

FAT on SD Card

AN0030 - Application Note

Introduction

This application note explains how to interface with an SD card with FAT file system using SPI.

This application note includes:

- This PDF document
- Source files (zip)
 - Example C-code
 - Multiple IDE projects

1 Introduction

The Secure Digital (SD) Card is a non-volatile memory card format developed by the SD Card Association for use in portable devices. It is based on flash memory technology and widely used in digital cameras, cell phones, ebook readers, tablet computers, netbook computers, media players, GPS receivers, and video game consoles. Ever since its adoption in the year 2000, the format has proven very popular and is considered the de-facto industry standard.

The SD card types from introduction until the present day are outlined in Table 1.1 (p. 2) and Table 1.2 (p. 2) .

Table 1.1. SD Card Size

Card type	Dimensions
SD, SDHC, SDXC, SDIO	32 x 24 x 2.1 mm
miniSD, miniSDHC, mini SDIO	21.5 x 20 x 1.4 mm
microSD, microSDHC, micro SDXC	15 x 11 x 1.0 mm

Table 1.2. Card Parameters

Card type	Year adopted	Size limit	Writing speed	FAT type
SD	2000	4GB	0.9 - 20MB/s	FAT16
SDHC	2006	32GB	2 - 40MB/s	FAT32
SDXC	2009	2TB	max 300MB/s	exFAT

The speed indications in Table 1.2 (p. 2) are orientational for most devices, their exact speed being specified by their belonging to one or other SD speed class.

Most SD cards are sold pre-formatted with a FAT file system on top of an MBR partition scheme as per the above table. The popularity of this file system allows the card to be accessed on virtually any host device with an SD reader.

SD cards are plain block devices and do not in any way imply any specific partition layout or file system thus partition schemes other than MBR partitioning and the FAT file systems can be used. Under Unix-like operating systems such as Linux or FreeBSD, SD cards can be formatted using, for example, the UFS, EXT3 or the ReiserFS file systems; under Mac OS X, SD cards can be partitioned as GUID devices and formatted with the HFS+ file system. Under Windows and some Unix systems, SD cards can be formatted using the NTFS and on later versions exFAT file system. However most consumer products will expect MBR partitioning and FAT16/FAT32 filesystem.

2 SD Card Access Methods

From the electrical standpoint, SD cards must support at least 3 access / data transfer modes :

- 1-bit SD
- 4-bit SD
- SPI

All cards must support all three modes, except for microSD where SPI is optional. SD cards must also be able to work at clock frequencies of up to 25MHz for regular cards, and 50MHz for high-speed cards. The next sections describe the currently used access modes.

2.1 One-bit SD

Separate command channel and data channel, with proprietary transfer format.

2.2 Four-bit SD

Uses a 4-bit wide bus plus additional control and command signals. Most widespread currently.

2.3 SPI

The Serial Peripheral Interface Bus access mode implements a simpler subset of the SD protocol for use with a standard SPI interface. While this mode is optional on microSD cards, it is widespread due to the extensive usage in embedded systems, especially resource-critical applications.

2.4 UHS-I

The Ultra High Speed mode is found exclusively on SDXC and SDHC cards. These products, with the UHS-I symbol, can support data transfer speeds up to 104 MB/s. UHS-I quadruples the extant top speed of 25 MB/s. UHS bus interfaces are backwards compatible. SDXC UHS-I and SDHC UHS-I memory cards can achieve best performance when paired with a UHS-I device and are designed to allow consumers to record HD resolution videos, plus perform other simultaneous recording functions.

2.5 UHS-II

Also available exclusively on SDXC and SDHC cards. The standard raises the data transfer speed to a maximum of 312 MB/s. It is part of the SD4.0 specification

3 SD Card Access with the EFM32

The present application note deals with the implementation of the SPI-based access mode to read data from / write data to a SDSC (standard SD) card using an EFM32 microcontroller.

