

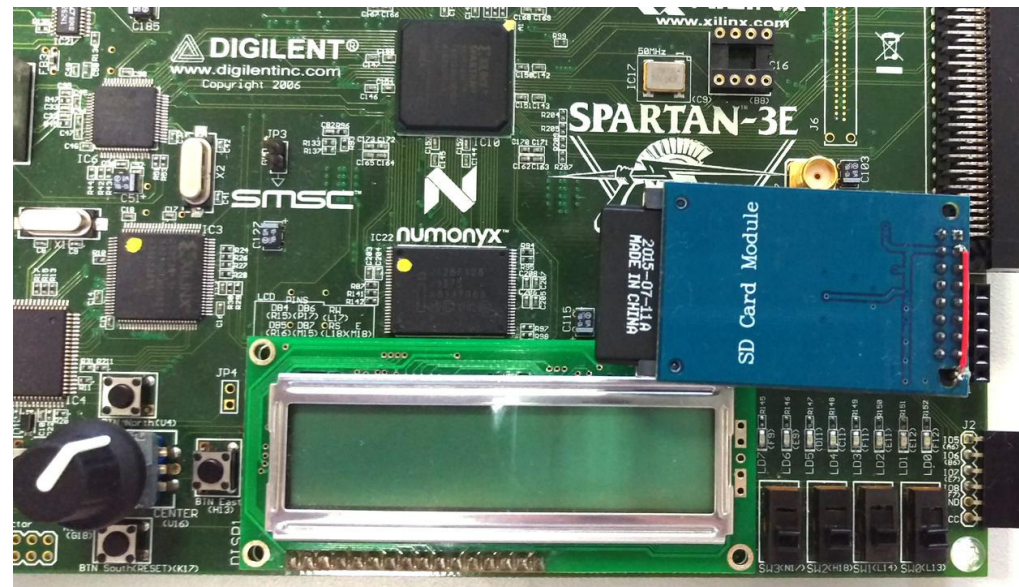
Lab 5: SD Card Reader Circuit



National Chiao Tung University
Chun-Jen Tsai
10/21/2016

Lab 5: SD Card Reader Circuit

- ❑ In this lab, you will design a circuit to read a text file from an SD card that has a sequence of 9 16-bit Hex numbers, and print the numbers in 3×3 format to the UART terminal
- ❑ You must demo the design to your TA on 11/1



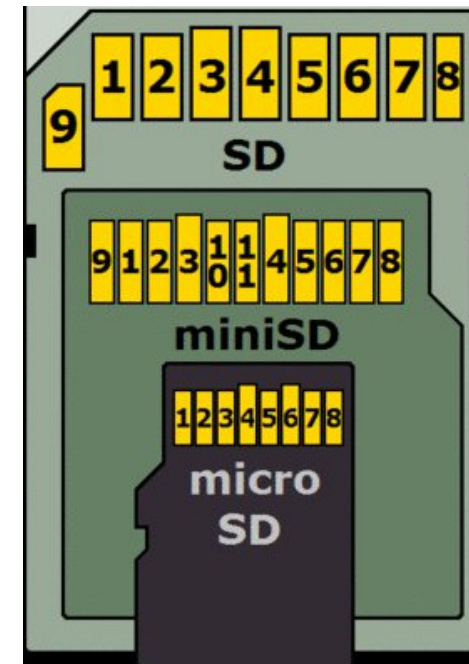
SD Card Specification

- ❑ The SD card that 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 7,736,319 (we use a 4GB SD card)
- ❑ In this lab, we use the serial SPI interface to read the SD card data

SD Card I/O Interface

- ❑ An SDHC card has three 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

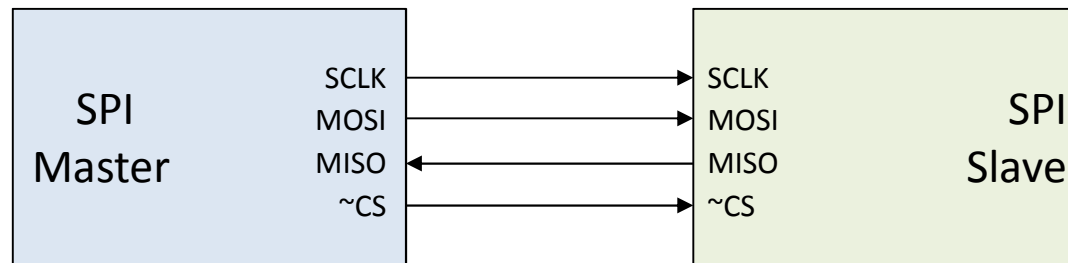


SD Card Initialization

- ❑ During the initialization phase, the SD card controller negotiates with the card to distinguish among different types of card: MMC, SD, SDHC, SDXC, 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, as long as the card can handle the speed

