ユニバーサルタイマの製作

［8pinPICマイコンを使って見よう］ 第 2 段　　－ 仙台電子工作同好会 #133 －

’05,3,19

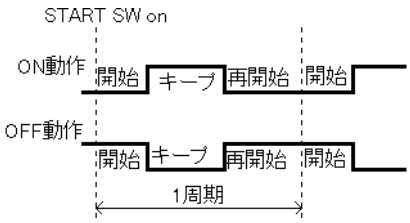
土合 靖

§はじめに

PICのデータROMに数値を書き込むことにより、動作時間(0.1 秒～年単位まで 0.1 秒刻み)、動作様式(単動作/連続動作、ON動作/OFF動作)を自由に設定できるタイマを製作する。これ1台でパルス発生器、オンディレータイマ、オフディレータイマ、低周波発振器、デイリータイマ、ウィークリータイマ等、広範囲な利用が可能である。8pinパッケージ+発振子(水晶)で済むので、タイマIC＋CRより小さく纏まる可能性もあり、コスト的にも大差はない。特に長時間動作においては、精度は言うに及ばず使用部品の面(CRの時定数で長時間タイマは困難)でも有利になる。精度が不要であれば、内蔵クロック(それでもCRよりは高精度)を使うことにより、外付け部品なしでもタイマができる。

事の切っ掛けは秋月のキット、「簡易PICシーケンサ」だった。「シーケンサ」に釣られて買ったのだが、要は上記の様な、動作時間と動作様式を設定できるタイマが4チャンネル付いている物だった。4チャンネルともスタートSW オンのタイミングを受けて動作を開始するようになっており、あるチャンネルの動作終了を受けて、別のチャンネルが動作を開始すると言った仕組みはないので「シーケンサ」と言うには些か気恥ずかしい物だった。ただ、パソコンのシリアルポートとつないで設定(データROM書換)可能であり、PICライタがなくても設定可能ではあるのだが、その分シリアルコネクタ等がつき結構、大きな基板になっていた。シーケンサと言っている所為か汎用性を狙った所為か、3端子Regを入れ24V作動を可能にし、PICでSSR程度は直接駆動できるにも拘わらずTRアレイ等も使われていた。4チャンネルを連係させながら制御したり、動作設定を頻繁に変えたりするケースはあまりないのではないかと言う思いもあり、「コンセプトは戯きで、思い切って機能を絞り込んだ物を作ってみるか」となった次第だ。

§動作説明



タイムチャートで説明する。「開始」「キープ」等の用語は「簡易PICシーケンサ」で使われているものをそのまま踏襲する。スタートSWオン後、タイマの動作開始までの時間を「開始時間」、タイマ動作を保持している時間を「キープ時間」と定義している。「再開始時間」はタイマ動作終了後、次のタイマ動作開始までの時間とするのが言葉としては自然なのだが、図に示す様、再開までの時間は「再開始時間」+「開始時間」となっている。従って、「開始時間」+「キープ時間」+「再開始時間」で1周期となる。

尚、単動作(ワンショット)の時は周期と言う概念はなく、「再開始時間」は意味を持たず、例えセットしても無視され動作に影響はない。タイマ作動時に出力がH(ハイ)になるのをON動作、L(ロー)になるのをOFF動作と称している。纏めると、以下の様になる。

・動作の種類を問わず、「キープ時間」=0とすると動作しない。
・単動作で「開始時間」に0以外の値を入れるとディレータイマとして働く。
・連続動作の場合は、通常「開始時間」または「再開始時間」の何れかを0にセットする。(両方に0以外の値を入れても不都合はないが、繰り返し周期が直感的には分らなくなり使い難い。両方とも0にするとタイマとして役に立たないのは言うまでもない。)

