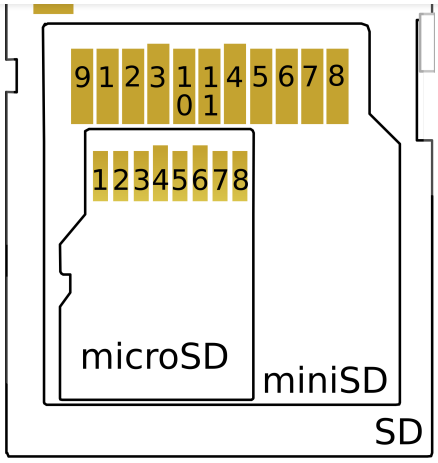
Interfacing Microcontrollers with SD Card

The secure digital card (SD) is a low cost, non-volatile memory card format developed by the SD Card Association. Since its inception back at the start of the century, the demand for this medium-sized, energy and space-efficient, the memory storage device has been growing at a fast rate. Therefore, to meet the market requirements, the SDA was set up as a non-profit organization to promote and create SD Card standards. There are various topics related to the SD card such as the different device families, speed classes, smart cards, card security and so on and it is used in various markets like digital cameras, personal computers, and embedded systems. Some of the standard variations include SD, SDHC, SDXC, SD-ultra high speed etc. The microSD is the miniaturized SD memory card format with a small form factor and is widely used in various electronic devices. What we are going to learn is the use of SD cards in an embedded system. To be specific, we will be dealing with the use of SD cards in small embedded systems.

Circuit and Interfacing

SD card has a native host interface apart from the SPI mode for communicating with master devices. The native interface uses four lines for data transfer where the microcontroller has SD card controller module and it needs separate license to use it. Since the SPI is a widely used protocol and it is available in most low-cost microcontrollers, the SPI mode is the widely used interface in low cost embedded systems. The working voltage range of SD family is 2.7V to 3.6V and this is indicated in the operation condition register (OCR).



PIN	SD	SPI
1	CD/DAT3	CS
2	CMD	D1
3	VSS1	VSS1
4	VDD	VDD
5	CLK	SCLK
6	VSS2	VSS2
7	DAT0	D0
8	DAT1	X
9	DAT2	X

SD Card pinout

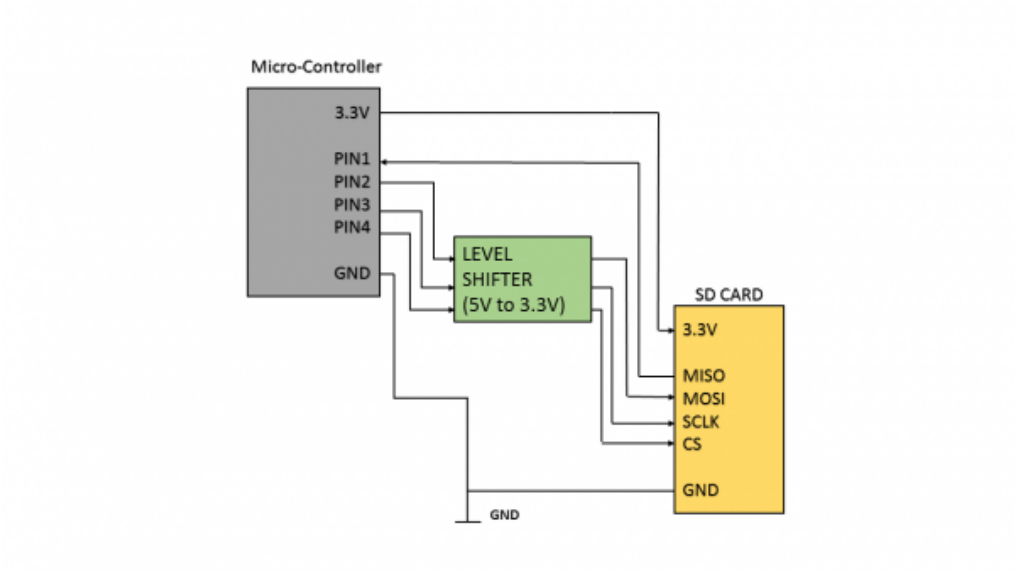
PIN	SD	SPI
1	DAT2	X
2	CD/DAT3	CS
3	CMD	DI
4	VDD	VDD
5	CLK	SCLK
6	VSS	VSS
7	DAT0	DO
8	DAT1	X

