P6 ALU

→ hard bugs
BUG 9

→ ALU was easy
TB's were hard

Fix Website
Dates

→ dont use # delays in your comb logic

ENGR 210 / CSCI B441

Latches + Flip Flops

Andrew Lukefahr

Announcements

P7 pushed back a week

- P7 Saturating Counter is out
- P8 – Elevator Controller is out
  - This one is hard.

* P9 SPI is out

* Wednesday is Exam Review

UPDATE: 'wire' vs 'logic'

SystemVerilog (NEW) Rules:

Just use 'logic'*

* ***EXCEPT***

```
logic foo = a & b;  (BAD - Initial values of a & b only)

wire foo = a & b;   (OK)

logic foo;
assign foo = a & b; (OK)
```

always_comb with case

```
module decoder (
        input [1:0] sel,
        output logic [3:0] out
        );

    always_comb begin
        out = 4'b0000; //default
        case(sel)
                2'b00: out=4'b0001;
                2'b01: out=4'b0010;
                2'b10: out=4'b0100;
                                    // what about sel==2'b11?
        endcase
    end

endmodule
```

Always specify defaults for **always_comb**!

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

Always specify defaults for `always_comb`!

Combinational Logic
→ not on time
→ a "combination" of inputs
→ and/or/NOT
→ output is combination of inputs

SEQUENTIAL LOGIC

→ sequence / time
→ output is combination of current inputs and past inputs

DFF: store a value

Sequential vs. Combinational

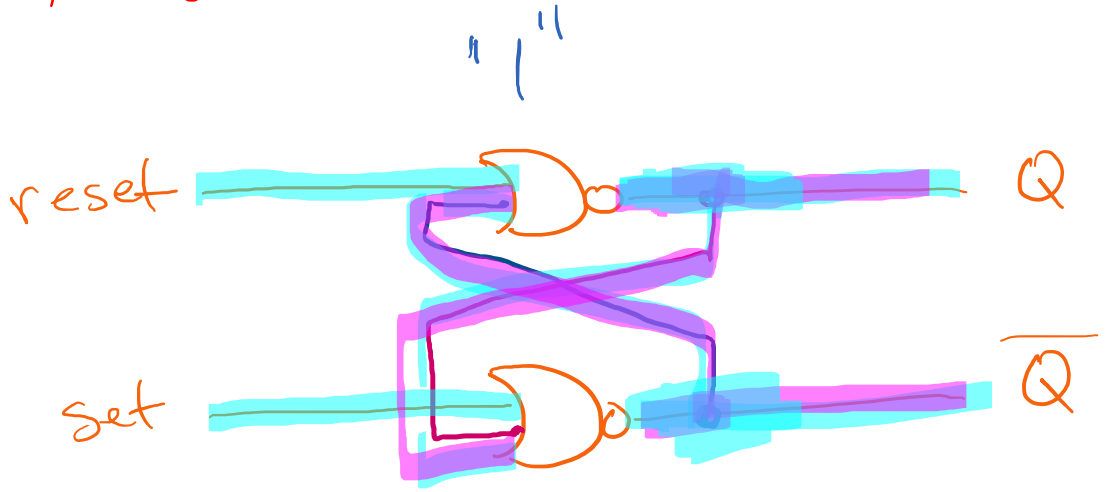
- Combinational Logic
  - The output is a combination of the current inputs only
- Sequential Logic
  - The output is a combination of the current **and past** inputs

SR Latch

NOR

A	B	Z
0	0	1
0	1	0
1	0	0
1	1	0

"Set-Reset Latch"



= 1

= 0

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

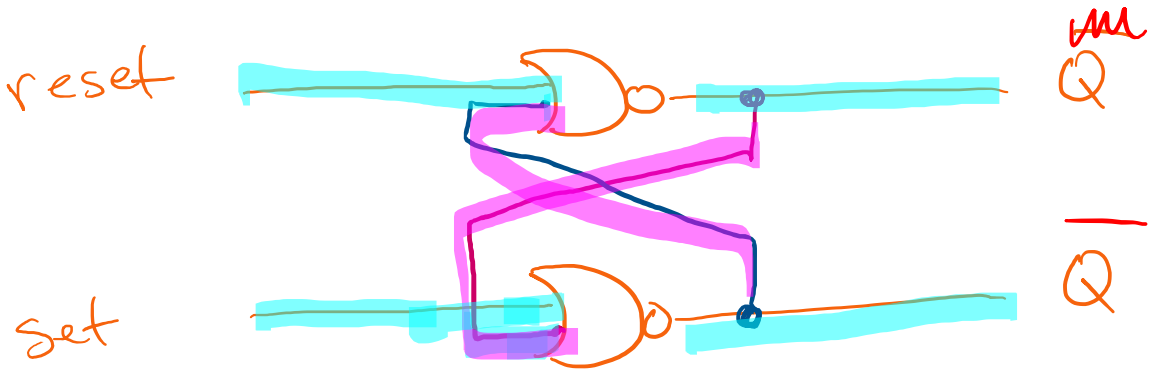
same inputs / different outputs

SR Latch w/ S=1 & R=1

Don't do this!

