Struttura file .vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
```

link a librerie e package

```
entity nome_del_modulo  is
port ( term1,term2 : in std_logic;
       term3,term4 : out std_logic) ;
end nome_del_modulo;
```

entity
interfaccia I/O del modulo

```
architecture tipo_architettura  of nome_del_modulo is
begin
  .......
end tipo_archiettura;
```

architecture
descrizione del circuito (comportamentale o strutturale)

Architecture

```
architecture tipo_architettura  of nome_del_modulo is

   [declarative_part]    definizione di tipi locali, segnali interni, componenti
begin
  [statement_part]       gli statements sono eseguiti in modo concorrente (parallelo)
                         rappresentano infatti istanze circuitali
end tipo_architettura;
```

Parte dichiarativa: Tipi

- segnali/porte: std_logic, bit
  std_logic_vector
  signed/unsigned
- variabili di stato: enumeration type
  (cioè fornito attraverso la lista dei possibili valori)

Structural description of a Full-Adder

```
library IEEE;
use IEEE.std_logic_1164.all;

entity FA is
port ( A,B,CIN : in std_logic;
       S, COUT : out std_logic);
end FA;

architecture STRUCTURAL of FA is
  component HA                          dichiarazione del componente HA
  port ( I1,I2   : in std_logic;
         SUM, CO : out std_logic);
  end component;
signal S1, C1, C2 : std_logic;          definizione di segnali interni

begin
  ha1 : HA port map( I1 => B, I2 => CIN, SUM => S1, CO =>C1); istanzia i moduli
ha2 : HA port map( I1 => A, I2 => S1, SUM => S, CO => C2);

   COUT <= C2 xor C1; assegnamenti

end STRUCTURAL;
```

Assegnamento <=

- **costrutto che rappresenta una istanza circuitale**
- **è eseguito ogni volta che il segnale nell'espressione a destra cambia valore (evento)**

es:

s <= a and b; s,a,b std_logic

z <= not s; z,s std_logic

cont <= a + b; cont,a,b unsigned/signed rapprensentati con lo stesso numero di bit unsigned(n downto 0)

cont <= a + conv_unsigned(1, 16); cont, a unsigned a 16 bit

- **VHDL non è case sensitive;**
- **segnali e variabili non possono incominciare con una cifra (10sec NON è un nome valido)**
- **non usare caratteri non alfanumerici**

Assegnamenti

```
five_seconds  <= '1' when current_count = conv_unsigned(2,16) else '0';


next_cicli <= conv_unsigned(0,5) when cicli = conv_unsigned(8,5) else cicli +
    conv_unsigned(1,5);

uscita(0) <= '0' ;
uscita(4 downto 1)  <=  Ingresso;   ingresso a 4 bit; uscita = ingresso * 2 (5 bit)
```

VHDL : struttura gerarchica

È possibile istanziare componenti circuitali nell'architecture purchè la loro interfaccia di I/O sia descritta (con il costrutto component) nella parte dichiarativa

```
architecture STRUCTURAL of FA is
  component HA                          dichiarazione del componente HA
  port ( I1,I2   : in std_logic;
         SUM, CO : out std_logic);
  end component;
signal S1, C1, C2 : std_logic;          definzione di segnali interni

begin
  ha1 : HA port map( I1 => B, I2 => CIN, SUM => S1, CO =>C1); istanze di componenti
  ha2 : HA port map( I1 => A, I2 => S1, SUM => S, CO => C2);

   COUT <= C2 xor C1;

end STRUCTURAL;
```


VHDL : Processi

- un processo rappresenta uno "statement" espanso, cioè una operazione non elementare composta da un insieme di operazioni elementari; i processi sono eseguiti in parallelo agli altri statements (processi , assegnament , istanze di blocchi)
- All'interno del processo l'esecuzione è sequenziale e possono essere definite delle variabili
- solo all'interno di processi possono essere inserite espressioni di controllo condizionali (IF, CASE, ..)
- l'uscita di un processo viene ricalcolata ogni volta che si ha un evento (cambiamento di valore) su uno dei segnali appartenenti alla ***sensitivity list***
- In VHDL sintetizzabile (cioè descrizione RTL), tutti gli ingressi del processo devono appartenere alla ***sensitivity list***
- I processi sono necessari per descrivere una rete sequenziale

Processi

```
process (sensitivity list)
   [declarative_part]

begin
   [statements]
end  process;
```


Esempio 1:

