PIC 16F648A を用いたストップウォッチの設計と試作

Design and Trial Manufacture of a Stop Watch with a PIC 16F648A Micro Computer

袴田　吉朗*

Yoshiro　HAKAMATA

Abstract: The material summarizes the design of a stop watch that uses a PIC 16F648A micro computer as a controller. In 2007, I made an incentive lecture on a manufacture of a stop watch to the Shizuoka Kita High School students. And I also undertook the task of teaching a manufacture of the stop watch. The activities were made in the chain of the Super Science High-school (SSH) program.

1．　はじめに

平成 19 年度に静岡北高校スーパーサイエンスハイスクール(SSH)におけるインセンティブ・レクチャー実施の依頼を受け，ストップウォッチに関する講義および製作実習を行った．本資料はストップウォッチの設計内容をとりまとめたものである．

2．　ストップウオッチの回路

図 1 に試作した回路を示す．3 桁で 9 分 59 秒までのカウント動作，1 秒以内の精度を目標にした．図 2 に外観を示す．制御回路の 8 ビット CMOS マイコン PIC 16F648A，アノードコモン形の 7 セグメント LED (3 個)，トランジスタおよびその他の部品からなる．電池はリチウム電池であり，電圧は 3V である．

(1)　リセット回路

10kΩの抵抗，22μF のコンデンサおよびスイッチで構成している．リセットボタンを押すと 22μF のコンデンサにおける電荷が 0 となり，/ MCLR 端子が L となってリセットがかかる．

その後 10kΩの抵抗を通して 22μF のコンデンサが電源電圧にまで徐々に充電されるので，PIC が再び動作するようになる．ハードウェアリセット時にはプログラムの処理は図 3 のフローチャートにおける先頭に移るので，LED の表示は 0 クリアされる．

