ADロガ□(FAT32対応)(PIC18F4585)

概要

前回製作した、「簡易ADロガ□(FAT32対応)」には、画面表示(LCD)や日付時刻(RTC)が搭載されていない ため若干使い勝手が良くありませんでした。

- 記録する前に、測定対象の電圧がどれくらいなのかが分からない。
- 日付時刻管理機能が搭載されていないため、測定時に日付時刻をメモする必要があった。

そこでこれらの問題点を解決し、より使い勝手の良いADロガーを製作しました。

仕様

◎記録媒体

- SDカード(FAT32)
- 記録されるファイル名は、“LOG_0001.TXT“~“LOG_9999.TXT”です。

◎入力チャネル

- 11チャネル

◎測定周期(サンプリング速度)

- 100msec□1sec□1min□10min

◎平均測定回数

- 1回、10回、100回、1000回

◎測定分解能

- A/D変換10ビット(1024)、基準電圧=5V
- 約4.9mV(5V÷1024)

◎入力電圧範囲

- 0V~5V(分圧抵抗を付加することにより範囲を拡張することが可能)

◎データ表示

- LCD(20文字×4行)

◎日付時刻管理

- RTC(リアルタイムクロック)搭載

動作原理(ハードウェア)

◎全体制御

- PIC18F4585を使用します。
- ADCモジュール<アナログデータをデジタルデータに変換するために使用、10ビット分解能×11チャネル>
- TIMERモジュール<正確なサンプリング周期を得るために使用>
- SPIモジュール<SDCと通信するために使用>
- EEPROMモジュール<各種設定値を記憶するために使用>

◎SDC

- FAT32対応のSDCを使用します。

◎LCD

- 20文字x4行のキャラクタ表示液晶モジュールを使用します。

◎RTC

- セイコーのリアルタイムクロックIC(RTC-8564NB)を使用します。
- PICとはI2Cインターフェース(2線式)で通信します。

◎3.3Vレギュレータ

- 低損失型3端子レギュレータ<TA48M033F>を使用します。(電圧差0.65Vで動作可能)
- SDC駆動用に必要な電圧です。

◎5Vレギュレータ

- 低損失型3端子レギュレータ<TA48M05F>を使用します。(電圧差0.5Vで動作可能)

◎電源

- ACアダプタ<8V~12V500mA程度>を使用します。

◎RTCバックアップ用コンデンサ

- RTCは、電源が切れると内容が消えてしまうので、コイン型の電気二重層コンデンサ(1F□5.5V)でバックアップを行います。

動作原理(ソフトウェア)

◎PICのクロックを32MHzに設定します□□init_osc()□ ◎コンパレータを設定します□□init_comparator()□ ◎ポートを設定します□□init_port()□ ◎ADCを設定します□□init_adc()□ ◎SDCを初期化します□□init_sdc_fat32()□ ◎RTCを初期化します□□init_rtc()□ ◎モードスイッチ(SW_MODE)が押下されていれば□RTCの日付時刻を1月1日0時0分0秒に設定します。 ◎タイマ□(10msecの周期割り込み用)を設定します□□init_timer()□ ◎EEPROMより「測定周期」、「平均化回数」を読み込みます□ ◎LEDを3回点滅させます。 ◎日付時刻をLCDに表示します。 ◎動作モードを取得します□□get_mode()□ ◎測定、表示、記録モード□run_mode_0()□

- CH1~CH11の電圧をAD変換で取り込みLCDに表示します。
- スタートスイッチ(SW_START)が押下されるのを待ちます。
- ファイルをオープンします。
- 測定および結果のSDカードへの書き込みを行います。
- ストップスイッチ(SW_STOP)が押下されるとファイルをクローズし終了します。

◎時刻合わせモード(run_mode_1())

- スイッチ(SW_HOUR)が押下されると「時」を+1しRTCに設定します。
- スイッチ(SW_MINUTE)が押下されると「分」を+1しRTCに設定します。

◎日付合わせモード(run_mode_2())

- スイッチ(SW_MONTH)が押下されると「月」を+1しRTCに設定します。
- スイッチ(SW_DAY)が押下されると「日」を+1しRTCに設定します。

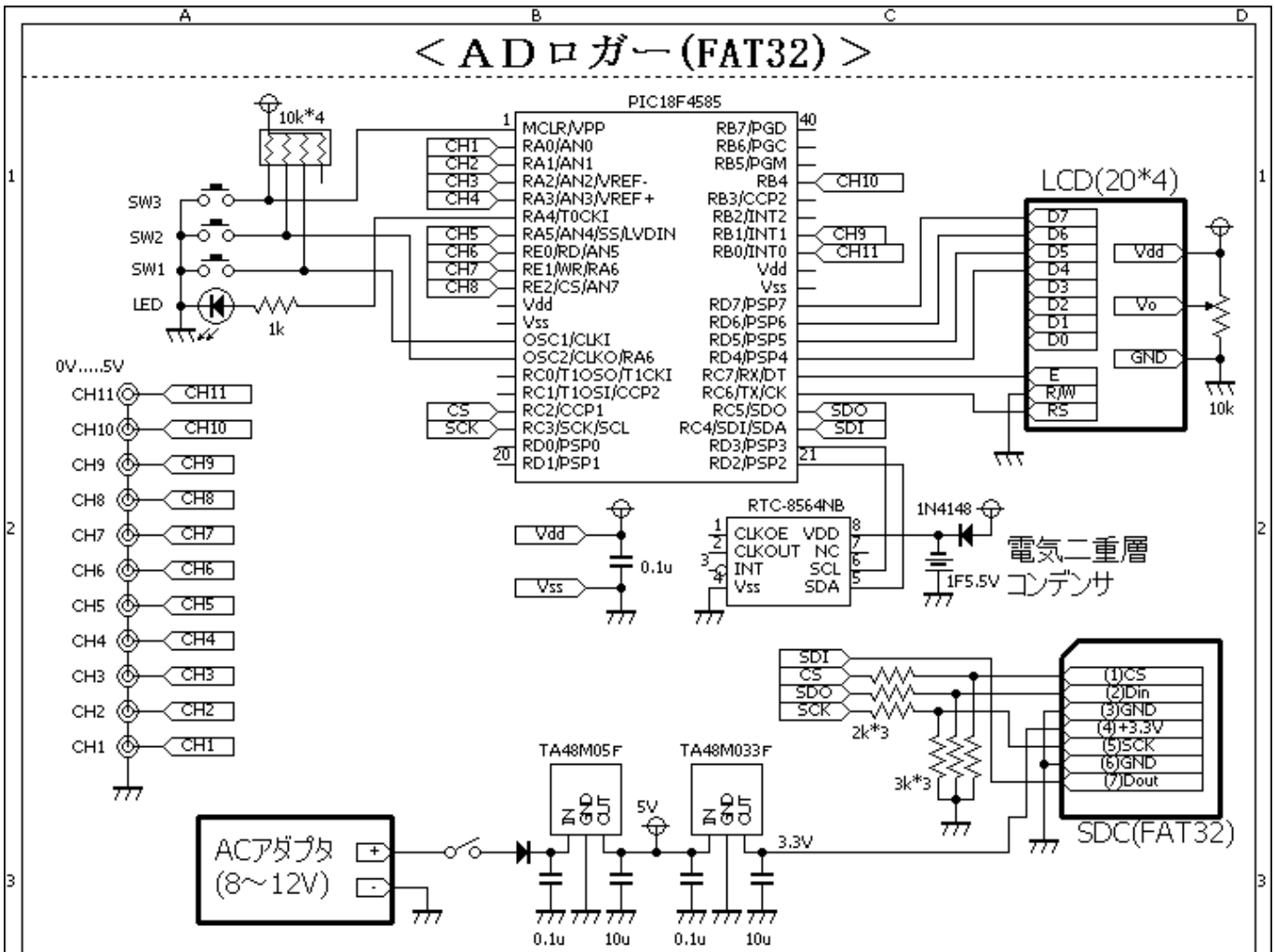
◎測定周期設定モード(run_mode_3())

- スイッチ(SW_UP)が押下されると「測定周期」を長くします(<100msec→1sec→1min→10min>
- スイッチ(SW_DOWN)が押下されると「測定周期」を短くします(<10min→1min→1sec→100msec>
- 設定値はEEPROMに記録されます。

◎平均化回数設定モード(run_mode_4())

- スイッチ(SW_UP)が押下されると「平均化回数」を長くします。<1回→10回→100回→1000回>
- スイッチ(SW_DOWN)が押下されると「測定周期」を短くします。<1000回→100回→10回→1回>
- 設定値はEEPROMに記録されます。

回路図



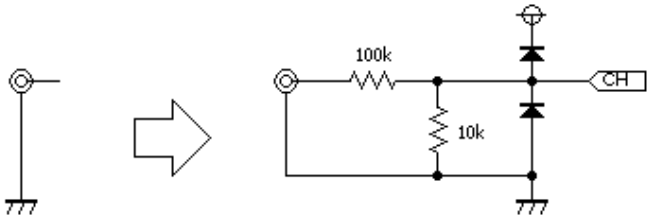
■スイッチの機能割り当て

SW1 [SW_START、SW_HOUR、SW_MONTH、SW_UP]
SW2 [SW_STOP、SW_MINUTE、SW_DAY、SW_DOWN]
SW3 [SW_MODE]