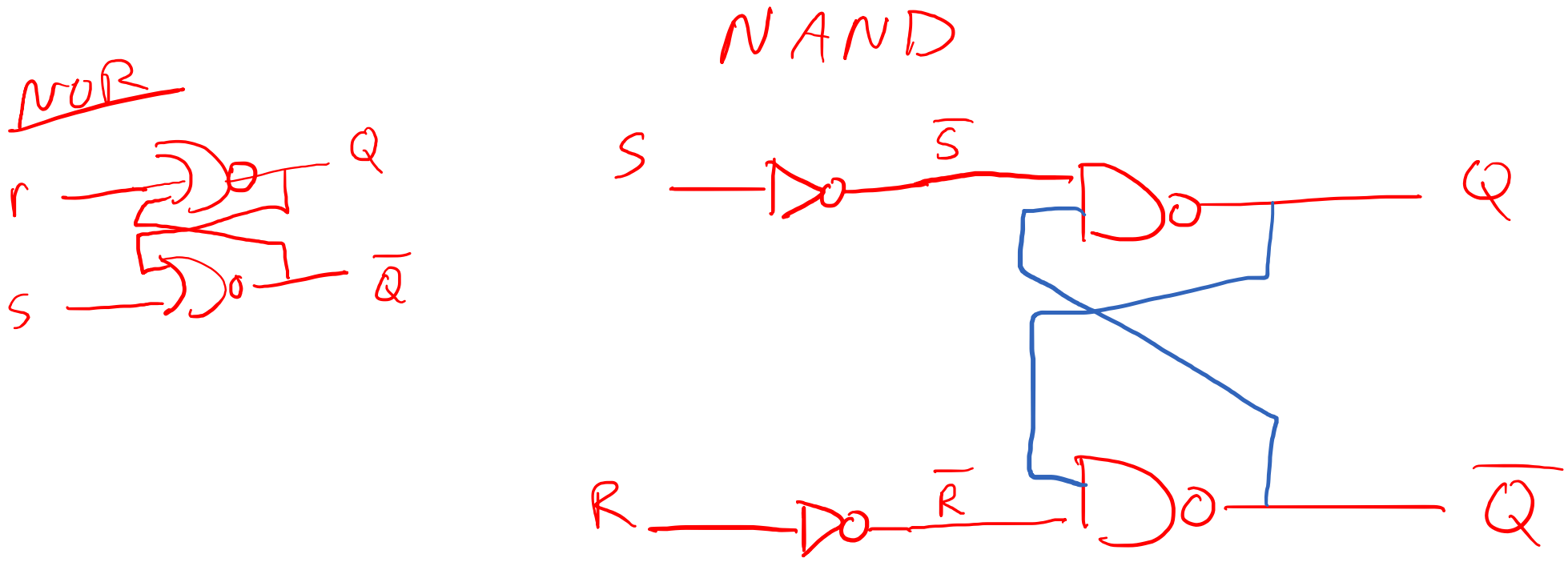
=1

=0



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

SR Latch w/NAND gates

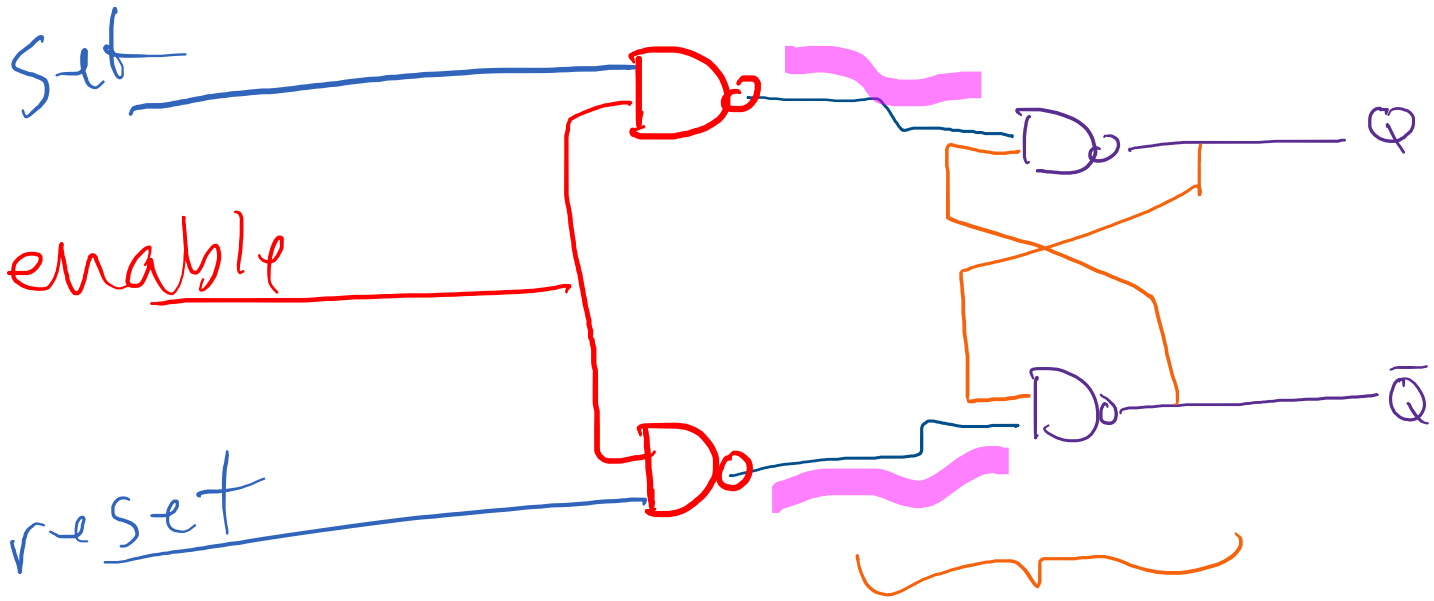


→ better setup for ~~Flip~~ Flip-Flops

→ easier for me to draw

SR Latch with Enable

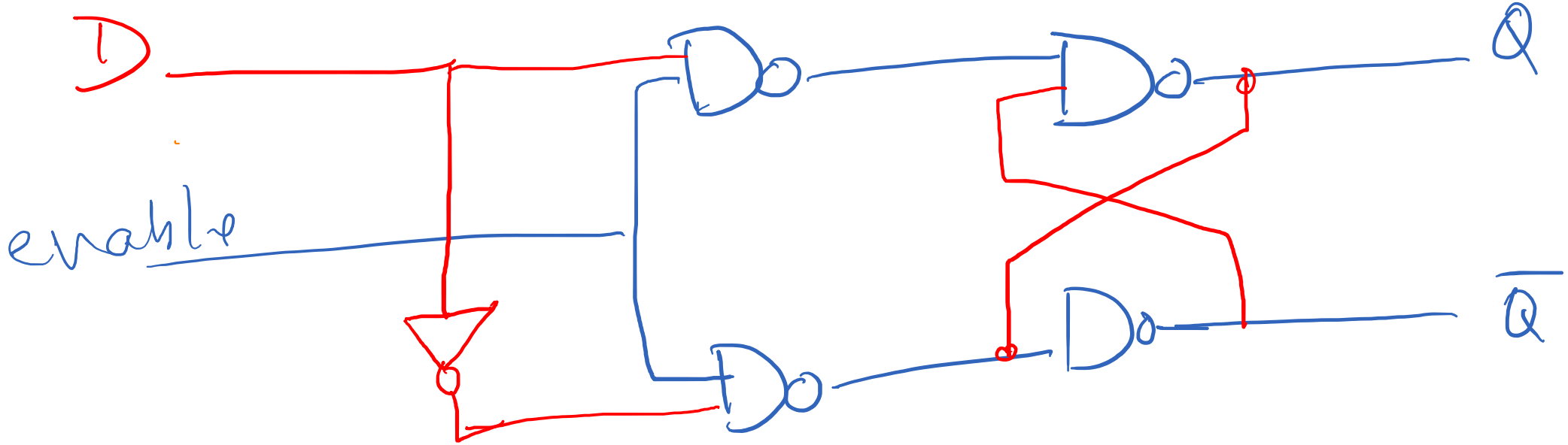
Prevent changes in S & R from changing circuit output



