



Electronics Systems (938II)

Lecture 2.3

Building Blocks of Electronic Systems – Latch, Flip-flop, Register

Sequential logic

- Sequential circuits that have “memory”, because its output depends on
 - Current input(s)
 - Previous output
 - Defined as current state
- Already mentioned in previous lectures, the most notorious and used sequential circuit is the register
 - Store data

Sequential logic

- A register consists of one (or more)
 - D flip-flop (or DFF)
 - The D flip-flop is built upon the D latch
 - The D latch is built upon the SR latch, that can be easily built using CMOS gates

Sequential logic

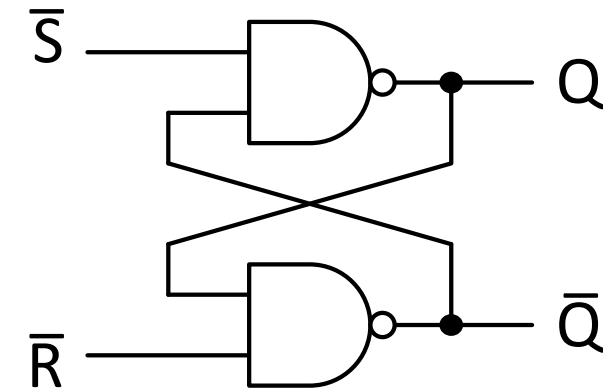
- A register consists of one (or more)
 - **D flip-flop** (or DFF)
 - The D flip-flop is built upon the **D latch**
 - The D latch is built upon the **SR latch**, that can be easily built using CMOS gates
- Let's see how they are made, starting from the base, the SR latch!

SR latch

- Can be defined as a circuit that
 - Takes in input the signals S and R
 - S = Set
 - R = Reset
 - Has output(s)
 - According to the inputs name
 - When **S = 1** and **R = 0** → set command → **output = 1**
 - When **S = 0** and **R = 1** → reset (or clear) command → **output = 0**
 - Otherwise → keep previous output (= latch)

SR latch

- Recalling functionality
 - **$S = 1, R = 0 \rightarrow \text{output} = 1$**
 - **$S = 0, R = 1 \rightarrow \text{output} = 0$**
 - Otherwise **$\rightarrow \text{latch}$**
- Please, note that
 - \bar{S} is the complement (or binary inverse) of S
 - \bar{R} is the complement (or binary inverse) of R



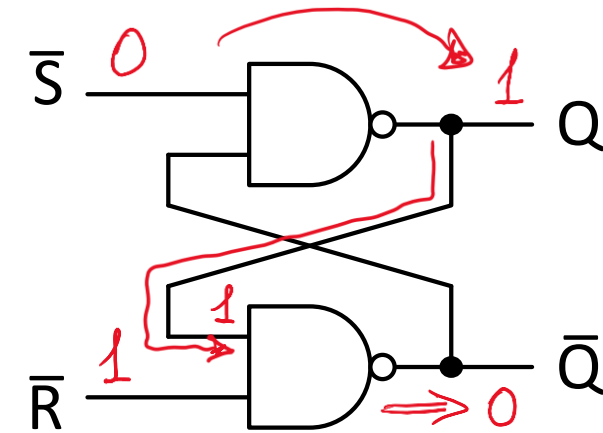
SR latch

- Recalling functionality

- $S = 1, R = 0 \rightarrow \text{output} = 1$
- $S = 0, R = 1 \rightarrow \text{output} = 0$
- Otherwise \rightarrow latch

- Please, note that

- \bar{S} is the complement (or binary inverse) of S
- \bar{R} is the complement (or binary inverse) of R

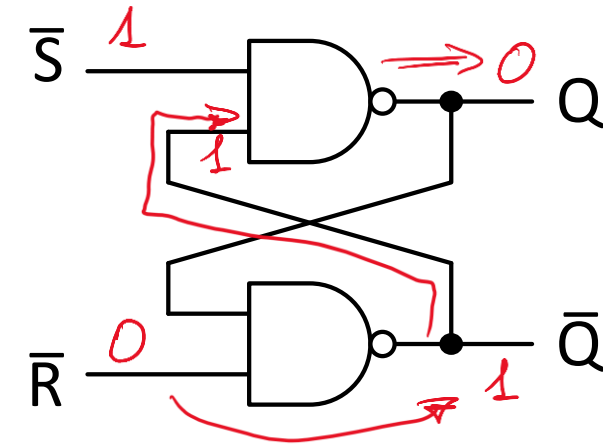


\bar{S}	\bar{R}	Q	\bar{Q}	
0	1	1	0	\rightarrow Set command (as expected)

\uparrow Q is the primary output

SR latch

- Recalling functionality
 - $S = 1, R = 0 \rightarrow \text{output} = 1$
 - $S = 0, R = 1 \rightarrow \text{output} = 0$
 - Otherwise \rightarrow latch
- Please, note that
 - \bar{S} is the complement (or binary inverse) of S
 - \bar{R} is the complement (or binary inverse) of R



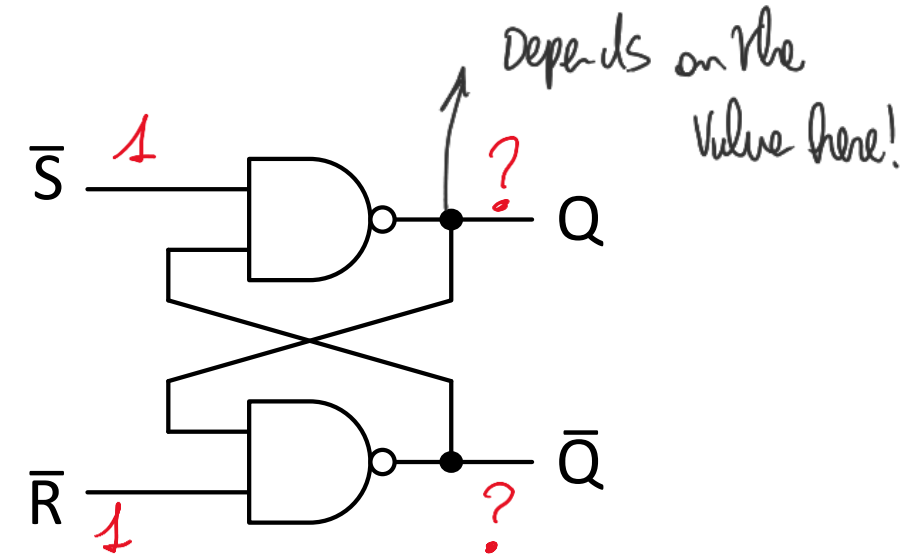
\bar{S}	\bar{R}	Q	\bar{Q}
0	1	1	0
1	0	0	1

➡ Set command (as expected)

➡ Reset command (as expected)

SR latch

- Recalling functionality
 - $S = 1, R = 0 \rightarrow \text{output} = 1$
 - $S = 0, R = 1 \rightarrow \text{output} = 0$
 - Otherwise \rightarrow latch
- Please, note that
 - \bar{S} is the complement (or binary inverse) of S
 - \bar{R} is the complement (or binary inverse) of R



\bar{S}	\bar{R}	Q	\bar{Q}
0	1	1	0
1	0	0	1
1	1	??	??

➡ Set command (as expected)

➡ Reset command (as expected)

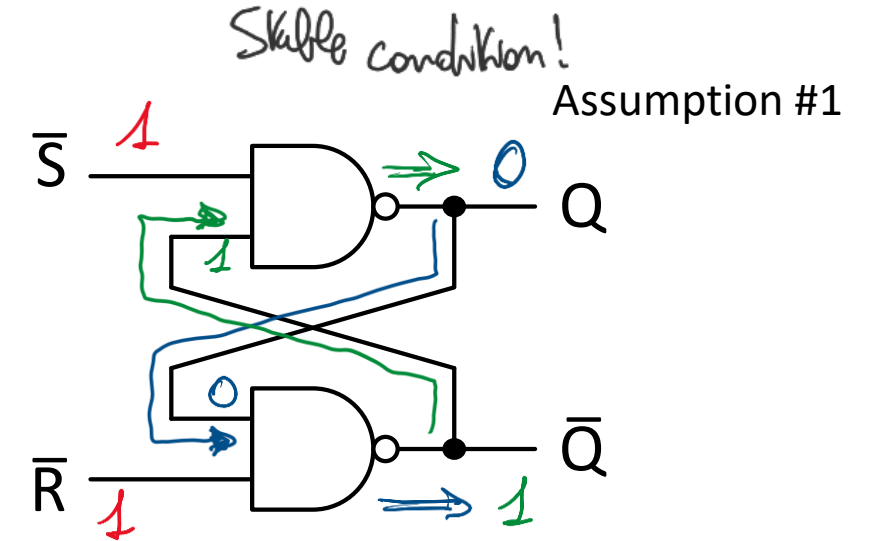
SR latch

- Recalling functionality

- $S = 1, R = 0 \rightarrow \text{output} = 1$
- $S = 0, R = 1 \rightarrow \text{output} = 0$
- Otherwise \rightarrow latch

- Please, note that

- \bar{S} is the complement (or binary inverse) of S
- \bar{R} is the complement (or binary inverse) of R



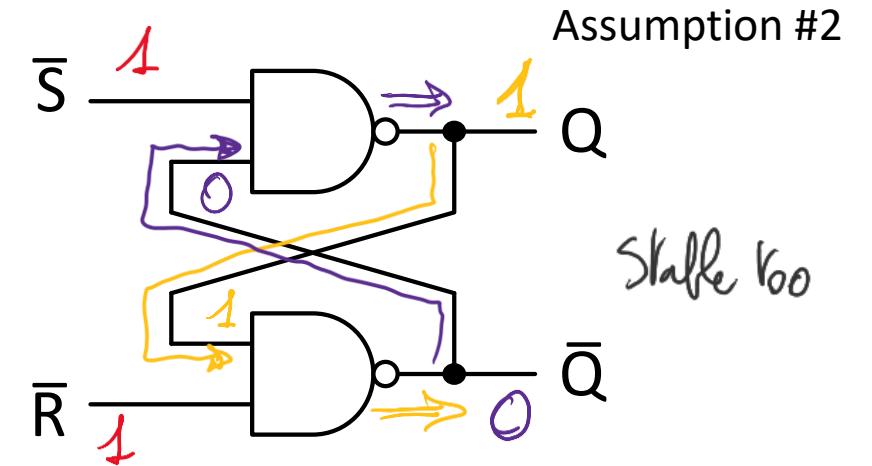
\bar{S}	\bar{R}	Q	\bar{Q}
0	1	1	0
1	0	0	1
1	1	??	??

➡ Set command (as expected)

➡ Reset command (as expected)

SR latch

- Recalling functionality
 - $S = 1, R = 0 \rightarrow \text{output} = 1$
 - $S = 0, R = 1 \rightarrow \text{output} = 0$
 - Otherwise \rightarrow latch
- Please, note that
 - \bar{S} is the complement (or binary inverse) of S
 - \bar{R} is the complement (or binary inverse) of R



\bar{S}	\bar{R}	Q	\bar{Q}
0	1	1	0
1	0	0	1
1	1	??	??

➡ Set command (as expected)

➡ Reset command (as expected)

SR latch

- Recalling functionality

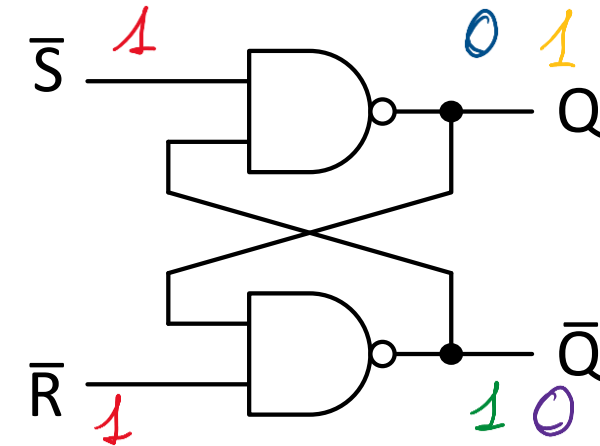
- $S = 1, R = 0 \rightarrow \text{output} = 1$
- $S = 0, R = 1 \rightarrow \text{output} = 0$
- Otherwise \rightarrow latch

- Please, note that

- \bar{S} is the complement (or binary inverse) of S
- \bar{R} is the complement (or binary inverse) of R

- Both assumptions hold. In other words, when $S = 0$ ($\bar{S} = 1$) and $R = 0$ ($\bar{R} = 1$), the latch keep the previous output(s): Q^* and \bar{Q}^*

- And in both cases $\bar{Q} = \text{inverse of } Q$



\bar{S}	\bar{R}	Q	\bar{Q}	
0	1	1	0	➡ Set command (as expected)
1	0	0	1	➡ Reset command (as expected)
1	1	Q^*	\bar{Q}^*	➡ Latch (as expected)

↑ previous value

SR latch

No need to make assumptions

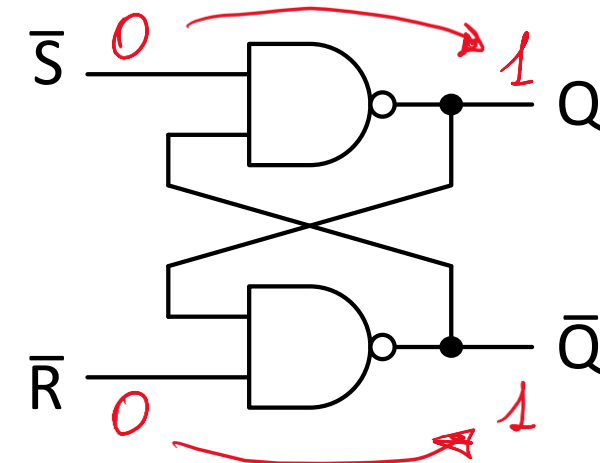
- Recalling functionality

- $S = 1, R = 0 \rightarrow \text{output} = 1$
- $S = 0, R = 1 \rightarrow \text{output} = 0$
- Otherwise \rightarrow latch

- Please, note that

- \bar{S} is the complement (or binary inverse) of S
- \bar{R} is the complement (or binary inverse) of R

- In this case, Q and \bar{Q} are forced to 1, and, in addition, $\bar{Q} = Q$!!!