■仕様

- 入力チャネル＜11チャネル＞
- 測定周期＜100msec、1sec、1min、10min＞
- 平均測定回数＜1回、10回、100回、1000回＞
- 測定分解能＜約4.9mV＞
- 入力電圧範囲＜0V～5V＞

■入力電圧範囲の変更方法



※抵抗で分圧することにより、入力電圧範囲を拡張することが出来ます。
※この例では、11倍になります。(0V～55V)

ソースコード

ad_logger_fat32_v3.c

```
//*********************************************************************
***********
/*
□□□□□ロガ□(FAT32)□□□
■入力チャネル
  ・11チャネル
■測定分解能
  ・約4.9mV
■入力電圧範囲
□□0V□5V
■記録メディア
□□□□□(FAT32)
■測定周期
□□100msec□1sec□1min□10min
■平均測定回数
  ・1回、10回、100回
*/
//*********************************************************************
***********
#define BYTE    unsigned        short
#define WORD    unsigned        int
#define DWORD   unsigned        long
//
#include "__Lib_FAT32.h"
//
sbit    SW1             at      RA7_bit;
sbit    SW2             at      RA6_bit;
sbit    SW3             at      RE3_bit;
sbit    LED             at      RA4_bit;
//
#define SW_START        SW1
#define SW_STOP         SW2
#define SW_MODE         SW3
//
#define SW_HOUR         SW1
#define SW_MINUTE       SW2
//
#define SW_MONTH        SW1
#define SW_DAY          SW2
//
#define SW_UP           SW1
#define SW_DOWN         SW2
//*********************************************************************
***********
extern  void    main();
extern  short   get_mode();
extern  void    run_mode_0();
```

```
extern  void    run_mode_1();
extern  void    run_mode_2();
extern  void    run_mode_3();
extern  void    run_mode_4();
//
extern  void    init_osc();
extern  void    init_port();
extern  void    init_comparator();
extern  void    init_adc();
extern  void    init_sdc_fat32();
extern  void    init_timer();
//
extern  void    init_lcd();
extern  void    RegistCustomChar();
extern  void    BarDisp(char row, char column, short dat, short bar);
//
extern  BYTE    RTC_8564_Read(BYTE addr);
extern  void    RTC_8564_Write(BYTE addr, BYTE dat);
extern  void    get_date_time(char *dat);
extern  void    set_date_time(char *dat);
extern  void    disp_date_time();
extern  void    init_rtc();
extern  void    clock2timedate2(__TIME *tm);
//
extern  DWORD   get_clock();
extern  void    set_clock(DWORD clock);
extern  void    clock2str(DWORD clock, char *str);
extern  void    clock2timedate(DWORD clock, __TIME *tm);
extern  long    wait_trigger(short mode);
extern  WORD    ADC_Get_Sample_Average(BYTE channel);
//************************************************************************
***********
char    *tytle = "ad_logger_fat32 v3.1";
char    buf[70];
char    *file_name = "log_0000.txt";
WORD    file_id = 0;
__HANDLE        fileHandle;
__TIME          TimeDate;
BYTE    td_hour, td_min, td_sec, td_month, td_day;
short   run_mode = 0;
short   cycle_mode = 1;
short   sampling_mode = 2;
char    *cycle_str[4] = {
        "100msec",
        "1sec   ",
        "1min   ",
        "10min  "
};
char    *sampling_str[4] = {
        "1    ",
```

```
        "10  ",
        "100 ",
        "1000"
};
short   change_flg = 0;
//*****************************************************************************
***********
//■メイン関数
void    main()
{
        short   cnt;
        //
        init_osc();
        init_comparator();
        init_port();
        init_adc();
        init_lcd();
        init_sdc_fat32();
        init_rtc();
        if (SW_MODE == 0) {
                Lcd_Out(1, 1, "date&time initialize");
                Lcd_Out(3, 1, "-- 01/01 00:00:00 --");
                while (SW_MODE == 0) {
                        Delay_ms(100);
                }
                set_date_time("01/01 00:00:00");
                Lcd_Cmd(_LCD_CLEAR);
        }
        init_timer();
        //
        cycle_mode = EEPROM_Read(0);
        if ((cycle_mode < 0) || (cycle_mode > 3)) {
                cycle_mode = 1;
        }
        sampling_mode = EEPROM_Read(1);
        if ((sampling_mode < 0) || (sampling_mode > 3)) {
                sampling_mode = 1;
        }
        //
        for (cnt = 0; cnt < 3; cnt++) {
                LED = 1;
                Delay_ms(100);
                LED = 0;
                Delay_ms(100);
        }
        //
        while (1) {
                get_date_time(buf);
                Lcd_Out(1, 1, buf);
                //
                switch (get_mode()) {
```

```
                case 0://測定、表示、記録
                        run_mode_0();
                        break;
                case 1://時刻合わせ
                        run_mode_1();
                        break;
                case 2://日付合わせ
                        run_mode_2();
                        break;
                case 3://測定周期設定
                        run_mode_3();
                        break;
                case 4://平均化回数設定
                        run_mode_4();
                        break;
                }
        }
}
//**********************************************************************
***********
void    run_mode_0()
{
        double  ch1, ch2, ch3, ch4, ch5, ch6, ch7, ch8, ch9, ch10,
ch11;
        long    clock;
        //
        ch1 = ADC_Get_Sample_Average(0);
        ch2 = ADC_Get_Sample_Average(1);
        ch3 = ADC_Get_Sample_Average(2);
        ch4 = ADC_Get_Sample_Average(3);
        ch5 = ADC_Get_Sample_Average(4);
        ch6 = ADC_Get_Sample_Average(5);
        ch7 = ADC_Get_Sample_Average(6);
        ch8 = ADC_Get_Sample_Average(7);
        ch9 = ADC_Get_Sample_Average(8);
        ch10 = ADC_Get_Sample_Average(9);
        ch11 = ADC_Get_Sample_Average(10);
        //
        ch1 *= 4.8828125;
        ch2 *= 4.8828125;
        ch3 *= 4.8828125;
        ch4 *= 4.8828125;
        ch5 *= 4.8828125;
        ch6 *= 4.8828125;
        ch7 *= 4.8828125;
        ch8 *= 4.8828125;
        ch9 *= 4.8828125;
        ch10 *= 4.8828125;
        ch11 *= 4.8828125;
        //
```

```
        WordToStr(ch1, buf);
        WordToStr(ch2, &buf[5]);
        WordToStr(ch3, &buf[10]);
        WordToStr(ch4, &buf[15]);
        Lcd_Out(2, 1, buf);
        WordToStr(ch5, buf);
        WordToStr(ch6, &buf[5]);
        WordToStr(ch7, &buf[10]);
        WordToStr(ch8, &buf[15]);
        Lcd_Out(3, 1, buf);
        WordToStr(ch9, buf);
        WordToStr(ch10, &buf[5]);
        WordToStr(ch11, &buf[10]);
        Lcd_Out(4, 1, buf);
        //
        if (SW_START == 1) {
                return;
        }
        //
        Lcd_Chr(1, 20, 0xFF);
        LED = 1;
        //
        file_id++;
        WordToStr(file_id, buf);
        file_name[4] = (buf[1] == ' ') ? '0' : buf[1];
        file_name[5] = (buf[2] == ' ') ? '0' : buf[2];
        file_name[6] = (buf[3] == ' ') ? '0' : buf[3];
        file_name[7] = (buf[4] == ' ') ? '0' : buf[4];
        Lcd_Out(2, 1, "                    ");
        Lcd_Out(3, 1, "                    ");
        Lcd_Out(3, 1, file_name);
        Lcd_Out(4, 1, "                    ");
        Delay_ms(500);
        Lcd_Out(3, 1, "                    ");
        //
        clock2timedate2(&TimeDate);
        FAT32_SetTime(&TimeDate);
        Delay_ms(10);
        FAT32_Delete(file_name);
        Delay_ms(10);
        fileHandle = FAT32_Open(file_name, FILE_WRITE);
        FAT32_Write(fileHandle, "$START ", 7);
        get_date_time(buf);
        buf[14] = '\r';
        buf[15] = '\n';
        FAT32_Write(fileHandle, buf, 16);
        set_clock(0);
        while (SW_STOP != 0) {
                clock = wait_trigger(cycle_mode);
                if (clock == -1) {
                        break;
```

```
                }
                //
                clock2str(clock, buf);
                //
                ch1 = ADC_Get_Sample_Average(0);
                ch2 = ADC_Get_Sample_Average(1);
                ch3 = ADC_Get_Sample_Average(2);
                ch4 = ADC_Get_Sample_Average(3);
                ch5 = ADC_Get_Sample_Average(4);
                ch6 = ADC_Get_Sample_Average(5);
                ch7 = ADC_Get_Sample_Average(6);
                ch8 = ADC_Get_Sample_Average(7);
                ch9 = ADC_Get_Sample_Average(8);
                ch10 = ADC_Get_Sample_Average(9);
                ch11 = ADC_Get_Sample_Average(10);
                //
                ch1 *= 4.8828125;
                ch2 *= 4.8828125;
                ch3 *= 4.8828125;
                ch4 *= 4.8828125;
                ch5 *= 4.8828125;
                ch6 *= 4.8828125;
                ch7 *= 4.8828125;
                ch8 *= 4.8828125;
                ch9 *= 4.8828125;
                ch10 *= 4.8828125;
                ch11 *= 4.8828125;
                //
                WordToStr(ch1, &buf[12]);
                WordToStr(ch2, &buf[17]);
                WordToStr(ch3, &buf[22]);
                WordToStr(ch4, &buf[27]);
                WordToStr(ch5, &buf[32]);
                WordToStr(ch6, &buf[37]);
                WordToStr(ch7, &buf[42]);
                WordToStr(ch8, &buf[47]);
                WordToStr(ch9, &buf[52]);
                WordToStr(ch10, &buf[57]);
                WordToStr(ch11, &buf[62]);
                buf[67] = '\r';
                buf[68] = '\n';
                FAT32_Write(fileHandle, buf, 69);
        }
        FAT32_Write(fileHandle, "$STOP  ", 7);
        get_date_time(buf);
        buf[14] = '\r';
        buf[15] = '\n';
        FAT32_Write(fileHandle, buf, 16);
        FAT32_Close(fileHandle);
        LED = 0;
```

```
        Lcd_Chr(1, 20, ' ');
}
//**********************************************************************
***********
void    run_mode_1()
{
        BYTE    td;
        //
        Lcd_Out(2, 1, "clock adjust        ");
