簡易周波数感応スイッチ

概要

ある特定の周波数に感応して□LEDを点灯させるスイッチを製作しました。 <用途>

- ラジオの時報(880Hz)を検出して、時計をリセットさせる。
- ホイッスルの音を検出して、試合中に何回ホイッスルが吹かれたかをカウントする。
- 特定の周波数でドアを開ける。(電子キー)
- 特定の周波数だけ通過させる。(バンドパスフィルタ)

<仕様>

- 感応周波数の範囲は、100Hz~1MHzとします。
- 設定スイッチ(SW_SET)で、感応周波数(設定値)を設定します□(EEPROMに書き込みます)
- 判定範囲は、±10Hz□±100Hz□±1KHz□±10KHzの4種を指定できます。
- 周波数感応(判定)モードとパソコン接続モードを指定できます。
  - 周波数感応(判定)モード
    → 設定値(感応周波数)と測定値を比較し、結果をLEDで表示します。
  - パソコン接続モード
    → 設定値(感応周波数)と測定値ををパソコンにデータ転送します□(9600bps)

動作原理(ハードウェア)

◎信号の増幅 入力信号の増幅には、トランジスタ(2SC1815)を自己バイアス方式で使用した簡単な回路としました。 用途(周波数帯域や増幅率など)に応じて変更してください。

◎判定結果の表示 判定結果は、LED_HおよびLED_Lを使用して表示します。

◎判定範囲の設定 設定値に対して、±何Hzの範囲であれば□OKなのかNGなのかを、スイッチ(SW_MODE_0□SW_MODE_1)で 指定します。

◎パソコンとの接続 必須ではありませんが、動作確認を行うために、設定値および測定値を、RS232Cでパソコンに送信します。 トランジスタ(2SC1815)を使用した、簡易なレベル変換(論理反転を含みます)を行っています。

動作原理(ソフトウェア)

◎メイン処理(main)

- 起動時にスイッチ(SW_SET)の状態によって、動作モードを決定します。
  - SW_SET=OFF状態であれば、周波数感応(判定)モードとします。
  - SW_SET=ON状態であれば、パソコン接続モードとします。
- EEPROMに書き込まれている、設定値(感応周波数)の値を読み込みます□(target)
- 周波数を測定するためのゲートタイム(100msec)を設定します□(CCPとTIMER1を使用□set_gate_time関数)
- 周波数を測定するためのカウンターを初期化します□(TIMER0を使用□init_counter関数)

- 周波数を測定(測定値)します(measurement関数)
- スイッチ(SW_SET)が押下されていれば、測定値を設定値とし、値をEEPROMに書き込みます。
- 周波数感応(判定)モードであれば、判定処理を呼び出します(judgment関数)
- パソコン接続モードであれば、設定値と測定値をUSART(RS232C)に出力します。

◎判定処理(judgment)

- 測定値、設定値、判定範囲を元に、判定を行います。
  - 測定値が設定値+判定範囲よりも大きい場合にはLED_Hを点灯させます。
  - 測定値が設定値-判定範囲よりも小さい場合にはLED_Lを点灯させます。
  - 測定値が設定値±判定範囲以内であればLED_HおよびLED_Lを点灯させます。

◎割り込み処理

- CCPとTIMER1を使用して100msecの割り込みを発生させます(set_gate_time関数)
- フラグ(flg)が“1“であれば、ゲートを開けます。フラグ(flg)を“2”にする。
- フラグ(flg)が“2“であれば、ゲートを閉じます。フラグ(flg)を“0”にする。

