

簡易ワンショット・ロガV2(PIC18F2620)

概要

前回製作した、「簡易ワンショット・ロガー」に機能追加を図りました。

<機能追加項目> トリガレベルを可変可能とします。

- 前回(固定方式)THRESHOLD_LOW(500mVを上回る)THRESHOLD_HIGH (4500mVを下回る)
- 今回(可変方式)THRESHOLD_LOW(設定電圧を上回る)THRESHOLD_HIGH (設定電圧を下回る)

設定はボリューム(VR)によって行います。LCDに各種値を表示させます。

- THRESHOLD_LOWとTHRESHOLD_HIGHの電圧値を表示します。
- モード番号を表示します。
- 測定の状態(開始と終了)を表示します。

動作原理

前回製作した、「簡易ワンショット・ロガー」に、LCD(16文字×2行)と、トリガレベル設定用のボリューム(VR)を 2個追加しました。

<THRESHOLD_LOW>

- VR2でTHRESHOLD_LOWの電圧を設定します。
- THRESHOLD_LOWを、上回るとトリガがかかります。

<THRESHOLD_HIGH>

- VR1でTHRESHOLD_HIGHの電圧を設定します。
- THRESHOLD_HIGHを、下回るとトリガがかかります。

<LCDへの表示>

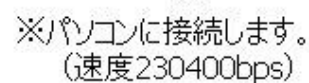
- スタートスイッチが押下されるまでTHRESHOLD_HIGHの値、THRESHOLD_LOWの値、モード番号をLCDに表示します。

<測定>

- 測定開始時に、LCDに “ start!” を表示します。
- 各モードに応じた測定をします。
- 測定結果をパソコンに、RS232C経由(通信速度230400bps)でデータ転送します。
- 測定終了時に、LCDに “ stop!” を表示します。

回路図

前回製作した、「簡易ワンショット・ロガー」にVR1VR2VR3LCDを追加しました。



・モード(03) スイッチ押下後直ちに記録を開始する。
 190個のデータをエンドレスで記録する。
 入力電圧が500mVを上回ると、更に、1710個のデータを記録し、停止する。

・モード(04) スイッチ押下後直ちに入力電圧をチェックし、入力電圧が4500mVを下回ると、
 1900個のデータを記録し、停止する。

・モード(05) スイッチ押下後直ちに記録を開始する。
 190個のデータをエンドレスで記録する。
 入力電圧が4500mVを下回ると、更に、1710個のデータを記録し、停止する。

・モード(11) スイッチ押下後直ちに記録を開始する。
 1900個のデータをエンドレスで記録する。
 スイッチが再度押下されると記録を停止する。

サンプルング速度 (約1msec□□データ数 (1900個)、記録時間□1.9sec□

・モード(06) スイッチ押下後直ちに記録を開始する。1900個のデータを記録すると停止する。

・モード(07) スイッチ押下後直ちに入力電圧をチェックし、入力電圧が500mVを上回ると、
 1900個のデータを記録し、停止する。

・モード(08) スイッチ押下後直ちに記録を開始する。
 190個のデータをエンドレスで記録する。
 入力電圧が500mVを上回ると、更に、1710個のデータを記録し、停止する。

・モード(09) スイッチ押下後直ちに入力電圧をチェックし、入力電圧が4500mVを下回ると、
 1900個のデータを記録し、停止する。

・モード(10) スイッチ押下後直ちに記録を開始する。
 190個のデータをエンドレスで記録する。
 入力電圧が4500mVを下回ると、更に、1710個のデータを記録し、停止する。

・モード(12) スイッチ押下後直ちに記録を開始する。
 1900個のデータをエンドレスで記録する。
 スイッチが再度押下されると記録を停止する。

*/

//*****
 *

```

#define LED PORTC.F0
#define SW_START PORTC.F1
#define SW_MODE_1 PORTC.F2
#define SW_MODE_2 PORTC.F3
#define SW_MODE_3 PORTC.F4
#define SW_MODE_4 PORTC.F5

