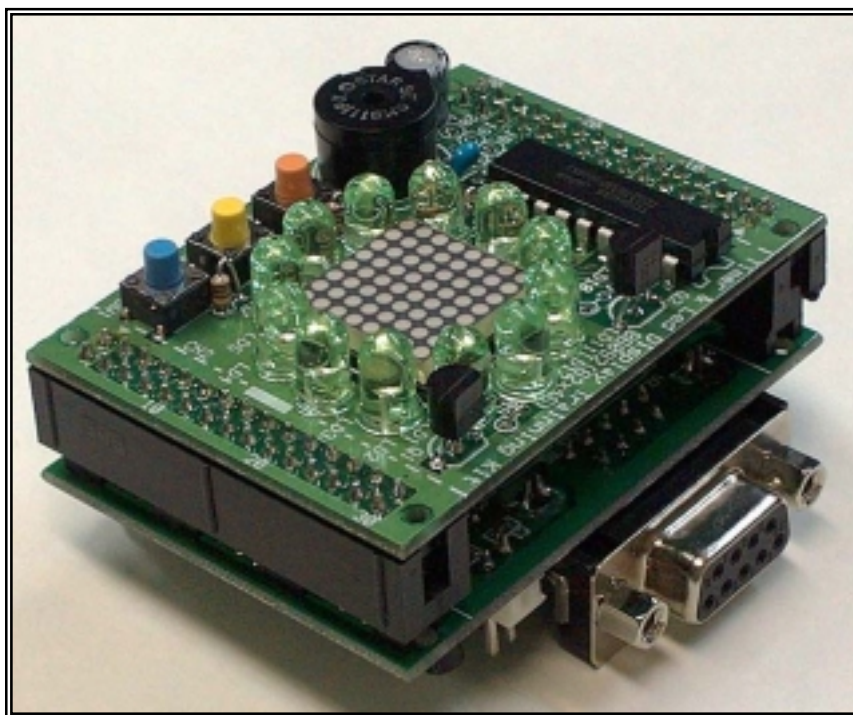


# TK-3687/TK-3687mini Option タイマ&LEDディスプレイ

## Version 2.01



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。たくさんの LED を制御するために人間の目の特徴を巧みに利用しています。このキットは 8×8ドットの LED ディスプレイを利用して、アニメーションを表示したり、タイマを作成します。全体の制御は TK-3687/TK-3687mini で行ないます。

## 目次

1	ハードの組み立て	P. 1
2	動作チェック	P. 4
3	C によるプログラミング	P. 5
4	LED のダイナミック点灯	P. 25
5	ルーレットを作ろう	P. 35
6	AD 変換値の表示	P. 45
7	メロディを奏でよう	P. 51
8	タイマ&LED ディスプレイへの応用	P. 59
	付録	P. 63

**(株)東洋リンクス**

# 1 ハードの組み立て

まずはハードを組み立てます。作業に入る前に工具と部品の確認をしましょう。

## ■ 工具

ハンダゴテ、ハンダ、ニッパ等、いずれも精密用を用意します。

## ■ 部品

次の部品表と照らし合わせて全ての部品がそろっているか確認して下さい。なお、TK-3687 と TK-3687mini では取り付けの関係で若干部品構成が異なります。

### タイマ&LEDディスプレイ実習キット(B6092)

	部品番号	型名, 規格	メーカー	数量	付属数量	備考
1	■ 共通					
2	U1	TD62783AP TD62784AP UDN-2983A	東芝 " SPRAGUE	1	1	*1
3	Q1,Q2	RN2001	東芝	2	2	
4	LD1~12			12	12	
5	LD13	SLA-9764	SANYO	1	1	
6	BZ	QMX-05 EPM121A0A	スター精密 メガセラ	1	1	*1
7	SW1~3	SKHHAK/AM/DC	ALPS	3	3	*1
8	R1~8	100~150Ω		8	8	
9	R9,10	4.7KΩ		2	0	*2
10	C1	47μF/16V		1	1	
11	C2	0.1μF		1	1	
12	PCB	B6092	東洋リンクス	1	1	
13						
14	■ TK-3687mini版					
15	CN1,2	HIF3FC-30PA-2.54DSA		2	2	*1, 基板に裏付け
16						
17						
18	■ TK-3687版					
19	CN1	HIF3FC-30PA-2.54DSA		1	1	*1, 基板に裏付け
20	CN2	HIF3FC-30PA-2.54DSA		1	0	*2
21	CN16	HIF3FC-30PA-2.54DSA		1	1	*1, TK-3687のCN16に実装
22	接続ケーブル	30芯フラットケーブル		1	1	
23						

(\*1)相当品を使用することがあります。  
(\*2)実装しないため付属していません。

## ■ ハンダ付けに入る前に…

ハンダゴテによる火傷には十分注意して下さい。万一火傷した場合は、すぐに氷か流水で冷やして下さい。

## ■ 部品のハンダ付け

部品の確認が済んだら、いよいよ実装です。実装図、完成写真をよく見ながらハンダ付けを行なって下さい。

IC は 1 番をあわせませす。プリント基板のシルク図の凹んでいる部分を IC の凹みにあわせませす。

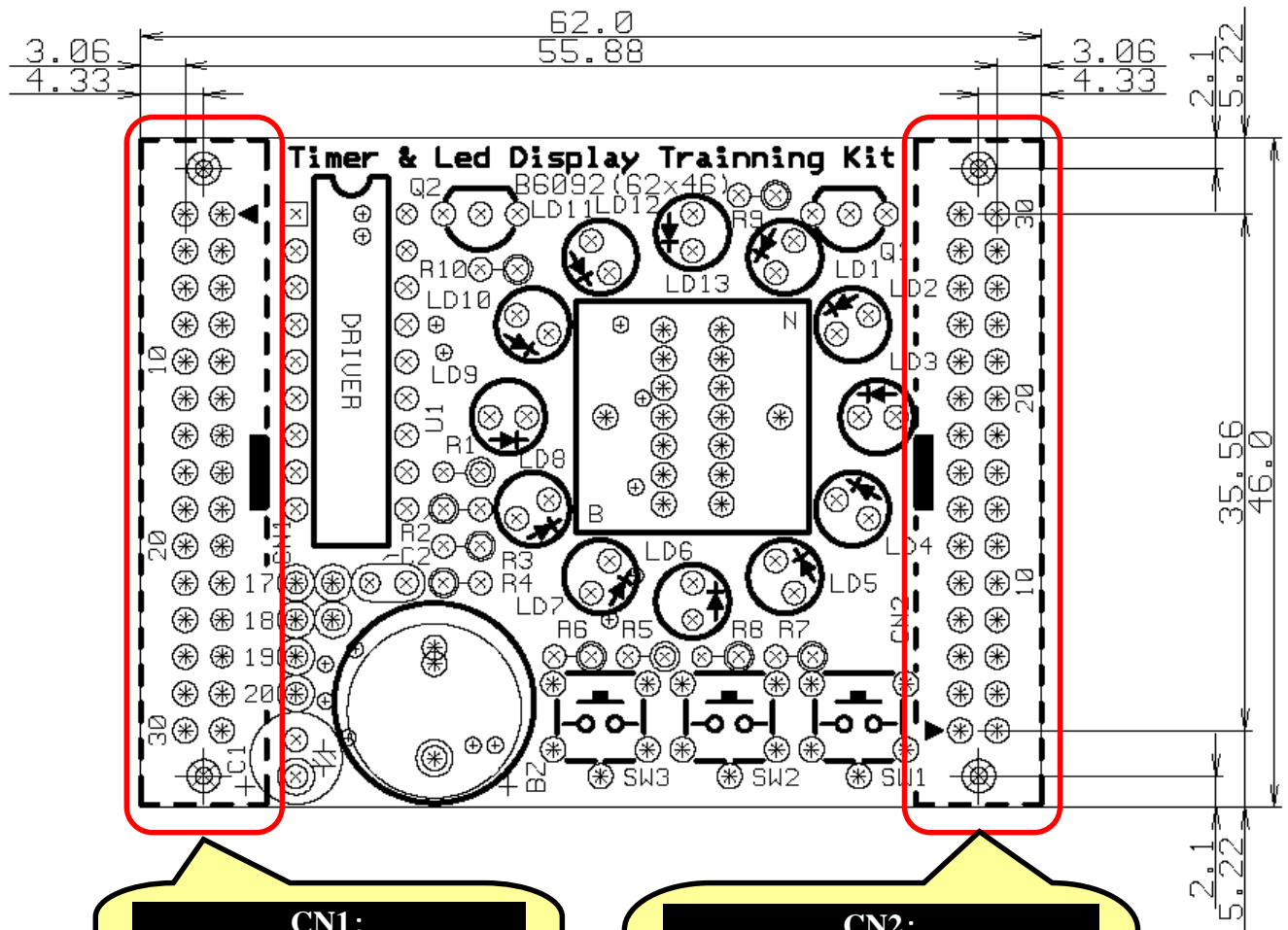
LED は足の長い方がアノードで、反対側がカソードです。右の写真を参考にして実装して下さい。



サウンドと電解コンデンサは+/-の極性があります。部品のマークとシルク図のマークに注意して取り付けましょう。

基本的には背の低い部品から実装し、最後に裏付けする部品をハンダ付けすると楽に作業が進められます。具体的には以下の順番で部品を実装します。

**LD13→U1→R1~8→C2→C1→Q1, 2→SW1~3→BZ→LD1~12→CN1, 2**

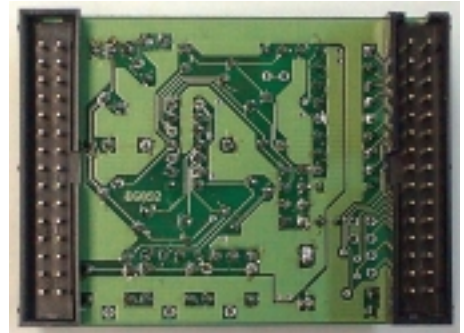
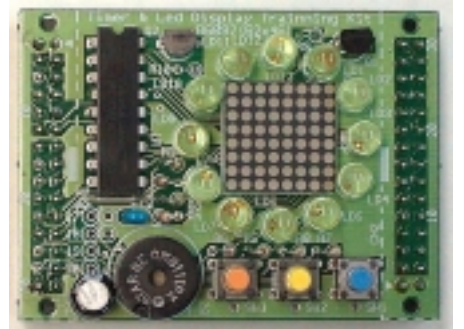
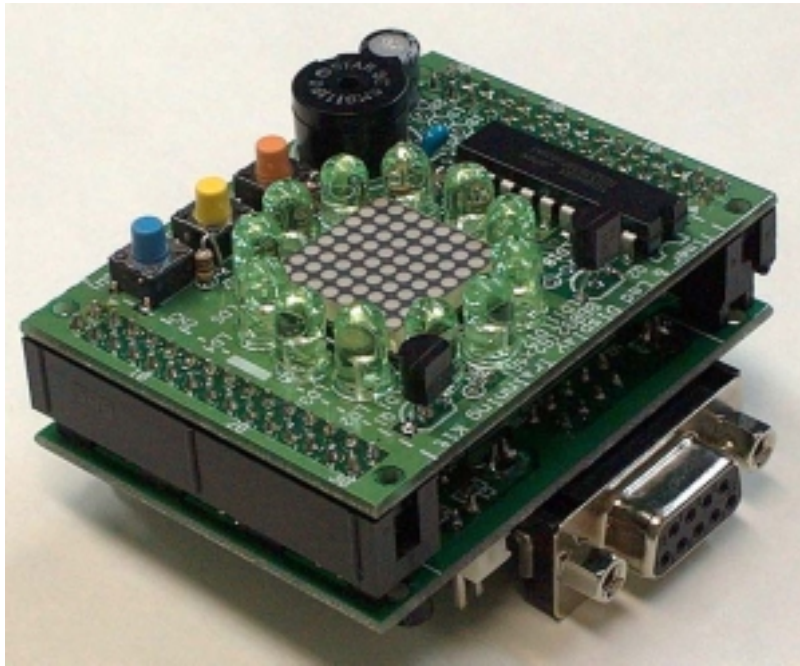


**CN1:**  
基板の裏に中央の切りかきを内側にして取り付けます。

**CN2:**  
TK-3687mini:  
基板の裏に中央の切りかきを内側にして取り付けます。  
TK-3687:  
実装しません。

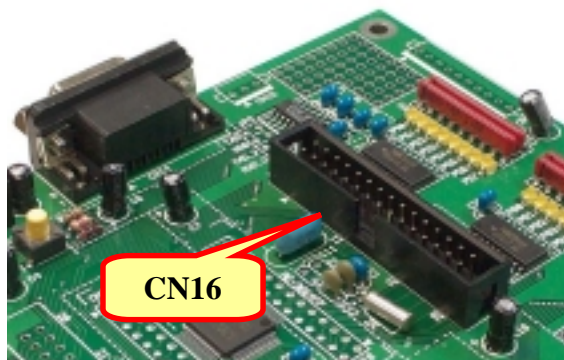
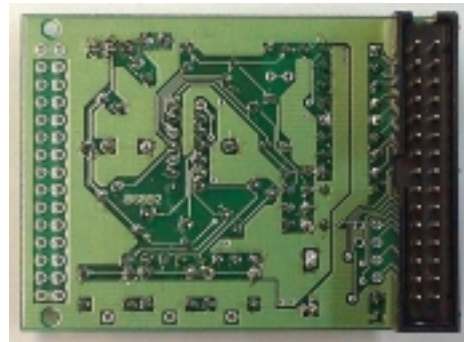
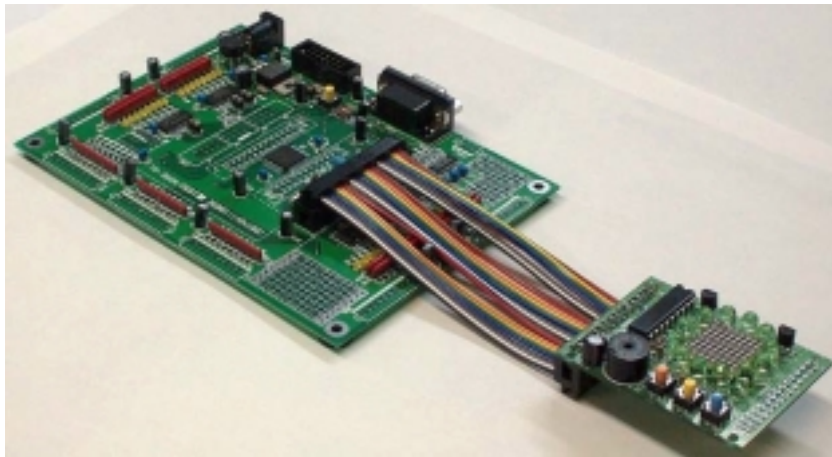
### ■ TK-3687mini 版の完成写真と接続方法

上下を間違えないように接続して下さい。



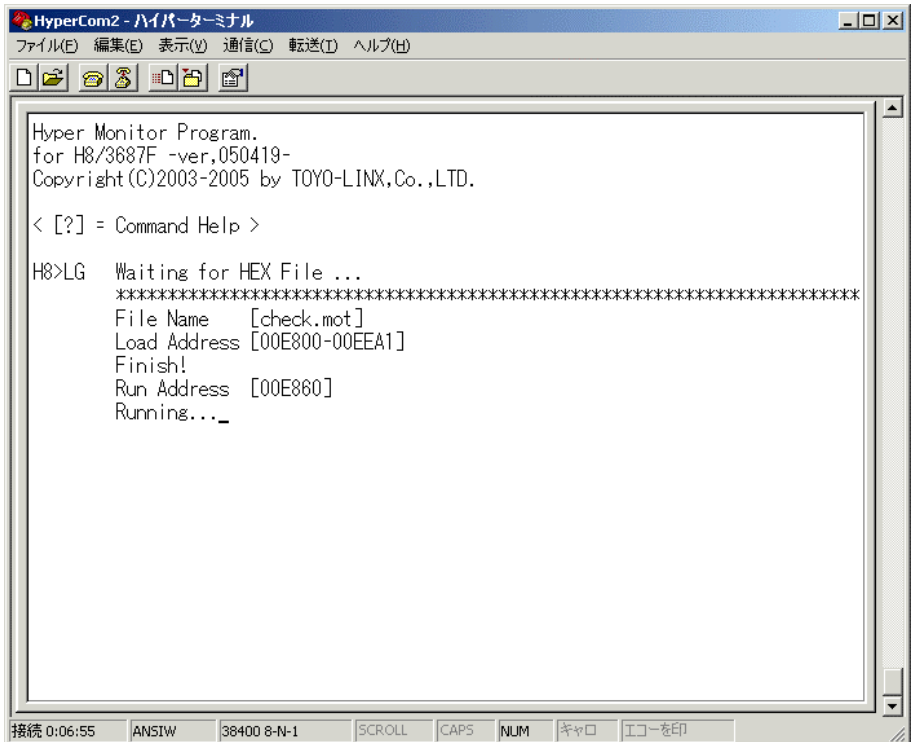
### ■ TK-3687 版の完成写真と接続方法

TK-3687 の CN16 に 30 ピンコネクタをハンダ付けします。30 芯フラットケーブルで CN1 と TK-3687 の CN16 を接続して下さい。



## 2 動作チェック

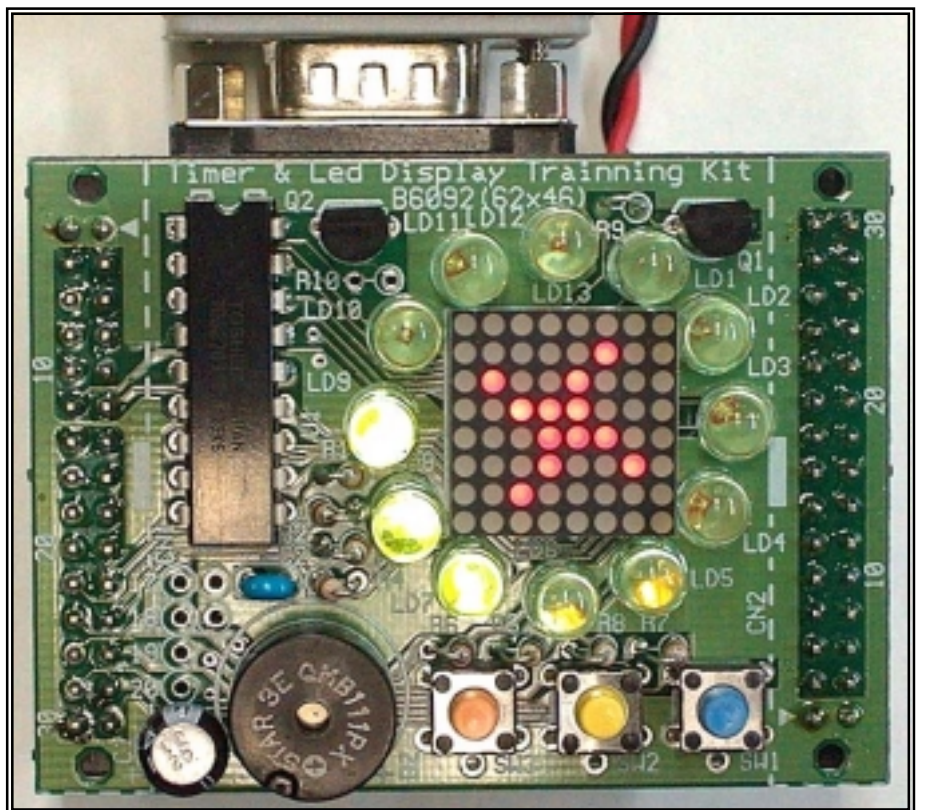
組み立てが終わったらプログラムを動かしてみよう。付属 CD-R 内のチェックプログラムを実行します。ハイパーモニタの「LG」コマンドで「(CD-ROM): ¥TK-3687mini ¥プログラム ¥check.mot」をダウンロードして実行してください(ハイパーモニタの詳しい使い方は「TK-3687 ユーザーズマニュアル」を見て下さい)。なお、ソースファイルは付録をご覧ください。



```
HyperCom2 - ハイパーターミナル
ファイル(F) 編集(E) 表示(V) 通信(C) 転送(D) ヘルプ(H)
[Icons]
Hyper Monitor Program.
for H8/3687F -ver,050419-
Copyright (C)2003-2005 by TOYO-LINK,Co.,LTD.
< [?] = Command Help >
H8>LG Waiting for HEX File ...
*****
File Name [check.mot]
Load Address [00E800-00EEA1]
Finish!
Run Address [00E860]
Running..._
接続 0:06:55 ANSIV 38400 8-N-1 SCROLL CAPS NUM キャロ エコーを印
```

周囲の LED がルーレットのように時計回りに点灯し、中央の LED ディスプレイにアニメーションが表示されます。SW2 か SW3 を押すと、LED ディスプレイの表示パターンが流し文字に切り替わります。もう一度押すとアニメーション表示に戻ります。また、SW1 を押すとメロディが流れます。

うまく動かないときは、もう一度部品のハンダ付けを確かめてください。ちゃんとハンダ付けされていますか。部品の足同士がハンダでブリッジしていませんか。部品の向きはありますか。TK-3687/TK-3687mini との接続方法は正しいでしょうか。動作しない原因のほとんどはハンダ付け不良です。



# 3 Cによるプログラミング

ルネサステクノロジは現在、High-performancr Embedded Wprkshop V. 4(HEW4)に対応した無償評価版コンパイラを公開しています。無償評価版コンパイラは、はじめてコンパイルした日から60日間は製品版と同等の機能と性能のままで試用できます。61日目以降はリンクサイズが64Kバイトまでに制限されますが、H8/3687はもともとアクセスできるメモリサイズが64Kまでバイトなので、この制限は関係ありません。また、無償評価版コンパイラは製品開発では使用できないのですが、H8/300H Tiny シリーズ(H8/3687 も含まれる)では許可されています。この項では無償評価版コンパイラのダウンロードからインストール、プログラムの入力とビルド、ハイパーH8によるダウンロードと実行までを説明します。

## 1. HEW の入手

HEW は株式会社ルネサステクノロジのホームページよりダウンロードします。ダウンロードサイトのURLは以下の通りです。



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

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

ダウンロードサイトの下の方にある「ダウンロードのページへ」をクリックして下さい。次のページで必須事項を入力してダウンロードを開始します。ダウンロード先はデスクトップにすると便利です。全部で 69.4MByte になりますので、ADSL か光回線でない、かなり大変なのが実情です。‘h8cv601r00.exe’というファイルがダウンロードされます。

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

この画面は 2005 年 4 月現在です

バージョン	公開日	更新内容			ダウンロード
		SHC/C++ コンパイラ	H8C/C++ コンパイラ	Tiny/SLP コンパイラ	
		更新一覧(SH)	更新一覧(H8)	更新一覧(Tiny/SLP)	Download
Device Updater V1.02	2004.11.05	更新一覧(SH)	更新一覧(H8)	更新一覧(Tiny/SLP)	-
Device Updater V1.01	2004.08.05	更新一覧(SH)	更新一覧(H8)	更新一覧(Tiny/SLP)	-
Device Updater V1.00	2004.06.21	更新一覧(SH)	更新一覧(H8)	更新一覧(Tiny/SLP)	-

デバイス名	対象コンパイラパッケージ	必要な設定
H8/28086 H8/28076 H8/38602	Tiny/SLPコンパイラ	プロジェクト構築後、ビルド前にCPUオプションを以下の手順より「Tiny」に変更してください。 1. HEWメニュー「オプション」→「H8 Tiny/SLP」を選択してください。 2. CPUダウンロードを選択してください。 3. CPUダウンロードリストの選択欄を、「Tiny」に変更してください。 4. ビルドを行ってください。

デバイスアップデート  
ダウンロードサイト  
[http://www.renesas.com/jpn/products/mpumcu/tool/download2/coding\\_tool/hew/utilities/device\\_updata/index.html](http://www.renesas.com/jpn/products/mpumcu/tool/download2/coding_tool/hew/utilities/device_updata/index.html)

ページの下の方にある「Download」をクリックしてください。ダウンロード先はデスクトップにすると便利です。全部で3.55MByteになります。‘hew\_du104.exe’というファイルがダウンロードされます。

### 最新版の HEW を手に入れましょう

HEW は頻繁にバージョンアップされます。HEW はルネサステクノジのマイコン全てに対応しているため、H8 シリーズはもとより、R8 シリーズや SH シリーズなど、対応するマイコンが増えるとそのたびにマイナーチェンジされるようです。また、その際に報告されていた不具合を一緒に修正することもあります。そのため、このマニュアルの情報もすぐに古くなってしまいが実情です。

