(v. PHRV1, appendice A.7, A.8, A.10)

Lezione 5: Reti Logiche Sequenziali Elementari

- (richiamo) Reti COMBINATORIE (RC o CN==Combinatorial Network)
  - In qualunque istante
    - le uscite sono funzione del valore che gli ingressi hanno in quell'istante
  - Il comportamento (uscite in funzione degli ingressi) è descritto da una tabella
  - NON CONTENGONO ANELLI DI RICHIUSURA

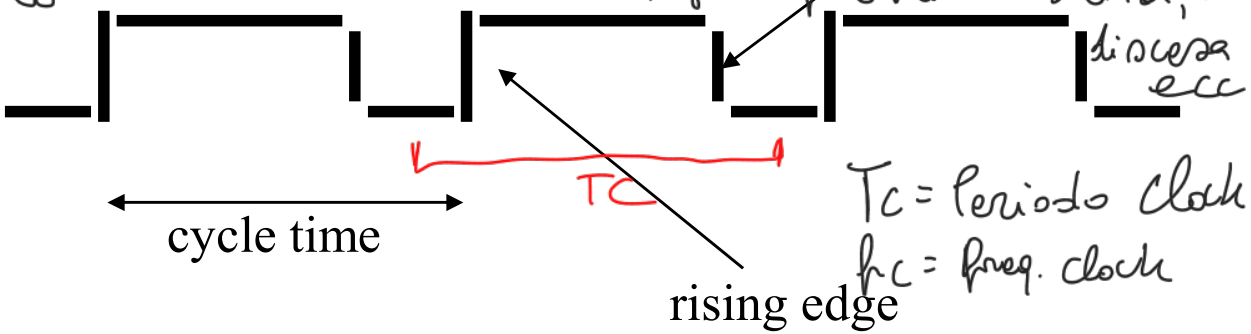
- (richiamo) Reti SEQUENZIALI
  - In un determinato istante
    - le uscite sono funzione del valore che gli ingressi hanno in quell'istante
    - e dei valori che gli ingressi hanno assunto precedentemente
  - Presentano Stati Interni (sono quindi reti dotate di MEMORIA)
    - → La descrizione è più complessa di una semplice tabella
  - PRESENTANO ANELLI DI RICHIUSURA

Reti Sequenziali Asincrone e Sincrone

- Le Reti Sequenziali possono essere di tipo
  - asincrono (non richiede un segnale di clock)
  - sincrono (richiede un segnale di clock)

- Definiamo quindi cosa è un SEGNALE DI CLOCK
  - Il segnale di clock (o semplicemente «clock») è un segnale periodico* (idealmente un'onda quadra) che serve per fornire un riferimento temporale ben preciso per effettuare le operazioni

Ad esempio si può prendere come riferimento il fronte in salita (rising edge) o quello in discesa (falling edge)



* Nota: caratterizzato quindi da un periodo Tc (cycle time) ovvero dalla sua frequenza fc=1/Tc

Nota2: Il segnale di clock viene tipicamente generato da un circuito elettronico chiamato "oscillatore" che usa la risonanza meccanica di un quarzo per generare una forma d'onda a una data frequenza

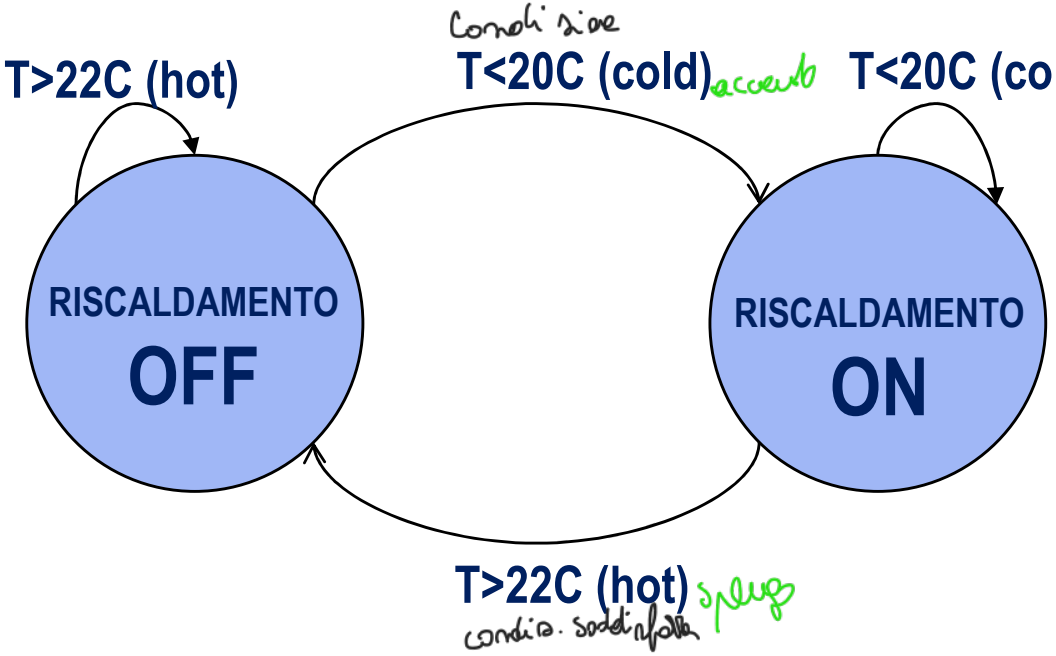
- Dobbiamo inoltre definire l'elemento di MEMORIA: per fare questo ci serviamo delle Macchine a Stati Finiti (FSM)

Possiamo implementare una Rete Seq. Sincrona con le

Macchine a Stati Finiti (o FSM==Finite State Machine)

Macchine a Stati Finiti

- Un gruppo di stati e di transizioni



- Ci si sposta da uno stato all'altro seguendo una linea di transizione SE la condizione dalla transizione è soddisfatta

Macchine stati FINITI è un gruppo di Stati e Transizioni

Macchine a Stati Finiti (2)

- Nel campo dell'architettura dei calcolatori, le FSM sono molto importanti in quanto godono delle proprietà di essere:
  - **complete**
    → tutti gli stati definiscono transizioni per tutti gli input
  - **deterministiche**
    → se si presenta un dato ingresso seguo sempre un dato arco

- Inoltre, le FSM possono essere implementate fisicamente tramite Reti Logiche dette **Reti Sequenziali**

- Le Reti Sequenziali possono avvicinarsi al comportamento ideale di una FSM facendo in modo che le transizioni fra uno stato ed il successivo avvengano in intervalli di tempo molto limitati:
  - o utilizzando particolari accorgimenti costruttivi (riduzione ALEE*)
  - o effettuando le transizioni su un fronte del clock

*Nota: la motivazione di rendere sensibili le reti agli ingressi solo in intervalli molto limitati risiede nella necessità evitare che oscillazioni temporanee dell'uscita si propaghino agli ingressi di altre reti generando ALEE (v. slide successive).

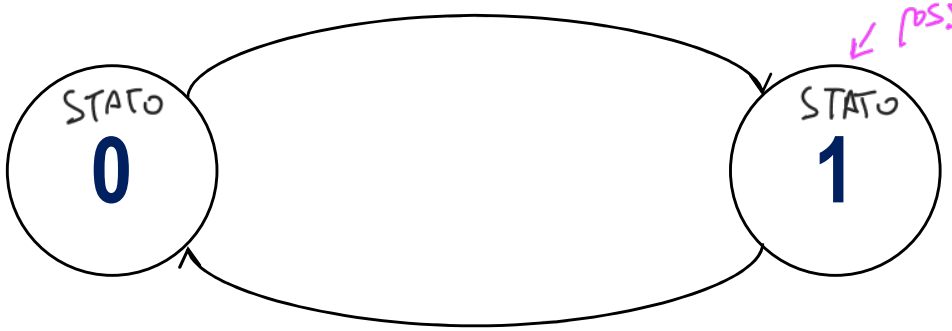
SENSIBILE solo su INTERVALLI LIMITATI (appena c'è una TRANSIZIONE DI STATO AL FRONTE CLOCK)

L'elemento di memoria

Voglio memorizzare da qualche parte un bit

- La FSM più semplice che si possa immaginare (e che varia il proprio stato) è una rete a due soli stati 0 e 1
  - Tale rete non rappresenta altro che un bit di informazione e quindi permette di implementare un elemento di memoria

archi = transizioni tra 0 e 1



posso cambiare tra gli stati

bit: elemento di memoria che varia tra 0 e 1 può cambiare stato tra 0 e 1

la modello come

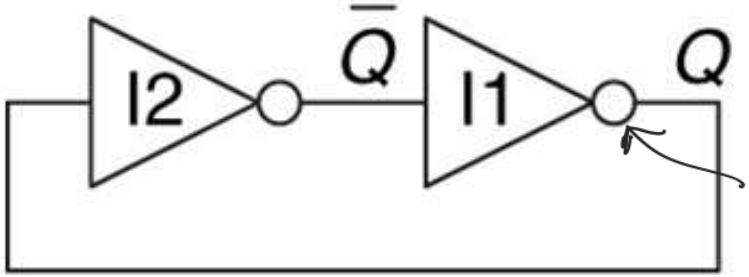
Rete Sequenziale, elemento di Memoria / macchine Stati Finiti con 2 Soli Stati

Posso realizzare un elemento memoria inverter, buon candidato

Realizzazione di elementi di memoria con reti combinatorie

- Abbiamo visto che l'elemento più semplice che riusciamo a realizzare facilmente su silicio è **un** inverter… con 2 MOS, un PMOS e NMOS
- Il modo più semplice per realizzare un elemento di memoria usando inverter è di utilizzare **due** inverter… richiusi ad anello

2 inverter in cascata e chiusi ad anello, rete sequenziale



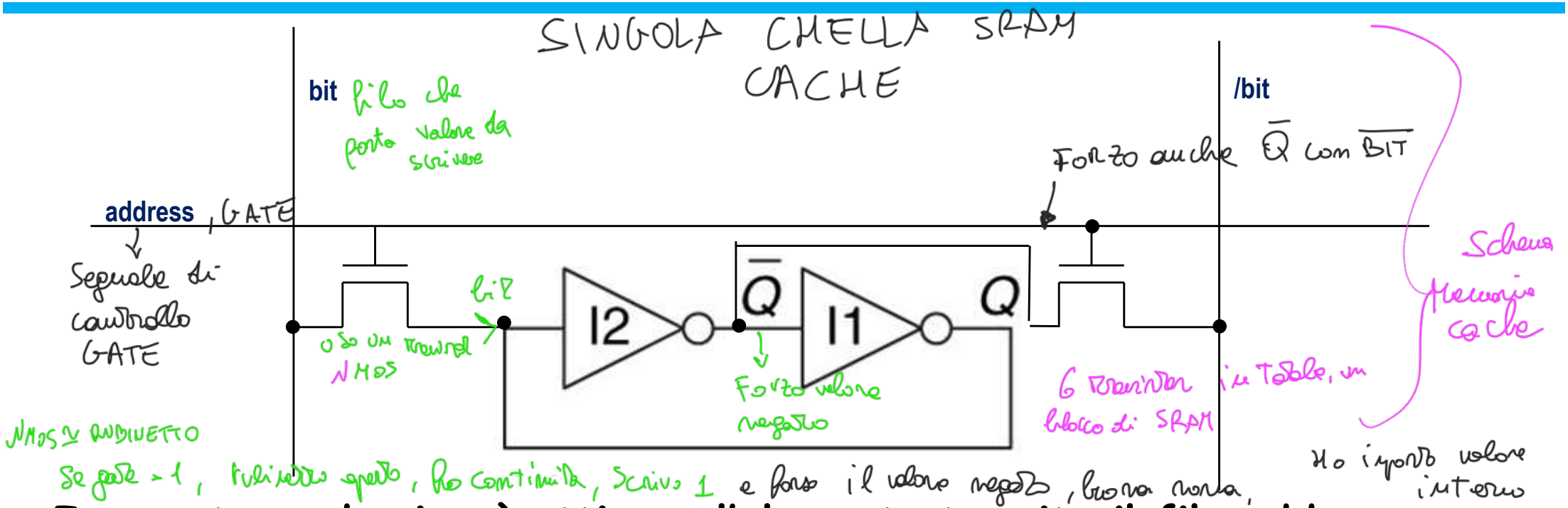
Se in Q metto 1, dopo I2 è 0, e dopo I1 c'è ancora 1

- Questo circuito digitale presenta DUE STATI stabili (si dice pertanto che è **bistabile**):
  - i) Q=0 (e di conseguenza /Q=1)
  - ii) Q=1 (e di conseguenza /Q=0)

Memorizza 1 bit - commuta tra 0 e 1

- Però presenta solo due uscite e **nessun ingresso**
- Se ben dimensionato fisicamente funziona, altrimenti potrebbe anche oscillare (comportamento **metastabile**)

Perfezionando il bistabile usando la tecnologia CMOS



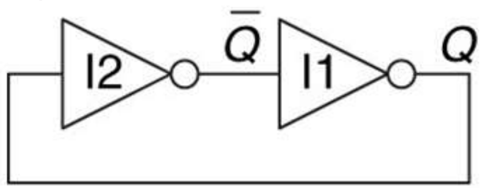
- **In questo modo si può attivare l'elemento tramite il filo address e:**
  - **Imporre un 1 logico (bit=1,/bit=0) oppure uno 0 logico (bit=0,/bit=1)**
  - **Oppure leggere il valore Q senza distruggere il contenuto**
- **Questo tipo di cella di memoria viene usata per realizzare le SRAM (Static RAM)* → l'informazione rimane memorizzata fintanto che la cella è alimentata**
  - **In tecnologia CMOS si realizza con 6 transistor (2 per ogni inverter e due per i due transistor di passo)**

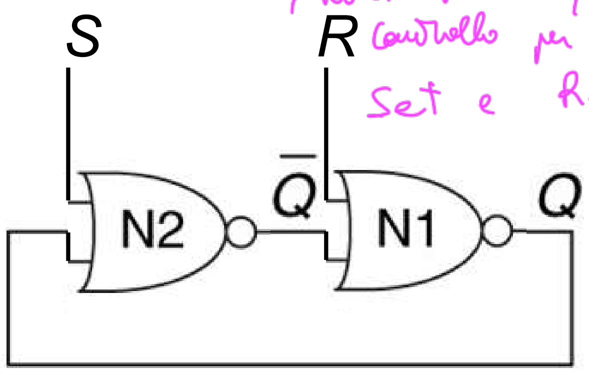
Cella non indipendente, devo pilotare address e imporre bit e /bit

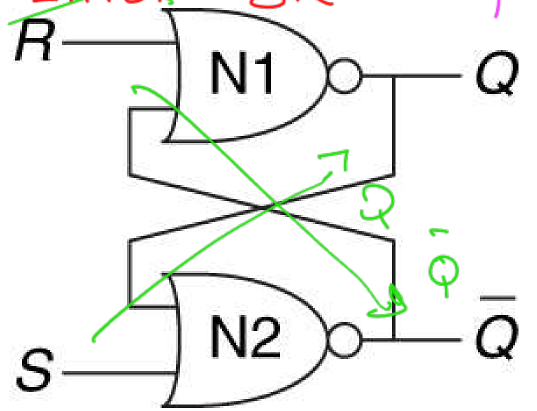
*Nota: approfondiremo le SRAM nella lezione 16, dedicate al sottosistema di memoria

Latch SR

- L'elemento di memoria realizzato con due inverter è in grado di memorizzare l'informazione ma non ha una forma "autocontenuta" per memorizzare il dato (bit e /bit devono essere collegati solo in certi istanti, via address)
- Posso allora utilizzare porte NOR (anziché porte NOT) e usare uno dei due ingressi per modificare lo stato (analoghi a /bit e bit della singola cella) realizzando così il cosiddetto latch SR







- Nota - chiameremo:
  - **Latch**: elementi che variano lo stato su un livello di un segnale di clock
  - **Flip-flop**: elementi che variano lo stato su un fronte di un segnale di clock

fronte di clock, più stabile

Elemento di memorizzazione in logica sequenziale asincrona

- Il latch SR
  - l'uscita dipende dai segnali presenti sugli ingressi e dai segnali precedentemente presentati in ingresso

Latch vengono caratterizzate con una Tabella verità che ne spiega comportamento

Tabella deve essere implementabile e verificabile

DIVERSA da Tabella di verità semplice, uscita di mealy da stato precedente

Tabella verità Latch SR

S	R	Q	/Q
0	0	Q	/Q
0	1	0	1
1	0	1	0
1	1	--	--

effetto reset

Uscita condizionata da stato prec. Se ingressi 0, conservo stato prec.

Voglio uscita 0

Voglio imporre uscita = 1

non specificato, Si dice che "conserva"

il circuito può oscillare

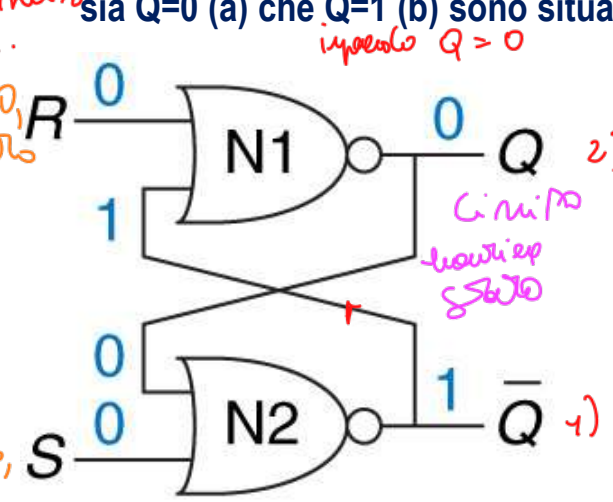
CASO FASTIDIOSO

SET RESET

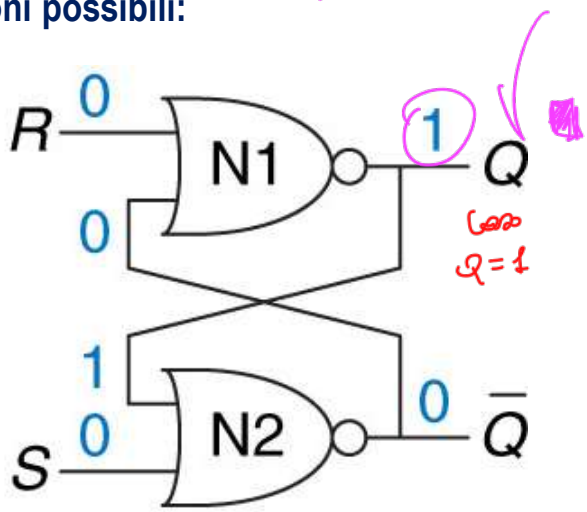
IL LATCH CONSERVA

Come si vede da questo esempio per SR=00, sia Q=0 (a) che Q=1 (b) sono situazioni possibili:

Verifico



(a)



(b)

Tabella di Verità

Se S e R = 0, conserva

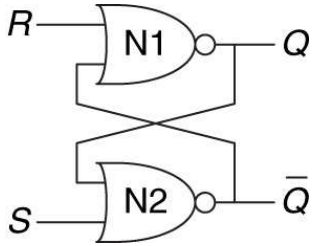
Schema logico

evitabile 1 Q 12?! caso limite

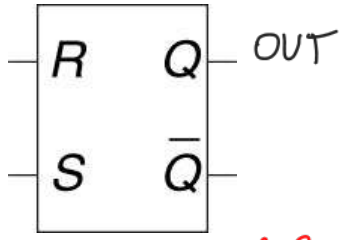
Latch SR in Verilog

modello con ritardo, oggetto reale, ritardi di propagazione

```
module Latch_SR(Q,QN,S,R)
   input S,R;
   output Q,QN;
   assign #1 Q=~(R|QN);
   assign #1 QN=~(S|Q);
endmodule
```

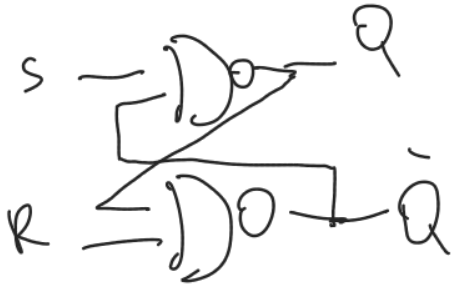


Simbolo →



Latch base Simbolo con quadratino

Rete sequenziale in Verilog, modelliamo ritardo



ogni NOR ha bisogno di 4 transistor, 2 nMOS in parall, e 2 pMOS serie

Transistor necessari (8 transistor=4x2NOR)

S-R cloccato

ANTEPONGO ad S e R un segnale di clock,

Il Latch SR è sempre sensibile ad ogni variazione di S e di R: per fare in modo che S e R vengano ascoltati solo in determinati istanti si può usare un segnale di abilitazione Ck

CLOCK COME CHIAVE — porte controllate dal clock

ascolto S e R quando clock è a 1

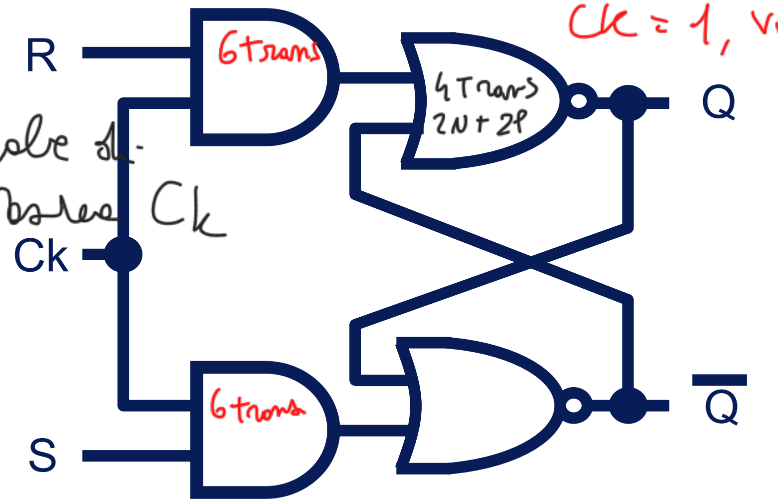
L'SR-cloccato, sente S e R solo quando il segnale Ck (clock) vale 1

Clock come chiave, porte AND bloccate => stato conservazione

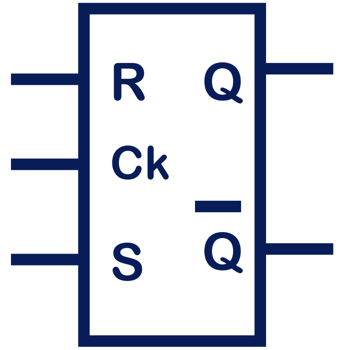
Ck	S	R	Q	/Q
1	0	0	Q	/Q
1	0	1	0	1
1	1	0	1	0
1	1	1	---	---
0	X	X	Q	/Q

Clock=1 vecchio comportamento; Clock=0 conservo; stato conservazione; controllo; se Ck=1

Tabella di Verità



Schema logico



Simbolo

Segnale di abilitazione Ck

Ck = 1, vecchio comportamento

quando clock è 0, le porte sono bloccate, conservano

se clock = 1, lo vecchio comportamento

Più controllabile il LATCH

AND: costa tra 4 NAND o inverter 2 ≈ 6 transistor

Layout simbolico: 20 transistor (=4x2NOR+6*2AND)

COSTO, NOR = 2 Nmos + 2 Pmos   AND = NOT e NAND in cascata, = 2 + 4 (2+2) = 6

S-R edge triggered

S-R edge triggered è un FF, sensibile sui fronti — Caso Flip Flop — sensibile sul fronte di discesa

Metto in cascata 2 SR clockati, incrocio i Q, e Q2 e nego il secondo FF

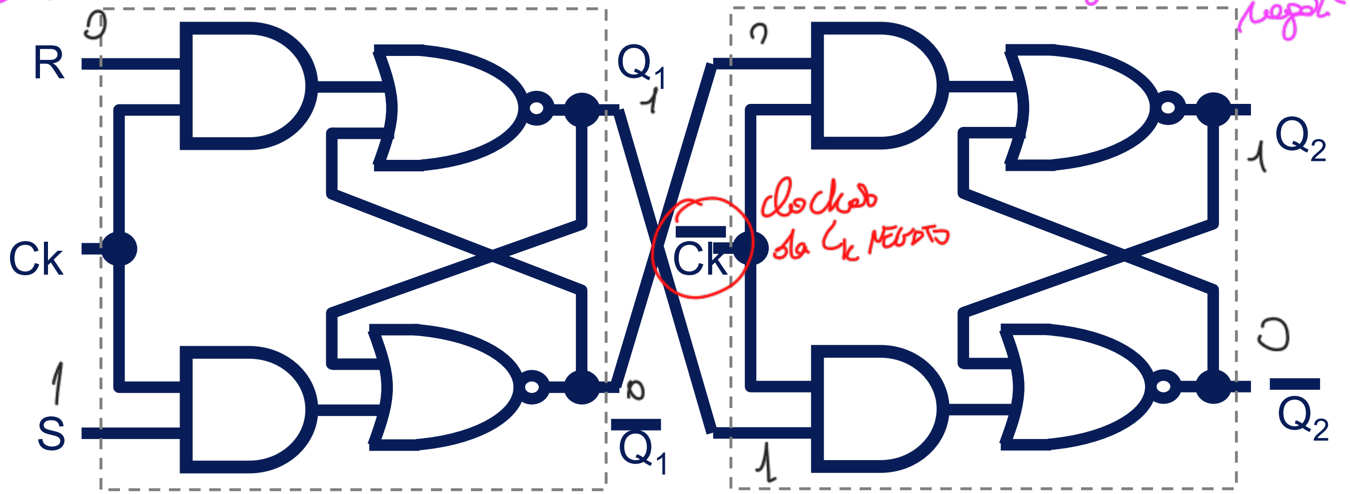
In questo caso vogliamo sensibilità solo sul fronte in discesa

2 S-R clockati in cascata, incrociando uscite e ingressi e 2° clock negato

Ck	S	R	Q	/Q
↓	0	0	Q	/Q
↓	0	1	0	1
↓	1	0	1	0
↓	1	1	---	---
0	X	X	Q	/Q
1	X	X	Q	/Q
↑	X	X	Q	/Q

Tabella di Verità

Sui fronti in salita, sono con Ck = 0 inizialmente, primo latch conserva, e lo passa al 2° che ora è comunque = 0 (negato, Ck forza 1)

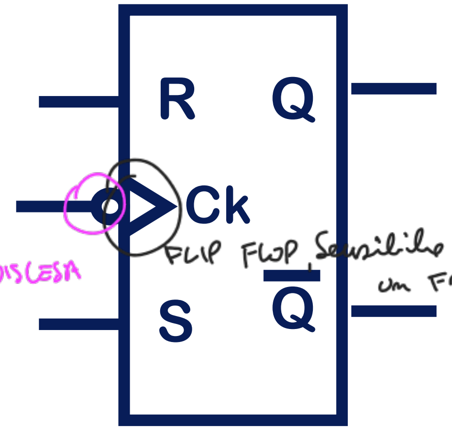


Schema logico

Fronte salita conserva, il primo latch è bloccato e Q1 e Q2 si conservano

Layout simbolico: 42 transistor
4*NOR+4*(NAND+NOT)+ 2 per /Ck=4*4+4*(4+2)+2

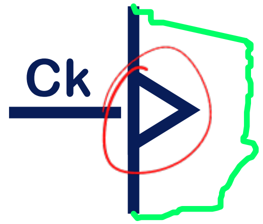
2 trans. per invertire Ck



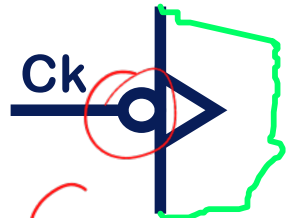
Simbolo

Sensibile fronte discesa — Flip Flop sensibile su un fronte

Può essere:



Sensibile sul fronte in salita



Sensibile sul fronte in discesa

Fronte in salita $C_k = 0$ (primo LATCH bloccato, conservo)

qualsiasi cosa ci sia, viene passato al 2° Latch lo stato precedente.

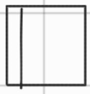
Dopo nel primo flip ~> le cose cambiano

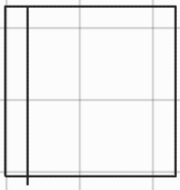
$C_k = 1$, 2° flip è chiuso, $\bar{C}_k = 0$ anche se è variato il primo, l'uscita non lo sente

~> Fronte in discesa

C_k = 1° LATCH ascolta, CAMPIONA UN ISTANTE PRIMA della DISCESA, e lo passa lo STATO al 2° Latch che ora è aperto perché $\bar{C}_k = 1$ ora, lo passa in uscita

Sensibilità solo sul fronte in discesa

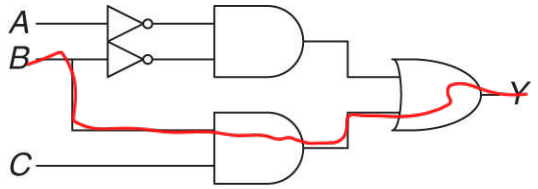
Se so che è un flip flop, noto = 



Complicaz. uscite quando varia ingresso

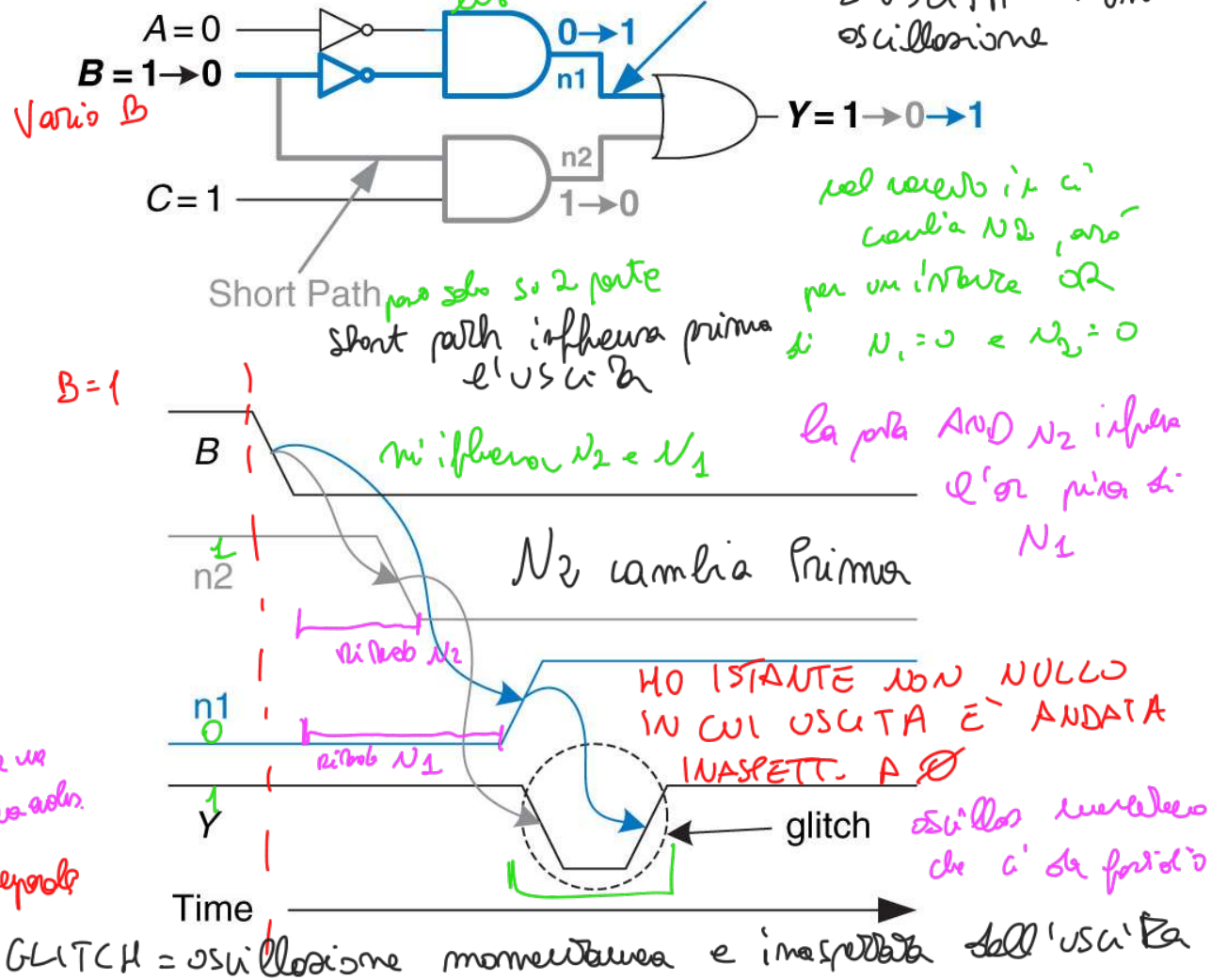
Alee (Glitches/Hazards)

