Metodologie di Progettazione Hardware-Software

Il Very High Speed Integrated Circuit Hardware Description Language (VHDL)

Gli Hardware Description Languages

- Gli HDL consentono lo sviluppo di un modello del comportamento dei sistema digitali.
- Gli HDL permettono l'eseguibilità del codice. Errori concettuali possono essere individuati attraverso l'esecuzione di simulazioni.
- Durante la fase di sviluppo la descrizione diventa sempre più dettagliata fino ad arrivare ad un modello utilizzabile per la realizzazione del prodotto.
- Il modello HDL viene utilizzato per la sintesi automatica: la trasformazione (automatica) da un modello meno dettagliato ad uno più dettagliato.
- Esistono dei tool di sintesi che partendo da una descrizione mediante un HDL producono una descrizione mediante componenti standard di circuiti integrati.
- Gli HDL permettono la riusabilità dei modelli. Blocchi funzionali (macro) frequentemente usate vengono conservate in apposite librerie.

Fasi di sviluppo di un progetto

Specifica
Vengono definiti i requisiti del sistema.

Progettazione a livello di sistema
Viene modellato il comportamento del sistema.

Progettazione logica
Viene definita la struttura del sistema.

Progettazione circuitale
Viene realizzata la conversione automatica del modello strutturale in un modello a livello di gate (netlist).

Layout
Il layout in una specifica tecnologia.

Il passaggio da una fase alla successiva richiede una verifica funzionale.

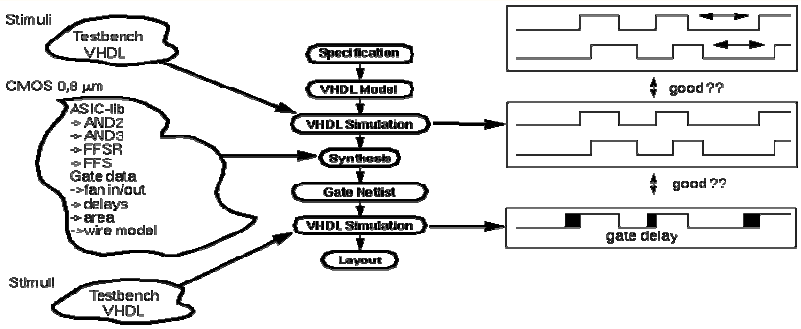
Gli HDL possono essere usati dalla fase di progetto a livello di sistema a quella a livello di gate.

Gli HDLs

- **Verilog HDL**: simile al linguaggio C. Standard
  - IEEE dal 1995;

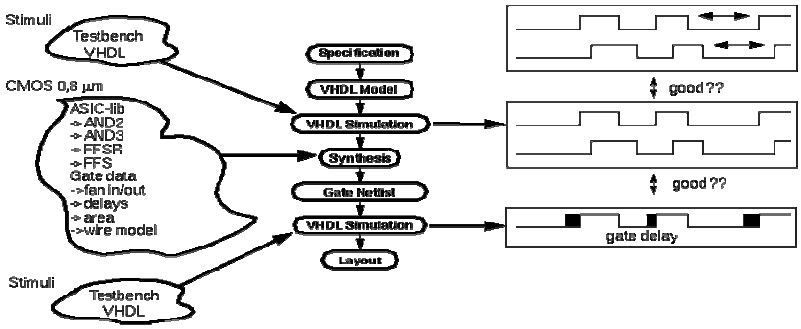
- • **VHDL**: ha la struttura di linguaggi tipo *Ada*.
  - Standard IEEE dal 1987 (1993). (IEEE-1076)

Sviluppo di modelli VHDL



- Lo sviluppo parte con la descrizione delle specifiche riguardanti le funzionalità da realizzare e le temporizzazioni.
- Talvolta viene realizzato un modello behavioral VHDL a partire dalle specifiche, di norma viene realizzato direttamente un modello RTL vhdl sintetizzabile sin dall'inizio.
- Il modello VHDL viene simulato e, se mostra il comportamento atteso, viene sintetizzato.

Sviluppo di modelli VHDL



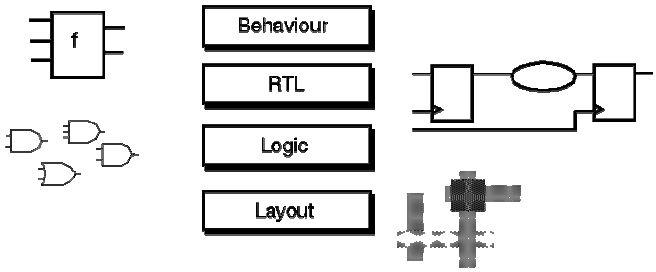
- Un tool di sintesi seleziona gli appropriati gate e flip-flops da una specifica libreria di standard cell al fine di riprodurre la descrizione VHDL.
- E' essenziale che per la procedura di sintesi che la somma dei ritardi nel path più lungo (dall'uscita all'ingresso di ogni Flip Flop) sia inferiore al periodo di clock.
- A partire dalla gate netlist vengono fatte delle simulazioni che tengono conto dei ritardi dei gate e di propagazione.

Concetti del VHDL

- Tipi di assegnazione:
  - Sequenziale
  - Concorrente

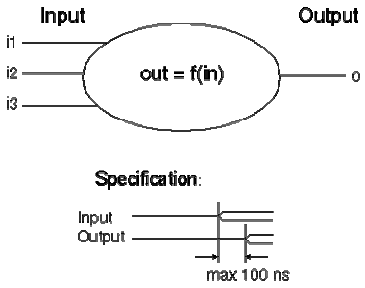
- Il VHDL è caratterizzato inoltre dalle seguenti tecniche di modeling:
  - **Astrazione**: permette di descrivere diverse parti con differente livello di dettaglio;
  - **Modularità**: permette di decomporre grandi blocchi funzionali e scrivere il modello di ogni parte;
  - **Gerarchia**: ogni modulo può essere a sua volta composto da un insieme di sottomoduli.

Livelli di astrazione nel design di un circuito integrato



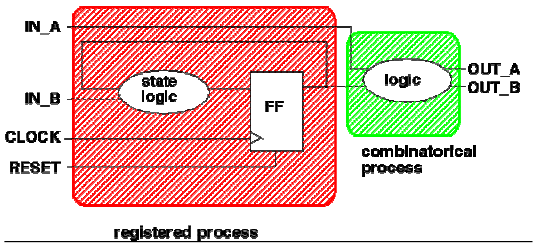
- A livello behavioral viene fatta una descrizione funzionale. Non si fa riferimento al clock di sistema. Di norma tale modello è simulabile, ma non sintetizzabile.
- A livello Register Transfer (RTL) il design viene diviso in logica combinatoria ed elementi di memoria. Solo un sottoinsieme del VHDL viene utilizzato.
- A livello logico il design è rappresentato come una netlist con gate logici (AND, OR, NOT, ...) ed elementi di memoria.
- A livello Layout le diverse celle della tecnologia target sono posizionate nel chip e le connessioni vengono definite.

Behavioral VHDL



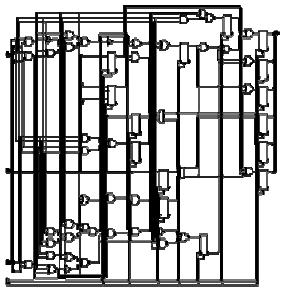
o <= i1 + i2 * i3 after 100 ns;

RT level



- A livello RT il comportamento viene descritto mediante due tipi di processi:
  - combinational process;
  - clocked process.
- I clocked process vengono realizzati con Flip-Flops e possono essere descritti mediante macchine a stati.
- Oltre ai segnali in ingresso e in uscita è necessario considerare i segnali di controllo.
- Il livello RT contiene informazioni di tipo strutturali in aggiunta a quelle funzionali.

