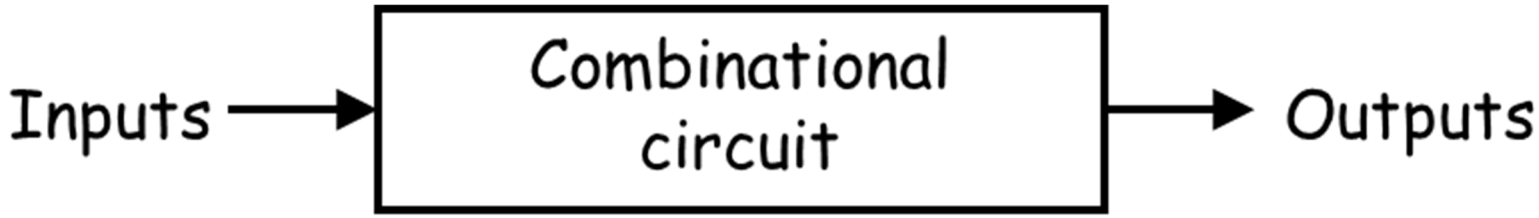
Lecture 8

Latches & Flip-flops

Learning outcomes

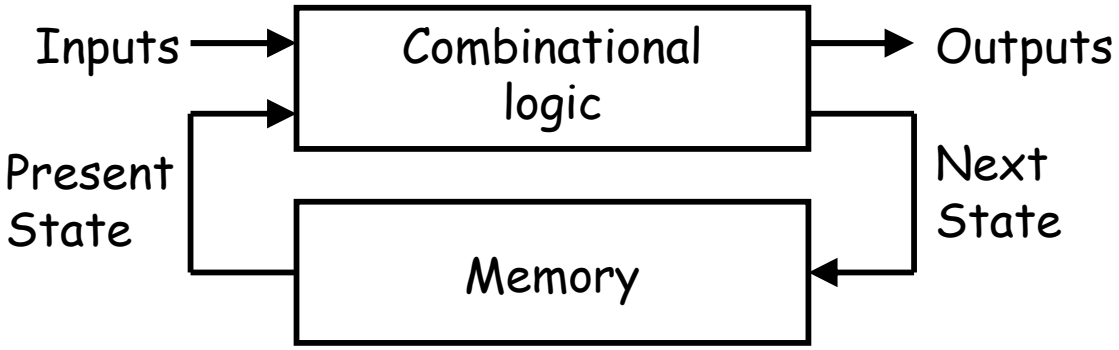
- Describe the difference between combinational and sequential systems.
- Describe the difference between synchronous and asynchronous systems.
- Construct and analyze the operation of latches made from NAND or NOR gates.
- Construct flip-flops using latches.
- Describe the operation of D, JK, and T flip-flops.

Combinational circuits



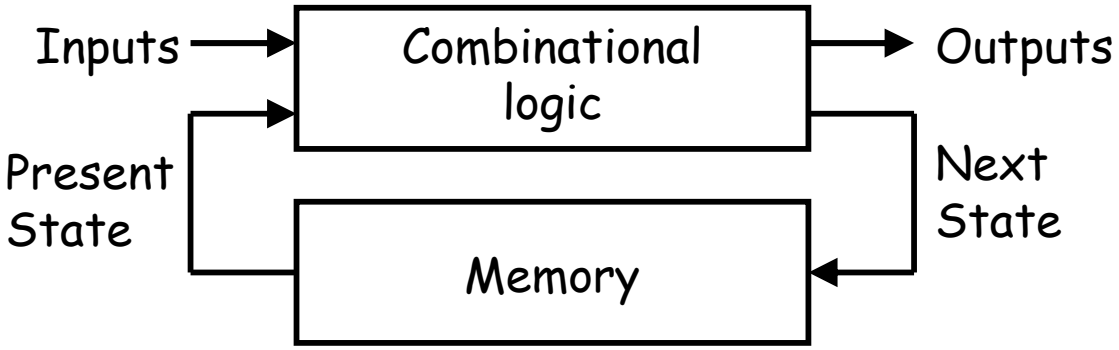
- So far you've just worked with **combinational circuits**, where applying the same inputs always produces the same outputs.
- This corresponds to a mathematical function, where every input has a single, unique output.
- Outputs is thus only a function of current inputs to the circuit.
- There is No internal memory.

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 means that the same inputs can yield *different* outputs, depending on what's stored in memory.
- Computers are sequential! For example, key presses and mouse clicks mean different things depending on which program is running and the state of that program.

Sequential Circuits

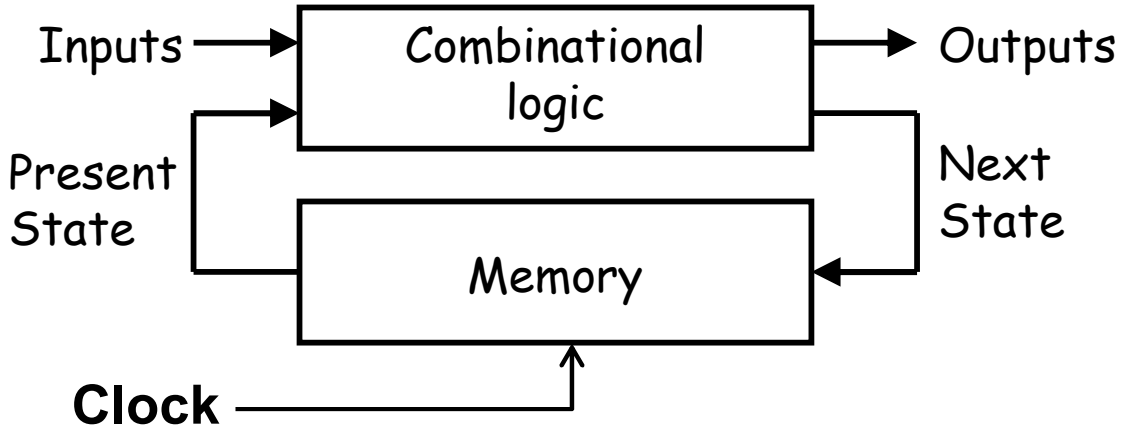


- A sequential circuit consists of memory elements and combinational logic:
  - Memory elements (latches or flip-flops) that store the Present State
  - Combinational logic that computes:
    - the circuit's outputs, which depend on the inputs and Present State
    - the circuit's Next State, which also depends on the inputs and Present State

Two Types of Sequential Circuits

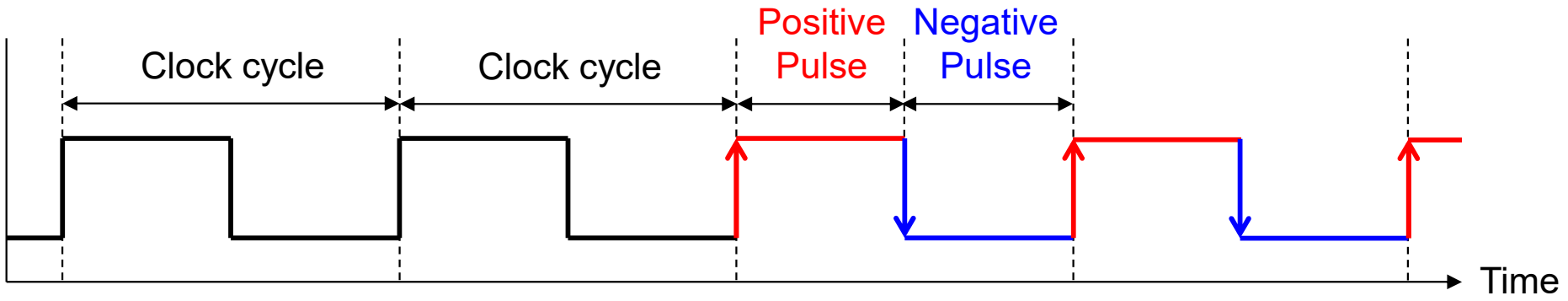
- **Synchronous** Sequential Circuit
  - Uses a clock signal as an additional input
  - Changes in the memory elements are controlled by the clock
  - Changes happen at discrete instances of time
- **Asynchronous** Sequential Circuit
  - No clock signal
  - Changes in the memory elements can happen at any instance of time
- Our focus will be on **Synchronous Sequential Circuits**
  - Easier to design and analyze than asynchronous sequential circuits

Synchronous Sequential Circuits



- Synchronous sequential circuits use a **clock signal**
- The clock signal is an input to the memory elements
- The clock determines **when** the memory should be updated
- The **present state** = output value of memory (stored)
- The **next state** = input value to memory (not stored yet)

The Clock



- Clock is a periodic signal = Train of pulses (1's and 0's)
- The same clock cycle repeats indefinitely over time
- **Positive Pulse**: when the **level** of the clock is **1**
- **Negative Pulse**: when the **level** of the clock is **0**
- **Rising Edge**: when the clock goes **from 0 to 1**
- **Falling Edge**: when the clock goes **from 1 down to 0**

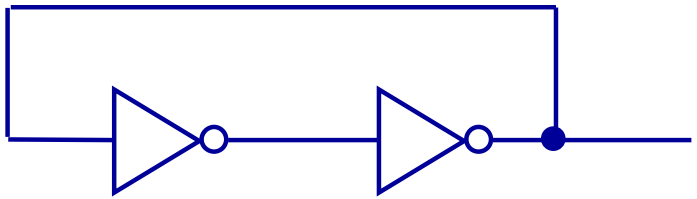
What exactly is memory?

- A memory should have at least three properties.
  - It should be able to hold a value.
  - You should be able to read the value that was saved.
  - You should be able to change the value that was saved.

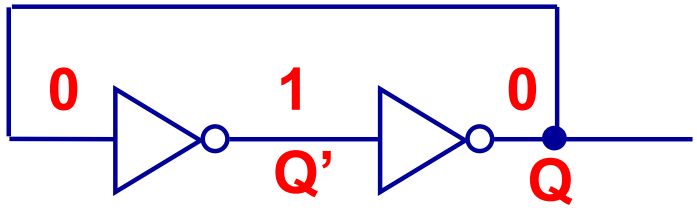
- We'll start with the simplest case, a one-bit memory.
  - It should be able to hold a single bit, 0 or 1.
  - You should be able to read the bit that was saved.
  - You should be able to change the value. Since there's only a single bit, there are only two choices:
    - Set the bit to 1
    - Reset, or clear, the bit to 0.

The basic idea of storage

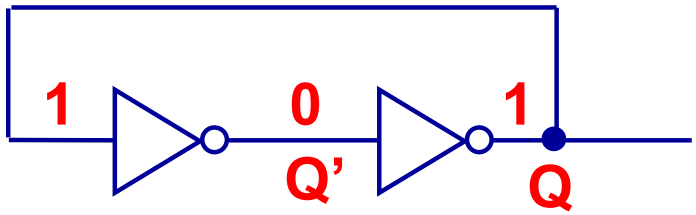
- How can a circuit "remember" anything, when it's just a bunch of gates that produce outputs according to the inputs?
- The basic idea is to use positive feedback to maintain storage indefinitely. We make a loop, so the circuit outputs are also inputs.



Two inverters, with **feedback**



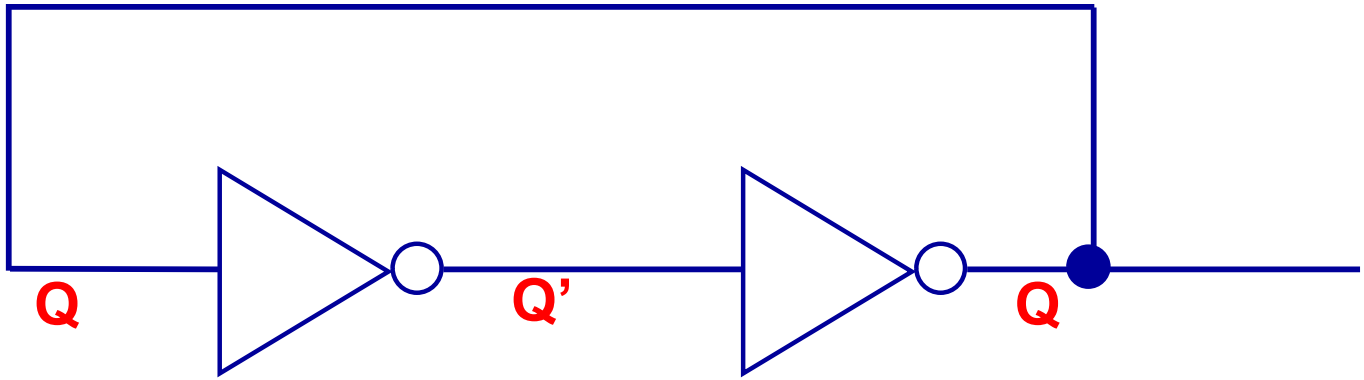
If the first input is **0**, a **0** gets fed back into it



If the first input is **1**, a **1** gets fed back into it

This circuit will hold its **state** forever - stable

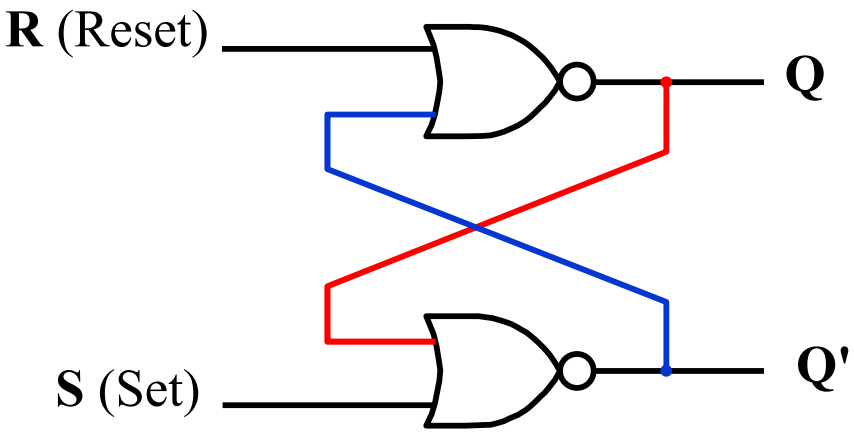
The basic idea of storage



- Does this satisfy the properties of memory?
  - These circuits "remember" the output Q because its value never changes. Similarly, Q' never changes either.
  - We can also "read" the output Q by attaching a probe or another circuit.
  - But we can't change the output Q! There are no external inputs here, so we can't control whether the output is Q=1 or Q=0.

RS latch

- Let's use NOR gates instead of inverters. The **RS (Reset-Set) latch** below has two inputs R and S, which will let us control the outputs Q and Q'.



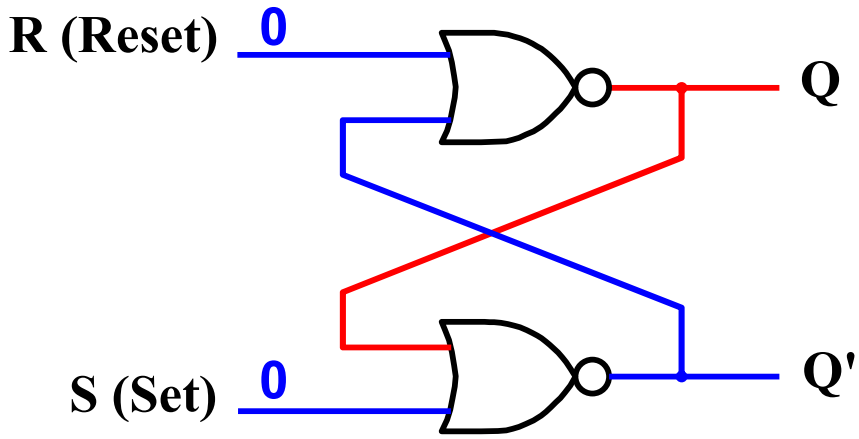
- Here Q and Q' feed back into the circuit. They're not only outputs, they're also inputs!
- To figure out how Q and Q' change, we have to look at not only the inputs S and R, but also the *current* values of Q and Q':

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

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

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

Store operation: RS = 00



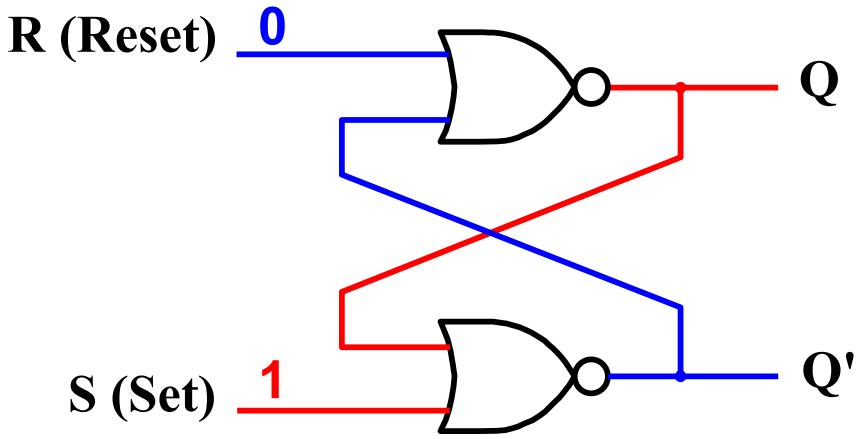
$$Q_{next} = (R + Q'_{current})'$$
$$Q'_{next} = (S + Q_{current})'$$

- The above equations reduce to:

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

- So when RS=00, then $Q_{next} = Q_{current}$. Whatever value Q has, it keeps.
- This is exactly what we need to **store** values in the latch.

Set operation: RS = 01



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

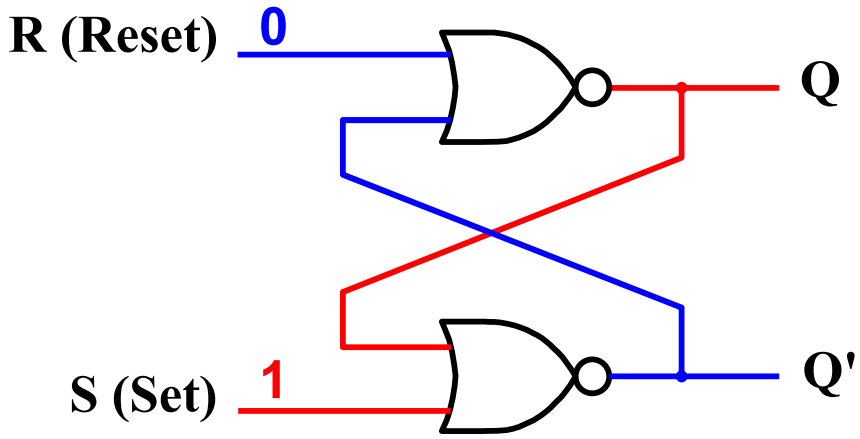
- Since S = 1, Q'_{next} is 0, *regardless* of $Q_{current}$: $Q'_{next} = (1 + Q_{current})' = 0$
- Then, this new value of Q' goes into the top NOR gate, along with R = 0

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

- 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 that 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**.

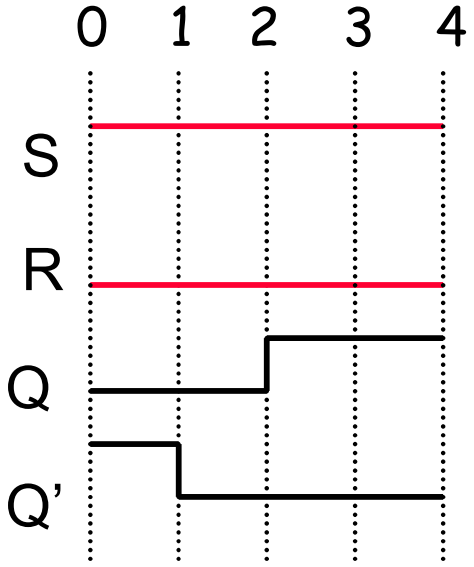
Latch delays

- Timing diagrams are especially useful in understanding how sequential circuits work. Below is a diagram which shows an example of how our latch outputs change with inputs **RS=01**.

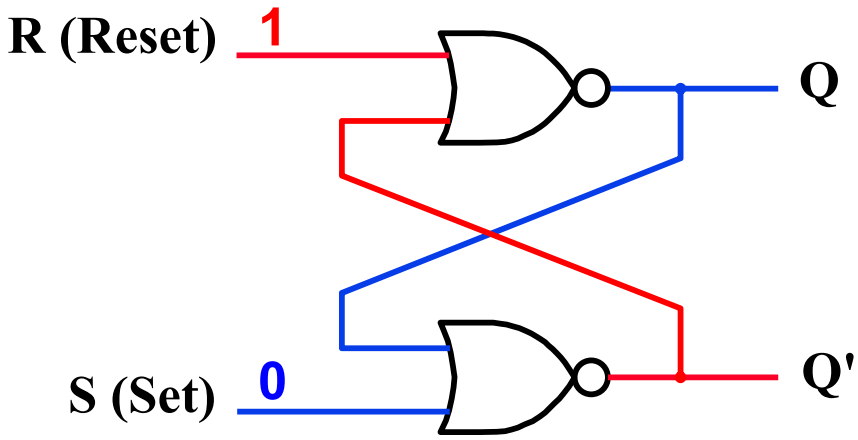


$$Q_{next} = (R + Q'_{current})'$$
$$Q'_{next} = (S + Q_{current})'$$

- Suppose that initially, Q = 0 and Q' = 1.
- Since S=1, Q' will change from 1 to 0 after one NOR-gate delay (marked by dotted vertical lines in the diagram for clarity).
- This change in Q', along with R=0, causes Q to become 1 after another gate delay.
- The latch then stabilizes until S or R change again.



Reset operation: RS = 10



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

- 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.

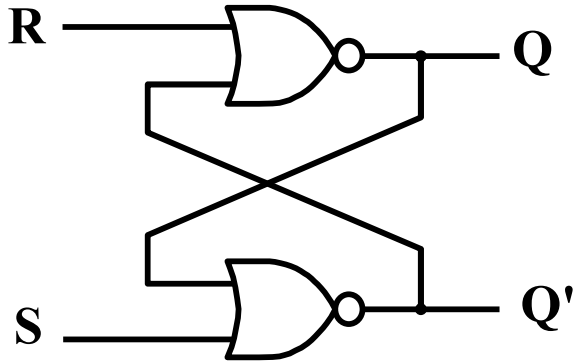
$$Q'_{next} = (0 + 0)' = 1$$

- 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}.

RS latches are memories!

- This little table shows that our latch provides everything we need in a memory: we can set it, reset it, and remember the current value.
- The output Q represents the data stored in the latch. It is sometimes 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 depend on their *current* values, as well as on the inputs R and S.

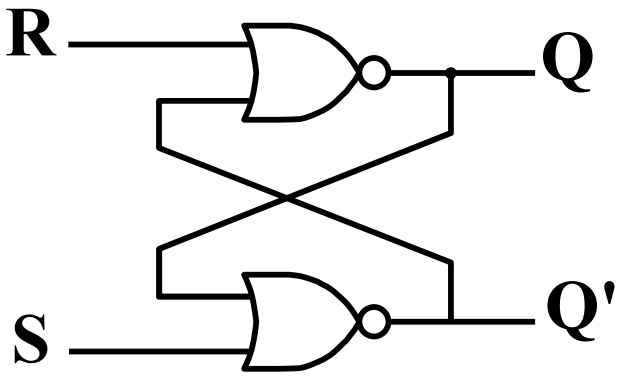
S	R	Q
0	0	No change
0	1	0 (reset)
1	0	1 (set)

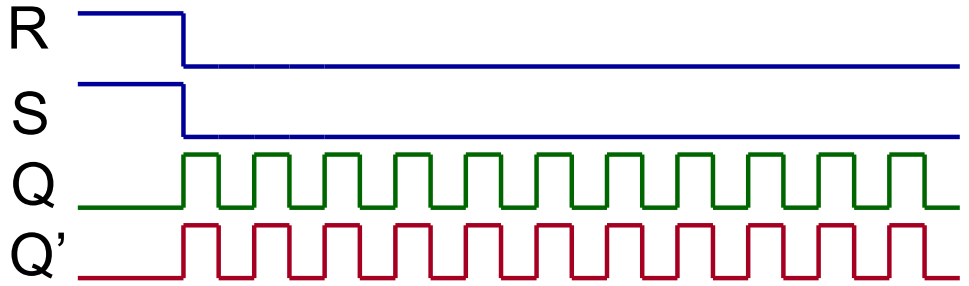


Inputs		Current		Next	
S	R	Q	$\overline{Q}$	Q	$\overline{Q}$
0	0	0	1	0	1
0	0	1	0	1	0
0	1	0	1	0	1
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	0	1	0

What about R=S=1 ?

- What does it mean to both set and reset at the same time?
- Both Q_{next} and Q'_{next} will become 0. This contradicts the assumption that Q and Q' are always complements.
- Another problem is what happens when **R=S=1** (**Q=Q'=0**) and R or S changes to 0?
  - S changes to zero first → **Reset wins** (Q=0, Q'=1)
  - R changes to zero first → **Set wins** (Q=1, Q'=0)
  - Both change to 0 at same time → **uncontrolled oscillation (unstable)**





What about R=S=1?

