Circuiti sequenziali

AA 2019-2020

Logica sequenziale

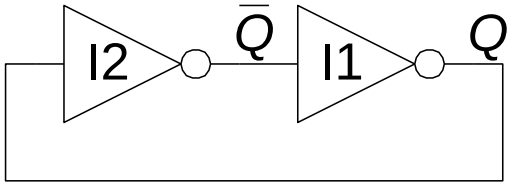
- Nei sistemi sequenziali l'output dipende sia dal valore corrente sia da valori precedenti dell'input. In tal senso si dice che il sistema ha *memoria*
- *Stato interno:* tutta l'informazione di un circuito che mantiene la *storia* di quel circuito ed è necessaria per prevedere il suo comportamento 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

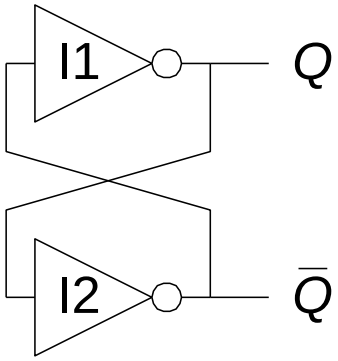
State elements

- 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



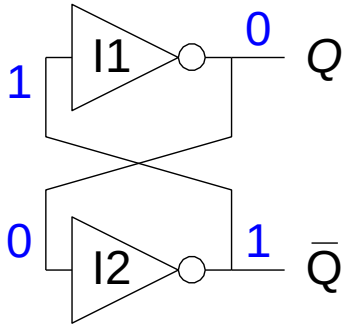


Analisi del circuito bistabile

- Considera i due possibili casi:

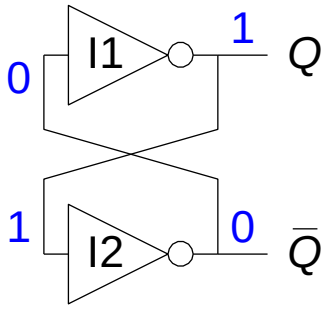
  Q = 0:

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



  Q = 1:

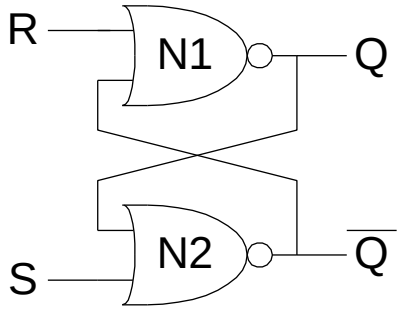
  allora $Q = 1$, $\bar{Q} = 0$ (consistente)



- 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:

  $S = 1, R = 0$

  $S = 0, R = 1$

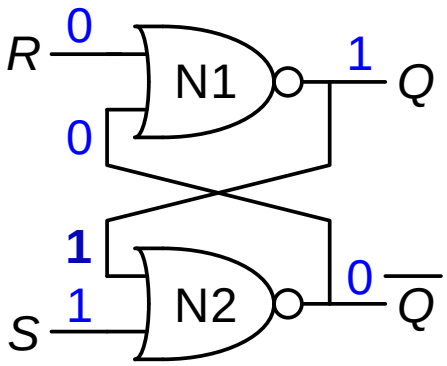
  $S = 0, R = 0$

  $S = 1, R = 1$

Analisi di un SR Latch

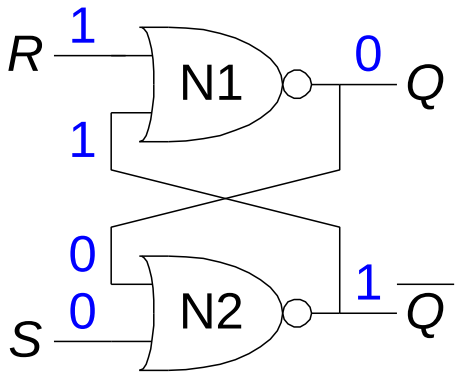
***S* = 1, *R* = 0:**

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



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

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

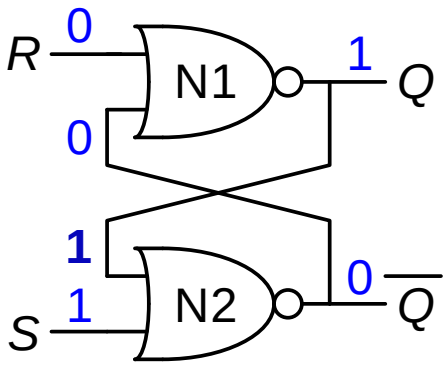


Analisi di un SR Latch

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

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

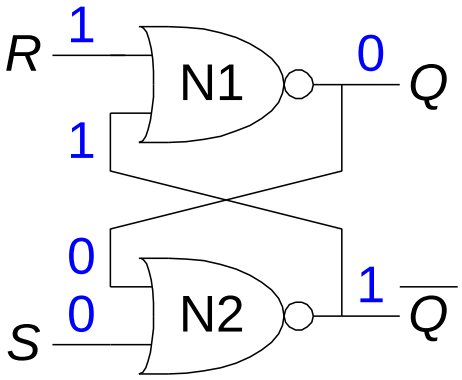
operazione Set



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

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

operazione Reset

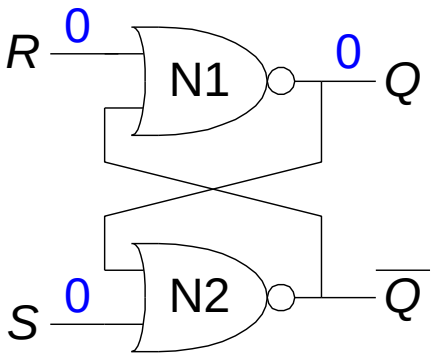


Analisi di un SR Latch

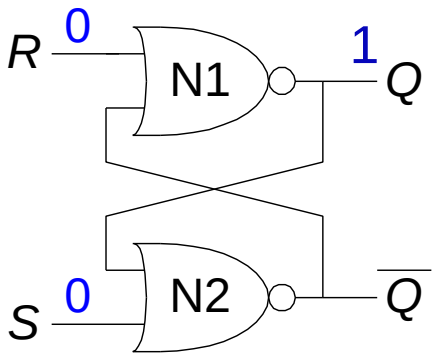
$S = 0, R = 0$:

allora $Q = Q_{prev}$

$Q_{prev} = 0$

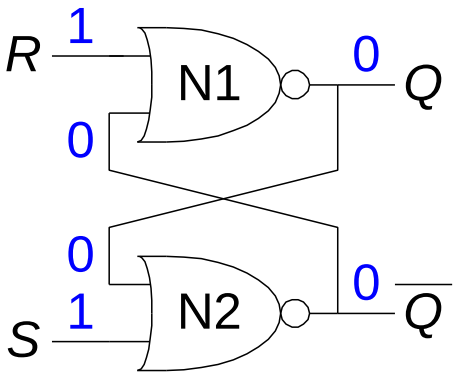


$Q_{prev} = 1$



$S = 1, R = 1$:

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



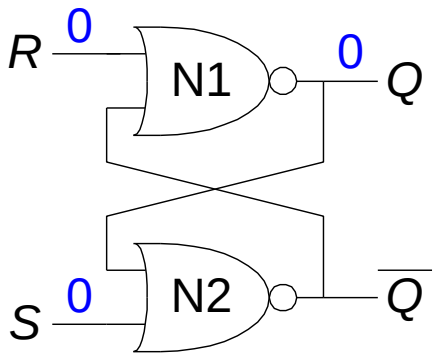
Analisi di un SR Latch

$S = 0, R = 0$:

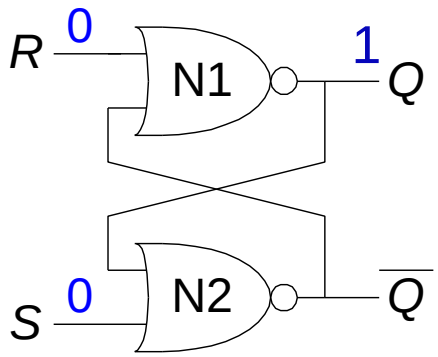
allora $Q = Q_{prev}$

Memoria

$Q_{prev} = 0$



$Q_{prev} = 1$

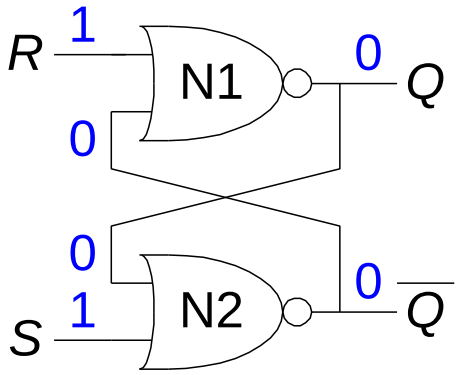


$S = 1, R = 1$:

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

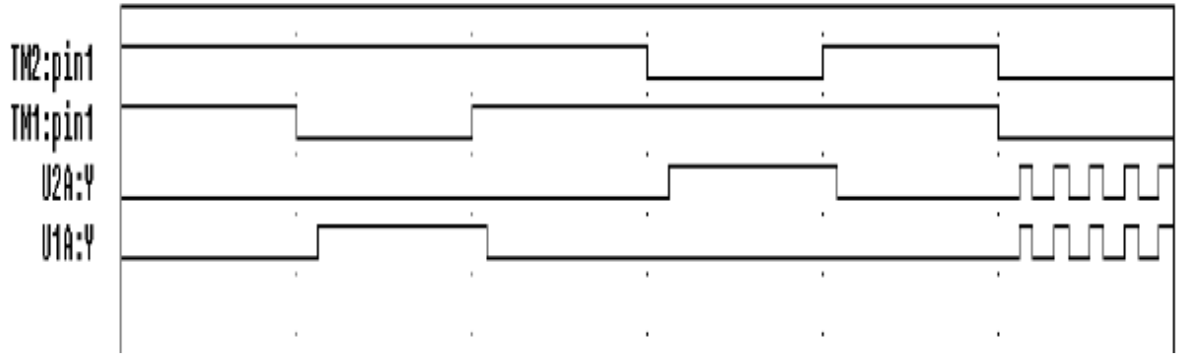
$Q \neq$ NOT Q

-> potenziali problemi !



Analisi di un SR Latch

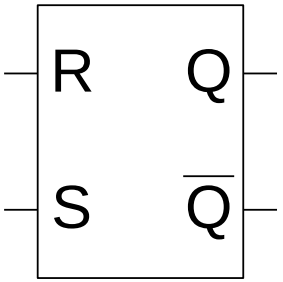
- Se dalla condizione S=R=1 si passa alla condizione S=R=0 allora
  - Se i tempi di propagazione sono uguali allora il circuito va in oscillazione
  - Nell'ipotesi, più realistica che le porte abbiano ritardi anche lievemente differenti, il circuito si mette in uno dei due stati possibili. Anche in questo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



Per quanto osservato, occorre evitare transizioni

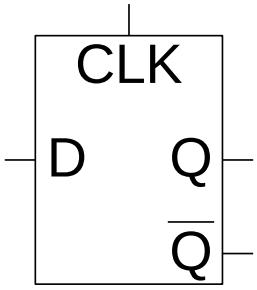
$$S = R = 1 \rightarrow S = R = 0$$

D Latch con abilitazione

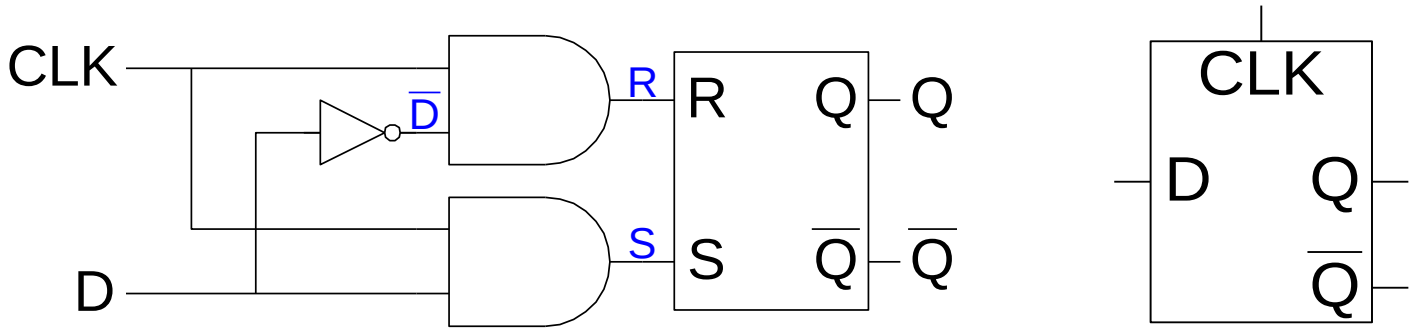
- 2 input: *CLK*, *D*
- ***CLK*:** controlla *quando* l'output cambia
- ***D*** (data input): controlla *in che cosa* l'output cambia
- Se ***CLK* = 1**,
  D passa fino a Q (*transparente*)
- Se ***CLK* = 0**,
  Q mantiene il suo valore precedente (*opaco*)

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

D Latch Symbol

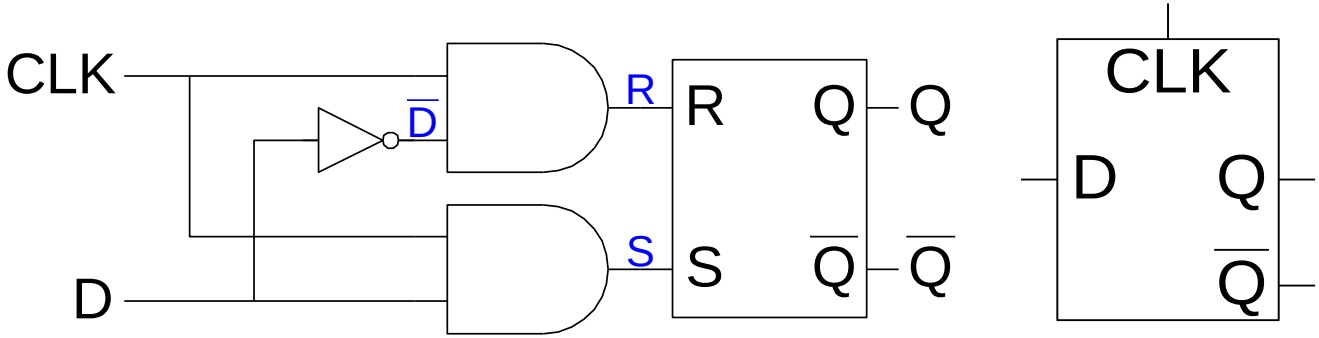


D Latch



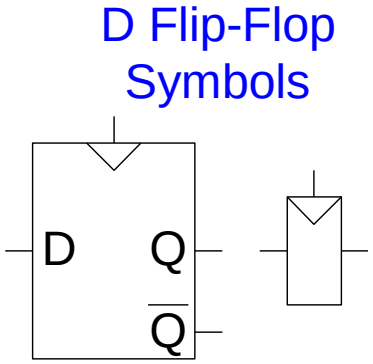
CLK	*D*	$\overline{D}$	*S*	*R*	*Q*	$\overline{Q}$
0	X	$\overline{X}$	0	0	Q_{prev}	$\overline{Q}_{prev}$
1	0	1	0	1	0	1
1	1	0	1	0	1	0

D Latch


CLK
D
$\overline{D}$
R
S
R
S
Q
$\overline{Q}$
Q
$\overline{Q}$
CLK
D
Q
$\overline{Q}$

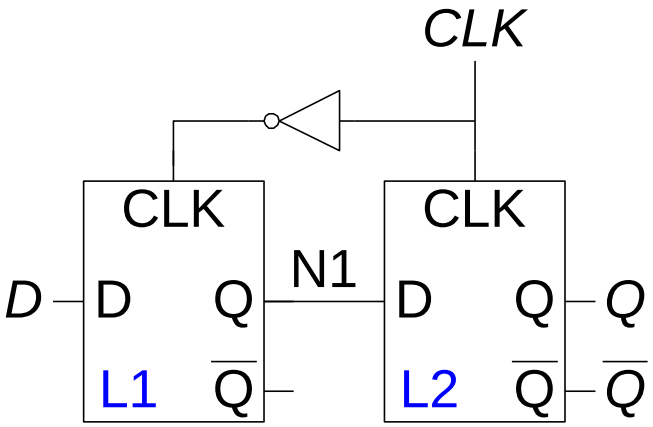
D Flip-Flop

- 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



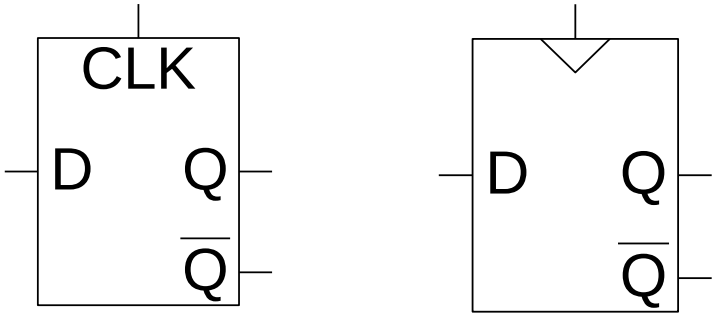
D Flip-Flop Master/Slave

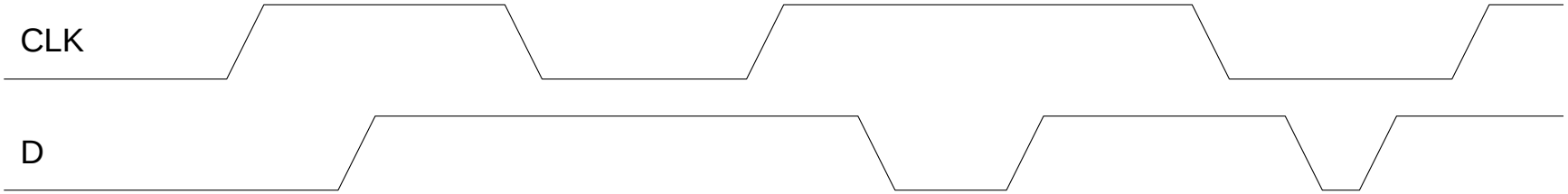
- 2 D latch (L1 e L2) controllati da clock complementari
- Quando ***CLK* = 0**
  - L1 è trasparente
  - L2 è opaco
  - *D* passa fino a N1
- Quando ***CLK* = 1**
  - L2 è trasparente
  - L1 è opaco
  - N1 passa fino a *Q*



- 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

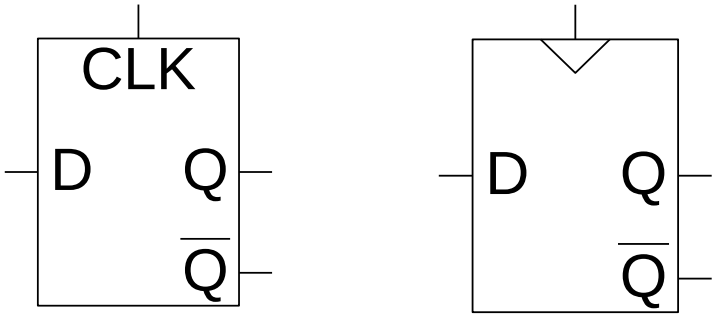


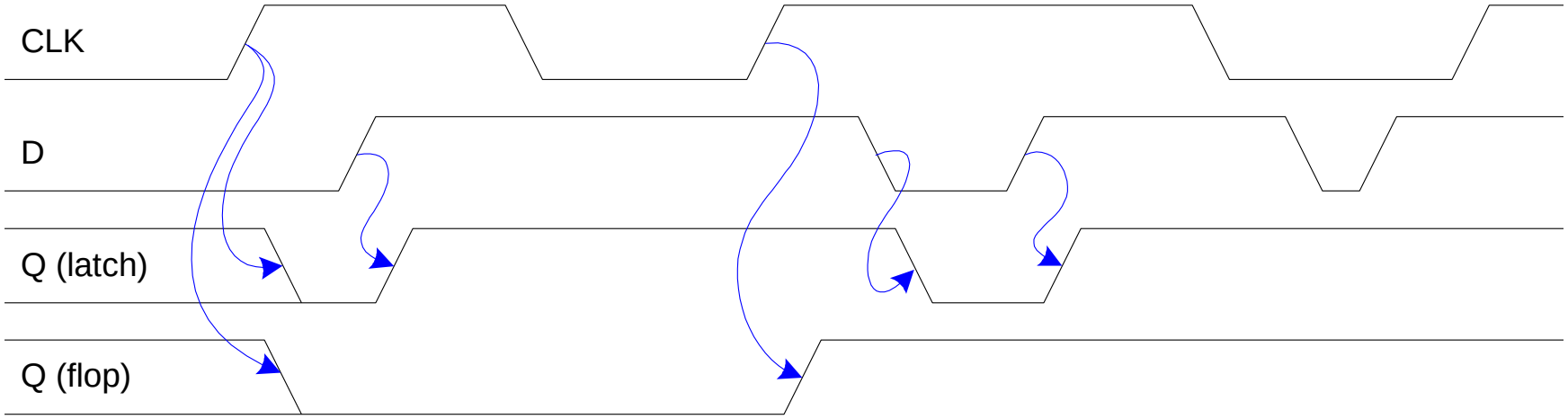


Q (latch)