Gate level



```
U86 : ND2 port map( A => n192, B => n191, Z => n188);
U87 : ND2 port map( A => I3_2, B => I2_0, Z => n175);
U88 : ND2 port map( A => I2_2, B => I3_0, Z => n173);
U89 : NR2 port map( A => mul_36_PROD_not_0,
      B => n174, Z => n185);
U90 : EN port map( A => n181, B => n182, Z => n180);
U91 : ND2 port map( A => I3_2, B => I2_1, Z => n181);
U92 : ND2 port map( A => I2_2, B => I3_1, Z => n182);
U93 : IVP port map( A => n180, Z => n192);
U94 : AO6 port map( A => n173, B => n174, C => n175,
      Z => n172);
U95 : NR2 port map( A => n174, B => n173, Z => n176);
U96 : ND2 port map( A => I3_1, B => I2_1, Z => n174);
U97 : EN port map( A => n183, B => n178,
      Z => product64_4);
U98 : ND3 port map( A => I2_2, B => I3_2, C => n174,
      Z => n183);
```

- Il comportamento viene descritto mediante un insieme di gate tra loro interconnessi mediante opportuni collegamenti.

ENTITY

- Nella progettazione di un modulo è necessario descrivere l'**Entity** ad essa associata.

- L'**Entity**:
  - definisce l'interfaccia (gli ingressi e le uscite) di un modulo;
  - non definisce il suo comportamento.

- Ogni **Entity** è caratterizzata da
  - un nome;
  - una sezione dichiarativa;
  - da un insieme di porte.

- Ogni porta ha associata il nome, il tipo e la sua direzione (IN, OUT, INOUT, BUFFER).

ENTITY

```
ENTITY Nome_entity IS
-- sezione dichiarativa
PORT ( p1: IN tipo1;
    p2: OUT tipo2;
    p3: BUFFER tipo3;
    p4: INOUT tipo4 );
END Nome_entity;
```

- BUFFER si comporta come OUT, ma permette di usare il valore in uscita all'interno della stessa ENTITY.

ENTITY: Esempi

```
ENTITY And4 IS
PORT ( a, b, c, d : IN std_logic;
       q       : OUT std_logic
     );
END And4;
```

```
ENTITY ffrs IS
PORT ( r,s  : IN std_logic;
    q,qn : BUFFER std_logic
);
END ffrs;
```

ENTITY: Esempi

```
ENTITY memoria IS
PORT ( addr : IN std_logic_vector(15 downto 0);
       cs, rw: IN std_logic;
       data : INOUT std_logic_vector(15 downto 0)
     );
END memoria;

ENTITY alu IS
PORT ( A, B: IN std_logic_vector(15 downto 0);
    AluOp: IN std_logic_vector(0 to 3);
       C: OUT std_logic_vector(15 downto 0)
   );
END alu;
```


ARCHITECTURE

- L'ARCHITECTURE contiene l'implementazione dell'ENTITY.
- E' sempre associata ad una specifica ENTITY.
- Una ENTITY può avere diverse ARCHITECTURE.

```
ARCHITECTURE Nome_architecture OF Nome_entity IS
  --sezione_dichiarativa_architecture
BEGIN
  --sezione_esecutiva_architecture
END Nome_architecture;
```

La sezione dichiarativa può contenere:

· tipi di dati, costanti, segnali, componenti, ...

La sezione esecutiva può contenere:

· istanze di componenti, istruzioni di assegnamento, process

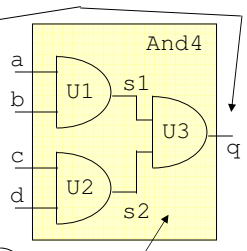
ARCHITECTURE

- La descrizione del comportamento di un modulo mediante l'ARCHITECTURE può essere fatta a diversi livelli di astrazione:
  - BEHAVIORAL, in cui un modulo viene descritto in termini di ciò che fa (come si comporta) piuttosto che di quali sono i suoi componenti e come sono interconnessi;
  - STRUCTURAL, in cui un modulo è descritto come una collezione di gate e componenti interconnessi per realizzare una determinata funzione.

- Il livello BEHAVIORAL può essere realizzato utilizzando due diversi approcci:
  - **Data flow**, che descrive come i dati si muovono attraverso il sistema, in termini di trasferimento di dati attraverso registri;
  - **Algoritmico (sequenziale)**, che descrive la sequenza di operazioni necessarie per realizzare una certa funzione.

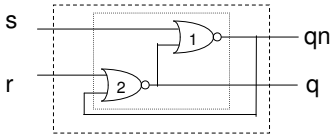
ARCHITECTURE di tipo strutturale

```
ENTITY And4 IS
PORT ( a, b, c, d : IN std_logic;
       q   :  OUT std_logic);
END And4;

ARCHITECTURE Netlist OF And4 IS
  COMPONENT and2 IS
   PORT (x,y : IN std_logic;
             z: OUT std_logic);
  END COMPONENT;
 signal s1,s2 : std_logic;

 BEGIN
  U1: and2 PORT MAP ( a,b, s1);
  U2: and2 PORT MAP (x <= c, z <=s2, y<=d);
  U3: and2 PORT MAP ( s1,s2, q);
 END Netlist;
```



ARCHITECTURE di tipo strutturale

```
ENTITY ffsr IS
Port ( s,r: IN std_logic;
    q, qn: BUFFER std_logic);
END ffsr;

ARCHITECTURE netlist OF ffsr IS
 COMPONENT nor2 IS
  PORT (a, b: IN std_logic;
         c: OUT std_logic);
 END COMPONENT;
BEGIN
 U1: nor2 PORT MAP ( s, q, qn);
 U2: nor2 PORT MAP ( b <= qn, c<= q, a<= r);
END netlist:
```



ARCHITECTURE Behavioral

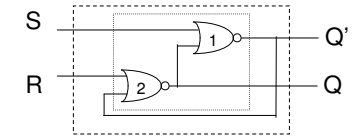
```
ENTITY And4 IS
PORT ( a, b, c, d : IN std_logic;
       q   :  OUT std_logic);
END And4;

ARCHITECTURE dataflow OF And4 IS
   signal s1,s2 : std_logic;

 BEGIN
  s1 <= a AND b AFTER 2 ns;
  s2 <= c AND d AFTER 2 ns;
  q <= s1 AND s2 AFTER 3 ns;

 END dataflow;
```

- L'esecuzione dei tre assegnamenti avviene in modo concorrente.
- L'ordine di esecuzione dipende dagli eventi (variazione del valore) sui segnali presenti sulla parte destra dell'assegnamento.
- Si dice che l'assegnamento è sensibile ai cambiamenti dei segnali presenti a destra del simbolo <=.
- L'insieme di tali segnali viene detta ***sensitivity list*** dell'assegnamento.

