4190.309A: Hardware System Design
Midterm Exam
April 26th, 2018
Professor Jae W. Lee
SOLUTIONS

Student ID #: _____

Name: _____

This is a closed book, closed notes exam.
120 Minutes
14 Pages

Total Score: 200 points

Notes:

- Please turn off all of your electronic devices (phones, tablets, notebooks, netbooks, and so on). A clock is available on the lecture screen.
- Please stay in the classroom until the end of the examination.

- You must not discuss the exam's contents with other students during the exam.
- You must not use any notes on papers, electronic devices, desks, or part of your body.

(This page is intentionally left blank. Feel free to use as you like.)

Part A: Short answers (20 points)

Question 1 (20 points)

Answer the following questions. Write down your answer only; you don't have to justify it.

(1) A mux-based bus generally has a longer propagation delay than a tri-state buffer-based bus when many modules are attached to the bus. (True/False)
False

(2) What is the mechanism used to provide a private virtual address space for each hardware component (or IP block)?
IOMMU

(3) What is the most serious problem of a fixed priority scheme in bus arbitration? Answer in one sentence.
Starvation problem – a lower priority module starves due to requests from higher-priority modules.

(4) In bus arbitration what is it called the *guaranteed minimum level of request service* (per request or master)?
Quality-of-Service (QoS)

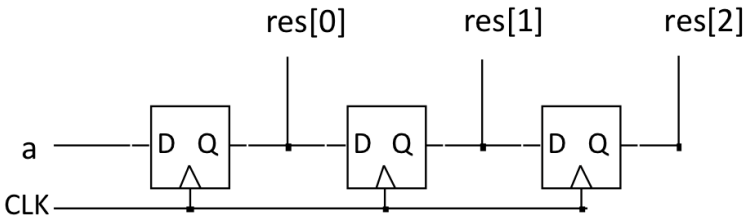
(5) Recently, the performance improvement of CPUs has been slowed down (True/False).
True

Part B: Verilog Basics (45 points)

Question 2 (25 points)

Alarak is a hardware engineer. He wrote the following Verilog code to implement a 3-bit shift register shown below:

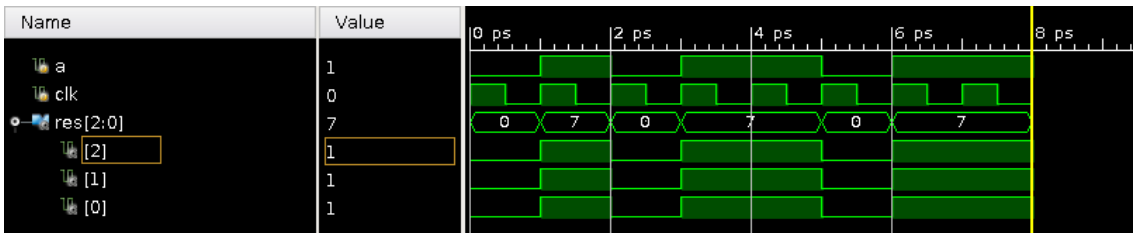
```
`timescale 1us/100ns
module shifter(
    input clk,
    input a,
    output reg [2:0] res
    );

    always @ (posedge clk)
    begin
        res[0] = a;
        res[1] = res[0];
        res[2] = res[1];
    end
endmodule
```



