EECS 151/251A Homework 2

Due 11:59pm Friday, September 17th, 2021

1 Avoid unintentional latch synthesis

For each of the following Verilog modules, which ones will generate latches? When submitting this question on Gradescope, if unintentional latches are generated, please select the signal names. If no unintentional latch is generated, select `CORRECT`.

(a)

```
input [1:0] a;
input b, c;
reg x, y;
always@(a or b or c) begin
    case (a)
        2'b00 : x=b;
        2'b01 : x=c;
        2'b11 : y=b & c;
        2'b10 : y=b | c;
    endcase
end
```

(b)

```
input a;
output reg [1:0] y;

always@ (*)
    if (a)
        y = 2'b01;
    else
        y[0] = 1;
```

(c)

```
input [1:0] x;
reg [1:0] y;
always @(*) begin
    y = 2'b00;
    if (x=2'b10)
        y = 2'b01;
    else if (x=2'b11)
        y = 2'b11;
end
```

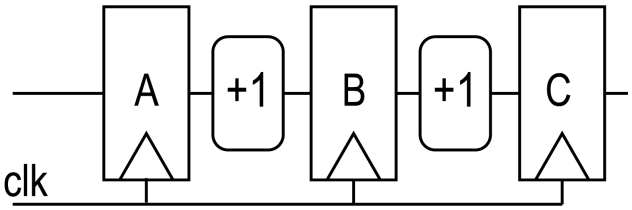
2 Find the errors

For each module, identify the common mistake in the code and put the line numbers in the box. For modules that have errors in more than one line, list the numbers an increasing order. The line number should be separated by ', ' (e.g., `1, 8, 10`). If the module is correct, put `CORRECT` in the box.

(a) The following module is meant to be a simple 2-to-1 multiplexer with the output being set to input a when sel=0, and b when sel=1.

```
1 module mux_2to1 (
2     input sel,
3     input [1:0] a,
4     input [1:0] b,
5     output out
6 );
7     assign out = out = (~sel & a) | (sel & b);
8 endmodule
```

(b) The following code is meant to implement the circuit below:




```
input a_in, b_in, c_in;
input clk;
reg a_out, b_out, c_out;
always @(posedge clk) begin
    a_out = a_in;
    b_out = b_in;
    c_out = c_in;
end
assign b_in = a_out + 1;
assign c_in = b_out + 1;
```

(c) The following module is meant to be a 4-to-1 multiplexer.

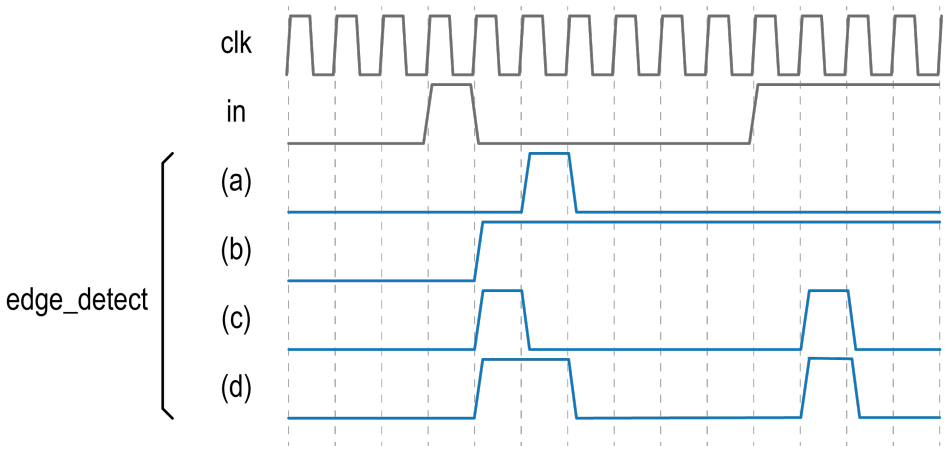
```
module mux_4to1 (
    input a,b,c,d,
    input [1:0] sel,
    output reg out
);
   always @ (a or b or c or d) begin
      case (sel)
         2'b00 : out = a;
         2'b01 : out = b;
         2'b10 : out = c;
         2'b11 : out = d;
      endcase
   end
endmodule
```

3 Edge detector circuit

The code below is a Verilog module that detects the edge of an input signal. The input signal is synchronous to the clock.