(2)　start / stop SW

RA0 端子 (PORTA) にタクトスイッチを接続し，100kΩの抵抗で電源にプルアップしている．

(3)　7 セグメント LED

アノードコモン形の LED を 3 個使用し，これらをダイナミック点灯させている．

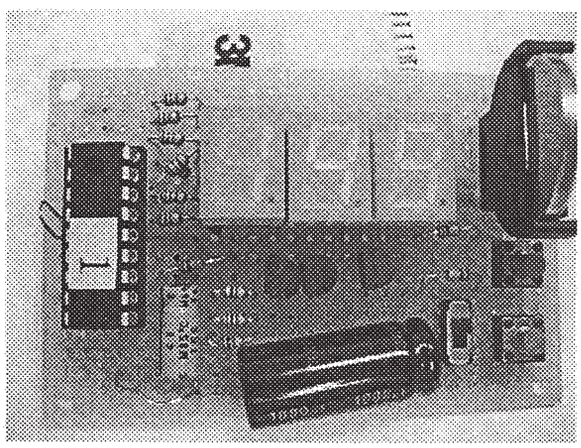

図2　完成したストップウオッチ（1 分 45 秒を表示）

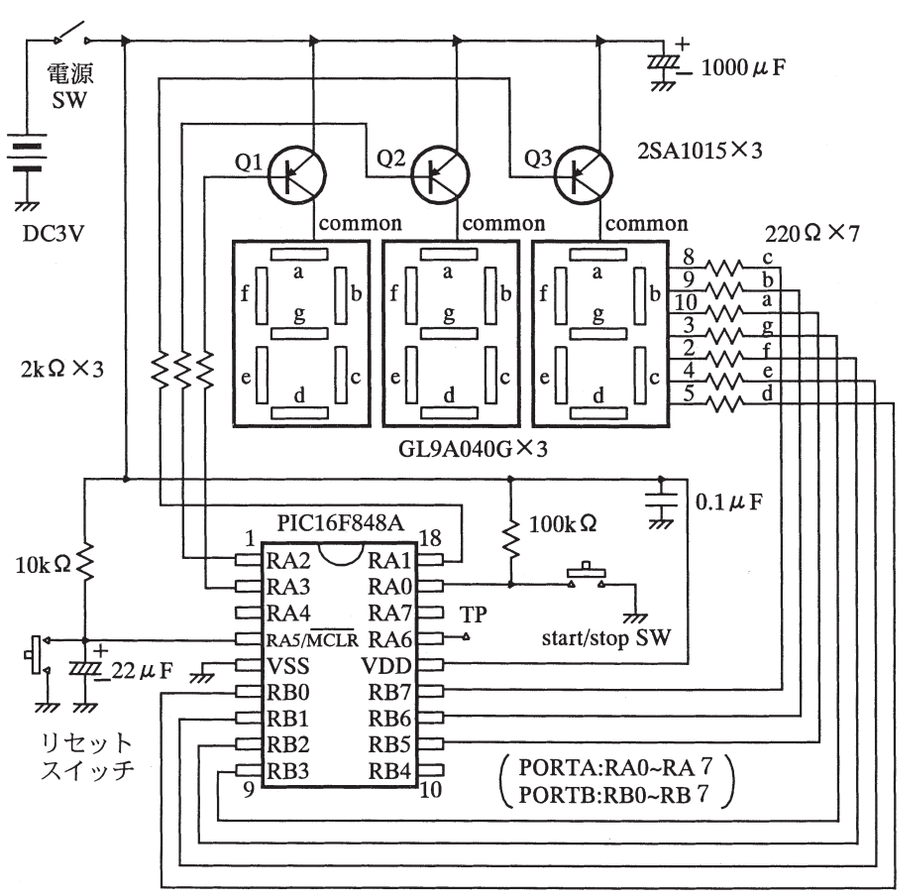

図1　ストップウオッチの回路図

2008 年 3 月 3 日受理

*理工学部　電気電子情報工学科

(4) 電流制限抵抗

7セグメントLEDに流す電流を制限するために220Ωの電流制限抵抗を用いている．LEDの端子電圧は約1.9Vであるからに流れる電流は約7mAとなり，幾分少なめである．

3. ストップウオッチのプログラム構成

(1) 回路全体のフローチャート

ストップウォッチの動作を図3の概略フローチャートに示す．プログラムはメインプログラムと割り込みサービスルーチンからなる．

① メインプログラム

- クロック周波数，入出力ピンおよびタイマー0割り込みの設定後，start / stop SWが押されるのを待つ．
- start / stop SWが押されたら，5ms毎のタイマー0割り込みを待つ無限ループを実行する（網掛けで示したループ）．
- 再びstart / stop SWが押されると，カウントを停止してその時のカウント値を表示する．
- 再度start / stop SWが押されると，続きからカウントする．
- 図3のフローチャートにはないが表示をクリアするためには，リセットスイッチを押す．

② 割り込みサービスルーチン（ISR）

- タイマー0割り込みが発生するとこのISRが5msごとにコールされる．割り込み終了時にプログラムを再開できるように必要なレジスタを待避した後，タイマー0の割り込みフラグT0IFを0クリアする．またカウンタTMR0に初期値を再設定して次の割り込みに備える．
- カウント値をLEDに表示する．
- この割り込みが200回発生したか否かで1秒の経過を計測し，このときカウント値である1バイト変数SEC_CNT（下位バイト）（あるいはSEC_CNTH（上位バイト））をカウントアップする．
- カウント値をBCDに変換後表示する．
- 保存したレジスタを復帰してメインプログラムに戻る．

(2) 各種レジスタの設定

① CMCONレジスタ

CMCONレジスタにおいてCM2:CM0=111とすることにより，コンパレータをオフとし，PORTAをI/Oピンとして使用できるようにする．設定を以下に示す．

```
MOVLW   0x07
MOVWF   CMCON
```

② PORTA，PORTBレジスタ

PICにおける入出力ピンは，「PORTレジスタ」と呼ばれる8ビットのレジスタである．このレジスタの1ビット毎が入出力ピンに対応しており，さらに「TRISレジスタ」を用いて各ピン毎に入力／出力を指定する（バンク1）．設定を以下に示す．

```
MOVLW   B'00000000'     ; PORTB
MOVWF   TRISB           ; RB0～RB7全出力
MOVLW   B'00010001'     ; PORTA
MOVWF   TRISA           ; RA0，RA4入力
```

③ PCONレジスタ

以下の命令で内蔵クロックを4MHzに設定する（バンク1）．

```
BSF     PCON，3         ; 4MHz
```

④ OPTION_REG

プリスケーラの設定を以下に示す（バンク1）．

```
MOVLW   B'11000100'     ; PS=32
MOVWF   OPTION_REG
```

⑤ TMR0

タイマーカウンターの初期値の設定を以下に示す（バンク0）．

```
T0_SETDATA      EQU     .100
MOVLW           T0_SETDATA
MOVWF           TMR0
```