- When both R=S change from 1 → 0 at same time:

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

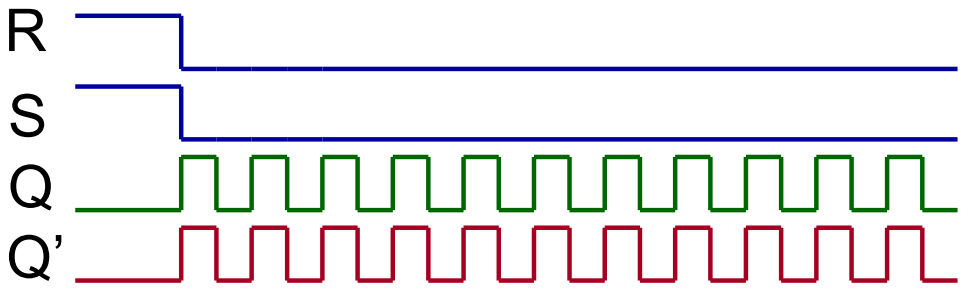
$$Q'_{next} = (0 + 0)' = 1$$

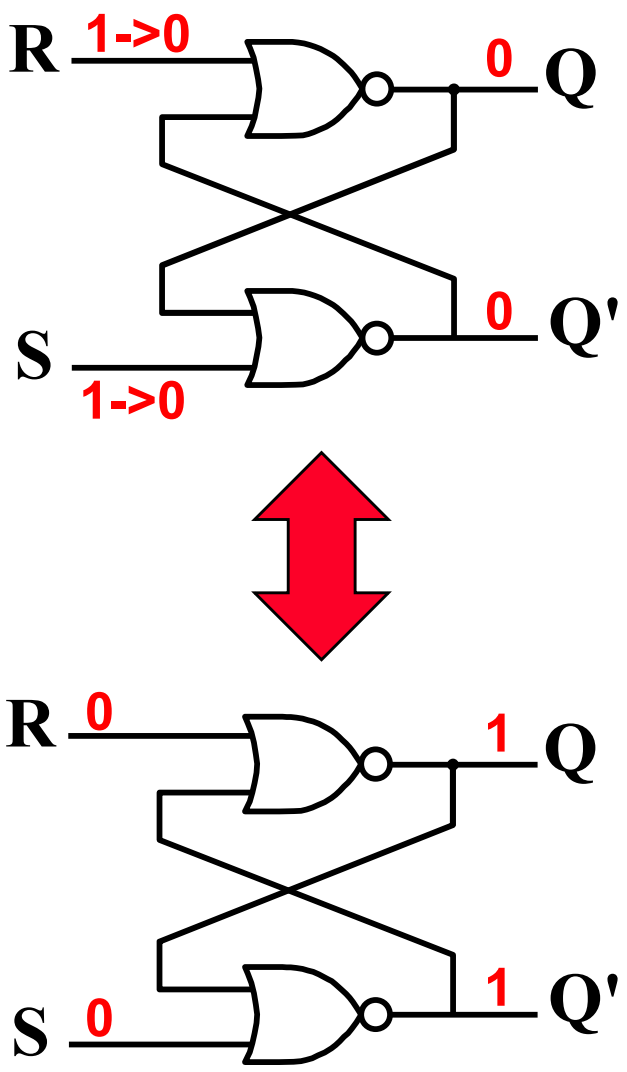
- But these new values go back into the NOR gates, and in the next step we get:

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

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

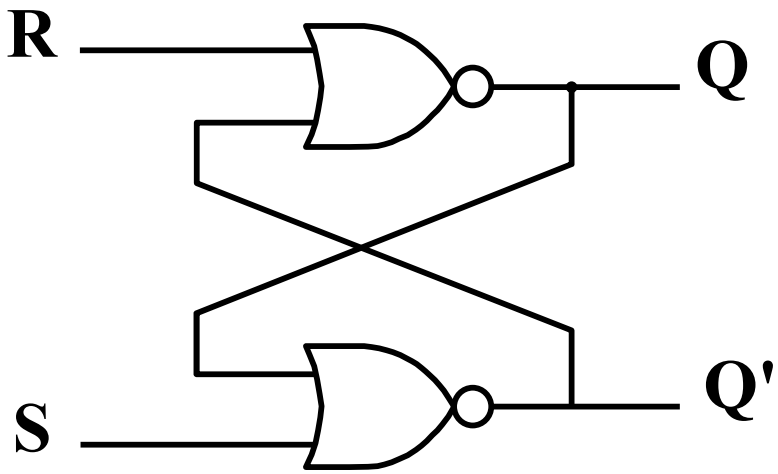
- The circuit enters an infinite loop, where Q and Q' cycle between 0 and 1 forever. This is actually the worst case, but the moral is **don't ever set RS=11 (make it invalid input)**!



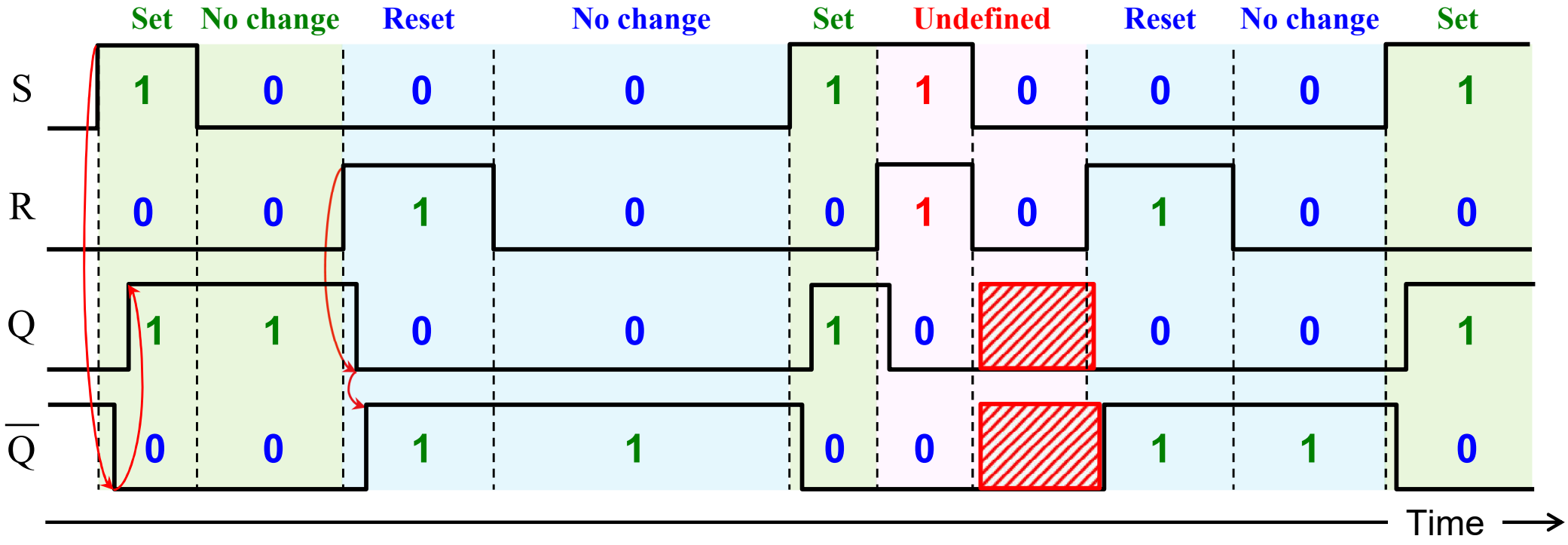


$$Q_{next} = (R + Q'_{current})'$$
$$Q'_{next} = (S + Q_{current})'$$

Timing Diagram for RS latch

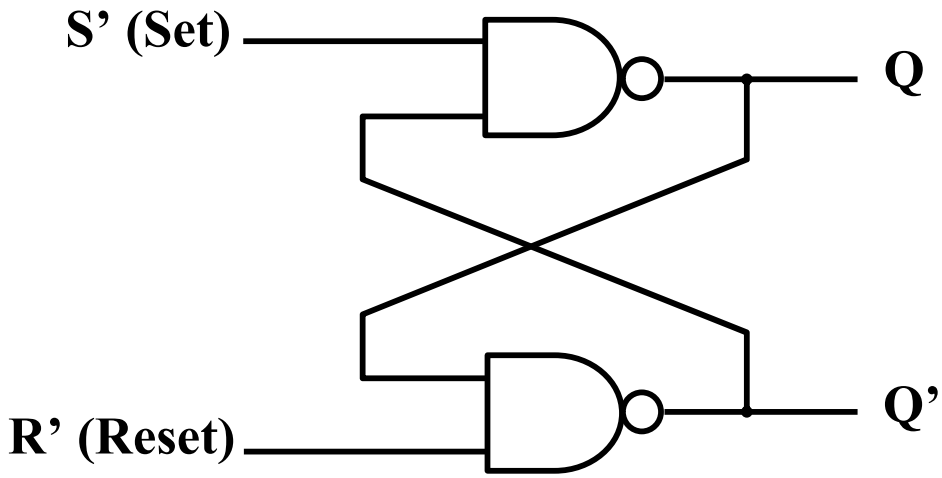


S	R	Q
0	0	No change
0	1	0 (reset)
1	0	1 (set)



R'S' latch

- There are several varieties of latches.
- You can use NAND instead of NOR gates.

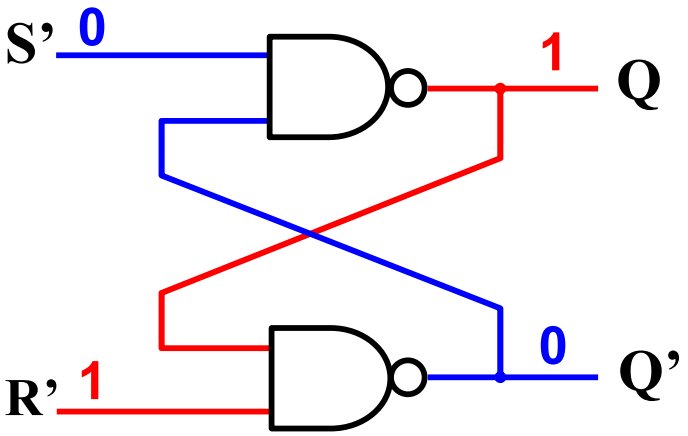


S'	R'	Q
1	1	No change
1	0	0 (reset)
0	1	1 (set)
0	0	Avoid!

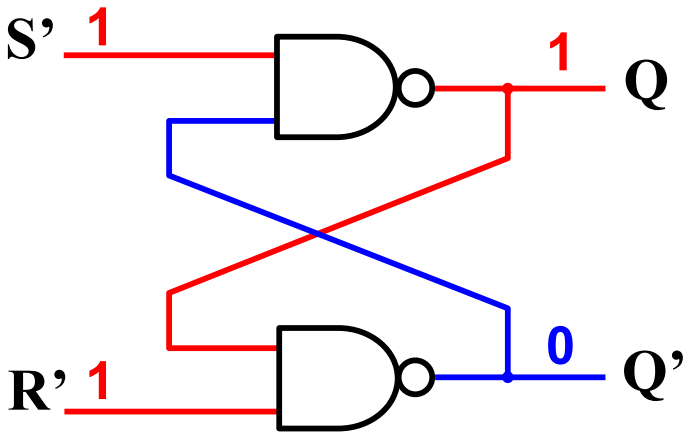
- This is just like an RS latch, but with inverted inputs, as you can see from the table.
- You can derive this table by writing equations for the outputs in terms of the inputs and the current state, just as we did for the RS latch.