S	R	E	Q	$\overline{Q}$
x	x	0	Q	$\overline{Q}$
1	0	1	1	0
0	1	1	0	1

* assume no S=1 & R=1

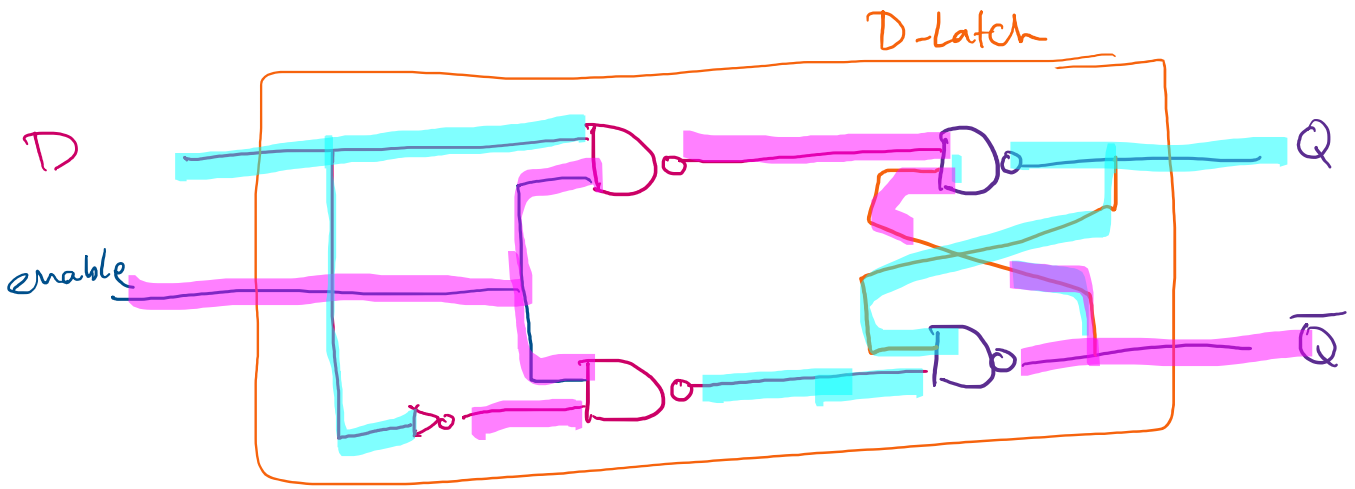
D-Latch "Data Latch"



