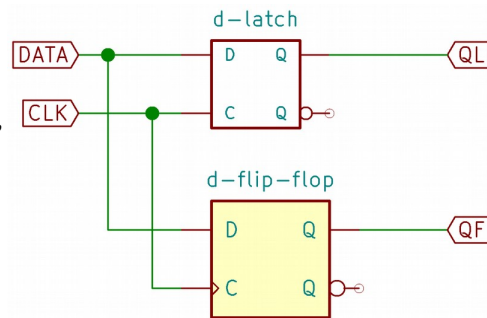
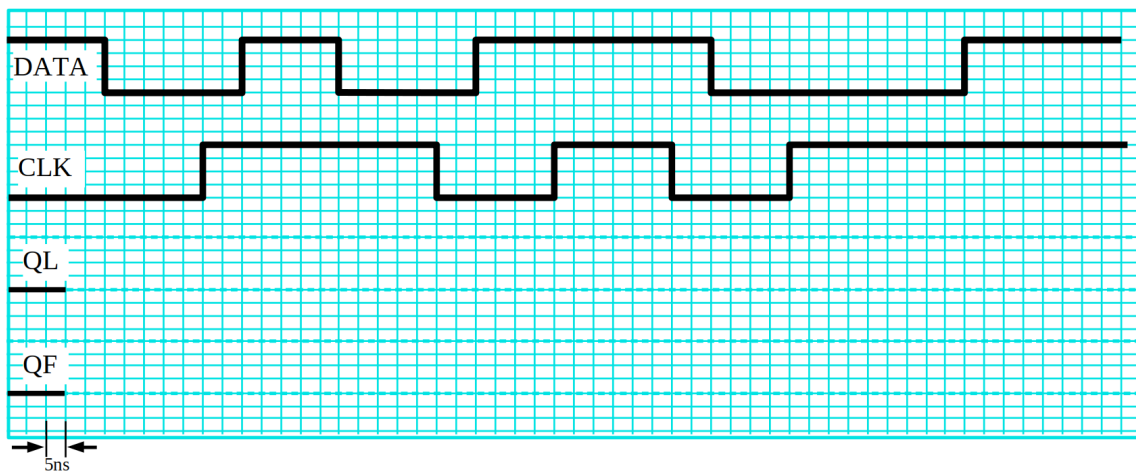


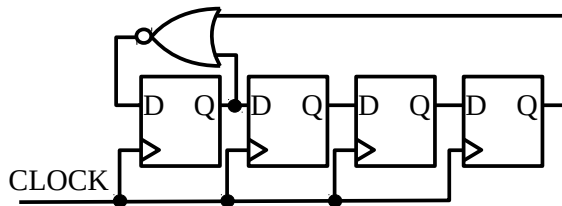
Use the following circuit consisting of a latch and a D flip-flop for the next two questions. Assume that, for each device in question,  $t_{pHL(C \rightarrow Q)} = 5 \text{ ns}$ ,  $t_{pLH(C \rightarrow Q)} = 10 \text{ ns}$ ,  $t_{pHL(D \rightarrow Q)} = 15 \text{ ns}$ , and  $t_{pLH(D \rightarrow Q)} = 20 \text{ ns}$ . (Not all parameters are relevant to all devices.) Assume that all setup and hold requirements are met. The initial states of QL and QF are 0.



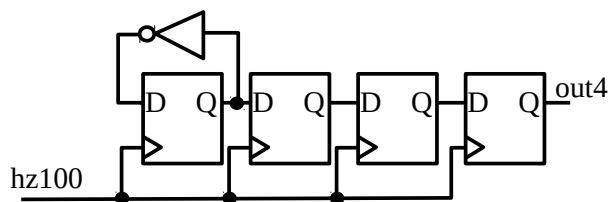
- 1) On the timing diagram below, draw the expected signal for QL.
- 2) On the same diagram below, draw the expected signal for QF.



