

ENGR 210 / CSCI B441  
“Digital Design”

# Flip Flops + Sequential Logic

Andrew Lukefahr

# Announcements

- P3 is out.
  - Adds “demo” requirement. Will need real hardware.
  - Demo: create video and upload to Canvas
- Labs/Office Hours
  - Will remain “Virtual” for now
  - You will have access to Luddy 4111
  - You can work from there or from home

# Hardware Pickup

- Box containing a Basys3 FPGA + microUSB cable
- Yours to keep for the semester, return @ at end of semester
- Used for P3 onward
- Can be used @ home or in Luddy 4111.

# Hardware Pickup

- Go to the Luddy Hall
- Go to the 4<sup>th</sup> floor.
- Go to the south wing.  
Your Crimson Card  
should get you access.



# Hardware Pickup

- Go into 4111
  - First door on the right
- Enter **2-3-5** for the passcode on the door

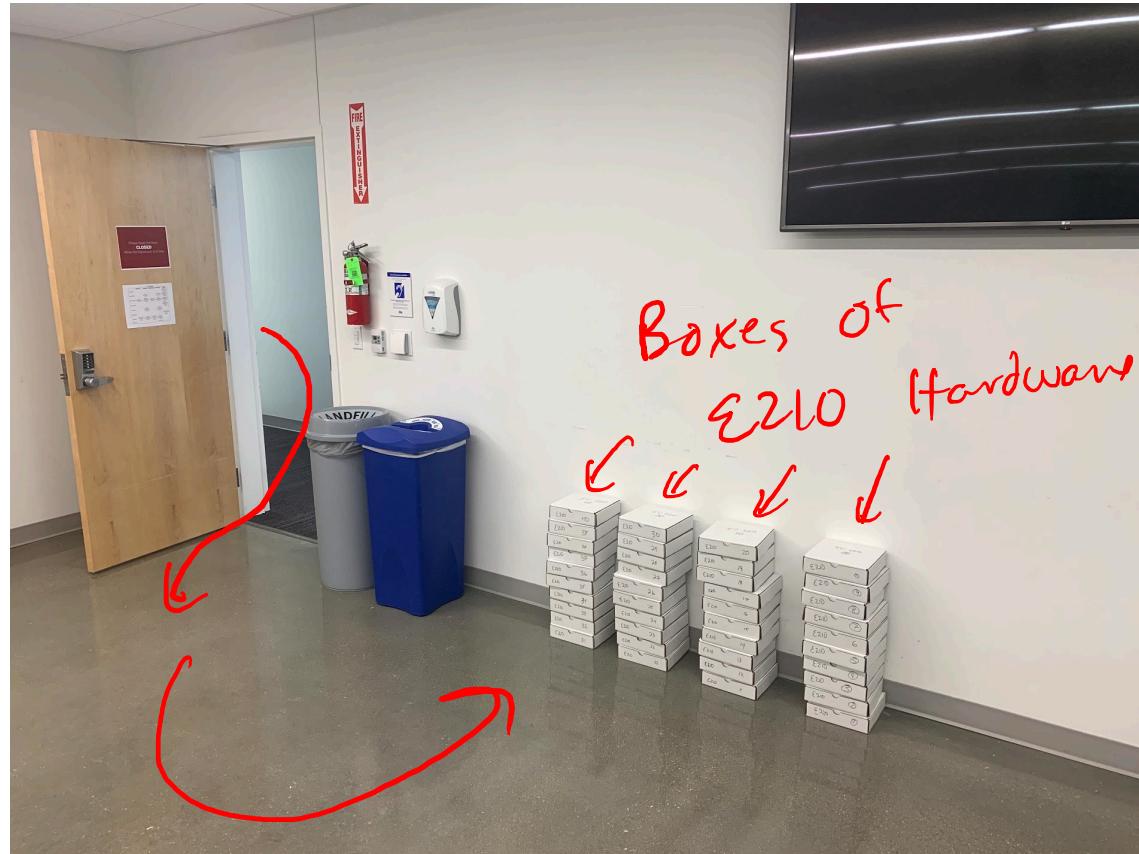
All separate,  
none at same time



# Hardware Pickup

- Turn left.
- E210 boxes should be there.
- Pick up a box.

Box says E210  
other boxes that say E315 ← NOT!



# Hardware Pickup

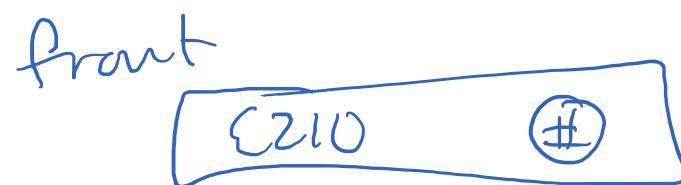
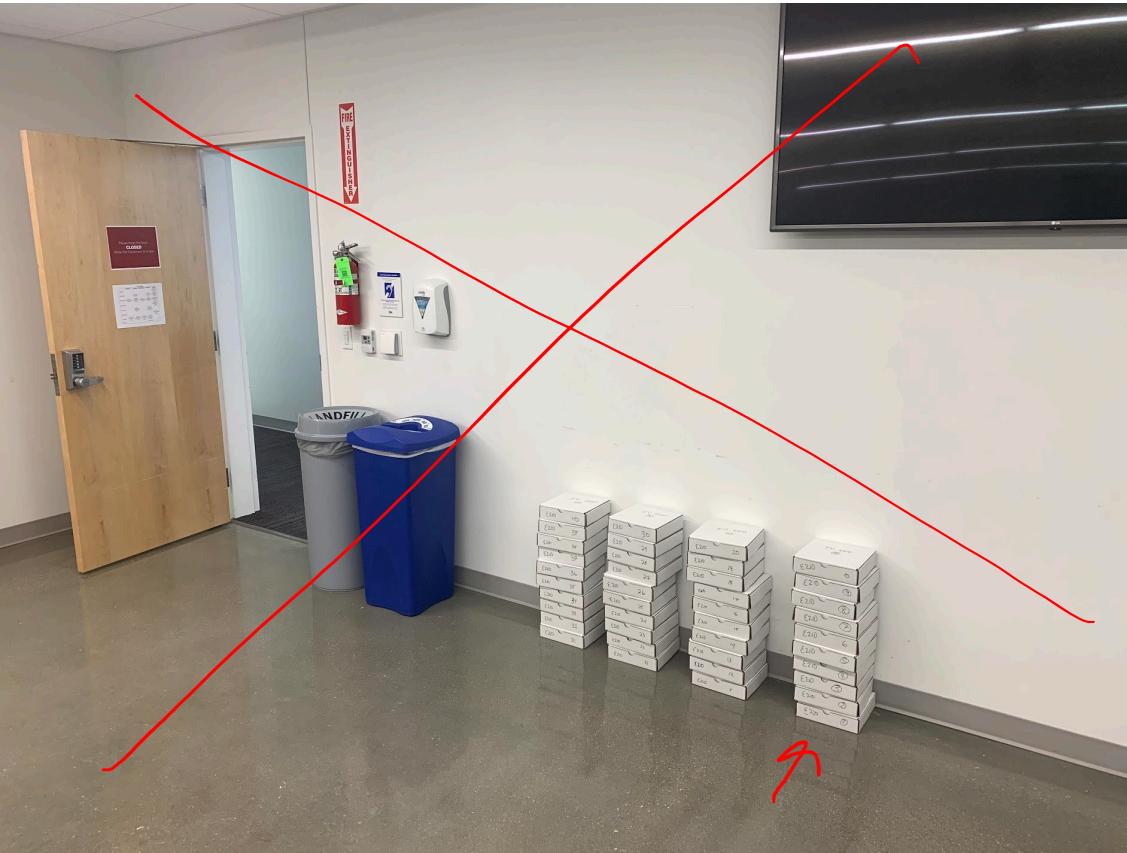
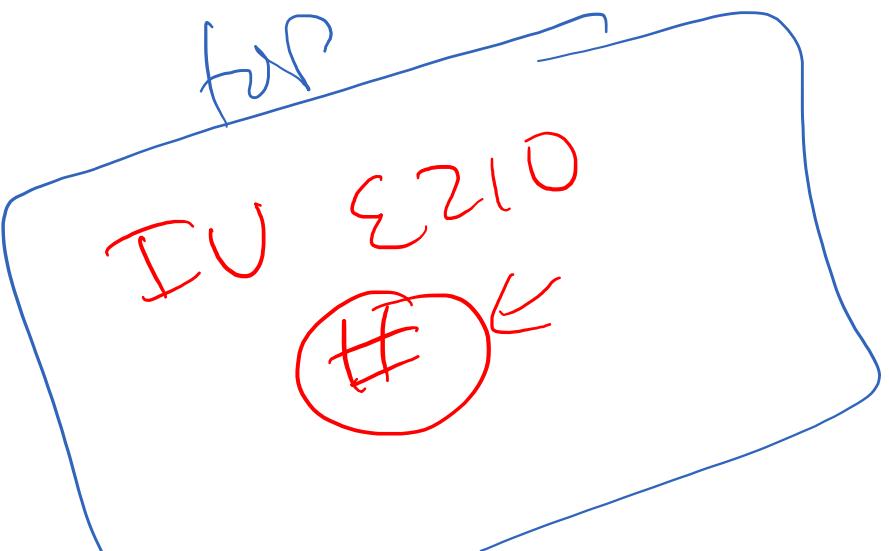
- Turn left.
- E210 boxes should be there.
- Pick up a box.



