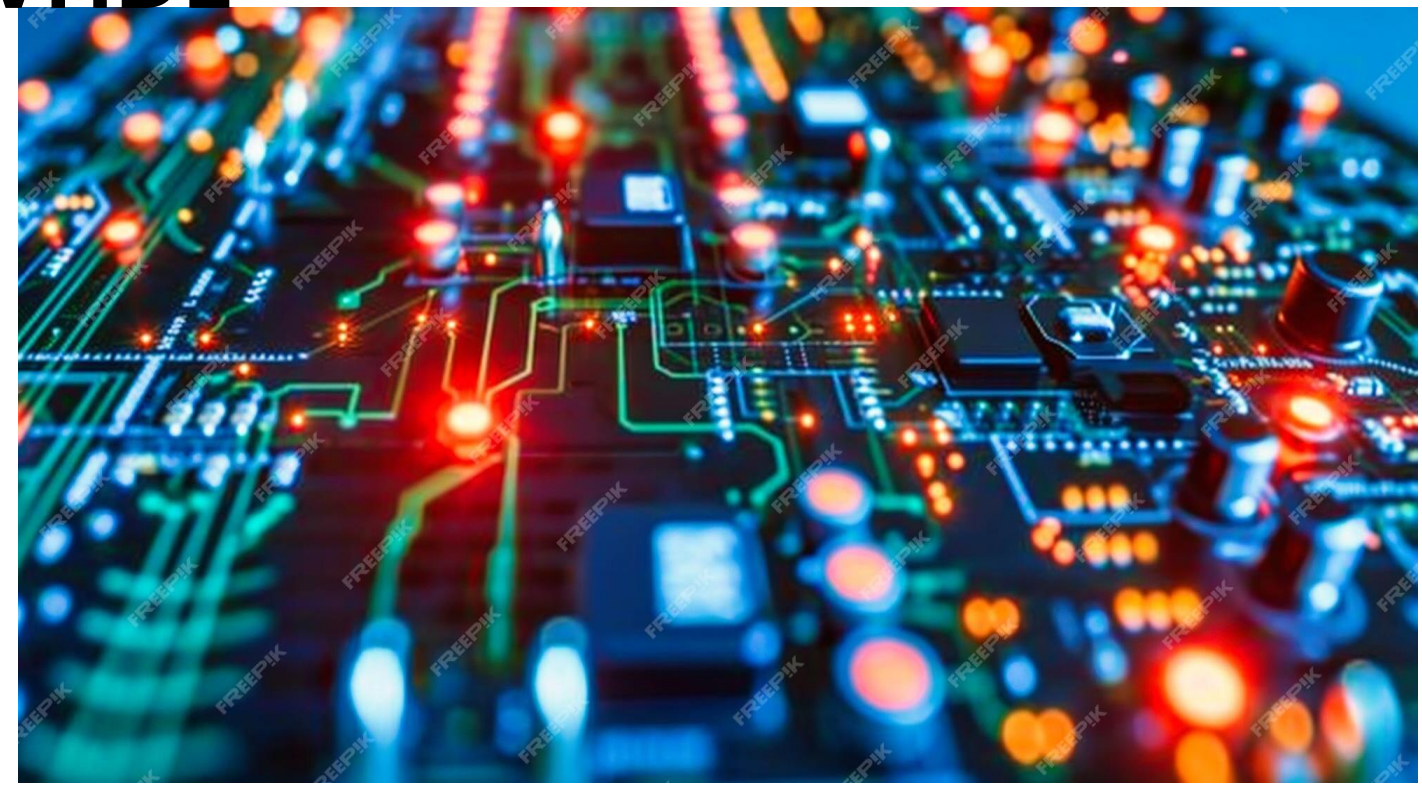
LAE1 VHDL

LAB7

FSM



https://homes.di.unimi.it/re/Corsi/lae1_25_26/

a.a. 2025-26

matteo.re@unimi.it

LAE1 VHDL

LAB7

FSM

Per ottenere il PDF delle slide:
https://homes.di.unimi.it/re/Corsi/lae1_25_26/L6.pdf

Per scaricare il codice di esempio direttamente nell'ambiente di esercitazione dopo aver effettuato il login utilizzare i seguenti comandi:

wget http://homes.di.unimi.it/re/Corsi/lae1_25_26/L7.tar.gz
tar –xvzf L7.tar.gz
cd L7

a.a. 2025-26

matteo.re@unimi.it

Obiettivi di apprendimento

- **Processo di definizione di una FSM**
  - Definizione FSM
  - Breve ripasso teoria
  - STG. STT, STT codificata
  - Logica di stato prossimo e logica di output
- **Descrizione di FSM in VHDL**
  - Descrizione di FSM in VHDL mono segmento e multi segmento
  - Serial bit sequence detector SBSdet (Mealy / multi segmento)
  - Serial bit sequence detector con sequenze di lunghezza diversa (Moore / mono segmento)

FSM : finite state machine

Agomenti degli ultimi incontri: logica sequenziale in VHDL, flip-flop, registri.

- La disponibilita' di elementi di memoria permette di immagazzinare informazione in modo **stabile**.
- L'informazione "stabile" permette di descrivere lo **stato** di un sistema (la rete logica) e di osservarne/attuarne le variazioni in **diversi istanti di tempo**.
- Alcune reti logiche possono muoversi lungo un **percorso prefissato** di stati e compiere operazioni complesse. Il percorso e' composto da un **insieme finito** di stati. Queste reti logiche sequenziali vengono dette **MACCHINE A STATI FINITI**.

FSM : definizione

Sistema matematico

$M = \{I, U, S, F, G\}$

formato da 3 INSIEMI

$I: \{i_1, i_2, \ldots, i_n\}$ *alfabeto di ingresso*

$U: \{u_1, u_2, \ldots, u_m\}$ *alfabeto di uscita*

$S: \{s_1, s_2, \ldots, s_k\}$ *insieme degli stati*

da 2 FUNZIONI

$F : S \times I \rightarrow U$ *funzione di uscita*

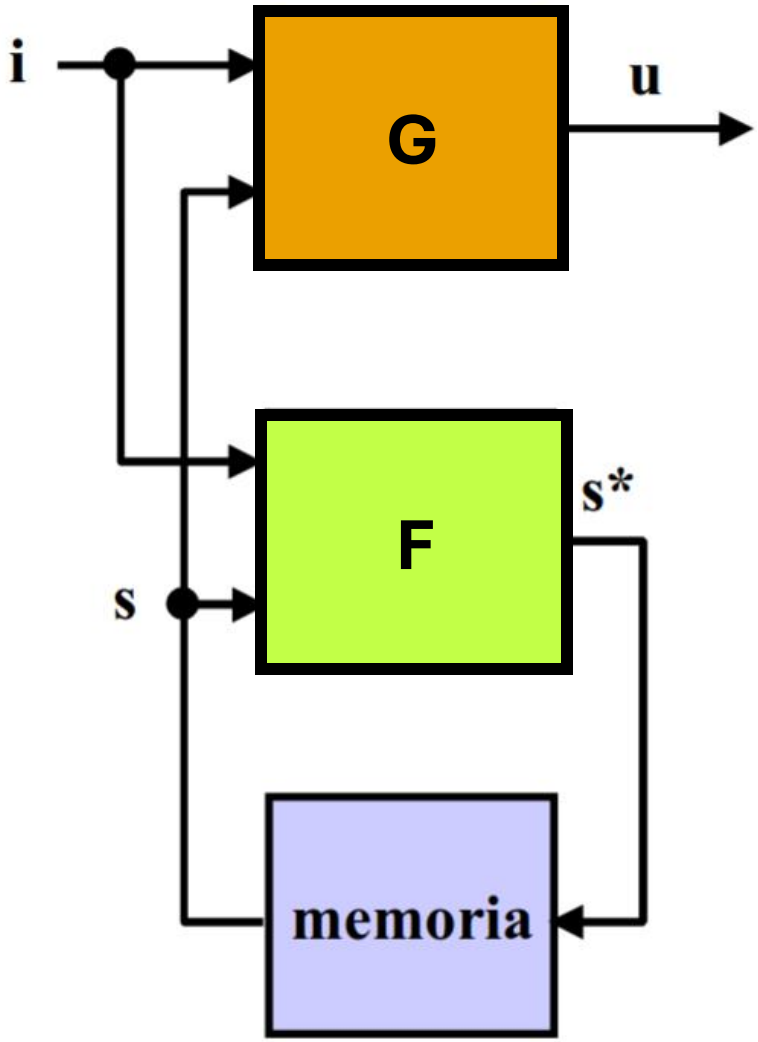
$G : S \times I \rightarrow S$ *funzione di stato*

e da una MEMORIA

che mantiene il "vecchio stato" s

fino a quando non è necessario

*sostituirlo con il "nuovo stato" s**



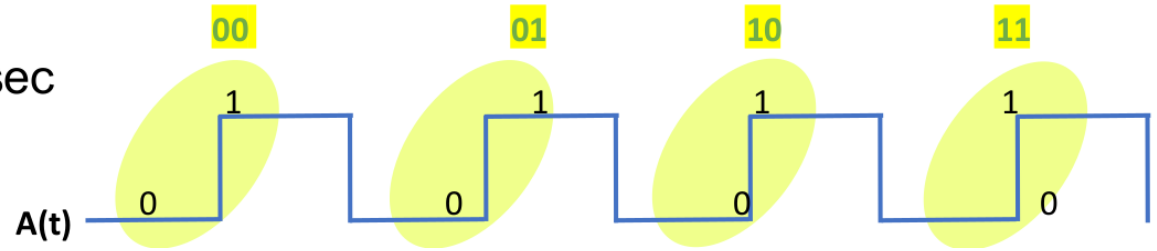
FSM : ripasso teoria (esempio)

Si sintetizzi una macchina a stati finiti di Moore che realizza un contatore modulo 4 che conta i fronti di salita di un segnale A(t) fornito sulla linea in ingresso. Il valore del segnale A(t) viene osservato ogni millisecondo. L'uscita è costituita da 2 linee che rappresentano, in codice binario, il valore del contatore.

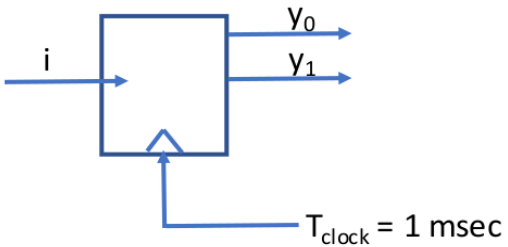
1) Dati rilevanti estrapolati dal testo:

"Fronti di salita": ingresso precedente = 0, ingresso attuale = 1

"Ogni msec": T_{clock} = 1 msec



2) Black box:



3) Determinare insiemi valori in ingresso e uscita

I = { 0, 1 }

Y = { 0^{00}, 1^{01}, 2^{10}, 3^{11} }

FSM : ripasso teoria (esempio)

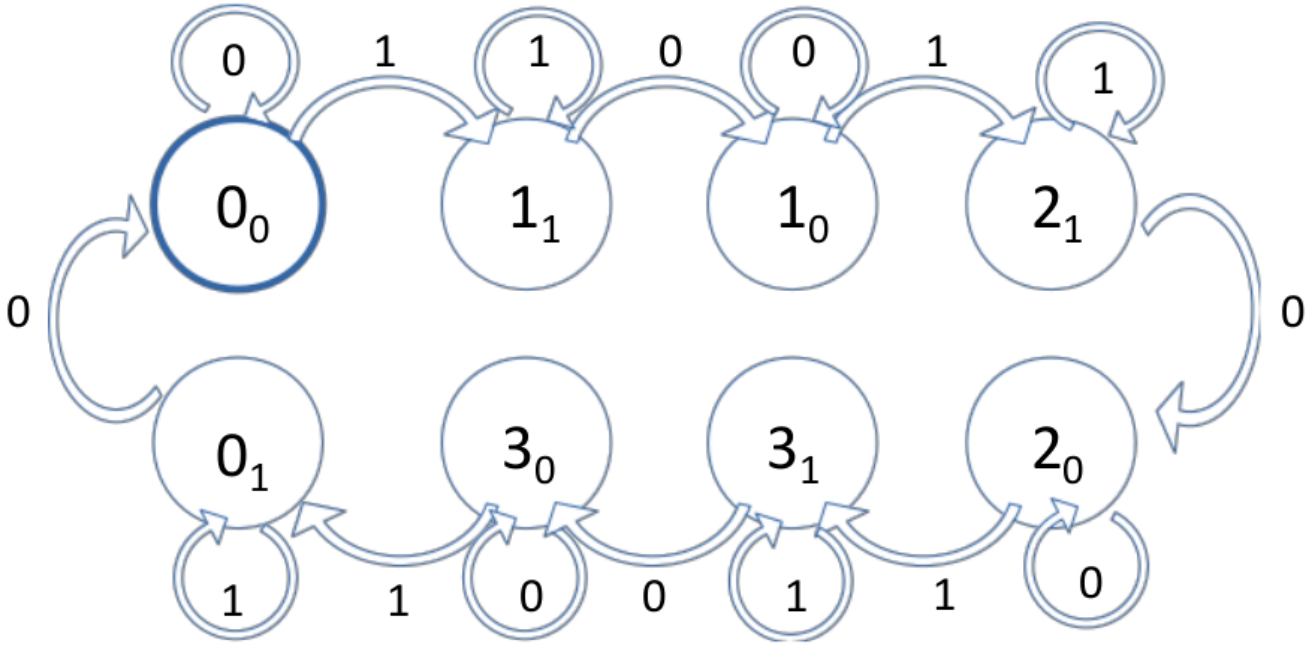
STG:

Stato iniziale:
I=0, N=0

num. stati : 8

i valori in ingresso

per y valori in uscita



FSM : ripasso teoria (esempio)

STT :

	I		Y	
	0	1		
Stato				
0_1	0_0	0_1	0	0
0_0	0_0	1_1	0	0
1_1	1_0	1_1	0	1
1_0	1_0	2_1	0	1
2_1	2_0	2_1	1	0
2_0	2_0	3_1	1	0
3_1	3_0	3_1	1	1
3_0	3_0	0_1	1	1

