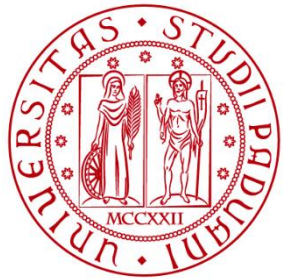
OF THE
DEPARTMENT OF
INFORMATION ENGINEERING



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Digital Systems

Introduction to Sequential Logic

Marta Bagatin, marta.bagatin@unipd.it

Degree Course in Information Engineering

Academic Year 2023-2024

Purpose of the Lesson

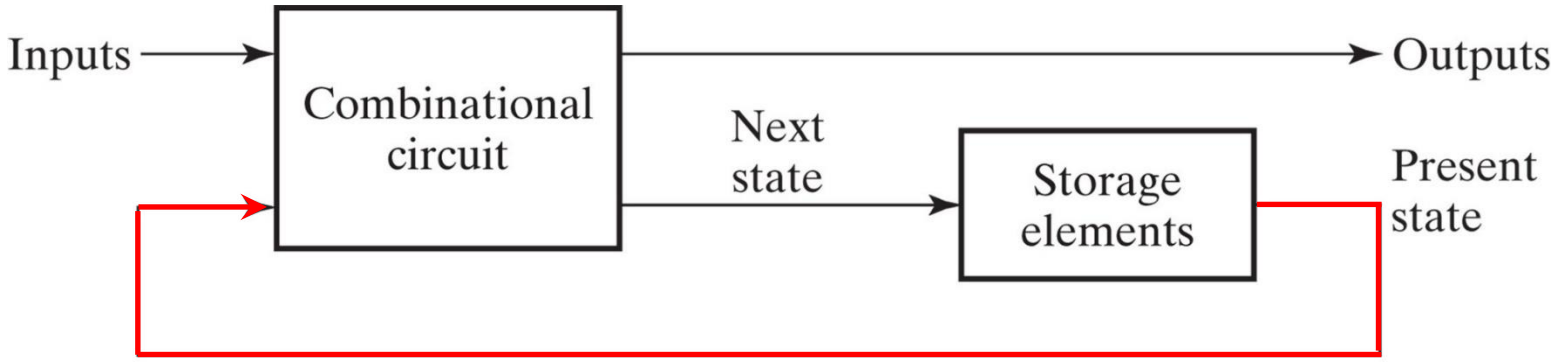
- Introduce sequential circuits and the concept of timing (synchronous vs. asynchronous systems)
- Study the basic sequential circuits used to store information
  - SR Latch
  - D Latch
  - D Flip-flop
- Understand the main parameters for the timing of flip-flops

Sequential circuit: Definition

- In a sequential circuit the **outputs depend not only on the current inputs, but also on their past history (state of the system)**
- Different from a combinational circuit, a sequential circuit **has memory**
- Using sequential circuits we can create digital systems capable of storing information between one operation and the next one

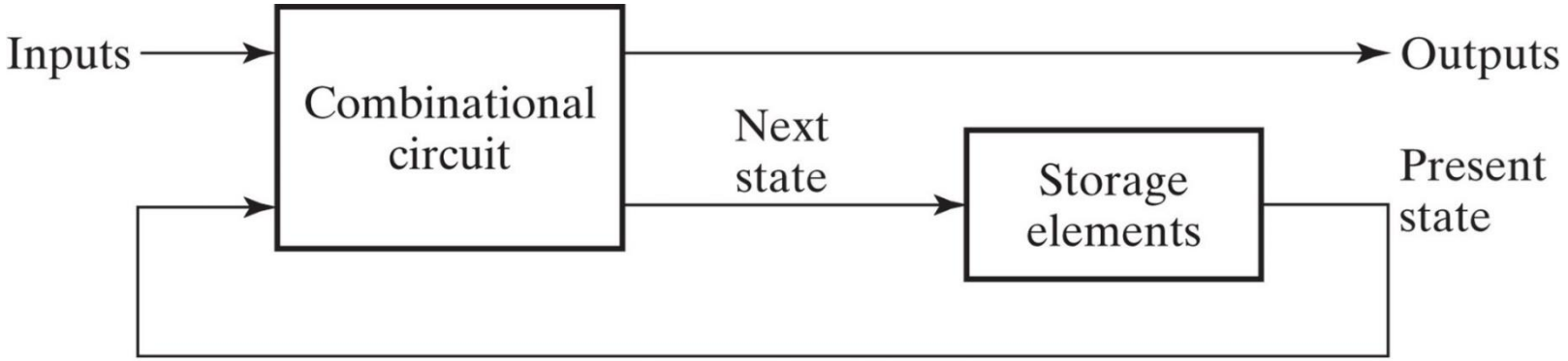
Block Diagram of a Sequential Circuit

- A sequential circuit is formed by **interconnecting a combinational circuit with one or more storage elements**



- At any given time, the binary information contained in the storage elements is called the state of the system
- There is a **feedback path**: the output of the storage element is connected to the input of the system
  - Note: combinational circuits are acyclic

Block Diagram of a Sequential Circuit



- The **outputs and the next state are calculated by the combinational part**, based on the inputs and the present state
- The **present state** is given by the outputs of the memory elements
- The **next state** is given by the inputs of the memory elements

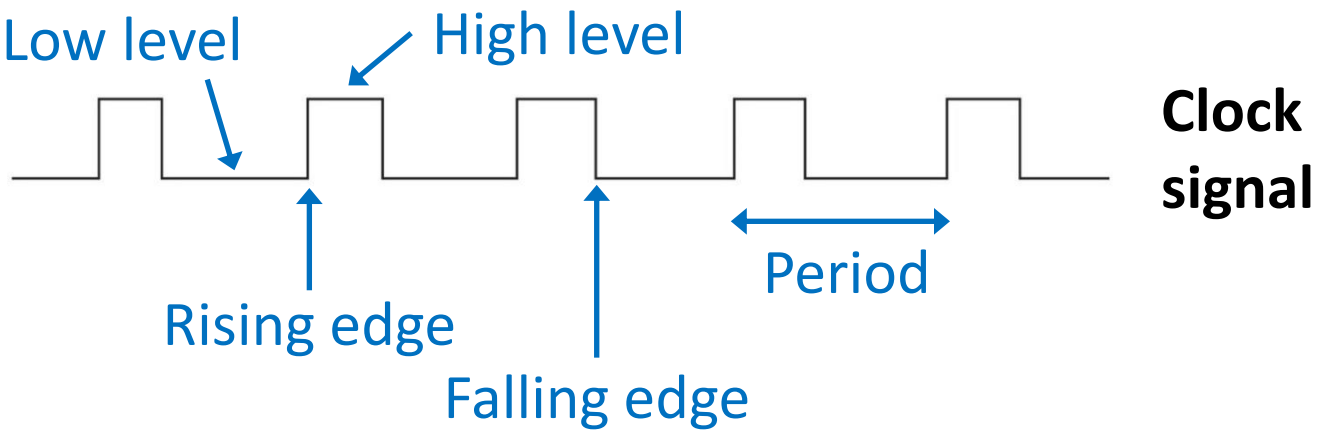
- A sequential circuit is specified by a time sequence of inputs, internal states, and outputs

Synchronous and Asynchronous Circuits

- A sequential circuit can be synchronous or asynchronous, depending on the **instants of time when its inputs are evaluated and its status is updated**
- **Synchronous sequential circuit** (also known as **Finite State Machine, FSM**): changes in the state of the system are synchronized with a signal called **clock**
  - The behavior of a synchronous circuit is determined by the knowledge of the signals at discrete instants of time
- **Asynchronous sequential circuit**: there is no clock signal, the state can change at any instant of time
- Most modern digital devices use synchronous circuits. Asynchronous circuits have some advantages (e.g. speed, power consumption), but they are generally more complex and less robust
  - In this course, we will only study synchronous circuits

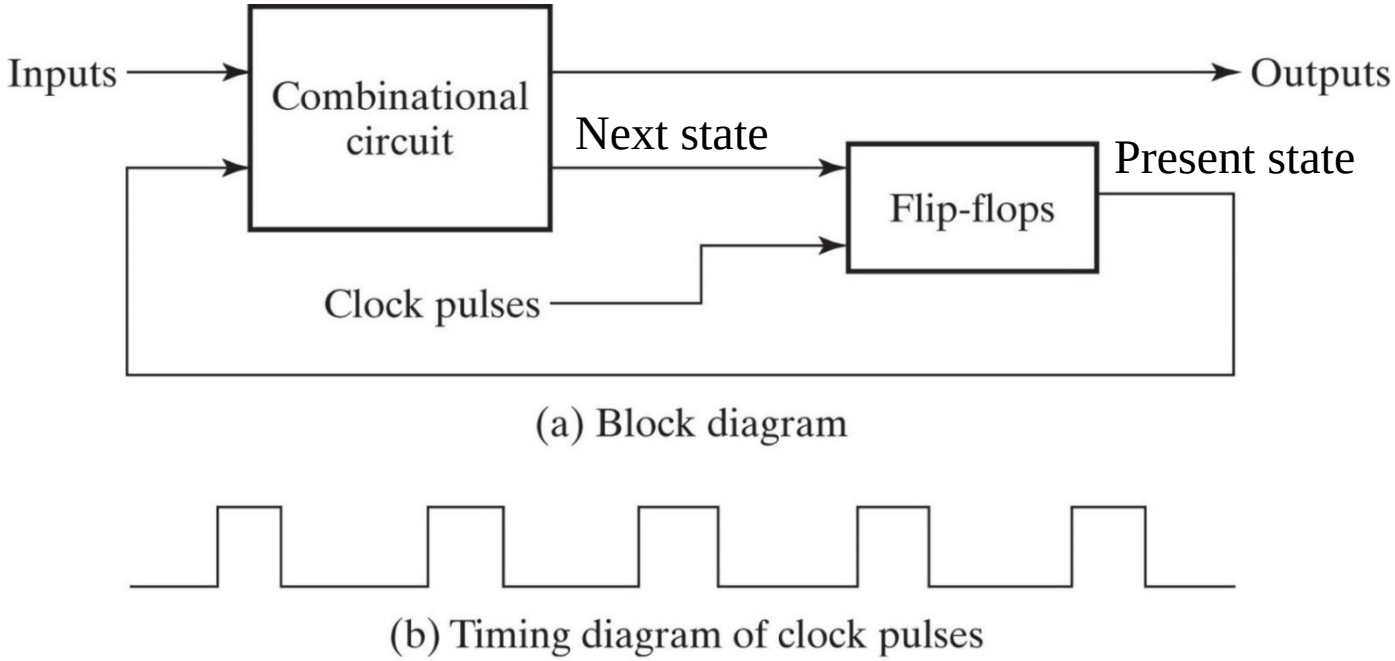
Synchronous Sequential Circuits

- **Timing is given by a periodic signal, called clock**, consisting of a square wave