◎周波数測定処理

- フラグ(flg)を“1“にして、測定を開始します。
- フラグ(flg)が“0”になるまで、処理を繰り返します。
- 測定値を戻します。

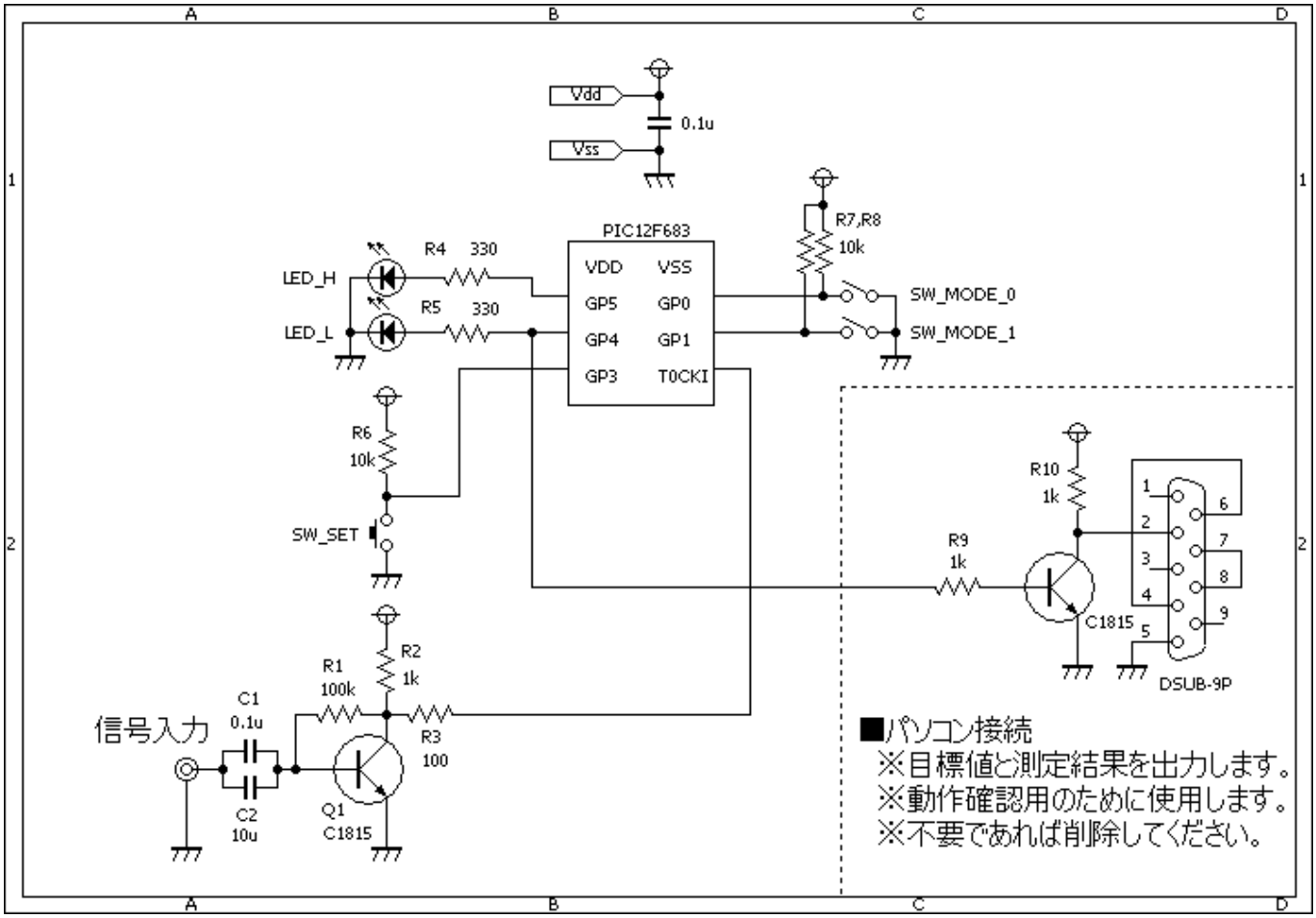
◎判定範囲取得処理

- スイッチ(SW_MODE_0、SW_MODE_1)の状態を元に、判定範囲を決定します。

|<400px>|

判定範囲（設定値に対して）	SW_MODE_1	SW_MODE_0
±10Hz	ON(0)	ON(0)
±100Hz	ON(0)	OFF(1)
±1KHz	OFF(1)	ON(0)
±10KHz	OFF(1)	OFF(1)

