Architetture e Reti logiche

— Esercitazioni VHDL —

a.a. 2003/04

◇ *VHDL* ◇

Stefano Ferrari

Università degli Studi di Milano

Dipartimento di Tecnologie dell'Informazione

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

Esempio

Differenza di comportamento?

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

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

La sintesi produrrà lo stesso circuito.

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>;
  ELSEIF <condition2>
  THEN <istruzione2>;
  ELSE <istruzione3>;
END IF;
```

Esempio (comportamentale)

```
architecture ARC1 of DEC24 is
begin
  P1: process (A, B)
  begin
    if (A='0' and B='0') then
      Z <= "1000";
    elsif (A='0' and B='1') then
      Z <= "0100";
    elsif (A='1' and B='0') then
      Z <= "0010";
    elsif (A='1' and B='1') then
      Z <= "0001"; end if;
  end process P1;
end ARC1;
```


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

Esempio

```
architecture ARC2 of DEC24 is
begin
  P2: process (A, B)
  begin
    case (A & B) is
      when "00" => Z <= "1000";
      when "01" => Z <= "0100";
      when "10" => Z <= "0010";
      when "11" => Z <= "0001";
    end case;
  end process P2;
end ARC2;
```


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)

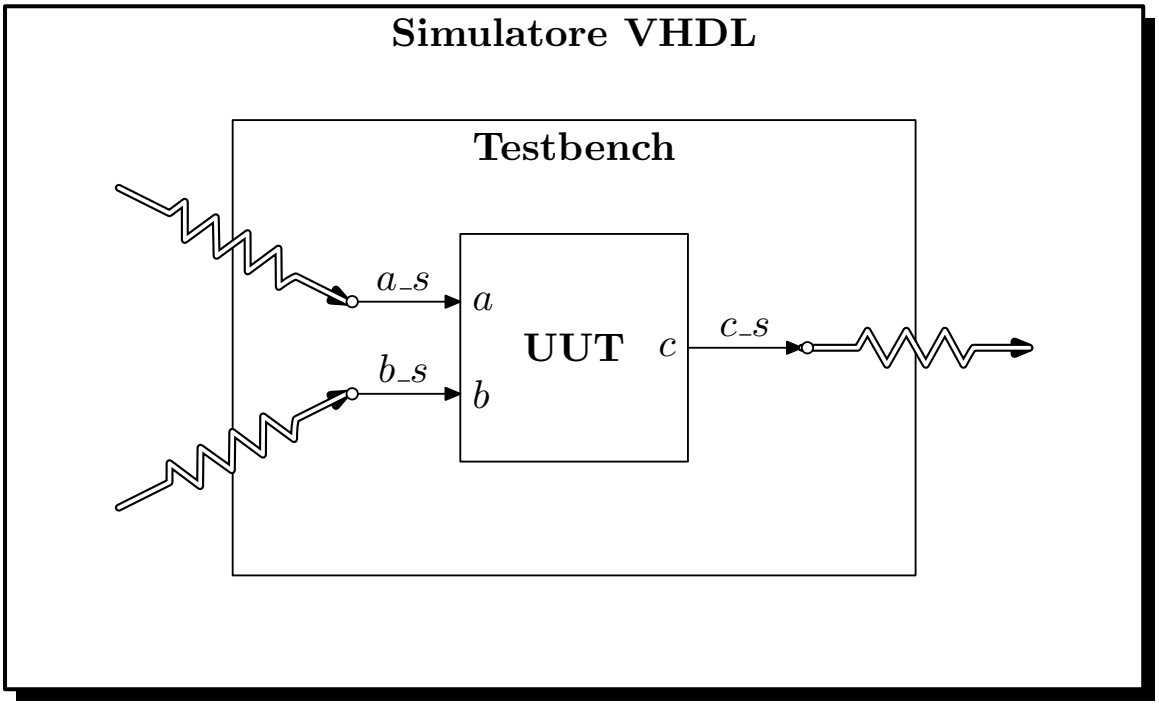
Attributo EVENT

- 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

Testbench (2)



Testbench (3)

- 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`**).

Testbench (4)

```
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);
```


Testbench (5)

```
  -- 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;
```