Materiale didattico

- Paolo Corsini, "Dalle porte AND, OR, NOT al Sistema calcolatore", edizioni ETS

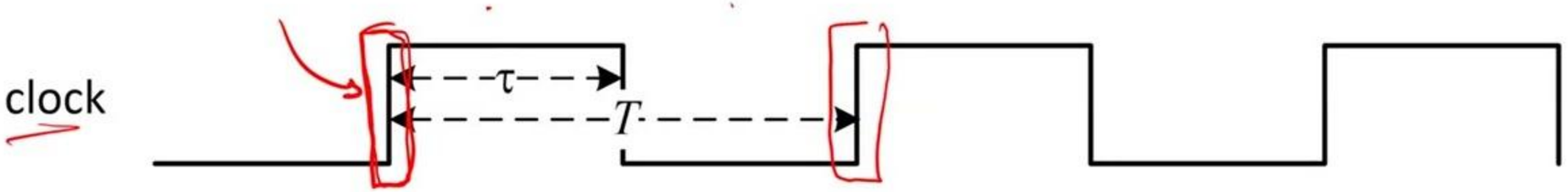
- Appunti sulle Reti Sequenziali - versione 25/09/2020 (con esercizi svolti)

Reti sequenziali sincronizzate

- Le RSS si evolvono **soltanto in corrispondenza di istanti temporali ben precisi**, detti appunto **istanti di sincronizzazione**.
- Tali istanti devono essere opportunamente distanziati (non possono essere troppo ravvicinati).
- Non sono i **cambiamenti di ingresso** che fanno evolvere una RSS, come era invece per le RC e le RSA (reti **asincrone**)

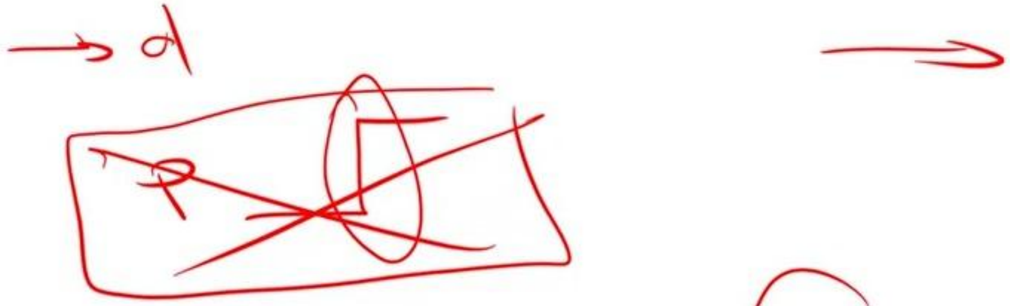
Reti sequenziali sincronizzate (cont.)

- La sincronizzazione si **realizza** portando alle reti un **segnale di ingresso particolare, detto clock**



- Forma d'onda **periodica**, di **frequenza nota** $\mathbf{1/T}$
- Non necessariamente il **duty-cycle** $\boldsymbol{\tau/T}$ è del 50%
  - Ma non può essere troppo piccolo (né troppo grande)
- L'evento che **sincronizza** la rete che riceve questo segnale è, di norma, il **fronte di salita del clock**.

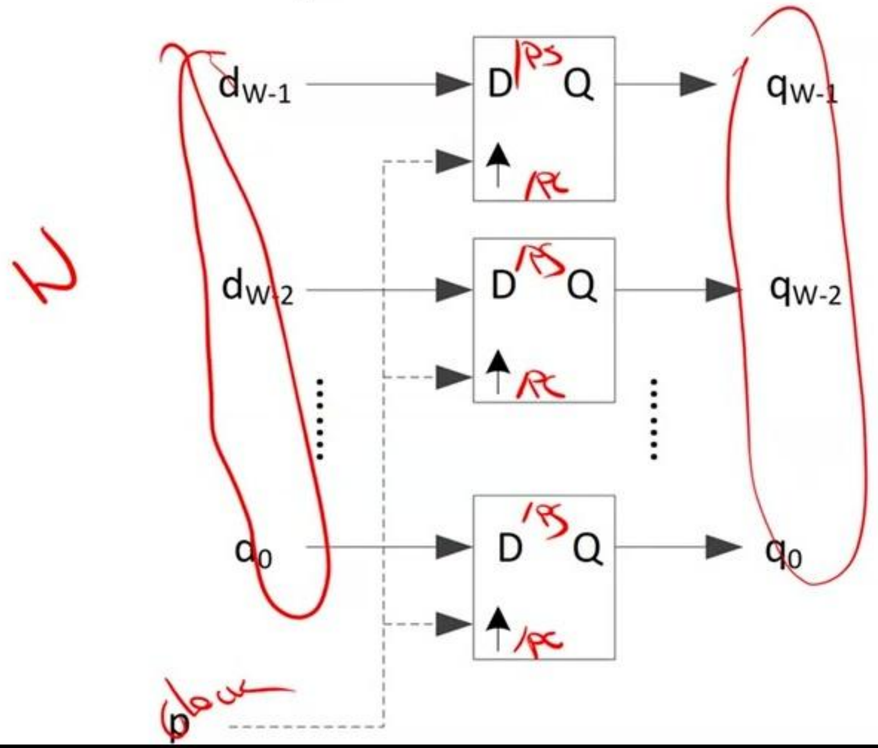
Registri

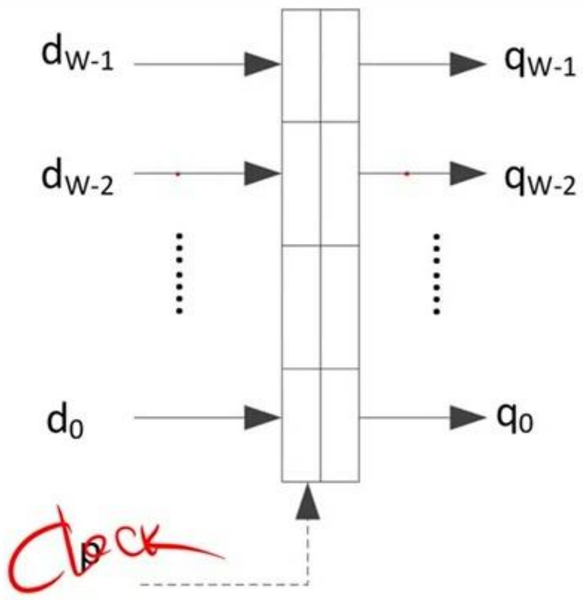


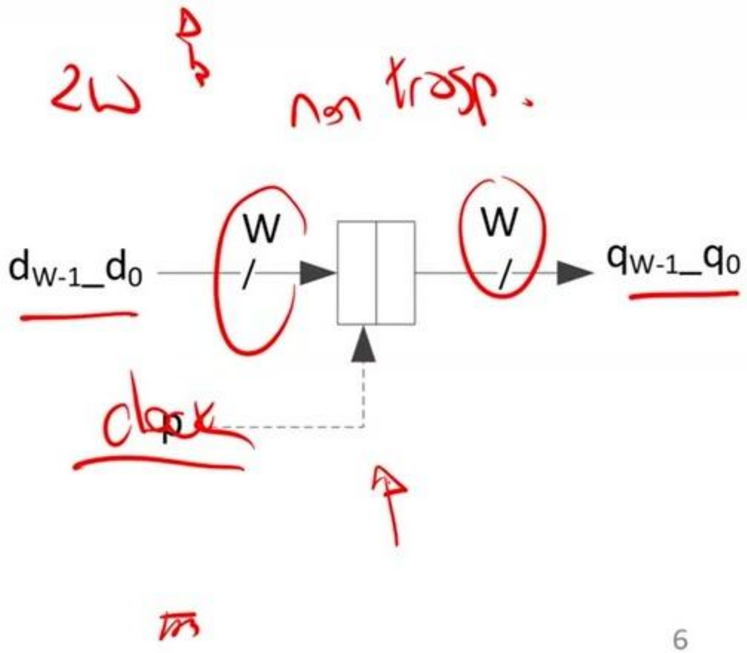
- Un **registro a W bit** è una **collezione di W D-Flip-Flop Positive-Edge-Triggered**, che hanno:
  a) ingressi d_i ed uscite q_i separati (cioè indipendenti)
  b) ingresso p a comune
- Un registro può essere visto come una **RSS**, in cui l'ingresso p funge da **segnale di sincronizzazione**

Registri (cont.)

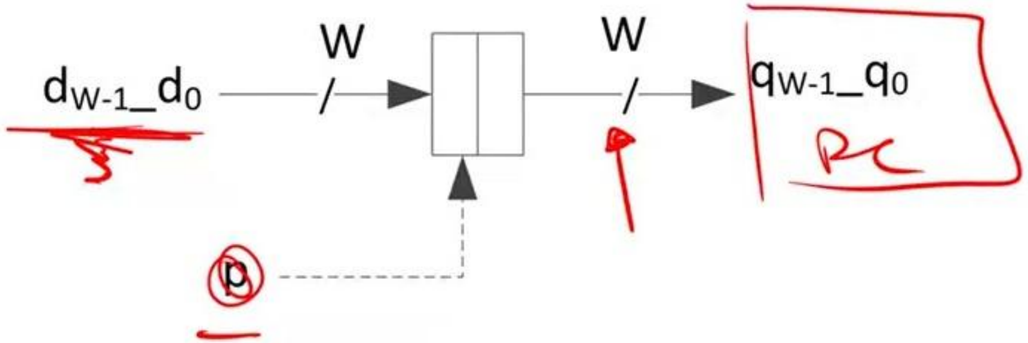
- Pur essendo il D-FF una rete sequenziale **asincrona**
  - se considero i due ingressi d e p nella loro generalità,
- Niente vieta di attribuire all'ingresso p un **valore speciale**, appunto quello di **segnale di sincronizzazione**, e vedere il D-FF come una rete sequenziale **sincronizzata**





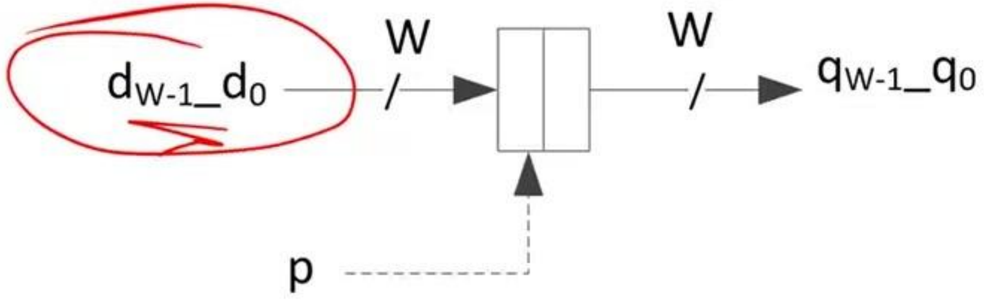


Registri (cont.)



- p non specifica più **un valore di ingresso**
  - posso smettere di contarlo tra gli ingressi
  - non mi interessa il suo valore, ma soltanto l'istante in cui transisce da 0 ad 1
- Il registro a W bit (W = **capacità** del registro) **ha W ingressi e W uscite**
- Stato di uscita del registro ad un certo istante: **contenuto** del registro
- **lettura** del registro
  - Utilizzo del contenuto (e.g., per fornire ingresso ad una RC)
- **scrittura** del registro
  - Memorizzazione di W bit in ingresso ad un istante di sincronizzazione
- **inizializzazione** del registro
  - collegare $/preset$ e $/preclear$ di ciascun D-FF alla variabile di $/reset$ o ad 1

Registri (cont.)

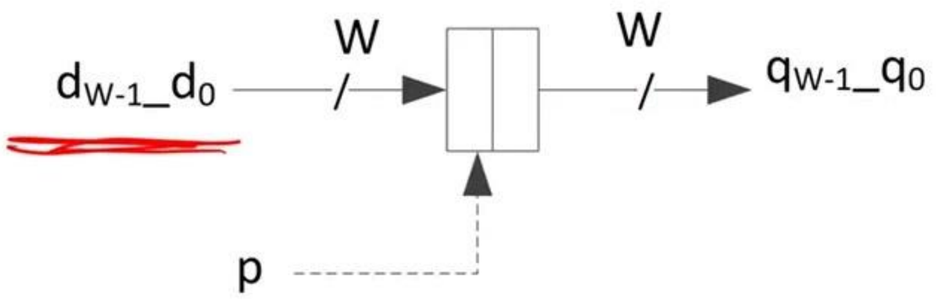


- Requisiti **di pilotaggio** per un registro
- Gli ingressi d_i devono essere **stabili** intorno al fronte di salita del clock,
  - per un tempo T_{setup} prima e T_{hold} dopo
- L'uscita cambia dopo $T_{prop} > T_{hold}$

- Tutto ciò che accade in ingresso al di fuori di questo intervallo è **irrilevante, e non verrà memorizzato**
- Posso montare un registro nei modi più barbari senza che si perda la prevedibilità dell'evoluzione del suo stato

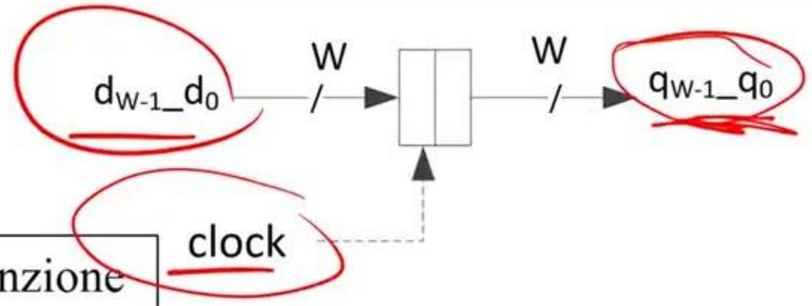
Registri (cont.)

RC
Sasinc.



- I registri memorizzano il proprio stato di ingresso al fronte di salita del clock
- Il fatto che **due stati di ingresso ai registri**, presentati su istanti di clock (fronti di salita) **consecutivi**, siano **identici, adiacenti o non adiacenti** non riveste alcuna importanza.
- Tra due fronti di salita del clock, **lo stato di ingresso ai registri può cambiare in qualunque modo (o non cambiare affatto)**.
- Al nuovo fronte di salita del clock, lo stato di ingresso presente verrà memorizzato
  - come se fosse un nuovo stato, anche se identico al precedente
- Inoltre, le **uscite cambiano T_{prop} dopo il fronte di salita del clock,** e **restano costanti** per tutto un periodo

Descrizione di registri in Verilog



Nomi dei registri **MAIUSCOLI** per convenzione

bloccante

Blocco **assign** – **assegnamento continuo** (aggiornamento uscite)

ass. continue

```
reg [W-1:0] REGISTRO;
wire clock, reset_;
wire [W-1:0] q_{W-1}_q_0; assign q_{W-1}_q_0=REGISTRO;
```

ass. procedurale non bloccante

```
always @(reset_==0) #1 REGISTRO<=contenuto_iniziale;
always @(posedge clock) if (reset_==1) #T_prop REGISTRO<=d_{W-1}_d_0;
```

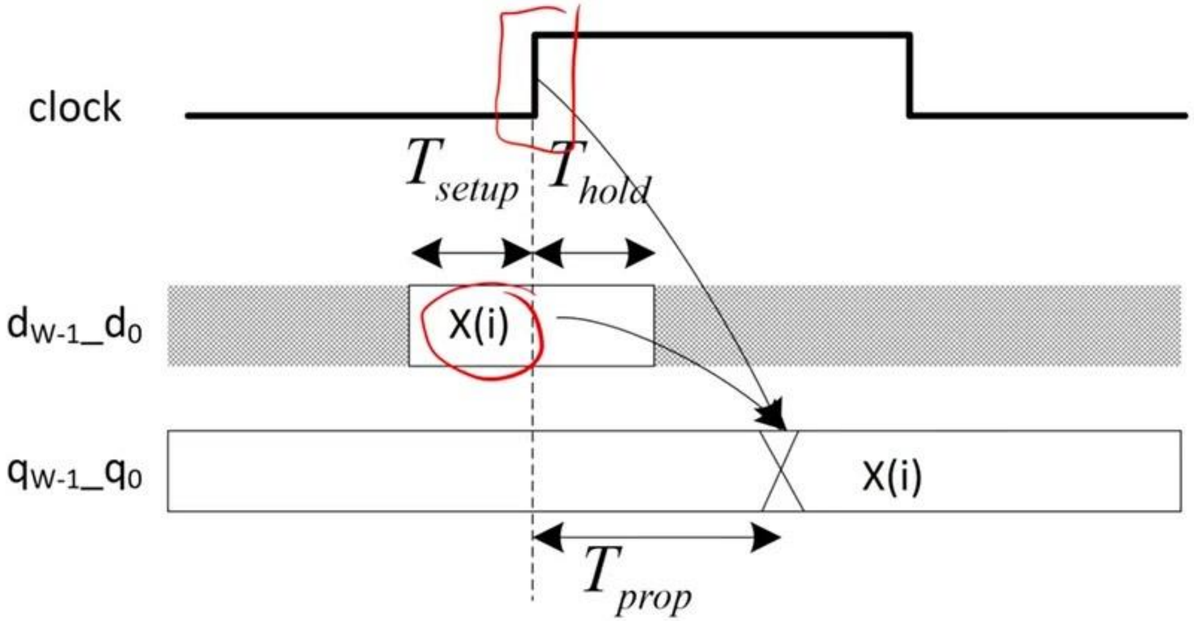
@: Controllo degli eventi

Tempo di propagazione. Conviene che sia >0, perché sennò nelle simulazioni non si capisce cosa succede.

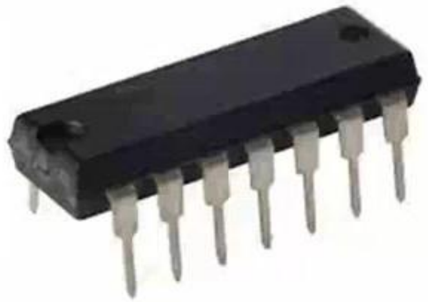
Assegnamento procedurale **non bloccante** "<="

Temporizzazione del registro

- La **temporizzazione** del registro è scritta di seguito.

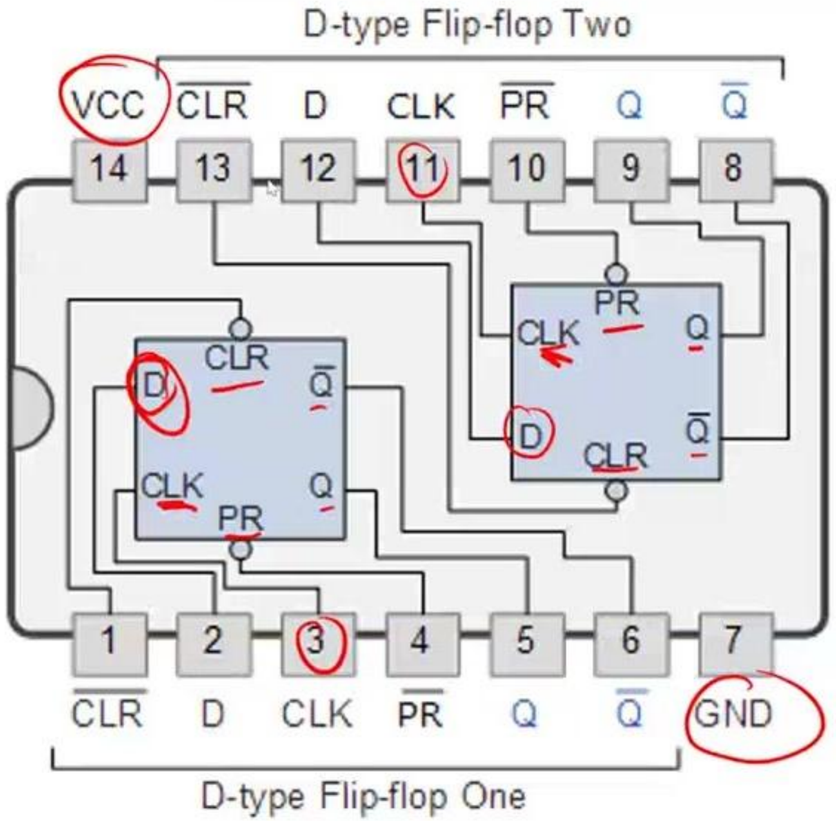


TI 74LS74 Dual D-type Flip Flop



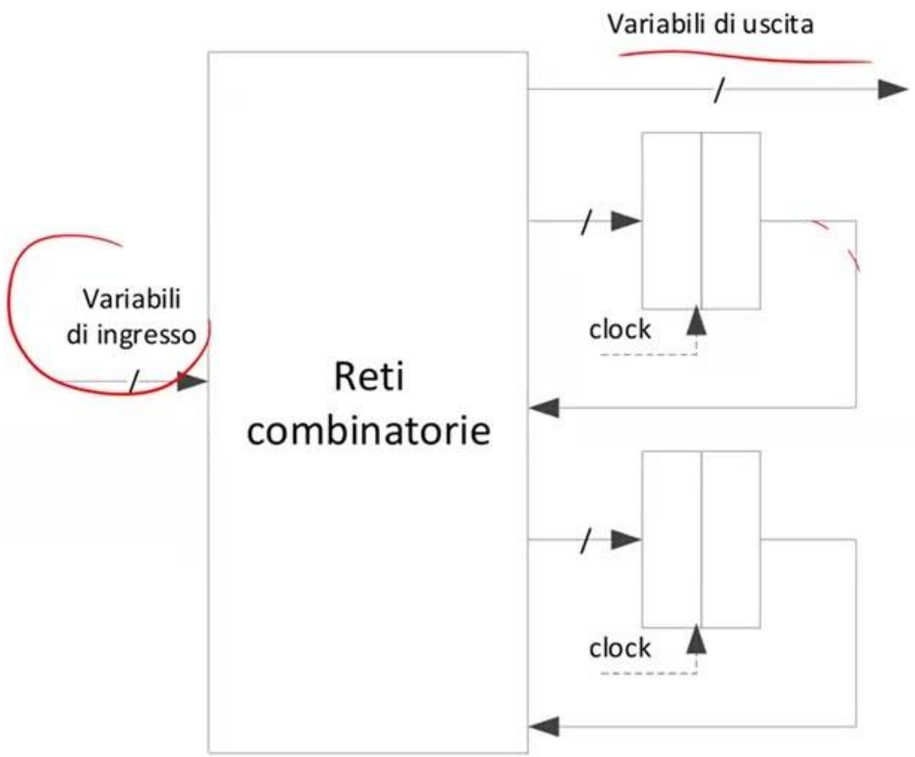
- Due D-FF autonomi ad un bit

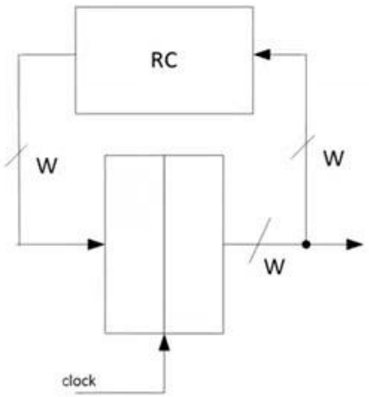
- 3,11: clock dei singoli registri
  - Se li collego insieme, ottengo un registro a 2 bit
- 13,10,1,4: inizializzazione al reset
- 5,9: uscita diretta
- 6,8: uscita negata
- 14,7: tensione e a massa (14, 7)

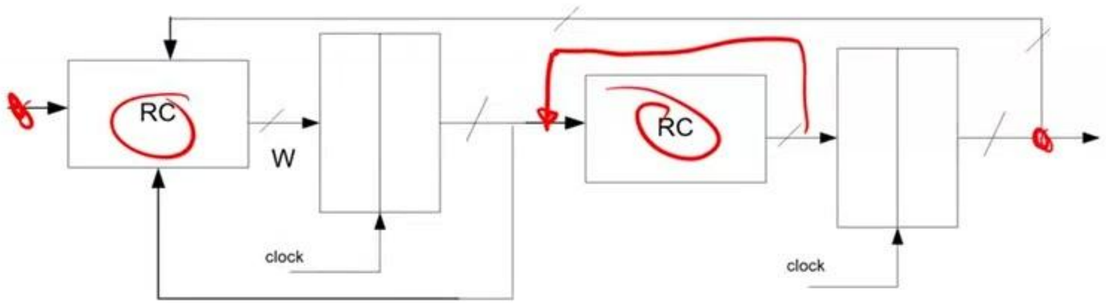


Rete sequenziale sincronizzata

- **Collezione di registri e di reti combinatorie**, montati arbitrariamente
  - purché non ci siano **anelli di reti combinatorie** (che darebbero vita ad una **rete sequenziale asincrona**),
  - i **registri hanno tutti lo stesso clock**.
- Ci possono essere, invece, anelli che **abbiano registri al loro interno**, in quanto questo non crea alcun problema





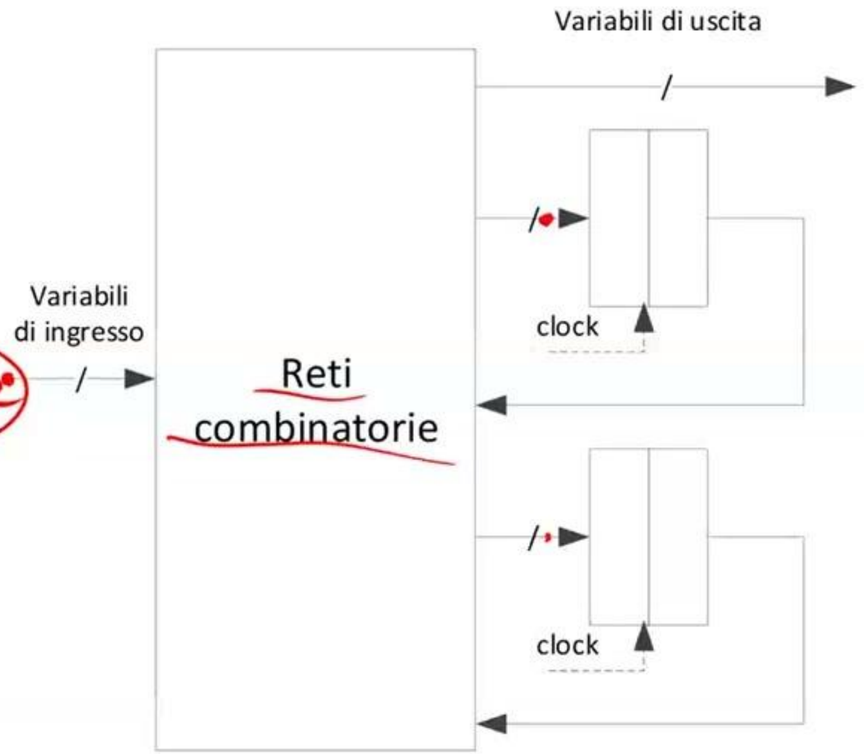


Regole di pilotaggio per una RSS

T ≥ . . . .

