

Sintassi del linguaggio VHDL

Linguaggi di descrizione dell'hardware

M. Favalli



DE Department of
Engineering
Ferrara

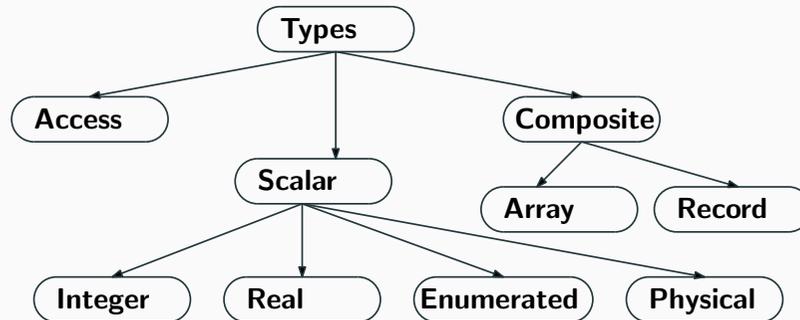
- La sintassi del VHDL ha molti punti di contatto con linguaggi software (deriva dall'ADA) e può essere studiata in maniera simile
 - Tipi di dato
 - Oggetti
 - Istruzioni
- Vedremo poi il modo in cui il VHDL gestisce le librerie

Sommario

- 1 Tipi di dato
- 2 Oggetti VHDL
- 3 Istruzioni
- 4 Package e librerie
- 5 Attributi
- 6 Operatori

Tipi di dato

- Nel VHDL tutte le dichiarazioni di porte, segnali e variabili devono avere associato un tipo di dato
- Oltre a mettere a disposizione tipi di dato predefiniti, il VHDL consente all'utente di definire i propri tipi di dato



- Range minimo (standard): da -2,147,483,647 a 2,147,483,647
- Esempio di assegnamento a variabili (diverse dai segnali)

```

architecture test_int of test is
begin
  process(x)
    variable a: integer;
  begin
    a:=1;
    a:=-1;
    a:=1.0; -- illegal
  end process;
end architecture test_int;
    
```

- Differenza con il C: il VHDL é strongly typed e gli assegnamenti irregolari vengono evidenziati durante l'analisi dei modelli

- Range minimo (standard): da -1.0E38 a 1.0E38
- Esempio di assegnamento

```

architecture test_real of test is
begin
  process(x)
    variable a: real;
  begin
    a:=1.3;
    a:=-7.5;
    a:=1; -- illegal
    a:=1.7E13;
    a:=5.3 ns; -- illegal
  end process;
end architecture test_real;
    
```

- Tipi enumerati predefiniti nel package standard: **bit** (e **bit_vector**), **boolean** e **character**
- Esempio di dichiarazione di tipo e assegnamento

```

architecture test_enum of test is
  type binary is (on,off); -- type declaration
begin
  process(x)
    variable a: binary; -- use of the type
  begin
    a:=on;
    a:=off;
  end process;
end architecture test_enum;
    
```

- Richiede un unità di misura
- Si deve specificare un range
- Esempio di dichiarazione

```
type resistance is range 0 to 10000000;
units
ohm;
kohm = 1000 ohm;
mohm = 1000 kohm;
end units
```

- Il tempo (time) é l'unico tipo fisico predefinito nel VHDL

- Utilizzato per raggruppare elementi dello stesso tipo in un singolo oggetto VHDL
- Il range puó essere unconstrained della dichiarazione, ma deve essere specificato quando si usa l'array
- Esempio di dichiarazione per un array monodimensionale (vettore)

```
type data_bus is array (0 to 31) of bit;
```

0	1	... element indices ...	31
0	0	... array values ...	1

Esempi di utilizzo

- Accesso ai singoli elementi di un array

```
variable x: data_bus;
variable y: bit;
....
y:=x(12); -- y get the value of the element at index 12
x(8):=y; -- the element at index 8 is assigned to y
```