        //“時”の設定
        if (SW_HOUR == 0) {
                while (SW_HOUR == 0) {
                        Delay_ms(100);
                }
                td_hour = RTC_8564_Read(4) & 0x3F;
                td = Bcd2Dec(td_hour);
                td++;
                if (td >= 24) {
                        td = 0;
                }
                td_hour = Dec2Bcd(td);
                RTC_8564_Write(4, td_hour);
        }
        //“分”の設定
        if (SW_MINUTE == 0) {
                while (SW_MINUTE == 0) {
                        Delay_ms(100);
                }
                td_min = RTC_8564_Read(3) & 0x7F;
                td = Bcd2Dec(td_min);
                td++;
                if (td >= 60) {
                        td = 0;
                }
                td_min = Dec2Bcd(td);
                RTC_8564_Write(3, td_min);
                RTC_8564_Write(2, 0);
        }
}
//**********************************************************************
***********
void    run_mode_2()
{
        BYTE    td;
        //
        Lcd_Out(2, 1, "date adjust         ");
        //“月”の設定
        if (SW_MONTH == 0) {
                while (SW_MONTH == 0) {
                        Delay_ms(100);
                }
```

```
                td_month = RTC_8564_Read(7) & 0x1F;
                td = Bcd2Dec(td_month);
                td++;
                if (td >= 13) {
                        td = 1;
                }
                td_month = Dec2Bcd(td);
                RTC_8564_Write(7, td_month);
        }
        //“日”の設定
        if (SW_DAY == 0) {
                while (SW_DAY == 0) {
                        Delay_ms(100);
                }
                td_day = RTC_8564_Read(5) & 0x3F;
                td = Bcd2Dec(td_day);
                td++;
                if (td >= 32) {
                        td = 1;
                }
                td_day = Dec2Bcd(td);
                RTC_8564_Write(5, td_day);
        }
}
//*****************************************************************************
***********
void    run_mode_3()
{
        Lcd_Out(2, 1, "cycle mode          ");
        //
        if (change_flg == 1) {
                change_flg = 0;
                Lcd_Out(3, 5, cycle_str[cycle_mode]);
        }
        //
        if (SW_UP == 0) {
                while (SW_UP == 0) {
                        Delay_ms(100);
                }
                //
                cycle_mode++;
                if (cycle_mode == 4) {
                        cycle_mode = 3;
                }
                Lcd_Out(3, 5, cycle_str[cycle_mode]);
        }
        //
        if (SW_DOWN == 0) {
                while (SW_DOWN == 0) {
                        Delay_ms(100);
```

```
                }
                //
                cycle_mode--;
                if (cycle_mode == -1) {
                        cycle_mode = 0;
                }
                Lcd_Out(3, 5, cycle_str[cycle_mode]);
        }
        //
        EEPROM_Write(0, cycle_mode);
}
//*********************************************************************
***********
void    run_mode_4()
{
        Lcd_Out(2, 1, "sampling mode       ");
        //
        if (change_flg == 1) {
                change_flg = 0;
                Lcd_Out(3, 5, sampling_str[sampling_mode]);
        }
        //
        if (SW_UP == 0) {
                while (SW_UP == 0) {
                        Delay_ms(100);
                }
                //
                sampling_mode++;
                if (sampling_mode == 4) {
                        sampling_mode = 3;
                }
                Lcd_Out(3, 5, sampling_str[sampling_mode]);
        }
        //
        if (SW_DOWN == 0) {
                while (SW_DOWN == 0) {
                        Delay_ms(100);
                }
                //
                sampling_mode--;
                if (sampling_mode == -1) {
                        sampling_mode = 0;
                }
                Lcd_Out(3, 5, sampling_str[sampling_mode]);
        }
        //
        EEPROM_Write(1, sampling_mode);
}
//*********************************************************************
***********
short   get_mode()
```

```
{
        if (SW_MODE == 1) {
                return (run_mode);
        }
        while (SW_MODE == 0) {
                Delay_ms(100);
        }
        //
        Lcd_Cmd(_LCD_CLEAR);
        //
        change_flg = 1;
        run_mode++;
        if (run_mode == 5) {
                run_mode = 0;
        }
        return (run_mode);
}
//*********************************************************************
***********
WORD    ADC_Get_Sample_Average(BYTE channel)
{
        int     cnt, i;
        DWORD   ad;
        //
        switch (sampling_mode) {
        case 0:
                i = 1;
                break;
        case 1:
                i = 10;
                break;
        case 2:
                i = 100;
                break;
        case 3:
                if (run_mode == 0) {
                        i = 100;
                } else {
                        i = 1000;
                }
                break;
        }
        ad = 0;
        for (cnt = 0; cnt < i; cnt++) {
                ad += ADC_Get_Sample(channel);
        }
        return (ad / i);
}
//*********************************************************************
***********
```

```
long    wait_trigger(short mode)
{
        long    clock;
        //
        switch (mode) {
        case 0: //100msec
                while (1) {
                        clock = get_clock();
                        if ((clock % 10) == 0) {
                                return(clock);
                        }
                        if (SW_STOP == 0) {
                                return(-1);
                        }
                }
                break;
        case 1: //1sec
                while (1) {
                        clock = get_clock();
                        if ((clock % 100) == 0) {
                                return(clock);
                        }
                        if (SW_STOP == 0) {
                                return(-1);
                        }
                }
                break;
        case 2: //1min
                while (1) {
                        clock = get_clock();
                        if ((clock % 6000) == 0) {
                                return(clock);
                        }
                        if (SW_STOP == 0) {
                                return(-1);
                        }
                }
                break;
        case 3: //10min
                while (1) {
                        clock = get_clock();
                        if ((clock % 60000) == 0) {
                                return(clock);
                        }
                        if (SW_STOP == 0) {
                                return(-1);
                        }
                }
                break;
        }
        return (-1);
```

```
}
//*****************************************************************************
***********
void    clock2str(DWORD clock, char *str) //"00:00:00:000"
{
        BYTE    hh, mm, ss, ms;
        char    tmp[4];
        //
        hh = (clock % 8640000) / 360000;
        mm = (clock % 360000) / 6000;
        ss = (clock % 6000) / 100;
        ms = clock % 100;
        //
        ByteToStr(hh, tmp);
        str[0] = (tmp[1] == ' ') ? '0' : tmp[1];
        str[1] = tmp[2];
        str[2] = ':';
        ByteToStr(mm, tmp);
        str[3] = (tmp[1] == ' ') ? '0' : tmp[1];
        str[4] = tmp[2];
        str[5] = ':';
        ByteToStr(ss, tmp);
        str[6] = (tmp[1] == ' ') ? '0' : tmp[1];
        str[7] = tmp[2];
        str[8] = ':';
        ByteToStr(ms, tmp);
        str[9] = (tmp[1] == ' ') ? '0' : tmp[1];
        str[10] = tmp[2];
        str[11] = '0';
        str[12] = 0x00;
}
//*****************************************************************************
***********
void    clock2timedate(DWORD clock, __TIME *tm)
{
        tm->Year   = 2012;
        tm->Month  = 12;
        tm->Day    = 27;
        tm->Hour   = (clock % 8640000) / 360000;
        tm->Minute = (clock % 360000) / 6000;
        tm->Second = (clock % 6000) / 100;
}
//*****************************************************************************
***********
void    clock2timedate2(__TIME *tm)
{
        tm->Year   = 2012;
        tm->Month  = RTC_8564_Read(7) & 0x1F;
        tm->Day    = RTC_8564_Read(5) & 0x3F;
        tm->Hour   = RTC_8564_Read(4) & 0x3F;
```

```
        tm->Minute = RTC_8564_Read(3) & 0x7F;
        tm->Second = RTC_8564_Read(2) & 0x7F;
}
//*****************************************************************************
***********
DWORD   clock = 0;
DWORD   clock_tmp = 0;
short   clock_flg = 0;
//
void    interrupt()
{
        if (PIR1.CCP1IF == 1) {
                PIR1.CCP1IF = 0;
                //
                clock++;
                switch (clock_flg) {
                case 1:
                        clock_flg = 0;
                        clock_tmp = clock;
                        break;
                case 2:
                        clock_flg = 0;
                        clock = clock_tmp;
                        break;
                }
        }
}
//*****************************************************************************
***********
DWORD   get_clock()
{
        clock_flg = 1;
        while (clock_flg == 1) {
        }