Figure 3.1 (p. 4) shows the SD card pinout and Table 3.1 (p. 4) the pin connections for both SD and SPI modes.

Figure 3.1. SD Card Pinout



In Table 3.1 (p. 4) the type column can have the following:

- I - input
- O - output
- S - supply/ground
- C - control line
- RSV - reserved, not used

Table 3.1. SD Card Pinout Description

Pin	SD card pinout - SD Mode			SD card pinout - SPI Mode		
	Name	Type	Description	Name	Type	Description
1	CD/DAT3	I/O/C	Card detection / Connection data line 3	CS	I	Chip selection in low status
2	CMD	C	Command/ Response line	DI	I	Data input
3	V _{ss} 1	S	GND	V _{ss} 1	S	GND
4	V _{dd}	S	Power supply	V _{dd}	S	Power supply
5	CLK	I	Clock	SCLK	I	Clock
6	V _{ss} 2	S	GND	V _{ss} 2	S	GND
7	DAT0	I/O/C	Connector data line 0	D0	O/C	Data output

Pin	SD card pinout - SD Mode			SD card pinout - SPI Mode		
	Name	Type	Description	Name	Type	Description
8	DAT1	I/O/C	Connector data line 1	RSV		
9	DAT2	I/O/C	Connector data line 2	RSV		

Apart from the above matching contacts, SD card connectors may have 2 additional contacts, that signal the presence of an SD card in the socket and whether the card is write protected respectively.

4 FAT File System

4.1 Introduction

The File Allocation Table (FAT) is a computer file system architecture widely used on many computer systems and most memory cards, because of its relative simplicity. The FAT file system is quite straightforward technically and supported by virtually all existing operating systems for personal computers, which makes it a useful format for flash memory cards and a convenient way to share data between operating systems.

4.2 FAT16

The FAT16 format, was introduced in 1987 with the expansion of historical 16-bit cluster addresses to 32 bits. In 1988 this improvement became generally available through MS-DOS 4.0 and OS/2 1.1. The limit on partition size was dictated by the 8-bit signed count of sectors per cluster, which had a maximum power-of-two value of 64. With the standard hard disk sector size of 512 bytes, this gives a maximum of 32 KB clusters, thereby fixing the "definitive" limit for the FAT16 partition size at 2 GB. On magneto-optical media, which can have 1KB or 2 KB sectors instead, this size limit is proportionally larger.

4.3 FAT32

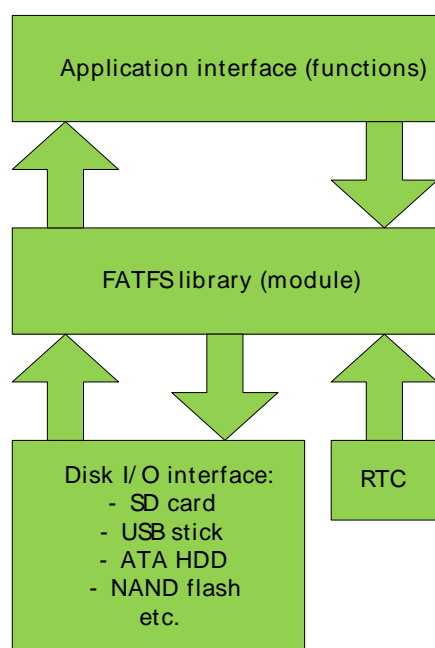
Historically, the FAT32 was introduced with Windows 95 OSR2. Cluster values are represented by 32-bit numbers, of which 28 bits are used to hold the cluster number, for a maximum of approximately 268 million clusters. This allows for drive sizes of up to 8 TB with 32 KB clusters, but the boot sector uses a 32-bit field for the sector count, limiting volume size to 2 TB on a hard disk with 512 byte sectors. The maximum possible size for a file on a FAT32 volume is 4 GB, a limit which can be well exceed by computer software today, yet deemed "reasonable" for the usual file types used in SD card (and especially embedded). Files larger than 4 GB require another formatting type such as NTFS.

The above results in NTFS being the most widespread file system format for computer systems, but FAT32 still being considered as the first choice in the applications typical to the use of solid state storage media.

4.4 The FATFS Library

The FatFs Generic FAT File System Module is a fully free FAT implementation in the form of a library and application interface module, destined to small embedded systems.

The FatFs is written in compliance with ANSI C and completely separated from the disk I/O layer, therefore it is independent of hardware architecture. It can be incorporated into low cost microcontrollers, from 8-bit to 32-bit and various architecture types. The diagram in Figure 4.1 (p. 7) shows how application interfaces the several data storage types using the FATFS library.

Figure 4.1. FATFS library interface diagram

The main features of this library are as follows :

- Windows compatible FAT file system, which is platform independent and easy to port
- Supports FAT12, FAT16 and FAT32 with 2 partitioning rules: FDISK and Super-floppy
- Very small footprint for code and work area
- Has a separate buffer for FAT structure and each file, suitable for fast multiple file access
- Various configuration options:
 - Multiple volumes (physical drives and partitions)
 - Multiple ANSI/OEM code pages including DBCS
 - Long file name support in ANSI/OEM or Unicode
 - RTOS support
 - Multiple sector size support
 - Read-only, minimized API, I/O buffer

The following is the list of functions in the Application Interface FatFs module (i.e. what is exactly implemented to access the SD card):

- `f_mount` - Register/Unregister a work area
- `f_open` - Open/Create a file
- `f_close` - Close a file
- `f_read` - Read file
- `f_write` - Write file
- `f_lseek` - Move read/write pointer, Expand file size
- `f_truncate` - Truncate file size
- `f_sync` - Flush cached data
- `f_opendir` - Open a directory
- `f_readdir` - Read a directory item
- `f_getfree` - Get free clusters
- `f_stat` - Get file status
- `f_mkdir` - Create a directory
- `f_unlink` - Remove a file or directory

- `f_chmod` - Change attribute
- `f_utime` - Change timestamp
- `f_rename` - Rename/Move a file or directory
- `f_mkfs` - Create a file system on the drive
- `f_forward` - Forward file data
- `f_chdir` - Change current directory
- `f_chdrive` - Change current drive
- `f_getcwd` - Retrieve the current directory
- `f_gets` - Read a string
- `f_putc` - Write a character
- `f_puts` - Write a string
- `f_printf` - Write a formatted string

4.5 The Disk I/O Interface

Since the FatFs module is completely separated from the disk I/O layer, it requires at least the following functions to access the physical media:

- `disk_initialize` - Initialize disk drive
- `disk_status` - Get disk status
- `disk_read` - Read sector(s)
- `disk_write` - Write sector(s)
- `disk_ioctl` - Control device dependent features
- `get_fattime` - Get current time

The low level disk I/O module is not a part of FatFs module so that it must be provided by user - in the case of our application, these are the SPI bus communication routines for the EFM32 (sample drivers are available in the original FatFs resources).

5 EFM32 Software Example

The software example project with the current application note uses the EFM32 CMSIS including DVK BSP (board support package) and demonstrates basic file-level operation with the SD card. It sets up access to DVK registers and supports FAT file system on the SD card. The firmware has been intended to have as hardware support the EFM32 Gecko Development Kit (EFM32-Gxxx-DK) and the initial target devices are the EFM32G290F128 and the EFM32G890F128, but it could run with minimal, if any, modifications on any EFM32 targets.

The steps required to run the example code are the following (guidelines are for usage with the IAR development environment, similar steps should be followed in case of other IDEs/compilers):

1. Connect the EFM32-Gxxx-DK to the PC
2. Compile and build the project code
3. Press CFG button and select Debug Control MCU
4. Download the code to the MCU
5. Optional: set a breakpoint to the last `while(1)` and press and press Go
6. Insert into the microSD card holder a FAT32 formatted micro-SD card
7. A text file named test.txt will be created containing the message "EFM32 ...the world's most energy friendly microcontrollers !".

