Lab 7: SD Card Reader Circuit

National Chiao Tung University
Chun-Jen Tsai
11/13/2015

Lab 7: SD Card Reader Circuit

- In this lab, you will design a circuit to read a text file from an SD card, and print the file to a UART terminal
- You will demo the design to your TA during the lab hours on 11/23

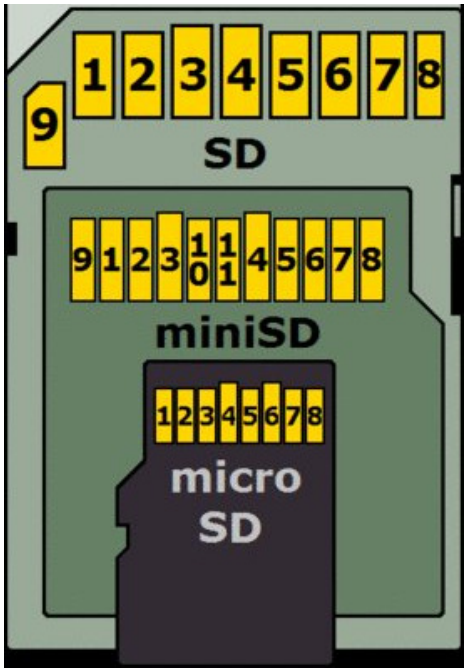
SD Card Specification

- The SD card we use follows the SDHC standard, formatted with the FAT32 file system
  - Note that we do not have to interpret the FAT32 file system to access a file stored in it!
- The physical structure is composed of 512-byte blocks, starting at block number 0, ends at block number 7,736,320 (we use a 4GB SD card)
- In this lab, we use the serial SPI interface to read SD card data

SD Card I/O Interface

- An SDHC card has different operation modes:
  - SPI mode
  - One-bit SD bus mode
  - Four-bit SD bus mode

SD Pin	Name	SPI Mode Function
1	nCS	SPI Card Select [CS] (Negative logic)
2	DI	SPI Serial Data In [MOSI]
3	VSS	Ground
4	VDD	Power
5	CLK	SPI Serial Clock [SCLK]
6	VSS	Ground
7	DO	SPI Serial Data Out [MISO]
8	NC	Unused
9	NC	Unused
10	NC	Reserved
11	NC	Reserved

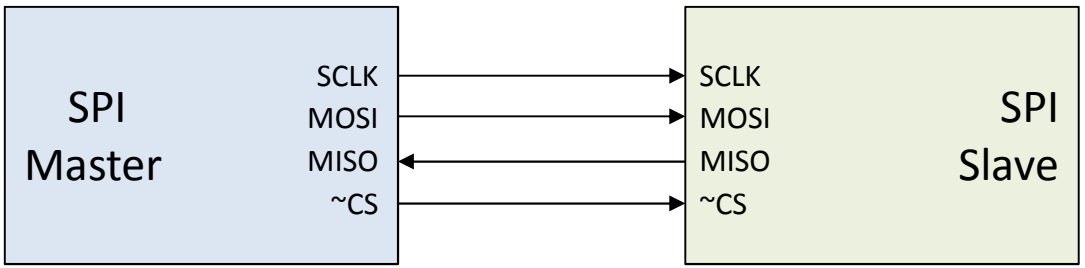


SD Card Initialization

- During the initialization phase, the SD card controller negotiates with the card to distinguish among different types of card: MMC, SD, SDHC, SDXH, etc.
  - The negotiation phase uses a slower clock (500kHZ)
- Once the card is initialized, the SD card controller can use a faster clock (e.g., the system clock) for read/write operations

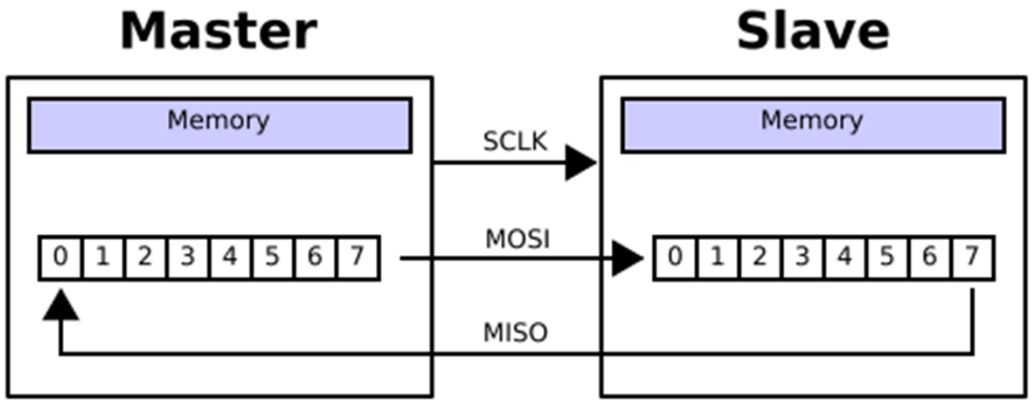
Serial Peripheral Interconnect (SPI)