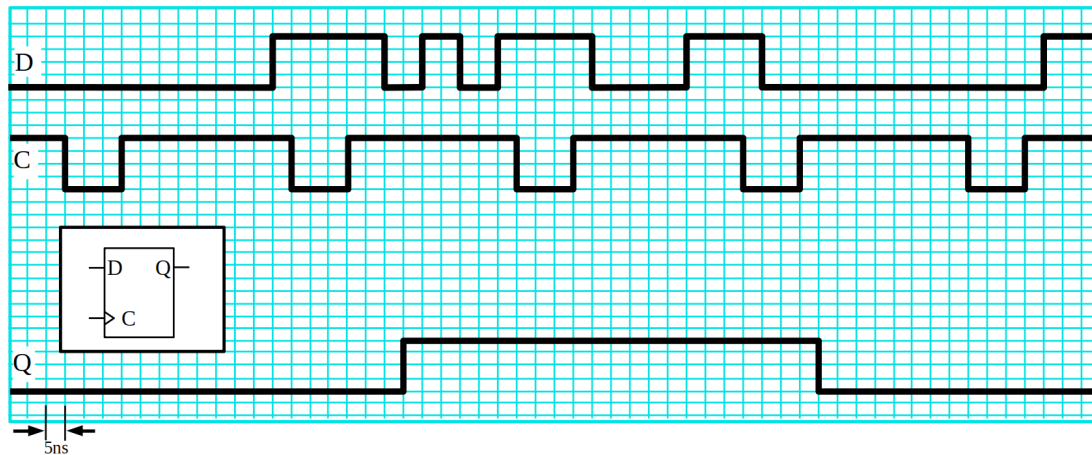
- 3) For the figure below, assume that  $t_{pHL(C \rightarrow Q)}$  and  $t_{pLH(C \rightarrow Q)}$  for the flip-flops are both 11 ns. The setup time ( $t_s$ ) for the flip-flops is 7 ns. The hold time ( $t_h$ ) for the flip-flops is 0 ns. The propagation delays ( $t_{pHL}$  and  $t_{pLH}$ ) for the NOR gate are 5 ns. If CLOCK is a periodic square wave, what is the minimum period that will avoid any flip-flop exhibiting metastability while the circuit still performs as it is expected to?



- 4) For the figure below, if hz100 is a 100 Hz square wave, describe the out4 signal.



Use the diagram below for the following questions about the use of a D flip-flop.



- 5) What type of event causes the flip-flop to copy the D input value to the Q output?
- 6) What is  $t_{pHL(C \rightarrow Q)}$ ? (in nanoseconds, not tickmarks)
- 7) What is  $t_{pLH(C \rightarrow Q)}$ ? (in nanoseconds)
- 8) What is the nominal setup time? (in nanoseconds)
- 9) What is the nominal hold time? (in nanoseconds)
- 10) What is the clock period? (in nanoseconds)
- 11) What is the clock's pulse width? (in nanoseconds)
- 12) What is the clock's duty cycle?

- 13) For the following Verilog statements, cross out all **begin** and **end** statements that can be removed without changing the behavior of the design.

```

always @ (posedge clock, posedge reset) begin
    if (reset == 1'b1) begin
        x <= 1'b0;
    end
    else if (a == 1'b1) begin
        x <= y;
    end
    else begin
        x <= 1'b1;
    end
    z <= x;
end

```

- 14) For the following Verilog statements, how many state elements are needed to realize this specification in hardware, and are they flip-flops or latches?

```

reg x,y;
wire z, clock;
...
always @ (posedge clock) begin
    y = z;
    x = y;
end

```

- 15) For the following Verilog code, how many state elements are needed to realize this specification in hardware, and are they flip-flops or latches?

```

reg [2:0] x;
wire [1:0] n;
...
always @ (*) begin
    case (n)
        2'b00: x = 3'b101;
        2'b01: x = 3'b000;
        2'b10: x = 3'b111;
        // Note: forgot the case for 2'b11.
    endcase
end

```

- 16) For the following Verilog statements, how many state elements are needed to realize this specification in hardware, and are they flip-flops or latches?

```
reg x,y;
wire a,b,c;
...
always @ (*) begin
    if (a == 1) begin
        x = b;
    end
    else begin
        y = c;
    end
end
end
```

- 17) For the following Verilog statements, how many state elements are needed to realize this specification in hardware, and are they flip-flops or latches?

