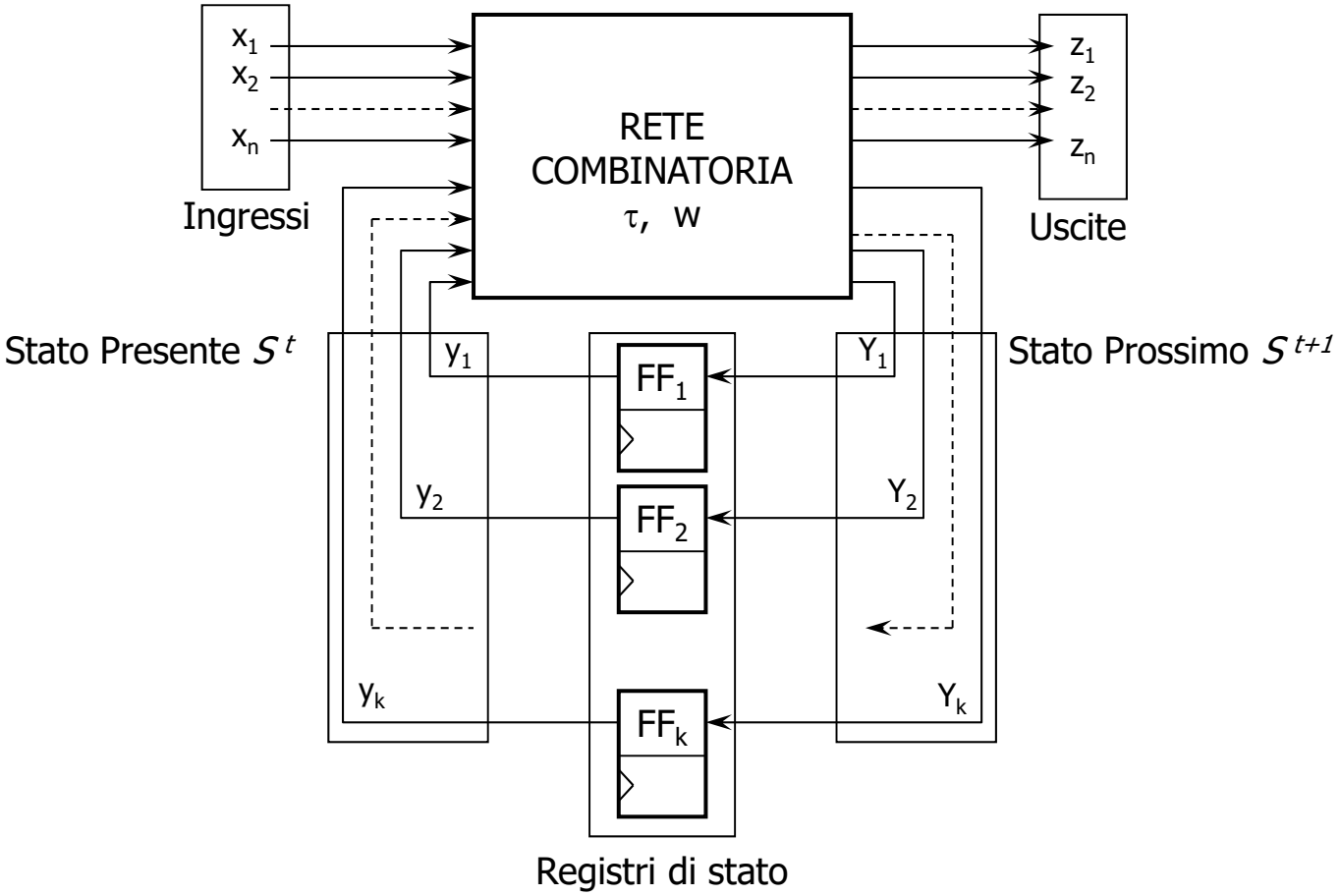
Macchina Sequenziale: architettura generale

- La struttura generale di una macchina sequenziale è la seguente:



Macchina sequenziale

- Il valore delle uscite all'istante *t* dipende dalla successione degli ingressi che precedono l'istante *t*
- Ciò implica il concetto di *stato*
- Una macchina sequenziale è definita dalla quintupla (I, U, S, τ, ω) :
  - *I* - Alfabeto di Ingresso
    - E' costituito dall'insieme *finito* dei *simboli* di ingresso
  - *U* - Alfabeto d'Uscita
    - E' costituito dall'insieme *finito* dei *simboli* d'uscita
  - *S* - Insieme degli Stati
    - Insieme *finito* e *non vuoto* degli *stati*
  - τ - Funzione stato prossimo
  - *w* - Funzione d'uscita

Macchina Sequenziale: architettura generale

- Il problema della sintesi comportamentale di una rete sequenziale consiste nella:
  - Identificazione delle funzioni τ e ω
  - Sintesi della rete combinatoria che le realizza
- Gli elementi di memoria sono costituiti da ***bistabili***
- La funzione di stato prossimo dipende dal tipo di bistabili utilizzati
- La funzione di uscita è indipendente dal tipo di bistabili utilizzati

Il concetto di stato

- Le uscite di un circuito sequenziale dipendono da tutta la storia degli ingressi
- Questo aspetto viene formalizzato grazie al concetto di ***stato***
- Lo *stato* di un circuito sequenziale:
  - E' un insieme di *variabili di stato*
  - Contiene tutta l'informazione necessaria a descrivere il *comportamento passato* del circuito
  - Contiene tutta l'informazione necessaria a definire il *comportamento futuro* del circuito

Il concetto di stato

- Il concetto di stato è legato al concetto di tempo discreto
- Lo stato di un circuito deve essere aggiornato ad ogni istante del tempo discreto, ovvero ad ogni ciclo di clock
- Lo stato di un circuito ad un dato istante t_k dipende:
  - dagli ingressi all'istante t_k
  - dallo stato precedente, ovvero dallo stato al tempo t_{k-1}
- Lo stato di un circuito deve essere pertanto memorizzato
- A tale scopo si utilizzano degli elementi di memoria detti *bistabili*

Macchina sequenziale

- Funzione stato prossimo τ
  - Ad ogni stato presente e per ogni simbolo di ingresso la funzione τ associa uno stato futuro:

    $$\tau : S \times I \rightarrow S$$

  - Ad ogni coppia *{stato, simbolo di ingresso}* è associato, se specificato, uno ed uno solo stato futuro.
- Funzione d'uscita ω
  - Genera il simbolo d'uscita
  - Macchine di Mealy. L'uscita dipende sia dallo stato sia dall'ingresso:

    $$\omega : S \times I \rightarrow U$$

  - Macchine di Moore. L'uscita dipende solamente dallo stato:

    $$\omega : S \rightarrow U$$

Tabella degli stati

- Una macchina sequenziale può essere descritta mediante la *Tabella degli stati*
- Indici di colonna sono i simboli di ingresso $i_\alpha \in I$
- Indici di riga sono i simboli di stato $s_j \in S$ che indicano lo stato presente
- Elementi sono:
  - Macchine di Mealy: La coppia $\{u_\beta, s_j\}$:
    - $u_\beta = w(i_\alpha, s_j)$ è il simbolo di uscita
    - $s_j = \tau(i_\alpha, s_j)$ è il simbolo stato prossimo
  - Macchine di Moore: Il simbolo stato prossimo s_j :
    - $s_j = w(i_\alpha, s_j)$ è il simbolo stato prossimo
- Nelle macchine di Moore i simboli d'uscita sono associati allo stato presente

Tabella degli stati

- Macchine di Mealy

	i_1	i_2	..
S_1^t	S_j^{t+1} / u_j	S_k^{t+1} / u_k	....
S_2^t	S_m^{t+1} / u_m	S_l^{t+1} / u_l	....
..	....	....	....

- Macchine di Moore

	i_1	i_2	..	
S_1^t	S_j^{t+1}	S_k^{t+1}	....	u_1
S_2^t	S_m^{t+1}	S_l^{t+1}	....	u_2
..	....	....	....	....

