簡易ワンショット・ロガ□(PIC18F2620)

概要

オシロスコープは、周期的な波形(繰り返し波形)を観測するにはとても重宝するツールです。 しかし、極たまにしか発生しない信号、1回きりの信号、トリガ前の信号などの波形観測には、あまり向いていません。

そこで、今回は、これらの問題を解決すべく、簡易ワンショット・ロガーを製作しました。

<仕様>

- 入力チャネル□1CH
- 入力電圧レベル→最大10V
- サンプリング速度→高速モード(約60usec)□低速モード(約1msec)
- サンプリング数→イベント前(190件)、イベント後(1710件)
- 測定時間→高速モードで約114msec□低速モードで約1.9秒
- トリガモード→立ち上がり(500mV)□立下り(4500mV)□連続
- データ転送□RS232C□230400bps

動作原理

データを蓄積する方法としては□EEPROMやSDカードなどがありますが、何れも書き込み時間に数msecを要し てしまいます。そこでPIC18F2620が内蔵する約4kバイトのメモリのみを使用して高速な記録を行います。

<測定モード> ◎サンプリング速度(約60usec)□データ数(1900個)、記録時間(約114msec)

- モード(01):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(02):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(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。

◎サンプリング速度(約1msec)□データ数(1900個)、記録時間(約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。

	SW1	SW2	SW3	SW4
モード(01)	ON	ON	ON	ON
モード(02)	OFF	ON	ON	ON
モード(03)	ON	OFF	ON	ON
モード(04)	OFF	OFF	ON	ON
モード(05)	ON	ON	OFF	ON
モード(06)	OFF	ON	OFF	ON
モード(07)	ON	OFF	OFF	ON
モード(08)	OFF	OFF	OFF	ON
モード(09)	ON	ON	ON	OFF
モード(10)	OFF	ON	ON	OFF
モード(11)	ON	OFF	ON	OFF
モード(12)	OFF	OFF	ON	OFF

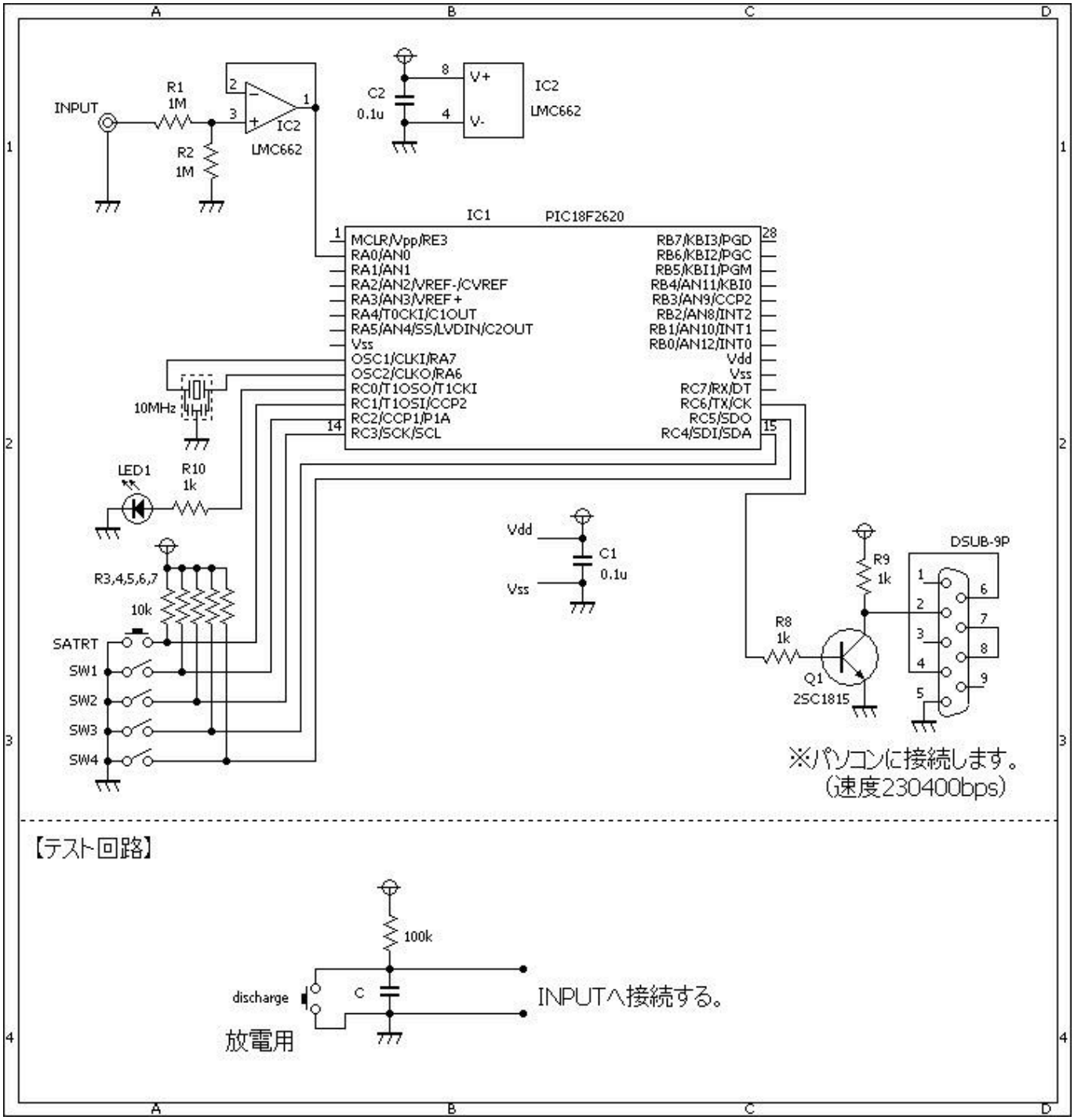
<イベント前のデータ蓄積> イベント前(トリガレベルに達する前)のデータは、メモリをリングバッファ(ring buffer)として使用し、トリガレベル に達するまで、データをエンドレスに蓄積し続けます。(リングバッファのサイズは、190件)