R'S' latch operation

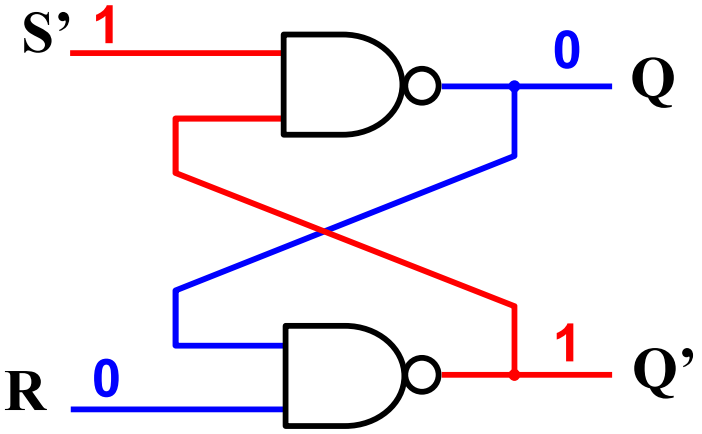
Set Operation



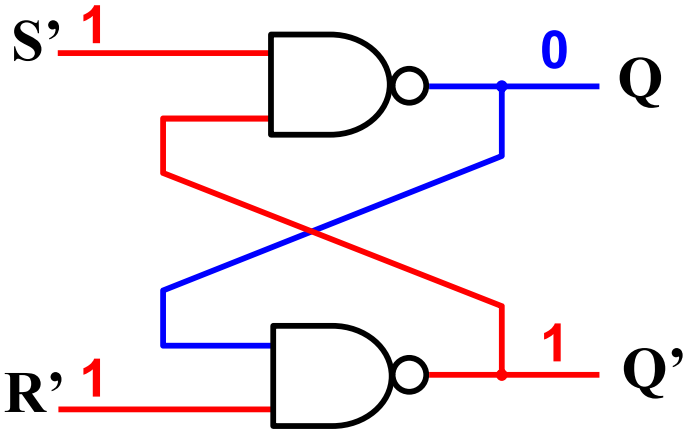
Store Operation



Reset Operation

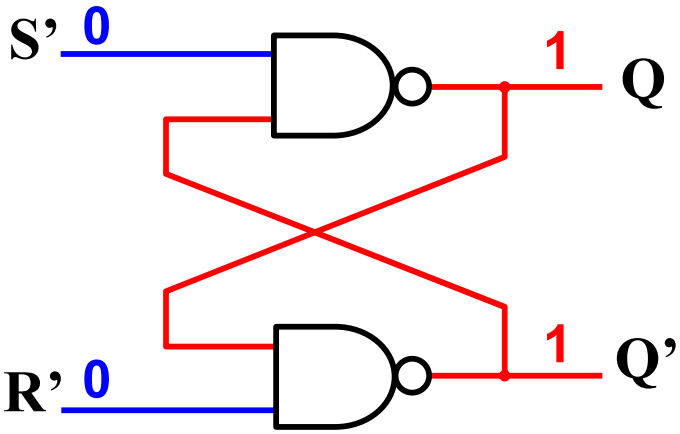


Store Operation

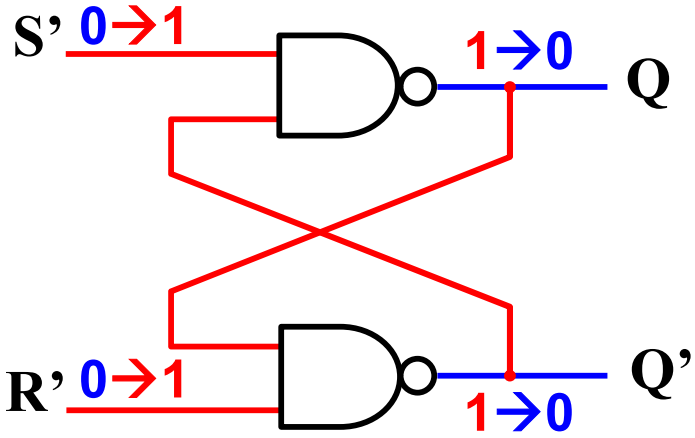


R'S' latch operation – Invalid operation

Invalid Operation

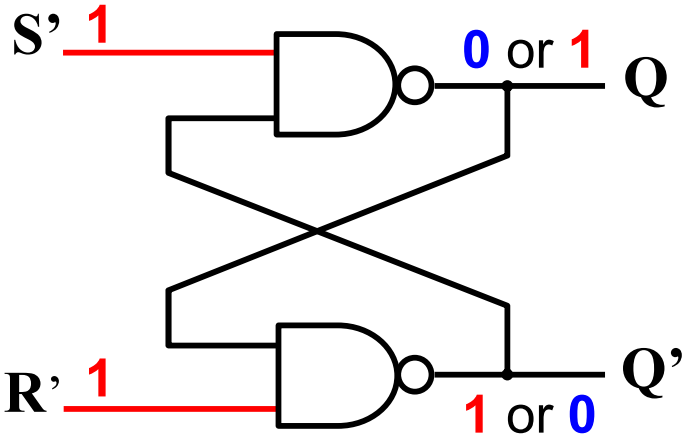


Oscillation Condition



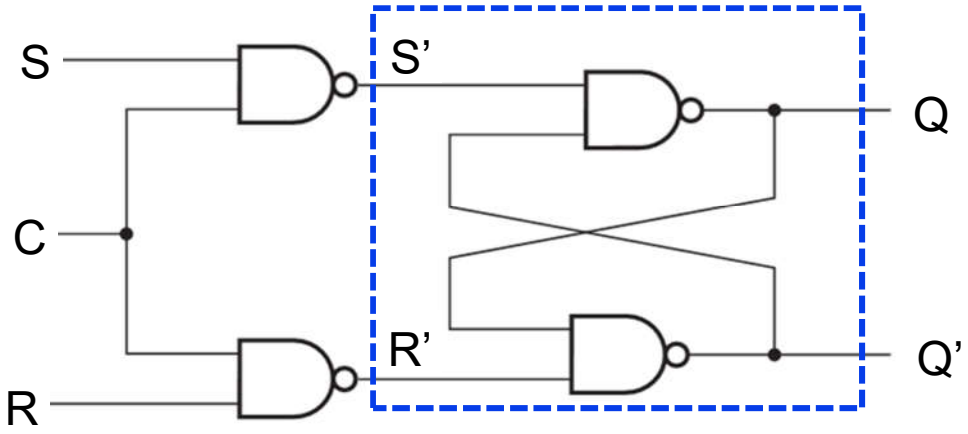
- S' = R' = 0 should never be used
- If S and R change from 0 → 1 simultaneously then oscillation occurs
- Final Q and Q' are unknown.

Unknown State



Enabled (Gated) RS latch

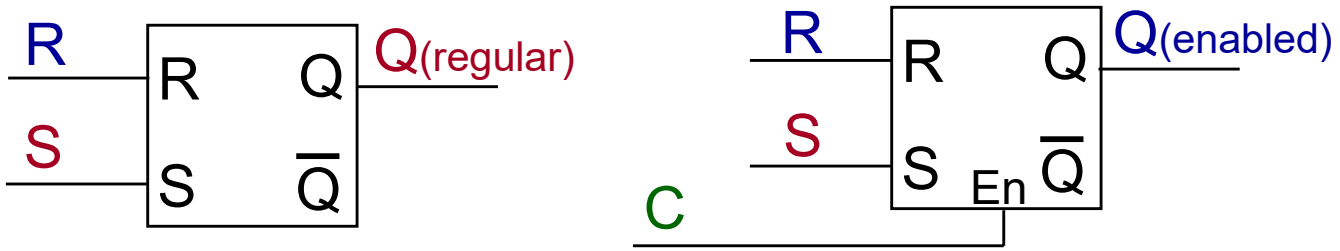
- An additional Enable input signal C is used:

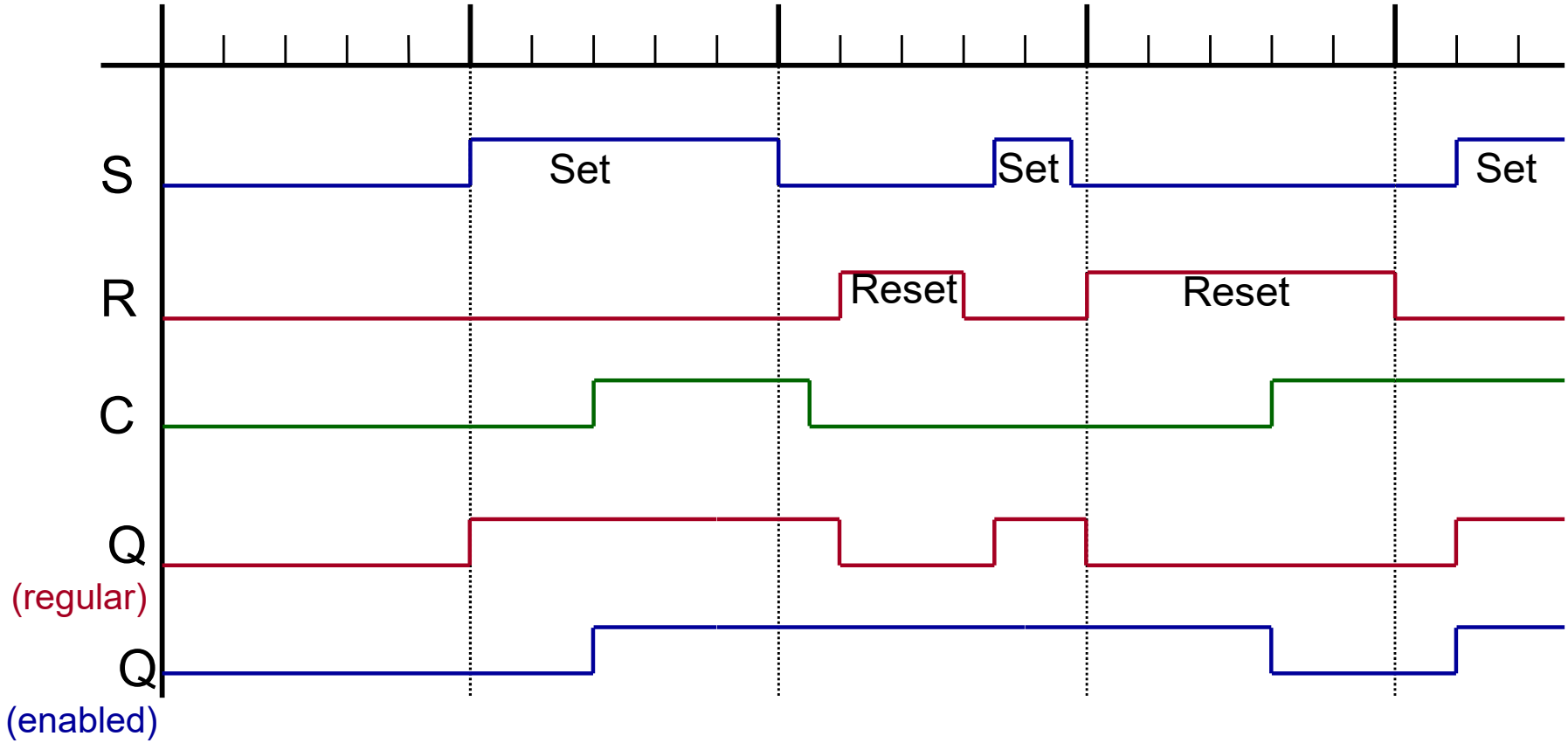


C	S	R	S'	R'	Q
0	x	x	1	1	No change
1	0	0	1	1	No change
1	0	1	1	0	0 (reset)
1	1	0	0	1	1 (set)
1	1	1	0	0	Avoid!

- Notice the hierarchical design!
  - The dotted blue box is the R'S' latch from the previous slide.
  - The first additional NAND gates are simply used to generate the correct inputs for the S'R' latch. The NAND gates invert the **S** and **R** inputs when **C=1**
- The control input acts just like an enable.
- Clock controls when the state of the latch can be changed
- When **C=0**, the latch remains in the same state
- When **C=1**, then normal latch operation

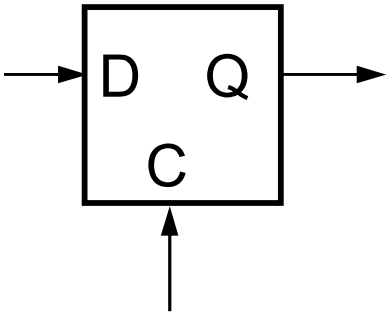
Regular vs. Enabled RS latches



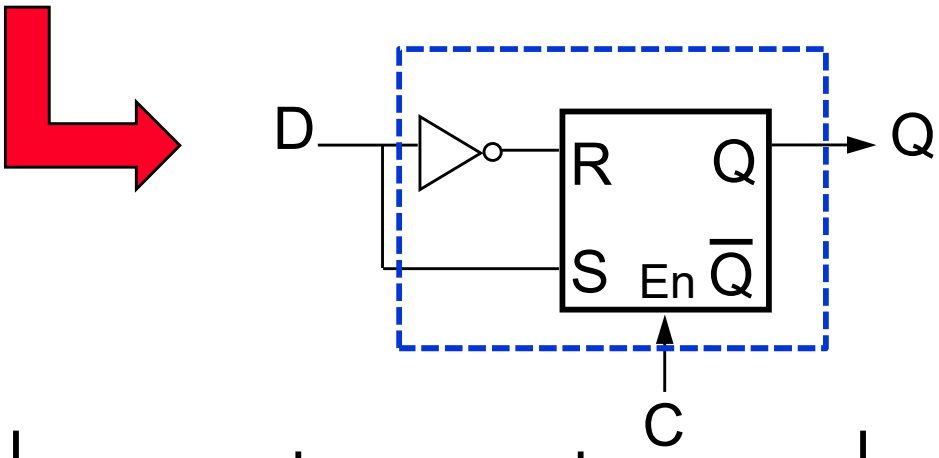


- Latch only changes when enable is asserted
- Assume above that latches have no time delay (ideal)