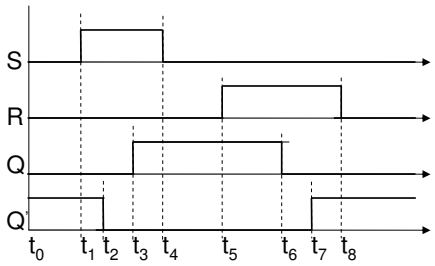
ARCHITECTURE Behavioral

```
ENTITY ffsr IS
  Port ( s,r: IN std_logic;
      q, qn: BUFFER std_logic);
   END ffsr;

ARCHITECTURE dataflow OF ffsr IS
 BEGIN
  qn <= s nor q AFTER 2 ns;
  q <= r nor qn AFTER 2 ns;

 END dataflow;
```


ARCHITECTURE Behavioral: ffsr




```
qn <= s nor q AFTER 2 ns;
q <= r nor qn AFTER 2 ns;
```



Analisi di funzionamento

$t=t_0=0$ ns: S=0, R=0, Q=0, Q'=1

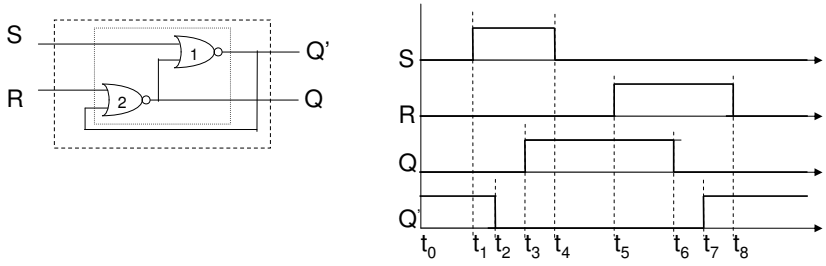
$t=t_1=10$ ns: E: (S= 0→1) ⇒ $I_1(t_1)=(1,0) \rightarrow O_1(t_2)=0 \rightarrow Q'(t_2)=0$

$t=t_2=12$ ns : E: (Q'= 1→0) ⇒ $I_2(t_2)=(0,0) \rightarrow O_2(t_3)=1 \rightarrow Q(t_3)=1$

$t=t_3=14$ ns : E: (Q = 0→1) ⇒ $I_1(t_3)=(1,1) \rightarrow O_1(t_3)=0 \rightarrow Q'(t_3)=0$

$t=t_4=15$ ns: E: (S= 1→0) ⇒ $I_1(t_4)=(0,1) \rightarrow O_1(t_4)=0 \rightarrow Q'(t_4)=0$

ARCHITECTURE Behavioral: ffsr



Analisi di funzionamento

t=t_5=20 ns E: (R=0→1) ⇒ $I_2(t_5)$=(1,0) → $O_2(t_6)$=0 → $Q(t_6)$=0

t=t_6=22 ns E: (Q=1→0) ⇒ $I_1(t_6)$=(0,0) → $O_1(t_7)$=1 → $Q'(t_7)$=1

t=t_7=24 ns E: (Q'= 0→1) ⇒ $I_2(t_7)$=(1,1) → $O_2(t_7)$=0 → $Q(t_7)$=0

t=t_8: 27 ns E: (R= 1→0) ⇒ $I_2(t_8)$=(0,1) → $O_2(t_8)$=0 → $Q(t_8)$=0

Assegnamento condizionale concorrente

```
Nome_segnale <=   Valore_1 WHEN condizione1
          ELSE
          Valore_2 WHEN condizione2
          ELSE
             ....
          Valore_N-1 WHEN condizioneN-1
   ELSE Valore_N;
```

La condizione è una qualunque espressione booleana.

Esempio di assegnamento condizionale

```
ENTITY codificatore IS
PORT (codin: IN std_logic_vector(3 downto 0);
            en : IN std_logic;
      codout: OUT std_logic_vector(1 downto 0)
        );
END codificatore;

ARCHITECTURE behavioralOF codificatore IS
 BEGIN
codout <= "10" AFTER 10 ns WHEN en='0' AND codin= "1000"
  ELSE "11" AFTER 10 ns WHEN en= '0' AND codin = "0001"
  ELSE "00" AFTER 10ns;

END behavioral;
```


Istruzione di selezione

```
WITH espressione SELECT
  Nome_segnale <=  Valore_1 WHEN scelta1,
         Valore_2 WHEN scelta2 | scelta3,
         Valore_3 WHEN scelta4 to scelta5,
            ...
         Valore_N WHEN OTHERS;
```

I valori delle scelte non possono ripetersi.

Per utilizzare scelta4 to scelta5 è necessario che per i valori di scelta sia definito un ordinamento

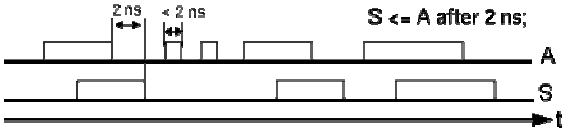
Esempio di istruzione di selezione

```
ENTITY selettore IS
 PORT (a0,a1,a2,a3 : IN std_logic;
       sel : IN integer range 0 to 7;
   q: OUT std_logic);
 END selettore;

ARCHITECURE Behavioral OF selettore IS
  BEGIN

  WITH sel SELECT
   q <= a0 WHEN 0,
      a1 WHEN 1 | 3 | 5,
      a2 WHEN 6 to 7,
      a3 WHEN OTHERS;

END Behavioral;
```


Modello dei ritardi

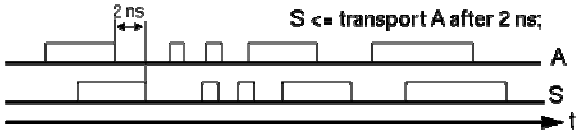
- **Inertial delay (meccanismo di default)**

  In un modello inerziale l'uscita del dispositivo cambia il suo valore solo se il nuovo valore permane per una quantità di tempo superiore alla sua inerzia.



- **Transport delay**

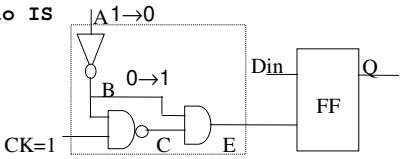
  Il transport delay rappresenta il ritardo di una linea, in cui ogni variazione viene propagata con un ritardo pari a quello specificato.



Delta di simulazione

- Questo meccanismo è usato per ordinare gli eventi durante una simulazione.

```
ARCHITECTURE Behavioural OF esempio IS
  signal B,C  : std_logic;
 BEGIN
   B <= NOT A;
   C <= NOT ( B AND CK) ;
   E <=  B AND C;
END Behavioral;
```



- Se gli eventi non sono opportunamente ordinati, simulazioni diverse potrebbero portare a risultati diversi.

AND prima	*NAND prima*
A<='0' -> Valutazione inverter (B='1')	A(1->0)-> Valutazione inverter(B='1')
B<='1'	B<='1'
Valuzione AND (E='1')	Valutazione NAND (C='0')
E<= '1'	C<='0'
Valutazione NAND (C='0')	Valutazione AND (E='0')
C<='0'	E<= '0'
Valutazione AND (E='0')	
E<= '0'	

Delta di simulazione

- Il meccanismo del Delta delay permette di realizzare simulazioni che non dipendono dall'ordine di valutazione delle istruzioni di assegnamento.
- Un Delta di simulazione consiste dei seguenti passi:
  - aggiornamento dei valori dei segnali;
  - valutazione del valore delle espressioni contenenti i segnali aggiornati al passo 1. e creazione della lista di segnali che devono cambiare il loro valore.
- I due passi vengono ripetuti fino a quando la lista dei segnali da aggiornare non è vuota.
- Il numero di Delta non modifica il tempo usato nella simulazione.
- I Delta di simulazione sono ortogonali al tempo di simulazione.

Delta di simulazione

```
BEGIN
 B <= NOT A;
 C <= NOT ( B AND CK) ;
 E <=  B AND C;
END Behavioral;
```

Tempo 10 ns	Delta1	A<= '0'
		Valutazione: (NOT A)
	Delta2	B<='1'
		Valutazione: NOT (B AND CLK)
		Valutazione: (B AND C)
	Delta3	C<= '0' E<= '1'
		Valutazione: (B AND C)
	Delta4	E<= '0'
Tempo 11 ns		

Parametrizzazione attraverso i GENERIC

I Generic sono un meccanismo usato per passare informazioni ad una istanza di una Entity.

Sono definiti nella sezione dichiarativa dell'ENTITY

```
ENTITY And2 IS
   GENERIC (rise, fall : TIME; load : INTEGER);
PORT (a, b: IN std_logic;
    c: OUT std_logic);
END And2;

ARCHITECTURE Behavioral OF And2 IS
   signal interno: std_logic;
BEGIN
  interno <= a AND b;
  c <= interno AFTER (rise + (load*2ns)) WHEN interno='1'
            ELSE
       interno AFTER (fall + (load*3ns))
END Behavioral;
```

I generic vengono trattati come delle costanti nell'architecture.

