COMSM1302
Overview of Computer Architecture

Lecture 5

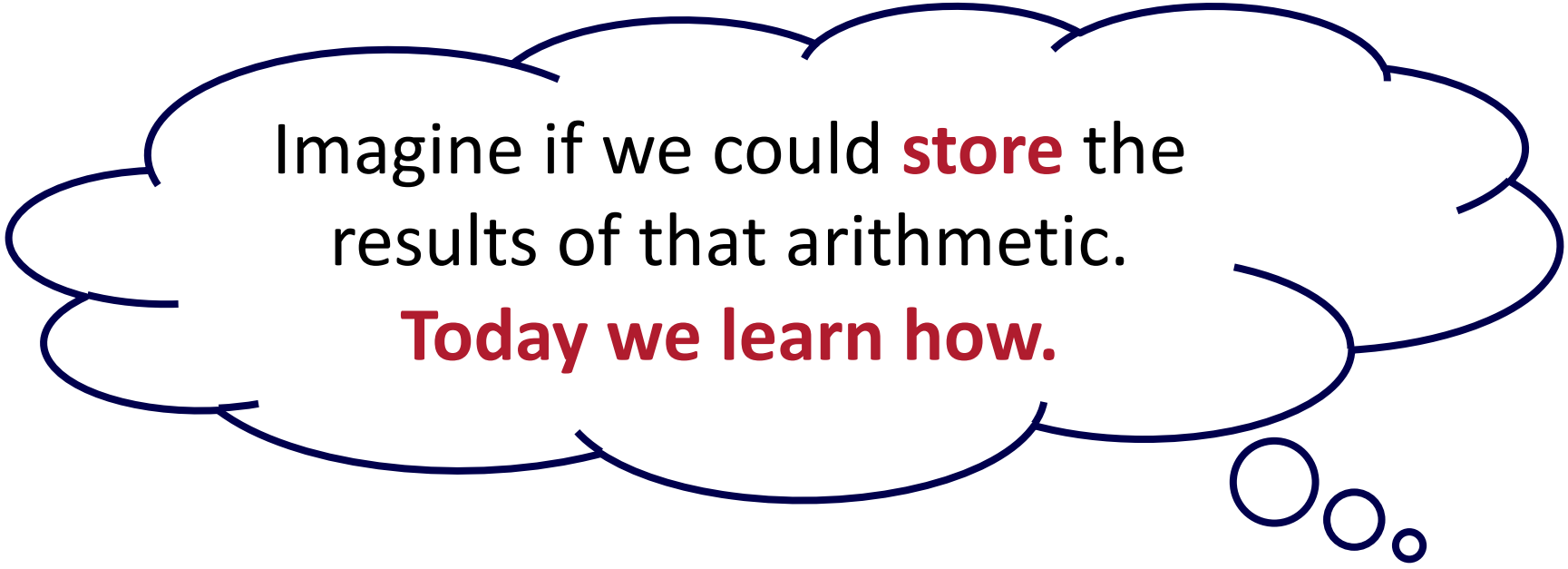
Storage

In this lecture

Foundations	• Data representation, logic, Boolean algebra.
Building blocks	• Transistors, transistor based logic, simple devices, **storage.**
Modules	• Memory, simple controllers, FSMs, processors and execution.
Programming	• Machine code, assembly, high-level languages, compilers.
Wrap-up	• Operating systems, energy aware computing.

Previous lecture

- We can do **basic arithmetic** with a **bunch of NAND gates**!

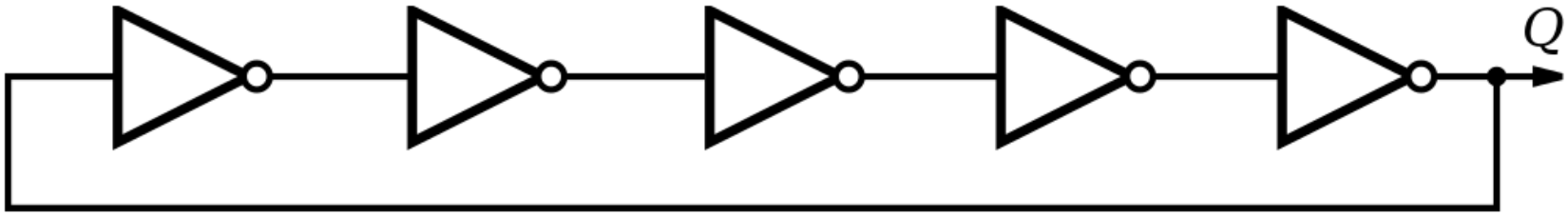


Combinatorial vs. sequential logic

- So far, everything we have done is **combinatorial** logic.
  - Input signals are combined in various ways to produce output signals.
  - By **connecting blocks together** we can build more complex circuits.
- If we **change inputs**, the **outputs change** shortly afterwards.
  - Signals take some (very small) time to propagate.

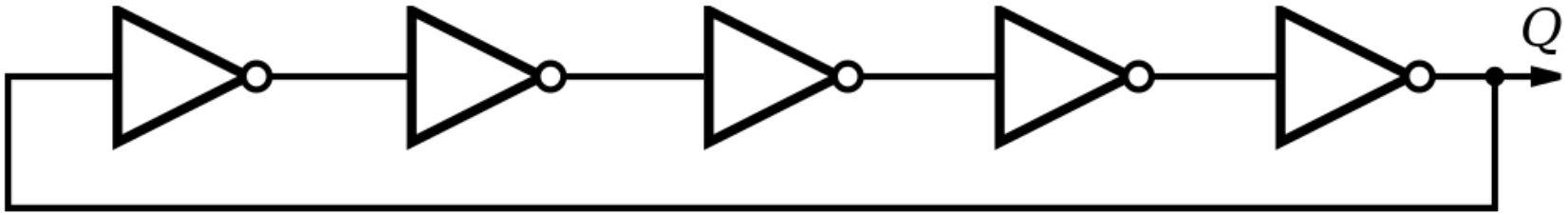
Ring Oscillator

What does the below device do?



Ring Oscillator

What does the below device do?



A **ring oscillator** is a device composed of an **odd number of NOT gates** in a **ring**. The output oscillates between two voltage levels, representing true and false. The NOT gates, or inverters, in a ring oscillator are **connected in a chain** and the output of the last inverter is fed back into the first, forming a circular chain, or a ring.

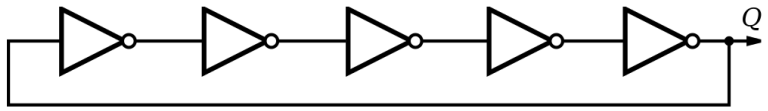
Can a circular chain composed of an even number of inverters be used as a ring oscillator?

Ring Oscillator

- The output oscillates between 1 and 0.
- How fast?

$$F_{osc} = \frac{1}{N \times 2D_{inv}}$$

- N is the number of inverters.



- D_{inv} – the inverter delay, is determined by:
  - Wire length
  - Voltage
  - Material
  - Device size
  - Temperature
  - …

Combinatorial vs. sequential logic

- What if we wanted to perform the following arithmetic:

$$Y = 24 + 15 + 100$$

  - But we can only add two numbers together at a time.
  - We can create a sequence:

$$X = 24 + 15$$

$$Y = X + 100$$

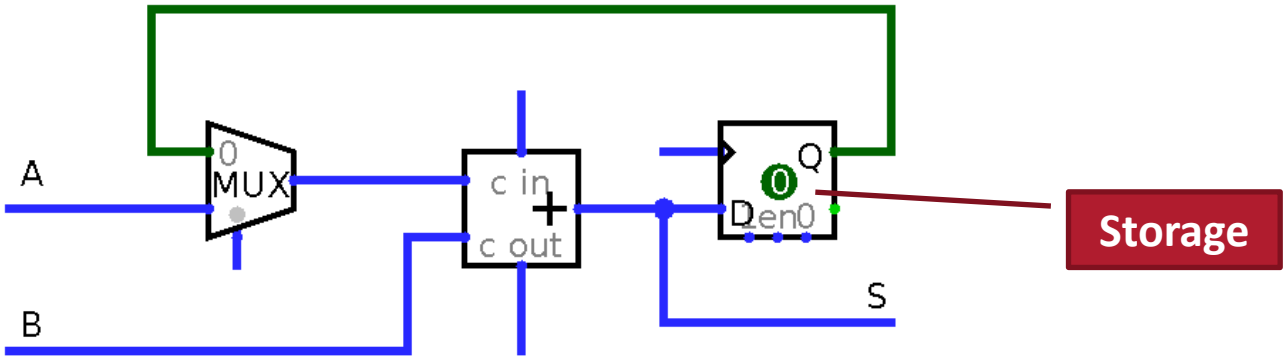
- We want to take the **output** of our **first addition** and make it one of the **inputs** to our **second addition**.

It's not *quite* that simple.

Combinatorial vs. sequential logic

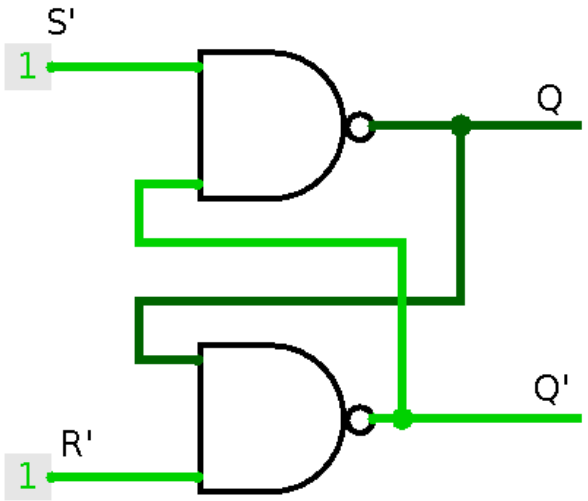
- To enforce a sequence reliably, we can:
  - Store result values
  - Control when values are stored
- This allows us to build a *sequential* system, combining **storage** and **combinatorial logic**.

Below: A *prototype* we'll explain, develop and complete!



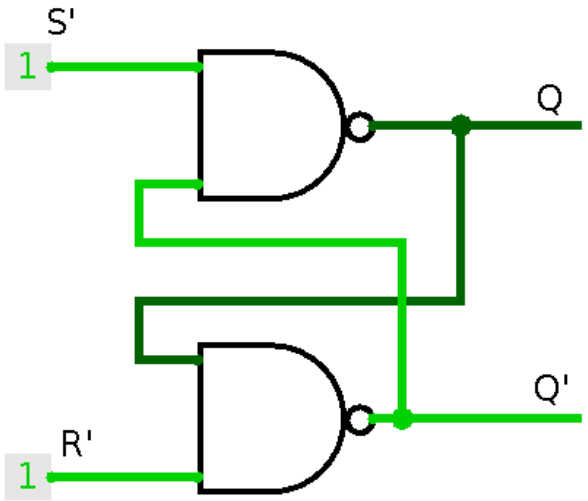
SR-latch

- Two NAND gates
- Input signals are **active-low**.
  - Denoted with top bar or tick, e.g. $\overline{S}$ or S'
  - active high/low tells you what logic value is necessary for you to activate an input
    - **active high** is active when set to 1
    - **active low** is active when set to 0
- Start from S' = R' = 1



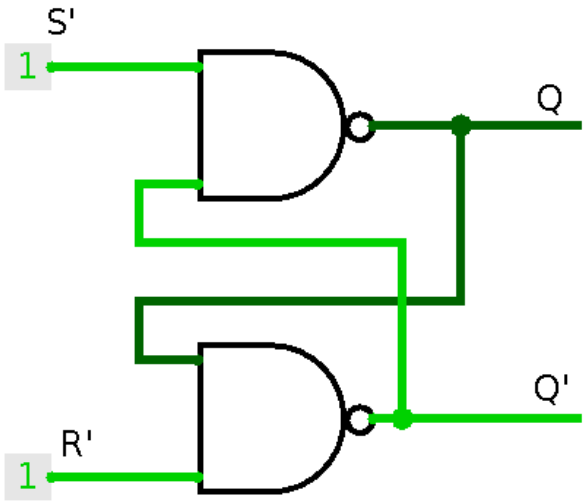
SR-latch

- Two NAND gates
- Input signals are **active-low**.
  - Denoted with top bar or tick, e.g. $\overline{S}$ or S'
  - active high/low tells you what logic value is necessary for you to activate an input
    - **active high** is active when set to 1
    - **active low** is active when set to 0
- Start from S' = R' = 1



SR-latch

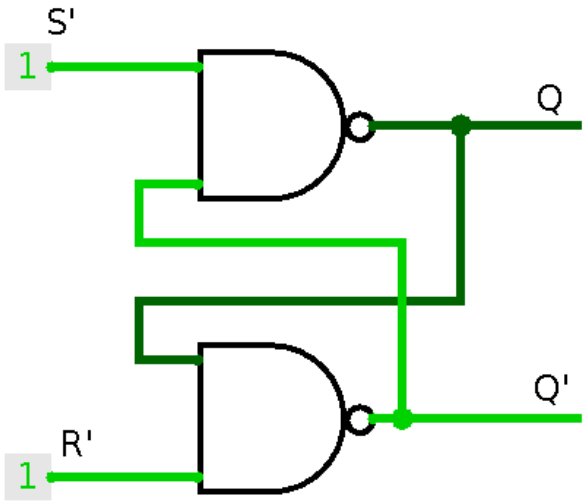
- Two NAND gates
- Input signals are **active-low**.
  - Denoted with top bar or tick, e.g. $\overline{S}$ or S'
  - active high/low tells you what logic value is necessary for you to activate an input
    - **active high** is active when set to 1
    - **active low** is active when set to 0
- Start from S' = R' = 1
- **S**et the latch
  - S' = 0



A	B	A NAND B
1	1	0
1	0	1
0	1	1
0	0	1

SR-latch

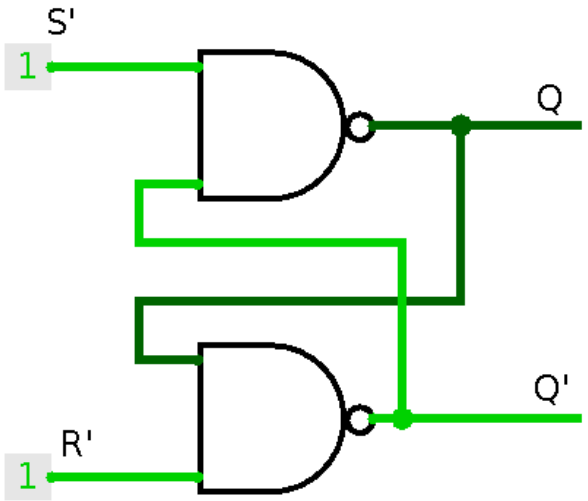
- Two NAND gates
- Input signals are **active-low**.
  - Denoted with top bar or tick, e.g. $\overline{S}$ or S'
  - active high/low tells you what logic value is necessary for you to activate an input
    - **active high** is active when set to 1
    - **active low** is active when set to 0
- Start from S' = R' = 1
- **S**et the latch
  - S' = 0



