

ミニ周波数カウンタ(kHz表示)

概要

通常、1個のPICで、周波数カウンタを製作する場合、測定した結果は、LCDに表示したり、パソコンへRS232Cでデータ転送したりします。7セグメントのLEDを使って、ダイナミック点灯させる事例は、あまり見かけません。

その理由は、測定周期と表示周期が

- 周波数カウンタの測定周期 1秒、0.1秒
- ダイナミック点灯の表示周期 5msec~1msec

というふうに、異なり共存が難しいためです。

今回は、この問題点を解決し、次のような仕様の周波数カウンタを製作します。

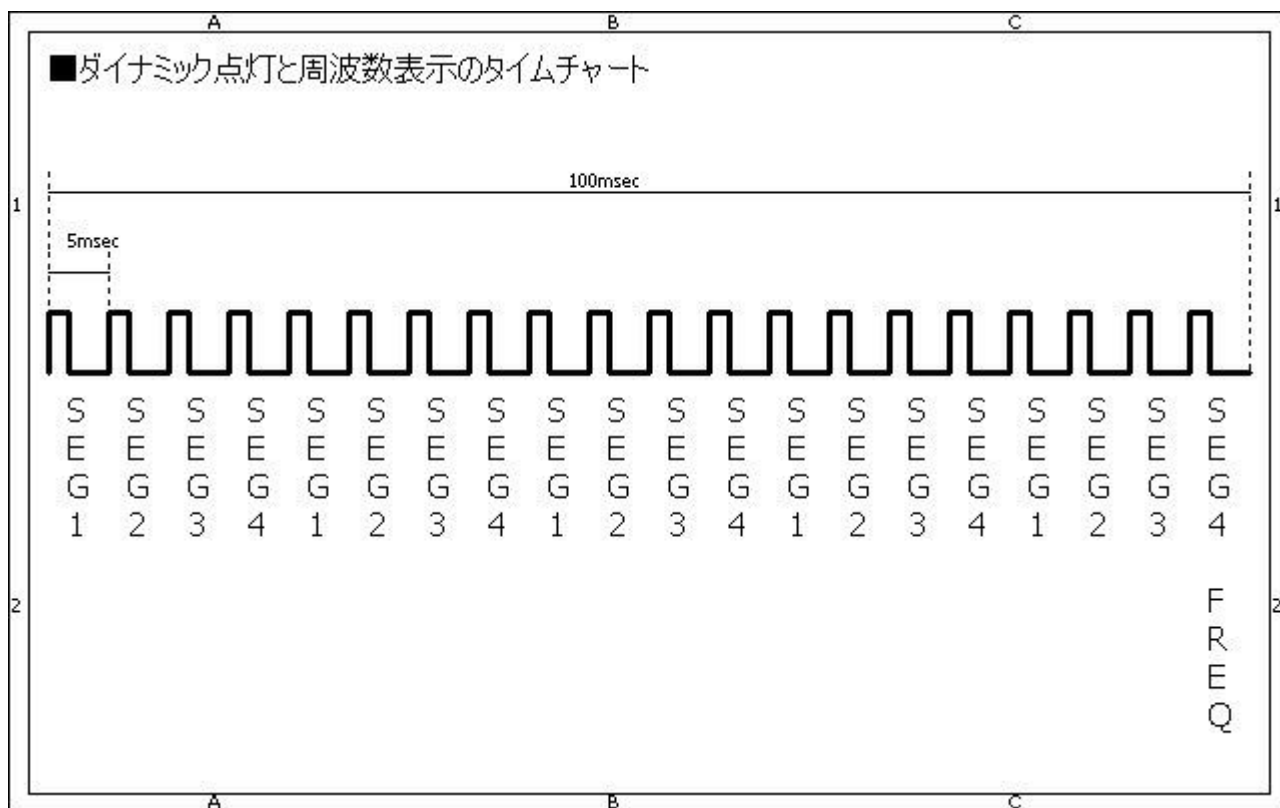
<仕様>

- ゲートタイムは0.1秒とする。表示はkHz単位とする。
- 測定範囲は、1kHz~15MHzまでとする。
- 1kHz~9999kHzまでは、4桁のkHz表示とする。
- 10MHz以上は、2桁のMHz+2桁のkHz表示とする。(例。12.34MHz)
- スイッチ切り替えで 0~455kHzの設定を可能とする。(ラジオ受信周波数表示用)

