8 番地(下位 16 ビット)				
ビット	ビット名	初期値	R/W	説明
7	ADF	0	R/W	A/D エンドフラグ。 0:[クリア条件]ADF=1 の状態で，ADF フラグをリードした後，ADF フラグに 0 をライトしたとき。 1:[セット条件](1)単一モード:A/D 変換が終了したとき。(2)スキャンモード:設定された全てのチャネルの A/D 変換が終了したとき。
6	ADIE	0	R/W	A/D インタラプトイネーブル。 0:A/D 変換終了による割り込み(ADI)要求を禁止。 1:A/D 変換終了による割り込み(ADI)要求を許可。
5	ADST	0	R/W	A/D スタート。 0:A/D 変換を停止。 1:(1)単一モード:A/D 変換を開始し，変換が終了すると自動的に 0 にクリア。(2)スキャンモード:A/D 変換を開始し，ソフトウェア，リセット，またはスタンバイモードによって 0 にクリアされるまで選択されたチャネルを順次連続変換。
4	SCAN	0	R/W	スキャンモード。 0:単一モード。 / 1:スキャンモード。
3	CKS	0	R/W	クロックセレクト。 0:変換時間＝266 ステート(MAX)。 1:変換時間＝134 ステート(MAX)。
2	CH2	0	R/W	チャネルセレクト。 [単一モード] 000:AN0 001:AN1 010:AN2 011:AN3 100:AN4 101:AN5 110:AN6 111:AN7 [スキャンモード] 000:AN0 001:AN0～1 010:AN0～2 011:AN0～3 100:AN4 101:AN4～5 110:AN4～6 111:AN4～7
1	CH1	0	R/W	
0	CH0	0	R/W	

マーキング行の中で「AD_CH」は AN0～3 を使うか AN4～7 を使うか指定する定数で，ソースリストの冒頭で 04h(2 進数で 00000100)に定義しています。それで，ADCSR＝7Fh(2 進数で 01111111)を設定しています。つまり，このプログラムでは AN4～7 をスキャンモードで AD 変換しています。

スキャンモードとは何でしょうか。第 8 章は単一モードで使いましたが，このときは指定したチャンネルの AD 変換が終了すると ADCSR の ADF＝1 になり，AD 変換割り込みがかかります。一方，スキャンモードのときは，設定したチャンネル全ての AD 変換を順番に行い，全て終了したときに ADCSR の ADF＝1 になり，AD 変換割込みがかかります。それで，1 回の AD 変換スタートで 4 チャンネル全ての AD 変換を行なうことができます。

AD 変換終了割込みのプログラムは次のようになっています。

```
/************************************************************************
     AD変換割込み
************************************************************************/
#pragma regsave (intprog_ad)
void intprog_ad(void)
{
     AD.ADCSR.BIT.ADST = 0;
     AD.ADCSR.BIT.ADIE = 0;
     AD.ADCSR.BIT.ADF = 0;
     PA.DR.BIT.B6 = 0;     //AD変換スタートからAD変換割込みまでの時間

     AdBuf[0][SamplingConst - SamplingCount] = AD.ADDRA / 0x100;
     AdBuf[1][SamplingConst - SamplingCount] = AD.ADDRB / 0x100;
     AdBuf[2][SamplingConst - SamplingCount] = AD.ADDRC / 0x100;
     AdBuf[3][SamplingConst - SamplingCount] = AD.ADDRD / 0x100;
```

```
    SamplingCount = SamplingCount - 1;
    if (SamplingCount==0){
        DsoStatus = 4;
        PA.DR.BIT.B5 = 0;     //最初のAD変換スタートから全データサンプリング終了までの時間
    }
}
```

黄色マーキング行が AD 値を読んでいる部分です。スキャンモードを使っているので，AD 変換終了割込みがかかったときには 4 チャンネル全ての AD 変換が終了しています。順番に読んで，それぞれ上位 8 ビットを AD バッファにストアします。

サンプリングの同時性について

DSO で大切な点の一つは「等間隔でサンプリングする」ということでした。ところで，複数のチャンネル入力がある場合はもう一つ大切なことがあります。それは，「全てのチャンネルを同時にサンプリングする」ということです。オシロスコープの画面を見るとわかるように，波形の横軸（時間軸）が同じ点は，同時に生じた出来事（電圧）を意味しているからです。

では，H8/3052 内蔵の AD 変換器で同時サンプリングができるのでしょうか。

残念ながら同時サンプリングはできません。ハードウェアマニュアルに掲載されている AD 変換器のブロック図を見ると，入力は 8 チャンネルありますが，そのあとのアナログマルチプレクサ回路でサンプル＆ホールド回路への入力を切り替えています。その信号が AD 変換されるわけなので，一度に 1 チャンネルのアナログ信号を AD 変換することしかできません。つまり，内蔵されている AD 変換器は 1 個だけなのです。

そのため，スキャンモードで AD 変換しても，それぞれのチャンネルの AD 値は，厳密には同時刻の値ではありません。CPU クロックが 25MHzのとき，スキャンモードの各チャンネル間には 5.12μs の時間差があります。つまり，AN4 と AN7には 15.36μs の時間差があることになります。

では，同時サンプリングするためにはどうすればよいのでしょうか。一つの方法は，AD変換器をチャンネルの数だけ用意して，同じタイミングで AD 変換をスタートすることです。コストは高くなりますが，もっとも単純で効果のある方法です。

別の方法は，AD 変換器は一つだけですが，サンプル＆ホールド回路をチャンネルの数だけ用意して同じタイミングでホールドし，あとはアナログマルチプレクサ回路で切り替えながら，順番にホールドした電圧を AD 変換します。アナログ回路が複雑になりますが，これも効果的な方法です。

「マイコン事始めキット」は学習用なので，サンプリングの同時性には目をつぶり，簡単な回路で実習できるようにしました。このプログラムを「DsoTiny」と呼んでいるのは DSO としては条件を満たしていない部分がある（小さな DSO，DSO に近いもの，という意味をこめている）からです。

では次に，サンプリング周期＝T 秒を決定し，ITU にセットする値を計算しましょう。AD 変換にはある程度の時間がかかりますから，それより短い間隔でサンプリングしようとしても不可能です。では，スキャンモード，AN4～AN7 選択の際，AD 変換スタート（ADST=1）から AD 変換終了（ADF=1）までにどれくらいの時間がかかるのでしょうか。

ハードウェアマニュアルに記載されていることをまとめると次のようになります。

	CKS=0			CKS=1		
	min	typ	max	min	typ	max
単一モードの AD 変換時間 スキャンモードの 1 回目の AD 変換時間 （単位　ステート）	259	－	266	131	－	134
スキャンモードの 2 回目以降の AD 変換時間 （単位　ステート）	－	256	－	－	128	－

「ステート」は，CPU クロック 1 周期の時間です。CKS=1 なので，AD 変換時間の合計は 134+128+128+128＝518 ステート（max）になります。TK-3052 の CPU クロックは 25MHz なので，

$$\frac{518}{25\times10^{6}(Hz)} = 20.72\times10^{-6}(s) = 20.72(\mu s)$$

です。この時間より短い間隔でサンプリングすることはできません。また，割り込みを受け付けたり，AD 値を読み込んだり，AD 変換スタートを設定したり，ということを考えると，ぎりぎりの値というよりも，ある程度の余裕が必要になります。

別の視点で考えてみましょう。1 チャンネルあたりのサンプル数は 256 個です。つまり，グラフの横軸方向は 256 個に分割されます。市販のオシロスコープの横軸方向は，大抵 10 目盛になっています。今回もそれを採用して 1 目盛 25 サンプル＝25 サンプル/Div ということにします。

さて，市販のオシロスコープの Time/Div の数字を見てみると，1，2，5 という数字の繰り返しになっていることに気付きます（ちなみに，1，2，5，10，20，50…は，対数目盛では等間隔に近くなります）。こちらも採用します。

ではここで，例として 1ms/Div のときのサンプリング周期と ITU にセットする値を計算してみましょう。1ms/Div が 25 サンプル/Div に対応しますので，サンプリング周期は，

$$\frac{1\times10^{-3}(s)}{25} = 40\times10^{-6}(s) = 40(\mu s)$$

です。ITU は CPU クロックの 1/1，1/2，1/4，1/8 のいずれかでカウントするよう指定します。CPU クロック＝25MHz の 1/1 でカウントする場合のカウント値は，

$$\frac{40\times10^{-6}(s)}{\dfrac{1}{25\times10^{6}(Hz)}} = 40\times10^{-6}\times25\times10^{6} = 1000$$

になります。同じように計算して，Time/Div によってサンプリング周期と ITU にセットするカウント値がどのようになるか表にまとめてみました。

Time/Div	サンプリング周期	ITU のカウント値（ジェネラルレジスタにセット）			
		プリスケーラ=Φ	プリスケーラ=Φ/2	プリスケーラ=Φ/4	プリスケーラ=Φ/8
0.1ms	4μs	100	50	25	12.5
0.2ms	8μs	200	100	50	25
0.5ms	20μs	500	250	125	62.5
1ms	40μs	1000	500	250	125
2ms	80μs	2000	1000	500	250
5ms	200μs	5000	2500	1250	625
10ms	400μs	10000	5000	2500	1250
20ms	800μs	20000	10000	5000	2500
50ms	2000μs=2ms	50000	25000	12500	6250
100ms	4000μs=4ms	×	50000	25000	12500
200ms	8000μs=8ms	×	×	50000	25000
500ms	20000μs=20ms	×	×	×	62500
1000ms	40000μs=40ms	×	×	×	×

赤でマーキングしている欄は設定できない条件です。サンプリング周期は 20.72μs 以上必要なので，それ以下は除外します。また，カウント値は 16 ビットなので，カウント値が 0000h～FFFFh（10 進で 0～65535）の範囲に収まらないものも除外します。

黄色でマーキングしている欄は，カウント値が割り切れないため誤差が出る条件です。

それで，青でマーキングしている欄の条件を採用します。

では，プログラムを見てみましょう。まず，Time/Div の設定は変数 TimeDiv に次のようにストアされています。

```
unsigned char   TimeDiv;        //Time/Div
                                // 0:  1ms/div  1:  2ms/div  2:  5ms/div
                                // 3: 10ms/div  4: 20ms/div  5: 50ms/div
                                // 6:100ms/div  7:200ms/div  8:500ms/div
```

データ要求コマンドを受信し，サンプリングをスタートする際に，ITU チャネル 0 を上の表の条件でイニシャライズしてカウントを始めます。

```
/************************************************************************
     ITU チャネル0 イニシャライズ（タイマスタート）
************************************************************************/
void init_itu0(void)
{

     ITU.TSTR.BIT.STR0     =     0;          //TCNT0 停止
     switch (TimeDiv){
          case 0:          //  1ms/Div
          case 1:          //  2ms/Div
          case 2:          //  5ms/Div
          case 3:          // 10ms/Div
          case 4:          // 20ms/Div
          case 5:          // 50ms/Div
               ITU0.TCR.BYTE    =     _00100000B;      //GRAのコンペアマッチクリア，φ/1
               break;
          case 6:          //100ms/Div
               ITU0.TCR.BYTE    =     _00100001B;      //GRAのコンペアマッチクリア，φ/2
               break;
          case 7:          //200ms/Div
               ITU0.TCR.BYTE    =     _00100010B;      //GRAのコンペアマッチクリア，φ/4
               break;
          case 8:          //500ms/Div
               ITU0.TCR.BYTE    =     _00100011B;      //GRAのコンペアマッチクリア，φ/8
```

```
                break;
            default:
                ITU0.TCR.BYTE   =    _00100000B;     //GRAのコンペアマッチクリア,φ/1
                break;
    }
    ITU0.TIOR.BYTE  =    _10001000B;     //コンペアマッチによる出力禁止
    ITU0.TCNT       =    0x0000;         //TCNT0=0
    switch (TimeDiv){
        case 0:         //  1ms/Div
                ITU0.GRA   =    1000;     //フェーズの周期(40us)
                break;
        case 1:         //  2ms/Div
                ITU0.GRA   =    2000;     //フェーズの周期(80us)
                break;
        case 2:         //  5ms/Div
                ITU0.GRA   =    5000;     //フェーズの周期(200us)
                break;
        case 3:         // 10ms/Div
                ITU0.GRA   =    10000;    //フェーズの周期(400us)
                break;
        case 4:         // 20ms/Div
                ITU0.GRA   =    20000;    //フェーズの周期(800us)
                break;
        case 5:         // 50ms/Div
                ITU0.GRA   =    50000;    //フェーズの周期(2000us=2ms)
                break;
        case 6:         //100ms/Div
                ITU0.GRA   =    50000;    //フェーズの周期(4000us=4ms)
                break;
        case 7:         //200ms/Div
                ITU0.GRA   =    50000;    //フェーズの周期(8000us=8ms)
                break;
        case 8:         //500ms/Div
                ITU0.GRA   =    62500;    //フェーズの周期(20000us=20ms)
                break;
        default:
                ITU0.GRA   =    1000;     //フェーズの周期(40us)
                break;
    }
    ITU0.TSR.BYTE   =    _11111000B;     //タイマステータスレジスタクリア
    ITU0.TIER.BYTE  =    _11111001B;     //コンペアマッチA割込みイネーブル
    ITU.TSTR.BIT.STR0 = 1;               //TCNT0 カウントスタート
}
```

ITU が動き出したら, ITU 割込みで AD 変換をスタート, AD 変換終了割込みで AD 値を読み込みます。そして, 256 個のサンプリングが終了したら, パソコンにデータを送信します。

以上で, TK-3052 のプログラムの説明は終わりです。その他の部分についてはソースリストをご覧ください。

7. パソコンのプログラム

パソコンのプログラムは VisualBasic6（以降 VB6 と表記）で作りました。VB6 をお持ちの方はプロジェクト（DsoTiny. vbp）を開くと詳細を見ることができます。プログラムの詳細はソースリストをご覧ください。各コンポーネントのイベント発生時にどのルーチンがコールされるか追いかけてみると理解が進むと思います。

なお，このプログラムは標準ライブラリのほかに「MSCOMM32. OCX」が必要です。VB6 をインストールしてあれば問題ありませんが，そうでないときは別途入手する必要があります。詳しくは弊社 CD の「_始めにお読みください」フォルダ内の「VB プログラムについて. pdf」をご覧ください。

■ スタートボタン

スタートボタンは，測定開始前は「スタート」を表示しクリックされたら測定開始，測定中は「ストップ」を表示しクリックされたら測定を中止します。

```
'-------------------------------------------------
'   スタートボタンをクリック
'-------------------------------------------------
Private Sub cmdStartButton_Click()
    If blnCommPort = False Then '待機中
        If CommPort_Open = False Then   '通信ポートオープン
            Exit Sub    'オープンできなかったときはサブルーチンを抜ける
        End If
        blnCommPort = True  '通信中にする
        cboCommPort.Enabled = False     '通信ポートの各設定を変更不可にする
        cboBaudRate.Enabled = False
        cboDataLength.Enabled = False
        cboStopBit.Enabled = False
        cboParity.Enabled = False
        cmdPrintButton.Enabled = False  '印刷ボタンクリック不可にする
        cmdStartButton.Caption = "ストップ" 'スタートボタンの表示をストップに変更する
        If chkSingleTrigger.Value = 0 Then  'メッセージの表示
            lblMessage.Caption = "計測中"
        Else
            lblMessage.Caption = "トリガ待ち"
        End If
        shpSingleTriggerIndication.FillColor = vbRed    'シングルトリガ待ちの表示
        Call Command_Send   '要求コマンドの送信
    Else                        '通信中
        MSComm1.Output = "/"    'マイコンボードにAD変換キャンセルのコマンドを送信
        blnCommPort = False '待機中にする
        Call CommPort_Close '通信ポートクローズ
        blnHeadderRcv = False   'ヘッダー未受信にする
        cboCommPort.Enabled = True  '通信ポートの各設定を変更可能に戻す
        cboBaudRate.Enabled = True
        cboDataLength.Enabled = True
        cboStopBit.Enabled = True
        cboParity.Enabled = True
        cmdPrintButton.Enabled = True   '印刷ボタンクリック可能に戻す
        lblMessage.Caption = ""         'メッセージの消去
        cmdStartButton.Caption = "スタート" 'スタートボタンの表示をスタートに戻す
        shpSingleTriggerIndication.FillColor = vbGreen  'シングルトリガ待ちから抜けたことを表示
    End If
End Sub
```

■　通信ポートのオープンとクローズ

測定開始の際にまず通信ポートを設定した条件でオープンします。オープンできない場合(指定できないポートを使おうとした，指定したポートがすでに使われている)，エラーを表示します。通信ポートの設定値は「DsoTiny.ini」に保存し，次回起動する際に保存した設定値で通信できるようにします。

測定終了時に通信ポートをクローズします。

```
'-------------------------------------------------
'   通信ポートのオープン
'-------------------------------------------------
Public Function CommPort_Open() As Boolean
    On Error GoTo ErrorHandler

    'コンボボックスから値を取得
    Call MSComm1_Set

    'COMポートオープン
    With MSComm1
        If .PortOpen = True Then .PortOpen = False
        .CommPort = intCommPort
        .Settings = strBaudRate + "," + strParity + "," + strDataLength + "," + strStopBit
        .RTSEnable = True
        .InBufferCount = 0
        .OutBufferCount = 0
        .PortOpen = True
    End With

    '設定値を'DsoTiny.ini'に書き込む
    Call IniWrite

    CommPort_Open = True
    Exit Function

ErrorHandler:
    If Err.Number = 8002 Then       '設定できないポートを使おうとした
        prompt = "使用できないポートを開こうとしました。" & Chr(13)
        prompt = prompt & "ポート番号を確認してください。"
        buttons = vbOKOnly + vbExclamation
        Title = "ポート番号の確認"
        ans = MsgBox(prompt, buttons, Title)
        CommPort_Open = False
    ElseIf Err.Number = 8005 Then   '開いているポートを使おうとした
        prompt = "使用中のポートを開こうとしました。" & Chr(13)
        prompt = prompt & "ポート番号を確認してください。"
        buttons = vbOKOnly + vbExclamation
        Title = "ポート番号の確認"
        ans = MsgBox(prompt, buttons, Title)
        CommPort_Open = False
    Else                            'その他のエラー
        prompt = "通信ポートを確認してください。"
        buttons = vbOKOnly + vbExclamation
        Title = "通信ポートの確認"
        ans = MsgBox(prompt, buttons, Title)
        CommPort_Open = False
    End If
End Function

'-------------------------------------------------
'   通信ポートのクローズ
'-------------------------------------------------
```

```
Public Sub CommPort_Close()
    If MSComm1.PortOpen = True Then
        MSComm1.PortOpen = False
    End If
End Sub
```

■　データ要求コマンドの送信について

文字列「SendData」に測定条件に応じてコマンド文字列を追記していきます。コマンド文字列が完成したらSendData を送信します。

```
'-------------------------------------------------
'   コマンド送信
'-------------------------------------------------
Public Sub Command_Send()
    Dim SendData As String

    intAdChRcvFlag = 0  'どのチャンネルをイネーブルにしたか（ADチャンネル受信フラグ）

    'ヘッダ
    SendData = "#"

    'チャンネル
    If chkCh1Enable.Value = 1 Then
        SendData = SendData + "1"
        intAdChRcvFlag = intAdChRcvFlag + 1
    Else
        SendData = SendData + "0"
    End If

    If chkCh2Enable.Value = 1 Then
        SendData = SendData + "1"
        intAdChRcvFlag = intAdChRcvFlag + 2
    Else
        SendData = SendData + "0"
    End If

    If chkCh3Enable.Value = 1 Then
        SendData = SendData + "1"
        intAdChRcvFlag = intAdChRcvFlag + 4
    Else
        SendData = SendData + "0"
    End If

    If chkCh4Enable.Value = 1 Then
        SendData = SendData + "1"
        intAdChRcvFlag = intAdChRcvFlag + 8
    Else
        SendData = SendData + "0"
    End If

    'カンマ
    SendData = SendData + ","

    'トリガチャンネル
    If optTriggerCh4.Value = True Then
        SendData = SendData + "4"
    ElseIf optTriggerCh3.Value = True Then
        SendData = SendData + "3"
```

```
    ElseIf optTriggerCh2.Value = True Then
        SendData = SendData + "2"
    Else
        SendData = SendData + "1"
    End If

    'オートトリガ or エッジトリガ
    If optAutoTrigger.Value = True Then
        SendData = SendData + "0"
    Else
        SendData = SendData + "1"
    End If

    '＋スロープ or －スロープ
    If optPlusEdge.Value = True Then
        SendData = SendData + "0"
    Else
        SendData = SendData + "1"
    End If

    'トリガレベル
    SendData = SendData + Right("0" + Hex(hsbTriggerLevel.Value) + "00", 4)

    'カンマ
    SendData = SendData + ","

    'Time/Div
    For i = 0 To 8
        If optTimeDiv(i).Value = True Then
            SendData = SendData + Right("0" + CStr(i), 2)
            Exit For
        End If
    Next i

    'デリミタ
    SendData = SendData + Chr(13)

    '送信
    MSComm1.Output = SendData
End Sub
```

■ データ応答コマンドの受信について

データを受信するとイベントが発生します。まず，どの通信ポートのイベントの何が発生したか判断し，受信イベントのときは「DataReceive」ルーチンをコールします。

```
'-------------------------------------------------
'   MSComm1イベント
'-------------------------------------------------
Private Sub MSComm1_OnComm()
    Select Case MSComm1.CommEvent
        Case comEvReceive   '受信イベント
            Call Data_Receive   'データ受信
    End Select
End Sub
```

指定したチャンネル全てのデータを受信したら波形を表示します。シングルトリガモードのときは「スタート」クリック待ちにし，そうでないときはデータ要求コマンドを送信します。

```
'-------------------------------------------------
'   データ受信
'-------------------------------------------------
Public Sub Data_Receive()
    Dim rcv As String

    rcv = MSComm1.Input     '受信データの取得

    If blnHeadderRcv = False Then   'ヘッダー未受信
        If rcv = "&" Then
            blnHeadderRcv = True    'ヘッダー受信/16ビットデータモード（将来の拡張用）
            intAdBit = 16
            strRcvData = rcv
        ElseIf rcv = "$" Then
            blnHeadderRcv = True    'ヘッダー受信/8ビットデータモード（現行）
            intAdBit = 8
            strRcvData = rcv
        End If
    Else                            'ヘッダー受信済み
        If Right(rcv, 1) = Chr(13) Then     'デリミタ受信？
            strRcvData = strRcvData + rcv
            Call Ad_Set(strRcvData) 'AD値の取得
            blnHeadderRcv = False
            If intAdChRcvFlag = 0 Then  '全チャンネル受信した
                picWave1.Cls
                picWave2.Cls
                picWave3.Cls
                picWave4.Cls
                Call Graph_Scale_Draw   '目盛の描画
                Call Wave_Draw          '波形の描画
                Call Trigger_Indication
                If chkSingleTrigger.Value = 1 Then  'シングルトリガモードのとき
                    blnCommPort = False             '通信待機中セット
                    Call CommPort_Close             '通信ポートのクローズ
                    cboCommPort.Enabled = True      '通信ポートの各設定を変更可能に戻す
                    cboBaudRate.Enabled = True
                    cboDataLength.Enabled = True
                    cboStopBit.Enabled = True
                    cboParity.Enabled = True
                    cmdPrintButton.Enabled = True   '印刷ボタンをクリック可能に戻す
                    lblMessage.Caption = ""         'メッセージの消去
                    shpSingleTriggerIndication.FillColor = vbGreen  'シングルトリガのインジケーションを緑にする
                    cmdStartButton.Caption = "スタート"
                Else
                    Call Command_Send   'シングルトリガモードでないとき，次のデータの送信要求
                End If
            End If
        Else
            strRcvData = strRcvData + rcv   '受信文字列追加
        End If
    End If
End Sub
```

