ALU

ENGR 210 / CSCI B441
“Digital Design”

Flip-Flops

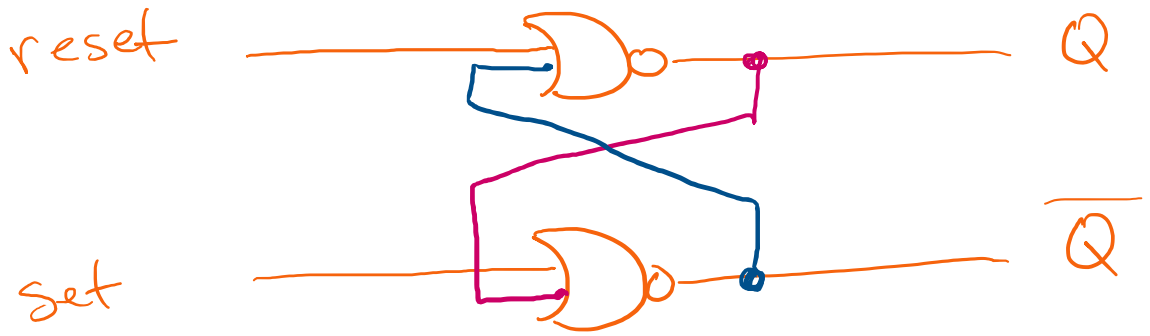
Andrew Lukefahr

Announcements

Password 2-3-5

- P2 due Friday
- P3 will add “demo” requirement. Will need hardware (below).
- Labs/Office Hours
  - Will remain “Virtual” for now
  - You will have access to Luddy 4111
  - You can work from there or from home
- Hardware
  - Still working on that, hopefully pickup on Friday.

Last Time: SR Latch

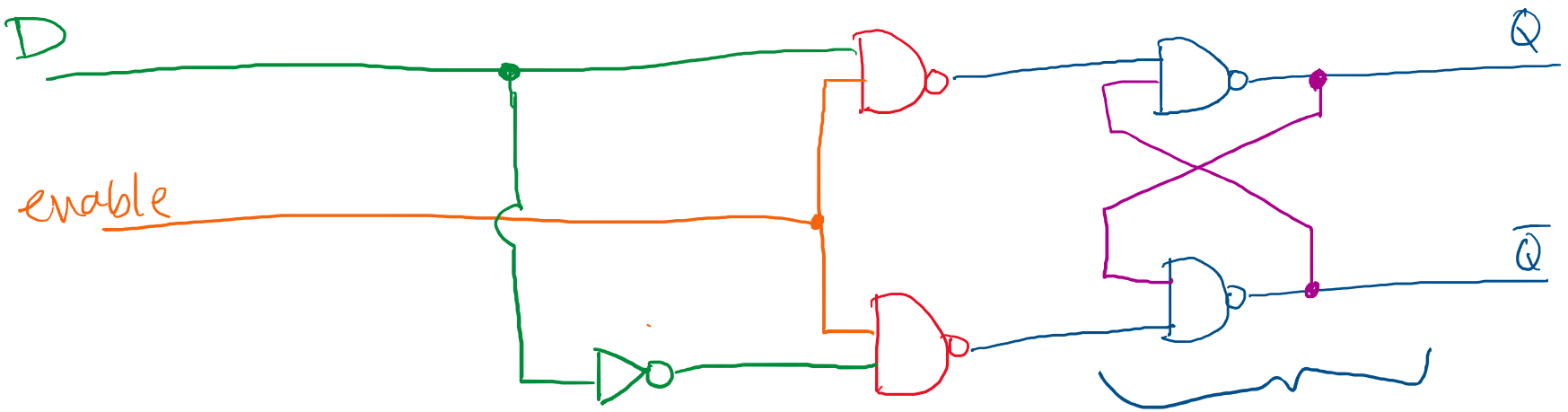


Set	reset	Q	$\overline{Q}$
0	1	0	1
0	0	0	1
1	0	1	0
0	0	1	0

same inputs, different output!

⇒ Internal state!

D-Latch



= 0

= 1

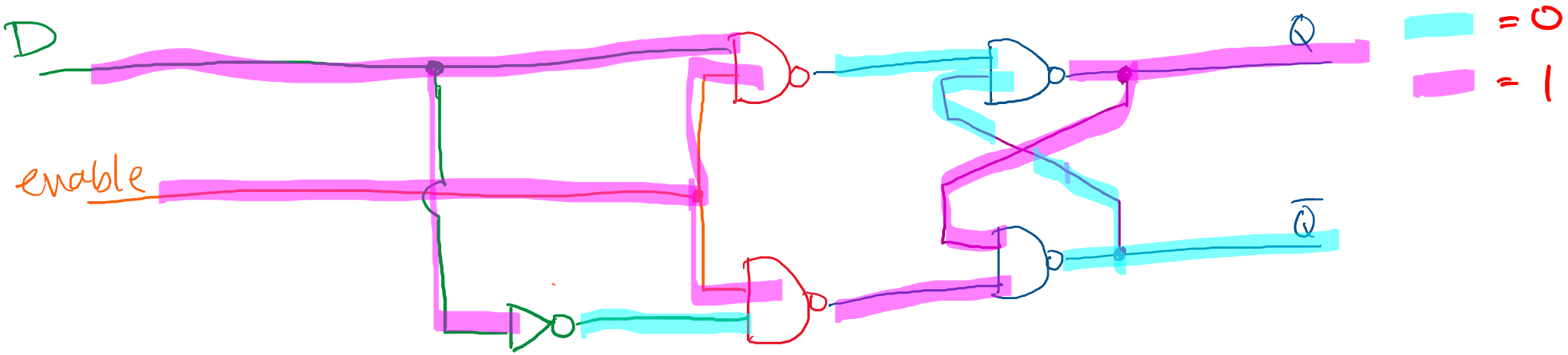
$\overline{SR}$ Latch

SR Latch w/ Enable

D Latch

"Q follows D ~~when~~ when en = 1, otherwise doesn't change"

