UM0168

AmebaPro File System

Abstract

This document introduces the AmebaPro FAT Filesystem, and uses the examples provided in AmebaPro SDK to show how to use the FATFS API to build and manage a filesystem.

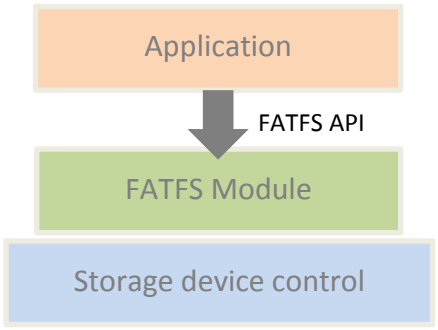
Table of Contents

1 File System Introduction

Storage is a key feature of embedded system. Ameba provides flexible method of storage management. In this document, three kinds of application scenarios will be mentioned.

1. Filesystem on Flash
2. Filesystem on SD Card and Flash

1.1 Module



AmebaPro utilizes FAT Filesystem Module to provide access to low level storage devices. Applications can manage and operate on the file system through FATFS API.

1.2 FATFS API

AmebaPro SDK uses open source FATFS module. The application interface provides various functions for applications to manipulate the file system.

- File Access
  - f_open - Open/Create a file
  - f_close - Close an open file
  - f_read - Read data from the file
  - f_write - Write data to the file
  - f_lseek - Move read/write pointer, Expand size
  - f_truncate - Truncate file size
  - f_sync - Flush cached data
  - f_forward - Forward data to the stream
  - f_expand - Allocate a contiguous block to the file
  - f_gets - Read a string
  - f_putc - Write a character
  - f_puts - Write a string
  - f_printf - Write a formatted string

- f_tell - Get current read/write pointer
- f_eof - Test for end-of-file
- f_size - Get size
- f_error - Test for an error

- Directory Access
  - f_opendir - Open a directory
  - f_closedir - Close an open directory
  - f_readdir - Read an directory item
  - f_findfirst - Open a directory and read the first item matched
  - f_findnext - Read a next item matched
- File and Directory Management
  - f_stat - Check existance of a file or sub-directory
  - f_unlink - Remove a file or sub-directory
  - f_rename - Rename/Move a file or sub-directory
  - f_chmod - Change attribute of a file or sub-directory
  - f_utime - Change timestamp of a file or sub-directory
  - f_mkdir - Create a sub-directory
  - f_chdir - Change current directory
  - f_chdrive - Change current drive
  - f_getcwd - Retrieve the current directory and drive
- Volume Management and System Configuration
  - f_mount - Register/Unregister the work area of the volume
  - f_mkfs - Create an FAT volume on the logical drive
  - f_fdisk - Create logical drives on the physical drive
  - f_getfree - Get total size and free size on the volume
  - f_getlabel - Get volume label
  - f_setlabel - Set volume label
  - f_setcp - Set active code page

More details about the usage of FATFS API, please visit http://elm-chan.org/fsw/ff/00index_e.html

2 Examples

2.1 FAT Filesystem on Flash

2.1.1 Software Setup

First, enable the Flash FATFS example in *platform_opts.h*:

```
/* For FLASH FATFS example*/
#define CONFIG_EXAMPLE_FLASH_FATFS          1
#if CONFIG_EXAMPLE_FLASH_FATFS
#define CONFIG_FATFS_EN       1
#if CONFIG_FATFS_EN
// fatfs version
#define FATFS_R_10C
#define FATFS_DISK_FLASH 1
#endif
#endif
```

Next, modify parameters in *ffconf.h*:

```
...
#define_USE_MKFS              1//0    /* 0:Disable or 1:Enable */
...
...
...
#define_MAX_SS         4096//512              1
...
```

Please note that the flash memory base for the flash filesystem used in the Flash FATFS example is defined in the file *flash_fatfs.c:*

```
...
#define FLASH_APP_BASE  0x180000
...
```

Finally, rebuild the project and download active application to DEV board.

2.1.2 Behavior Description

In this example, we demonstrate how to use FATFS on AmebaPro flash memory and manage files and directories in the filesystem.

First, we use FATFS API to register flash disk driver and get a drive number for the flash drive. We use this drive number as its path and mount to an FATFS object.

Next, the example list files currently exist in the flash memory, clear all files and directories, and list files again to check if the drive is all clean and empty.