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

## 2. HEW のインストール

ダウンロードした ‘h8cv601r00. exe’ をダブルクリックしてください。すると、インストールが始まります。画面の指示に従ってインストールしてください。

次に、無償評価版コンパイラをアップデートします。ダウンロードした ‘hew\_du104. exe’ をダブルクリックしてください。インストールが始まります。画面の指示に従ってインストールしてください。

## 3. メモリマップの確認

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

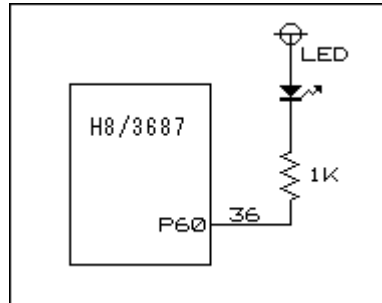
0000 番地	モニタプログラム ‘ハイパーH8’		ROM/フラッシュメモリ (56K バイト)
DFFF 番地			
E000 番地	未使用		未使用
E7FF 番地			
E800 番地	ハイパーH8 ユーザ割り込みベクタ		RAM (2K バイト)
E860 番地	PResetPRG	リセットプログラム	
	PIntPRG	割り込みプログラム	
EA00 番地	P	プログラム領域	
	C	定数領域	
	C\$DSEC	初期化データセクションのアドレス領域	
	C\$BSEC	未初期化データセクションのアドレス領域	
	D	初期化データ領域	
FFFF 番地			
F000 番地	未使用		未使用
F6FF 番地			
F700 番地	I/O レジスタ		I/O レジスタ
F77F 番地			
F780 番地	B	未初期化データ領域 初期化データ領域 (変数領域)	RAM (1K バイト) フラッシュメモリ書換え用 ワークエリアのため、 FDT と E7 使用時は、 ユーザ使用不可
	R		
FB7F 番地			
FB80 番地			
FD80 番地	S	スタック領域	RAM (1K バイト)
FDFE 番地			
FE00 番地	ハイパーH8 ワークエリア		
FF7F 番地			
FF80 番地	I/O レジスタ		I/O レジスタ
FFFF 番地			

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



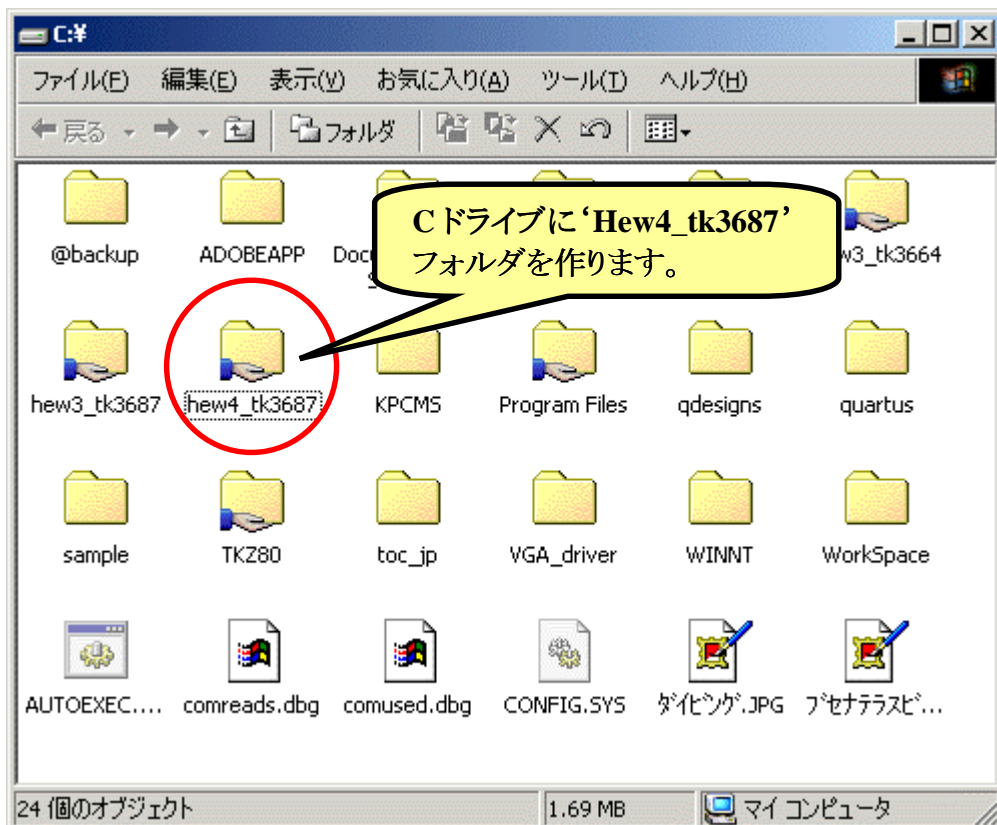
#### 4. プロジェクトの作成

ここで作るプログラムは P60 につないだ LED を点滅させるというものです。回路図は次のとおりです。



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

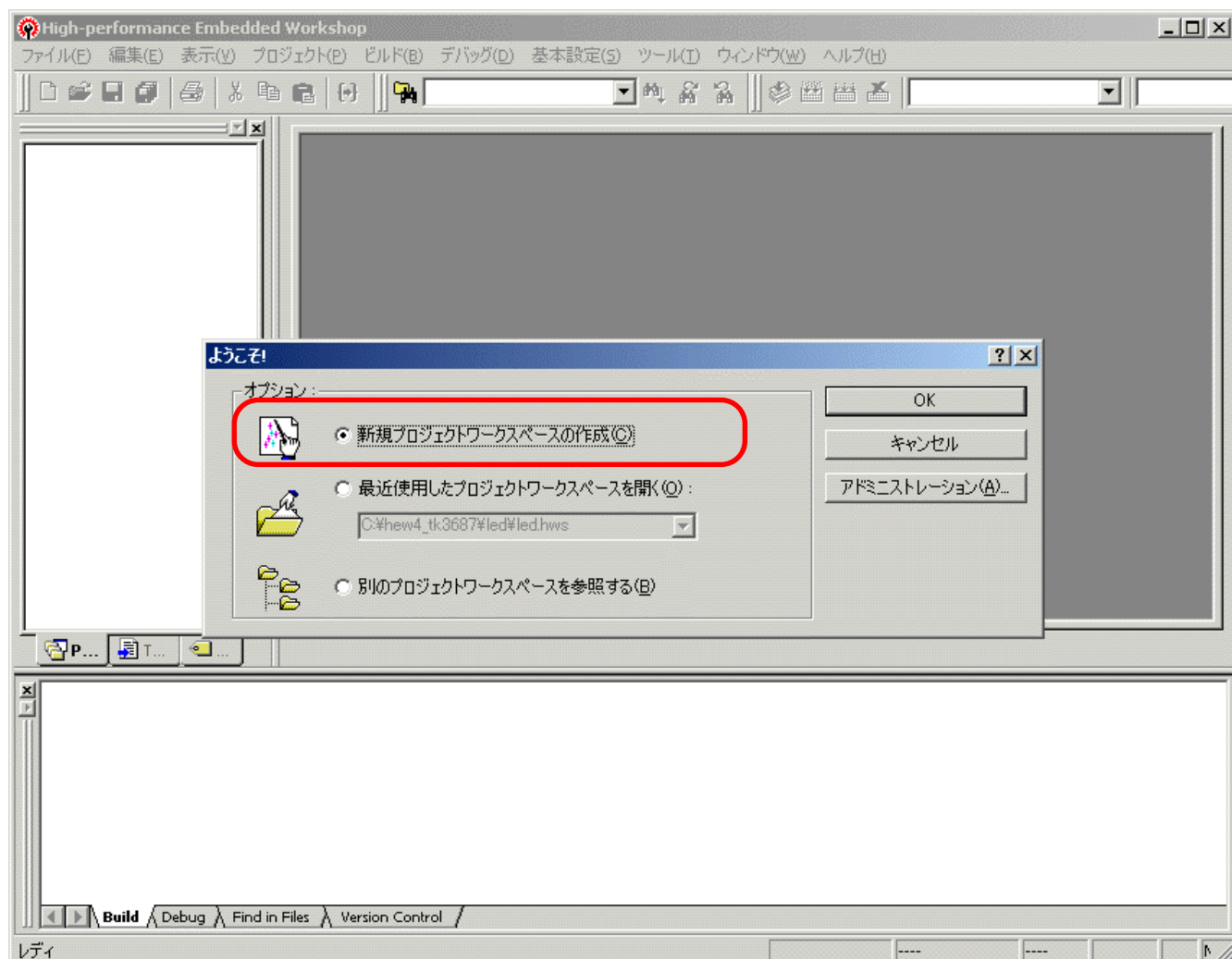
しかしその前に、HEW 専用作業フォルダを作っておきましょう。Cドライブに‘Hew4\_tk3687’を作ってください。このマニュアルのプロジェクトは全てこのフォルダに作成します。



では、HEW を起動しましょう。スタートメニューから起動します。



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



### 前に作ったプロジェクトを使うとき

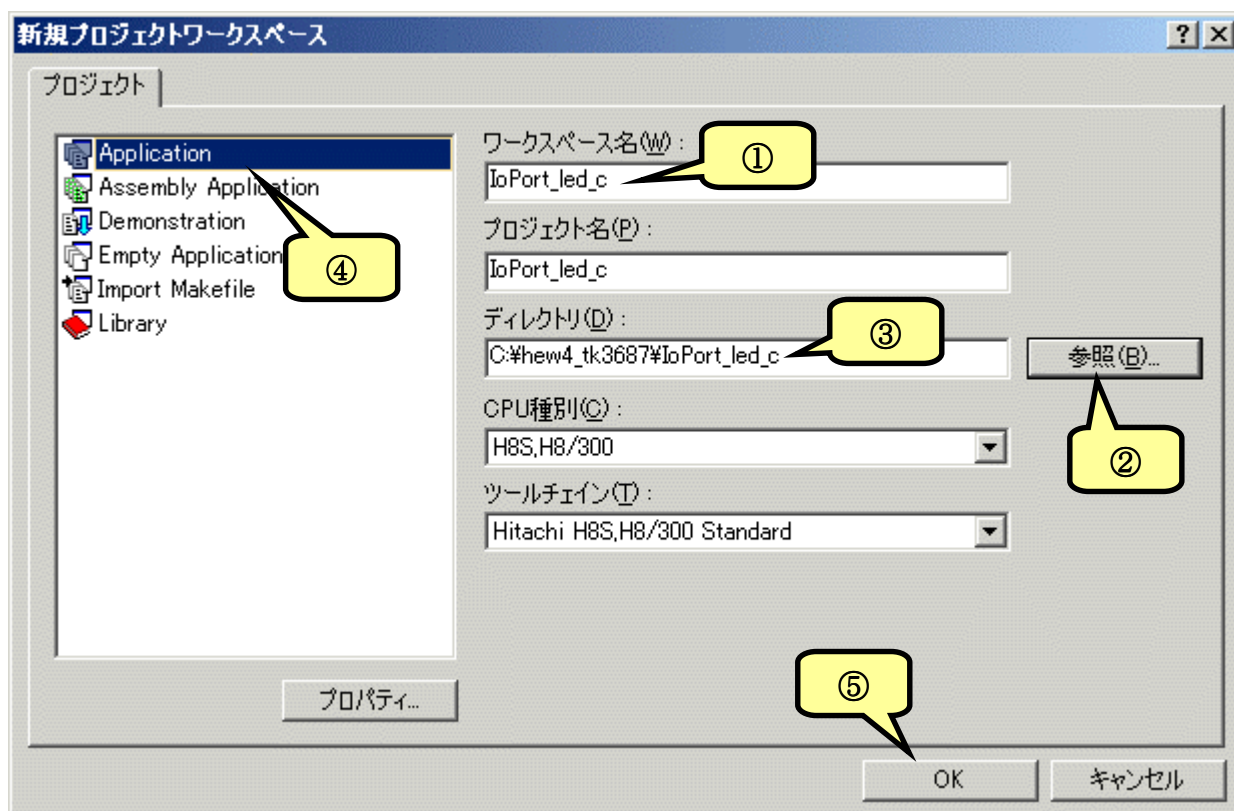
その場合は、「ようこそ!」ダイアログで「最近使用したプロジェクトワークスペースを開く」を選択して‘OK’をクリックします。そのプロジェクトの最後に保存した状態で HEW が起動します。

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

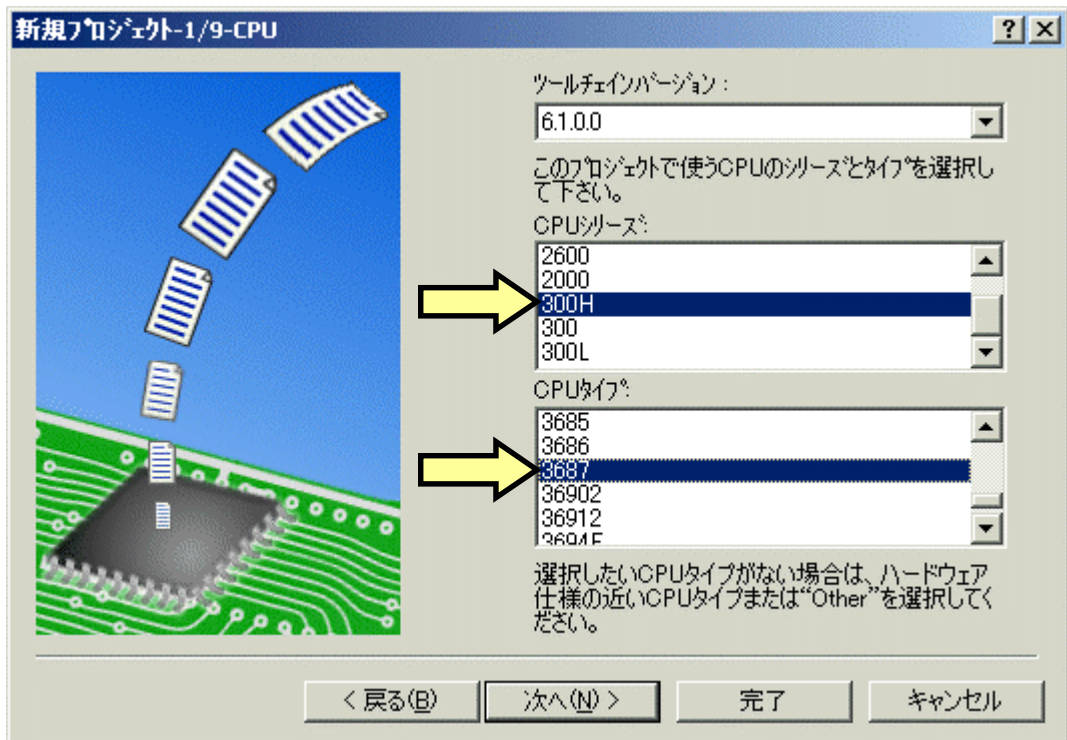
ワークスペースの場所を指定します。②右の「参照 (B) ...」ボタンをクリックします。そして、あらかじめ用意した HEW 専用作業フォルダ(ここでは Hew4\_tk3687)を指定します。設定後、「ディレクトリ (D)」が正しいか確認して下さい。(③)

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

入力が終わったら⑤「OK」をクリックして下さい。



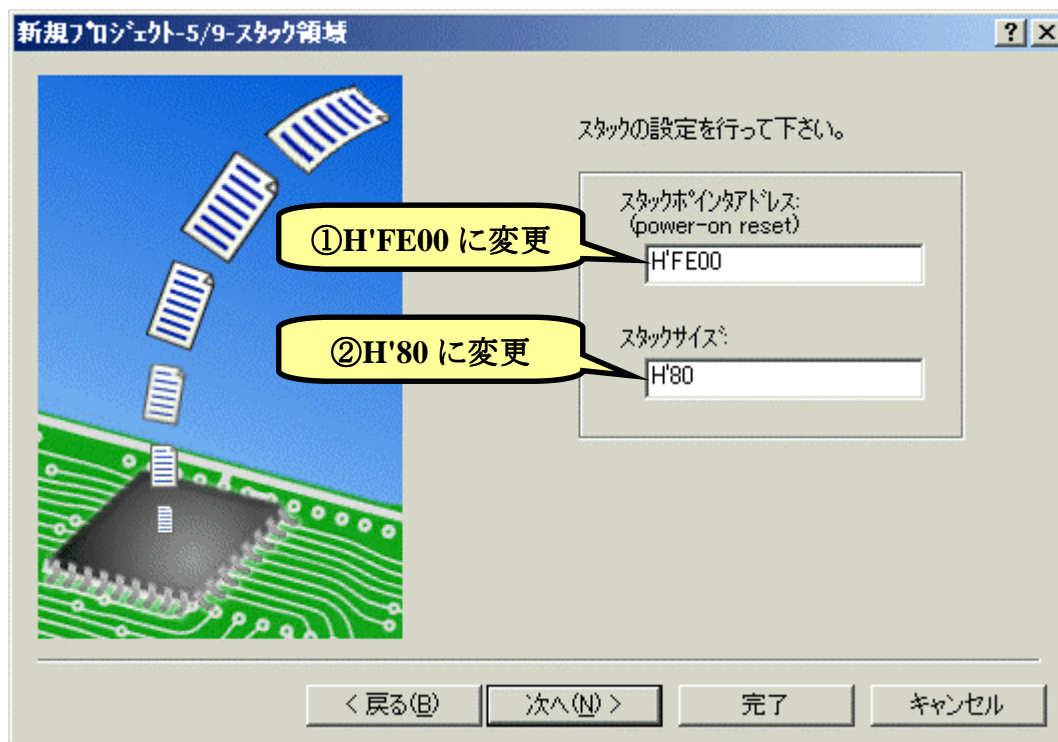
「新規プロジェクト-1/9-CPU」で、使用する CPU シリーズ(300H)と、CPU タイプ(3687)を設定し、「次へ(N) >」をクリックします。



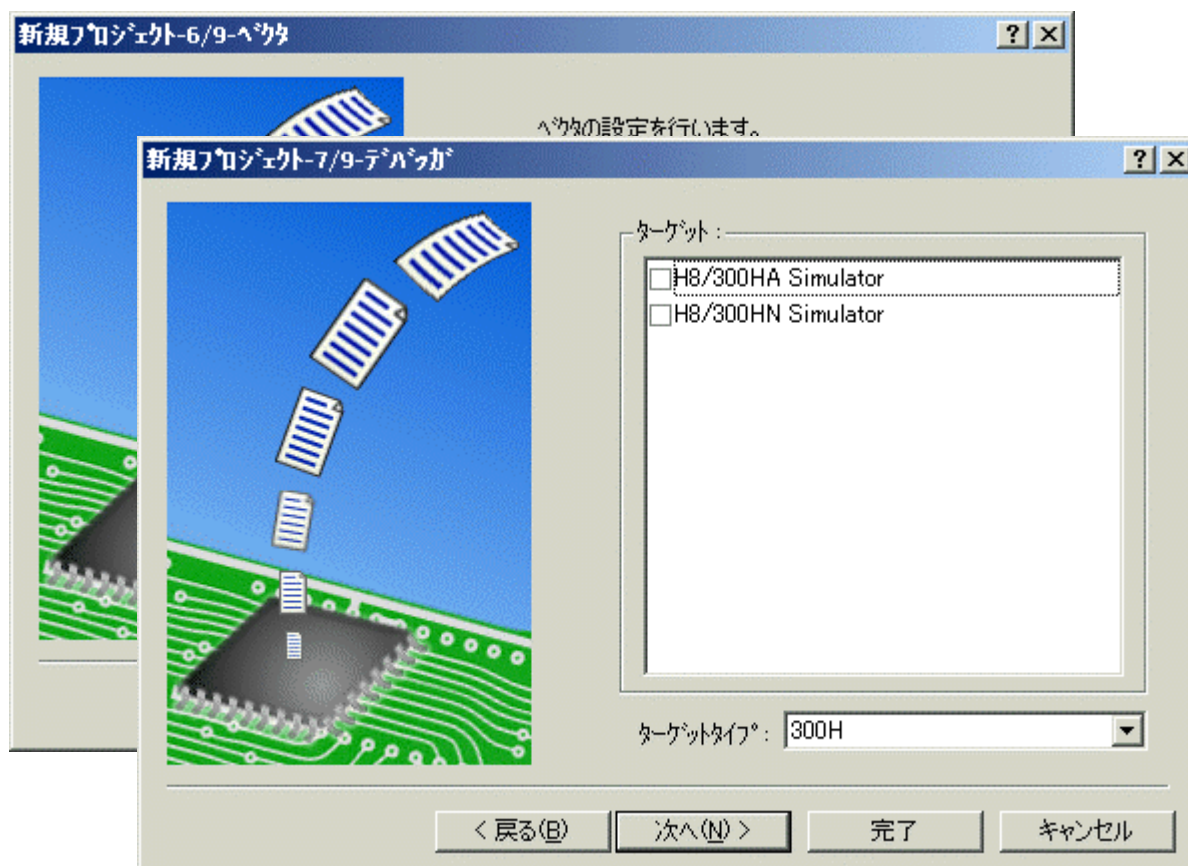
「新規プロジェクト-2/9-オプション」、「新規プロジェクト-3/9-生成ファイル」、「新規プロジェクト-4/9-標準ライブラリ」は変更しません。「次へ(N) >」をクリックして次の画面に進みます。



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



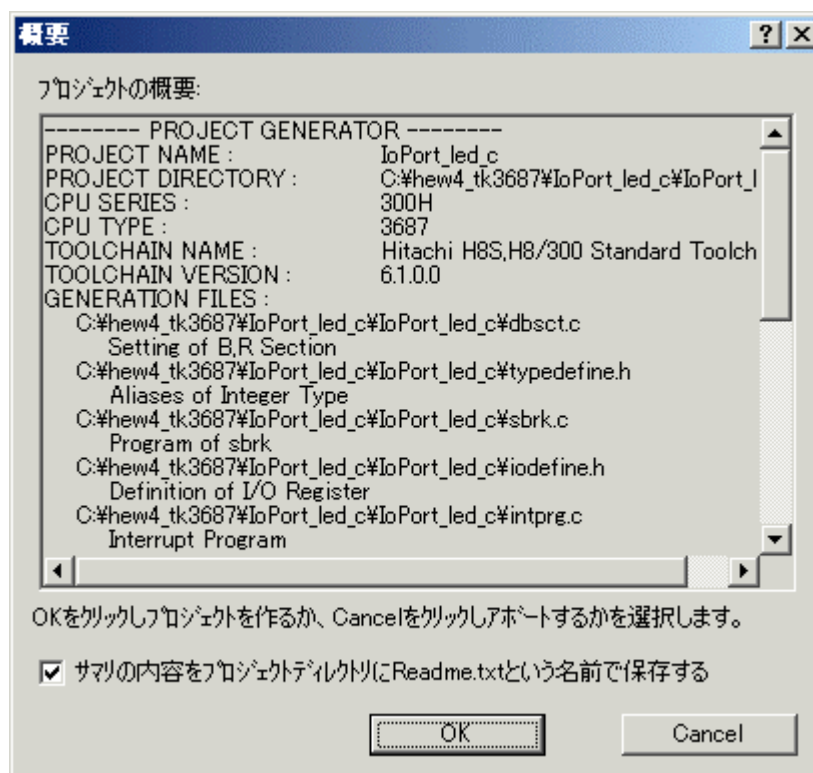
「新規プロジェクト-6/9-ベクタ」、「新規プロジェクト-7/9-デバッガ」は変更しません。「次へ(N) >」をクリックして順に次の画面に進みます。