```

```

#define BUFF_SIZE 1800
#define BEFOR_SIZE 180
#define AFTER_SIZE 1620

```

/*

```

#define THRESHOLD_LOW ((500.0 / 4.8828125) / 2.0)
#define THRESHOLD_HIGH ((4500.0 / 4.8828125) / 2.0)

```

```
*/  
  
//*****  
*  
  
static      int      ad[BUFF_SIZE], befor_cnt, after_cnt,  
normal_cnt, tmp;  
static      char      buf[16];  
static      int      THRESHOLD_LOW, THRESHOLD_HIGH;  
  
//*****  
*  
  
void      data_transfer_1()  
{  
    static      int      cnt;  
    //  
    UART1_Write_Text("\r\n$START\r\n");  
    for (cnt = 0; cnt < BUFF_SIZE; cnt++) {  
        WordToStr(((double)ad[cnt]) * 48.828125 * 2.0, buf);  
        buf[6] = 0x00;  
        buf[5] = buf[4];  
        buf[4] = '.';  
        UART1_Write_Text(buf);  
        UART1_Write_Text("\r\n");  
    }  
    UART1_Write_Text("$STOP\r\n");  
}  
  
void      data_transfer_2()  
{  
    static      int      cnt;  
    //  
    UART1_Write_Text("\r\n$START\r\n");  
    for (cnt = 0; cnt < BEFOR_SIZE; cnt++) {  
        tmp = ad[befor_cnt];  
        befor_cnt++;  
        if (befor_cnt == BEFOR_SIZE)  
            befor_cnt = 0;  
        WordToStr(((double)tmp) * 48.828125 * 2.0, buf);  
        buf[6] = 0x00;  
        buf[5] = buf[4];  
        buf[4] = '.';  
        UART1_Write_Text(buf);  
        UART1_Write_Text("\r\n");  
    }  
    for (cnt = BEFOR_SIZE; cnt < (BEFOR_SIZE + AFTER_SIZE); cnt++)  
{  
        WordToStr(((double)ad[cnt]) * 48.828125 * 2.0, buf);  
        buf[6] = 0x00;
```

```
        buf[5] = buf[4];
        buf[4] = '.';
        UART1_Write_Text(buf);
        UART1_Write_Text("\r\n");
    }
    UART1_Write_Text("$STOP\r\n");
}

void    data_transfer_3()
{
    static    int    cnt;
    //
    UART1_Write_Text("\r\n$START\r\n");
    for (cnt = 0; cnt < BUFF_SIZE; cnt++) {
        tmp = ad[normal_cnt];
        normal_cnt++;
        if (normal_cnt == BUFF_SIZE)
            normal_cnt = 0;
        WordToStr(((double)tmp) * 48.828125 * 2.0, buf);
        buf[6] = 0x00;
        buf[5] = buf[4];
        buf[4] = '.';
        UART1_Write_Text(buf);
        UART1_Write_Text("\r\n");
    }
    UART1_Write_Text("$STOP\r\n");
}

//*****
*

void    measurement_01()
{
    for (normal_cnt = 0; normal_cnt < BUFF_SIZE; normal_cnt++) {
        ad[normal_cnt] = Adc_Read(0);
    }
    //
    LED = 1;
    data_transfer_1();
    LED = 0;
}

void    measurement_02()
{
    after_cnt = 0;
    while (1) {
        tmp = Adc_Read(0);
        if (tmp > THRESHOLD_LOW)
            break;
    }
    //
}
```

```
    ad[after_cnt] = tmp;
    for (after_cnt++; after_cnt < BUFF_SIZE; after_cnt++) {
        ad[after_cnt] = Adc_Read(0);
    }
    //
    LED = 1;
    data_transfer_1();
    LED = 0;
}

void      measurement_03()
{
    befor_cnt = 0;
    while (1) {
        tmp = Adc_Read(0);
        if (tmp > THRESHOLD_LOW)
            break;
        ad[befor_cnt] = tmp;
        befor_cnt++;
