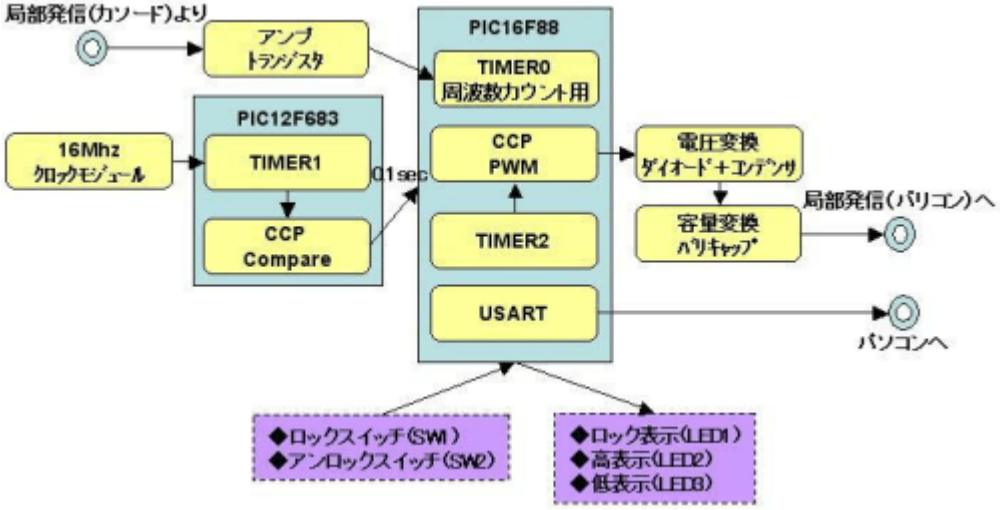
周波数ロックユニット(じっとしてなさい君V2)