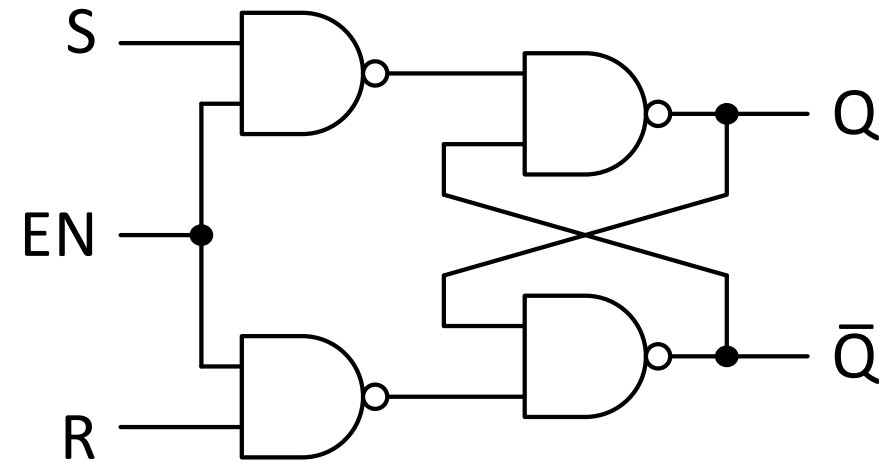


\bar{S}	\bar{R}	Q	\bar{Q}
0	1	1	0
1	0	0	1
1	1	Q^*	\bar{Q}^*
0	0	1	1

- ➔ Set command (as expected)
- ➔ Reset command (as expected)
- ➔ Latch (as expected)
- ➔ **Forbidden!** ($\bar{Q} = Q$)

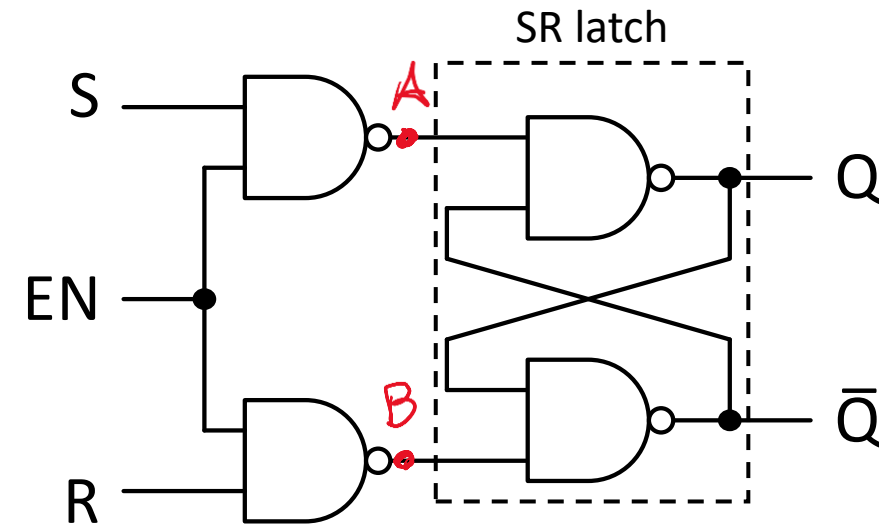
SR latch with Enable

- Let's add an enable signal (EN) to the SR latch
 - SR latch with Enable



SR latch with Enable

- Looking at the circuit
 - $A = \bar{S}$ of SR latch
 - $B = \bar{R}$ of SR latch

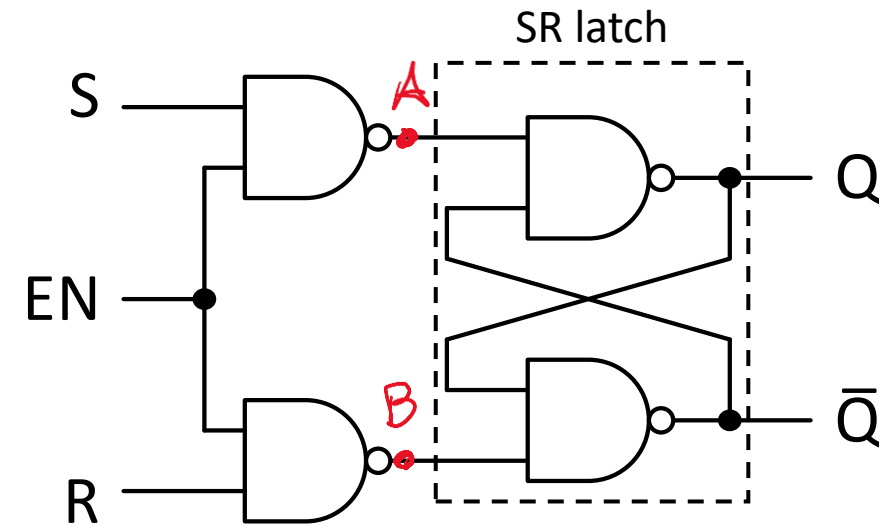


\bar{S}	\bar{R}	Q	\bar{Q}
0	1	1	0
1	0	0	1
1	1	Q^*	\bar{Q}^*
0	0	1	1

SR latch truth table

SR latch with Enable

- Looking at the circuit
 - $A = \bar{S}$ of SR latch
 - $B = \bar{R}$ of SR latch
- If $EN = 0$, $A = 1$ and $B = 1$
 - $\bar{S} = 1$ and $\bar{R} = 1 \rightarrow$ SR latch: latch state

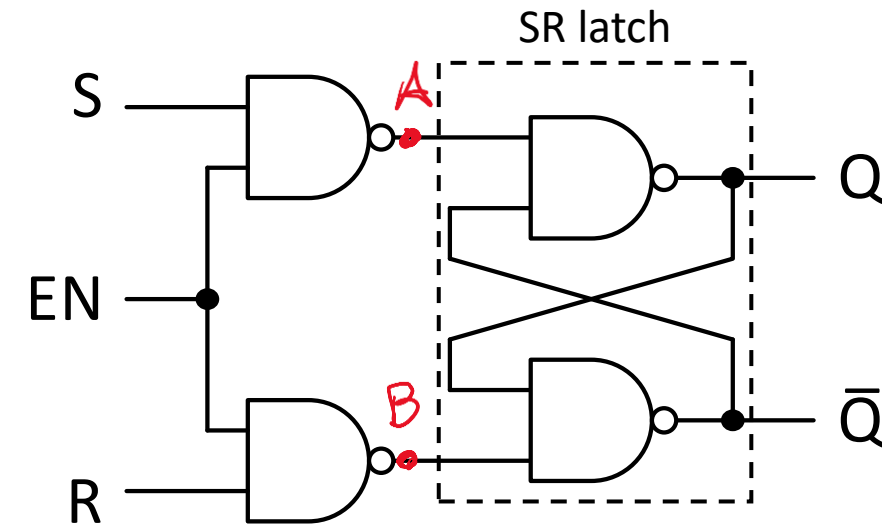


\bar{S}	\bar{R}	Q	\bar{Q}
0	1	1	0
1	0	0	1
1	1	Q^*	\bar{Q}^*
0	0	1	1

SR latch truth table

SR latch with Enable

- Looking at the circuit
 - $A = \bar{S}$ of SR latch
 - $B = \bar{R}$ of SR latch
- If $EN = 0$, $A = 1$ and $B = 1$
 - $\bar{S} = 1$ and $\bar{R} = 1 \rightarrow$ SR latch: latch state



\bar{S}	\bar{R}	Q	\bar{Q}
0	1	1	0
1	0	0	1
1	1	Q^*	\bar{Q}^*
0	0	1	1

SR latch truth table

- If $EN = 1$
 - $A = \bar{S} \cdot \overline{EN} = \bar{S} + \overline{EN} = \bar{S} + \bar{1} = \bar{S} + 0 = \bar{S}$
 - $B = \bar{R} \cdot \overline{EN} = \bar{R} + \overline{EN} = \bar{R} + \bar{1} = \bar{R} + 0 = \bar{R}$
 - It behaves the same as before!

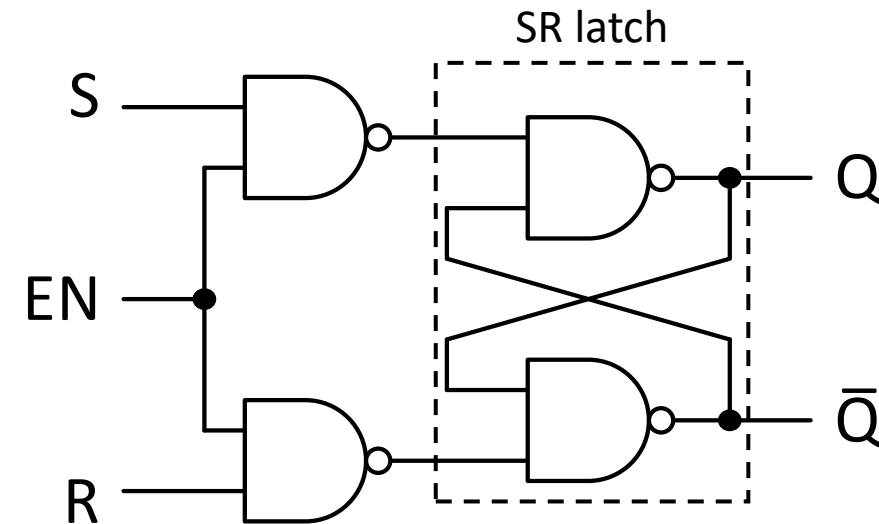
SR latch with Enable

- If $EN = 0$, $A = 1$ and $B = 1$
 - $\bar{S} = 1$ and $\bar{R} = 1 \rightarrow$ SR latch: latch state
- If $EN = 1$
 - It behaves the same as before!

↓

EN	\bar{S}	\bar{R}	S	R	Q	\bar{Q}
0	1	1	X	X	Q^*	\bar{Q}^*
1	0	1	1	0	1	0
1	1	0	0	1	0	1
1	1	1	0	0	Q^*	\bar{Q}^*
1	0	0	1	1	1	1

Any value
"don't care"



\bar{S}	\bar{R}	Q	\bar{Q}
0	1	1	0
1	0	0	1
1	1	Q^*	\bar{Q}^*
0	0	1	1

SR latch truth table

SR latch with Enable

- Recap

Truth table of SR latch with Enable

EN	S	R	Q	\bar{Q}
0	X	X	Q^*	\bar{Q}^*
1	1	0	1	0
1	0	1	0	1
1	0	0	Q^*	\bar{Q}^*
1	1	1	1	1

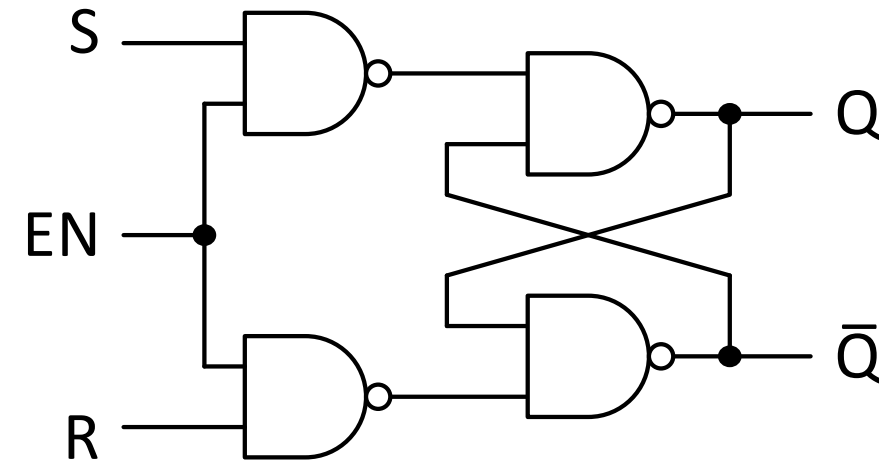
➡ Latch (because 'disabled': EN = 0)