Note

1. If the optional breakpoint is set it will normally be trapped. If any errors occur (such as removing the card while the program is running, bad card contacts, insertion of a non-FAT32 formatted card, etc#), then the breakpoint will not be trapped, which is an indication of an error.
2. The prototype board may need to be removed in order for this example to work correctly.

The firmware performs the following:

- Initializes the file write buffer
- Detects if the microSD card is present
- Checks if it is FAT32 formatted
- Opens the test.txt file for write operation. If this file does not exist, it will be created
- Sets the file write pointer to the first location
- Writes the text buffer to the file
- Closes the file
- Opens the test.txt file for read operation
- Sets the file read pointer to the first location
- Reads the file
- Closes the file
- Compares the buffers

The EM microsd.c driver file provides all the low level SPI routines used for the FatFS system. The driver can be found in the Simplicity Studio install folder under \kits\EFM32_Gxxx_DK\drivers\microsd.c

The SPI low level functions available are:

1. `void MICROSD_init(void)`

Description: the SPI is configured as a master and does not use interrupts for receiver/transmitter routines

Pins allocation and configuration

- `GPIO_PinModeSet(gpioPortC, 11, gpioModePushPull, 0); /* MOSI */`
- `GPIO_PinModeSet(gpioPortC, 10, gpioModePushPull, 0); /* MISO */`

- `GPIO_PinModeSet(gpioPortC, 8, gpioModePushPull, 0); /* CS */`
- `GPIO_PinModeSet(gpioPortC, 9, gpioModePushPull, 0); /* Clock */`

Remarks:

- the FatFS system controls the SPI CS directly via the `CS_HIGH()` and `CS_LOW()` macros
- the FatFS system needs a low SPI speed for initialization through the macros `FCLK_SLOW()` and another speed for normal operation `FCLK_FAST()`.

2. `void xmit_spi(BYTE dat)`

Description: transmits one byte from EFM to SD via SPI

3. `BYTE rcvr_spi(void)`

Description: Receives one byte from SD to EFM via SPI

4. `void rcvr_spi_m(BYTE *p)`

Description: Receives one byte from SD to EFM via SPI, where the passed parameter is a pointer to the character to receive

5. `BYTE wait_ready(void)`

Description: Waits for the card ready condition

6. `void deselect(void)`

Description: Deselects the card and releases the SPI bus

7. `int select(void)`

Description: Selects the card and waits for the ready condition

8. `void power_on(void)`

Description: Turns on SD card power. As the DVK does not support socket power control, it only enables the SPI clock

9. `void power_off(void)`

Description: Turns off SD card power. As the DVK does not support socket power control, it only disables the SPI clock

10. `int rcvr_datablock(BYTE *buff, UINT btr)`

Description: Receives a data packet from MMC

```
11int xmit_datablock(const BYTE *buff, BYTE token)
```

Description: Sends a data packet to MMC

```
12BYTE send_cmd (BYTE cmd, DWORD arg)
```

Description: Sends a command packet to MMC

Note

Further information on how to use a MMC/SDC is available from: http://elm-chan.org/docs/mmc/mmc_e.html

6 Revision History

6.1 Revision 1.08

2014-05-07

Updated example code to CMSIS 3.20.5

Changed to Silicon Labs license on code examples

Added project files for Simplicity IDE

Removed makefiles for Sourcery CodeBench Lite

6.2 Revision 1.07

2013-10-14

New cover layout

6.3 Revision 1.06

2013-05-08

Added software projects for ARM-GCC and Atollic TrueStudio.

Renamed MICROSD_init to MICROSD_Init to comply with new driver naming conventions.

6.4 Revision 1.05

2012-11-12

Adapted software projects to new kit-driver and bsp structure.