- Detto t_i l'i-esimo fronte di salita del clock, lo stato di ingresso ai registri deve essere **stabile in** $\left[\boldsymbol{t_i - T_{setup}}, \boldsymbol{t_i + T_{hold}}\right]$, per ogni i

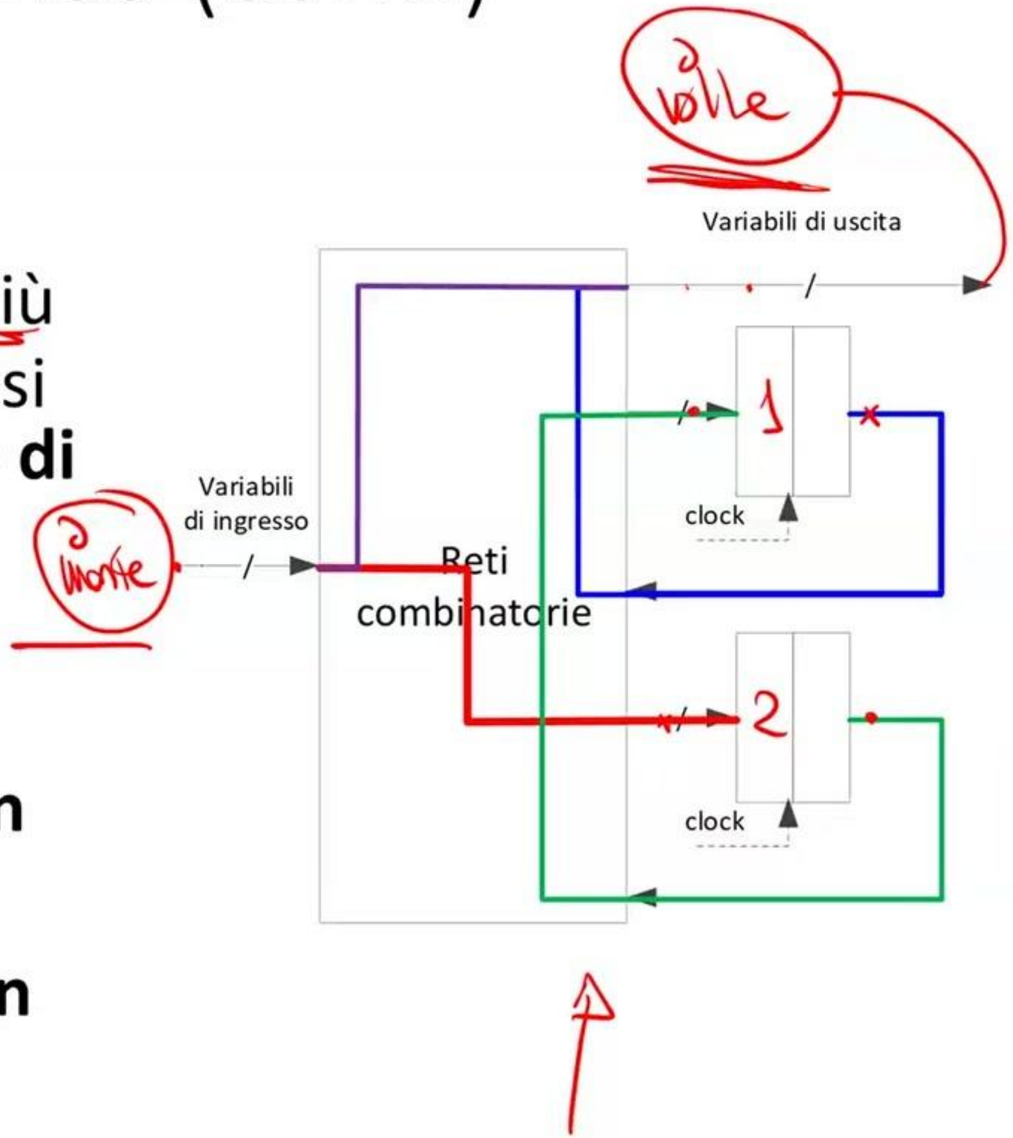
- Non posso fare il clock **veloce quanto voglio**. morte
- Se voglio che uno stato di ingresso, attraverso le reti combinatorie, concorra a formare gli ingressi ai registri, dovrò **dare il tempo a chi pilota la rete:**
  a) di produrre un nuovo stato di ingresso
  b) di farlo arrivare, attraverso le RC, fino in ingresso ai registri



Regole di pilotaggio per una RSS (cont.)

- Ritardi caratteristici di una RSS

- $T_{in_to_reg}$: il tempo di attraversamento della più lunga catena fatta di **sole** reti combinatorie che si trovi tra **un piedino di ingresso** fino **all'ingresso di un registro**
- $T_{reg_to_reg}$: (… … …) **l'uscita di un registro** e **l'ingresso di un registro**
- $T_{in_to_out}$: (… … …) **un piedino di ingresso** e **un piedino di uscita**
- $T_{reg_to_out}$: (… … …) **l'uscita di un registro** e **un piedino di uscita**



Regole di pilotaggio per una RSS (cont.)

$T \geq \ldots$

- **Tre vincoli temporali**
- Ingressi dei registri **costanti in** $\boldsymbol{[t_i - T_{setup}, t_i + T_{hold}]}$
- **Vincolo di pilotaggio in ingresso:** chi pilota gli ingressi ("a monte" della RSS) deve avere almeno un tempo T_{a_monte} per poterli cambiare
  - Al netto di tutti i ritardi, dovrò lasciare una finestra larga almeno T_{a_monte} in ogni clock per il pilotaggio della rete
- **Vincolo di pilotaggio in uscita**: chi usa le uscite ("a valle" della RSS) deve averle stabili per un tempo T_{a_valle} per poterle usare
  - Al netto di tutti i ritardi, dovrò lasciare una finestra larga almeno T_{a_valle} in ogni clock perché si possano usare le uscite

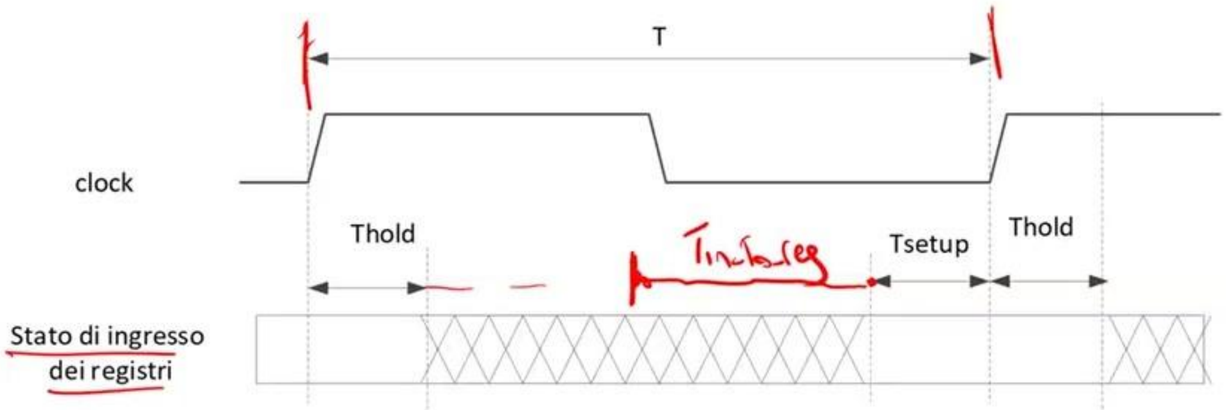
Disuguaglianze di temporizzazione

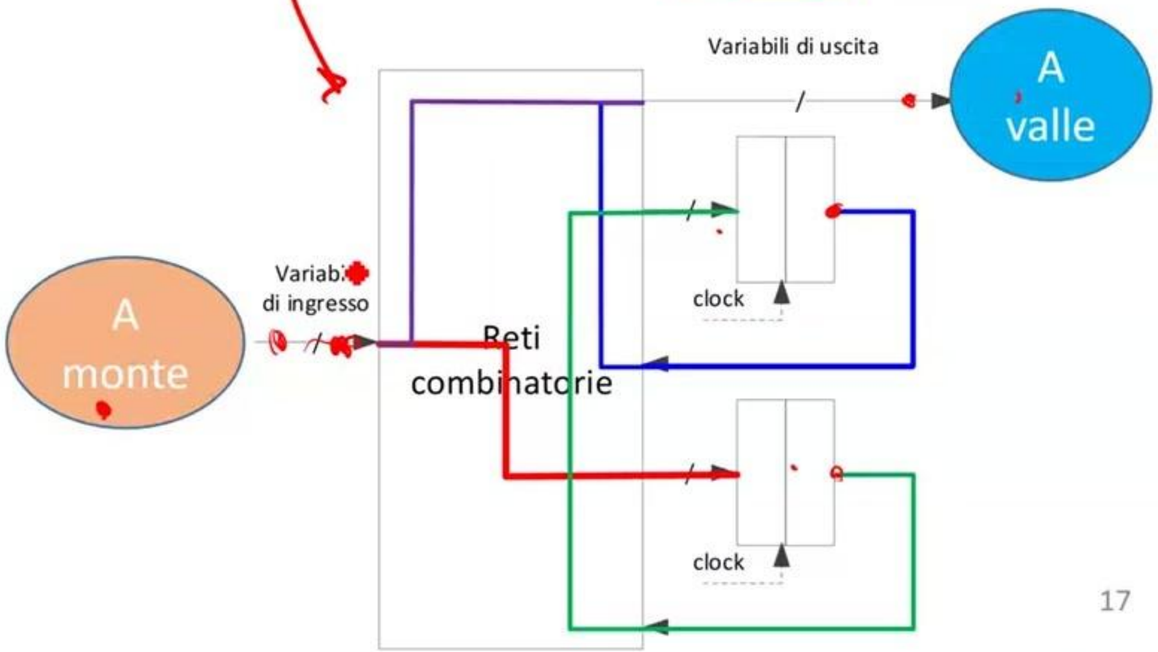
1) $T \geq T_{hold} + T_{a_monte} + T_{in_to_reg} + T_{setup}$ (in -> reg)

2) $T \geq T_{prop} + T_{reg_to_reg} + T_{setup}$ (reg -> reg)

3) $T \geq T_{hold} + T_{a_monte} + T_{in_to_out} + T_{a_valle}$ (in -> out)

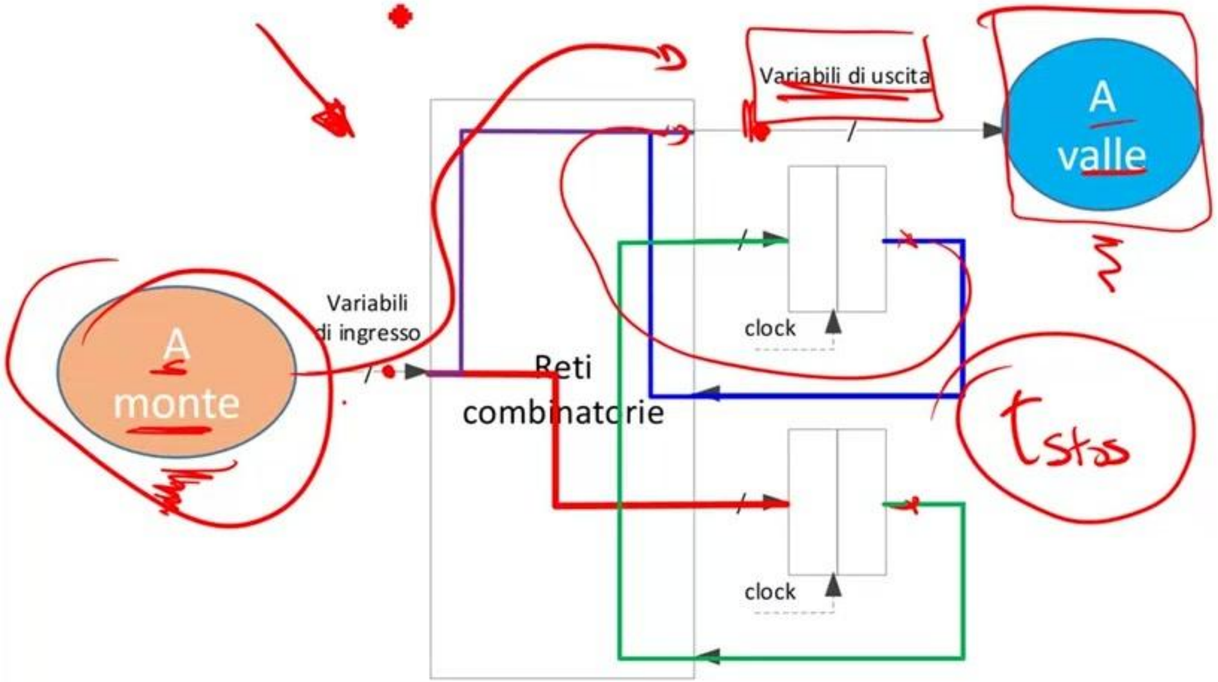
4) $T \geq T_{prop} + T_{reg_to_out} + T_{a_valle}$ (reg -> out)

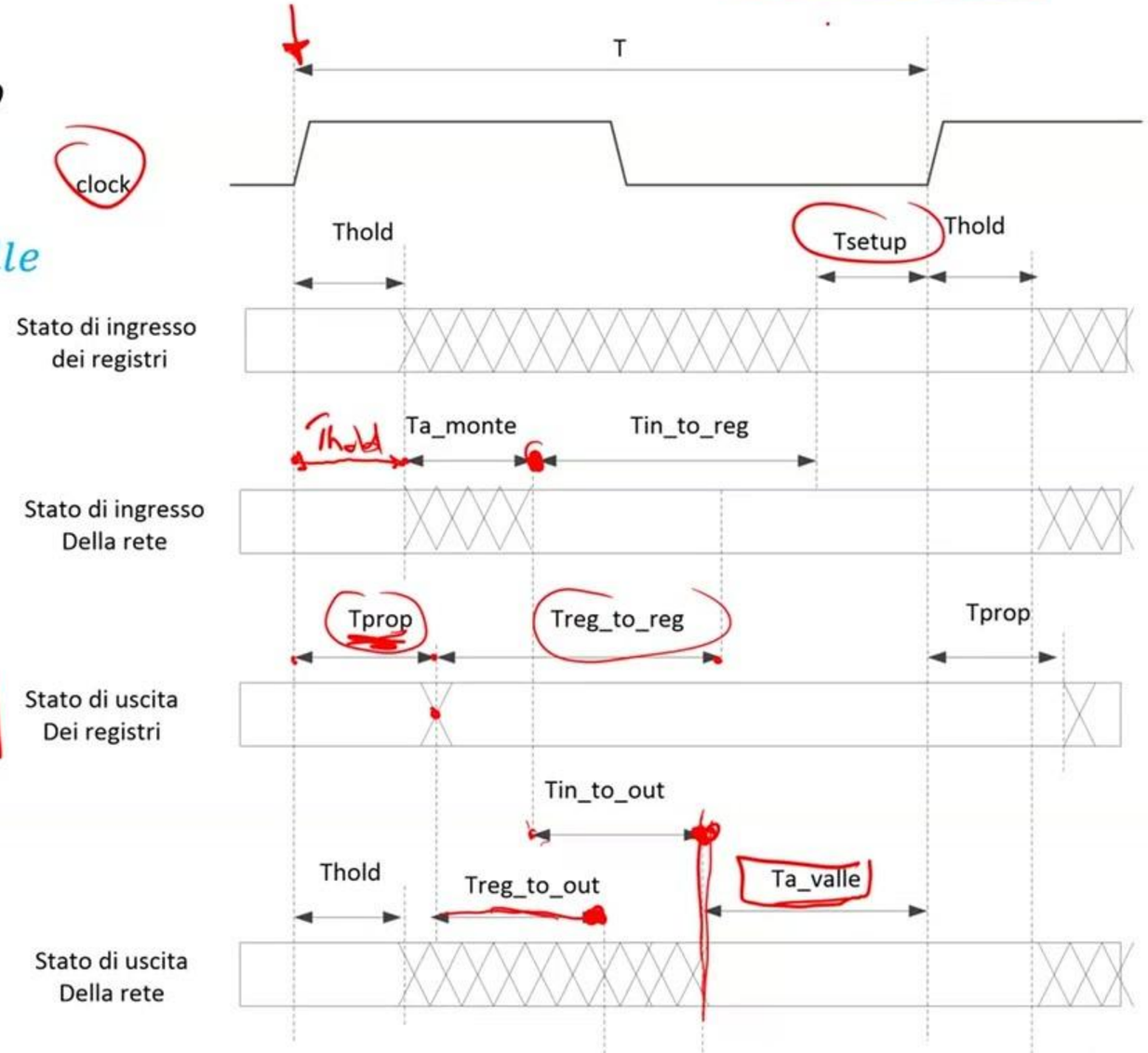




Disuguaglianze di temporizzazione (cont.) esempio

1) $T \geq T_{hold} + T_{a_monte} + T_{in_to_reg} + T_{setup}$
2) $T \geq T_{prop} + T_{reg_to_reg} + T_{setup}$
3) $T \geq T_{hold} + T_{a_monte} + T_{in_to_out} + T_{a_valle}$
4) $T \geq T_{prop} + T_{reg_to_out} + T_{a_valle}$





Disuguaglianze di temporizzazione (cont.)

- Ci sono alcune sottigliezze da tenere in conto:
- $\boldsymbol{T_{sfas}}$ il **massimo sfasamento tra due clock**
  - Stesso clock portato a elementi diversi: a qualche registro arriverà prima e a qualche altro dopo
  - Molto piccolo, trascurabile
- $\boldsymbol{T_{reg}}$: se un registro è formato da $W > 1$ bit, è impensabile che cambino **tutti contemporaneamente**.
  - tempo ulteriore da attendere dopo T_{prop} per essere certi che lo stato di uscita di un registro sia cambiato **per intero**.
  - Possiamo scrivere $T_{prop}{}' = T_{prop} + T_{reg}$ e dimenticarcene

Disuguaglianze di temporizzazione (cont.)

$$1)\ T \geq T_{hold} + T_{a_monte} + T_{\text{in_to_reg}} + T_{setup}$$

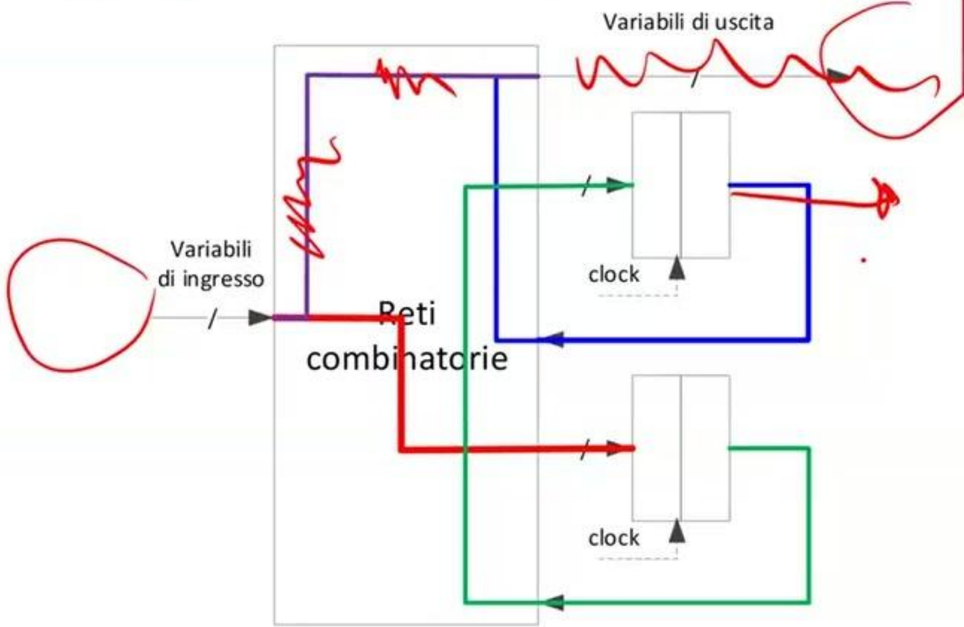
$$2)\ T \geq T_{prop} + T_{\text{reg_to_reg}} + T_{setup}$$

$$\cancel{3)\ T \geq T_{hold} + T_{a_monte} + T_{\text{in_to_out}} + T_{a_valle}}$$

$$4)\ T \geq T_{prop} + \boxed{\cancel{T_{\text{reg_to_out}}}} + T_{a_valle}$$

Tsetup Thold Tprop

Mealy



- Qual è la disuguaglianza più vincolante?
  - La n.3
- Potremmo voler **vietare qualche percorso**
  - ad esempio in->out
  - Reti su modello di **Moore:** non esiste via combinatoria tra ingresso ed uscita
- Uscite prese direttamente da registri
  - Nella 4) scompare il termine $T_{\text{reg_to_out}}$
  - Reti su modello di **Mealy ritardato**

Nota sul pilotaggio delle RSS

- **Lo stato di ingresso** di una RSS viene campionato dai registri **all'arrivo del clock**
- Cosa faccia lo stato di ingresso tra due clock **non ha alcuna importanza**, purché si stabilizzi in tempo
- Non ci interessa

  5 dock

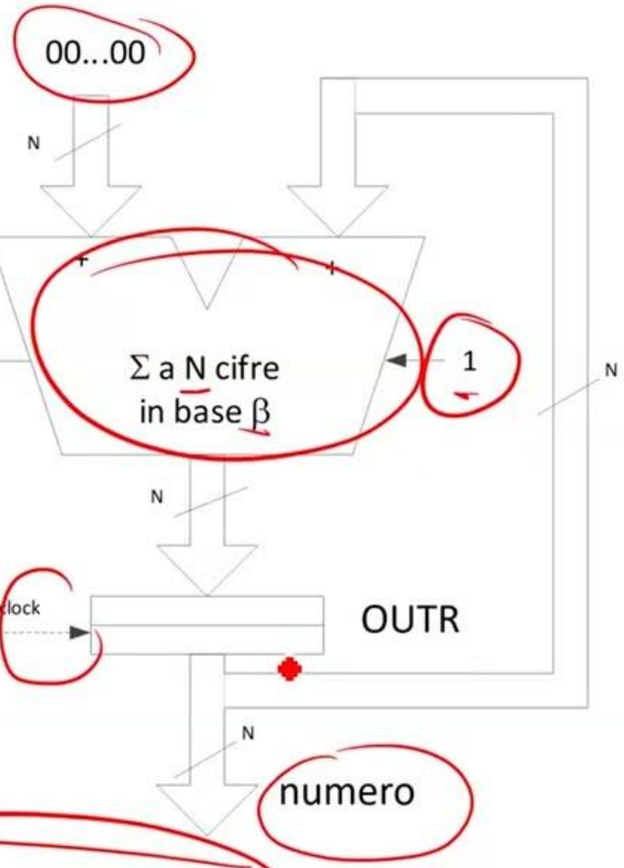
  - se cambia di n bit, con $n > 1$
  - se non cambia affatto, e rimane identico per due fronti di salita del clock
- In quest'ultimo caso sarà **comunque** interpretato come due stati di ingresso **differenti** (perché presentati ad istanti differenti)
- Le RSS si evolvono **all'arrivo del clock**, non quando **cambiano gli ingressi**

Contatori

- RSS il cui stato di uscita può essere visto come un **numero naturale ad n cifre in base β**
  - contatori a 2 cifre in base 10 BCD
  - contatori a n cifre in base 2
- Ad ogni clock, il contatore fa la seguente cosa:
  - **incrementa** l'uscita di uno (modulo β^n) - **contatore up**
  - **decrementa** l'uscita di uno (modulo β^n) - **contatore down**
  - **incrementa o decrementa** a seconda del valore di una **variabile di comando** (contatore up/down)

Contatore a N cifre in base β

- Modulo sommatore + registro
  - Supponiamo che N cifre in base β possano essere rappresentate su W bit

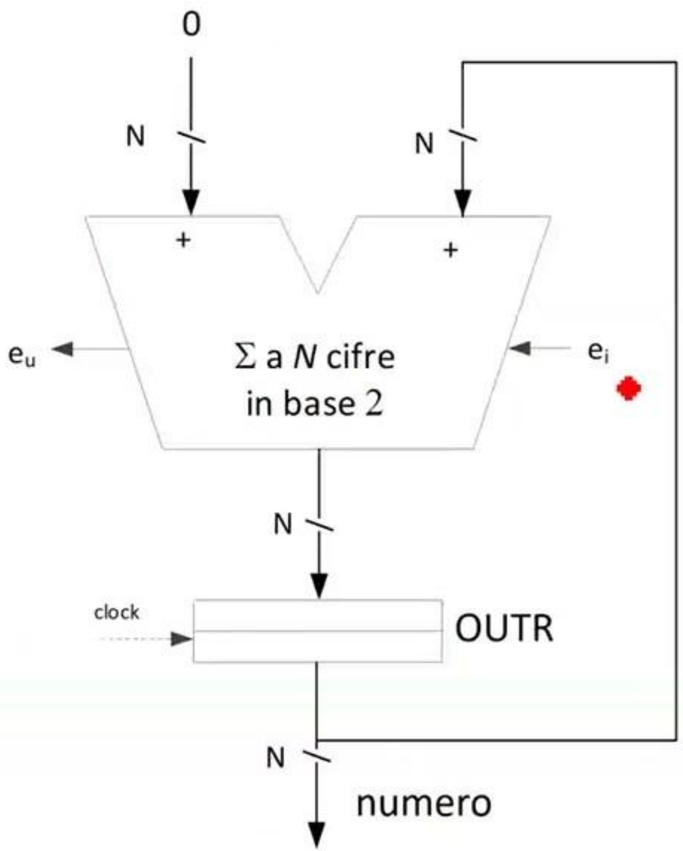



```
module ContatoreUp_Ncifre_BaseBeta(numero,clock,reset_);
  input clock,reset_;
  output [W-1:0] numero;
  reg [W-1:0] OUTR; assign numero=OUTR;
  always @(reset_==0) #1 OUTR<=0;
  always @(posedge clock) if (reset_==1) #3
        OUTR<= Inc_N_cifre_beta(OUTR);
endmodule
```

```
function [W-1:0] Inc_N_cifre_beta;
input [W-1:0] ingresso;
      casex(ingresso)
        <cod_0> : Inc_N_cifre_beta = <cod_1>;
        <cod_1> : Inc_N_cifre_beta = <cod_2>;
        …
        default : Inc_N_cifre_beta = 'BXXX…XXX;
      endcase
endfunction
```

Contatore a N cifre in base 2

- **Soltanto in questo caso** posso usare l'**operatore di somma + del Verilog**
  - «+» descrive una RC **sommatore ad N cifre in base 2**




```
module ContatoreUp_Ncifre_Base2(numero,clock,reset_);
   input clock,reset_;
   output [N-1:0] numero;

   reg [N-1:0] OUTR; assign numero=OUTR;

   always @(reset_==0) #1 OUTR<=0;
   always @(posedge clock) if (reset_==1) #3 OUTR<=numero+1;
endmodule
```

Contatori down

- Fin qui ho parlato di contatori **up**. Se voglio fare contatori **down**,
  - In base 2 scrivo "$-$" al posto di "$+$". In Verilog esiste l'operatore predefinito "$-$", che descrive un sottrattore in base 2 ad *N* cifre
  - in base β generica cambio la funzione "sommatore" scritta prima in un'altra funzione, che sostituisce la codifica **precedente**

```
function [W-1:0] Dec_N_cifre_beta;
input [W-1:0] ingresso;
      casex(ingresso)
        <cod_0> : Dec_N_cifre_beta = <cod_β^n − 1>;
        <cod_1> : Dec_N_cifre_beta = <cod_0>;
        …
        default : Dec_N_cifre_beta = 'BXXX…XXX;
      endcase
endfunction
```