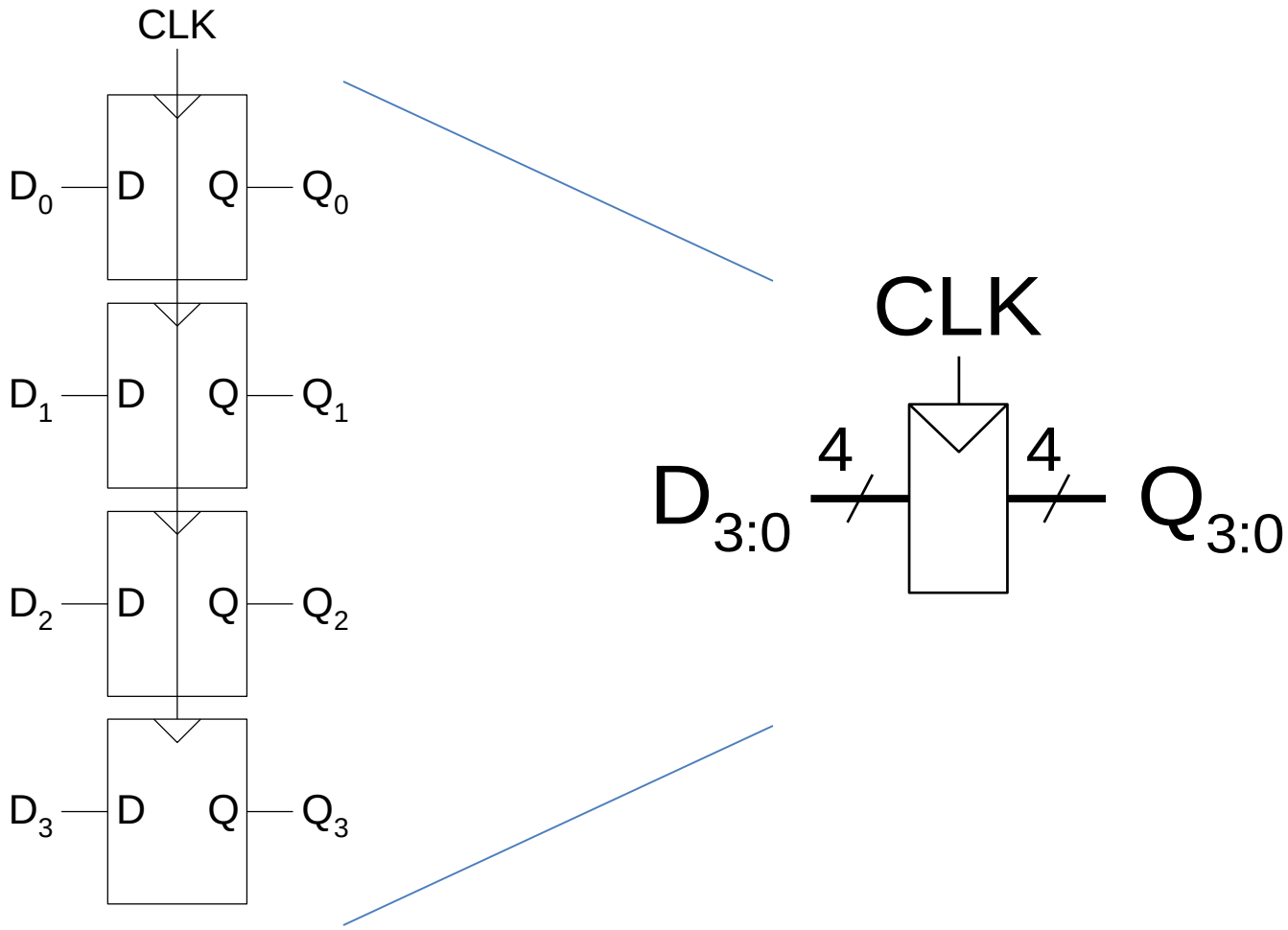
Q (flop)

D Latch vs. D Flip-Flop



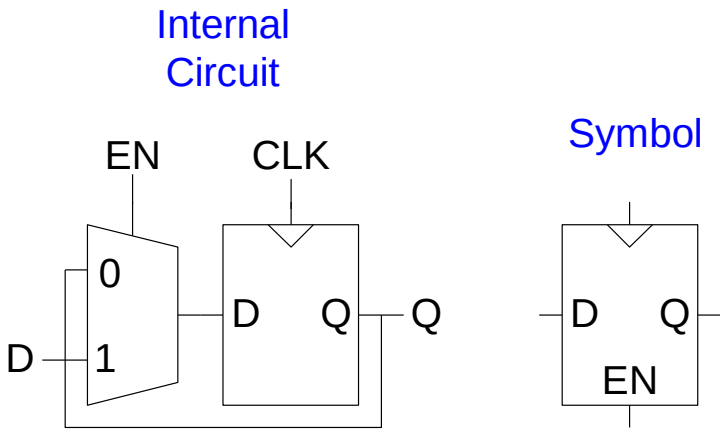


Registri: Multi-bit Flip-Flop



Flip-Flops “enabled”

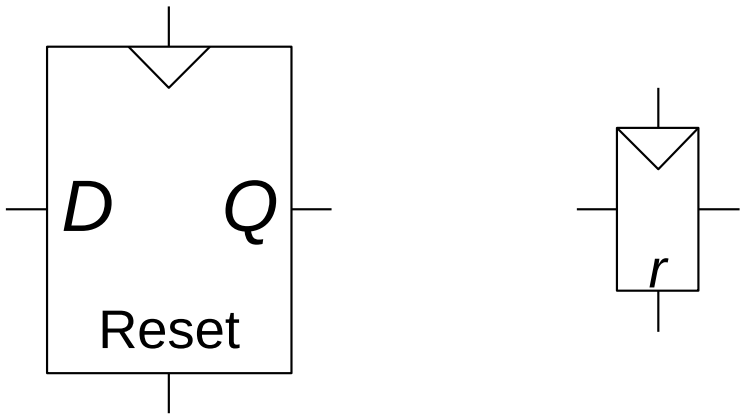
- *Inputs: CLK, D, EN*
- L’input enable (*EN*) stabilisce quando un nuovo valore di *D* è memorizzato
- ***EN* = 1:** *D* passa fino a *Q* (clock: 0→1)
- ***EN* = 0:** il flip-flop mantiene il suo stato precedente



Flip-Flops “resettabili”

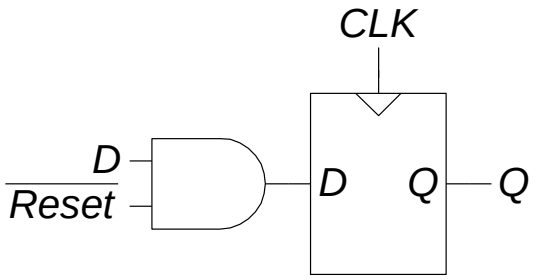
- *Inputs: CLK, D, Reset*
- ***Reset* = 1:** *Q* = 0
- ***Reset* = 0:** il flip-flop si comporta “normalmente” come un D flip-flop

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:

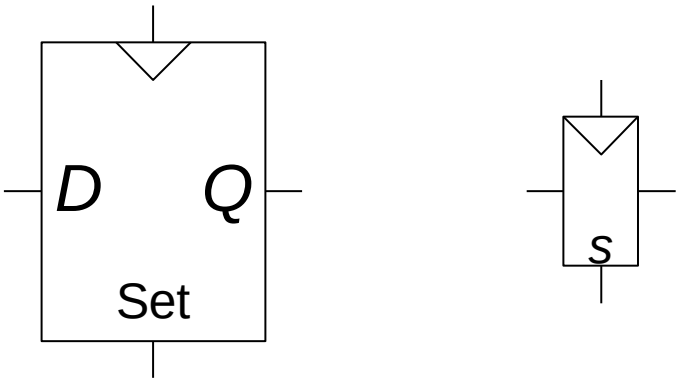


- Per i flip-flop asincroni occorre modificare il circuito interno del flip-flop

Flip-Flops “settabili”

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

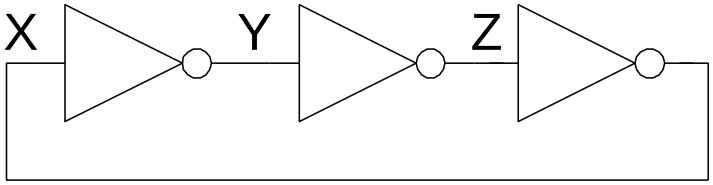
Symbols

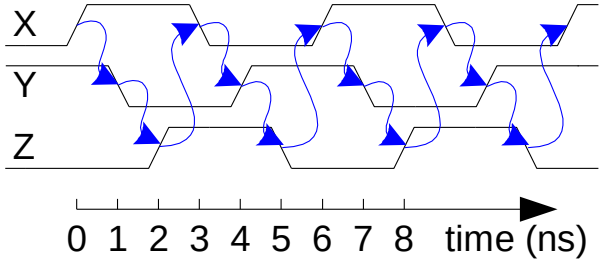


Esercizi

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

Criticità nella logica sequenziale





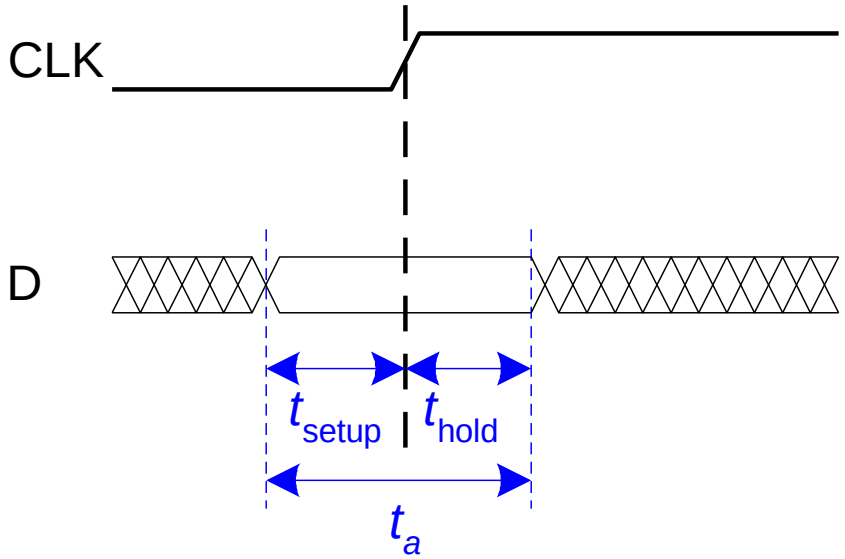
- Questi circuiti vengono detti “astabili” poiché hanno un comportamento oscillante
- Il periodo di oscillazione dipende dai ritardi degli inverter
- Idealmente è di 6 ns tuttavia può variare a causa di diversi fattori
  - differenze nella manifattura
  - temperatura
- Circuito *asincrono*: l’output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

Timing

- Flip-flop samples *D* at clock edge
- *D* must be stable when sampled
- Similar to a photograph, *D* must be stable around clock edge
- If not, metastability can occur

Input Timing Constraints

- **Setup time:** t_{setup} = time *before* clock edge data must be stable (i.e. not changing)
- **Hold time:** t_{hold} = time *after* clock edge data must be stable
- **Aperture time:** t_a = time *around* clock edge data must be stable ($t_a = t_{setup} + t_{hold}$)

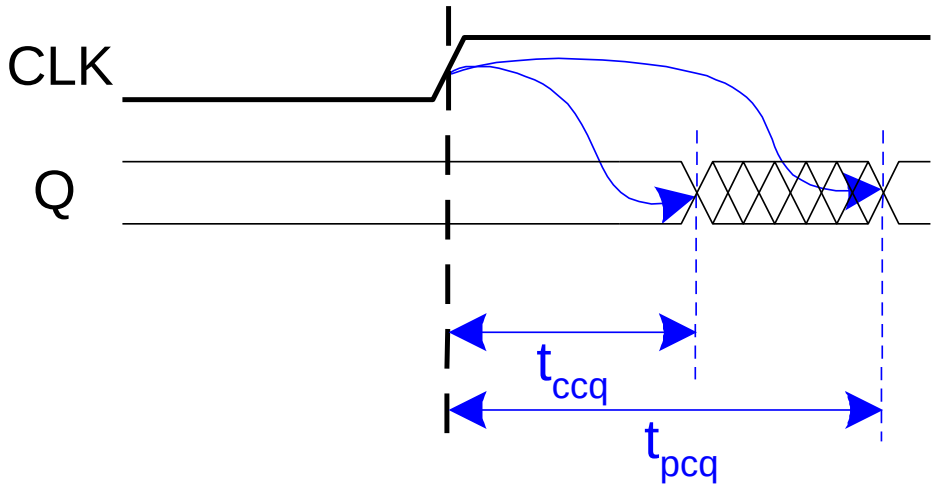


Dynamic Discipline

- Synchronous sequential circuit inputs must be stable during aperture (setup and hold) time around clock edge
- Specifically, inputs must be stable
  - at least t_{setup} before the clock edge
  - at least until t_{hold} after the clock edge

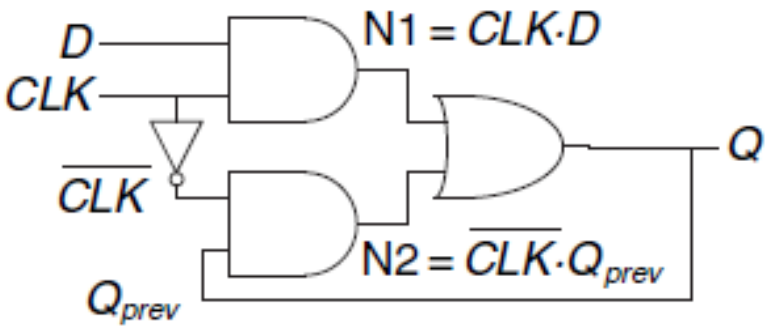
Output Timing Constraints

- **Propagation delay: t_{pcq}** = time after clock edge that the output *Q* is guaranteed to be stable (i.e., to stop changing)
- **Contamination delay: t_{ccq}** = time after clock edge that *Q* might be unstable (i.e., start changing)



La tabella delle transizioni

- I circuiti sequenziali sono descritti attraverso una tabella delle transizioni (diversa dalla tabella di verità !)
- le uscite della rete sono una funzione combinatoria degli ingressi (D, CLK) e delle uscite dei latch

