Circuiti sequenziali

Logica sequenziale

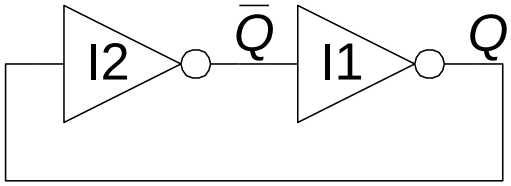
- Nei sistemi sequenziali l'output dipende sia dal valore corrente sia da valori precedenti dell'input. In tal senso si dice che il sistema ha *memoria*
- *Stato interno:* tutta l'informazione di un circuito che mantiene la *storia* di quel circuito ed è necessaria per prevedere il suo comportamento futuro
- Come vedremo lo stato di un sistema sarà memorizzato in componenti come i latches e flip-flop
- Un circuito sequenziale (*sincrono*) avrà una topologia ben precisa:
  - logica combinatoria: definisce l'evoluzione del sistema
  - Banchi di flip-flop: servono a memorizzare gli stati del sistema
- Un aspetto peculiare dei sistemi sequenziali è quello della retroazione (*feedback*), ovvero il segnali di output vengono riportati in input

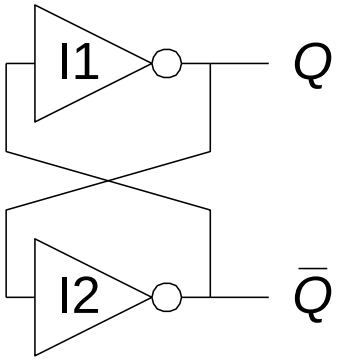
State elements

- Lo stato di un circuito influenzerà l'evoluzione del sistema
- Gli *state elements* sono tutte quelle componenti circuitali che vengono adoperate per memorizzare lo stato di un circuito
  - Circuiti bistabili
  - SR Latch
  - D Latch
  - D Flip-flop

Circuito bistabile

- *building block* per altri state elements
- Two outputs: Q, $\overline{Q}$
- No inputs



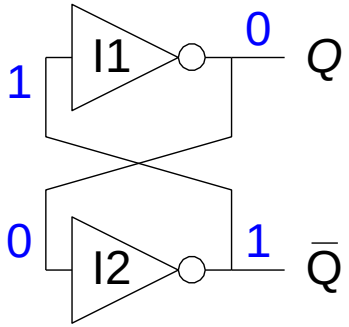


Analisi del circuito bistabile

- Considera i due possibili casi:

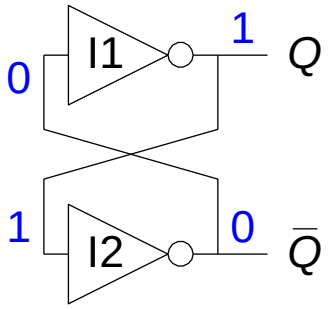
  Q = 0:

  allora $Q = 0$, $\bar{Q} = 1$ (consistente)



  Q = 1:

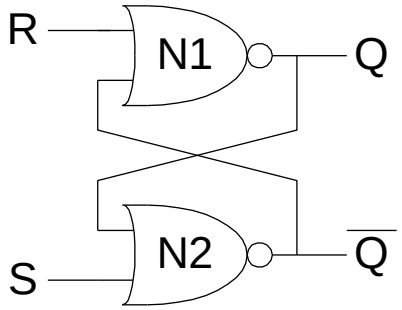
  allora $Q = 1$, $\bar{Q} = 0$ (consistente)



- Memorizza 1 bit nella variabile di stato Q (or $\bar{Q}$)
- Ma non ci sono input per controllare questo stato!

SR (Set/Reset) Latch

- SR Latch



- Consideriamo i 4 possibili stati:

  $S = 1, R = 0$

  $S = 0, R = 1$

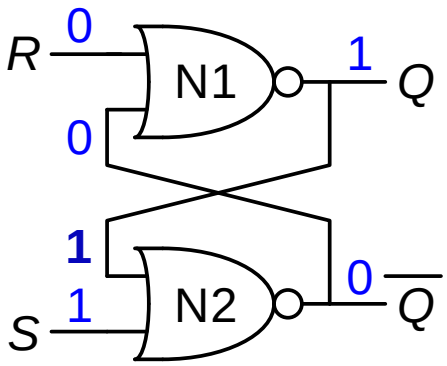
  $S = 0, R = 0$

  $S = 1, R = 1$

Analisi di un SR Latch

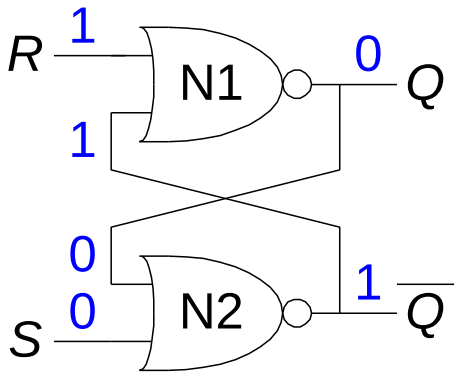
***S* = 1, *R* = 0:**

allora $Q = 1$ e $\overline{Q} = 0$



***S* = 0, *R* = 1:**

allora $Q = 0$ e $\overline{Q} = 1$

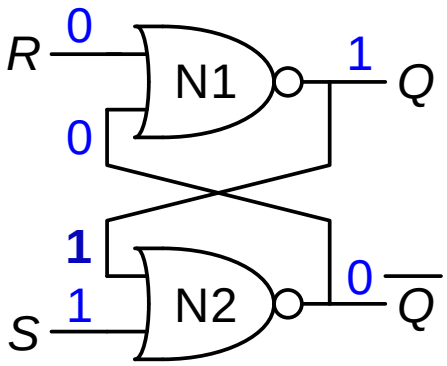


Analisi di un SR Latch

***S* = 1, *R* = 0:**

allora $Q = 1$ e $\overline{Q} = 0$

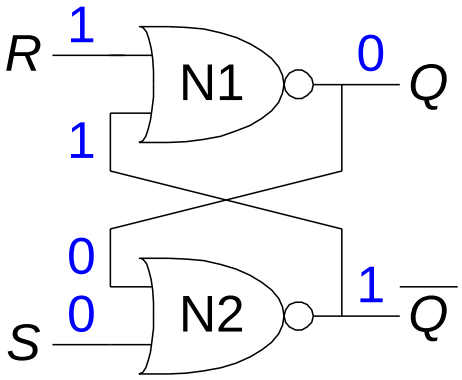
operazione Set



***S* = 0, *R* = 1:**

allora $Q = 0$ e $\overline{Q} = 1$

operazione Reset

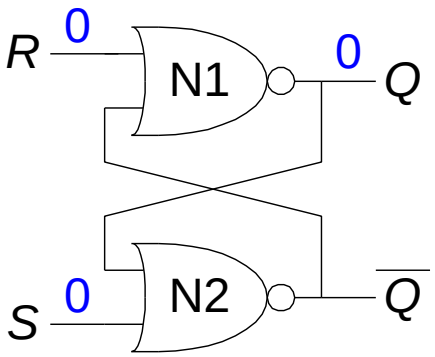


Analisi di un SR Latch

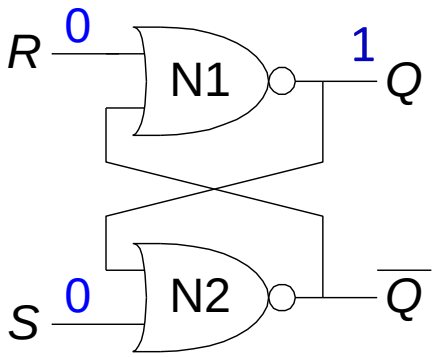
$S = 0, R = 0$:

allora $Q = Q_{prev}$

$Q_{prev} = 0$

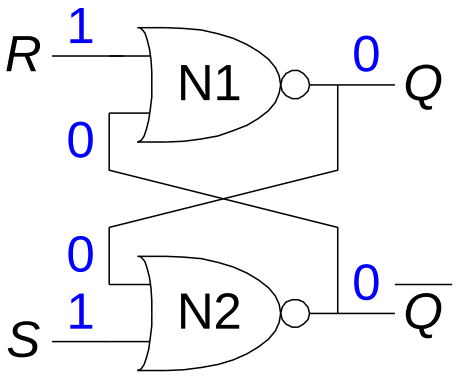


$Q_{prev} = 1$



$S = 1, R = 1$:

allora $Q = 0, \bar{Q} = 0$



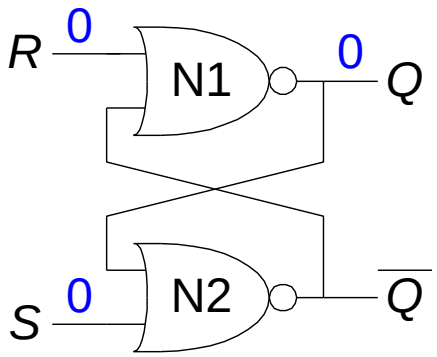
Analisi di un SR Latch

$S = 0, R = 0$:

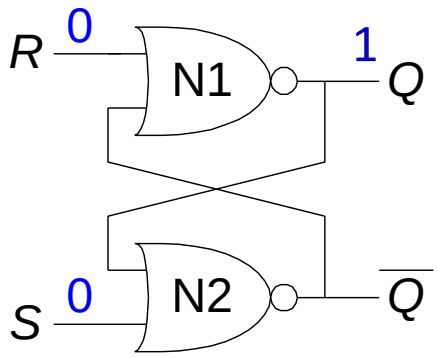
allora $Q = Q_{prev}$

Memoria

$Q_{prev} = 0$



$Q_{prev} = 1$

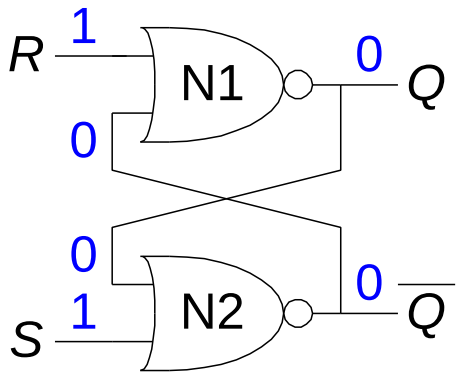


$S = 1, R = 1$:

allora $Q = 0, \bar{Q} = 0$

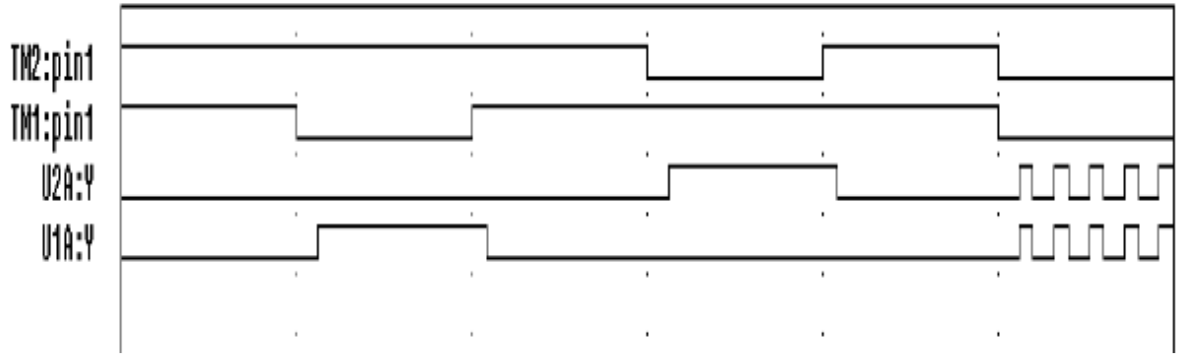
Stato non valido

$Q \neq$ NOT $\bar{Q}$



Analisi di un SR Latch

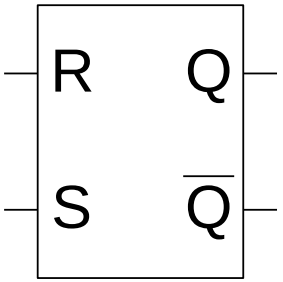
- Se dalla condizione S=R=1 si passa alla condizione S=R=0 allora
  - Se i tempi di propagazione sono uguali allora il circuito va in oscillazione
  - Nell'ipotesi, più realistica che le porte abbiano ritardi anche lievemente differenti, il circuito si mette in uno dei due stati possibili. Anche in questo caso, però, lo stato finale non è predicibile



Simbolo per un SR Latch

- SR sta per Set/Reset
  - Memorizza un bit (*Q*)
- **Set:** Pone l'output a 1
  (*S* = 1, *R* = 0, *Q* = **1**)
- **Reset:** Pone l'output a 0
  (*S* = 0, *R* = 1, *Q* = **0**)

SR Latch Symbol

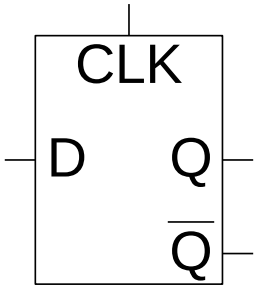


Occorre evitare lo stato non valido S = R = 1

D Latch

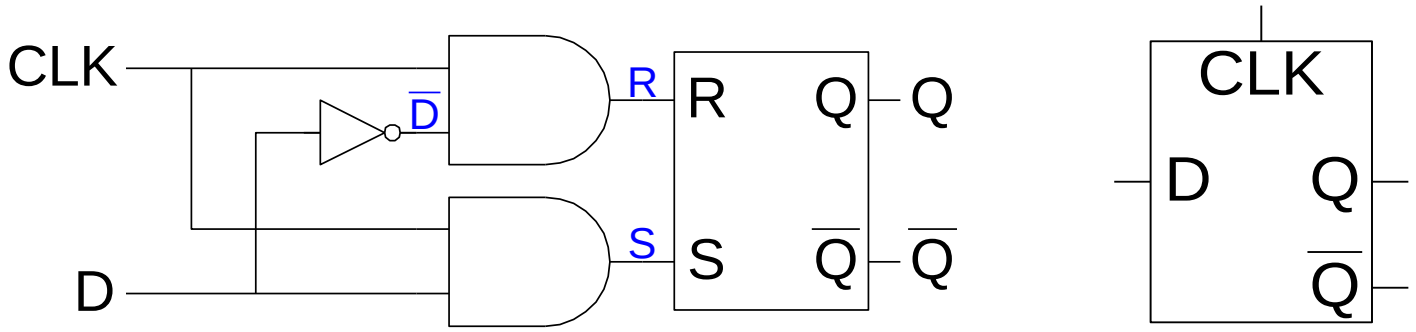
- 2 input: *CLK*, *D*
- ***CLK*:** controlla *quando* l'output cambia
- ***D*** (data input): controlla *in che cosa* l'output cambia
- Se ***CLK* = 1**,
  D passa fino a Q (*transparente*)
- Se ***CLK* = 0**,
  Q mantiene il suo valore precedente (*opaco*)

D Latch Symbol



Evita lo stato non valido in cui $Q \neq \text{NOT } \overline{Q}$

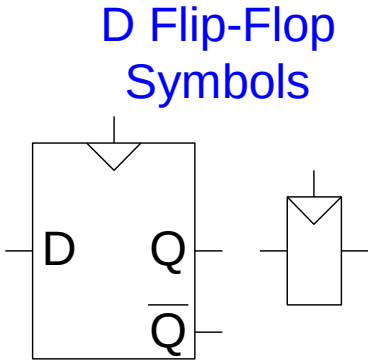
D Latch



CLK	*D*	$\overline{D}$	*S*	*R*	*Q*	$\overline{Q}$
0	X	$\overline{X}$	0	0	Q_{prev}	$\overline{Q}_{prev}$
1	0	1	0	1	0	1
1	1	0	1	0	1	0

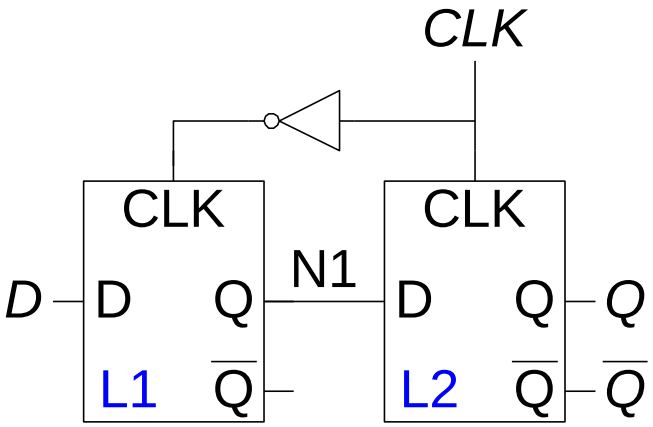
D Flip-Flop

- Inputs: *CLK*, *D*
- Funzione:
  - Quando *CLK* passa da 0 a 1, *D* passa fino a *Q*
  - Altrimenti, *Q* mantiene il suo valore precedente
- *Q* cambia solo durante la transizione di *CLK* da 0 a 1
- Queste tipologie di componenti sono dette *edge-triggered* perché non sono pilotate da un valore ma da una transizione



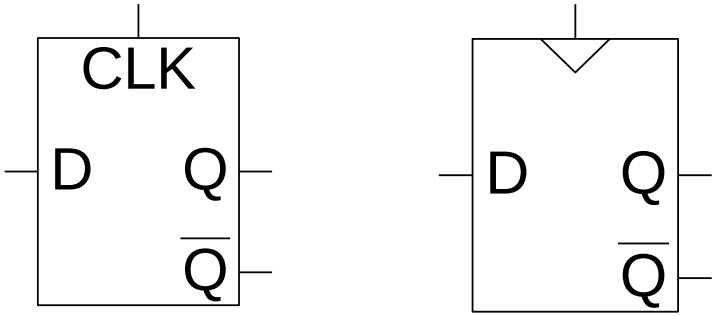
D Flip-Flop

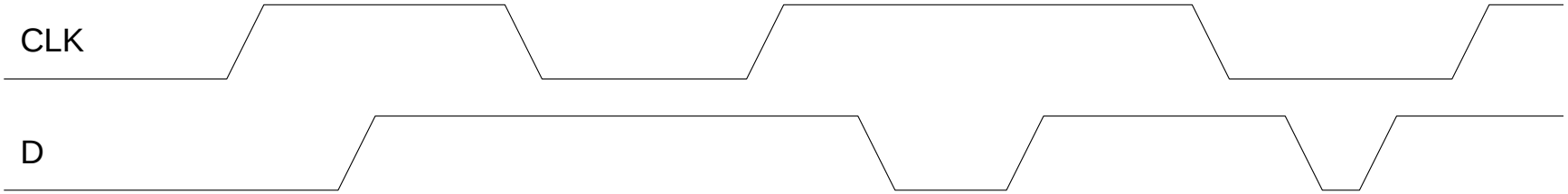
- 2 D latch (L1 e L2) controllati da clock complementari
- Quando ***CLK* = 0**
  - L1 è trasparente
  - L2 è opaco
  - *D* passa fino a N1
- Quando ***CLK* = 1**
  - L2 è trasparente
  - L1 è opaco
  - N1 passa fino a *Q*



