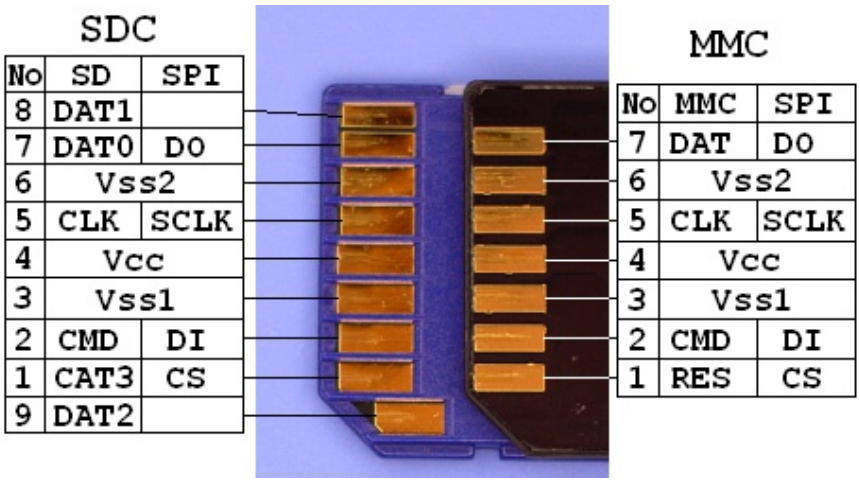
How to Use MMC/SDC

Update: November 3, 2010

The *Secure Digital Memory Card* (SDC below) is the de facto standard memory card for mobile equipments. The SDC was developped as upper-compatible to *Multi Media Card* (MMC below) so that the SDC compleant equipments can also use an MMC with a few considerations. There are also reduced size versions, such as *RS-MMC*, *miniSD* and *microSD*, with same function. The MMC/SDC has a microcontroller in it, the flash memory controls (erasing, reading, writing and error controls) are completed inside of the memory card. The data is transferred between the memory card and the host controller as data blocks in unit of 512 bytes, so that it can be seen like a generic hard disk drive from view point of upper level layers. The currentry defined file system for the memory card is FAT12/16 with FDISK patitioning rule. The FAT32 is defined for only high capacity (>= 4G) cards.

This page describes the basic knowledge and miscellaneous things that I become aware, on using MMC/SDC with small embedded system. I believe that this information must be a useful getting started notes for people who is going to begin to enjoy MMC/SDC.

Pinout



miniSD | microSD

Right photo shows the contact surface of the SDC/MMC. The MMC has seven contact pads and the SDC has nine contact pads that two additional contacts to MMC. The three of the contacts are assigned for power supply so that the number of effective signals are four (MMC) and six (SDC). Therfore the data transfer between the host and the card is done via a synchronous serial interface.

The working supply voltage range is indicated in a special function register and it should be read and comfirmed the operating voltage range. However, the supply voltage can be fixed to a proper value because the MMC/SDC works at supply voltage of *2.7 to 3.6 volts*. The current consumption can reach up to several ten milliamperes, so that the host system should consider to supply 100 miliamperes at least.