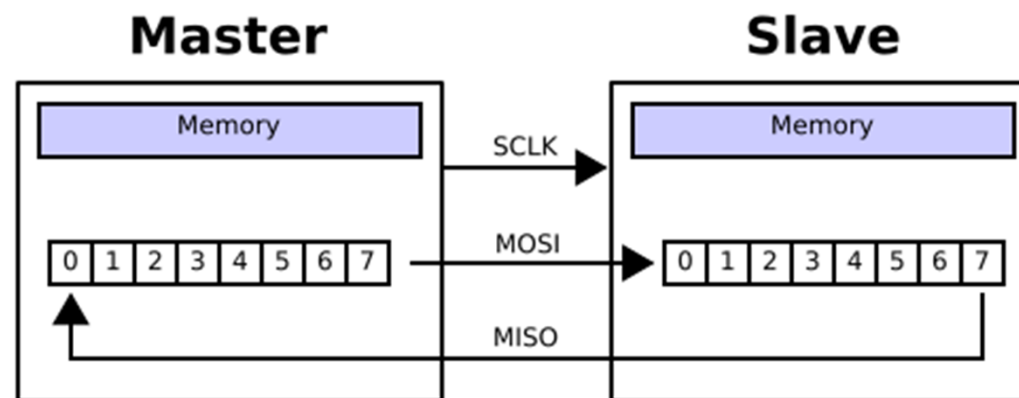
Serial Peripheral Interconnect (SPI)

- ❑ 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 selects the target slave via the CS pin first, then sends the 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
 - The data sampling clock edge (rising or falling) also depends on the device → read data sheet of the device!

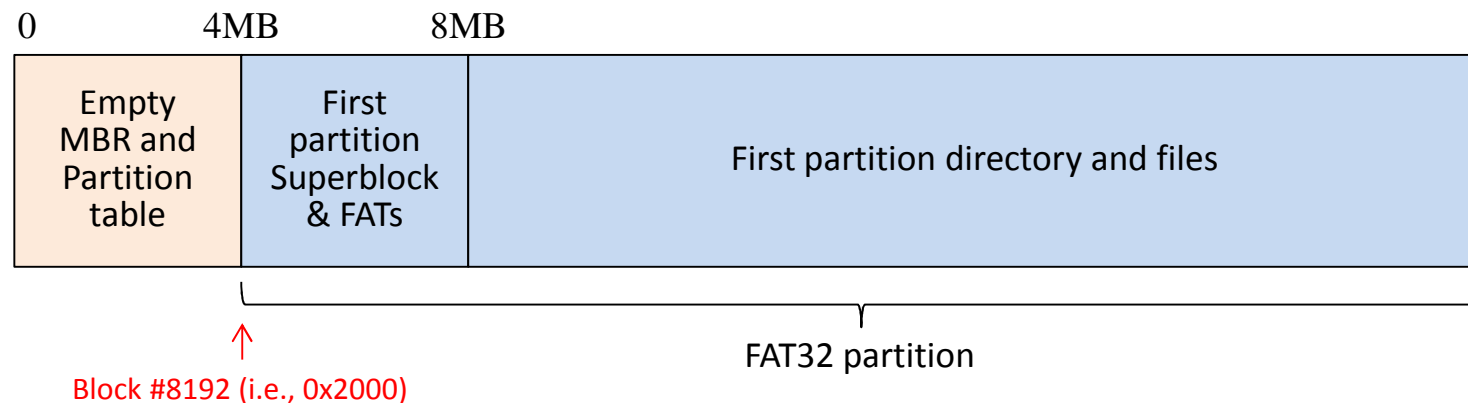


Physical Structure and File Systems

- ❑ 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 must first partition the SD card and then format a logical file system on that partition
- ❑ An SD card usually has one partition. However, it is possible to store multiple partitions and multiple file systems on a single SD card