# Hardware Pickup

- Go to Canvas

- Select “P3 Box Number”
- Enter your box number



Board FPG  
(back)

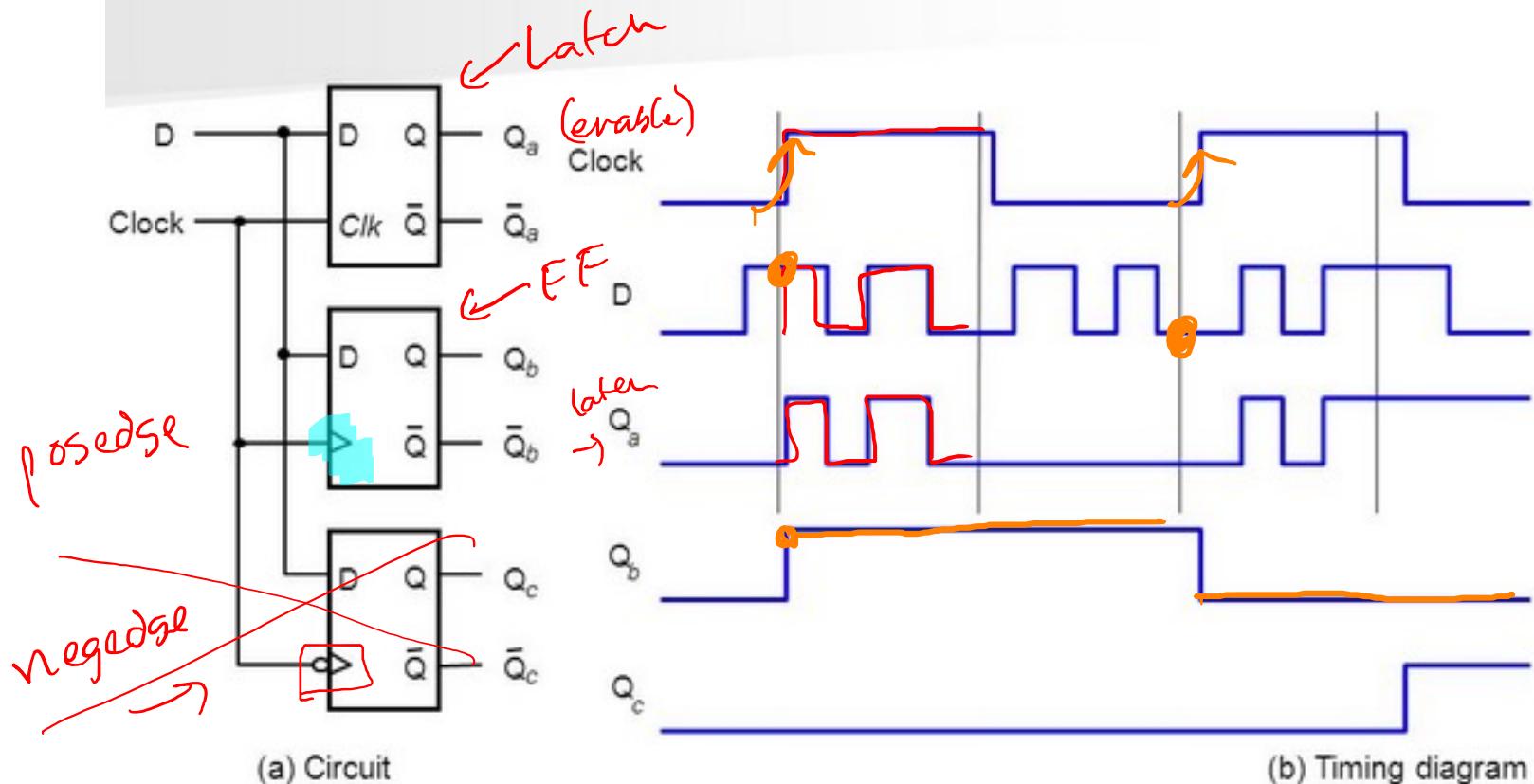
# Remote Students

- We're working on that. Hopefully more details this afternoon.

- 2 weeks to pick up  
- Canvas announcement

→ Go to lab  
→ Install locally  
→

# D Latch versus D Flip-Flop



Comparison of level-sensitive and edge-triggered devices

51

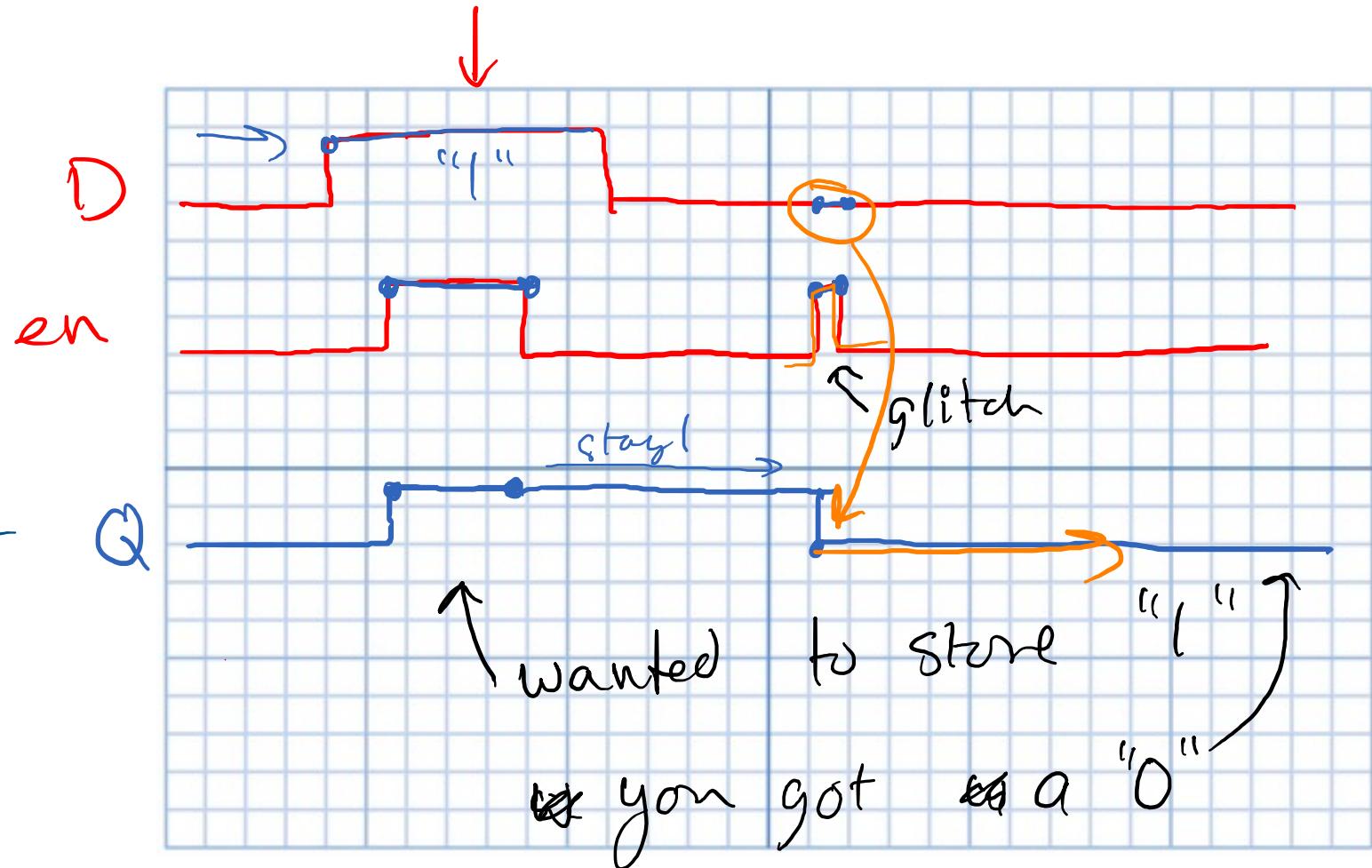
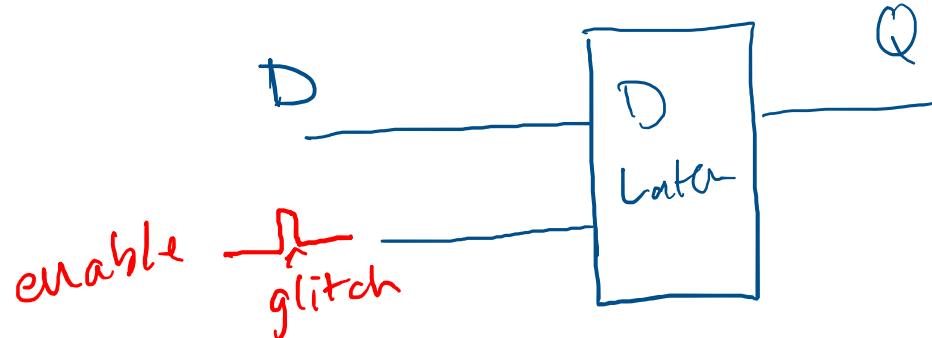
Latch  $\rightarrow$  follows input ( $D$ ) when enable ( $Clk$ ) is high

