Architettura degli Elaboratori

Informatica per la Comunicazione Digitale

Università degli Studi di Milano

Lezione 8:

Blocchi funzionali **sequenziali**

latch, Flip-Flop, e sincornizzazione

Marco Tarini

1

Memoria: la traccia lasciata dal passato

- Una rete combinatoria è sempre **priva di memoria**:
  cioè, la sua uscita dipende **solo**
  dai valori applicati ai suoi ingressi *in quel momento*
  - $out_t = \mathbf{F}(in_t)$ — comportamento **ideale**
- anzi, per la precisione, *poco prima*
  - $out_t = \mathbf{F}(in_{t-\Delta t})$ — comportamento **reale**, con ritardo

uscita del circuito al tempo t — *funzione booleana calcolata* — *ingressi del circuito a tempo t - Δt* — *il «tempo di commutazione» del circuito*

- Il circuito «non si ricorda» di cosa sia successo prima di allora
- Invece, per eseguire certe elaborazioni, anche molto semplici, occorrerebbe conservare memoria di eventi passati
  - Ad es., «l'uscita valga 1 se l'ingresso è stato 1 per un certo tempo»

2

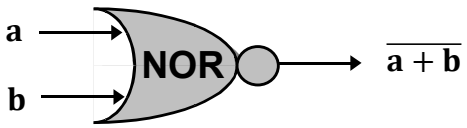
Può un circuito «memorizzare» qualcosa?

- Vorremmo insomma un circuito che «si ricordi» cosa gli è successo prima
  - la sua uscita deve dipendere anche dalla *storia* dei suoi ingressi
  - e non solo da quelli attuali!
- I circuiti **sequenziali** possono avere questo comportamento
  - vediamo come
- Il primo che analizziamo è chiamato *bistabile SR*, o *SR-latch*
  - si tratta di un **blocco funzionale** di tipo **sequenziale**

def: quelli che hanno anche collegamenti **retroazionati** (a differenza dei circuiti **combinatori**)

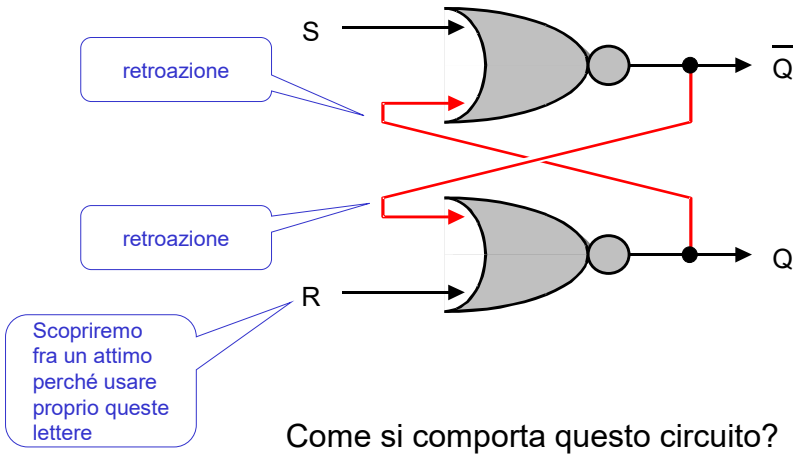
Li chiamiano *sequenziali* proprio perché il loro output può dipendere anche dalla *sequenza* degli input.

Ripasso preliminare: la porta NOR



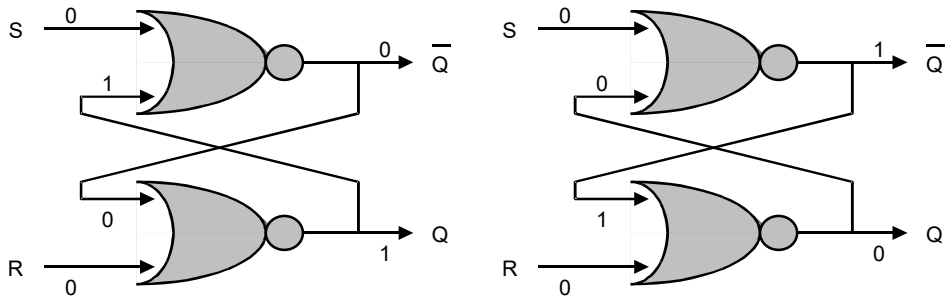
a	b	a NOR b
0	0	1
0	1	0
1	0	0
1	1	0

Il «Bistabile-SR» («SR-latch»)



Come si comporta questo circuito?
Proviamo ad adottare la tecnica della simulazione…

Come si comporta il bistabile SR: **con input mantenuto a S = 0, R = 0**



configurazione stabile A

configurazione stabile B

In entrambe le configuraz, ogni porta sta producendo l'output corretto (dato il suo input). Per questo sono dette **stabili** (e si manterranno *indefinitivamente*).

Stati di memorizzazione del bistabile

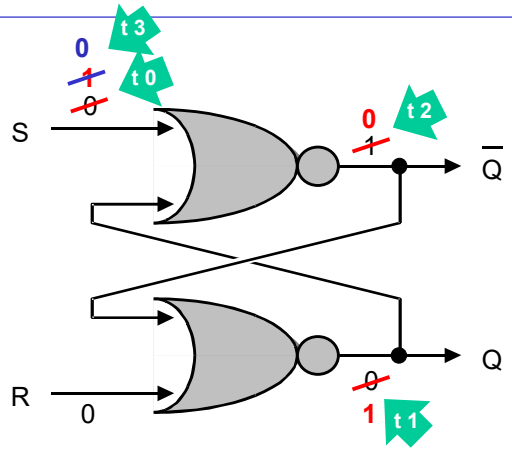
- Quando i due input (S e R) sono a 0,
  il *Latch* ammette quindi **due** stati stabili (per questo è detto «**bi**stabile»)
  - in un dato momento, si troverà in uno dei due stati
- Possiamo dire che lo stato attuale «memorizza» un bit
  - quando Q = 1 (e \Q = 0), il bistabile **«sta memorizzando 1»**
  - quando Q = 0 (e \Q = 1), il bistabile **«sta memorizzando 0»**

- Nota:
  a parità di ingressi (cioè S = R = 0) l'uscita Q ammette due possibili valori.
  - E' un comportamento ben diverso da qualsiasi rete combinatoria!
  - Non è possibile descrivere il comportamento di un circuito così con una tabella di verità :
    cosa dovremmo mettere a riga S,R =0 0 ?
    Forse «*dipende…*»

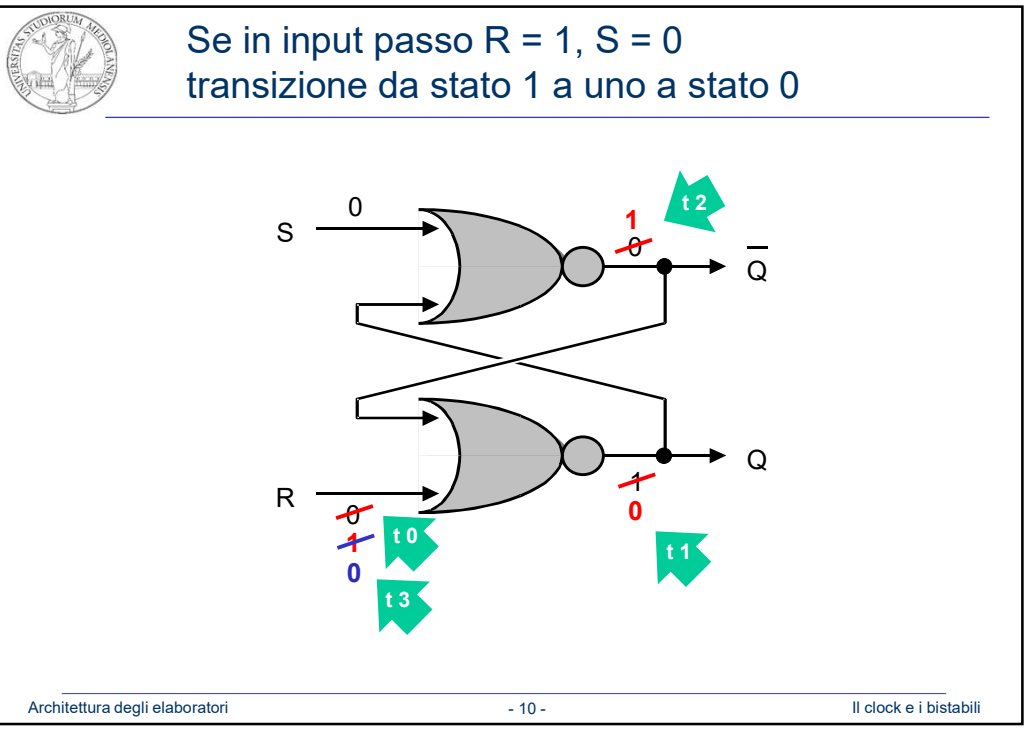
S,R	Q
0,0	???