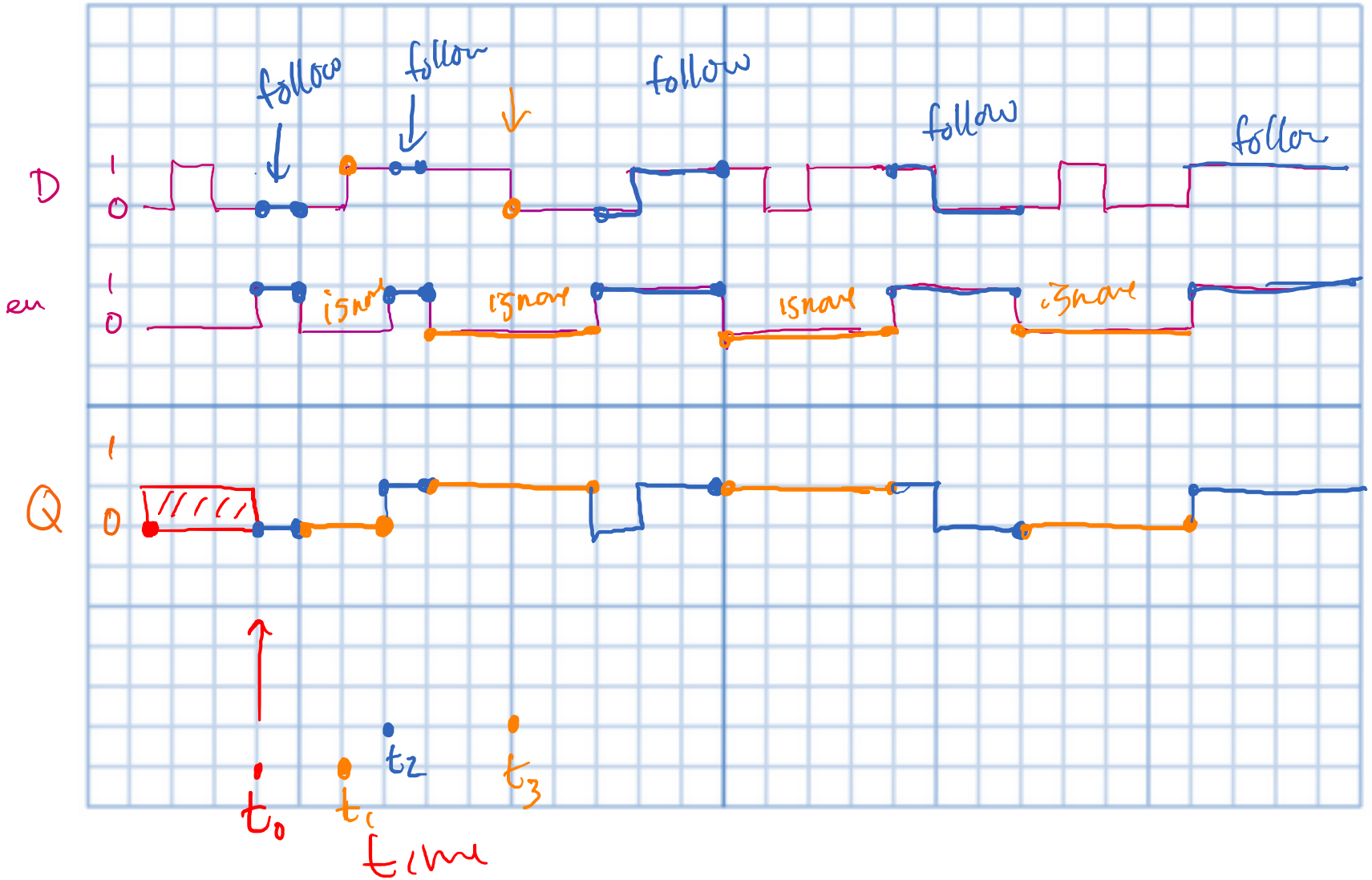
D-Latch



D	enable	Q	$\overline{Q}$
0	0	Q	$\overline{Q}$
0	1	0	1
1	0	Q	$\overline{Q}$
1	1	1	0

"might be on the test!"

Inputs to D Latches

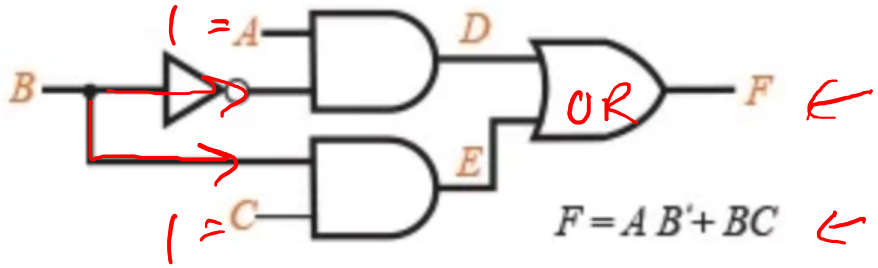


Glitches

→ unintended, short, errors in boolean logic

→ caused by gate delays

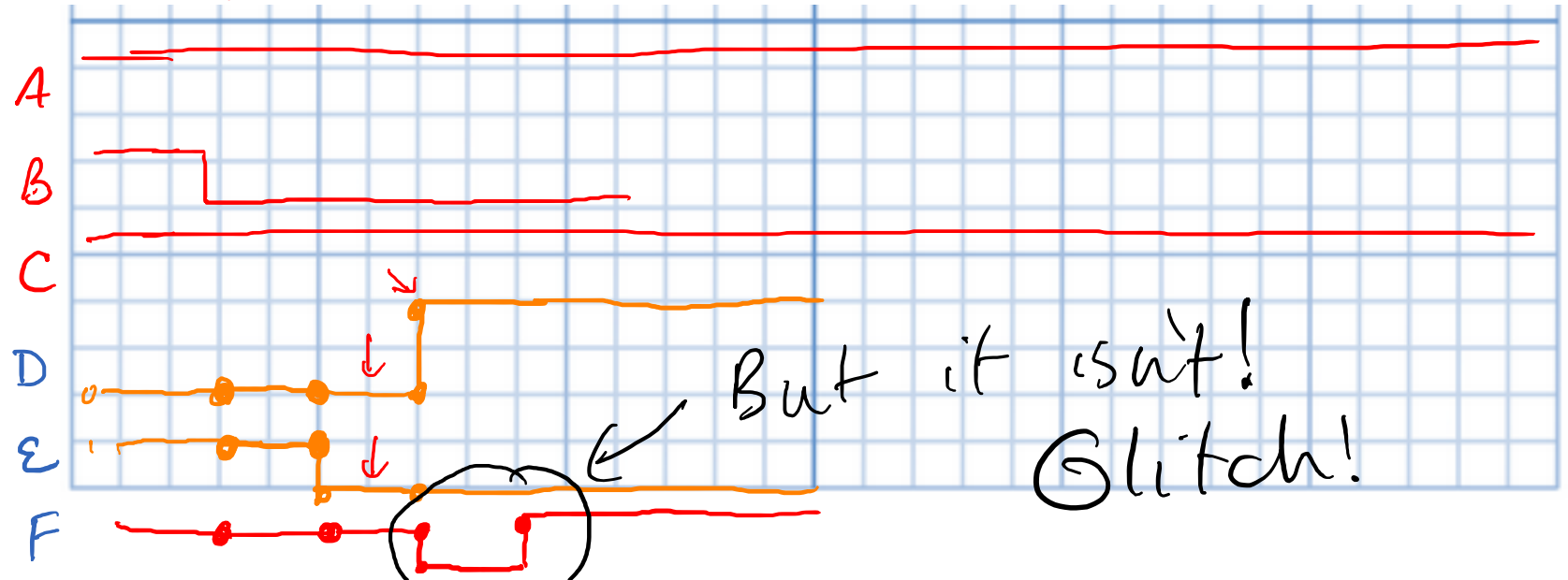
- Assume 10ps / gate.
- A=1, C=1, B falls
- What is F?



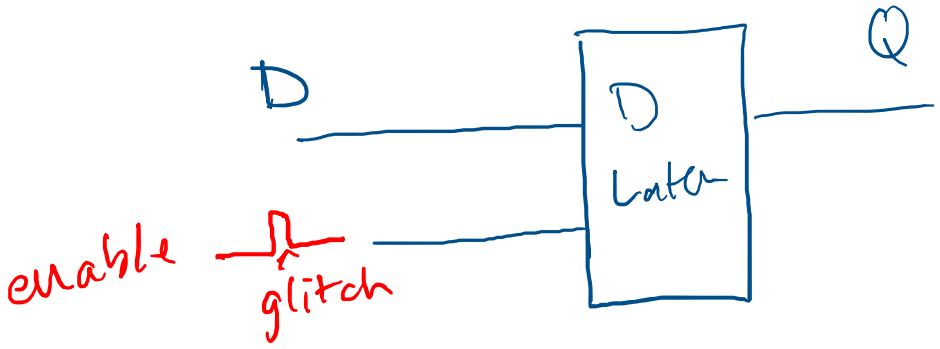
← should always be true!