<イベント後のデータ蓄積> 単純にバッファサイズに達するまで、データを蓄積し続けます。(バッファのサイズは、1710件)

<データ転送>測定が完了すると、メモリ内のデータ(1900件)を、RS232C経由(通信速度230400bps)で、パソコンに転送しま す。転送フォーマットは次のようになります。(電圧の単位はmVです)

```
$START
1234.5
2345.6
:
:
$STOP
```

回路図



ソースコード

※開発環境には□□mikroC PRO□を使用しています。

trigger_adc_v3.c

```
//*****************************************************************************
*
/*
    ＜簡易ワンショット・ロガー＞

  ■動作モード
```

```
    ◎サンプリング速度（約50usec□□データ数（1900個）、記録時間□95msec□
        ・モード(01)　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(02)　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(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。

    ◎サンプリング速度（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               1900
#define                 BEFOR_SIZE              190
```

```
#define                 AFTER_SIZ               1710

#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];

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

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) {
```

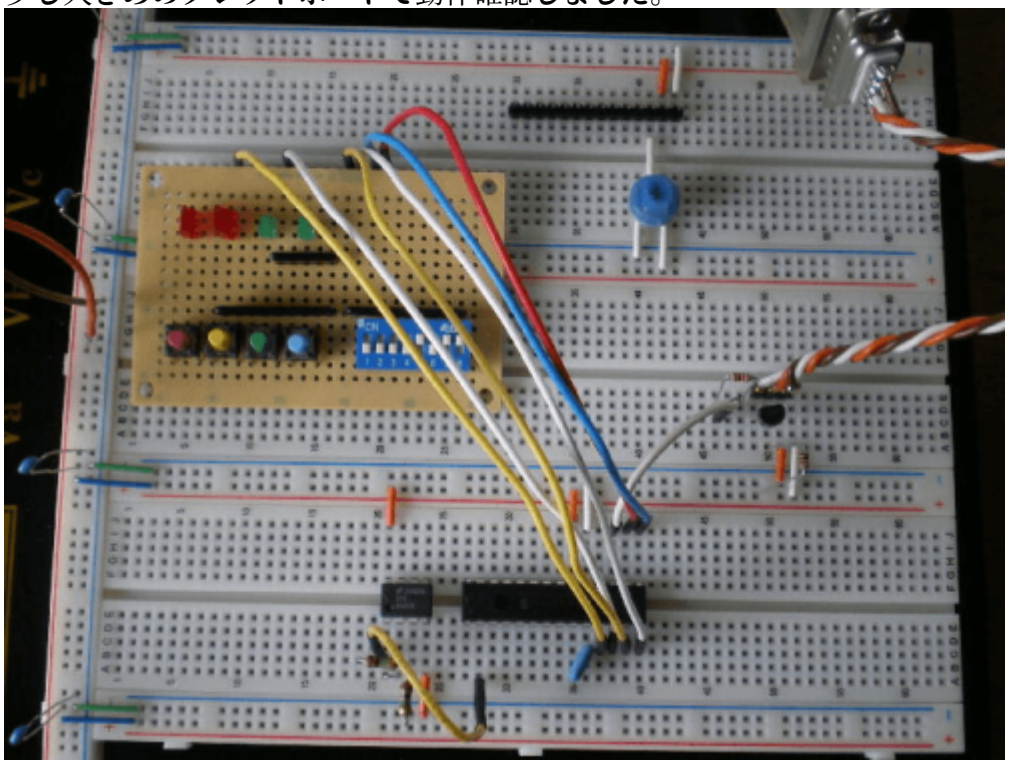
```
                ad[normal_cnt] = Adc_Read(0);
                normal_cnt++;
                if (befor_cnt == BUFF_SIZE) {
                        normal_cnt = 0;
                }
        }
        //
        LED = 1;
        data_transfer_3();
        LED = 0;
}

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)
                ;
}

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

void        main()
{
        static        short        cnt;
        //ポートの設定
        TRISA = 0b00111101;
        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;
        //□□□変換の設定
        ADCON1.PCFG3 = 1;
        ADCON1.PCFG2 = 1;
        ADCON1.PCFG1 = 1;
        ADCON1.PCFG0 = 0;
        //
        UART1_Init(230400);
        Delay_ms(500);
        //
        for (cnt = 0; cnt < 5; cnt++) {
                LED = 1;
                Delay_ms(100);
                LED = 0;
                Delay_ms(100);
        }
        //
        while (1) {
                SwitchONcheck();
                //
                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;
```

