Test

ENGR 210 / CSCI B441
"Digital Design"

Latches +
Flip Flops

Andrew Lukefahr

Announcements

- P2 due Friday
  - Last of the remote-only projects.

- If work is remote next week, email me!
  lukefahr@iu.edu
- Otherwise, expect more details hopefully later today
  Luddy 4111

Last Time

- Verilog
  - Wire vs Logic
  - Always_comb
- Ripple-Carry Adder
  - Subtraction
  - Carry vs. Overflow
- Sequential Logic

assign x = ? ← wire

always_comb ← logic

x = ?

+ defaults

```
always_comb begin
    y = 2; // default
    if(x)
        y = 3;
end
```

```
module foo(
    input x;
    output logic y;
);
```

~~assign y = ~x;~~

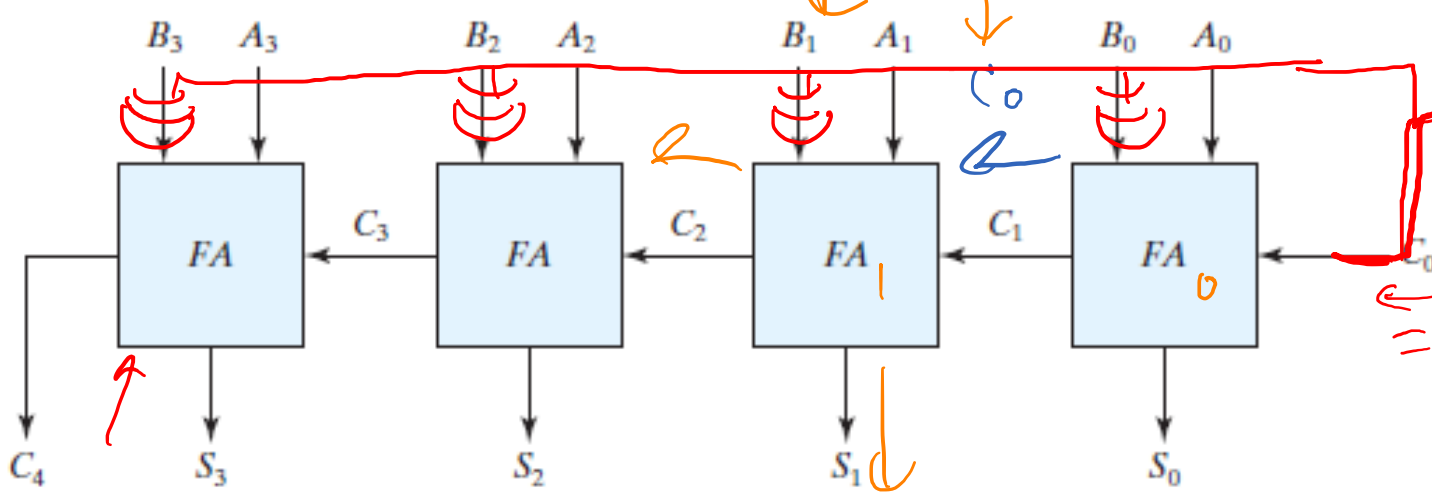
```
always_comb
    y = x
```

C

```
int x = 2;
```

x = 3

Ripple-Carry Adder



$A = A_3\ A_2\ A_1\ A_0$

$B = B_3\ B_2\ B_1\ B_0$

$A - B =$

$A + (-B) =$

$\rightarrow A + (\sim B + 1)$

B → xor → ~B

Sub = C_0

Sub = 0 ⇒ Add ($C_0 = 0$)

Sub = 1 ⇒ Subtract ($C_0 = 1$)

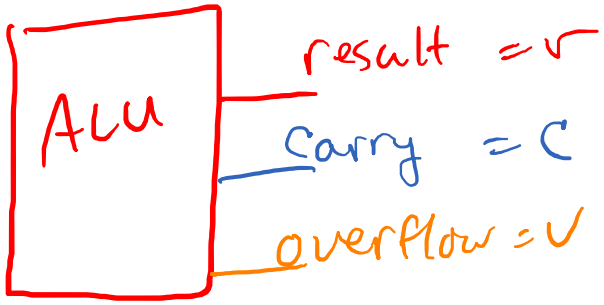
Sub = $A + (\sim B + 1)$ combined w/

```
wire [9:0] A9 = {1'b0, A};
```

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

How do I store data?



Unsigned Arith

$$\begin{array}{r} A \\ +\ B \\ \hline \{c, r\} \end{array}$$

c is the 9th bit

c = 1 ⇒ error

Signed Arith

$$\begin{array}{r} A \\ +\ B \\ \hline r \end{array}$$

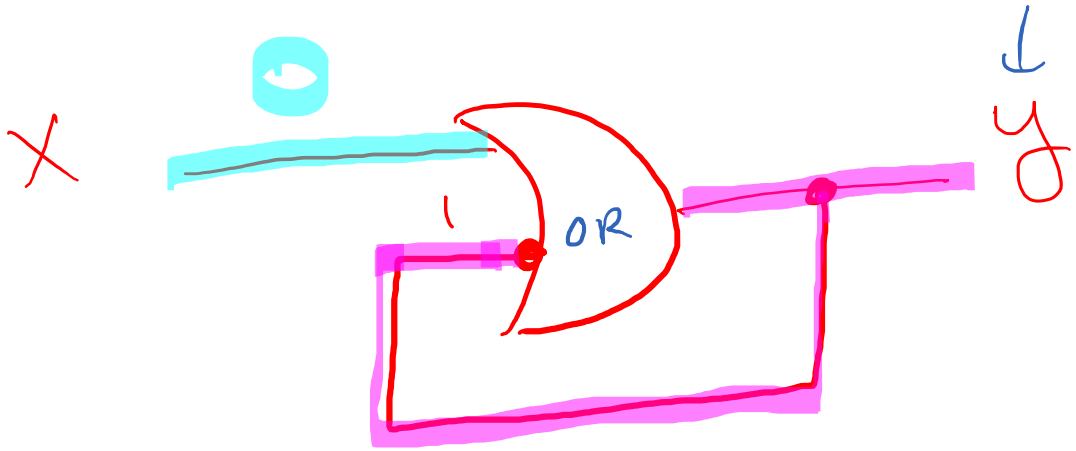
v = 1 error

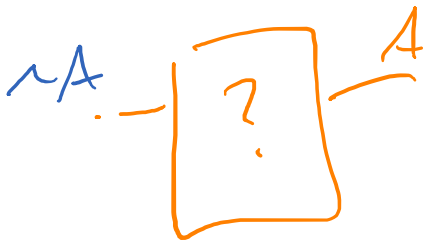
How do I store data?


→ "1" or "true" or "hot"

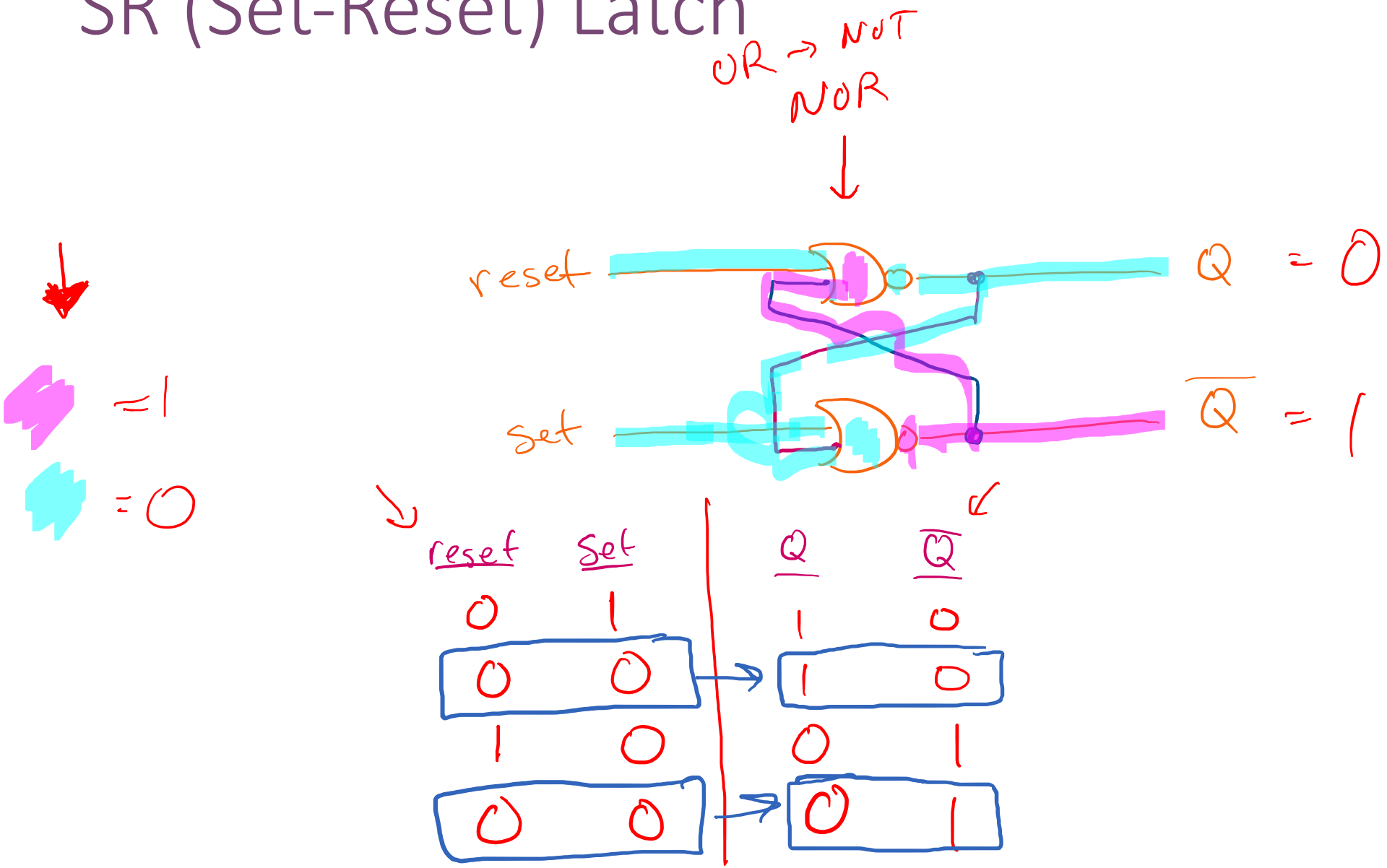
→ "0" or "false" or "cold"