回路図



ソースコード

freq_switch.c

```
//*****************************************************************************
*
/*
      ＜簡易周波数感応スイッチ＞
    ・感応周波数□100Hz□1MH□
    ・判断範囲□±10Hz□±100Hz□±1KHz□±10KHz
*/
//*****************************************************************************
*
//SWITCH&LED
sbit    SW_MODE_0   at  GPIO.B0;
sbit    SW_MODE_1   at  GPIO.B1;
sbit    SW_SET      at  GPIO.B3;
sbit    LED_H       at  GPIO.B5;
sbit    LED_L       at  GPIO.B4;
//OTHER
#define BYTE        unsigned  short
#define WORD        unsigned  int
#define DWORD       unsigned  long
//*****************************************************************************
```

```
*
extern  void    main();
extern  void    set_gate_time();
extern  void    init_counter();
extern  void    interrupt();
extern  DWORD   measurement();
extern  void    Soft_Uart_Write_String(char *buf);
extern  DWORD   get_scope();
extern  void    judgment(DWORD freq, DWORD target, DWORD scope);
//*****************************************************************
*
//■■■メイン関数■■■
void    main()
{
        DWORD   freq;
        char    buf[11];
        short   mode;
        union {
                DWORD   _DWORD;
                BYTE    _BYTE[4];
        }       target;
        //
        OSCCON = 0b01110000;
        CMCON0 = 0b00000111;
        ANSEL  = 0b00000000;
        TRISIO = 0b00001111;
        //
        LED_H = 0;
        LED_L = 0;
        //動作モードを決定します。
        mode = 0;
        if (SW_SET == 0) {
                while (Button(&GPIO, 3, 1, 1) == 0) {
                        LED_H = 1;
                        Delay_ms(100);
                        LED_H = 0;
                        Delay_ms(100);
                }
                //
                mode = 1;
                //
                Soft_Uart_Init(&GPIO, 3, 4, 9600, 0);
        }
        //目標値をEEPROMより読み込みます。
        target._BYTE[0] = EEPROM_Read(0);
        target._BYTE[1] = EEPROM_Read(1);
        target._BYTE[2] = EEPROM_Read(2);
        target._BYTE[3] = EEPROM_Read(3);
        if (target._DWORD > 1000000) {
                target._DWORD = 1000000;
        }
```