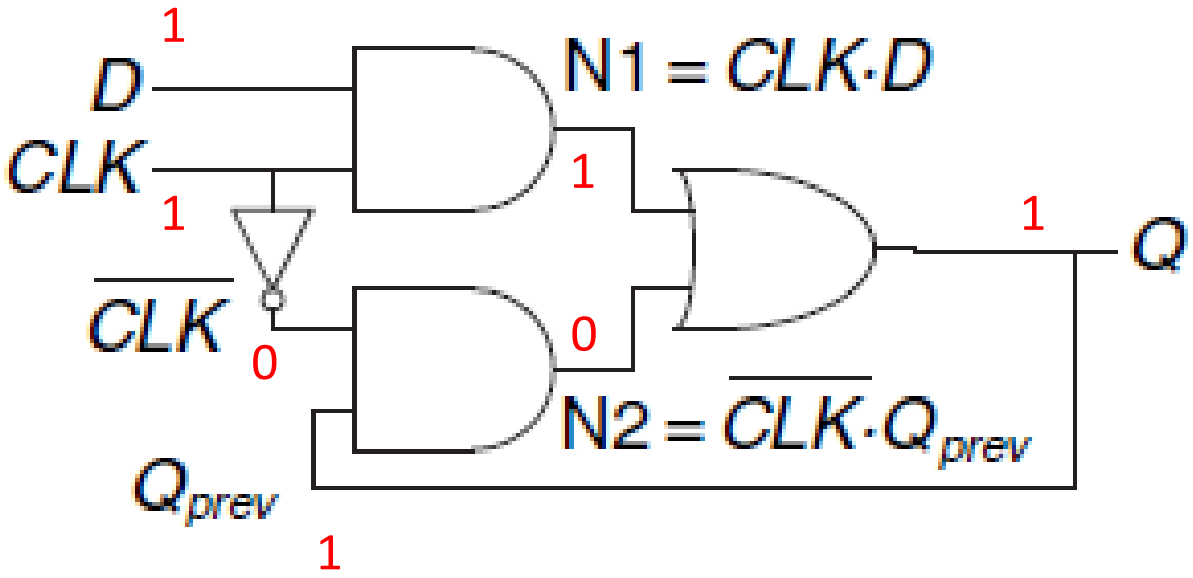


$$Q = CLK \cdot D + \overline{CLK} \cdot Q_{prev}$$

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

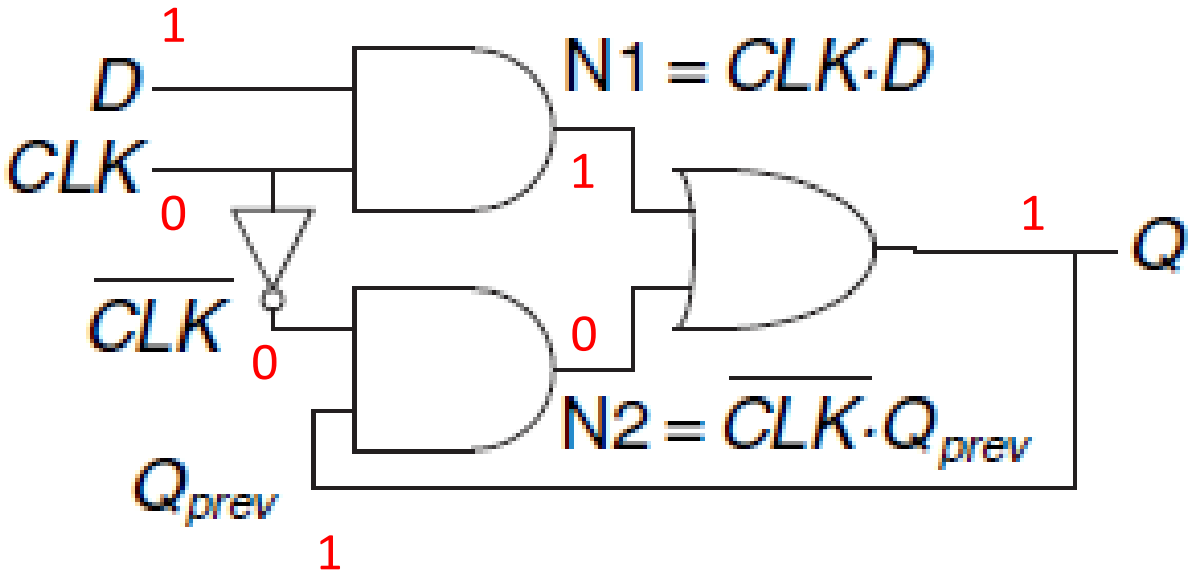
Criticità nella logica sequenziale

- D=1, CLK=1 ➡ Q=1



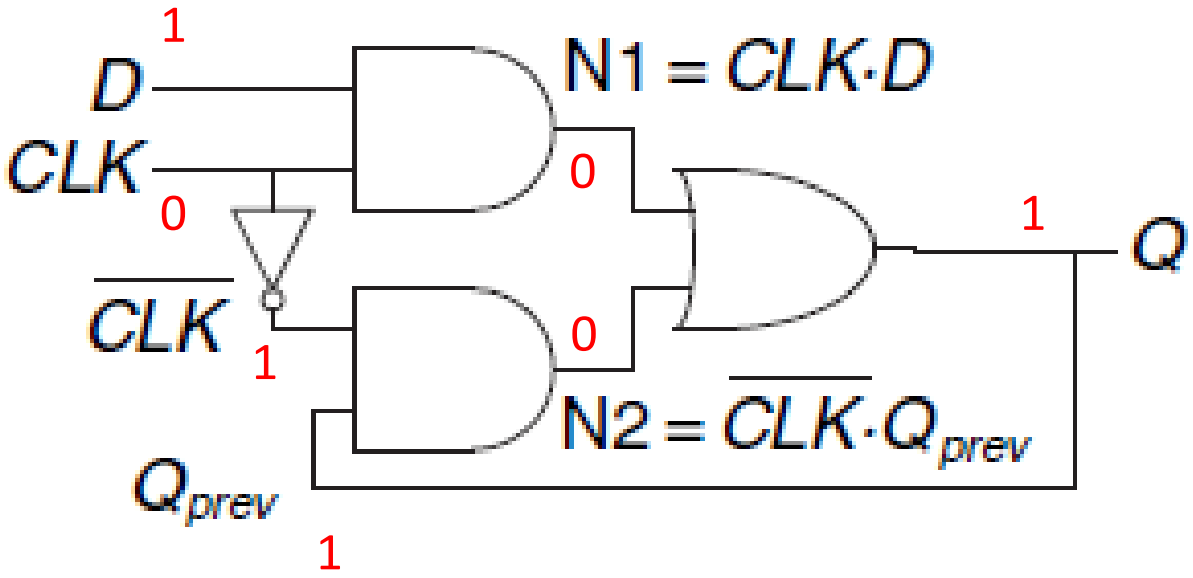
Criticità nella logica sequenziale

t_0 CLK 1→0



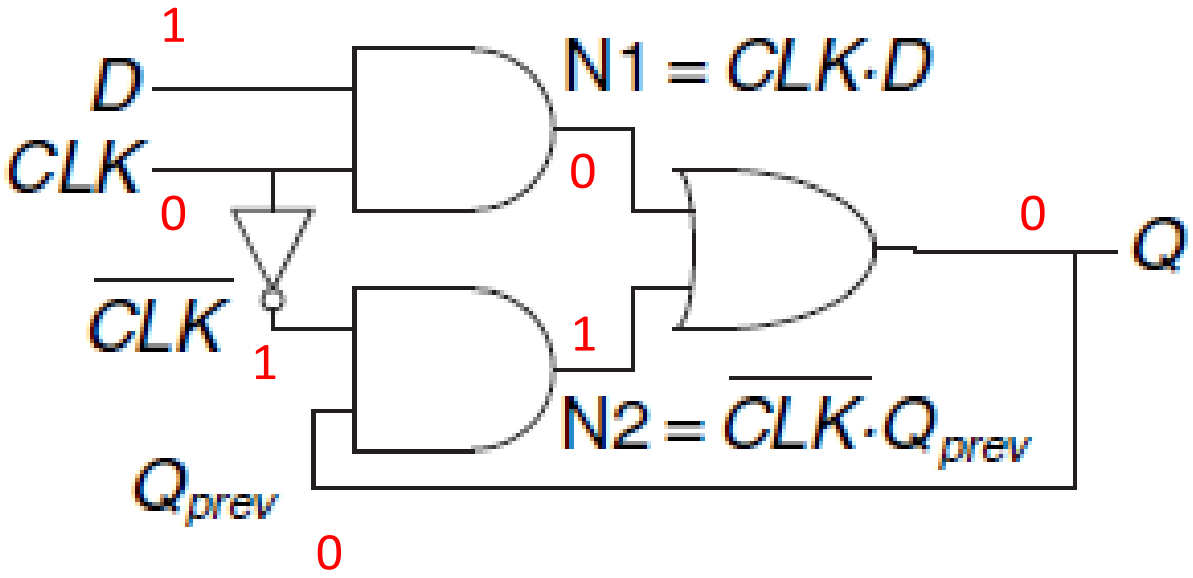
Criticità nella logica sequenziale

t_1



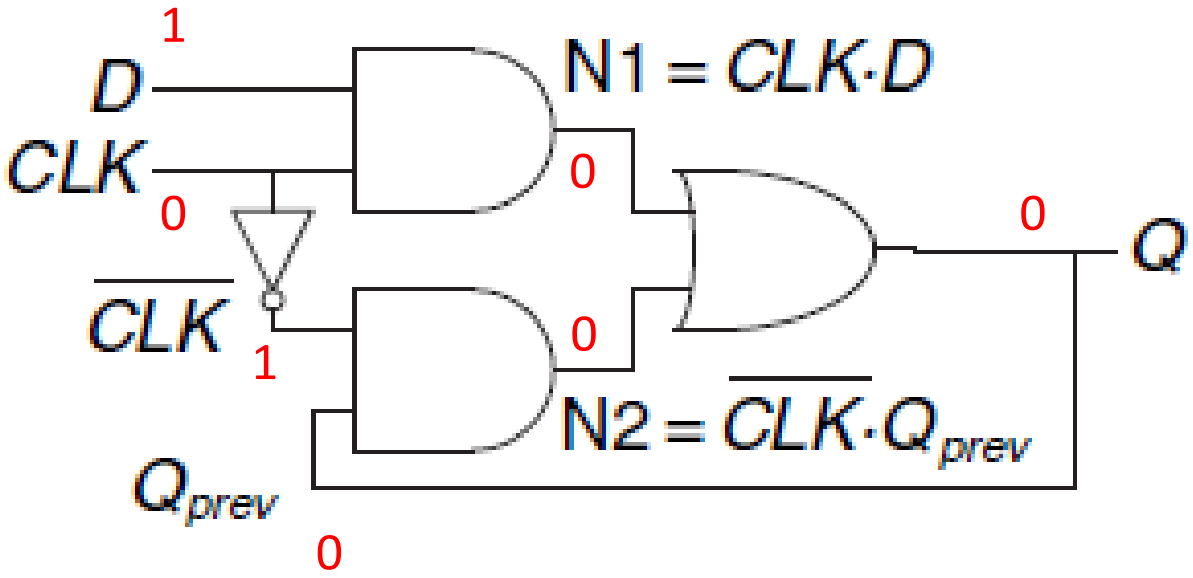
Criticità nella logica sequenziale

t_2



Criticità nella logica sequenziale

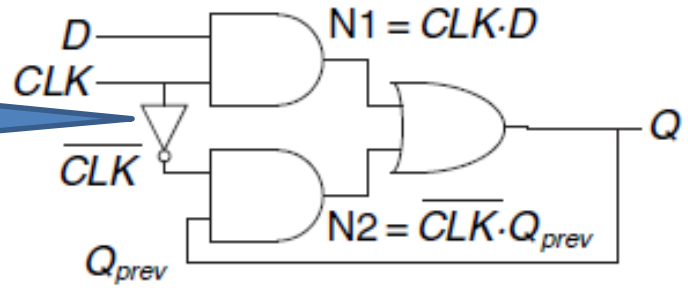
t_3



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=0 !!!

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
- Il registro disciplina la propagazione dei dati e consente al sistema di essere sincronizzato col clock: circuito *sincrono*

Logiche sequenziali sincrone

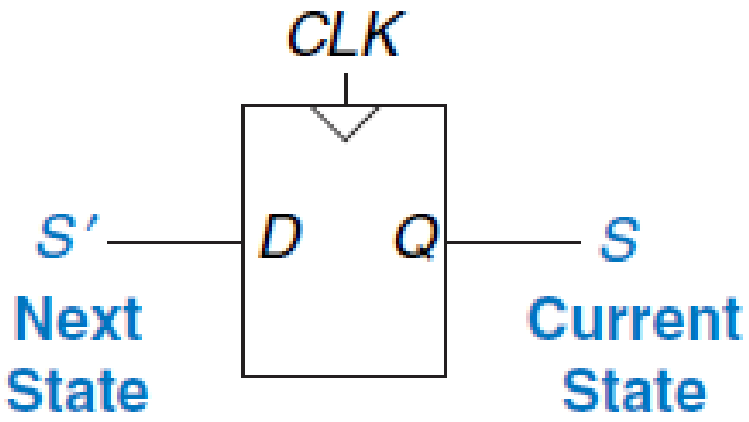
- In generale un circuito sequenziale sincrono ha un insieme finito di stati $\{S_0,\ldots, S_{k-1}\}$
- Logica combinatoria: $out=f(in)$
- Logica sequenziale sincrona: $Out = f(in,s_c)$ $s_n = g(in,s_c)$

Design di logiche sequenziali sincrone

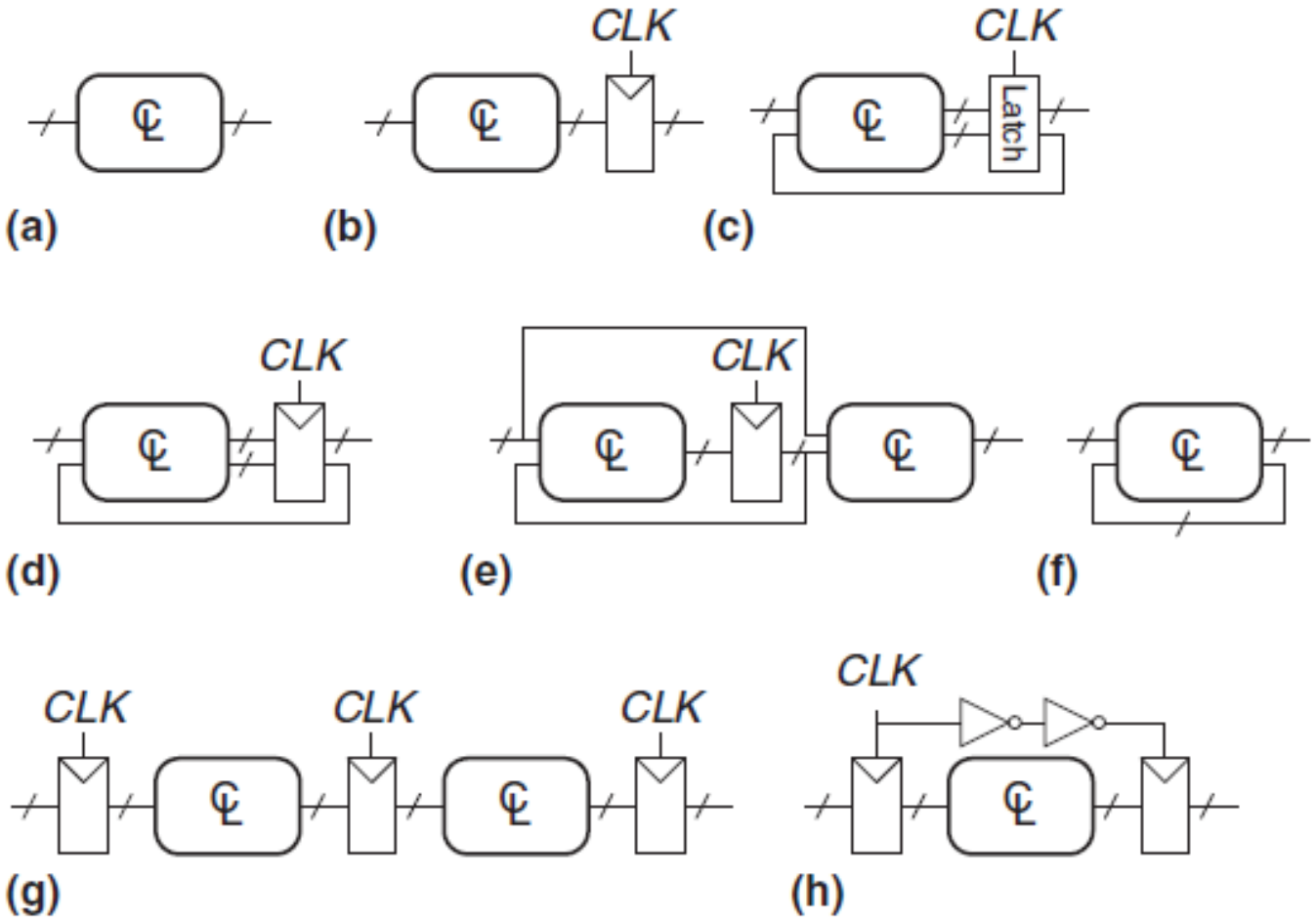
- Inserire registri nella retroazione
- I registri determinano lo **stato** $S_0,\ldots, S_{k-1}$ del sistema
- I cambiamenti di stato sono determinati dalle transizioni del clock: il sistema è sincronizzato con il clock
- *Regole* di composizione:
  - Ogni componente è un registro o un circuito combinatorio
  - Almeno un componente è un registro
  - Tutti i registri sono sincronizzati con un unico clock
  - Ogni loop 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



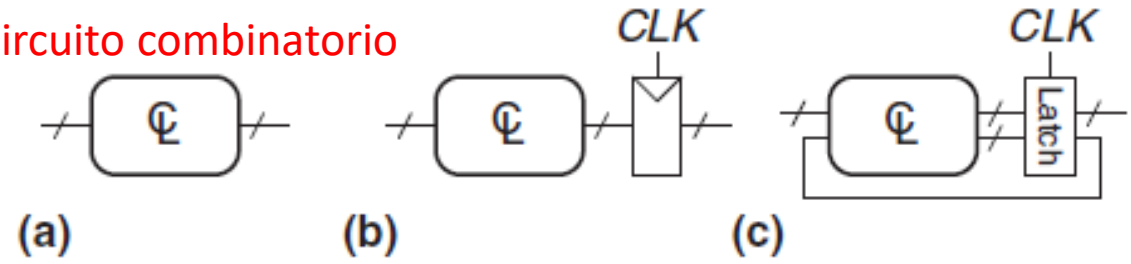
Esempi



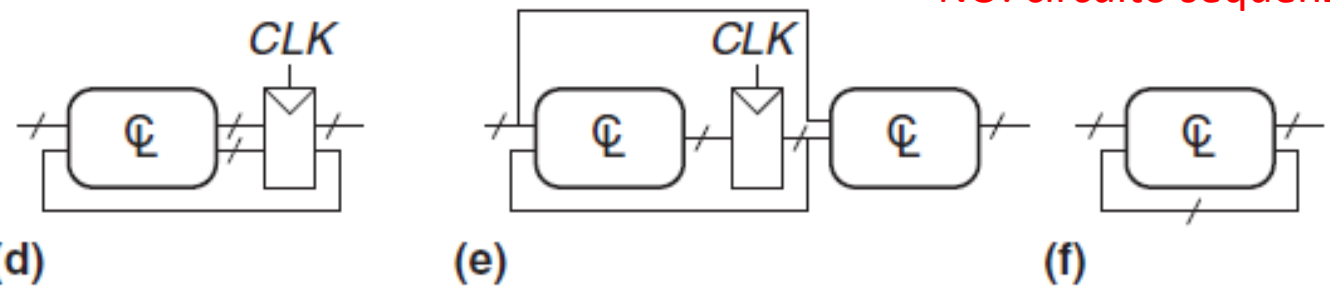
Esempi

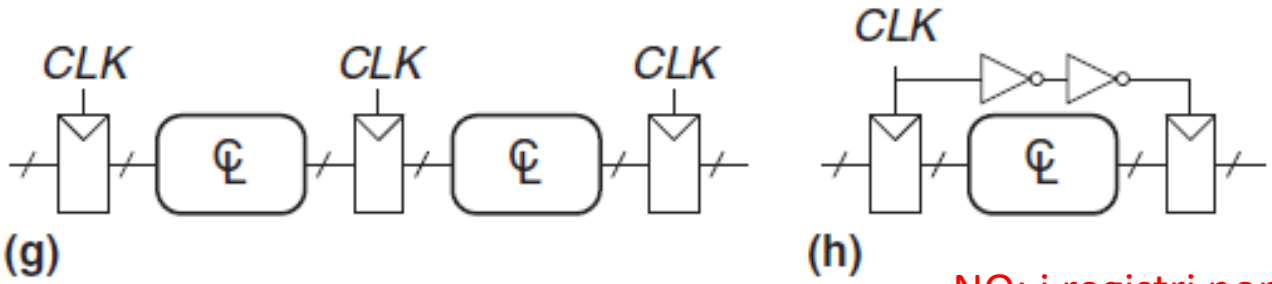
NO: circuito combinatorio

NO: latch e non flip-flop



NO: circuito sequenziale asincrono

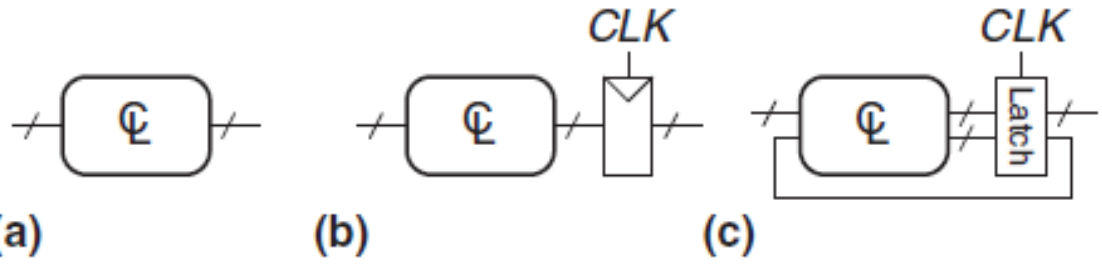




NO: i registri non hanno lo stesso clock

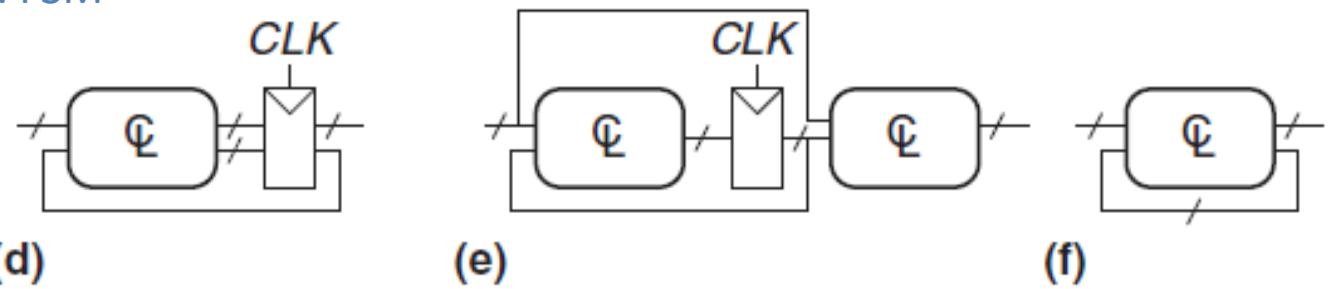
Esempi

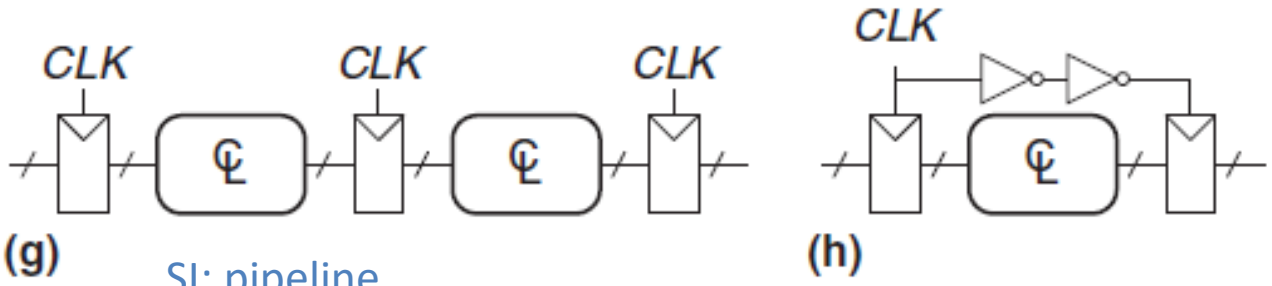
SI: ma senza feedback



SI: FSM

SI: FSM

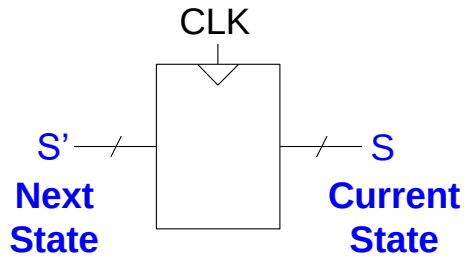


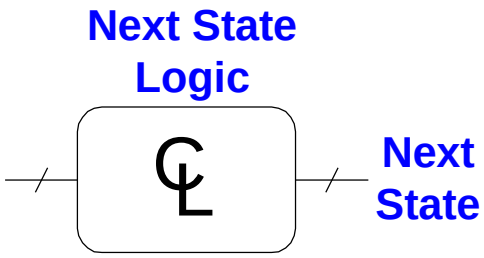


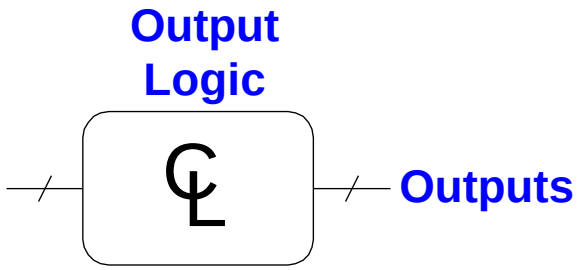
SI: pipeline

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)



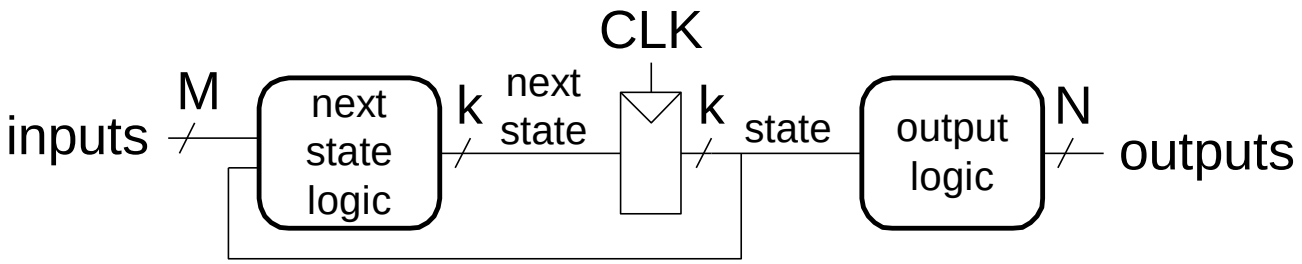




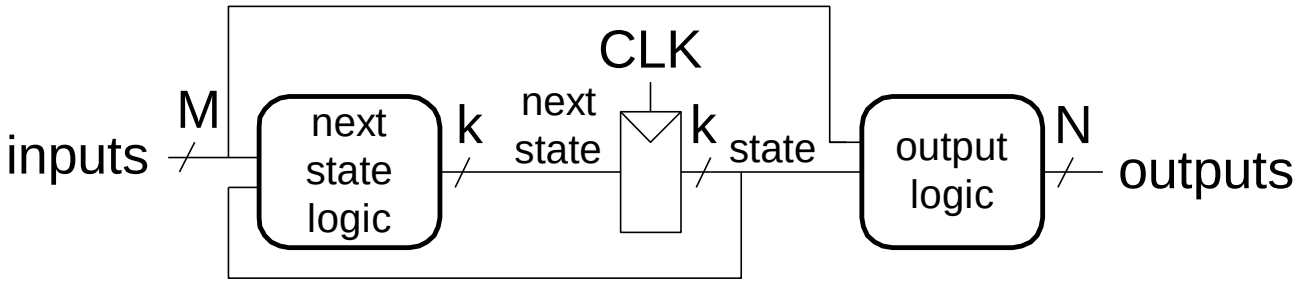
Finite State Machines

- s_n dipende sia dall'input che da s_n $s_n = g(in, s_c)$
- 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