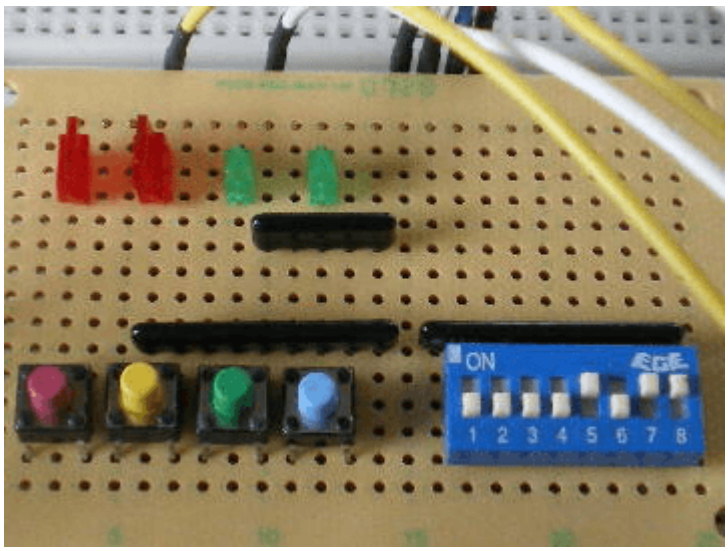
```
                case 9:
                        measurement_10();
                        break;
                case 10:
                        measurement_11();
                        break;
                case 11:
                        measurement_12();
                        break;
                }
        }
}

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

動作確認

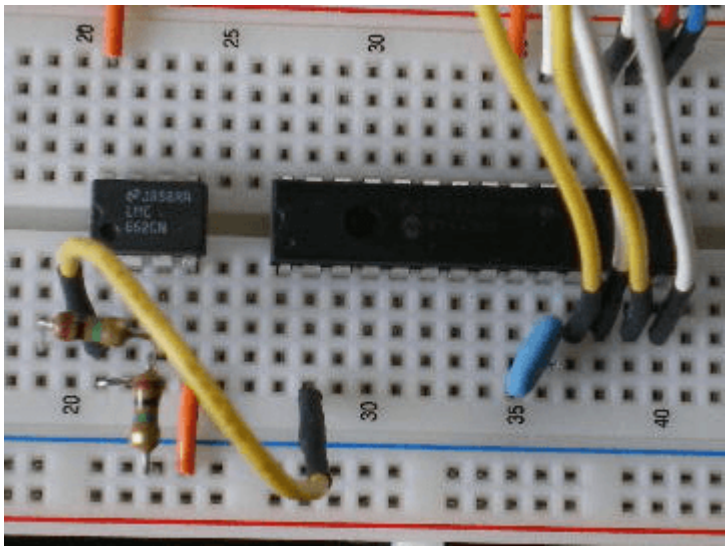
少し大きめのブレッドボードで動作確認しました。



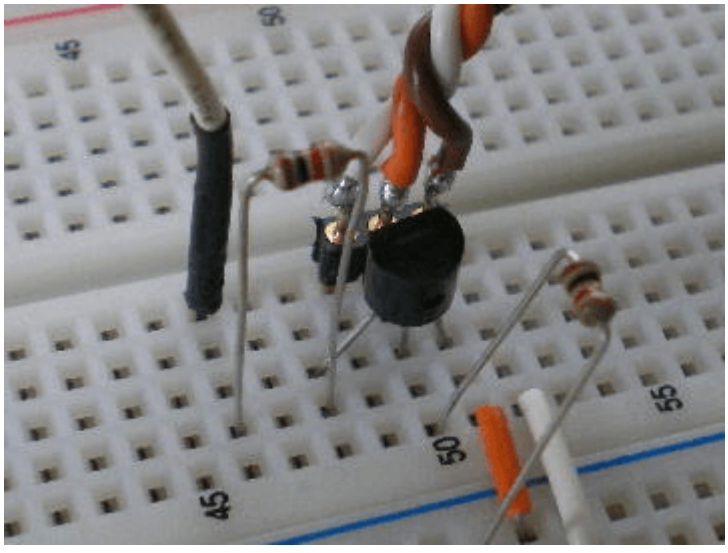
左側:LED□プッシュスイッチ、ディップスイッチを汎用的に使えるように、基板に実装しました。 右側:

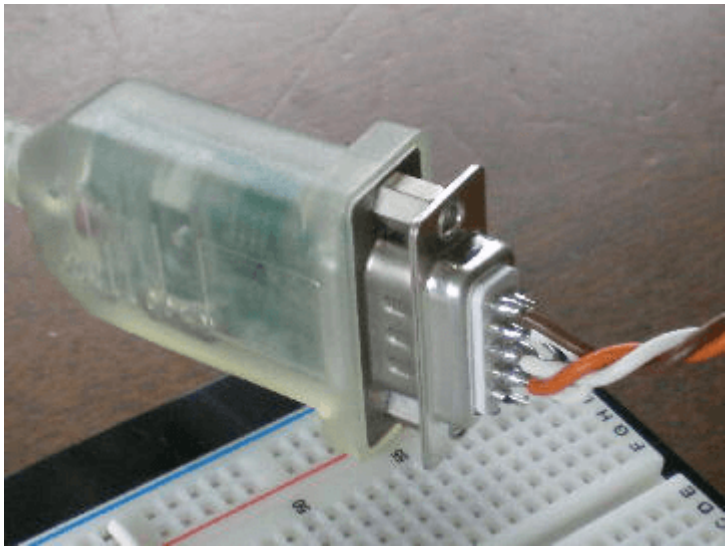


オペアンプ(LMC662)と**PIC18F2620です。**

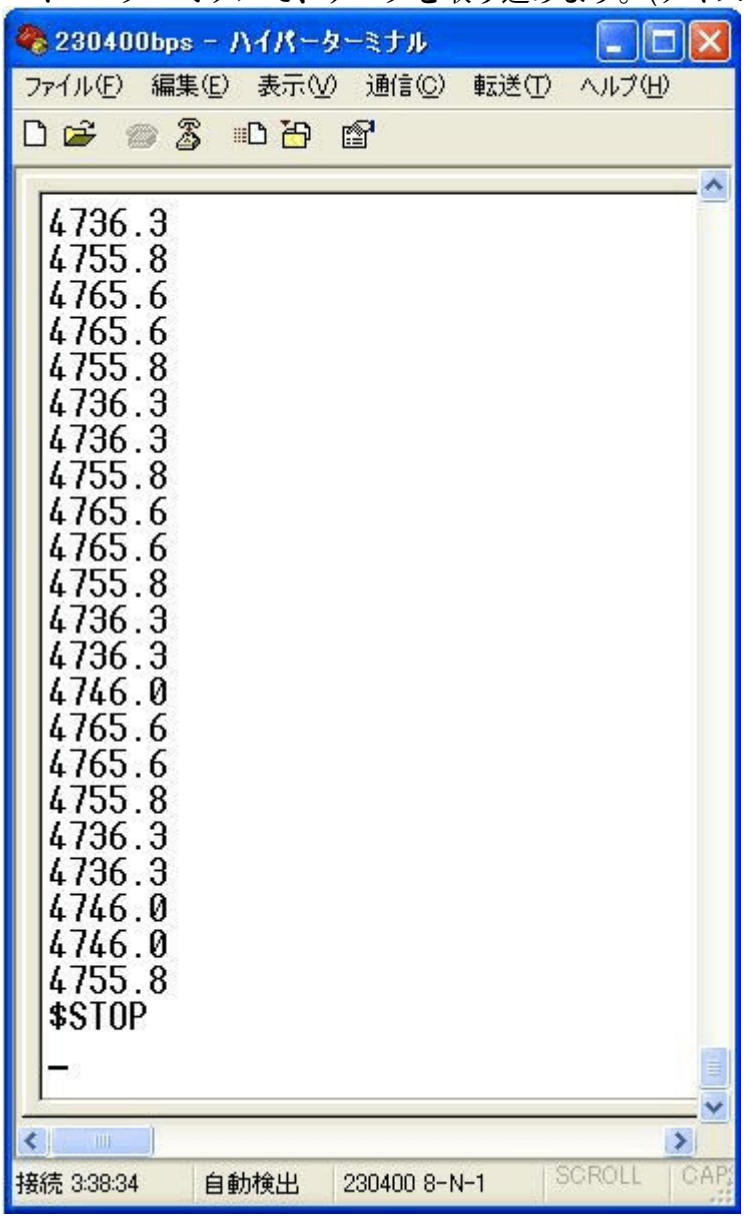


左側:RS232C**レベル**変換用**のトランジスタ**(2SC1815)**です。** 右側:USB**シリアル**変換**モジュールです。**

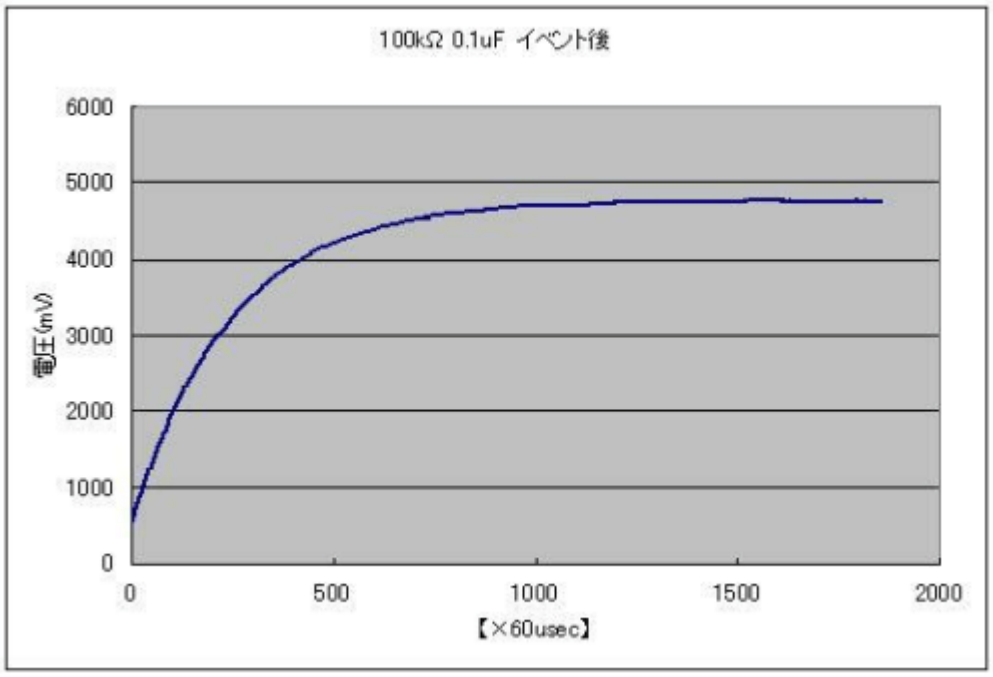