- **Può accadere che variando un ingresso, si creino variazioni multiple dell'uscita**
- **Esempio: rete con alee**



Y	AB 00	01	11	10
C = 0	1	0	0	0
C = 1	1	1	1	0

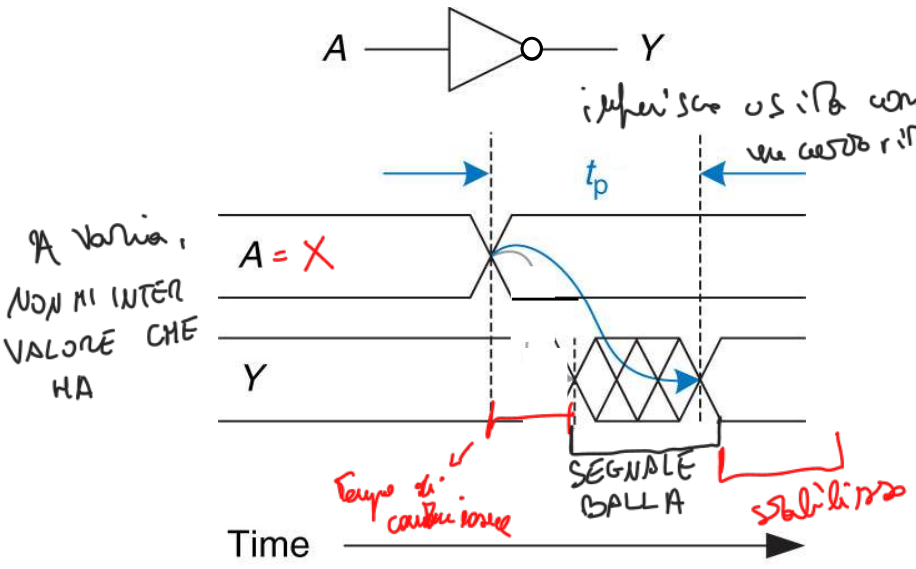
$Y = \overline{A}\,\overline{B} + BC$



→ **È importante essere in grado di analizzare il comportamento temporale di una rete logica (timing)**

Timing

- Finora abbiamo analizzato solo l'aspetto funzionale della rete senza tenere conto dei tempi di propagazione del segnale fra ingresso e uscita (behavior)
- Abbiamo però visto come si possa calcolare il tempo di propagazione per un semplice inverter CMOS



- Il tempo di propagazione t_p è il tempo necessario affinché l'uscita sia stabile dal momento in cui ho variato l'ingresso
- Si definisce invece tempo di contaminazione t_{cd} il tempo che intercorre da quando ho variato l'ingresso a quando inizia a variare l'uscita

Se pretendo di cambiare un segnale es. il fronte di un clock

tsetup e thold

devo preparare i miei segnali con un certo anticipo T_{setup} e poi devo mantenere per un altro tempo di tempo T_{HOLD}, se voglio variare devo

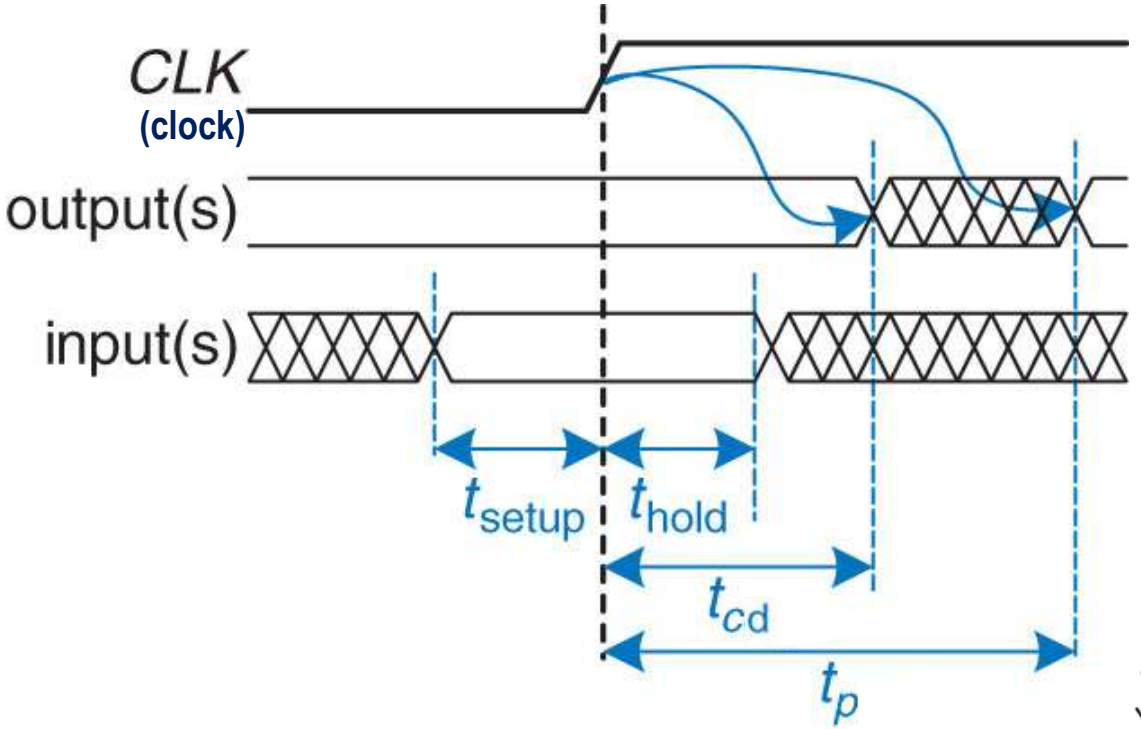
- **I segnali di ingresso SR devono essere stabili durante un fronte del clock (vanno preparati un po' prima → tsetup)**

su fronte di un orologio

- **Dopo il fronte si possono tranquillamente variare i segnali (dopo aver atteso thold)**

$T_{HOLD} < T_p$ e T_{cd}

Se pretendo



1) in una rete ho capito che se cambio un ingresso, devo attendere un certo tempo affinché l'uscita cambi

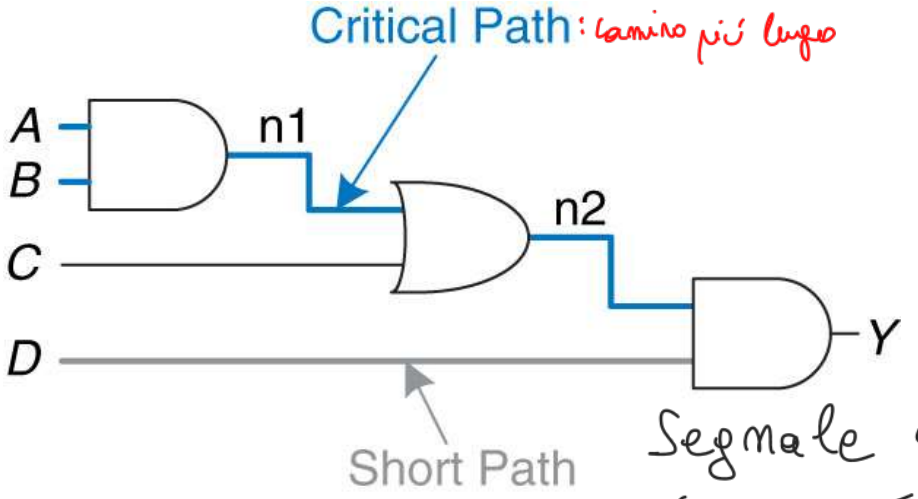
=> DEFINISCO tsetup e thold

Se pretendo di variare un segnale sul fronte di un clock devo PREPARLE PRIMA il mio segnale con UN CERTO ANTICIPO e devo ANCHE MANTENERLI per un certo tempo

- **Il tempo di propagazione tp è il tempo necessario affinché l'uscita sia stabile dal momento in cui ho variato l'ingresso**
- **Si definisce invece tempo di contaminazione tcd il tempo che intercorre da quando ho variato l'ingresso a quando inizia a variare l'uscita**

Critical path

- Oltre ai ritardi dovuti alla singola porta logica, quando si interconnettono più porte in cascata può accadere che determinati segnali giungano alle varie porte intermedie con ritardi diversi. Esempio:



- In particolare il cammino più lungo fra un ingresso e l'uscita è chiamato cammino critico (critical path)

Segnale che influenza direttamente uscita o comunque cammino più corto

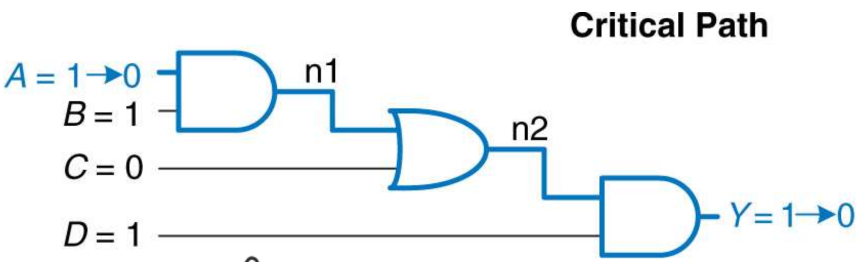
Calcolo dei tempi

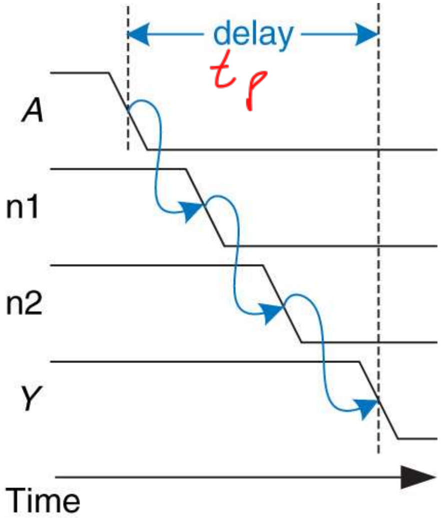
TEMPO Propagazione RETE = Somma RITARDI degli el. del CRITICAL PATH

TEMPO CONTAMINAZ: SOMMA TEMPI CONTAMIN. della SHORT PATH

- **Cambiando l'ingresso A da 1 a 0**

Ritardo tra inizio variazione e varia. uscita Y se cambio i miei ingressi





Cond. del cammino critico

$t_p = 2t_{p\_AND} + t_{p\_OR}$

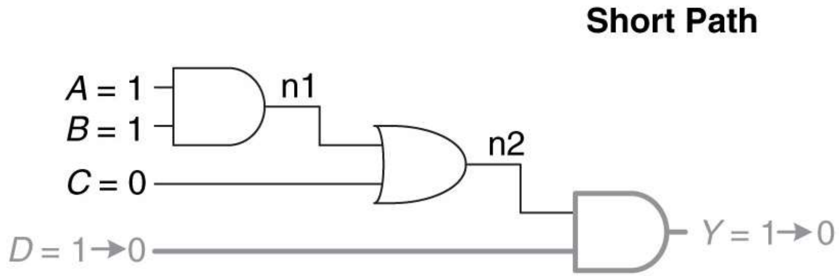
Tempo Propag = ritardi del path più lungo

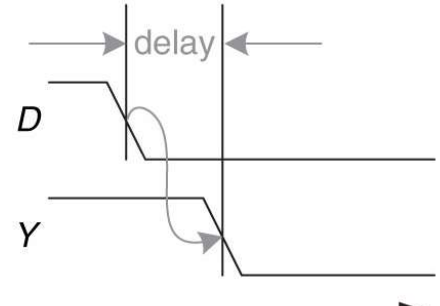
ritardo tra varia. ingresso e uscita

Metodi per calcolare tempi delle RETI COMBINATORIE il tp e tcd conoscendo i tempi delle singole porte

tempo propag: ritardo long path

- **Cambiando l'ingresso D da 1 a 0**





$t_{cd} = t_{cd\_AND}$

Tempo contaminaz = somma tempi nel cammino più corto

Ho dunque 2 metodi per calcolare i tempi

Tempo contaminaz. = Ritardo short path

Tempi di propagazione tipici

più ingressi, più ritardi

Gate	t_p (ps)
NOT	30
2-input AND (6 transistor)	60
3-input AND	80
4-input OR, AND ...	90
tristate (A to Y) (porta con ingresso di controllo uscita)	50
tristate (enable to Y)	35

Ho più ingressi ho più ritardi perché devo passare attraverso più porte

D-Latch , PIÙ STABILE dell'SR CLOCKATO

- Il latch SR è problematico perché si comporta in modo impredicibile quando SR=11 perche'?
- I segnali S e R definiscono sia cosa (memorizzare sull'uscita) che *quando* (clock) le uscite vengono cambiate
  - Il progetto dei circuiti digitali è più semplice se si definisce separatamente cosa cambiare e quando cambiarlo
  - Questo è esattamente cosa fa il D-latch

- D-latch
  - Il segnale D controlla il dato ovvero *cosa* deve essere l'uscita
  - Il segnale CLK controlla *quando* lo stato deve essere cambiato

però se un S-R clocked, metto un NOT in ingresso che vincola S e R

Separo cosa cambiare e quando

D-LATCH, FLIP FLOP SR con INVERTER tra S e R

D-Latch o D-Transparent

NOT in ingresso che VINCOLA S e R

- Ha due ingressi (D e Ck):
  - D rappresenta il dato (indica COSA verrà memorizzato)
  - Ck è il clock (indica QUANDO memorizzare)

sensibile al clock = 1, LATCH

D	Ck	Q
X	0	Q
0	1	0
1	1	1

CONSENSO

S=0 R=1

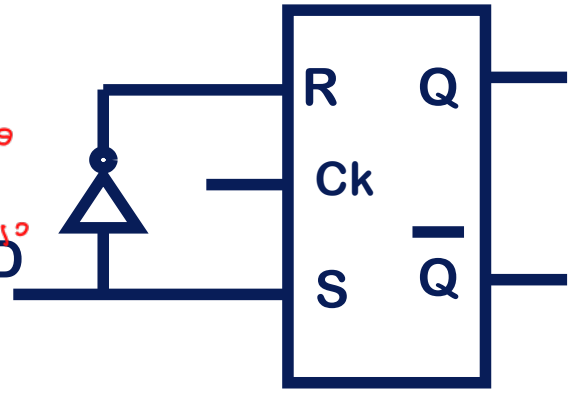
S=1, R=0

USCITA segue l'ingresso D

Tabella di Verità

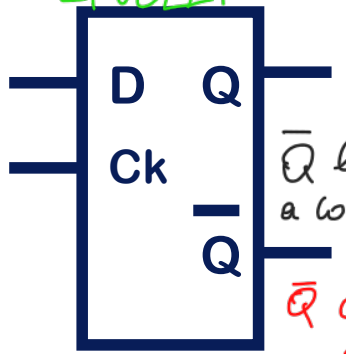
USCITA INSEGUE INGRESSO è TRASPARENTE

Con clock a 0, qualsiasi cosa accada in ingresso, consenso



Schema logico

Sono sensibile sui LIVELLI



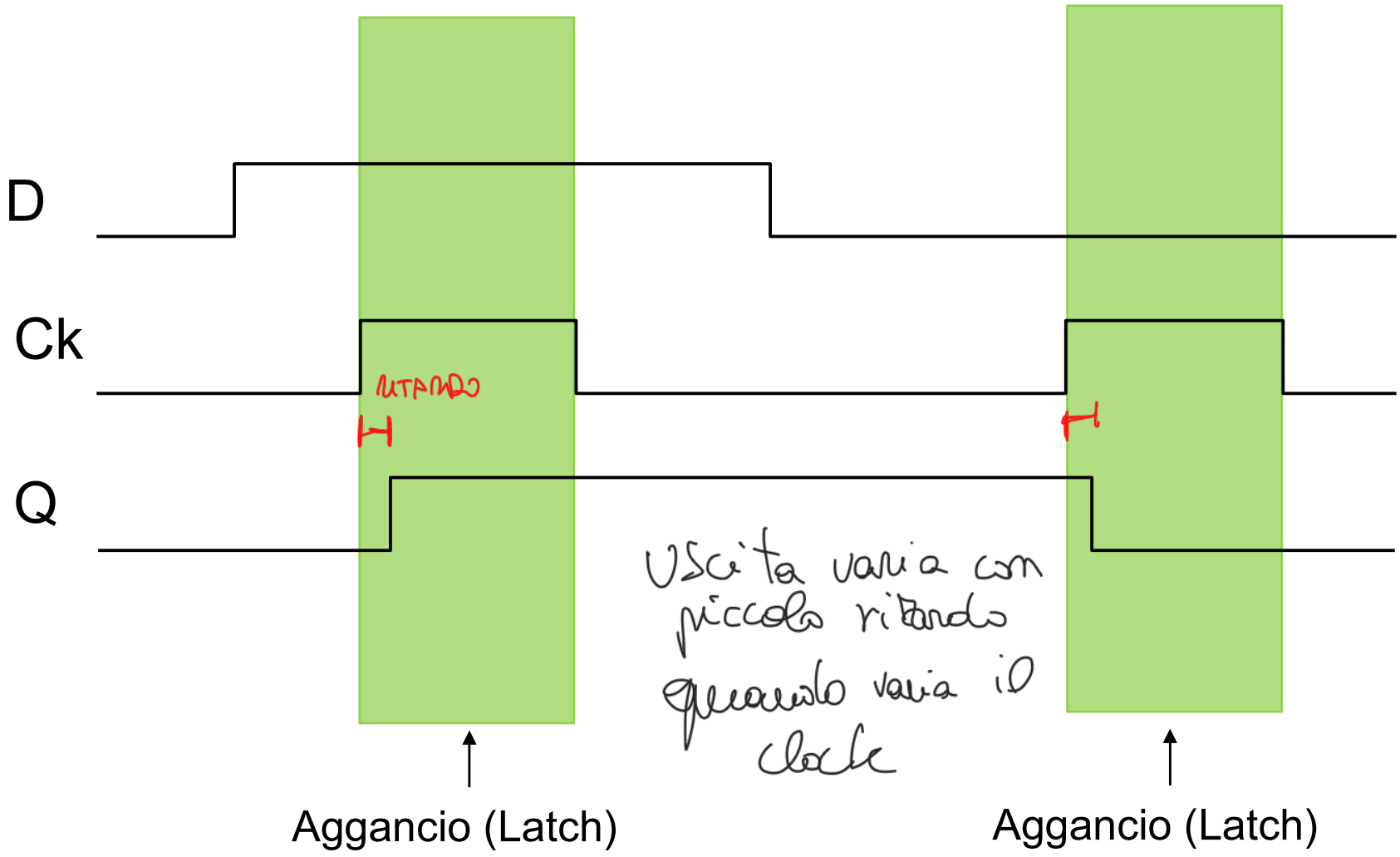
$\bar{Q}$ lo ho a costo 0

$\bar{Q}$ c'è a costo 0, posso usarlo in un punto diverso

Simbolo del D-Latch

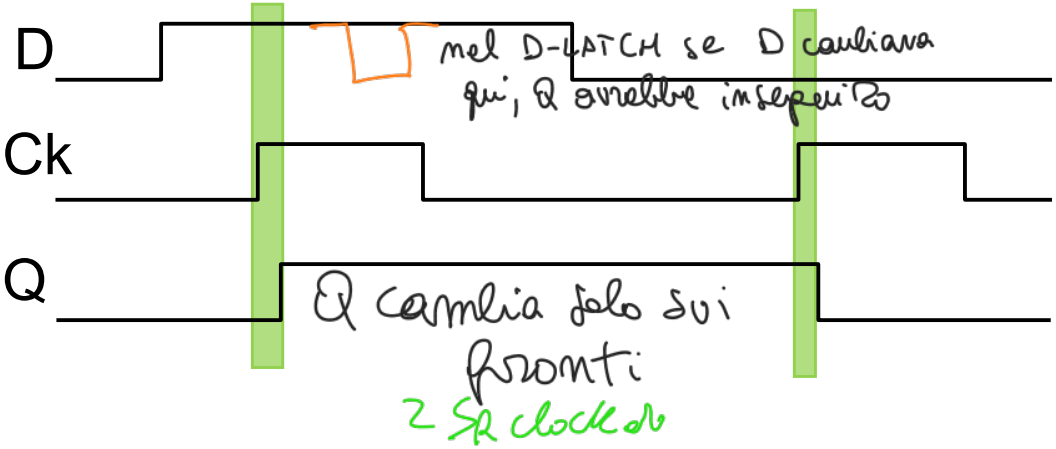
Layout simbolico: (22 transistor=20 SR-cloccato + 2 per NOT)

D-Latch o D-Transparent (esempio)



D-edge triggered

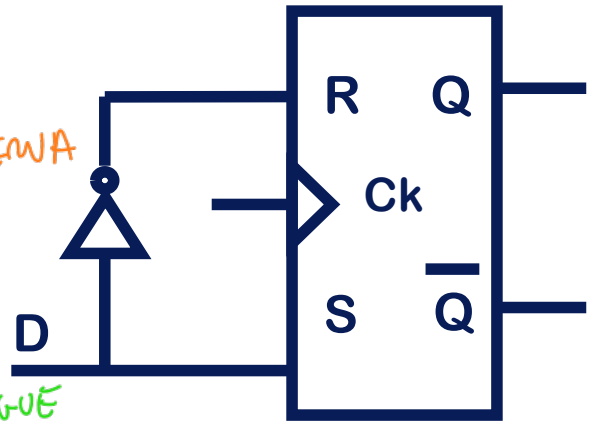
• L'uscita cambia solo sul fronte in salita del clock



Flip Flop commuta solo sul fronte di salita

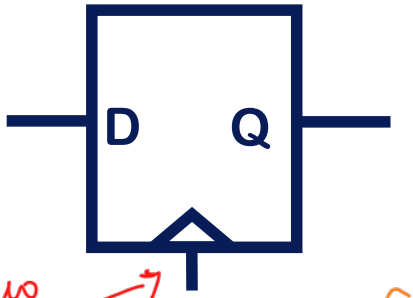
D	Ck	Q	
X	1	Q	CONSERVA
X	0	Q	
X	⊓	Q	
0	⊓	0	INSEGUE
1	⊓	1	

Tabella di Verità



Schema logico

Più stabile ma costa il doppio dei transistor



No Pallino → sensibile SALITA

Sono sensibile sul fronte

Simbolo del D-Edge-Triggered

Layout simbolico: 44 transistors
(42 per SR-ET + 2 per NOT)

FLIP FLOP D *negative edge triggered*

SPECULARE RISPETTO AL PRECEDENTE

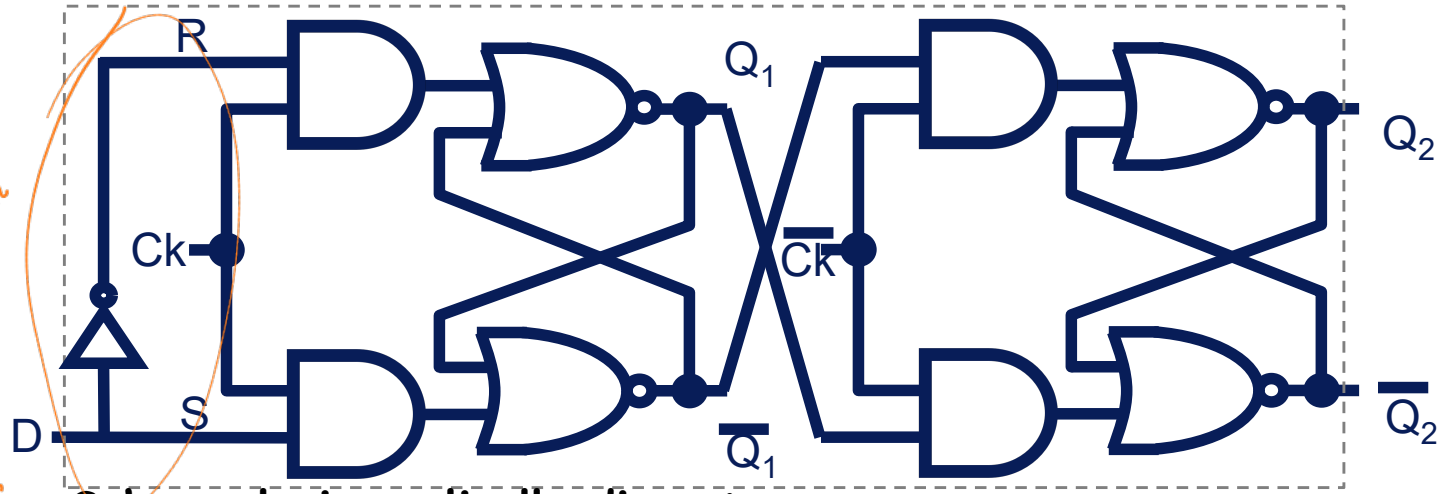
In questo caso vogliamo sensibilità solo sul fronte in discesa

D	Ck	Q
X	1	Q
X	0	Q
X	↓	Q
0	↓	0
1	↓	1

SR con NOT

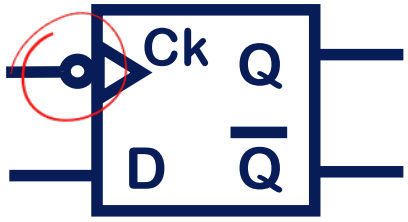
in due casi

Tabella di Verità



Schema logico a livello di porte

Layout simbolico: 44 transistor
=4*NOR+4*(NAND+NOT)+ 2 per /Ck
+ 2 per la not su D =4*4+4*(4+2)+2+2



Simbolo

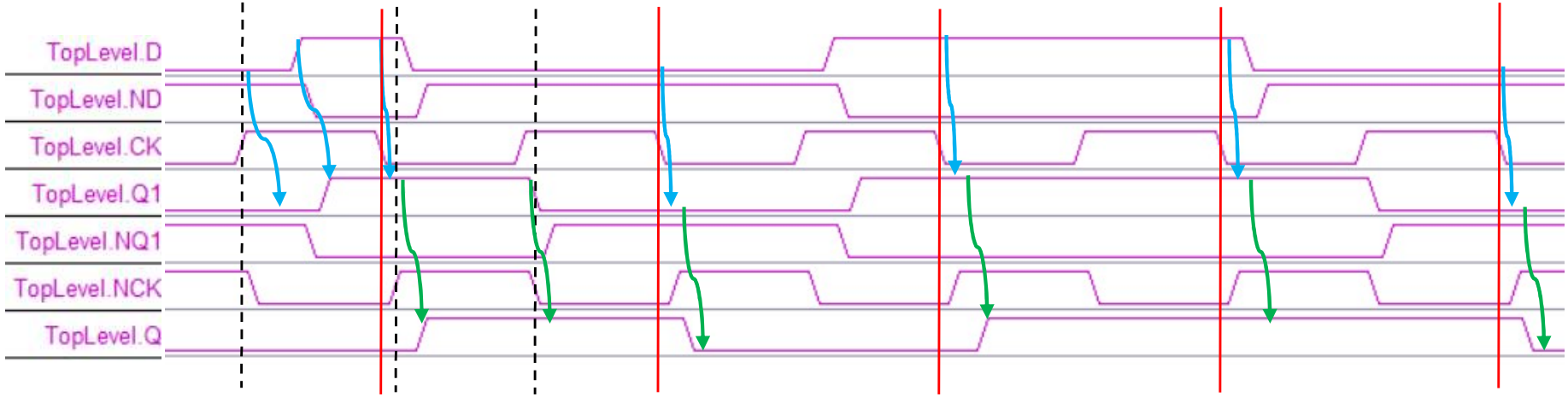


Diagramma Temporale

Realizzazione in Verilog

```
`timescale 1ns/1ps
module TopLevel;
 reg reset_; initial begin reset_=0; #22 reset_=1; #500; $stop; end
 reg clock;  initial clock=0; always #10 clock<=(!clock);
 reg D;
 wire ND = dsrms.r;
 wire  CK = dsrms.mysrms.ck;
 wire  Q1 = dsrms.mysrms.q1;
 wire NQ1 = dsrms.mysrms.qbar1;
 wire NCK = dsrms.mysrms.nck;
 wire Q;
 initial begin
   wait(reset_==1); D<=0; #32 D<=1; #8; D<=0; #30 D<=1; #30; D<=0; #1000
   $finish;
 end
 DET_SRMS dsrms(D,clock,Q);
endmodule
```

```
module CSR(s,r,ck,q,qbar); // Clocked SR
  input  s, r, ck;
  output q, qbar;
  wire   s1, r1, s2, r2;
  assign s2 = q;
  assign r2 = qbar;
  assign #1 qbar = ~(s1|s2);
  assign #1 q    = ~(r1|r2);
  assign s1      = s & ck;
  assign r1      = r & ck;
endmodule
```

```
module SRMS(s,r,ck,q);  // SR Master-Slave
  input s, r, ck; output q;
  wire  nck, q1, qbar1, q2, qbar2;
  assign #1 nck = ~ck;
  assign q = q2;
  CSR master( s,    r, ck,  q1,qbar1);
  CSR slave (q1,qbar1,nck,  q2,qbar2);
endmodule
```

```
module DET_SRMS(d, ck, q);  // D-Edge Triggered via SR Master-Slave
  input d, ck; output q;
  wire  s, r;
  assign s = d;
  assign #1 r = ~d;
  SRMS mysrms(s,r,ck,q);
endmodule
```

Esercizio

D-FF, Flip Flop principe

Reg in verilog, un D-FF

- **Descrivere un D-FF in Verilog**

Modi migliori

```
module DFF(clock,D,Q,Qbar);
   input clock,D;
   output Q,Qbar;
   reg STAR;
   assign Q=STAR, Qbar=~STAR; //Qbar è l'opposto di Q
   always @(posedge clock) //assegna sul fronte in salita del clock
   STAR<=D;
endmodule
```

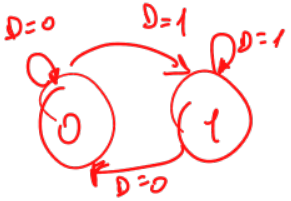
ho definito un D-FlipFlop

STATUS REGISTER

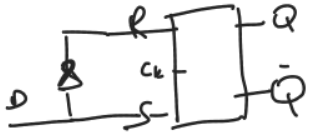
Reg = D FLIP FLOP

stato caldo solo con fronte positivo clock

negedge (fronte in discesa)



D	ck	Q	$\bar{Q}$
X	0	Q	$\bar{Q}$
1	1	1	0
0	1	0	1



D Latch
. inp. out.
reg STAR

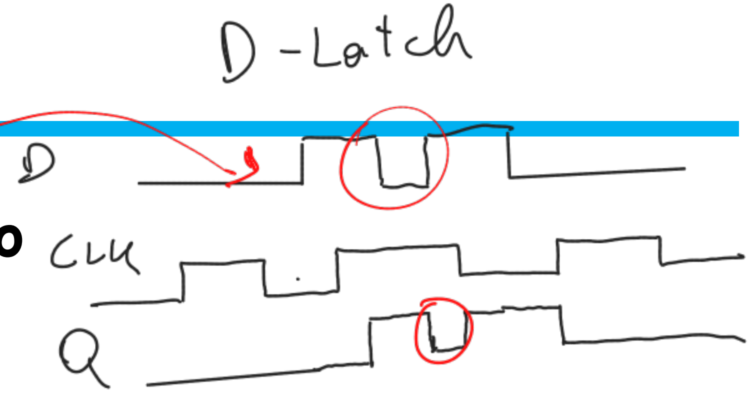
D-Latch

always @(clock==1) STAR <= D

D-FF

always @(posedge clock) STAR <= D

Confronto Latch vs. Flip-flops



- **Latch:**
  l'uscita cambia non appena cambia l'ingresso
  E
  il clock ha valore alto

- **Flip-flop:**
  l'uscita cambia solo su un fronte del clock
  (metodologia edge-triggered)

USCITA pari al valore memorizzato

- **Elementi in comune:** USCITA sempre DISPONIBILE da leggere
  - l'uscita è pari al valore memorizzato nell'elemento
    (non è necessario fare azioni particolari per leggere il valore)
  - il cambiamento dell'uscita avviene rispetto a un segnale di clock

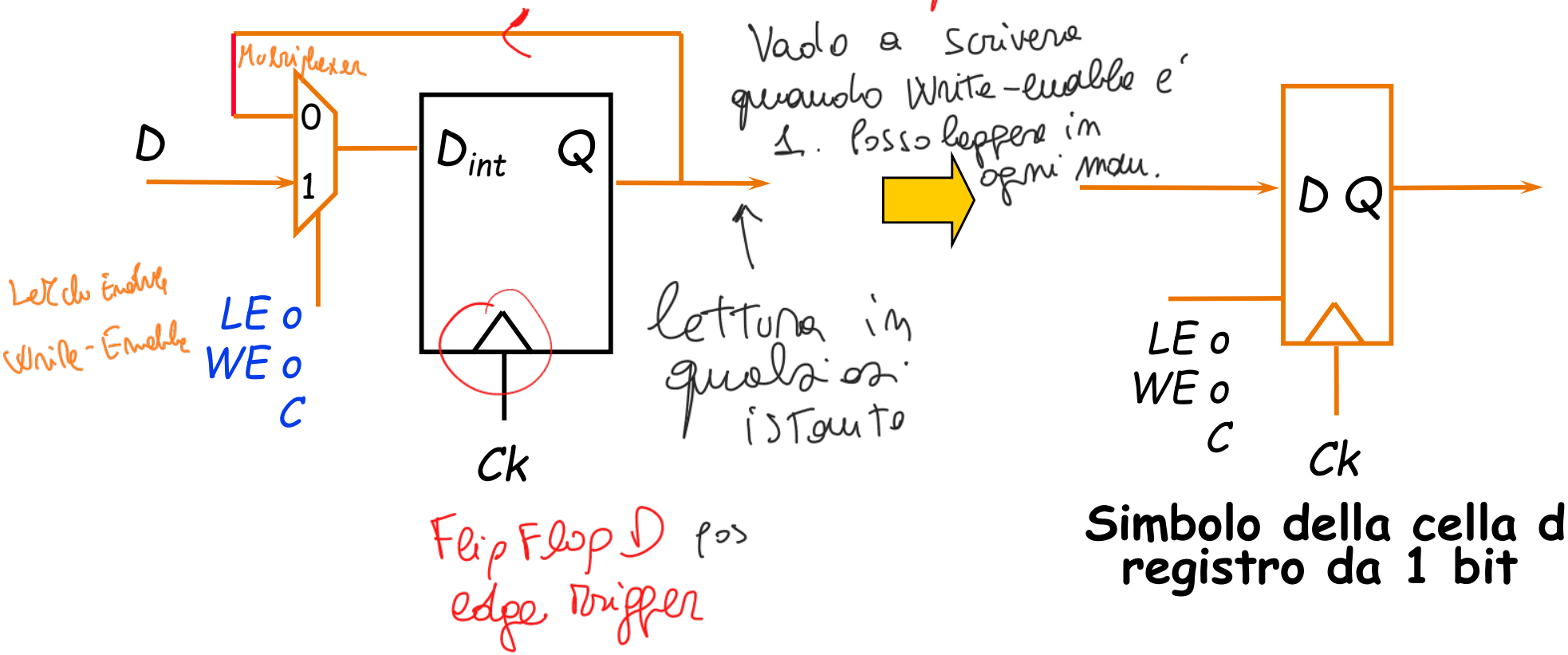
- **Nota:**
  nella logica sincrona si definisce l'istante in cui
  un valore è pronto per essere letto sugli ingressi del flip-flop
  - non si vuole leggere un dato D
    mentre la rete che genera D sta ancora "calcolandone" il valore

Nelle Reti, uso clock per congelare uscita e evitare glitch

Congelo risultato in flip-flop per poterlo leggere, lo vedo immune ai glitch

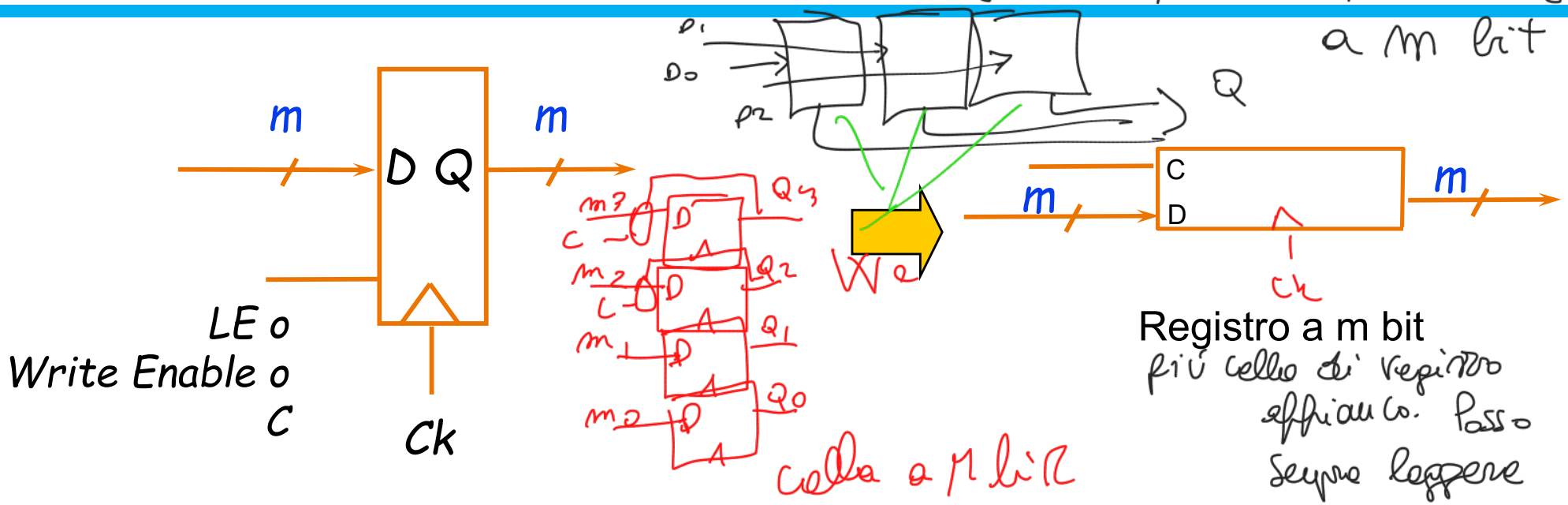
Cella di registro

- **Aggiungo un multiplexer il cui controllo si chiama**
  - **LE = Latch Enable o anche**
    WE = Write Enable o anche
    C = Control



Simbolo della cella di registro da 1 bit

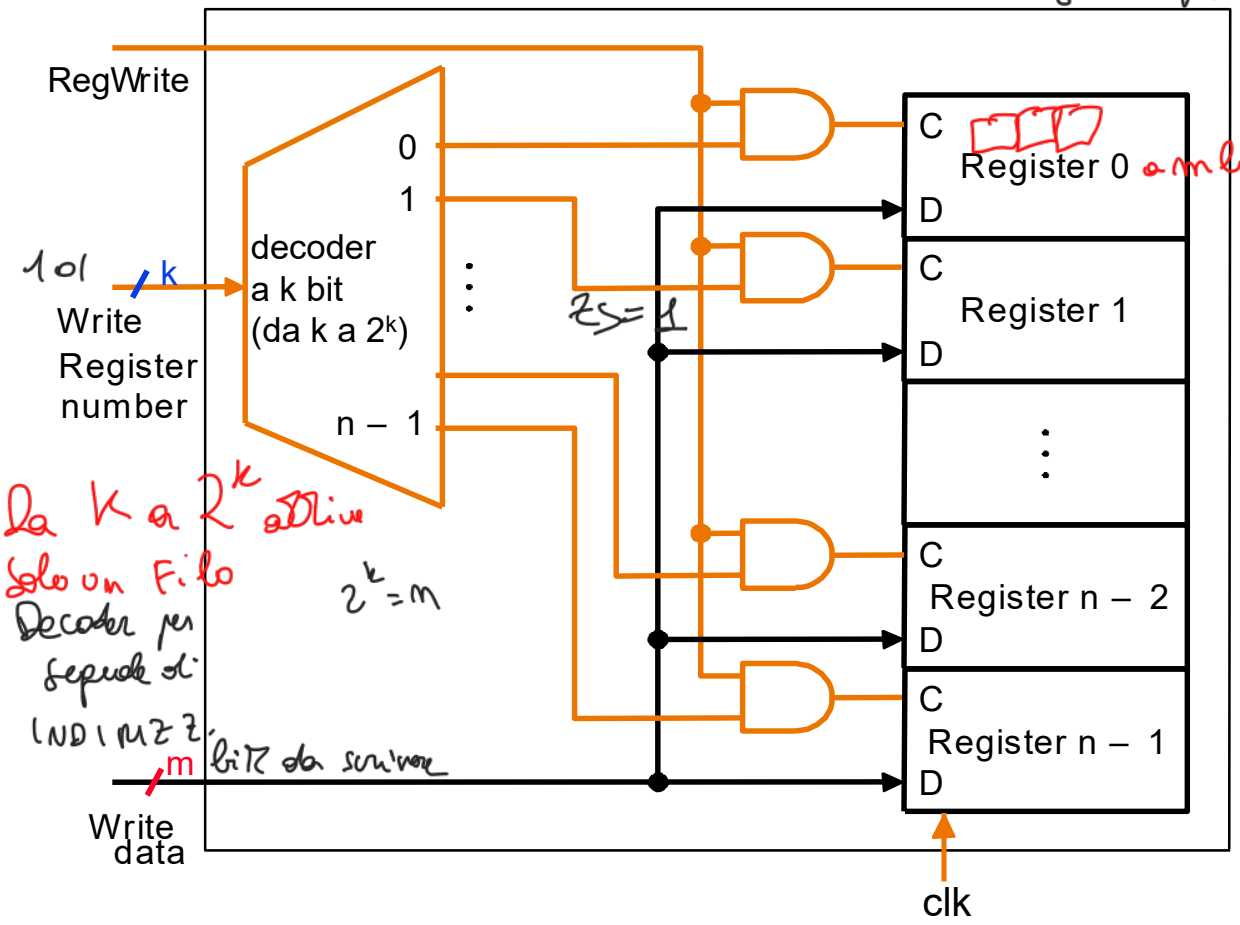
Registro a m bit

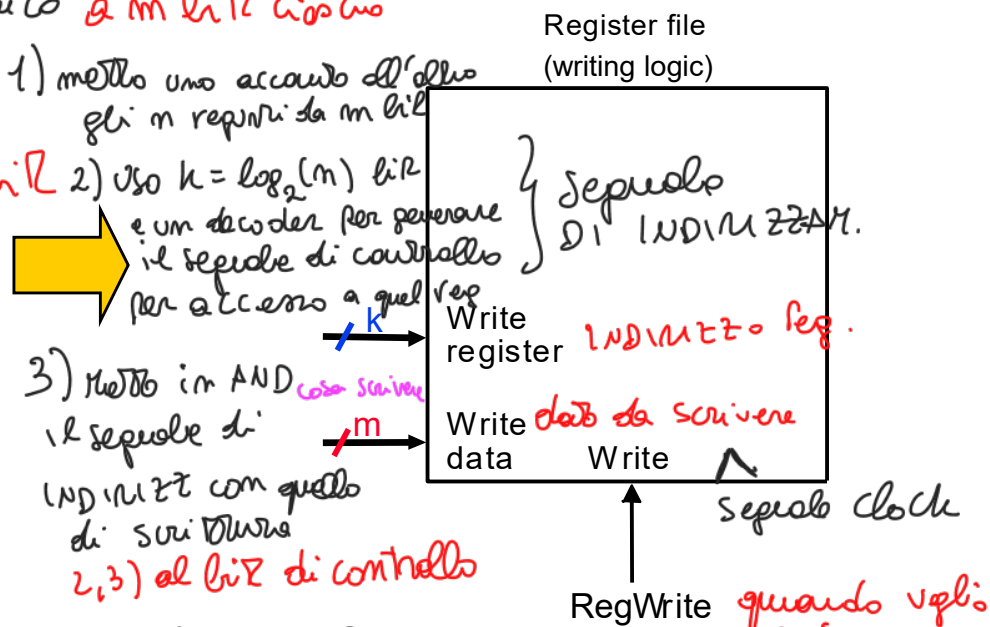


- **Nell'ultima rappresentazione si intende implicitamente che sia presente un segnale Ck per il clock**

- **Register file:**
  - **Composto da n registri di questo tipo**
  - **Occorre della logica di scrittura**
  - **Occorre della logica di lettura**

Register File: logica di scrittura (richiamo)





n = numero di registri disponibili (e.g. 32)

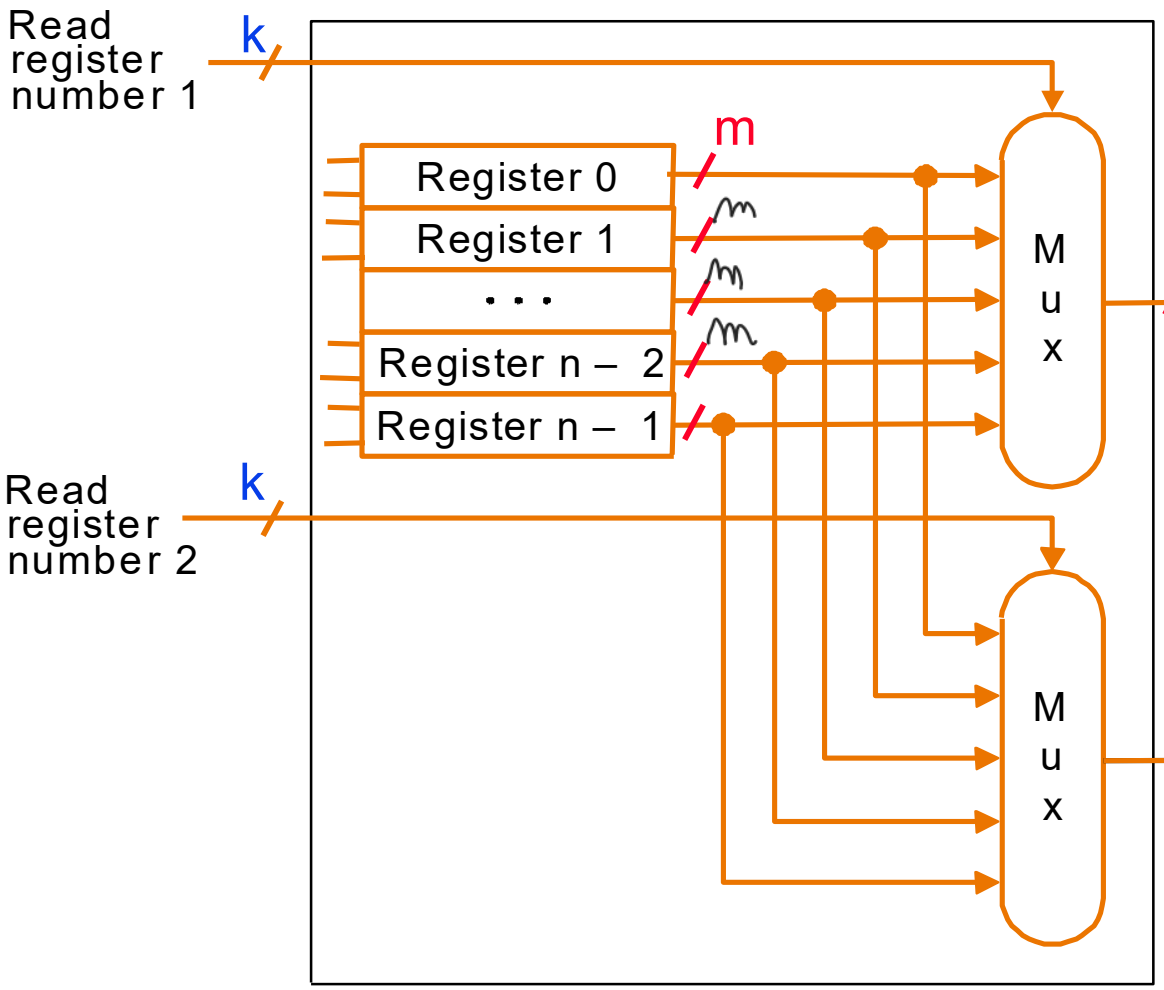
k = $\log_2(n)$ (e.g. 5)

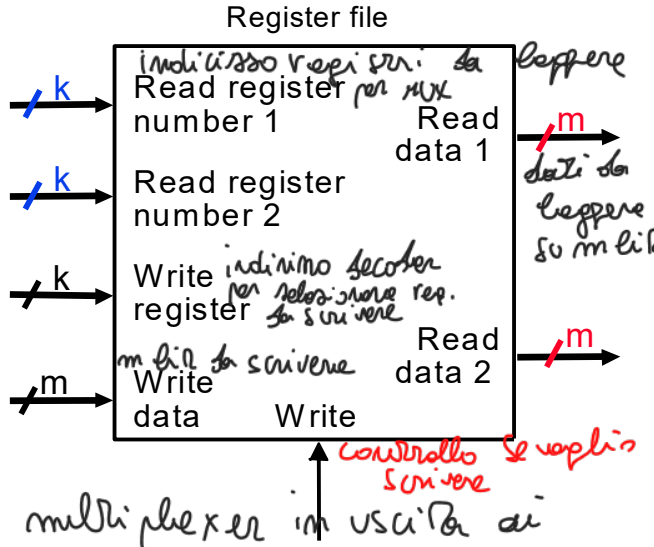
m = larghezza dei registri (e.g. 64)

Register File: logica di lettura (richiamo)

Voglio leggere 2 registri, es. somma fra 2 operandi

- Si usano i flip-flop di tipo D





Per leggere registri

1) Post-pongo multiplexer in uscita ai registri (n ingressi da m bit)

2) ho k bit ($k = \log_2(n)$) che mi pilotano il multiplexer

ho 2^k registri, in uscita ho read data 1, 2

$k = \log_2 n$, regole di controllo che mi dice quanto scegliere