UNIVERSITEIT
iYUNIVESITHI
STELLENBOSCH
UNIVERSITY

100
1918 · 2018

forward together · saam vorentoe · masiye phambili

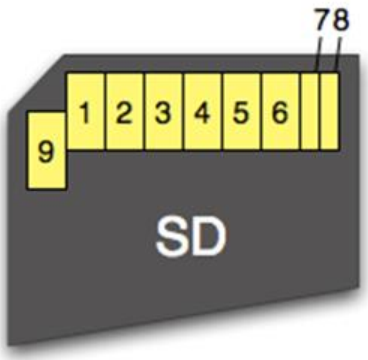
Computer Systems / Rekenaarstelsels 245 - 2020

Lecture 21

Serial Communication – SPI Example/ Seriële Kommunikasie – SPI Voorbeeld

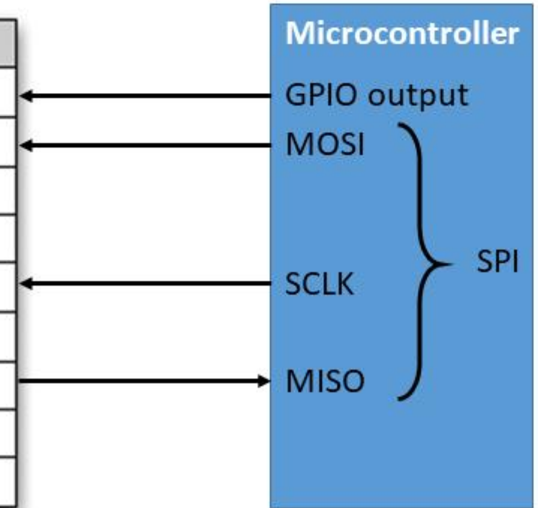
Dr Rensu Theart & Dr Lourens Visagie

SPI Interfacing examples
SPI Koppelvlak voorbeelde

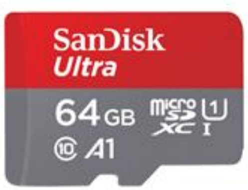


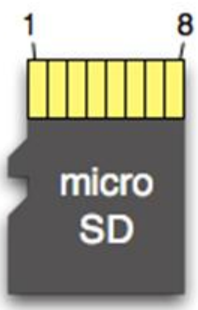
Pin	SD	SPI
1	CD/DAT3	CS
2	CMD	DI
3	VSS1	VSS1
4	VDD	VDD
5	CLK	SCLK
6	VSS2	VSS2
7	DAT0	DO
8	DAT1	X
9	DAT2	X

Slave device

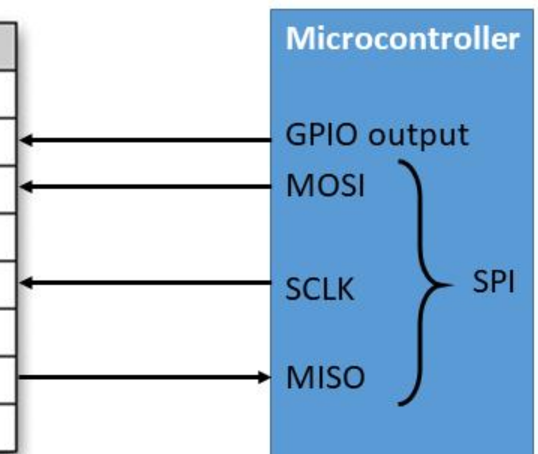


Master device



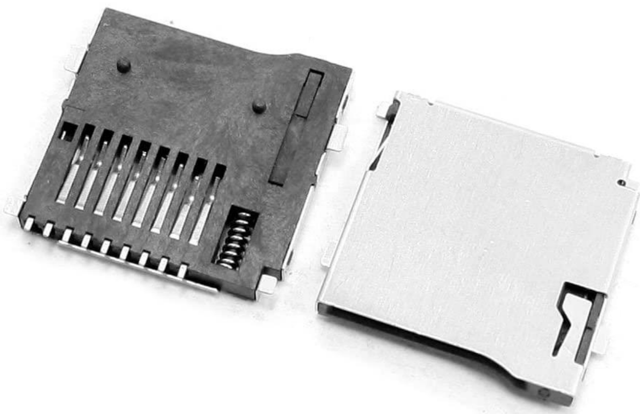


Pin	SD	SPI
1	DAT2	X
2	CD/DAT3	CS
3	CMD	DI
4	VDD	VDD
5	CLK	SCLK
6	VSS	VSS
7	DAT0	DO
8	DAT1	X



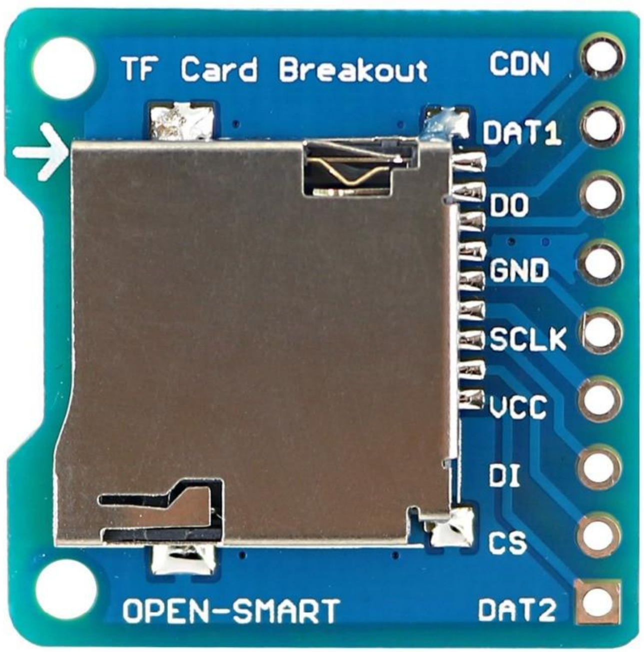
SD Card holders and modules
SD Kaart houers en modules

