簡易ADロガ□(FAT32対応)(PIC18F2523)

概要

今迄に、SDカード(FAT16)を利用したロガーやタイマーを各種製作してきましたが、最大容量が2GBの制約がありました。しかし、使用用途によってはファイル容量をもっと増やしたいという要望がありました。

SDカード<FAT16>(例)

この程、マイクロエレクトロニカ(mikroElektronika)社より□FAT32のライブラリが提供されたので、早速適用してみました□FAT32は最大で2TBと十分に大きいので上記のような問題は発生しないのではと思っています。

SDカード<FAT32>(例)

◎FAT32ライブラリ

- 1333712413_fat32_library_mikroc_pic.mpkg

Dependancy Routines

- FAT32_Dev_Init
- FAT32_Dev_Read_Sector
- FAT32_Dev_Write_Sector
- FAT32_Dev_SectorCount
- FAT32_Dev_Multi_Read_Start
- FAT32_Dev_Multi_Read_Stop
- FAT32_Dev_Multi_Read_Sector
- FAT32_Put_Char

Library Routines

- FAT32_Init
- FAT32_Format
- FAT32_ScanDisk
- FAT32_GetFreeSpace
- FAT32_Dir
- FAT32_MakeDir
- FAT32_ChangeDir
- FAT32_FindFirst
- FAT32_FindNext
- FAT32_Exists
- FAT32_Rename
- FAT32_Delete
- FAT32_DeleteRec
- FAT32_Open
- FAT32_Close
- FAT32_Read
- FAT32_Write
- FAT32_Tell
- FAT32_Seek
- FAT32_Eof
- FAT32_Size
- FAT32_SetAttr
- FAT32_GetAttr
- FAT32_GetFileHandle
- FAT32_SetTime
- FAT32_IncTime
- FAT32_GetCTime
- FAT32_GetMTime
- FAT32_GetError
- FAT32_MakeSwap

◎パッケージマネージャ(ライブラリを[mikroC PRO]に登録するためのツール)

- package-manager

仕様

◎記録媒体

- SDカード(FAT32)

◎入力チャネル

- 8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

◎測定分解能

- A/D変換12ビット(4096)、基準電圧=3.3V
- 約0.8mV(3.3V÷4096)

◎入力電圧範囲

- 0V~3.3V(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)

動作原理(ハードウェア)

