EE 577B Project Phase 1
Spring 2014 Nazarian

Score:___/100

Student ID: _____ Name: _____
Assigned: Thursday, February 20th
Due: Friday, March 7th at 11:59pm (View/Complete)
Late submissions will be accepted two days after the deadline with a maximum of 15% penalty for each day. For each day, submissions between 12 and 1am: 2%, 1 and 2am: 4%, 2 and 3am: 8% and after 3am: 15%.

Notes:

- The final project phases are based on teams of up to 2 students. No collaboration is allowed across teams. Please watch the first lecture of this course regarding the academic integrity policies and also refer to the syllabus for a summary of AI policies (including the penalties for any violation.) If you have any doubts about what is allowed or prohibited in this course, please contact the instructor.

Introduction

This Project is intended to provide the students an opportunity to learn about DDR2 SDRAM devices and their source synchronous double data rate bus interface timing. A Verilog model provided by Denali for 512Mb (32Mb x16 four bank) DDR2 would be used in this project. Students are going to design and implement a DDR2 controller in Verilog HDL and simulate their designs along with Denali's DDR2 model using Cadence NC-Verilog. The DDR2 controller is going to provide a simple FIFO based front-end that would support write and read transactions like scalar, block and atomic to and from the DDR2 SDRAM. Students would refer to the JEDEC DDR2 SDRAM Standard (JESD79-2C) for all timing, bus interface and initialization specifications. The controller would initialize the DDR2 model (chip) with the given parameters like CAS latency and Burst length etc. Normal data transactions would start after a successful completion of the DDR2 initialization sequence.

• At the first time you run DDR2 simulation, you need to copy the provided setup.csh file to your account and source it
• DDR2 part to be used for the project: Micron MT47H32M16BN-37E
• System clock frequency of DDR2 controller = 500MHz
• DDR2 clock to be run at 250 MHz
• (For Project Part 1-2) Oklahoma State University's 0.18um standard cell library to be used for synthesis. The information on each standard cell in the library can be found at http://avatar.ecen.okstate.edu/projects/scells/tsmc018/indexframe.html

Figure 1 shows a cartoon of DDR2 controller. Students have full freedom to improve the design to achieve higher data bandwidth. The heart of the DDR2 controller core is the logic that would be responsible for the following tasks:

(1) Start initialization sequence logic for the DDR2. When INITDDR input is asserted (captured high at the positive edge of the clock) this logic would initialize the DDR2. It asserts an output signal READY when initialization sequence is completed.

(2) Fetch the command/data queued in the input FIFOs and complete the DDR2 Transaction with correct timing without wasting a single unnecessary cycle.

(3) Before de-queuing a read command from input Cmd/Addr FIFO the FSM must ensure that the output FIFO's would have enough space to accommodate the data received from DRAM in response to a read command.

(4) Match or generate correct return address for the data received from the DRAM.