- Confronto e assegnamento

```
signal x,y,w: array (0 to 7) of bit;
....
if (x=y) then
  w <= "11001011"; -- bit is seen as a character
  w(6) <= '0'; -- and bit array as a string
end if;
```

Esempi di utilizzo

- Accesso a sotto vettori e concatenamento

```
variable x, y: array (0 to 15) of bit;
variable w: array (0 to 31) of bit;
....
x:=w(0 to 15);
y:=w(16 to 31);
w:=y & x;
```

- I tipi di dato `bit_vector` e `std_logic_vector` sono essenzialmente stringhe per cui le costanti sono nella forma "1000", le costanti di tipo vettore per altri tipi di dato hanno un diverso formato

```
variable x: array (0 to 3) of integer;
...
x <= (6, 7, 8, 4);
```

- Quando un array porta informazioni numeriche può essere conveniente avere a sinistra il bit di maggior peso
- Questo può essere fatto utilizzando `downto` nella definizione del range

```
type reg_type is array (15 downto 0) of bit;
```

15	14	... element indices ...	0
0	0	... array values ...	1

- La keyword `downto` va sempre utilizzata quando l'indice di sinistra è più grande di quello di destra
- Per l'utilizzo di questo tipo di array valgono le considerazioni viste in precedenza

- Utilizzati per raggruppare elementi di tipi differenti in un singolo oggetto VHDL
- Gli elementi sono referenziati tramite i nomi dei campi
- Esempio di dichiarazione e utilizzo

```
type binary is (on, off);
type switch_info is record
    status: binary;
    idnumber: integer;
end record;
....
variable switch: switch_info;
switch.status:=on;
switch.idnumber:=39;
```

- Simile ai puntatori in altri linguaggi
- Memorizzazione dinamica
- Utile per realizzare a un livello astratto vari tipi di code, etc. emulando eventualmente programmi software
- Viene utilizzato nel package TextIO
- Può essere assegnato esclusivamente a variabili
- Una volta dichiarata una variabile di questo tipo, si rendono disponibili due funzioni `new` e `deallocate` che consentono di allocare e deallocare memoria per tale variabile

```
process (x)
    type fifo_el_type is array (0 to 3) of std_logic;
    type fifo_el_access is access fifo_el_type;
    variable fifo_ptr : fifo_el_access := NULL;
    variable temp_ptr : fifo_el_access := NULL;
begin
    temp_ptr:=new fifo_el_type;
    temp_ptr.ALL=('0','1','0','1');
    fifo_ptr:=temp_ptr;
end;
```

- A partire da un tipo di dato unconstrained si può definire un subtype che definisce dei vincoli sul tipo di dato di partenza
- Il range può includere tutto il range del tipo di partenza
- Assegnamenti fuori dal range sono illegali e vengono rivelati run time, mentre alla compilazione si verificano solo i tipi
- Sintassi della dichiarazione

```
subtype name is base_type range <user range>;
```

- Esempio

```
subtype first_ten is integer range 0 to 9;
```

- A cosa servono?
- Supponiamo di voler realizzare una descrizione behavioral di un'architettura che elabora dei numeri naturali senza voler entrare nei dettagli dell'implementazione
- Sappiamo comunque che non useremo più di 12 bit per rappresentare l'informazione
- Possiamo usare gli interi, ma che accade se per esempio si genera un numero più grande di $2^{12} - 1$ o negativo e non ce ne accorgiamo durante la simulazione
- Si può allora definire un sottotipo

```
subtype natural_data is integer range 0 to 4095;
```