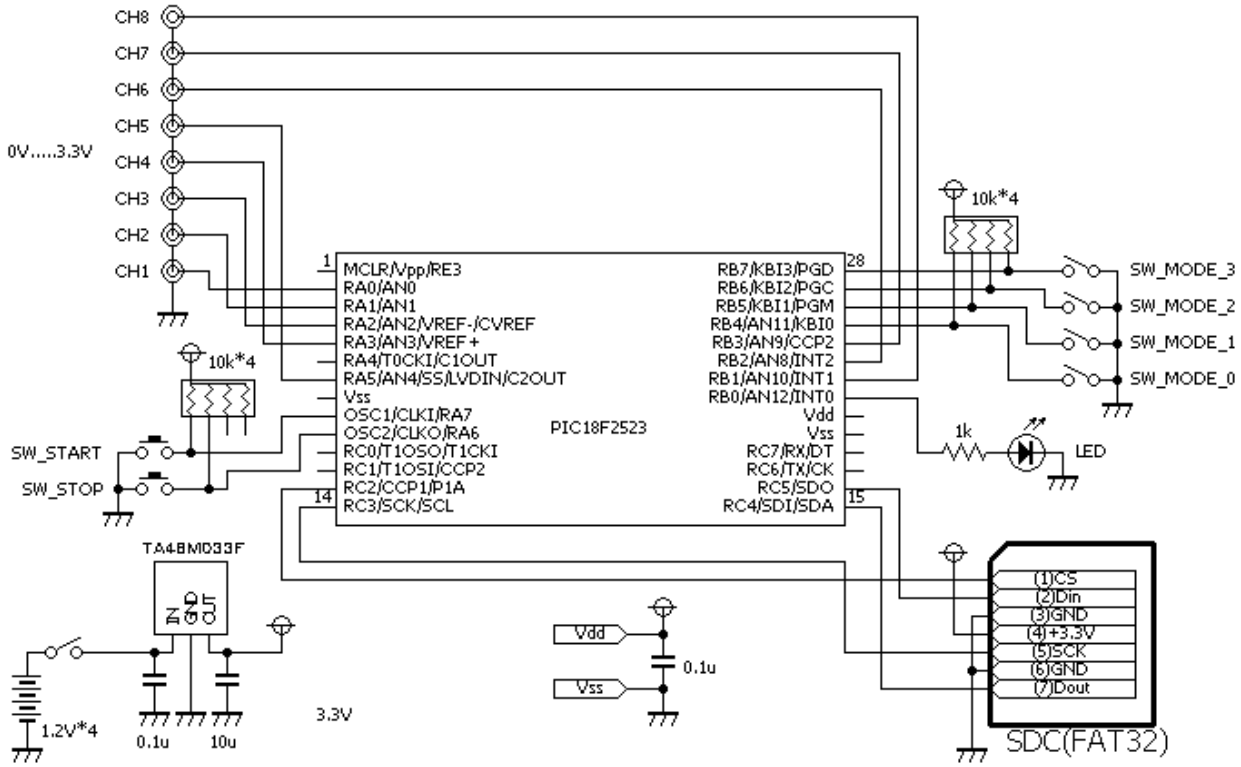
◎SDカードの電圧は、3.3Vなので全体を3.3V駆動とします。 ◎電源は、1.2Vの充電池を4本使用し、4.8Vとします□ ◎4.8Vを3.3Vの三端子レギュレータで3.3Vに降圧します。 ◎アナログデータは直接PICに入力するので入力電圧範囲は、0V~3.3Vとします。 ◎分圧抵抗を付加することにより入力電圧範囲を拡張することができます。

動作原理(ソフトウェア)

◎事前に、[mikroC PRO]に、FAT32ライブラリを登録します□ ◎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()□ ◎タイマ□(10msecの周期割り込み用)を設定します□□init_timer()□ ◎LEDを3回点滅させます。 ◎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□□measurement()□ ◎ファイルをクローズします。

回路図

＜簡易ＡＤロガー(FAT32)V2＞



■入力チャネル
- 8チャネル

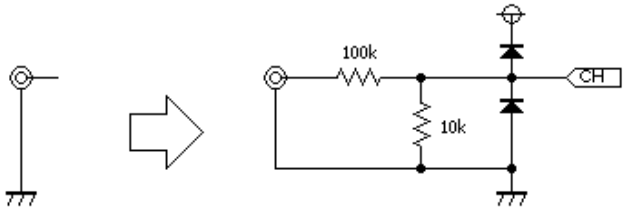
■測定周期
- 100msec -> SW_MODE_1=ON SW_MODE_0=ON
- 1sec -> SW_MODE_1=ON SW_MODE_0=OFF
- 1min -> SW_MODE_1=OFF SW_MODE_0=ON
- 10min -> SW_MODE_1=OFF SW_MODE_0=OFF

■平均測定回数
- 1回 -> SW_MODE_3=ON SW_MODE_2=ON
- 10回 -> SW_MODE_3=ON SW_MODE_2=OFF
- 100回 -> SW_MODE_3=OFF SW_MODE_2=ON
- 100回 -> SW_MODE_3=OFF SW_MODE_2=OFF

■測定分解能
- 約0.8mV

■入力電圧範囲
- 0V～3.3V

■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～36.3V)

