Flip-Flops and Registers

Exercise 1: Is a signal named **$\overline{\text{overload}}$** active-high or active-low? Is there an overload if this signal is high? What if the signal was named **overload**?

$\overline{\text{overload}}$: active low

H → false : no overload. (overload is false).

overload : active high

H → true : overload (overload is true.)

Exercise 2: Come up with an appropriate name for a signal that is at 3 V when a door is open and 0 V when the door is closed.

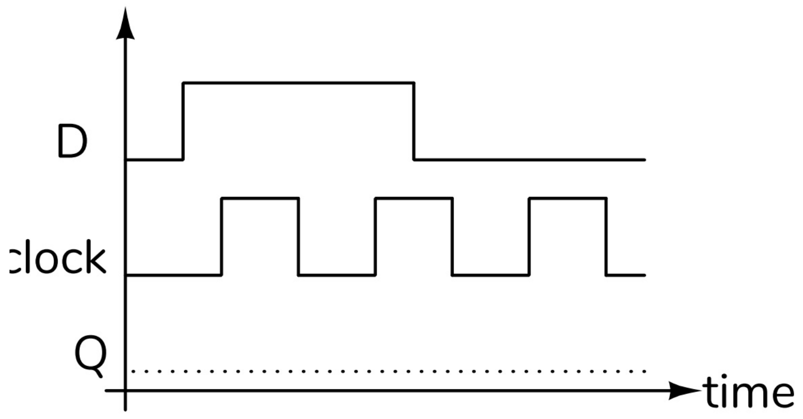
3V → H → open

0 V → L → closed

door_open ✓ active high

$\overline{\text{door\_closed}}$ active low

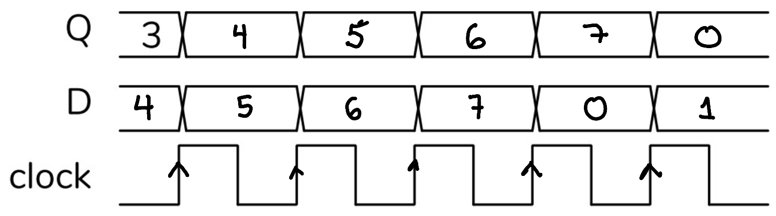
Exercise 3: Fill in the waveform for the Q signal in the diagram above.

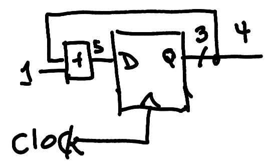


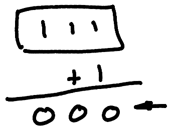
Exercise 4: What would be another name for a 1-bit register?

FF (flip-flop).

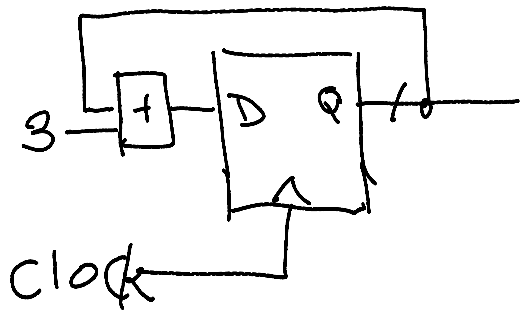
Exercise 5: Assuming a 3-bit counter, fill in the values of D and Q in the diagram above.

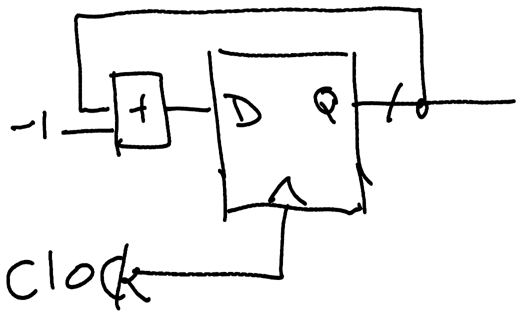


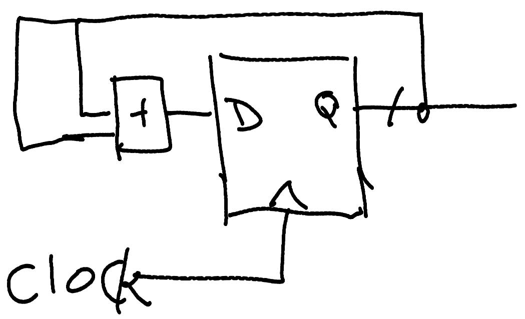


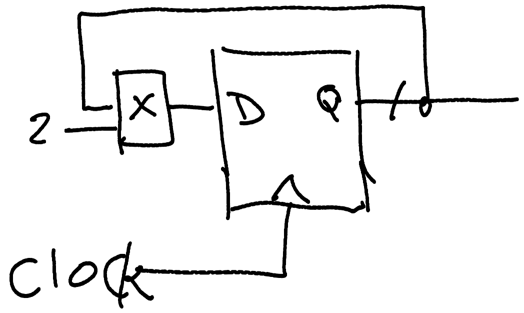


Exercise 6: Draw the block diagram for a counter that: (a) counts up by 3? (b) counts down by 1? (c) whose value doubles on each clock edge?