D-Latch

= 1

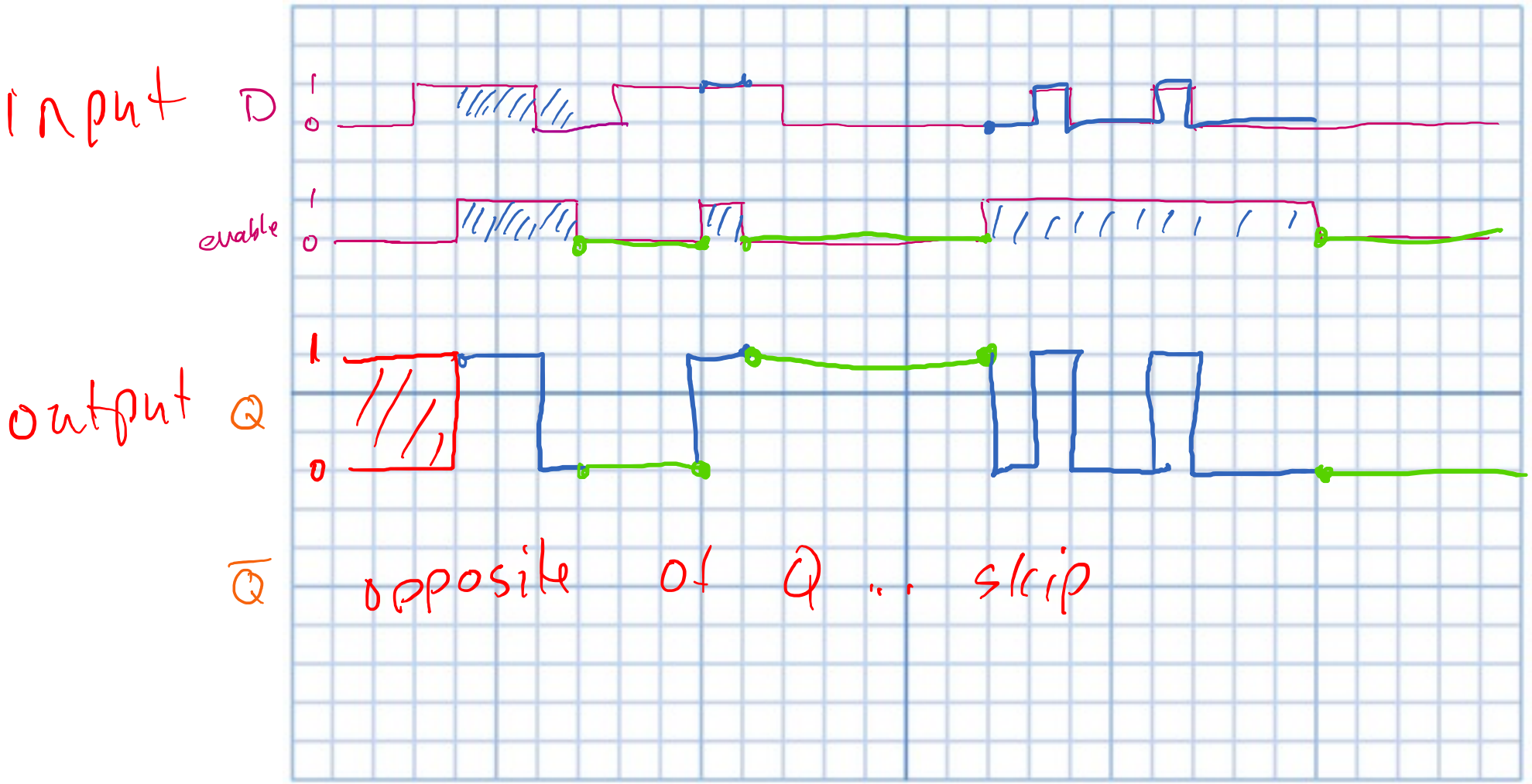
= 0



D	en	Q	$\overline{Q}$
–	0	0	1
1	1	1	0
0	1	0	1

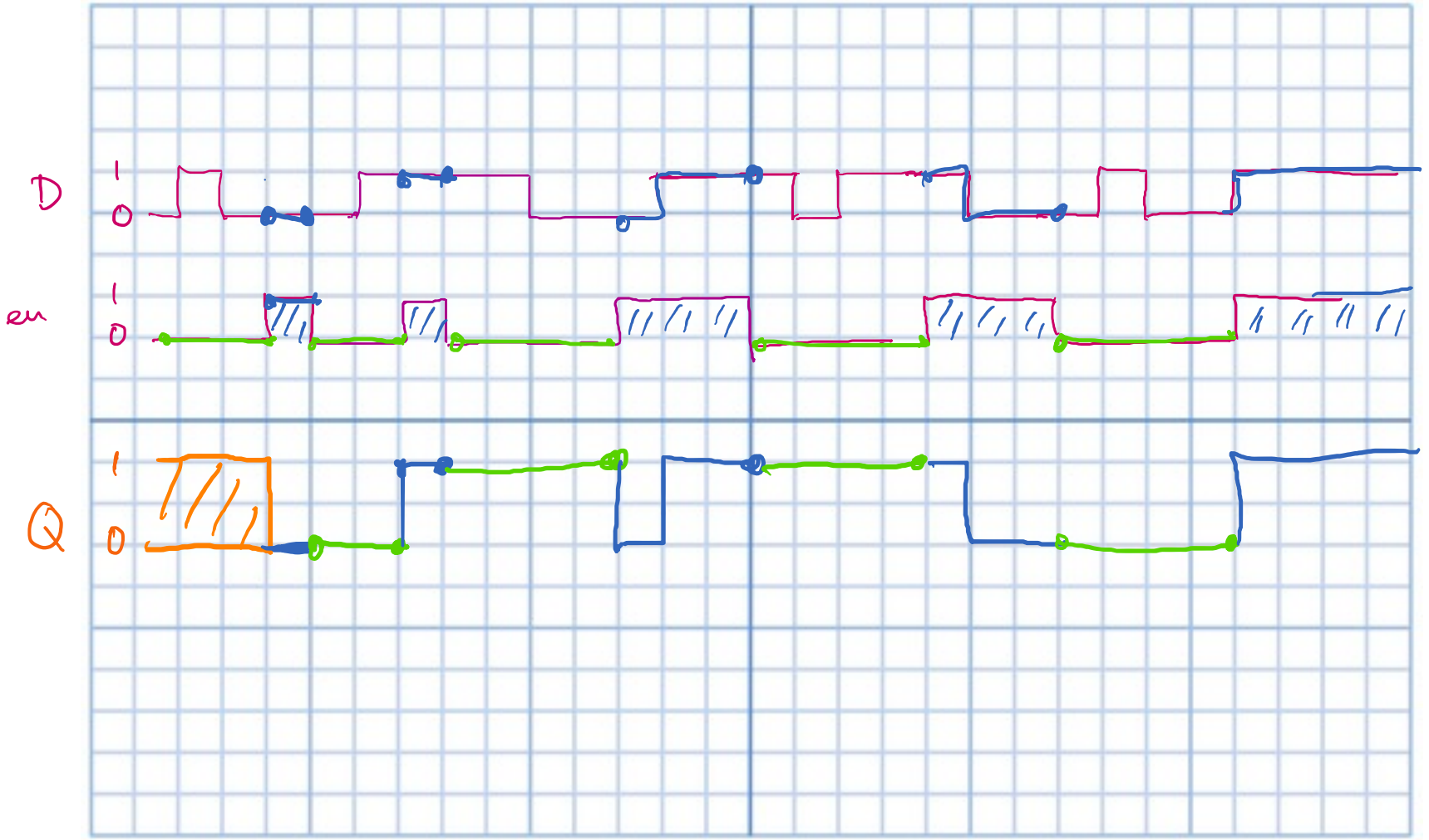
Output Follows Input when Enable=1

Inputs to D Latches

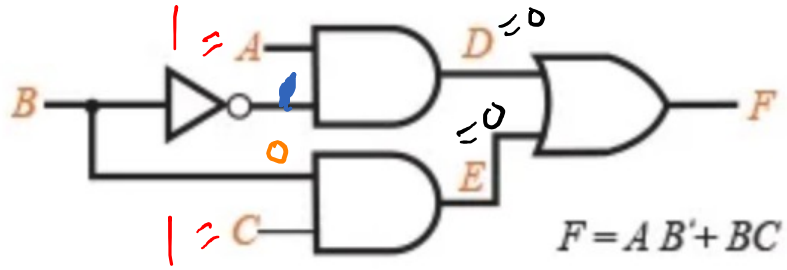


Inputs to D Latches

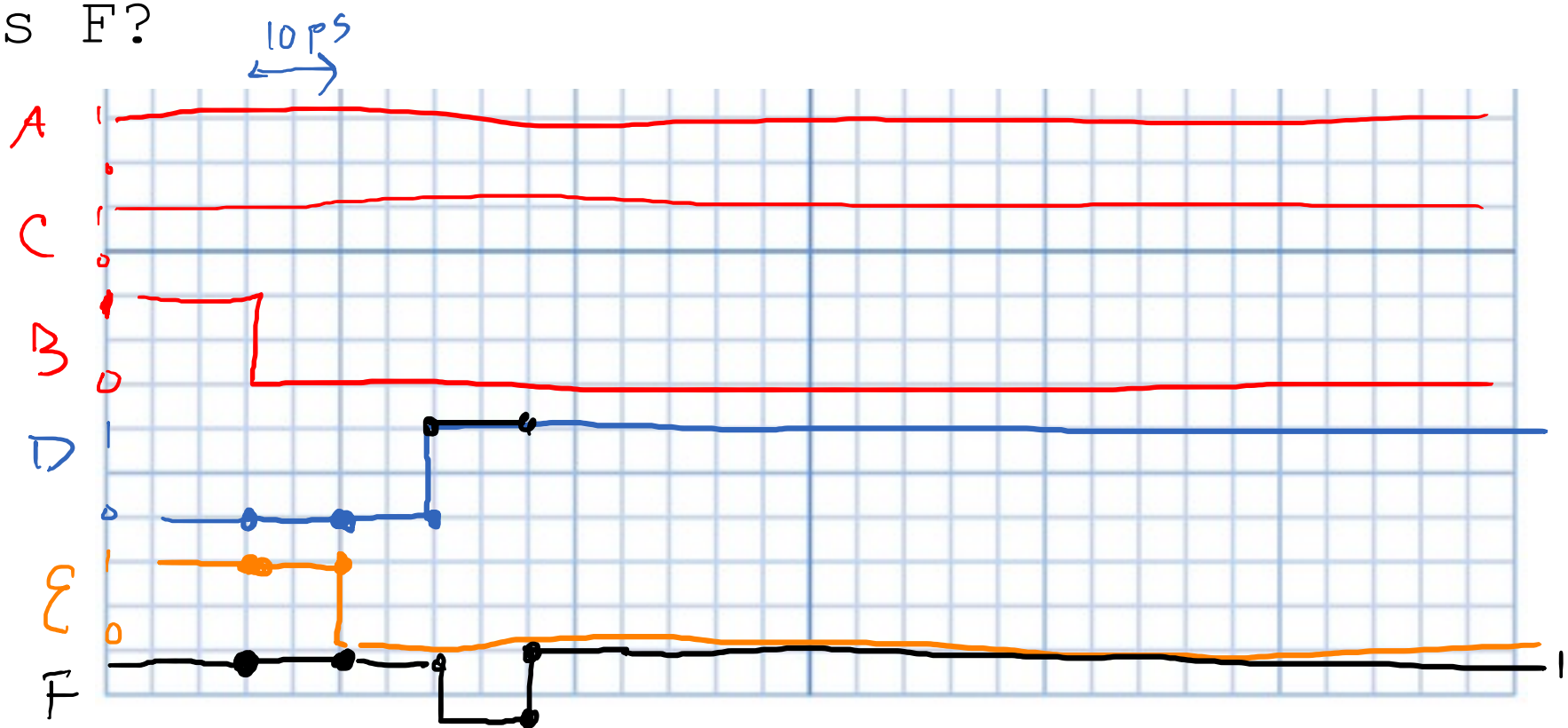
Output Follows Input when Enable=1



Glitches



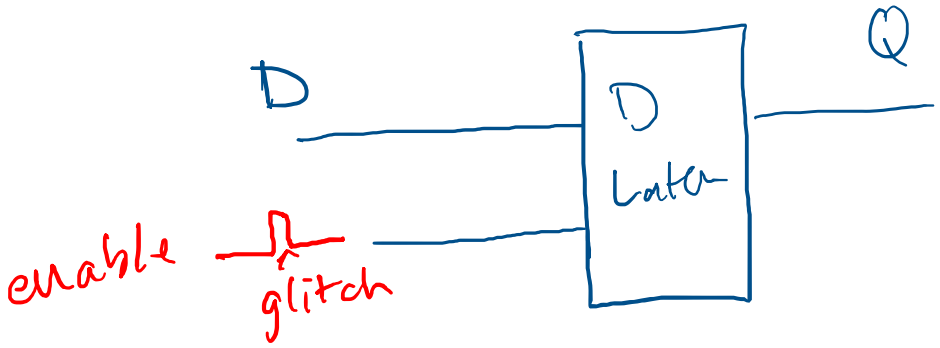
- Assume 10ps / gate.
- A=1, C=1, B falls
- What is F?