ソースコード

simple_ad_logger_fat32_v2.c

```
//*********************************************************************************
***********
/*
    ＜簡易□□ロガ□(FAT32)□□□
  ・アナログ入力８チャネル
□□□□□(FAT32)
*/
//*********************************************************************************
***********
#define BYTE    unsigned        short
#define WORD    unsigned        int
#define DWORD   unsigned        long
//
#define FAT32
//
#ifdef  FAT32
#include "__Lib_FAT32.h"
#endif
//
sbit    SW_START        at      RA7_bit;
sbit    SW_STOP         at      RA6_bit;
//
sbit    SW_MODE_0       at      RB4_bit;
sbit    SW_MODE_1       at      RB5_bit;
sbit    SW_MODE_2       at      RB6_bit;
sbit    SW_MODE_3       at      RB7_bit;
//
sbit    LED             at      RB0_bit;
//*********************************************************************************
***********
extern  void    main();
extern  void    measurement();
//
extern  void    init_osc();
extern  void    init_port();
extern  void    init_comparator();
extern  void    init_adc();
extern  void    init_sdc_fat32();
extern  void    init_sdc();
extern  void    init_timer();
//
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, short
sampling_count);
extern  short   get_cycle_mode();
extern  short   get_average_cnt();
//*****************************************************************
***********
char    buf[64];
char    *file_name = "log_0000.txt";
WORD    file_id = 0;
#ifdef  FAT32
__HANDLE        fileHandle;
__TIME          TimeDate;
#endif
//*****************************************************************
***********
//■メイン関数
void    main()
{
        DWORD   clock;
        short   cnt;
        //
        init_osc();
        init_comparator();
        init_port();
        init_adc();
#ifdef  FAT32
        init_sdc_fat32();
#else
        init_sdc();
#endif
        init_timer();
        //
        for (cnt = 0; cnt < 3; cnt++) {
                LED = 1;
                Delay_ms(500);
                LED = 0;
                Delay_ms(500);
        }
        //
        while (1) {
                if (SW_START != 0) {
                        continue;
                }
                //
                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];
#ifdef  FAT32
                clock2timedate(get_clock(), &TimeDate);
                FAT32_SetTime(&TimeDate);
                FAT32_Delete(file_name);
                fileHandle = FAT32_Open(file_name, FILE_WRITE);
                FAT32_Write(fileHandle, "$START\r\n", 8);
#else
                Mmc_Fat_Assign(file_name, 0);
                Mmc_Fat_Delete();
                Mmc_Fat_Set_File_Date(2012, 12, 27, 12, 34, 56);
                Mmc_Fat_Assign(file_name, 0xA0);
                Mmc_Fat_Rewrite();
                Mmc_Fat_Write("$START\r\n", 8);
#endif
                measurement();
#ifdef  FAT32
                FAT32_Write(fileHandle, "$STOP\r\n", 7);
                FAT32_Close(fileHandle);
#else
                Mmc_Fat_Write("$STOP\r\n", 7);
#endif
                LED = 0;
        }
}
//*********************************************************************
***********
void    measurement()
{
        double  ch1, ch2, ch3, ch4, ch5, ch6, ch7, ch8;
        long    clock;
        short   cnt;
        //
        cnt = get_average_cnt();
        //
        while (SW_STOP != 0) {
                clock = wait_trigger(get_cycle_mode());
                if (clock == -1) {
                        break;
                }
                //
                clock2str(clock, buf);
                //
                ch1 = ADC_Get_Sample_Average(0, cnt);
                ch2 = ADC_Get_Sample_Average(1, cnt);
                ch3 = ADC_Get_Sample_Average(2, cnt);
                ch4 = ADC_Get_Sample_Average(3, cnt);
                ch5 = ADC_Get_Sample_Average(4, cnt);
                ch6 = ADC_Get_Sample_Average(8, cnt);
```

```
                ch7 = ADC_Get_Sample_Average(9, cnt);
                ch8 = ADC_Get_Sample_Average(10, cnt);
                //
                ch1 *= 0.8056640625;
                ch2 *= 0.8056640625;
                ch3 *= 0.8056640625;
                ch4 *= 0.8056640625;
                ch5 *= 0.8056640625;
                ch6 *= 0.8056640625;