FF  $\rightarrow$  ~~follows~~ output follows input only on rising edge of enable ( $Clk$ )

"Warnis: Inferring Latex"

Always specify  
defaults for  
always\_comb!

# Glitches on D-Latches



**BLOCKING (=) FOR**

**always\_comb**

**NON-BLOCKING (<=) for**

**always\_ff**

# Blocking vs. NonBlocking

```
always_comb  
begin  
    x = a + 1; ✓  
    y = x + 1; ✓  
    z = z + 1; ✓  
    x = a + 2; ←  
end
```

```
always_ff @ (posedge clk)  
begin  
    x <= a + 1;  
    y <= x + 1;  
    z <= z + 1;  
    T  
end
```

*all happen @ same time*

Ordering does NOT matter

→ Ordering matters

$x = a + 1$        $x = a + 2$   
 $x = a + 2 \Rightarrow x = a + 1$   
 $y = x + 1$        $y = x + 1$

$x \leftarrow a + 1;$   
 $y \leftarrow x + 1;$   
 $y \leftarrow x + 1$

# Flip-Flop in Verilog

```
module d_ff (
    input          d, //data
    input          clk, //clock
    output logic q   //output register
);  
  
    always_ff @(posedge clk)
        begin
            q <= d; //non-blocking assign
        end  
  
endmodule
```

```

module counter(
    input          clk,    //clock
    input          rst,    //reset
    output logic   out    //output
);

```

```

    logic [3:0] Q;
    logic [3:0] sum;

```

*Seq logic DFF*

```

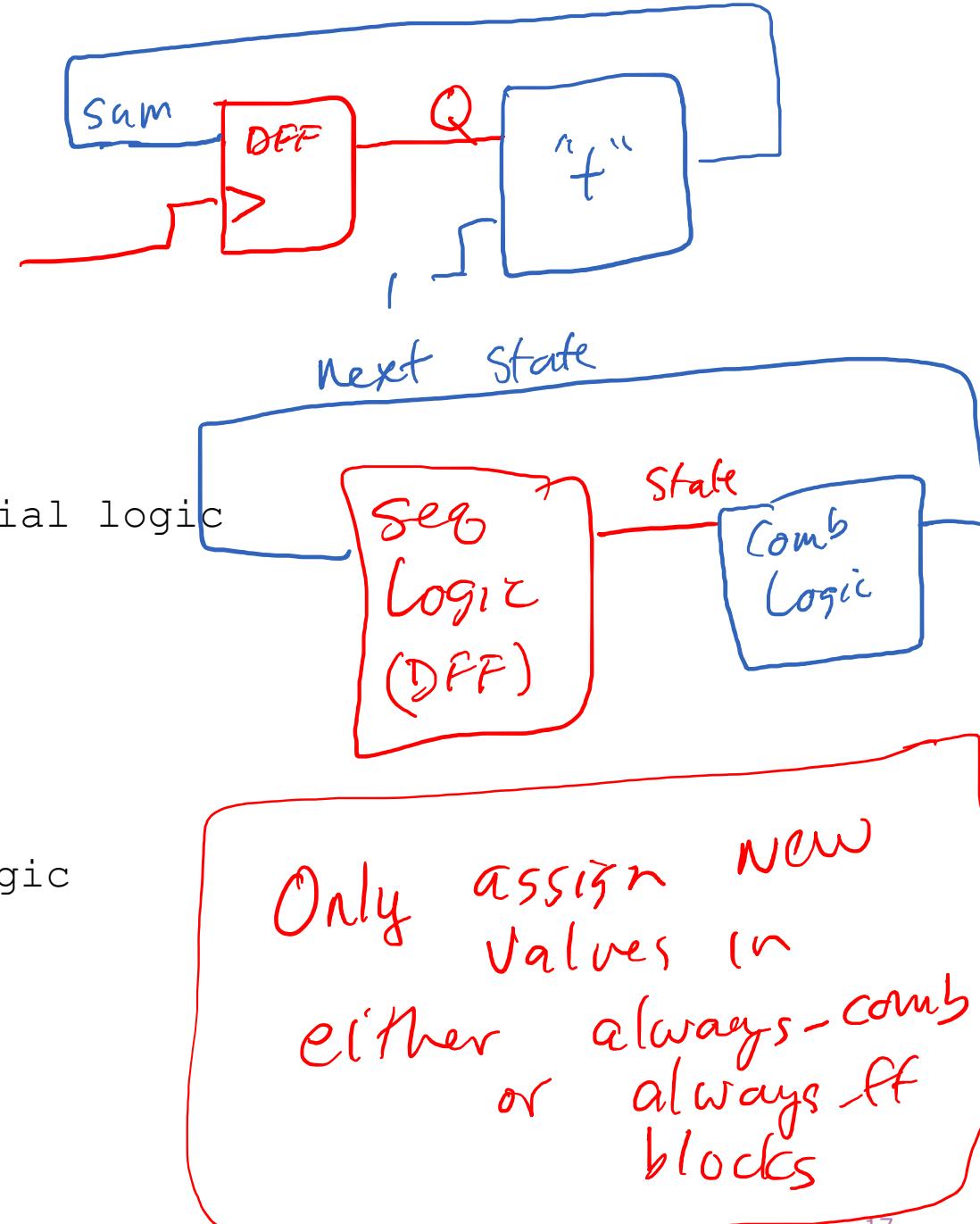
        always_ff @ (posedge clk) // <- sequential logic
        begin
            if (rst) Q <= 4'h0;
            else Q <= sum; // non-blocking
        end
    
```

