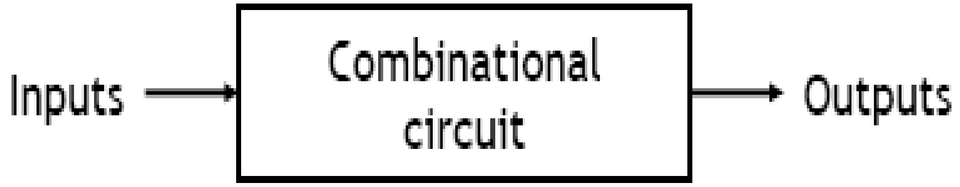
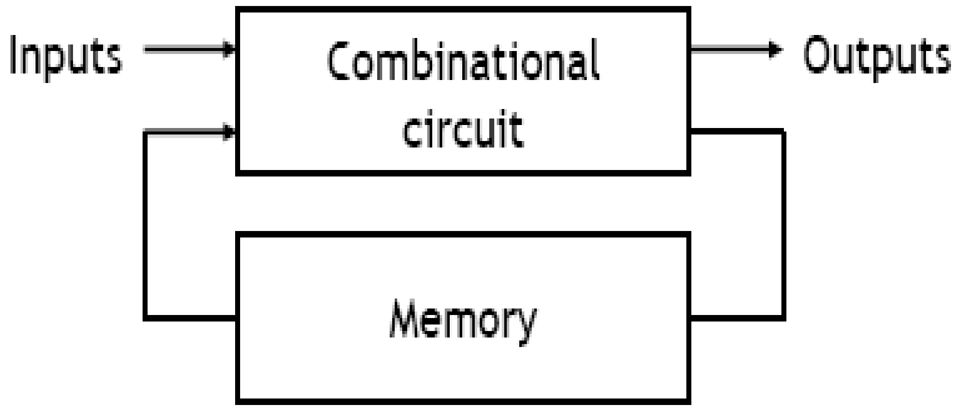
Combinational circuits



- So far we've only worked with combinational circuits, where applying the same inputs always produces the same outputs.

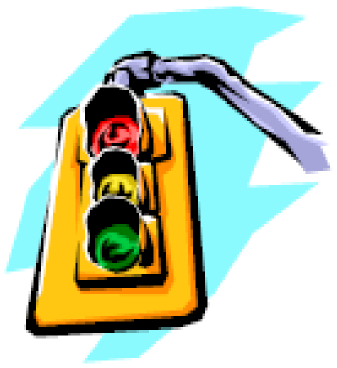
Sequential circuits



- In contrast, the outputs of a sequential circuit depend on not only the inputs, but also the state, or the current contents of some memory.
- This makes things more difficult to understand since the same inputs can yield *different* outputs, depending on what's stored in memory.
- The memory contents can also change as the circuit runs, so the *order* in which things occur makes a difference.

Examples of sequential devices

- Many real-life devices are sequential in nature.
  - Combination locks open if you enter numbers in the right order.
  - Elevators move up or down and open or close in response to buttons that are pressed on different floors and in the elevator itself.
  - Traffic lights change from red to green depending on whether a car is waiting at the intersection.
- More importantly for us, computers are sequential! For instance, key presses and mouse clicks have different effects based on which program is loaded into memory, and the state of that program.



What exactly is memory?

- A memory should support at least three operations.

  1. It should be able to hold a value.
  2. You should be able to *read* the value that is saved.
  3. You should be able to *change* that value.

- We'll start with the simplest case, a one-bit memory.

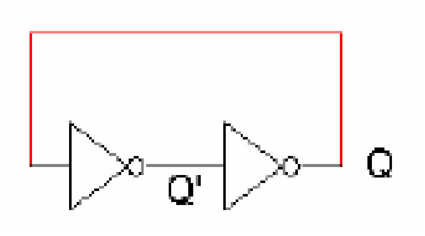
  1. It should be able to hold a single bit, 0 or 1.
  2. You should be able to read the bit that is saved.
  3. You should be able to change the bit.
     - You can set the bit to 1
     - You can reset or clear the bit to 0.

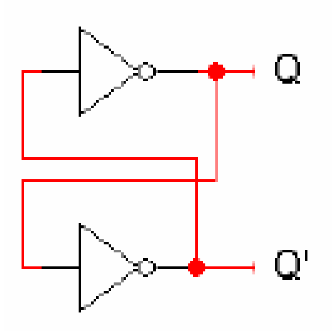
The basic idea of storage

- How can a circuit remember anything, when it’s just a bunch of gates that produce outputs according to the inputs?
- The idea is to make a loop in a circuit, so the outputs are *also* inputs.

The basic idea of storage

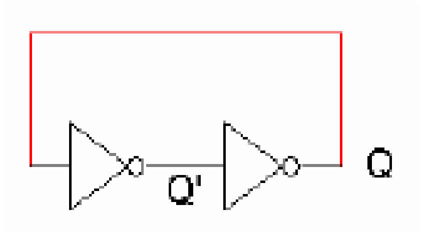
- How can a circuit remember anything, when it's just a bunch of gates that produce outputs according to the inputs?
- The idea is to make a loop in a circuit, so the outputs are *also* inputs.
- Here is one initial attempt, shown with two equivalent layouts.

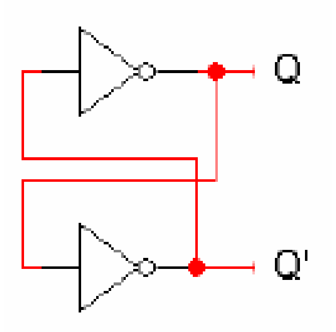




The basic idea of storage

- How can a circuit remember anything, when it's just a bunch of gates that produce outputs according to the inputs?
- The idea is to make a loop in a circuit, so the outputs are *also* inputs.
- Here is one initial attempt, shown with two equivalent layouts.

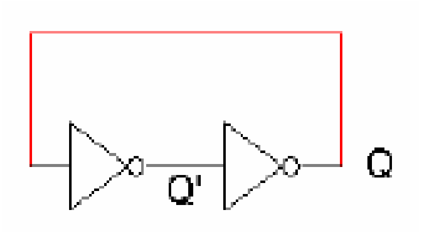


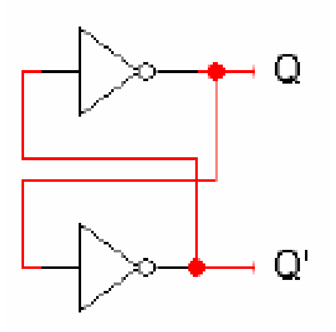


- Does this satisfy the properties of memory?
  - These circuits "remember" Q since its value never changes. Similarly, Q' never changes either.

The basic idea of storage

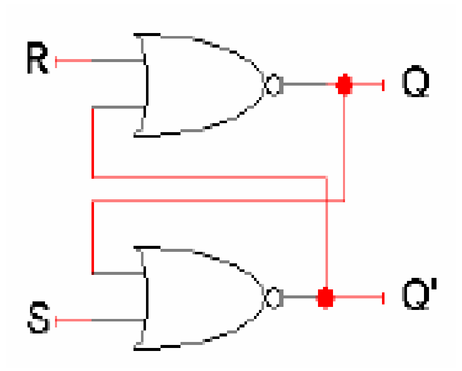
- How can a circuit remember anything, when it's just a bunch of gates that produce outputs according to the inputs?
- The idea is to make a loop in a circuit, so the outputs are *also* inputs.
- Here is one initial attempt, shown with two equivalent layouts.





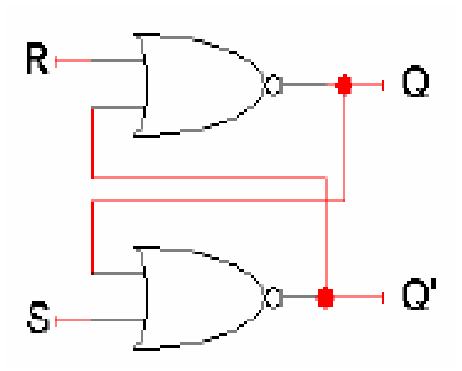
- Does this satisfy the properties of memory?
  - These circuits "remember" Q since its value never changes. Similarly, Q' never changes either.
  - We can "read" Q by sending it to another gate or device, as usual.
  - But we can't *change* Q! There are no external inputs here, so we can't control whether Q=1 or Q=0.

- Let's use NOR gates instead of inverters. The SR latch here has two inputs S and R, which will let us control the outputs Q and Q'.



- Q and Q' feed back into the circuit, so they're not only outputs, they're also inputs!

- Let's use NOR gates instead of inverters. The SR latch here has two inputs S and R, which will let us control the outputs Q and Q'.



- Q and Q' feed back into the circuit, so they're not only outputs, they're also inputs!
- To figure out how Q and Q' change, we must look at not only the inputs S and R, but also the *current* values of Q and Q'.

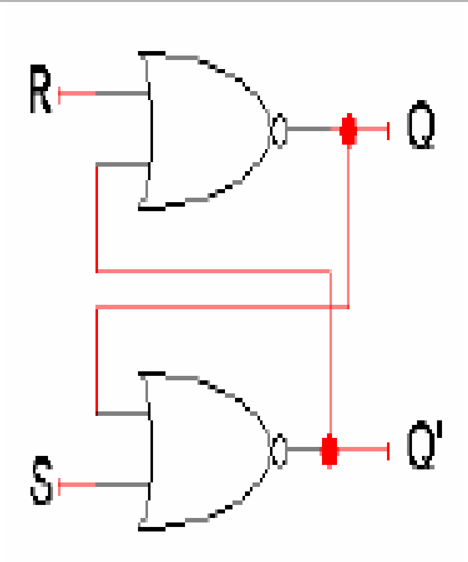
$$Q_{next} = (R + Q'_{current})'$$
$$Q'_{next} = (S + Q_{current})'$$

- Let's see how different input values for S and R affect this thing.