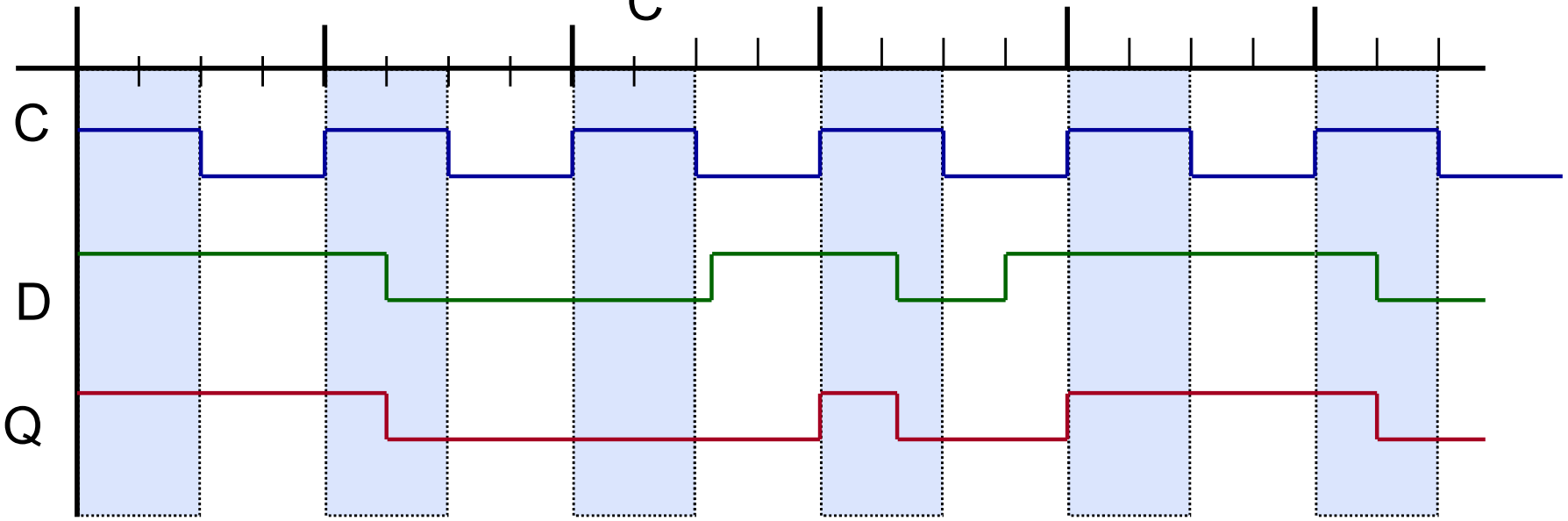
Enabled/Gated D-Latch



- One data input D → S=D and R=D'
- No undefined state since S ≠ R
- **Output follows (Q=D) input** when latch is **enabled**
- **No change** to output when latch is **NOT enabled**

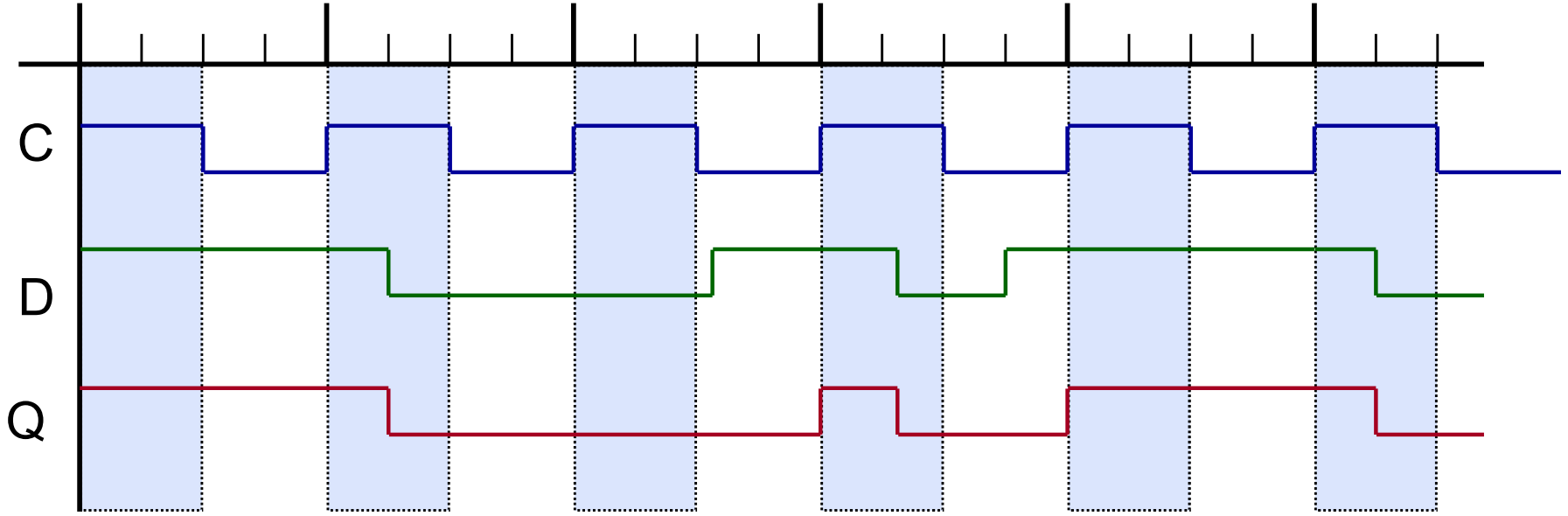


C	D	Q
0	x	No change
1	0	0
1	1	1



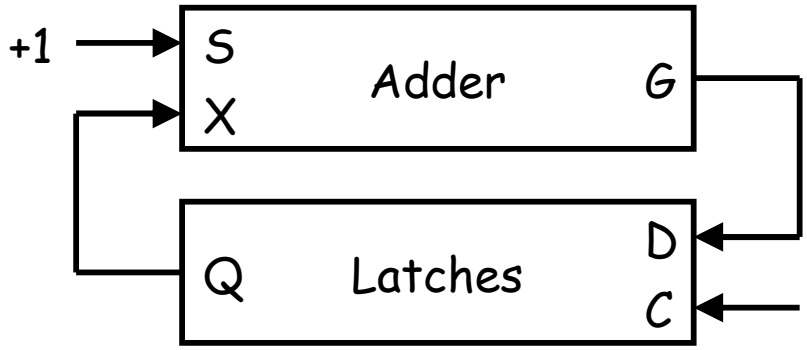
Level-sensitive memory

- A latch is a level Sensitive memory: as long as the enable signal is asserted, output can change
- In a Level-sensitive memory, output may change a number of times depending on the duration of the enable pulse.



Combinational Cycles - Example

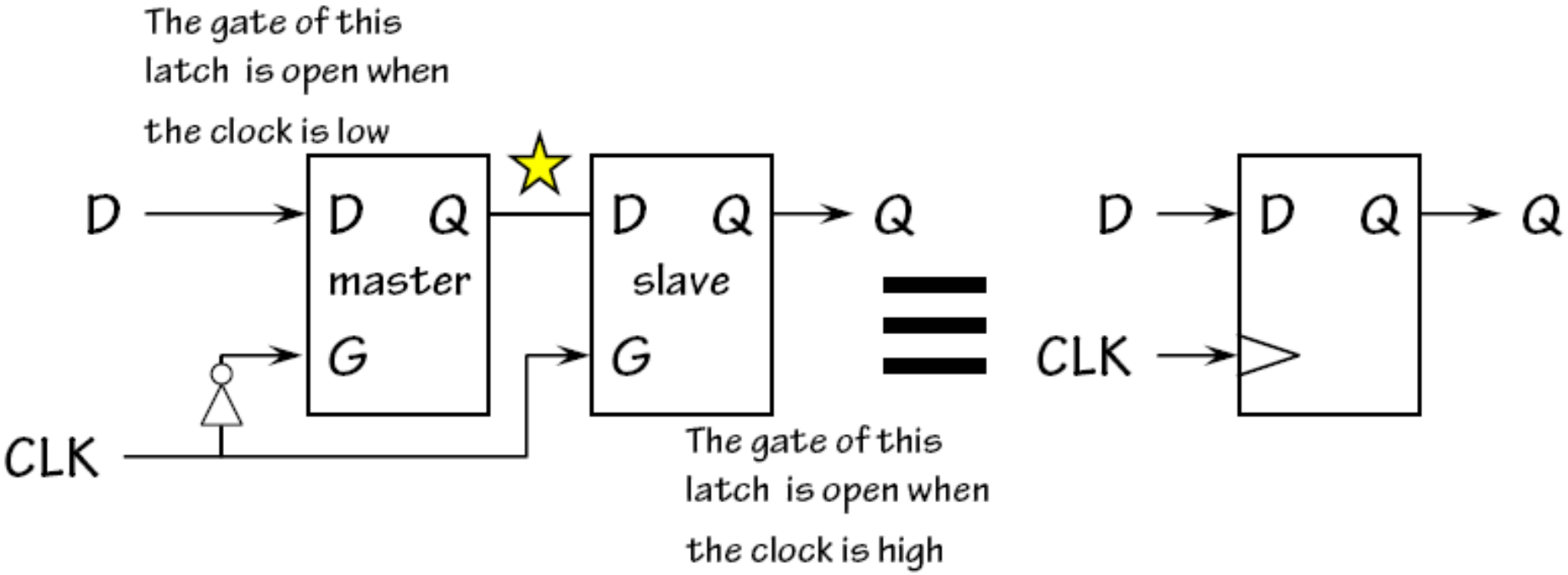
- We can connect a latch to an adder combinational circuit



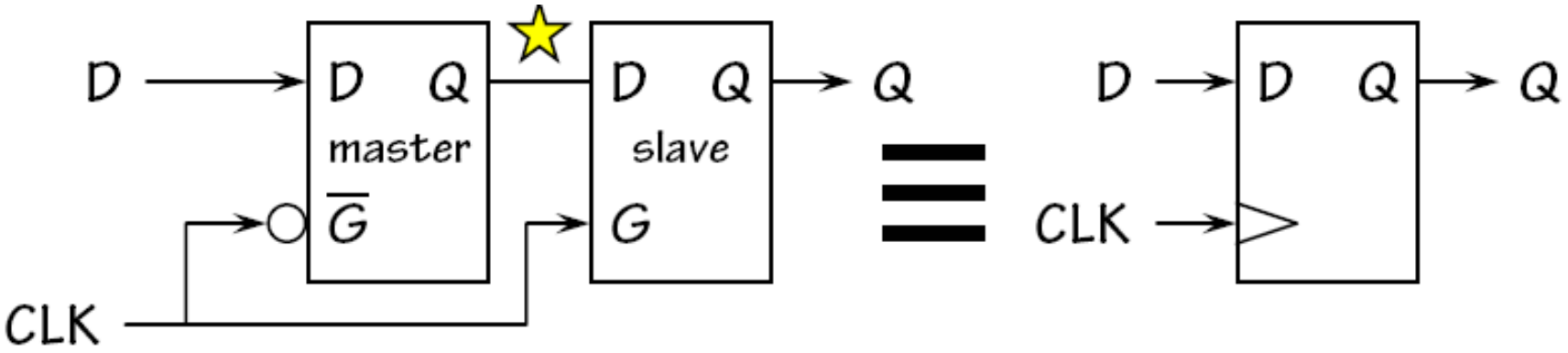
- Let's say these latches contain some value that we want to increment.
  - The Adder read the current latch value.
  - It applies the "G = X + 1" operation.
  - The incremented value is stored back into the latches.
- At this point, we have to stop the cycle, so the latch value doesn't get incremented again by accident.