動作原理

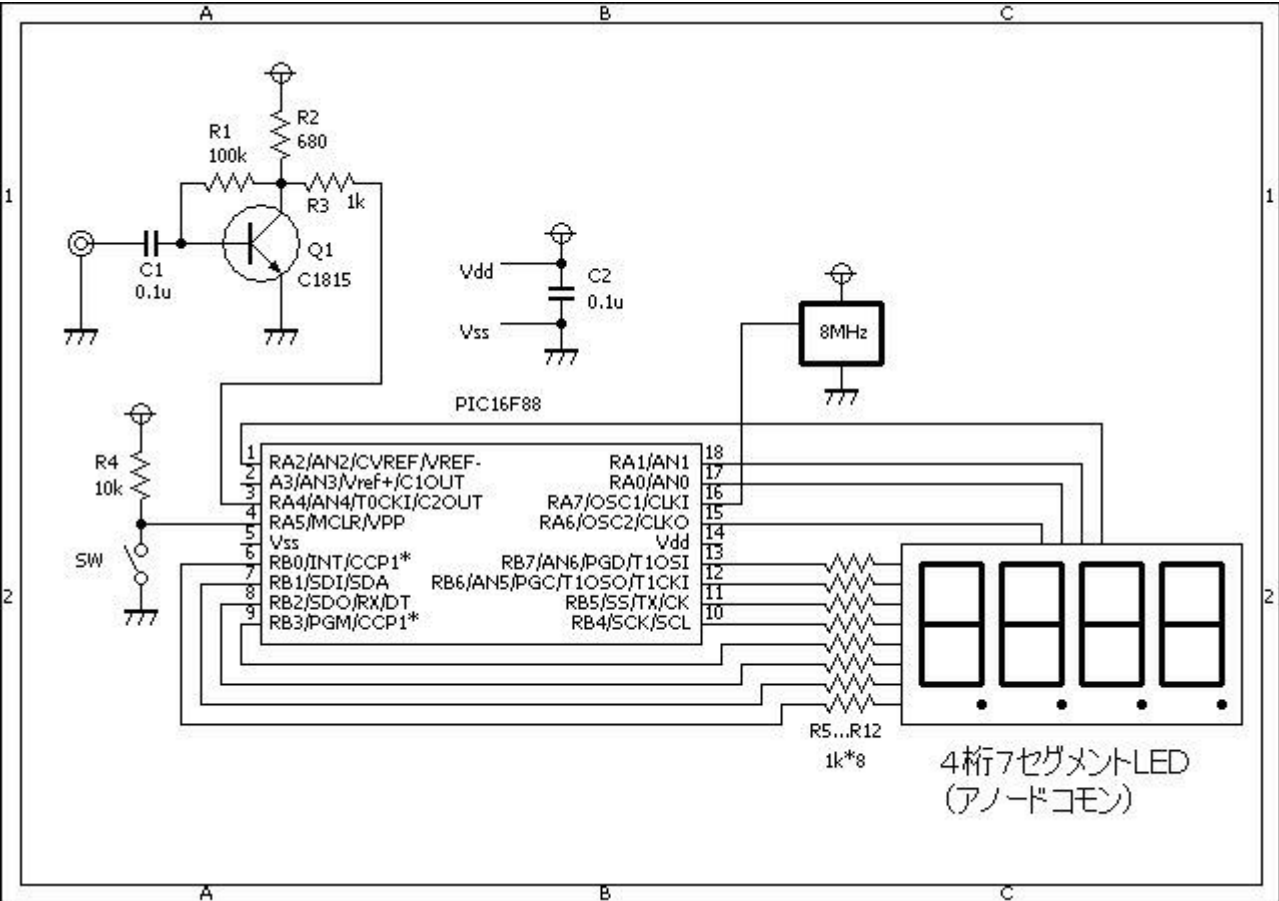
周波数カウンタの測定周期の1秒や0.1秒は、5msecの200倍または20倍に相当するので、測定周期と表示周期を、共通のタイマー割り込みで処理させることが出来そうです。つまり、