A	B	A NAND B
1	1	0
1	0	1
0	1	1
0	0	1

SR-latch

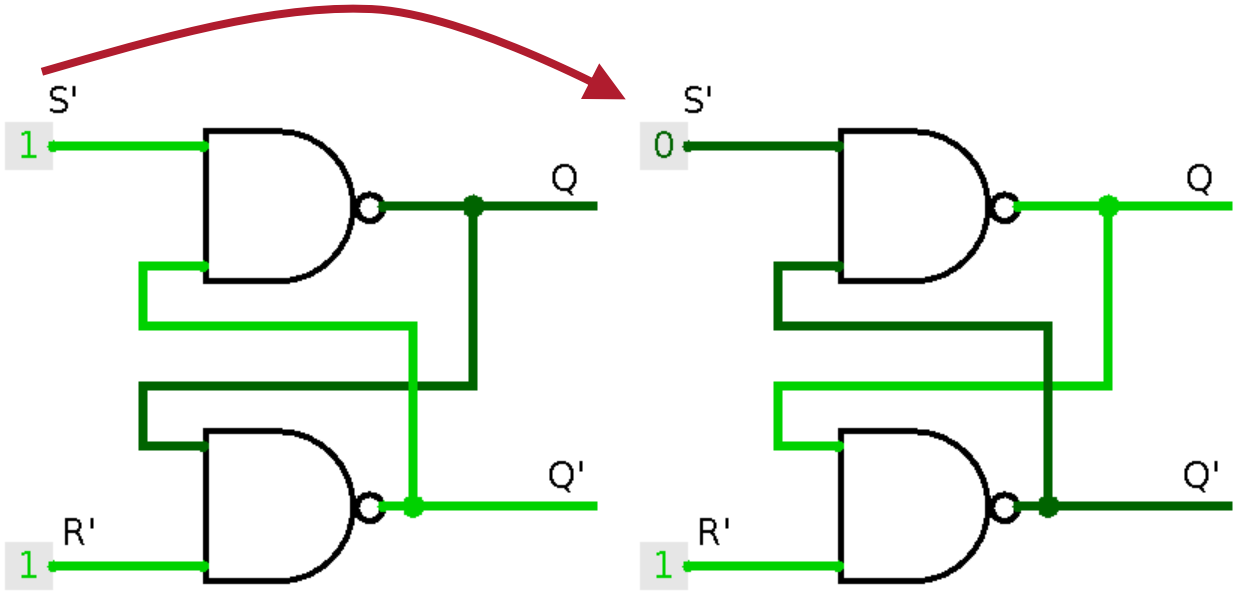
- Two NAND gates
- Input signals are **active-low**.
  - Denoted with top bar or tick, e.g. $\overline{S}$ or S'
  - active high/low tells you what logic value is necessary for you to activate an input
    - **active high** is active when set to 1
    - **active low** is active when set to 0
- Start from S' = R' = 1
- **S**et the latch
  - S' = 0
- **R**eset the latch
  - R' = 0



A	B	A NAND B
1	1	0
1	0	1
0	1	1
0	0	1

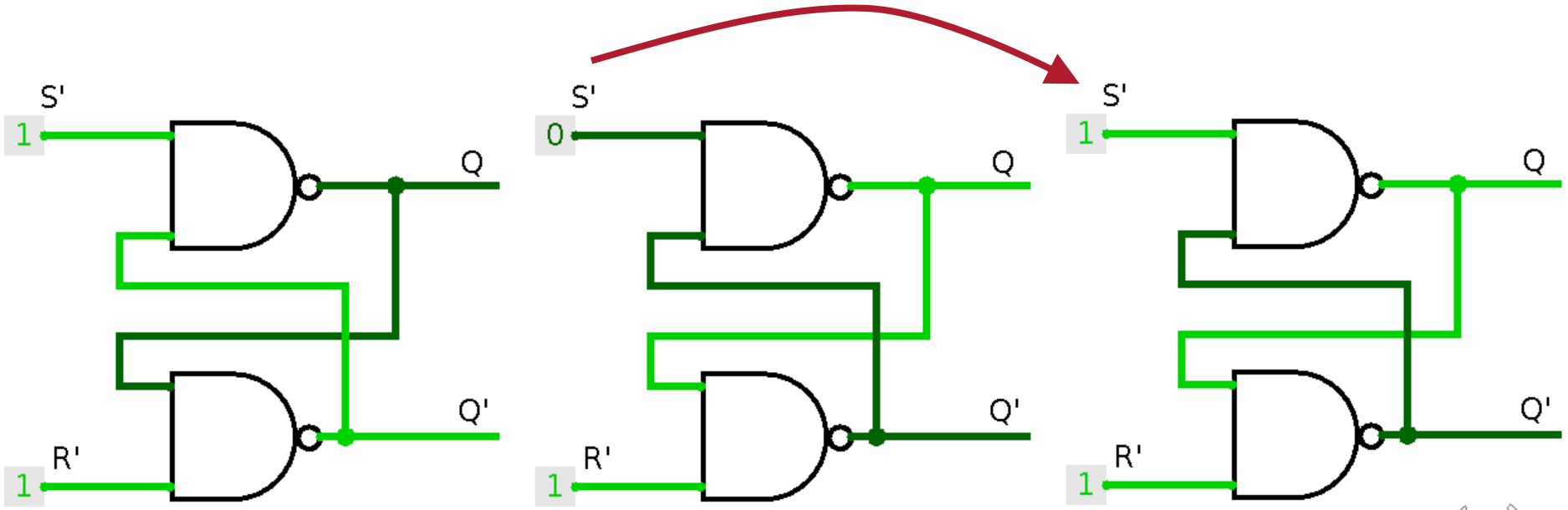
Setting the latch

- When S' transitions to low, Q is **set high** (and Q' its inverse).



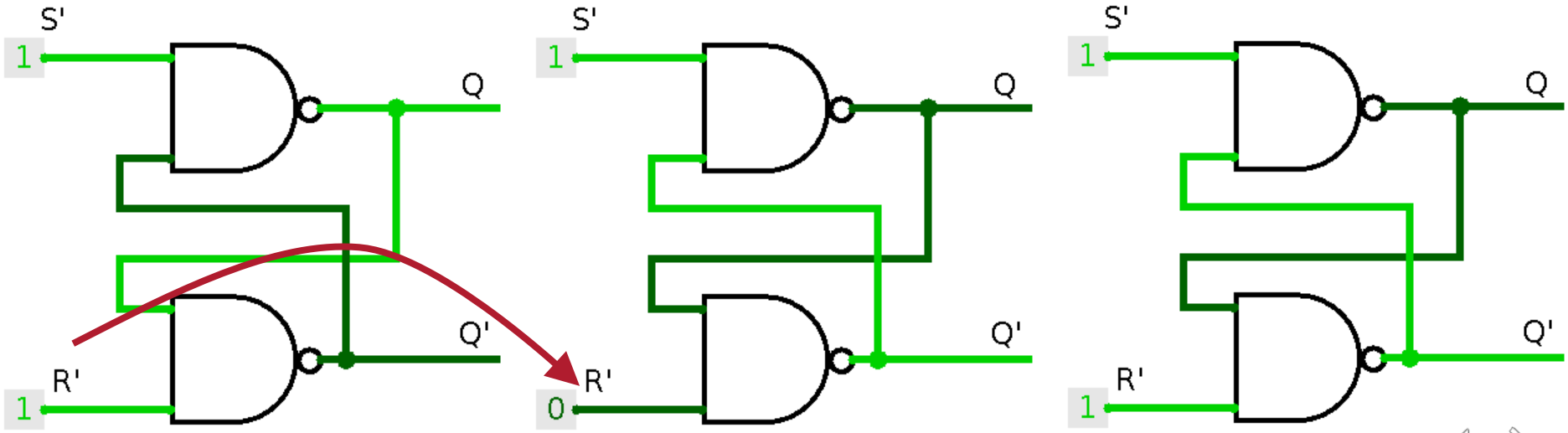
Setting the latch

- When S' transitions to low, Q is **set high** (and Q' its inverse).
- Upon S' returning to high… Q retains the same high value.



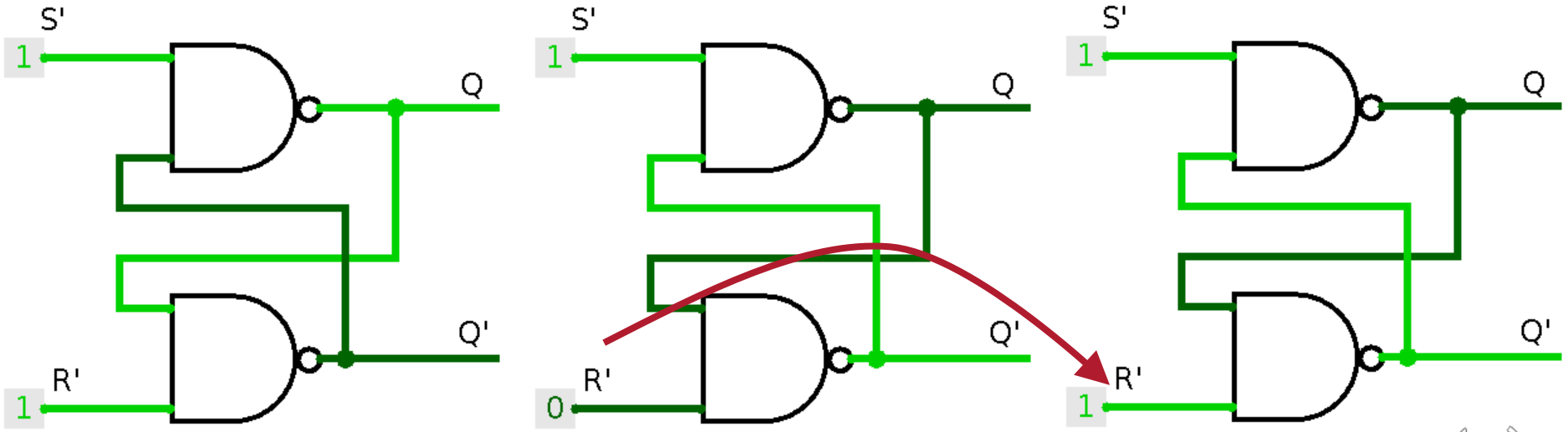
Resetting the latch

- When R' transitions to low, Q is **reset low**.



Resetting the latch

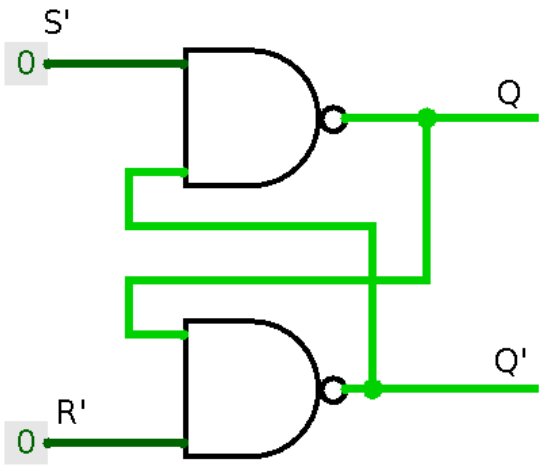
- When R' transitions to low, Q is **reset low**.
- Upon R' returning to high… Q retains the same low value.



SR inputs

A	B	A NAND B
1	1	0
1	0	1
0	1	1
0	0	1

S’	R’	Q
1	1	Hold
1	0	0
0	1	1
0	0	

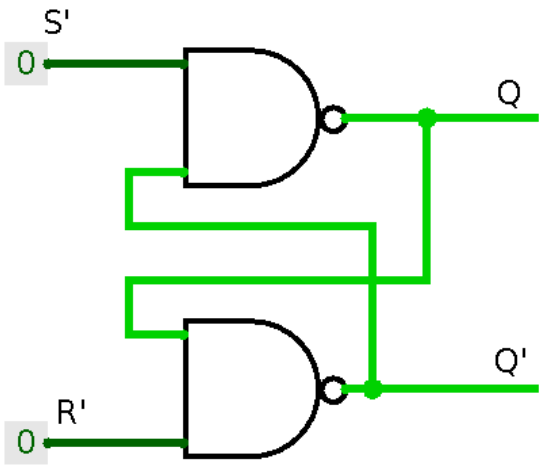


- There are **three valid** combinations of S’ and R’ for the SR NAND latch.

SR inputs

A	B	A NAND B
1	1	0
1	0	1
0	1	1
0	0	1

S’	R’	Q
1	1	Hold
1	0	0
0	1	1
0	0	Not allowed

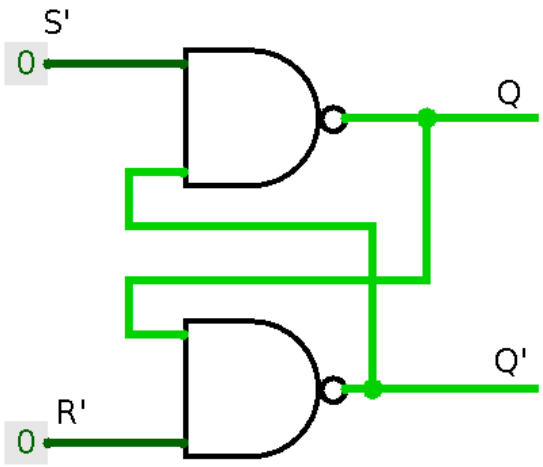


- There are **three valid** combinations of S’ and R’ for the SR NAND latch.
- Both S’ and R’ should **not be active** (low) together.

SR inputs

A	B	A NAND B
1	1	0
1	0	1
0	1	1
0	0	1

S'	R'	Q
1	1	Hold
1	0	0
0	1	1
0	0	Not allowed



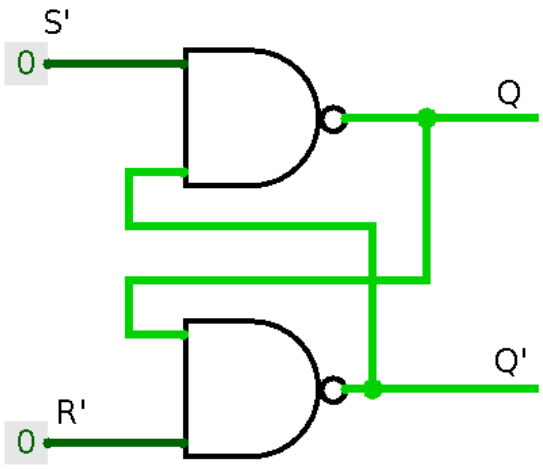
- There are **three valid** combinations of S' and R' for the SR NAND latch.
- Both S' and R' should **not be active** (low) together.

Why?

SR inputs

A	B	A NAND B
1	1	0
1	0	1
0	1	1
0	0	1

S’	R’	Q	Q’
1	1	Q_prev	Q’_prev
1	0	0	1
0	1	1	0
0	0		

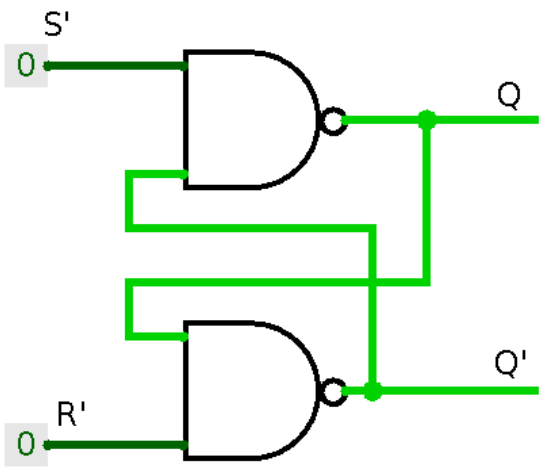


- If both S’ and R’ were active at the same time, then the output ...

SR inputs

A	B	A NAND B
1	1	0
1	0	1
0	1	1
0	0	1

S'	R'	Q	Q'
1	1	Q_prev	Q'_prev
1	0	0	1
0	1	1	0
0	0		



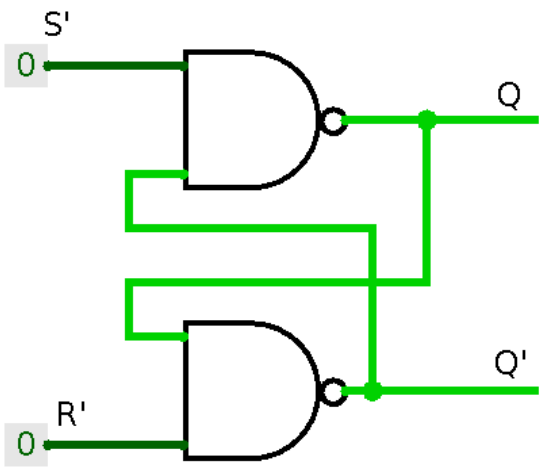
- If both S' and R' were active at the same time, then the output **would not make any sense**.

