

## EECS 151/251A Homework 2

Due Friday, February 2<sup>th</sup>, 2018

### Problem 1: Verilog

- A) The following code describes a 3-bit linear-feedback shift register (LFSR), which generates a repeating pattern of pseudo-random numbers.

```
module lfsr(  
    input [2:0] R,  
    input Load,  
    input Clock,  
    output reg [2:0] Q  
);  
  
    always@ (posedge Clock)  
        if (Load)  
            Q <= R;  
        else  
            Q <= {Q[1], Q[0] ^ Q[2], Q[2]};  
endmodule
```

- (a) Draw the circuit that corresponds to the code, using combinational logic blocks (gates etc.) and state elements.
- (b) If 100 is loaded into the LFSR initially, what is the sequence of numbers generated? List the binary numbers it generates, start from 001.
- B) The code below is similar to the code in the previous, but blocking assignments are used. The example below should illustrate why blocking assignments for synchronous logic should be avoided if possible.

```
module lfsr(R, Load, Clock, Q) ;  
    input [2:0] R;  
    input Load, Clock;  
    output reg [2:0] Q;  
  
    always@ (posedge Clock)  
        if (Load)  
            Q <= R;  
        else begin  
            Q[0] = Q[2];  
            Q[1] = Q[0] ^ Q[2] ;  
            Q[2] = Q[1];  
        end
```

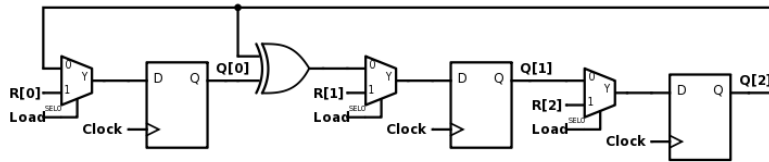
```

end
endmodule

```

- (a) Draw the circuit that corresponds to the code.
- (b) If 001 is loaded initially, what is the sequence of numbers generated?

A) (a) -

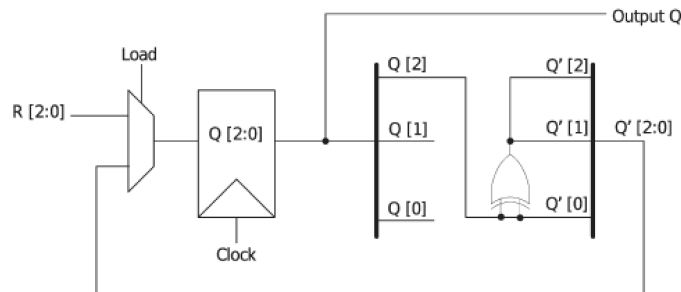


(b) -

Cycle	Q[2:0]
0	100
1	011
2	110
3	111
4	101
5	001
6	010
7	100

The question confusingly asked to list the pattern from 001, so from this sequence: {001, 010, 100, 011, 110, 111, 101} then back to 001.

B) (a) -



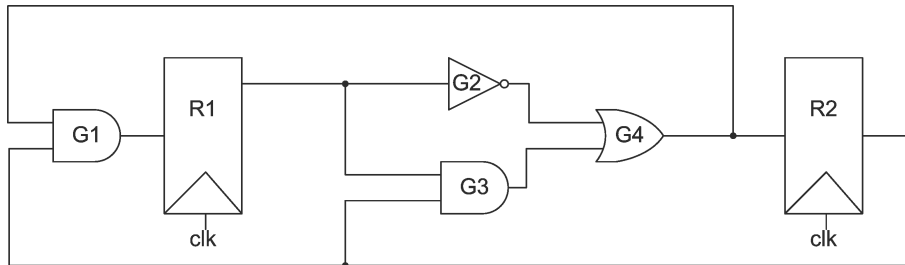
(b) - The generator always outputs 000 after the rst cycle.

Cycle	Q[2:0]
0	001
1	000
2	000
⋮	⋮

## Problem 2: Timing

The timing parameters for the circuit below are as follows:

Setup time	50 ps
Hold time	45 ps
Clock-to-q delay	40 ps (min), 60 ps (max)
Propagation delay per gate	80 ps (min), 100 ps (max)



- A) Identify the critical path (i.e. the longest timing path) or paths in this circuit. You may make use of the register names (R1 and R2) and the gate names (G1, G2, G3, G4) to help you describe the critical path(s).
- B) Use your answer in part (a) to calculate the maximum clock rate at which this circuit can be reliably run.
- C) Can the register hold times ever be violated in this circuit? Why?

A) There are three critical paths.

Path 1: from output of R1, through G2, G4, G1, to input of R1.

Path 2: from output of R1, through G3, G4, G1, to input of R1.

Path 3: from output of R2, through G3, G4, G1, to input of R1.

B)  $T_{clk-q} + T_{d,G2} + T_{d,G4} + T_{d,G1} + T_{su} \leq T_{clk} \Rightarrow T_{clk} \geq 410ps$  so  $F_{clk,max} \approx 2.44GHz$

C) No. The shortest path in the circuit is R2 through G1 to R1. Minimum clk-to-q + minimum gate delay = 120ps. This is larger than the hold time of 45ps, hence hold time will not be violated.

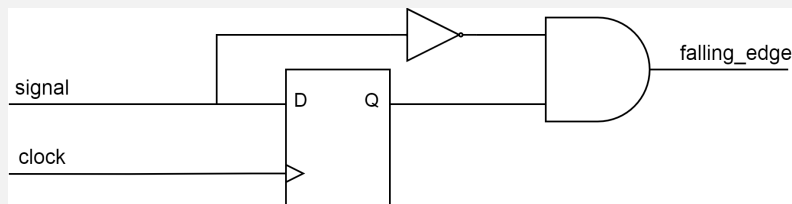
### Problem 3: Edge Detector Circuit

Design a Verilog module to detect the falling edge of an input signal. The input signal is synchronous to the clock, and when a falling edge is seen, a 1 clock cycle pulse should be emitted as an output signal.

```
module edge_detector (input signal, input clk, output falling_edge);
    /* Your code here */
endmodule
```

```
module edge_detector (input signal, input clk, output falling_edge);
    reg last_signal;
    always @ (posedge clk)
        last_signal <= signal;
    assign falling_edge = last_signal && ~signal;
endmodule
```

Draw your Verilog implementation of this circuit as a schematic of gates and flip-flops.



### Problem 4: Build a Multiplier Using an Accumulator

Construct a multiplier by accumulating the results of an adder. Here is a Verilog template to complete.

```
module multiplier (
    input clk, input start, output done,
    input [15:0] arg1, input [15:0] arg2,
    output [31:0] product
);
    reg [15:0] counter;
    reg [31:0] accumulator;
    /* Your code here */
endmodule
```

You can assume that the `start` input is pulsed for one clock cycle and after it is pulsed the `arg1` and `arg2` inputs are held stable until the `done` output goes high. Also assume that the logic driving

the multiplier is well behaved (it doesn't toggle start until the multiplier is done with the last computation).

```
module multiplier (  
    input clk, input start, output done,  
    input [15:0] arg1, input [15:0] arg2,  
    output [31:0] product  
);  
reg [15:0] counter;  
reg [31:0] accumulator;  
always @ (posedge clk)  
    if (start) counter <= 0;  
    else counter <= counter == arg2 ? counter : counter + 1;  
  
always @ (posedge clk)  
    if (start) accumulator <= 0;  
    else accumulator <= counter == arg2 ? accumulator : accumulator + arg1;  
  
assign done = counter == arg2;  
assign product = accumulator;
```