- In caso di violazioni di range il simulatore ci avvisa

Sommario

- **Tutte le dichiarazioni di porte, segnali e variabili VHDL devono includere il loro tipo e sottotipo**
- Il VHDL mette a disposizione tipi di dato utilizzabili
 - nella descrizione di sistemi digitali al livello logico gate
 - nella descrizione di tali sistemi a livelli di astrazione più alti
 - per realizzare un qualsiasi algoritmo (il VHDL è funzionalmente completo)
- L'utilizzatore può aggiungere altri tipi di dato a quelli predefiniti dal VHDL
 - enti di standardizzazione, produttori di tool di EDA e di IC ne hanno sviluppati di rilevanti

Oggetti VHDL

- Nel VHDL sono disponibili 4 tipi di oggetti
 - **costanti**
 - **variabili**
 - **segnali**
 - **file**
- Lo scope di un oggetto nel VHDL é il seguente:
 - gli oggetti definiti in un package VHDL sono disponibili nelle descrizioni che lo usano
 - gli oggetti dichiarati in una entity sono disponibili a tutte le architetture associate con quella entity
 - gli oggetti dichiarati in un architettura sono disponibili a tutte le istruzioni in tale architettura
 - gli oggetti dichiarati in un processo sono disponibili solo in tale processo

- Nome assegnato a un valore specifico di un tipo
- Aggiornamento e leggibilità
- La dichiarazione di una costante può omettere il suo valore così da differirne l'assegnamento per rendere possibile una riconfigurazione
- Sintassi della dichiarazione

```
constant constant_name: type_name [:=value];
```

- Esempio

```
constant pi: real:=3.14;  
constant speed: integer;
```

- Meccanismo per la **memorizzazione locale all'interno dei processi**
 - contatori, valori intermedi
- Lo scope é il processo in cui sono dichiarate
 - Il VHDL-93 mette a disposizione variabili globali
- Tutti gli assegnamenti a variabili hanno luogo immediatamente senza nessun ritardo delta o specificato dall'utilizzatore
- Sintassi della dichiarazione

```
variable variable_name : type_name [:=value];
```

- Esempi

```
variable opcode: bit_vector(3 downto 0) := "0000";  
constant freq: integer;
```

- **Comunicazione fra processi**
- I segnali reali dei sistemi sono mappati sui segnali
- Tutti gli assegnamenti ai segnali avvengono con un ritardo delta o specificato dall'utente
- Sintassi della dichiarazione

```
signal signal_name : type_name [:=value];
```

- Esempio

```
signal b: bit;  
b <= '0' after 5 ns, '1' after 10 ns;
```

Specifica una waveform, con una lista di assegnamenti ciascuno dei quali corrisponde a un valore e un ritardo con cui questo viene attuato rispetto all'istante in cui viene eseguito il processo

- Il valore iniziale delle variabili é riferito alla prima volta in cui la simulazione esegue il processo
 - errore: usare il valore iniziale di una variabile per assegnarle un certo valore ogni volta che il processo viene eseguito
- Il valore iniziale dei segnali é riferito all'istante (t_0^-) precedente all'inizio della simulazione
 - 'U' nel tipo di dato `std_logic`

Segnali

```
architecture tests of test is
  signal x: bit:= '1';
  signal y: bit:= '0';
begin
  process(in_sig, x, y)
  begin
    x <= in_sig xor y;
    y <= in_sig xor x;
  end process;
end architecture tests;
```

Variabili

```
architecture testv of test is
  signal y: bit:= '0';
begin
  process(in_sig, y)
  variable x: bit:= '1';
  begin
    x := in_sig xor y;
    y <= in_sig xor x;
  end process;
end architecture tests;
```

Assumendo una transizione da 1 a 0 di `in_sig` quali sono i valori risultanti di `y` in entrambi i casi?

