Note su VHDL

22 gennaio 2004

Design Entity

- L'unità di base di un modello VHDL consiste nella Design Entity, che può rappresentare un intero sistema, un circuito stampato, un circuito integrato oppure una porta logica elementare.
- La *Entity Declaration* definisce l'interfaccia del modello.
- L'*Architecture Body* definisce la funzionalità del modello.
- All'interno di un modello VHDL, ad una stessa Entity Declaration possono corrispondere diverse Architecture Body.

Struttura di un modello VHDL

- Ad una singola interfaccia di un modello VHDL possono corrispondere diverse Architecture Body ciascuna descritta ad un diverso livello di astrazione (comportamentale, dataflow, strutturale o mista)
- Ogni diversa architettura rappresenta una diversa realizzazione della stessa funzionalità del modello per mettere in luce un diverso aspetto progettuale cioè fornisce una diversa implementazione di una stessa funzionalità.

Entity Declaration

- Definisce una entity e la sua interfaccia con l'ambiente esterno, non definisce la funzionalità del modello.
- Formato:

```
entity ENTITY_NAME is
  port (PORT_LIST); -- lista dei segnali
                    -- di interfaccia;
end [ENTITY_NAME];
```

- Esempio: Definizione della entity `nor_gate`.

```
entity NOR_GATE is
  port (A, B : in bit;
            C : out bit);
end NOR_GATE;
```

Istruzione `PORT`

- Identifica l'insieme dei segnali di interfaccia della entity per comunicare con l'ambiente esterno.
- Formato:

```
port (NAME_LIST: mode type;
        ...
      NAME_LIST: mode type);
```

- `mode` identifica la direzione del segnale cioè la direzione del flusso dati attraverso una porta (**`in`**, **`out`**, **`inout`**) .
- **`type`** identifica il tipo del segnale cioè un insieme di valori che il segnale può assumere (esempio 0 e 1) e un insieme di operazioni che possono essere eseguite sul segnale.

Architecture Body

- Descrive la funzionalità del modello attraverso la definizione delle relazioni funzionali tra gli ingressi e le uscite di un modello.
- Formato:

```
architecture BODY_NAME of ENTITY_NAME is
   -- istruzioni dichiarative
begin
   -- istruzioni per descrivere la
   -- funzionalita' del modello
end [BODY_NAME];
```

- Esempio: Definizione della architecture `nor_gate`.

```
architecture DATA_FLOW of NOR_GATE is
begin
   C <= A nor B;
end DATA_FLOW;
```

Tipi di dato VHDL

Nel package `STD.STANDARD`:

- `BOOLEAN` (false, true)
- `BIT` ('0', '1')
- `INTEGER` (intervallo che dipende dall'implementazione)

Nel package `IEEE.STD_LOGIC_1164`:

- `STD_ULOGIC` ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-')
- `STD_LOGIC` ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-')

Vettori

vector

```
port (a: in std_logic_vector(0 to 3);
      b: out bit_vector(3 downto 1));
```

array

```
type T_DATA is array (0 to 3) of
          std_logic_vector(7 downto 0);
constant DATA : T_DATA := ("00000000",
                           "00000001",
                           "00000010",
                           "00000100");
```

Nota: per costanti binarie si usano gli apici (`'1'`), mentre per i vettori di due o più bit si usano le doppie virgolette (`"001"`)

- L'assegnamento:

```
signal Z_BUS: bit_vector(3 downto 0);
signal A_BUS: bit_vector(1 to 4);
-- ...
Z_BUS <= A_BUS;
```

- equivale a:

```
Z_BUS(3) <= A_BUS(1); -- referenziazione per indici
Z_BUS(2) <= A_BUS(2);
Z_BUS(1) <= A_BUS(3);
Z_BUS(0) <= A_BUS(4);
```

- equivale a:

```
Z_BUS(3 downto 1) <= A_BUS(1 to 3); -- bit-slicing
Z_BUS(0) <= A_BUS(4);
```

I segnali si possono comporre per formare un bus tramite due operatori:

- Concatenazione (solo sul lato destro dell'assegnamento):

```
architecture EXAMPLE of CONCATENATION is
signal BYTE : bit_vector (8 downto 0);
signal A_BUS, B_BUS : bit_vector (3 downto 0);
signal C : bit;
begin
   BYTE   <= A_BUS & C & B_BUS;
end EXAMPLE;
```

- Aggregazione:

```
architecture EXAMPLE of AGGREGATES is
   signal BYTE :  bit_vector (7 downto 0);
   signal Z_BUS : bit_vector (3 downto 0);
   signal A_BIT, B_BIT, C_BIT, D_BIT : bit;
begin
   Z_BUS <= ( A_BIT, B_BIT, C_BIT, D_BIT );
   ( A_BIT, B_BIT, C_BIT, D_BIT ) <= bit_vector("1011");
   ( A_BIT, B_BIT, C_BIT, D_BIT ) <= BYTE(3 downto 0);

   BYTE <= ( 7 => '1', 5 downto 1 => '1', 6 => B_BIT, others => '0') ;
end EXAMPLE;
```