- The clock signal is distributed within the system so that all memory elements are updated only in the presence of specific events on the clock (e.g. rising edge)
- Synchronous circuits are the most used sequential circuits, as their design is simpler and they are more reliable

Synchronous Sequential Circuits

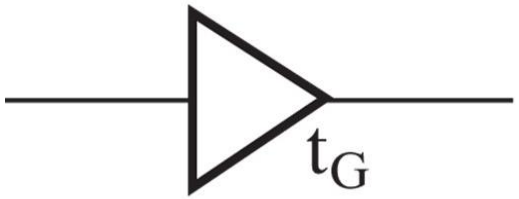


(a) Block diagram

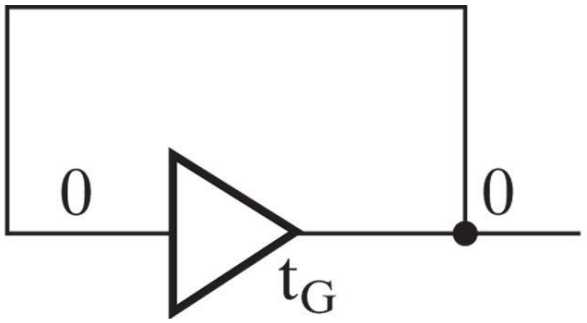
(b) Timing diagram of clock pulses

- The clock signal **is one of the inputs of the memory elements**, which in synchronous sequential circuits are called flip-flops
- The **flip-flop output can only change in the presence of specific events** of the clock (e.g. at the rising edge of the clock). Thus, the state is updated (not necessarily changes) at each clock cycle: the next state becomes the present state

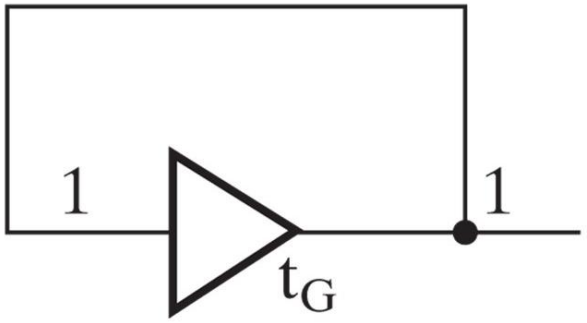
Memory Elements



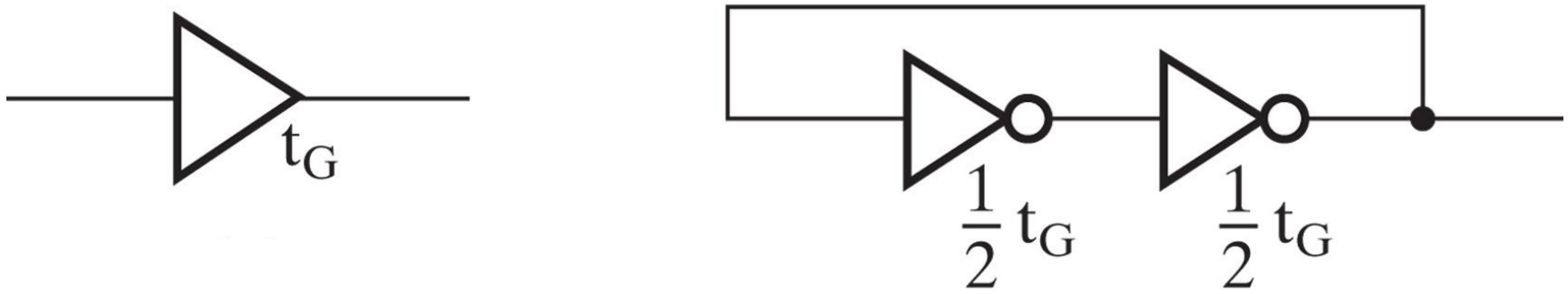
- Buffer with delay t_G.: information present at the buffer input at time t appears at the buffer output at time t + t_G



- Let us now add a **feedback path**: we connect the buffer output to its input
- If the input has been constant ('0' or '1') for at least t_G, then the buffer output will still be stable at that value at time t +t_G
- We have created a memory element, i.e. an element that stores information indefinitely



Memory Elements



- The most common way to realize a memory element is to connect **two inverters in cascade** (each with a delay time of ½ t_G.). In this way, the signal is complemented twice and the original signal is obtained at the output:

$$\overline{\overline{X}} = X$$

  - This is the basic operation principle of a SRAM (Static Random Access Memory) cell
- In the following slides, we will see other ways to create memory elements with additional inputs that allow us to control the value of the stored data

Latch and Flip-flop

Flip-flop

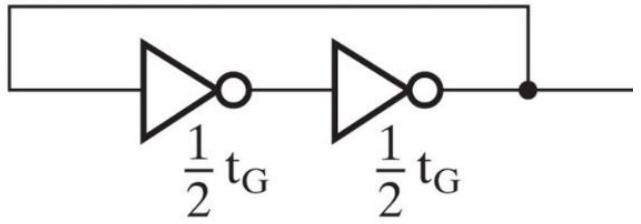
- The memory elements used in synchronous sequential circuits are called **flip-flops**
- A flip-flop is a **binary memory element able to store a single bit and timed by a clock signal**
  - Inputs: input signal and clock
  - Outputs: output signal and (optional) complemented output
- Flip-flops are **made with basic elements called latches**

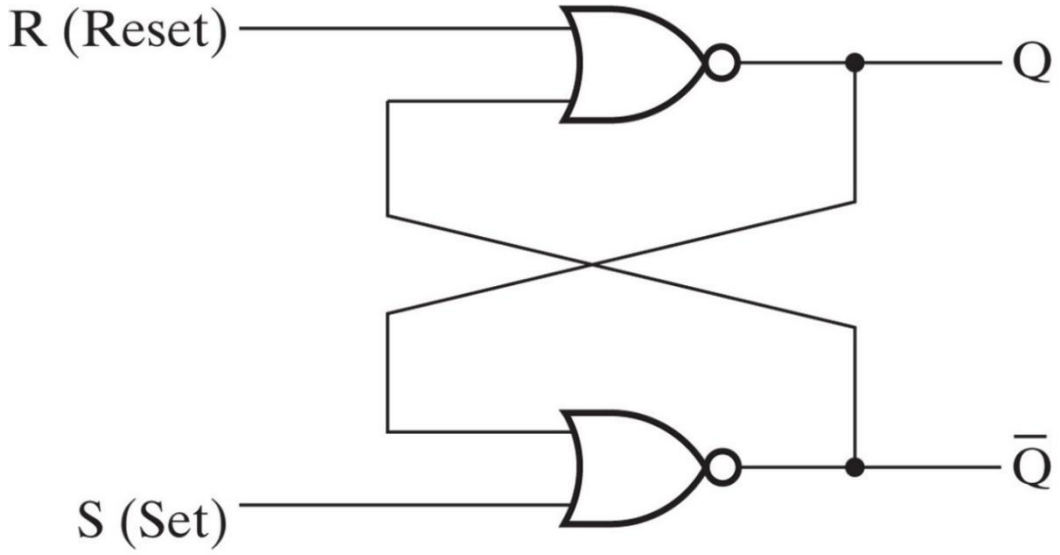
Latch

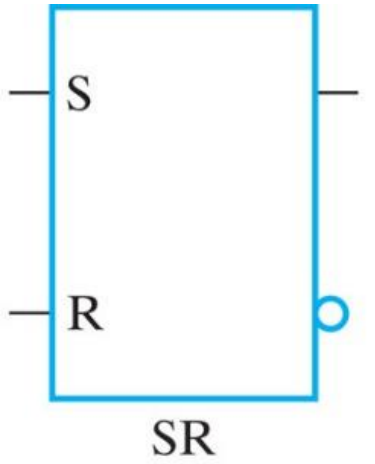
- The latch is the **basic buildng block** of sequential circuits
- It is a **bistable circuit** (i.e. a circuit with two stable states) storing a single bit of information and it is not clocked

SR Latch

- The **Set Reset (SR) latch** is derived from the simplest memory structure with two crossed-coupled inverters, replacing the inverters with **two NOR gates**

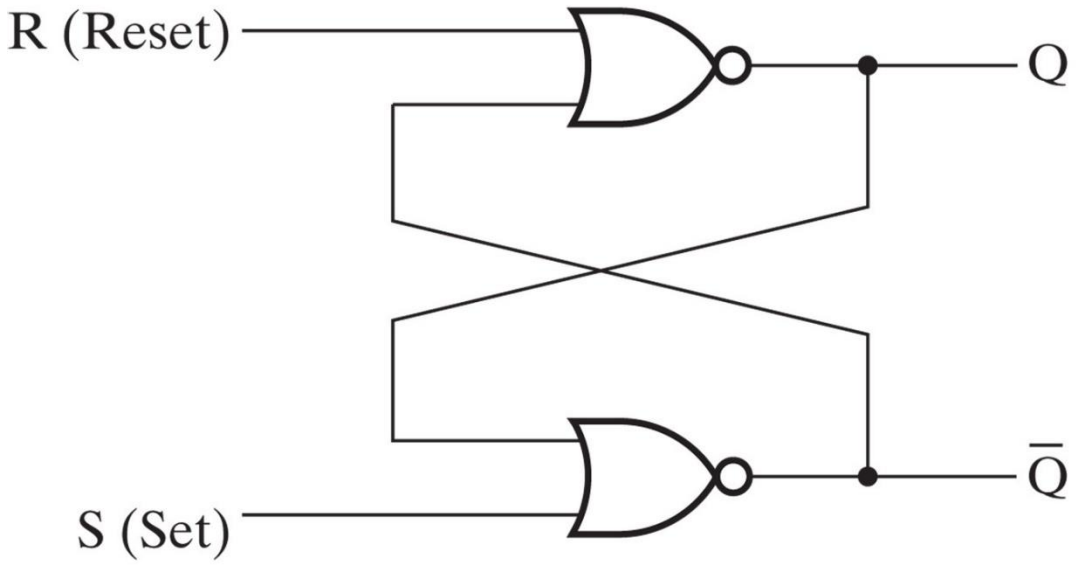






By changing the value of the inputs R and S we can force the value of the bit stored in the latch