Differenza fra segnali e variabili

Architettura `tests`: la simulazione da luogo a un'oscillazione

time	in_sig	x	y	
0^-	1	1	0	
t_0	0	1	0	eval. proc. → events
$t_0 + \delta$	0	0	1	eval. proc. → events
$t_0 + 2\delta$	0	1	0	eval. proc. → events
$t_0 + 3\delta$	0	0	1

Architettura `testv`: la rete si stabilizza dopo la prima valutazione del processo

time	in_sig	x	y	
0^-	1	1	0	
t_0	0	0	0	eval. proc. → no event

Differenza fra segnali e variabili

La differenza chiave sta nel ritardo degli assegnamenti dei segnali, mentre quello delle variabili é istantaneo

```
architecture signal of test is
  signal out_1: bit; -- out_2 is a PO
begin
  process(a, b, c, out_1)
  begin
    out_1 <= a nand b;
    out_2 <= out_1 xor c;
  end process;
end architecture signal;
```

time	a	b	c	out_1	out_2
0	0	1	1	1	0
t_0	1	1	1	1	0
$t_0 + \delta$	1	1	1	0	0
$t_0 + 2\delta$	1	1	1	0	1

Non appena a cambia (t_0), il processo viene valutato e `out_3` assume immediatamente il nuovo valore che da luogo a un cambiamento di `out_4` programmato a $t_0 + \delta$

```
architecture var of test is
begin
  process (a, b, c)
  variable out_3: bit;
  begin
    out_3 := a nand b;
    out_4 <= out_3 xor c;
  end process;
end architecture var;
```

time	a	b	c	out_3	out_4
0	0	1	1	1	0
t_0	1	1	1	0	0
$t_0 + \delta$	1	1	1	0	1

- **A livello di architettura si possono utilizzare solo i segnali**
- Dentro ai processi possiamo scegliere tenendo conto
 - delle diverse caratteristiche di segnali e variabili nel modello timing
 - che l'assegnamento a una variabile per il simulatore é computazionalmente meno oneroso di quello a un segnale
- Tipicamente si utilizzano variabili per i calcoli intermedi, con i segnali assegnati alla fine del processo
- Variabili globali

File

- I file danno modo a un progetto VHDL di comunicare con l'ambiente esterno
- Dichiarazioni di file simili a quelle di linguaggi sw
- I file possono essere aperti sia in lettura che in scrittura
 - Nel VHDL87, i file sono aperti e chiusi quando gli oggetti associati entrano o escono dallo scope corrente (es. un file associato a un processo si apre tutte le volte che il processo viene valutato)
 - Nel VHDL93 sono state aggiunte procedure esplicite `file_open()` e `file_close()`
- Il package `standard` definisce le routine base di I/O per i tipi VHDL
- Il package `textio` definisce routine piú potenti per la gestione di file di testo (importanti estensioni nel VHDL-2008)

Istruzioni

- **Istruzioni sequenziali vs istruzioni concorrenti**
- Il VHDL é inerentemente un linguaggio concorrente
 - tutti i processi VHDL sono eseguiti concorrentemente
 - gli assegnamenti concorrenti dei segnali sono processi on-line
- **Le istruzioni VHDL sono eseguite sequenzialmente all'interno di un processo**
- Processi concorrenti con esecuzione sequenziale danno la massima flessibilit 
 - supportano vari livelli di astrazione
 - supportano la modellistica di eventi concorrenti e sequenziali come si osserva nei sistemi reali

- La granularit  base della concorrenza e' il processo
- **Meccanismo per avere la concorrenza (a tempo di simulazione):**
 - i processi comunicano fra di loro mediante i segnali
 - gli assegnamenti di segnali richiedono un ritardo prima che si assuma un nuovo valore
 - il tempo di simulazione avanza quando tutti i processi attivi si completano
 - **l'effetto e' il processing concorrente in cui l'ordine in cui i processi sono effettivamente eseguiti dal simulatore non cambia il comportamento**
- Istruzioni VHDL concorrenti : block, process, assert, signal assignment, procedure call, component instantiation