Operatori

Operatori definiti su
`BOOLEAN`, `BIT`, `BIT_VECTOR`, `STD_LOGIC`, `STD_LOGIC_VECTOR`

logici **`AND NAND OR NOR XOR NOT`**

relazionali `= /= < > <= >=`

Gli operatori relazionali sui vettori sono definiti bit a bit, allineati a destra:
`"111"> "1011"` è vero

Descrizione strutturale

Esempio:

```
ENTITY latch IS
  port ( s, r: in bit;
         q, nq: out bit);
END latch;

ARCHITECTURE structure of latch IS
   component nor_gate     -- componente da includere
      port (a,b: in bit;
              c: out bit);
   end component;
BEGIN
   n1: nor_gate port map (r, nq, q);          -- istanze del componente incluso
   n2: nor_gate port map (b=>q, a=>s, c=>nq);
END structure;
```

Assegnamento selettivo

- Il costrutto **`WITH-SELECT-WHEN`** è utilizzabile all'interno di una descrizione concorrente
- Sintassi:

```
WITH <expression> SELECT
<signal_name> <= <signal_or_value> WHEN <condition1>,
                 <signal_or_value> WHEN <condition2>,
                  ...
                 <signal_or_value> WHEN OTHERS;
```

- le condizioni devono essere mutuamente esclusive
- tutti i valori possibili dell'espressione devono essere coperti
- Esempio:

```
ENTITY and2 IS
   PORT (a : IN BIT_VECTOR (1 DOWNTO 0);
         y : OUT BIT);
END and2;

ARCHITECTURE arch_and2 OF and2 IS
BEGIN
   WITH a SELECT
      y <=  '1' WHEN "11",
            '0' WHEN "00",
            '0' WHEN "01",
```

```
            '0' WHEN "10";
END arch_and2;
```

- L'uso di **OTHERS** rende la descrizione più sintetica e leggibile, come si può apprezzare all'aumentare delle possibili combinazioni:

```
ENTITY and12 IS
   PORT (a : IN BIT_VECTOR (11 DOWNTO 0);
         y : OUT BIT);
END and12;

ARCHITECTURE arch_and12 OF and12 IS
BEGIN
   WITH a SELECT
      y <=  '1' WHEN "111111111111",
            '0' WHEN OTHERS;
END arch_and12;
```

Assegnamento condizionato

- Il costrutto **WHEN-ELSE** è utilizzabile all'interno di una descrizione concorrente
- Sintassi:

```
<signal_name> <= <signal_or_value> WHEN <cond1> ELSE
                 <signal_or_value> WHEN <cond2> ELSE
                 -- ...
                 <signal_or_value> WHEN <condN> ELSE
                 <signal_or_value>;
```

- le condizioni vengono valutate in sequenza e ci si ferma alla prima verificata
- deve sempre esistere un assegnamento non condizionato
- Esempio:

```
ENTITY and2 IS
   PORT (a, b : IN BIT;
         y : OUT BIT);
END and2;

ARCHITECTURE arch_and2 OF and2 IS
BEGIN
   y<='1' WHEN (A='1' AND B='1') ELSE
      '0';
END arch_and2;
```

- C'è sempre più di un modo per descrivere la stessa cosa:

```
ENTITY and2 IS
       PORT (a: IN BIT_VECTOR (1 DOWNTO 0);
         y : OUT BIT);
END and2;

ARCHITECTURE arch_and2_bis OF and2 IS
BEGIN
   y<='0' WHEN a(0)='0' ELSE
      '0' WHEN a(1)='0' ELSE
      '1';
END arch_and2_bis;
```

- oppure:

```
ENTITY and2 IS
       PORT (a: IN BIT_VECTOR (1 DOWNTO 0);
             y : OUT BIT);
END and2;

ARCHITECTURE arch_and2_tris OF and2 IS
BEGIN
   y<='1' WHEN a="11" ELSE
      '0';
END arch_and2_tris;
```

Process

- Un modello VHDL è un insieme di processi che interagiscono tra loro in parallelo (concorrenza tra processi)
- costituiti da istruzioni sequenziali
- e che si scambiano informazioni tramite i segnali
- Sintassi:

```
[<label>:] PROCESS [(<sensitivity_list>)]
BEGIN
   <seq_instr1>
   ...
   <seq_instrN>
END PROCESS [<label>];
```

Sensitivity list

- Un processo viene eseguito quando avviene un *evento* nella sua *sensitivity list*.
  - L'esecuzione di un processo causa uno o più eventi alle sue uscite.
  - Tali eventi posso attivare altri processi.
- La sensitivity list deve contenere tutti i segnali che vengono letti all'interno del processo
- La sensitivity list viene utilizzata solo durante la simulazione, non durante la sintesi.
  - è importante che durante la simulazione il modello abbia un comportamento il più fedele possibile al comportamento dopo la sintesi