■ 波形の描画について

波形を描画するPictureオブジェクトは4つ用意されています(picWave1, picWave2, picWave3, picWave4)。「1 画面表示」のときは「picWave1」を,「2 画面表示」のときは「picWave1」と「picWave2」を,「4 画面表示」のときは「picWave1」～「picWave4」を表示します(それぞれの「Visible」プロパティをTrueにする)。

波形はそれぞれのPictureオブジェクトにLineメソッドを使って描画します。「1画面表示」のときは「picWave1」に全てのチャンネルのAD値を,「2 画面表示」のときは「picWave1」にチャンネル1とチャンネル3のAD値,「picWave2」にチャンネル2とチャンネル4のAD値を,「4画面表示」のときは「picWave1」～「picWave4」にそれぞれチャンネル1～チャンネル4のAD値を表示します。

```
'-------------------------------------------------
'   波形の描画
'-------------------------------------------------
Public Sub Wave_Draw()
    If chkCh1Enable.Value = 1 Then  'Ch1を表示する
        picWave1.PSet (0, 255 - lngAdBuf(0, 0) / 256), vbRed
        For i = 1 To SamplingConst - 1
            picWave1.Line -(i, 255 - lngAdBuf(0, i) / 256), vbRed
        Next i
    End If

    If chkCh2Enable.Value = 1 Then  'Ch2を表示する
        If optDisplay1.Value = True Then    '1画面表示のとき
            picWave1.PSet (0, 255 - lngAdBuf(1, 0) / 256), vbBlue
            For i = 1 To SamplingConst - 1
                picWave1.Line -(i, 255 - lngAdBuf(1, i) / 256), vbBlue
            Next i
        Else                                '2画面表示と4画面表示のとき
            picWave2.PSet (0, 255 - lngAdBuf(1, 0) / 256), vbBlue
            For i = 1 To SamplingConst - 1
                picWave2.Line -(i, 255 - lngAdBuf(1, i) / 256), vbBlue
            Next i
        End If
    End If

    If chkCh3Enable.Value = 1 Then  'Ch3を表示する
        If optDisplay4.Value = True Then    '4画面表示のとき
            picWave3.PSet (0, 255 - lngAdBuf(2, 0) / 256), vbMagenta
            For i = 1 To SamplingConst - 1
                picWave3.Line -(i, 255 - lngAdBuf(2, i) / 256), vbMagenta
            Next i
        Else                                '1画面表示と2画面表示のとき
            picWave1.PSet (0, 255 - lngAdBuf(2, 0) / 256), vbMagenta
            For i = 1 To SamplingConst - 1
                picWave1.Line -(i, 255 - lngAdBuf(2, i) / 256), vbMagenta
            Next i
        End If
    End If

    If chkCh4Enable.Value = 1 Then  'Ch4を表示する
        If optDisplay4.Value = True Then        '4画面表示のとき
            picWave4.PSet (0, 255 - lngAdBuf(3, 0) / 256), vbGreen
            For i = 1 To SamplingConst - 1
                picWave4.Line -(i, 255 - lngAdBuf(3, i) / 256), vbGreen
            Next i
        ElseIf optDisplay2.Value = True Then    '2画面表示のとき
            picWave2.PSet (0, 255 - lngAdBuf(3, 0) / 256), vbGreen
            For i = 1 To SamplingConst - 1
                picWave2.Line -(i, 255 - lngAdBuf(3, i) / 256), vbGreen
            Next i
```

```
        Else                                    '1画面表示のとき
            picWave1.PSet (0, 255 - lngAdBuf(3, 0) / 256), vbGreen
            For i = 1 To SamplingConst - 1
                picWave1.Line -(i, 255 - lngAdBuf(3, i) / 256), vbGreen
            Next i
        End If
    End If
End Sub
```

■ 印刷機能について

印刷は，表示されている DSO の画面イメージを取り込んで，Picture オブジェクト（picPrintImage）に貼り付け，Picture オブジェクトの PaintPicture メソッドを使って印刷しています。この方法については Microsoft の WEB に詳しい資料があります。詳細については下記をご覧ください。

http://support.microsoft.com/kb/161299/ 「画面、フォーム、ウィンドウを取り込んで印刷する方法」

8. プログラムを改造する

今回用意した TK-3052 プログラムや VB プログラムをベースに機能を追加することで，もっと使いやすくすることができます。

■ トリガ位置の変更

今回のプログラムは，トリガ条件後の波形を取り込みます。しかし，トリガ条件前の波形が見られると，もっと便利になります。何かが起きた結果を観測するだけでなく，何かが起きる原因を観測できるようになるからです。

■ 波形取り込み後の時間軸の拡大，縮小，スライド機能

今回のプログラムはサンプリングデータ数は256個に固定しました。それにあわせてパソコンに表示するグラフの横軸も 256 点にしています。これを，メモリが許す限りサンプリングデータ数を増やし，波形取り込み後に Time/Div を変更して波形を時間方向に拡大・縮小したり，トリガから離れた場所の波形をスライドして見たりする機能を追加すると，かなり便利になります。

そのほかにもいろいろ考えられると思います。ぜひ挑戦してみてください。

付録

TK-3052 の組み立て

タイマ＆LED ディスプレイキット（B6092）の組み立て

ジョイント基板の部品表，回路図，実装例

デジタルストレージオシロスコープ追加版　ジョイント基板の部品表，回路図，実装例

TK-3052 回路図

USB-RS485 変換モジュール

クライアント①

クライアント②

ハンドアセンブルの方法

C 言語のセクションについて

Hterm 用「intprg. c」

ターミナルソフトによる Hterm 用モニタプログラムの使い方

Hterm 用モニタプログラムのコマンド一覧

TK-3052 の組み立て

CD の「TK-3052 ユーザーズマニュアル」第 1 章をご覧ください。

タイマ&LED ディスプレイキット(B6092)の組み立て

CD の「TK-3687/TK-3687mini Option タイマ&LED ディスプレイ」第 1 章をご覧ください。TK-3687mini 版で組み立てます。

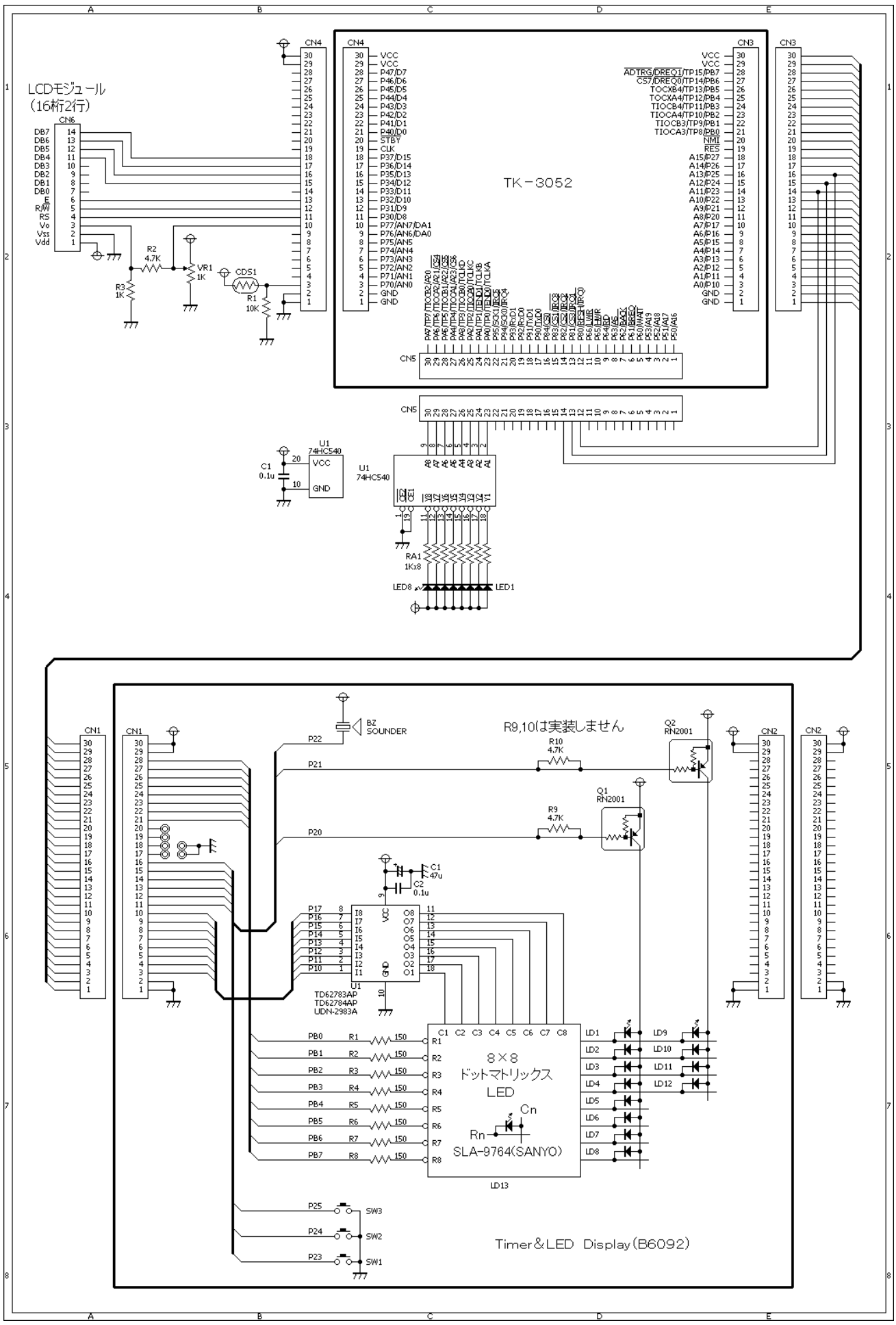
ジョイント基板の部品表, 回路図, 実装例

部品がそろっているかパーツリストと比較して確認してください。

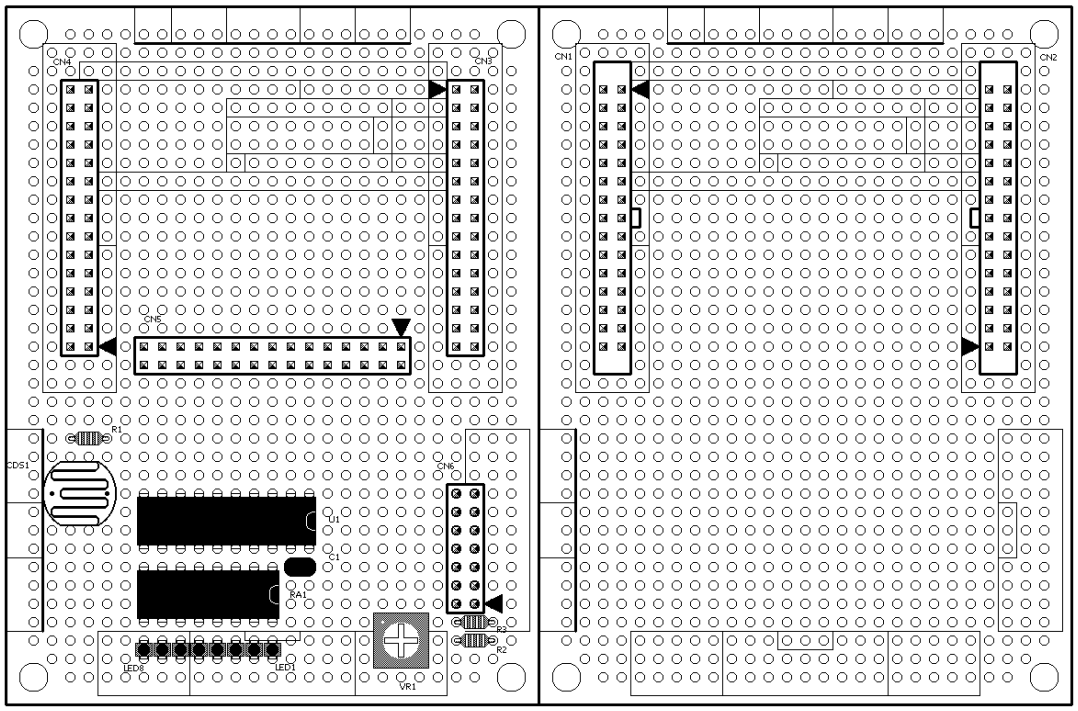
TK-3052 版マイコン事始めキット 部品表

部品番号	型名・規格	メーカ	数量	備考
●TK3052 接続用部品				
CN3,4,5	PS-30PE-D4T1-PN1	航空電子	3	相当品可
●タイマ&LED ディスプレイキット(B6092)接続用部品				
CN1,2	HIF3FB-30DA-2.54DSA	HRS	2	相当品可
●LED 関連部品				
LED1～8	HLMP/QLMP シリーズ	AVAGO	8	相当品可
U1	74HC540AP		1	20 ピン IC ソケット付属
RA1	898-3-R1K	BI	1	相当品可,16 ピン IC ソケット付属
C1	0.1μF		1	
●CDS 関連部品				
CDS1			1	
R1	10KΩ		1	
●LCD 関連部品				
LCD モジュール	SC1602*B	SUNLIKE	1	秋月電子 P-00040
CN6	(14 ピン,オスメス対)		1	LCD モジュール付属
VR1	1KΩ		1	
R2	4.7KΩ		1	
R3	1KΩ		1	
スペーサ	10～11mm		1	
ビス	M3		2	
●その他				
ユニバーサル基板	B6093	東洋リンクス	1	
ラッピングケーブル	2m		1	
メッキ線			0	ハンダ面結線用,ラッピングケーブルの被覆を剥いて流用,抵抗やコンデンサの切り取った足を使用。

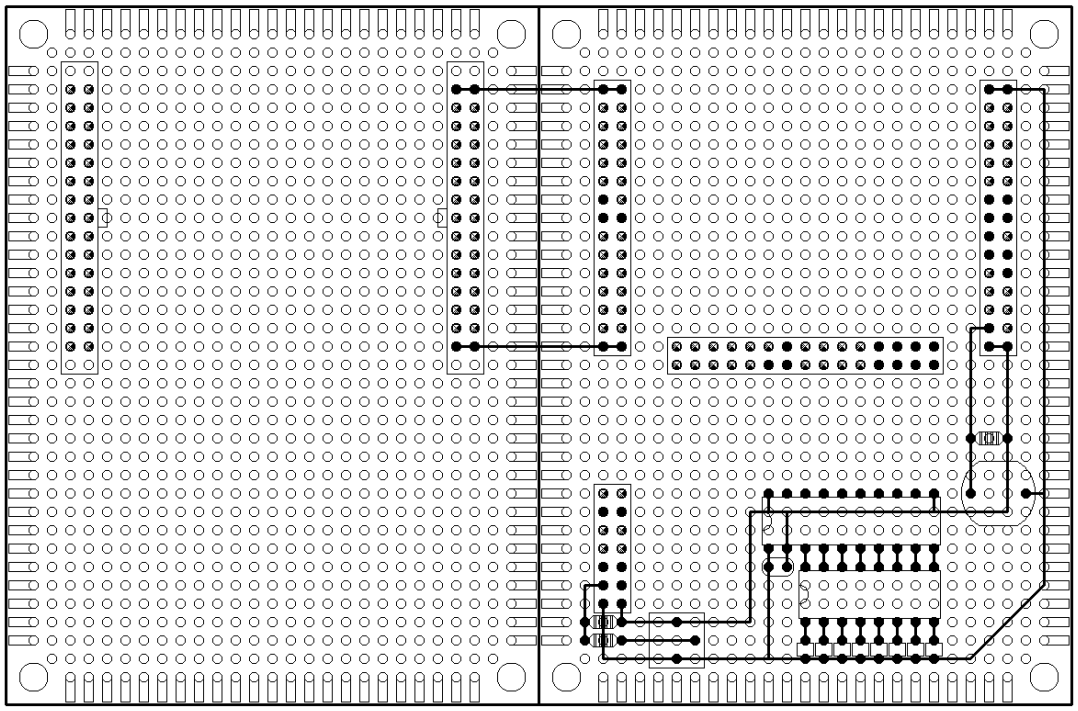
部品がそろっていたら, 回路図, 実装図, 写真を参考にして組み立ててください。

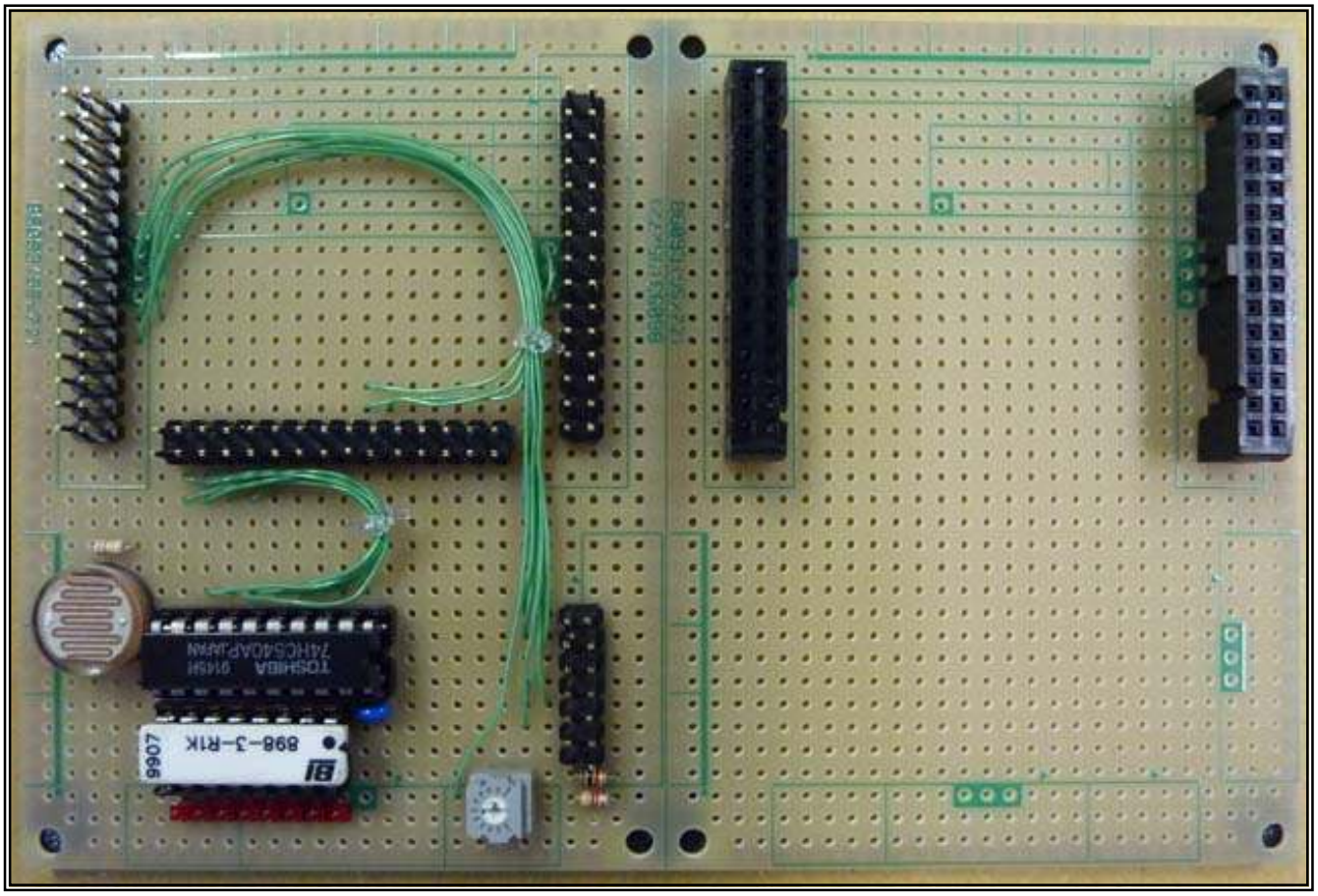

LCDモジュール
(16桁2行)
CN6
DB7
DB6
DB5
DB4
DB3
DB2
DB1
DB0
E
R/W
RS
Vo
Vss
Vdd
R2
4.7K
VR1
1K
R3
1K
CDS1
R1
10K
CN4
CN3
TK-3052
CN5
U1
74HC540
C1
0.1u
RA1
1Kx8
LED8
LED1
BZ
SOUNDER
R9,10は実装しません
R10
4.7K
R9
4.7K
Q1
RN2001
Q2
RN2001
C1
47u
C2
0.1u
CN1
CN2
P22
P21
P20
P17
P16
P15
P14
P13
P12
P11
P10
U1
TD62783AP
TD62784AP
UDN-2983A
PB0
PB1
PB2
PB3
PB4
PB5
PB6
PB7
R1
R2
R3
R4
R5
R6
R7
R8
150
8×8
ドットマトリックス
LED
Cn
Rn
SLA-9764(SANYO)
LD13
LD1
LD2
LD3
LD4
LD5
LD6
LD7
LD8
LD9
LD10
LD11
LD12
P25
P24
P23
SW3
SW2
SW1
Timer&LED Display(B6092)

