

温度計(LM35DZ直結)(PIC16F876)

概要

LM35DZを使用した温度計を良く見かけます。そこでは大抵LM35DZの出力をオペアンプを利用して、10倍に増幅してから処理をしています。<10倍に増幅するメリット>

- 温度係数が、10.0mV/°Cと小さいので10倍にすると温度係数を100.0mV/°Cとすることが出来る。
- そうすることにより、小数点以下の単位まで温度を測定することが出来る。
- つまりPICのA/Dの精度では、最小単位が約5mVなので(A/DのVref+が5Vの場合 $5\text{mV} \approx 5\text{V} \div 1023$)
 - 増幅をしないと、0.5 が限度となるが、
 - 10倍に増幅すると、0.05 までの精度で測定が可能となる。

<10倍に増幅するデメリット>

- 部品点数(オペアンプ+抵抗)が多くなる。
- 増幅率を10倍に精度よく調整するのは結構難儀。
- 温度測定範囲が0~50°C($5\text{V} \approx 50 \times 100\text{mV}$)と半減してしまう。

0.05 までの精度を求めると増幅も致し方無いのですが、増幅をしなくても、もう少し精度(せめて倍)を上げることはできないものなのでしょうか?

<LM35DZの規格>

- 摂氏()に比例(直読可能)した電圧出力(例:20 のとき、出力200mV)
- 測定温度範囲:0~100
- 精度: ± 1
- 温度係数:10.0mV/°C
- 電源電圧:DC4V~20V 低消費電流:60μA
- 低出力インピーダンス:0.1Ω

動作原理

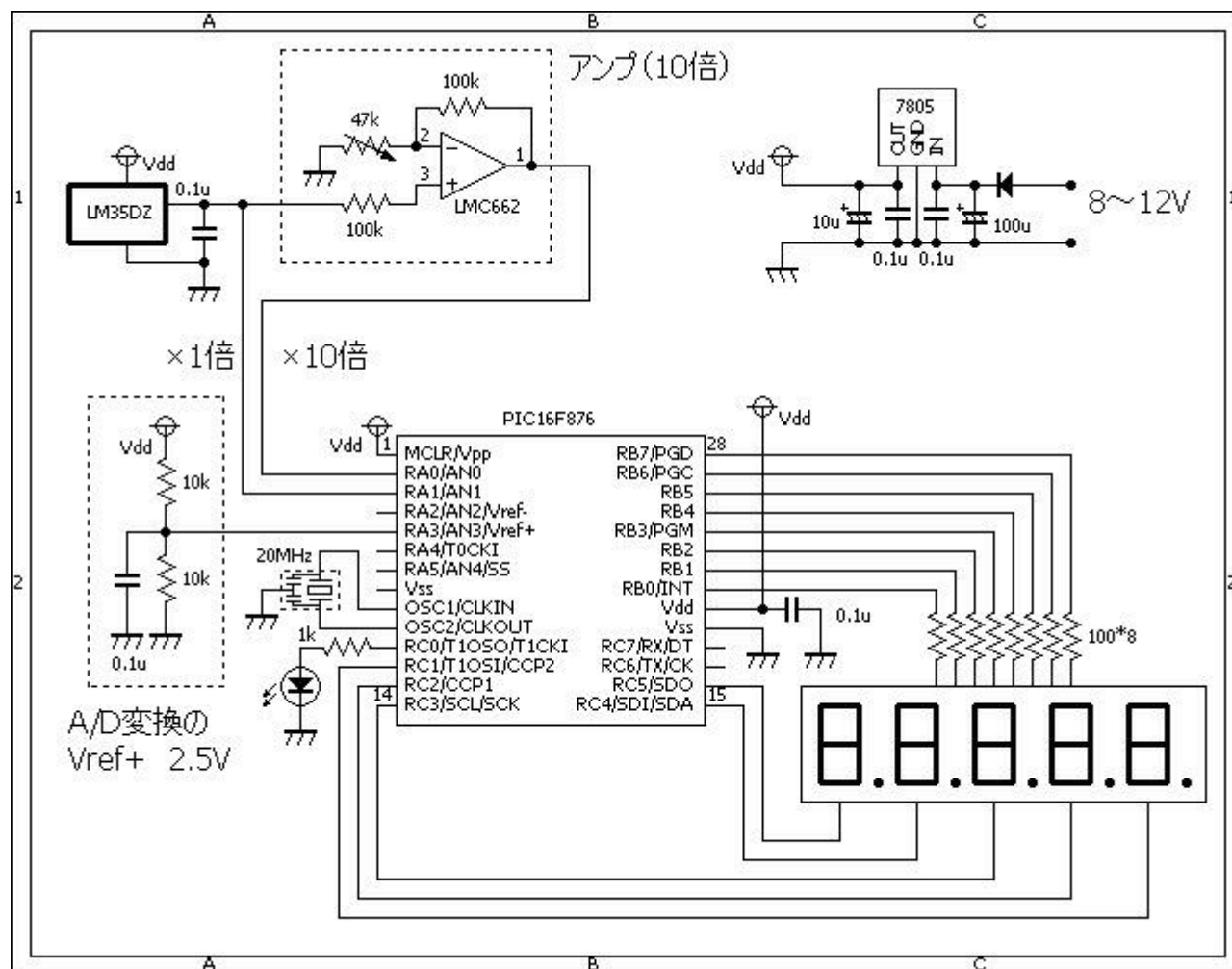
A/D変換のVref+が5Vの場合には、最小分解能が 5mV($5\text{V} \div 1023$)となりますがPICの規格ではVref+を2.5Vまで下げることが出来るので、最小分解能を、 $\approx 2.5\text{mV}$ とすることができ、単純に倍の精度を得ることが出来ます。