- 5msec周期のタイマー割り込みを発生させる。
- 割り込み処理では、7セグメント(4桁)のLEDをダイナミック点灯させる。
- 割り込み処理では、0.1秒(5msec×20回)間の、周波数をカウントする。



回路図

通常、ダイナミック点灯をさせる場合、7セグメントLEDへの供給電圧の制御は、トランジスタ等を使用します。そして輝度を得るために、セグメントに接続する抵抗も、小さめの値(100Ω前後)にします。今回は、PICのポート直としました。そのため、ポートの最大電流を考慮し、1KΩと大きめの値にします。



ポート	7SEG
PORTB(0)	G
PORTB(1)	A
PORTB(2)	F
PORTB(3)	B
PORTB(4)	C
PORTB(5)	dp
PORTB(6)	dp
PORTB(7)	E

ソースコード

7seg_4disp.c

```
// ****
*
/*
< ミニ周波数カウンタ(kHz表示) >
*/
// ****
*

#define DATA0 0b00100001
#define DATA1 0b11100111
#define DATA2 0b00110100
```

```
#define DATA3 0b10100100
#define DATA4 0b11100010
#define DATA5 0b10101000
#define DATA6 0b00101000
#define DATA7 0b11100001
#define DATA8 0b00100000
#define DATA9 0b10100000
#define DATA_SPACE 0b11111111

#define SPACE 10

#define NON_SEG 0b00000000
#define SEG1 PORTA.F6
#define SEG2 PORTA.F0
#define SEG3 PORTA.F1
#define SEG4 PORTA.F2

#define ON 1
#define OFF 0

//*****
*

short seg_flg, data1, data2, data3, data4, dot;
short tbl[11] = {DATA0, DATA1, DATA2, DATA3, DATA4, DATA5, DATA6,
DATA7, DATA8, DATA9, DATA_SPACE};

short fc_flg;

void interrupt()
{
    if (PIR1.CCP1IF == 1) {
        PIR1.CCP1IF = 0;
        //周波数カウンタ処理
        switch (fc_flg) {
            case -1:
                break;
            case 0:
                TRISA.F4 = 1; //ゲートを開ける。
                fc_flg++;
                break;
            case 20:
                TRISA.F4 = 0; //ゲートを閉める。
                PORTA.F4 = 0;
                fc_flg = -1;
                break;
            default:
                fc_flg++;
                break;
        }
    }
}
```

```
//□□□□□桁)点灯処理
switch (seg_flg) {
case 0:
    seg_flg = 1;
    SEG4 = OFF;
    PORTB = (dot == 1) ? tbl[data1] & 0b11011111 : tbl[data1];
    SEG1 = ON;
    break;
case 1:
    seg_flg = 2;
    SEG1 = OFF;
    PORTB = (dot == 2) ? tbl[data2] & 0b11011111 : tbl[data2];
    SEG2 = ON;
    break;
case 2:
    seg_flg = 3;
    SEG2 = OFF;
    PORTB = (dot == 3) ? tbl[data3] & 0b11011111 : tbl[data3];
    SEG3 = ON;
    break;
case 3:
    seg_flg = 0;
    SEG3 = OFF;
    PORTB = (dot == 4) ? tbl[data4] & 0b11011111 : tbl[data4];
    SEG4 = ON;
    break;
}
}

//*****
*

unsigned long    FreqMeasurement()
{
    unsigned long    freq;
    //
    TRISA.F4 = 0;          //ゲートを閉める。
    PORTA.F4 = 0;
    INTCON.T0IF = 0;
    TMR0 = 0;
    freq = 0;
    //
    fc_flg = 0;            //測定開始
    //測定
    while (fc_flg != -1) {
        if (INTCON.T0IF == 1) {
            INTCON.T0IF = 0;
            freq++;
        }
    }
}
```

```
    if (INTCON.T0IF == 1) {
        INTCON.T0IF = 0;
        freq++;
    }
    freq = ((freq * 256) + TMR0) * 8 * 10;
    return (freq);
}

//*****
*

void main()
{
    static    char    buf[16];
    static    long    freq, tmp;
    //
    TRISA = 0b00100000;
    TRISB = 0b00000000;
    //    OSCCON = 0b01110000;    // クロックを8Mhzに設定する。
    ANSEL = 0b00000000;    // □□□変換は使用しない。
    // TIMER0の設定
    OPTION_REG.T0CS = 1;
    OPTION_REG.PSA = 0;
    OPTION_REG.PS2 = 0;
    OPTION_REG.PS1 = 1;
    OPTION_REG.PS0 = 0;
    // 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 = 0xE2;    // 5msec... (1÷8000000)*4*8*1250
    CCPR1H = 0x04;    //
    //
    SEG1 = OFF;
    SEG2 = OFF;
    SEG3 = OFF;
    SEG4 = OFF;
    seg_flg = 0;
    data1 = SPACE;
```

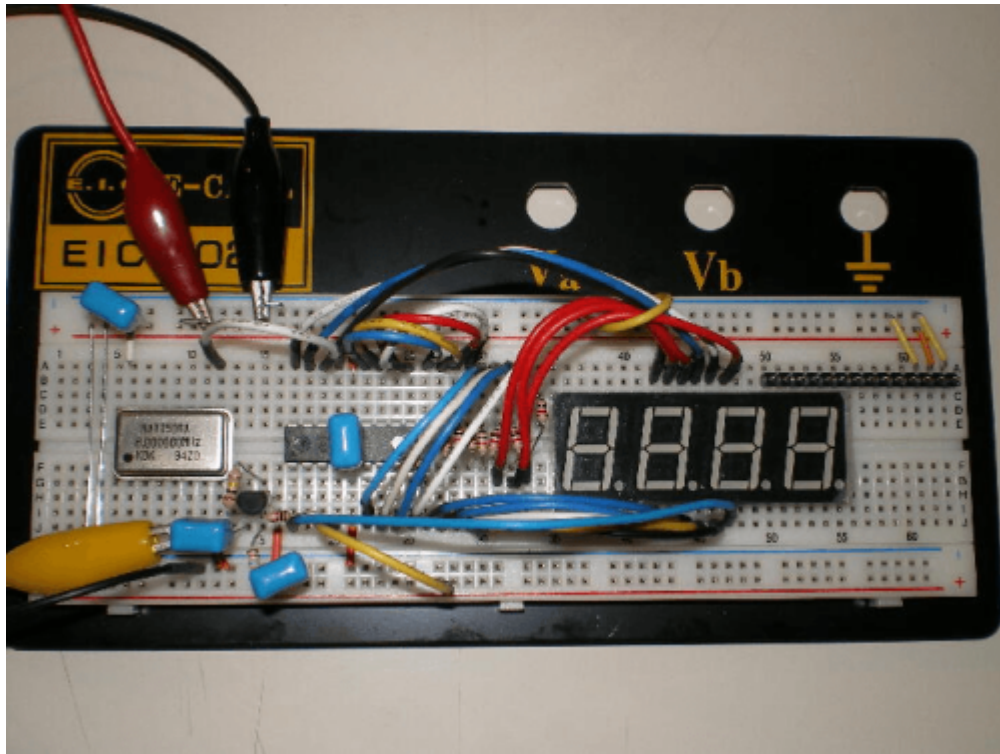
```

data2 = SPACE;
data3 = SPACE;
data4 = SPACE;
dot = 0;
fc_flg = -1;
// 割り込みを許可する。
INTCON.PEIE = 1;
INTCON.GIE = 1;
//
T1CON.TMR1ON = 1;
//
while (1) {
    //測定
    freq = FreqMeasurement();
    tmp = freq / 1000;
    //kHz未満四捨五入
    if ((freq - (tmp * 1000)) >= 500)
        tmp++;
    //-455kHz有無
    if (PORTA.F5 == 0)
        tmp = tmp - 455;
    //表示
    LongToStr(tmp, buf);
    if (tmp < 10000) {
        dot = 0;
        data1 = buf[7] == ' ' ? SPACE : buf[7] - '0';
