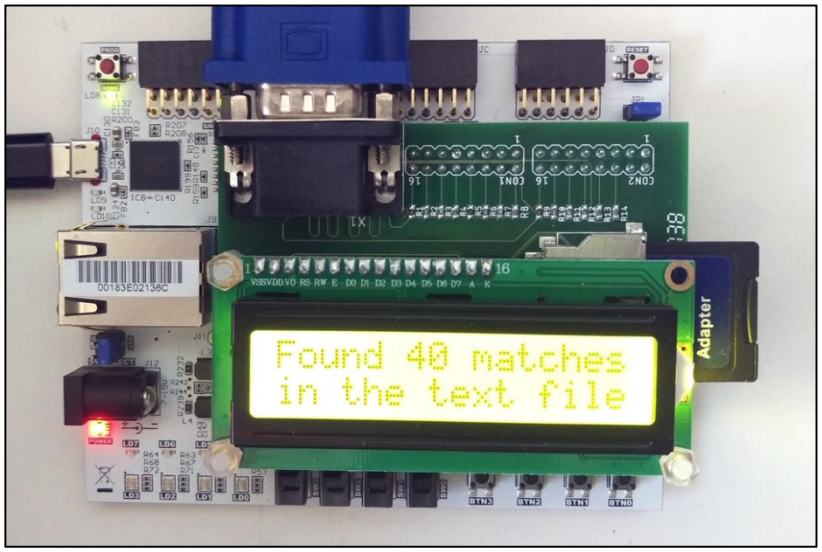
Lab 6: SD Card Reader Circuit

National Chiao Tung University
Chun-Jen Tsai
11/3/2017

Lab 6: SD Card Reader Circuit

- In this lab, you will design a circuit to read a text file from an SD card, and count the number of occurrence of “the” in the text and print it on the 1602 LCD
  - The number shown shall be a decimal number
  - The search of “the” shall be case insensitive
- The deadline of the lab is on 11/21



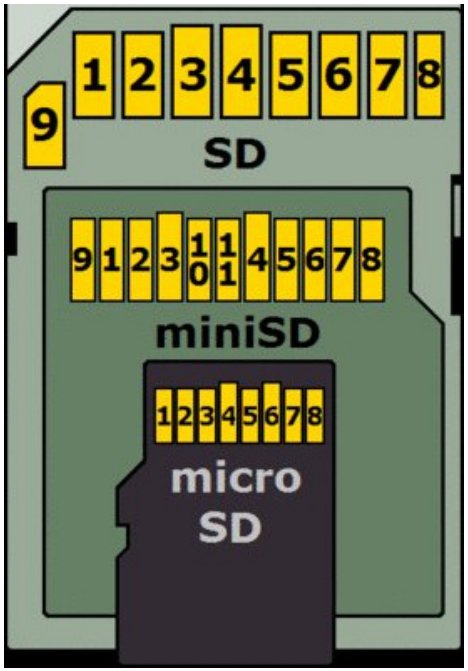
SD Card Specification

- The SD card that we use follows the SDHC standard, formatted with the FAT32 file system
- 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

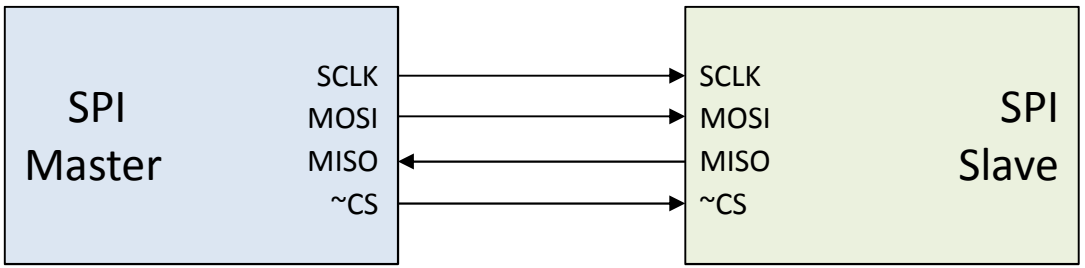


SD Card Initialization

- During the initialization phase, the SD card controller negotiates with the card to determine which type of card is used: MMC, SD, SDHC, SDXC, …, etc.
  - The controller uses a slower clock (500kHz) to talk to the SD card during the negotiation phase
- 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