Disk Partitions

- ❑ A physical disk can have several disk partitions, each partition can be formatted to a file system
- ❑ Typical partition structure of an SDHC card:



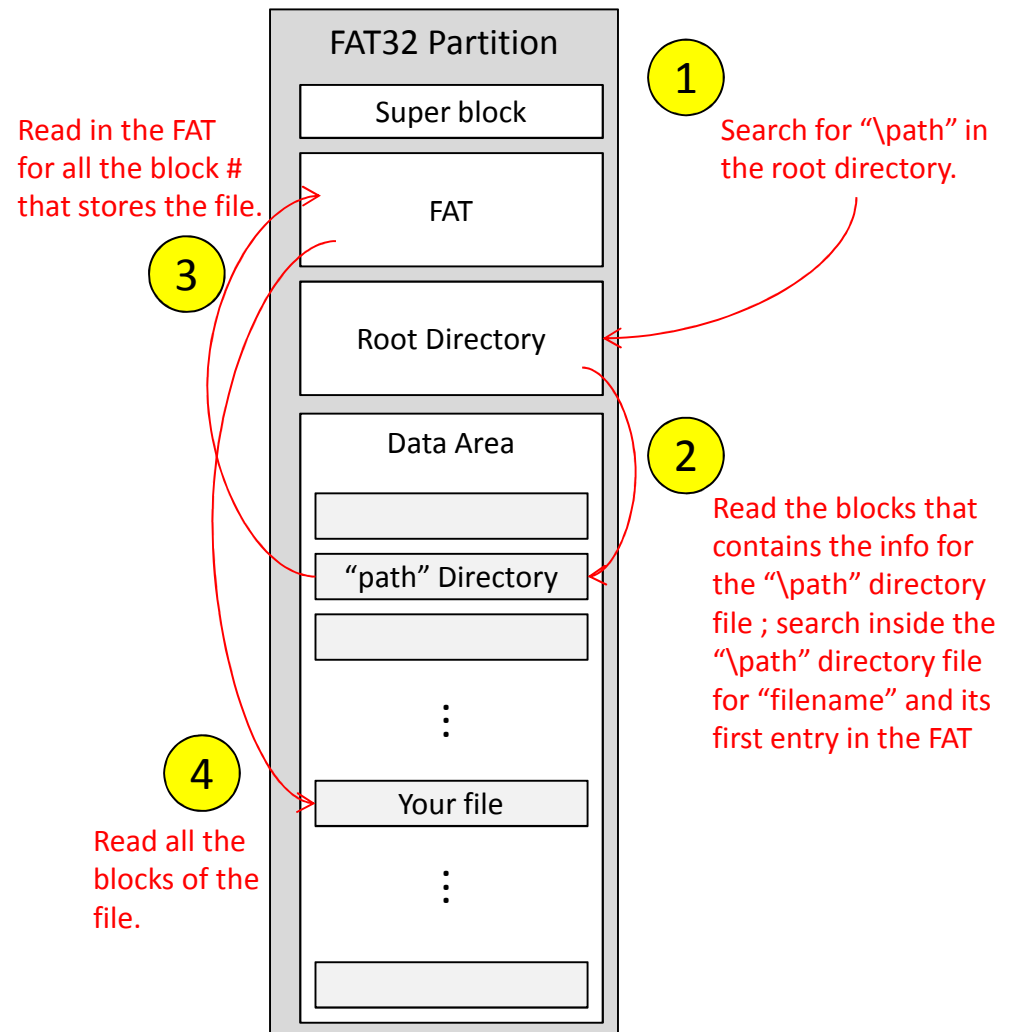
- ❑ If the card is bootable or has more than one partitions, then the Master Boot Record (MBR) and the partition table will not be empty

FAT32 Structure

- ❑ The FAT32 file system is a standard by Microsoft, and popularly used for mass storage devices
- ❑ An FAT32 file system has the following components:
 - Boot sector: 512 bytes, possibly block#0, a.k.a. the super block
 - The boot sector contains information such as the size of the FAT, root directories, boot code, 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

File Structure of an FAT32 Partition

- ❑ To read a file of “\path\name” in an FAT32 file system, we should follow the four steps shown in the figure.
- ❑ However, it is possible to read a **small** file without going through these steps!

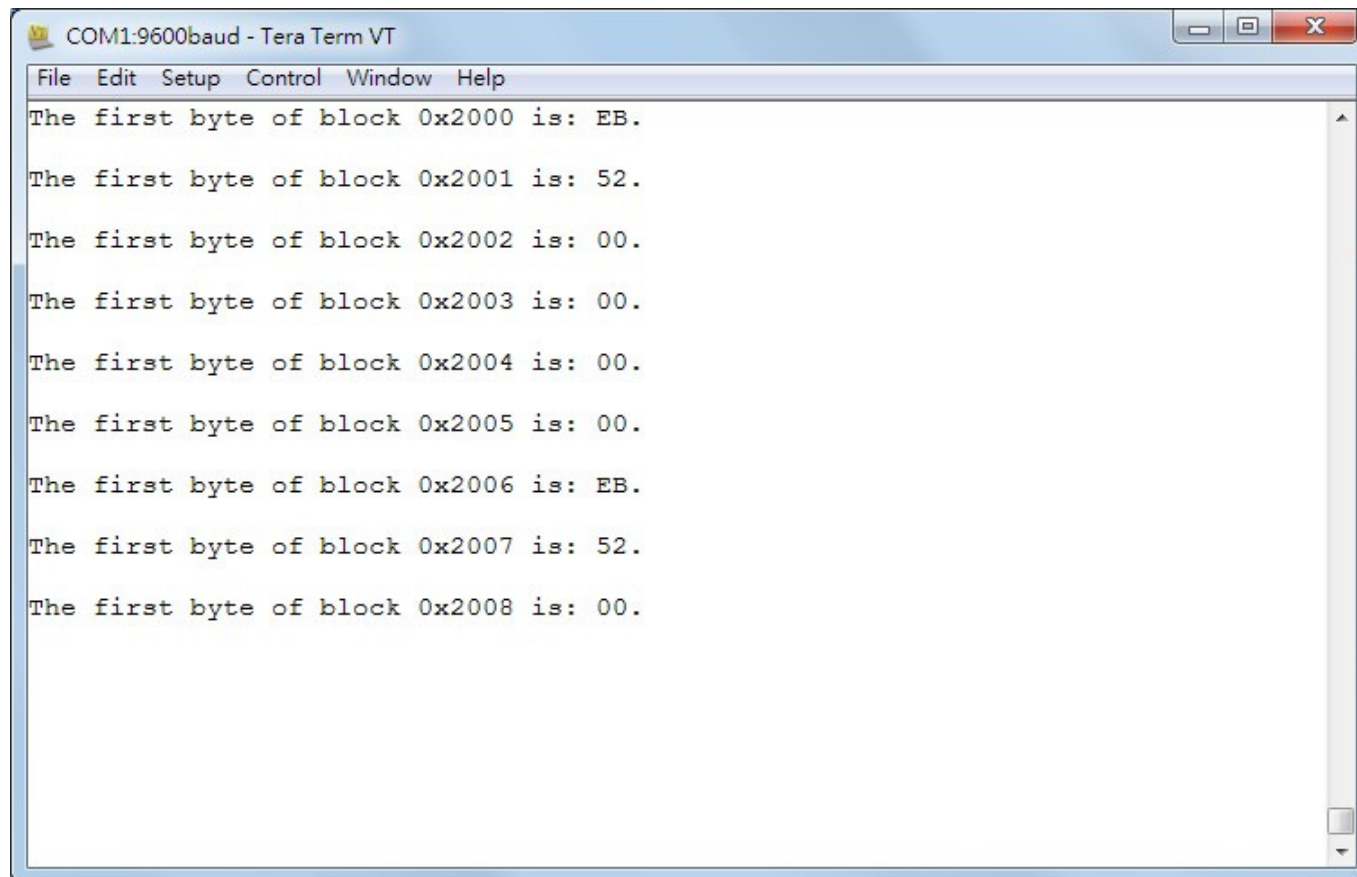


The Sample Code Package of Lab5

- ❑ For lab5, the sample package has a top-level circuit, `lab5.v`, and an SD card controller, `sd_card.v`
 - There are other supporting files such as: `debounce.v`, `uart.v`, `clk_divider.v`, and `lab5.ucf`
- ❑ The circuit performs the following actions:
 - When the board is powered up, the SD card controller will initialize itself and enter a ready state
 - Every time the user press the west button, the circuit will read a block of data from the SD card (starting at block #8192), and then print a message to the UART

