

簡易時計(PIC単体)

概要

オークションで、8.000000MHzという、高精度のクリスタルオシレータが、安価で手に入りましたので、早速、簡易な時計を製作してみました。

通常ですとRTC(リアルタイムクロック)モジュールを利用するところなのですが、折角、高精度のクリスタルオシレータが手に入りましたので、今回は、PIC単体で実現してみました。その分、ソフトに重

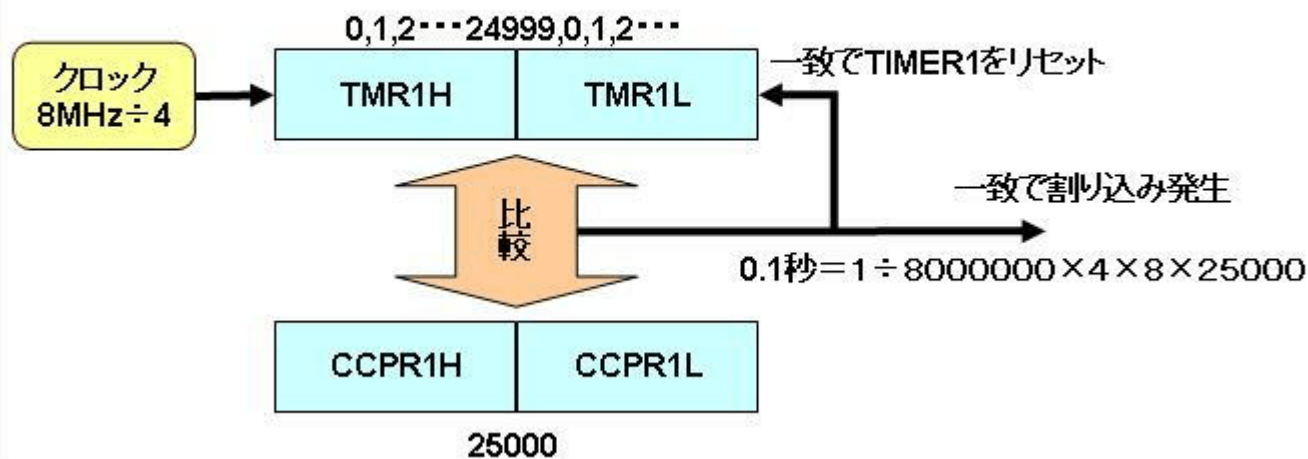
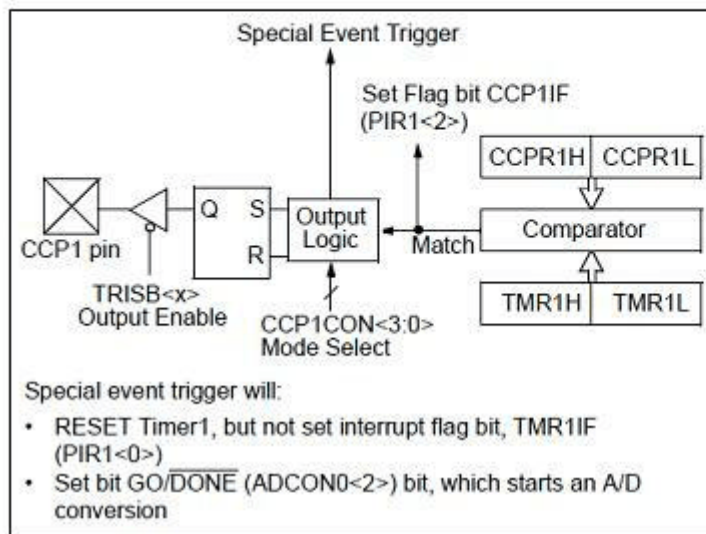