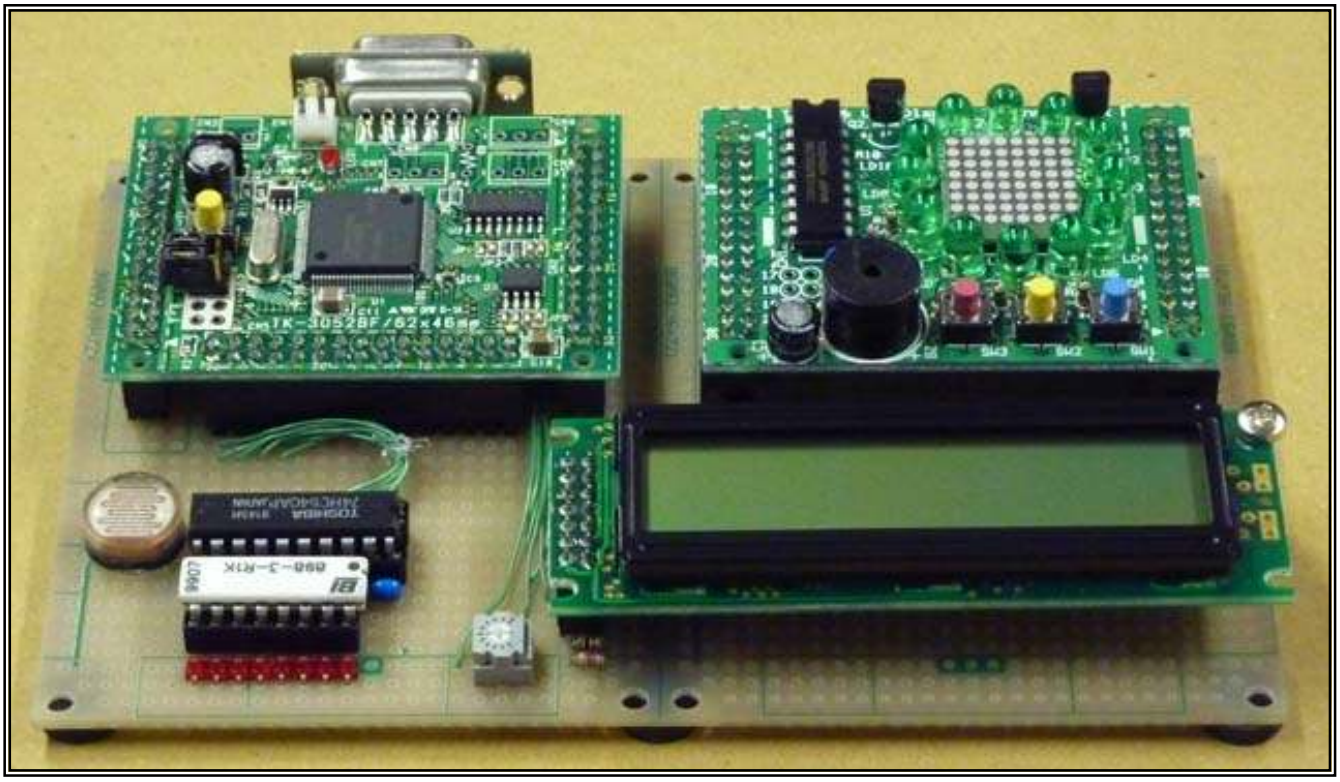
【表面】



【裏面】





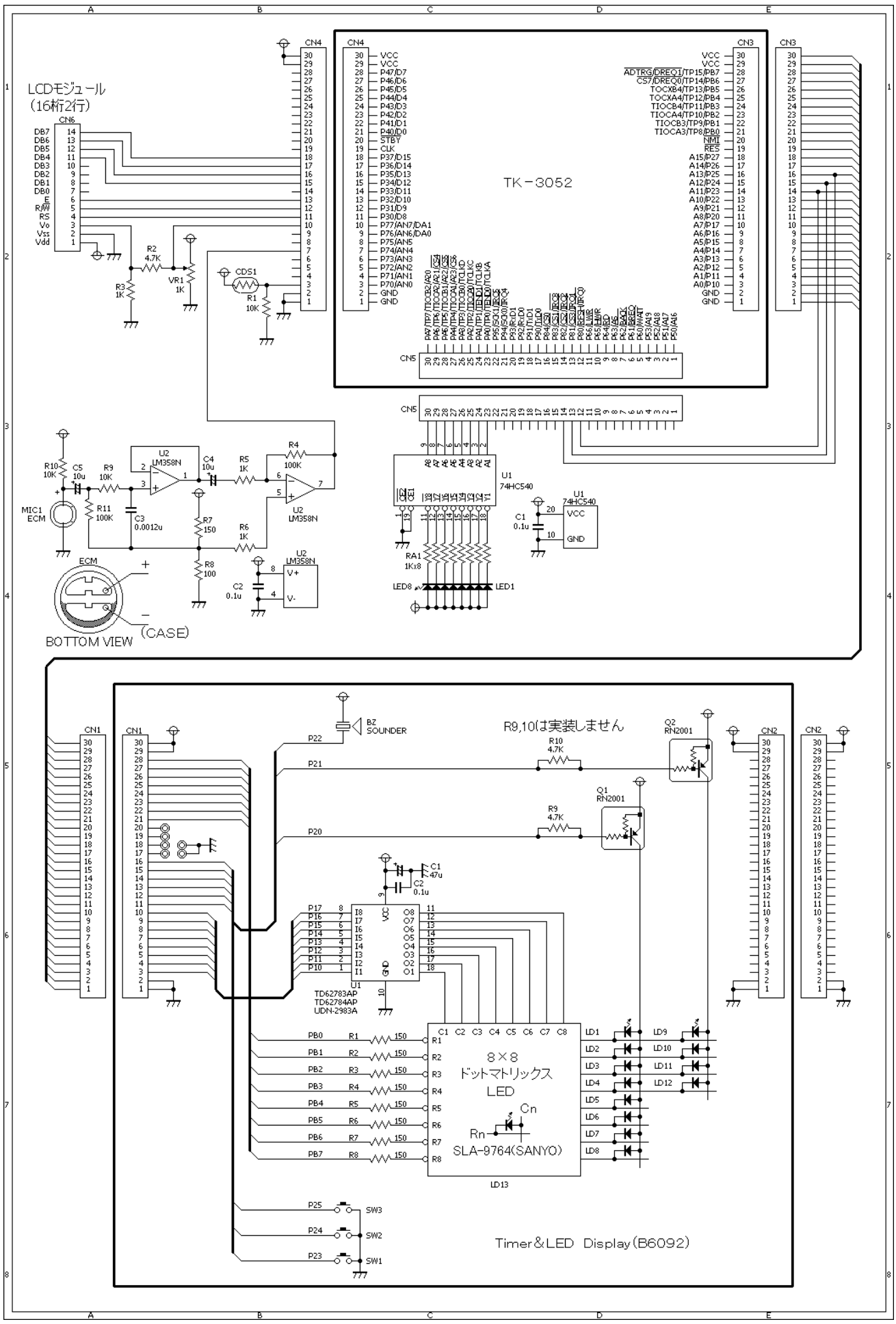


デジタルストレージオシロスコープ追加版　ジョイント基板の部品表，回路図，実装例

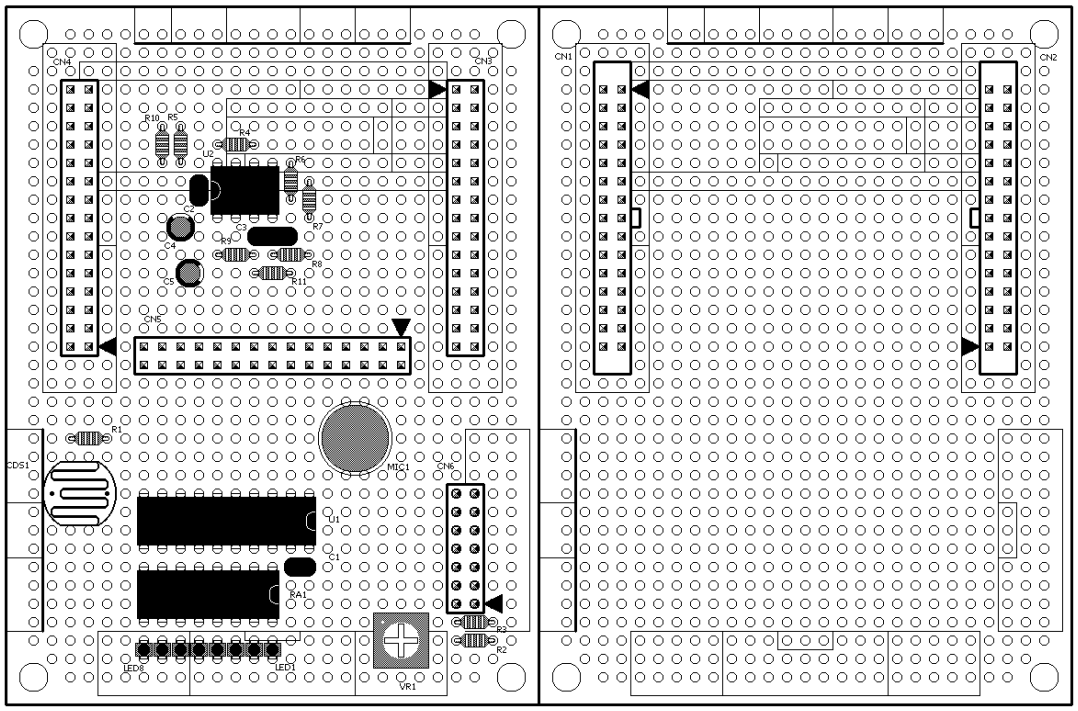
部品がそろっているかパーツリストと比較して確認してください。

デジタルストレージオシロスコープ追加　TK-3052 版マイコン事始めキット　部品表				
部品番号	**型名，規格**	**メーカー**	**数量**	**備考**
●TK-3052 接続用部品				
CN3，4，5	PS-30PE-D4T1-PN1	航空電子	3	相当品可
●タイマ&LED ディスプレイキット（B6092）接続用部品				
CN1，2	HIF3FB-30DA-2.54DSA	HRS	2	相当品可
●LED 関連部品				
LED1～8	HLMP/QLMP シリーズ	AVAGO	8	相当品可
U1	74HC540AP		1	20 ピン IC ソケット付属
RA1	898-3-R1K	BI	1	相当品可，16 ピン IC ソケット付属
C1	0．1μF		1	
●CDS 関連部品				
CDS1			1	
R1	10KΩ		1	
●LCD 関連部品				
LCD モジュール	SC1602BS*B	SUNLIKE	1	秋月電子 P-00040
CN6	（14 ピン，オスメス対）		1	LCD モジュール付属
VR1	1KΩ		1	
R2	4．7KΩ		1	
R3	1KΩ		1	
スペーサ	10～11 ㎜		1	
ビス	M3		2	
●デジタルストレージオシロスコープ関連部品				
U2	LM358N		1	相当品可，8 ピン IC ソケット付属
MIC1	C9767BB422LFP	DB Products	1	ECM，相当品可
R4，11	100KΩ		2	
R5，6	1KΩ		2	
R7	150Ω		1	
R8	100Ω		1	
R9，10	10KΩ		2	
C2	0．1μF（セラ）		1	
C3	0．0012μF（フィルム）		1	
C4，5	10μF/10V 以上（ケミ）		2	
●その他				
ユニバーサル基板	B6093	東洋リンクス	1	
ラッピングケーブル	3m		1	
メッキ線			0	ハンダ面結線用，ラッピングケーブルの被覆を剥いて流用，抵抗やコンデンサの切り取った足を使用

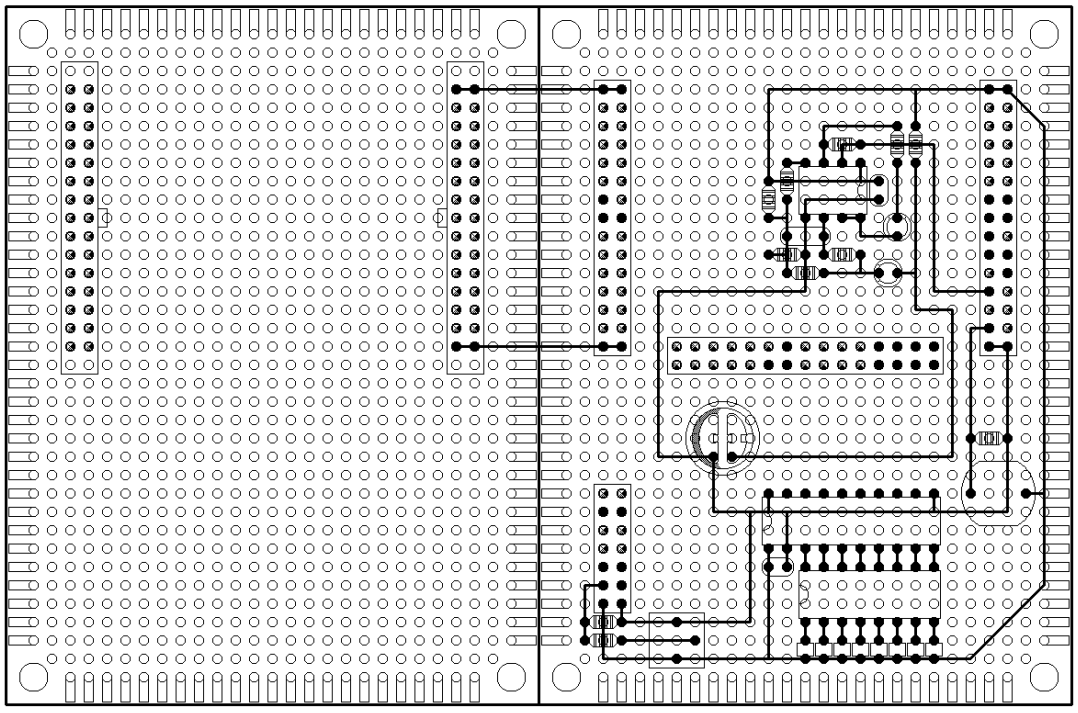
部品がそろっていたら，回路図，実装図，写真を参考にして組み立ててください。

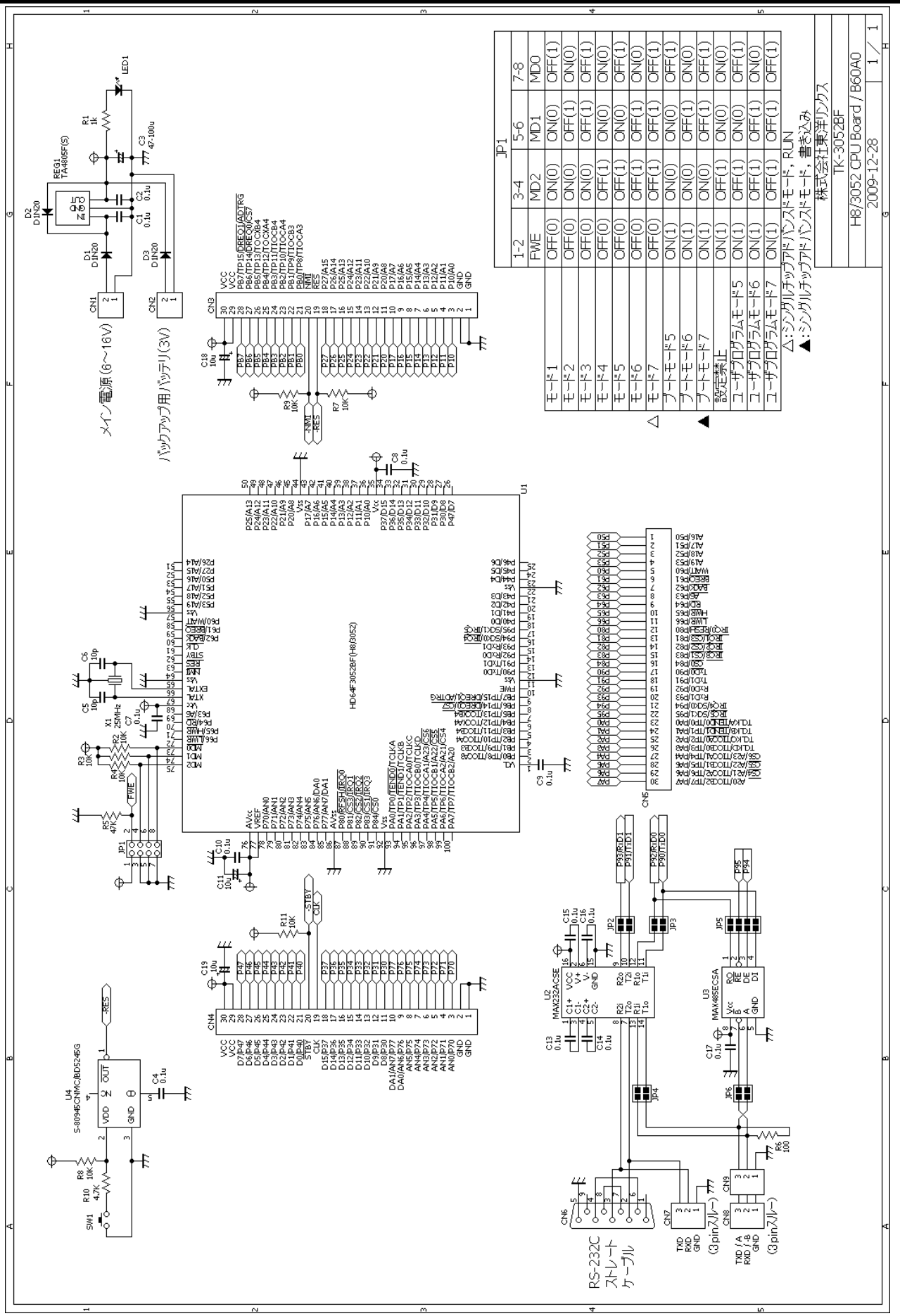

LCDモジュール
(16桁2行)
CN6
CN4
CN3
CN5
TK-3052
R2
4.7K
VR1
1K
R3
1K
CDS1
R1
10K
U2
LM358N
U1
74HC540
MIC1
ECM
BOTTOM VIEW
(CASE)
R9,10は実装しません
BZ
SOUNDER
Q1
RN2001
Q2
RN2001
TD62783AP
TD62784AP
UDN-2983A
8×8
ドットマトリックス
LED
SLA-9764(SANYO)
SW1
SW2
SW3
Timer&LED Display(B6092)

【表面】

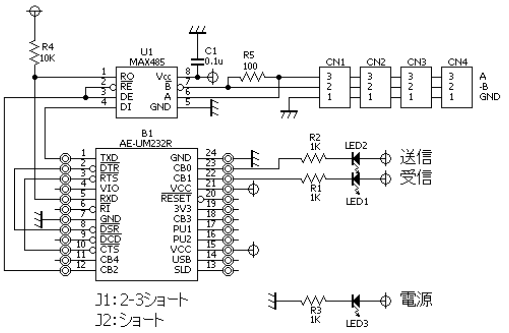

CN4
CN3
CN1
CN2
R10
R5
R4
U2
R8
C2
C3
R7
C4
R9
R6
C5
R11
CN5
R1
CDS1
MIC1
CN6
U1
C1
R41
R3
R2
LED6
LED1
VR1

【裏面】

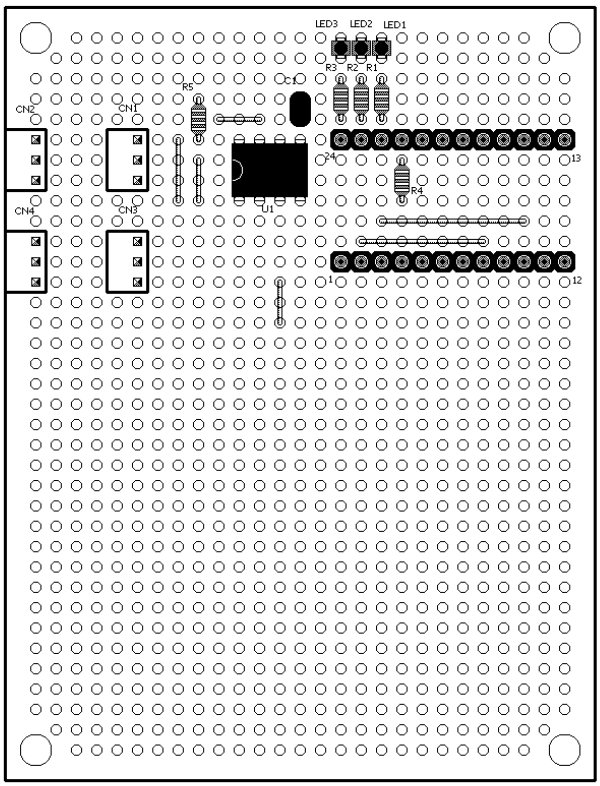


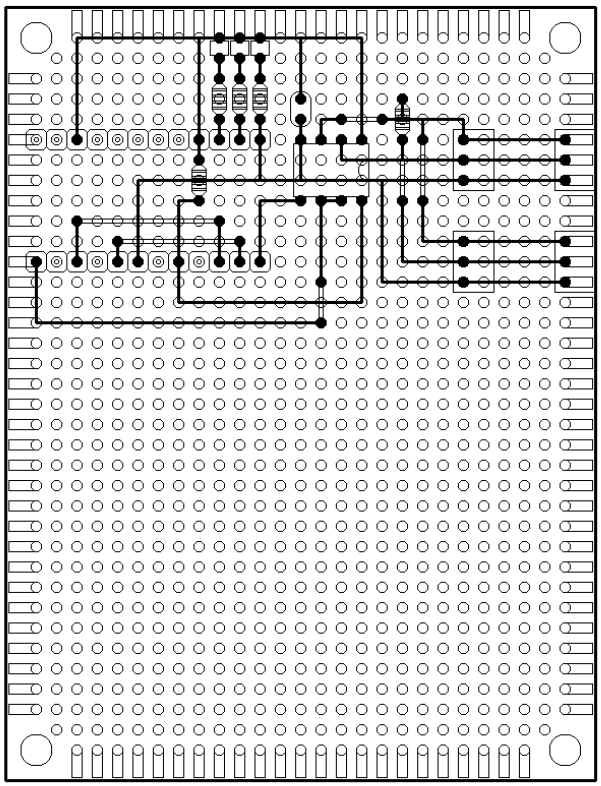

JP1
1-2 FWE
3-4 MD2
5-6 MD1
7-8 MD0
モード1 OFF(0) ON(0) ON(0) OFF(1)
モード2 OFF(0) ON(0) OFF(1) ON(0)
モード3 OFF(0) ON(0) OFF(1) OFF(1)
モード4 OFF(0) OFF(1) ON(0) ON(0)
モード5 OFF(0) OFF(1) ON(0) OFF(1)
モード6 OFF(0) OFF(1) OFF(1) ON(0)
モード7 OFF(0) OFF(1) OFF(1) OFF(1)
△ ブートモード5 ON(1) ON(0) ON(0) OFF(1)
ブートモード6 ON(1) ON(0) OFF(1) ON(0)
◀ ブートモード7 ON(1) ON(0) OFF(1) OFF(1)
設定禁止 ON(1) OFF(1) ON(0) ON(0)
ユーザプログラムモード5 ON(1) OFF(1) ON(0) OFF(1)
ユーザプログラムモード6 ON(1) OFF(1) OFF(1) ON(0)
ユーザプログラムモード7 ON(1) OFF(1) OFF(1) OFF(1)
△:シングルチップアドバンスドモード, RUN
▲:シングルチップアドバンスドモード, 書き込み
株式会社東洋リクス
TK-3052BF
H8/3052 CPU Board / B60A0
2009-12-28
1 / 1
メイン電源(6~16V)
バックアップ用バッテリ(3V)
RS-232C ストレートケーブル
HD64F3052BF(H8/3052)

