Un linguaggio per la descrizione dello hardware: il 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.

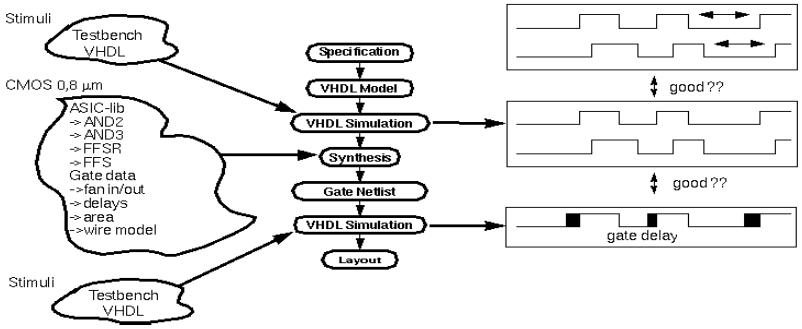
Gli HDL possono essere usati dalla fase di progetto a livello di sistema a quella a livello di gate.

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



- 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.

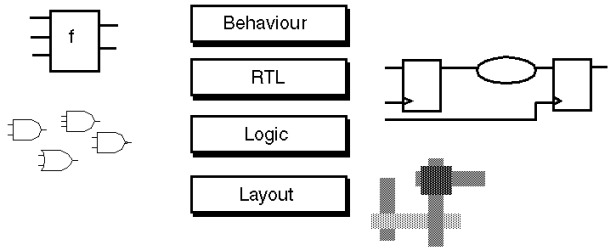
Concetti del VHDL

- Il VHDL separa esplicitamente l'interfaccia (ENTITY) dal corpo del componente (ARCHITECTURE)

- 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.

ENTITY

- 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 BIT;
        q          : OUT BIT
     );
END And4;
```

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

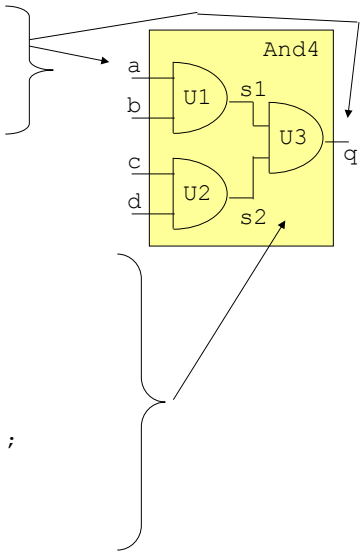
Architecture Structural

Un and a 4 ingressi and4 può essere realizzato con uno schematico che utilizza come componente and2 così definito:

```
ENTITY and2 IS
PORT ( x, y: IN BIT;
       z   : OUT BIT);
END and2;

ARCHITECTURE beh OF and2 IS

 BEGIN
   z <= x AND y;

 END beh;
```

Architecture: Structural And4

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

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

 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;
```



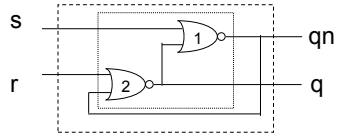
Architecture: Structural ffsr

Un ffsr può essere realizzato con uno schematico che utilizza come componente and2 così definito:

```
ENTITY nor2 IS
  PORT (a, b: IN BIT;
                c: OUT BIT);
 END nor2;

ARCHITECTURE beh of nor2 is
BEGIN
 c <= NOT ( a OR b) ;
END beh;
```