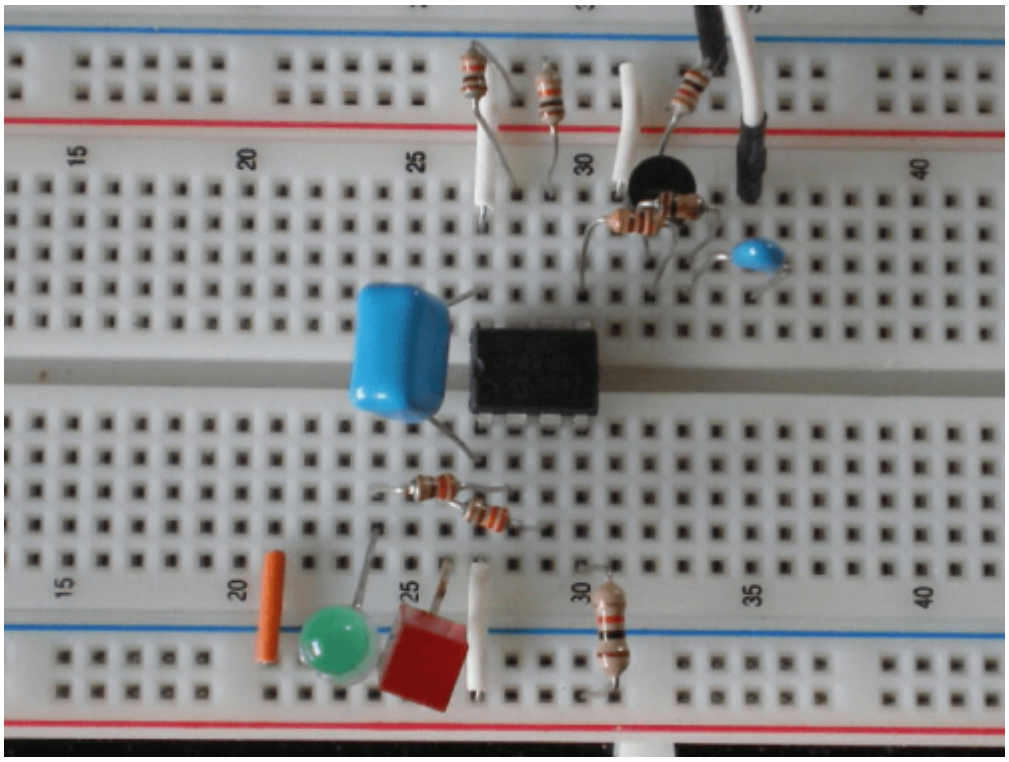
```
        //ゲートタイム(100msec)を設定します。
        set_gate_time();
        //カウンターを初期化します。
        init_counter();
        //
        while (1) {
                //周波数を測定します。
                freq = measurement();
                //目標値を設定します。
                if (SW_SET == 0) {
                        target._DWORD = freq;
                        //目標値をEEPROMに書き込みます。
                        EEPROM_Write(0, target._BYTE[0]);
                        EEPROM_Write(1, target._BYTE[1]);
                        EEPROM_Write(2, target._BYTE[2]);
                        EEPROM_Write(3, target._BYTE[3]);
                }
                //
                switch (mode) {
                case 0:  //測定結果を判定します。
                        judgment(freq, target._DWORD, get_scope());
                        break;
                case 1:  //目標値と測定結果をUSART□RS232C□出力します。
                        LongWordToStr(target._DWORD, buf);
                        Soft_Uart_Write_String(buf);
                        LongWordToStr(freq, buf);
                        Soft_Uart_Write_String(buf);
                        Soft_Uart_Write_String("\r\n");
                        break;
                }
        }
}
//*********************************************************************
*
//■■■判定関数■■■
void    judgment(DWORD freq, DWORD target, DWORD scope)
{
        if (freq > (target + scope)) {
                LED_H = 1;
                LED_L = 0;
                return;
        }
        if (freq < (target - scope)) {
                LED_H = 0;
                LED_L = 1;
                return;
        }
        LED_H = 1;
        LED_L = 1;
}
//*********************************************************************
```

```
*
//■■■ゲートタイム設定関数■■■
void    set_gate_time()
{
        // CCPの設定
        PIE1.CCP1IE = 1;
        PIR1.CCP1IF = 0;
        CCP1CON = 0b00001011;
        CCPR1L = 0xA8;          // 0.1sec...(1÷8000000)*4*8*25000
        CCPR1H = 0x61;
        // TIMER1の設定
        PIE1.TMR1IE = 0;
        PIR1.TMR1IF = 0;
        TMR1L = 0;
        TMR1H = 0;
        T1CON.T1CKPS0 = 1;
        T1CON.T1CKPS1 = 1;
        T1CON.TMR1ON = 1;
}
//*********************************************************************
*
//■■■カウンター初期化関数■■■
void    init_counter()
{
        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;
}
//*********************************************************************
*
//■■■割り込み関数■■■
short   flg = 0;
void    interrupt()
{
        if (PIR1.CCP1IF == 1) {
                PIR1.CCP1IF = 0;
                //
                switch (flg) {
                case 1:
                        TRISIO.B2 = 1;
                        flg = 2;
                        break;
                case 2:
                        TRISIO.B2 = 0;
                        GPIO.B2 = 0;
```

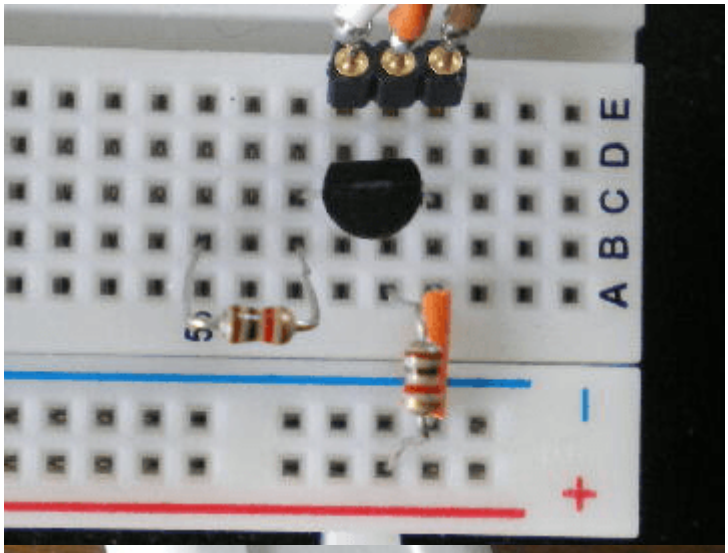
```
                        flg = 0;
                        break;
                }
        }
}
//*********************************************************************
*
//■■■周波数測定関数■■■
DWORD   measurement()
{
        DWORD   freq;
        //
        TRISIO.B2 = 0;
        GPIO.B2 = 0;
        freq = 0;
        TMR0 = 0;
        INTCON.T0IF = 0;
        flg = 1;
        //割り込みを許可します。
        INTCON.PEIE = 1;
        INTCON.GIE = 1;
        //
        while (flg != 0) {
                if (INTCON.T0IF == 1) {
                        INTCON.T0IF = 0;
                        freq++;
                }
        }
        if (INTCON.T0IF == 1) {
                INTCON.T0IF = 0;
                freq++;
        }
        //割り込みを停止します。
        INTCON.PEIE = 0;
        INTCON.GIE = 0;
        //
        return (((freq * 256) + TMR0) * 10);
}
//*********************************************************************
*
//■■■USART文字列送信関数■■■
void        Soft_Uart_Write_String(char *dat)
{
        while (*dat != 0x00) {
                Soft_Uart_Write(*dat);
                dat++;
        }
}
//*********************************************************************
*
//■■■判定範囲取得関数■■■
```