        if (befor_cnt == BEFOR_SIZE) {
            befor_cnt = 0;
        }
    }
    //
    after_cnt = BEFOR_SIZE;
    ad[after_cnt] = tmp;
    for (after_cnt++; after_cnt < (BEFOR_SIZE + AFTER_SIZE);
after_cnt++) {
        ad[after_cnt] = Adc_Read(0);
        Delay_us(10);
    }
    //
    LED = 1;
    data_transfer_2();
    LED = 0;
}

void      measurement_04()
{
    after_cnt = 0;
    while (1) {
        tmp = Adc_Read(0);
        if (tmp < THRESHOLD_HIGH)
            break;
    }
    //
    ad[after_cnt] = tmp;
    for (after_cnt++; after_cnt < BUFF_SIZE; after_cnt++) {
        ad[after_cnt] = Adc_Read(0);
    }
}
```

```
//
LED = 1;
data_transfer_1();
LED = 0;
}

void measurement_05()
{
    befor_cnt = 0;
    while (1) {
        tmp = Adc_Read(0);
        if (tmp < THRESHOLD_HIGH)
            break;
        ad[befor_cnt] = tmp;
        befor_cnt++;
        if (befor_cnt == BEFOR_SIZE) {
            befor_cnt = 0;
        }
    }
    //
    after_cnt = BEFOR_SIZE;
    ad[after_cnt] = tmp;
    for (after_cnt++; after_cnt < (BEFOR_SIZE + AFTER_SIZE);
after_cnt++) {
        ad[after_cnt] = Adc_Read(0);
        Delay_us(10);
    }
    //
    LED = 1;
    data_transfer_2();
    LED = 0;
}

void measurement_06()
{
    for (normal_cnt = 0; normal_cnt < BUFF_SIZE; normal_cnt++) {
        ad[normal_cnt] = Adc_Read(0);
        Delay_us(1000 - 50);
    }
    //
    LED = 1;
    data_transfer_1();
    LED = 0;
}

void measurement_07()
{
    after_cnt = 0;
    while (1) {
        tmp = Adc_Read(0);
        if (tmp > THRESHOLD_LOW)
```

```
                break;
            }
            //
            ad[after_cnt] = tmp;
            Delay_us(1000 - 50);
            for (after_cnt++; after_cnt < BUFF_SIZE; after_cnt++) {
                ad[after_cnt] = Adc_Read(0);
                Delay_us(1000 - 50);
            }
            //
            LED = 1;
            data_transfer_1();
            LED = 0;
        }

void      measurement_08()
{
    befor_cnt = 0;
    while (1) {
        tmp = Adc_Read(0);
        if (tmp > THRESHOLD_LOW)
            break;
        ad[befor_cnt] = tmp;
        befor_cnt++;
        if (befor_cnt == BEFOR_SIZE) {
            befor_cnt = 0;
        }
        Delay_us(1000 - 60);
    }
    //
    after_cnt = BEFOR_SIZE;
    ad[after_cnt] = tmp;
    Delay_us(1000 - 60);
    for (after_cnt++; after_cnt < (BEFOR_SIZE + AFTER_SIZE);
after_cnt++) {
        ad[after_cnt] = Adc_Read(0);
        Delay_us(1000 - 50);
    }
    //
    LED = 1;
    data_transfer_2();
    LED = 0;
}

void      measurement_09()
{
    after_cnt = 0;
    while (1) {
        tmp = Adc_Read(0);
        if (tmp < THRESHOLD_HIGH)
```

```
                break;
            }
            //
            ad[after_cnt] = tmp;
            for (after_cnt++; after_cnt < BUFF_SIZE; after_cnt++) {
                ad[after_cnt] = Adc_Read(0);
                Delay_us(1000 - 50);
            }