```
architecture behavior1  of MUX
   begin
     process(sel, A,B)
       begin
         case sel is
            when “00” => Y <= A;
             when “01” => Y <= B;
             when others => Y <= ‘0’;
         end case;
      end process;
   end behavior1;
```

imp: i costrutti condizionali (case, if) devono assegnare un valore per tutte le configurazioni possibili

Esempio 2:

```
architecture behavior2  of MUX
   begin
     process(sel, A,B)

   Y <= '0';   --- valore  assegnato a Y se non esplicitamente
                    -- definito
begin
            if  (sel = "00") then
                    Y <= A;
             elsif  (sel = "01") then
                     Y <= B;
            end if;
       end process;
    end behavior2;
```

imp: i costrutti condizionali (case, if) devono assegnare un valore per tutte le configurazioni possibili

Regole

1. Un segnale non può essere utilizzato due volte come destinazione (fuorchè all'interno di un processo) *(la violazione comporterebbe un conflitto elettrico)*
2. I costrutti if/case possono essere utilizzati solo all'interno di processi e deve essere definito il valore delle uscite per tutte le configurazioni possibili *(la violazione comporterebbe l'inserimento di latch parassiti)*
3. Una porta di uscita non può essere usata come ingresso da statement/processi o blocchi gerarchici
4. L'esecuzione degli statement/processi o blocchi gerarchici è sempre concorrente (*non ha alcuna importanza l'ordine con cui sono scritti, rappresentano infatti istanze di componenti circuitali)*

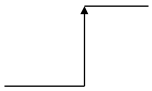
Attributi associati a un segnale

Funzioni di simulazione legate al comportamento di segnali

- signal'event — vero se c'è un evento sul segnale (cioè una transizione)

Es:

CLK'event and CLK='1



Register description using VHDL

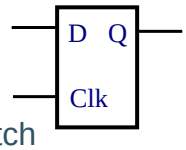
```
library IEEE;
  use IEEE.std_logic_1164.all;
--Flip Flop con reset asincrono
entity FF is
port (CLK, RESET, D : in std_logic;
                Q          : out std_logic
                );
end entity;

architecture behavioral of FF is
begin
 process(CLK, reset)
   begin
     if RESET='1' then
                Q <= '0';
              elsif CLK'event and CLK='1' then
                Q <= D;
    end if;
 end process;
end behavioral;
```

Asynchronous reset

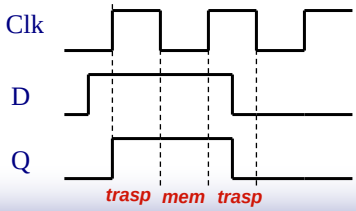
In corrispondenda della transizione L-> H del segnale CLK viene aggiornato Q con il valore attualmente presente all'ingresso D

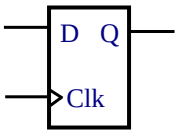
Latch versus Register

- Latch
  stores data when clock is low or high
- Register
  stores data when clock rises (or falls)

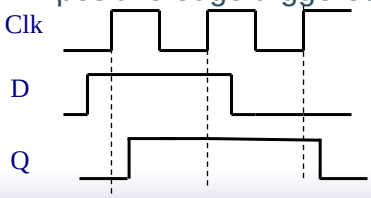


Ex: positive latch





Ex: positive edge triggered

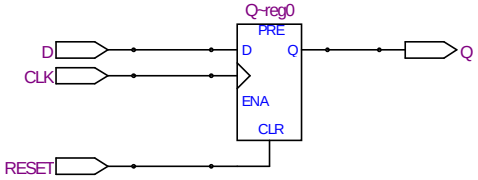


Register description


```
library IEEE;
use IEEE.std_logic_1164.all;

--Flip Flop con reset asincrono
entity FF is
port (CLK, RESET, D : in std_logic;
            Q           : out std_logic
            );
end FF;