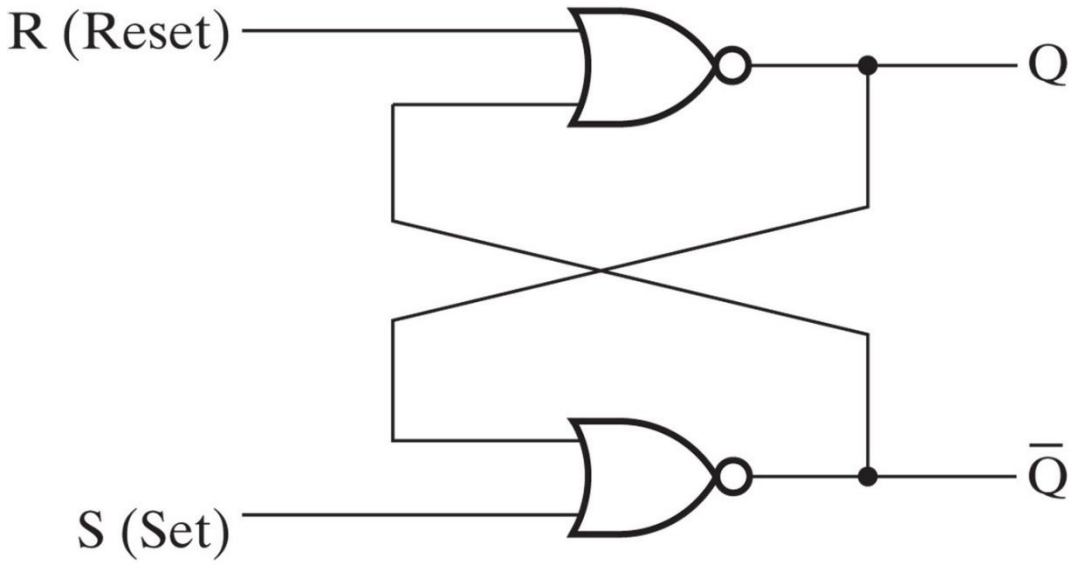
SR Latch: Operation



NOR gate: it is enough that one input is '1' for the output to be '0'

- **S = '1', R = '0'**: $\overline{Q} = 0,\ Q = 1$ => **Set state**
- **R = '1', S = '0'**: $Q = 0,\ \overline{Q} = 1$ => **Reset state**
- **S = R = '0'**: **Memory state**, the circuit maintains the previous state
- During normal operation, Q and $\overline{Q}$ they are the complements of each other
- **S = R = '1'**: **Forbidden/undefined state** -> both Q and $\overline{Q}$ are '0', violating the requirement that the outputs are the complements of each other -> when (S,R) switch from (1, 1) to (0, 0) it is not possible to predict the value of the outputs

SR Latch: Operation and Function Table

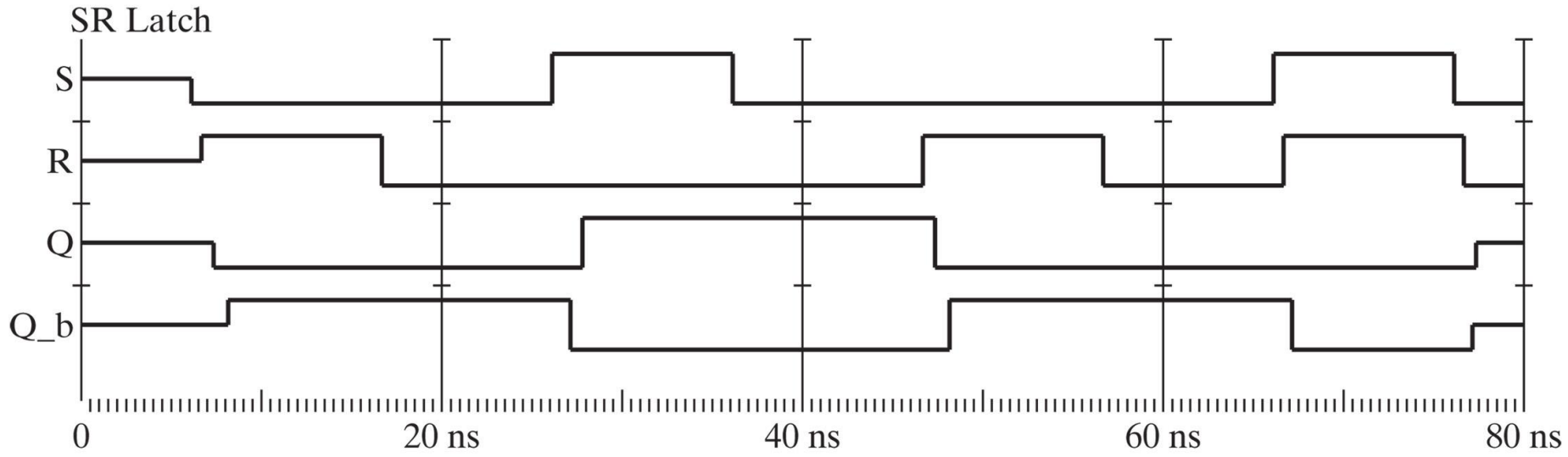


S	R	Q	$\overline{Q}$	
1	0	1	0	Set state
0	0	1	0	
0	1	0	1	Reset state
0	0	0	1	
1	1	0	0	Undefined

NOR gate: it is enough that one input is '1' for the output to be '0'

- **S = '1', R = '0'**: $\overline{Q} = 0,\ Q = 1$ => **Set state**
- **R = '1', S = '0'**: $Q = 0,\ \overline{Q} = 1$ => **Reset state**
- **S = R = '0'**: **Memory state**, the circuit maintains the previous state
- During normal operation, Q and $\overline{Q}$ they are the complements of each other
- **S = R = '1'**: **Forbidden/undefined state** -> both Q and $\overline{Q}$ are '0', violating the requirement that the outputs are the complements of each other -> when (S,R) switch from (1, 1) to (0, 0) it is not possible to predict the value of the outputs

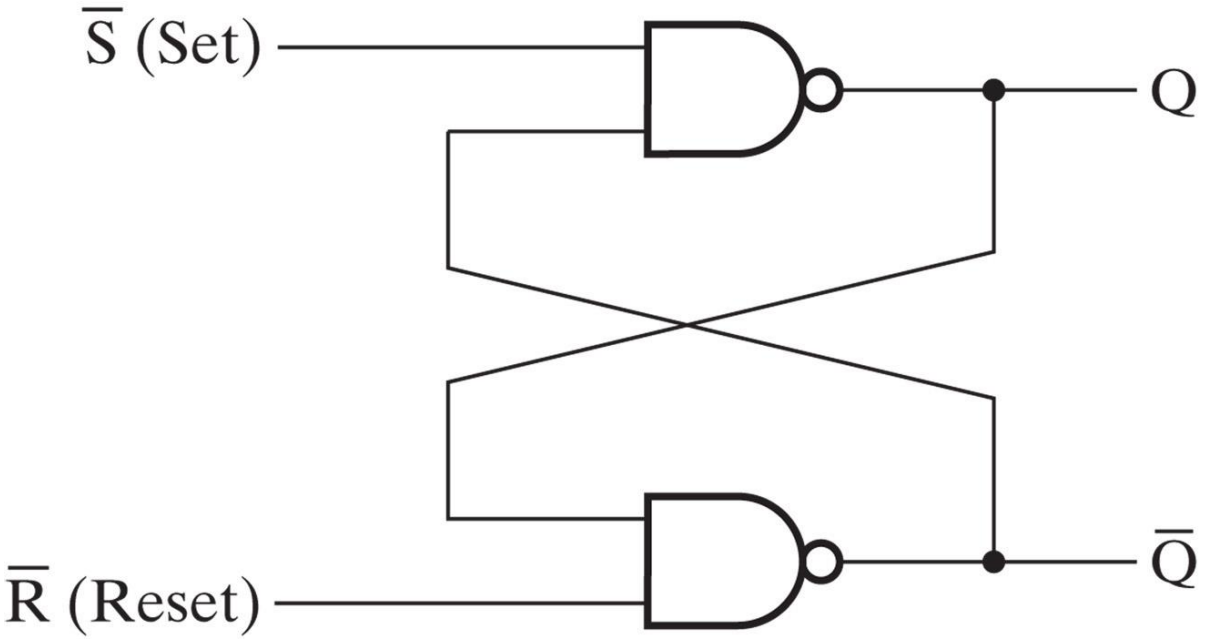
SR Latch: Time Diagram

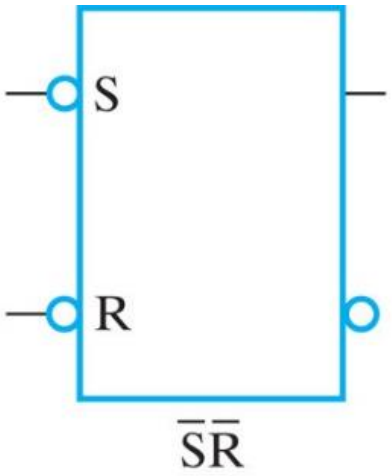


- Logic simulation of the temporal behavior of outputs Q e $\overline{Q}$ (Q_b) in response to changes in the inputs S and R
- When S = R = 1, both Q e $\overline{Q}$ go to 0. At the next transition S = R = 0, the output goes to an intermediate state between logic '0' and '1'
  - This is the simulation of an **ideal situation**, where commutations of S and R occur simultaneously and the two NOR gates have the same delay
  - In a **real** circuit, the value of the outputs after the commutation of inputs (S, R) from (1,1) to (0,0) depends on the order in which S and R commute and on the NOR gates delay

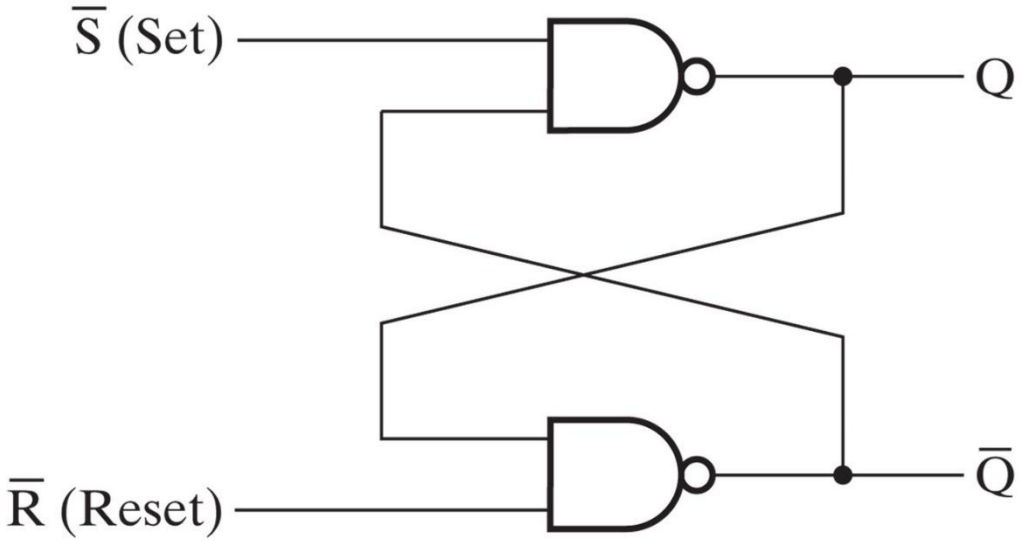
$\overline{S}\,\overline{R}$ Latch

- It is obtained from the Set Reset (SR) latch, replacing the two NOR gates with **two NAND gates**