- SPI is introduced by Motorola in 1980's for their MCU
  - Short-distance synchronous serial communications for SD cards, LCDs screens, audio codecs, boot flash, etc.
  - A four-wire, full-duplex, master-slave serial bus
  - One master, multiple slaves
  - Open-loop transmission, no slave acknowledgement protocol



SPI Data Communication

- The master select the target slave via CS first, then sends clock signal to the slave
- The master and slave exchange data one bit per clock cycle using shift registers
  - Data sizes can be of 8-, 12-, or 16-bit, depending on the device
  - Data sampling clock edge (rising or falling) also depending on the device → read data sheet!



Physical Structure and File Systems

- The physical structure of an SD card is simply composed of a series of 512-byte blocks
- To create directories to store files on the card, we need to format a logical file system on it
- There are two popular file systems for SD cards: FAT32 and exFAT
- In this lab, our SD card is formatted with the FAT32 file system, but you do not need to understand the file system to read a file from it (to be explained later)

FAT32 Structure

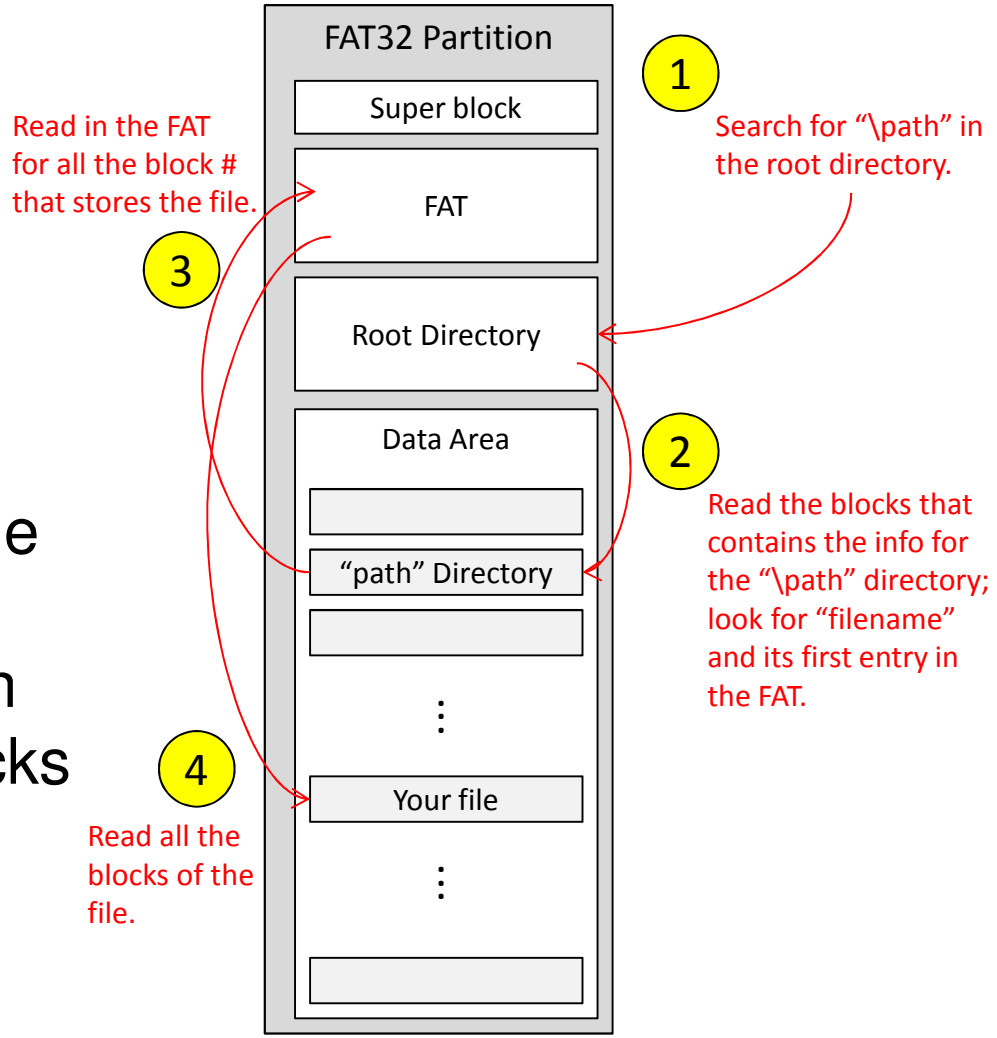
- The FAT32 file system is an simple standard defined by Microsoft, and popularly used for mass storage devices
- A FAT32 file system has the following components:
  - Boot sector: 512 bytes, block #0, also called the super block
    - The boot sector contains information such as the size of FAT, root directories, etc.
  - File allocation table (FAT): a table that shows which file is stored in which allocation units (each allocation unit is composed of several consecutive blocks)
  - Root directory: contains file names, file attributes, and the first allocation unit of all files in the root directory
  - Data area: the data blocks that actually store files