Architecture: Structural ffsr

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

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

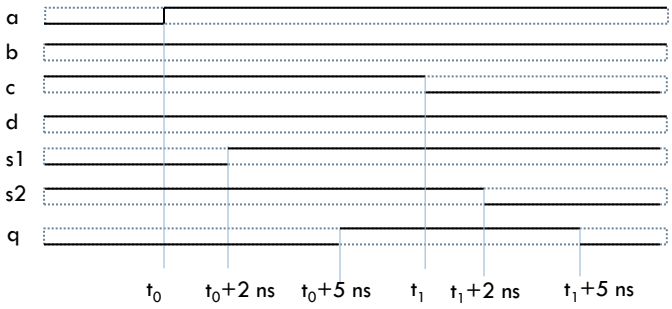


ARCHITECTURE: Behavioral

```
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.

Sensitivity list



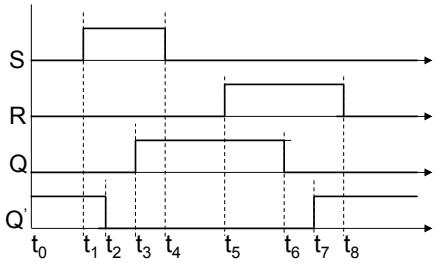
t0: evento su a –> viene eseguita **`s1 <= a AND b AFTER 2 ns;`**
t0+2ns: evento su s1 -> viene eseguita **`q <= s1 AND s2 AFTER 3 ns;`**
t0 + 5ns: evento su q -> nessun ulteriore esecuzione
t1: evento su c -> viene eseguita **`s2 <= c AND d AFTER 2 ns;`**
t1+2ns: evento su s2 -> viene eseguita **`q <= s1 AND s2 AFTER 3 ns;`**
t1 + 5ns: evento su q -> nessun ulteriore esecuzione

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
SR

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



t_0: S=0, R=0, Q=0, Q'=1

t_1: evento su S –> viene eseguita `qn <= s nor q AFTER 2 ns;`

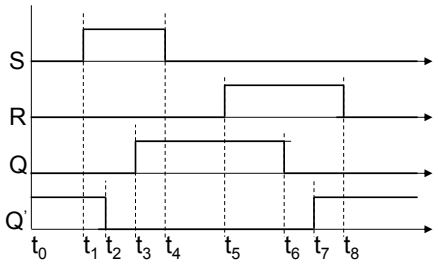
t_2=t_1+2ns: evento su qn-> viene eseguita `q <= r nor qn AFTER 2 ns;`

t_3:=t_1+4ns: evento su q -> viene eseguita `qn <= s nor q AFTER 2 ns;`

nessuna variazione su qn -> nessun altra esecuzione

Architecture: Behavioral
SR

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

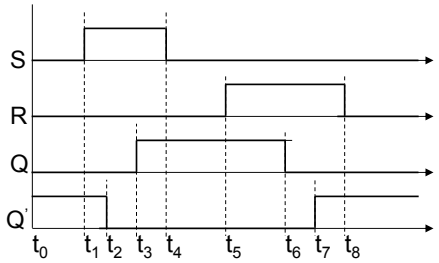


t_4: evento su S –> viene eseguita `qn <= s nor q AFTER 2 ns;`

nessuna variazione su qn -> nessun altra esecuzione

Architecture: Behavioral
SR

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



t_5: evento su R –> viene eseguita `q <= r nor qn AFTER 2 ns;`

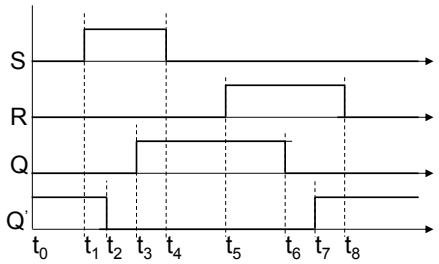
$t_6=t_5+2ns$: evento su q-> viene eseguita `qn <= s nor q AFTER 2 ns;`

$t_7:=t_1+4ns$: evento su qn -> viene eseguita `q <= r nor qn AFTER 2 ns;`

nessuna variazione su q -> nessun altra esecuzione

Architecture: Behavioral
SR

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



t_8: evento su R –> viene eseguita `q <= r nor qn AFTER 2 ns;`

nessuna variazione su q -> nessun altra esecuzione

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.

Il Multiplexer 2x1

Dispositivo che permette di selezionare uno degli *n* ingressi e presentarlo in uscita

- Con n linee di ingresso un multiplexer richiede un numero di linee di comando pari a $\lceil \log_2(n) \rceil$

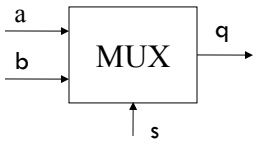


Tabella della verità

a	b	s	q
0	-	0	0
1	-	0	1
-	0	1	0
-	1	1	1

f(a,b,s)= a s'+b s

Il Multiplexer 2x1 (v1)

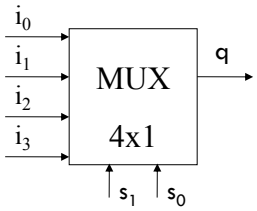
```
library ieee;
use ieee.std_logic_1164.all;

entity mux2x1 is

port ( a, b: in bit;
       s: in bit;
       z: out bit );
end mux2x1;

architecture behavioral_v1 of mux2x1 is

begin
   z <= a after 1 ns when s = '0' else
       b after 1 ns ;
end behavioral_v1;
```

Il Multiplexer 2x1 (v2)

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

entity mux2x1 is

port ( a, b: in bit;
       s: in bit;
       z: out bit );
end mux2x1;

architecture behavioral_v2 of mux2x1 is

begin
   z <= ( a and  not s ) or ( b and s) after 1;
end behavioral_v2;
```

II Multiplexer 4x1

Con 4 linee di ingresso sono richieste due linee di comando



$f(i_3,i_2,i_1,i_0,s_1,s_0) = i_3 s_1 s_0 + i_2 s_1 s_0' + i_1 s_1' s_0 + i_0 s_1' s_0'$