きが置かれています。

動作原理

0.1秒の正確な、時間(割り込みタイミング)を得るためにCCPモジュールをコンペアモードで使

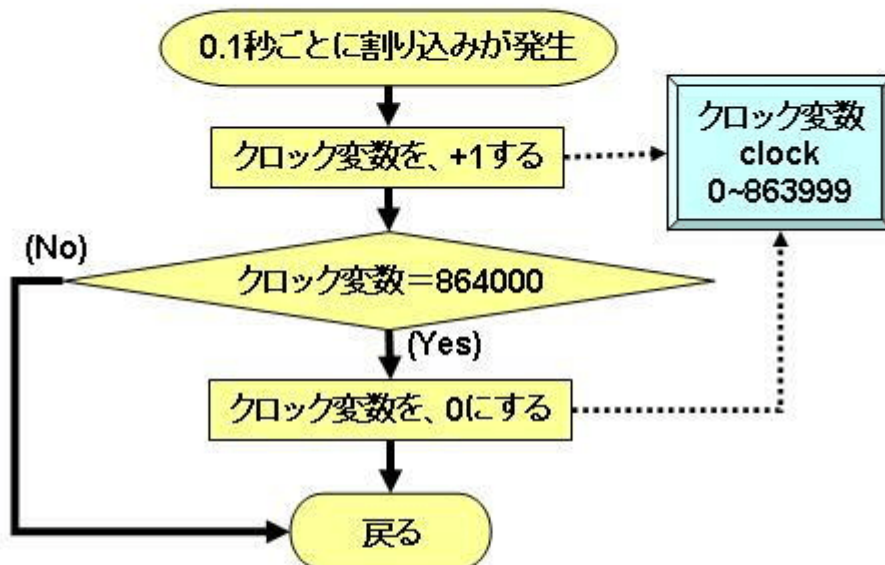
CCPモジュールをコンペアモードで使用して、0.1秒の割り込みを発生させる。



割り込み処理では、時刻データ(クロック変数)を生成します。とても単純です。

0.1秒の割り込みから、時刻データを生成する。

24時間は、0.1秒で換算すると、864000回の割り込みがあれば良いことになります。
 $864000 = 24\text{時間} \times 60\text{分} \times 60\text{秒} \div 0.1$



例えば、12時34分56秒のときの、クロック変数の値は、452960になります。
 $452960 = (12\text{時} \times 36000) + (34\text{分} \times 600) + (56\text{秒} \times 10)$

クロック変数(clock)から、時、分、秒を得る。

時(hh) = $\text{clock} \div 36000$;
 分(mm) = $(\text{clock} - (36000 \times \text{hh})) \div 600$;
 秒(ss) = $(\text{clock} - (36000 \times \text{hh}) - (600 \times \text{mm})) \div 10$;

時刻表示では、クロック変数から、時、分、秒を求め、表示します。その時に、クロック変数へのアクセスを、メイン処理と割り込み処理の間で、排他制御を考慮しなければなりません。つまり、