➡ Set command (as expected: S = 1, R = 0)

➡ Reset command (as expected: S = 0, R = 1)

➡ Latch (as expected: S = 0 and R = 0)

➡ **Forbidden!** ($\bar{Q} = Q = 1$)



D latch

- Can be defined as a circuit that
 - Takes in input the signals D and EN
 - D = Data
 - EN = Enable
 - Has output(s)
 - Has then following functionality:
 - When EN = 0 → latch (keep previous output(s))
 - When EN = 1 → D sets the primary output (Q): $Q = D$

D latch

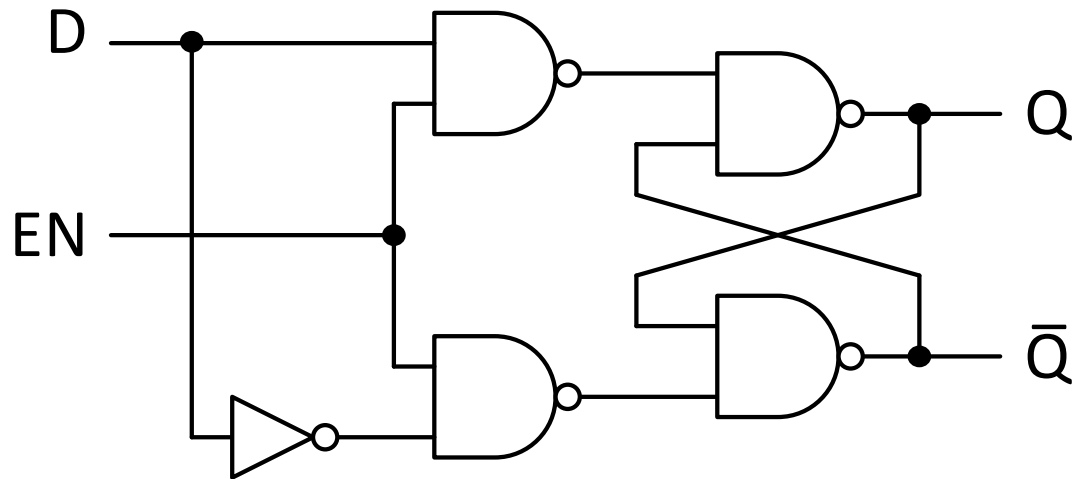
- Recalling functionality
 - When **EN = 0** → **latch** (keep previous output(s))
 - When **EN = 1** → D sets the primary output (Q): $Q = D$
- It can be built using the SR latch with Enable
 - When **EN = 0** → **latch**
 - When **EN = 1** → D sets the primary output (Q): $Q = D$
 - $D = 1 \rightarrow Q = 1 \rightarrow S = 1, R = 0$
 - $D = 0 \rightarrow Q = 0 \rightarrow S = 0, R = 1$

D latch

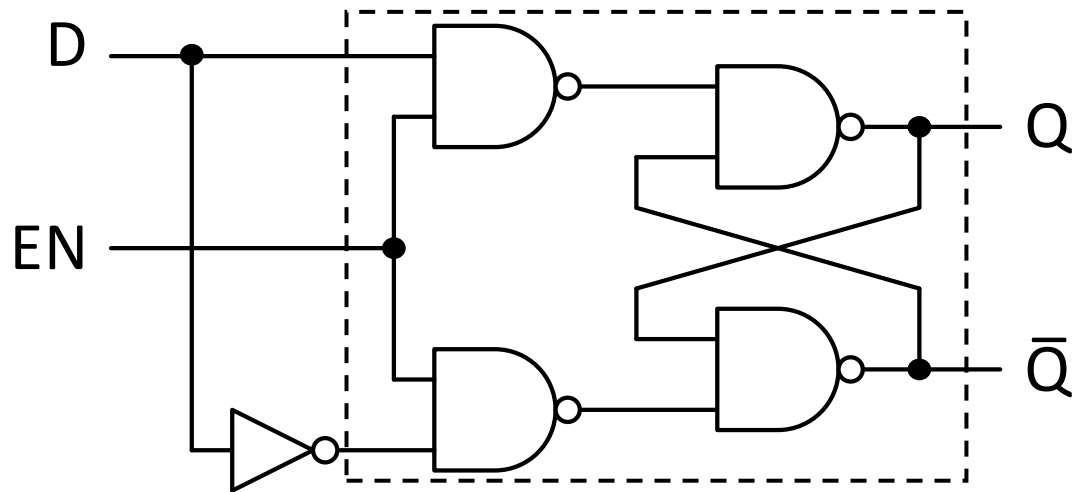
- Recalling functionality
 - When **EN = 0** → **latch** (keep previous output(s))
 - When **EN = 1** → D sets the primary output (Q): $Q = D$
- It can be built using the SR latch with Enable
 - When **EN = 0** → **latch**
 - When **EN = 1** → D sets the primary output (Q): $Q = D$
 - $D = 1 \rightarrow Q = 1 \rightarrow S = 1, R = 0$
 - $D = 0 \rightarrow Q = 0 \rightarrow S = 0, R = 1$

$$\left. \begin{array}{l} D = 1 \rightarrow Q = 1 \rightarrow S = 1, R = 0 \\ D = 0 \rightarrow Q = 0 \rightarrow S = 0, R = 1 \end{array} \right\} \rightarrow \left\{ \begin{array}{l} S = D \\ R = \bar{D} \end{array} \right.$$

D latch



D latch

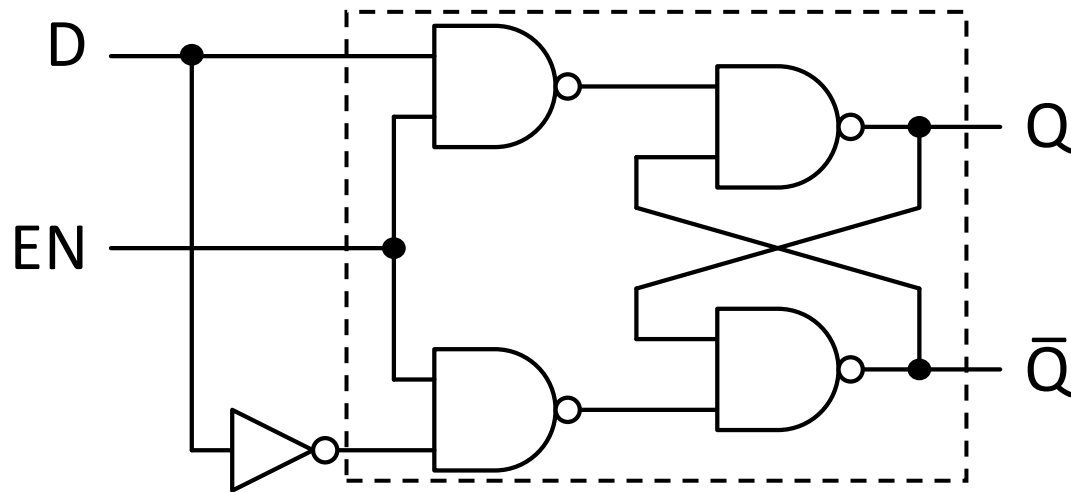


SR latch with Enable

EN	S	R	Q	\bar{Q}
0	X	X	Q^*	\bar{Q}^*
1	1	0	1	0
1	0	1	0	1
1	0	0	Q^*	\bar{Q}^*
1	1	1	1	1

Truth table of SR latch with Enable

D latch



SR latch with Enable

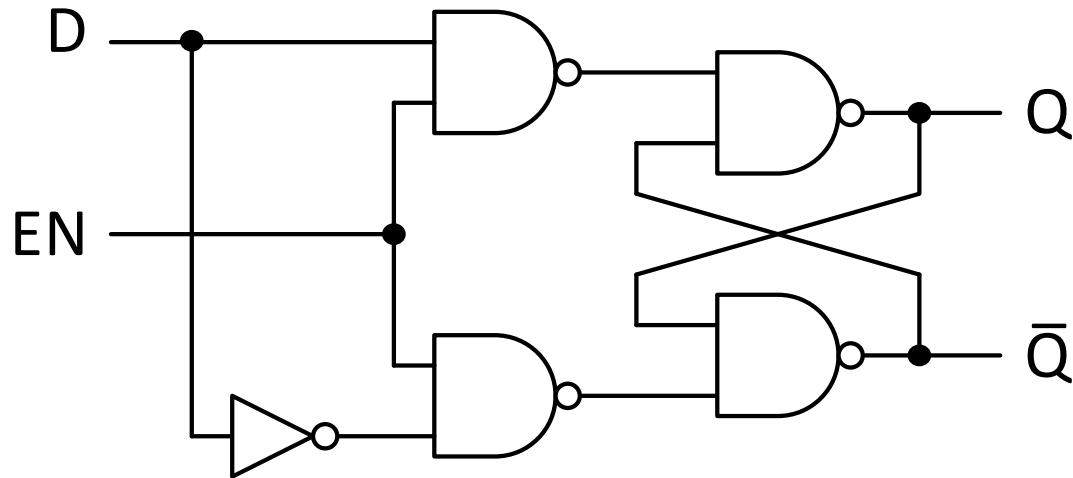
EN	S	R	Q	\bar{Q}
0	X	X	Q^*	\bar{Q}^*
1	1	0	1	0
1	0	1	0	1
1	0	0	Q^*	\bar{Q}^*
1	1	1	1	1

Truth table of SR latch with Enable

- If $EN = 0$, it acts like before (SR latch with Enable)
- If $EN = 1$
 - If $D = 0 \rightarrow S = 0$ and $R = 1 \rightarrow$ Reset command ($Q = 0$)
 - If $D = 1 \rightarrow S = 1$ and $R = 0 \rightarrow$ Set command ($Q = 1$)

D latch

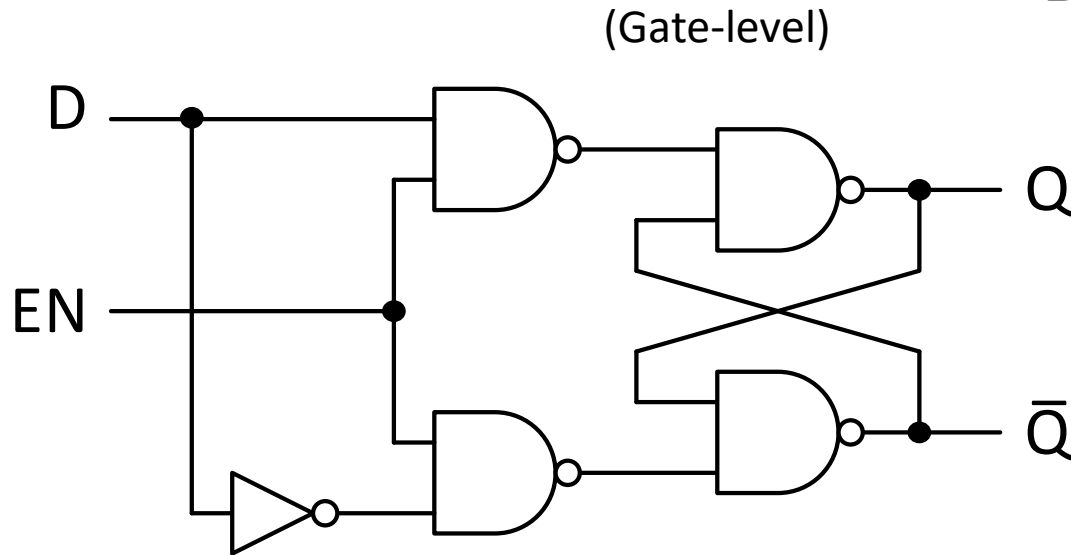
(Gate-level)



(Truth table)

EN	D	Q	\bar{Q}
0	X	Q^*	\bar{Q}^*
1	0	0	1
1	1	1	0

D latch



IDC about the behaviour of D

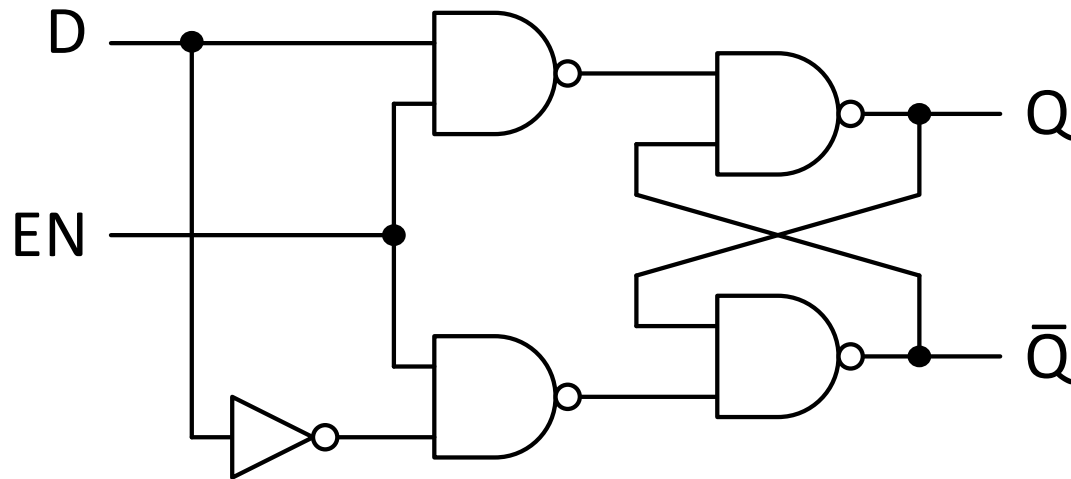
(Truth table)