                ch7 *= 0.8056640625;
                ch8 *= 0.8056640625;
                //
                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]);
                buf[52] = '\r';
                buf[53] = '\n';
                //
#ifdef  FAT32
                FAT32_Write(fileHandle, buf, 54);
#else
                Mmc_Fat_Write(buf, 54);
#endif
        }
}
//*********************************************************************
***********
short   get_cycle_mode()
{
        if ((SW_MODE_1 == 0) && (SW_MODE_0 == 0)) {
                return (0);
        }
        if ((SW_MODE_1 == 0) && (SW_MODE_0 == 1)) {
                return (1);
        }
        if ((SW_MODE_1 == 1) && (SW_MODE_0 == 0)) {
                return (2);
        }
        if ((SW_MODE_1 == 1) && (SW_MODE_0 == 1)) {
                return (3);
        }
}
//*********************************************************************
***********
short   get_average_cnt()
{
```

```
        if ((SW_MODE_3 == 0) && (SW_MODE_2 == 0)) {
                return (1);
        }
        if ((SW_MODE_3 == 0) && (SW_MODE_2 == 1)) {
                return (10);
        }
        if ((SW_MODE_3 == 1) && (SW_MODE_2 == 0)) {
                return (100);
        }
        if ((SW_MODE_3 == 1) && (SW_MODE_2 == 1)) {
                return (100);
        }
}
//*****************************************************************************
***********
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);
}
//*****************************************************************************
***********
WORD    ADC_Get_Sample_Average(BYTE channel, short sampling_count)
{
        int     cnt;
        DWORD   ad;
        //
        ad = 0;
        for (cnt = 0; cnt < sampling_count; cnt++) {
                ad += ADC_Get_Sample(channel);
        }
        return (ad / sampling_count);
}
//*****************************************************************************
***********
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;
}
//*****************************************************************************
***********
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。
#ifdef  FAT32
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) {
                LED = 1;
                Delay_ms(50);
                LED = 0;
                Delay_ms(50);
        }
        SPI1_Init_Advanced(_SPI_MASTER_OSC_DIV4,
_SPI_DATA_SAMPLE_MIDDLE, _SPI_CLK_IDLE_LOW, _SPI_LOW_2_HIGH);
}
#endif
//*********************************************************************
**********
//■□□□を初期化する関数です。
#ifdef  FAT16
sfr  sbit  Mmc_Chip_Select           at  RC2_bit;
sfr  sbit  Mmc_Chip_Select_Direction  at  TRISC2_bit;
//
void    init_sdc()
{
        //□□□□□□□□の初期化
        SPI1_Init_Advanced(_SPI_MASTER_OSC_DIV64,
_SPI_DATA_SAMPLE_MIDDLE, _SPI_CLK_IDLE_LOW, _SPI_LOW_2_HIGH);
        if (Mmc_Fat_Init()) {
                while (1) {
                        LED = 1;
                        Delay_ms(50);
                        LED = 0;
                        Delay_ms(50);
                }
        }
        SPI1_Init_Advanced(_SPI_MASTER_OSC_DIV16,
_SPI_DATA_SAMPLE_MIDDLE, _SPI_CLK_IDLE_LOW, _SPI_LOW_2_HIGH);
}
#endif
//*********************************************************************
**********
void    init_osc()
{
        OSCCON.IRCF2 = 1;
        OSCCON.IRCF1 = 1;
        OSCCON.IRCF0 = 1;
        OSCTUNE.PLLEN = 1;
}
//*********************************************************************
**********
void    init_adc()
{
        ADCON1.PCFG3 = 0;
        ADCON1.PCFG2 = 1;
        ADCON1.PCFG1 = 0;
```

```
        ADCON1.PCFG0 = 0;
        ADCON1.VCFG1 = 0;
        ADCON1.VCFG0 = 0;
        //
        ADC_Init();
}
//*********************************************************************
***********
void    init_port()
{
        TRISA = 0b11111111;
        TRISB = 0b11111110;
        TRISC = 0b11111111;
        //
        LED = 0;
}
//*********************************************************************
***********
void    init_comparator()
{
        CMCON.CM2 = 1;
        CMCON.CM1 = 1;
        CMCON.CM0 = 1;
}
//*********************************************************************
***********
```