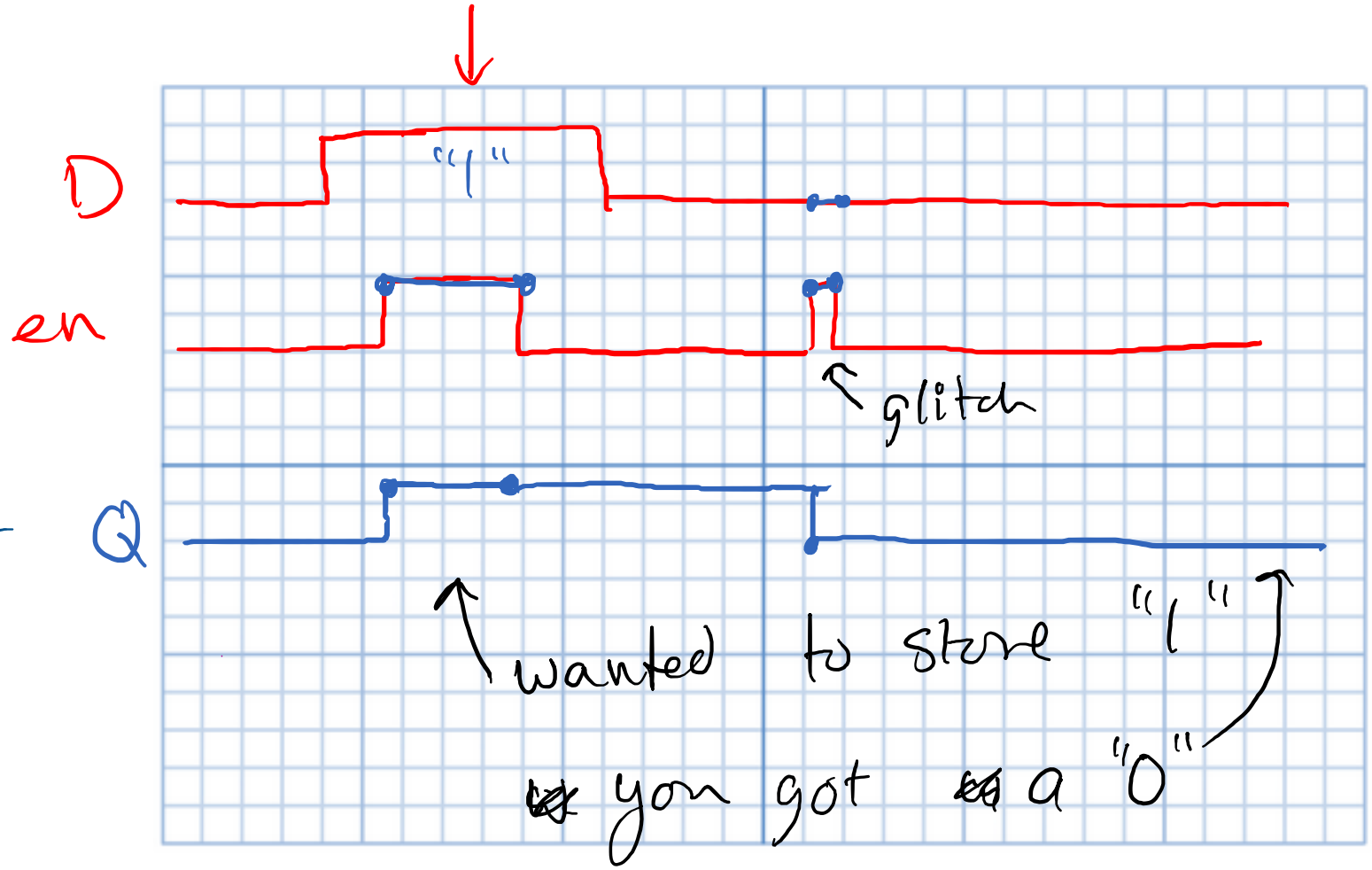
A=1 B=1 C=1

10ps

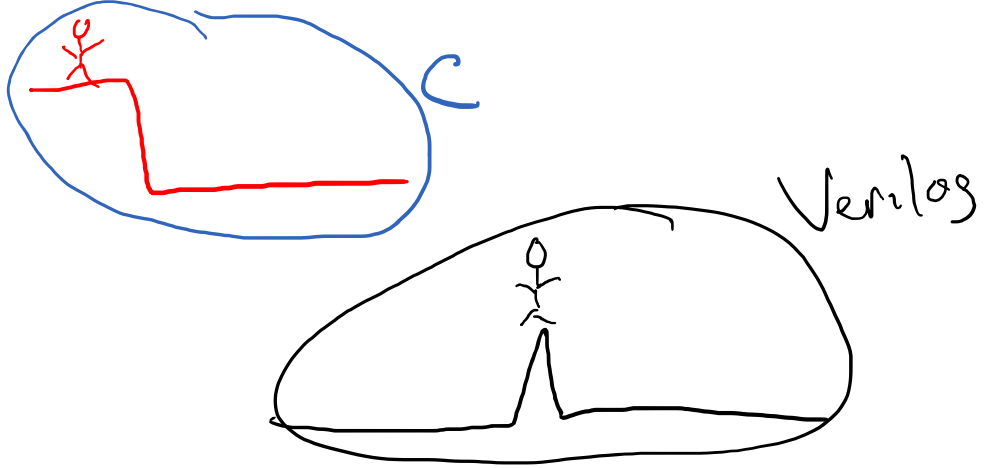


Glitches on D-Latches

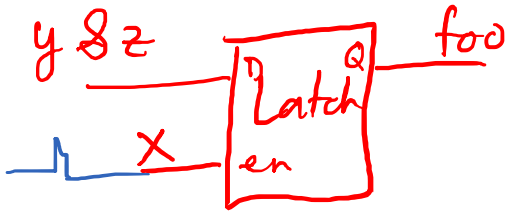




Inferred Latches




```
wire x,y,z;
reg foo, bar ;

always_comb begin
     if (x) foo =  y & z; //bad:
     if (x) bar = y | z;  // what if ~x?
end
```



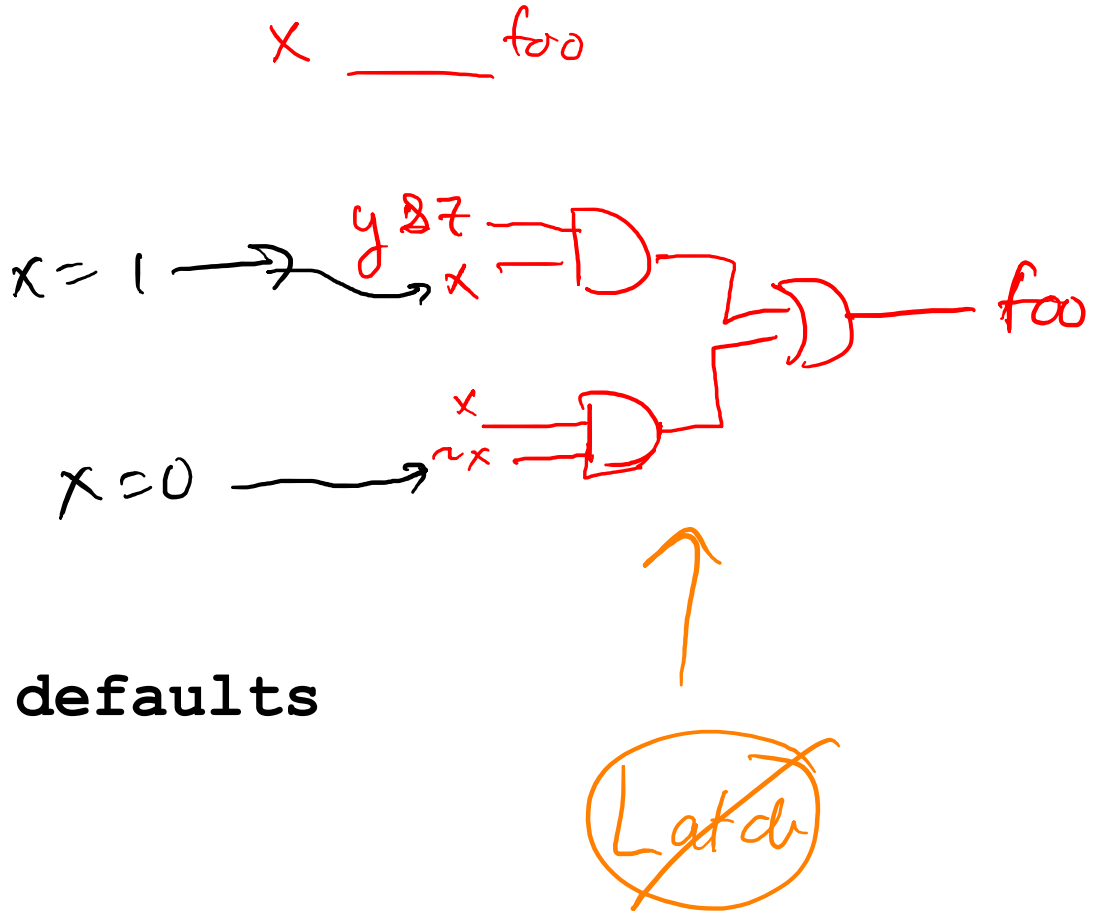
guess Latch?

What if `x == 0`?

Inferred Latches

```
wire x,y,z;
reg foo, bar ;

always_comb begin
     if (x) foo =  y & z; //bad:
     if (x) bar = y | z;  // what if ~x?
end
```