x	y	next value of y
0	0	0
1	?	1

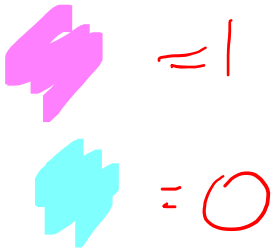


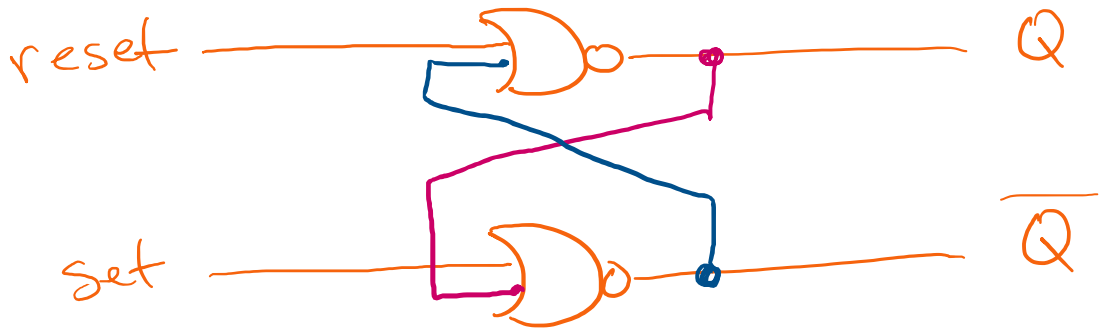


SR (Set-Reset) Latch



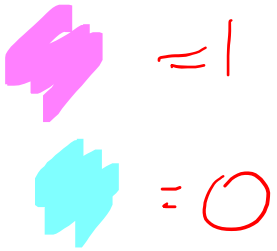
SR (Set-Reset) Latch

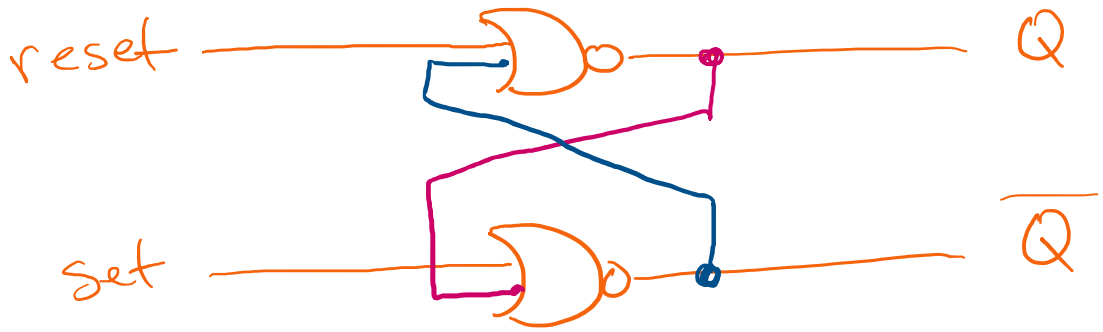




reset	set	Q	$\overline{Q}$
0	1	1	0

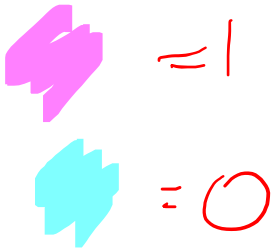
SR Latch

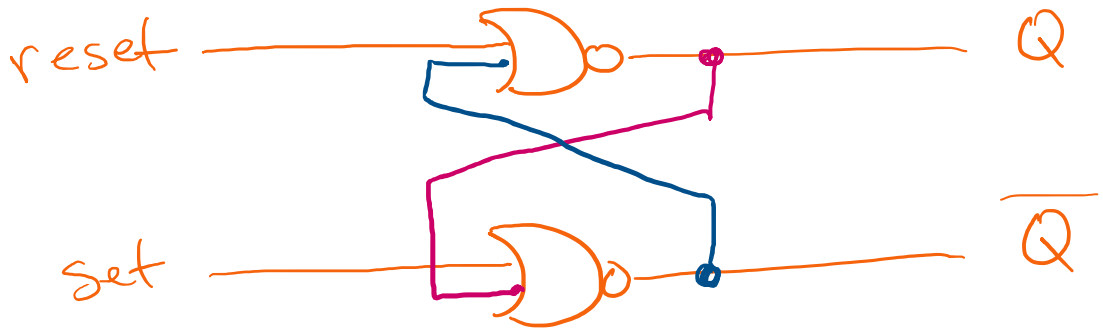




reset	set	Q	$\overline{Q}$
0	1	1	0
0	0	1	0

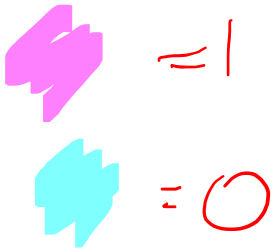
SR Latch

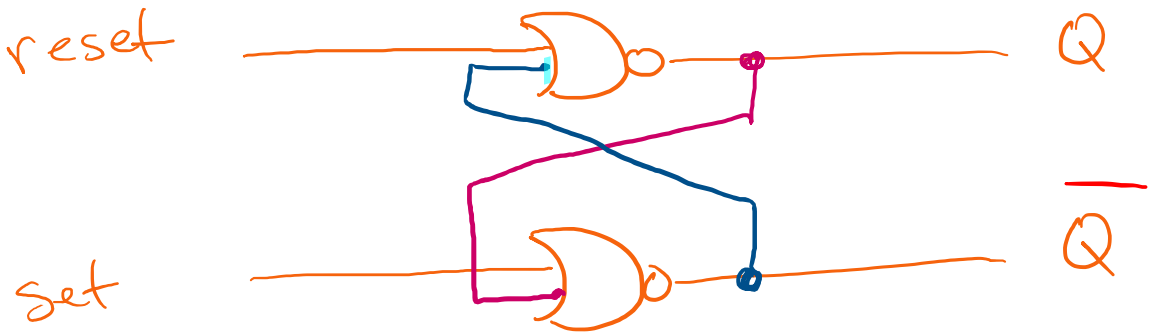




reset	set	Q	$\overline{Q}$
0	1	1	0
0	0	1	0
1	0	0	1

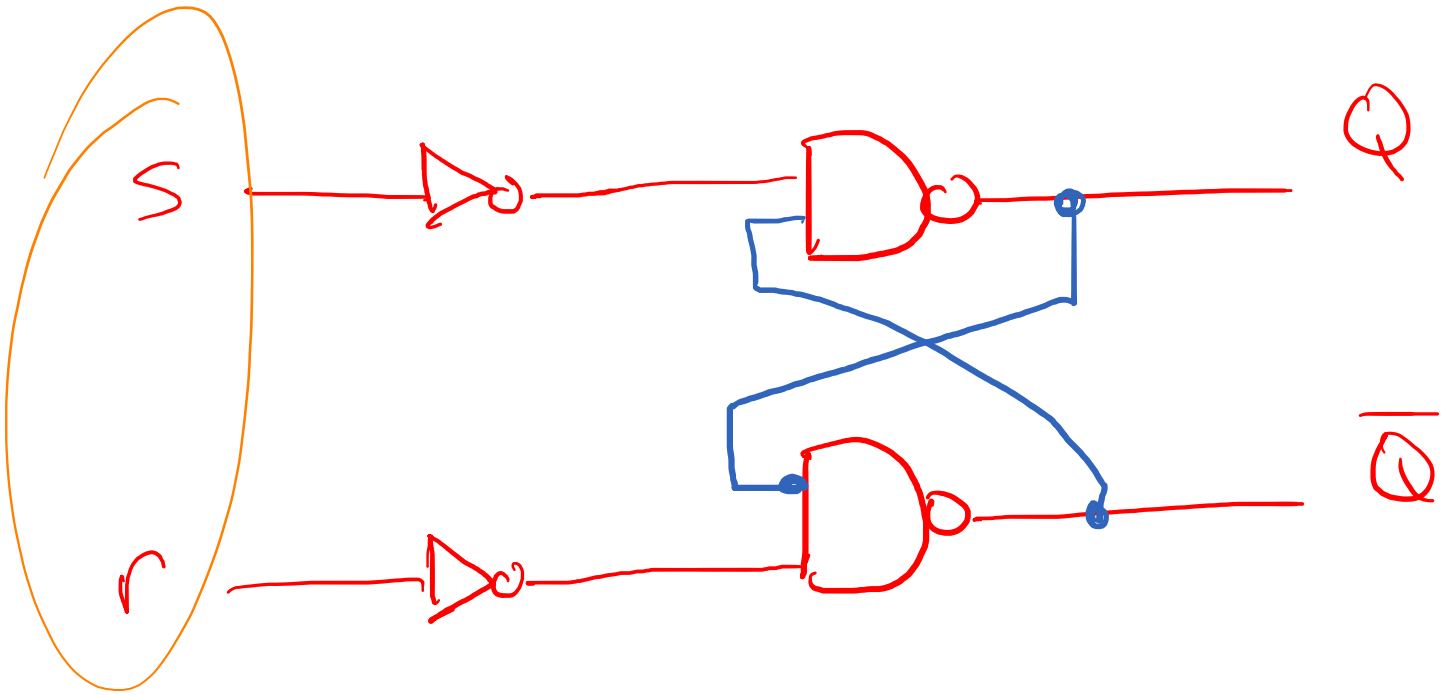
SR Latch w/ S=1 & R=1

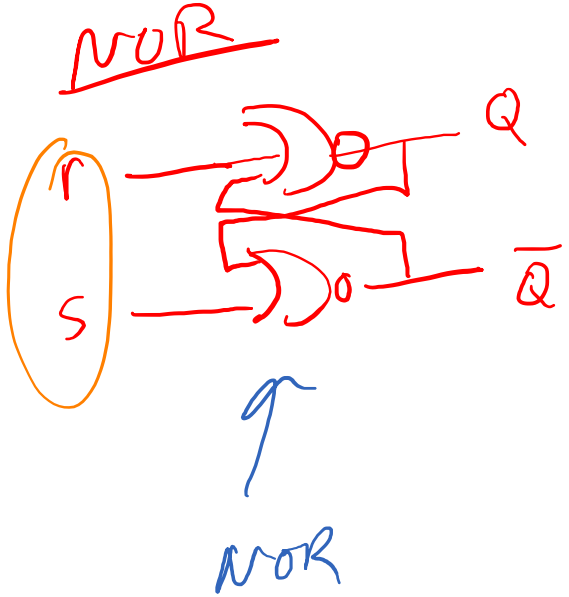




set	reset	Q	$\overline{Q}$
1	1	0	0
0	0	0,1,0,1,0,1,0,1,...	