Master Slave Flip-Flop

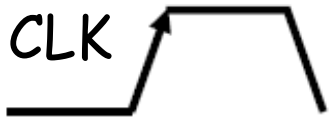
- This solution consists in adding two “gates” and open one at a time
- Master-Slave D Flip-Flop is a combination of 2 latches. The first latch is master and responds to one transition of the clock and the 2nd latch to the other transition to the clock.
- Therefore, the final output changes during one transition of clock



Master Slave Flip-Flop

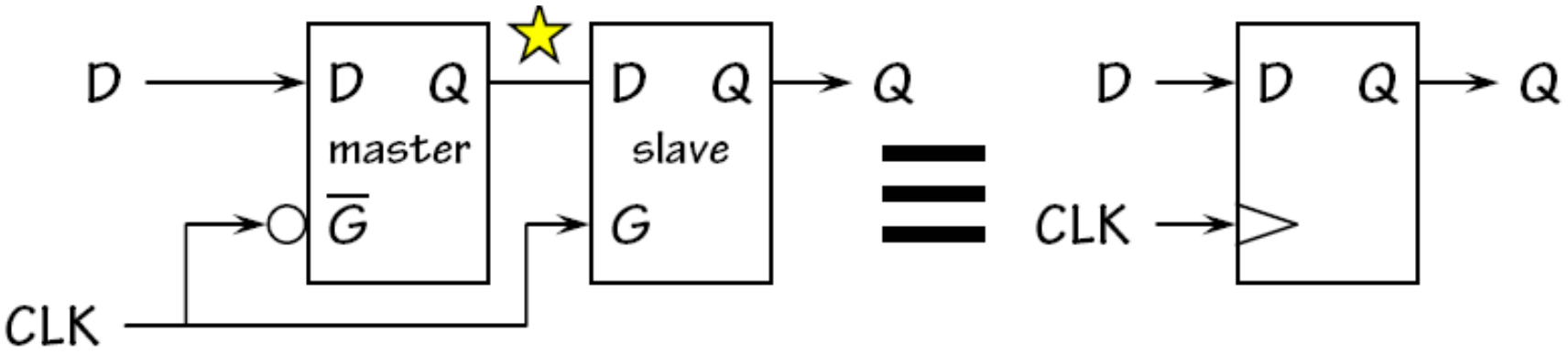


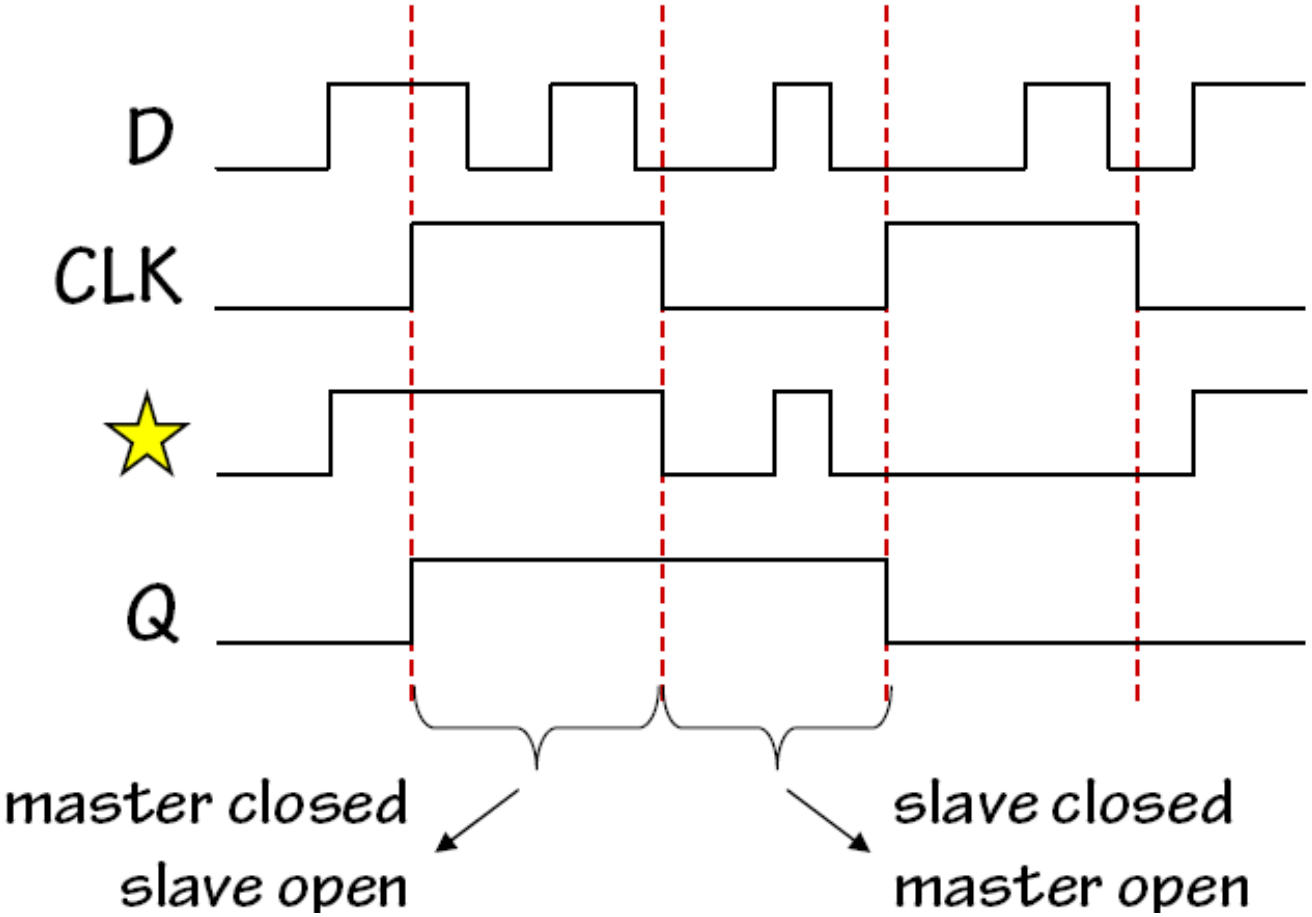
- Only one latch "transparent" at any time:
  - master closed when slave is open
  - slave closed when master is open
  - no combinational path through flip flop
- Q only changes shortly after 0 → 1 transition of CLK, so flip flop appears to be "triggered" by rising edge of CLK:



- Major disadvantage is that control inputs (e.g input D) must be held stable when the master latch is transparent, that is for CLK low.

Master Slave Flip-Flop

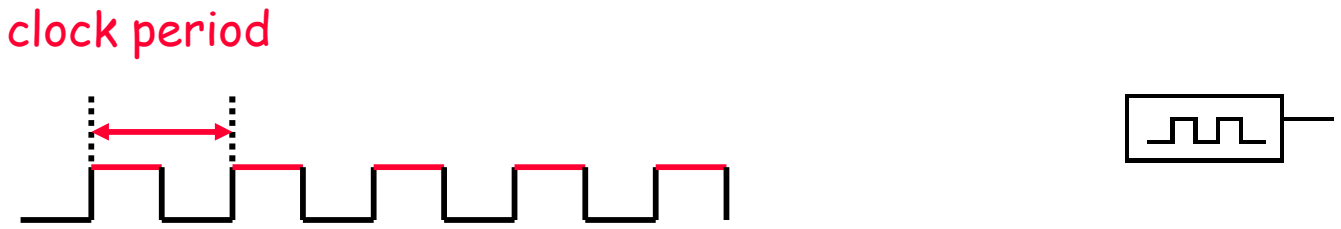




Clocks and Flip-Flops

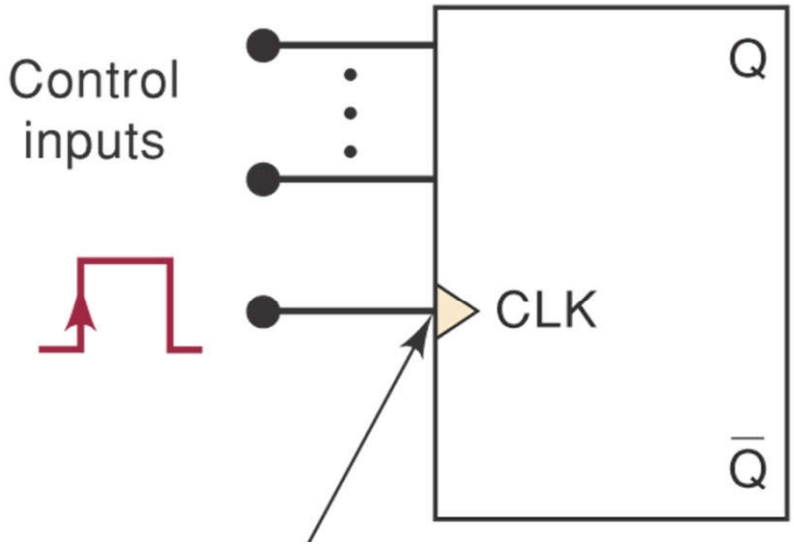
- The key to proper operation is to modify the latches to form flip-flops, which are memory elements that can change state only at well-defined times, that is on one of the other's clock transitions (from 0 to 1 for a rising/positive clock edge or from 1 to 0 for a falling/negative clock edge.

- Clocks are often used to synchronize circuits:
  - They generate a repeating, predictable pattern of 0s and 1s that can trigger certain events in a circuit, such as writing to a latch.
  - If several circuits share a common clock signal, they can coordinate their actions with respect to one another.

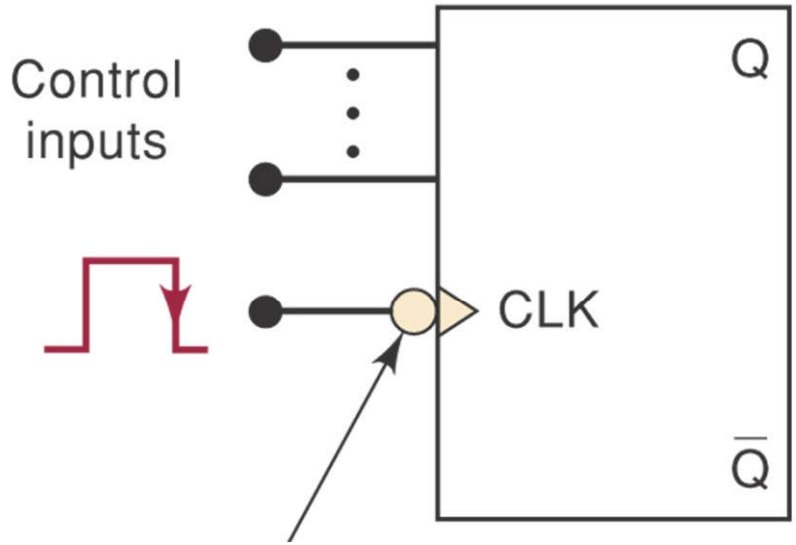


Symbols for Edge-Triggering

- Edge-Triggered Flip-Flop: Transition (output change) can happen only during clock pulse transition
- Clock pulse transition can be positive/rising clock transition or negative/falling clock transition
- The “triangle” symbol means edge-triggered

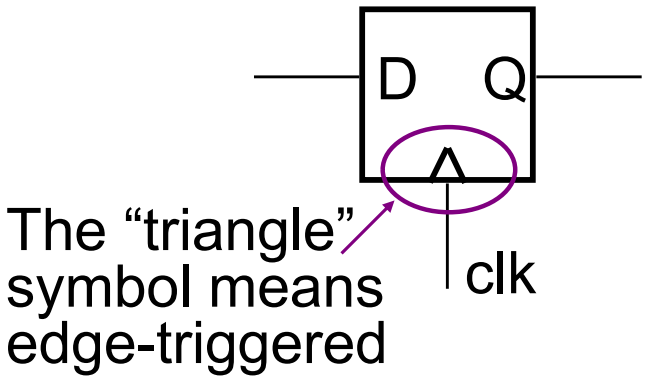


Positive-Edge Triggering

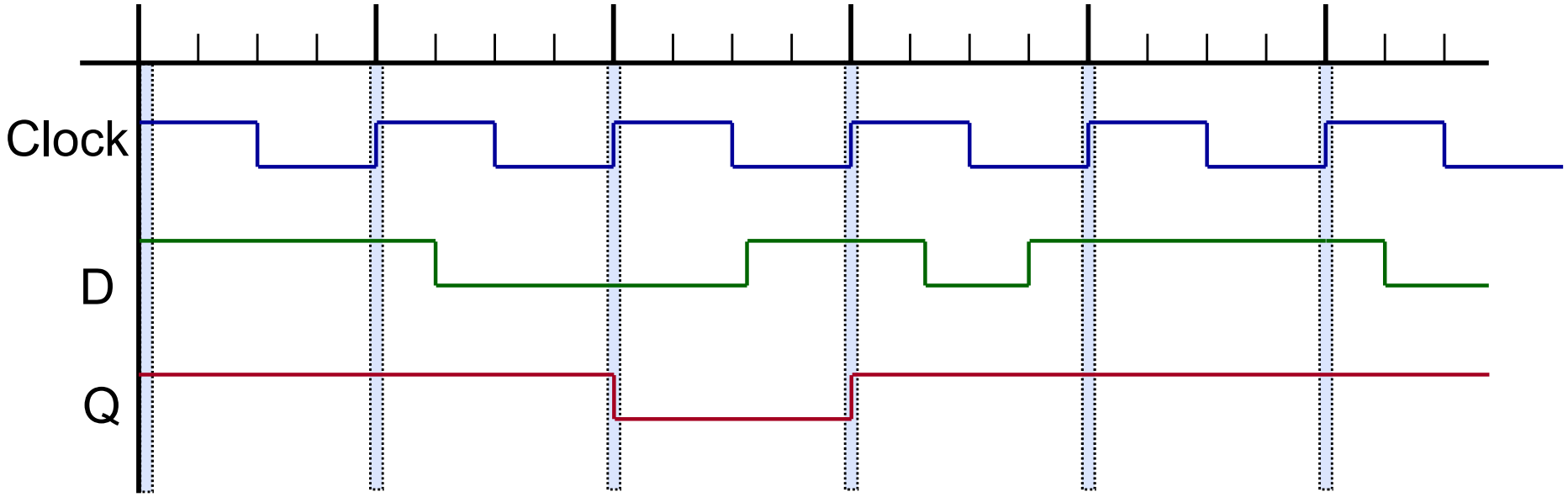


Negative-Edge Triggering

Positive Edge-triggering

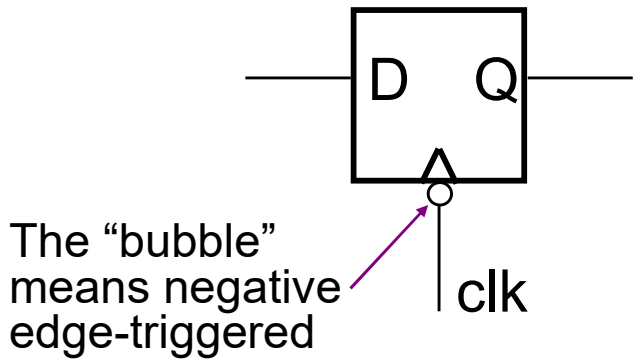


In a positive edge-triggered D Flip-Flop, the output looks at the input only during the instant that the clock changes from low to high.