§「簡易 PIC シーケンサ」での設定方法

待機時(タイマ不作動状態)、パソコンからの信号を受けてデータROMを書き変える様になっている。パソコン画面上で「○○時間○○分○○秒○」の○部分に数字を入れることで、データを書き換える事ができる。実際のデータROMの使い方としては、ひとつの時間データに24ビット(=3バイト、PICのデータROMは8ビット幅なので3個分)を割り当てている。データの1が 0.1 秒に相当する。画面上では最大 99 時間59分59秒9まで設定できるのだが、ROM上の実データでは次のようになる。

0.9 秒→9

59秒→590

59分=59×60=3540秒→35400

99時間=99×3600=356400秒→3564000

∴3564000+35400+590+9=392399(10進表示)

上記数字を16進に変換すると5FCCFとなり、3バイト確保しておけば充分に収まることが分る。見方を変えて3バイトに収納できる最大データ、FFFFFFではどの位の時間に相当するかを計算すると

FFFFFFh=16777215d(hは16進、dは10進の意味)だから、1677721.5秒=466時間2分1秒5

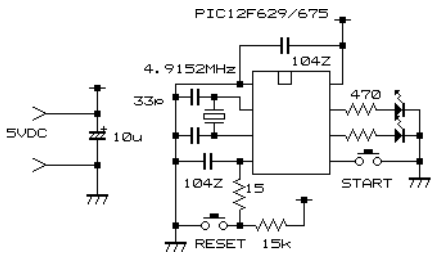
即ち20日弱となる。従って、「簡易 PIC シーケンサ」はパソコン画面からではなく、PICライタ等を使いデータROMを直接書き換えれば、プログラム変更なしで20日近い長時間タイマとして使える可能性がある。

データ収納の様子をもっと具体的に説明すると、下表のようになる。

ch1Su	ch1Sm	ch1Sl	ch2Su	ch2Sm	ch2Sl	ch3Su	ch3Sm	ch3Sl	ch4Su	ch4Sm	ch4Sl
ch1Ku	ch1Km	ch1Kl	ch2Ku	ch2Km	ch2Kl	ch3Ku	ch3Km	ch3Kl	ch4Ku	ch4Km	ch4Kl
ch1Ru	ch1Rm	ch1Rl	ch2Ru	ch2Rm	ch2Rl	ch3Ru	ch3Rm	ch3Rl	ch4Ru	ch4Rm	ch4Rl
action											

4個のタイマで夫々「開始時間」「キープ時間」「再開始時間」を持っており、それらの時間データに3バイトずつ割り当てられている。データROMの0番地から順に、チャンネル1開始時間(ch1S)上位バイト、中位バイト、下位バイト、次にch2Sの上中下、ch3Sの上中下、ch4Sの上中下の順に並び、以下同様にキープ時間(表2行目)、再開始時間(表3行目)データが配置されている。結局、3バイト×4チャンネル×3種類の時間で、合計36個のデータ(番地で言うと0～23h(=35d))が並ぶ。24h番地のデータ(action)はタイマの動作様式で、上位4ビットはチャンネル毎(bit7-ch4, bit6-ch3, bit5-ch2, bit4-ch1)のON/OFF動作(0:ON,1:OFF)の区別、bit1 が単/連続動作(0:単,1:連続)の区別を表している。25h番地のデータはパソコンとの通信時にチェック用に使われているようだが、今回寺子屋で取り上げる物では無関係なので触れない。

§今回の製作物



回路図を以下に示すが、至って簡単なものである。

8pinPICでは電源ピン以外の6本までポートとして使えるのだが、本機の場合、水晶発振子で2本、リセット端子(pin4)、スタートSW(pin5を入力ポートとして使う)に夫々1本ずつ取られるので、パソコンに依る書込を不採用としても、タイマは2チャンネル(GP0-pin7,GP1-pin6)しか取れない。しかしながら、2チャンネルで上手く行けば18pinPICで4チャンネルに拡張するのは難しくない(プログラムの僅かな修正でOK)し、1チャンネルで済む場合は余ったポートにセンサ信号を入れたり、停電検出に使ったりする余地があるので実用性は充分あると思う。