- SD card holders are a convenient way to insert or remove SD cards



SD Card holders and modules
SD Kaart houers en modules

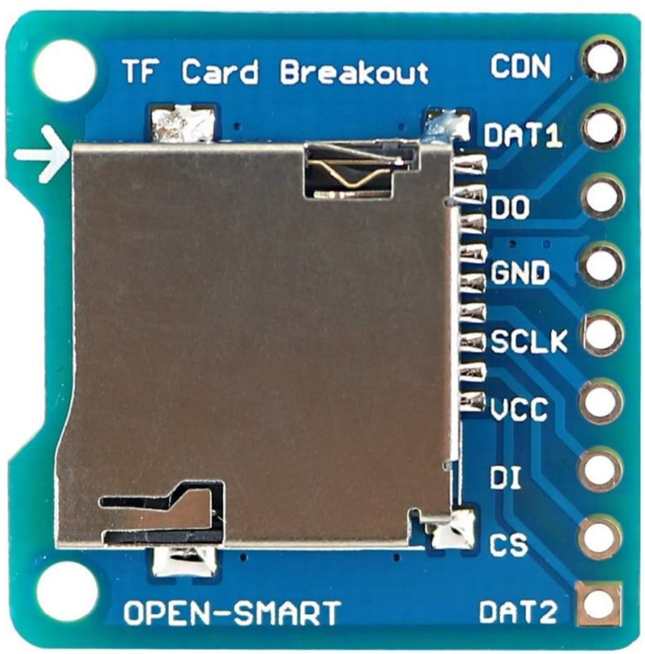
- SD card modules sometimes include additional circuitry:
- 3.3V regulator
- Buffer/level shifting ICs for clock and data



SD Card holders and modules

SD Kaart houers en modules

- SD card modules sometimes include additional circuitry:
- 3.3V regulator
- Buffer/level shifting ICs for clock and data



SD Card supply voltage
SD Kaart spanningstoevoer

Physical Layer Simplified Specification Version 7.10

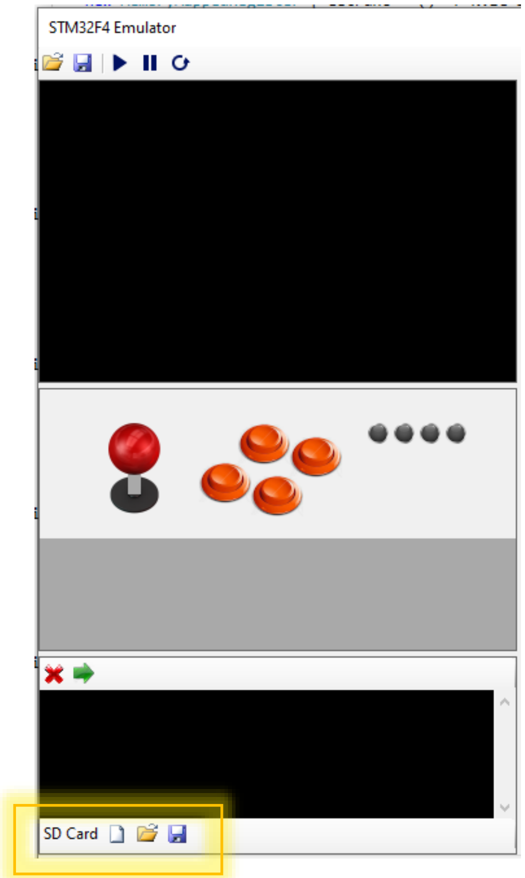
2. System Features

- Targeted for portable and stationary applications
- Capacity of Memory
  (1) Standard Capacity SD Memory Card (SDSC): Up to and including 2 GB
  (2) High Capacity SD Memory Card (SDHC): More than 2GB and up to and including 32GB
  (3) Extended Capacity SD Memory Card (SDXC): More than 32GB and up to and including 2TB
  (4) Ultra Capacity SD Memory Card (SDUC): More than 2TB and up to and including 128TB
- Voltage range:
  High Voltage SD Memory Card – Operating voltage range: 2.7-3.6 V
  UHS-II SD Memory Card – Operating voltage range VDD1: 2.7-3.6 V, VDD2: 1.70-1.95V
  SD Express Memory Card - Operating voltage range VDD1: 2.7-3.6 V, VDD2: 1.70-1.95V and optional VDD3: 1.14-1.30V (operated instead of VDD2 if supported).

Emulator SD card
Emuleerder SD kaart

- 16 MB capacity
- SD card memory is stored in RAM on your PC – it is not automatically persisted to hard disk
- You can manually store and load the SD card state (it will save to a .ISO file)
- Memory card state is preserved when you reset the emulator
- Use the new/format button if you want to clear the card contents

