簡易周波数カウンタV2（校正機能搭載）

概要

以前にも簡易周波数カウンタを製作しましたが、今回は、部品点数を減らすと共に機能強化を図りました。

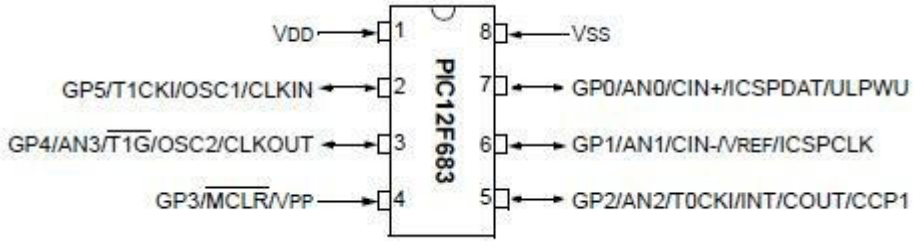
一般的には、周波数カウンタには、高精度のクロックモジュール(例えば16,000,000Hz等)を使いますが、今回はあえて使用せずにPIC内臓のクロック(8MHz)を使用しました。 しかし、それだけでは精度が得られませんので“精度校正機能“を実装しました。

<搭載機能>

- 精度校正機能
- プリスケール値切替機能(1/1、1/8)
- ゲートタイム切替機能(1秒、0.1秒)
- 表示単位切替機能(Hz、kHz)

動作原理

PIC内臓のクロックの精度を測定してみますと、数kHzの単位で誤差がありました。 測定方法は、クロックの指定で、内臓OSC(HFINTOSC)の8MHzとし、尚且つ、クロックOUTモードを指定します。 そうすることによりCLKOUT端子からクロックが1/4に分周されて出力されます。 今回使用したPICでは、約2.002MHz(つまり8.008MHz)ありました。 このままでは、クロックが8kHz高いので測定結果が入力信号の周波数よりも低くなってしまいます。



正確なゲートタイム(1秒、0.1秒)を得るためにPICが内蔵しているTIMER1モジュール(16ビット)を使っています。 そこで、この時のタイマーに設定する値を“±”微調整することにより、校正(Calibration)する方式を採用しました。 <ゲートタイムが1秒のときのタイマー設定値>

- 0.000004秒=(1秒÷8,000,000Hz)×4×8
- 250,000回=1秒÷0.000004秒
- 12,144回=65536-(250,000回-(65536×3回))
- 0x2F70回=12,144回
- ▼この0x2F70回をTIMER1に設定します。この値を微調整し校正します。

<ゲートタイムが0.1秒のときのタイマー設定値>

- 0.00004秒=(0.1秒÷8,000,000Hz)×4×8
- 25,000回=1秒÷0.00004秒
- 40,536回=65536-25,000回
- 0x9E58回=40,536回

