EECS 373 Fall 2017 (Review) Homework 1

Robert Dick

Due 13 September before lecture.

Name:

Unique name:

Submit the assignment via Gradescope.

Use the reference material in the References section of the course website, and review your EECS 270 notes.

You may need to search on the web or in the library for Question 11. Feel free to stop by my office or post on Piazza!

1 Combinational Logic Review

1. **(3 pts.)** Consider the logic statement b'c' + bc' + ab'c.
   (a) Show the truth table.
   (b) Show a minimal two-level expression.
   (c) Show the minimal two-level gate-level schematic.
2. **(3 pts.)** Consider a 2:1 MUX with inputs i_0, i_1, s and output p.
   (a) Show the truth table.
   (b) Using only the following gate types, show a gate-level schematic implementing the MUX: NOT, AND2, OR2, and XOR2.
3. **(3 pts.)** Consider a full adder with inputs ci, a, and b and outputs co and s.
   (a) Show the truth table.
   (b) Using only the following gate types, show a gate-level schematic implementing the full adder: NOT, NAND2, NOR2, and XNOR2.

2 Verilog for Combinational Logic

4. **(3 pts.)** Consider the following function. You may refer to the Verilog tutorial in the References section of the website.

$$p = a'b + (ac')' + b \oplus c \quad (1)$$

   (a) Show an assign statement implementing the function.

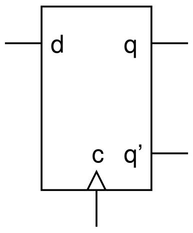

Figure 1: D flip-flop.

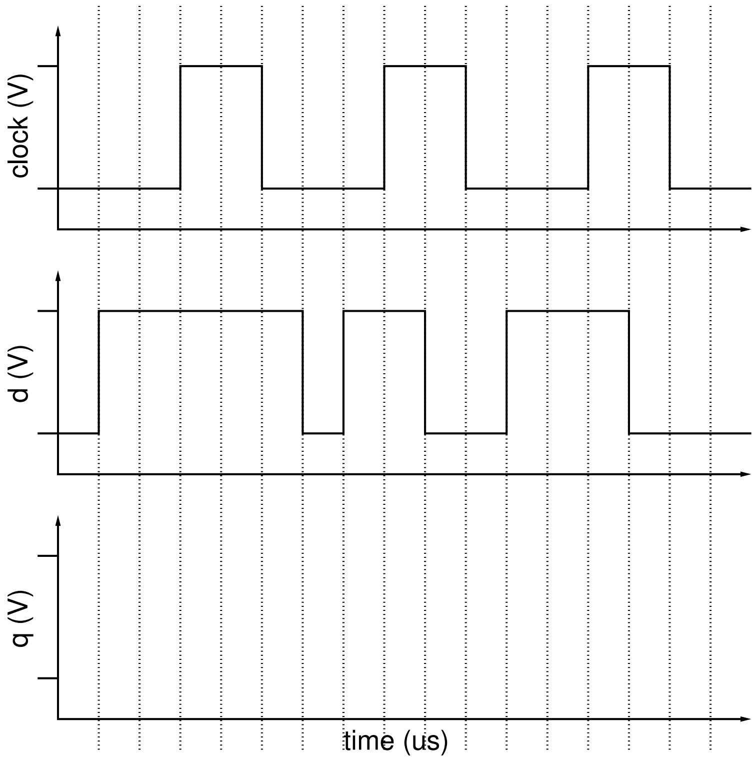

Figure 2: Incomplete D flip-flop timing diagram.

(b) Show an assign statement implementing a 2:1 MUX, using the IO names in Question 2.

5. **(3 pts.)** Treat the adder in the tutorial as a template.
   (a) Implement a module for the 2:1 MUX called MUX21.
   (b) Implement a module for a 4-to-1 MUX called MUX41 that has inputs i_0, i_1, i_2, i_3, s_0, and s_1 and output p. It should reuse/instantiate the 2:1 MUX module you just implemented.
6. **(3 pts.)** Redo Question 5 using an always @* block instead of an assign statement.

3 Sequential Logic Review

7. **(3 pts.)** Consider the rising edge triggered D flip-flop shown in Figure 1. At, and only at, rising clock edges, this sequential element transfers the input (d) value to its output (q). At other times, q does not change. An inverted output q' is also commonly provided. Due to internal delays resulting from

parasitic RC networks within the D flip-flop, it is important for d to remain stable for a brief period before (setup time) and after (hold time) the rising clock edge. Many other sequential elements and finite state machines use D flip-flops as building blocks. D flip-flops are often used in parallel with independent inputs and outputs (q and d values), but a shared clock line (c) to store multiple bits. Complete the timing diagram in Figure 2. If q is unknown at any time, indicate this by drawing both low-voltage and high-voltage lines with cross-hatching between them.

8. **(3 pts.)** Using full adders and D flip-flops as building blocks, draw a counter that increments its two-bit output on every rising clock edge and wraps around to zero after counting to three. Include a clear input (e), that sets q to 0 if e is 1 on a rising clock edge, regardless of the value of d.

4 Verilog for Sequential Logic

9. **(3 pts.)** Refer to the Verilog tutorial in the References section of the website and, in particular, to the arbiter machine shown in that tutorial.
   (a) How many bits are used to store the arbiter state?
   (b) Given the number of states, how many state variables are actually required?
   (c) What are the advantages and disadvantages to the different implementations of next-state logic shown in Section 3.2 of the tutorial? Keep your answer to a few terse sentences.

10. **(3 pts.)** Draw the state transition diagram implemented by the code in Figure 3. Your diagram should resemble Figure 2a in the tutorial. You need not include reset.

11. **(3 pts.)** Consider five devices, each of which may need to communicate a bit at a time with any other. Instead of using 10 wires to connect all pairs of devices, a bus, or shared interconnect, could be used. Whenever a device needs to communicate, it can "drive" or apply a voltage to the interconnect. Using tri-state drivers to connect device outputs to the bus would be one option.
   (a) What is a tri-state driver?
   (b) List the three states the output of a tri-state driver can have?
   (c) Use 1-2 sentences to describe the Hi-Z state.
   (d) For a device to send a 1 on the bus, what state would its driver need to be in?
   (e) ... what states would the drivers of the other devices on the bus need to have?
   (f) What would happen if a device drove a wire with a 1 and another drove it with a 0 at the same time?
   (g) If the input to an inverter is Hi-Z, what is the output?

12. **(3 pts.)** An open-collector scheme is an alternative to the tri-state drivers used in Question 11.
   (a) What does it mean to use an "open collector"/"open drain" signaling scheme?
   (b) What role does the pull-up resistor play?
   (c) What state does a device output need to be in to send a 0?
   (d) ... a 1?
   (e) ... when it is not attempting to drive the 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 = two;
                        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'b0101;
                                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
```

Figure 3: Example code.