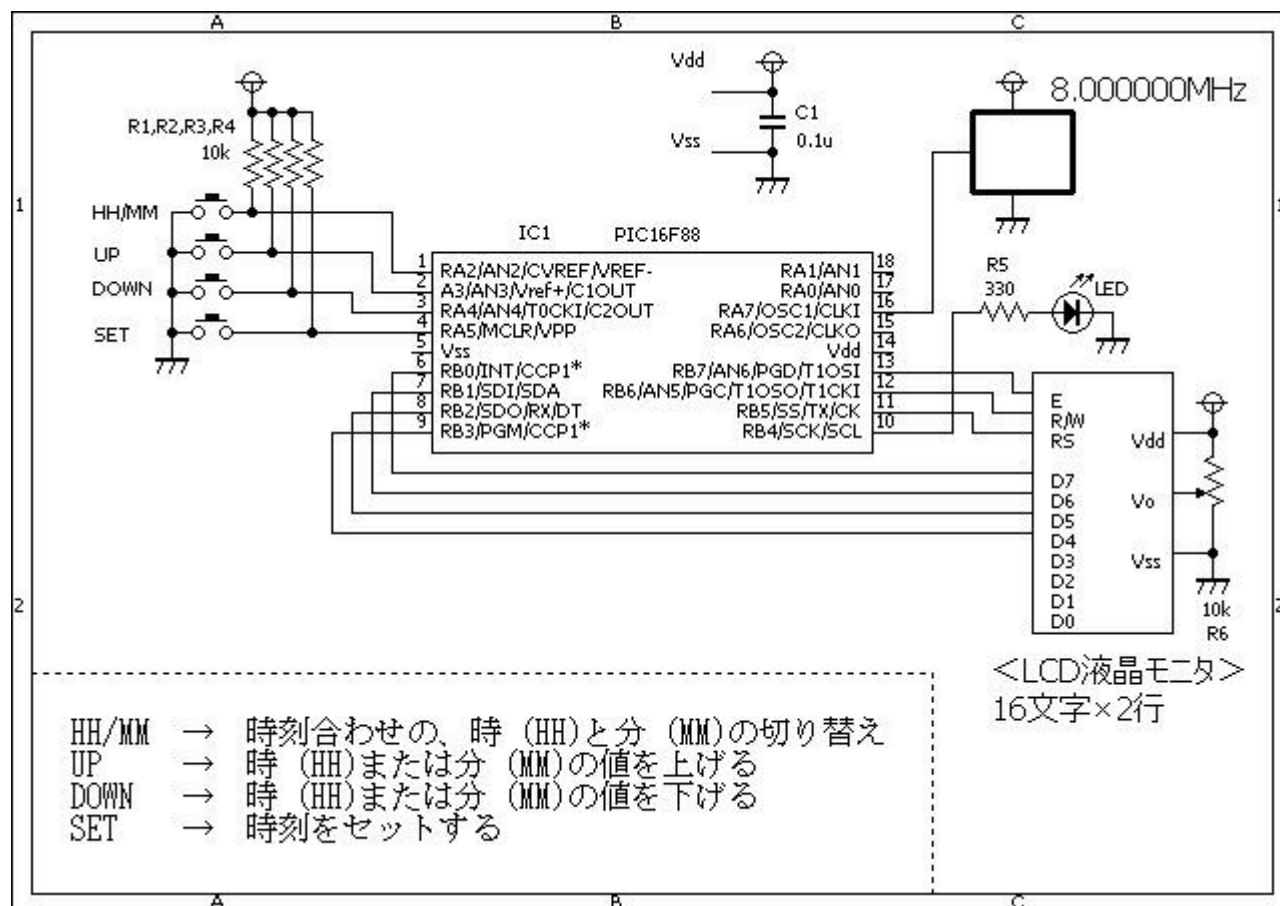
- メイン処理で、クロック変数を参照しようとしているときに、
- 割り込み処理が、クロック変数を+1しようすると、

タイミングによっては、不具合が生じます。従って、

- メイン処理では、参照前に、クロック変数をロックし、参照後に、ロックを解除します。
- 割り込み処理では、ロックされているときには、クロック変数にアクセスしないようにします。

こうすることによって、不具合を防ぐことができます。

回路図



ソースコード

[clock_v2.c](#)

```
//*****
*

#define      ON          0
#define      OFF         1

#define      SW_MODE      &PORTA, 2
#define      SW_UP        &PORTA, 3
#define      SW_DOWN      &PORTA, 4
#define      SW_SET       &PORTA, 5

#define      LED          PORTB.F4

#define      TRUE         255
#define      FALSE        0

#define      LOCK         1
#define      UNLOCK        0

//*****
*
```

```

static long    clock;
static short   lock;

void    interrupt()
{
    if (PIR1.CCP1IF == 1) {
        PIR1.CCP1IF = 0;
        //
        LED = ~LED;
        //
        if (lock == UNLOCK) {
            clock++;
            if (clock == 864000)
                clock = 0;
        }
    }
}

//*****
*

void    main()
{
    static    char    buf[16];
    static    short    hh, mn, ss, hh_tmp, mn_tmp, ss_tmp, mode;
    static    long    tmp;
    //
    //    OSCCON = 0b01110000;           // クロックは8MHz
    CMCON    = 0b00000111;           // コンパレータは使用しない。
    ANSEL    = 0b00000000;           // A/D変換は使用しない。
    TRISA    = 0b11111111;
    TRISB    = 0b00000000;
    OPTION_REG.F7 = 0;               // PORTBをプルアップする。
    // TIMER1の設定
    PIE1.TMR1IE = 0;
    PIR1.TMR1IF = 0;
    T1CON.T1CKPS0 = 1;
    T1CON.T1CKPS1 = 1;
    T1CON.TMR1ON = 0;
    TMR1L = 0;
    TMR1H = 0;
    // CCPの設定
    PIE1.CCP1IE = 1;
    PIR1.CCP1IF = 0;
    CCP1CON.CCP1M3 = 1;
    CCP1CON.CCP1M2 = 0;
    CCP1CON.CCP1M1 = 1;
    CCP1CON.CCP1M0 = 1;
    CCPR1L = 0xA8;           // 0.1sec...10hz...クロックが8Mhzの時
    CCPR1H = 0x61;           // 0.1sec...(1÷8000000)*4*8*25000

```

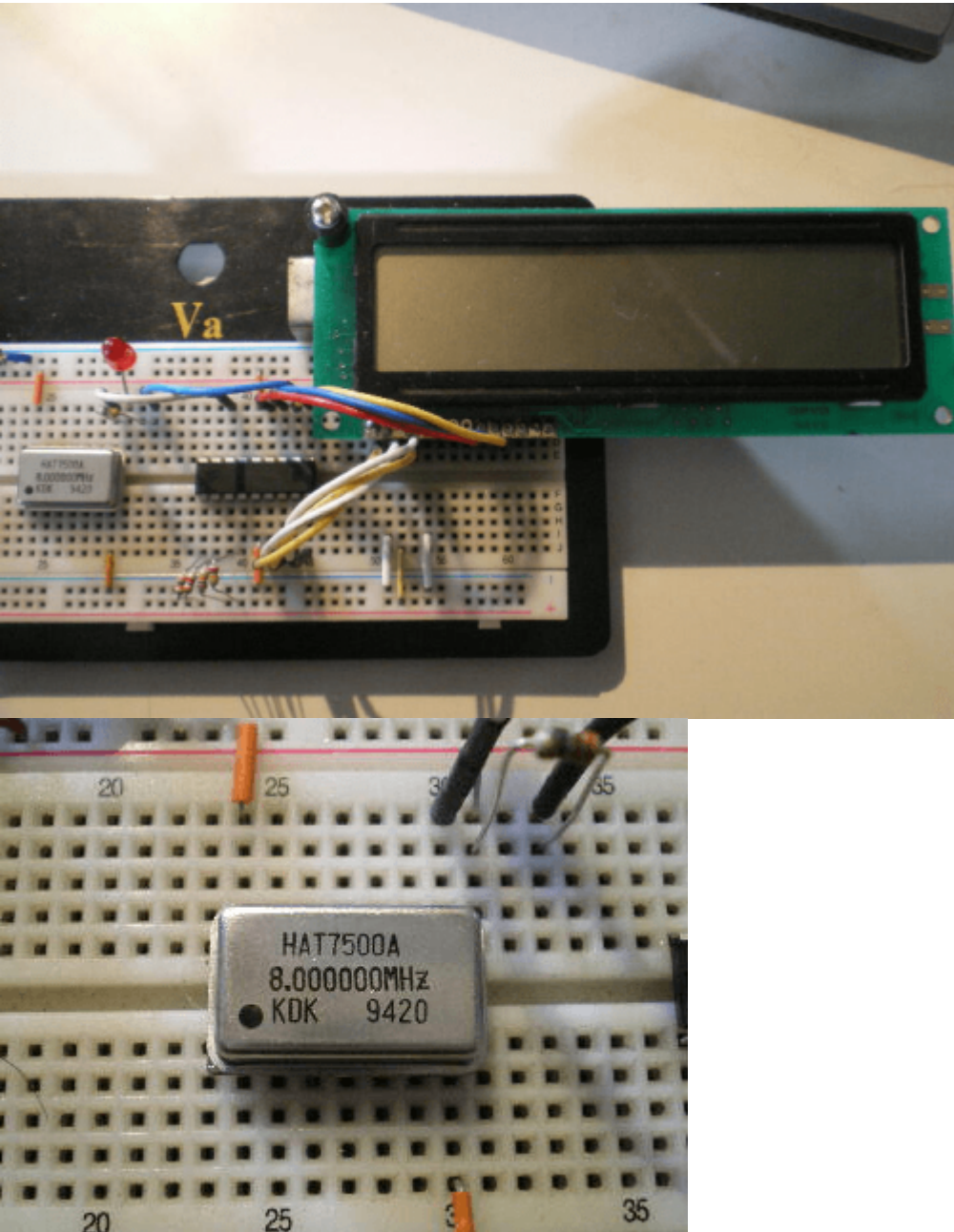
```
//
Lcd_Custom_Config(&PORTB, 0, 1, 2, 3, &PORTB, 5, 6, 7);
Lcd_Custom_Cmd(LCD_CURSOR_OFF);
Lcd_Custom_Cmd(LCD_CLEAR);
//
clock = 0;
lock = 0;
mode = 0;
hh_tmp = 0;
mn_tmp = 0;
ss_tmp = 0;
// 割り込みを許可する。
INTCON.PEIE = 1;
INTCON.GIE = 1;
//
T1CON.TMR1ON = 1;
//
while (1) {
    hh = clock / 36000;
    mn = (clock - (36000 * (long)hh)) / 600;
    ss = (clock - (36000 * (long)hh) - (600 * (long)mn)) / 10;
    //
    LongToStr(clock, buf);
    Lcd_Custom_Out(2, 10, &buf[4]);

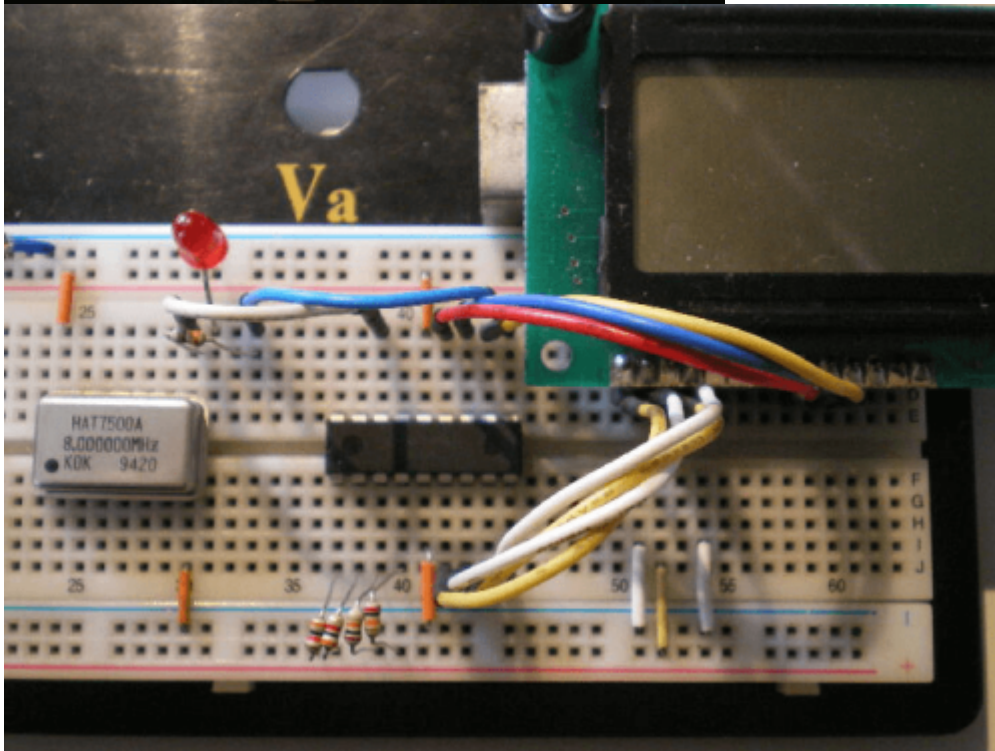
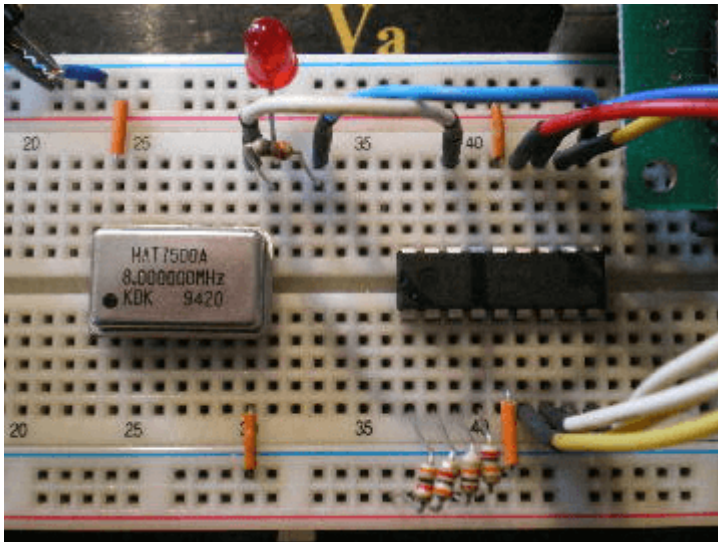
    ByteToStr(hh, buf);
    buf[1] = buf[1] == ' ' ? '0' : buf[1];
    Lcd_Custom_Out(1, 1, &buf[1]);
    Lcd_Custom_Out(1, 3, ":");
    //
    ByteToStr(mn, buf);
    buf[1] = buf[1] == ' ' ? '0' : buf[1];
