Laboratorium 11 Obsługa kart SD/MMC (logger temperatury)

Ćwiczenie ma na celu zapoznanie użytkownika z obsługą typowych kart SD/MMC za pomocą mikrokontrolera ATMEGA16.

Karty SD można odczytywać i zapisywać za pomocą szybkiego interfejsu 4-bitowego jak i również z wolniejszej metody wykorzystującej magistralę SPI. Ćwiczenie dotyczy tej drugiej metody.
Karty SD wymagają zasilania oraz sygnałów na poziomie 3,3V. Ponieważ wykorzystywana płytka prototypowa standardowo wykorzystuje 5V, do obsługi kart wykorzystano moduł wraz z konwerterem napięć (ST2378E): KAmodMMC.

Typowo na kartach SD znajduje się system plików w standardzie FAT32.
Do obsługi odczytu i zapisu kart wraz z obsługą systemu plików FAT32 wygodnie jest wykorzystać gotową bibliotekę. Ponieważ mikrokontroler ATMEGA16 ma mało pamięci wykorzystana zostanie uproszczona biblioteka: Petit FatFs, dostępna na stronie:

http://elm-chan.org/fsw/ff/00index_p.html

Uwaga: biblioteka standardowo zawiera obsługę systemu plików, nie posiada jednak niskopoziomowych instrukcji obsługi karty SD.

Udostępnia ona ponadto obsługę plików na systemie plików FAT32 z następującymi ograniczeniami:
  – brak możliwości tworzenia nowych plików
  – brak możliwości zmieniania rozmiaru pliku
  – zapis do pliku następuje od początku sektora, dane do końca sektora dopełniane są przez wartości 0.
  – brak możliwości tworzenia katalogów
  – brak możliwości odczytu rozmiaru plików

Petit FatFs udostępnia następujące funkcje:

  • pf_mount - zamontowanie urządzenia
  • pf_open - otwarcie pliku
  • pf_read - odczyt pliku
  • pf_write - zapis pliku
  • pf_lseek - przesunięcie wskaźnika pliku
  • pf_opendir - otwarcie katalogu
  • pf_readdir - odczytanie zawartości katalogu.

Ścieżka do pliku zapisywana jest w formacie „katalog/plik.txt"

W udostępnianej konfiguracji wykorzystywana jest uproszczona tabela znaków ASCII, bez możliwości stosowania znaków narodowych.

Część 1: Przygotowanie
  – Podłączyć moduł KAmodMMC do płyty prototypowej (patrz instrukcja do modułu KamodMMC).:
      +5V do VTG
      GND do GND
      DT3 do PB4
      CMD do PB5
      DTO do PB6
      CLK do PB7

Uwaga: zworka JP1 pozwala na włączenie lub wyłączenie karty.

- – Podłączyć wyświetlacz LCD jak we wcześniejszych projektach.
- – Podłączyć termometr DS18B20 do gniazda 1-wire.
- – Z katalogu C:\Użytkownicy\Publiczny\PSM\Biblioteki pobrać plik archwium zawierający bibliotekę petitfs-atmega16.zip wraz z niskopoziomową obsługą operacji na karcie SD/MMC.
- – Przygotować plik main.c oraz wykorzystać pliki lcd.h, lcd.c z poprzednich projektów do obsługi wyświetlacza LCD oraz pliki do obsługi termomentru 1-wire: DS18B20. Przygotować plik Makefile dla projektu.

Napisać program w main.c, który odczyta z karty SD z pliku „*info/opis.txt*" 16 znaków i wyświetli w górnej części wyświetlacza LCD.

Część 2:
Dopisać do programu część odczytującą temperaturę z termometru DS18B20 w pętli, co 1 sekundę i zapisywać ją wraz z numerem pomiaru (wg formatu: "<nr pomiaru>\t<temperatura>\n") do pliku „*data/dane.txt*" na karcie SD.

Uwaga: Plik musi istnieć i mieć odpowiednią wielkość aby można było zmieścić w nim dane pomiarowe.