Diagramma degli stati

- Spesso, la stesura della *Tabella degli stati* è preceduta da una rappresentazione grafica ad essa equivalente, denominata *Diagramma degli stati*
- Il Diagramma degli stati è un *grafo orientato G(V,E,L)*
  - *V* - Insieme dei *nodi*
    - Ogni nodo rappresenta uno stato
    - Ad ogni nodo è associato un simbolo d'uscita (macchine di Moore)
  - *E* - Insieme degli *archi*
    - Ogni arco rappresenta le transizioni di stato
  - *L* - Insieme degli:
    - Ingressi e Uscite (macchine di Mealy)
    - Ingressi (macchine di Moore)

Esempio – Macchina di Mealy

- Questo esempio mostra l'equivalenza delle due rappresentazioni nel caso di una macchina di Mealy

Diagramma degli stati

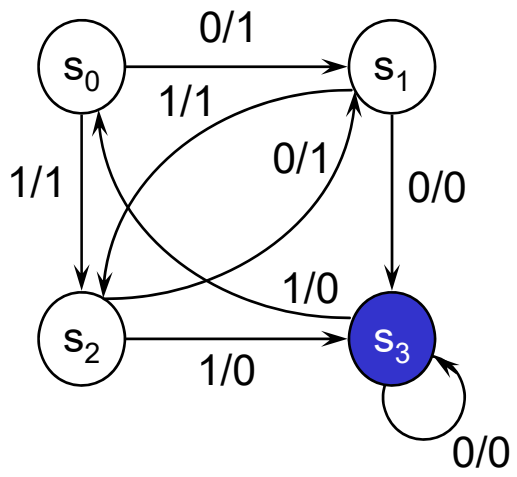


Tabella degli stati

	0	1
S_0	$S_1/1$	$S_2/1$
S_1	$S_3/0$	$S_2/1$
S_2	$S_1/1$	$S_3/0$
S_3	$S_3/1$	$S_0/0$

Esempio – Macchina di Moore

- Questo esempio mostra l'equivalenza delle due rappresentazioni nel caso di una macchina di Moore

Diagramma degli stati

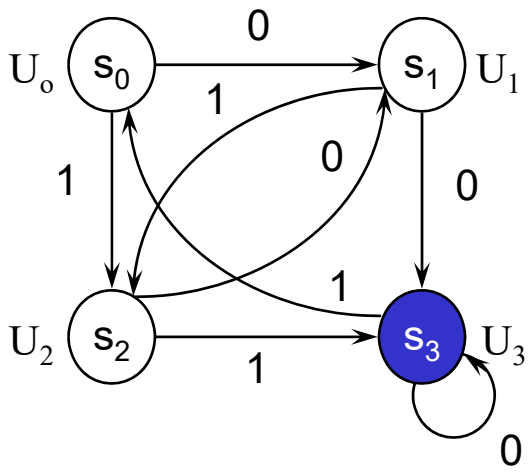


Tabella degli stati

	0	1	U
S_0	S_1	S_2	U_o
S_1	S_3	S_2	U_1
S_2	S_1	S_3	U_2
S_3	S_3	S_0	U_3

Codifica degli Stati (1/2)

- La minimizzazione del numero di stati consente di ridurre il numero di elementi di memoria necessari a codificare gli stati stessi
  - A valle della minimizzazione realizzo la tabella degli stati
- E' necessario codificare gli stati, cioè indicare come sono rappresentati sui flip-flop:
  - Tabella delle transizioni: per un dato input indica le codifica dello stato successivo
  - Tabella delle eccitazioni: per un dato input indica l'ingresso ai flip-flop che codificano lo stato prossimo
  - Nota: se uso Flip-Flop D le due tabelle coincidono

E.g. per input x=0

transizione S1(0,1)->S2(1,0)

Quali flip-flop sto usando nelle tabelle di eccitazione quì indicate?

