1. Giới thiệu MMC/SD card

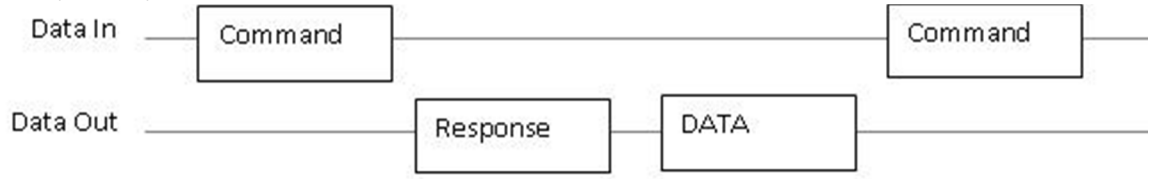
MMC là viết tắt của MultiMedia Card. Đây là loại thẻ nhớ sử dụng bộ nhớ NAND flash để lưu trữ dữ liệu được giới thiệu lần đầu vào năm 1997 bởi Siemens AG và SanDisk. Đối với các ứng dụng nhúng ở mức vi điều khiển, MMC/SD card là sự lựa chọn thích hợp cho các ứng dụng cần lưu trữ dữ liệu vì kết nối phần cứng với thẻ MMC/SD đơn giản, hỗ trợ giao tiếp SPI, đồng thời dung lượng bộ nhớ lớn(có thể lên tới 32GBs). Bộ nhớ của thẻ MMC/SD được tổ chức dạng block, tương tự như ổ cứng(hardisk) trong máy tính. Do vậy cũng như ổ cứng, MMC/SD sử dụng tập lệnh ATA để giao tiếp với phần mềm điều khiển.

Một số đặc điểm khi thẻ MMC/SD hoạt động ở chế độ SPI

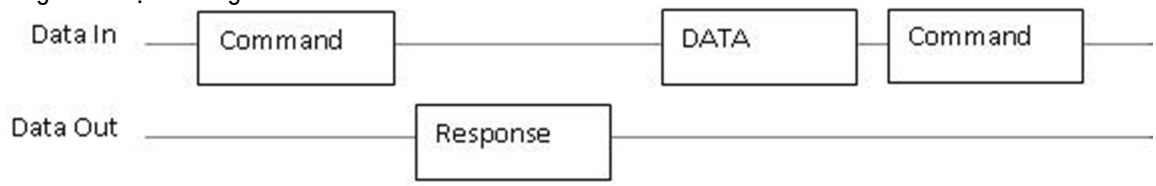
+ Truyền dữ liệu đồng bộ trên 3 tín hiệu: CLK, DI, DO.
+ Kích hoạt thẻ thông qua tín hiệu : Chip Select(CS).
+ Tần số hoạt động từ 0-20MHz.
+ Hỗ trợ truy cập Single Block và Multi Block.
+ Sơ đồ kết nối

Thứ tự chân	Chân kết nối	Loại	Mô tả
1	CS	Input	Chip Select
2	DI	Input/Push Pull	Data Input
3	VSS	-	
4	VDD	-	
5	BCLK	Input	Clock Source
6	VSS2	-	
7	DO	Output/Push Pull	Data Output
8	-		
9	-		

Khi hoạt động ở chế độ SPI, các lệnh điều khiển và dữ liệu được truyền chung trên 2 tín hiệu DI và DO
Khi đọc dữ liệu từ thẻ

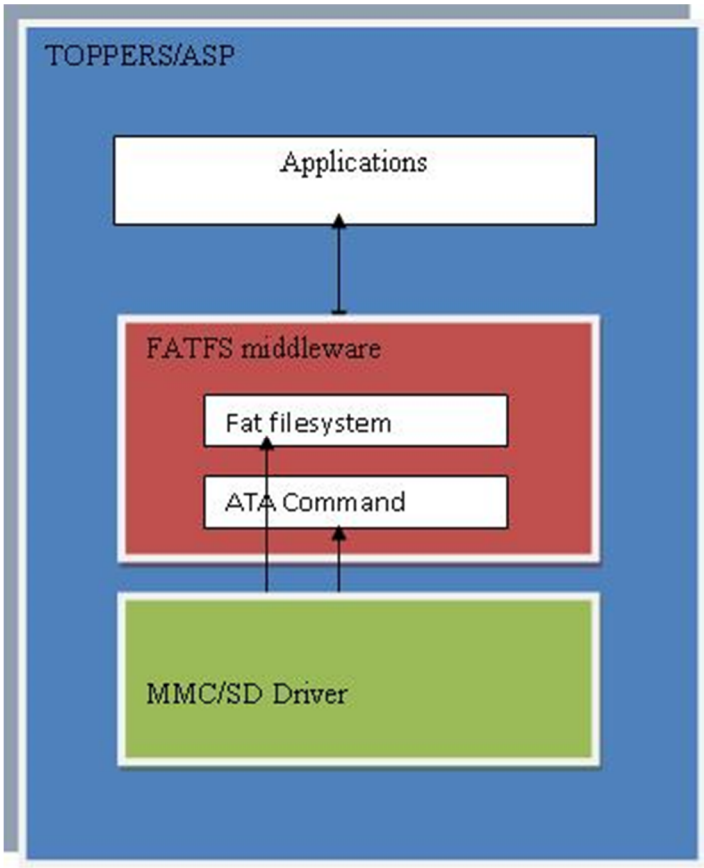


Khi ghi dữ liệu xuống thẻ



2. FATFS

FatFS là bộ mã nguồn miễn phí hỗ trợ định dạng FAT(File Allocation Table) được sử dụng rộng rãi trong hệ điều hành Windows. Tổ chức của FatFS được mô tả như sau



FatFS sẽ cung cấp giao diện các hàm cơ bản để thực thi các thao tác file trên thẻ MMC/SD tương tự như lập trình file trên máy tính.