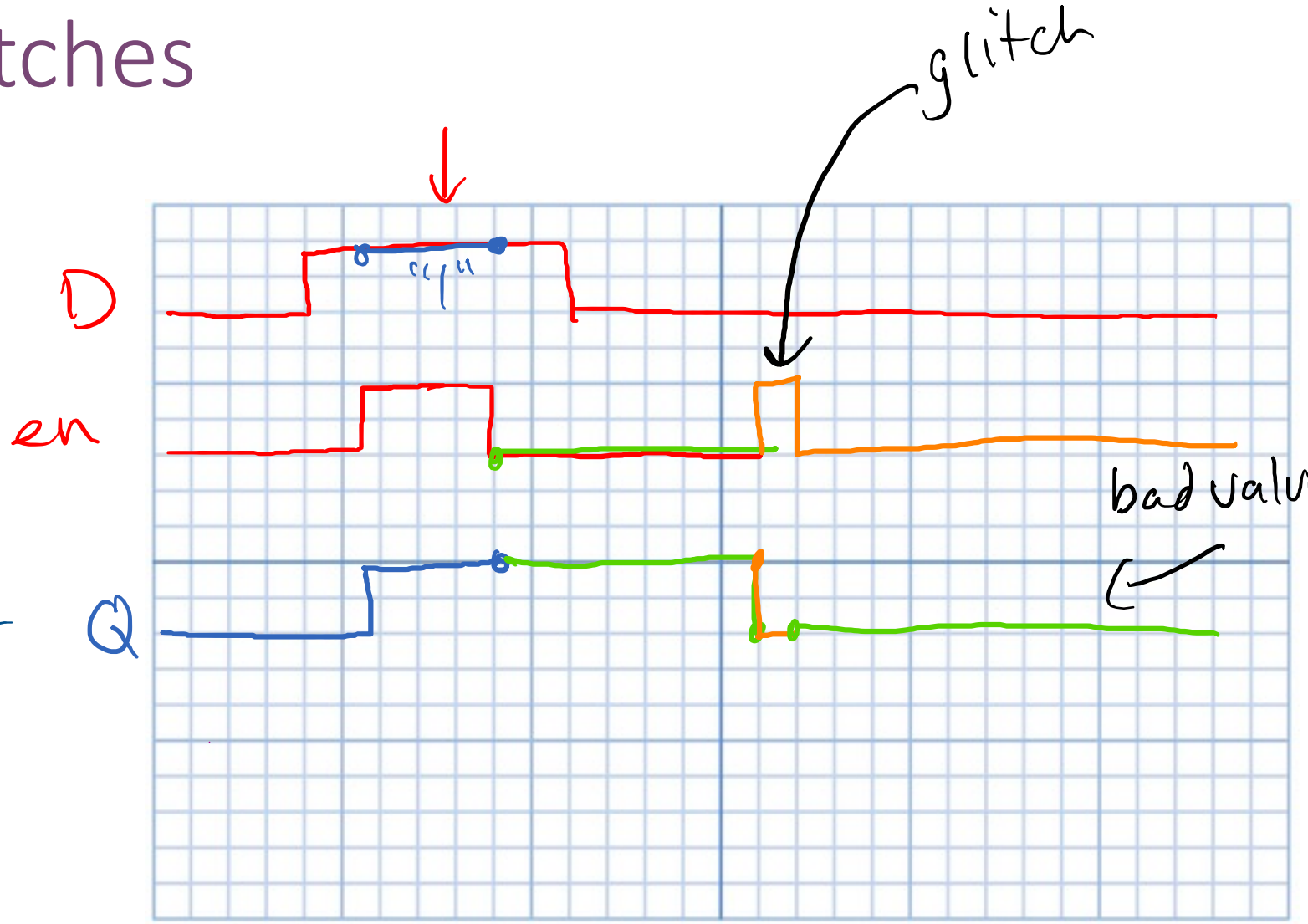


Glitch

- Unintended short errors in Boolean logic
- Caused by imbalance in gate delays

Glitches on D-Latches



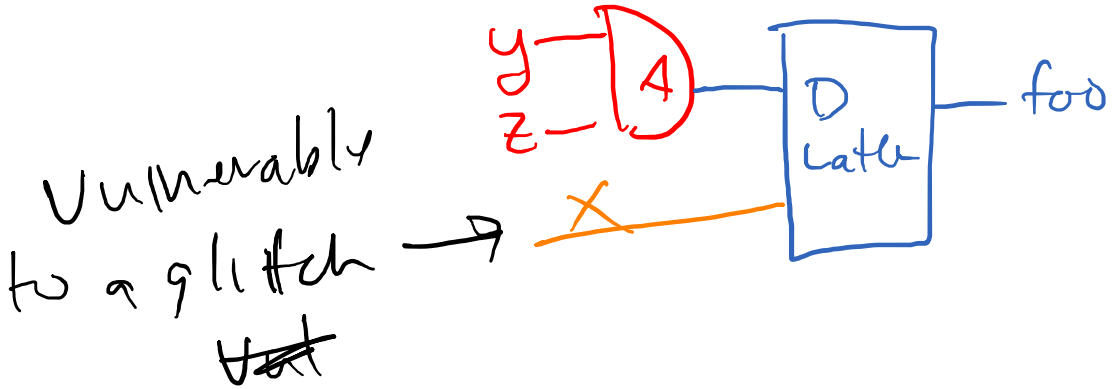


What's wrong here?

```
logic x,y,z;
logic foo, bar ;

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

No default Case

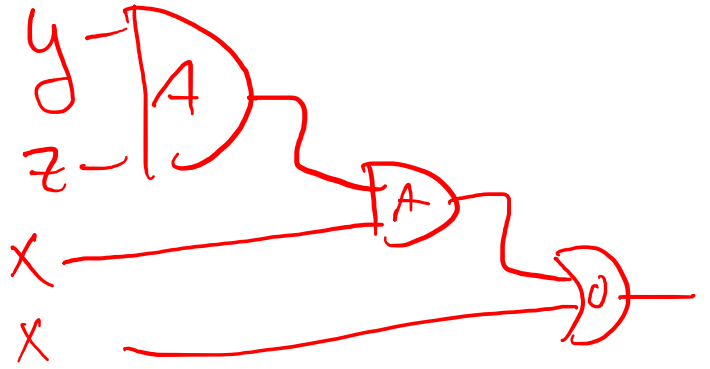
Inferred Latches