上記回路は機能確認用の言わば実験回路だから、電源は外付けの5Vとし、タイマ出力には目視確認用にLEDを接続しているが、実用回路ではSSRや小型リレーを直接駆動したり、TR等を介してより大きな負荷を駆動することになる。電源はきっちりと定電圧化された5Vである必要はなく、電池(3Vで充分働く)やAC100Vを整流してツェナダイオードでクランプした程度の簡易な物でも構わない。スタートやリセットをSW操作で人為的に与えているが、外部信号と連動させても良いことは言うまでもない。

§ソフトウェア(プログラム)

タイマ割込みを使い0.1秒毎に時間データをダウンカウントし、データが0になった時点で出力ポートを変化させる。割込み時間の精度がタイマの精度そのものになるので、水晶発振子に依るクロックを使っている。別紙に示すフローチャートとプログラム(C言語を使用)のソースファイルに沿って説明する。

電源投入時またはリセットSW操作時に、マイコンがリセットされプログラムの実行が始まる。フローチャートのメインルーチン「START」の部分である。実際にやることは「初期設定」で、ソースファイルでは182～192行に書かれた内容である。次の「データ初期値代入」だがデータROMから数値データを引き出して変数に入れる操作で

ある。メインルーチン中に実際の手続きを長々と書くとプログラムが分りにくく、また同じ作業を他の箇所でもやることから、関数(サブルーチン)として43～83行に定義してある。変数(start1 etc.)には32bit整数を用意した。

データROM中の時間データの配置だが、C言語では32bit変数が使えること、「簡易 PIC シーケンサ」のように24bitとすると、PICライタの画面でEEPROMの配列が1行当たり16個である関係上、どこが区切れか分り難くなることを勘案して1個の時間データに4バイト割り当てた。これにより1行目は「開始時間」4チャンネル分で占められ、行の途中から全く意味合いの違うデータに変わることがなくなり見易くなる。同様に2,3行目は夫々「キープ時間」、「再開始時間」のみで占められる。尚、本機ではタイマが2チャンネルしか取れないが、18pinPICで4チャンネルに拡張した時のプログラムとの共通化を狙い4チャンネル分確保している。4バイトの割り当てにより、扱える最大時間は以下の様に膨大なものとなる。(実用性は?)

FFFFFFFFh=4294967295d→429496729.5秒→4971日→13年

次の「割込カウンタセット」だが、源発振:4.9152MHz、プリスケーラ 1/32、TMR0フリーラン(00h→FFh カウントアップ FFh からオーバフローして 00h に戻る時点で割込み発生)に依る割込みだから、クロック4個分で1命令サイクルであることに注意すれば、

4915200Hz×(1/32)×(1/256)×(1/4)=150Hz

となり、150Hz の頻度で割込みが掛かることになる。時間データのダウンカウントは 0.1秒毎(=10Hz)だから、割込み15回中1回だけ減算処理をし、後の14回は何もせず次の割込みを待つ必要がある。そこでカウンタに初期値15をセットして割込みに備える。(ソースファイル197行)

次はスタートSWが押される(=ポート2がLになる)のを待つ。スタート信号が入ったら割込み許可を出し、割込みの受付を可能にする。(ソースファイル202行) その前の行でTMR0にFFhをセットしているが、これをしないとスタート信号が入ってから割込みが掛かるまでの時間が一律に決まらず、例えばS(開始時間)=0なら即座にタイマ出力が出なくてはならないのに、SWを押すタイミングで遅れ時間がバラ付くと言う不具合を生じる。割込み許可後、メインルーチンではアイドルフラグをチェックし、先に進まず待機する(アイドルフラグの初期値に1を与えているので)様になっている(ソースファイル207～209行)ので、ここから必ず割込み処理に飛ぶことになる。