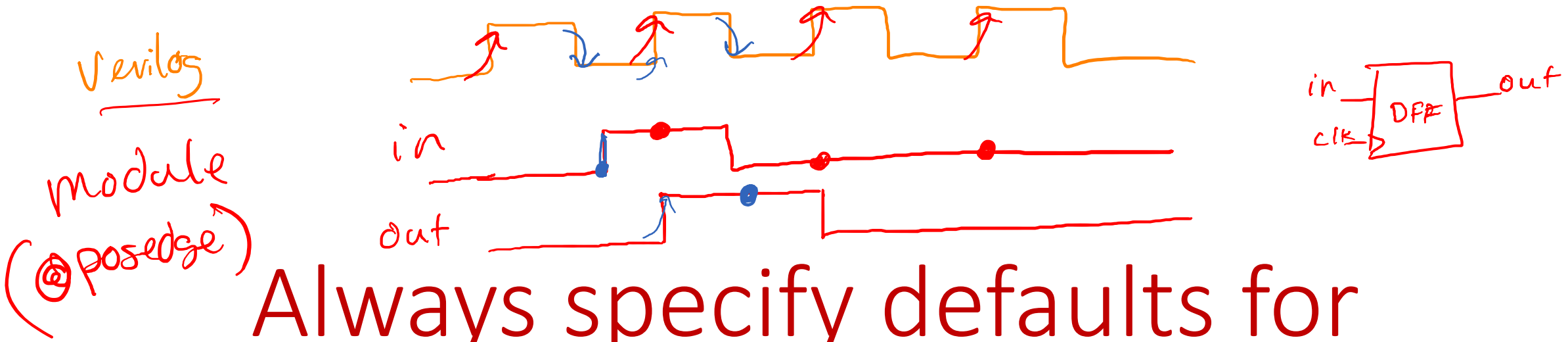
Defaults



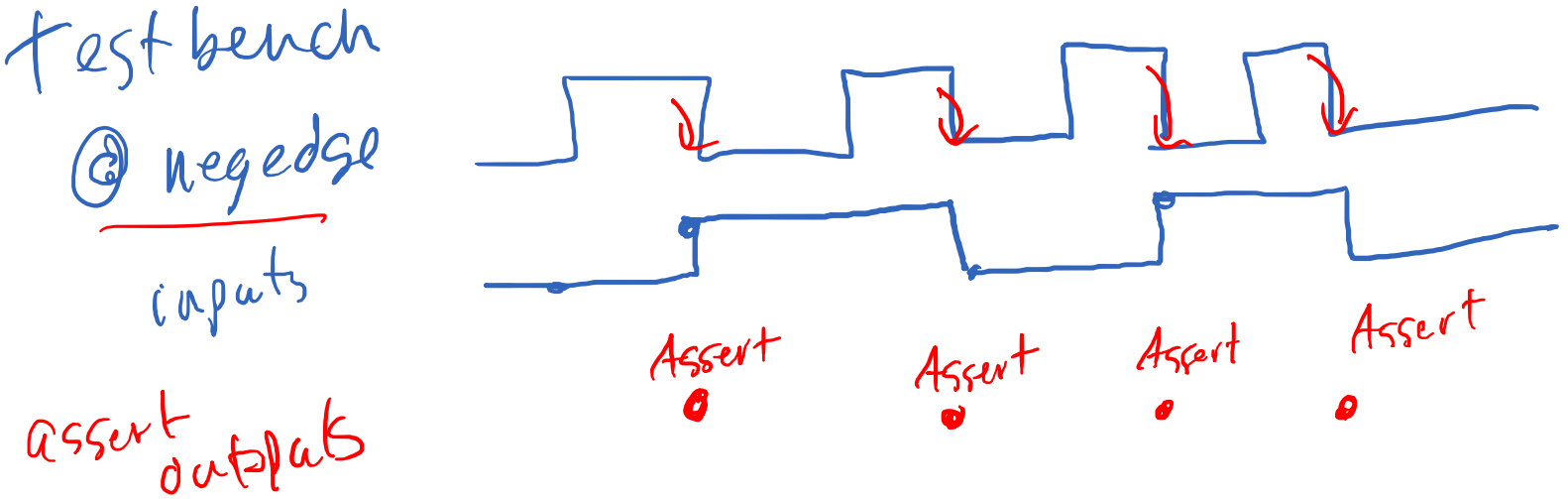

```
wire x,y,z;
reg foo, bar ;

always_comb begin
    foo = x; bar = x; //good: defaults
     if (x) foo =  y & z; //
     if (x) bar = y | z ; //
end
```

What if `x == 0`? `foo = bar = x`!

Always specify defaults for **`always_comb`**!

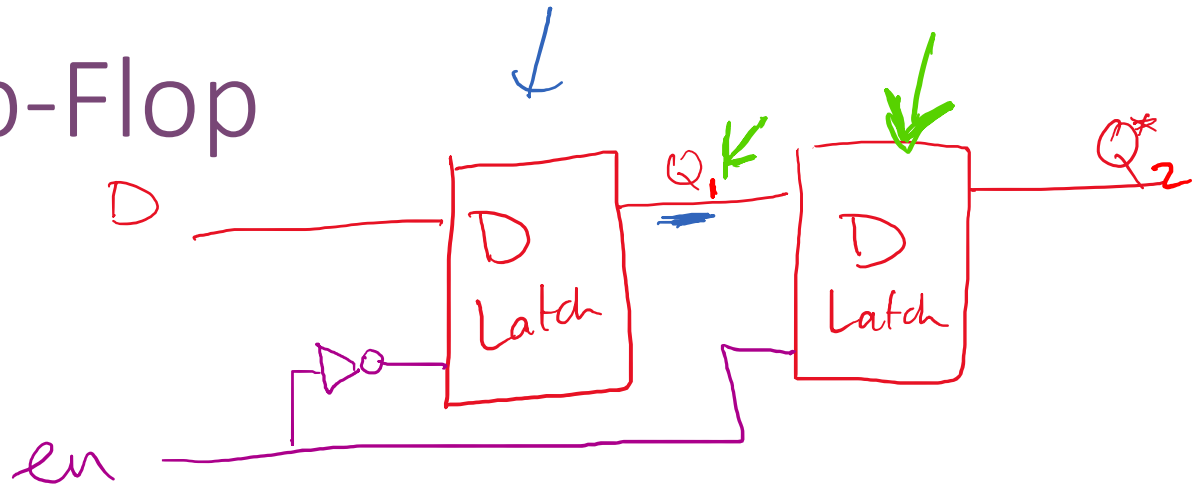


Always specify defaults for **`always_comb`**!



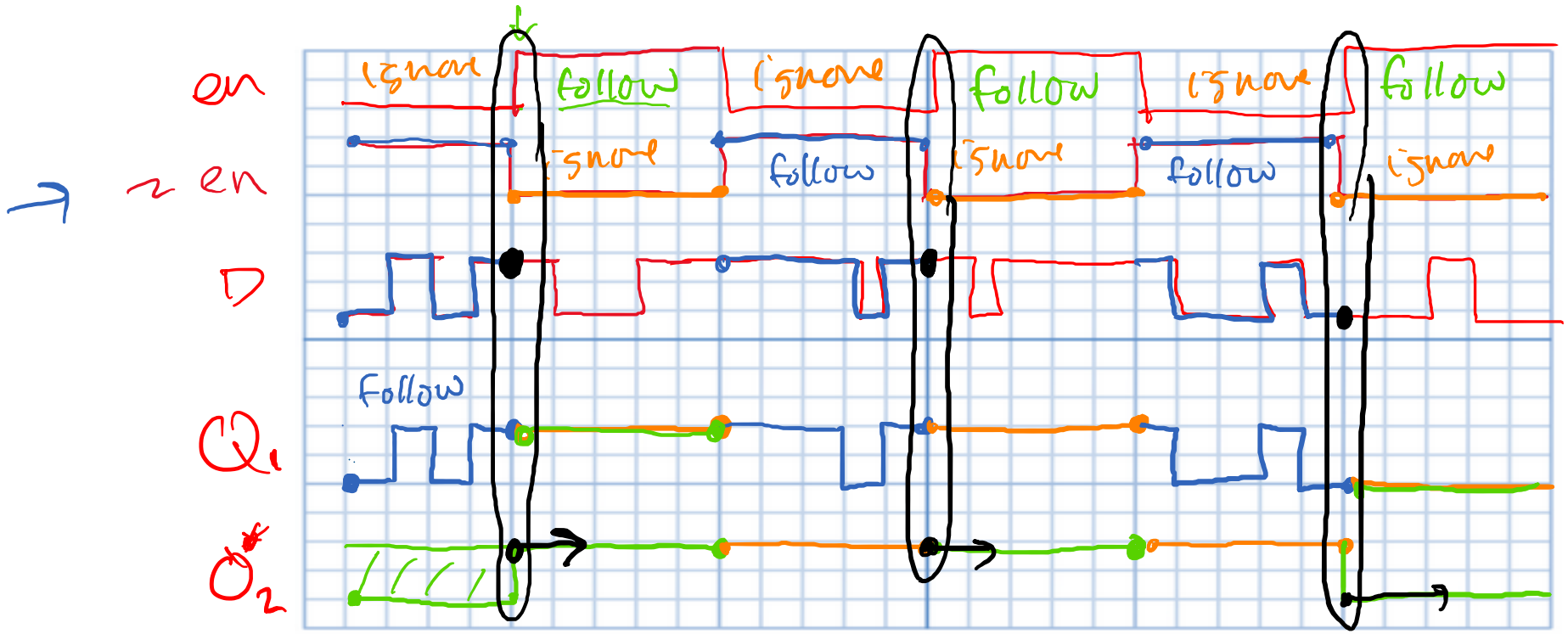
Always specify defaults for `always_comb`!