Next, the example uses **f_mkdir** API to create a directory named *"ameba_dir"* in the root of the filesystem and use **f_open** to create a file named *"ameba_dir_file"* in ameba_dir. Then list files to show the created directory and file.

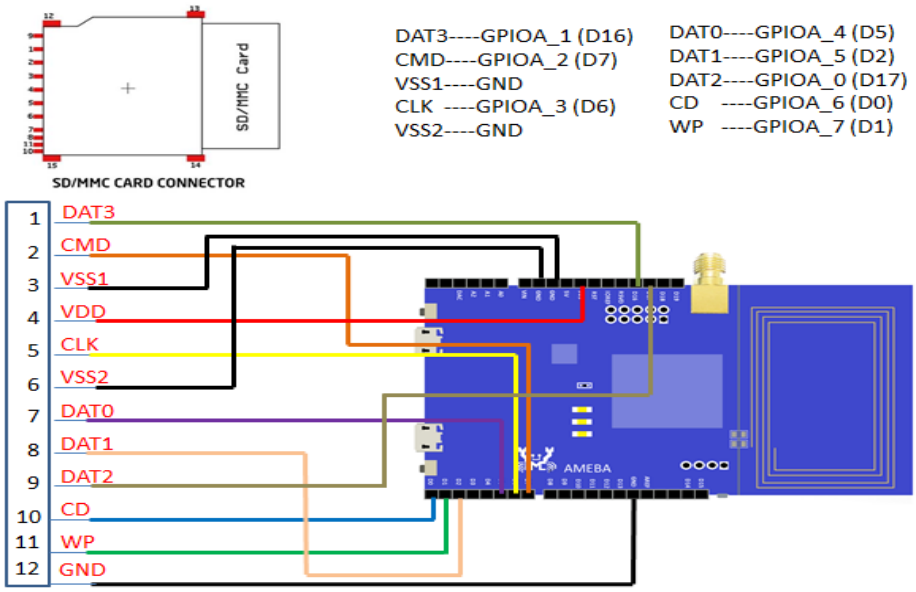
Next, we create a file named *"ameba_root_file"* at the root of the drive, and use **f_write** API to try to write some content to the file. Then use **f_read** API to read from the file to check if the content written to the file can be read back correctly.

At the end, we list all files and directories in the drive.

2.2 Dual FAT Filesystem - File system on both SD Card and Flash

2.2.1 Hardware Setup

Connect your Ameba DEV board with SD/MMC card connector before moving on, you can refer to the follow picture. After then, plug in a compatible SD card to the card connector.



2.2.2 Software Setup

Enable the DUAL FATFS example in *platform_opts.h*:

```
/* For Dual FATFS example*/
#define CONFIG_EXAMPLE_FATFS_DUAL          1
#if CONFIG_EXAMPLE_FATFS_DUAL
#define CONFIG_FATFS_EN      1
#if CONFIG_FATFS_EN
// fatfs version
#define FATFS_R_10C
#define FATFS_DISK_SD        1
#define FATFS_DISK_FLASH     1
#endif
```

Next, modify parameters in *ffconf.h*:

```
...
#define_USE_MKFS             1//0    /* 0:Disable or 1:Enable */
...
...
...
#define_MAX_SS        4096//512             1
...
```

Please note that the flash memory base for the flash filesystem used in the Flash FATFS example is defined in the file *flash_fatfs.c:*

```
...
#define FLASH_APP_BASE  0x180000
...
```

Finally, rebuild the project and download active application to DEV board.

2.2.3 Behavior Description

In this example, we demonstrate how to use FATFS on both AmebaPro flash memory and SD card, and manage files and directories in the two filesystems.

First, we use FATFS API to register flash disk driver and SD disk driver, and each drive gets a drive number. We use the drive number as drive path and mount flash drive and SD drive, each with a FATFS object.

Next, the example clears files currently exist in both drives, and list files again to check if the drives are all clean and empty.

Next, the example tests operations on the SD drive. We create a new file(*"sd_file"*) and perform read/write to the file, then create a new directory(*"sd_dir"*) and open a new file in the directory(*"sd_file2"*).

Next, the example tests similar operations on the flash drive. Create a new file(*"flash_file"*) and perform read/write to the file. Then we create a new directory(*"flash_dir"*) and open a new file in the directory(*"flash_file2"*).

After all operations, we list all files and directories in each drive.