EN	D	Q	\bar{Q}
0	X	Q^*	\bar{Q}^*
1	0	0	1
1	1	1	0

- In addition, the forbidden condition of the SR latch never occurs
 - Thanks to the NOT gate, whatever the value of D, it happens that
 - Either $S = 1$ and $R = 0$
 - Or $S = 0$ and $R = 1$
 - It can never happen neither $S = 0$ and $R = 0$, nor $S = 1$ and $R = 1$ *(also safety measure)*

D latch

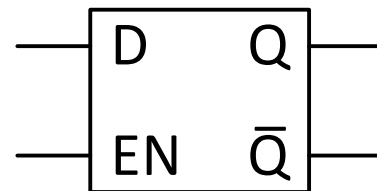
(Gate-level)



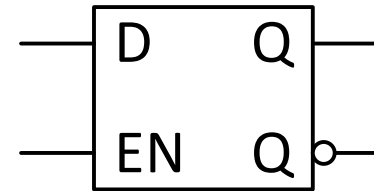
(Truth table)

EN	D	Q	\bar{Q}
0	X	Q^*	\bar{Q}^*
1	0	0	1
1	1	1	0

- Logic symbol(s)



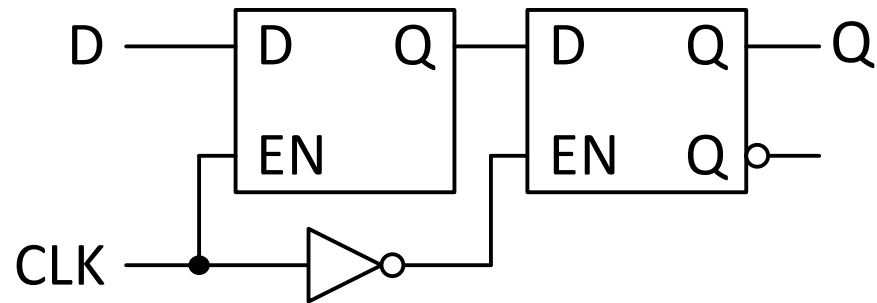
(RTL)



our will be q and I negate what's on my left.

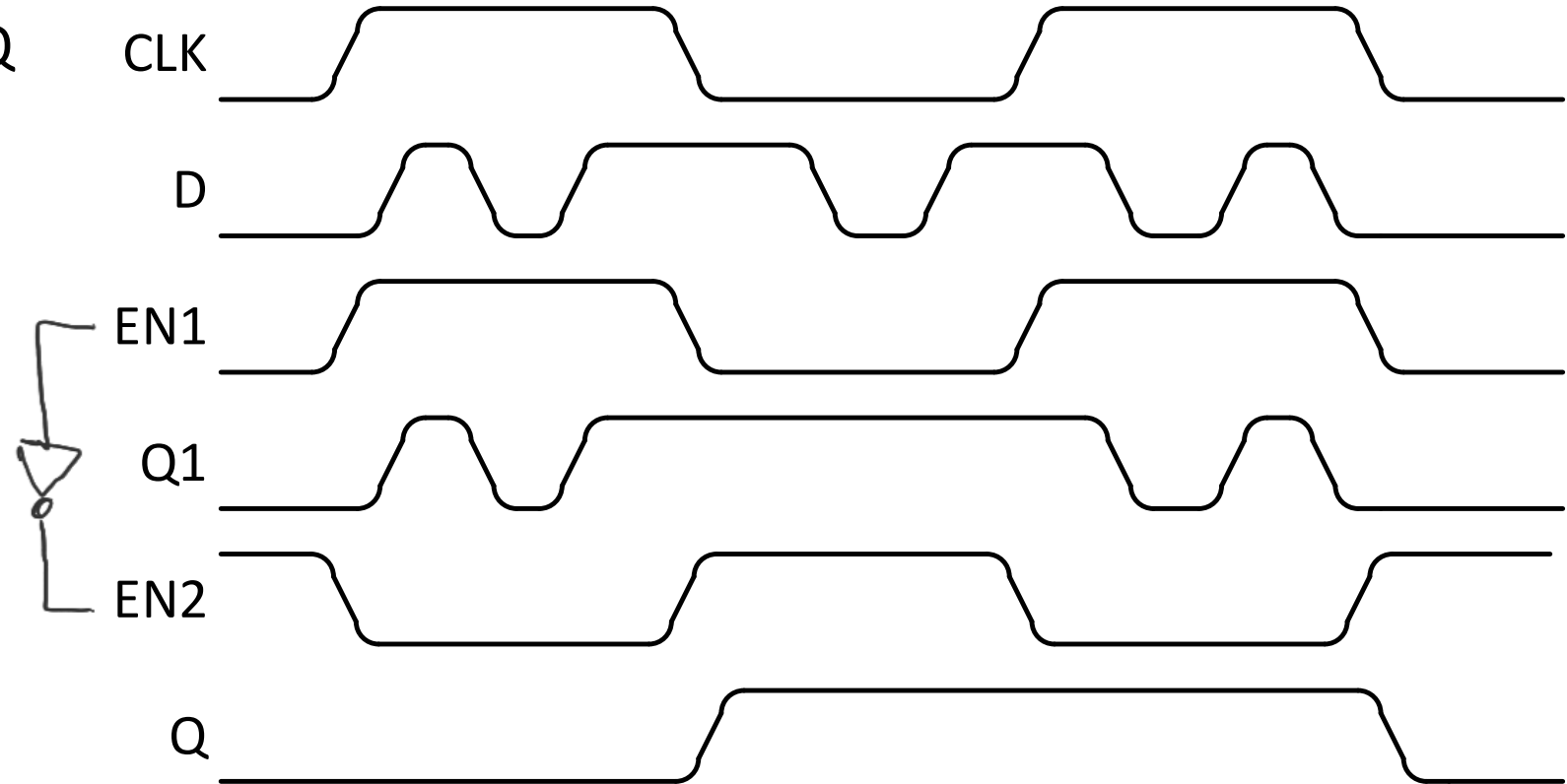
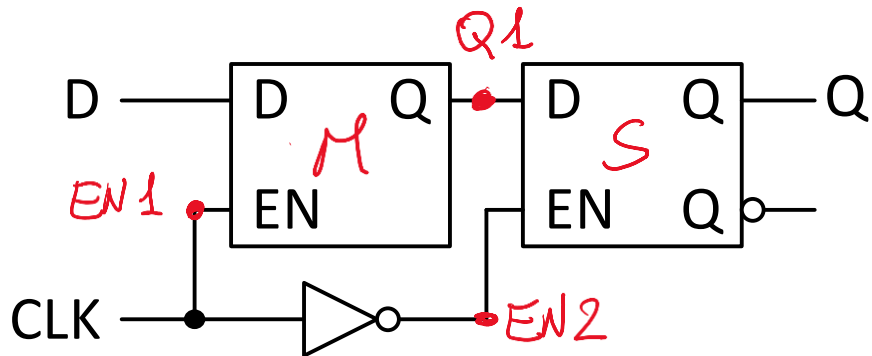
D flip-flop (or DFF)

- As any other flip-flop, it is built by cascading two latches (of the same type)
 - Two D latches in this case

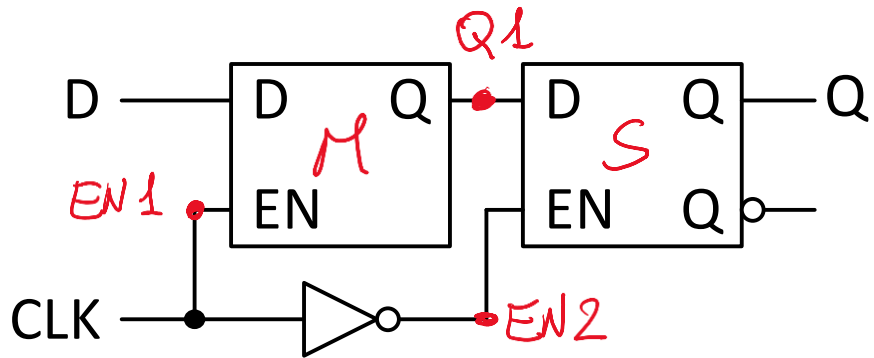


- Let's see what happen by using a waveform

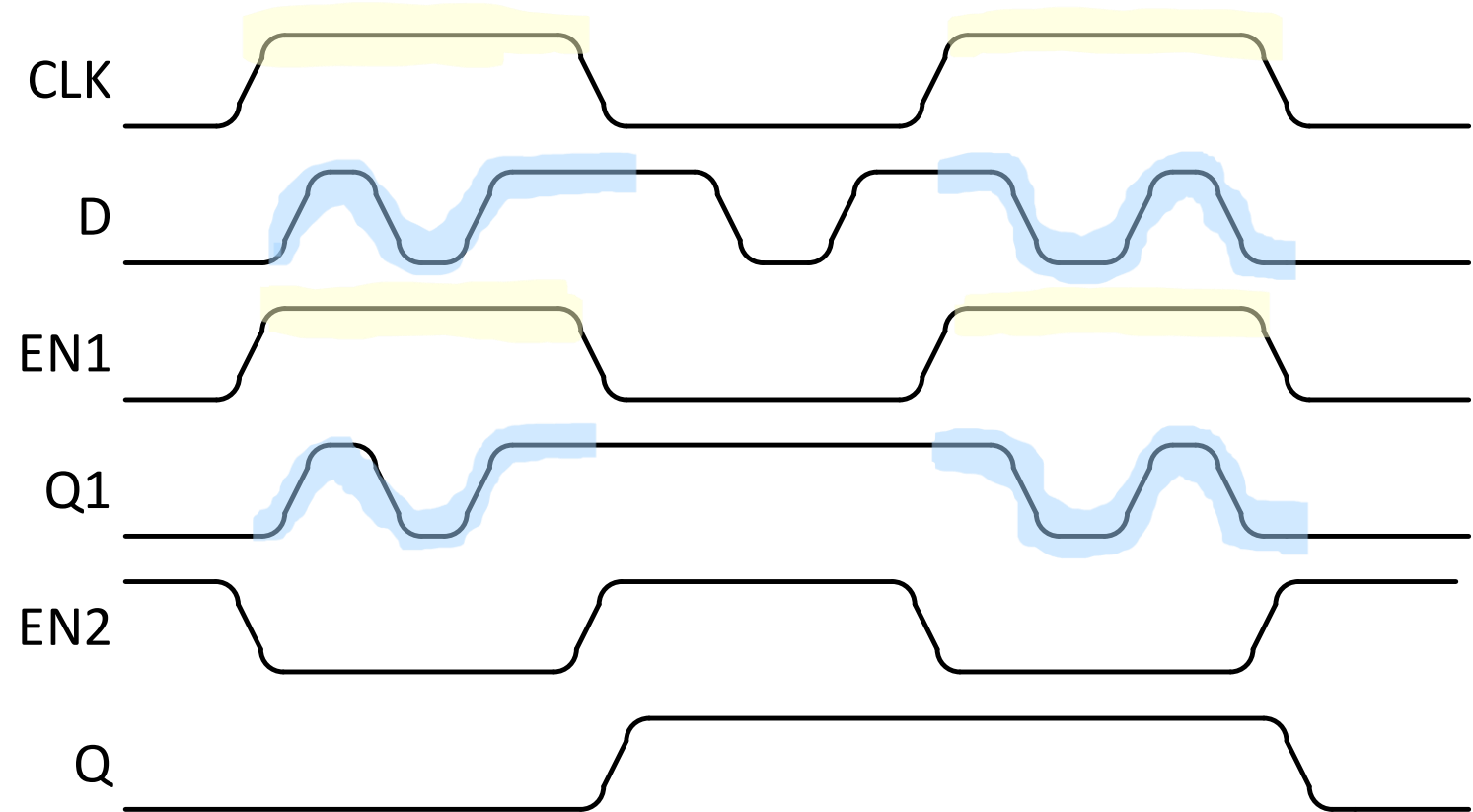
D flip-flop (or DFF)