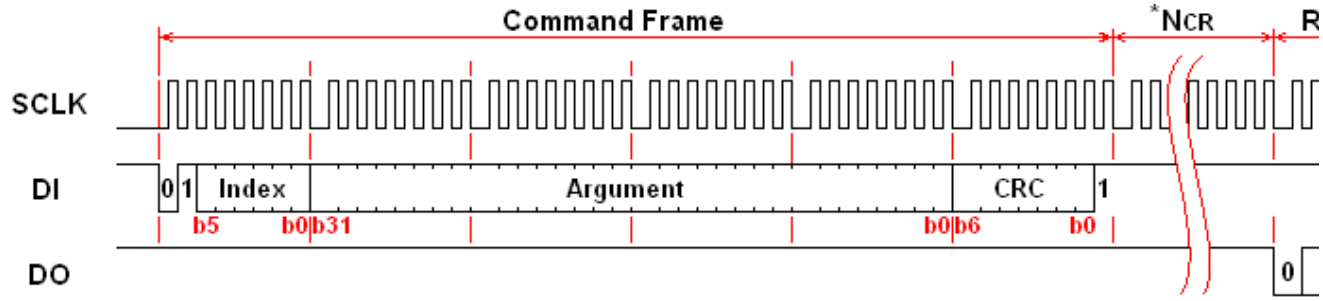
SPI Mode

The SPI mode is an alternative operating mode that defined to use the memory cards without MMC/SDC native host interface. The communication protocol for the SPI mode is simple compared to the its native mode. The MMC/SDC can be attached to the most microcontrollers via a generic SPI interface or GPIO ports. Therefore the SPI mode is suitable for low cost embedded applications. Especialy, there is no reason to attempt to use native mode with a cheap microcontroller that has no native MMC/SDC interface. For SDC, the 'SPI mode 0' is defined for its SPI mode. For MMC, it is not the SPI timing, both latch and shift actions are defined with rising edge of the SCLK, but it seems work in SPI mode 0 in SPI mode. Thus the *SPI Mode 0* (CPHA=0, CPOL=0) is the proper setting for MMC/SDC interface, but SPI mode 3 also works as well in most case.

Command and Response

In SPI mode, the data direction on the signal line is fixed and the data is transferred in byte oriented serial communication. The command frame from host to card is a fixed length (six bytes) packet that shown below. When a command frame is

transmitted to the card, a response to the command (R1, R2 or R3) will able to be read from the card. Because data transfer is driven by serial clock generated by host, the host must continue to read bytes, send a 0xFF and get the received data, until receive any valid response. The command response time (NCR) is 0 to 8 bytes for SDC, 1 to 8 bytes for MMC. The CS signal must be held low during a transaction (command, response and data transfer if exist). The CRC field is optional in SPI mode, but it is required as a bit field to compose a command frame. The DI signal must be kept high during read transfer.



SPI Command Set

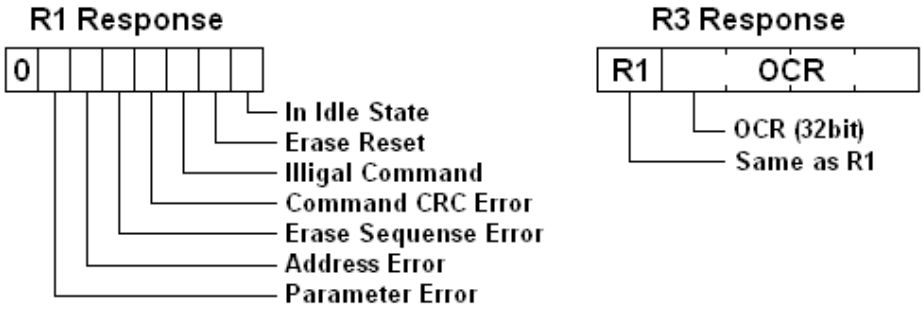
Each command is expressed in abbreviation like GO_IDLE_STATE or CMD<n>, <n> is the number of the command index and the value can be 0 to 63. Following table describes only commands that to be usually used for generic read/write and card initialization. For details on all commands, please refer to spec sheets from MMCA and SDCA.

Command Index	Argument	Response	Data	Abbreviation	Descri
CMD0	None(0)	R1	No	GO_IDLE_STATE	Softwar reset.
CMD1	None(0)	R1	No	SEND_OP_COND	Initiate initializ process
ACMD41(*1)	*2	R1	No	APP_SEND_OP_COND	For onl SDC. Initiate initializ process
CMD8	*3	R7	No	SEND_IF_COND	For onl SDC V2 Check voltage range.

CMD9	None(0)	R1	Yes	SEND_CSD	Read C register
CMD10	None(0)	R1	Yes	SEND_CID	Read C register
CMD12	None(0)	R1b	No	STOP_TRANSMISSION	Stop to data.
CMD16	Block length[31:0]	R1	No	SET_BLOCKLEN	Change block si
CMD17	Address[31:0]	R1	Yes	READ_SINGLE_BLOCK	Read a block.
CMD18	Address[31:0]	R1	Yes	READ_MULTIPLE_BLOCK	Read multipl blocks.
CMD23	Number of blocks[15:0]	R1	No	SET_BLOCK_COUNT	For only MMC. Define number blocks t transfe with ne multi-bl read/wr comma
ACMD23(*1)	Number of blocks[22:0]	R1	No	SET_WR_BLOCK_ERASE_COUNT	For only SDC. D number blocks t pre-era with ne multi-bl write comma
CMD24	Address[31:0]	R1	Yes	WRITE_BLOCK	Write a block.
CMD25	Address[31:0]	R1	Yes	WRITE_MULTIPLE_BLOCK	Write multipl blocks.
CMD55(*1)	None(0)	R1	No	APP_CMD	Leading comma ACMD< comma
CMD58	None(0)	R3	No	READ_OCR	Read O