Istruzioni sequenziali

- Le istruzioni dentro un **process** vengono eseguite sequenzialmente
- I modelli sequenziali sono utili per rappresentare il comportamento di un sistema digitale a livelli di astrazione pi  alti di quello gate
- Quindi all'interno di un processo sono disponibili i costrutti di controllo di un qualsiasi linguaggio sw
- **Ad esempio, la rappresentazione VHDL del grafo di transizione dello stato (STG) di una macchina a stati finiti (FSM) é un modello sequenziale**
- **La sua realizzazione come rete sincrona descritta a livello gate é concorrente**

Istruzioni sequenziali: multiplexer parallelo a 2 vie

```

library ieee;
use ieee.std_logic_1164.all;
entity mux is
  port (x,y: in std_logic_vector(3 downto 0);
        s: in std_logic;
        z: out std_logic_vector(3 downto 0));
end entity mux;

architecture behav of mux is
begin
  process (x, y, s)
  begin
    if (s='0') then
      z <= x;
    elsif (s='1') then
      z <= 'y';
    else
      z <= "XXXX"; -- in case s is non boolean
    end if;
  end process;
end architecture behav;

```

Package e librerie

- I costrutti dichiarati dentro architetture e entities non sono visibili ad altri componenti VHDL (per le regole sullo scope)
- Package e librerie danno la capacità di riutilizzare gli stessi costrutti in più modelli VHDL
 - gli oggetti dichiarati in un package possono essere utilizzati in tutti i modelli che fanno riferimento a quel package
 - esempi di package: `std` che è sempre visibile, `ieee.std_logic_1164` IEEE standard 1164

Packages

Esempio di dichiarazione di package

- I package VHDL consistono di due parti
 - **dichiarazione**: contiene le dichiarazioni degli oggetti definiti nel package
 - **body**: contiene le definizioni necessarie per gli oggetti nella dichiarazione (sottoprogrammi ad esempio)
- Esempi di oggetti VHDL contenuti in un package
 - tipi e sottotipi
 - costanti
 - sottoprogrammi
 - clausole di utilizzo
 - segnali
 - attributi
 - componenti

```
package complex_numbers is
  constant pi: real:=3.14;
  constant eu: real:=2.71;
  type cmpx is record
    real_p: real;
    imm_p: real;
  end record;
  function product (x: in cmpx, y: in cmpx) return cmpx;
  function module (x: in cmpx) return real;
end package complex_numbers;
```

Alcuni oggetti richiedono solo una dichiarazione, altri necessitano di dettagli da specificare nel body (es. procedure e funzioni)

Il body include le descrizioni funzionali degli oggetti dichiarati

```
package body complex_numbers is
function product (x: in cmpx, y: in cmpx) return cmpx is
variable p: cmpx;
begin
  p.real_p:=x.real_p*y_real_p-x_imm.p*y_imm_p;
  p.imm_p:=x.real_p*y_imm_p+x.imm_p*y.real_p;
  return p;
end function product;
function module (x: in cmpx, y: in cmpx) return real is
variable m: real;
begin
  m:=x.real_p**2+x_imm.p**2;
  return m;
end function module;
end package complex_numbers;
```

- I package devono essere resi visibili per poterne utilizzare il contenuto
- La clausola `use` rende i package visibili ad architetture, entities e altri package
- **Utilizzo selettivo di parti di un package**

```
use complex_numbers.pi, complex_numbers.cmpx;
.... entity ....
.... architecture ....
```

- **Utilizzo di tutto il package**

```
use complex_numbers.all;
.... entity ....
.... architecture ....
```

Librerie

- Simili a direttori di file
- Le librerie VHDL contengono entities, architetture, e packages che sono stati analizzati (i.e. compilati)
- Facilitano l'amministrazione di progetti complessi
 - librerie di progetti esistenti
- Le librerie sono accessibili mediante un nome logico
 - Il progetto corrente viene compilato dentro la libreria `work`
 - `work` e `std` sono sempre disponibili
- Molte librerie sono fornite dai venditori di tool EDA o di moduli IP
 - librerie proprietarie e IEEE standard