D Flip-Flop



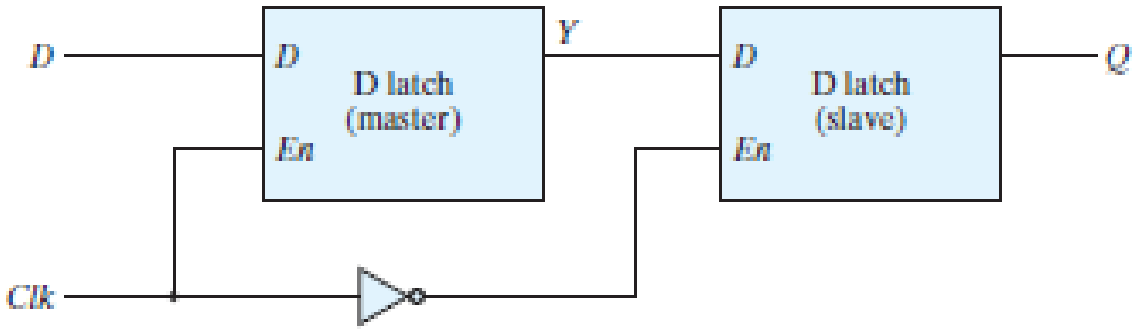
* no gate delays

D Latch:
Q follows D when en = 1



Master-slave D flip-flop

One way to construct a flip-flop is to employ two latches in a special configuration that isolates the output of the flip-flop and prevents it from being affected while the input to the flip-flop is changing.

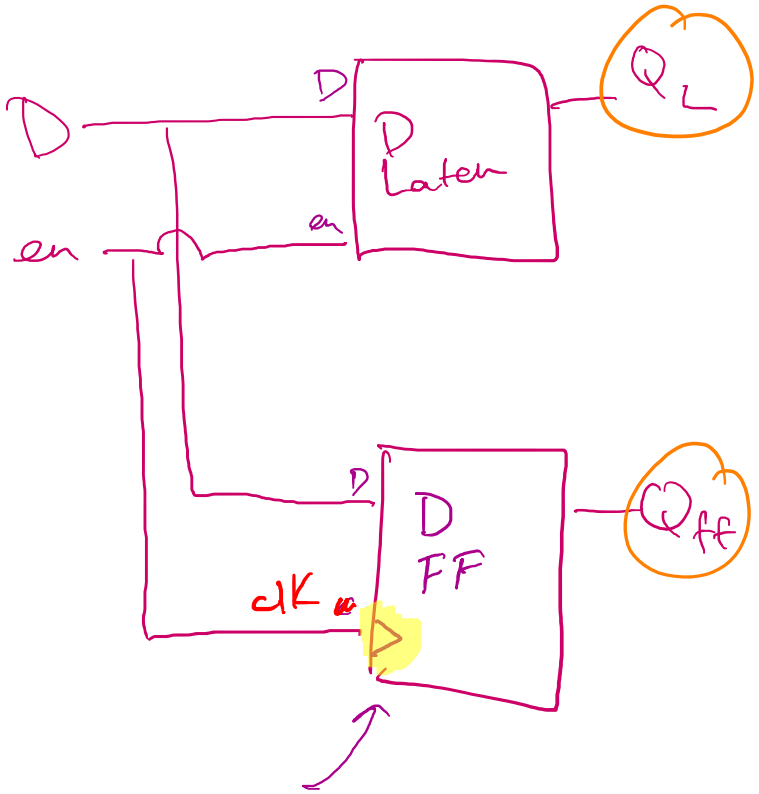


neg edge!

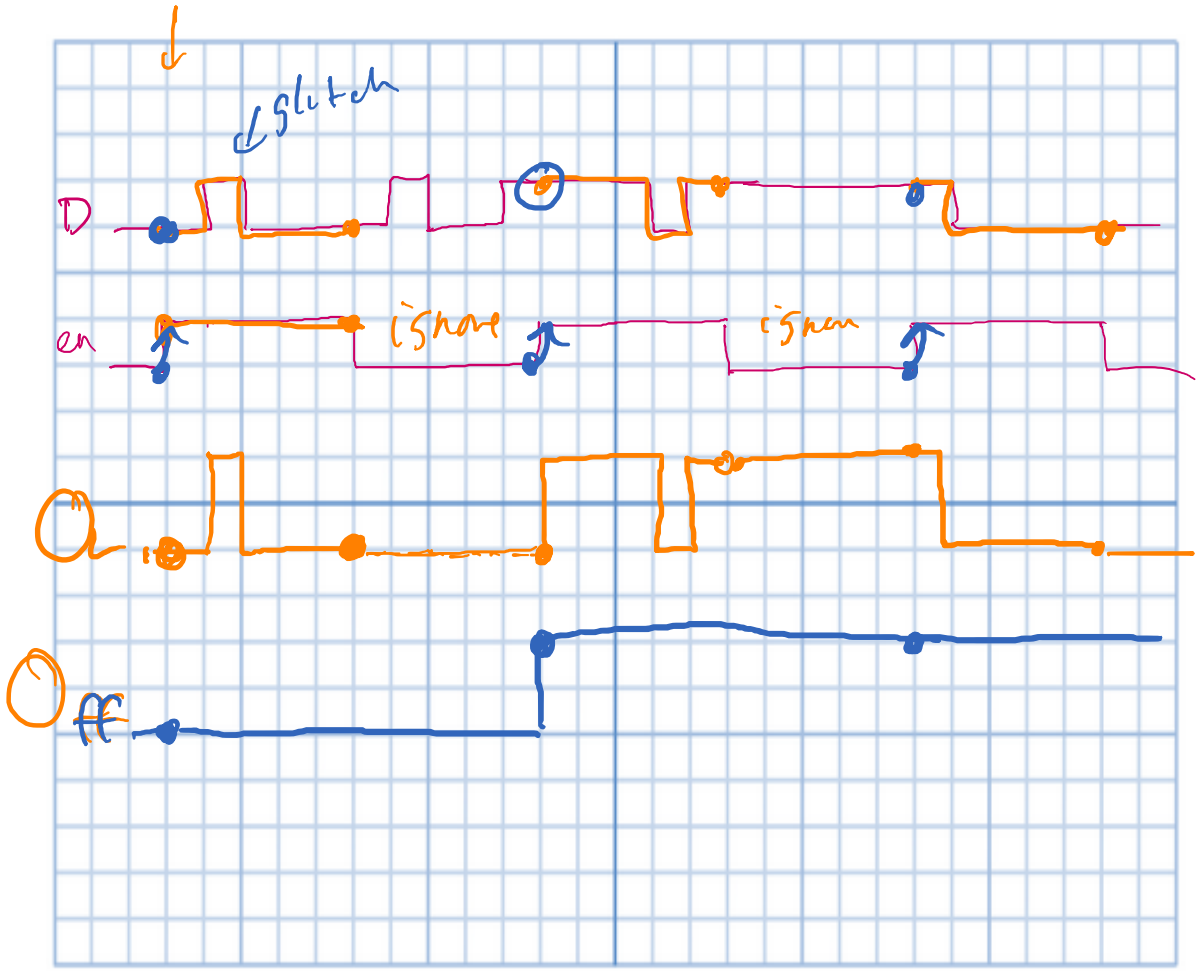
When the *Clk* is at the logic-1 level, the data from the external *D* input are transferred to the master. The slave is disabled and any change in the input changes the master output at *Y*, but cannot affect the slave output.

When the *Clk* returns to 0, the master is disabled and is isolated from the *D* input. At the same time, the slave is enabled and the value of *Y* is transferred to the output of the flip-flop at *Q* . Thus, a change in the output of the flip-flop can be triggered only by and during the ***transition of the clock from 1*** **to 0**.