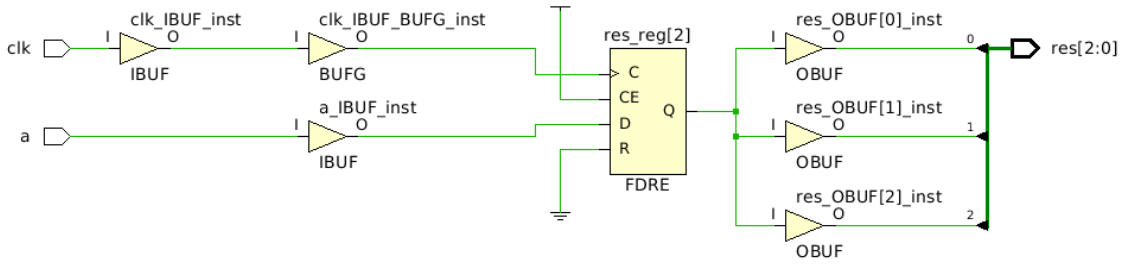
(1) He run simulation with the following testbench. Draw simulated waveforms on the following page. (`$finish` means the end of the simulation)

```
`timescale 1us/100ns
module block_test();
    reg a;
    reg clk;
    wire [2:0] res;

    initial begin
        clk = 1'b1;
        a = 1'b0;
        #1 a = 1'b1;
        #1 a = 1'b0;
        #1 a = 1'b1;
        #1 a = 1'b1;
        #1 a = 1'b0;
        #1 a = 1'b1;

        #2 $finish;
    end

    always #0.5 clk = ~clk;

    shifter shift(
    .clk(clk),
    .a(a),
    .res(res)
    );

endmodule
```

Answer:



Waveform – Also, allows 1 cycle delayed res[2:0] signals



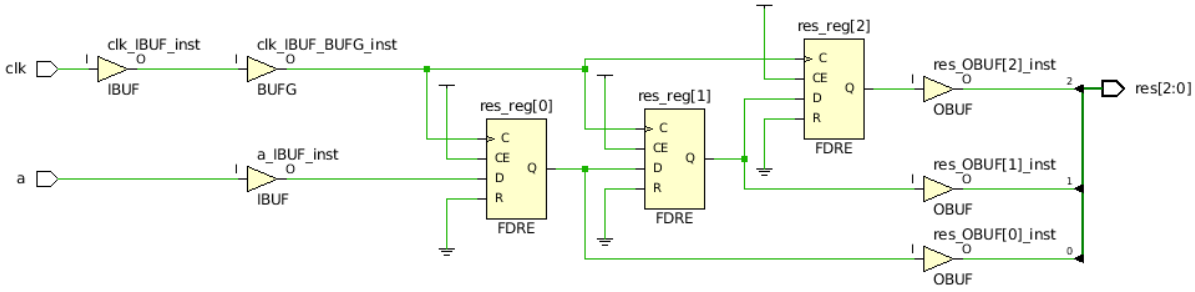
Schematic after synthesis

(2) What is the problem with his design? Point it out and fix his Verilog code. Note that you cannot use delay control to fix it.

Answer:

```
`timescale 1us/100ns
module shifter(
    input clk,
    input a,
    output reg [2:0] res
    );

    always @ (posedge clk)
    begin
        res[0] <= a;
        res[1] <= res[0];
        res[2] <= res[1];
    end
endmodule
```

```
`timescale 1us/100ns
module shifter(
    input clk,
    input a,
    output reg [2:0] res
    );

    always @ (posedge clk)
    begin
        res[2] = res[1];
        res[1] = res[0];
        res[0] = a;
    end
endmodule
```



Schematic after synthesis

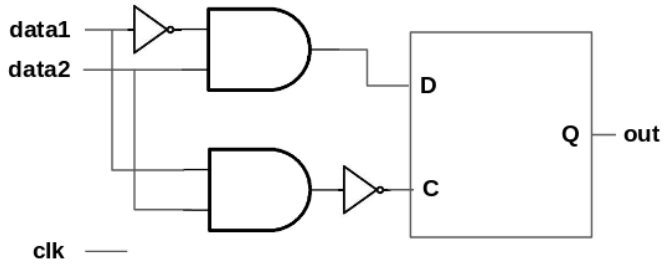
Question 3 (20 points)

Draw a synthesized circuit for each of the two modules. You can use any combinational/sequential logic blocks we covered in the lectures.

