周波数ロックユニットV3

概要

以前に周波数ロックユニットを作成しましたがPICを2個使用した、少し複雑な構成になっていました。そこで今回は、PICを1個だけ使用し、同等の機能を実現してみました。

動作原理

基本的には周波数カウンタの応用です。

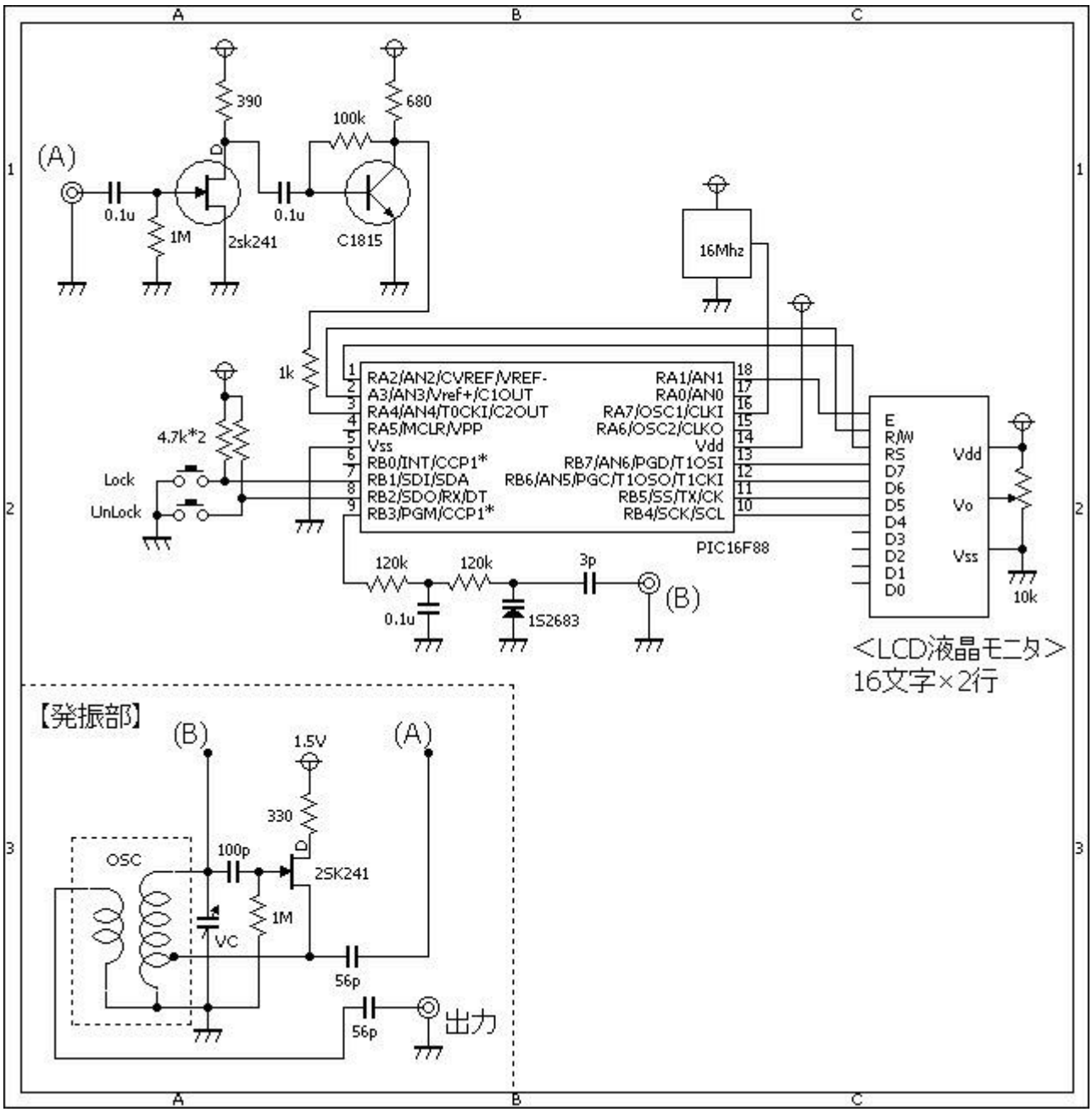
1. 通常は、周波数カウンタとして動作します。
2. ロックSWが押されると、その時の周波数F1を記憶します。
3. 周波数F2を測定します。
4. F1とF2比較し、F2が小さければ(100Hz以上)、発振周波数を上げるために、バリキャップの静電容量を小さくします。
5. F1とF2比較し、F2が大きければ(100Hz以上)、発振周波数を下げるために、バリキャップの静電容量を大きくします。
6. 1.に戻る。