    Lcd_Custom_Out(1, 4, &buf[1]);
    Lcd_Custom_Out(1, 6, ":");
    //
    ByteToStr(ss, buf);
    buf[1] = buf[1] == ' ' ? '0' : buf[1];
    Lcd_Custom_Out(1, 7, &buf[1]);
    //
    if (Button(SW_MODE, 10, ON) == TRUE) {
        while (Button(SW_MODE, 10, OFF) == FALSE)
            ;
        mode = (mode == 0) ? 1 : 0;
    }
    //
    if (Button(SW_UP, 10, ON) == TRUE) {
        while (Button(SW_UP, 10, OFF) == FALSE)
            ;
        if (mode == 0) {
            hh_tmp++;
            if (hh_tmp == 24)
```

```
        hh_tmp = 0;
    } else {
        mn_tmp++;
        if (mn_tmp == 60)
            mn_tmp = 0;
    }
}
//
if (Button(SW_DOWN, 10, ON) == TRUE) {
    while (Button(SW_DOWN, 10, OFF) == FALSE)
        ;
    if (mode == 0) {
        hh_tmp--;
        if (hh_tmp < 0)
            hh_tmp = 23;
    } else {
        mn_tmp--;
        if (mn_tmp < 0)
            mn_tmp = 59;
    }
}
//
if (Button(SW_SET, 10, ON) == TRUE) {
    while (Button(SW_SET, 10, OFF) == FALSE)
        ;
    lock = LOCK;
    cclock = ((long)hh_tmp * 36000) + ((long)mn_tmp * 600);
    lock = UNLOCK;
}
//
ByteToStr(hh_tmp, buf);
buf[1] = buf[1] == ' ' ? '0' : buf[1];
Lcd_Custom_Out(2, 1, &buf[1]);
if (mode == 0)
    Lcd_Custom_Out(2, 3, "?");
else
    Lcd_Custom_Out(2, 3, ":");
//
ByteToStr(mn_tmp, buf);
buf[1] = buf[1] == ' ' ? '0' : buf[1];
Lcd_Custom_Out(2, 4, &buf[1]);
if (mode == 1)
    Lcd_Custom_Out(2, 6, "?");
else
    Lcd_Custom_Out(2, 6, ":");
//
ByteToStr(ss_tmp, buf);
buf[1] = buf[1] == ' ' ? '0' : buf[1];
Lcd_Custom_Out(2, 7, &buf[1]);
}
}
```