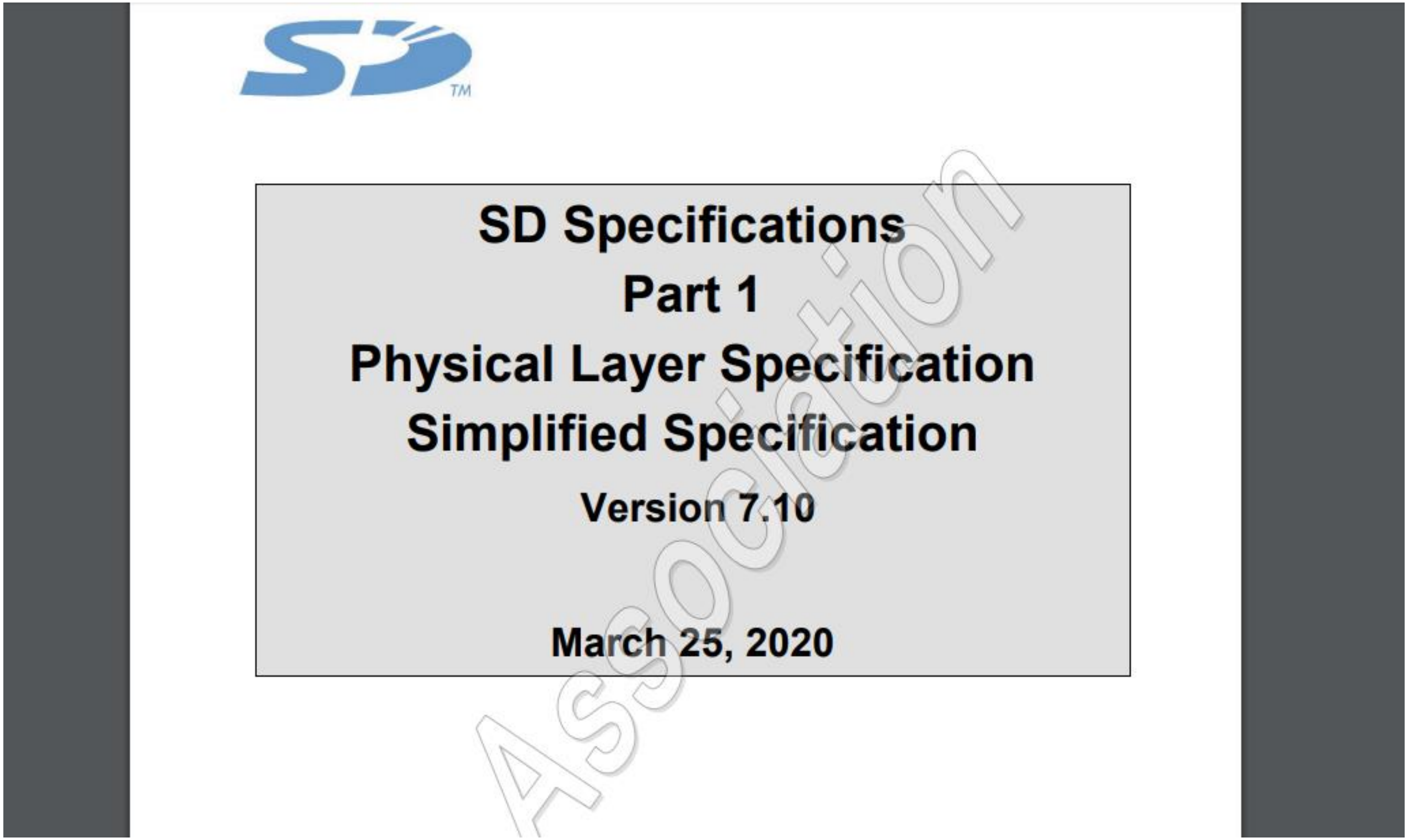
MCU pin	Signal
PA5	SCK
PA6	MISO
PA7	MOSI
PA15	CS



Low-level SD card interfacing – initialise
Lae-vlak SD kaart koppelvlak - inisialisering

- Set Chip-Select (CS) line low to enable the SD card
- Send command 0 (go to idle state)
- Request 1-byte response – SD card should return 0x01: idle bit set, and no errors

SD Card Specification
SD Kaart Spesifikasie



SD Specifications
Part 1
Physical Layer Specification
Simplified Specification

Version 7.10

March 25, 2020

https://www.sdcard.org/downloads/pls/index.html

SD Card Specification
SD Kaart Spesifikasie

(*1) Note: Card returns busy when
- Card executes internal initialization process.
- If the card is High capacity SD Memory Card, there are 2 cases
  1. CMD8 was not issued before ACMD41
  2. ACMD41 is issued with HCS=0

(*2) Note: 2.1mm SD Memory Card can be initialized using CMD1 and Thin (1.4mm) SD Memory Card can be initialized using CMD1 only after firstly initialized by using CMD0 and ACMD41. In any of the cases CMD1 is not recommended because it may be difficult for the host to distinguish between MultiMediaCard and SD Memory Card.
If the SD card is initialized by CMD1 and the host treat it as MMC card, not SD card, the Data of the card may be damaged because of wrong interpretation of CSD and CID registers

(*3) Note: If SDUC card, card can stay at busy status and does not reply ready to host during ACMD41 to let host know SDUC card cannot use SPI mode.

Figure 7-1 : SD Memory Card State Diagram (SPI mode)

7.2.1 Mode Selection and Initialization

The SD Card is powered up in the SD mode. It will enter SPI mode if the CS signal is asserted (negative) during the reception of the reset command (CMD0). If the card recognizes that the SD mode is required it will not respond to the command and remain in the SD mode. If SPI mode is required, the card will switch to SPI and respond with the SPI mode R1 response.

The only way to return to the SD mode is by entering the power cycle. In SPI mode, the SD Card protocol state machine in SD mode is not observed. All the SD Card commands supported in SPI mode are always available.

7.3.1.1 Command Format

All the SD Memory Card commands are 6 bytes long. The command transmission always starts with the left most bit of the bit string corresponding to the command codeword. All commands are protected by a CRC (see Section 4.5). The commands and arguments are listed in Table 7-3.