割込み処理では、フローにある様に開始時にレジスタの退避、抜け出る時にレジスタの復旧をする必要があるが、C言語ではコンパイル時、自動的に当該操作を付けてくれるので、ソースファイルには記述する必要はない。レジスタ退避後、まずやることは割込みカウンタのチェックで、15の時は具体的な処理に進みそれ以外の時は何もしないが、何れの場合も次にカウンタを1減じる。カウンタの初期値は15だから、スタート後初めての割り込みでは必ず具体的処理の方に行くので、ポートの出力変化等が遅れず都合が良い。割込みが掛かる度に割込みカウンタは減算されていくので、やがて0になる。0になったら、その時点で初期値の15に戻す。これによりカウンタが15の時だけ具体的処置を実行し、残りの14回は何もしないという事になるので、150Hzの割り込み10Hzのダウンカウントが実現できる。

割込みから抜けるとメインルーチンの元の場所に戻って来るが、割込みで具体的処置をした時にはアイドルフラグが下ろされているので、戻った場所で足踏みをせずに先に進むことになる。ここで0になっていない時間データが残っていないか、必要な処置が終わっているか等をチェックする。終わっていなければ次の割込みを待ち、終わっていれば時間データを再セットする。その後、連続動作時は次の割込みから同じ事を繰り返し、単動作では割込みを禁止して次のスタート信号を待つ。

割り込み中の実質的な処理(=割込みカウンタが15の時)について説明する。メインルーチンに復帰した時、足踏みにならない様に、まずアイドルフラグを下ろす。次にS(開始時間)をチェックし、0でなければ1減じるだけで抜ける。0ならばK0フラグを見て、立っていれば何もせずK(キープ時間)のチェックに移る。K0フラグがクリアであればポート変化の後、Kチェックに移る。KチェックではSチェックと同様に、0でなければ1減じるだけで抜け、0なら停止アクション(変化させたポートを元に戻す)、K0フラグセットを経てR(再開始時間)チェックに移る。Rチェックでは0なら何もせず、0でなければ1減じて抜ける。

考え方の基本は「Sが0になった時点でポート変化させ、Kが0になった時点でポートを戻す」だが、K0フラグを考慮しないとKが既に0になっているにも拘わらず毎回「開始アクション」が起きる事になる。その後すぐに「停止アクション」を通過するので、ポートが変わり放しにはならないものの、一瞬(実測20 μs程度)ポートが変化し不

都合である。尚、Kが最初から0の時はデータROMから読み込む時点でK0フラグをセットし無用な動作を防いでいる。

§実用時の留意点他

ユニバーサルの名前どおり様々な使い方ができる。単動作ではスタート後直ちにキープ時間だけ出力を出す使い方の他、開始時間をセットすればディレー(遅延)タイマとして使える。何れの時間も極端な長時間が設定できるので、「1週間(1月、半年、1年・・・)後に何かをする」と言うような事も可能である。単動作の場合は再開始時間と言う概念は意味を持たないので、例えセットしても0と読み変える様にしてある。

反復動作の場合は、ウィンカやフラッシャのような比較的早い周期の用途の他、繰り返し周期を24時間や1週間としたデイリー、ウィークリタイマとしても使える。筆者の場合は以下の設定で曝気ポンプの間欠運転に使っている。

ch1:OFF動作　S＝0　K＝8時間　R＝16時間　　ポンプの制御

ch2:ON動作　S＝0　K＝0.2 秒　R＝0.8 秒　　LED点滅(タイマの動作表示)

10AM にスタートSWを押したので、10～18時休止、その後翌朝10時まで運転というサイクルで動いている。運転再開が18時の時報に完全に同期する必要があると言うような用途では使えないが、実用上は問題のない精度で動いている。OFF動作としたのは、曝気ポンプと言う負荷の性質上、停電等でリセットされた場合、復帰時はONにならないと不都合だからである。

複数のタイマの内、使う必要がないチャンネルは時間データに0を書き込んでおけば良い。追加のプログラムが必要だが、不要なポートにセンサ信号を入れたり、停電検出をしたりする、例えばCdSを使い暗くなったら点灯し、11PMには消灯すると言ったこともできる。