SR Latch w/NAND gates

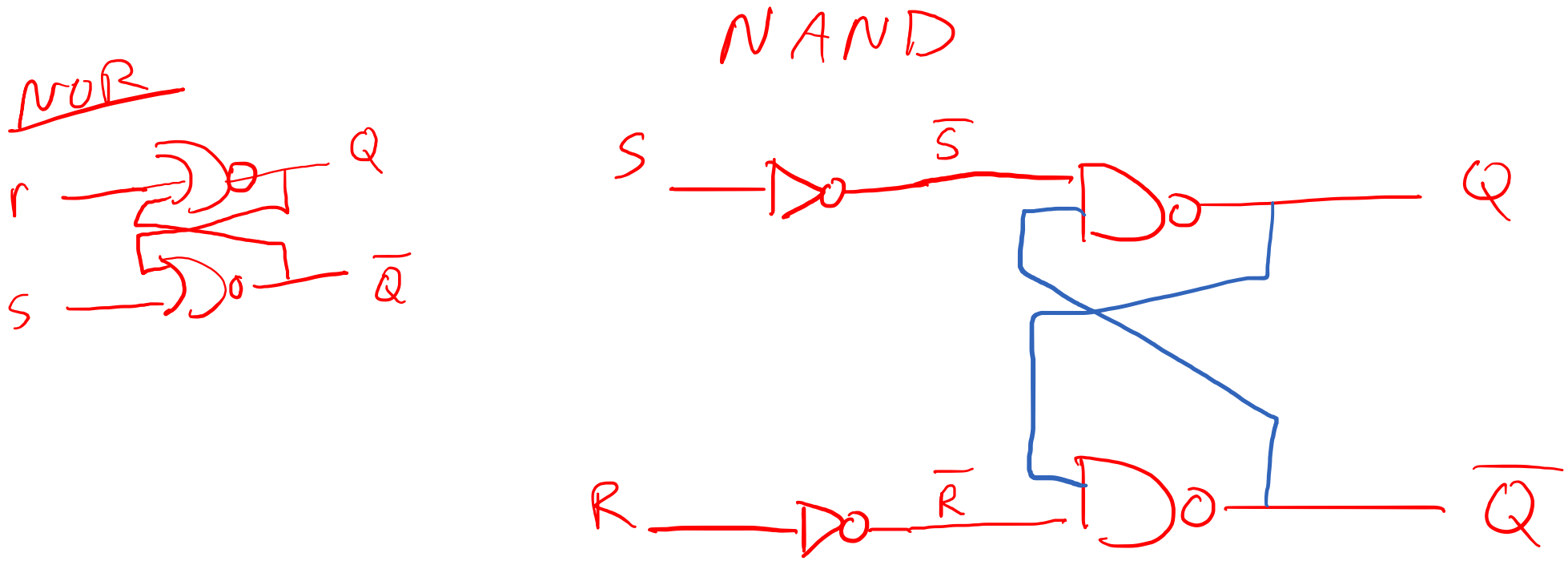




* drawn w/ NAND Not NOR

S & R flip sides

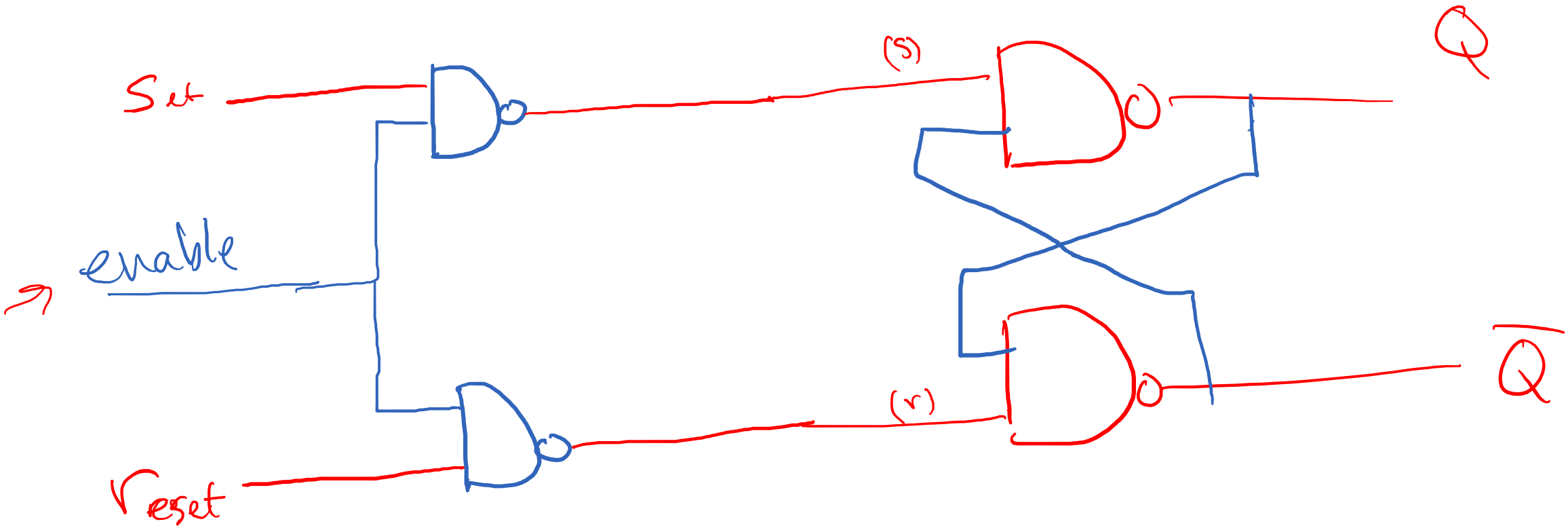
SR Latch w/NAND gates



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

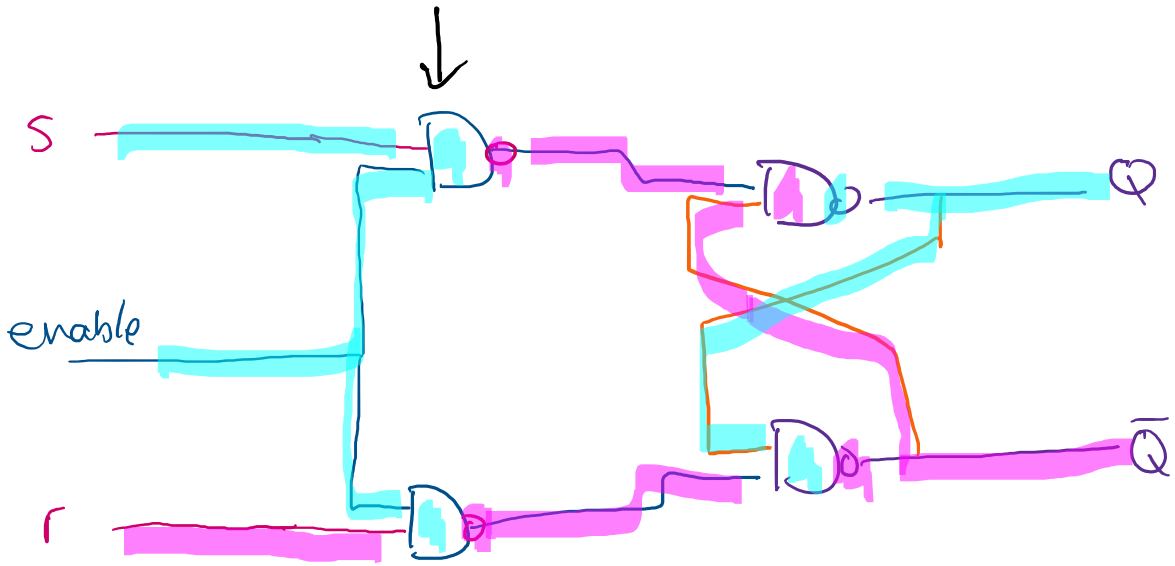
→ easier for me to draw

SR Latch with Enable



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

added enable bit

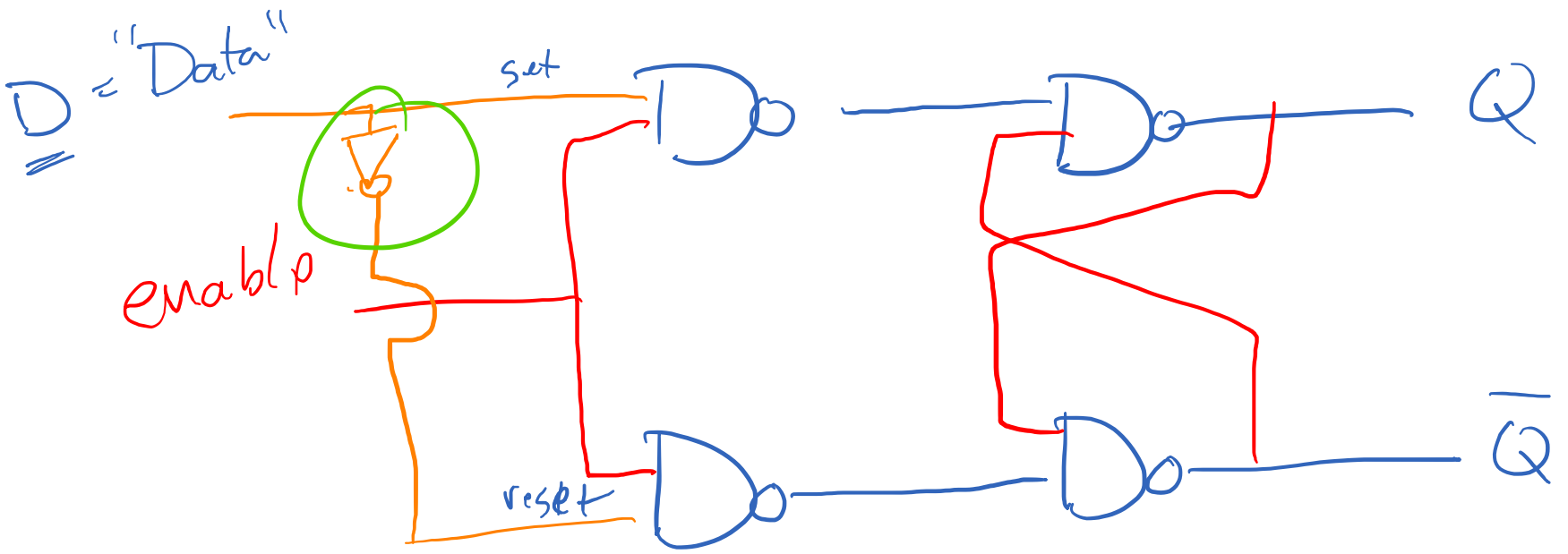