Disk Partitions

- A physical disk can have many disk partitions, each partition can have different file systems
- Although our SD card has only one FAT32 partition, the physical location of the super block on the disk is at block # 8192, not block #0.
  - Block #0 ~ #8191 have some system information, such as the partition table

File Structure of an FAT32 Partition

- The right way to read a file "\path\filename" in an FAT32 file system should follow the four steps shown in the figure
- However, if we know the content of the first few bytes of the file, we can simply scan all the blocks for the file location

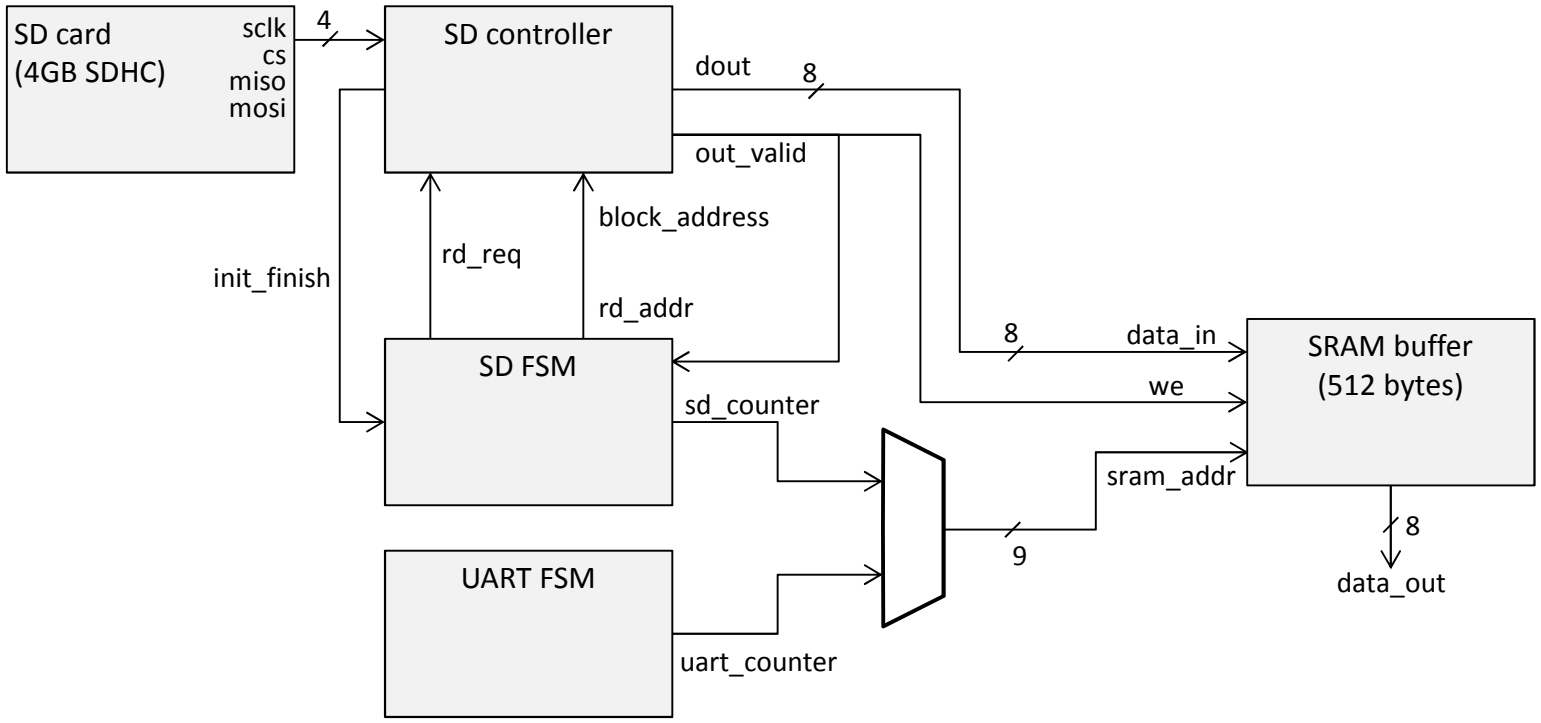


The Sample Code Package of Lab7

- For lab7, we will give you a sample code package that contains a top-level application, lab7.v, and an SD card controller, sd_card.v
  - There are other supporting files such as: debounce.v, uart.v, clk_divider.v, and lab7.ucf
- The circuit performs the following actions:
  - When the board is powered up, the SD card controller will initialize the card, and then lights up all the LEDs
  - The user then hit the reset button to reset the SD controller
  - After that, when the west button is pressed, the circuit will read the super block (#8192) from the SD card, and display the first byte of the block using the LED (should be 0xeb)

System Diagram of the Sample Code

- The block diagram of the Lab7 sample code:

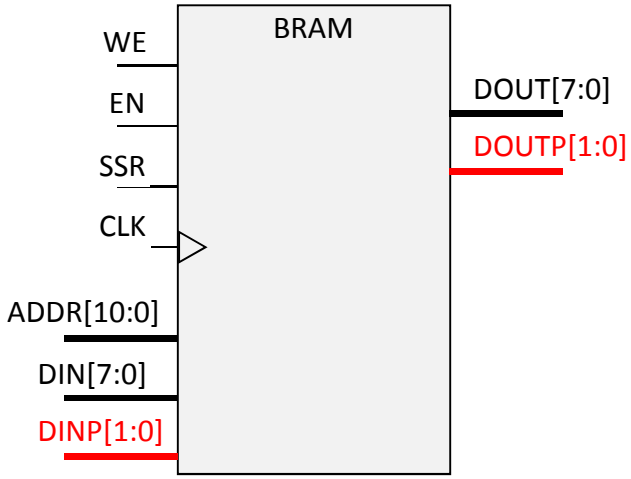


SD Controller Signals

- rd_req: triggers the reading of a block
- block_address: the block # of the SD card to read
- init_finish: SD card initialization is finished?
- dout: output one byte of data in the block.
- out_valid: the output byte in "dout" is ready

Block RAM on FPGA

- A block ram (BRAM) is a high speed small SRAM block
- On Spartan3e, each SRAM is of 2KB plus, where each data byte has an extra parity bit
- The physical I/O ports of a BRAM when configured as a single-port 8-bit data memory:



BRAM Signals (1/2)

