Architetture e reti logiche

Esercitazioni VHDL

a.a. 2006/07

Uso del VHDL

Stefano Ferrari

UNIVERSITÀ DEGLI STUDI DI MILANO
DIPARTIMENTO DI TECNOLOGIE DELL'INFORMAZIONE

Note generali

Il VHDL è un linguaggio *case insensitive*: non fa differenza tra lettere maiscole e minuscole.

Gli identificatori sono nomi iniziano con una lettera e possono avere sia lettere che cifre numeriche nei simboli successivi. Il simbolo *underscore*, "_", è permesso, ma non come primo simbolo di un nome.

I commenti iniziano con "--" e occupano tutta la rimanente parte della riga. I commenti vengono ignorati dal compilatore e possono quindi essere utilizzati per documentare il codice.

La formattazione del codice viene usata per migliorare la leggibilità, ma non modifica il significato dei costrutti coinvolti.

Specifica concettuale

La specifica ad alto livello consiste in una descrizione della funzionalità desiderata, generalmente in una forma discorsiva e incompleta.

Esempio:

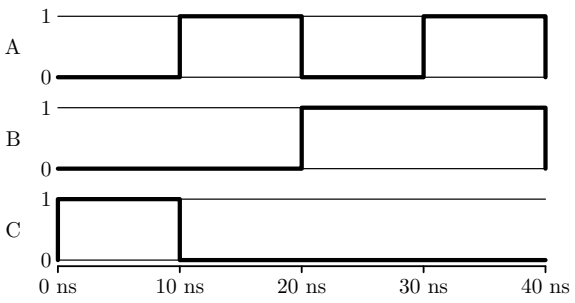
- Creare una entity che riceve in ingresso due segnali digitali e genera in uscita un singolo segnale.
- Se entrambi i segnali di ingresso sono bassi, il segnale di uscita deve essere alto.
- Per ogni altra combinazione degli ingressi, l'uscita deve essere bassa.

Analisi della specifica

Le specifiche vengono analizzate e formalizzate.

Per esempio, la seguente tabella di verità modella il comportamento del componente prima descritto:

Ingressi		Uscita
A	B	C
0	0	1
0	1	0
1	0	0
1	1	0



ma anche un grafico può servire allo scopo.

Entity Declaration

Definisce una entity e la sua interfaccia con l'ambiente esterno (non la funzionalità del modello).

Sintassi:

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

Gli elementi della dichiarazione sono:

- il nome della entity (rappresentato da `ENTITY_NAME`);
- la lista dei segnali di interfaccia (rappresentato da `PORT_LIST`).

Entity Declaration: 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.

Sintassi:

```
port (NAME_LIST: MODE SIG_TYPE;
          ...
      NAME_LIST: MODE SIG_TYPE);
```

- `MODE` identifica la direzione del segnale cioè la direzione del flusso dati (**`in`**, **`out`**, **`inout`**) .
- `SIG_TYPE` identifica il tipo del segnale, cioè l'insieme di valori che può assumere (esempio 0 e 1) e l'insieme di operazioni che possono esservi eseguite.

Architecture Body

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

Sintassi:

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

Architecture Body: esempio

Definizione della architecture NOR_GATE:

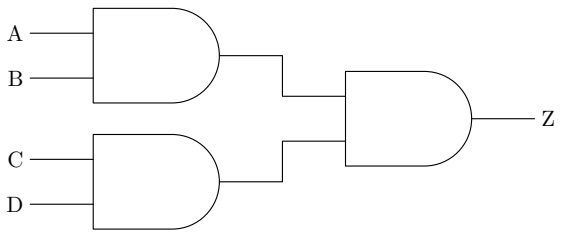
```
architecture DATA_FLOW of NOR_GATE is
begin
   C <= A nor B;
end DATA_FLOW;
```


Concorrenza

Le strutture hardware sono intrinsecamente concorrenti e composte dall'interconnessione di componenti elementari.

Le attività sono svolte in parallelo dai diversi componenti.

Una descrizione strutturale descrive l'interconnessione tra componenti elementari.