Bit position	47	46	[45:40]	[39:8]	[7:1]	0
Width (bits)	1	1	6	32	7	1
Value	'0'	'1'	x	x	x	'1'
Description	start bit	transmission bit	command index	argument	CRC7	end bit

Table 7-1 : Command Format

SD Card Specification
SD Kaart Spesifikasie

7.3.1.1 Command Format

All the SD Memory Card commands are 6 bytes long. The command transmission always starts with the left most bit of the bit string corresponding to the command codeword. All commands are protected by a CRC (see Section 4.5). The commands and arguments are listed in Table 7-3.

Bit position	47	46	[45:40]	[39:8]	[7:1]	0
Width (bits)	1	1	6	32	7	1
Value	'0'	'1'	x	x	x	'1'
Description	start bit	transmission bit	command index	argument	CRC7	end bit

Table 7-1 : Command Format

CMD0 bits	0	1	0 0 0 0 0 0	0000000…..0	1001010	1

CMD0 bytes	0x40		0x00,0x00,0x00,0x00	0x95

SD Card Specification
SD Kaart Spesifikasie

7.3.1.1 Command Format

All the SD Memory Card commands are 6 bytes long. The command transmission always starts with the left most bit of the bit string corresponding to the command codeword. All commands are protected by a CRC (see Section 4.5). The commands and arguments are listed in Table 7-3.

Bit position	47	46	[45:40]	[39:8]	[7:1]	0
Width (bits)	1	1	6	32	7	1
Value	'0'	'1'	x	x	x	'1'
Description	start bit	transmission bit	command index	argument	CRC7	end bit

Table 7-1 : Command Format

Algorithm for this in the spec document

CMD0 bits	0	1	0 0 0 0 0 0	0000000.....0	1001010	1

CMD0 bytes	0x40	0x00,0x00,0x00,0x00	0x95

SD Card Specification
SD Kaart Spesifikasie

In idle state: The card is in idle state and running the initializing process.
Erase reset: An erase sequence was cleared before executing because an out of erase sequence command was received.
Illegal command: An illegal command code was detected.
Communication CRC error: The CRC check of the last command failed.
Erase sequence error: An error in the sequence of erase commands occurred.
Address error: A misaligned address that did not match the block length was used in the command.
Parameter error: The command's argument (e.g. address, block length) was outside the allowed range for this card.

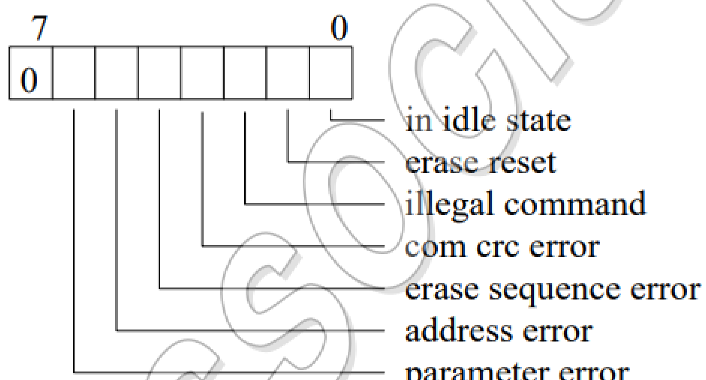


Figure 7-9 : R1 Response Format

Low-level SD card interfacing – request CSD
Lae-vlak SD kaart koppelvlak – vra CSD aan

- CSD = Card Specific Data register
- Not part of SD card memory, but a "register" that can be queried

- CID = Card identification Data: Another register that can be read.
- Contains manufacturer information

- There are other registers as well, but it's possible to use the card without knowing their contents :)

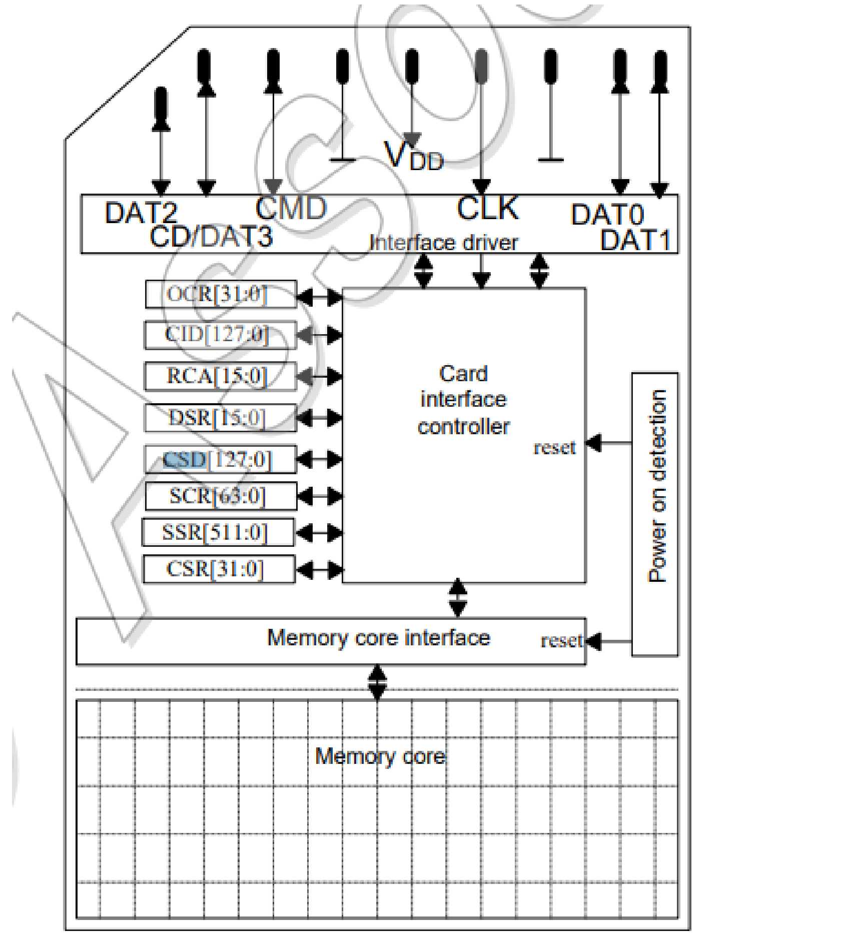


Figure 3-12 : SD Memory Card Architecture

Low-level SD card interfacing – request CSD
Lae-vlak SD kaart koppelvlak – vra CSD aan