regular $\overline{SR}$ Latch

* assume no ~~S=1~~ & ~~r=1~~

=> S=0, r=0

D-Latch

"Data" Latch



D latch

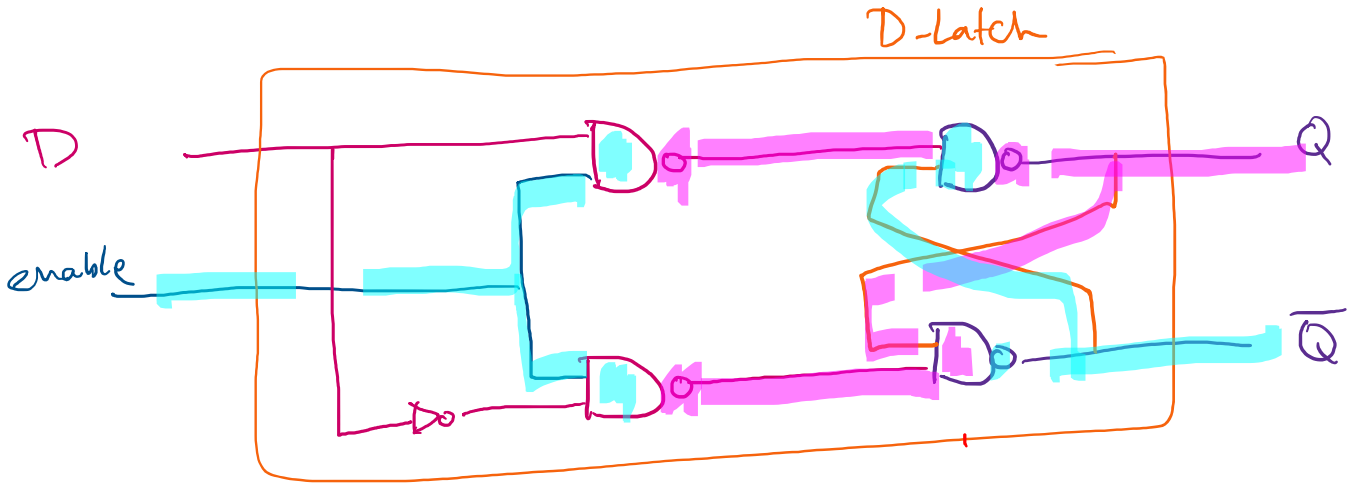
SR latch

SR Latch w/ Enable

D-Latch

= 1

= 0



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

when en = 1 ⇒

$Q = D$, $\overline{Q} = \overline{D}$

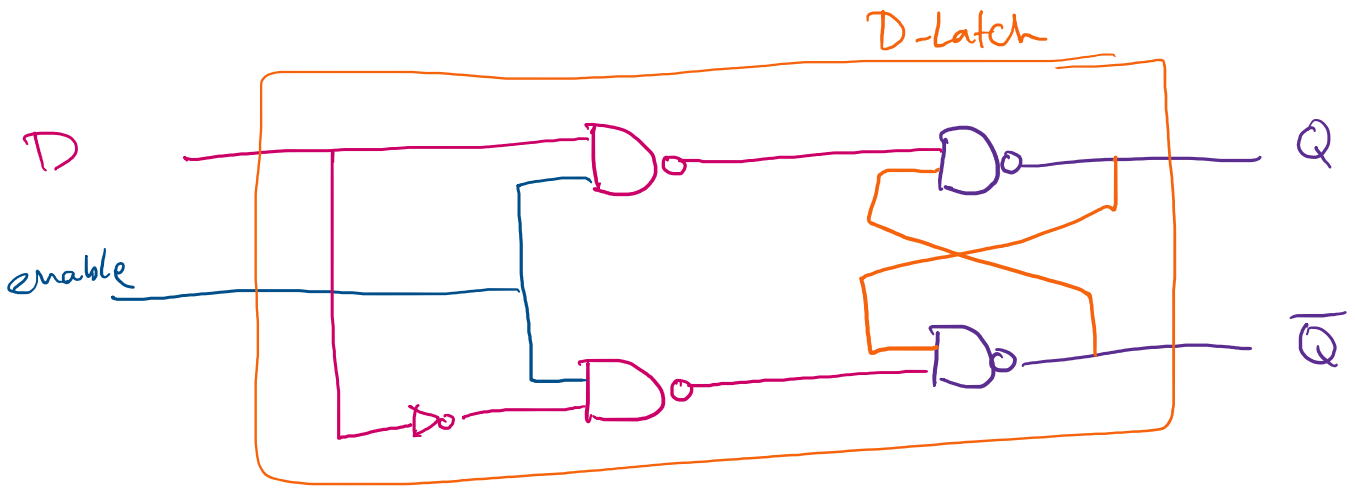
when en = 0 ⇒

$Q = Q$, $\overline{Q} = \overline{Q}$

D-Latch

= 1

= 0

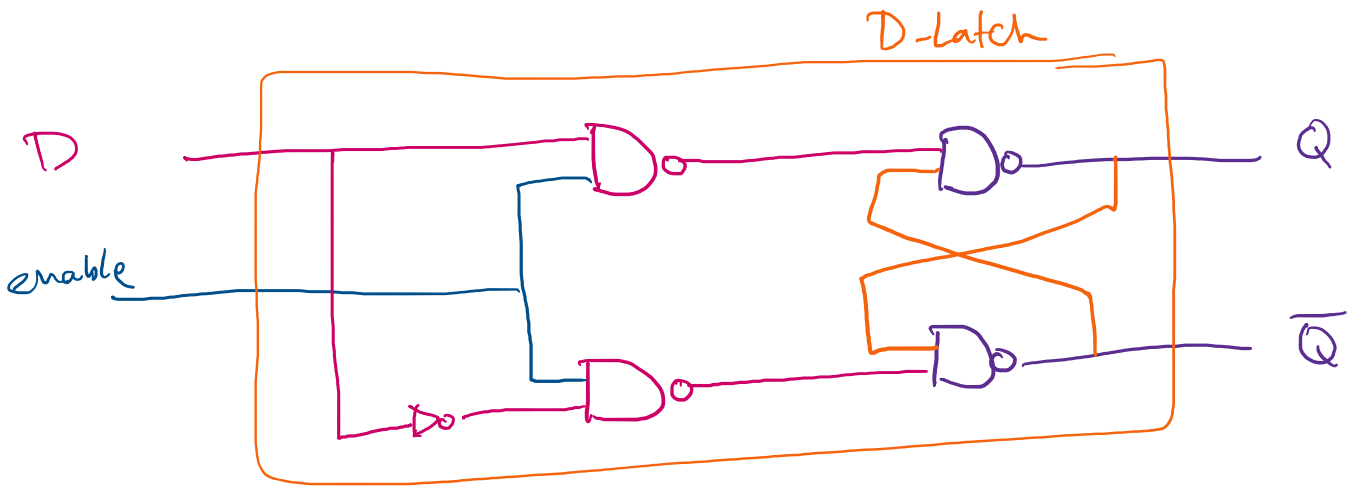