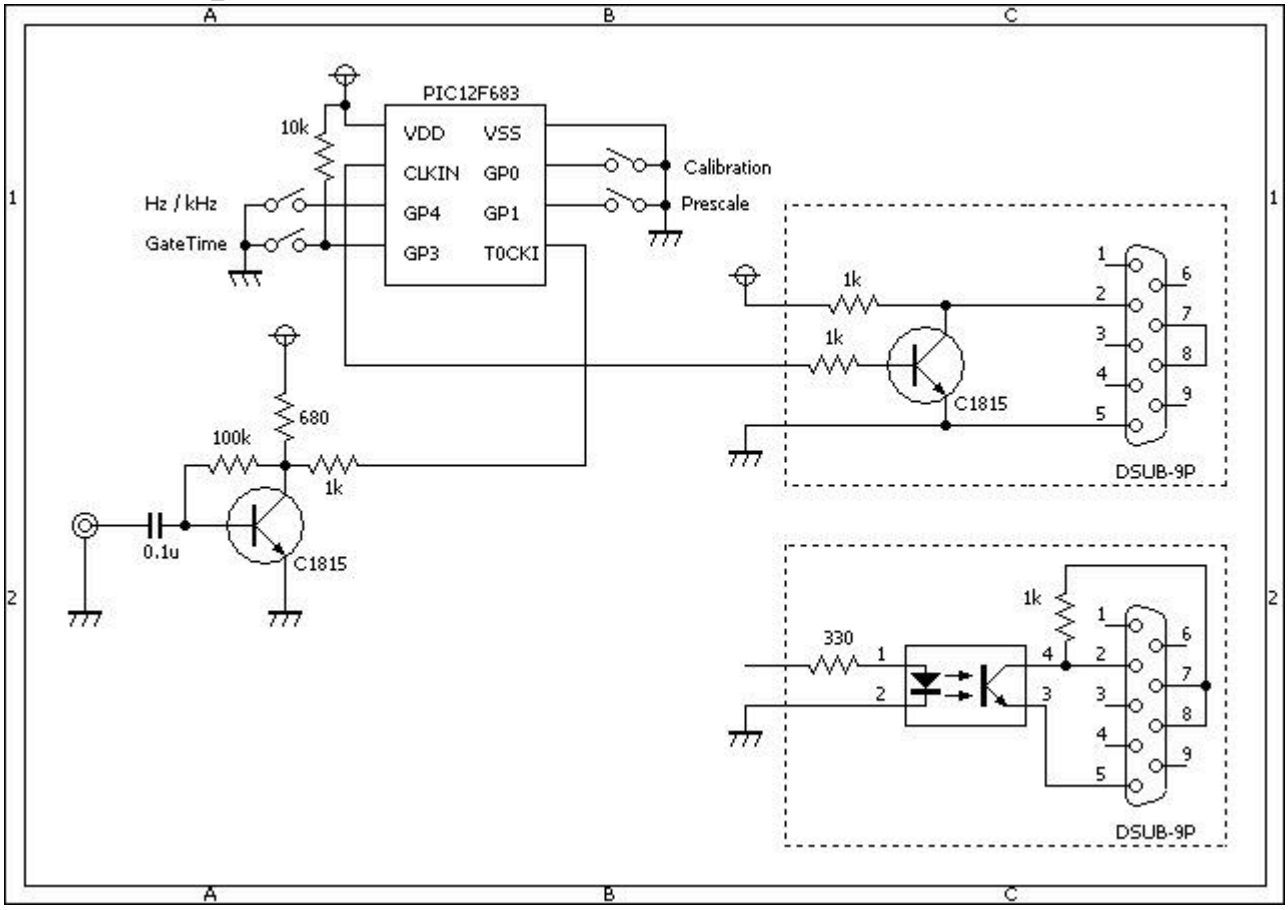
- ▼この0x9E58回をTIMER1に設定します。この値を微調整し校正します。

校正時(Calibrationスイッチを ON)には、高精度な1,000,000Hzの信号を入力します。 この時プログラムの処理では、測定結果が1,000,000Hzに近づくように TIMER1へ設定する値を“±“しながら自動的に微調整します。

尚、校正が完了した設定値(1秒用、0.1秒用)は、EEPROM(Electrically Erasable Programmable ROM)へ保存し、起動する毎に毎回設定する煩わしさを軽減しています。

回路図

とても簡単です RS232Cへのレベル変換は、点線内の回路のどちらを利用しても構いません。



ソースコード

FreqCounterS2.c

```
/*
  『簡易周波数カウンター（校正（Calibration）機能付き）』

＜構成概要＞
・内臓のクロック□□□□を使用する。
・計測結果は□□□□□□□□□□□□□□□□で送信する。
・校正機能により精度向上を可能とする。
・ゲートタイムは１秒、０．１秒とする。
```

```
・プリスケール値は１／１、１／８とする。

<SW割り当て>
・SW1: 1（校正しない）、0（校正する）
・SW2: 1（プリスケール値１／１）、0（プリスケール値１／８）
・SW3: 1（ゲートタイム１秒）、0（ゲートタイム０．１秒）
・SW4: 1（Hz表示）、0（kHz表示）

<デフォルトの校正値>
・CAL100msec
      25000=0.1/((1/8000000)*4*8)
      0x9E58=65536-25000
・CAL1sec
      250000=1/((1/8000000)*4*8)
      0x2F70=65536-(250000-(65536*3))

*/

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

#define       SW1         GPIO.F0
#define       SW2         GPIO.F1
#define       SW3         GPIO.F3
#define       SW4         GPIO.F4

#define       GATETIME_100MSEC    10
#define       GATETIME_1SEC             1

static unsigned    int    CAL1sec;
static unsigned    int    CAL100msec;

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

static short    interruptCnt;

void   interrupt()
{
    if (PIR1.TMR1IF == 1) {
        PIR1.TMR1IF = 0;
        //
        interruptCnt--;
        if (interruptCnt == 0) {
            TRISIO.F2 = 0;          //ゲートを閉める。
            GPIO.F2 = 0;
            T1CON.TMR1ON = 0;     // TIMER1を停止する。
        }
    }
}
```