FSM : ripasso teoria (esempio)

STT codificata :

Assegniamo ad ogni stato una codifica binaria

x_2	x_1	x_0
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

	I		Y	
	0	1		
Stato				
0_1	0_0	0_1	0	0
0_0	0_0	1_1	0	0
1_1	1_0	1_1	0	1
1_0	1_0	2_1	0	1
2_1	2_0	2_1	1	0
2_0	2_0	3_1	1	0
3_1	3_0	3_1	1	1
3_0	3_0	0_1	1	1

Gli stati necessari sono 8 : codifica su 3 bit

FSM : ripasso teoria (esempio)

STT codificata :

Assegniamo ad ogni stato una codifica binaria

x_2	x_1	x_0
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

	I		Y	
	0	1		
Stato				
000	0_0	000	0	0
0_0	0_0	1_1	0	0
1_1	1_0	1_1	0	1
1_0	1_0	2_1	0	1
2_1	2_0	2_1	1	0
2_0	2_0	3_1	1	0
3_1	3_0	3_1	1	1
3_0	3_0	000	1	1

Gli stati necessari sono 8 : codifica su 3 bit

FSM : ripasso teoria (esempio)

STT codificata :

Assegniamo ad ogni stato una codifica binaria

		I						Y	
			0			1			
x_2	x_1	x_0							
0	0	0	0	0	1	0	0	0	0
0	0	1	0	0	1	0	1	0	0
0	1	0	0	1	1	0	1	0	1
0	1	1	0	1	1	1	0	0	1
1	0	0	1	0	1	1	0	0	1	0
1	0	1	1	0	1	1	1	0	1	0
1	1	0	1	1	1	1	1	0	1	1
1	1	1	1	1	1	0	0	0	1	1

FSM : ripasso teoria (esempio)

STT codificata :

Assegniamo ad ogni stato una codifica binaria

		I							Y	
			0			1				
x_2	x_1	x_0								
0	0	0	0	0	1	0	0	0	0	0
0	0	1	0	0	1	0	1	0	0	0
0	1	0	0	1	1	0	1	0	0	1
0	1	1	0	1	1	1	0	0	0	1
1	0	0	1	0	1	1	0	0	1	0
1	0	1	1	0	1	1	1	0	1	0
1	1	0	1	1	1	1	1	0	1	1
1	1	1	1	1	1	0	0	0	1	1
			x_{2^*}	x_{1^*}	x_{0^*}				y_1	y_0

Funzioni di uscita :

$$Y = g(x) : \begin{cases} y_1 = x_2 \\ y_0 = x_1 \end{cases}$$

FSM : ripasso teoria (esempio)

STT codificata :

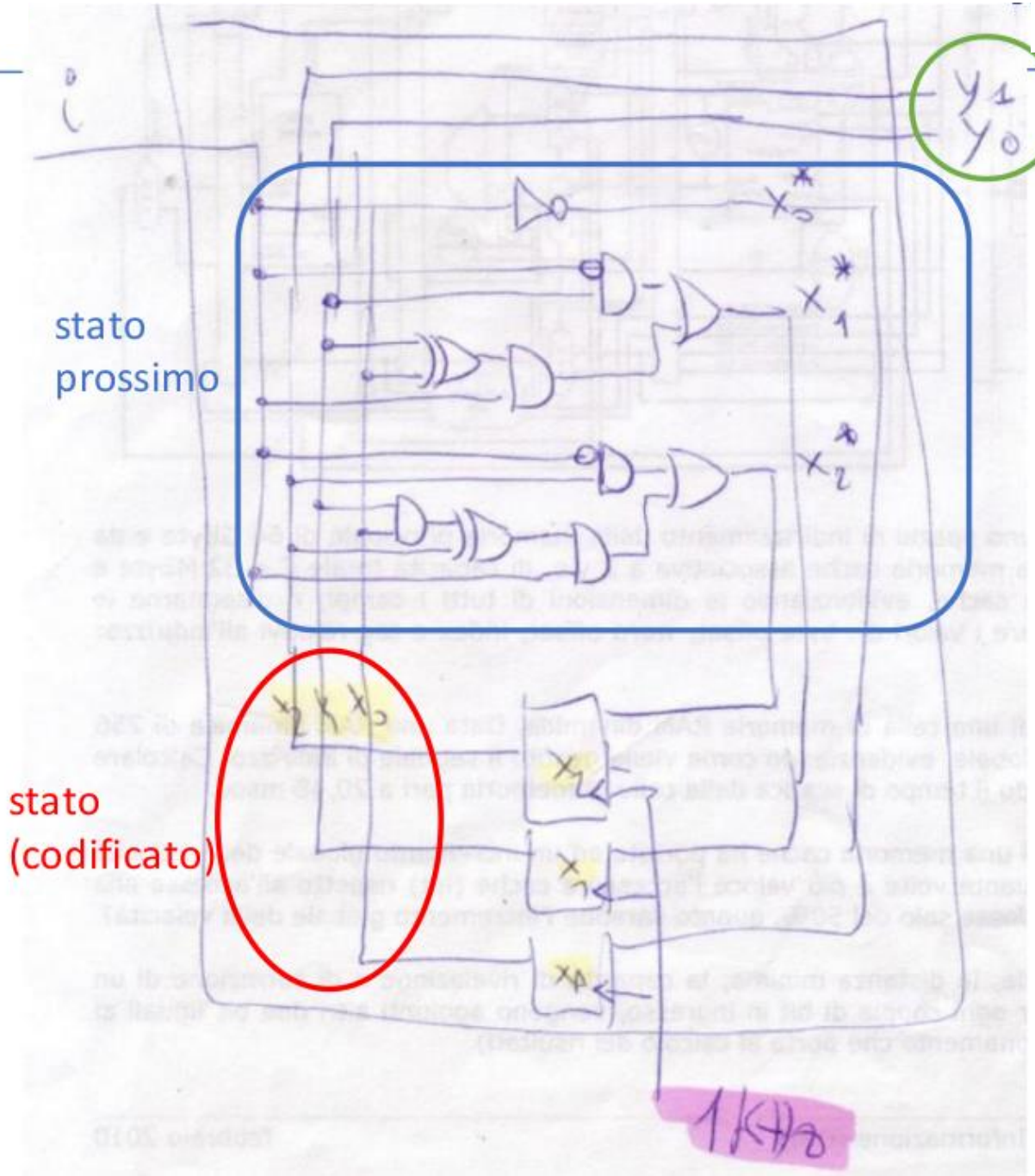
Assegniamo ad ogni stato una codifica binaria

	I	0			1			Y	
x_2	x_1	x_0							
0	0	0	0	0	1	0	0	0	0	0
0	0	1	0	0	1	0	1	0	0	0
0	1	0	0	1	1	0	1	0	0	1
0	1	1	0	1	1	1	0	0	0	1
1	0	0	1	0	1	1	0	0	1	0
1	0	1	1	0	1	1	1	0	1	0
1	1	0	1	1	1	1	1	0	1	1
1	1	1	1	1	1	0	0	0	1	1
			x_{2^*}	x_{1^*}	x_{0^*}				y_1	y_0

Funzioni di stato prossimo :

$$X^* = f(X,I): \begin{cases} x_{0^*} = \sim i \\ x_{1^*} = \sim i x_1 + i(x_0 \oplus x_1) \\ x_{2^*} = \sim i x_2 + i(x_2 \oplus x_0 x_1) \end{cases}$$

FSM :
ripasso
teoria



FSM : finite states machine

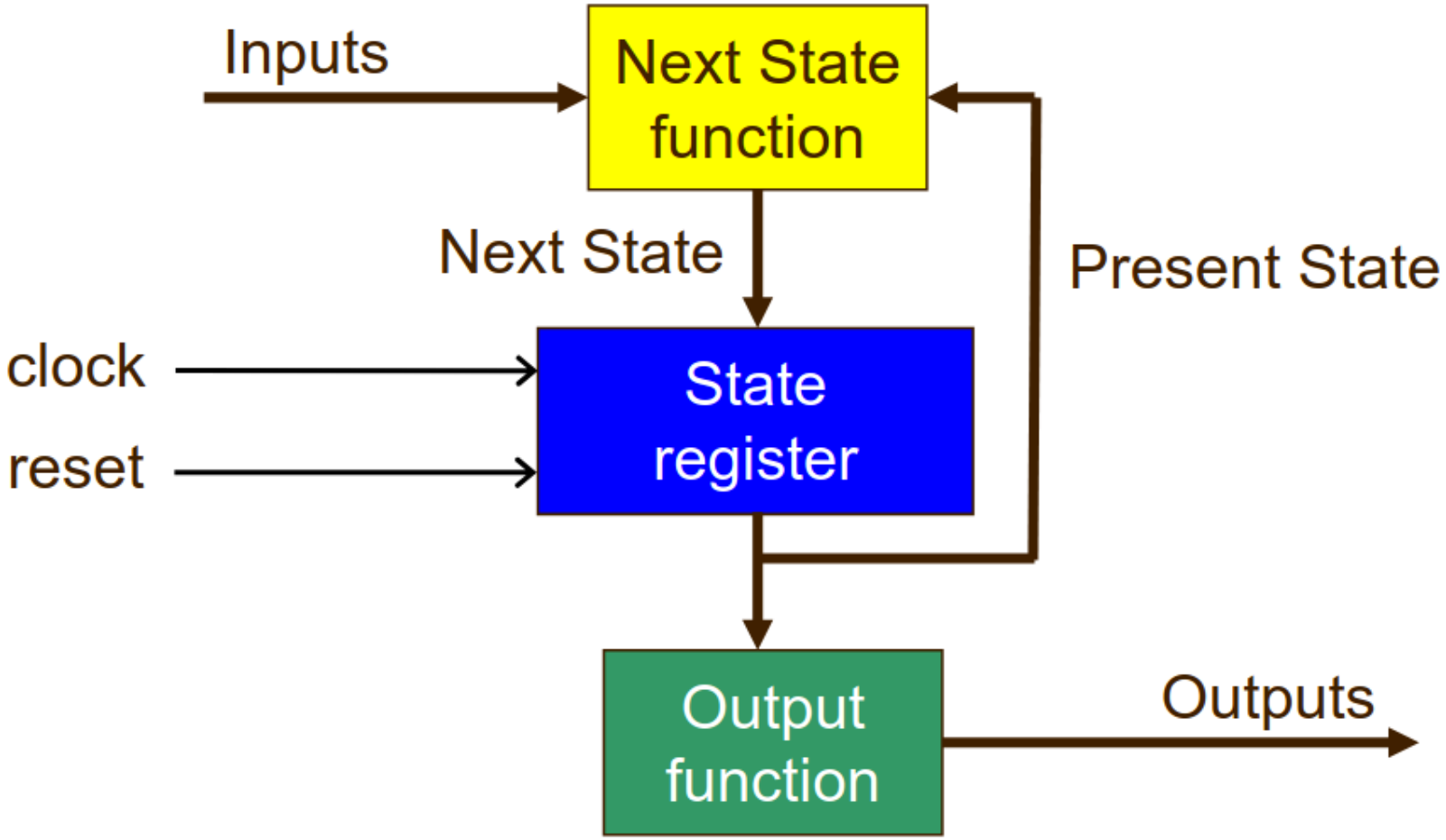
Due categorie principali di FSM. Moore e Mealy.

Differenza principale: in FSM Moore output dipende unicamente da stato corrente. In FSM Mealy dipende da stato corrente + input corrente

Altre differenze : Mealy , a parita' di problema considerato, richiede un numero minore di stati rispetto a Moore. Mealy e' piu' reattiva (calcola output non appena I valori in input cambiano) risponde con un ciclo di clock di anticipo rispetto a Moore. Ma lo fa aggiungendo complessita' (incremento cammino critico).

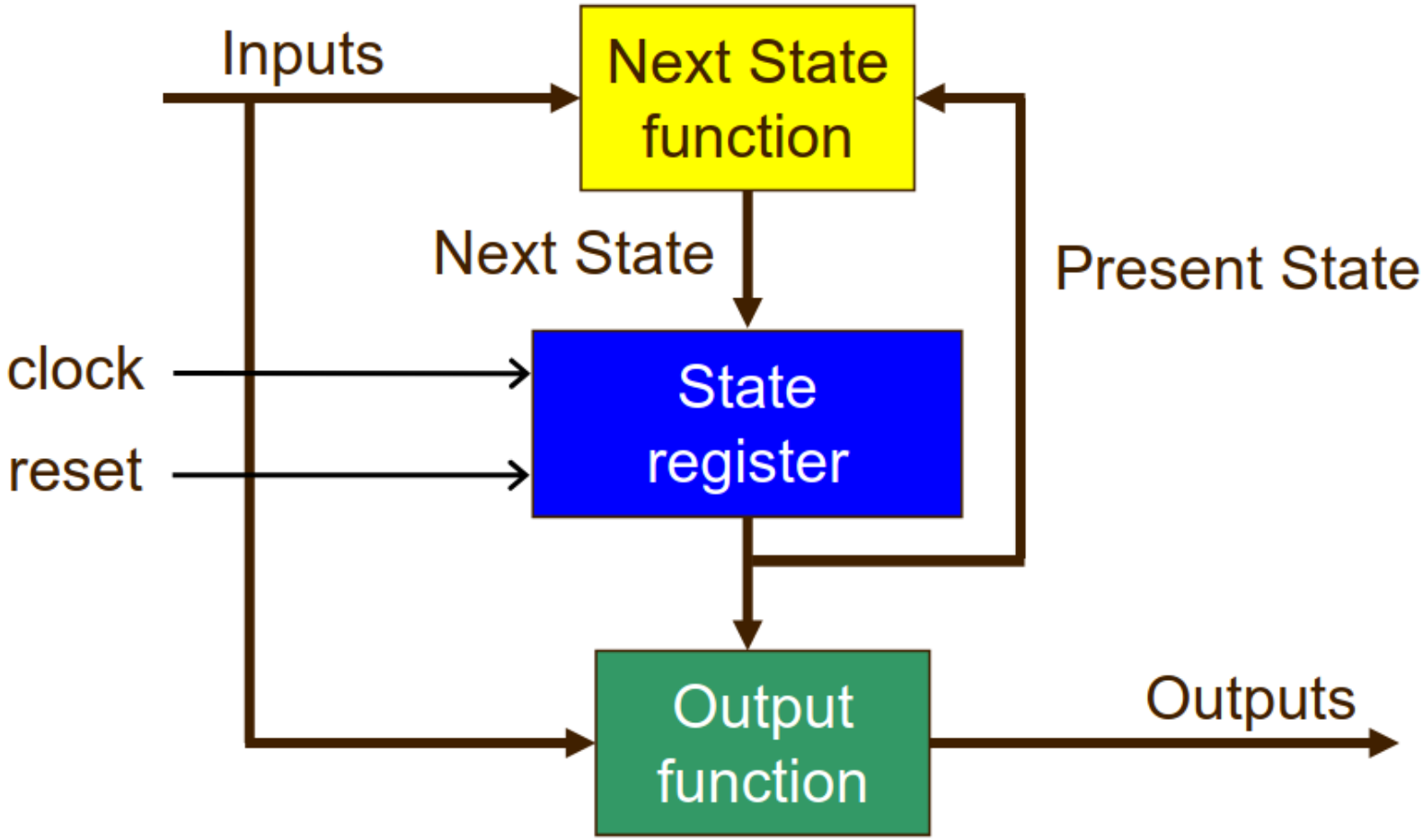
FSM : finite state machine

Schema astratto di una FSM **Moore** :