architecture behavioral of FF is
begin
 process(CLK, reset)
   begin
            if RESET='1' then
        Q <= '0';
            elsif CLK'event and CLK='1' then
        Q <= D;
     end if;
 end process;
end behavioral;
```

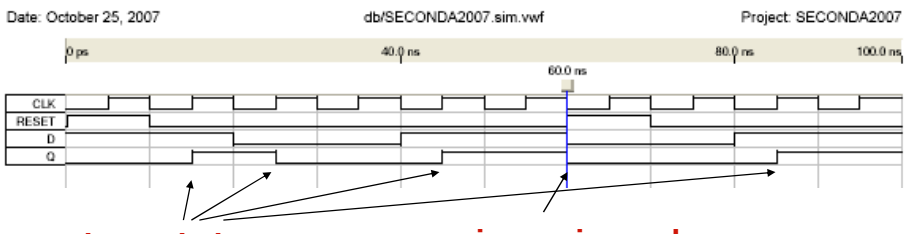
FF with Asynchronous reset

RTL viewer



Simulazione funzionale

Suggerimento per la generazione delle forme d'onda di ingresso:
fare variare gli ingressi in corrispondenza del fronte non di campionamento del clock (Tsu e Thold rispettati)



Uscita varia in corrispondenza del fronte positivo del clock

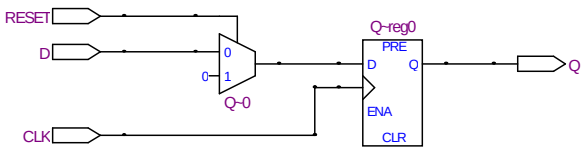
o in corrispondenza dell'attivazione del Reset asincrono

Register description

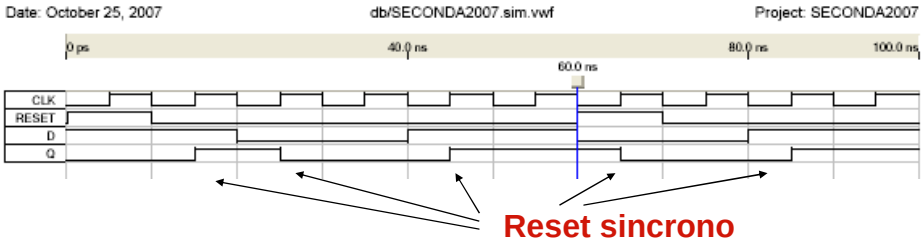
```
library IEEE;
use IEEE.std_logic_1164.all;

--Flip Flop con reset sincrono
entity FF is
port (CLK, RESET, D : in std_logic;
            Q           : out std_logic
            );
end FF;
architecture behavioral of FF is
begin
 process(CLK)
   begin
     if CLK'event and CLK='1' then
       if RESET='1' then
         Q <= '0';
       else
         Q <= D;
       end if;
     end if;
 end process;
end behavioral;
```

FF with Synchronous reset

RTL viewer



Simulazione funzionale



Reset sincrono
l'uscita varia sempre in corrispondenza del fronte positivo del clock

Register description

```
use IEEE.std_logic_1164.all;

--Flip Flop con reset sincrono
entity FF is
port (CLK, RESET, D, ENABLE : in
std_logic;
            Q          : out std_logic
            );
end FF;

architecture behavioral of FF is
begin
 process(CLK)
   begin
     if CLK'event and CLK='1' then
       if RESET='1' then
         Q <= '0';
       elsif ENABLE = '1' then
         Q <= D;
       end if;
     end if;
 end process;
end behavioral;
```

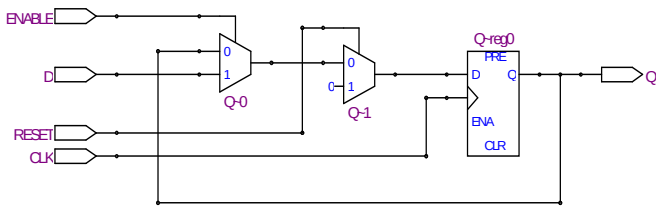
synchronous reset and enable

Register description

```
use IEEE.std_logic_1164.all;

--Flip Flop con reset sincrono
entity FF is
port (CLK, RESET, D, ENABLE : in
std_logic;
            Q           : out std_logic
            );
end FF;

architecture behavioral of FF is
begin
 process(CLK)
   begin
     if CLK'event and CLK='1' then
       if RESET='1' then
         Q <= '0';
       elsif ENABLE = '1' then
         Q <= D;
       end if;
     end if;
 end process;
end behavioral;
```

synchronous reset and enable

RTL viewer

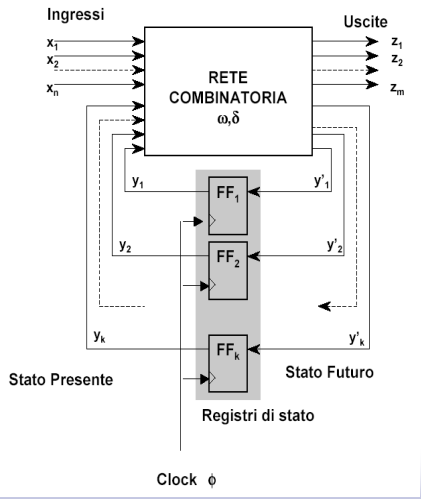


Macchina a Stati Finiti (FSM)

Z(n)=f(x(n)) => Sistema Combinatorio

Z(n)=f(x(n),x(n-1),x(n-2),…..) => Macchina a Stati finiti

Struttura di una FSM



- La rete combinatoria realizza le funzioni ω e δ (tabelle di verità) che definiscono le uscite (z) e lo stato futuro (y')
- Rete sincrona LLC (Level Level Clocked)
  - La macchina cambia stato ad ogni fronte attivo del clock (ogni nuovo "colpo di clock")
  - I registri di stato (FF) memorizzano il valore presente delle variabili di stato

Es: Contatore value(n) = value(n-1) + '1'