Storing a value: SR = 00

- What if S = 0 and R = 0?
- The equations on the right reduce to:

$$Q_{next} = (0 + Q'_{current})' = Q_{current}$$
$$Q'_{next} = (0 + Q_{current})' = Q'_{current}$$



$$Q_{next} = (R + Q'_{current})'$$
$$Q'_{next} = (S + Q_{current})'$$

- So when SR = 00, then $Q_{next} = Q_{current}$.
- This is exactly what we need to store values in the latch.

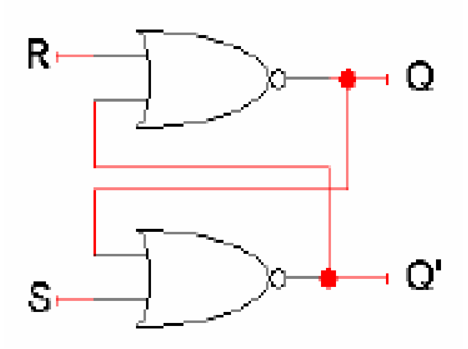
Setting the latch: SR = 10

- What if S = 1 and R = 0?
- Since S = 1, Q'_{next} is 0, *regardless* of $Q_{current}$.

$$Q'_{next} = (1 + Q_{current})' = 0$$

- Then this new value of Q' goes into the top NOR gate, along with R = 0.

$$Q_{next} = (0 + 0)' = 1$$



$$Q_{next} = (R + Q'_{current})'$$
$$Q'_{next} = (S + Q_{current})'$$

Setting the latch: SR = 10

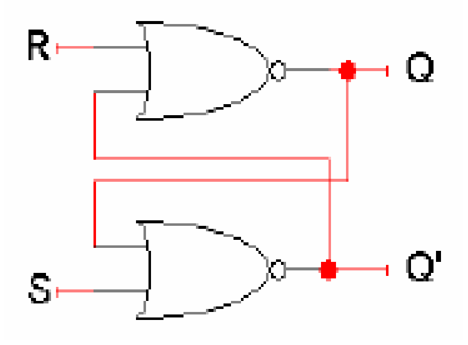
- What if S = 1 and R = 0?
- Since S = 1, Q'_{next} is 0, *regardless* of $Q_{current}$.

$$Q'_{next} = (1 + Q_{current})' = 0$$

- Then this new value of Q' goes into the top NOR gate, along with R = 0.

$$Q_{next} = (0 + 0)' = 1$$

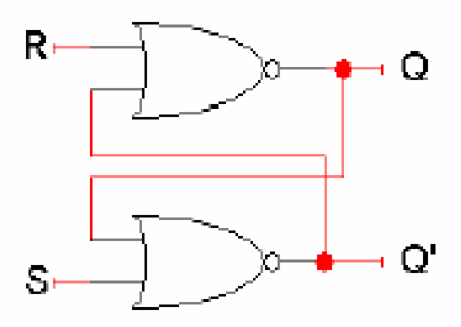
- So when SR = 10, then $Q'_{next} = 0$ and $Q_{next} = 1$. This is how you set the latch to 1; the S input stands for "set."
- Notice it can take up to two steps (two gate delays) from the time S becomes 1 to the time Q_{next} becomes 1.
- But once Q_{next} becomes 1, the outputs will stop changing. This is a stable state.



$$Q_{next} = (R + Q'_{current})'$$
$$Q'_{next} = (S + Q_{current})'$$

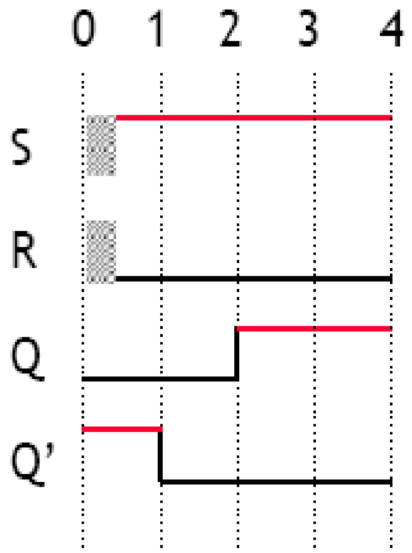
Latch delays

- Timing diagrams are especially useful for seeing how sequential circuits work.
- Here is a diagram which shows an example of how our latch outputs change with inputs SR = 10.



0. Let's say that Q = 0 and Q' = 1 initially.
1. Since S = 1, Q' changes from 1 to 0 after one NOR-gate delay (marked with vertical lines in the timing diagram).
2. This change in Q', along with R = 0, causes Q to become 1 after another gate delay.
3. The latch then stabilizes until S or R change again.

$$Q_{next} = (R + Q'_{current})'$$
$$Q'_{next} = (S + Q_{current})'$$



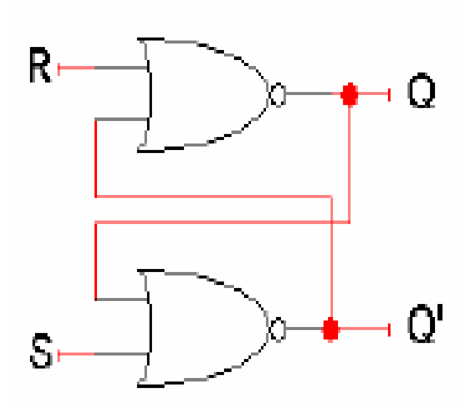
Resetting the latch: SR = 01

- What if S = 0 and R = 1?
- Since R = 1, Q_{next} is 0, *regardless* of $Q_{current}$.

$$Q_{next} = (1 + Q'_{current})' = 0$$

- Then this new value of Q goes into the bottom NOR gate, where S = 0.

$$Q'_{next} = (0 + 0)' = 1$$



$$Q_{next} = (R + Q'_{current})'$$
$$Q'_{next} = (S + Q_{current})'$$

Resetting the latch: SR = 01

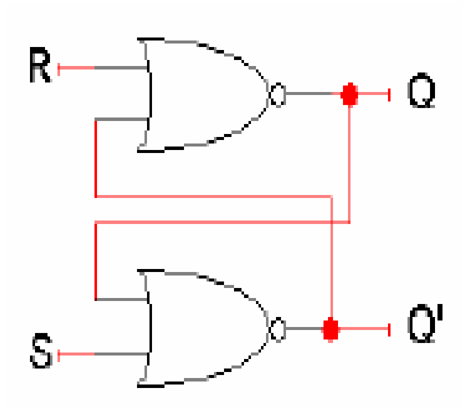
- What if S = 0 and R = 1?
- Since R = 1, Q_{next} is 0, *regardless* of $Q_{current}$.

$$Q_{next} = (1 + Q'_{current})' = 0$$

- Then this new value of Q goes into the bottom NOR gate, where S = 0.

$$Q'_{next} = (0 + 0)' = 1$$

- So when SR = 01, then $Q_{next} = 0$ and $Q'_{next} = 1$. This is how you reset, or clear, the latch to 0; the R input stands for "reset."
- Again, it can take two gate delays before a change in R propagates to the output Q'_{next}.



$$Q_{next} = (R + Q'_{current})'$$
$$Q'_{next} = (S + Q_{current})'$$

SR latches are memories!

- This characteristic table shows that our latch provides everything we need in a memory: we can set it, reset it, or keep the current value.

S	R	Q
0	0	No change
0	1	0 (reset)
1	0	1 (set)

- The output Q represents the data stored in the latch. It is also called the state of the latch.

- We can expand the table above into a state table, which explicitly shows that the *next* values of Q and Q' depend on their *current* values, as well as on the inputs S and R.

Inputs		Current		Next	
S	R	Q	Q'	Q	Q'
0	0	0	1	0	1
0	0	1	0	1	0
0	1	0	1	0	1
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	0	1	0

SR latches are sequential!

- Note that for SR = 00, the next value of Q could be *either* 0 or 1, depending on the current value of Q.
- So the same inputs can produce different outputs, depending on whether the latch is currently set or reset.
- This is different from the combinational circuits that we've seen so far, where the same inputs always generate the same outputs.

S	R	Q
0	0	No change
0	1	0 (reset)
1	0	1 (set)

Inputs		Current		Next	
S	R	Q	Q'	Q	Q'
0	0	0	1	0	1
0	0	1	0	1	0
0	1	0	1	0	1
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	0	1	0

What about SR = 11?

- *Both* Q_{next} and Q'_{next} would become 0, which contradicts the assumption that Q and Q' are always complements.

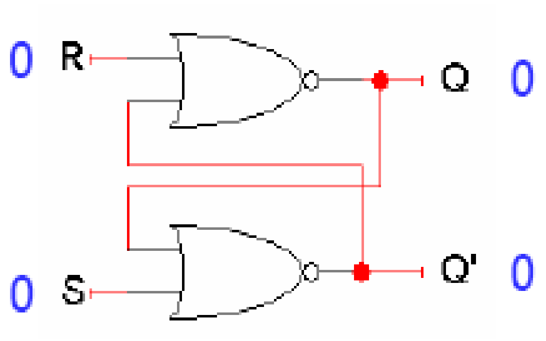
$$Q_{next} = (R + Q'_{current})'$$
$$Q'_{next} = (S + Q_{current})'$$

What about SR = 11?