*Comb logic*

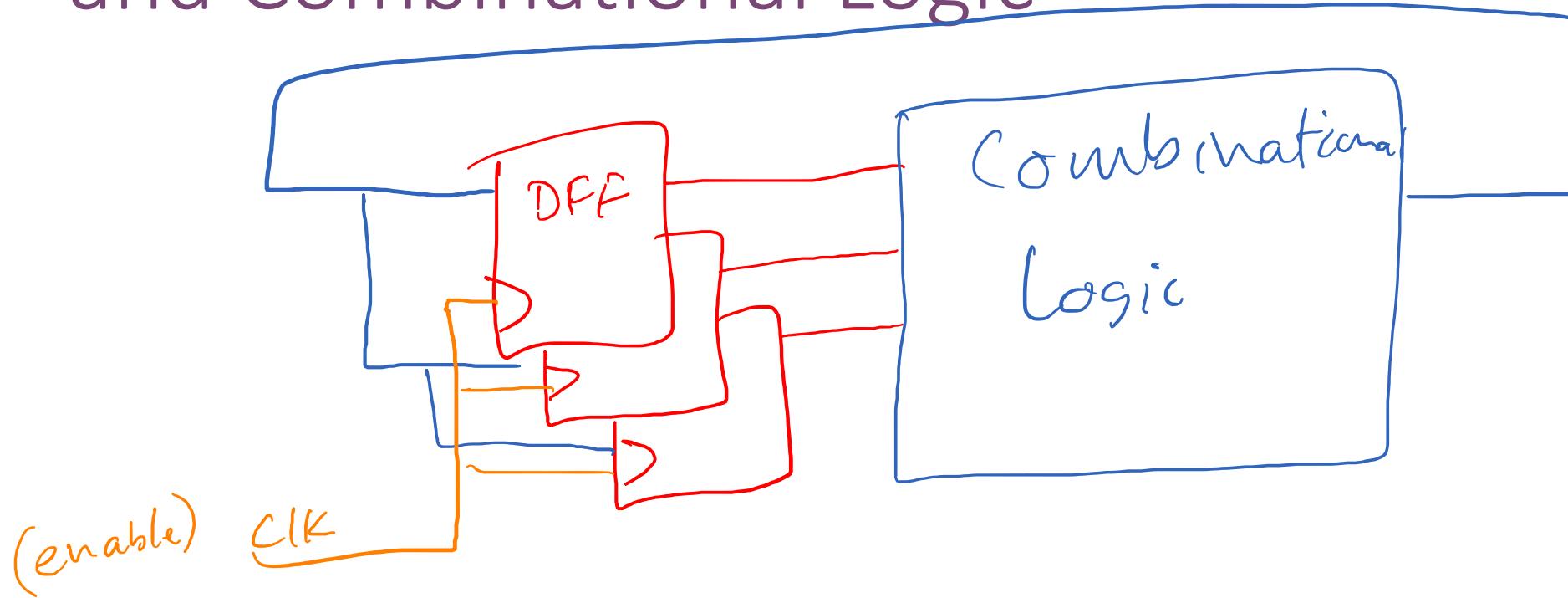
```

        always_comb begin // <- combinational logic
            sum = Q + 4'h1; // blocking
            out = sum[3];
        end
    
```

endmodule

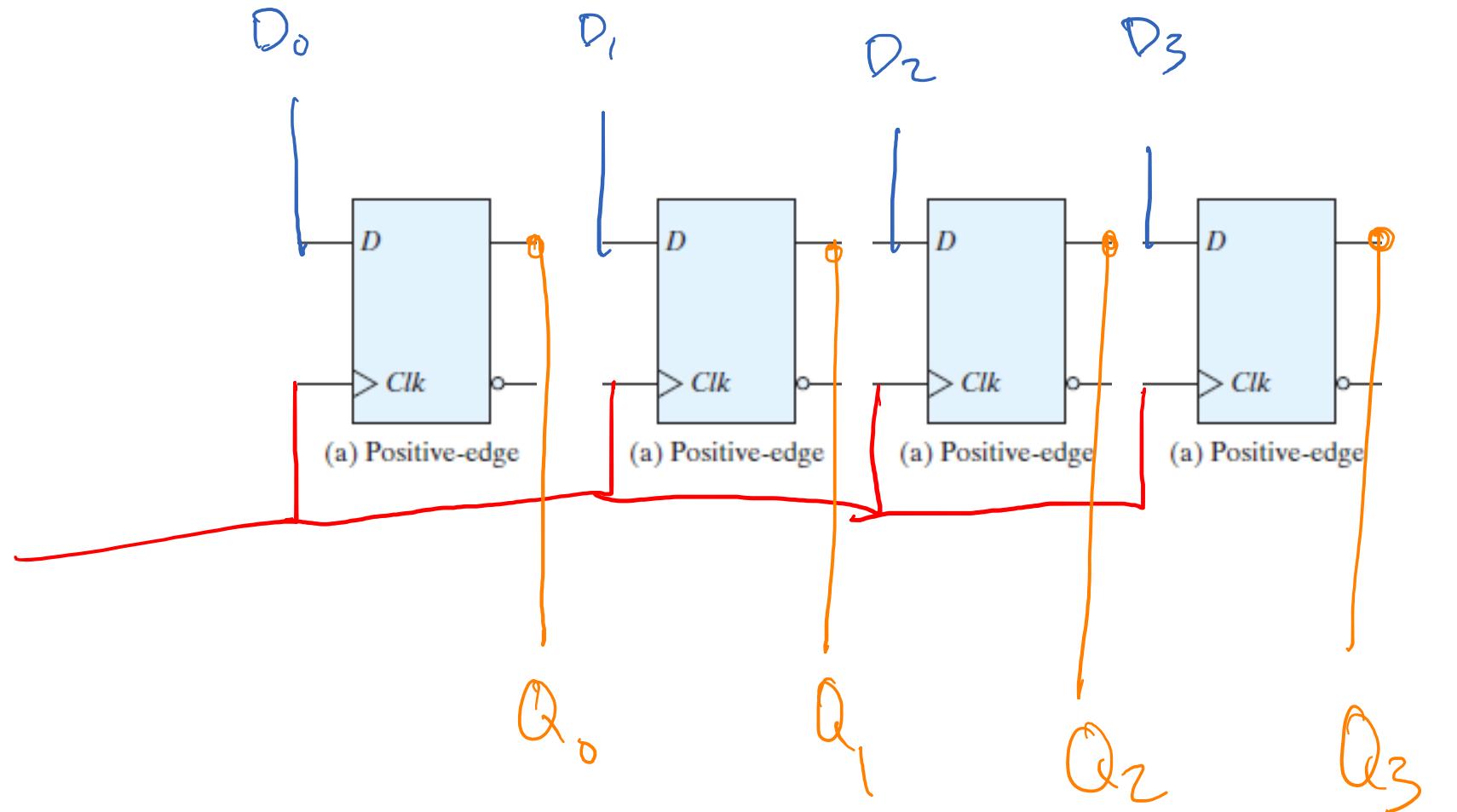


# Sequential Logic uses both Flip-Flops and Combinational Logic



# Registers

Switches



← remembered →

# 4-bit Register in Verilog

```
module d_ff (
    input      [3:0]  d,      //data
    input
    output logic [3:0] q       //output register
);
```

```
→ always_ff @ (posedge clk)
begin
    q <= d; //non-blocking assign
end
        4 bit assign
endmodule
```