```
library IEEE;
 use IEEE.std_logic_1164.all;
 use IEEE.std_logic_arith.all;

entity COUNTER is
 port (
   CLK   : in  std_logic;
   RESET : in  std_logic;
   VALUE : out unsigned(15 downto 0)
   );
end COUNTER;
```



```
architecture A of COUNTER is
signal next_value, value_temp : unsigned(15 downto 0);
begin  -- A
 process(CLK)
  begin
   if CLK'event and CLK='1' then
    if RESET='1' then
     value_temp <= (others => '0');
    else
     value_temp <= next_value;
    end if;
   end if;
 end process;
 VALUE <= value_temp;
 next_value <= value_temp + conv_unsigned(1,16);
end A;
```

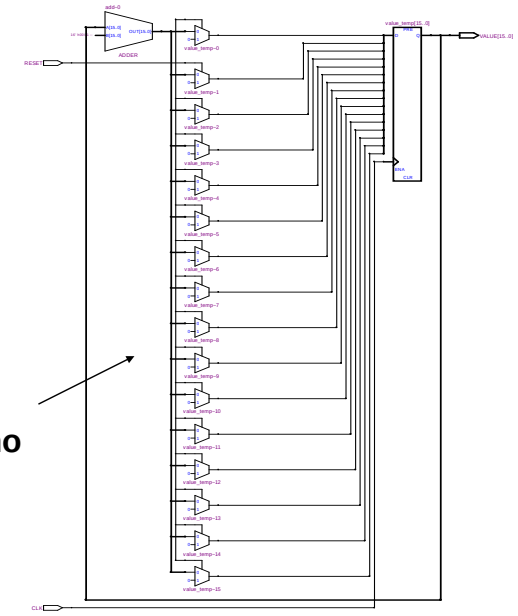
Parte sequenziale

uscita

Parte combinatoria

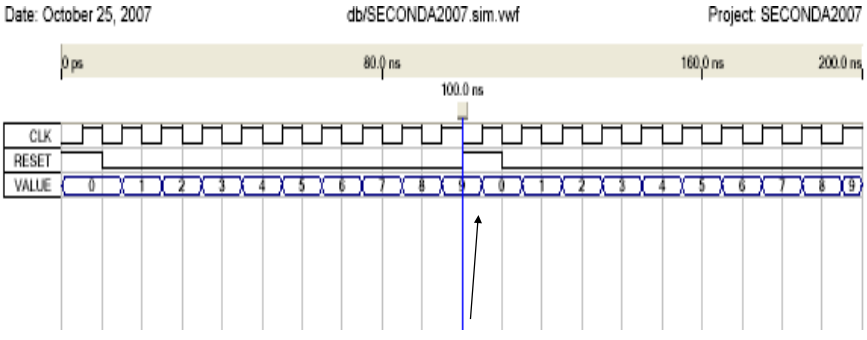
Stato futuro

RTL Viewer



Reset sincrono

Simulazione funzionale



Reset sincrono