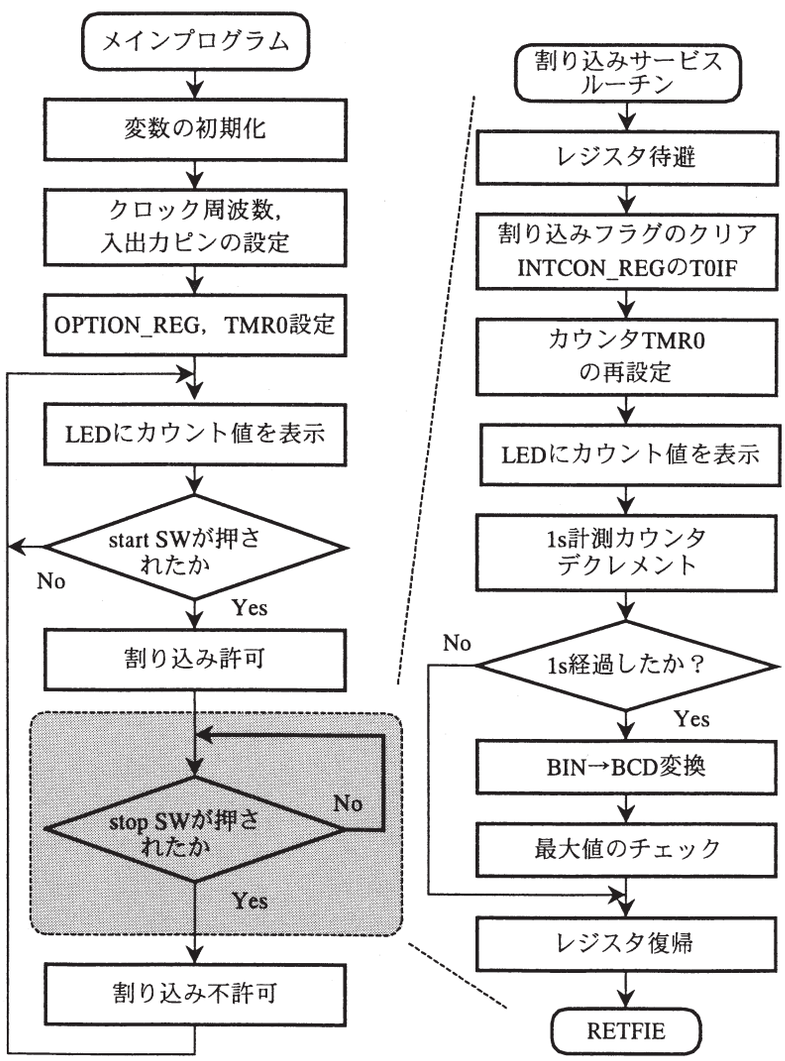

図3 ストップウオッチ・プログラムの概略フローチャート

(3) 内蔵クロックの設定

図 4 に PIC16F648A のクロック回路を示す．図 4 における CPUCLK（*Fosc*）は，PIC の 1 命令が 4 命令サイクルからできているため，更にカウンタによって1/4分周されて使用される．

部品点数を減らすために PIC に内蔵されている内部発振器を使用した．このため切り替えスイッチ 1 を用いて INTOSC に設定した．これは，プログラムのコンフィギュレーションにおいて INTOSC と記述することにより行っている．このとき RA6 / OSC2 ピンからは *Fosc* / 4=1MHz のクロックが出力される．また RA7 / OSC1 ピンは通常の IO ピンとして使用できる．

なお INTOSC_RC に設定すると，RA6 / OSC2，RA7 / OSC1 ピンとも通常の IO ピンとして使用できる．

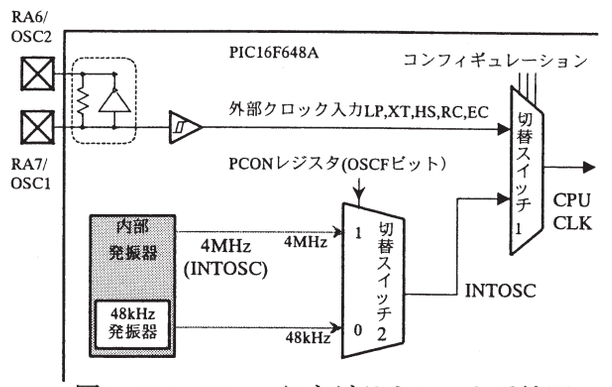

図4 PIC16F848Aにおけるクロック系統図

内部発振器の周波数は切り替えスイッチ 2 を用いて設定する．発振周波数は，プログラムにおいて図 5 に示す PCON レジスタの OSCF ビットに書き込む値によって以下の値になる．