*1:ACMD<n> means a command sequense of CMD55-CMD<n>.
*2: Rsv(0)[31], HCS[30], Rsv(0)[29:0]
*3: Rsv(0)[31:12], Supply Voltage(1)[11:8], Check Pattern(0xAA)[7:0]

SPI Response



There are three command response formats, *R1*, *R2* and *R3*, depends on the command index. A byte of response R1 is returned for most commands. The bit field of R1 response is shown in right image, the value 0x00 means successful. When any error occured, corresponding status bit in the response will be set. The R3 response (R1 and trailing 32-bit OCR) is for only CMD58.

Some commands take a time longer than NCR and it responds *R1b*. It is an R1 response followed by busy flag (DO is held low as long as internal process is in progress). The host controller should wait for end of the process until 0xFF is received.

Initialization Procedure for SPI Mode

After power on reset, MMC/SDC enters its native operating mode. To put it SPI mode, follwing procedure must be performed like this flow.

Power ON (Insersion)

After supply voltage reached 2.2 volts, wait for a millisecond at least. *Set DI and CS high and apply more than 74 clock pulses to SCLK* and the card will go ready to accept native commands.

Software Reset

Set SPI clock rate between 100kHz and 400kHz and then send a *CMD0* with CS low to reset the card. The card samples CS signal when a CMD0 is received. If the CS signal is low, the card enters SPI mode. Since the CMD0 must be sent as a native command, the CRC field must have a valid value. When once the card enters SPI mode, the CRC feature is disabled and the CRC is not checked, so that command transmission routine can be written with the hardcorded CRC value that valid for

only CMD0 and CMD8. When the CMD0 is accepted, the card will enter idle state and respond R1 response with In Idle State bit (0x01). The CRC feature can also be switched with CMD59.

Initialization

In idle state, the card accepts only CMD0, CMD1, ACMD41 and CMD58. Any other commands will be rejected. In this time, read OCR with CMD58, check working voltage range indicated by the OCR. In case of the system sypply voltage is out of working voltage range, the card must be rejected. Note that all cards work at voltage range of 2.7 to 3.6 volts at least. The card initiates initialization when a *CMD1* is received. To poll end of the initialization, the host controller must send CMD1 and check the response until end of the initialization. When the card is initialized successfuly, In Idle State bit in the R1 response is cleared (R1 resp changes 0x01 to 0x00). The initialization process can take *hundreds of milliseconds* (large cards tend to longer), so that this is a consideration to determin the time out value. After the In Idle State bit cleared, generic read/write commands will able to be accepted.

Because *ACMD41* instead of CMD1 is recommended for SDC, trying ACMD41 first and retry with CMD1 if rejected, is ideal to support both type of the cards.

The SPI clock rate should be changed to fast as possible to optimize the read/write performance. The TRAN_SPEED field in the CSD indicates the maximum clock rate of the card. The maximum clock rate is 20MHz for MMC, 25MHz for SDC in most case. Note that the clock rate will able to be fixed to 20/25MHz in SPI mode because there is no open-drain condition that restricts the clock rate.