今回はC言語を使ったがソースファイルは簡単になったものの、32bit変数等を使っている所為か実際のプログラムはかなり大きくなり、1Kワードのプログラムメモリが殆ど満杯になった。ソースファイルの t1_read()関数等を見ると分るが、Rデータだけは3バイト分しか読んでいない。ここも4バイト読み込むとメモリ不足でコンパイルできなかったのだ。

最後に、ソフトウェアで作ったタイマ故、多少の遅れがある筈なので、単動作、S＝0、K=1(両チャンネルとも)の条件で、「スタートSWonからch1の立ち上がりまでの遅れ」と「ch1の立ち上がりからch2の立ち上がり迄の遅れ」を実測した。

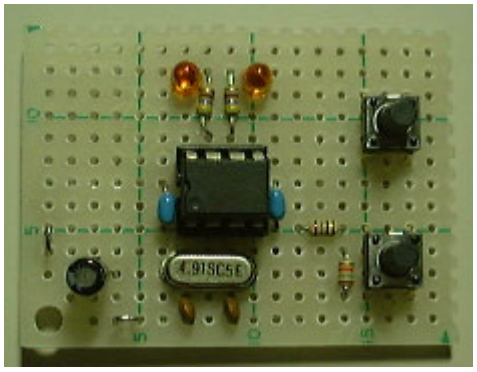
Start～ch1　　50 μs(C言語)　　19 μs(アセンブラ)

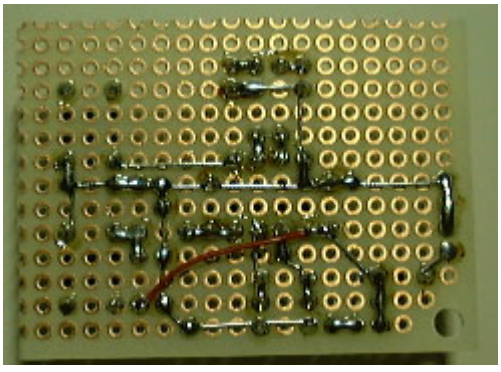
ch1～ch2　　25 μs(C言語)　　18 μs(アセンブラ)

C言語とアセンブラで、それほど大きな差は出なかった。50 μs にしてもパルス幅の 0.1sに比べれば 1/2000 であり、取るに足らない量と言って差し支えないだろう。