  S' = 0, R' = 0, **Q = Q'**

SR inputs

A	B	A NAND B
1	1	0
1	0	1
0	1	1
0	0	1

S'	R'	Q	Q'
1	1	Q_prev	Q'_prev
1	0	0	1
0	1	1	0
0	**0**	**1**	**1**



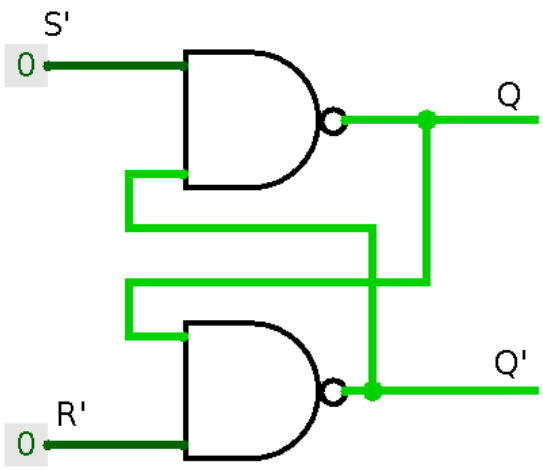
- If both S' and R' were active at the same time, then the output **would not make any sense**.

  S' = 0, R' = 0, **Q = Q'**

SR inputs

A	B	A NAND B
1	1	0
1	0	1
0	1	1
0	0	1

S'	R'	Q	Q'
1	1	Q_prev	Q'_prev
1	0	0	1
0	1	1	0
0	0	not allowed	

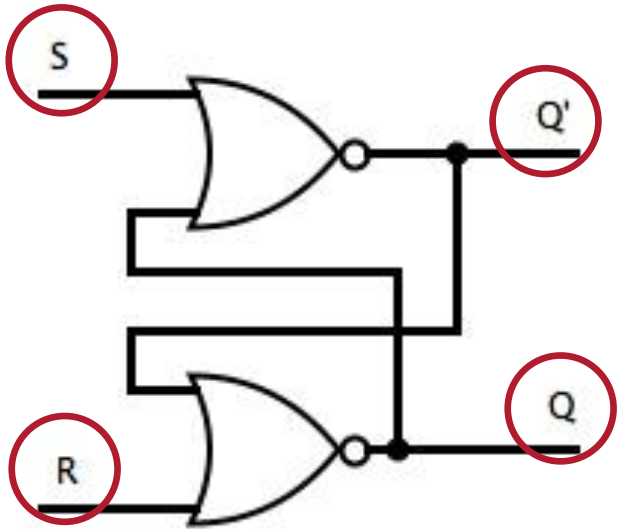


- If both S' and R' were active at the same time, then the output **would not make any sense**.

  S' = 0, R' = 0, **Q = Q'**

Different latch types

- We have seen an SR (Set-Reset) latch using NAND.
  - An SR latch can also be built from NOR gates.

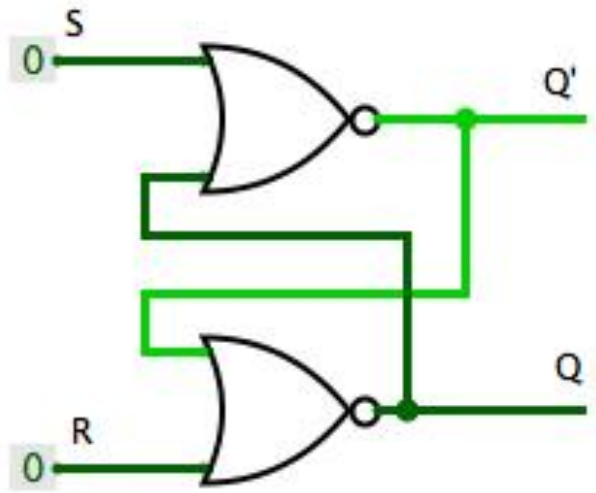


How does this work?

SR latch with NOR gates

A	B	A NOR B
1	1	0
1	0	0
0	1	0
0	0	1

- We have seen an SR (Set-Reset) latch using NAND.
  - An SR latch can also be built from NOR gates.



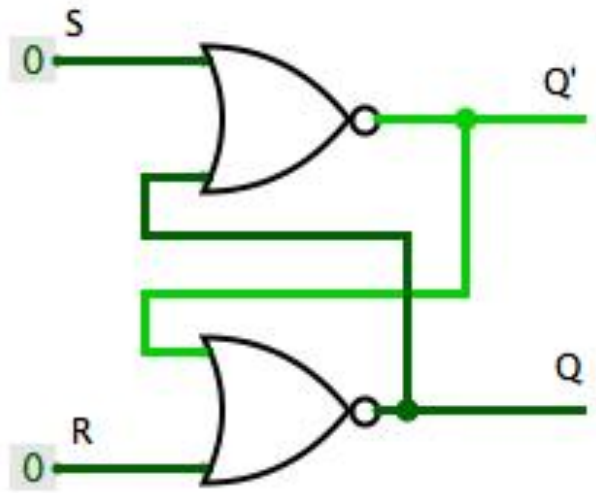
How does this work?

S	R	Q	Q_next	Q'_next
0	0	0		

SR latch with NOR gates

A	B	A NOR B
1	1	0
1	0	0
0	1	0
0	0	1

- We have seen an SR (Set-Reset) latch using NAND.
  - An SR latch can also be built from NOR gates.



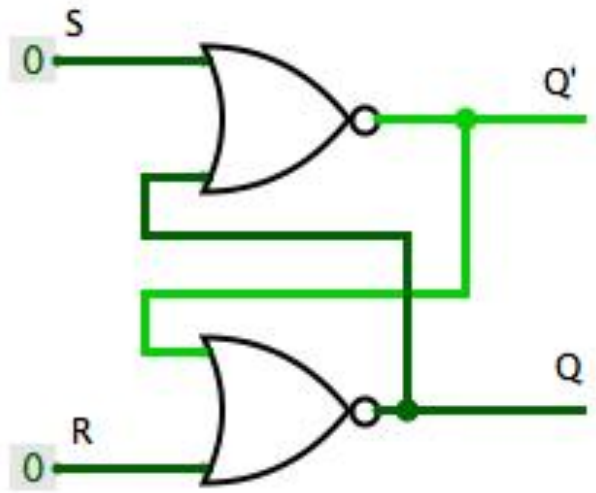
How does this work?

S	R	Q	Q_next	Q'_next
0	0	0	0	1

SR latch with NOR gates

A	B	A NOR B
1	1	0
1	0	0
0	1	0
0	0	1

- We have seen an SR (Set-Reset) latch using NAND.
  - An SR latch can also be built from NOR gates.



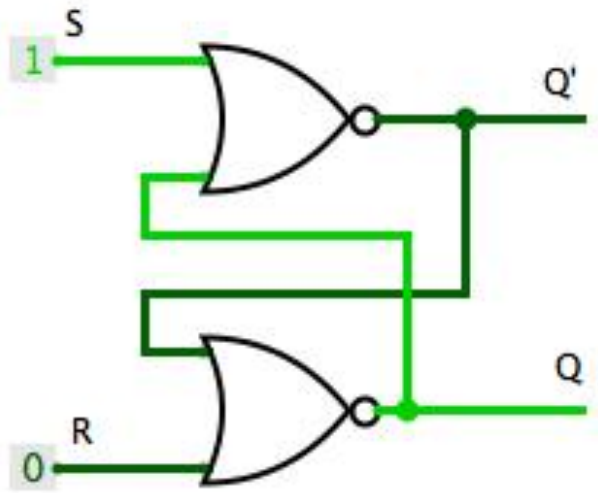
How does this work?

S	R	Q	Q_next	Q'_next
0	0	0	**0**	1
1	0	**0**		

SR latch with NOR gates

A	B	A NOR B
1	1	0
1	0	0
0	1	0
0	0	1

- We have seen an SR (Set-Reset) latch using NAND.
  - An SR latch can also be built from NOR gates.



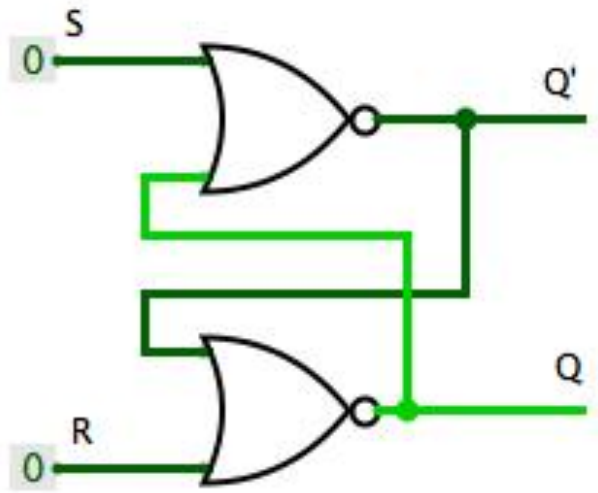
How does this work?

S	R	Q	Q_next	Q'_next
0	0	0	0	1
1	0	0	1	0

SR latch with NOR gates

A	B	A NOR B
1	1	0
1	0	0
0	1	0
0	0	1

- We have seen an SR (Set-Reset) latch using NAND.
  - An SR latch can also be built from NOR gates.



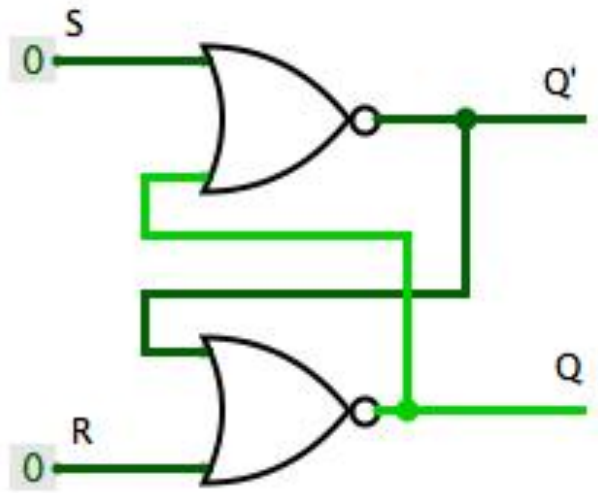
How does this work?

S	R	Q	Q_next	Q'_next
0	0	0	0	1
1	0	0	**1**	0
0	0	**1**		

SR latch with NOR gates

A	B	A NOR B
1	1	0
1	0	0
0	1	0
0	0	1

- We have seen an SR (Set-Reset) latch using NAND.
  - An SR latch can also be built from NOR gates.



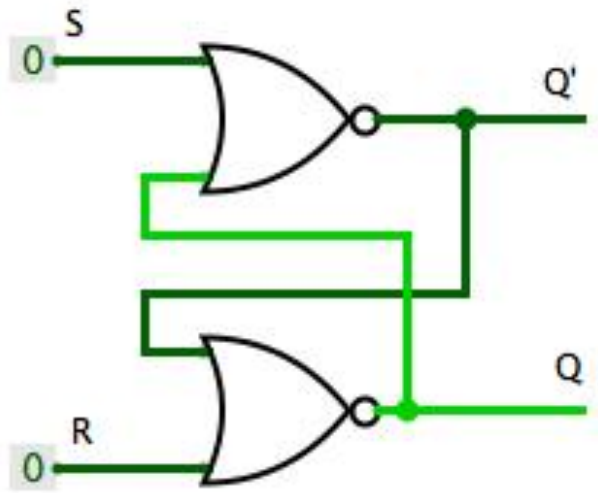
How does this work?

S	R	Q	Q_next	Q'_next
0	0	0	0	1
1	0	0	1	0
0	0	1	1	0

SR latch with NOR gates

A	B	A NOR B
1	1	0
1	0	0
0	1	0
0	0	1

- We have seen an SR (Set-Reset) latch using NAND.
  - An SR latch can also be built from NOR gates.



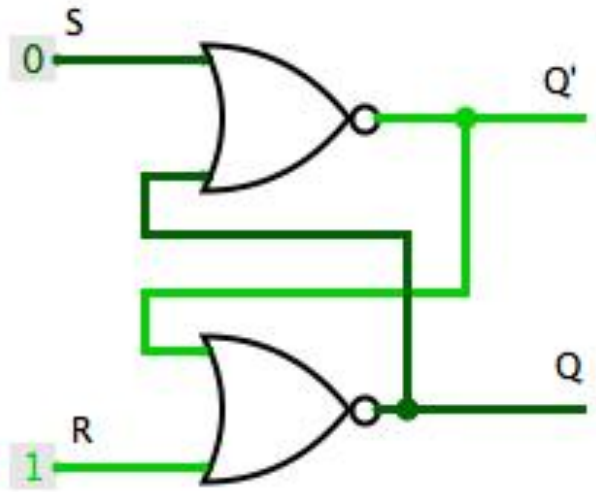
How does this work?

S	R	Q	Q_next	Q'_next
0	0	0	0	1
1	0	0	1	0
0	0	1	**1**	0
0	1	**1**		

SR latch with NOR gates

A	B	A NOR B
1	1	0
1	0	0
0	1	0
0	0	1

- We have seen an SR (Set-Reset) latch using NAND.
  - An SR latch can also be built from NOR gates.



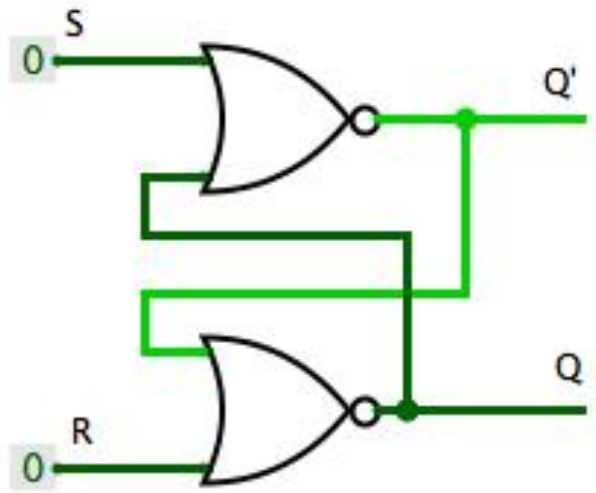
How does this work?

S	R	Q	Q_next	Q'_next
0	0	0	0	1
1	0	0	1	0
0	0	1	1	0
0	1	1	0	1

SR latch with NOR gates

A	B	A NOR B
1	1	0
1	0	0
0	1	0
0	0	1

- We have seen an SR (Set-Reset) latch using NAND.
  - An SR latch can also be built from NOR gates.



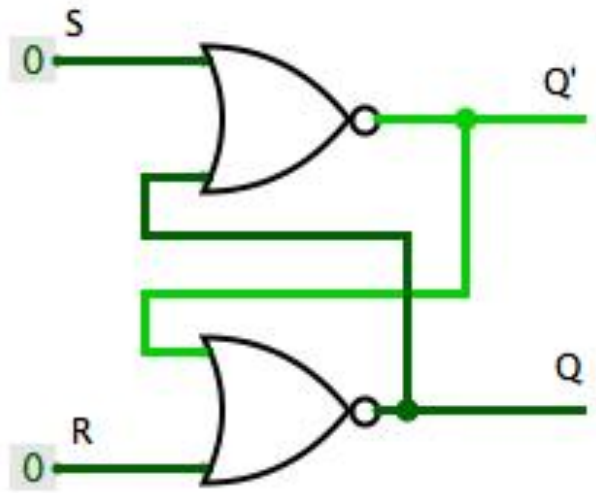
How does this work?

S	R	Q	Q_next	Q'_next
0	0	0	0	1
1	0	0	1	0
0	0	1	1	0
0	1	1	**0**	1
0	0	**0**		

SR latch with NOR gates

A	B	A NOR B
1	1	0
1	0	0
0	1	0
0	0	1

- We have seen an SR (Set-Reset) latch using NAND.
  - An SR latch can also be built from NOR gates.



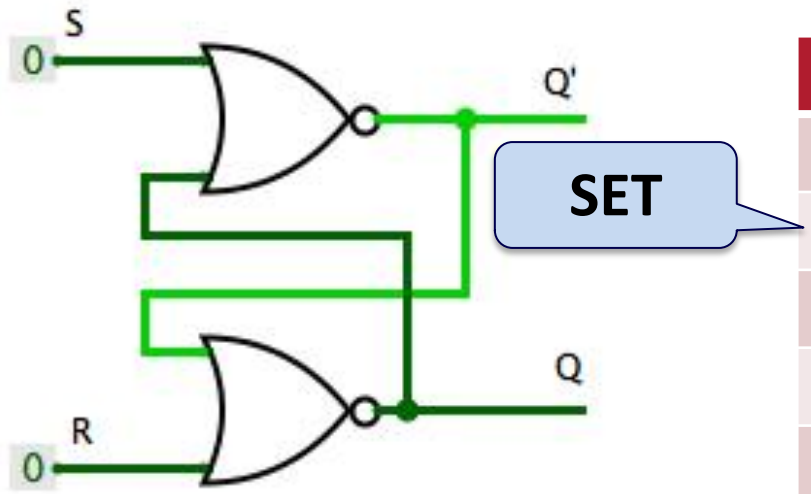
How does this work?