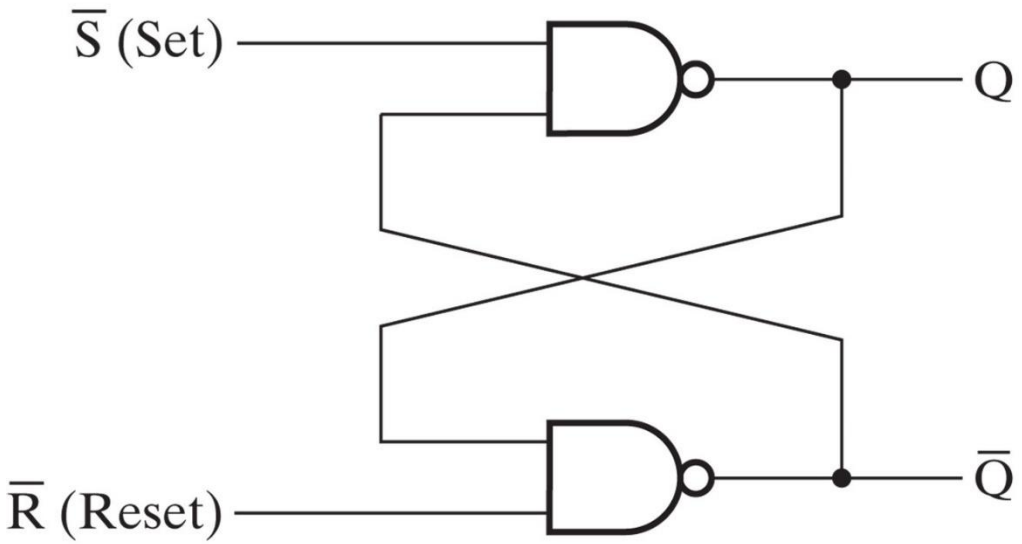


$\overline{S}\,\overline{R}$ Latch: Operation



NAND gate: it is enough that one input is '0' for the output to be '1'

$\overline{S}\,\overline{R}$ Latch: Operation and Function Table



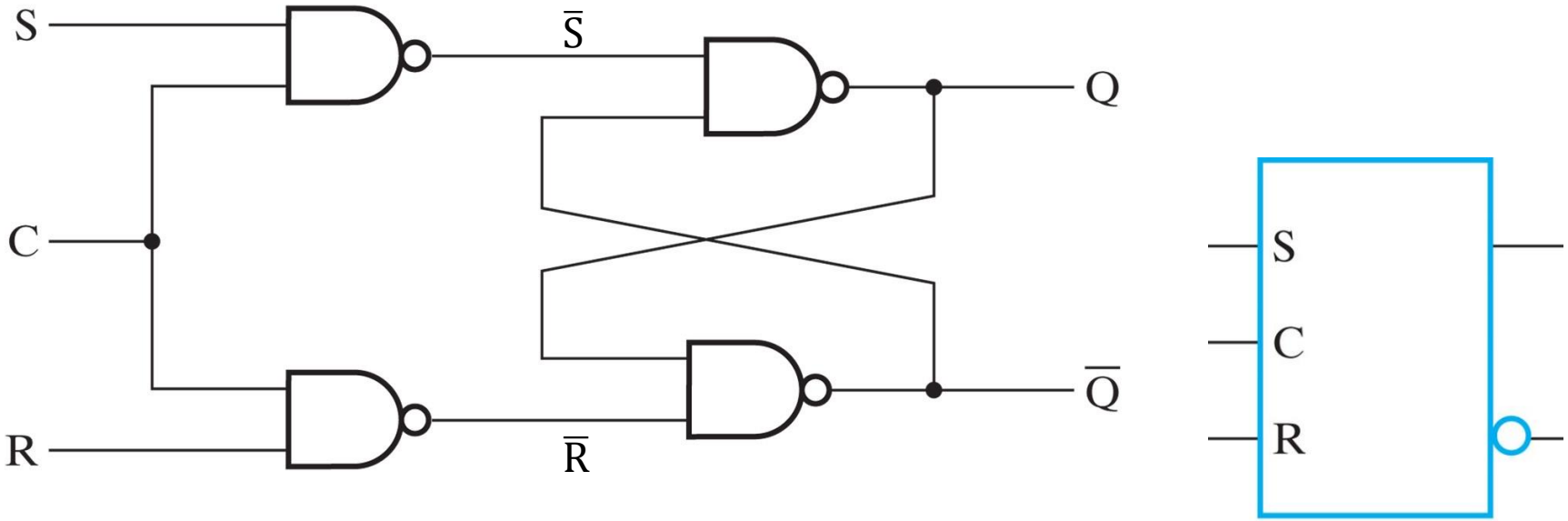
$\overline{S}$	$\overline{R}$	Q	$\overline{Q}$	
0	1	1	0	Set state
1	1	1	0	
1	0	0	1	Reset state
1	1	0	1	
0	0	1	1	Undefined

NAND gate: it is enough that one input is '0' for the output to be '1'

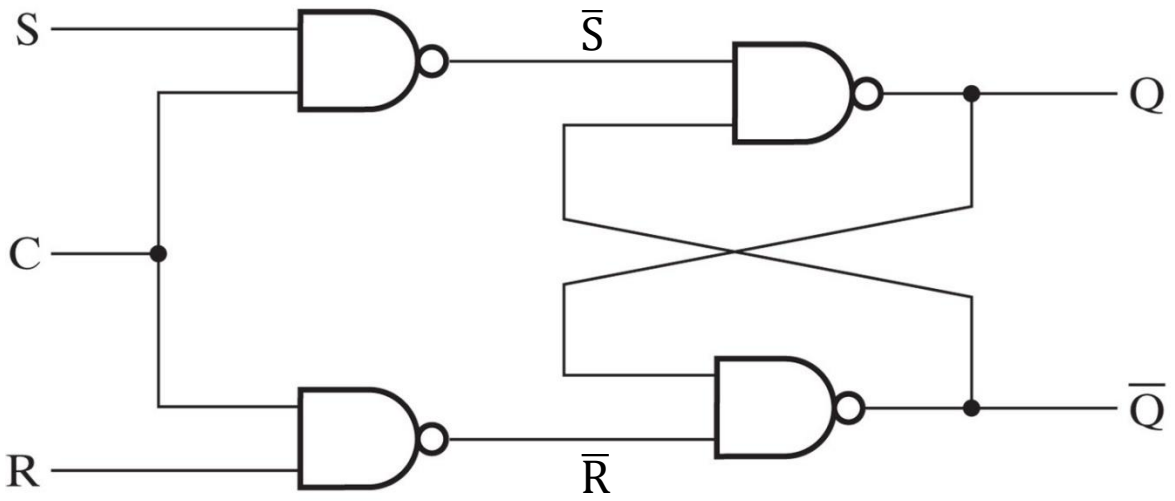
- $\overline{\mathbf{S}} = '\mathbf{0}', \overline{\mathbf{R}} = '\mathbf{1}'$: $Q = 1,\ \overline{Q} = 0$, => **Set state**
- $\overline{\mathbf{R}} = '\mathbf{0}', \overline{\mathbf{S}} = '\mathbf{1}'$: $\overline{Q} = 1, Q = 0$, => **Reset state**
- $\overline{\mathbf{S}} = \overline{\mathbf{R}} = '\mathbf{1}'$: **Memory state**, the circuit maintains the previous state
- $\overline{\mathbf{S}} = \overline{\mathbf{R}} = '\mathbf{0}'$: **Forbidden/undefined state** -> both Q and $\overline{Q}$ are '1', violating the requirement that the outputs are the complements of each other -> if $(\overline{\mathbf{S}}, \overline{\mathbf{R}})$ commute from (0, 0) to (1, 1), the values of Q and $\overline{Q}$ cannot be predicted

SR Latch with Control Input

- Compared to the SR latch with NAND gates, it adds a further **control input C**, which acts as an **enable signal**, enabling or not the change of the latch state



SR Latch with Control Input

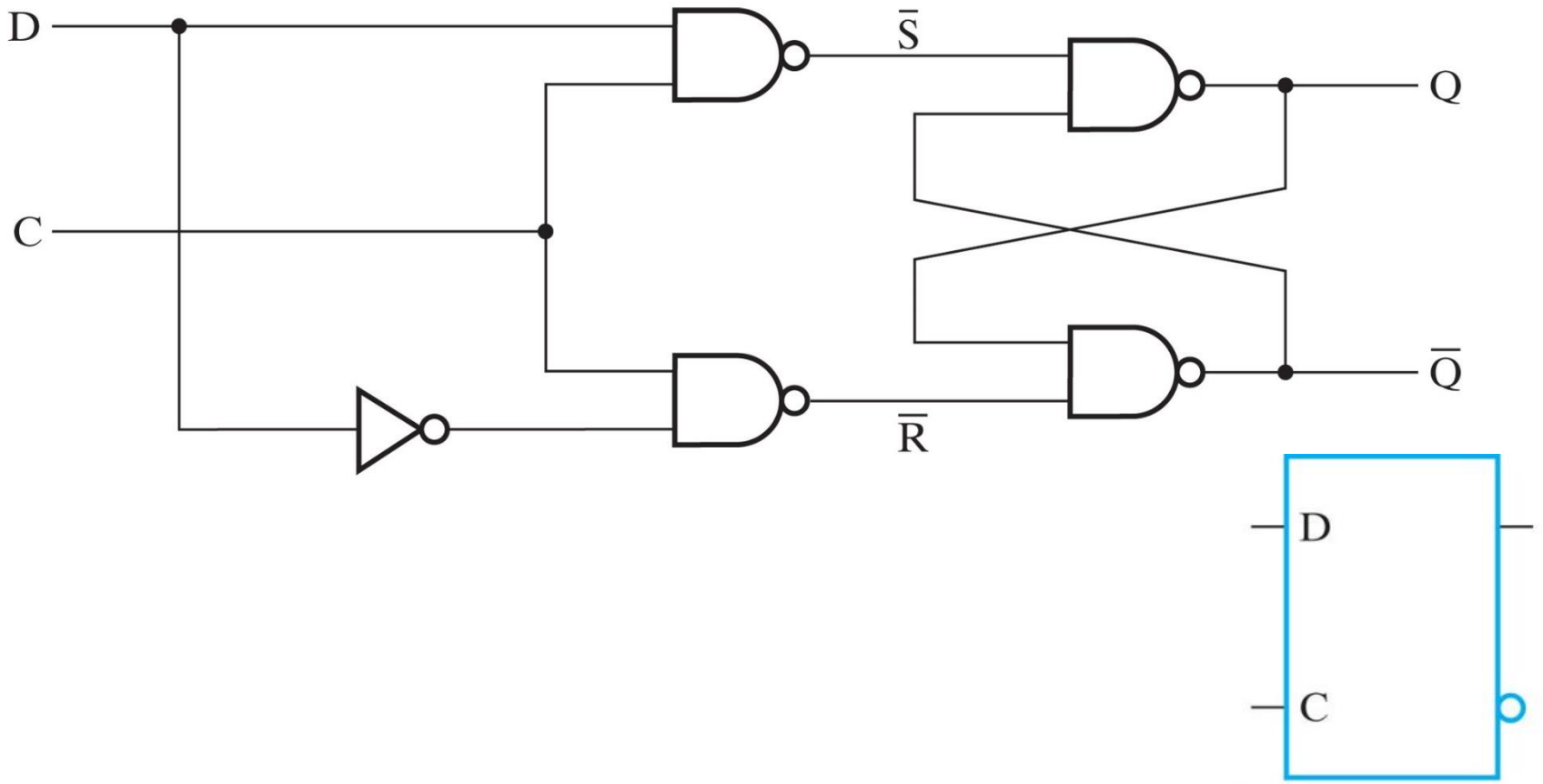


C	S	R	Next state of Q
0	X	X	No change
1	0	0	No change
1	0	1	Q = 0; Reset state
1	1	0	Q = 1; Set state
1	1	1	Undefined