D Flip-Flop vs. D Latch

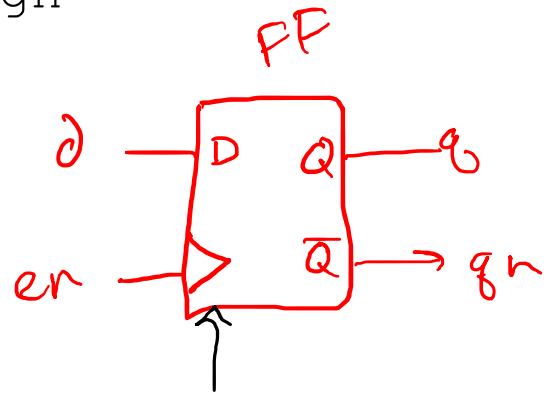


the " > " symbol tells you it is Flip-flop



D Flip-Flop in Verilog

```
module d_ff (
    input d,              //data
    input en,             //enable
    output reg q          //reg-isters hold state
);

    always_ff@(posedge en ) //pos-itive edge of en-able
        begin
                q <= d; //non-blocking assign
                qn<= ~d; //optional

        end

endmodule
```

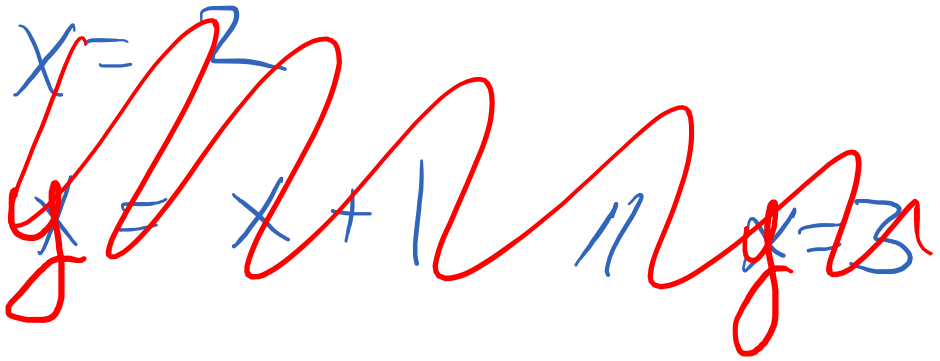


D Flip-Flop w/ Clock

```
module d_ff (
    input d,              //data
    input clk,            //clock
    output reg q          //reg-isters hold state
);

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

endmodule
```

Blocking vs. NonBlocking Assignments

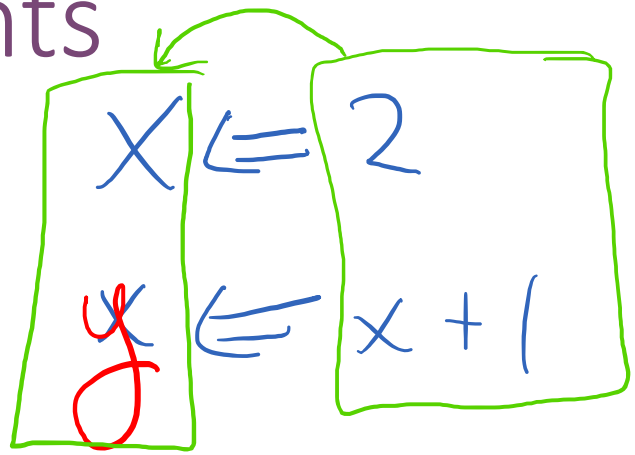
- Blocking Assignments (= in Verilog)
  - Execute in the order they are listed in a sequential block;
  - Upon execution, they immediately update the result of the assignment before the next statement can be executed.



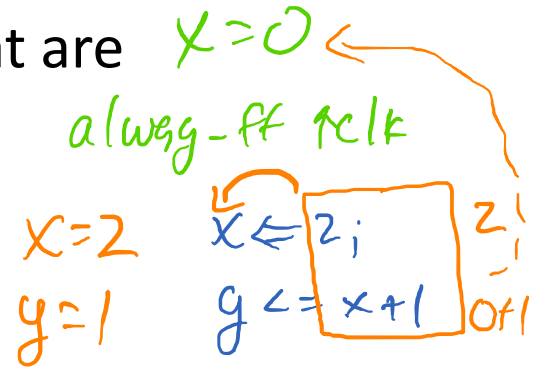
```
x = 2;
y = x + 1;
```

→ x = 2
y = 3

Blocking vs. NonBlocking Assignments



- Non-blocking assignments (<= in Verilog):
  - Execute concurrently
  - Evaluate the expression of all right-hand sides of each statement in the list of statements before assigning the left-hand sides.
  - Consequently, there is no interaction between the result of any assignment and the evaluation of an expression affecting another assignment.
  - Nonblocking procedural assignments be used for all variables that are assigned a value within an edge-sensitive cyclic behavior.



Blocking vs. NonBlocking

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

LHS RHS

bad in always_comb

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

1+1 0+1 0+1

1+1=2 2+1=3 1+1=2

start x=0, y=0 z=0, a=0

a=1

x = 1+1 = 2

y = 2+1 = 3

z = 0+1 = 1

x=2, z=1

x = 2;
y = 3;
z = 1+1 = 2

x=2
y=3
z=2+1=3

start: x=0, y=0 z=0, a=0

a=1, clk↑

x = 2
y = 1
z = 1

clk↑

x=2
y=3
z=2

Blocking vs. Non-Blocking Assignments

- ONLY USE BLOCKING (=) FOR COMBINATIONAL LOGIC
  - `always_comb`
- ONLY USE NON-BLOCKING (<=) FOR SEQUENTIAL LOGIC
  - `always_ff`
- Disregard what you see/find on the Internet!

BLOCKING (=) FOR always_comb

never hold state

+ defaults!

No Flip Flops!

NON-BLOCKING (<=) for always_ff

always for Flip Flops (always hold state)

D-FlipFlop w/Clock

$q \rightarrow d \rightarrow q_{new} \rightarrow d_{new} \rightarrow q_{new_2}$

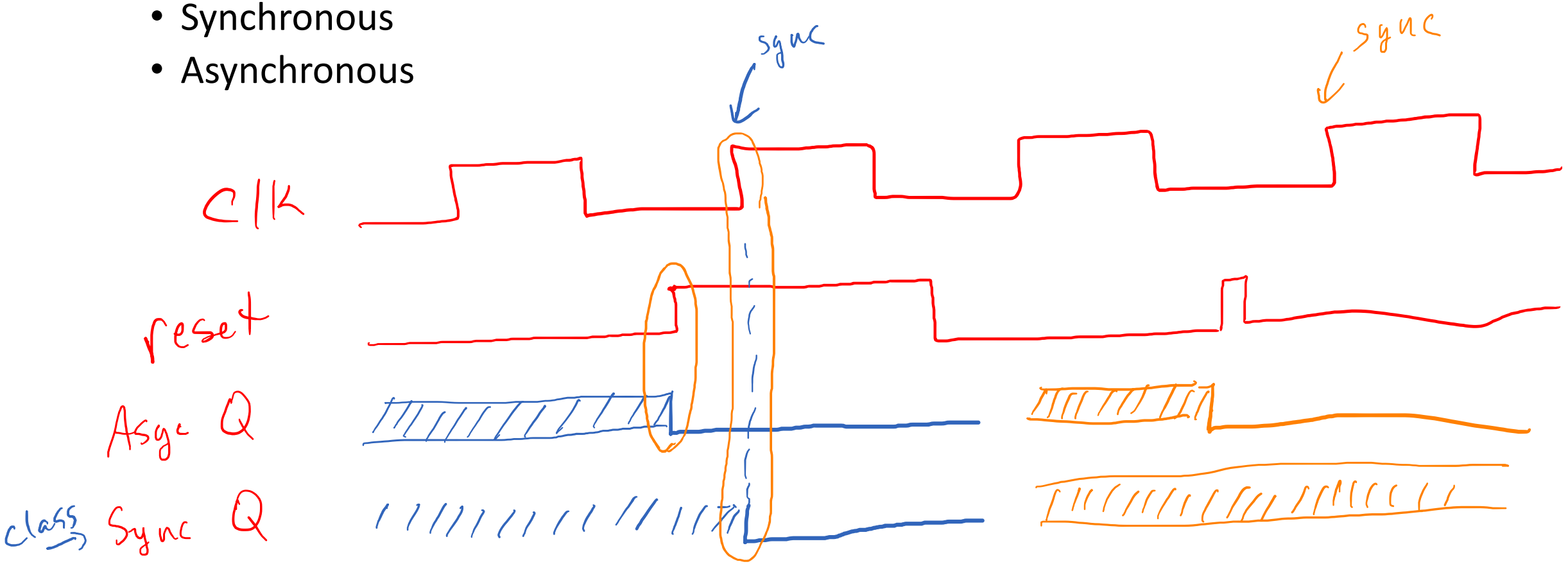
```
module d_ff (
     input d,              //data
     input clk,            //clock
     output reg q          //reg-isters hold state
);

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

endmodule
```

logic

What is q before posedge clk?