- Quindi, D passa fino a Q *sulla transizione di CLK da 0 a 1*
- Ulteriori variazioni di D quando CLK=1 non passano a Q perché L1 è opaco

D Latch vs. D Flip-Flop

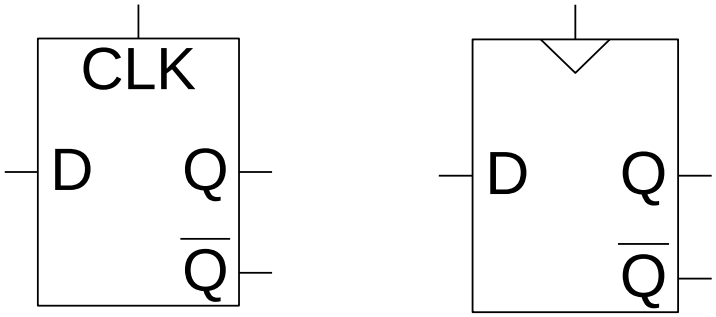


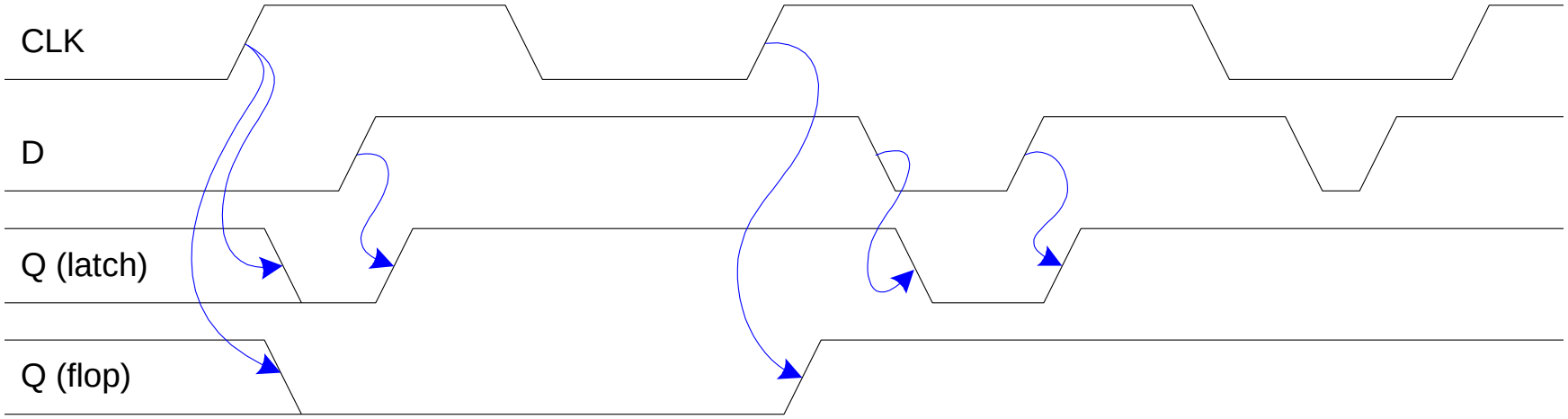


Q (latch)

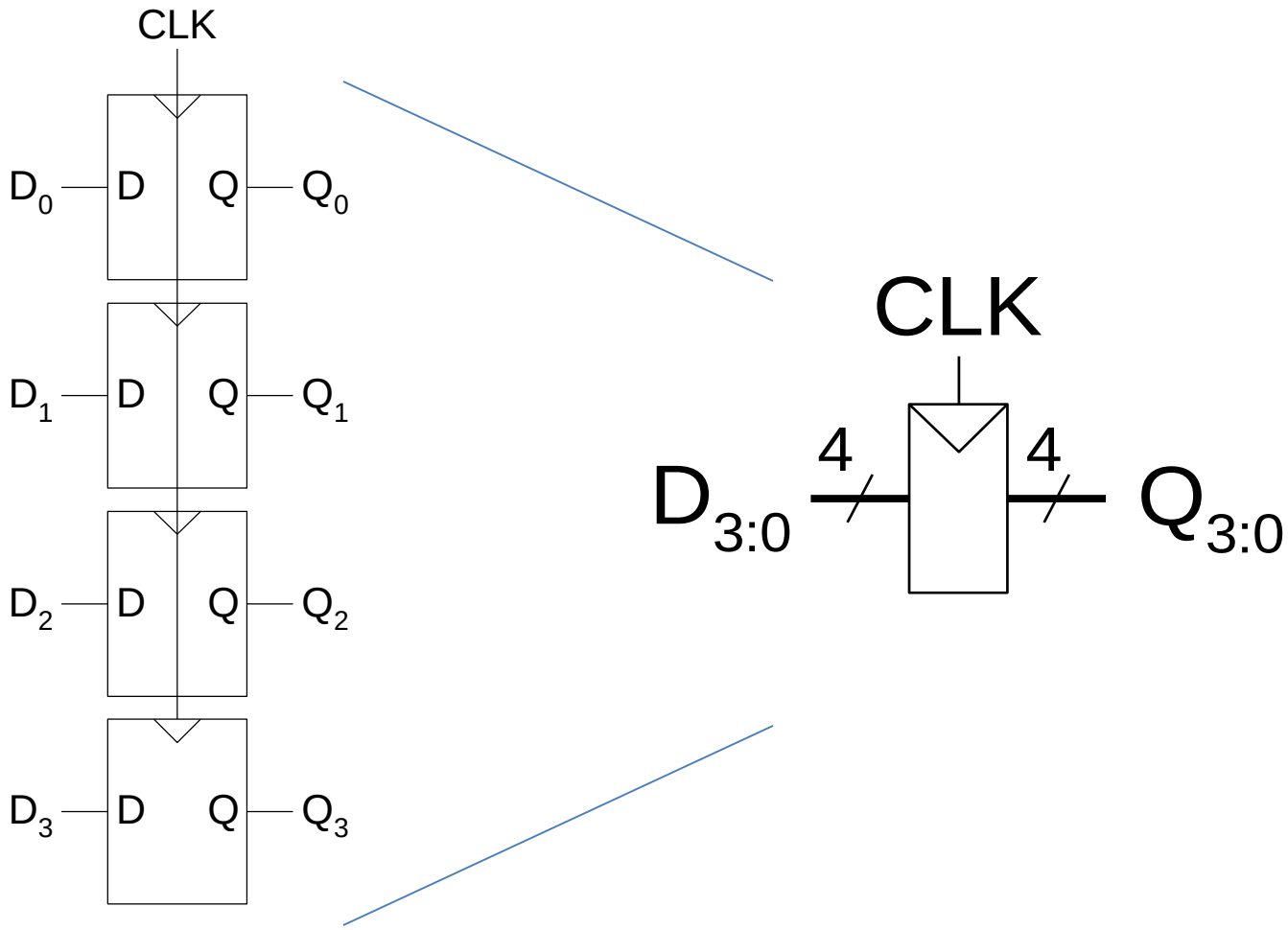
Q (flop)

D Latch vs. D Flip-Flop



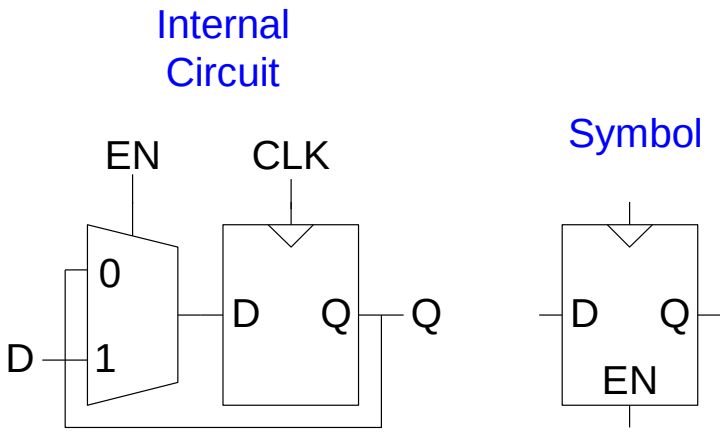


Registri: Multi-bit Flip-Flop



Flip-Flops “enabled”

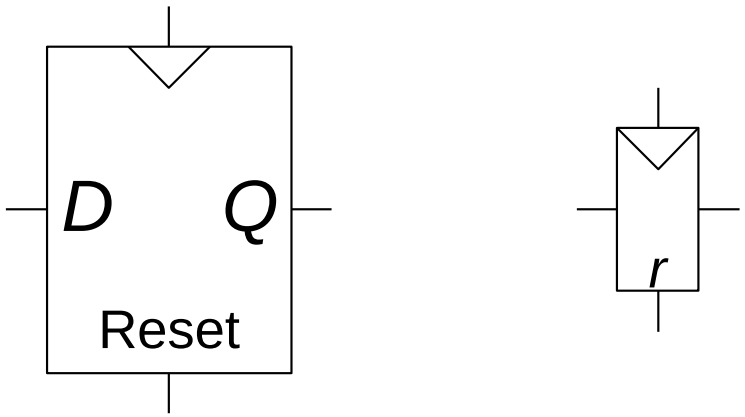
- *Inputs: CLK, D, EN*
- L’input enable (*EN*) stabilisce quando un nuovo valore di *D* è memorizzato
- ***EN* = 1:** *D* passa fino a *Q* (clock: 0→1)
- ***EN* = 0:** il flip-flop mantiene il suo stato precedente



Flip-Flops “resettabili”

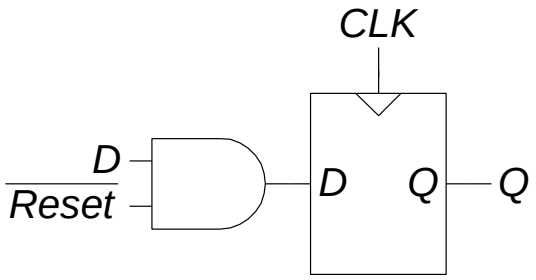
- *Inputs: CLK, D, Reset*
- ***Reset* = 1:** *Q* = 0
- ***Reset* = 0:** il flip-flop si comporta “normalmente” come un D flip-flop

Symbols



Flip-Flops “resettabili”

- Vi sono due tipi di flip-flop resettabili:
  - **Sincroni:** il reset è pilotato dal clock
  - **Asincroni:** il reset avviene non appena *Reset* = 1
- Flip-flop sincroni:

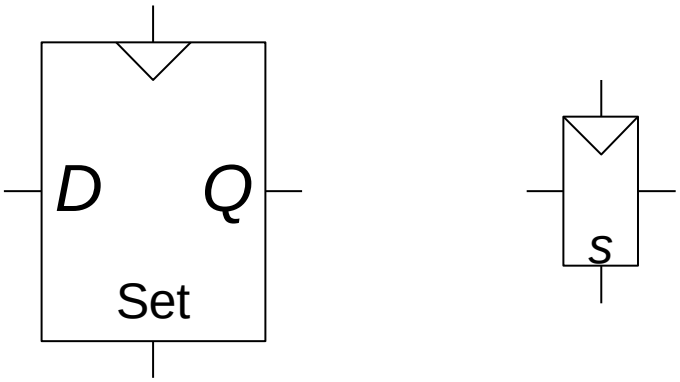


- Per i flip-flop asincroni occorre modificare il circuito interno del flip-flop

Flip-Flops “settabili”

- *Inputs: CLK*, *D*, *Set*
- ***Set* = 1:** *Q*=1
- ***Set* = 0:** il flip-flop si comporta “normalmente” come un D flip-flop

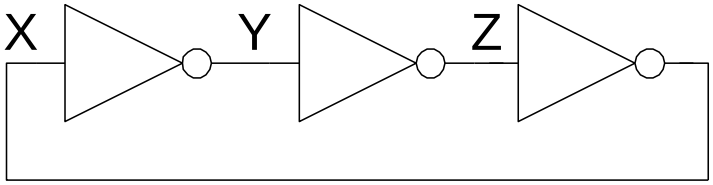
Symbols

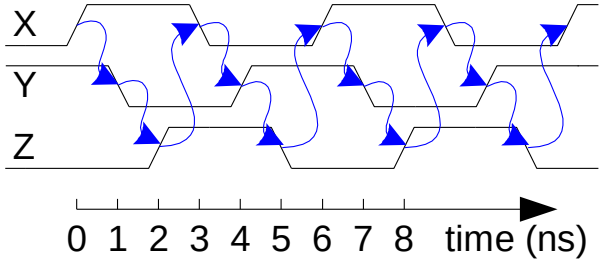


Esercizi

- Esercizi 3.1, 3.3, 3.5, 3.7, 3.8, 3.13, 3.15

Criticità nella logica sequenziale

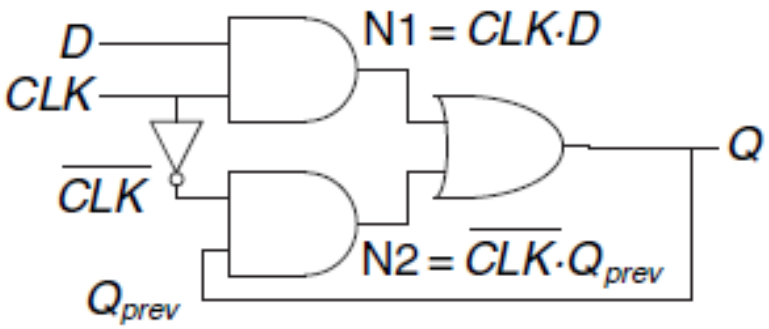




- Questi circuiti vengono detti “astabili” poiché hanno un comportamento oscillante
- Il periodo di oscillazione dipende dai ritardi degli inverter
- Idealmente è di 6 ns tuttavia può variare a causa di diversi fattori
  - differenze nella manifattura
  - temperatura
- Circuito *asincrono*: l’output è retroazionato in maniera diretta

Criticità nella logica sequenziale

- In casi più complessi che comprendono l'uso di più porte AND, NOT, OR il comportamento di una rete asincrona può dipendere fortemente dai ritardi accumulati sui singoli cammini