D	en	Q	$\overline{Q}$
0	0	Q	$\overline{Q}$

D-Latch

= 1

= 0

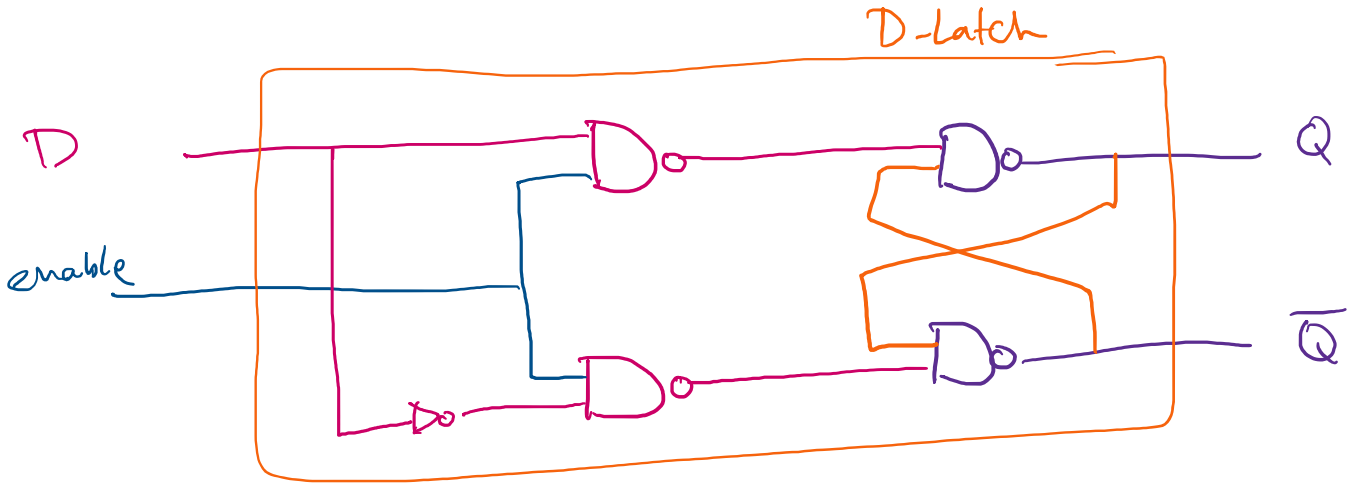


D	en	Q	$\overline{Q}$
0	0	Q	$\overline{Q}$
1	0	Q	$\overline{Q}$

D-Latch


= 1

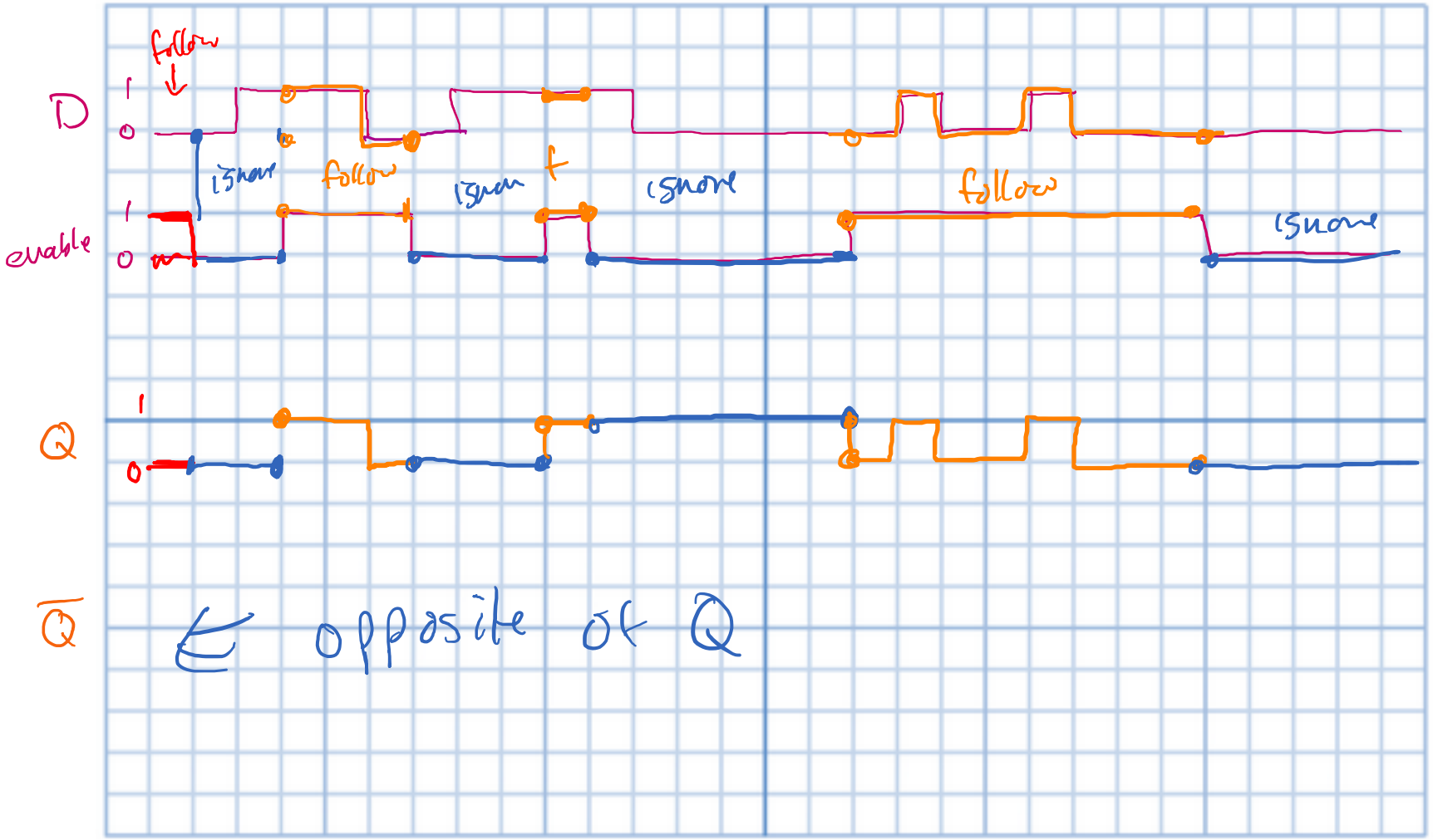
= 0



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

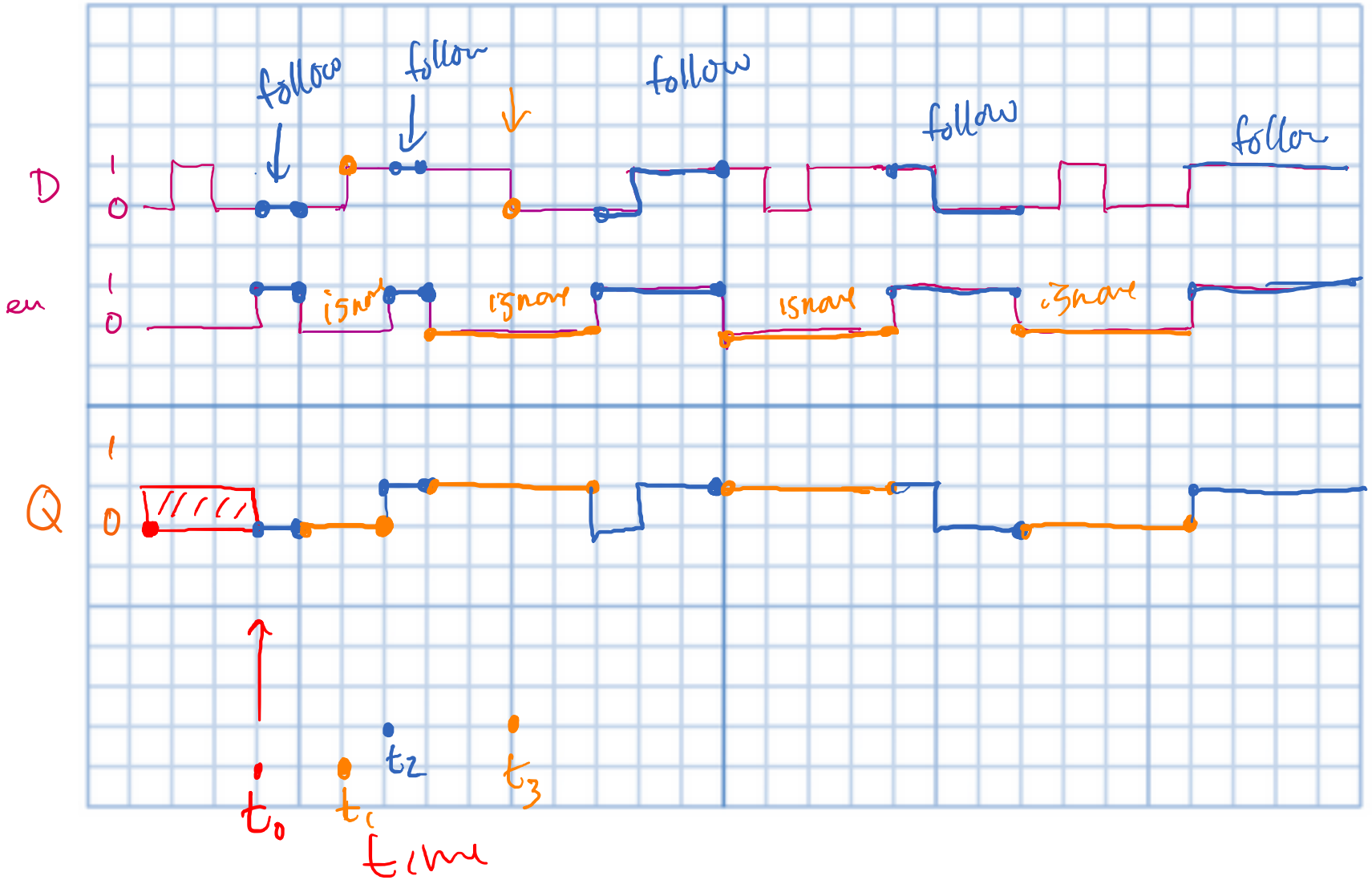
* assume negligable gate delays

Inputs to D Latches



Q follows D when enable = 1, otherwise Q ignores D.