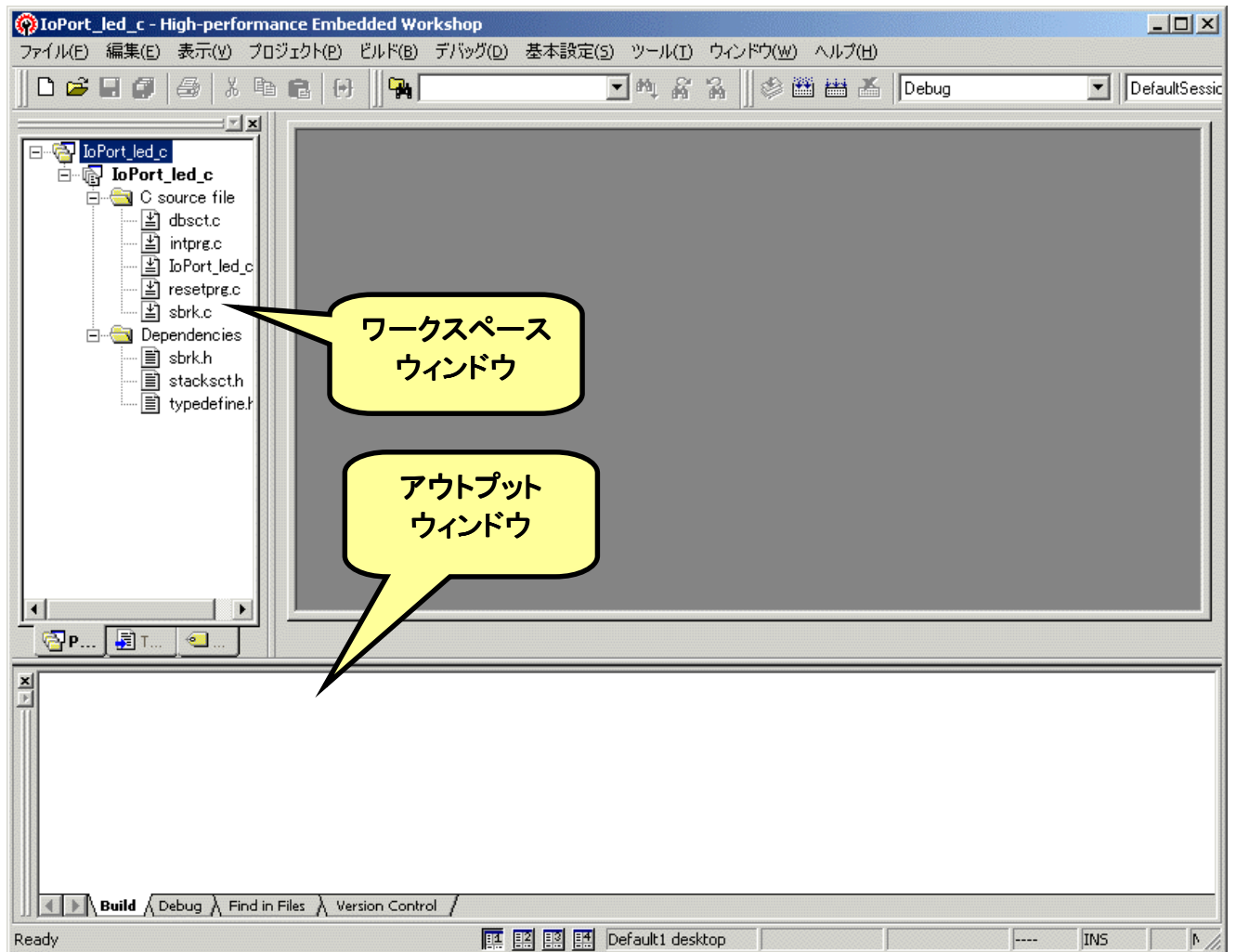
次は「新規プロジェクト-9/9-生成ファイル名」です。ここも変更しません。「完了」をクリックします。



すると、「概要」が表示されるので「OK」をクリックします。

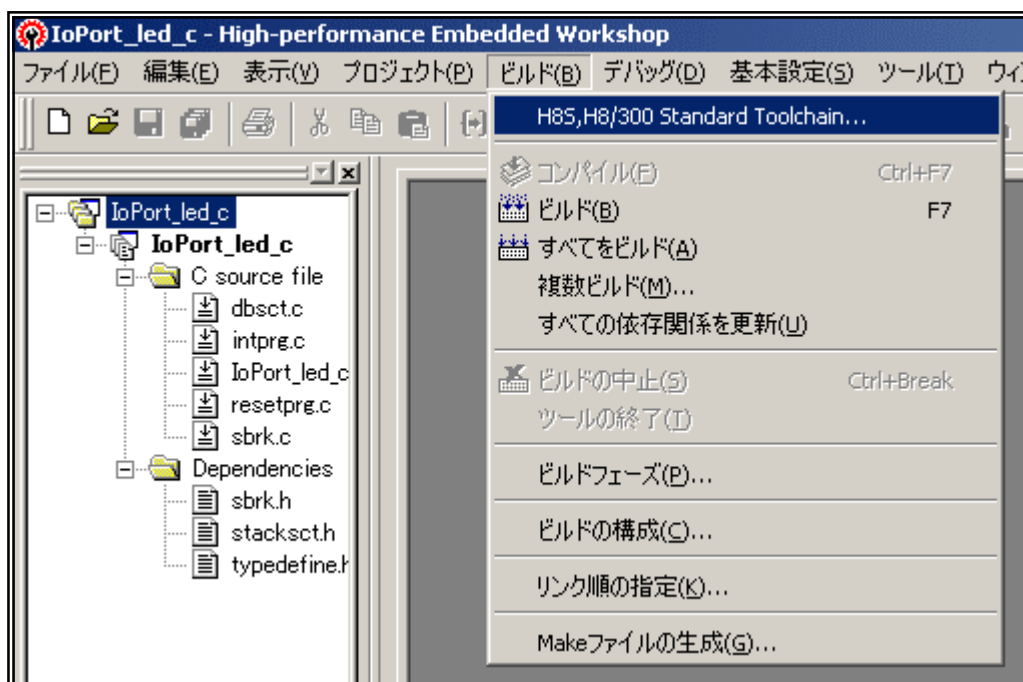


これで、プロジェクトワークスペースが完成します。HEW はプロジェクトに必要なファイルを自動生成し、それらのファイルは左端のワークスペースウィンドウに一覧表示されます。

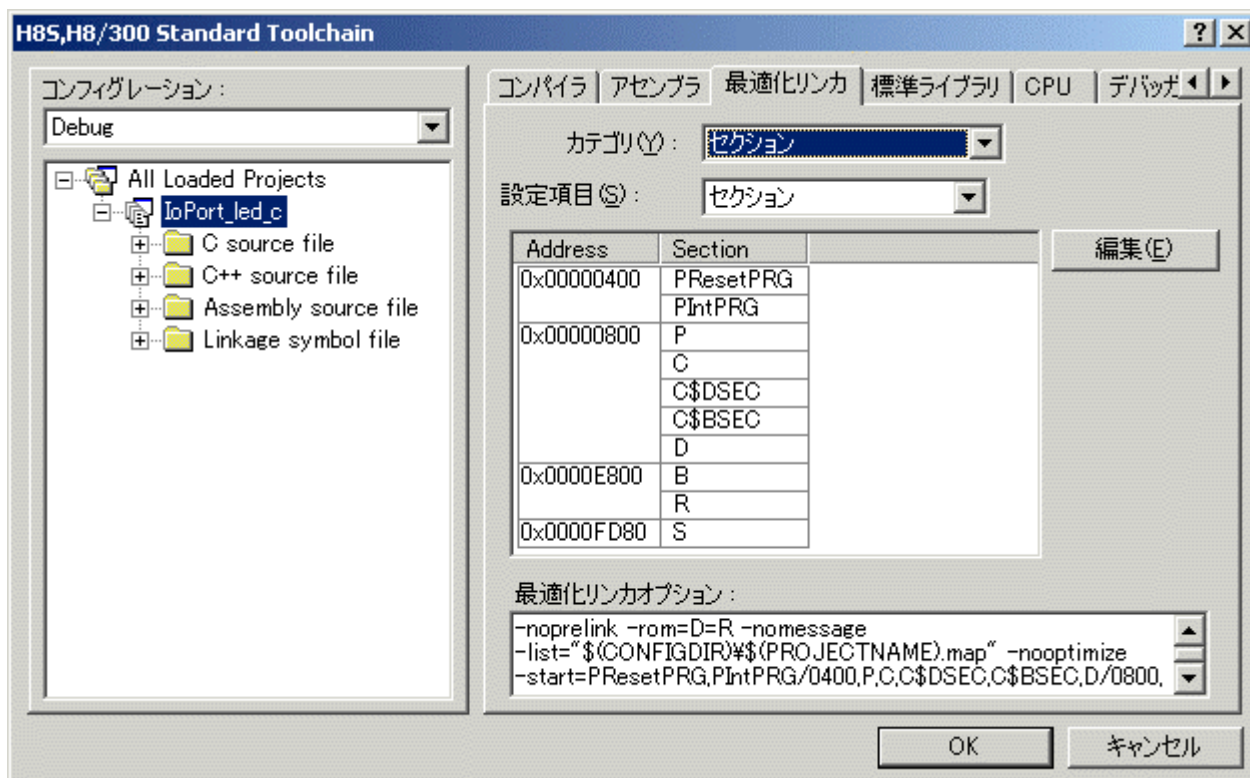


さて、これでプロジェクトは完成したのですが、ハイパーH8 を使うためにセクションを変更してプログラムが RAM 上にできるようにします。(当然ながら、ハイパーH8 を使わないときは変更する必要はなく、そのまま OK。)

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

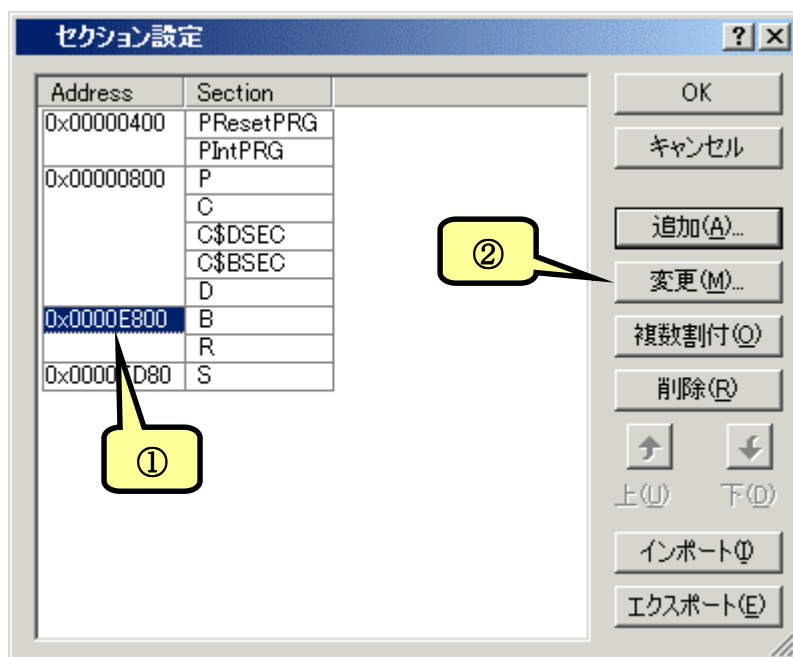


すると、「H8S, H8/300 Standard Toolchain」ウィンドウが開きます。「最適化リンカ」のタブを選び、「カテゴリ(Y)」のドロップダウンメニューの中から「セクション」を選択します。すると、下図のような各セクションの先頭アドレスを設定する画面になります。「編集(E)」ボタンをクリックしてください。

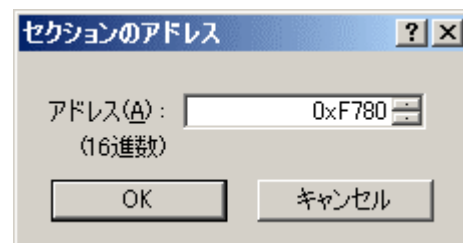




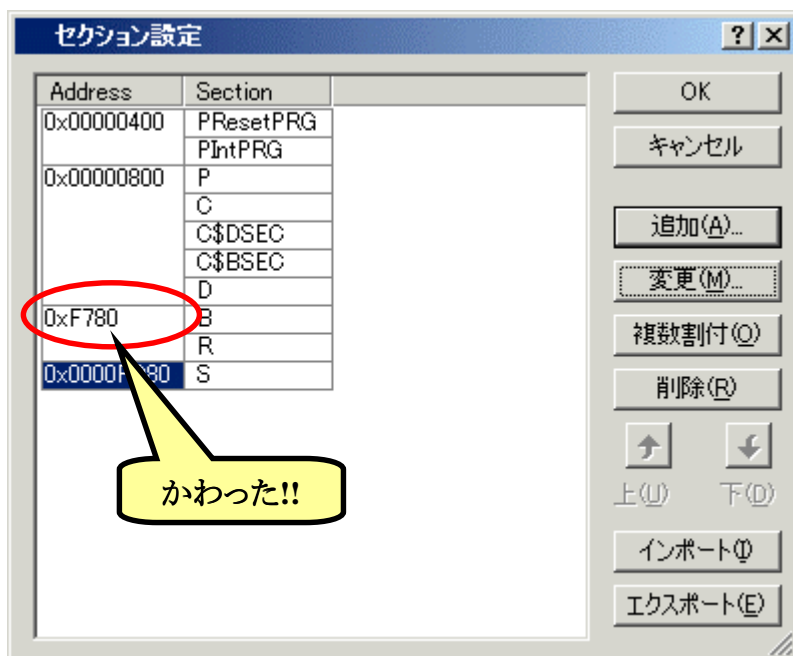
「セクション設定」ダイアログが開きます。それでは、「1. メモリマップの確認」で調べたメモリマップにあわせて設定していきましょう。最初に「B」Section のアドレスを変更します。デフォルトでは E800 番地になっていますね。①「0x0000E800」というところをクリックして下さい。それから、②「変更(M)...」をクリックします。



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



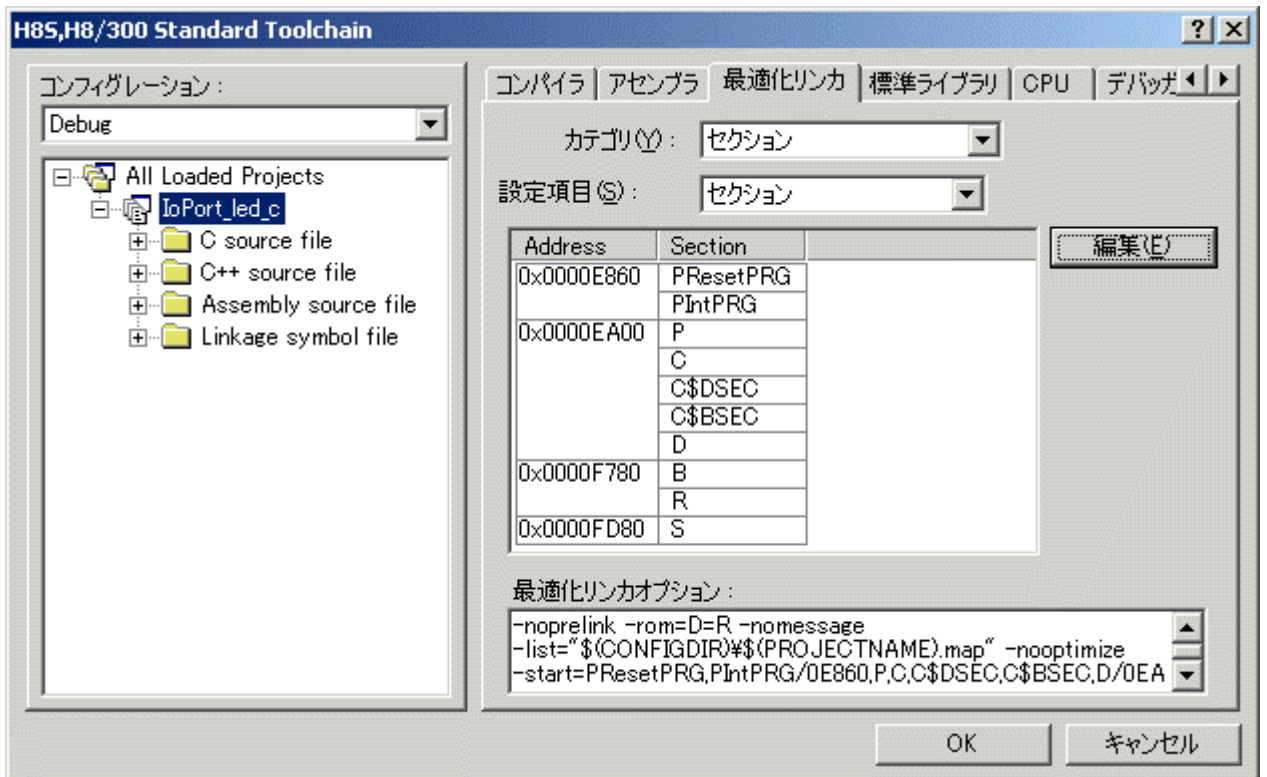
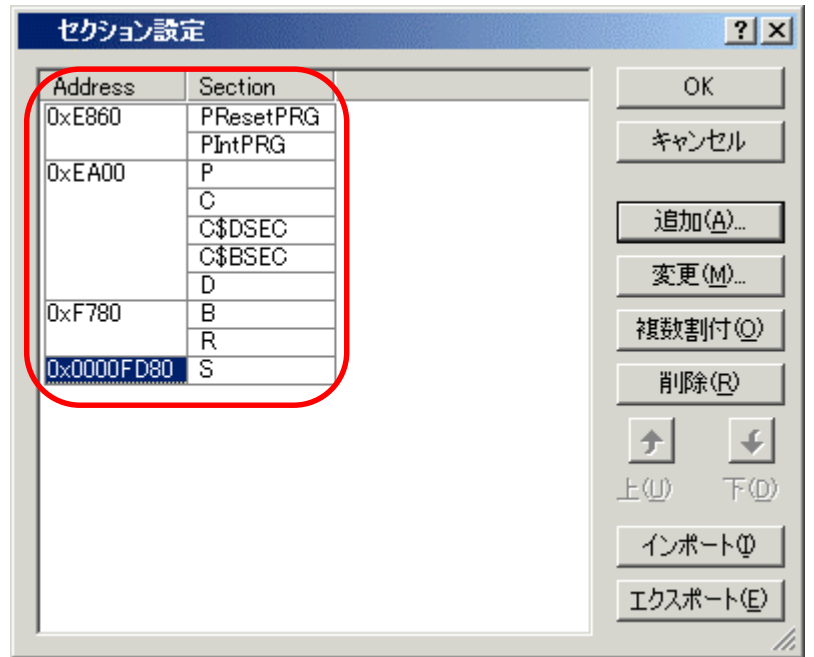
すると…



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

### セクション設定の保存

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



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

## 5. プログラムの入力

---

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

```
/*  
*****  
/* FILE      :IoPort_led.c  
/* DATE      :Wed, Apr 20, 2005  
/* DESCRIPTION :Main Program  
/* CPU TYPE   :H8/3687  
/*  
/* This file is generated by Renesas Project Generator (Ver.4.0).  
/*  
*****  
*/  
  
#ifdef __cplusplus  
extern "C" {  
void abort(void);  
#endif  
void main(void);  
#ifdef __cplusplus  
}  
#endif  
  
void main(void)  
{  
  
}  
  
#ifdef __cplusplus  
void abort(void)  
{  
  
}  
#endif
```

このファイルに追加・修正していきます。下記のリストのとおり入力してみてください。なお、C 言語の文法については、HEW をインストールしたときに一緒にコピーされる「H8S, H8/300 シリーズ C/C++コンパイラ, アセンブラ, 最適化リンカージェディタ 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
/* DATE      :Wed, Apr 20, 2005
/* DESCRIPTION:Main Program
/* CPU TYPE  :H8/3687
/*
/* This file is programed by TOYO-LINX Co.,Ltd. / yKikuchi
/*
/*
/*****
          インクルードファイル
*****/
#include <machine.h> // H8特有の命令を使う
#include "iodefine.h" // 内蔵I/Oのラベル定義

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

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

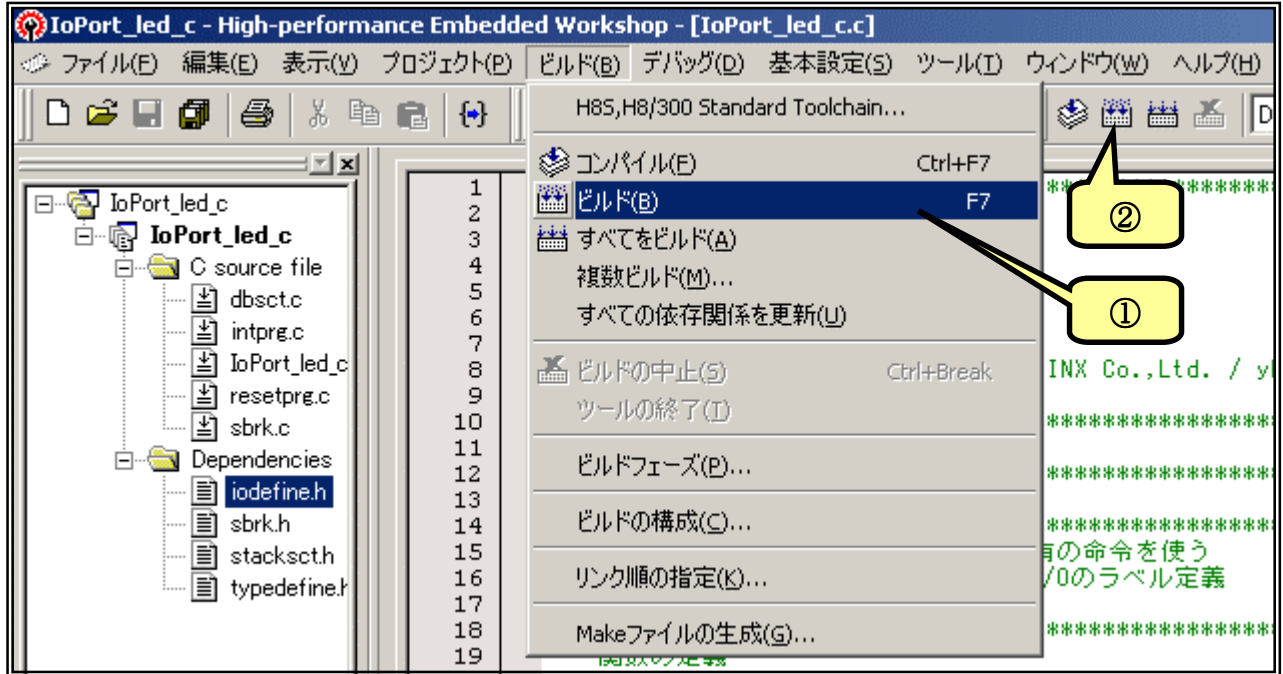
    while(1){
        IO.PDR6.BIT.B0 = 0; // LEDオン
        wait();
        IO.PDR6.BIT.B0 = 1; // LEDオフ
        wait();
    }
}

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

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

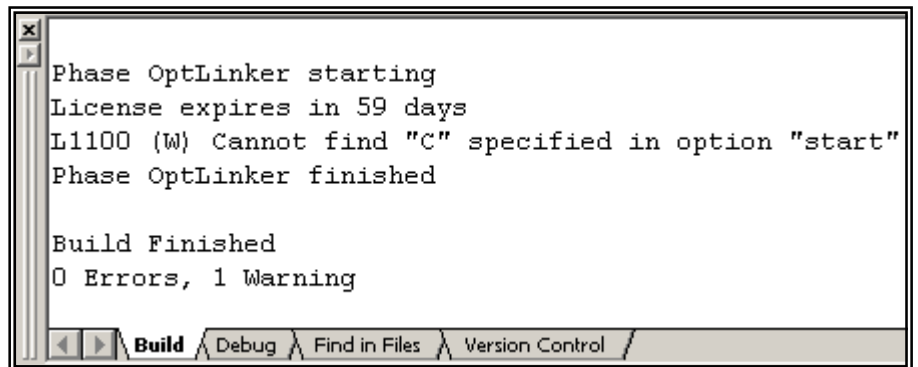
## 6. ビルド!!