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[];
```

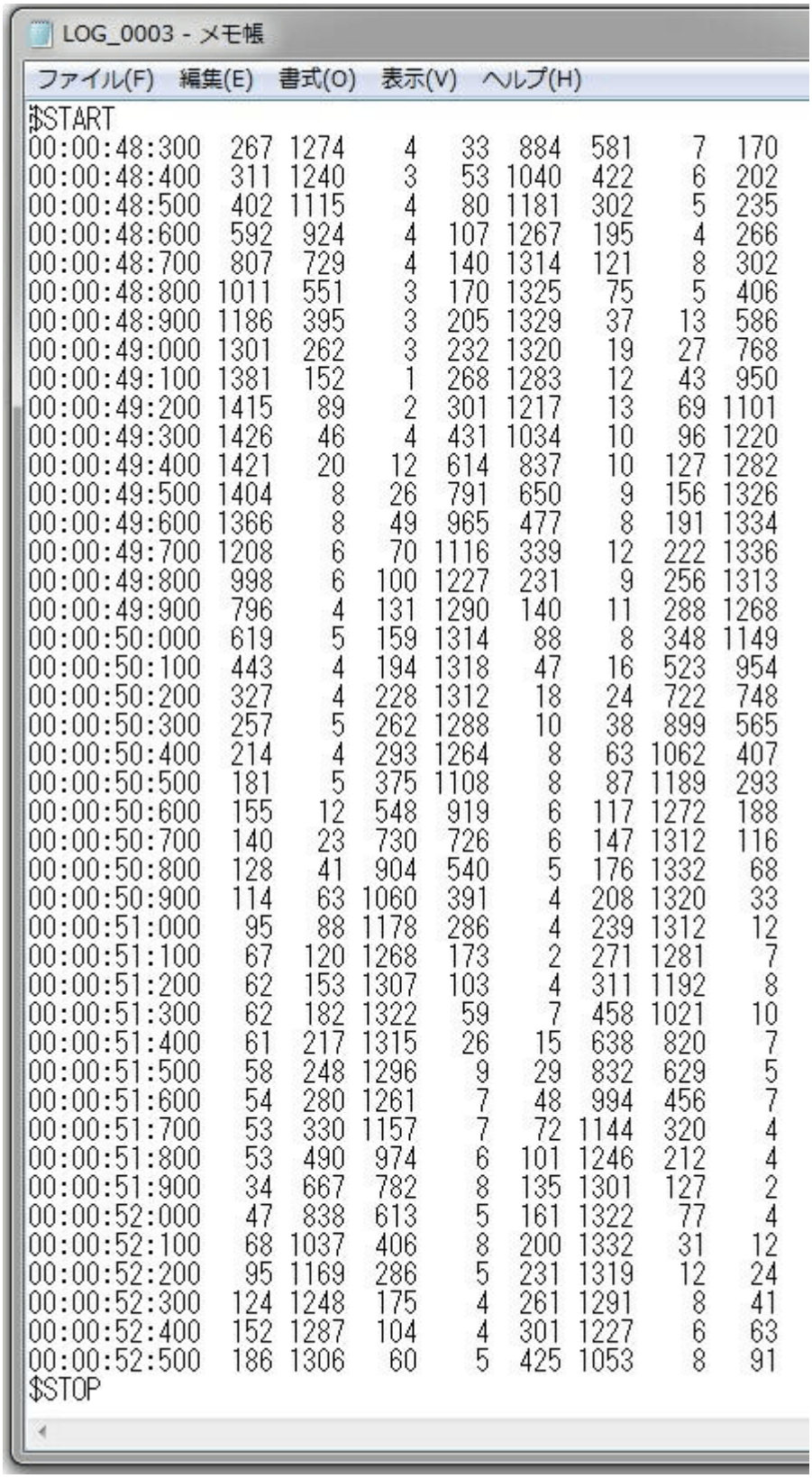
```
extern const uint8  FAT32_MAX_FILES;
extern const uint8  f32_fsi_template[SECTOR_SIZE];
extern const uint8  f32_br_template[SECTOR_SIZE];
extern       __FILE fat32_fdesc[];
extern       __RAW_SECTOR f32_sector;