8

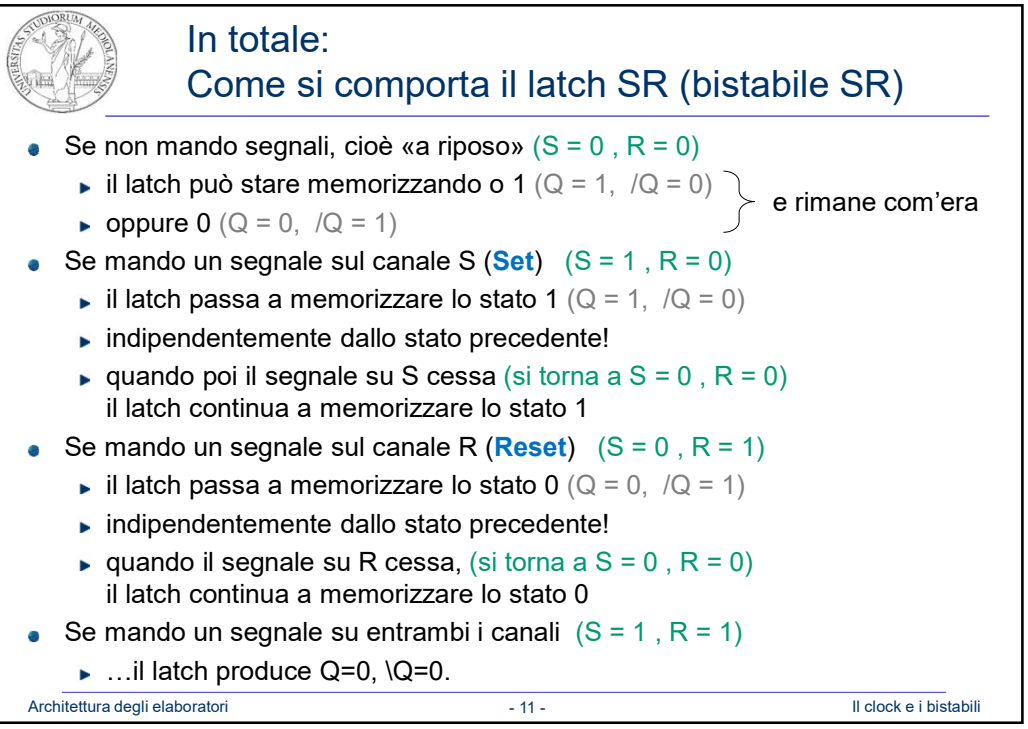
Se in input passo R = 0, S = 1: Transizione da stato 0 a stato 1



9


Se in input passo R = 1, S = 0
transizione da stato 1 a uno a stato 0
S
0
1
0
t 2
Q
Q
R
0
1
0
t 0
t 3
1
0
t 1
Architettura degli elaboratori
- 10 -
Il clock e i bistabili

10

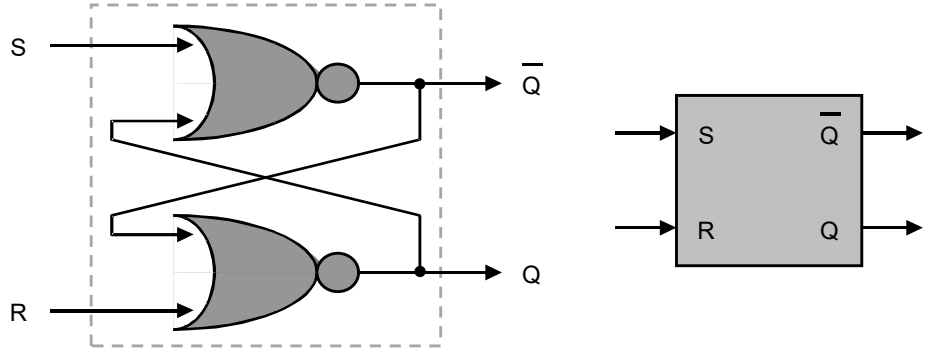

In totale:
Come si comporta il latch SR (bistabile SR)
Se non mando segnali, cioè «a riposo» (S = 0 , R = 0)
il latch può stare memorizzando o 1 (Q = 1, /Q = 0)
oppure 0 (Q = 0, /Q = 1)
e rimane com'era
Se mando un segnale sul canale S (Set) (S = 1 , R = 0)
il latch passa a memorizzare lo stato 1 (Q = 1, /Q = 0)
indipendentemente dallo stato precedente!
quando poi il segnale su S cessa (si torna a S = 0 , R = 0)
il latch continua a memorizzare lo stato 1
Se mando un segnale sul canale R (Reset) (S = 0 , R = 1)
il latch passa a memorizzare lo stato 0 (Q = 0, /Q = 1)
indipendentemente dallo stato precedente!
quando il segnale su R cessa, (si torna a S = 0 , R = 0)
il latch continua a memorizzare lo stato 0
Se mando un segnale su entrambi i canali (S = 1 , R = 1)
…il latch produce Q=0, \Q=0.
Architettura degli elaboratori
- 11 -
Il clock e i bistabili

11

Digressione: Cosa succede in un bistabile SR se S = R = 1

- Fino a che S = R = 1, entrambe le uscite valgono 0
  - Q = 1 , \Q = 0
  - (nota: nessuna semantica è associata a questo stato)
- Ma se tento di tornare a riposo (S = R = 0), l'evoluzione successiva è imprevedibile:
  - se R va a 0 prima (anche poco) di S, rimango nello stato Q = 1 , \Q = 0
  - se S torna a 0 prima (anche poco) di R, rimango nello stato Q = 0 , \Q = 1
  - è sostanzialmente impossibile far commutare S e R simultaneamente
- S = 1 e R = 1 non è considerata un input utile, o **VALIDO**

Rappresentazione a livello più alto (come blocco funzionale sequenziale)



Come descrivere formalmente il comportamento di un circuito sequenziale?

- Come **tabella di verità**?

Quale delle due? Dipende dallo **stato attuale**, che dipende dagli **input precedenti**

S	R	Q	$\bar{Q}$	
0	0	come era prima		**due** configurazioni stabili!
0	1	0	1	una sola configurazione stabile
1	0	1	0	una sola configurazione stabile
1	1	0	0	input considerato invalido

Come descrivere formalmente il comportamento di un circuito sequenziale?

