ELETTRONICA

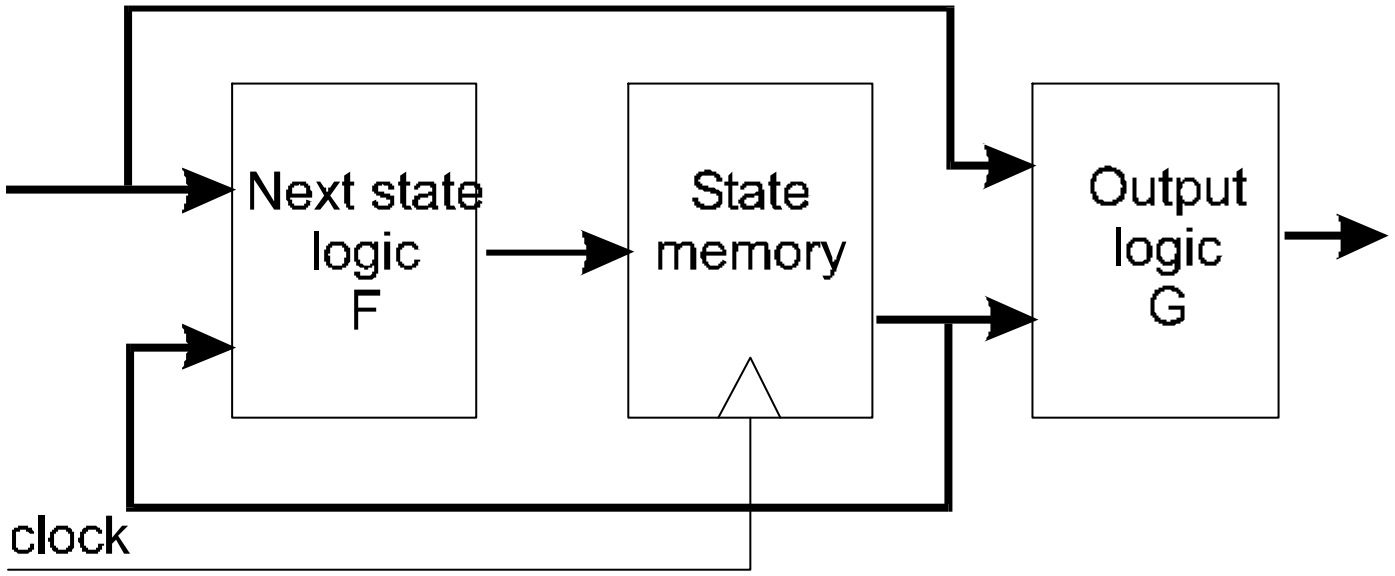
Macchine a stati finiti

Prof. Luigi Raffo
Dipartimento di Ingegneria Elettrica ed Elettronica
Università di Cagliari

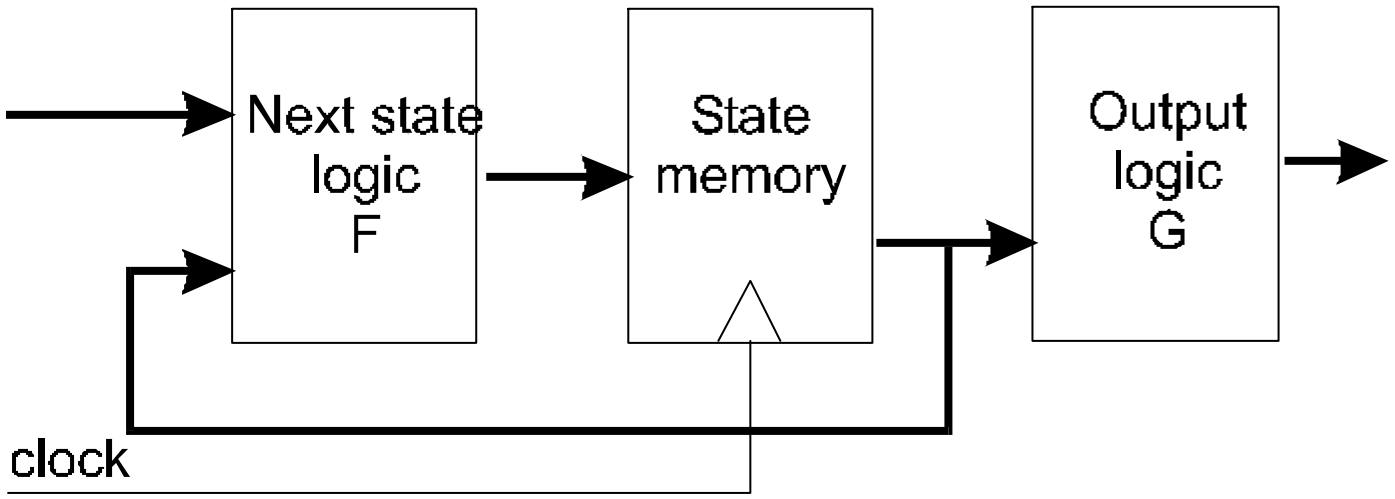
Modelli di macchine a stati

Una rete sequenziale sincrona puo` essere quindi rappresentata come un insieme di registri, una rete combinatoria che ne aggiorna il contenuto ed una seconda rete che prende gli ingressi e lo stato e crea l'uscita. Il numero di registri nella state memory definisce il massimo numero di rappresentazioni possibili per la memoria di stato: 2 alla numero di bit. Da questo deriva il termine macchine a stati finiti.

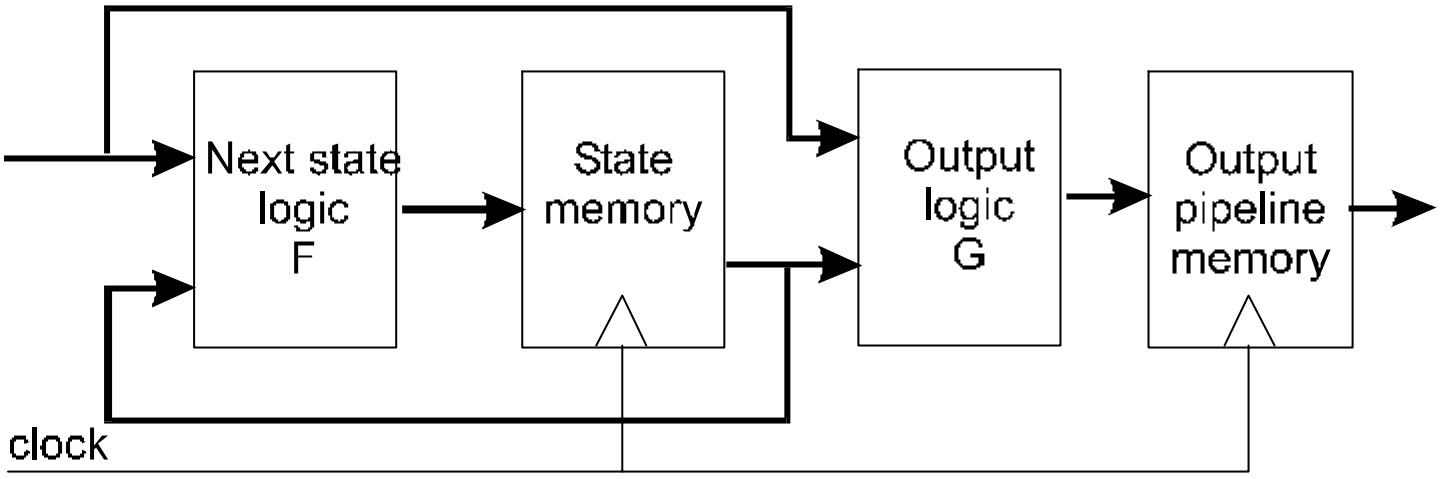
Questo corrisponde al modello di macchina a stati finiti di Mealy:



Che puo` essere usata nella forma semplificata di macchina a stati finiti di Moore che non prevede un collegamento diretto tra ingresso e uscita.



La forma piu` completa e` invece quella che prevede un ulteriore registro in uscita per sincronizzare le transizioni con i fronti di clock.

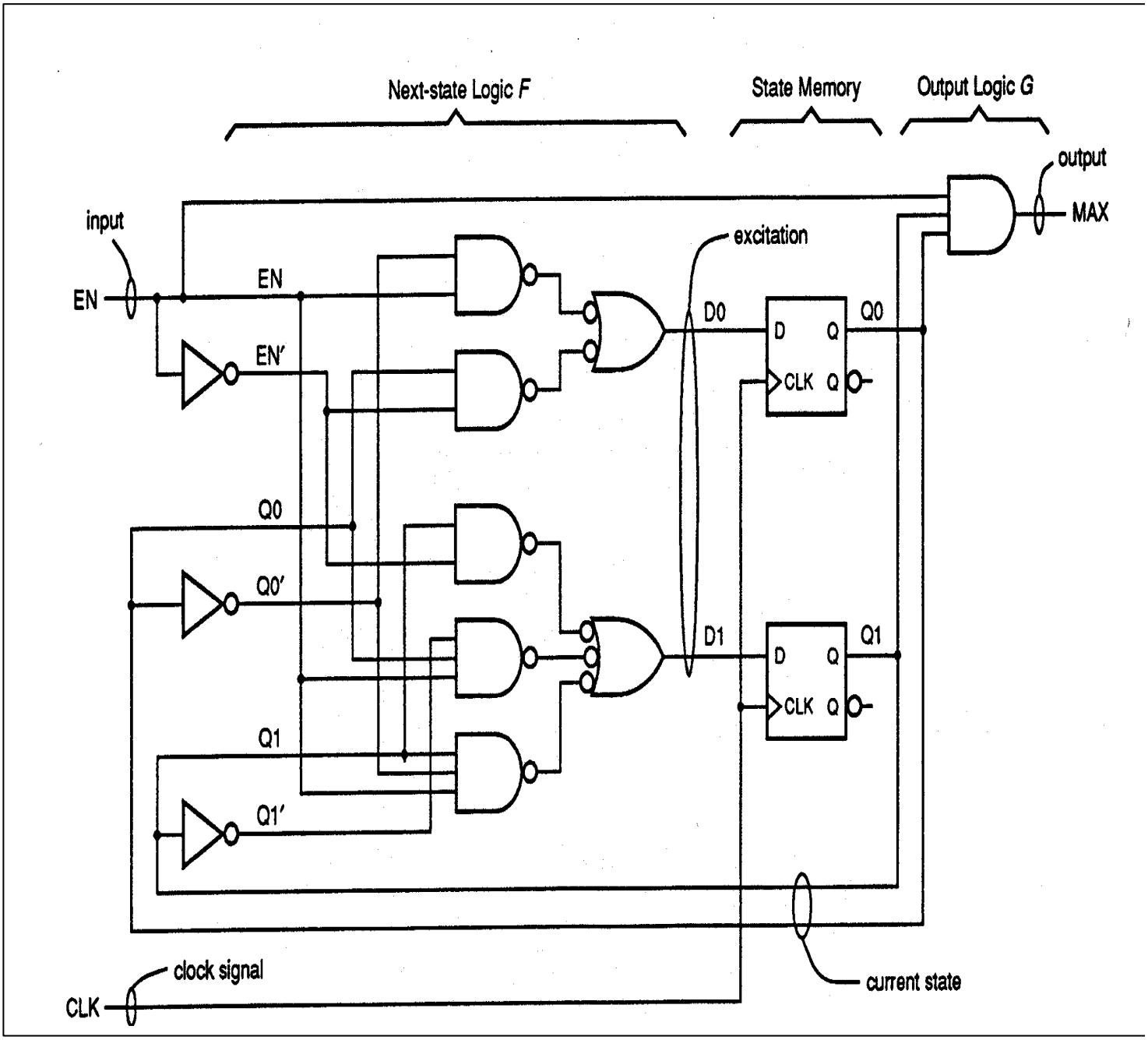


Analisi di una macchina a stati finiti

I passi per analizzare una macchina a stati sono:

1. determinare le funzioni F e G (next state e output logic)
2. utilizzare F e G per scrivere la tabella di transizione, dello stato successivo e quella di uscita.
3. (opzionale) tracciare un diagramma degli stati.

Consideriamo il circuito di esempio



Si determina prima la funzionalita` di F, rilevando le equazioni che governano D0 e D1.

$$D0=Q0\ EN' + Q0'\ EN$$

$$D1=Q1\ EN' + Q1'\ Q0\ EN + Q1\ Q0'\ EN$$

D1 e D0 rappresenta quindi lo stato che verra` scritto dentro la memoria di stato al prossimo fronte di clock positivo (vedi funzionamento del flip-flop D, se non fossero flip-flop di tipo D questo non sarebbe vero), quindi corrispondono a Q1* e Q0*.

$$Q0^* = Q0\ EN' + Q0'\ EN$$

$$Q1^* = Q1\ EN' + Q1'\ Q0\ EN\ Q1\ Q0'\ EN$$

Se si analizza invece la rete di uscita G, si ha che

$$MAX=Q1\ Q0\ EN$$

Tabelle di transizione, stato e uscita

In base alle equazioni trovate si traccia:

1. la tabella delle transizioni che mette in relazione lo stato attuale e l'ingresso con lo stato successivo;
2. la tabella dello stato successivo (assegnando un nome arbitrario ad ogni rappresentazione dello stato nella memoria;
3. la tabella dello stato successivo e delle uscite dalla quale si ricava il diagramma degli stati.

	EN	
Q1Q0	**0**	**1**
00	**00**	**01**
01	**01**	**10**
10	**10**	**11**
11	**11**	**00**

Q1*Q0*

Transizioni

	EN	
S	**0**	**1**
A	**A**	**B**
B	**B**	**C**
C	**C**	**D**
D	**D**	**A**

S*

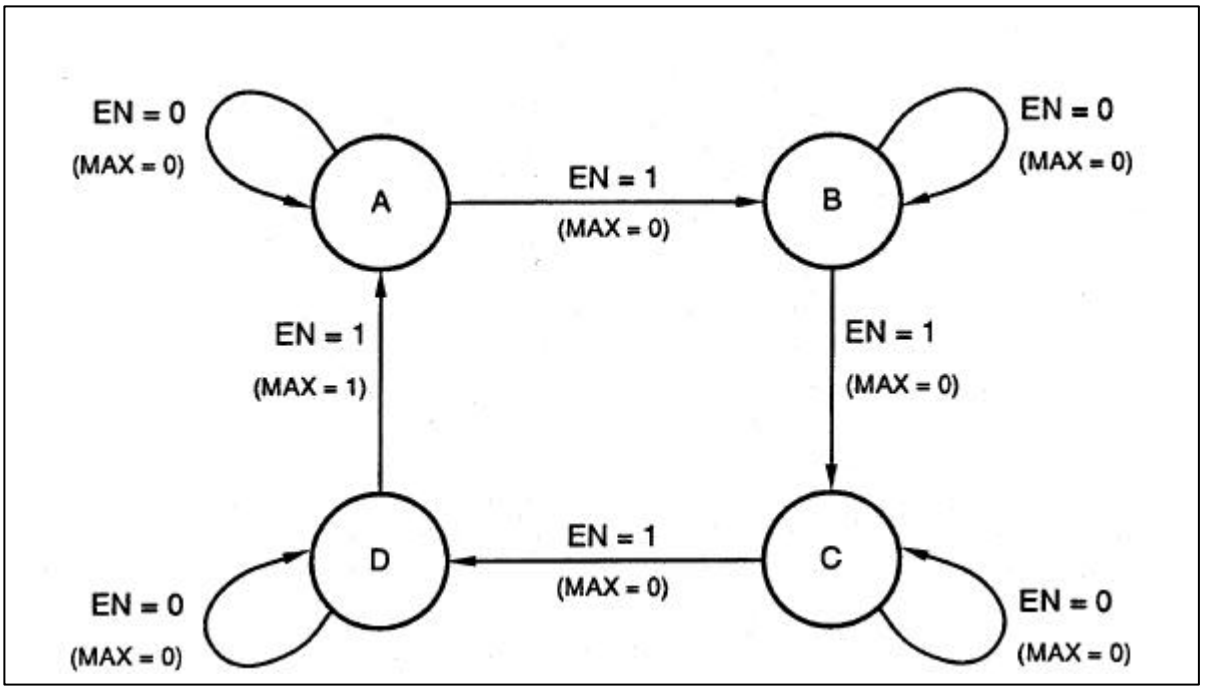
Stato Successivo

	EN	
S	**0**	**1**
A	**A,0**	**B,0**
B	**B,0**	**C,0**
C	**C,0**	**D,0**
D	**D,0**	**A,1**

S*,MAX

Stato Successivo e Uscita

Quando si traccia un diagramma degli stati e` necessario che per ogni stato per ogni combinazione degli ingressi ci sia uno e uno solo stato in cui si finisce.



Esempio di Moore

Se riconsideriamo il circuito e tagliamo il collegamento tra l'ingresso EN e la rete di uscita

MAX = Q1 Q0

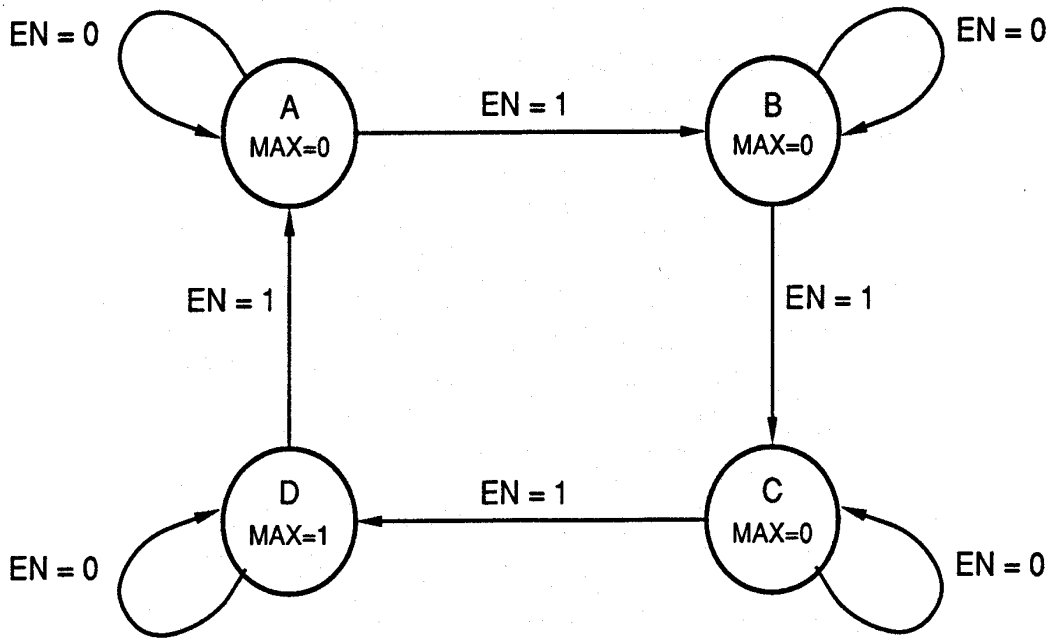
La macchina a stati diventa una macchina di Moore e l'uscita dipende solo dallo stato attuale (non dall'ingresso) quindi si puo` aggiungere una colonna specifica per le uscite. Per far cambiare l'uscita deve cambiare lo stato.

	EN		MAXS
S	**0**	**1**	
A	**A**	**B**	**0**
B	**B**	**C**	**0**
C	**C**	**D**	**0**
D	**D**	**A**	**1**
	S*		

Stato Successivo e Uscita

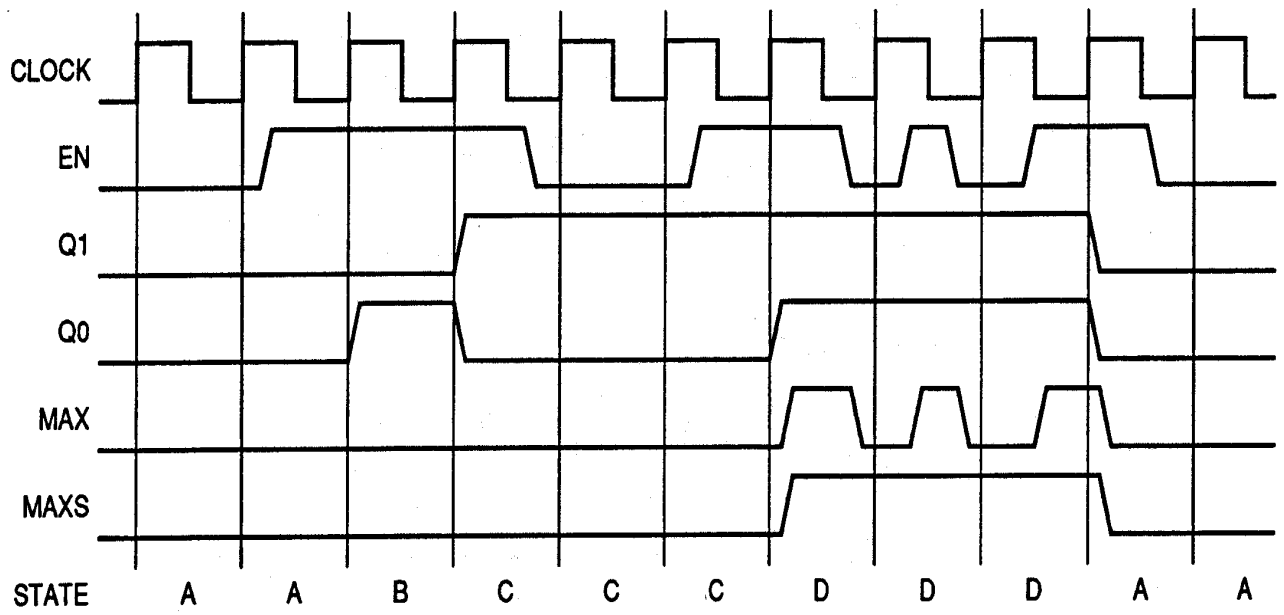
Deve risultare assolutamente evidente che **si e` scelto di tagliare il filo di EN solo per avere un esempio di macchina di Moore, le due macchine sono diverse una dall'altra**.

Se si fosse voluto ottenere la stessa funzionalita` usando una macchina di Moore invece che di Mealy la strategia sarebbe stata completamente diversa**.**



Diagrammi temporali

Un altro modo per analizzare la macchina a stati e` quello di visualizzare le forme d'onda, in cui risulta evidente la differenza tra il comportamento di una macchina di Mealy e di Moore.



Notare in particolare l'andamento dello stato nel tempo

Implementazione strutturale

Il modulo puo` essere implementato sia a livello pseudo-circuitale (al limite con assign al posto di porte logiche) oppure con modelli comportamentali [mcs1.v]

```
`timescale 1ns/100ps
module ffd (ck,d,q,qn);
input ck,d;
output q,qn;
reg q=0; //just for simulation
always @(posedge ck)
 q=d;
assign qn=~q;
endmodule

module macstatiff(ck,en,max,maxs);
 input ck,en;
 output max,maxs;
 wire [1:0] dummy_status;
 wire d0,d1,q0,q1;
 assign d0= q0&~en|~q0&en;
 assign d1= q1&~en|~q1&q0&en|q1&~q0&en;
 ffd ffd1(ck,d0,q0,dummy1);
 ffd ffd2(ck,d1,q1,dummy2);
 assign max=q1&q0&en;
 assign maxs=q1&q0;
 assign dummy_status={q1,q0};//just for out waves
endmodule
```

Avendo da simulare una macchina senza inizializzazione (reset, vedremo meglio piu` avanti), per simulare ho messo q=0 nel flip-flop D. Ovviamente questo non e` un metodo corretto visto che non ha un equivalente fisico/circuitale/realizzativo.

Dummy_status c'e` solo per esigenze di visualizzazione (limiti del CAD se vogliamo) ma non implica una variazione del circuito.

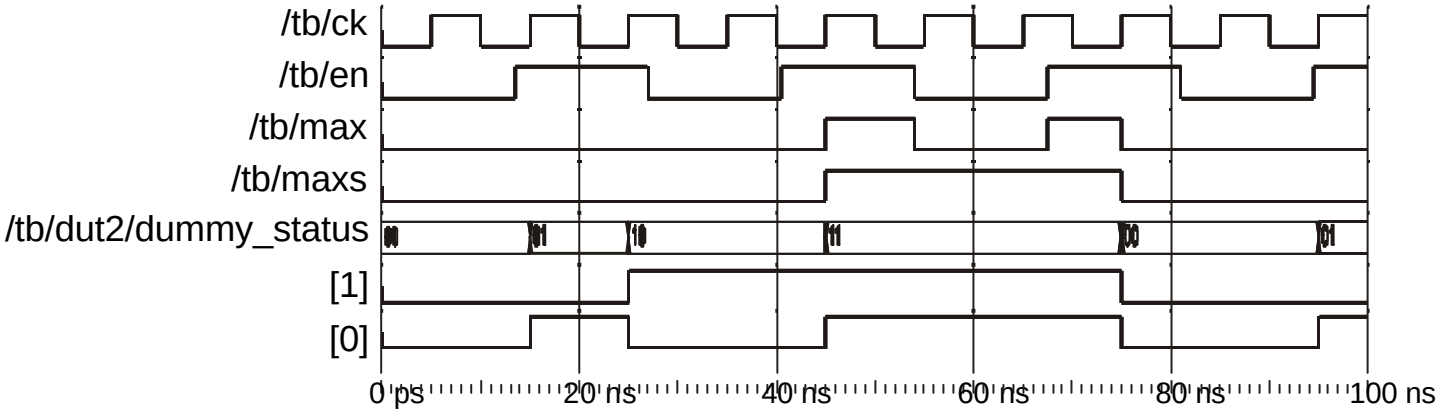
Testbench

Il testbench di una rete sequenziale e` completamente diverso rispetto a quello di una rete combinatoria. Non e` assolutamente sufficiente o significativo passare semplicemente ogni combinazione degli ingressi, bisogna dare una serie di pattern in ingresso che valutino tutti i modi di funzionamento del circuito. Idealmente bisognerebbe fare in modo da andare in tutti gli stati e per ognuno di essi verificare la transizione per qualsiasi combinazione di ingressi.

Come primo esempio consideriamo un testbench molto semplice:

```
module tb;
reg ck=0;
reg en=0;
macstatiff dut2(ck,en, max, maxs);
always
 #5 ck=~ck;
always
 #13.5 en=~en;
endmodule
```

Con questo testbench esiste qualche problema in simulazione?[mcs1.ps]



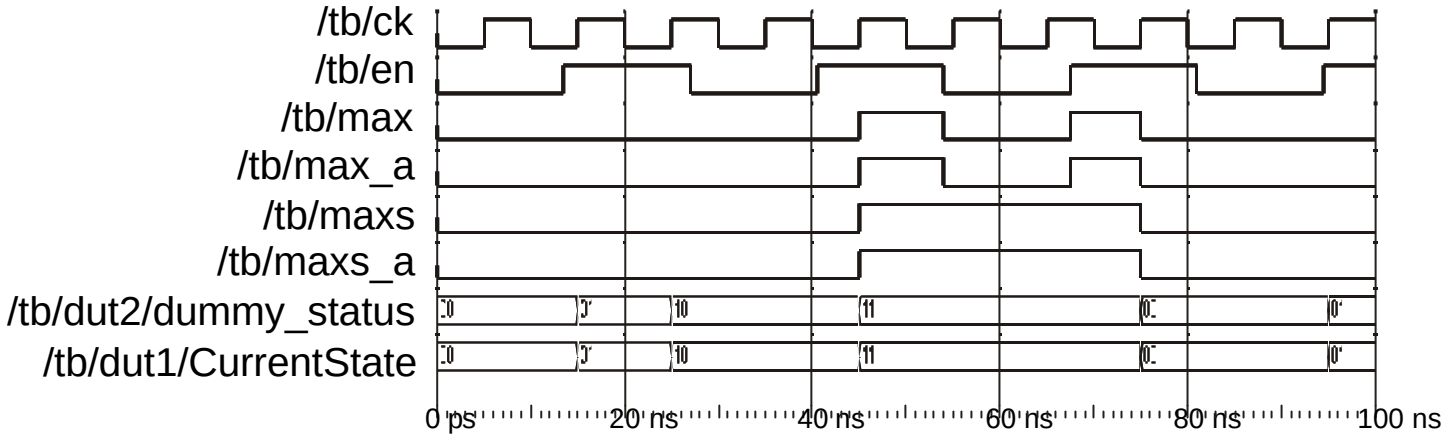
Esercizio: scivere un testbench che riprenda esattamente le forme d'onda del diagramma temporale d'esempio e verificare l'andamento dello stato.

Implementazione algoritmica Verilog

Dalla tabella dello stato successivo ricavata si puo` scrivere la descrizione algoritmica della macchina a stati. [mcs2.v]

```
module macstati (ck,en,max,maxs);
input ck,en;
output max,maxs;
reg max;
reg [1:0] NextState, CurrentState=0;//for simul.
parameter [1:0] A=0,B=1,C=2,D=3;
always @(en or CurrentState)
  case(CurrentState)
   A: if(en==1) NextState=B;
       else NextState=A;
   B: if(en==1) NextState=C;
       else NextState=B;
   C: if(en==1) NextState=D;
       else NextState=C;
   D: if(en==1) NextState=A;
       else NextState=D;
   default: NextState=A;
endcase

always @(posedge ck)
CurrentState=NextState;

always @(CurrentState or en)
  case(CurrentState)
   A: max=0;
   B: max=0;
   C: max=0;
   D: if(en==0) max=0; else max=1; // Mealy
  endcase
assign maxs=CurrentState==D? 1:0; // Moore
endmodule
```

Confrontare le uscite delle due versioni (strutturale ed algoritmica) per verificare che le due versioni corrispondano [mcs2.ps].

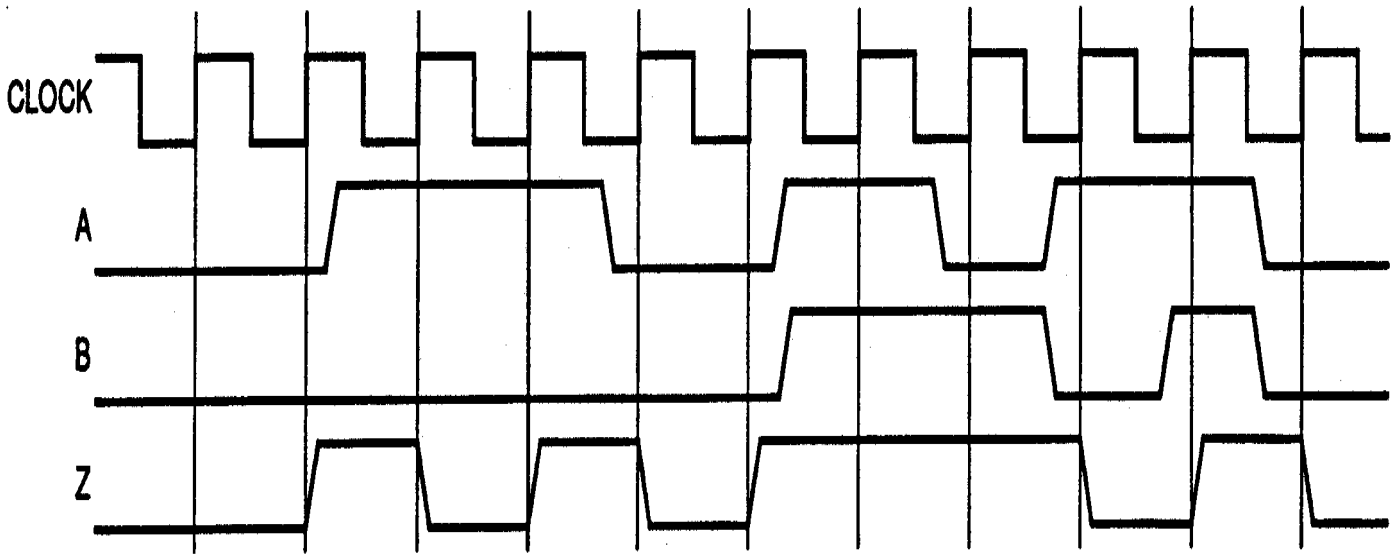


Sintesi di una macchina a stati.

Il percorso di progettazione e` esattamente l'inverso di quello di analisi, si parte dalla descrizione funzionale, si passa alla tabella dello stato successivo o al diagramma degli stati fino ad arrivare alla tabella delle transizioni e al circuito.

Iniziamo con un esempio.

Si vuole progettare una macchina a stati sincrona con due ingressi A e B ed una uscita Z che valga 1 se A ha avuto lo stesso valore in corrispondenza degli ultimi due fronti di clock, o B e` rimasta ad 1 da quando Z valeva 1 (cioe` un B ad 1 non permette a Z di tornare a 0)



In questo caso di preferisce iniziare a trattare direttamente la tabella dello stato successivo.

Meaning	S	AB 00	AB 01	AB 11	AB 10	Z
Initial state	INIT					0
		S*				

Meaning	S	AB 00	AB 01	AB 11	AB 10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0					0
Got a 1 on A	A1					0
		S*				

Meaning	S	AB 00	AB 01	AB 11	AB 10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK	OK	A1	A1	0
Got a 1 on A	A1	A0	A0	OK	OK	0
Got two equal A input	OK					1
		S*				

Meaning	S	AB 00	AB 01	AB 11	AB 10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK	OK	A1	A1	0
Got a 1 on A	A1	A0	A0	OK	OK	0
Got two equal A input	OK	?	OK	OK	?	1
		S*				

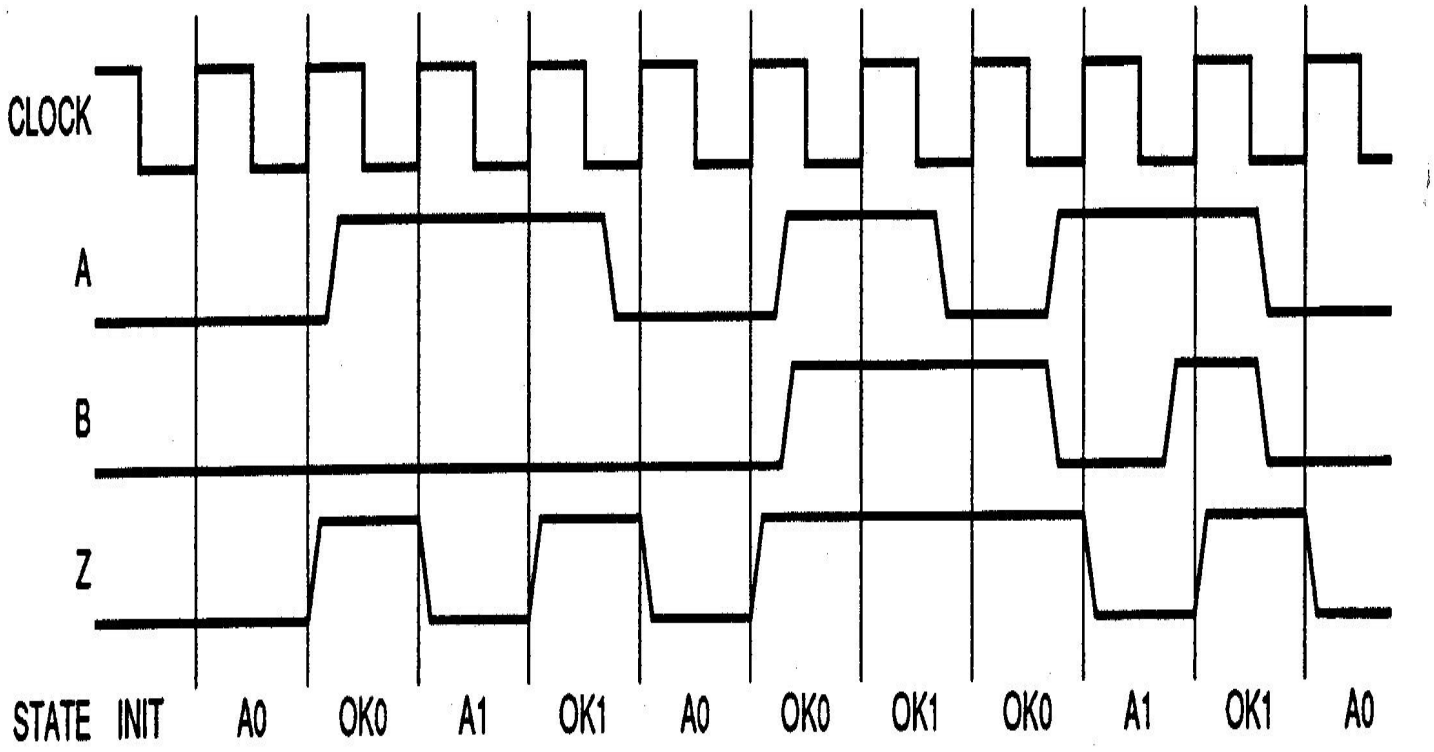
Non e` possibile determinare cosa mettere come stato successivo ad OK se riceviamo uno 00 o un 10, perche` non sappiamo come si e` arrivati allo stato OK. Non resta che sdoppiare lo stato OK.

Meaning	S	AB 00	AB 01	AB 11	AB 10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK0	OK0	A1	A1	0
Got a 1 on A	A1	A0	A0	OK1	OK1	0
Got two equal, A=0 last	OK0					1
Got two equal, A=1 last	OK1					1
		S*				

Meaning	S	AB 00	AB 01	AB 11	AB 10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK0	OK0	A1	A1	0
Got a 1 on A	A1	A0	A0	OK1	OK1	0
Got two equal, A=0 last	OK0	OK0	OK0	OK1	A1	1
Got two equal, A=1 last	OK1	A0	OK0	OK1	OK1	1
		S*				

Diagramma temporale

Dal grafico temporale che segue si puo` vedere l'evolversi dello stato al variare degli ingressi.



Esercizio: descrivere in Verilog algoritmico la macchina a stati e ricavarne le forme d'onda, ottenendo anche l'indicazione dello stato in cui il circuito si trova (per il momento assegnare una codifica qualunque allo stato).

Stati equivalenti

Con un minore spirito critico si poteva arrivare a tabelle dello stato successivo piu` complesse come la seguente:

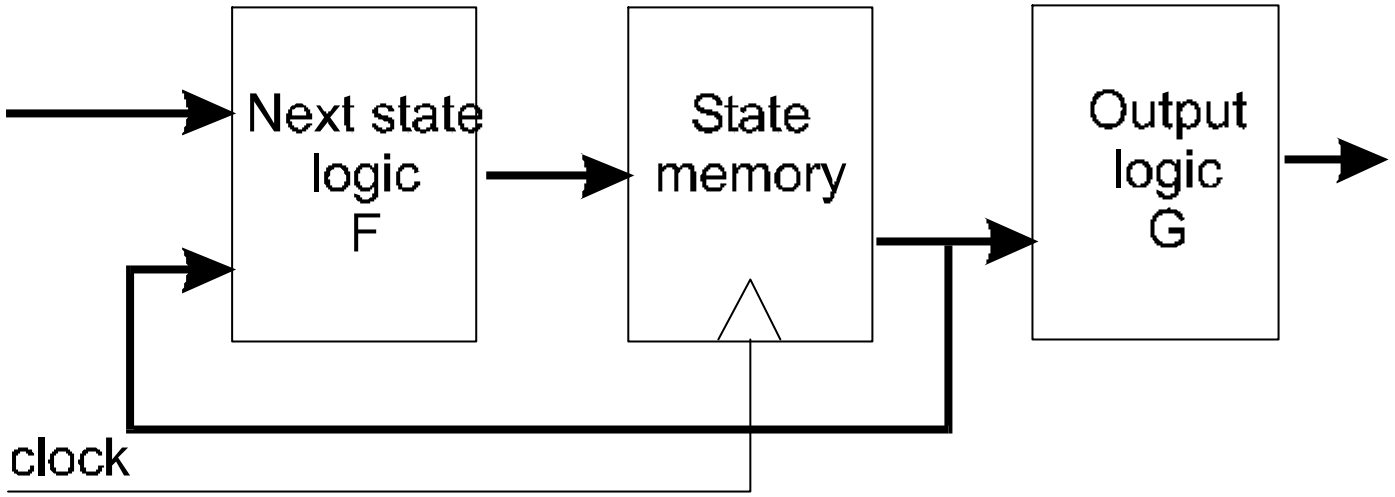
Meaning	S	AB 00	AB 01	AB 11	AB 10	Z
Initial state	**INIT**	**A0**	**A0**	**A1**	**A1**	**0**
Got a 0 on A	**A0**	**OK00**	**OK00**	**A1**	**A1**	**0**
Got a 1 on A	**A1**	**A0**	**A0**	**OK11**	**OK11**	**0**
Got 00 on A	**OK00**	**OK00**	**OK00**	**OKA1**	**A1**	**1**
Got 11 on A	**OK11**	**A0**	**OKA0**	**OK11**	**OK11**	**1**
OK, got a 0 on A	**OKA0**	**OK00**	**OK00**	**OKA1**	**A1**	**1**
OK, got a 1 on A	**OKA1**	**A0**	**OKA0**	**OK11**	**OK11**	**1**
		S*				

Ci si accorge pero` che gli stati OK00 e OKA0 (e OK11 e OKA1) sono equivalenti perche`:

1. hanno uscite identiche
2. per ogni combinazione di ingressi hanno stato successivo identico o equivalente (si deve supporre l'equivalenza e verificarla)

Quindi la tabella viene semplificata rimuovendo gli stati OK00 e OK11 e si ottiene la precedente.

Assegnamento degli stati



Per costruire il circuito devo assegnare ad ogni stato una combinazione di bit della memoria di stato. Visto che di solito i flip-flop D sono dotati di un ingresso di reset (o clear) per azzerare il loro contenuto, e` facile portare la macchina nello stato 0 anche non sfruttando la rete combinatoria di eccitazione. Per questo quando e` possibile e` meglio assegnare la configurazione 0 allo stato iniziale o di reset. Per il resto esistono diverse tecniche applicabili, tutte sono orientate a diminuire le risorse necessarie, in termini di flip-flop e delle reti combinatorie di eccitazione e uscita.

Alcune sono le seguenti:

1. Simplest. Gli stati vengono numerati secondo la codifica binaria dei primi numeri. Numero minimo di bit.
2. Decomposed. Si usa il numero minimo di bit ma si cerca di dare ad ogni bit un ruolo/significato.
3. One-hot. Si usa un bit per ogni stato. Semplifica le logiche di eccitazione (spesso) ma aumenta il numero di flip-flop.
4. Almost One-hot. Si usa la One-hot ma lasciando allo stato di init/reset lo stato 0. Si usa un flip-flop di meno e agevola il reset.

State Name	Assignment			
	Simplest	Decomposed	One-hot	Almost One-hot
INIT	000	000	00001	0000
A0	001	100	00010	0001
A1	010	101	00100	0010
OK0	011	110	01000	0100
OK1	100	111	10000	1000

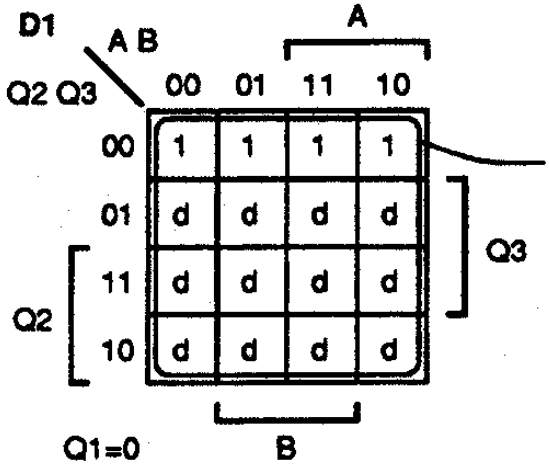
Supponendo di scegliere la Decomposed si ottiene la seguente tabella di transizione:

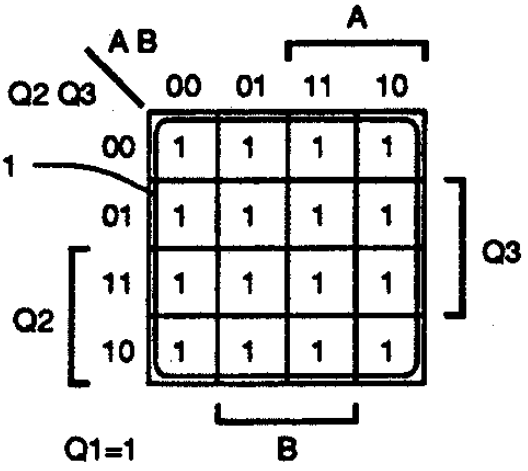
Name	Q1Q2Q3	AB				Z
		00	01	11	10	
INIT	000	100	100	101	101	0
A0	100	110	110	101	101	0
A1	101	100	100	111	111	0
OK0	110	110	110	111	101	1
OK1	111	100	110	111	111	1
		Q1*Q2*Q3*				

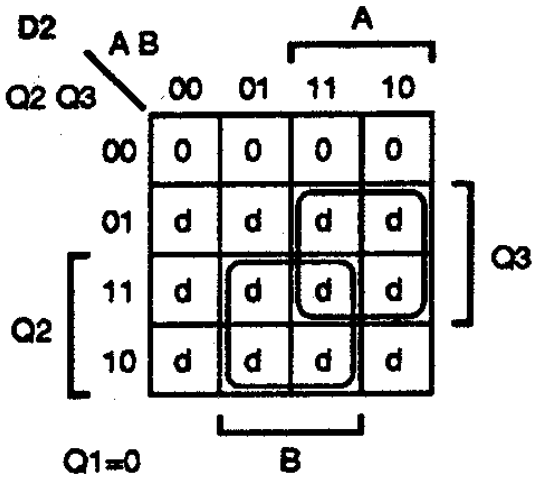
Se si considera di utilizzare flip-flop di tipo D, il segnale D1=Q1*, il D2=Q2* e D3=Q3*.

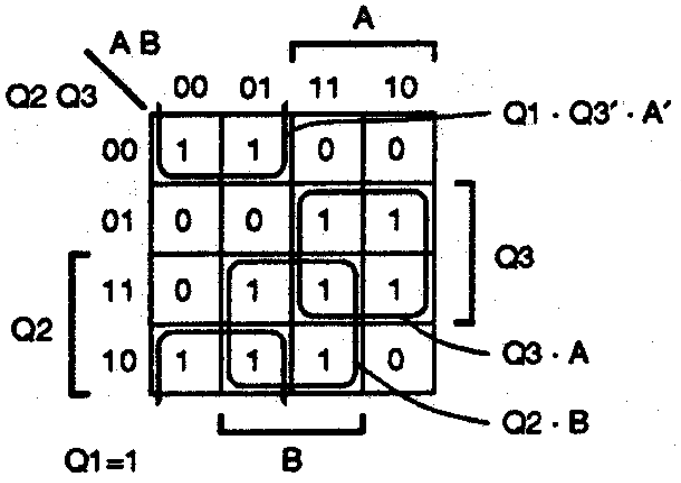
Ma gli altri stati esistono, cosa posso fare? Perche` dovrebbero essere eccitati? E se si rompe il circuito?

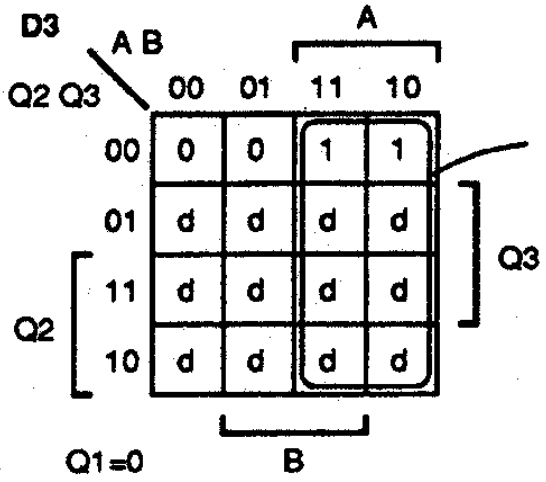
Considero don't care

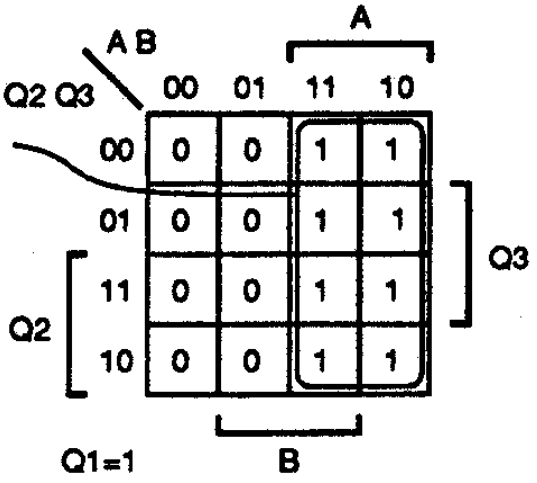












Da quest'ultima si ottiene il circuito:

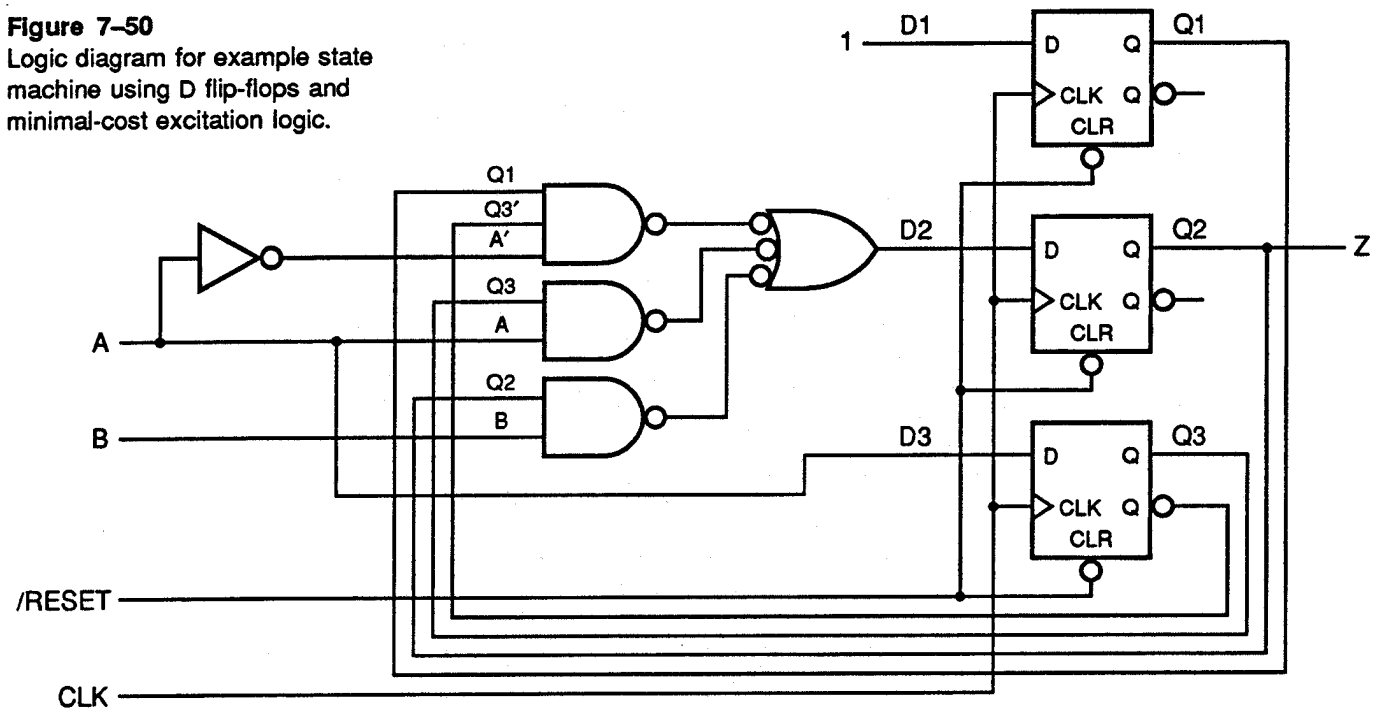


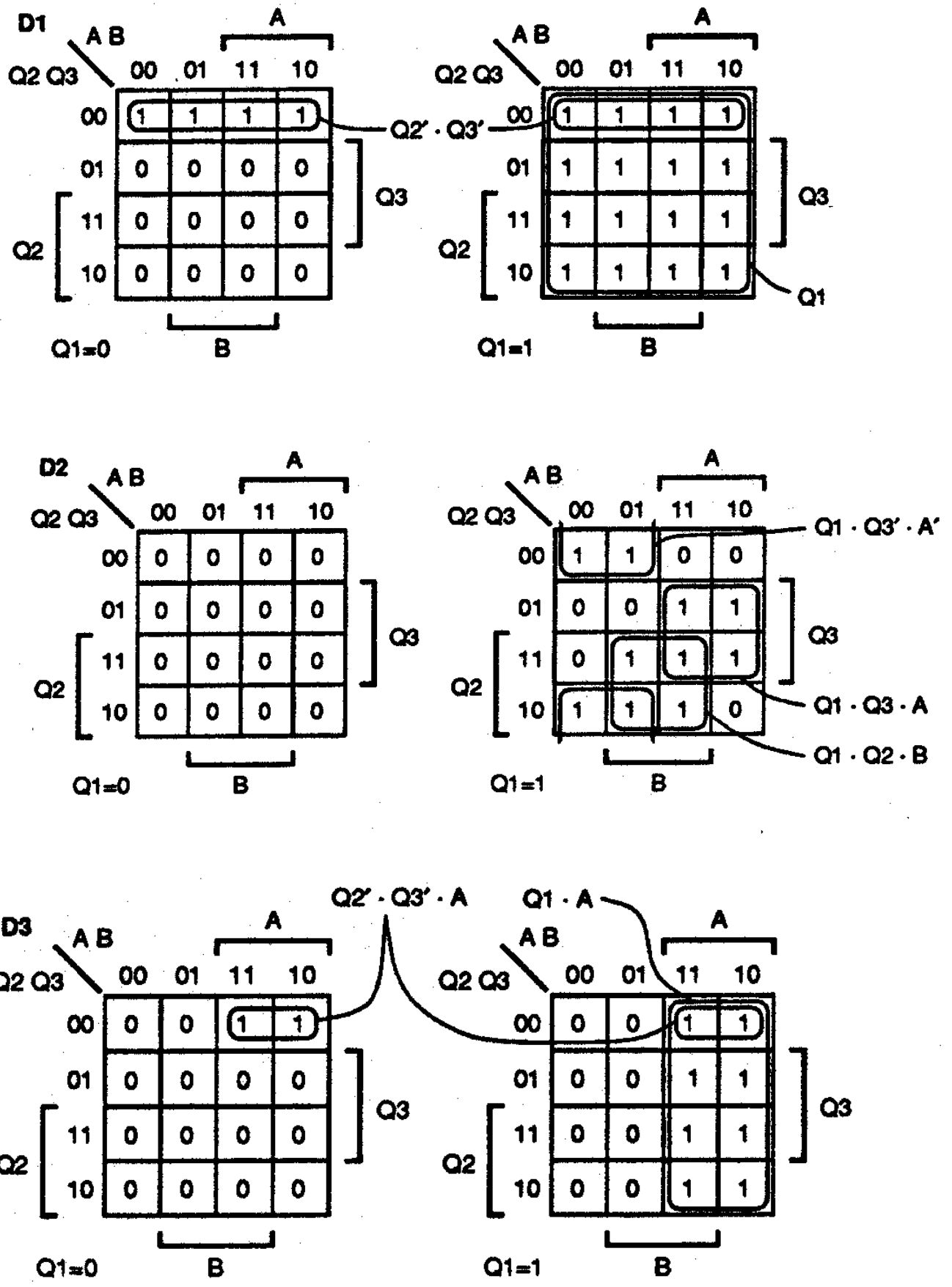
Figure 7–50
Logic diagram for example state machine using D flip-flops and minimal-cost excitation logic.

A seconda di come raggruppo i termini nelle mappe di karnaugh quando vado a realizzare il circuiti sostituisco di fatto al don't care o un 1 o uno 0. Si perde traccia del fatto che erano don't care sono dei stati ben definiti, che non dovrebbero essere di norma mai raggiunti.

Esercizio: compilare la mappa delle transizioni completa per il circuito realizzato.

Esercizio: procedere alla progettazione della stessa macchina sfruttando flip-flop di altro tipo.

Considero tutti 0 cioe` di INIT



Inizializzazione di un circuito sequenziale

Visto che ogni circuito reale ha un meccanismo di inizializzazione va prevista una opportuna tecnica di reset. Tale tecnica puo` essere correttamente implementata prevedendo un ingresso di reset (tipicamente asincrono) sui flip-flop D e sulla macchina a stati al livello superiore. Una possibile implementazione del flip-flop D e` quindi:

```
module dff(ck, reset, d, q);
 input ck, reset, d;
 output q;
 reg q;

always @(posedge ck or posedge reset)
begin
 if(reset)
  q<=0;
 else
  q<=d;
end
endmodule
```

Non e` invece consigliabile usare initial (espliciti o impliciti nella definizione) per inizializzare q perche` privi di senso fisico/implementativo/circuitale.

Esercizio: scrivere l'implementazione algoritmica Verilog e quella circuitale con reset e confrontare i risultati su un testbench

Implementazione algoritmica

Una possible implementazione algoritmica e` la seguente [mcsAB.v]:

```
`timescale 1ns/1ns
module macchstati(z,ck,a,b,reset_l);
output z;
input ck,a,b,reset_l;
reg [2:0] CS,NS;
parameter[2:0]  Init=3'b000,  A0=3'b100,  A1=3'b101,
OK0=3'b110, OK1=3'b111;
reg z;
always @(CS or a or b)
   case (CS)
      Init:if(a==0) NS=A0;
                 else NS=A1;
      A0:if(a==0) NS=OK0;
                 else NS=A1;
      A1:if(a==0) NS=A0;
                 else NS=OK1;
      OK0:case({a,b})
             2'b00:NS=OK0;
             2'b01:NS=OK0;
             2'b10:NS=A1;
             2'b11:NS=OK1;
             default:NS=Init;
          endcase
      OK1:case({a,b})
             2'b00:NS=A0;
             2'b01:NS=OK0;
             2'b10:NS=OK1;
             2'b11:NS=OK1;
             default:NS=Init;
          endcase
      default:NS=Init;
   endcase
```

```
always @(posedge ck or negedge reset_l)
if(reset_l) CS=NS;
  else CS=Init;

always @(CS)
    case(CS)
        OK0:z=1;
        OK1:z=1;
        default:z=0;
    endcase
endmodule
```

Implementazione strutturale

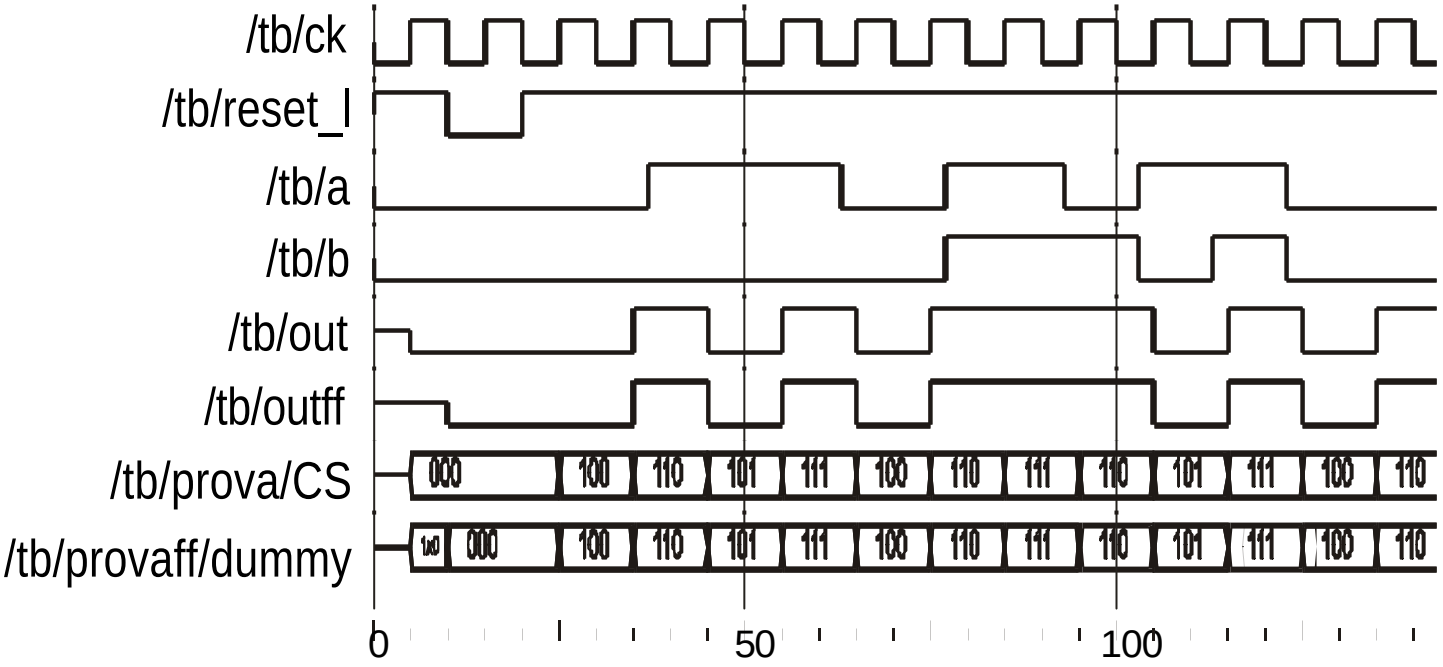
```
module ffd(q,qn,d,ck,reset_l);
input d,ck,reset_l;
output q,qn;
reg q;
always@(posedge ck or negedge reset_l)
 if (reset_l) q=d;
  else q=0;
assign qn=~q;
endmodule

module macchstatiff(z,ck,a,b,reset_l);
output z;
input ck,a,b,reset_l;
supply1 uno;
wire [2:0]dummy;
ffd ffd1(q1,qn1,d1,ck,reset_l);
ffd ffd2(q2,qn2,d2,ck,reset_l);
ffd ffd3(q3,qn3,d3,ck,reset_l);
assign d1=uno;
assign d3=a;
not(na,a);
nand(d2,na1,na2,na3);
nand(na1,q1,qn3,na);
nand(na2,q3,a);
nand(na3,q2,b);
assign z=q2;
assign dummy={q1,q2,q3};
endmodule
```

Testbench

```
module tb;
reg a=0,b=0;
reg ck=0;
reg reset_l=1;
macchstati  prova(out,ck,a,b,reset_l);
macchstatiff provaff(outff,ck,a,b,reset_l);
initial
begin
    #10 reset_l=0;
    #10 reset_l=1;
   #17 a=1;
    #26 a=0;
    #14 a=1;
    #16 a=0;
    #10 a=1;
    #20 a=0;
   #20 $stop;
end
 initial
    begin
        #20;
        #57 b=1;
        #26 b=0;
        #10 b=1;
        #10 b=0;
    end
  always
    #5 ck=~ck;
endmodule
```

Forme d’onda

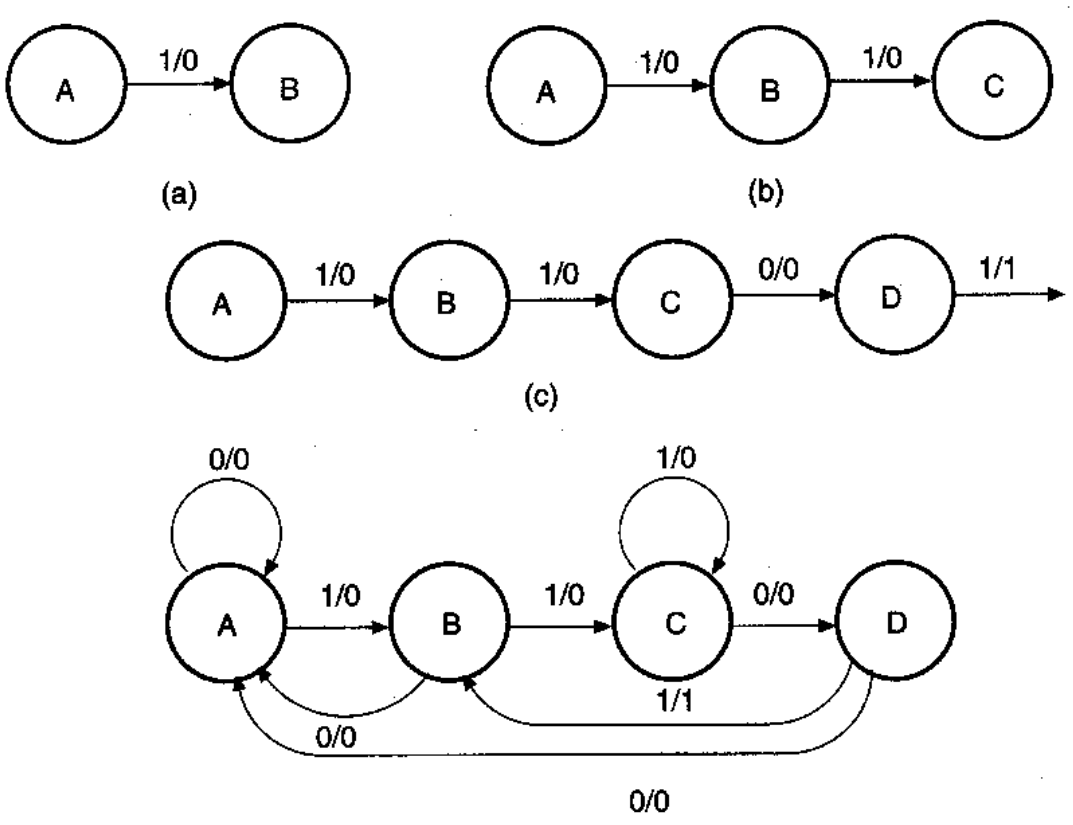
Simulate si ottengono le seguenti curve[mcsAB.ps] (da confrontare con il risultato desiderato nelle specifiche.



Riconoscitore di sequenza 1

Il sistema ha un ingresso IN di un bit (piu` un RESET ovviamente) ed una uscita Z. Il sistema deve riconoscere il verificarsi di una sequenza 1101 sull'ingresso IN. In particolare, riconosciuta la sequenza 110, nello stato successivo Z vale 1 solo quando X vale 1. Si possono riutilizzare i bit per piu` sequenze.

Quest'ultimo aspetto la rende una macchina di Mealy. Proviamo a progettarla direttamente dal diagramma degli stati.



A cui corrisponde la seguente tabella dello stato successivo:

	IN	
S	**0**	**1**
A	**A,0**	**B,0**
B	**A,0**	**C,0**
C	**D,0**	**C,0**
D	**A,0**	**B,1**
	S*,Z	

Sintesi strutturale

Per semplificare la rete di uscita si decide di assegnare a D la configurazione 11, per semplificare il reset 00 a Init. Resta 01 che assegniamo a B e 10 che assegniamo a C.

Ne deriva la seguente tabella delle transizioni.

	IN	
Q1Q0	**0**	**1**
00	**00,0**	**01,0**
01	**00,0**	**10,0**
10	**11,0**	**10,0**
11	**00,0**	**01,1**

Q1*Q0*,Z

Da cui con le mappe di Karnaugh si ottiene

D1=Q1Q0' +Q1'Q0X D0=Q1'Qo'X+Q1Q0X+Q1Q0'X'

OUT=Q1Q0X

Che in Verilog si puo` scrivere come segue [seqmil.v]

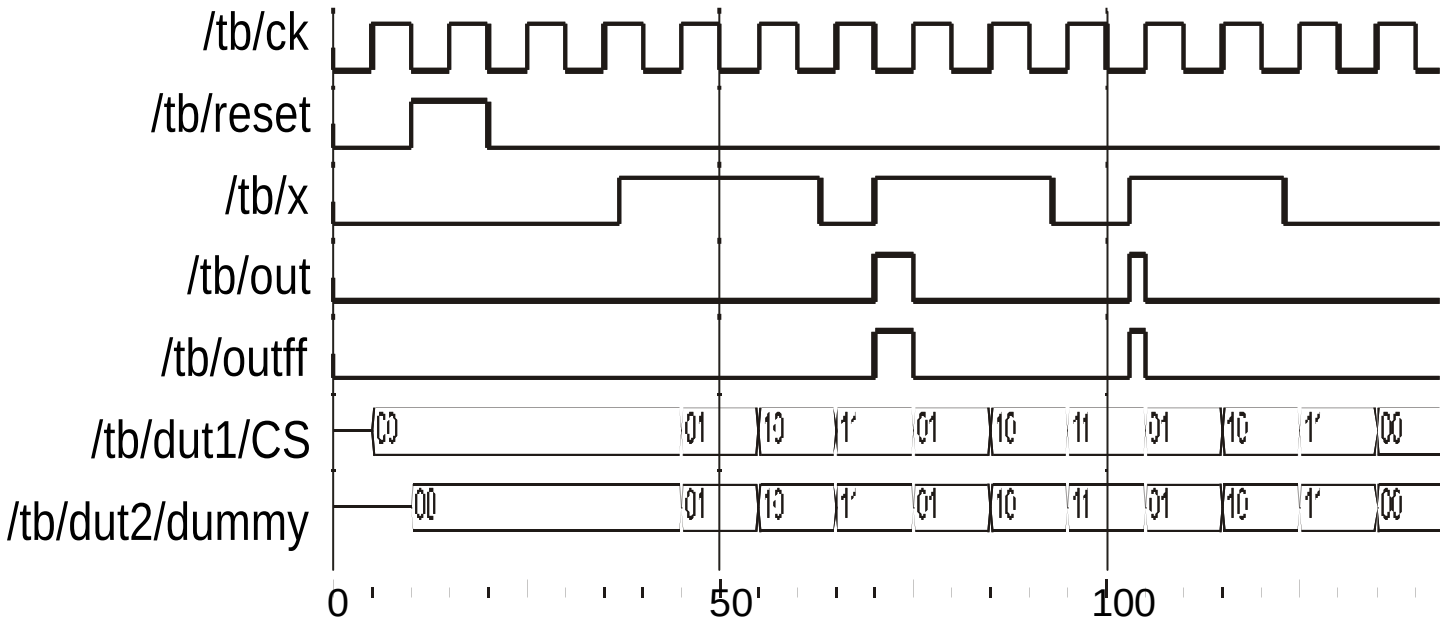
```
`timescale 1ns/1ns
module macchstatiff(z,ck,in,reset);
output z;
input ck,in,reset;
wire [1:0]dummy;
ffd ffd0(q0,qn0,d0,ck,reset);
ffd ffd1(q1,qn1,d1,ck,reset);
not(nin,in);
nand(d1,a1,a2);
nand(a1,q1,qn0);
nand(a2,qn1,q0,in);
nand(d0,a3,a4,a5);
nand(a3,qn1,qn0,in);
nand(a4,q1,q0,in);
nand(a5,q1,qn0,nin);
and(z,q1,q0,in);
assign dummy={q1,q0};
endmodule
```

```
module ffd(q,qn,d,ck,reset);
input d,ck,reset;
output q,qn;
reg q;
always@(posedge ck or posedge reset)
 if (reset) q=0;
  else q=d;
assign qn=~q;
endmodule

module macchstati(z,ck,in,reset);
output z;
input ck,in,reset;
reg [1:0] CS,NS;
parameter[1:0]    A=2'b00,    B=2'b01,    C=2'b10,
D=2'b11;
reg z;
always @(CS or in)
    case (CS)
        A:NS= in?B:A;
        B:NS= in?C:A;
        C:NS= in?C:D;
        D:NS= in?B:A;
        default:NS=A;
    endcase
always @(posedge ck or posedge reset)
if(reset) CS=A;
  else CS=NS;
always @(CS or in)
    case(CS)
        A,B,C:z=0;
        D:z=in;
        default:z=0;
    endcase
endmodule
```

Testbench

```
module tb;
reg in=0;
reg ck=0;
reg reset=0;
macchstati  dut1(out,ck,in,reset);
macchstatiff dut2(outff,ck,in,reset);
initial
begin
    #10 reset=1;
   #10 reset=0;
   #17 in=1;
    #26 in=0;
    #7 in=1;
    #23 in=0;
    #10 in=1;
    #20 in=0;
   #20 $stop;
end
always
    #5 ck=~ck;
endmodule
```

Simulato si ha[seqmil.ps]



Riconoscitore di sequenza 2

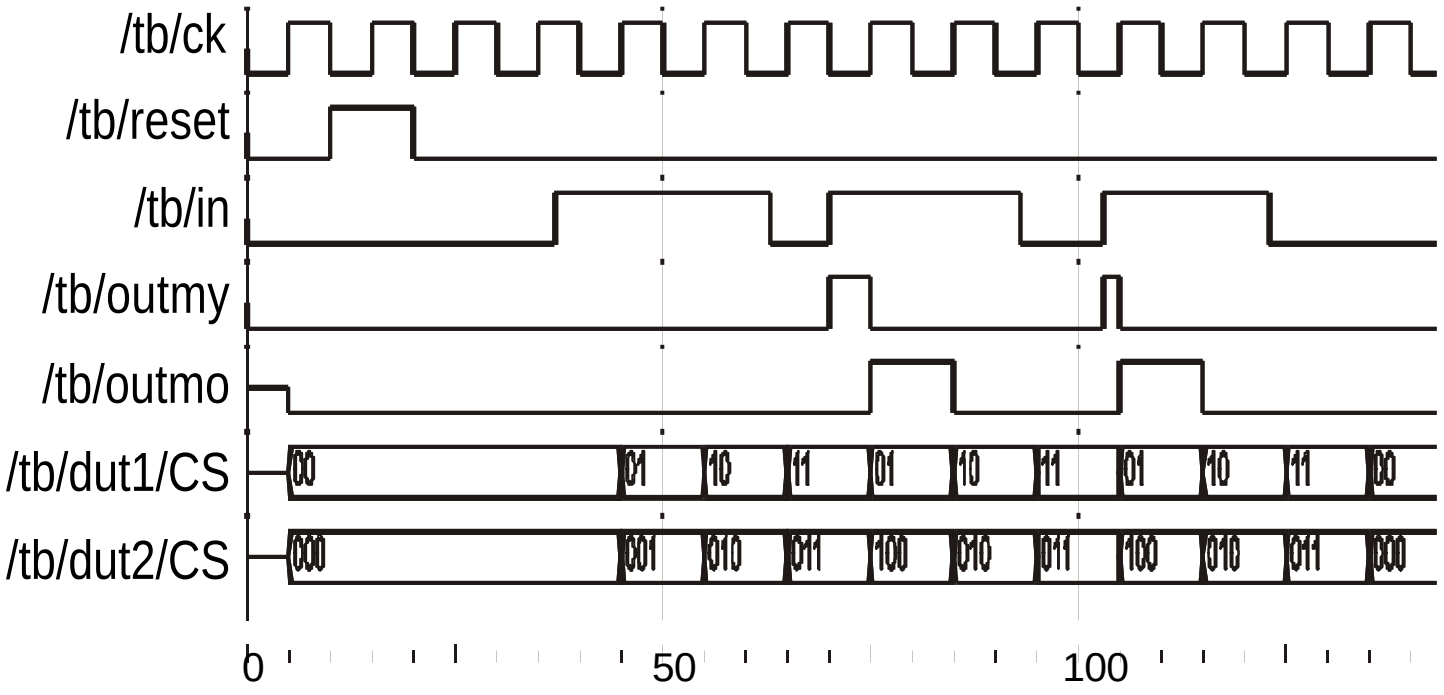
Il sistema ha un ingresso IN di un bit (piu` un RESET ovviamente) ed una uscita Z. Il sistema deve riconoscere il verificarsi di una sequenza 1101 sull'ingresso IN. In particolare, riconosciuta la sequenza nello stato successivo Z vale 1. Si possono riutilizzare i bit per piu` sequenze.

	IN		Z
S	0	1	
INIT	INIT	GOT1	0
GOT1	INIT	GOT11	0
GOT11	GOT110	GOT11	0
GOT110	INIT	OK	0
OK	INIT	GOT11	1
	S*		

Fare la sintesi a livello di porte di questo circuito.

Confrontare le due versioni algoritmiche [seqmymo.v] di questo e del precedente con un opportuno testbench che ne evidenzi le differenze.

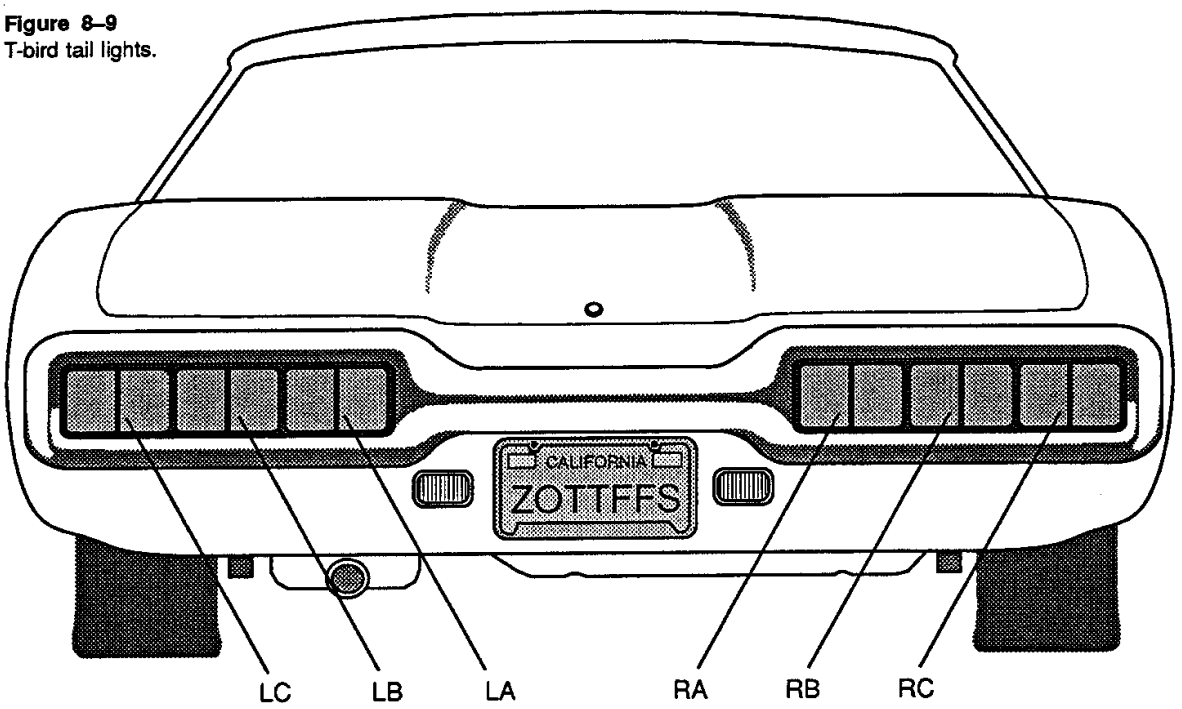
[seqmymo.ps]



Controllo frecce Ford Thunderbird 1965

Si vuole controllare la sequenza di accensione delle frecce. Si hanno tre comandi LEFT, RIGHT e HAZ (emergenza). I tre comandi sono separati.

Figure 8–9
T-bird tail lights.



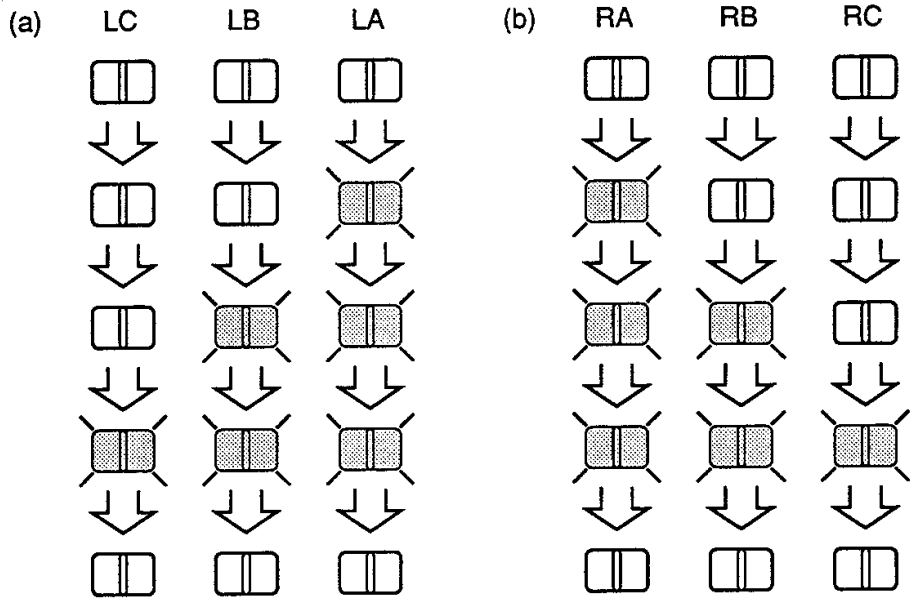
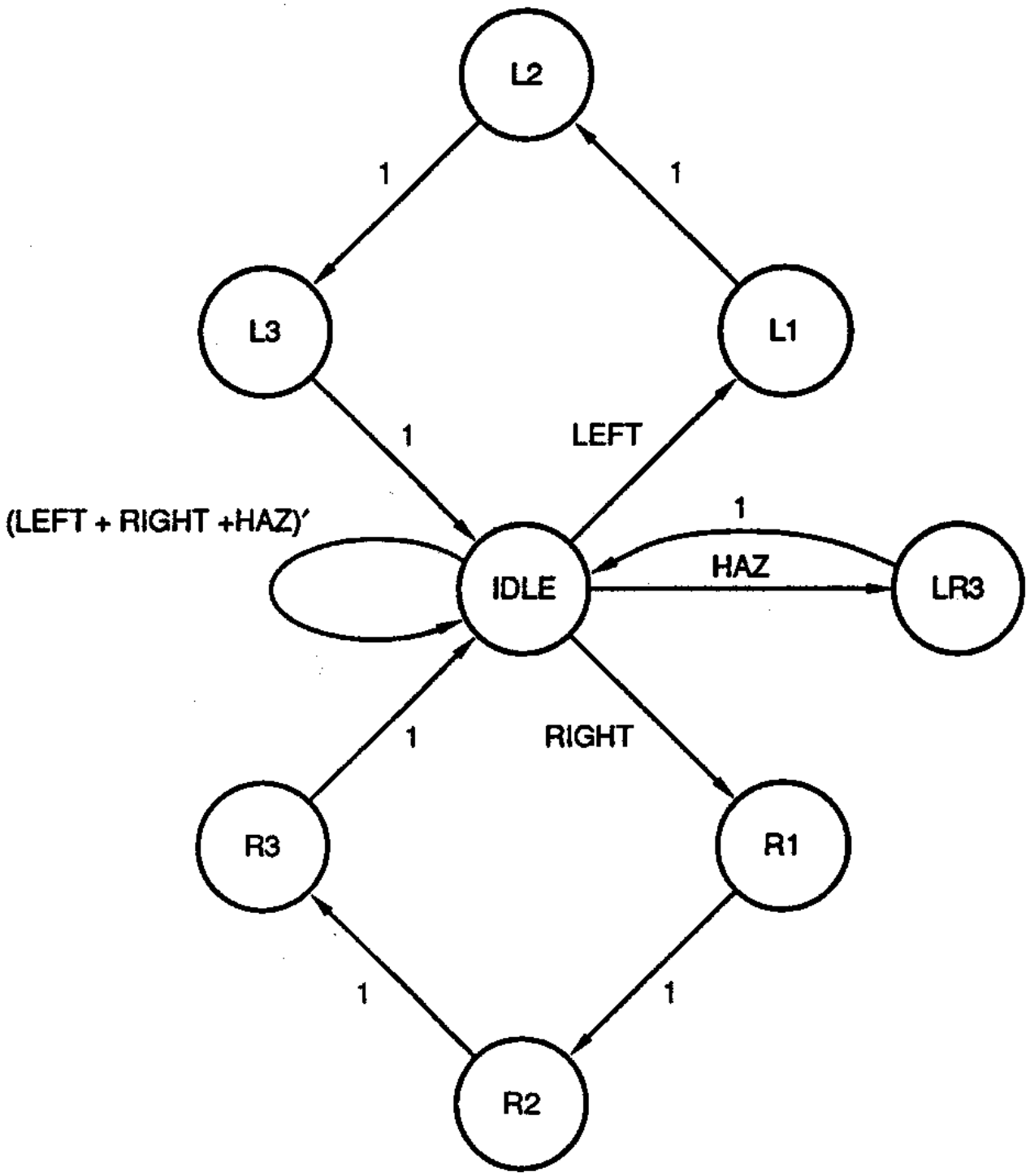


Figure 8–10 Flashing sequence for T-bird tail lights: (a) left turn; (b) right turn.

Si disegna un primo diagramma degli stati.



Perche` e` sbagliato?
Correggerlo e procedere alla progettazione in Verilog.