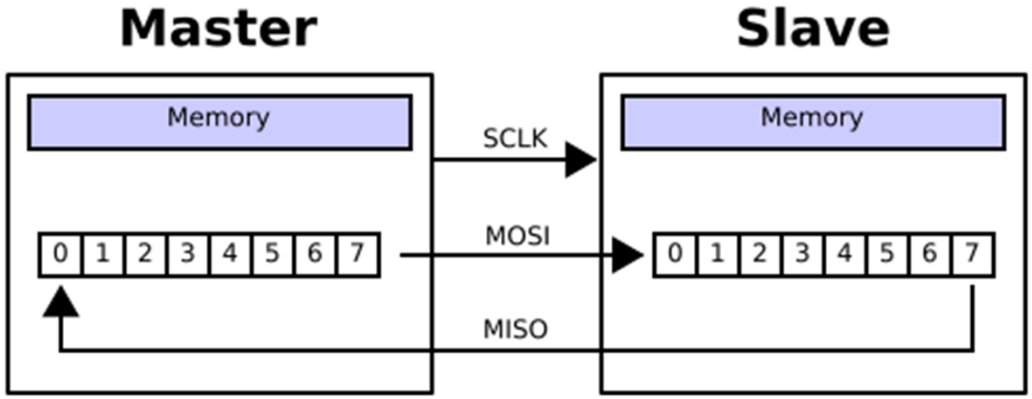
Serial Peripheral Interconnect (SPI)

- 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!

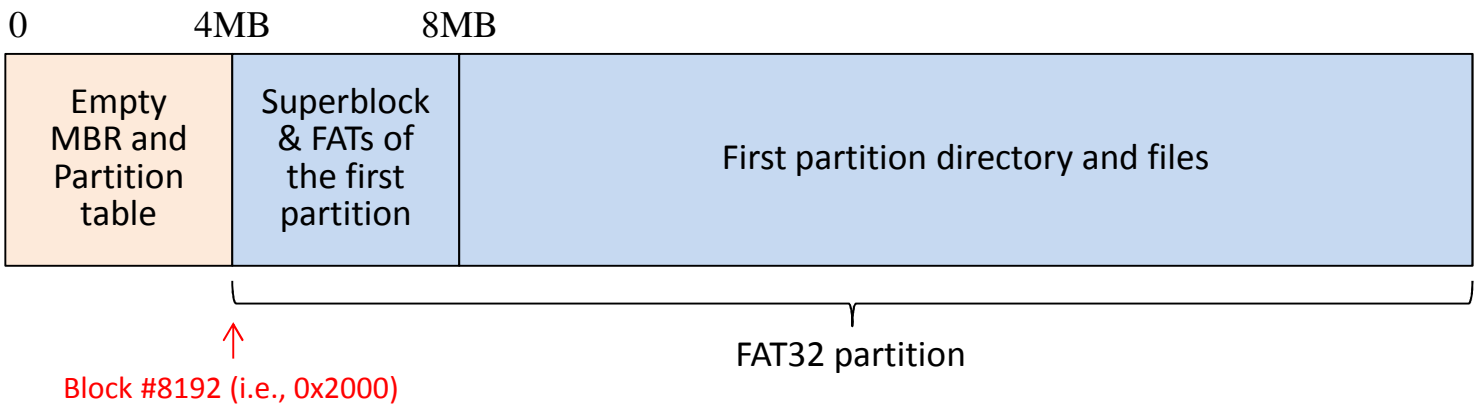


Physical Structure and File Systems

- The logical structure of an SD card is simply composed of a sequence of 512-byte blocks
  - Physically, SD cards are divided into 4KB ~ 32KB sectors
- To create directories for file storage 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:



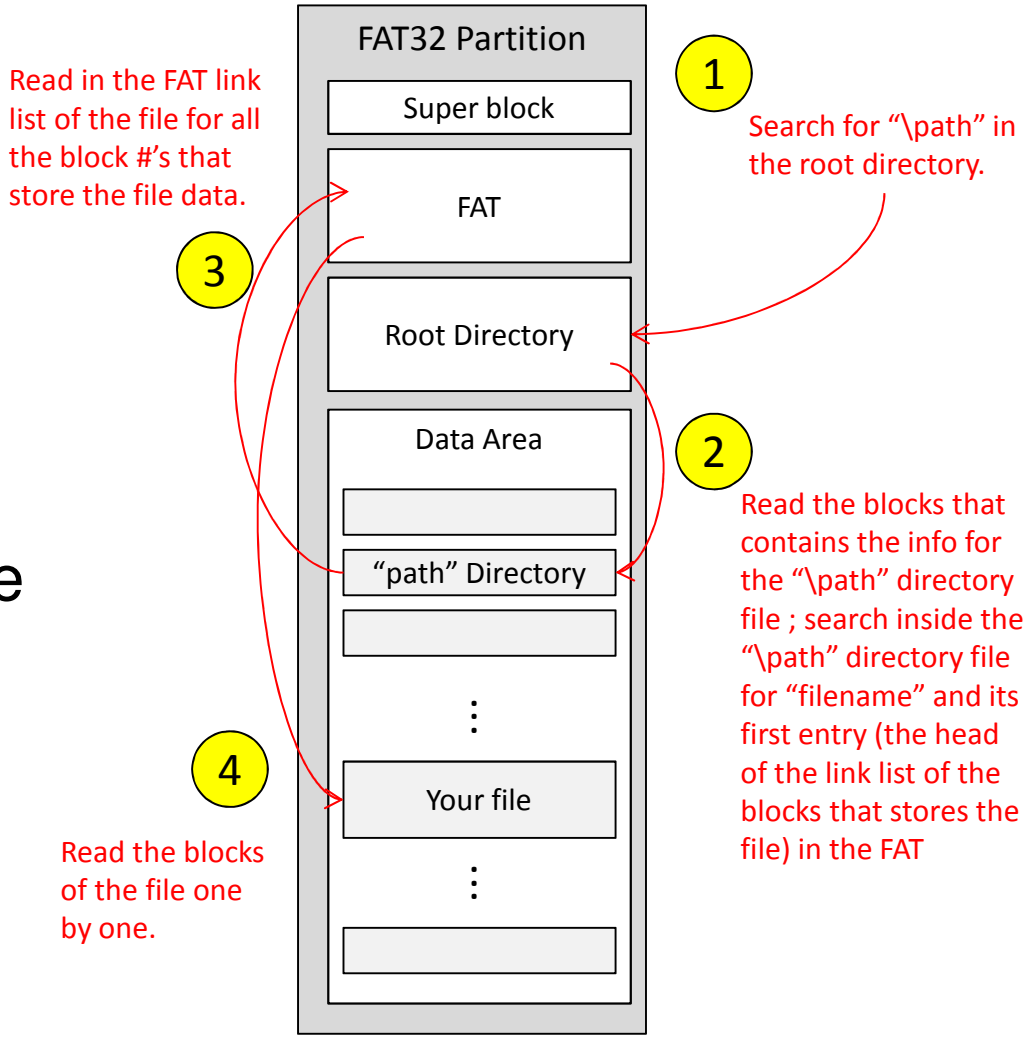
- If the card is bootable or has more than one partitions, then the Master Boot Record (MBR) and the partition table will not be empty

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); FAT basically contains many link lists of block numbers (one list per file)
  - 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, one shall follow the four steps shown in the figure.
- However, it is possible to read a small file without going through these steps!