Inputs to D Latches

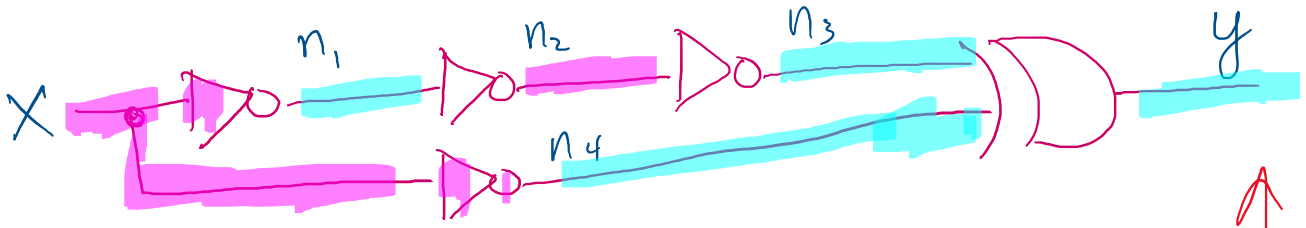


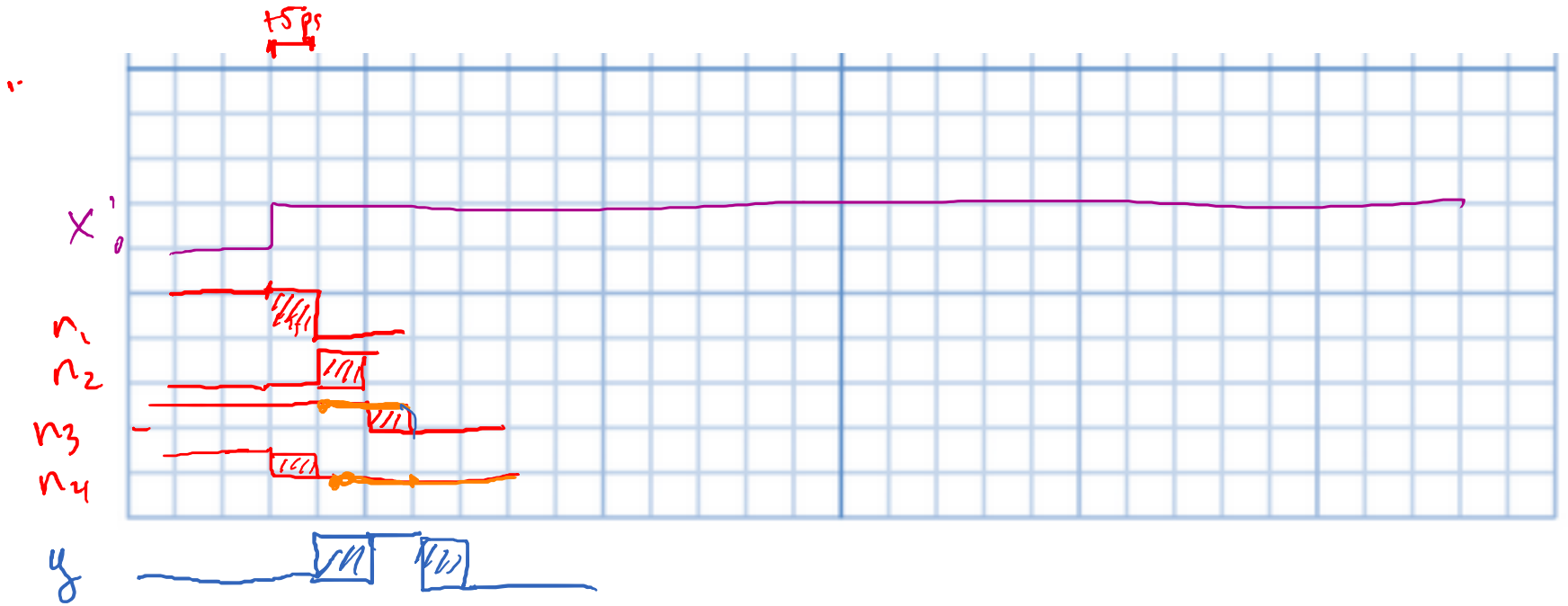
Glitches

→ unintended, short, errors in boolean logic

→ caused by gate delays

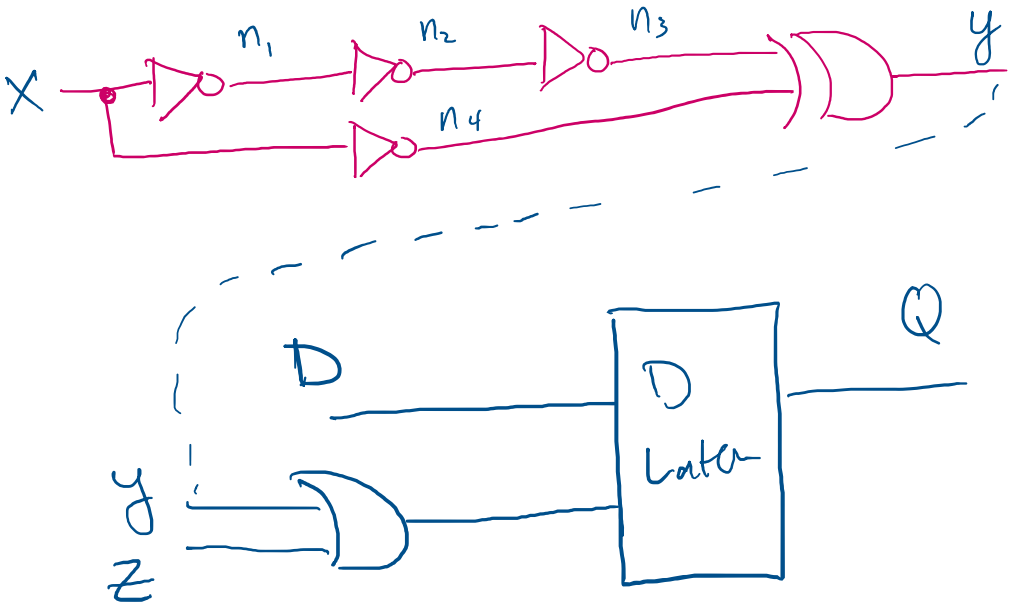
- Assume 5ps / gate.
- x changes from 0 to 1.
- What happens to y?