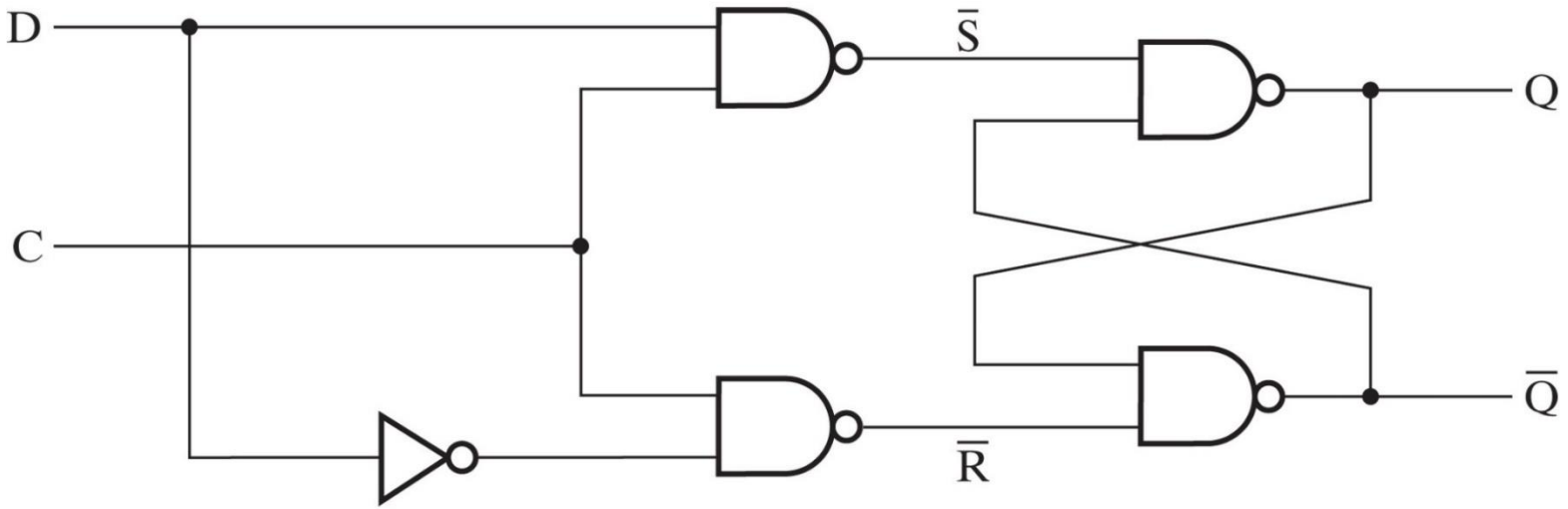
- **C = '0': latch is not enabled** (quiescent state), it maintains the previous state, regardless of the values of S and R inputs
- **C = '1': latch works like an SR latch**
  - **C = '1', S = R = '0'**: **Memory state**
  - **C = '1', S = '0', R = '1'**: **Reset state** => Q = '0'
  - **C = '1',S = '1', R = '0'**: **Set state** =>Q ='1'
  - **C = S = R = '1'**: forbidden state (both outputs go to '1')

D Latch

- The D latch **solves the problem of the forbidden state**, avoiding the condition that $\overline{S}$ and $\overline{R}$ are simultaneously equal to '1'
  - 2 inputs: Data (D), Control (C)



D Latch: Operation and Function Table

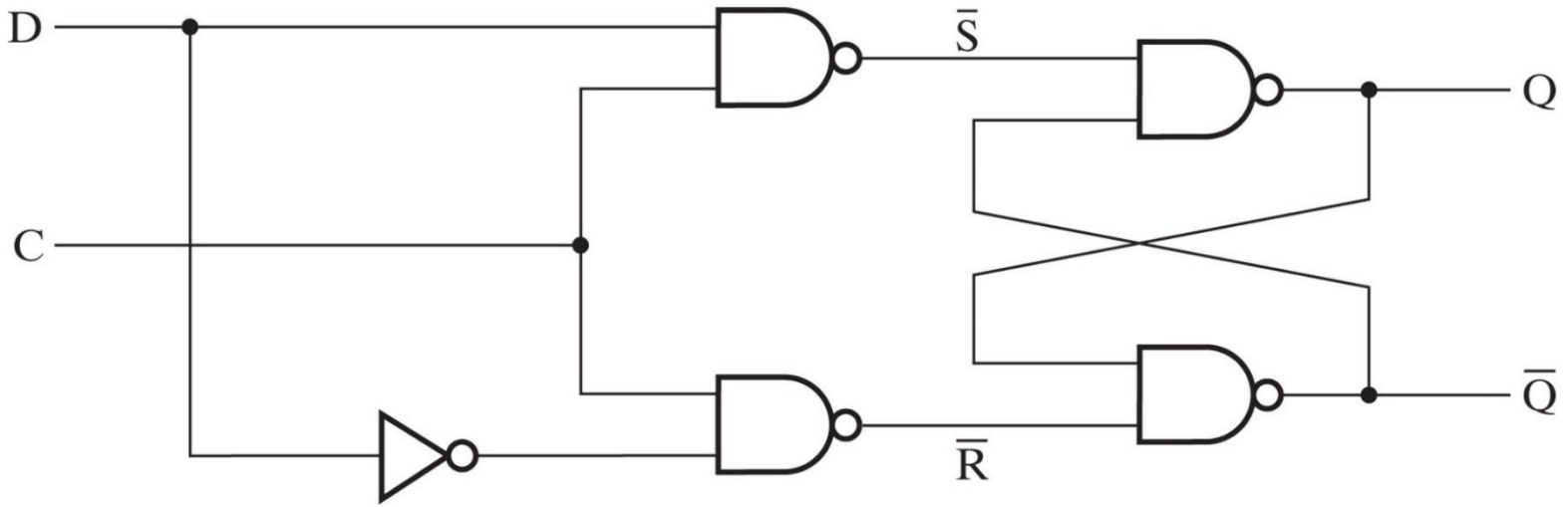


NAND gate: it is enough that one input is '0' for the output to be '1'

C	D	Next state of Q
0	X	No change
1	0	Q = 0; Reset state
1	1	Q = 1; Set state

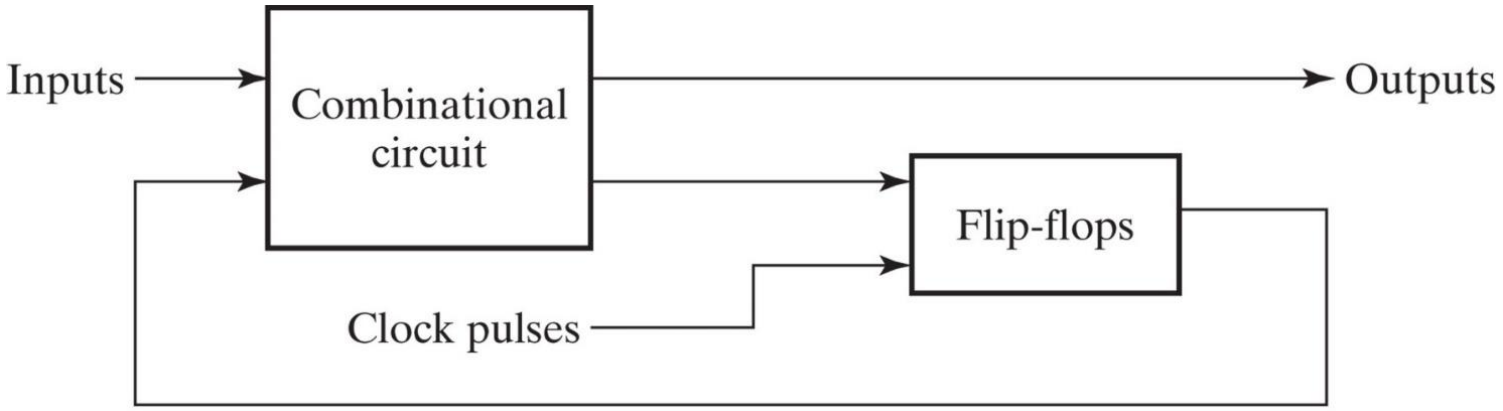
- C = '0': the circuit maintains the previous state, no matter the value of input D
- C = '1': input D is transferred to output Q
  - C = '1', D = '0': reset status => Q = '0'
  - C = '1', D = '1': set status => Q ='1'
- The output follows changes in the data D input, as long as control input C is enabled

Transparency



- A change in the control input induces a change in the latch output, i.e. it acts as a **trigger**
  - We call «**trigger**» a **change in the value of an input that enables changes in the value of the output** in a memory element
- When C = 1, the output of the latch reflects (with a certain delay time) any change in the input data D: in other words, the latch is **transparent**
- The **latch is sensitive to the logic level of the control input**