S	R	Q	Q_next	Q'_next
0	0	0	0	1
1	0	0	1	0
0	0	1	1	0
0	1	1	0	1
0	0	0	0	1

SR latch with NOR gates

A	B	A NOR B
1	1	0
1	0	0
0	1	0
0	0	1

- We have seen an SR (Set-Reset) latch using NAND.
  - An SR latch can also be built from NOR gates.



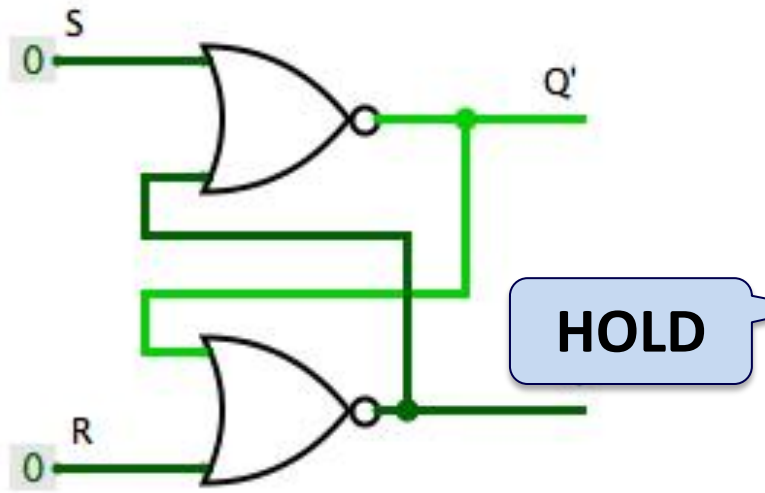
How does this work?

S	R	Q	Q_next	Q'_next
0	0	0	0	1
1	**0**	0	**1**	**0**
0	0	1	1	0
0	1	1	0	1
0	0	0	0	1

SR latch with NOR gates

A	B	A NOR B
1	1	0
1	0	0
0	1	0
0	0	1

- We have seen an SR (Set-Reset) latch using NAND.
  - An SR latch can also be built from NOR gates.



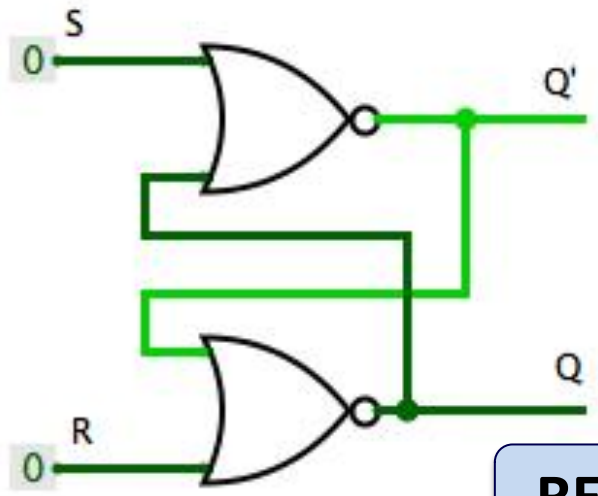
How does this work?

S	R	Q	Q_next	Q'_next
0	0	0	0	1
1	0	0	1	0
0	**0**	**1**	**1**	**0**
0	1	1	0	1
0	0	0	0	1

SR latch with NOR gates

A	B	A NOR B
1	1	0
1	0	0
0	1	0
0	0	1

- We have seen an SR (Set-Reset) latch using NAND.
  - An SR latch can also be built from NOR gates.



How does this work?

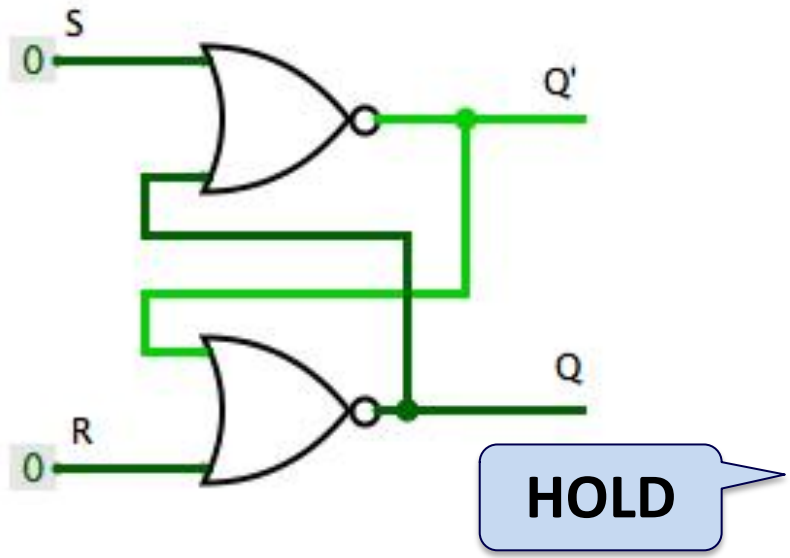
S	R	Q	Q_next	Q'_next
0	0	0	0	1
1	0	0	1	0
0	0	1	1	0
0	**1**	1	**0**	**1**
0	0	0	0	1

RESET

SR latch with NOR gates

A	B	A NOR B
1	1	0
1	0	0
0	1	0
0	0	1

- We have seen an SR (Set-Reset) latch using NAND.
  - An SR latch can also be built from NOR gates.



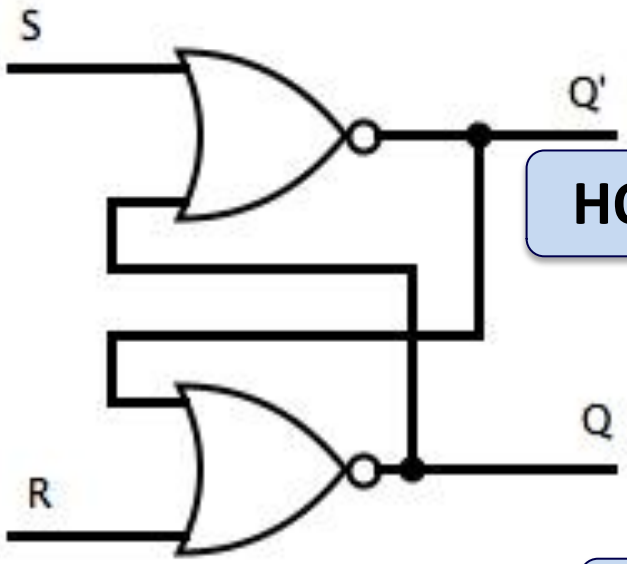
How does this work?

S	R	Q	Q_next	Q'_next
0	0	0	0	1
1	0	0	1	0
0	0	1	1	0
0	1	1	0	1
0	**0**	**0**	**0**	**1**

SR latch with NOR gates

A	B	A NOR B
1	1	0
1	0	0
0	1	0
0	0	1

- We have seen an SR (Set-Reset) latch using NAND.
  - An SR latch can also be built from NOR gates.



How does this work?

S	R	Q	Q'
0	0	Q_prev	Q'_prev
0	1	0	1
1	0	1	0
1	1		

HOLD

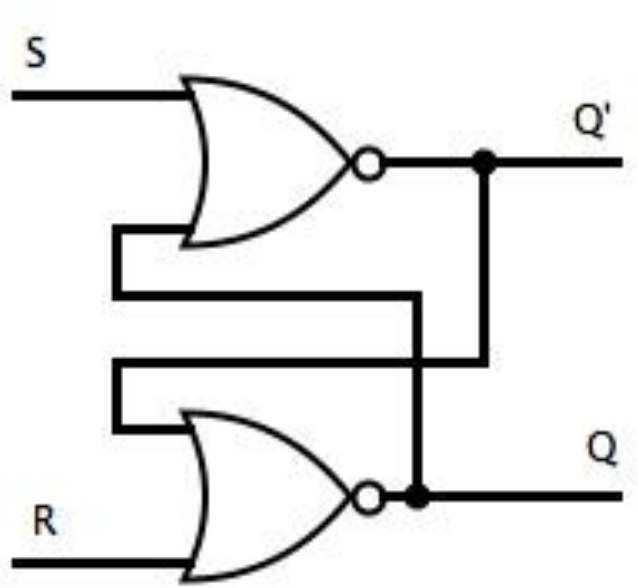
SET

RESET

SR latch with NOR gates

A	B	A NOR B
1	1	0
1	0	0
0	1	0
0	0	1

- We have seen an SR (Set-Reset) latch using NAND.
  - An SR latch can also be built from NOR gates.



How does this work?

S	R	Q	Q'
0	0	Q_prev	Q'_prev
0	1	0	1
1	0	1	0
1	1	not allowed	

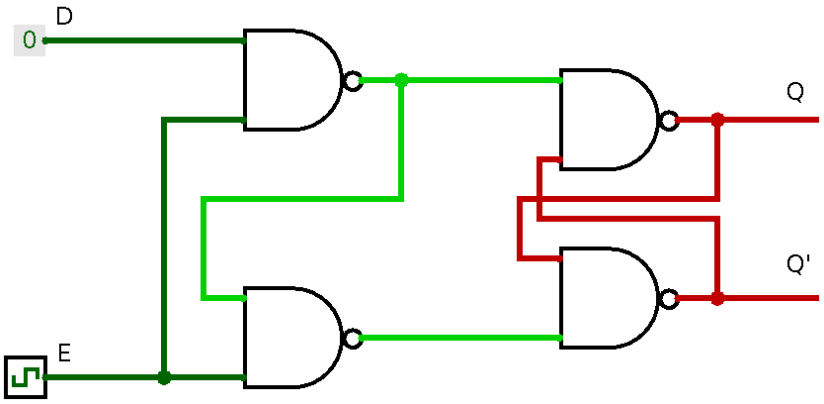
Different latch types

- SR (Set-Reset) latch
  - using NAND gates
  - using NOR gates
- Other latches:
  - **D latch**
  - JK latch

D-latch

- Instead of separate set/reset inputs, one input, **D**, specifies the **data** value, and a second input, **E**, **enable**s propagation.
- If E is low, the previous output values are retained.

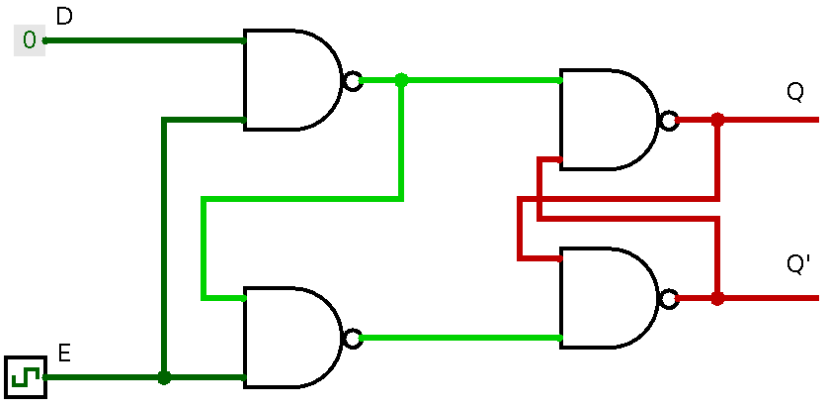
D	E	Q	Q’
0	0	Q_prev	Q’_prev
0	1	0	1
1	0	Q_prev	Q’_prev
1	1	1	0



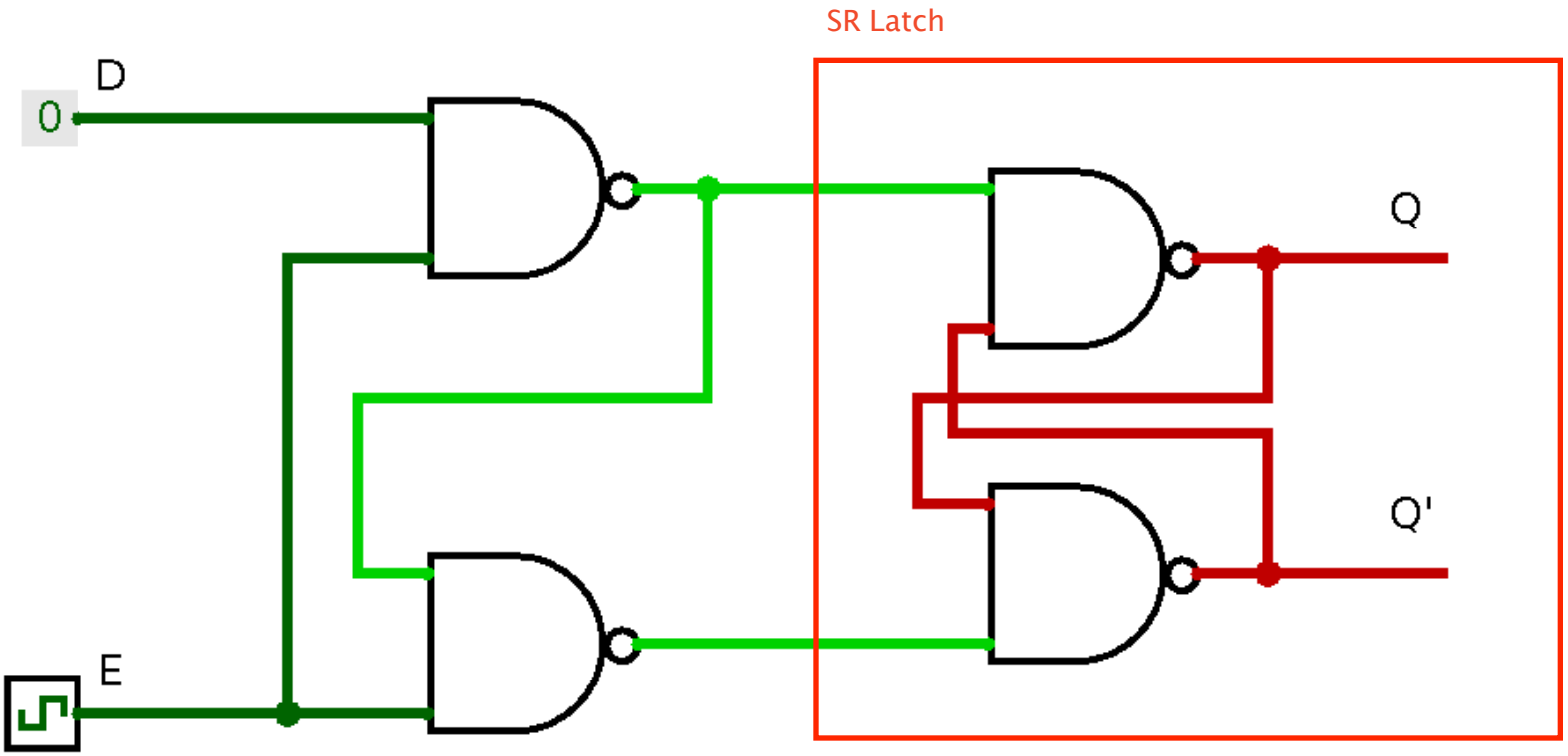
D-latch

- Instead of separate set/reset inputs, one input, **D**, specifies the **data** value, and a second input, **E**, **enable**s propagation.
- If E is low, the previous output values are retained.
- No forbidden inputs.
  - Why?