OSCF=1 4MHz（typical）

=0 48kHz（typical）

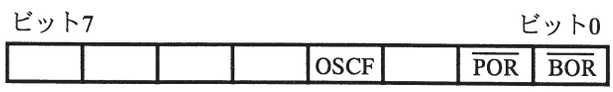

図5 PCONレジスタ（アドレス8Eh）

(4) タイマー0 割り込みによる 1s パルスの作成

図 3 に示したように 1s 毎にパルスを発生させる必要があり PIC16F648A におけるタイマー0 割り込みを使用している．

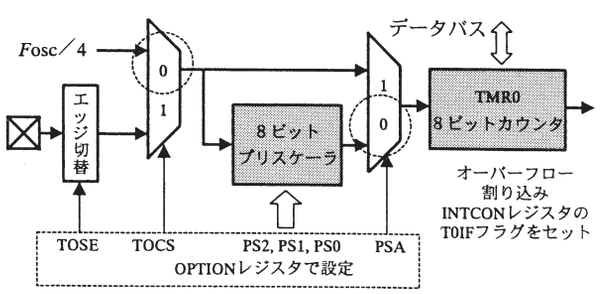

図6 タイマー0の内部構成

図 6 にタイマー0 の内部構成を示す．8 ビットのプリスケーラおよび 8 ビットの TMR0 カウンタ本体からなる合計 16 ビットのカウンタによって構成されている．したがって最大で 65536 分周が可能である．

Fosc は図 4 に示した CPUCLK であり，*Fosc* を 1/4 分周したクロックがタイマー0 に入力されている．これは 1 命令が 4 個の命令サイクルからなるためである．図 7 はプログラムにおいてプリスケーラを設定するためのタイマー0 制御用レジスタ（OPTION_REG）の内容である．

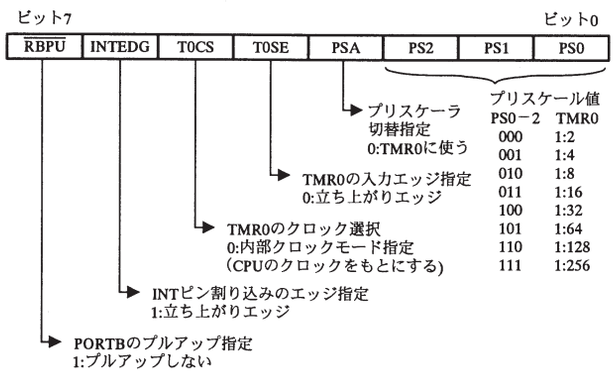

図7 OPTION_REGレジスタの内容

本プログラムでは *Fosc*=4MHz を使用したので，この場合におけるタイマー0 の動作を図 8 によって説明する．図 8 (a) はタイマー0 への入力クロックである．パルスの周期は周波数の逆数であるから $\frac{4}{F_{OSC}} = 1\mu s$ となる．プリスケーラ値は図 7 に示した 8 通りのみの設定が可能である．いまプリスケール値を *PS* =32 に設定すれば，図 8 (b) に示すようにプリスケーラにおける出力パルスの周期は

$$\frac{4}{Fosc} \times PS = 1\mu s \times 32 = 32\mu s \qquad (1)$$

となる．このパルスが TMR0 本体に入力されてさらに分周される．TMR0 本体は 8 ビットのカウンタであり，最大でも得られる周期は

$$32\mu s \times 256 = 8192\mu s = 8.192\,\mathrm{ms} \qquad (2)$$

に過ぎない．したがってタイマー0 では 5ms 周期のパルスを得るようにして，これをプログラムにおいて 200 回カウントして 1s の時刻を得るものとする．

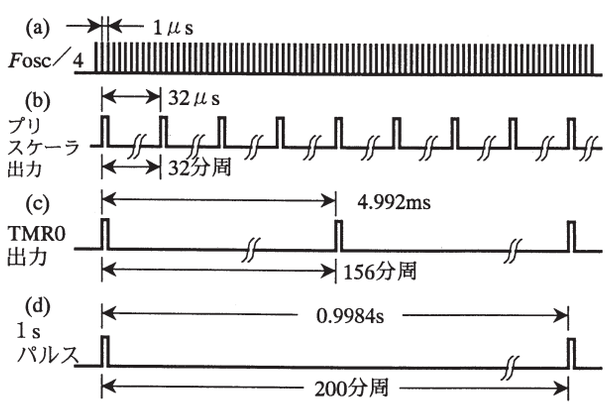

図8 タイマー0による分周動作（*Fosc*=4MHzの例）

$$\frac{5\,\mathrm{ms}}{32\mu s} = 156.25 \qquad (3)$$

であるから，式(3)の値を整数化して TMR0 本体が 156 カウント

した時にタイマー0から1個パルスが出力される（割り込みがかかる）ようにすれば良い．TMR0本体は0から255まで順次カウント数が増大していくアップカウンタである．したがって最初に256−156＝100（＝64H；Hは16進数の意味）を設定しておけば，残り156をカウントした時にタイマー0からパルスが出力され目的を達成できる．