+ FatFS hỗ trợ các kiểu format FAT12, FAT16, FAT32.

+ Các hàm giao diện lập trình: f_mount, f_open, f_close, f_read, f_write,f_lseek, f_sync, f_opendir, f_readdir, f_getfree, f_stat, f_mkdir, f_untrnk, f_chmod, f_rename, f_mkfs.

FatFS gồm 2 phần chính là phần FAT bao gồm các hàm liên quan đến File Allocation Table, và phần ATA liên quan đến xử lý các lệnh ATA để giao tiếp với thẻ nhớ.

Tổ chức file chính của FatFs gồm:

+ file ff.c: gồm các hàm hỗ trợ FAT.

+ file ata.c: gồm các hàm giao tiếp ATA command.

Để giao tiếp với phần điều khiển (driver) của thẻ MMC/SD, FatFS yêu cầu 5 hàm giao diện

+ disk_initialize: hàm khởi tạo kết nối tới thẻ MMC/SD, đồng thời kích hoạt cho thẻ ở trạng thái sẵn sàng hoạt động.

+ disk_write: ghi dữ liệu vào một vùng sector nhất định.

+ disk_read: đọc dữ liệu từ một vùng sector cho trước.

+ disk_status: lấy trạng thái của thẻ.

+ disk_ioctl: các hàm xử lý các yêu cầu khác sẽ được bố trí xử lý ở đây.

Kiến trúc của FatFS đơn giản do đó rất tiện cho việc "port" sang một hệ nhúng khác. Người dùng không cần có kiến thức sâu về FAT vẫn có thể sử dụng FatFS vào hệ thống của mình.

3. Giao tiếp với thẻ MMC/SD

Thẻ MMC/SD có 4 chế độ hoạt động là: InActive, Card Identification, Data Transfer, Interrupt. Thông thường thẻ sẽ hoạt động ở hai chế độ Card Identification và Data Transfer.

Để giao tiếp trao đổi dữ liệu với thẻ MMC/SD, vi điều khiển phải phát lệnh điều khiển xuống thẻ. Định

dạng lệnh MMC được tổ chức gồm 48 bit như sau

Vị trí bit	47	46	[45-40]	[39-8]	[7-1]	0
Độ lớn	1	1	6	32	7	1
Giá trị	0	1	x	x	x	1
Mô tả	Start bit	Transmission bit	Command index	Argument	CRC7	End bit

Ở chế độ SPI, checksum luôn có giá trị là 0xFE ngoại trừ lúc khởi động vì lúc này tính năng tính checksum vẫn còn hoạt động. Khi nhận lệnh từ vi điều khiển, thẻ MMC/SD luôn trả lời lại bằng 1 byte. Nếu giá trị trả về là 0xFF có nghĩa là thẻ bận.