- *Both* Q_{next} and Q'_{next} would become 0, which contradicts the assumption that Q and Q' are always complements.
- Another problem is what happens if we then make S = 0 and R = 0 together.

$$Q_{next} = (0 + 0)' = 1$$
$$Q'_{next} = (0 + 0)' = 1$$

$$Q_{next} = (R + Q'_{current})'$$
$$Q'_{next} = (S + Q_{current})'$$



What about SR = 11?

$$Q_{next} = (R + Q'_{current})'$$
$$Q'_{next} = (S + Q_{current})'$$

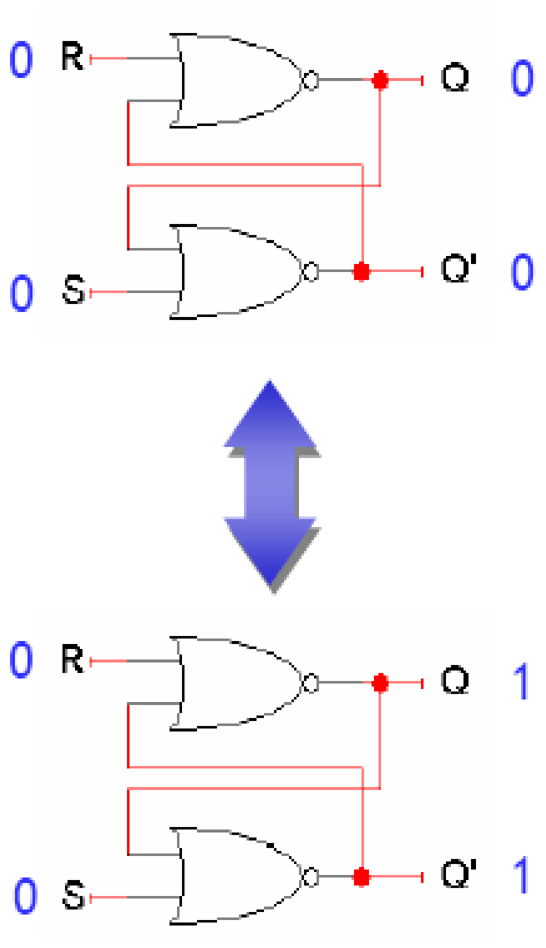
- *Both* Q_{next} and Q'_{next} would become 0, which contradicts the assumption that Q and Q' are always complements.
- Another problem is what happens if we then make S = 0 and R = 0 together.

$$Q_{next} = (0 + 0)' = 1$$
$$Q'_{next} = (0 + 0)' = 1$$

- But these new values go back into the NOR gates, and we then get Q = Q' = 0 again.

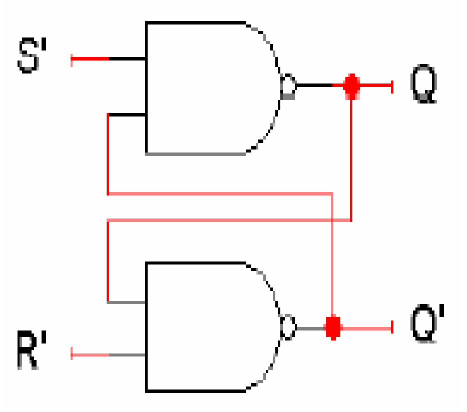
$$Q_{next} = (0 + 1)' = 0$$
$$Q'_{next} = (0 + 1)' = 0$$

- So the circuit enters an infinite loop, where Q and Q' cycle between 0 and 1 forever.
- Don't set SR = 11!



S’R’ latch

- There are several other variations of the basic latch.
- You can use NAND instead of NOR gates to get a S’R’ latch.

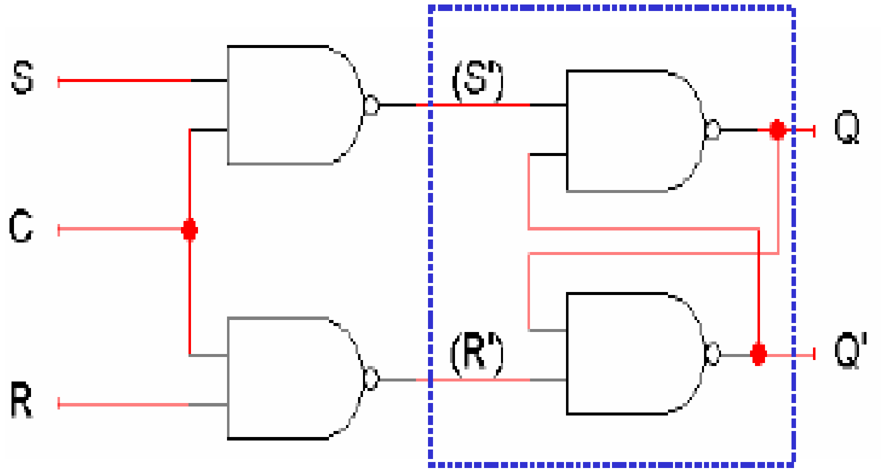


S’	R’	Q
1	1	No change
1	0	0 (reset)
0	1	1 (set)
0	0	Avoid!

- This is just like an SR latch but with inverted inputs, as you can see from the table.

An SR latch with a control input

- Here is an SR latch with a control input C, which acts like an enable.

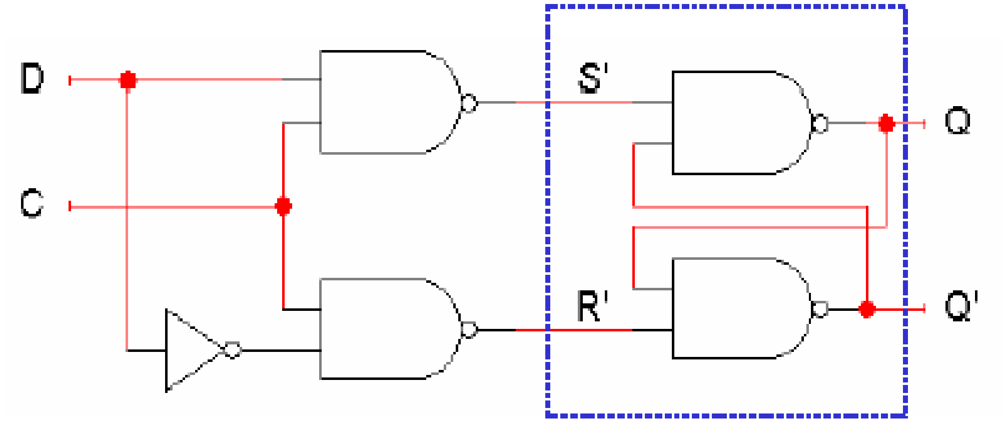


C	S	R	S’	R’	Q
0	x	x	1	1	No change
1	0	0	1	1	No change
1	0	1	1	0	0 (reset)
1	1	0	0	1	1 (set)
1	1	1	0	0	Evil!

- Notice the hierarchical design!
  - The dotted blue box contains the S’R’ latch from the previous slide.
  - The additional NAND gates are simply used to generate appropriate inputs for the S’R’ latch.

D latch

- A D latch is also based on an S'R' latch. The additional gates generate the S' and R' signals, based on inputs D ("data") and C ("control").
  - When C = 0, S' and R' are both 1, so Q does not change.
  - When C = 1, the latch output Q will equal the input D.



C	D	Q
0	x	No change
1	0	0
1	1	1

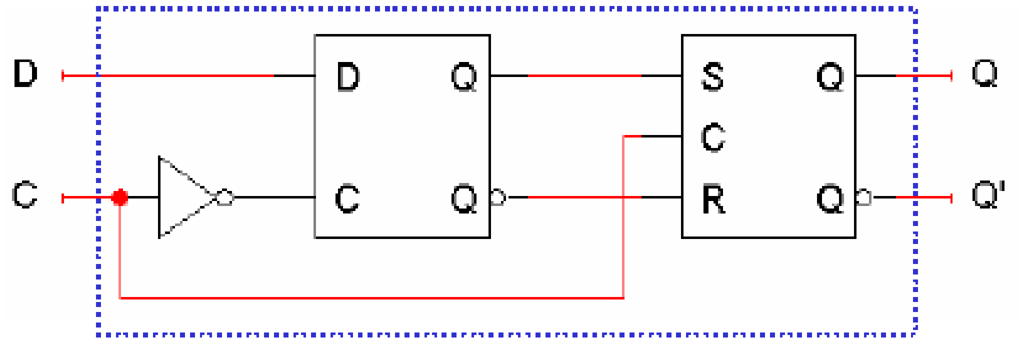
- There are two main advantages of a D latch.
  - No more messing with one input for set and another input for reset!
  - This latch has no "bad" input combinations to avoid. Any of the four possible assignments to C and D are valid.

Flip-Flops

- Latches are “transparent” (=> any change on the inputs is seen at the outputs immediately).
- This causes synchronization problems!
- Solution: use latches to create flip-flops that can respond (update) ONLY on SPECIFIC times (instead of ANY time).

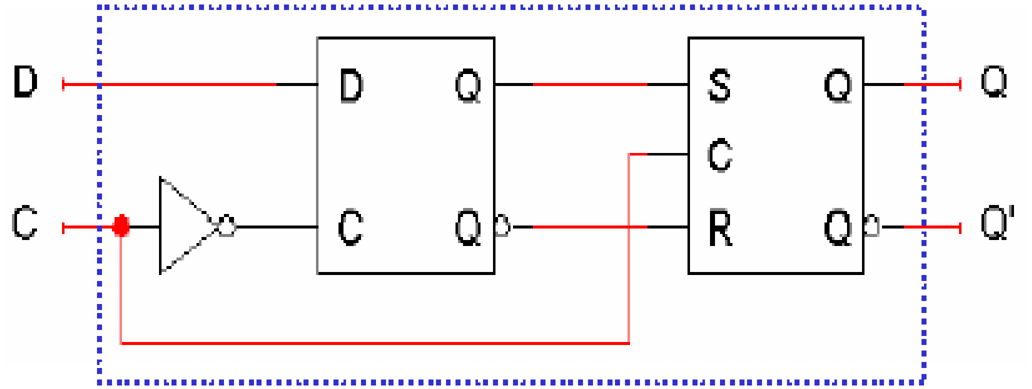
D FLIP-FLOP

- Here is the internal structure of a D-type flip-flop.
  - The flip-flop inputs are C and D, and the outputs are Q and Q’.
  - The D latch on the left is the master, while the SR latch on the right is called the slave.