TMR0本体の設定値としてこの値を用い，さらにプログラムにおいて200回カウントするものとすれば

$$32\mu s \times 156 \times 200 = 998400\mu s = 0.9984s \qquad (4)$$

となり正確には1秒とならない．9分59秒間では

$$0.9984s \times (60 \times 9 + 59) = 598.042s \qquad (5)$$

となる．正確には9分59秒＝599秒であるから計算上の誤差は1秒以下（遅れ）である．

(5) LEDのダイナミック点灯制御

PICによって7セグメントLEDを制御して複数桁を表示しようとする場合には，以下のような課題を解決する必要がある．

a) 7セグメントLEDで3桁表示するためには，最低でも21本のPIC入出力ピンが必要である．しかしPIC16F648Aの入出力ピンは最大16本でありこのままでは対応できない．

b) すべてのLEDを常時点灯させるようにすると消費電力が大きくなり，特に電池駆動の場合には電池が早く消耗するという問題がある．

上記の課題を一挙に解決する方法が「LEDのダイナミック点灯制御」と呼ばれる方法である．図1に示した回路は，図9に示す7セグメントLED（アノードコモン形）を用いて3桁の数字を表示する回路例である．

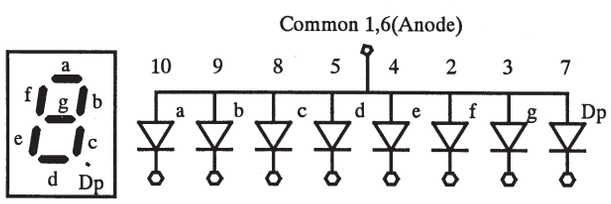

図9　7セグメントLED（アノードコモン）の接続

3個の7セグメントLEDにおける同じ記号のカソード電極は共通に接続（バス接続）し，電流制限抵抗を介してPICの出力ピン（PORTB）に接続している．一方，素子内部で共通接続されているアノード端子は，PNPトランジスタ2SA1015を介してPICにおけるPORTAから出力されるstrobe出力（桁ドライブ）に接続する．PNPトランジスタはエミッタ（E）に対してベース（B）の電圧を低くすると電流が流れるので，strobe出力がLになるとLEDが点灯する．このときstrobe出力が同時にLとならないようにプログラムで制御する必要がある．

図10にストップウオッチで使用するダイナミック点灯制御（3桁の場合）のタイミングチャートを示す．1秒の桁におけるデータを出力した後1秒の桁ドライブ信号（strobe信号）を5ms間Lにして1秒桁のLEDを点灯させ，その後10秒桁，1分桁についても同様に行う．その後は同じ動作を繰り返す．

このようにすると，ある瞬間では一つの桁だけが点灯しているが，人間の目には「残像現象」があり，一度光を見ると光が消えた後も約100ms程度はその光を連続して見ているように錯覚し各桁が連続して点灯しているように見えることになる．

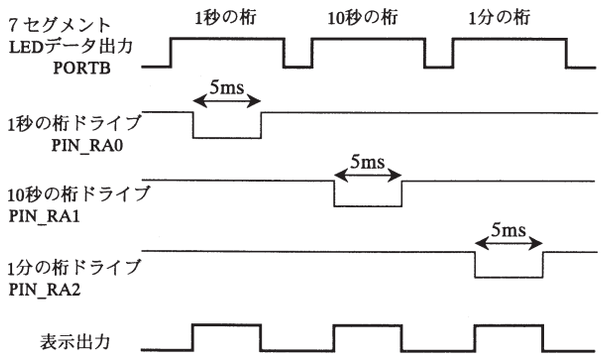

図10　ダイナミック点灯制御のタイミングチャート

図11にダイナミック点灯制御のフローチャートを示す．5ms毎に割り込みがかかるので，この度に変数COLUMNの値を1→2→4→1→・・・と巡回させ点灯を制御する．関数の冒頭で全LEDを消灯させ，その後COLUMNの値にしたがって1桁ずつ点灯させている．最初はこの部分をメインプログラムにおける網掛けを施した部分に挿入していたが，期待通りの動作にならなかったのでISRに移行させた．