- CSD = Card Specific Data register
- Contains information regarding
  - Card capacity
  - Read/write block sizes
  - Access times
  - And other things…
- 16 bytes total
- Different version SD card (Standard capacity, high/extended capacity, ultra capacity) have different ways of arranging data in the CSD

5.3.2 CSD Register (CSD Version 1.0)

Name	Field	Width	Value	Cell Type	CSD-slice
CSD structure	CSD_STRUCTURE	2	00b	R	[127:126]
reserved	-	6	00 0000b	R	[125:120]
data read access-time-1	TAAC	8	xxh	R	[119:112]
data read access-time-2 in CLK cycles (NSAC*100)	NSAC	8	xxh	R	[111:104]
max. data transfer rate	TRAN_SPEED	8	32h or 5Ah	R	[103:96]
card command classes	CCC	12	01x110110101b	R	[95:84]
max. read data block length	READ_BL_LEN	4	xh	R	[83:80]
partial blocks for read allowed	READ_BL_PARTIAL	1	1b	R	[79:79]
write block misalignment	WRITE_BLK_MISALIGN	1	xb	R	[78:78]
read block misalignment	READ_BLK_MISALIGN	1	xb	R	[77:77]
DSR implemented	DSR_IMP	1	xb	R	[76:76]
reserved	-	2	00b	R	[75:74]
device size	C_SIZE	12	xxxh	R	[73:62]
max. read current @VDD min	VDD_R_CURR_MIN	3	xxxb	R	[61:59]
max. read current @VDD max	VDD_R_CURR_MAX	3	xxxb	R	[58:56]
max. write current @VDD min	VDD_W_CURR_MIN	3	xxxb	R	[55:53]
max. write current @VDD max	VDD_W_CURR_MAX	3	xxxb	R	[52:50]
device size multiplier	C_SIZE_MULT	3	xxxb	R	[49:47]
erase single block enable	ERASE_BLK_EN	1	xb	R	[46:46]
erase sector size	SECTOR_SIZE	7	xxxxxxxb	R	[45:39]
write protect group size	WP_GRP_SIZE	7	xxxxxxxb	R	[38:32]
write protect group enable	WP_GRP_ENABLE	1	xb	R	[31:31]
reserved (Do not use)		2	00b	R	[30:29]
write speed factor	R2W_FACTOR	3	xxxb	R	[28:26]
max. write data block length	WRITE_BL_LEN	4	xxxxb	R	[25:22]
partial blocks for write allowed	WRITE_BL_PARTIAL	1	xb	R	[21:21]
reserved	-	5	00000b	R	[20:16]
File format group	FILE_FORMAT_GRP	1	xb	R/W(1)	[15:15]
copy flag	COPY	1	xb	R/W(1)	[14:14]
permanent write protection	PERM_WRITE_PROTECT	1	xb	R/W(1)	[13:13]
temporary write protection	TMP_WRITE_PROTECT	1	xb	R/W	[12:12]
File format	FILE_FORMAT	2	xxb	R/W(1)	[11:10]
reserved		2	00b	R/W	[9:8]
CRC	CRC	7	xxxxxxxb	R/W	[7:1]
not used, always'1'	-	1	1b	-	[0:0]

Table 5-4 : The CSD Register Fields (CSD Version 1.0)

Low-level SD card interfacing – request CSD
Lae-vlak SD kaart koppelvlak – vra CSD aan

- Calculate card memory capacity from parameters in the CSD
- For a Standard Capacity SD card (which is what we have in the emulator):

$$\text{memory capacity} = \text{BLOCKNR} * \text{BLOCK\_LEN}$$

Where

$$\text{BLOCKNR} = (\text{C\_SIZE}+1) * \text{MULT}$$

$$\text{MULT} = 2^{\text{C\_SIZE\_MULT}+2} \qquad (\text{C\_SIZE\_MULT} < 8)$$

$$\text{BLOCK\_LEN} = 2^{\text{READ\_BL\_LEN}}, \qquad (\text{READ\_BL\_LEN} < 12)$$

Low-level SD card interfacing – request CSD
Lae-vlak SD kaart koppelvlak – vra CSD aan

- CSD is obtained using same sequence as normal data block read
- Send command 9
- Read R1 response
- Read block data (16 bytes)
    - Start token (1x byte)
    - 16 data bytes
    - 2 bytes checksum
    - Stop token (1x byte)