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



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

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

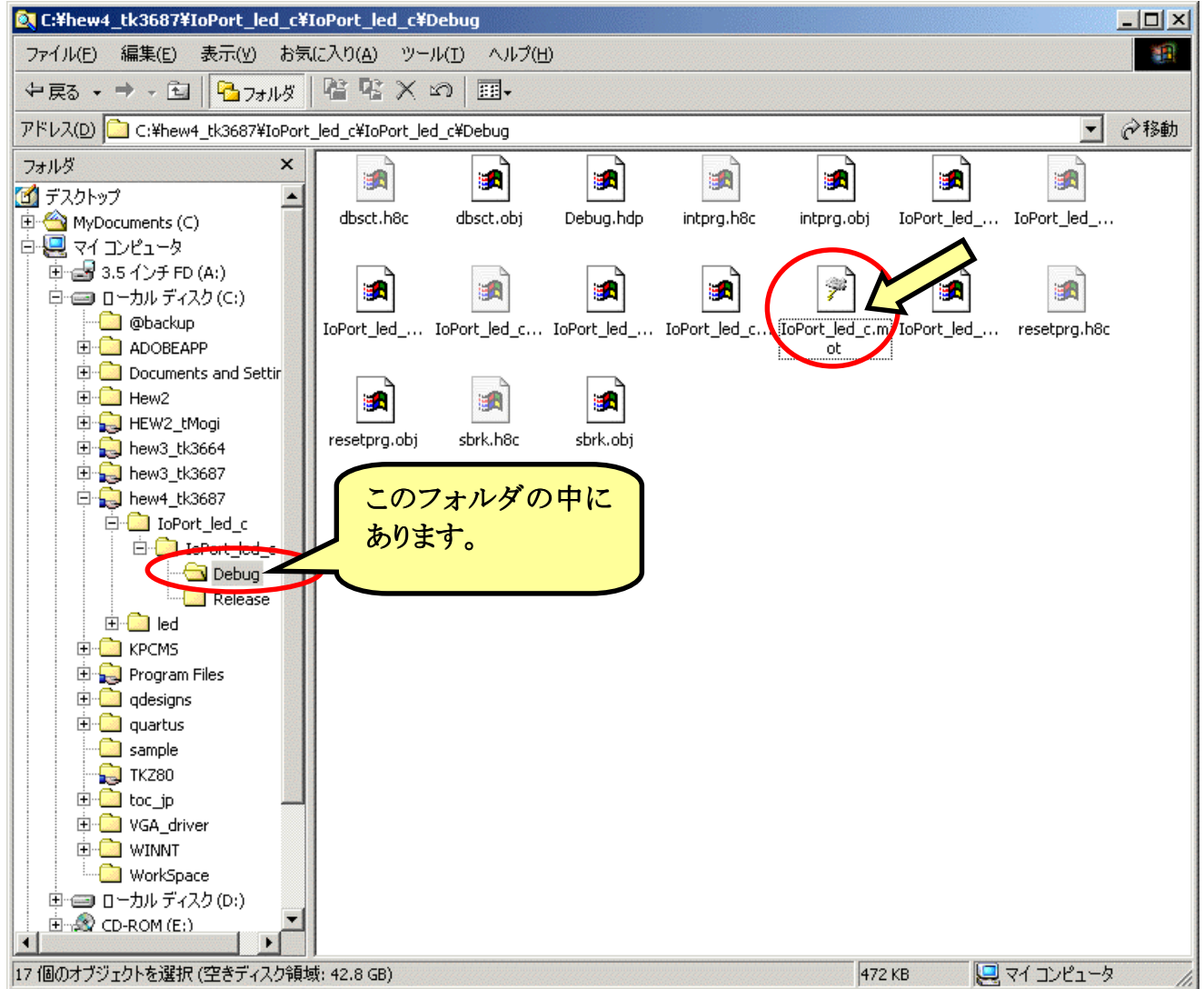


さて、図では「1 Warning」と表示されています。これは「まちがいではないかもしれないけど、念のため確認してね」という警告表示です。例えばこの図の「L1100(W) Cannot find "C" specifind in option "start"」は、Cセクションを設定したのにCセクションのデータがないとき表示されます。今回のプログラムではCセクションは使っていないので、この警告が出てても何も問題ありません。

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

## 7. ダウンロードと実行

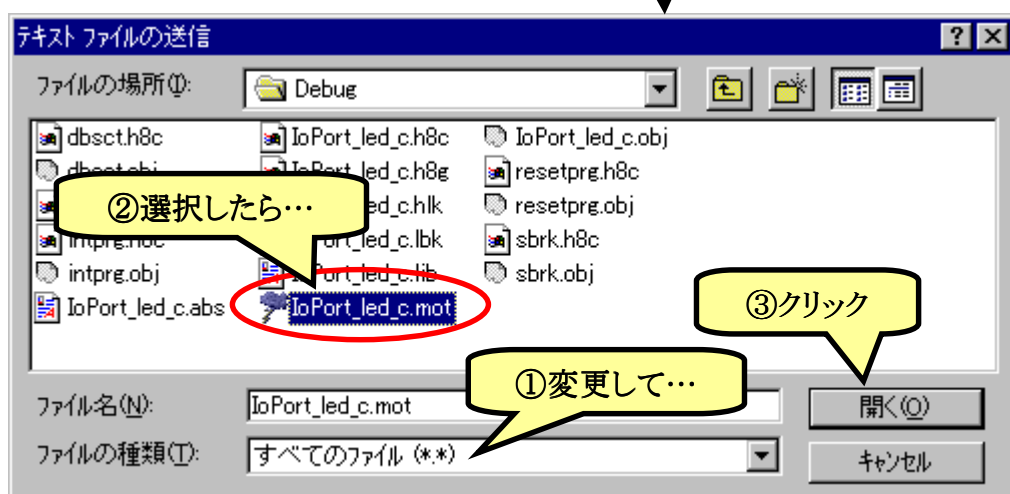
アセンブルすると‘IoPort\_led\_c. mot’というファイルが作られます。拡張子が‘. mot’のファイルは「S タイプファイル」と呼ばれていて、マシン語の情報が含まれているファイルです。このファイルは次のフォルダ内に作られます。



それでは実行してみましょう。ハイパーH8 を起動して下さい。パソコンのキーボードから‘LG’と入力して‘Enter’キーを押します。



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



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



```
38400bps - ハイパーターミナル
ファイル(F) 編集(E) 表示(V) 通信(C) 転送(T) ヘルプ(H)

Hyper Monitor Program.
for H8/3687F -ver,040809-
Copyright (C)2003-2004 by TOYO-LINX,Co.,LTD.

< [?] = Command Help >

H8>LG  Waiting for HEX File ...
*****
File Name   [IoPort_1.mot]
Load Address [00E800-00EA9D]
Finish!
Run Address  [00E860]
Running..._
```

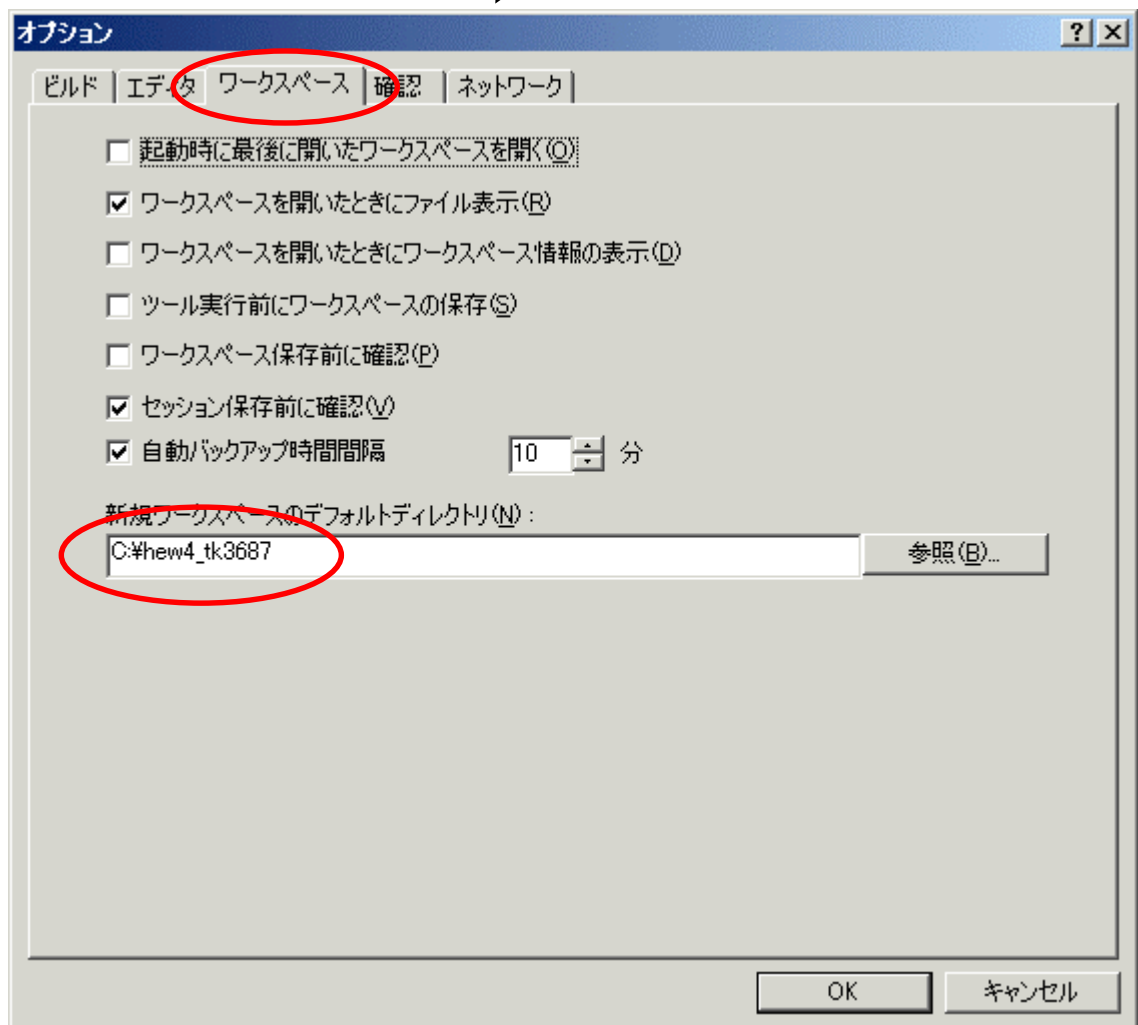
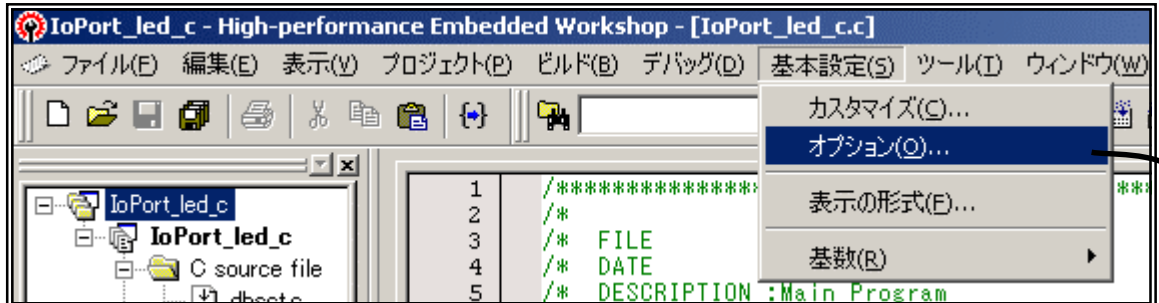
いかがでしょうか。「タイマ&LED ディスプレイ」で実行すると, LED ディスプレイの一番上の行のLEDが点滅します。うまく動作しないときはプログラムの入力ミスの可能性が大です。もう一度ちゃんと入力しているか確認してみてください。



## 8. 便利な設定

### ■ ワークスペースの指定

ワークスペースのデフォルトディレクトリを指定することができます。一回指定すると二回目以降はこのディレクトリがデフォルトになります。

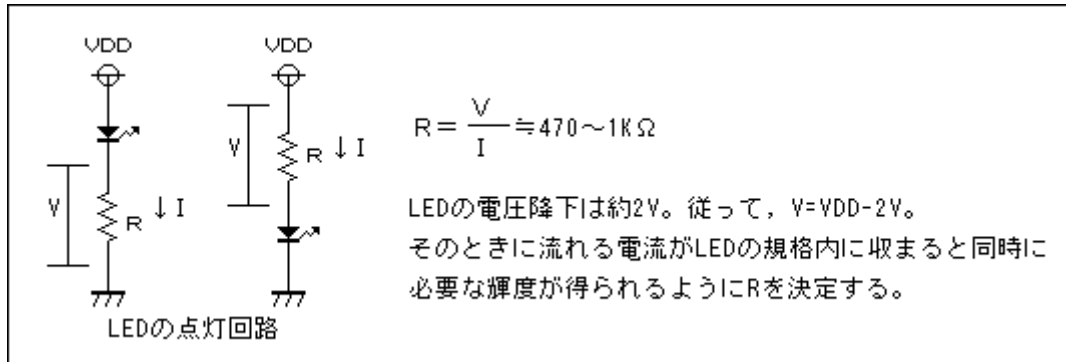


## 4 LED のダイナミック点灯

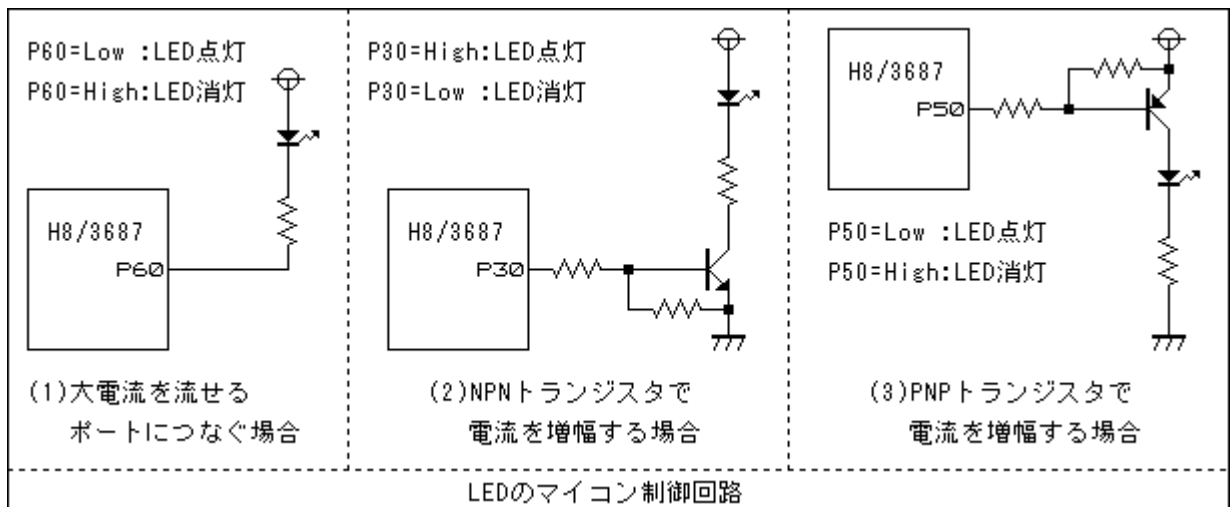
チェックプログラムでは、たくさんの LED を制御して文字やアニメーションを表示しています。たくさんの LED を点灯することには、定番ともいえる一つのテクニックが関係しています。この章ではそのテクニックを説明します。

### ■ LED の基本的な点灯方法

次のように電源と抵抗と LED 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、これだと光りっぱなしで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。



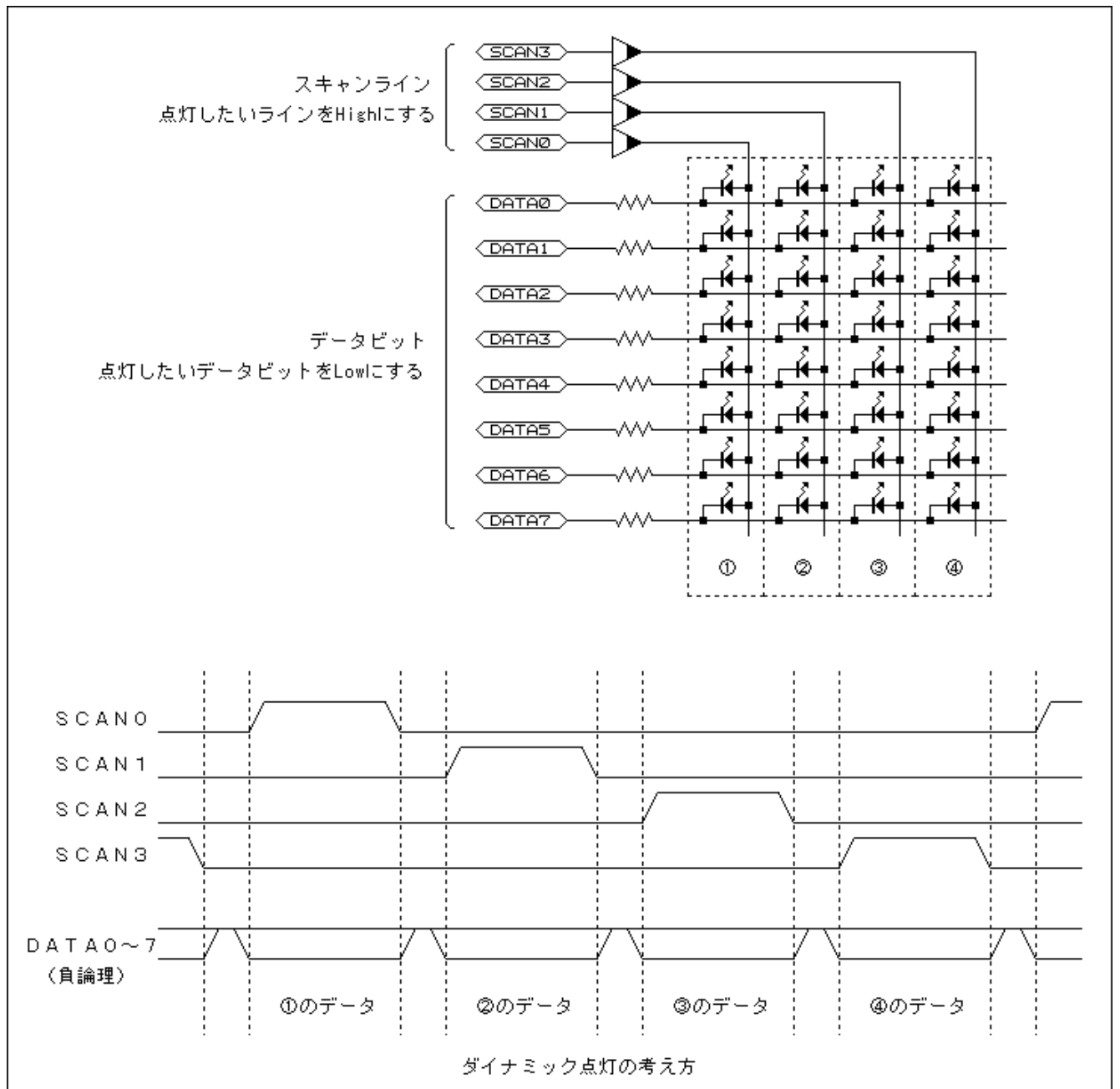
(1) の場合、P60 を Low にすると LED 点灯，High にすると LED 消灯になります。TK-3687/TK-3687mini で使用しているワンチップマイコン H8/3687 の P60～67 は、他のポートと異なり Low レベル出力のとき 20mA まで流すことができますので、LED の点灯など比較的大きな電流を流す必要があるときに使うことができます。大きな電流を流せないポートを使うときは(2)や(3)のようにトランジスタを使って電流を増幅するか、ドライバ IC で電流を増幅します。

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。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。

## ■ LED のダイナミック点灯

ダイナミック点灯というのは、一言でいえば人間の目の錯覚を利用して LED が点灯しているように見せかける方法です。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。

もちろん、瞬間々を見れば最大で 8 個の LED しか点灯していませんが、人間の目には残像現象という性質があるため、LED が消えてもすぐにはわかりません。で、わからないうちにもう一度同じ LED を点灯すると、その LED が消えたと感じないわけです。

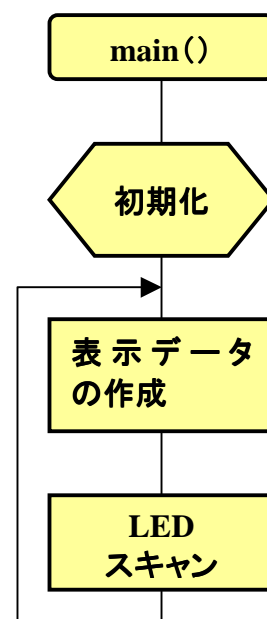
①→②→③→④→①…という切り替えを人間の目で分からないくらいの速さで行なえば、全ての LED が同時に点灯しているように見せかけることができます。

付録最後の回路図を見てください。回路図を見ると P30～P37, P50, P51 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。最初なので、このうち P50 と P51 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‘TimingCnt’をメインループ 1 周ごとに+1 し、表示データのローテートの間隔と、ダイナミックスキャンのタイミングを作っています。P50 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 P60～67 に負論理で出力します。同じように P51 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 P60～67 に負論理で出力します。‘TimingCnt’で得られるタイミングで、P50=Low/P51=High と P50=High/P51=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 個ず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、‘TimingCnt’で得られる間隔でローテートします。

ハイパーモニタの「LG」コマンドで「(CD-ROM): ¥TK-3687mini ¥お ション ¥伊\_LED ¥フ ク ラム ¥kaiten\_01.mot」をダウンロードして実行してください。



### ソースファイル(kaiten\_01.c)

```

/*****
/*
/* FILE      :kaiten_01.c
/* DATE      :Mon, Aug 01, 2005
/* DESCRIPTION :Main Program
/* CPU TYPE  :H8/3687
/*
/* This file is programed by TOYO-LINX Co.,Ltd. / yKikuchi
/*
*****/

*****

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

*****

    定数の定義（直接指定）
*****
//LED表示 -----
#define      DRV_LOGIC 0x300 //ドライバの入力論理
                //負論理入力のビットを ' 1 ' にする

*****

    グローバル変数の定義とイニシャライズ(RAM)
*****
unsigned int  TimingCnt      = 0; //タイミングカウンタ
unsigned int  DisplayData    = 0x0200; //表示データ
unsigned int  ScanFlag       = 0; //スキャンフラグ
unsigned int  RotateFlag     = 0; //ローテートフラグ

*****

    関数の定義

```

```

*****/
void      init_io(void);
void      main(void);

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

    // メインループ -----
    while(1){
        //表示データ作成(ローテート)
        if ((TimingCnt&0x8000)!=RotateFlag){
            RotateFlag = TimingCnt & 0x8000;
            DisplayData= DisplayData << 1;
            if ((DisplayData & 0x1000)!=0) {DisplayData = (DisplayData | 0x0001) & 0x0fff;}
        }

        //LEDスキャン
        if ((TimingCnt&0x0100)!=ScanFlag){
            ScanFlag = TimingCnt & 0x0100;
            IO.PDR5.BYTE = 0x07; //消灯,スキャンデータ
            IO.PDR6.BYTE = 0xff; //消灯,表示データ
            switch (ScanFlag){
                case 0x0000:
                    IO.PDR6.BYTE = ~(unsigned char)(DisplayData & 0x00ff); //表示データセット
                    IO.PDR5.BYTE = 0xfe; //スキャンデータセット
                    break;
                case 0x0100:
                    IO.PDR6.BYTE = ~(unsigned char)(DisplayData / 0x0100); //表示データセット
                    IO.PDR5.BYTE = 0xfd; //スキャンデータセット
                    break;
            }
        }

        //タイミングカウンタ+1
        TimingCnt++;
    }
}

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

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

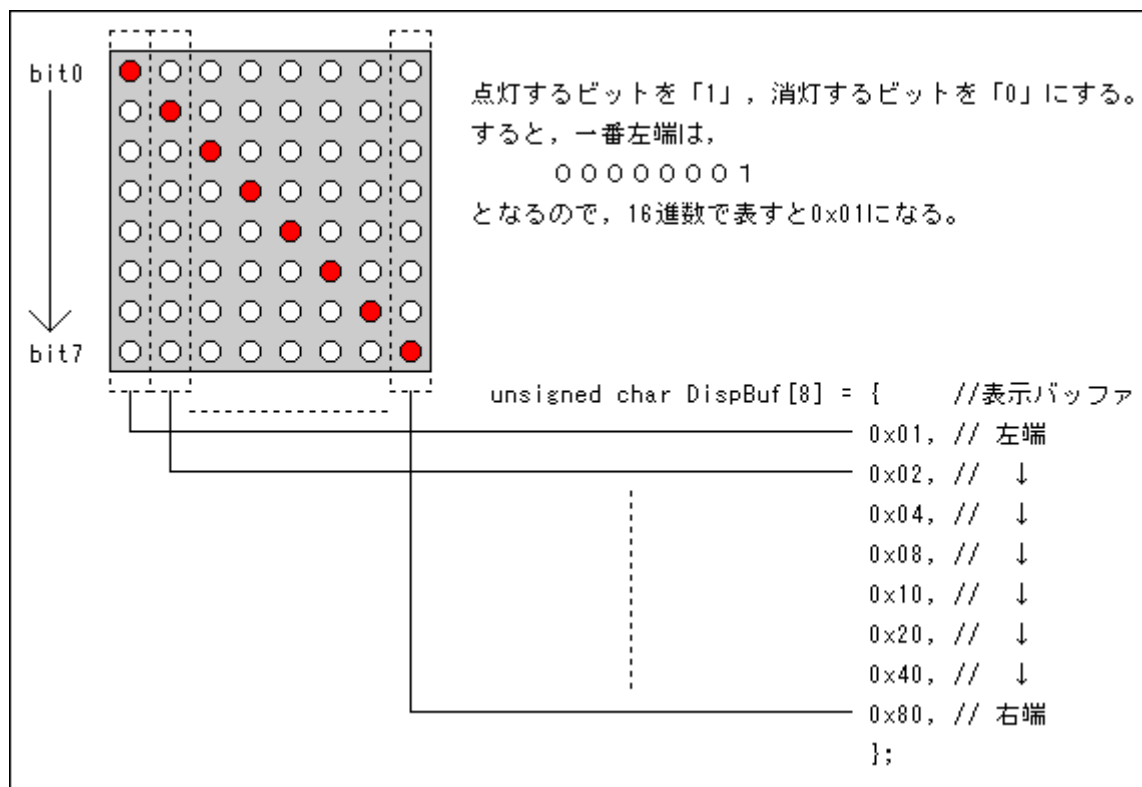
    IO.PCR6      = 0xff; //ポート6,P60-67出力
    IO.PDR6.BYTE = 0xff;
}

