Lezione E16

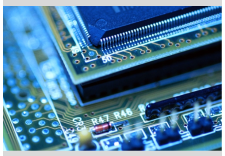
VHDL

Sistemi embedded e real-time

29 gennaio 2013

Marco Cesati

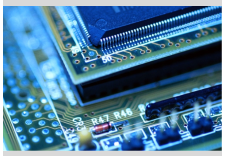
Dipartimento di Ingegneria Civile e Ingegneria Informatica
Università degli Studi di Roma Tor Vergata



Di cosa parliamo in questa lezione?

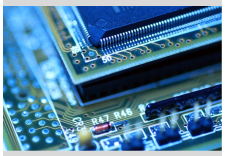
In questa lezione si continua a descrivere il linguaggio di programmazione dell'hardware VHDL

1. La descrizione comportamentale
2. Il costrutto process
3. Sincronizzazione con segnale di clock
4. Variabili
5. Segnali
6. Collegamento di componenti



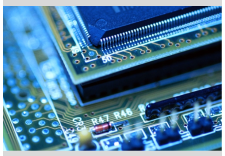
La descrizione comportamentale

- La descrizione strutturale si basa sulla semplice descrizione dei collegamenti tra porte logiche e componenti
  - I “puristi” distinguono anche tra descrizione strutturale e descrizione del flusso di dati
- La descrizione comportamentale consente di specificare l’architettura (implementazione) di un componente descrivendo come deve comportarsi
- La descrizione dell’interfaccia del componente è identica
  - basata sul costrutto `entity`
- È possibile utilizzare contemporaneamente diversi tipi di descrizione
  - una parte dell’architettura ha una descrizione comportamentale
  - un’altra parte ha una descrizione strutturale



La descrizione comportamentale (2)

- La descrizione comportamentale è particolarmente adatta quando il circuito è facilmente esprimibile come un algoritmo
- In particolare questa descrizione è conveniente implementando automi a stati finiti
- In generale è necessario preservare le informazioni interne del componente (ad esempio, lo stato dell’automa)
  - uso di flip-flop e registri
  - la logica prodotta è sequenziale, non combinatoria
- Per ottenere buone prestazioni con elevate frequenze di funzionamento si implementano circuiti sincroni
  - le macchine sincrone sono pilotate dal fronte attivo di un segnale di clock

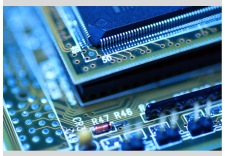


Il costrutto process

La descrizione comportamentale è basata sul costrutto `process`

```
NOMEPROC : process( <lista segnali> ) is
    <elenco segnali e variabili>
begin
    <istruzioni in sequenza>
end process;    -- oppure end NOMEPROC;
```

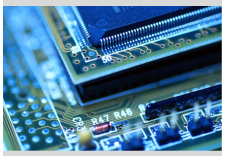
- Le istruzioni all'interno del costrutto `process` sono eseguite in sequenza
  - analogo ad un linguaggio di programmazione del software
  - diversi costrutti `process` sono "eseguiti" in parallelo tra loro
- I circuiti corrispondenti ad un `process` sono attivati solo nelle circostanze definite dalla "`lista segnali`" dopo la parola chiave "`process`"
- La memoria "locale" del `process` è definita prima del "`begin`"



Sensitivity list

- È l'elenco dei segnali che segue il costrutto "`process`"
- Indica i segnali che attivano il circuito descritto
- In effetti il circuito è attivato per ogni cambiamento di stato di uno qualsiasi dei segnali nella lista

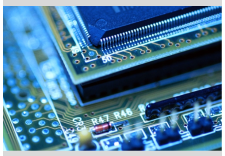
```
entity Prova is
   port (a :  IN std_logic;
         z :  OUT std_logic);
end entity;
architecture Behav of Prova is
begin
   process (a) is
   begin
      z <= not a;
   end process;
end architecture;
```

Sincronizzazione con segnale di clock

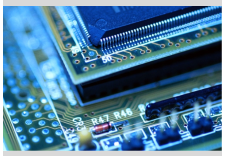
Per realizzare un componente attivato da un fronte di un segnale di clock:

- Elencare il segnale di clock come ingresso logico nell'interfaccia del componente
- Elencare il segnale di clock nella sensitivity list
- All'interno del costrutto "`process`", utilizzare il costrutto `if...then` per condizionare l'esecuzione
- Le condizioni possono essere meglio specificate con gli *attributi* dei segnali , ad esempio:
  - `clk'event`: vale `True` se il segnale `clk` cambia stato
  - `clk'last_value`: valore prima del cambiamento di stato
  - Esistono vari altri attributi, meno utilizzati