Features of Flip-flops

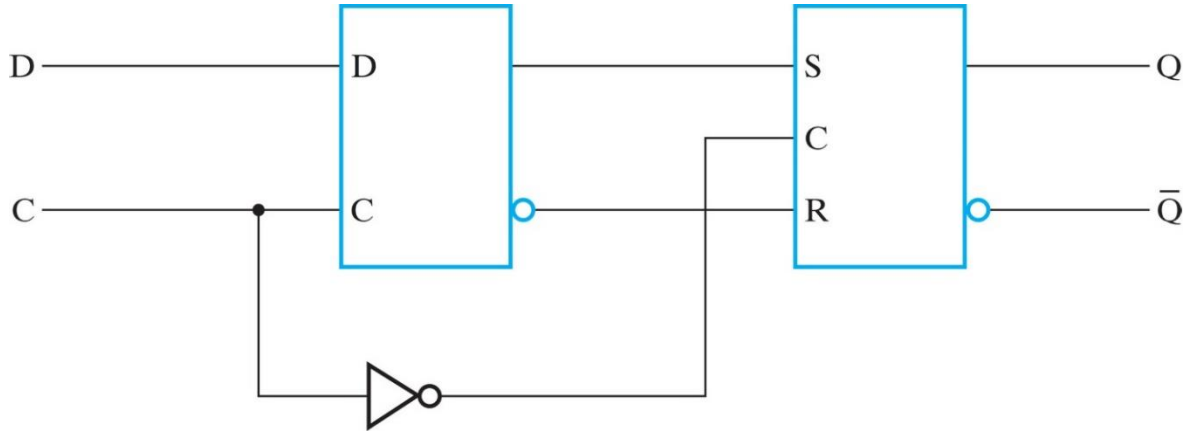


- The presence of the feedback implies that the input of the flip-flop depends in part on its output. To correctly operate in a clocked synchronous circuit, a flip-flop **must NOT be transparent**: the output must not 'see' the changes of the input within the same clock period (but only between two consecutive clock periods, e.g. on the rising edge)
  - If not, we would have multiple unwanted changes in the flip-flop output (resulting in unpredictable situation with state that may keep changing until the clock returns to 0). In other words, we want the new state to depend ONLY on the immediately preceding state

  => Between two consecutive clock edges, a flip-flop must be therefore insensitive to the logic level of its inputs

How to Build a Flip-flop with Latches

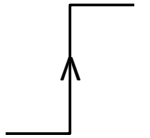
- A common way to create a flip-flop is to connect two latches, with the left latch (**Master**) triggered by the clock and the right latch (**Slave**) by the complemented clock

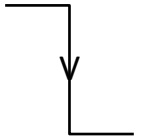


- **This is NOT a transparent structure**: Q can only change at the following clock cycle with respect to a change in D
- Depending on the types of latches we use, we can get
  - Level-sensitive flip-flops (sensitive to clock level, may be '0' or '1')
  - Transition-sensitive flip-flops (sensitive to clock transitions, may be rising edge or falling edge): focus in this course

Edge-Triggered Flip-flops

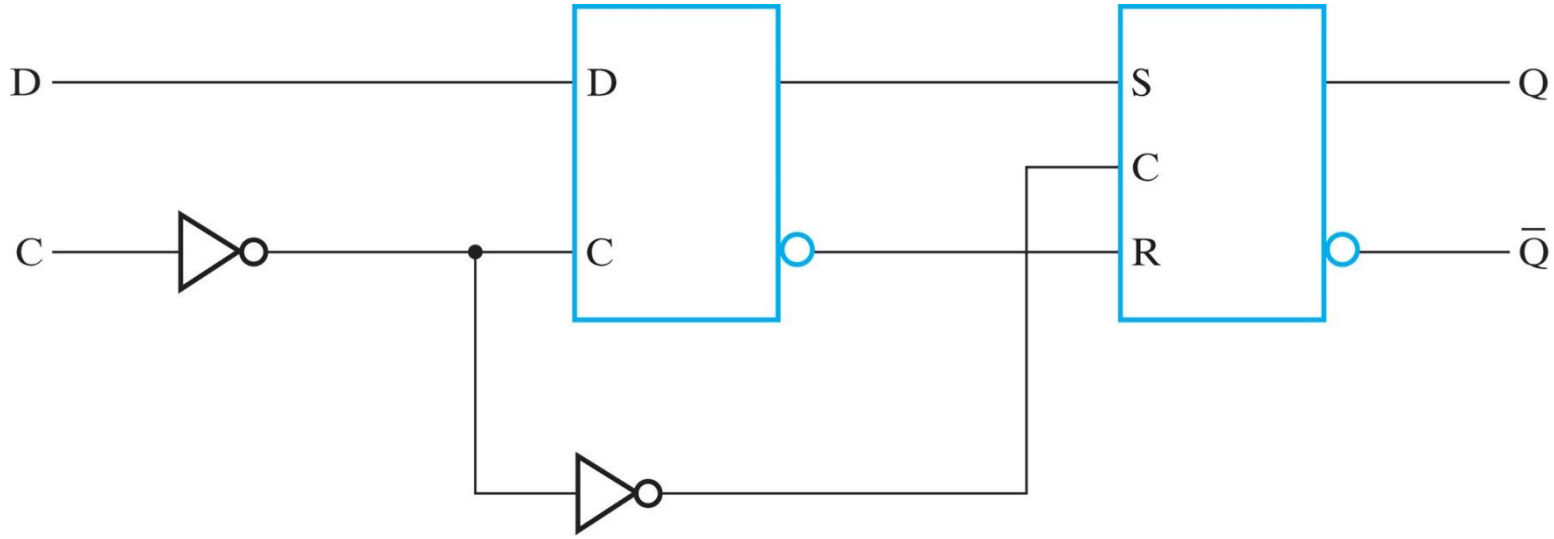
- Edge-triggered flip-flops can **change their status** only in correspondence to a **variation (= edge) of the clock from '0' to '1' or from '1' to '0'**. They are disabled in all the other instants of time (e.g. when the clock is high)
- This type of flip-flop is more used than level-sensitive flip-flop, being faster and easier to design
- Edge-triggered flip-flop are divided into
  - **Positive-Edge-Triggered**: sensitive to the rising edges of the clock
  - **Negative-Edge-Triggered**: sensitive to the falling edges of the clock





Positive Edge-Triggered D Flip-flop

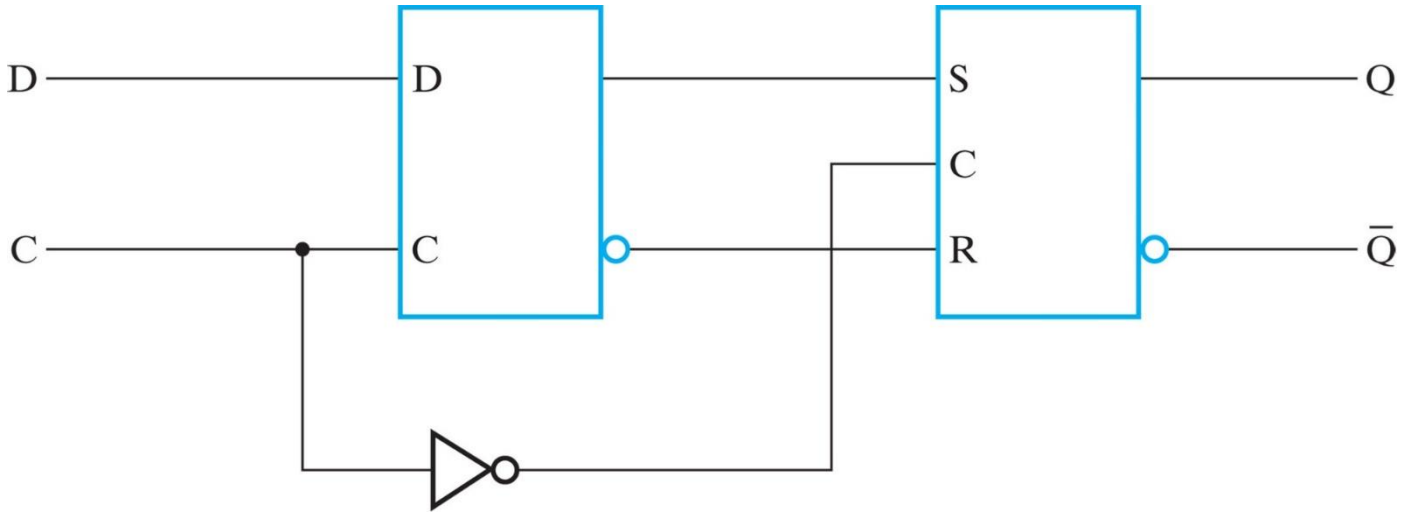
- A Positive Edge-Triggered (PET) flip-flop (sensitive to rising edges) can be obtained by connecting **two latches in cascade**, with the complemented clock driving the control input of the **master** and the direct clock driving the control input of the **slave**



  - The master is enabled when C = 0, the slave is enabled when C = 1
  - As long as C = 0: the master transfers the value of D to the Set input of the slave, while the slave is disabled and maintains the previous state
  - At the rising edge of the clock, the slave is enabled and the last value of D sampled by the master is transferred to output Q

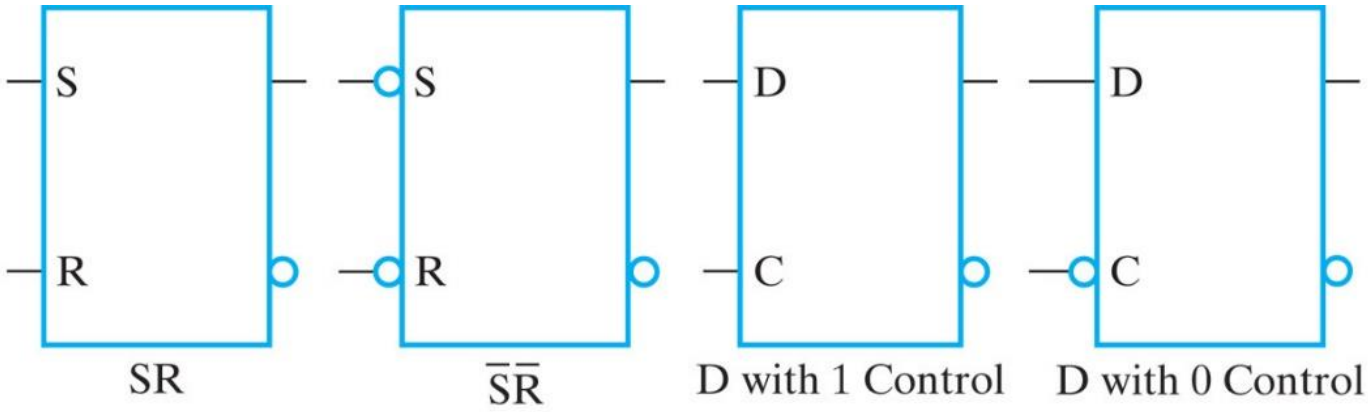
Negative Edge-Triggered D Flip-flop

- A Negative-Edge-Triggered (NET) flip-flop (sensitive to falling edges) can be obtained by connecting **two latches in cascade**, with the clock driving the control input of the **master** and the complemented clock driving the control input of the **slave**