FSM : finite state machine

Schema astratto di una FSM **Mealy** :



Definizione di una FSM - es. 1

Descrizione testuale della FSM:

"Il circuito **osserva** una linea in ingresso ad ampiezza 1 bit che rappresenta informazione ricevuta in modo **seriale** (1 bit alla volta). L'informazione ricevuta viene letta a gruppi di **3 bit**. Una sequenza **111** rappresenta un messaggio dell'unita' di trasmissione che segnala un problema avvenuto sulla linea di trasmissione. Il circuito "osservatore" dei dati dovra' attivare una linea di output di nome **ERR** per segnaare il problema."

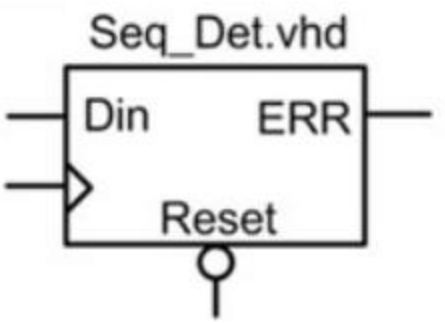
Descrizione di una FSM - es. 1

Descrizione testuale della FSM:

Questa FSM e' un esempio di **sequence detector** (Seq_Det). I possibili valori sulle sue linee di input (Din) e output (ERR) sono I seguenti:

1)

Din= {0,1} **ERR** = {0,1}



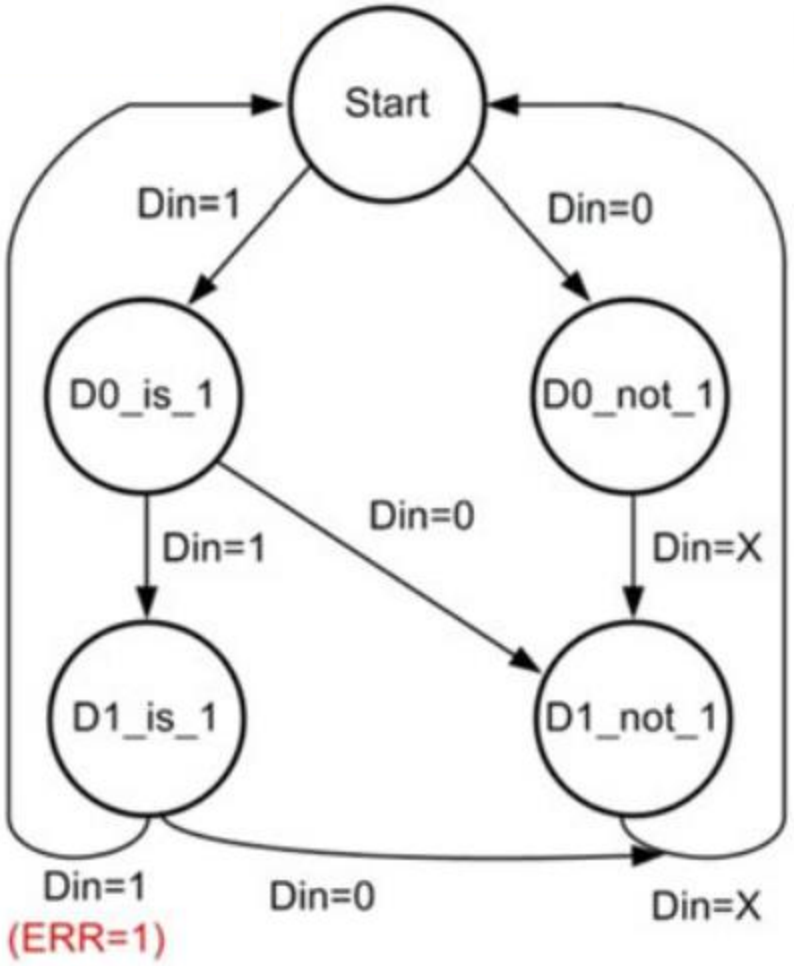
Come per tutti I circuiti e' possibile una rappresentazione di tipo black **box**.

Din : ... , D0, D1, D2, D0, D1, D2, D0, D1, D2, ...

bit sequence #1 **bit sequence #2** **bit sequence #3**

Descrizione di una FSM

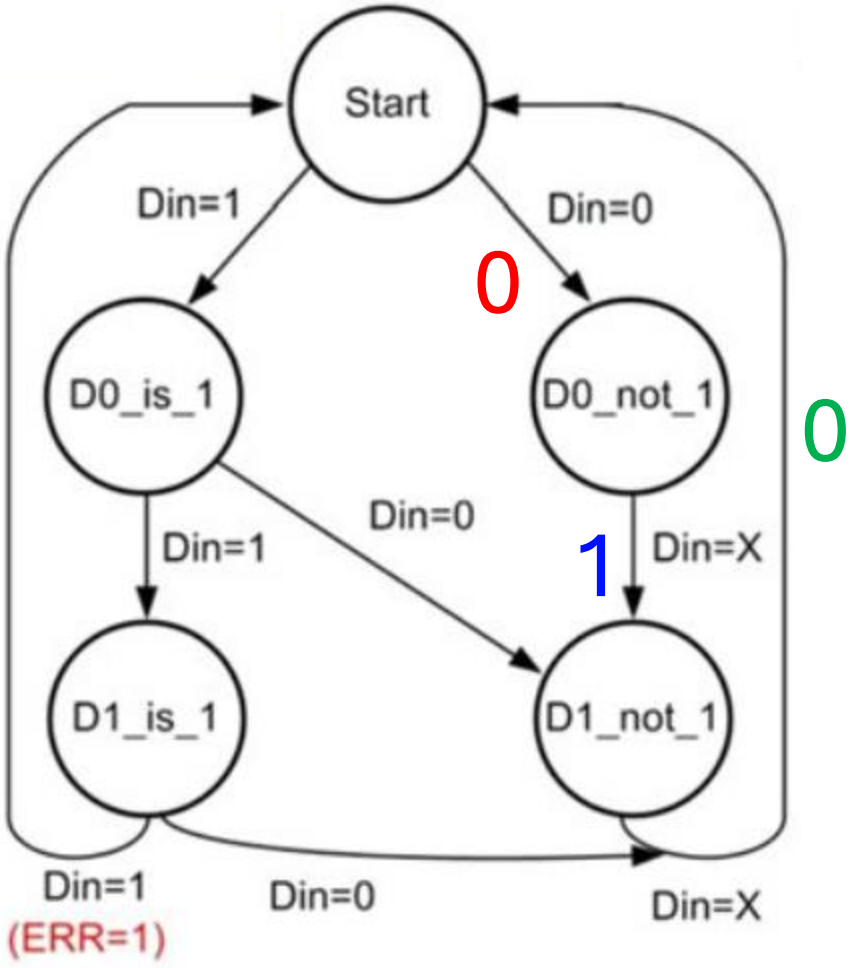
Diagramma di transizione tra gli stati (STD):



Descrizione di una FSM

Diagramma di transizione tra gli stati (STD):

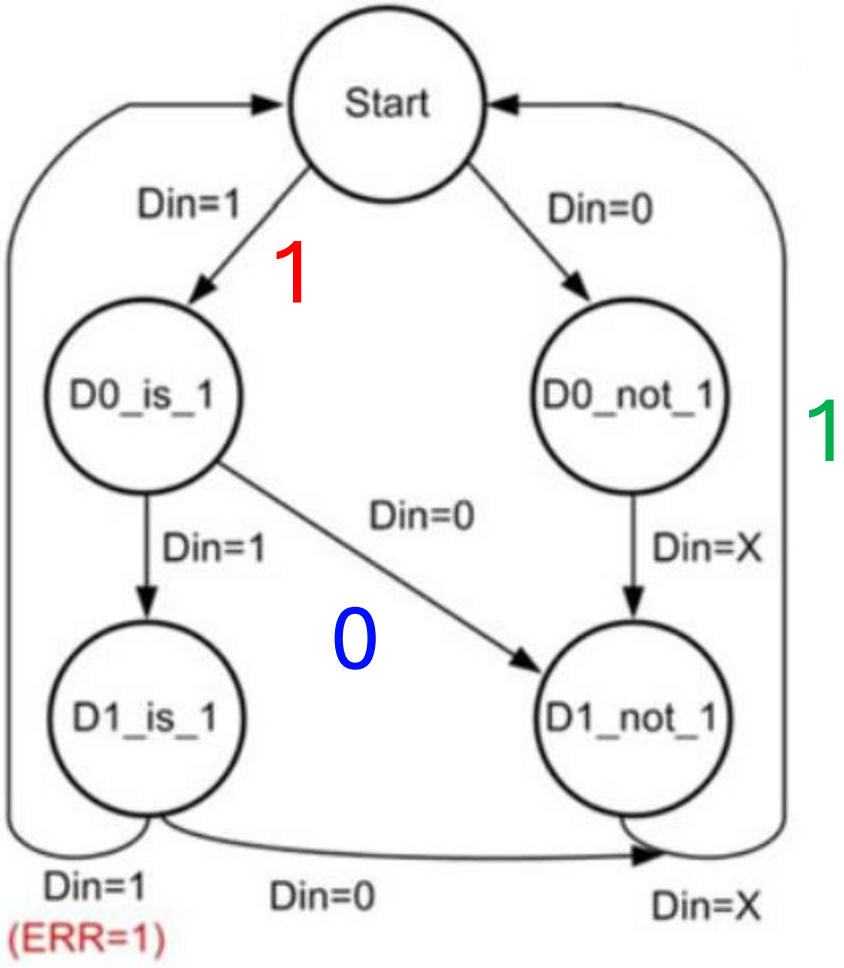
Din seq:
0 -> 1 -> 0



Descrizione di una FSM

Diagramma di transizione tra gli stati (STD):

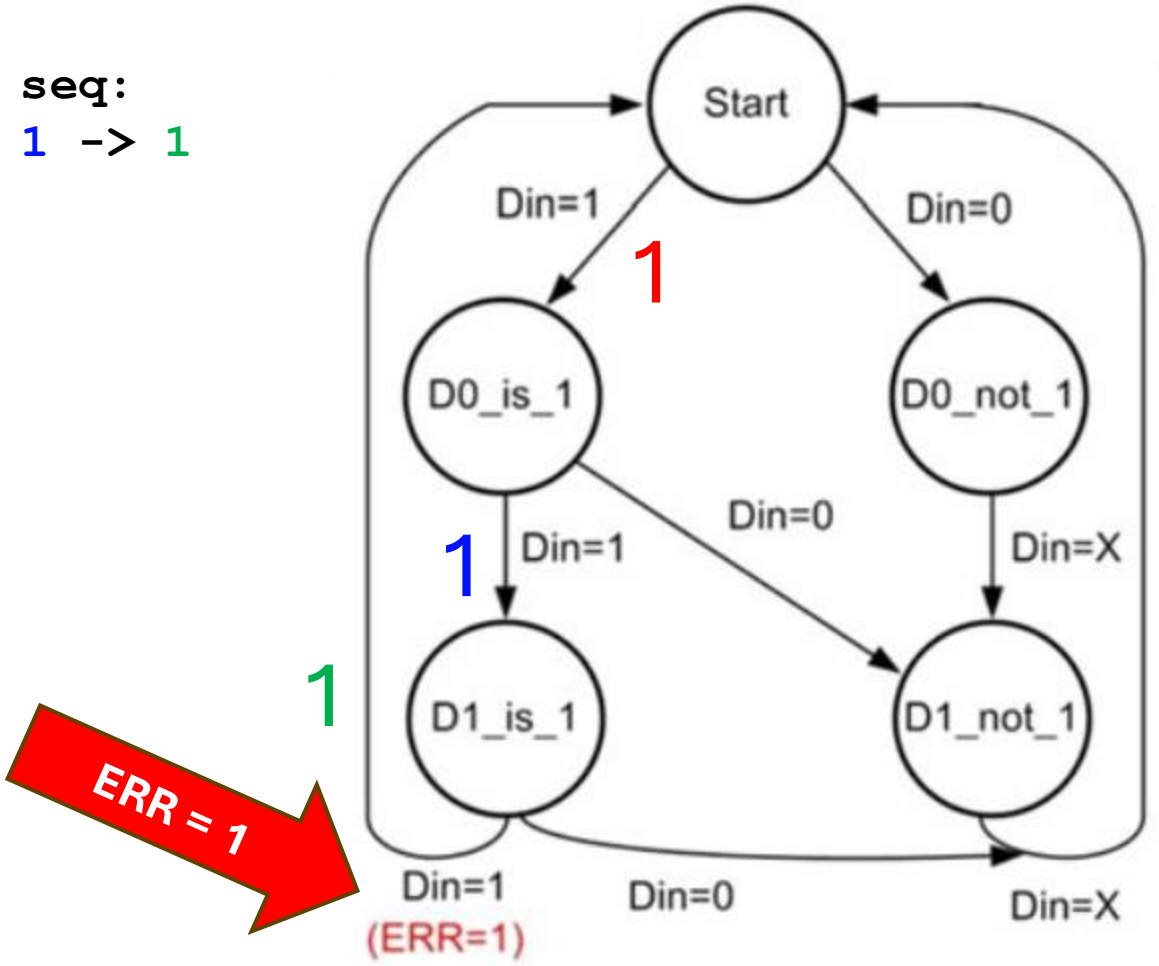
Din seq:
1-> 0 -> 1



Descrizione di una FSM

Diagramma di transizione tra gli stati (STD):

Din seq:
1-> 1 -> 1



STD definisce un percorso **CHIUSO** in cui la combinazione di stato **corrente** e **input** determina in modo **univoco** lo stato di **arrivo**.