つまり、温度で換算すると、0.5 単位を0.25 単位まで、精度を高めることが出来ることになります。また、測定温度範囲も、0-100 までと半減することなくカバーできます。何よりも最大のメリットは、ハードウェア的に調整する箇所が無いということだと思います。回路的にはLM35DZの直結の値と10倍増幅の値の両方を測定し、温度換算した値を5桁の7セグメントLEDにダイナミック点灯方式で表示させました。

<四捨五入の仕方> LM35DZの出力が125mV(12.5°C)とした場合、これを小数点以下の温度を四捨五入すると13 になります。これをプログラムでは次のようにして実現します。

```
int i;
```

回路図



ソースコード

```
//*****
*
#define SEG1 PORTC.F5
#define SEG2 PORTC.F4
#define SEG3 PORTC.F3
#define SEG4 PORTC.F2
#define SEG5 PORTC.F1

//*****
```

```
*

static unsigned char segTbl[13] = {
    0b10111110, // 0
    0b10100000, // 1
    0b11011010, // 2
    0b11110010, // 3
    0b11100100, // 4
    0b01110110, // 5
    0b01111110, // 6
    0b10100110, // 7
    0b11111110, // 8
    0b11110110, // 9
    0b01000000, // -
    0b11111111, // all-on
    0b00000000, // all-off
};

//*****
*

unsigned char ChangeSegData(unsigned char c)
{
    switch (c) {
        case '0':
            c = (segTbl[0]);
            break;
        case '1':
            c = (segTbl[1]);
            break;
        case '2':
            c = (segTbl[2]);
            break;
        case '3':
            c = (segTbl[3]);
            break;
        case '4':
            c = (segTbl[4]);
            break;
        case '5':
            c = (segTbl[5]);
            break;
        case '6':
            c = (segTbl[6]);
            break;
        case '7':
            c = (segTbl[7]);
            break;
        case '8':
            c = (segTbl[8]);
            break;
    }
}
```

```
    case '9':
        c = (segTbl[9]);
        break;
    case '-':
        c = (segTbl[10]);
        break;
    case '*':
        c = (segTbl[11]);
        break;
    case ' ':
        c = (segTbl[12]);
        break;
    default:
        c = (segTbl[10]);
        break;
}
return(c);
}

//*****
*

static unsigned char dat1, dat2, dat3, dat4, dat5, datCnt;

void interrupt(){
    if (INTCON.T0IF == 1) {
        INTCON.T0IF = 0;
        //
        SEG1 = 1;
        SEG2 = 1;
        SEG3 = 1;
        SEG4 = 1;
        SEG5 = 1;
        //
        switch (datCnt) {
            case 0:
                PORTB = ChangeSegData(dat1);
                SEG1 = 0;
                datCnt++;
                break;
            case 1:
                PORTB = ChangeSegData(dat2);
                SEG2 = 0;