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

unsigned   long    FreqMeasurement(unsigned short gateTime)
{
    unsigned    long    freq;
    //ゲートを閉める。
    TRISIO.F2 = 0;
    GPIO.F2 = 0;
    // TIMER0の設定
    INTCON.T0IF = 0;
    TMR0 = 0;
    // TIMER1の設定
    PIR1.TMR1IF = 0;
    switch (gateTime) {
    case GATETIME_100MSEC:
        interruptCnt = 1;
        TMR1L = CAL100msec & 0xFF;
        TMR1H = (CAL100msec >> 8) & 0xFF;
        break;
    case GATETIME_1SEC:
        interruptCnt = 4;
        TMR1L = CAL1sec & 0xFF;
        TMR1H = (CAL1sec >> 8) & 0xFF;
        break;
    }
    //
    freq = 0;
    //割り込みを許可する。
    INTCON.PEIE = 1;
    INTCON.GIE = 1;
    // □□□□□□を開始する。
    T1CON.TMR1ON = 1;
    //遅延する。
    Delay_Cyc(2);
    asm    nop
    asm    nop
    asm    nop
    asm    nop
    asm    nop
    asm    nop
    asm    nop
    asm    nop
    //
    TRISIO.F2 = 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 = (freq * 256) + TMR0;
    return (freq);
}

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

void   Soft_Uart_Write_String(char *buf)
{
    short     len, i;
    len = strlen(buf);
    for (i = 0; i < len; i++) {
        INTCON.GIE = 0;
        Soft_Uart_Write(buf[i]);
        INTCON.GIE = 1;
    }
}