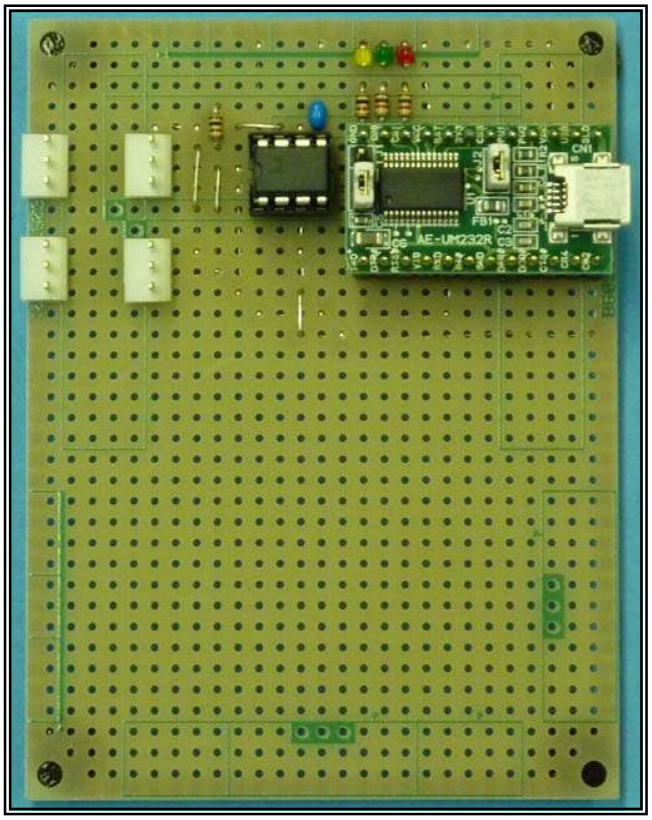
USB-RS485 変換モジュール

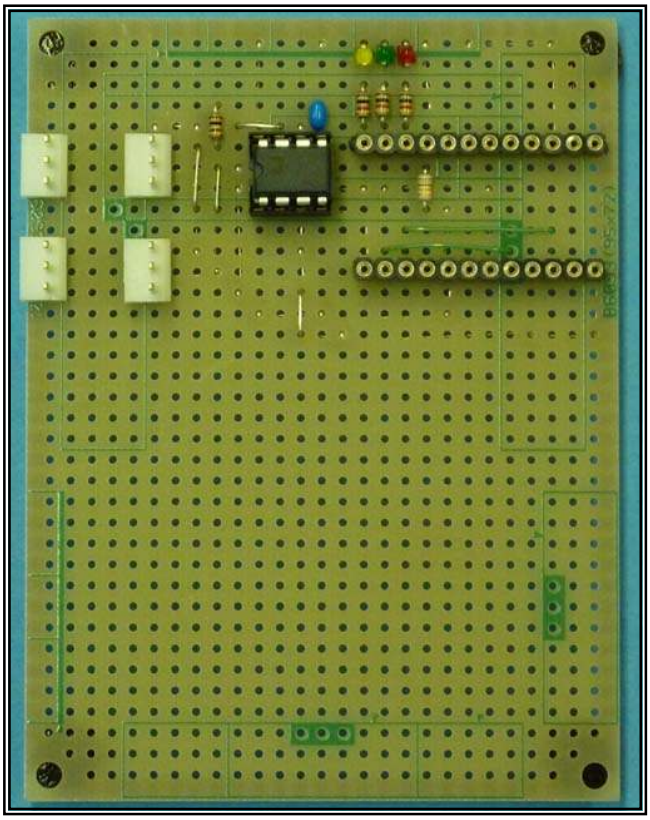


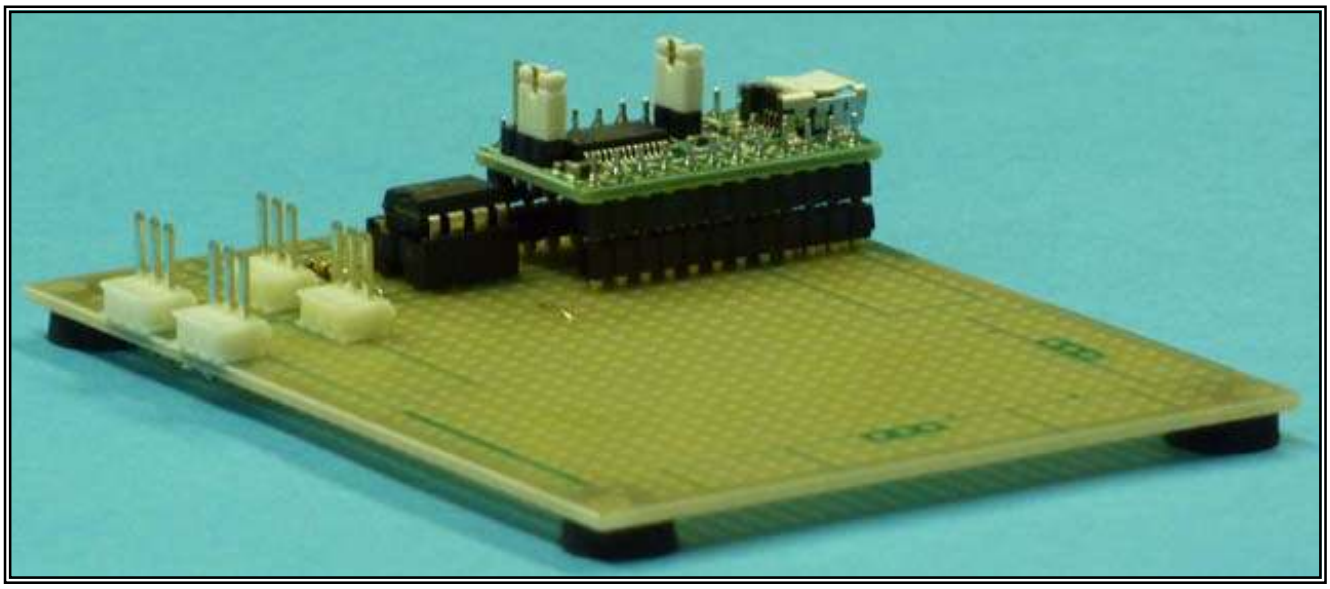
B1	AE-UM232R	秋月電子	1	ICソケット付き
U1	MAX485	MAXIM	1	相当品可，ICソケット付き
R1,2,3	1KΩ		3	LEDの明るさで調整する
R4	10KΩ		1	
R5	100Ω		1	
C1	0.1μF		1	
LED1	HLMPシリーズ	AVAGO	1	赤(受信)，相当品可
LED2	HLMPシリーズ	AVAGO	1	緑(送信)，相当品可
LED3	HLMPシリーズ	AVAGO	1	黄(電源)，相当品可
CN1,2,3,4	B3P-SHF-1AA	JST	4	
PCB	B6093	東洋リンクス	1	
ケーブル	50cm		1	ラッピング用，配線に使用



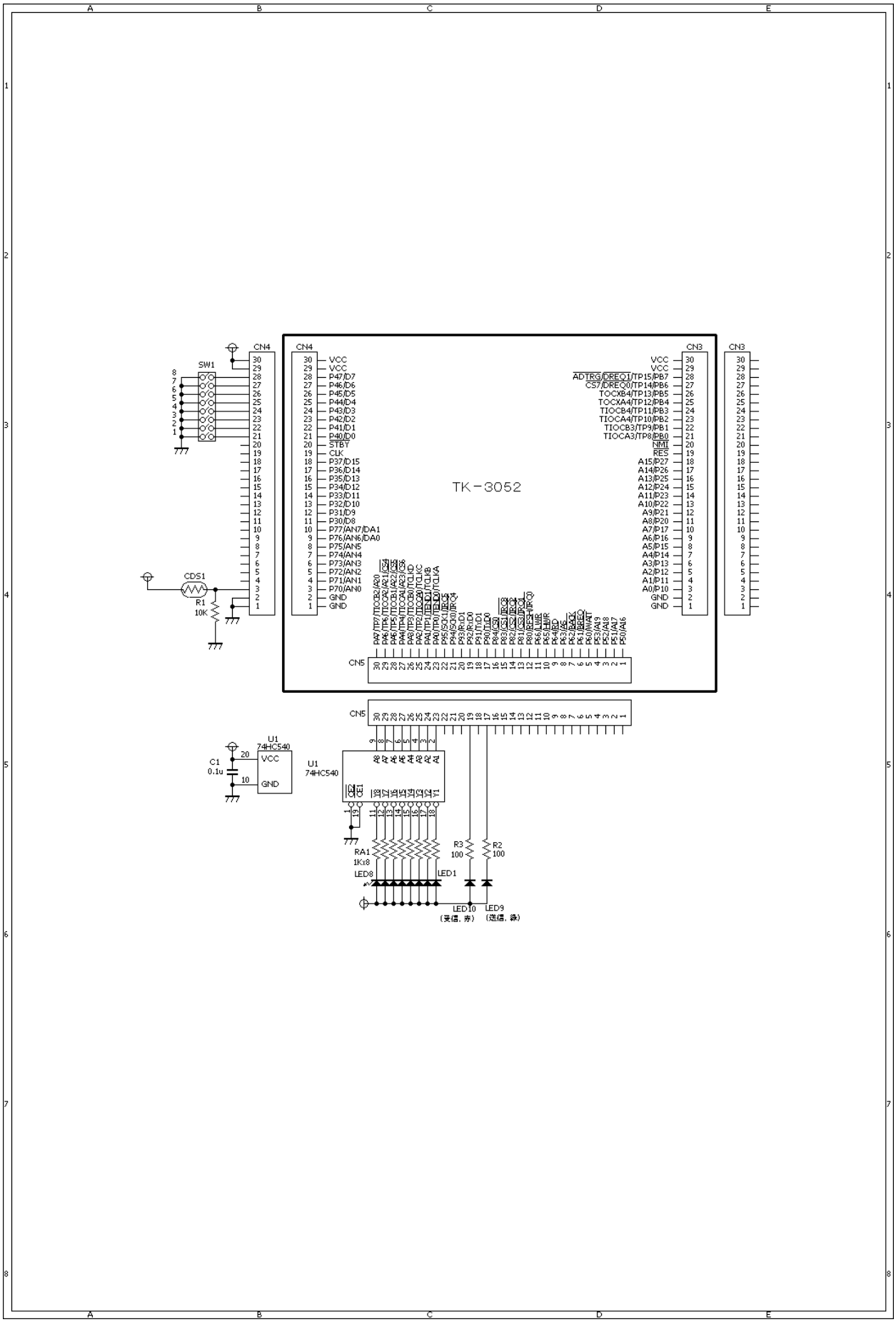


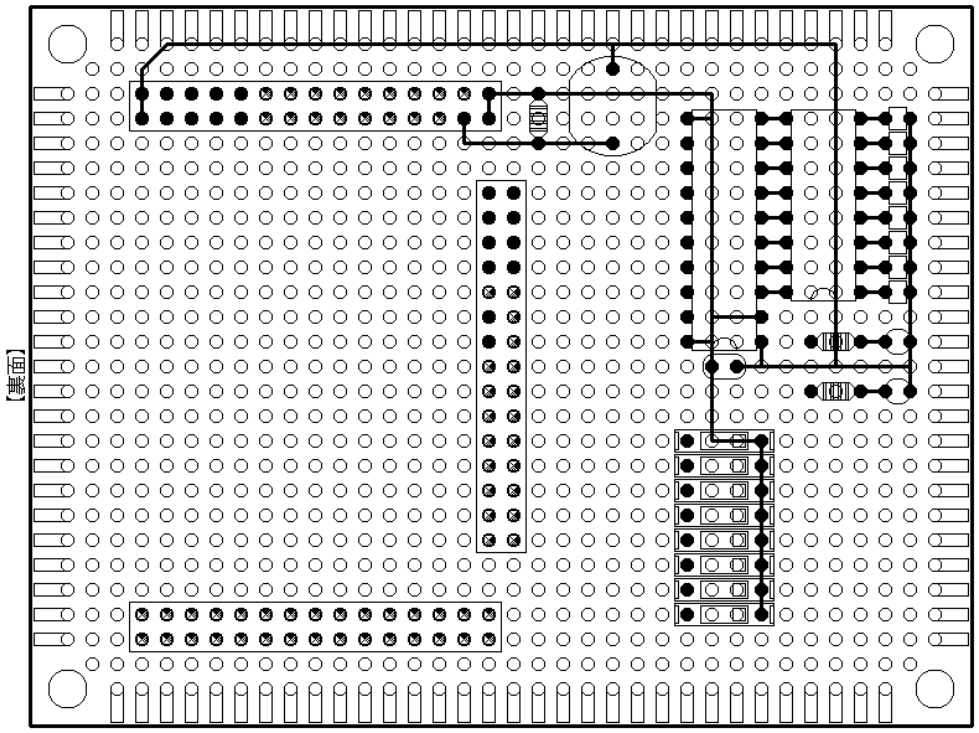

AE-UM232R

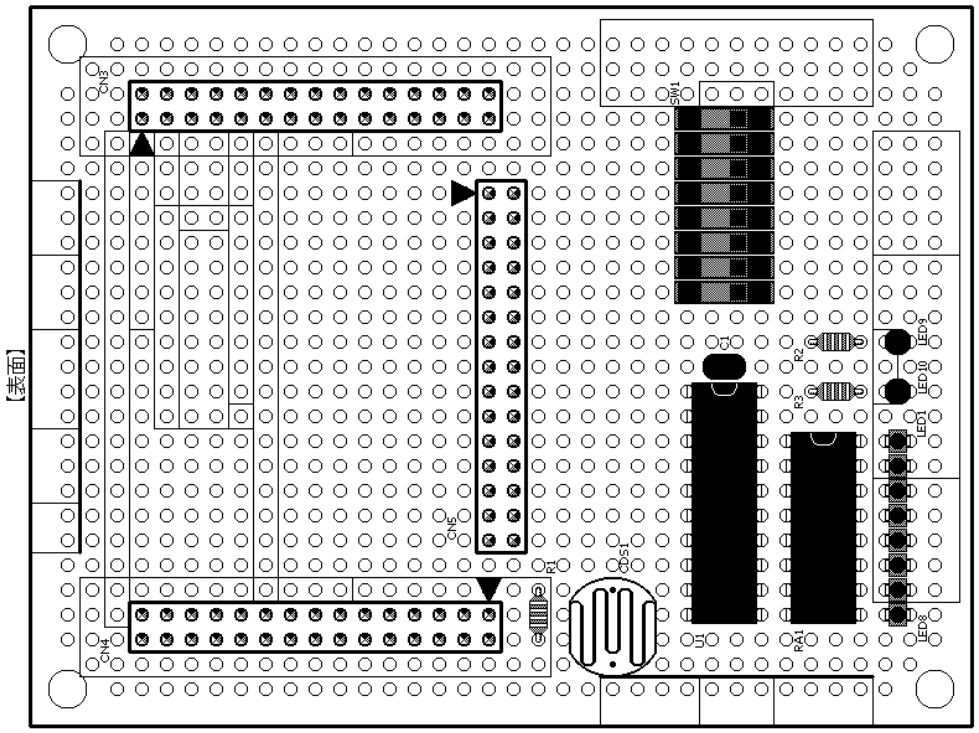


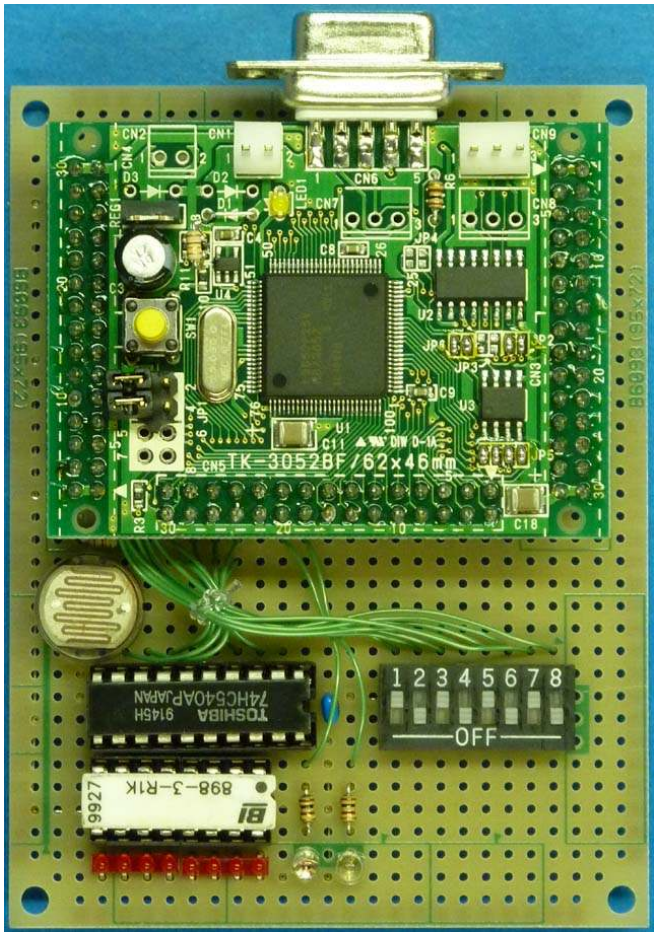


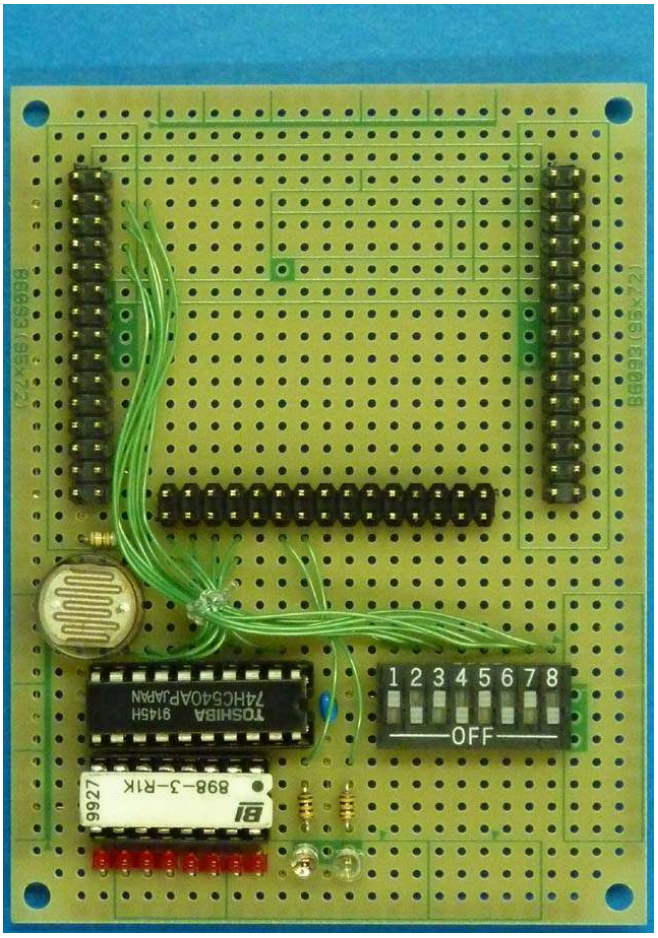
クライアント ①


TK-3052
CN4
CN3
CN5
SW1
CDS1
R1
10K
C1
0.1u
U1
74HC540
RA1
1Kx8
R3
100
R2
100
LED8
LED1
LED10
(受信, 赤)
LED9
(送信, 緑)