Valore su linea di uscita dipende **unicamente** da stato corrente E input : **FSM Mealy.**

Descrizione di una FSM

Tabella di transizione tra gli stati (STT):

Stato corrente	**Din**	**Stato prossimo**	**ERR**
Start	0	D0_not_1	0
Start	1	D0_is_1	0
D0_is_1	0	D1_not_1	0
D0_is_1	1	D1_is_1	0
D0_not_1	X	D1_not_1	0
D1_not_1	X	Start	0
D1_is_1	0	Start	0
D1_is_1	1	Start	**1**

Descrizione di una FSM

STT codificata : procedendo in modalita' "carta e penna" il prossimo step sarebbe quello di assegnare ad ogni stato un codice (binario) che lo identifichi in modo univoco e sostituire questi codici identificativi al posto dei nomi degli stati nella STT.

Nel nostro esempio abbiamo 5 stati … servirebbero **almeno 3 bit** per poterli codificare.

Una possibile codifica sarebbe : start=000 , d0_not_1=001, d0_is_1=010, d1_not_1=011, d1_is_1=100 .

Ma in VHDL **non serve** una codifica … basta definire un tipo di dato ad hoc che possa assumere **solo** I valori degli stati in STG!

FSM in VHDL

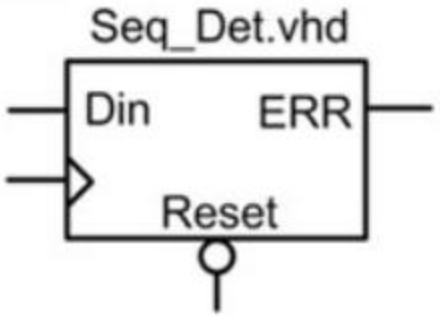
Le FSM possono essere descritte in modo semplice **utilizzando I processi**. Diversi approcci (un unico processo / piu' di un processo). Ognuno di essi ha aspetti pisitivi e negativi.

- **Multi processo:** maggior pulizia del codice VHDL. Maggior controllo (processi diversi possono avere sensitivity list diverse). Maggior pericolo di effetti inattesi dovuti a processi che non reagiscono ai segnali come atteso.
- **Unico processo:** Piu' semplice (tutti I segnali possono essere inseriti in un'unica sensitivity list). FSM complesse rchiedono processi molto lunghi da scrivere (gestione di tutti I singoli stati).

Descrizione di una FSM in VHDL

Entity :

```
library ieee;
use ieee.std_logic_1164.all;

entity seq_det is
        port(clk, reset : in std_logic;
             din : in std_logic;
             err : out std_logic);
end entity;
```



Descrizione di una FSM

Architecture : Questa e' piu' complessa. E' necessario ragionare per modellare il comportamento della FSM.

Abbiamo detto che una FSM puo' transire solo attraverso un insieme **finito** di stati. Anche I **percorsi** tra gli stati dipendono dalle connessioni nel diagramma delle transizioni tra gli stati (STD). I percorso **non sono** liberi … alcune transizioni tra stati **non** fanno parte di STD … quindi **non** possono verificarsi.

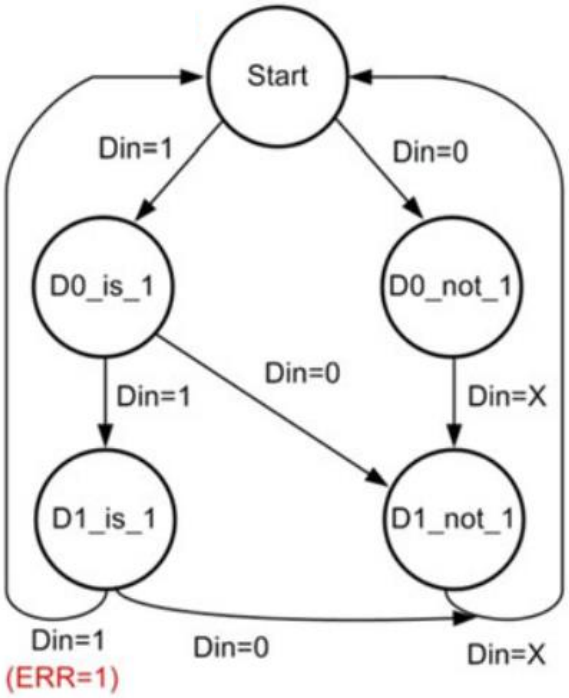
Per definire il comportamento di una FSM dobbiamo descrivere il suo STD!

Descrizione di una FSM

Architecture : Partiamo col definire l'insieme (finito) dei possibili stati. Per farlo definiamo un nuovo tipo di dato: **state** .

```
architecture arch of seq_det is
    type state is (start, d0_is_1, d0_not_1, d1_is_1, d1_not_1);
    signal current_state, next_state : state;
begin
```

Definiamo, inoltre, **due segnali** ... uno per lo stato **corrente** ed uno per lo stato **prossimo**. Il tipo di questi segnali sara' **state** (uno tra I possibili stati ammessi).



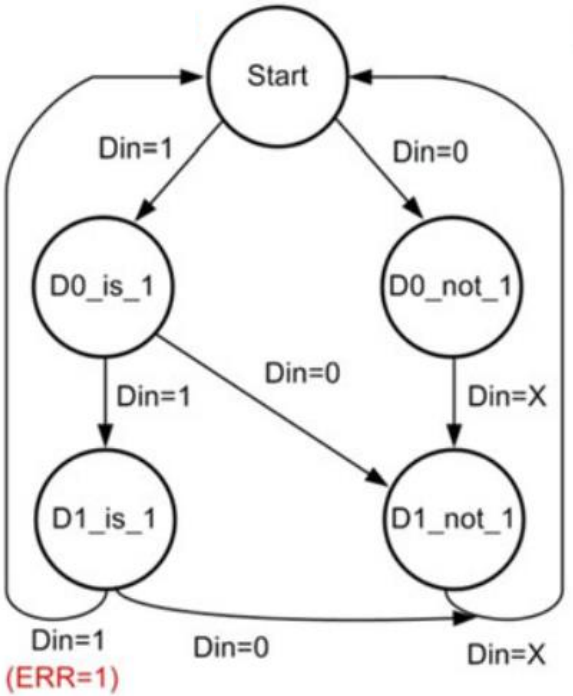
Descrizione di una FSM

Architecture : Partiamo col definire l'insieme (finito) dei possibili stati. Per farlo definiamo un nuovo tipo di dato: **state** .

```
architecture arch of seq_det is
    type state is (start, d0_is_1, d0_not_1, d1_is_1, d1_not_1);
    signal current_state, next_state : state;
begin
```

Abbiamo definito la collezione di stati (tipo state) ed I segnali per modellare una logica che imponga che qualsiasi transizione tra stato corrente e stato prossimo sia **confinata alla collezione di stati ammessi**.

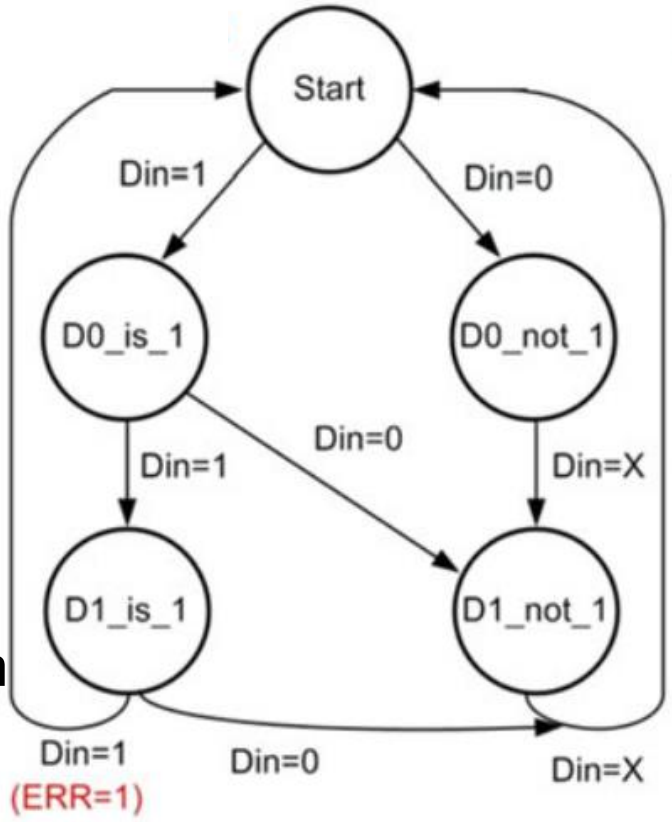
Resta da capire come imporre limitazioni anche al set delle transizioni possibili.



Descrizione di una FSM

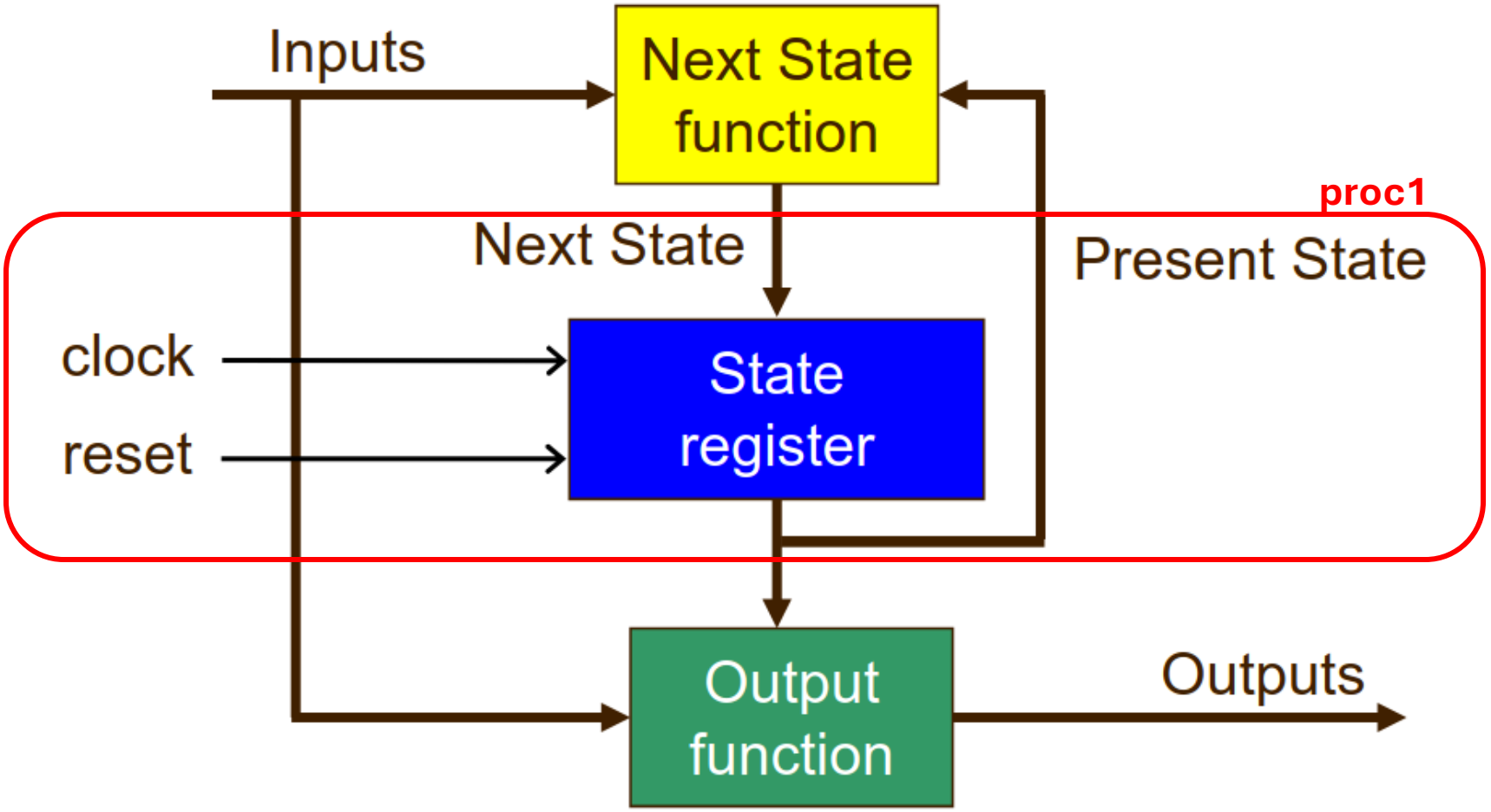
Architecture :

Dobbiamo garantire a FSM la possibilita' di transire da uno stato all'altro in corrispondenza di ogni **fronte di salita** del clock. Questo **non ha** a che fare con una specifica transizione!



Vogliamo solo descrivere un sistema (la FSM) che in presenza di segnale **reset** torna ad uno **stato iniziale** e in presenza di un fronte di **salita** del clock passa **da stato corrente a stato prossimo**.