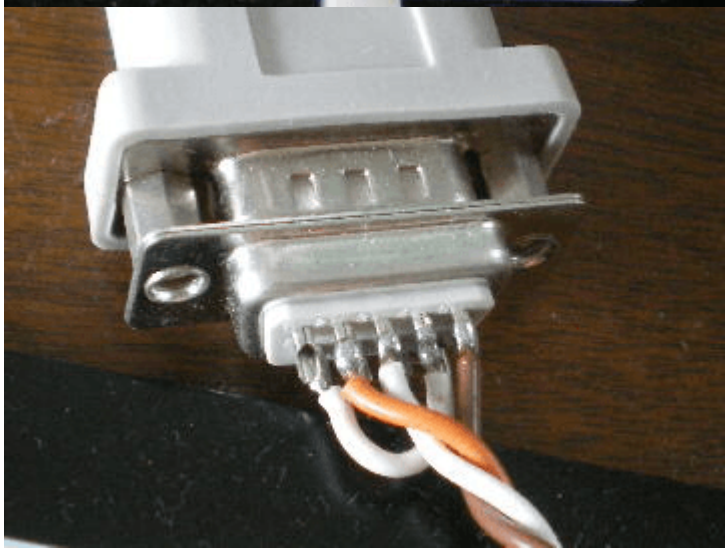
```
DWORD   get_scope()
{
        if ((SW_MODE_1 == 0) && (SW_MODE_0 == 0)) {
                return (10);        //±10Hz
        }
        if ((SW_MODE_1 == 0) && (SW_MODE_0 == 1)) {
                return (100);       //±100Hz
        }
        if ((SW_MODE_1 == 1) && (SW_MODE_0 == 0)) {
                return (1000);      //±1KHz
        }
        if ((SW_MODE_1 == 1) && (SW_MODE_0 == 1)) {
                return (10000);     //±10KHz
        }
}
//*****************************************************************************
*
```