Parametrizzazione attraverso i GENERIC

```
ENTITY  and4 IS
   PORT (a, b, c, d: IN std_logic;
         q: OUT std_logic);
END and4;

ARCHITECTURE Schematic OF and4 IS

 COMPONENT  And2 IS
   GENERIC (rise, fall : TIME:= 10 ns;
                              load : INTEGER:= 1);
   PORT (a, b: IN std_logic;
    c: OUT std_logic);
 END COMPONENT;

SIGNAL s1,s2: std_logic;
BEGIN

  A1: And2 GENERIC MAP (10 ns, 12 ns, 3)
                  PORT MAP (a,b,s1);
  A2: And2 GENERIC MAP (10 ns, 12 ns, 3)
           PORT MAP (c,d,s2);
  A3: And2 PORT MAP (s1,s2,q);
END Schematic;
```


Architecture: modello sequenziale

```
ENTITY ffsr IS
   Port ( s,r: IN std_logic;
          q, qn: OUT std_logic);
    END ffsr;

ARCHITECTURE seq OF ffsr IS
BEGIN
 PROCESS(s, r)
  BEGIN
   IF s = '1' AND r = '0' THEN
    q <= '1' AFTER 2 ns;
    qn <= '0' AFTER 4 ns;
   ELSIF s = '0' AND r = '1' THEN
    qn <= '1' AFTER 2 ns;
    q <= '0' AFTER 4 ns;
   ELSE q <= '1' AFTER 2 ns;
      qn <= '0' AFTER 2 ns;
   END IF;
END PROCESS;
END seq;
```

PROCESS

```
[label:] PROCESS (sensitivity list)
         sezione_dichiarativa_process
       BEGIN
       sequenza_istruzioni
       END PROCESS [label];
```

- **Un process contiene istruzioni eseguite in modo sequenziale.**
- **E' definito all'interno dell'ARCHITECTURE.**
- **Più process vengono eseguiti in modo concorrente.**
- **L'esecuzione del process è determinata da un evento su uno dei segnali della sensitivity list.**

PROCESS

- I process comunicano tra loro attraverso i signal.

```
ARCHITECTURE Behavioral OF Esempio IS
signal c : std_logic;

pa: PROCESS (a,b)
       BEGIN
         c <= a  AND b;
       e <= not c;
        END PROCESS  pa;

pb:  PROCESS (a,c)
       BEGIN
         d <= NOT c AND a;
       END PROCESS  pb;
END Behavioral
```

Variabili

```
VARIABLE nome_variabile :tipo_variabile [:=valore_iniziale];
```

ES.

```
VARIABLE a : integer range 0 to 15;
```

Le variabili possono essere definite in un process e utilizzate solo nello stesso process.

Assegnamento ad una variabile:

```
a := espressione;
```

A differenza dei signal, l'assegnamento del nuovo valore ad una variabile avviene immediatamente.

Le variabili conservano il loro valore fino alla successiva esecuzione del process.

Istruzione IF

```
IF condizione THEN
   sequenza_di_istruzioni
[ELSIF  condizione THEN
   sequenza_di_istruzioni]
....
[ELSE
   sequenza_di_istruzioni]
END IF;
```

ELSIF può essere ripetuto più di una volta.

ELSE può apparire una sola volta.

Condizione è in tutti i casi un'espressione booleana

MUX

```
ENTITY mux2x1 IS
PORT ( a, b, sel : IN std_logic;
   q :OUT std_logic);
END mux2x1;
ARCHITECTURE sequenziale OF mux2x1 IS
BEGIN
  PROCESS(a,b,sel)
   BEGIN
    IF sel = '0' THEN q <= a;
      ELSIF sel='1' THEN q<= b;
      ELSE q <= 'X';
    END IF;
  END PROCESS;
END sequenziale;
```


FFD

```
ENTITY ffd IS
PORT ( rst,d, clk: IN std_logic;
    q: OUT std_logic );
END ffd;

ARCHITECTURE behavioral OF ffd IS
BEGIN
 PROCESS(CLK,RST)
  BEGIN
  IF RST='1' THEN q <= '0'   ;
  ELSIF clk='0' AND clk'EVENT AND clk'LAST_VALUE='1'
      THEN q<= d AFTER 2ns;
   END IF;
  END PROCESS;
END behavioral;
```

Istruzione CASE

```
CASE espressione IS
  WHEN valore_1 =>
      sequenza_di_istruzioni
  WHEN valore_2 | valore_3 =>
       sequenza_di_istruzioni
  WHEN valore_4 to valore_N =>
      sequenza_di_istruzioni
  WHEN OTHERS  =>
      sequenza_di_istruzioni
END CASE;
```


Selettore

```
ENTITY selettore IS
 PORT (a0,a1,a2,a3 : IN std_logic;
       sel : IN integer range 0 to 7;
   q: OUT std_logic);
 END selettore;

ARCHITECTURE sequenziale OF selettore IS
   BEGIN
  PROCESS(a0,a1,a2,a3,sel)
   BEGIN
   CASE sel IS
      WHEN 0       => q <= a0;
      WHEN 1 | 3 | 5   => q <= a1;
      WHEN 6 to 7      => q <= a2;
      WHEN OTHERS      => q <= a3;
   END CASE;
  END PROCESS;
END Behavioral;
```

Istruzione WHILE

```
[nome_label:]     WHILE condizione LOOP
                    sequenza_di_istruzioni
                  END LOOP [nome_label] ;
```


Confronto tra due vettori

```
ENTITY confronta IS
   PORT (a,b : IN std_logic_vector(7 downto 0);
      ris: OUT std_logic);
END confronta;

ARCHITECTURE sequenziale OF confronta IS
BEGIN
 PROCESS(a,b)
   VARIABLE ind: integer;
   VARIABLE uguale: std_logic;
    BEGIN
     ind:=0; uguale:='1';
     WHILE (uguale='1' AND ind<=7) LOOP
       IF a(ind) /= b(ind) THEN uguale := '0';
        END IF;
       ind := ind +1;
     END LOOP;
     ris <= uguale;
 END PROCESS;
END sequenziale;
```

Istruzione FOR

```
[nome_label :]
FOR identificatore IN intervallo_discreto LOOP
   sequenza_di_istruzioni
END LOOP  [nome_label] ;
```

L'identificatore :

- non deve essere dichiarato;
- può essere utilizzato solo in lettura;
- non è visibile fuori dal FOR.

Calcolo del numero di '1' in un vettore

```
ENTITY calcola IS
PORT (    a : IN std_logic_vector(7 downto 0);
      b : OUT integer range 0 to 8 );
END calcola;

ARCHITECTURE seq OF calcola IS
  BEGIN
PROCESS(a)
  VARIABLE conta : integer range 0 to 8;
  BEGIN
  conta := 0;
  FOR i IN 0 to 7 LOOP
    IF a(i) = '1' THEN conta := conta +1;
    END IF;
  END LOOP;
  b <= conta;
END PROCESS;
END seq;
```

GENERATE

- Utilizzato per la creazione di schematici con struttura regolare
- Un GENERATE può includere altri GENERATE

```
name: FOR i IN 0 TO d-1 GENERATE
        --concurrent statements
        END GENERATE name
```

Esempio

```
AND32: For i in 0 to 31 generate
       AI: and2 port map (A(i), B(i), C(i));
       end generate AND32;
```


GENERATE: IF-SCHEME

Permette la creazione condizionata di componenti

```
name: IF condizione GENERATE
                --concurrent-statements
           END GENERATE name;
```

Non può essere usato ELSE o ELSIF

Esempio

```
GA: for i in 0 to d-1 generate
     GA0: if i=0 generate
                a0: adder1 port map
(A(0),B(0),'0',C(0),R(0));
              end generate GA0;

     GAI: if i>0 generate
              ai: adder1 port map (A(i),B(i),C(i-1),C(i),
R(i));
                end generate GAI;
end generate GA;
```

Istruzione ASSERT

E' usata per fornire una stringa al progettista durante la simulazione

```
ASSERT condizione
[REPORT stringa]
[SEVERITY livello_di_severity]
```

La stringa viene visualizzata se la condizione è falsa.