            //
            LED = 1;
            data_transfer_1();
            LED = 0;
        }

void    measurement_10()
{
    befor_cnt = 0;
    while (1) {
        tmp = Adc_Read(0);
        if (tmp < THRESHOLD_HIGH)
            break;
        ad[befor_cnt] = tmp;
        befor_cnt++;
        if (befor_cnt == BEFOR_SIZE) {
            befor_cnt = 0;
        }
        Delay_us(1000 - 60);
    }
    //
    after_cnt = BEFOR_SIZE;
    ad[after_cnt] = tmp;
    Delay_us(1000 - 60);
    for (after_cnt++; after_cnt < (BEFOR_SIZE + AFTER_SIZE);
after_cnt++) {
        ad[after_cnt] = Adc_Read(0);
        Delay_us(1000 - 50);
    }
    //
    LED = 1;
    data_transfer_2();
    LED = 0;
}

void    measurement_11()
{
    normal_cnt = 0;
    while (SW_START == 1) {
        ad[normal_cnt] = Adc_Read(0);
        normal_cnt++;
        if (befor_cnt == BUFF_SIZE) {
            normal_cnt = 0;
        }
    }
}
```

```
    }
}
//
LED = 1;
data_transfer_3();
LED = 0;
}

void      measurement_12()
{
    normal_cnt = 0;
    while (SW_START == 1) {
        ad[normal_cnt] = Adc_Read(0);
        normal_cnt++;
        if (befor_cnt == BUFF_SIZE) {
            normal_cnt = 0;
        }
        Delay_us(1000 - 50);
    }
    //
    LED = 1;
    data_transfer_3();
    LED = 0;
}

//*****
*

void      SwitchONcheck()
{
    while (Button(&PORTC, 1, 1, 0) == 0)
        ;
    while (Button(&PORTC, 1, 1, 1) == 0)
        ;
}

//*****
*

// Lcd pinout settings
sbit LCD_RS at RB2_bit;
sbit LCD_EN at RB3_bit;
sbit LCD_D7 at RB7_bit;
sbit LCD_D6 at RB6_bit;
sbit LCD_D5 at RB5_bit;
sbit LCD_D4 at RB4_bit;

// Pin direction
sbit LCD_RS_Direction at TRISB2_bit;
sbit LCD_EN_Direction at TRISB3_bit;
```

```
sbit LCD_D7_Direction at TRISB7_bit;
sbit LCD_D6_Direction at TRISB6_bit;
sbit LCD_D5_Direction at TRISB5_bit;
sbit LCD_D4_Direction at TRISB4_bit;

void main()
{
    static int cnt;
    static double ad;
    //ポートの設定
    TRISA = 0b00111111;
    TRISB = 0b11000000;
    TRISC = 0b10111110;
    //コンパレータは使用しない。
    CMCON.CM2 = 1;
    CMCON.CM1 = 1;
    CMCON.CM0 = 1;
    //A/D変換の設定
    ADCON1.PCFG3 = 1;
    ADCON1.PCFG2 = 1;
    ADCON1.PCFG1 = 0;
    ADCON1.PCFG0 = 0;
    //UART1の初期化
    UART1_Init(230400);
    //LCDの初期化
    Lcd_Init();
    Lcd_Cmd(_LCD_CURSOR_OFF);
    Lcd_Cmd(_LCD_CLEAR);
    Lcd_Out(1, 1, "one-shot logger v2");
    for (cnt = 0; cnt < 15; cnt++) {
        Lcd_Chrc(2, (cnt + 1), 0xFF);
        LED = 1;
        Delay_ms(100);
        LED = 0;
        Delay_ms(100);
    }
    Lcd_Cmd(_LCD_CLEAR);
    Lcd_Out(1, 1, "H:");
    Lcd_Out(1, 7, "mV");
    Lcd_Out(2, 1, "L:");
    Lcd_Out(2, 7, "mV");
    Lcd_Out(1, 10, "mode=");
    //
    while (1) {
        //THRESHOLD_HIGHの設定(この値以下をトリガレベルとする)
        ad = 0.0;
        for (cnt = 0; cnt < 1000; cnt++) {
            ad += Adc_Read(1);
        }
        ad = ad / 1000.0;
        WordToStr(ad * 4.8828125 * 2.0, buf);
    }
}
```