```

次は P30~37 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。

P30 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 ディスプレイの左端の LED の点灯パターンを P60~67 に負論理で出力します。同じように P31 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 P60~67 に負論理で出力します。あとはこれを P37(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。

次のプログラムは '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。



ハイパーモニタの「LG」コマンドで「(CD-ROM) : ¥TK-3687mini¥オフ ション¥タイマ\_LED¥フ° ログラム ¥scan00.mot」をダウンロードして実行してください。

### ソースファイル(scan00.c)

```

/*****
/*
/* FILE      :scan.c
/* DATE      :Fri, Sep 12, 2005
/* DESCRIPTION :Main Program
/* CPU TYPE  :H8/3687
/*
/* This file is programed by TOYO-LINUX Co.,Ltd. / yKikuchi
/*
/*****

#include <machine.h> //H8特有の命令を使う
#include "iodefine.h" //内蔵I/Oのラベル定義

```

```

/*****
定数の定義（直接指定）
*****/
//LED表示 -----
#define      DRV_LOGIC  0x300 //ドライバの入力論理
                //負論理入力のビットを‘1’にする

/*****
グローバル変数の定義とイニシャライズ(RAM)
*****/
unsigned long  TimingCnt = 0;      //タイミングカウンタ
unsigned int   ScanFlag  = 0;      //スキャンフラグ
unsigned char  DispBuf[8] = {      //表示バッファ
    0x01,      // 左端
    0x02,      //
    0x04,      //
    0x08,      //
    0x10,      //
    0x20,      //
    0x40,      //
    0x80      // 右端
};

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

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

    // メインループ -----
    while(1){

        //LEDスキャン
        if ((TimingCnt & 0x0700) != ScanFlag){
            ScanFlag = TimingCnt & 0x0700;

            IO.PDR3.BYTE = 0x00; //消灯,スキャンデータ
            IO.PDR6.BYTE = 0xff; //消灯,表示データ

            switch (ScanFlag){
                case 0x0000:
                    IO.PDR6.BYTE = ~DispBuf[0]; //表示データセット
                    IO.PDR3.BYTE = 0x01;      //スキャンデータセット
                    break;
                case 0x0100:
                    IO.PDR6.BYTE = ~DispBuf[1]; //表示データセット
                    IO.PDR3.BYTE = 0x02;      //スキャンデータセット
                    break;
                case 0x0200:
                    IO.PDR6.BYTE = ~DispBuf[2]; //表示データセット
                    IO.PDR3.BYTE = 0x04;      //スキャンデータセット
                    break;
                case 0x0300:
                    IO.PDR6.BYTE = ~DispBuf[3]; //表示データセット

```

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

```

        IO.PDR3.BYTE = 0x08;          //スキャンデータセット
        break;
    case 0x0400:
        IO.PDR6.BYTE = ~DispBuf[4];  //表示データセット
        IO.PDR3.BYTE = 0x10;        //スキャンデータセット
        break;
    case 0x0500:
        IO.PDR6.BYTE = ~DispBuf[5];  //表示データセット
        IO.PDR3.BYTE = 0x20;        //スキャンデータセット
        break;
    case 0x0600:
        IO.PDR6.BYTE = ~DispBuf[6];  //表示データセット
        IO.PDR3.BYTE = 0x40;        //スキャンデータセット
        break;
    case 0x0700:
        IO.PDR6.BYTE = ~DispBuf[7];  //表示データセット
        IO.PDR3.BYTE = 0x80;        //スキャンデータセット
        break;
    }
}

//タイミングカウンタ+1
TimingCnt++;
}
}

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

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

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

```

‘DispBuf’の内容を変更してビルドしてみましょう。いろいろなパターンを表示してみましょう。



次は表示が自動的に変化するようにしてみましょう。次のプログラムは 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。ハイパーモニタの「LG」コマンドで「(CD-ROM) : ¥TK-3687mini¥アプリケーション¥タイマ\_LED¥プログラム¥countup\_01.mot」をダウンロードして実行してください。(概略フローチャート、プログラムの考え方は前回と同じ)

### ソースファイル(countup\_01.c)

```

/*****
/*
/* FILE      :countup_01.c
/* DATE      :Wed, Aug 10, 2005
/* DESCRIPTION :Main Program
/* CPU TYPE  :H8/3687
/*
/* This file is programed by TOYO-LINX Co.,Ltd. / yKikuchi
/*
/*****

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

*****
      定数の定義（直接指定）
*****
//LED表示 -----
#define      DRV_LOGIC 0x300 //ドライバの入力論理
              //負論理入力のビットを ' 1 ' にする

*****
      定数エリアの定義(ROM)
*****
//キャラクタデータ(4×8)
const unsigned char LEDDispData[][4] = {
    {0x00,0xff,0x81,0xff}, // 0
    {0x00,0x02,0xff,0x00}, // 1
    {0x00,0xf1,0x91,0x9f}, // 2
    {0x00,0x89,0x89,0xff}, // 3
    {0x00,0x1f,0x10,0xff}, // 4
    {0x00,0x8f,0x89,0xf9}, // 5
    {0x00,0xff,0x89,0xf9}, // 6
    {0x00,0x0f,0x01,0xff}, // 7
    {0x00,0xff,0x89,0xff}, // 8
    {0x00,0x9f,0x91,0xff}, // 9
    {0x00,0xff,0x11,0xff}, // A
    {0x00,0xff,0x88,0xf8}, // B
    {0x00,0xff,0x81,0x81}, // C
    {0x00,0xf8,0x88,0xff}, // D
    {0x00,0xff,0x89,0x89}, // E
    {0x00,0xff,0x09,0x09}, // F
};

*****
      グローバル変数の定義とイニシャライズ(RAM)
*****
unsigned int   TimingCnt   = 0; //タイミングカウンタ
unsigned char  DisplayData = 0x00; //表示データ
unsigned int   ScanFlag    = 0; //スキャンフラグ

```

```

unsigned int   CountupFlag   =   0;           //カウントアップフラグ
unsigned char  DispBuf[8]    =   {0x00,0x00,0x00,0x00
                                ,0x00,0x00,0x00,0x00}; //表示バッファ

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

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

    // メインループ -----
    while(1){
        //表示データ作成(ローテート)
        if ((TimingCnt&0x8000)!=CountupFlag){
            CountupFlag = TimingCnt & 0x8000;
            DispBuf[0] = LEDDispData[DisplayData / 0x10][0];
            DispBuf[1] = LEDDispData[DisplayData / 0x10][1];
            DispBuf[2] = LEDDispData[DisplayData / 0x10][2];
            DispBuf[3] = LEDDispData[DisplayData / 0x10][3];
            DispBuf[4] = LEDDispData[DisplayData & 0x0f][0];
            DispBuf[5] = LEDDispData[DisplayData & 0x0f][1];
            DispBuf[6] = LEDDispData[DisplayData & 0x0f][2];
            DispBuf[7] = LEDDispData[DisplayData & 0x0f][3];
            DisplayData++;
        }

        //LEDスキャン
        if ((TimingCnt&0x0700)!=ScanFlag){
            ScanFlag = TimingCnt & 0x0700;
            IO.PDR3.BYTE = 0x00; //消灯,スキャンデータ
            IO.PDR6.BYTE = 0xff; //消灯,表示データ
            switch (ScanFlag){
                case 0x0000:
                    IO.PDR6.BYTE = ~DispBuf[0]; //表示データセット
                    IO.PDR3.BYTE = 0x01; //スキャンデータセット
                    break;
                case 0x0100:
                    IO.PDR6.BYTE = ~DispBuf[1]; //表示データセット
                    IO.PDR3.BYTE = 0x02; //スキャンデータセット
                    break;
                case 0x0200:
                    IO.PDR6.BYTE = ~DispBuf[2]; //表示データセット
                    IO.PDR3.BYTE = 0x04; //スキャンデータセット
                    break;
                case 0x0300:
                    IO.PDR6.BYTE = ~DispBuf[3]; //表示データセット
                    IO.PDR3.BYTE = 0x08; //スキャンデータセット
                    break;
                case 0x0400:
                    IO.PDR6.BYTE = ~DispBuf[4]; //表示データセット
                    IO.PDR3.BYTE = 0x10; //スキャンデータセット
                    break;
                case 0x0500:
                    IO.PDR6.BYTE = ~DispBuf[5]; //表示データセット

```

```

        IO.PDR3.BYTE = 0x20;          //スキャンデータセット
        break;
    case 0x0600:
        IO.PDR6.BYTE = ~DispBuf[6];  //表示データセット
        IO.PDR3.BYTE = 0x40;        //スキャンデータセット
        break;
    case 0x0700:
        IO.PDR6.BYTE = ~DispBuf[7];  //表示データセット
        IO.PDR3.BYTE = 0x80;        //スキャンデータセット
        break;
    }
}

//タイミングカウンタ+1
TimingCnt++;
}
}

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

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

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

```

## 5 ルーレットを作ろう

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。この章ではルーレットを作ってみます。

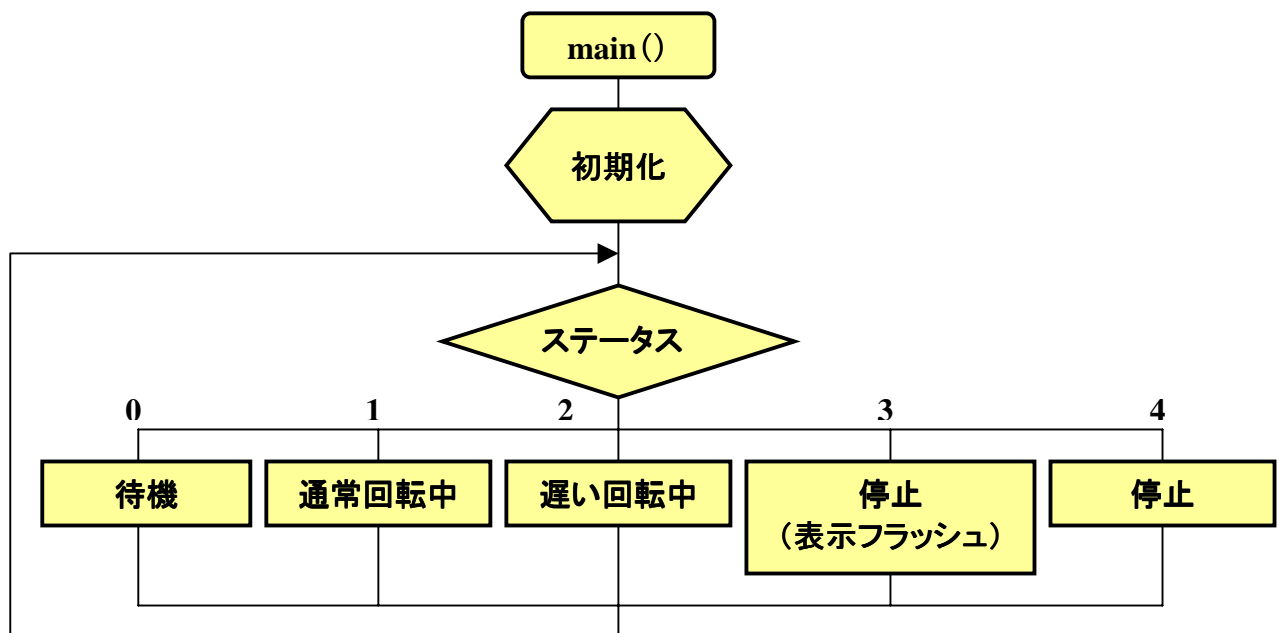
### ■ 仕様

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 個ずつ点灯させます。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。

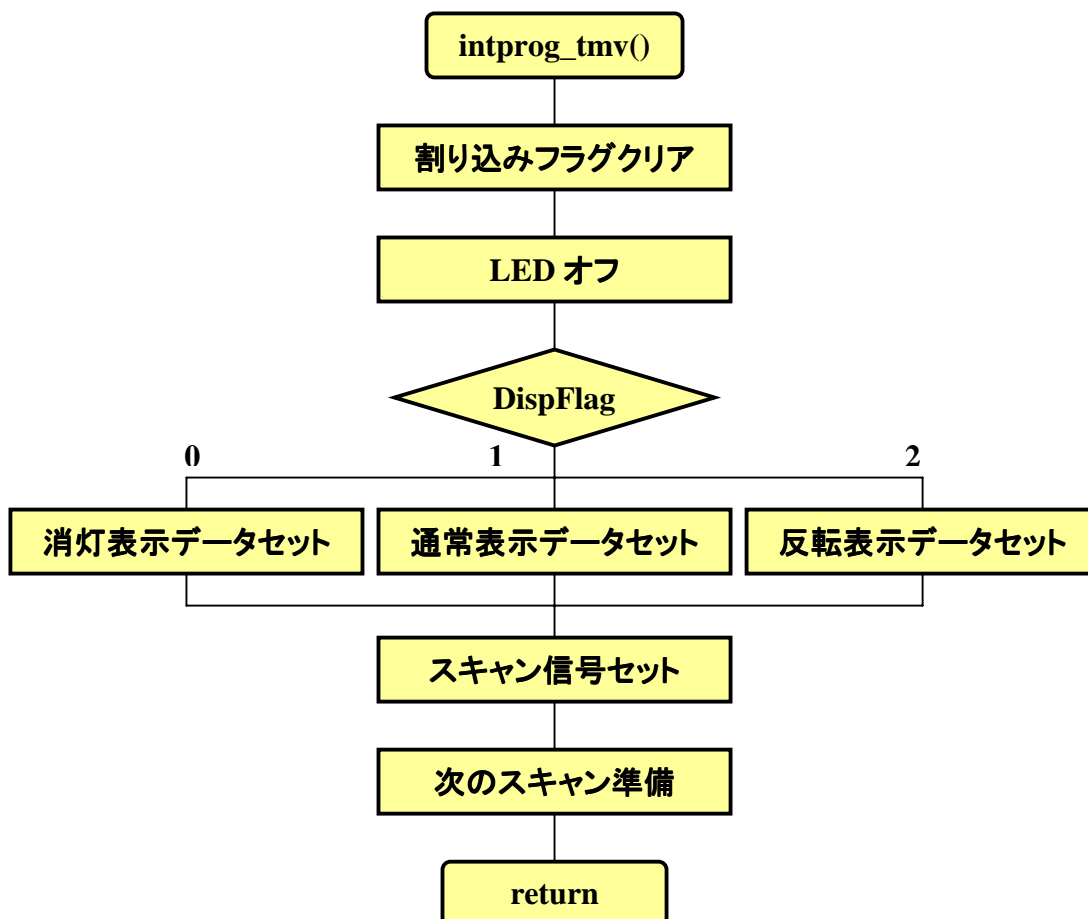
なお、最初にスイッチが押されるまでの間は、スタンバイ表示として周囲の LED を交互に点滅させます。

### ■ 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)



前の章まではダイナミックスキャンもメインルーチンで行なっていました。本来、LED の表示はメインとは関係なくバックグラウンドで行なうほうがすっきりします。それで、このプログラムではダイナミックスキャンはタイマ V の割込み ('intprog\_tmv()') ルーチンで行ないます。表示するデータを DispBuf[0]～[9]にセットし、割り込みルーチンではスキャン信号に応じて表示データを DispBuf[0]～[9]から取り出してポートにセットします。おまけの機能として DispFlag の値によって、消灯、通常表示、反転表示を選択できるようにします。この機能を利用して、数字が決まったときにフラッシュ表示するようにします。



LED の回転速度はタイマ B1 割込みを利用したソフトウェアタイマで設定します。タイマ B1 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とスイッチ入力処理を行ないます。

## ■ ダウンロードと実行

ハイパーモニタの「LG」コマンドで、付属 CD-R 内の「(CD-ROM) : ¥TK-3687mini¥ﾌﾟｼﾞｵﾝ¥ﾀｲﾑ\_¥LED¥ﾌﾟ ﾛｸﾞ ﾚﾙ¥roulette\_01.mot」をダウンロードして実行してください

## ソースファイル(roulette\_01.c)

```
/*
 *
 * FILE      : roulette_01.c
 * DATE      : Mon, Jun 20, 2005
 * DESCRIPTION : Main Program
 * CPU TYPE   : H8/3687
 *
 * This file is programmed by TOYO-LINX Co.,Ltd. / yKikuchi
 */
/*****
 *
 * インクルードファイル
 *
 *****/
#include <machine.h> //H8特有の命令を使う
#include "iodefine.h" //内蔵I/Oのラベル定義

/*****
 *
 * 定数の定義 (直接指定)
 *
 *****/
//LED表示 -----
#define DRV_LOGIC 0x300 //ドライバの入力論理
//負論理入力のビットを ' 1 ' にする

//ソフトウェアタイマ -----
#define T0 80 //T0(80ms)
#define T1 1000 //T1(1000ms)
#define T2 5000 //T2(5000ms)
#define T3 0 //T3(ms)

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

//キャラクタデータ(8×8)
const unsigned char LEDDispData[][8] = {
{0x00,0x7e,0xa1,0x91,0x89,0x85,0x7e,0x00}, // 0
{0x00,0x00,0x00,0x82,0xff,0x80,0x00,0x00}, // 1
{0x00,0x82,0xc1,0xa1,0x91,0x89,0x86,0x00}, // 2
{0x00,0x41,0x81,0x81,0x8d,0x93,0x61,0x00}, // 3
{0x00,0x30,0x28,0x24,0x22,0xff,0x20,0x00}, // 4
{0x00,0x4f,0x89,0x89,0x89,0x89,0x71,0x00}, // 5
{0x00,0x7c,0x8a,0x89,0x89,0x89,0x70,0x00}, // 6
{0x00,0x01,0xe1,0x11,0x09,0x05,0x03,0x00}, // 7
{0x00,0x76,0x89,0x89,0x89,0x89,0x76,0x00}, // 8
{0x00,0x0e,0x91,0x91,0x91,0x51,0x3e,0x00}, // 9
{0x00,0x02,0xff,0x00,0xff,0x81,0xff,0x00}, //10
{0x00,0x02,0xff,0x00,0x00,0x02,0xff,0x00}, //11
};

/*****
 *
 * グローバル変数の定義とイニシャライズ (RAM)
 *
 *****/
// ルーレットに関係した変数 -----
unsigned char RouletteStatus = 0; //ルーレットステータス
```

```

// 0:待機
// 1:通常回転中
// 2:遅い回転中
// 3:停止
unsigned int    RouletteDisp    =    0x0800;    //表示データ(ルーレット部)
unsigned char   RouletteData    =    0;        //表示データに対応した数値
unsigned char   RouletteStopCnt =    0;        //停止カウンタ
unsigned char   RouletteFlash   =    10 ;      //フラッシュ

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

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

// ソフトウェアタイマに関係した変数 -----
struct SoftTimer{                //ソフトウェアタイマの構造体タグ
    unsigned char   Status;      //タイマステータス
// 0:停止中(タイマ未使用)
// 1:スタート指令
// 2:カウント中
// 3:カウント終了
    unsigned int    Count;      //タイマカウンタ
};
struct SoftTimer TimT0;          //T0タイマ(ルーレットの通常回転)
struct SoftTimer TimT1;          //T1タイマ(ルーレットの遅い回転)
struct SoftTimer TimT2;          //T2タイマ(ルーレット結果表示時間)
struct SoftTimer TimT3;          //T3タイマ(予備)

/*****
関数の定義
*****/
void    dec_soft_timer(struct SoftTimer *,unsigned int);
void    init_io(void);
void    init_soft_timer(void);
void    init_tmb1(void);
void    init_tmv(void);
void    intprog_tmb1(void);
void    intprog_tmv(void);
void    main(void);
void    roulette(void);
void    set_disp_data(unsigned char);
void    switch_in(void);

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

```

```

init_io();
init_tmv();
init_tmb1();
init_soft_timer();

// メインループ -----
while(1){
    roulette();
}
}

/*****
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;
}
}
}

```

```

/*****

```

```

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

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

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

/*****
    タイマV イニシャライズ
*****/
void init_tmv(void)
{
    TV.TCSR.V.BYTE = 0x00;    //TOMV端子は使わない
    TV.TCOR.A      = 156;    //周期=1ms(1kHz)
    TV.TCR.V1.BYTE = 0x01;    //TRGVトリガ入力禁止,
    TV.TCR.V0.BYTE = 0x4b;    //コンペアマッチA 割込みイネーブル
                                //コンペアマッチA でTCNTVクリア
                                //内部クロック /128(=156.25kHz)
}

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

    //表示を消す
    IO.PDR5.BYTE = (IO.PDR5.BYTE & 0xfc) | (0x00 ^ (DRV_LOGIC/0x100));
    IO.PDR3.BYTE = 0x00 ^ DRV_LOGIC;
    IO.PDR6.BYTE = 0xff;

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

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

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

/*****
    タイマB1 イニシャライズ
*****/
void init_tmb1(void)
{
    TB1.TMB1.BYTE = 0xf9;    //オートリロード,内部クロック /2048

```

```

TB1.TLB1      = 0-97;      //周期=10ms(100Hz)
IRR2.BIT.IRRTB1 = 0;      //タイマB1割込み要求フラグ クリア
IENR2.BIT.IENTB1 = 1;     //タイマB1割込み要求イネーブル
}

/*****
    タイマB1 割込み(10ms)
*****/
#pragma regsave (intprog_tmb1)
void intprog_tmb1(void)
{
    //タイマB1割込み要求フラグ クリア
    IRR2.BIT.IRRTB1 = 0;
    //ソフトウェアタイマ T0
    if (TimT0.Status==1 || TimT0.Status==2){
        dec_soft_timer(&TimT0,T0/10);
    }
    //ソフトウェアタイマ T1
    if (TimT1.Status==1 || TimT1.Status==2){
        dec_soft_timer(&TimT1,T1/10);
    }
    //ソフトウェアタイマ T2
    if (TimT2.Status==1 || TimT2.Status==2){
        dec_soft_timer(&TimT2,T2/10);
    }
    //ソフトウェアタイマ T3
    if (TimT3.Status==1 || TimT3.Status==2){
        dec_soft_timer(&TimT3,T3/10);
    }
    //スイッチ入力
    switch_in();
}

/*****
    ソフトウェアタイマのデクリメント
-----
    引数      *pst      ソフトウェアタイマ構造体のポインタ
            initial     タイマカウンタの初期値
*****/
void dec_soft_timer(struct SoftTimer *pst,unsigned int initial)
{
    if (pst->Status==1){                //タイマスタート指令
        pst->Status      = 2;          //カウント中セット
        pst->Count       = initial;    //タイマカウンタ初期化
    }
    pst->Count--; //カウンタ-1
    if (pst->Count==0)                //カウンタが0になった
        pst->Status      = 3;          //カウント終了セット
}

/*****
    ソフトウェアタイマのイニシャライズ
*****/
void init_soft_timer(void)
{
    TimT0.Status = 0; TimT1.Status = 0; TimT2.Status = 0; TimT3.Status = 0;
}

/*****
    スイッチ入力
*****/
void switch_in(void)