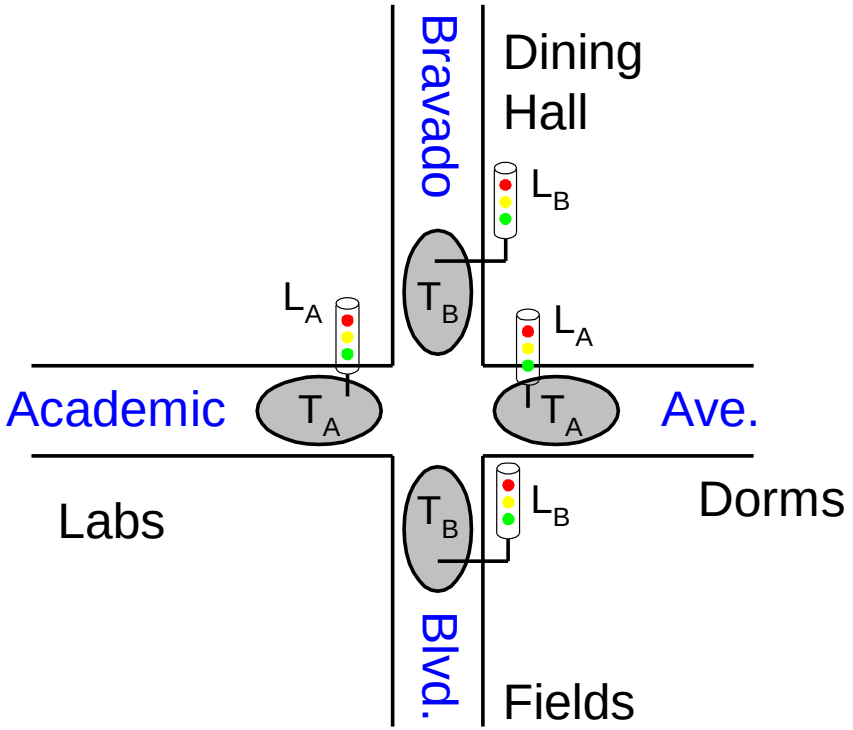


Mealy FSM



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

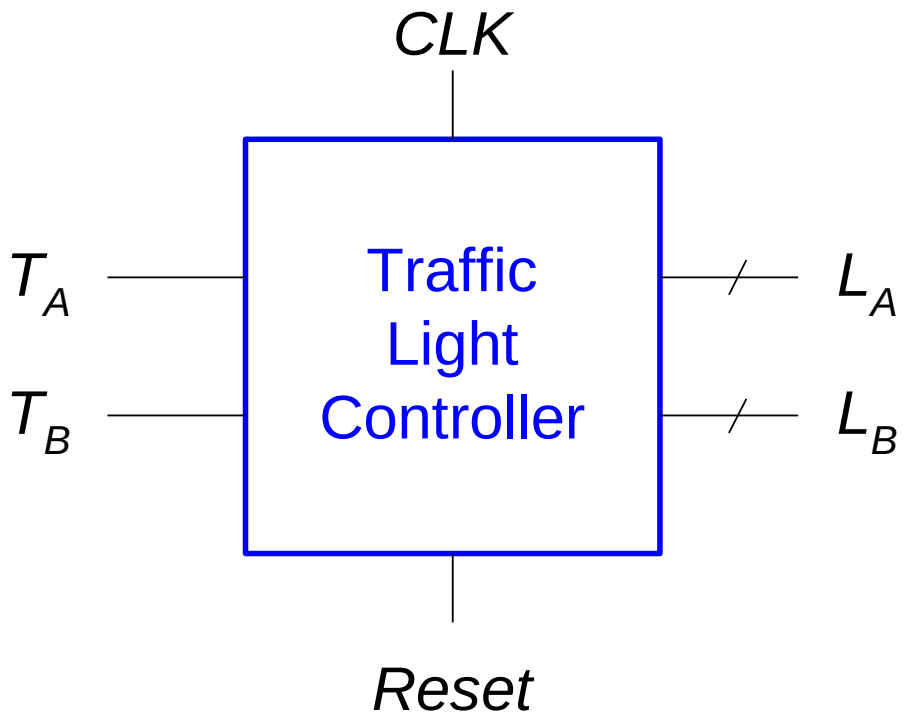


Diagramma di transizione: Moore FSM

- **Stati:** etichettate con gli outputs
- **Transizioni:** etichettate con gli inputs

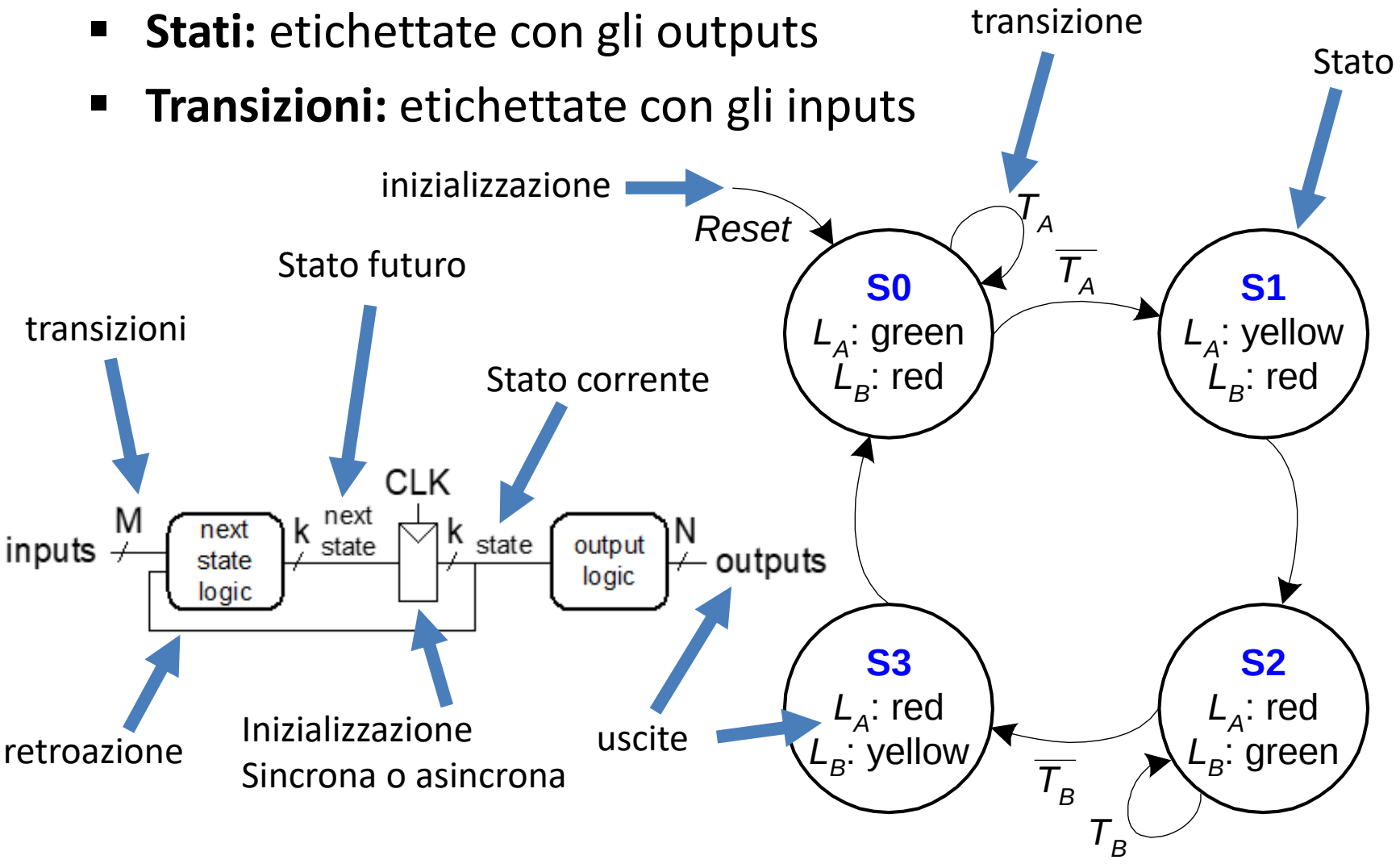
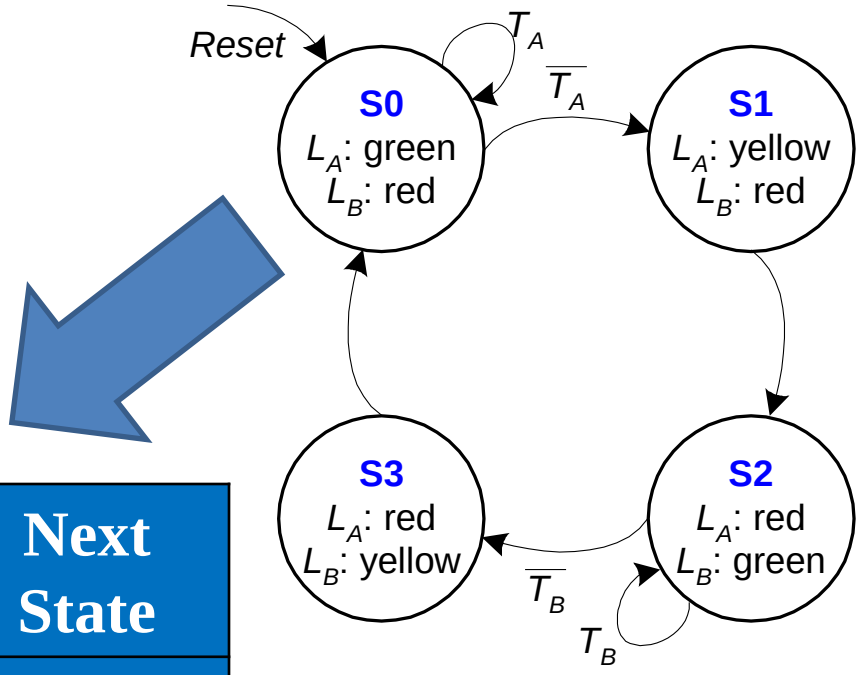


Tabella di stato

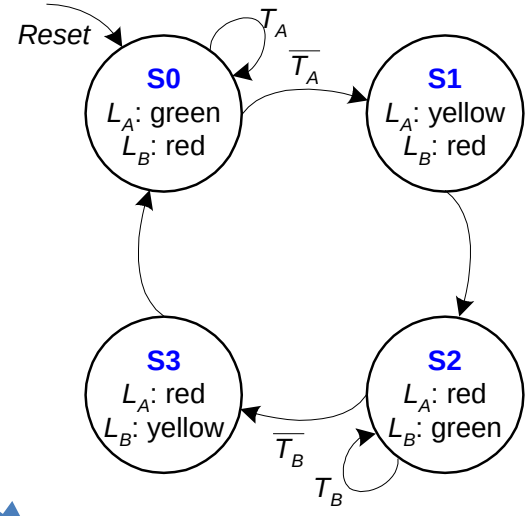


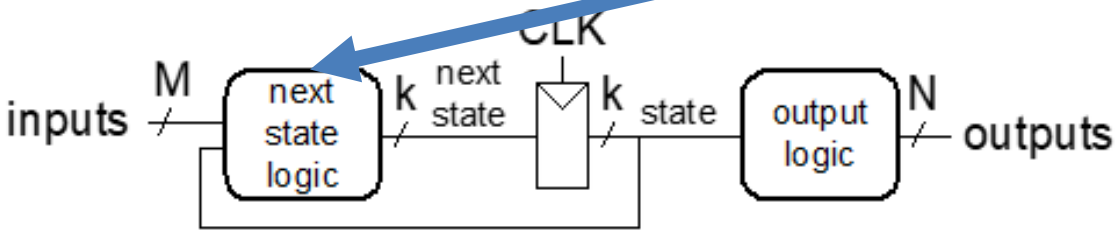
Current State	Inputs		Next State
S	T_A	T_B	S'
S0	0	X	S1
S0	1	X	S0
S1	X	X	S2
S2	X	0	S3
S2	X	1	S2
S3	X	X	S0

Tabella di transizione

State	Encoding
S0	00
S1	01
S2	10
S3	11

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

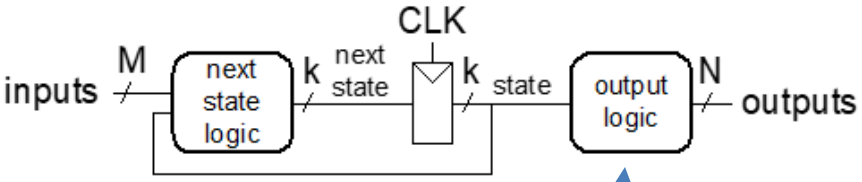




$$S'_1 = S_1 \oplus S_0$$

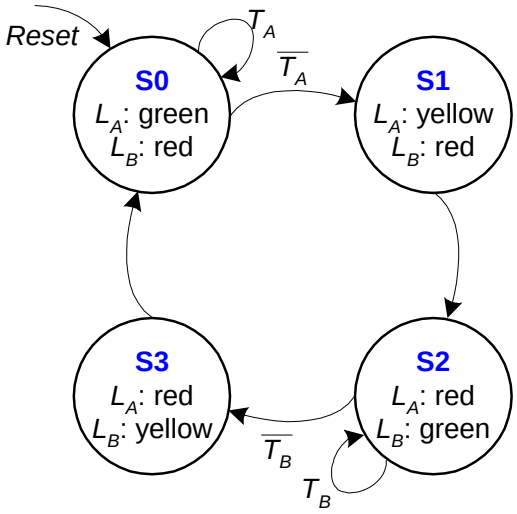
$$S'_0 = \overline{S_1}\,\overline{S_0}\,\overline{T_A} + S_1\overline{S_0}\,\overline{T_B}$$

Tabella dell'output



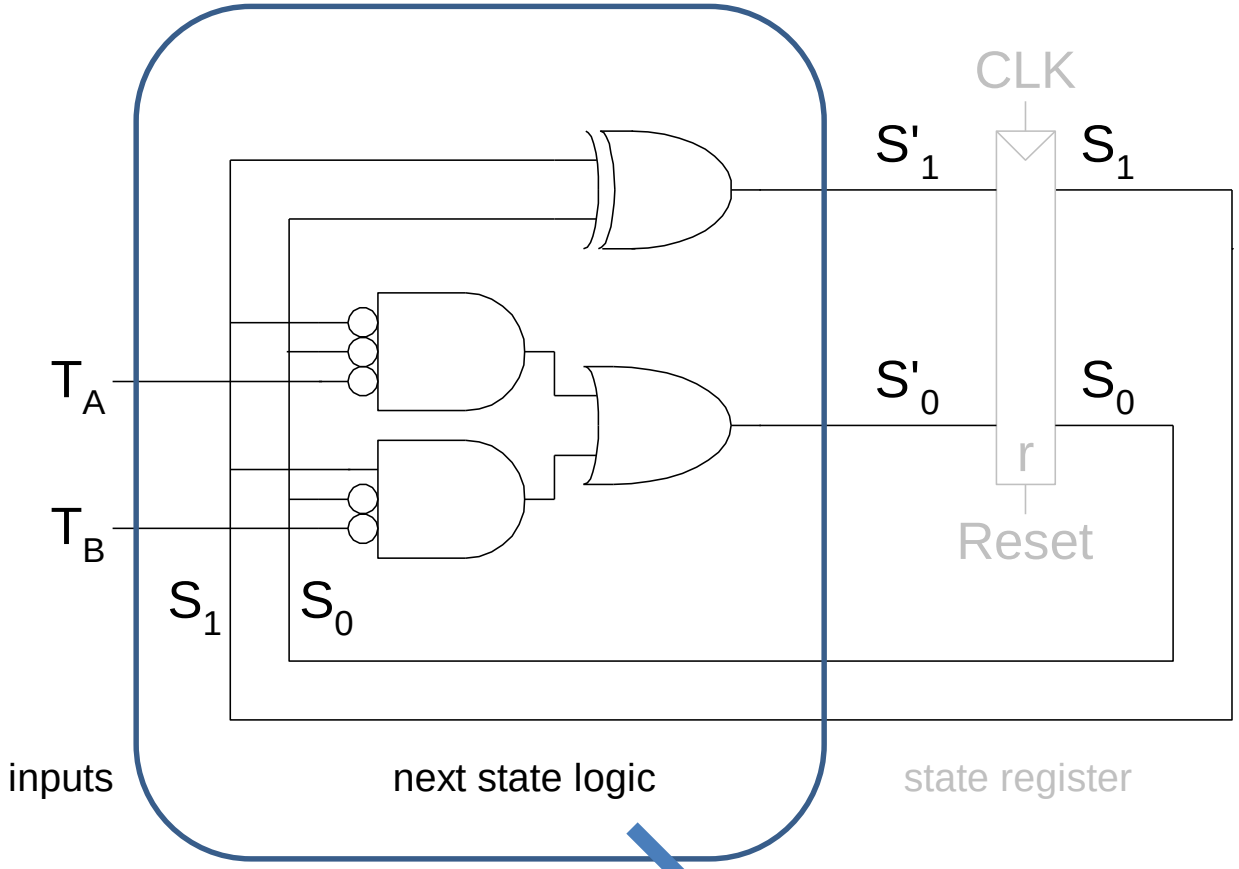
Current State		Outputs			
S_1	S_0	L_{A1}	L_{A0}	L_{B1}	L_{B0}
0	0	0	0	1	0
0	1	0	1	1	0
1	0	1	0	0	0
1	1	1	0	0	1

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



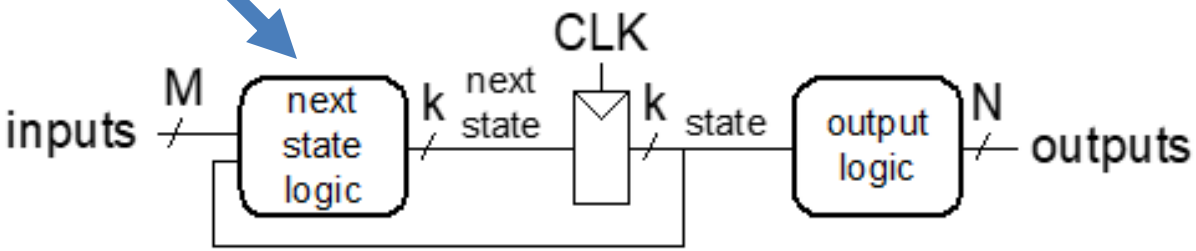
Output	Encoding
green	00
yellow	01
red	10