MicroSD Card pinout

Most micro-controllers use the SPI communication protocol to interface with the SD cards. The SD cards have a microcontroller that shows their availability to the master controller(microcontroller). The micro-controller sees the SD card as an addressable sector on which read/write functions are possible. Once the microcontroller is in the SPI mode, communication between the master and the slave is done via 4 pins viz. clock, chip select, data in and data out. It should be

Most development boards have a dedicated SD card slot. But to understand the connections, let us analyze this fairly simple circuit.

Circuit



If the logic level of the microcontroller is different than the SD card, level shifter needs to be used for converting the line voltages.
The MISO (master in serial out) pin should be connected to the SDI (serial data in) pin on the microcontroller.
The MOSI (master out serial in) pin should be connected to the SDO (serial data out) pin on the microcontroller.
The SCK (serial clock) pin should be connected to the SCK (serial clock) pin on the microcontroller.
The CS (chip select) pin should be connected to the corresponding CS pin on the microcontroller or on any digital I/O pin on the microcontroller. A common ground is provided.

IMPORTANT NOTE: While connecting the power supply, make sure that the supply is drawn from a 3.3V supply as a 5V supply would see your card go up in smoke.

Interfacing

Set the directions of each of the four pins correctly. While activating the SPI mode, an ideal configuration should be, mode(0,0) and the phase with input sampled at the middle of data out. Also, the clock frequency should be set in the range of 100 kHz and 400 kHz prior to initializing the card. Once the initialization is done, the clock can be set to a more desired frequency.

SD Commands

Next comes the tricky part, initializing the SD card and performing the raw data communication. A systematic approach to programming the software would make the task pretty easy.
But first, it is important to learn how the micro-controller activates the SD card. There are a fixed set of commands and responses, which must be followed to create a command to response structure in our program. The data is transmitted in a byte-oriented format with a definite length.
The following table shows the necessary **commands** to the card and the corresponding response from the card.

				reset
CMD1	None	R1	NO	Initiate initialization process
ACMD41	2	R1	NO	For only SDC. Initiate initialization process
CMD8	3	R7	NO	For only SDC V2. Check voltage range.
CMD9	None	R1	YES	Read CSD register
CMD10	None	R1	YES	Read CID register
CMD12	None	R1b	NO	Stop to read data
CMD16	Block Length(31:0)	R1	NO	Change R/W block size
CMD17	Address(31:0)	R1	YES	Read block
CMD18	Address(31:0)	R1	YES	Read multiple blocks
CMD23	Number of blocks(15:0)	R1	NO	For only MMC. Define number of blocks to transfer with next multi-block R/W command
ACMD23	Number of blocks(22:0)	R1	NO	For only SDC. Define number of blocks to pre-erase with next multi block write command
CMD24	Address(31:0)	R1	YES	Write a block
CMD25	Address(31:0)	R1	YES	Write multiple blocks
CMD55	None	R1	NO	Leading command of

number 64.

Example:

For CMD0: command number 0 + 64 = 64 = 0x40 in hexadecimal.