【裏面】

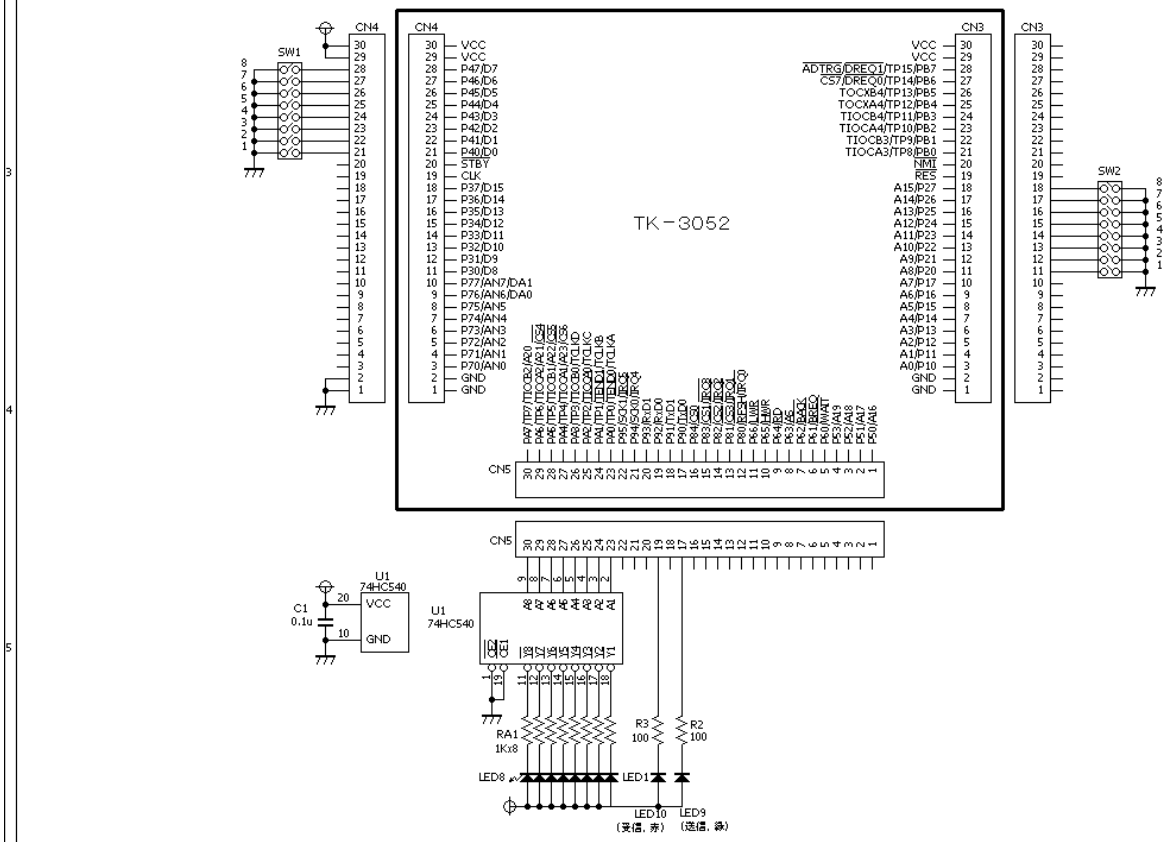

【表面】
CN3
SW1
C1
R2
R3
LED9
LED10
LED1
CN5
R1
OSC1
U1
RA1
LED8
CN4

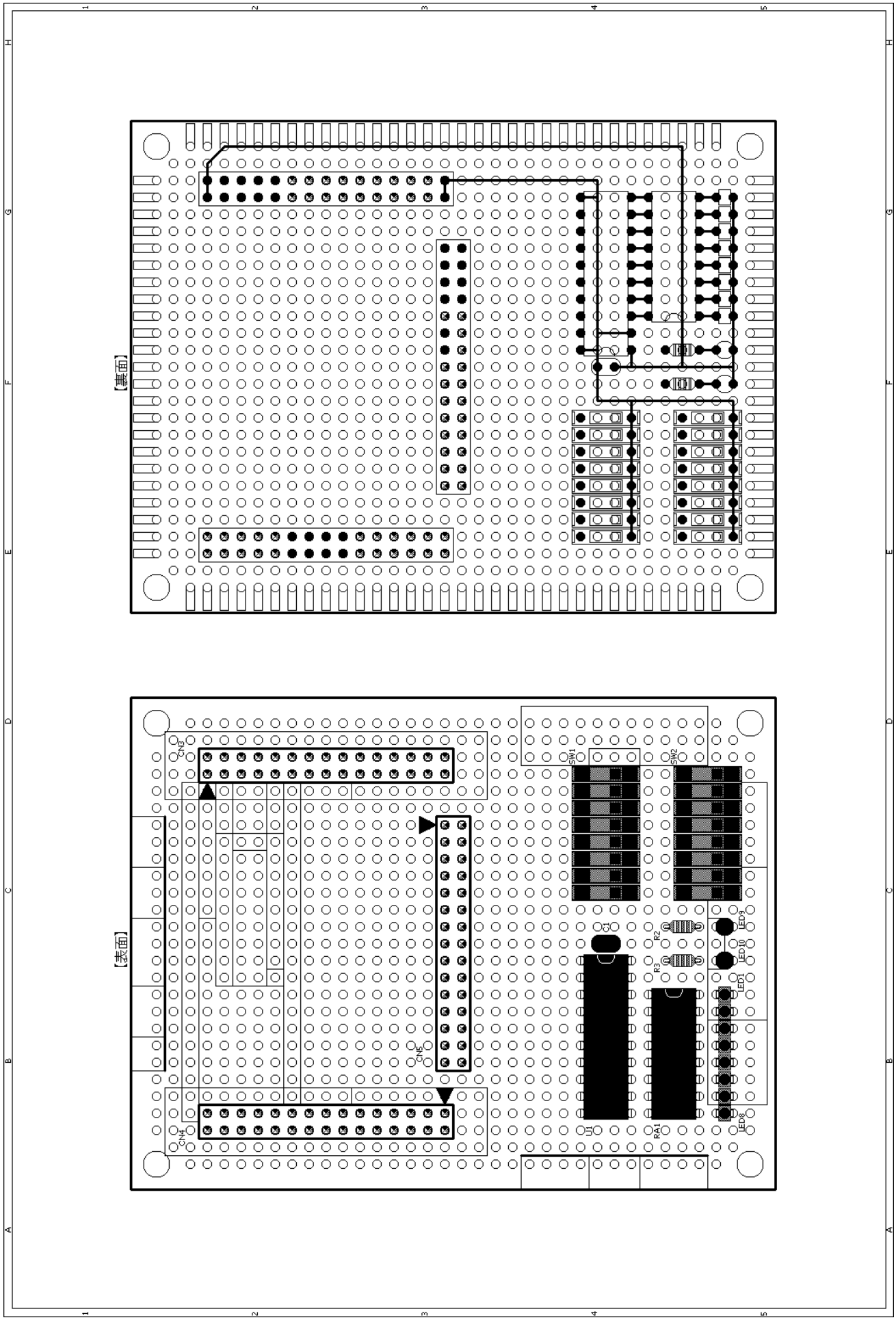


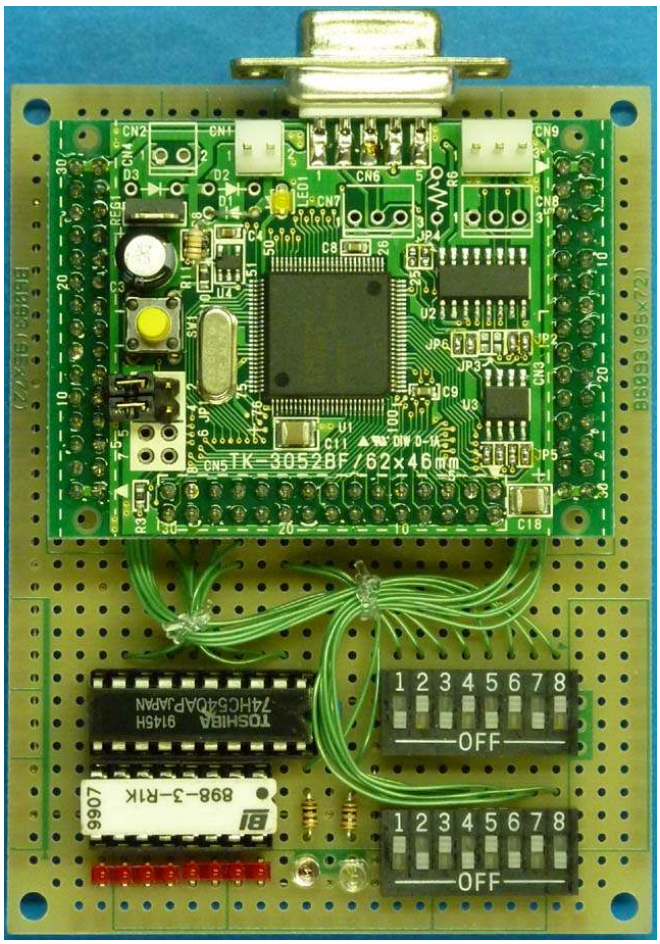


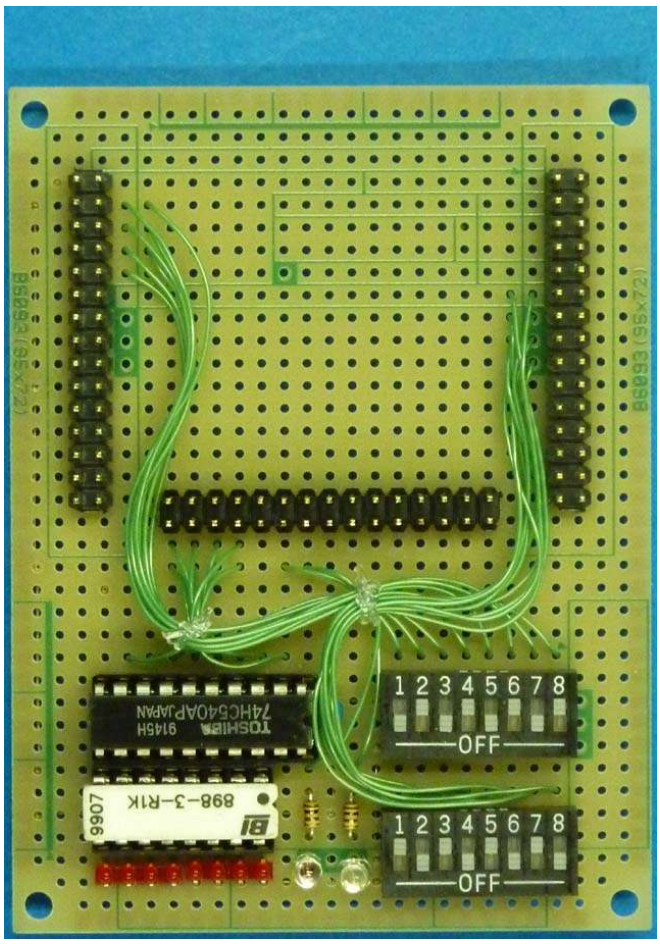
クライアント　①　部品表				
部品番号	型名・規格	メーカ	数量	備考
CN3,4,5	PS-30PE-D4T1-PN1	航空電子	3	相当品可
LED1～8	HLMP/QLMP シリーズ	AVAGO	8	相当品可
LED9	（緑）		1	
LED10	（赤）		1	
U1	74HC540AP		1	20 ピン IC ソケット付属
RA1	898-3-R1K	BI	1	相当品可,16 ピン IC ソケット付属
C1	0.1μF		1	
CDS1			1	
R1	10KΩ		1	
R2,3	100Ω		2	
SW1	DSS108	FUJISOKU	1	相当品可

クライアント ②


CN4
SW1
CN3
SW2
TK-3052
CN5
U1
74HC540
C1
0.1u
RA1
1Kx8
R3
100
R2
100
LED8
LED1
LED10
(受信, 赤)
LED9
(送信, 緑)


【裏面】
【表面】
CN3
CN4
CN5
SW1
SW2
C1
R2
R3
LED1
LED9
LED10
LED8
U1
RA1





クライアント　②　部品表				
部品番号	型名・規格	メーカ	数量	備考
CN3,4,5	PS-3OPE-D4T1-PN1	航空電子	3	相当品可
LED1～8	HLMP/QLMP シリーズ	AVAGO	8	相当品可
LED9	(緑)		1	
LED10	(赤)		1	
U1	74HC540AP		1	20 ピン IC ソケット付属
RA1	898-3-R1K	BI	1	相当品可,16 ピン IC ソケット付属
C1	0.1μF		1	
R2,3	100Ω		2	
SW1,2	DSS108	FUJISOKU	2	相当品可

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 2 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
FE800	F8	_main:	MOV.B	#H'FF,R0L	ポートAのイニシャライズ(出力に設定)
FE801	FF				
FE802	38		MOV.B	R0L,@H'FFD1	
FE803	D1				
FE804	7F	LOOP:	BSET	#0,@H'FFD3	LEDオン(PA0=1)
FE805	D3				
FE806	70				
FE807	00				
FE808	7F		BCLR	#0,@H'FFD3	LEDオフ(PA0=0)
FE809	D3				
FE80A	72				
FE80B	00				
FE80C	40		BRA	LOOP	LOOPにジャンプ
FE80D	F6				
FE80E					
FE80F					

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'FF,R0L          ;ポートAのイニシャライズ（出力に設定）
```

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‘FF’です。

それで，‘MOV.B #H'FF,R0L’をマシン語に変換すると‘F8’‘FF’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'FFD1
```

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‘D1’です。

それで，‘MOV.B R0L,@H'FFD1’をマシン語に変換すると‘38’‘D1’になります。

それでは 3 番目の命令です。

```
BSET    #0,@H'FFD3          ;LEDオン  (PA0=0)
```

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‘D3’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'FFD3’をマシン語に変換すると‘7F’‘D3’‘70’‘00’になります。

4 番目の命令です。

```
BSET    #0,@H'FFD3          ;LEDオフ  (PA0=1)
```

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‘D3’です。

第 3 バイトはそのまま‘72’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'FFD3’をマシン語に変換すると‘7F’‘D3’‘72’‘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=FE800 だったとすると，‘F8’，次に‘FF’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'FF,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，‘FF’を取り出し終わったときには PC=FE802 になっている，という点です。つまり，‘MOV.B #H'FF,R0L’を CPU が実行する時には PC=FE802 になっています。

このことを頭において，PC=FE80C 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=FE80E 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 FE80C ではなく FE80E になります。それで，FE80E 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 FE804 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 FE80E，②ジャンプ先‘LOOP:’は FE804 番地なので，③FE804-FE80E=-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

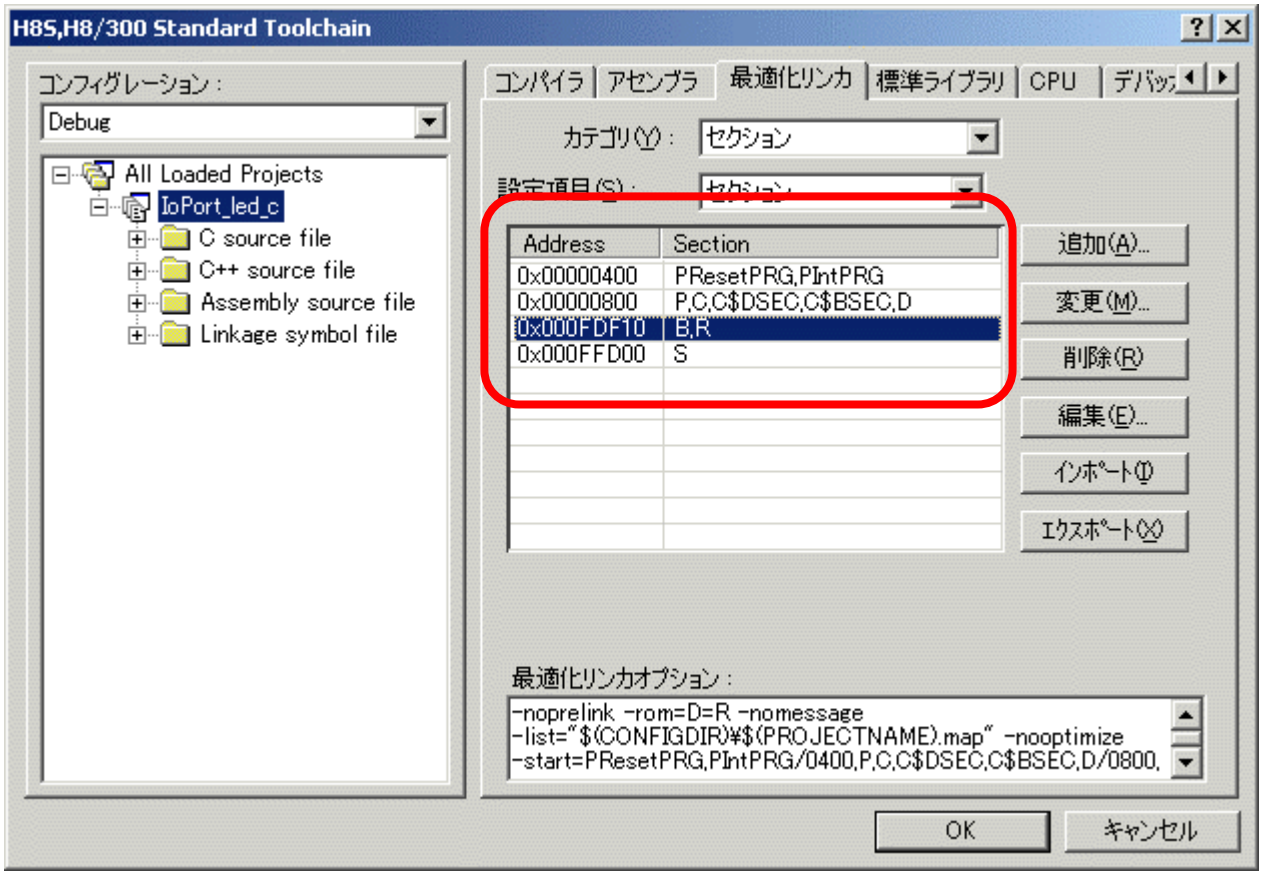
16 ビットデータのとき:		
10 進	2 進	16 進
-32768	1000000000000000	8000
-32767	1000000000000001	8001
∫	∫	∫
-2	1111111111111110	FFFE
-1	1111111111111111	FFFF
0	0000000000000000	0000
1	0000000000000001	0001
∫	∫	∫
32766	0111111111111110	7FFE
32767	0111111111111111	7FFF

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’

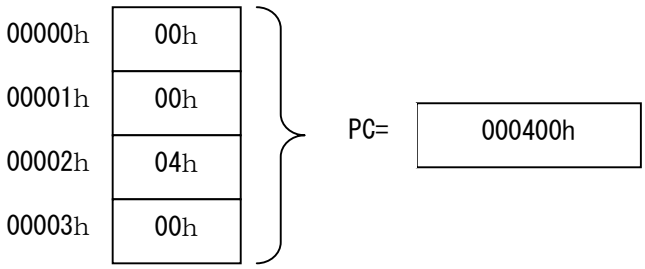
C 言語のセクションについて

HEW はメモリ空間をセクションに分割して，複数のソースファイルをリンクする際に，ベクタテーブル領域，プログラム領域，データ領域，スタック領域などの管理を行なっています。「Standard Toolchain」で確認すると，プロジェクト作成時のセクションは次のようになっています。



PResetPRG	：	リセットプログラム領域。（‘resetprg. c’）
PIntPRG	：	割り込みプログラム領域。（‘intprg. c’）
P	：	プログラム領域。（アプリケーションのソースファイル）
C	：	定数領域。
C$DSEC	：	初期化データセクションのアドレス領域。
C$BSEC	：	未初期化データセクションのアドレス領域。
D	：	初期化データ領域。（初期値）
B	：	未初期化データ領域。
R	：	初期化データ領域。（D がコピーされる）
S	：	スタック領域。（‘stacksct. h’）

H8/300H シリーズのマイコンの電源オンからのプログラムの起動の仕組みとソースファイルの関係です。電源オン，または RESET 端子に 20 システムクロック以上の Low パルスを加えてから High にすると，マイコンはベクタアドレス 0h～3h 番地のデータをリードしてプログラムカウンタにセットし，そこからプログラムの実行を開始します。上記の例だと，0h～03h には PResetPRG セクションの始まるアドレス‘00000400h’がセットされているので，そのうち下位 24 ビットの‘000400h’がプログラムカウンタ（PC）にセットされます。



ResetPRG セクションの内容は‘resetprg. c’で記述しています(HEW が自動生成)。抜粋すると,

```
#pragma section ResetPRG
__entry(vect=0) void PowerON_Reset(void)
{
      set_imask_ccr((_UBYTE)1);
     _INITSCT();
     set_imask_ccr((_UBYTE)0);
     main();
     sleep();
}
```

となっています。マシン語を見ると,

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
00400	7A		MOV.L	#H'000FFF00,ER7	ER7=SPの初期設定
00401	07				
00402	00				
00403	0F				
00404	FE				
00405	00				
00406	04		ORC.B	#H'80,CCR	割り込みをマスクする
00407	80				
00408	5E		JSR	@__INITSCT:24	
00409	00				
0040A	08				
0040B	02				
0040C	06		ANDC.B	#H'7F,CCR	割り込みのマスク解除
0040D	7F				
0040E	5E		JSR	@_main:24	mainルーチンをコール
0040F	00				
00410	08				
00411	00				
00412	01		SLEEP		
00413	80				
00414	54		RTS		
00415	70				

となり, すぐに‘main’ルーチンに処理を移しています。HEW が生成する‘main’ルーチンは次のとおりです(抜粋)。

```
void main(void)
{
}
```

マシン語を見てみると,

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
00800	54	_main	RTS		
00801	70				

P セクションの始まる 800h 番地から配置されています。このような手順を踏んで, プログラムは実行されていきます。

Hterm 用「intprg. c」

```
/************************************************************************/
/*                                                                      */
/*  FILE        :intprg.c                                               */
/*  DATE        :Wed, Jun 04, 2010                                      */
/*  DESCRIPTION :Interrupt Program                                      */
/*  CPU TYPE    :H8/3052F                                               */
/*                                                                      */
/*  This file is programed by TOYO-LINX Co.,Ltd. / yKikuchi             */
/*                                                                      */
/************************************************************************/

/*----------------------------------------------------------------------
このファイルはルネサスエレクトロニクスが配布しているモニタプログラム上で
RAMにダウンロードした割り込みプログラムを実行するために使用します。

通常はHEWが自動生成した「intprg.c」をそのまま使用すれば，割り込みプログラ
ムに分岐することができます。ベクタテーブルの領域に割り込みプログラムの先
頭アドレスが自動的に生成されるからです。

しかし，ベクタテーブルの領域はフラッシュメモリのため，モニタプログラムで
ダウンロードしても書き換えることはできません。その結果，割り込みプログラ
ムに分岐することができません。

この問題を解決するためにルネサスエレクトロニクスが配布しているモニタプロ
グラムでは，RAM上に擬似的なベクタテーブルの領域を割り当てることができます。
モニタプログラムは割り込みがかかると，この擬似的なベクタテーブルを参照し
て割り込みプログラムに分岐します。東洋リンクスがカスタマイズしたモニタプ
ログラムでは，FE000～FE0FF番地が擬似的なベクタテーブルの領域になります。

HEWが自動生成する「intprg.c」では「__interrupt(vect= )」擬似命令を使って
います。しかし，これを使用するとベクタテーブルは必ずフラッシュメモリに割
り付けられます。そこでこのファイルでは「__interrupt(vect= )」擬似命令を
使わずに配列として擬似的なベクタテーブルを作成します。

作業手順：
  ①HEWが自動生成した「intprg.c」を削除します。

  ②かわりに，このファイルをフォルダにコピーします。

  ③HEWの「Standard Toolchaine」ダイアログでセクション指定します。
    FE000番地に「CVECTBL」セクションを追加してください。

あとは，もとの「intprg.c」のときと同じように，割り込みプログラムを追加し
ます。このファイルの最後のほうに関数の実体があります。「sleep()」となっ
ているところに追加してください。
----------------------------------------------------------------------*/

#include    <machine.h>

typedef void (*fp)(void);

extern void PowerON_Reset(void);
```

```
#pragma interrupt(INT_NMI)
#pragma interrupt(INT_TRAP1)
#pragma interrupt(INT_TRAP2)
#pragma interrupt(INT_TRAP3)
#pragma interrupt(INT_TRAP4)
#pragma interrupt(INT_IRQ0)
#pragma interrupt(INT_IRQ1)
#pragma interrupt(INT_IRQ2)
#pragma interrupt(INT_IRQ3)
#pragma interrupt(INT_IRQ4)
#pragma interrupt(INT_IRQ5)
#pragma interrupt(INT_WOVI)
#pragma interrupt(INT_CMI)
#pragma interrupt(INT_IMIA0)
#pragma interrupt(INT_IMIB0)
#pragma interrupt(INT_OVI0)
#pragma interrupt(INT_IMIA1)
#pragma interrupt(INT_IMIB1)
#pragma interrupt(INT_OVI1)
#pragma interrupt(INT_IMIA2)
#pragma interrupt(INT_IMIB2)
#pragma interrupt(INT_OVI2)
#pragma interrupt(INT_IMIA3)
#pragma interrupt(INT_IMIB3)
#pragma interrupt(INT_OVI3)
#pragma interrupt(INT_IMIA4)
#pragma interrupt(INT_IMIB4)
#pragma interrupt(INT_OVI4)
#pragma interrupt(INT_DEND0A)
#pragma interrupt(INT_DEND0B)
#pragma interrupt(INT_DEND1A)
#pragma interrupt(INT_DEND1B)
#pragma interrupt(INT_ERI0)
#pragma interrupt(INT_RXI0)
#pragma interrupt(INT_TXI0)
#pragma interrupt(INT_TEI0)
#pragma interrupt(INT_ERI1)
#pragma interrupt(INT_RXI1)
#pragma interrupt(INT_TXI1)
#pragma interrupt(INT_TEI1)
#pragma interrupt(INT_ADI)

void INT_NMI(void);
void INT_TRAP1(void);
void INT_TRAP2(void);
void INT_TRAP3(void);
void INT_TRAP4(void);
void INT_IRQ0(void);
void INT_IRQ1(void);
void INT_IRQ2(void);
void INT_IRQ3(void);
void INT_IRQ4(void);
void INT_IRQ5(void);
void INT_WOVI(void);
```

```
void INT_CMI(void);
void INT_IMIA0(void);
void INT_IMIB0(void);
void INT_OVI0(void);
void INT_IMIA1(void);
void INT_IMIB1(void);
void INT_OVI1(void);
void INT_IMIA2(void);
void INT_IMIB2(void);
void INT_OVI2(void);
void INT_IMIA3(void);
void INT_IMIB3(void);
void INT_OVI3(void);
void INT_IMIA4(void);
void INT_IMIB4(void);
void INT_OVI4(void);
void INT_DEND0A(void);
void INT_DEND0B(void);
void INT_DEND1A(void);
void INT_DEND1B(void);
void INT_ERI0(void);
void INT_RXI0(void);
void INT_TXI0(void);
void INT_TEI0(void);
void INT_ERI1(void);
void INT_RXI1(void);
void INT_TXI1(void);
void INT_TEI1(void);
void INT_ADI(void);


#pragma section VECTBL

const fp VectorTable[] =
{
    PowerON_Reset,          //  vector  0 Reset
    (fp)0L,                 //  vector  1 Reserved
    (fp)0L,                 //  vector  2 Reserved
    (fp)0L,                 //  vector  3 Reserved
    (fp)0L,                 //  vector  4 Reserved
    (fp)0L,                 //  vector  5 Reserved
    (fp)0L,                 //  vector  6 Reserved
    INT_NMI,                //  vector 7 NMI
    INT_TRAP1,              //  vector 8 TRAP
    INT_TRAP2,              //  vector 9 TRAP
    INT_TRAP3,              //  vector 10 TRAP
    INT_TRAP4,              //  vector 11 TRAP
    INT_IRQ0,               //  vector 12 IRQ0
    INT_IRQ1,               //  vector 13 IRQ1
    INT_IRQ2,               //  vector 14 IRQ2
    INT_IRQ3,               //  vector 15 IRQ3
    INT_IRQ4,               //  vector 16 IRQ4
    INT_IRQ5,               //  vector 17 IRQ5
    (fp)0L,                 //  vector 18 Reserved
    (fp)0L,                 //  vector 19 Reserved
```

```
    INT_WOVI,               //  vector 20 WOVI
    INT_CMI,                //  vector 21 CMI
    (fp)0L,                 //  vector 22 Reserved
    (fp)0L,                 //  vector 23 Reserved
    INT_IMIA0,              //  vector 24 IMIA0
    INT_IMIB0,              //  vector 25 IMIB0
    INT_OVI0,               //  vector 26 OVI0
    (fp)0L,                 //  vector 27 Reserved
    INT_IMIA1,              //  vector 28 IMIA1
    INT_IMIB1,              //  vector 29 IMIB1
    INT_OVI1,               //  vector 30 OVI1
    (fp)0L,                 //  vector 31 Reserved
    INT_IMIA2,              //  vector 32 IMIA2
    INT_IMIB2,              //  vector 33 IMIB2
    INT_OVI2,               //  vector 34 OVI2
    (fp)0L,                 //  vector 35 Reserved
    INT_IMIA3,              //  vector 36 IMIA3
    INT_IMIB3,              //  vector 37 IMIB3
    INT_OVI3,               //  vector 38 OVI3
    (fp)0L,                 //  vector 39 Reserved
    INT_IMIA4,              //  vector 40 IMIA4
    INT_IMIB4,              //  vector 41 IMIB4
    INT_OVI4,               //  vector 42 OVI4
    (fp)0L,                 //  vector 43 Reserved
    INT_DEND0A,             //  vector 44 DEND0A
    INT_DEND0B,             //  vector 45 DEND0B
    INT_DEND1A,             //  vector 46 DEND1A
    INT_DEND1B,             //  vector 47 DEND1B
    (fp)0L,                 //  vector 48 Reserved
    (fp)0L,                 //  vector 49 Reserved
    (fp)0L,                 //  vector 50 Reserved
    (fp)0L,                 //  vector 51 Reserved
    INT_ERI0,               //  vector 52 ERI0
    INT_RXI0,               //  vector 53 RXI0
    INT_TXI0,               //  vector 54 TXI0
    INT_TEI0,               //  vector 55 TEI0
    INT_ERI1,               //  vector 56 ERI1
    INT_RXI1,               //  vector 57 RXI1
    INT_TXI1,               //  vector 58 TXI1
    INT_TEI1,               //  vector 59 TEI1
    INT_ADI                 //  vector 60 ADI
};


#pragma section IntPRG

void INT_NMI(void)      { /*sleep();*/ }
void INT_TRAP1(void)    { /*sleep();*/ }
void INT_TRAP2(void)    { /*sleep();*/ }
void INT_TRAP3(void)    { /*sleep();*/ }
void INT_TRAP4(void)    { /*sleep();*/ }
void INT_IRQ0(void)     { /*sleep();*/ }
void INT_IRQ1(void)     { /*sleep();*/ }
void INT_IRQ2(void)     { /*sleep();*/ }
void INT_IRQ3(void)     { /*sleep();*/ }
```