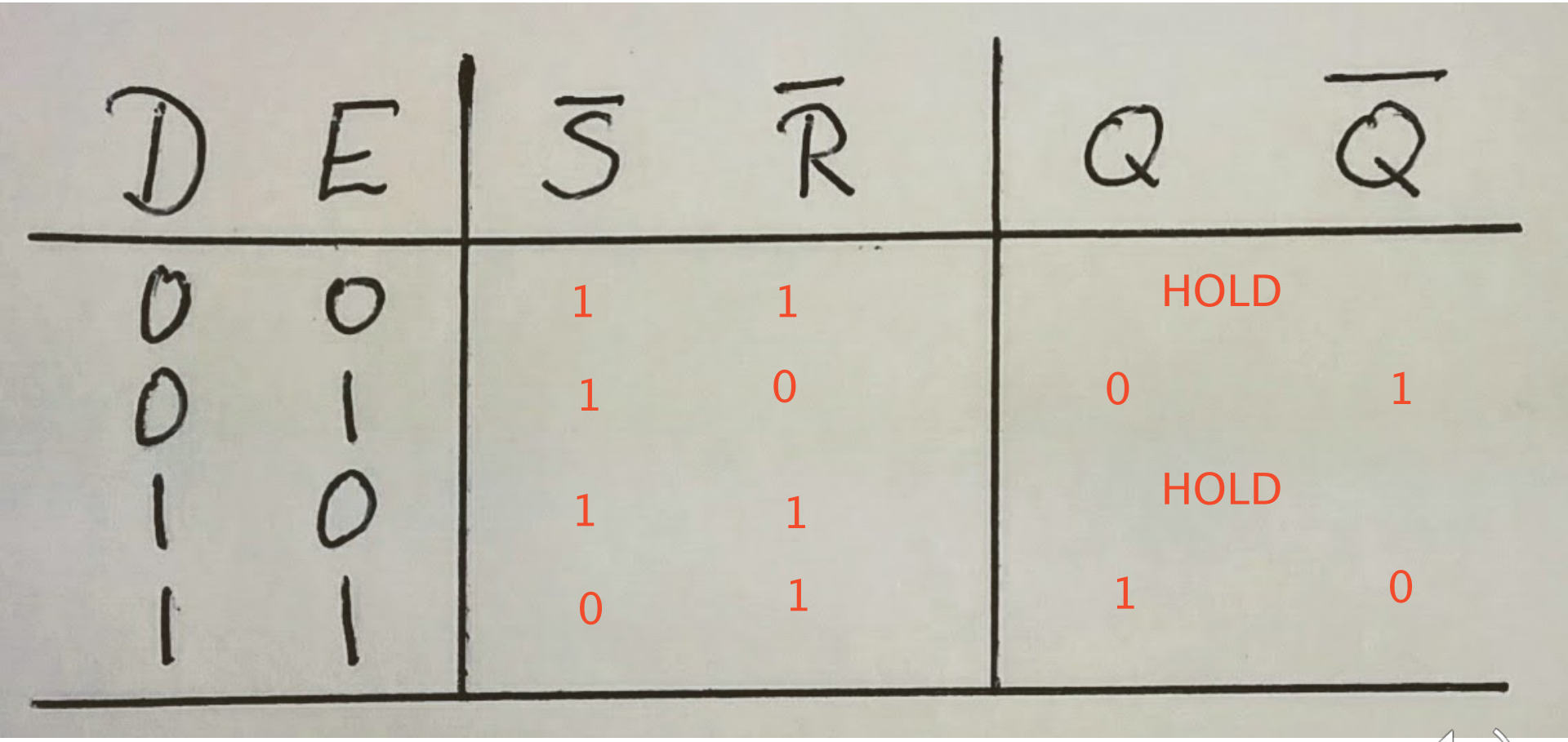
D	E	Q	Q’
0	0	Q_prev	Q’_prev
0	1	0	1
1	0	Q_prev	Q’_prev
1	1	1	0



D-latch internal signals



D-latch truth table - I



D	E	$\overline{S}$	$\overline{R}$	Q	$\overline{Q}$
0	0	1	1	HOLD	
0	1	1	0	0	1
1	0	1	1	HOLD	
1	1	0	1	1	0

D-latch truth table - II

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

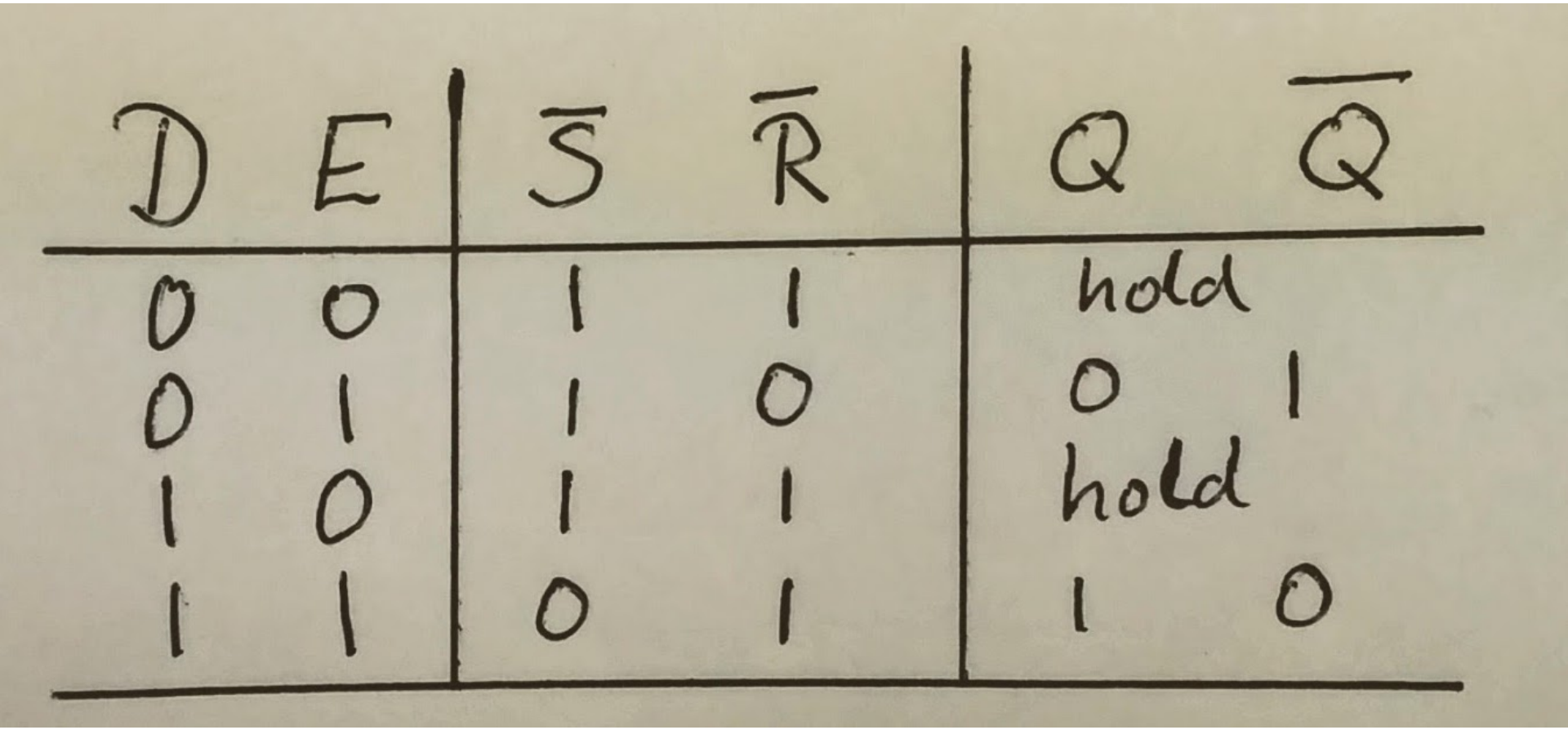
D-latch truth table - II

S'	R'	Q	Q'
1	1	hold	
1	0	0	1
0	1	1	0
0	0	Not allowed	

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

D-latch truth table - III

S'	R'	Q	Q'
1	1	hold	
1	0	0	1
0	1	1	0
0	0	Not allowed	

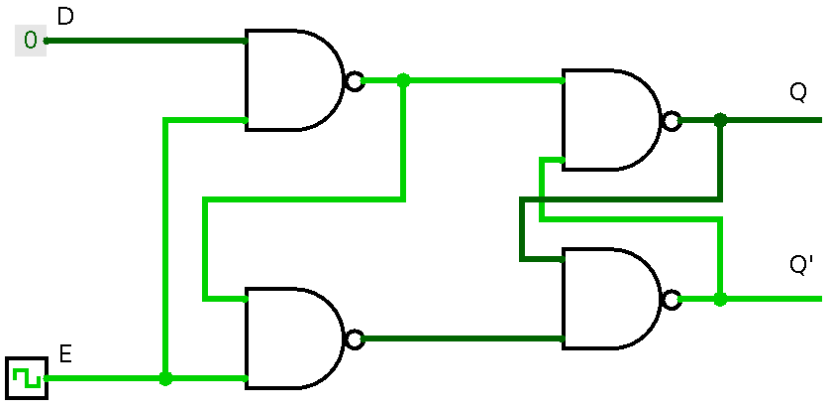


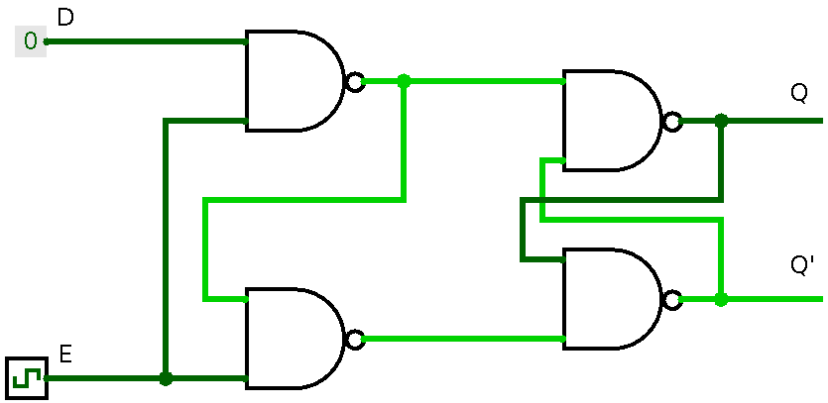
D-latch truth table - III

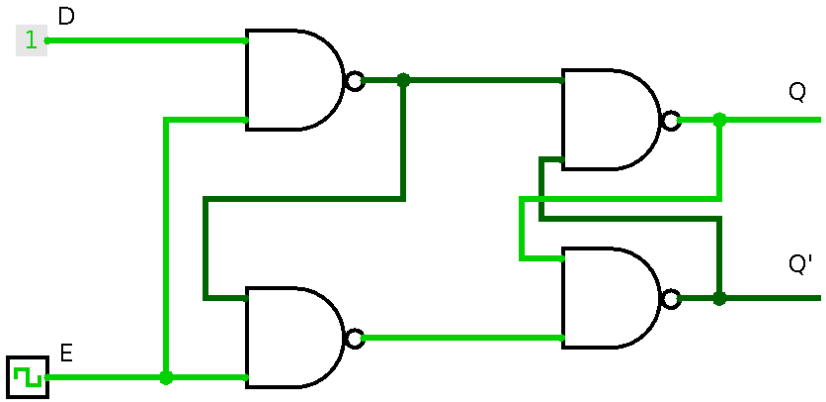
S'	R'	Q	Q'
1	1	hold	
1	0	0	1
0	1	1	0
0	0	Not allowed	

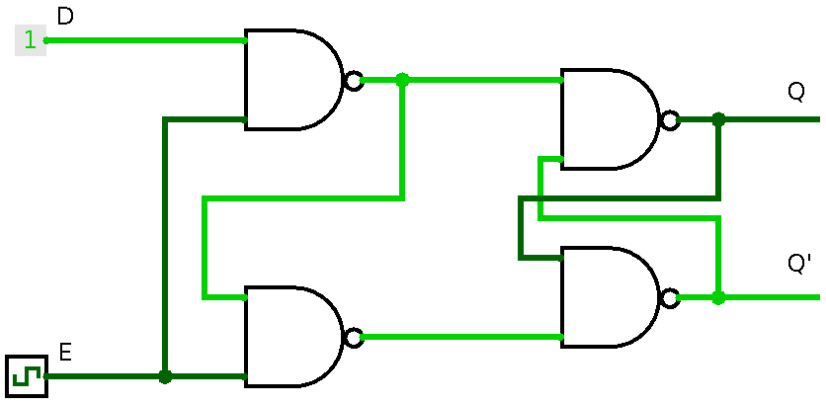
D	E	$\overline{S}$	$\overline{R}$	Q	$\overline{Q}$
0	0	1	1	hold	
0	1	1	0	0	1
1	0	1	1	hold	
1	1	0	1	1	0