Schema della logica di prossimo stato

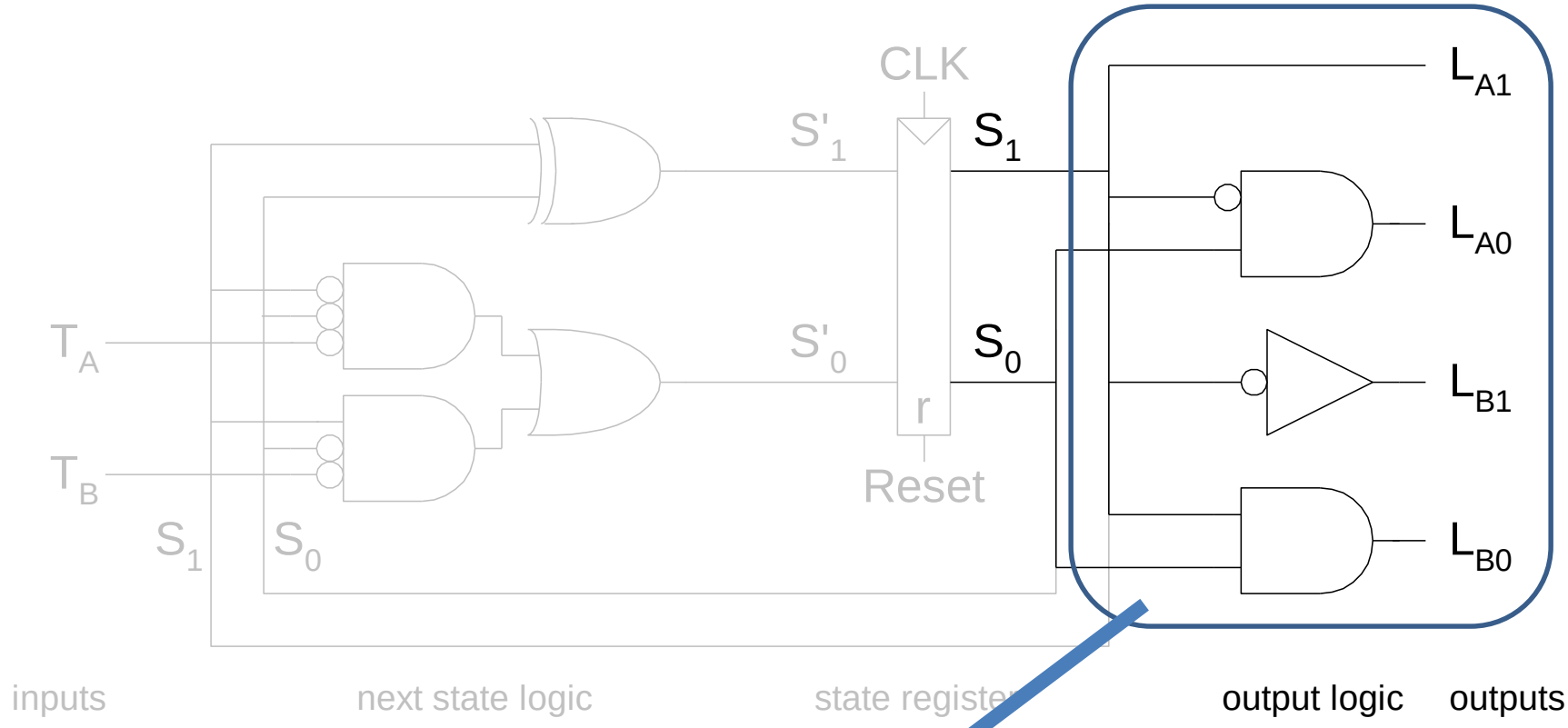


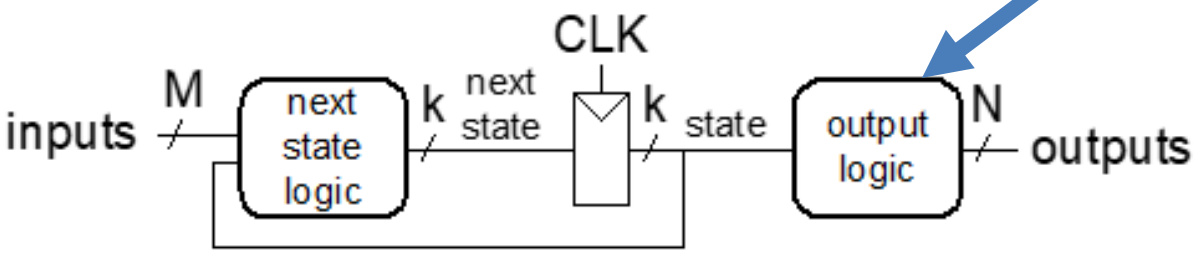
$$S'_1 = S_1 \oplus S_0$$

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



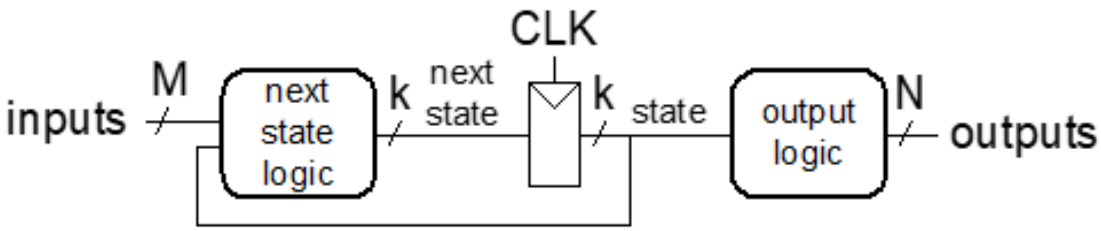
Schema della logica di output

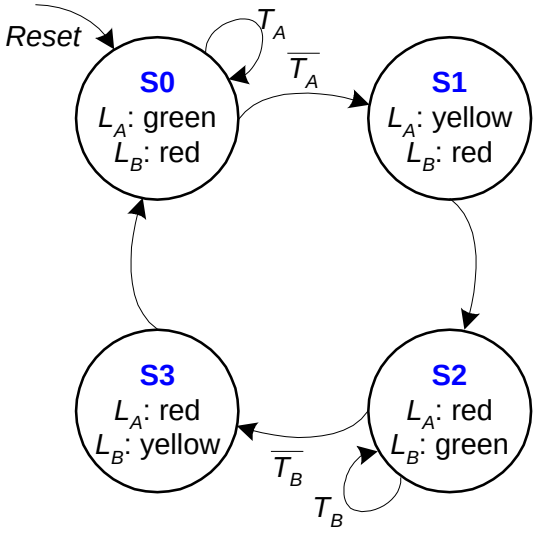


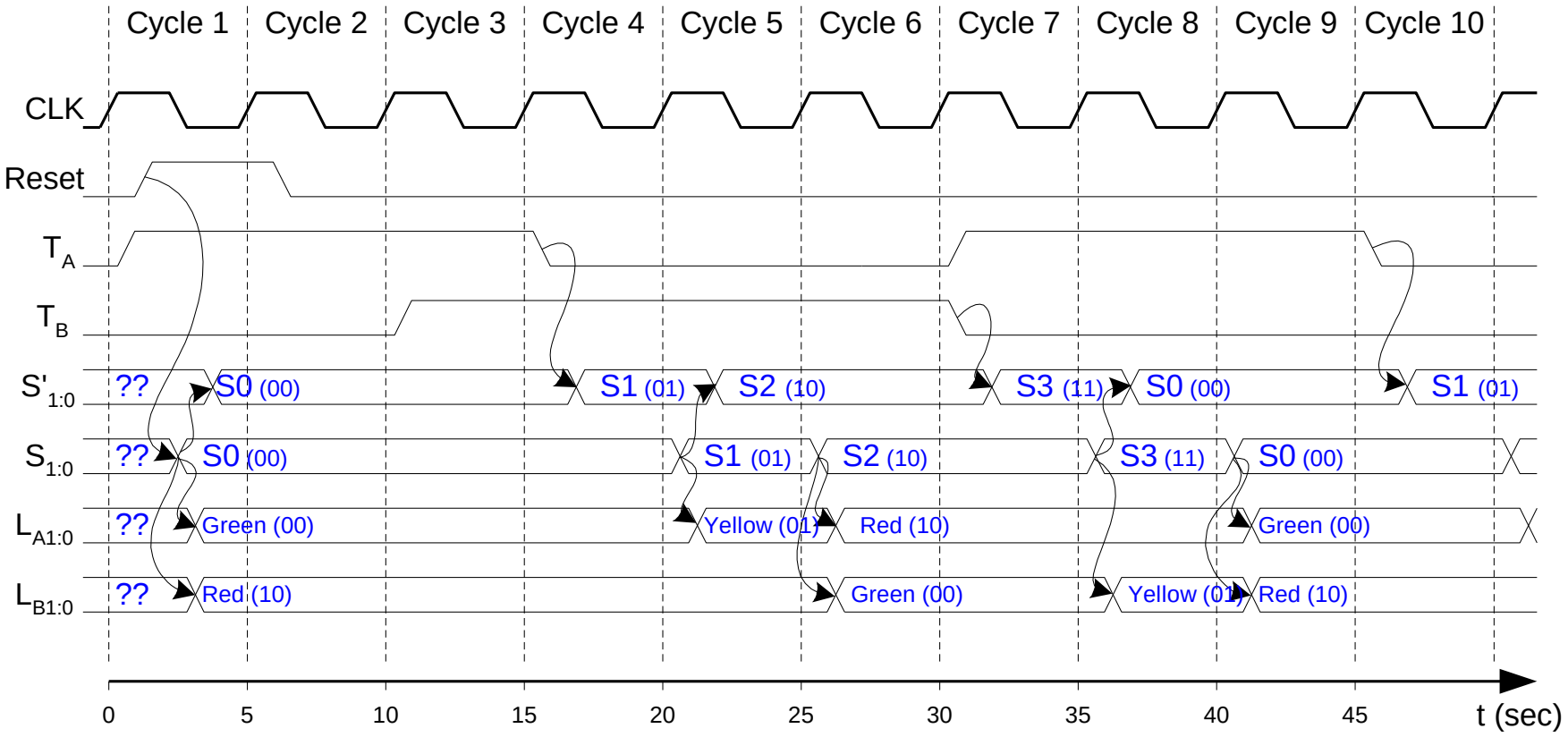


$$L_{A1} = S_1$$
$$L_{A0} = S_1 S_0$$
$$L_{B1} = S_1$$
$$L_{B0} = S_1 S_0$$

FSM Timing Diagram





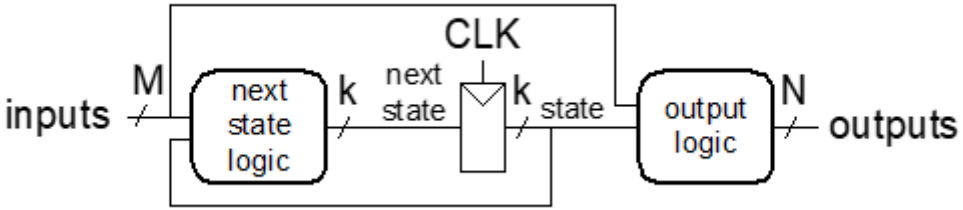


Encoding degli stati

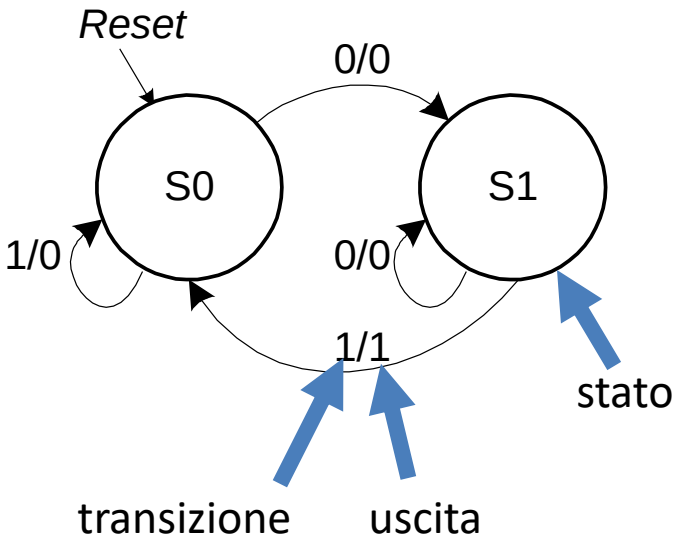
- Encoding binario:
  - i.e., per 4 stati, 00, 01, 10, 11
- Encoding *one-hot*
  - Un bit per stato
  - Solo un bit HIGH alla volta
  - i.e., per 4 stati, 0001, 0010, 0100, 1000
  - Richiede più flip-flops
  - Spesso la logica combinatoria associata è più semplice

Moore vs Mealy FSM

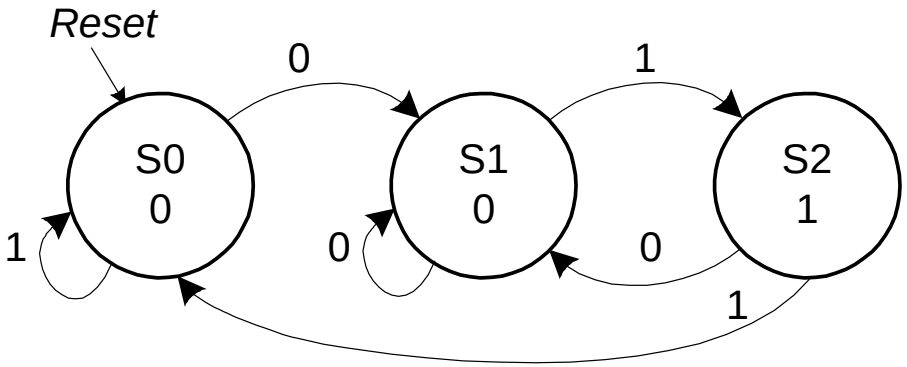
Alyssa P. Hacker has a snail that crawls down a paper tape with 1's and 0's on it. The snail smiles whenever the last two digits it has crawled over are 01. Design Moore and Mealy FSMs of the snail's brain.



Mealy FSM



Moore FSM



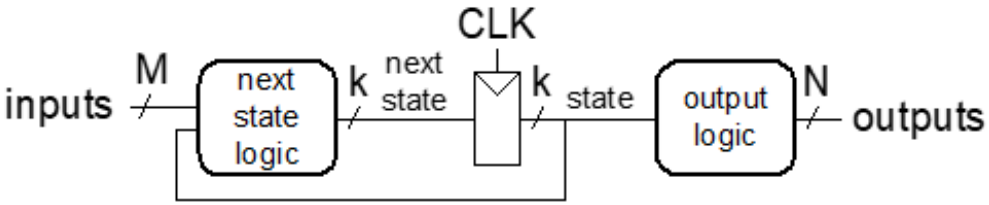
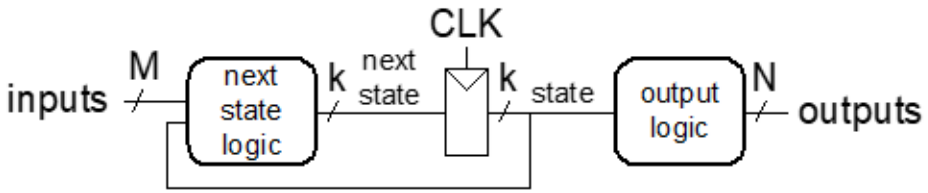
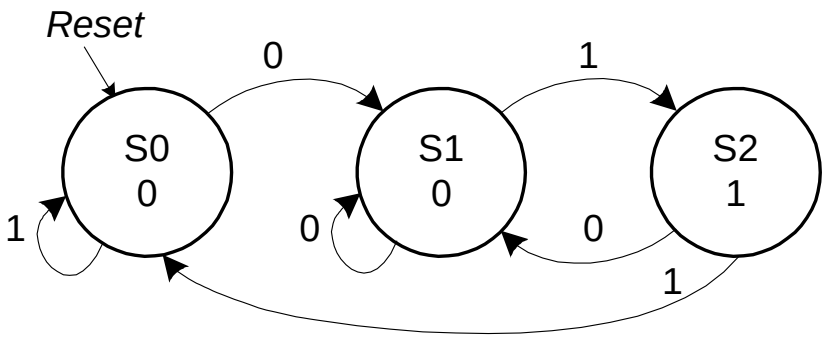


Tabella transizione Moore FSM



Moore FSM



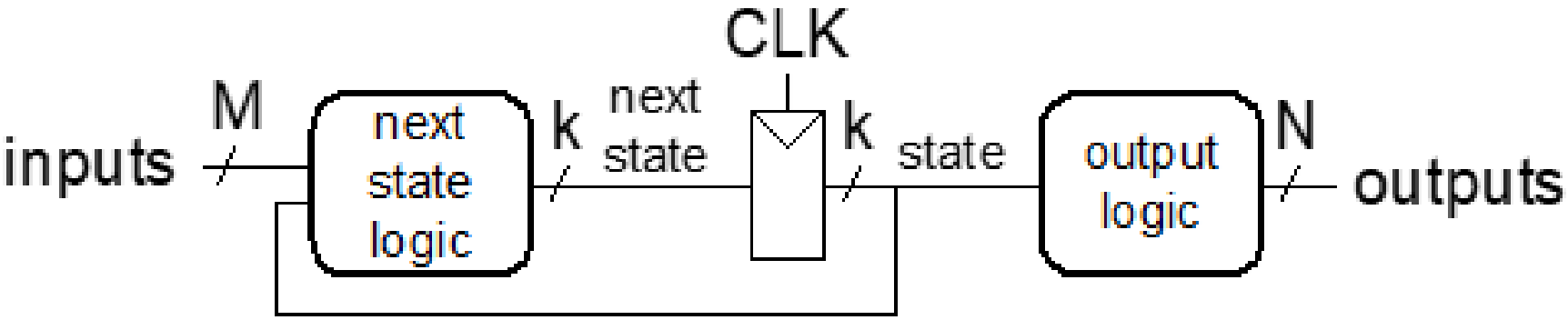
Current State		Inputs	Next State	
S_1	S_0	A	S'_1	S'_0
0	0	0	0	1
0	0	1	0	0
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0

State	Encoding
S0	00
S1	01
S2	10

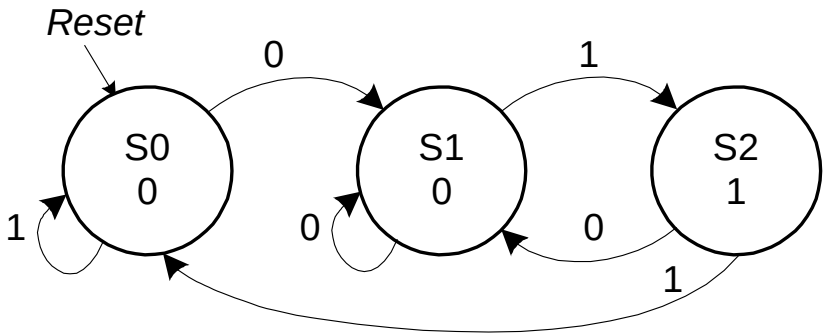
$$S_1' = S_0 A$$

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

Tabella output Moore FSM



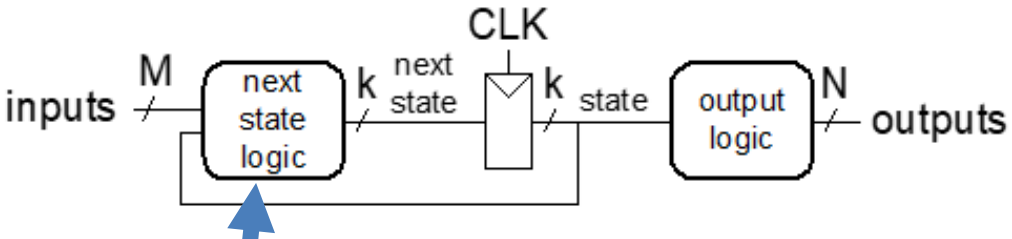
Moore FSM



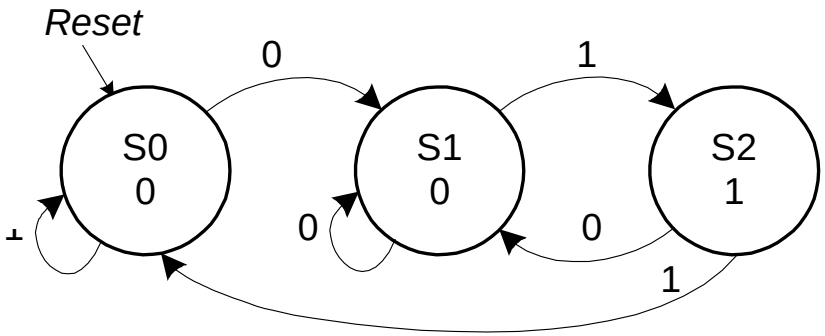
Current State		Output
S_1	S_0	Y
0	0	0
0	1	0
1	0	1

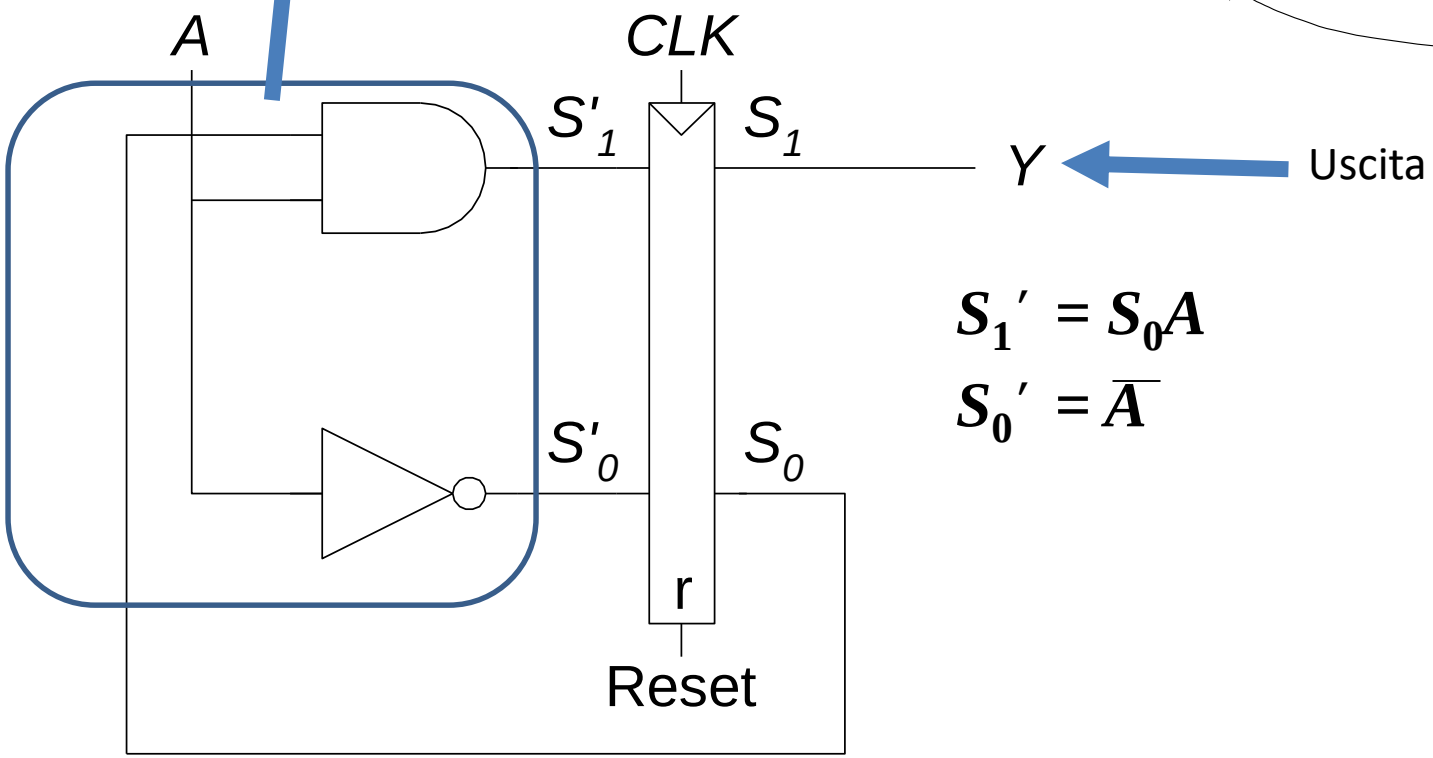
$$Y = S_1$$

Schema logica di prossimo stato e uscite (Moore)