```

```

{
    switch(SwStatus){
        case 0:
            SwData1 = ~IO.PDR5.BYTE & 0x38;
            if (SwData1!=0) {SwStatus = 1;}
            else          {SwData2 = SwData3 =0;}
            break;
        case 1:
            if (SwData1==(~IO.PDR5.BYTE & 0x38)){
                SwData2 = SwData1;
                SwData4 = SwData4 | (SwData2 & (~SwData3));
                SwData3 = SwData2;
            }
            SwStatus = 0;
            break;
    }
}

/*****
  表示データのセット
  *****/
void set_disp_data(unsigned char data)
{
    unsigned char i;

    for (i=0; i<8; i++){
        DispBuf[i] = LEDDispData[data][i];
    }
}

```

タイマ V の割り込みとタイマ B1 の割り込みを使うため、「intprg.c」を次のように修正します。

### ソースファイル(intprg.c)

```

/*****
  /*
  /* FILE      :intprg.c
  /* DATE      :Mon, Jun 20, 2005
  /* DESCRIPTION :Interrupt Program
  /* CPU TYPE  :H8/3687
  /*
  /* This file is generated by Renesas Project Generator (Ver.4.0).
  /*
  *****/

#include <machine.h>

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

#pragma section IntPRG
// vector 1 Reserved

// vector 2 Reserved

// vector 3 Reserved

// vector 4 Reserved

```

追加

```

// vector 5 Reserved

// vector 6 Reserved

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

// vector 21 Reserved

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

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

// vector 31 Reserved

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

```

変更

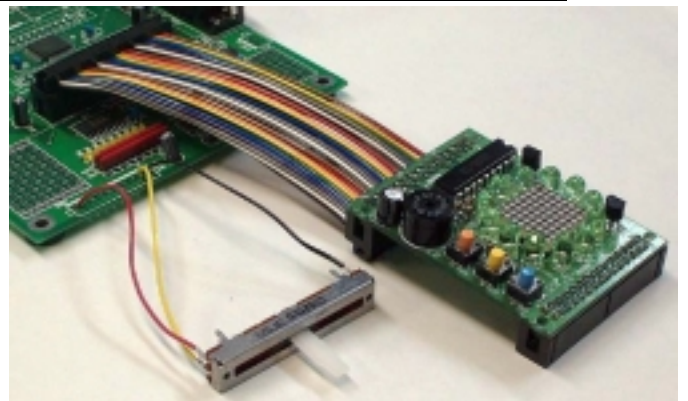
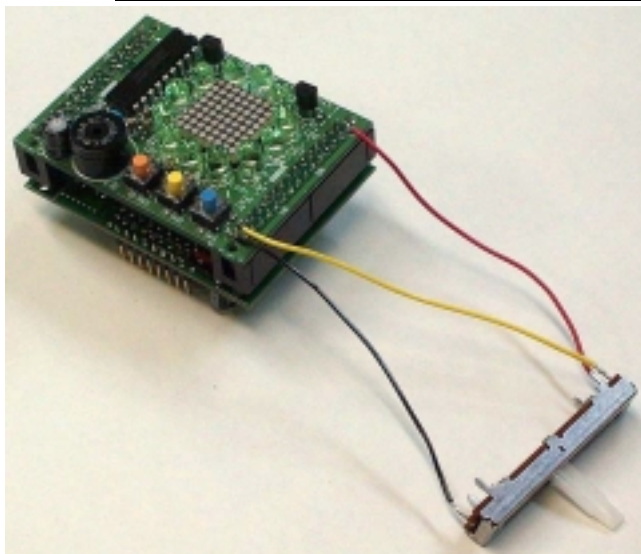
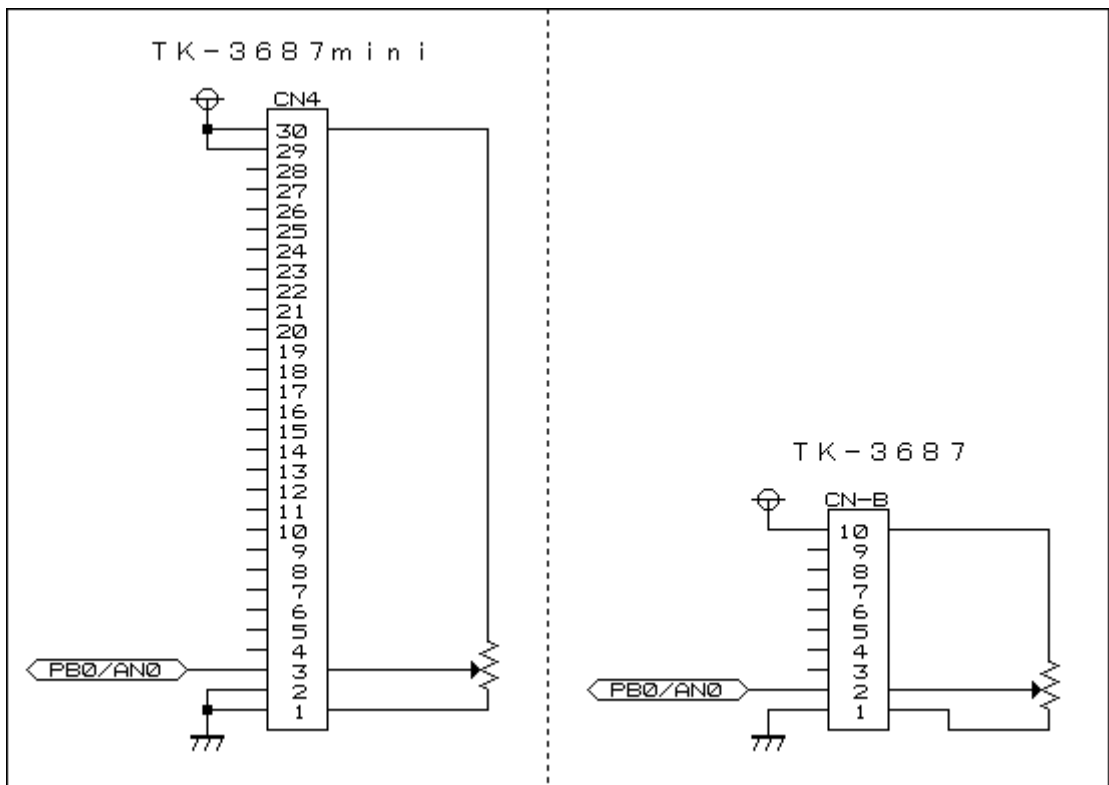
変更

## 6 AD 変換値の表示

LED ディスプレイの応用例として H8/3687 に内蔵されている AD コンバータで取得した AD 値を表示します。

### ■ 仕様

H8/3687 には 10 ビットの AD コンバータが内蔵されています。AN0 (PB0) にボリュームをつなぎ読み取った AD 値を LED ディスプレイに表示します。ノイズ除去のため、4096 回連続して AD 値を読み取り、その平均値を最終的な AD 値とします。なお、表示は 2 桁のため、上位 8 ビットを 16 進数で表示することになります (00~FF)。



## ■ プログラムの説明

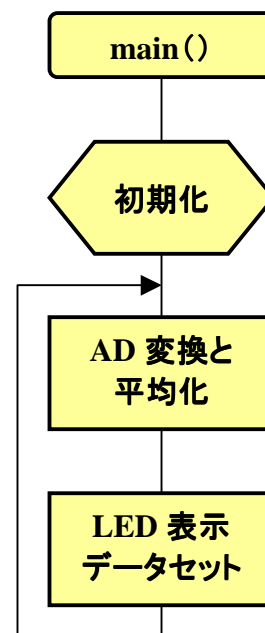
AD 変換と平均化は 'ad\_conv()' ルーチンで行なっています。連続して 4096 回 AD 値を読み取り AdcBuf に加算します。4096 回読み取ったら AdcBuf を 4096 で割って平均値を求め AdcData にセットします。

LED に表示する数字の形は LEDDispData テーブルになっています。求めた AdcData の上位 8 ビットに応じてテーブルからデータを取り出し DispBuf [0]~[7] にセットします。

概略フローとソースリストを以下に示します。

## ■ ダウンロードと実行

ハイパーモニタの「LG」コマンドで、付属 CD-R 内の「(CD-ROM) : ¥TK-3687mini¥お♯シヨん¥タイマ\_LED¥♯プログラム¥adconv\_01.mot」をダウンロードして実行してください。



## ソースファイル(adconv\_01.c)

```
/**
 *
 * FILE      :adconv_01.c
 * DATE      :Wed, Aug 10, 2005
 * DESCRIPTION :Main Program
 * CPU TYPE   :H8/3687
 *
 * This file is programmed by TOYO-LINX Co.,Ltd. / yKikuchi
 */
/**
 *
 * インクルードファイル
 */
#include <machine.h> //H8特有の命令を使う
#include "iodefine.h" //内蔵I/Oのラベル定義

/**
 * 定数の定義 (直接指定)
 */
//LED表示 -----
#define DRV_LOGIC 0x300 //ドライバの入力論理
//負論理入力のビットを '1' にする

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

//キャラクタデータ(4x8)
const unsigned char LEDDispData[][4] = {
{0x00,0xff,0x81,0xff}, // 0
{0x00,0x02,0xff,0x00}, // 1
```

```

    {0x00,0xf1,0x91,0x9f}, // 2
    {0x00,0x89,0x89,0xff}, // 3
    {0x00,0x1f,0x10,0xff}, // 4
    {0x00,0x8f,0x89,0xf9}, // 5
    {0x00,0xff,0x89,0xf9}, // 6
    {0x00,0x0f,0x01,0xff}, // 7
    {0x00,0xff,0x89,0xff}, // 8
    {0x00,0x9f,0x91,0xff}, // 9
    {0x00,0xff,0x11,0xff}, // A
    {0x00,0xff,0x88,0xf8}, // B
    {0x00,0xff,0x81,0x81}, // C
    {0x00,0xf8,0x88,0xff}, // D
    {0x00,0xff,0x89,0x89}, // E
    {0x00,0xff,0x09,0x09}, // F
};

/*****
    グローバル変数の定義とイニシャライズ(RAM)
*****/
// AD値に関係した変数 -----
unsigned int   AdcData       = 0; //平均値
unsigned long  AdcBuf;        //加算バッファ
unsigned int   AdcCnt;       //加算カウンタ

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

/*****
    関数の定義
*****/
void          ad_conv(void);
void          init_ad(void);
void          init_io(void);
void          init_tmv(void);
void          intprog_tmv(void);
void          main(void);

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

    // メインループ -----
    while(1){
        ad_conv(); //AD変換&平均化

        //表示データセット
        DispBuf[0] = LEDDispData[AdcData / 0x1000][0];
        DispBuf[1] = LEDDispData[AdcData / 0x1000][1];
        DispBuf[2] = LEDDispData[AdcData / 0x1000][2];
        DispBuf[3] = LEDDispData[AdcData / 0x1000][3];
    }
}

```



```

DispBuf[4] = LEDDispData[(AdcData / 0x0100) & 0x0f][0];
DispBuf[5] = LEDDispData[(AdcData / 0x0100) & 0x0f][1];
DispBuf[6] = LEDDispData[(AdcData / 0x0100) & 0x0f][2];
DispBuf[7] = LEDDispData[(AdcData / 0x0100) & 0x0f][3];
}
}

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

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

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

/*****
A/D変換器イニシャライズ
*****/
void init_ad(void)
{
    AD.ADCSR.BYTE = 0x00; //割り込みディセーブル, 単一モード, 134ステート, CHO
}

/*****
A/D変換 & 平均化
*****/
void ad_conv(void)
{
    unsigned int i;

    AdcBuf = 0;
    for (i=0; i<4096; i++){          //平均回数=4096回
        AD.ADCSR.BIT.ADST = 1;      //AD変換スタート
        while(AD.ADCSR.BIT.ADF==0) {} //ADエンドフラグ=1まで待つ
        AD.ADCSR.BIT.ADF = 0;       //ADエンドフラグクリア
        AdcBuf = AdcBuf + AD.ADDRA;  //加算
    }
    AdcData = (unsigned int)(AdcBuf / 4096); //平均
}

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

```

```

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

    //表示を消す
    IO.PDR5.BYTE = (IO.PDR5.BYTE & 0xfc) | (0x00 ^ (DRV_LOGIC/0x100));
    IO.PDR3.BYTE = 0x00 ^ DRV_LOGIC;
    IO.PDR6.BYTE = 0xff;

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

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

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

```

## ソースファイル(intprg.c)

```

/*****
/*
/* FILE      :intprg.c
/* DATE      :Wed, Aug 10, 2005
/* DESCRIPTION :Interrupt Program
/* CPU TYPE  :H8/3687
/*
/* This file is generated by Renesas Project Generator (Ver.4.0).
/*
*****/

```

```
#include <machine.h>
```

追加

```
extern void intprog_tmv(void);
```

```

#pragma section IntPRG
// vector 1 Reserved

// vector 2 Reserved

// vector 3 Reserved

// vector 4 Reserved

// vector 5 Reserved

// vector 6 Reserved

```

```

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

// vector 21 Reserved

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

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

// vector 31 Reserved

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

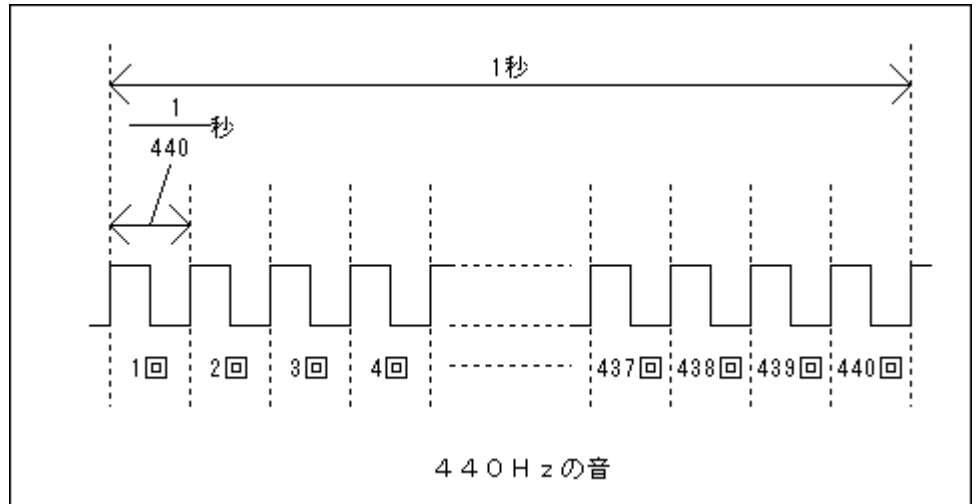
```

変更

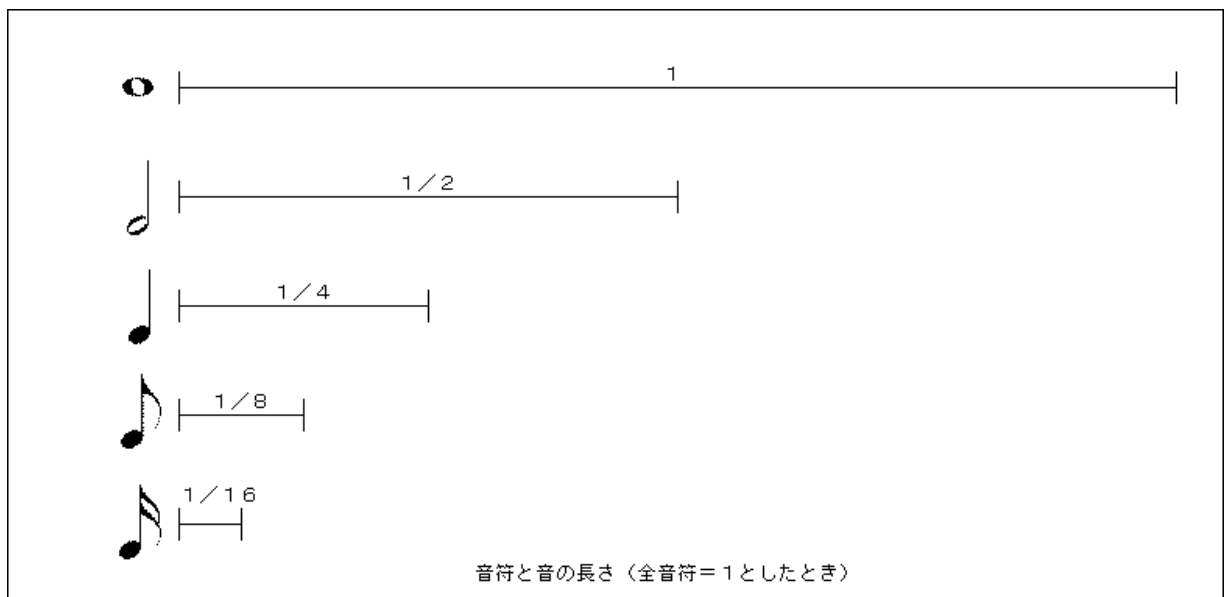
## 7 メロディを奏でよう

動作チェックを行なったときに使用したプログラムはメロディを奏でました。どのようにすればメロディを流すことができるでしょうか。考えてみましょう。

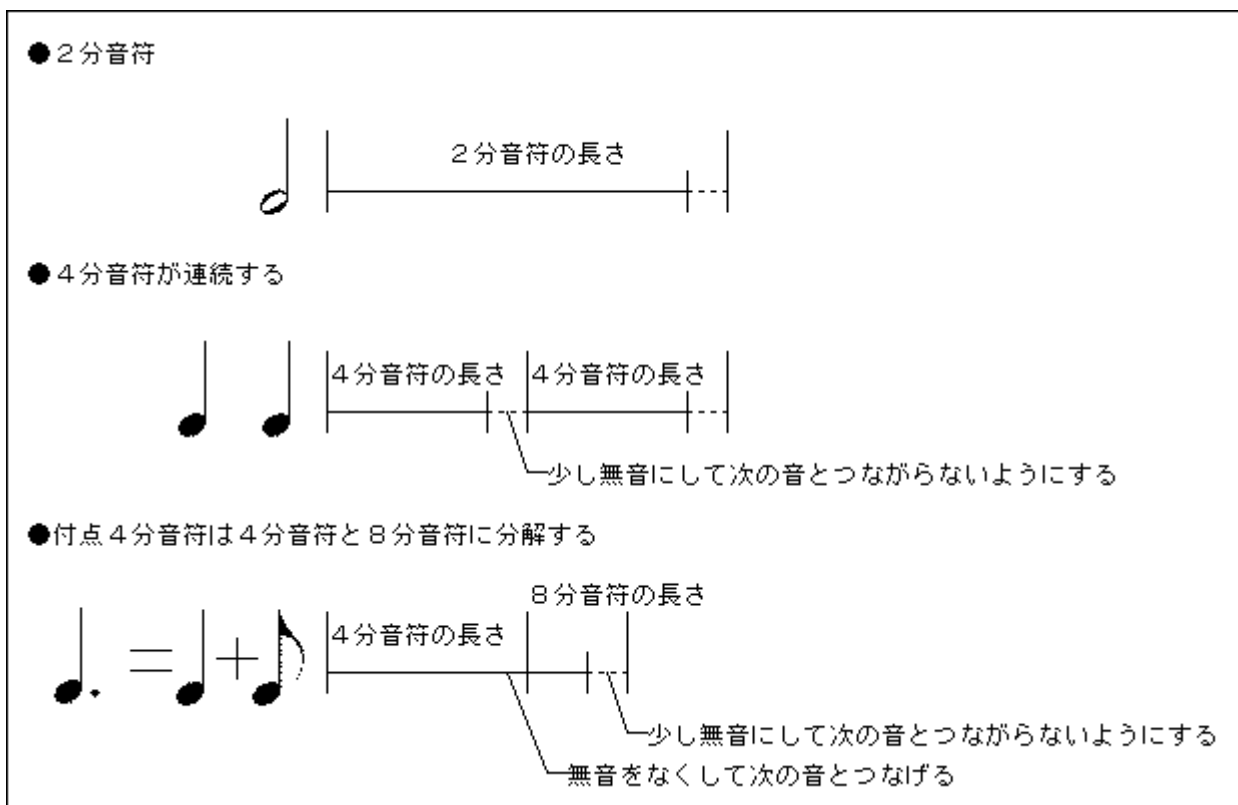
メロディにまず必要なのは、ドレミファソラシド、つまり音階です。音階は物理的には音の周波数のことです。で、周波数とは何かといえば、1秒間に何回くりかえすか、ということです(単位は Hz:ヘルツ)。このプログラムの基準音はラ(A)ですが、周波数は 440Hz になります。今回使ったサウンドという部品は、ある周波数のパルス信号を加えると、その周波数の音を出します。というわけで、出したい音の周波数のパルス信号を P52 から出力することで、特定の音階の音を出しています。あとは、周波数をいろいろ変えればメロディになっていきます。例えば、440Hz の音を出すときには右の図のように P52 からパルス信号を出力します。



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が、半分ずつになっていきます(例:4分音符二つで2分音符一つの長さ)。普通は曲の速さによって基準となる長さを変えていきます。楽譜の左上に「♩=120」という記号があるのを見たことがあるでしょうか。これは1分間に4分音符が120個になる速さで演奏する、という意味です。このプログラムはこれを採用しました。というわけで、全音符が2秒、2分音符が1秒、4分音符が0.5秒、8分音符が0.25秒…の長さになります。



音符の長さでもう一つ重要なのは、音符の長さの全部で音を出すか、ということです。例えば、同じ音階の4分音符が2つ並んでいるのと、2分音符との違いです。トータル音の長さ(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。で、この4分音符は音符の長さの全部で音を出すよう指定して、次の8分音符と音をつないで一つの音にします。

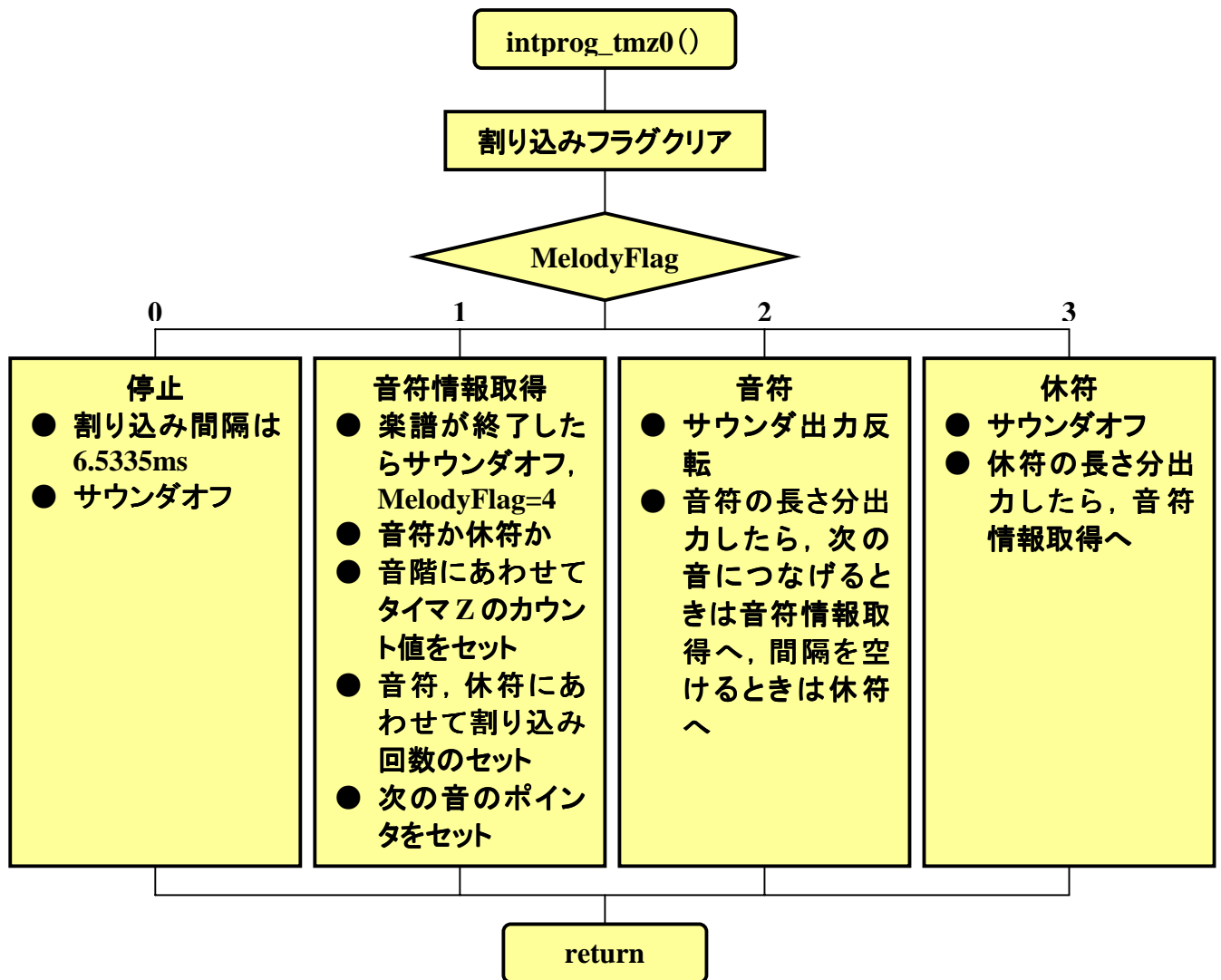


メロディの重要な部分の最後は休止です。これは無音状態をどれくらい続けるかで指定します。無音の長さは音符の長さと同じ方法で指定します。

プログラム中では全音符の音階をテーブルとして持たせています。また、楽譜もテーブルで持たせています。この二つのテーブルを使って、タイマ Z の割込み間隔を調整し、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(タイマ Z 割り込み 'intprog\_tmz0()')ルーチンとソースリストをご覧ください。(次ページ)