////////////////////////////////////////////////////////////////////////
/////////
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ます。
2. 測定周期を設定します。
    - 100msec → SW_MODE_1=ON SW_MODE_0=ON
    - 1sec → SW_MODE_1=ON SW_MODE_0=OFF
    - 1min → SW_MODE_1=OFF SW_MODE_0=ON
    - 10min → SW_MODE_1=OFF SW_MODE_0=OFF
3. 平均測定回数を設定します。
    - 1回 → SW_MODE_3=ON SW_MODE_2=ON
    - 10回 → SW_MODE_3=ON SW_MODE_2=OFF
    - 100回 → SW_MODE_3=OFF SW_MODE_2=ON
    - 100回 → SW_MODE_3=OFF SW_MODE_2=OFF
4. 電源を入れます。
5. LEDが1秒周期で3回点滅するのを確認します。
6. スタートスイッチ(SW_START)を押下します。
7. LEDが点灯します。
8. ストップスイッチ(SW_STOP)を押下します。
9. 電源を切ります。
10. 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。

名前	更新日時	種類	サイズ
LOG_0001	2012/12/27 0:00	テキスト ドキュメント	20 KB
LOG_0002	2012/12/27 0:00	テキスト ドキュメント	1 KB
LOG_0003	2012/12/27 0:00	テキスト ドキュメント	3 KB
LOG_0004	2012/12/27 0:00	テキスト ドキュメント	3 KB
LOG_0005	2012/12/27 0:00	テキスト ドキュメント	2 KB

12. ファイルの内容を確認します。



LOG_0003 - メモ帳

ファイル(F) 編集(E) 書式(O) 表示(V) ヘルプ(H)

```
$START
00:00:48:300  267 1274    4   33  884  581    7  170
00:00:48:400  311 1240    3   53 1040  422    6  202
00:00:48:500  402 1115    4   80 1181  302    5  235
00:00:48:600  592  924    4  107 1267  195    4  266
00:00:48:700  807  729    4  140 1314  121    8  302
00:00:48:800 1011  551    3  170 1325   75    5  406
00:00:48:900 1186  395    3  205 1329   37   13  586
00:00:49:000 1301  262    3  232 1320   19   27  768
00:00:49:100 1381  152    1  268 1283   12   43  950
00:00:49:200 1415   89    2  301 1217   13   69 1101
00:00:49:300 1426   46    4  431 1034   10   96 1220
00:00:49:400 1421   20   12  614  837   10  127 1282
00:00:49:500 1404    8   26  791  650    9  156 1326
00:00:49:600 1366    8   49  965  477    8  191 1334
00:00:49:700 1208    6   70 1116  339   12  222 1336
00:00:49:800  998    6  100 1227  231    9  256 1313
00:00:49:900  796    4  131 1290  140   11  288 1268
00:00:50:000  619    5  159 1314   88    8  348 1149
00:00:50:100  443    4  194 1318   47   16  523  954
00:00:50:200  327    4  228 1312   18   24  722  748
00:00:50:300  257    5  262 1288   10   38  899  565
00:00:50:400  214    4  293 1264    8   63 1062  407
00:00:50:500  181    5  375 1108    8   87 1189  293
00:00:50:600  155   12  548  919    6  117 1272  188
00:00:50:700  140   23  730  726    6  147 1312  116
00:00:50:800  128   41  904  540    5  176 1332   68
00:00:50:900  114   63 1060  391    4  208 1320   33
00:00:51:000   95   88 1178  286    4  239 1312   12
00:00:51:100   67  120 1268  173    2  271 1281    7
00:00:51:200   62  153 1307  103    4  311 1192    8
00:00:51:300   62  182 1322   59    7  458 1021   10
00:00:51:400   61  217 1315   26   15  638  820    7
00:00:51:500   58  248 1296    9   29  832  629    5
00:00:51:600   54  280 1261    7   48  994  456    7
00:00:51:700   53  330 1157    7   72 1144  320    4
00:00:51:800   53  490  974    6  101 1246  212    4
00:00:51:900   34  667  782    8  135 1301  127    2
00:00:52:000   47  838  613    5  161 1322   77    4
00:00:52:100   68 1037  406    8  200 1332   31   12
00:00:52:200   95 1169  286    5  231 1319   12   24
00:00:52:300  124 1248  175    4  261 1291    8   41
00:00:52:400  152 1287  104    4  301 1227    6   63
00:00:52:500  186 1306   60    5  425 1053    8   91
$STOP
```