Stato	X=0
(0,1)	(1,0)
(1,0)	…

Stato	X=0
(0,1)	(1,1)
(1,0)	…

Codifica degli Stati (2/2)

- La codifica influenza la complessità della rete combinatoria perchè lo stato corrente è input alla rete e, inoltre, l'uscita della rete è eccitazione dei flip-flop per lo stato successivo
- Scegliere la lunghezza del codice -> numero di flip-flop
  - Il minimo è l'intero superiore di $log_2 |S|$, con |S| numero degli stati
- *Codifica sparsa* con un bit alto per stato -> tanti bit quanti sono gli stati (|S|=4): 0001, 0010, 0100, 1000
- Criteri di codifica a conteggio binario con numero di bit minimo :
  - E.g. (|S|=5) S0=000, S1=001, S2=010, S3=011, S4=100, S5=101
- Criteri a minima distanza (adiacenza):
  - Due stati con uguale stato prossimo a seguito dello stesso ingresso, devono avere codifiche adiacenti (1bit di differenza)
  - Stati prossimi dello stesso stato con ingressi adiacenti devono essere codificati con codifica adiacente
  - Stati con uscite uguali per il medesimo ingresso devono avere codifiche adiacenti

Sintesi di una macchina sequenziale

- La sintesi si svolge nei seguenti passi:
  1. Realizzazione del *diagramma degli stati* a partire dalle specifiche informali del problema
  2. Costruzione della *tabella degli stati*
  3. Riduzione del numero degli stati: *ottimizzazione*
  4. Assegnamento degli stati: *codifica*
  5. Costruzione della *tabella delle eccitazioni*
  6. Sintesi della rete combinatoria che realizza la funzione stato prossimo
  7. Sintesi della rete combinatoria che realizza la funzione d'uscita

Esempio: contatore mod 4

- Uscita z=1 se il contatore conta 2.
- L'input x determina se il conteggio va in avanti o all'indietro (0 o 1)
- Il Clock scandisce il conteggio (per es. fronte negativo)
- Le variabili per rappresentare gli stati sono dette *variabili di stato*

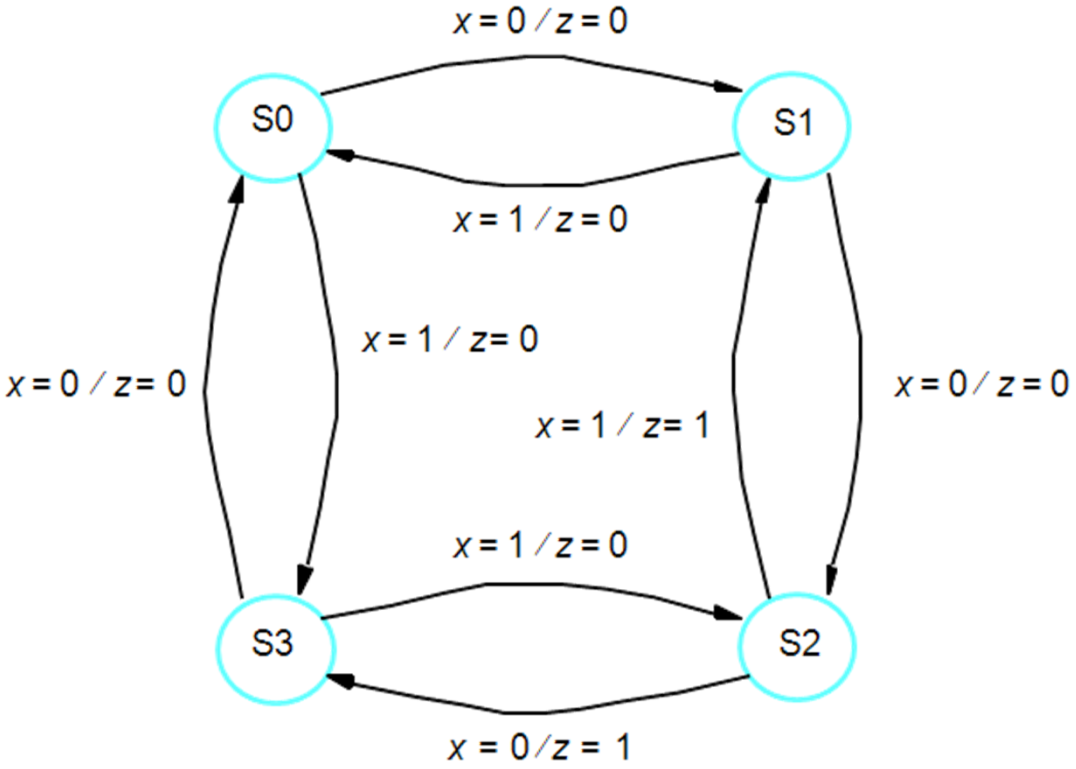


Tabelle degli stati e circuito

Present state	Next state		Output z	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
S0	S1	S3	0	0
S1	S2	S0	0	0
S2	S3	S1	1	1
S3	S0	S2	0	0