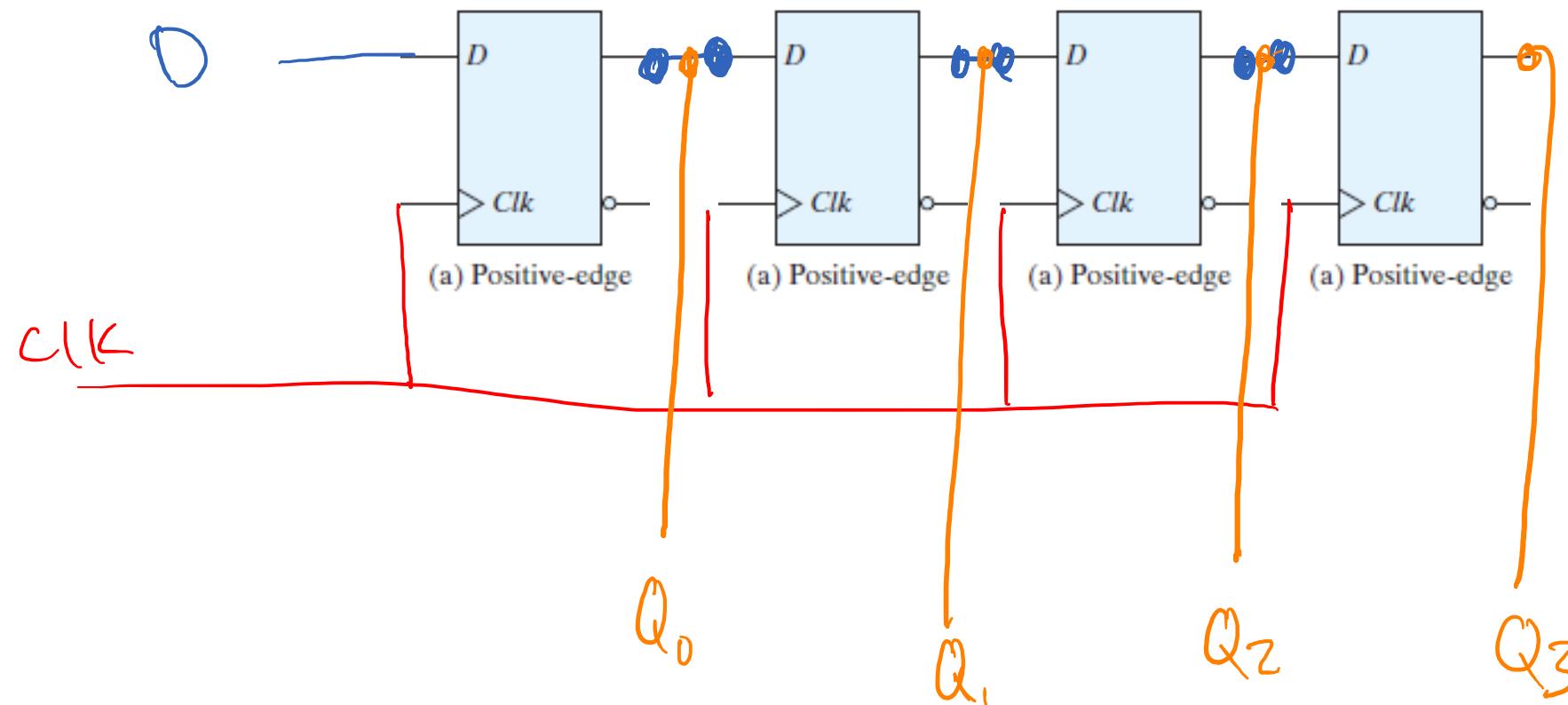
# 4-bit Register in Verilog

```
module d_ff (
    input      [3:0]  d,    //data
    input                  clk,   //clock
    output logic [3:0] q     //output register
) ;

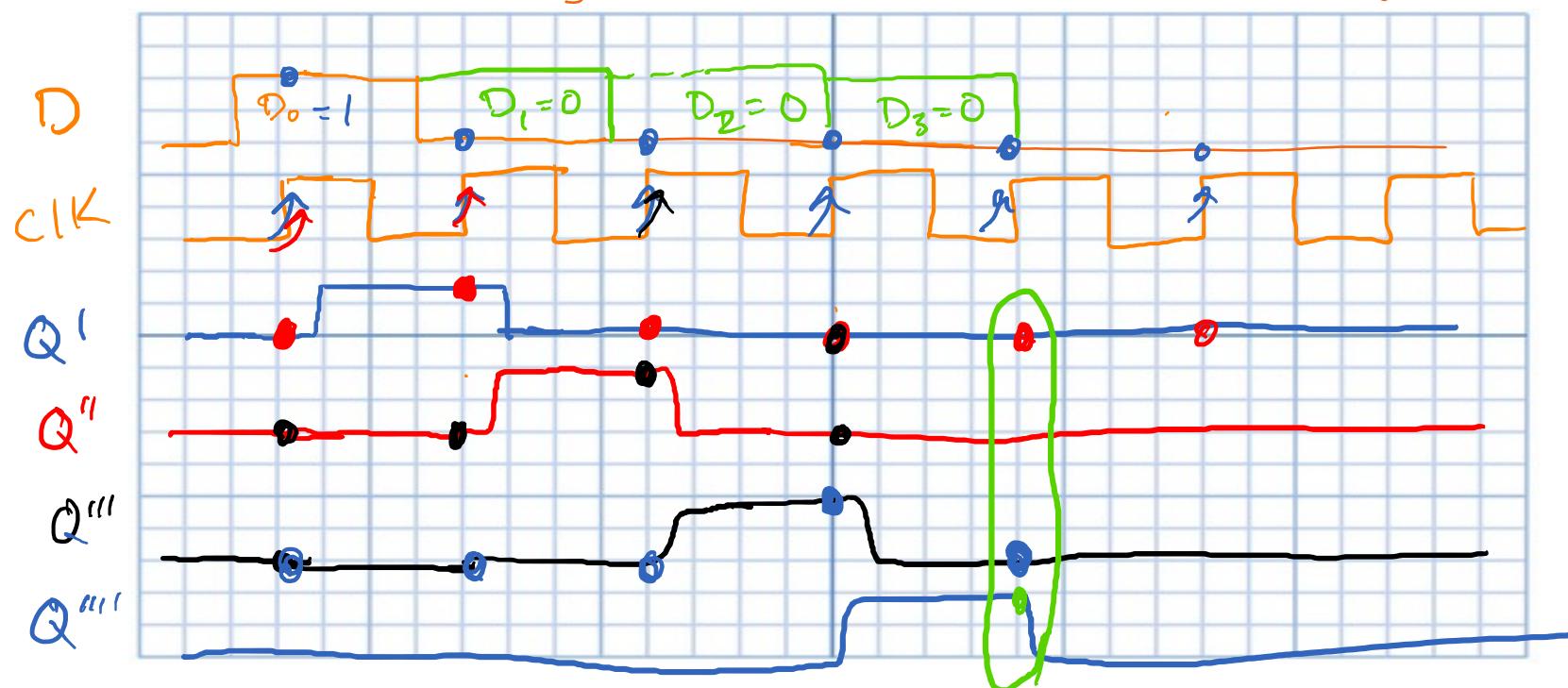
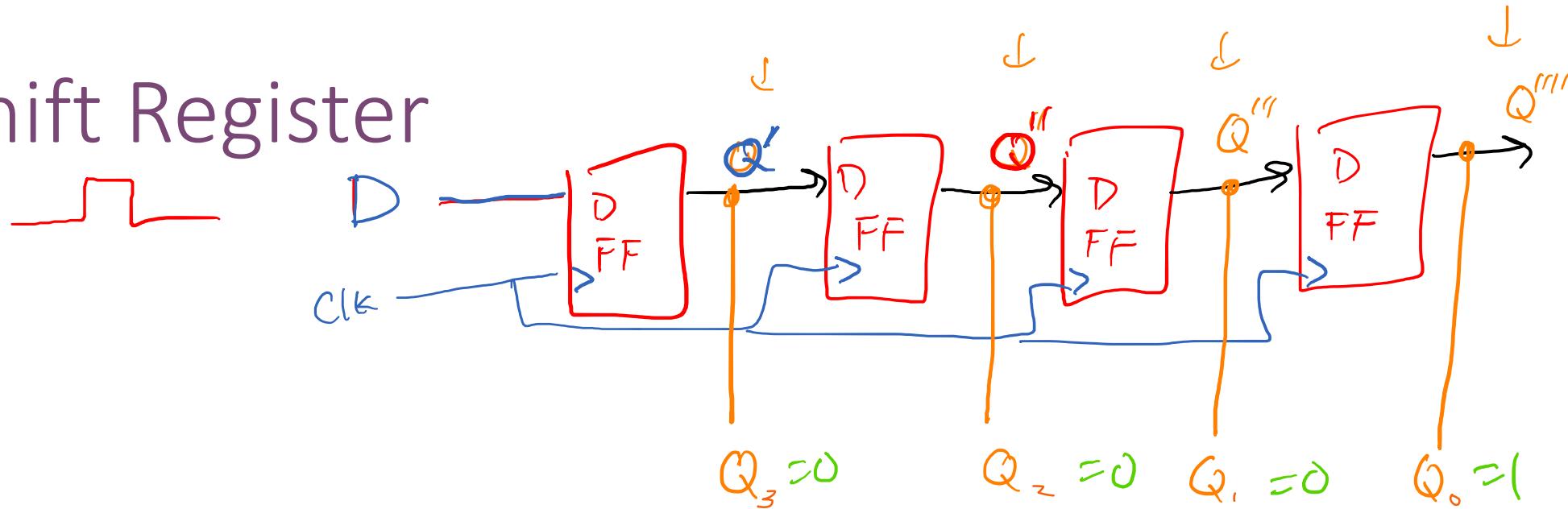
    always_ff @(posedge clk)
    begin
        q <= d; //non-blocking assign
    end

endmodule
```