D-latch simulation

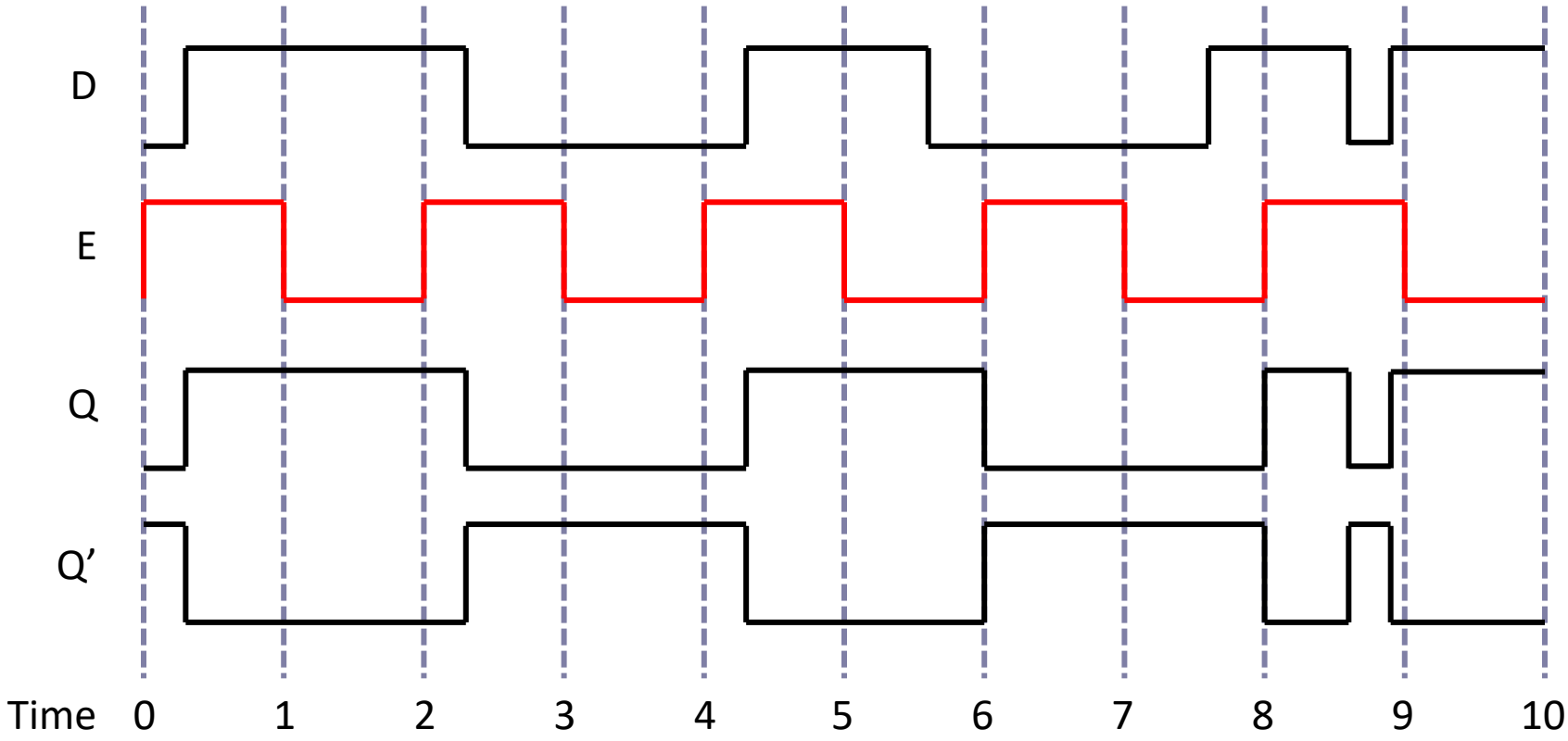




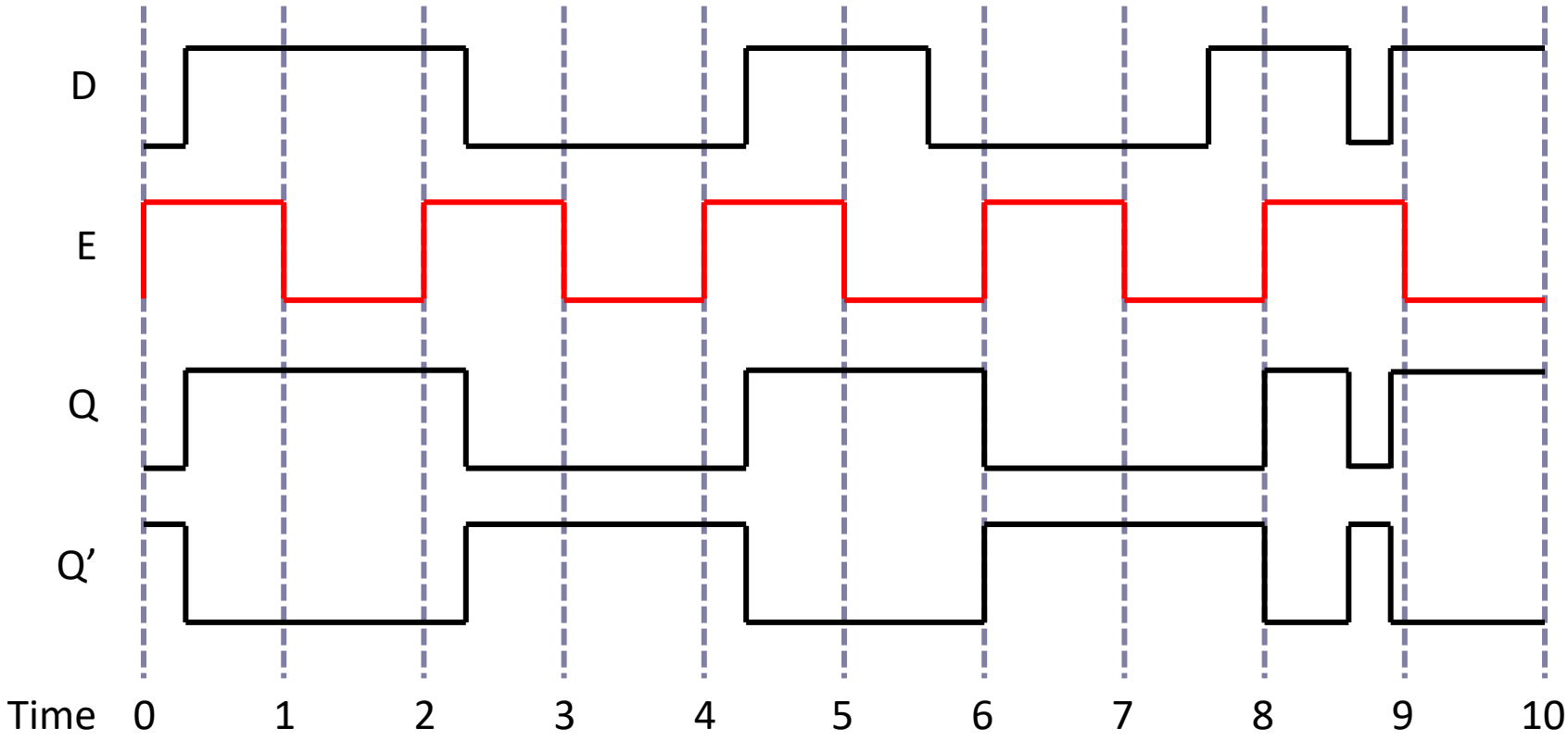




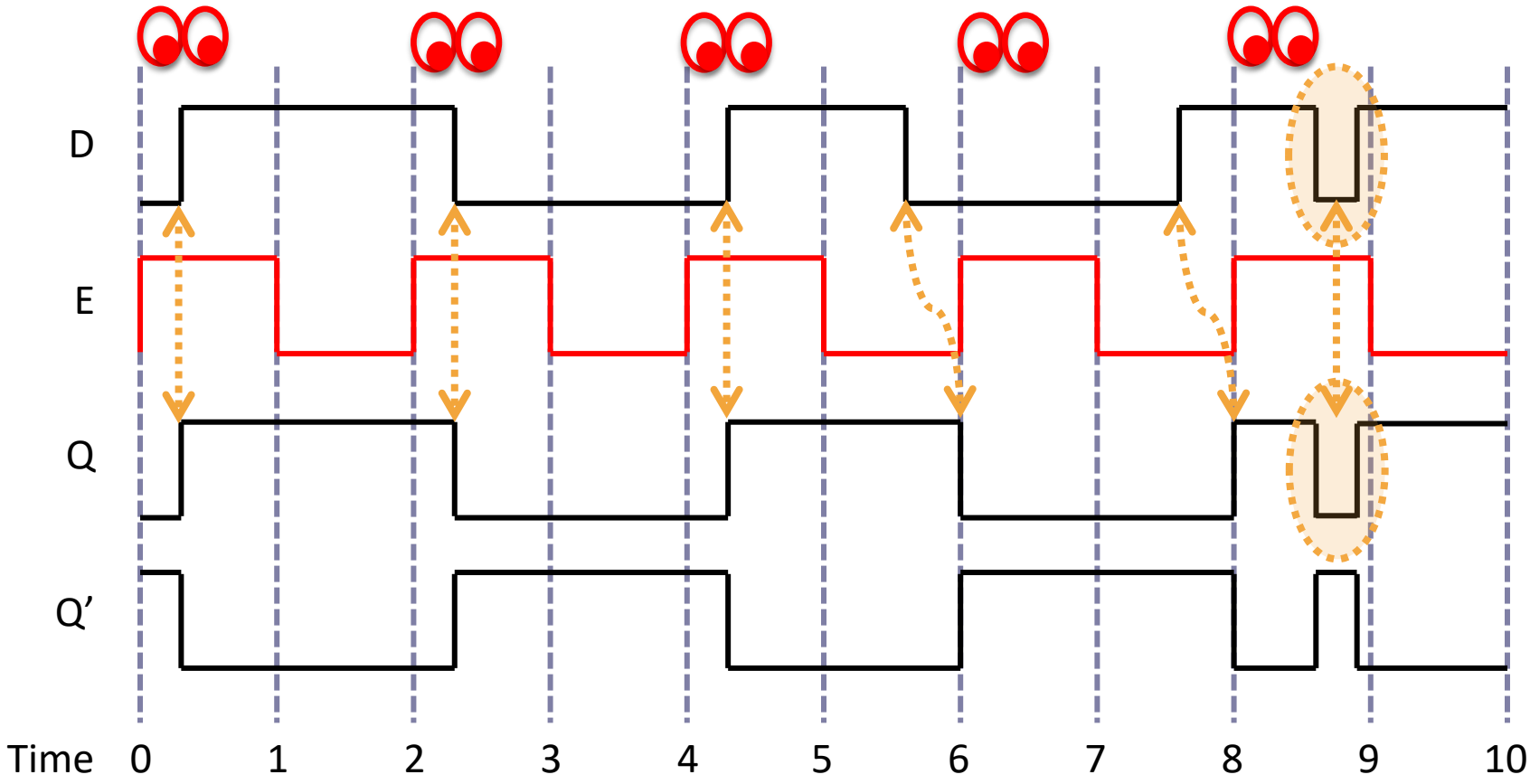
Waveform



Waveform

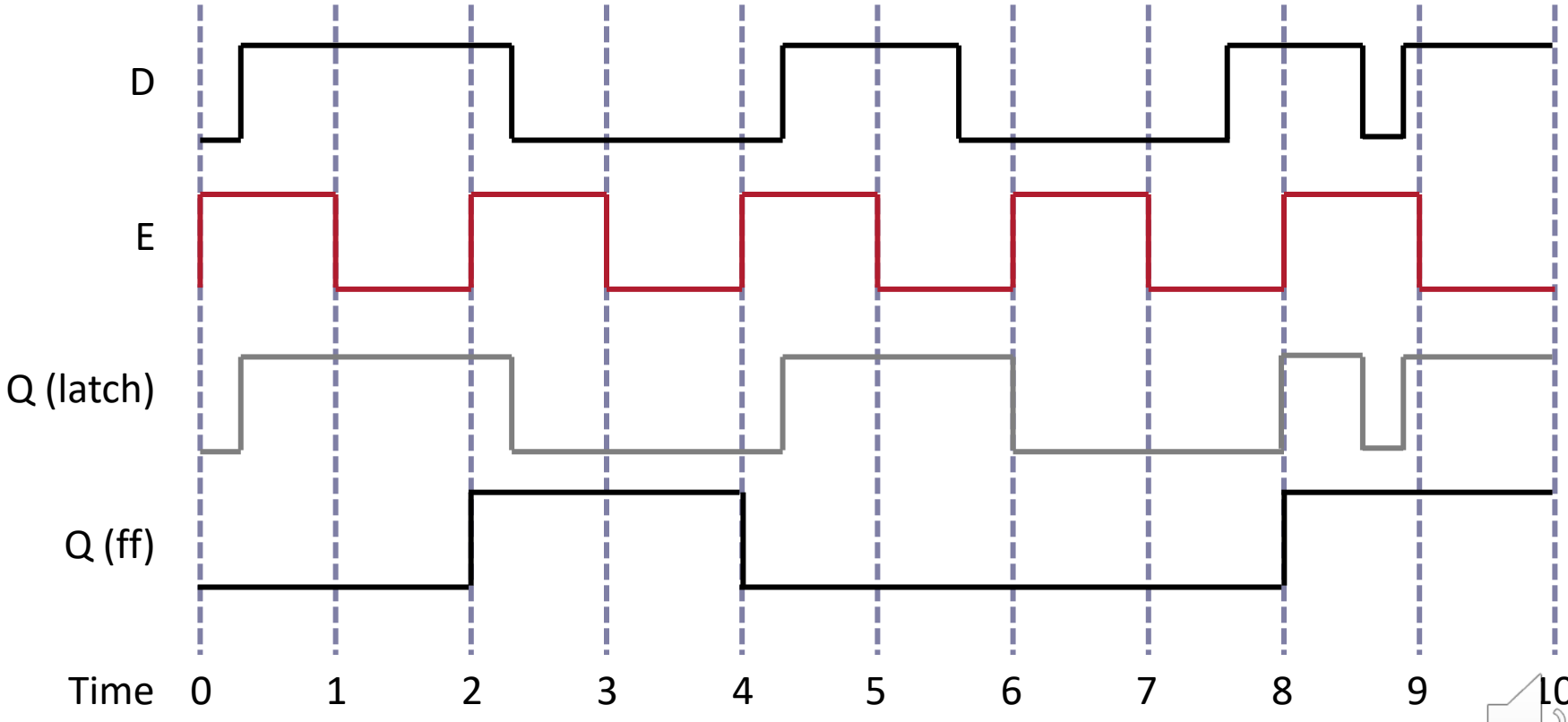


Waveform



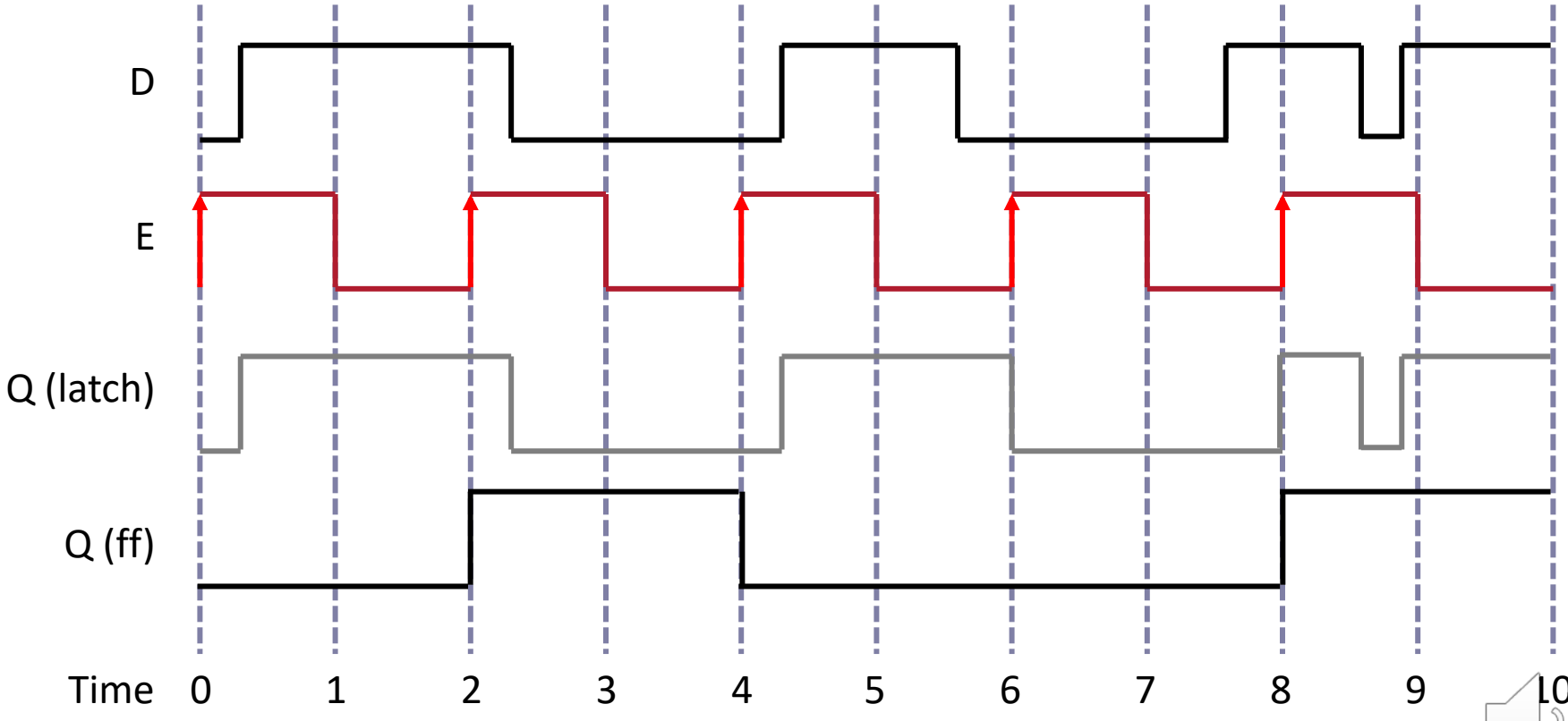
Level vs. edge triggering

- Latches are *level* sensitive.
- We can also build *edge* sensitive devices - Flip-flops