Present state	Next state		Output z	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
$y_2 y_1$	$Y_2\ Y_1$	$Y_2\ Y_1$		
0 0	0 1	1 1	0	0
0 1	1 0	0 0	0	0
1 0	1 1	0 1	1	1
1 1	0 0	1 0	0	0

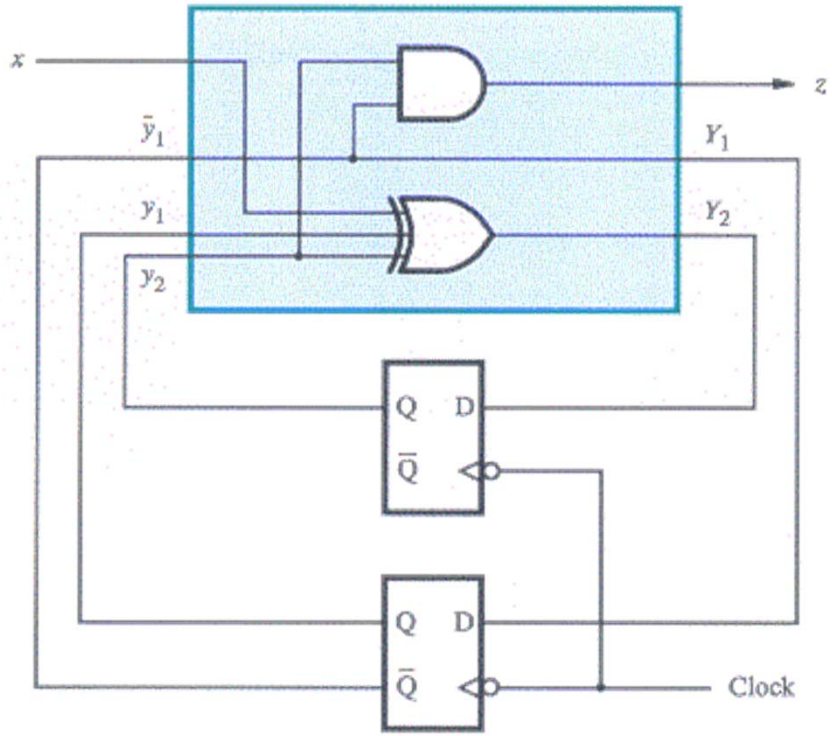


Figure A.50. Implementation of the up/down counter.

Diagramma di temporizzazione e FSM

- Osserviamo che la rete combinatoria non introduce ritardo nella variazione delle variabili di stato, a differenza della rete sequenziale (in realtà sui gates c'è, ma è trascurabile rispetto ai flip flop)
  - *Macchina a stati finiti (FSM)*
- Se le variabili del circuito sono comandate dallo stesso clock si parla di *circuiti sequenziali sincroni*

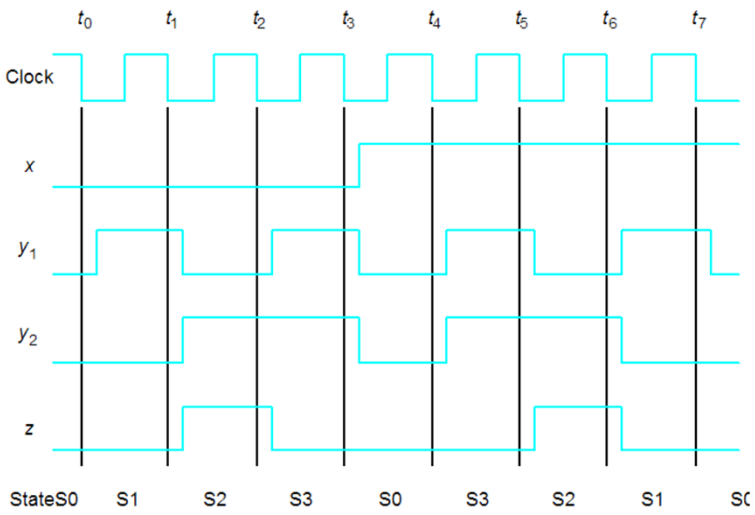


Figure A.51. Timing diagram for the circuit in Figure A.50.