The Output of Lab5 Sample Circuit

- ❑ A message is printed every time the West button is pressed. The figure shows the output of 9 button hits:

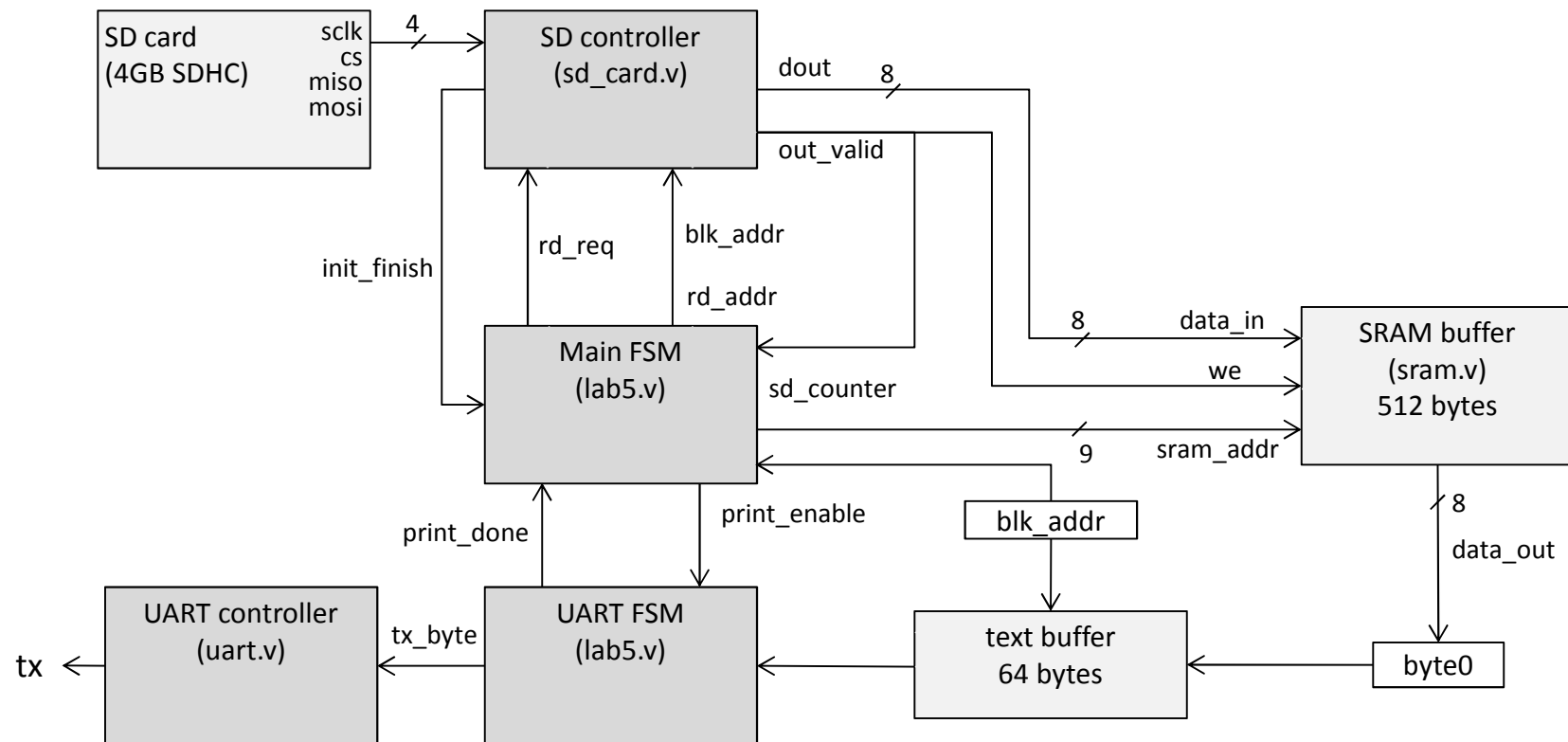


The screenshot shows a Tera Term VT window titled "COM1:9600baud - Tera Term VT". The window contains the following text:

```
File Edit Setup Control Window Help
The first byte of block 0x2000 is: EB.
The first byte of block 0x2001 is: 52.
The first byte of block 0x2002 is: 00.
The first byte of block 0x2003 is: 00.
The first byte of block 0x2004 is: 00.
The first byte of block 0x2005 is: 00.
The first byte of block 0x2006 is: EB.
The first byte of block 0x2007 is: 52.
The first byte of block 0x2008 is: 00.
```