<バリキャップ制御電圧を発生させる方法> PIC内臓のPWMモジュールを使用します。精度は10ビットなので約5mVです。範囲は、約0V~5Vです PWMの周波数は固定にし、

- デューティ値を大きくすることにより制御電圧を高くし、
- デューティ値を小さくすることにより制御電圧を低くします。

他にもD/A変換専用のICを使用する方法や、R-2Rラダー抵抗を使用する方法があります。好みに応じて選択してください。

回路図



ソースコード

FreqLockV3.c

```
/*
  <周波数カウンター>
*/

//*****************************************************************
*

#define   LOCK_SW                PORTB.F1
#define   UNLOCK_SW             PORTB.F2
```

```
//*****************************************************************************
*

void   interrupt()
{
    if (PIR1.TMR1IF == 1) {
        PIR1.TMR1IF = 0;
        //
        TRISA.F4 = 0;             //ゲートを閉める。
        PORTA.F4 = 0;
        T1CON.TMR1ON = 0;    // TIMER1を停止する。
    }
}

//*****************************************************************************
*

void   Pwm_Change_DutyEx(unsigned int duty_ratio)
{
    CCPR1L = duty_ratio >> 2;
    CCP1CON.F6 = duty_ratio & 0b00000001;
    CCP1CON.F7 = (duty_ratio & 0b00000010) >> 1;
}

//*****************************************************************************
*

unsigned   long    FreqMeasurement100msec()
{
    unsigned    long    freq;
    //
    TRISA.F4 = 0;         //ゲートを閉める。
    PORTA.F4 = 0;
    // TIMER0の設定
    INTCON.T0IF = 0;
    TMR0 = 0;
    // TIMER1の設定
    TMR1L = 0xB0;
    TMR1H = 0x3C;
    //
    freq = 0;
    //割り込みを許可する。
    INTCON.PEIE = 1;
    INTCON.GIE = 1;
    //開始
    T1CON.TMR1ON = 1;
    //
    Delay_Cyc(2);
    asm    nop;
    asm    nop;
    asm    nop;
```