## ■ ダウンロードと実行

ハイパーモニタの「LG」コマンドで、付属 CD-R 内の「(CD-ROM) : ¥TK-3687mini¥オプ ション¥タイマ\_LED¥ワウ ラム¥melody\_01.mot」をダウンロードして実行してください。



### ソースファイル(melody\_01.c)

```

/*****/
/*                                     */
/* FILE      :melody_01.c             */
/* DATE      :Tue, Jun 28, 2005      */
/* DESCRIPTION :Main Program         */
/* CPU TYPE   :H8/3687               */
/*                                     */
/* This file is programed by TOYO-LINX Co.,Ltd. / yKikuchi */
/*                                     */
/*****/

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

*****
    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

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

/*****
      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
};

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

    // メインループ -----
    while(1){
        if (MelodyFlag==0){
            GakufuPnt = Gakufu_0;

```

```

        MelodyFlag = 1;
    }
    else if (MelodyFlag==4){
        MelodyFlag = 0;
    }
}

/*****
I/Oポート イニシャライズ
*****/
void init_io(void)
{
    IO.PMR5.BYTE = 0x00;    //ポート5,汎用入出力ポート
    IO.PUCR5.BYTE = 0x38;    //ポート5,P53-55内蔵プルアップオン
    IO.PCR5 = 0x07;    //ポート5,P50-52出力,P53-P57入力
    IO.PDR5.BYTE = 0x07;
}

/*****
音階テーブル
{(タイマZのGRAにセットする値),(全音符の長さ,割込回数)}
*****/
const unsigned int OnkaiTbl[][2] = {
    {22727, 882},    //74h,A ,ラ ,220.00Hz
    {21452, 934},    //75h,A# ,ラ# ,233.08Hz
    {20248, 988},    //76h,B ,シ ,246.94Hz

    {19111,1048},    //77h,C ,ド ,261.63Hz
    {18039,1110},    //78h,C# ,ド# ,277.18Hz
    {17026,1176},    //79h,D ,レ ,293.66Hz
    {16070,1246},    //7Ah,D# ,レ# ,311.13Hz
    {15169,1320},    //7Bh,E ,ミ ,329.63Hz
    {14317,1398},    //7Ch,F ,ファ ,349.23Hz
    {13514,1480},    //7Dh,F# ,ファ#,369.99Hz
    {12755,1570},    //7Eh,G ,ソ ,392.00Hz
    {12039,1662},    //7Fh,G# ,ソ# ,415.30Hz
    {11364,1760},    //80h,A ,ラ ,440.00Hz    // 基準音
    {10726,1866},    //81h,A# ,ラ# ,466.16Hz
    {10124,1976},    //82h,B ,シ ,493.88Hz

    { 9556,2094},    //83h,C ,ド ,523.25Hz
    { 9019,2218},    //84h,C# ,ド# ,554.37Hz
    { 8513,2350},    //85h,D ,レ ,587.33Hz
    { 8035,2490},    //86h,D# ,レ# ,622.25Hz
    { 7584,2638},    //87h,E ,ミ ,659.26Hz
    { 7159,2794},    //88h,F ,ファ ,698.46Hz
    { 6757,2960},    //89h,F# ,ファ#,739.99Hz
    { 6378,3136},    //8Ah,G ,ソ ,783.99Hz
    { 6020,3324},    //8Bh,G# ,ソ# ,830.61Hz
    { 5682,3520},    //8Ch,A ,ラ ,880.00Hz
    { 5363,3730},    //8Dh,A# ,ラ# ,932.33Hz
    { 5062,3952},    //8Eh,B ,シ ,987.77Hz
};

/*****
タイマZ イニシャライズ
*****/
void init_tmz(void)
{
    TZ.TSTR.BYTE = 0x00;    //TCNT0,1 停止
    TZ0.TCR.BYTE = 0x21;    //GRAのコンペアマッチでTCNT=0, /2
}

```



```

TZ0.TIORA.BYTE = 0x00; //GRAはアウトプットコンペアレジスタ
//コンペアマッチによる出力禁止
TZ0.TSR.BYTE = 0x00; //割り込みフラグクリア
TZ0.TIER.BYTE = 0x01; //コンペアマッチインタラプトイネーブルA
TZ0.GRA = 0xffff; //メロディなしのときは6.5535msで割り込みをかける
TZ0.TCNT = 0x0000; //TCNT0=0
TZ.TSTR.BYTE = 0x01; //TCNT0 カウントスタート
}

/*****
    タイマZ チャンネル0 割り込み
*****/
#pragma regsave (intprog_tmz0)
void intprog_tmz0(void)
{
    unsigned char  Onkai;
    unsigned char  Onpu;
    unsigned char  Kyufu = 0;

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

    //メロディ
    switch (MelodyFlag){
        //停止 -----
        case 0:
            TZ0.GRA = 0xffff; //メロディなしのときは6.5535msで割り込みをかける
            IO.PDR5.BIT.B2 = 1; //サウンドオフ
            break;
        //音符情報取得 -----
        case 1:
            if (*GakufuPnt==0xffff){ //楽譜終了
                IO.PDR5.BIT.B2 = 1; //サウンドオフ
                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

            //音階にあわせてタイマZのカウント値をセットする,割り込み間隔の調整
            TZ.TSTR.BIT.STRO = 0;
            TZ0.GRA = OnkaiTbl[Onkai - (0x80 - KIJUN_ON)][0];
            TZ0.TCNT = 0x0000;
            TZ.TSTR.BIT.STRO = 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:
    IO.PDR5.BIT.B2 = ~IO.PDR5.BIT.B2;    //サウンド出力反転
    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:
    IO.PDR5.BIT.B2 = 1;    //サウンドオフ
    KyufuCnt--;
    if (KyufuCnt==0)    {MelodyFlag = 1;}    //終了したら次の音符に移る
    break;
}
}

```

### ソースファイル(intprg.c)

```

/*****
/*
/* FILE      :intprg.c
/* DATE      :Tue, Jun 28, 2005
/* DESCRIPTION :Interrupt Program
/* CPU TYPE  :H8/3687
/*
/* This file is generated by Renesas Project Generator (Ver.4.0).
/*
*****/

```

追加

```
#include <machine.h>
```

```
extern void intprog_tmz0(void);
```

```
#pragma section IntPRG
```

```
// vector 1 Reserved
```

```
// vector 2 Reserved
```

```
// vector 3 Reserved
```

```
// vector 4 Reserved
```

```
// vector 5 Reserved
```

```
// vector 6 Reserved
```

```
// vector 7 NMI
```

```
__interrupt(vect=7) void INT_NMI(void) {/* sleep(); */}
```

```
// vector 8 TRAP #0
```

```
__interrupt(vect=8) void INT_TRAPO(void) {/* sleep(); */}
```

```

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

// vector 21 Reserved

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

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

// vector 31 Reserved

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

```

変更

## 8 タイマ&LED ディスプレイへの応用

この章では、これまでのまとめてして、グラフィック&メロディ、99秒タイマ、99分タイマ、デジタル時計を作ります。そして、電源オンでプログラムを切り替えて使えるようにしてみましょう。

### ■ FDTによるプログラムのダウンロード

このプログラムはサイズの関係でハイパーH8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H8/3687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。

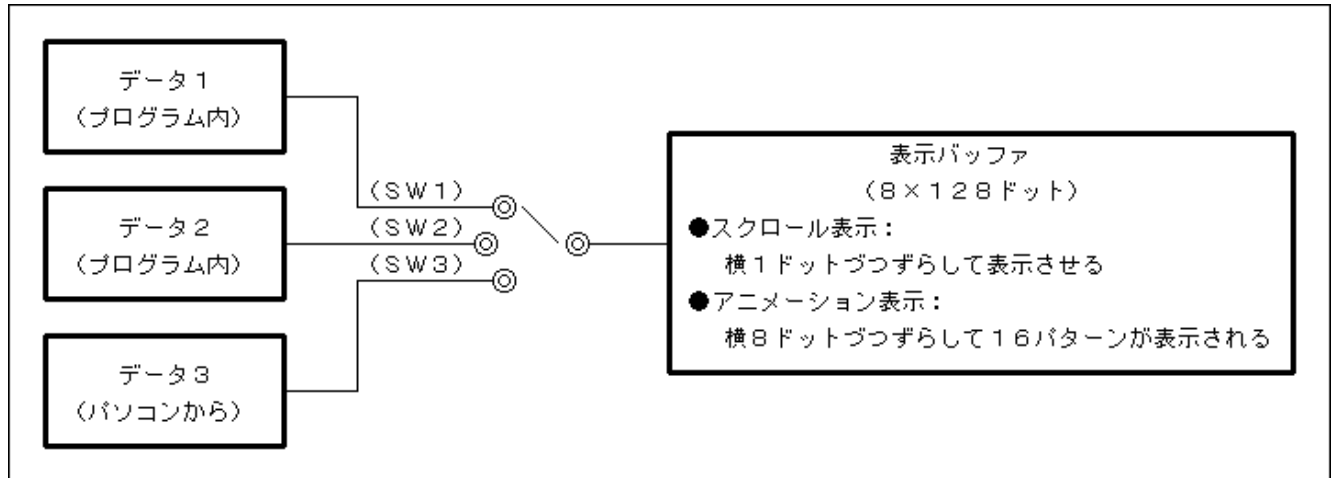
フラッシュメモリにダウンロードするプログラムは、付属CD-R内の「(CD-ROM)：¥TK-3687mini¥アプリケーション¥timer\_led¥プログラム¥timer\_led.mot」です。FDTの使い方についてはCD-R内のマニュアル、TK-3687miniは「TK-3687mini組み立て手順書」、TK-3687は「TK-3687ユーザ向けFDTでの書き込み手順」を参考にして下さい。

### ■ プログラムの動かし方

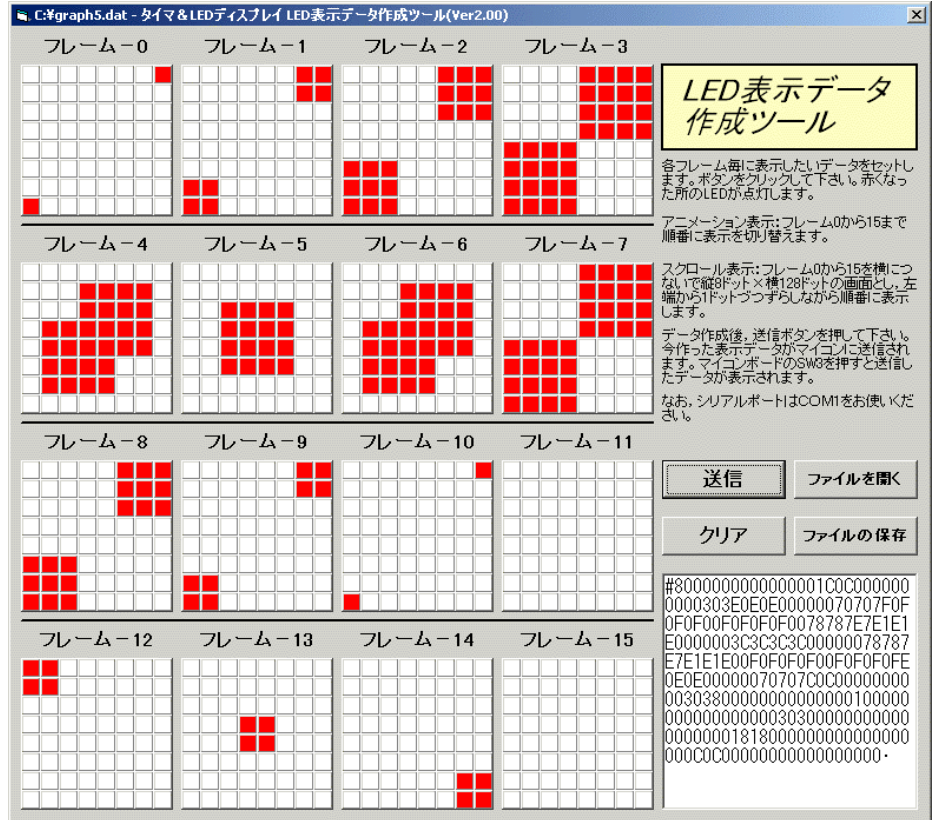
電源オン(または、TK-3687/TK-3687miniのリセットスイッチ‘SW1’を押す)のときに、「タイマ&LEDディスプレイ」のどのスイッチが押されているかで4種類の違うプログラムがスタートします。

#### 1. グラフィック&メロディ(何も押さないで電源オン)

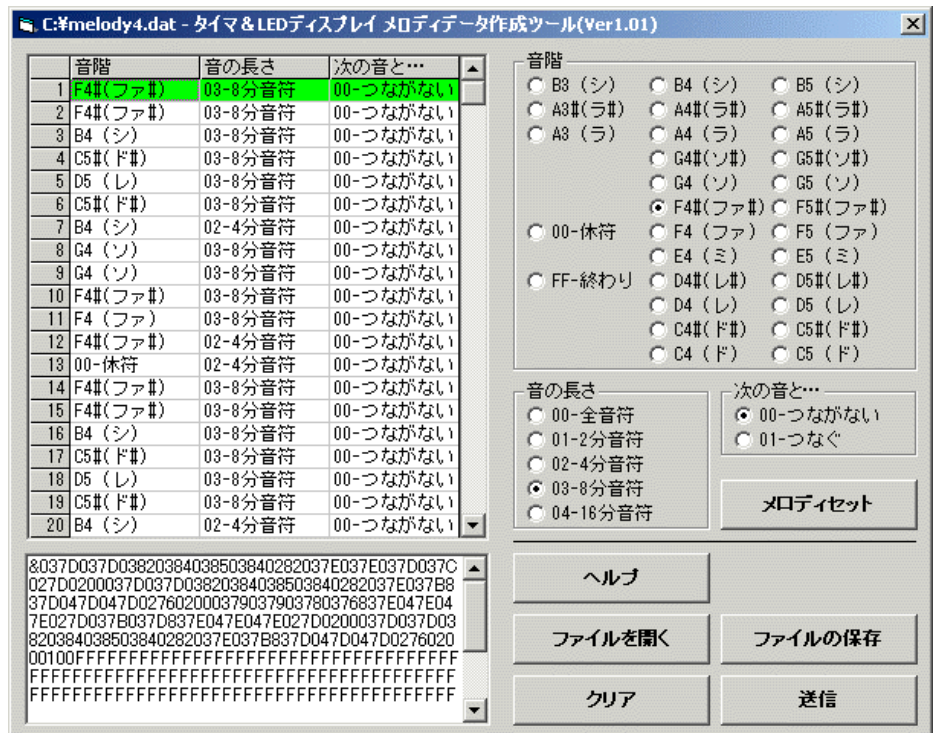
SW1, 2, 3 を押すとそれぞれにダウンロードされている表示データを表示しメロディデータを演奏します。選択されているデータと同じスイッチを押すと、スクロール表示とアニメーション表示を切り替えます。



SW3 のデータ 3 はパソコンで作成し登録します。パソコンのシリアルポート ( COM1 ) と TK-3687/TK-3687mini をシリアルケーブルでつなぎます。表示データの作成は付属の「 (CD-ROM) : ¥TK-3687mini¥ソフトウェア ¥LED¥vb¥txgraph.exe 」を実行してください。表示したいデータを作成し「送信」ボタンをクリックすると転送され表示されます。なお、データを全消去するときには「クリア」ボタンを押したあと、「送信」ボタンをクリックします。



また、メロディデータの作成は付属の「 (CD-ROM) : ¥TK-3687mini¥ソフトウェア ¥LED¥vb¥txmelody.exe 」を実行してください。演奏したいデータを作成し「送信」ボタンをクリックすると転送され演奏を開始します。なお、データを全消去するときには「クリア」ボタンを押したあと、「送信」ボタンをクリックします。



なお、パソコンからマイコンへどのようなデータを送信しているかは、付録の「転送フォーマット」をご覧ください。

## 2. 99 秒タイマ(SW1 を押しながら電源オン)

秒の設定は SW2 を押すとプラス 1 秒, SW3 を押すとマイナス 1 秒します。設定したら SW1 を押して下さい。減算が始まります。0 秒になるとブザーが鳴って知らせます。何かスイッチを押すとブザーが鳴り止んで設定した秒に戻ります。

## 3. 99 分タイマ(SW2 を押しながら電源オン)

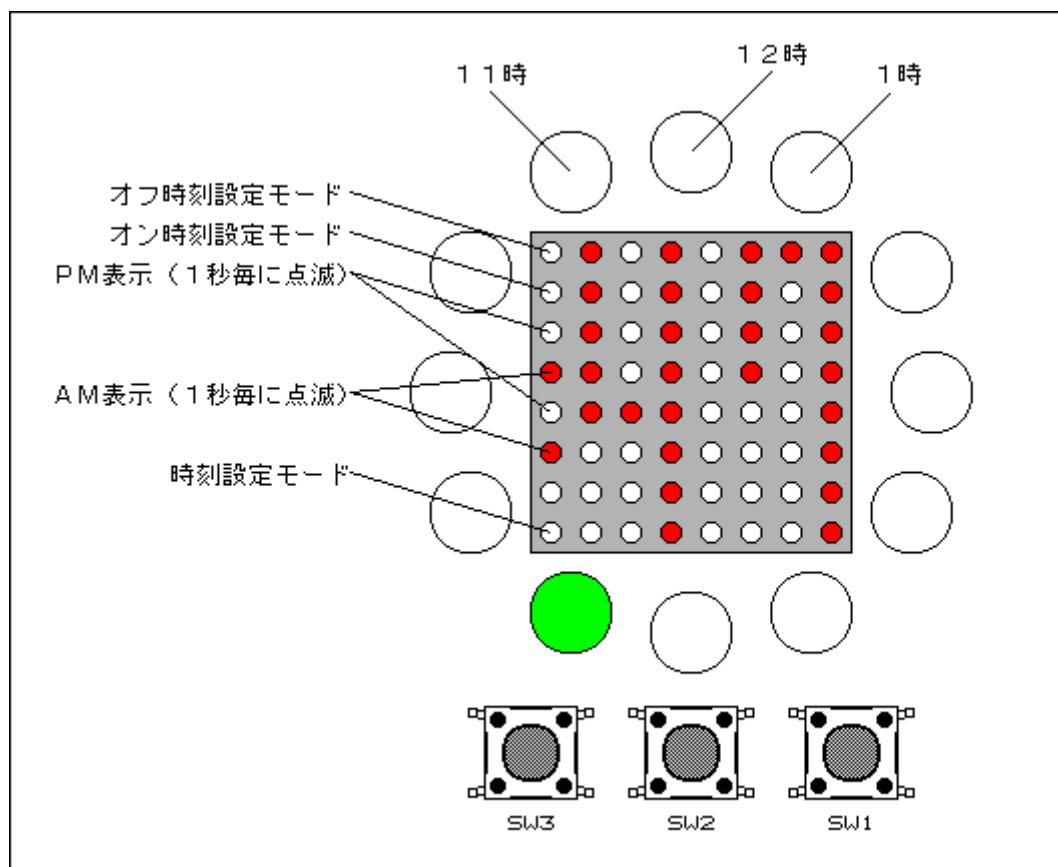
分の設定は SW2 を押すとプラス 1 分, SW3 を押すとマイナス 1 分します。設定したら SW1 を押して下さい。減算が始まり, 1 秒毎に数値が点滅します。残り 10 秒をきると秒表示になります。そして, 0 秒になるとブザーが鳴って知らせます。何かスイッチを押すとブザーが鳴り止んで設定した分に戻ります。

## 4. デジタル時計(SW3 を押しながら電源オン)

SW3 を押すと, 現在時刻→時刻設定モード→オン時刻設定モード→オフ時刻設定モード→現在時刻・・・と切り替わります。

SW1 を押すと「時」が+1, SW2 を押すと「分」が+1 されます。ただし, 現在時刻を表示しているときに SW1 を押すとしばらくの間「秒」を表示します。

現在時刻がオン時刻からオフ時刻までの間, ポート7において P70=Low, P71=High になります。それ以外は P70=High, P71=Low になります。また, オン時刻になると 1 秒間ブザーが鳴ります。

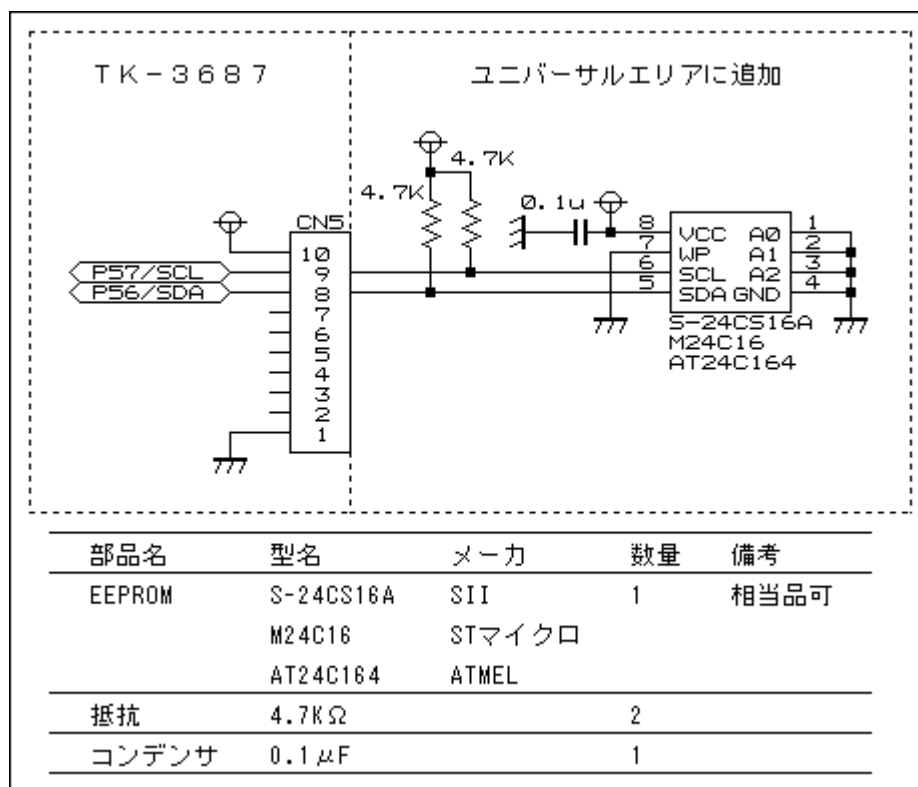


ソースファイルは CD-R をご覧下さい。(timer\_led. c)

## ■ バックアップ回路の追加(TK-3687のみ)

TK-3687 はバックアップ機能がないため、ダウンロードした表示データやメロディデータ、設定したタイマの時間などは、電源をオフしたりマイコンをリセットしたりすると消えてしまいます。一方、TK-3687mini は EEPROM が実装されているため、これらのデータを保持することができます。