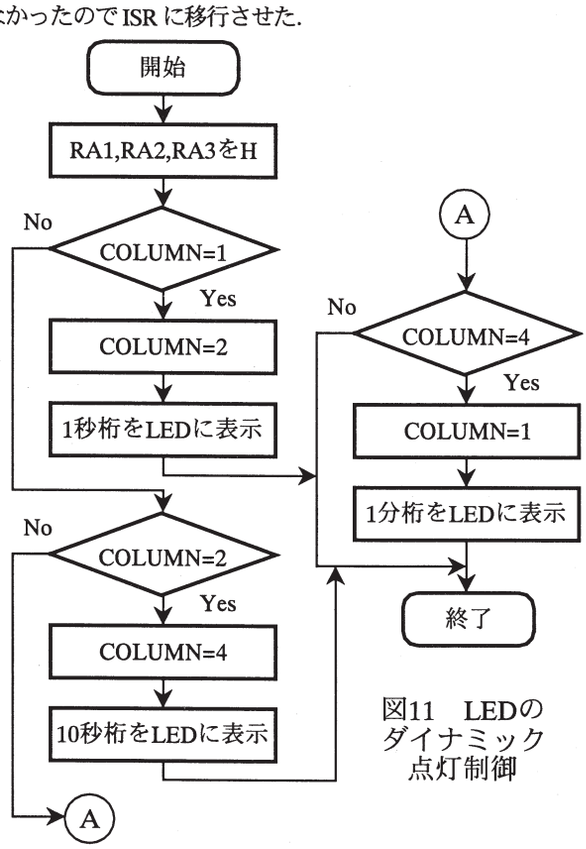

図11　LEDのダイナミック点灯制御

(6) バイナリ→BCD変換

0〜9分59秒の時刻データを秒単位で表すと0〜599秒になる．この時刻データを，ISRにおいて2個の1バイト変数SEC_CNT（下位バイト）およびSEC_CNTH（上位バイト）にバイナリデータとして格納している．したがってLEDに表示する前に，バイナリデータをBCDデータ（MINIT，SEC_HIGH，SEC_LOW）

に変換して，各 LED に対応させておく必要がある．

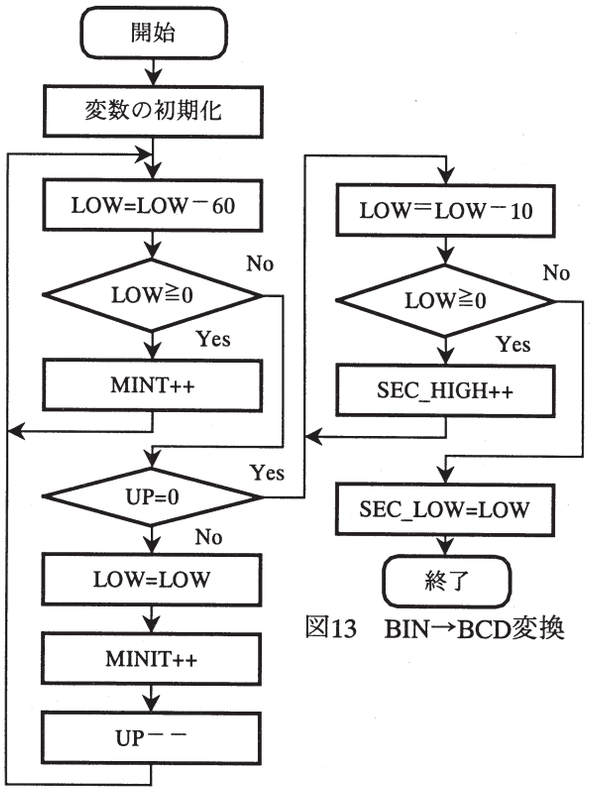

図13　BIN→BCD変換

図13がBIN→BCD変換プログラムのフローチャートであり，文献［2］を参考にして作成した．LOW は SEC_CNT を格納する一時変数，UP は SEC_CNTH を格納する一時変数である．処理は以下のようになる．

① まず，最初のブロックでは LOW=SEC_CNT から差が負になるまで 60 を引いていき，引くことができた回数を MINIT に格納する．

② 次のブロックでは，もし上位桁 UP=SEC_CNTH が 0 ならばこれ以上引けないので，10 秒桁を求めるために④に移る．

③ 0 でなければ，上位桁から借りてくることにより 60 が引けることになる．ここで LOW の値として1番目のブロックで計算した負になっている LOW の値を用いるが，この理由については(7)において説明する．

④ 3 番目のブロックでは，すでに UP=0 となっている．この条件下で LOW から 10 を引いていき，負になるまで引き続けていって10秒桁を求め，これを SEC_HIGH に格納する．

⑤ LOW の残りを SEC_LOW とする．

	ビット 7 6 5 4 3 2 1 0	
1	0 0 0 0 0 0 0 0	+256
0	1 1 1 1 1 1 1 1	+255
0	1 1 1 1 1 1 1 0	+254
	:	② :
0	1 0 0 0 0 0 0 1	+129
0	1 0 0 0 0 0 0 0	+128
0	0 1 1 1 1 1 1 1	+127
0	0 1 1 1 1 1 1 0	+126
	:	:
0	0 0 0 0 0 1 0 0	+4
0	0 0 0 0 0 0 1 1	+3
0	0 0 0 0 0 0 1 0	+2
0	0 0 0 0 0 0 0 1	+1
0	0 0 0 0 0 0 0 0	+0
1	1 1 1 1 1 1 1 1	-1
1	1 1 1 1 1 1 1 0	-2
	:	① :
1	1 0 0 0 0 0 0 1	-127
1	1 0 0 0 0 0 0 0	-128