```
logic x,y,z;
logic 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;
logic foo, bar ;

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

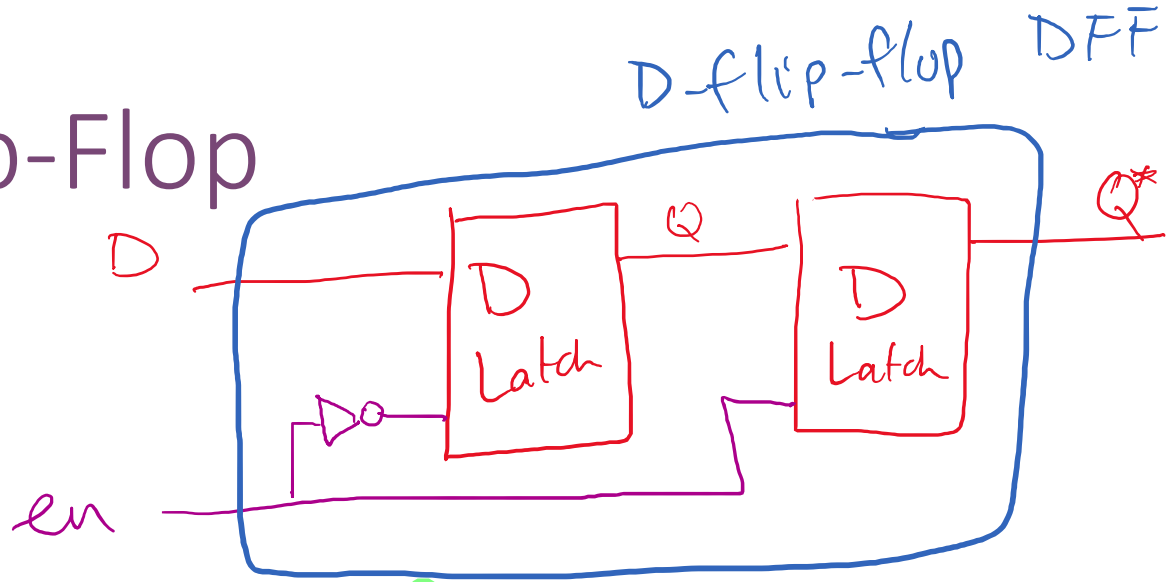


What if `x == 0`? `foo = bar = x`!

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

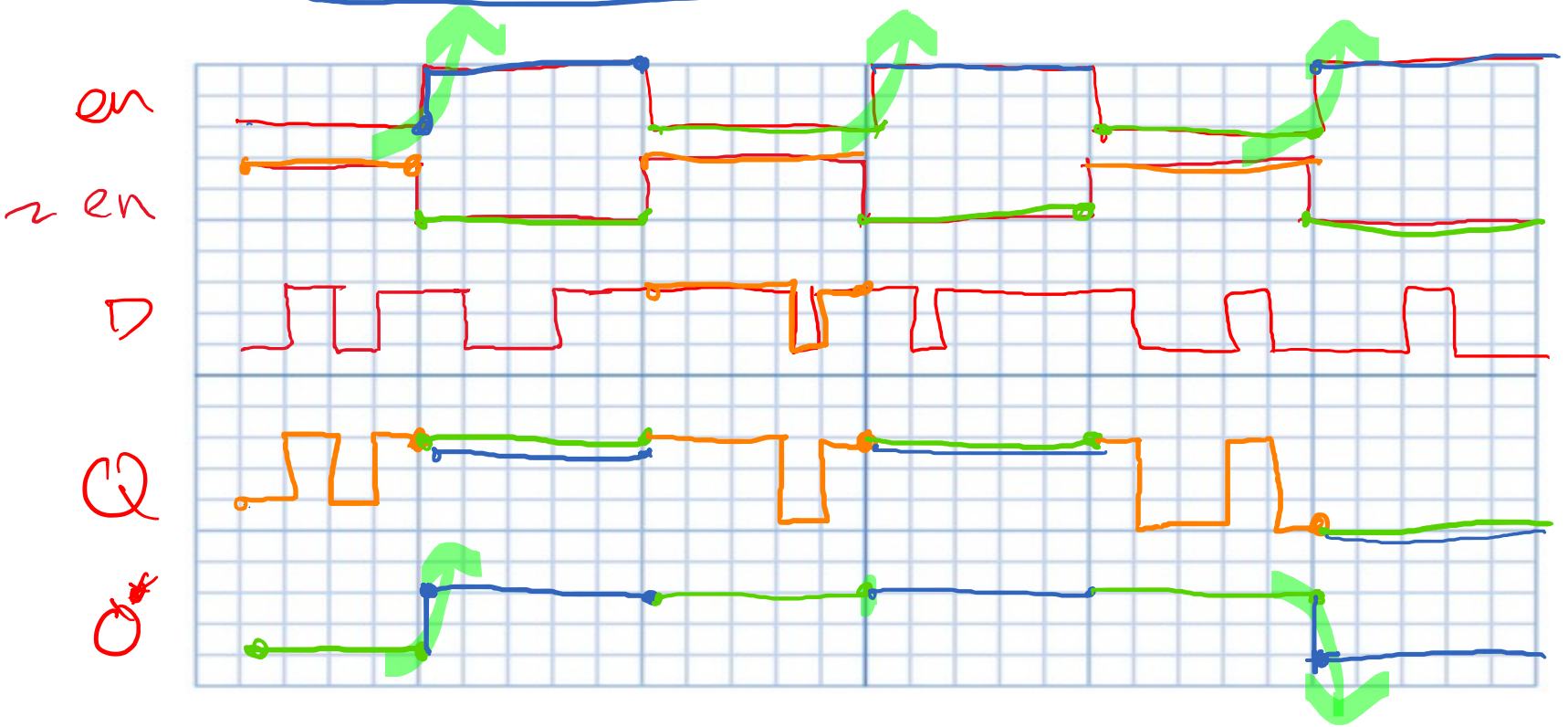
Always specify defaults for `always_comb`!

D Flip-Flop



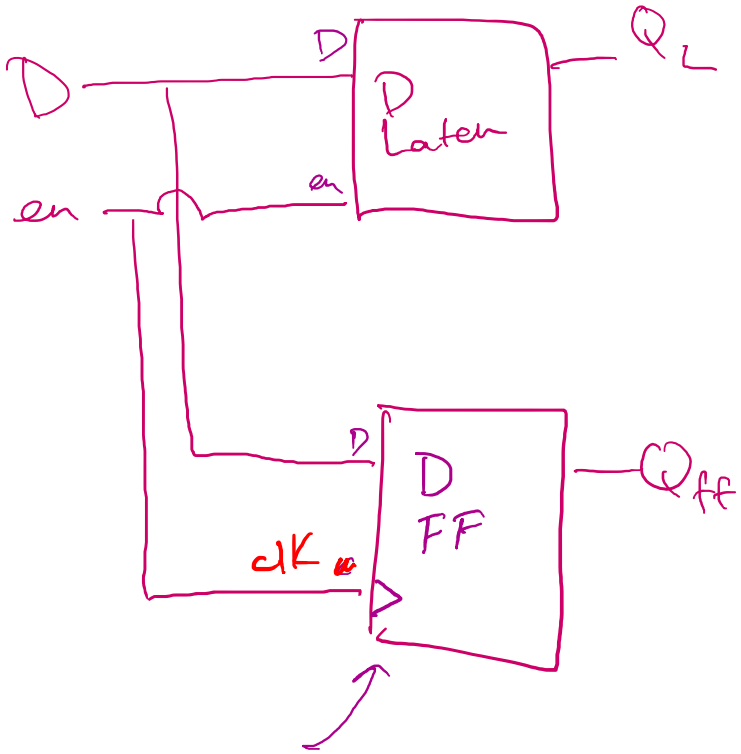
* no gate delays

Stop here!