Moore FSM

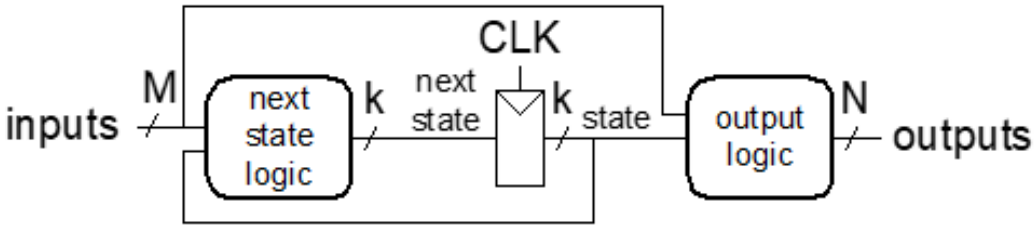




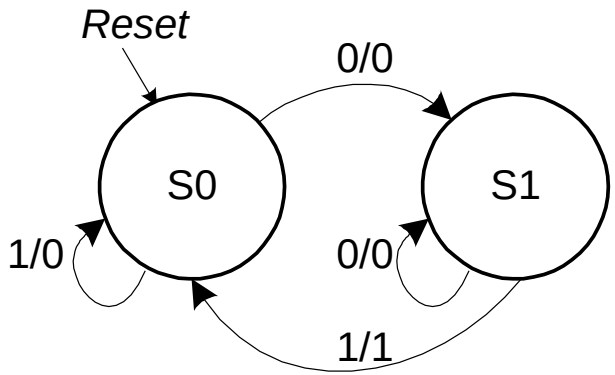
$$S_1' = S_0 A$$

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

Tabella transizione/output Mealy FSM



Mealy FSM



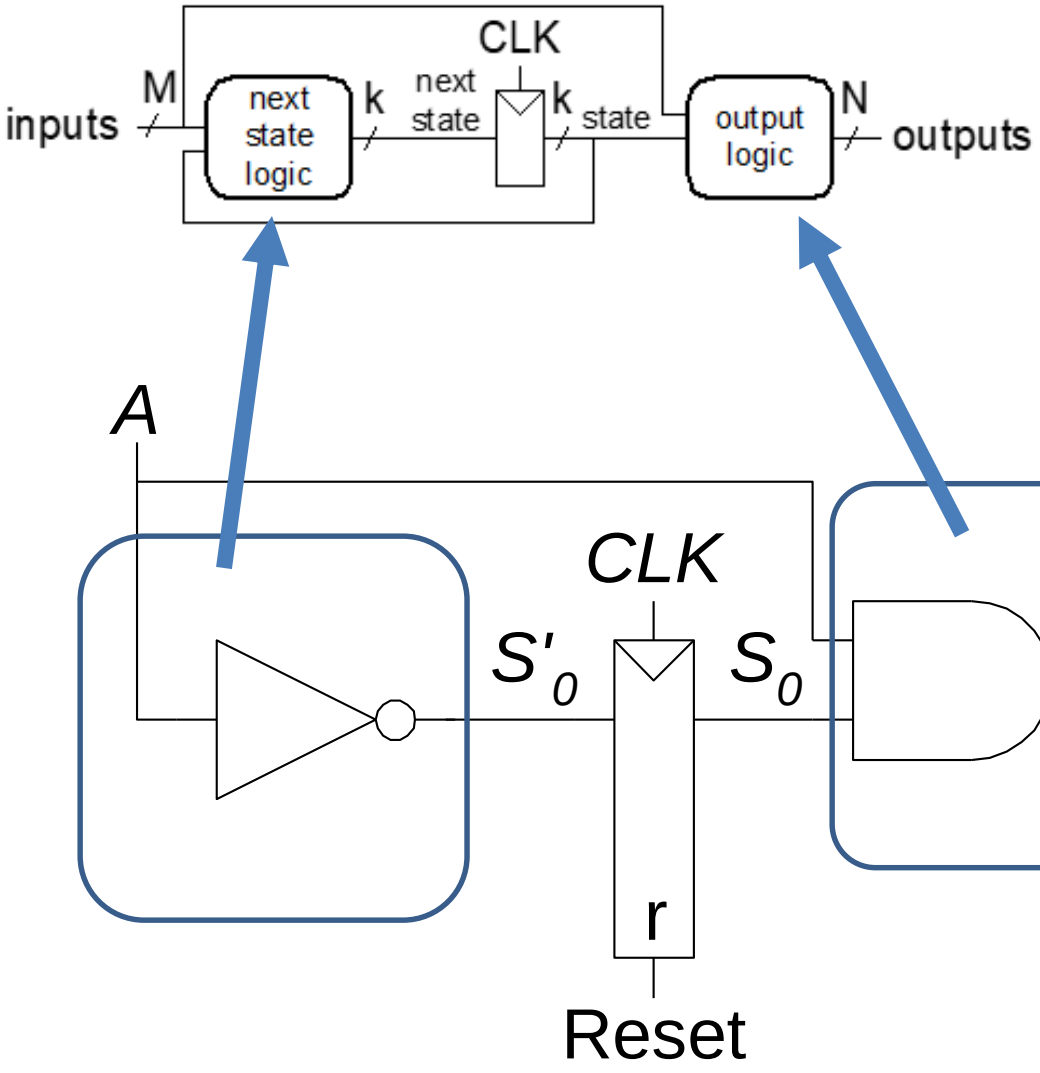
Current State	Input	Next State	Output
S	A	S'	Y
0	0	1	0
0	1	0	0
1	0	1	0
1	1	0	1

State	Encoding
S0	0
S1	1

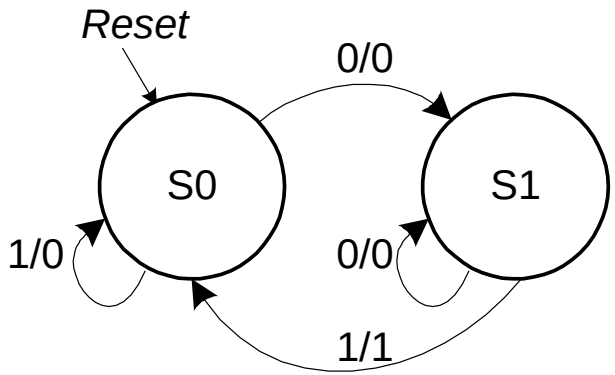
$$S' = \overline{A}$$

$$Y = S\,A$$

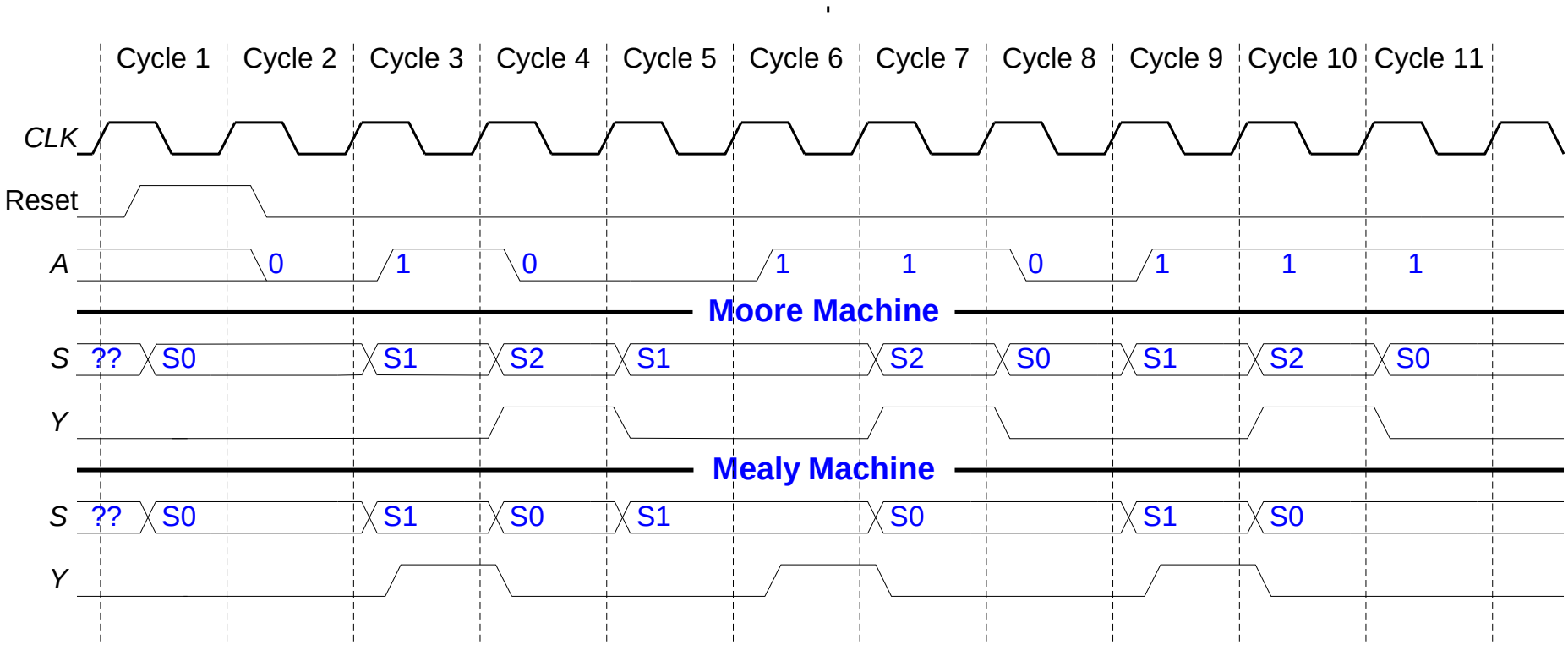
Schema Mealy FSM



Mealy FSM



Moore & Mealy Timing



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

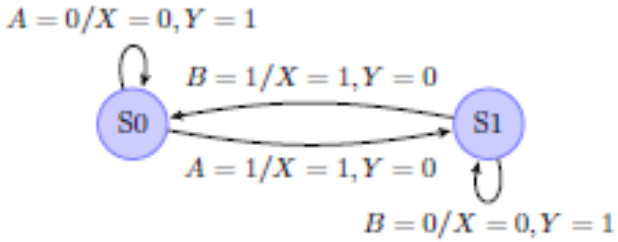
- Progettare una Mealy FSM F con due input (A e B) e un output Q.
  - Q=1 sse A e B assumono rispettivamente il valore precedente
  - es:

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: _____

Slide aggiuntive
utili per approfondimenti

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

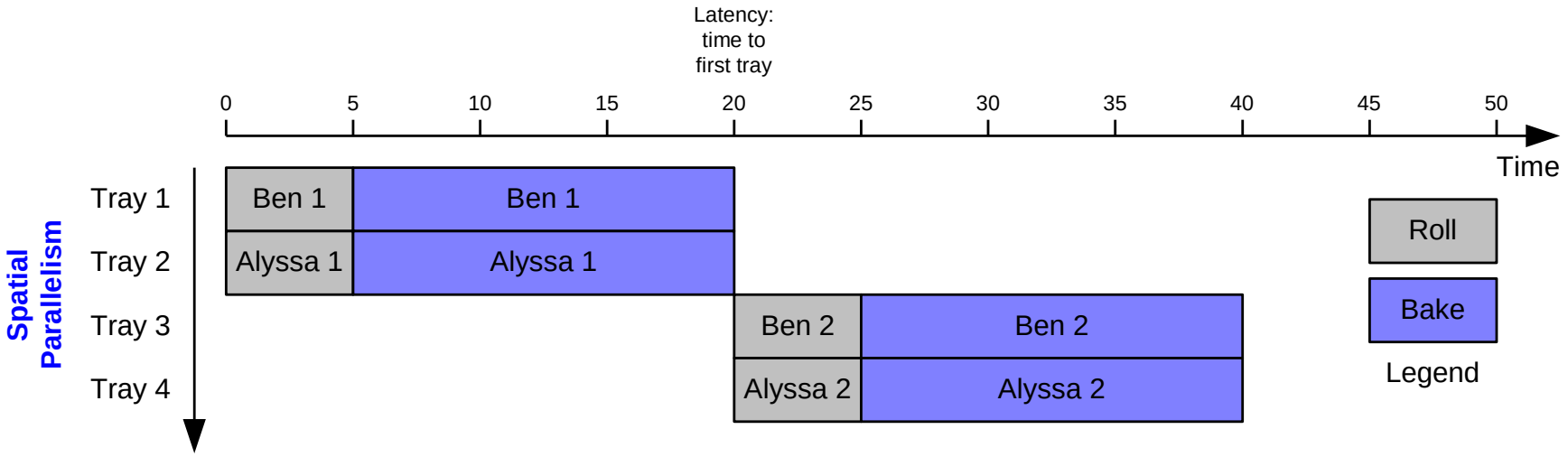
Latency = 5 + 15 = 20 minuti = **1/3 h**

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.

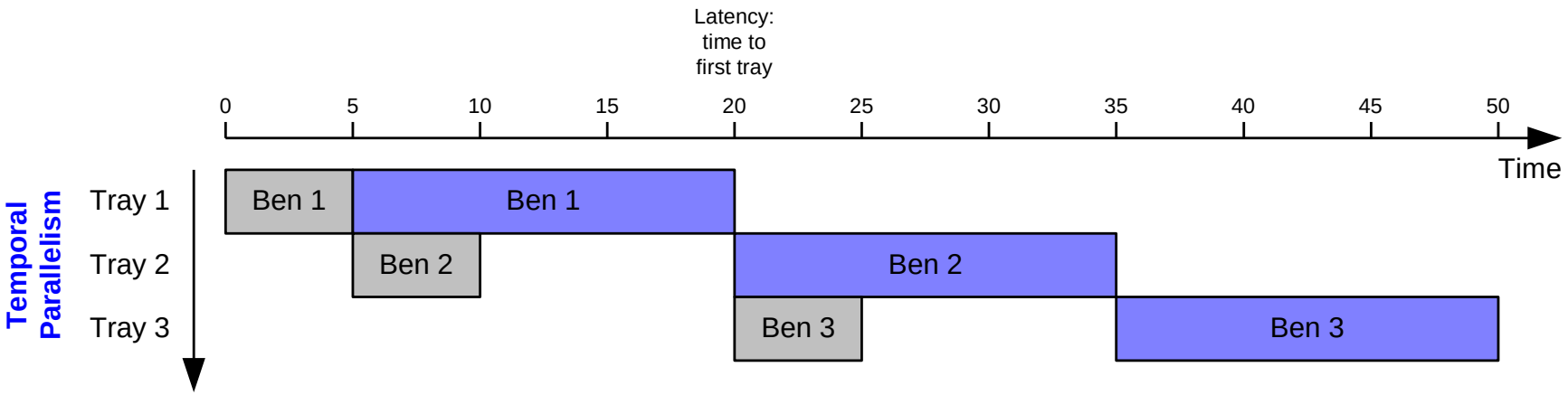
Parallelismo spaziale



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

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

Parallelismo temporale

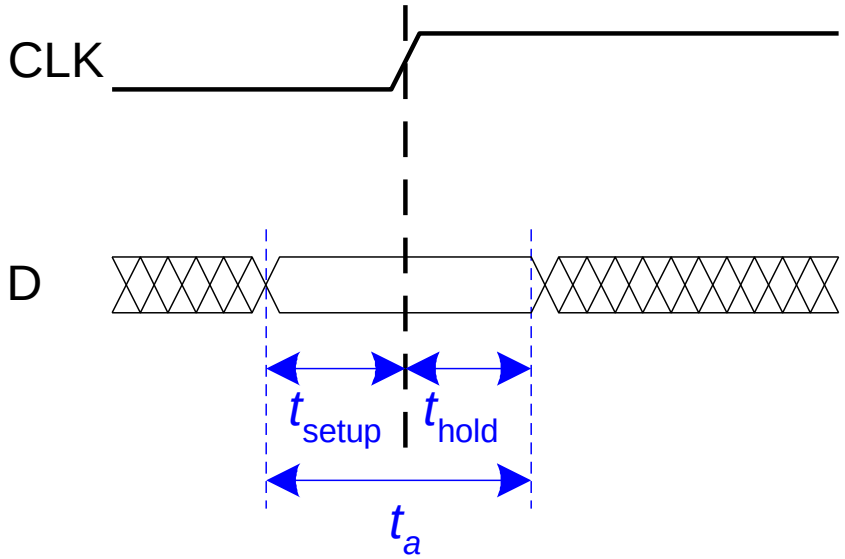


Latency = 5 + 15 = 20 minuti = **1/3 h**

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

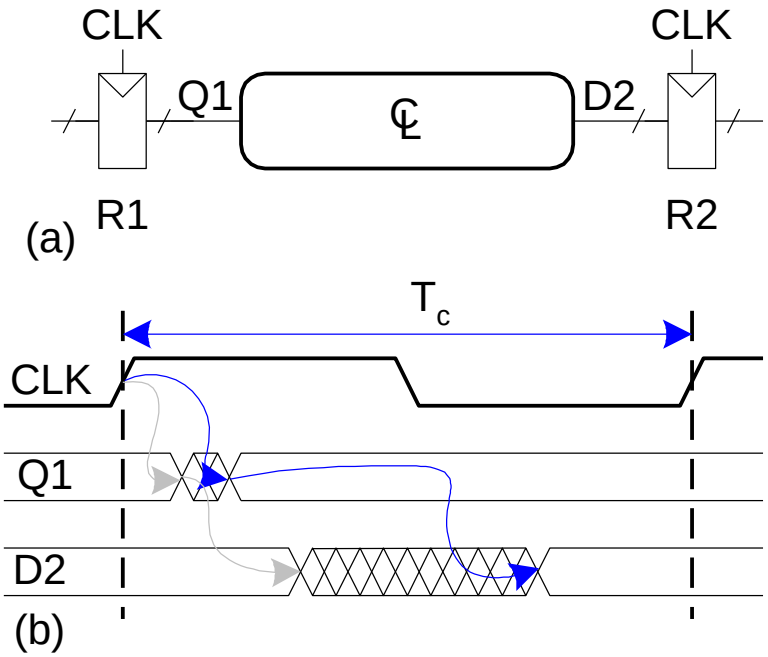
Input Timing Constraints

- **Setup time:** t_{setup} = time *before* clock edge data must be stable (i.e. not changing)
- **Hold time:** t_{hold} = time *after* clock edge data must be stable
- **Aperture time:** t_a = time *around* clock edge data must be stable ($t_a = t_{setup} + t_{hold}$)



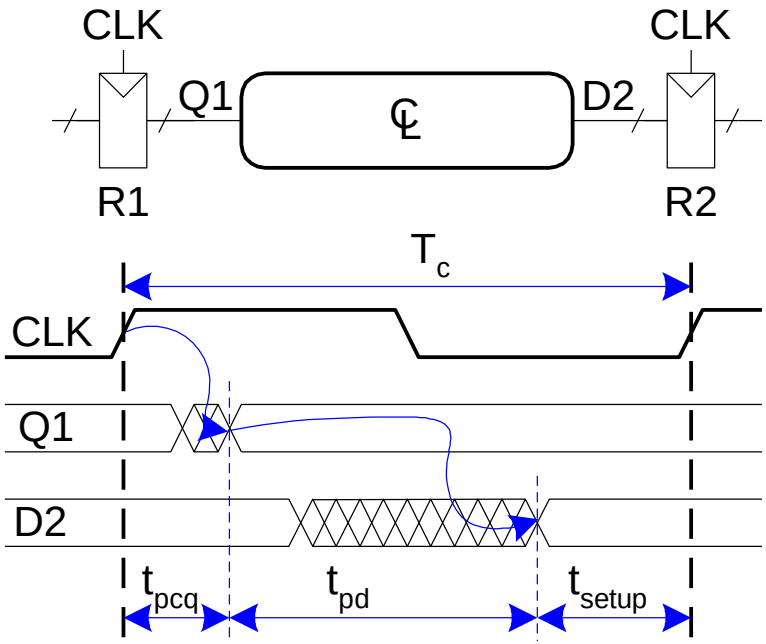
Dynamic Discipline

- The delay between registers has a **minimum** and **maximum** delay, dependent on the delays of the circuit elements



Setup Time Constraint

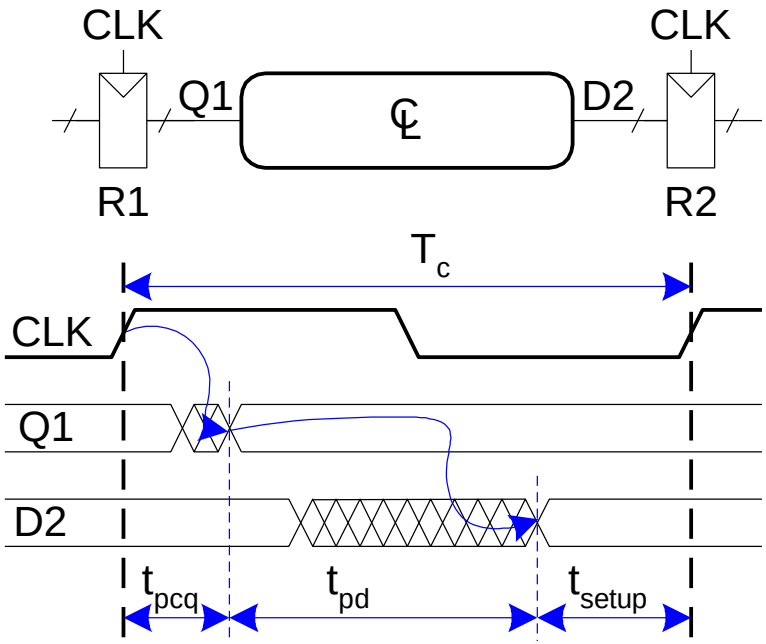
- Depends on the **maximum** delay from register R1 through combinational logic to R2
- The input to register R2 must be stable at least t_{setup} before clock edge



$$T_c \geq$$

Setup Time Constraint

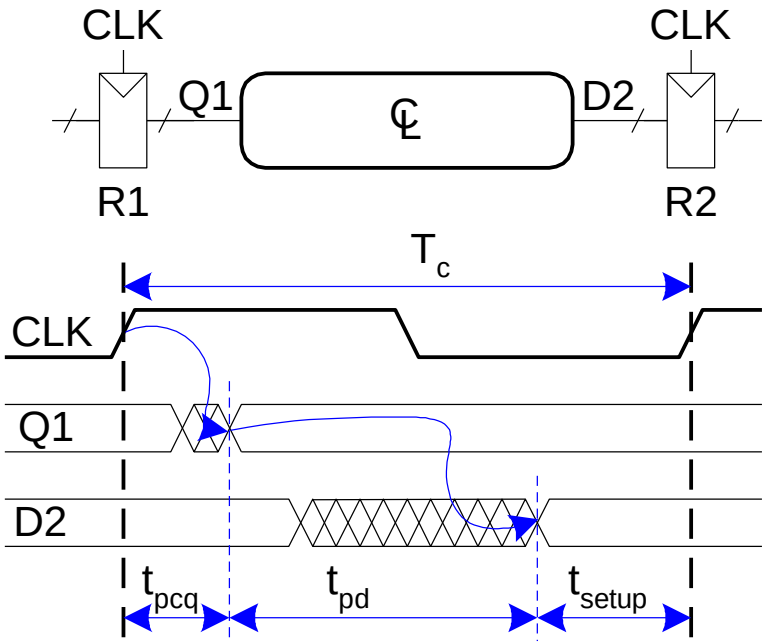
- Depends on the **maximum** delay from register R1 through combinational logic to R2
- The input to register R2 must be stable at least t_{setup} before clock edge



$$T_c \geq t_{pcq} + t_{pd} + t_{setup}$$

Setup Time Constraint

- Depends on the **maximum** delay from register R1 through combinational logic to R2
- The input to register R2 must be stable at least t_{setup} before clock edge