```
architecture A of COUNTER is
signal next_value, value_temp : unsigned(15 downto 0);
begin  -- A
  process(CLK)
    begin
      if CLK'event and CLK='1' then
        if RESET='1' then
          value_temp <= (others => '1');
        else
          value_temp <= next_value;
        end if;
      end if;
    end process;
  VALUE <= value_temp;
  next_value <= value_temp -  conv_unsigned(1,16);
end A;
```

decrementa

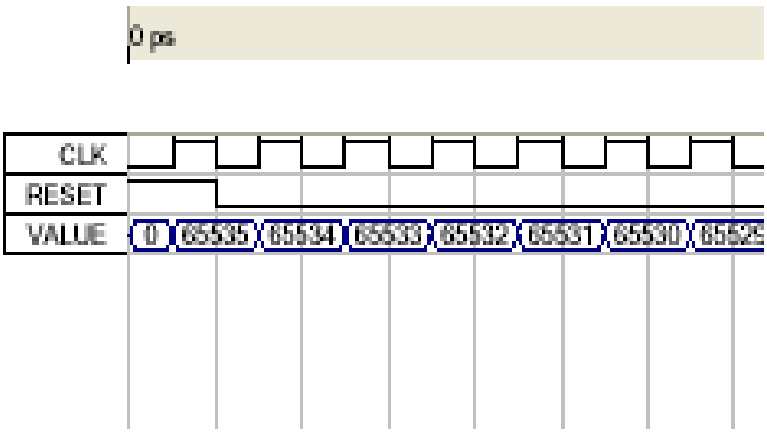
Parte sequenziale

uscita

Parte combinatoria

Stato futuro

Simulazione funzionale



Contatore con ingresso di enable

- Cambiano solo la parte sequenziale e la lista delle porte di ingresso!

```
architecture A of COUNTER is

signal next_value, value_temp : unsigned(15 downto 0);

begin
 process(CLK) ---REGISTRO CON INGRESSO DI RESET  e ENABLE

   begin
        if CLK'event and CLK='1' then
                if RESET='1' then
                     value_temp <= (others => '0');
                elsif ENABLE = '1' then
                   value_temp <= next_value;
                end if;
        end if;
   end process;

 VALUE <= value_temp;
  next_value <= value_temp + conv_unsigned(1,16);

end A;
```


Inizializzazione di registri

es:

flag <= ‘1’; flag std_logic
cont <= conv_unsigned(0, 16); cont unsigned a 16 bit
cont <= (others => ‘0’); cont unsigned a n bit
cont <= (others => ‘1’); cont unsigned a n bit

Lista moduli da avere nella propria libreria

- FF con reset sincrono e/o asincrono e ingresso di enable
- Contatori in avanti o indietro con ingresso di reset e enable

Macchina a Stati Finiti

DESCRIZIONE DI MEALY

$$Y(n+1) = F(X(n) , Y(n))$$

$$Z(n) = F(Y(n) , X(n))$$

DESCRIZIONE DI MOORE

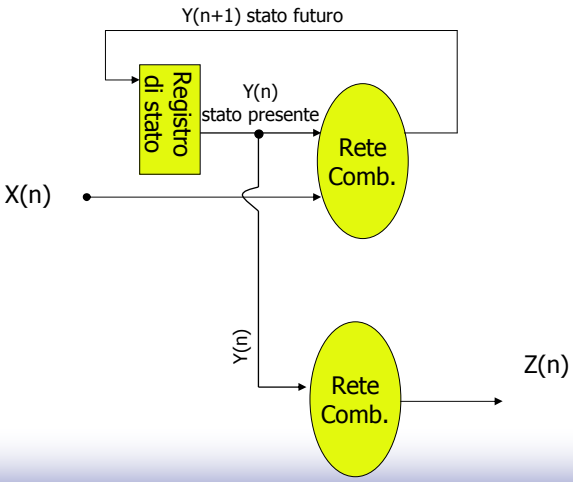
$$Y(n+1) = F(X(n) , Y(n))$$

$$Z(n) = F(Y(n))$$

Definizioni

- Una FSM è una macchina di Moore se, ad ogni istante, il valore delle uscite dipende esclusivamente dallo stato attuale e non dal valore attuale degli ingressi.
- Una FSM è una macchina di Mealy se, ad ogni istante, il valore delle uscite dipende sia dallo stato attuale che dall' attuale valore degli ingressi.
- Durante la permanenza in uno stato, una variazione dell'ingresso di una macchina di Mealy può ripercuotersi immediatamente sull'uscita.
- Una macchina di Moore può essere trasformata in una macchina di Mealy con lo stesso numero di stati.
- Una macchina di Mealy può essere trasformata in una macchina di Moore con un numero di stati generalmente maggiore.

FSM: Implementazione Fisica (Moore)



FSM: Implementazione VHDL (Moore)


```
architecture struct of FSM is

type stato is (a,b,c,d);
signal cs,ns  : stato;

begin

   -- Processo sequenziale
  process(clk,reset)
  begin
    if reset='1' then
         cs <= a;
    elsif clk'event and clk='1' then
         cs <= ns;
    end if;
  end process;
```

```
--- parte combinatoria

process(cs,ingressi)
  begin
    case cs is
        when a=> out <= uscita1;
                  if ingressi=… then
                    ns <= a;
                  elsif ingressi=… then
                    ns <= b;
                  else
                    ns <= c;
                  end if;
    when b=> out <= uscita2;
                ns <= c;
    when others => out<= uscita3;
                    if ingressi=… then
                      ns <= c;
                    else
                      ns <= b;
                    end if;
    end case;
    end process;
```

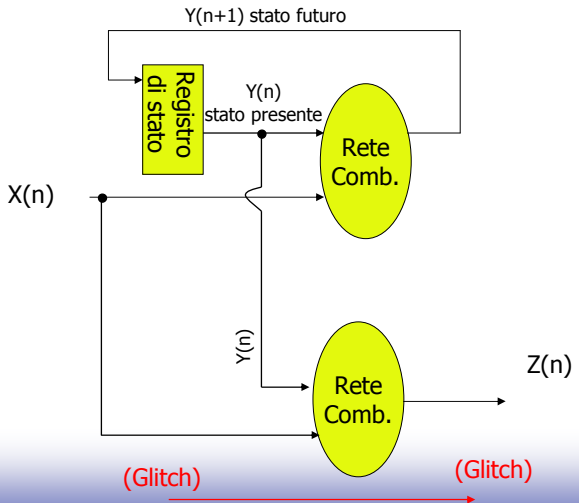

FSM: Implementazione VHDL (Moore)


```
architecture struct of FSM is
type stato is (a,b,c,d);
signal cs,ns  : stato;
begin
   -- Processo sequenziale
  process(clk,reset) – reset asincrono
  begin
    if reset='1' then
         cs <= a;
    elsif clk'event and clk='1' then
         cs <= ns;
    end if;
  end process;
```

Nella sensitivity list lo stato corrente (cs) e tutti i segnali che sono a destra delle istruzioni di assegnamento

```
--- parte combinatoria deve essere definito il valore
--- delle uscite e dello stato futuro per ogni configurazione
----possibile degli ingressi e dello stato presente
process(cs,ingressi)
  begin
    case cs is
        when a=> out <= uscita1;
                  if ingressi=… then
                    ns <= a;
                  elsif ingressi=… then
                    ns <= b;
                  else
                    ns <= c;
                  end if;
    when b=> out <= uscita2;
                ns <= c;
    when others => out<= uscita3;
                    if ingressi=… then
                      ns <= c;
                    else
                      ns <= b;
                    end if;
    end case;
    end process;
```

FSM: Implementazione Fisica (Mealy)



FSM: Implementazione VHDL (Mealy)


```
architecture struct of FSM is

type stato is (a,b,c,d);
signal cs,ns  : stato;

begin
   -- Processo sequenziale
  process(clk,reset)
  begin
    if reset='1' then
         cs <= a;
    elsif clk'event and clk='1' then
         cs <= ns;
    end if;
  end process;
```

```
--- parte combinatoria
process(cs,ingressi)
  begin
    case cs is
        when a=>
                if ingressi = ...then
                  out <= uscita1;
                elsif ingressi=... then
                  out <= uscita2;
                else
                    out=uscita3;
                end if;
                if ingressi=... then
                  ns <= a;
                elsif ingressi=... then
                  ns <= b;
                else
                  ns <= c;
                end if;
 [.....]
end case;
end process;
```

FSM: Implementazione VHDL (Mealy)


```
Architecture struct of FSM is
type stato is (a,b,c,d);
signal cs,ns  : stato;
begin
   -- Processo sequenziale
  process(clk,reset) – reset asincrono
  begin
    if reset='1' then
         cs <= a;
    elsif clk'event and clk='1' then
         cs <= ns;
    end if;
  end process;
```

Nella sensitivity list lo stato corrente (cs) e tutti I segnali che sono a destra delle istruzioni di assegnamento

```
--- parte combinatoria deve essere definito il valore
--- delle uscite e dello stato futuro per ogni configurazione
----possibile degli ingressi e dello stato presente

process(cs,ingressi)
  begin
    case cs is
        when a=>
                  if ingressi = ...then
                    out <= uscita1;
                  elsif ingressi=... then
                    out <= uscita2;
                  else
                     out=uscita3;
                  end if;
                  if ingressi=... then
                     ns <= a;
                  elsif ingressi=... then
                     ns <= b;
                  else
                     ns <= c;
                  end if;
  [.....]
end case;
end process;
```