D-FF's with Reset

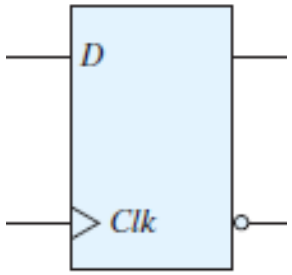
- Two different ways to build in a reset
  - Synchronous
  - Asynchronous



D-FF’s with Reset

- Two different ways to build in a reset
  - Synchronous
  - Asynchronous

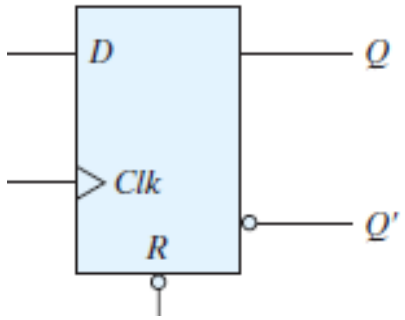
Verilog models of D flip-flop



Edge triggered D flip-flop:

```
logic Q;
always_ff @ (posedge clk)
  Q <= D;
```

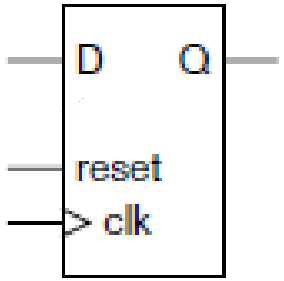
No reset ff



Edge triggered, asynchronous reset D flip-flop:

```
logic Q;
always_ff @ (posedge clk, negedge rst)
  if (~rst) Q <= 1'b0;  //asynch. reset
  else Q <= D;
```

Not used in class



Edge triggered, synchronous reset, clock enable D flip-flop:

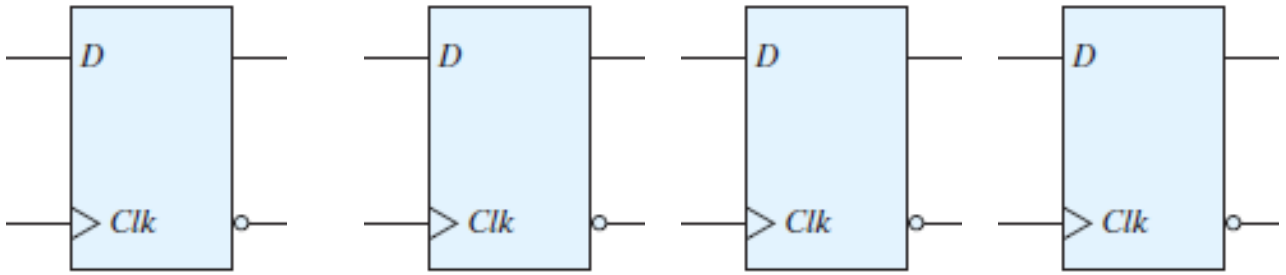
```
logic Q;
always_ff @(posedge clk)
  if (reset) Q <= 1'b0; // synch. reset
  else Q <= d;
```

Blocking Procedural Assignments

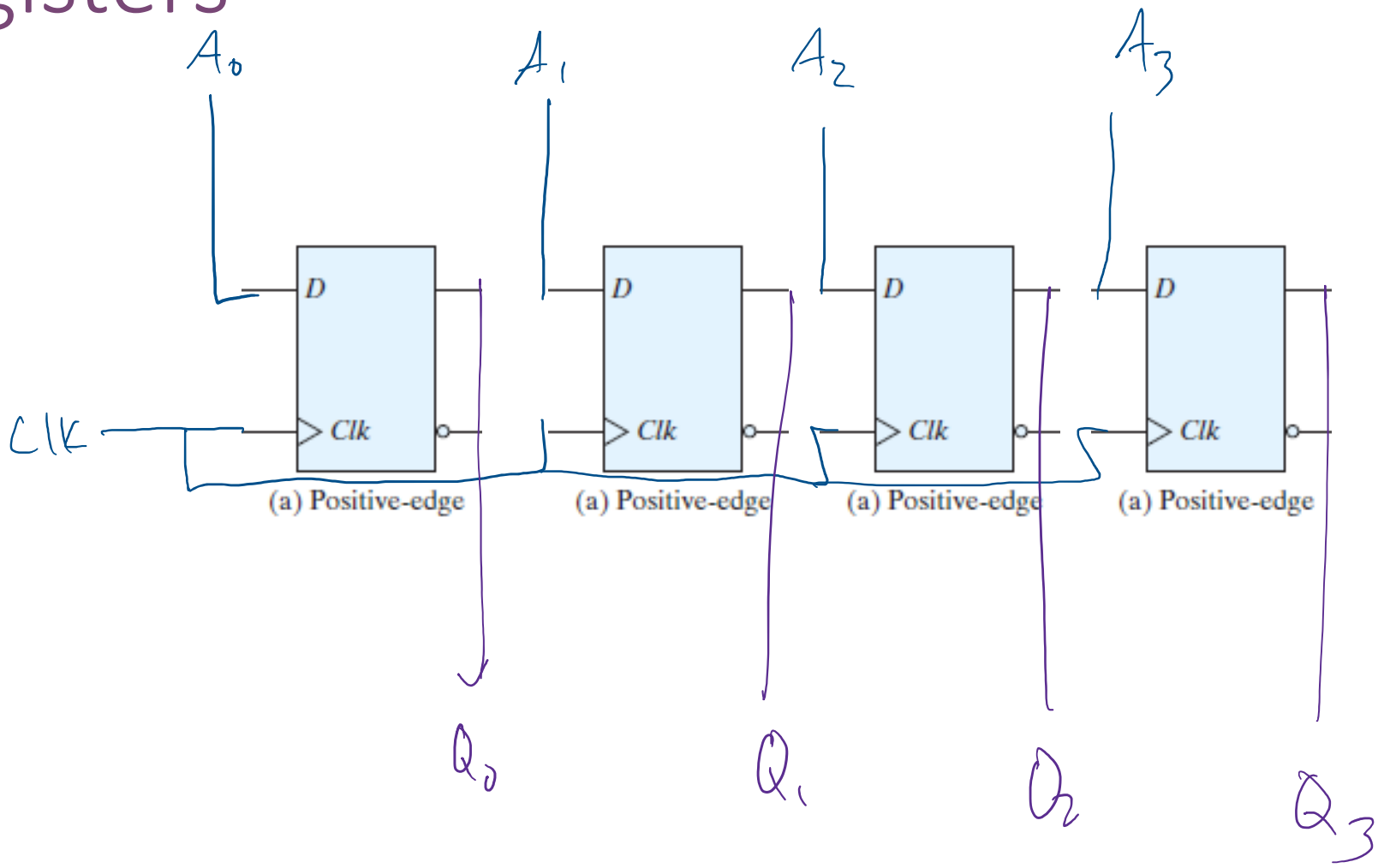
Blocking procedural assignments (symbol =):

- Execute sequentially in the order that they are listed in a sequential block;
- When they execute, they have an immediate effect on the contents of memory before the next statement can be executed.
- Used to model level sensitive behavior, such as combinational logic.

Registers


D
Clk
(a) Positive-edge
D
Clk
(a) Positive-edge
D
Clk
(a) Positive-edge
D
Clk
(a) Positive-edge

Registers



registers in a CPU?

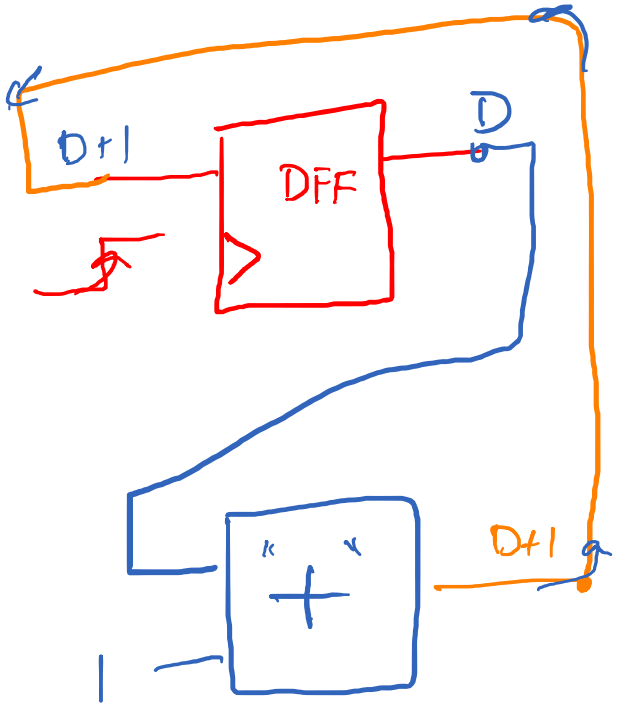
4-bit Register in Verilog

```
module d_ff (
    input               d,    //data
    input               clk,  //clock
    output              q     //output register
);

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

endmodule
```

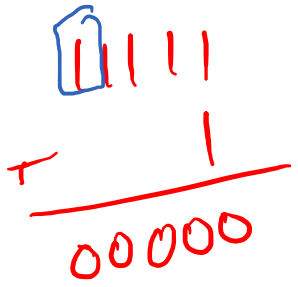
4-bit Register in Verilog

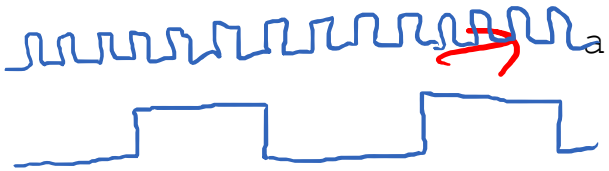
```
module d_ff (
    input       [3:0]   d,   //data
    input               clk, //clock
    output logic [3:0] q   //output register
);

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

endmodule
```