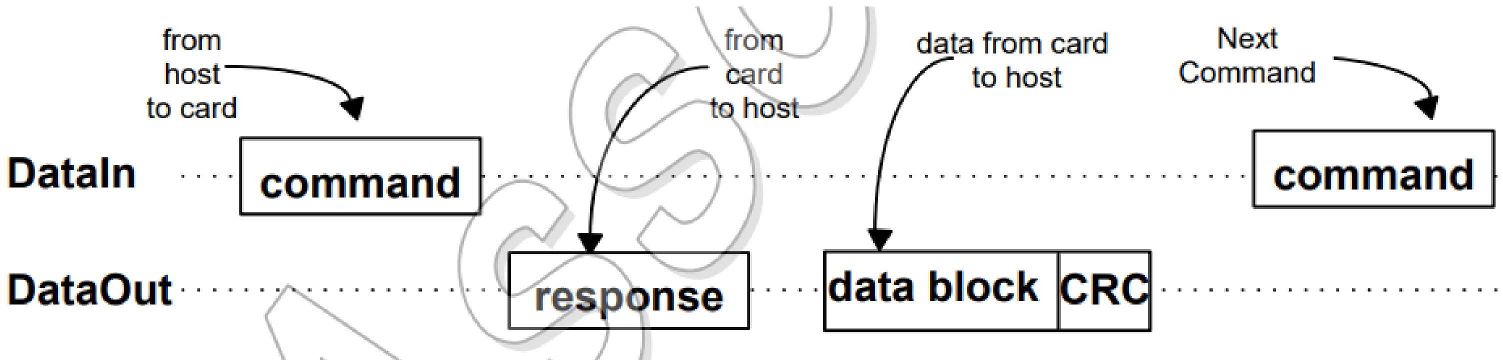


Figure 7-3 : Single Block Read Operation

Low-level SD card interfacing – request CSD
Lae-vlak SD kaart koppelvlak – vra CSD aan

- Use supplied library functions (sd.c and sd.h)
- SD_SendCommand(9, 0)
- SD_RxDataBlock(buffer, 16)

Low-level SD card interfacing – request CSD
Lae-vlak SD kaart koppelvlak – vra CSD aan

Hex	Binary
00	0000 0000
26	0010 0110
00	0000 0000
32	0011 0010
5F	0101 1111
59	0101 1001
01	0000 0001
F4	1111 0100
3E	0011 1110
F9	1111 1001
4F	0100 1111
FF	1111 1111
92	1001 0010
40	0100 0000
50	0101 0000
01	0000 0001

5.3.2 CSD Register (CSD Version 1.0)

Name	Field	Width	Value	Cell Type	CSD-slice
CSD structure	CSD_STRUCTURE	2	00b	R	[127:126]
reserved	-	6	00 0000b	R	[125:120]
data read access-time-1	TAAC	8	xxh	R	[119:112]
data read access-time-2 in CLK cycles (NSAC*100)	NSAC	8	xxh	R	[111:104]
max. data transfer rate	TRAN_SPEED	8	32h or 5Ah	R	[103:96]
card command classes	CCC	12	01x110110101b	R	[95:84]
max. read data block length	READ_BL_LEN	4	xh	R	[83:80]
partial blocks for read allowed	READ_BL_PARTIAL	1	1b	R	[79:79]
write block misalignment	WRITE_BLK_MISALIGN	1	xb	R	[78:78]
read block misalignment	READ_BLK_MISALIGN	1	xb	R	[77:77]
DSR implemented	DSR_IMP	1	xb	R	[76:76]
reserved	-	2	00b	R	[75:74]
device size	C_SIZE	12	xxxh	R	[73:62]
max. read current @VDD min	VDD_R_CURR_MIN	3	xxxb	R	[61:59]
max. read current @VDD max	VDD_R_CURR_MAX	3	xxxb	R	[58:56]
max. write current @VDD min	VDD_W_CURR_MIN	3	xxxb	R	[55:53]
max. write current @VDD max	VDD_W_CURR_MAX	3	xxxb	R	[52:50]
device size multiplier	C_SIZE_MULT	3	xxxb	R	[49:47]
erase single block enable	ERASE_BLK_EN	1	xb	R	[46:46]
erase sector size	SECTOR_SIZE	7	xxxxxxxb	R	[45:39]
write protect group size	WP_GRP_SIZE	7	xxxxxxxb	R	[38:32]
write protect group enable	WP_GRP_ENABLE	1	xb	R	[31:31]
reserved (Do not use)		2	00b	R	[30:29]
write speed factor	R2W_FACTOR	3	xxxb	R	[28:26]
max. write data block length	WRITE_BL_LEN	4	xxxxb	R	[25:22]
partial blocks for write allowed	WRITE_BL_PARTIAL	1	xb	R	[21:21]
reserved	-	5	00000b	R	[20:16]
File format group	FILE_FORMAT_GRP	1	xb	R/W(1)	[15:15]
copy flag	COPY	1	xb	R/W(1)	[14:14]
permanent write protection	PERM_WRITE_PROTECT	1	xb	R/W(1)	[13:13]
temporary write protection	TMP_WRITE_PROTECT	1	xb	R/W	[12:12]
File format	FILE_FORMAT	2	xxb	R/W(1)	[11:10]
reserved		2	00b	R/W	[9:8]
CRC	CRC	7	xxxxxxxb	R/W	[7:1]
not used, always'1'	-	1	1b	-	[0:0]

Table 5-4 : The CSD Register Fields (CSD Version 1.0)

Low-level SD card interfacing – request CSD
Lae-vlak SD kaart koppelvlak – vra CSD aan

Hex	Binary	
00	0000 0000	
26	0010 0110	
00	0000 0000	
32	0011 0010	
5F	0101 1111	
59	0101 1001	READ_BL_LEN
01	0000 0001	
F4	1111 0100	C_SIZE
3E	0011 1110	
F9	1111 1001	C_SIZE_MULT
4F	0100 1111	
FF	1111 1111	
92	1001 0010	
40	0100 0000	
50	0101 0000	
01	0000 0001	