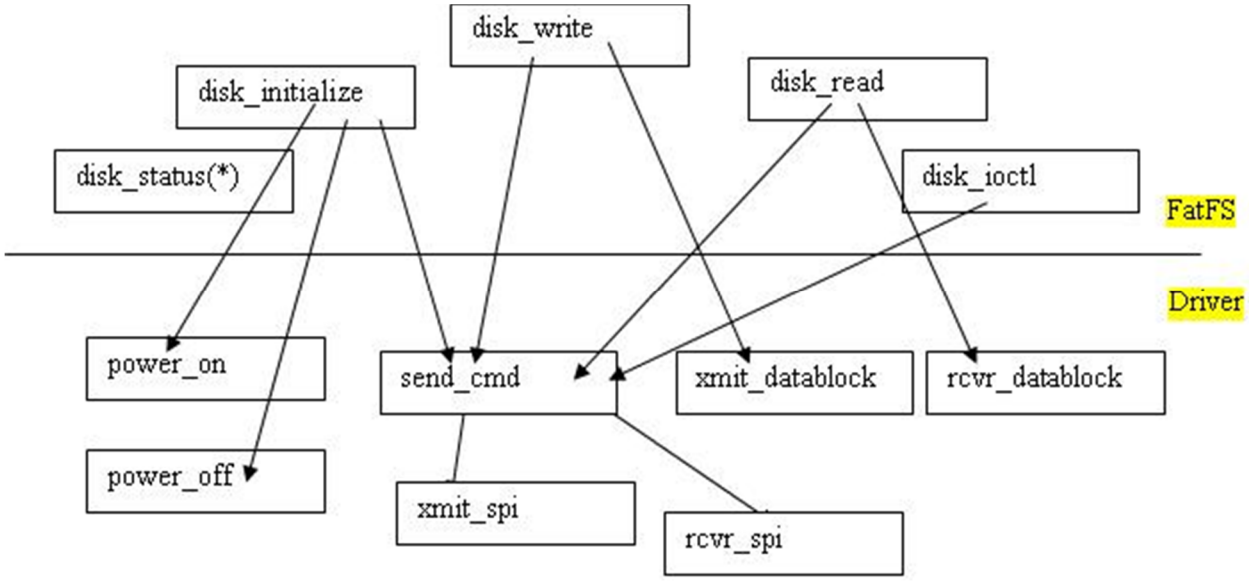
Các lệnh MMC/SD thường gặp

Mã lệnh	Ký hiệu	Mô tả
CMD0	GO_IDLE_STATE	Reset thẻ về trạng thái idle
CMD1	SEND_OP_CODE	Yêu cầu thẻ gửi nội dung thông tin của Operating Condition Regiters
CMD8	SEND_EXT_CSD	Yêu cầu thẻ gửi thông tin các thanh ghi CSD(Card Specific Data) dưới dạng block dữ liệu.
CMD9	SEND_CSD	Yêu cầu thẻ gửi thông tin cụ thể của thanh ghi CSD.
CMD10	SEND_CID	Yêu cầu gửi các thông tin CID(Card Information Data).
CMD12	STOP_TRANSMISSION	Ngưng trao đổi dữ liệu
CMD16	SET_BLOCKLEN	Thiết lập độ lớn tính theo byte của một block dữ liệu, giá trị mặc này được lưu trong CSD
CMD17	READ_SINGLE_BLOCK	Đọc một block dữ liệu
CMD18	READ_MULTIPLE_BLOCK	Đọc nhiều block dữ liệu. Số lượng block được thiết lập bởi lệnh CMD23
CMD23	SET_BLOCK_COUNT	Thiết lập số lượng block dữ liệu để ghi hoặc đọc.
CMD24	WRITE_BLOCK	Ghi một block dữ liệu.
CMD25	WRITE_MULTIPLE_BLOCK	Ghi nhiều block dữ liệu. Số lượng block được thiết lập bởi lệnh CMD23
CMD55	APP_CMD	Thông báo cho thẻ nhớ lệnh tiếp theo là lệnh riêng của ứng dụng chứ không phải là lệnh chuẩn của MMC.

4. Porting FatFS vào board STM32-GEM3M-2

Như phân tích ở trên, chúng ta chỉ cần chú ý vào 5 hàm giao diện với thẻ MMC/SD của FatFS.

Các hàm này sẽ trực tiếp gọi đến các hàm điều khiển phần cứng ta thường hay gọi là driver để trao đổi dữ liệu trực tiếp với thẻ.



(*): disk_status không cần thiết phải cài đặt.

Như vậy chúng ta chỉ cần cài đặt các hàm ở khối "Driver" phù hợp với phần cứng của board STM32-GEM3M-2 là được.

+ hàm power_on(): nhiệm vụ hàm này là thiết lập các chân tín hiệu ra của STM32 cho phù hợp với kết nối SPI tới thẻ.

+ hàm power_off(): giải phóng các thiết lập trong hàm power_on().

+ hàm send_cmd(): gửi lệnh xuống thẻ theo đúng định dạng lệnh MMC/SD được mô tả trong bảng 1.

```
/* Send command packet */
xmit_spi(cmd);                          /* Start + Command index */
xmit_spi((BYTE)(arg >> 24));            /* Argument[31..24] */
xmit_spi((BYTE)(arg >> 16));            /* Argument[23..16] */
xmit_spi((BYTE)(arg >> 8));             /* Argument[15..8] */
xmit_spi((BYTE)arg);                    /* Argument[7..0] */
n = 0x01;                               /* Dummy CRC + Stop */
if (cmd == CMD0) n = 0x95;              /* Valid CRC for CMD0(0) */
if (cmd == CMD8) n = 0x87;              /* Valid CRC for CMD8(0x1AA) */
xmit_spi(n);
```

+ hàm xmit_datablock(): gửi một khối dữ liệu xuống

```
xmit_spi(token);                                /* Xmit data token */
if (token != 0xFD) {                            /* Is data token */
        wc = 0;
        do {                                    /* Xmit the 512 byte data block to MMC */
                xmit_spi(*buff++);
                xmit_spi(*buff++);
        } while (--wc);
        xmit_spi(0xFF);                         /* CRC (Dummy) */
        xmit_spi(0xFF);
        resp = rcvr_spi();                      /* Reveive data response */
        if ((resp & 0x1F) != 0x05)              /* If not accepted, return with error */
                return 0;
}
```

+ hàm rcvr_datablock():

```
do {	/* Wait for data packet in timeout of 200ms */
		token = rcvr_spi();
#ifdef SUPPORT_TIMEOUT
} while ((token == 0xFF) && Timer1);
#else
} while ((token == 0xFF));
#endif
if(token != 0xFE)
		return 0;		/* If not valid data token, retutn with error */
do {	/* Receive the data block into buffer */
		rcvr_spi_m(buff++);
		rcvr_spi_m(buff++);
		rcvr_spi_m(buff++);
		rcvr_spi_m(buff++);
} while (btr -= 4);
rcvr_spi();							/* Discard CRC */
rcvr_spi();
```

Trước khi nhận dữ liệu, kiểm tra xem thẻ MMC/SD có đang bận hay không

+ hàm xmit_spi():

```
#define SPI_SD			SPI1
#define xmit_spi(dat)		sd_raw_rw_spi(dat)
BYTE sd_raw_rw_spi(BYTE b_data)
{
		BYTE Data = 0;
		  /* Wait until the transmit buffer is empty */
		  //while (SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_TXE) == RESET);
		  /* Send the byte */
		  SPI_I2S_SendData(SPI_SD, b_data);
		  /* Wait until a data is received */
		  while (SPI_I2S_GetFlagStatus(SPI_SD, SPI_I2S_FLAG_RXNE) == RESET);
		  /* Get the received data */
		  Data = SPI_I2S_ReceiveData(SPI_SD);
		  /* Return the shifted data */
		  return Data;
}
```

+ hàm rcvr_spi():

```
static
BYTE rcvr_spi (void)
{
		return sd_raw_rw_spi(0xFF);
}
```

4. Kết luận

FatFS với mã nguồn nhỏ gọn, rõ ràng đã cung cấp khả năng tuyệt vời truy cập file dưới định dạng FAT. Kiến trúc của FatFS rất phù hợp với các ứng dụng nhúng, tính khả chuyển(port) cho phép người phát triển sử dụng lại mã nguồn trên nhiều nền tảng phần cứng khác nhau. Người lập trình không cần phải

hiểu rõ về định dạng FAT vẫn có thể sử dụng thành thạo bộ mã nguồn này trong các ứng dụng có điều khiển thẻ MMC/SD của mình. Đây chính là lợi ích lớn nhất của các middleware như FatFS.