13. **ファイルの**内容を、Excelにコピー&ペーストします。

Microsoft Excel - 簡易ＡＤロガー(FAT32)

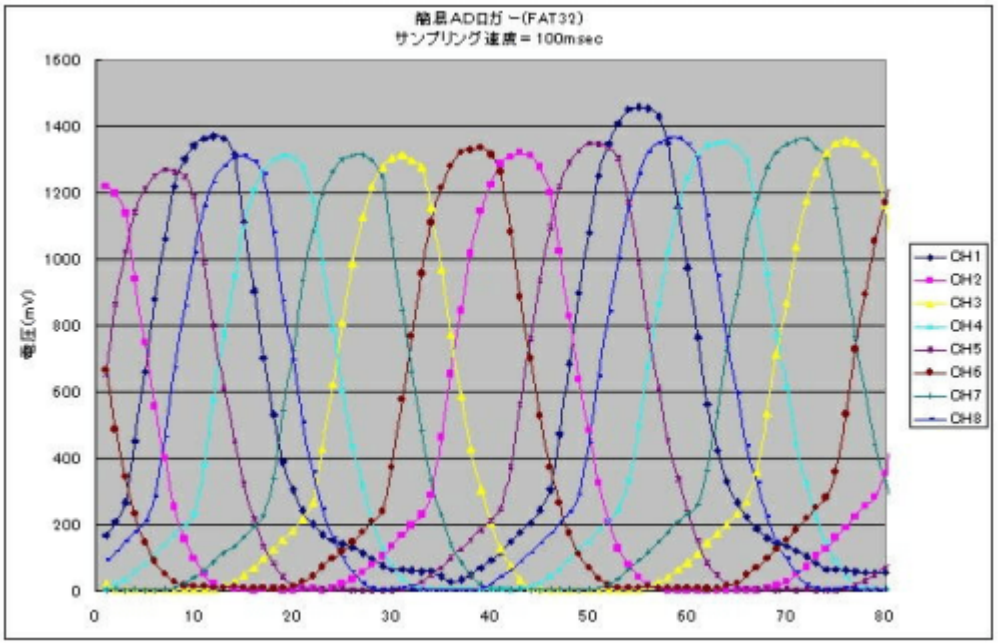
ファイル(F) 編集(E) 表示(V) 挿入(I) 書式(O) ツール(T) データ(D)