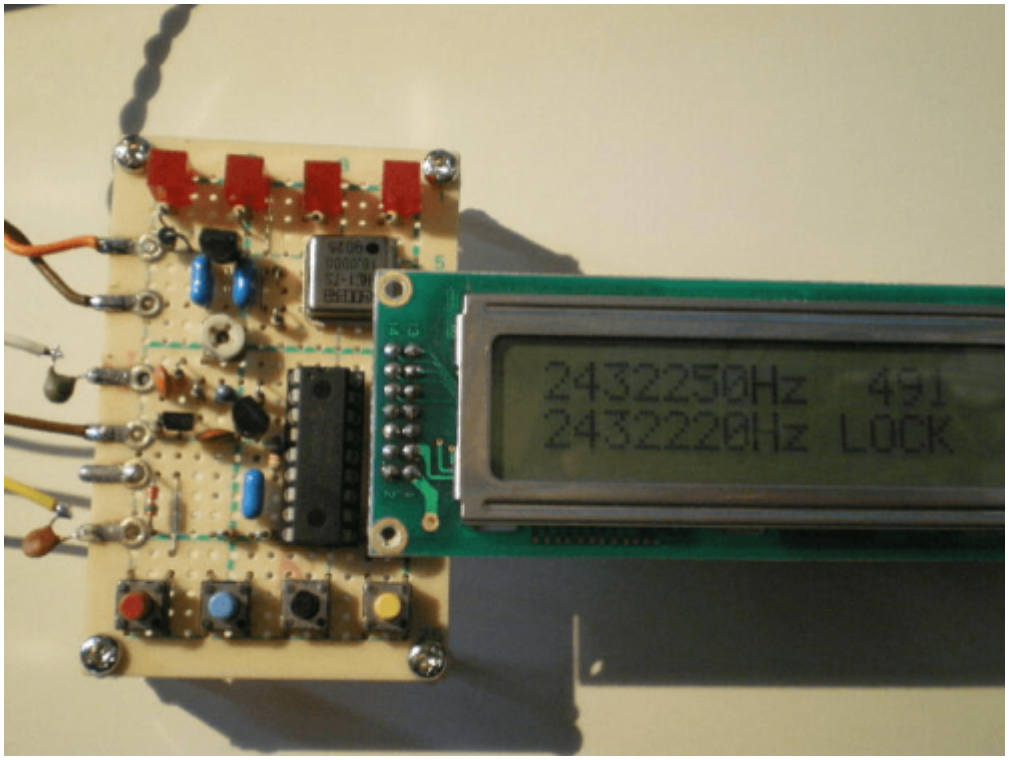
```
    asm    nop;
    asm    nop;
    asm    nop;
    asm    nop;
    //
    TRISA.F4 = 1;         //ゲートを開ける。
    //測定
    while (T1CON.TMR1ON != 0) {
        if (INTCON.T0IF == 1) {
            INTCON.T0IF = 0;
            freq++;
        }
    }
    if (INTCON.T0IF == 1) {
        INTCON.T0IF = 0;
        freq++;
    }
    freq *= 256;
    freq += TMR0;
    return (freq);
}

//*****************************************************************************
*

void main()
{
    static    unsigned    long    freq, lockFreq;    // 0...4294967295
    static    unsigned    char    buf[12];
    static    unsigned    short    cnt;
    static    unsigned    int        duty;
    static    unsigned    short    lockMode;
    //アナログの設定
//    OSCCON = 0b01110000;         // クッロクを8Mhzに設定する。
    ANSEL  = 0b00000000;    //→今回は使用しない。
    //ポートの設定
    TRISA = 0b11110001;
    TRISB = 0b00000111;
    OPTION_REG.F7 = 0;    // PORTBをプルアップ設定する。
    // TIMER0の設定
    INTCON.T0IE = 0;
    INTCON.T0IF = 0;
    OPTION_REG.T0CS = 1;
    OPTION_REG.T0SE = 0;
    OPTION_REG.PSA  = 1;
    OPTION_REG.PS0  = 0;
    OPTION_REG.PS1  = 0;
    OPTION_REG.PS2  = 0;
    // TIMER1の設定
    PIE1.TMR1IE = 1;
```

```
    PIR1.TMR1IF = 0;
    T1CON.T1CKPS0 = 1;
    T1CON.T1CKPS1 = 1;
    T1CON.TMR1ON = 0;
    //
    Lcd_Custom_Config(&PORTB,7,6,5,4,&PORTA,2,3,1);
    Lcd_Custom_Cmd(LCD_CURSOR_OFF);
    Lcd_Custom_Out(1, 1, "FreqLockV3");
    Delay_ms(1000);
    Lcd_Custom_Cmd(LCD_CLEAR);
    //
    Pwm_Init(100000);    // 100Khz
    PR2 = 0xFF;
    duty = 1024 / 2;
    Pwm_Change_DutyEx(duty);
    Pwm_Start();
    lockMode = 0;
    //
    while (1) {
        freq = 0;
        for (cnt = 0; cnt < 1; cnt++) {
            freq += FreqMeasurement100msec();
        }
        freq *= 10;
        LongToStr(freq, buf);
        Lcd_Custom_Out(1, 1, &buf[3]);
        Lcd_Custom_Out(1, 9, "Hz ");
        //
        if (LOCK_SW == 0) {
            lockMode = 1;
            lockFreq = freq;
            Lcd_Custom_Out(2, 1, &buf[3]);
            Lcd_Custom_Out(2, 9, "Hz ");
        }
        //
        if (UNLOCK_SW == 0) {
            lockMode = 0;
            Lcd_Custom_Out(1, 12, "    ");
            Lcd_Custom_Out(2, 1, "                ");
            duty = 1024 / 2;
            Pwm_Change_DutyEx(duty);        //バリキャップの電圧を中央にする。
        }
        //
        if (lockMode == 1) {
            if ((freq > (lockFreq - 100)) && (freq < (lockFreq + 100)))
{
                Lcd_Custom_Out(2, 12, "LOCK ");
            }
            if ((freq < (lockFreq - 100))) {
                Lcd_Custom_Out(2, 12, "LOWER");
                duty++;
```