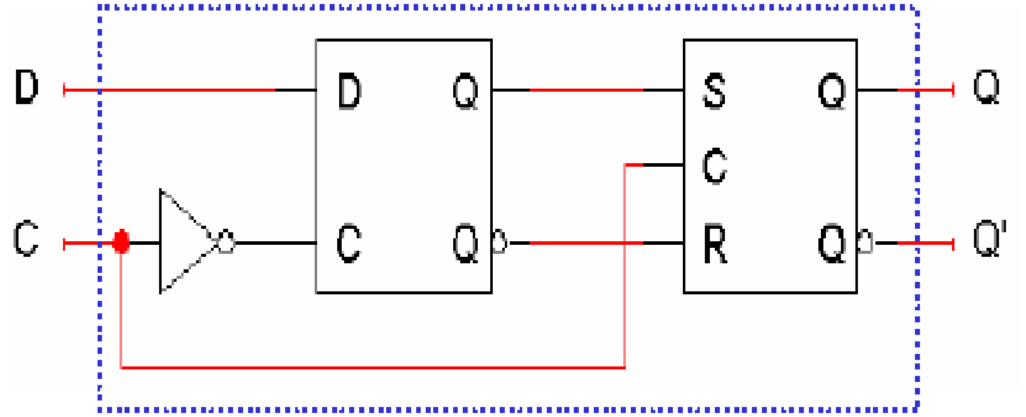
- Note the layout here.
  - The flip-flop input D is connected directly to the master latch.
  - The master latch output goes to the slave.
  - The flip-flop outputs come directly from the slave latch.

D flip-flops when C=0



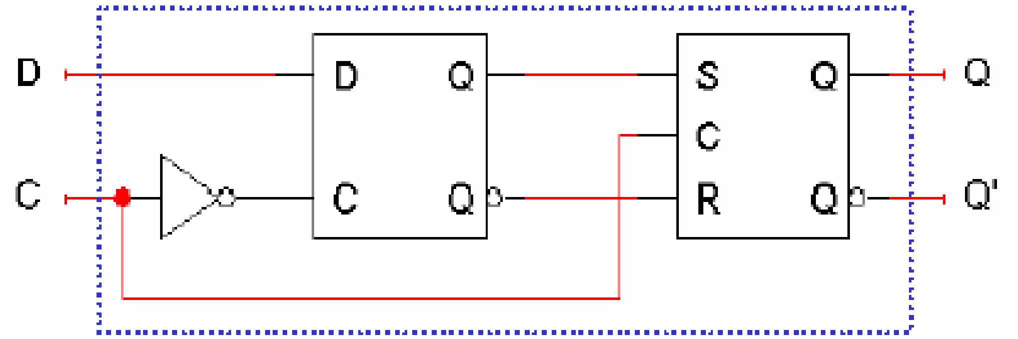
- The D flip-flop's control input C enables either the master D latch or the slave SR latch, but *not* both.
- What happens when C = 0?
  - The master latch is enabled, and it tracks the flip-flop input D. When D changes, the master's output changes too.
  - The slave is disabled, so the D latch output has no effect on it. Thus, the slave just maintains the flip-flop's current state.

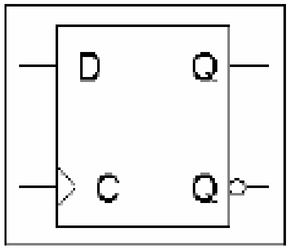
D flip-flops when C=1



- Several things happen *as soon as* C changes from 0 to 1.
  - The master is disabled. Its output will be the *last* D input value seen, just before C became 1. Any subsequent changes to the D input will have no effect on the master latch while C = 1.
  - On the other hand, the slave is enabled. Its output changes to reflect the master's state, which again is the D input value from right when C became 1.

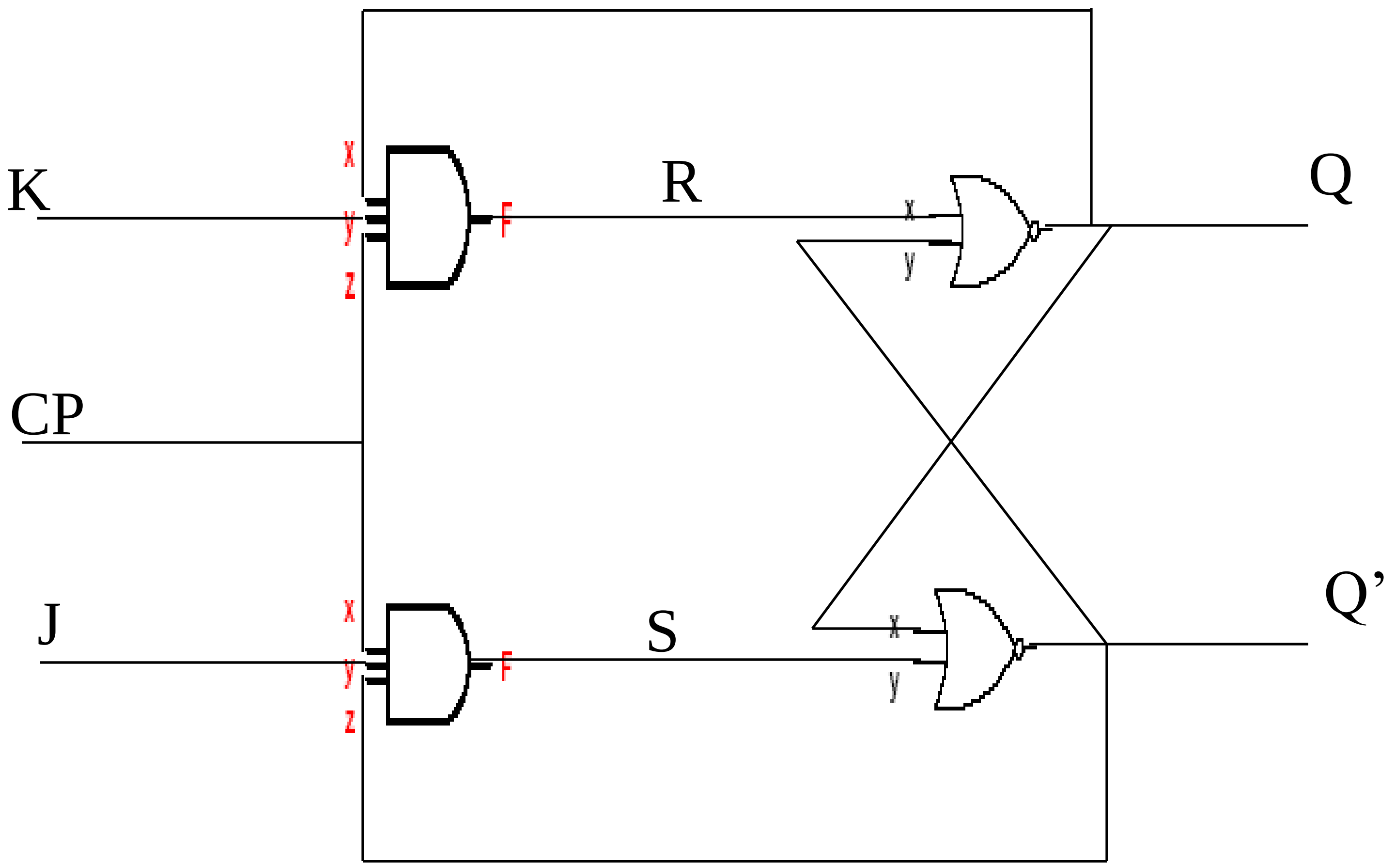
Positive edge triggering



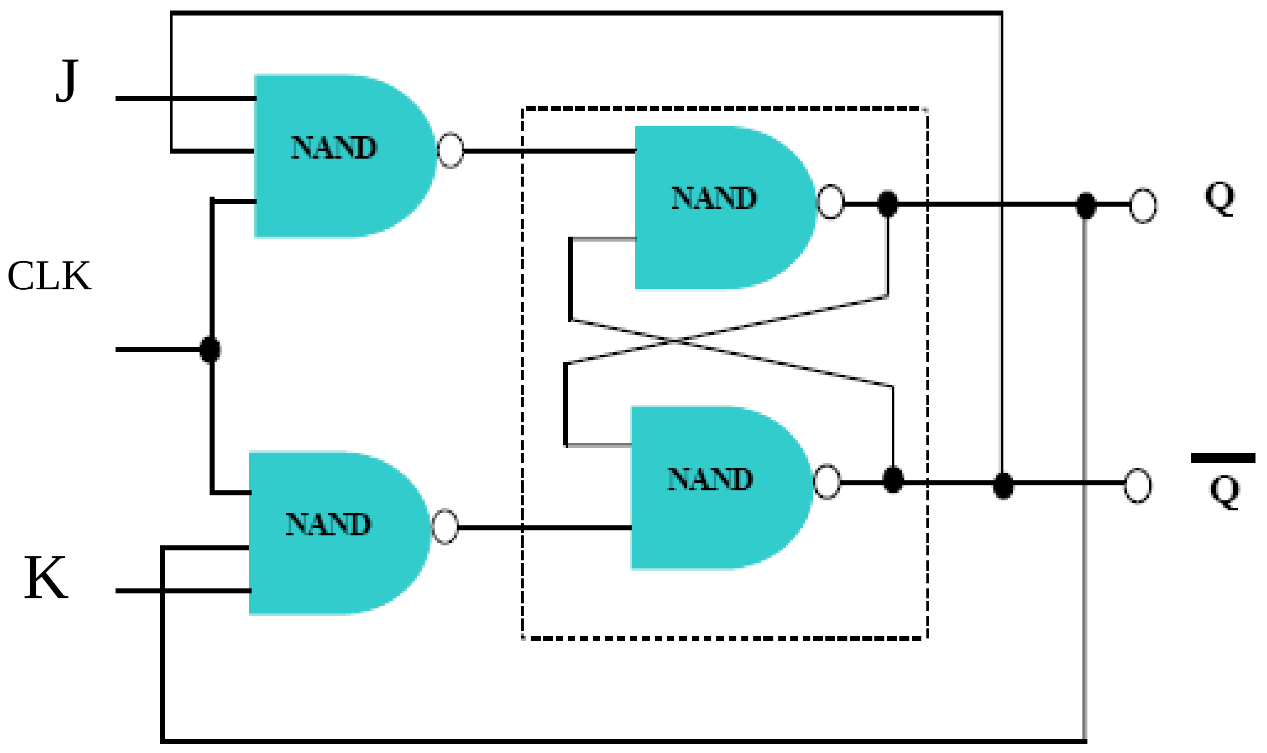


- This is called a positive edge-triggered flip-flop.
  - The flip-flop output Q changes *only* after the positive edge of C.
  - The change is based on the flip-flop input values that were present right at the positive edge of the clock signal.
- The D flip-flop's behavior is similar to that of a D latch, except for the positive edge-triggered nature, which is not explicit in this table.