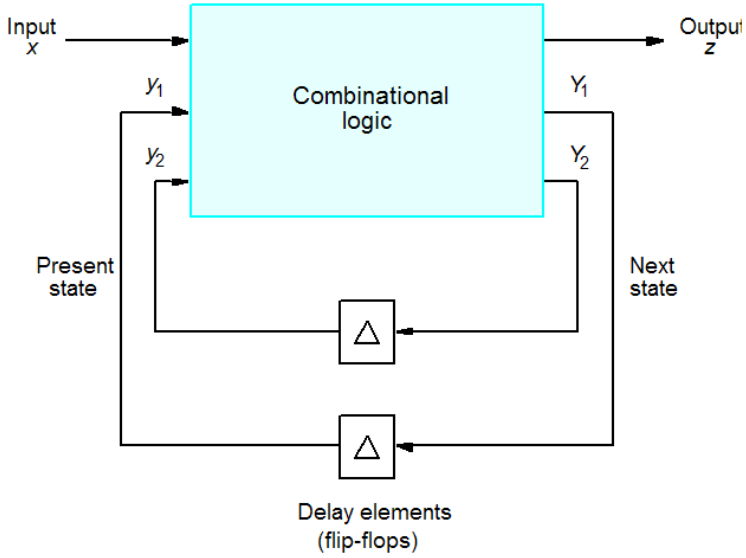


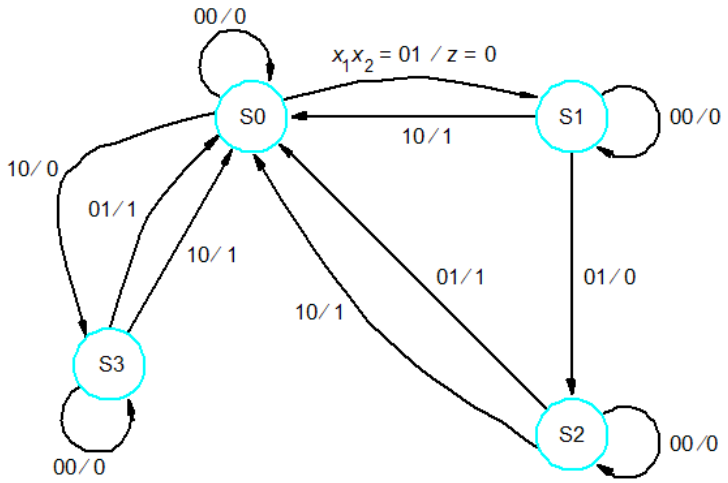
Figure A.52. A formal model of a finite state machine.

Esempio di FSM (1/2)

- Distributore automatico a 30 cents (erogazione prodotto). Accetta 25 e 10 cents (mai assieme) e non dà resto

Esempio di FSM (1/2)

- Distributore automatico a 30 cents (erogazione prodotto, z). Accetta 25 (x1) e 10 (x2) cents (mai assieme) e non dà resto



$x_1 = 1$ ~ quarter deposited
$x_2 = 1$ ~ dime deposited
$z = 1$ ~ dispense merchandise (i.e., a total of 30 cents deposited)
Input combination $x_1 x_2$ = 11 cannot occur

Present state	Next state				Outputz			
	$x_1x_2=00$	$x_1x_2=01$	$x_1x_2=10$	$x_1x_2=11$	$x_1x_2=00$	$x_1x_2=01$	$x_1x_2=10$	$x_1x_2=11$
y_2y_1	$Y_2\,Y_1$	$Y_2\,Y_1$	$Y_2\,Y_1$	$Y_2\,Y_1$				
S0	0 0	0 0	0 1	1 1	-	0	0	0	-
S1	0 1	0 1	1 0	0 0	-	0	0	1	-
S2	1 0	1 0	0 0	0 0	-	0	1	1	-
S3	1 1	1 1	0 0	0 0	-	0	1	1	-

Figure A.54. Assigned state table for the vending machine example.

Esempio di FSM (2/2)

Present state	Next state				Outputz			
	$x_1x_2=00$	$x_1x_2=01$	$x_1x_2=10$	$x_1x_2=11$	$x_1x_2=00$	$x_1x_2=01$	$x_1x_2=10$	$x_1x_2=11$
y_2y_1	Y_2Y_1	Y_2Y_1	Y_2Y_1	Y_2Y_1				
S0 0 0	0 0	0 1	1 1	-	0	0	0	-
S1 0 1	0 1	1 0	0 0	-	0	0	1	-
S2 1 0	1 0	0 0	0 0	-	0	1	1	-
S3 1 1	1 1	0 0	0 0	-	0	1	1	-

Figure A.54. Assigned state table for the vending machine example.

x_1	x_2	y_2	y_1	Y_2	Y_1	z
0	0	0	0	0	0	0
0	0	0	1	0	1	0
0	0	1	0	1	0	0
0	0	1	1	1	1	0
0	1	0	0	0	1	0
0	1	0	1	1	0	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	1	0
1	0	0	1	0	0	1
1	0	1	0	0	0	1
1	0	1	1	0	0	1
1	1	0	0	d	d	d
1	1	0	1	d	d	d
1	1	1	0	d	d	d
1	1	1	1	d	d	d

Esercizio

- Si vuole realizzare il Sistema di controllo di una porta automatica dotata di:
  - Sensore di presenza (apertura porta)
  - Sensore di pressione (blocca la chiusura della porta)
  - Sensori di fine corsa (porta aperta/porta chiusa)
  - Pulsante di apertura forzata (tiene la porta aperta anche in assenza di persone)
  - Serratura di chiusura porta (tiene la porta chiusa, ad esempio durante la notte)

Dispositivi programmabili (PLD)

- Programmable Logic Device (PLD): array di elementi programmabili che implementano funzioni in forma di somme di prodotti

• PLA: programmabile sia sulle AND che sulle OR
• PAL: programmabile sugli input alle AND ma non sulle OR
• CPLD: interconnessione di più PAL
• FPGA: realizzano circuiti logici più complessi (RAM, ROM, sistemi embedded) e veloci. Sono molto facili da sviluppare su VLSI, rispetto a quelli custom.

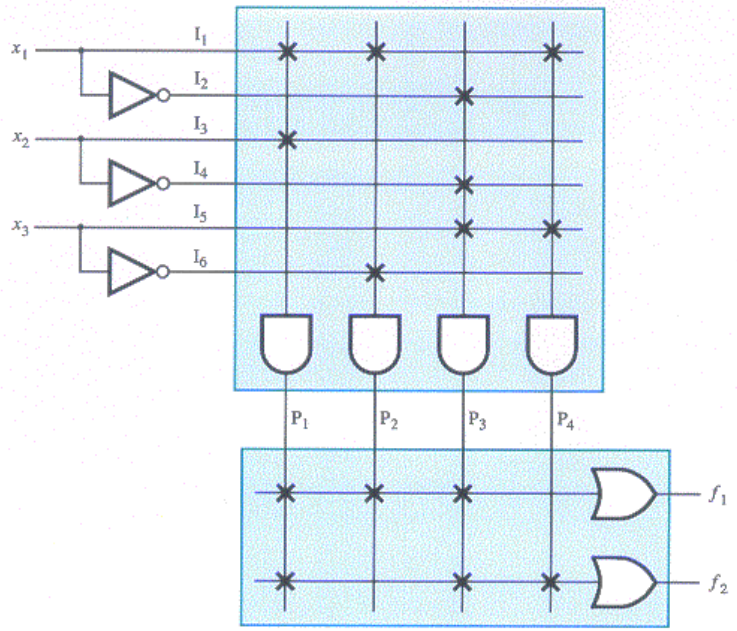


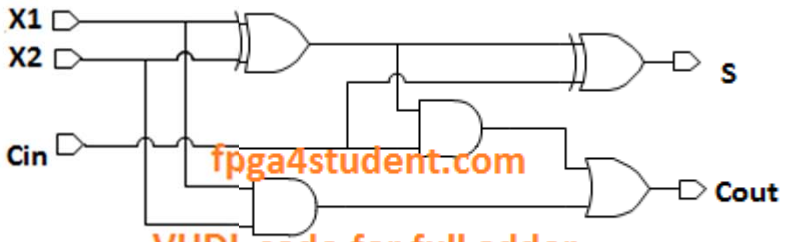
Figure A.42. A simplified sketch of the PLA in Figure A.41.

Hardware Description Languages

- Sono linguaggi sviluppati per descrivere la struttura e il comportamento di circuiti digitali
- Consentono la simulazione e la sintesi di sistemi digitali
- Possono essere usati in congiunzione con una FPGA per una prototipazione veloce di sistemi hw o per dotare un Sistema di HW configurabile
- I principali HDL sono
  - Verilog
  - VHDL

Esempio di descrizione strutturale di un full-adder

```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity Full_Adder_Structural_VHDL is
  port(
 X1, X2, Cin : in std logic;
 S, Cout : out std_logic
 );
end Full_Adder_Structural_VHDL;
architecture structural of Full_Adder_Structural_VHDL is
signal a1, a2, a3: std_logic;
begin
  a1 <= X1 xor X2;
  a2 <= X1 and X2;
  a3 <= a1 and Cin;
  Cout <= a2 or a3;
  S <= a1 xor Cin;
end structural:
```



VHDL code for full adder

Testbench per il full-adder

```
Library IEEE;
USE IEEE.Std_logic_1164.all;
-- fpga4student.com
-- FPGA projects, VHDL projects, Verilog projects
-- VHDL code for full adder
-- Testbench code of the structural code for full adder
entity Testbench structural adder is
end Testbench_structural_adder;

architecture  behavioral of Testbench structural_adder is
 component Full_Adder_Structural_VHDL
   port(
  X1, X2, Cin : in std logic;
  S, Cout : out std_logic
  );
 end component;
 signal A,B,Cin: std logic:='0';
 signal S,Cout: std_logic;
begin
 structural_adder: Full_Adder_Structural_VHDL port map
   (
    X1 => A,
    X2 => B,
    Cin => Cin,
    S => S,
    Cout => Cout
   );
  process
begin
 A <= '0';
 B <= '0';
 Cin <= '0';
 wait for 100 ns;
 A <= '0';
 B <= '0';
 Cin <= '1';
 wait for 100 ns;
 A <= '0';
 B <= '1';
 Cin <= '0';
 wait for 100 ns;
 A <= '0';
 B <= '1';
 Cin <= '1';
 wait for 100 ns;
 A <= '1';
 B <= '0';
 Cin <= '0';
 wait for 100 ns;
 A <= '1';
 B <= '0';
 Cin <= '1';
 wait for 100 ns;
 A <= '1';
 B <= '1';
 Cin <= '0';
 wait for 100 ns;
 A <= '1';
 B <= '1';
 Cin <= '1';
 wait for 100 ns;
end process;

end behavioral;
```

Esempio di descrizione behavioral di un full-adder

```
 library ieee;
 use ieee.std_logic_1164.all;
 use IEEE.STD_LOGIC_unsigned.ALL;
use IEEE.NUMERIC_STD.ALL;
 entity Full_Adder_Behavioral_VHDL is
   port(
  X1, X2, Cin : in std_logic;
  S, Cout : out std_logic
  );
 end Full_Adder_Behavioral_VHDL;
 architecture Behavioral of Full_Adder_Behavioral_VHDL is
 signal tmp: std_logic_vector(1 downto 0);
 begin
   process(X1,X2,Cin)
   begin
 tmp <= ('0'& X1) + ('0'& X2) +('0'& Cin) ;
   end process;
   S <= tmp(0);
   Cout <= tmp(1);
 end Behavioral;
 Library IEEE;
USE IEEE.Std_logic_1164.all;
-- fpga4student.com
-- FPGA projects, VHDL projects, Verilog projects
-- VHDL code for full adder
-- Testbench code of the behavioral code for full adder
entity Testbench_behavioral_adder is
end Testbench_behavioral_adder;
```