図14　2の補数による符号付き2進整数

(7)　上位桁からのボローの計算

(4)の③において，UP が 0 でない場合に LOW の値として 1 番目のブロックで計算した結果として負になっている値を用いるが，これは以下の理由による．

図 14 は，2 の補数による符号付き 2 進整数を表したものである．ビット 7 が 1 となるブロック①は 8 ビットにおける－128～－1 の負数を表すが，これらのブロックにおけるビットの並びは+128～+255 を表すブロック②における並びと全く同じである．ブロック①②は UP の値によって識別できる．このため UP=1 が確認できれば，(6)③のような計算を行っても差し支えないことが分かる．

(8)　時刻データにおける最大値のチェックと初期化

時刻データが最大値 599 になったとき，各種の変数を初期化する必要がある．これを図 15 に示すフローチャートに従って行っている．

	c b a	g f e d
0	0 0 0 1	1 0 0 0
1	0 0 1 1	1 1 1 1
2	1 0 0 1	0 1 0 0
3	0 0 0 1	0 1 1 0
4	0 0 1 1	0 0 1 1
5	0 1 0 1	0 0 1 0
6	0 1 0 1	0 0 0 0
7	0 0 0 1	1 1 1 1
8	0 0 0 1	0 0 0 0
9	0 0 0 1	0 0 1 1

図16　LEDのデータ

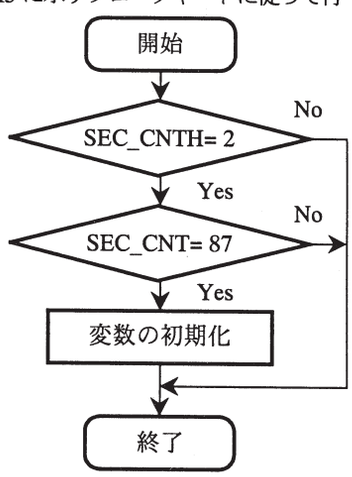

図15　時刻データの最大値の検出

(9)　BCD→7セグメント変換

アノードコモン形の 7 セグメント LED（GL9A040G）を使用している．ピン配置を考慮してポート B の出力データを図 16 のように設定した．カソードコモン形におけるデータのビットを反転して呼び出すことを基本としているが，配線の都合上から上位 4 ビットと下位 4 ビットを交換している．

(10)　ボタン押下の検出

start／stop SW が押されたか否かをプログラムで検出するときに，接点がついたり離れたりする「チャタリング」が生ずる．

本プログラムでは，このためいったん入力の変化を検出してから，チャタリングが終了するのにかかる時間であると予想される 5ms の間タイマーを入れて待つようにした（図 17）．その後に再度データを入力し，最初に検出したデータ (0) と同じデータ (0) であることが確認できた場合に次の処理に進むようにした．もし再度確認した時にデータが 1 であれば，上記の条件が満足されるまで待つようにしている．

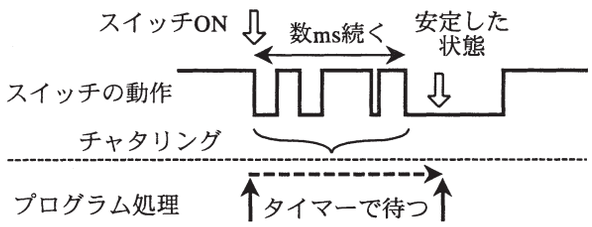

図17　スイッチ（キー）入力の方法

なお実際のタクトスイッチを押した場合の動作を測定したところ，チャタリングは予想よりは少なかったものの，L となる時間が最低でも約 100ms と長いことが分かった．このためスイッチを押して離してもすぐ応答せず，反応が鈍いという症状が出て使い勝手が良くなかった．そこで再読込後 PORTA の RA0 を強制的に H となるように制御している．

4．試作結果および高精度化の検討

(1)　全部で 10 台試作した．市販のストップウォッチと比較しながら 9 分 59 秒間のカウント試験を行ったが，最悪 10 秒程度の誤差の出るものがあった．そこで以下の検討を行った．