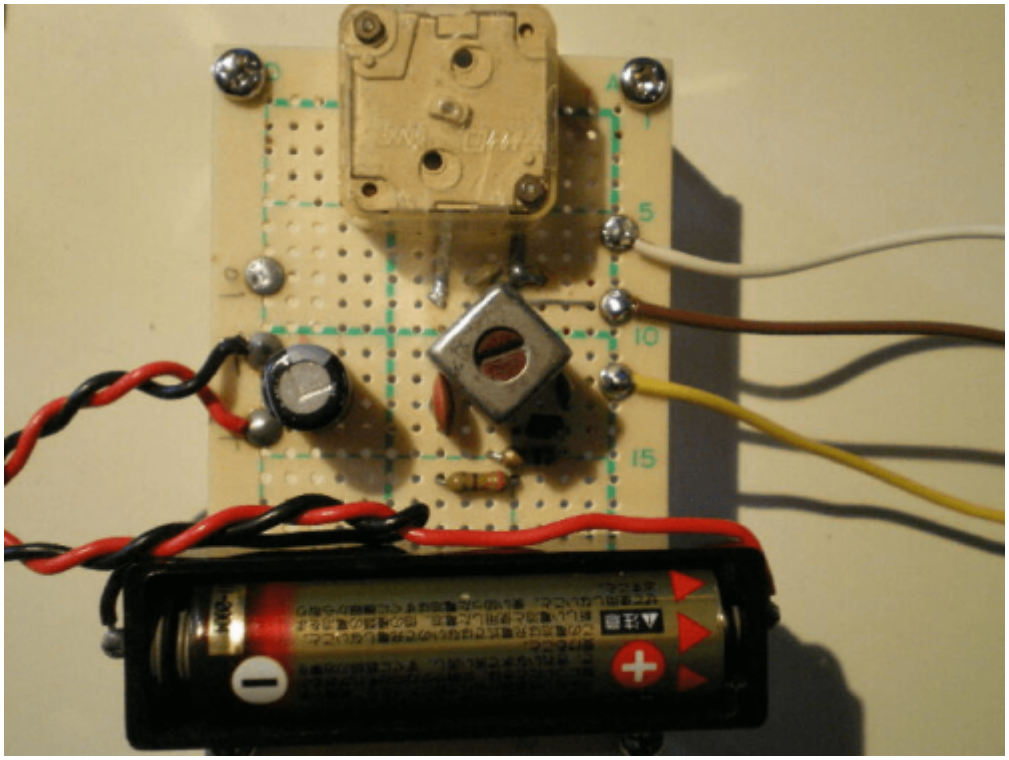
```
                Pwm_Change_DutyEx(duty);     //周波数が低いのでバリキャップの
電圧を高くする。
            }
            if ((freq > (lockFreq + 100))) {
                Lcd_Custom_Out(2, 12, "UPPER");
                duty--;
                Pwm_Change_DutyEx(duty);     //周波数が高いのでバリキャップの
電圧を低くする。
            }
            LongToStr(duty, buf);
            Lcd_Custom_Out(1, 12, &buf[7]);
        }
    }
}

//*****************************************************************************
*
```