C	D	Q
0	x	No change
1	0	0 (reset)
1	1	1 (set)

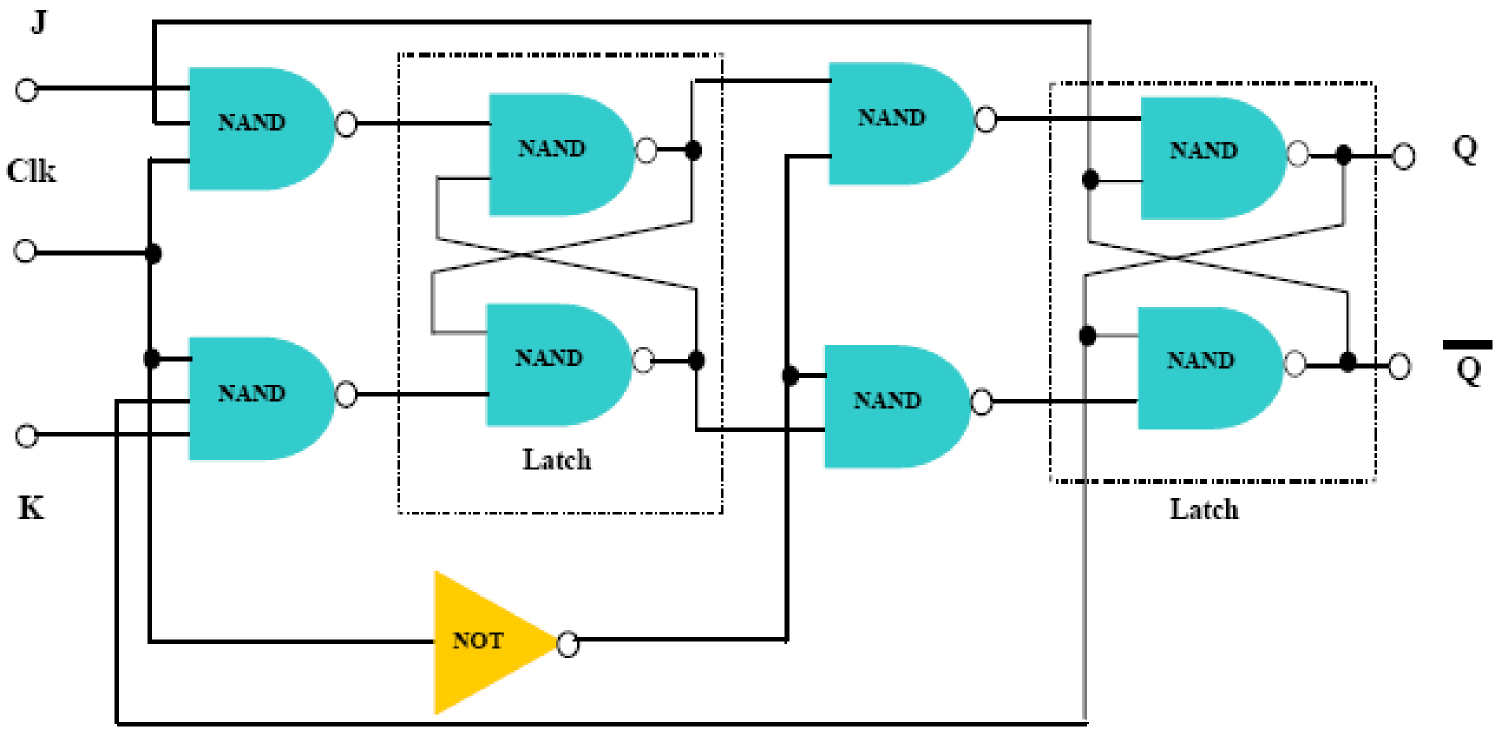


JK



JK using NAND gates

MASTER-SLAVE CONFIGURATION


J
Clk
K
NAND
NAND
NAND
NAND
Latch
NAND
NAND
NOT
NAND
NAND
Latch
Q
$\overline{Q}$

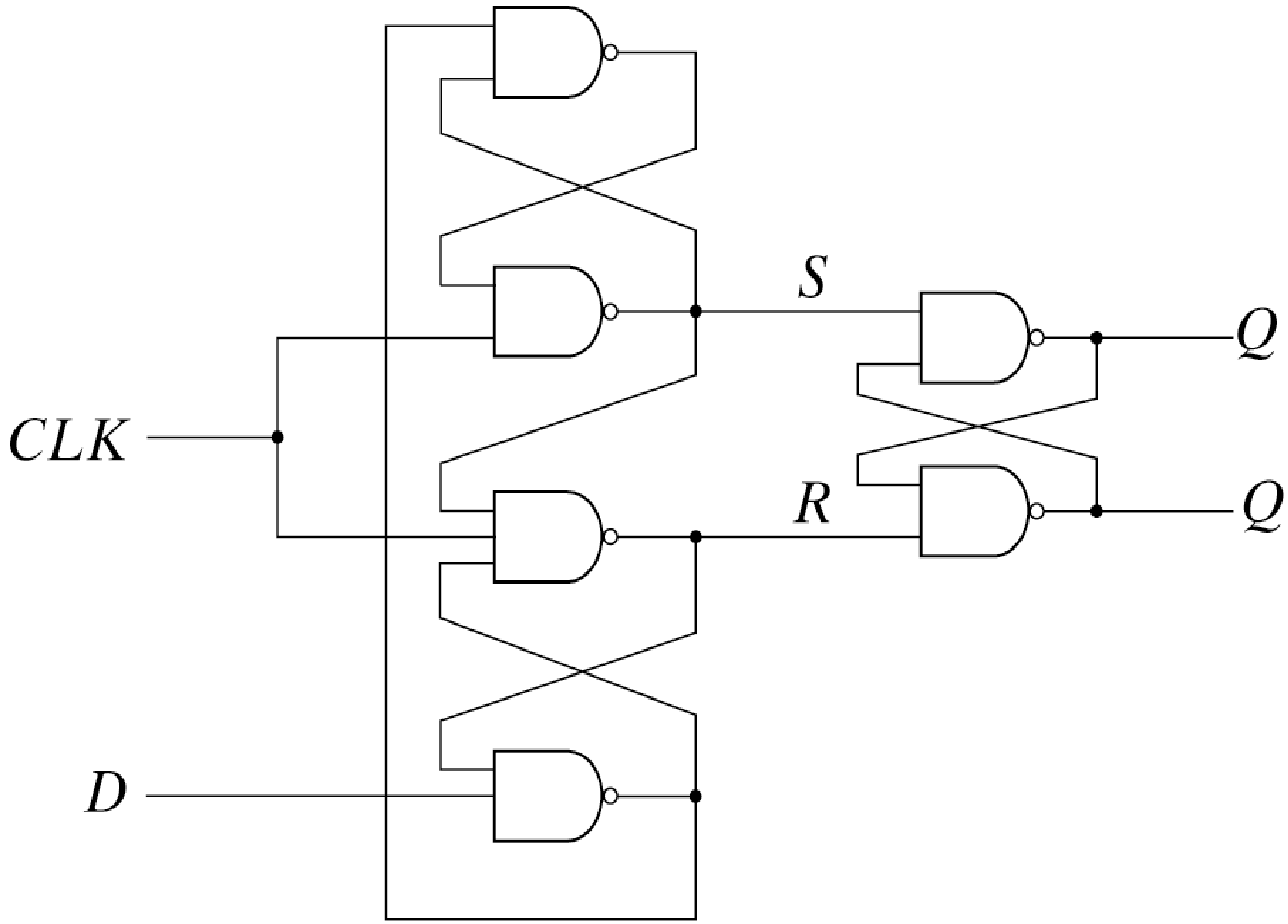


Fig. 5-10 *D*-Type Positive-Edge-Triggered Flip-Flop

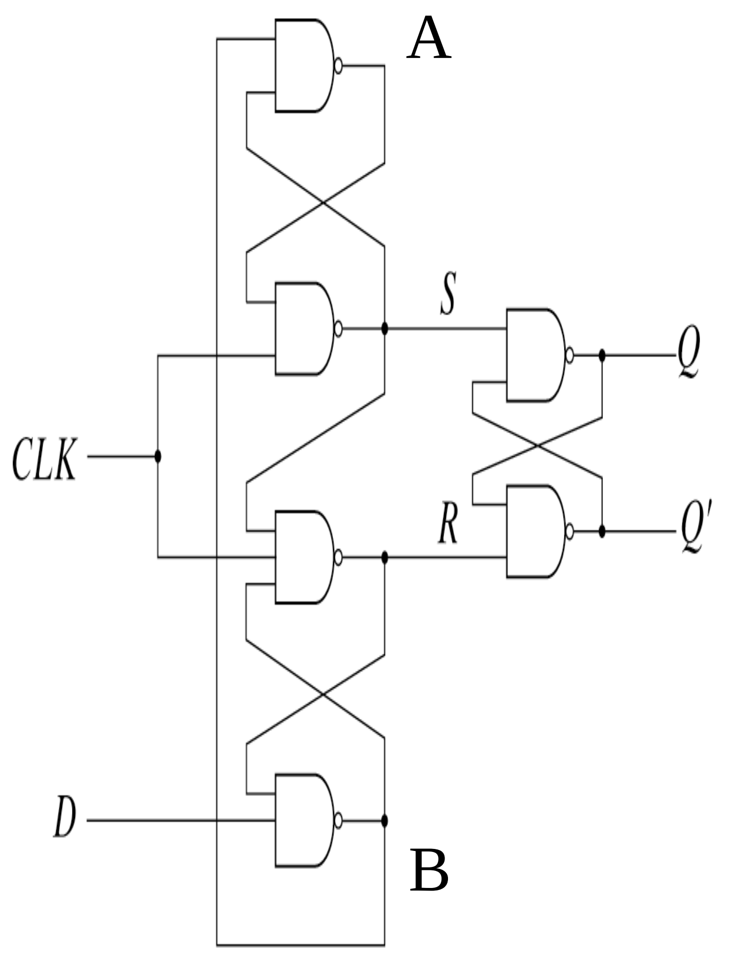


Fig. 5-10 *D*-Type Positive-Edge-Triggered Flip-Flop

Case I:

- **D = 0, CLK = 0**
- **D = 0 => B = 1**
- **CLK = 0 => R =1, S = 1**
- **B = 1, S = 1 => A = 0**
- **ASRB = 0111**
- **SR = 11 => No change**

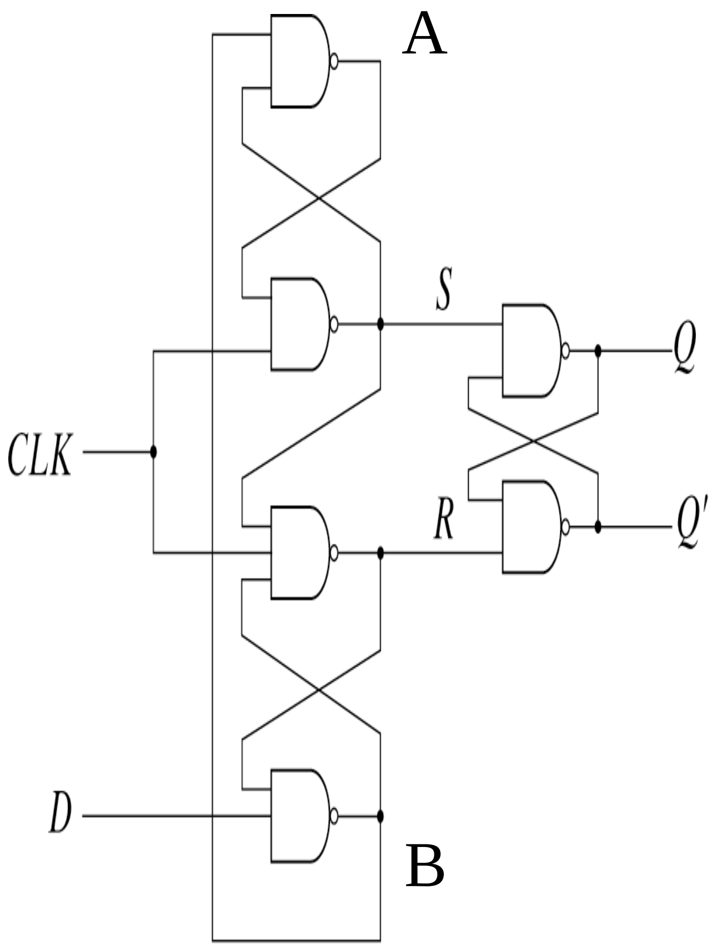


Fig. 5-10 *D*-Type Positive-Edge-Triggered Flip-Flop

In case I make CLK=1

i.e., ASRB=0111

- **D=0 => B=1**
- **B=1, S=1 => A=0**
- **A=0 => S=1**
- **S=1,B=1,CLK=1 => R=0**