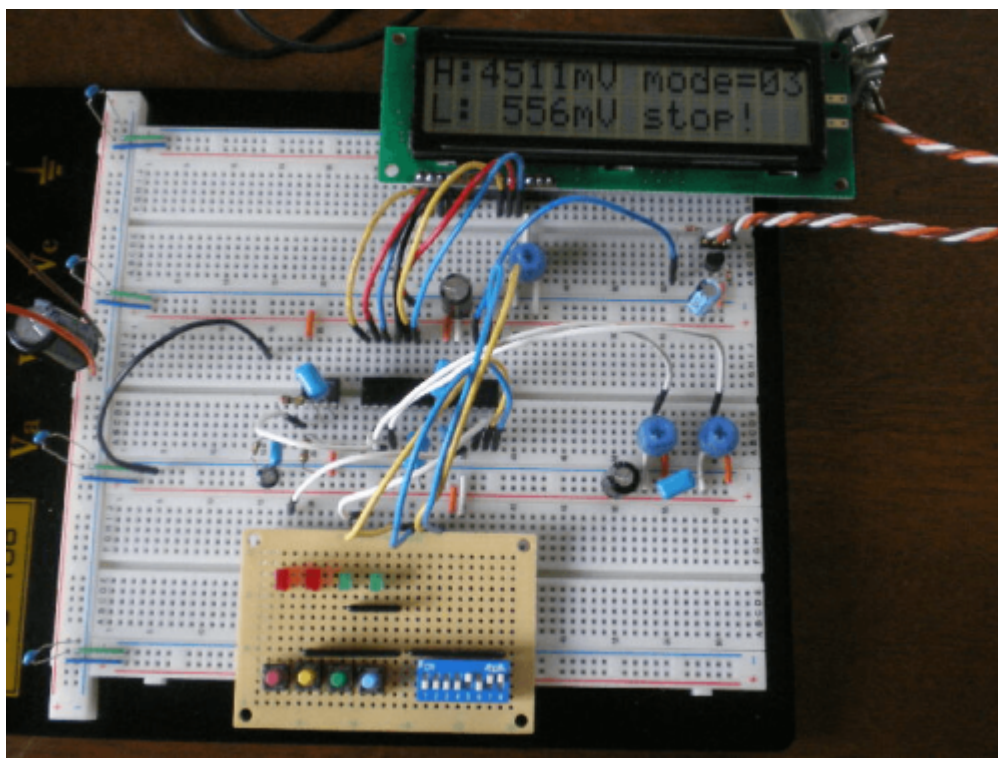
```
Lcd_Out(1, 3, &buf[1]);
THRESHOLD_HIGH = ad;
    //THRESHOLD_LOWの設定(この値以上をトリガレベルとする)
ad = 0.0;
for (cnt = 0; cnt < 1000; cnt++) {
    ad += Adc_Read(2);
}

ad = ad / 1000.0;
WordToStr(ad * 4.8828125 * 2.0, buf);
Lcd_Out(2, 3, &buf[1]);
THRESHOLD_LOW = ad;
//測定モードの設定
ByteToStr(((PORTC >> 2) & 0x0F) + 1, buf);
buf[1] = (buf[1] == ' ') ? '0' : buf[1];
Lcd_Out(1, 15, &buf[1]);
    //SwitchONcheck();
if (SW_START != 0) {
    continue;
}

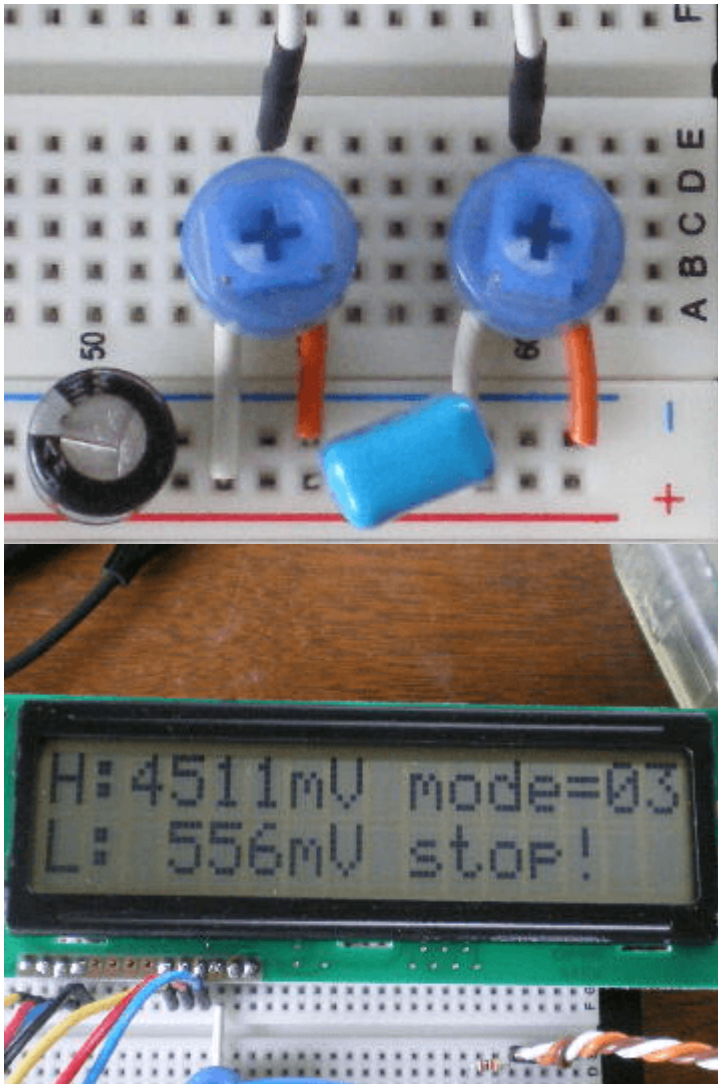
Lcd_Out(2, 10, "start!");
//測定
switch ((PORTC >> 2) & 0x0F) {
case 0:
    measurement_01();
    break;
case 1:
    measurement_02();
    break;
case 2:
    measurement_03();
    break;
case 3:
    measurement_04();
    break;
case 4:
    measurement_05();
    break;
case 5:
    measurement_06();
    break;
case 6:
    measurement_07();
    break;
case 7:
    measurement_08();
    break;
case 8:
    measurement_09();
    break;
case 9:
```

```
        measurement_10();  
        break;  
    case 10:  
        measurement_11();  
        break;  
    case 11:  
        measurement_12();  
        break;  
    }  
    Lcd_Out(2, 10, "stop! ");  
}  
}  
  
//*****  
*
```

動作確認



左側:トリガレベル設定用のVRです。多回転VRを使用した方が設定がし易いです。 右側:左上(THRESHOLD_HIGH) 左下(THRESHOLD_LOW) 右上(モード番号)、右下(状態表示)



その他の機能は、「簡易ワンショット・ロガー」と同じなので、其方を参照して下さい。

From:
<http://deepsky.jp/wiki/> - うごくといいな

Permanent link:
<http://deepsky.jp/wiki/doku.php?id=elechobby:picdic:otherpic:186&rev=1588247626>

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