6.5 Revision 1.04

2012-8-12

Updated file paths for fatfs

6.6 Revision 1.03

2012-04-20

Adapted software projects to new peripheral library naming and CMSIS_V3.

6.7 Revision 1.02

2011-10-21

Updated IDE project paths with new kits directory.

6.8 Revision 1.01

2011-04-11

Added software project for EFM32G290F128.

6.9 Revision 1.00

2011-03-22

Initial revision.

A Disclaimer and Trademarks

A.1 Disclaimer

Silicon Laboratories intends to provide customers with the latest, accurate, and in-depth documentation of all peripherals and modules available for system and software implementers using or intending to use the Silicon Laboratories products. Characterization data, available modules and peripherals, memory sizes and memory addresses refer to each specific device, and "Typical" parameters provided can and do vary in different applications. Application examples described herein are for illustrative purposes only. Silicon Laboratories reserves the right to make changes without further notice and limitation to product information, specifications, and descriptions herein, and does not give warranties as to the accuracy or completeness of the included information. Silicon Laboratories shall have no liability for the consequences of use of the information supplied herein. This document does not imply or express copyright licenses granted hereunder to design or fabricate any integrated circuits. The products must not be used within any Life Support System without the specific written consent of Silicon Laboratories. A "Life Support System" is any product or system intended to support or sustain life and/or health, which, if it fails, can be reasonably expected to result in significant personal injury or death. Silicon Laboratories products are generally not intended for military applications. Silicon Laboratories products shall under no circumstances be used in weapons of mass destruction including (but not limited to) nuclear, biological or chemical weapons, or missiles capable of delivering such weapons.

A.2 Trademark Information

Silicon Laboratories Inc., Silicon Laboratories, Silicon Labs, SiLabs and the Silicon Labs logo, CMEMS®, EFM, EFM32, EFR, Energy Micro, Energy Micro logo and combinations thereof, "the world's most energy friendly microcontrollers", Ember®, EZLink®, EZMac®, EZRadio®, EZRadioPRO®, DSPLL®, ISOmodem®, Precision32®, ProSLIC®, SiPHY®, USBXpress® and others are trademarks or registered trademarks of Silicon Laboratories Inc. ARM, CORTEX, Cortex-M3 and THUMB are trademarks or registered trademarks of ARM Holdings. Keil is a registered trademark of ARM Limited. All other products or brand names mentioned herein are trademarks of their respective holders.

B Contact Information

Silicon Laboratories Inc.

400 West Cesar Chavez

Austin, TX 78701

Please visit the Silicon Labs Technical Support web page:

<http://www.silabs.com/support/pages/contacttechnicalsupport.aspx>

and register to submit a technical support request.

Table of Contents

1. Introduction	2
2. SD Card Access Methods	3
2.1. One-bit SD	3
2.2. Four-bit SD	3
2.3. SPI	3
2.4. UHS-I	3
2.5. UHS-II	3
3. SD Card Access with the EFM32	4
4. FAT File System	6
4.1. Introduction	6
4.2. FAT16	6
4.3. FAT32	6
4.4. The FATFS Library	6
4.5. The Disk I/O Interface	8
5. EFM32 Software Example	9
6. Revision History	12
6.1. Revision 1.08	12
6.2. Revision 1.07	12
6.3. Revision 1.06	12
6.4. Revision 1.05	12
6.5. Revision 1.04	12
6.6. Revision 1.03	12
6.7. Revision 1.02	12
6.8. Revision 1.01	12
6.9. Revision 1.00	13
A. Disclaimer and Trademarks	14
A.1. Disclaimer	14
A.2. Trademark Information	14
B. Contact Information	15
B.1.	15

List of Figures

3.1. SD Card Pinout 4

4.1. FATFS library interface diagram 7

List of Tables

1.1. SD Card Size 2

1.2. Card Parameters 2

3.1. SD Card Pinout Description 4

silabs.com