- **SR=10 => RESET**
- **ASRB=0101**

Only R has changed here after one gate delay. If D changes after R has stabilized there will be no effect on output.

This is Hold Time.

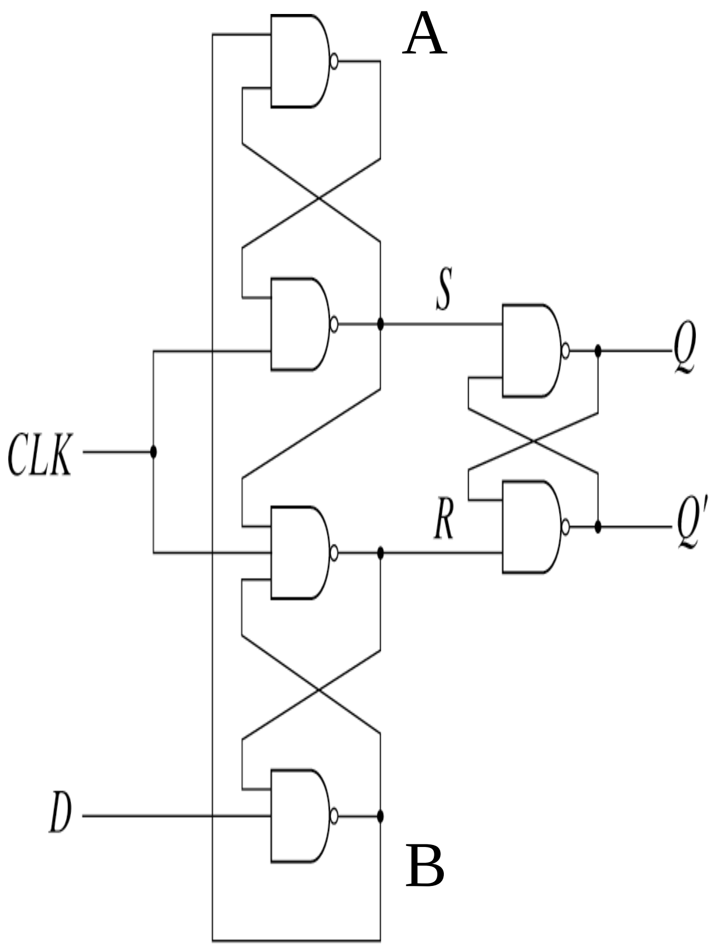


Fig. 5-10 *D*-Type Positive-Edge-Triggered Flip-Flop

Case II:

- **D = 1, CLK = 0**
- **CLK = 0 => R =1, S = 1**
- **D = 1, R = 1 => B = 0**
- **B = 0 => A = 1**
- **ASRB = 1110**
- **SR = 11 => No change**

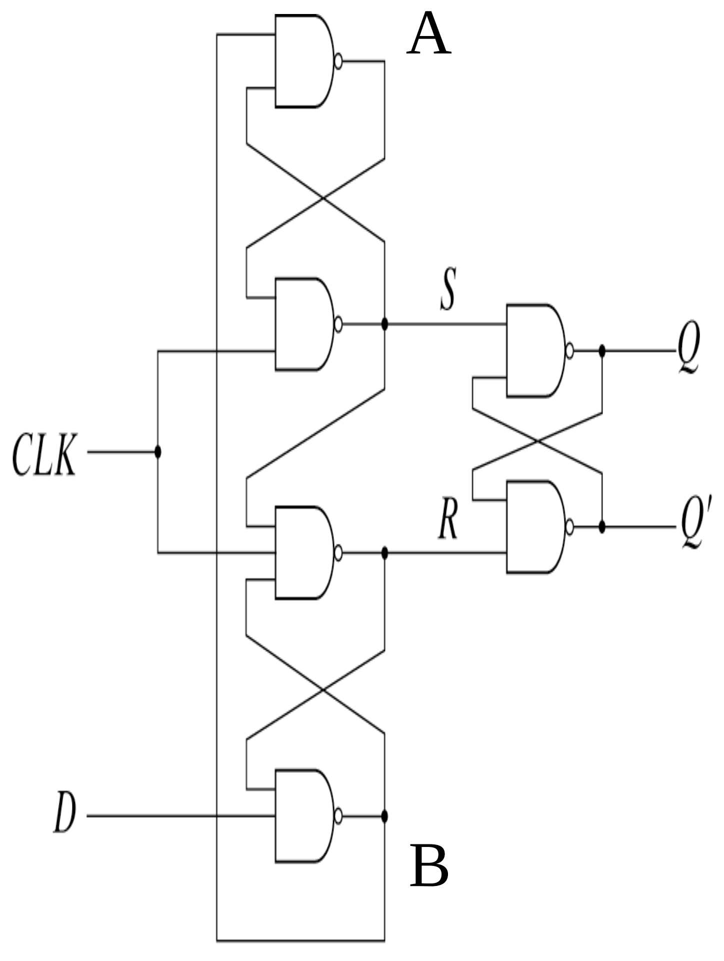


Fig. 5-10 *D*-Type Positive-Edge-Triggered Flip-Flop

In case II make CLK = 1

- **ASRB = 1110**
- **A=1,CLK=1 => S=0 =>R=1**
- **R=1, D=1 => B=0**
- **B=0, => A=1**
- **SR=01 => SET**
- **ASRB=1010**

Only S has changed here after one gate delay. If D changes after S has stabilized there will be no effect on output.