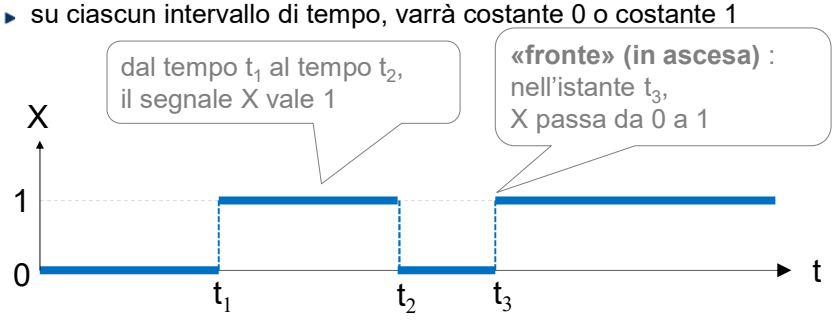
- Come avevamo visto, la **tabella di verità** è un buon modo per descrive in modo completo il comportamento di qualsiasi circuito *combinatorio*
  - Perché questo comportamento non dipende dalla sequenza degli input
- Tuttavia, la **tabella di verità** *non* è un buon modo per descrive in modo completo il comportamento di un circuito *sequenziale*
  - che potrebbe anche non presentare *alcuna* configurazione stabile
- Per descrivere il comportamento di un circuito sequenziale in risposta a una sequenza di input possiamo usare un **diagramma temporale**

Il diagramma temporale di un segnale

Il valore logico (0 o 1) in ciascun cavo (di input, di output, o intermedio) è una **funzione del tempo** X = F(t)

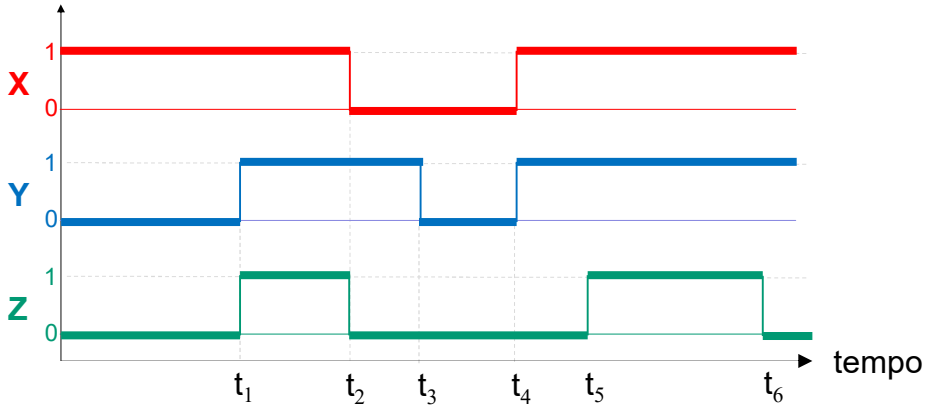
Posso disegnare questa funzione con un plot:

- in ascissa: il tempo *t*
  - composto da vari intervalli
- in ordinata: il valore *logico* del segnale
  - su ciascun intervallo di tempo, varrà costante 0 o costante 1



Il diagramma temporale di un circuito

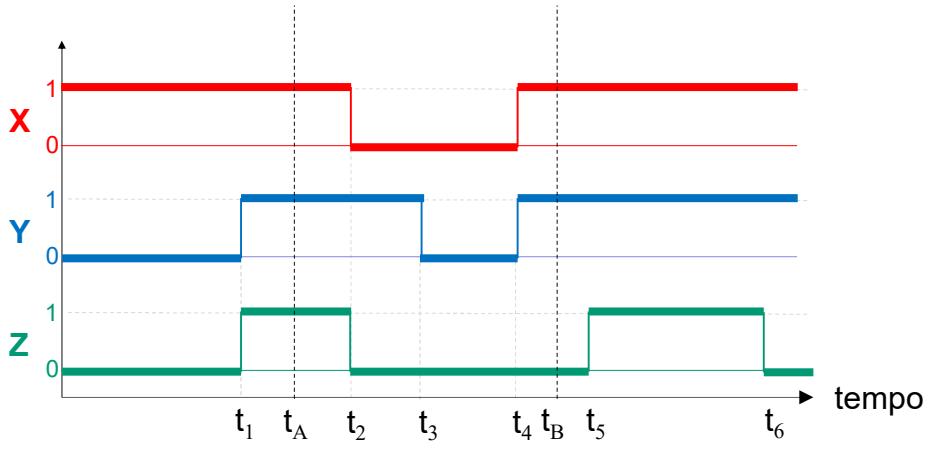
Il **diagramma temporale** del circuito è costituito da un asse tempo e alcuni plot del valore dei segnali (uno per ciascun segnale di input, output, e, se necessario, interno)



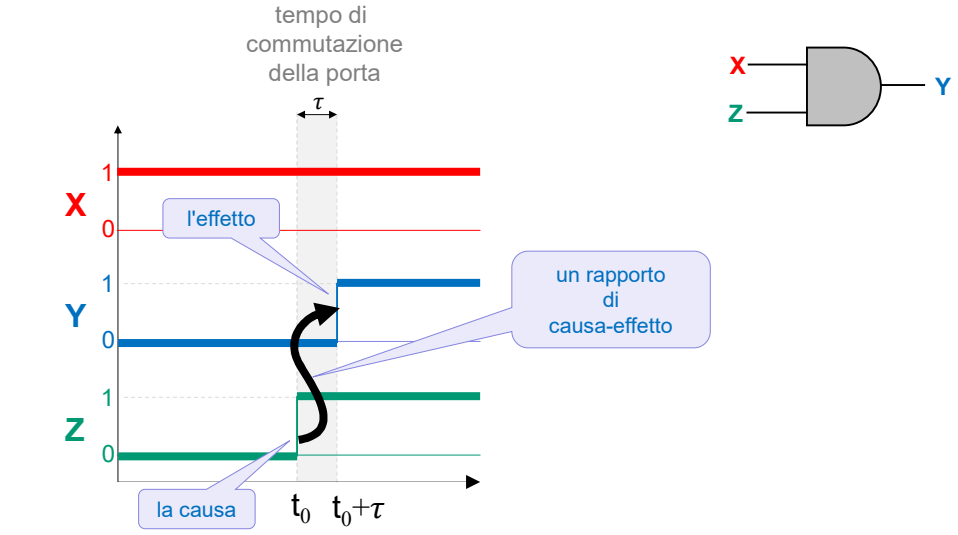
Il diagramma temporale di un circuito

osservazione al tempo t_A : X = 1, Y = 1, Z = 1

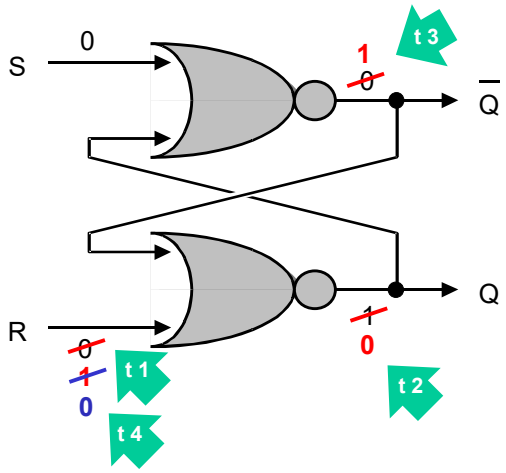
osservazione al tempo t_B : X = 1, Y = 1, Z = 0