Bài 2: **Giao tiếp cơ bản MMC/SD Card với STM32**

Demo này cung cấp phần driver cho SPI của STM32 khi giao tiếp với thẻ nhớ MMC/SD. Ở đây STM32F103RDT6 dùng SPI1 chạy trên KítOPENCMX-STM3210D, nếu muốn dùng SPI2 chỉ việc thay file diskio.c trong project bằng file diskio khác (có đính kèm trong folder involve) rồi biên dịch lại là Ok. Có thể chạy trên các Kít khác của WWW.ARM.VN chỉ với một vài thay đổi nhỏ về cấu hình phần cứng.

Demo này thực hiện việc ghi một khối dữ liệu từ bộ đệm buff1[512] như bên dưới vào sector thứ 200, sau đó đọc ra lại chứa vào bộ đệm buff2[512], rồi hiện thị ra màn hình bằng USART1.

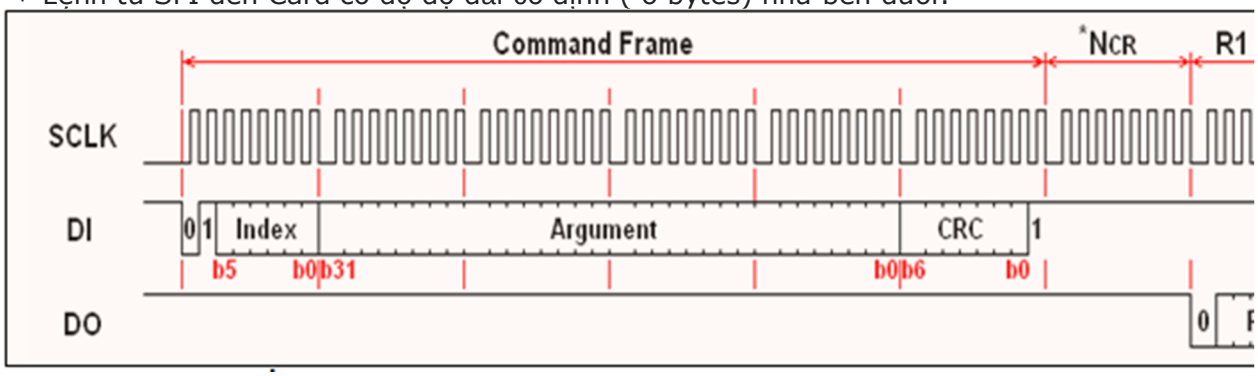
buff1[512]= "Sharing your knowledge and experience in ARM Viet Nam";

Để biết cách dùng Driver SPI này giao tiếp đọc một file bất kỳ từ SD Card thì tham khảo demo DOSFS.

1. Lệnh và đáp ứng của Micro SD Card

+ Trong SPI mode, hướng của dữ liệu trên đường tín hiệu được cố định, dữ liệu truyền đồng bộ nối tiếp theo từng byte .

+ Lệnh từ SPI đến Card có độ độ dài cố định (6 bytes) như bên dưới:



Hình 1. Giao tiếp giữa SD Card và SPI

Trong đó:

SCLK : SPI2_SCK

DI : SPI2_MOSI

DO : SPI2_MISO

NCR: thời gian đáp ứng của lệnh (tùy vào từng loại Card mà có thời gian khác nhau)

+ Khi một khung lệnh được truyền đến Card, một đáp ứng tương ứng cho lệnh đó (R1, R2,R3) có thể được đọc từ Card. Vì việc chuyển dữ liệu được lái bằng xung clock của SPI do đó sau khi truyền xong khung lệnh SPI cần tiếp tục cấp xung clock cho Card thì mới có thể nhận được đáp ứng từ Card (bằng cách gửi liên tục giá trị 0xFF và đọc giá trị trả về cho tới khi nhận được đáp ứng đúng).

a) Cấu trúc lệnh của SD Card

Một khung lệnh có độ dài 6 bytes gồm các trường như bên dưới

Vị trí bit	47	46	[45- 40]	[39 – 8]	[7 – 1]	0
Kích thước	1	1	6	32	7	1
Giá trị	0	1	x	x	x	1
Mô tả	Start bit	Transmittion	Command	argument	CRC7	End

		bit	index			bit

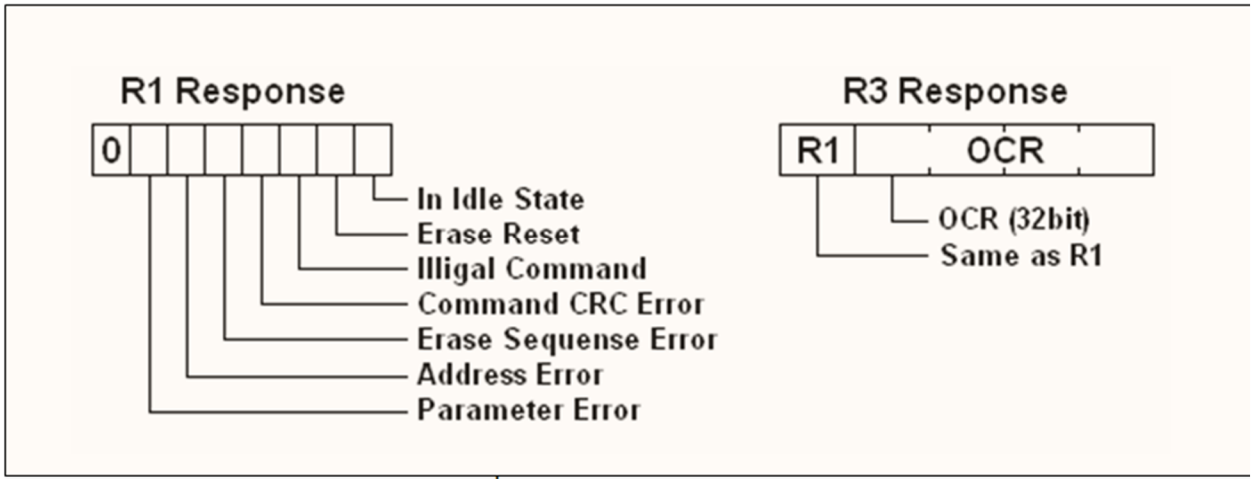
Một số lệnh thường gặp khi giao tiếp với Micro SD Card