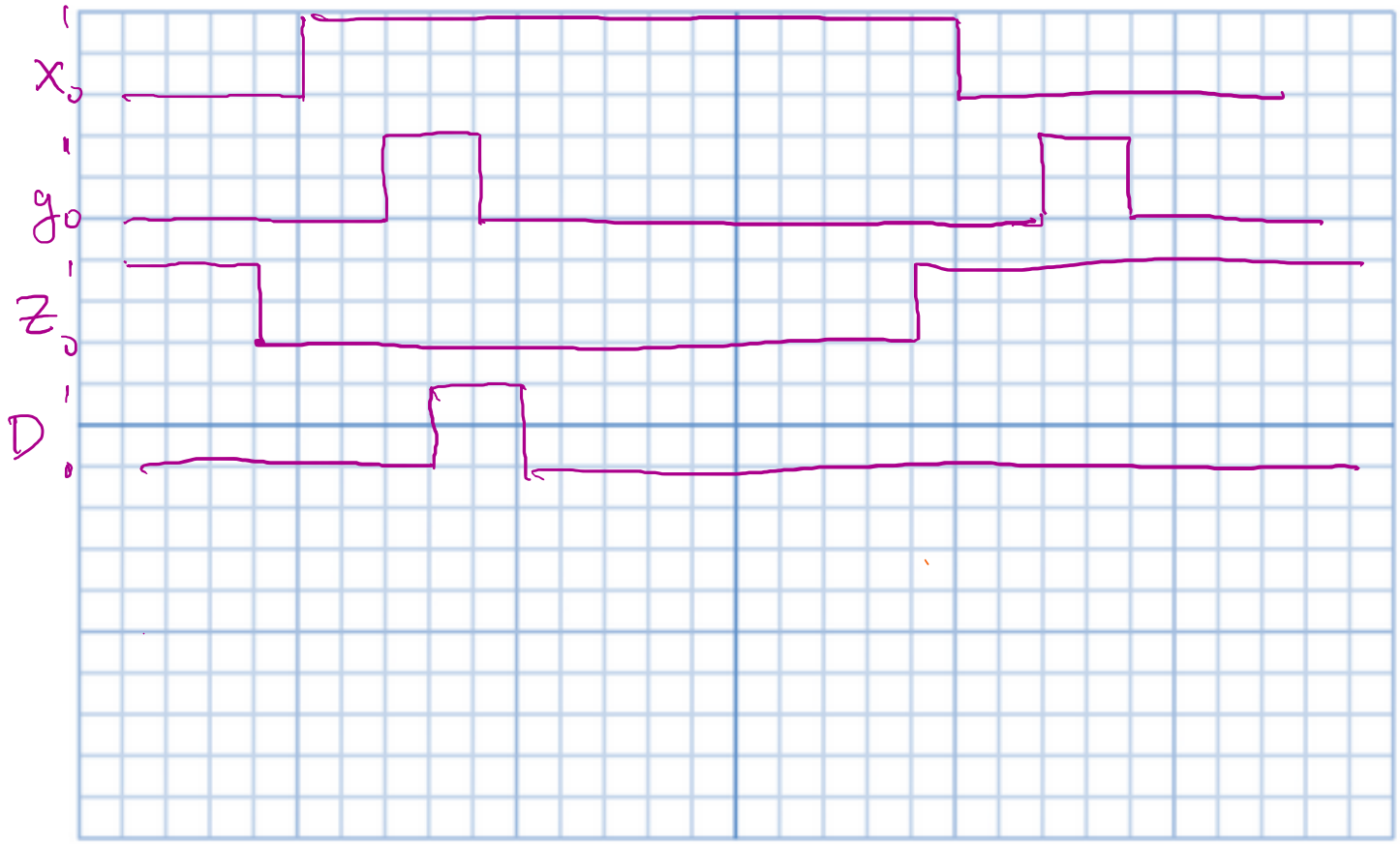




should always be zero

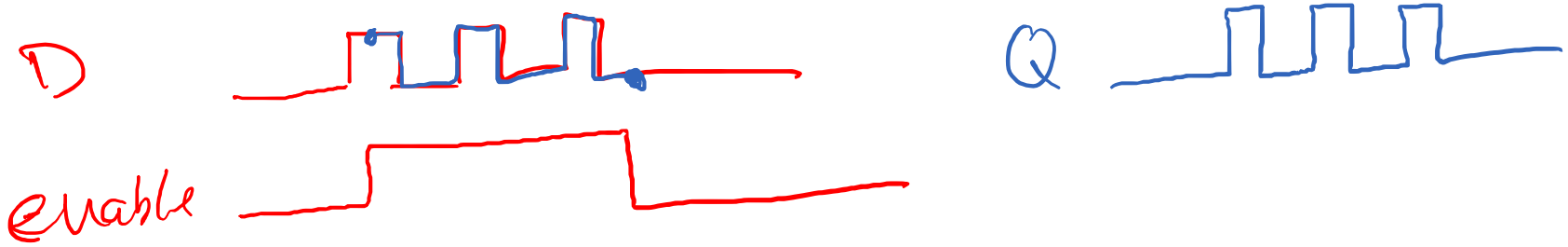
Glitches on D-Latches





Levels vs. Edges

D Latch → Q follows D whenever enable is 1



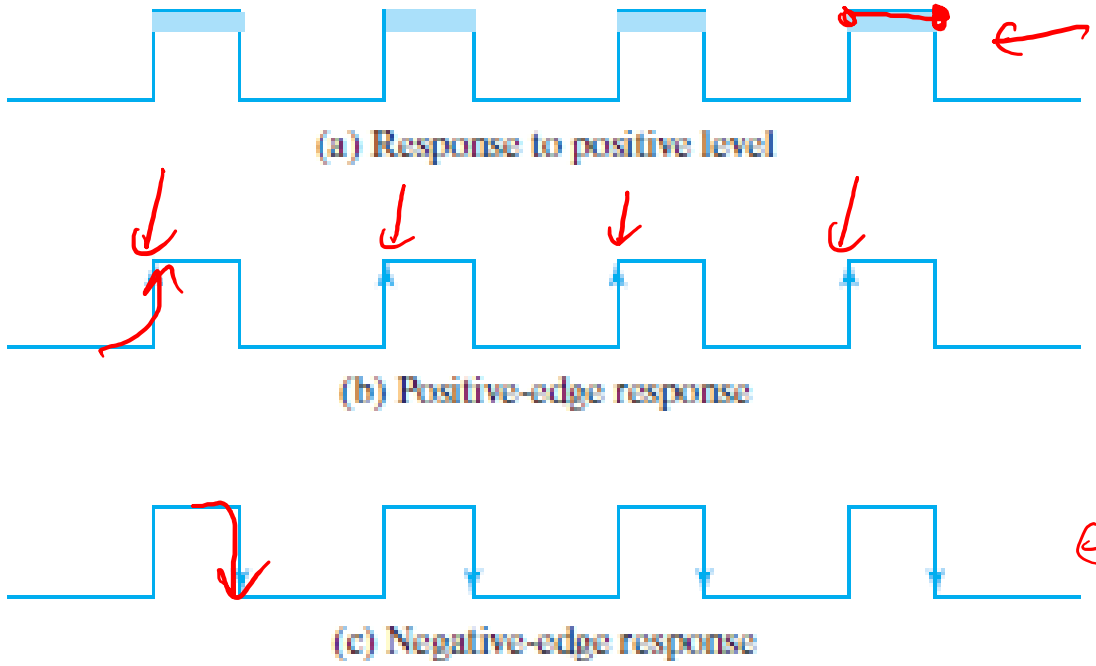
D flip-flop → Q follows D on rising edge of enable

Trigger

The state of a latch or flip-flop is switched by a change in the control input. This momentary change is called a *trigger*, and the transition it causes is said to trigger the flip-flop.

The problem with the latch is that it responds to the *level* of a clock pulse.

The key to the proper operation of a flip-flop is to trigger it only during the control (clock) signal *transition* .

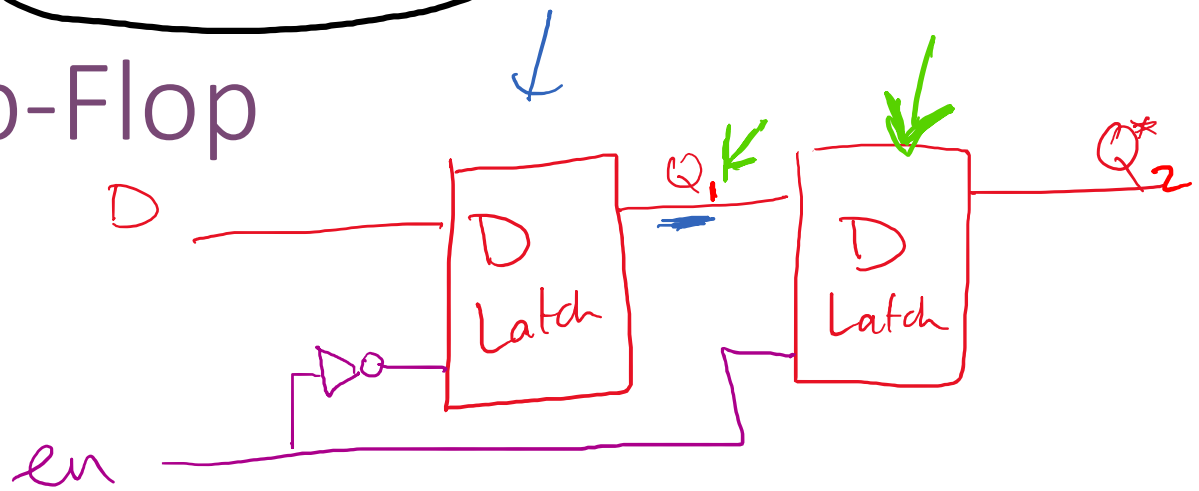


Storage elements that operate with signal levels (rather than signal transitions) are referred to as latches ; those controlled by a clock transition are flip-flops .

Flip-flop can be implemented with two latches, connected in master-slave configuration.

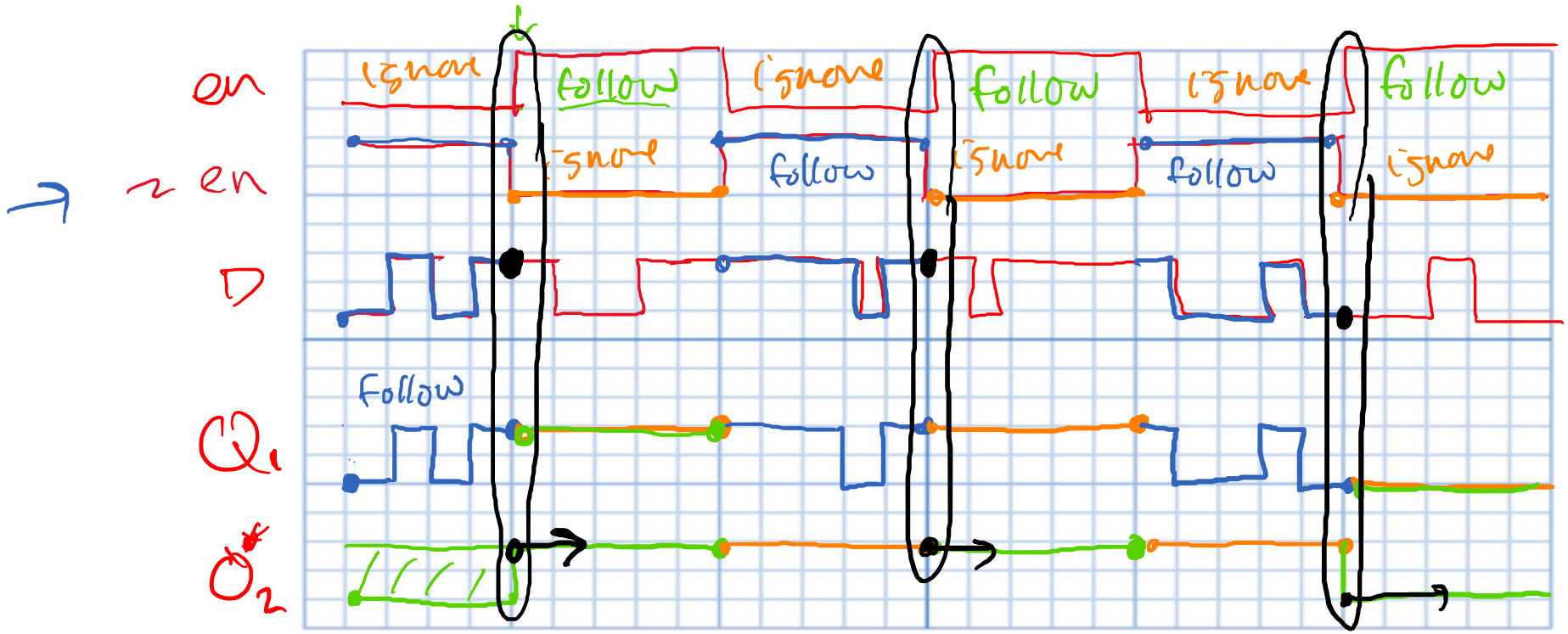
Stopped here!

D Flip-Flop



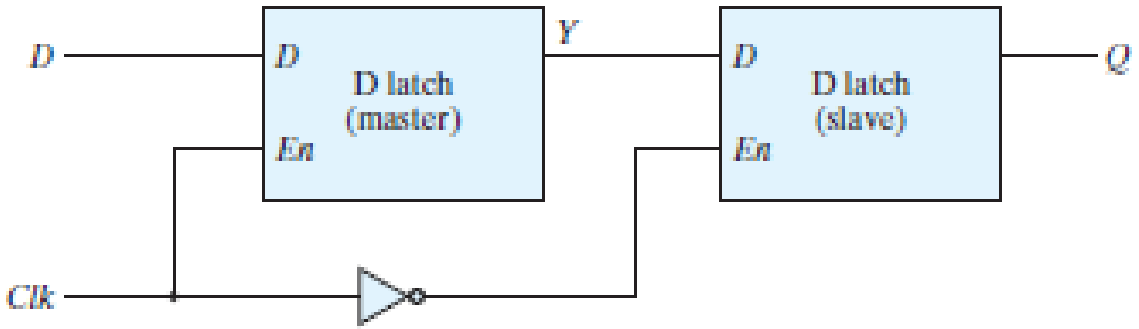
* no gate delays

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.



