EECS 373 Winter 2015 Homework #1a

Due Tuesday Jan 13th (in lecture). 50 points.

Name: _______________________________ uniqname: ________________

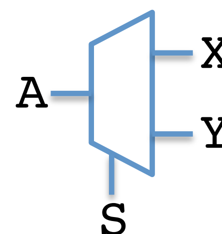
You are to turn in this sheet as a cover page for your assignment. The rest of the assignment should be stapled to this page. This is an individual assignment; all of the work should be your own. **Assignments that are unstapled, lack a cover sheet, or are difficult to read will lose *at least* 50% of the possible points and we may not grade them at all.**

The purpose of this assignment is to remind you about basic logic and Verilog—it's a review of EECS 270.

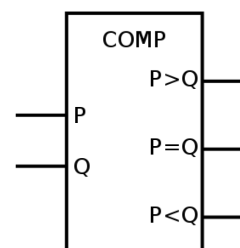
Review of combinational logic

Combinational logic is a type of logic where the output is purely determined by the current input. This is in contrast to sequential logic where the output also depends on previous inputs. In this document we will generally use the asterisk * as AND, the plus + as OR, a single tick ' as negation and a circled-plus ⊕ as XOR.

Q1. Consider the logic statement `((AB)+(A'C))'`
   a. Write the truth table for that logic statement
   b. Draw the gates that implement that logic statement (without simplification).



Q2. Consider a 1-2 DEMUX as drawn on the right.
   a. Write a truth table for this circuit.
   b. Using standard 2-input gates (AND, OR, XOR) and the NOT gate, draw a circuit which implements the same function (has the same truth table).



Q3. Consider a comparator as drawn on the right.
   a. Write a truth table for this circuit.
   b. Using standard 2-input gates (AND, OR, XOR) and the NOT gate, draw a circuit which implements the same function (has the same truth table).

Answer the following questions about implementing combinational logic in Verilog. You will need to look at the combinational logic tutorial found with this homework assignment on the website.

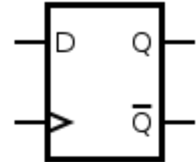
Q4. Provide a single-line assign statement which implements the following
   a. `X=(A'*B)+(A'*C)'+(B*C')`
   b. A 1-to-2 DEMUX. Use `A` and `S` as the inputs and `X` and `Y` as the outputs.

Q5. Using the adder in the tutorial as a template, implement
   a. A **module** implementing a 1-to-2 DEMUX called DEMUX21 which takes two inputs (A,S) and generates two outputs (X,Y).
   b. A module implementing a 1-to-4 DEMUX called DEMUX41 which takes 3 bits of input (A, S[1:0]) and generates four bits of output (Z[3:0]). It should do all its work by instantiating the 2-to-1 DEMUX of part a. as needed.

In addition to assign statements, combinational logic can be implemented in an "`always @*`" block.

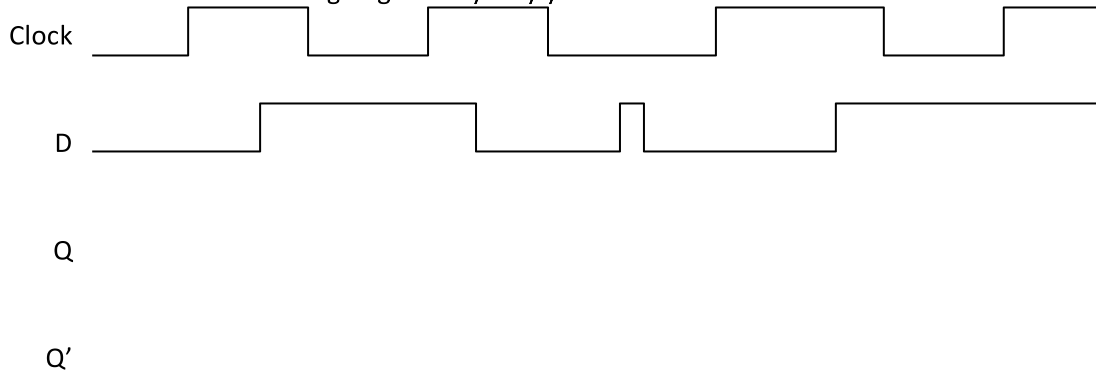
Q6. Redo problem (Q4.a) using an always @* block rather than an assign statement.

Review of Sequential logic



Sequential logic revolves around storage elements. There are a number of different types of storage elements, but for the most part digital designers use only one: the D flip-flop. It's a device that has two inputs "D" and "clock" and one output "Q". The value of D is copied to Q when clock rises (transitions from a 0 to a 1). Further the value of D cannot change too soon before or after that rising edge of clock. Finally it is common to have the value "Q" ***and*** its inverse (Q') as outputs. It is typically drawn as shown on the right (the triangle being the clock input). Nearly every storage unit you will encounter is built out of D flip-flops.

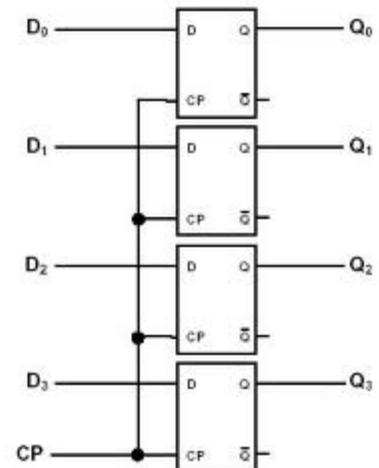
Q7. *Neatly* complete the following timing diagram for a D flip-flop. Indicate that the value is unknown before the first rising edge in any way you see fit.



It is commonly the case that we wish to store larger values than just a single bit. As such we generally use flip-flops in parallel and call them registers. The following diagram shows a typical 4-bit register (though it uses a slightly different symbol for a D flip-flop).

A counter is a device that that counts how many times a certain event has occurred. See http://en.wikipedia.org/wiki/Counter.

Q8. Using full adders and D flip-flops as building blocks, draw a 2-bit modulo counter. That is, on each rising edge the counter's output value is incremented by 1, but where 3 wraps around to 0 (so it is a modulo-4 counter). You do not need to worry about reset.



4-bit register, from http://cpuville.com/register.htm

Now look at the Verilog sequential logic overview on the website.

Q9. Answer the following questions about the supplied overview

a. How many bits are used to store the state? How many are the least that could have been used?
b. Section 3.2 has two implementations of the next state logic. What are the pros and cons of each?

Q10. Draw the state transition diagram which is implemented by the code on the next page. Your diagram should resemble figure 2a from the tutorial. You need not include reset.

Q11. Define the term "setup time" and the term "hold time" with respect to a D flip-flop. Your answer should include a (simple) timing diagram which illustrates your point.

For the following question, you may need to use Wikipedia or some other source. It's likely you didn't learn this material in EECS 270.

Q12. Say you have a group of devices, each of which may wish to communicate with each other. One way to do this is to have a shared wire (or group of wires called a bus) that each device could drive when needed. Say we have 5 devices (called "A", "B", etc.) connected to this wire.

a. One option we'd have is to use tri-state drivers.
   i. What is a tri-state device? What are the "three states" that can be driven by a device and what exactly is the "HiZ" state?
   ii. Say device "A" wanted to send a "1" on the wire. What would device A need to drive? What would the other devices need to drive?
   iii. What would happen if two devices tried to drive the same wire at the same time?
   iv. If you drive "HiZ" into an inverter, what will be the output of the inverter?
b. Another option we've got is to use an open-collector scheme.
   i. What does it mean to use an open collector? What role does the "pull-up" resistor play?
   ii. Say device "A" wanted to send a "1" on the wire. What would device A need to drive? What would the other devices be driving when they weren't trying to use the bus?
   iii. Say device "A" wanted to send a "0" on the wire. What would device A need to drive? What would the other devices be driving when they weren't trying to use the bus?
   iv. You know that the data on your bus is 70% 0's and 30% 1's. How might you modify the open-collector scheme to be more energy efficient?
   v. What two factors determine the maximum line speed (how fast you can alternate between "0" and "1") for an open-collector bus?

```
module statem(clk, in, reset, out);

input clk, in, reset;
output [3:0] out;

reg [3:0] out;
reg [1:0] state;
reg [1:0] next_state;

parameter zero=2'd0, one=2'd1, two=2'd2, three=2'd3;

always @*
     begin
          case (state)
               zero:
                    begin
                         out = 4'b0000;
                         next_state = one;
                    end
               one:
                    begin
                         out = 4'b0001;
                         if(in)
                              next_state = one;
                         else
                              next_state = two;
                    end
               two:
                    begin
                         out = 4'b0010;
                         if(in)
                              next_state = one;
                         else
                              next_state = three;
                    end
               three:
                    begin
                         out = 4'b0100;
                         next_state = zero;
                    end
               default:
                    begin
                         out = 4'b0000;
                         next_state = zero;
                    end
          endcase
     end

always @(posedge clk)
     begin
          if (reset)
               state <= zero;
          else
               state <= next_state;
     end

endmodule
```