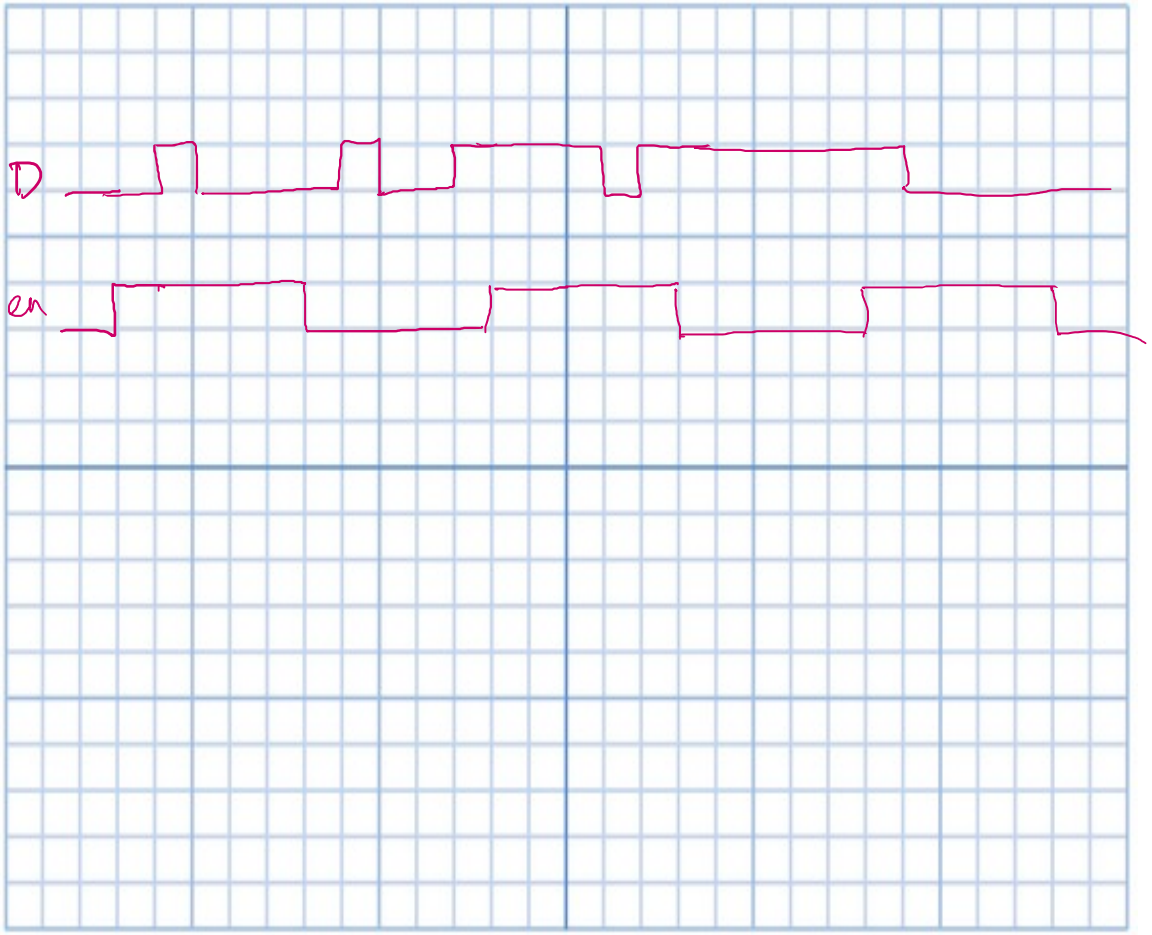


Levels vs. Edges

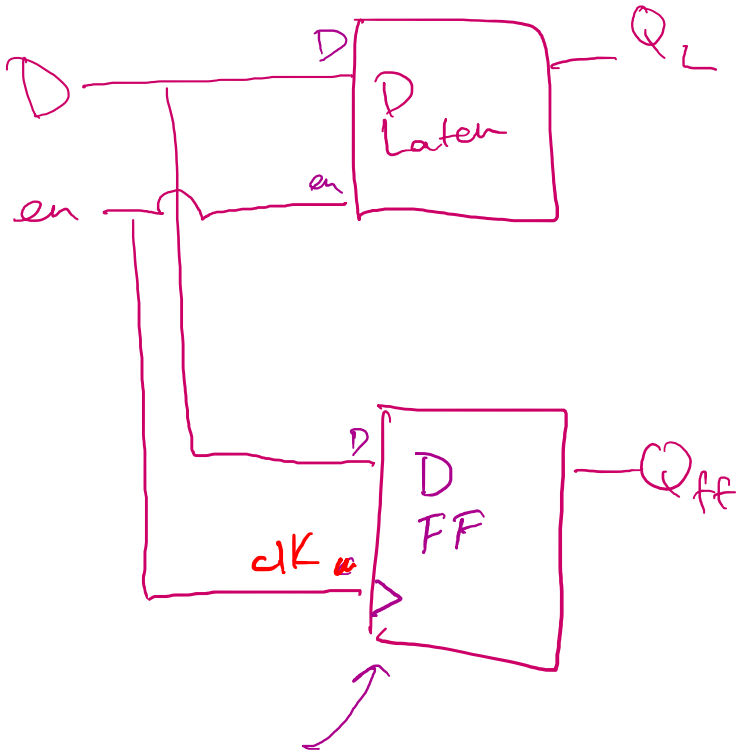
D Flip-Flop vs. D Latch

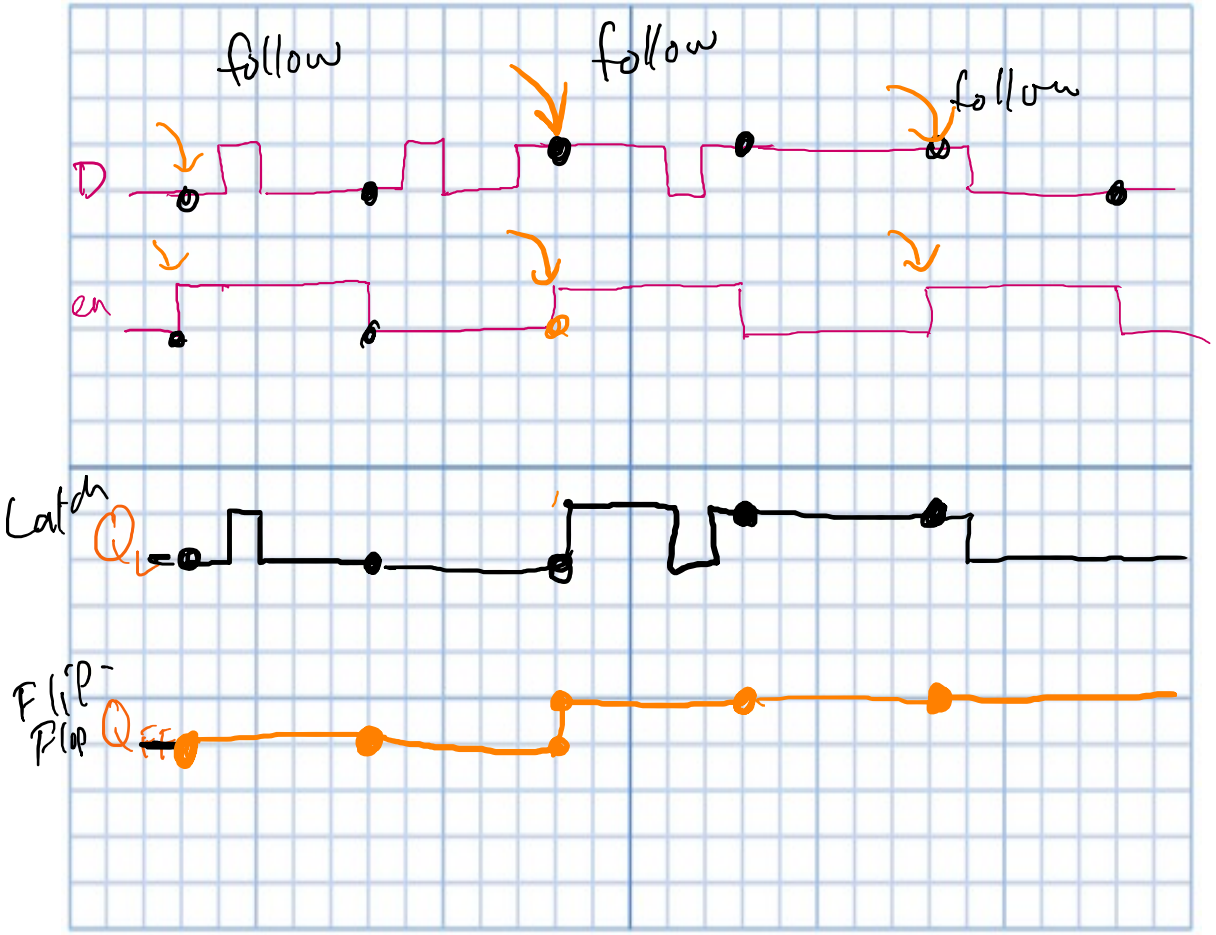


the " > " symbol tells you it is Flip-flop



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 logic q    //always block
);

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

endmodule
```

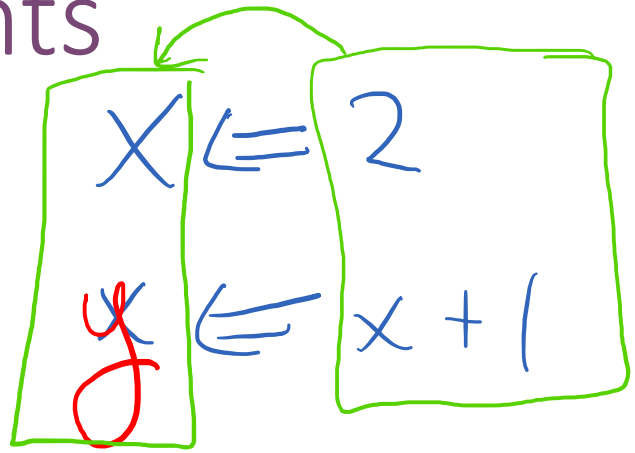
D Flip-Flop w/ Clock

```
module d_ff (
    input d,            //data
    input clk,          //clock
    output logic q      //always block
);

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

endmodule
```

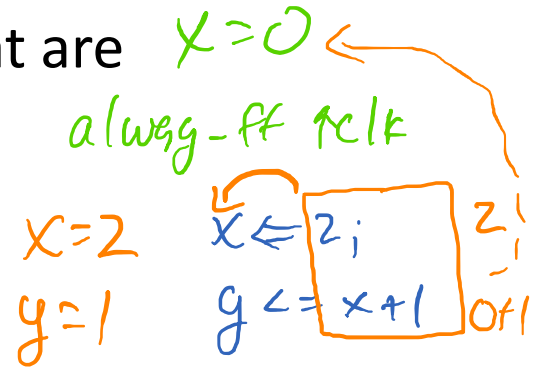
Blocking vs. NonBlocking Assignments

- 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.

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. 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`

NON-BLOCKING (<=) for `always_ff`

D-FlipFlop w/Clock

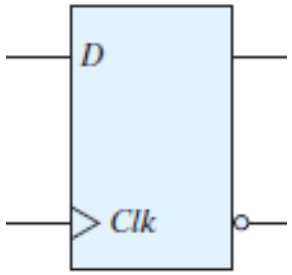
```
module d_ff (
    input d,            //data
    input clk,          //clock
    output logic q      //in always block
);

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

endmodule
```

What is `q` before `posedge clk`?

D-FF's with Reset

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

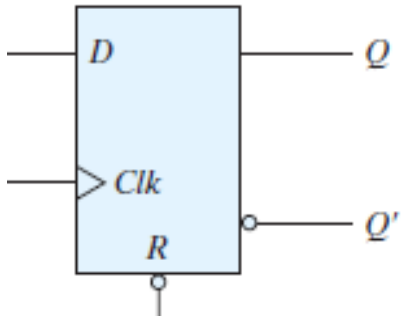
Verilog models of D flip-flop



Edge triggered D flip-flop:

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

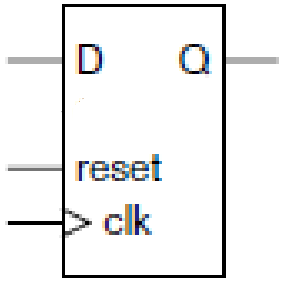
No reset ff



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;
```

D-FlipFlop w/Reset

```
module d_ff (
    input d,                    //data
    input clk,                  //clock
    input rst,                  //reset
    output logic q              //output
);

      always_ff @( posedge clk )
      begin
            if (rst) q <= 'h0;
            else q <= d;
      end

endmodule
```