System Diagram of the Sample Code

- ❑ The block diagram of the Lab5 sample code:



SD Controller Signals

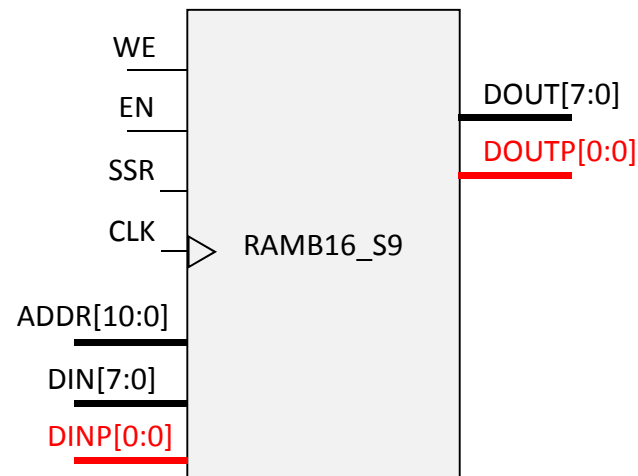
- ❑ The key signals that controls SD card operations:
 - `rd_req`: Triggers the reading of a block
 - `block_addr[31:0]`: The block # of the SD card to read
 - `init_finish`: SD card initialization is finished?
 - `dout[7:0]`: Output one byte of data in the block per clock cycle.
 - `sd_valid`: The output byte in `dout[7:0]` is ready
- ❑ If you set `rd_req` to 1 for one clock cycle, the SD card controller will read the data of the target block one byte at a time. Each time a data byte (of the 512 bytes) is ready, the flag `sd_valid` raise to 1 for one clock cycle.

Creation of SRAM Module

- ❑ In this lab, we need to create an Static RAM (SRAM) circuit module to receive the SD card data
 - Compare to Dynamic RAM (DRAM), an on-chip SRAM module can sustain a sequence of single-cycle read/write requests
- ❑ In Verilog, we can instantiate an SRAM module using explicit declaration or through inference
- ❑ On FPGAs, there are many high speed small memory devices that can be used to synthesize SRAM
 - These devices are called Block RAMs (BRAMs)
 - On Spartan3-500E, there are 20 **configurable** 18-kbit BRAMs

SRAM on FPGAs

- ❑ For example, the physical I/O ports of a BRAM can be configured to **emulate** a single-port 8-bit data, 1-bit parity memory with 2048 cells
 - The SRAM module can be instantiated by using the ISE predefined module name “RAMB16_S9”
 - The parity bit (DINP, DOUTP) of each byte is simply a 9th data bit, the SRAM do not perform parity checking or generation



BRAM Signals (1/2)