動作確認

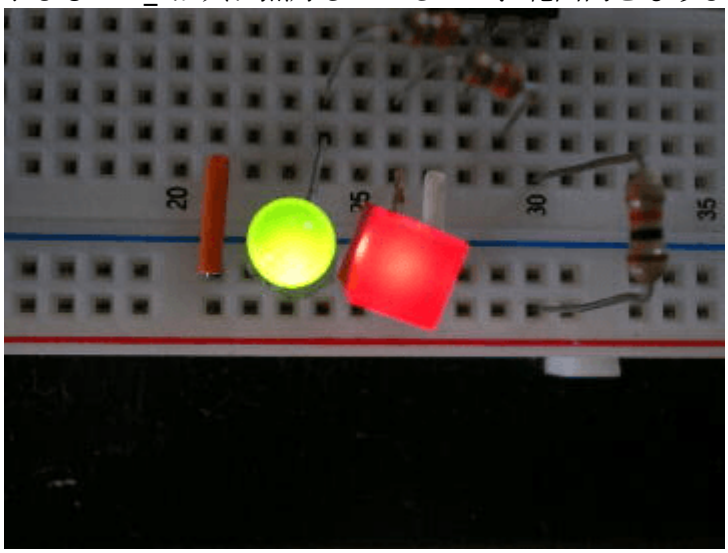


パソコン接続部です。

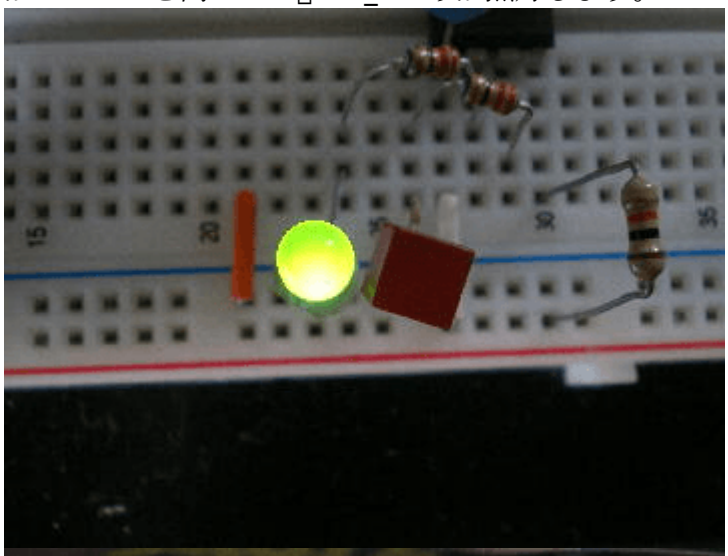


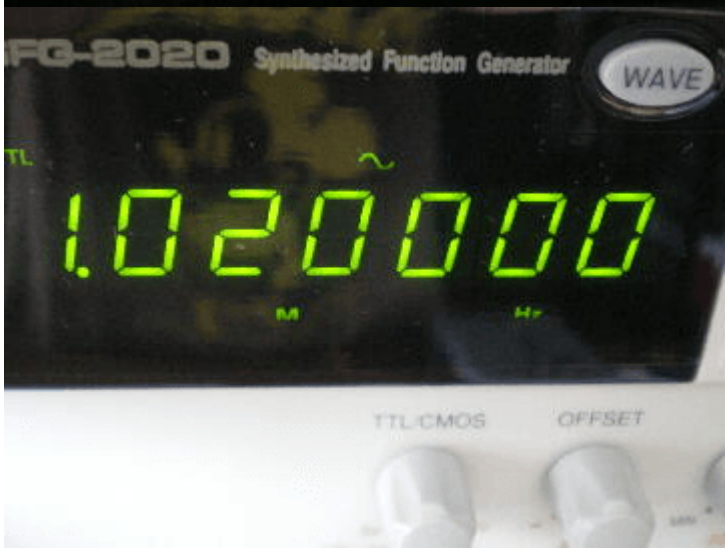


設定値を1MHz±10kHzに設定しました LED_HおよびLED_Lが共に点灯しているので、範囲内となります。