        return (clock_tmp);
}
//*****************************************************************************
***********
void    set_clock(DWORD clock)
{
        clock_tmp = clock;
        clock_flg = 2;
        while (clock_flg == 2) {
        }
}
//*****************************************************************************
***********
//■周期割り込み設定関数(10msec)
void    init_timer()
{
        // CCPの設定
```

```
        PIE1.CCP1IE = 1;
        PIR1.CCP1IF = 0;
        CCP1CON.CCP1M3 = 1;
        CCP1CON.CCP1M2 = 0;
        CCP1CON.CCP1M1 = 1;
        CCP1CON.CCP1M0 = 1;
        CCPR1L = 0x10;          // 10msec...クロックが32Mhzの時
        CCPR1H = 0x27;          //
10msec...(1÷32000000)*4*8*10000(0x2710)
        // TIMER1の設定
        PIE1.TMR1IE = 0;
        PIR1.TMR1IF = 0;
        TMR1L = 0;
        TMR1H = 0;
        T1CON.TMR1CS = 0;
        T1CON.T1CKPS0 = 1;
        T1CON.T1CKPS1 = 1;
        T1CON.TMR1ON = 1;
        //
        INTCON.PEIE = 1;
        INTCON.GIE = 1;
}
//*****************************************************************************
***********
//■□□□を初期化する関数です。
sfr  sbit  Mmc_Chip_Select            at  RC2_bit;
sfr  sbit  Mmc_Chip_Select_Direction  at  TRISC2_bit;
//
void    init_sdc_fat32()
{
        SPI1_Init_Advanced(_SPI_MASTER_OSC_DIV64,
_SPI_DATA_SAMPLE_MIDDLE, _SPI_CLK_IDLE_LOW, _SPI_LOW_2_HIGH);
        while (FAT32_Init() < 0) {
                Lcd_Out(2, 1, "SDC(FAT32) error!");
                Delay_ms(500);
                Lcd_Out(2, 1, "                ");
                Delay_ms(500);
        }
        SPI1_Init_Advanced(_SPI_MASTER_OSC_DIV4,
_SPI_DATA_SAMPLE_MIDDLE, _SPI_CLK_IDLE_LOW, _SPI_LOW_2_HIGH);
        //
        Lcd_Out(2, 1, "SDC(FAT32) complete!");
        Delay_ms(500);
        Lcd_Cmd(_LCD_CLEAR);
}
//*****************************************************************************
***********
void    init_osc()
{
        OSCCON.IRCF2 = 1;
```

```
        OSCCON.IRCF1 = 1;
        OSCCON.IRCF0 = 1;
        OSCTUNE.PLLEN = 1;
}
//**********************************************************************
***********
void    init_adc()
{
        ADCON1.PCFG3 = 0;
        ADCON1.PCFG2 = 0;
        ADCON1.PCFG1 = 0;
        ADCON1.PCFG0 = 0;
        ADCON1.VCFG1 = 0;
        ADCON1.VCFG0 = 0;
        //
        ADC_Init();
}
//**********************************************************************
***********
void    init_port()
{
        TRISA = 0b11101111;
        TRISB = 0b00010011;
        TRISC = 0b11111111;
        TRISD = 0b11111111;
        TRISE2_bit = 1;
        TRISE1_bit = 1;
        TRISE0_bit = 1;
        //
        LED = 0;
}
//**********************************************************************
***********
void    init_comparator()
{
        CMCON.CM2 = 1;
        CMCON.CM1 = 1;
        CMCON.CM0 = 1;
}
//**********************************************************************
***********
//■□□□初期関数
sbit    LCD_D7              at    RD7_bit;
sbit    LCD_D6              at    RD6_bit;
sbit    LCD_D5              at    RD5_bit;
sbit    LCD_D4              at    RD4_bit;
sbit    LCD_EN              at    RC7_bit;
sbit    LCD_RS              at    RC6_bit;
sbit    LCD_D7_Direction    at    TRISD7_bit;
sbit    LCD_D6_Direction    at    TRISD6_bit;
sbit    LCD_D5_Direction    at    TRISD5_bit;
```

```
sbit    LCD_D4_Direction    at    TRISD4_bit;
sbit    LCD_EN_Direction    at    TRISC7_bit;
sbit    LCD_RS_Direction    at    TRISC6_bit;
//
void    init_lcd()
{
        short   cnt;
        //
        Lcd_Init();
        RegistCustomChar();
        Lcd_Cmd(_LCD_CLEAR);
        Lcd_Cmd(_LCD_CURSOR_OFF);
        Lcd_Out(1, 1, tytle);
        for (cnt = 0; cnt <= 100; cnt++) {
                BarDisp(2, 1, cnt, 100);
                BarDisp(3, 1, cnt, 100);
                BarDisp(4, 1, cnt, 100);
                Delay_ms(10);
        }
        Lcd_Cmd(_LCD_CLEAR);
}
//**********************************************************************
***********
//■■■キャラクタ登録関数■■■
const char character0[] = { 0, 0, 0, 0, 0, 0, 0, 0};
const char character1[] = { 0,16,16,16,16,16, 0, 0};
const char character2[] = { 0,24,24,24,24,24, 0, 0};
const char character3[] = { 0,28,28,28,28,28, 0, 0};
const char character4[] = { 0,30,30,30,30,30, 0, 0};
const char character5[] = { 0,31,31,31,31,31, 0, 0};
//
void    RegistCustomChar()
{
        char    i;
        //
        LCD_Cmd(64);
        for (i = 0; i<=7; i++) {
                LCD_Chr_Cp(character0[i]);
        }
        for (i = 0; i<=7; i++) {
                LCD_Chr_Cp(character1[i]);
        }
        for (i = 0; i<=7; i++) {
                LCD_Chr_Cp(character2[i]);
        }
        for (i = 0; i<=7; i++) {
                LCD_Chr_Cp(character3[i]);
        }
        for (i = 0; i<=7; i++) {
                LCD_Chr_Cp(character4[i]);
        }
```

```
        for (i = 0; i<=7; i++) {
                LCD_Chr_Cp(character5[i]);
        }
        LCD_Cmd(_LCD_RETURN_HOME);
}
//*****************************************************************************
***********
//■■■バー表示関数■■■
void    BarDisp(char row, char column, short dat, short bar)
{
        short   j, k, cnt;
        //
        j = dat / 5;
        k = dat - (j * 5);
        //
        switch (row) {
        case 1:
                Lcd_Cmd(_LCD_FIRST_ROW);
                break;
        case 2:
                Lcd_Cmd(_LCD_SECOND_ROW);
                break;
        case 3:
                Lcd_Cmd(_LCD_THIRD_ROW);
                break;
        case 4:
                Lcd_Cmd(_LCD_FOURTH_ROW);
                break;
        }
        //
        for (cnt = 1; cnt < column; cnt++) {
                Lcd_Cmd(_LCD_MOVE_CURSOR_RIGHT);
        }
        //
        for (cnt = 0; cnt < j; cnt++) {
                Lcd_Chr_Cp(5);
        }
        Lcd_Chr_Cp(k);
        for (cnt++; cnt < (bar / 5); cnt++) {
                Lcd_Chr_Cp(' ');
        }
}
//*****************************************************************************
***********
#define ACK     1
#define NO_ACK  0
//■□□□からのデータ取得関数です。
BYTE    RTC_8564_Read(BYTE addr)
{
        BYTE   dat;
        //
```

```
        Soft_I2C_Start();
        Soft_I2C_Write(0xA2);
        Soft_I2C_Write(addr);
        Soft_I2C_Start();
        Soft_I2C_Write(0xA3);
        dat = Soft_I2C_Read(NO_ACK);
        Soft_I2C_Stop();
        //
        return (dat);
}
//*****************************************************************************
***********
//■□□□へのデータ書き込み関数です。
void    RTC_8564_Write(BYTE addr, BYTE dat)
{
        Soft_I2C_Start();
        Soft_I2C_Write(0xA2);
        Soft_I2C_Write(addr);
        Soft_I2C_Write(dat);
        Soft_I2C_Stop();
}
//*****************************************************************************
***********
//■□□□から日付時刻（月日時分秒）を取得する関数です。
//設定文字列のフォーマット→"12/27 12:34:56"
void    get_date_time(char * dat)
{
        BYTE    td_month, td_day, td_hour, td_min, td_sec;
        //日付時刻（月日時分秒）の□□□からの取得
        td_month = RTC_8564_Read(7) & 0x1F;
        td_day = RTC_8564_Read(5) & 0x3F;
        td_hour = RTC_8564_Read(4) & 0x3F;
        td_min = RTC_8564_Read(3) & 0x7F;
        td_sec = RTC_8564_Read(2) & 0x7F;
        //日付時刻（月日時分秒）の□□□への書き込み
        dat[0] = (td_month >> 4) + '0';
        dat[1] = (td_month & 0x0F) + '0';
        dat[2] = '/';
        dat[3] = (td_day >> 4) + '0';
        dat[4] = (td_day & 0x0F) + '0';
        dat[5] = ' ';
        dat[6] = (td_hour >> 4) + '0';
        dat[7] = (td_hour & 0x0F) + '0';
        dat[8] = ':';
        dat[9] = (td_min >> 4) + '0';
        dat[10] = (td_min & 0x0F) + '0';
        dat[11] = ':';
        dat[12] = (td_sec >> 4) + '0';
        dat[13] = (td_sec & 0x0F) + '0';
        dat[14] = 0x00;
}
```

```
//*****************************************************************************
***********
//■□□□日付時刻（月日時分秒）設定関数
//設定文字列のフォーマット→"12/27 12:34:56"
void    set_date_time(char *dat)
{
        //日付時刻（月日時分秒）の□□□への書き込み
        td_month = ((dat[0] - '0') << 4) + (dat[1] - '0');
        td_day   = ((dat[3] - '0') << 4) + (dat[4] - '0');
        td_hour  = ((dat[6] - '0') << 4) + (dat[7] - '0');
        td_min   = ((dat[9] - '0') << 4) + (dat[10] - '0');
        td_sec   = ((dat[12] - '0') << 4) + (dat[13] - '0');
        //
        RTC_8564_Write(7, td_month);      //月の書き込み
        RTC_8564_Write(5, td_day);        //日の書き込み
        RTC_8564_Write(4, td_hour);       //時の書き込み
        RTC_8564_Write(3, td_min);        //分の書き込み
        RTC_8564_Write(2, td_sec);        //秒の書き込み
}
//*****************************************************************************
***********
//■□□□を初期化する関数です。
sbit    Soft_I2C_Scl            at  RD3_bit;
sbit    Soft_I2C_Sda            at  RD2_bit;
sbit    Soft_I2C_Scl_Direction  at  TRISD3_bit;
sbit    Soft_I2C_Sda_Direction  at  TRISD2_bit;
//
void    init_rtc()
{
        BYTE    td;
        //
        Soft_I2C_Init();
        //書き込みと読み出しのテストをします。
        td = Dec2Bcd(10);  //2011
        RTC_8564_Write(8, td);   //年の書き込み
        td = RTC_8564_Read(8);   //年の読み出し
        if (Bcd2Dec(td) != 10) {
                while (1) {
                        Lcd_Out(2, 1, "RTC error!");
                        Delay_ms(500);
                        Lcd_Out(2, 1, "          ");
                        Delay_ms(500);
                }
        }
        //
        Lcd_Out(2, 1, "RTC complete!");
        Delay_ms(500);
        Lcd_Cmd(_LCD_CLEAR);
}
//*****************************************************************************
***********
```