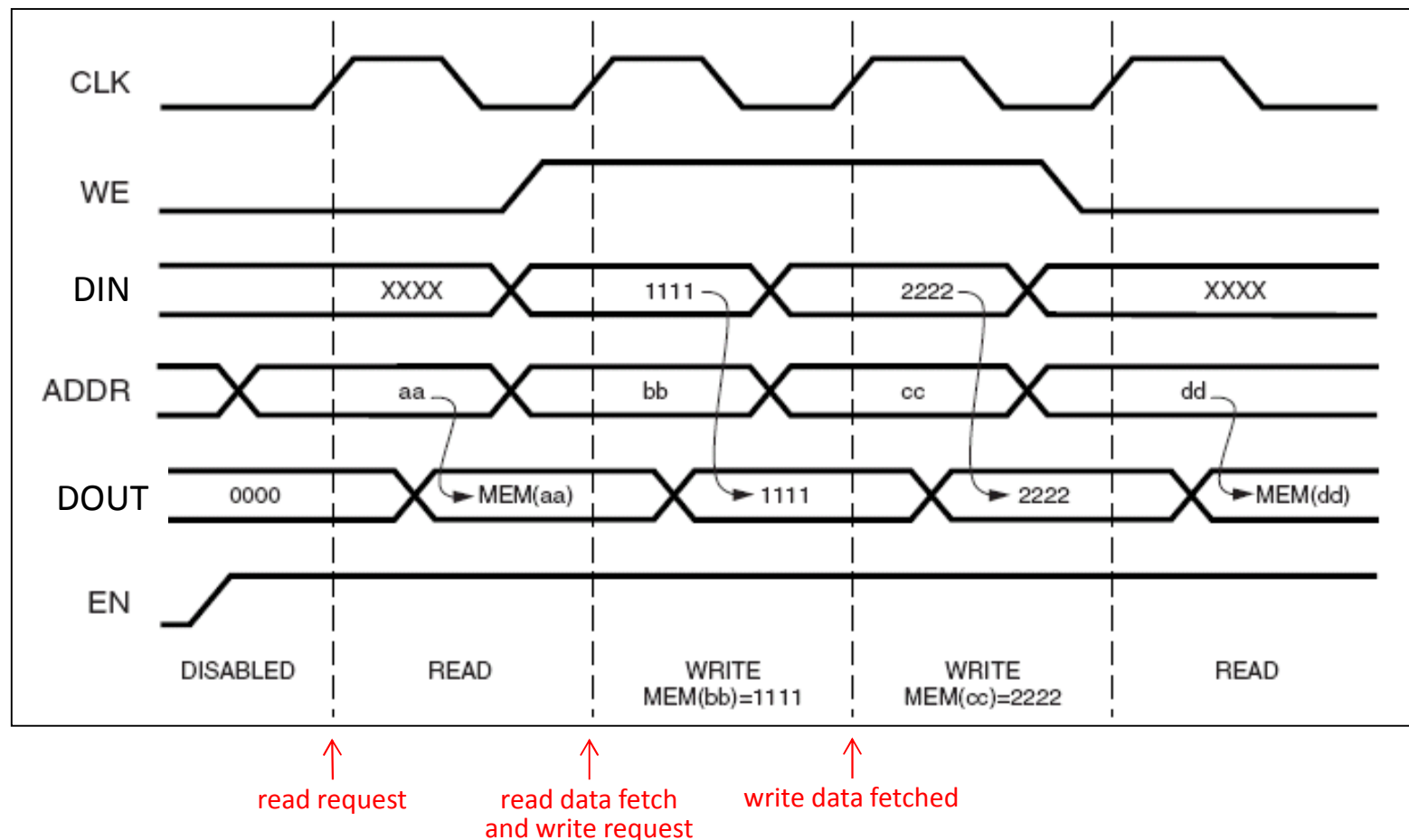
- ❑ **CLK** – Clock
 - Independent clock pins for synchronous operations
- ❑ **EN** – Enable
 - The read, write and reset functionality of the port is only active when this signal is enabled
- ❑ **WE** – Write enable
 - When active, the contents of the data input bus are written to the RAM, and the new 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
- ❑ **ADDR** – Address
 - The address bus selects the memory cells for read or write

BRAM Signals (2/2)

- ❑ **SSR** – Set/Reset
 - Set the memory to all zeros or all ones
- ❑ **DIN** – Data input port
 - The DI port provides the new data to be written into the RAM
- ❑ **DOUT** – Data output port
 - The DOUT port reflects the contents of the memory cells referenced by the address bus at the last active clock edge
 - During a write operation, the DOUT port reflects the DIN port
- ❑ **DINP, DOUTP** – Parity ports
 - Parity bits of the DIN and DOUT ports

Timing Diagram

- For single-port BRAM:



Instantiates an SRAM thru Inference

- ❑ The following Verilog code infers an SRAM block:

```
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
```