概要

ラジオの周波数(局部発振周波数)をロック(固定)するためのものです。 真空管ラジオなどの時間の経過と共に周波数が変動するような場合に便利です。

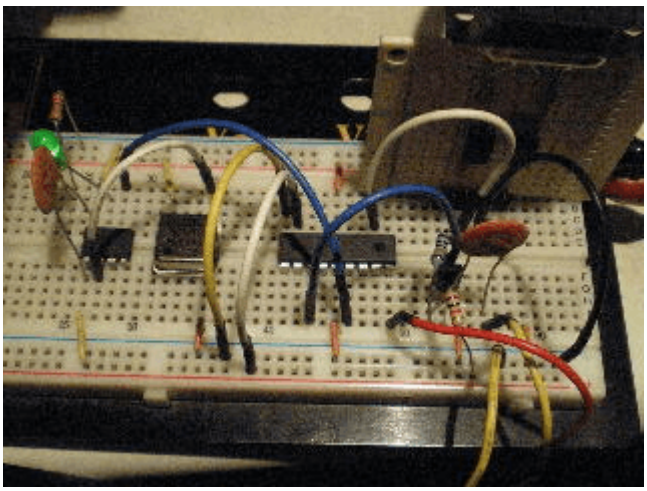
周波数ロックユニットの構造原理



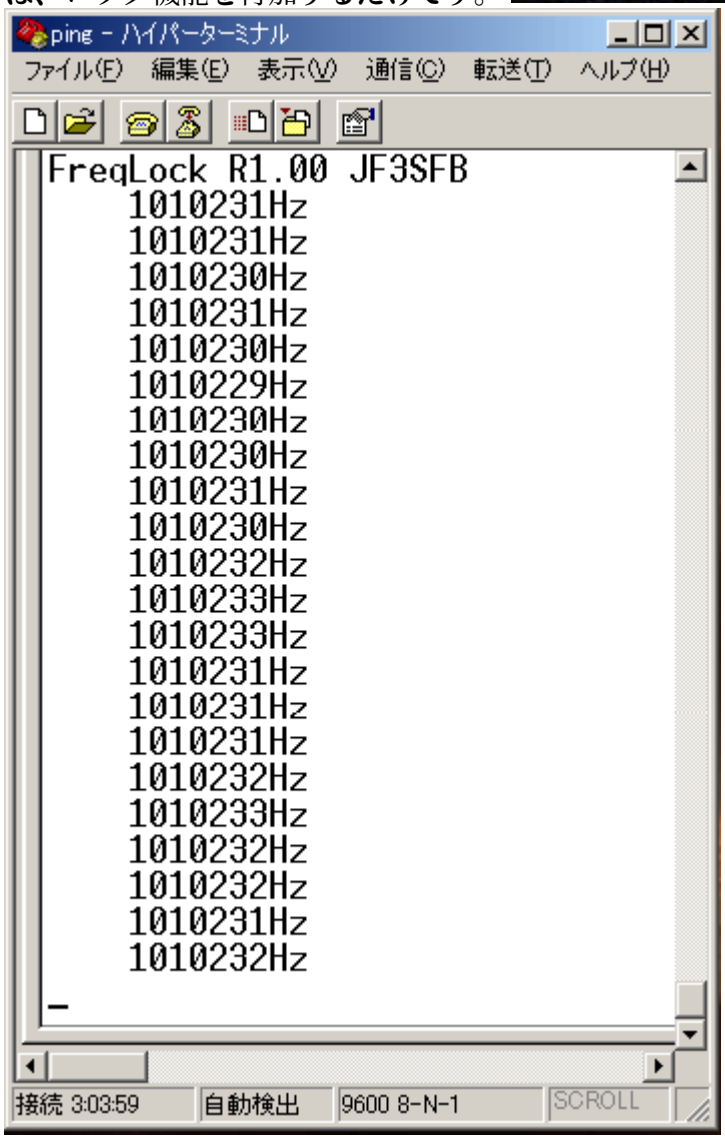
- CCPモジュールを2個使用するためPICを2個使用しました。
- PIC12F683では、正確なクロック(0.1秒)を出力します。
- PIC16F88では、周波数をカウントし、その値によって出力電圧を制御します。
- 出力電圧は、可変容量ダイオード(バリキャップ (variable capacitance diode)、バラクタ (varactor diode))
  1S2683によって、コンデンサ(容量)へと変換されます。
- カウントした周波数は、USART(RS232C)経由で送信可能です。(9600,8,N)
  ※SW1とSW2を同時に押しながら起動すると送信モードになります。
- 周波数の精度は、最大2.5Mhz(1Hz単位)、最大20Mhz(8Hz単位)が可能です。
- ゲートタイムは、1秒、0.1秒が可能です。
- ロック精度は、±100Hzとします。(プログラム上で変更可能です)
- 出力電圧は、0V~5Vで精度は約5mV(5÷1024)になります。
- この電圧をバリキャップに加えます。
- 使用するバリキャップによって可変可能な容量(キャパシタンス)は異なってきます。(要注意)

実験

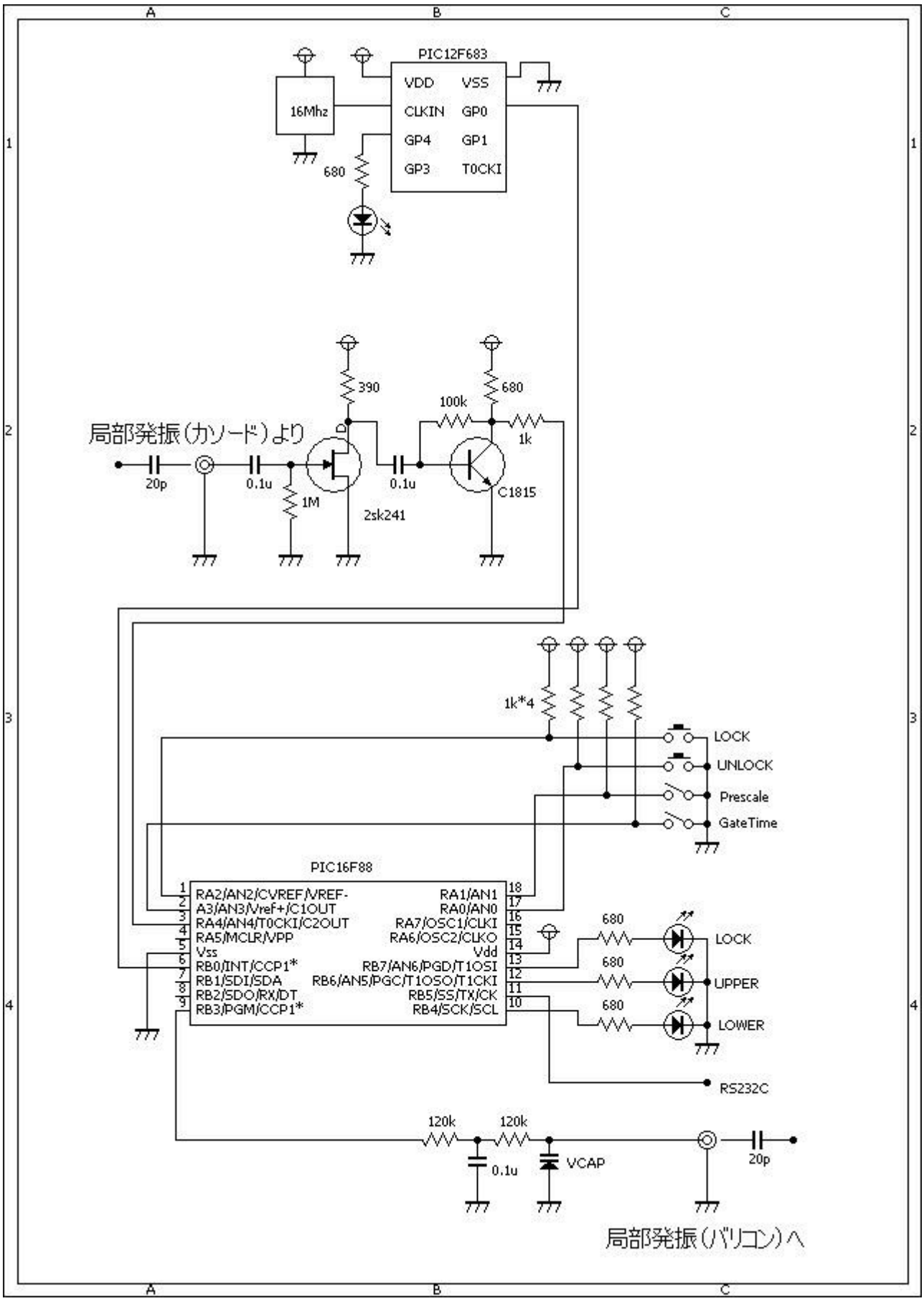
PIC12F683とPIC16F88を使用した、基本的な周波数カウンタとしての機能は確認できました。 これに後