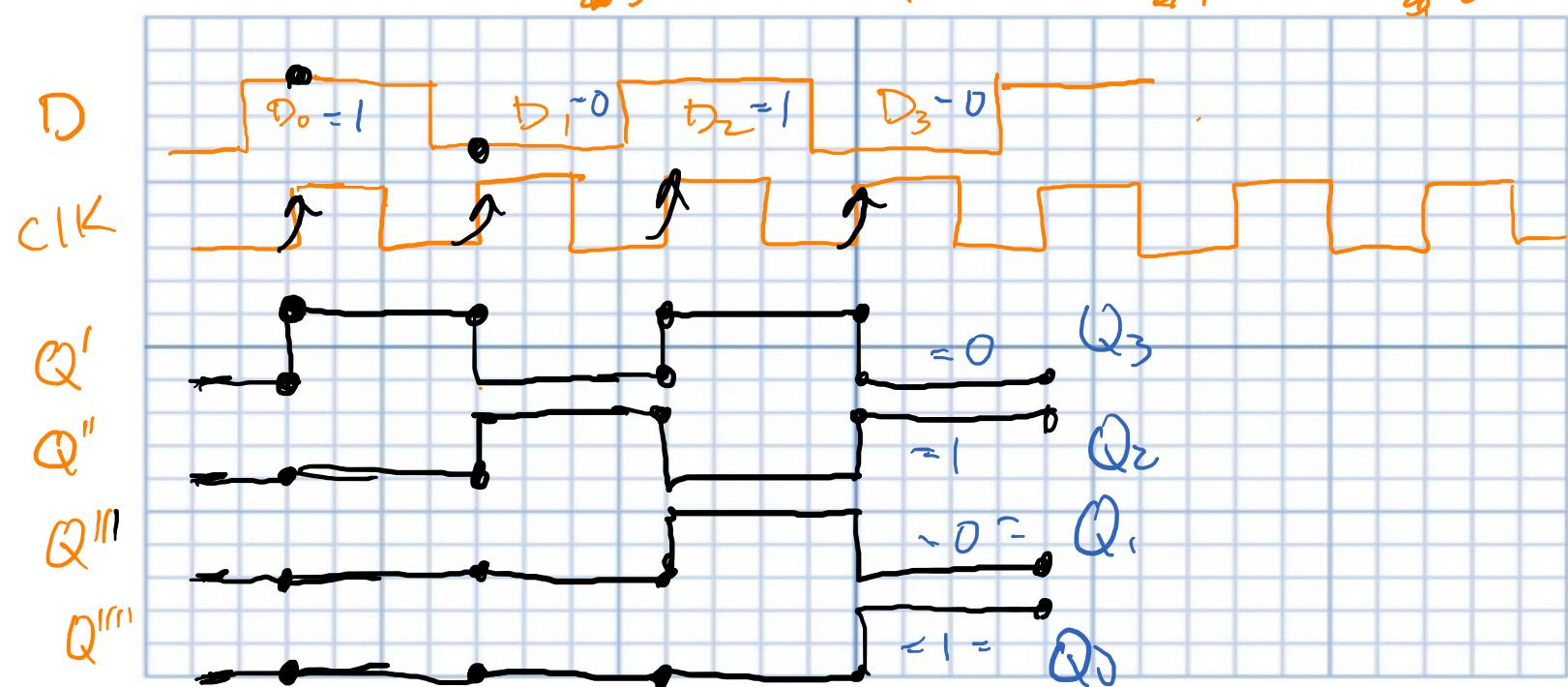
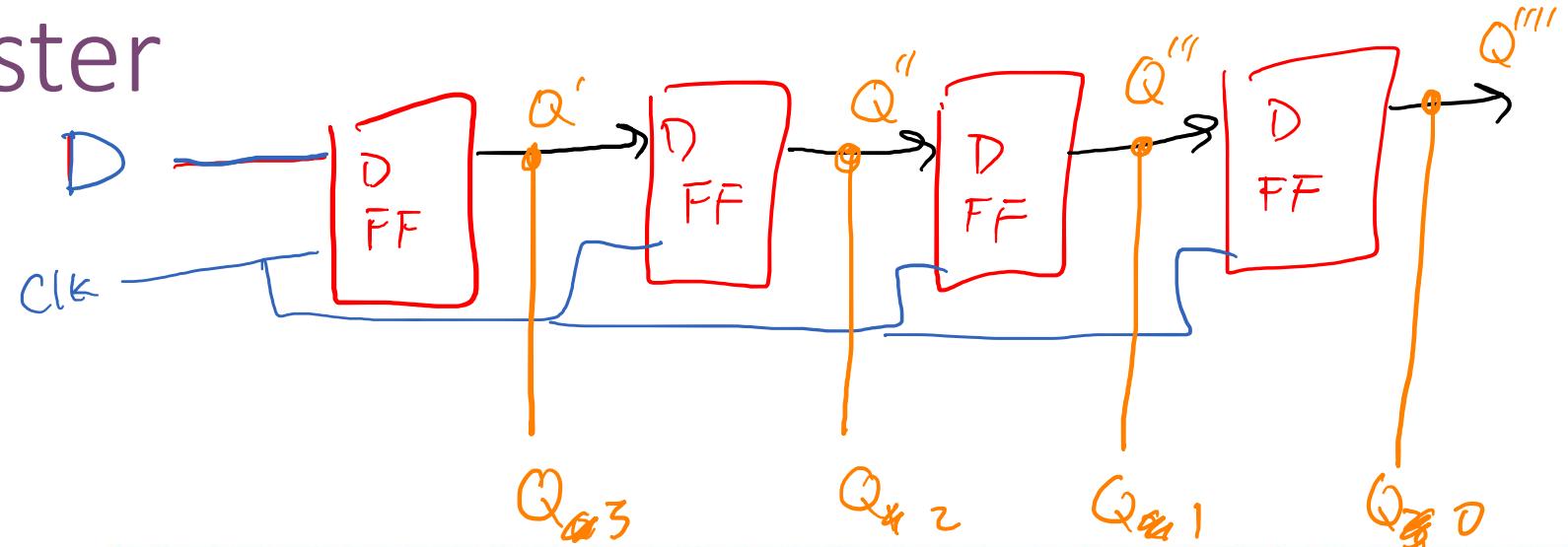
# D Flip-Flops as Shift Registers