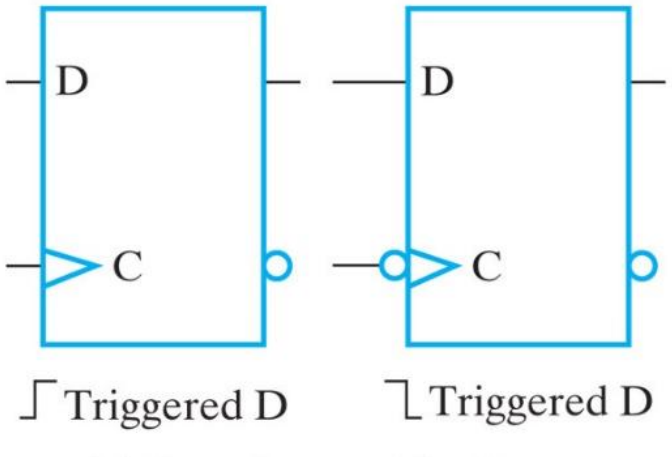


- The master is enabled when C = 1, the slave is enabled when C = 0
- On the rising edge of the clock, the master transfers the value of D to the Set input of the slave, while the slave is disabled and maintains the previous state
- On the falling edge of the clock, the slave is enabled and D is transferred to output Q

Symbols



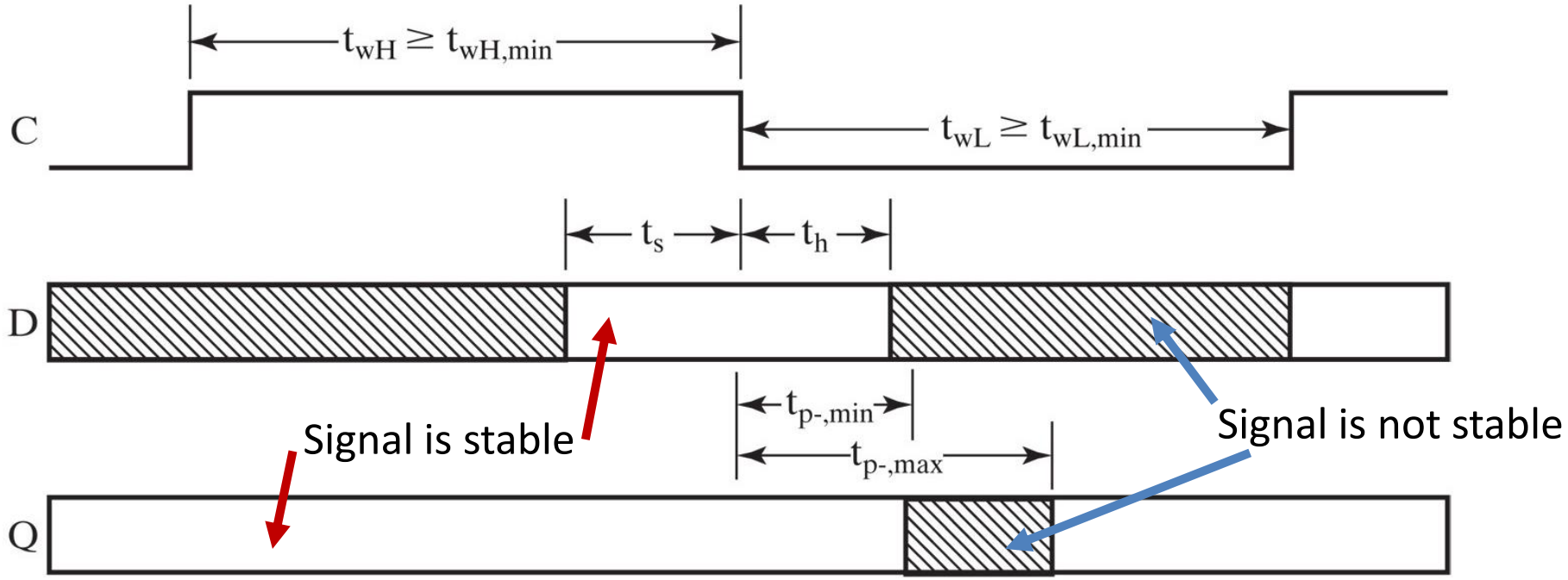
Latches



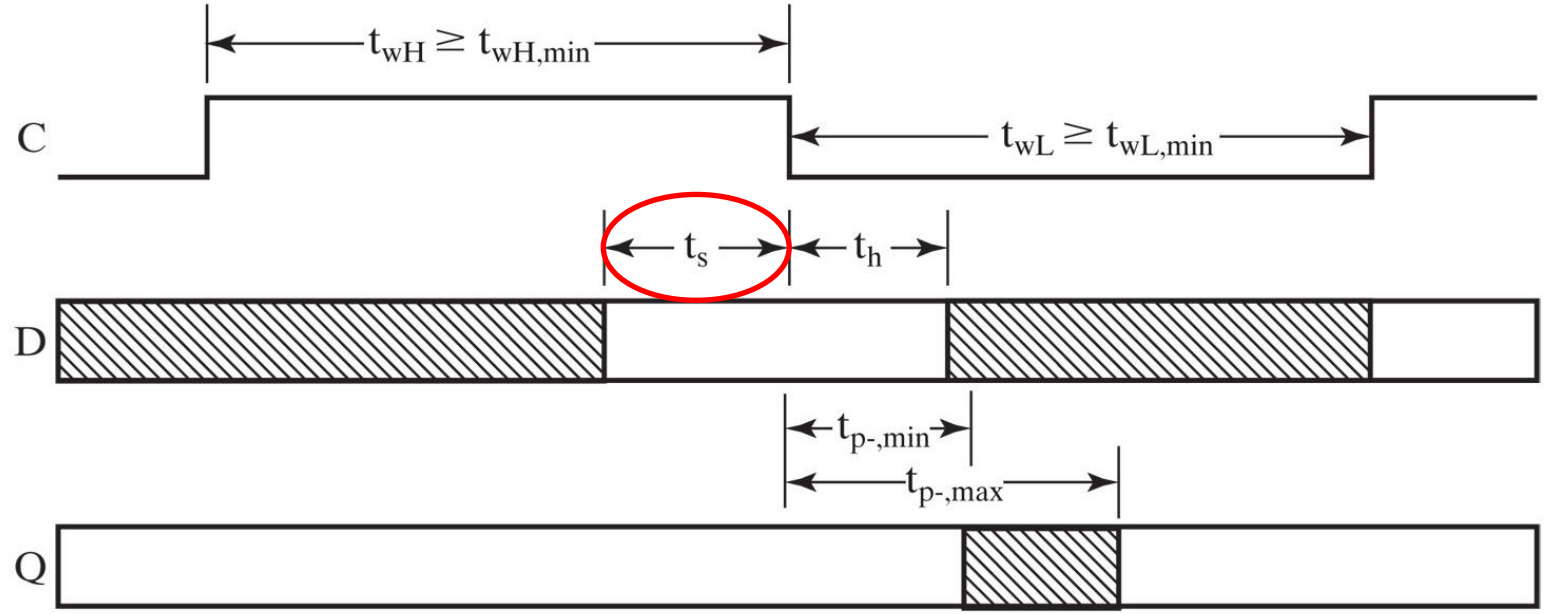
Edge-triggered Flip-flops

Flip-flop Timing

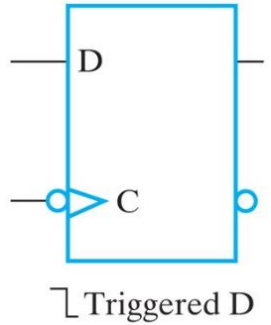
- For the correct functionality of a flip-flop, specific **timing constraints** must be met. Otherwise there is no guarantee that the correct values will be stored



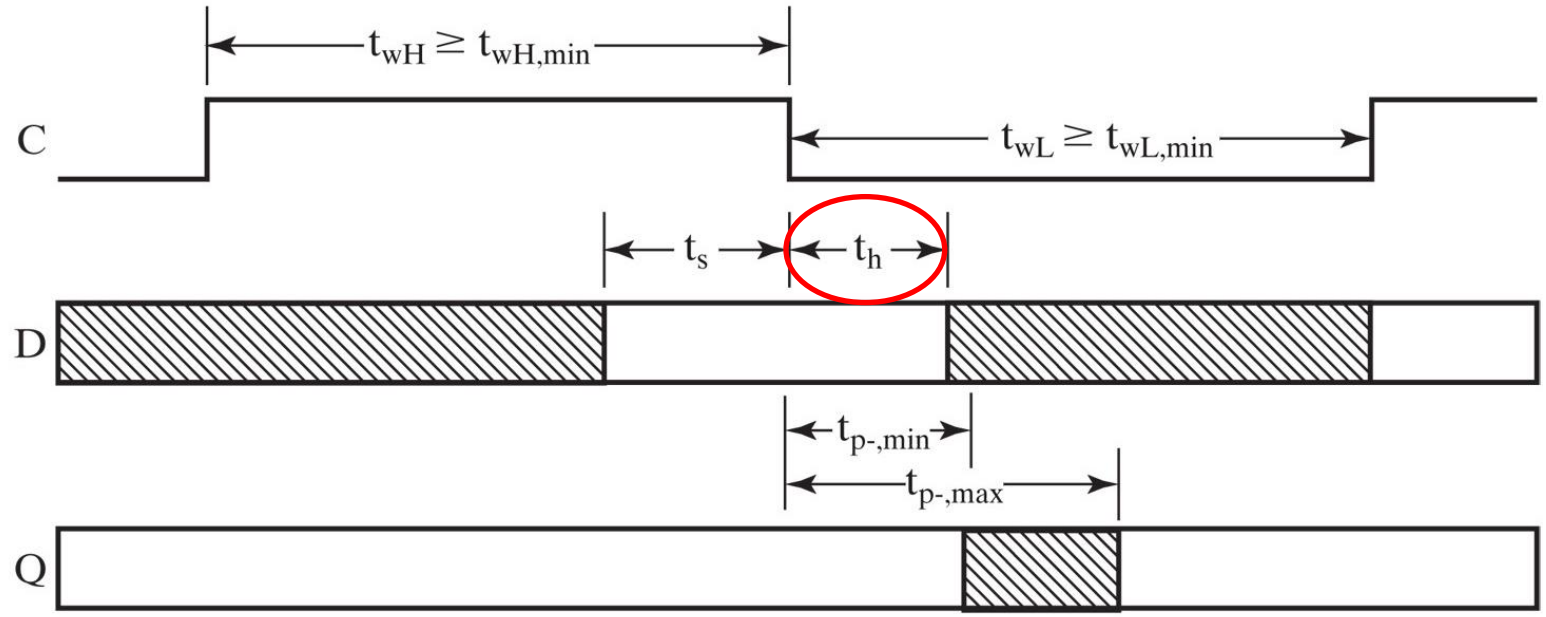
Flip-flop: Set-up Time



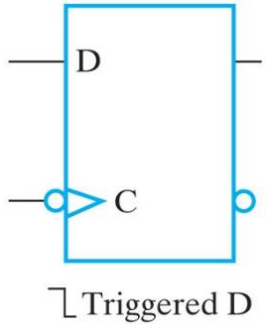
- Set-up time is a **constraint on the input**
- **Set-up time** (t_s): minimum time for which the **inputs must be stable at the desired value, BEFORE the clock edge** that causes a change in the output
  - Otherwise the master may be at an intermediate level when its value is transferred to the slave