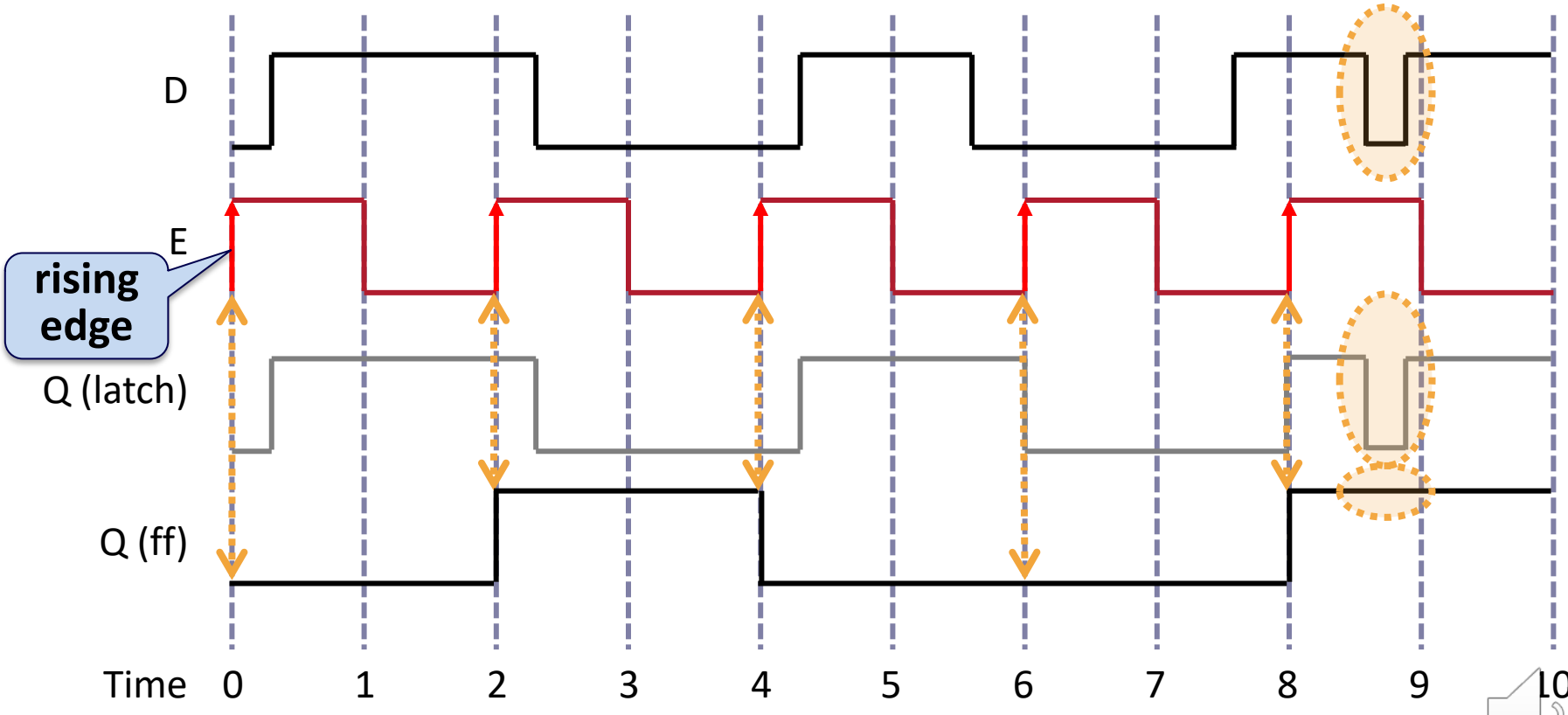
Level vs. edge triggering

- Latches are *level* sensitive.
- We can also build *edge* sensitive devices - Flip-flops

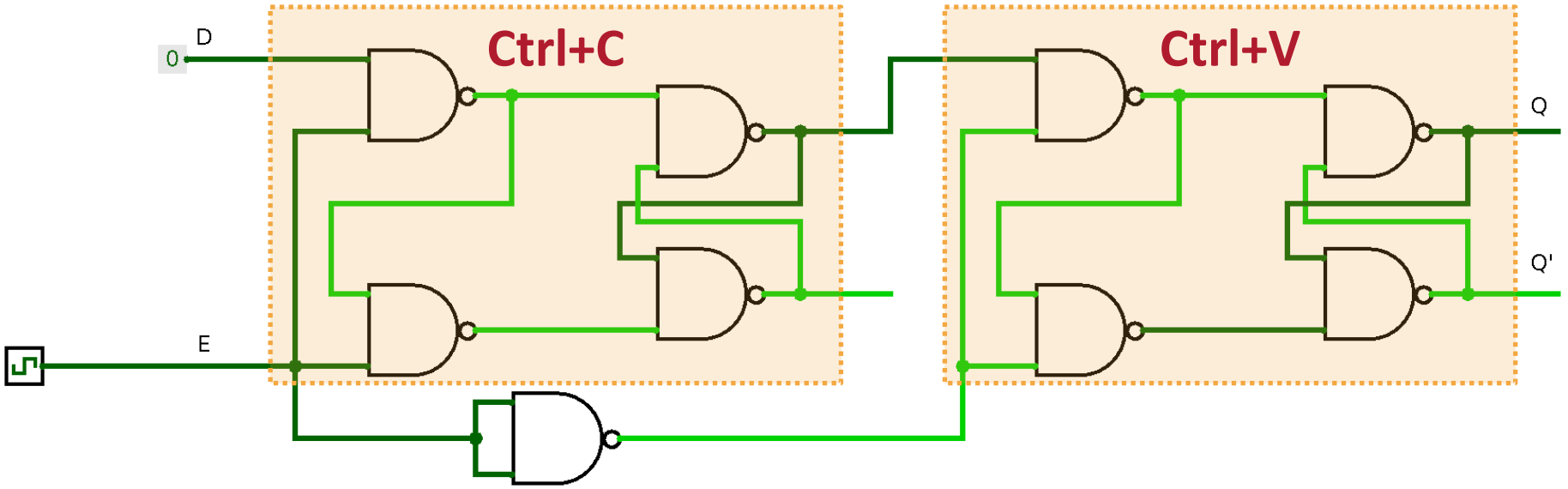


Level vs. edge triggering

- Latches are *level* sensitive.
- We can also build *edge* sensitive devices - Flip-flops



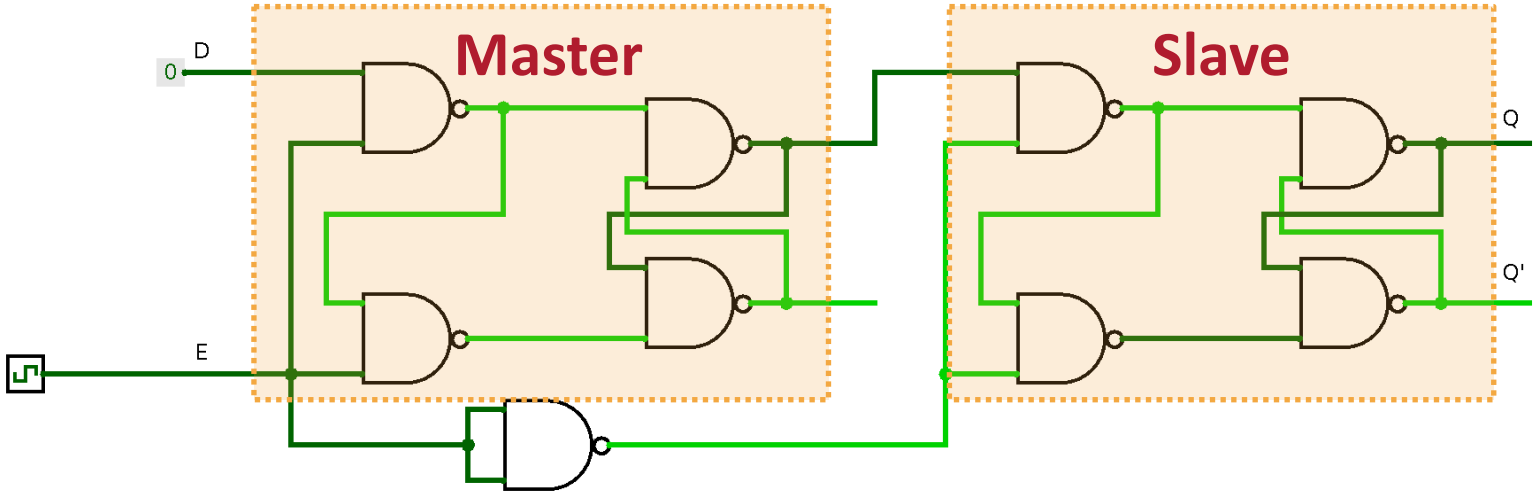
Building a D-type flip-flop



- Two D-type latches.
- Inverted enable signal to the second latch.

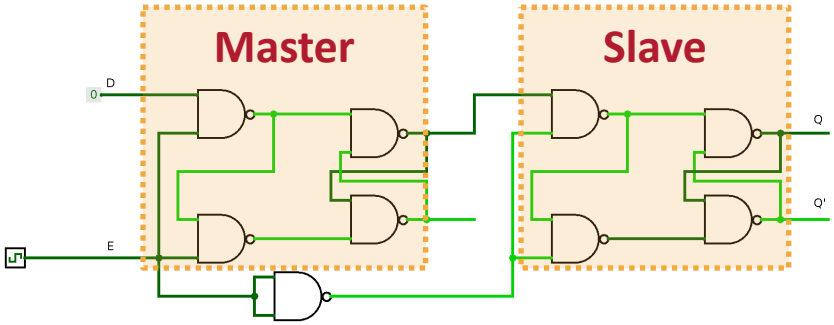
Building a D-type flip-flop

- The example shown is a master-slave D-type flip-flop.
- The master latch is enabled on E high.
  - The slave is disabled, its output is **held**.
- The slave latch is enabled on E low.
  - The master is disabled, its output is held.



D-type flip-flop transition

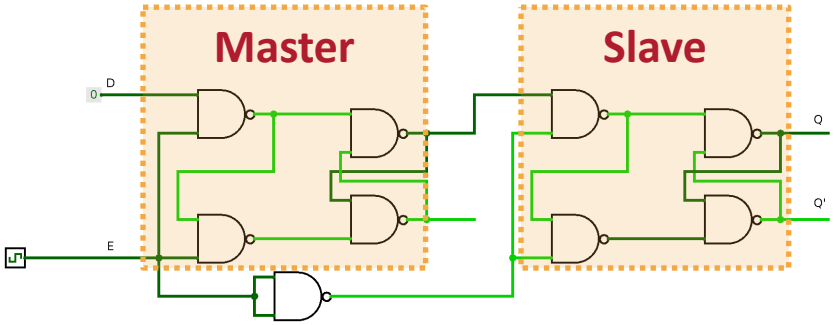
What have we built?



D	E >	Q
0	Rising	Q_prev
0	Falling	0
1	Rising	Q_prev
1	Falling	1

D-type flip-flop transition

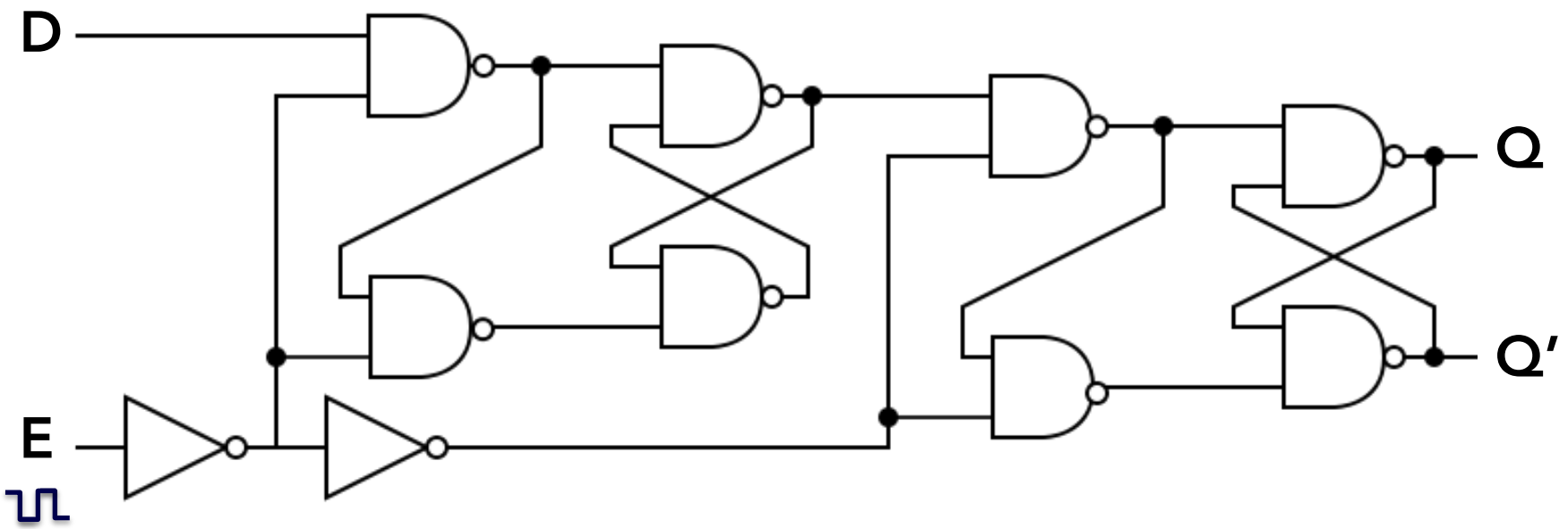
What have we built?



D	E >	Q
0	Rising	Q_prev
0	Falling	0
1	Rising	Q_prev
1	Falling	1

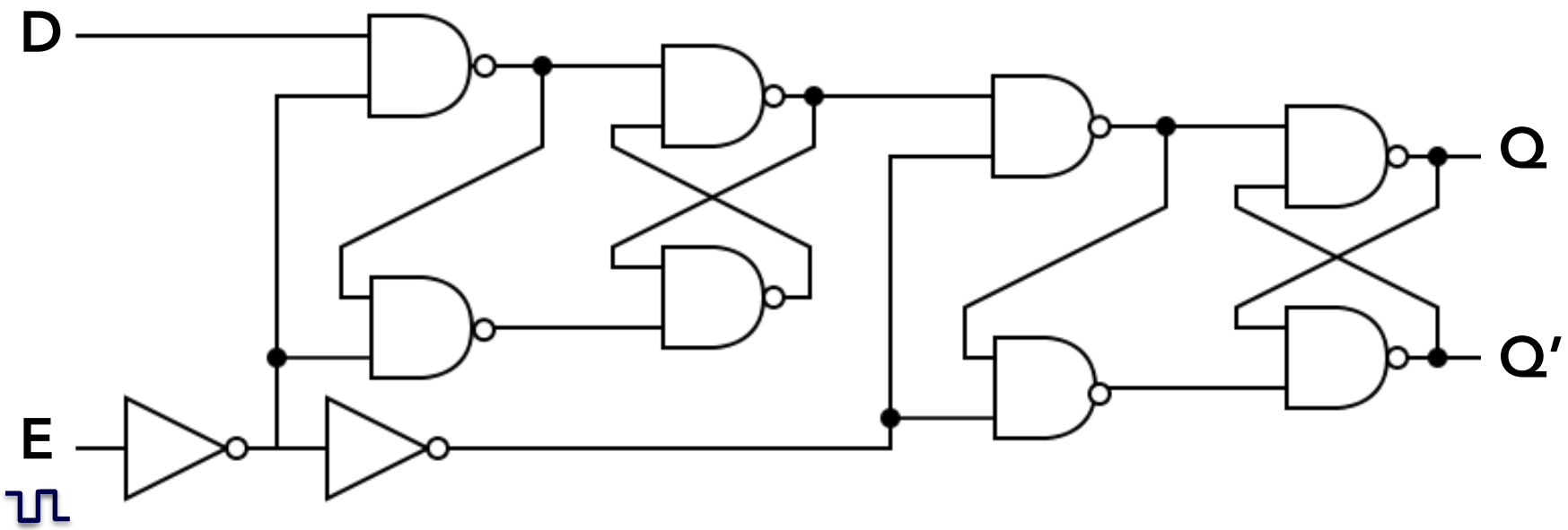
- We have built a **falling-edge** or **negative-edge triggered** flip-flop.
  - Note, the timing diagram earlier demonstrated *positive edge* behaviour.
- There is **more than one way** to make a D-type FF.
  - Explore different designs in the next NAND lab.
- How would we make it **positive-edge**?

How can we make it **positive-edge**?



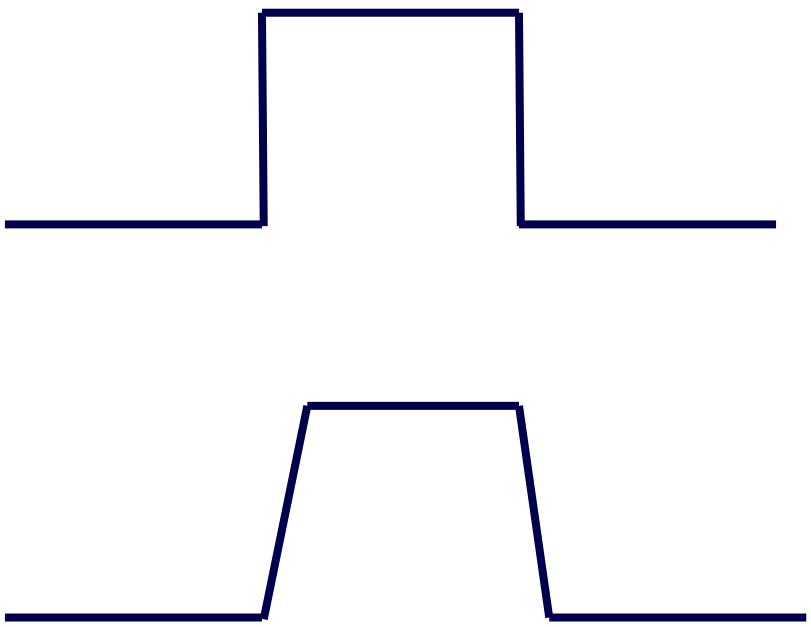
... by adding an inverter that negates the enable signal, E.