Contatori - Ingresso di abilitazione

- Un contatore può essere dotato di un **ingresso di abilitazione e_i**
  - se e_i =1, all'arrivo del clock **conta** (up o down, a seconda di come lo faccio)
  - se e_i =0, all'arrivo del clock **conserva** l'ultimo valore

Modifiche alla descrizione in base 2

```
module ContatoreUp_Ncifre_Base2 (numero, clock, reset_, ei)
  input              clock, reset_, ei;
  […]
  always @(posedge clock) if (reset==1) #3
       OUTR<=OUTR+{'B00…00,ei};
endmodule
```

N-1

Blocco *always* per una base β generica

```
always @(posedge clock) if (reset==1) #3
    case(ei)
      0: OUTR<=OUTR;
      1: OUTR<=Inc_N_cifre_beta(OUTR);
    endcase
```

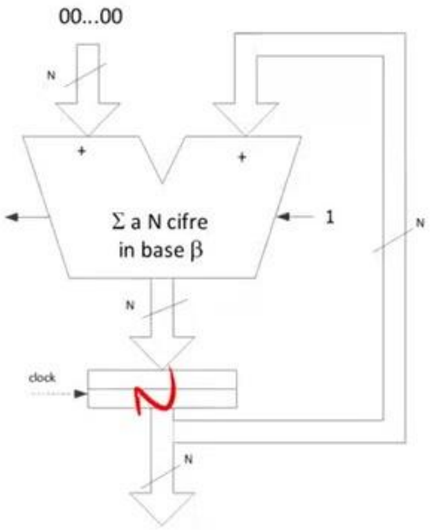
ei

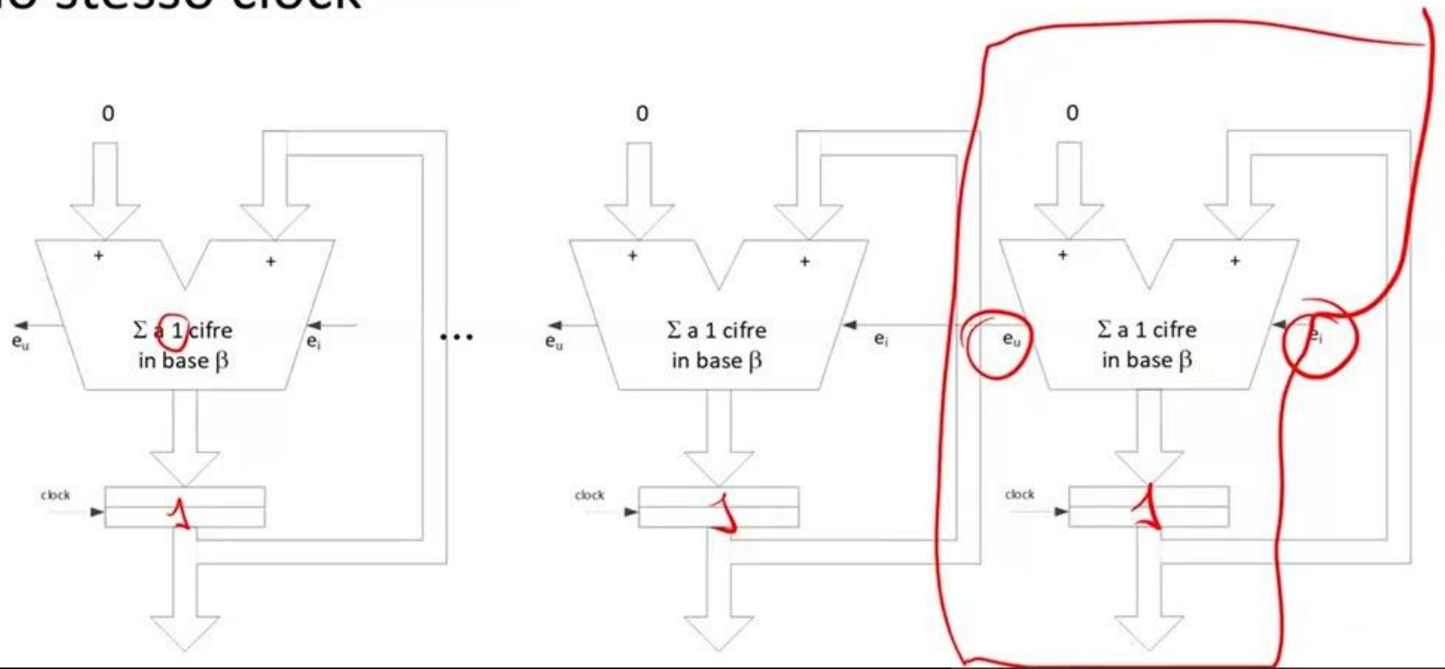
0 j

(R+1) j

Contatori – scomposizione in moduli

- In qualunque base, un contatore ad *N* cifre può sempre essere scomposto come una serie di contatori **ad una cifra**
  - **collegati mediante catena dei riporti** (*ripple carry*)
- In questo caso il registro è la giustapposizione di N registri
  - Ciascuno contiene una singola cifra
  - Tutti hanno lo stesso clock



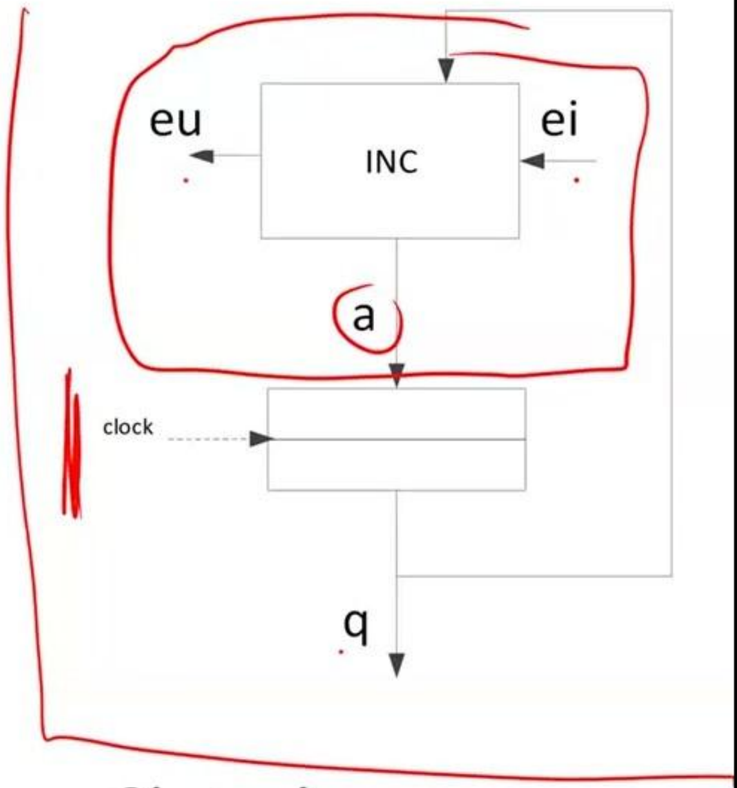


Elemento contatore in base 2

- Descrizione in Verilog

q	ei	a	eu
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

```
module Elemento_Contatore_Base_2(eu,q,ei,clock,reset_);
  input clock,reset_;
  input ei;
  output eu,q;
  reg OUTR; assign q=OUTR;
  wire a; // variabile di uscita dell'incrementatore
  assign {a,eu}= ({q,ei}=='B00) ?'B00:
                 ({q,ei}=='B01) ?'B10:
                 ({q,ei}=='B10) ?'B10:
                 /*({q,ei}=='B11)*/'B01;
  always @(reset_==0) #1 OUTR<=0;
  always @(posedge clock) if (reset_==1) #3 OUTR<=a;
endmodule
```



Sintesi

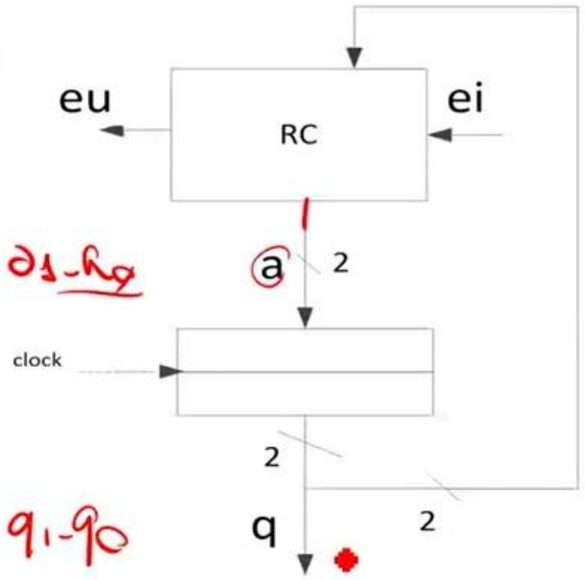
```
assign a=q^ei;
assign eu=(q&ei);
```

Elemento contatore in base 3

- 2 bit per codificare una cifra: **0='B00, 1='B01, 2='B10**
- Descrizione

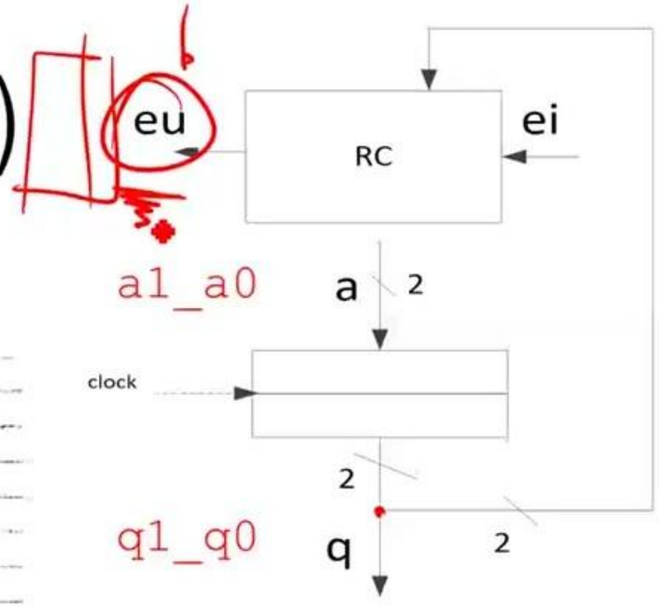
```
module Elemento_Contatore_Base_3(eu,q1_q0,ei,clock,reset_);
   input clock,reset_;
   input ei;
   output eu;
   output [1:0] q1_q0;
   reg [1:0] OUTR; assign q1_q0=OUTR;
   wire [1:0] a1_a0; // variabile di uscita dell'incrementatore
   assign {a1_a0,eu}= ({q1_q0,ei}=='B000)?'B000:
                      ({q1_q0,ei}=='B010)?'B010:
                      ({q1_q0,ei}=='B100)?'B100:
                      ({q1_q0,ei}=='B001)?'B010:
                      ({q1_q0,ei}=='B011)?'B100:
                      ({q1_q0,ei}=='B101)?'B001:
                      /* default */ 'BXXX;
   always @(reset_==0) #1 OUTR<='B00;
   always @(posedge clock) if (reset_==1) #3 OUTR<=a1_a0;
endmodule
```

funzione(q1-q0, ei)



q_1	q_0	e_i	a_1	a_0	e_u
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	1	0	0
1	0	0	1	0	0
1	0	1	0	0	1
others			-	-	-

Elemento contatore in base 3 (cont.)



- Sintesi

e_i \ q_1q_0	00	01	11	10
0	0	0	-	1
1	0	1	-	0

a_1

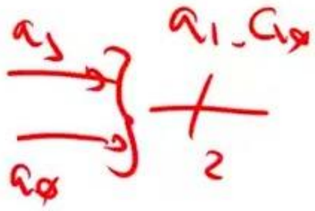
e_i \ q_1q_0	00	01	11	10
0	0	1	-	0
1	1	0	-	0

a_0

e_i \ q_1q_0	00	01	11	10
0	0	0	-	0
1	0	0	-	1

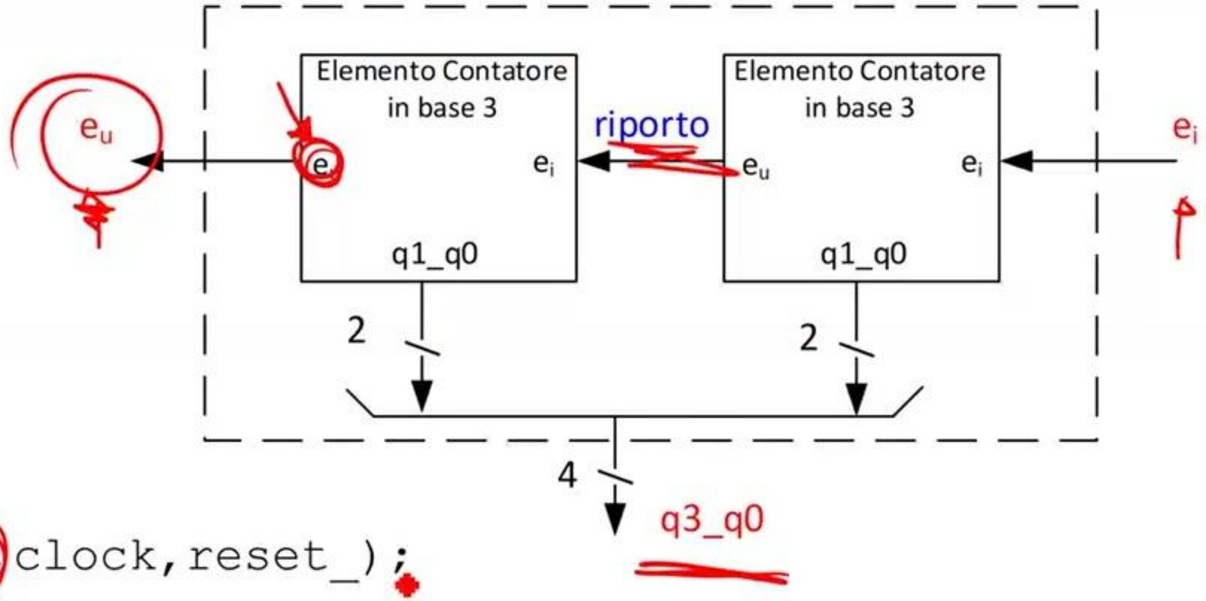
e_u

$$a_1 = q_1 \cdot \overline{e_i} + q_0 \cdot e_i \qquad a_0 = q_0 \cdot \overline{e_i} + \overline{q_1} \cdot \overline{q_0} \cdot e_i \qquad e_u = q_1 \cdot e_i$$




```
wire [1:0] a1_a0;        // variabile di uscita dell'incrementatore
wire a1, a0;             // variabili a un bit che compongono a1_a0
assign a1_a0={a1,a0};
assign eu=q1_q0[1]&ei;
assign a1=(q1_q0[0]&ei)|(q1_q0[1]&~ei);
assign a0=(~q1_q0[1]&~q1_q0[0]&ei)|(q1_q0[0]&~ei);
```

Cont. a 2 cifre in base 3




```
module Elemento_Contatore_Base_3(eu,q1_q0,ei,clock,reset_);
```

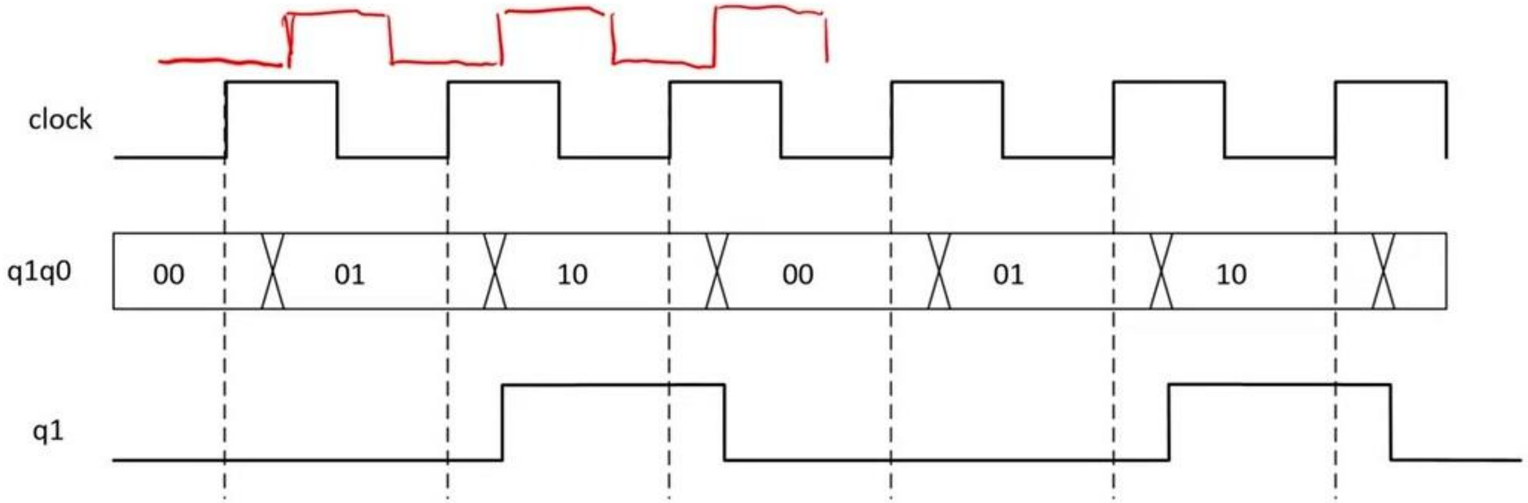
- Connetto insieme due elementi contatore

```
module Contatore_Base_3_2Cifre(eu,q3_q0,ei,clock,reset_);
   input clock,reset_;
   input ei;
   output eu;
   output [3:0] q3_q0;
     wire riporto;

     Elemento_Contatore_Base_3  LSD(riporto,q3_q0[1:0],ei,clock,reset_),
                                MSD(eu,q3_qo[3:2],riporto,clock,reset_);
 endmodule
```

Contatori e divisione in frequenza

- Possono essere usati per **dividere la frequenza di clock** per un dato valore
  - Ad esempio, posso usare il **MSB** del contatore in base 3 per ottenere un clock che va 3 volte più lento



  - L'MSB di un contatore a N cifre in base 2 che riceve clock a periodo T è un clock a periodo $2^N \cdot T$
  - Usare **solo** uscite dei registri per fare i clock (mai uscite combinatorie)

Modello di Moore (cont.)

7) Si adegua alla seguente **legge di temporizzazione**:

"Dato S, stato interno marcato ad un certo istante, e dato X ingresso ad un certo istante immediatamente **precedente l'arrivo di un segnale di sincronizzazione**,

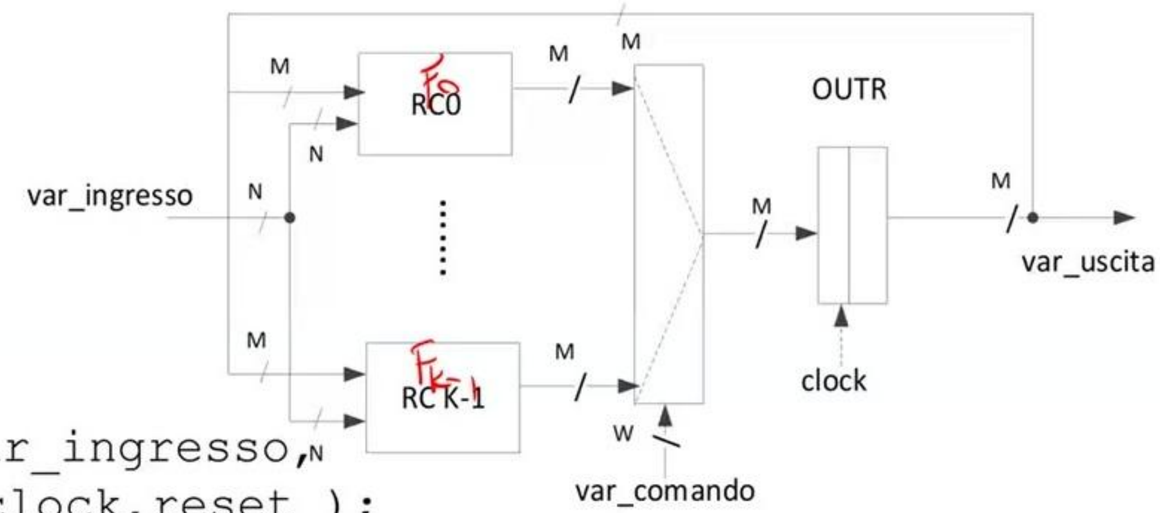
a) individuare il **nuovo stato interno da marcare** $\boldsymbol{S' = A(X, S)}$;

b) attendere $\boldsymbol{T_{prop}}$ dopo l'arrivo del segnale di sincronizzazione;

c) promuovere $\boldsymbol{S'}$ al rango di **stato interno marcato.**

E, inoltre, **individuare <u>continuamente</u>** $\boldsymbol{Z = B(S)}$ e presentarlo in uscita

Registro multifunzionale

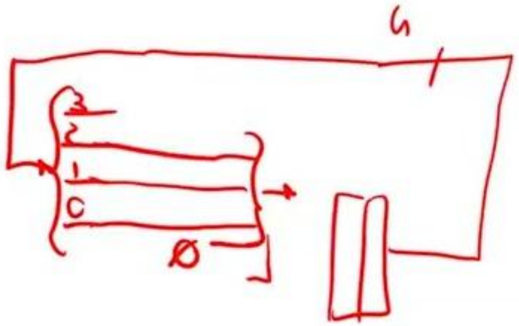


/reset

```
module Registro_Multifunzionale(var_uscita,var_ingresso,
                                var_comando,clock,reset_);
  input clock,reset_;
  input [N-1:0] var_ingresso;
  input [W-1:0] var_comando;
  output [M-1:0] var_uscita;
  reg [M-1:0] OUTR; assign var_uscita=OUTR;
  always @(reset_==0) #1 OUTR<=contenuto_iniziale;
  always @(posedge clock) if (reset_==1) #3
   casex(var_comando)
    0 : OUTR<=F0(var_ingresso,OUTR);
    ...
    ...
    K-1: OUTR<=FK-1(var_ingresso,OUTR);
    endcase
endmodule
```

Esempio

OUTR <= {1'B0, OUTR[3:1]}



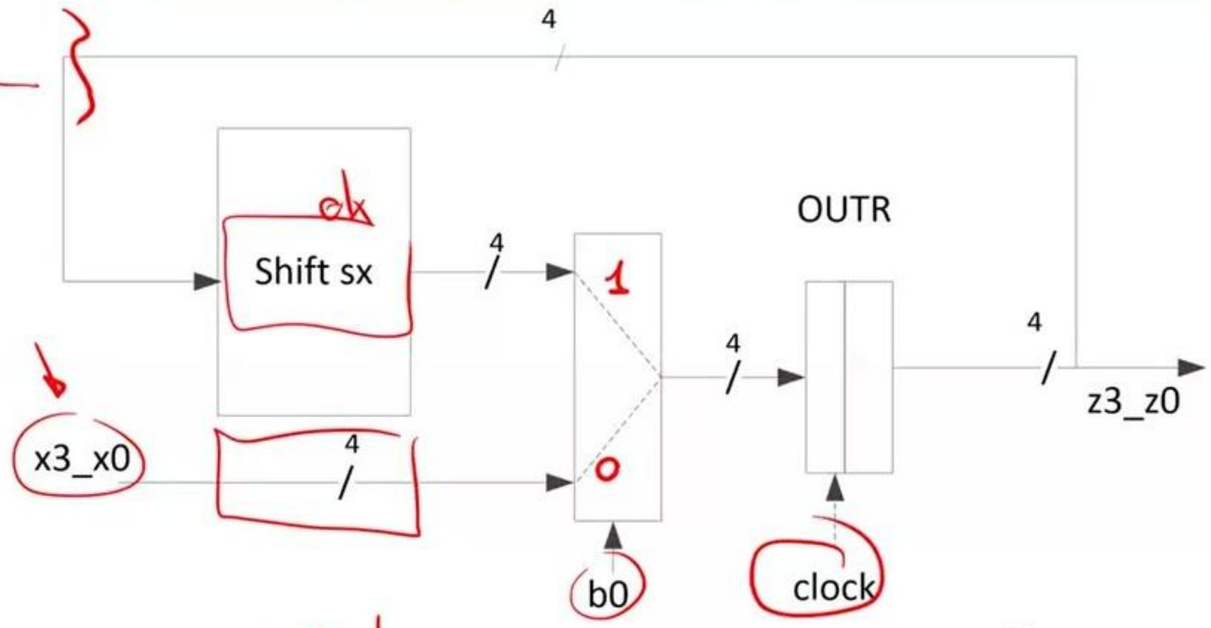
- Registro (**bifunzionale**) di caricamento/traslazione sx.
- In base ad una variabile di comando **b0**
  - **carica** un nuovo valore, cioè memorizza M bit ex novo, oppure
  - **trasla** a sinistra il proprio contenuto.

```
module Registro_CaricaParallelo_TraslaSinistro(z3_z0,x3_x0,b0,clock,reset_);
  input clock,reset_;
  input [3:0] x3_x0;
  input b0;
  output [3:0] z3_z0;
  reg [3:0] OUTR; assign z3_z0=OUTR;
  always @(reset_==0) #1 OUTR<='B0000;
  always @(posedge clock) if (reset_==1) #3
    casex(b0)
    'B0: OUTR<=x3_x0;
    'B1: OUTR<={OUTR[2:0],1'B0};
    endcase
endmodule
```

OUTR <= {OUTR[3], ...}

OUTR [3:2] <=

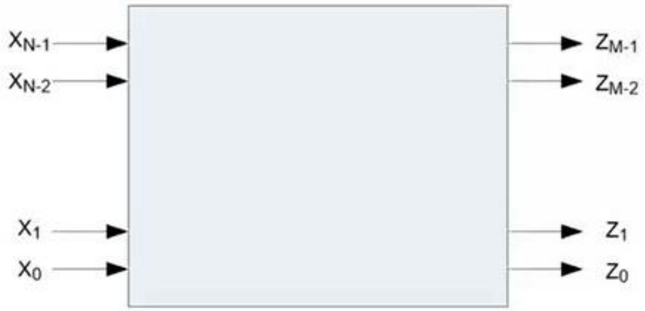
begin OUTR[3]<=OUTR[2]; ... ; end



Modelli di reti sequenziali sincronizzate

- Ne esistono tre
  - Modello di **Moore**
  - Modello di **Mealy**
  - Modello di **Mealy ritardato**

Modello di Moore



- Una **RSS di Moore** è rappresentata come segue
  1) un insieme di N variabili logiche di ingresso.
  2) un insieme di M variabili logiche di uscita.
  3) Un **meccanismo di marcatura**, che ad ogni istante marca uno **stato interno presente**, scelto tra un insieme **finito** di K stati interni $\boldsymbol{S} \equiv \{S_0, \ldots, S_{K-1}\}$
  4) Una **legge di evoluzione nel tempo** $A: \boldsymbol{X} \times \boldsymbol{S} \rightarrow \boldsymbol{S}$, che mappa quindi una coppia (stato di ingresso, stato interno) in un nuovo stato interno.
  5) Una **legge di evoluzione nel tempo** $B: \boldsymbol{S} \rightarrow \boldsymbol{Z}$, che decide lo stato di uscita basandosi sullo stato interno.
     - **Nota**: tale legge **non è** più generale, del tipo $B: \boldsymbol{X} \times \boldsymbol{S} \rightarrow \boldsymbol{Z}$. Se fosse più generale saremmo **fuori dal modello**).
  6) La rete riceve **segnali di sincronizzazione**, come transizioni da 0 a 1 del segnale di clock

Modello di Moore (cont.)