        data2 = buf[8] == ' ' ? SPACE : buf[8] - '0';
        data3 = buf[9] == ' ' ? SPACE : buf[9] - '0';
        data4 = buf[10] == ' ' ? SPACE : buf[10] - '0';
    } else {
        dot = 2;
        data1 = buf[6] == ' ' ? SPACE : buf[6] - '0';
        data2 = buf[7] == ' ' ? SPACE : buf[7] - '0';
        data3 = buf[8] == ' ' ? SPACE : buf[8] - '0';
        data4 = buf[9] == ' ' ? SPACE : buf[9] - '0';
    }
    Delay_ms(500);
}
}

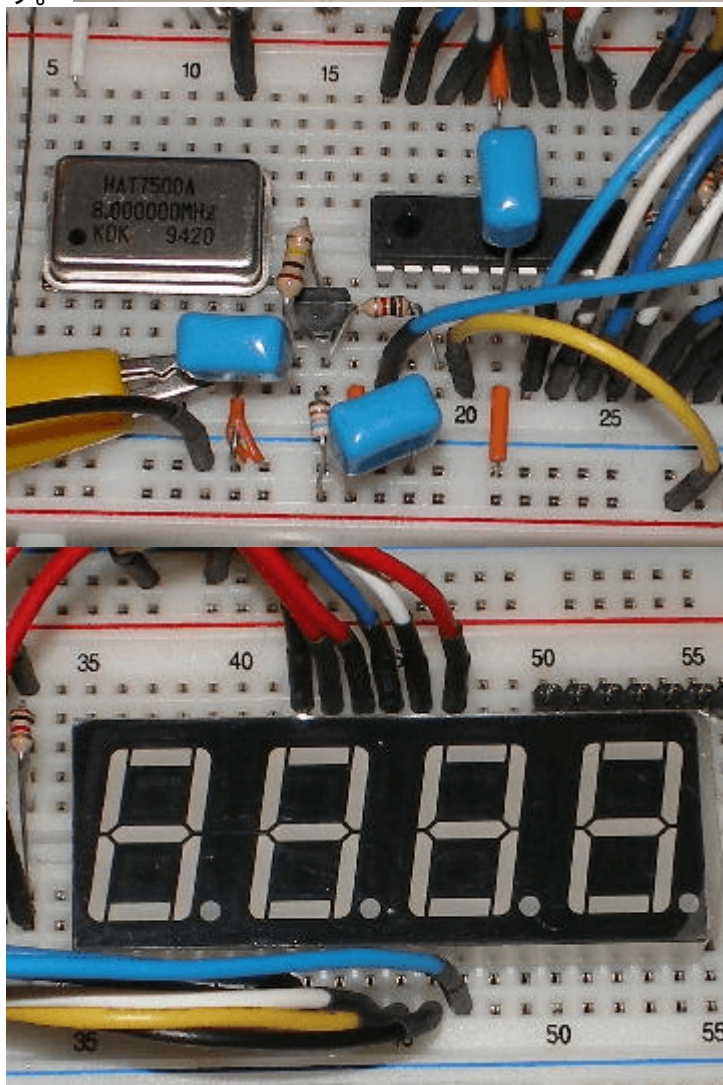
//*****
*
```