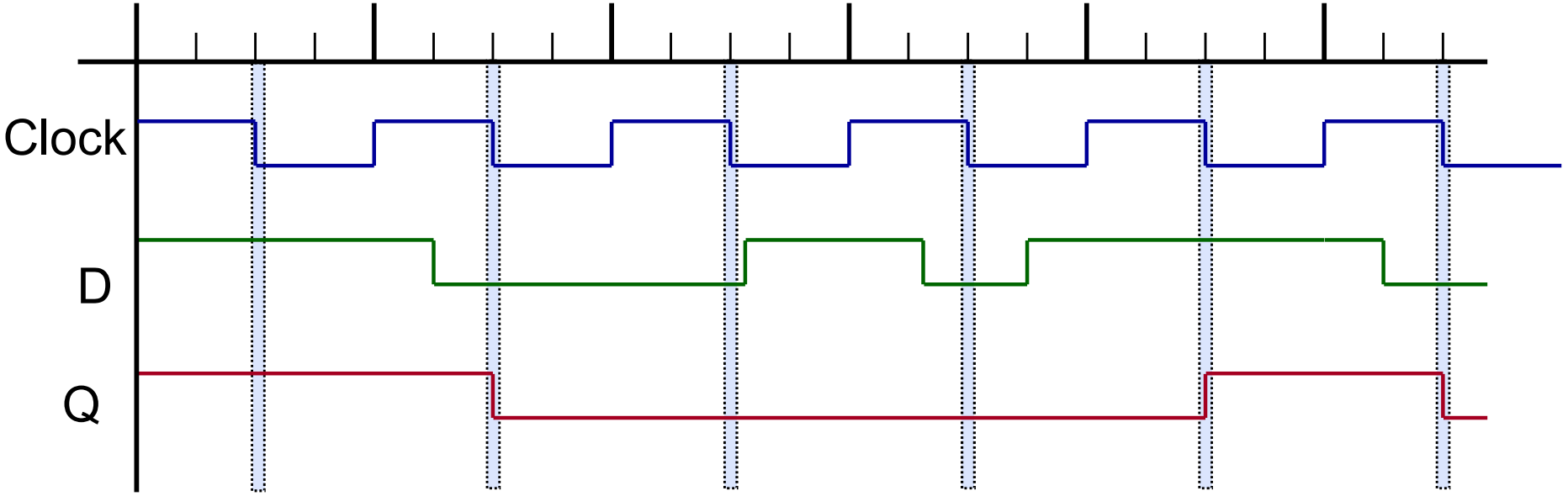


Positive Edge-triggered D Flip-flop: Every rising edge, output is set to the input

Negative Edge-triggering



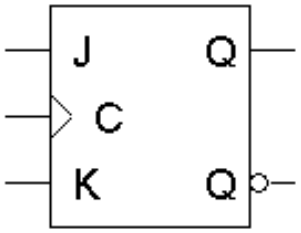
In a negative edge-triggered D Flip-Flop, the output looks at the input only on the falling edge of the clock.



Negative Edge-triggered D Flip-flop: Every falling edge, output is set to the input.

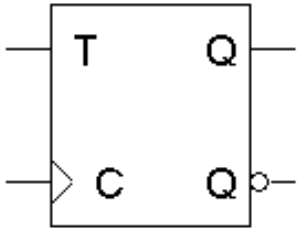
Flip-flop variations

- We can make different versions of flip-flops based on the D flip-flop, just like we made different latches based on the RS latch.
- A **JK flip-flop** has inputs that act like S and R, but the inputs JK=11 are used to *complement* the flip-flop's current state.



C	J	K	Q_{next}
0	x	x	No change
1	0	0	No change
1	0	1	0 (reset)
1	1	0	1 (set)
1	1	1	$Q'_{current}$

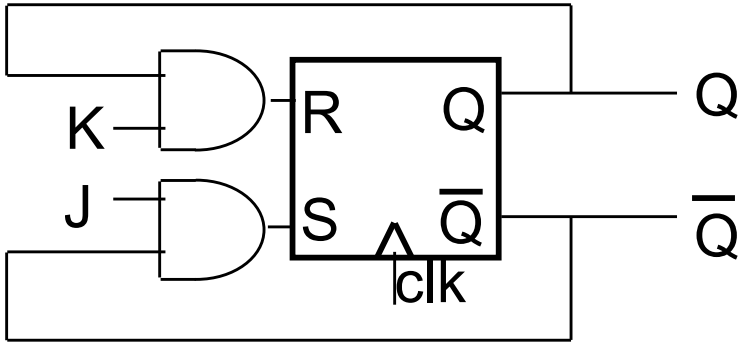
- A **T flip-flop** can only maintain or complement its current state.



C	T	Q_{next}
0	x	No change
1	0	No change
1	1	$Q'_{current}$

J-K Flip-Flops

We want to eliminate the forbidden state of the R-S Latch (when R and S are both 1).
Idea: Q, $\overline{Q}$ are always different. Use them to control the input.



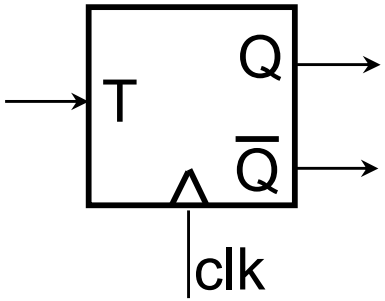
J_t	K_t	Q_t	Q_{t+1}	
0	0	0	0	Hold
0	0	1	1	
0	1	0	0	Reset
0	1	1	0	
1	0	0	1	Set
1	0	1	1	
1	1	0	1	Toggle
1	1	1	0	

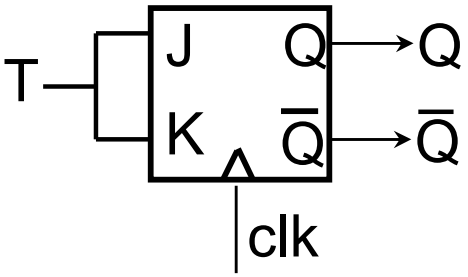
J, K act just like Set and Reset, except: When they're both 1, we get a toggle.

Toggle Flip-Flops

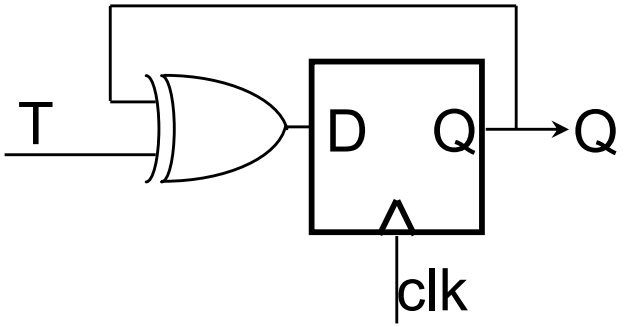
Build a flip-flop that toggles its state on each clock edge when it is enabled (T is the enable).

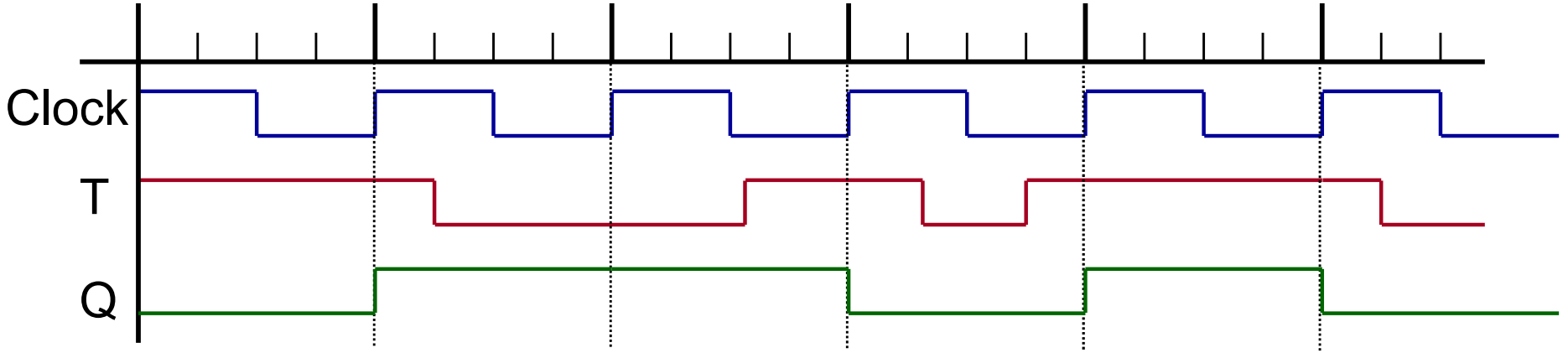
Q^+ is the Q output after the clock changes





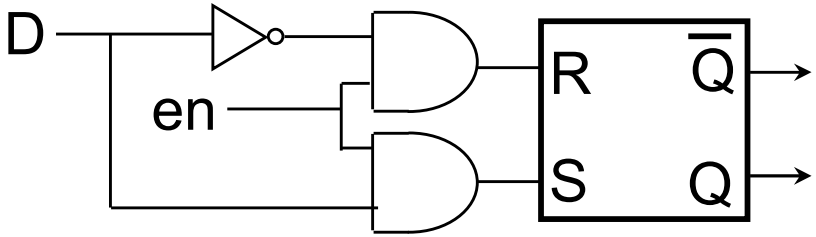
T	Q_t	Q_{t+1}
0	0	0
0	1	1
1	0	1
1	1	0



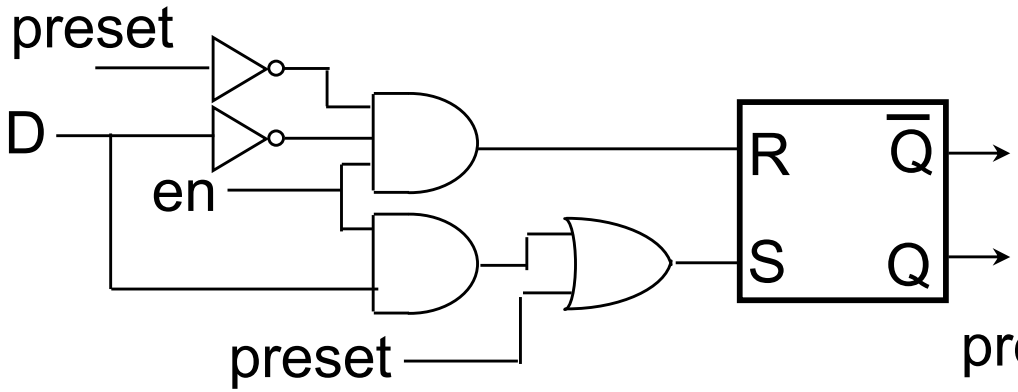


Asynchronous Presets and Clears

Clear forces the output low regardless of the other inputs.
Preset forces the output high regardless of the other inputs.