(2)　プログラムにおける流れの見直し

大きな誤差の出る原因は，図 3 のフローチャートにおいて「1s 計測カウンタのデクレメント」～「最大値チェック」の部分を ISR に入れており，時刻のカウント値によって ISR の処理時間の異なることが一要因と考えられる．特に 1 秒経過後の処理時間が大きい．このため，「1s 計測カウンタのデクレメント」以外の部分をメインプログラムの網掛けを施した部分に再度移行してみた．

なおタイミングを制御するために ISR に変数 start_flag を導入し，ISR がコールされたときにこの変数を H とし，メインへ移行した部分を抜ける時に L となるように変更を加えた．この手法は，LD-CELP コーデックで取った手法[3]と同じである．

この結果，処理の流れによって ISR の処理時間が大きく変化することはなくなった．また LED のダイナミック点灯制御も問題なく実行できた．

(3)　内蔵クロック周波数の測定

(2)の変更を織り込んだプログラムを 2 号機に書き込み，カウント試験を実施してみた．しかしながらシミュレーションでは ISR のコールされる周期が 5ms になっており，しかも処理によってほぼ一定であることを確認できたが，9 分 59 秒の試験では 6 秒程度進んでいた．動作時の電源電圧は 2.48V であった．旧プログラムを使用した 1 号機の電源電圧は 1.9V にまで低下していたが，同じ試験を行ったところ 1 秒以内の誤差であった．

そこで 2 号機のクロック周波数をテストポイント(15 番ピン)において測定してみたところ，電源電圧 2.7V～2.3V において 1.0093MHz～1.0089MHz であり，1%ほど進んでいた．消費電流は 70mA～20mA であった．

(4)　外付け 32.768kHz クロックへの変更

試作した結果では電池の消耗がかなり激しく，また内蔵クロック周波数の精度が良くないことが分かった．このため外付けの 32.768kHz 水晶振動子を用い，周波数の低下による低消費電力化と，TMR0 カウンタの再設定を不要とさせる高精度化を図ることにした．プログラムの変更点は以下の 4 点である．

・コンフィギュレーションで LP_OSC に設定
・PCON レジスタをコメントアウト
・OPTION_REG において PSA=1 に設定
・TMR0 カウンタの初期値を T0_SETDATA=0 に設定

以上の設定により，タイマー0 割り込みの周期は 256 クロック（31.25ms）となり，カウンタの再設定は不要となる．

このプログラムを 4 号機に実装して 9 分 59 秒のカウント試験を行ったところ，約 1 秒の進みであり精度が向上した．このときの電源電圧は 2.48V，電流は 12mA である．精度は向上したものの，ダイナミック点灯の周期が 31ms と長くなったため，チラツキが非常に気になるようになってしまった．

そこで，ダイナミック点灯制御の流れを図 18 のように変更した．上部の数字はクロック数である．ポイントを以下に示す．

① 図 11 において，冒頭における全 LED 消灯から各桁 LED の点灯までの時間を NOP により調整し，同一となるようにした．
② strobe 出力が L となる期間を 5ms に調整するために，3.54ms（29 クロック）のタイマーを後続させた．
③ 1 分桁を表示後，全 LED を消灯させ，各桁の点灯時間を等しくなるようにして，各桁が同じ明るさで点灯するようにした．
④ その後 start_flag を 0 とし，割り込み待ちに入り，ISR とタイミングをとっている．この時間は 1 秒経過後のバイナリ→BCD 変換に要する時間を稼ぐために必須である．

まだ 4MHz クロックの場合に比べると幾分気にはなるものの，実用上問題のないチラツキに抑えることができた．

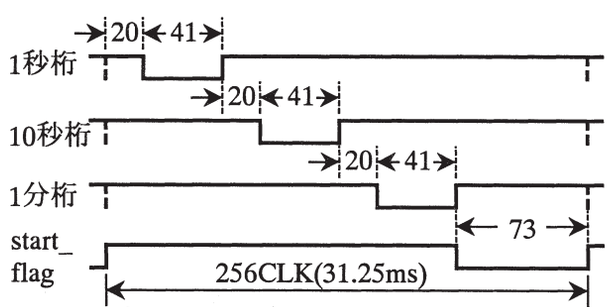

図18　ダイナミック点灯制御のタイムチャート

5．むすび

PIC16F648A を使用したストップウォッチの設計と試作結果について述べた．

［参考文献］

1) PIC16F627A/628A/648A Data Sheet Flash-Based, 8-Bit CMOS Micro controllers with nano Watt Technology
2) 後閑哲也，“改訂版 電子工作のための PIC16F 活用ガイドブック”，技術評論社，平成 19 年
3) 袴田吉朗，溝口真規，緒方渉，菅野純一，“LD-CELP アルゴリズムを適用した音声 CODER の開発”，静岡理工科大学紀要，第 11 巻，pp.35－59（2003）