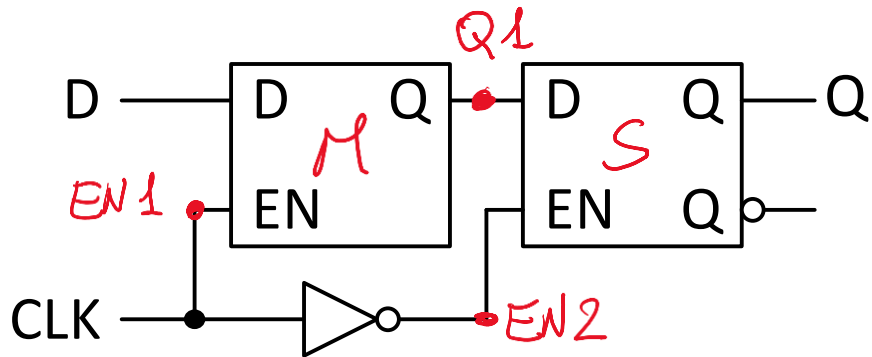
D flip-flop (or DFF)



- When $EN1 = CLK$ is 1, $Q1 = D$

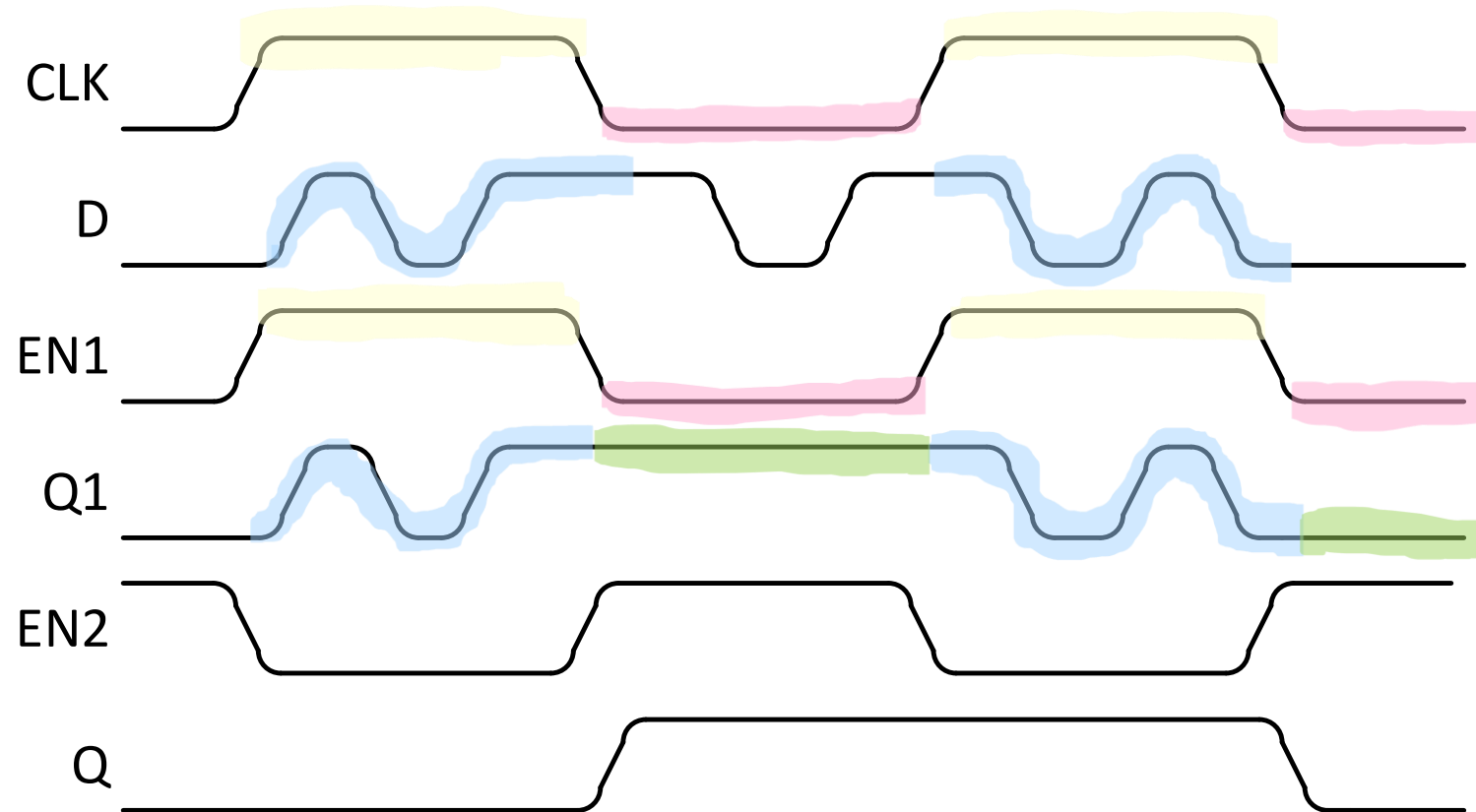


D flip-flop (or DFF)

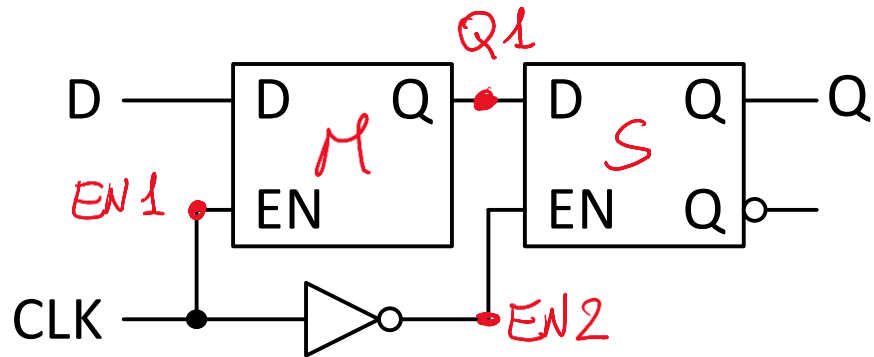


- When *EN1* = CLK is 1, *Q1* = D
- Otherwise, M latches and keep the last value of *Q1*

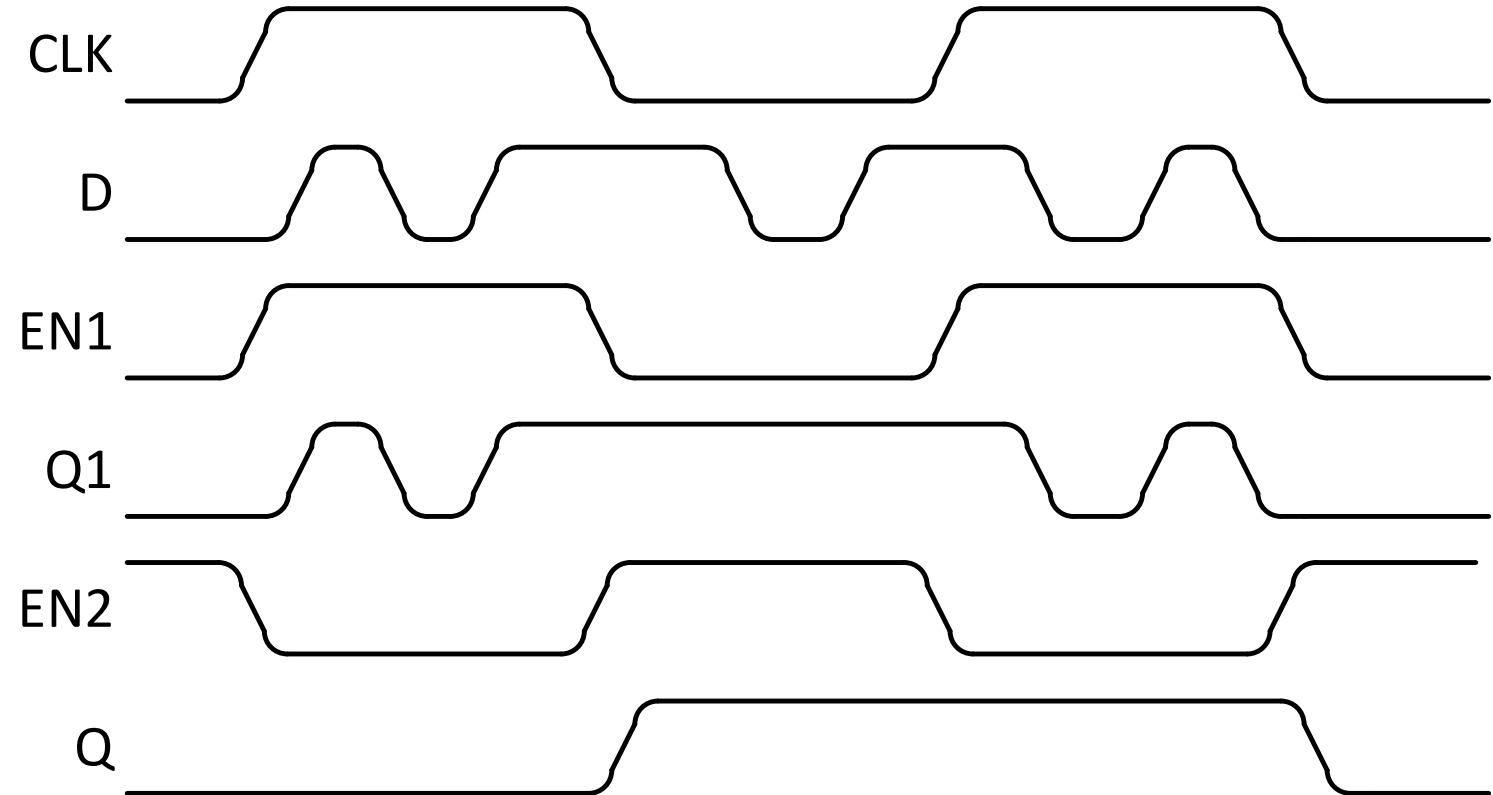
When the falling edge of CLK occurs, the output may change.



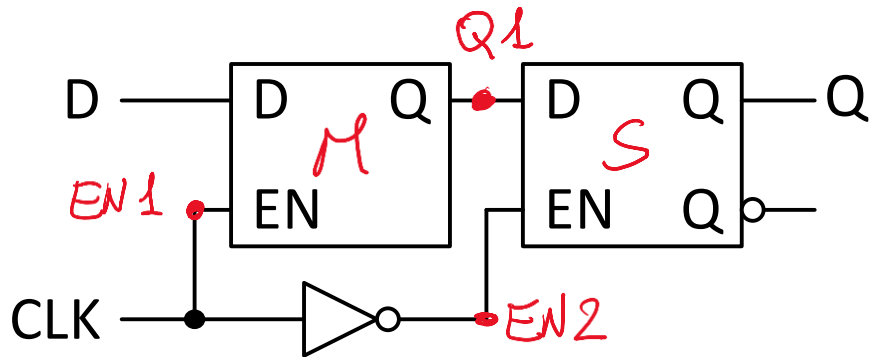
D flip-flop (or DFF)



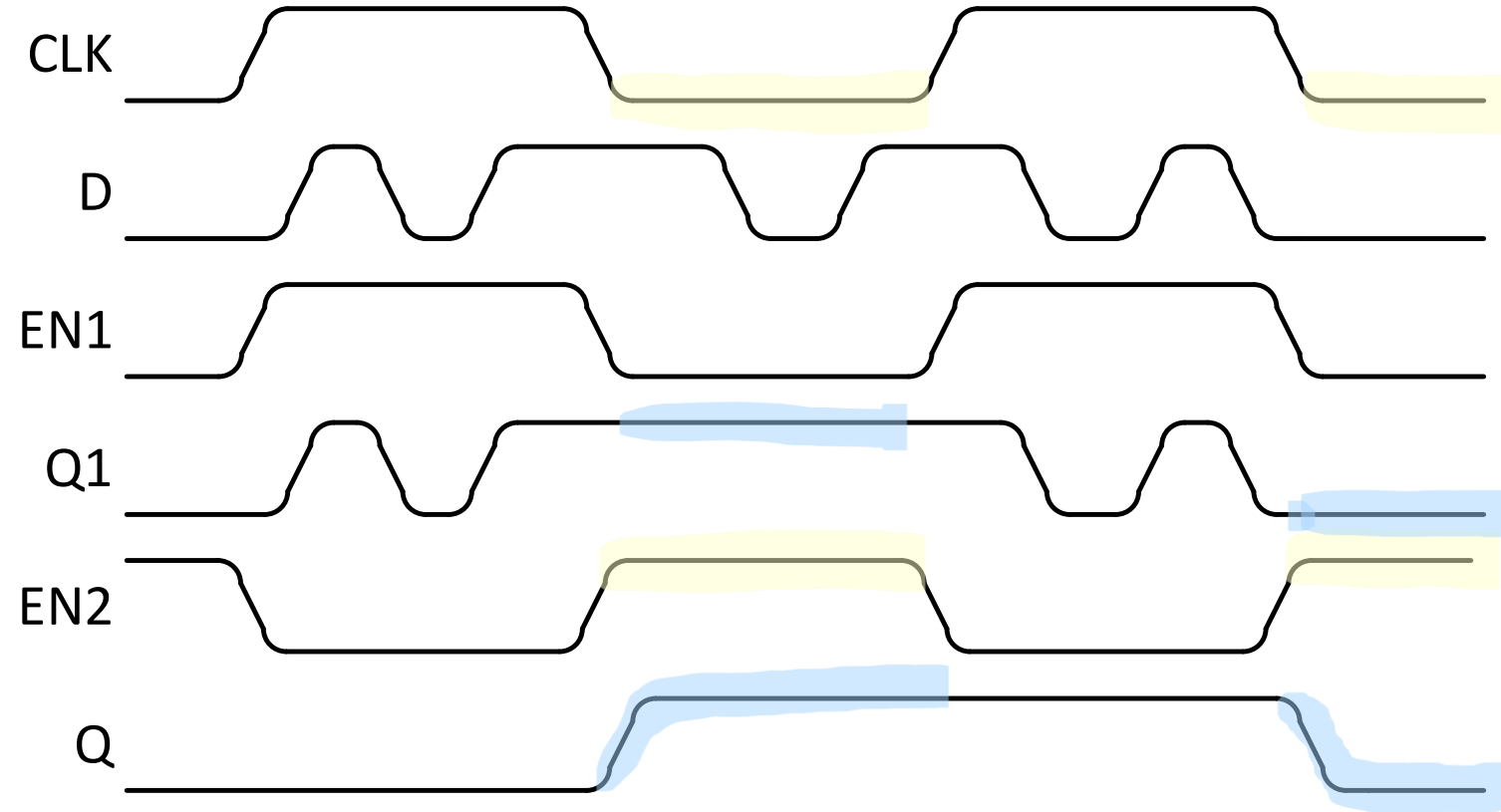
- Similarly, it happens for DFF2, but EN2 is the inverse of CLK
 - NOT gate
- So, ...