Il diagramma temporale: rapporti di causa-effetto fra fronti

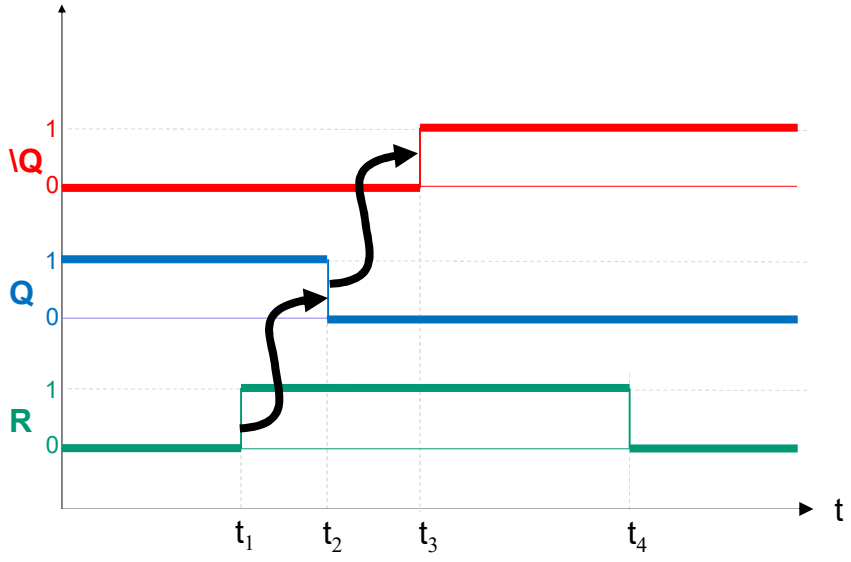


Esempio: «un latch SR viene *resettato* attraverso una temporanea attivazione di R»



22

Esempio: «un latch SR viene *resettato* attraverso una temporanea attivazione di R»



23

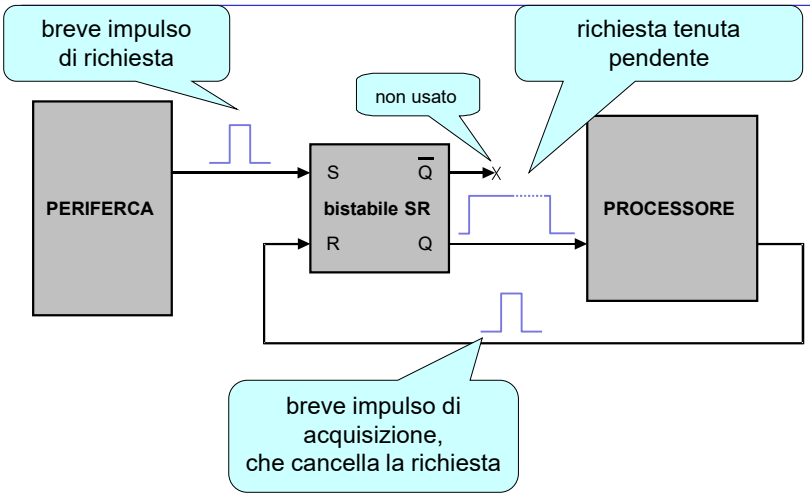
Usi tipici del latch SR

- Il bistabile SR è un blocco funzionale (sequenziale) molto usato
- Ne vedremo due usi tipici:
  - È utilissimo come l'interfaccia fra una periferica (esempio, una *keyboard*) e l'elaboratore centrale
  - Lo useremo come blocco funzionale per progettare di altri tipi di bistabili, più raffinati o specializzati

Esempio di Latch usato come interfaccia fra periferica e processore

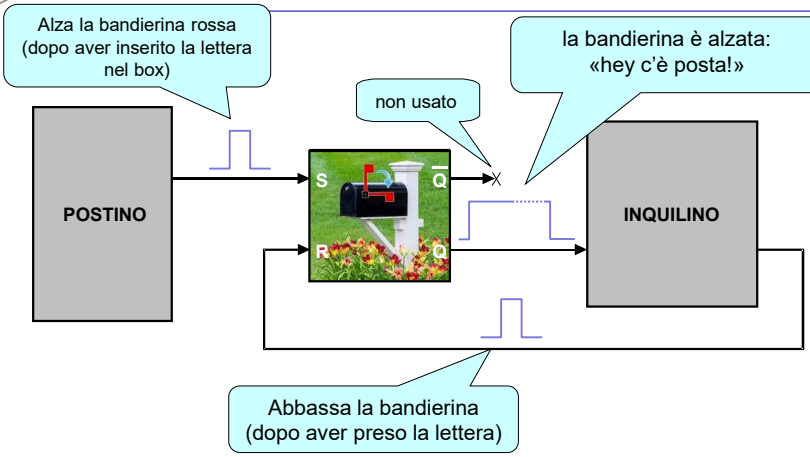
- Si supponga di avere una periferica (ad es. una tastiera) che deve mandare un segnale di richiesta (ad es. un «è stato premuto un tasto») a un processore, che deve reagire di conseguenza
  - La periferica notifica questo generando un breve impulso di «richiesta»
  - Il processore potrebbe essere occupato e non in grado di rispondere subito alla richiesta (cioè non in grado di onorarla subito)
  - Il Latch tiene traccia della «richiesta pendente» fino a quando il processore la potrà onorare
- Soluzione: interporre tra periferica e processore un SR-latch che
  - riceva l'impulso di richiesta proveniente dalla periferica, e lo memorizzi (S = 1 per un breve istante ⎍)
  - mantenga attiva la richiesta pendente (Q=1 indefinitivamente ⎴) fintantoché il processore non sia disponibile a onorarla
  - cancelli la richiesta (Q torna a 0 indefintivametne ⎣), quando il processore segnali, con un impulso (R = 1 per un breve istante ⎍), di averla preso in carico la richiesta, e di stare procedendo ad onorarla

Esempio di Latch usato come interfaccia fra periferica e processore



Il bistabile SR funziona come adattatore, o interfaccia, tra periferica e processore

Esempio di Latch usato come interfaccia fra periferica e processore

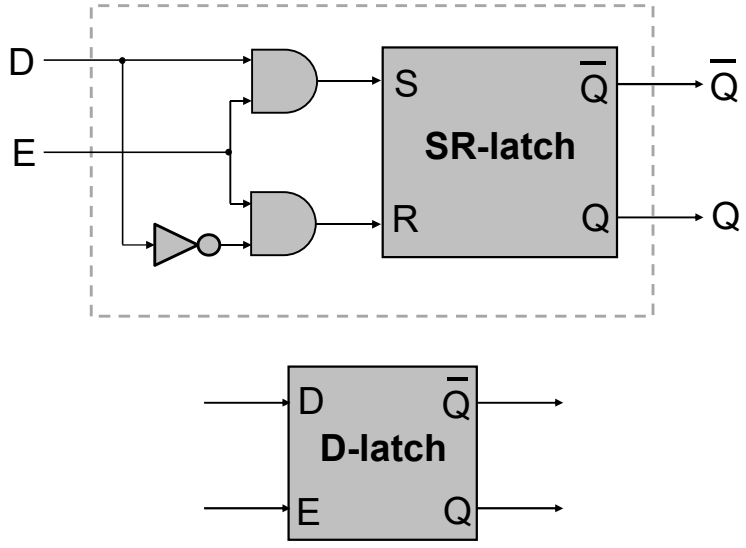


Il bistabile SR funziona come adattatore, o interfaccia, tra periferica e processore

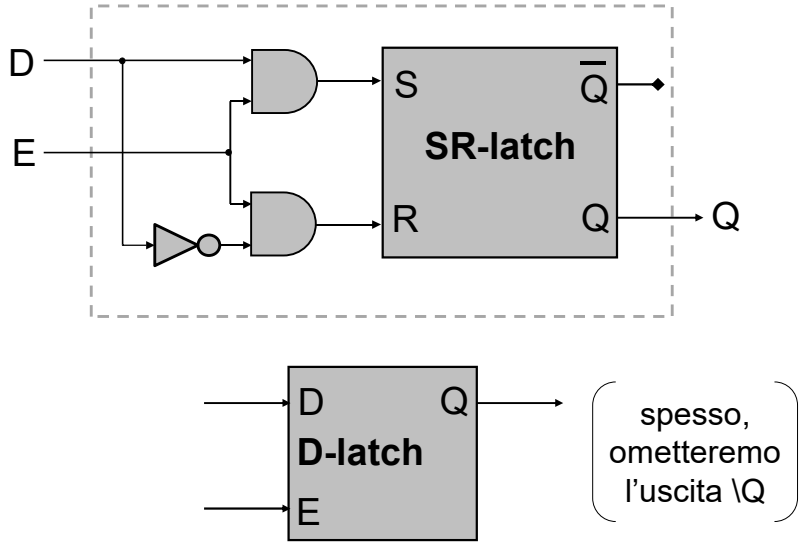
Bistabile D con guardia («gated D-latch», o anche solo «D-latch»)