§基板

今回は部品点数が少ないので、写真で済ませる事を了承願いたい。

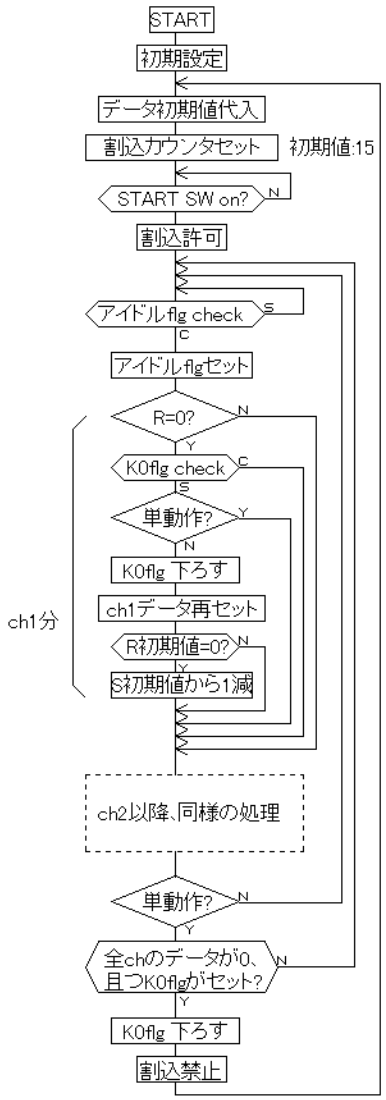




上:スタートSW

下:リセットSW

メインルーチン



------- SW操作待ち

------- Idle Flagが0の間は

何もせずに次の割込を待つ

150Hzの頻度で割込が掛かるが
0.1秒毎の減算なので、15回の内
14回は何もしない

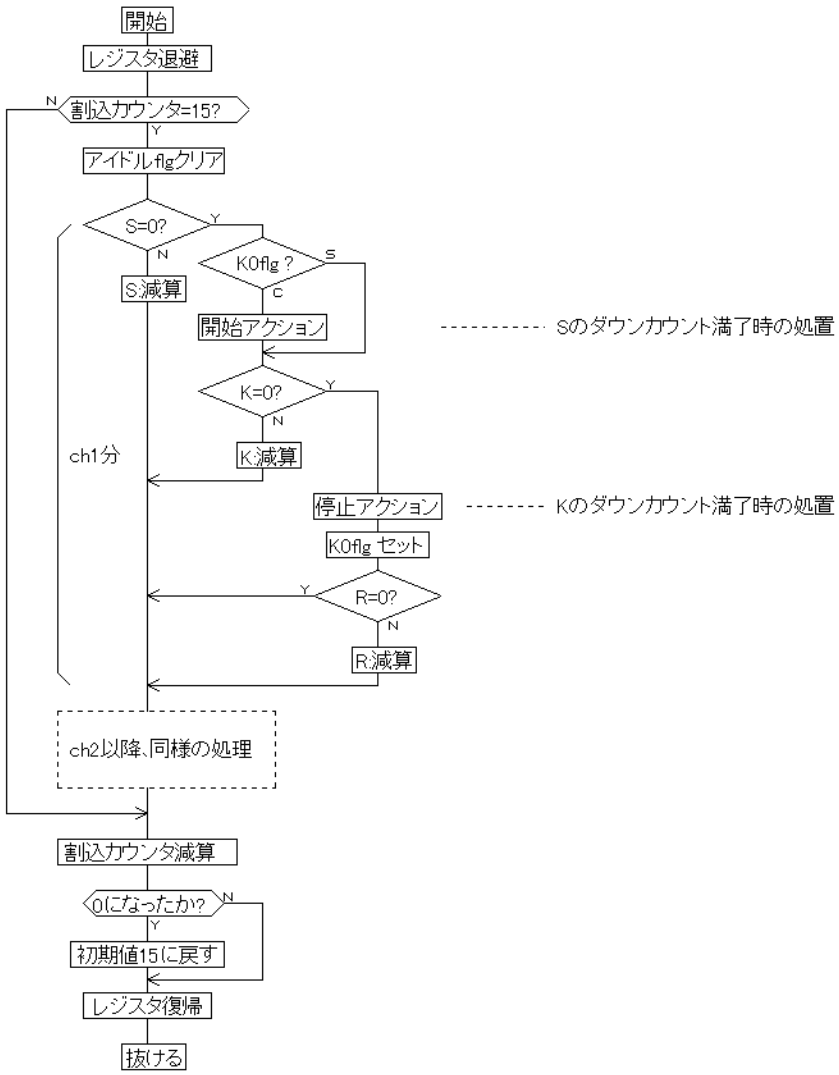
S:set ビットが1

C:clear ビットが0

Y:yes

N:no

割込処理



---------- Sのダウンカウント満了時の処置

-------- Kのダウンカウント満了時の処置

ユニバーサルタイマフローチャート

2005, 3, 12
L. C. D. R&D

```
/*****************************************************************
  ユ ニ バ ー サ ル タ イ マ 制 御 プ ロ グ ラ ム
                                    2CH.出力 PIC12F629使用

    U_tim12f.c                                  L.C.D.R&D
                                                Mar. 13,2005
*****************************************************************/

#include    <12f629.h>
#fuses HS,NOWDT,MCLR,NOBROWNOUT
#use fast_io(A)

/*ポート割り当て
  0:タイマ1 (OUT) active H
  1:タイマ2 (OUT) active H
  2:START SW (IN) active L
  3:/MCLR RESET SW
  4:X'tal 4.9152MHz
  5: - do. -                */

/************** 変数領域の確保 **************/
int32 start1;           //CH1 start data
int32 start2;           //CH2 start data
int32 keep1;            //CH1 keep data
int32 keep2;            //CH2 keep data
int32 restart1;         //CH1 restart data
int32 restart2;         //CH2 restart data
int action;             //bit(7,6,5,4):CH(4,3,2,1) 1=OFF動作 0=ON動作
                        //bit1:1=繰返し動作 0=単動作  bit3,2,0:××
int int_cnt;            //割込み回数カウンタ
int1 k0_flg1 = 0;       //K(keep data)が0の時セット
int1 k0_flg2 = 0;
int1 idle_flg = 1;      //1の時データチェックをせず割込み待ち

/************** EEPROM DATA *************/
#rom 0x2100 =
    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,   //s data 1～4ch 各4バイト 左側 上位バイト
     0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,   //k data 1～4ch 各4バイト 左側 上位バイト
     0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,   //r data 1～4ch 各4バイト 左側 上位バイト
     0} /* CH4,CH3,CH2,CH1,×,×,1/0,×:上位4ビット 1=OFF動作 0=ON動作
                        ;bit1 1=反復動作 0=単動作                */

void t1_read()          //Timer1 data(start/keep/restart)COPY
{
    start1 = 0x1000000*(read_eeprom(0x00)) + 0x10000*(read_eeprom(0x01))
                       + 0x100*(read_eeprom(0x02)) + read_eeprom(0x03);
    keep1 = 0x1000000*(read_eeprom(0x10)) + 0x10000*(read_eeprom(0x11))
                       + 0x100*(read_eeprom(0x12)) + read_eeprom(0x13);
    if (keep1 == 0)
    {
        k0_flg1 = 1;
    }
    if (bit_test(action,1))
    {
        restart1 = 0x10000*(read_eeprom(0x21))  // 0x1000000*(read_eeprom(0x20)) +
                           + 0x100*(read_eeprom(0x22)) + read_eeprom(0x23);
    }
    else                    //単動作時は強制的に0にする
    {
        restart1 = 0;
    }
}

void t2_read()          //Timer2 data(start/keep/restart)COPY
{
    start2 = 0x1000000*(read_eeprom(0x04)) + 0x10000*(read_eeprom(0x05))
                       + 0x100*(read_eeprom(0x06)) + read_eeprom(0x07);
    keep2 = 0x1000000*(read_eeprom(0x14)) + 0x10000*(read_eeprom(0x15))
                       + 0x100*(read_eeprom(0x16)) + read_eeprom(0x17);
    if (keep2 == 0)
    {
        k0_flg2 = 1;
    }
    if (bit_test(action,1))