5.3.2 CSD Register (CSD Version 1.0)

Name	Field	Width	Value	Cell Type	CSD-slice
CSD structure	CSD_STRUCTURE	2	00b	R	[127:126]
reserved	-	6	00 0000b	R	[125:120]
data read access-time-1	TAAC	8	xxh	R	[119:112]
data read access-time-2 in CLK cycles (NSAC*100)	NSAC	8	xxh	R	[111:104]
max. data transfer rate	TRAN_SPEED	8	32h or 5Ah	R	[103:96]
card command classes	CCC	12	01x110110101b	R	[95:84]
max. read data block length	READ_BL_LEN	4	xh	R	[83:80]
partial blocks for read allowed	READ_BL_PARTIAL	1	1b	R	[79:79]
write block misalignment	WRITE_BLK_MISALIGN	1	xb	R	[78:78]
read block misalignment	READ_BLK_MISALIGN	1	xb	R	[77:77]
DSR implemented	DSR_IMP	1	xb	R	[76:76]
reserved	-	2	00b	R	[75:74]
device size	C_SIZE	12	xxxh	R	[73:62]
max. read current @VDD min	VDD_R_CURR_MIN	3	xxxb	R	[61:59]
max. read current @VDD max	VDD_R_CURR_MAX	3	xxxb	R	[58:56]
max. write current @VDD min	VDD_W_CURR_MIN	3	xxxb	R	[55:53]
max. write current @VDD max	VDD_W_CURR_MAX	3	xxxb	R	[52:50]
device size multiplier	C_SIZE_MULT	3	xxxb	R	[49:47]
erase single block enable	ERASE_BLK_EN	1	xb	R	[46:46]
erase sector size	SECTOR_SIZE	7	xxxxxxxb	R	[45:39]
write protect group size	WP_GRP_SIZE	7	xxxxxxxb	R	[38:32]
write protect group enable	WP_GRP_ENABLE	1	xb	R	[31:31]
reserved (Do not use)		2	00b	R	[30:29]
write speed factor	R2W_FACTOR	3	xxxb	R	[28:26]
max. write data block length	WRITE_BL_LEN	4	xxxxb	R	[25:22]
partial blocks for write allowed	WRITE_BL_PARTIAL	1	xb	R	[21:21]
reserved	-	5	00000b	R	[20:16]
File format group	FILE_FORMAT_GRP	1	xb	R/W(1)	[15:15]
copy flag	COPY	1	xb	R/W(1)	[14:14]
permanent write protection	PERM_WRITE_PROTECT	1	xb	R/W(1)	[13:13]
temporary write protection	TMP_WRITE_PROTECT	1	xb	R/W	[12:12]
File format	FILE_FORMAT	2	xxb	R/W(1)	[11:10]
reserved		2	00b	R/W	[9:8]
CRC	CRC	7	xxxxxxxb	R/W	[7:1]
not used, always'1'	-	1	1b	-	[0:0]

Table 5-4 : The CSD Register Fields (CSD Version 1.0)

Low-level SD card interfacing – request CSD
Lae-vlak SD kaart koppelvlak – vra CSD aan

memory capacity = BLOCKNR * BLOCK_LEN

Where

BLOCKNR = (C_SIZE+1) * MULT

MULT = $2^{C\_SIZE\_MULT+2}$ (C_SIZE_MULT < 8)

BLOCK_LEN = $2^{READ\_BL\_LEN}$, (READ_BL_LEN < 12)

$C\_SIZE = 011111010000_2 = 2000$

$C\_SIZE\_MULT = 010_2 = 2$

$READ\_BL\_LEN = 1001_2 = 9$

$MULT = 2^{(2+2)} = 16$

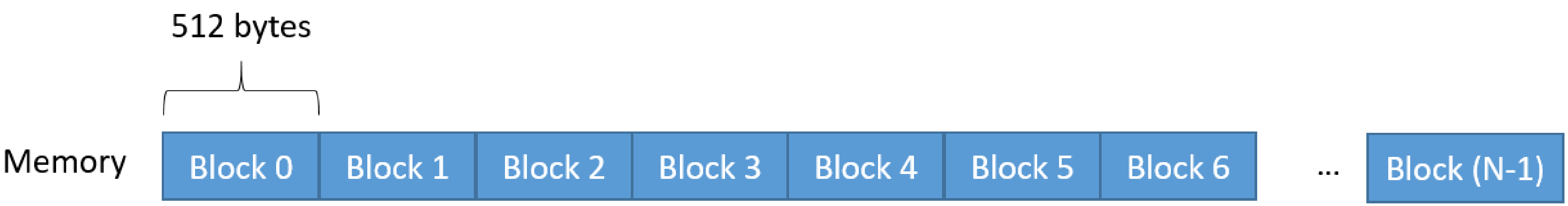
BLOCKNR = (2000+1)*MULT = 2001*16 = 32016

BLOCK_LEN = 2^9 = 512

Memory capacity = 32016 * 512 = 16392192 bytes

SD card file system
SD kaart leêrstelsel

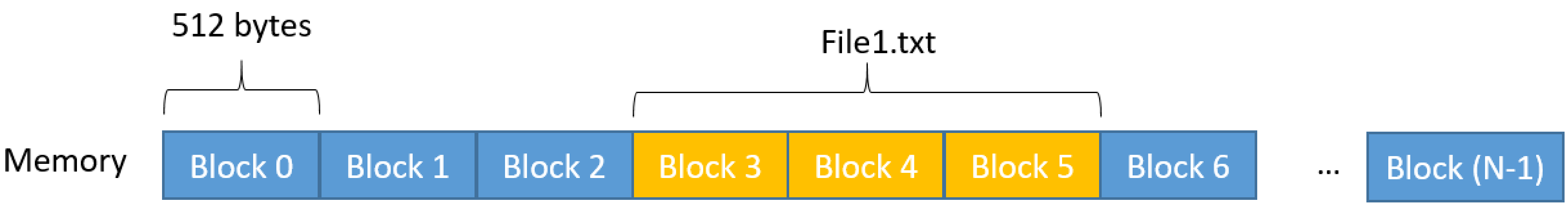
- SD card memory is arranged in blocks of 512 bytes



- Low-level data read and write exposes SD card memory as a flat structure. I.e: Read 512 bytes starting from address 1024. Write 1024 bytes starting at address 2048
- The existence of files and folders on the SD card is something that is implemented on top of this. This is called a file-system