(1)

```
module module1(
    input clk,
    input data1,
    input data2,
    output reg out
    );
    always @(*)
      case({data1, data2})
        2'b10: out = 0;
        2'b01: out = 1;
        2'b00: out = 0;
      endcase
endmodule
```

Answer:

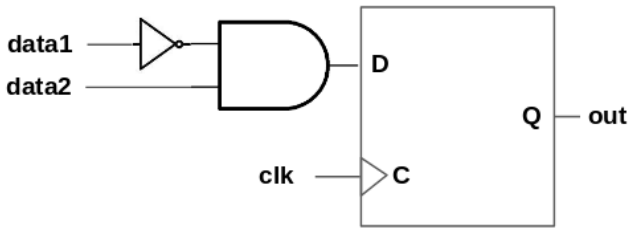


(Note the latch.)

(2)

```
module module2(
    input clk,
    input data1,
    input data2,
    output reg out
    );
    always @(posedge clk)
      casex({data1, data2})
        2'b1x: out = 0;
        2'b01: out = 1;
        2'b00: out = 0;
      endcase
endmodule
```

Answer:



(Note the flip-flop.)

Part C: Verilog Simulation (30 points)

Question 4 (30 points)

xilinx_fmac is the floating point MAC IP used in Lab 2-4. It takes 16 cycles to produce a single multiply-and-accumulate result. At time step zero, *rst* was zero and set to one after two cycles. What will be the TCL console output (i.e., number of cycles) when the simulation ends?

```
module MAC_without_accmulator #(parameter DATA_BW = 32)
(input clk, input rst);

	reg v_in;
	wire v_out;
	wire [DATA_BW – 1 : 0] d_out;
	reg [DATA_BW – 1 : 0] sum, constant;
	reg [15 : 0] accumulation_counter, cycles;

	always @(posedge clk or negedge rst) begin
		if(rst == 0)
			// Initialize all register values to zero.
		else begin
			sum <= sum + (v_out * d_out);
			accumulation_counter <= accumulation_counter + v_out;
			cycles <= cycles + 1;
			v_in <= cycles[0];

			if(accumulation_counter[4] == 1) begin
				$display(“Number of cycles : %d”, cycles);
				$finish;
			end
		end
	end

	xilinx_fmac fpu(
	 .aclk(clk), .aresetn(rst),
	 .s_axis_a_tvalid(v_in), .s_axis_a_tdata(cycles),
	 .s_axis_b_tvalid(v_in), .s_axis_b_tdata(cycles),
	 .s_axis_c_tvalid(v_in), .s_axis_c_tdata(constant),
	 .m_axis_result_tvalid(v_out), .m_axis_result_tdata(d_out));
endmodule
```

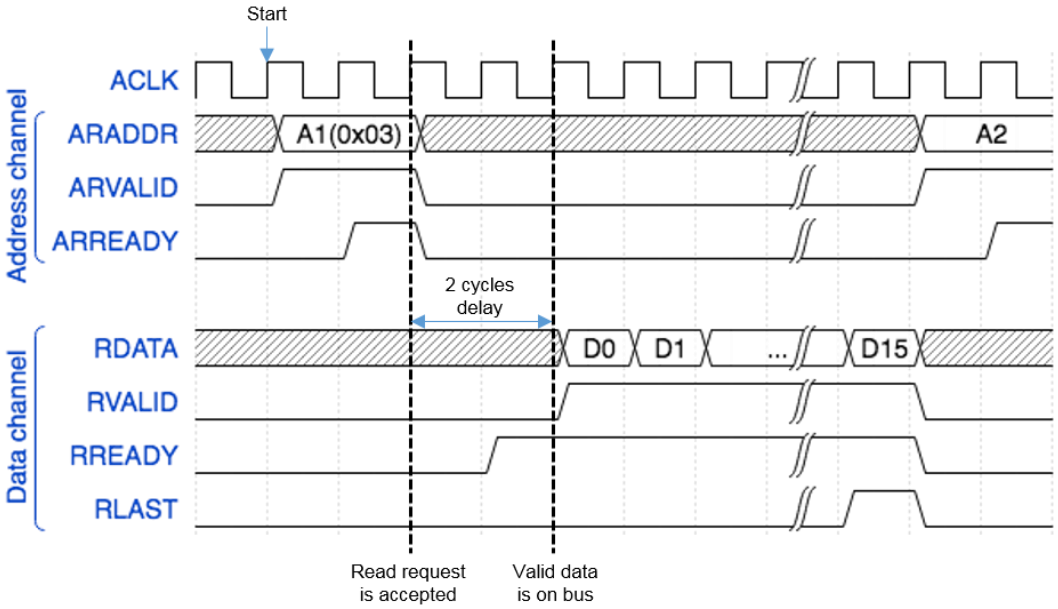
Answer : 49

(Full credit for 49 – 1 cycle is delayed due to accumulation_counter gets assigned with a non-blocking assignment (<=) – Partial credits given to 48. Otherwise, zero)

Part D: AMBA AXI Protocol (30 points)

Question 5 (30 points)

SNU Computers Inc. (SCI) has decided to build a deep learning hardware accelerator. It integrates an AXI4-compatible bus architecture with *non-split transactions* (i.e., a second request cannot start until the first request is completed). The timing diagram of a single memory transaction between the accelerator (AXI-Master) and a memory module (AXI-Slave), called XRAM, is shown below. Answer the following questions.



An example timing behavior of burst read with length=16 and size=1B.
Total cycles: 2 (Address) + 2 (XRAM delay) + 16 (16 consecutive data)

(1) Assuming burst size is 1 byte and burst length is 16 (as shown in the figure), what will be the address of D15 if the burst mode is INCR? What if the burst mode is FIXED?
FIXED: 0x03, INCR: 0x12

(2) The accelerator should read 512 *float* elements (say, reading all data in a "float A[512]" array) from XRAM. Assuming the max burst size if 8 bytes and max burst length is 16, what is the *minimum* number of cycles it takes to read the entire array?
4*512B / (16*8B) * (16+2+2) = 320 cycles

(3) Answer the same question in (2) assuming the AXI bus supports *split transactions*.
4*512B / (16*8B) * (16) + 2 + 2 = 260 cycles

Part E: Design with Zynq (45 points)

Question 6 (25 points)

Ben Bitdiddle is designing a system-on-chip (SoC) with 5 slave IPs whose address mapping is as shown below (Figure (a)). The CPU tries to communicate those IPs using memory-mapped I/O. Figure (b) shows a snapshot of the TLB and page table. Assuming 32-bit address space with 4KB pages, answer the following questions on the next page.