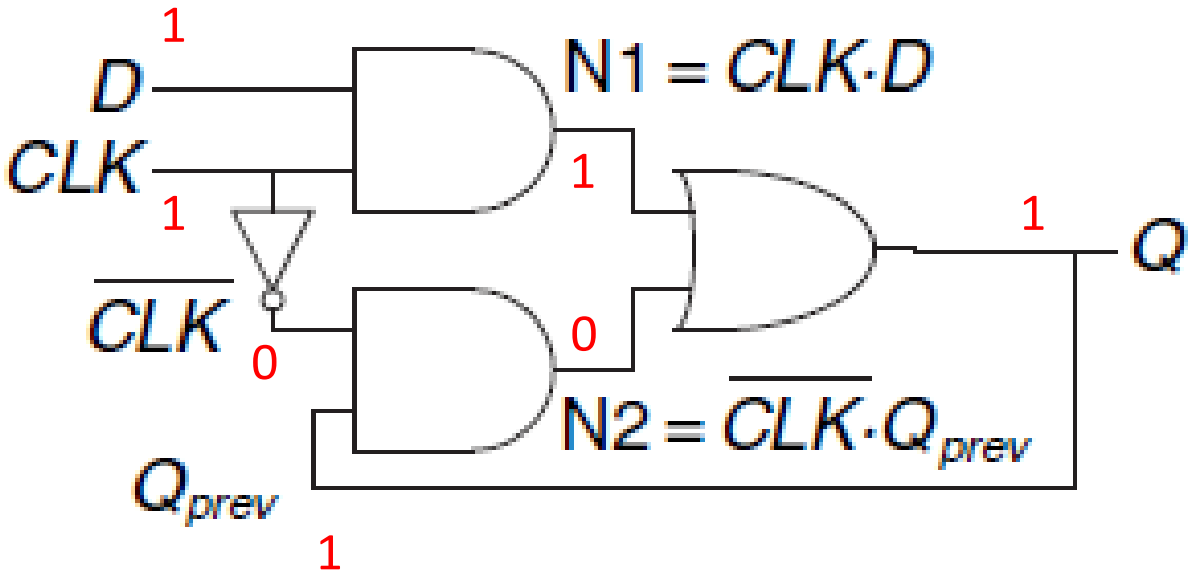


$$Q = CLK \cdot D + \overline{CLK} \cdot Q_{prev}$$

CLK	D	Q_{prev}	Q
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

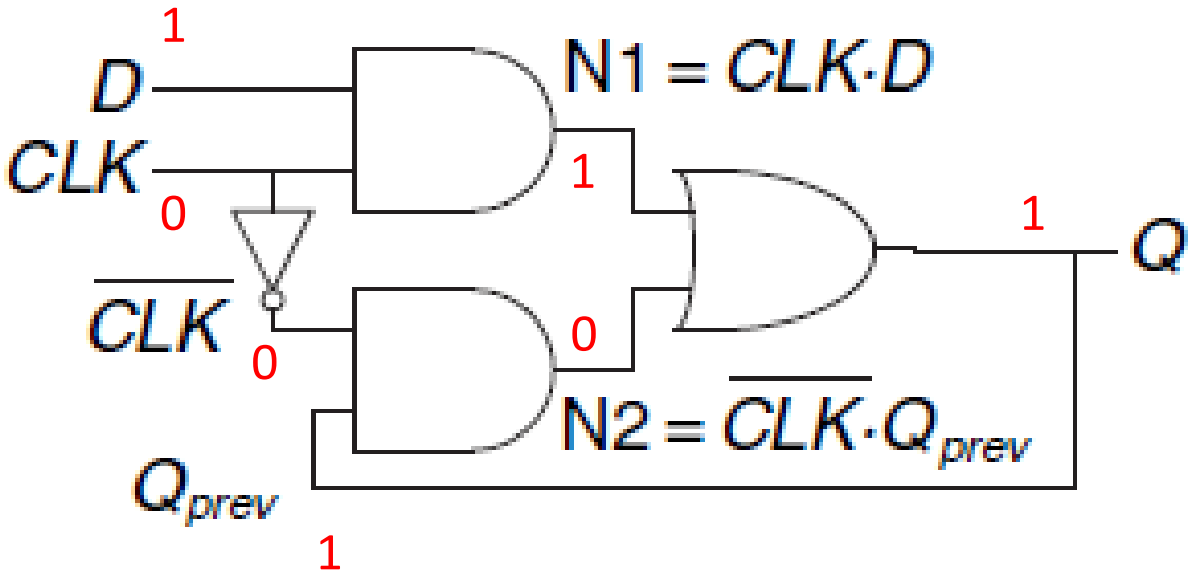
Criticità nella logica sequenziale

- D=1, CLK=1 ➡ Q=1



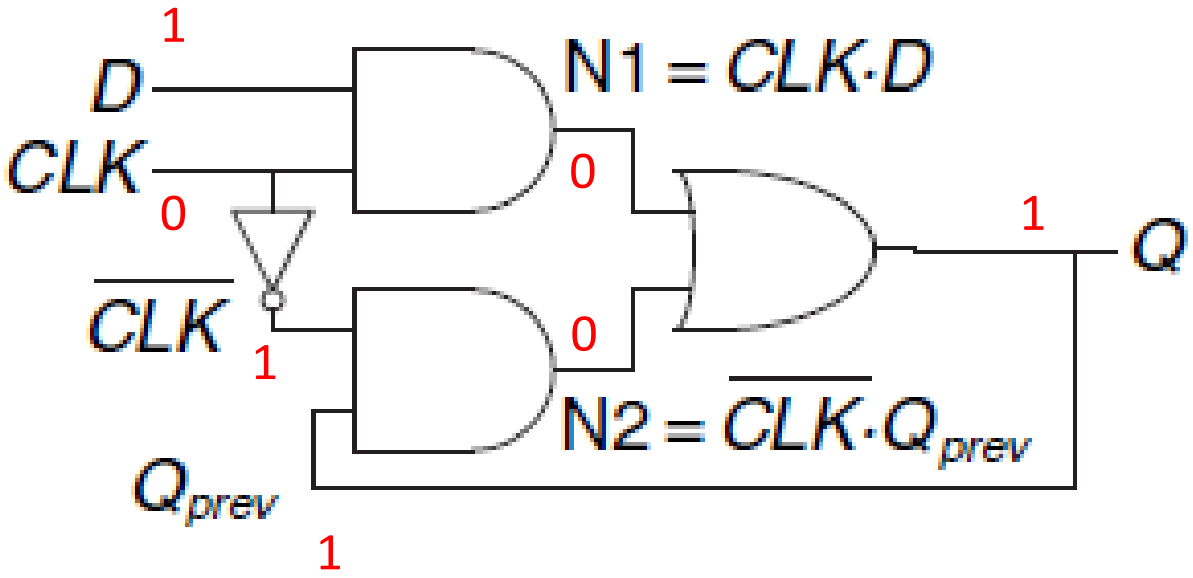
Criticità nella logica sequenziale

t_0 CLK 1→0



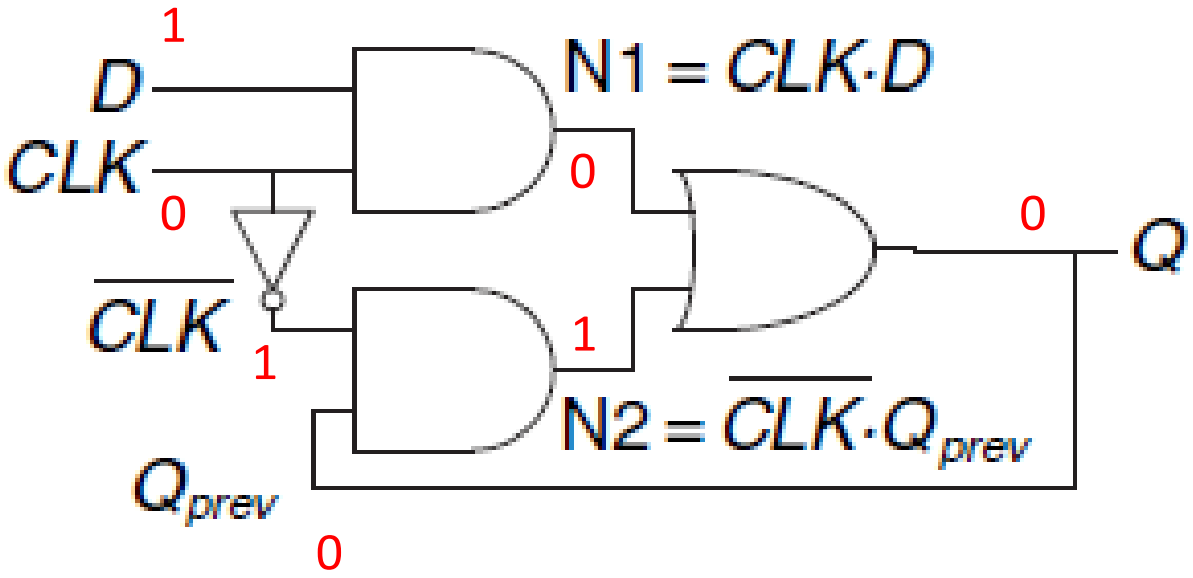
Criticità nella logica sequenziale

t_1



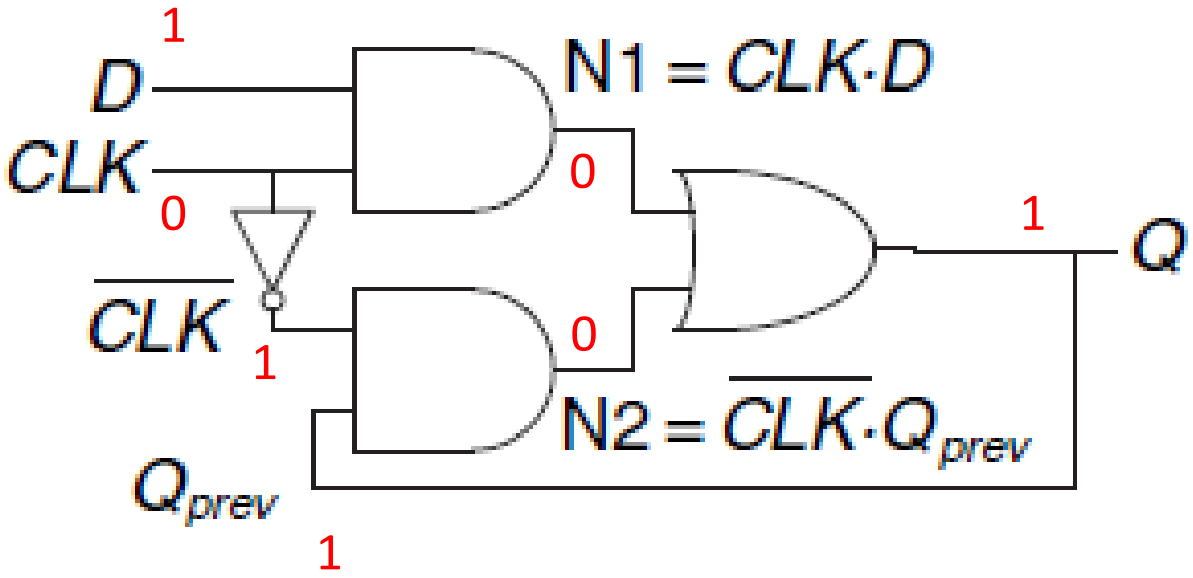
Criticità nella logica sequenziale

t_2



Criticità nella logica sequenziale

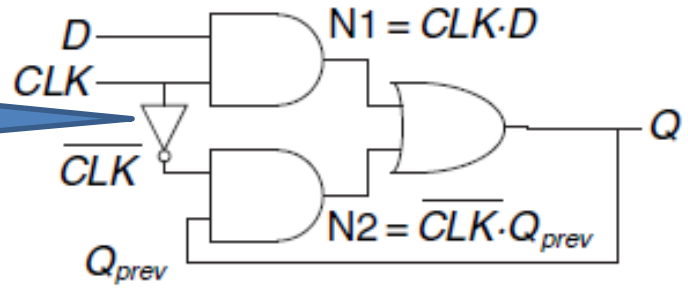
t_3 (= t_1)



Criticità nella logica sequenziale

- In casi più complessi che comprendono l'uso di più porte AND, NOT, OR il comportamento di una rete asincrona può dipendere fortemente dai ritardi accumulati sui singoli cammini

Ritardo dell'inverter > ritardi delle altre porte



- D=1, CLK=1 ➡ Q=1
- CLK=0 ➡ Q oscilla ($Q=Q_{prev}=1$)

Logiche sequenziali sincrone

- I circuiti asincroni presentano delle criticità a volte difficilmente analizzabili
  - Dipendono dalla struttura fisica dei componenti
- Per questo si cerca di evitare di retroazionare l'output in maniera diretta e si interpone un registro nel ciclo di retroazione
- *Nell'ipotesi che il clock sia più lento del ritardo accumulato sul cammino,* il registro consente al sistema di essere sincronizzato col clock: circuito *sincrono*

Logiche sequenziali sincrone

- In generale un circuito sequenziale sincrono ha un insieme finito di stati $\{S_0,\ldots, S_{k-1}\}$
- Logica combinatoria:

  $out=f(in)$

- Logica sequenziale sincrona:

  $out=f(in,s_c)$

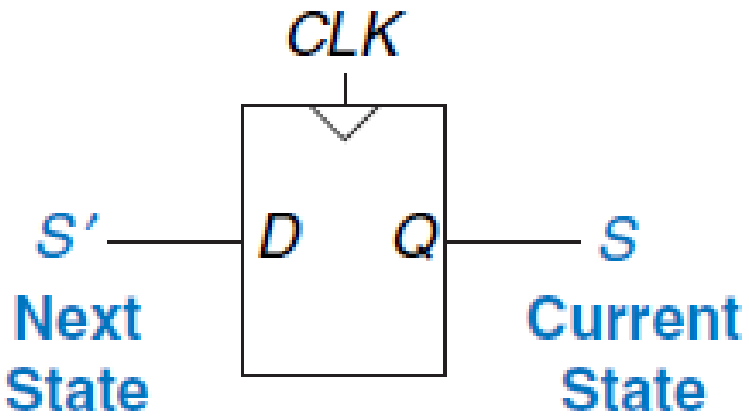
  $s_n = g(in,s_c)$

Design di logiche sequenziali sincrone

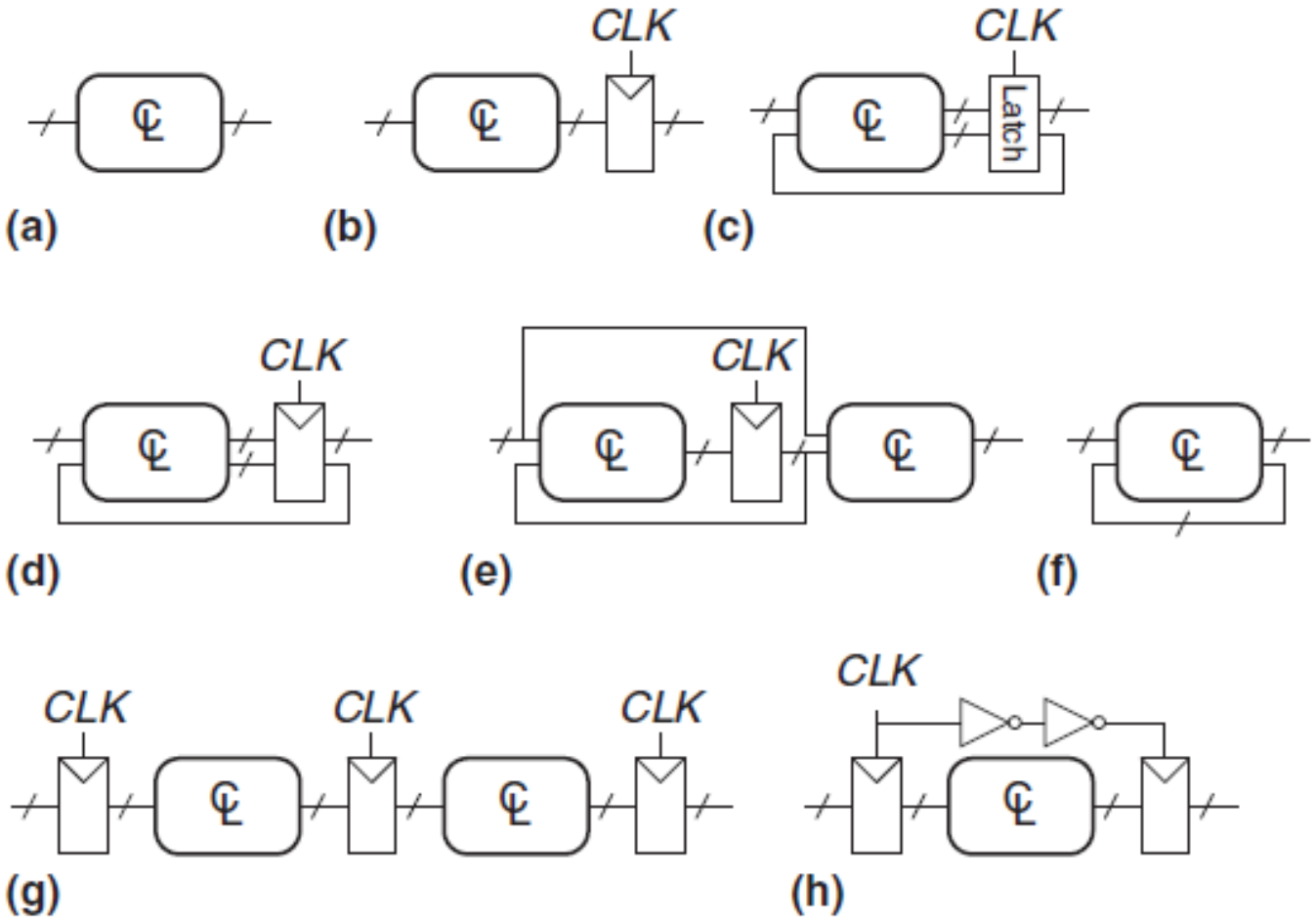
- Inserire registri nei cammini ciclici
- I registri determinano lo **stato** S_0,…, S_{k-1} del sistema
- I cambiamenti di stato sono determinati dalle transizioni del clock: il sistema è sincronizzato con il clock
- *Regole* di composizione:
  - Ogni componente è un registro o un circuito combinatorio
  - Almeno un componente è un registro
  - Tutti i registri sono sincronizzati con un unico clock
  - Ogni ciclo contiene almeno un registro
- Due tipici circuiti sequenziali sincroni
  - Finite State Machines (FSMs)
  - Pipelines

Current state /Next state

- Un flip-flop D è il più semplice circuito sequenziale sincrono
  - Q= s_c
  - D= s_n