(5) Refresh logic: This logic would cyclically issue DRAM refresh commands

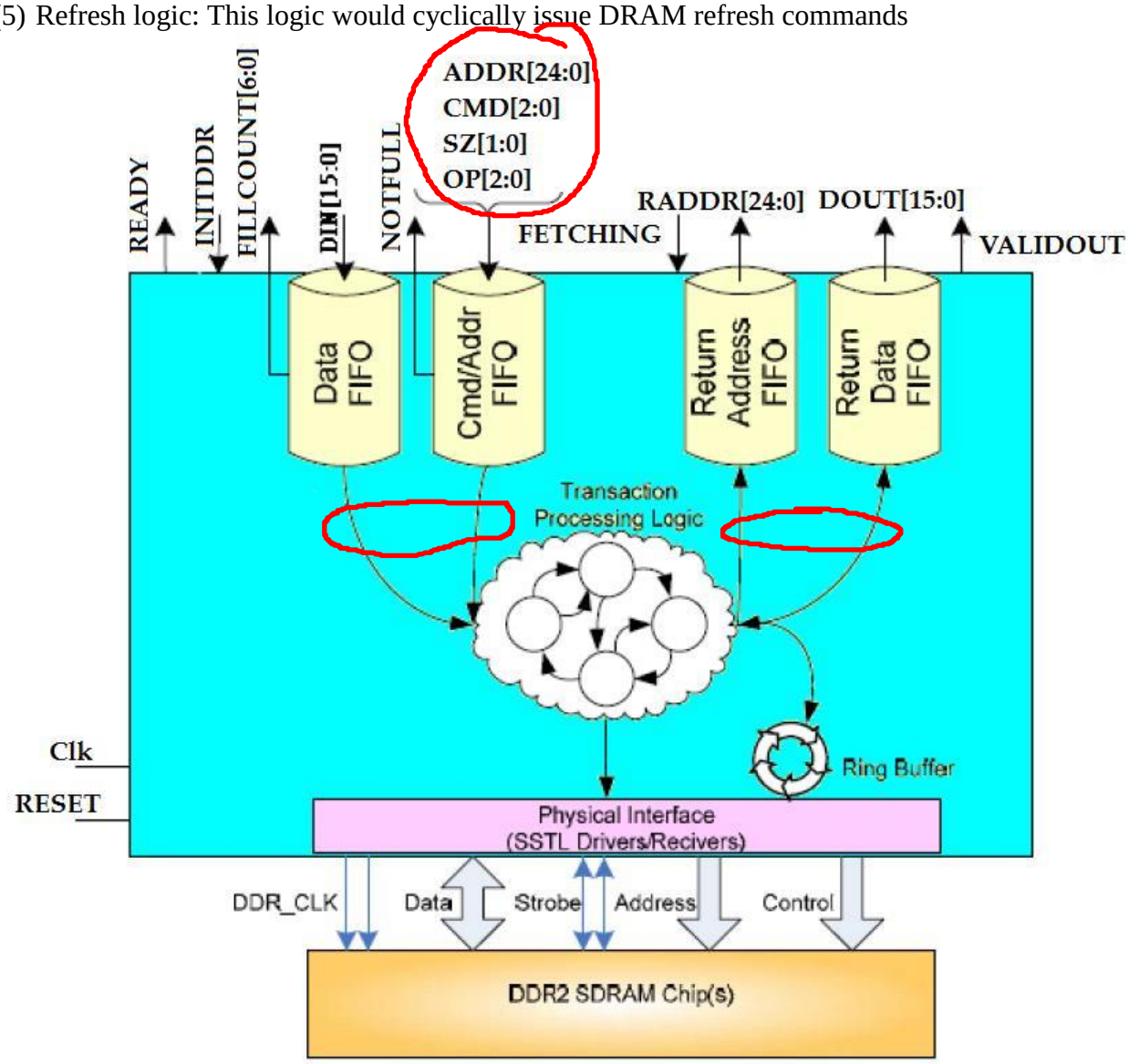

Figure 1

DDR2 Controller Internal Bus I/Os

I/O Name	Width	IN/OUT	Description
CMD[2:0]	3 bit	IN	000: No Operation (nop) TBA

SZ[1:0]	2 bit	IN	00: Block size of 8 words 01: Block size of 16 words 10: Block size of 24 words 11: Block size of 32 words
OP[2:0]	3 bit	IN	TBA
DIN[15:0]	16 bit	IN	Data to be written into the DDR2 SDRAM
ADDR[24:0]	25 bit	IN	Address of the location to be accessed
DOUT[15:0]	16 bit	OUT	Data retuned: Read from DRAM
RADDR[24:0]	25 bit	OUT	Address of the returned data
VALIDOUT	1 bit	OUT	Output valid: High when the address and data at the output of the controller is valid
FETCHING	1 bit	IN	This input tells the controller that the data and address at its output is being fetched this cycle
FILLCOUNT[6:0]	7 bit	OUT	Fill count of the internal input 64 deep data FIFO
NOTFULL	1 bit	OUT	The internal command FIFO is not full
INITDDR	1 bit	IN	Starts DDR2 Initialization sequence. This input is ignored until the sequence is complete.
READY	1 bit	OUT	Goes high after the DDR2 initialization is complete.
RESET	1 bit	IN	Synchronous active high reset input to the controller
CLK	1 bit	IN	500 MHz

DDR2 Controller Configuration Parameters

Parameter	Values
BL	Burst Length = 8 (You need to set A[2:0] in MRS refer to Figure 34 on the data sheet)
BT	Burst type = sequential { A[3] = 0 in MRS }
CL	CAS Latency = 3 (You need to set A[6:4] in MRS refer to Figure 34 on the data sheet)
AL	Additive Latency = 1 (You need to set A[6:4] in MRS refer to Figure 34 on the data sheet)

DDR2 PADs:

All DDR2 I/O Pins are connected through SSTL18 driver/receiver except for the following:
(1) ck and ckbar would be connected using differential SSTL18 driver.
(2) dqsbar would not be connected and set to Hi-Z by setting A[11:10] = 2'b01 of EMR

Bus Interface Protocol

All data, address and control signals are presented and captured in the current cycle. Example of input interface:

The data, address, command, size and operation (OP) is presented valid and stable well **before (setup time) the positive edge of the clock**. In case of block write command, during the first cycle all input fields would be valid. In the subsequent SZ-1 cycles only data would be valid. Designer must mask out the invalid fields during the cycles they are not expected to be valid.
Example of the out interface:

During a cycle if VALIDOUT is high the DOUT and RADDR must be valid. If FETCHING is captured high this cycle the controller must output the next valid data and address in the next cycle and keep the VALIDOUT high, only if valid data is available.
[Hint: The input signal FETCHING can be used to generate the "read" of the output FIFO]

Parameterized synchronous FIFO

A FIFO (first-in, first-out) buffer is a special-purpose register file which operates as follows: When the input **"put"** is asserted, if the output **"full"** is not active, data at the input **"data_in"** is captured and stored into the queue. A previously putted data, at the head of the queue, remains available at the output **"data_out"**. The reader of the FIFO checks the output signal **"empty"** and asserts input **"get"** to tell the FIFOs that it is collecting the data from the head of the queue. The FIFO puts the next data at the output **"data_out"** in the next cycle. In most FIFOs, including this one, **"put"** and **"get"** can be asserted simultaneously unless the FIFO is **"full"** or **"empty"** respectively**.** The common practical implementation of a FIFO is the circular FIFO. In this form, there is a write pointer **"wr_ptr"** pointing to the current row to be written. When the **"put"** signal is asserted, data at the input is written into the location and the write pointer **"wr_ptr"** advances to the next row to be written. When the last row is reached, **"wr_ptr"** advances to the first row. Similarly, there is a read pointer **"rd_ptr"** which points to the next row to be read; when **"get"** is asserted, **"rd_ptr"** advances to the next row to be read in the next cycle. Output **"empty"** is asserted when **"wr_ptr"** and **"rd_ptr"** point to the same row; it indicates that no valid data has been written into the FIFO yet. The output **"full"** is asserted when **"wr_ptr"** has advanced so far ahead of **"rd_ptr"** that the next **"put"** signal would over-write the data at the head of the queue before it has been read out, thus causing loss of valid data in the queue. Write a parameterized Verilog module for a circular FIFO of rows defined as power of two (i.e. for 32 row "DEPTH_P2 = 5") and "WIDTH = 8" bit width. Assume the data and control ports described above are synchronous to a single clock **"clk"**. Use synchronous reset signal which resets **"wr_ptr"** and **"rd_ptr"** to row 0 thus making the FIFO empty.

```
module FIFO (clk, reset, data_in, put, get, data_out, empty, full, fillcount);
parameter DEPTH_P2 = 5; // 2^ DEPTH_P2;
parameter WIDTH = 8; output
[WIDTH-1:0] data_out; output
empty, full;
output [DEPTH_P2:0] fillcount; //Why do we need an more bit in Figure 1?
input [WIDTH-1:0] data_in;
input put, get;
input reset, clk;
//your code here
endmodule
```

Note:
(1) You may take the FIFO design from Lab3 part2 (some changes are required if needed).
(2) This parameterized FIFO design can be the proto-type of the DATA FIFO, CMD/Address FIFO, Return Address FIFO, and Return Data FIFO in your controller.
(3) You are free to change the FIFOs for Phase2 and Phase3.

Implement the following components of the DDR2 controller:

(1) Parameterized synchronous FIFOs with width and depth parameters which are specified above.

(2) DDR2 Initialization State machine.

(3) Implement the SSTL18 based interface for the DDR2 using (instantiating) the two SSTL18 models provided. Notice that every IO of DDR2 controller that connects to a DDR2 chip must go through an independent SSTL driver/receiver. When sending data from the controller to Denali module, set TS=1 and RI = "don't care". When receiving data from Denali module to the controller, set TS= 0 and RI =1

Implement the DDR2 controller module populate it with the following:

(1) 16 bit wide and 64 deep input "Data FIFO". Logic to connect to "write" input of FIFO looking at the "CMD" input.

(2) 64 deep FIFO to store {CMD, ADDR, SZ, OP} fields. Logic to connect to "write" input of FIFO looking at the "CMD" input.

(3) 64 deep return data and address FIFOs. These two FIFOs may be combined to reduce complexity. Implement the logic to connect the read input of this FIFO.

(4) The SSTL interface.

(5) DDR2 Initialization State Machine

(6) In Part1, you need to complete the given testbench.