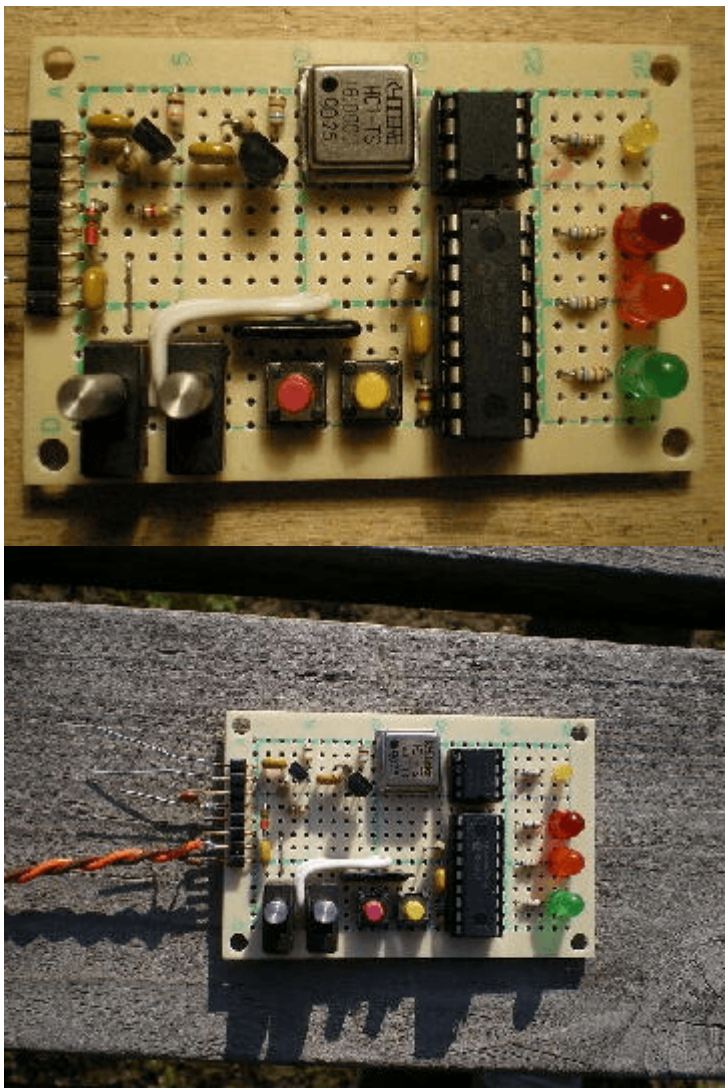


は、ロック機能を付加するだけです。



回路図


PIC12F683
VDD
VSS
CLKIN
GP0
GP4
GP1
GP3
T0CKI
16Mhz
680
390
680
100k
1k
局部発振(カソード)より
20p
0.1u
1M
0.1u
2sk241
C1815
1k*4
LOCK
UNLOCK
Prescale
GateTime
PIC16F88
RA2/AN2/CVREF/VREF-
A3/AN3/Vref+/C1OUT
RA4/AN4/T0CKI/C2OUT
RA5/MCLR/VPP
Vss
RB0/INT/CCP1*
RB1/SDI/SDA
RB2/SDO/RX/DT
RB3/PGM/CCP1*
RA1/AN1
RA0/AN0
RA7/OSC1/CLKI
RA6/OSC2/CLKO
Vdd
RB7/AN6/PGD/T1OSI
RB6/AN5/PGC/T1OSO/T1CKI
RB5/SS/TX/CK
RB4/SCK/SCL
680
LOCK
680
UPPER
680
LOWER
RS232C
120k
120k
0.1u
VCAP
20p
局部発振(バリコン)へ



- LOCK_LED(緑色)
- UPPER_LED(橙色)
- LOWER_LED(赤色)