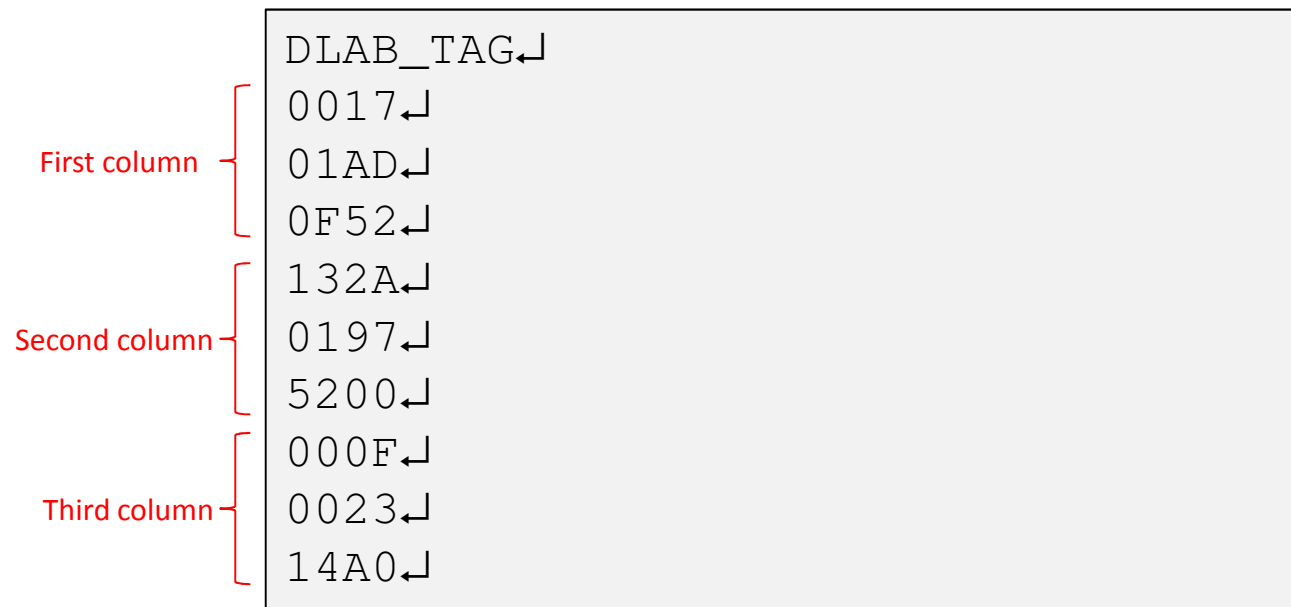
What You Need to Do for Lab5

- ❑ Modify the circuit such that:
 - Every time the west button is pressed, it triggers the SD card controller to read data blocks (starting at block #8192 all the way up to #7736319)
 - If a data block has been read into the SRAM buffer, check the first 8 bytes of the SRAM buffer to see if it begins with the string “DLAB_TAG”
 - If the string is matched, read the following text from the SRAM buffer and print the numbers to the UART device in a 3×3 matrix format

- ❑ Note that the text file will be less than 512 bytes so it can always be stored inside one SD card block

The Format of the Input Text File

- ❑ An example of the input text file is as follows. Note that the 3×3 matrix is stored in the text file in column-major format!



```
DLAB_TAG↵
0017↵
01AD↵
0F52↵
132A↵
0197↵
5200↵
000F↵
0023↵
14A0↵
```

The diagram illustrates the column-major storage of a 3×3 matrix. The text is grouped into three columns by red brackets on the left:

- First column:** 0017, 01AD, 0F52
- Second column:** 132A, 0197, 5200
- Third column:** 000F, 0023, 14A0

Note: ↵ stands for CR/LF (i.e., 0x0D, 0x0A)

The Format of Lab5 Output

- ❑ After reading the 9 numbers, you should print them out to the Tera Term window in the following row-major format:

```
The matrix is:↵
[ 0017, 132A, 000F ]↵
[ 01AD, 0197, 0023 ]↵
[ 0F52, 5200, 14A0 ]↵
↵
```

- ❑ Note that there is no need to do text-to-hex conversion!
You just have to reorder the text numbers for printing.

Final Comment

- ❑ The key lessons from this lab are:
 - How to control the SD card controller to read data files
 - How to read a 3×3 matrix stored in column-major order in the SRAM using the row-major order so that you can print the 2D matrix to the UART device

- ❑ Note that it is much easier to use a register array as the SD card buffer, instead of using SRAM.
However, you are not allowed to do that in this lab!!!