Bảng 1. Một số lệnh thường gặp của SD Card

Mã lệnh	***Ký hiệu***	***Mô tả***
CMD0	GO_IDLE_STATE	Reset thẻ về trạng thái idle
CMD1	SEND_OP_CODE	Yêu cầu thẻ gửi nội dung thông tin của Operating Condition Regiters
CMD8	SEND_EXT_CSD	Yêu cầu thẻ gửi thông tin các thanh ghi CSD(Card Specific Data) dưới dạng block dữ liệu.
CMD9	SEND_CSD	Yêu cầu thẻ gửi thông tin cụ thể của thanh ghi CSD.
CMD10	SEND_CID	Yêu cầu gửi các thông tin CID(Card Information Data).
CMD12	STOP_TRANSMISSION	Ngưng trao đổi dữ liệu
CMD16	SET_BLOCKLEN	Thiết lập độ lớn tính theo byte của một block dữ liệu, giá trị mặc này được lưu trong CSD
CMD17	READ_SINGLE_BLOCK	Đọc một block dữ liệu
CMD18	READ_MULTIPLE_BLOCK	Đọc nhiều block dữ liệu. Số lượng block được thiết lập bởi lệnh CMD23
CMD23	SET_BLOCK_COUNT	Thiết lập số lượng block dữ liệu để ghi hoặc đọc.
CMD24	WRITE_BLOCK	Ghi một block dữ liệu.
CMD25	WRITE_MULTIPLE_BLOCK	Ghi nhiều block dữ liệu. Số lượng block được thiết lập bởi lệnh CMD23
MD55	APP_CMD	Thông báo cho thẻ nhớ lệnh tiếp theo là lệnh riêng của ứng dụng chứ không phải là lệnh chuẩn

b) Đáp ứng của SD Card

Có 3 dạng đáp ứng, tùy vào lệnh được gửi mà ta có dạng R1, R2 và R3. Trong đó đáp ứng R1 cho phần lớn các lệnh.



Hình 2. Cấu trúc đáp ứng R1 và R3

• Đáp ứng R1: có độ dài 8 bit, gồm 7 bit trạng thái. Khi một lỗi xuất hiện thì bit tương ứng sẽ được đặt lên 1. Giá trị 0x00 nghĩa là thành công.

• Đáp ứng R3: gồm R1 cộng với 32 bit OCR, chỉ dành cho lệnh CMD58.

• Ngoài ra còn có một số dạng đápkhác nhưng ứng ít gặp.

2. Khởi tạo SD Card

Ở trạng thái Idle, SD Card chỉ chấp nhận CMD0, CMD1, ACMD41và CMD58, mọi lệnh khác

sẽ bị từ chối.
Các bước khởi tạo Card:
1) Gửi lệnh CMD1 đưa Cadr rời trạng Idle (gửi lệnh CMD1 và đợi nhận Response thích hợp, Response thay đổi từ 0x01 sang 0x00).
2) Nếu muốn thay đổi độ dài của khối dữ liệu thì gửi lệnh CMD16 (mặc định là 512 bytes).

3. Quá trình truyền dữ liệu giữa Host và SD Card

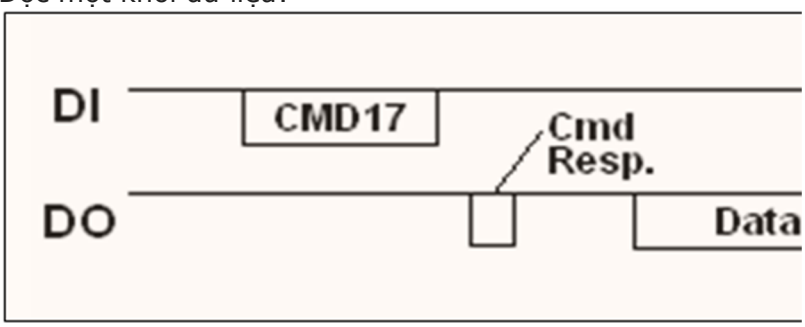
+ Trong quá trình trao đổi dữ liệu, một hoặc nhiều khối dữ liệu sẽ được gửi hoặc nhận sau đáp ứng của lệnh.
+ Một khối dữ liệu được vận chuyển giống như một gói dữ liệu bao gồm 3 trường: Data Token, Data Block, CRC.