処理説明

<PIC12F683の処理> 0.1sec(10Hz)の正確なクロックを出力するだけの単純な機能です CCPをコンペアモードで使用して、0.1秒周期で割り込みを発生させ、短いパルスを出力します。

- クッロクを16Mhzとする。プリスケーラを8とするCCPの設定値を50000とする。
- 0.1 = (1÷16000000)*4*8*50000

<PIC16F88の処理>

1. 内臓モジュールやポート等の初期化をします。
2. PWMのdutyを50%(1024/2)にします。
3. LOCK_SWとUNLOCK_SWが共に押されていたらRS232C送信モードとします。
4. CLEDを点滅させます。(特に意味のある処理ではありません)
5. RS232C送信モードであれば、タイトルを送信します。
6. 割り込みを許可します。
7. プリスケーラの設定を行います。

8. Gate Timeの設定を行います。
9. タイマ値や周波数値をクリアします。
10. 設定されたGate Timeの間だけ計測を行います。
11. 計測された周波数を補正します。
12. RS232C送信モードであれば、周波数を文字列に変換して送信します。
13. LOCKするかUNLOCKするかをチェックします。
14. もしもロックモードなら、
    - 現在の周波数の値を基準値(f1)として保存します。
    - 計測した周波数(f2)と基準値(f1)を比較し許容範囲ならロックと判断します。
    - (f2)<(f1)であれば□PWMのdutyをプラスして電圧を上げます。
    - (f2)>(f1)であれば□PWMのdutyをマイナスして電圧を下げます。
15. もしもアンロックモードなら、
    - PWMのdutyをデフォルト値(50%(1024/2))に設定します。
16. 少しディレイ(10msec)してから7.へ戻ります。

ソースコード

PIC12F683のソースコード

Clock100msec.c

```
/*
□0.1sec□10Hz□の正確なクロックを出力する。

□CCPをコンペアモードで使用して、0.1秒周期で割り込みを発生させる。
  クッロクを16Mhzとする。プリスケーラを8とする□CCPの設定値を50000とする。
□0.1 = (1÷16000000)*4*8*50000

*/

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

void   interrupt(){
    if (PIR1.CCP1IF == 1) {
        PIR1.CCP1IF = 0;
        //
        GPIO.F0 = 1;
        GPIO.F4 = ~GPIO.F4;
        GPIO.F0 = 0;
    }
}

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

void main()
{
    // クロックの設定　→今回は外付けの16Mhzクロックモジュールを使用する。
```

```
    OSCCON = 0b01110000;
    //コンパレータの設定　→今回は使用しない。
    CMCON0 = 0b00000111;
    //アナログの設定　→今回は使用しない。
    ANSEL  = 0b00000000;
    //ポートの設定
    TRISIO = 0b00101000;
    OPTION_REG.F7 = 0;
    //入力割り込みの設定　→今回は使用しない。
    INTCON.INTE = 0;
    INTCON.INTF = 0;
    OPTION_REG.INTEDG = 0;
    //入力割り込み（変化）の設定　→今回は使用しない。
    INTCON.GPIE = 0;
    INTCON.GPIF = 0;
    // CCPの設定
    PIE1.CCP1IE = 1;
    PIR1.CCP1IF = 0;
    CCP1CON = 0b00001011;
    CCPR1L = 0x50;    // 0.1sec...10hz...クロックが16Mhzの時
    CCPR1H = 0xC3;    // 0.1sec...(1÷16000000)*4*8*50000
    // TIMER0の設定　→今回は使用しない。
    INTCON.T0IE = 0;
    INTCON.T0IF = 0;
    TMR0 = 0;
    OPTION_REG.T0CS = 0;
    OPTION_REG.T0SE = 0;
    OPTION_REG.PSA  = 0;
    OPTION_REG.PS0  = 0;
    OPTION_REG.PS1  = 0;
    OPTION_REG.PS2  = 0;
    // TIMER1の設定
    PIE1.TMR1IE = 0;
    PIR1.TMR1IF = 0;
    TMR1L = 0;
    TMR1H = 0;
    T1CON.T1CKPS0 = 1;
    T1CON.T1CKPS1 = 1;
    T1CON.TMR1ON = 1;
    // TIMER2の設定　→今回は使用しない。
    PIE1.TMR2IE = 0;
    PIR1.TMR2IF = 0;
    T2CON.TMR2ON = 0;
    T2CON.T2CKPS0 = 0;
    T2CON.T2CKPS1 = 0;
    TMR2 = 0;
    //割り込み(全体)の設定
    INTCON.PEIE = 1;
    INTCON.GIE = 1;
    while (1) {
```

```
    }
}

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

PIC16F88のソースコード

FreqLock.c

```
/*
  ＜周波数ロックユニット＞
*/

#define   PRESCALE_SW    PORTA.F1
#define   GATETIME_SW    PORTA.F3

#define   LOCK_SW        0b00000100
#define   UNLOCK_SW    0b00000001

#define   LOCK_LED    PORTB.F4
#define   UPPER_LED    PORTB.F6
#define   LOWER_LED    PORTB.F7