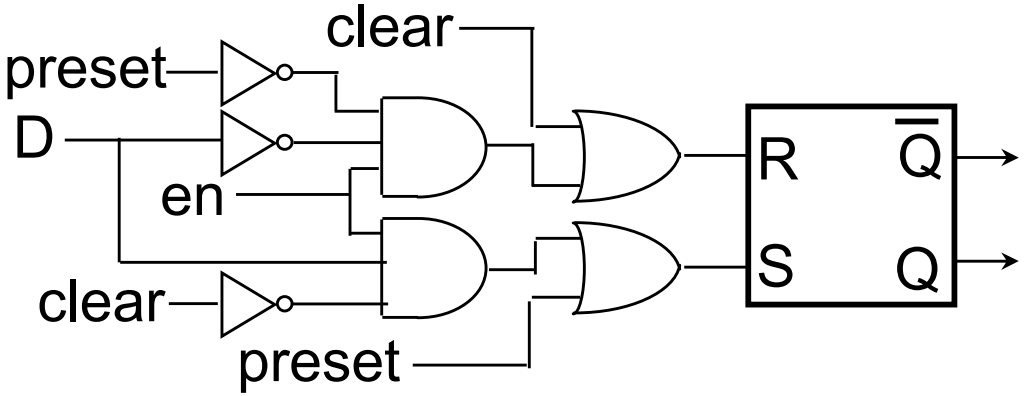


Ordinary level-sensitive D-latch



Preset - Force S=1, R=0

Clear - Force S=0, R=1



Asynchronous - doesn't matter whether clock is high or low

Characteristic/state tables

- The tables that we've made so far are called characteristic or tables.
- A characteristic or state table answers the following question:
  - After the occurrence of a clock pulse, what is the next state **Q(t+1)** of the circuit given the current state **Q(t)** and given the current inputs.
  - For simplicity, the control inputs are not usually listed.

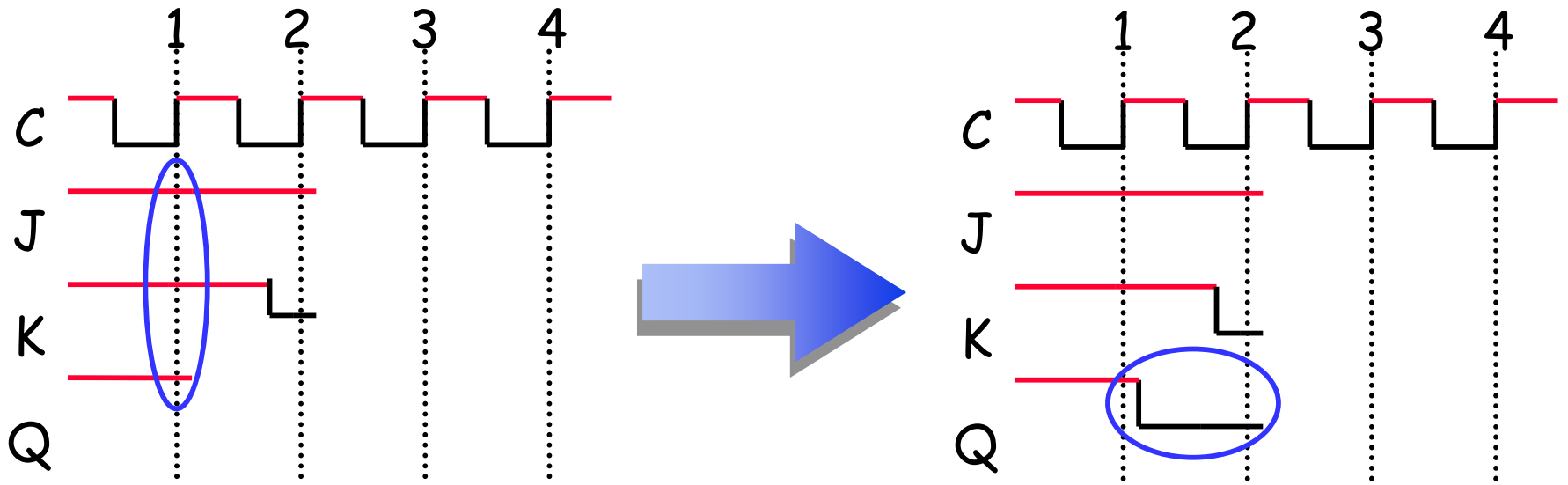
D	Q(t+1)	Operation
0	0	Reset
1	1	Set

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

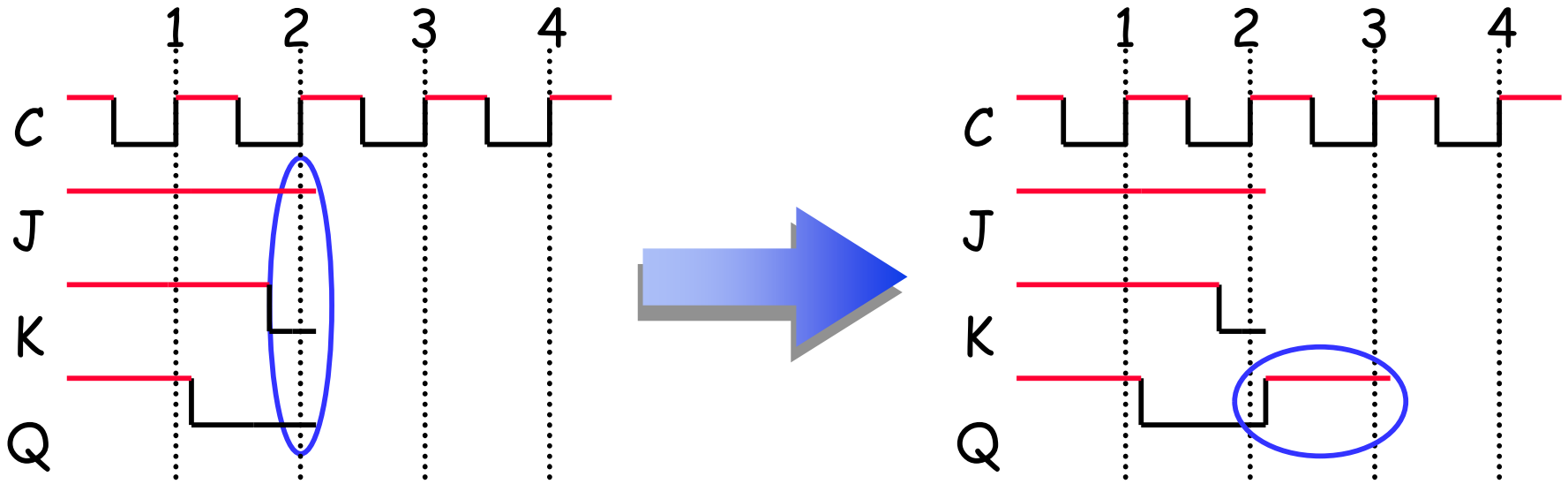
Flip flop timing diagrams

- "Present state" and "next state" are relative terms.
- In the example JK flip-flop timing diagram on the left, you can see that at the first positive clock edge, J=1, K=1 and Q(1) = 1.
- We can use this information to find the "next" state, Q(2) = Q(1)'.
- Q(2) appears right after the first positive clock edge, as shown on the right. It will not change again until after the second clock edge.



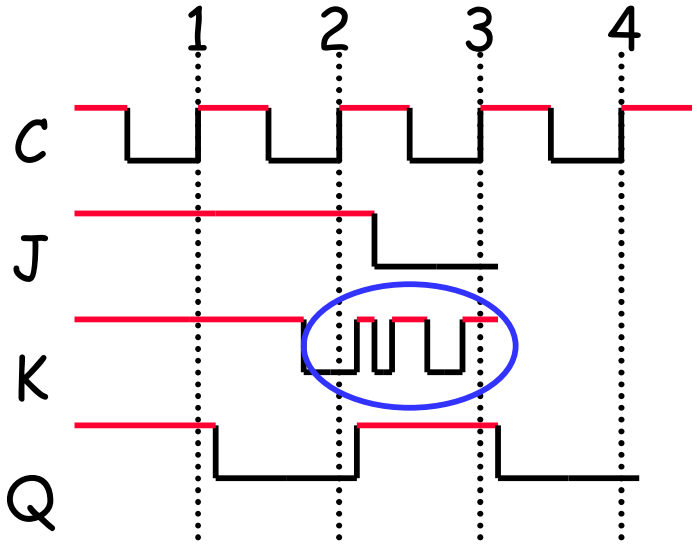
“Present” and “next” are relative

- Similarly, the values of J, K and Q at the second positive clock edge can be used to find the value of Q during the third clock cycle.
- When we do this, Q(2) is now referred to as the “present” state, and Q(3) is now the “next” state.



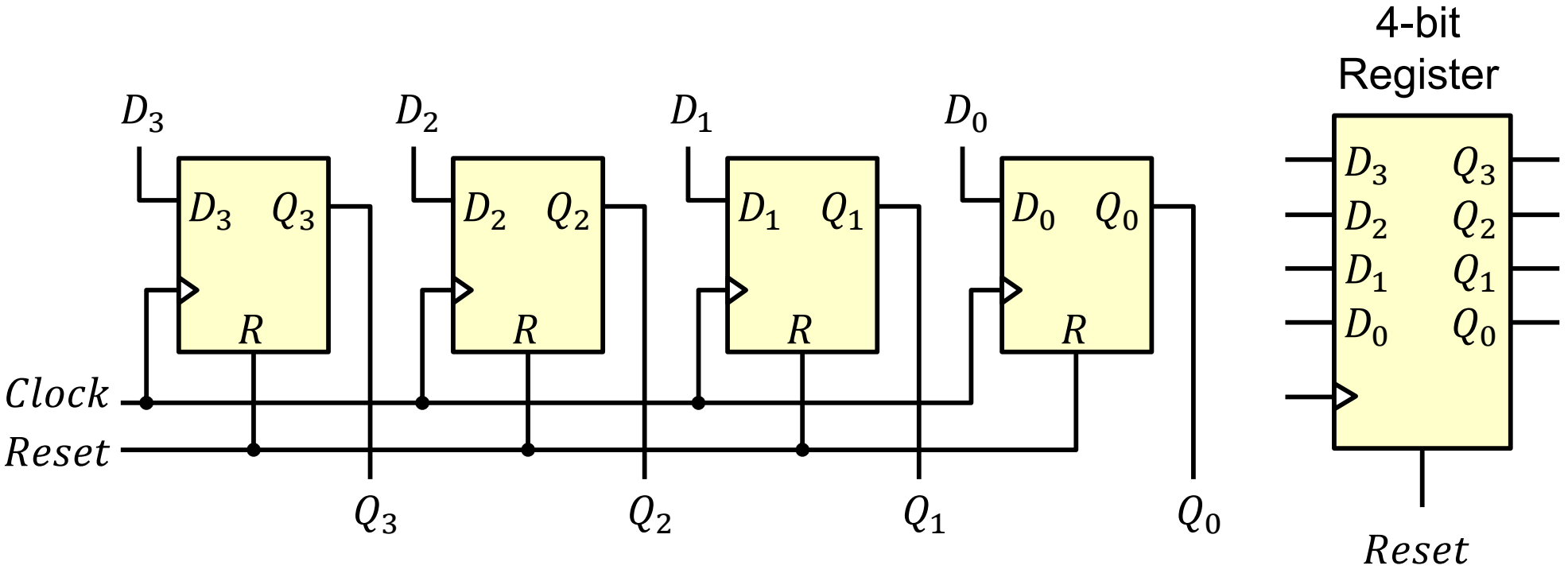
Positive edge triggered

- One final point to repeat: the flip-flop outputs are affected only by the input values *at the positive edge*.
  - In the diagram below, K changes rapidly between the second and third positive edges.
  - But it's only the input values at the third clock edge (K=1, and J=0 and Q=1) that affect the next state, so here Q changes to 0.
- This is a fairly simple timing model. In real life there are "setup times" and "hold times" to worry about as well, to account for internal and external delays.



Registers

- Registers can hold larger quantities of data than individual flip-flops.
- An n-bit register consists of n flip-flops and stores n bits.
- **Common clock:** data is loaded in parallel at the same clock edge.
- **Common reset:** All Flip-Flops are reset in parallel.



Summary

- Two types of Memory elements: latches and flip-flops.
- Latches are level-sensitive, flip-flops are edge-triggered.
- There are several different kinds of flip-flops, but they all serve the same basic purpose of storing bits.
- A flip-flop can be described using a characteristic/state table.
- To use memory in a larger circuit, we need to:
  - Keep the memory disabled until new values are ready to be stored.
  - Enable the memory just long enough for the update to occur.
- A clock signal is used to synchronize circuits and operations.
- The clock period reflects how long combinational operations take.

Acknowledgments

- Credit is acknowledged where credit is due. Please refer to the full list of references.