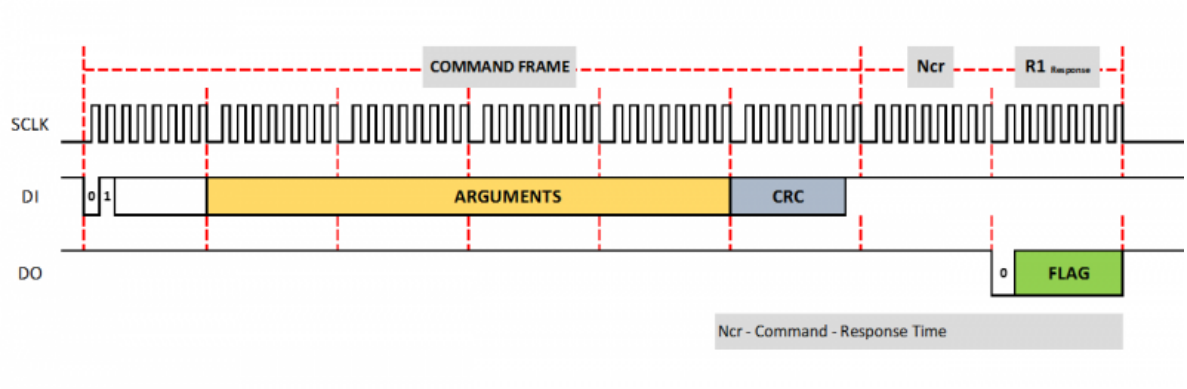
For CMD1: command number 1 + 64 = 65 = 0x41 in hexadecimal.

And so on.

This is followed by a set of four bytes which are known as the arguments. These arguments usually contain the address of a data or the length of a block.

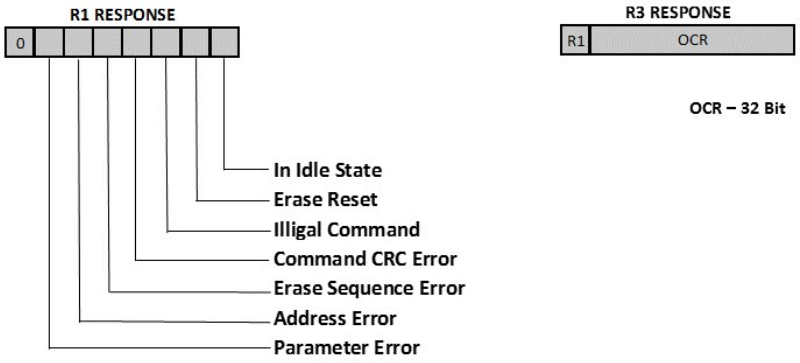
The last byte is the CRC (Cyclic Redundancy Check) Byte. Most commands in the SPI mode does not require a check byte if the CRC feature isn't enabled. For some commands like CMD0, the CRC is 0x95 and in most cases, a 0xFF is sent. Enabling the CRC requires you to send the correct check byte from the micro-controller. So, ensure whether the CRC feature is enabled or disabled.

A **command frame** looks like this-



The card will receive a command when the DO pin (Data Out) is driven high as shown above. The CS pin (Chip Select) must be driven high to low before sending the command and must be kept low during the process. The time between a command and its response is known as the command response time (NCR). As mentioned earlier, regarding the clock pulses from the micro-controller, the corresponding response byte sent back from the card to the microcontroller should be driven by the serial clock pulses of the micro-controller. There is a chance that the response byte from the card might get skipped by the microcontroller due to the absence of a driving clock pulse. Hence it is necessary to make sure that an 8-Bit clock pulse is sent to the Card soon after the Command Frame is sent

Furthermore, let us analyze the different **types of responses** that we get from the card and what they mean.



An R1 response, 0x01 means that the command sent prior to the response has resulted in the card going into an idle state.
A response byte 0x00 means that the command has been accepted and the card will be waiting for the proposed event to take place. If any other bits in an R1 response is set, it is the result of an error and it'll be down to the factor mentioned in each R1 response bit in the figure.