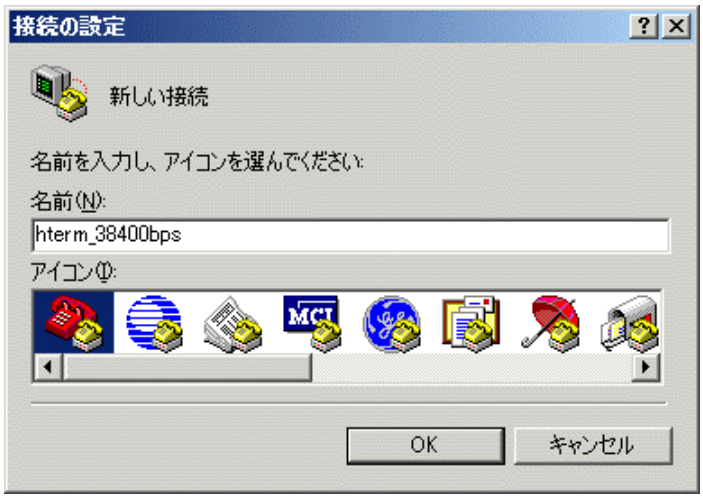
```
void INT_IRQ4(void)     { /*sleep();*/ }
void INT_IRQ5(void)     { /*sleep();*/ }
void INT_WOVI(void)     { /*sleep();*/ }
void INT_CMI(void)      { /*sleep();*/ }
void INT_IMIA0(void)    { /*sleep();*/ }
void INT_IMIB0(void)    { /*sleep();*/ }
void INT_OVI0(void)     { /*sleep();*/ }
void INT_IMIA1(void)    { /*sleep();*/ }
void INT_IMIB1(void)    { /*sleep();*/ }
void INT_OVI1(void)     { /*sleep();*/ }
void INT_IMIA2(void)    { /*sleep();*/ }
void INT_IMIB2(void)    { /*sleep();*/ }
void INT_OVI2(void)     { /*sleep();*/ }
void INT_IMIA3(void)    { /*sleep();*/ }
void INT_IMIB3(void)    { /*sleep();*/ }
void INT_OVI3(void)     { /*sleep();*/ }
void INT_IMIA4(void)    { /*sleep();*/ }
void INT_IMIB4(void)    { /*sleep();*/ }
void INT_OVI4(void)     { /*sleep();*/ }
void INT_DEND0A(void)   { /*sleep();*/ }
void INT_DEND0B(void)   { /*sleep();*/ }
void INT_DEND1A(void)   { /*sleep();*/ }
void INT_DEND1B(void)   { /*sleep();*/ }
void INT_ERI0(void)     { /*sleep();*/ }
void INT_RXI0(void)     { /*sleep();*/ }
void INT_TXI0(void)     { /*sleep();*/ }
void INT_TEI0(void)     { /*sleep();*/ }
void INT_ERI1(void)     { /*sleep();*/ }
void INT_RXI1(void)     { /*sleep();*/ }
void INT_TXI1(void)     { /*sleep();*/ }
void INT_TEI1(void)     { /*sleep();*/ }
void INT_ADI(void)      { /*sleep();*/ }
```

ターミナルソフトによるルネサス製「組み込み型モニタ」の使い方

Windows のバージョンによっては Hterm が動作しないことがあります。そのときはターミナルソフトでルネサス製「組み込み型モニタ」を利用します。ここでは，WindowsXP まで標準で付属していた「ハイパーターミナル」でモニタプログラムを動かしてみます。（ほかのターミナルソフトも使用できます。以下の情報を参考にしてください。）

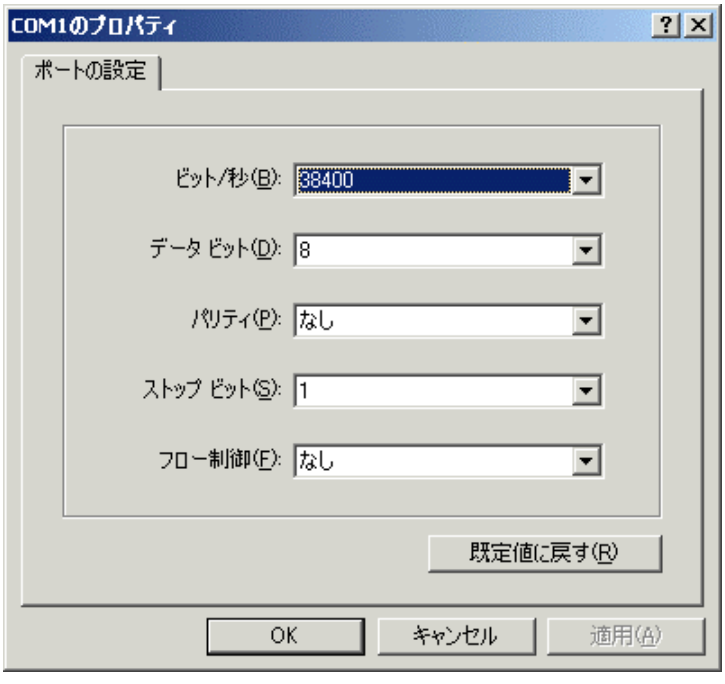
まず，あらかじめ TK-3052 をつなぐパソコンの COM ポートの番号を調べておきます。このマニュアルでは COM1 を使います。

スタートメニューからハイパーターミナルを起動してください。「接続の設定」ダイアログが開きます。今回は名前に「hterm_38400bps」とつけました。「OK」をクリックします。



次に使用する COM ポート番号を指定します。このパソコンでは COM1 を指定しました。「OK」をクリックします。

「COM1 のプロパティ」ダイアログが開きます。ポートの設定を行ないます。「OK」をクリックします。

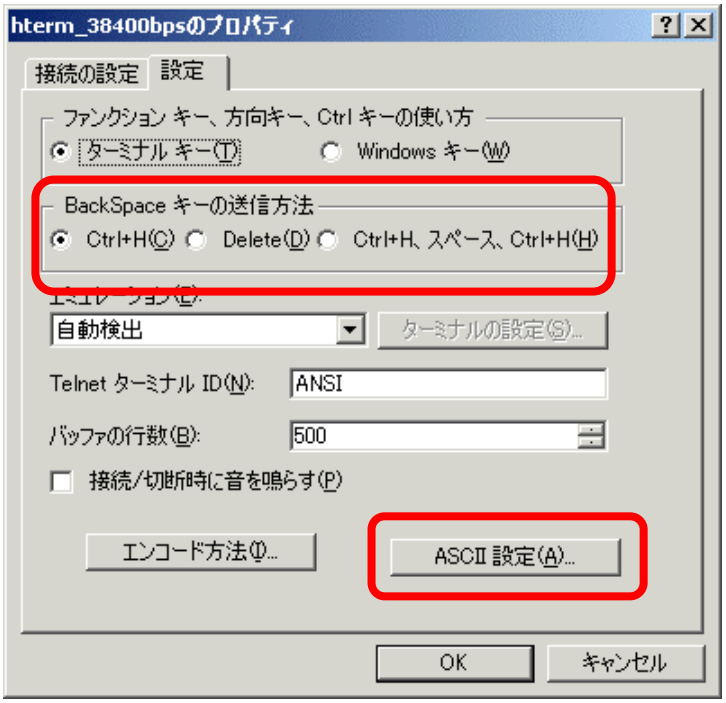


通常はこれで使用できます。しかし，もしターミナルの表示が本マニュアルと異なっていたり，バックスペースキーが使えないようであれば，このページに記されている設定を確認してください。

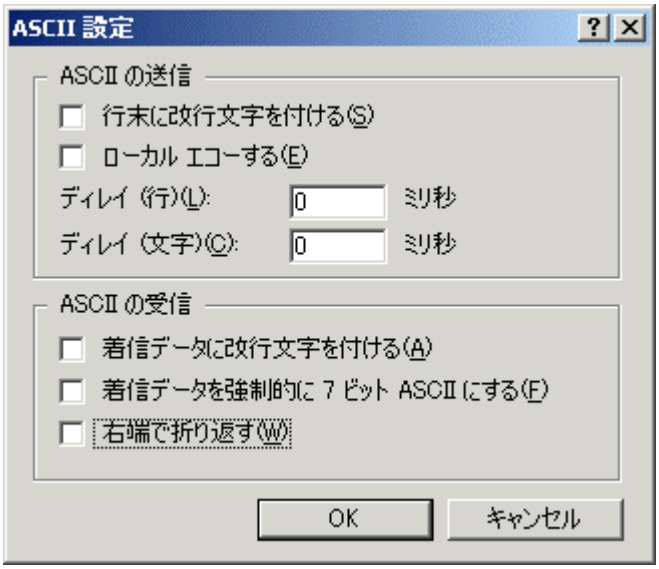
いったん通信を切断します。右の図の「切断」ボタンをクリックしてください。

次に右の図の「プロパティ」のボタンをクリックします。

「hterm_38400bps のプロパティ」ダイアログが開きます。「設定」タブをクリックします。「BackSpace キーの送信方法」は「Ctrl+H(C)」を選択します。次に「ASCII 設定」ボタンをクリックします。

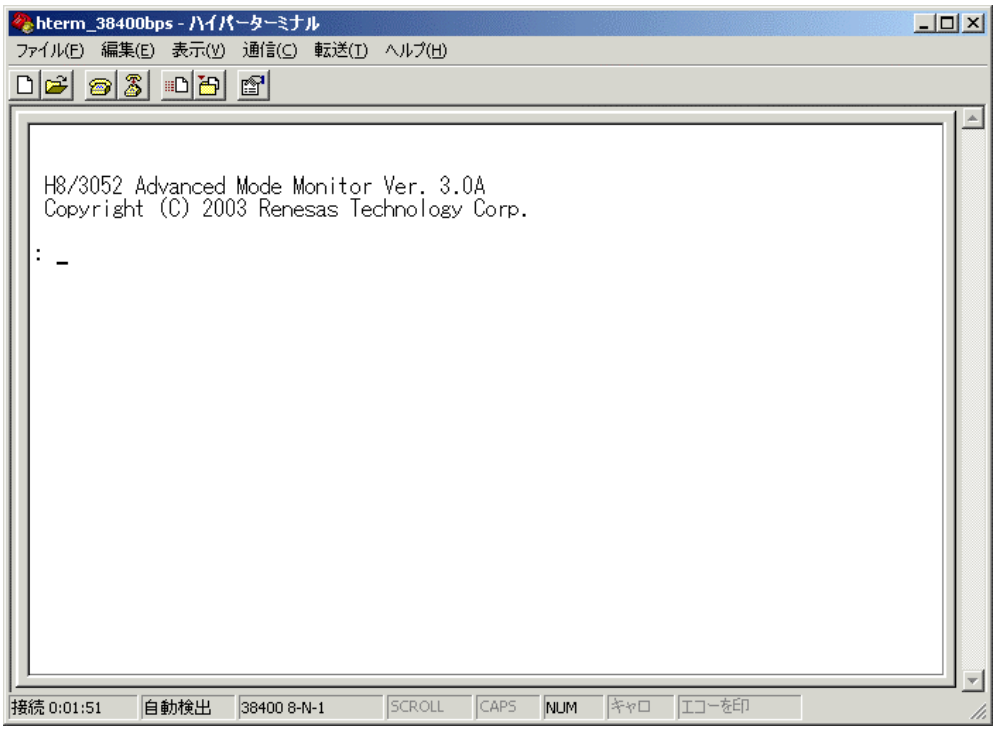


「ASCII 設定ダイアログ」が開きます。すべてのチェックを外します。「OK」をクリックします。さらに「OK」をクリックして「hterm_38400bps のプロパティ」ダイアログを閉じます。



これで設定は終わりました。右の図の「電話」ボタンをクリックしてください。

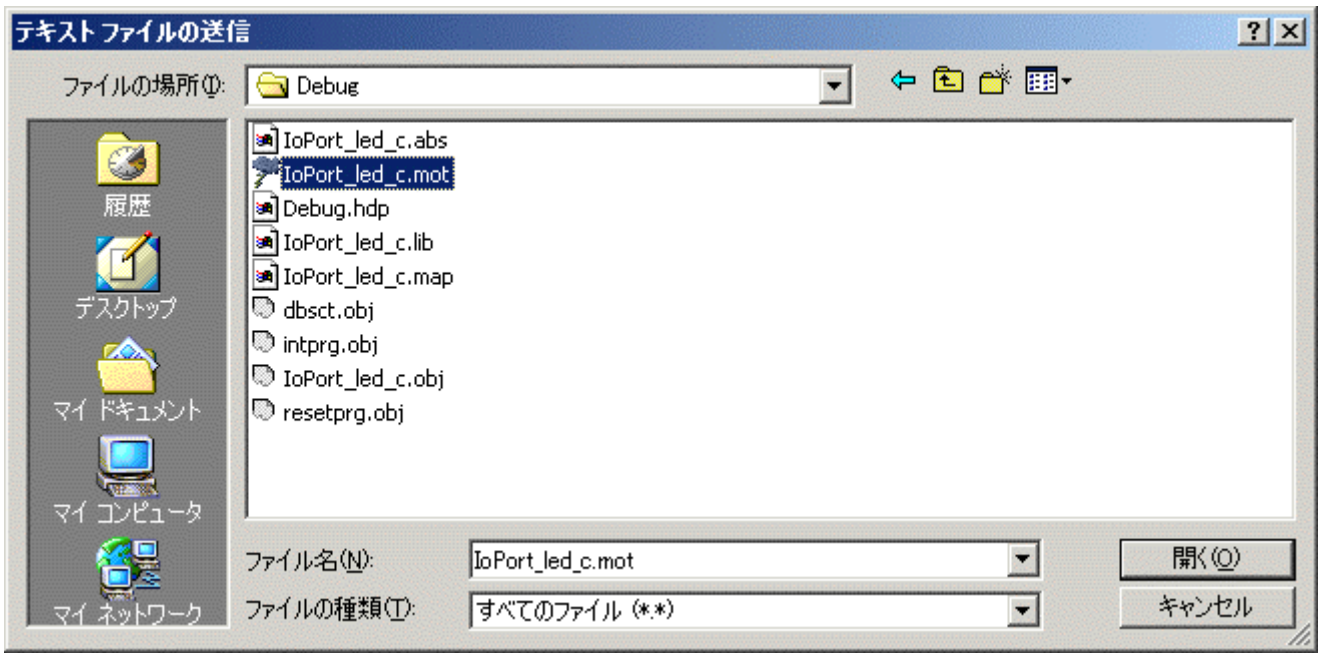
TK-3052 の電源をオンしてください。ハイパーターミナルの画面に次のように表示されたら OK です。



それでは，第 4 章「I/O ポートの使い方」，「4. LED の点滅」の，最初の例題のプログラム（IoPort_led_c）を実行してみましょう。まずプログラムをダウンロードします。「L」「Enter」と入力してください。

次に，メニューから「転送」→「テキストファイルの送信」を選択します。

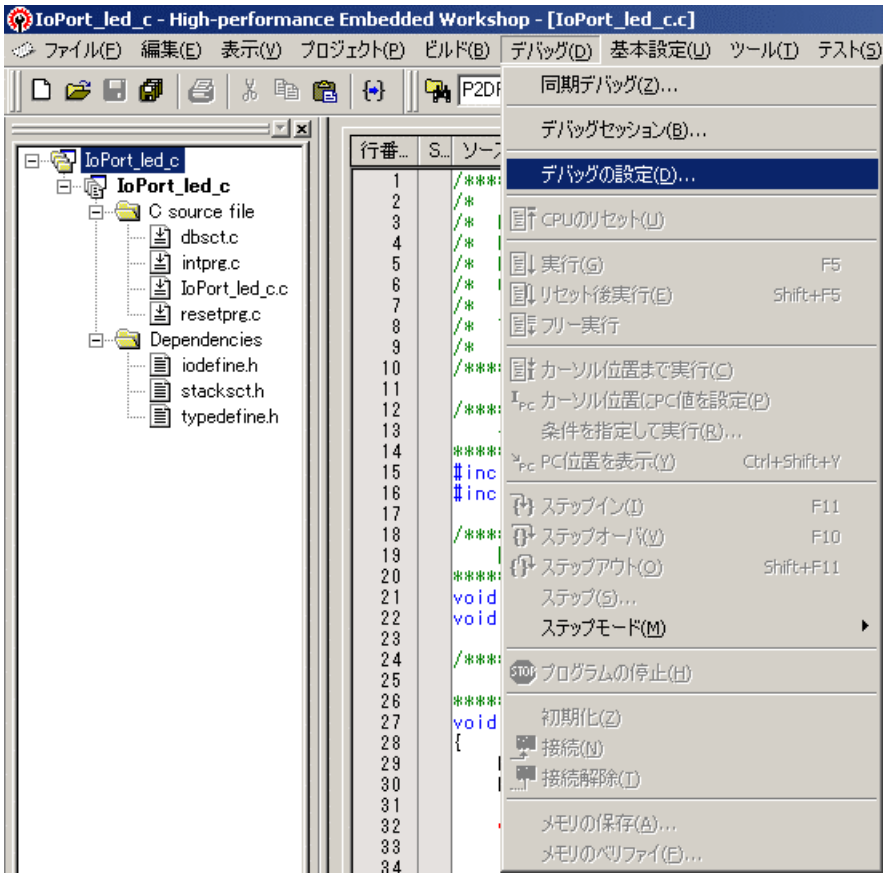
「テキストファイルの送信」ダイアログが開きます。「IoPort_led. mot」をダブルクリックしてください。



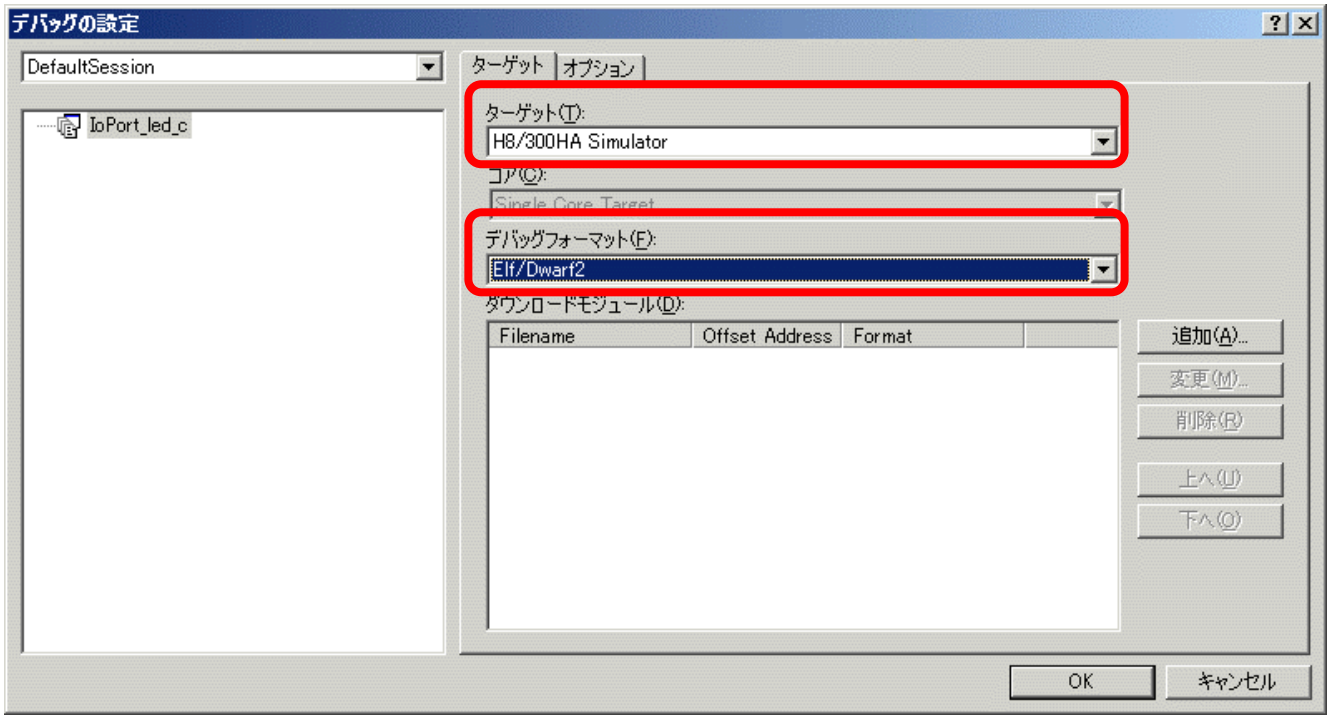
ダウンロードが始まります。次のように表示されたらダウンロード終了です。

次にダウンロードしたプログラムを実行しますが，そのために，プログラムの配置されているアドレスを HEW で見られるようにします。

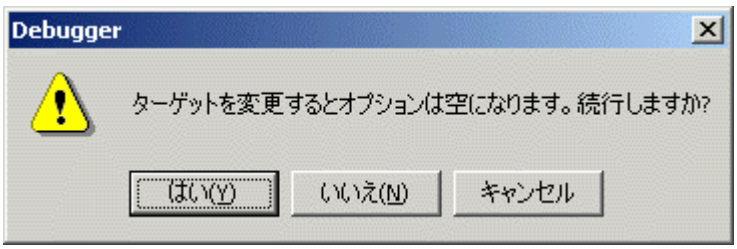
HEW のメニューから，「デバッグ」→「デバッグの設定」を選択します。



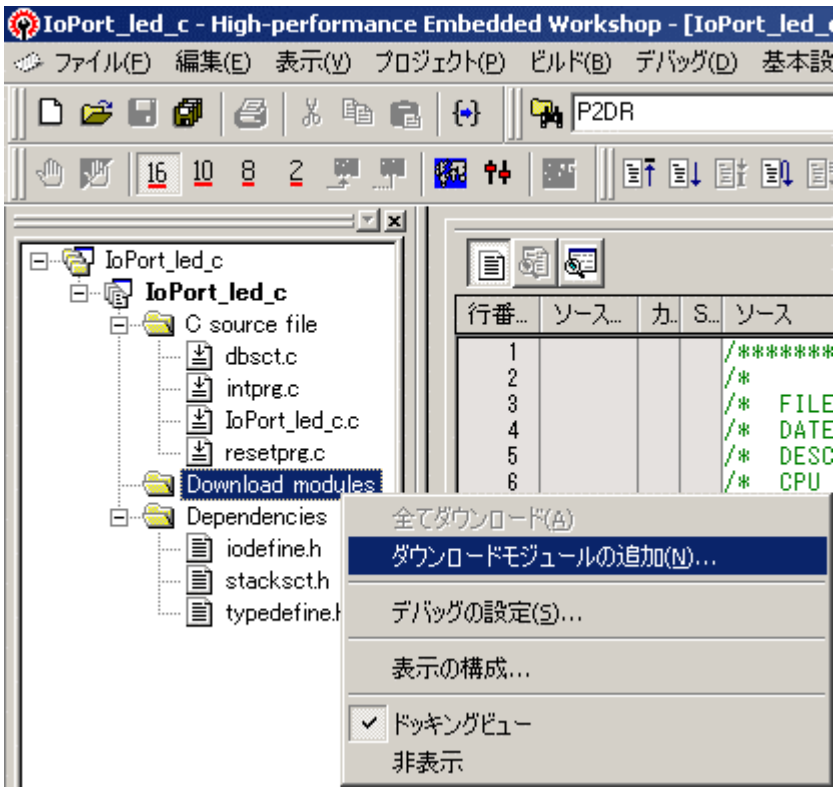
「デバッグの設定」ダイアログが開きますので，「ターゲット」を「H8/300HA Simulator」，「デバッグフォーマット」を「Elf/Dwarf2」に設定して「OK」をクリックします。