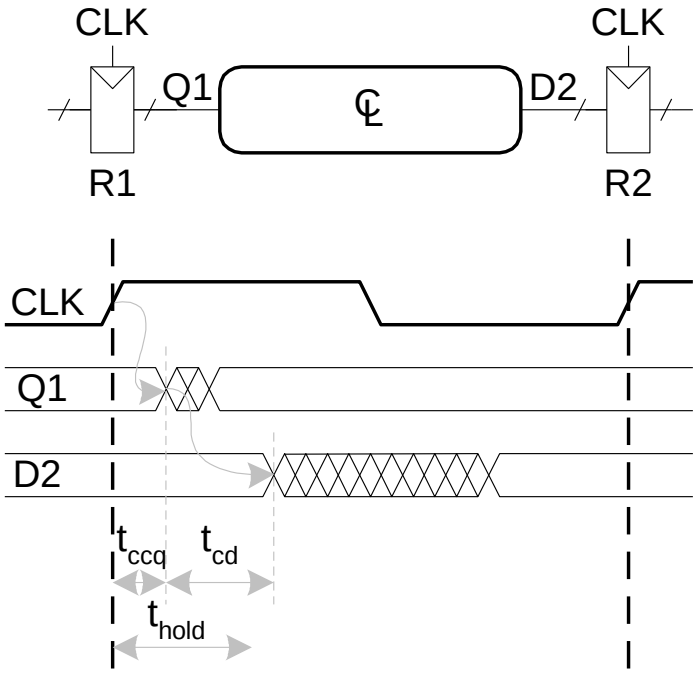
$$T_c \geq t_{pcq} + t_{pd} + t_{setup}$$

$$t_{pd} \leq T_c - (t_{pcq} + t_{setup})$$

$(t_{pcq} + t_{setup})$: *sequencing overhead*

Hold Time Constraint

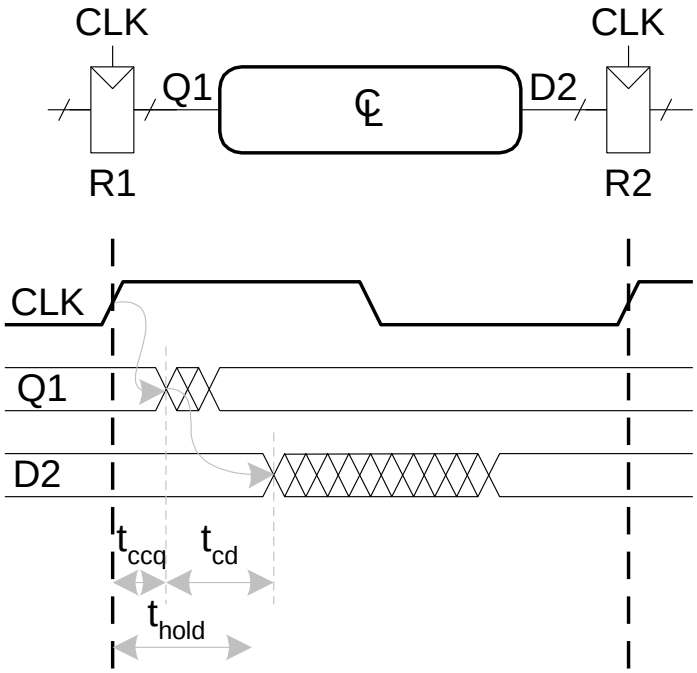
- Depends on the **minimum** delay from register R1 through the combinational logic to R2
- The input to register R2 must be stable for at least t_{hold} after the clock edge



t_{hold} <

Hold Time Constraint

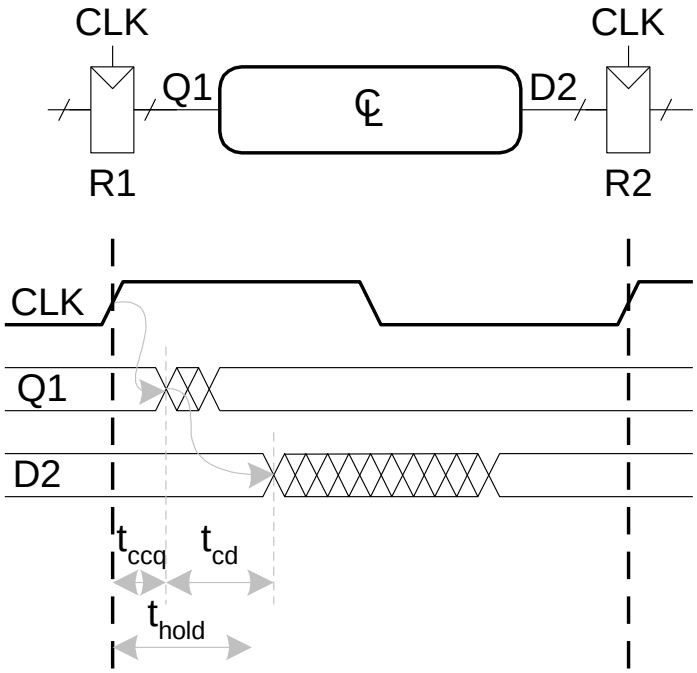
- Depends on the **minimum** delay from register R1 through the combinational logic to R2
- The input to register R2 must be stable for at least t_{hold} after the clock edge



$$t_{hold} < t_{ccq} + t_{cd}$$

Hold Time Constraint

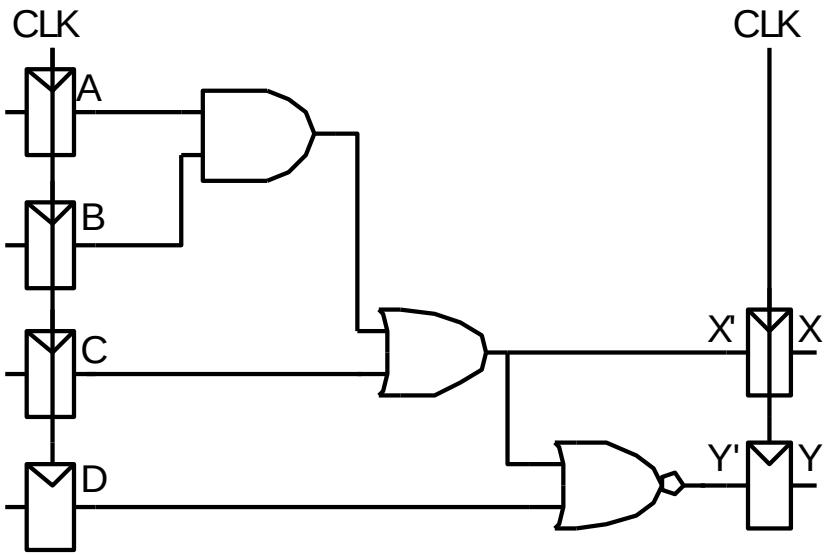
- Depends on the **minimum** delay from register R1 through the combinational logic to R2
- The input to register R2 must be stable for at least t_{hold} after the clock edge



$$t_{hold} < t_{ccq} + t_{cd}$$

$$t_{cd} > t_{hold} - t_{ccq}$$

Timing Analysis



Timing Characteristics

t_{ccq} = 30 ps
t_{pcq} = 50 ps
t_{setup} = 60 ps
t_{hold} = 70 ps

per gate:
t_{pd} = 35 ps
t_{cd} = 25 ps

t_{pd} = 3 x 35 ps = 105 ps

t_{cd} = 25 ps

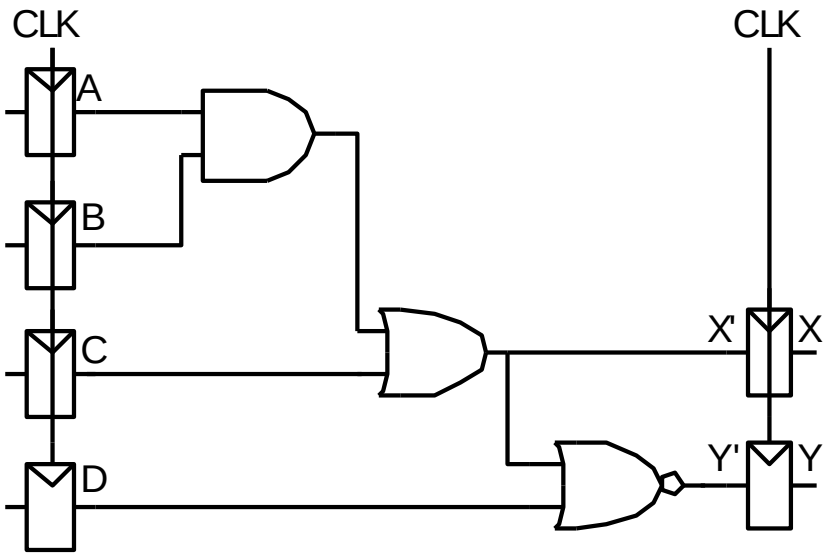
Setup time constraint:

$T_c \geq$?

Hold time constraint:

$t_{ccq} + t_{cd} > t_{hold}$?

Timing Analysis



Timing Characteristics

t_{ccq} = 30 ps
t_{pcq} = 50 ps
t_{setup} = 60 ps
t_{hold} = 70 ps

per gate:
t_{pd} = 35 ps
t_{cd} = 25 ps

t_{pd} = 3 x 35 ps = 105 ps

t_{cd} = 25 ps

Setup time constraint:

$T_c \geq (50 + 105 + 60)$ ps = 215 ps

$f_c = 1/T_c$ = 4.65 GHz

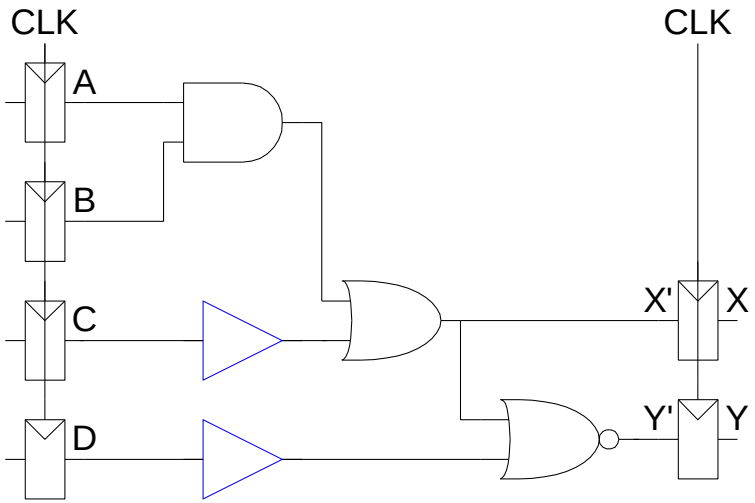
Hold time constraint:

$t_{ccq} + t_{cd} > t_{hold}$?

(30 + 25) ps > 70 ps ? **No!**

Timing Analysis

Add buffers to the short paths:



t_{pd} = 3 x 35 ps = 105 ps

t_{cd} = 2 x 25 ps = 50 ps

Setup time constraint:

$T_c \geq$?

Timing Characteristics

t_{ccq} = 30 ps
t_{pcq} = 50 ps
t_{setup} = 60 ps
t_{hold} = 70 ps

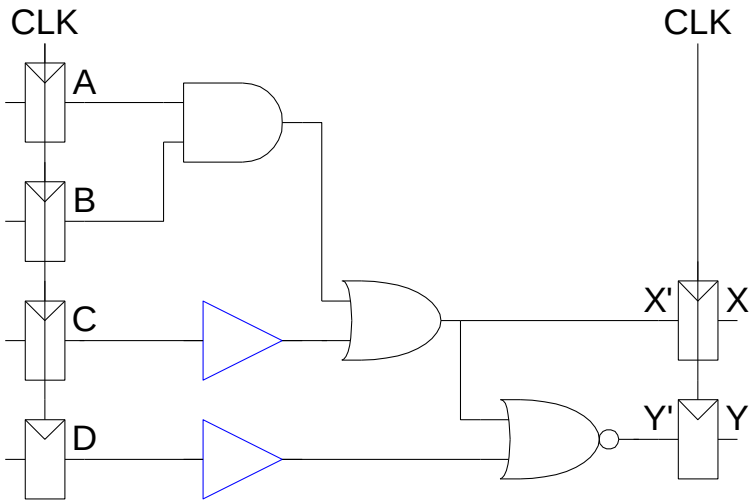
per gate:
t_{pd} = 35 ps
t_{cd} = 25 ps

Hold time constraint:

$t_{ccq} + t_{cd} > t_{hold}$?

Timing Analysis

Add buffers to the short paths:



Timing Characteristics

t_{ccq} = 30 ps
t_{pcq} = 50 ps
t_{setup} = 60 ps
t_{hold} = 70 ps

per gate:
t_{pd} = 35 ps
t_{cd} = 25 ps

t_{pd} = 3 x 35 ps = 105 ps

t_{cd} = 2 x 25 ps = 50 ps

Setup time constraint:

$T_c \geq (50 + 105 + 60)$ ps = 215 ps

$f_c = 1/T_c$ = 4.65 GHz

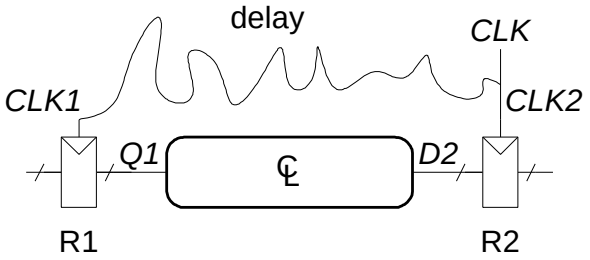
Hold time constraint:

$t_{ccq} + t_{cd} > t_{hold}$?

(30 + 50) ps > 70 ps ? **Yes!**

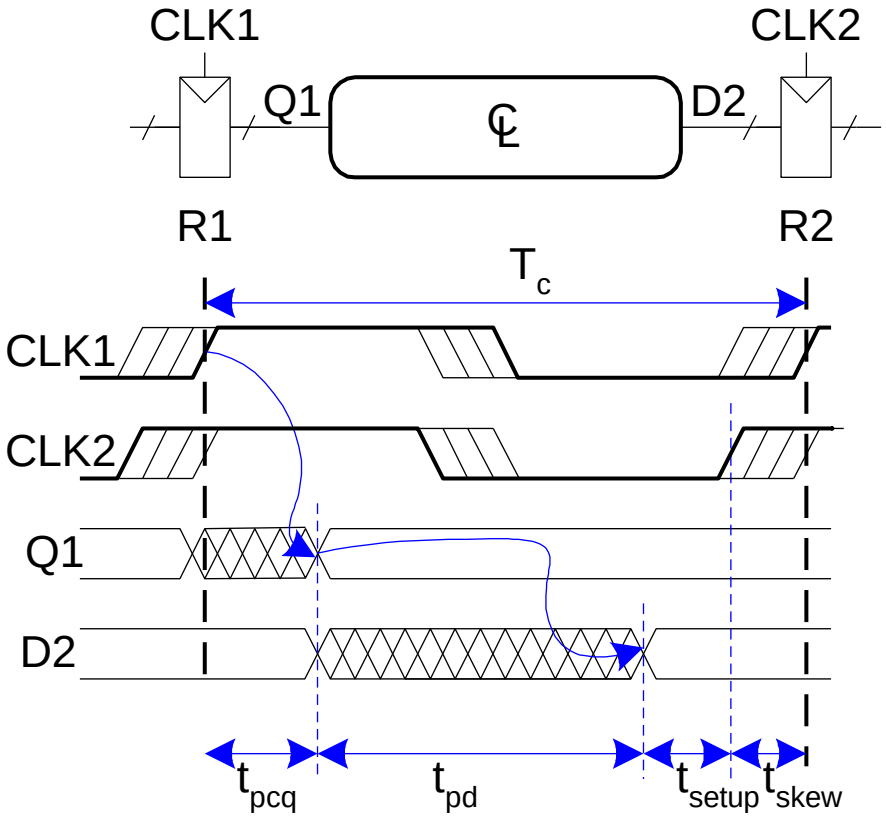
Clock Skew

- The clock doesn't arrive at all registers at same time
- **Skew:** difference between two clock edges
- Perform **worst case analysis** to guarantee dynamic discipline is not violated for any register – many registers in a system!



Setup Time Constraint with Skew

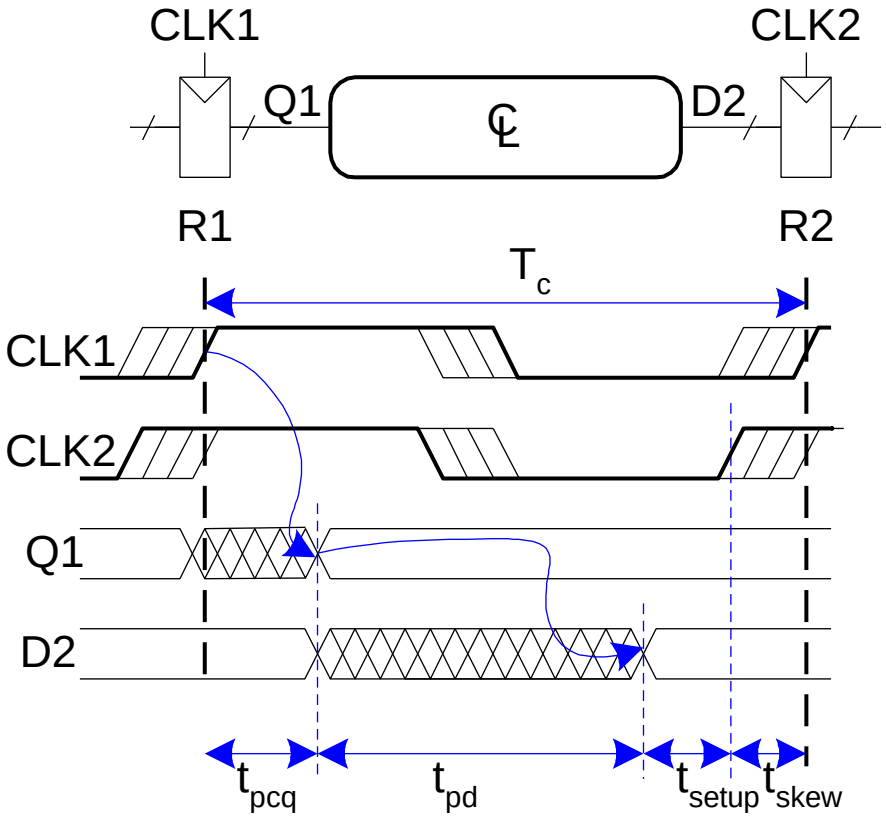
- In the worst case, CLK2 is earlier than CLK1