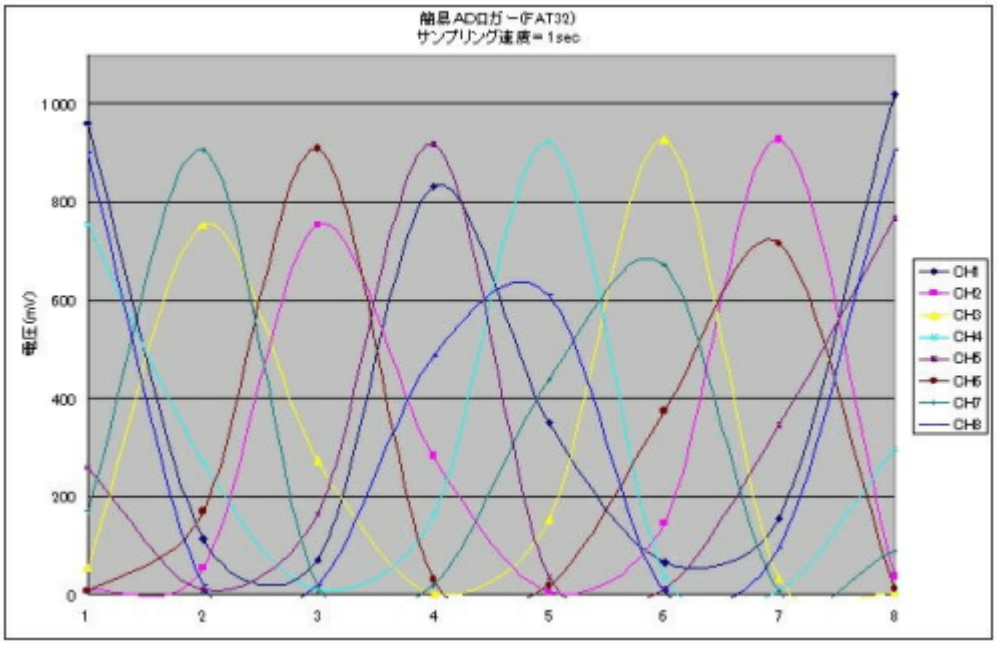
100% ＭＳ Ｐゴシック

E38

	A	B	C	D	E	F	G	H	I
1	TIME	CH1	CH2	CH3	CH4	CH5	CH6	CH7	CH8
2	00:00:06:000	962	7	54	756	259	8	169	899
3	00:00:07:000	113	53	754	269	9	169	907	19
4	00:00:08:000	71	754	273	9	165	909	17	20
5	00:00:09:000	832	283	2	160	919	30	18	485
6	00:00:10:000	351	5	154	924	35	19	437	609
7	00:00:11:000	67	145	929	38	12	375	671	8
8	00:00:12:000	155	927	34	8	347	717	8	95
9	00:00:13:000	1019	38	4	296	767	11	89	906
10	00:00:14:000	168	4	269	808	12	82	894	152
11	00:00:15:000	62	246	840	9	74	865	174	5
12	00:00:16:000	273	862	3	64	849	213	8	191
13	00:00:17:000	960	7	56	810	252	11	180	936
14	00:00:18:000	115	52	772	286	7	170	938	20
15	00:00:19:000	63	716	337	9	160	941	26	16
16	00:00:20:000	740	400	4	145	949	43	11	403
17	00:00:21:000	495	5	130	945	59	10	314	775
18	00:00:22:000	70	111	940	82	8	236	874	8
19	00:00:23:000	110	912	112	7	221	898	10	53
20	00:00:24:000	973	147	3	198	920	12	40	710
21	00:00:25:000	269	5	181	936	16	27	612	457
22	00:00:26:000	65	163	940	26	18	497	558	6
23	00:00:27:000	165	936	33	12	398	674	9	103
24	00:00:28:000	1028	43	4	288	786	12	85	913
25	00:00:29:000	184	6	229	888	13	67	854	211

14. 内容**をグラフ**表示**させます**。





From:
http://www.deepsky.jp/wiki/ - **うごくといいな**

Permanent link:
http://www.deepsky.jp/wiki/doku.php?id=elechobby:picdic:otherpic:194&rev=1588254111

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