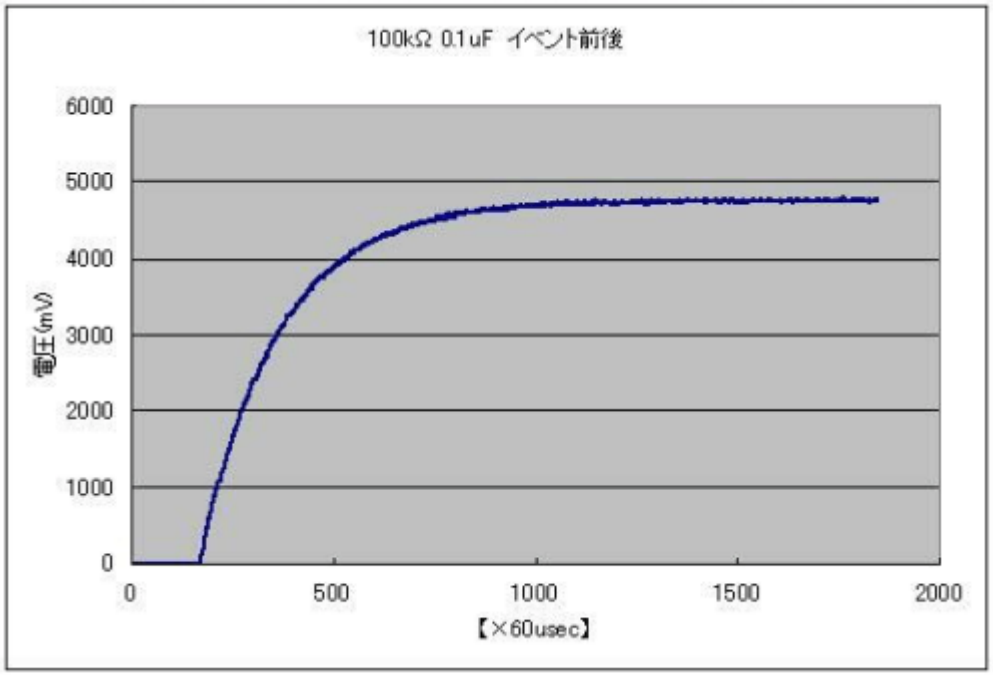


ハイパーターミナルで、データを取り込みます。(テキストのキャプチャー機能を使います)

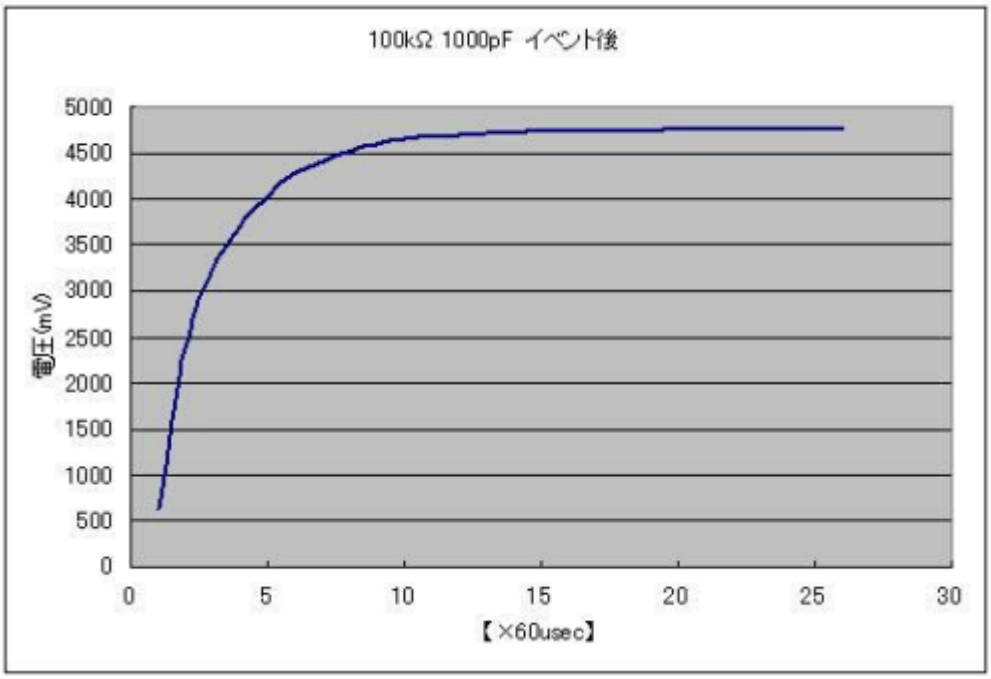


取り込んだデータをグラフ表示させて見ました□ CR積分回路(100kΩ□0.1uF)の波形です。サンプリング周期=60usec□トリガレベル=500mV 上側:イベント後の波形です。※トリガレベル以前のデータを見ることが出来ません。 下側:イベント前後の波形です。

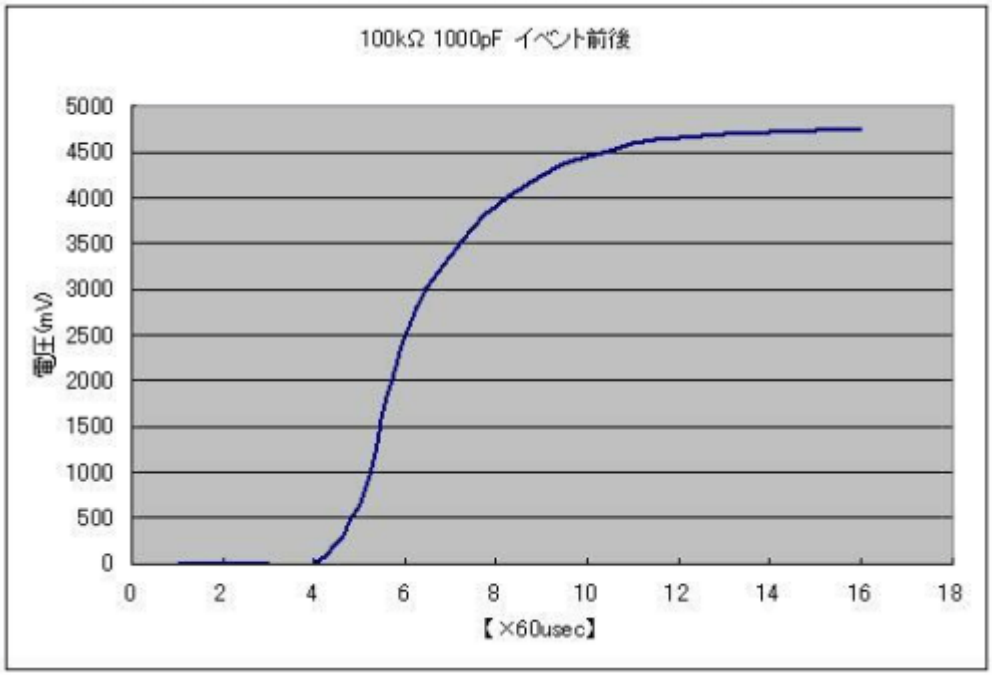