7) Si adegua alla seguente **legge di temporizzazione**:

"Dato S, stato interno marcato ad un certo istante, e dato X ingresso ad un certo istante immediatamente **precedente l'arrivo di un segnale di sincronizzazione**,

a) individuare il **nuovo stato interno da marcare** $\boldsymbol{S' = A(X, S)}$;

b) attendere $\boldsymbol{T_{prop}}$ dopo l'arrivo del segnale di sincronizzazione;

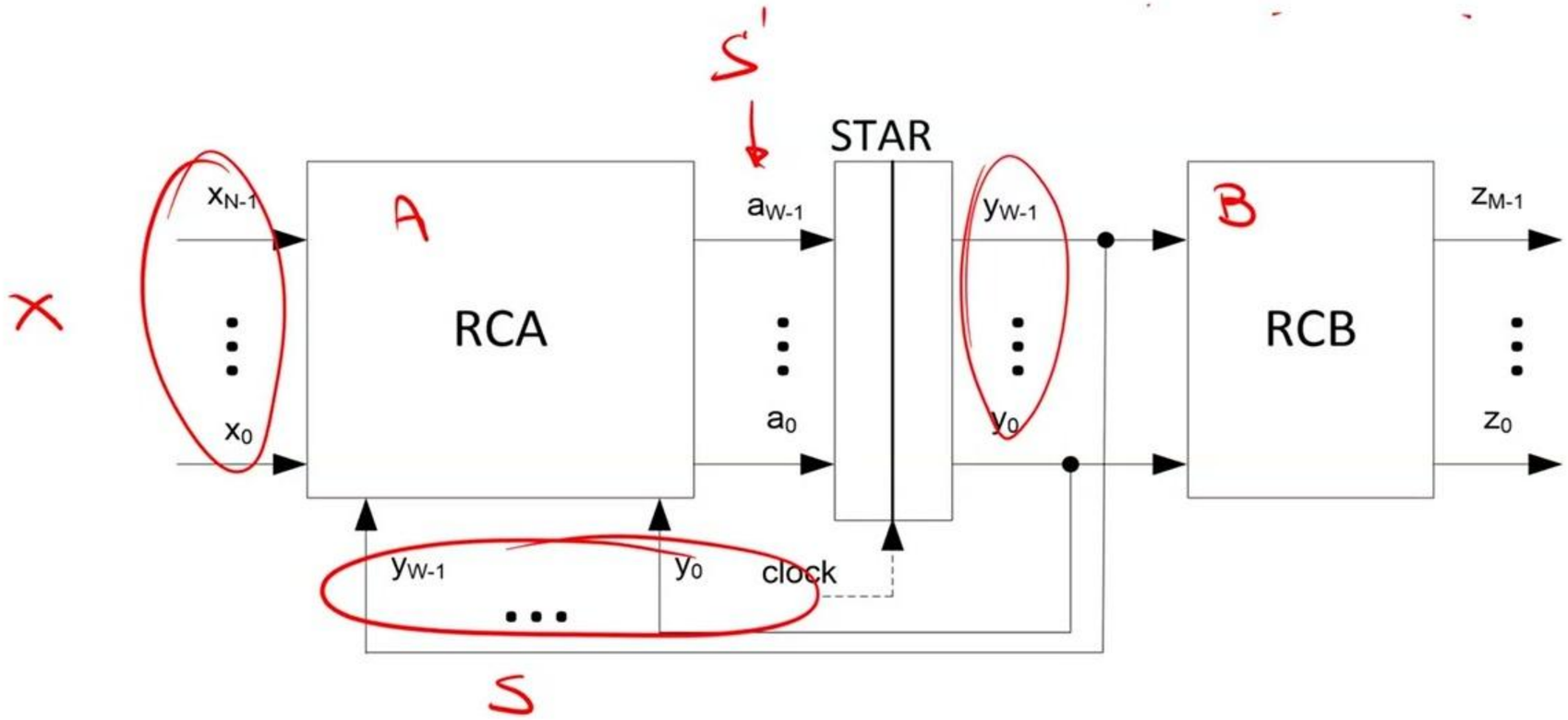
c) promuovere $\boldsymbol{S'}$ al rango di **stato interno marcato.**

E, inoltre, **individuare continuamente** $\boldsymbol{Z = B(S)}$ e presentarlo in uscita

Struttura di una RSS di Moore

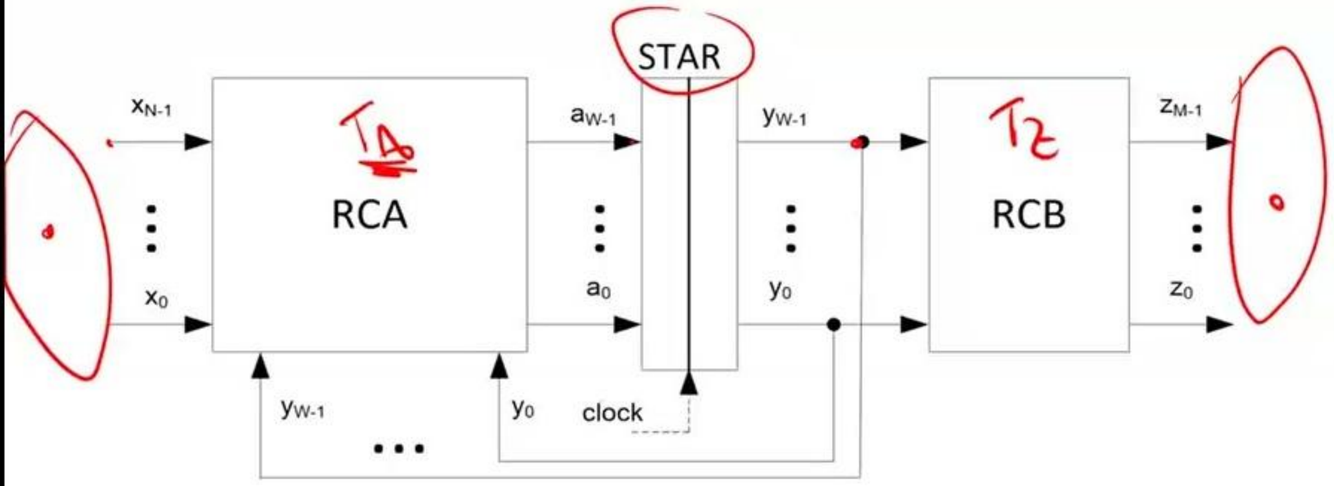
- **STAR: STAtus Register**, memorizza lo stato interno presente (marcato)
  - Batteria di D-FF, **non trasparenti**

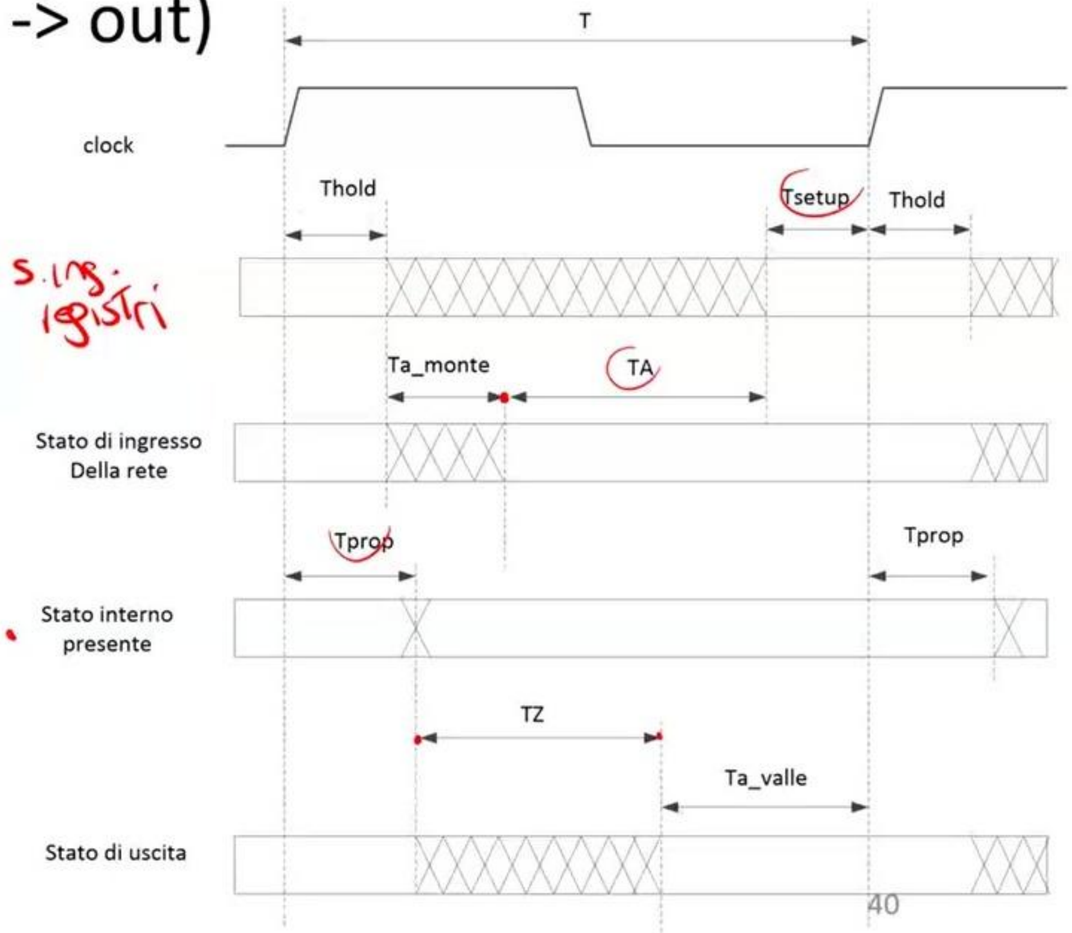
$S_0 \; S_1 \; \ldots \; S_{k-1}$



Vincoli di temporizzazione per una RSS di Moore

- $T \geq T_{hold} + T_{a_monte} + T_A + T_{setup}$ (in -> STAR)
- $T \geq T_{prop} + T_A + T_{setup}$ (STAR -> STAR)
- $T \geq T_{prop} + T_Z + T_{a_valle}$ (STAR -> out)





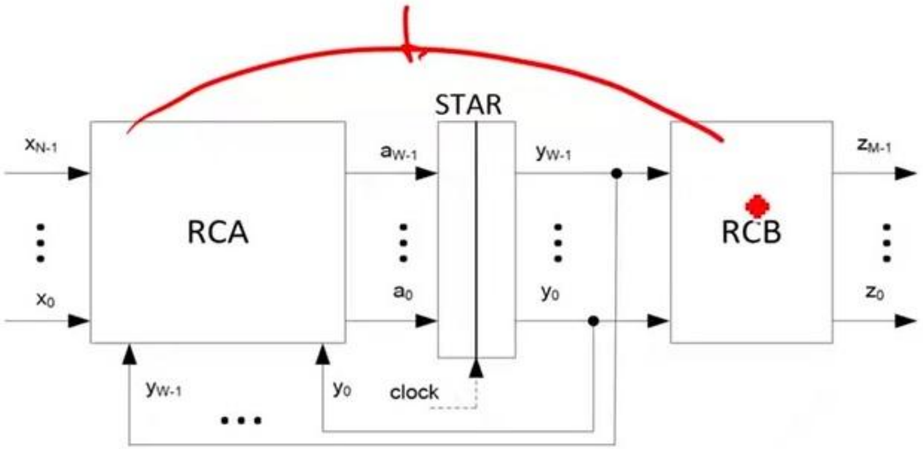
$z_i \propto x_{i-1}, x_{i-2}, x_{i-3} \ldots x_0, S_0$

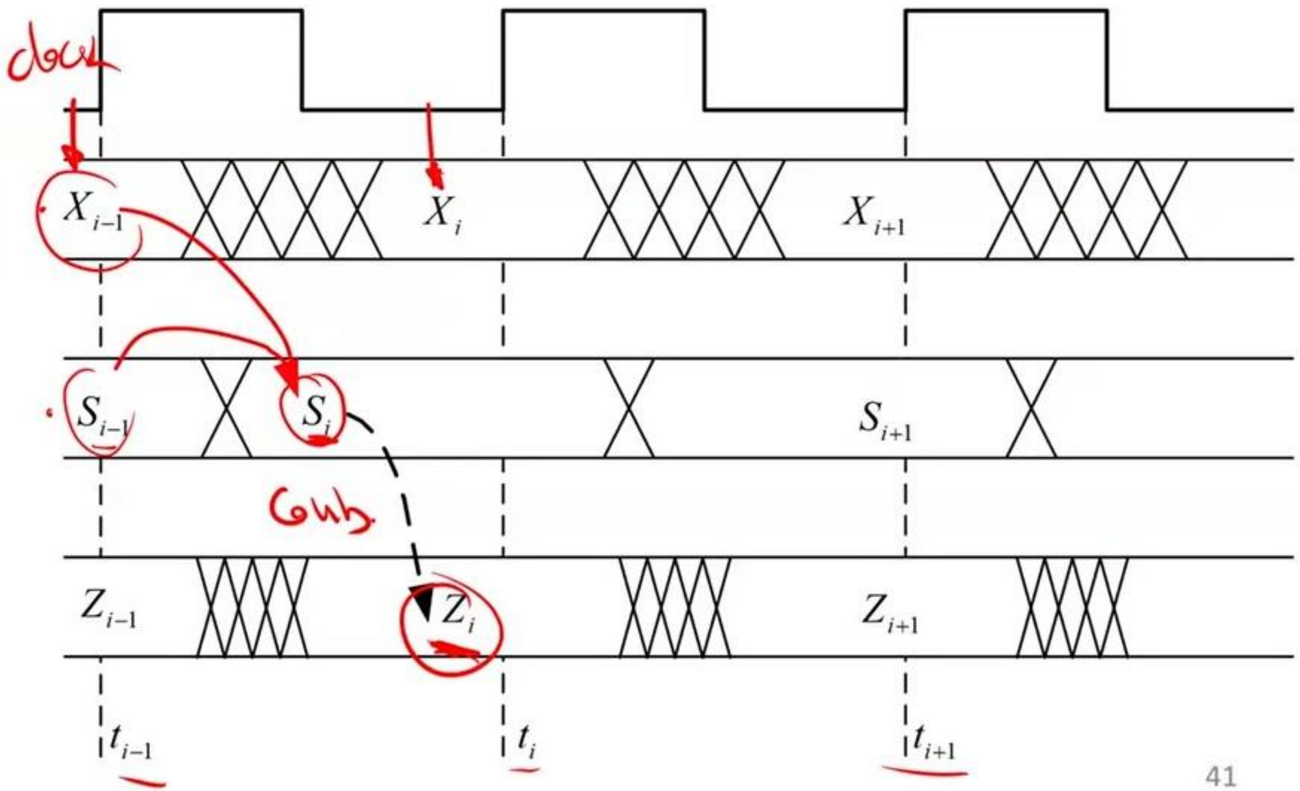
Evoluzione nel tempo di una RSS di Moore

perde un clock

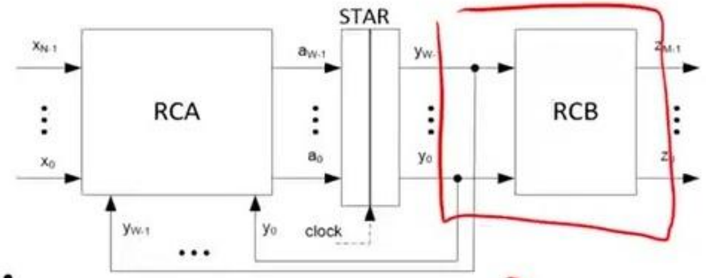
- $t_i, i \geq 0$, **istanti di sincronizzazione**,
- $X[t_i], S[t_i], Z[t_i]$ stati di **ingresso, interno e di uscita** all'istante t_i

$$S[t_{i+1}] = A(X[t_i], S[t_i])$$
$$Z[t_i] = B(S[t_i])$$





Descrizione di una RSS di Moore




```
module Rete_di_Moore(zM-1,...,z0,xN-1,...,x0,clock,reset_);
  input clock,reset_;
  input xN-1,...,x0;
  output zM-1,...,z0;
  reg [W-1:0] STAR; [parameter S0=codifica0,...,SK-1=codificaK-1;]
  assign {zM-1,...,z0} = (STAR==S0) ?BS0 :
                         (STAR==S1) ?BS1 :
                         ...
                         /*(STAR==SK-1)*/ BSK-1;
  always @(reset_==0) #1 STAR<=stato_interno_iniziale;
  always @(posedge clock) if (reset_==1) #3
   casex(STAR)
     S0 : STAR<=AS0(xN-1,...,x0);
     S1 : STAR<=AS1(xN-1,...,x0);
     ...
     SK-1: STAR<=ASK-1(xN-1,...,x0);
   endcase
endmodule
```

(x==0) ? q0 :
(x==1) ? q1 :
// (x==2) ? q2

STAR==SK-2 ? BSK-2 :

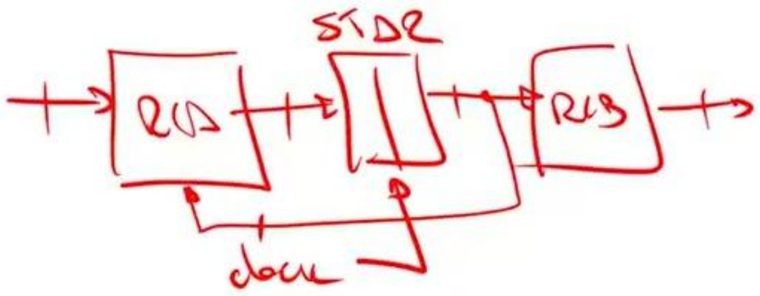
Legge B

A: X × S → S

S0 AS0 X → S
S1 AS1 X → S
S2

Legge A

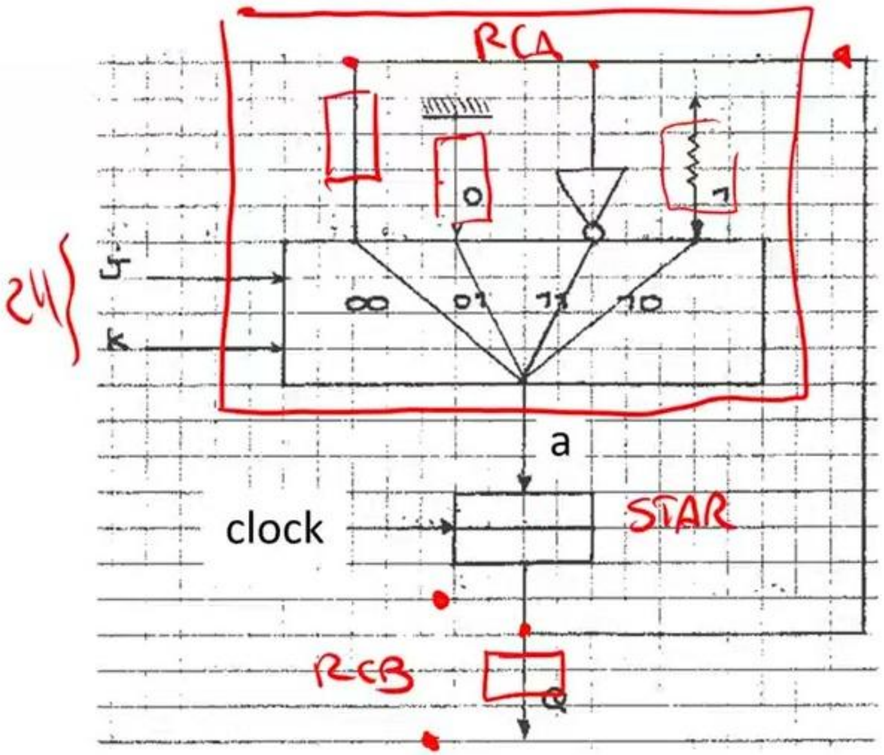
Esempio: Flip-Flop JK



- RSS con due ingressi ed un'uscita che, all'arrivo del clock, valuta i suoi due ingressi ***j* e *k***, e si comporta come segue:

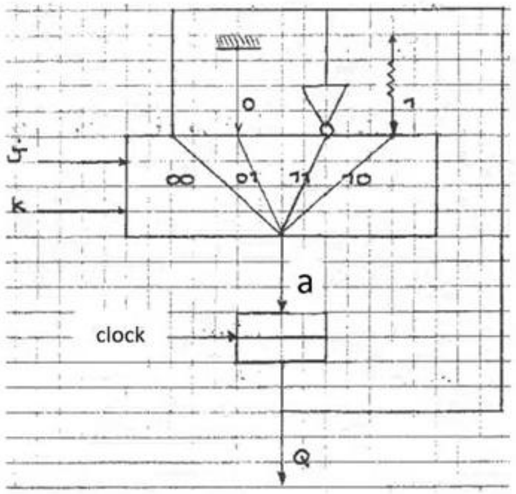
jk	**Azione in uscita**
00	Conserva
10	Setta
01	Resetta
11	Commuta

descr
↓
sintesi



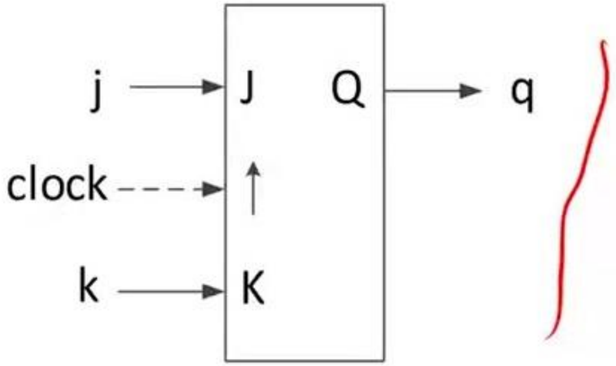
Un modo (non ottimizzato) di vedere questa rete è come **registro multifunzionale ad un bit**.

Esempio: Flip-Flop JK (cont.)



- Posso dare una **tabella di applicazione,** simile a quella del Latch SR

jk	Azione in uscita
00	Conserva
10	Setta
01	Resetta
11	Commuta

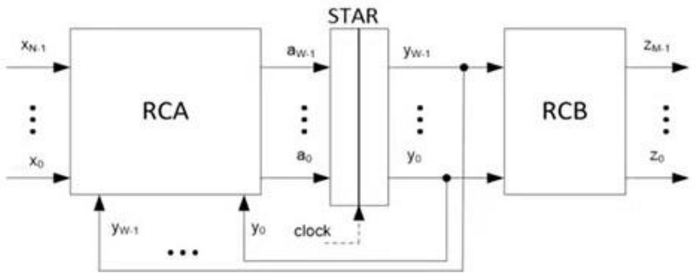


al prossimo clock

q	q'	j	k
0	0	0	-
0	1	1	-
1	0	-	1
1	1	-	0

- Il FF JK è una rete di Moore. Infatti, non c'è una via combinatoria dagli ingressi all'uscita, che è invece funzione soltanto del contenuto del registro.

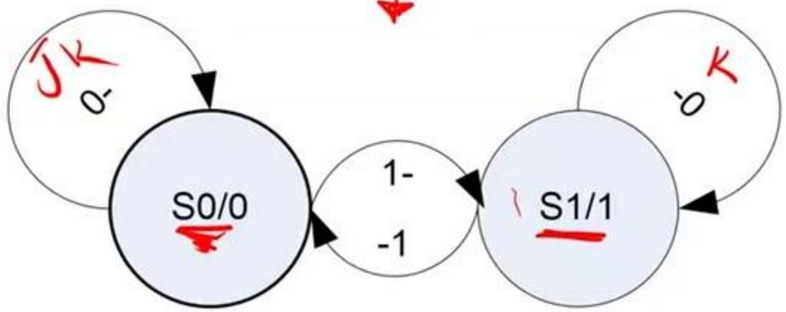
Esempio: Flip-Flop JK (cont.)



- Il FF JK serve a **memorizzare un bit**,
  - **due stati interni**, S_0 e S_1, nei quali memorizzo rispettivamente 0 e 1
  - Codificherò questi stati interni con **una variabile di stato** che varrà 0 e 1
  - **RCB è un cortocircuito**

- Posso **descrivere** il FF-JK usando tabelle e grafi di flusso, o Verilog

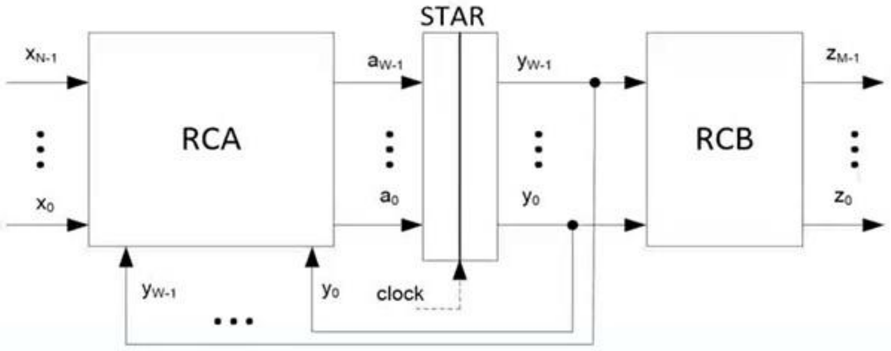
SR d.l.



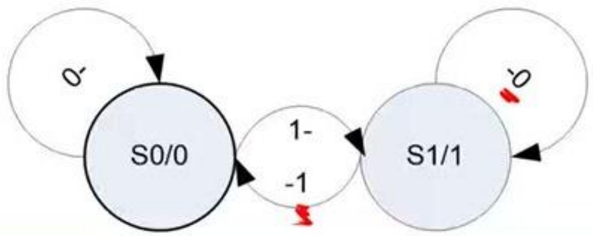
RSA

	jk → 00	01	11	10	q
S0	S0	S0	S1	S1	0
S1	S1	S0	S0	S1	1

Esempio: Flip-Flop JK (cont.)



- Descrizione in Verilog



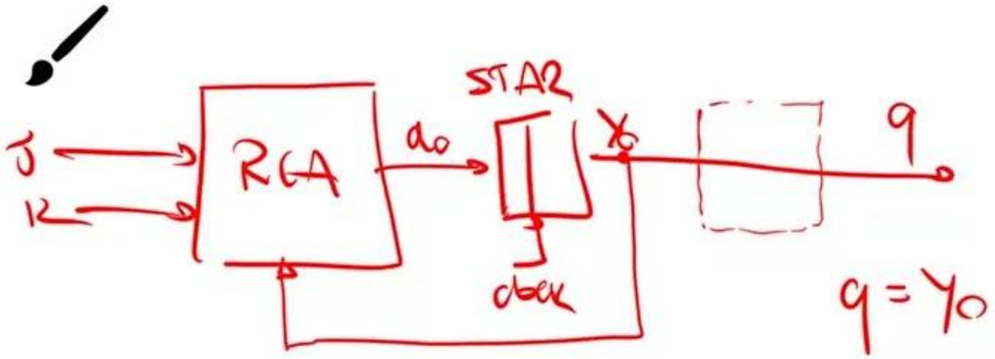

```
module FlipFlop_JK(q,j,k,clock,reset_);
  input clock,reset_;
  input j,k;
  output q;
  reg STAR; parameter S0='B0,S1='B1;
  assign q=(STAR==S0)?0:1;
  always @(reset_==0) #1 STAR<=S0;
  always @(posedge clock) if (reset_==1) #3
    casex(STAR)
     S0: STAR<=(j==0)?S0:S1;
     S1: STAR<=(k==0)?S1:S0;
    endcase
endmodule
```

codifica stati interni

B

A

Esempio: Flip-Flop JK (cont.)