FSM : finite states machine

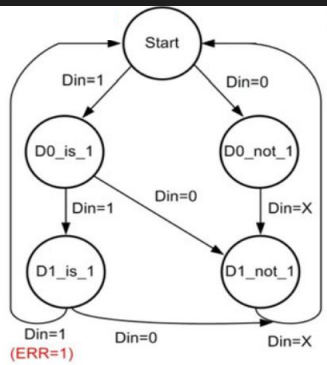


Descrizione di una FSM

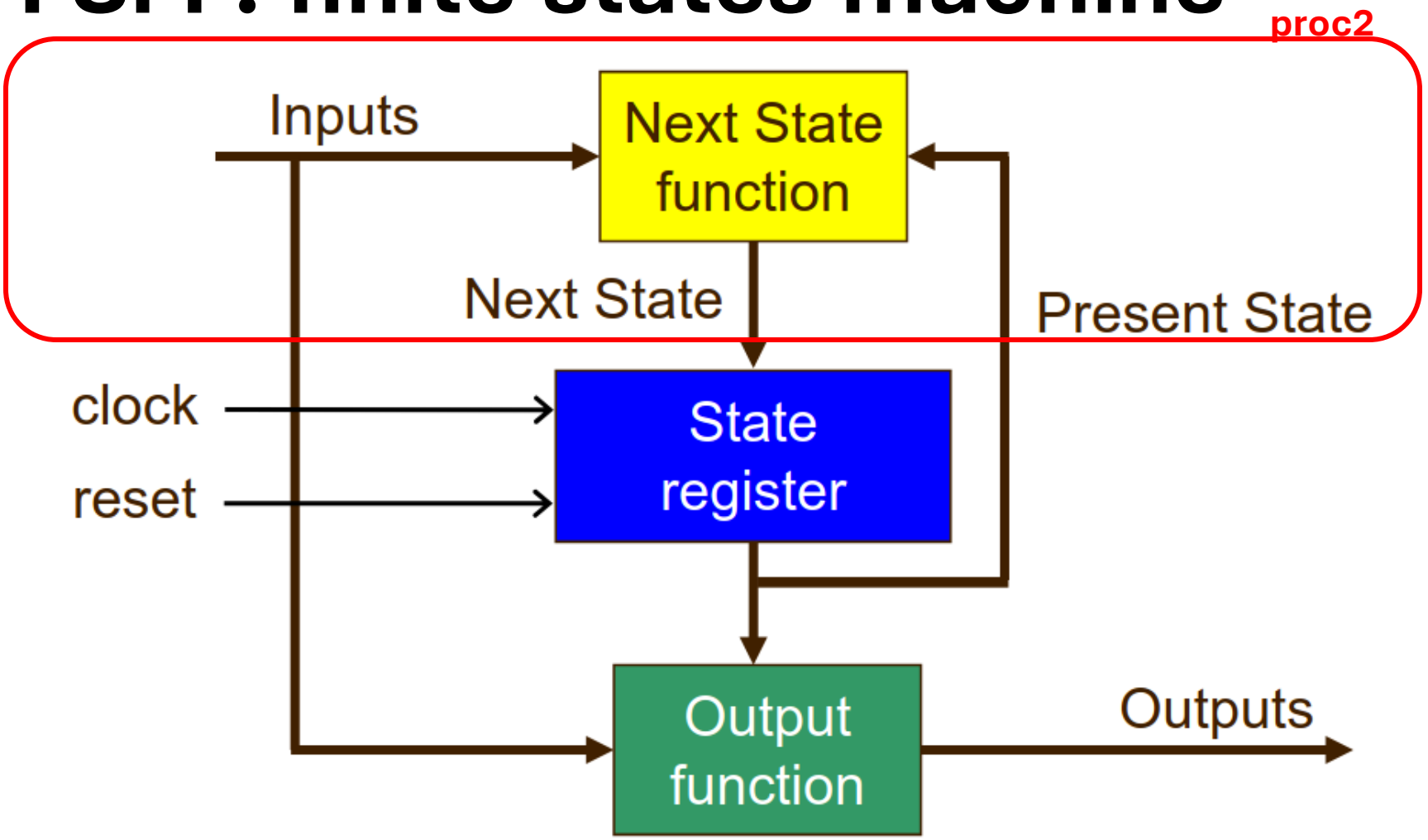
Architecture :

```
-- PROCESS 1 : state memory :
        state_memory : process(clk, reset)
        begin
                if(reset = '0') then
                        current_state <= start;
                elsif(clk'event and clk='1') then
                        current_state <= next_state;
                end if;
        end process;
```

Ora serve un processo (separato) che definisca le possibili transizioni (frecce) in STD.



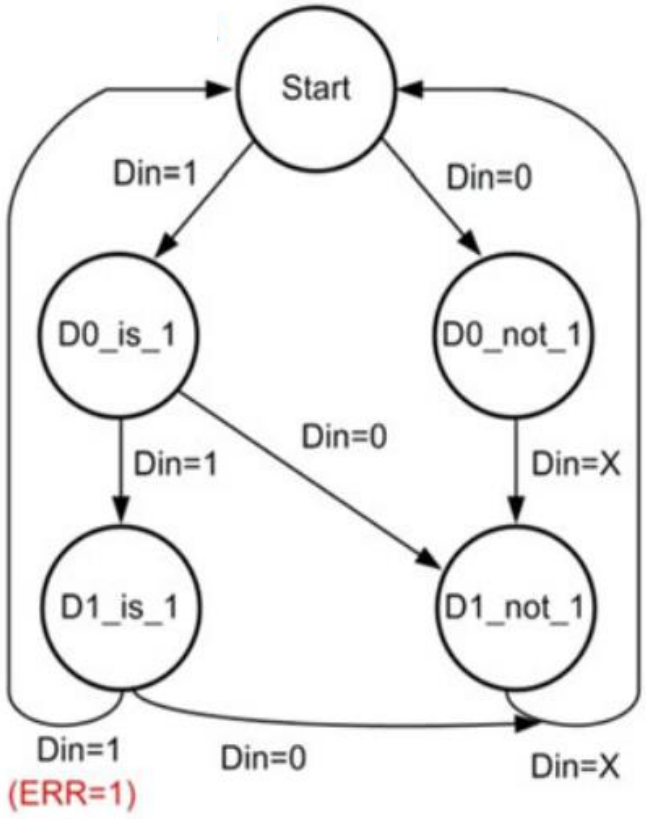
FSM : finite states machine



Descrizione di una FSM

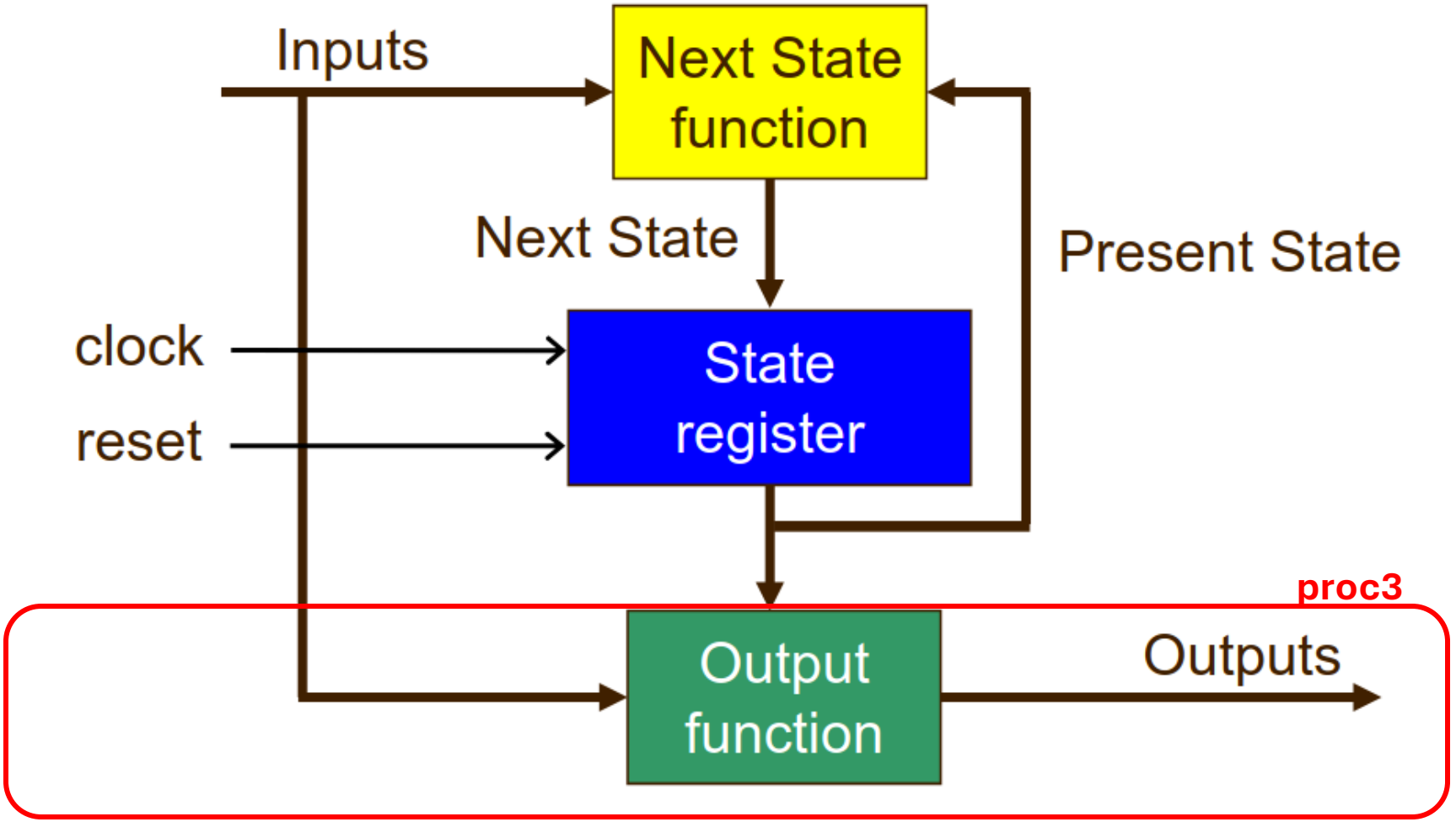
Architecture :

```
-- PROCESS 2 : next state logic :
        next_state_logic : process(current_state, din) is
        begin
                case(current_state) is
                        when start => if(din = '1') then
                                next_state <= d0_is_1;
                        else
                                next_state <= d0_not_1;
                        end if;
                        when d0_is_1 => if(din = '1') then
                                next_state <= d1_is_1;
                        else
                                next_state <= d1_not_1;
                        end if;
                        when d1_is_1 => next_state  <= start;
                        when d0_not_1 => next_state <= d1_not_1;
                        when d1_not_1 => next_state <= start;
                        when others => next_state <= start;
                end case;
        end process;
```



Manca solo la definizione della logica di output! Per essa avremo un processo separato....

FSM : finite states machine



Descrizione di una FSM

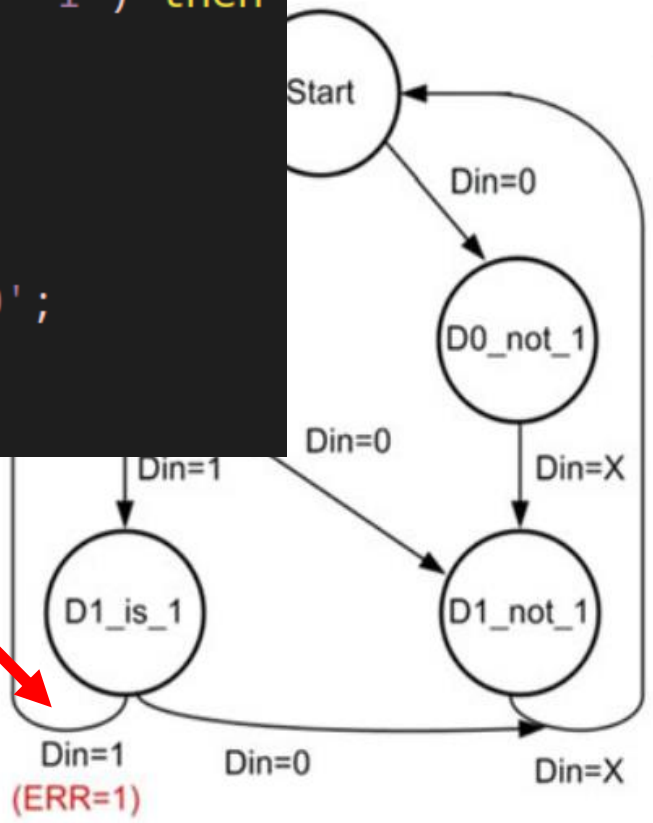
Architecture :

```
-- PROCESS 3 : OUTPUT LOGIC
        output_process : process(current_state, din) is
        begin
                case(current_state) is
                        when d1_is_1 => if(din = '1') then
                                        ERR <= '1';
                        else
                                        ERR <= '0';
                        end if;
                        when others => ERR <= '0';
                end case;
        end process;
```

Questo conclude la descrizone VHDL della FSM.

Ricordatevi di inserire

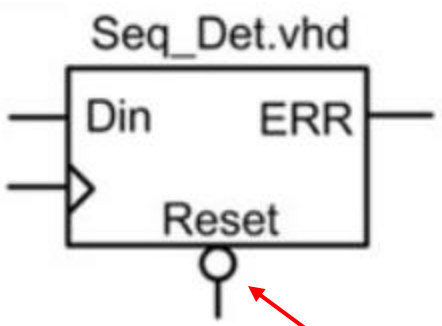
end architecture;



Testbench FSM

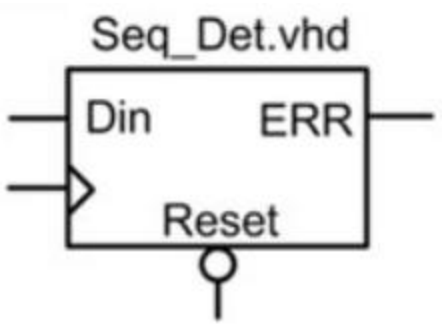
Elementi <u>necessari</u> :

- Processo di **clock**
- Processo stimoli su **segnale/i in input** e su segnale di reset asincrono.