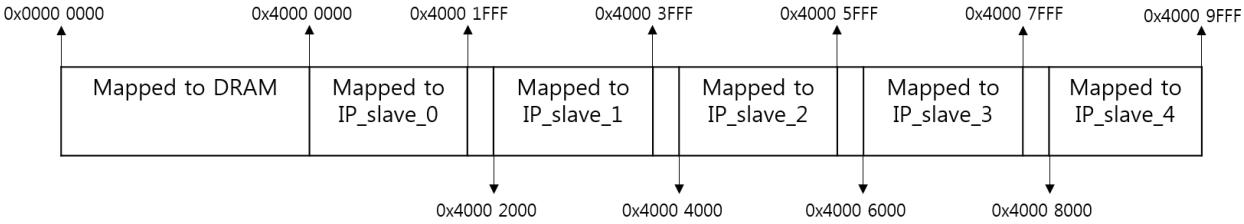


Figure (a) Address mapping table

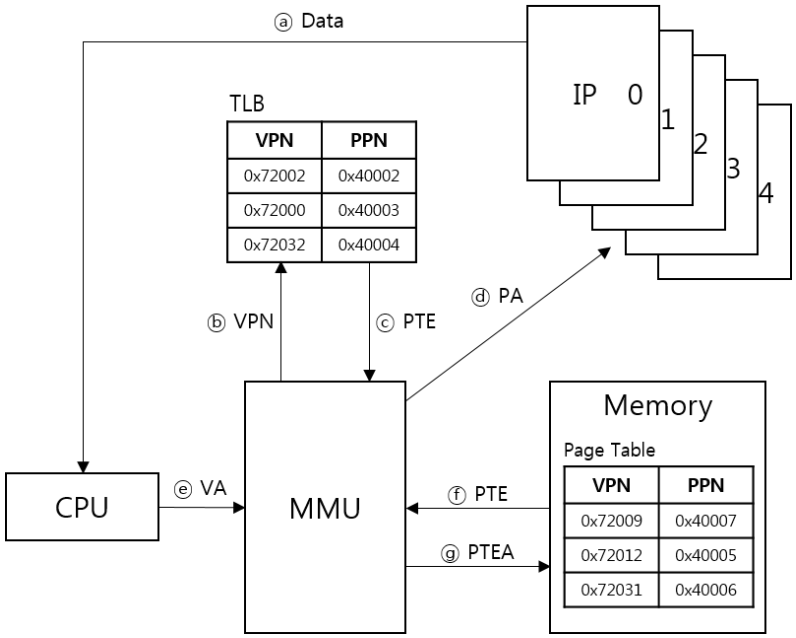


Figure (b) TLB and page table snapshot

(1) Let's assume that integer pointer *my_addr contains 0x7200 9032 (virtual address). What is the physical address of my_addr?

0x4000 7032 (20 MSBs are used for VPN, and 12 LSBs for page offset)

(2) Which slave IP is mapped to *my_addr?

IP_slave_3

(3) What is the address within the slave IP address space being accessed by my_addr?

0x4000 7032 – 0x4000 6000 = 0x1032

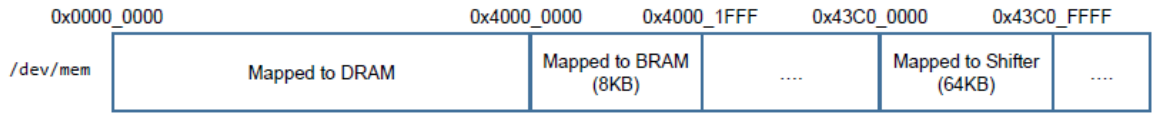
(4) Write down the sequence of the access path for my_addr. We completed the first one for you.

e →b→g→f(→c)→d→a

Note: C may or may not be included. Both get full credits.

Question 7 (20 points)

We use mmap() to access our custom IPs on the Zedboard. Assuming the following memory map, fill in the four blanks in the code.

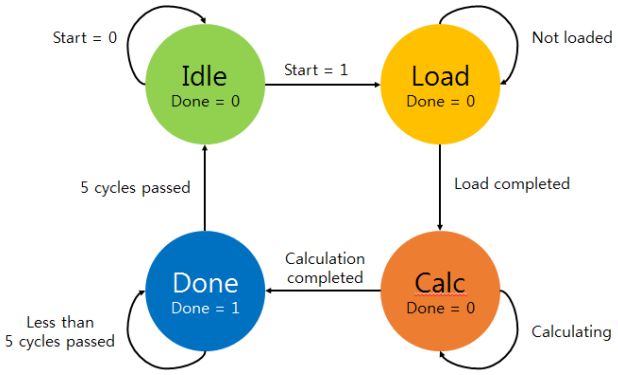



```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys/mman.h>
#define SIZE 4

int main(int argc, char** argv)
{
  int i;
  int foo = open("/dev/mem", O_RDWR | O_NONBLOCK);
  int *fpga_bram = mmap(NULL, (1)__2*SIZE*sizeof(int)__, PROT_WRITE,
MAP_SHARED, foo, (2)__0x40000000__);
  int *fpga_ip   = mmap(NULL, (3)___sizeof(int)___, PROT_WRITE,
MAP_SHARED, foo, (4)__0x43C00000__);

  // initialize memory
  for (i = 0; i < SIZE; i++)
    *(fpga_bram + i) = (float) (i * 2);
  for (i = SIZE; i < SIZE * 2; i++)
    *(fpga_bram + i) = 0.0f;

  printf("%-10s%-10s\n", "addr", "FPGA(hex)");
  for (i = 0; i < SIZE * 2; i++)
    printf("%-10d%-10X\n", i, *(fpga_bram + i));

  // run ip
  *(fpga_ip) = 0x5555;
  while (*fpga_ip == 0x5555);

  printf("%-10s%-10s\n", "addr", "FPGA(hex)");
  for (i = 0; i < SIZE * 2; i++)
    printf("%-10d%-10X\n", i, *(fpga_bram + i));

  return 0;
}
```