Initializing the SD Card

Now, as far as sending the commands are concerned, there is an order in which they must be sent. Only the commands, CMD0, CMD1, ACMD41, CMD58 and CMD59 will be accepted when the card is in its idle state. Sending any other command will likely to yield an illegal response.

TECHNICAL NOTE: After interfacing the card, the micro-controller must always send a set of bytes, which we will refer to as dummy bytes. One dummy byte is 0xFF. These dummy bytes have a simple yet significant purpose. Prior to the initialization, the card must know the frequency at which the data is being sent. By sending around 75

command is sent, it is a good practice to send at least one dummy byte. A logical explanation for this is that communication is driven by the clock pulse of the micro-controller. The clock pulse is sent only when the data buffer is filled. After every response is sent and prior to the next command or between command and response, the SCK will stop generating pulses due to an empty data buffer. To ensure the continual transmission of clock pulses between every command, fill the data buffer with a junk value such as a dummy byte. To create a dummy byte function with the number of loops as the argument.

The **first command** to be sent to the card is the **CMD0** command. The structure of every other command-response should be based on this model.

- Drive the DO pin high.
- Drive the CS pin from high to low.
- Send a dummy byte.
- Then send the following 6-byte command continuously
  First byte: 0x40
  Next four bytes: 0x00000000
  CRC byte: 0x95

- Send another dummy byte or as many as required, to generate the SCK giving time for the SD card to respond to the command request.
- Wait for the receive flag bit to set and then read the response from the SD card or create a loop to read the response a few number of times. The response should reach back within the command to response (Ncr) time. Or else, something must be wrong.

- The desired response is 0x01, meaning the card is in the idle state and we are good to go.

Send the **CMD8** command next to check the version of your SD card. The difference in the 6-byte commands are
First byte: 0x48
Next four bytes: 0x000001AA
CRC byte: 0x87

A 0x01 response means that you have a version 2 SD card. The 0x01 response is followed by the 4 bytes 0x00, 0x00, 0x01, 0xAA in the order of their transmission from the SD card which is, in fact, the argument you send in your command.

If the response is 0x05, it means the card is a version 1 or an MMC card. If the card is actually a version 2 SD card, then this response is the result of an illegal command. Also, the card is now in the idle state.

Once the above two commands (CMD0 and CMD8) are done, it is safe to say that our SD Card is working in good condition and ready for data Read/Write.
Additionally, just to ensure whether the SD Card is functioning in the correct working voltage, send the **CMD58** Command.

Next, we must initiate the i**nitialization** process. For this send a **CMD1** command and wait for response 0x00, meaning the idle state bit is cleared.
If you are using an SDC or for the purpose of creating a general code, it is advisable to send an **ACMD41** command first to begin the initialization process and then send the **CMD1** command if the **ACMD41** is rejected.
Either way, the initialization process might take a few hundred milliseconds, so iterating a few number of times to check for the response byte would be advisable.

Suppose the response of CMD8 resulted in 0x05, a few considerations have to be made regarding the initialization procedure. Version 1 cards may not support the ACMD41 command. And even if they do support the ACMD41 command, send the CMD55 command and check for the response, which will be 0x05 instead of 0x01 and then instead of sending the CMD41 command, send a CMD1 command to complete the initialization process.

Read/Write SD Card

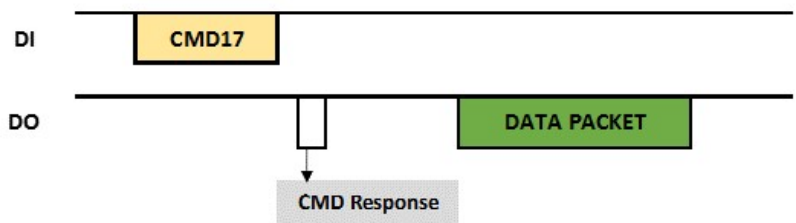
block.
For this, send **CMD16** command. The arguments for this command should specify the block length.
Example: To set the block length to 512 bytes, send the 4-byte argument as 0x00000200.
Send a 0xFF as the CRC byte and wait for a 0x00 response byte.