    {
        restart2 = 0x10000*(read_eeprom(0x25))  // 0x1000000*(read_eeprom(0x24)) +
                           + 0x100*(read_eeprom(0x26)) + read_eeprom(0x27);
    }
    else                    //単動作時は強制的に0にする
    {
```

```
 81         restart2 = 0;
 82     }
 83 }
 84
 85 //************** 割り込み処理 **************
 86 #int_timer0
 87 void dwncount()
 88 {
 89     if (int_cnt == 15)
 90     {
 91         idle_flg = 0;                              //アイドルフラグクリア
 92         if (start1 == 0)                           //1ch.の処理
 93         {
 94             if (k0_flg1 == 0)                      //K0flagセット時はPORT変化を避ける
 95             {
 96                 if (bit_test(action,4))
 97                 {
 98                     output_low(PIN_A0);
 99                 }
100                 else
101                 {
102                     output_high(PIN_A0);
103                 }
104             }
105             if (keep1 == 0)
106             {
107                 if (bit_test(action,4))
108                 {
109                     output_high(PIN_A0);
110                 }
111                 else
112                 {
113                     output_low(PIN_A0);
114                 }
115                 k0_flg1 = 1;                       //K0flagセット
116                 if (restart1 != 0)
117                 {
118                     restart1--;
119                 }
120             }
121             else
122             {
123                 keep1--;
124             }
125         }
126         else
127         {
128             start1--;
129         }
130         if (start2 == 0)                           //2ch.の処理
131         {
132             if (k0_flg2 == 0)                      //K0flagセット時はPORT変化を避ける
133             {
134                 if (bit_test(action,5))
135                 {
136                     output_low(PIN_A1);
137                 }
138                 else
139                 {
140                     output_high(PIN_A1);
141                 }
142             }
143             if (keep2 == 0)
144             {
145                 if (bit_test(action,5))
146                 {
147                     output_high(PIN_A1);
148                 }
149                 else
150                 {
151                     output_low(PIN_A1);
152                 }
153                 k0_flg2 = 1;                       //K0flagセット
154                 if (restart2 != 0)
155                 {
156                     restart2--;
157                 }
158             }
159             else
160             {
```

```
161                 keep2--;
162             }
163         }
164         else
165         {
166             start2--;
167         }
168     }
169     int_cnt--;
170     if (int_cnt == 0)
171     {
172         int_cnt = 15;
173     }
174 }
175
176 /***********************************************
177     メ イ ン ル ー チ ン
178 ***********************************************/
179 void main()
180 {
181     int init_port;                                  //ポート初期状態
182     setup_comparator(NC_NC_NC_NC);                  //コンパレータ不使用