Podłączyć przycisk SW0 do PD7 (pamiętać o skonfigurowaniu wewnętrznych rezystorów podciągających do +5V w porcie mikrokontrolera) i sprawdzać w pętli, czy nastąpiło naciśnięcie. Jeśli tak, to sfinalizować zapis na karcie SD i zatrzymać program (restart możliwy jest przez reset).

Po wykonaniu serii pomiarów, zatrzymać program (SW0) i wyłączyć zasilanie. Wyjąć kartę z modułu KAmodMMC, podłączyć do czytnika USB i sprawdzić na komputerze zapis danych w pliku. Wyniki można zaprezentować na wykresie (temperatura w zależności od czasu).

UWAGA:
Kartę SD proszę wyjmować tylko przy wyłączonym zasilaniu płyty prototypowej.

Opis funkcyjny biblioteki petit FatFs

pf_mount

The pf_mount fucntion mounts/unmounts a volume.

```
FRESULT pf_mount (
  FATFS*  fs  /* [IN] Pointer to the work area */
);
```

Parameters

fs
> Pointer to the work area (file system object) to be registered.

Return Values

FR_OK (0)
> The function succeeded.

FR_NOT_READY
> The drive could not be initialized due to a disk error or no medium.

FR_DISK_ERR
> An error occured in the disk function.

FR_NO_FILESYSTEM
> There is no valid FAT partition on the disk.

Description

The `pf_mount()` function registers/unregisters a work area to the Petit FatFs module. The volume is mounted on registration. The volume must be mounted with this function prior to any other file function and after every media changes. To unregister the work area, specify a NULL to the `fs`, and then the work area can be discarded.

QuickInfo

Always available.

pf_open

The pf_open function opens an existing file.

```
FRESULT pf_open (
  const char* path  /* [IN] Pointer to the file neme */
);
```

Parameters

path
> Pointer to a null-terminated string that specifies the file name to open.

Return Values

FR_OK (0)
> The function succeeded.

FR_NO_FILE
> Could not find the file.

FR_NO_PATH
> Could not find the path.

FR_DISK_ERR
> The function failed due to an error in the disk function, a wrong FAT structure or an internal error.

FR_NOT_ENABLED
> The volume has not been mounted.

Description

The file must be opend prior to use `pf_read()` and `pf_lseek()` function. The open file is valid until next open.

Example

```c
void main (void)
{
    FATFS fs;           /* Work area (file system object) for the volume */
    BYTE buff[16];      /* File read buffer */
    UINT br;            /* File read count */
    FRESULT res;        /* Petit FatFs function common result code */


    /* Mount the volume */
    pf_mount(&fs);
    if (res) die(res);

    /* Open a file */
    res = pf_open("srcfile.dat");
    if (res) die(res);

    /* Read data to the memory */
    res = pf_read(buff, 16, &br);     /* Read data to the buff[] */
    if (res) die(res);                /* Check error */
    if (br != 16) die(255);           /* Check EOF */

    ....

    /* Forward data to the outgoing stream */
    do
        res = pf_read(0, 512, &br);  /* Send data to the stream */
    while (res || br != 512);         /* Break on error or eof */

    ....

    /* Unregister the work area before discard it */
    f_mount(0);
}
```

QuickInfo

Always available.

pf_read

The pf_read function reads data from the file.

```
FRESULT pf_read (
  void* buff,  /* [OUT] Pointer to the read buffer */
  WORD btr,    /* [IN]  Number of bytes to read */
  WORD* br     /* [OUT] Pointer to the variable to return number of bytes read
*/
);
```

Parameters

buff
> Pointer to the buffer to store the read data. A NULL specifies the destination is an outgoing stream.

btr
> Number of bytes to read.

br
> Pointer to the WORD variable to return number of bytes read.

Return Values

FR_OK (0)
> The function succeeded.

FR_DISK_ERR
> The function failed due to an error in the disk function, a wrong FAT structure or an internal error.

FR_NOT_OPENED
> The file has not been opened.

FR_NOT_ENABLED
> The volume has not been mounted.

Description

The read pointer in the file system object advances in number of bytes read. After the function succeeded, `*br` should be checked to detect end of file. In case of `*br < btr`, it means the read pointer reached end of file during read operation.