Regarding the commands for the read/write, the general structure for sending the commands is the same, but with a few additional details as well.

The 4-byte argument of every command will contain the specified address of the sector from where the data is being read or written to.

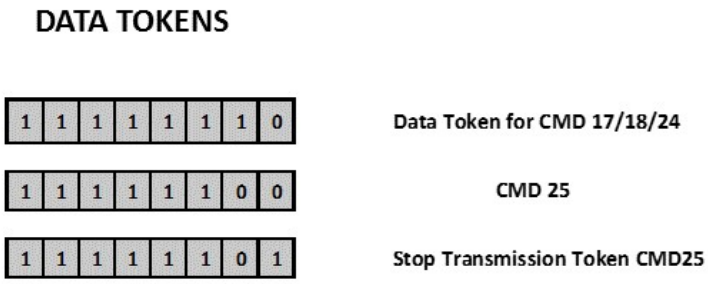
Send the **CMD17** command for a s**ingle read block** and wait for the expected response 0x00. Once that is done, wait for the SD card to send another response, 0xFE. This response, transmitted by the SD card, is an indication for the start of continuous transmission of a block of data. At the end of this, send a dummy byte or two and the process is complete.

Single Block Read Timing Diagram



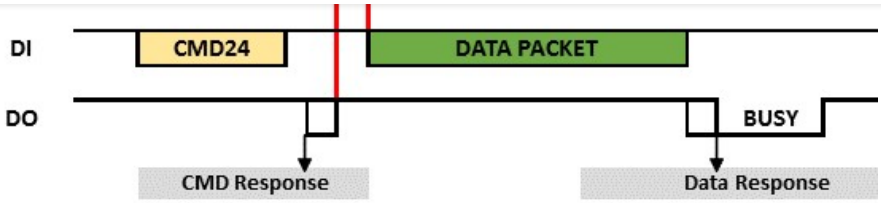
Multi-Block Read Timing Diagram

In the case of **CMD18** for a **multi-read block**, follow the procedure up until the 0xFE response. After this, the card will be transmitting blocks of data till it receives a **CMD12** command, which is the indication from the microcontroller that the block read is over.

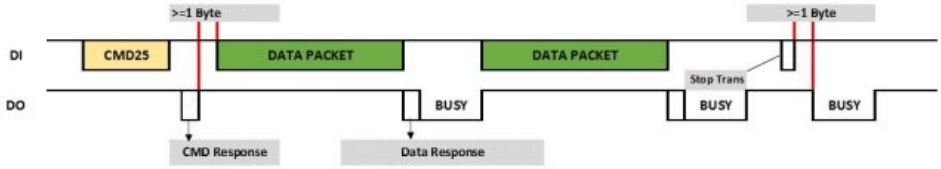


In case of performing a **single block write** or a **multi-block write**, send **CMD24** or **CMD25** respectively and wait for the response 0x00. Every block write requires a start token 0xFE in case of a single block write and 0xFC in case of a multi-block write to enable the card to start writing a single block of data onto the specified sector. Ensure 0xFC is sent for every block-write during a multi-block write. After every block writes, a stop token 0xFD must be issued (not required in the case of single block write). After the completion of all the data block write, a **CMD13 command is mandatory**.
Do not forget to send dummy bytes between responses as well as prior to sending start tokens.