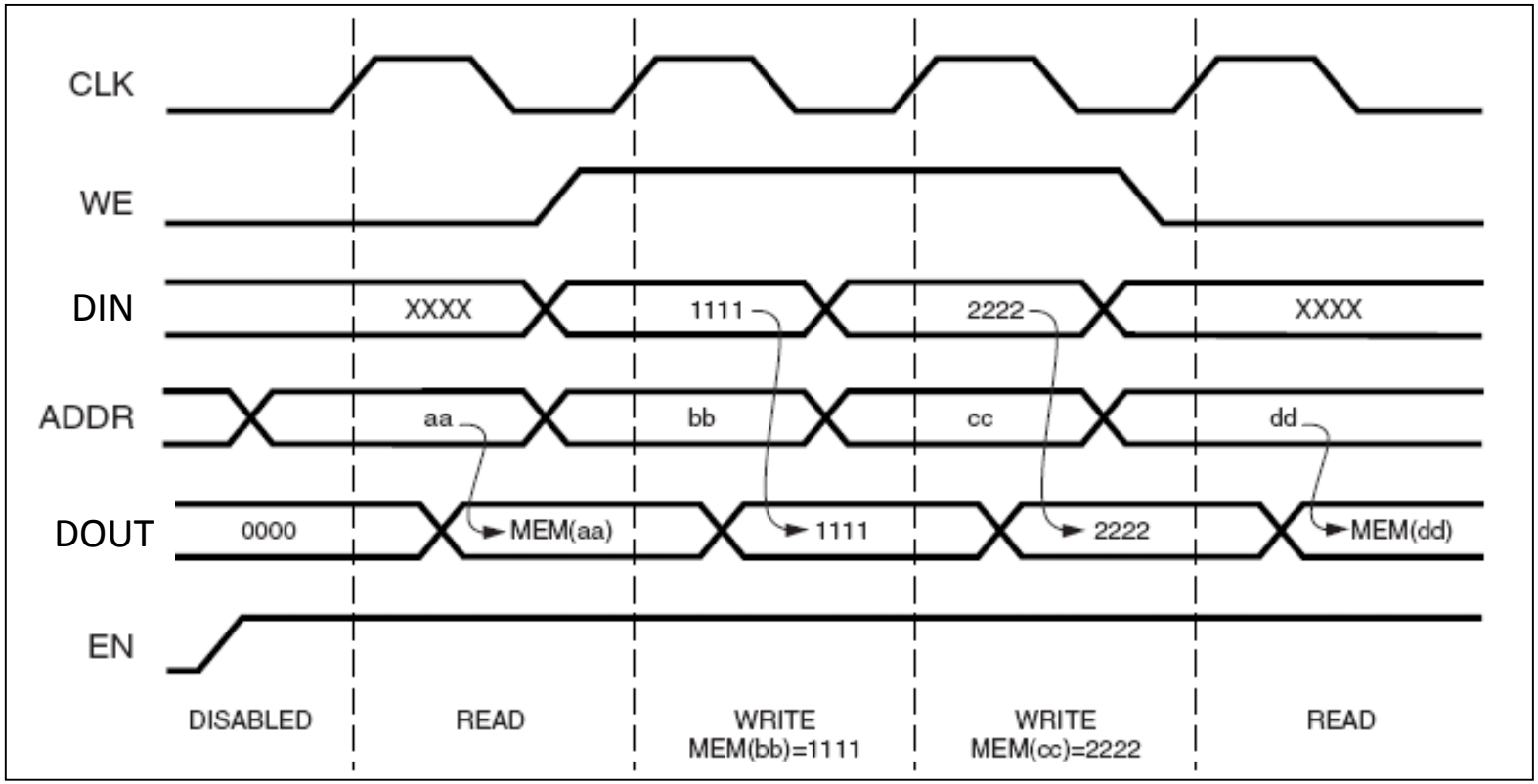
- **Clock — CLK**
  - Independent clock pins for synchronous operations
- **Enable — EN**
  - The read, write and reset functionality of the port is only active when this signal is enabled
- **WriteEnable — WE**
  - When active, the contents of the data input bus are written to the RAM, and the new data also reflects on the data out bus
  - When inactive, a read operation occurs and the contents of the memory cells reflect on the data out bus
- **Address Bus—ADDR**
  - The address bus selects the memory cells for read or write

BRAM Signals (2/2)

- **SET/RESET — SSR**
  - Reset operation does not affect the memory cells; the data output bus set to the default value SRVAL
- **Data In Bus — DIN**
  - The DI port provides the new data to be written into the RAM
- **Data Out Bus — DOUT**
  - The data out bus reflects the contents of the memory cells referenced by the address bus at the last active clock edge
  - During a write operation, the data out bus reflects the data in bus
- **Parity Buses – DINP, DOUTP**
  - Parity bits input/output bus

Timing Diagram

- For single port block RAM:



SRAM Inference

- The following Verilog code infers an SRAM:

```
reg  [7:0] sram[511:0];
wire       we, en;
reg  [7:0] data_out;
wire [7:0] data_in;
wire [8:0] sram_addr;

always @(posedge clk) begin // Write data into the SRAM block
  if (en && we) begin
    sram[sram_addr] <= data_in;
  end
end

always @(posedge clk) begin // Read data from the SRAM block
  if (en && we)             // If data is being written into SRAM,
    data_out <= data_in;    // forward the data to the read port
  else
    data_out <= sram[sram_addr]; // Send data to the read port
end
```

What You Need to Do for Lab7

- Modify the circuit such that:
  - When the west button is pressed, it triggers the SD card controller to read data blocks (starting at block #0) of the SD card and search for a data block that begins with the string “DLAB_TAG”
  - Once it find a data block with that tag, the whole data block will be treated as a NULL-terminated string and printed to the UART terminal
- Note that the text file will be less than 512 bytes so it can always be stored in the 512-byte SRAM buffer