Esempio di concorrenza

```
entity AND4 is
   port (A, B, C, D: in bit;
                 Z: out bit);
end AND4;

architecture DATA_FLOW of AND4 is
signal T1, T2: bit;
begin
   T1 <= A and B;
   T2 <= C and D;
   Z <= T1 and T2;
```


Oggetti

Costanti assegnati ad un valore iniziale che non può essere modificato:

```
constant REG_LEN: integer := 16
```

Variabili ricevono un valore che può essere cambiato attraverso un istruzione di assegnamento:

```
variable VAR_A: integer := 3;
VAR_A := NEW_VALUE;
```

Segnali ricevono dei valori associati ad un fattore temporale e che possono essere cambiati attraverso un istruzione di assegnamento tra segnali:

```
signal SIG_A: bit;
SIG_A <= NEW_VALUE after 1 ms;
```

Assegnamento

I segnali sono caratteristici della descrizione hardware per modellare l'informazione che transita nel tempo o su una singola net oppure su un bus.

L'istruzione di assegnamento tra segnali stabilisce un collegamento definitivo tra di essi.

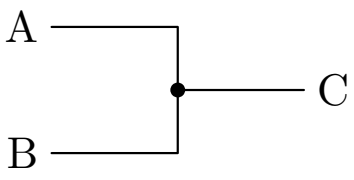
Esempio:

```
signal A, Z : bit;
Z <= A;
```


Assegnamento (2)

In presenza di istruzioni di assegnamento multiple è necessaria una funzione di risoluzione :

```
signal A, B, C : bit;
C <= A;
C <= B;
```



Tipi di dato VHDL

Nel package STD.STANDARD:

- BOOLEAN (`false`, `true`);
- BIT (`'0'`, `'1'`);
- INTEGER (intervallo che dipende dall'implementazione);
- TIME (intervallo che dipende dall'implementazione);
- tipi di dati definiti dall'utente.

Il VHDL è un linguaggio fortemente tipizzato.

L'assegnamento tra segnali di diverso tipo richiede l'uso di appositi convertitori.

Accesso a tipi comunemente usati

La condivisione da parte di diverse entity di dichiarazioni di tipi e di operazioni tra tipi viene fatta attraverso dei costrutti noti come package.

Tipi, procedure e funzioni comunemente usate sono poste in package standard, come Std_Logic_1164 (IEEE standard).

La dichiarazione **`library`** rende visibile ad un modello VHDL una libreria selezionata.

La dichiarazione **`use`** rende un package visibile ad un modello.

Esempio:

```
library IEEE;
use IEEE.Std_Logic_1164.all;
```

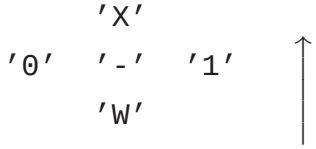
Tipi in IEEE.STD_LOGIC_1164

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','-').`

Tipo STD_(U)LOGIC

`STD_LOGIC` risolve i conflitti (U sta per *unresolved*!).



- `'U'` undefined
- `'X'` forcing unknown
- `'0'` forcing 0
- `'1'` forcing 1
- `'Z'` high impedance
- `'W'` weak unknown
- `'L'` weak 0
- `'H'` weak 1
- `'-'` don't care

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")

Vettori (2)

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);
```

Vettori (3)

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

- Concatenazione:

```
architecture EXAMPLE of CONCATENATION is
   signal BYTE : bit_vector (7 downto 0);
   signal A_BUS, B_BUS : bit_vector (2 downto 0);
   signal C, D : bit;
begin
   BYTE <= A_BUS & C & B_BUS & D;
end EXAMPLE;
```

va usata solo sul lato destro dell'assegnamento;

Vettori (4)

- 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

Gli operatori definiti sui tipi:
`BOOLEAN`, `BIT`, `BIT_VECTOR`, `STD_LOGIC` e `STD_LOGIC_VECTOR`

si dividono in:

- logici: **`AND NAND OR NOR XOR NOT`**;
- relazionali: `= /= < > <= >=`.

Gli operatori relazionali sui vettori sono definiti bit a bit, allineati a destra:

`"111"` > `"1011"` risulta `true`.

Operatori (2)

Gli operatori aritmetici sono definiti sui tipi numerici (e.g, `integer`) e fisici (e.g., `time`:
`=`, `/=`, `<`, `>`, `<=`, `>=`.

Sono inoltre definiti per `STD_LOGIC` e `STD_LOGIC_VECTOR`, ma non sui tipi `BIT`, `BIT_VECTOR`:

```
architecture myarch of esempio is
  signal t : bit_vector(0 to 7);
  signal s : std_logic_vector(0 to 7);
begin
   t <= t + "00000001";  -- errore!!!
   s <= s + "00000001";  -- OK
end myarch;
```

Esercizio 1

Disegnare lo schematico corrispondente alla seguente descrizione VHDL:

```
ENTITY example1 IS
   PORT ( x1, x2, x3 : IN BIT;
                  f : OUT BIT );
END example1;

ARCHITECTURE LogicFunc OF example1 IS
BEGIN
   f <= (x1 AND x2) OR (NOT x2 AND x3);
END LogicFunc;
```


Esercizio 2

Disegnare lo schematico corrispondente alla seguente descrizione VHDL e costruirne la tabella di verità:

```
ENTITY fulladd IS
  PORT (Cin, x, y : IN STD_LOGIC;
          s, Cout : OUT STD_LOGIC );
END fulladd;

ARCHITECTURE LogicFunc OF fulladd IS
BEGIN
   s <= x XOR y XOR Cin;
   Cout <= (x AND y) OR (Cin AND x) OR (Cin AND y);
END LogicFunc;
```

Descrizione strutturale

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

ARCHITECTURE structure of latch IS
   component nor_gate
      port (a,b: in bit;
              c: out bit);
   end component;
   signal t1, t2 : bit;
BEGIN
```

(continua)

Descrizione strutturale (2)

```
ARCHITECTURE structure of latch IS
   component nor_gate
      port (a,b: in bit;
              c: out bit);
   end component;
   signal t1, t2 : bit;
BEGIN
   n1: nor_gate port map (r, t1, t2);
   n2: nor_gate port map (b=>t2, a=>s, c=>t1);
   nq <= t1;
   q <= t2;
END structure;
```

Esercizio 3

```
ENTITY adder3 IS
  PORT (        Cin : IN STD_LOGIC;
         x2, x1, x0 : IN STD_LOGIC;
         y2, y1, y0 : IN STD_LOGIC;
         s2, s1, s0 : OUT STD_LOGIC;
               Cout : OUT STD_LOGIC);
END adder3;

ARCHITECTURE Structure OF adder3 IS
  SIGNAL c1, c2 : STD_LOGIC;
BEGIN
   stage0: fulladd PORT MAP(Cin,x0,y0,s0,c1);
```

(continua)

Esercizio 3 (2)

```
ARCHITECTURE Structure OF adder3 IS
  SIGNAL c1, c2 : STD_LOGIC;
BEGIN
   stage0: fulladd PORT MAP(Cin,x0,y0,s0,c1);
   stage1: fulladd PORT MAP(c1,x1,y1,s1,c2);
   stage2: fulladd PORT MAP(Cin=>c2, Cout=>Cout, x=>x2,
                            y=>y2, s=>s2);
END Structure;
```

Esercizio 3 bis

```
ENTITY adder3 IS
  PORT (Cin : IN STD_LOGIC;
        X, Y : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
           S : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
        Cout : OUT STD_LOGIC ) ;
END adder3;

ARCHITECTURE Structure OF adder3 IS
  SIGNAL C : STD_LOGIC_VECTOR(1 TO 2) ;
BEGIN
  stage0: fulladd PORT MAP (Cin, X(0), Y(0), S(0), C(1));
  stage1: fulladd PORT MAP (C(1), X(1), Y(1), S(1), C(2));
  stage2: fulladd PORT MAP (C(2), X(2), Y(2), S(2), Cout);
END Structure;
```


Costrutti concorrenti

Sono utilizzabili nelle sezioni concorrenti della descrizione di una **`architecture`**.

I costrutti principali sono:

- assegnamento selettivo;
- assegnamento condizionato.

Assegnamento selettivo

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

Il costrutto **WITH**-**SELECT**-**WHEN** è utilizzabile quando:

- le condizioni devono essere mutuamente esclusive;
- tutti i valori possibili dell'espressione devono essere coperti.

Assegnamento selettivo: 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;
```

Assegnamento selettivo: esempio (2)

L'uso di **OTHERS** rende la descrizione più sintetica e leggibile:

```
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

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

Nel costrutto **WHEN-ELSE**:

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

Assegnamento 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;
```


Assegnamento condizionato: esempio (2)

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;
```

Assegnamento condizionato: esempio (3)

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


Processo

Alcune operazioni sono troppo complesse per poter essere descritte da una singola istruzione o costrutto.

I processi:

- sono costrutti che permettono di descrivere un'azione mediante una sequenza di istruzioni;
- interagiscono tra loro in parallelo (concorrenza tra processi);
- si scambiano informazioni tramite i segnali.

L'utilizzo di una variabile viene permesso solo in ambiente sequenziale.

Processo (2)

l processo è il costrutto principale per la modellazione di componenti sequenziali.