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

void main()
{
    static    unsigned    long    freq, temp;    // 0...4294967295
    static    unsigned    char    buf[12], prescaler, gateTime;
    //
    OSCCON = 0b01110000;    //クロックは8Mhz
    ANSEL  = 0b00000000;    //今回は使用しない。
    CMCON0 = 0b00000111;    //今回は使用しない。
    TRISIO = 0b00011111;
    OPTION_REG.F7 = 0;        // PORTをプルアップ設定する。
    WPU.F0 = 1;
    WPU.F1 = 1;
    WPU.F4 = 1;
    // 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;
```

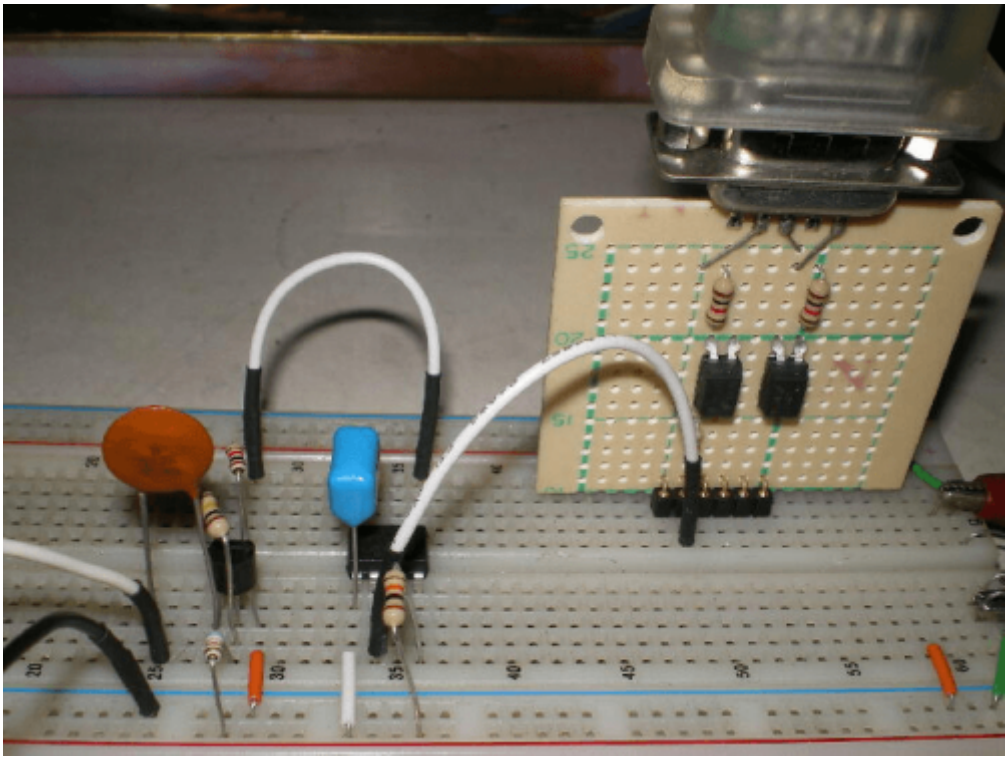
```
    T1CON.T1CKPS0 = 1;
    T1CON.T1CKPS1 = 1;
    T1CON.TMR1ON = 0;
    //  保存されている校正値の取り込み
    CAL100msec = Eeprom_Read(1);
    CAL100msec = CAL100msec << 8;
    CAL100msec = CAL100msec | Eeprom_Read(0);
    if ((CAL100msec < (0x9E58 - 1000)) || (CAL100msec > (0x9E58 +
1000)))
        CAL100msec = 0x9E58;
    //
    CAL1sec = Eeprom_Read(3);
    CAL1sec = CAL1sec << 8;
    CAL1sec = CAL1sec | Eeprom_Read(2);
    if ((CAL1sec < (0x2F70 - 1000)) || (CAL1sec > (0x2F70 + 1000)))
        CAL1sec = 0x2F70;
    //  変数の初期化
    prescaler = 1;
    gateTime = GATETIME_1SEC;
    //
    Soft_Uart_Init(GPIO, 3, 5, 9600, 0);
    Soft_Uart_Write_String("Frequency Counter (S) V2\r\n");
    //
    while (1) {
        //測定
        freq = FreqMeasurement(gateTime);
        //補正
        freq *= prescaler * gateTime;
        //プリスケーラの切り替え
        if (SW2 == 1) {
            OPTION_REG.PSA  = 1;
            OPTION_REG.PS1  = 0;
            prescaler = 1;
        } else {
            OPTION_REG.PSA  = 0;
            OPTION_REG.PS1  = 1;
            prescaler = 8;
        }
        //ゲートタイムの切り替え
        if (SW3 == 1) {
            gateTime = GATETIME_1SEC;
        } else {
            gateTime = GATETIME_100MSEC;
        }
        //表示レンジの切り替え&表示
        if (SW4 == 1) {
            LongToStr(freq, buf);
            Soft_Uart_Write_String(buf);
            Soft_Uart_Write_String("Hz");
        } else {
            temp = freq / 1000;
```

```
            if ((freq - (temp * 1000)) > 500) {
                temp++;
            }
            LongToStr(temp, buf);
            Soft_Uart_Write_String(buf);
            Soft_Uart_Write_String("kHz");
        }
        // 校正
        if (SW1 == 0) {
            if ((freq > 1000000) && (gateTime == GATETIME_100MSEC))
                CAL100msec++;
            if ((freq < 1000000) && (gateTime == GATETIME_100MSEC))
                CAL100msec--;
            if ((freq > 1000000) && (gateTime == GATETIME_1SEC))
                CAL1sec++;
            if ((freq < 1000000) && (gateTime == GATETIME_1SEC))
                CAL1sec--;
            // 校正値の保存
            Eeprom_Write(0, (CAL100msec & 0xFF));
            Delay_ms(20);
            Eeprom_Write(1, ((CAL100msec >> 8) & 0xFF));
            Delay_ms(20);
            Eeprom_Write(2, (CAL1sec & 0xFF));
            Delay_ms(20);
            Eeprom_Write(3, ((CAL1sec >> 8) & 0xFF));
            Delay_ms(20);
            // 校正値の表示
            if (gateTime == GATETIME_100MSEC)
                LongToStr(CAL100msec, buf);
            else
                LongToStr(CAL1sec, buf);
            Soft_Uart_Write_String(buf);
            Soft_Uart_Write_String("\r\n");
        } else {
            Soft_Uart_Write_String("\r\n");
        }
    }
}

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

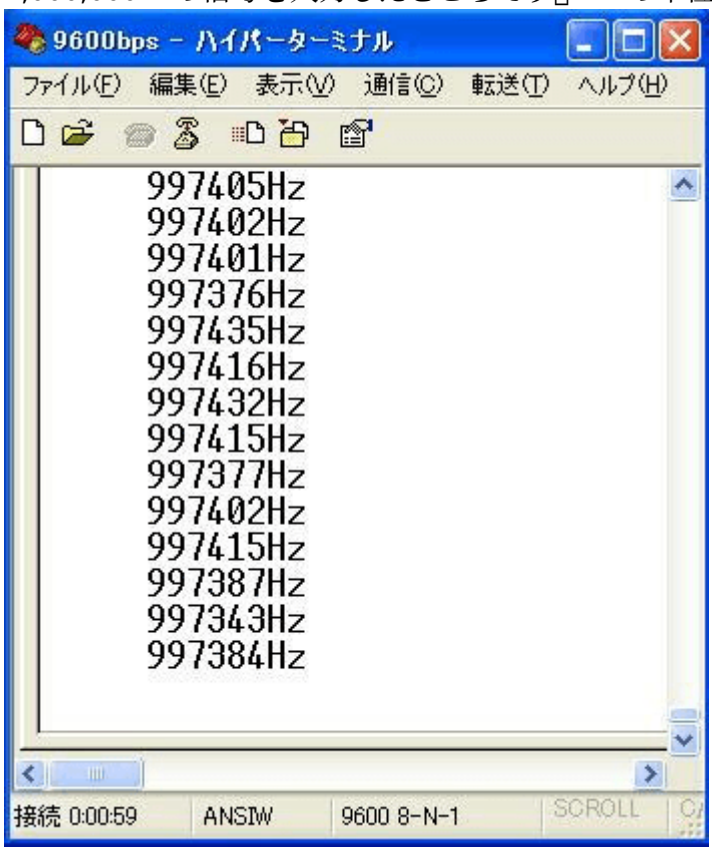
動作確認

いつものブレッドボードで確認しました。RS232Cへのレベル変換には、以前に製作しましたRS232Cレベル変換ユニットを使用しました。 このように頻繁に使う回路は、モジュール化しておくと何かと便

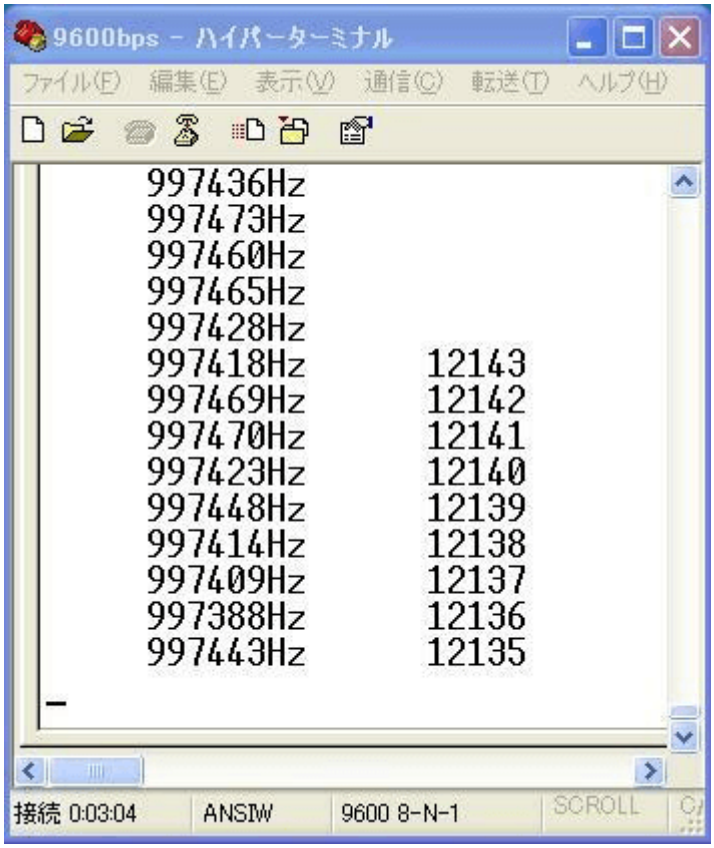


利です。

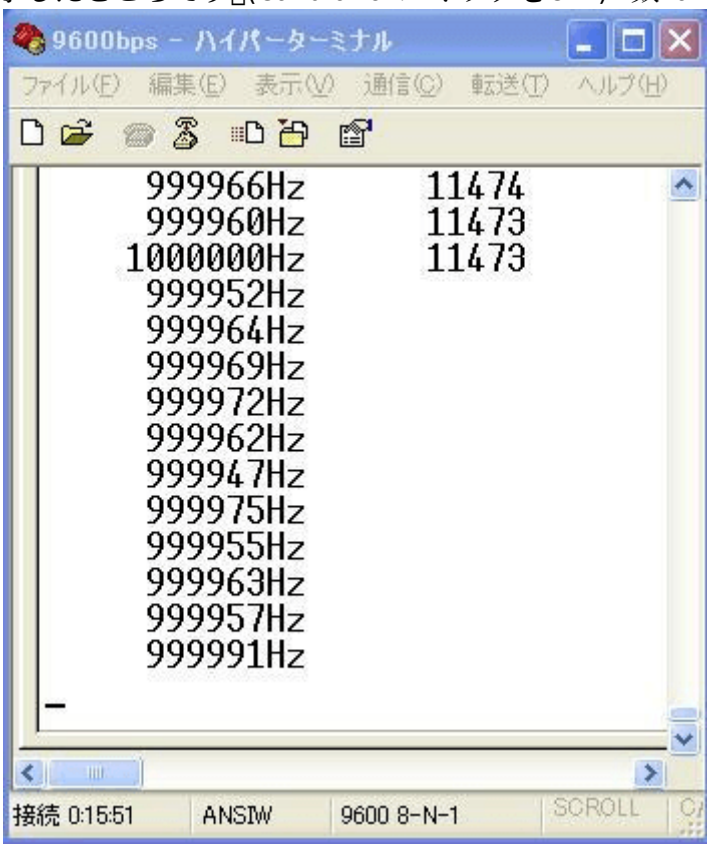
1,000,000Hzの信号を入力したところです kHzの単位で誤差がありますね。(校正前)



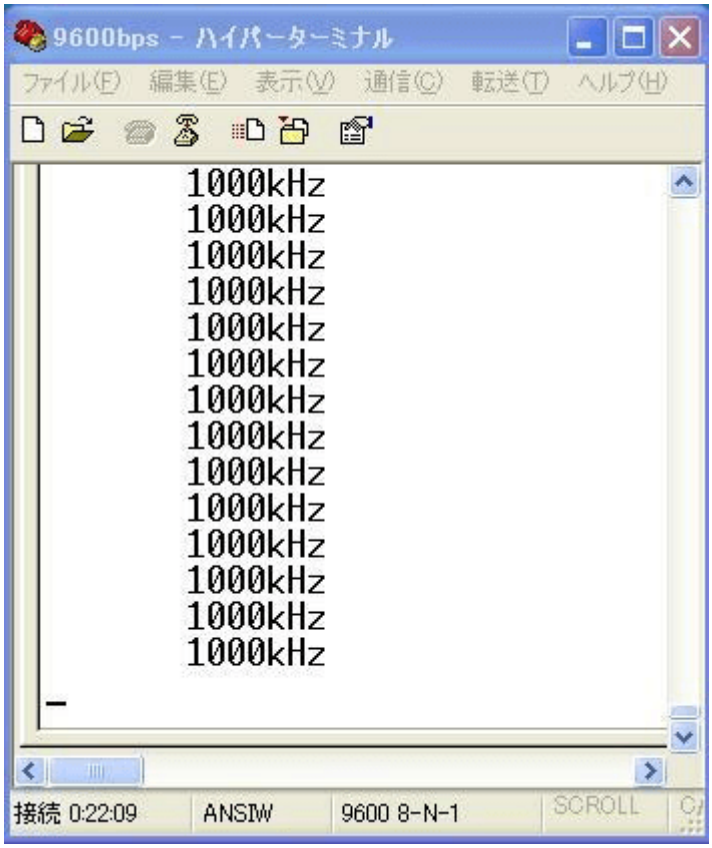
校正を開始したところです(CalibrationスイッチをON) 右側の数値は校正用の数値です。この場合(測定値が1MHzよりも低い場合)ですと、測定値が1MHzになる まで校正用の数値が下がっていきます。



校正が完了したところです(CalibrationスイッチをOFF) “数10”Hz位の誤差まで精度を上げることが出来ました。



kHz表示をしたところです kHz未満は四捨五入しています。



如何ですか? kHz以上での精度で構わない場合には、本方式で十分かと思われます。