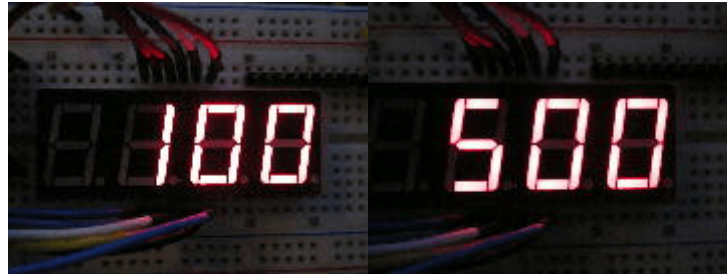
動作確認

いつものブレッドボードで動作確認しましたが、基板に実装すれば、かなりコンパクトになると思いま

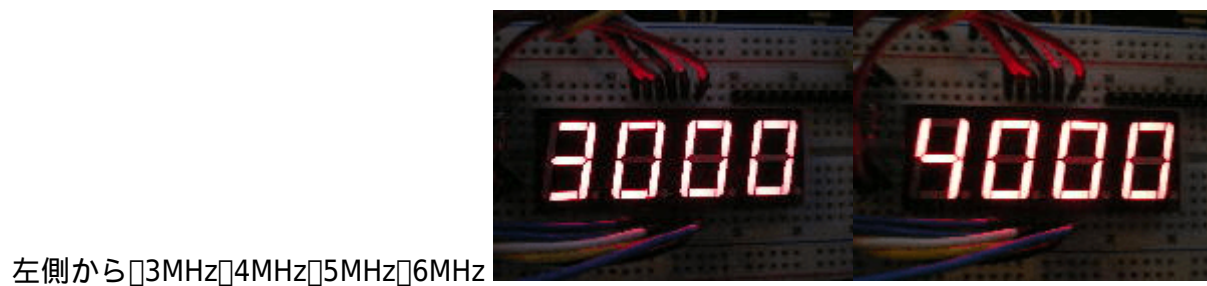
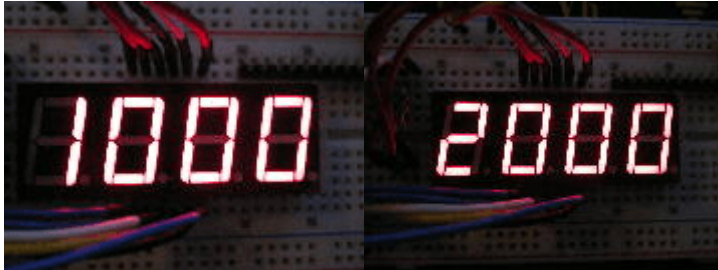


す。

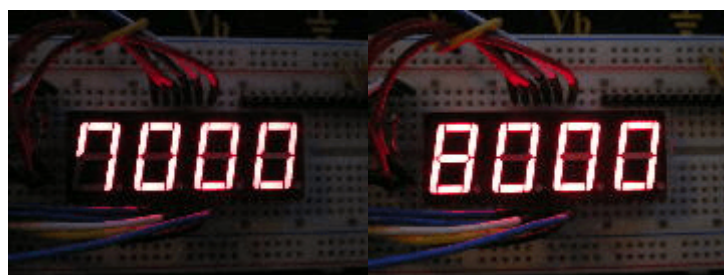
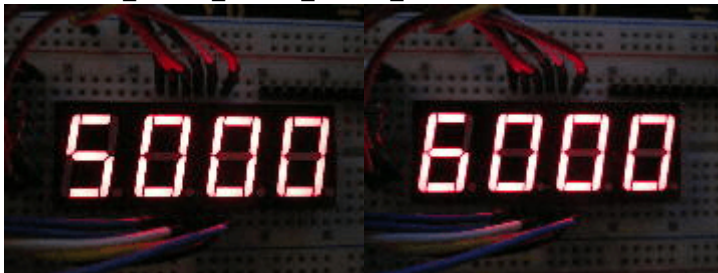