FAT32ヘッダファイル

__Lib_FAT32.h

```
#ifdef __MIKROC_PRO_FOR_PIC32__
  #define __PIC32__
#endif

#ifdef __MIKROC_PRO_FOR_PIC__
  #define __PIC__
#endif

#ifdef __MIKROC_PRO_FOR_DSPIC__
  #define __dsPIC__
#endif

#ifdef __MIKROC_PRO_FOR_AVR__
  #define __AVR__
#endif

#ifdef __MIKROC_PRO_FOR_ARM__
  #define __ARM__
#endif

#define CR 0x0D // carrige return
#define LF 0x0A // line feed

////////////////////////////////////////////////////////////////////////
/////////
typedef unsigned short      uint8;
typedef   signed short      int8;
typedef unsigned int        uint16;
typedef   signed int        int16;
typedef unsigned long       uint32;
typedef   signed long       int32;
typedef unsigned long long  uint64;
typedef   signed long long  int64;

static const uint16 SECTOR_SIZE = 512;

////////////////////////////////////////////////////////////////////////
/////////
//
//  modes of file operation:
//      - read   ( resets cursor to 0 )
//      - write  ( resets cursor to 0 )
//      - append ( leaves cursor as is )
//
////////////////////////////////////////////////////////////////////////
/////////
```

```
static const uint8
    FILE_READ       = 0x01,
    FILE_WRITE      = 0x02,
    FILE_APPEND     = 0x04;

//////////////////////////////////////////////////////////////////////////
////////
//
//  status and error codes returned by functions
//
//////////////////////////////////////////////////////////////////////////
////////
static  const int8
    ////////////////////////
    //  general status
    ////////////////////////
    OK                  =   0,
    ERROR               =  -1,
    FOUND               =   1,
    ////////////////////////
    //  system errors
    ////////////////////////
    E_READ              =  -1,
    E_WRITE             =  -2,
    E_INIT_CARD         =  -3,
    E_BOOT_SIGN         =  -4,
    E_BOOT_REC          =  -5,
    E_FILE_SYS_INFO     =  -6,
    E_DEVICE_SIZE       =  -7,
    ////////////////////////
    //  space related errors
    ////////////////////////
    E_LAST_ENTRY        = -10,
    E_FREE_ENTRY        = -11,
    E_CLUST_NUM         = -12,
    E_NO_SWAP_SPACE     = -13,
    E_NO_SPACE          = -14,
    ////////////////////////
    //  dir related errors
    ////////////////////////
    E_DIR_NAME          = -20,
    E_ISNT_DIR          = -21,
    E_DIR_EXISTS        = -22,
    E_DIR_NOTFOUND      = -23,
    E_DIR_NOTEMPTY      = -24,
    ////////////////////////
    //  file related errors
    ////////////////////////
    E_FILE_NAME         = -30,
    E_ISNT_FILE         = -31,
    E_FILE_EXISTS       = -32,
```

```
    E_FILE_NOTFOUND     = -33,
    E_FILE_NOTEMPTY     = -34,
    E_MAX_FILES         = -35,
    E_FILE_NOTOPENED    = -36,
    E_FILE_EOF          = -37,
    E_FILE_READ         = -38,
    E_FILE_WRITE        = -39,
    E_FILE_HANDLE       = -40,
    ////////////////////////
    //  time related errors
    ////////////////////////
    E_TIME_YEAR         = -50,
    E_TIME_MONTH        = -51,
    E_TIME_DAY          = -52,
    E_TIME_HOUR         = -53,
    E_TIME_MINUTE       = -54,
    E_TIME_SECOND       = -55;

////////////////////////////////////////////////////////////////////////
/////////
// partition entry in MBR, 16B size
typedef struct
{
    uint8       State[1];       // partition state. ACTIVE=0x80,
INACTIVE=0x00
    uint8       __1[3];
    uint8       Type[1];        // partition type.  FAT16=0x06,
FAT32=0x0B
    uint8       __2[3];
    uint8       Boot[4];        // boot record sector number of
partition
    uint8       Size[4];        // partition size in sectors
}
FAT32_PART;

////////////////////////////////////////////////////////////////////////
/////////
// MBR, 512B size
typedef struct
{
    uint8       __1[446];
    FAT32_PART  Part[4];
    uint8       BootSign[2];       // boot signature
}
FAT32_MBR;

////////////////////////////////////////////////////////////////////////
/////////
// Boot Record, 512B size
typedef struct
```

```
{
    uint8       JmpCode[3];         // only way to say for certain if
it is boot sector
    uint8       __1[8];
    uint8       BytesPSect[2];      // # of bytes per sector
    uint8       SectsPClust[1];     // # of sectors per cluster
    uint8       Reserved[2];        // # of reserved sectors
    uint8       FATCopies[1];       // # of FAT copies
    uint8       __2[4];
    uint8       MediaDesc[1];       // media descriptor
    uint8       __3[10];
    uint8       Sects[4];           // # of sectors in partition
    uint8       SectsPFAT[4];       // # of sectors per FAT table
    uint8       Flags[2];           // flags
    uint8       __4[2];
    uint8       RootClust[4];       // number of root dir cluster
    uint8       FSISect[2];         // FSI sector (offset from boot)
    uint8       BootBackup[2];      // sector number of boot sector
backup
    uint8       __5[14];
    uint8       ExtSign[1];         // extended signature - 0x29
    uint8       __6[4];
    uint8       VolName[11];        // volume name
    uint8       FATName[8];         // FAT name (FAT32)
    uint8       __7[420];
    uint8       BootSign[2];        // boot signature
}
FAT32_BR;

////////////////////////////////////////////////////////////////////////
/////////
// File System Info, 512B size
typedef struct
{
    uint8       FirstSign[4];
    uint8       __1[480];
    uint8       FSISign[4];
    uint8       FreeSects[4];
    uint8       AllocClust[4];
    uint8       __2[14];
    uint8       BootSign[2];
}
FAT32_FSI;

////////////////////////////////////////////////////////////////////////
/////////
typedef struct
{
    uint8       Entry[4];
}
FAT32_FATENT;
```

```
////////////////////////////////////////////////////////////////////////
/////////
// directory entry, 32B size
typedef struct
{
    uint8       NameExt[11];    // file/directory name
    uint8       Attrib[1];      // file/directory attribute
    uint8       __1[2];
    uint8       CTime[2];       // create time
    uint8       CDate[2];       // create date
    uint8       ATime[2];
    uint8       HiClust[2];     //  <-------------------|
    uint8       MTime[2];       //  modification time   |
    uint8       MDate[2];       //  modification date   |
    uint8       LoClust[2];     //  <-------------------|----- 1st
cluster of file/folder
    uint8       Size[4];        // file size
}
FAT32_DIRENT;
////////////////////////////////////////////////////////////////////////
/////////
////////////////////////////////////////////////////////////////////////
/////////

typedef uint32  __CLUSTER;
typedef uint32  __SECTOR;
typedef uint32  __ENTRY;

typedef int8    __HANDLE;

////////////////////////////////////////////////////////////////////////
/////////
// Time structure
typedef struct
{
    uint16      Year;
    uint8       Month;
    uint8       Day;
    uint8       Hour;
    uint8       Minute;
    uint8       Second;
}
__TIME;

////////////////////////////////////////////////////////////////////////
/////////
// Partition info
typedef struct
{
```