Single Block Write Timing Diagram

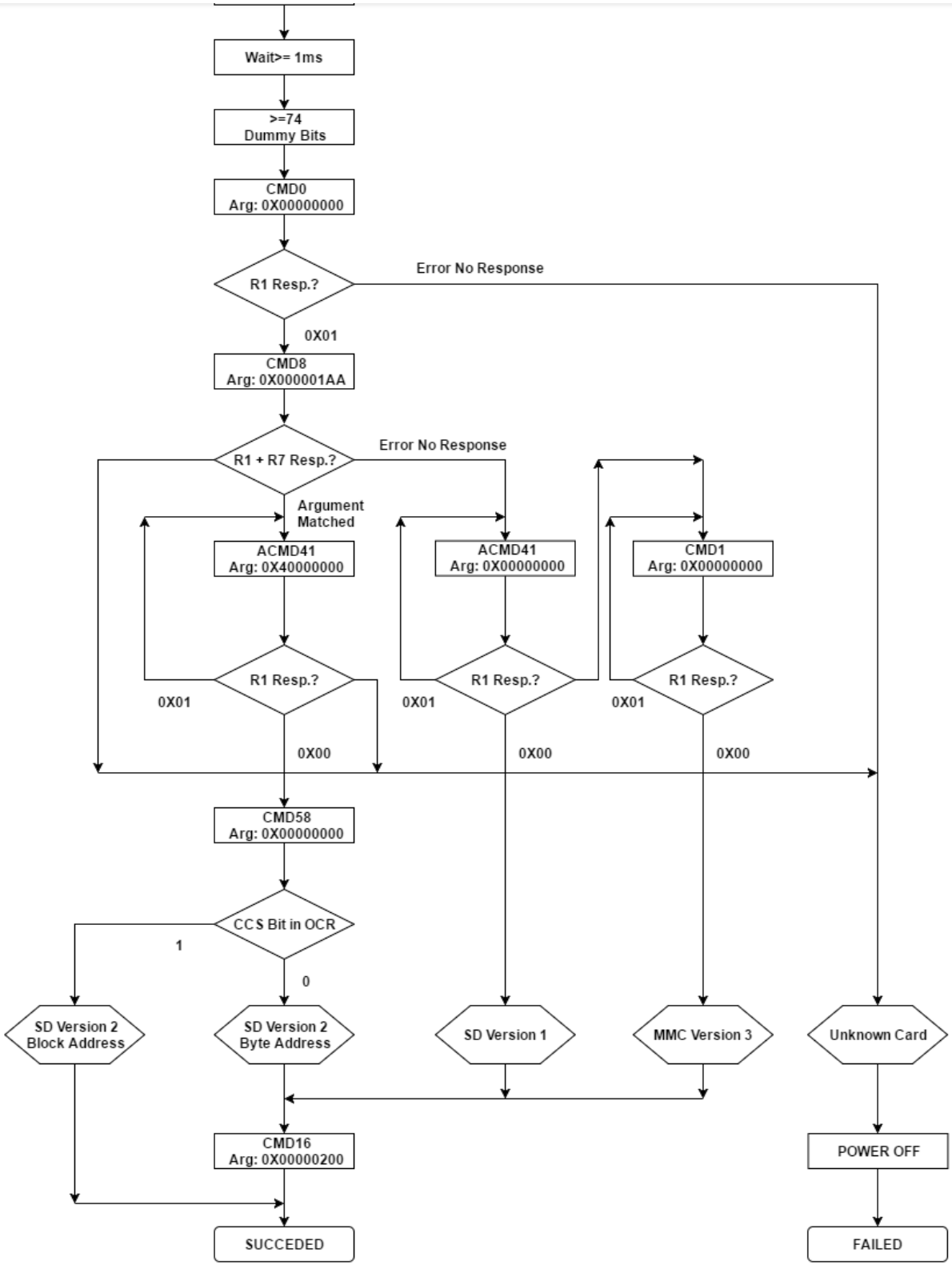


Multi-Block Write Timing Diagram



Interfacing Microcontrollers with SD Card – Flow diagram

Creating a function for each of the read/write sections is the ideal way to code the software. This is the basic structure to be followed during read/write operations performed on raw data on an SD card.



Further reading

- Raw SD Read/Write Using PIC18F4550

Spread the love, share this

Sensor
Boards