#define   LOCK_MODE    1
#define   UNLOCK_MODE    0

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

static short    endFlag;
static short    gateTime;

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

void   interrupt(){
    if (INTCON.INTF == 1) {
        INTCON.INTF = 0;
        //
        if (endFlag == 0xFF) {    //何もしない。
            return;
        }
        if (endFlag == 0) {
            TRISA.F4 = 1;         //ゲートを開ける。
            endFlag++;
            return;
        }
        endFlag++;
        if (endFlag > gateTime) {
```

```
            TRISA.F4 = 0;           //ゲートを閉める。
            PORTA.F4 = 0;
            endFlag = 0xFF;
            return;
        }
    }
}

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

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

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

void   Usart_Write_String(char *buf)
{
    short     len, i;
    len = strlen(buf);
    for (i = 0; i < len; i++) {
        Usart_Write(buf[i]);
    }
}

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

void main()
{
    static    unsigned    short    rs232cSendFlag;
    static    unsigned    short    mode;
    static    unsigned    long    freq;        // 0...4294967295
    static    unsigned    long    freqSave;    // 0...4294967295
    static    unsigned    char    buf[20];
    static    short                prescaler;
    static    unsigned    int        duty;
    //クロックの設定(8Mhz)
    OSCCON = 0b01110000;
    //コンパレータの設定　→今回は使用しない。
    CMCON = 0b00000111;
    //アナログの設定　→今回は使用しない。
    ANSEL  = 0b00000000;
    //ポートの設定
    TRISA = 0b11111111;
```

```
    TRISB = 0b00000101;
    OPTION_REG.F7 = 0;
    //
    INTCON.PEIE = 0;
    INTCON.GIE = 0;
    //入力割り込みの設定
    INTCON.INTE = 1;
    INTCON.INTF = 0;
    OPTION_REG.INTEDG = 0;
    //入力割り込み（変化）の設定　→今回は使用しない。
    INTCON.RBIE = 0;
    INTCON.RBIF = 0;
    // CCPの設定　→今回は使用しない。
    PIE1.CCP1IE = 0;
    PIR1.CCP1IF = 0;
    CCP1CON = 0b00001111;
    CCPR1L = 0;
    CCPR1H = 0;
    // TIMER0の設定
    INTCON.T0IE = 0;
    INTCON.T0IF = 0;
    TMR0 = 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　→今回は使用しない。
    PIE1.TMR1IE = 0;
    PIR1.TMR1IF = 0;
    TMR1L = 0;
    TMR1H = 0;
    T1CON.T1CKPS0 = 0;
    T1CON.T1CKPS1 = 0;
    T1CON.TMR1ON = 0;
    // TIMER2の設定　→今回は使用しない。
    PIE1.TMR2IE = 0;
    PIR1.TMR2IF = 0;
    T2CON.TMR2ON = 0;
    T2CON.T2CKPS0 = 0;
    T2CON.T2CKPS1 = 0;
    TMR2 = 0;
    //
    TRISA.F4 = 0;
    PORTA.F4 = 0;
    endFlag = 0xFF;
    duty = 1024 / 2;
    //
    Pwm_Init(8000);    // 8Khz
    PR2 = 0xFF;
```