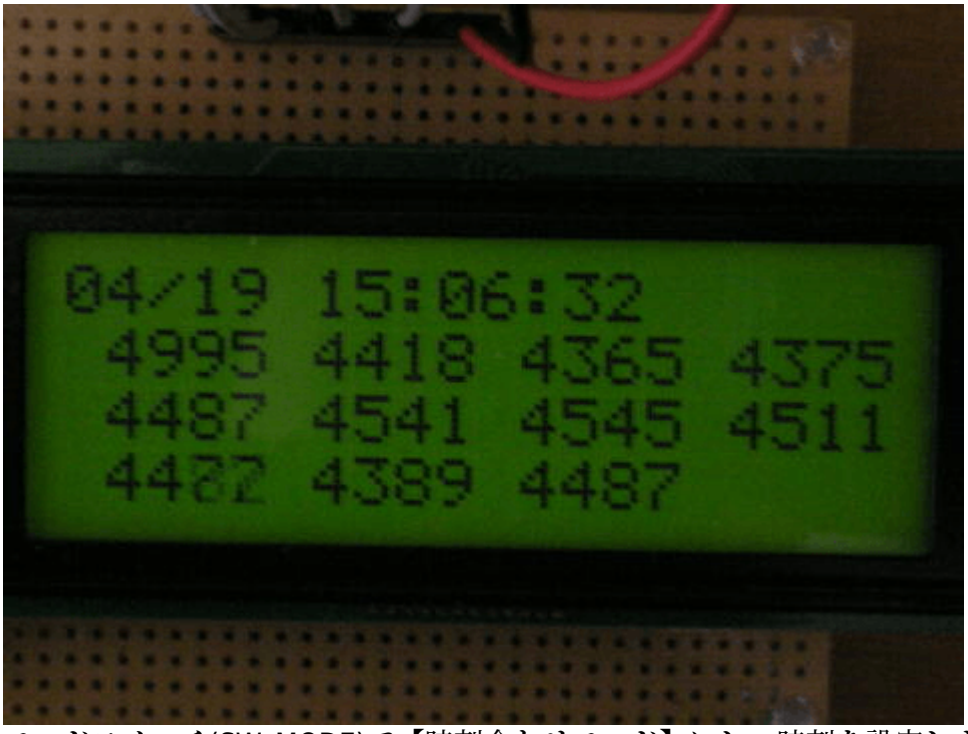
```
    uint8       State;  // partition status
    uint8       Type;   // partition type
    __SECTOR    Boot;   // partition start sector
    uint32      Size;   // partition size
}
__PART;

////////////////////////////////////////////////////////////////////////
/////////
// FS info
typedef struct
{
    __PART      Part[1];
    uint16      BytesPSect;  // bytes in sector
    uint8       SectsPClust; // sectors in one cluster
    uint16      Reserved;
    uint8       FATCopies;   // number of FAT table copies
    uint32      SectsPFAT;   // number of sectors in one FAT table
    uint16      Flags;
    __SECTOR    FAT;        // sector # of first FAT table
    __CLUSTER   Root;       // cluster # of root directory
    __SECTOR    FSI;        // sector # of FSI sector (relative to boot
sector)
    __SECTOR    Data;       // sector # of Data area
}
__INFO;

////////////////////////////////////////////////////////////////////////
/////////
typedef struct
{
    char        NameExt[13];  // file/directory name
    uint8       Attrib;       // file/directory attribute

    uint32      Size;         // file/directory size
    __CLUSTER   _1stClust;    // file/directory start cluster

    __CLUSTER   EntryClust;   // file/directory entry cluster
    __ENTRY     Entry;        // file/derectory entry index in entry
cluster
}
__DIR;

////////////////////////////////////////////////////////////////////////
/////////
typedef struct
{
    __CLUSTER   _1stClust;    // file start cluster
    __CLUSTER   CurrClust;    // current file cluster

    __CLUSTER   EntryClust;   // directory entry cluster
```

```
    __ENTRY     Entry;        // derectory entry number in the entry
cluster

    uint32      Cursor;       // current file position (carret)
    uint32      Length;       // file size

    uint8       Mode;         // file open mode
}
__FILE;

/*
 * buffer for mmc/sd card sector r/w handling
 */
typedef struct
{
    __SECTOR        fSectNum;
    char            fSect[SECTOR_SIZE]; // sector buffer
}
__RAW_SECTOR;

////////////////////////////////////////////////////////////////////////
/////////
extern const char   CRLF_F32[];
extern const uint8  FAT32_MAX_FILES;
extern const uint8  f32_fsi_template[SECTOR_SIZE];
extern const uint8  f32_br_template[SECTOR_SIZE];
extern       __FILE fat32_fdesc[];
extern       __RAW_SECTOR f32_sector;

////////////////////////////////////////////////////////////////////////
/////////
extern  int8 FAT32_Dev_Init         (void);
extern  int8 FAT32_Dev_Read_Sector  (__SECTOR sc, char* buf);
extern  int8 FAT32_Dev_Write_Sector (__SECTOR sc, char* buf);
extern  int8 FAT32_Put_Char         (char ch);

////////////////////////////////////////////////////////////////////////
/////////
int8    FAT32_Init      (void);
int8    FAT32_Format    (char *devLabel);

int8    FAT32_ReadDir   (__DIR *pDE);
int8    FAT32_ChangeDir (char *dname);
int8    FAT32_MakeDir   (char *dname);
int8    FAT32_Dir       (void);
int8    FAT32_Delete    (char *fn);
int8    FAT32_DeleteRec (char *fn);
int8    FAT32_Exists    (char *name);
int8    FAT32_Rename    (char *oldName, char *newName);
int8    FAT32_Open      (char *fn, uint8 mode);
```

```
int8    FAT32_Eof       (__HANDLE fHandle);
int8    FAT32_Read      (__HANDLE fHandle, char* rdBuf, uint16 len);
int8    FAT32_Write     (__HANDLE fHandle, char* wrBuf, uint16 len);
int8    FAT32_Seek      (__HANDLE fHandle, uint32 pos);
int8    FAT32_Tell      (__HANDLE fHandle, uint32 *pPos);
int8    FAT32_Close     (__HANDLE fHandle);
int8    FAT32_Size      (char *fname, uint32 *pSize);
int8    FAT32_MakeSwap  (char *name, __SECTOR nSc, __CLUSTER *pCl);

int8    FAT32_SetTime  (__TIME *pTM);
int8    FAT32_IncTime  (uint32 Sec);

int8    FAT32_GetCTime  (char *fname, __TIME *pTM);
int8    FAT32_GetMTime  (char *fname, __TIME *pTM);

int8    FAT32_GetError  (void);
////////////////////////////////////////////////////////////////////////
////////
```