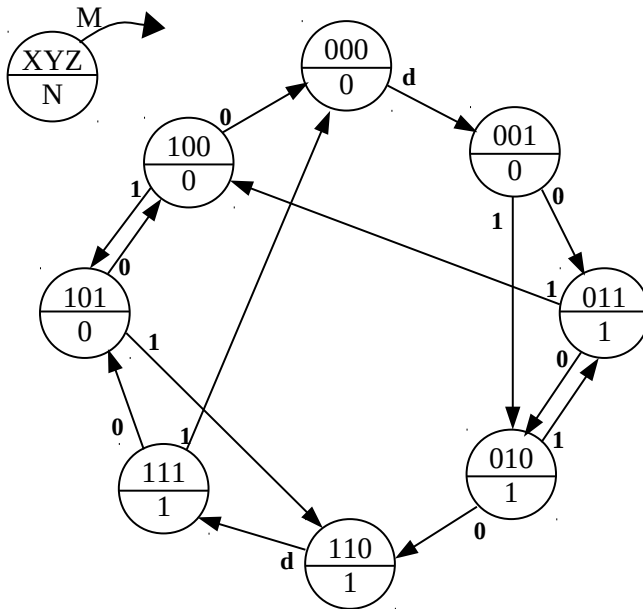
```
reg x;
wire a,b;
...
always @ (*) begin
    if (a == 1)
        x = b;
end
end
```

- 18) For the following Verilog statements,

```
reg x;
wire a,b;
...
always @ (*) begin
    if (b == 1'b0)
        x = a;
    else
        x = ~a;
end
end
```

what kind of circuit will be produced to realize this specification in hardware? (i.e., what is the relationship between **x**, **a**, and **b**?)

19) Given the following state-transition diagram, complete the present state / present output / next state table, and use it for the rest of the questions on this page.



X	Y	Z	M	N	X*	Y*	Z*
0	0	0	0				
0	0	0	1				
0	0	1	0				
0	0	1	1				
0	1	0	0				
0	1	0	1				
0	1	1	0				
0	1	1	1				
1	0	0	0				
1	0	0	1				
1	0	1	0				
1	0	1	1				
1	1	0	0				
1	1	0	1				
1	1	1	0				
1	1	1	1				

20) The start state is 000. If M is always 0, what repeating output pattern is produced on N?

21) The start state is 000. If M is always 1, what repeating output pattern is produced on N?

- 22) For the following Moore machine, whose start state is 000, choose next state values such that the Z bit follows the pattern: **0 1 1 1 1 0 0 0 1 1 1 1 0 0 ...**

Note that there are many arrangements of next state values that will produce the correct pattern for the Z bit, and many others that will not. You do not need to make an optimal choice. Any next state combination that produces the correct pattern for Z is as good as any of the others. Draw a line through any unused states.

X	Y	Z	X*	Y*	Z*
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

- 23) For the following Verilog code complete the counter equations to implement a 4-bit synchronous up-counter **that resets at strange starting and maximum values.**

```

reg [3:0] Q;
assign left[0] = Q[3];
always @ (posedge hz100, posedge reset) begin
    if (reset == 1)          begin Q <= 4'b0111; end
    else if (Q == 4'b1011)   begin Q <= 4'b0111; end
    else                     begin Q <= next_Q;   end
else
    always @ (Q) begin
        next_Q[0] =

        next_Q[1] =

        next_Q[2] =

        next_Q[3] =
    end

```

- 24) For the previous question, if the hz100 signal is a 100 Hz square wave, at what frequency does the LED on left[0] flash? (You may express your answer as a fraction.)

- 25) Complete the Verilog case statement below to implement a sequence recognizer that will light the **green** LED if it reads the input sequence 0 0 1 1 on pb[1] at each rising edge of pb[0]. Once **green** is illuminated, it should remain on until reset. If, at any point, the input sequence diverges from 0 0 1 1, it should illuminate **red** and keep it lit until reset.

```

module top(hz100, reset, red, green, ..., pb);
    input hz100, reset;
    output red, green;
    input [20:0] pb;

    reg [2:0] state, next;

    always @ (posedge pb[0], posedge reset) begin
        if (reset == 1'b1) begin
            state <= 3'b000;
        end
        else begin
            state <= next;
        end
    end

    assign green = state == 3'b100; // show green for state 100
    assign red = state == 3'b111; // show red for state 111

    always @ (*) begin
        case ({state, pb[1]})

            default: next = 3'b111; // go to red

            4'b0000: next = 3'b001; // advance to 001

            // Complete the implementation below...

        endcase
    end
endmodule

```