# Shift Register



# Shift Register



# Shift-Register in Verilog

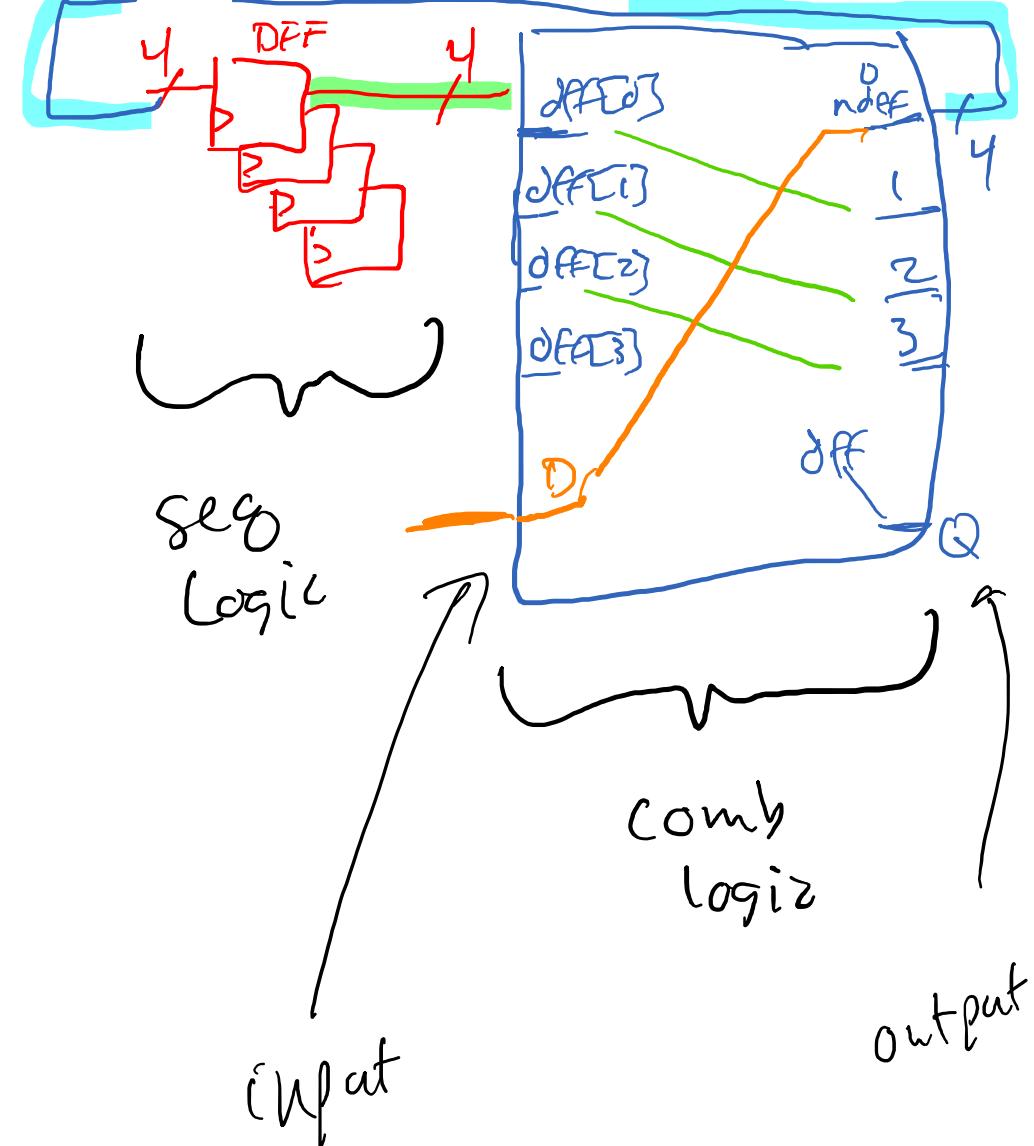
```

module shift_register (
    input clk, rst, D,
    output [3:0] Q );
    logic [3:0] off; // start
    logic [3:0] ndff; // next state
    always_ff @(posedge clk) begin
        if (rst) begin
            off <= 4'h0;
        end
        else begin
            off <= ndff;
        end
    end
    always_comb begin
        ndff = { dff[2:0], D };
    end
    assign Q = off;
endmodule

```

Sel Logik

Comb Logik



# Shift-Register in Verilog

```
module shift_register (
    input clk, rst, D,
    output [3:0] Q );

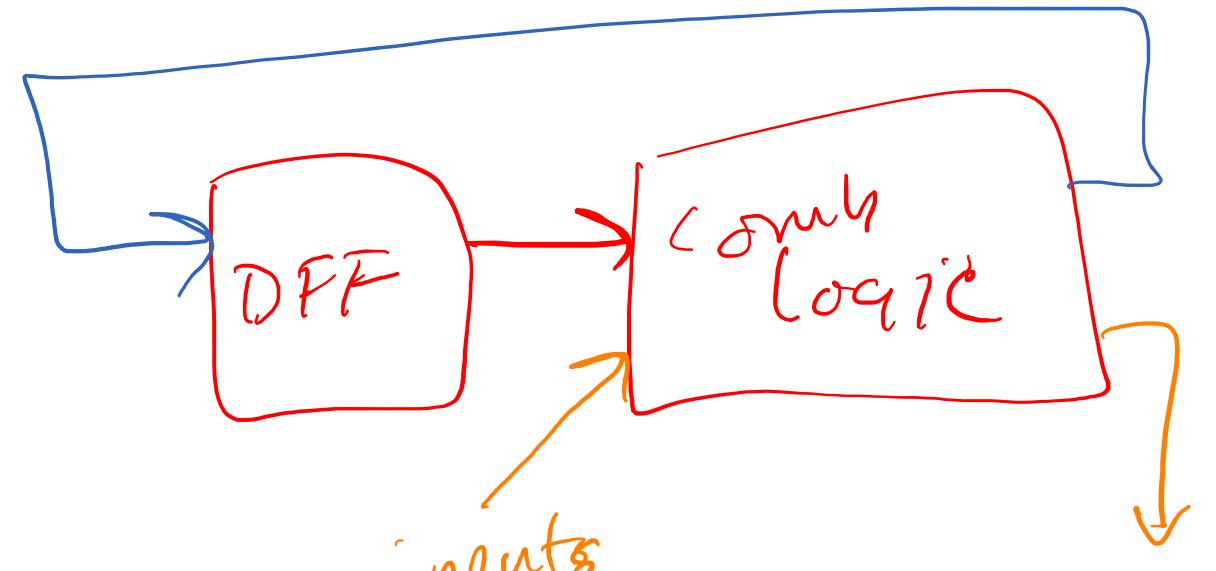
    logic [3:0] dff;
    logic [3:0] next_dff;

    always_ff (@posedge clk) begin
        if (rst) dff <= 4'h0;
        else      dff <= next_dff;
    end

    always_comb
        next_dff = { dff[2:0], D };

    assign Q = dff;

endmodule
```



Inputs can affect output or state

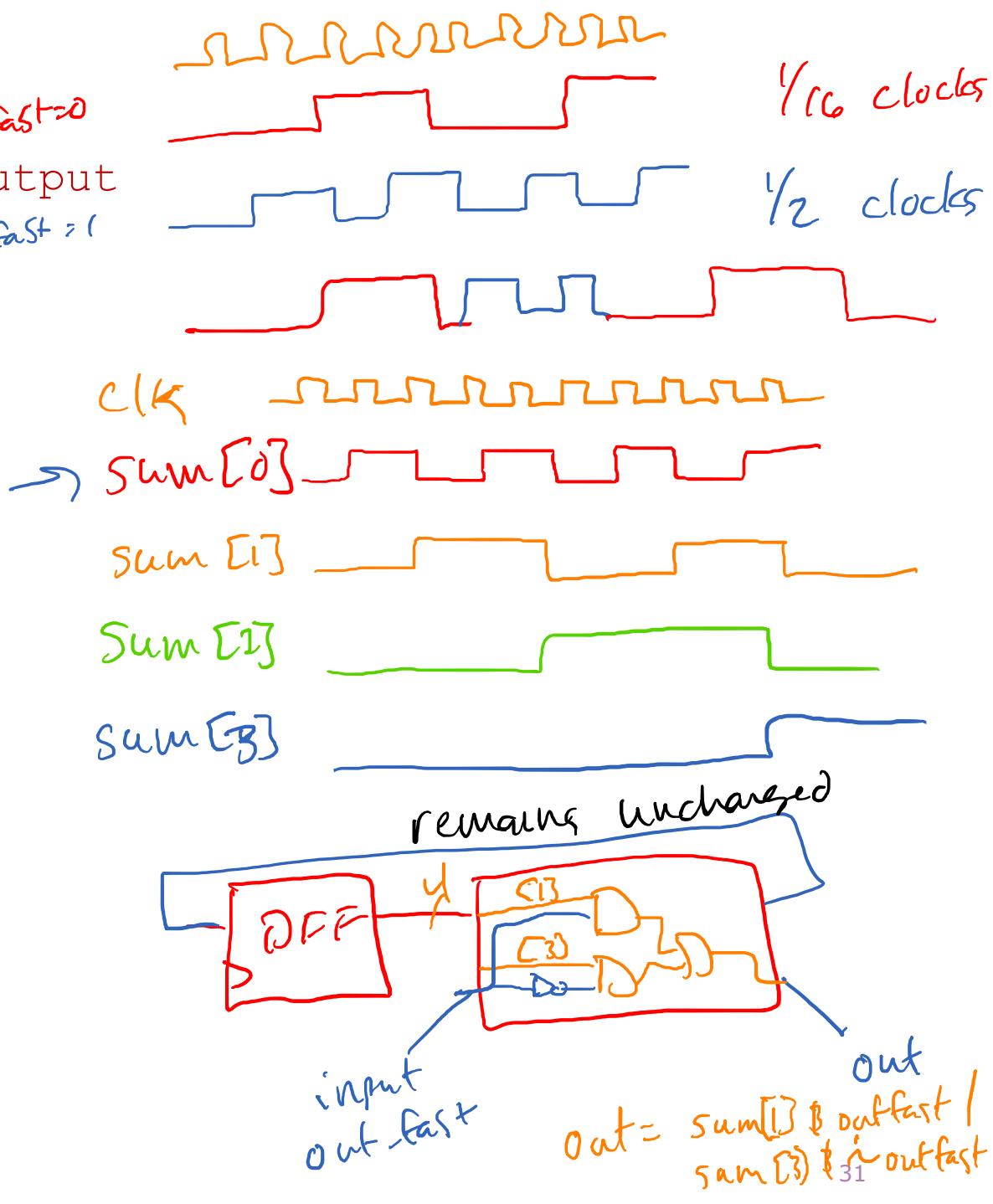
```

module counter(
    input clk, rst
    input          out_fast, //faster output
    output logic   out    //output      out_fast=0
);
    logic [3:0] Q;
    logic [3:0] sum;

    always_ff @(posedge clk) begin
        if (rst) Q <= 4'h0;
        else     Q <= sum;
    end

    always_comb begin
        sum = Q + 4'h1;
        ✓ out = sum[3]; //default
        ✓ if(out_fast) out = sum[i];
    end
endmodule