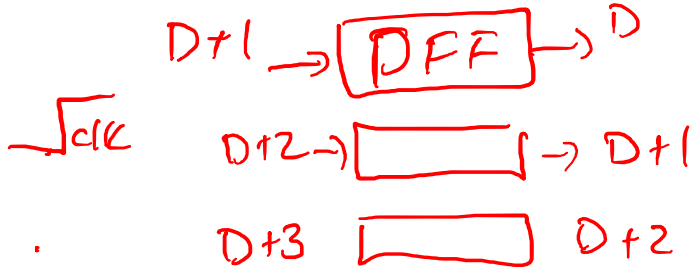
What does this module do?




```
module mystery(
    input clk,          //clock
    input rst,          //reset
    output logic out    //output
);
    logic [3:0] D;
    wire [4:0] sum;

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

    always_comb // <- combinational logic
    {out,sum} = {0,D} + 5'h1;  //blocking

endmodule
```



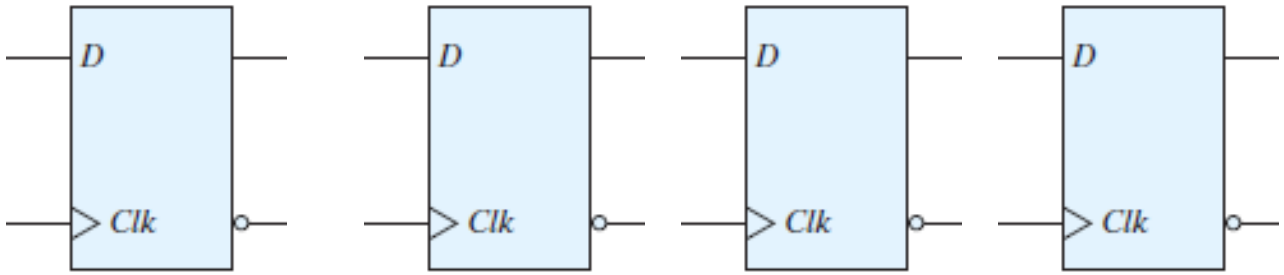




1bit 4bits 5bit + 5bit

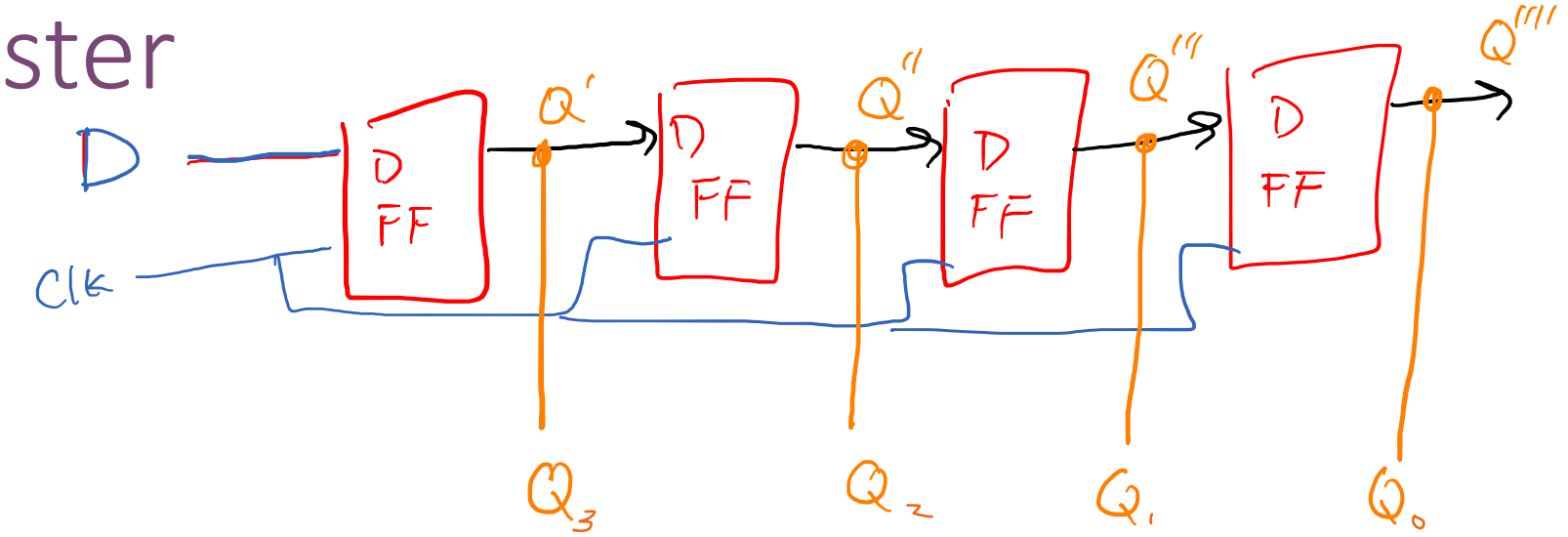
Start here!

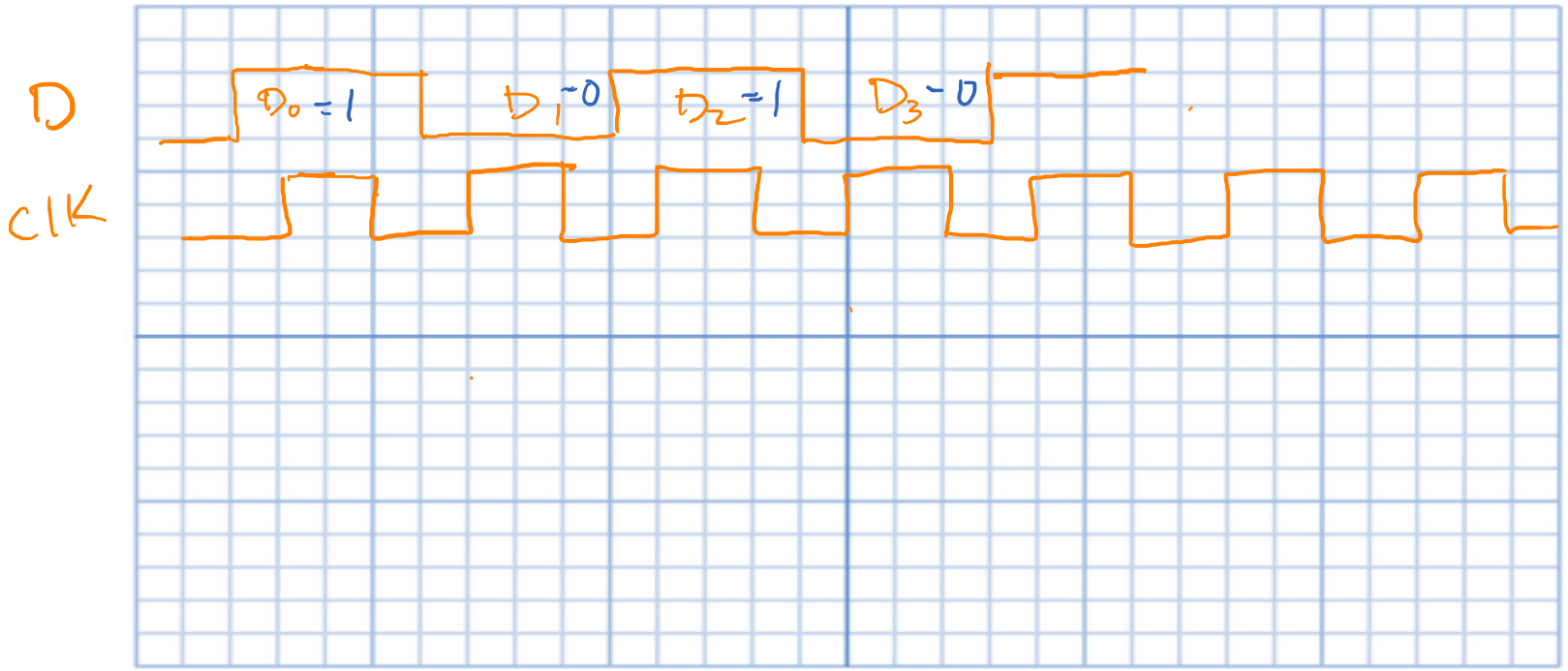
D Flip-Flops as Shift Registers



(a) Positive-edge (a) Positive-edge (a) Positive-edge (a) Positive-edge

Shift Register





Next Time

- State machines