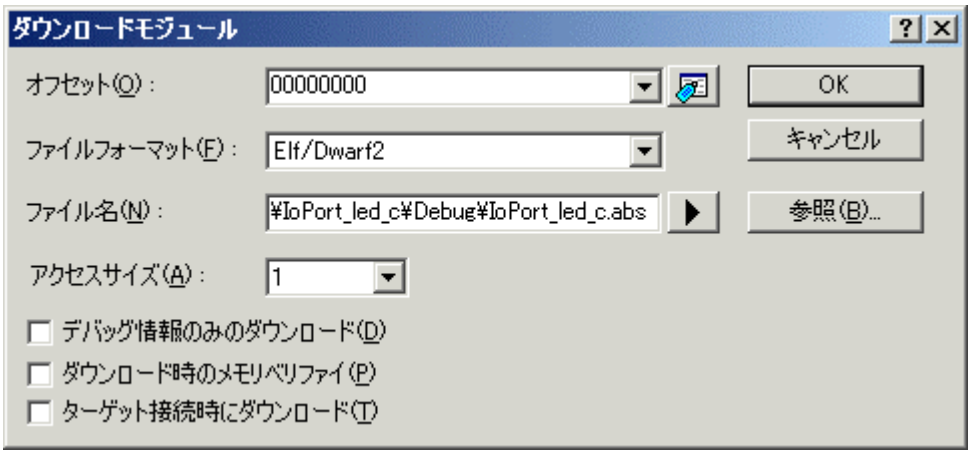
「Debugger」ダイアログが開くことがありますが，「はい」をクリックします。



ダウンロードモジュールを追加します。プロジェクトウィンドウの「Download module」を右クリックしてください。表示されるメニューから「ダウンロードモジュールの追加」を選択します。



「ダウンロードモジュール」ダイアログが開きます。「参照」をクリックするとダウンロードモジュールを選択するウィンドウが開きます。ここで「IoPort_led_c. abs」を選択してください。最後に「OK」をクリックします。



そうすると，HEW の画面にアドレスが表示されるようになります。

行番…	ソース…	力…	S…	ソース
1				/***/
2				/* */
3				/* FILE :IoPort_led_c.c */
4				/* DATE :Wed, Jun 02, 2010 */
5				/* DESCRIPTION :Main Program */
6				/* CPU TYPE :H8/3052F */
7				/* */
8				/* This file is programed by TOYO-LINX Co.,Ltd. / yKikuchi */
9				/* */
10				/***/
11				
12				/**
13				インクルードファイル
14				**/
15				#include <machine.h> // H8特有の命令を使う
16				#include "iodefine.h" // 内蔵I/Oのラベル定義
17				
18				/**
19				関数の定義
20				**/
21				void main(void);
22				void wait(void);
23				
24				/**
25				メインプログラム
26				**/
27	0FE800			void main(void)
28				{
29	0FE800			PA.DDR = 0xff; // ポートAを出力に設定
30	0FE804			PA.DR.BYTE = 0x00; // ポートA出力クリア
31				
32	0FE814			while(1){
33	0FE808			PA.DR.BIT.B0 = 1; // LEDオン
34	0FE80C			wait();
35	0FE80E			PA.DR.BIT.B0 = 0; // LEDオフ
36	0FE812			wait();
37				}
38				}
39				
40				/**
41				ウェイト
42				**/
43	0FE816			void wait(void)
44				{
45				unsigned long i;
46				
47	0FE816			for (i=0;i<2083333;i++){}
48	0FE820			}
49				

IoPort_led_c.c

ちなみに，HEW の画面に，プログラムの配置されているアドレスだけではなく，コンパイルした結果どのようなマシン語に変換されたかを表示することもできます。


```
1                                                                               /*********************************************************************/
2                                                                               /*                                                                   */
3                                                                               /*  FILE        :IoPort_led_c.c                                      */
4                                                                               /*  DATE        :Wed, Jun 02, 2010                                   */
5                                                                               /*  DESCRIPTION :Main Program                                        */
6                                                                               /*  CPU TYPE    :H8/3052F                                            */
7                                                                               /*                                                                   */
8                                                                               /*  This file is programed by TOYO-LINX Co.,Ltd. / yKikuchi           */
9                                                                               /*                                                                   */
10                                                                              /*********************************************************************/
11
12                                                                              /*********************************************************************
13                                                                                  インクルードファイル
14                                                                              *********************************************************************/
15                                                                              #include    <machine.h>     // H8特有の命令を使う
16                                                                              #include    "iodefine.h"    // 内蔵I/Oのラベル定義
17
18                                                                              /*********************************************************************
19                                                                                  関数の定義
20                                                                              *********************************************************************/
21                                                                              void        main(void);
22                                                                              void        wait(void);
23
24                                                                              /*********************************************************************
25                                                                                  メインプログラム
26                                                                              *********************************************************************/
27                                                                              void main(void)
28                                                                              {
29                                                                                  PA.DDR      =   0xff;   // ポートAを出力に設定
        0FE800  F8FF            _main           MOV.B     #H'FF,R0L
        0FE802  38D1                            MOV.B     R0L,@H'FFFFD1:8
30                                                                                  PA.DR.BYTE  =   0x00;   // ポートA出力クリア
        0FE804  1888                            SUB.B     R0L,R0L
        0FE806  38D3                            MOV.B     R0L,@H'FFFFD3:8
31
32                                                                                  while(1){
        0FE814  40F2                            BRA       @H'FE808:8
33                                                                                      PA.DR.BIT.B0 = 1;   // LEDオン
        0FE808  7FD37000                        BSET      #0,@H'FFFFD3:8
34                                                                                      wait();
        0FE80C  5508                            BSR       @_wait:8
35                                                                                      PA.DR.BIT.B0 = 0;   // LEDオフ
        0FE80E  7FD37200                        BCLR      #0,@H'FFFFD3:8
36                                                                                      wait();
        0FE812  5502                            BSR       @_wait:8
        0FE814  40F2                            BRA       @H'FE808:8
37                                                                                  }
38                                                                              }
39
40                                                                              /*********************************************************************
41                                                                                  ウェイト
42                                                                              *********************************************************************/
43                                                                              void wait(void)
44                                                                              {
45                                                                                  unsigned long i;
46
47                                                                                  for (i=0;i<2083333;i++){}
        0FE816  7A00001FCA05    _wait           MOV.L     #H'001FCA05,ER0
        0FE81C  1B70                            DEC.L     #1,ER0
        0FE81E  46FC                            BNE       @H'FE81C:8
48                                                                              }
        0FE820  5470                            RTS
49
```

では実行してみましょう。プログラムは「PResetPRG」セクションからスタートします。HEW で「resetprg. c」を開いてください。FE400h 番地から始まっています。それで,「G FE400」「Enter」と入力します。

LED が点滅すれば OK です。実行を終了するときは TK-3052 のリセットスイッチをオンします。

今度はメインループの先頭(while 文の中の先頭)でブレークします。FE808h 番地なので「B FE808」「Enter」と入力します。

先ほどと同じように「G FE400」「Enter」と入力します。今度は FE808h 番地で止まります。あとは, 止まりたいところにブレークを設定して実行し, きちんと動くか確認していきます。

なお, 一度設定したブレークアドレスを解除するときは「B – FE808」「Enter」と入力します。

ルネサスエレクトロニクスのモニタプログラムには便利なコマンドがいくつもあります。次項では, 各コマンドを解説します。

Hterm 用モニタプログラムのコマンド一覧

■ コマンドの機能一覧

A	Assemble	１行アセンブル
B	Breakpoint	ブレークポイントの設定，表示，解除
D	Dump	メモリ内容のダンプ
DA	DisAssemble	逆アセンブル
F	Fill	データの書き込み
G	Go	ユーザプログラムの実行
H8	H8 status	内蔵周辺機能の状態表示
L	Load	ユーザプログラムのダウンロード
M	Memory	メモリ内容の表示，変更
R	Register	ＣＰＵレジスタの一覧表示
S	Step	シングルステップの実行
.	.<register>	ＣＰＵレジスタの表示，変更
?	help	コマンドヘルプ

■ コマンド形式

H8/300H アドバンストモード用の組み込み型モニタが持っているコマンドの詳細は以下の形式で説明されています。

《 Ｂ （ブレーク...） 》	→	コマンド名
（１）コマンドフォーマット	→	コマンドの入力方法について説明しています。 （下記のコマンドフォーマットの読み方を参照ください。）
（２）機能	→	コマンドの機能概要を説明しています。
（３）解説	→	コマンド機能の詳細を例を挙げて説明しています。
（４）注意事項	→	コマンドを使用する上で特に注意すべき事項が記載してあります。
（５）備考	→	コマンドを使用する上での補足説明が記載してあります。

コマンドフォーマットの特殊記号は以下に示す意味を持っています。

- ″<パラメータ>″ は< >で囲まれた内容を指定することを意味します。
- ″[ ]″ は[ ]で囲まれた内容が省略可能であることを意味します。
- [RET] は改行キーを入力することを意味します。

また，アドレスの表現はアドレス空間が１Ｍと 16Mでは異なるため，以下の指定で行ってください。

- １Mの場合 → 16 進５桁以内（ H'0 ～ H'FFFFF まで指定可能）
- 16Mの場合 → 16 進６桁以内（ H'0 ～ H'FFFFFF まで指定可能）

このページ以降の資料は，株式会社ルネサスエレクトロニクス作成の「Hterm. hlp」をコピー，抜粋したものです。

■ 《 A （1行アセンブル） 》

（1）コマンドフォーマット

: A 〈アドレス〉 [RET]

〈アドレス〉: 命令を埋め込むメモリの先頭アドレス

（2）機能

指定されたアドレスより命令を埋め込みます。
コマンド投入後は下記の操作が可能です。

- [RET] を入力すると２バイト先の番地を表示します。
- ^ [RET] を入力すると２バイト手前の番地を表示します。
- 〈命令〉 [RET] を入力するとメモリの内容を〈命令〉に変更します。
- . [RET] を入力するとＡコマンドを終了します。

（3）解説

```
: DA 1000 100F [RET]
  <ADDR>  <CODE>                <MNEMONIC> <OPERAND>
  001000  0000                  NOP
  001002  0000                  NOP
  001004  0000                  NOP
  001006  0000                  NOP
  001008  0000                  NOP
  00100A  0000                  NOP
  00100C  0000                  NOP
  00100E  0000                  NOP
: A 1000 [RET]
  001000  >  CMP.B R0L,R1L [RET]
  001002  >  BEQ   1000 [RET]
  001004  >  JSR   @2000:24 [RET]
  001008  >  MOV.L @(100:24,ER3),ER5 [RET]
  001012  >  [RET]
  001014  >  ^ [RET]
  001012  >  RTE [RET]
  001014  >  . [RET]
: DA 1000 1014 [RET]
  <ADDR>  <CODE>                <MNEMONIC> <OPERAND>
  001000  1C89                  CMP.B      R0L,R1L
  001002  47FC                  BEQ        001000:8
  001004  5E002000              JSR        @H'002000:24
  001008  010078306B2500000100  MOV.L      @(H'000100:24,ER3),ER5
  001012  5670                  RTE
  001014  0000                  NOP
```

（4）注意事項

アセンブルを行う番地は偶数番地でなければなりません。アドレスを奇数番地とした場合は「Invalid Start Address」のエラーメッセージを表示します。また，〈命令〉の中で指定する値はアドレッシングモードの指定を行う[:2, :3, :8, :16, :24, :32]以外全て 16 進表現です。全てが 16 進表現であるため H' の指定も不要です。

（5）備考

Ａコマンドではオペランドサイズを省略した場合，以下の解釈でオペランドサイズを扱います。

- バイトサイズ

  ADD, ADDX, AND, ANDC, BAND, BCLR, BIAND, BILD, BIOR, BIST, BIXOR, BNOT, BOR, BSET, BST, BTST, BXOR, CMP, DAA, DAS, DEC, DIVXS, DIVXU, EEPMOV, INC, LDC, MOV, MOVFPE, MOVTPE, MULXS, MULXU, NEG, NOT, OR, ORC, ROTL, ROTR, ROTXL, ROTXR, SHAL, SHAR, SHLL, SHLR, STC, SUB, SUBX, XOR, XORC

- ワードサイズ

EXTS, EXTU, POP, PUSH

・ロングワードサイズ

ADDS, SUBS

また，アドレッシングモードは以下の指定が可能です。

・レジスタ直接

R0L ～ R7L, R0H ～ R7H, R0 ～ R7, E0 ～ E7, ER0 ～ ER7, SP, CCR

・レジスタ間接

@ER0 ～ @ER7, @SP

・ポストインクリメントレジスタ間接

@ER0+ ～ @ER7+, @SP+

・プリデクリメントレジスタ間接

@-ER0 ～ @-ER7, @-SP

・ディスプレースメント付きレジスタ間接

@(disp,ER0) ～ @(disp,ER7), @(disp,SP)

指定したディスプレースメントにより８ビット，16 ビットディスプレースメントを自動的に切り替えます。

@(disp:8,ER0) ～ @(disp:8,ER7), @(disp:8,SP)

８ビットディスプレースメント付きレジスタ間接

@(disp:16,ER0) ～ @(disp:16,ER7), @(disp:16,SP)

16 ビットディスプレースメント付きレジスタ間接

・絶対アドレス

@abs　指定したアドレスにより８ビット，16 ビット，24 ビット絶対アドレスを自動的に切り替えます。

@abs:8　８ビット絶対アドレス

@abs:16　16 ビット絶対アドレス

@abs:24　24 ビット絶対アドレス

・イミディエイト

#imm　命令のオペランドサイズにより２ビット，３ビット，８ビット，16 ビット，32 ビットイミディエイトを自動的に切り替えます。

#imm:2　TRAPA 命令での２ビットイミディエイト

#imm:3　ビット操作命令での３ビットイミディエイト

#imm:8　８ビットイミディエイト

#imm:16　16 ビットイミディエイト

#imm:32　32 ビットイミディエイト

・ＰＣ相対

disp　指定した分岐先の番地により８ビット，16 ビット相対を自動的に切り替えます。

disp:8　８ビット相対

disp:16　16 ビット相対

・メモリ間接

@@abs, @@abs:8

■ 《 B （ブレークポイントの設定，解除，表示） 》

（１）コマンドフォーマット

（ａ） ：B 〈アドレス〉[RET]

（ｂ） ：B －[〈アドレス〉][RET]

（ｃ） ：B [RET]

〈アドレス〉：ブレークポイントを設定，解除するアドレス

－ ：設定値の解除

（２）機能

ユーザプログラムを停止するアドレス（ブレークポイント）を設定，解除，表示します。

（ａ） 最大８個までのブレークポイントが設定できます。

（ｂ） 設定されているブレークポイントを解除します。〈アドレス〉を省略すると全て解除します。

（ｃ） 設定されているブレークポイントを表示します。

（３）解説

（ａ）ブレークポイントの設定

：B 100 [RET]

H'100 番地にブレークポイントを設定します。
既に設定されているアドレスを指定した場合は「Duplicate Breakpoint」のエラーメッセージを表示します。

：B 1000 [RET]

：B 200 [RET]

続けて H'1000 番地，H'200 番地にブレークポイントを設定します。
設定できるブレークポイントの個数は最大８個までです。ブレークポイントの設定が８個を超えた場合は「Full Breakpoint」のエラーメッセージを表示します。

：B 15000 [RET]

：G [RET]

```
Break at PC=015000
PC=015000  CCR=80:I.......  SP=00FFFF00
ER0=00000000  ER1=00000000  ER2=00000000  ER3=00000000
ER4=00000000  ER5=00000000  ER6=00000000  ER7=00FFFF00
```

ブレークポイントにユーザプログラムが到達した場合，上記のメッセージを表示してユーザプログラムを停止します。

（ｂ）ブレークポイントの解除

：B - 1000 [RET]

H'1000 番地に設定してあるブレークポイントを解除します。
指定されたアドレスにブレークポイントがない場合は「Not Find Breakpoint」のエラーメッセージを表示します。

：B - [RET]

アドレスを省略すると設定されている全てのブレークポイントを解除します。

（ｃ）ブレークポイントの表示

：B [RET]

```
<ADDR>      001000      000200
```

ブレークポイントが H'1000 番地，H'200 番地に設定してあります。

（４）注意事項

（ａ）ブレークポイントをＲＯＭ領域，内蔵周辺機能のレジスタ領域，およびメモリが接続されていない領域に設定した場合，Ｇ，Ｓコマンド実行時の動作は保証されません。

（ｂ）ブレークポイントは命令の先頭アドレスにのみ設定可能です。命令の先頭以外に設定した場合，Ｇ，Ｓコマンド実行時の動作は保証されません。

■ 《 D （メモリ内容のダンプ） 》

（１）コマンドフォーマット

：D 〈アドレス１〉[〈アドレス２〉][;〈サイズ〉][RET]

〈アドレス１〉：ダンプするメモリの先頭アドレス

〈アドレス２〉：ダンプするメモリの最終アドレス

〈サイズ〉　　：表示単位の指定

B　　：１バイト単位

W　　：２バイト単位

L　　：４バイト単位

省略時：１バイト単位

（２）機能

（ａ）指定されたメモリ領域の内容を表示します。〈アドレス２〉が省略された場合，256 バイトをダンプします。

（ｂ）メモリダンプの表示は１行 16 バイト単位です。

（３）解説

（ａ）メモリ内容のダンプ

：D 1000 [RET]

```
<ADDR>                  <  D  A  T  A  >                    < ASCII CODE >
001000  00 01 02 03 04 05 06 07  08 09 0A 0B 0C 0D 0E 0F  "................"
001010  10 11 12 13 14 15 16 17  18 19 1A 1B 1C 1D 1E 1F  "................"
001020  20 21 22 23 24 25 26 27  28 29 2A 2B 2C 2D 2E 2F  " !"#$%&'()*+,-./"
001030  30 31 32 33 34 35 36 37  38 39 3A 3B 3C 3D 3E 3F  "0123456789:;<=>?"
001040  40 41 42 43 44 45 46 47  48 49 4A 4B 4C 4D 4E 4F  "@ABCDEFGHIJKLMNO"
001050  50 51 52 53 54 55 56 57  58 59 5A 5B 5C 5D 5E 5F  "PQRSTUVWXYZ[¥]^_"
001060  60 61 62 63 64 65 66 67  68 69 6A 6B 6C 6D 6E 6F  " abcdefghijklmno"
001070  70 71 72 73 74 75 76 77  78 79 7A 7B 7C 7D 7E 7F  "pqrstuvwxyz{|}~."
001080  80 81 82 83 84 85 86 87  88 89 8A 8B 8C 8D 8E 8F  "................"
001090  90 91 92 93 94 95 96 97  98 99 9A 9B 9C 9D 9E 9F  "................"
0010A0  A0 A1 A2 A3 A4 A5 A6 A7  A8 A9 AA AB AC AD AE AF  "................"
0010B0  B0 B1 B2 B3 B4 B5 B6 B7  B8 B9 BA BB BC BD BE BF  "................"
0010C0  C0 C1 C2 C3 C4 C5 C6 C7  C8 C9 CA CB CC CD CE CF  "................"
0010D0  D0 D1 D2 D3 D4 D5 D6 D7  D8 D9 DA DB DC DD DE DF  "................"
0010E0  E0 E1 E2 E3 E4 E5 E6 E7  E8 E9 EA EB EC ED EE EF  "................"
0010F0  F0 F1 F2 F3 F4 F5 F6 F7  F8 F9 FA FB FC FD FE FF  "................"
```

H'1000 番地よりメモリ内容をダンプします。

最終アドレスを指定しないと 256 バイトのメモリ内容をダンプします。

：D 1030 105B [RET]

```
<ADDR>                  <  D  A  T  A  >                    < ASCII CODE >
001030  30 31 32 33 34 35 36 37  38 39 3A 3B 3C 3D 3E 3F  "0123456789:;<=>?"
001040  40 41 42 43 44 45 46 47  48 49 4A 4B 4C 4D 4E 4F  "@ABCDEFGHIJKLMNO"
001050  50 51 52 53 54 55 56 57  58 59 5A 5B              "PQRSTUVWXYZ["
```

最終アドレスを指定すると最終アドレスまでのメモリ内容をダンプします。

：D 1030 105B;W [RET]

```
<ADDR>                  <  D  A  T  A  >                    < ASCII CODE >
001030  3031  3233  3435  3637  3839  3A3B  3C3D  3E3F  "0123456789:;<=>?"
001040  4041  4243  4445  4647  4849  4A4B  4C4D  4E4F  "@ABCDEFGHIJKLMNO"
001050  5051  5253  5455  5657  5859  5A5B              "PQRSTUVWXYZ["
```

サイズをワード単位とすると２バイト単位でメモリ内容をダンプします。

：D 1030 105B;L [RET]

```
<ADDR>                  <  D  A  T  A  >                    < ASCII CODE >
001030  30313233    34353637    38393A3B    3C3D3E3F    "0123456789:;<=>?"
001040  40414243    44454647    48494A4B    4C4D4E4F    "@ABCDEFGHIJKLMNO"
001050  50515253    54555657    58595A5B                "PQRSTUVWXYZ["
```

サイズをロングワード単位とすると４バイト単位でメモリ内容をダンプします。

（４）注意事項

（ａ）ワード単位で表示を行う場合，先頭アドレスは偶数番地，最終番地は奇数番地でなければなりません。先頭アドレスが奇数番地の場合は「Invalid Start Address」，最終アドレスが偶数番地の場合は「Invalid End Address」のエラーメッセージを表示します。また，ロングワード単位で表示を行う場合，先頭アドレスは偶数番地，最終番地は先頭アドレス＋３のＮ倍の番地でなければなりません。