Check Your Result

Your DDR2 is successfully initialized if you can see the following message:

```
*Denali* Class: ddr_II  Instance: "tb.XDDR0" Size: 32768Kx16

*Denali* Class: internal  Instance: "tb.XDDR0(initStatus_registers)" Size: 1x32

*Denali* Class: internal  Instance: "tb.XDDR0(cfg)" Size: 1x32

*Denali* Class: internal  Instance: "tb.XDDR0(mode_registers)" Size: 4x32

*Denali* Class: internal  Instance: "tb.XDDR0(model_status_registers)" Size: 1x3
2

*Denali* Class: internal  Instance: "tb.XDDR0(bank_status_registers)" Size: 16x3
2

ncsim> probe -create -shm tb -all -depth all
Created default SHM database ncsim.shm
Created probe 1
ncsim> # simvision -input ./scripts/ddr2sdram.sv
ncsim> run
*Denali* <tb.XDDR0>@200501 ns :: EMRS2: PASR coverage set to Higher banks
*Denali* <tb.XDDR0>@200501 ns :: EMRS2: PASR num banks set to 4
*Denali* <tb.XDDR0>@200513 ns :: EMRS3: All fields are set to 0
*Denali* <tb.XDDR0>@200525 ns :: EMRS: Enabling DLL
*Denali* <tb.XDDR0>@200525 ns :: EMRS: Disabling DQS#
*Denali* <tb.XDDR0>@200525 ns :: EMRS: Set Additive Latency to 3
*Denali* <tb.XDDR0>@200525 ns :: EMRS: ODT disabled
*Denali* <tb.XDDR0>@200525 ns :: EMRS: Output Driver Impedence Control set to Fu
ll Strength.
*Denali* <tb.XDDR0>@200537 ns :: MRS: Set CAS Latency to 4
*Denali* <tb.XDDR0>@200537 ns :: MRS: Set Burst Length to 4
*Denali* <tb.XDDR0>@202177 ns :: MRS: Set CAS Latency to 4
*Denali* <tb.XDDR0>@202177 ns :: MRS: Set Burst Length to 4
*Denali* Detected[tb.XDDR0] INITIALIZATION_COMPLETE @202177 ns :: Initialization
 completed successfully!
*Denali* <tb.XDDR0>@202993 ns :: EMRS: Disabling DQS#
*Denali* <tb.XDDR0>@202993 ns :: EMRS: Set Additive Latency to 3
*Denali* <tb.XDDR0>@202993 ns :: EMRS: ODT Rtt set to 75 ohm
*Denali* <tb.XDDR0>@202993 ns :: EMRS: Output Driver Impedence Control set to Fu
ll Strength.
*Denali* <tb.XDDR0>@203005 ns :: EMRS: Disabling DQS#
*Denali* <tb.XDDR0>@203005 ns :: EMRS: Set Additive Latency to 3
*Denali* <tb.XDDR0>@203005 ns :: EMRS: ODT Rtt set to 75 ohm
*Denali* <tb.XDDR0>@203005 ns :: EMRS: Output Driver Impedence Control set to Fu
ll Strength.
Simulation complete via $finish(1) at time 203043 NS + 3
./tb/ddr2_controller_tb.v:61           $finish;
ncsim> exit
ncsim: Memory Usage - 37.8M program + 11.7M data = 49.5M total
ncsim: CPU Usage - 0.7s system + 11.9s user = 12.6s total (16.9s, 74.6% cpu)
```

Note: The burst length of your design should be 8, CL should be 3, AL =1.

Grading Strategy

1) DDR2 controller model (including Initialization Engine and FIFO's) in RTL: 60/100
2) DDR2 controller model (including Initialization Engine and FIFO's) in POST-SYN: 40/100

Submission guidelines

1. For the controller design of project phase1, create two folders, "RTL" and "POST", which contain your RTL design and POST-SYN design respectively.
2. Write a "readme.txt" file which explains all the necessary steps that we need to follow in order to run the simulations. The "readme.txt" file has to be as BRIEF and CLEAR as possible. We will contact you for required explanations. Make your **contact information**, namely your e-mail, as clear as possible.
3. Create a folder, "SYN", which contains **the tcl script** that you use for synthesis, **area report**, **timing report**, and so on.
4. "tar" the required files using the following command (tar_filename should be your full name => FirstName1_LastName1_FirstName2_LastName2)

   aludra > tar cvf tar_filename.tar *.v *.out *.txt
5. Submit the tar file using View/Complete tab.
6. One submission per group. (If you and your partner both submit the design, please inform the TAs.)
7. No report required.