Livello di severity:

- note
- warning
- error (default)
- failure

ASSERT: Esempio

```
PROCESS (b,c)
 BEGIN
b<=c;
   ASSERT (c/=a)
   REPORT "Il Valore di C è uguale a quello di a"
   SEVERITY ERROR;
END PROCESS
```

ASSERT: esempio ffd (1/2)

```
entity ffd is
 port (clk,din: in std_logic;
  dout: out std_logic);
 end ffd;

architecture beh of ffd is

begin
  process(clk,din)
   variable last_d_change: TIME:= 0 ns;
   variable last_d_value: std_logic:='X';
   variable last_clk_value: std_logic:='X';
```


ASSERT: esempio ffd (2/2)

```
begin
   if last_d_value /= din then
        last_d_change := now;
        last_d_value:=din;
   end if;

   if last_clk_value /= clk then
  last_clk_value := clk;

     if clk = '0' then       dout <= din;
       assert ( now - last_d_change >= 5 ns)
        report "setup violation"
        severity warning;
      end if;
   end if;
  end process;
end beh;
```

Istruzione WAIT

- Questa istruzione sospende l'esecuzione di un process o sottoprogramma.
- L'esecuzione può riprendere quando si verifica una certa condizione.
- Abbiamo tre possibili modi di esprimere questa condizione.
  - - WAIT ON elenco_segnali
  - - WAIT UNTIL condizione
  - - WAIT FOR tempo
- Con WAIT ON l'esecuzione riprende se c'è un evento su uno dei segnali elencati.
- Con WAIT UNTIL l'esecuzione riprende quando la condizione diventa vera.
- Con WAIT FOR l'esecuzione riprende dopo un tempo fissato.
- In un process non è possibile avere contemporaneamente sensitivity list e WAIT.

Istruzione WAIT

```
PROCESS
 BEGIN
   WAIT ON a;
   IF a='1' THEN
      b<= "10";
   ELSE b <= "01";
   END IF:
   WAIT ON c,d;
   F <= c AND d;
END PROCESS;
```

```
PROCESS
 BEGIN
   WAIT UNTIL a='1';
   c <=  d;
   WAIT UNTIL a='0';
   c <= f;
END PROCESS;
```

```
PROCESS
 BEGIN
   z <= a;
   WAIT FOR 20 ns;
   Z <= b;
   WAIT UNTIL c='1' FOR 40    ns;
   assert (c='1')
   report "time out violation"
   severity error
   z <= c;
   WAIT ON a FOR 60 ns;
END PROCESS;
```

WAIT ON

```
PROCESS
BEGIN
  IF sel = '1' THEN
     Z<= A;
  ELSE
     Z<= B;
  END IF;
  WAIT ON A,B,SEL;
END PROCESS
```

```
PROCESS(A,B,SEL)
BEGIN
  IF sel = '1' THEN
     Z<= A;
  ELSE
     Z<= B;
  END IF;
END PROCESS
```


wait: costrutti non sintetizzabili

```
process
 begin
  wait on a;
   d<= di;
 end process;
```

- -- non è sintetizzabile poiché non è presente un edge

```
process
 begin
  wait for 10 ns;
   d<= di;
 end process;
```

- -- non è sintetizzabile poiché non è presente un edge

wait: costrutti non sintetizzabili

```
process
 begin
  wait until c'event and c=1 ;
  d<= d1;
  wait until d'event and d=1 ;
  d<=d2;
 end process;
```

- -- non è sintetizzabile poiché tutti gli edge devono dipendere dallo stesso segnale

wait: costrutti non sintetizzabili

```
process
 begin
  wait until c'event and c='1' ;
  d<= d1;
  wait until c'event and c='0' ;
  d<=d2;
 end process;
```

-- non è sintetizzabile poiché i wait devono contenere un solo edge

wait: costrutti non sintetizzabili

```
process
 begin
  wait until d'event and c='1' ;
  d<= d1;
  wait until c'event and c='0' ;
  d<=d2;
 end process;
```

-- non è sintetizzabile poiché i wait devono contenere un solo edge

wait: costrutti non sintetizzabili

```
process
 begin
  wait until c'event;
  d<= d1;
  wait until c'event;
  d<=d2;
 end process;
```

-- non è sintetizzabile poiché non può essere determinata la polarità dell'edge

wait: costrutti non sintetizzabili

```
entity es is
  port (a,c : in integer;
        d : out integer);
end es;
architecture beh of es is
 begin
  process
   begin
      wait until c'event and c= 2;
      d<= a;
  end process;
end beh;
```

- Non è sintetizzabile perché il clock deve essere di un solo bit

Tipi di dati

Tipi scalari
- **INTEGER;**
- **REAL;**
- **Tipo Fisici;**
- **BOOLEAN;**
- **CHARCTER;**
- **BIT**
- **ENUMERATO;**

Tipo composto
- **ARRAY**
- **RECORD**

FILE
ACCESS

Definizione di un nuovo tipo

```
TYPE Nome_tipo IS Definizione_tipo;
```

TIPO Enumerato

```
TYPE quattrovalori IS ('0', '1', 'X','Z');

TYPE std_ulogic IS
    ('U','X','0','1','Z','W','L','H','-');

TYPE istruzioni IS (add, sub, lw, sw, mov, beq);

TYPE Stato IS
     (Fetch, Decode, Execute, MemAccess, WriteBack);
```


TIPO FISICO

```
TYPE TIME IS RANGE <implementation defined>
  UNITS
  fs;
  ps=1000 fs;
  ns=1000ps;
  us=1000ns;
  ms=1000us;
  sec=1000ms;
  min=60 sec;
  hr = 60 min;
  END UNITS;

TYPE CURRENT IS RANGE 0 TO 1000000000
  UNITS
  na;
  ua=1000na;
  ma=1000ua;
  a=1000ma;
  END UNITS;
```

ARRAY

Tipi predefiniti : bit_vector(array di bit) e string (array di character)

Definizione di un array monodimensionale

```
TYPE Nome_array IS ARRAY ( range_indici) OF
  Tipo_base;
```

Es.

```
TYPE data_bus IS array( 0 to 31) OF BIT;

TYPE byte IS ARRAY(7 downto 0) OF std_logic;

TYPE word IS ARRAY(31 downto 0) OF std_logic;
```


Riferimento agli elementi di un ARRAY monodimensionale

```
ARCHITECTURE Behavioral OF test IS
   SIGNAL w,x,y: word;
   SIGNAL b,c,f: byte;
   SIGNAL d,e: std_logic;
BEGIN
   d<= w(i);
   b<=y(7 downto 0);
   c<=b;
   x(15 downto 0) <=b&c;       -- concatenazione
   y(31 downto 16) <=b&b;
   y(15 downto 0) <= "1010101010101010";
   e <= '1';
   f<=('1','0','0','1','1','1','0','1');
    --aggregazione

END Behavioral;
```

ARRAY multidimensionali

```
TYPE Nome_array IS ARRAY(Range_1,.., Range_N) OF
  Tipo_base;

Es.
TYPE Memoria IS ARRAY (0 to memsize-1, 0 to dim-
  1) OF std_logic;

VARIABLE DataMem : Memoria;

DataMem(i,j) := A;
```

- **Nel caso in cui definiamo un array di array il riferimento agli elementi è differente.**

```
TYPE MemDati IS ARRAY (0 to memsize-1) OF word;

VARIABLE Mem: MemDati;

Mem(i)(j):=A;
```


ARRAY con dimensione non specificata

- **Sono array la cui dimensione non è specificata al momento della sua dichiarazione.**

```
TYPE Nome_tipo IS ARRAY (NATURAL RANGE <>) OF Tipo_base
```

Es.

```
TYPE std_logic_vector IS ARRAY (NATURAL RANGE <>) OF
  std_logic;
```

- **Questo tipo di array vengono utilizzati per definire gli argomenti di sottoprogrammi o entity port.**

Esempio: AndVect

```
ENTITY AndVect IS
  PORT ( a: IN std_logic_vector(7 downto 0);
                    z: OUT std_logic;
END AndVect;

ARCHITECTURE Behavioral OF AndVect IS
   BEGIN
   Process(a)
     VARIABLE temp : std_logic;
    BEGIN
       temp:='1';
       FOR i IN 0 to 7 LOOP
       temp:= temp AND a(i);
    END LOOP;
    z <= temp;
     END Process;
 END Behavioral;
```


RECORD

```
TYPE Nome_record IS
  RECORD    campo1: tipo1;
         campo2: tipo2;
         ....
       campoN: tipoN;
  END RECORD;
```

Record: Esempio

```
TYPE opcode_type IS (add, sub, lw, sw, mov, beq);

TYPE istruzione IS
   RECORD opcode:  opcode_type;
          src1:    integer range 0 to 31;
          src2: integer range 0 to 31;
          dst:  integer range 0 to 31;
   END RECORD;

VARIABLE ist: istruzione;
VARIABLE op :  opcode_type;

op:= ist.opcode;

ist := (add, 1,2,3);
```


SUBTYPE

La dichiarazione dei SUBTYPE è utilizzata per definire sottoinsiemi di un tipo.

Permette di evitare la definizione di un nuovo tipo.

```
type MY_WORD is array (15 downto 0) of std_logic;
subtype SUB_WORD is std_logic_vector (15 downto
0);

subtype MS_BYTE is integer range 15 downto 8;
subtype LS_BYTE is integer range 7 downto 0;
```

SUBPROGRAMS

Consiste di funzioni e procedure

Function

- il nome di una funzione può essere un operatore;
- può avere un numero arbitrario di parametri di ingresso;
- restituisce un solo valore;

Procedure

- può avere numero arbitrario di parametri di ogni possibile direzione (IN,OUT,INOUT)
- istruzione RETURN opzionale (non restituisce alcun valore !)

E' possibile realizzare l'overloading dei subprogram

I parametri possono essere constants, signals, variables o files.

Esiste una versione sequenziale e concorrente delle function e delle procedure.

FUNCTION

```
FUNCTION Nome_Function( par1: tipo1, ..., parN:
  tipoN)
  RETURN TipoValore_restituito
IS
--sezione_dichiarativa_function
BEGIN
-- sezione_esecutiva
 RETURN valore_restituito;
END  Nome_Function
```

Esempi di function

```
FUNCTION  AndVect  (  op1:  std_logic_vector;  op2  :
  std_logic_vector ) RETURN std_logic_vector
   IS
  VARIABLE temp:
       std_logic_vector(op1'length-1 downto 0);
 BEGIN
   FOR i IN 0 to op1'length-1 LOOP
  temp(i) := op1(i) AND op2(i);
   END LOOP;
   RETURN temp;
 END AndVect;


FUNCTION  rising_edge  (  SIGNAL  clk:  std_logic)  RETURN
  BOOLEAN
  IS
 BEGIN
  IF (clk'EVENT) AND (clk='1') THEN RETURN true;
   ELSE RETURN false;
  END IF;
 END rising_edge;
```


PROCEDURE

```
PROCEDURE Nome_procedure
( par1: dir1 tipo1; ....; parN: dirN tipoN)
  IS
--sezione_dichiarativa_procedure
BEGIN
--sezione_esecutiva_procedure
END  Nome_procedure;
```

dove diri ∈{IN, OUT,INOUT}

Esempio di PROCEDURE

```
PROCEDURE And32
( op1: IN word; op2 : IN word; ris: OUT word)
    IS
  VARIABLE temp: word;
 BEGIN
   FOR i IN 0 to 31 LOOP
  temp(i) := op1(i) AND op2(i);
   END LOOP;
   ris := temp;
 END And32;
```


Esempio di Procedure

```
TYPE bus_parità IS RECORD
          valori: word;
          pari: std_logic;
       END RECORD;

PROCEDURE calcoloparità (x : INOUT bus_parità)
   IS
  VARIABLE p: INTEGER;
BEGIN
   p:=0;
   FOR i IN 0 to 31 LOOP
      IF (x.valori(i)='1' THEN p:=p+1;
      END IF;
  END LOOP;
  IF (p MOD 2 = 1) x.pari := '1';
    ELSE x.pari := '0';
  END IF;
END calcoloparità;
```

PACKAGE

Contiene elementi che possono essere condivisi tra diverse entità.

Consiste di due parti:
- una sezione dichiarativa, che definisce l'interfaccia del package;
- il corpo del package, che definisce il comportamento del package.

Package declaration

Può contenere le seguenti dichiarazioni:
- dichiarazioni di subprogram, dichiarazioni di tipi e sottotipi
- constant o deferred constant, signal
- dichiarazione di file, dichiarazione di componenti

Package body

Può contenere le seguenti dichiarazioni:
- dichiarazioni di subprogram, definizione dei subprogram
- dichiarazione di di tpi e sottotipi, costanti
  dichiarazione di file

PACKAGE (1)

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

package dati is
 subtype w16 is std_logic_vector(15 downto 0);
 subtype w32 is std_logic_vector(31 downto 0);
constant z32 : w32
   :="ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";
 constant zero32 :w32 :=
  "00000000000000000000000000000000";

 type  vect is array ( natural range <>) of std_logic;
 type vect16 is array (natural range <>) of w16;
 type vect32 is array (natural range <>) of w32;
 function vect2int(vettore : std_logic_vector) return
  integer;
 end dati;
```

PACKAGE (2)

```
package body dati is
  function vect2int(vettore : std_logic_vector)
  return integer is
  variable risultato: integer:=0;
  variable dim: integer;

  begin
  dim:= vettore'length;
  for ind in dim-1 downto 0 loop
   risultato:=risultato*2;
   if vettore(ind)='1' then
   risultato:=risultato+1;
   end if;
  end loop;
  return risultato;
 end vect2int;

end dati;
```


Il tipo File

- **`TYPE nome_tipo_file IS FILE OF Tipo_base`**

- **Esempio:**

```
TYPE integer_file IS FILE OF INTEGER;
```

- **Dichiarazione di un oggetto di tipo file**

```
FILE nomefile : nometipo [ OPEN modalità IS "path
  name" ]
```

dove modalità è
{WRITE_MODE, READ_MODE, APPEND_MODE}

- **Esempio:**

```
FILE myfile_readmode: integer_file

  FILE myfile_writemode: integer_file OPEN
WRITE_MODE IS "/test/esempi/data_file"
```

Il tipo file

- Apertura di un file

```
FILE_OPEN ( nome_file, "pathname", modalità
apertura);
```

- Chiusura di un file

```
FILE_CLOSE( nome_file);
```


Operazione di accesso ai file

- READ(file,data)
  Legge da file e restituisce in data il valore letto
- WRITE(file,data)
  Scrive su file il valore presente in data
- ENDFILE(file)
  Restituisce true se si è alla fine del file

File di testo

- Il tipo TEXT è definito nel package std.textio

  FILE nome_file: TEXT;

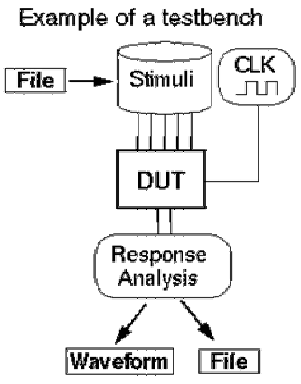
Un file di testo viene visto come un insieme di LINE

- readline(nome_file,nome_line);
  Legge una line da file

- writeline(nome_file,nome_line);
  Scrive una line su file

Per leggere da tastiera si usa *input* come nome del file, per visualizzare sullo schermo si usa *output*

Simulazione: Testbench



Un TestBench è una entità che

- -fornisce gli stimoli (testvectors) per il Device Under Test (DUT) ;
- -non deve essere sintetizzabile;
- -non ha bisogno di porte per l'esterno;
- -contiene al suo interno come component il DUT;

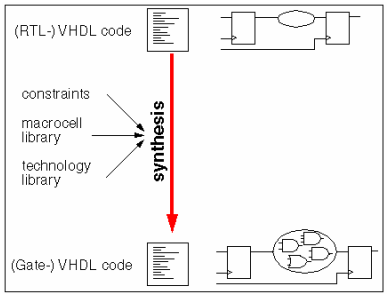
ESEMPIO di TESTBENCH (1/2)

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

entity TESTBNCH is
end TESTBNCH;

architecture stimulus of TESTBNCH is

--. Sezione dichiarativa

component ADDER4 is
    port (
        a,b: in std_logic_vector(3 downto 0);
        ci: in std_logic;
        s: out std_logic_vector(3 downto 0);
        co: out std_logic);
end component;

signal a,b: std_logic_vector(3 downto 0);
signal ci: std_logic;
signal s: std_logic_vector(3 downto 0);
signal co: std_logic;
```


ESEMPIO di TESTBENCH (2/2)

```
begin
    DUT: ADDER4 port map ( a, b, ci, s, co );

    StimuliA: process
      begin
        a <= "0010";
        wait for 20 ns;
       a <= "0110";
        wait for 20 ns;
        a <= "0110";
    wait for 20 ns;
end process StimuliA;

    b  <= "1001" AFTER 0ns, "0001" AFTER 30 ns,
    "1101" AFTER 50 ns ;
    ci<= '0' AFTER 0ns, '1' AFTER 60 ns;

end stimulus;
```

Processo di sintesi



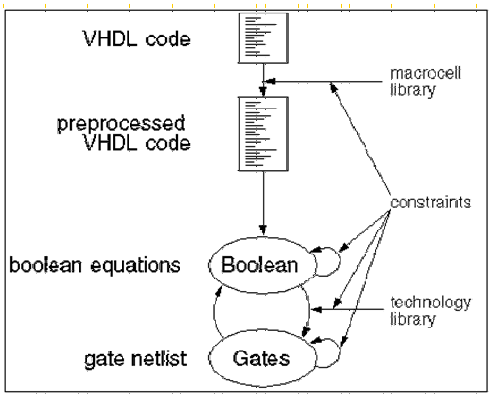
Trasformazione da una descrizione astratta in una più dettagliata.

Es. L'operatore "+" è trasformato in una netlist;
"if (VEC_A = VEC_B) then"
è realizzato come un comparatore che controlla un multiplexer

Solo un sottoinsieme del VHDL è sintetizzabile

Tool differenti supportano sottoinsiemi differenti.

Processo di sintesi



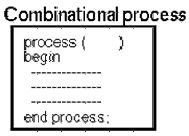
Contraints
•speed
•area
•power

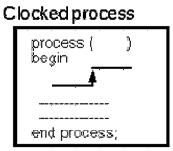
Macrocells
•adder
•comparator
•bus interface

Optimizations
•boolean: mathematic
•gate: technological

Regole per la progettazione RTL

- Un sistema può essere descritto come un insieme di combinational process e clocked process.
- Le seguenti regole è necessario rispettare:

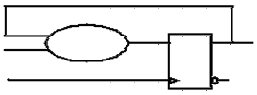




- Sensitivity list completa
- Istruzione IF completa o assegnamento di default

Struttura if-elsif-elsif-end if in cui:

- Il primo IF è un reset
- L'ultimo elsif contiene il test sul clock
- Non è presente l' else.



Logica Combinatoria

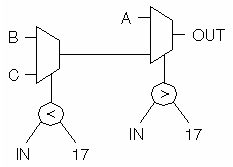
Non creare loop all'interno del process

```
process (X, B)
 begin
  X <= X + B;
end process;
```

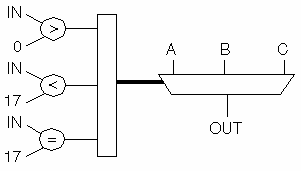
Logica Combinatoria

- Il modo in cui si scrive il codice influenza la sintesi

```
if (IN > 17) then
   OUT <= A ;
elsif (IN < 17) then
   OUT <= B ;
else
   OUT <=  C ;
end if ;
```



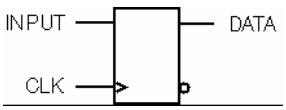

```
case IN is
 when 0 to 16 => OUT <= B ;
 when 17 =>  OUT <= C ;
 when others => OUT <= A ;
end case ;
```



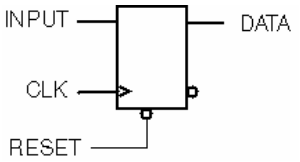
Logica sequenziale

- Un meccanismo di reset è richiesto per inizializzare i registri

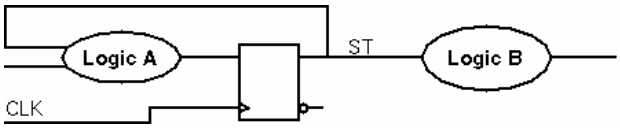
```
process
begin
  wait until CLK`event and
  CLK=`1`;
  DATA <= INPUT ;
end process;
```




```
process(CLK,RESET)
begin
 if (RESET = `1`) then
    DATA <= `0` ;
 elsif (CLK`event and CLK=`1`)
   then   DATA <= INPUT ;
end if ;
end process ;
```

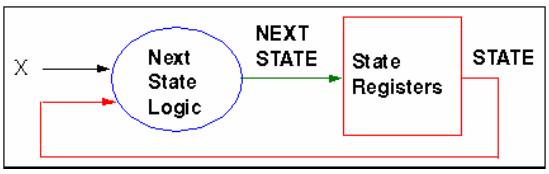


Logica sequenziale



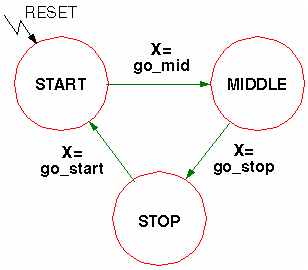

```
LOGIC_A: process
begin
wait until CLK`event and CLK=`1`;
--   Logic A
end process LOGIC_A;


LOGIC_B: process (ST)
begin
--  Logic B
end process LOGIC_B;
```


Realizzazione di una FSM in VHDL



- Può essere realizzata con uno o due process

Esempio:



Realizzazione con un solo process

```
FSM_FF: process (CLK, RESET)
begin
  if RESET='1' then
     STATE <= START  ;
  elsif CLK'event and CLK='1' then
     case  STATE  is
       when  START  => if  X=GO_MID  then STATE <= MIDDLE  ;  end if ;
       when  MIDDLE  => if  X=GO_STOP  then STATE <= STOP  ; end if ;
       when  STOP    => if  X=GO_START  then STATE <= START ;       end
     if ;
       when  others  =>  STATE <= START  ;
     end case ;
  end if ;
end process FSM_FF ;
```


Realizzazione con due process

```
FSM_FF: process (CLK, RESET)
begin
  if RESET='1' then
     STATE <= START  ;
  elsif CLK'event and CLK='1' then
     STATE  <=  NEXT_STATE  ;
 end if;
end process FSM_FF ;

FSM_LOGIC: process ( STATE , X)
begin
case  STATE  is
     when  START   => if  X=GO_MID then NEXT_STATE <= MIDDLE; end if;
     when  MIDDLE  => ...
     when  others  =>  NEXT_STATE <= START  ;
end case ;
end process FSM_LOGIC ;
```

Quanti Processi ?

Struttura e leggibilità

Separazione tra logica combinatoria ed elementi di memoria.
=> 2 processes

Gli stati FSM cambiano con speciali ingressi.
=> 1 process è più comprensibile

La rappresentazione grafica suggerisce un solo process.
=> 1 process

Quanti Processi ?

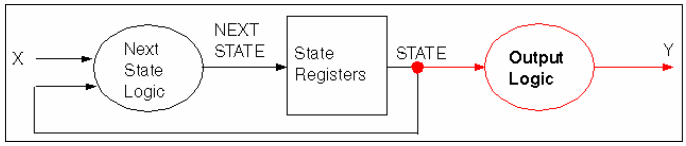
Simulazione

L'individuazione degli errori è più semplice con due process.
=>2 processes

Sintesi

Due processi portano generalmente a una netlist più piccola.
=> 2 processes

FSM MOORE



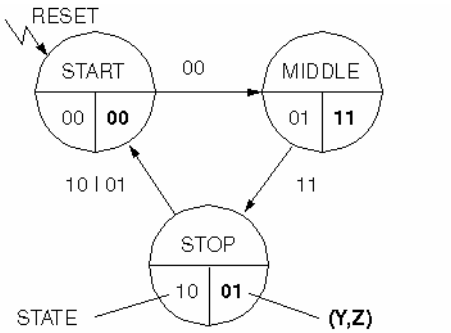
Three Processes

```
architecture RTL of MOORE is
begin
REG: -- Clocked Process
CMB: -- Combinational Process

OUTPUT: process (STATE)
begin
-- Output Logic
end process OUTPUT ;
end RTL ;
```

Two Processes

```
architecture RTL of MOORE is
begin
REG: process (CLK, RESET)
  begin
   --State Registers Inference
   --with Next State Logic
 end process REG ;

OUTPUT: process (STATE)
 begin
  -- Output Logic
 end process OUTPUT ;
end RTL ;
```


FSM Moore: esempio

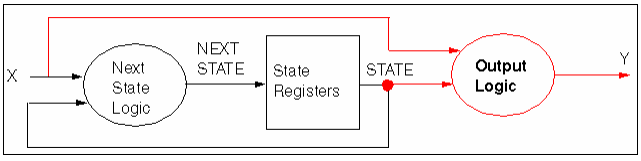



```
subtype STATE_TYPE is std_ulogic_vector(1 downto 0);
constant START   : STATE_TYPE := "00";
constant MIDDLE  : STATE_TYPE := "01";
constant STOP    : STATE_TYPE := "10";
```

FSM Moore: esempio

```
architecture RTL of MOORE_TEST is
    signal STATE,NEXTSTATE : STATE_TYPE ;
begin
 REG: process (CLK, RESET) begin
    if RESET=`1` then    STATE <= START ;
    elsif CLK`event and CLK=`1` then
      STATE <= NEXTSTATE ;
end if ;    end process REG ;

CMB: process (A,B,STATE) begin
    NEXT_STATE <= STATE;
    case STATE is
      when START => if (A or B)=`0` then NEXTSTATE  <= MIDDLE ;  end if ;
      when MIDDLE => if (A and B)=`1` then NEXTSTATE <= STOP ;  end if ;
      when STOP    => if (A xor B)=`1` then NEXTSTATE <= START ;  end if ;
      when others => NEXTSTATE <= START ;
    end case ;
 end process CMB ;

Y <= ,'1' when STATE=MIDDLE else ,'0` ;
Z <= ,'1' when STATE=MIDDLE or STATE=STOP else ,'0` ;
end RTL;
```


FSM MEALY



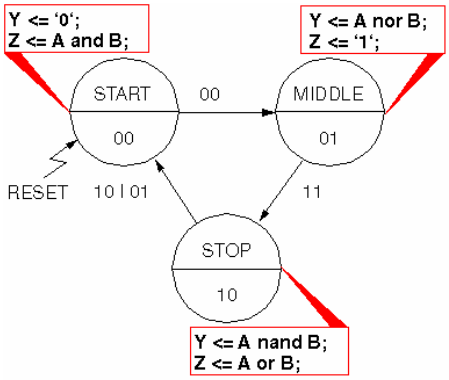
Three Processes

```
architecture RTL of MEALY is
begin
REG: -- Clocked Process
CMB: -- Combinational Process

OUTPUT: process (STATE, X)
begin
-- Output Logic
end process OUTPUT ;
end RTL ;
```

Two Processes

```
architecture RTL of MEALY is
begin
MED: process (CLK, RESET)
begin
-- State Registers Inference with
Next State Logic
end process MED ;

OUTPUT: process (STATE, X)
begin
-- Output Logic
end process OUTPUT ;
end RTL ;
```

FSM Mealy: esempio




```
subtype STATE_TYPE is std_ulogic_vector(1 downto 0);
constant START    : STATE_TYPE := "00";
constant MIDDLE   : STATE_TYPE := "01";
constant STOP     : STATE_TYPE := "10";
```

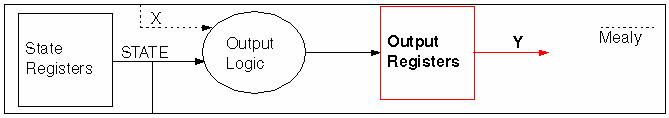

FSM Mealy: esempio

```
architecture RTL of MEDVEDEV_TEST is
signal STATE,NEXTSTATE : STATE_TYPE ;
begin
REG: · · ·   -- clocked STATE process

CMB: · · ·   -- Come gli esempio di Moore

OUTPUT: process (STATE, A, B)
 begin
case STATE is
  when START   => Y <= `0` ;  Z <= A and B ;
  when MIDLLE  =>Y <= A nor B ; Z <= '1' ;
  when STOP    =>  Y <= A nand B ; Z <= A or B ;
  when others  => Y <= `0` ; Z <= '0' ;
    end case;
  end process OUTPUT;
end RTL ;
```

Registered Output

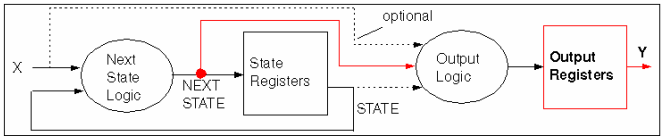
Risolve i problemi di Mealy: path lunghi, spike, loop.

Soluzione 1:



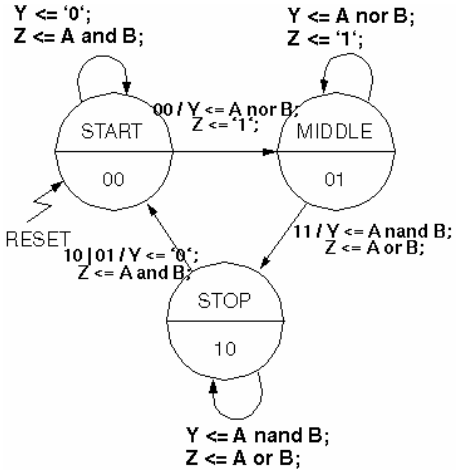
Richiede un periodo di clock addizionale

Soluzione 2:



Non è richiesto un altro ciclo.

Registered Output: Esempio



Registered Output: Esempio

```
architecture RTL of REG_TEST2 is
     signal Y_I , Z_I : std_ulogic ;
     signal STATE,NEXTSTATE : STATE_TYPE ;
begin

REG: · · ·   -- clocked STATE process

CMB: · · ·   -- Come negli altri esempi

OUTPUT: process ( NEXTSTATE , A, B)
begin
case NEXTSTATE is
     when START   => Y_I<= `0` ; Z_I<= A and B ;
     · · ·
end process OUTPUT

OUTPUT_REG: process(CLK)
begin
if CLK'event and CLK='1' then Y <= Y_I ; Z <= Z_I ;
end if ;
end process OUTPUT_REG ;
end RTL ;
```