- Dalla tabella di flusso si passa velocemente alla sintesi di RCA e RCB
- Basta scegliere una **codifica** per gli stati interni ($S_0 = 'B0, S_1 = 'B1$)

	jk 00	01	11	10	q
S0	S0	S0	S1	S1	0
S1	S1	S0	S0	S1	1

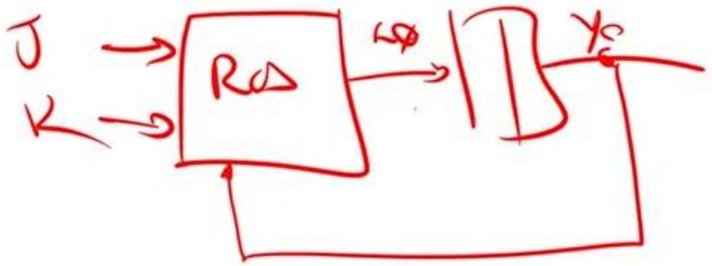
t.t → mk

y0 \ jk	00	01	11	10
0	0	0	1	1
1	1	0	0	1

a_0

$$a_0 = j \cdot \overline{y_0} + \bar{k} \cdot y_0, \quad q = y_0$$

Esempio: FF-JK (cont.)



- Sintesi scritta in Verilog

DESCR.

SINTESI

$$a_0 = j \cdot \overline{y_0} + \bar{k} \cdot y_0, \quad q = y_0$$

```
module FlipFlop_JK(q,j,k,clock,reset_);
  input clock,reset_;
  input j,k;
  output q;
  reg STAR; parameter S0='B0,S1='B1;
  assign q=(STAR==S0)?0:1;

  always @(reset_==0) #1 STAR<=S0;
  always @(posedge clock) if (reset_==1) #3
    casex(STAR)
      S0: STAR<=(j==0)?S0:S1;
      S1: STAR<=(k==0)?S1:S0;
    endcase
endmodule
```

```
wire a0;
assign a0=j&~STAR&~k;
assign q=STAR;
```

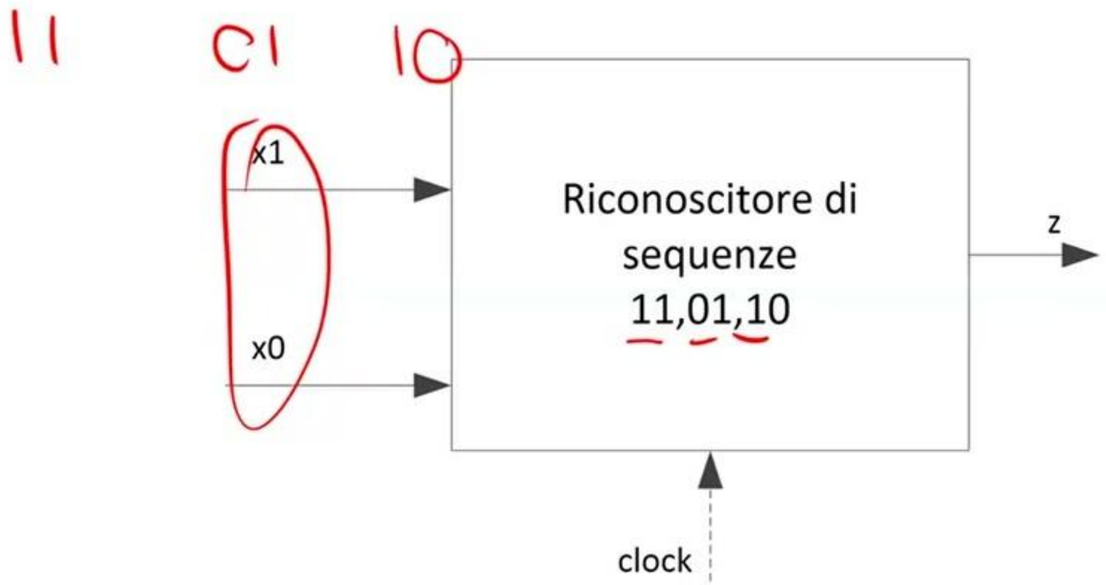
#1

+ STAR

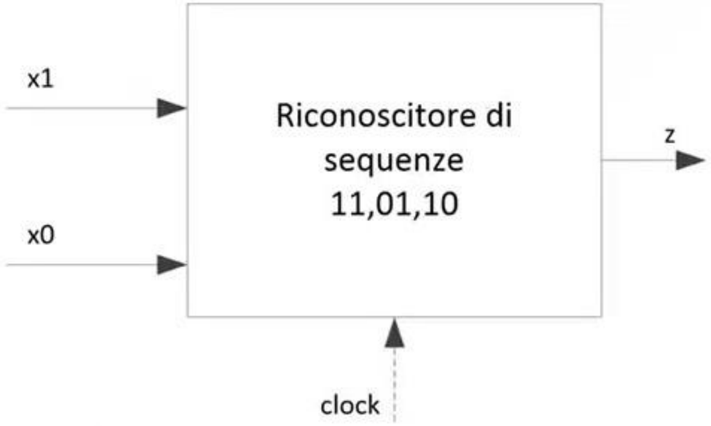
```
STAR<=a0;
```

Esempio: riconoscitore di sequenze

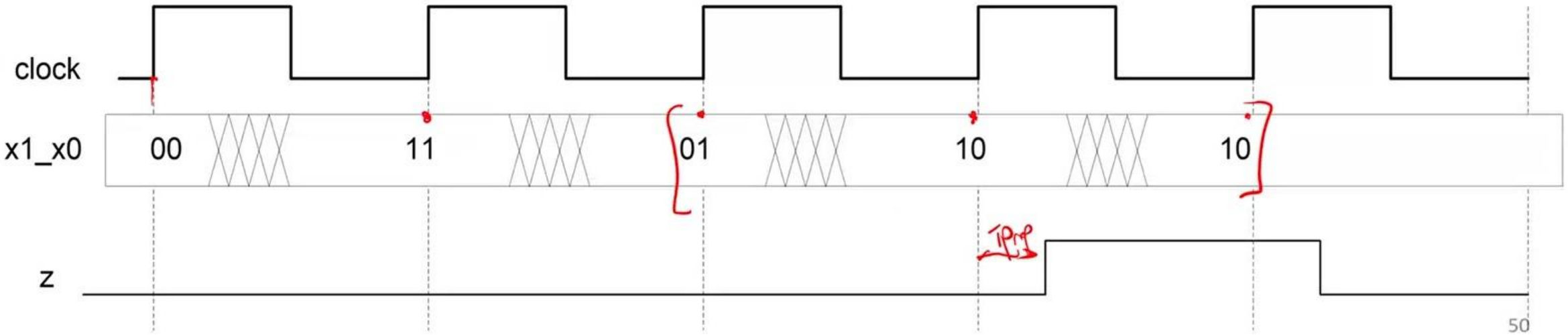
- Rete che ha N ingressi (in questo caso $N = 2$, sono sequenze di 2 bit), ed un'uscita. A parole, l'evoluzione della rete è la seguente:
- **"l'uscita va ad 1 soltanto quando si è presentata la sequenza degli stati di ingresso voluta (11,01,10), ed è a zero altrimenti".**



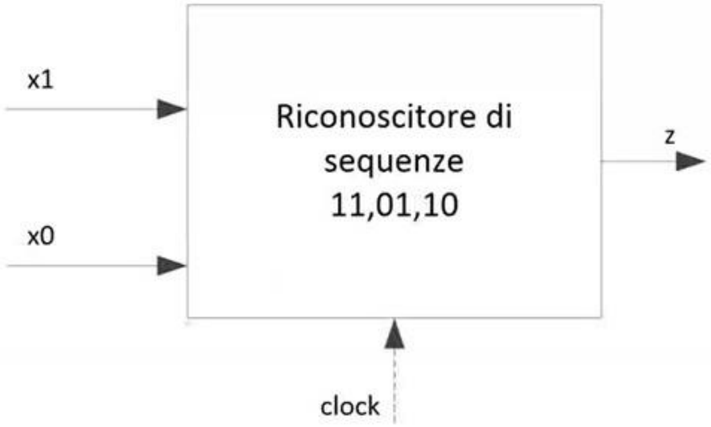
Riconoscitore di sequenze (cont.)



- Rete che sblocca la serratura di una cassaforte in cui la combinazione è data da una sequenza di parole di N bit, che si devono presentare **in tre clock consecutivi.**
- Se un valore permane per più di un ciclo di clock, la sequenza è **diversa**.

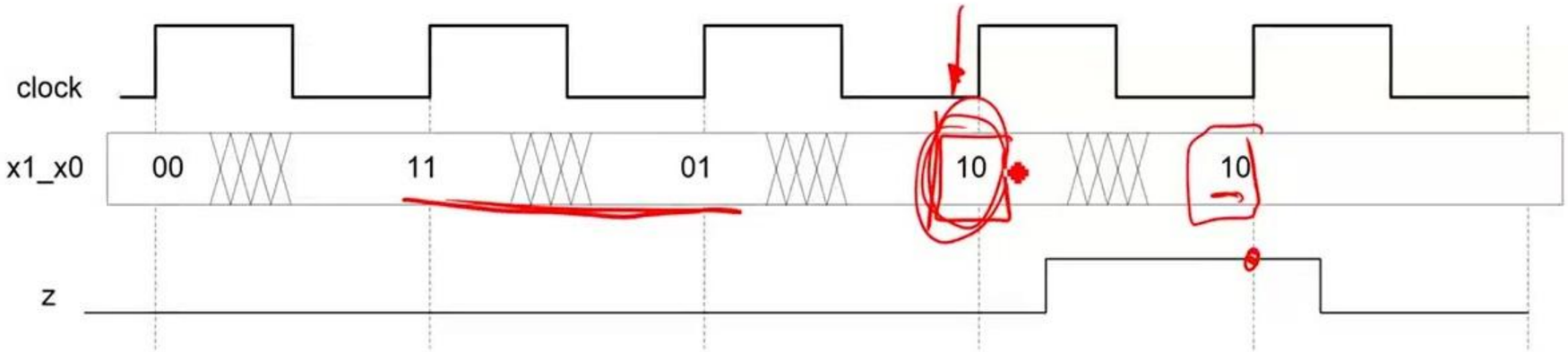


Riconoscitore di sequenze (cont.)



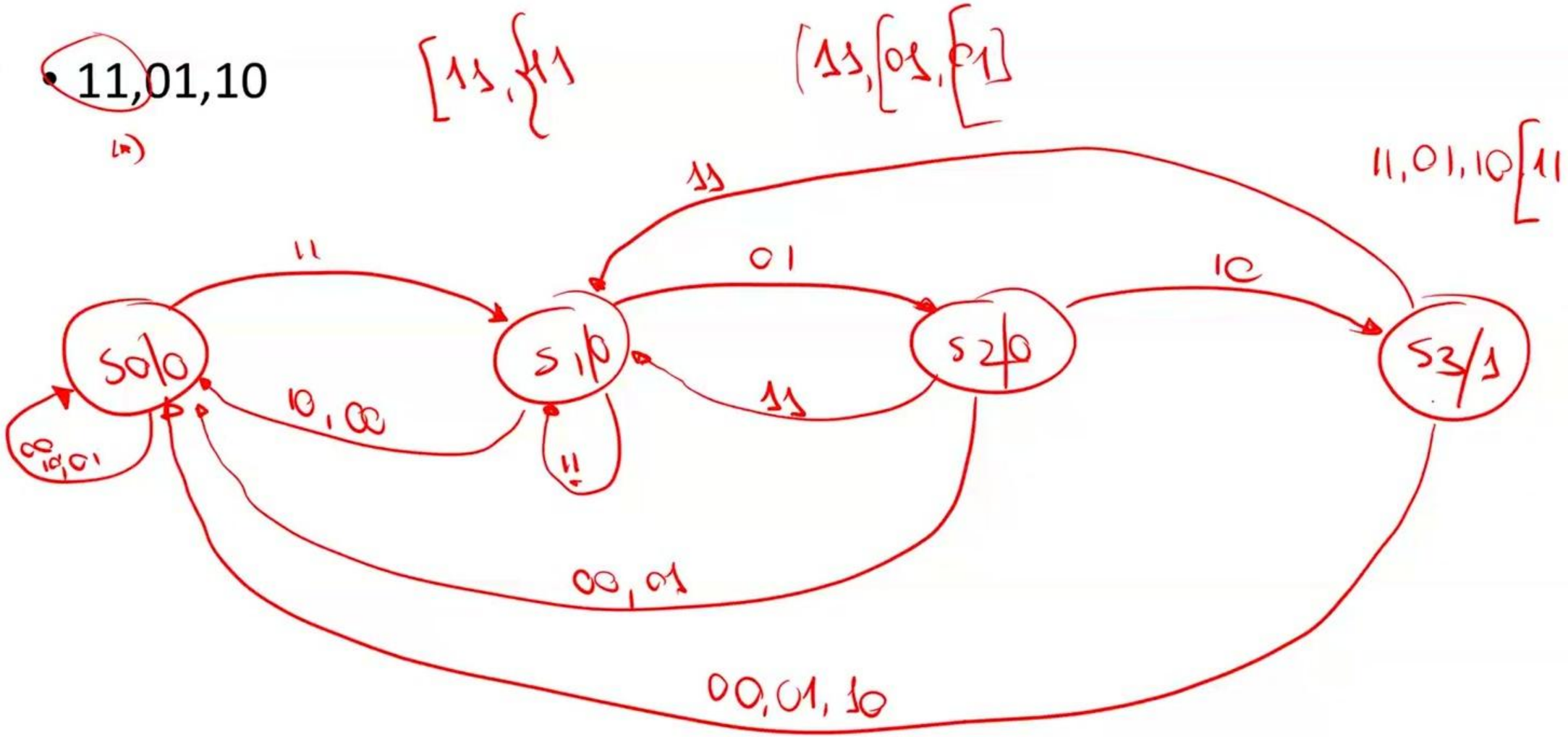
K+1

- Rete **con memoria**: deve ricordare una data sequenza di stati di ingresso.
- In particolare, va memorizzato **il numero di passi di sequenza corretti consecutivi** visti finora
  - quindi sono richiesti $\boldsymbol{K+1}$ **stati interni per una sequenza di** $\boldsymbol{K}$ **stati di ingresso da riconoscere** (ciascuno stato interno corrispondente a $0, 1, \ldots, K$ passi riconosciuti).
  - L'ultimo di questi stati interni avrà un'uscita ad 1, gli altri a 0.



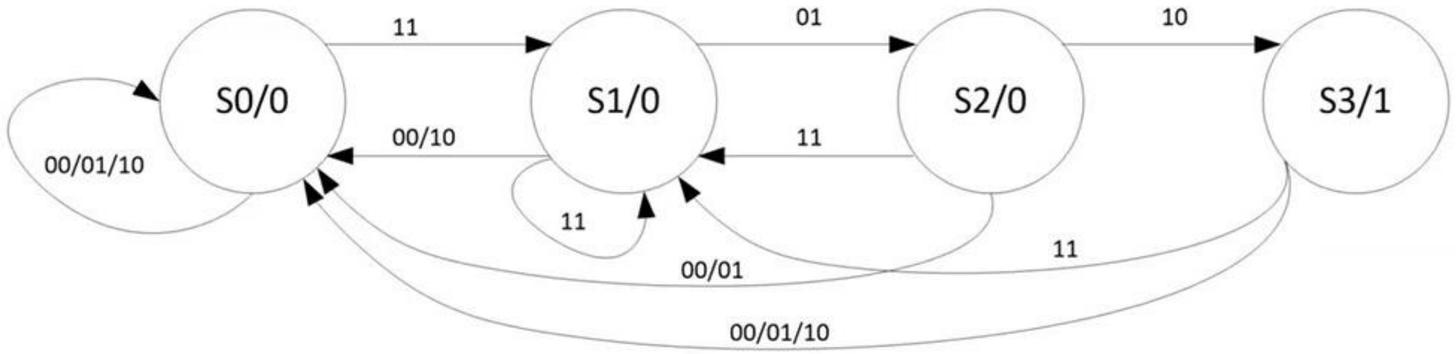
Descrizione del riconoscitore di sequenza

- 11,01,10



Descrizione del riconoscitore di sequenza (cont.)

- Dal grafo di flusso si passa alla tabella di flusso in maniera ovvia



	x_1x_0 00	01	11	10	z
S_0	S0	S0	S1	S0	0
S_1	S0	S2	S1	S0	0
S_2	S0	S0	S1	S3	0
S_3	S0	S0	S1	S0	1

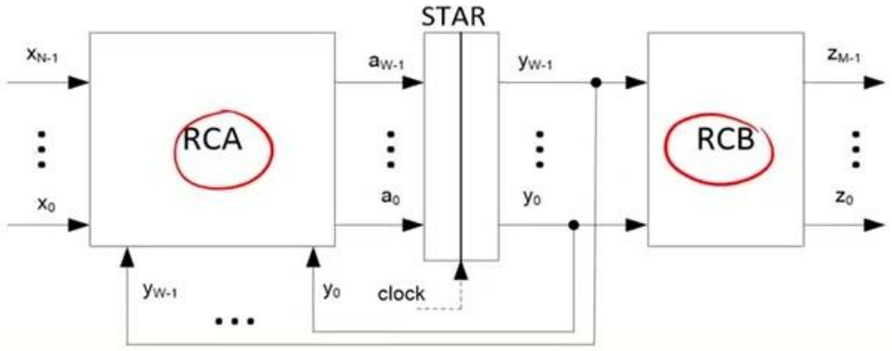
Descrizione del riconoscitore di sequenza (cont.)

- Dal grafo/tabella di flusso si passa alla descrizione Verilog in maniera ovvia

```
module Riconoscitore_di_Sequenza(z,x1_x0,clock,reset_);
  input clock,reset_;
  input [1:0] x1_x0;
  output z;
  reg [1:0] STAR; parameter S0='B00, S1='B01, S2='B10, S3='B11;
  assign z=(STAR==S3)?1:0;
  always @(reset_==0) #1 STAR<=S0;
  always @(posedge clock) if (reset_==1) #3
   casex(STAR)
    S0: STAR<=(x1_x0=='B11)?S1:S0;
    S1: STAR<=(x1_x0=='B01)?S2:(x1_x0=='B11)?S1:S0;
    S2: STAR<=(x1_x0=='B10)?S3:(x1_x0=='B11)?S1:S0;
    S3: STAR<=(x1_x0=='B11)?S1:S0;
   endcase
endmodule
```

x_1x_0	00	01	11	10	z
S_0	S0	S0	S1	S0	0
S_1	S0	S2	S1	S0	0
S_2	S0	S0	S1	S3	0
S_3	S0	S0	S1	S0	1

Sintesi del riconoscitore di seq.



DFF

$$\begin{cases} a_1 = \overline{x_1} \cdot x_0 \, \overline{y_1} \cdot y_0 + x_1 \cdot \overline{x_0} \cdot y_1 \cdot \overline{y_0} \\ a_0 = x_1 \cdot x_0 + x_1 \cdot y_1 \cdot \overline{y_0} \end{cases}$$

$z = y_1 \cdot y_0$

- Corrisponde a:
  - **scegliere** di un modello di sintesi
  - sintetizzare le due reti combinatorie RCA e RCB del modello
  - Per farlo va scelta una codifica (qualunque) per gli stati interni

x_1x_0	00	01	11	10	z
S_0	S0	S0	S1	S0	0
S_1	S0	S2	S1	S0	0
S_2	S0	S0	S1	S3	0
S_3	S0	S0	S1	S0	1

TF

Sint	y_1y_0
S_0	00
S_1	01
S_2	10
S_3	11

y_1y_0 \ x_1x_0	00	01	11	10	z
00	00	00	01	00	0
01	00	10	01	00	0
11	00	00	01	00	1
10	00	00	01	11	0

a_1a_0

MK

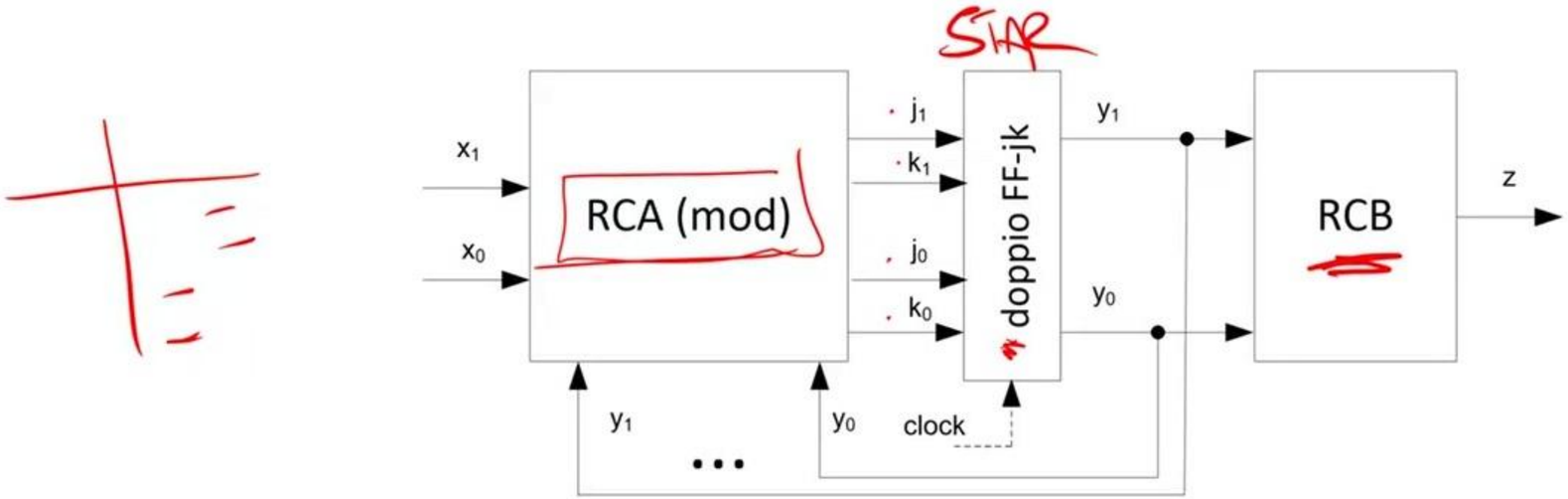
$$a_1 = \overline{x_1} \cdot x_0 \cdot \overline{y_1} \cdot y_0 + x_1 \cdot \overline{x_0} \cdot y_1 \cdot \overline{y_0}$$

$$a_0 = x_1 \cdot x_0 + x_1 \cdot y_1 \cdot \overline{y_0}$$

$$z = y_1 \cdot y_0.$$

Sintesi del riconoscitore di sequenza (cont.)

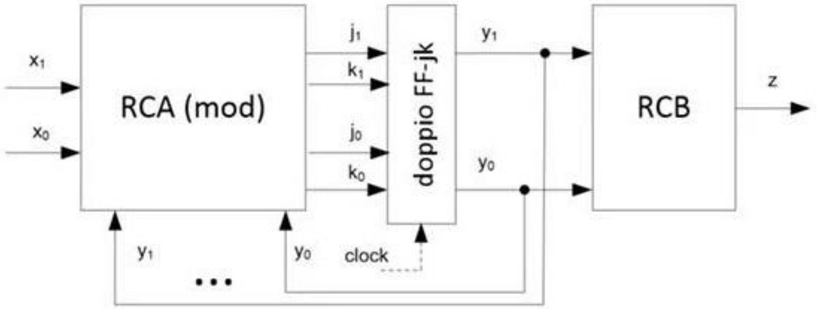
- Potremmo scegliere **un diverso modello di sintesi**
  - Uso FF JK invece che D-FF come elementi di marcatura
  - Sono pur sempre registri che memorizzano un bit ciascuno



  - La rete combinatoria B rimane identica
  - La rete combinatoria A deve essere modificata – ed ha il **doppio delle uscite**

Sintesi con modello a FF JK

- In realtà la RCA viene spesso più semplice
  - Le TV sono piene di non specificati



q	q'	j	k
0	0	0	-
0	1	1	-
1	0	-	1
1	1	-	0

y_1y_0 \ x_1x_0	00	01	11	10
00	00	00	01	00
01	00	10	01	00
11	00	00	01	00
10	00	00	01	11

a_1a_0

y_1y_0 \ x_1x_0	00	01	11	10
00	0	0	0	0
01	0	1	0	0
11	0	0	0	0
10	0	0	0	1

a_1

y_1y_0 \ x_1x_0	00	01	11	10
00	0	0	1	0
01	0	0	1	0
11	0	0	1	0
10	0	0	1	1

a_0

y_1y_0 \ x_1x_0	00	01	11	10
00				
01				
11				
10				

y_1y_0 \ x_1x_0	00	01	11	10
00				
01				
11				
10				

$$j_1 = \overline{x_1} \cdot x_0 \cdot y_0$$

$$k_1 = \overline{x_1} + x_0 + y_0$$

$$j_0 = x_1 \cdot y_1 + x_1 \cdot x_0$$

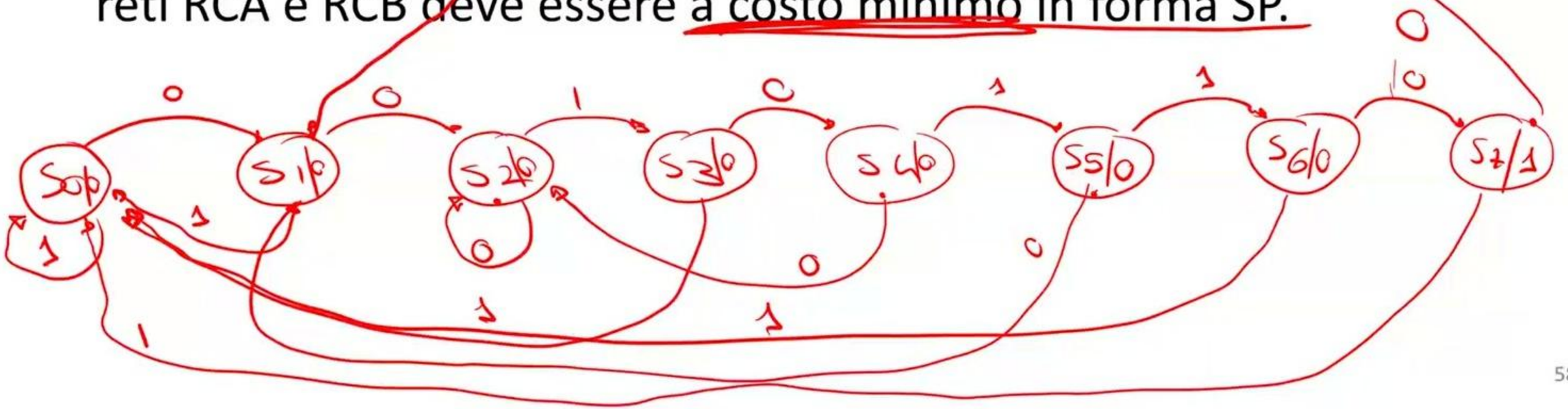
$$k_0 = \overline{x_1} + \overline{x_0}$$

$$z = y_1 \cdot y_0$$

Esercizio

K K+1 S_0 - — S_7 cc1[00 ---1[0
8 000 111 0[00

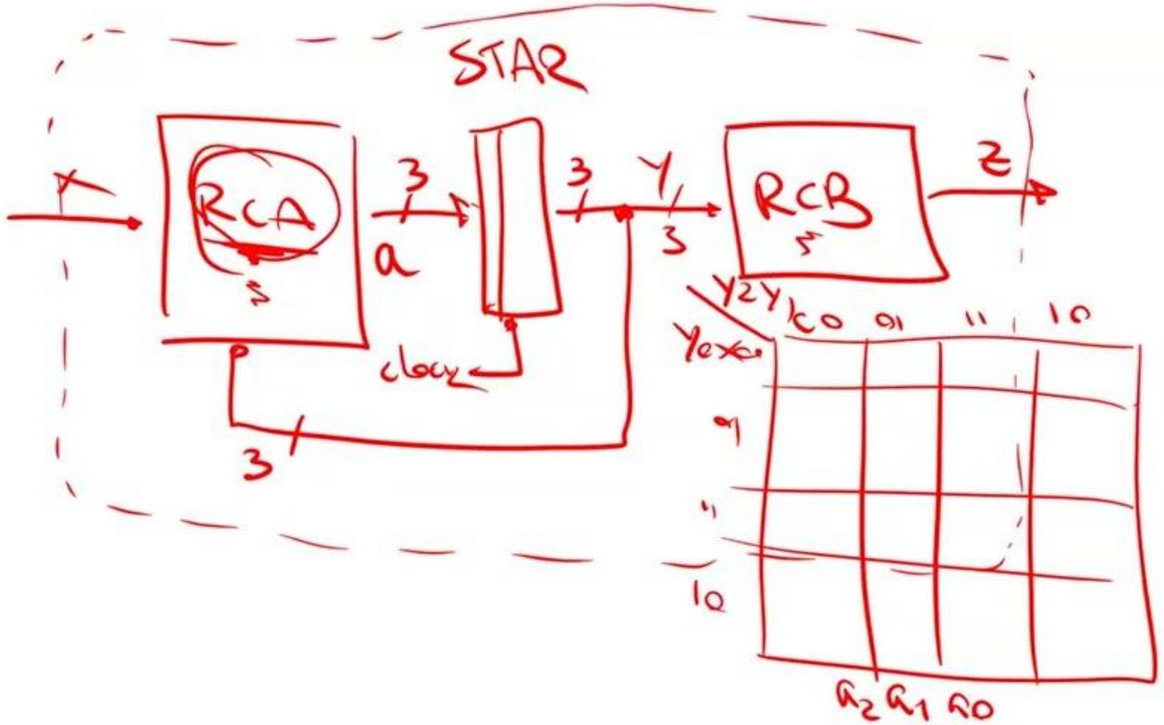
1. Descrivere una rete sequenziale sincronizzata di Moore ad 1 ingresso che riconosce la sequenza **0,0,1,0,1,1,0**. Si presti particolare attenzione a non perdere nessuna sequenza, e non si considerino valide sequenze interallacciate.
2. Sintetizzare la rete descritta al punto precedente. La sintesi delle reti RCA e RCB deve essere a costo minimo in forma SP.



Soluzione - descrizione

Sequenza: 0,0,1,0,1,1,0

- 7 stati di ingresso consecutivi.
- Devo prevedere 7+1=8 stati interni,
- l'ultimo dei quali avrà un'uscita pari ad 1



$y_2y_1y_0$	x	0	1	z	Parte di seq. gia' riconosciuta all'ingresso nello stato
000	S0	S_1	S_0	0	-
001	S1	S_2	S_0	0	0
010	S2	S_2	S_3	0	0,0
011	S3	S_4	S_0	0	0,0,1
100	S4	S_2	S_5	0	0,0,1,0
101	S5	S_1	S_6	0	0,0,1,0,1
110	S6	S_7	S_0	0	0,0,1,0,1,1
111	S7	S_1	S_0	1	0,0,1,0,1,1,0

Soluzione - sintesi

- Codifica degli stati: $S_i = (i)_{b2}$
- Modello di sintesi: D-FF come marcatura

$y_2y_1y_0$ \ x	0	1	z
000	**001**	000	0
001	**010**	000	0
010	010	**011**	0
011	**100**	000	0
100	010	**101**	0
101	001	**110**	0
110	**111**	000	0
111	001	000	1

y_1y_0 \ y_2x	00	01	11	10
	$a_2a_1a_0$			
00	001	000	101	010
01	010	000	110	001
11	100	000	000	001
10	010	011	000	111

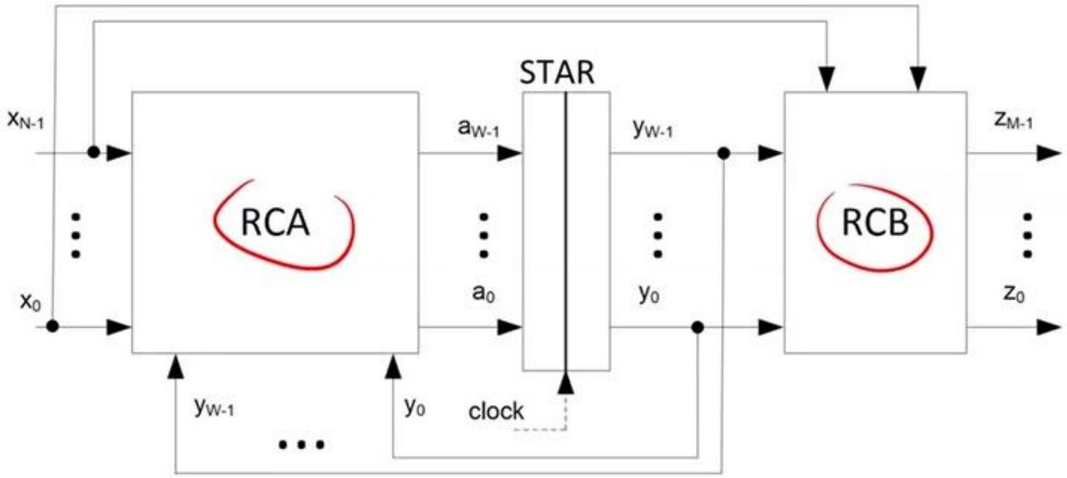
$$a_2 = y_2 \cdot x \cdot \overline{y_1} + \overline{y_2}\, y_1 y_0 \overline{x} + y_2 y_1 \overline{y_0}\, \overline{x}$$

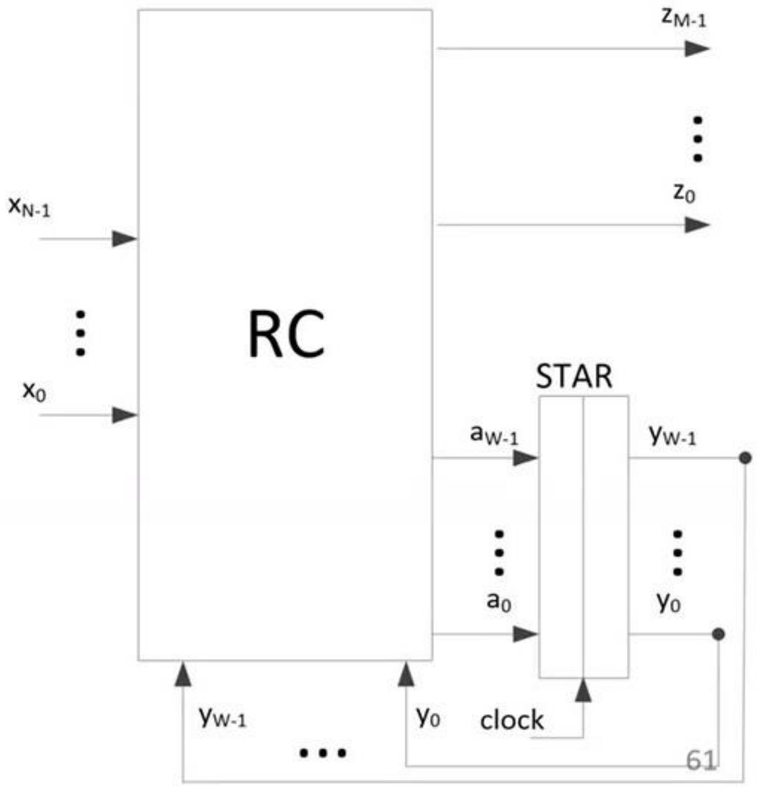
$$\boxed{z = y_2 \cdot y_1 \cdot y_0}$$

Modello di Mealy

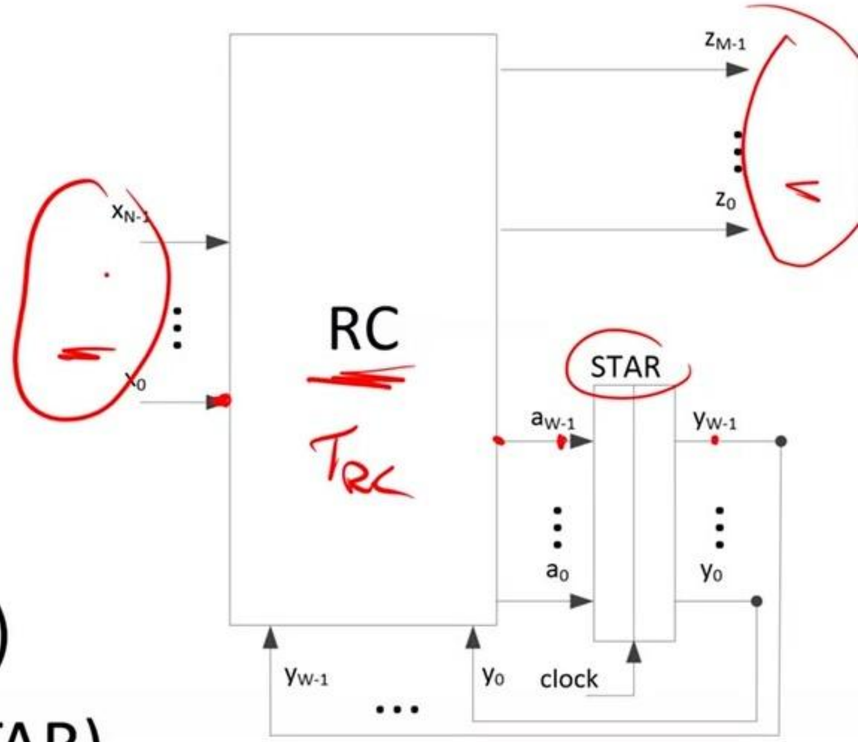
- Modello di Moore
  - L'uscita è funzione **soltanto dello stato interno presente**: $B: \boldsymbol{S} \rightarrow \boldsymbol{Z}$.

- **Modello di Mealy**
  - Rendo la legge B più generale, $B: \boldsymbol{X} \times \boldsymbol{S} \rightarrow \boldsymbol{Z}$
  - L'uscita dipende anche dallo **stato di ingresso**





Leggi di temporizzazione



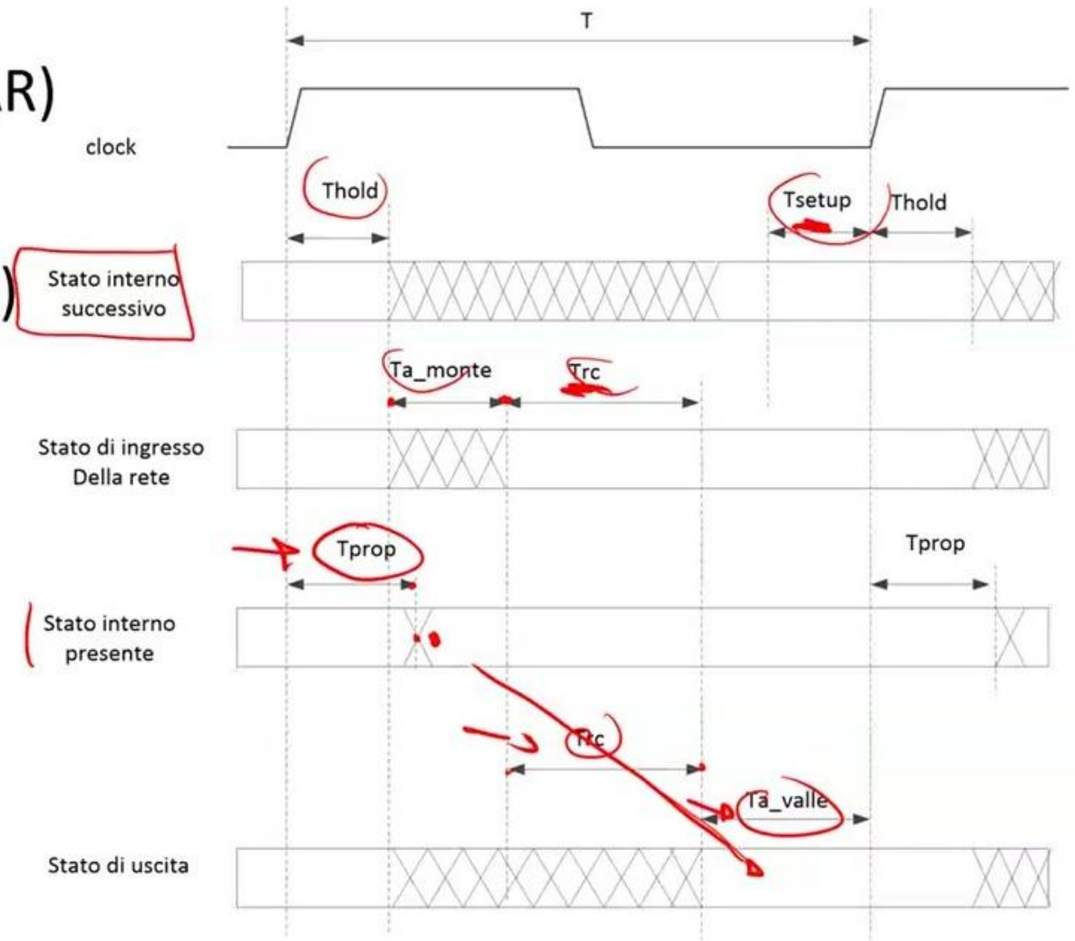
Node

- $T \geq T_{hold} + T_{a_monte} + T_{RC} + T_{setup}$ (in->STAR)
- $T \geq T_{prop} + T_{RC} + T_{setup}$ (STAR->STAR)
- $T \geq T_{hold} + T_{a_monte} + T_{RC} + T_{a_valle}$ (in->out)
- $T \geq T_{prop} + T_{RC} + T_{a_valle}$ (STAR->out)

Mesh

Leggi di temporizzazione (cont.)

- $T \geq T_{hold} + T_{a_monte} + T_{RC} + T_{setup}$ (in->STAR)
- $T \geq T_{prop} + T_{RC} + T_{setup}$ (STAR->STAR)
- $T \geq T_{hold} + T_{a_monte} + T_{RC} + T_{a_valle}$ (in->out)
- $T \geq T_{prop} + T_{RC} + T_{a_valle}$ (STAR->out)



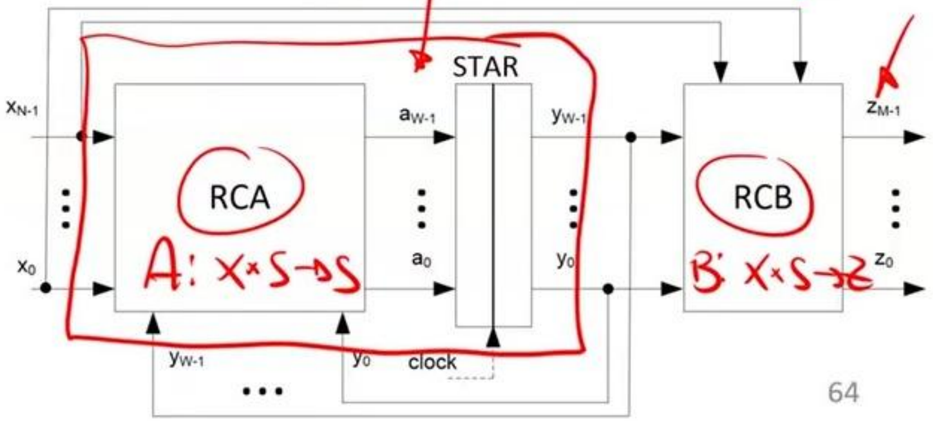
Descrizione di una rete di Mealy in Verilog

```
module Rete_di_Mealy(zM-1,...,z0,xN-1,...,x0,clock,reset_);
  input clock,reset_;
  input xN-1,...,x0;
  output zM-1,...,z0;
  reg [W-1:0] STAR; parameter S0=codifica0,...,SK-1=codificaK-1;
  assign {zM-1,...,z0} = (STAR==S0)? BS0(xN-1,...,x0) :
                                 ...
                                 ...
                                 ...
                          /*(STAR==SK-1)*/ BSK-1(xN-1,...,x0);
  always @(reset_==0) #1 STAR<=stato_interno_iniziale;
  always @(posedge clock) if (reset_==1) #3
   casex(STAR)
    S0 : STAR<=AS0(xN-1,...,x0);
    ...
    ...
    ...
    SK-1 : STAR<=ASK-1(xN-1,...,x0);
   endcase
endmodule
```

B

always

assign



Moore $Z_i \propto X_{i-1}, X_{i-2} \ldots X_0, S_0$

Modello di Mealy (cont.)

$Z_i \propto X_i, X_{i-1} \ldots X_0, S_0$

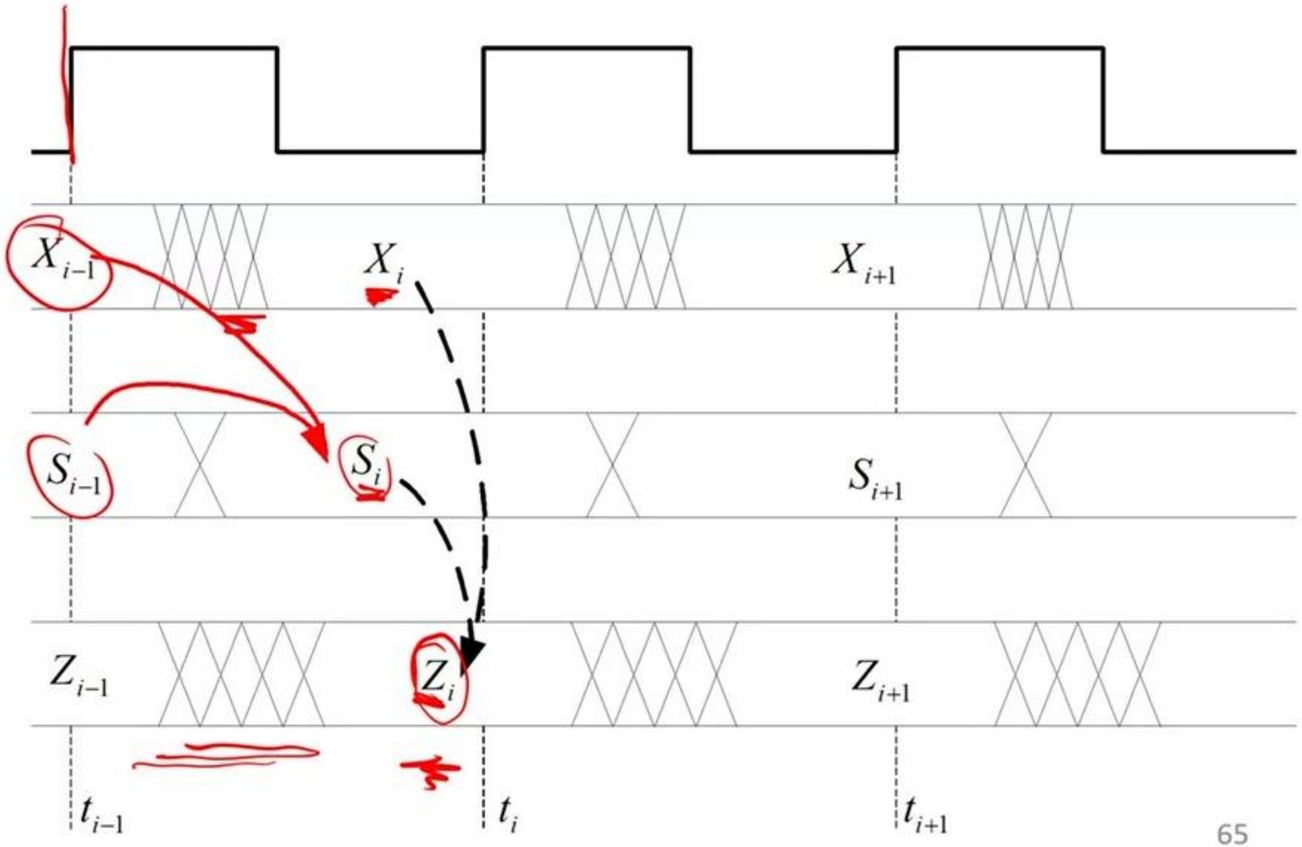
- **Al variare dell'ingresso** posso produrre un **nuovo stato di uscita** senza dover aspettare il successivo fronte del clock

$$\boldsymbol{S[t_{i+1}] = A(X[t_i], S[t_i])}$$

$$\boldsymbol{Z[t_i] = B(X[t_i], S[t_i])}$$

In una rete di Moore è:

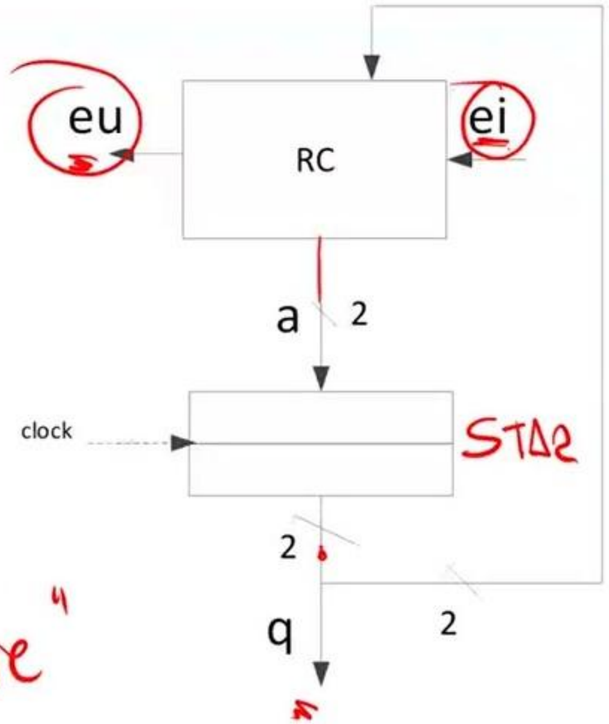
$$Z[t_i] = B(S[t_i])$$



Esempio – contatore espandibile in base 3

"di Mealy"

- È una rete di Mealy
  - L'uscita **eu** infatti dipende in modo combinatorio dagli ingressi
- Rete con **tre stati interni**
  - Cifre 0, 1, 2 in base 3
- Legge $A: \boldsymbol{X} \times \boldsymbol{S} \rightarrow \boldsymbol{S}$: calcola una nuova cifra
  - Dato il contenuto del registro ed il riporto entrante
- Legge $B: \boldsymbol{X} \times \boldsymbol{S} \rightarrow \boldsymbol{Z}$: calcola q ed eu
  - Dato il contenuto del registro ed il riporto entrante



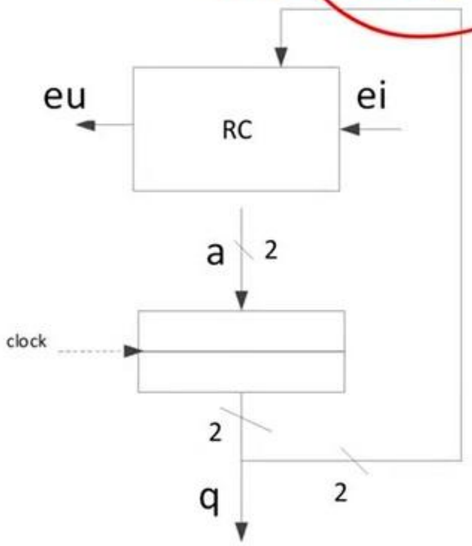
"di Moore"

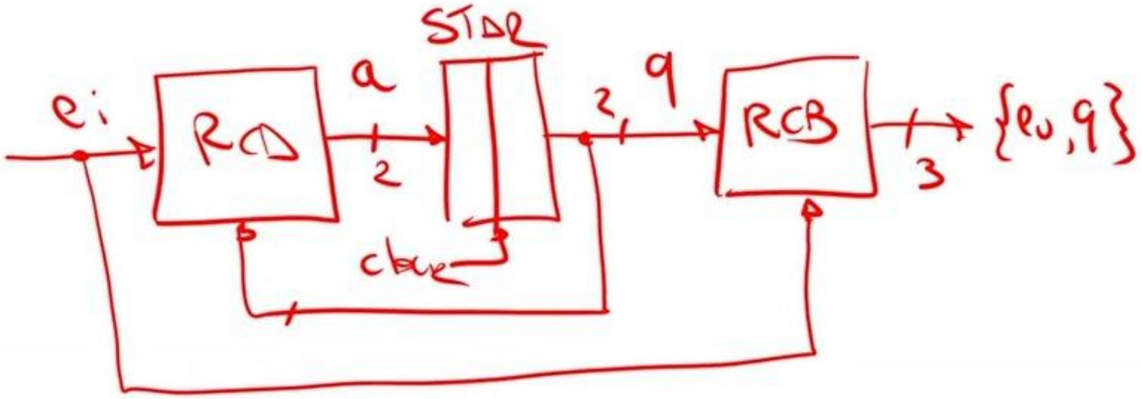
- Possiamo descriverlo con un **grafo** o una **tabella di flusso**

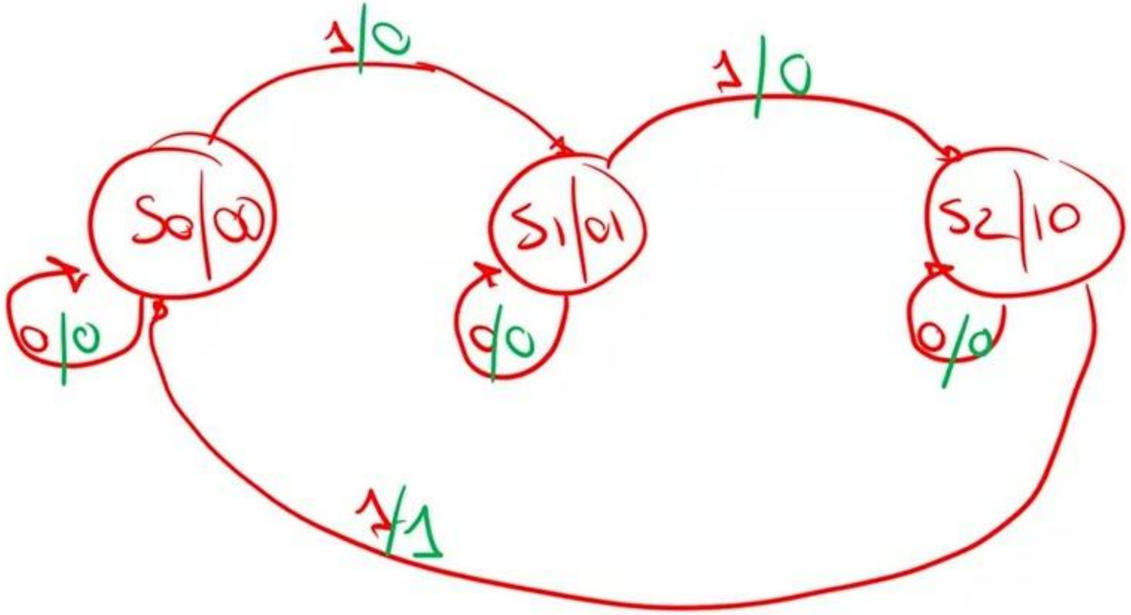
Esempio (cont.)

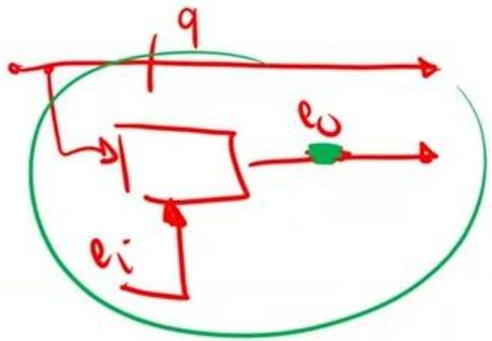
$B_i: S \to Z$

$B_i: X \times S \to Z$

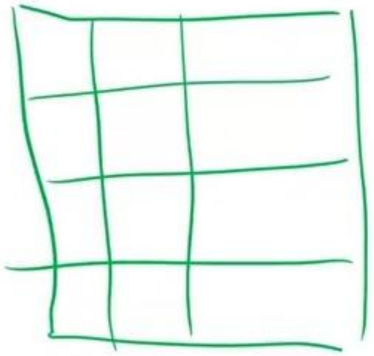








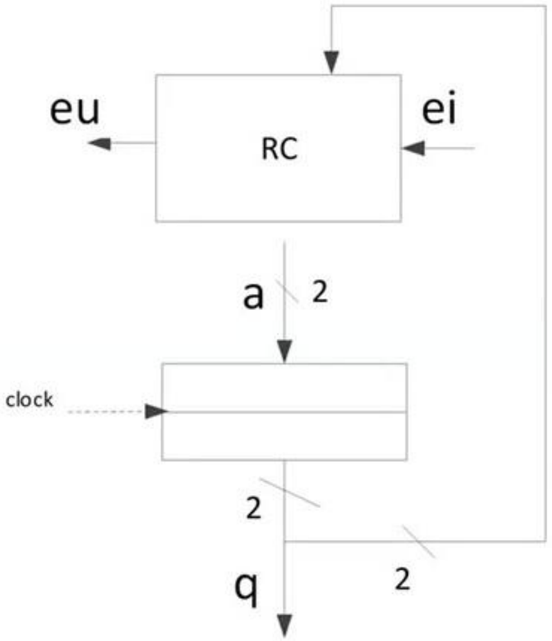
Esempio (cont.)



q_1q_0 \ e_i	0	1
00	0	0
01	0	0
11	–	–
10	0	1

e_u

$e_u = q_1\bar{q}_0 \cdot e_i$



- Le uscite vanno scritte **sugli archi / nelle caselle**
  - Oppure parte negli stati (uscite di Moore) e parte sugli archi (uscite di Mealy)

e_i \ q_1q_0 e_u	0	1
(00) S0	S0 00 0	S1 00 0
(01) S1	S1 01 0	S2 01 0
(10) S2	S2 10 0	S0 10 1

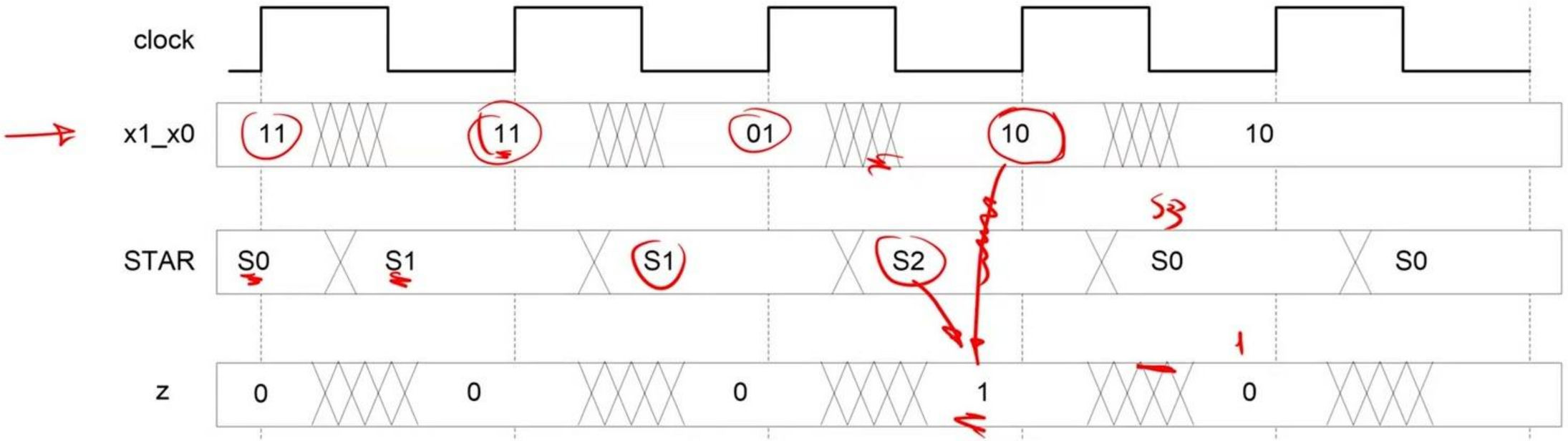
e_i \ e_u	0	1	q_1q_0
S0	S0 0	S1 0	00
S1	S1 0	S2 0	01
S2	S2 0	S0 1	10

q_1q_0 \ e_1	0	1
00	00	01
01	01	10
11	–	–
10	10	00

a_1a_0

Esempio: riconoscitore di sequenza 11, 01, 10

- Già fatto come **rete di Moore**. Descriviamolo come **rete di Mealy**
  - Vediamo una possibile evoluzione nel tempo



- In S2, con ingresso 10, posso **portare l'uscita ad 1**
  - Senza aspettare il clock successivo

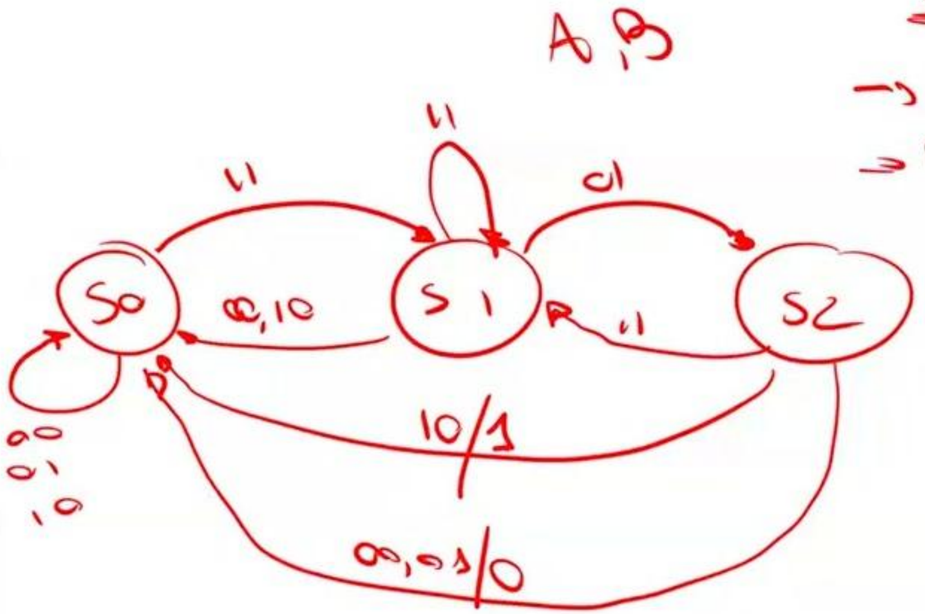
Esempio: riconoscitore di sequenza 11, 01, 10

- Descrizione tramite grafo di flusso

Temporizzazioni

→ A pres.

→ B comb.



- Tabella di flusso

A,B

	x_1x_0 00	01	11	10
S0	S0/0	S0/0	S1/0	S0/0
S1	S0/0	S2/0	S1/0	S0/0
S2	S0/0	S0/0	S1/0	S0/1

Esempio: riconoscitore di sequenza 11, 01, 10

```
module Riconoscitore_di_Sequenze(z,x1_x0,clock,reset_);
  input clock,reset_;
  input [1:0] x1_x0;
  output z;
  reg [1:0] STAR; parameter S0='B00,S1='B01,S2='B10;
  assign z=((STAR==S2) & (x1_x0=='B10))?1:0;
  always @(reset_==0) #1 STAR<=S0;
  always @(posedge clock) if (reset_==1) #3
    casex(STAR)
     S0: STAR<=(x1_x0=='B11)?S1:S0;
     S1: STAR<=(x1_x0=='B01)?S2:(x1_x0=='B11)?S1:S0;
     S2: STAR<=(x1_x0=='B11)?S1:S0;
    endcase
endmodule
```

x_1x_0	00	01	11	10
S0	S0/0	S0/0	S1/0	S0/0
S1	S0/0	S2/0	S1/0	S0/0
S2	S0/0	S0/0	S1/0	S0/1

Moore vs. Mealy

- **Moore**
- Legge B meno flessibile
  - Più stati interni

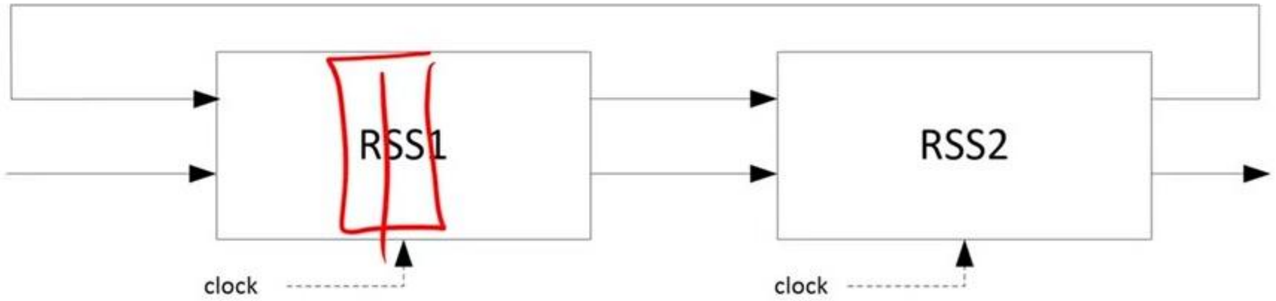
- **Mealy**
- Legge B più flessibile
  - Meno stati interni

Stessa potenza descrittiva
C'è un teorema che ci garantisce che dato un problema risolto con Moore si trova una rete di Mealy che lo risolve, **e viceversa**

- Clock più veloce
- Uscita **un clock in ritardo** rispetto agli ingressi
- **Non trasparente**

- Clock più lento
- Uscita «**in pari**» con gli ingressi
- **Trasparente**

Moore vs. Mealy



- Un montaggio come quello di figura **può essere instabile** se le reti sono **entrambe di Mealy**
  - Si può creare un anello combinatorio (RSA)

- Se almeno una delle due reti è di Moore non ci sono problemi
  - Dentro l'anello c'è comunque un registro

Esercizio

S_0 S_1 S_2 S_3

il pre. s. ingresso 00 00 01 11 10

- **Descrivere e sintetizzare** una RSS di Mealy che ha due variabili di ingresso $\boldsymbol{x_1}$ e $\boldsymbol{x_0}$, ed una variabile di uscita *z*.
- La rete si evolve nel seguente modo:
  - se lo stato d'ingresso corrente è **uguale al precedente**, $\boldsymbol{z} = \boldsymbol{x_1} \cdot \boldsymbol{x_0}$
  - altrimenti, se lo stato d'ingresso corrente **non è uguale al precedente**, $\boldsymbol{z} = \boldsymbol{x_1} \oplus \boldsymbol{x_0}$

- **Nota**: il primo stato di uscita della rete è non significativo.

Soluzione

$S_{in} = S_{in\,prec}$

$z = x_1 \cdot x_0$

$z = x_1 \oplus x_0$

- È necessario memorizzare **l'ultimo stato di ingresso** visto dalla rete
  - Quattro possibilità, quindi **quattro stati interni**

S. interno	S. ingresso precedente
S0	00
S1	01
S2	11
S3	10

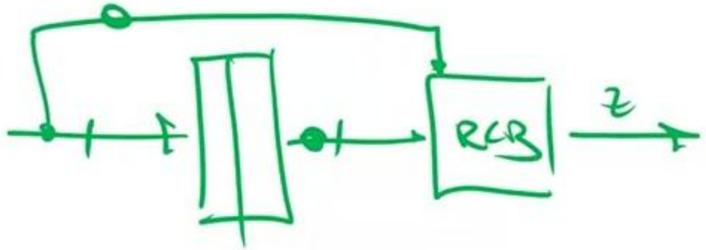
$S_{in.cor}$ — A $X \times S \to S$ — $X \to S$

$S_{in\,prec}$

x_1x_0	00 (S_0)	01 (S_1)	11 (S_2)	10 (S_3)
S_0	S_0 0	S_1 1	S_2 0	S_3 1
S_1	S_0 0	S_1 0	S_2 0	S_3 1
S_2	S_0 0	S_1 1	S_2 1	S_3 1
S_3	S_0 0	S_1 1	S_2 0	S_3 0

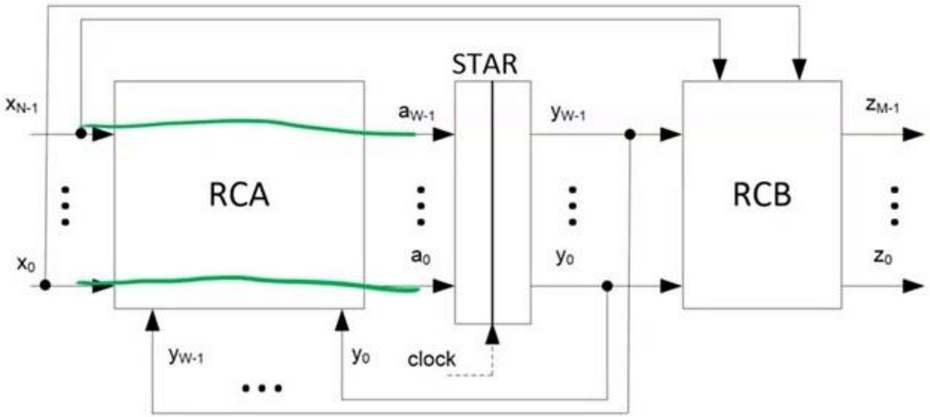
Soluzione (cont.)

$a_1 = x_1$
$a_0 = x_0$



	x_1x_0	00	01	11	10
00	S_0	$S_0/0$	$S_1/1$	$S_2/0$	$S_3/1$
01	S_1	$S_0/0$	$S_1/0$	$S_2/0$	$S_3/1$
11	S_2	$S_0/0$	$S_1/1$	$S_2/1$	$S_3/1$
10	S_3	$S_0/0$	$S_1/1$	$S_2/0$	$S_3/0$

- Per la sintesi scelgo
  - Un modello di sintesi (e.g., D-FF come elementi di marcatura)
  - Una codifica degli stati interni: S0=00, S1=01, S2=11, S3=10



y_1y_0 \ x_1x_0	00	01	11	10
00	00	01	11	10
01	00	01	11	10
11	00	01	11	10
10	00	01	11	10

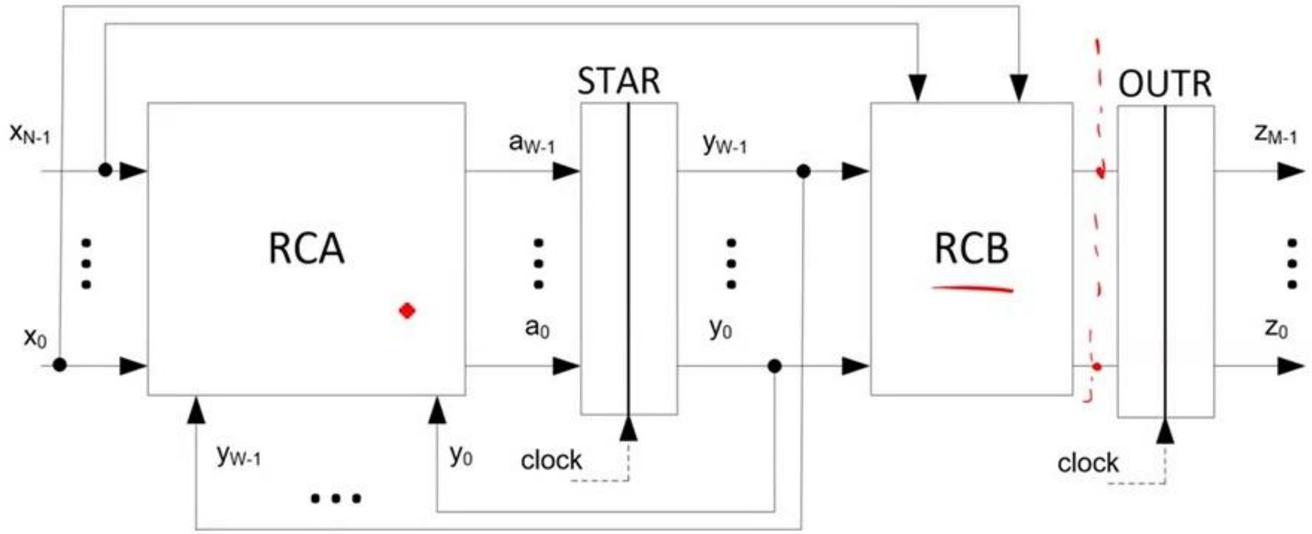
a_1a_0

y_1y_0 \ x_1x_0	00	01	11	10
00	0	1	0	1
01	0	0	0	1
11	0	1	1	1
10	0	1	0	0

z

Modello di Mealy ritardato

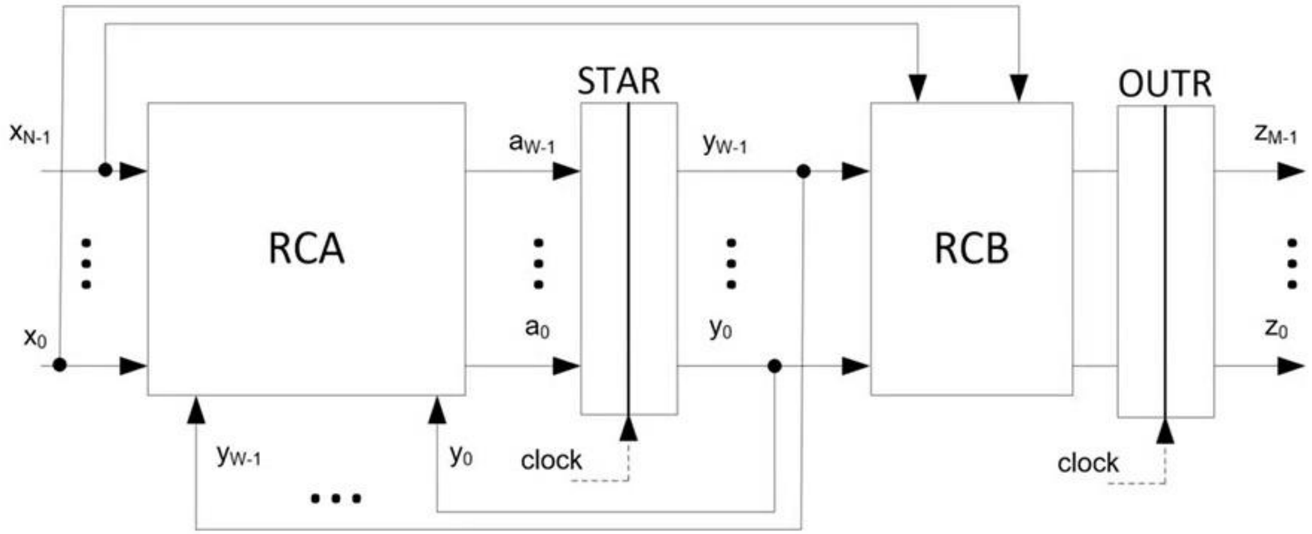
- Si parte da una rete di Mealy, e si mette in uscita **un altro registro OUTR**, con lo stesso clock di STAR

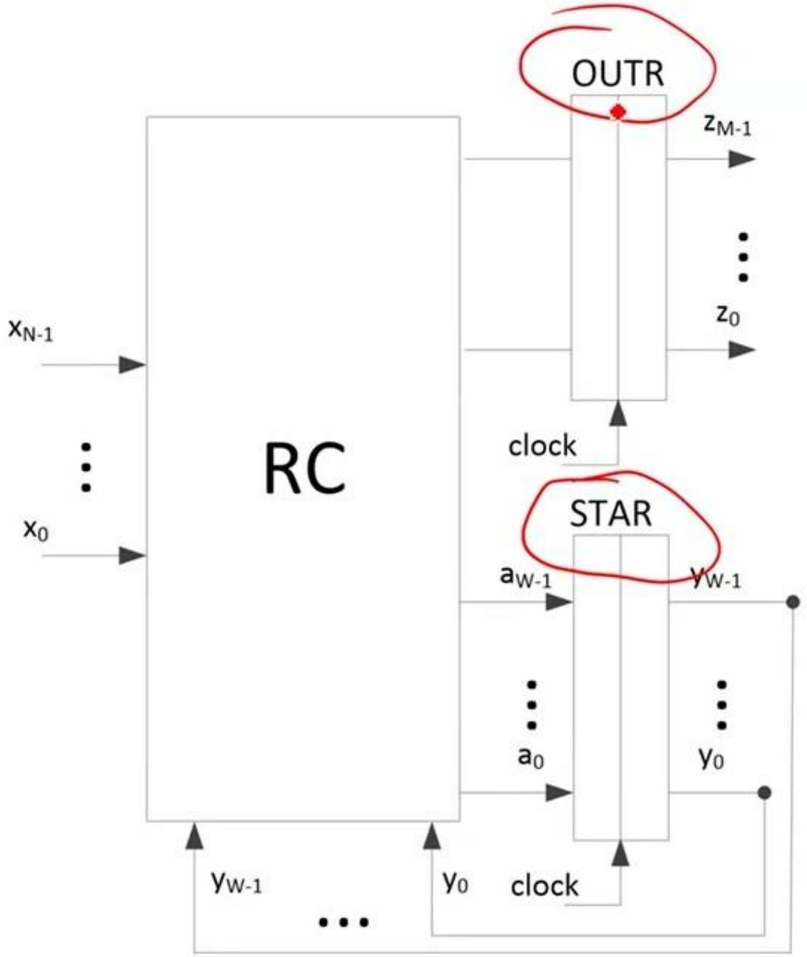


- Le uscite
  - variano sempre **all'arrivo del clock**, dopo un tempo T_{prop}
  - variano in maniera **netta, senza oscillazioni** (come invece può succedere in una rete di Mealy
  - rimangono stabili per **l'intero ciclo di clock.**
  - sono **non trasparenti**.

Modello di Mealy ritardato (cont.)

- RCA ed RCB hanno gli stessi ingressi,
  - posso disegnare la rete anche come nella figura a destra





Modello di Mealy ritardato (cont.)

- Una rete di Mealy ritardato
  - Ha una **legge di evoluzione nel tempo** del tipo A: $\boldsymbol{X} \times \boldsymbol{S} \rightarrow \boldsymbol{S}$, che mappa quindi una coppia (stato di ingresso, stato interno) in un nuovo stato interno.
  - Una **legge di evoluzione nel tempo** del tipo B: $\boldsymbol{X} \times \boldsymbol{S} \rightarrow \boldsymbol{Z}$, che mappa quindi una coppia (stato di ingresso, stato interno) in un nuovo stato di uscita.
- Si adegua alla seguente **legge di temporizzazione**:

"Dato S, stato interno presente (marcato) ad un certo istante, e dato X stato di ingresso ad un certo istante **precedente l'arrivo di un segnale di sincronizzazione**,

1) individuare **SIA** il **nuovo stato interno da marcare** $\boldsymbol{S'} = \boldsymbol{A}(\boldsymbol{X}, \boldsymbol{S})$, **SIA il nuovo stato di uscita** $\boldsymbol{Z} = \boldsymbol{B}(\boldsymbol{X}, \boldsymbol{S})$
2) **attendere** $\boldsymbol{T_{prop}}$ **dopo l'arrivo del segnale di sincronizzazione**
3) **promuovere** $\boldsymbol{S'}$ al rango di **stato interno marcato, e promuovere** $\boldsymbol{Z}$ al rango di **nuovo stato di uscita**

Modello di Mealy ritardato (cont.)

prima del clock

$$s' = A(x, s)$$

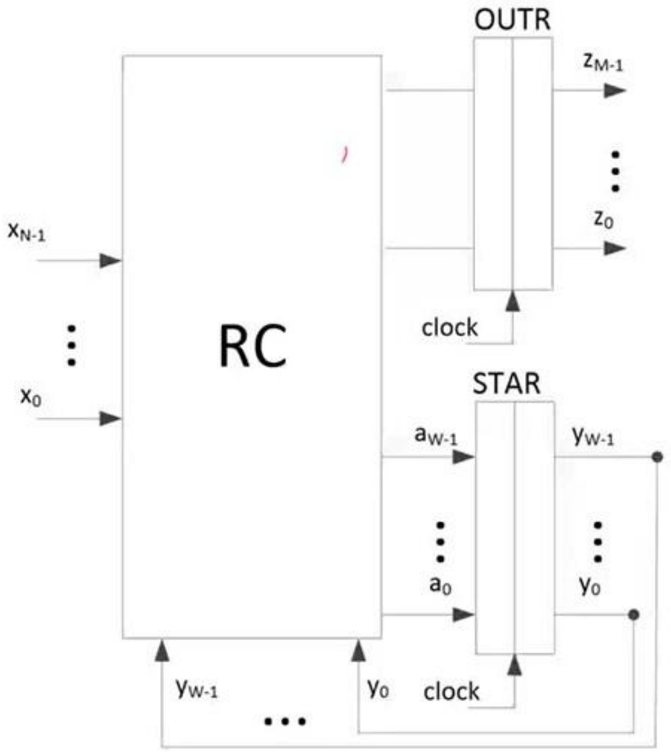
dopo clock

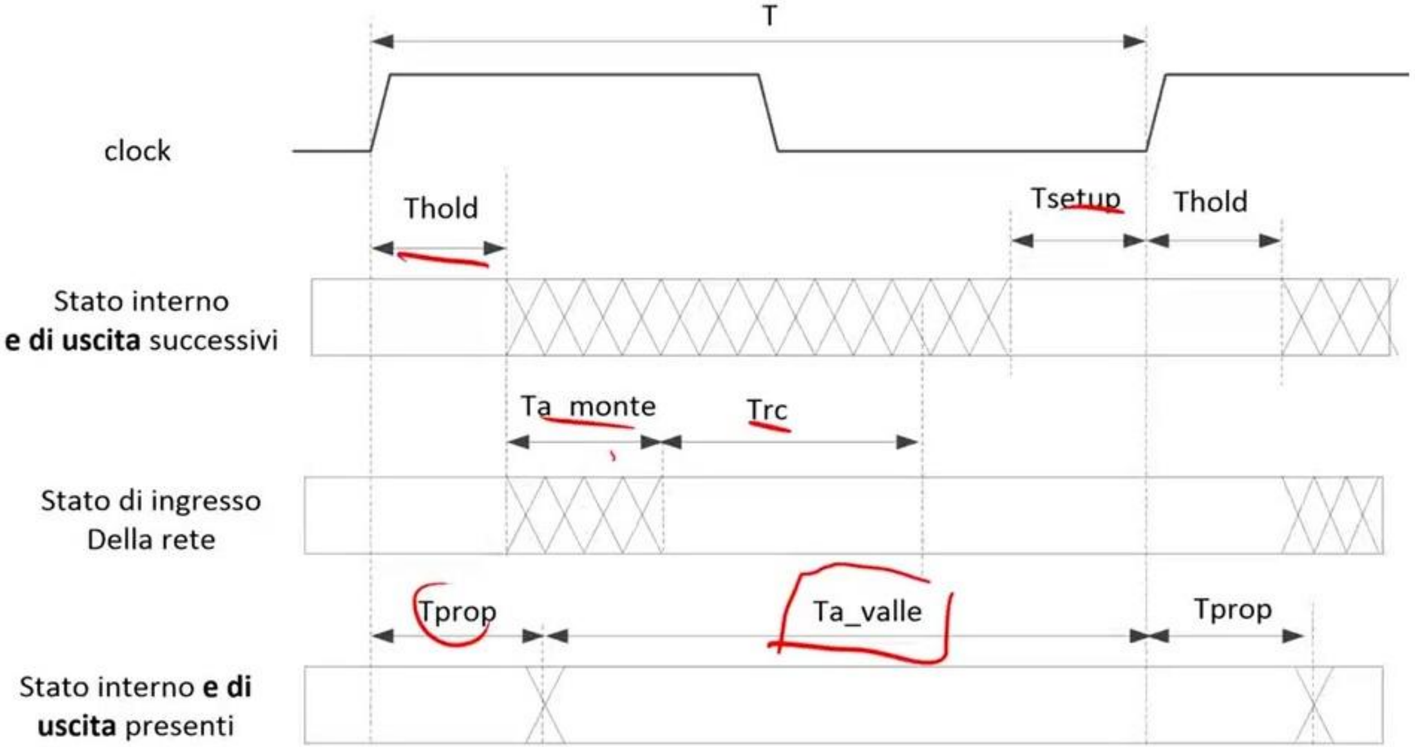
$$z = B(x, s)$$

- **Attenzione:**
- Sia la legge A, sia la legge B sono **procedurali**
- Sia il nuovo stato interno, sia il nuovo stato di uscita
  - vengono marcati **dopo l'arrivo del clock**
  - Dipendono dallo stato di ingresso e dallo stato interno **prima del clock**

Vincoli di temporizzazione per una RSS di Mealy R.

- $T \geq T_{hold} + T_{a_monte} + T_{RC} + T_{setup}$ (in -> reg)
- $\cancel{T \geq T_{prop} + T_{RC} + T_{setup}}$ STAR OUTR (reg -> reg)
- $T \geq T_{prop} + T_{a_valle}$ OUTR (reg -> out)





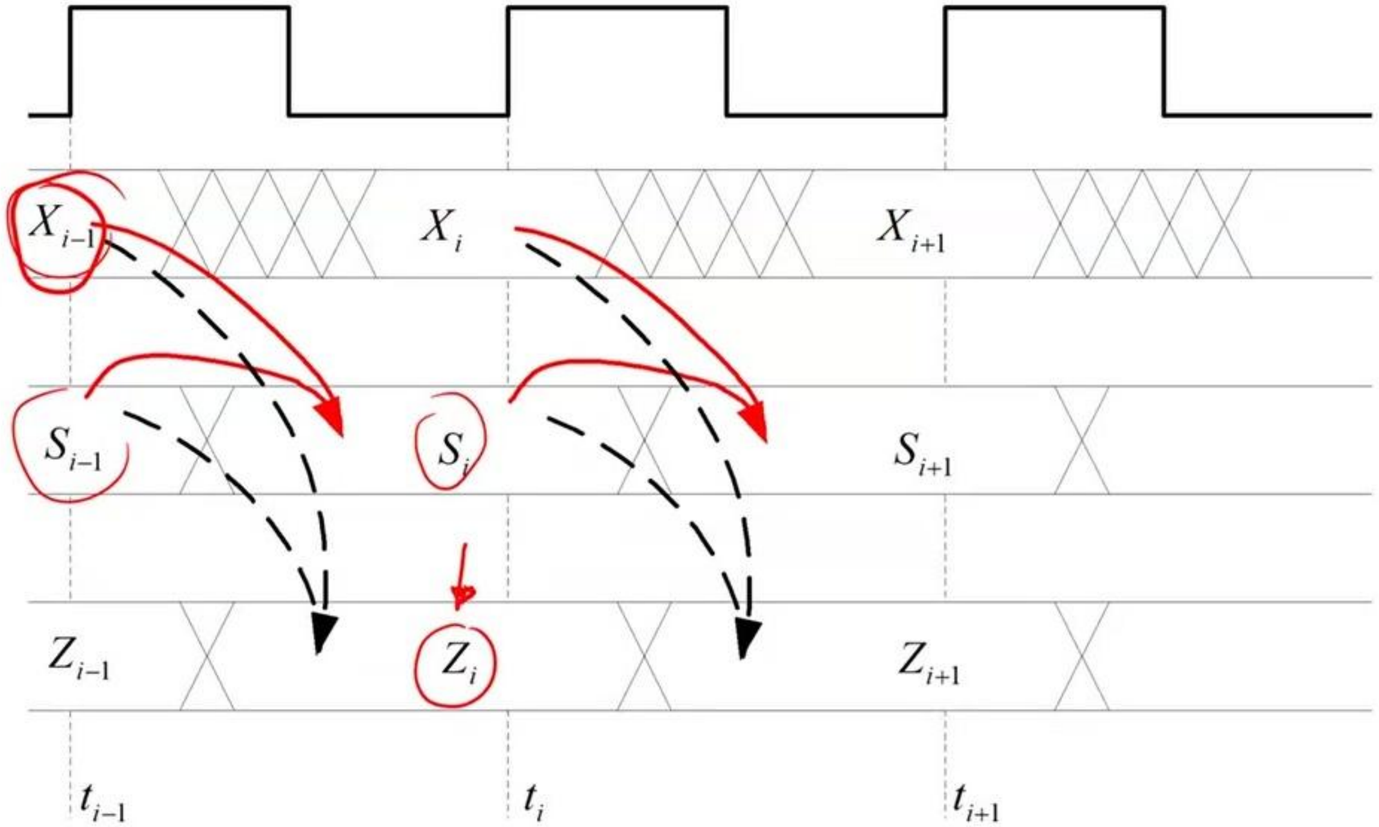
Evoluzione di una RSS di Mealy Ritardato

$Z_i \propto X_{i-1} \ldots X_0, S_0$

- Detti t_i, $i \geq 0$, gli **istanti di sincronizzazione**, e detti $X[t_i], S[t_i], Z[t_i]$ gli stati di **ingresso, interno e di uscita** all'istante t_i, sarà

$$\boldsymbol{S[t_{i+1}] = A(X[t_i], S[t_i])}$$

$$\boldsymbol{Z[t_{i+1}] = B(X[t_i], S[t_i])}$$



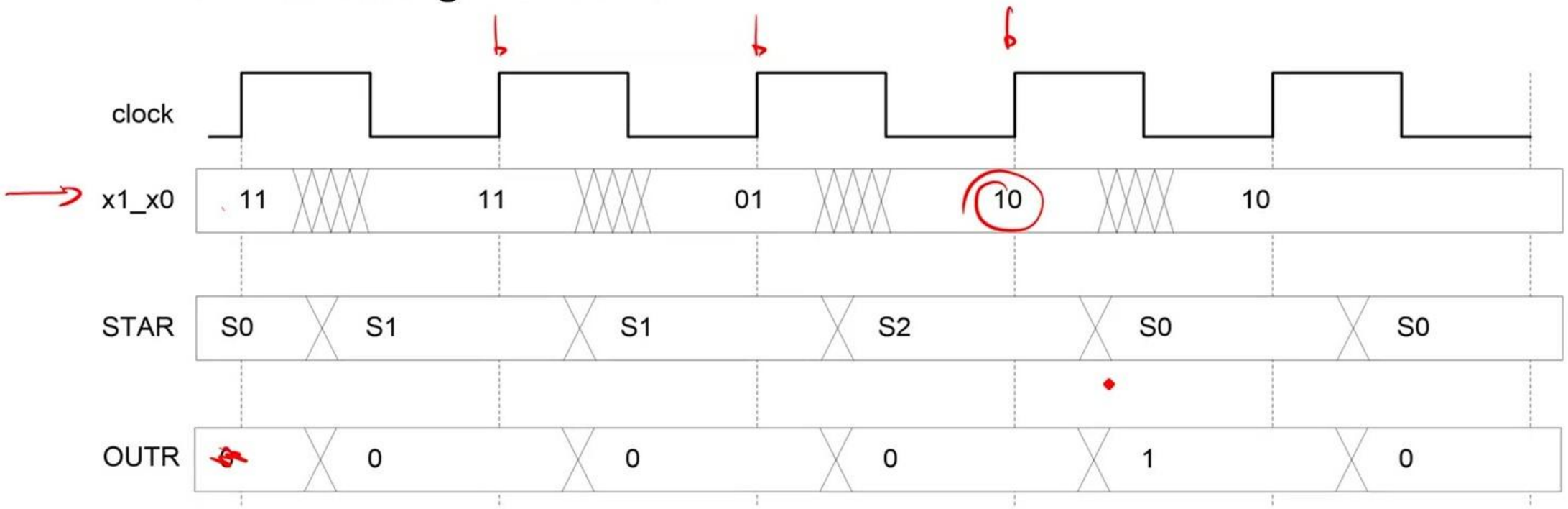
Moore: $Z[t_i] = B(S[t_i])$

Mealy: $Z[t_i] = B(X[t_i], S[t_i])$

Esempio: temporizzazione

```
S0: begin OUTR<=0; STAR<=(x1_x0=='B11)?S1:S0; end
S1: begin OUTR<=0; STAR<=(x1_x0=='B01)?S2:(x1_x0=='B11)?S1:S0; end
S2: begin OUTR<=(x1_x0=='B10)?1:0; STAR<=(x1_x0=='B11)?S1:S0; end
```

- L'uscita è un clock in ritardo rispetto all'ingresso che la genera
  - Per via del registro OUTR



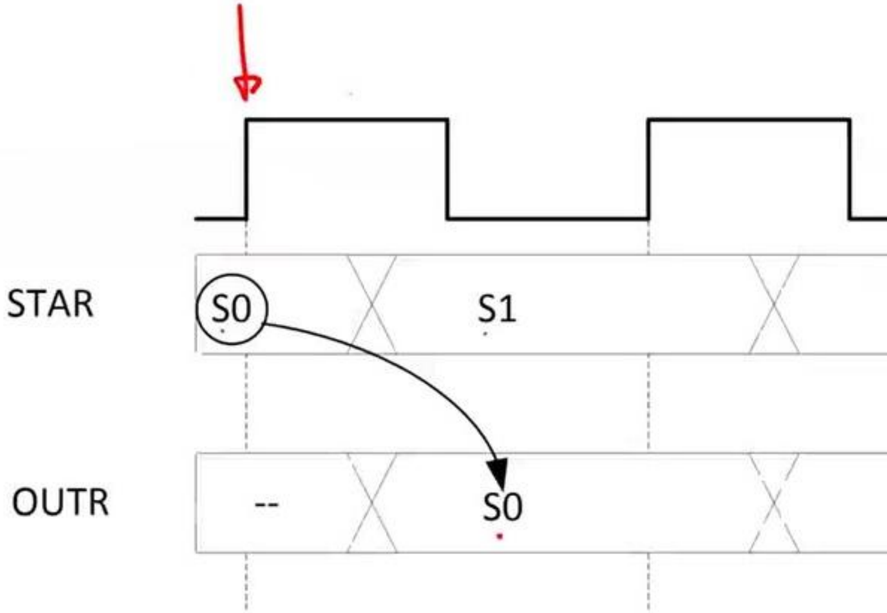
Assegnamenti procedurali in Verilog

- Due o più assegnamenti procedurali **racchiusi tra `begin`…`end`** verranno resi operativi **contemporaneamente**, all'arrivo del clock
- L'ordine in cui sono scritti è **ininfluente**

- È bene fare molta attenzione alla **temporizzazione**

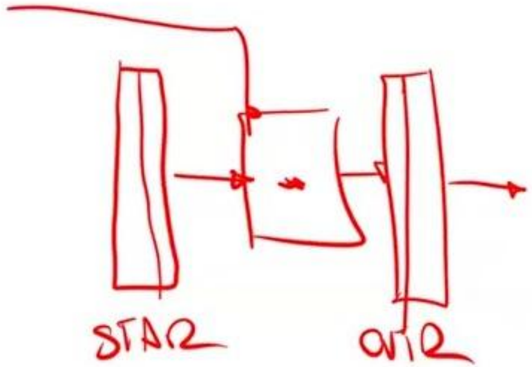
```
S0: begin STAR<=S1; OUTR<=STAR; end
```

Quale sarà il valore che OUTR memorizza all'arrivo del clock?



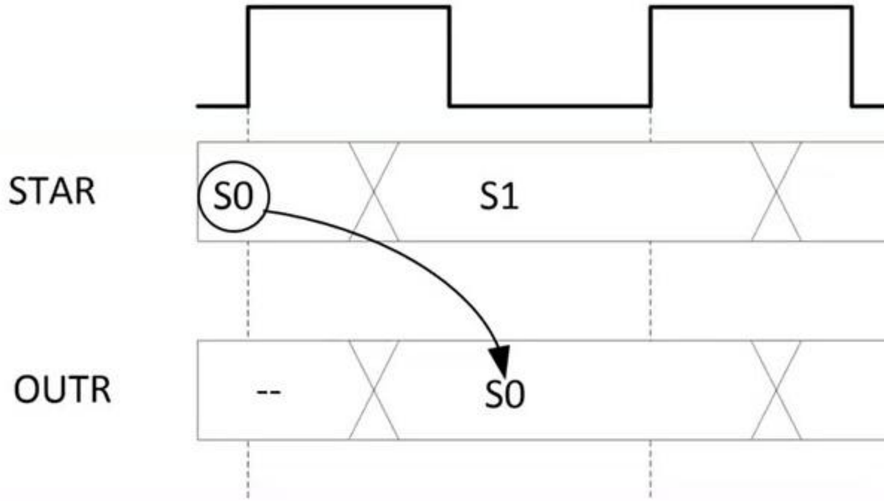
Assegnamenti procedurali in Verilog (cont.)

```
S0: begin STAR<=S1; OUTR<=STAR; end
```



Lettura
(prima del clock)

Scrittura
(dopo il clock)



- Nello stesso **`begin`**…**`end`** (cioè nell'ambito dello stesso ciclo di clock) non posso **contemporaneamente** assegnare un valore ad un registro **ed** usare il nuovo valore.
- Il nuovo valore sarà utilizzabile soltanto a partire dal clock successivo.

Esempio: riconoscitore di sequenza 11, 01, 10

- Descrizione tramite tabella/grafo di flusso
  - **La stessa che nel caso di rete di Mealy**
  - Nelle tabelle/grafi di flusso **non si vedono le temporizzazioni** delle leggi A, B

Mealy

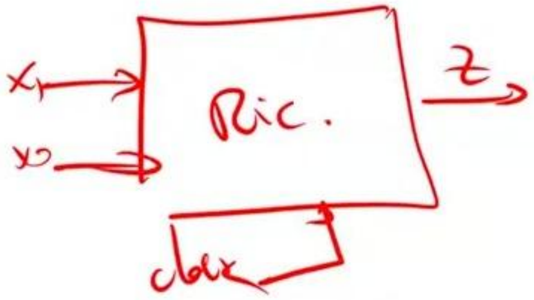
A, B

B : X×S → Z

x_1x_0	00	01	11	10
S0	S0/0	S0/0	S1/0	S0/0
S1	S0/0	S2/0	S1/0	S0/0
S2	S0/0	S0/0	S1/0	S0/1

Esempio: riconoscitore di sequenza 11, 01, 10

- A livello di Verilog, invece, queste differenze sono evidenti
  - La legge B è realizzata tramite assegnamenti **procedurali**
  - L'evoluzione di STAR resta identica a quella di una rete di Mealy



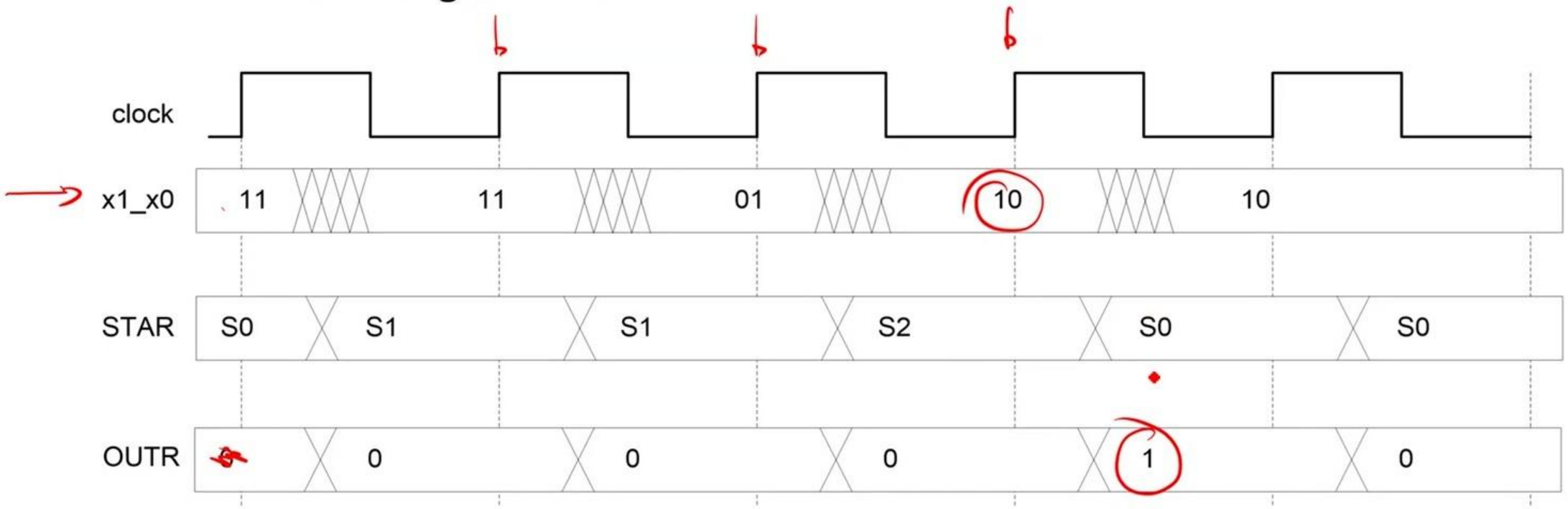

```
module Riconoscitore_di_Sequenze(z,x1_x0,clock,reset_);
  input clock,reset_;
  input [1:0] x1_x0;
  output z;
  reg [1:0] STAR; parameter S0='B00,S1='B01,S2='B10;
  reg OUTR; assign z=OUTR;
  always @(reset_==0) #1 begin OUTR<=0; STAR<=S0; end
  always @(posedge clock) if (reset_==1) #3
   casex(STAR)
    S0: begin OUTR<=0; STAR<=(x1_x0=='B11)?S1:S0; end
    S1: begin OUTR<=0; STAR<=(x1_x0=='B01)?S2:(x1_x0=='B11)?S1:S0; end
    S2: begin OUTR<=(x1_x0=='B10)?1:0; STAR<=(x1_x0=='B11)?S1:S0; end
   endcase
endmodule
```

x_1x_0	00	01	11	10
S0	S0/0	S0/0	S1/0	S0/0
S1	S0/0	S2/0	S1/0	S0/0
S2	S0/0	S0/0	S1/0	S0/1

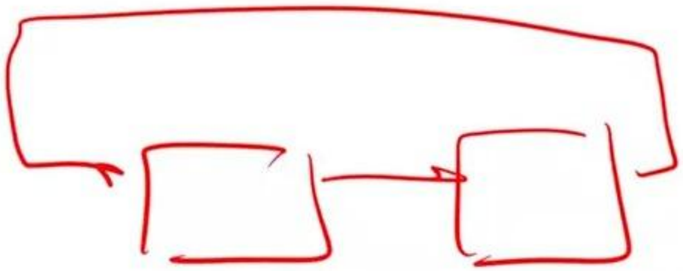
Esempio: temporizzazione

```
S0: begin OUTR<=0; STAR<=(x1_x0=='B11)?S1:S0; end
S1: begin OUTR<=0; STAR<=(x1_x0=='B01)?S2:(x1_x0=='B11)?S1:S0; end
S2: begin OUTR<=(x1_x0=='B10)?1:0; STAR<=(x1_x0=='B11)?S1:S0; end
```

- L'uscita è un clock in ritardo rispetto all'ingresso che la genera
  - Per via del registro OUTR

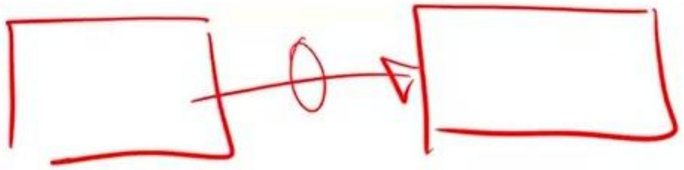


Modello di Mealy ritardato (cont.)

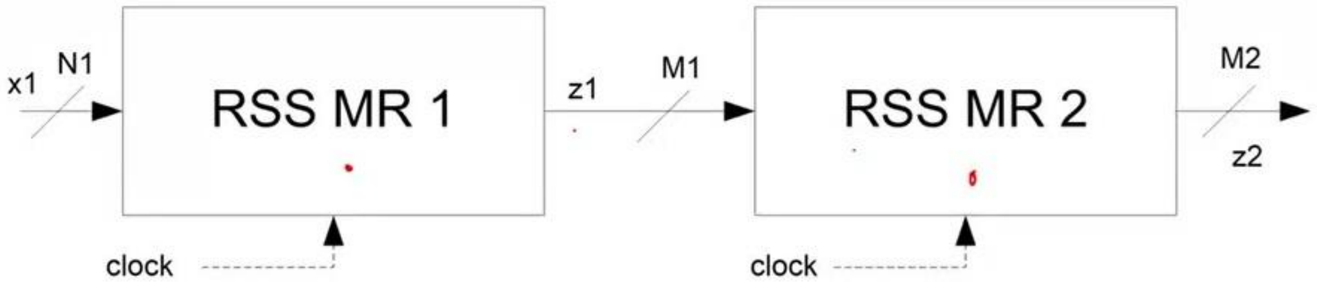


- Le reti di Mealy ritardato sono **non trasparenti**
  - Posso montarle in qualunque configurazione senza problemi

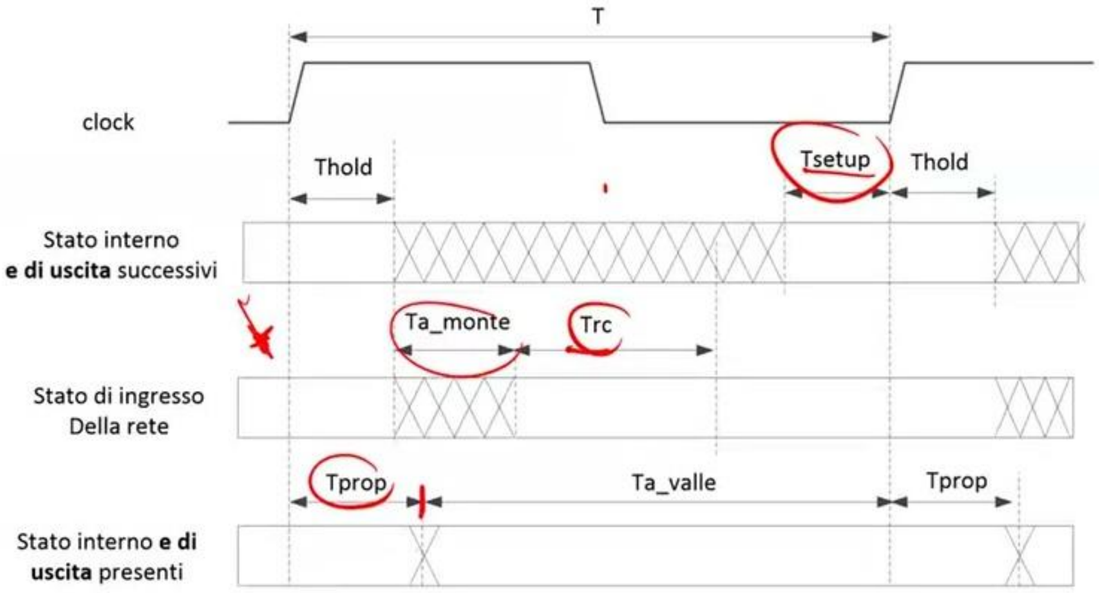
- Le uscite sono costanti per un intero periodo di clock
  - Posso montare **catene** di reti di Mealy ritardato con lo stesso clock **arbitrariamente lunghe**
  - Non avrò mai **problemi di temporizzazione**



Esempio



- Lo stato di ingresso di MR2 (stato di uscita di MR1) deve essere pronto $T_{RC_2} + T_{setup}$ prima del clock.
- Lo stato di uscita di MR1 è pronto **già** T_{prop} **dopo il fronte del clock**.
- Basta che $T \geq T_{prop} + T_{RC_2} + T_{setup}$ perché MR1 possa pilotare MR2
- Ma **questa disuguaglianza è vera**
  - è infatti quella che regola il percorso reg->reg in MR2



Riepilogo sulle reti di Mealy Ritardato

- **sono non trasparenti;**
- hanno una legge B flessibile, che mi porta in genere a risolvere problemi usando un **numero minore di stati interni**;
- hanno **uscite stabili**, che cambiano in tempi certi
- **non sono rallentate** da percorsi combinatori troppo lunghi
  - ricordare le disuguaglianze di temporizzazione
- possono essere **montate in cascata** senza problemi di pilotaggio se hanno lo stesso clock
- le uscite possono essere **connesse in reazione** senza problemi di stabilità