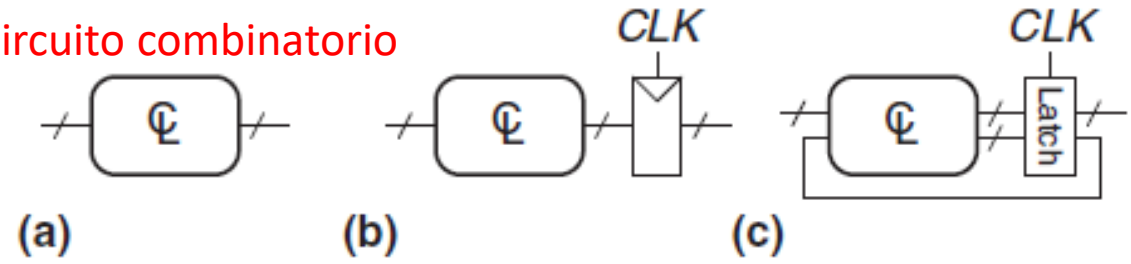
Esempi



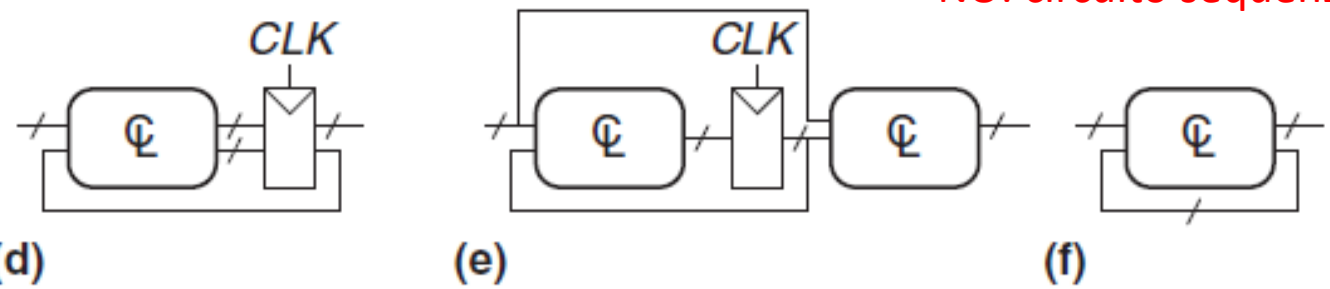
Esempi

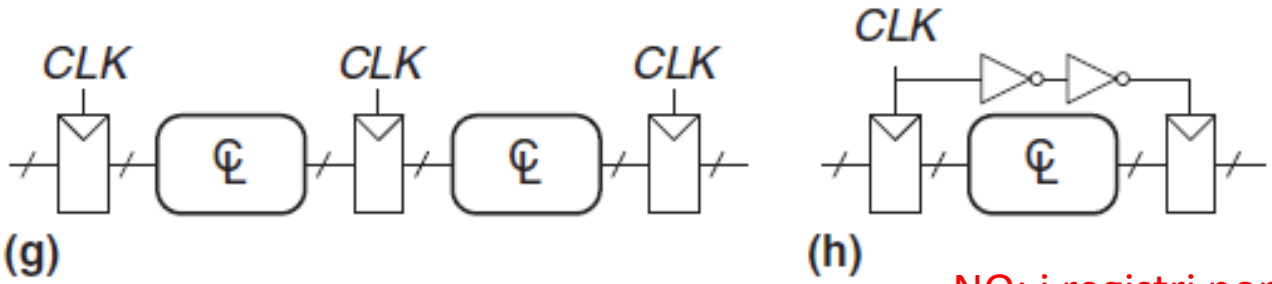
NO: circuito combinatorio

NO: latch e non flip-flop



NO: circuito sequenziale asincrono

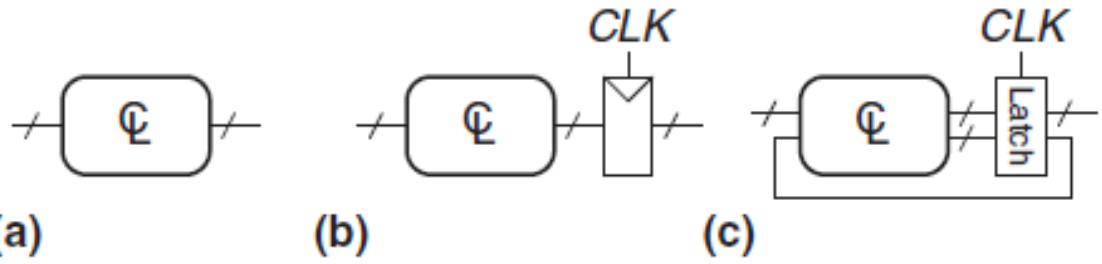




NO: i registri non hanno lo stesso clock

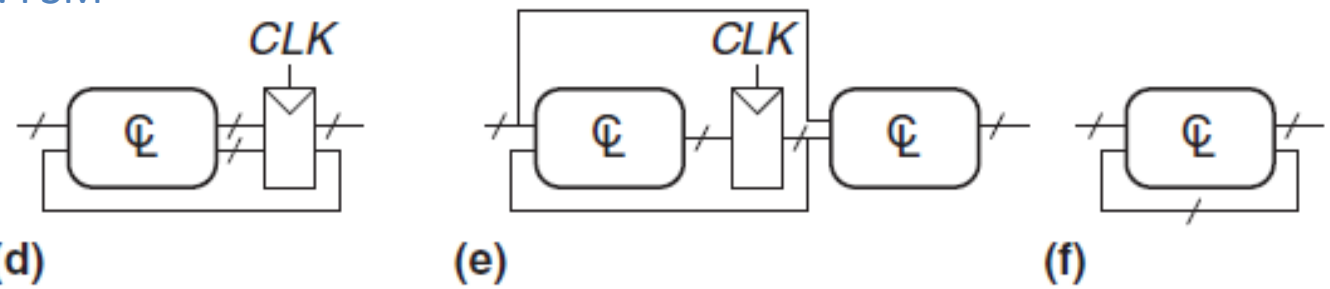
Esempi

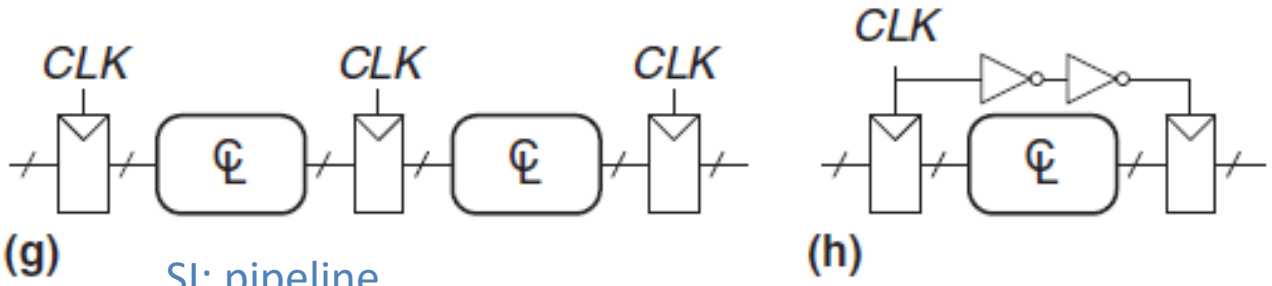
SI: ma senza feedback



SI: FSM

SI: FSM

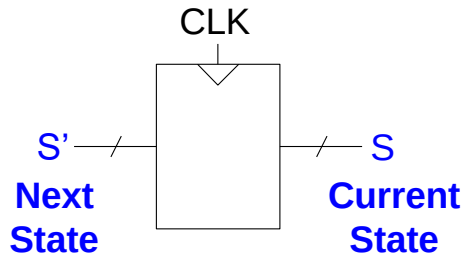


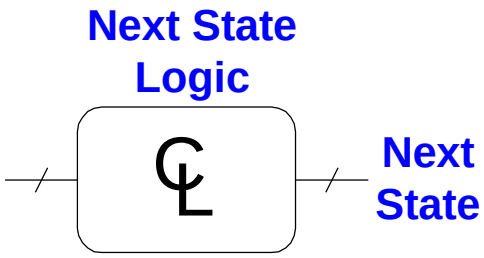


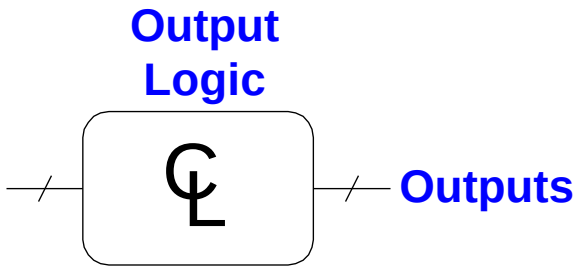
SI: pipeline

Finite State Machines

- **State register**
  - Memorizzano lo stato corrente
  - Caricano il prossimo stato (clock edge)
- **Logica combinatoria**
  - “Computa” il prossimo stato (g)
  - “Computa” gli output (f)







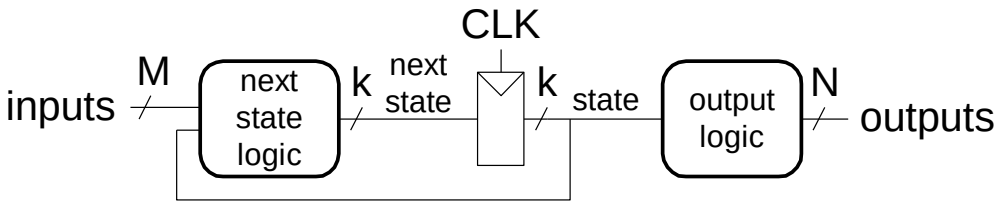
Finite State Machines

- s_n dipende sia dall'input che da s_c

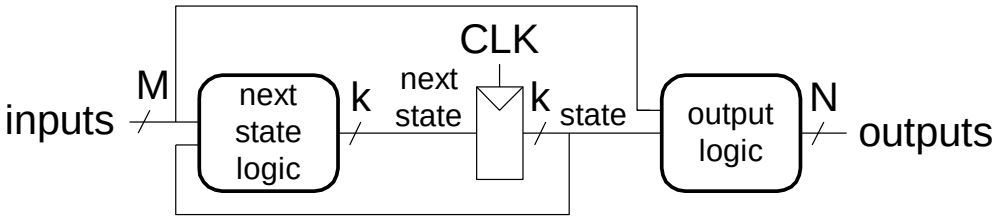
$$s_n = g(in, s_c)$$

- 2 tipi di FSM a seconda della logica di output:
  - **Moore FSM:** $out = f(s_c)$
  - **Mealy FSM:** $out = f(in, s_c)$

Moore FSM

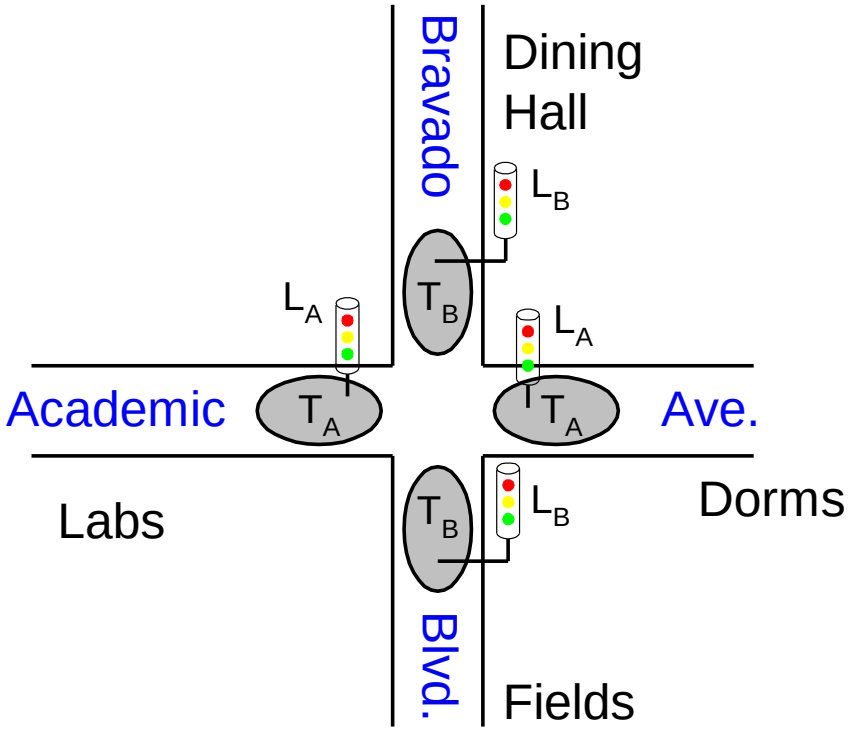


Mealy FSM



Esempio: semaforo

- Sensori: T_A, T_B (TRUE quando c'è traffico)
- Luci: L_A, L_B



Semaforo: *black box*

- Inputs: *CLK*, *Reset*, T_A, T_B
- Outputs: L_A, L_B

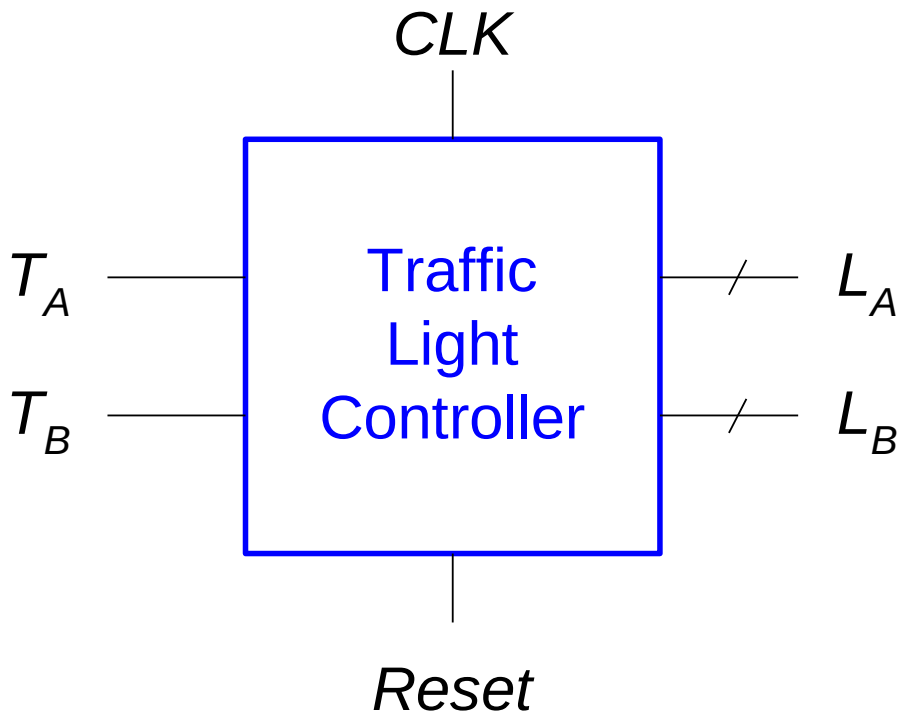
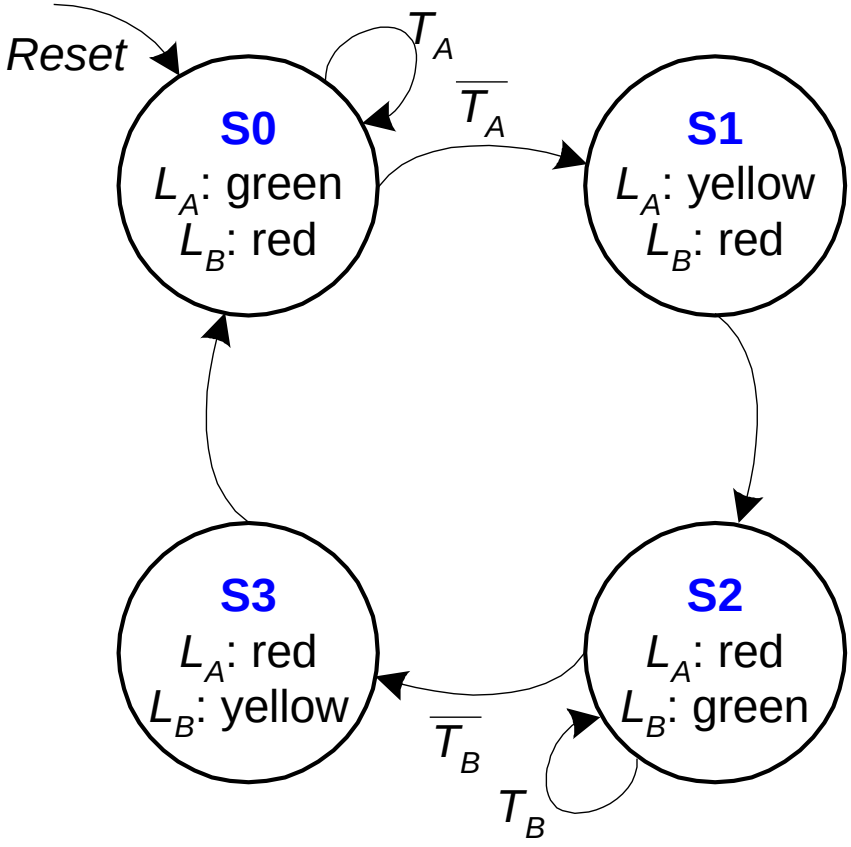


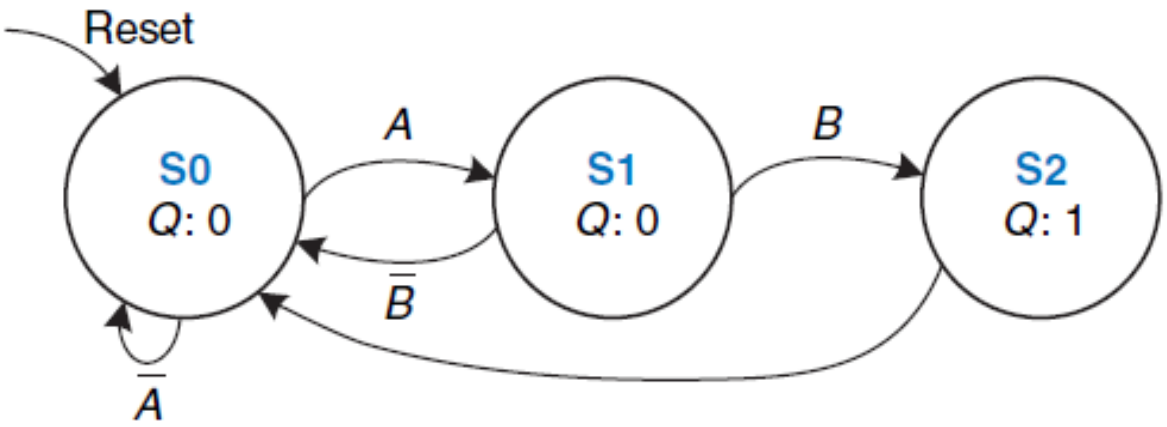
Diagramma di transizione: Moore FSM

- **Stati:** labellati con gli outputs
- **Transizioni:** labellate con gli inputs



Esempio Moore FSM

- Quale è il comportamento della FSM seguente?



Esempio Mealey FSM

- Quale è il comportamento della FSM seguente?

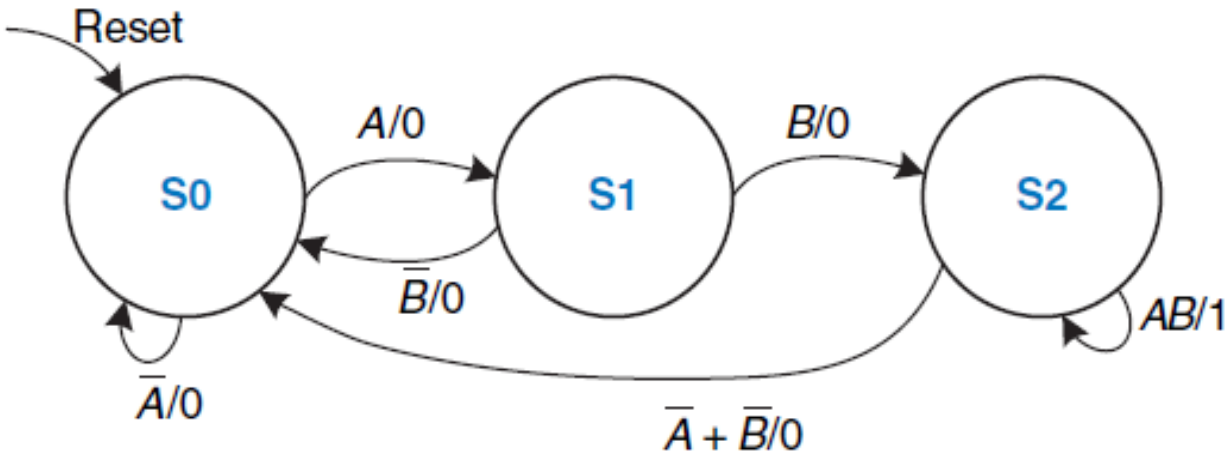


Diagramma di transizione: Moore FSM

- **Stati:** labellati con gli outputs
- **Transizioni:** labellate con gli inputs

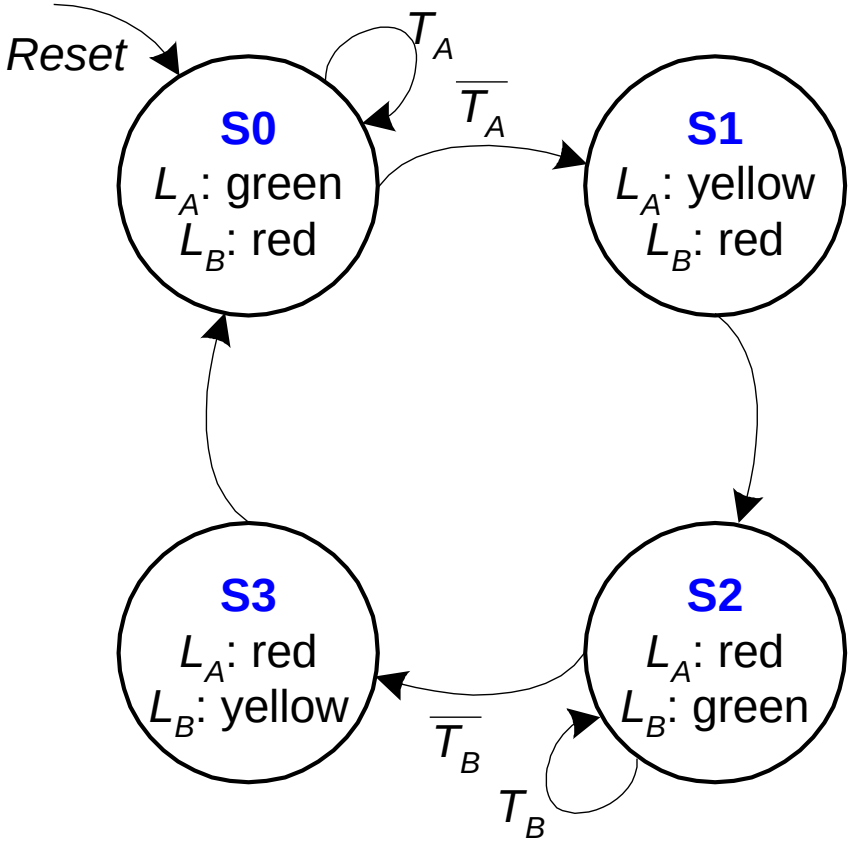


Tabella di transizione

Current State	Inputs		Next State
S	T_A	T_B	S'
S0	0	X	S1
S0	1	X	S0
S1	X	X	S2
S2	X	0	S3
S2	X	1	S2
S3	X	X	S0

Tabella di transizione

Current State		Inputs		Next State	
S_1	S_0	T_A	T_B	S'_1	S'_0
0	0	0	X	0	1
0	0	1	X	0	0
0	1	X	X	1	0
1	0	X	0	1	1
1	0	X	1	1	0
1	1	X	X	0	0

State	Encoding
S0	00
S1	01
S2	10
S3	11

$$S'_1 = S_1 \oplus S_0$$

$$S'_0 = \overline{S_1}\,\overline{S_0}\,\overline{T_A} + S_1\overline{S_0}\,\overline{T_B}$$

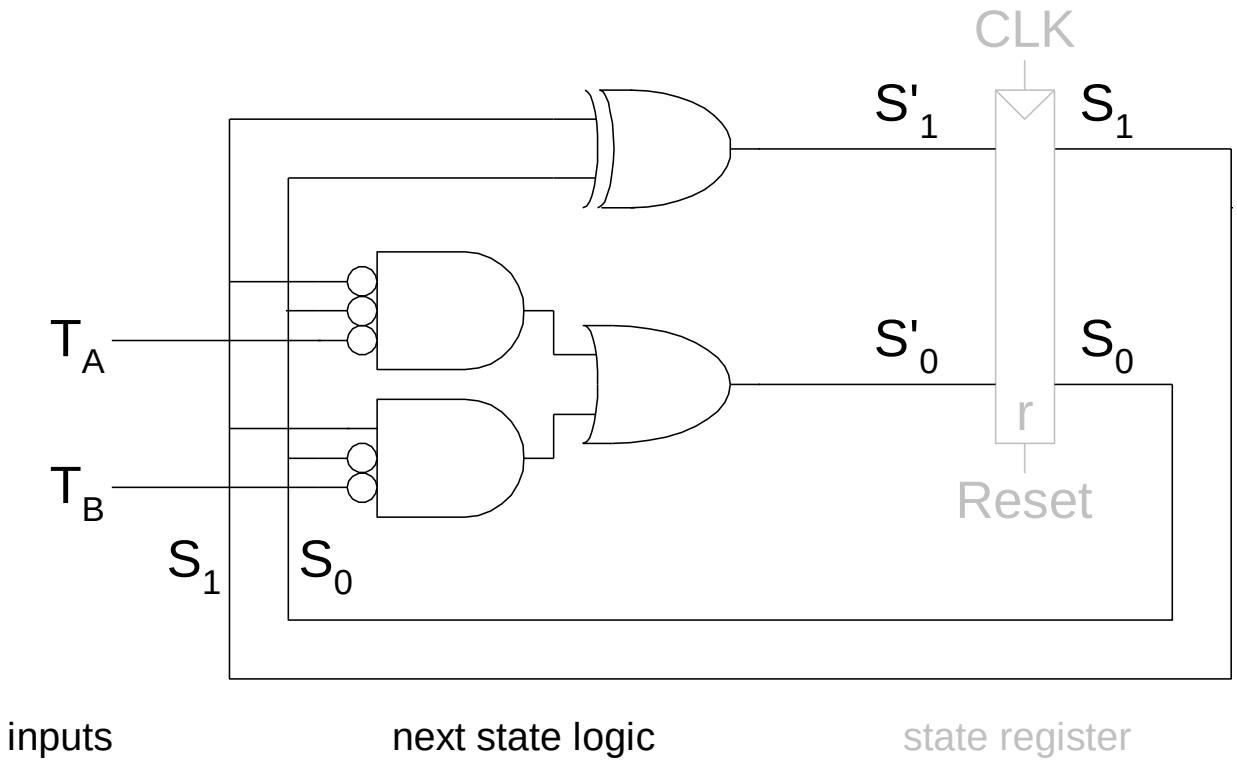
Tabella dell'output

Current State		Outputs			
S_1	S_0	L_{A1}	L_{A0}	L_{B1}	L_{B0}
0	0	0	0	1	0
0	1	0	1	1	0
1	0	1	0	0	0
1	1	1	0	0	1

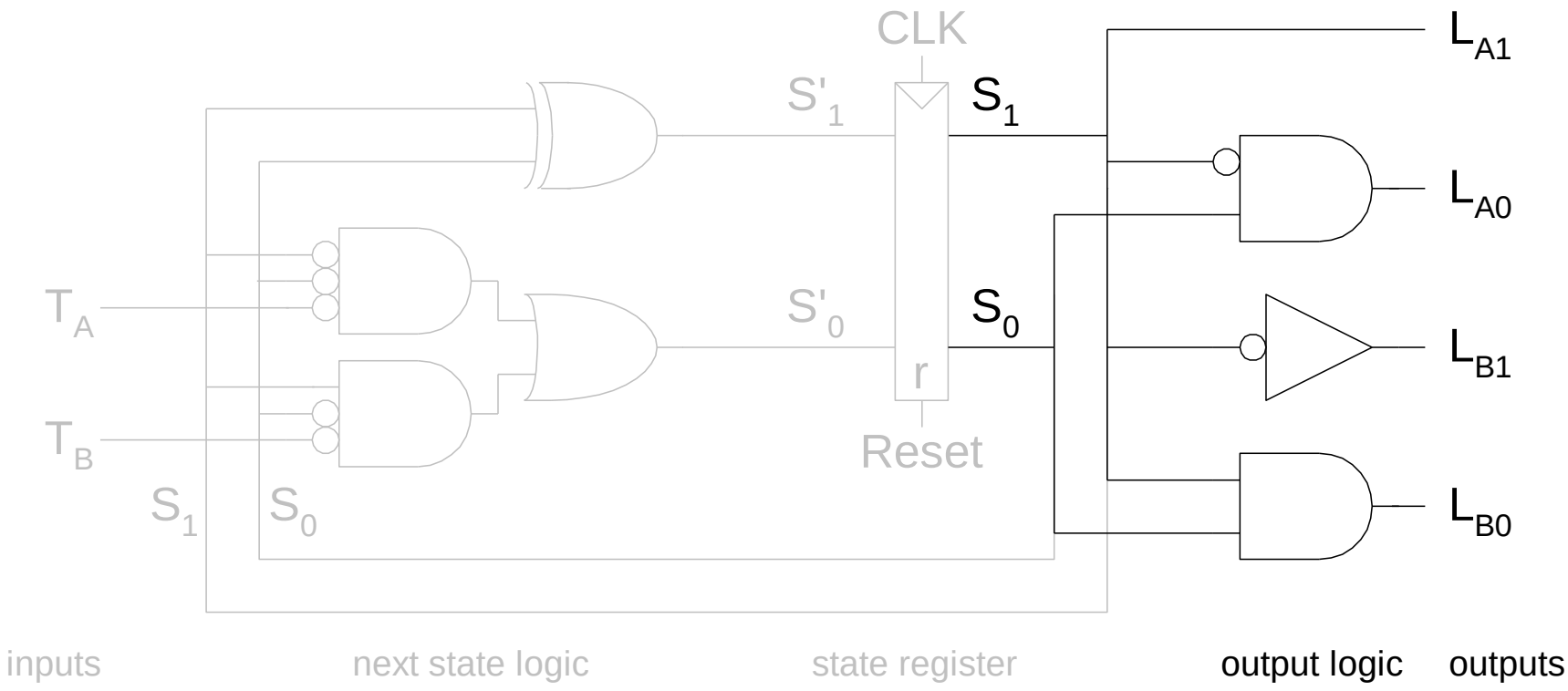
Output	Encoding
green	00
yellow	01
red	10

$$L_{A1} = S_1$$
$$L_{A0} = \overline{S_1} S_0$$
$$L_{B1} = \overline{S_1}$$
$$L_{B0} = S_1 S_0$$

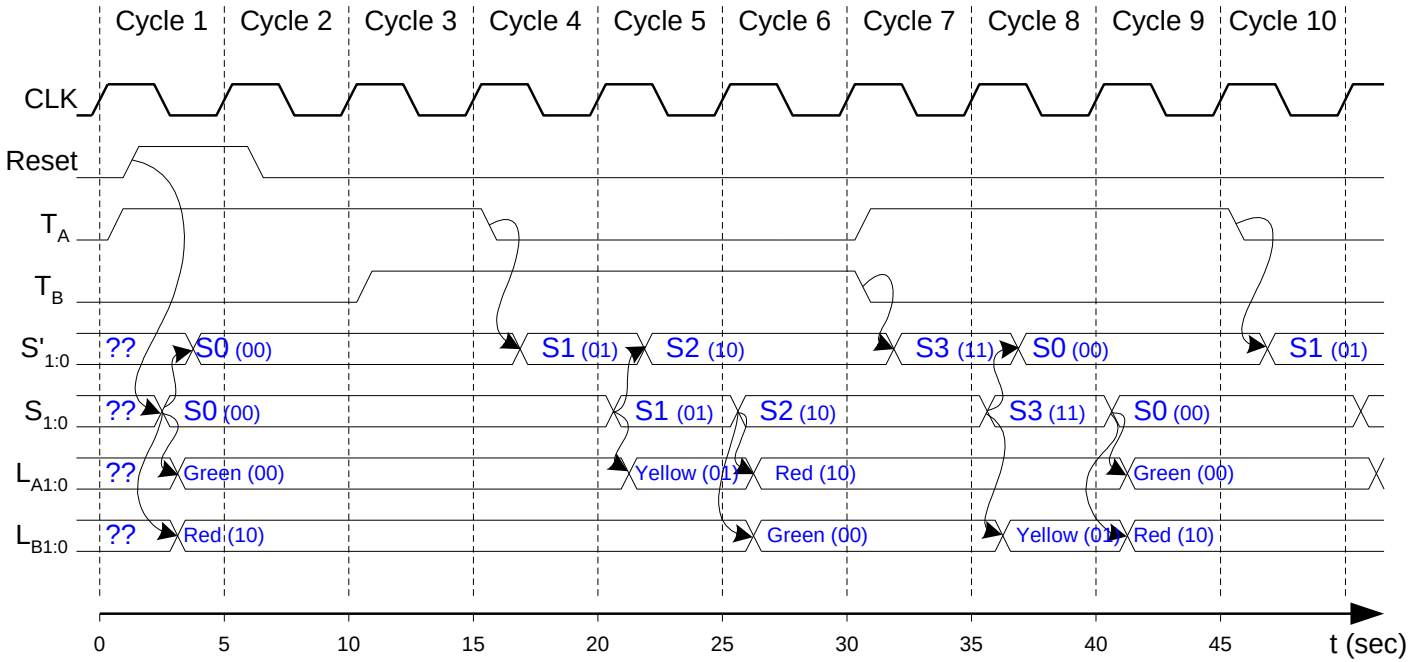
Schema della logica di transizione

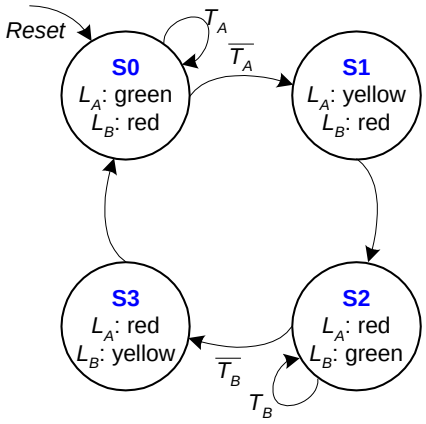


Schema della logica di output



FSM Timing Diagram



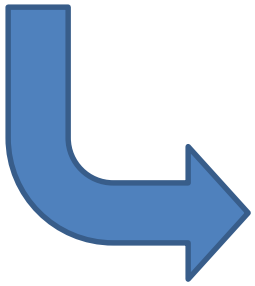


Encoding degli stati

- Encoding binario:
  - i.e., per 4 stati, 00, 01, 10, 11
- Encoding *one-hot*
  - Un bit per stato
  - Solo un bit HIGH alla volta
  - i.e., per 4 stati, 0001, 0010, 0100, 1000
  - Richiede più flip-flops
  - Spesso la logica combinatoria associata è più semplice

Moore vs Mealy FSM

Alyssa P. Hacker has a snail that crawls down a paper tape with 1's and 0's on it. The snail smiles whenever the last two digits it has crawled over are 01. Design Moore and Mealy FSMs of the snail's brain.



10011000101101

goal=1
dist_prox_goal ≥ 2
10011000101101

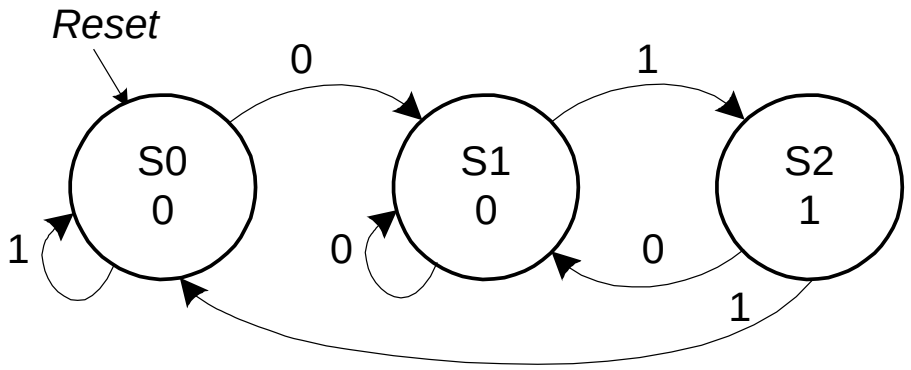
goal=0
dist_prox_goal≥2
10011000101101

goal=0
dist_prox_goal ≥1
10011000101101

Moore vs Mealy FSM

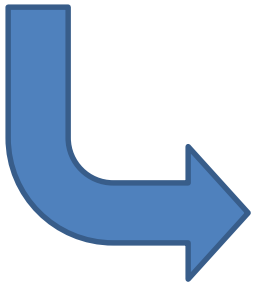
Alyssa P. Hacker has a snail that crawls down a paper tape with 1's and 0's on it. The snail smiles whenever the last two digits it has crawled over are 01. Design Moore and Mealy FSMs of the snail's brain.

Moore FSM



Moore vs Mealy FSM

Alyssa P. Hacker has a snail that crawls down a paper tape with 1's and 0's on it. The snail smiles whenever the last two digits it has crawled over are 01. Design Moore and Mealy FSMs of the snail's brain.



10011000101101

0→
goal=0; dist_prox_goal≥1
1→goal=0; dist_prox_goal≥2
10011000101101

0→goal=0; dist_prox_goal≥1
1→ goal=1; dist_prox_goal≥2
10011000101101

Moore vs Mealy FSM

Alyssa P. Hacker has a snail that crawls down a paper tape with 1's and 0's on it. The snail smiles whenever the last two digits it has crawled over are 01. Design Moore and Mealy FSMs of the snail's brain.

Mealy FSM

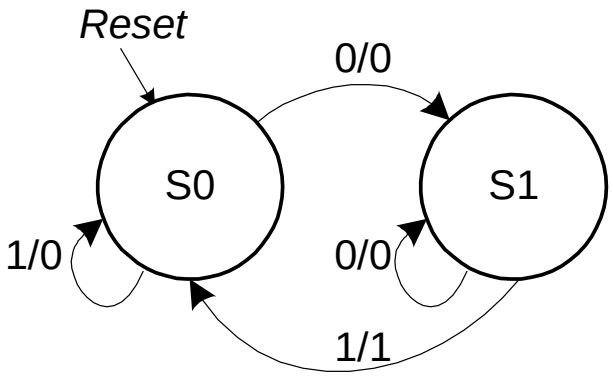


Tabella transizione Moore FSM

Current State		Inputs	Next State	
S_1	S_0	A	S'_1	S'_0
0	0	0	0	1
0	0	1	0	0
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0

State	Encoding
S0	00
S1	01
S2	10

$$S_1' = S_0 A$$
$$S_0' = \overline{A}$$

Tabella transizione Moore FSM

Current State		Inputs	Next State	
S_1	S_0	A	S'_1	S'_0
0	0	0	0	1
0	0	1	0	0
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0

State	Encoding
S0	00
S1	01
S2	10

$$S_1' = S_0 A$$
$$S_0' = \overline{A}$$

Manca S_1 negato, perché?

Tabella output Moore FSM

Current State		Output
S_1	S_0	Y
0	0	0
0	1	0
1	0	1

$$Y = S_1$$

Schema Moore FSM

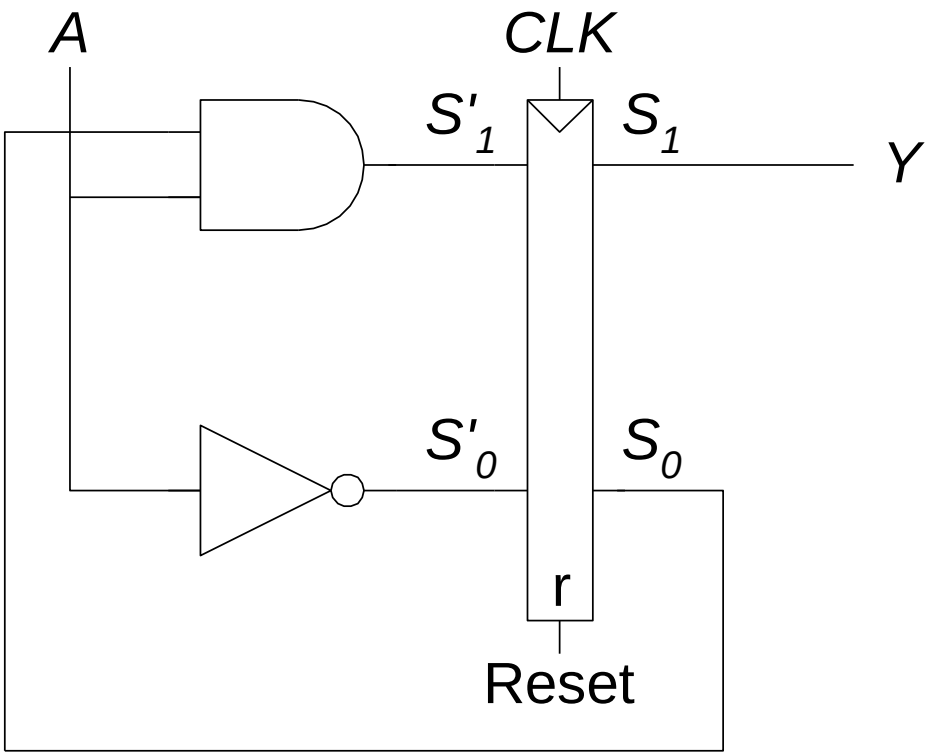


Tabella transizione/output Mealy FSM

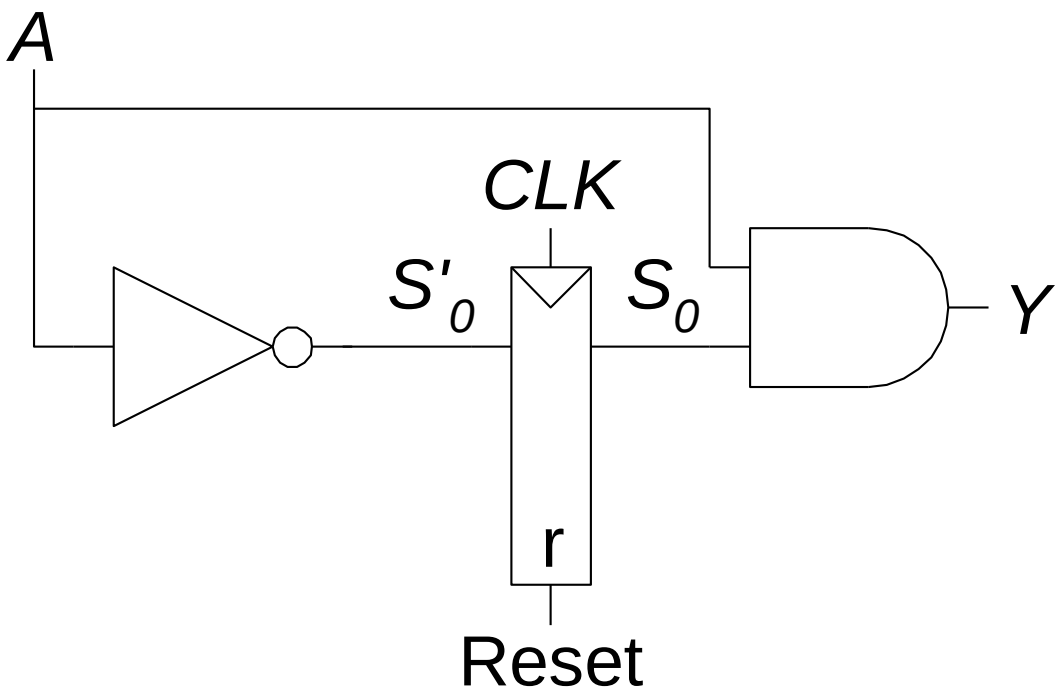
Current State	Input	Next State	Output
S	A	S'	Y
0	0	1	0
0	1	0	0
1	0	1	0
1	1	0	1

State	Encoding
S0	0
S1	1

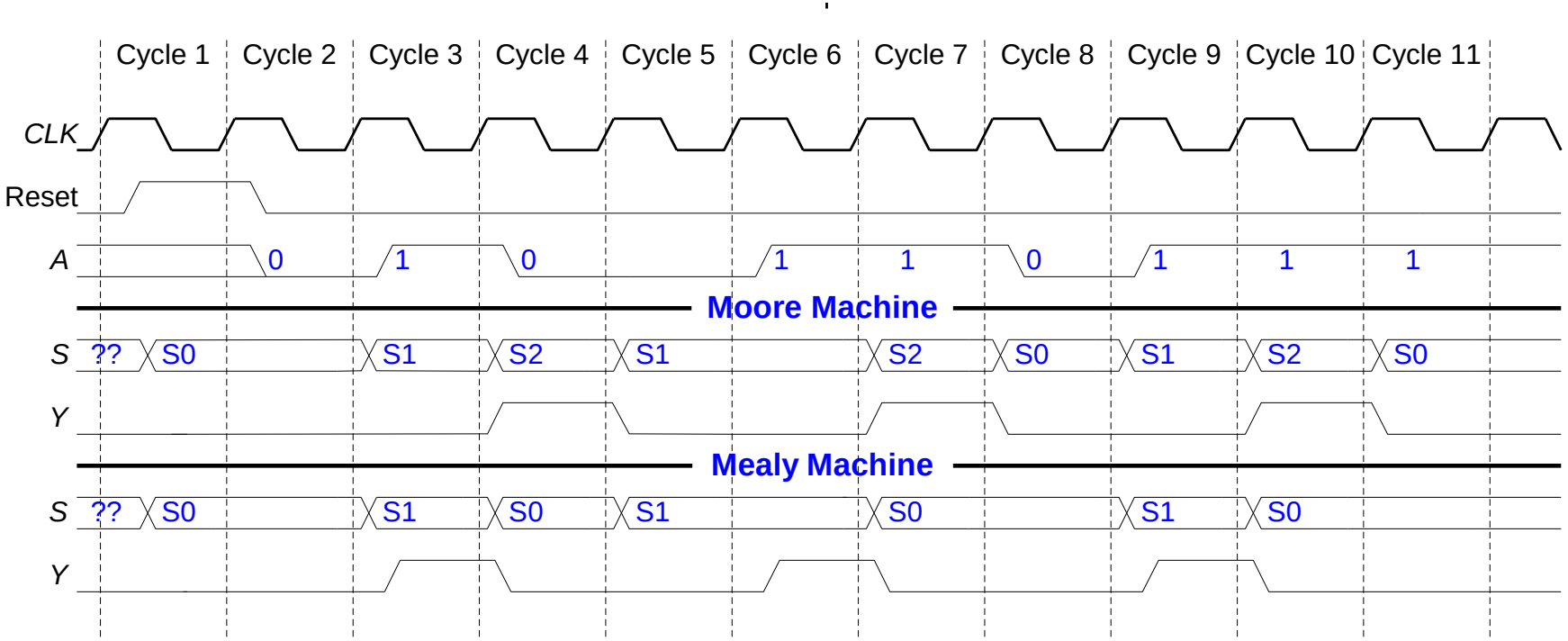
$$S' = \overline{A}$$

$$Y = S\,A$$

Schema Mealy FSM



Moore & Mealy Timing

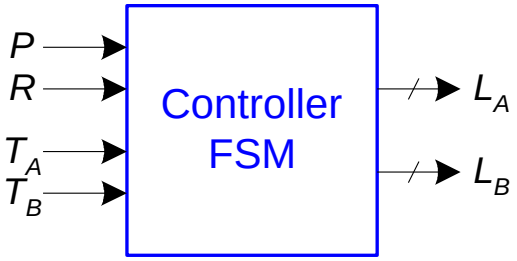


Fattorizzazione di FSM

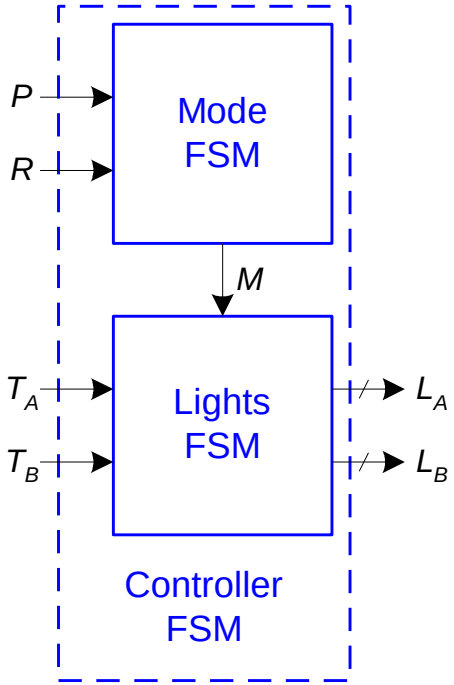
- Fattorizzare consiste nel suddividere una FSM complessa in FSM più piccole che interagiscono fra loro
- Esempio: Considerate di voler modificare il controller di semafori per tener conto di possibili parate
  - Altri due inputs: *P*, *R*
  - Se ***P* = 1**, entra in modalità *Parade* e il semaforo di Bravado Blvd rimane verde
  - Se ***R* = 1**, lascia la modalità *Parade*

Parade FSM

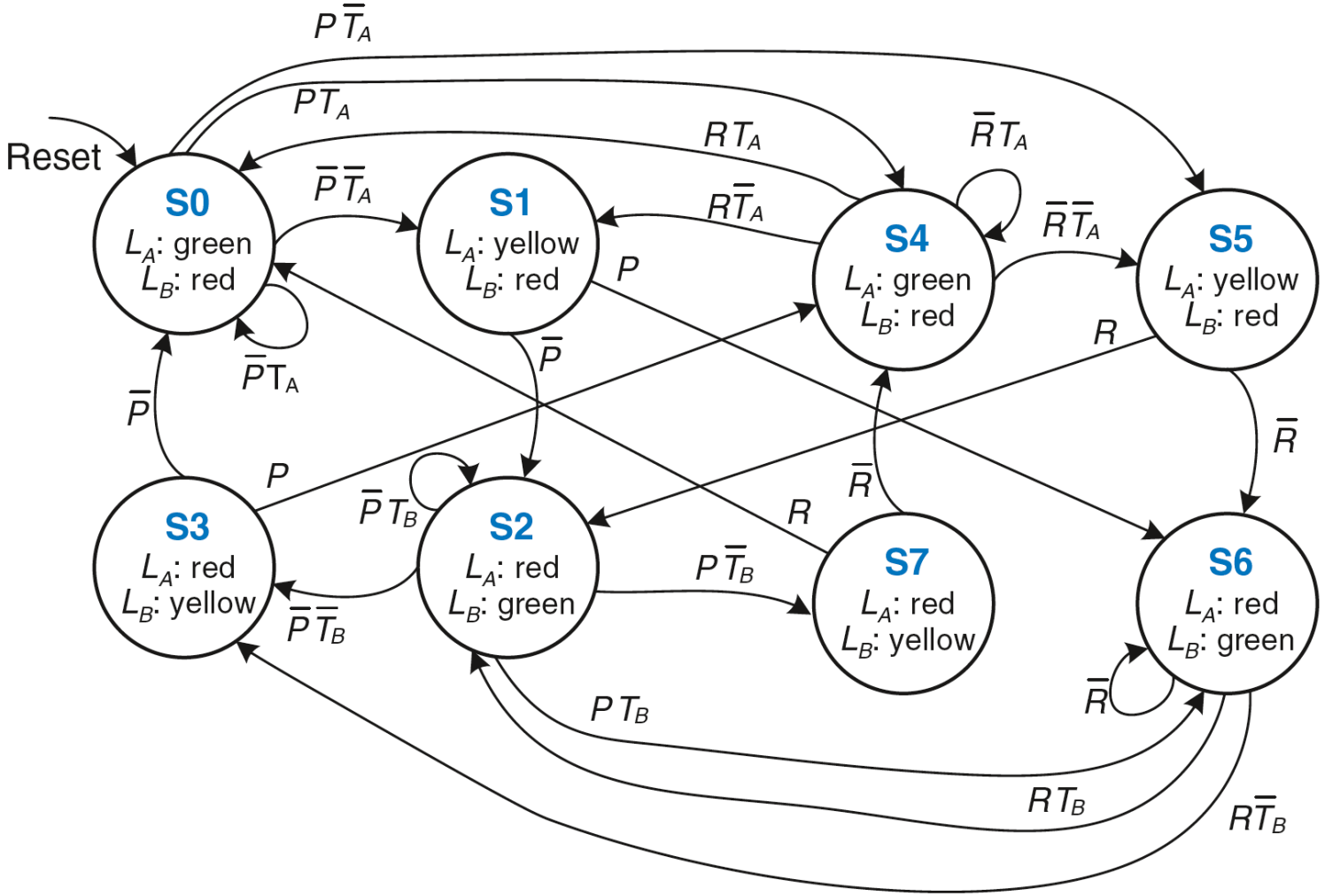
FSM non fattorizzato



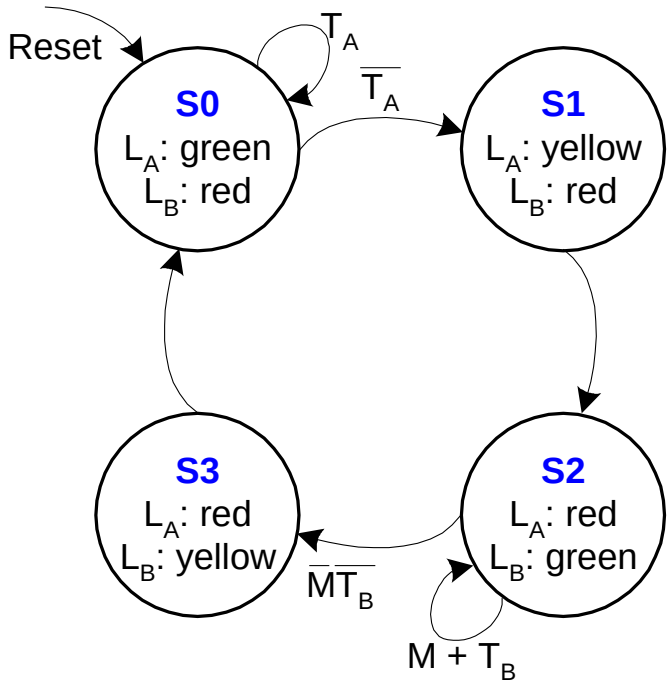
FSM fattorizzato



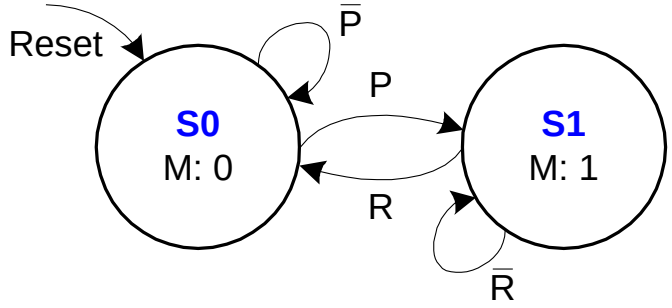
FSM non fattorizzato



FSM fattorizzato



Lights FSM



Mode FSM

Progettare una FSM

1. Identificare gli input e output
2. Abbozzare uno state transition diagram
3. Scrivere la state transition table
4. Selezionare un encoding degli stati
5. Macchina di Moore:
   a. Riscrivere la state transition table con l'encoding degli stati
   b. Scrivere la output table
5. Macchina di Mealy:
   combinare la state transition table e la output table con gli encoding degli stati
6. Scrivere le equazioni booleane relative alla logica di prossimo stato e alla logica di output
7. Minimizzare le equazioni
8. Fare uno schema del circuito

Esempio

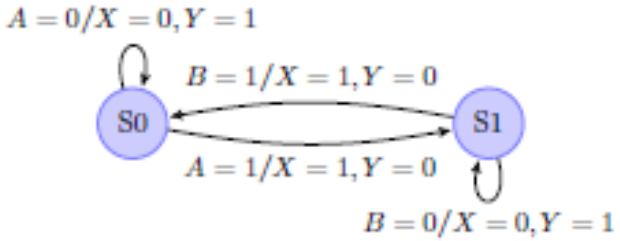
- Progettare una Mealy FSM F con due input (A e B) e un output Q.
  - Q=1 sse A e B assumono rispettivamente il valore precedente
  - es:

A	0	0	1	0	0	0	1	1	0
B	1	1	1	1	1	0	1	1	1
Q	0	1	0	0	1	0	0	1	0

Esercizi

- Esercizi 3.23, 3.31

3. Il seguente diagramma di transizione per una macchina di Mealy ha due input A e B e due output X e Y. Indicare le formule SOP **minime** relative alla variabile di stato (S) e alle due variabili di output.



Codifica dello stato:

stato	S
S0	0
S1	1

Formule minime SOP:

- S': _____
- X: _____
- Y: _____

Parallelismo

- **2 tipi di parallelismo:**
  - **Spaziale**
    - duplicare l'hardware per eseguire più task contemporaneamente
  - **Temporale**
    - Il task è suddiviso in più fasi
    - Le diverse fasi sono eseguite in pipelining

Latenza e throughput

- **Token:** Gruppo di input da processare per ottenere un output significativo
- **Latency:** Tempo che occorre ad un token per essere processato e produrre un output
- **Throughput:** Numero di output prodotti per unità di tempo

il parallelismo incrementa il throughput

Esempio di parallelismo

- Ben vuole fare delle torte
- 5 minuti per preparare una torta
- 15 minuti per cucinarla

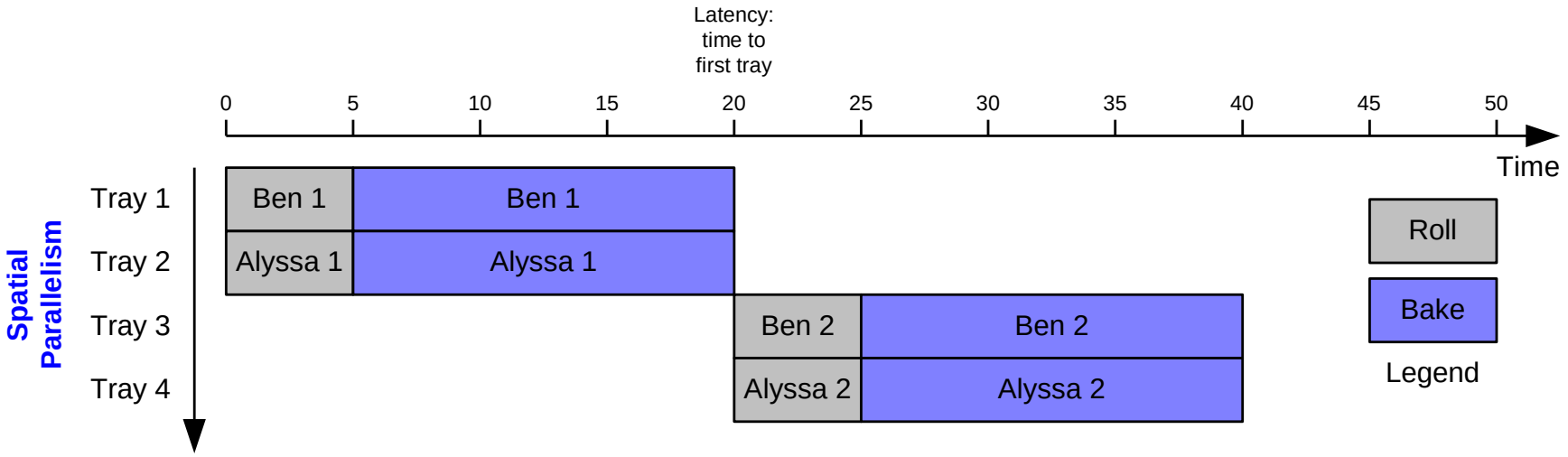
Latency = 5 + 15 = 20 minuti = **1/3 h**

Throughput = $\frac{1}{latency}$ = **3 torte/h**

Esempio di parallelismo

- **parallelismo spaziale:** Ben chiede a Allysa di aiutarlo, usando anche il suo forno
- **parallelismo temporale:**
  - 2 fasi: preparare e cucinare
  - Mentre una torta è in forno, Ben prepara ne prepara un'altra, etc.

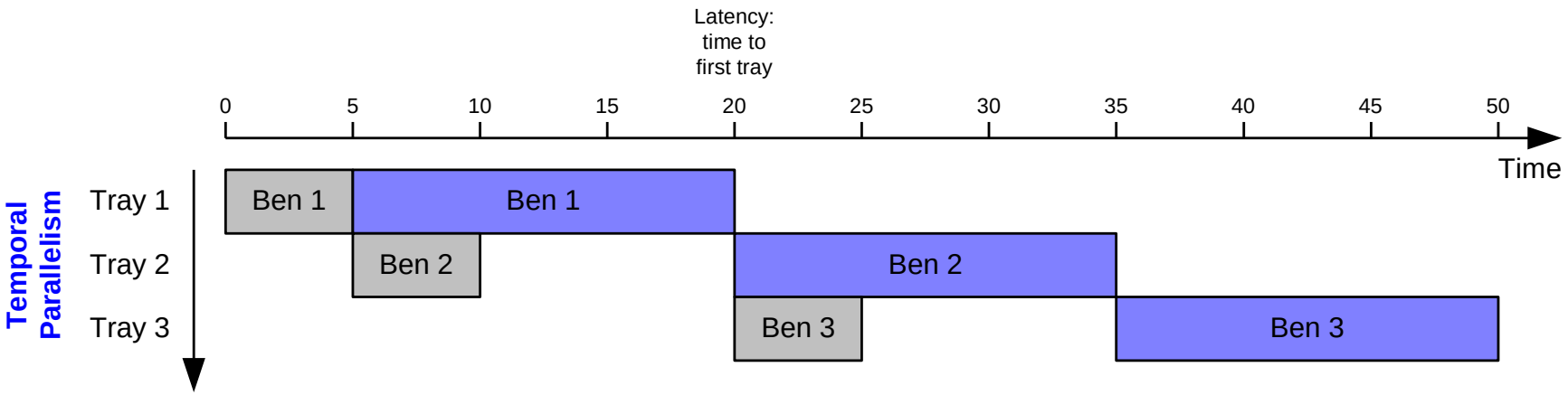
Parallelismo spaziale



Latency = 5 + 15 = 20 minuti = **1/3 h**

Throughput = $2 \frac{1}{latency}$ = **6 torte/h**

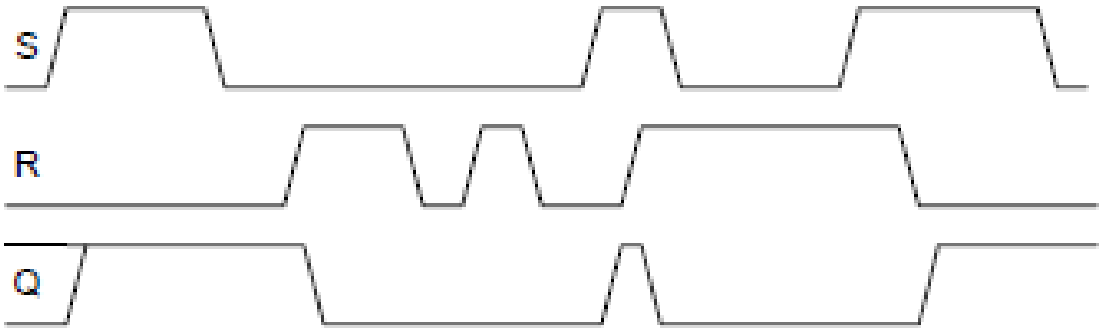
Parallelismo temporale



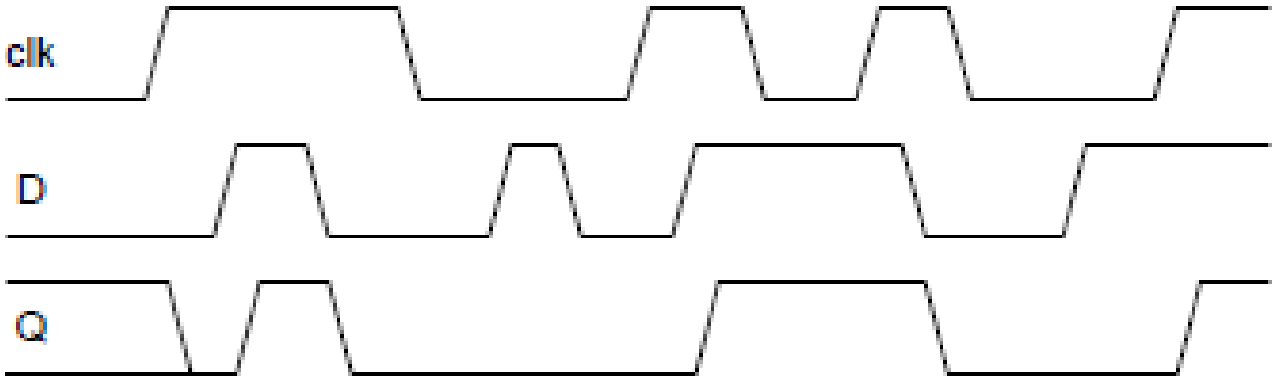
Latency = 5 + 15 = 20 minuti = **1/3 h**

Throughput $\approx \frac{4}{65} 60 \approx$ **3,7 torte/h**

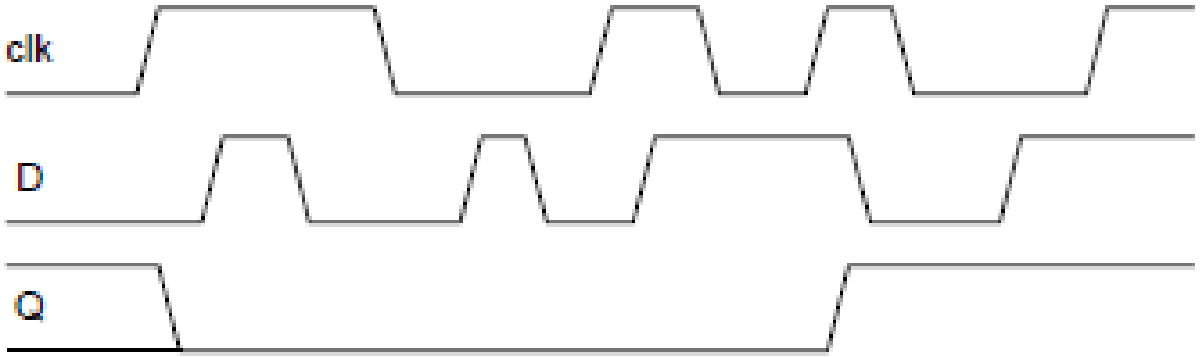
Esercizio 3.1



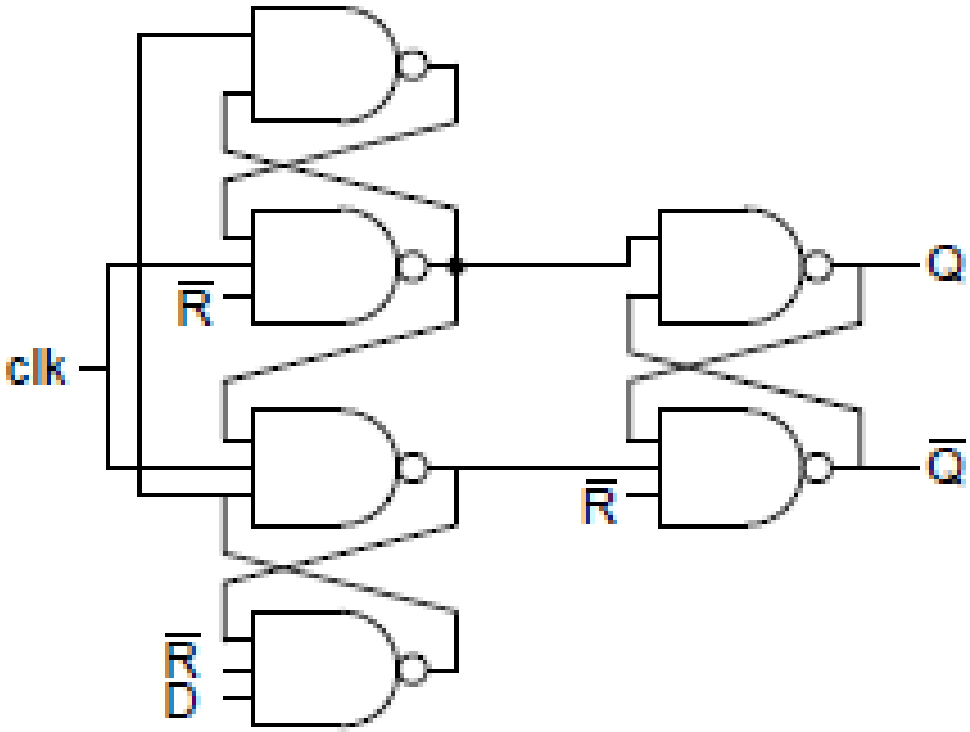
Esercizio 3.3



Esercizio 3.5

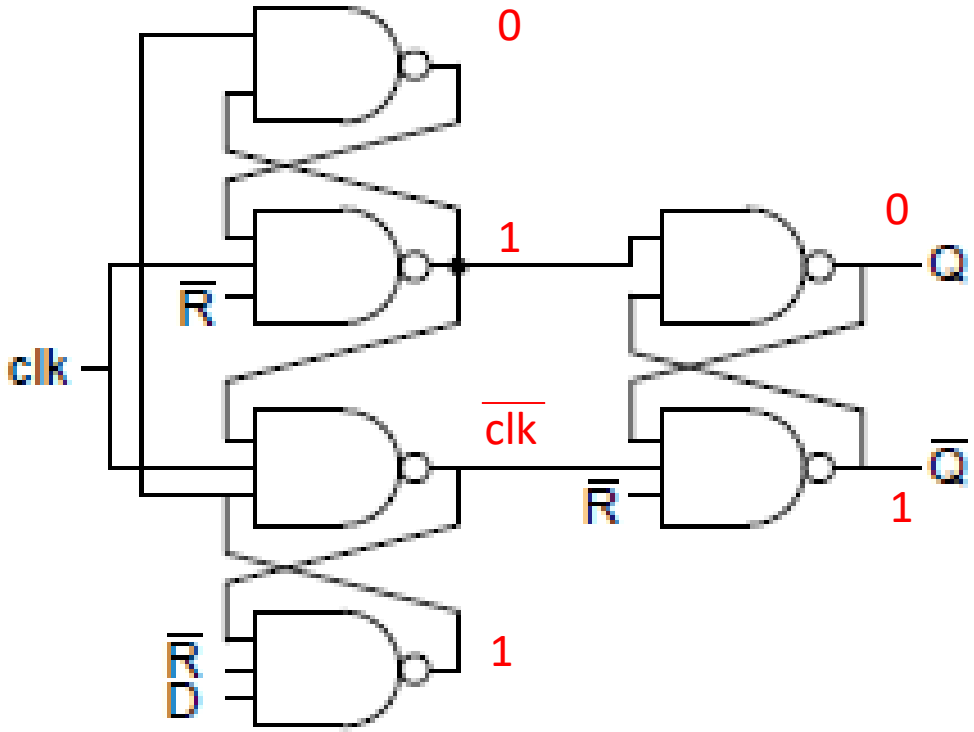


Esercizio 3.13



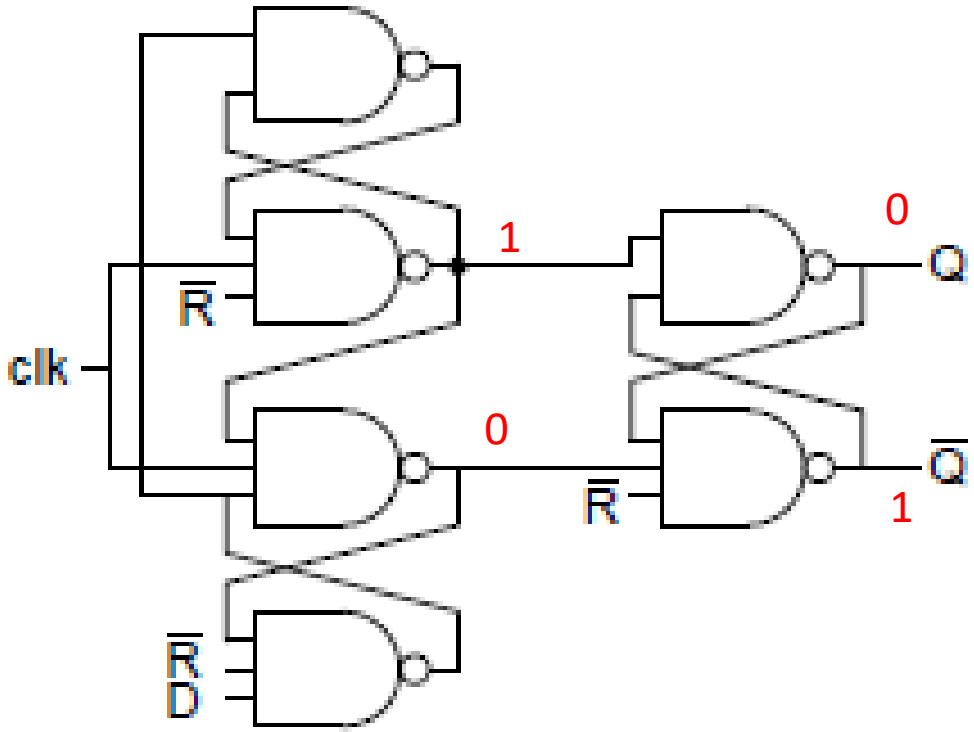
Esercizio 3.13

R=1



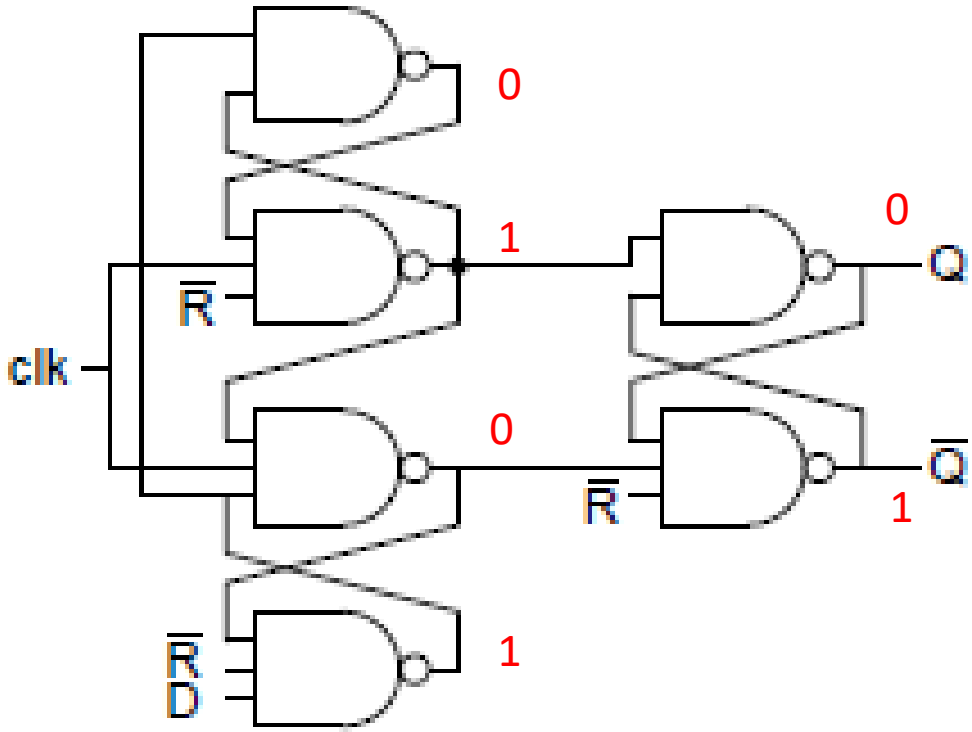
Esercizio 3.13

R→0 clk=1



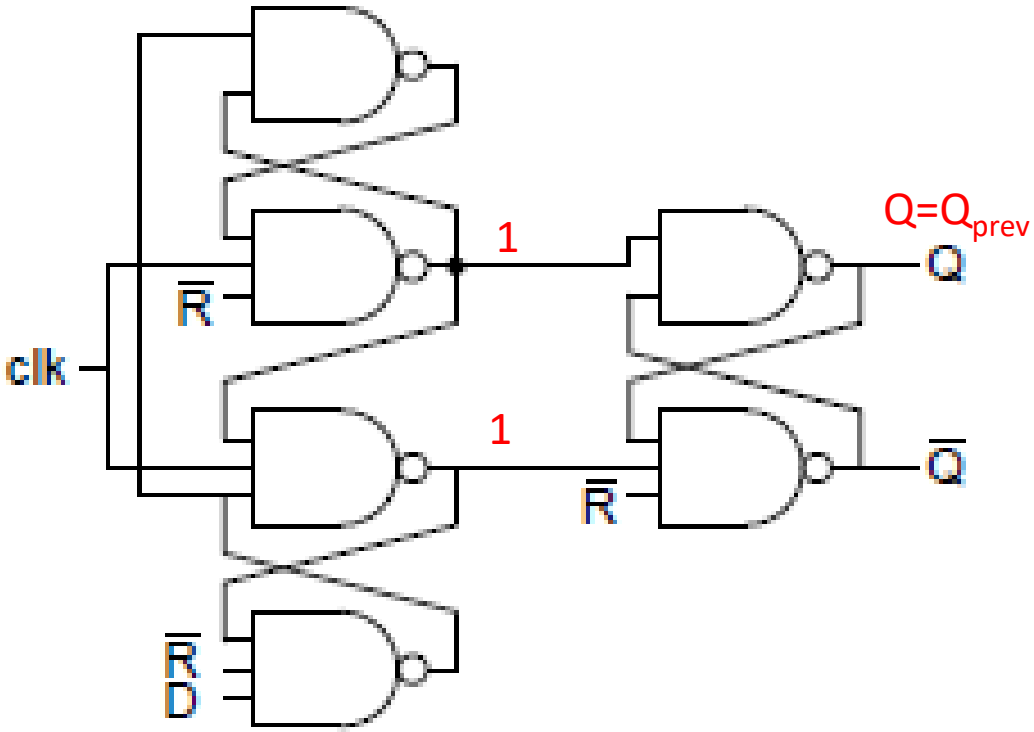
Esercizio 3.13

R→0 clk=1



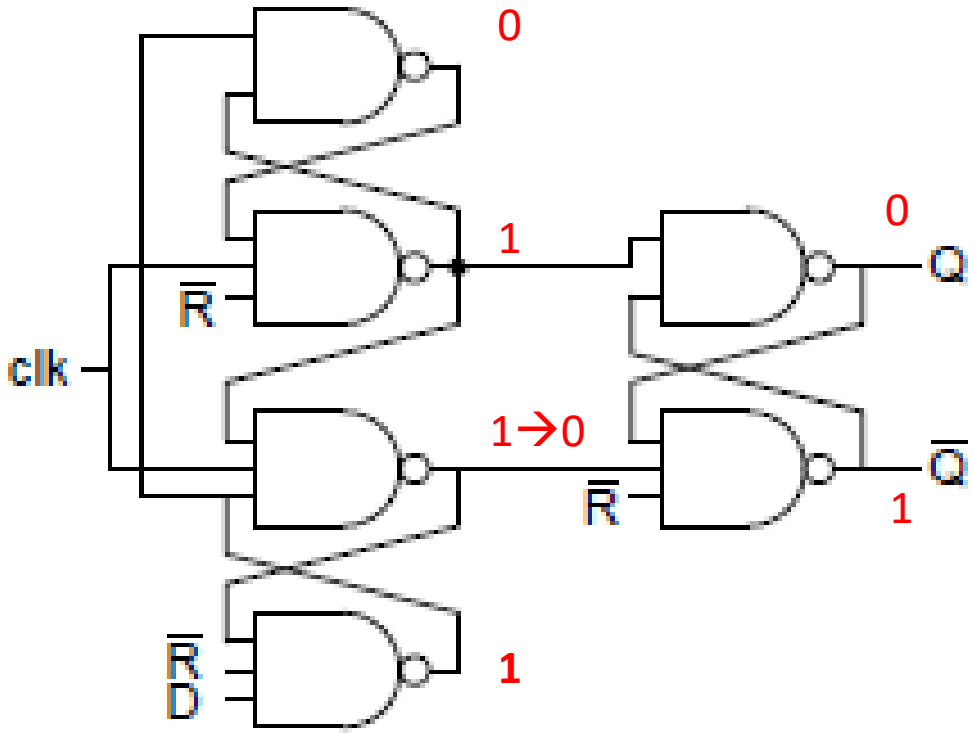
Esercizio 3.13

R=0, clk=0



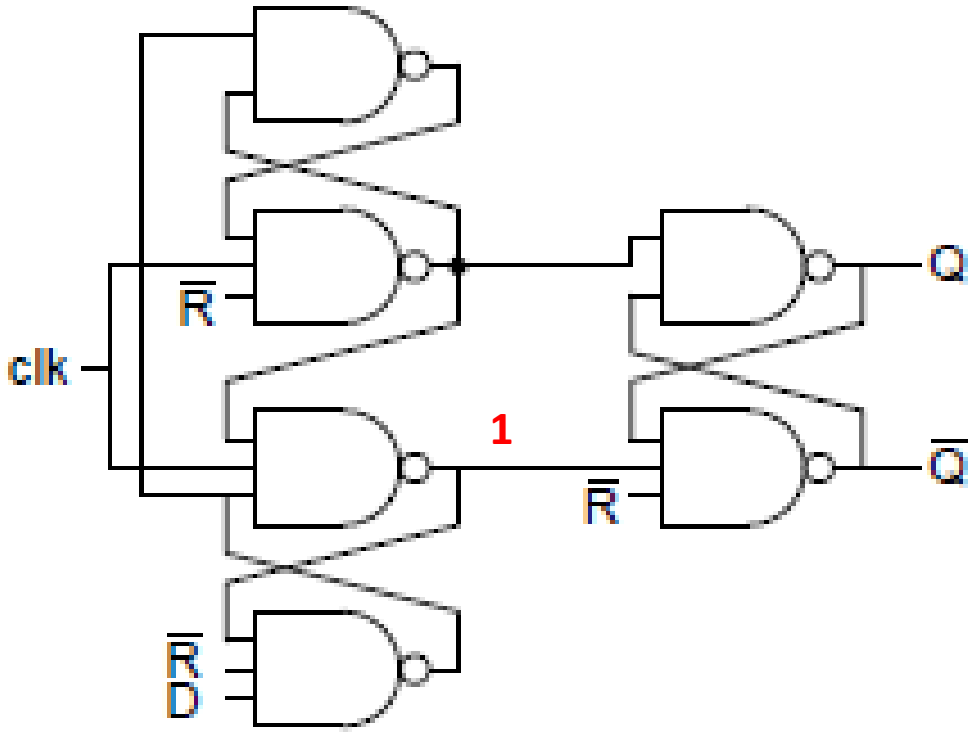
Esercizio 3.13

R=0 clk→1 D=0



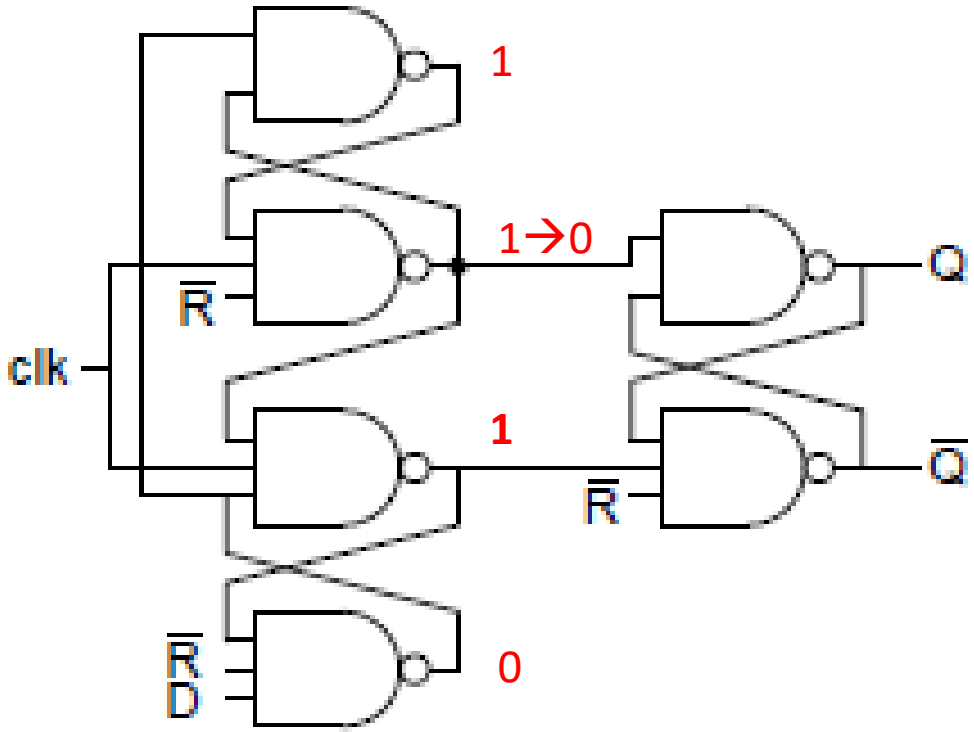
Esercizio 3.13

R=0 clk→1 D=1



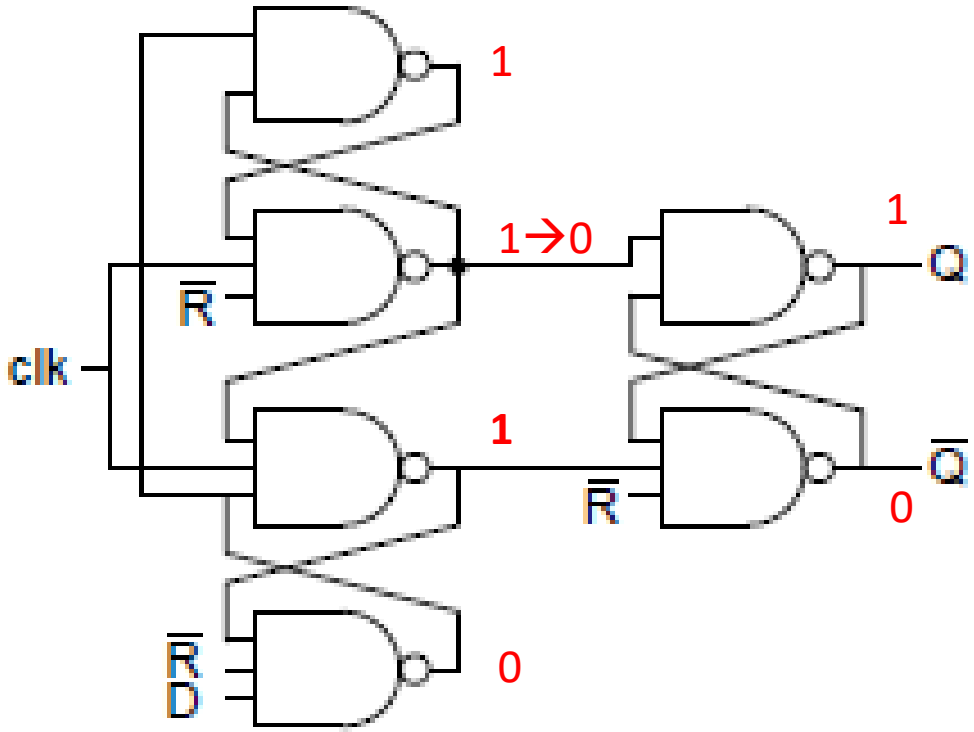
Esercizio 3.13

R=0 clk→1 D=1

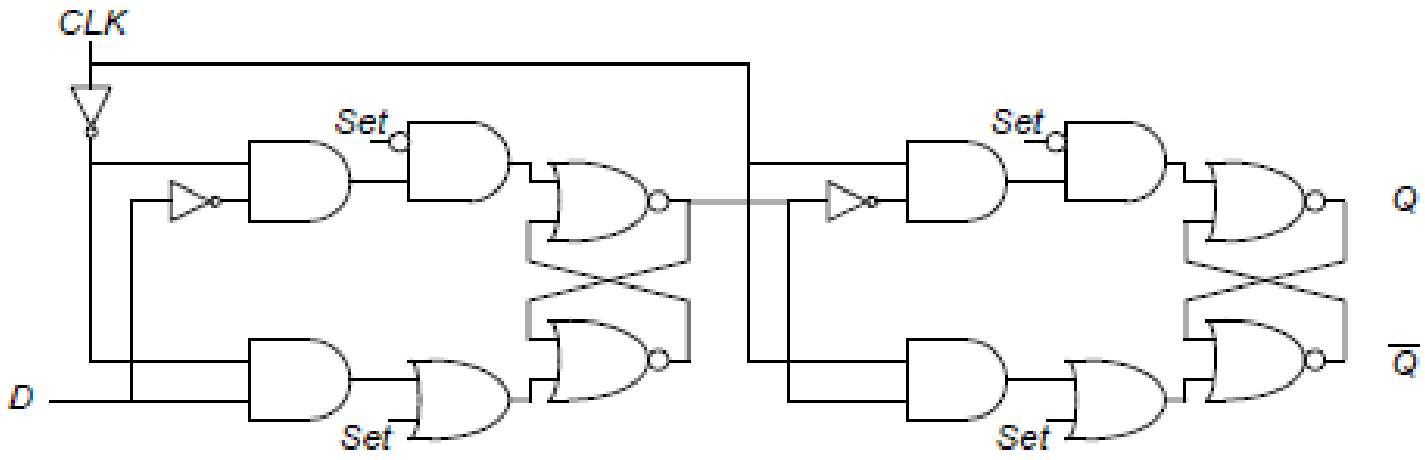


Esercizio 3.13

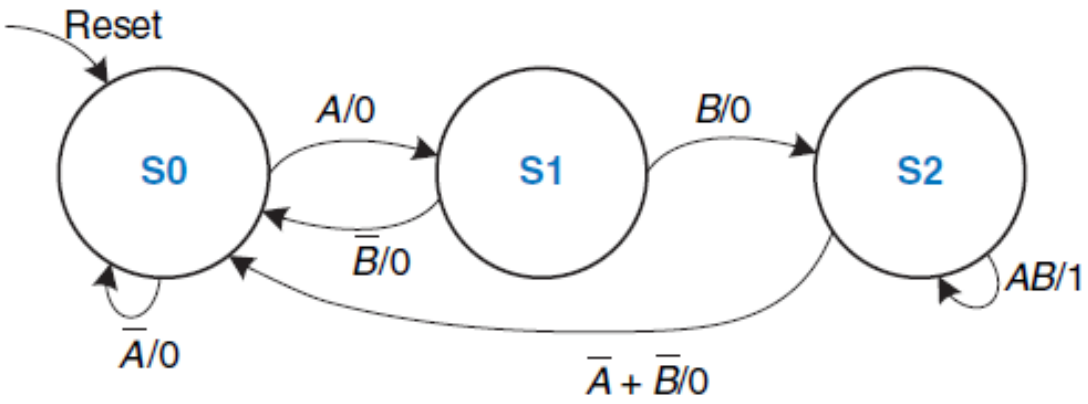
R=0 clk→1 D=1



Esercizio 3.15



Esercizio 3.23



Esercizio 3.23

current state		inputs		next state		output
s_1	s_0	a	b	s'_1	s'_0	q
0	0	0	X	0	0	0
0	0	1	X	0	1	0
0	1	X	0	0	0	0
0	1	X	1	1	0	0
1	0	1	1	1	0	1
1	0	0	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	0	0	0

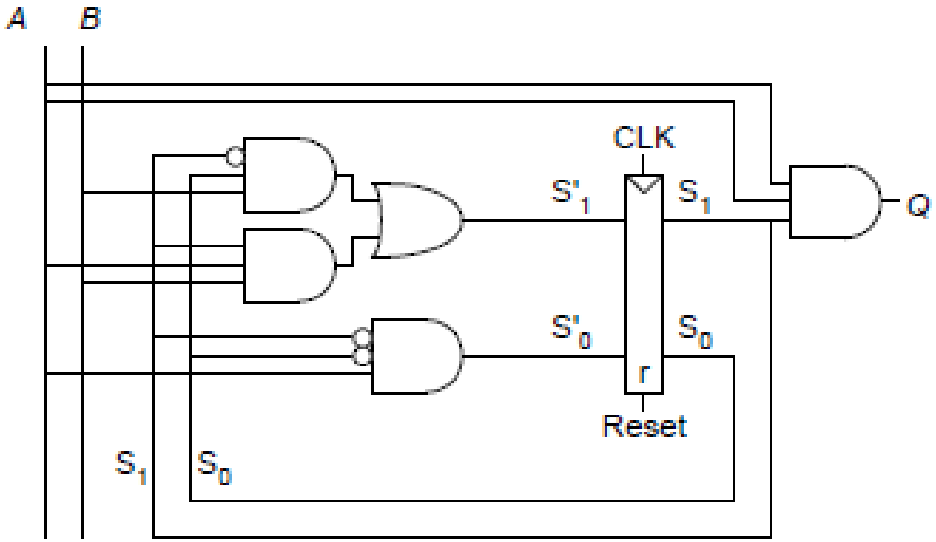
state	encoding $s_{1:0}$
S0	00
S1	01
S2	10

Esercizio 3.23

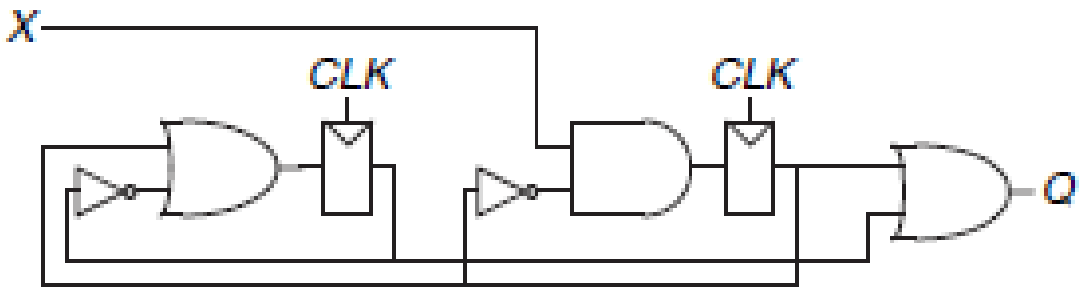
$$S'_1 = \overline{S_1} S_0 B + S_1 A B$$

$$S'_0 = \overline{S_1}\,\overline{S_0} A$$

$$Q' = S_1 A B$$



Esercizio 3.31



Esercizio 3.31

current state		input	next state	
s_1	s_0	x	s'_1	s'_0
0	0	0	0	1
0	0	1	1	1
0	1	0	0	0
0	1	1	1	0
1	X	X	0	1

TABLE 3.7 State transition table with binary encodings for Exercise 3.31

current state		output
s_1	s_0	q
0	0	0
0	1	1
1	X	1

TABLE 3.8 Output table for Exercise 3.31

Esercizio 3.31