そこで、TK-3687 にも EEPROM を追加して設定データをバックアップできるようにしましょう。回路図は次のとおりです。



ここで使用している EEPROM は I<sup>2</sup>C バスに接続するタイプで、16K ビット(2048×8 ビット)の容量があります。H8/3687 には I<sup>2</sup>C コントローラが内蔵されているので、比較的簡単に I<sup>2</sup>C デバイスを接続できます。

# 付録



## 動作チェックプログラム, “check. c”のソースリスト

### ソースファイル(check.c)

```
/*
 *
 * FILE      :check.c
 * DATE      :Thu, Jun 30, 2005
 * DESCRIPTION :Main Program
 * CPU TYPE  :H8/3687
 *
 * This file is programmed by TOYO-LINX Co.,Ltd. / yKikuchi
 */
/*****

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

/*****

          定数の定義 (直接指定)
*****/
#define      DRV_LOGIC 0x300 //ドライバの入力論理
                //負論理入力のビットを ' 1 ' にする

#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

/*****

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

//テストアニメーションデータ
const unsigned char AnimeData[][8] = {
    {0x81,0x00,0x00,0x00,0x00,0x00,0x00,0x81}, // 0
    {0x00,0x42,0x00,0x00,0x00,0x00,0x42,0x00}, // 1
    {0x00,0x00,0x24,0x00,0x00,0x24,0x00,0x00}, // 2
    {0x00,0x00,0x00,0x18,0x18,0x00,0x00,0x00}, // 3
    {0x00,0x00,0x3c,0x24,0x24,0x3c,0x00,0x00}, // 4

    {0x00,0x7e,0x42,0x42,0x42,0x7e,0x00}, // 5
    {0xff,0x81,0x81,0x81,0x81,0x81,0x81,0xff}, // 6
    {0xfd,0x01,0x81,0x81,0x81,0x81,0x80,0xbf}, // 7
    {0xf9,0x01,0x01,0x81,0x81,0x80,0x80,0x9f}, // 8
    {0xf1,0x01,0x01,0x01,0x80,0x80,0x80,0x8f}, // 9

    {0xe1,0x01,0x01,0x00,0x00,0x80,0x80,0x87}, //10
    {0xc1,0x01,0x00,0x00,0x00,0x00,0x80,0x83}, //11
    {0x81,0x00,0x00,0x00,0x00,0x00,0x00,0x81}, //12
    {0x83,0x80,0x00,0x00,0x00,0x00,0x01,0xc1}, //13
    {0x83,0x84,0x40,0x00,0x00,0x02,0x21,0xc1}, //14

    {0x83,0x84,0x48,0x20,0x04,0x12,0x21,0xc1}, //15
    {0x83,0x84,0x48,0x38,0x1c,0x12,0x21,0xc1}, //16
}
```

```

    {0x02,0x84,0x48,0x38,0x1c,0x12,0x21,0x40}, //17
    {0x00,0x04,0x48,0x38,0x1c,0x12,0x20,0x00}, //18
    {0x00,0x00,0x08,0x38,0x1c,0x10,0x00,0x00}, //19

    {0x00,0x00,0x00,0x18,0x18,0x00,0x00,0x00}, //20
    {0x00,0x00,0x18,0x24,0x24,0x18,0x00,0x00}, //21
    {0x00,0x3c,0x42,0x42,0x42,0x42,0x3c,0x00}, //22
    {0x3c,0x42,0x81,0x81,0x81,0x81,0x42,0x3c}, //23
    {0x42,0x81,0x00,0x00,0x00,0x00,0x81,0x42}, //24

    {0x81,0x00,0x00,0x00,0x00,0x00,0x00,0x81}, //25
    {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00} //26
};

//テストスクロールデータ
const unsigned char ScrollData[] = {
    0x80,0x40,0x20,0x10,0x08,0x04,0x02,0x01,0x02,0x04, // 0- 9
    0x08,0x10,0x20,0x40,0x80,0xc0,0xe0,0xf0,0xf8,0xfc, // 10- 19
    0xfe,0xff,0x7f,0x3f,0x1f,0x0f,0x07,0x03,0x01,0x80, // 20- 29
    0xc0,0xe0,0xf0,0xf8,0xfc,0xfe,0xff,0x7f,0x3f,0x1f, // 30- 39
    0x0f,0x07,0x03,0x01,0x00,0x3e,0x51,0x49,0x45,0x3e, // 40- 49
    0x00,0x00,0x42,0x7f,0x40,0x00,0x00,0x42,0x61,0x51, // 50- 59
    0x49,0x46,0x00,0x21,0x41,0x45,0x4b,0x31,0x00,0x18, // 60- 69
    0x14,0x12,0x7f,0x10,0x00,0x27,0x45,0x45,0x45,0x39, // 70- 79
    0x00,0x3c,0x4a,0x49,0x49,0x30,0x00,0x01,0x71,0x09, // 80- 89
    0x05,0x03,0x00,0x36,0x49,0x49,0x49,0x36,0x00,0x06, // 90- 99
    0x49,0x49,0x29,0x1e,0x00,0x01,0x01,0x02,0x02,0x04, //100-109
    0x04,0x08,0x08,0x10,0x10,0x20,0x20,0x40,0x40,0x80, //110-119
    0x80,0x40,0x20,0x10,0x08,0x04,0x02 //120-126
};

/*****
グローバル変数の定義とイニシャライズ(RAM)
*****/
unsigned char DisplayNo = 0;

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

unsigned char ScrollCnt = 0;
unsigned char AnimeCnt = 0;
unsigned int RouletteData = 0x0003;

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

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

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

/*****
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
};

/*****
メインプログラム
*****/
void main(void)
{
    // ポートイニシャライズ -----
    IO.PCR3      = 0xff; //ポート3,P30-37出力
    IO.PDR3.BYTE = 0x00 ^ DRV_LOGIC;

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

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

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

```

```

//タイマZイニシャライズ -----
TZ.TSTR.BYTE = 0x00; //TCNT0,1 停止
TZ0.TCR.BYTE = 0x21; //GRAのコンペアマッチでTCNT=0, /2
TZ0.TIORA.BYTE = 0x00; //GRAはアウトプットコンペアレジスタ
//コンペアマッチによる出力禁止

TZ0.TSR.BYTE = 0x00; //割込みフラグクリア
TZ0.TIER.BYTE = 0x01; //コンペアマッチインターラプトイネーブルA
TZ0.GRA = 0xffff; //メロディなしのときは6.5535msで割込みをかける
TZ0.TCNT = 0x0000; //TCNT0=0
TZ.TSTR.BYTE = 0x01; //TCNT0 カウントスタート

// メインループ -----
while(1){
    switch_in();
    if ((SwData4 & 0x08)!=0){ //SW1が押されたらメロディ出力指示
        if (MelodyFlag==0) {GakufuPnt = Gakufu_0; MelodyFlag = 1;}
        else {MelodyFlag = 0;}
        SwData4 = SwData4 & 0xf7;
    }
    if (MelodyFlag==4) {MelodyFlag = 0;}

    if ((SwData4 & 0x30)!=0){ //SW2かSW3が押されたら次の表示パターンへ
        DisplayNo++; if (DisplayNo>1) {DisplayNo = 0;}
        SwData4 = SwData4 & 0xcf;
    }

    switch (DisplayNo){
        //マトリックスLED表示 アニメーションデータ
        case 0:
            DispBuf[0] = AnimeData[AnimeCnt][0];
            DispBuf[1] = AnimeData[AnimeCnt][1];
            DispBuf[2] = AnimeData[AnimeCnt][2];
            DispBuf[3] = AnimeData[AnimeCnt][3];
            DispBuf[4] = AnimeData[AnimeCnt][4];
            DispBuf[5] = AnimeData[AnimeCnt][5];
            DispBuf[6] = AnimeData[AnimeCnt][6];
            DispBuf[7] = AnimeData[AnimeCnt][7];
            AnimeCnt++;
            if (AnimeCnt>26) {AnimeCnt = 0;}
            break;
        //マトリックスLED表示 スクロールデータ
        case 1:
            DispBuf[0] = ScrollData[ScrollCnt];
            DispBuf[1] = ScrollData[ScrollCnt+1];
            DispBuf[2] = ScrollData[ScrollCnt+2];
            DispBuf[3] = ScrollData[ScrollCnt+3];
            DispBuf[4] = ScrollData[ScrollCnt+4];
            DispBuf[5] = ScrollData[ScrollCnt+5];
            DispBuf[6] = ScrollData[ScrollCnt+6];
            DispBuf[7] = ScrollData[ScrollCnt+7];
            ScrollCnt++;
            if (ScrollCnt>119) {ScrollCnt = 0;}
            break;
    }

    //ルーレット表示
    DispBuf[8] = (unsigned char)(RouletteData & 0x00ff);
    DispBuf[9] = (unsigned char)(RouletteData / 0x0100);
    RouletteData = RouletteData << 1;
    if ((RouletteData & 0x1000)!=0) {RouletteData = (RouletteData | 0x0001) & 0xffff;}
}

```

```

        wait();

        //ルーレット表示
        DispBuf[8] = (unsigned char)(RouletteData & 0x00ff);
        DispBuf[9] = (unsigned char)(RouletteData / 0x0100);
        RouletteData = RouletteData << 1;
        if ((RouletteData & 0x1000)!=0) {RouletteData = (RouletteData | 0x0001) & 0x0fff;}

        wait();
    }
}

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

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

    //表示を消す
    IO.PDR5.BYTE = (IO.PDR5.BYTE & 0xfc) | (0x00 ^ (DRV_LOGIC/0x100));
    IO.PDR3.BYTE = 0x00 ^ DRV_LOGIC;
    IO.PDR6.BYTE = 0xff;

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

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

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

/*****

```

```

音階テーブル
{(タイマZのGRAにセットする値),(全音符の長さ,割込回数)}
*****/
const unsigned int OnkaiTbl[][2] = {
    {22727, 882},          //A ,ラ ,220.00Hz
    {21452, 934},          //A# ,ラ# ,233.08Hz
    {20248, 988},          //B ,シ ,246.94Hz

    {19111,1048},          //C ,ド ,261.63Hz
    {18039,1110},          //C# ,ド# ,277.18Hz
    {17026,1176},          //D ,レ ,293.66Hz
    {16070,1246},          //D# ,レ# ,311.13Hz
    {15169,1320},          //E ,ミ ,329.63Hz
    {14317,1398},          //F ,ファ ,349.23Hz
    {13514,1480},          //F# ,ファ# ,369.99Hz
    {12755,1570},          //G ,ソ ,392.00Hz
    {12039,1662},          //G# ,ソ# ,415.30Hz
    {11364,1760},          //A ,ラ ,440.00Hz          // 基準音
    {10726,1866},          //A# ,ラ# ,466.16Hz
    {10124,1976},          //B ,シ ,493.88Hz

    { 9556,2094},          //C ,ド ,523.25Hz
    { 9019,2218},          //C# ,ド# ,554.37Hz
    { 8513,2350},          //D ,レ ,587.33Hz
    { 8035,2490},          //D# ,レ# ,622.25Hz
    { 7584,2638},          //E ,ミ ,659.26Hz
    { 7159,2794},          //F ,ファ ,698.46Hz
    { 6757,2960},          //F# ,ファ# ,739.99Hz
    { 6378,3136},          //G ,ソ ,783.99Hz
    { 6020,3324},          //G# ,ソ# ,830.61Hz
    { 5682,3520},          //A ,ラ ,880.00Hz
    { 5363,3730},          //A# ,ラ# ,932.33Hz
    { 5062,3952},          //B ,シ ,987.77Hz
};

/*****
    タイマZ チャンネル0 割込み
*****/
#pragma regsave (intprog_tmz0)
void intprog_tmz0(void)
{
    unsigned char  Onkai;
    unsigned char  Onpu;
    unsigned char  Kyufu = 0;

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

    //メロディ
    switch (MelodyFlag){
        //停止 -----
        case 0:
            TZ0.GRA = 0xffff;    //メロディなしのときは6.5535msで割込みをかける
            IO.PDR5.BIT.B2 = 1; //サウンドオフ
            break;
        //音符情報取得 -----
        case 1:
            if (*GakufuPnt==0xffff){ //楽譜終了
                IO.PDR5.BIT.B2 = 1; //サウンドオフ
                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

//音階にあわせてタイマZのカウンタ値をセットする, 割込み間隔の調整
TZ.TSTR.BIT.STRO = 0;
TZO.GRA = OnkaiTbl[Onkai - (0x80 - KIJUN_ON)][0];
TZO.TCNT = 0x0000;
TZ.TSTR.BIT.STRO = 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:
    IO.PDR5.BIT.B2 = ~IO.PDR5.BIT.B2; //サウンド出力反転
    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:
    IO.PDR5.BIT.B2 = 1; //サウンドオフ
    KyufuCnt--;
    if (KyufuCnt==0) {MelodyFlag = 1;} //終了したら次の音符に移る
    break;
}
}

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

    for (i=0; i<166666; i++){
}

```

## ソースファイル(intprg.c)

```
/*
*****
/* FILE      : intprg.c
/* DATE      : Thu, Jun 30, 2005
/* DESCRIPTION : Interrupt Program
/* CPU TYPE  : H8/3687
/*
/* This file is generated by Renesas Project Generator (Ver.4.0).
/*
*****
*/

#include <machine.h>

extern void intprog_tmv(void);
extern void intprog_tmz0(void);

#pragma section IntPRG
// vector 1 Reserved

// vector 2 Reserved

// vector 3 Reserved

// vector 4 Reserved

// vector 5 Reserved

// vector 6 Reserved

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



```
// vector 21 Reserved

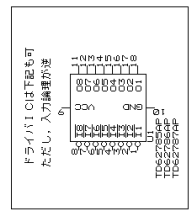
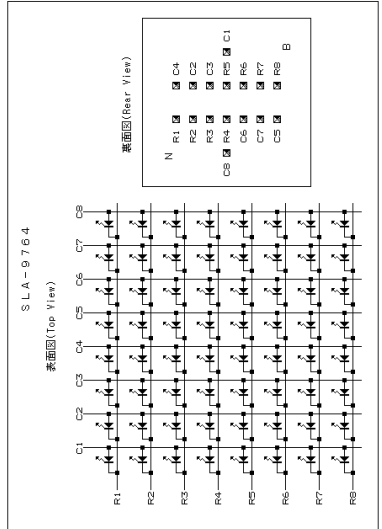
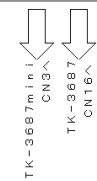
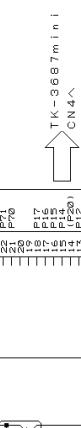
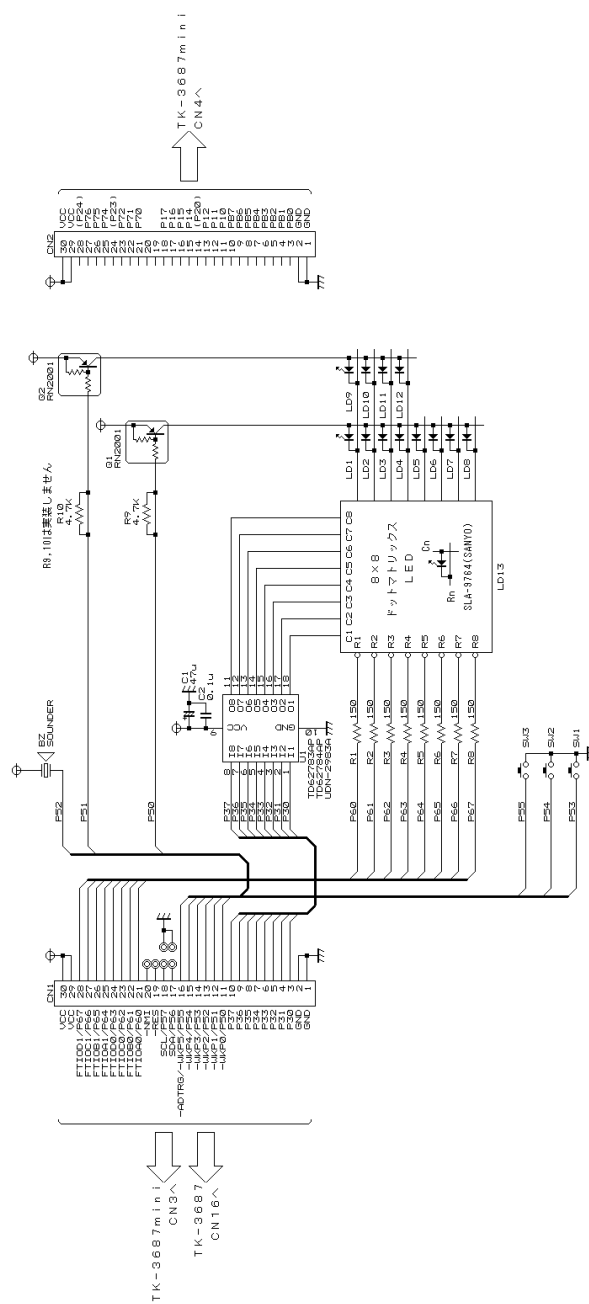
// vector 22 Timer V
__interrupt(vect=22) void INT_TimerV(void) {intprog_tmV();}
// vector 23 SCI3
__interrupt(vect=23) void INT_SCI3(void) {/* sleep(); */}
// vector 24 IIC2
__interrupt(vect=24) void INT_IIC2(void) {/* sleep(); */}
// vector 25 ADI
__interrupt(vect=25) void INT_ADI(void) {/* sleep();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

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

// vector 31 Reserved

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

# 回路図



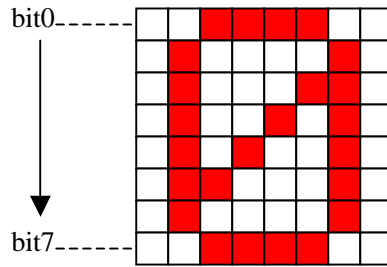
株式会社 東洋リンクス  
タイマ&LEDディスプレイ実習キット  
B6032  
2005-7-4  
1 / 1

## 転送フォーマット

「8 タイマ&LED ディスプレイへの応用」の中で、LED 表示データとメロディデータをパソコンからマイコンに送信しました。ここでは、データの転送フォーマットを説明します。

### ■ LED 表示データ

8ドット×8ドットを1フレームとし、フレーム-0~15までの16画面のデータをマイコンに送信します。1フレームのデータは右のようになります。



2進数	=	16進数	アスキーコード
00000000	=	00h	30h 30h
01111110	=	7Eh	37h 45h
10100001	=	A1h	41h 31h
10010001	=	91h	39h 31h
10001001	=	89h	38h 39h
10000101	=	85h	38h 35h
01111110	=	7Eh	37h 45h
00000000	=	00h	30h 30h

これを左から順番に並べると1フレームのデータになります。

30h, 30h, 37h, 45h, 41h, 31h, 39h, 31h, 38h, 39h, 38h, 35h, 37h, 45h, 30h, 30h

さらにフレーム-0~15まで並べ、ヘッダー(#, 23h)と終了コード(0Dh)を付けると完成です。このデータをフレーム-0とすると転送フォーマットは次のようになります。

ヘッダー							
# 23h							
フレーム-0							
左列							右列
30h : 30h	37h : 45h	41h : 31h	39h : 31h	38h : 39h	38h : 35h	37h : 45h	30h : 30h
フレーム-1							
左列							右列
H : L	H : L	H : L	H : L	H : L	H : L	H : L	H : L
↓							
フレーム-15							
左列							右列
H : L	H : L	H : L	H : L	H : L	H : L	H : L	H : L
終了コード							
0Dh		H : 2進数上位4ビットのアスキーコード L : 2進数下位4ビットのアスキーコード					

## ■ メロディデータ

一つの音符(休符も含む)を16ビットで表します。

ビット	機能	説明
15	次の音と...	0-つながない / 1-つなぐ(スラー, タイ, 付点音符で使用)
14	音の長さ	00-全音符(全休符)
13		01-2分音符(2分休符)
12		02-4分音符(4分休符)
11		03-8分音符(8分休符)
10		04-16分音符(16分休符)
9		
8		
7	音階	76-B3 (シ)    82-B4 (シ)    8E-B5 (シ)
6		75-A3# (ラ#)    81-A4# (ラ#)    8D-A5# (ラ#)
5		74-A3 (ラ)    80-A4 (ラ)    8C-A5 (ラ)
4		7F-G4# (ソ#)    8B-G5# (ソ#)
3		7E-G4 (ソ)    8A-G5 (ソ)
2		7D-F4# (ファ#)    89-F5# (ファ#)
1		7C-F4 (ファ)    88-F5 (ファ)
0		7B-E4 (ミ)    87-E5 (ミ)
		7A-D4# (レ#)    86-D5# (レ#)
		79-D4 (レ)    85-D5 (レ)
		78-C4# (ド#)    84-C5# (ド#)
		00-休符    77-C4 (ド)    83-C5 (ド)

例えば, ♪(8分音符)のA4(ラ)で次の音とつながないときは'0380'になります。これをアスキーコードに直して,「30h, 33h, 38h, 30h」で一つの音符を表します。

音符の数は128個固定です。それより音符が少ないメロディのときは残りの音符をFFFFhで埋めます(FFFFhはそれ以降メロディがないことを示す)。全てをアスキーコードにし,ヘッダー(&, 26h)と終了コード(0Dh)を付けると完成です。

ヘッダー															
'& 26h															
音符-1		音符-2		音符-3		音符-4									
A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
音符-5		音符-6		音符-7		音符-8									
A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
↓															
音符-125				音符-126				音符-127				音符-128			
A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
終了 コード															
0Dh															

A,B,C,D: アスキーコード

## ■ アスキーコード

これまでの転送フォーマットの説明の中で、アスキーコードという表現がでてきました。アスキーコードとは何でしょうか。

データ転送を行なう場合、文字を表すためのコード(文字に数字を割り当てたもの)が必要になります。コード体系には何種類かありますが、ASCII コードはアメリカの国内規格である ANSI 規格をもとに作成された、アルファベットや記号を表すことができる 7 ビットのコードです(128 文字まで表せる)。

日本ではアルファベットだけでは不便なので、ASCII コードにプラス  $\alpha$  する形でカタカナも表せるようにしました。その方法の一つが 8 ビットのコード体系にすることで、256 文字まで表せます。この規格は JIS で規定されており、「ローマ文字・カタカナ用 8 単位符号」と呼ばれています。通常、アスキーコードというと、このコード体系のことを指しているようです(通称のため、あいまいなことがある)。

ハイパーモニターで使用する「ハイパーターミナル」は、アスキーコードを受信するとその文字を表示し、キーボードを叩くとその文字のアスキーコードを送信します。

アスキーコード表は次のとおりです。

		上位 4 ビット															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
下 位 4 ビ ット	0	NUL	DLE	SP	0	@	P	`	p				-	タ	ミ		
	1	SOH	DC1	!	1	A	Q	a	q			。	ア	チ	ム		
	2	STX	DC2	“	2	B	R	b	r			「	イ	ツ	メ		
	3	ETX	DC3	#	3	C	S	c	s			」	ウ	テ	モ		
	4	EOT	DC4	\$	4	D	T	d	t			、	エ	ト	ヤ		
	5	ENQ	NAK	%	5	E	U	e	u			。	オ	ナ	ユ		
	6	ACK	SYN	&	6	F	V	f	v			ヲ	カ	ニ	ヨ		
	7	BEL	ETB	‘	7	G	W	g	w			ア	キ	ヌ	ラ		
	8	BS	CAN	(	8	H	X	h	x			イ	ク	ネ	リ		
	9	HT	EM	)	9	I	Y	i	y			ウ	ケ	ノ	ル		
	A	LF	SUB	*	:	J	Z	j	z			エ	コ	ハ	レ		
	B	VT	ESC	+	;	K	[	k	{			オ	サ	ヒ	ロ		
	C	FF	FS	,	<	L	¥	l				ヤ	シ	フ	ワ		
	D	CR	GS	-	=	M	]	m	}			ユ	ス	ハ	ソ		
	E	SO	RS	.	>	N	^	n	~			ヨ	セ	ホ	。		
	F	SI	US	/	?	O	_	o	DEL			ッ	ソ	マ	。		

上の表の黄色の欄は制御文字で、通信の制御や画面の制御に用いられます。また、空欄は未定義で、システムやデバイスによって任意に利用することができます。

ところで、日本語の場合、漢字も表現できなければいけません。そうすると当然 8 ビットでは足りなくて、16 ビットのコード体系を使うこととなります。これも規格が決まっています。興味のある方は調べてみてください。

## 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](mailto:toyolinx@va.u-netsurf.jp)

URL: <http://www2.u-netsurf.ne.jp/~toyolinx>

20051207