Testbench FSM

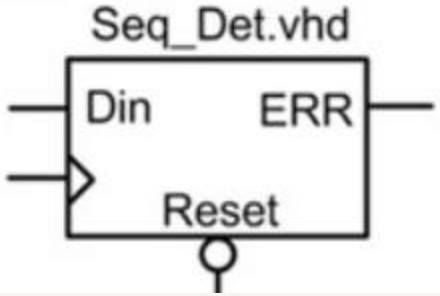
Mealy FSM sequence detector




```
process is
begin
        clk <= not clk;
        wait for 5 ns;
end process;

stimulus: process is
begin
 -- write your test here
       reset <= '0'; -- reset the FST in start state
       wait for 5 ns;
       reset <= '1';
       wait  for 5 ns;
            -- triplet 1
       din <= '1';
       wait for 10 ns;
       din <= '0';
       wait for 10 ns;
       din <= '0';
       wait for 10 ns;
            -- triplet 2
       din <= '1';
       wait for 10 ns;
       din <= '1';
       wait for 10 ns;
       din <= '1';
       wait for 10 ns;
            -- triplet 3 (error signal triplet + reset)
       din <= '1';
       wait for 10 ns;
       din <= '1';
       wait for 10 ns;
       reset <= '0';
       din <= '1';
       wait for 10 ns;
            -- single bit
       reset <= '1';
       din <= '0';
       wait for 10 ns;
            -- tot simulation time : 190 ns
wait;
end process;
```

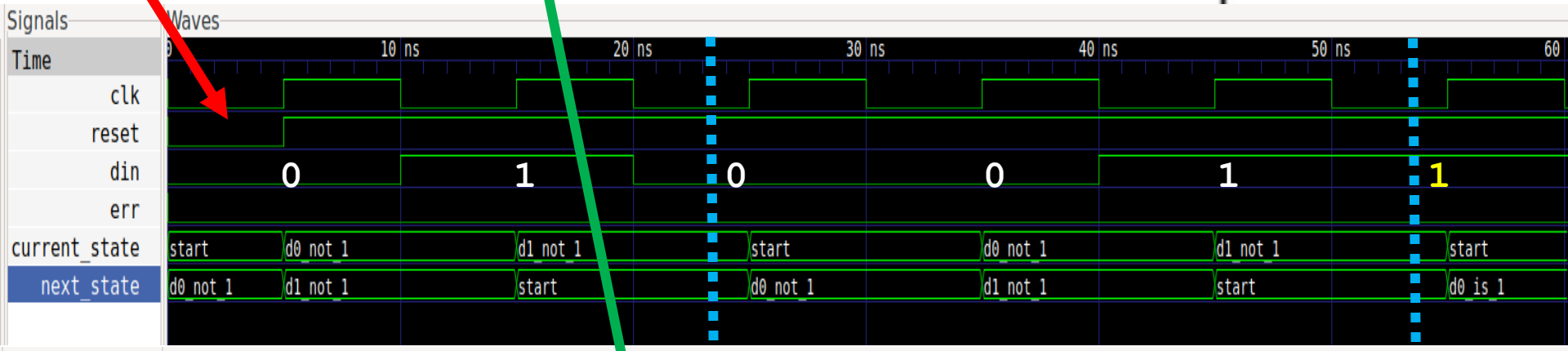
Testbench FSM

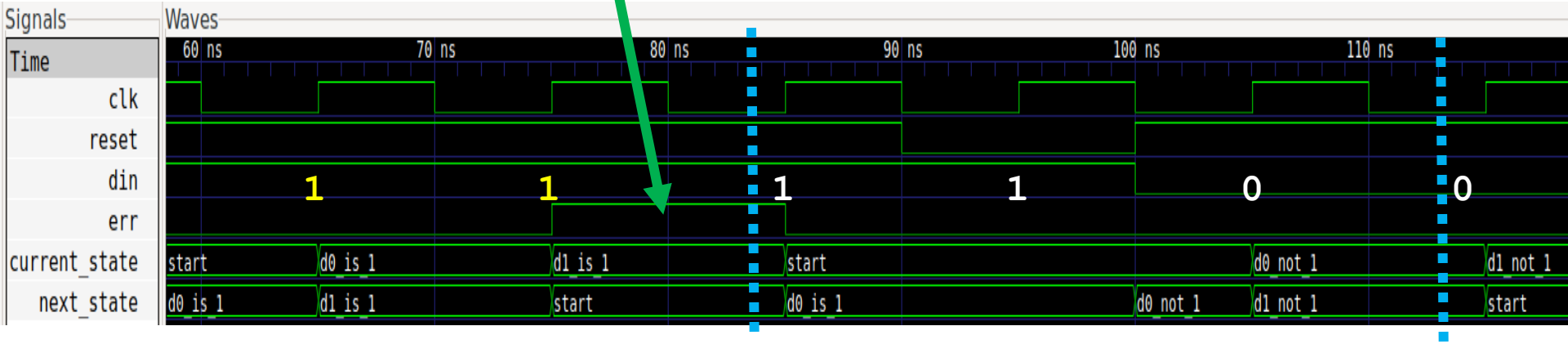
Mealy FSM sequence detector



ERR attivato!

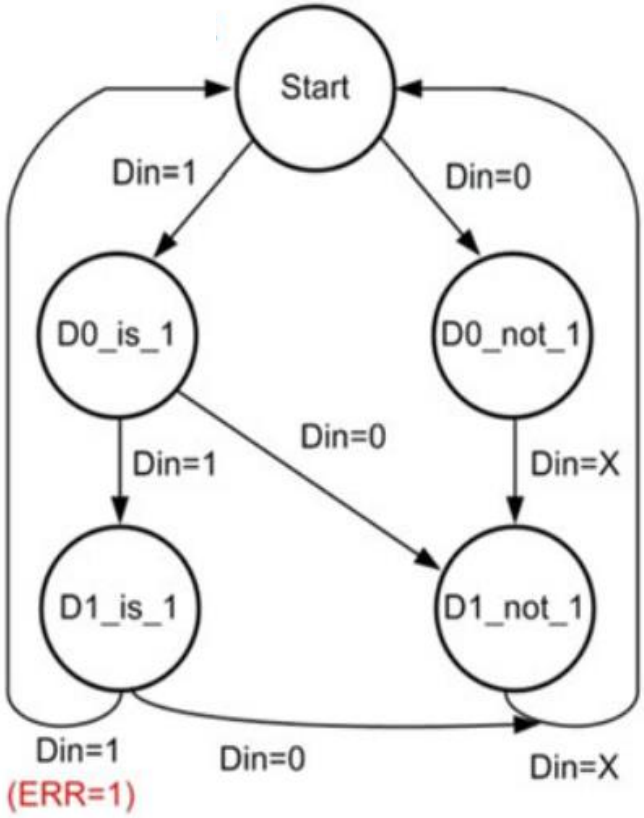
reset



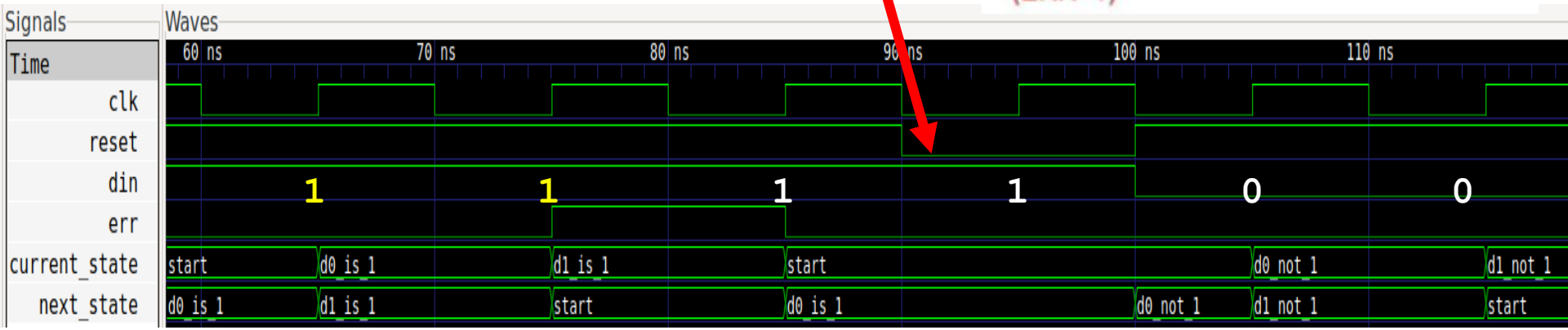


Testbench FSM

Mealy FSM sequence detector

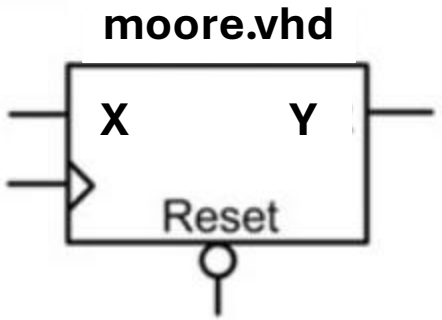


reset

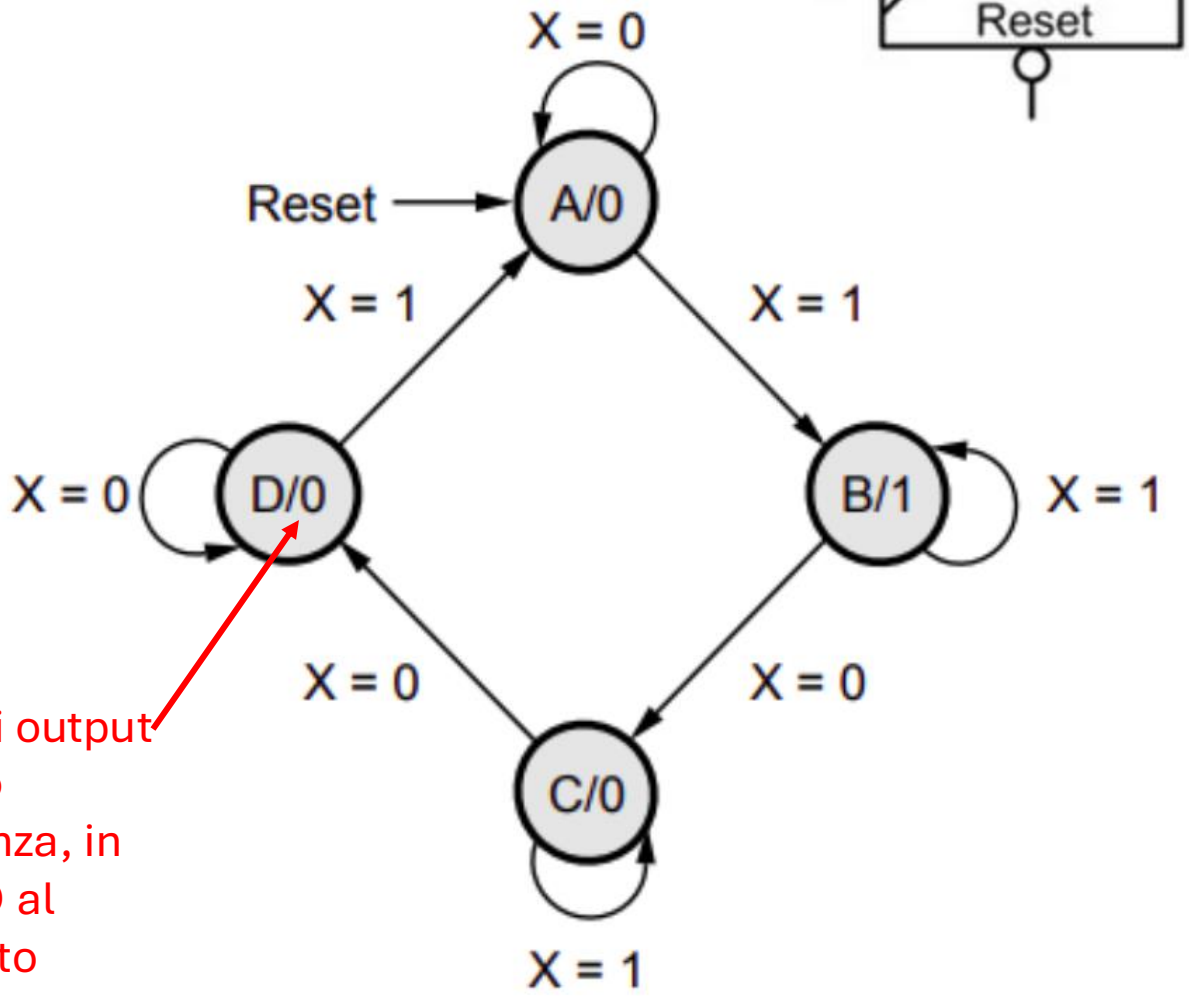


FSM Moore – es. 2

X = {0,1} Y = {0,1}



STG:

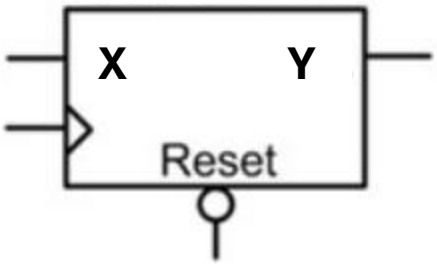


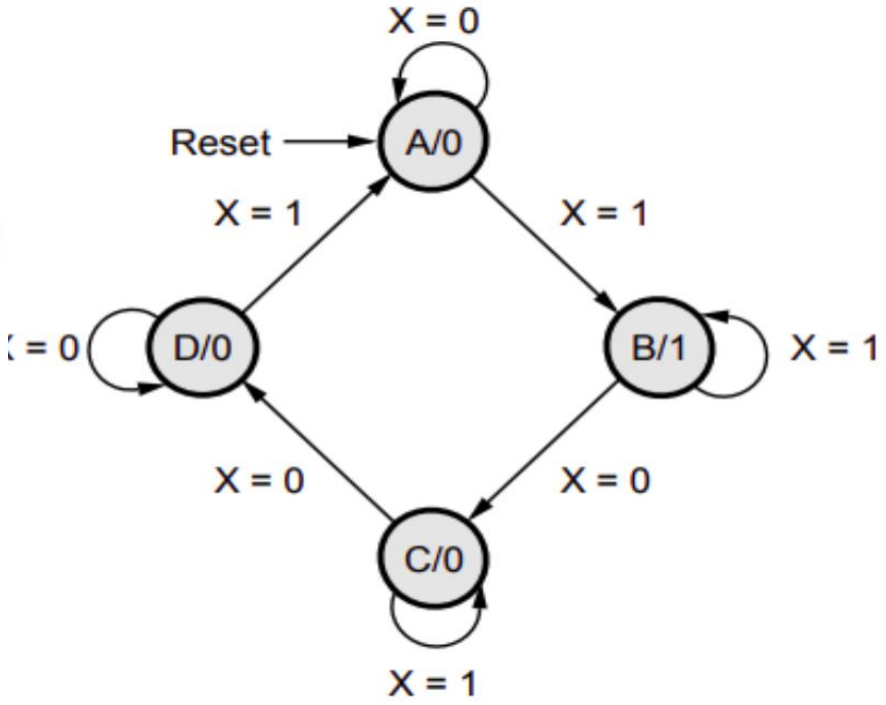
FSM **Moore** … il valore di output dipende **solo** dallo **stato corrente** e, di conseguenza, in STG e' riportato **DENTRO** al cerchio che indica lo stato (insieme al nome dello stato)

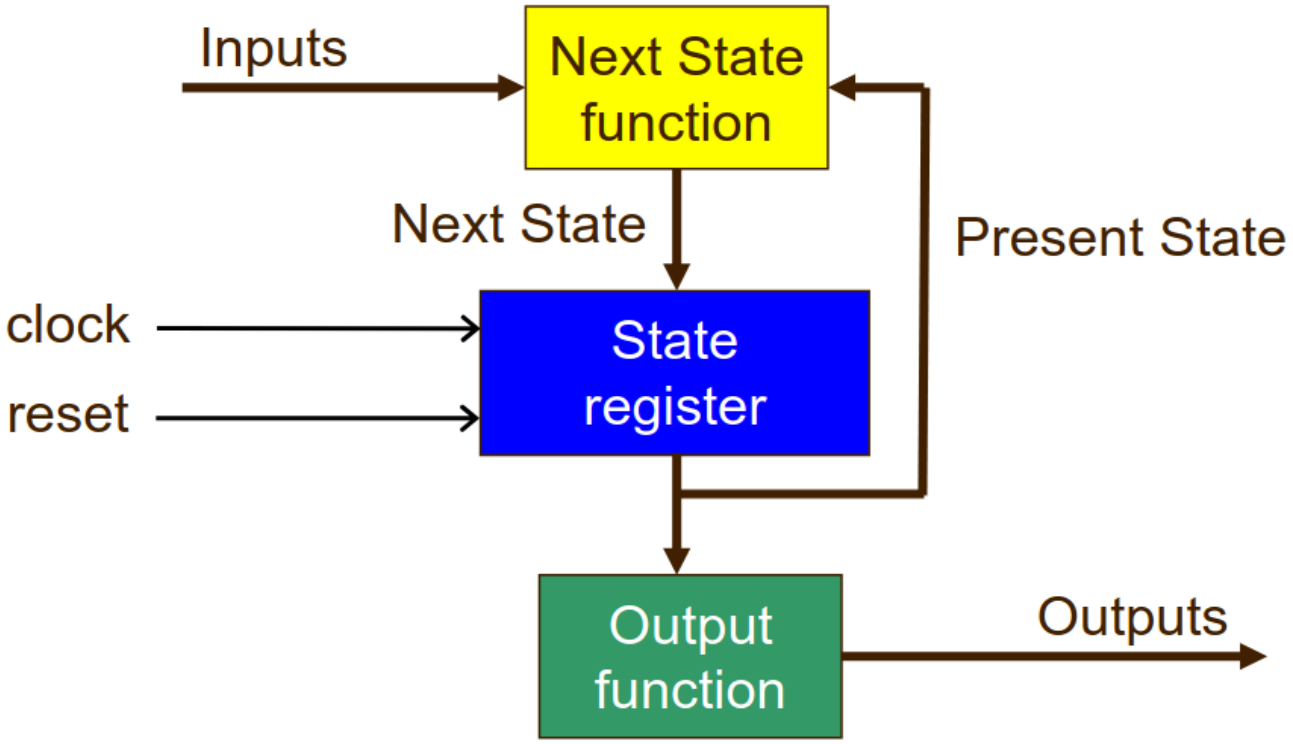
FSM Moore

X = {0,1} Y = {0,1}

moore.vhd





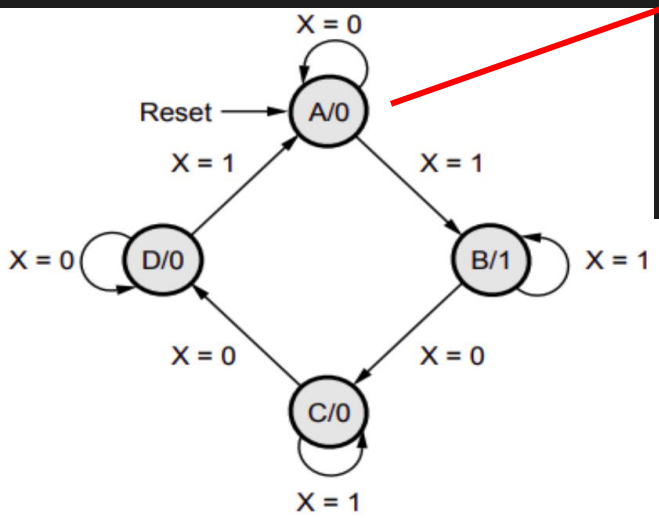


FSM Moore

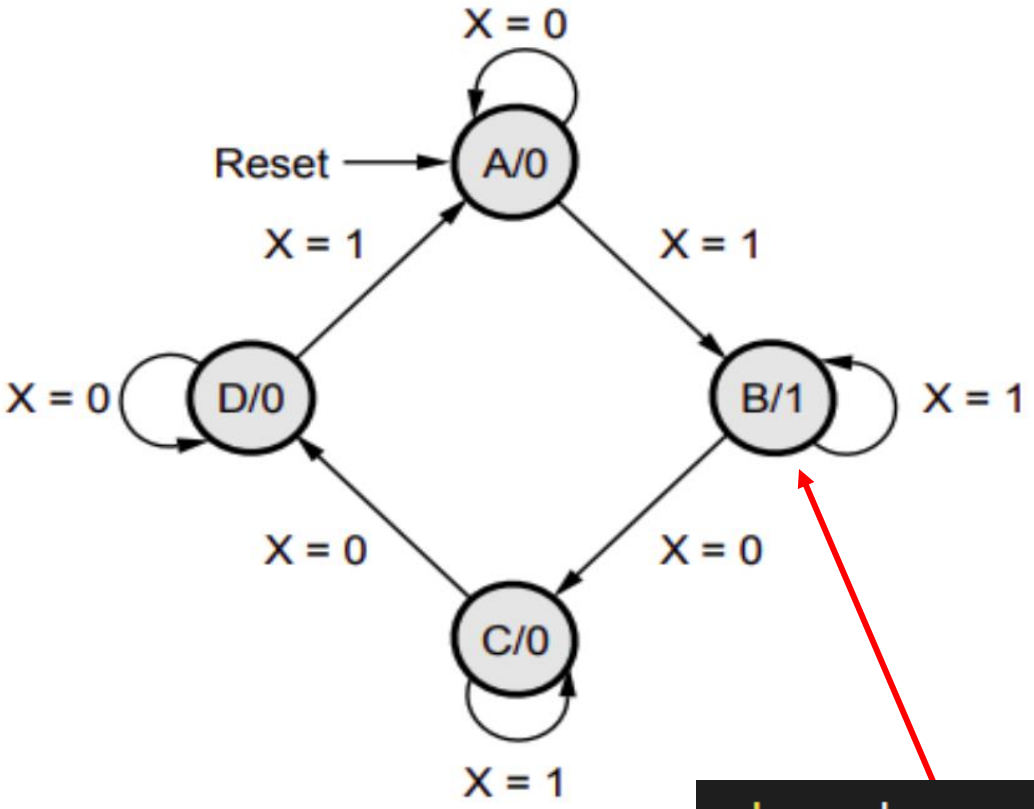
```
library ieee;
use ieee.std_logic_1164.all;

entity moore is
        port(   clk, reset, x : in std_logic;
                y : out std_logic);
end entity;

architecture arch of moore is
        type state is (a,b,c,d);
        -- abbiamo 1 solo segnale di stato.
        signal z : state;
begin
```

FSM Moore

```
begin
        -- cambiandone il valore definiamo il next_state
        process (reset, clk)
        -- un unico processo che gestisce reset/avanzamento stato
        begin
                if reset = '0' then z <= a;
                elsif(clk'event and clk = '1') then
                        case z is
                        -- NB: In process (seq.) .NON si usa with/select ma case!
                        -- per OGNI stato modelliamo le transizioni
                        when a => if(x='0') then -- STATO a
                                        z <= a; -- next_state sara' a se x=0
                                else
                                        z <= b; -- next_state sara' b se x!=0
                                end if;
```

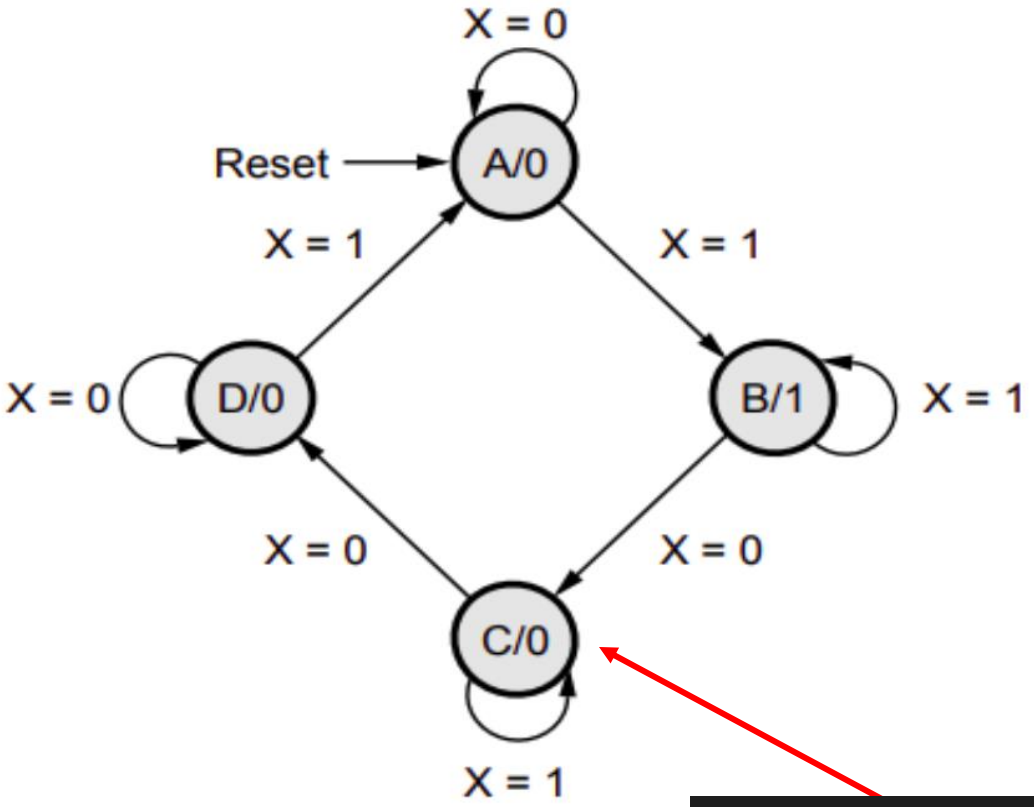


FSM Moore



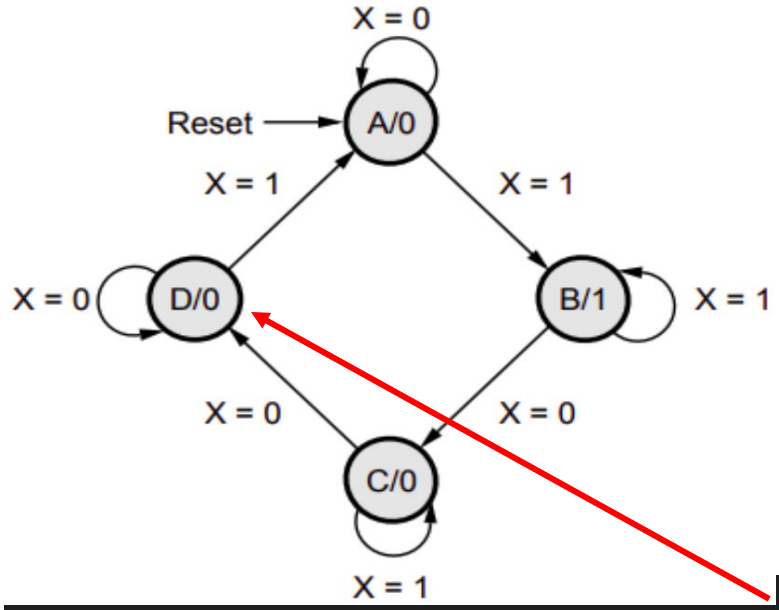

```
when b => if(x ='0') then -- STATO b
                    z <= c;
          else
                    z <= b;
          end if;
```

FSM Moore




```
when c => if(x='0') then -- STATO c
                  z <= d;
          else
                  z <= c;
          end if;
```