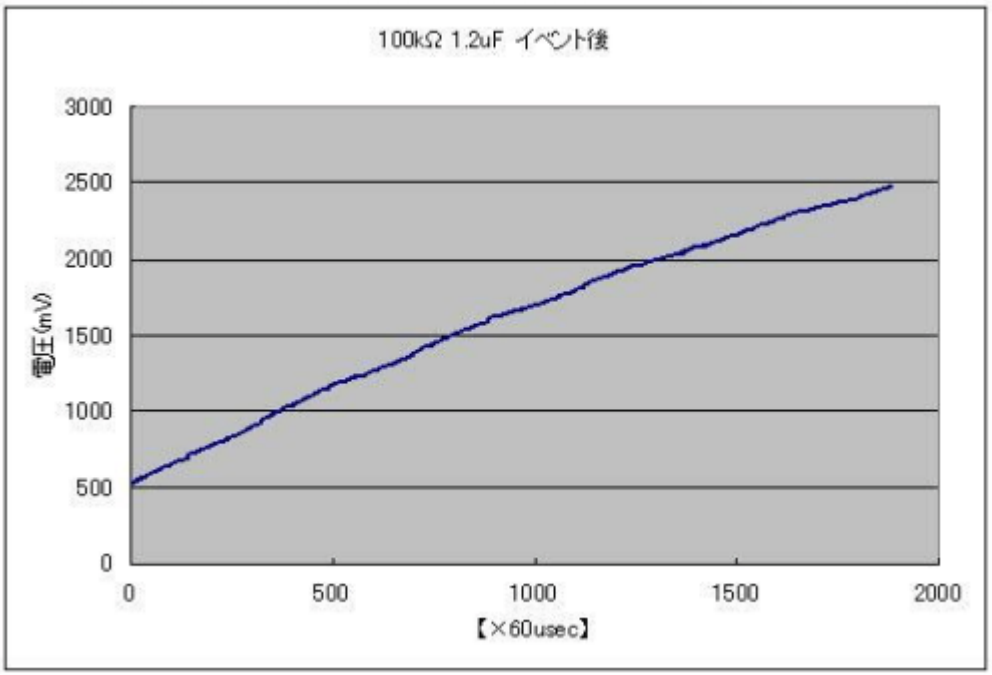
CR積分回路(100kΩ□1000pF)の波形です。サンプリング周期=60usec□トリガレベル=500mV 上側:イベント後の波形です。※トリガレベル以前のデータを見ることが出来ません。 下側:イベント前後の波形



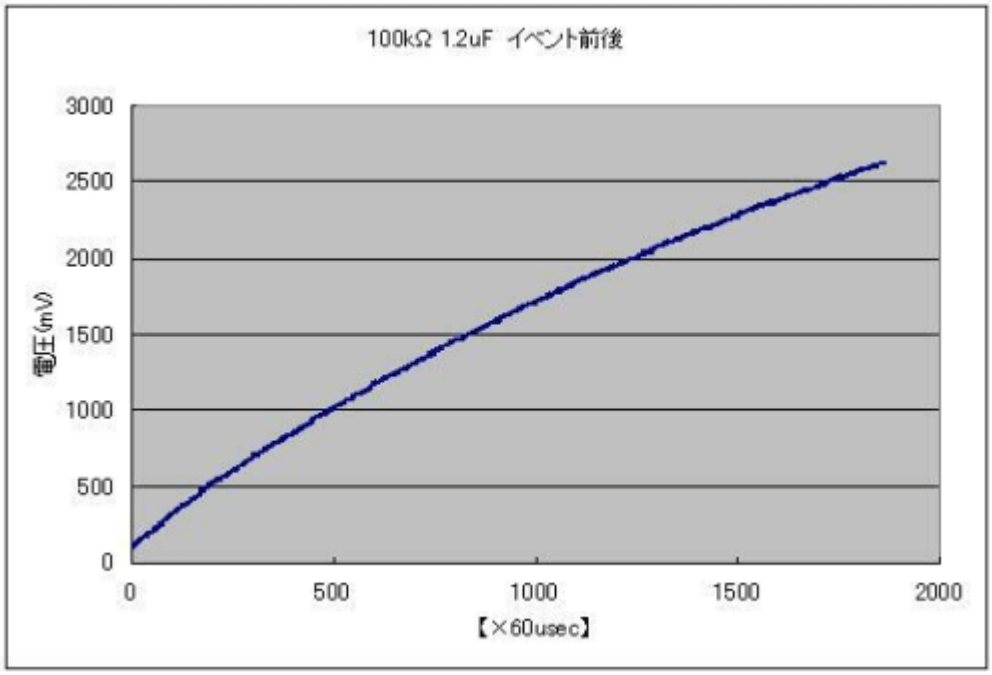
です。



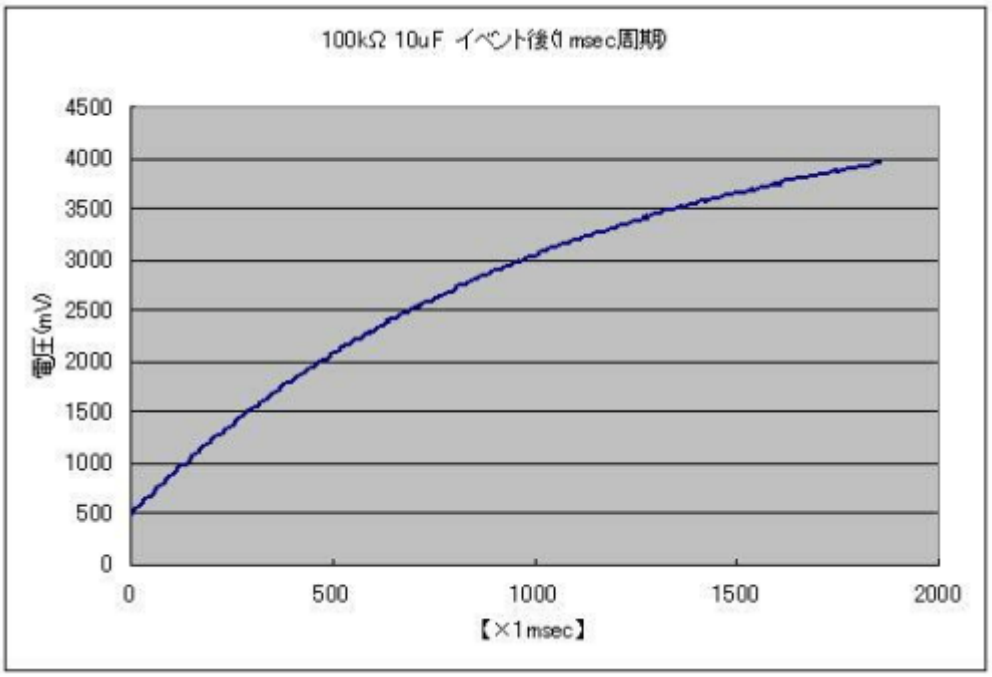