If a NULL is given to the `buff`, the read bytes will be forwarded to the outgoing stream instead of the memory. The streaming function will be typically built-in the `disk_readp()` function.

QuickInfo

Available when `_USE_READ == 1`.

pf_write

The pf_write function writes data to the file.

```
FRESULT pf_write (
  const void* buff, /* [IN]  Pointer to the data to be written */
  WORD btw,         /* [IN]  Number of bytes to write */
  WORD* bw          /* [OUT] Pointer to the variable to return number of bytes
```

```
written */
);
```

Parameters

buff
>   Pointer to the data to be wtitten. A NULL specifies to finalize the current write operation.

btw
>   Number of bytes to write.

bw
>   Pointer to the WORD variable to return number of bytes read.

Return Values

FR_OK (0)
>   The function succeeded.

FR_DISK_ERR
>   The function failed due to an error in the disk function, write protected, a wrong FAT structure or an internal error.

FR_NOT_OPENED
>   The file has not been opened.

FR_NOT_ENABLED
>   The volume has not been mounted.

Description

The write function has some restrictions listed below:

- Cannot create file. Only existing file can be written.
- Cannot expand file size.
- Cannot update time stamp of the file.
- Write operation can start/stop on the sector boundary.
- Read-only attribute of the file cannot block write operation.

File write operation must be done in following sequence.

1. `pf_lseek(ofs);` read/write pointer must be moved to sector bundary prior to initiate write operation or it will be rounded-down to the sector boundary.
2. `pf_write(buff, btw, &bw);` Initiate write operation. Write first data to the file.
3. `pf_write(buff, btw, &bw);` Write next data. Any other file function cannot be used while a write operation is in progress.
4. `pf_write(0, 0, &bw);` Finalize the write operation. If read/write pointer is not on the sector boundary, left bytes in the sector will be filled with zero.

The read/write pointer in the file system object advances in number of bytes written. After the function succeeded, `*bw` should be checked to detect end of file. In case of `*bw < btw`, it means the read/write pointer reached end of file during the write operation. Once a write operation is initiated, it must be finalized or the written data can be lost.

QuickInfo

Available when `_USE_WRITE == 1`.

pf_lseek

The pf_lseek function moves the file read pointer of the open file.

```
FRESULT pf_lseek (
  DWORD offset        /* [IN] File offset in unit of byte */
);
```

Parameters

offset
> Number of bytes where from start of the file

Return Values

FR_OK (0)
> The function succeeded.

FR_DISK_ERR
> The function failed due to an error in the disk function, a wrong FAT structure or an internal error.

FR_NOT_OPENED
> The file has not been opened.

Description

The `pf_lseek()` function moves the file read pointer of the open file. The `offset` can be specified in only origin from top of the file.

Example

```
/* Move to offset of 5000 from top of the file */
res = pf_lseek(5000);

/* Forward 3000 bytes */
res = pf_lseek(fs.fptr + 3000);

/* Rewind 2000 bytes (take care on wraparound) */
res = pf_lseek(fs.fptr - 2000);
```

QuickInfo

Available when `_USE_LSEEK == 1`.

pf_opendir

The pf_opendir function opens a directory.

```
FRESULT pf_opendir (
  DIR* dp,            /* [OUT] Pointer to the blank directory object structure */
  const char* path  /* [IN]  Pointer to the directory name */
);
```

Parameters

dp
      Pointer to the blank directory object to be created.

path
      Pinter to the null-terminated string that specifies the <u>directory name</u> to be opened.

Return Values

FR_OK (0)
      The function succeeded and the directory object is created. It is used for subsequent calls to read the directory entries.

FR_NO_PATH
      Could not find the path.

FR_NOT_READY
      The disk drive cannot work due to no medium in the drive or any other reason.

FR_DISK_ERR
      The function failed due to an error in the disk function, a wrong FAT structure or an internal error.

FR_NOT_ENABLED
      The volume has no work area.

Description

The `pf_opendir()` function opens an exsisting directory and creates the directory object for subsequent calls. The directory object structure can be discarded at any time without any procedure.