SD card file system
SD kaart leêrstelsel

- Example:



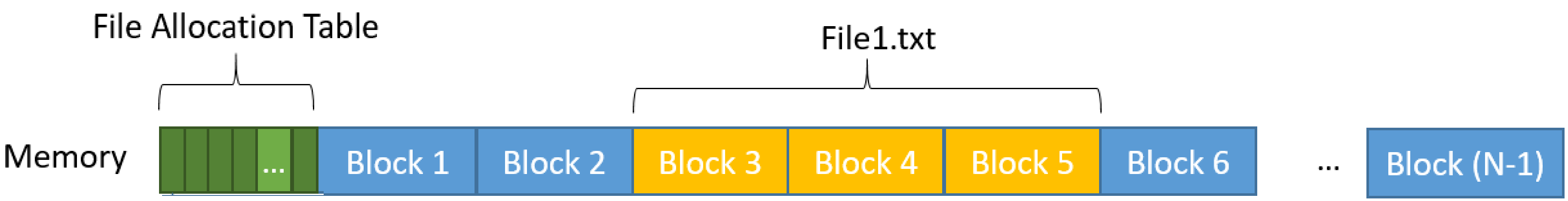
File information (meta-data)

- uint32_t Date_created // 4 bytes
- char Filename[12] // 12 bytes
- uint32_t Start_address // 4 bytes
- uint32_t Size // 4 bytes

- A file is written to the SD card to span over a few 512 byte blocks
- To know where the file begins and ends, we need a structure with information to keep track of it
- (Also include other information in the structure, like filename and date)

SD card file system
SD kaart leêrstelsel

- Example:



File information (meta-data)

- uint32_t Date_created // 4 bytes
- char Filename[12] // 12 bytes
- uint32_t Start_address // 4 bytes
- uint32_t Size // 4 bytes

- Use a reserved space at the beginning of the SD card to save all the file information. This is called the File Allocation Table (FAT)
- There are different types of file systems – FAT16 and FAT32 are common for SD cards
- Use FatFs library provided by STM to manage all the complexity for you

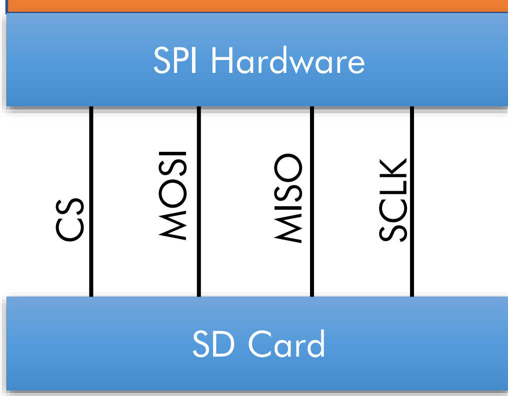
SD card file system
SD kaart leêrstelsel

Main program	`Main.c: int main()`

FatFs (FAT File System library)	`f_open(…)`
	`f_read(…)`
	`f_write(…)`

SD Card SPI library	`SD_Init(…)`
	`SD_Read(…)`
	`SD_Write(…)`

SPI interface (HAL)	`HAL_SPI_Transmit(…)`
	`HAL_SPI_Receive(…)`
	`HAL_SPI_TransmitReceive(…)`



SD card file system
SD kaart leêrstelsel

UM1721
User manual

Developing applications on STM32Cube™ with FatFs

Introduction

The STM32Cube™ is an STMicroelectronics original initiative to make developers' lives easier by reducing development effort, time and cost. STM32Cube™ covers the whole STM32 portfolio.

STM32Cube™ includes:

- STM32CubeMX, a graphical software configuration tool that allows the generation of C initialization code using graphical wizards.
- A comprehensive embedded software platform, delivered per Series (such as STM32CubeF4 for STM32F4 Series)
  - The STM32Cube™ HAL, STM32 abstraction layer embedded software ensuring maximized portability across the STM32 portfolio,

SD card file system
SD kaart leêrstelsel

4.5 FatFs APIs

The FatFs APIs layer implements file system APIs. It uses disk I/O interface to communicate with the appropriate physical drive. The set of APIs is divided into four groups:

- Group of APIs that operates with logical volume or partition.
- Group of APIs that operates with directory.
- Group of APIs that operates with file.
- Group of APIs that operates with both file and directory.

The following list describes what FatFs can do to access the FAT volumes:

- f_mount(): Register/Unregister a work area
- f_open(): Open/Create a file
- f_close(): Close a file
- f_read(): Read a file
- f_write(): Write a file
- f_lseek(): Move read/write pointer, Expand a file size
- f_truncate(): Truncate a 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(): Check if the object is exist and get status
- f_mkdir(): Create a directory

SD card file system
SD kaart leêrstelsel

4.6 FatFs low level APIs

Since the FatFs module is completely separate from the disk I/O and RTC module, it requires some low level functions to operate the physical drive: read/write and get the current time. Because the low level disk I/O functions and RTC module are not a part of the FatFs module, they must be provided by the user.

The FatFs Middleware solution provides low level disk I/O drivers for some supported disk drives (RAMDisk, microSD, USBDisk).

An additional interface layer diskio.c has been added to add/remove dynamically (link) physical media to the FatFs module, providing low level disk I/O functions as mentioned below:

- disk_initialize(): Initializes the physical disk drive
- disk_status(): Returns the selected physical drive status
- disk_read(): Reads sector(s) from the disk
- disk_write(): Writes sector(s) to the disk
- disk_ioctl(): Controls device-specified features
- get_fattime(): Returns the current time

Application program MUST NOT call these functions, they are only called by FatFs file system functions such as, f_mount(), f_read() or f_write().