Data Token	Data block	CRC

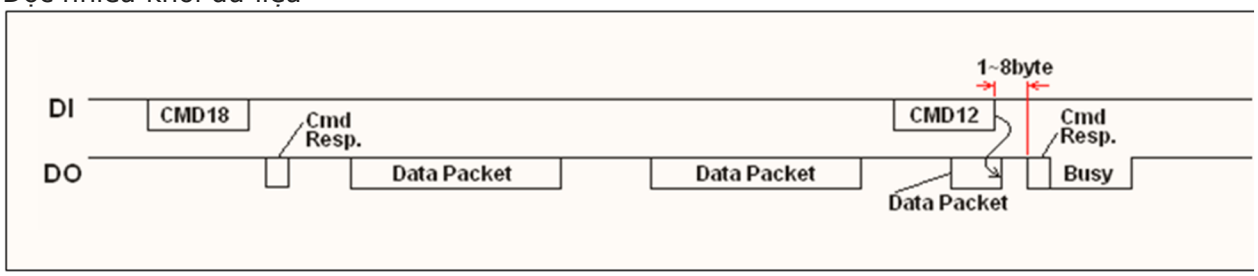
Data Token:
Có 3 dạng Data Token cho 3 nhóm lệnh khác nhau như bên dưới:

1	1	1	1	1	1	1	0
1	1	1	1	1	1	0	0
1	1	1	1	1	1	0	1

Data Token cho CMD17/18/24
Data Token cho CMD25
Data Token cho CMD25 (ngừng chuyển dữ liệu)
Đọc một khối dữ liệu:



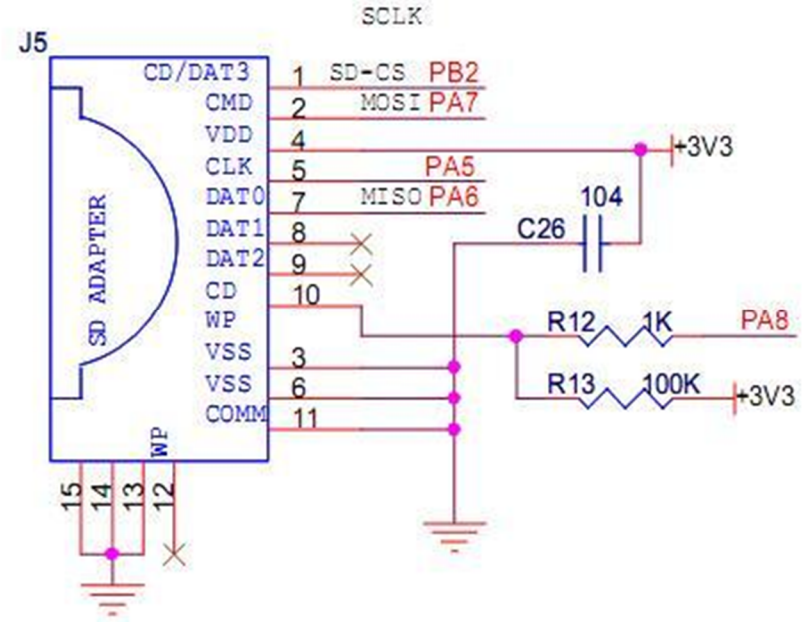
Hình 3. Đọc một khối dữ liệu
DI: MOSI
DO: MISO
Quá trình đọc một khối dữ liệu
+ Tham số (argument) trong lệnh CMD17 xác định địa chỉ bắt đầu của khối dữ liệu cần đọc.
+ Khi lệnh CMD17 được chấp nhận, hoạt động đọc dữ liệu bắt đầu diễn ra, dữ liệu sẽ được gửi đến Host.
+ Sau khi Host nhận được một Data Token thích hợp, bộ điều khiển sẽ bắt đầu nhận dữ liệu và 2 bytes CRC theo sau Data token.
+ Host phải nhận 2 bytes CRC mặc dù có thể không dùng đến nó.
+ Nếu có lỗi xuất hiện, thì Error token sẽ được nhận thay vì Data packet.
Đọc nhiều khối dữ liệu



Hình 4. Đọc nhiều khối dữ liệu
Quá trình đọc nhiều khối dữ liệu
+ Tham số trong lệnh CMD18 xác định địa chỉ bắt đầu của một dãy khối dữ liệu liên tiếp.

+ Khi lệnh CMD18 được chấp nhận, hoạt động đọc dữ liệu sẽ diễn ra, dữ liệu sẽ được gửi đến Host.
+ Sau khi Host nhận được Response thích hợp, bộ điều khiển sẽ bắt đầu nhận dữ liệu.
+ Hoạt động nhận dữ liệu chỉ kết thúc khi gửi lệnh CMD12, dữ liệu nhận được theo sau lệnh CMD12 không có ý nghĩa, do đó nó cần được bỏ qua trước khi nhận Respose cho lệnh CMD12.

4. Kết nối SD Card với giao diện SPI



Hình 5. SD Card kết nối với SPI1

Bài 3 :Lập trình định dạng FAT File trong SD Card với STM32

Tác giả: Phạm Văn Vang (pvvang2807@gmail.com)

Demo này dùng Driver MMC/SD Card (như trình bày ở phần trước) và bộ thư viện DOSFS để đọc file *.txt từ thư mục gốc của thẻ nhớ.
Để chạy demo này cần tạo một file *.txt rồi chép vào thư mục gốc. Demo chạy trên Kit OPENCMX-STM3210D.

1. Cấu trúc file chung của một SD Card

+ Hầu hết tất cả các ổ đĩa cứng đều có định dạng tương tự nhau: mỗi ổ đĩa được chia thành các phân vùng (partition), số lượng phân vùng tùy vào dung lượng của ổ đĩa, tối đa là 4 phân vùng. Mỗi phân vùng chứa nhiều Cluster, mỗi Cluster chứa nhiều Sector.
+ Khi một file được lưu vào ổ đĩa thì nó sẽ được lưu vào các Cluster, nếu một Cluster đã dùng để lưu một file nào đó thì nó không thể dùng để lưu 1 file khác mặt dù có thể file đó vẫn chưa chiếm hết Cluster đó, điều này gây ra lãng phí bộ nhớ.