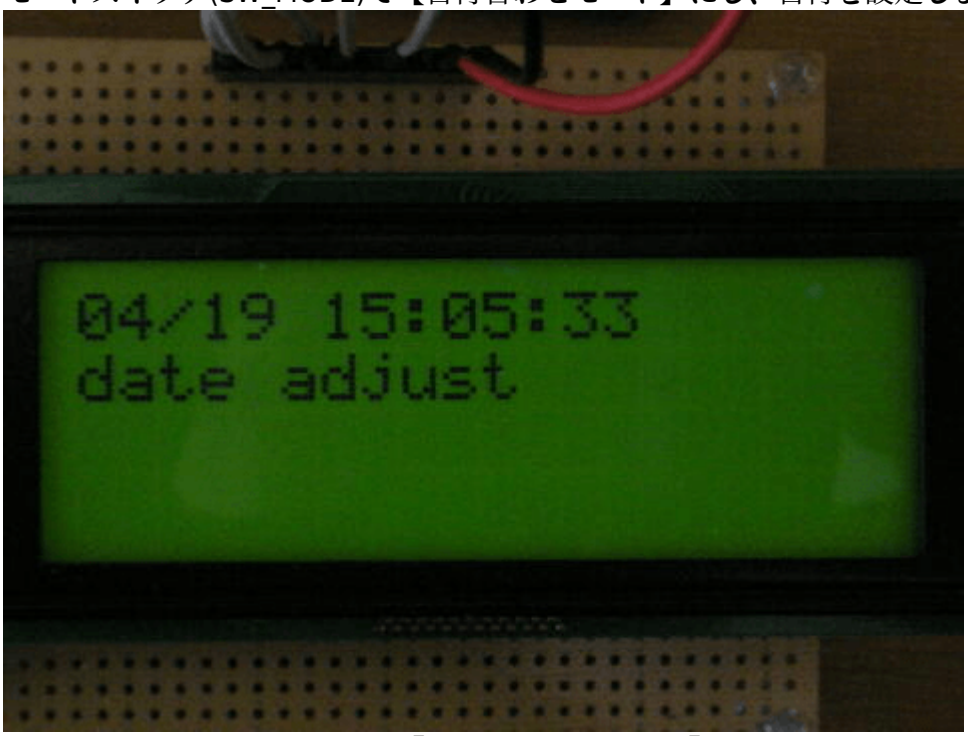
動作確認

1. SDカード(FAT32)をカードスロットに挿入し、電源を入れます。
2. LEDが1秒周期で3回点滅するのを確認します。
3. 日付時刻およびCH1~CH11の電圧値がLCDに表示されます。【測定、表示、記録モード】

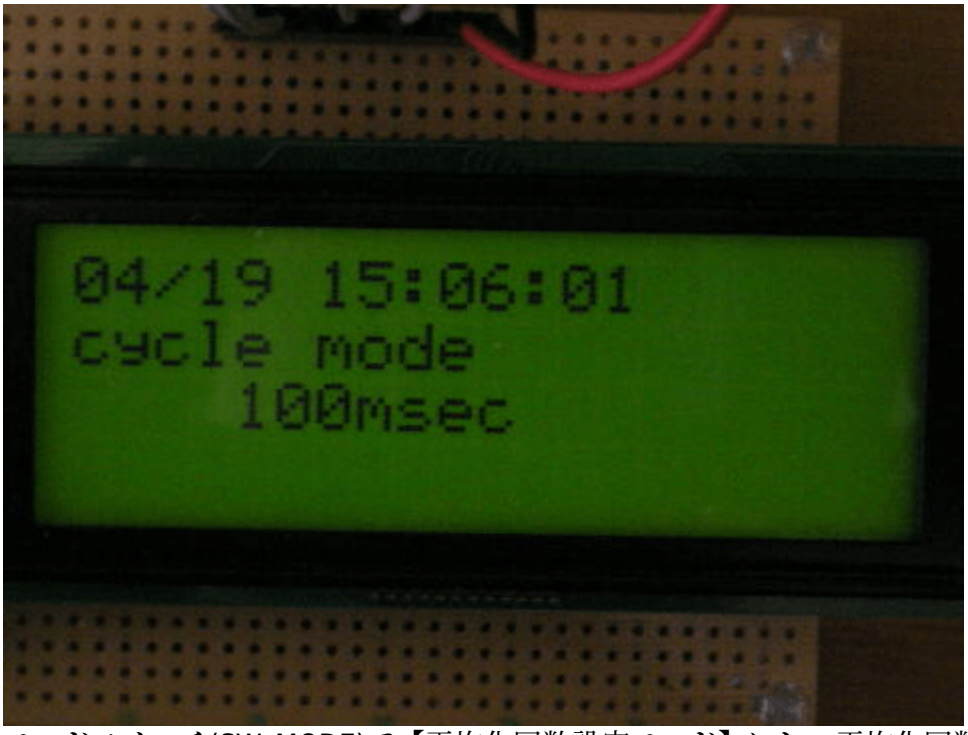


4. モードスイッチ(SW_MODE)で【時刻合わせモード】にし、時刻を設定します。

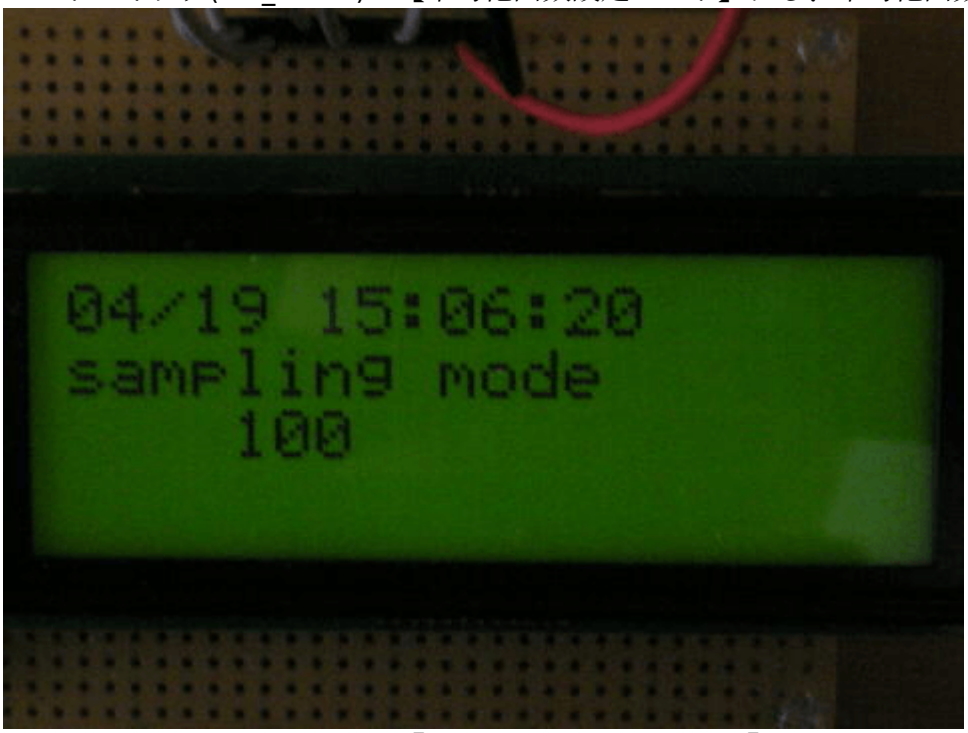
5. **モードスイッチ**(SW_MODE)**で【**日付合**わせモード】にし、**日付**を**設定**します。**



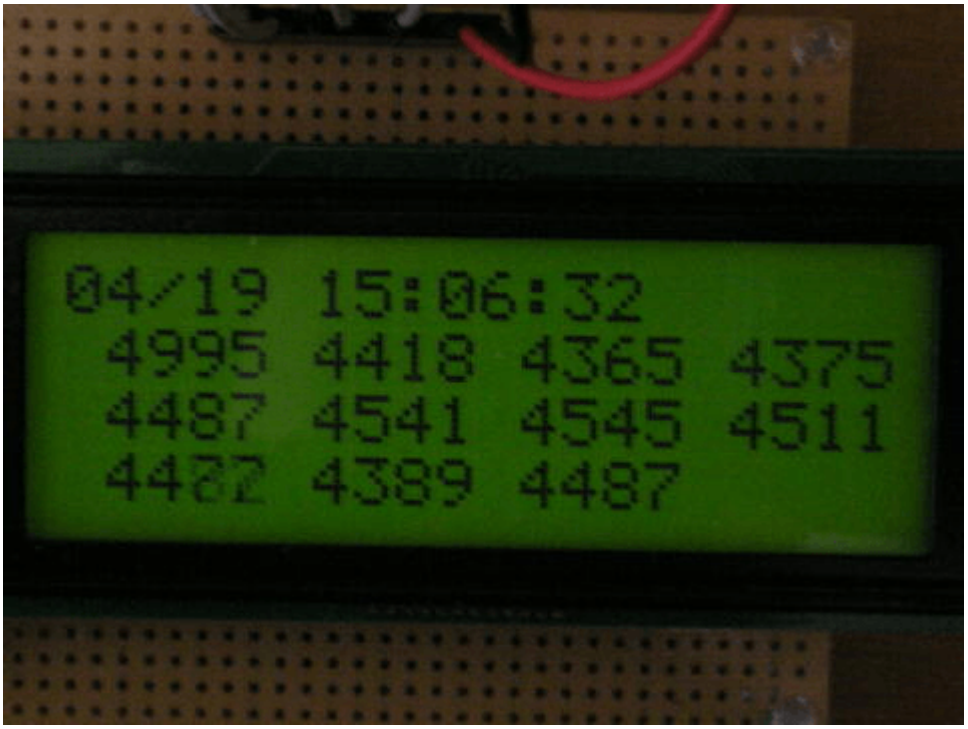
6. **モードスイッチ**(SW_MODE)**で【**測定周期設定**モード】にし、**測定周期**を**設定**します。**



7. **モードスイッチ**(SW_MODE)**で【**平均化回数設定**モード】にし、**平均化回数**を設定します。**

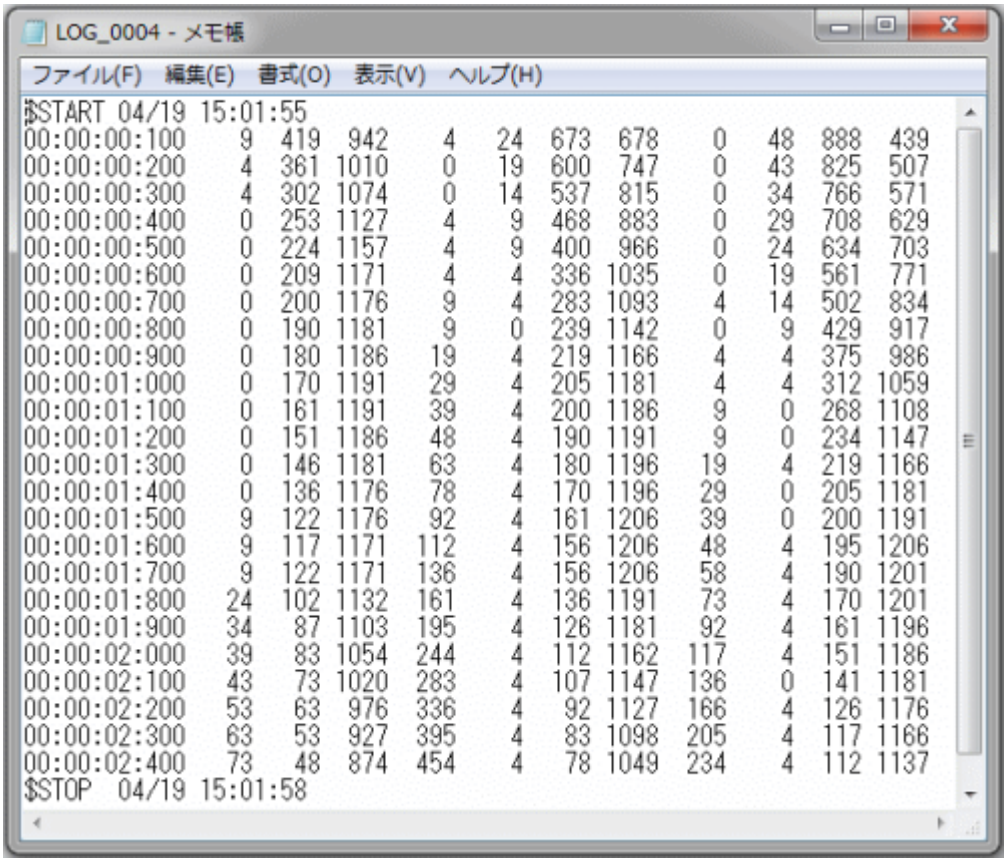


8. **モードスイッチ**(SW_MODE)**で【**測定、表示、記録**モード】にし、スタートスイッチ**(SW_START)**を押下します**LED**が点灯します。**