QuickInfo

Available when `_USE_DIR == 1`.

pf_readdir

The pf_readdir function reads directory entries.

```
FRESULT pf_readdir (
  DIR* dp,       /* [IN]  Pointer to the open directory object */
  FILINFO* fno  /* [OUT] Pointer to the file information structure */
);
```


Parameters

dp
      Pointer to the open directory object.

fno
      Pointer to the file information structure to store the read item.

Return Values

FR_OK (0)
      The function succeeded.

FR_DISK_ERR

The function failed due to an error in the disk function, a wrong FAT structure or an internal error.

FR_NOT_OPENED

The directory object has not been opened.

Description

The `pf_readdir()` function reads directory entries in sequence. All items in the directory can be read by calling this function repeatedly. When all directory entries have been read and no item to read, the function returns a null string into member `f_name[]` in the file information structure without error. When a null pointer is given to the `fno`, the read index of the directory object will be rewinded.

Sample Code

```
FRESULT scan_files (char* path)
{
    FRESULT res;
    FILINFO fno;
    DIR dir;
    int i;


    res = pf_opendir(&dir, path);
    if (res == FR_OK) {
        i = strlen(path);
        for (;;) {
            res = pf_readdir(&dir, &fno);
            if (res != FR_OK || fno.fname[0] == 0) break;
            if (fno.fattrib & AM_DIR) {
                sprintf(&path[i], "/%s", fno.fname);
                res = scan_files(path);
                if (res != FR_OK) break;
                path[i] = 0;
            } else {
                printf("%s/%s\n", path, fno.fname);
            }
        }
    }

    return res;
}
```

QuickInfo

Available when `_USE_DIR == 1`.

FATFS

The `FATFS` structure holds dynamic work area of the logical drive and a file. It is given by application program and registerd/unregisterd to the Petit FatFs module with pf_mount function. There is no member that can be changed by application programs.

```
typedef struct {
    BYTE    fs_type;      /* FAT sub type */
```

```
    BYTE    csize;       /* Number of sectors per cluster */
    BYTE    flag;        /* File status flags */
    BYTE    pad1;
    WORD    n_rootdir;   /* Number of root directory entries (0 on FAT32) */
    CLUST   n_fatent;    /* Number of FAT entries (= number of clusters + 2) */
    DWORD   fatbase;     /* FAT start sector */
    DWORD   dirbase;     /* Root directory start sector (Cluster# on FAT32) */
    DWORD   database;    /* Data start sector */
    DWORD   fptr;        /* File read/write pointer */
    DWORD   fsize;       /* File size */
    CLUST   org_clust;   /* File start cluster */
    CLUST   curr_clust;  /* File current cluster */
    DWORD   dsect;       /* File current data sector */
} FATFS;
```

FILINFO

The FILINFO structure holds a file information returned by pf_readdir function.

```
typedef struct {
    DWORD   fsize;       /* File size */
    WORD    fdate;       /* Last modified date */
    WORD    ftime;       /* Last modified time */
    BYTE    fattrib;     /* Attribute */
    char    fname[13];   /* File name */
} FILINFO;
```

Members

fsize

Indicates size of the file in unit of byte. This is always zero when it is a directory.

fdate

Indicates the date that the file was modified or the directory was created.

bit15:9

Year origin from 1980 (0..127)

bit8:5

Month (1..12)

bit4:0

Day (1..31)

ftime

Indicates the time that the file was modified or the directory was created.

bit15:11

Hour (0..23)

bit10:5

Minute (0..59)

bit4:0

Second / 2 (0..29)

fattrib

Indicates the file/directory attribute in combination of AM_DIR, AM_RDO, AM_HID, AM_SYS and AM_ARC.

fname[]

Indicates the file/directory name in 8.3 format null-terminated string.

DIR

The `DIR` structure is used for the work area to read a directory by pf_oepndir, pf_readdir function.

```
typedef struct {
    WORD    index;      /* Current read/write index number */
    BYTE*   fn;         /* Pointer to the SFN (in/out) {file[8],ext[3],status[1]}
*/
    CLUST   sclust;     /* Table start cluster (0:Static table) */
    CLUST   clust;      /* Current cluster */
    DWORD   sect;       /* Current sector */
} DIR;
```