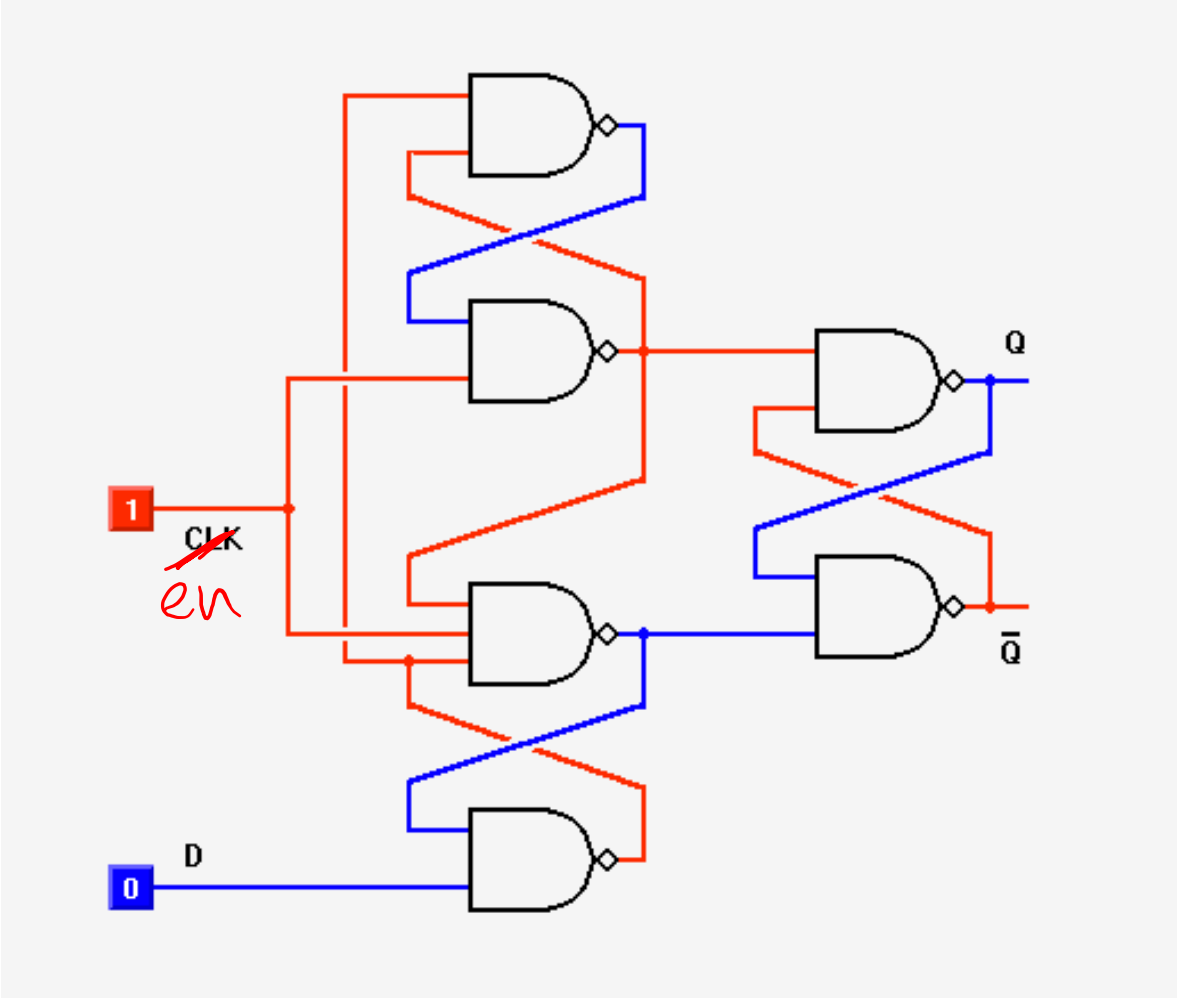
neg edge!

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.

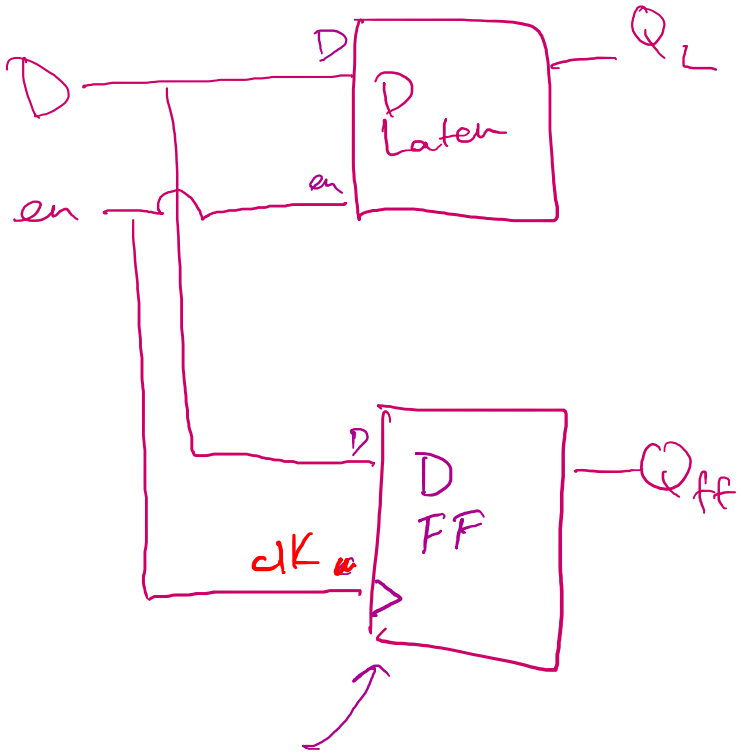
When the *Clk* returns to 0, the master is disabled and is isolated from the *D* input. At the same time, the slave is enabled and the value of *Y* is transferred to the output of the flip-flop at *Q* . Thus, a change in the output of the flip-flop can be triggered only by and during the ***transition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

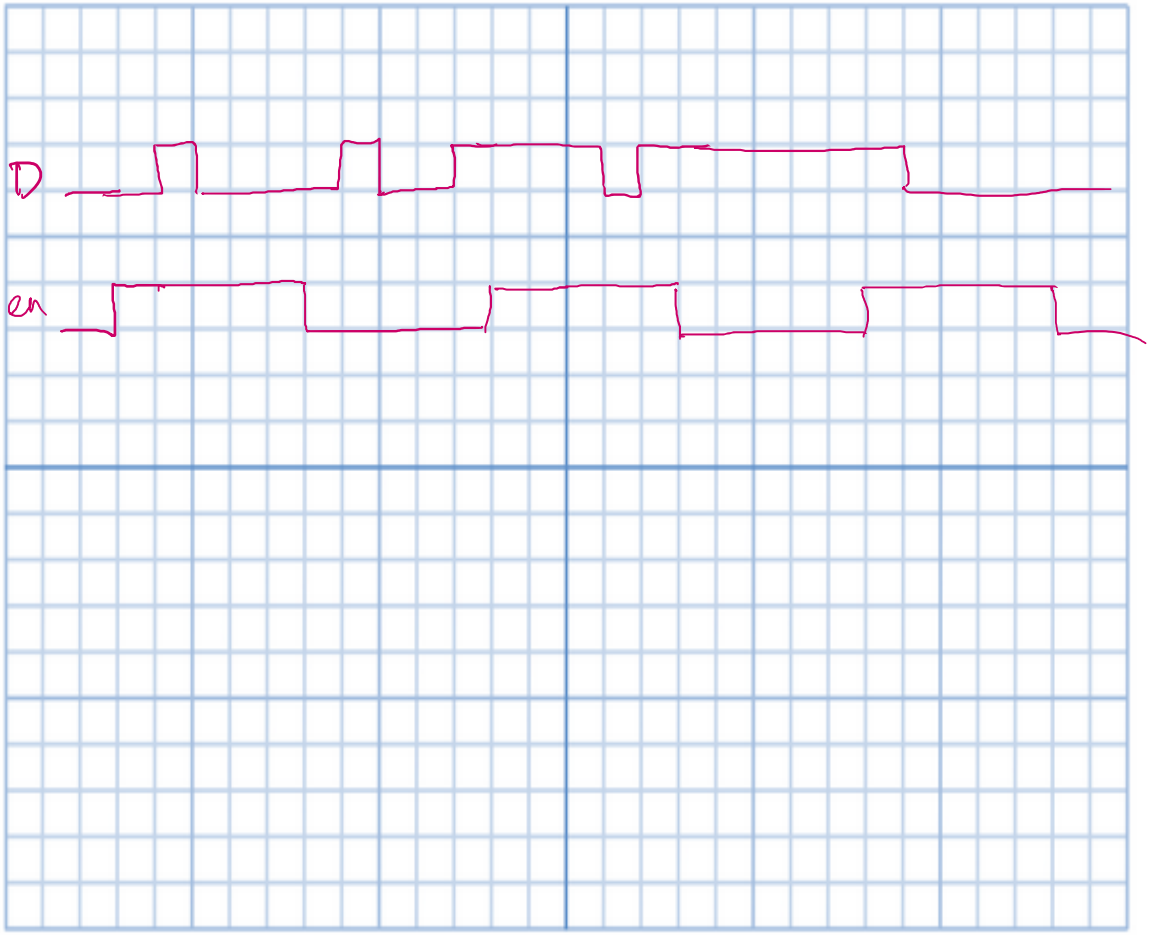
Another D Flip-Flop

P2
fb filo



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
                  q <= ~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

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

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

Blocking vs. Non-Blocking Assignments

- MY RULES:
- ONLY USE BLOCKING (=) FOR COMBINATIONAL LOGIC
- ONLY USE NON-BLOCKING (<=) FOR SEQUENTIAL LOGIC
- Disregard what you see/find on the Internet!