Sintassi:

```
[<label>:] process [(<sensitivity_list>)]
           begin
             <seq_instr1>
               ...
             <seq_instrN>
           end process [<label>];
```


Esecuzione di un processo

Ci sono due modi per controllare l'esecuzione di un processo:

- **`wait;`**
- *sensitivity list*.

Il processo deve essere inteso come un'unica istruzione: eventuali variazioni dei segnali che avvengono nei processi prendono corpo solo al termine del processo.

Sensitivity list

La sensitivity list di un processo è un elenco di segnali associati al processo stesso.

Una variazione del valore di un segnale viene definito un evento sul segnale.

Un evento su un segnale indicato nella sensitivity list di un processo, causa l'esecuzione di quest'ultimo.

Esempio

Qual'è la 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;
```

Nella simulazione il primo processo sarà in grado di attivarsi anche su eventi del segnale SEL.

Costrutti sequenziali

I costrutti sequenziali possono essere solo usati in ambito sequenziale, cioè all'interno di un **`process`**.

Essi sono:

- **`if-then-else;`**
- **`case-when;`**
- **`loop`**.

Costrutto `if-then-else`

Sintassi:

```
if <condition>
  then <instruction1>;
  [else <instruction2>;]
end if;
```

```
if <condition1>
  then <instruction1>;
  elsif <condition2>
  then <instruction2>;
  [else <instruction3>;]
end if;
```

L'ordine con cui le condizioni vengono valutate forza la priorità sulle istruzioni eseguite.

Esempio

```
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;
```


Costrutto case-when

Il costrutto **case-when** va utilizzato quando tutte le condizioni sono mutuamente esclusive e hanno la stessa priorità.

```
case <selection_signal> is
   when <value_1_selection_signal> => <instr_1>;
   when <value_2_selection_signal> => <instr_2>;
   ...
   when <value_n_selection_signal> => <instr_n>;
   [when others => <instruction_others>;]
end case;
```

Tutti i possibili valori del segnale di selezione devono essere coperti, eventualmente ricorrendo alla clausola **others**.

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;
```


Costrutto ciclici

Esistono diversi costrutti ciclici:

- **loop;**
- **while-loop;**
- **for-loop;**

Poiché il VDHL descrive hardware, un ciclo descrive una ripetizione di uno schema circuitale.

Costrutto loop

Sintassi:

```
loop
   <instruction>;
   exit when <condition>;
end loop;
```


Costrutto while-loop

Sintassi:

```
while (<condition>) loop
   <instruction>;
end loop;
```

Costrutto `for-loop`

Sintassi:

```
for <var> in <min_value> to <max_value> loop
   <instruction>;
end loop;
```

Oppure:

```
for <var> in <max_value> downto <min_value> loop
   <instruction>;
end loop;
```

Le variabili usate per controllare il numero di iterazioni, devono essere di tipo `integer`.

Temporizzazione

Il costrutto **`after`** causa un ritardo nell'assegnamento:

```
NEW_SIGNAL <= SIGNAL_EXPR after TIME_PERIOD;
```

dove `TIME_PERIOD` deve essere di tipo `time`.

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;
```

Costrutti `wait`

Il costrutto **`wait`** può essere usato per controllare un processo in alternativa alla sensitivity list.

Esistono diverse forme per questo costrutto:

- **`wait on;`**
- **`wait until;`**
- **`wait for.`**

Un processo senza sensitivity list (quindi controllato da **`wait`**) viene mandato in esecuzione subito, e, una volta eseguita l'ultima istruzione, riprende ad essere eseguito, finché non incontra una istruzione **`wait`**.

Esempio

Il costrutto **`wait`** può essere usato per generare segnali periodici:

```
clock_p : process
begin
  clock <= '0';
  wait for 10 ns;
  clock <= '1';
  t <= X;
  wait for 10 ns;
end process;
```

Questo frammento di codice descrive un'onda quadra con periodo 20 ns.

Attributo event

Ogni segnale ha una serie di attributi ad esso associati, denotati dal costrutto:

```
<nome_segnale>'<nome_attributo>
```

Uno dei più usati è l'attributo booleano `event`:

- assume il valore `true` solo nell'istante in cui il segnale associato cambia di valore (si verifica un evento sul segnale).

Può essere utilizzato per individuare un fronte di salita:

- `clock'event` **and** `clock='1'` assume il valore `true` solo in occasione del fronte di salita (`clock` cambia valore ed assume il valore `'1'`).