Sincronizzazione con segnale di clock (2)

```
entity Prova is
   port (a :  IN std_logic;
         clk :  IN std_logic;
           z :  OUT std_logic);
end entity;
architecture Behav of Prova is
begin
   process (clk) is
   begin
      -- fronte attivo: salita
      if clk'event and clk = '1' then
         z <= not a;
      end if;
   end process;
end architecture;
```

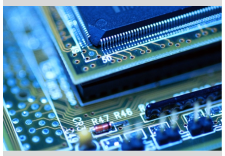



Variabili

- Le *variabili* consentono di memorizzare dati all'interno di un costrutto `process`
- Esempio (tra `process` e `begin`):

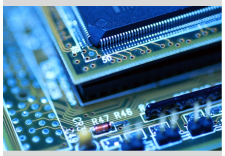
```
variable STATO : integer := -1;
```

- Se non indicato, il valore iniziale dipende dal tipo di dati
  - Per gli scalari, è il valore "all'estrema sinistra" dell'intervallo
- Generalmente le variabili sono "private"
  - Una variabile può essere visibile da un solo costrutto `process`
  - Tipicamente è dichiarata all'interno di un `process`
- Per assegnare un nuovo valore: `STATO := 42;`
- Le modifiche al valore di una variabile sono "immediate"
  - I flip-flop sono modificati mentre si "esegue" l'assegnazione



Variabili (2)

```
process (clk) is
   variable state :  integer := 0;
begin
   if clk'event and clk = '1' then
      case state is
         when 0 =>
           z <= (not a); state := 1;
         when 1 =>
           z <= a;       state := 2;
         when 2 =>
           z <= '0';     state := 0;
         when others =>
           z <= '0';     state := 0;
      end case;
   end if;
end process;
```

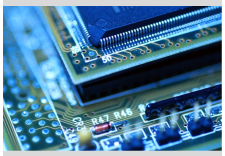



Segnali

- I *segnali* consentono di memorizzare dati all'interno di un costrutto `process`
- Esempio (tra `architecture` e `begin`):

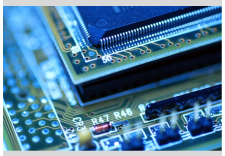
```
signal STATO : integer := -1;
```

- Se il segnale appare soltanto in descrizioni strutturali, è considerato un semplice collegamento senza memoria
- Se invece il segnale appare entro un costrutto `process`, è un elemento di memoria simile ad una variabile, ma
  - I segnali possono essere condivisi tra più processi
  - La modifica del valore di un segnale all'interno di un costrutto `process` avviene solo al termine del costrutto
- Per assegnare un nuovo valore: `STATO <= 42;`



Segnali (2)

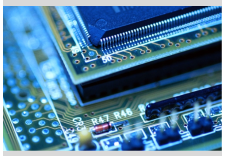
```
architecture Behav of Prova is
   signal state :  integer := 0;
begin
   process (clk) is
   begin
      if clk'event and clk = '1' then
         case state is
           when 0 =>
             z <= (not a); state <= 1;
           when 1 =>
             z <= a;       state <= 2;
           when 2 =>
             z <= '0';     state <= 0;
           when others =>
             z <= '0';     state <= 0;
      :
```

Differenza tra variabili e segnali

```
architecture Behav of Quiz is
   signal S : integer := 0;
begin
   process is
      variable V : integer := 0;
   begin
      S <= 1;
      V := 1;
      x <= S;
      y <= V;
   end process;
end architecture;
```

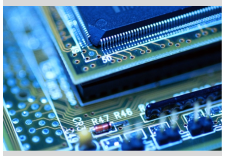
Se all'inizio `S` e `V` sono `0`, che valore hanno `x` e `y` al termine del processo? ***x = 0 y = 1***



Collegamento di componenti

- Nella descrizione strutturale si possono collegare tra loro componenti complessi
  - definiti dall'utente
  - contenuti in librerie
- Si deve ripetere la definizione dell'interfaccia del componente utilizzando il costrutto `component`
- Si deve poi istanziare ciascun componente indicando
  - Il nome dell'istanza
  - Il tipo di componente
  - Come instradare i segnali di ingresso e uscita

```
nome_comp :  tipo_comp port map (sig1 => sig2, ...);
```

Collegamento di componenti (2)

```
entity Esempio is
   port(a :  in STD_LOGIC;
        z :  out STD_LOGIC));
end entity;

architecture Struct of Esempio is
   component BlackBox is
      port(ing :  in STD_LOGIC;
           usc :  out STD_LOGIC));
   end component;
   signal tmp :  STD_LOGIC;
begin
   bb1 :  BlackBox port map (a, tmp);
   bb2 :  BlackBox port map (
             ing => tmp, usc => z);
end architecture;
```