```
    Pwm_Change_DutyEx(duty);
    Pwm_Start();
    //
    if (((LOCK_SW & PORTA) == 0) && ((UNLOCK_SW & PORTA) == 0))
        rs232cSendFlag = 1;
    else
        rs232cSendFlag = 0;
    //
    Usart_Init(9600);
    LOCK_LED = 0;
    UPPER_LED = 0;
    LOWER_LED = 0;
    Delay_ms(250);
    LOCK_LED = 1;
    Delay_ms(250);
    UPPER_LED = 1;
    Delay_ms(250);
    LOWER_LED = 1;
    Delay_ms(250);
    LOCK_LED = 0;
    UPPER_LED = 0;
    LOWER_LED = 0;
    //
    if (rs232cSendFlag == 1) {          // RS232Cに出力するか？
        Usart_Write_String("FreqLock R1.00 JF3SFB\r\n");
    }
    //
    TRISB.F0 = 1;
    //割り込み(全体)の設定
    INTCON.PEIE = 1;
    INTCON.GIE = 1;
    while (1) {
        if (PRESCALE_SW == 1) {     //プリスケーラの切り替え
            OPTION_REG.PSA  = 1;
            OPTION_REG.PS0  = 0;
            OPTION_REG.PS1  = 0;
            prescaler = 1;
        } else {
            OPTION_REG.PSA  = 0;
            OPTION_REG.PS0  = 0;
            OPTION_REG.PS1  = 1;
            prescaler = 8;
        }
        if (GATETIME_SW == 1) {     // Gate Timeの切り替え
            gateTime = 1;
        } else {
            gateTime = 10;
        }
        //初期化
        TMR0 = 0;
```

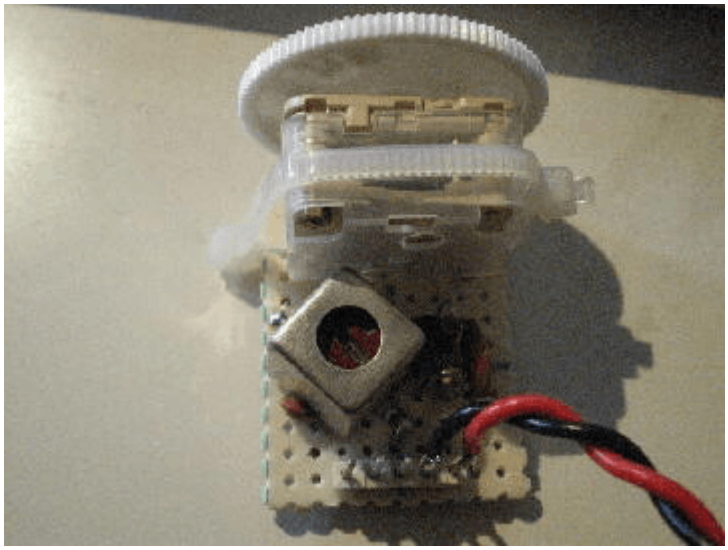
```
        INTCON.T0IF = 0;
        freq = 0;
        //開始
        endFlag = 0;
        //測定
        while (endFlag != 0xFF) {
            if (INTCON.T0IF == 1) {
                INTCON.T0IF = 0;
                freq++;
            }
        }
        freq *= 256;
        freq += TMR0;
        //補正
        if (prescaler == 8)
            freq *= 8;
        if (gateTime == 1)
            freq *= 10;
        //表示
        if (rs232cSendFlag == 1) {          // RS232Cに出力するか？
            LongToStr(freq, buf);
            Usart_Write_String(buf);
            Usart_Write_String("Hz\r\n");
        }
        //
        if ((LOCK_SW & PORTA) == 0) {          // LOCKスイッチが押されたか？
            mode = LOCK_MODE;
            freqSave = freq;
        }
        if ((UNLOCK_SW & PORTA) == 0) {     // UNLOCKスイッチが押されたか？
            mode = UNLOCK_MODE;
        }
        //
        if (mode == LOCK_MODE) {     // LOCKモード
            if ((freq > (freqSave - 100)) && (freq < (freqSave + 100)))
{
                LOCK_LED = 1;
                UPPER_LED = 0;
                LOWER_LED = 0;
            }
            if (freq < (freqSave - 100)) {
                LOCK_LED = 0;
                UPPER_LED = 0;
                LOWER_LED = 1;
                duty++;
                Pwm_Change_DutyEx(duty);     //周波数が低いので電圧を高くする。
            }
            if (freq > (freqSave + 100)) {
                LOCK_LED = 0;
                UPPER_LED = 1;
                LOWER_LED = 0;
```

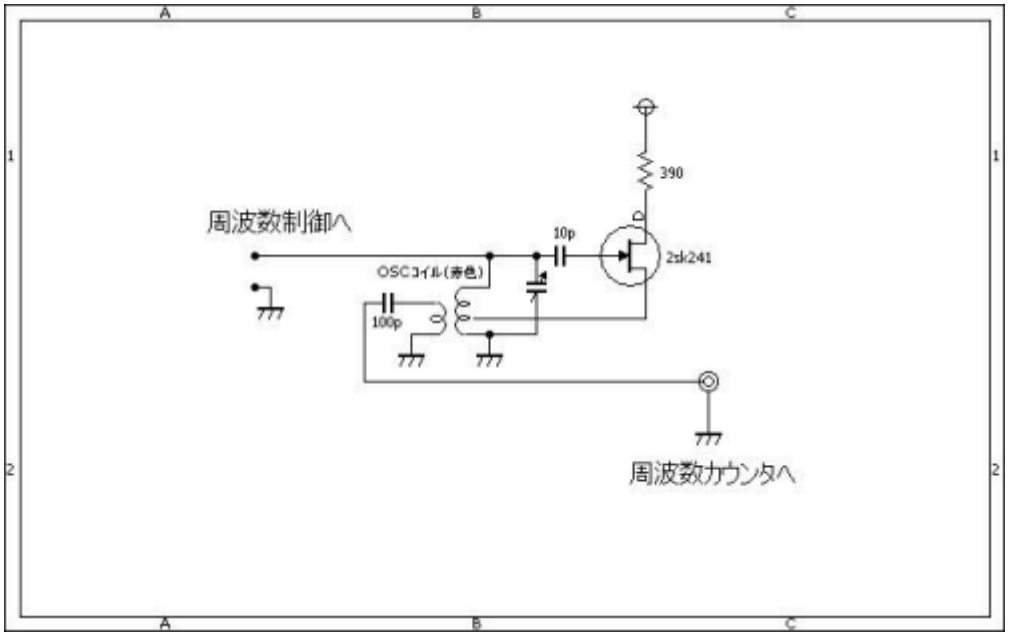
```
                duty--;
                Pwm_Change_DutyEx(duty);      //周波数が高いので電圧を低くする。
            }
        } else {    // UNLOCKモード
            LOCK_LED = 0;
            UPPER_LED = 0;
            LOWER_LED = 0;
            duty = 1024 / 2;
            Pwm_Change_DutyEx(duty);          //電圧を中央にする。
        }
        //
        Delay_ms(10);
    }
}