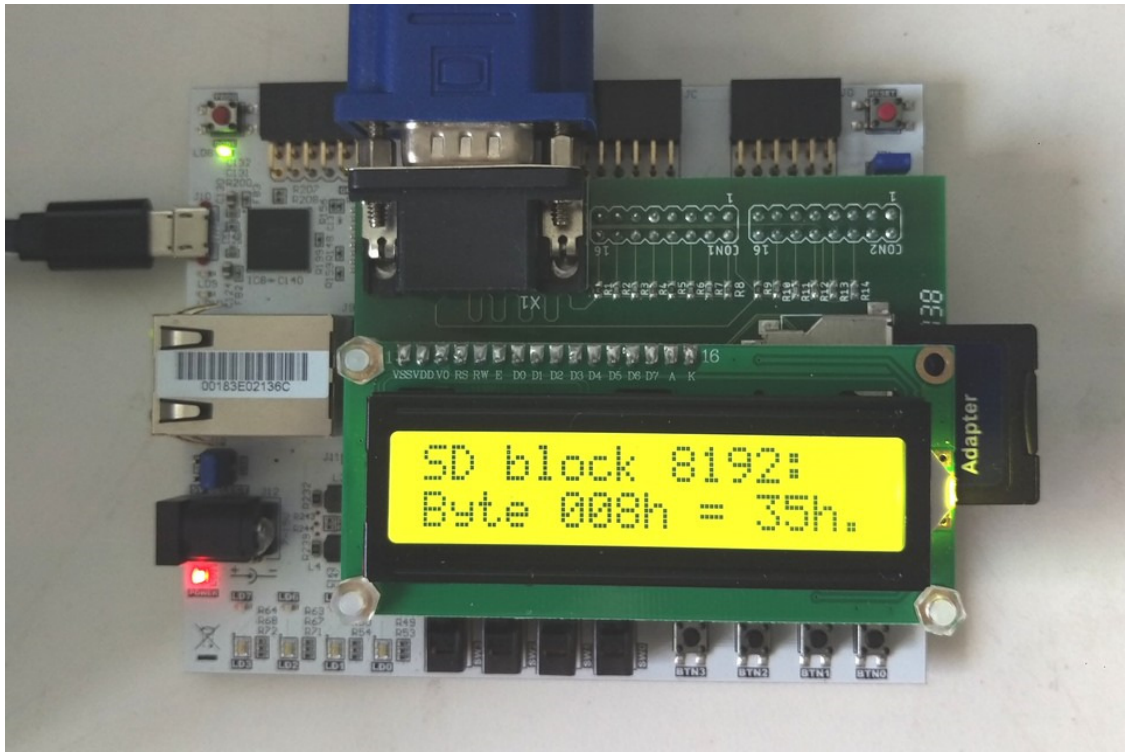


The Sample Project Files of Lab 6

- The sample project files of Lab 6 has a top-level circuit, `lab6.v`, an SD card controller, `sd_card.v`, and other supporting files such as: `debounce.v`, `LCD_module.v`, `clk_divider.v`, and `lab6.xdc`
- The circuit performs the following actions:
  - When the board is powered up, the SD card controller will initialize itself and enter a ready state
  - When the user presses BTN2, the circuit reads a block of data from the SD card, and then print the first byte in the block on the LCD module
  - Every time BTN2 is pressed, the next byte will be displayed