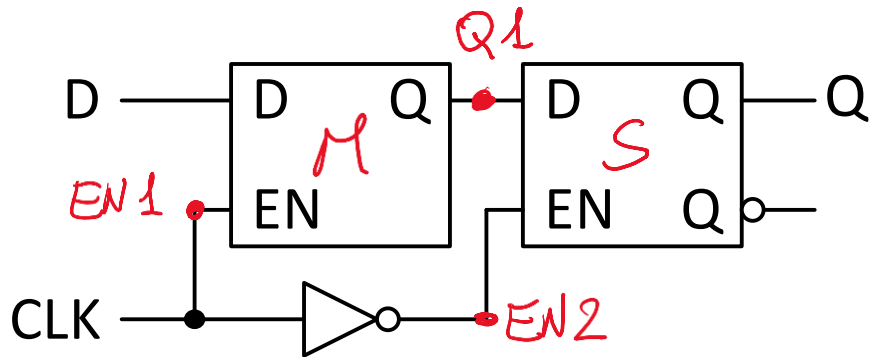
D flip-flop (or DFF)



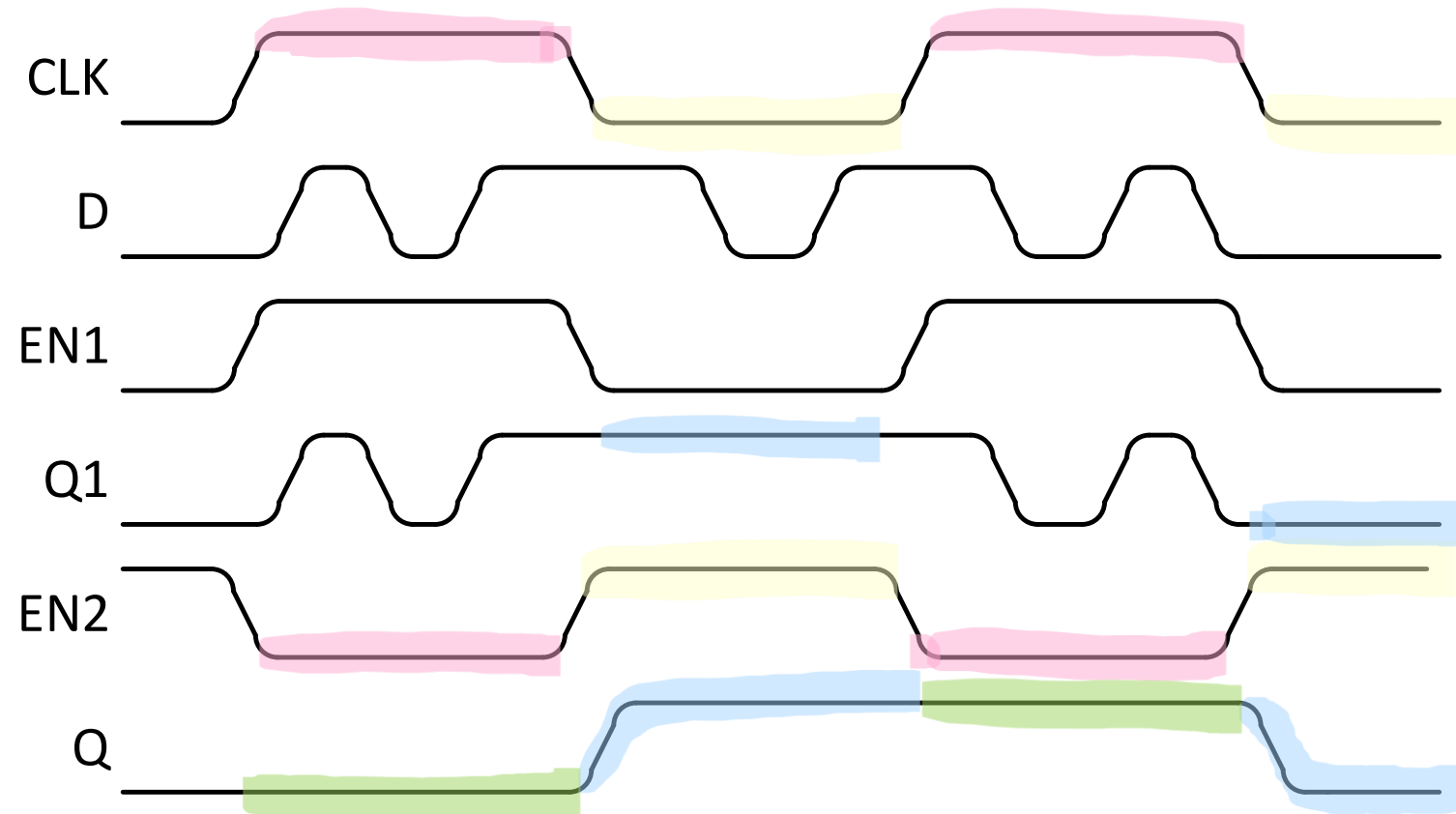
- When $CLK = 0$, $EN2 = 1$, $Q = Q1$
 - $Q1$ is the data input of S



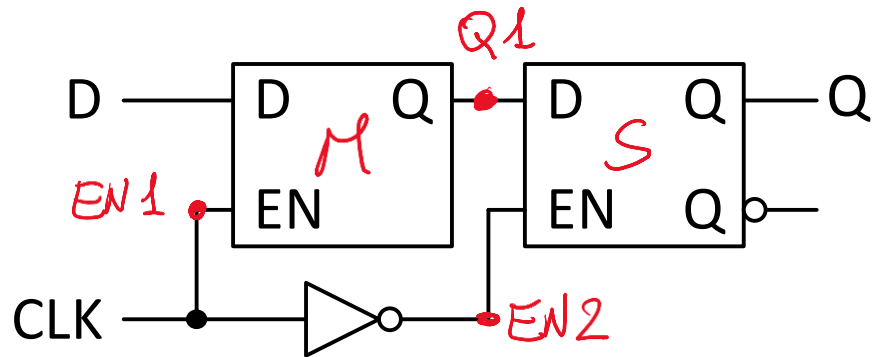
D flip-flop (or DFF)



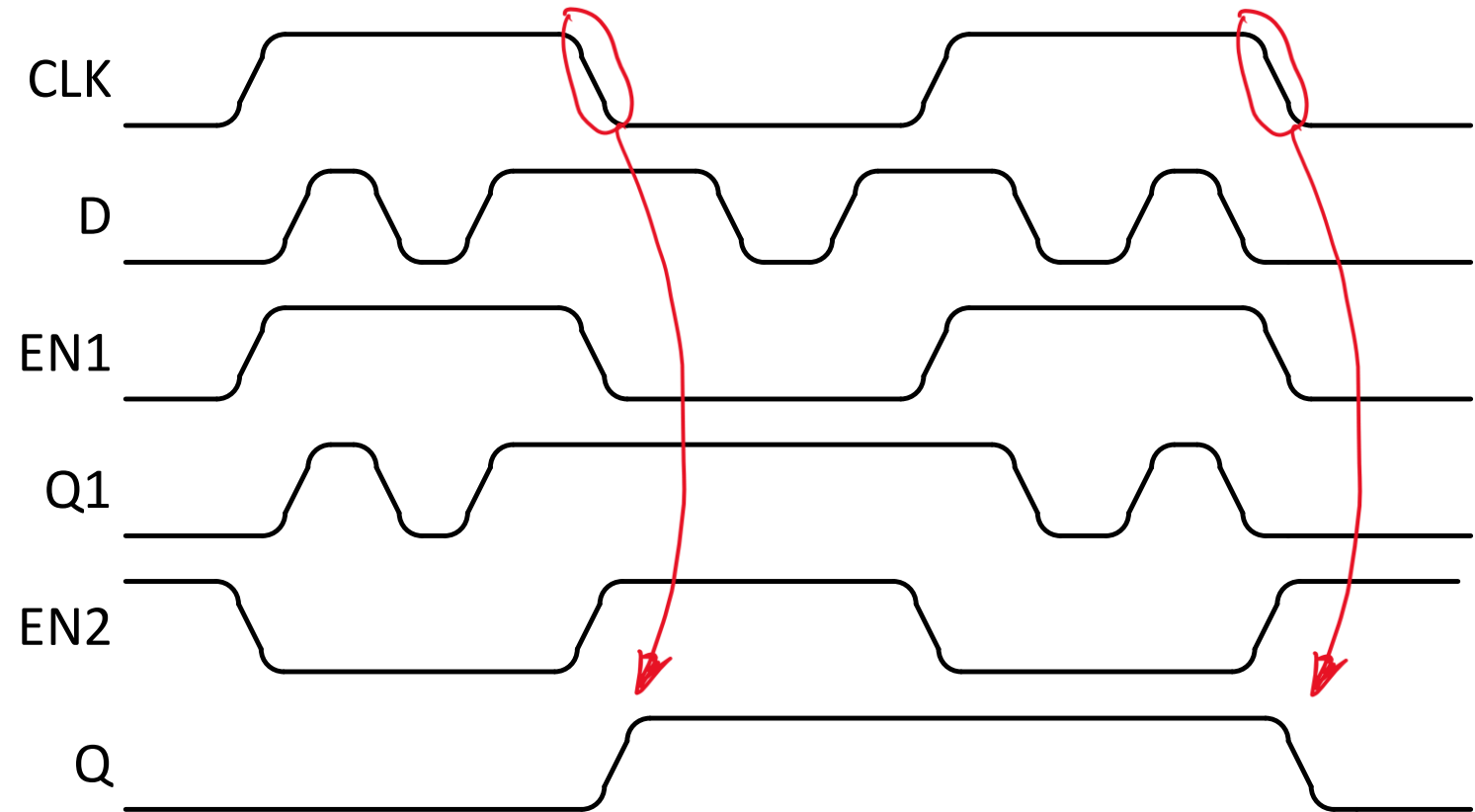
- When $CLK = 0$, $EN2 = 1$, $Q = Q1$
 - $Q1$ is the data input of S
- Otherwise, S latches and keep the last value of $Q2$



D flip-flop (or DFF)



- In other words, the (primary) output of the DFF, i.e. Q, changes only at the **falling edges** of the signal CLK



D flip-flop (or DFF)

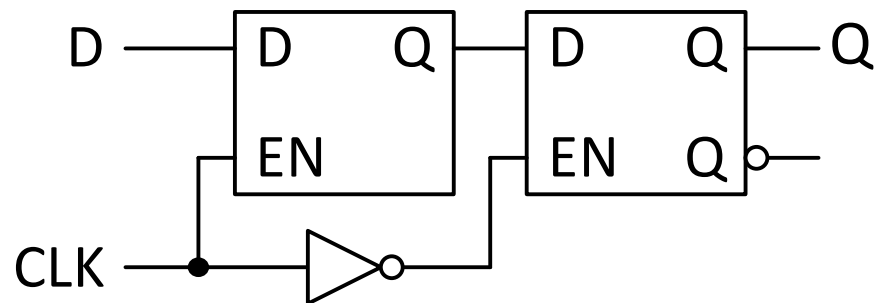
- CLK signal is typically called clock signal, and it is a periodic signal which oscillates between 0 and 1
- Since the DFF is 'activated' by the (falling) edge of CLK, it said to be **edge-triggered**
 - In particular, negative-edge-triggered, since it is sensitive to the falling edge of clock signal

Is active on an edge of the EN signal. The D Latch, the output changes when the EN level is 1 (the latch is active when the EN level is the one required)

D flip-flop (or DFF)

- CLK signal is typically called clock signal, and it is a periodic signal which oscillates between 0 and 1
- Since the DFF is 'activated' by the (falling) edge of CLK, it said to be **edge-triggered**
 - In particular, negative-edge-triggered, since it is sensitive to the falling edge of clock signal