（ｂ）Ｄコマンドで内蔵周辺機能のレジスタ領域を表示した場合，メモリ内容の 16 進数と ASCII コードの表示が異なることがあります。

（５）備考

Ｄコマンドではメモリの内容をリードする際，サイズで指定された単位でメモリ内容のリードを行います。すなわち，表示単位が１バイトの場合はバイトサイズでリードし，表示単位が２バイトの場合はワードサイズでリードし，表示単位が４バイトの場合はロングワードサイズでリードを行います。

■ 《 ＤＡ （逆アセンブル） 》

（１）コマンドフォーマット

：DA〈アドレス１〉[〈アドレス２〉] [RET]

〈アドレス１〉：逆アセンブルするメモリの先頭アドレス

〈アドレス２〉：逆アセンブルするメモリの最終アドレス

（２）機能

指定されたメモリ領域の内容を逆アセンブルします。〈アドレス２〉が省略された場合，16 命令文を逆アセンブルします。

（３）解説

メモリ内容の逆アセンブル

```
: DA 1000 [RET]
  <ADDR>  <CODE>              <MNEMONIC> <OPERAND>
  001000  6CD3                MOV.B      R3H,@-ER5
  001002  0A0B                INC.B      R3L
  001004  7A0100001028        MOV.L      #H'00001028:32,ER1
  00100A  5C0002DA            BSR        0012E8:16
  00100E  40C8                BRA        000FD8:8
  001010  5C0002C4            BSR        0012D8:16
  001014  7A050020FF56        MOV.L      #H'0020FF56:32,ER5
  00101A  FB0F                MOV.B      #H'0F:8,R3L
  00101C  1A80                SUB.L      ER0,ER0
  00101E  01006DD0            MOV.L      ER0,@-ER5
  001022  1A0B                DEC.B      R3L
  001024  46F8                BNE        00101E:8
  001026  5470                RTS
  001028  0820                ADD.B      R2H,R0H
  00102A  0800                ADD.B      R0H,R0H
  00102C  6E68000A            MOV.B      @(H'00000A:16,ER6),R0L
```

最終アドレスを指定しないと 16 命令文の逆アセンブルを行います。

```
: DA 1000 1005 [RET]
  <ADDR>  <CODE>              <MNEMONIC> <OPERAND>
  001000  6CD3                MOV.B      R3H,@-ER5
  001002  0A0B                INC.B      R3L
  001004  7A0100001028        MOV.L      #H'00001028:32,ER1
```

最終アドレスを指定すると最終アドレスを含む命令まで逆アセンブルを行います。

■ 《 F （データの書き込み） 》

（１）コマンドフォーマット

：F〈アドレス１〉〈アドレス２〉〈書き込みデータ〉[RET]

〈アドレス１〉　：書き込みするメモリの先頭アドレス

〈アドレス２〉　：書き込みするメモリの最終アドレス

〈書き込みデータ〉：１バイトの書き込みデータ

（２）機能

指定されたメモリ領域に，〈書き込みデータ〉で指定された１バイトのデータを書き込みます。

（３）解説

（ａ）データの書き込み

```
：F 1000 10FF AA [RET]
```

H'1000 番地から H'10FF 番地までのメモリ領域に対して H'AA のデータを書き込みます。

（ｂ）データの書き込みの失敗

```
：F 100000 1001FF 55 [RET]
  Failed at 100015 , Write = 55 , Read = 04
```

Ｆコマンドでは書き込みデータのベリファイチェックを行います。ベリファイチェックでエラーが検出された場合は上記のメッセージを表示します。

■ 《 G （ユーザプログラムの実行） 》

（１）コマンドフォーマット

：G [〈アドレス〉] [RET]

〈アドレス〉：実行するユーザプログラムの先頭アドレス

（２）機能

現在のプログラムカウンタ値ないしは，〈アドレス〉で指定したアドレスよりユーザプログラムを実行します。

（３）解説

（ａ）ユーザプログラムの実行

```
：G [RET]
```

現在のプログラムカウンタ値よりユーザプログラムを実行します。

```
：G 100000 [RET]
```

H'100000 番地よりユーザプログラムを実行します。

（ｂ）ユーザプログラムの停止

```
：B 15000 [RET]
：G [RET]
  Break at PC=015000
  PC=015000  CCR=80:I.......  SP=00FFFF00
  ER0=00000000  ER1=00000000  ER2=00000000  ER3=00000000
  ER4=00000000  ER5=00000000  ER6=00000000  ER7=00FFFF00
```

ブレークポイントに到達するとユーザプログラムは停止します。

```
：G 10100 [RET]
NMI (Abort Switch ON) 入力
  Abort at PC=013014
  PC=013014  CCR=80:I.......  SP=00FFFF00
  ER0=00000000  ER1=00000000  ER2=00000000  ER3=00000000
  ER4=00000000  ER5=00000000  ER6=00000000  ER7=00FFFF00
```

ＮＭＩ入力を行うとユーザプログラムを強制停止します。

（４）注意事項

実行する最初の命令が不当命令の場合は「Invalid Instruction」のエラーメッセージを表示し，ユーザプログラムの実行を行いません。

■ 《 Ｈ８ （内蔵周辺機能の状態表示） 》

（１）コマンドフォーマット

：H8 [<周辺機能名>] [RET]

<周辺機能名>：状態表示したい周辺機能の名称

H8/3052 の場合

```
DMAC0 - Direct Memory Access Controller 0
DMAC1 - Direct Memory Access Controller 1
ITU   - 16bit Integrated Timer pulse Unit
ITU0  - 16bit Integrated Timer pulse Unit 0
ITU1  - 16bit Integrated Timer pulse Unit 1
ITU2  - 16bit Integrated Timer pulse Unit 2
ITU3  - 16bit Integrated Timer pulse Unit 3
ITU4  - 16bit Integrated Timer pulse Unit 4
TPC   - programmable Timing Pattern Controller
WDT   - Watch Dog Timer
SCI0  - Serial Communication Interface 0
SCI1  - Serial Communication Interface 1
I/O   - I/O port
D/A   - D/A converter
A/D   - A/D converter
INTC  - INTerrupt Controller
BSC   - BuS Controller,etc.
```

（２）機能

内蔵周辺機能のレジスタの状態を表示します。

（３）解説

（ａ）Ｉ／Ｏポートの表示

：H8 I/O [RET]

<REG>	<ADDR>	<CODE>	< 7	6	5	4	3	2	1	0 >
P1DDR	FFC0	FF								
P1DR	FFC2	11111100								
P2DDR	FFC1	FF								
P2DR	FFC3	11111111								
P2PCR	FFD8	00000000								
P3DDR	FFC4	FF								
P3DR	FFC6	11111111								
P4DDR	FFC5	FF								
P4DR	FFC7	11111111								
P4PCR	FFDA	00000000								
P5DDR	FFC8	FF								
P5DR	FFCA	1111								
P5PCR	FFDB	0000								
P6DDR	FFC9	FF								
P6DR	FFCB	.1111011								
P7DR	FFCE	11111111	AN7 DA1	AN6 DA0	AN5	AN4	AN3	AN2	AN1	AN0
P8DDR	FFCD	FF								
P8DR	FFCF	...11111					IRQ3	IRQ2	IRQ1	IRQ0
P9DDR	FFD0	FF								
P9DR	FFD2	..001110			SCK1 IRQ5	SCK0 IRQ4	RXD1	RXD0	TXD1	TXD0
PADDR	FFD1	FF								
PADR	FFD3	11111111	TP7 TIOCB2	TP6 TIOCA2	TP5 TIOCB1	TP4 TIOCA1	TP3 TIOCB0	TP2 TIOCA0	TP1 TEND1	TP0 TEND0

```
                                                        TCLKD  TCLKC  TCLKB  TCLKA
 PBDDR   FFD4 FF
 PBDR    FFD6 11111111   TP15   TP14   TP13   TP12   TP11   TP10   TP9    TP8
                         DREQ1  DREQ0  TOCXB4 TOCXA4 TIOCB4 TIOCA4 TIOCB3 TIOCA3
                         ADTRG
```

Ｉ／Ｏポートの状態表示では，現状の動作モードで使用可能なものを表示します。

<REG>ではレジスタの名称，<ADDR>ではレジスタの番地，<CODE>ではレジスタの値を表示します。なお，<CODE>では各レジスタの仕様に合わせて，バイト単位またはビット単位で値を表示します。また，<76543210>では各Ｉ／Ｏポートとの兼用端子機能を表示します。

（ｂ）Ｉ／Ｏポート以外の表示

```
: H8 DMAC0 [RET]
 <REG>  <ADDR> <CODE>   < 7      6      5      4      3      2      1      0 >
 MARA    FF20 FFFFFFFF
 ETCRA   FF24 01FF
 IOARA   FF26 FF
 DTCRA   FF27 00000000   DTE    DTSZ   DTID   RPE    DTIE   DTS2   DTS1   DTS0
 MARB    FF28 FFFFFFFF
 ETCRB   FF2C 00FF
 IOARB   FF2E FF
 DTCRB   FF2F 00000000   DTE    DTSZ   DTID   RPE    DTIE   DTS2   DTS1   DTS0
: H8 ITU0 [RET]
 <REG>  <ADDR> <CODE>   < 7      6      5      4      3      2      1      0 >
 TCR     FF64 .0000000         CCLR1  CCLR0  CKEG1  CKEG0  TPSC2  TPSC1  TPSC0
 TIOR    FF65 .000.000          IOB2   IOB1   IOB0          IOA2   IOA1   IOA0
 TIER    FF66 .....000                                      OVIE   IMIEB  IMIEA
 TSR     FF67 .....000                                      OVF    IMFB   IMFA
 TCNT    FF68 0000
 GRA     FF6A FFFF
 GRB     FF6C FFFF
```

Ｉ／Ｏポート以外の状態表示では，指定された周辺機能の状態を表示します。

<REG>ではレジスタの名称，<ADDR>ではレジスタの番地，<CODE>ではレジスタの値を表示します。なお，<CODE>では各レジスタの仕様に合わせて，ワード単位，バイト単位またはビット単位で値を表示します。また，<76543210>ではビット単位に意味のあるレジスタのみ対応するビットの意味を表示します。

■ 《 L （ユーザプログラムのダウンロード） 》

（１）コマンドフォーマット
：L ファイル名 [RET]

（２）機能
ホスト端末より，指定されたロードモジュールをメモリ上にダウンロードします。なお，ロードするプログラムはターミナルソフトの「テキストファイルの送信」機能を使用してください。また，送信るファイルはモトローラファイル（S タイプフォーマット，ファイル拡張子 MOT）です。

（３）解説
（ａ）ユーザプログラムのダウンロード
：L [RET]
Top Address=010000
End Address=0136F1
（ｂ）ダウンロードの失敗
：L [RET]
******** S Type Format Error ********
Ｓタイプフォーマットでないロードモジュールを転送すると上記のエラーメッセージを表示します。
：L [RET]
******** Check Sum Error ********
チェックサムが不正の場合は上記のエラーメッセージを表示します。

■ 《 M （メモリ内容の表示，変更） 》

（１）コマンドフォーマット
：M〈アドレス〉[;〈サイズ〉] [RET]
〈アドレス〉：表示，変更を行うメモリの先頭アドレス
〈サイズ〉 ：表示，変更の単位の指定
B ：１バイト単位
W ：２バイト単位
L ：４バイト単位
省略時 ：１バイト単位

（２）機能
指定されたアドレスのメモリ内容を〈サイズ〉で指定した単位で表示，変更します。コマンド投入後は下記の操作が可能です。
・[RET] を入力すると次のメモリ内容を表示します。
・^ [RET] を入力すると前のメモリ内容を表示します。
・〈データ〉[RET] を入力するとメモリの内容を〈データ〉に変更します。
・. [RET] を入力するとＭコマンドを終了します。

（３）解説
（ａ）バイト単位の表示，変更
：M 1000 [RET]
001000 00 ? [RET]
001001 3B ? AA [RET]
001002 23 ? BC [RET]
001003 D5 ? ^ [RET]
001002 BC ? ^ [RET]
001001 AA ? . [RET]
H'1001 番地と H'1002 番地の内容を H'AA と H'BC に変更します。
（ｂ）ワード単位の表示，変更
：M 1000;W [RET]
001000 BCD5 ? 1234 [RET]
001002 67D1 ? ABCD [RET]
001004 B80A ? ^ [RET]

```
001002  ABCD  ?  ^ [RET]
001000  1234  ?  [RET]
001002  ABCD  ?  [RET]
001004  B80A  ?  . [RET]
```

H'1000 番地から H'1003 番地までの内容を H'1234 と H'ABCD に変更します。

（ｃ）ロングワード単位の表示，変更

```
: M 1000;L [RET]
001000  BCD567D1  ?  12345678 [RET]
001004  B80AABCD  ?  ABCDEF [RET]
001008  72DFEA00  ?  ^ [RET]
001004  00ABCDEF  ?  ^ [RET]
001000  12345678  ?  [RET]
001004  00ABCDEF  ?  . [RET]
```

H'1000 番地から H'1007 番地までの内容を H'12345678 と H'00ABCDEF に変更します。

（ｄ）ベリファイチェック

```
: M 1000 [RET]
001000  BC  ?  34 [RET]
001001  D1  ?  AB [RET]
001002  B8  ?  12 [RET]
**** Verify Error ****
001002  B8  ?
```

Ｍコマンドではメモリ内容の変更の際にベリファイエラーが検出されると，再び当該アドレスの内容を表示してコマンド待ち状態となります。なお，内蔵周辺機能のレジスタ領域に対してはベリファイチェックを行いません。

（４）注意事項

ワード単位,ロングワード単位でメモリの表示,変更を行う場合は,アドレスは偶数番地でなければなりません。アドレスを奇数番地とした場合は「Invalid Start Address」のエラーメッセージを表示します。

（５）備考

Ｍコマンドではメモリの内容をリード／ライトする際，サイズで指定された単位でリード／ライトを行います。すなわち，サイズが１バイトの場合はバイトサイズでリード／ライトし，サイズが２バイトの場合はワードサイズでリード／ライトし，サイズが４バイト単位の場合はロングワードサイズでリード／ライトを行います。

■ 《 Ｒ （ＣＰＵレジスタの一覧表示） 》

（１）コマンドフォーマット

```
: R [RET]
```

（２）機能

ＣＰＵのコントロールレジスタ，汎用レジスタの一覧を表示します。

（３）解説

```
: R [RET]
PC=000000  CCR=FF:IUHUNZVC  SP=00FFFF00
ER0=00000000  ER1=00000000  ER2=00000000  ER3=00000000
ER4=00000000  ER5=00000000  ER6=00000000  ER7=00FFFF00

PC  : プログラムカウンタ
CCR : コンディションコードレジスタ
[IUHUNZVC] :  I : 割込みマスク・ビット      U : ユーザ・割込みマスク・ビット
              H : ハーフ・キャリー・フラグ  U : ユーザ・ビット
              N : ネガティブ・フラグ        Z : ゼロ・フラグ
              V : オーバフロー・フラグ      C : キャリー・フラグ
SP : スタックポインタ
ER0～ER7 : 汎用レジスタ
```

■ 《 S （シングルステップの実行） 》

（１）コマンドフォーマット

：S [<実行ステップ数>] [RET]

<実行ステップ数>：実行する命令数（10 進数２桁で表現）

（２）機能

ユーザプログラムのシングルステップ実行を行い，レジスタ内容と実行した命令を表示します。

（ａ） 現在のプログラムカウンタ値から指定された命令数分のユーザプログラムを実行します。

（ｂ） 実行ステップ数が省略されると１命令だけユーザプログラムを実行します。

（ｃ） 実行ステップ数で０を指定すると 100 命令分ユーザプログラムを実行します。

（３）解説

（ａ）１命令だけの実行

```
: S [RET]
  PC=200106  CCR=80:I.......  SP=00FFFF00
  ER0=01234567  ER1=00000000  ER2=00000000  ER3=00000000
  ER4=00000000  ER5=00000000  ER6=00000000  ER7=00FFFF00
  200100  7A0001234567          MOV.L       #H'01234567:32,ER0
```

実行ステップ数が省略されると１命令だけユーザプログラムを実行します。

（ｂ）複数命令の実行

```
: S 3 [RET]
  PC=200106  CCR=80:I.......  SP=00FFFF00
  ER0=01234567  ER1=00000000  ER2=00000000  ER3=00000000
  ER4=00000000  ER5=00000000  ER6=00000000  ER7=00FFFF00
  200100  7A0001234567          MOV.L       #H'01234567:32,ER0

  PC=20010C  CCR=88:I...N..  SP=00FFFF00
  ER0=01234567  ER1=89ABCDEF  ER2=00000000  ER3=00000000
  ER4=00000000  ER5=00000000  ER6=00000000  ER7=00FFFF00
  200106  7A0189ABCDEF          MOV.L       #H'89ABCDEF:32,ER1

  PC=20010E  CCR=80:I.......  SP=00FFFF00
  ER0=01234567  ER1=89ABCDEF  ER2=01234567  ER3=00000000
  ER4=00000000  ER5=00000000  ER6=00000000  ER7=00FFFF00
  20010C  0F82                  MOV.L       ER0,ER2
```

実行ステップ数を指定すると指定された命令数分だけ連続的にユーザプログラムを実行します。

（４）注意事項

実行する命令が不当命令の場合は「Invalid Instruction」のエラーメッセージを表示しユーザプログラムの実行を行いません。

■ 《 .<register> （ＣＰＵレジスタの表示，変更） 》

（１）コマンドフォーマット

：．〈レジスタ名〉[〈データ〉] [RET]

〈レジスタ名〉：表示，変更を行うＣＰＵのレジスタ名

〈データ〉　　：設定値

（２）機能

ＣＰＵのコントロール／汎用レジスタの内容を表示，変更します。

〈データ〉を指定すると当該のレジスタのみ変更を行います。

〈データ〉を省略すると当該のレジスタから順番に会話形式でレジスタ値の表示，変更を行います。

・[RET] を入力すると次のレジスタ内容を表示します。

・＾ [RET] を入力すると前のレジスタ内容を表示します。

・〈データ〉[RET] を入力するとレジスタの内容を〈データ〉に変更します。

・． [RET] を入力するとコマンドを終了します。

なお，レジスタの表示順は以下の通りです。

```
ER0, ER1, ER2, ER3, ER4, ER5, ER6, ER7, PC, CCR, SP
R0,  R1,  R2,  R3,  R4,  R5,  R6,  R7
E0,  E1,  E2,  E3,  E4,  E5,  E6,  E7
R0L, R1L, R2L, R3L, R4L, R5L, R6L, R7L
R0H, R1H, R2H, R3H, R4H, R5H, R6H, R7H
```

（３）解説

（ａ）レジスタの変更

```
: R [RET]
  PC=000000  CCR=FF:IUHUNZVC  SP=00FFFF00
  ER0=00000000  ER1=00000000  ER2=00000000  ER3=00000000
  ER4=00000000  ER5=00000000  ER6=00000000  ER7=00FFFF00
: .ER0 12345678 [RET]
: .R1 ABCD [RET]
: .R2L EF [RET]
: R [RET]
  PC=000000  CCR=FF:IUHUNZVC  SP=00FFFF00
  ER0=12345678  ER1=0000ABCD  ER2=000000EF  ER3=00000000
  ER4=00000000  ER5=00000000  ER6=00000000  ER7=00FFFF00
```

〈データ〉を指定すると当該レジスタのみ変更を行います。

（ｂ）レジスタの表示，変更

```
: R [RET]
  PC=000000  CCR=80:I.......  SP=00FFFF00
  ER0=00000000  ER1=00000000  ER2=00000000  ER3=00000000
  ER4=00000000  ER5=00000000  ER6=00000000  ER7=00FFFF00
: .ER6 [RET]
  ER6=00000000  ?  12345678 [RET]
  ER7=00FFFF00  ?  [RET]
  PC=000000  ?  200100 [RET]
  CCR=80  ?  [RET]
  SP=00FFFF00  ?  ^ [RET]
  CCR=80  ?  ^ [RET]
  PC=200100  ?  ^ [RET]
  ER7=00FFFF00  ?  FFFF10 [RET]
  PC=200100  ?  . [RET]
: R [RET]
  PC=200100  CCR=80:I.......  SP=00FFFF10
  ER0=00000000  ER1=00000000  ER2=00000000  ER3=00000000
  ER4=00000000  ER5=00000000  ER6=12345678  ER7=00FFFF10
```

〈データ〉を省略すると会話形式でレジスタの表示，変更を行います。

■ 《 help （コマンドヘルプ） 》

（１）コマンドフォーマット

：[<コマンド名>] ? [RET]

〈コマンド名〉：使用方法を表示したいコマンドの名称

（２）機能

コマンドの使用方法を表示します。

（ａ） 〈コマンド名〉を省略するとモニタが持っているコマンドの一覧を表示します。

（ｂ） 〈コマンド名〉を指定すると該当のコマンドの使用方法を表示します。

（３）解説

（ａ）コマンドの一覧表示

```
: ? [RET]
 Monitor Vector 00000 - 000FF
 Monitor ROM    00100 - 059EB
 Monitor RAM    FDF10 - FDFE3
 User    Vector FE000 - FE0FF

 .  : Changes contents of H8/300H registers.
 A  : Assembles source sentences from the keyboard.
 B  : Sets or displays or clear breakpoint(s).
 D  : Displays memory contents.
 DA : Disassembles memory contents.
 F  : Fills specified memory range with data.
 G  : Executes real-time emulation.
 H8 : Displays contents of H8/3052 peripheral registers.
 L  : Loads user program into memory from host system.
 M  : Changes memory contents.
 R  : Displays contents of H8/300H registers.
 S  : Executes single emulation(s) and displays instruction and registers.
```

〈コマンド名〉を省略するとモニタのメモリマップ及びコマンドの一覧を表示します。

（ｂ）コマンドの詳細表示

```
: B ? [RET]
 1. Sets breakpoint at specified address.
      B <address> [RET]
 2. Displays breakpoint(s).
      B [RET]
 3. Clear breakpoint(s).
      B - [<address>] [RET]
   <address> : address of breakpoint
: D ? [RET]
 Displays memory contents.
      D <address1> [<address2>] [;<size>] [RET]
   <address1> : dump area start address
   <address2> : dump area end   address
   <size>     : B -- byte
                W -- word
                L -- long word
: F ? [RET]
 Fills specified memory range with data.
      F <address1> <address2> <data> [RET]
   <address1> : filling area start address
   <address2> : filling area end   address
   <data>     : filling byte data
: H8 ? [RET]
```

```
Displays contents of H8/3003 peripheral registers.
     H8 <name> [RET]
  <name> : DMAC0 - Direct Memory Access Controller 0
           DMAC1 - Direct Memory Access Controller 1
           ITU   - 16bit Integrated Timer pulse Unit
           ITU0  - 16bit Integrated Timer pulse Unit 0
           ITU1  - 16bit Integrated Timer pulse Unit 1
           ITU2  - 16bit Integrated Timer pulse Unit 2
           ITU3  - 16bit Integrated Timer pulse Unit 3
           ITU4  - 16bit Integrated Timer pulse Unit 4
           TPC   - programmable Timing Pattern Controller
           WDT   - Watch Dog Timer
           SCI0  - Serial Communication Interface 0
           SCI1  - Serial Communication Interface 1
           I/O   - I/O port
           D/A   - D/A converter
           A/D   - A/D converter
           INTC  - INTerrupt Controller
           BSC   - BuS Controller,etc.
```

〈コマンド名〉を指定すると当該コマンドの詳細な使い方を表示します。

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

20120210