This is once again Hold Time.

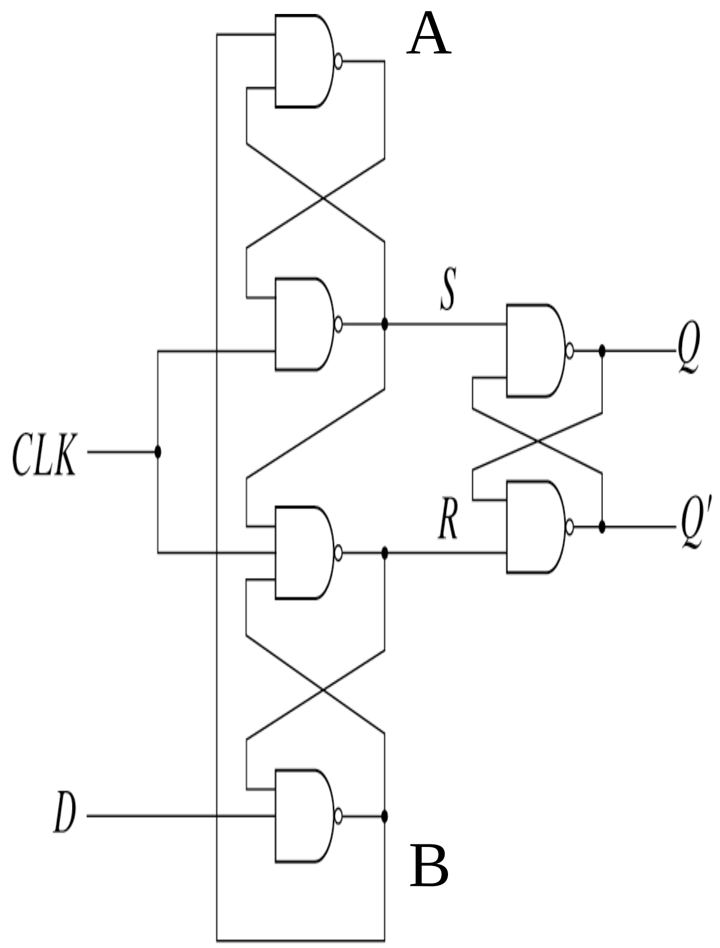


Fig. 5-10 *D*-Type Positive-Edge-Triggered Flip-Flop

Going to D=0, CLK=1 from D=1, CLK = 0

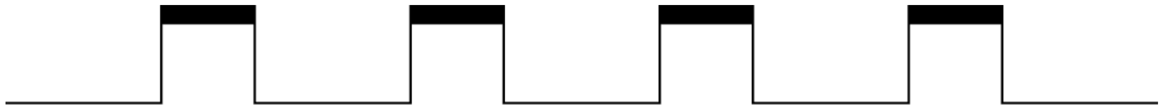
ASRB=1110

- **D=0 => B=1**
- **CLK=1, A=1 => S=0**
- **S=0 => A=1**
- **S=0 => R=1**
- **SR=01 => SET**
- **ASRB=1011**

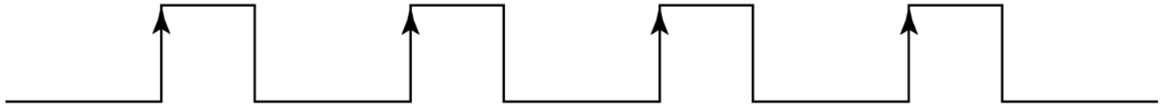
Wrong result

Correct sequence will be:

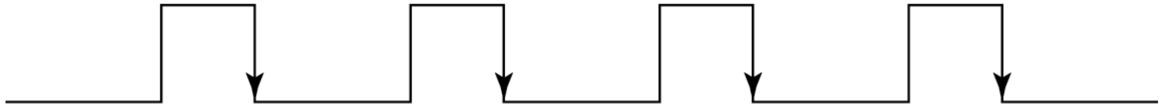
1. **Go from case II to case I, i.e., let ASRB change from 1110 to 0111. Here A takes two gate delays to change. This is Setup time.**
2. **Now go from case I to D=0, CLK=1.**



(a) Response to positive level

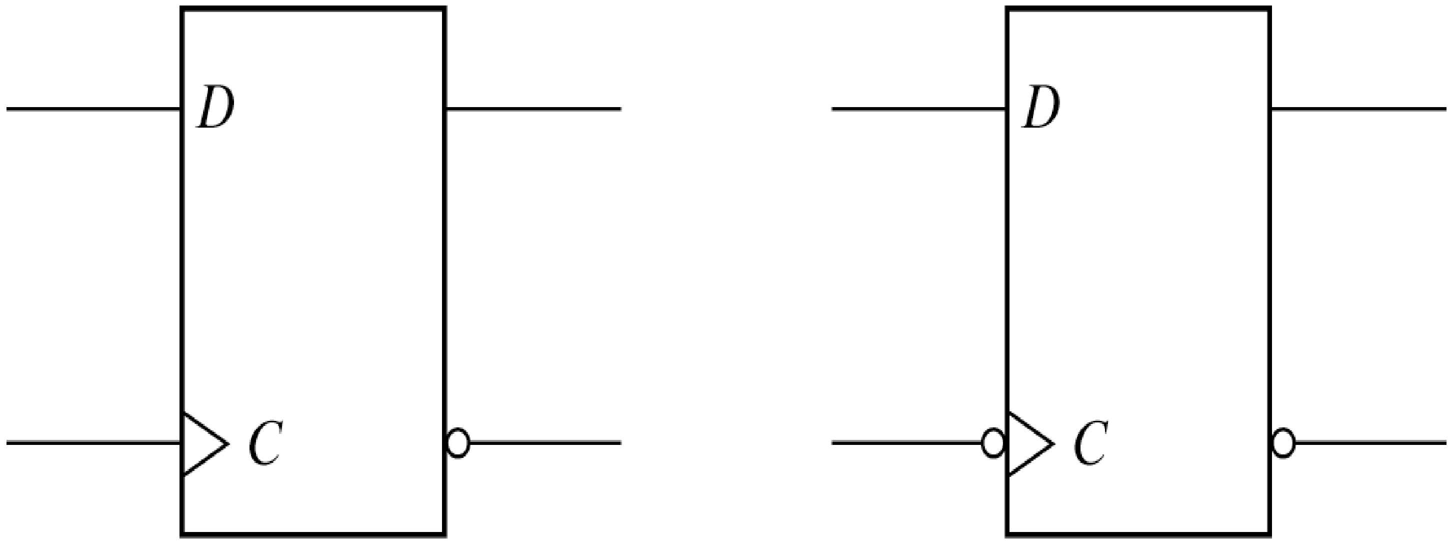


(b) Positive-edge response



(c) Negative-edge response

Fig. 5-8 Clock Response in Latch and Flip-Flop

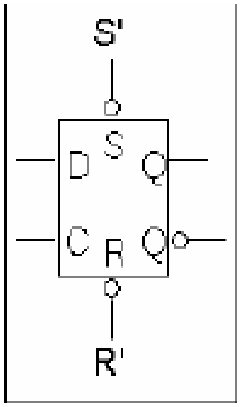


(a) Positive-edge (a) Negative-edge

Fig. 5-11 Graphic Symbol for Edge-Triggered *D* Flip-Flop

Direct inputs

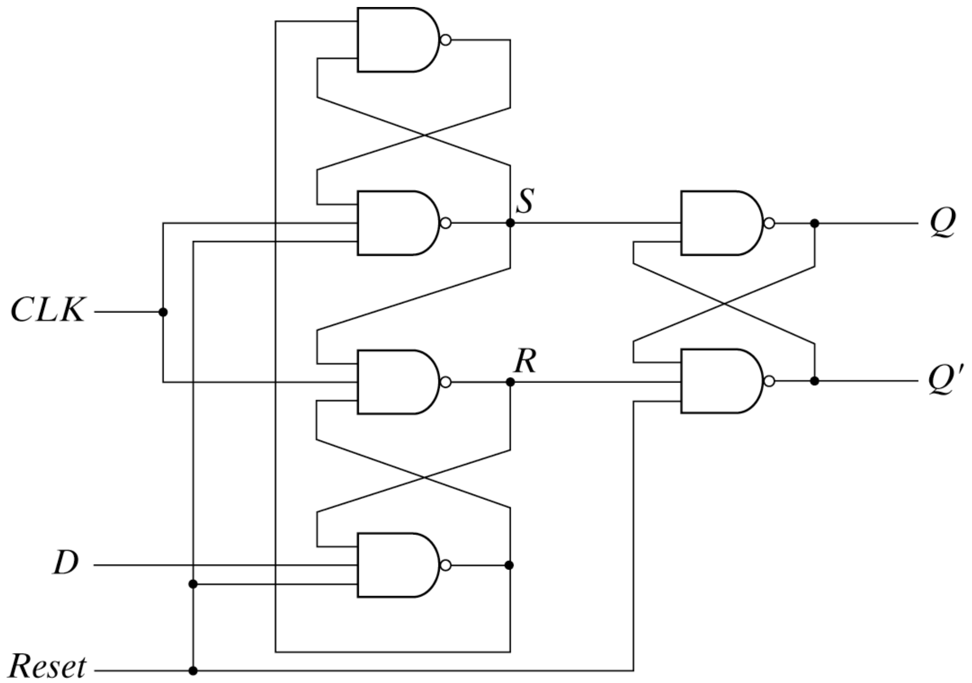
- One last thing to worry about… what is the starting value of Q?
- We could set the initial value synchronously, at the next positive clock edge, but this actually makes circuit design more difficult.
- Most flip-flops provide direct inputs, or asynchronous inputs, that let you immediately set or clear the state, regardless of the clock input.
  - You would "reset" the circuit once, to initialize the flip-flops.
  - The circuit would then begin its regular, synchronous operation.