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

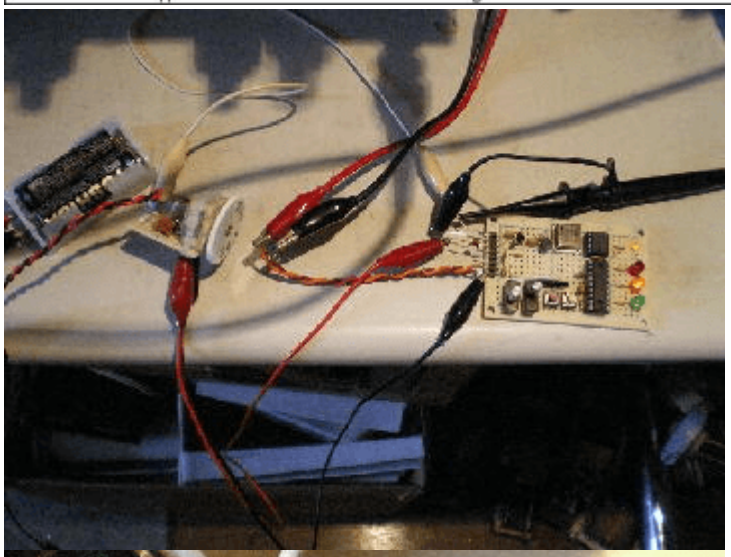
動作確認

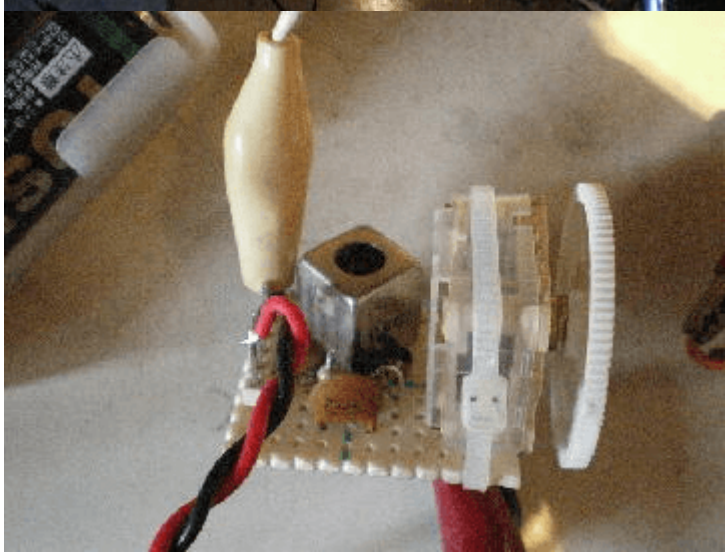
確認用に急遽作成したFETを使用した簡易発信器です。



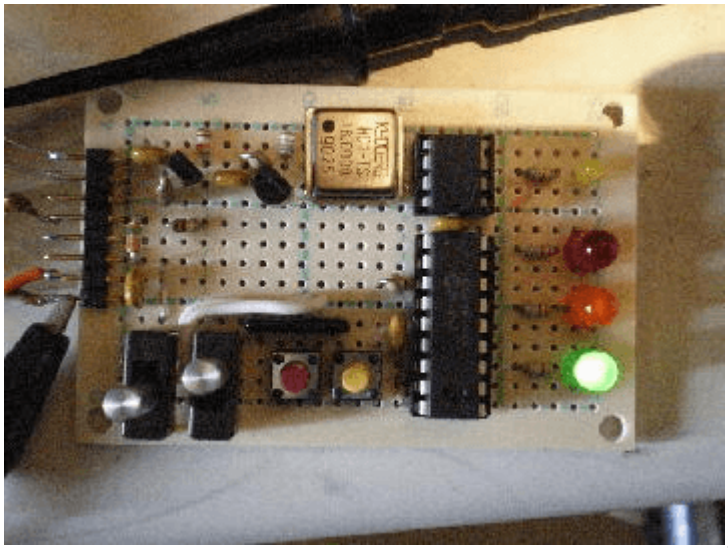


全体の接続の様子です。

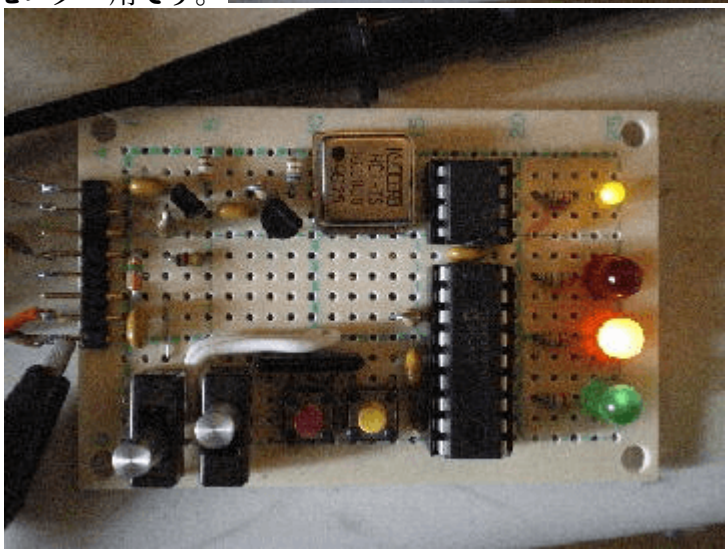




ロックした状態です。右上の黄色い小さなLED

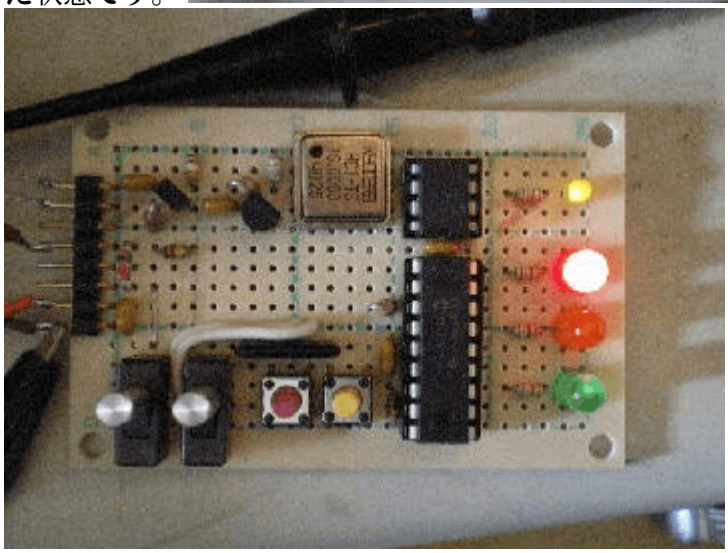


はクロックのモニター用です。

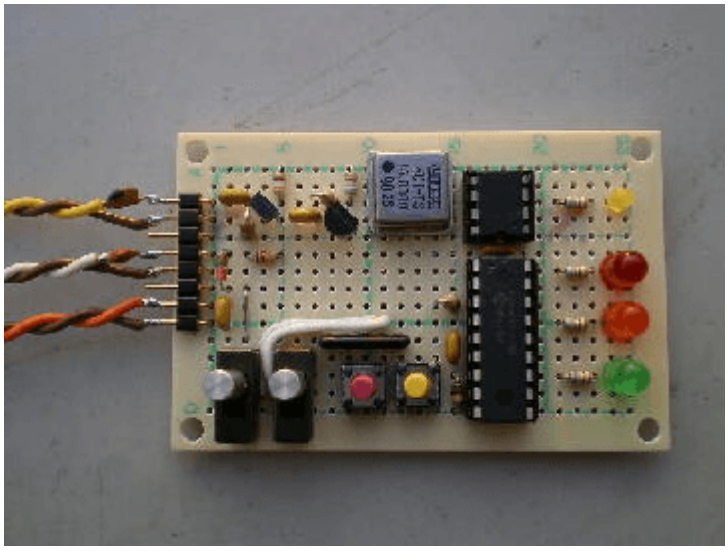


周波数が高くなった状態です。

周波数が低くなった状態です。



取り扱い説明



<接続> 茶色:全てGNDです。 黄色:周波数を入力します。 白色:周波数を制御します。 橙色:+5V(安定化電圧)を入力します。

<スイッチ> 左のスライドSW:GateTime切り替え:上側[0.1SEC]、下側[1SEC] 右のスライドSW:Prescale切り替え:上側[1/1]、下側[1/8] 赤色プッシュSW:ロック 黄色プッシュSW:アンロック

<LED> 黄色(小)LED:クロックのモニター用(0.1SEC点灯、0.1SEC消灯を繰り返す) 赤色LED:周波数が低い 橙色LED:周波数が高い 緑色LED:ロック状態

<その他> 電源を投入すると、緑色LED、橙色LED、赤色LEDの順に点灯します。 赤色プッシュSWと黄色プッシュSWを同時に押しながら電源を投入すると、 以降の処理で計測した周波数をUSARTへ出力します。