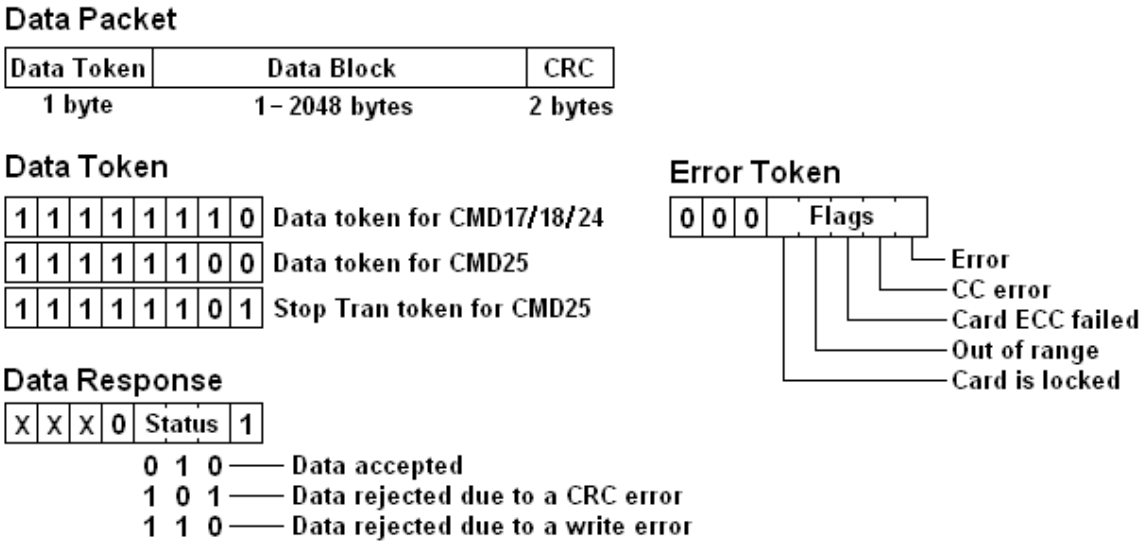
The initial block length can be set larger than 512 on 2GB card, so that the block size should be re-initialized with CMD16 if needed.

How to support SDC Ver2 and high capacity cards

After the card enters idle state with a CMD0, send a *CMD8* with argument of 0x000001AA and correct CRC prior to initialization process. When the CMD8 is rejected with an illigal command error (0x05), the card is SDC V1 or MMC. When the CMD8 is accepted, R7 response (R1(0x01) and trailing 32 bit data) will be returned. The lower 12 bits in the return value 0x1AA means that the card is SDC V2 and it can work at voltage range of 2.7 to 3.6 volts. If not the case, the card must be rejected. And then initiate initialization with ACMD41 with HCS (bit 30). After the initialization completed, read OCR and check CCS (bit 30) in the OCR. When it is set, subsequent data read/write operations that described below are commanded in block address insted of byte address. The block size is always fixed to 512 bytes.

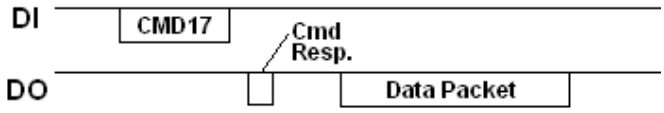
Data Transfer

Data Packet and Data Response



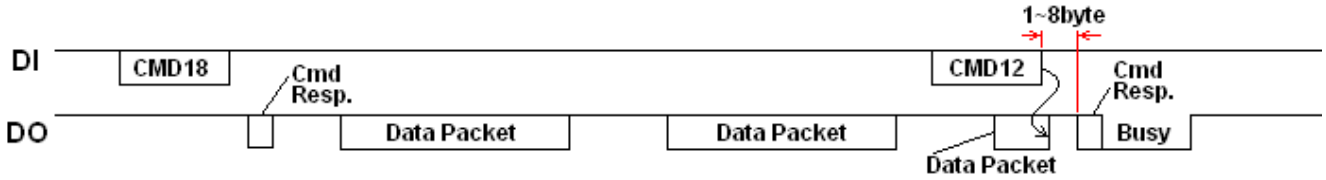
In a transaction with data transfer, one or more data blocks will be sent/received after command response. The data block is transferred as a data packet that consist of Token, Data Block and CRC. The format of the data packet is showin in right image and there are three data tokens. As for Stop Tran token that means end of multiple block write, it is used in single byte without data block and CRC.