CR積分回路(100kΩ□1.2uF)の波形です。サンプリング周期=60usec□トリガレベル=500mV 上側:イベント後の波形です。※トリガレベル以前のデータを見ることが出来ません。 下側:イベント前後の波形で



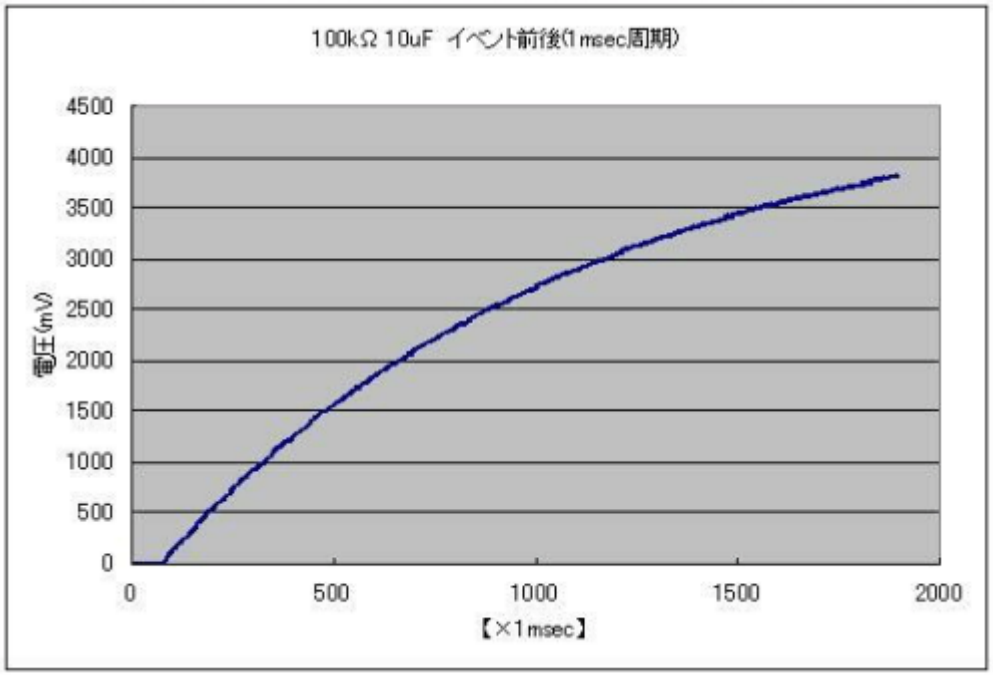
す。



CR積分回路(100kΩ□10uF)の波形です。サンプリング周期=1msec□トリガレベル=500mV 上側:イベント後の波形です。※トリガレベル以前のデータを見ることが出来ません。 下側:イベント前後の波形で



す。



トランス式ACアダプタ(上側)とスイッチング式ACアダプタ(下側)の立ち上がりの波形です。