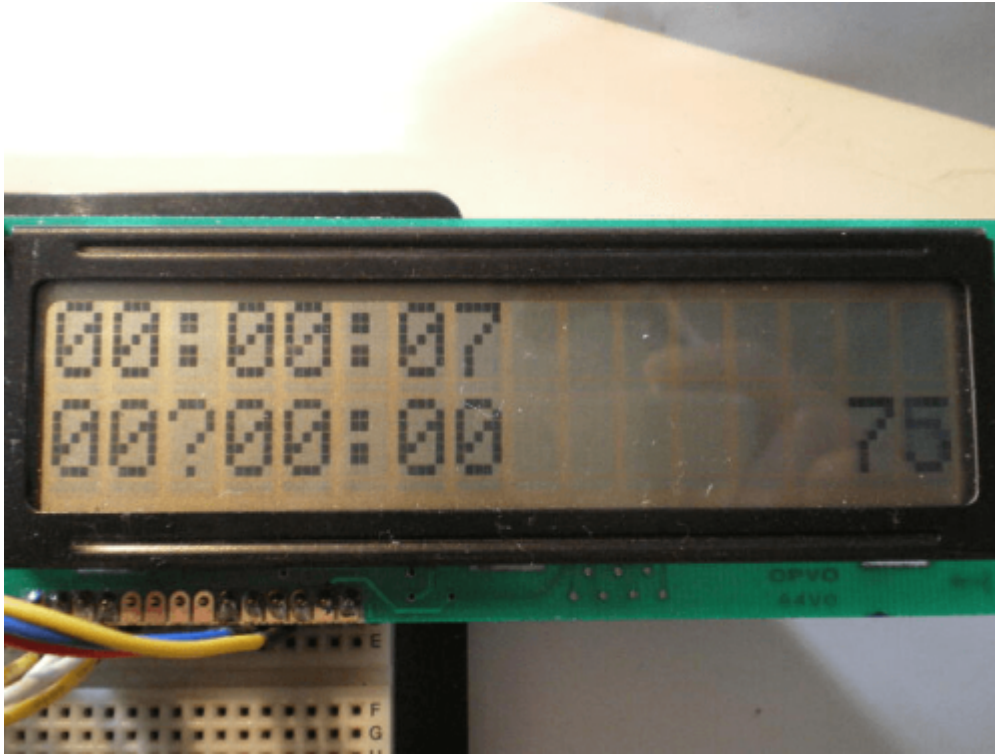
// *****
*

動作確認

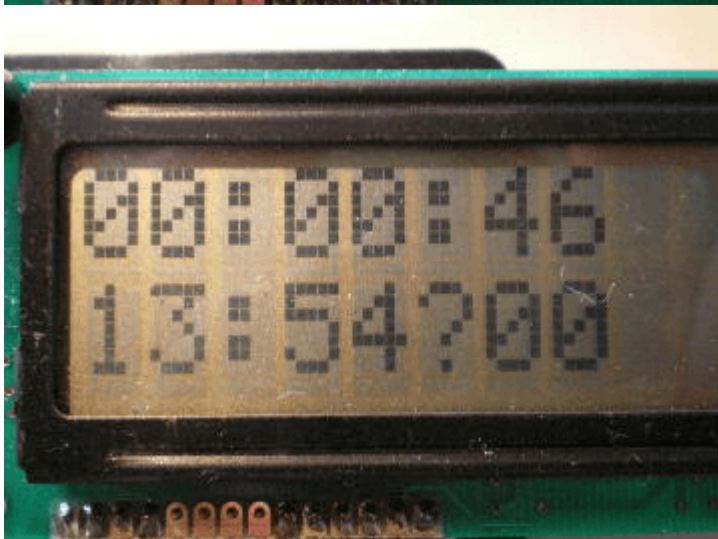
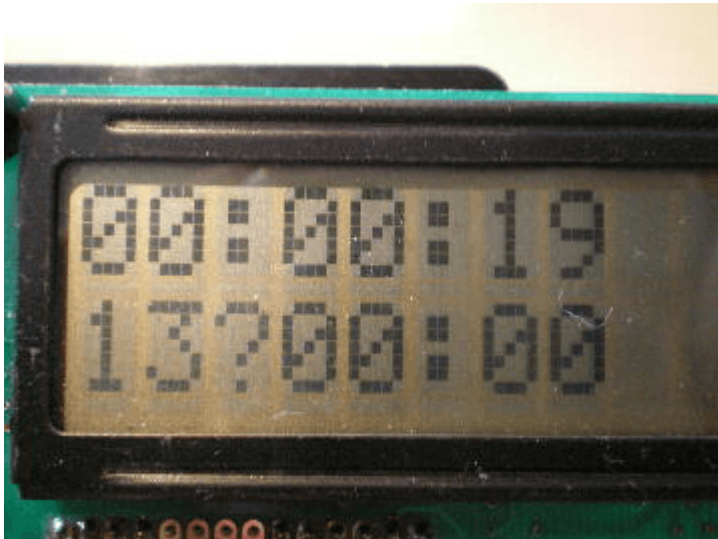




電源投入直後です。左上:現在時刻です。左下:時刻合わせ用の時刻です。秒は00固定です。右下:クロック変数(clock)の値です。0.1秒単位で+1されます。



13時54分00秒に時刻合わせをしているところです。左側:HH/MMスイッチで、“時”の“:”が“?”になるようにして□UP/DOWNスイッチで“13”に設定します。右側:HH/MMスイッチで、“分”の“:”が“?”になるようにして□UP/DOWNスイッチで“54”に設定します。



SETスイッチを押すと、現在時刻が、セットされます。



そのときの、クロック変数はかなり大きな値になりますね。



如何ですか? 高精度のクリスタルオシレータを使用したのでRTCを使用するよりも、精度は高くなっていると思います。

またPIC単体で実現したのでPICのモジュールの仕組みや、ソフト(割り込み処理、排他制御、時刻の相互換算など)の理解にも役立つのではないのでしょうか?

お手持ちのクリスタルオシレータ(1MHz~20MHz位の範囲で)があれば、是非、活用してみてください。発振周波数が8MHzでなくても、少しのソフト修正で対応できますので。。。

著作権表示 **copyright notice**

このページは稲崎様の閉鎖したHPのコピーで、著作権は稲崎様にあります。[詳細](#) This page is a copy of Mr. Inasaki's closed website, and the copyright is held by him.[Details](#)

From:

<http://deepsky.jp/wiki/> - うごくといいな

Permanent link:

<http://deepsky.jp/wiki/doku.php?id=elechobby:picdic:pic16f88:108>

Last update: **2020/05/01 18:45**