II Multiplexer 4x1 (structural)

```
entity mux4x1 is
port (i0,i1,i2,i3 : in bit;
      s1,s0:        in bit;
      z:            out bit );
end mux4x1;

architecture schematic of mux4x1 is
  component mux2x1 is
   port (     a, b: in bit;
              s: in bit;
              z: out bit );
  end component;
 signal z1,z2: bit;

begin
  m0: mux2x1 port map ( a=> i0, z => z1, b => i1, s => s0);
  m1: mux2x1 port map (i2,i3,s0,z2);
  m2: mux2x1 port map (z1,z2,s1,z);
end schematic;
```

Il Multiplexer 4x1 (behavioral)

```
entity mux4x1 is
port (i0,i1,i2,i3 : in bit;
      s1,s0:        in bit;
      z:            out bit );
end mux4x1;

architecture behavioral of mux4x1 is

begin
  z <= i0  after 1 ns when s1 = '0' and s0 = '0' else
       i1  after 1 ns when s1 = '0' and s0 = '1' else
       i2  after 1 ns when s1 = '1' and s0 = '0' else
       i3  after 1 ns;

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

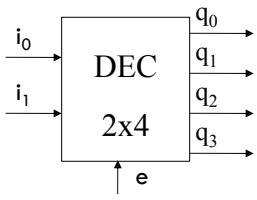
Esempio con with

```
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 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;
```

Il decodificatore (in logica diretta)

Un decodificatore (1 su *m*) accetta in ingresso un codice di *n* bit e presenta in uscita *m*=2*n* linee, sulle quali asserisce solo quella che corrisponde alla codifica in ingresso

- Numerando le linee di uscita da 0 a 2*n*-1, viene asserita quella che corrisponde al numero presente in ingresso



$q_0 = e\ i_1'\ i_0'$

$q_1 = e\ i_1'\ i_0$

$q_2 = e\ i_1\ i_0'$

$q_3 = e\ i_1\ i_0$

Decoder (v1 con with)

```
entity decoder2x4 is
port ( i: in integer range 0 to 4;
       en: in std_logic;
       q: out std_logic_vector( 0 to 3)
       );
 end decoder2x4;

 architecture dataflow of decoder2x4 is
 signal t: std_logic_vector(0 to 3);
 begin

    with i select
     t <=  "1000" when 0,
           "0100" when 1,
           "0010" when 2,
           "0001" when 3,
           "0000" when others;

     q <= t after 2 ns when en= '1' else
          "0000";
 end dataflow;
```

Decoder (v2)

```
entity decoder2x4 is
port ( i: in bit_vector(0 to 1);
       en: in bit;
       q: out bit_vector( 0 to 3)
       );
 end decoder2x4;

 architecture dataflow of decoder2x4 is
 signal t: bit_vector(0 to 3);
 begin

     t <=  "1000" when i="00" else
           "0100" when i="01" else
           "0010" when i="10" else
           "0001" ;

     q <= t after 2 ns when en= '1' else
          "0000";
 end dataflow;
```

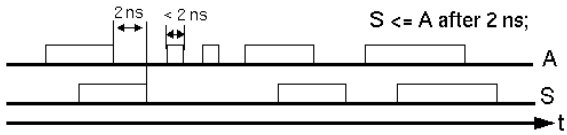
Decoder (v3)

```
entity decoder2x4 is
port ( i: in bit_vector(0 to 1);
       en: in bit;
       q: out bit_vector( 0 to 3)
       );
 end decoder2x4;

 architecture dataflow of decoder2x4 is
  begin

     q <=  "1000" after 2 ns when i="00" and en = '1' else
           "0100" after 2 ns when i="01" and en = '1' else
           "0010" after 2 ns when i="10" and en = '1' else
           "0001" after 2 ns when i="10" and en = '1' else
            "0000" after 2 ns;


 end dataflow;
```

Modello dei ritardi

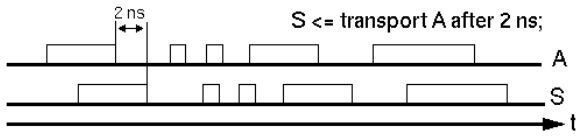
- **Inertial delay (meccanismo di default)**

  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**

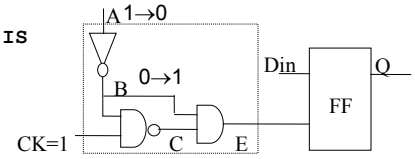
  Il transport delay rappresenta il ritardo di una linea, in cui ogni variazione viene propagata con un ritardo pari a quello specificato.



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		

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.

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;
```