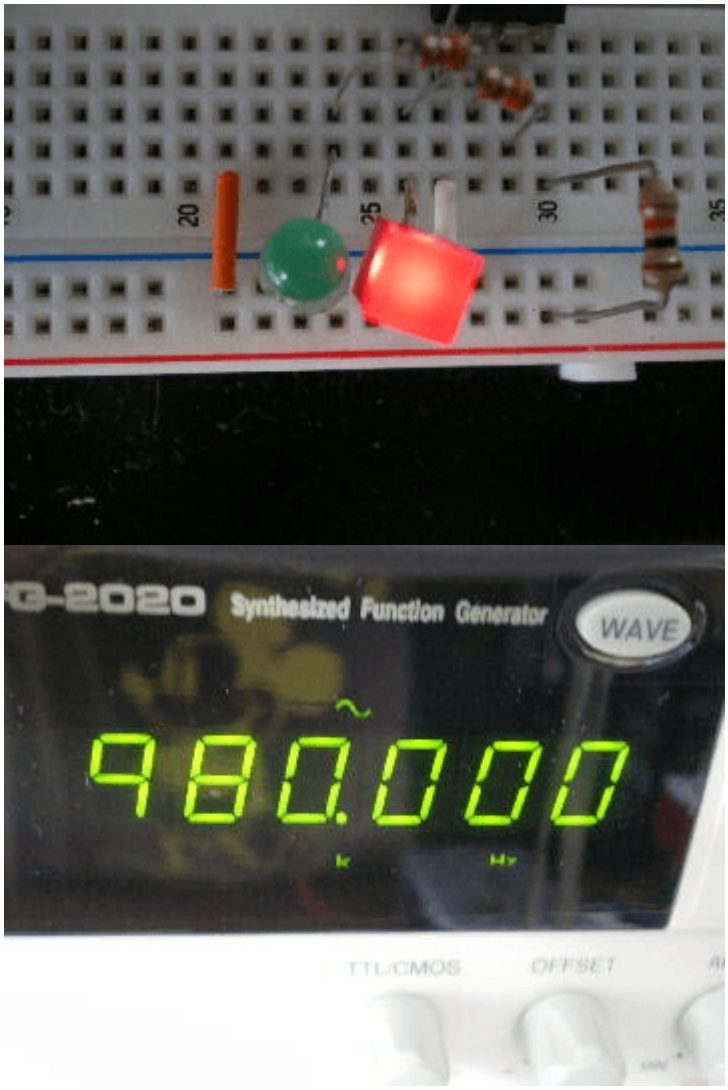


設定値(1MHz±10kHz)に対して、測定値が1.02MHzと高いのでLED_Hのみが点灯します。

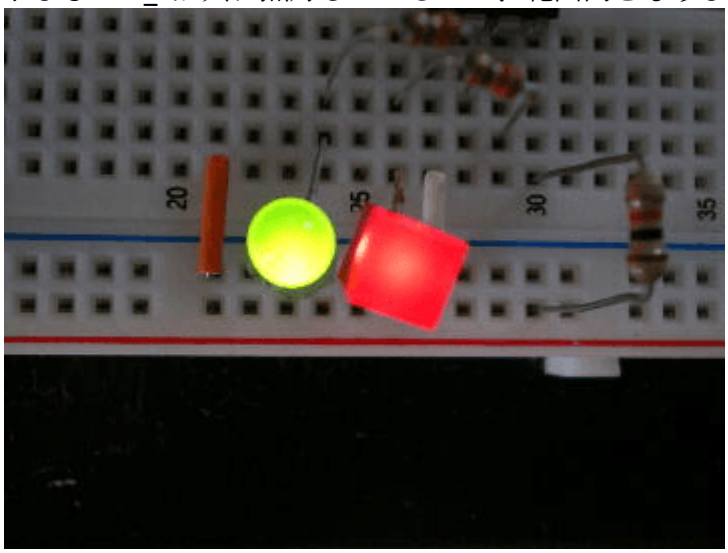




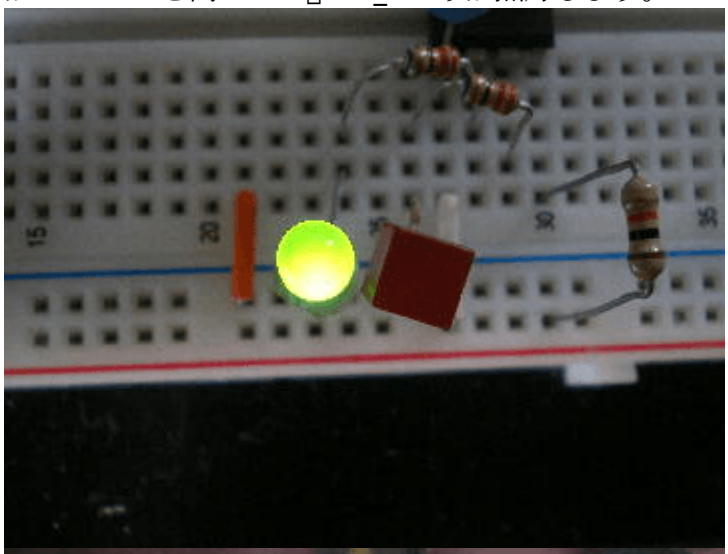
設定値(1MHz±10kHz)に対して、測定値が0.98MHzと低いのでLED_Lのみが点灯します。

