Circuiti sequenziali

Circuiti sequ

ATTENZIONE!

BLU = richiamo teoria

ROSSO = esercizio laboratorio

Circuiti sequenziali

- Richiami
- Circuiti sequenziali
- Memorizzazione dello stato
- Bistabili
- Architetture asincrone e sincrone
- Cancelli (controllo dei segnali in ingresso)
- Cancelli (controllo della propagazione dei segnali in parti diverse del circuito)
- D-type Flip-Flop, registro (PIPO/SISO/SIPO)

Richiami

Riferimenti: slide teoria

In questa serie iniziale di slide discuteremo la differenza tra circuiti combinatori (memoryless) e sequenziali.

In seguito ci concentreremo sul concetto di **Memoria** (e su quello di **stato**). Passeremo poi al concetto di **sincronia**.

In questo laboratorio vedremo: Latch SC (asincrono), Latch SR (sincrono), Latch D (sincrono), Flip-Flop di tipo DT (configurazione master-slave).
Per quanto riguarda Logisim, ci eserciteremo nella creazione di componenti riutilizzabili e nella personalizzazione del loro aspetto.

Circuiti combinatori = circuiti **senza memoria:**

Le uscite al tempo *t* dipendono unicamente dagli ingressi al tempo *t*

$$out_i = f_i\left(in_1, in_2, \cdots, in_M\right) \quad , \quad i = 1 \cdots N$$

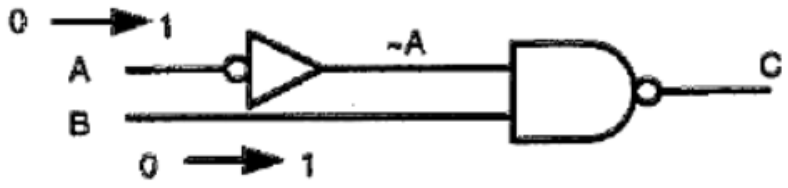
- Ogni uscita ***out$_i$*** è una **funzione logica** degli ingressi ***in$_i$ … in$_M$***
- Indipendenza dal tempo
- È impossibile modificare la "storia" del circuito ➔ non c'è memoria

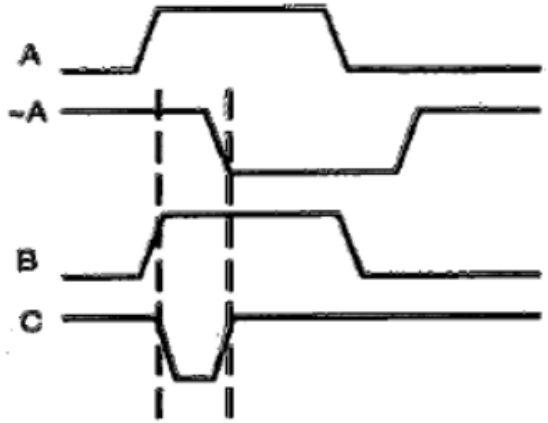
Circuito combinatorio

Al crescere della complessità dei circuiti combinatori, si verificano fenomeni critici:

DATA RACES: *transizioni spurie (glitch) su un segnale che deve rimanere costante*

➔ C'è un ***limite alla complessità*** dei circuiti combinatori!

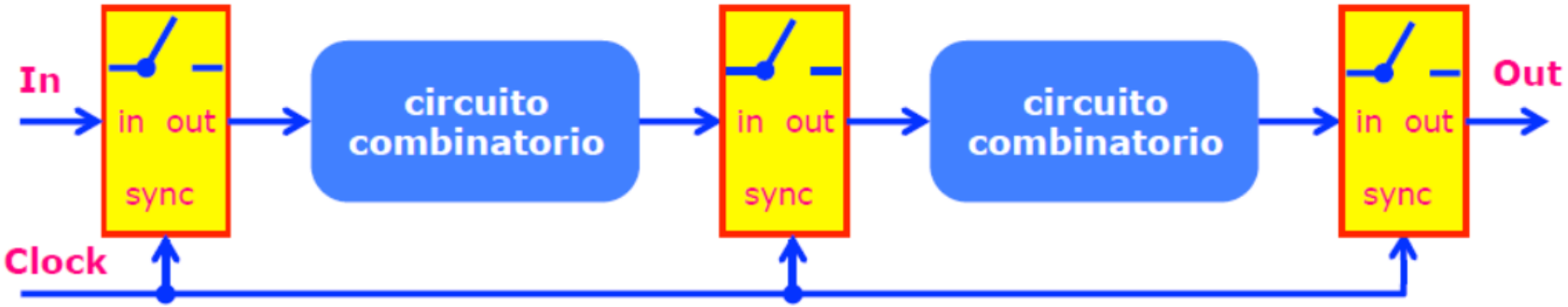




IDEA:

*spezzo il circuito in **SEZIONI** ed inserisco tra le sezioni delle **BARRIERE** al segnale, che APRO solo quando i segnali sono STABILI.*

STRUTTURA ARCHITETTURE SINCRONE



Cancello → Circuito combinatorio → Cancello

- Sincronizzazione: la logica combinatoria deve terminare la propria commutazione in tempo utile
- Dimensionamento del periodo di clock **T**:
  - La commutazione del clock deve avvenire **dopo** che la logica combinatoria abbia terminato tutte le commutazioni
  - Il tempo necessario alla logica combinatoria per commutare dipende dal *cammino critico*

ARCHITETTURE SINCRONE vs ARCHITETTURE ASINCRONE

- **Architettura sincrona:**

  l'elaborazione e propagazione dei segnali è scandita da un **orologio comune a tutto il circuito (clock)**

  - Il Clock regola l'attività dei "cancelli"
  - Il circuito ha il tempo di stabilizzarsi (transitori critici) fino al successivo impulso di Clock

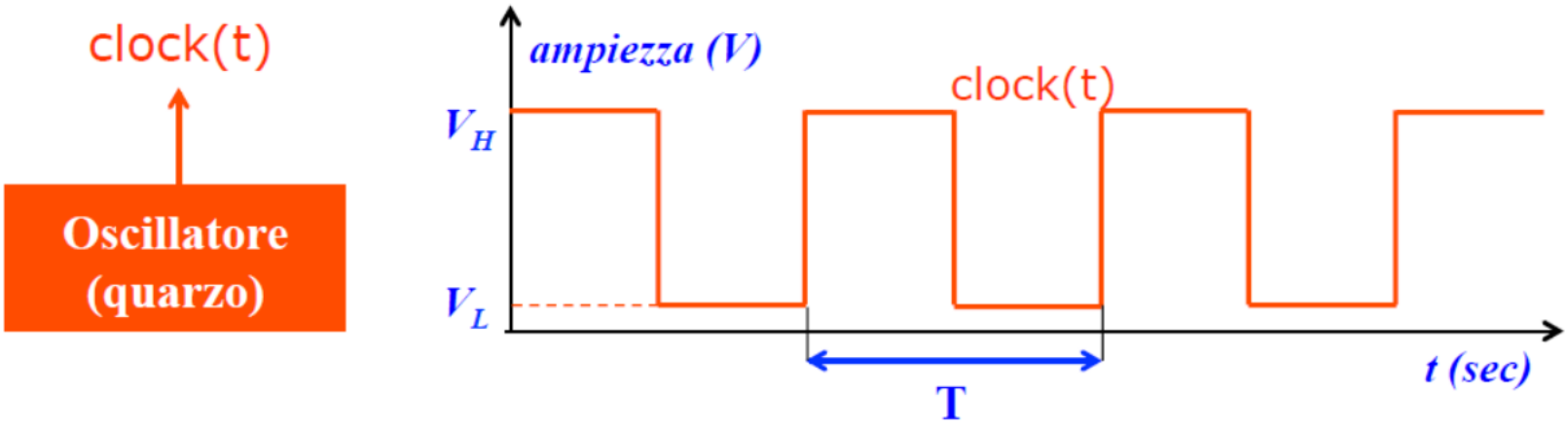
- **Architettura asincrona:**

  l'elaborazione e propagazione dei segnali avviene **in modo incontrollato**, secondo le velocità di reazione dei circuiti

  - Non ci sono cancelli
  - Non devo mai aspettare l'impulso di clock → massima velocità

- **Progettazione sincrona**
  - il controllo dei transitori/cammini critici è limitato alla parte di circuito tra due cancelli (porte di sincronizzazione)
- **Progettazione asincrona**
  - Devo progettare il circuito in modo che nessun transitorio/cammino critico causi problemi → analisi di tutti i transitori critici possibili.

CLOCK



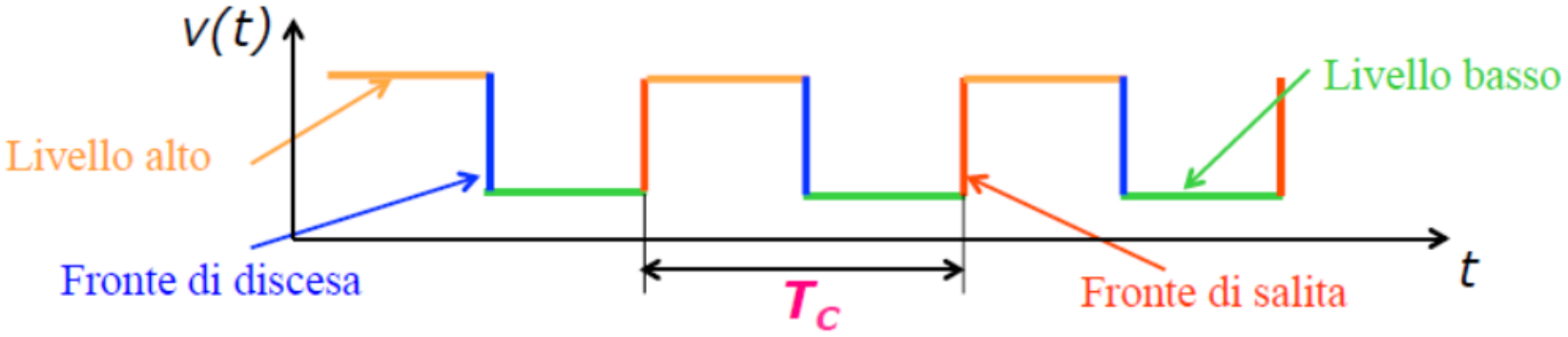
Periodo: **T** [s] durata di 1 ciclo (secondi)

Frequenza: ***f*** [Hz]=[s^{-1}] numero di cicli al secondo

$$T = 1/f$$

- Tempo di salita e discesa trascurabile, rispetto al periodo T.

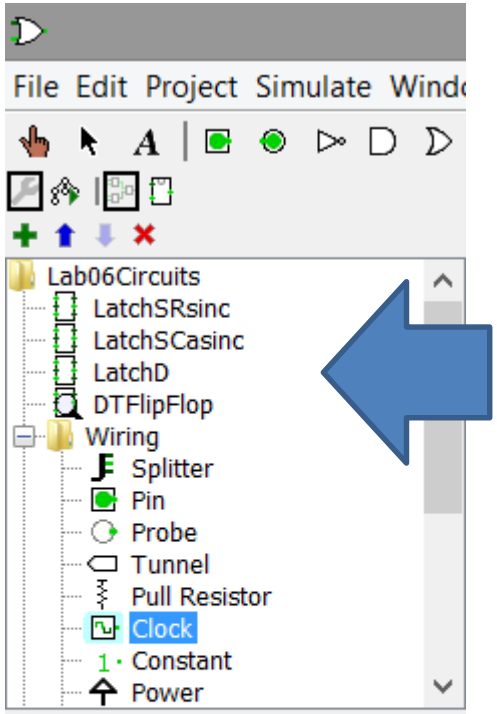
UTILIZZO CLOCK



- **Architettura sensibile ai livelli:**
  - Le variazioni di stato avvengono quando il clock è alto (basso). La parte bassa (alta) del ciclo di clock permette la propagazione tra sottocircuiti, così che i segnali si siano stabilizzati quando il clock cambia livello.

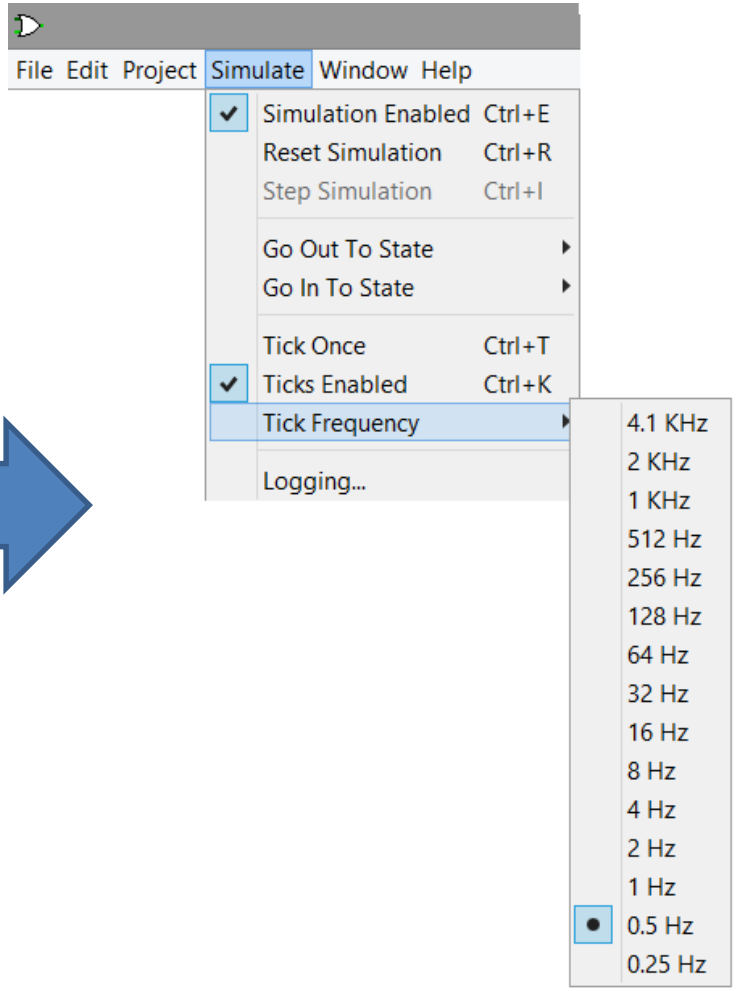
- **Architettura sensibile ai fronti:**
  - Le variazioni di stato avvengono in corrispondenza di un fronte di clock (salita o discesa).

Componente CLOCK in Logisim :



Dove si trova?

Attivazione per la la simulazione…



Tool: Clock

Facing	East
High Duration	1 Tick
Low Duration	1 Tick
Label	
Label Location	West
Label Font	SansSerif Plain 12

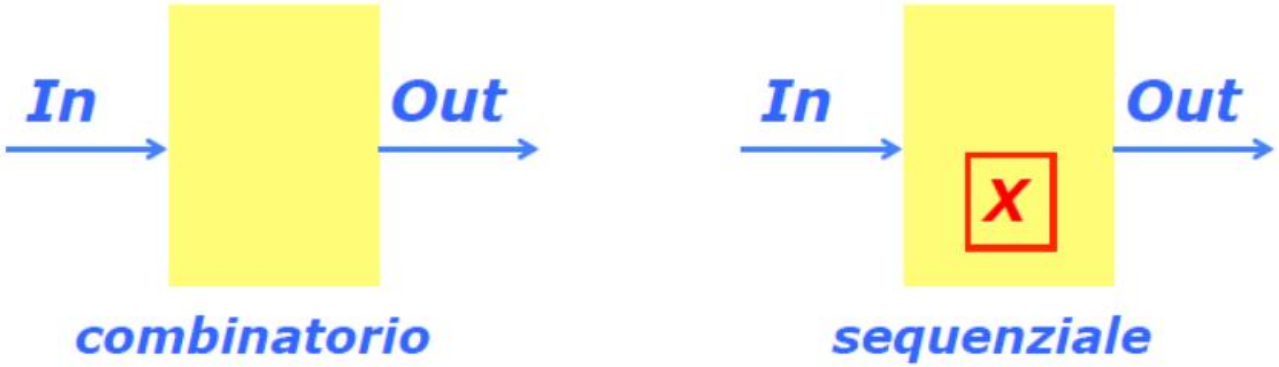
CIRCUITI SEQUENZIALI

- **Circuiti combinatori** = circuiti senza memoria
  - Gli output al tempo *t* dipendono unicamente dagli input al tempo *t*

    $Out = f(In)$
  - Per consentire ad un dispositivo di mantenere le informazioni, sono necessari circuiti con memoria

- **Circuiti sequenziali** = circuiti con memoria (stato)
  - La memoria contiene lo **stato (X)** del sistema:

    $Out = f(In, X)$



BISTABILI :

Circuiti sequenziali sono caratterizzati dal loro **STATO**, esso:

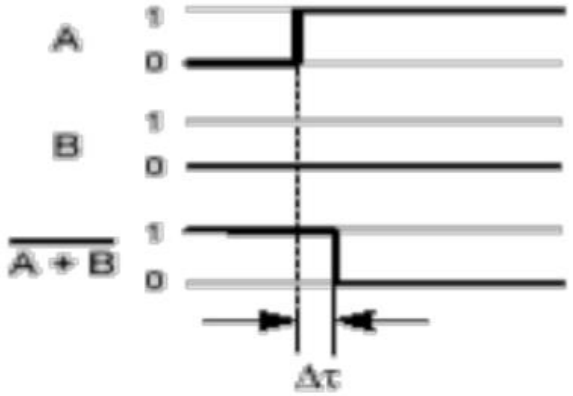
- Riassume il funzionamento negli istanti **precedenti** e **DEVE** essere immagazzinato.
- Necessita di **MEMORIA** (bistabili → registri → memorie)

Elemento base della memoria è il **BISTABILE**, un elemento in grado di mantenere **indefinitamente** il valore di input.
Il suo valore di **USCITA** coincide con lo **STATO**
Memoria : **insieme di bistabili**

Ne esistono diversi tipi:
- Bistabili **non temporizzati** (asincroni) o **temporizzati** (sincroni)
- Bistabili sincroni che commutano su **LIVELLO (LATCH)** o su **FRONTE (flip-flop)**

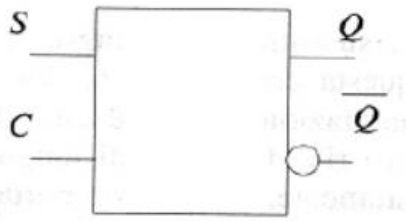
BISTABILI : LATCH Set-Clear (SC) asincrono

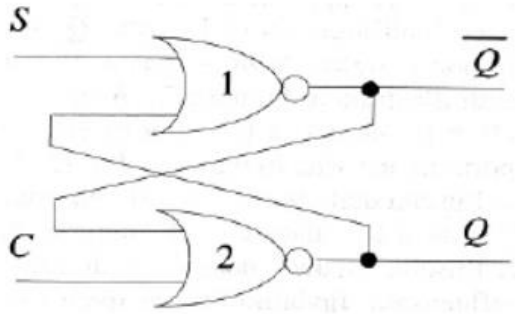
Porta NOR



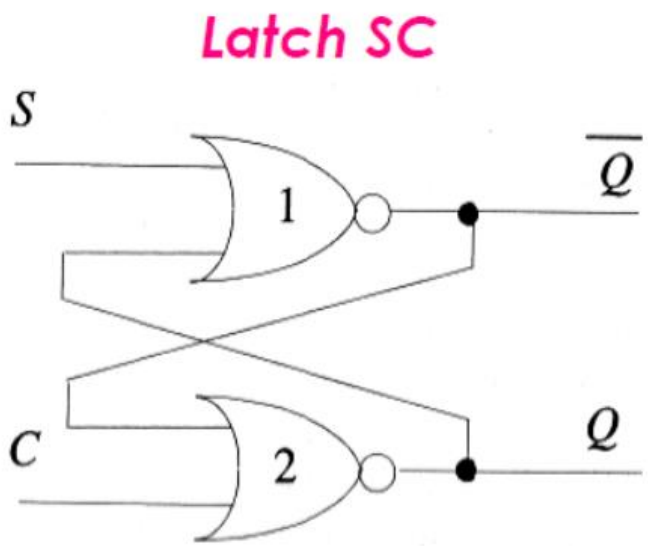
- Una coppia di porte NOR retroazionate può memorizzare un bit!

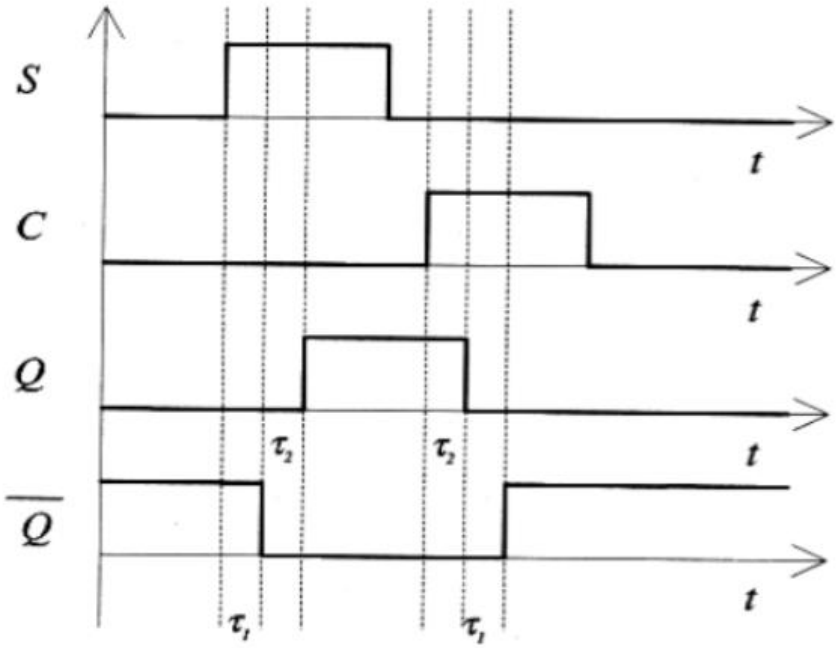
Latch Set-Clear





BISTABILI : LATCH Set-Clear (SC) asincrono



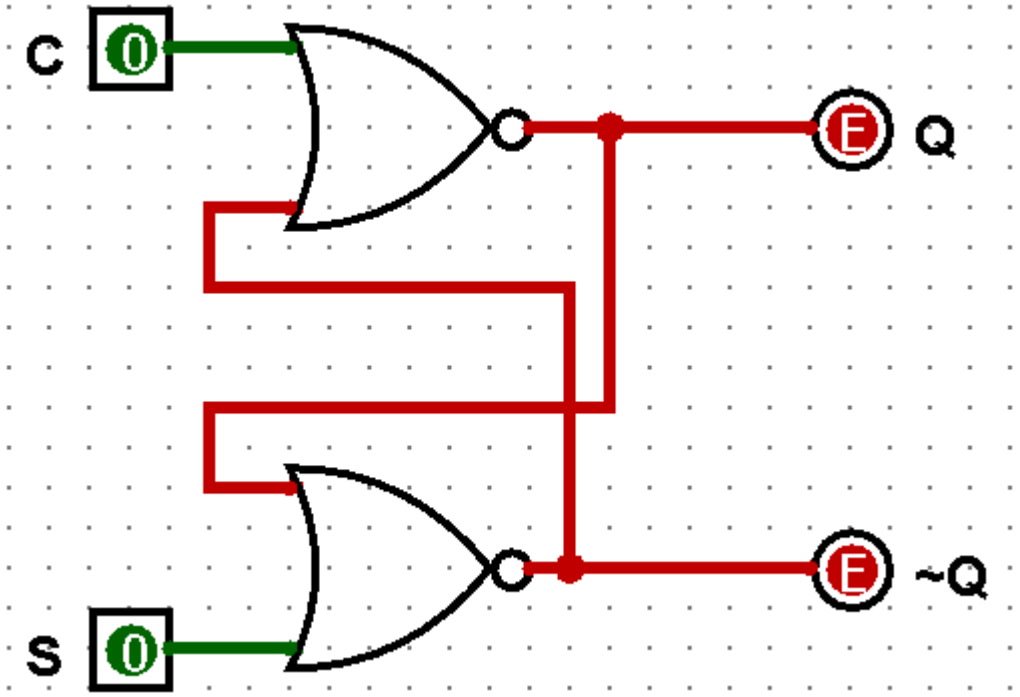


- Funzionamento:
  - Set: C = 0, S → 1 Q → 1 (~Q → 0)
  - Reset: S = 0, C → 1 Q → 0 (~Q → 1)

  - Comportamenti anomali:
  - S = 1 , C: 0→1 Q=~Q=0 (anomalia)

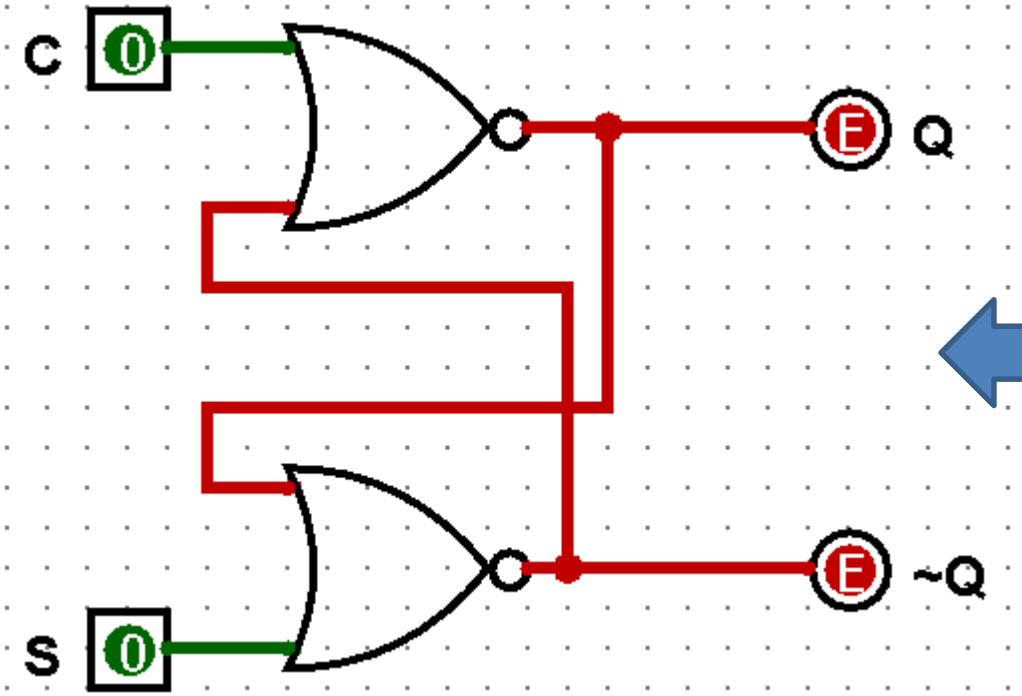
NB

LATCH Set-Clear (SC) asincrono in Logisim



NB: Prima di iniziare i test **Disabilitate** la simulazione, **Resettate** la simulazione e **riabilitate** la simulazione. Otterrete una figura come quella presente in questa slide.

LATCH Set-Clear (SC) asincrono in Logisim

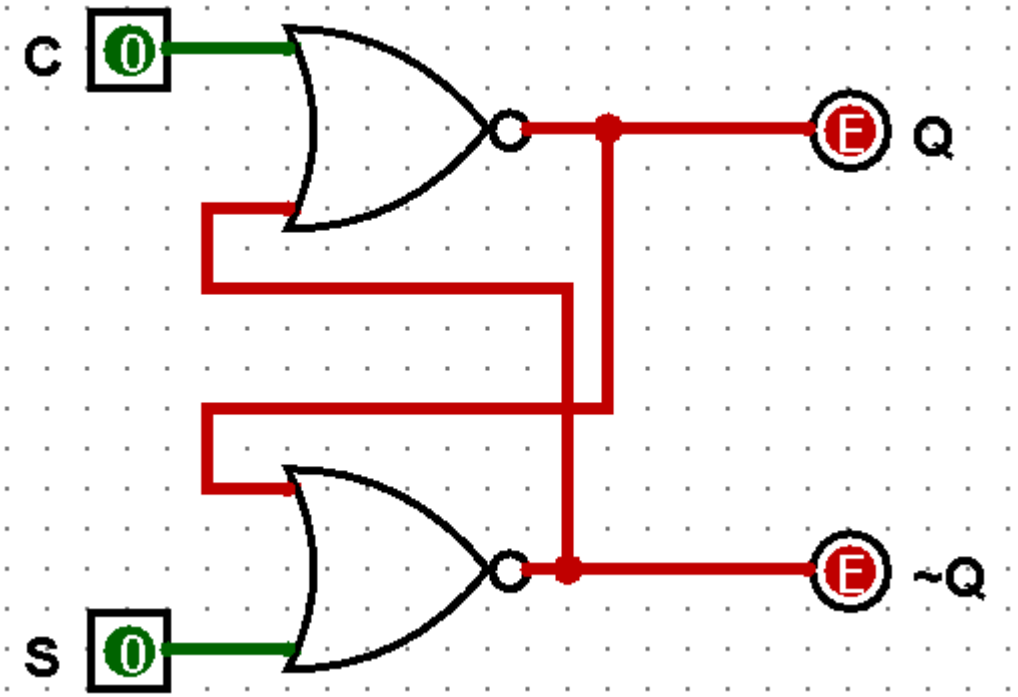


NB: Prima di iniziare i test **Disabilitate** la simulazione, **Resettate** la simulazione e **riabilitate** la simulazione. Otterrete una figura come quella presente in questa slide.

Effettuate queste operazioni IN QUEST'ORDINE.

I : Q diventa 1 quando **S**(et) è impostato a 1
II : Q «ricorda» 1 quando S è reimpostato a 0
III: Q ritorna a 0 quando **C**(lear) è impostato a 1
IV: Q resta 0 quando **C**(lear) è reimpostato a 0

LATCH Set-Clear (SC) asincrono in Logisim

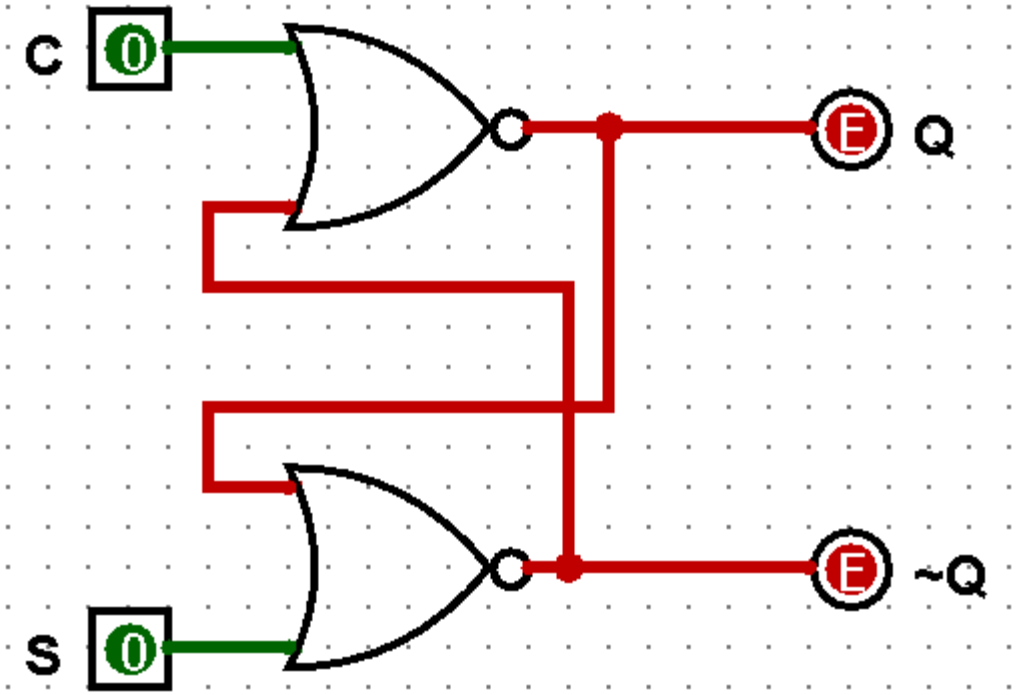


MEMORIA (dello stato)

Latch SC asincrono
Memorizza **lo stato** di 1 bit

I : Q diventa 1 quando **S**(et) è impostato a 1
II : Q «ricorda» **1** quando **S** è reimpostato a 0
III: Q ritorna a 0 quando **C**(lear) è impostato a 1
IV: Q «ricorda» **0** quando **C** è reimpostato a 0

LATCH Set-Clear (SC) asincrono in Logisim



NOTE:

Il circuito è **ASINCRONO** (non c'è clock)

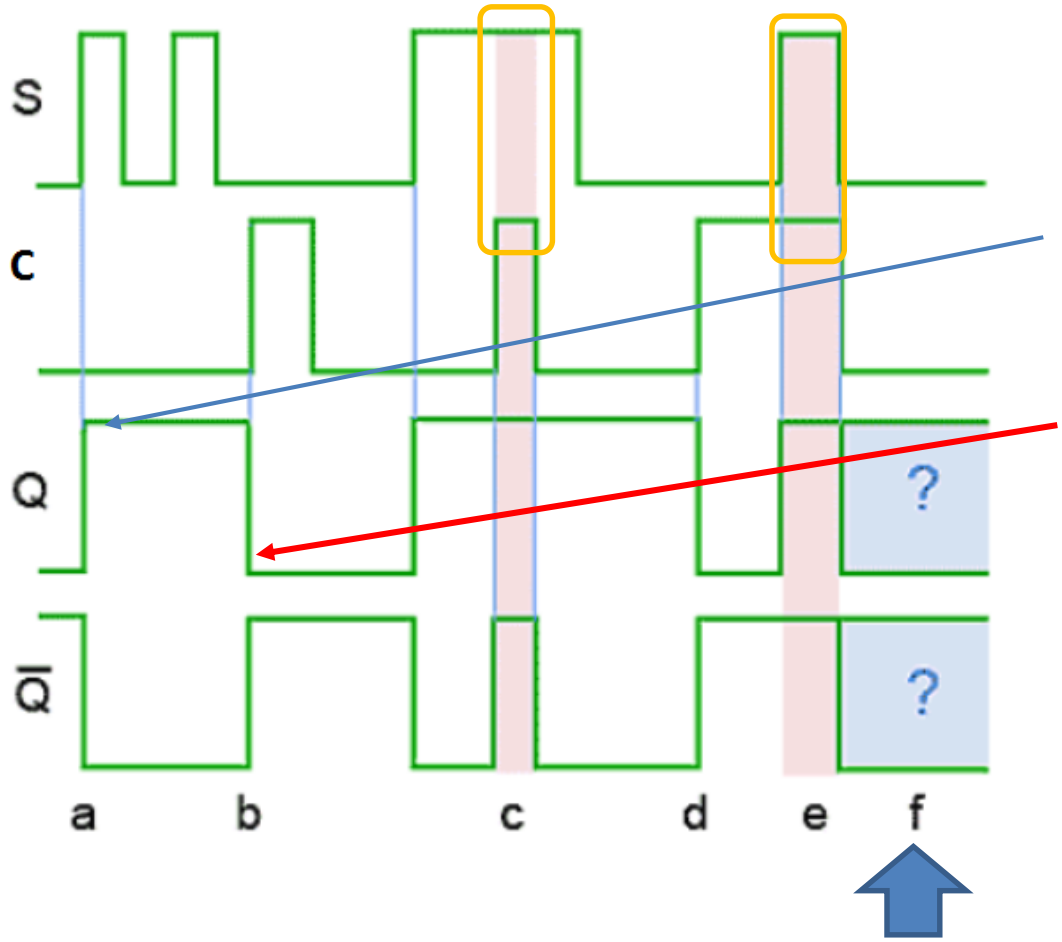
C (clear) annulla lo stato

S (Set) resetta lo stato

Il tempo di transizione totale è **3** :

es. **C → Q , Q → !Q**

LATCH Set-Clear (SC) asincrono in Logisim



OUTPUT INDETERMINATO IN f

Come cambiano Q e !Q al variare di S e C ?

(a) S va in stato «alto» e imposta Q che rimane invariato fino all'istante b

(b) S è «basso» e C diventa «alto» resettando (clear) Q

(c) Sia S che C sono alti e questo causa uno **stato non ammesso** in cui sia Q che !Q sono «alti» . Dopo l'istante c Q rimane «alto» fino a d

(d) C diventa «alto» resettando Q

(e) E' un'altra **configurazione non ammessa** in cui Q=!Q=«alto»

LATCH Set-Clear (SC) asincrono

Tabella delle transizioni: $Q^* = f(Q, I) = f(Q, S, C)$

Q: valore dell'uscita attuale: stato **corrente**

Q*: uscita al tempo successivo: stato **prossimo**

$$Q^* = f(Q, I)$$

Q \ I	SC = 00	SC = 01	SC = 11	SC = 10
0	0	0	X	1
1	1	0	X	1

↑ $Q^* = Q$ ↑ Clear / Reset ↑ SET

LATCH Set-Clear (SC) asincrono : tabella transizioni

- Tabella delle transizioni ***f*** :

$$Q^* = f(I, Q)$$

Q \ SC	SC=00	SC=01	SC=11	SC=10
Q=0	0	0	X	1
Q=1	1	0	X	1

S	C	Q	Q*
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	X
1	1	1	X

NB

- Considerando lo stato Q come ingresso ottengo la tabella delle verità di Q*:

LATCH Set-Clear (SC) asincrono : tabella verità Q*

S	C	Q	$Q*$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	X
1	1	1	X

S = 1
C = 0 → **1**

Q = !Q = 0 (ANOMALIA)

Queste configurazioni non sono ammesse Poiché renderebbero indeterministico Q* (lo stato prossimo) del circuito.

E' possibile impostarne arbitrariamente i valori per semplificare le funzioni

LATCH Set-Clear (SC) asincrono : Esercizi

ESERCIZIO 1:

Si derivi la SOP e la si semplifichi, implementandone il circuito corrispondente. Si assuma prima **X = 0** e poi X = 1 per le due uscite indeterminate della tabella di transizione (tabella di verità di Q*, prossimo stato).

SOP, con X=0

$$
\begin{aligned}
Q* &= \neg S\neg CQ + S\neg C\neg Q + S\neg CQ \\
&= \neg S\neg CQ + S\neg C(\neg Q + Q) && \text{(Raccoglimento)} \\
&= \boxed{\neg S\neg CQ} + \boxed{S\neg C} && \text{(Inverso)}
\end{aligned}
$$

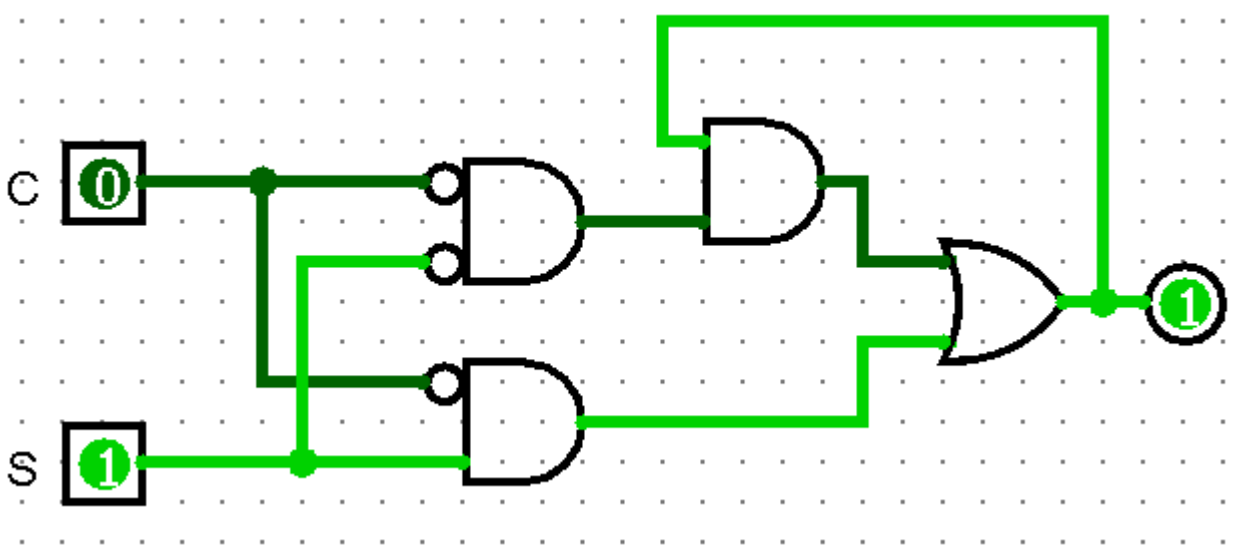
Status quo

Rivoluzione

LATCH Set-Clear (SC) asincrono : Esercizi

ESERCIZIO 1:
Si derivi la SOP e la si semplifichi, implementandone il circuito corrispondente. Si assuma prima **X = 0** e poi X = 1 per le due uscite indeterminate della tabella di transizione (tabella di verità di Q*, prossimo stato).

X = 0



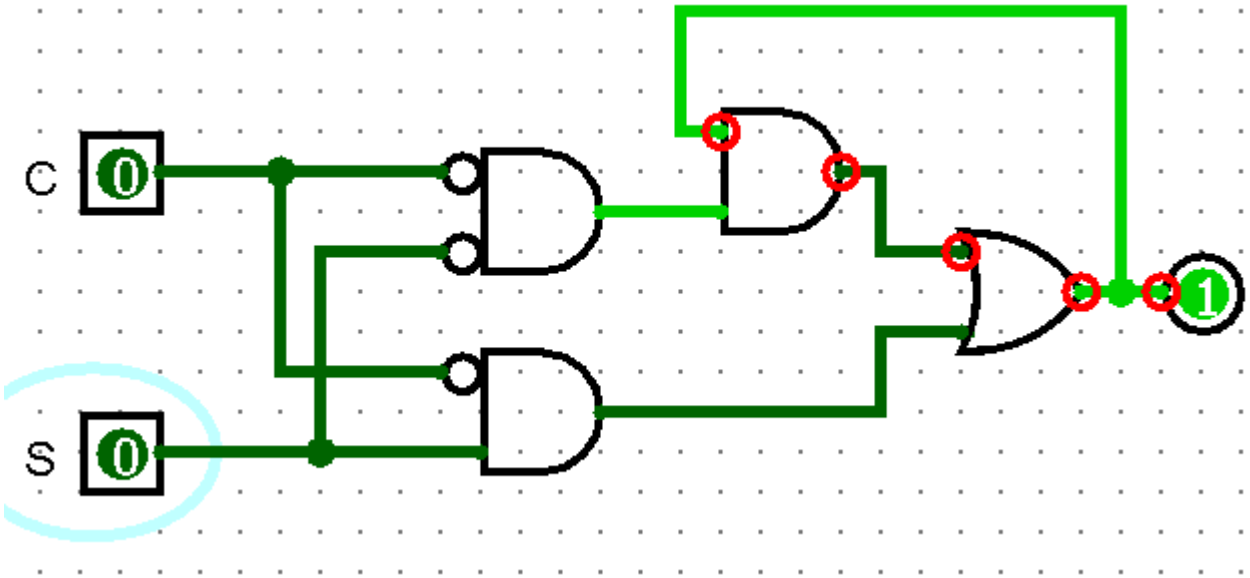
COSA SUCCEDE SE LO SIMULIAMO?

LATCH Set-Clear (SC) asincrono : Esercizi

ESERCIZIO 1:

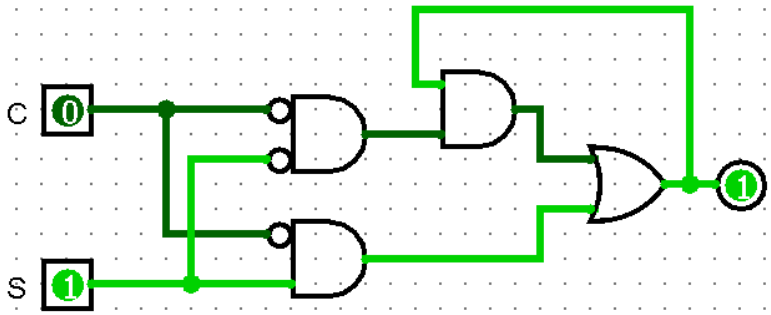
Si derivi la SOP e la si semplifichi, implementandone il circuito corrispondente. Si assuma prima **X = 0** e poi X = 1 per le due uscite indeterminate della tabella di transizione (tabella di verità di Q*, prossimo stato).

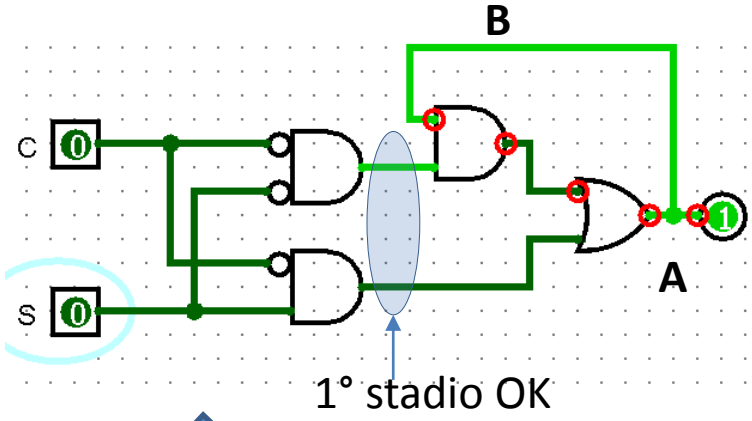
X = 0



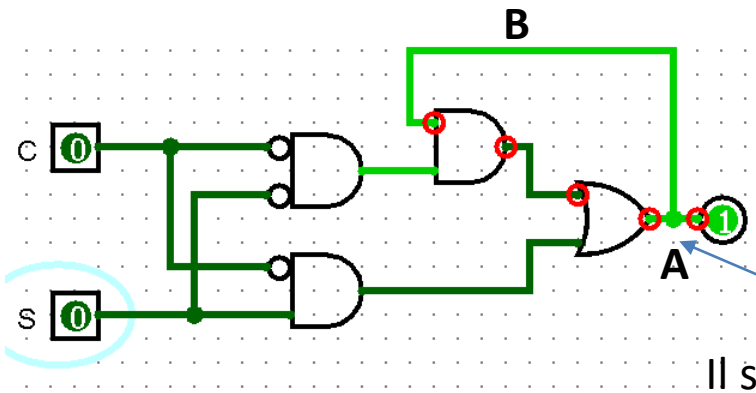
OSCILLAZIONE APPARENTE

LATCH Set-Clear (SC) asincrono





Nella stessa iterazione in qui Logisim determina che A è 0 determina anche che B è 1 … nell'iterazione successiva Logisim determina che A è 1 ma anche che B è 0… OSCILLA ALL'INFINITO!

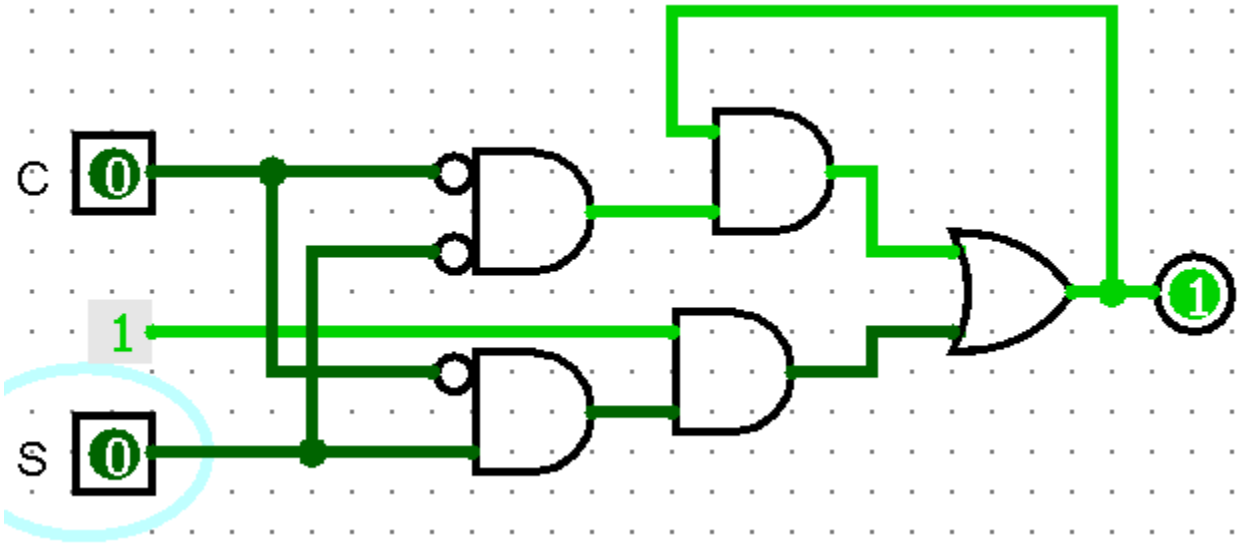


OSCILLAZIONE APPARENTE

Il segnale A dovrebbe essere 1 ma a causa di diversi ritardi in questo momento è 0

LATCH Set-Clear (SC) asincrono

SOLUZIONE PROBLEMA OSCILLAZIONE APPARENTE

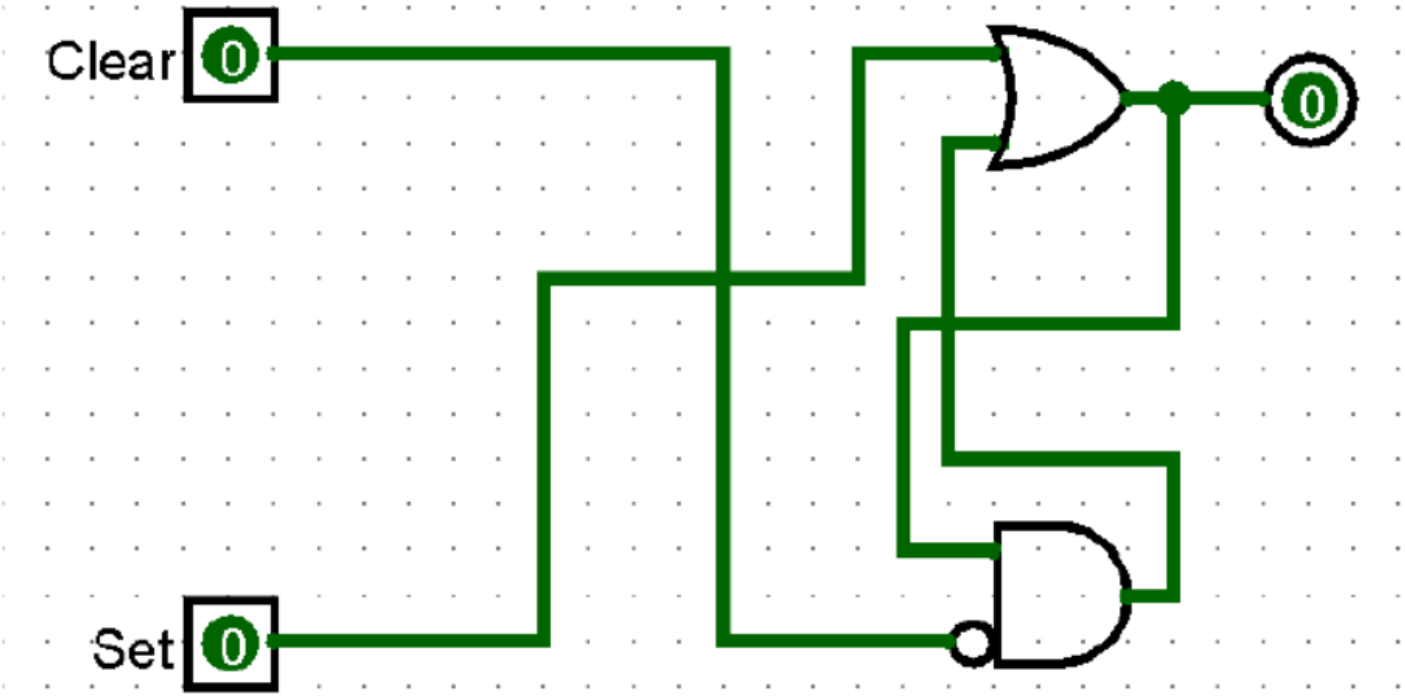


Perché non usare S al posto di un input costante aggiuntivo? (provate!)

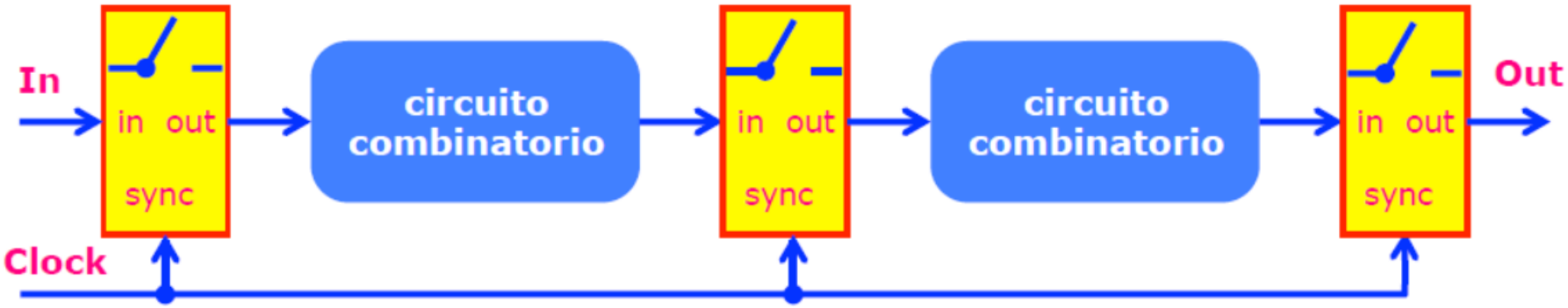
LATCH Set-Clear (SC) asincrono

SOP, con X = 1

$$Q* = \quad \neg CQ + S$$



Architetture sincrone



Cancello → Circuito combinatorio → Cancello

Domande:

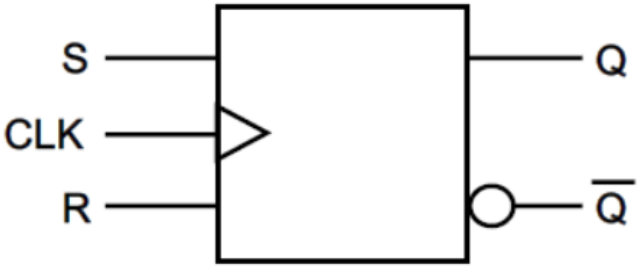
Come lo realizzo il «cancello»?
Come agisce il cancello? Fa passare i segnali in input SOLO SE il clock è in livello «alto»
E' possibile rendere sincrono Il circuito che abbiamo appena visto (LATCH SC **asincrono**)?

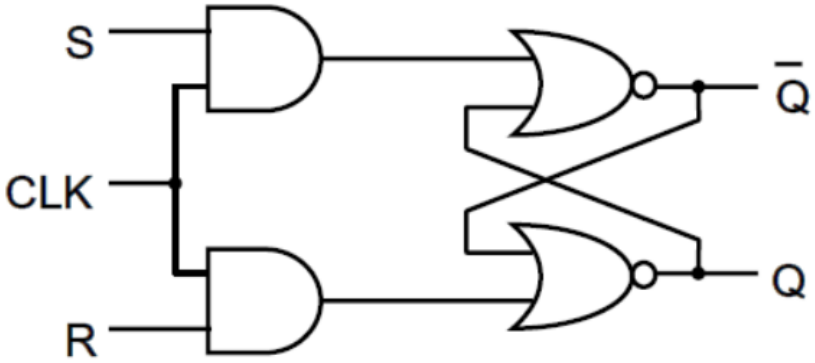
LATCH Set-Reset (SR) sincrono

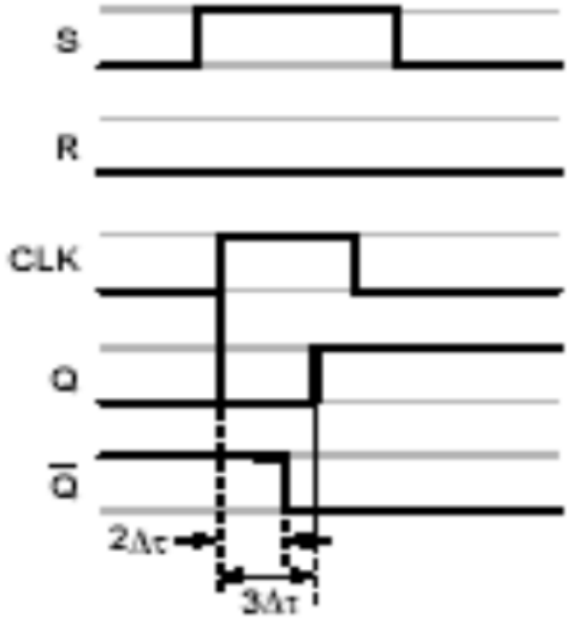
Latch Set-Reset (SR) sincrono

Struttura: Latch SR + Porte AND tra il clock e gli ingressi.

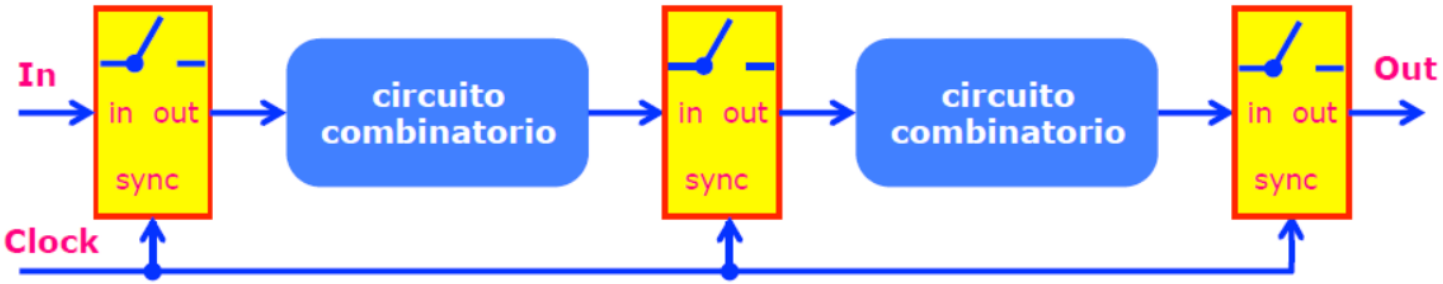
- Solo quando il clock è alto i "cancelli" (porte AND) fanno passare gli input → LATCH



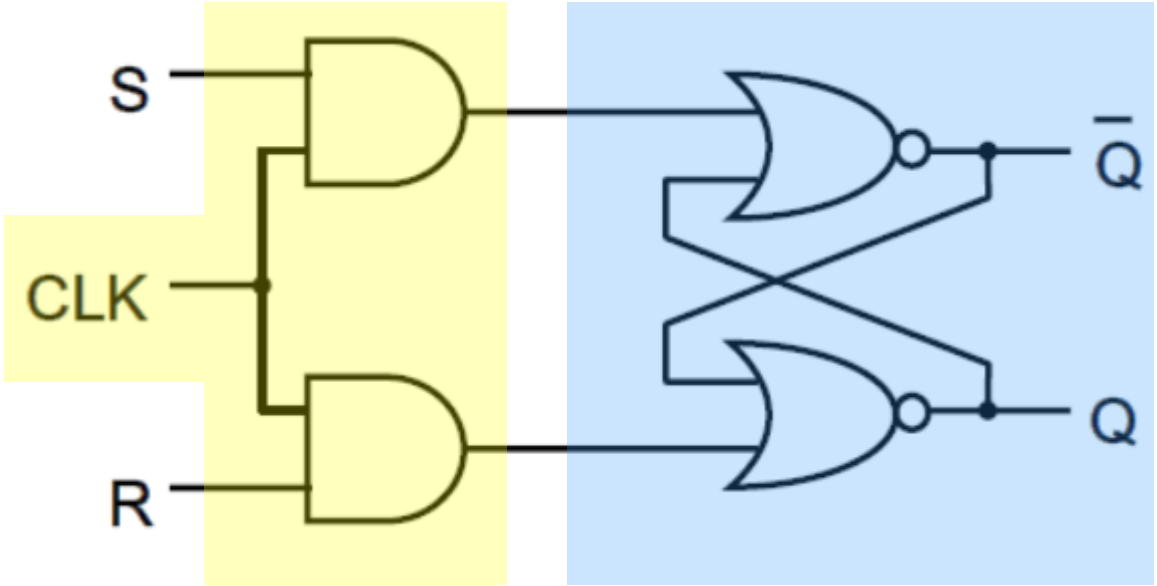




LATCH Set-Reset (SR) sincrono



Cancello → Circuito combinatorio → Cancello



NOTE:

Utili nei casi in cui ho diversi componenti e voglio che essi siano posti tutti sotto il controllo del <u>medesimo</u> clock (es. memorizzazione simultanea dello stato di 8 bit)

LATCH Set-Reset (SR) sincrono

ESERCIZIO 2:

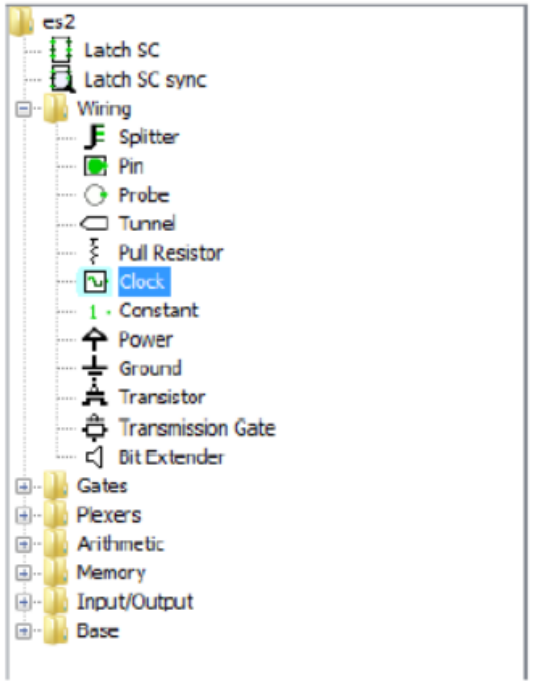
Si aggiunga un clock (freq. 0.5 Hz) e 2 porte AND al circuito LATCH SC asincrono per ottenere un circuito LATCH SR sincrono.

Architetture degli Elaboratori e delle Reti I

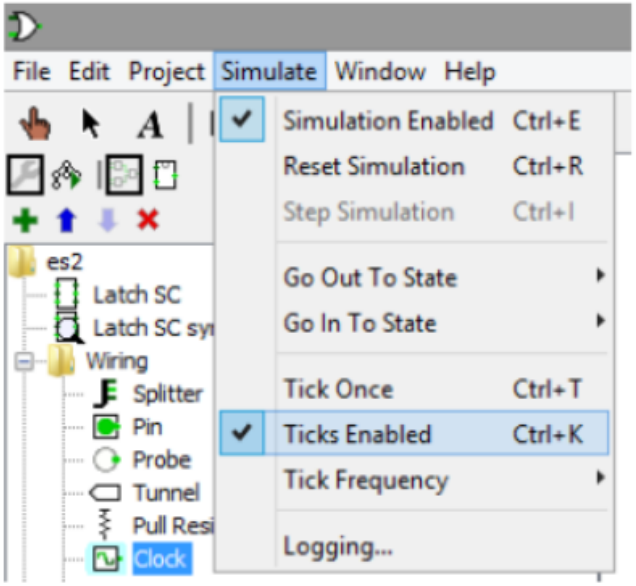
Laboratorio – linea 2 (G-Z)

6

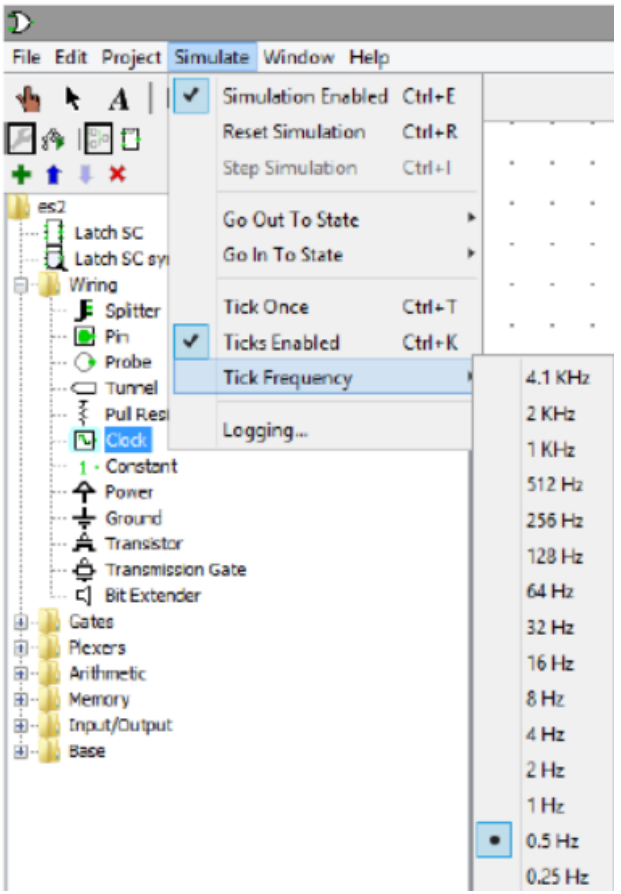
Per aggiungere il clock



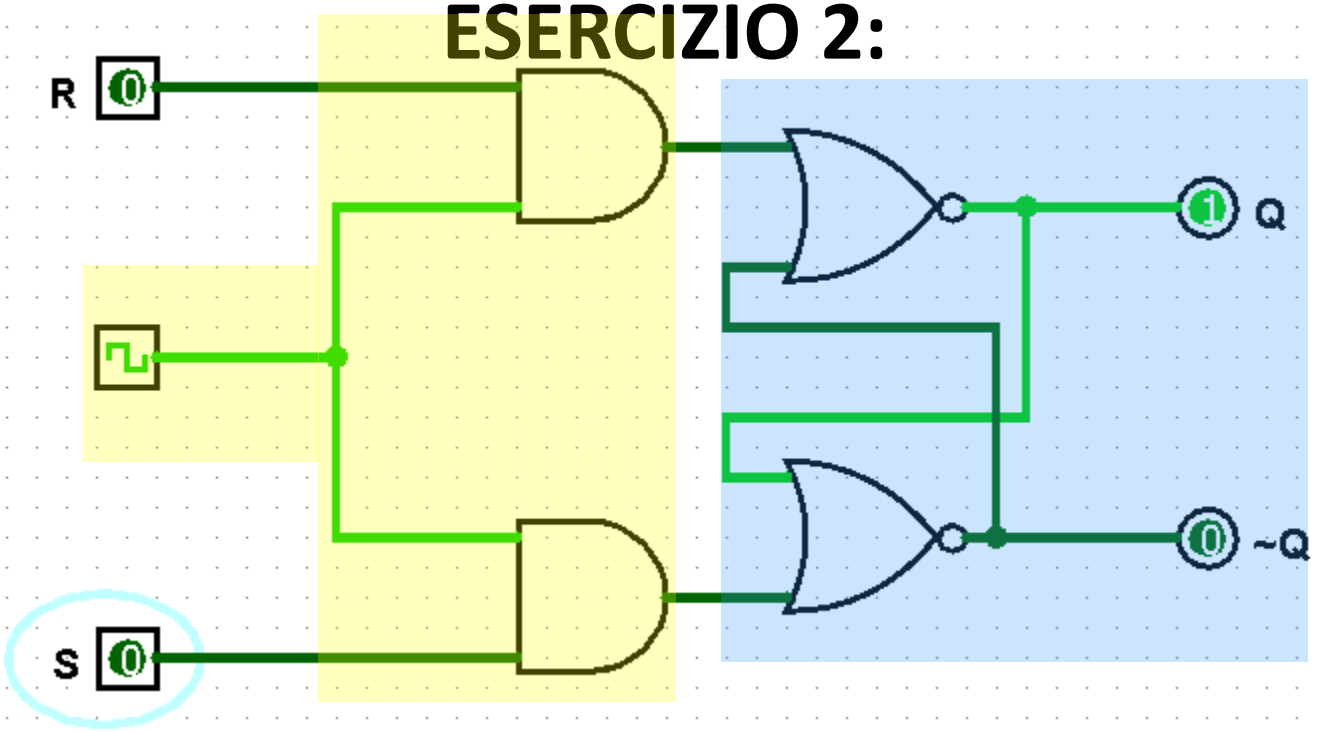
Per attivare il clock durante la simulazione



Per cambiare la frequenza del clock



ESERCIZIO 2:



COSA SUCCEDEREBBE SE IL RITARDO DELLE PORTE FOSSE MOLTO ALTO?

Riardo molto alto → oscillazioni impredicibili nello stato di uscita del bistabile
Motivo: i segnali non avrebbero tempo di propagarsi da ingressi a uscite in 1 ciclo di clock
Soluzione: abbassare la frequenza di clock

LATCH sincrono D

ESERCIZIO 3:

Utilizzando il circuito **(il componente)** del latch SR sincrono realizzato in esercizio 2 si realizzi un latch sincrono D

Suggerimento: è presente un clock … il latch sincrono D memorizza il valore di **D** quando il clock è alto

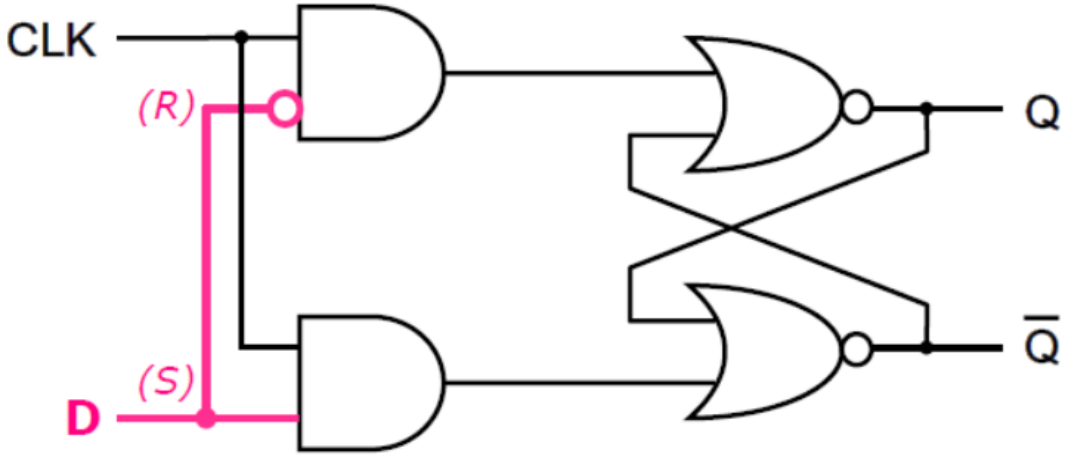
LATCH sincrono D

Latch SR è un po' scomodo: 2 ingressi separati per memorizzare "0" o "1"

IDEA: piloto **S e R** con un solo ingresso: **D**

D = 1 → S=1, R=0 (Set) → **Q=1**

D = 0 → S=0, R=1 (Reset) → **Q=0**



LATCH D:

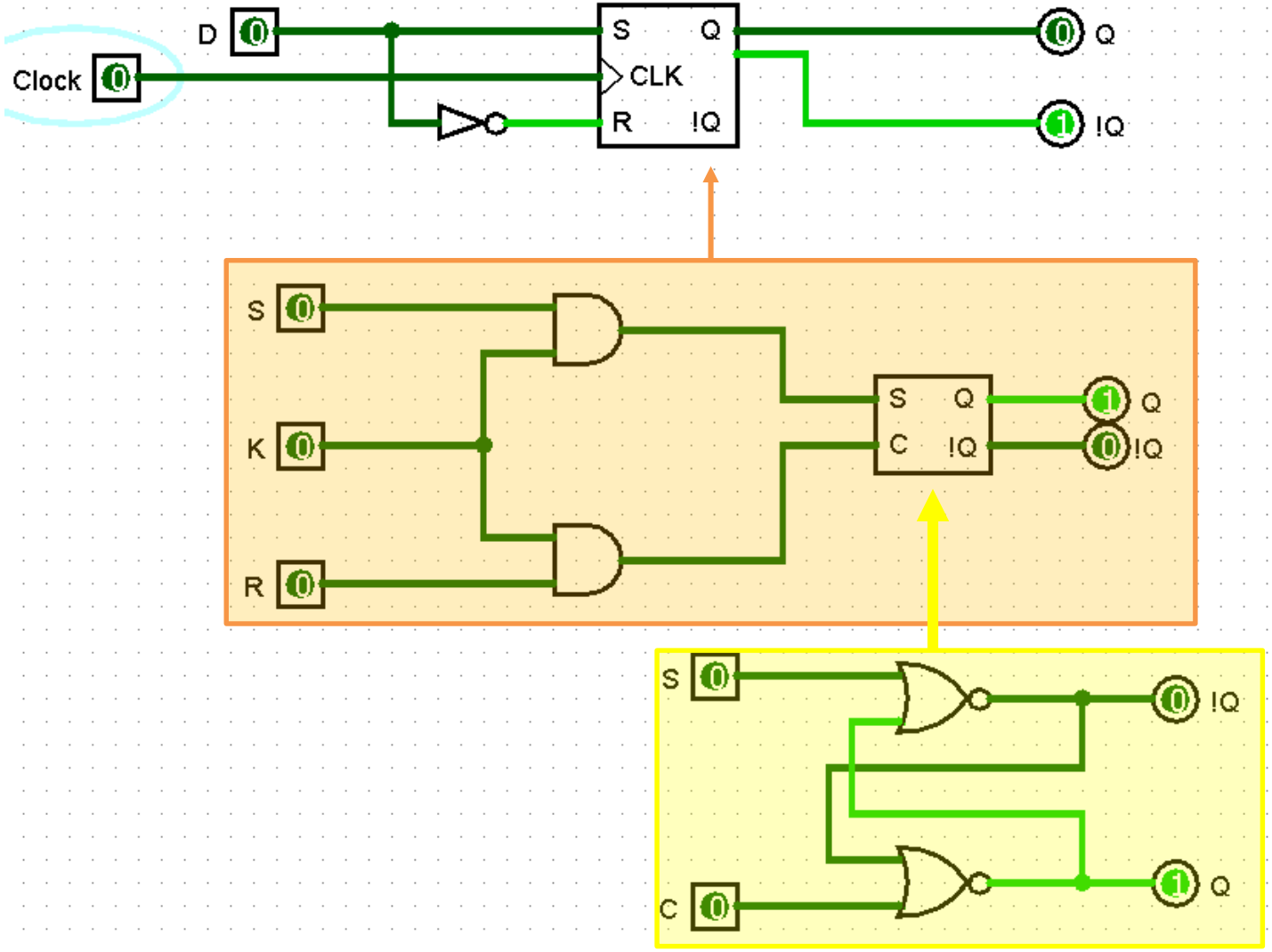
Clock ALTO (CLK=1):

- L'uscita Q insegue l'ingresso D (con 3Δτ di ritardo)

Clock BASSO (CLK=0):

- L'uscita Q rimane bloccata sull'ultimo valore assunto

LATCH sincrono D

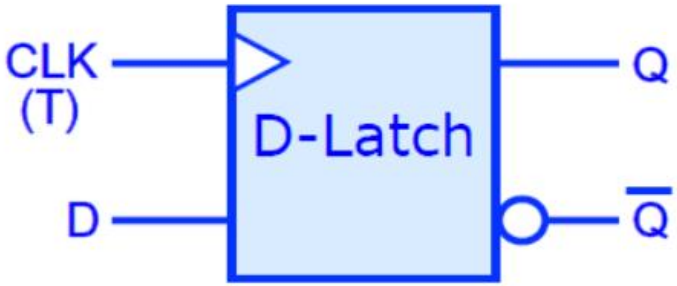


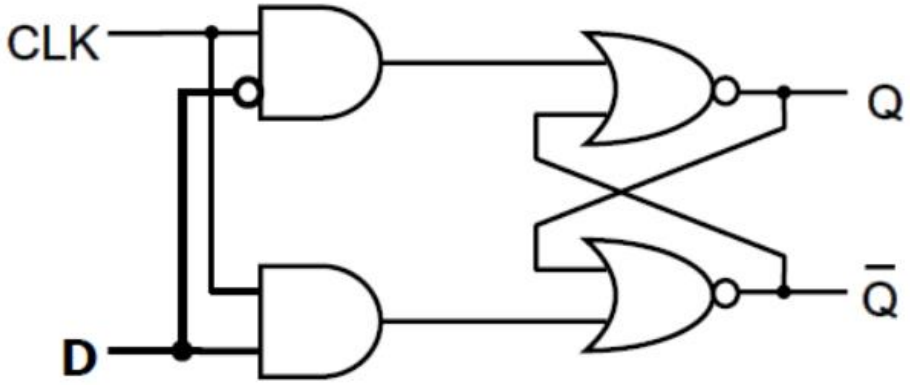
LATCH sincrono D

LATCH D sincrono:

Memorizza il valore presente all'ingresso dati quando il clock è alto, altrimenti (clock=0) si mantiene sul valore memorizzato

if CLK = 1 **then** Q*=D **else** Q*=Q





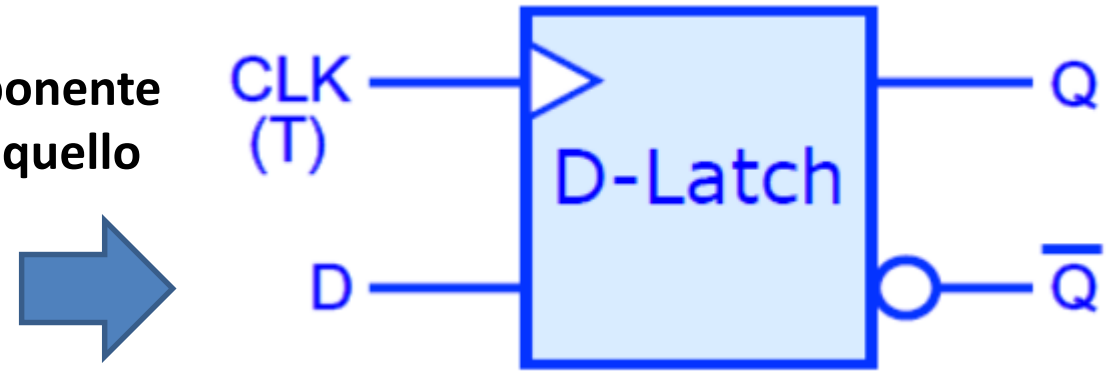
LATCH sincrono D

ESERCIZIO 3:

Utilizzando il circuito **(il componente)** del latch SR sincrono realizzato in esercizio 2 si realizzi un latch sincrono D

Suggerimento: è presente un clock … il latch sincrono D memorizza il valore di **D** quando il clock è alto

Modificare l'aspetto del componente in modo che sia coerente con quello presente nelle slide di teoria.



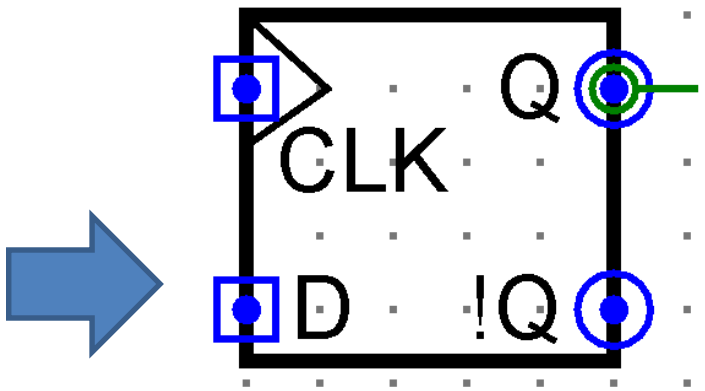
LATCH sincrono D

ESERCIZIO 3:

Utilizzando il circuito **(il componente)** del latch SR sincrono realizzato in esercizio 2 si realizzi un latch sincrono D

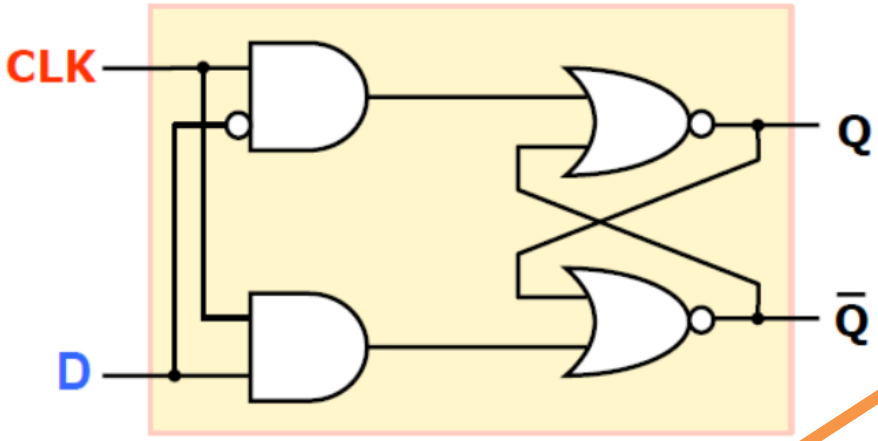
Suggerimento: è presente un clock … il latch sincrono D memorizza il valore di **D** quando il clock è alto

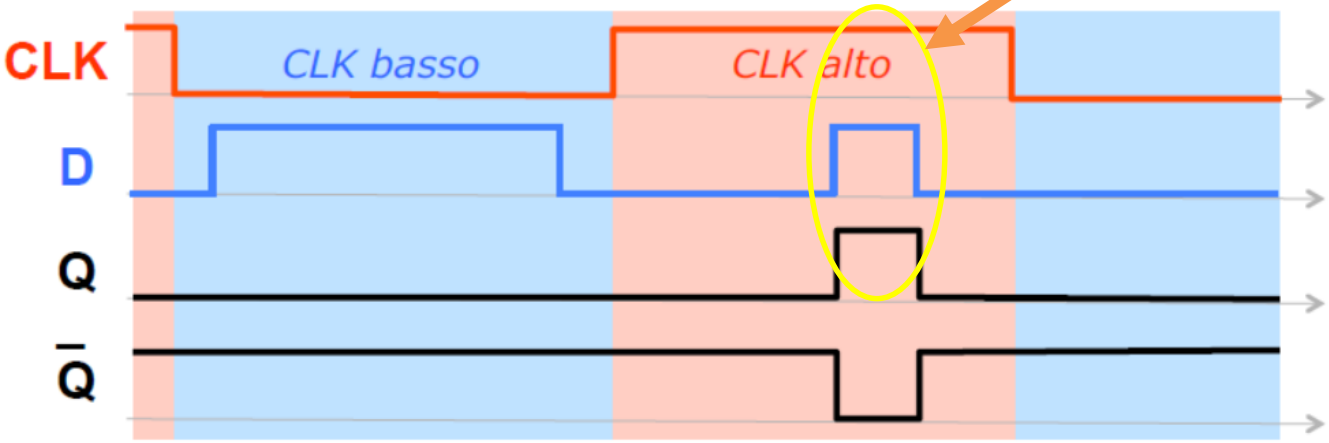
Modificare l'aspetto del componente in modo che sia coerente con quello presente nelle slide di teoria.



Ipotesi: dentro, al posto di scrivere CLK si
Potrebbe mettere D-Latch … il clock si riconosce (il triangolo) e anche D

LATCH sincrono D



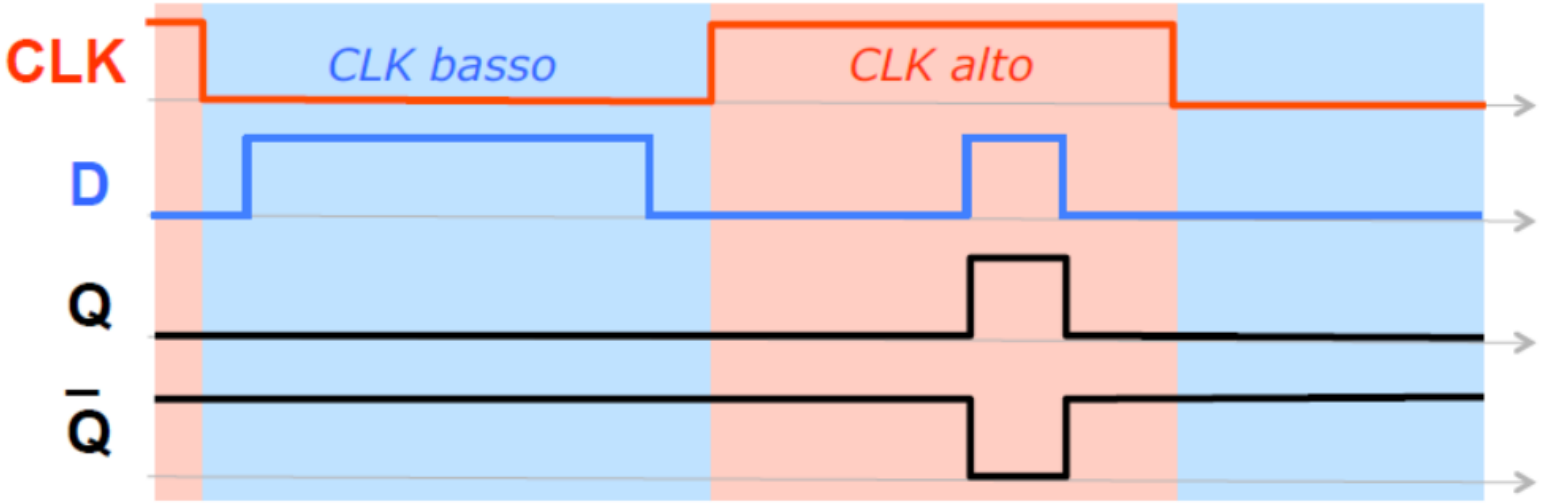


NOTE:

Q insegue D se e solo se CLK è alto.

VERIFICATE usando il circuito appena creato e simulando.

LATCH sincrono D



Problemi di Latch-D:

Il nostro filo conduttore in tutto il laboratorio è quello di vedere come, attraverso dei circuiti, è possibile realizzare una **MEMORIA**, **qualcosa che catturi lo stato di una informazione in un dato istante t**.

Per usare una analogia potremmo parlare di un componente che funziona come una sorta di «macchina fotografica». Latch-D non funziona in questo modo …

LATCH : bistabili sensibili al livello (di clock)

I latch sono dispositivi **trasparenti:** per tutto il tempo in cui il clock è a livello alto, il valore di D viene riportato in uscita

Clock = 1 → Q*=D (uscita collegata all'ingresso)

NB

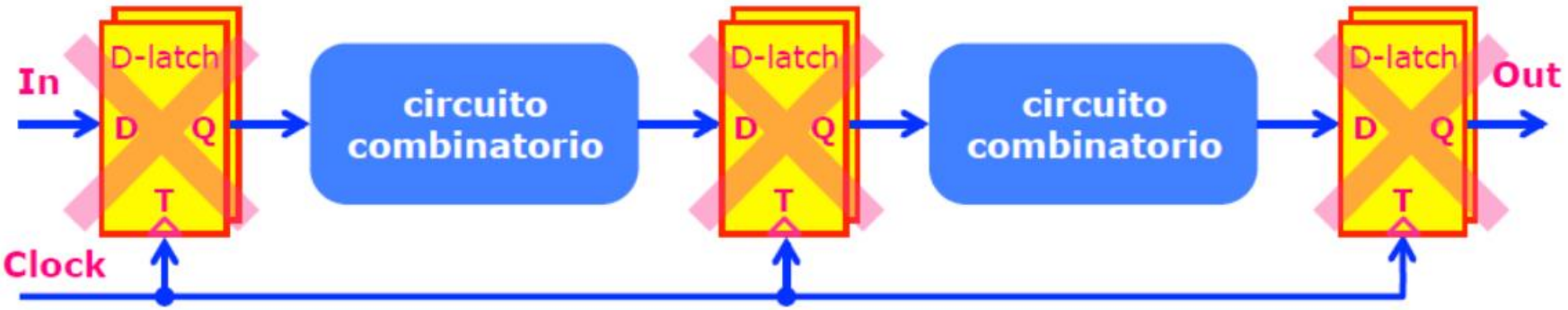
Possono funzionare come **"cancelli"**?

NO! Nel momento in cui "apro" (T=1), i segnali possono attraversare **TUTTI gli stadi**.

Soluzione: "cancello doppio"

Quando apro in ingresso, l'uscita è chiusa, e viceversa.

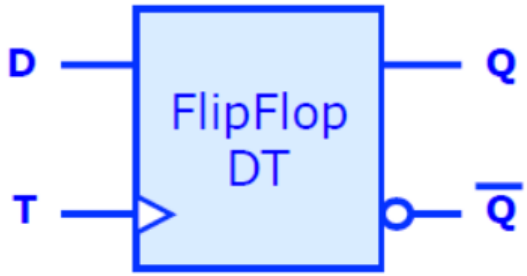
Il cancello posto in **ingresso** può solo impedire ai segnali di **entrare** nel circuito ma appena entrano essi possono raggiungere le uscite



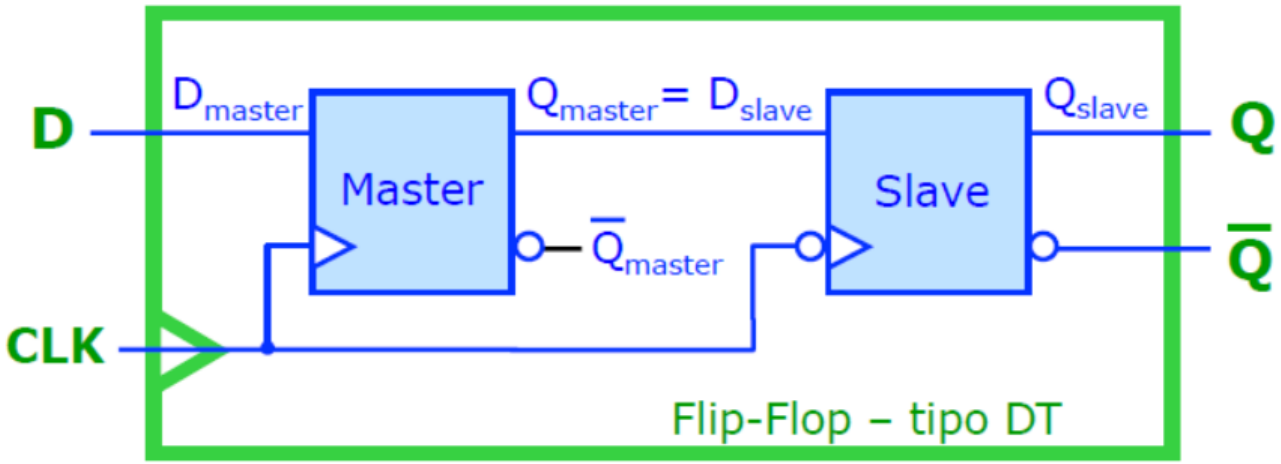
bistabili sensibili al fronte (edge) : FLIP-FLOP

FLIP-FLOP: bistabile *edge sensitive* (attivo sui fronti del clock):

lo stato (uscita) commuta solo in corrispondenza del fronte di salita o di discesa del clock.



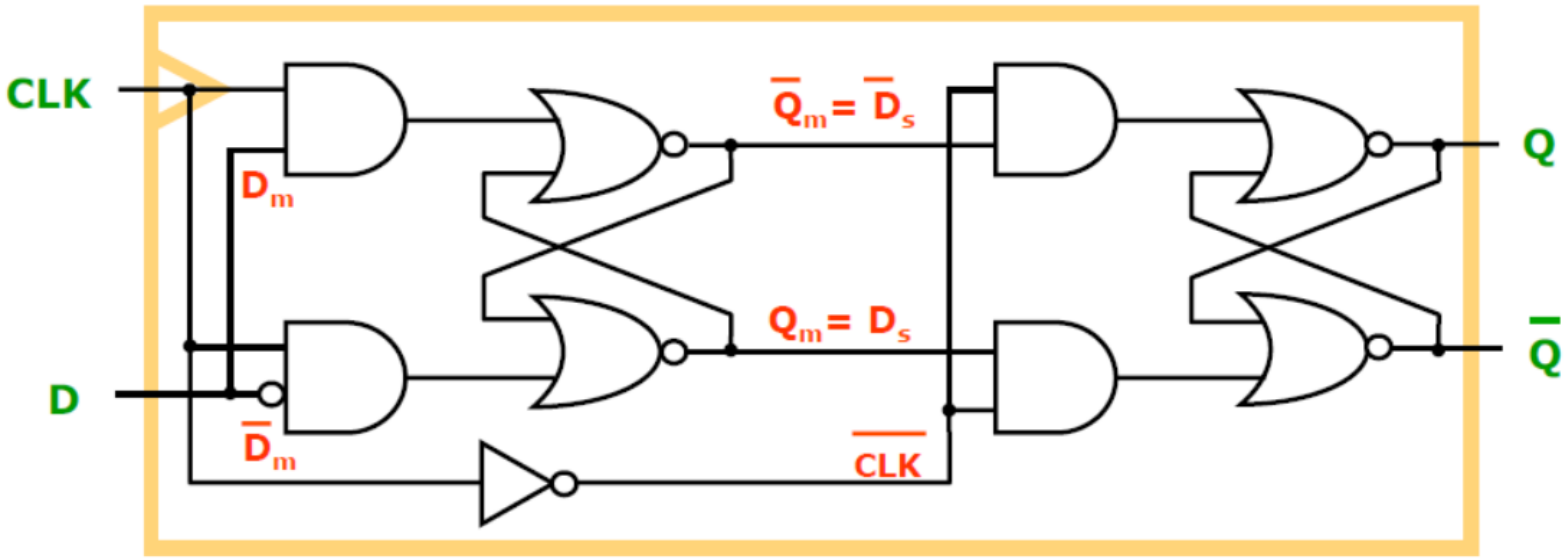
Flip-Flop tipo DT
configurazione "Master-Slave":



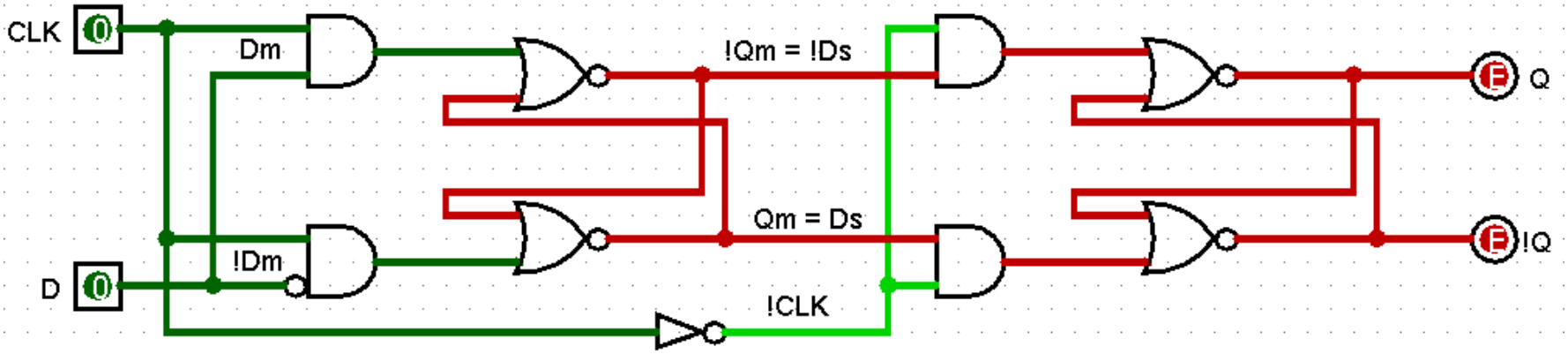
D Type (master-slave) Flip-Flop

ESERCIZIO 4:

Seguendo lo schema sotto riportato si implementi in Logisim un D Type FlipFlop.



D Type (master-slave) Flip-Flop in Logisim



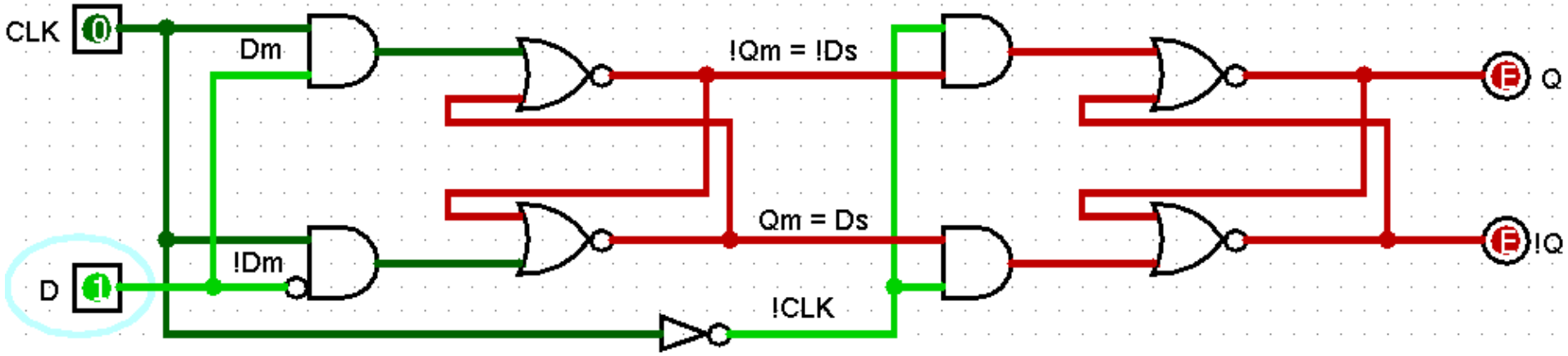
NB: l'unico segnale di clock controlla ENTRAMBI i Latch-D
ma prima di arrivare a Slave è negato

SIMULIAMO …

NB: Prima di iniziare i test **Disabilitate** la simulazione, **Resettate** la simulazione e **riabilitate** la simulazione. Otterrete una figura come quella presente in questa slide.

D Type (master-slave) Flip-Flop in Logisim

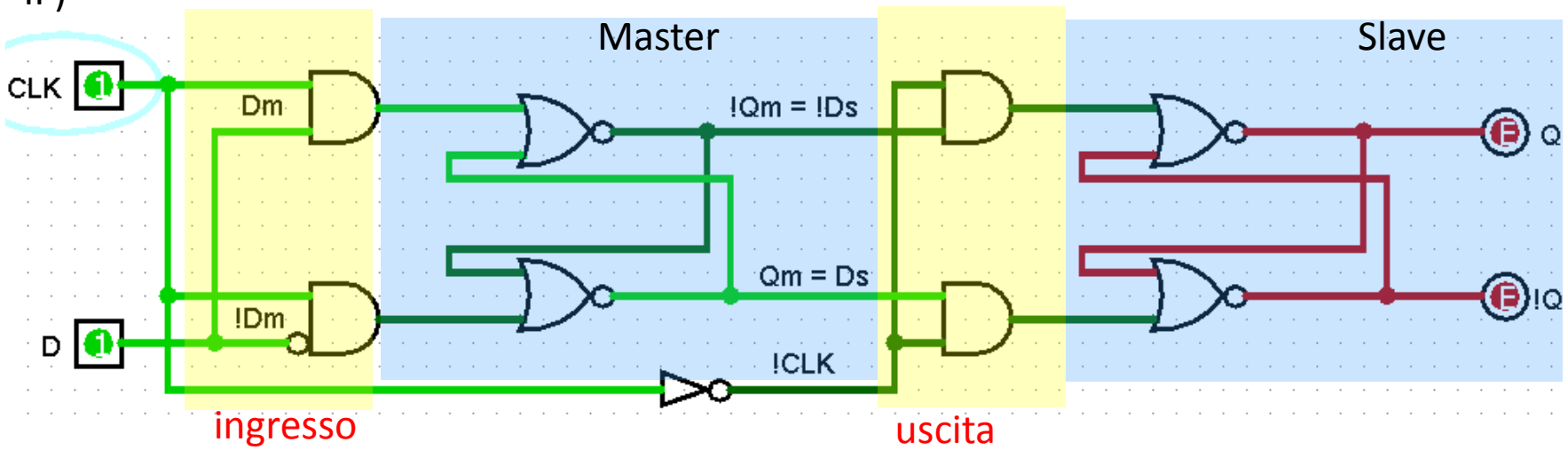
I)



I) Imposto il valore di D a 1 . Lo stato di D è «pronto» a farsi memorizzare nell'uscita stabile di Master ma questo non è possibile perché il clock è basso e il primo cancello blocca l'ingresso del segnale nella parte combinatoria di Master.

D Type (master-slave) Flip-Flop in Logisim

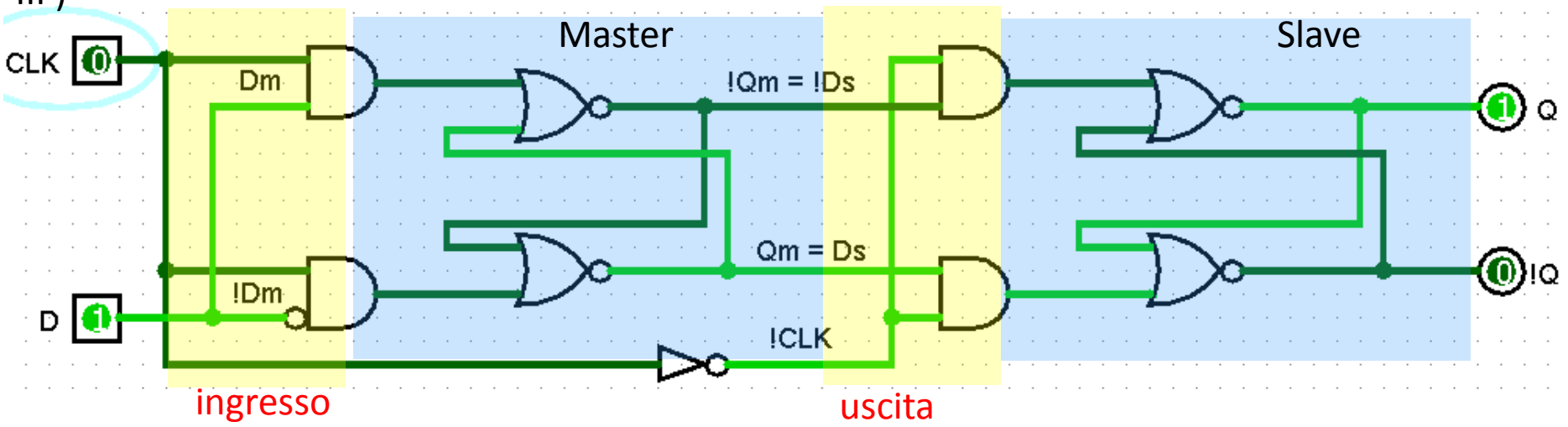
II)



II) Imposto il valore di CLK a 1 . Ora il segnale che proviene da D (che è rimasto 1) può raggiungere l'uscita stabile di Latch Master. In questo momento (o meglio, nel **FRONTE** in cui CLK è passato da stato basso (0) a stato alto (1), **L'INGRESSO E' APERTO** (il segnale in D è «arrivato» a Qm (di Latch Master).
MA L'USCITA E' BLOCCATA… il segnale non si può propagare in Latch Slave.

D Type (master-slave) Flip-Flop in Logisim

III)

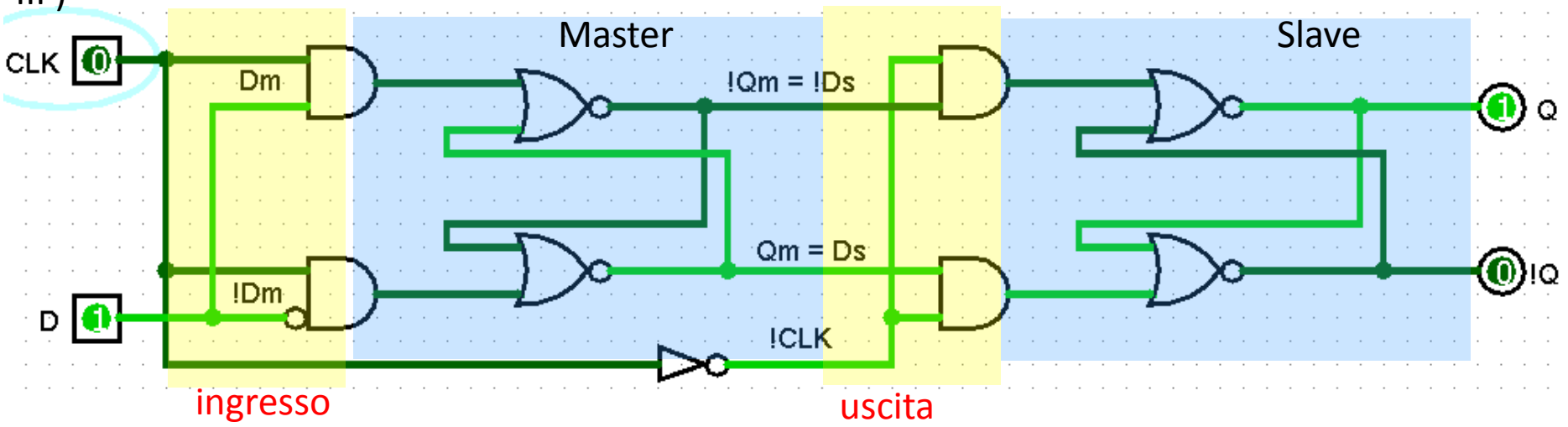


III) Reimposto il valore di CLK a 0 . **FRONTE DI DISCESA**. Ora il segnale proveniente da D e «stabilizzato» in Q di Master è libero (**stato corrente: INGRESSO bloccato / USCITA aperta**) di propagarsi in Latch Slave e di raggiungere l'uscita **Q**.

Note: All'inizio della simulazione (CLK=0, D=0) l'ingresso era bloccato ma l'uscita era aperta. Durante il fronte di discesa siamo ritornati a questa condizione (a parte il valore di D che è ancora impostato a 1).

D Type (master-slave) Flip-Flop in Logisim

III)

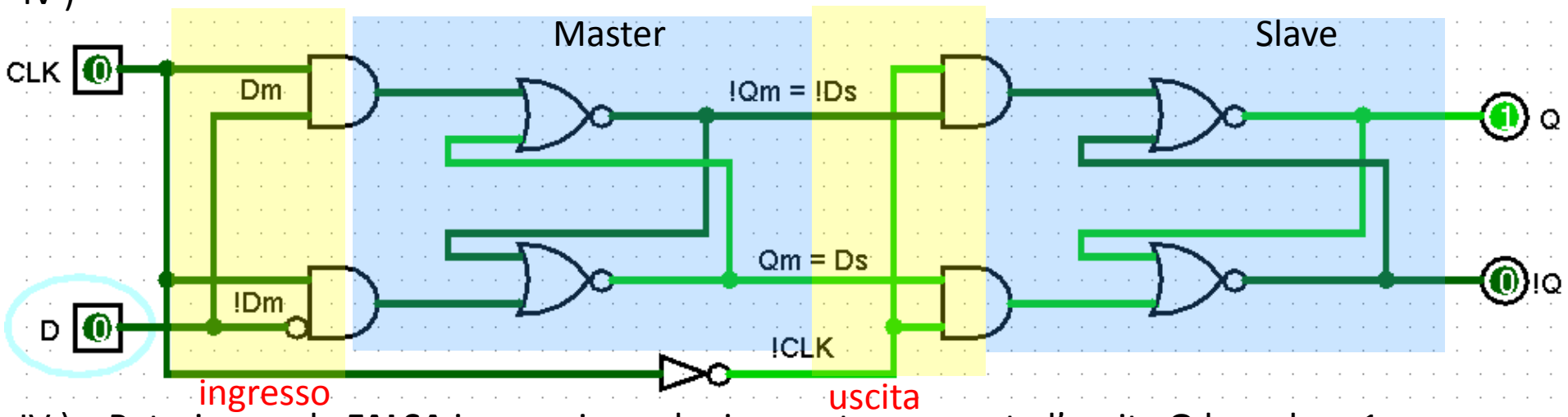


III) Reimposto il valore di CLK a 0 . **FRONTE DI DISCESA**. Ora il segnale proveniente da D e «stabilizzato» in Q di Master è libero (**stato corrente: INGRESSO bloccato / USCITA aperta**) di propagarsi in Latch Slave e di raggiungere l'uscita **Q**.

- **FLOP (clock BASSO)**: l'uscita stabile del latch MASTER viene propagato al latch SLAVE
  - L'ingresso è bloccato, l'uscita è stabile.

D Type (master-slave) Flip-Flop in Logisim

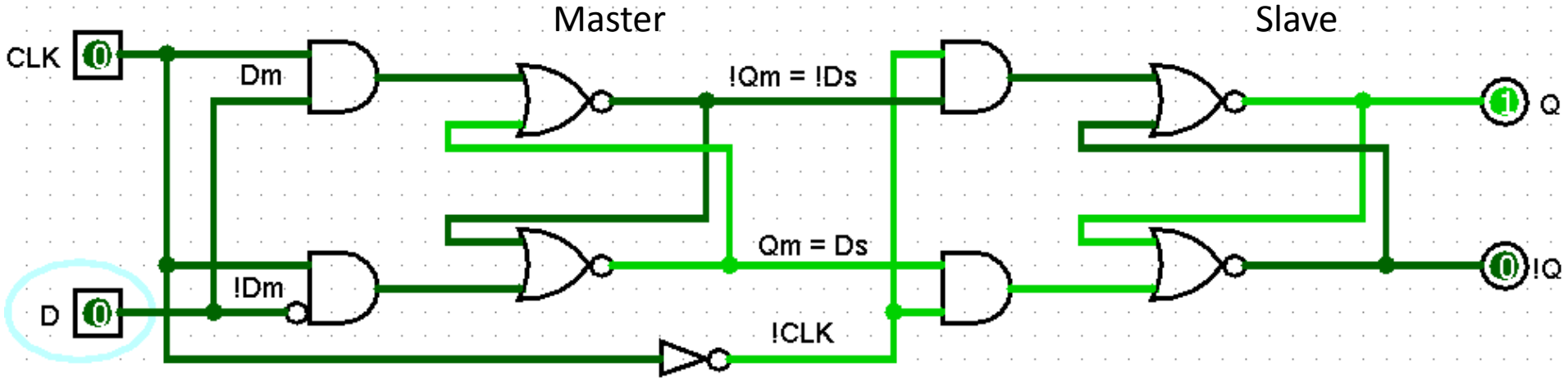
IV)



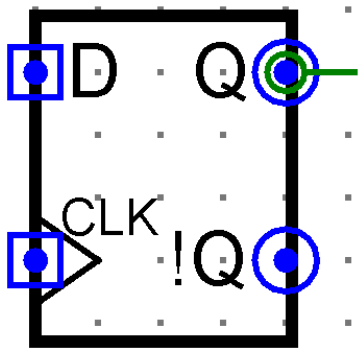
IV) Potrei avere la **FALSA** impressione che in questo momento l'uscita **Q** ha valore 1 a causa del fatto che lo stato di D è 1. Ma sappiamo che non è così (ora l'**INGRESSO** è chiuso) e, infatti, se imposto il valore di D a 0, l'uscita del circuito rimane stabile al valore che ha assunto durante l'ultimo **FRONTE** (ossia il fronte di discesa) del clock.

NOTE: DType master-slave Flip-Flop NON E' SENSIBILE AL LIVELLO DEL CLOCK MA AI FRONTI (DI SALITA/DISCESA) DEL CLOCK.E i segnali sono isolati da barriere (in ingresso, prima di Master e tra Master e Slave). Queste barriere si aprono/chiudono solo in corrispondenza dei fronti.

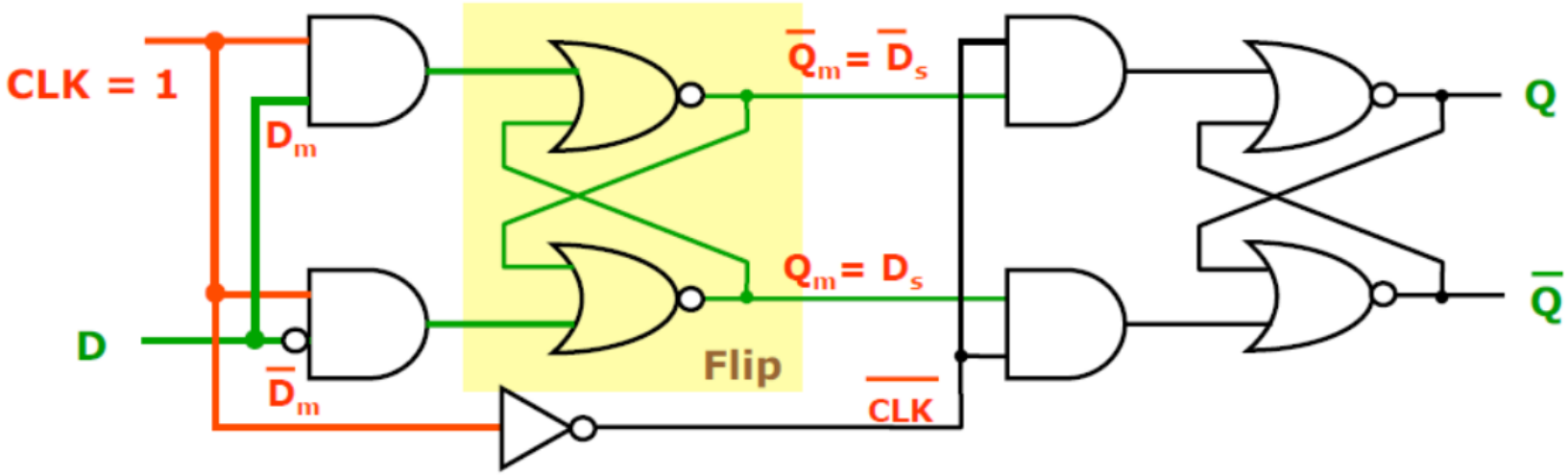
D Type (master-slave) Flip-Flop in Logisim

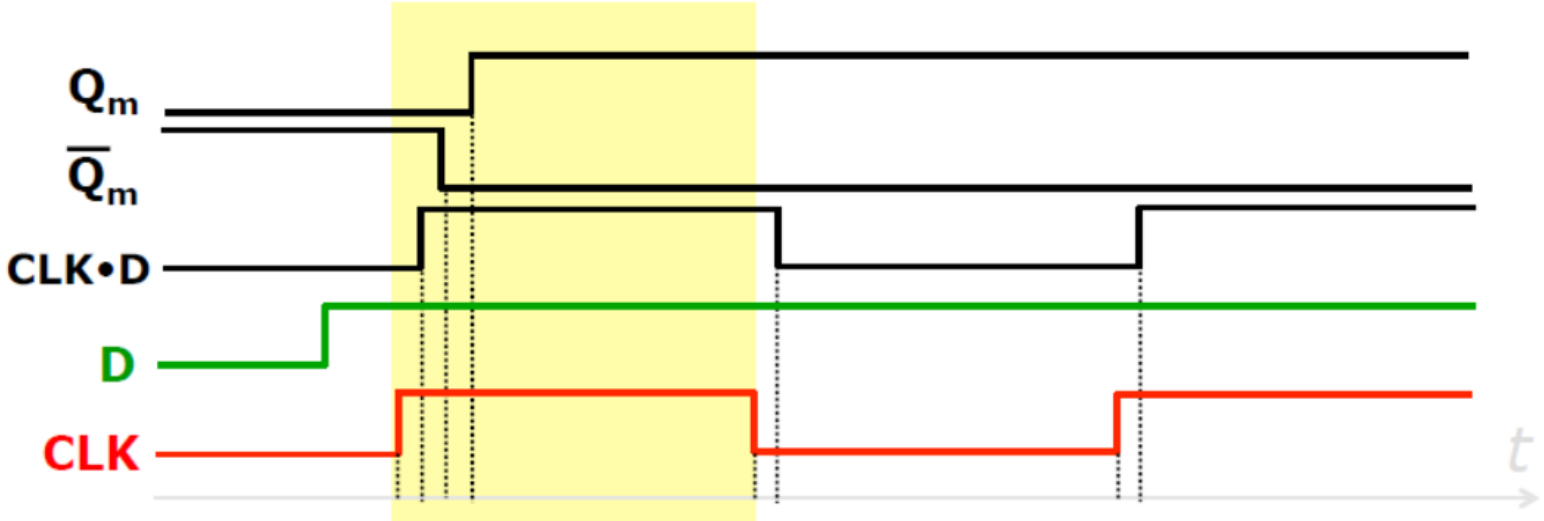


Personalizziamo l'aspetto del componente in questo modo

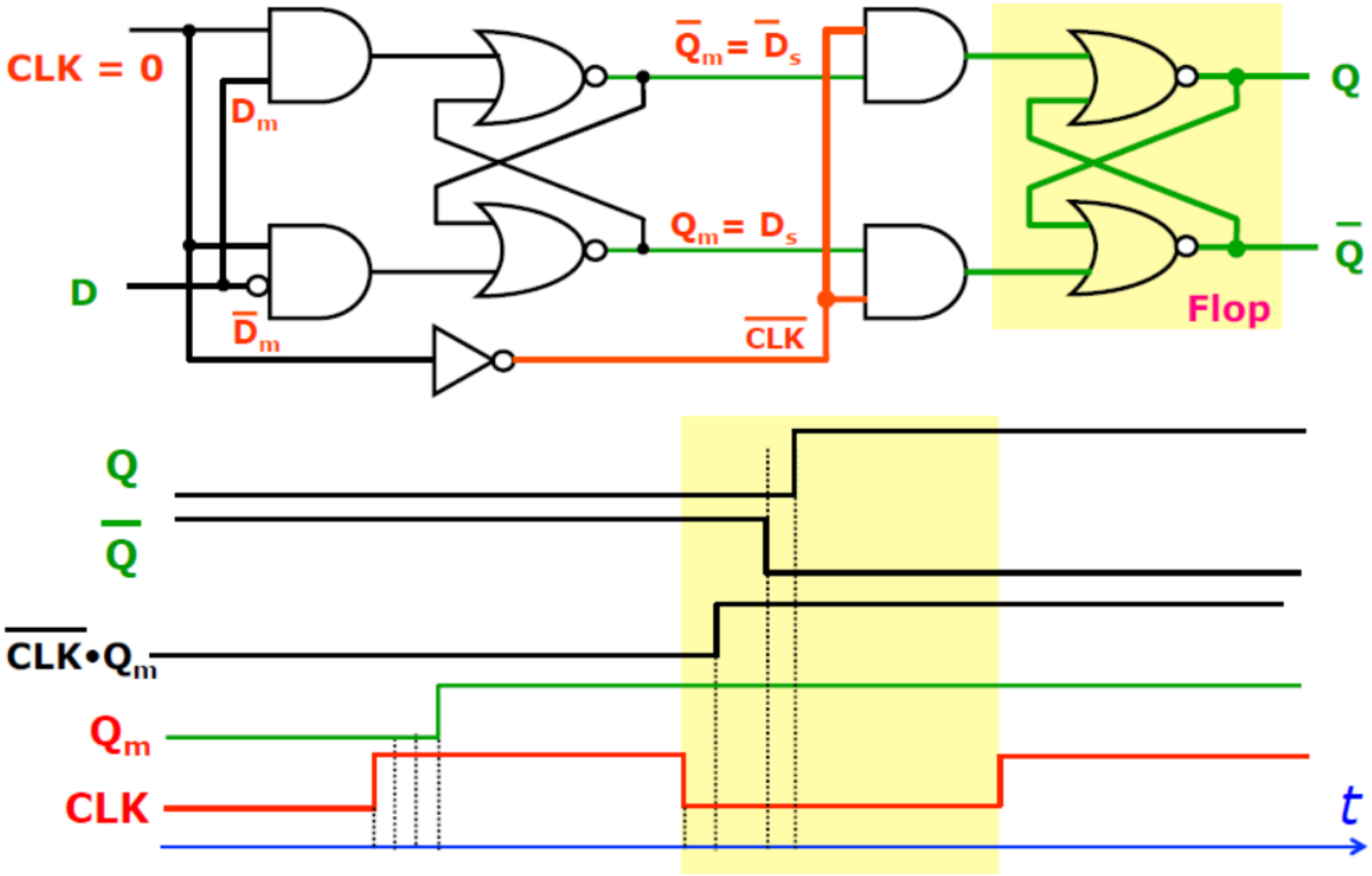


D Type (master-slave) Flip-Flop: FLIP



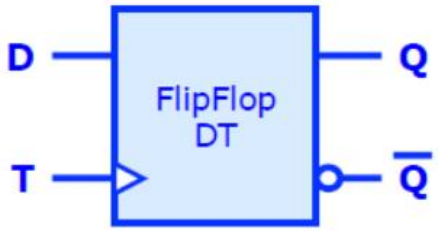


D Type (master-slave) Flip-Flop: FLOP

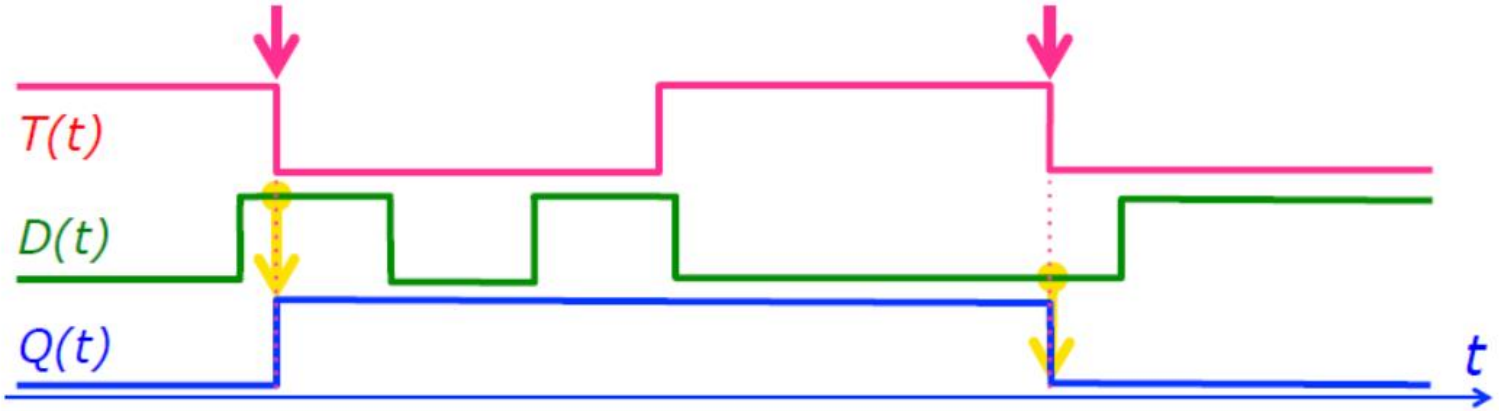


Funzionamento Flip-Flop:

- Fronte di SALITA – FLIP
  - Attivato lo stadio **MASTER**
  - Uscita (stadio slave) invariata
- **Fronte di DISCESA – FLOP**
  - Attivato stadio **SLAVE**
  - Memorizzato il dato sull'ingresso:
  - Presenta il dato memorizzato in uscita:
  - Ingresso isolato



D → STATO
STATO → Q



Registro (Parallel In/ Parallel Out – PIPO)

Registro: unità di memorizzazione di parole di **N bit**

Struttura: **N Flip-flop tipo DT**

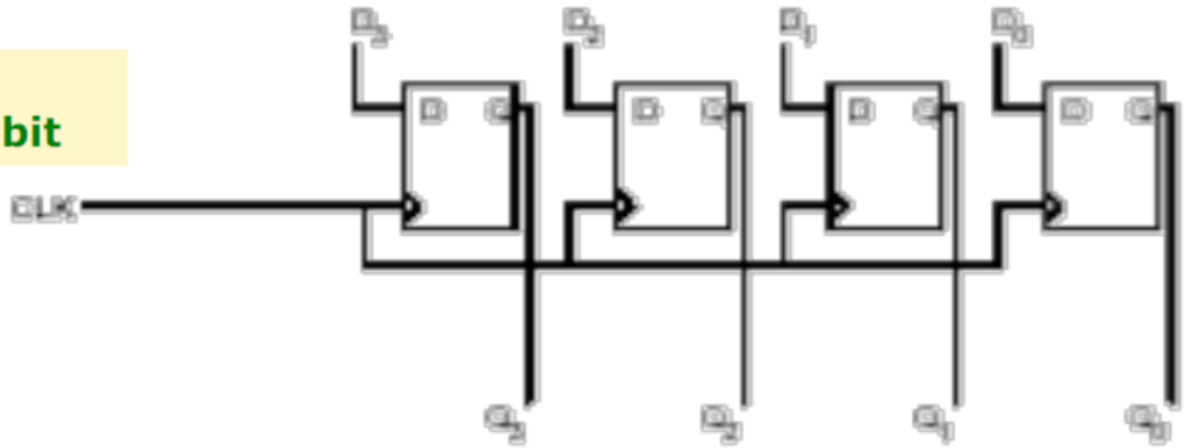
Operazioni: **LETTURA**:

I dati memorizzati sono sempre presenti sulle uscite Q

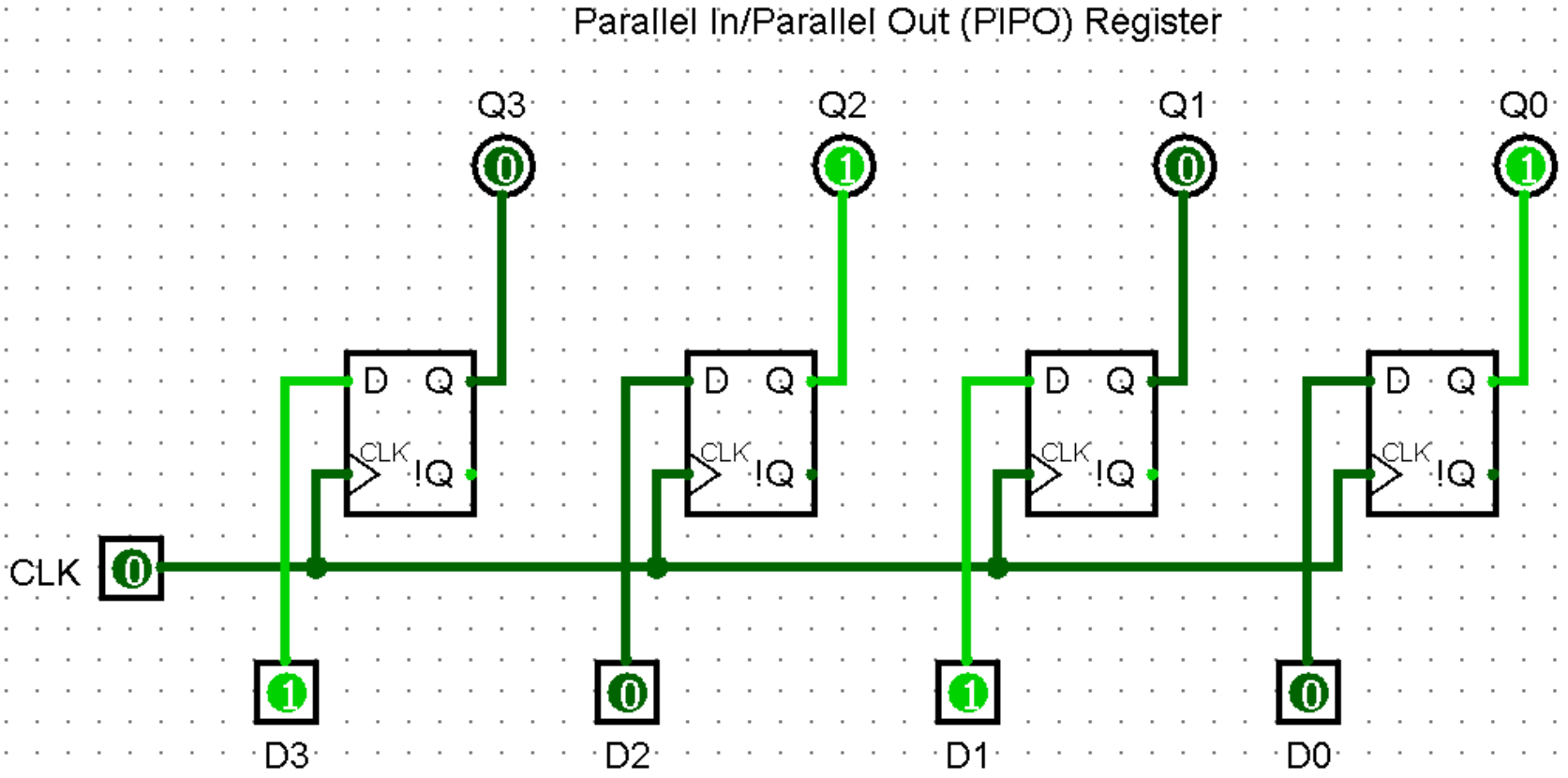
SCRITTURA:

L'impulso di CLK memorizza i dati sugli ingressi D

Esempio: registro a 4 bit



Registro PIPO (4bit) in Logisim



Un impulso di clock (fronte salita/fronte discesa) memorizza stabilmente i valori di N bit nel registro. N = n. bit memorizzabili = n. componenti DType Flip-Flop