                datCnt++;
                break;
            case 2:
                PORTB = ChangeSegData(dat3);
                SEG3 = 0;
                datCnt++;
                break;
```

```
        case 3:
            PORTB = ChangeSegData(dat4);
            SEG4 = 0;
            datCnt++;
            break;
        case 4:
            PORTB = ChangeSegData(dat5);
            SEG5 = 0;
            datCnt = 0;
            break;
        default:
            break;
    }
}
if (PIR1.TMR1IF == 1) {
    PIR1.TMR1IF = 0;
    PORTC.F0 = ~PORTC.F0;
}
}

//*****
*

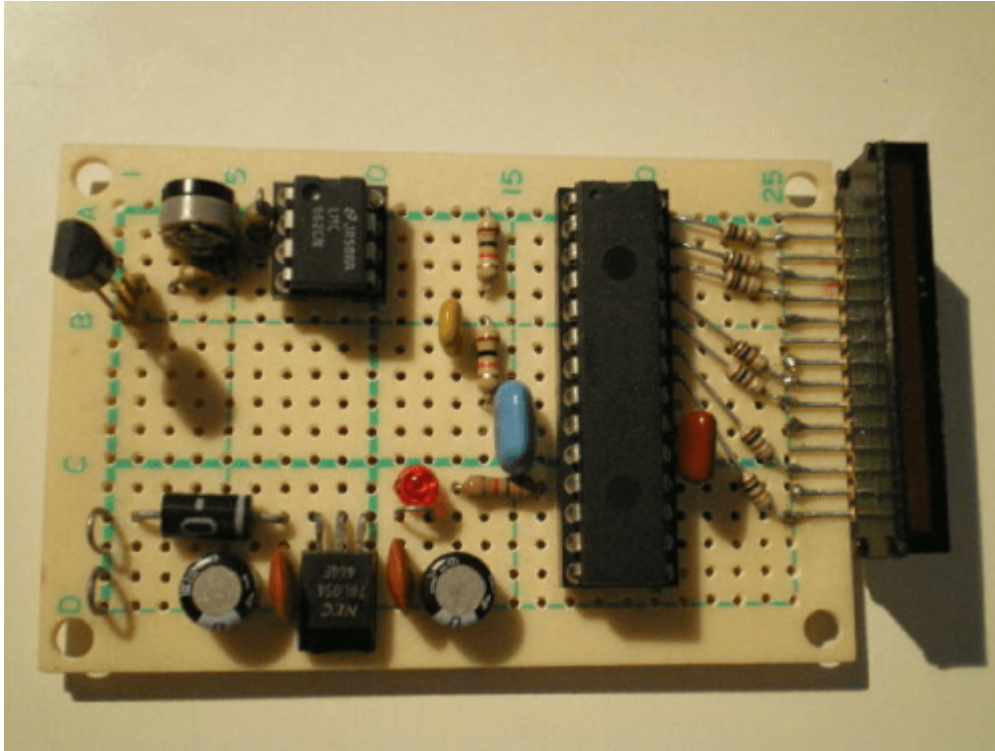
void main()
{
    unsigned    int        ad0, ad1, tmp;
    unsigned    char       buf[10];
    //
    TRISA = 0b00001011;
    TRISB = 0b00000000;
    TRISC = 0b00000000;
    OPTION_REG = 0b00000000;
    // ad0を使用する。
    ADCON1.PCFG3 = 0;
    ADCON1.PCFG2 = 1;
    ADCON1.PCFG1 = 0;
    ADCON1.PCFG0 = 1;
    // CCP1 モジュールは使用しない。
    CCP1CON.CCP1M3 = 0;
    CCP1CON.CCP1M2 = 0;
    CCP1CON.CCP1M1 = 0;
    CCP1CON.CCP1M0 = 0;
    // CCP2 モジュールは使用しない。
    CCP2CON.CCP1M3 = 0;
    CCP2CON.CCP1M2 = 0;
    CCP2CON.CCP1M1 = 0;
    CCP2CON.CCP1M0 = 0;
    // タイマー 0 を設定する。
    INTCON.T0IE = 1;
    INTCON.T0IF = 0;
    OPTION_REG.T0CS = 0;
```

```
OPTION_REG.PSA = 0;
OPTION_REG.PS2 = 1;
OPTION_REG.PS1 = 0;
OPTION_REG.PS0 = 1;
// タイマー 1 を設定する。
PIE1.TMR1IE = 1;
PIR1.TMR1IF = 0;
T1CON.T1CKPS0 = 1;
T1CON.T1CKPS1 = 1;
T1CON.TMR1ON = 1;
//
PORTC = 0xFF;
datCnt = 0;
dat1 = ' ';
dat2 = ' ';
dat3 = ' ';
dat4 = ' ';
dat5 = ' ';
//
INTCON.PEIE = 1;    // これ以降の処理で割り込みを許可する。
INTCON.GIE = 1;    // これ以降の処理で割り込みを許可する。
//
while (1) {
    ad0 = Adc_Read(0);    // 温度センサーの出力をオペアンプで10倍した値。
    ad1 = Adc_Read(1);    // 温度センサーの出力をそのままの値。
    // 小数点以下の温度を四捨五入する。1度=100mV
    ad0 = (int)((double)ad0 * 2.48);
    if ((ad0 - ((ad0 / 100) * 100)) >= 50)
        ad0 = (ad0 / 100) + 1;
    else
        ad0 = (ad0 / 100);
    IntToStr(ad0, buf);
    dat5 = buf[4];
    dat4 = buf[5];
    // 小数点以下の温度を四捨五入する。1度=10mV
    ad1 = (int)((double)ad1 * 2.48);
    if ((ad1 - ((ad1 / 10) * 10)) >= 5)
        ad1 = (ad1 / 10) + 1;
    else
        ad1 = (ad1 / 10);
    IntToStr(ad1, buf);
    dat2 = buf[4];
    dat1 = buf[5];
    //
    dat3 = '-';
    Delay_ms(500);
}
}
```