Mark Boot Record (MBR)	Reserved Region	Partition 0	Partitton 1	Partition 2	Partition 3

Hình 1. Cấu Trúc Của Ổ Đĩa

+ Sector đầu tiên của ổ đĩa là MBR, nó chứa ***Executable Code*** và thông tin của ***4 phân vùng*** (partition)như bên dưới:

Bảng 1. Mark Boot Record

Offset	Description	Size

000h	Executable Code (Boots Computer)	446 bytes
1Beh	1st Partition Entry	16 bytes
1Ceh	2nd Partition Entry	16 bytes
1DEh	3rd Partition Entry	16 bytes
1EEh	4th Partition Entry	16 bytes
1FEh	Executable Marker (55h AAh)	2 bytes

+ Thông tin của mỗi phân vùng được chứa trong 16 bytes , bao gồm các trường:

Bảng 2. Thông tin của một phân vùng

Offset	Description	Size
00h	Current State of Partition (00h=Inactive, 80h=Active)	1 Byte
01h	Beginning of Partition - Head	1Byte
02h	Beginning of Partition - Cylinder/Sector (See Below)	2 Bytes
04h	Type of Partition (See List Below)	1 Byte
05h	End of Partition - Head	1 Byte
06h	End of Partition - Cylinder/Sector	2 Bytes
08h	**Starting sector of the partition**	**4 Bytes**
0Ch	Number of Sectors in the Partition	4 Bytes

+ Thông tin quan trọng ở đây là **Starting sector of the partition**, nó cũng chính là địa chỉ của ***Boot Sector*** của mỗi phân vùng.
+ Muốn giao tiếp được với SD Card cần tìm và đọc được Sector này.

2. Cấu trúc file của mỗi phân vùng

Phân vùng là nơi mà ta cần tìm ra để có thể giao tiếp đọc-ghi file lên SD card.
Mỗi phân vùng có cấu trúc lưu trữ thông tin chung như bên dưới:

Contents	Boot sector	FS Information Sector (FAT32 only)	More Reserved Sector (optional)	File Allocation Table #1	File Allocation Table #2	Root Directory (FAT16/12 Only)	Data region (directories and file)
Size in sector	Number of reserved sectors (Reserved sectors)			(number of FATs)*(sectors per FAT)		(number of root entries*32)/Bytes per sector	NumberOfClusters *SectorsPerCluster

Hình 2. Cấu trúc chung của mỗi phân vùng

* Cấu trúc file của phân vùng được tổ chức theo dạng FAT (File Allocation Table). Bao gồm 4 phần:

a. Reserved sectors: nằm ở vùng đầu tiên của một phân vùng. Sector đầu tiên của Reserved sectors là ***Boot sector***, nó chứa tất cả các thông tin về phân vùng.
b. FAT Region: nó gồm hai bản copy của ***File Allocation Table***, bản thứ hai rất hiếm khi dùng đến. Nó được định vị tới vùng dữ liệu (Data Region), sẽ đề cập ở phần sau.
c. Root Directory Region: nó là ***một bảng thư mục(directory table)*** chứa thông tin về các files và các thư mục trong thư mục gốc.
d. Data Region: đây là vùng thật sự chứa các files dữ liệu và các thư mục con.

* **Chi tiết về các vùng quan trọng cần nắm rõ**

Boot sector

+ Có kích thước 1 sector, nằm đầu tiên của mỗi phân vùng (không phải là sector đầu tiên

của ổ đĩa), chứa các thông tin quan trọng về phân vùng.
+ Bao gồm các trường như bảng dưới:

Bảng 3. Thông tin chứa trong Boot sector

Offset	Decription	Size
00h	Jump Code + NOP	3 Bytes
03h	OEM Name	8 Bytes
0Bh	Bytes Per Sector	2 Bytes
0Dh	Sectors Per Cluster	1 Byte
0Eh	Reserved Sectors	2 Bytes
10h	Number of Copies of FAT	1 Byte
11h	Maximum Root Directory Entries	2 Bytes
13h	Number of Sectors in Partition Smaller than 32MB	2 Bytes
15h	Media Descriptor (F8h for Hard Disks)	1 Byte
16h	Sectors Per FAT	2 Byte
18h	Sectors Per Track	2 Bytes
1Ah	Number of Heads	2 Bytes
1Ch	Number of Hidden Sectors in Partition	4 Bytes
20h	Number of Sectors in Partition	4 Bytes
24h	Logical Drive Number of Partition	2 Bytes
26h	Extended Signature (29h)	1 Byte
27h	Serial Number of Partition	4 Bytes
2Bh	Volume Name of Partition	11 Bytes
36h	FAT Name (FAT16)	8 Bytes
3Eh	Executable Code	448 Bytes
1FEh	Executable Marker (55h AAh)	2 Bytes

+ Như nói ở trên Boot sector này chứa tất cả các thông tin ta cần phải biết để giao tiếp với SD card như: số sector dự trữ, số byte trong 1 sector, số sector trong 1 cluster,số bảng FAT copy (thường là 1)…

File Allocation Table

+ Là một danh sách các mục nhập ánh xạ đến mỗi Cluster trong vùng dữ liệu.
Khi ghi một file vào SD Card, trường hợp dung lượng file lớn hơn 1 cluster thì file sẽ được lưu trong nhiều cluster và chú ý là các cluster này có thể không liên tiếp nhau; do đó bảng FAT này giúp ta tìm ra cluster tiếp theo chứa file.
+ Nó gồm các mục nhập mỗi mục nhập chứa một trong 5 thông tin sau:
1. Số của Cluster tiếp theo trong dãy Cluster của file dữ liệu.
2. Kết thúc chuỗi các Cluster trong file dữ liệu.
3. Mục nhập đánh dấu một Cluster xấu.
4. Mục nhập đánh dấu một Cluster dự trữ.
5. Giá tri 0 chỉ ra một Cluster chưa sử dụng.