WAIT ON

- L'istruzione **wait on** può sostituire la sensitivity list
- **wait** è mutuamente esclusivo rispetto alla sensitivity list
- Esempio:

```
process
begin   if (SEL = '1') then
      OUT <= A;
   else
      OUT <= B;
   end if;
   wait on A, B, SEL;
end process;
```

IF-THEN-ELSE

- **if-then-else** è un costrutto *sequenziale*
- ```
  IF <condition>
    THEN <istruzione1>;
    ELSE <istruzione2>;
  END IF;
  ```
- la condizione deve essere un'espressione booleana
- estensione:

  ```
  IF <condition1>
    THEN <itruzione1>;
    ELSIF <condition2>
    THEN <istruzione2>;
    ELSE <istruzione3>;
  END IF;
  ```

CASE-WHEN

- ```
  CASE <selection_signal> IS
     WHEN <value_1_selection_signal> => <instr_1>;
     WHEN <value_2_selection_signal> => <instr_2>;
     WHEN <value_3_selection_signal> => <instr_3>;
     ...
     WHEN <value_n_selection_signal> => <instr_n>;
     WHEN OTHERS => <istruzione_others>;
  END CASE;
  ```
- Tutti i possibili valori del segnale di selezione devono essere coperti

loop

- **while-loop**

  ```
  WHILE (<condition>) LOOP
     <instruction>;
  END LOOP;
  ```
- **for-loop**

  ```
  FOR <var> IN <min_value> TO <max_value> LOOP
     <instruction>;
  END LOOP;

  FOR <var> IN <max_value> DOWNTO <min_value> LOOP
     <instruction>;
  END LOOP;
  ```

Temporizzazione

- Delay:

  ```
  NEW_SIGNAL <= SIGNAL_EXPR after TIME_PERIOD;
  ```
- Esempi:

```
C <= A and B after 10 ns;
D_OUT <=  1  after 10 ns,
          0  after 20 ns,
          1  after 30 ns,
          0  after 40 ns;
Q <=  1  after 10 ns when b =  1  else
      0  after 5 ns;
```

WAIT

- il cambiamento di uno o più segnali (WAIT ON)
- il verificarsi di una espressione booleana (WAIT UNTIL)
- la fine di un determinato intervallo di tempo (WAIT FOR)

Attributo EVENT

- ogni segnale ha una serie di attributi associati, denotati dal costrutto:
  `<nome_segnale>'<nome_attributo>`
- uno dei più usati è l'attributo booleano `event`
  - esempio: `clock'event`
- `event` assume il valore `true` solo nell'istante in cui il segnale associato cambia di valore (si verifica un evento sul segnale)
- `event` è utilizzato per individuare un fronte di risalita:
  - esempio: `clock'event and clock='1'` assume il valore `true` solo in occasione del fronte di risalita (`clock` cambia valore ed assume il valore `'1'`)

Testbench

- *test bench* = banco di prova
- per verificare che un componente di comporti secondo le specifiche di progetto, lo si sottopone ad una prova:
  - si forzano in ingresso stimoli controllati
  - e si verifica che le uscite corrispondano
- questo procedimento in VHDL è realizzabile mediante un simulatore VHDL:
  - si costruisce una entità che descrive il test da eseguire (*testbench*)
    - dovrà inglobare l'entità da verificare (*unit under test*, UUT)
    - dovrà contenere una descrizione degli stimoli da applicare in ingresso alla uut
- l'entità di testbench non necessita di porte di ingresso o di uscita
- i segnali di ingresso possono essere fatti variare nel tempo, in modo da coprire più configurazioni
- talvolta le combinazioni dei valori dei segnali di ingresso sono troppo numerose per poter essere valutate, quindi il test viene limitato alle configurazioni più significative
- i segnali possono essere descritti utilizzando istruzioni concorrenti (assegnamento con clausola **`after`**) o sequenziali (costrutti **`wait`** e **`wait for`** in istruzioni **`process`**).

```
entity Testbench is
end Testbench;

architecture behav of Testbench is

  -- interfaccia dell'unita' sotto test
  component UUT
    port (
       a, b: in  bit;
       c   : out bit);
  end component;

  signal a_s, b_s, c_s: bit;

begin
  -- inclusione dell'unita' sotto test
  test_unit: UUT port map (a_s, b_s, c_s);

  -- descrizione dei segnali di stimolo

  a_p: process -- andamento del segnale a_s
    begin
      a_s <= '0';
      wait for 10 ns;
      a_s <= '1';
      wait for 10 ns;
    end process;

  b_p: process   -- andamento del segnale b_s
    begin
      b_s <= '0';
      wait for 20 ns;
      b_s <= '1';
      wait for 20 ns;
    end process;
end behav;
```