- Totale: il bistabile SR (*SR-latch*) memorizza un bit e ha due ingressi:
  - S «set», per portare lo stato a 1
  - R «reset», per portare lo stato a 0
- Quindi S e R hanno una funzione sia di *comando* che di *dato*:
  - indicano **che va memorizzato** un bit, e al contempo
  - indicano **quale bit** memorizzare
- Molto spesso è più utile un bistabile che separi le due cose.
- Il bistabile D con guardia (*gated D-latch,* o *D-latch*) ha due ingressi:
  - D «data» indica **quale** dato memorizzare (0 oppure 1)
  - E «enable» indica ***se*** memorizzarlo oppure no (è il *gate*, la guardia)
    (se E vale 1, D viene memorizzato)
    (se E vale 0, D viene ignorato)

Gated D-latch: implementazione



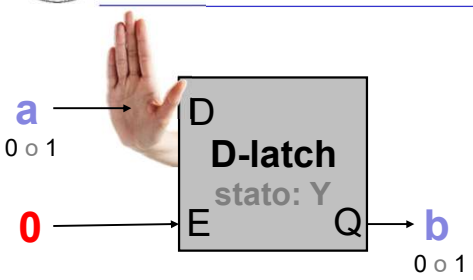
Gated D-latch: implementazione



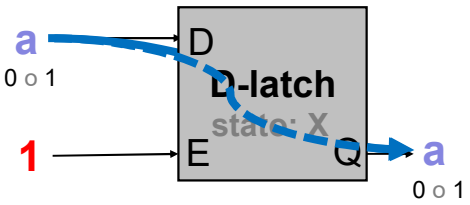
Gated D-latch: implementazione Note

- Il segnale E viene messo in AND con entrambi gli input S-R del SR-latch
- **quando E = 0**, allora S = R = 0 (e D non conta, perchè **0**·X = 0)
  - quindi il bistabile *mantiene sempre il suo stato*
  - si dice che il D-latch è **congelato**
  - oppure «**opaco**»:
    l'output Q non dipende più da D
    l'input D "non si vede" da Q
- **quando E = 1**, allora S = D, R = \D (**1**·X = X)
  - quindi il bistabile *assume lo stato indicato da D*
    (D = 1 → S=1, R=0 , e il latch assume 1)
    (D = 0 → S=0, R=1 , e il latch assume 0)
  - nota: questo impedisce che l'input "illegale" S=1,R=1
    venga mai fornito al SR-latch (olè 👌)
  - si dice che il D-latch è «**trasparente**», in condizione di «**trasparenza**»,
    perché l'input D "si vede" da Q

Gated D-latch: comportamento



E=0 → **bloccato**, **frozen**, **opaco**.
(*leggo* dal latch il bit **b** memorizzato in precedenza)



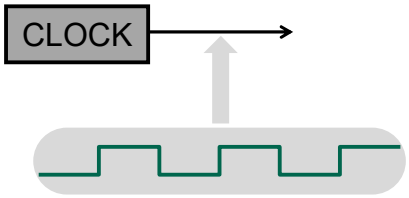
E=1 → **attivo**:
(*memorizzo* nel latch il bit **a** che mando nel canale D)
e **trasparente**:
(da Q esce direttamene lo stesso **a**)

Sincronizzazione

- Un SR-latch è sempre ASINCRONO
  - In *qualsiasi istante* si mandi un impulso S (o R), il latch assume lo stato 1 (o 0)
- Un D-latch può essere reso SINCRONO, cioè sincronizzato con un clock, se il suo gate (E – enabled) è connesso ad un clock
- Per ottenere questo scopo occorre:
  - disporre di un **segnale di clock** che scandisca gli istanti o intervalli di tempo in cui le transizioni di stato possono avvenire
  - sincronizzare ogni bistabile con questo segnare di clock
  - ad ogni ciclo di clock, tutti i bistabili memorizzano il segnale che hanno in ingresso

Il clock

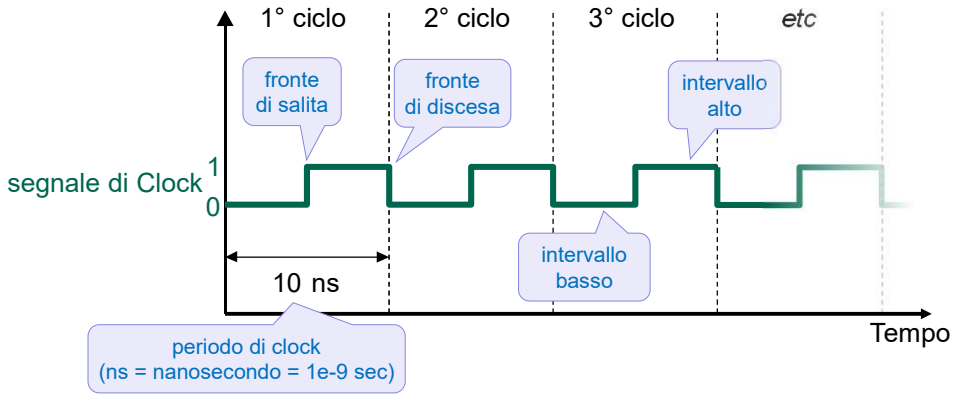
- Il **clock** è dispositivo che produce un semplice segnale binario con andamento **periodico** nel tempo (1 poi 0 poi 1 poi 0…)
  - il segnale viene diramato all'intero dell'intero processore
- Come è costruito questo dispositivo al suo interno?
  - Questa domanda pertiene al *livello dei dispositivi* – non ce ne occupiamo
- Il segnale di clock (o «di temporizzazione») viene usato per scandire il tempo



Segnale di Clock

48

Anatomia di un segnale di clock (segnale periodico)



Frequenza di clock = 1 / periodo di clock
Qui: periodo = 1 / 10 ns → Frequenza = 100 MHz

49

Note terminologiche sul segnale di clock

Convenzioni che adottiamo

- Il **fronte di discesa** del clock è considerato l'ultimo (ed il primo) attimo di ogni ciclo di clock
- Il **fronte di salita** del clock avviene a metà di un ciclo di clock
- Un **ciclo di clock** = intervallo alto + intervallo basso
- L'**intervallo basso** di clock perdura per la prima metà di un ciclo di clock
- L'**intervallo alto** di clock perdura per la seconda metà di un ciclo di clock

Metafora:

- Un **ciclo di clock**: giornata completa di 24 ore, da tramonto a tramonto
- L'**intervallo basso**: notte (12 ore)
- L'**intervallo alto**: giorno (12 ore)
- Il **fronte di salita**: alba (istante)
- Il **fronte di discesa**: tramonto (istante) – fine di una giornata, e inizio della successiva

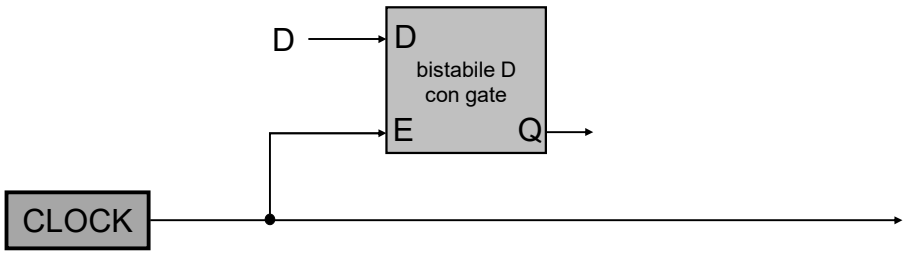
Frequenze tipiche di clock

Periodo di clock (misura: secondi)			**Frequenza di Clock** (misura: Hertz = 1 / secondo)		
1 secondo	1 s	10^0 sec	1 Hertz	1 Hz	10^0 Hz
1 millisecondo	1 ms	10^{-3} sec	1 KiloHertz	1 KHz	10^3 Hz
1 micro secondo	1 μs	10^{-6} sec	1 MegaHertz	1 MHz	10^6 Hz
1 nano secondo	1 ns	10^{-9} sec	1 GigaHertz	1 GHz	10^9 Hz

La **frequenza di clock** di un tipico processore si misura attualmente in qualche GigaHertz (quindi, milioni di cicli di clock al sec)

Bistabile D sincronizzato

- Idea: connettere il clock all'entrata E di un **D latch**



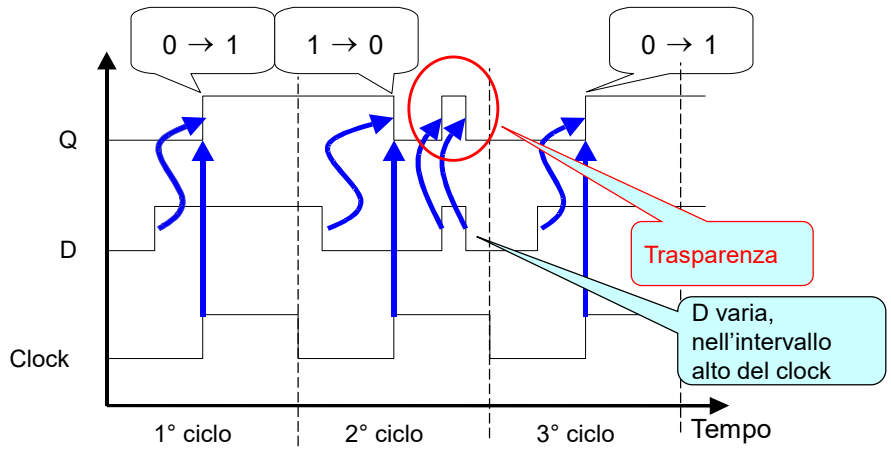
- Ottengo un bistabile **D sincronizzato**: ("sincronizzato"… con il clock)
  - Nell'intervallo basso del clock,
    il bistabile è **congelato** (l'ingresso D è ignorato)
  - Nell'intervallo alto del clock,
    il bistabile è **attivo**: l'ingresso D viene memorizzato (bene!)
    e **trasparente**: l'uscita Q riflette direttamente D (male!)

Sincronizzazione «sul livello»

Questo modo di sincronizzare un bistabile con il clock è detto **sincronizzazione «sul livello»**:

- quando il clock vale 0: (cioè nell'intervallo basso del clock)
  il bistabili sono congelati.
  In questo intervallo, l'output del latch è stabile
- quando il clock vale 1: (cioè nell'intervallo alto del clock)
  gli ingressi ai vari bistabili dell'archietettura sono efficaci
  ⇒ e anche trasparenti (qui il problema)

Diagramma temporale



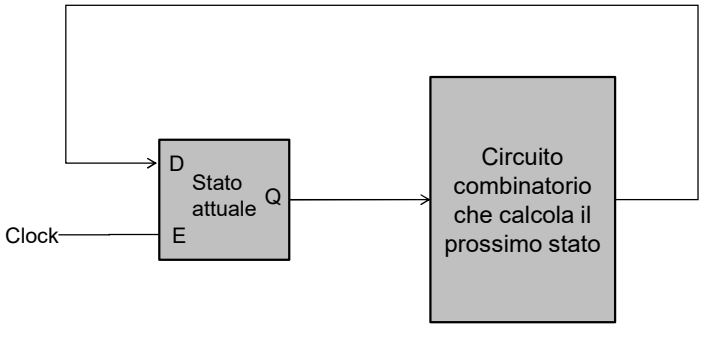
Nel 2º ciclo, mentre clock = 1, si verifica il fenomeno della trasparenza del bistabile

Architettura degli elaboratori - 58 - Il clock e i bistabili

58

Un elaboratore ha elementi che calcolano ed elementi che memorizzano i calcoli

- Schema base semplificato di un circuito che elaboratore



Architettura degli elaboratori - 59 - Il clock e i bistabili

59

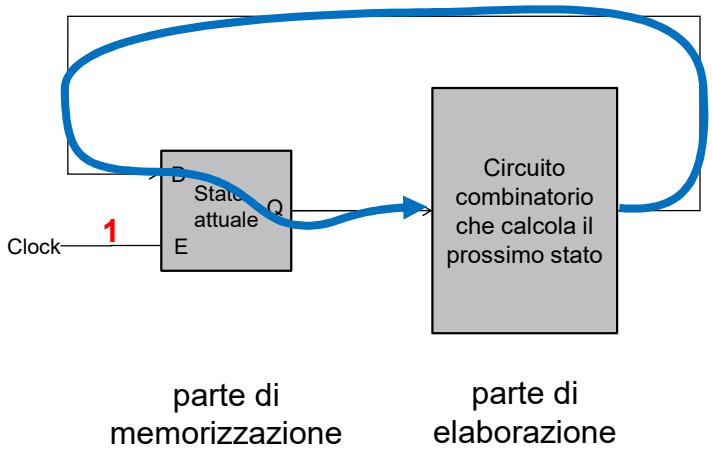
Effetti negativi della trasparenza

Struttura generale di un elaboratore

- Un insieme di bistabili memorizza lo stato attuale dell'elaboratore
  (un bit per ogni bistabile), e lo rende disponibile all'elaborazione.
- Un insieme di circuiti combinatori come la ALU
  calcola un passo della computazione:
  prendono in input lo **stato attuale**
  (il segnale che proviene dalle uscite Q dei bistabili),
  e producono in output lo **stato successivo**,
  (che è rimandato alle entrate D dei bistabili, per essere memorizzato)
- ad ogni ciclo di clock, viene effettuato un passo di computazione:
  1. Elaborazione:
     i circuiti combinatori elaborano il nuovo stato
     Memorizzazione:
     i bistabili memorizzano il nuovo stato prodotto dai circuiti combinatori

Il problema della trasparenza

- Durante l'**intervallo alto**, i bistabili sincronizzati sul livello sono trasparenti
  e il circuito combinatorio è messo come in diretto contatto coi suoi stessi output



Effetti negativi della trasparenza 2/3

Il problema indotto dalla trasparenza è:

- Quando i bistabili sono in stato di trasparenza,
  (cioè per metà del ciclo di clock)
  la parte combinatoria dell'elaboratore (ad esempio, la ALU)
  ha in pratica i propri output
  collegati direttamente ai propri input.
- Soltanto quando il clock vale 0
  (cioè solo per l'altra metà del ciclo di clock),
  i bistabili sono congelati / non trasparenti,
  e i circuiti combinatori hanno gli input costanti,
  mentre se i loro output variano fino ad arrivare alla configurazione finale
  (per quegli input)

Effetti negativi della trasparenza 3/3

Quindi, se i bistabili sono «sincronizzati sul livello»:

- **Intervallo basso del clock:**
  i bistabili sono bloccati,
  e forniscono un input stabile al circuito di elaborazione (combinatorio).
  Il circuito combinatorio può quindi procedere ad elaborare i propri output,
  avendo i propri input tenuti ad un valore costante.
- **Intervallo alto del clock:**
  si memorizza l'elaborazione prodotta dai circuiti combinatori (ok!),
  ma poi i bistabili permangono in stato di trasparenza per un intero mezzo ciclo.
  In questo intervallo, il circuito combinatori è in pratica connesso in input ai
  propri stessi output, e non procedere davvero con l'elaborazione.

Totale: LA META' DEL TEMPO E' INUTILIZZABILE ai fini dell'elaborazione!

Certamente, è necessario che ci sia un momento in cui il risultato attuale della computazione viene immagazzinato; questo dovrebbe però essere il più breve possibile; idealmente, un istante.

Ci serve una modalità diversa di sincronizzazione.

Sincronizzazione «sul fronte»

In un bistabile **sincronizzato «sul fronte»**:

- gli ingressi sono efficaci solo nell'**istante** in cui il clock passa da 1 a 0
  - cioè al *fronte di discesa del clock*
  - l'istante che considiamo la fine (e l'inizio) di ogni ciclo di clock
  - (variante possibile: oppure quello di salita, quando il clock passa da 0 a 1)
- in *entrambi* gli intervalli alto e basso del clock il bistabile è congelato!
  - il bistabile non è *mai* in condizione di trasparenza
  - dunque l'uscita del bistabile può commutare solo (poco dopo a) il fronte in discesa del clock
  - l'uscita del bistabile resta invariata per tutta la durata del ciclo di clock (eccettuando il brevissimo tempo di commutazione del bistabile, che avviene all'inizio del clock)

Il Flip-Flop

FL1P-FL0P

- Un bistabile *sincronizzato sul fronte* è detto «**Flip-Flop**»
- Il flip-flop è l'unità standard per la memorizzazione di un bit in un computer
  - ad esempio: un registro da 16 bit = 16 flip-flop
- Nota: tutti i bit di memoria sono flip-flop … sincronizzati con il clock
  - lo stesso segnale di clock propagato su ciascuno!
- Vediamo ora come realizzare un flip-flop

- Avvertanza: in alcuni contesti (specialmente in passato), si chiama "Flip-Flop" qualsiasi bistabile, anche un D-latch o un SR-latch

Durata dell'intervallo di clock: una nota

- Il periodo di clock deve comunque durare abbastanza
  da consentire ai *tutti* i circuiti combinatori usati
  (che agiscono da una memorizzazione all'altra)
  di produrre il loro output definitivo
  (e presentarlo ai bistabili per la memorizzazione successiva)
  - se il tempo di commutazione eccede il periodo di clock,
    nei bistabili memorizzerebbero l'output delle rete combinatorie non ancora definitivo e quindi errato sbagliato: disastro.
- In pratica, occorre stimare attentamente la durata del tempo di commutazione dei circuiti combinatori e accertarsi che il clock abbia un periodo maggiore in ogni caso
  - Quindi: circuiti combinatori *più lenti* necessitano una frequenza del clock *minore*
  - L'obiettivo pratico di realizzare circuiti combinatori maggiormente veloci è proprio quello di consentire una maggiore frequenza di clock

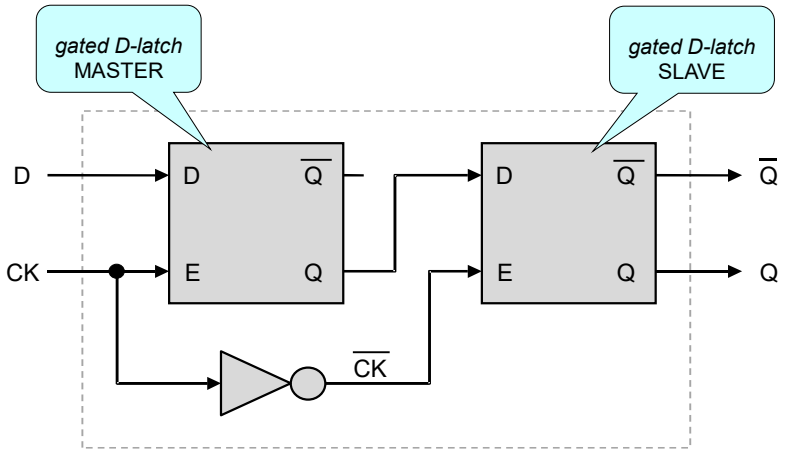
Overclocking e Underclocking

- L'**overclocking** di un processore:
  aumento della frequenza del clock
  al di sopra di quanto inizialmente previsto dai suoi progettatori
  - può avere conseguenze disastrose!
  - I circuiti combinatori potrebbero non aver più il tempo sufficiente a produrre i segnali definitivi da memorizzare
- **Underclocking**, o **downclocking**:
  diminuzione della frequenza di clock
  - può essere utile: il computer va ad un ritmo più lento, ma consuma meno
  - Ricorda: i gate logici CMOS consumano (quasi) solo quando cambia il loro input (e quindi output)
  - Durante l'ultima parte del circuito di clock, i segnali sono invariati
- Esempio: molti laptop sono progettati in modo da entrare in modalità di downclocking quando la batteria residua è bassa e non è disponibile l'alimentazione di corrente

Implementare il sincronismo sul fronte
I flip-flop “master-slave”

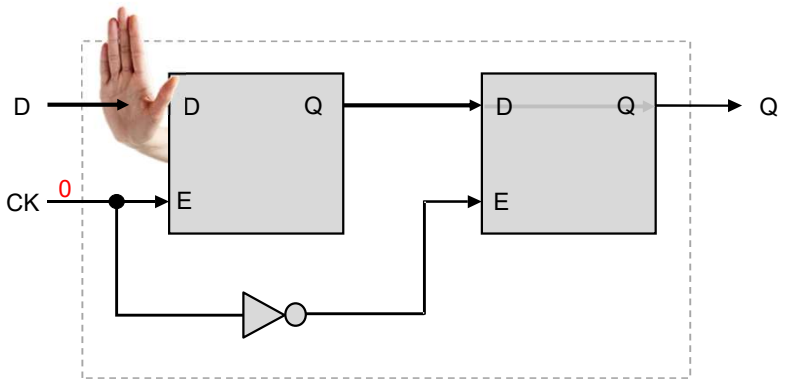
- I flip-flop cosiddetti “master-slave” (o a “memoria ausiliaria”) sono i più usati per realizzare il sincronismo sul fronte
  - rappresentano un ottimo compromesso tra complessità, costo e robustezza.

Flip-flop master-slave: schema



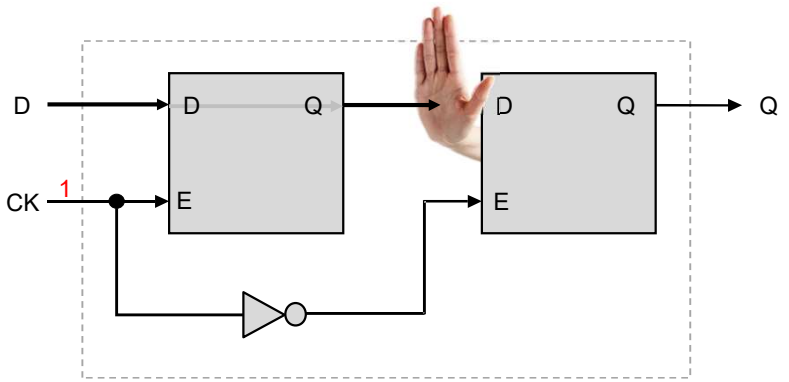
- Due *bistabili D con gate,* in cascata, sincronizzati su livello alto e basso risp.; l’insieme dei due non si trova mai in condizione di trasparenza!

Flip-flop master-slave nell'intervallo basso (1° metà del ciclo)



non trasparente

Flip-flop master-slave nell'intervallo alto (2° metà del ciclo)



non trasparente

Il Flip-Flop master/slave non è trasparente

- Il flip-flop, nel suo complesso, non è mai in stato di trasparenza:
  - nell'intervallo basso, il D-latch master è bloccato
  - nell'intervallo alto, il D-latch slave è bloccato
  - esattamente uno dei due è sempre bloccato
  - l'input del Flip-Flop non è mai propagato direttamente fino all'output
- Nota: a rigor di termini, i due latch sono entrambi bloccati per un brevissimo intervallo di tempo al fronte di discesa del clock, e entrambi sbloccati per un brevissimo intervallo di tempo al fronte di salita del clock, (a causa il tempo di commutazione della porta NOT), ma non è mai in stato di doppia trasparenza abbastanza a lungo da permettere ad un segnale di propagarsi dall'input all'output del flip-flop

Come il Flip-Flop master/slave implementa la **Sicronia sul Fronte** (di discesa)

- Quando si passa dall'intervallo alto a quello basso del clock (**fronte in discesa**) il latch master passa da *trasparente* a *bloccato*:
  - Il segnale che il latch master sta memorizzando *in quel preciso istante,* cioè il valore (0 o 1) presente nel suo input in quel momento, diventa il suo stato definitivo del master per l'intero il ciclo basso del clock
  - quello stesso segnale arriva (quasi) subito all'uscita del latch master, e di qui trova la "strada aperta" fino all'uscita dell'intero flip-flop, dato che lo slave è appena entrato in stato di trasparenza (e vi permarrà per tutto il ciclo basso)
  - Molto presto (dopo un breve tempo di commutazione dei latch) il segnale si sarà dunque propagato anche attraverso lo slave, fino all'uscita del flip-flop
- Se, durante il ciclo di clock che è appena iniziato, si presenterà un valore diverso all'entrata del flip-flop, questo verrà fermato dal master (nella prima metà del ciclo) o dallo slave (nella seconda), e comunque non arriverà all'uscita

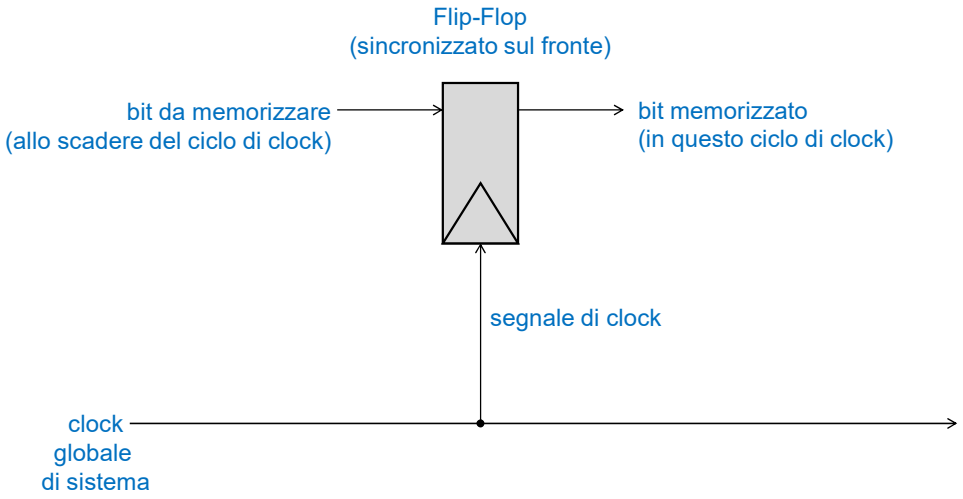
Risultato: il Flip-Flop memorizza il segnale preso nell'istante in cui scatta il fronte di discesa, e lo mantiene stabile in output per l'intero ciclo di clock.

Detto con una metafora

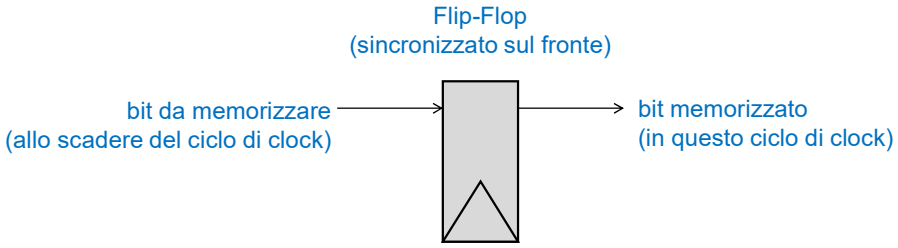
- Il latch master è un ufficio aperto di “giorno” e chiuso di “notte”. Lo slave: viceversa.
- Ad ogni “tramonto”, il master chiude improvvisamente il suo “portone d’ingresso”.
- Il valore (0 o 1), presente *in quel preciso istante* all’ingresso del master (e quindi del flip-flop), diventa quindi lo stato finale del master, per tutta la notte
- Si propaga (quasi) subito fino all’uscita del master
- Di qui, dato che è ancora notte (è appena iniziata!), quello stesso segnale trova lo slave ”aperto” (cioè in stato di trasparenza), e lo attraversa fino all’uscita, raggiungendo quindi l’uscita dell’intero flip-flip
- Quindi, pochissimo dopo il tramonto (giusto dopo il breve tempo di commutazione dei due latch) il segnale avrà percorso tutto il flip-flip dall’entrata all’uscita
- Se, durante quella stessa notte, si presenterà un valore diverso all’entrata del flip-flop, questo verrà fermato dal master, che è chiuso fino all’alba
- Ma all’alba, quando finalmente il master apre i battenti, si troverà ancora bloccato dallo slave, che rimane chiuso tutto il giorno e apre solo al tramonto successivo.

Risultato: il Flip-Flop memorizza il segnale preso nell’istante del tramonto, lo propaga (quasi) subito all’output, e lo mantiene stabile in output fino al prossimo tramonto

Flip-flop: rappresentazione come blocco funzionale



Flip-flop: rappresentazione come blocco funzionale



Maggiore astrazione:
spesso, ometteremo il clock e il suo segnale
(che è implicitamente diramato a tutte i blocchi che presentano un triangolo).
Il triangolo significa «input di sincronizzazione connesso al clock globale».

SR-Latch, D-Latch, flip-flop: velocità e costo

- Quanto tempo passa da quando si presenta un input ad un latch (attivo, non bloccato), a quando questo input viene (memorizzato e) presentato alla sua uscita?
  - Cioè: qual è il *tempo di propagazione* di un latch / flip-flop?
  - Per rispondere, basta osservare quante porte debbano essere attraversate.
  - Se (semplificando) il tempo di propagazione di ogni porta (AND, NOR, …) è τ, allora:
    - Latch SR, tempo di progazione basso: 2 τ
    - Latch D, di propagazione più elevato: 3 τ
    - Flip-flop, ancora più elevato: 6 τ

tau

- Quanto sono cari da realizzare questi circuiti?
  - Per rispondere in modo approssimativo, basta contare le porte logiche utilizzate (ma le NOT possiamo non contarle).
  - Semplificando:
    - Un SR-latch: 2 porte
    - Un D-latch: 4 porte
    - Un flip-flop: 8 porte

Conclusione: i flip-flop sono bit «di lusso»: più cari e più lenti rispetto ad altri semplici latch

Riassumendo: questi circuiti hanno *uno stato*. Memorizzano 1 oppure 0. Ma…

- Latch SR
  - **Asincrono**
  - Ottimo per comunicazione fra periferiche e unità centrali
  - Non adatto per memorizzare l'esito di una computazione (è sempre «in ascolto», memorizzerebbe anche i risultati intermedi e temporanei)
- Latch D
  - **Sincronizzato sul livello**
  - Problema della trasparenza (è trasparente per tutto l'intervallo alto del clock)
- Flip-Flop
  - **Sincronizzato sul fronte**
  - Non ha il problema della trasparenza
  - Il nostro modo standard di realizzare «un bit di memoria»

88

Domande ed esercizi

- Prova a disegnare il diagramma temporale di un SR-latch che, inizialmente in stato 0, riceva un breve impulso S = 1.
  Includi tutti i canali (ingresso, uscita, retroazioni etc – sono 4 in tutto).
  Ricorda di considerare il tempo di commutazione delle porte.
  Riporta anche i rapporti di causa-effetto fra i fronti
- Se il mio circuito combinatorio richiede un tempo di calcolo di 0.3 nanosecondi, quale di queste frequenze di clock posso adottare (usando flip-flip per memorizzare il suo output)?
  - 10 MHz
  - 200 MHz
  - 2 GHz
  - 4 GHz

89