```



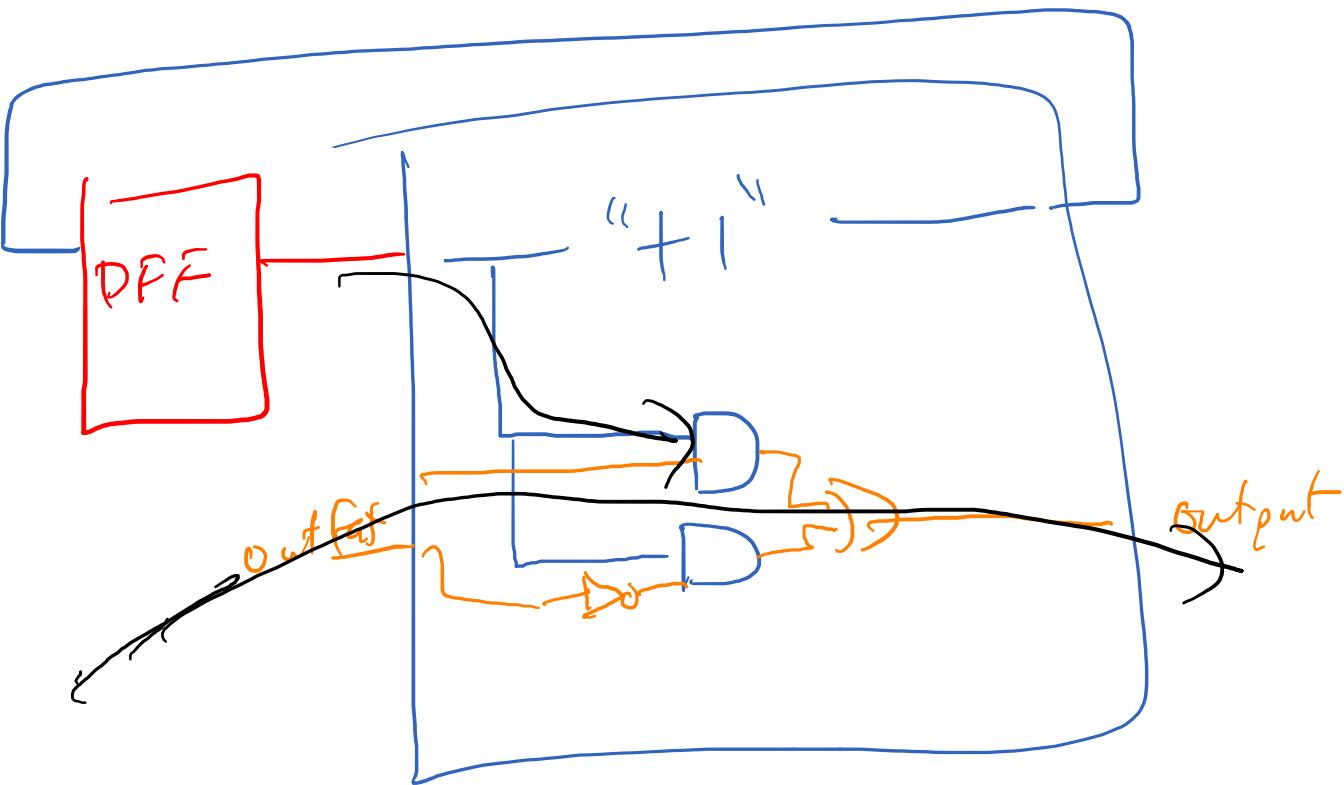
```

module counter(
    input clk, rst
    input           out_fast, //faster output
    output logic     out      //output
);
    logic [3:0] Q;
    logic [4:0] sum;

    always_ff @(posedge clk) begin
        if (rst) Q <= 4'h0;
        else     Q <= sum;
    end

    always_comb begin
        sum = Q + 4'h1;
        out = sum[3];
        if (out_fast) out = sum[1];
    end
endmodule

```



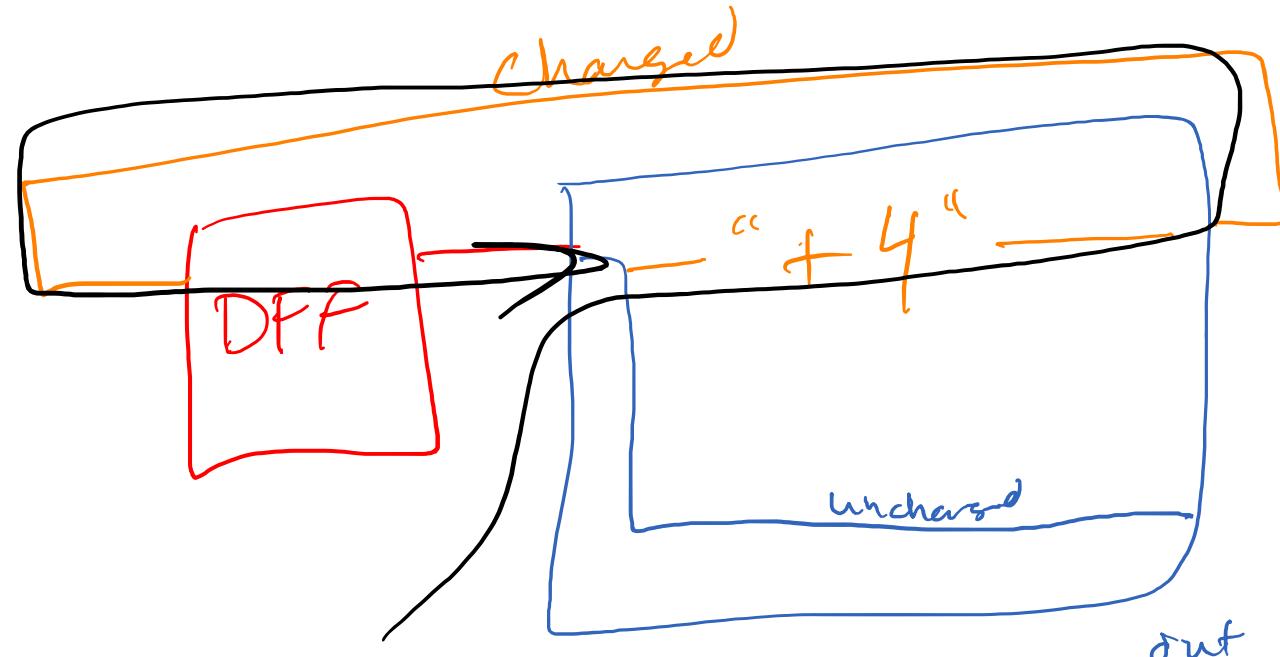
```

module counter(
    input clk, rst
    input          out_fast, //faster output
    output logic    out     //output
);
    logic [3:0] Q;
    logic [4:0] sum;

    always_ff @(posedge clk) begin
        if (rst) Q <= 4'h0;
        else      Q <= sum;
    end

    always_comb begin
        sum = Q + 4'h1;
        out = sum[3];
        if (out_fast) sum = {0,Q} + 5'h4;
    end
endmodule

```



# Next Time

- Finite State Machines (FSMs)