Part F: Finite State Machine (30 points)

Question 8 (30 points)

Alice Hacker is designing an FSM that calculates a dot product of two vectors using a PE. The PE contains a BRAM and a floating point MAC IP, and the PE controller has the following states.



IDLE: Wait until the start signal is on.
LOAD: Load two vectors into the controller. One vector is given to PE and saved at the BRAM of it. The other vector is saved at another BRAM of PE controller.
CALC: Calculate the dot product with sequential MAC operations. On each MAC, the corresponding elements of two vectors are multiplied and accumulated to the previous MAC result.
DONE: Display the calculated result for 5 cycles.

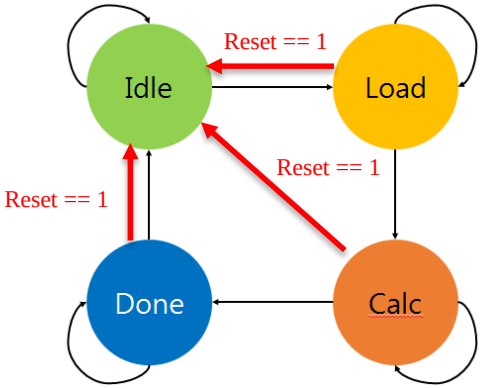
(1) Her implementation of DONE state is as follows. Assume the counter is initialized to 0 before getting into DONE state. What should be the value of N to be consistent with the FSM design?
Answer: 4

```
// determine and update state
    always @(posedge clk or negedge reset)
        present_state <= reset ? next_state : IDLE;

    always @(*)
        case(present_state)
            IDLE : next_state = start? LOAD : IDLE;
            LOAD : next_state = (addr == 32)? CALC : LOAD;
            CALC : next_state = (dvalid && addr == 15)? DONE : CALC;
            DONE : next_state = (counter == N)? IDLE : DONE;
        endcase

...
case (present_state)
...
    DONE :
        begin
            if (counter < N)
                counter <= counter + 1;
            else
                done_reg <= 0;
         end
endcase
```

(2) Alice has decided to extend the functionality of the original FSM. For each of the following three extensions draw new transition(s) if necessary by putting extra arrows and briefly describing conditions when those arrows are taken. You don't have to worry about changes in the datapath.

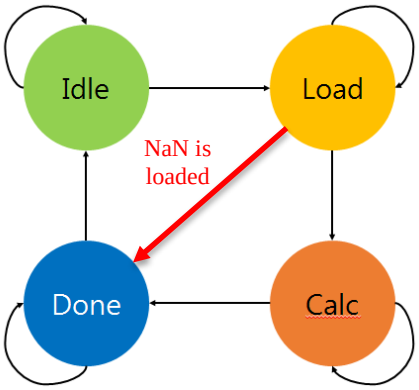
(a) Whenever the reset signal is on, reset the PE controller and wait for a new start signal. (Reset is triggered by rst inputs at posedge clk)

Answer:



(b) If a newly loaded element in either vector is NaN, stop loading immediately and display NaN as a result.

Answer:



(c) Suppose each vector has 32 elements but the BRAM can hold at most 16 elements of both vectors at a time. Alice proposes the following steps to calculate a 32-element dot product.

1. Calculate the dot product for the first 16 elements.
2. Accumulate the MAC results of the next 16 elements onto the output of Step 1.
3. Display the output of Step 2 as final result.

Answer: