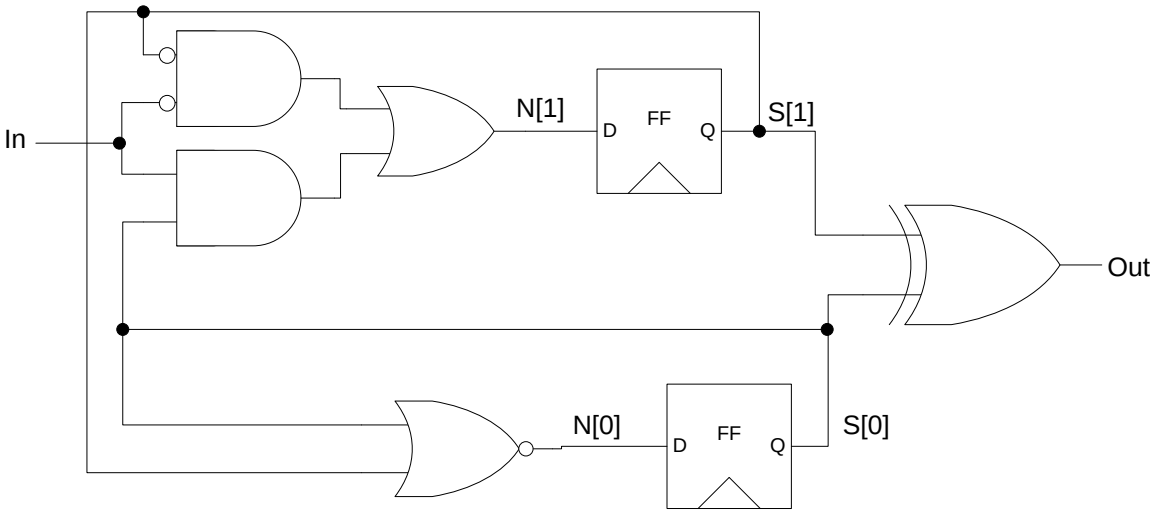
University of California at Berkeley
College of Engineering
Department of Electrical Engineering and Computer Science

EECS150, Spring 2010

Quiz 3 Solutions

Consider the following circuit for questions 1-2:



1. Write a Verilog description of the circuit, using continuous assignment wherever possible.

   For each of the combinational signals, write down the equation based on the gates in the circuit. Assigns can be used for all the combinational signals in the circuit. The always@(posedge Clock) block implements a Clock edge-triggered propagation of N data to S (flipflops). Notice S is a reg because it is assigned to in an always block, while N is declared as a wire. Synchronous Reset for the flipflops was intended, but this was not graded since the problem and circuit diagram did not specify how to use the Reset signal.

```
module QuizCircuit(Clock, Reset, In, Out);
    input Clock, Reset, In;
    output Out;

    reg [1:0] S;
    wire [1:0] N;

    always @ (posedge Clock) begin
        if (Reset) S <= 2'b00;
        else S <= N;
    end

    assign N[1] = (~S[1]&~In)|(S[0]&In);
    assign N[0] = ~(S[0] | S[1]);
    assign Out = S[1] ^ S[0];
endmodule
```

2. Write a Verilog description of the circuit, using only always blocks. The most straight forward way is to assign the N and Out signals as above, with the always block syntax. Combinational logic can be implemented inside of always@* block. Since the three signals N[1], N[0], and Out do not depend on each other, they can be assigned in the always block in any order safely. Notice Out and N are reg because they are assigned in an always block.

```
module QuizCircuit(Clock, Reset, In, Out);
    input Clock, Reset, In;
    output reg Out;

    reg [1:0] S, N;

    always @(posedge Clock) begin
        if (Reset) S <= 2'b00;
        else S <= N;
    end

    always @* begin
        N[1] = (~S[1]&~In)|(S[0]&In);
        N[0] = ~(S[0] | S[1]);
        Out = S[1] ^ S[0];
    end
endmodule
```

An alternate implementation is to use a case statement for assigning N, which looks like a state machine. One way to figure this out is to write the truth table for N[1], N[0].

S[1]	S[0]	In	N[1]	N[0]
0	0	0	1	1
0	0	1	0	1
0	1	0	1	0
0	1	1	1	0
1	0	0	0	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	0

For each of the four cases for the value of S, assign N based on the input In.

```
module QuizCircuit(Clock, Reset, In, Out);
    input Clock, Reset, In;
    output reg Out;

    reg [1:0] S, N;

    always @(posedge Clock) begin
        if (Reset) S <= 2'b00;
        else S <= N;
    end

    always @* begin
        Out = S[1] ^ S[0];

        case (S)
            2'b00 : begin
```

```
                if (In) N = 2'b01;
                else N = 2'b11;
            end
            2'b01 : N = 2'b10;
            2'b10 : N = 2'b00;
            2'b11 : begin
                if (In) N = 2'b10;
                else N = 2'b00;
            end
        endcase
    end
endmodule
```

The state-transition diagram is shown below.