# Exam I

E210/B441, Spring 2021

Name: ________KEY________________________

Email: ________________________________

I have neither **given** nor **received** unauthorized aid on this examination, nor have I **concealed** any knowledge of unauthorized aid by others.

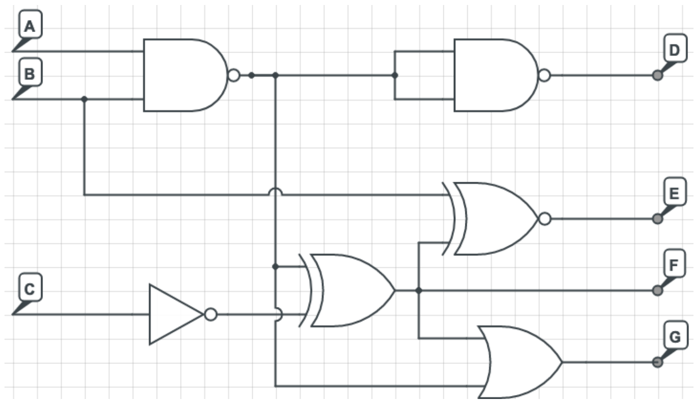
Signature: ________________________________

This page intentionally left blank.

Name: ____________________________ Email: __________________________________

## 1. Boolean Logic Part 1 (10 Points) (Released Early)

Construct the truth table for the following circuit:



| Inputs | | | Outputs | | | |
|---|---|---|---|---|---|---|
| **A** | **B** | **C** | **D** | **E** | **F** | **G** |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 |

Name: ___________________________ Email: ________________________________

## 2. Boolean Logic Part 2 (10 Points)

Write the SystemVerilog module definition that implements the following truth table below. A, B, and C are inputs. X and Y are outputs. Note, this does **NOT** have to utilize *minimized* Boolean logic.

| Inputs | | | Outputs | |
|---|---|---|---|---|
| A | B | C | X | Y |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 |

(5 pts per assign, 1 pts per equation, 1 pt for correct OR)

```
module truth_table (
     input a, b, c,
     output x,y
);

X and Y are reversed for some

assign x = (~a & ~b & ~c) | (~a & b & c) |
           ( a & ~b & ~c) | ( a & b & c);

assign y = (~a &  b & ~c) | (~a &  b & c) |
           ( a & ~b & ~c) | ( a & ~b & c);


-- OR --

assign x = ~ (b ^ c);
assign y =   (a ^ b);



endmodule
```

Name: ___________________________ Email: ________________________________

## 3. Multiplexer & Testbench (20 points)

a. (6 pts) A Multiplexer (or Mux) is a device that selects between several digital inputs and forwards the selected input to a single output line. The selection is directed by a separate set of inputs known as select lines. A multiplexer of $2^n$ inputs has *n* select lines, which are used to select which input line to send to the output. [wiki]

Please implement a 2-input multiplexer in SystemVerilog below. If s==0, the output z should match the input a, otherwise z should match input b.

```
module mux2 (
     input     a,
     input     b,
     input     s,
     output logic z
);

always_comb begin
     z = a; //default
     if (s == 'h1)
          z = b;
end

-- OR --

assign z = (s == 'h0 ? a : b);

-- OR --

assign z = (a & ~s) | (b & s);


endmodule
```

b. (12 pts) Please implement a testbench for the 2-input multiplexer in SystemVerilog. To receive all points, your testbench should test all possible inputs. BUGFIX

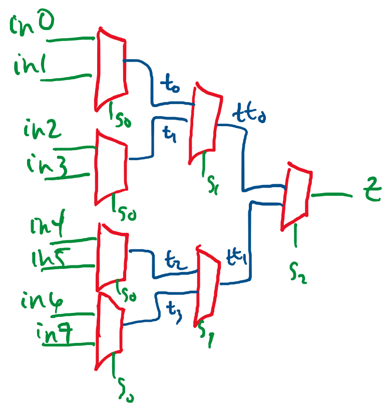
1 pt for each test case (8x), 1 pt for setting a,b,s, 1 pt for assert, 1pt delay, 1 pt for $display/$finish

```
module mux2_tb();

logic a,b;
logic s;
wire  z;

mux2 DUT (.a(a), .b(b), .s(s), .z(z) );

task mux2Test(
     input aT, bT, sT, zT
);

    a = aT; b = bT; s = sT;
    #1
    assert( z == zT ) else
    $fatal(1, "bad:  a:%b b:%b s:%b z:%b, zT:%b", a, b, s,
z, zT);
endtask

initial begin

          mux2Test(0,0,0,0); //1'h0 or 'h0 also work
          mux2Test(0,0,1,0);
          mux2Test(0,1,0,0);
          mux2Test(0,1,1,1);

          mux2Test(1,0,0,1);
          mux2Test(1,0,1,0);
          mux2Test(1,1,0,1);
          mux2Test(1,1,1,1);

          $display("@@@Passed");
          $finish();
end



endmodule
```