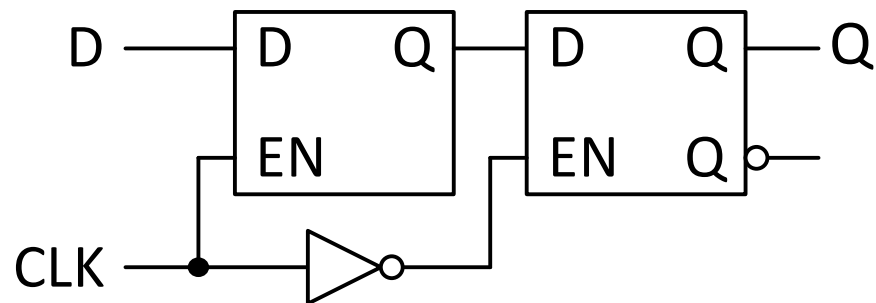
Negative-edge-triggered D flip-flop



D flip-flop (or DFF)

- CLK signal is typically called clock signal, and it is a periodic signal which oscillates between 0 and 1
- Since the DFF is 'activated' by the (falling) edge of CLK, it said to be **edge-triggered**
 - In particular, negative-edge-triggered, since it is sensitive to the falling edge of clock signal
 - A positive-edge-triggered DFF (sensitive to the rising edge of the clock signal) can be obtained by inverting CLK at the input with another NOT gate

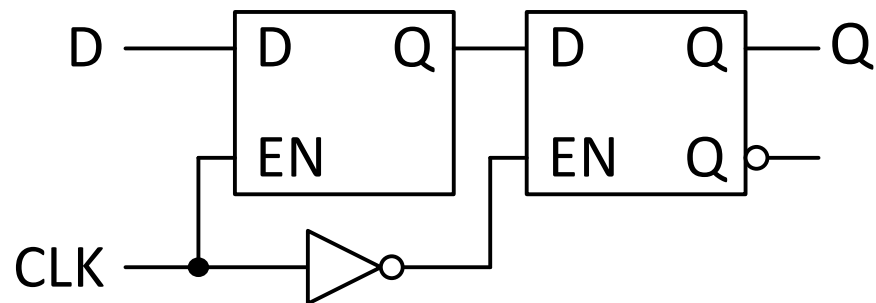
Negative-edge-triggered D flip-flop



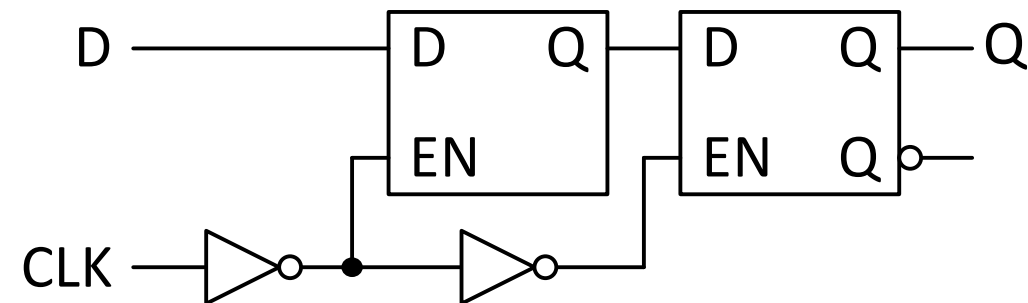
D flip-flop (or DFF)

- CLK signal is typically called clock signal, and it is a periodic signal which oscillates between 0 and 1
- Since the DFF is 'activated' by the (falling) edge of CLK, it said to be **edge-triggered**
 - In particular, negative-edge-triggered, since it is sensitive to the falling edge of clock signal
 - A positive-edge-triggered DFF (sensitive to the rising edge of the clock signal) can be obtained by inverting CLK at the input with another NOT gate

Negative-edge-triggered D flip-flop

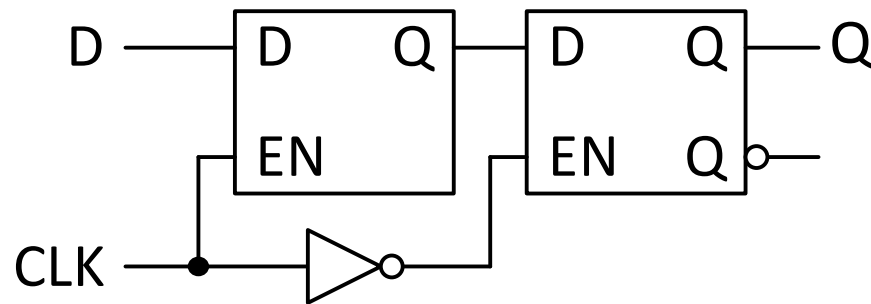


Positive-edge-triggered D flip-flop

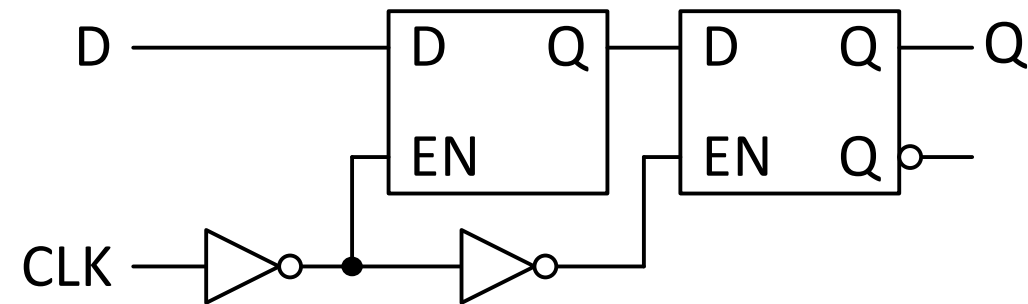


D flip-flop (or DFF)

Negative-edge-triggered D flip-flop



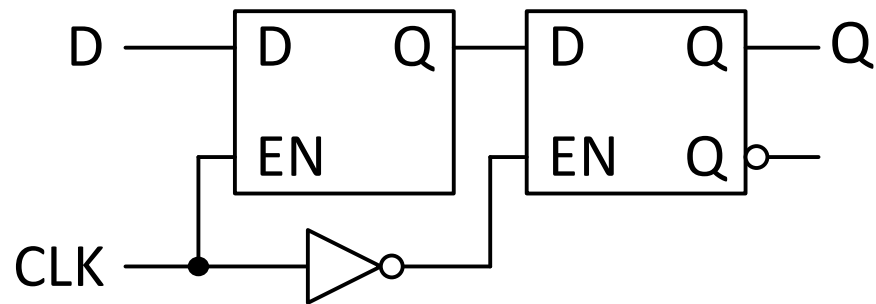
Positive-edge-triggered D flip-flop



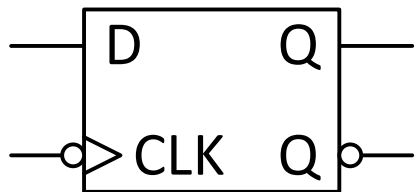
- This architecture is historically called Master-Slave architecture
 - Leftmost latch = Master, rightmost latch = Slave
 - It cannot be longer used for ethical reasons
 - Saying only because in case of further investigation you are very likely to find more matches using this historical name
 - Current name is Primary-Secondary (respectively)

D flip-flop (or DFF)

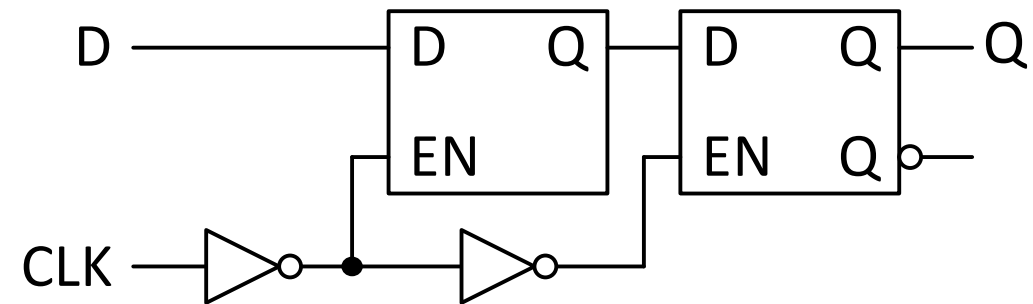
Negative-edge-triggered D flip-flop



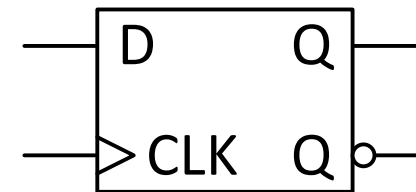
Functional symbol = logic symbol



Positive-edge-triggered D flip-flop



Functional symbol = logic symbol



D flip-flop (or DFF)

- Please note that the logic symbol of a flip-flop has an empty triangle on the signal used as clock!

Edge triggering signal


- This to indicate that that input is sensitive to the edge of the connected signal

 CLK → Positive-edge-triggered: active on the rising edge (of CLK)

 CLK → Negative-edge-triggered: active on the falling edge (of CLK)

Remember this!

- Not to the level (as in the case of latches)

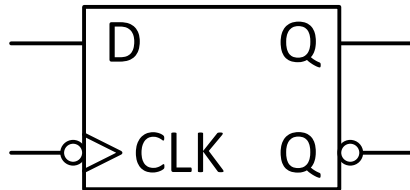
 EN → Positive-level-sensitive: active when EN = 1



 EN → Negative-level-sensitive: active when EN = 0

D flip-flop (or DFF)

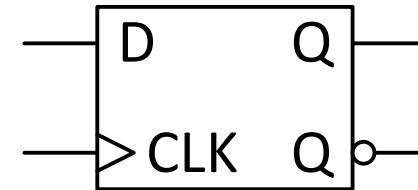
- For this reason, typically the (triggering) edge of the clock signal is indicated in the truth table of the DFF



Negative-edge-triggered D flip-flop



D	CLK	Q	\bar{Q}
0		0	1
1		1	0
X	0	Q^*	\bar{Q}^*
X	1	Q^*	\bar{Q}^*

Positive-edge-triggered D flip-flop

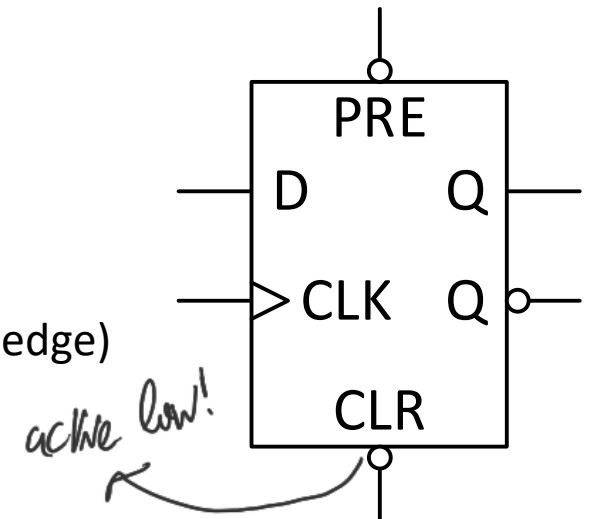
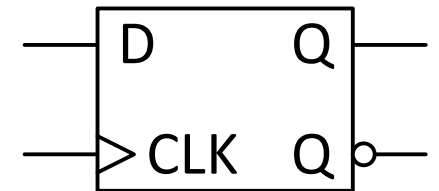


D	CLK	Q	\bar{Q}
0		0	1
1		1	0
X	0	Q^*	\bar{Q}^*
X	1	Q^*	\bar{Q}^*

logic level. We only care about the level

D flip-flop (or DFF)

- Typically, the positive-edge-triggered D flip-flop is the most used one
 - For this reason, it is generally indicated as just flip-flop
 - When one hears of a flip-flop, one implicitly refers to a D flip-flop
- Additional solutions exist (for both edge-triggered DFFs) that count
 - An asynchronous reset (or clear): CLR
 - To reset the DFF regardless of clock signal (an its activating edge)
 - An asynchronous preset: PRE
 - To preliminary set to 1 the DFF output regardless of clock signal (an its activating edge)
 - They are typically active-low signals



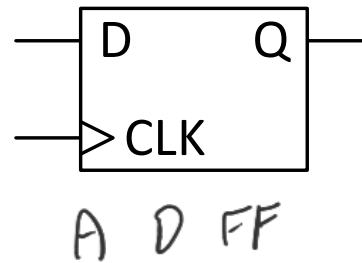
Register

- Finally, we can define what is a Register
 - A register is a group of D flip-flops that store a binary value and share the same clock signal
 - ↳ that one in parallel!
 - The number of DFFs depends on the bit width of the value to be stored
 - $N \text{ bits} \rightarrow N \text{ DFFs}$