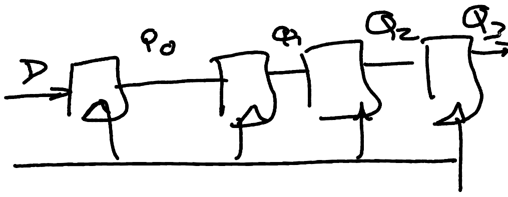


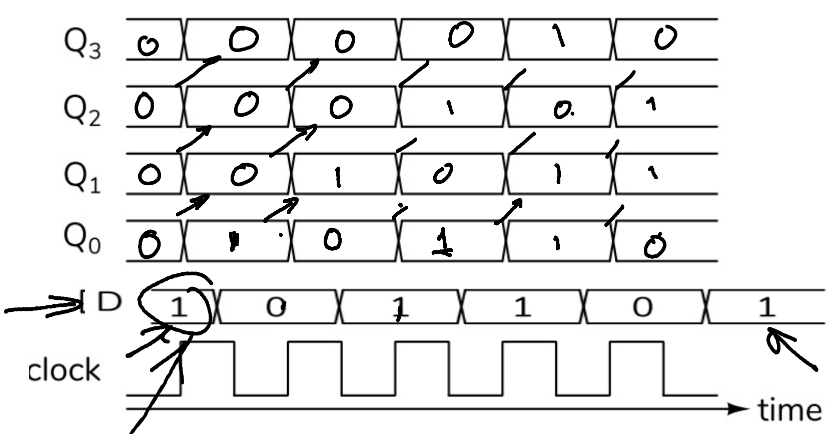


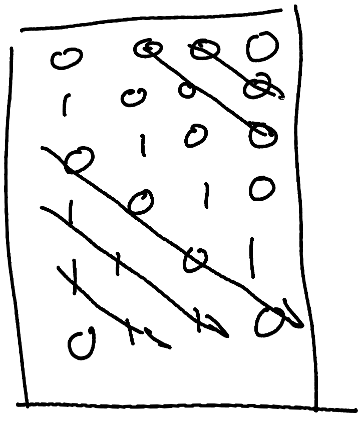




Exercise 7: Fill in the diagram above for a 4-bit shift register. Assume the initial value of each flip-flop is zero. Which is the oldest (first) value the D waveform? Which flip-flop holds the oldest value?

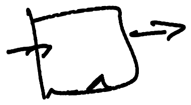




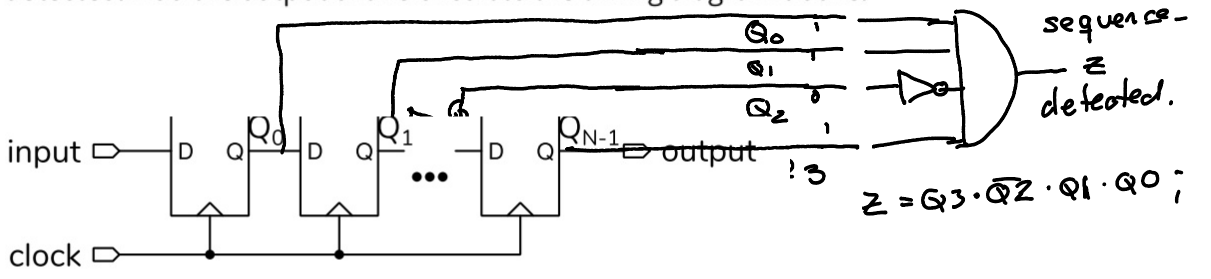


oldest bit on left.

Q_3 has oldest (first) value



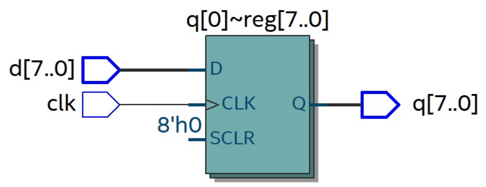
Exercise 8: Add parallel outputs to the shift register schematic. Draw a circuit whose output is high when the sequence 1, 0, 1, 1 is detected. Add the output of this circuit to the timing diagram above.