## 4. More Multiplexer (10 points)

Please implement an 8-input multiplexer in SystemVerilog below. This time `s` should select the corresponding bit within the input `in`. **Your module *must* utilize `mux2` from above**. You may assume a correct `mux2` implementation.

```
1 pt /level 1 mux, 2 pts /level 2 mux

    module mux8 (
         input    [7:0] in,
         input    [2:0] s,
         output         z
    );

    wire [3:0] t; //temp - level 1
    wire [1:0] tt; //temp - level 2

    mux2 m0 (.a(in[0]), .b(in[1]), .s(s[0]), .z(t[0]) );
    mux2 m1 (.a(in[2]), .b(in[3]), .s(s[0]), .z(t[1]) );
    mux2 m2 (.a(in[4]), .b(in[5]), .s(s[0]), .z(t[2]) );
    mux2 m3 (.a(in[6]), .b(in[7]), .s(s[0]), .z(t[3]) );

    mux2 m4 (.a(t[0]),  .b(t[1]),  .s(s[1]), .z(tt[0]) );
    mux2 m5 (.a(t[2]),  .b(t[3]),  .s(s[1]), .z(tt[1]) );

    mux2 m6 (.a(tt[0]), .b(tt[1]), .s(s[2]), .z(z) );
```

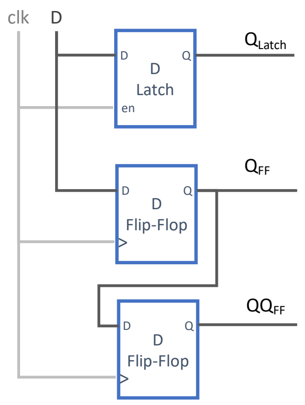


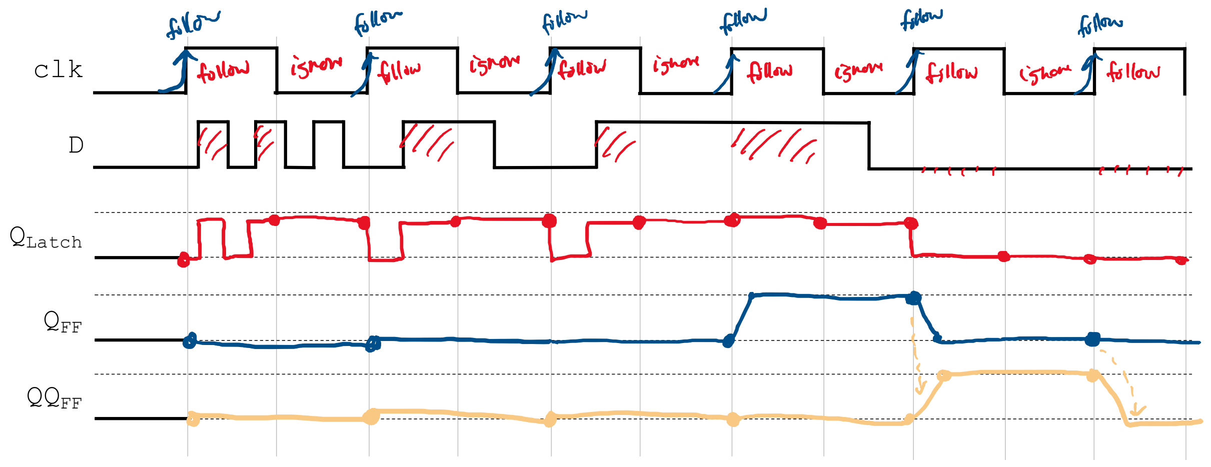
```
    endmodule
```

Name: ___________________________ Email: _________________________________

## 5. Sequential Logic (15 Points)

Given the following circuit, draw the timing diagram of the outputs $Q_{Latch}$, $Q_{FF}$, and $QQ_{FF}$. You may assume all latches and flops initially output 0. Assume a small delay in the output of the latch or flip-flop.