FSM Moore

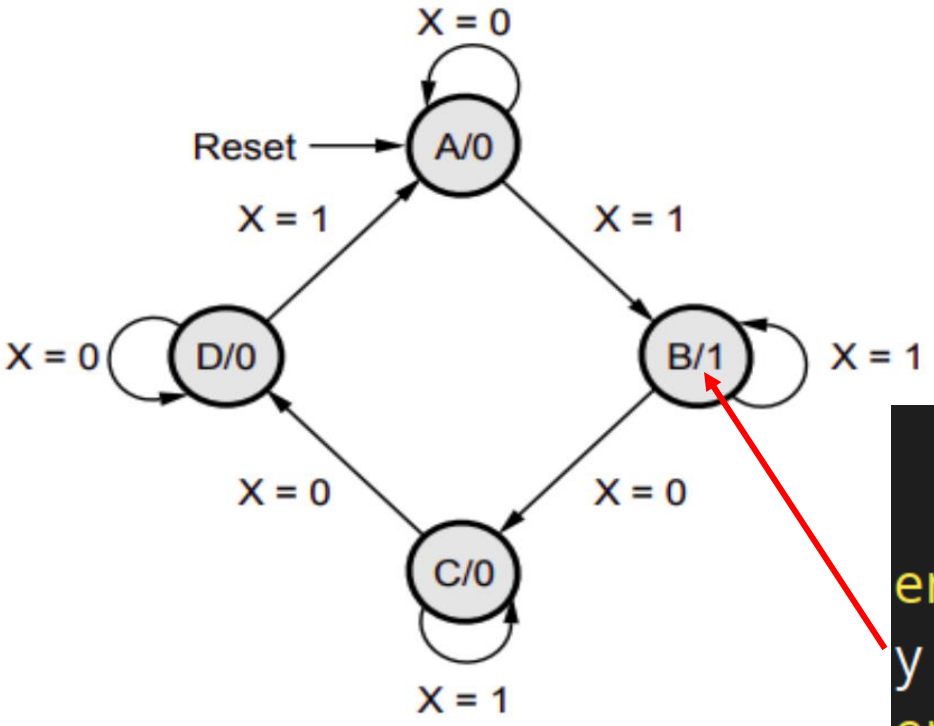



```
                when d => if(x='0') then -- STATO d
                                z <= d;
                          else
                                z <= a;
                          end if;
            end case;
            end if;
end process;
```

FSM Moore

Resta da definire il calcolo dell'**uscita**:

- Questo esempio e' una FSM Moore
- Uscita dipende **unicamente** dallo stato corrente!

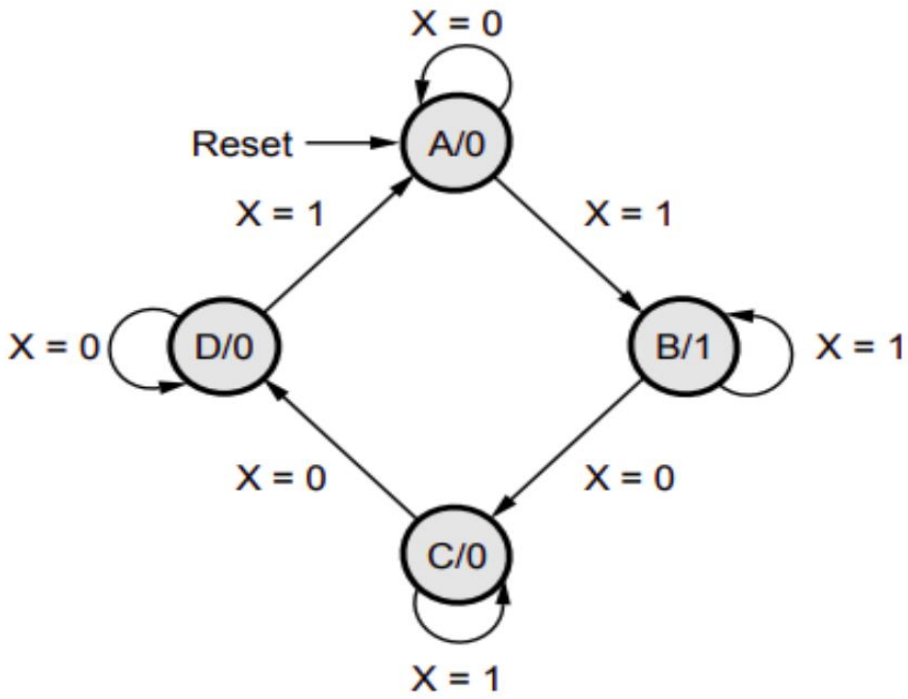


out of process

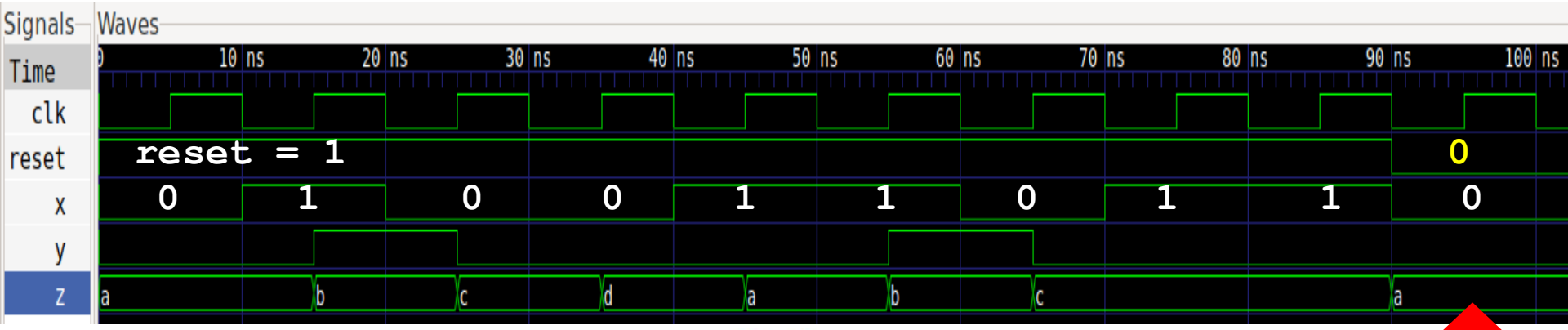
```
                    end case;
                    end if;
end process;
y <= '1' when z=b else '0';
end architecture;
```

FSM Moore - testbench

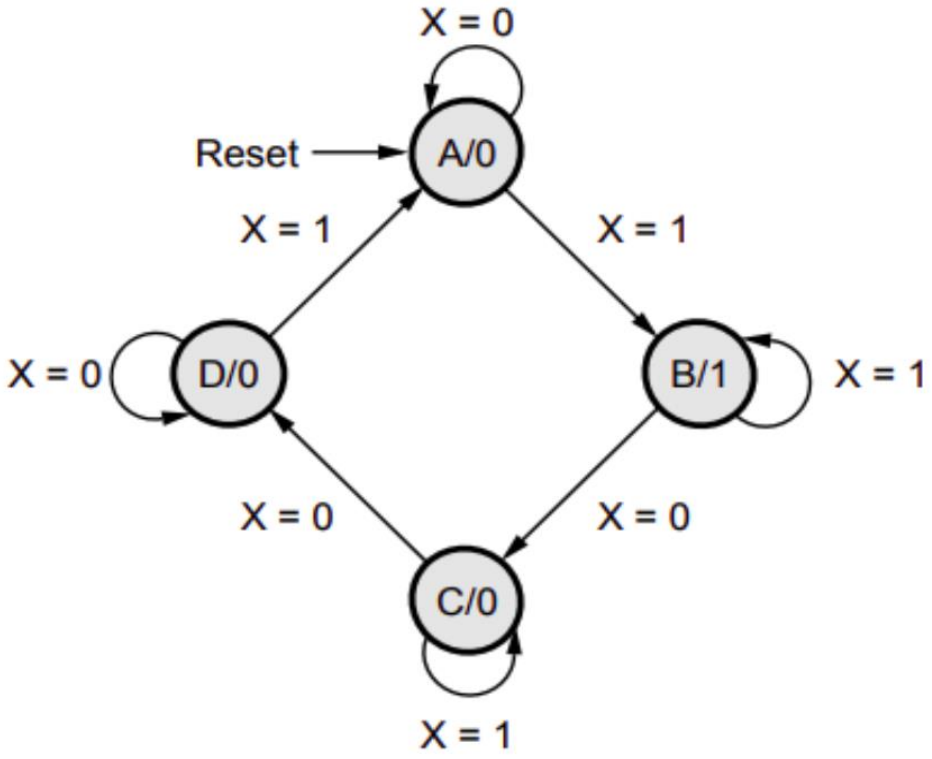
```
-- write your test here
        x <= '0'; -- reset initialized to '1'
        wait for 10 ns;
        x <= '1';
        wait for 10 ns;
        x <= '0';
        wait for 10 ns;
        x <= '0';
        wait for 10 ns;
        x <= '1';
        wait for 10 ns;
        x <= '1';
        wait for 10 ns;
        x <= '0';
        wait for 10 ns;
        x <= '1';
        wait for 10 ns;
        x <= '1';
        wait for 10 ns;
        reset <= '0';
        x <= '0';
        wait for 10 ns;
wait;
```



FSM Moore - testbench



FSM Moore meno "**reattiva**" di FSM Mealy ... per reagire ha bisogno diosservare un fronte di salita in piu' rispetto a FSM Mealy!



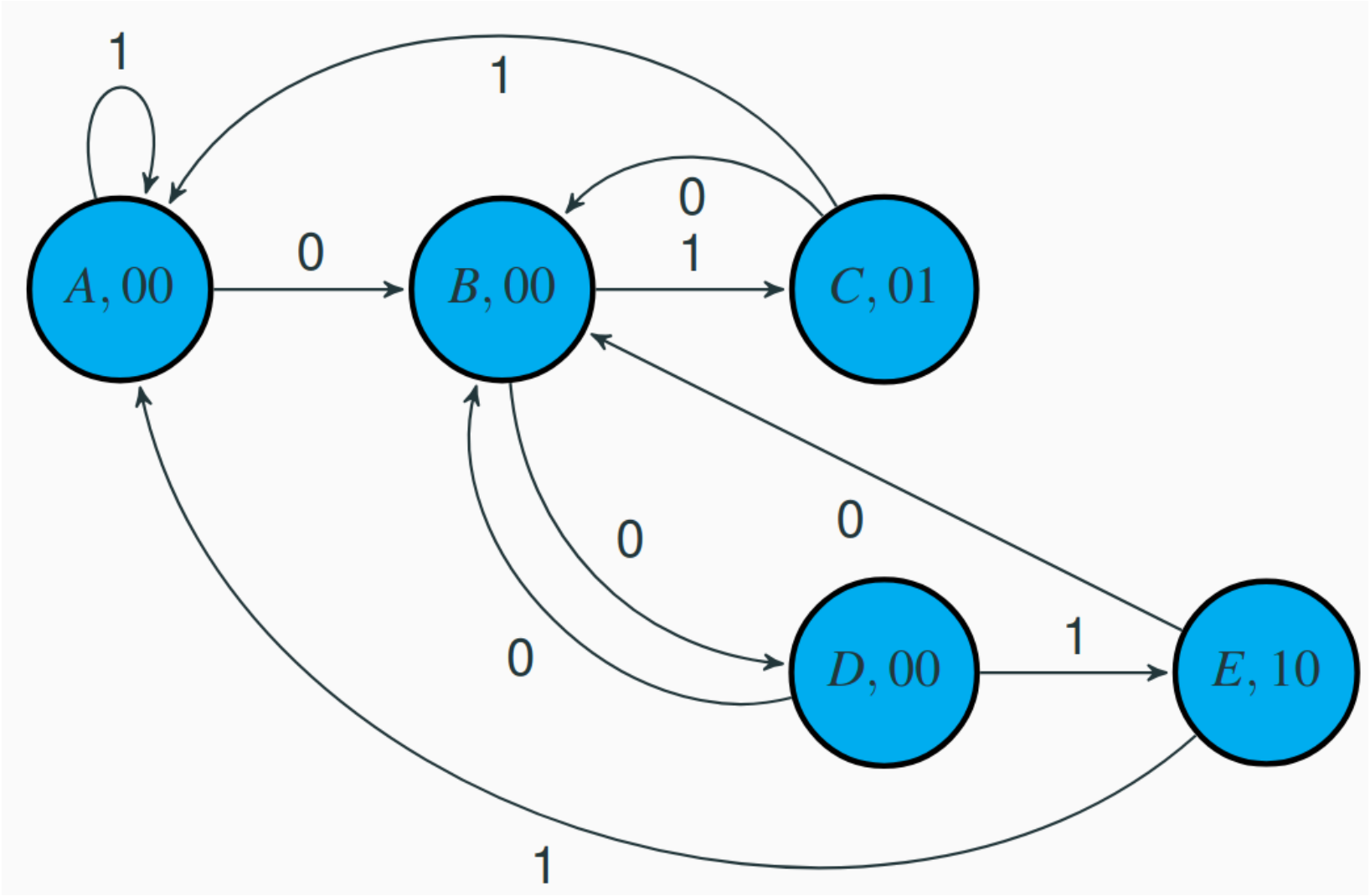
Reset in stato a

FSM - homework

Descrivere, analizzare e simulare in VHDL una FSM **Moore** che osservi una linea seriale di input x di ampiezza 1 bit e che emetta, su una linea di output, 01 se ha osservato in ingresso la sequenza 01, 10 se ha osservato in ingresso la sequenza 001 ed emetta 00 altrimenti.

Nella prossima slide verra' fornito STG della FSM da descrivere.

FSM - homework



FSM - homework : VINCOLI ESERCIZIO

- **Non** potete utilizzare il tipo **std_logic_vector** (l'uscita e' espressa dai valori **y** e **w** (in quest'ordine).
- Le sequenze da riconoscere (oltre ad avere diversa lunghezza) **NON** sono sovrapponibili (001 **non** puo' generare entrambi gli output).
- Descrizione SM di tipo **multi segment**.
- 1 processo che gestisca un segnale di reset e attua l'avanzamento da stato corrente a stato prossimo in presenza di un fronte di salita del clock.
- 1 processo che gestisca le transizioni tra stati.
- 1 processo che gestisca I valori di output sulla base del solo stato corrente (Moore).
- Se il segnale di reset (non mostrato in STG) e' alto FSM torna in stato A