$T_c \geq ?$

Setup Time Constraint with Skew

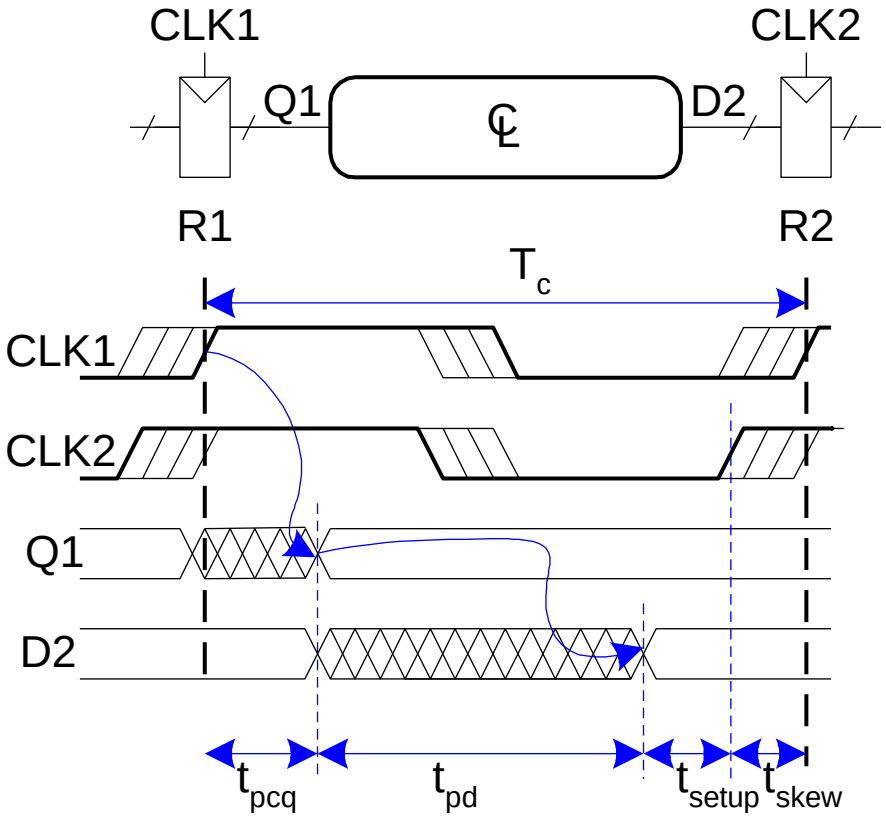
- In the worst case, CLK2 is earlier than CLK1



$$T_c \geq t_{pcq} + t_{pd} + t_{setup} + t_{skew}$$

Setup Time Constraint with Skew

- In the worst case, CLK2 is earlier than CLK1

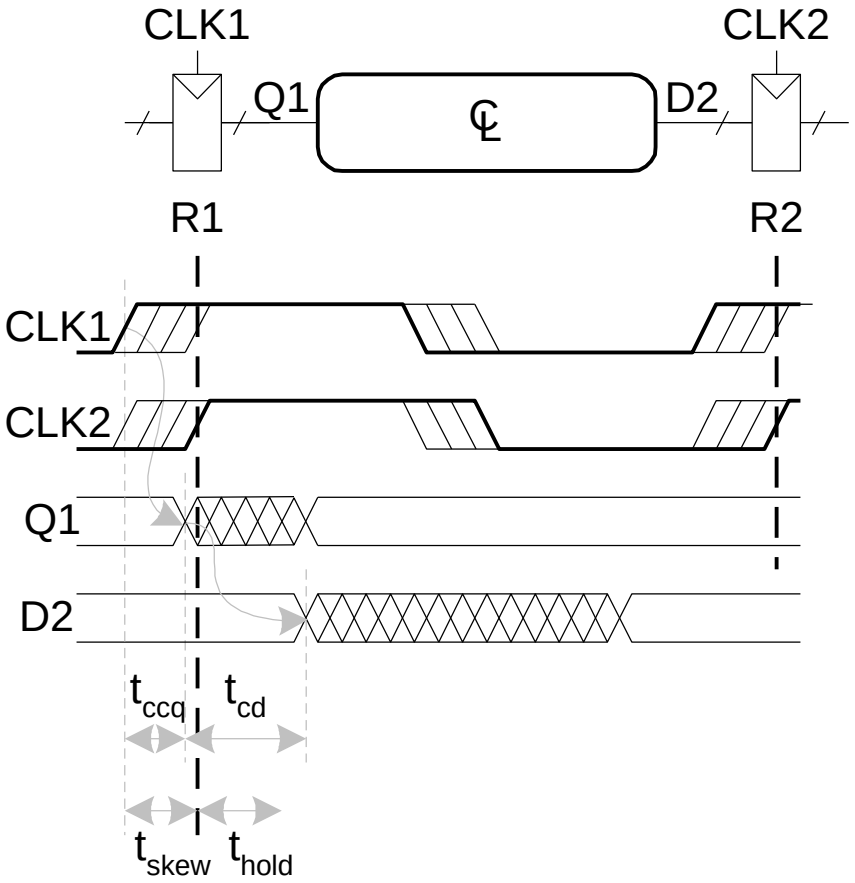


$$T_c \geq t_{pcq} + t_{pd} + t_{setup} + t_{skew}$$

$$t_{pd} \leq T_c - (t_{pcq} + t_{setup} + t_{skew})$$

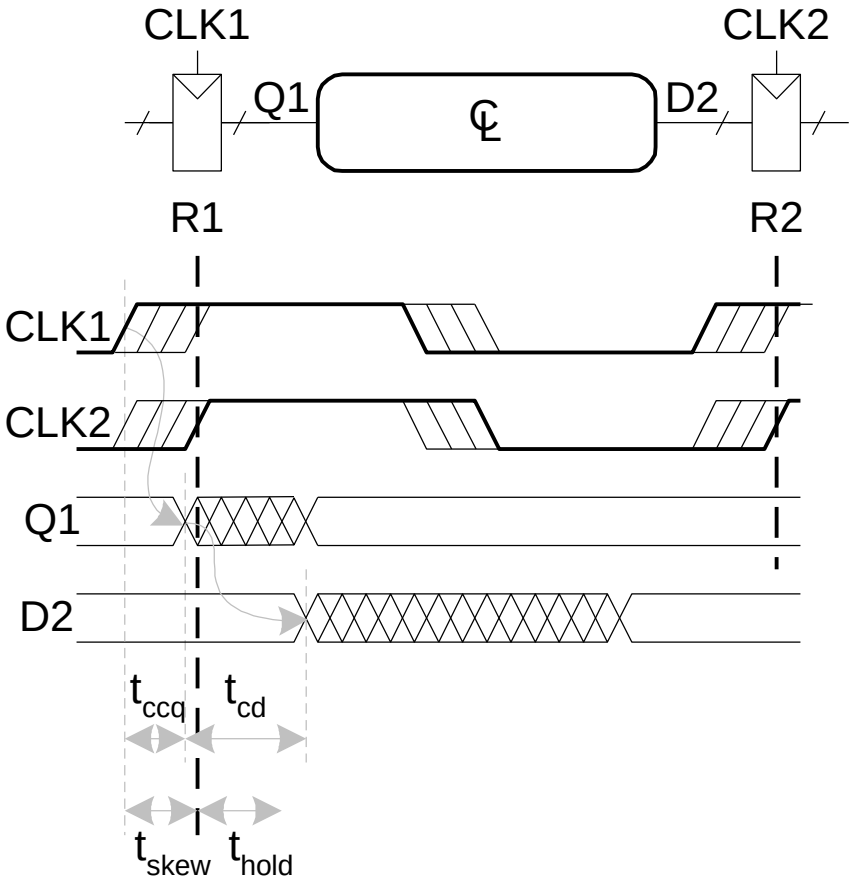
Hold Time Constraint with Skew

- In the worst case, CLK2 is later than CLK1



Hold Time Constraint with Skew

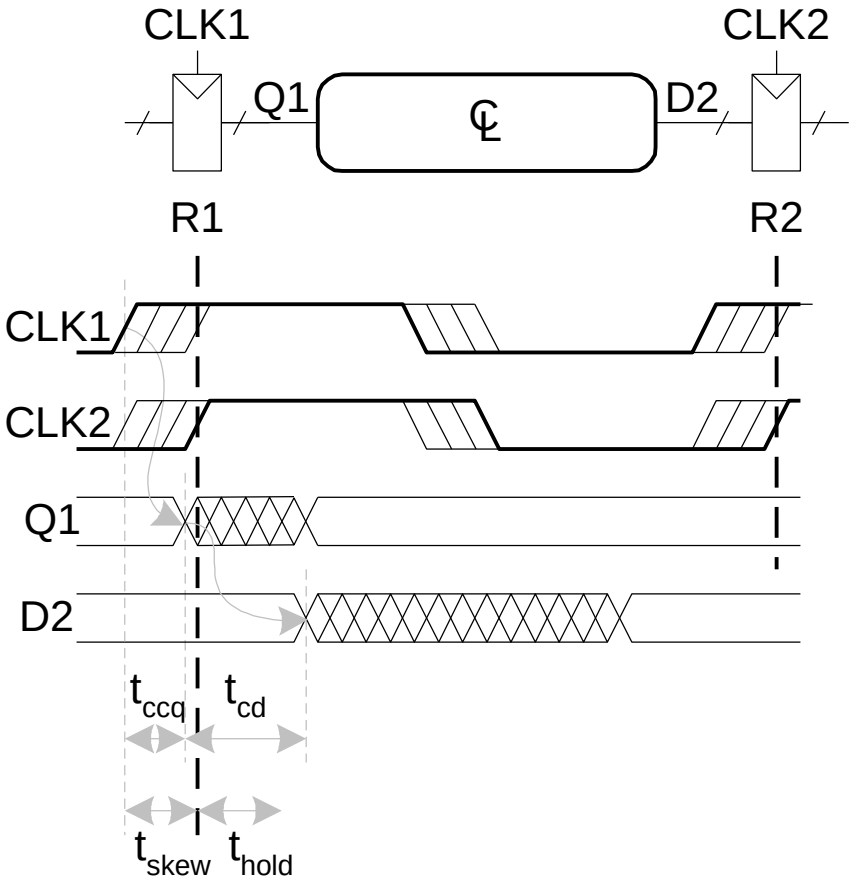
- In the worst case, CLK2 is later than CLK1



$$t_{ccq} + t_{cd} > t_{hold} + t_{skew}$$

Hold Time Constraint with Skew

- In the worst case, CLK2 is later than CLK1

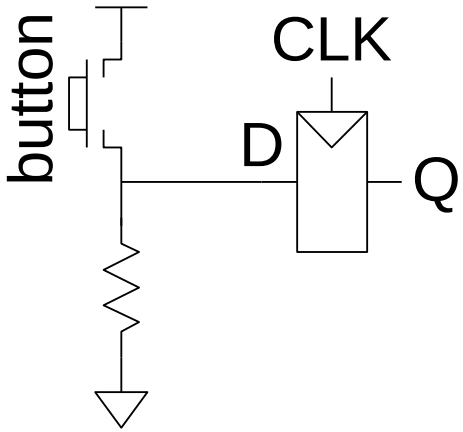


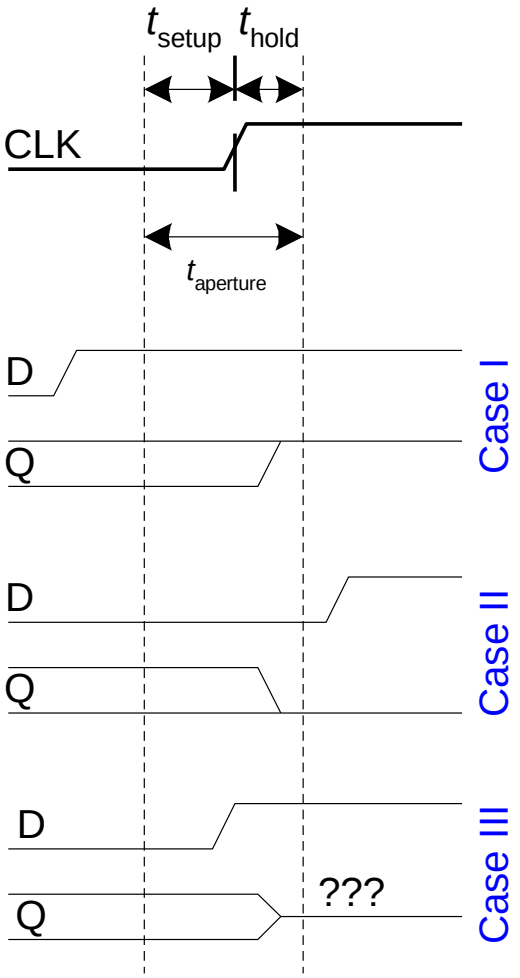
$$t_{ccq} + t_{cd} > t_{hold} + t_{skew}$$

$$t_{cd} > t_{hold} + t_{skew} - t_{ccq}$$

Violating the Dynamic Discipline

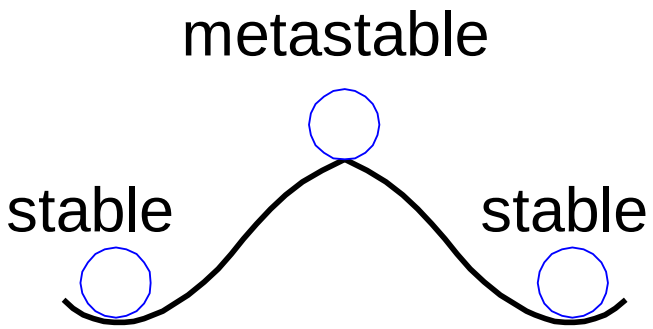
Asynchronous (for example, user) **inputs** might violate the dynamic discipline





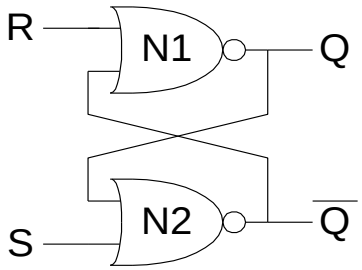
Metastability

- **Bistable devices:** two stable states, and a metastable state between them
- **Flip-flop:** two stable states (1 and 0) and one metastable state
- If flip-flop lands in metastable state, could stay there for an undetermined amount of time



Flip-Flop Internals

- Flip-flop has **feedback**: if *Q* is somewhere between 1 and 0, cross-coupled gates drive output to either rail (1 or 0)



- **Metastable signal:** if it hasn't resolved to 1 or 0
- If flip-flop input changes at random time, **probability that output *Q* is metastable** after waiting some time, *t*:

$$\mathbf{P}(t_{res} > t) = (T_0/T_c)\, e^{-t/\tau}$$

t_{res} : time to resolve to 1 or 0

T_0, τ : properties of the circuit

Metastability

- **Intuitively:**

  T_0/T_c: probability input changes at a bad time (during aperture)

  $$P(t_{res} > t) = (T_0/T_c)\, e^{-t/\tau}$$

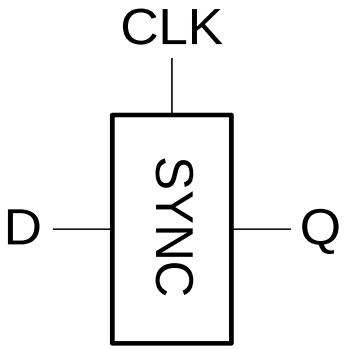
  τ: time constant for how fast flip-flop moves away from metastability

  $$P(t_{res} > t) = (T_0/T_c)\, e^{-t/\tau}$$

- If flip-flop samples metastable input, if you wait long enough (t), the output will have resolved to 1 or 0 with high probability.

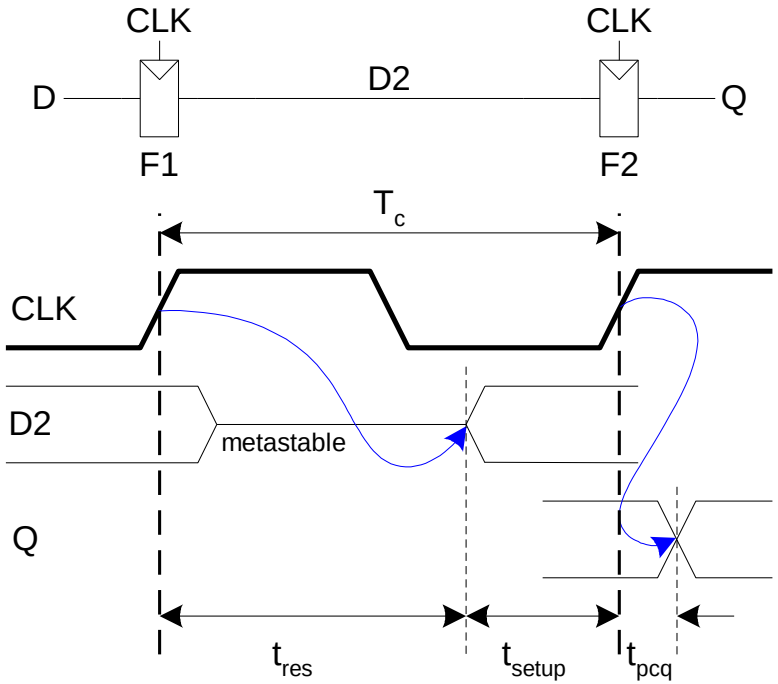
Synchronizers

- **Asynchronous inputs are inevitable** (user interfaces, systems with different clocks interacting, etc.)
- **Synchronizer goal:** make the probability of failure (the output *Q* still being metastable) low
- Synchronizer cannot make the probability of failure 0



Synchronizer Internals

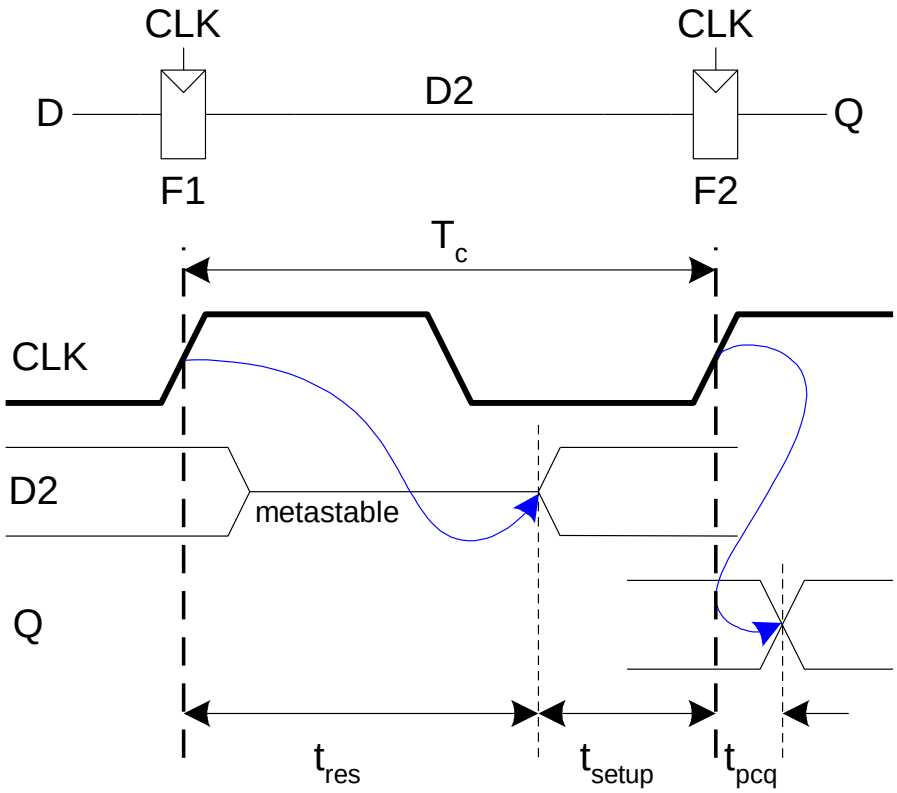
- Synchronizer: built with two back-to-back flip-flops
- Suppose D is transitioning when sampled by F1
- Internal signal D2 has (T_c - t_{setup}) time to resolve to 1 or 0



Synchronizer Probability of Failure

For each sample, probability of failure is:

$$\mathbf{P(failure)} = (T_0/T_c)\, e^{-(T_c - t_{setup})/\tau}$$



Synchronizer Mean Time Between Failures

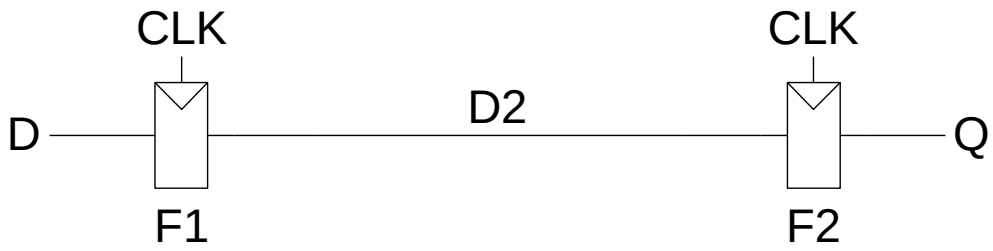
- If asynchronous input changes once per second, probability of failure per second is *P*(failure).
- If input changes *N* times per second, probability of failure per second is:

$$P(\text{failure})/\text{second} = (NT_0/T_c)\, e^{-(T_c - t_{setup})/\tau}$$

- Synchronizer fails, on average, 1/[*P*(failure)/second]
- Called ***mean time between failures***, MTBF:

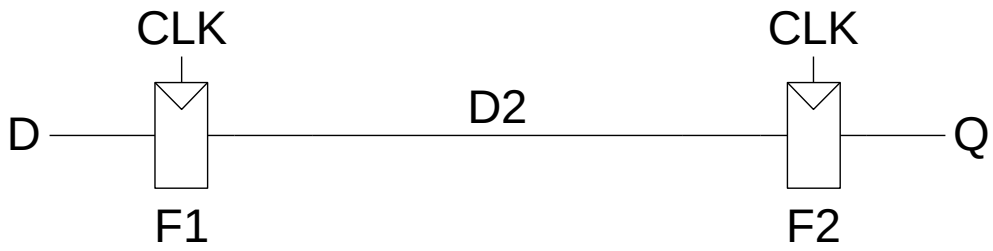
$$\text{MTBF} = 1/[P(\text{failure})/\text{second}] = (T_c/NT_0)\, e^{(T_c - t_{setup})/\tau}$$

Example Synchronizer



- Suppose: T_c = 1/500 MHz = 2 ns τ = 200 ps
  T_0 = 150 ps t_{setup} = 100 ps
  N = 10 events per second
- What is the probability of failure? MTBF?

Example Synchronizer



- Suppose: T_c = 1/500 MHz = 2 ns τ = 200 ps
  T_0 = 150 ps t_{setup} = 100 ps
  N = 10 events per second
- What is the probability of failure? MTBF?

$$P(\text{failure}) = (150\text{ ps}/2\text{ ns})\, e^{-(1.9\text{ ns})/200\text{ ps}}$$
$$= \mathbf{5.6 \times 10^{-6}}$$
$$P(\text{failure})/\text{second} = 10 \times (5.6 \times 10^{-6})$$
$$= 5.6 \times 10^{-5} / \text{second}$$
$$\text{MTBF} = 1/[\text{P(failure)/second}] \approx \mathbf{5\text{ hours}}$$