How can we make it **positive-edge**?



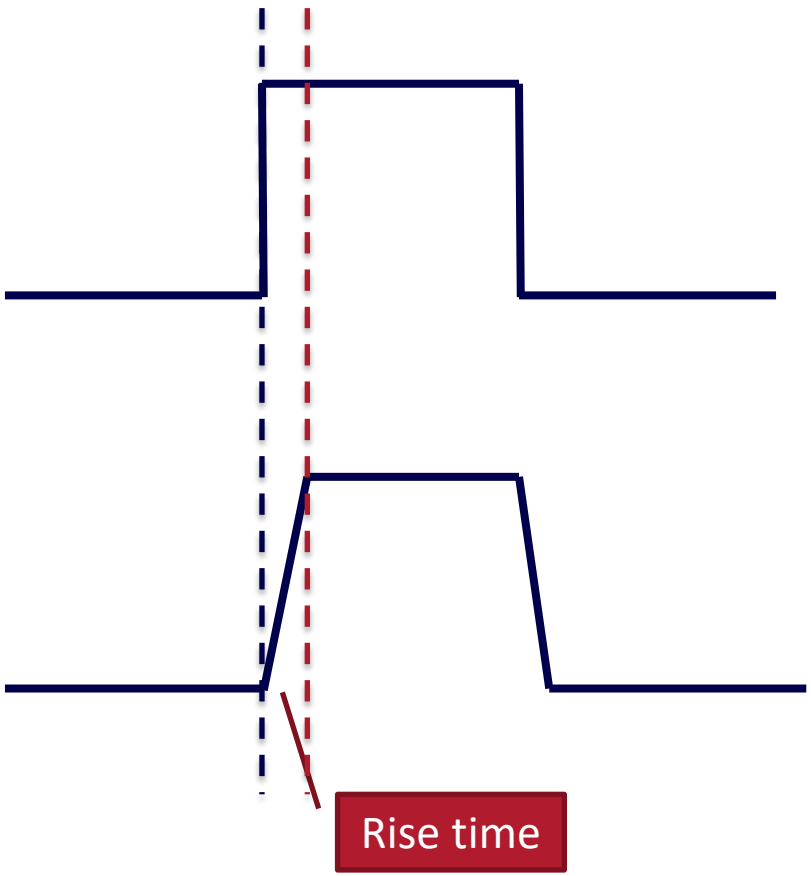
... by adding an inverter that negates the enable signal, E.

Timing



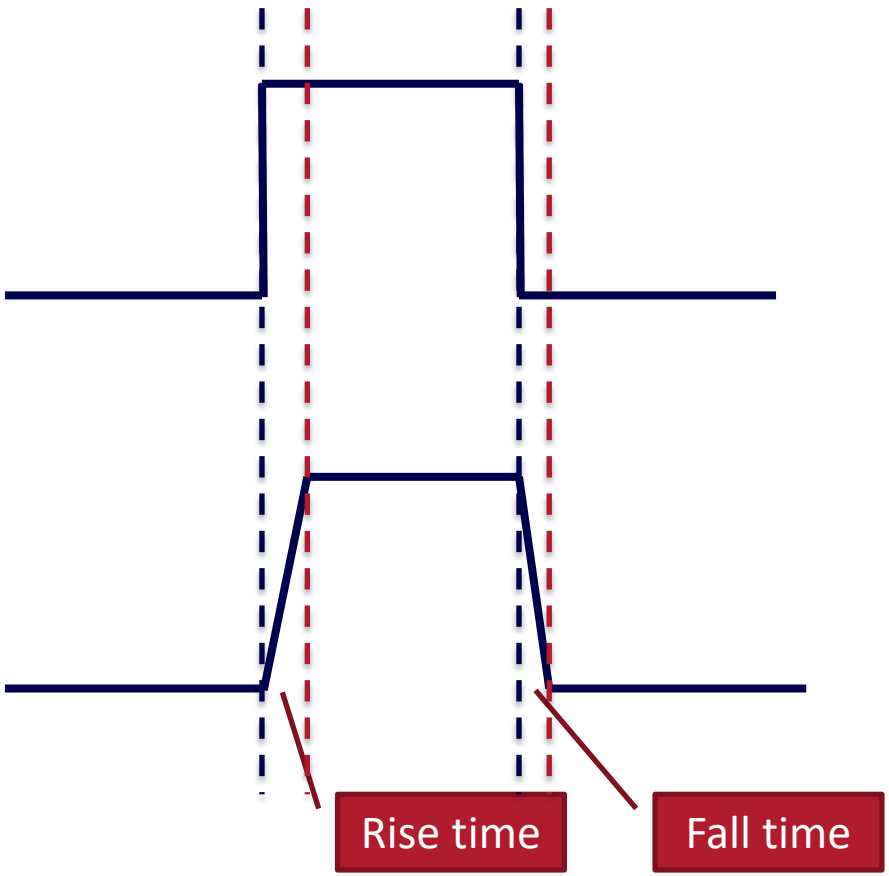
- We imagine signal transitions as instant.
- But they're not!
  - Wires and transistors have capacitances.
  - They have a charge/discharge time.
  - Remember the Ring Oscillator!

Timing: rise times



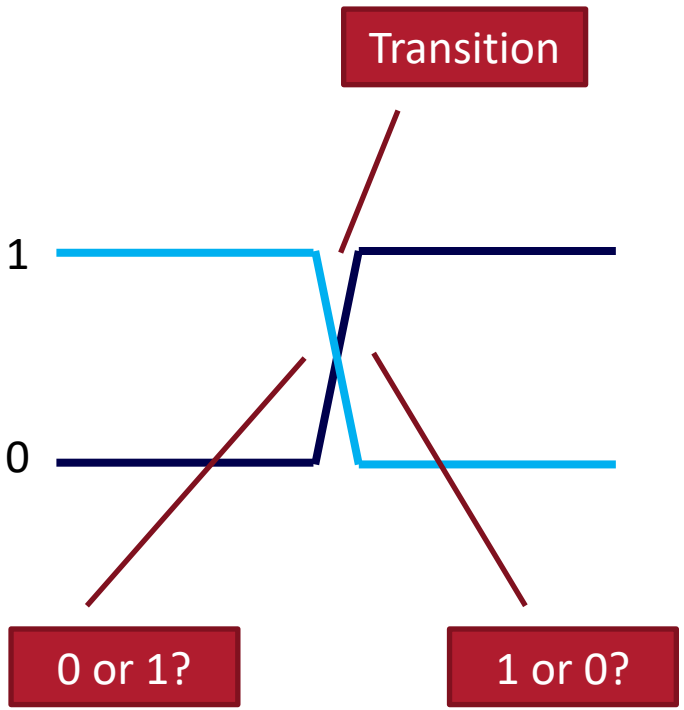
- We imagine signal transitions as instant.
- But they're not!
  - Wires and transistors have capacitances.
  - They have a charge/discharge time.
  - Remember the Ring Oscillator!
- This results in rise & fall times.

Timing: rise times and fall times



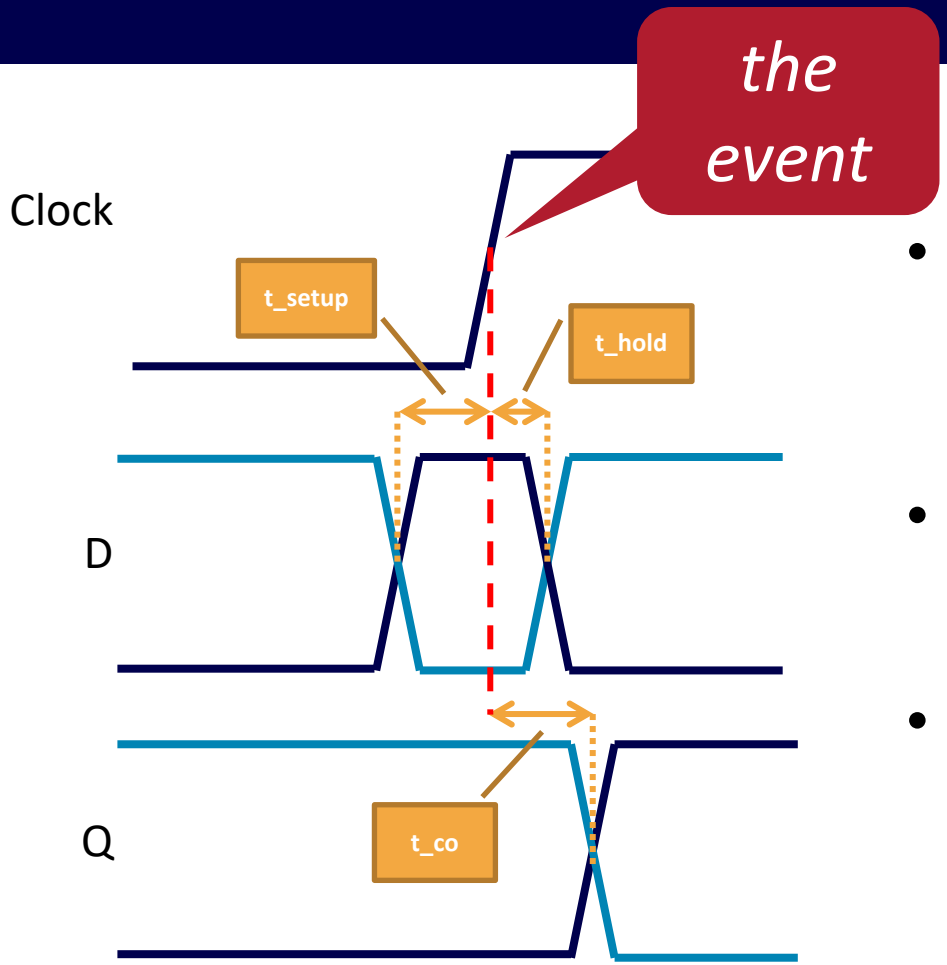
- We imagine signal transitions as instant.
- But they're not!
  - Wires and transistors have capacitances.
  - They have a charge/discharge time.
  - Remember the Ring Oscillator!
- This results in rise & fall times.

Timing: rise times and fall times



- When a signal changes there is a period of time where **we don't know its value**.
- So, if events (i.e. clock edges) happen at the wrong time, then **unexpected behaviour can occur**.

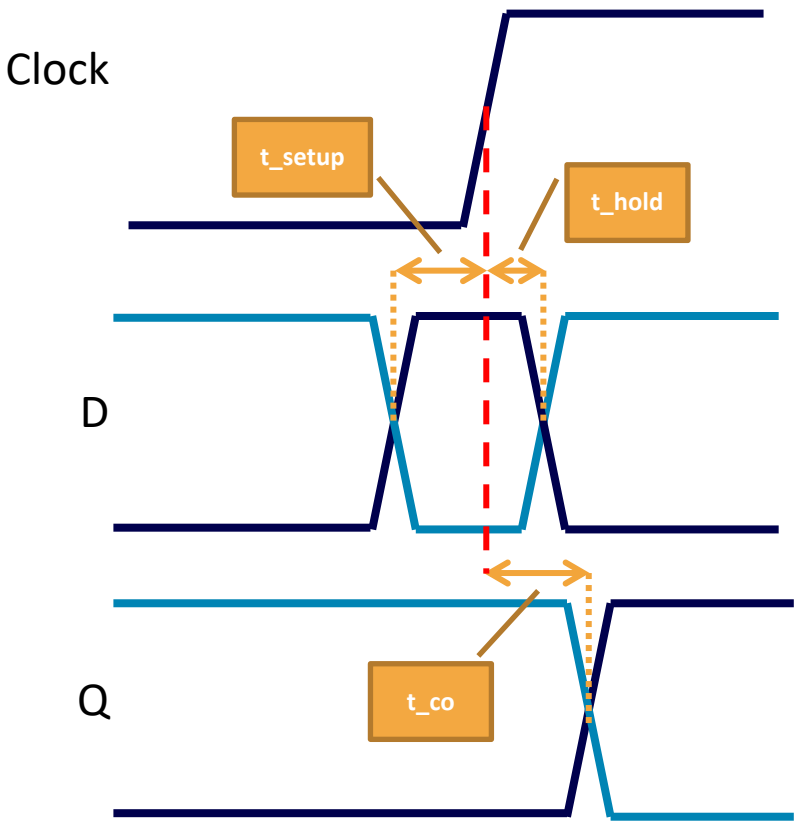
Timing constraints



We specify **timing constraints** to try to avoid these problems.

- Setup time (t_setup)
  - How long the signal should be settled **before** the **event**, e.g. rising clock edge.
- Hold time (t_hold)
  - How long the signal should remain settled **after** the **event.**
- Clock-to-output (t_co)
  - The time between the **event** and the **output changing**.

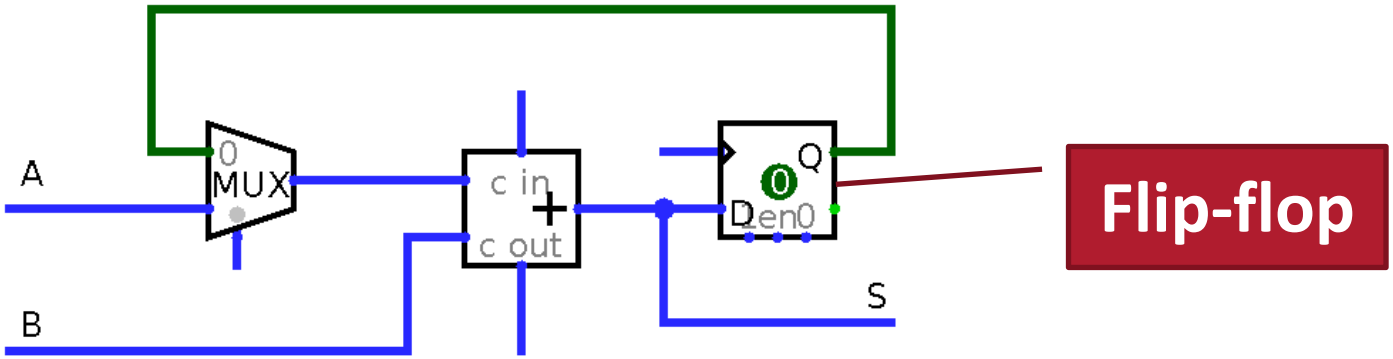
Timing constraints



- We specify **timing constraints** to try to avoid these problems.
- Setup time (t_setup)
  - How long the signal should be settled **before** the **event**, e.g. rising clock edge.
- Hold time (t_hold)
  - How long the signal should remain settled **after** the **event.**
- Clock-to-output (t_co)
  - The time between the **event** and the **output changing**.

Combinatorial vs. sequential logic

- To enforce a sequence reliably, we can:
  - Store result values
  - Control when values are stored
- This allows us to build a sequential system, combining **storage** and **combinatorial logic**.



What have we got now?

- We have enough devices to perform **sequences of operations**.
  - Results can be **stored and reused** in the next operation.
  - We have to set or reset our inputs **by hand** at each stage in the sequence.
  - We have a **clock** that we have to operate **by hand**.
- We have flip-flops – single units of **memory**.
  - Soon we will have more complex memories.
  - And ways of controlling the system **without** manual (by hand) intervention.

In this lecture

Foundations	• Data representation, logic, Boolean algebra.
Building blocks	• Transistors, transistor based logic, simple devices, **storage.**
Modules	• Memory, simple controllers, FSMs, processors and execution.
Programming	• Machine code, assembly, high-level languages, compilers.
Wrap-up	• Operating systems, energy aware computing.

In the next lecture

Foundations	• Data representation, logic, Boolean algebra.
Building blocks	• Transistors, transistor based logic, simple devices, storage.
Modules	• **Memory,** simple controllers, FSMs, processors and execution.
Programming	• Machine code, assembly, high-level languages, compilers.
Wrap-up	• Operating systems, energy aware computing.