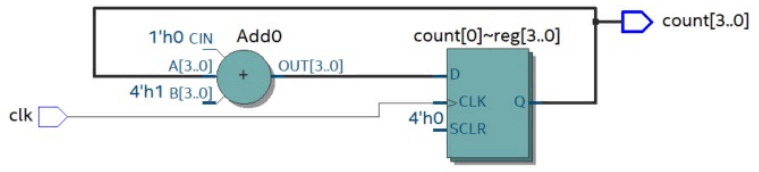
$Z = Q3 \cdot \overline{Q2} \cdot Q1 \cdot Q0$;

Exercise 9: What would you change to make an 8-bit register? A 4-bit counter? A 3-bit shift register? Follow the course coding conventions.

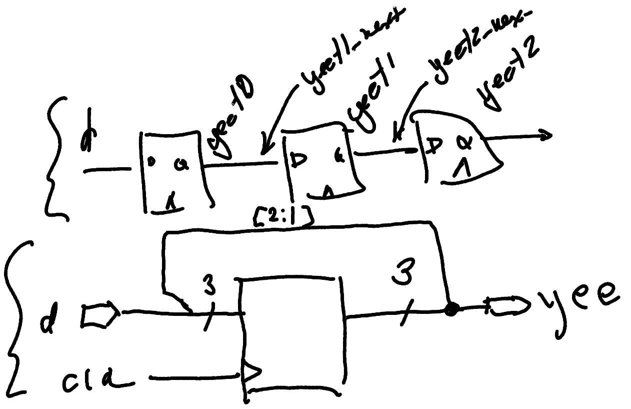
```
module test
  ( input logic [7:0] d,
    input logic clk,
    output logic [7:0] q ) ;

   always_ff @(posedge clk) q = d ;

endmodule
```

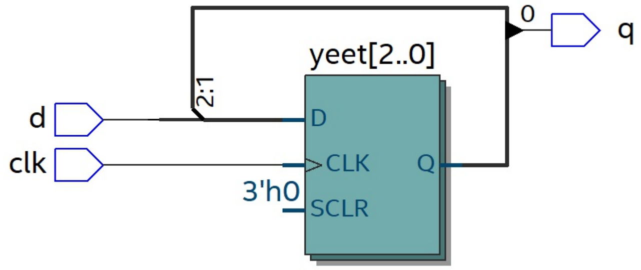


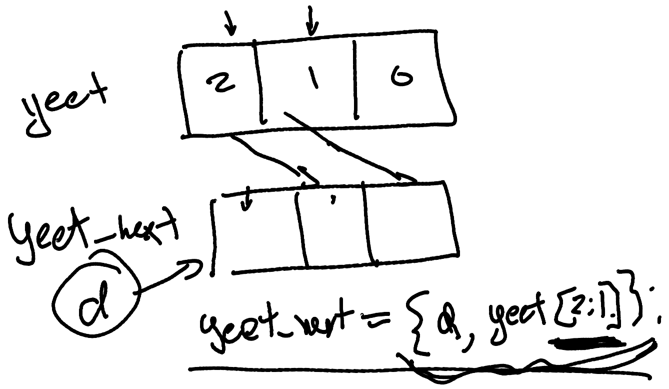
```
 module test
   ( // input logic d,
     input logic clk,
     output logic [3:0] count ) ;

    logic [3:0] count_next ;

    assign count_next = count + 1'b1 ;

    always_ff @(posedge clk) count = count_next ;

 endmodule
```



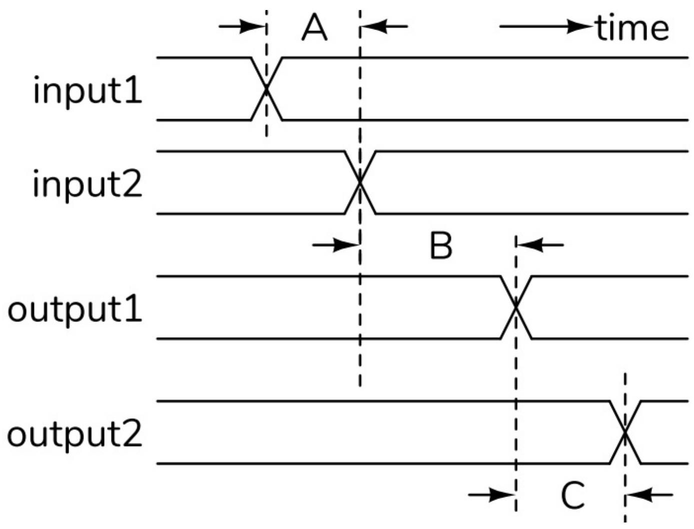
```
module test
  ( input logic d,
    input logic clk,
    output logic q ) ;

  logic [2:0] yeet, yeet_next ;

  assign yeet_next = { d, yeet[2:1] } ;

  always_ff @(posedge clk) yeet = yeet_next ;

  assign q = yeet[0] ;

endmodule
```







Exercise 10: Label the specifications A through C as requirements or guaranteed responses.



input → input : requirement

input → output : response

output → output : response