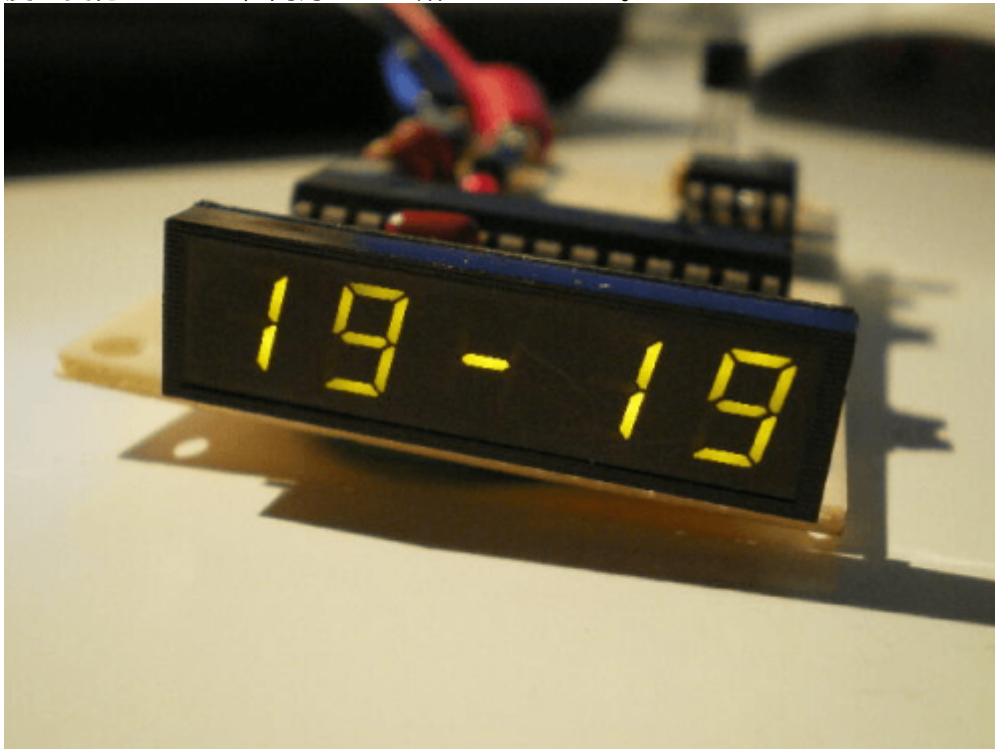
//*****

*

動作確認



向かって、左が、オペアンプを経由したときの値、右が直結のときの値です。指でLM35DZに触れて温度を変化させても、両方の差は殆どありません。



如何ですか? 温度の小数点以下を四捨五入して表示する程度の精度を望むなら、部品点数も少なくともシンプルで宜しいかと。。。その時には表示桁数も2-3桁で十分ですね。

Last update:
2025/10/17 14:27

elechobby:picdic:otherpic:165 <http://deepsky.jp/wiki/doku.php?id=elechobby:picdic:otherpic:165&rev=1588236635>

From:

<http://deepsky.jp/wiki/> - うごくといいな

Permanent link:

<http://deepsky.jp/wiki/doku.php?id=elechobby:picdic:otherpic:165&rev=1588236635>

Last update: **2025/10/17 14:27**