183 /* 12F675を使う場合 ----------------------------
184     setup_adc_ports(NO_ANALOGS);                    //A/D不使用
185   ----------------------------------------------*/
186     set_tris_a(0b100);                              //GP2入力､GP0,1出力
187     port_a_pullups(0b100);                          //内部pull up 使用
188     setup_timer_0(RTCC_INTERNAL | RTCC_DIV_32);     //TMR0プリスケーラ(1/32)
189     enable_interrupts(INT_RTCC);                    //TMR0割込み使用
190     action = read_eeprom(0x30);                     //action data(ON/OFF,単/連)COPY
191     init_port = (action >> 4);                      //上位4bitが初期値ゆえ､右シフト
192     output_a(init_port);                            //ポートの初期状態決定
193     while(1)
194     {
195         t1_read();                                  //データ初期値読込
196         t2_read();
197         int_cnt = 15;                               //割込カウンタ初期化
198         while (input(PIN_A2))                       //Start SW操作待ち
199         {
200         }
201         set_timer0(0xff);
202         enable_interrupts(GLOBAL);                  //割込み許可
203         do
204         {
205             do
206             {
207                 while (idle_flg == 1)               //アイドルフラグセット時は
208                 {                                   //何もせず割込み待ち
209                 }
210                 idle_flg = 1;                       //アイドルフラグセット
211                 if (restart1 == 0)
212                 {
213                     if (k0_flg1 == 1)
214                     {
215                         if (bit_test(action,1))
216                         {
217                             k0_flg1 = 0;            //K0flag下ろす
218                             t1_read();              //データ再読込
219                             if (restart1 == 0)      //R初期値が0の時は
220                             {                       //S初期値から1減じる
221                                 start1--;
222                             }
223                         }
224                     }
225                 }
226                 if (restart2 == 0)
227                 {
228                     if (k0_flg2 == 1)
229                     {
230                         if (bit_test(action,1))
231                         {
232                             k0_flg2 = 0;            //K0flag下ろす
233                             t2_read();              //データ再読込
234                             if (restart2 == 0)      //R初期値が0の時は
235                             {                       //S初期値から1減じる
236                                 start2--;
237                             }
238                         }
239                     }
240                 }
```

```
241             }
242             while (bit_test(action,1));             //反復動作
243         }
244         while (!((start1 == 0) && (start2 == 0) && (keep1 == 0) && (keep2 == 0)
245                         && (k0_flg1 == 1) && (k0_flg2 == 1)));
246         k0_flg1 = 0;                                //K0flag下ろす
247         k0_flg2 = 0;                                // - do. -
248         disable_interrupts(GLOBAL);                 //割込み禁止
249     }
250 }
```