Attributi

- Gli attributi VHDL mettono a disposizione informazioni su certi oggetti VHDL
 - tipi e sottotipi
 - procedure e funzioni
 - segnali, variabili e costanti
 - entity, architetture
 - configurazioni e componenti
 - package
- Forma generale

```
name' attribute_identifier
```

- Il VHDL ha diversi tipi di attributi predefiniti di cui qui vengono forniti alcuni esempi
 - **x'event** - ritorna TRUE quando c'è un evento sul segnale **x**
 - **x'last_value** - ritorna il valore precedente di **x**
 - **y'high** - ritorna il valore più alto nel range di **y** dove **y** è un oggetto VHDL o un tipo di dato
 - **x'stable(τ)** - ritorna TRUE quando nessun evento è avvenuto sul segnale **x** dall'istante corrente τ fino all'istante $\tau - \tau$

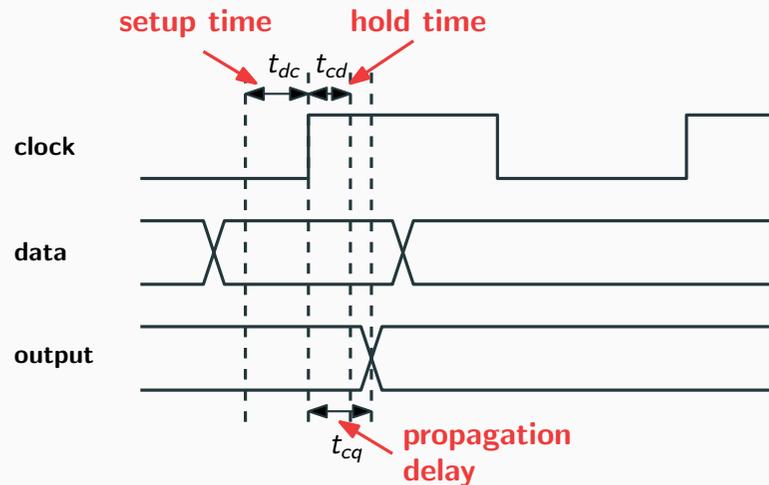
Attributi: esempio di registro

- **Utilizzo degli attributi per descrivere al livello comportamentale un registro a 8 bit**
- Specifiche:
 - campionamento sul fronte di salita del clock
 - memorizzazione solo se il segnale di enable è alto
 - tempo di setup: t_{dc}
 - tempo di risposta: t_{cq}
- Come tipo di segnale si utilizzerà **std_logic**
- Il modello è principalmente orientato alla simulazione e consente di determinare eventuali problemi nel timing del circuito

Parametri timing dei flip-flop

- Il propagation delay o tempo di risposta t_{cq} è il tempo che passa dal fronte di campionamento al cambiamento dell'uscita
- Il dato deve poi rimanere stabile da un certo tempo prima (tempo di setup, t_{dc}) del fronte di campionamento a un certo tempo dopo (tempo di hold, t_{cd})
 - se questi vincoli non sono rispettati, il flip-flop può campionare un valore errato o entrare in uno stato detto di metastabilità

Flip-flop positive edge-triggered



```
entity 8_bit_reg is
    generic (tdc, tcq : time);
    port(enable, clk : in std_logic;
          d : in std_logic_vector(7 downto 0);
          q : out std_logic_vector(7 downto 0));
end entity 8_bit_reg;
```

Si noti che la dimensione del registro può essere resa parametrica

```
entity reg is
    generic (n: integer;
            tdc, tcq : time);
    port(enable, clk : in std_logic;
          d : in std_logic_vector(n-1 downto 0);
          q : out std_logic_vector(n-1 downto 0));
end entity reg;
```