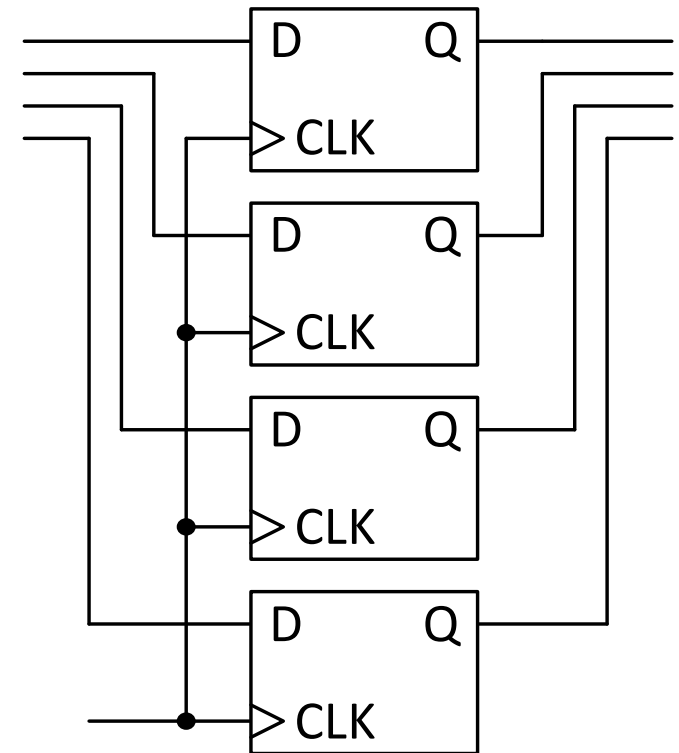
Register

- Some examples
 - Omitting \bar{Q}

1-bit register

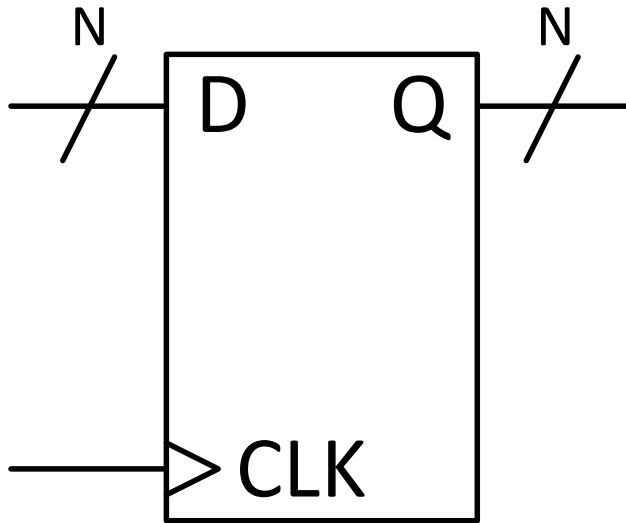


4-bit register



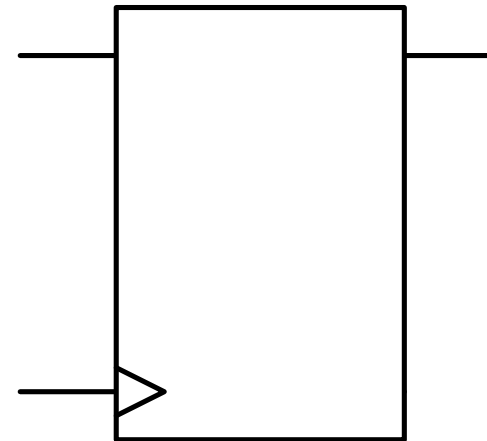
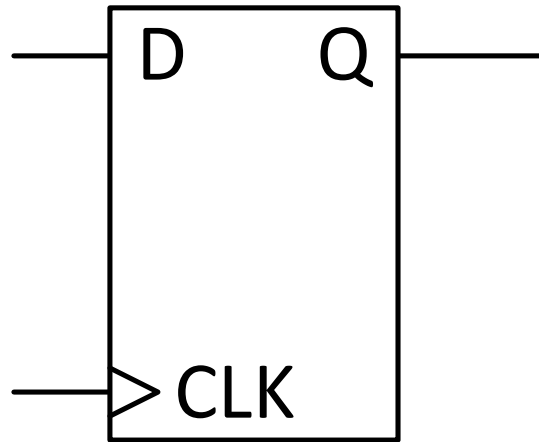
Register

- Functional symbol



Register

- Functional symbol
 - For simplicity, in RTL schematics you can find the following simplified symbols
 - I will do the same in the next lectures



Register


- Since the change of the register state occurs on a periodic signal (the clock), the register is a **synchronous** circuit
 - As well as DFF
- Like DFFs, registers can also have an asynchronous reset!
 - Which is shared between all the DFFs composing the register

Exercise with SystemVerilog

- Implementation of an 8-bit Register and simulation with Modelsim

Exercise with SystemVerilog

- Implementation of an 8-bit Register and simulation with Modelsim

```
module reg_8 (  
    input          clk  
  
    ,input          [7:0] d  
    ,output reg     [7:0] q  
);  
  
  
endmodule
```



Exercise with SystemVerilog

- Implementation of an 8-bit Register and simulation with Modelsim

```

module reg_8 (
    input          clk

    ,input          [7:0] d
    ,output reg     [7:0] q
);

    always_ff @ (posedge clk)
        q <= d;

endmodule
  
```

sensitivity list

- **Non-blocking assignment**

- **reg signal** Mandatory!
- **always_ff block** flip flop
 - Triggering edge of the clock signal in the sensitivity list
- operator <=

↓
if ns edge triggered

Exercise with SystemVerilog

- Implementation of an 8-bit Register and simulation with Modelsim

```

module reg_8 (
    input          clk


    ,input          [7:0] d
    ,output reg [7:0] q
);

    always_ff @ (posedge clk)
        q <= d;

endmodule
  
```

- Non-blocking assignment**

- reg signal
- always_ff block
 - Triggering edge of the clock signal in the sensitivity list
- operator <=



CLK	D	Q	\bar{Q}
0	-	Q^*	\bar{Q}^*
1	-	Q^*	\bar{Q}^*
0	0	0	1
1	1	1	0

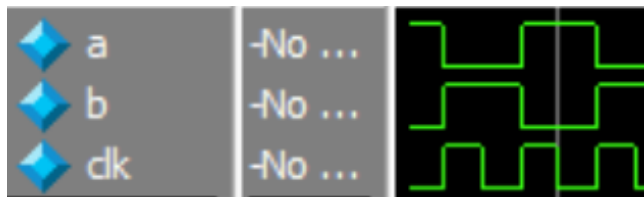
Exercise with SystemVerilog

- Non-blocking vs blocking assignment

- **Non-blocking assignment**

- Evaluated in sequence
- But assigned only after all are evaluated

```
always_ff @ (posedge clk) begin
    a <= b;
    b <= a;
end
```

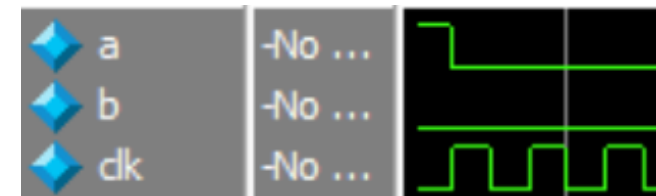


When posedge, the values are evaluated but not assigned. At the end of the block the signals are assigned

- **Blocking assignment**

- Assigned 'immediately'

```
always_ff @ (posedge clk) begin
    a = b;
    b = a;
end
```



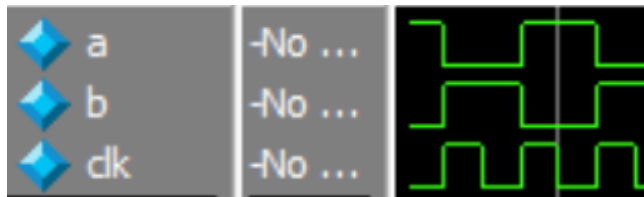
Exercise with SystemVerilog

- Non-blocking vs blocking assignment

- **Non-blocking assignment**

- Evaluated in sequence
- But assigned only after all are evaluated

```
always_ff @ (posedge clk) begin
    a <= b;
    b <= a;
end
```

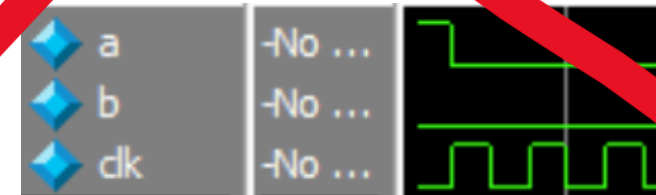


Pay attention!

- **Blocking assignment**

- Assigned 'immediately'

```
always_ff @ (posedge clk) begin
    a = b;
    b = a;
end
```



Exercise with SystemVerilog

- Non-blocking vs blocking assignment

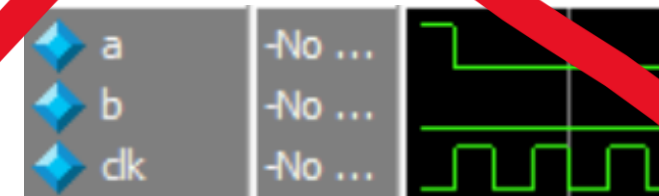
Pay attention!

- **Blocking assignment**

- Assigned 'immediately'

Do not use blocking assignment (=) inside always_ff blocks to describe sequential circuits!

```
always_ff @(posedge clk) begin
    a = b;
    b = a;
end
```



Exercise with SystemVerilog

- Implementation of an 8-bit Register and simulation with Modelsim

```
module reg_8 (  
    input          clk  
    ,input          rst_n  
    ,input          [7:0] d  
    ,output reg [7:0] q  
);  
  
always_ff @ (posedge clk or negedge rst_n)  
    if(!rst_n)  
        q <= '0; → whenever the rst, fill with 0  
    else  
        q <= d;  
  
endmodule
```

- Adding asynchronous reset

- port
- signal (edge) inside sensitivity list
 - asynchronous w.r.t. to clock signal
 - must activate the always_ff block regardless of the clock
 - specify required edge
 - negedge → active-low
 - posedge → active-high

Exercise with SystemVerilog

- Implementation of an 8-bit Register and simulation with Modelsim

```

module reg_8 (
    input          clk
    ,input          rst_n
    ,input          [7:0] d
    ,output reg [7:0] q
);

always_ff @ (posedge clk or negedge rst_n)
if (!rst_n) if(rst_n)
    q <= '0;
else
    q <= d;

endmodule
  
```

Handwritten annotations:

- posedge** (written above the sensitivity list)
- if(rst_n)** (written in red, replacing ~~if (!rst_n)~~)
- if active-high** (written in a yellow box with a red border, with an arrow pointing to the **if(rst_n)** condition)

- Adding **asynchronous reset**

- port
- signal (edge) inside sensitivity list
 - asynchronous w.r.t. to clock signal
 - must activate the always_ff block regardless of the clock
 - specify required edge
 - negedge → active-low
 - posedge → active-high

Exercise with SystemVerilog

- Implementation of an 8-bit Register and simulation with Modelsim

```
module reg_8 (  
    input          clk  
    ,input          rst_n  
    ,input          [7:0] d  
    ,output reg [7:0] q  
);  
  
    always_ff @ (posedge clk)  
        if(!rst_n)  
            q <= '0;  
        else  
            q <= d;  
  
endmodule
```

- Adding **synchronous reset**
 - port
 - but nothing more
 - no signal (edge) in the sensitivity list
 - also reset is synchronous to clock, so reset must not activate the always_ff block

Force rst_n: porta di sistema a
uno stato di partenza.

Exercise with SystemVerilog

Register without reset

```

module reg_8 (
    input          clk
    ,input         [7:0] d
    ,output reg    [7:0] q
);

    always_ff @ (posedge clk)
        q <= d;

endmodule

```

Register with asynchronous (active-low) reset

```

module reg_8 (
    input          clk
    ,input         rst_n
    ,input         [7:0] d
    ,output reg    [7:0] q
);

    always_ff @ (posedge clk or negedge rst_n)
        if(!rst_n)
            q <= '0;
        else
            q <= d;

endmodule

```

Register with asynchronous (active-high) reset

```

module reg_8 (
    input          clk
    ,input         rst
    ,input         [7:0] d
    ,output reg    [7:0] q
);

    always_ff @ (posedge clk or posedge rst)
        if(rst)
            q <= '0;
        else
            q <= d;

endmodule

```

Register with synchronous (active-low) reset

```

module reg_8 (
    input          clk
    ,input         rst_n
    ,input         [7:0] d
    ,output reg    [7:0] q
);

    always_ff @ (posedge clk)
        if(!rst_n)
            q <= '0;
        else
            q <= d;

endmodule

```

Register with synchronous (active-high) reset

Exercise with SystemVerilog

- Implementation of an 8-bit Register and simulation with Modelsim
 - You can find all the files about this exercise in the dedicated folder on the Team of the course
 - File > Electronics Systems module > Crocetti > Exercises > 2.3
 - Try to simulate on your own
 - However, I also included two .do files to automate the waveform and the simulation
 - wave.do and sim.do
 - Refer to README.txt file



Thank you for your attention

Luca Crocetti
(luca.crocetti@unipi.it)