FAT Filesystem component

REV A

Publication Date: 2013/12/16

XMOS

Table of Contents

1 Overview

IN THIS CHAPTER

The FAT Filesystem module is used to read/write files into the FAT filesystem

1.1 Features

- APIs to access FAT filesystem
- Port of FatFS - FAT file system module R0.09 (C)ChaN, 2011 (http://elm-chan.org/fsw/ff/00index_e.html).

1.2 Memory requirements

Resource	Usage
Stack	480 bytes
Program	15 Kbytes

1.3 Resource requirements

Resource	Usage
Timers	1
Clocks	1
Threads	1

1.4 Performance

SPI mode 3 is used by the SD card driver. The performance measured includes FAT filesystem performance along with the SD card driver on SPI interface.

R/W	PERFORMANCE
WRITE	1061 KBytes/s
READ	314 KBytes/s

2 Hardware requirements

IN THIS CHAPTER

2.1 Recommended hardware

2.1.1 sliceKIT

This module may be evaluated using the sliceKIT modular development platform, available from digikey. Required board SKUs are:

- XP-SKC-L2 (sliceKIT L2 Core Board)
- XA-SK-XTAG2 (sliceKIT xTAG adaptor)
- XA-SK-FLASH 1V0 Slice Card

2.2 Demonstration applications

2.2.1 Display controller application

- Package: sc_sdcard
- Application: app_sdcard_test

This demo uses the `module_FatFs` along with the `module_sdcardSPI` and `module_spi_master`.

Required board SKUs for this demo are:

- XP-SKC-L16 (sliceKIT L16 Core Board) plus XA-SK-XTAG2 (sliceKIT xTAG adaptor)

3 API

IN THIS CHAPTER

- module: module_sdcardSPI module_spi_master

The below section details the APIs in the application. For details about the FatFS APIs please refer to the respective repositories.

3.1 Configuration defines

The `module_FatFs` requires configurations defined in ffconf.h. The module requires nothing to be additionally defined. However defines can be tuned in ffconf.h. Some of the defines are:

_FS_TINY
: This defines if the sector buffer in the file system object or the individual file object should be used. When _FS_TINY is set to 1, FatFs uses the sector buffer in the file system object.

_FS_READONLY
: Setting _FS_READONLY to 1 defines read only configuration. By default this is set to 0.

_FS_MINIMIZE
: This define sets minimization level to remove some functions. By default this is set to 0 to include full set of functions.

_FS_READONLY
: Setting _FS_READONLY to 1 defines read only configuration. By default this is set to 0.

3.2 API

- `ff.c`
- `ccsbcs.c_`
- `ff.h`
- `ffconf.h`
- `integer.h`
- `diskio.h`

The FatFs module provides APIs to read/write files to SD card.

The FatFs APIs are as follows:

`FRESULT f_mount(BYTE vol, FATFS *fs)`
: Function to mount/unmount file system object to the FatFs module.

This function has the following parameters:

`vol` Logical drive number to be mounted/unmounted n.

`fs` Pointer to new file system object (NULL for unmount)

`FRESULT f_open(FIL *fp, const TCHAR *path, BYTE mode)`

Function to open or create a file.

This function has the following parameters:

`fp` Pointer to the blank file object.

`path` Pointer to the file name.

`mode` Access mode and file open mode flags.

`FRESULT f_read(FIL *fp, void *buff, UINT btr, UINT *br)`

Function to read data from a file.

This function has the following parameters:

`fp` Pointer to the file object.

`buff` Pointer to data buffer.

`btr` Number of bytes to read.

`br` Pointer to number of bytes read.

`FRESULT f_lseek(FIL *fp, DWORD ofs)`

Function to move file pointer of a file object.

This function has the following parameters:

`fp` Pointer to the file object .

`ofs` File pointer from top of file .

`FRESULT f_close(FIL *fp)`

Function to close an open file object.

This function has the following parameters:

`fp` Pointer to the file object to be closed

`FRESULT f_opendir(DIR *dj, const TCHAR *path)`

Function to create a directory object.

This function has the following parameters:

`dj` Pointer to directory object to create.

`path` Pointer to the directory path.

`FRESULT f_readdir(DIR *dj, FILINFO *fno)`

Function to read directory entry in sequence.

This function has the following parameters:

`dj` Pointer to the open directory object.

`fno` Pointer to file information to return.

`FRESULT f_stat(const TCHAR *path, FILINFO *fno)`

Function to get the file status.

This function has the following parameters:

`path` Pointer to the file path.

`fno` Pointer to file information to return .

`FRESULT f_write(FIL *fp, const void *buff, UINT btw, UINT *bw)`

Function to write data to the file.

This function has the following parameters:

`fp` Pointer to the file object.

`buff` Pointer to the data to be written.

`btw` Number of bytes to write.

`bw` Pointer to number of bytes written.

`FRESULT f_getfree(const TCHAR *path, DWORD *nclst, FATFS **fatfs)`

Function to get number of free clusters on the drive.

This function has the following parameters:

`path` Pointer to the logical drive number (root dir).

`nclst` Pointer to the variable to return number of free clusters.

`fatfs` Pointer to pointer to corresponding file system object to return.

`FRESULT f_truncate(FIL *fp)`

Function to truncate a file.

This function has the following parameters:

`fp` Pointer to the file object to be truncated.

`FRESULT f_unlink(const TCHAR *path)`

Function to delete an existing file or directory.

This function has the following parameters:

`path` Pointer to the file or directory path.

`FRESULT f_mkdir(const TCHAR *path)`

Function to create a new directory.

This function has the following parameters:

`path` Pointer to the directory path.

`FRESULT f_chmod(const TCHAR *path, BYTE value, BYTE mask)`

Function to change attribute of a file or directory.

This function has the following parameters:

`path` Pointer to the file path.

`value` Attribute bits.

`mask` Attribute mask to change.

`FRESULT f_rename(const TCHAR *path_old, const TCHAR *path_new)`

Function to rename a file or directory.

This function has the following parameters:

`path_old` Pointer to the old name.

`path_new` Pointer to the new name.

`FRESULT f_chdrive(BYTE drv)`

Function to change current drive.

This function has the following parameters:

`drv` Function to change current drive.

`FRESULT f_chdir(const TCHAR *path)`

Function to change current directory.

This function has the following parameters:

`path` Pointer to the directory path.

`FRESULT f_getcwd(TCHAR *path, UINT sz_path)`

Function to get current directory.

This function has the following parameters:

`path` Pointer to the directory path.

`sz_path` Size of path.

`FRESULT f_fdisk(BYTE pdrv, const DWORD szt[], void *work)`

Function to divide a physical drive into multipe partitions.

This function has the following parameters:

`pdrv` Physical drive number.

`szt` Pointer to the size table for each partitions.

`work` Pointer to the working buffer.

`int f_putc(TCHAR c, FIL *fp)`

Function to put a character to the file.

This function has the following parameters:

`c` The character to be output.

`fp` Pointer to the file object.

`int f_printf(FIL *fil, const TCHAR *str, ...)`

Function to print a formatted string into a file.

This function has the following parameters:

`fil` Pointer to the file object.

`str` Pointer to the formatted string.

`...` Optional arguments...

The FatFs APIs use the module_sdcardSPI APIs.

4 Programming guide