Single Block Read



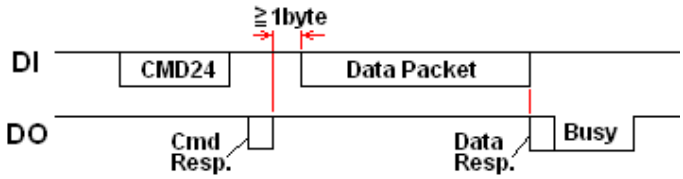
The argument specifies the location to start to read *in unit of byte or block*. The sector address specified by upper layer must be scaled properly. When a CMD17 is accepted, a read operation is initiated and the read data block will be sent to the host. After a valid data token is detected, the host controller receives following data field and two byte CRC. The CRC bytes must be flushed even if it is not needed. If any error occured during the read operation, an error token will be returned instead of data packet.

Multiple Block Read



The Multiple Block Read command reads multiple blocks in sequense from the specified address. When number of transfer blocks has not been sepecified before this command, the transaction will be initiated as an open-ended multiple block read, the read operation will continue until stopped with a CMD12. The received byte immediataly following CMD12 is a stuff byte, it should be discarded before receive the response of the CMD12.

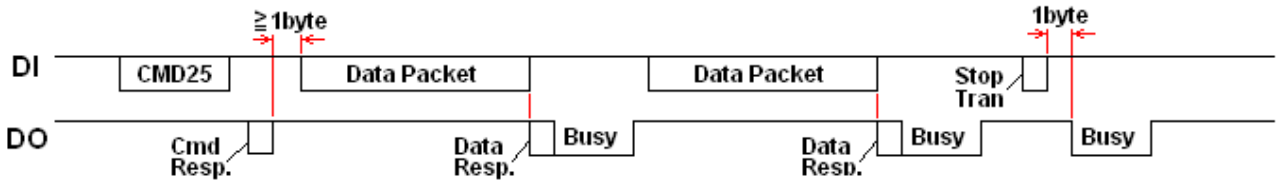
Single Block Write



When a write command is accepted, the host controller sends a data packet to the card after a byte space. The packet format is same as Block Read command. The CRC field can have any invalid value unless the CRC function is enabled. When a data packet has been sent, the card responds a Data Response immediataly following the data packet. The data response trails a busy flag to process the write operation. Most cards cannot change write block size and it is fixed to 512.

In principle of the SPI mode, the CS signal must be asserted during a transaction, however there is an exception to this rule. When the card is busy, the host controller can deassert CS to release SPI bus for any other SPI devices. The card will drive DO signal low again when reselect it during internal process is in progress. Therefore a preceding busy check (wait ready immediataly before command and data packet) instead of post wait can eliminate waste wait time. In addition the internal process is initiated a byte after the data response, this means eight clocks are required to initiate internal write operation. The state of CS signal during the eight clocks is negligible so that it can done by bus release process described below.

Multiple Block Write

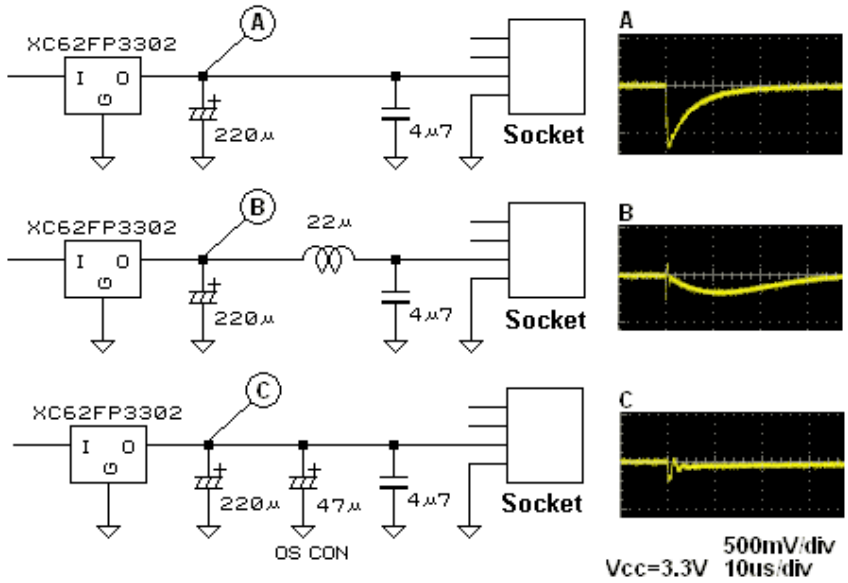


The Multiple Block Read command writes multiple blocks in sequense from the specified address. When number of transfer blocks has not been sepecified prior to this command, the transaction will be initiated as an *open-ended multiple block write*, the write operation will continue until it is terminated with a Stop Tran token. The busy flag will appear on the DO line a byte after the Stop Tran token. As for SDC, the multiple block write transaction must be terminated with a Stop Tran token independent of the transfer type, pre-defined or open-ended.