Registro: architecture primo tentativo

Analisi

- Rilevazione di un fronte di salita su `clk`
- Utilizzo dell'attributo `stable` per verificare il tempo di setup

```
architecture first_attempt of 8_bit_reg is
begin
    proces(clk)
    begin
        if (enable='1') and (clk='1') then
            if (d'stable(tdc)) then
                q <= d after tcq;
            else
                q <= (others => 'X') after tcq;
            end if;
        end if;
    end process;
end first_attempt;
```

- Cosa accade se teniamo conto anche del valore 'X' nel tipo di logica utilizzata?
 - `clk='1'` funzionerebbe bene per il tipo di dato `bit` che ha solo i valori 0 e 1
 - nel modello indicato una transizione $X \rightarrow 1$ del clock risulterebbe nel campionamento del dato
 - se il segnale di enable ha il valore X il modello si comporta come se avesse il valore 0
- Il modello non fa alcuna verifica sul segnale di enable che potrebbe dare luogo a violazioni del tempo di setup

```

architecture behav of 8_bit_reg is
begin
  proces (clk)
  begin
    if (enable='1') and (clk='1')
      and (clk'last_value='0') then
      if (d'stable(tdc) and enable'stable(tdc)) then
        q <= d after tcq;
      else
        q <= (others => 'X') after tcq;
      end if;
    elsif (enable='X') or ((clk='1') and
      (clk'last_value='X')) then
      q <= (others => 'X') after tcq;
    end if;
  end process;
end behav;

```

- Il comando `d'stable(tdc)` ritorna un valore falso se anche un solo segnale su 8 viola il tempo di setup
- Quindi mettere a `X` tutte le uscite del registro é pessimistico
- Questo atteggiamento di solito é quello seguito dai progettisti
- Si noti che il package `std_logic_1164` mette a disposizione una funzione che individua i fronti di salita `rising_edge(clk)`
 - infatti il tipo di dato `std_logic` ha diversi valori ('H' e 'L' da considerare come 0 e 1, 'Z' che nella maggior parte dei casi corrisponde a X)
- La sintesi ignora le parti riguardanti il tempo di setup e la gestione dei valori a X, limitandosi a riconoscere la presenza di 8 FF edge triggered e a instanziarli

Operatori

Operatori

- Il VHDL consente di descrivere espressioni complesse
- Livelli di precedenza in ordine decrescente
 - operatori vari: `**`, `abs`, `not`
 - operatori moltiplicativi: `*`, `/`, `mod`, `rem`
 - operatori di segno: `+`, `-`
 - operatori di addizione: `+`, `-`, `&`
 - operatori di shift: `sll`, `srl`, `sla`, `sra`, `rol`, `ror`
 - operatori relazionali: `=`, `/=`, `<`, `<=`, `>`, `>=`
 - operatori logici: `and`, `or`, `nand`, `nor`
- In caso di dubbi, usare le parentesi
- Ogni operatore é predefinito per un certo tipo di dato, ma può essere esteso (overloaded)
- Nelle espressioni non é possibile la promozione di oggetti VHDL

L'operatore di concatenamento &

```
variable shifted, shiftin: bit_vector(0 to 3);
....
shifted:=shiftin(1 to 3) & '0';
```

	0	1	2	3
shiftin	1	0	0	1
shifted	0	0	1	0

↙ ↙ ↙

L'operatore esponente **

```
variable x: integer;
variable y: real;
....
x:=5**5; -- 5^5 OK
y:= 0.5^3 -- 0.5^3 OK
x:=4**0.5 -- 4^0.5, illegal
y:=0.5**(-2) -- 0.5^-2 OK
```

Conclusioni

- Il linguaggio VHDL supporta la concorrenza al livello di architettura
- Nei processi il modello di esecuzione del linguaggio é sequenziale
 - le strutture di controllo del VHDL nei processi sono simili a quella dei linguaggi di programmazione di tipo imperativo
- La sintassi supporta sia il modeling dell'hardware, sia quello di algoritmi generici
- Package e librerie supportano la manutenzione e il riutilizzo del software