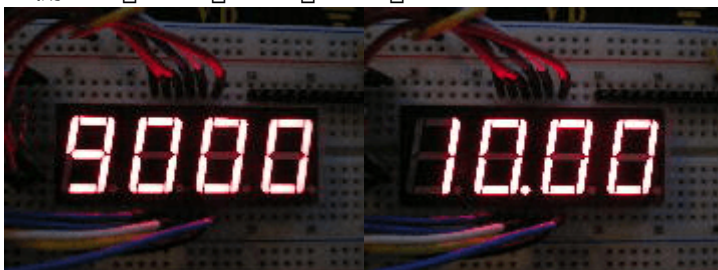
左側から□100kHz□500kHz□1MHz□2MHz

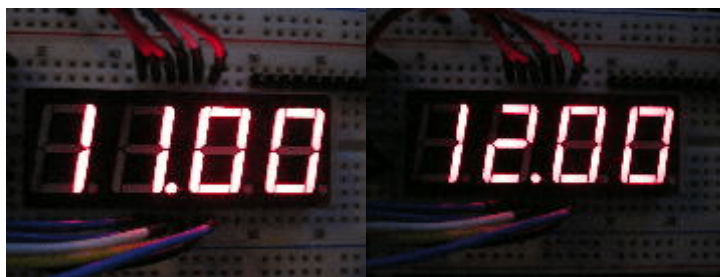


左側から□3MHz□4MHz□5MHz□6MHz

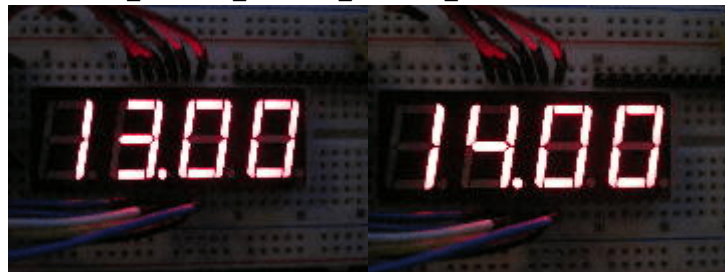


左側から□7MHz□8MHz□9MHz□10MHz

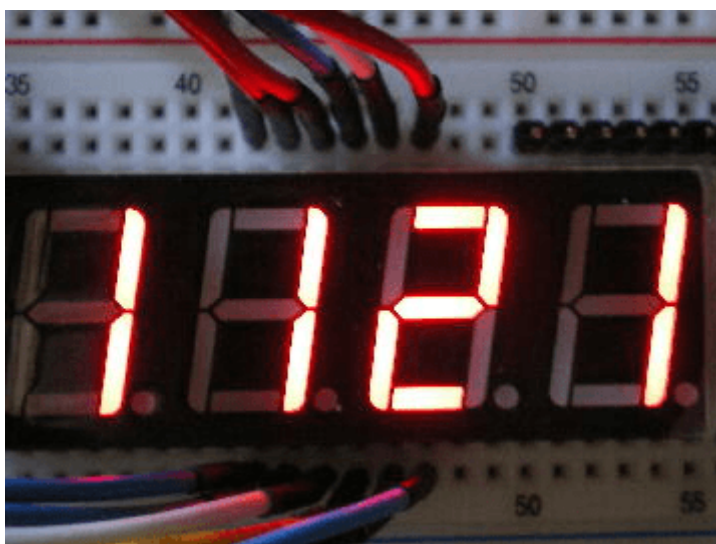
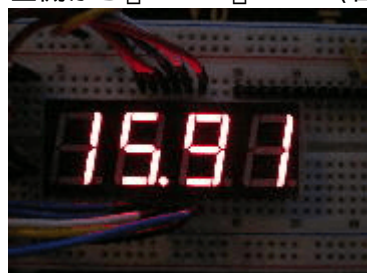




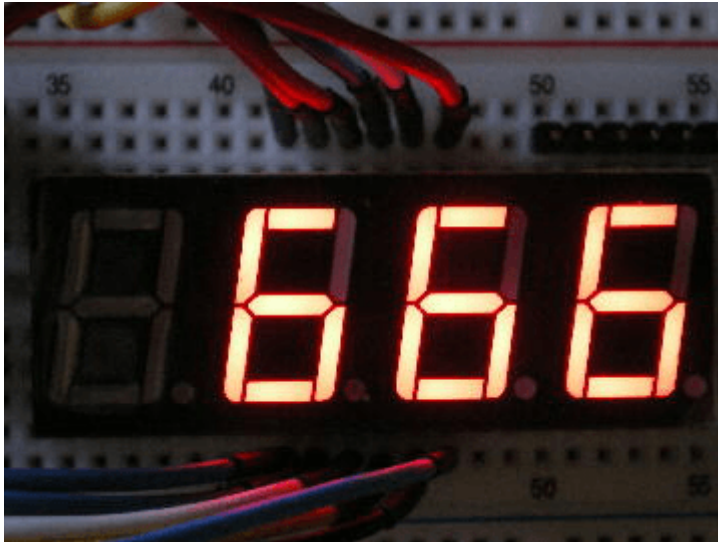
左側から□11MHz□12MHz□13MHz□14MHz



左側から□15MHz□16MHz(若干、精度が悪くなります)



左側:通常表示 右側:-455kHz表示



如何ですか? これを自作ラジオに取り付ければ、受信周波数を直読することが簡単に出来ますね。{😊}!

外付けの8MHzクリスタルオシレータを省いてPIC内蔵の8MHzクロックを使用することにより、精度は少し悪くなりますが、とてもコンパクトな、周波数カウンタを実現することが出来ます。

From:

<http://deepsky.jp/wiki/> - うごくといいな

Permanent link:

<http://deepsky.jp/wiki/doku.php?id=elechobby:picdic:pic16f88:110&rev=1588208009>

Last update: 2025/10/17 14:27