+ Hai mục nhập đầu tiên chứa hai giá trị đặc biệt
- Mục nhập thứ nhất chứa bản copy của Media Decriptor.
- Mục nhập thứ hai chứa end-of-cluster-chain marker

Bởi vì hai Cluster đầu tiên chứa giá trị đặc biệt thành ra không có Cluster 0 và 1. Cluster đầu tiên theo sau Root directory là Cluster 2.

Bảng 4. Giá trị của các mục nhập trong FAT

FAT12	FAT16	FAT32	Mô tả
0x000	0x0000	0x00000000	Cluster chưa dùng
0x001	0x0001	0x00000001	Giá trị dự trữ

0x002–0xFEF	0x0002–0xFFEF	0x00000002–0x0FFFFFEF	Cluster đã dùng, giá trị này chỉ đến Cluster tiếp theo
0xFF0–0xFF6	0xFFF0–0xFFF6	0x0FFFFFF0–0x0FFFFFF6	Giá trị dự trữ
0xFF7	0xFFF7	0x0FFFFFF7	Cluster xấu hay dự trữ
0xFF8–0xFFF	0xFFF8–0xFFFF	0x0FFFFFF8–0x0FFFFFFF	Cluster cuối của file

* **Root Directory Region**

Là một loại đặc biệt của file dùng để trình bày một thư mục, có cấu tạo theo dạng bảng. Mỗi thư mục hay file lưu trữ trong nó được tạo thành bởi một mục nhập 32 bytes chứa các thông tin như tên, phần mở rộng, thuộc tính…

Bảng 5. Cấu trúc của Directory Table

Byte thứ	0 -7	8 - 10	11-25	26 – 27	28 - 31
File 1	Tên file	Phần mở rộng	Thuộc tính, ngày	Cluster đầu	Kích thước
File 2	Tên file	Phần mở rộng	Thuộc tính, ngày	Cluster đầu	Kích thước
…	…	…	…	…	…
File n	Tên file	Phần mở rộng	Thuộc tính, ngày	Cluster đầu	Kích thước

Thông tin cần thiết ở đây là cluster bắt đầu của file hay thư mục con.

Data Region

Chứa dữ liệu cùa file, bao gồm nhiều cluster. Chú ý là mỗi cluster chỉ chứa dữ liệu của một file, không có trường hợp một cluster chứa dữ liệu của nhiều file khác nhau.

3. Các bước để tìm và đọc file (chỉ dừng lại ở thư mục gốc)

Từ những phần trình bày ở trên ta đưa ra các bước để tìm và đọc một file trong thư mục gốc của SD Card

1) Đọc MBR: (sector đầu tiên)

Từ MBR ta xác định được địa chỉ của Sector bắt đầu của phân vùng cần tìm (chú ý là với ổ đĩa có dung lượng nhỏ thì chỉ có 1 phân vùng (partition 0)). Đây cũng chình là Boot Sector. Xem ***Bảng 1***

2) Đọc Boot Sector:

Từ Boot Sector ta xác định được các thông tin cần thiết về cấu trúc ổ đĩa như chỉ ra ở ***Bảng 3.***

Xác định

- Sector bắt đầu của Bảng FAT.
- Địa chỉ sector bắt đầu của Root Directory (đây là phần chủ yếu).

3) Từ Root Directory xác định các thư mục con và các file chứa trong thư muc gốc (ở đây chỉ dừng lại ở các file nằm ở thư mục gốc).

Thư mục gốc chứa tất cả 512 mục nhập (file hoặc thư mục con), mỗi mục nhập có kích thước 32 bytes như ***Bảng 5*** (-> Root Directory có kích thước 32 sectors)

4) Khi tìm được một file trong thư mục gốc nó sẽ cho ta biết địa chỉ của Cluster bắt đầu của File***. (startclus) Bảng 5***

Từ ***startclus*** này ta tìm được:

+ Sector bắt đầu của File theo công thức: (dựa vào ***Hình 2***)

Start_sec = Start_sec_par + Reserved sectors + 2 Secperfat + Rootentries* 32/ 512 + (startclus – 2) *Secperclus*

Trong đó:

- Start_sec : sector bắt đầu của file ứng với ***startclus***
- Start_sec_par: sector bắt đầu của phân vùng (chính là Boot Sector)
- Reserved sectors: số sector dự trữ trong phân vùng (nằm ở FAT region)
- Secperfat: số sector có trong bảng FAT, vì có 2 bảng FAT nên phải nhân 2
- Rootentries: số lượng thư mục gốc, mỗi thư mục có độ lớn 32 bytes

- Secperclus :Số lượng Sector chứa trong 1 cluster.

Chú ý: những thông số này có ở ***bước 2***

5) Xác định cluster tiếp theo chứa file (trường hợp file có dung lượng lớn phải chứa trong các cluster khác nhau)

Dựa vào bảng FAT như đã trình bày phần trên.

Tóm tắt:

+ Phần trên trình bày cấu trúc lưu trữ file của 1 ổ đĩa cũng như là SD Card.

+ Để giao tiếp với SD Card các bạn có thể dùng 2 bộ thư viện đã hỗ trợ sẵn các hàm thực hiện các bước ở trên: thư viện DOSFS và thư việnFATFS