9. **ストップスイッチ**(SW_STOP)を押下します。LEDが消灯します。
10. 電源を切ります。SDカードを抜いて、パソコンのカードスロットに挿入します。
11. エクスプローラで記録ファイルが存在することを確認します。

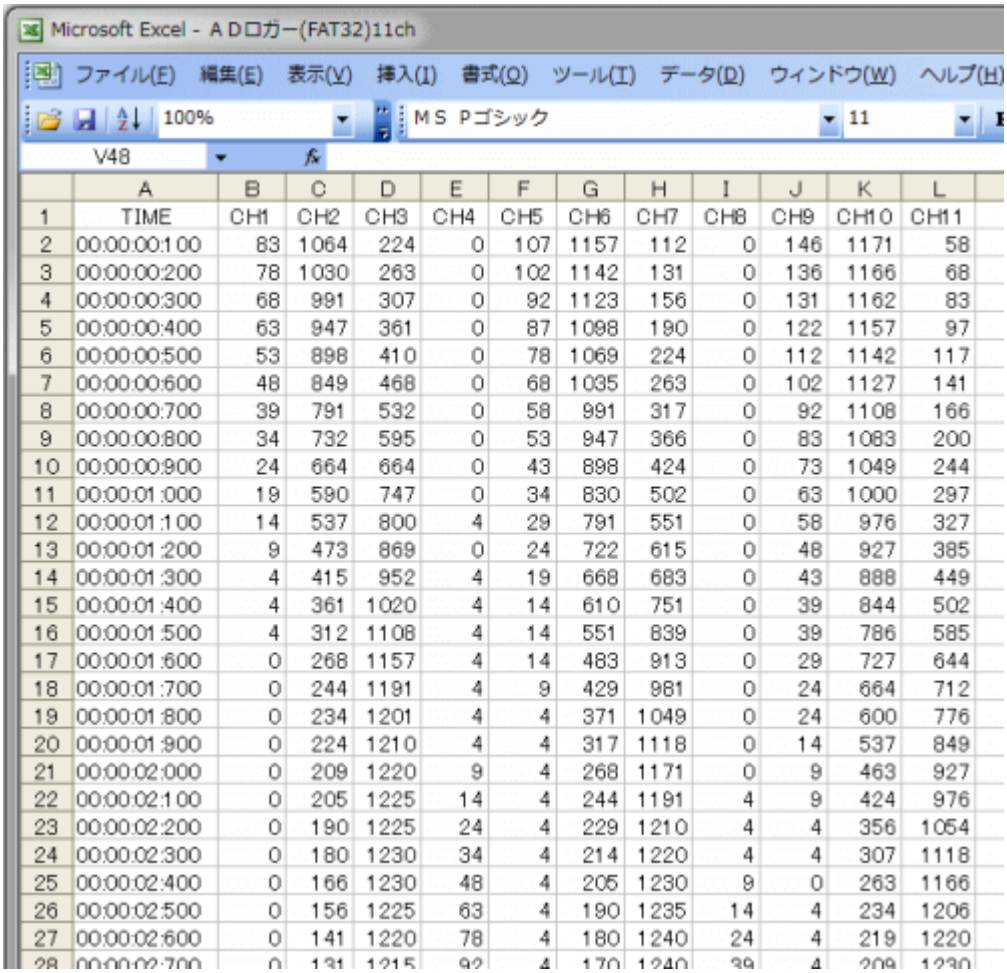
12. ファイルの内容を確認します。


LOG_0004 - メモ帳

ファイル(F) 編集(E) 書式(O) 表示(V) ヘルプ(H)

```
$START 04/19 15:01:55
00:00:00:100     9  419  942    4   24  673  678    0   48  888  439
00:00:00:200     4  361 1010    0   19  600  747    0   43  825  507
00:00:00:300     4  302 1074    0   14  537  815    0   34  766  571
00:00:00:400     0  253 1127    4    9  468  883    0   29  708  629
00:00:00:500     0  224 1157    4    9  400  966    0   24  634  703
00:00:00:600     0  209 1171    4    4  336 1035    0   19  561  771
00:00:00:700     0  200 1176    9    4  283 1093    4   14  502  834
00:00:00:800     0  190 1181    9    0  239 1142    0    9  429  917
00:00:00:900     0  180 1186   19    4  219 1166    4    4  375  986
00:00:01:000     0  170 1191   29    4  205 1181    4    4  312 1059
00:00:01:100     0  161 1191   39    4  200 1186    9    0  268 1108
00:00:01:200     0  151 1186   48    4  190 1191    9    0  234 1147
00:00:01:300     0  146 1181   63    4  180 1196   19    4  219 1166
00:00:01:400     0  136 1176   78    4  170 1196   29    0  205 1181
00:00:01:500     9  122 1176   92    4  161 1206   39    0  200 1191
00:00:01:600     9  117 1171  112    4  156 1206   48    4  195 1206
00:00:01:700     9  122 1171  136    4  156 1206   58    4  190 1201
00:00:01:800    24  102 1132  161    4  136 1191   73    4  170 1201
00:00:01:900    34   87 1103  195    4  126 1181   92    4  161 1196
00:00:02:000    39   83 1054  244    4  112 1162  117    4  151 1186
00:00:02:100    43   73 1020  283    4  107 1147  136    0  141 1181
00:00:02:200    53   63  976  336    4   92 1127  166    4  126 1176
00:00:02:300    63   53  927  395    4   83 1098  205    4  117 1166
00:00:02:400    73   48  874  454    4   78 1049  234    4  112 1137
$STOP  04/19 15:01:58
```

13. **ファイルの**内容を、Excelにコピー&ペーストします。


Microsoft Excel - ADロガー(FAT32)11ch

ファイル(F) 編集(E) 表示(V) 挿入(I) 書式(O) ツール(T) データ(D) ウィンドウ(W) ヘルプ(H)

100% MS Pゴシック 11

V48

	A	B	C	D	E	F	G	H	I	J	K	L
1	TIME	CH1	CH2	CH3	CH4	CH5	CH6	CH7	CH8	CH9	CH10	CH11
2	00:00:00:100	83	1064	224	0	107	1157	112	0	146	1171	58
3	00:00:00:200	78	1030	263	0	102	1142	131	0	136	1166	68
4	00:00:00:300	68	991	307	0	92	1123	156	0	131	1162	83
5	00:00:00:400	63	947	361	0	87	1098	190	0	122	1157	97
6	00:00:00:500	53	898	410	0	78	1069	224	0	112	1142	117
7	00:00:00:600	48	849	468	0	68	1035	263	0	102	1127	141
8	00:00:00:700	39	791	532	0	58	991	317	0	92	1108	166
9	00:00:00:800	34	732	595	0	53	947	366	0	83	1083	200
10	00:00:00:900	24	664	664	0	43	898	424	0	73	1049	244
11	00:00:01:000	19	590	747	0	34	830	502	0	63	1000	297
12	00:00:01:100	14	537	800	4	29	791	551	0	58	976	327
13	00:00:01:200	9	473	869	0	24	722	615	0	48	927	385
14	00:00:01:300	4	415	952	4	19	668	683	0	43	888	449
15	00:00:01:400	4	361	1020	4	14	610	751	0	39	844	502
16	00:00:01:500	4	312	1108	4	14	551	839	0	39	786	585
17	00:00:01:600	0	268	1157	4	14	483	913	0	29	727	644
18	00:00:01:700	0	244	1191	4	9	429	981	0	24	664	712
19	00:00:01:800	0	234	1201	4	4	371	1049	0	24	600	776
20	00:00:01:900	0	224	1210	4	4	317	1118	0	14	537	849
21	00:00:02:000	0	209	1220	9	4	268	1171	0	9	463	927
22	00:00:02:100	0	205	1225	14	4	244	1191	4	9	424	976
23	00:00:02:200	0	190	1225	24	4	229	1210	4	4	356	1054
24	00:00:02:300	0	180	1230	34	4	214	1220	4	4	307	1118
25	00:00:02:400	0	166	1230	48	4	205	1230	9	0	263	1166
26	00:00:02:500	0	156	1225	63	4	190	1235	14	4	234	1206
27	00:00:02:600	0	141	1220	78	4	180	1240	24	4	219	1220
28	00:00:02:700	0	131	1215	92	4	170	1240	39	4	209	1230

14. 内容**をグラフ**表示**させます**。