Reading CSD and CID

These are same as Single Block Read except for the data block length. The CSD and CID are sent to the host as *16 byte data block*. For details of the CMD, CID and OCR, please refer to the MMC/SDC specs.

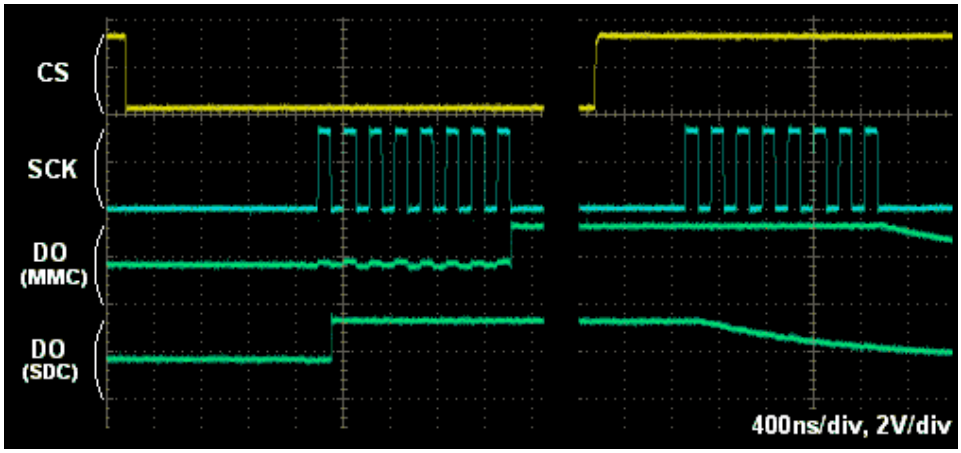
Cosideration to Bus Floating and Hot Insertion



Any signals that can be floated should be pulled low or high properly via a resister. This is a generic design rule on MOS devices. Because DI and DO are normally high, they should be pulled-up. According to SDC/MMC specs, from 50k to 100k ohms is recommended to the value of pull-up registers. However the clock signal is not mentioned in the SDC/MMC specs because it is always driven by host controller. When there is a possibility of floating, it should be pulled to the normal state, low.

The MMC/SDC can hot insertion/removal but some considerations to the host circuit are needed to avoid an incorrect operation. For example, if the system power supply (Vcc) is tied to the card socket directly, the Vcc will dip at the instant of contact closed due to a charge current to the capacitor that built in the card. 'A' in the right image is the scope view and it shows that occureing a voltage dip of about 600 millivolts. This is a sufficient level to trigger a brown out detector. 'B' in the right image shows that an inductor is inserted to block the surge current, the voltage dip is reduced to 200 millivoits. A low ESR capacitor, such as OS-CON, can eliminate the voltage dip dratiscally like shown in 'C'. However the low ESR capacitor can cause an oscillation of LDO regulator.

Cosideration on Multi-slave Configuration



In the SPI bus, each slave device is selected with separated CS signals, and plural devices can be attached to an SPI bus. Generic SPI slave device drives/releases its DO signal by CS signal asynchronously to share an SPI bus. However MMC/SDC drives/releases DO signal in *synchronising to the SCLK*. This means there is a posibility of bus conflict with MMC/SDC and any other SPI slaves that attached to an SPI bus. Right image shows the drive/release timing of the MMC/SDC (the DO signal is pulled to 1/2 vcc to see the bus state). Therefore to make MMC/SDC release DO signal, the master device must send a byte after CS signal is deasserted.

Optimization of Write Performance

Most MMC/SDC employs NAND Flash Memory as a memory array. The NAND flash memory is cost effective and it can read/write *large* data fast, but on the other hand, there is a disadvantage that rewriting a *part* of data is inefficient. Generally the flash memory requires to erase existing data before write a new data, and minimum unit of erase operation (called erase block) is larger than write block size. The typical NAND flash memory has a block size of 512/16K bytes for write/erase operation, and recent monster card employs large block chip (2K/128K). This means that rewriting entire data in the erase block is done in the card even if write only a sector (512

bytes).