| | | | | | |
|---|---|---|---|---|---|
| Qlatch | 1 pt. | 1 pt. | 1 pt | 1 pt | 1 pt |
| Qff | 1 pt | 1 pt | 1 pt | 2 pt | 1 pt |
| QQff | 1 pt | | | 1pt | 2 pt |

Name: ___________________________ Email: _________________________________

## 6. State Machines (35 Points)

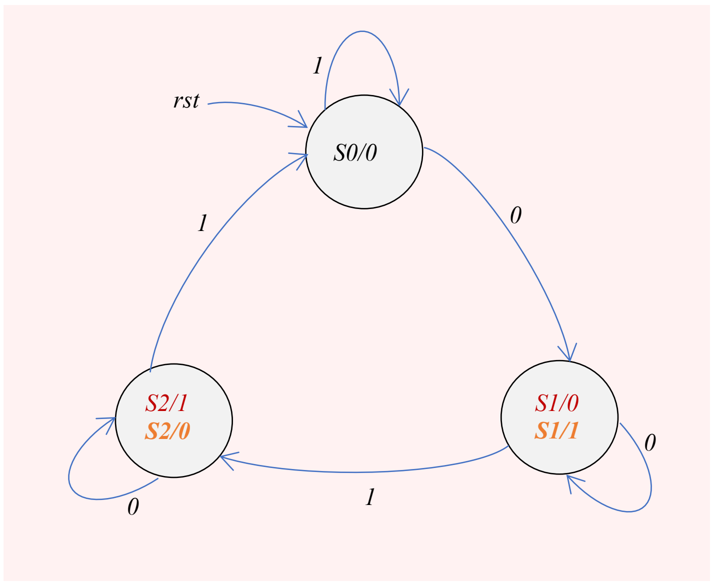
a. (10 pts) A state table or a state transition diagram models the behavior of a clocked sequential circuit. **Draw a state transition diagram** for the state table shown below left. The table below right shows the output in each state. When reset is asserted (`rst == 1`) the state machine enters state `S0`. The input to the state machine is `in`, the output is `out`.

| *State* | *rst* | *in* | *Next state* |
|---|---|---|---|
| *X* | 1 | X | S0 |
| *S0* | 0 | 1 | S0 |
| *S0* | 0 | 0 | S1 |
| *S1* | 0 | 0 | S1 |
| *S1* | 0 | 1 | S2 |
| *S2* | 0 | 0 | S2 |
| *S2* | 0 | 1 | S0 |

| *State* | *out* |
|---|---|
| *S0* | 0 |
| *S1* | **0/1** |
| *S2* | **1/0** |

Type 1
Type 2

1 pt per arrow, 1 pt per state



Name: ___________________________          Email: ________________________________

b. **(25 pts) Write a SystemVerilog module** that implements the state machine described above. Assume that the frequency of the clock input *clk* does not need scaling. Code that results in inferred latches will be considered incorrect.

1pt enum states, 3 pt always_ff, 3pts output/state(9), 2pts/state transition(12)

```
module state_machine (
      input clk,
      input rst,
      input in,
      output logic out
);

enum {S0, S1, S2} state, nextState;

// Transition to next state
always_ff @(posedge clk) begin
  if ( rst )
    state <= S0;
  else
    state <= nextState;
end

// Next state + output function
always_comb begin
  nextState = state;
  case (state)
     S0: begin
         out = 1'b0;
         if (in) nextState = S1;
         else    nextState = S0; //optional
     end

     S1: begin
         out = 1'b0; // 1'b1
         if (in) nextState = S2;
         else    nextState = S1; //optional
     end

     S2: begin
         out = 1'b0; // 1'b0
         if (in) nextState = S0;
         else    nextState = S2; //optional
     end
   endcase
end


endmodule
```