動作確認

基板を流用したので、余計な部品(4個のLED□赤色SW□黄色SW)が付いていますが無視してください。 黒色SWはLock用、青色SWはUnLock用です。



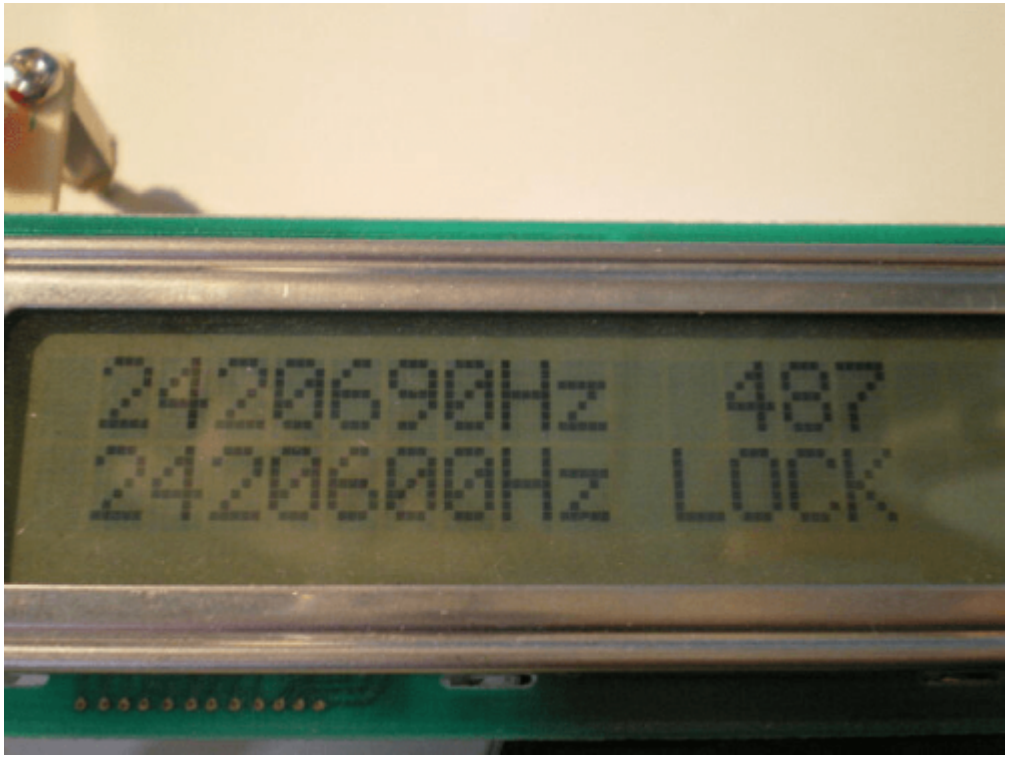
発振部の一例です□(FET1個使用) 手元にあった部品で作成しました。乾電池1本で動作します。周波数範囲は約500kHz~2.5MHzまでです。



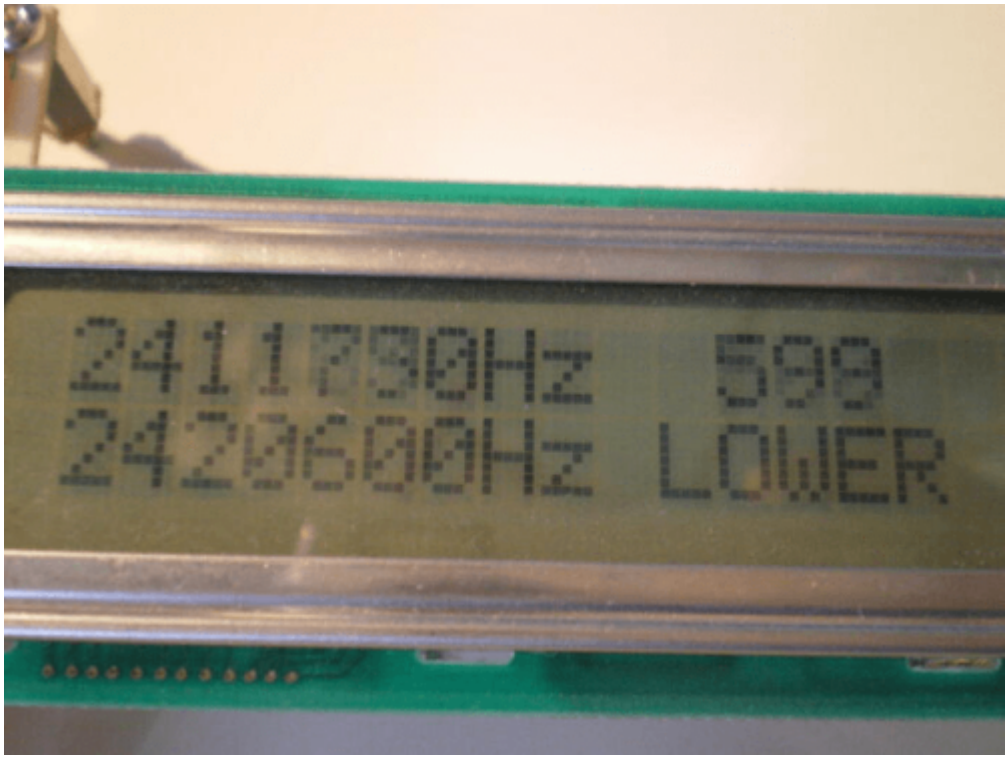
UnLockの状態です。通常の周波数カウンタと同じです。精度は10Hzです。ゲートタイムは0.1秒です。

Lockした状態です±100Hz以内にロックします。 上の周波数が現時点の周波数です。 下の周波数がロック周波数です。(この周波数に合わせるように制御します) 右上の数値は、バリキャップ(バリアブルコンデンサ)の制御値です。 この制御値に約5mVを掛けると“ バリキャップ制御電圧 “ になります。

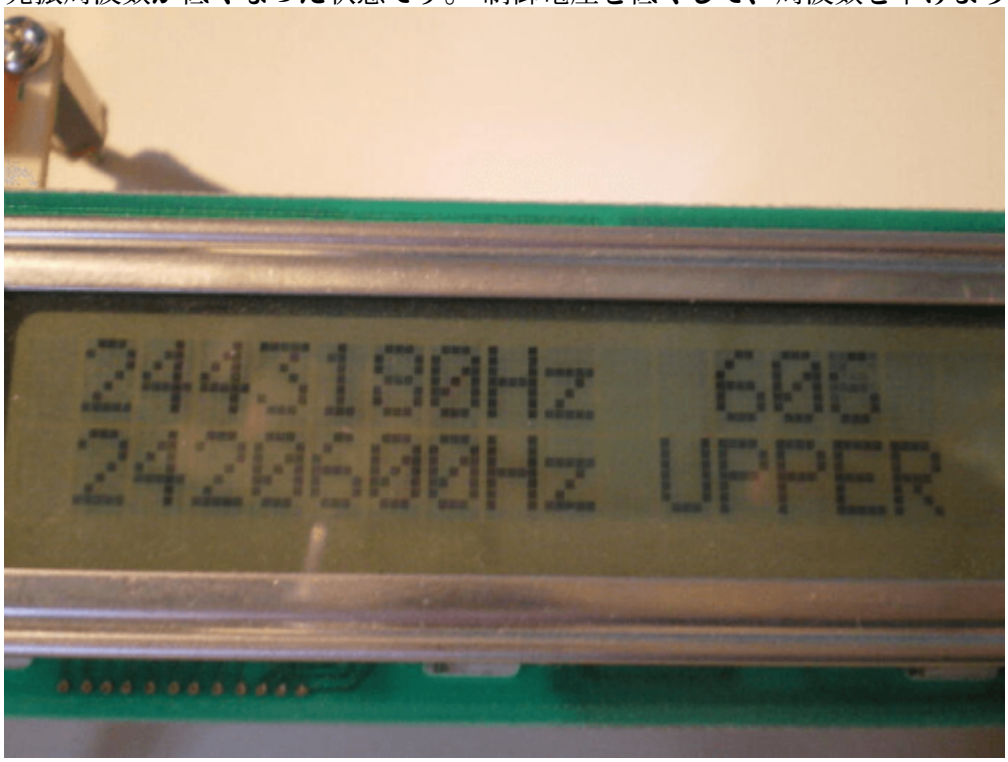
- 中央値は512です。(約2.5V)
- 数値が大きくなると静電容量が小さくなるので、発振周波数は高くなります。
- 数値が小さくなると静電容量が大きくなるので、発振周波数は低くなります。



発振周波数が低くなった状態です。 バリキャップ制御電圧を高くして、周波数を上げようとします。



発振周波数が低くなった状態です。 制御電圧を低くして、周波数を下げようとします。



如何ですか? 真空管ラジオの局部発振のように周波数変動が大きい発振器に本ユニットを付加すると安定度の高い発振器へと生まれ変わりますね。。。 😄