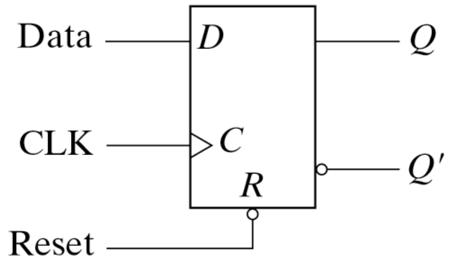
S'	R'	C	D	Q
0	0	x	x	Avoid
0	1	x	x	1 (set)
1	0	x	x	0 (reset)
1	1	0	x	No change
1	1	1	0	0 (reset)
1	1	1	1	1 (set)

Direct inputs set or reset the flip-flop asynchronously

Set S'R' = 11 for normal operation of the flip-flop



(a) Circuit diagram



(b) Graphic symbol

R	*C*	*D*	*Q*	*Q′*
0	X	X	0	1
1	↑	0	0	1
1	↑	1	1	0

(b) Function table

Fig. 5-14 *D* Flip-Flop with Asynchronous Reset

Characteristic tables

- The tables that we've made so far are called characteristic tables.
  - They show the next state Q(t+1) in terms of the current state Q(t) and the inputs.
  - For simplicity, the control input C is usually not listed.
  - Again, these tables don't indicate the positive edge-triggered nature of the flip-flops.

D	Q(t+1)	Operation
0	0	Reset
1	1	Set

J	K	Q(t+1)	Operation
0	0	Q(t)	No change
0	1	0	Reset
1	0	1	Set
1	1	Q'(t)	Complement

T	Q(t+1)	Operation
0	Q(t)	No change
1	Q'(t)	Complement

Characteristic equations

- We can also write characteristic equations, where the next state Q(t+1) is defined in terms of the current state Q(t) and the flip-flop inputs.

D	Q(t+1)	Operation
0	0	Reset
1	1	Set

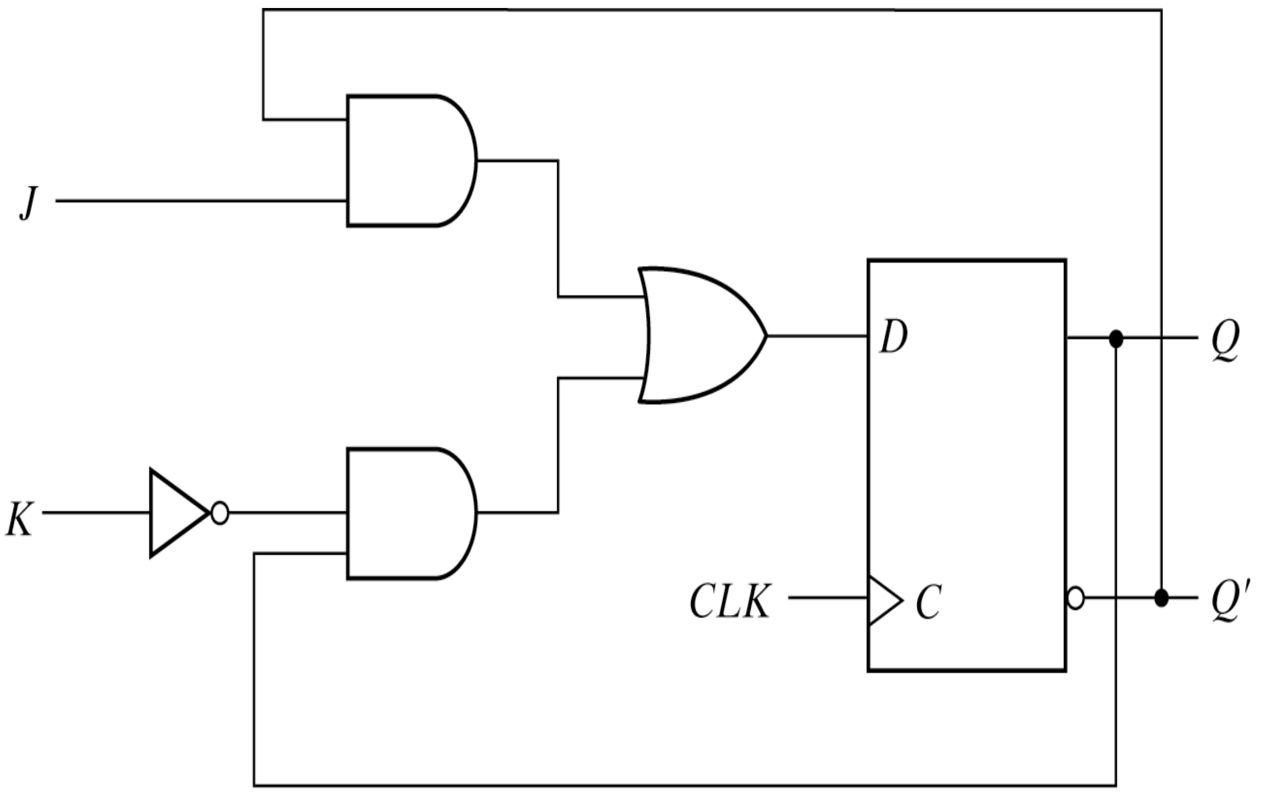
$Q(t+1) = D$

J	K	Q(t+1)	Operation
0	0	Q(t)	No change
0	1	0	Reset
1	0	1	Set
1	1	Q'(t)	Complement

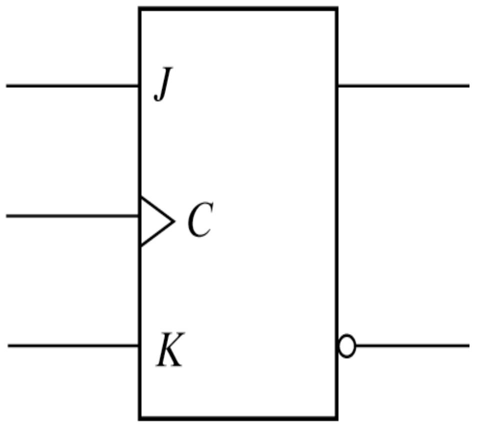
$Q(t+1) = K'Q(t) + JQ'(t)$

T	Q(t+1)	Operation
0	Q(t)	No change
1	Q'(t)	Complement

$$\begin{aligned} Q(t+1) &= T'Q(t) + TQ'(t) \\ &= T \oplus Q(t) \end{aligned}$$

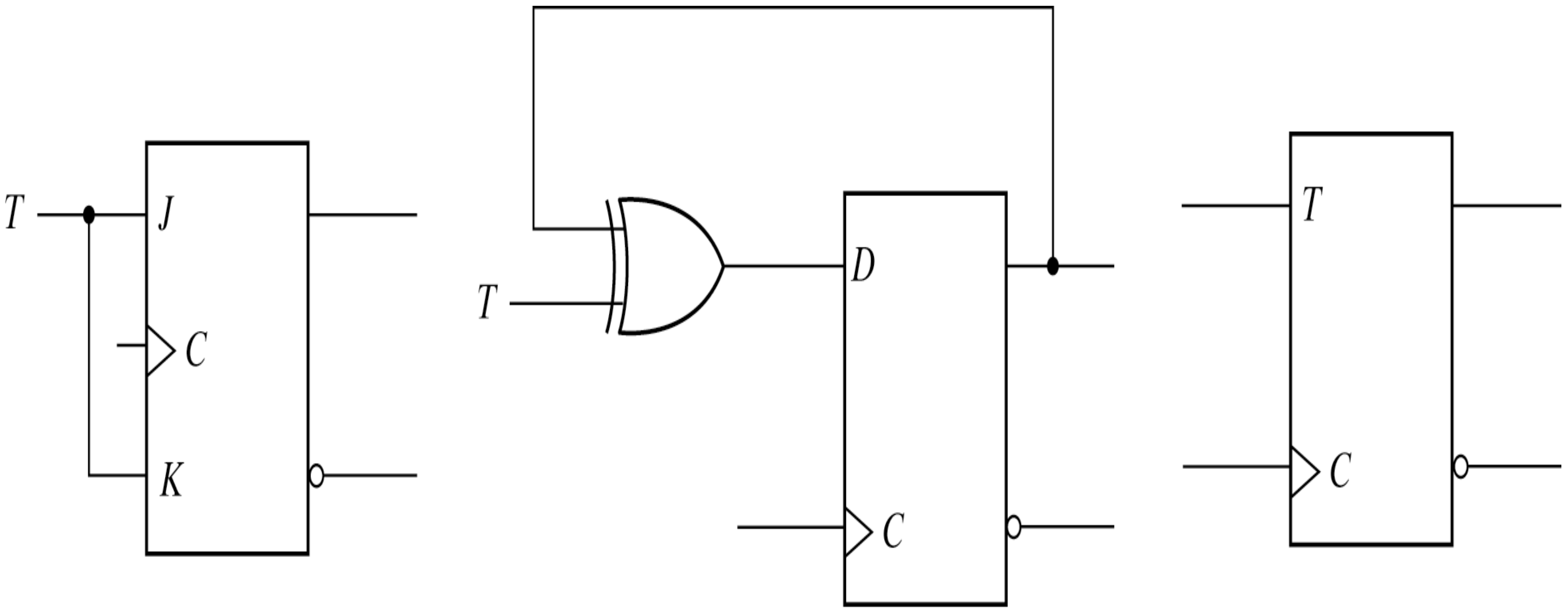


(a) Circuit diagram



(b) Graphic symbol

Fig. 5-12 *JK* Flip-Flop



(a) From *JK* flip-flop (b) From *D* flip-flop (c) Graphic symbol

Fig. 5-13 T Flip-Flop