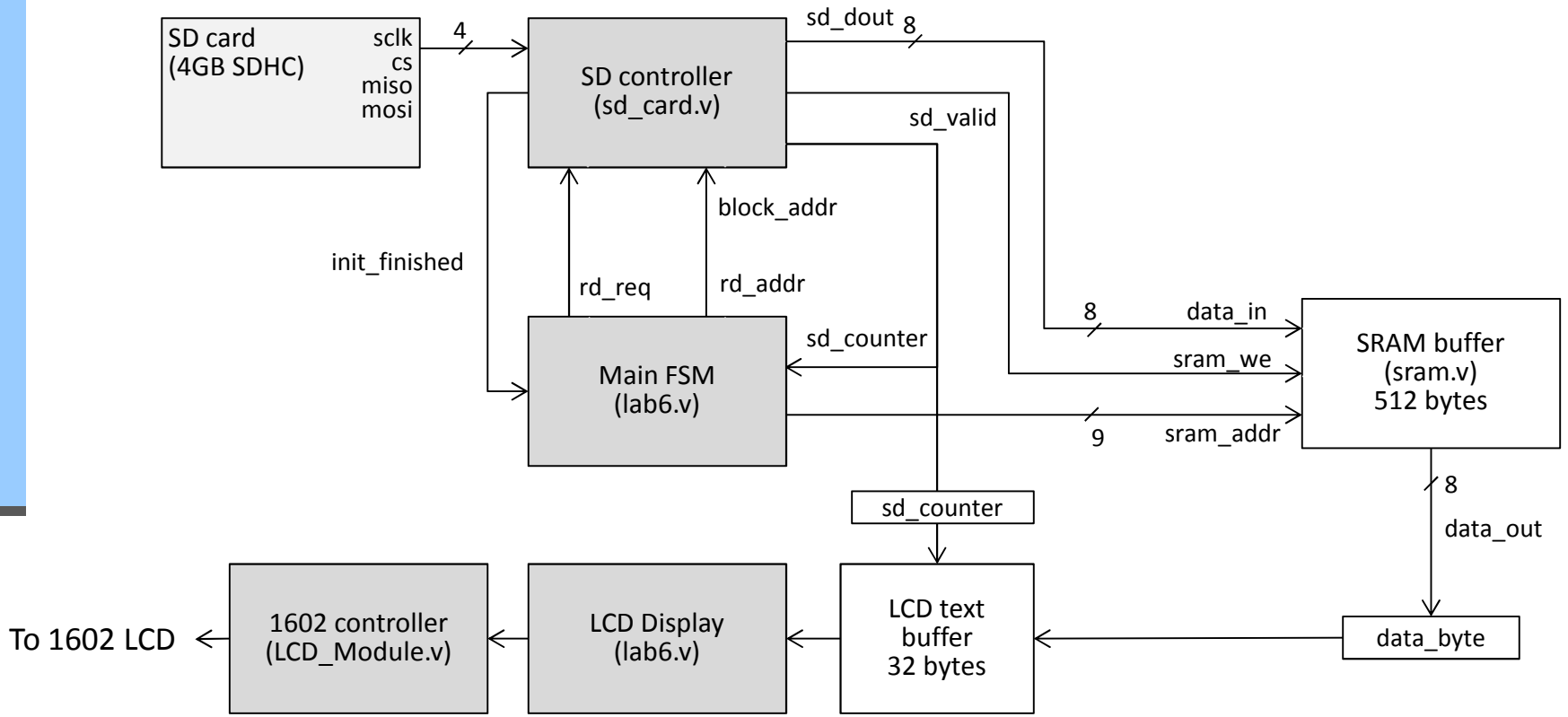
The Output of Lab 6 Sample Circuit

- For example, the following message is printed after 9 button hits:



System Diagram of the Sample Code

- The block diagram of the sample code of Lab 6:



SD Controller Signals

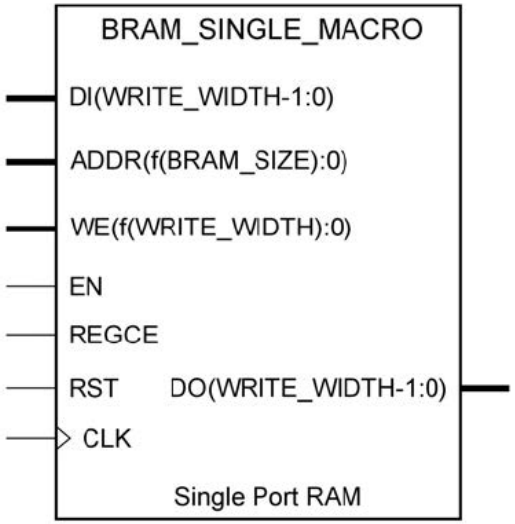
- The key signals that controls SD card operations:
  - rd_req: Triggers the reading of a block
  - block_addr[31:0]: The block # of the SD card to read
  - init_finished: SD card initialization is finished?
  - dout[7:0]: Output one byte of data in the block per clock cycle.
  - sd_valid: The output 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 the SRAM Module

- In this lab, we need to create a static RAM (SRAM) circuit module to receive the SD card data
  - Compared to dynamic RAM (DRAM), an on-chip SRAM module can sustain a sequence of single-cycle read/write requests
- On FPGAs, there are many high speed small memory devices that can be used to synthesize SRAM
  - There are two possible devices on FPGA for SRAM synthesis: distributed RAMs and block RAMs (BRAMs)
  - On Artix-7 35T, there are 313 kbits of distributed RAMs and 50 blocks of 36-kbit BRAMs

SRAM on FPGAs

- In Verilog, we can instantiate an SRAM module using explicit declaration† or implicit inferencing
  - For example, a single-port SRAM can be instantiated using the module BRAM_SINGLE_MACRO in Vivado



Port	Direction	Width	Function
DO	Output	See Configuration Table below.	Data output bus addressed by ADDR.
DI	Input	See Configuration Table below.	Data input bus addressed by ADDR.
ADDR	Input	See Configuration Table below.	Address input bus.
WE	Input	See Configuration Table below.	Byte-Wide Write enable.
EN	Input	1	Write/Read enables.
RST	Input	1	Output registers synchronous reset.
REGCE	Input	1	Output register clock enable input (valid only when DO_REG=1).
CLK	Input	1	Clock input.

† Xilinx UG953, *Vivado Design Suite 7 Series FPGA and Zynq-7000 All Programmable SoC Libraries Guide*, page 120.

General SRAM Signals (1/2)

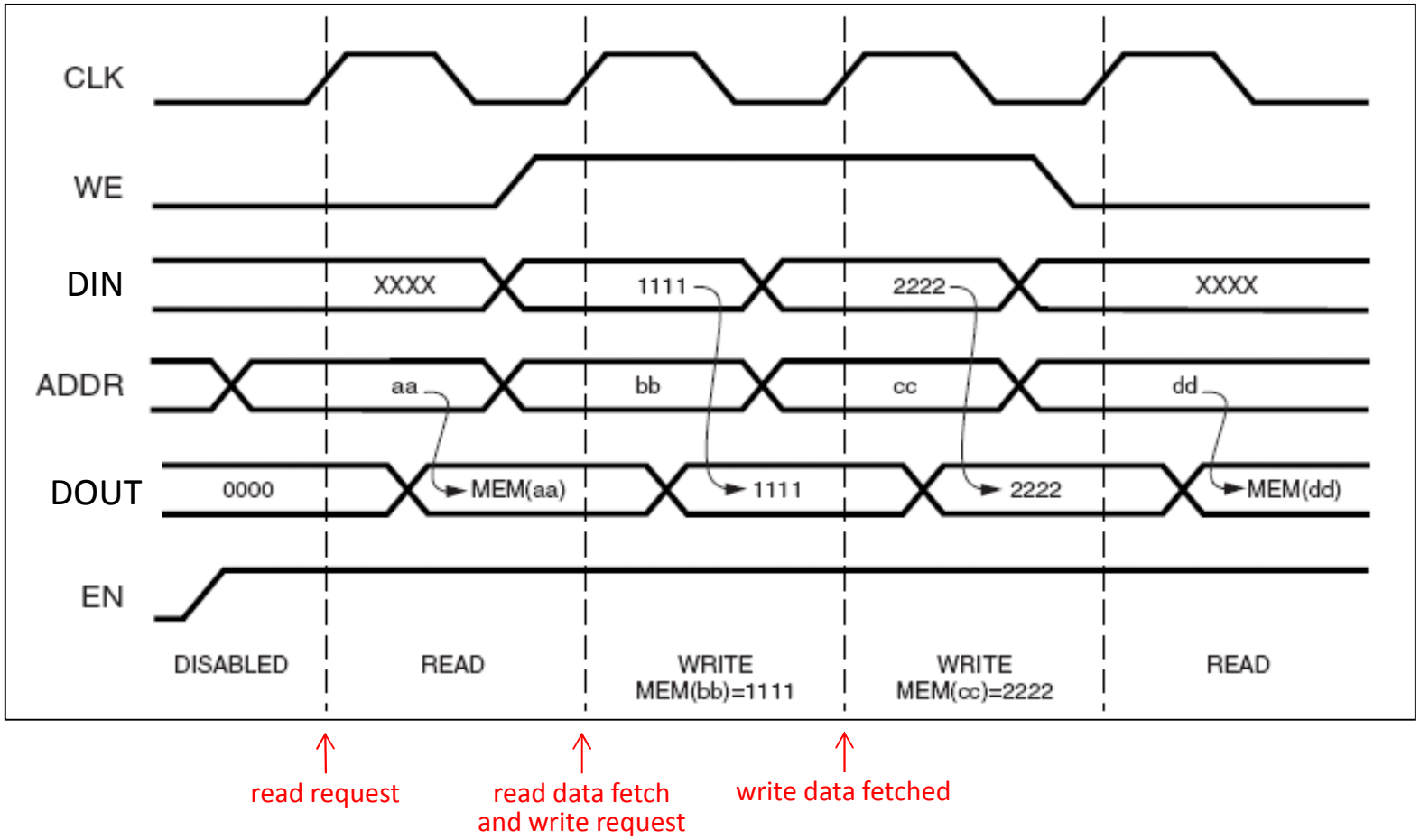
- **CLK** – Clock
  - Independent clock pins for synchronous operations
- **EN** – Enable
  - The read, write and reset functionality of the port is only active when this signal is enabled
- **WE** – Write enable
  - When active, the contents of the data input bus are written to the RAM, and the new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

General SRAM Signals (2/2)

- **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

Timing Diagram

- For single-port SRAM:



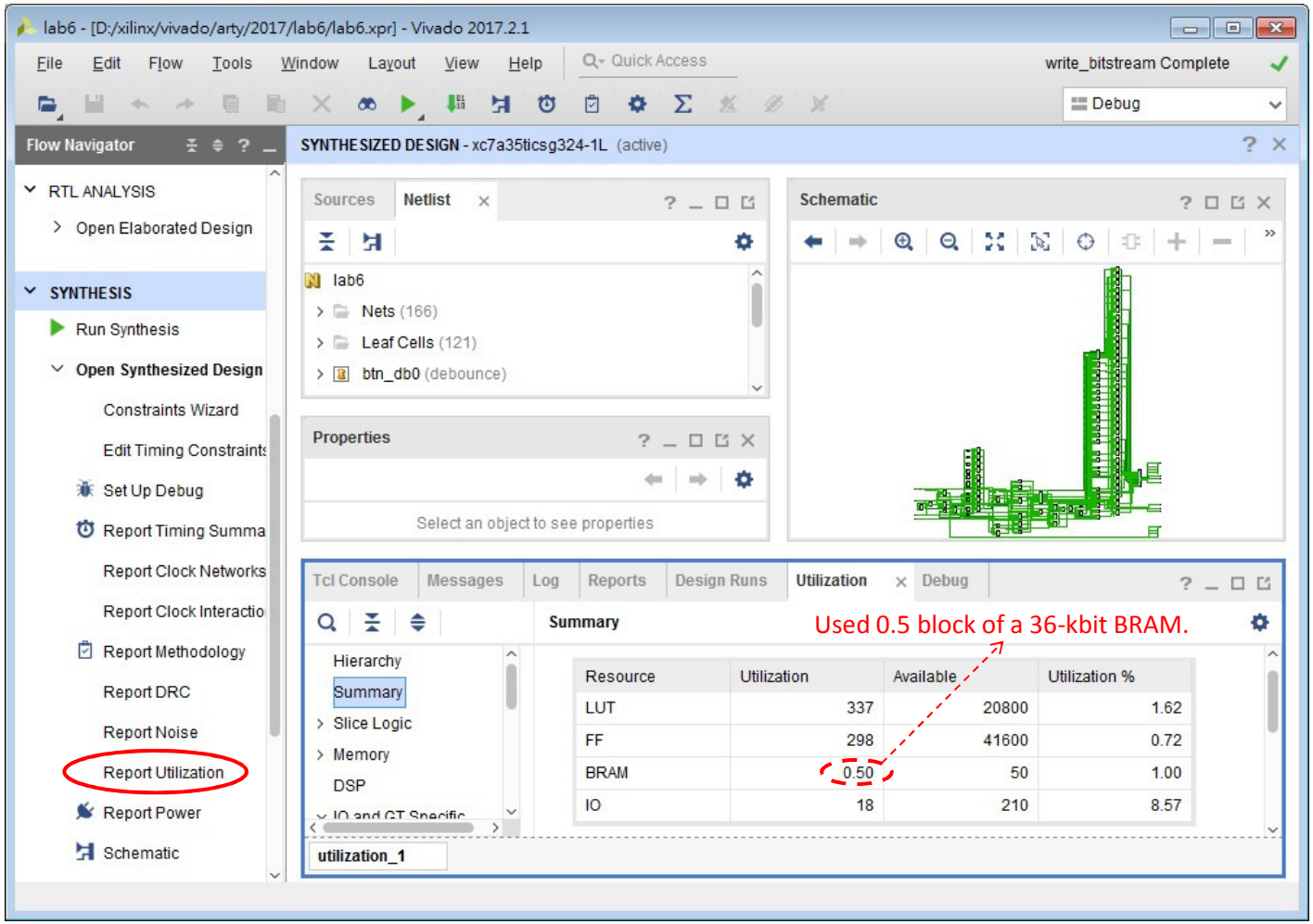
Instantiates an SRAM via Inference

- The following Verilog code infers an SRAM block:
  - The minimal allocation size of SRAM on Artix-7s is 18-kbit

```
reg  [7:0] sram[511:0];
wire       sram_we, sram_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 (sram_en && sram_we) begin
    sram[sram_addr] <= data_in;
  end
end

always @(posedge clk) begin // Read data from the SRAM block
  if (sram_en && sram_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
```

Check FPGA Resource Utilization

- Verify that your SRAM inference is successful:



What You Need to Do for Lab 6

- You must design a circuit that:
  - When BTN2 is pressed the circuit reads the SD card and search for an ASCII file, test.txt
  - The file begins with the word “DLAB_TAG” and ends with the word “DLAB_END”
  - The circuit reads through the text file and counts the number of occurrences of “the” in the text file, and print the decimal number to the 1602 LCD module
  - The search for the word “the” shall be case insensitive
  - Each “the” is separated by punctuation marks or line feeds from the other words
  - There shall be less than 100 occurrences of “the” in the text file

The Format of the Input Text File

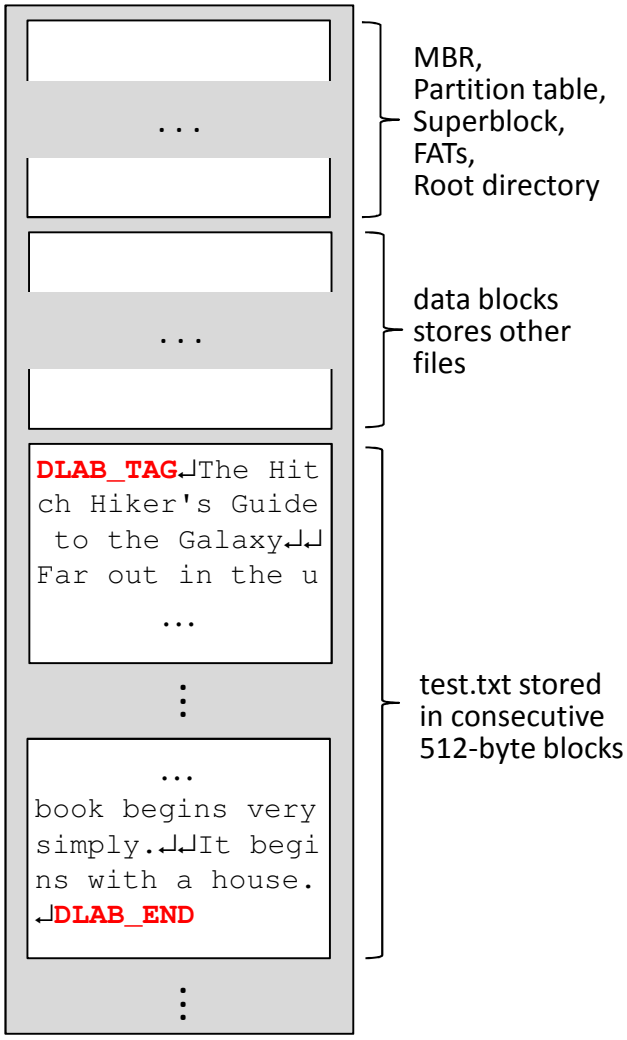
- The sample input text file, test.txt, is as follows.

```
DLAB_TAG↵
The Hitch Hiker's Guide to the Galaxy↵
↵
Far out in the uncharted backwaters of the unfashionable end of
the western spiral arm of the Galaxy lies a small unregarded
yellow sun.↵

                                . . . . .


But the story of this terrible, stupid Thursday, the story of its
extraordinary consequences, and the story of how these
consequences are inextricably intertwined with this remarkable
book begins very simply.↵
↵
It begins with a house.↵
DLAB_END↵
```

Note: ↵ stands for 0x0A.

The layout of the File on the SD card

- A newly formatted SD card will have its initial files stored in consecutive 512-byte blocks
- After some files are deleted, new files added may be stored in non-contiguous blocks
- You can assume test.txt is stored in consecutive blocks



Some Comments

- Although it is easier to use a register array as the SD card buffer, however, you are not allowed to do that in this lab! You must use an SRAM as the SD card buffer
- Note that the signal 'sd_valid' may not always be 1 during the reading of a block of 512 bytes
- The word "the" may lay across two sectors
- It is better not to use a large SRAM to read the whole file before searching for all the occurrences of "the"