Benchmark

I examined the read/write performance of some MMC/SDC with a cheap 8 bit MCU (ATmega64 @9.2MHz) on the assumption that an embedded system *with limited memory size*. For reason of memory size, `write()` and `read()` ware performed in 2048 bytes at a time. The result is: Write: 77kB/sec, Read: 328kB/sec on the 128MB SDC, Write: 28kB/sec, Read: 234kB/sec on the 512MB SDC and Write: 182kB/sec, Read: 312kB/sec on the 128MB MMC.

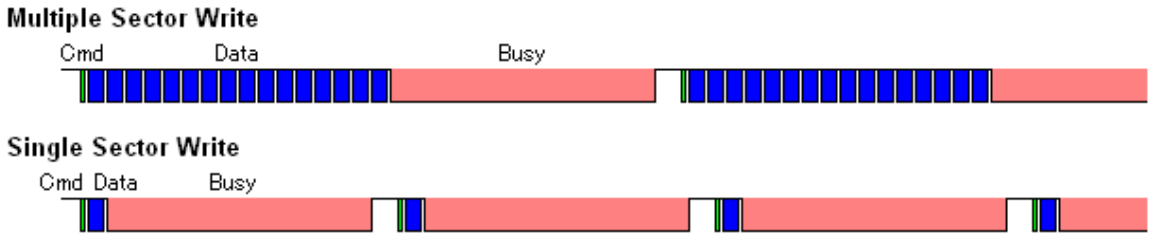
Therefor the write performance of the 512MB SDC was very poor that one third value of 128MB SDC. Generally the read/write performance of the mass storage device increases proportional to its recording density, however it sometimes appears a tendency of opposite on the memory card. As for the MMC, it seems to be several times faster than SDC, it is not bad performance. After that time, I examined some SDCs supplied from different makers, and I found that PQI's SDC was as fast as Hitachi's MMC but Panasonic's and Toshiba's one was very poor performances.

Erase Block Size

To analys detail of write operation, busy time (number of polling cycles) after sent a write data is typed out to console in the low level disk write function. Multiple numbers on a line indicates data blocks and a Stop Tran token that issued by a multiple block write transaction.

In resulut of the analysis, there is a different of internal process between 128MB SDC and 512MB SDC. The 128MB SDC rewrites erase block at end of the mutiple block write transaction. The 512MB SDC seems have 4K bytes data buffer and it rewrites erase block every 4K bytes boundary. Therefor it cannot compared directly but the processing time of rewriting an erase block can be read 3800 for 128MB SDC and the 512MB SDC taeks 30000 that 8 times longer than 128MB SDC. Judging from this resulut, it seems the 128MB SDC uses a small block chip and the 512MB SDC uses a large block or MLC chip. Ofcourse the larger block size decreases the performance on pertial block rewriting. In 512MB SDC, only an area that 512K bytes from top of the memory is relatively fast. This can be read from write time in `close()`. It might any special processing is applied to this area for fast FAT accsess.

Improving Write Performance



To avoid this bottleneck and increase the write performance, number of blocks per write transaction must be large as possible. Of course all layers between the application and the media must support multiple sector write feature. For low level SDC/MMC write function, it should inform number of write sectors to the card prior to the write transaction for efficient internal write process. This method called `pre-defined multiple block write'. The pre-definition command is not the same between MMC (CMD23) and SDC (ACMD23).

The memory cards are initially patitioned and formatted to align the allocation unit to the erase block. When re-patition or re-format the memory card with a device that not compliant to MMC/SDC (this is just a PC) with no care, the optimization will be broken and the write performance might be lost. I tried to re-format 512MB SDC in FAT32 with a PC, the write performance measured in file copy was lowerd to one several. Therefore the re-formatting the card should be done with MMC/SDC compliant equipments rather than PC.

Links

- MMCA - Multimedia Card Association
- SDA - SD Card Association
- SDHC Physical Layer Spec.
- About SPI
- Generic FAT file system module with sample code to control *MMC/SDSC/SDHC*