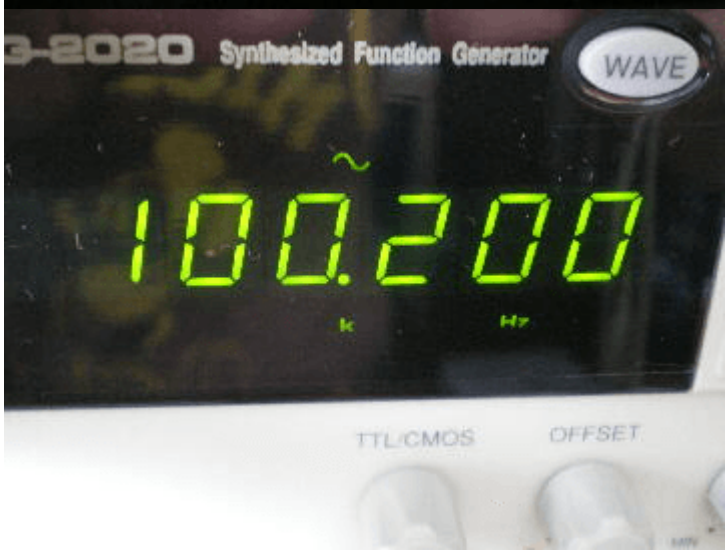


設定値を100kHz±1kHzに設定しました LED_HおよびLED_Lが共に点灯しているので、範囲内となります。

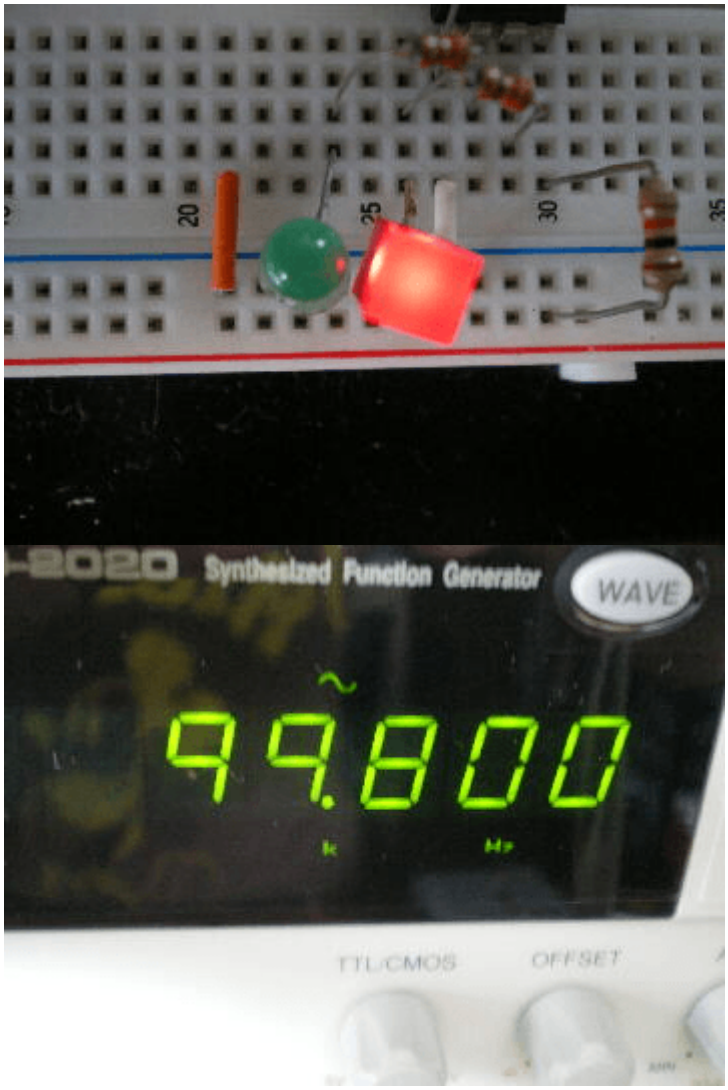


設定値(100kHz±100Hz)に対して、測定値が100.2kHzと高いのでLED_Hのみが点灯します。





設定値(100kHz±100Hz)に対して、測定値が99.8kHzと低いのでLED_Lのみが点灯します。



パソコン接続モードにして、設定値と測定値をUSART(RS232C)に出力します 左側=設定値(感応周波数)です。 右側=測定値です。