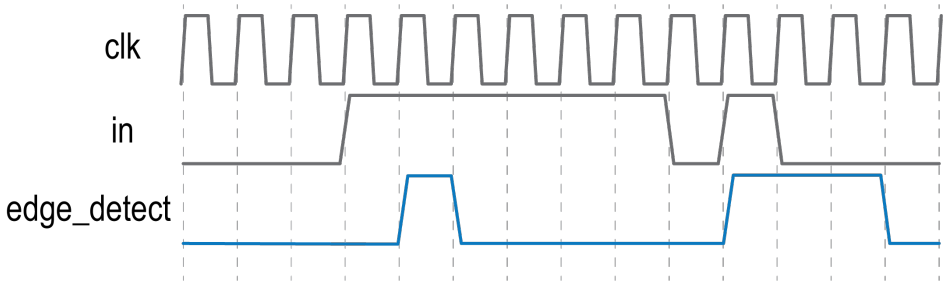
```
module edge_detect (
    input clk,
    input in,
    output edge_detect
);
    reg d;
    always@(posedge clk) begin
        d <= in;
        edge_detect <= (~d)&in;
    end
endmodule
```

(a) Which of the following waveforms is correct? [ ]



(b) To detect both the rising and falling edge of the input signal, please complete the code in the following Verilog module. It should emit a one clock period pulse every time there is a falling or rising edge.

Here is an example waveform:




```
module edge_detect (
    input clk,
    input in,
    output edge_detect
);
    reg d;
    always@(posedge clk) begin
        d <= in;
        edge_detect <= _______;
    end
endmodule
```

4 Flops and shift registers

4.1 JK flip-flop

A JK flip-flop has the truth table shown below. Please fill in the blanks in the Verilog module below.

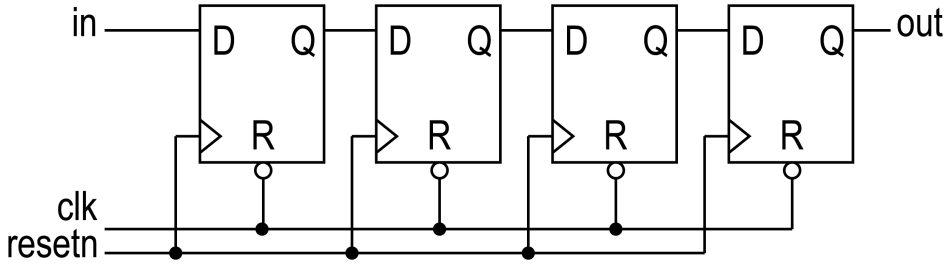
J	K	Q_{new}
0	0	Q_{old}
0	1	0
1	0	1
1	1	$\sim Q_{old}$

```
module jk_ff (
    input j,
    input k,
    input clk,
    output q
);

   always @ (posedge clk)
      case (__1__)
         2'b00 : q <= q;
         2'b01 : q <= 0;
         2'b10 : q <= __2__;
         2'b11 : q <= __3__;
      endcase
endmodule
```

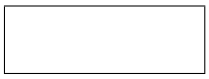
4.2 Shift register with reset

Fill in the blanks in the following Verilog module to complete the 4-bit shift register with synchronous reset.



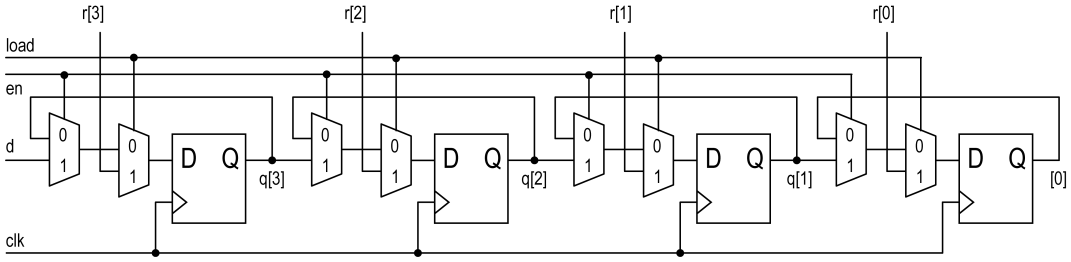

```
module shift_reg (
    input clk,
    input resetn,
    input in,
    output out
);

    reg [2:0] mid;
    always @(__1__)
        if (~resetn)
            {out, mid} <= __2__;
        else
            {out, mid} <= __3__;
endmodule
```



4.3 Shift register with parallel input (251A Only, Optional for 151)

The shift register below can do everything that the shift register in the previous question does, plus load the data to all stages simultaneously. Also, the shift function can be disabled to make each flop keep its output. This 4-bit shift register is implemented using four `mux_dff` modules. Please fill in the blanks in the following code.




```
module mux_dff (
    input load,
    input en,
    input clk,
    input d,
    input r,
    output q
);
    always @(posedge clk)
    if (___1___)
        q <= {___2___};
    else if (___3___)
        q <= {___4___};
    else
        q <= q;
endmodule
```

[ ]

[ ]

[ ]

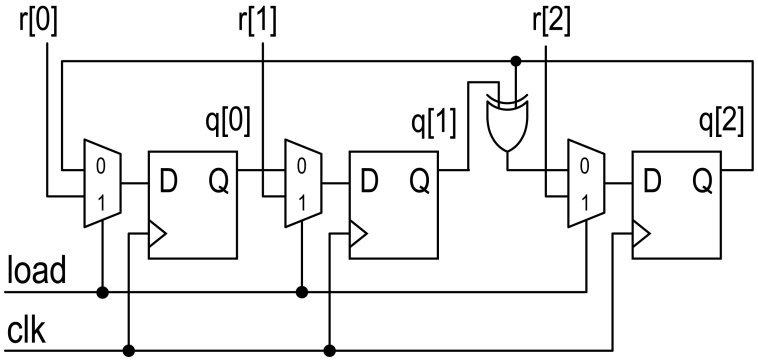
[ ]

```
module shift_reg (
    input clk,
    input load,
    input en,
    input d,
    input [3:0] r,
    output [3:0] q
);
    mux_dff mux0(load, en, clk, __5__, r[3], q[3]);
    mux_dff mux1(load, en, clk, __6__, r[2], q[2]);
    mux_dff mux2(load, en, clk, q[2], r[1], q[1]);
    mux_dff mux3(load, en, clk, q[1], r[0], q[0]);

endmodule
```

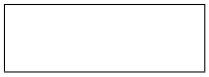
5 LFSR

A linear feedback shift register (LFSR) is a system that generates bits from a register and a feedback function. After several iterations, the register returns to a previously known state and starts again in a loop. The number of iterations is called its period. The following circuit describes a 3-bit LFSR that generates a pseudo-random binary sequence (PRBS).



(a) Fill in the blanks in the following Verilog module.

```
module lfsr(
    input [2:0] r,
    input clk,
    input load,
    output reg [2:0] q
);
    always @ (posedge clk)
    if (load)
        q <= r;
    else
        q <= {______, ______, ______};
endmodule
```



(b) If `110` is loaded into the LFSR initially, What is the repetition period of this PRBS generator? What is the The repeating sequence of numbers generated? List N binary numbers it generates starting from 110, where N is the period of this generator. Binary numbers are separated by '`, `' For example, if its period=3, list first 3 numbers it generates: `110, 111, 000`.

The period is:

The repeating sequence is: