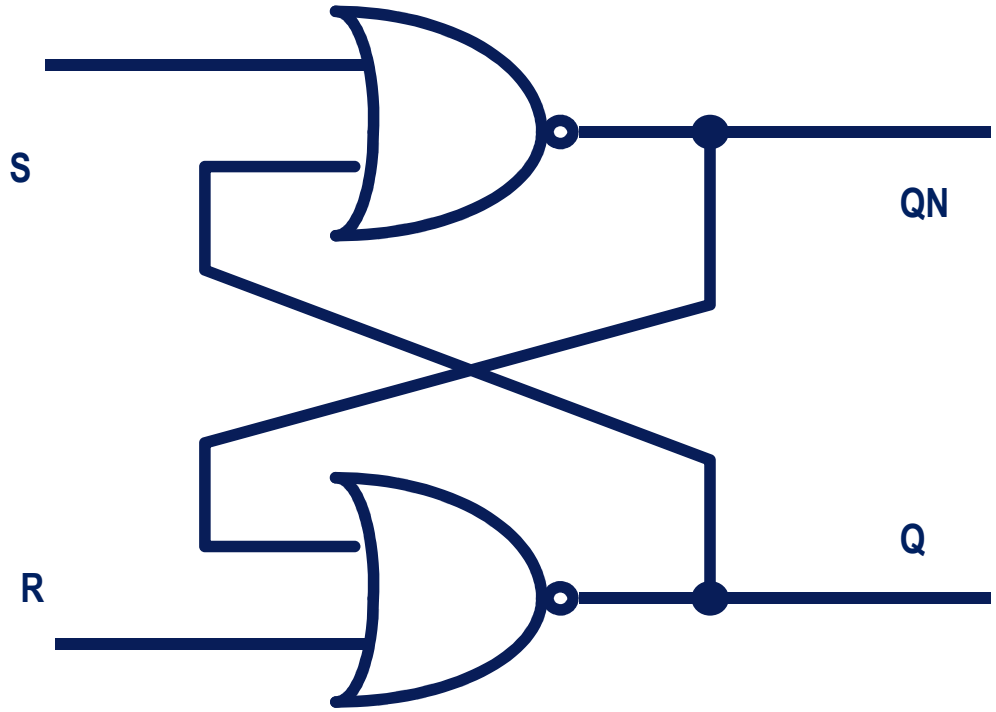
LEZIONE N° 94

Reti Logiche Sequenziali
(parte seconda)

Latch SR in Verilog

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



RAM 256x4 in Verilog

Cortesia da Prof. P. Corsini

```
module Ram(data, address, s_, mr_, mw_);
  input [7:0]  address;
  inout [3:0]  data;
  input        s_;
  input        mr_;
  input        mw_;
    reg  [3:0] LOCATION[0:255]; // 256 memory cells
    wire alfa=~(s_|mr_);
    wire beta=~(s_|mw_);

  assign data=(alfa==1)? LOCATION[address]:'BZZ ;    //READ
  always @(mw_ or data)if(beta==1) LOCATION[address]= data; //WRITE

endmodule
```

ROM 4x8

Cortesia da Prof. P. Corsini

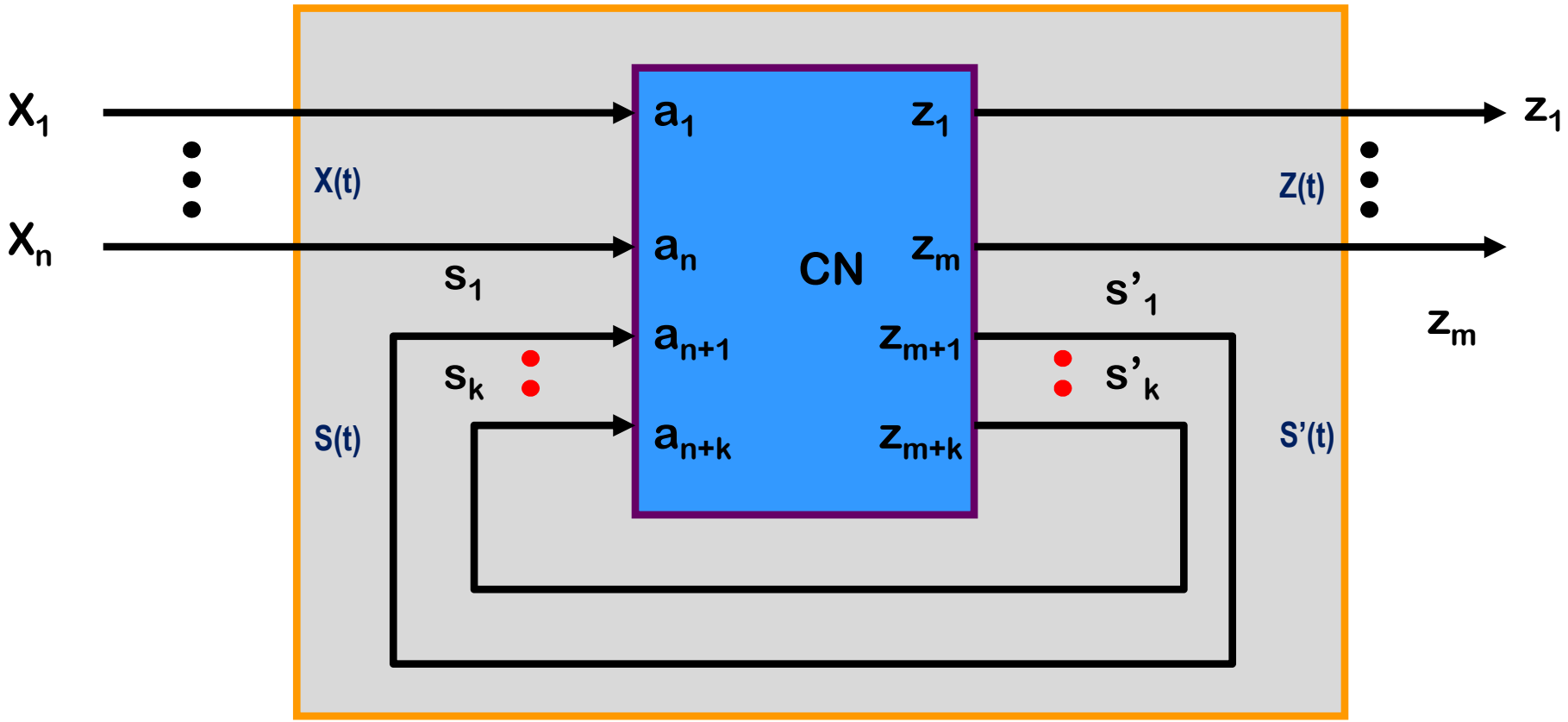
```
module Eprom(d7_d0, address, s_, oe_);
  input  [1:0]  address;
  output [7:0]  d7_d0;
  input         s_;
  input         oe_;

  wire alfa=~(s_|oe_);
  assign d7_d0=(alfa==0)? 'BZZ : LOCATION(address);

   function [7:0] LOCATION;
    input [1:0] address;
    casex(address)
     0:      LOCATION = 0;
     1:      LOCATION = 1;
     2:      LOCATION = 2;
     3:      LOCATION = 3;
    endcase
   endfunction
endmodule
```

Macchina di Mealy asincrona

- Le variabili d'uscita, in un determinato istante, sono funzione del valore degli ingressi e delle variabili di stato

$$Z(t)=F(\ X(t),\ S(t)\ )$$

$$S'(t)=G(\ X(t),\ S(t)\ )$$



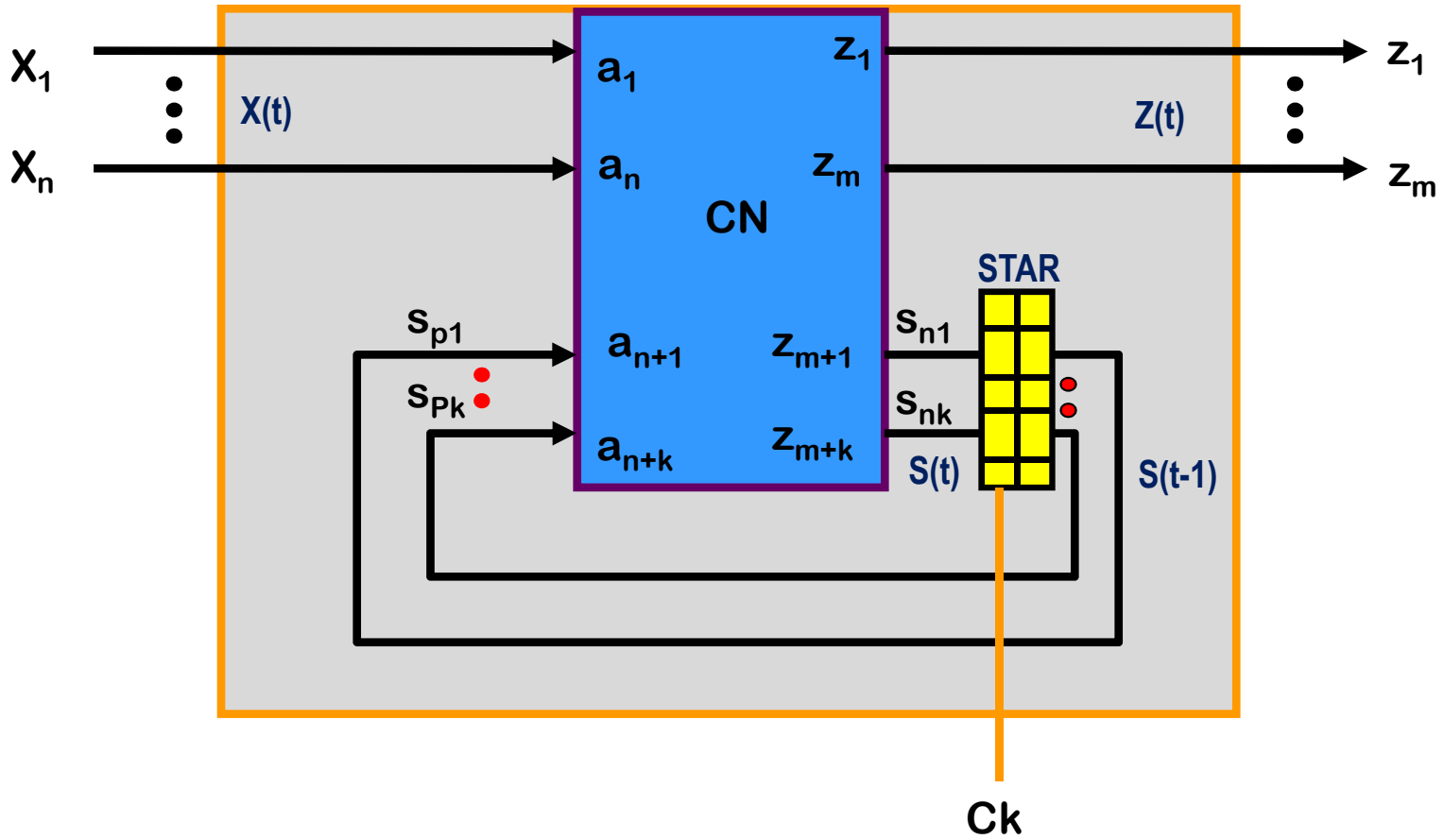
La rete CN è una rete combinatoria

Macchina di MEALY (sincronizzata)

- Le uscite sono funzioni delle variabili di stato e degli ingressi

$$Z(t)=f(\ X(t),\ S(t-1)\ )$$

$$S(t)=g(\ X(t),\ S(t-1)\ )$$



STAR e' costituito da Flip-Flop D

Macchina di MOORE (sincronizzata)

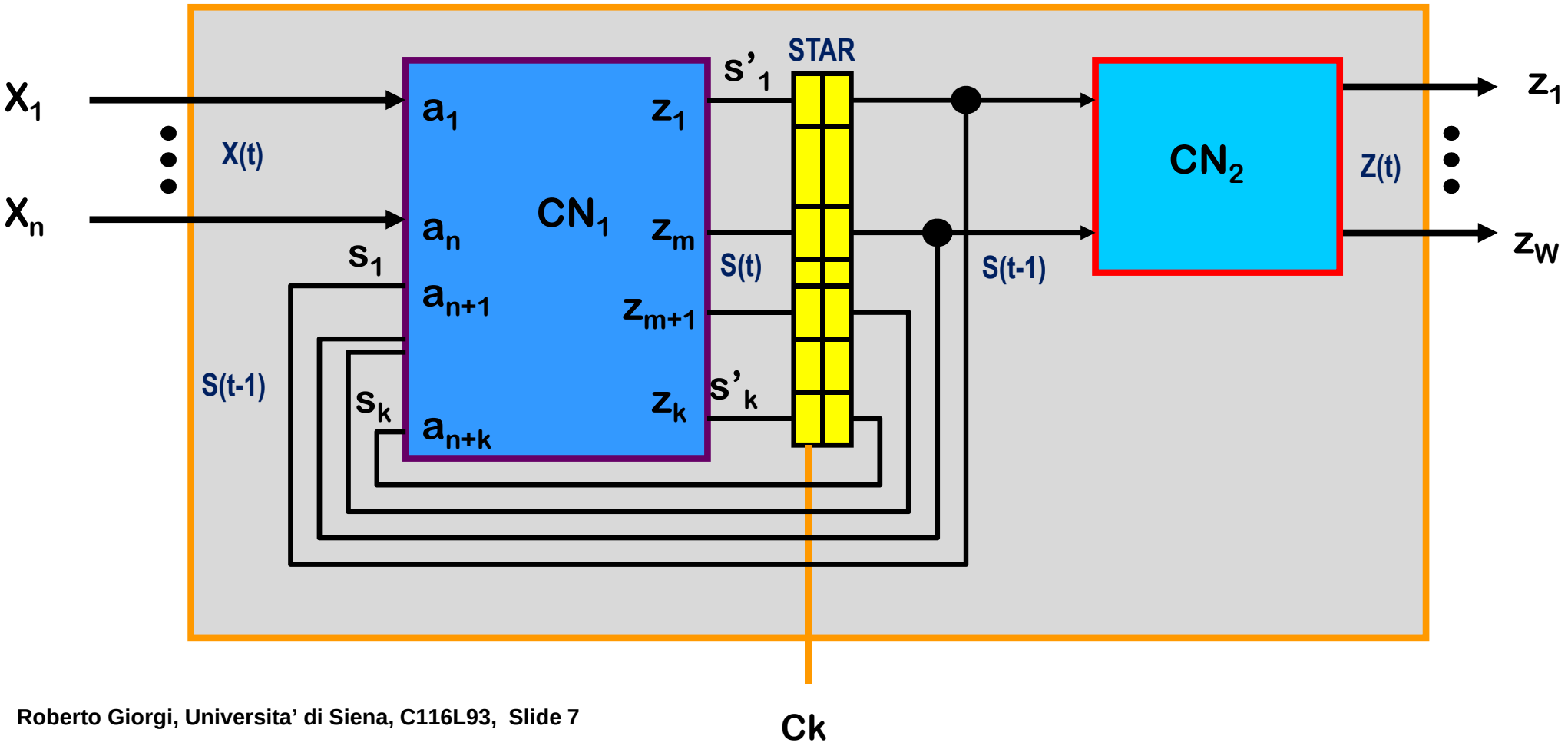
- Le variabili d'uscita, in un determinato istante, sono funzione del sole variabili di stato

$$Z(t)=f(\ S(t-1)\ )$$

$$S(t)=g(\ X(t),\ S(t-1)\ )$$

In MOORE: l'uscita dipende solo dallo stato

S(t-1) e' lo stato presente
S(t) e' lo stato successivo



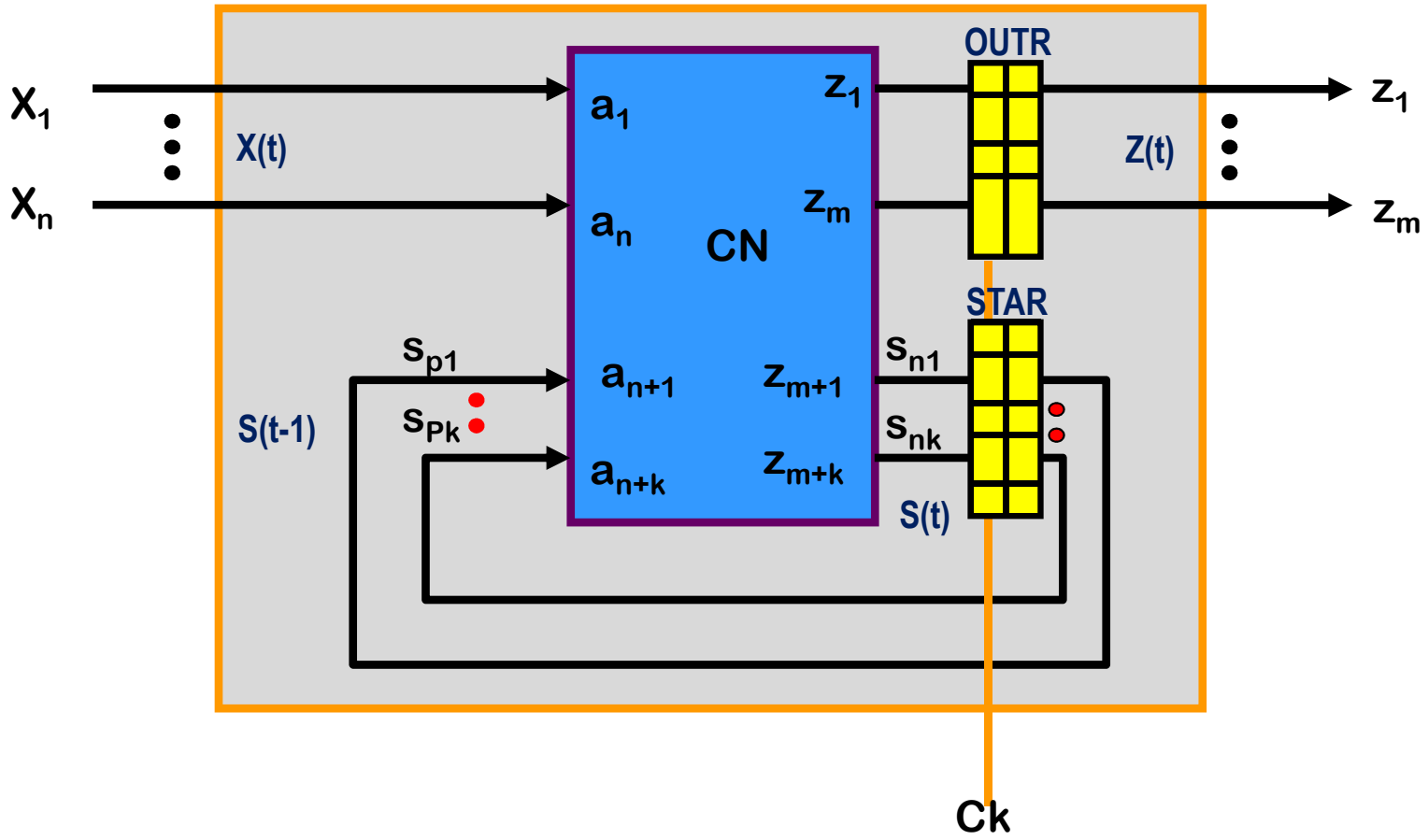
Macchina di Mealy Ritardata (sincronizzata)

- Le uscite sono funzioni delle variabili di stato e degli ingressi, ma risultano sincronizzate

$$Z(t)=f(\ X(t-1),\ S(t-1)\ )$$

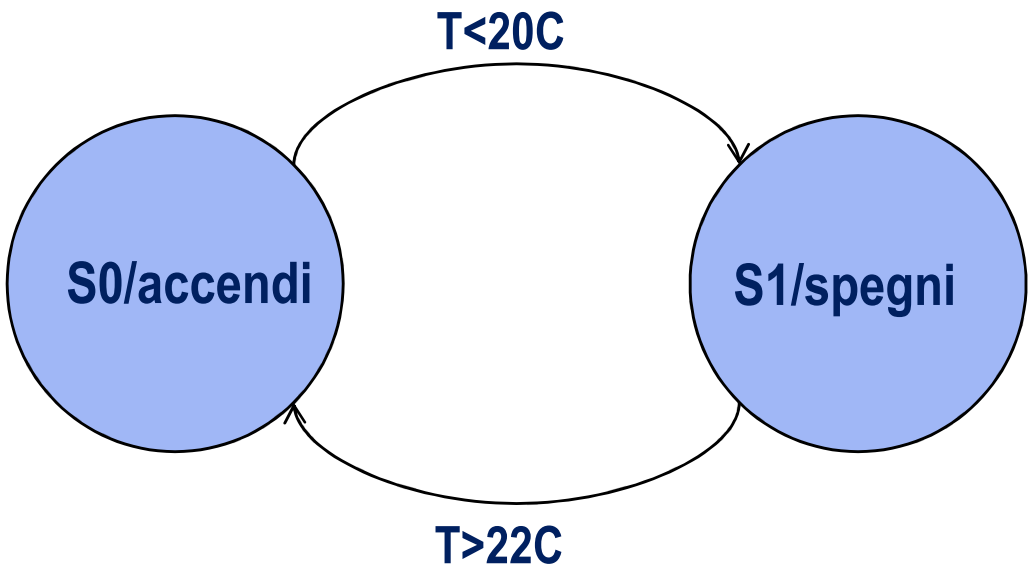
$$S(t)=g(\ X(t),\ S(t-1)\ )$$

In MEALY: l'uscita dipende sia dallo stato che dagli ingressi



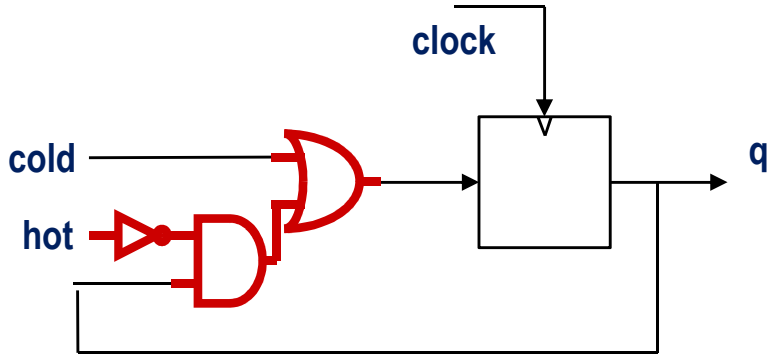
Specifica di una FSM per implementazione con Moore

- S0 → Riscaldamento Acceso
- S1→ Riscaldamento Spento



cold/hot \ CN1	00	01	11	10
Off (0)	0	0	-	1
On (1)	1	0	-	1

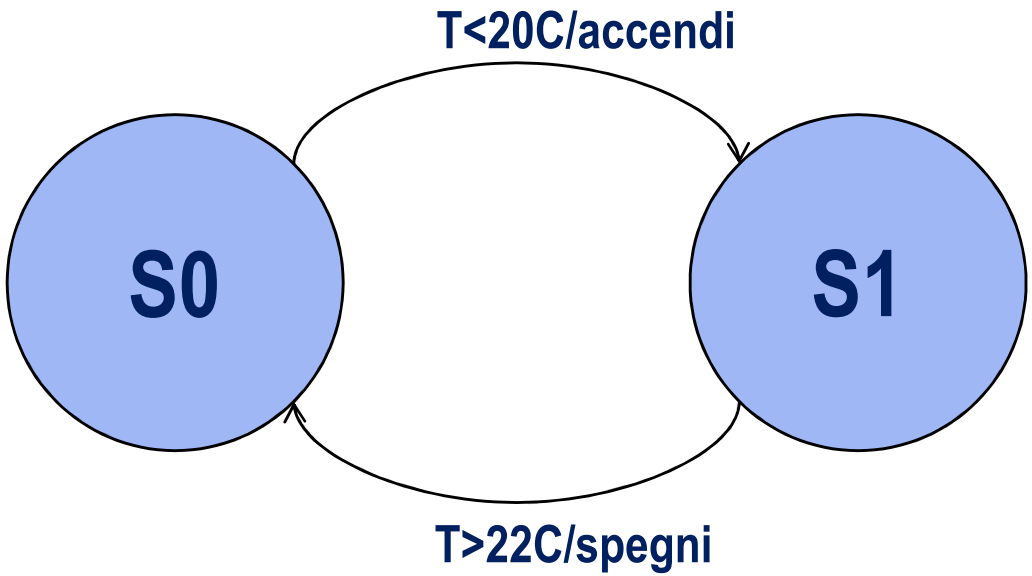
$qnext = cold + \overline{hot} * q$



In genere MOORE produce piu' stati, ma si puo' implementare usando solo LUT

Specifica di una FSM per implementazione con Mealy

- S0 → Riscaldamento Spento
- S1→ Riscaldamento Acceso



FARE PER ESERCIZIO

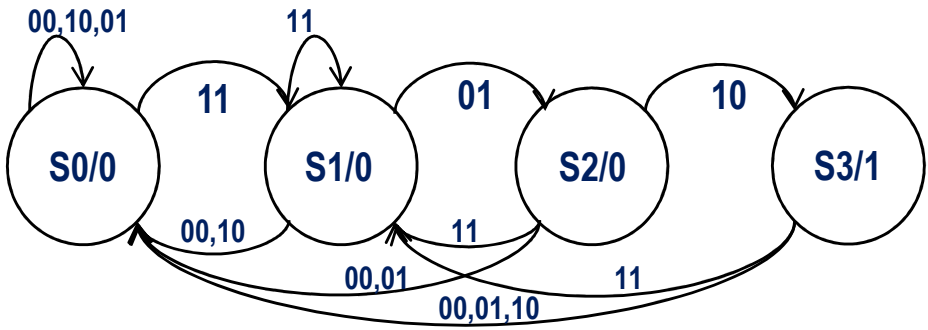
In genere MEALY produce meno stati, ma puo' richiedere sia LUT che logica aggiuntiva

Template generale per Moore in Verilog

```
module Moore(z,x,clock,reset_);
  input           clock, reset_;
  input  [N-1:0]  x;
  output [M-1:0]  z;
  reg    [W-1:0]  STAR;
  parameter S0=codifica0, …, SKmeno1=codificaKmeno1;
  assign z=(STAR==S0)?codificaf0:
           (STAR==S1)?codificaf1:
            …
           /*(STAR==SKmeno1)*/ codificafKmeno1;
  always @(reset_==0) #1 STAR<=stato_iniziale;
  always @(posedge clock) if (reset_==1) #3
    casex(STAR)
      S0: STAR<=g0(x);
      S1: STAR<=g1(x);
       …
      SKmeno1: STAR<=gKmeno1(x);
    endcase
endmodule
```

Esempio – riconoscitore di sequenza 11,01,10 con Moore

```
module ricseq110110moore(z,x,clock,reset_);
  input          clock, reset_;
  input  [1:0]    x;
  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<=(x=='B11)?S1:S0;
      S1: STAR<=(x=='B01)?S2:(x=='B11)?S1:S0;
      S2: STAR<=(x=='B10)?S3:(x=='B11)?S1:S0;
      S3: STAR<=(x==`B11)?S1:S0;
  endcase
endmodule
```

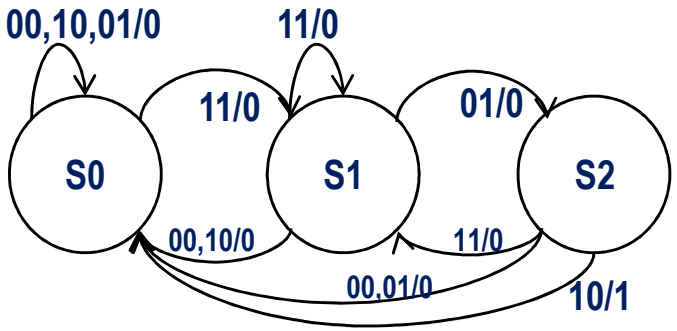


Template generale per Mealy in Verilog

```
module Mealy(z,x,clock,reset_);
  input           clock, reset_;
  input  [N-1:0]  x;
  output [M-1:0]  z;
  reg [W-1:0]     STAR;
  parameter  S0=codifica0, …, SKmeno1=codificaKmeno1;
  assign z=(STAR==S0)?f0(x):
           (STAR==S1)?f1(x):
            …
           /*(STAR==SKmeno1)*/ fKmeno1(x);
  always @(reset_==0) #1 STAR<=stato_iniziale;
  always @(posedge clock) if (reset_==1) #3
    casex(STAR)
      S0: STAR<=g0(x);
      S1: STAR<=g1(x);
       …
      SKmeno1: STAR<=gKmeno1(x);
    endcase
endmodule
```

Esempio – riconoscitore di sequenza 11,01,10 con Mealy

```
module ricseq110110mealy(z,x,clock,reset_);
  input           clock, reset_;
  input  [1:0]     x;
  output           z;
  reg [1:0]        STAR;
  parameter  S0=‘B00, S1=`B01, S2=`B10;
  assign z=((STAR==S2)&(x==`B10))?1:0;
  always @(reset_==0) #1 STAR<=S0;
  always @(posedge clock) if (reset_==1) #3
    casex(STAR)
      S0: STAR<=(x==‘B11)?S1:S0;
      S1: STAR<=(x==‘B01)?S2:(x==‘B11)?S1:S0;
      S2: STAR<=(x==‘B11)?S1:S0;
  endcase
endmodule
```

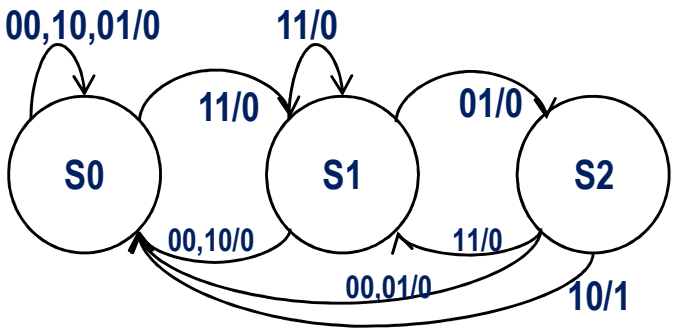


Template generale per Mealy Ritardato in Verilog

```
module MealyRitardato(z,x,clock,reset_);
  input           clock, reset_;
  input  [N-1:0]  x;
  output [M-1:0]  z;
  reg    [W-1:0]  STAR;
  reg    [M-1:0]  OUTR;
  parameter  S0=codifica0, …, SKmeno1=codificaKmeno1;
  assign     z=OUTR;
  always @(reset_==0) #1 begin STAR<=…; OUT<=…; end;
  always @(posedge clock) if (reset_==1) #3
    casex(STAR)
      S0: begin OUTR<=f0(x); STAR<=g0(x); end
      S1: begin OUTR<=f1(x); STAR<=g1(x); end
       …
      SKmeno1: begin OUTR<=fKmeno1(x); STAR<=gKmeno1(x); end
    endcase
endmodule
```

Esempio – riconoscitore di sequenza 11,01,10 con Mealy Rit

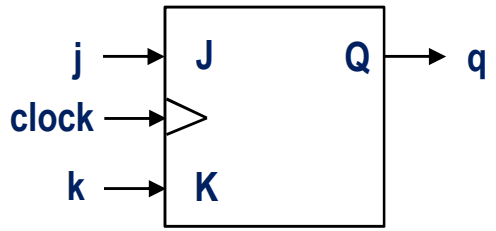
```
module ricseq110110mealyritardato(z,x,clock,reset_);
  input           clock, reset_;
  input  [1:0]     x;
  output           z;
  reg [1:0]        STAR;
  reg              OUTR;
  parameter  S0=‘B00, S1=`B01, S2=`B10;
  assign     z=OUTR;
  always @(reset_==0) #1 begin; OUR<=0; STAR<=S0; end
  always @(posedge clock) if (reset_==1) #3
    casex(STAR)
      S0: OUTR<=0; STAR<=(x==‘B11)?S1:S0; end
      S1: OUTR<=0; STAR<=(x==‘B01)?S2:(x==‘B11)?S1:S0; end
      S2: OUTR<=(x==‘B10)?1:0; STAR<=(x==‘B11)?S1:S0; end
  endcase
endmodule
```



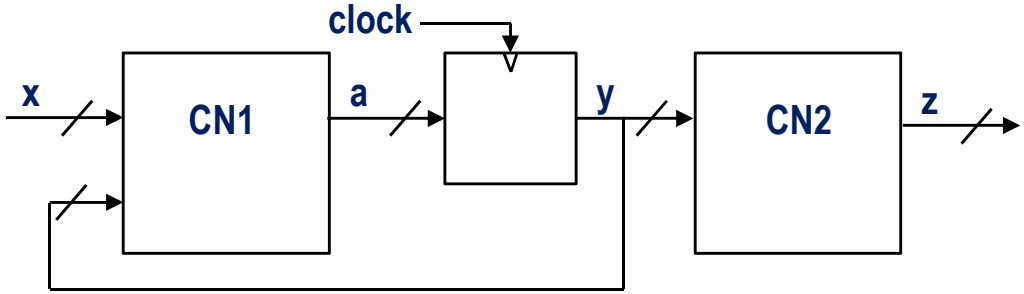
Flip-Flop JK

- Il FF-JK e' tale che in corrispondenza del fronte in salita del clock:
- per JK=10 l'uscita va a 1,
  per JK=01 l'uscita va a 0,
  per JK=00 il FF conserva (l'uscita resta inalterata)
  per JK=11 il FF commuta (l'uscita diventa il suo negato)

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



Sintesi del FF-JK con Moore



- 2 stati → STAR ha 1 bit
- → a ha 1 bit, y ha 1 bit
  z ha 1 bit, (x ha 2 bit)

per JK=10 l'uscita va a 1,
per JK=01 l'uscita va a 0,
per JK=00 il FF conserva
per JK=11 il FF commuta

Tabella delle transizioni e delle uscite

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

Stato	y0
S0	0
S1	1

CN1

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

a

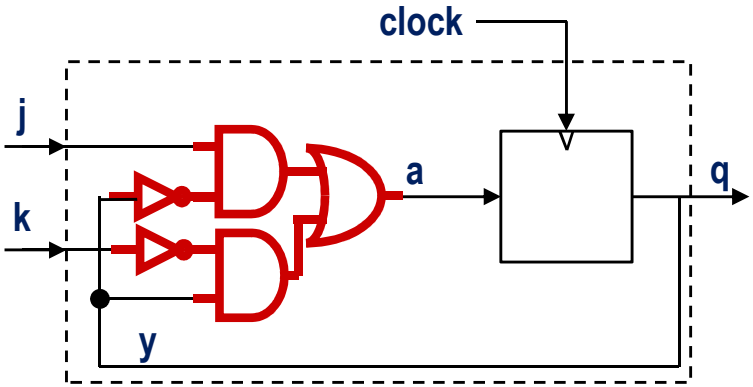
$a = j\,\overline{y} + \overline{k}\,y$

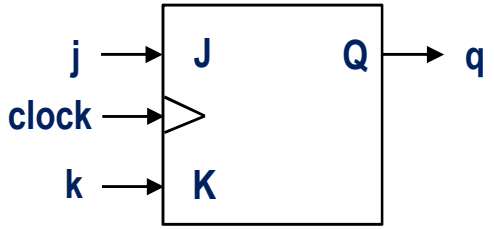
CN2

y	0	1
	0	1

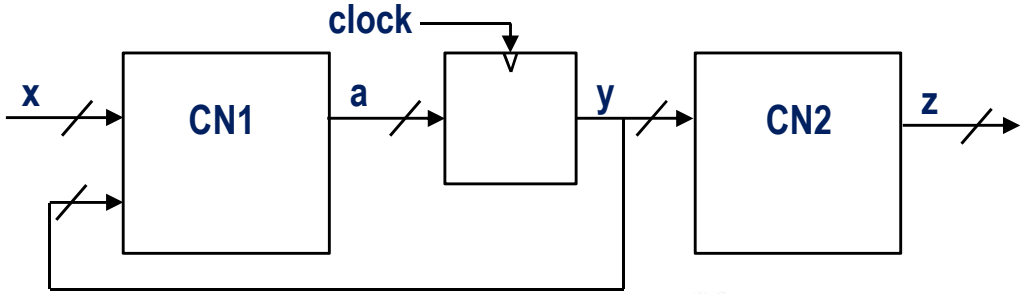
z

$z = y$



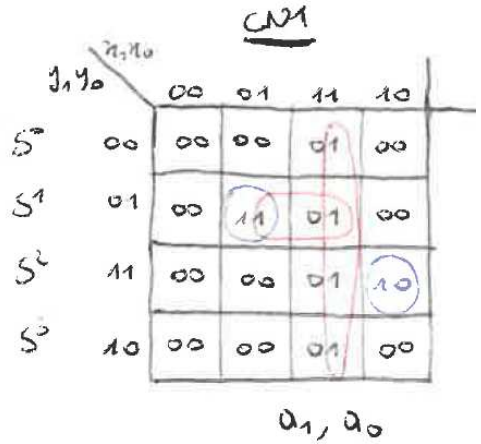


Sintesi del riconoscitore di sequenza 11,01,10 con Moore



- 4 stati → STAR ha 2 bit
- → a ha 2 bit, y ha 2 bit
  z ha 1 bit, (x ha 2 bit)

Stato	y1 y0
S0	00
S1	01
S2	11
S3	10



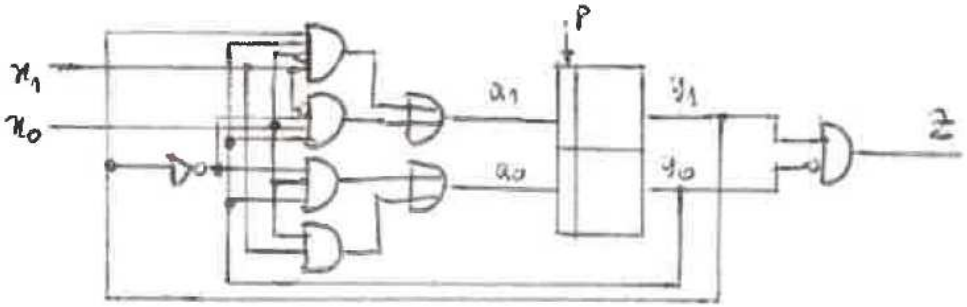
$$a_1 = \bar{x}_1 x_0 \bar{y}_1 y_0 + x_1 \bar{x}_0 y_1 y_0$$

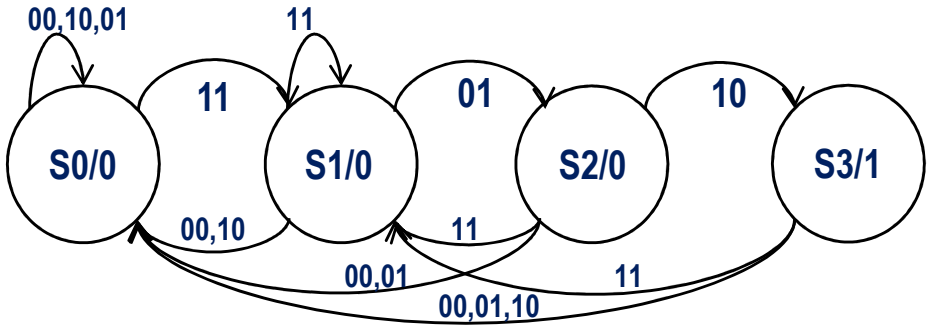
$$a_0 = x_1 x_0 + x_0 \bar{y}_1 y_0$$

CN2

y_1 y_0	z
00	0
01	0
11	0
10	1

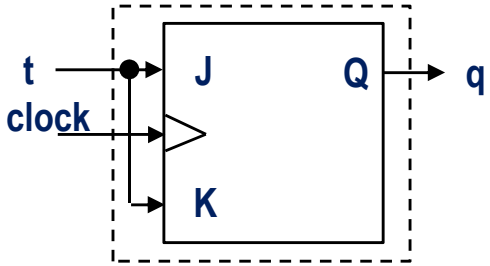
$z = y_1\bar{y}_0$



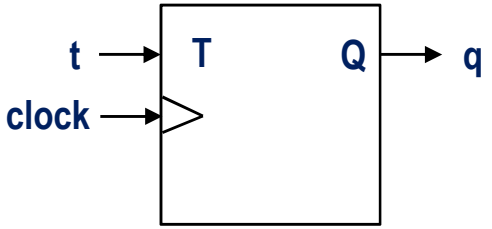


Flip-Flop T (detto FF contatore)

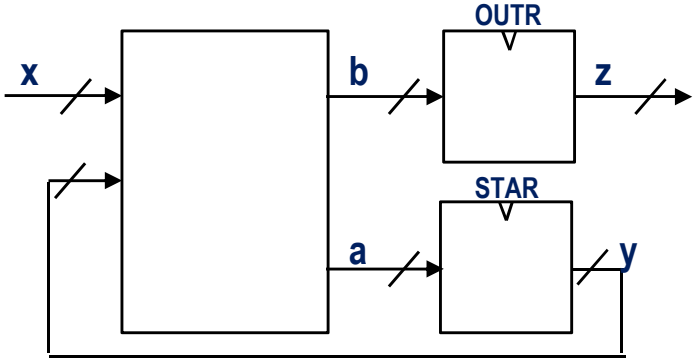
- Il FF-T e' tale che in corrispondenza del fronte in salita del clock:
- per T=0 il FF conserva
  (l'uscita rimane inalterata)
  per T=1 il FF commuta
  (l'uscita diventa il suo negato)




```
module FFT(q,t,clock,reset_);
  input           clock, reset_;
  input            t;
  output           q;
  reg              STAR;
  parameter  S0=0, S1=1;
  assign q=(STAR==S0)?0:1;
  always @(reset_==0) #1 STAR<=stato_iniziale;
  always @(posedge clock) if (reset_==1) #3
    casex(STAR)
      S0: STAR<=(t==0)?S0:S1;
      S1: STAR<=(t==0)?S1:S0;
    endcase
endmodule
```



Sintesi del riconoscitore di sequenza 11,01,10 con Mealy Rit



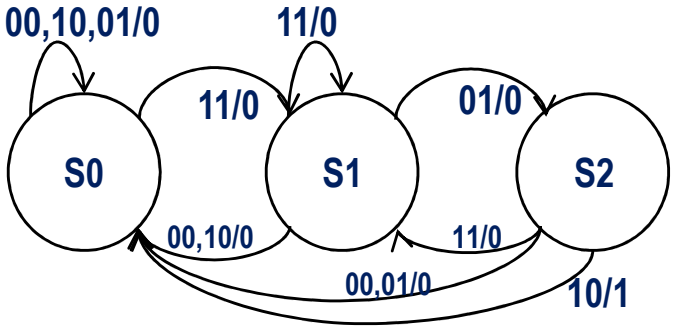


Tabella delle transizioni

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

Stato	y1y0
S0	00
S1	11
S2	01

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

a_1a_0

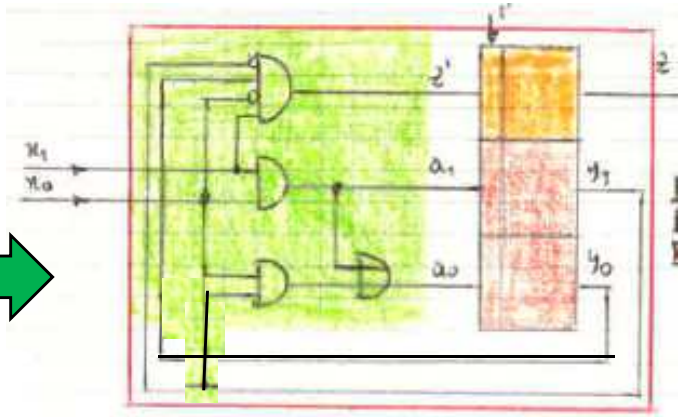
y_1y_0 \ x_1x_0	00	01	11	10
00	0	0	0	0
01	0	0	0	1
11	0	0	0	0
10	-	-	-	-

b_0

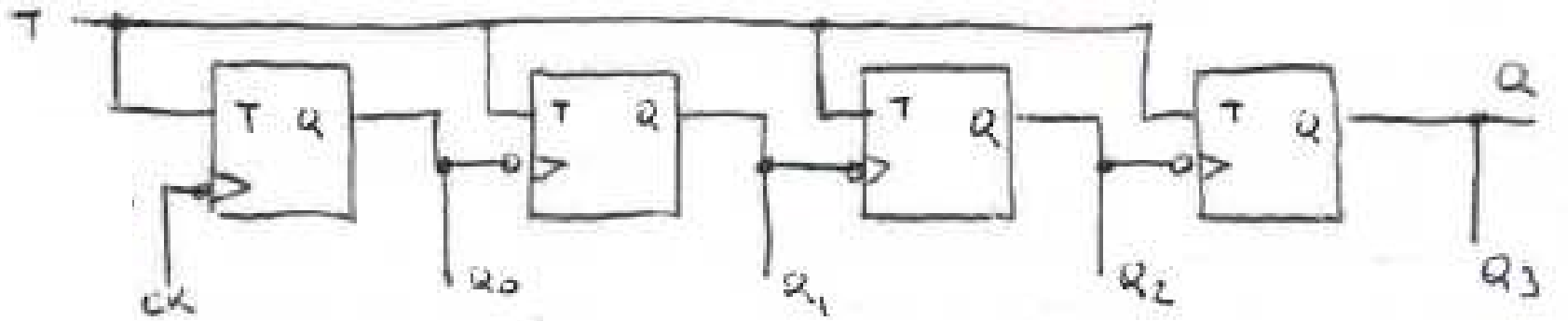
$a1 = x1\ x0$

$a0 = x1\ x0 + x0\ y1$

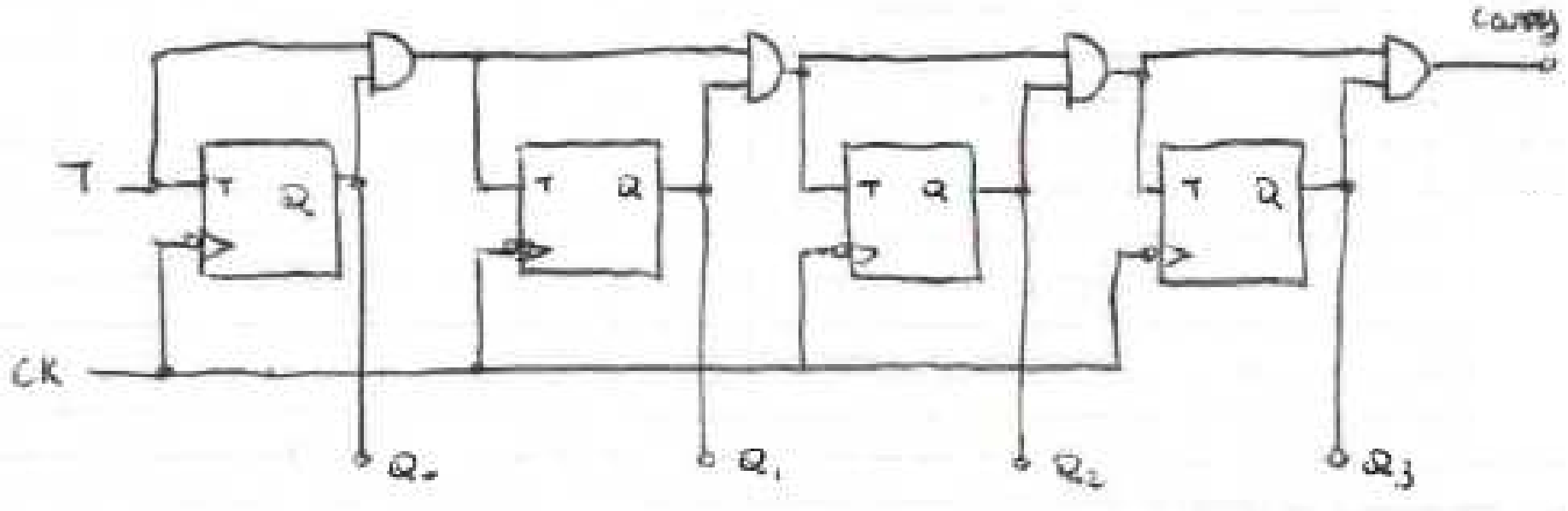
$b0 = x1\ \overline{x0}\ \overline{y1}\ y0$



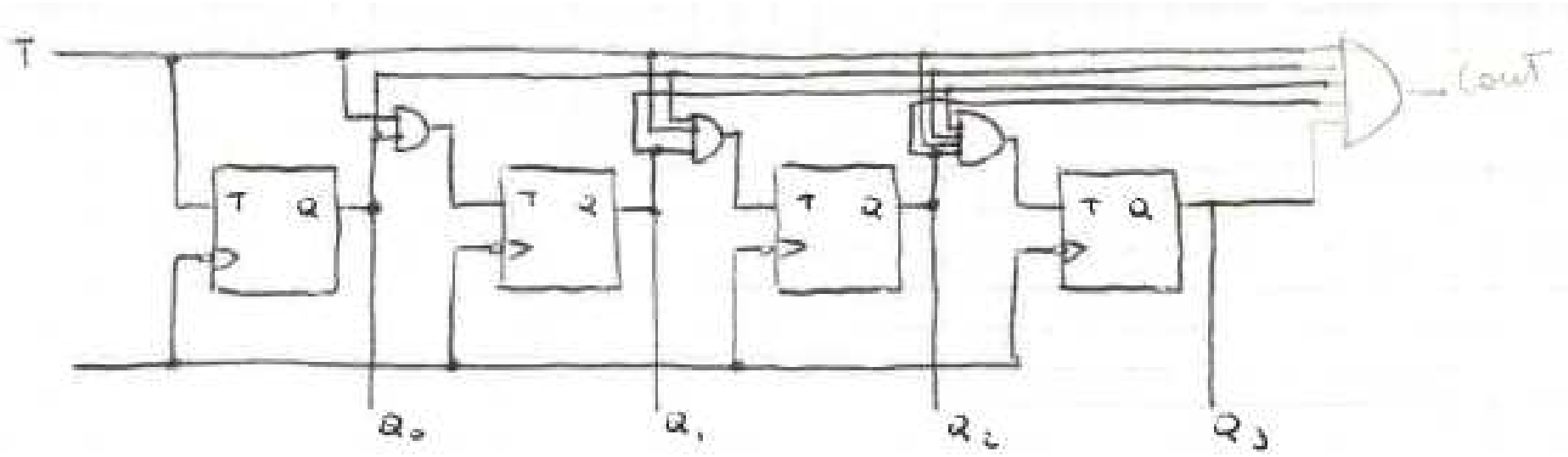
Ripple Counter



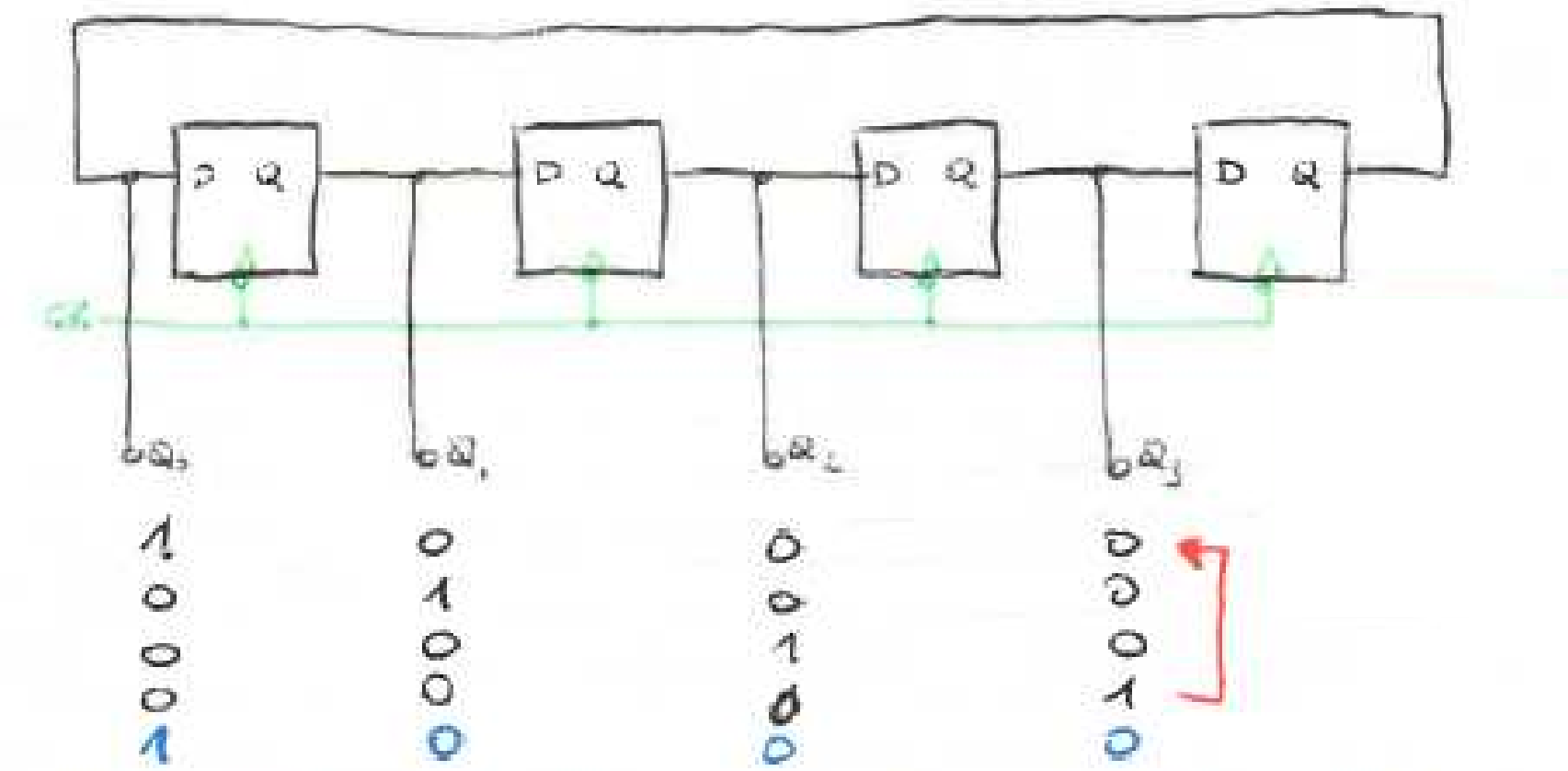
Serial Carry Counter



Parallel Carry Counter



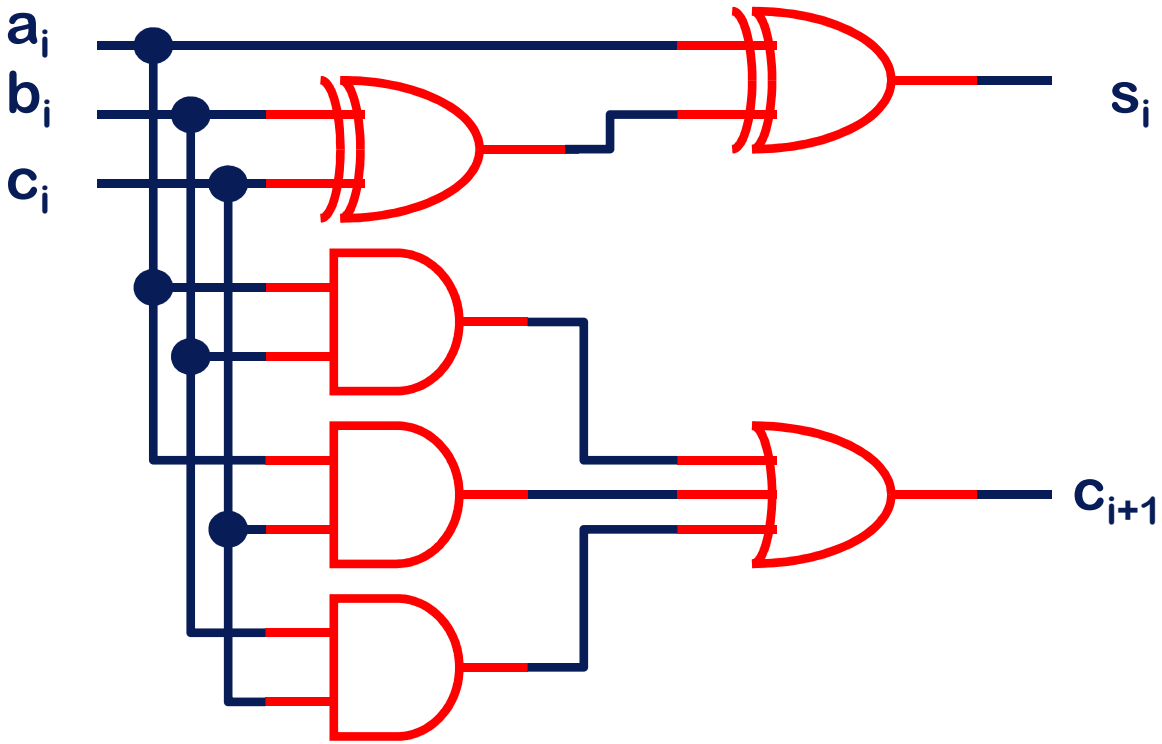
Ring Counter

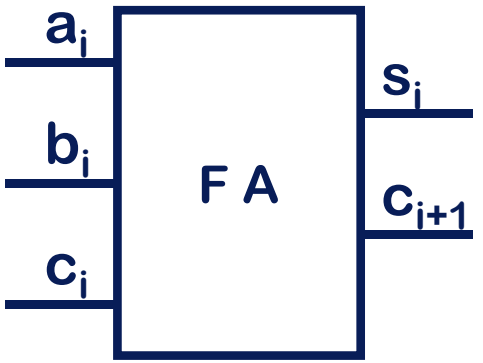


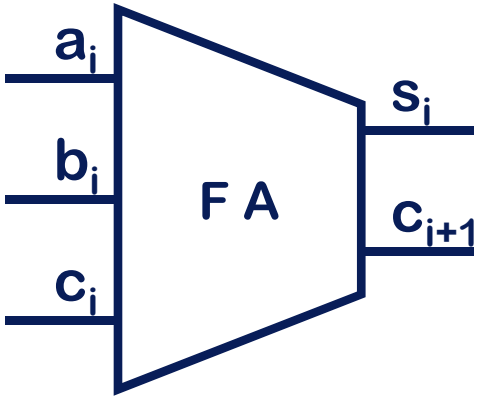
Full Adder (richiamo)

$$s_i = a_i \oplus b_i \oplus c_i = a_i \oplus (b_i \oplus c_i)$$

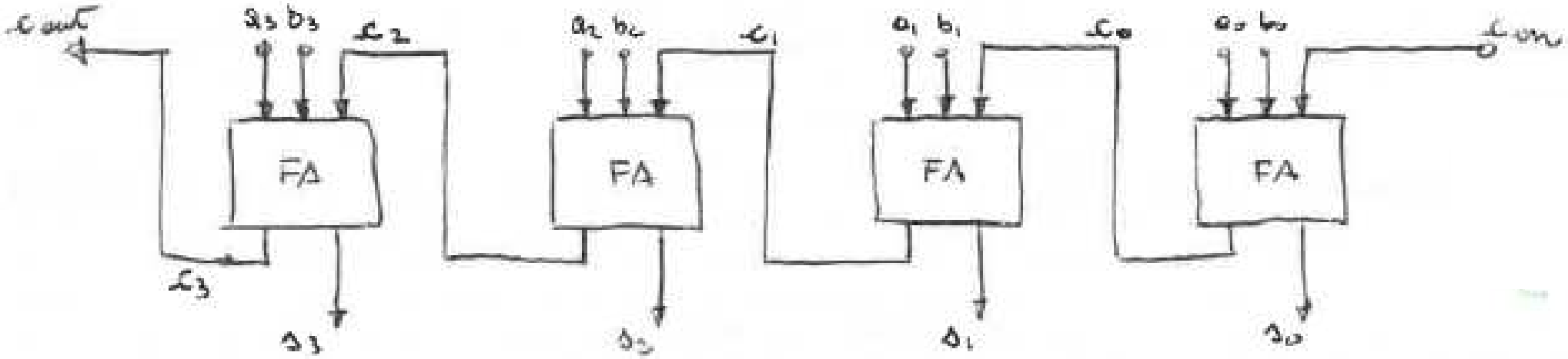
$$c_{i+1} = a_i \cdot b_i + a_i \cdot c_i + b_i \cdot c_i$$







Sommatore Parallelo con riporto seriale



$$\tau_{cout} = N \cdot \tau_C$$

$$\tau_{tot} = (N-1) \cdot \tau_C + \tau_{FA}$$

N=numero di bit del sommatore
τ_C = ritardo per generare il carry
τ_{FA} = ritardo del Full-Adder singolo
τ_{tot} = ritardo totale del sommatore

Carry Look-Ahead Adder

$$s_i = a_i \oplus b_i \oplus c_{i-1}$$

$$c_i = a_i b_i + a_i c_{i-1} + b_i c_{i-1} = a_i b_i + c_{i-1}(a_i + b_i) = a_i b_i + c_{i-1}(a_i \oplus b_i)$$

ai+bi si puo' sostituire col suo xor perche' nell'espressione quando ai=1 e bi=1 anche ai*bi=1

$$\left.\begin{aligned} G_i &\triangleq a_i b_i \\ P_i &\triangleq a_i \oplus b_i \end{aligned}\right] \Rightarrow \begin{aligned} s_i &= P_i \oplus c_{i-1} \\ c_i &= G_i + P_i \cdot c_{i-1} \end{aligned}$$

Gi=generate
Pi=propagate

$$c_0 = G_0 + P_0 \cdot c_{in}$$

$$c_1 = G_1 + P_1 \cdot c_0 = G_1 + G_0 P_1 + P_1 P_0 \cdot c_{in}$$

$$c_2 = G_2 + P_2 c_1 = G_2 + G_1 P_2 + G_0 P_1 P_2 + P_2 P_1 P_0 \cdot c_{in}$$

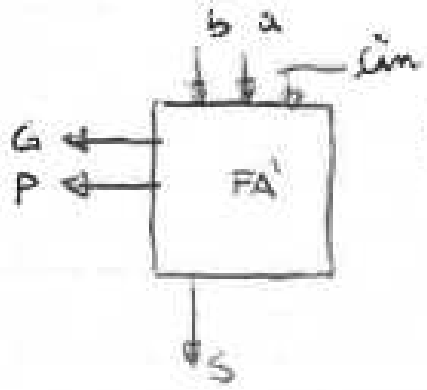
$$c_3 = G_3 + P_3 \cdot c_2 = G_3 + G_2 P_3 + G_1 P_2 P_3 + G_0 P_1 P_2 P_3 + P_3 P_2 P_1 P_0 c_{in}$$

Carry Look-Ahead Adder (2)

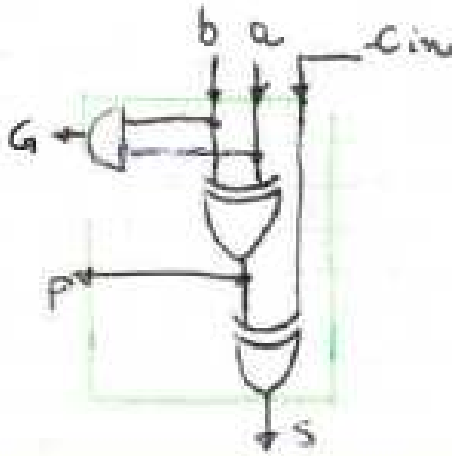
C_{i-1}	a_i	b_i	s_i	c_i
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

in questi due casi il carry in uscita è presente perché "si genera" a causa dei "2 uno"

in questi due casi il carry in uscita è presente perché "si propaga" dal F. A. precedente !!

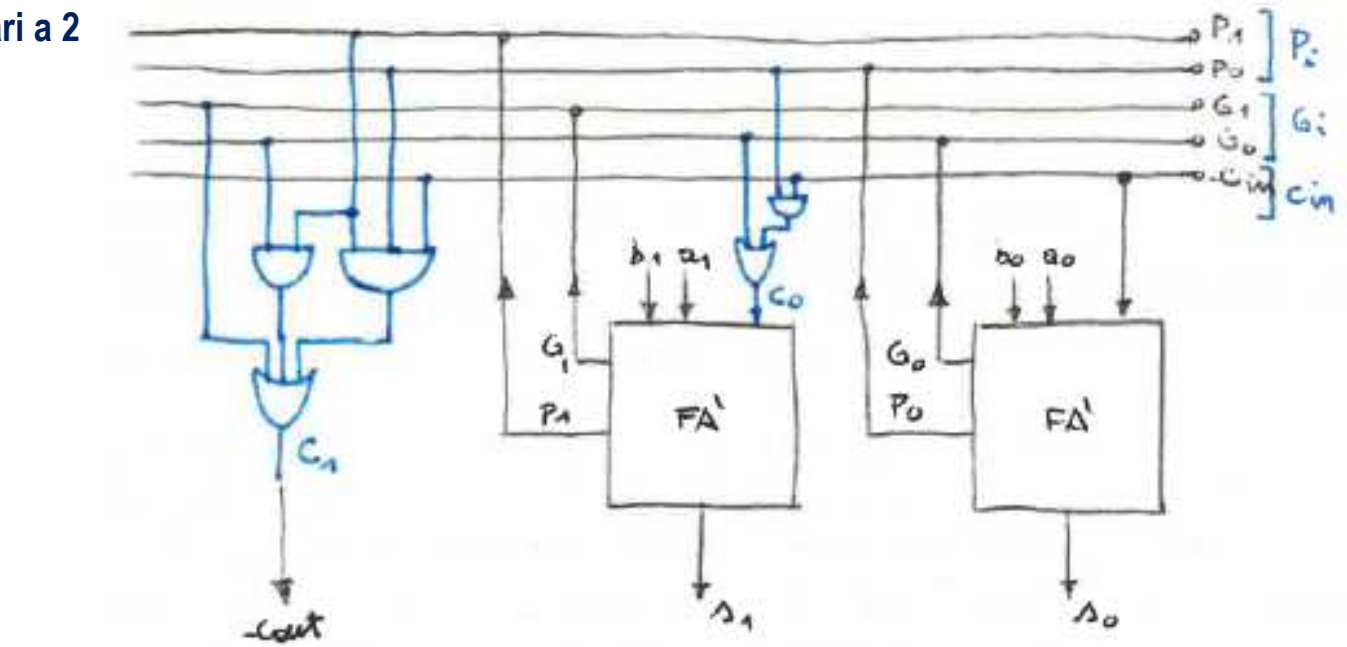


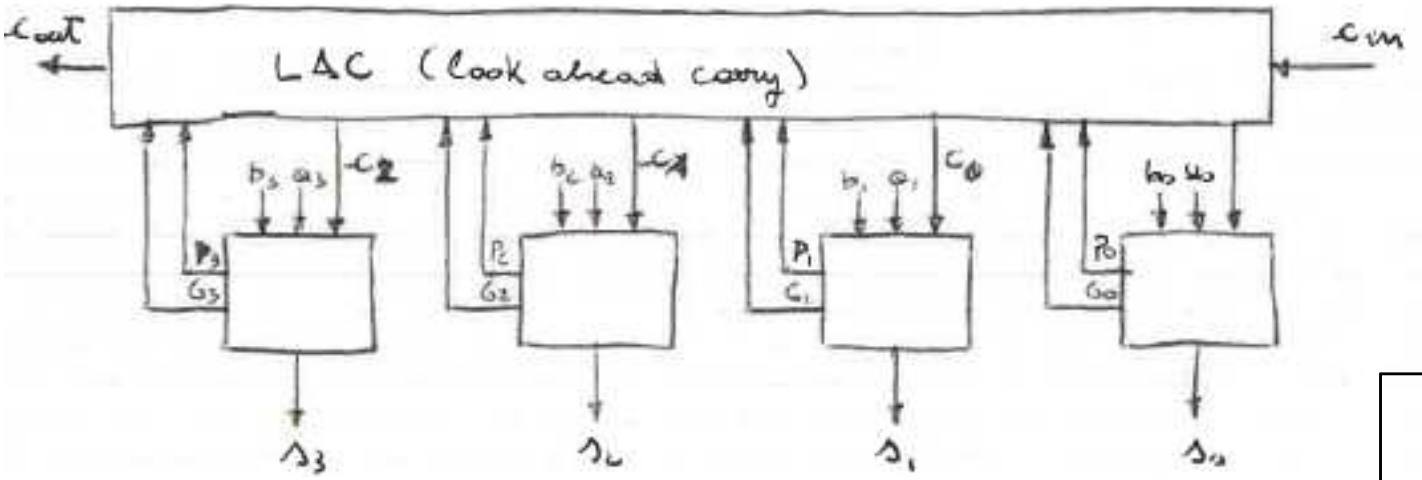
in cui la rete FA' è



Carry Look-Ahead Adder (3)

Es. Nel caso pari a 2





$$\tau_{cout} = \tau_{LAC}$$

$$\tau_{tot} = \tau_{LAC} + \tau_{FA'}$$