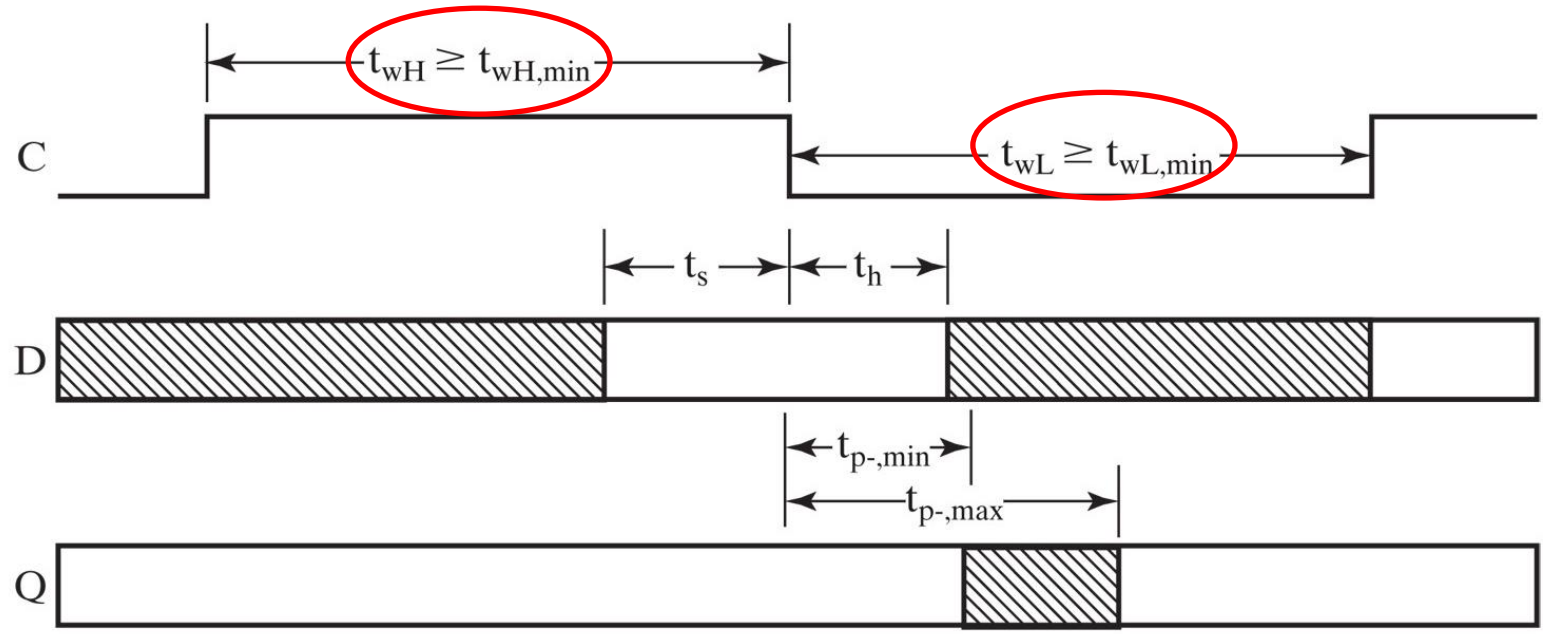
Flip-flop: Hold Time



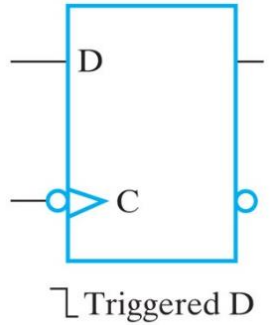
- Hold time is a **constraint on the input**
- **Hold time (t_h)**: minimum time for which the **inputs must be stable at the desired value, AFTER the clock edge** that causes a change in the output
  - Otherwise the master could react to the change in the inputs (as it is transparent) and change the value transferred to the slave



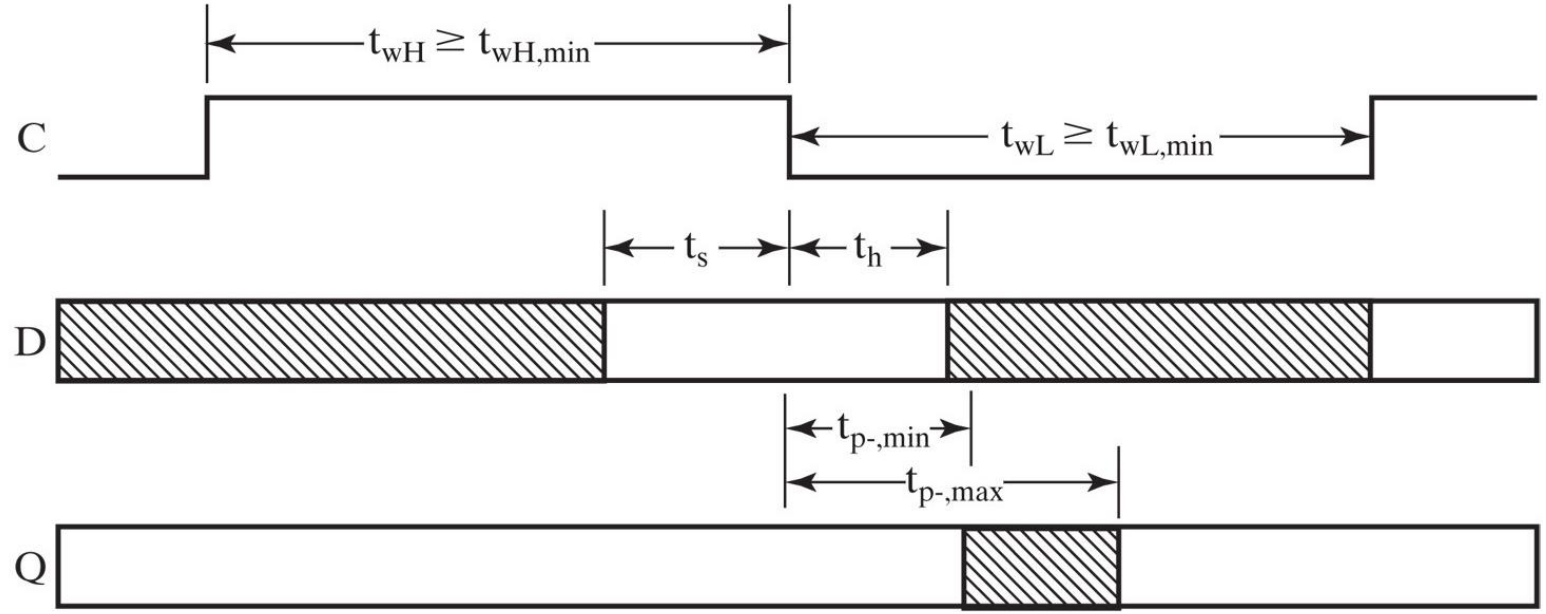
Flip-flop: Clock Pulse Width



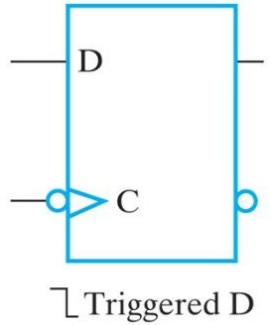
- Constraint on the clock time period
- **Minimum clock pulse width** ($\mathbf{t_w}$): <u>minimum clock period</u> to ensure that the master has the time to correctly capture the input data



Flip-flop: Propagation Time



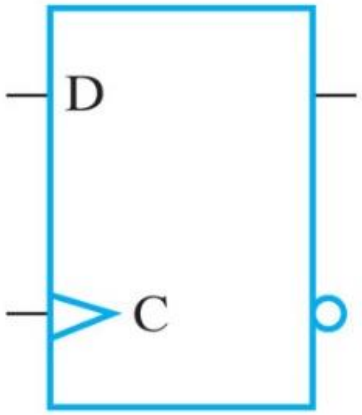
- Propagation time is referred to the **output**
- **Propagation time** (**t_p**): time lapse **between the clock transition acting as a trigger and the time when the output is stable**



NOTE: t_s, t_h, t_p are parameters characterizing the flip-flop at the physical (circuital) level, not at the logical level. They are determined by the implementation technology and by external parameters (temperature, supply voltage, ...)

Positive Edge Triggered Flip-flop

- It is good practice that all the flip-flops within a circuit are of the same type, in order to react in the same way in the presence of the same clock variations
- From here on, unless otherwise specified, we will assume that all flip-flops are **positive-edge-triggered D flip-flops**



Triggered D

A note on Terminology

Linus Torvalds
(Linux)

Adapted from Wikipedia:

- In **2003**, after receiving a discrimination complaint from a county employee, the County of Los Angeles in California asked that manufacturers, suppliers and contractors stop using master and slave terminology on products. Following complaints, the County of Los Angeles issued a statement saying that the decision was "nothing more than a request". Media analytics company Global Language Monitor placed the terms first in their annual list of politically charged language for 2004.
- In **2018**, after a heated debate, developers of the Python programming language replaced the term. The Black Lives Matter movement in the United States sparked renewed discussion and terminology changes in **2020**. Some have argued that the change is superficial and that companies should make real change to support the Black community; some experienced programmers and conservative software engineers criticized the move as political correctness that added unnecessary complications. **Google's developer documentation style guide recommends avoiding the term master in software documentation, especially in combination with slave**. In 2020, GitHub replaced the default master git branch with main.
- Various **replacement terms** for master or slave have been proposed and implemented: primary/secondary, main/ replica or subordinate, initiator/target, requester/responder, controller/device, host/worker o proxy, leader/ follower, director/performer.
- Python switched to **main, parent, and server**; and **worker, child, and helper**, depending on context. The Linux kernel has adopted a similar policy to use more specific terms in new code or documentation. Other projects and standards have used alternative terms since their inception.

Summary

- The outputs of a **sequential system** depend not only on current inputs, but also on their past history (= **state** of the system). A sequential system is obtained connecting a combinational block to **memory elements**
- A **synchronous** sequential circuit is timed by a **clock signal**: state variables can only change in correspondence with certain events in the clock (e.g. rising edge)
- The **latch** is the basic non-clocked sequential element (it is sensitive to the logic level of the input signals)
- The **flip-flop** consists of latches and it is clock regulated (e.g. positive-edge-triggered flip-flops are sensitive to clock rising edges)
- To operate correctly, a flip-flop must meet some timing constraints (**setup time** and **hold time**)

Disclaimer

Figures from *Logic and Computer Design Fundamentals*, Fifth Edition, GE Mano |Kime| Martin