


VHDL made easy!



D. Pellerin D. Taylor
Prentice-Hall

Introduzione al VHDL

A.A. 2006/07 1 VHDL 2

Cos'è il VHDL

VHDL: linguaggio per descrivere sistemi digitali

Usi:

- per la modellazione e la simulazione
- per il design entry per la sintesi automatica
- per la verifica

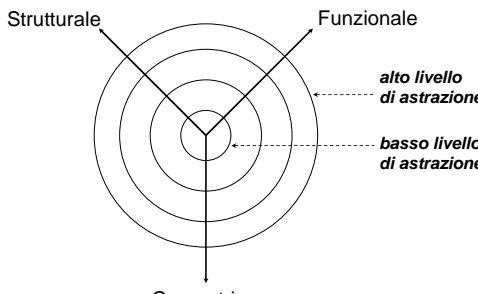
A.A. 2006/07 1 VHDL 3

Scopi

- rendere più affidabile il processo di progetto, minimizzando costi e tempi
- evitare gli errori di progetto
- incremento di produttività usando una metodologia di progetto top-down strutturata.

A.A. 2006/07 1 VHDL 4

Domini e Livelli di Modellazione



Strutturale Funzionale

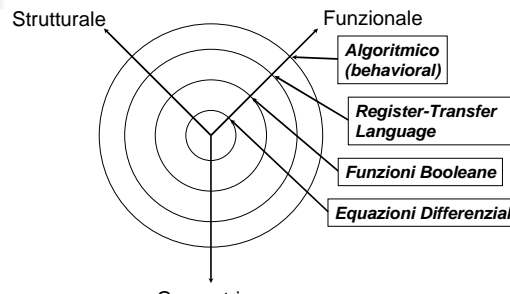
alto livello di astrazione

basso livello di astrazione

Geometrico

"Y-chart" di Gajski & Kahn

A.A. 2006/07 1 VHDL 5



Strutturale Funzionale

Algoritmico (behavioral)

Register-Transfer Language

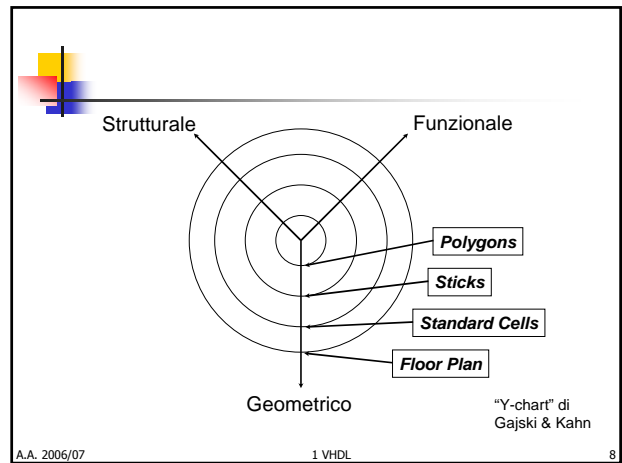
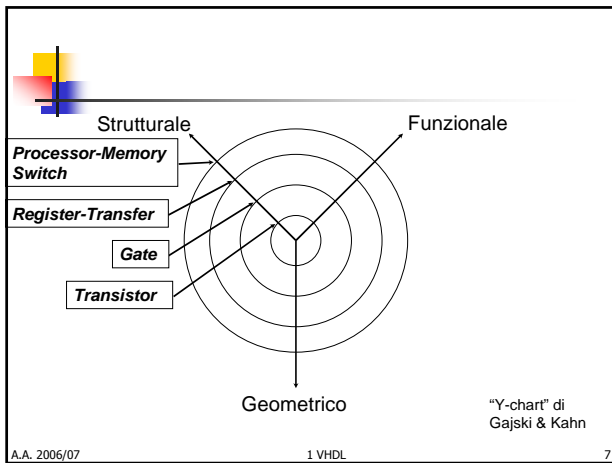
Funzioni Booleane

Equazioni Differenziali

Geometrico

"Y-chart" di Gajski & Kahn

A.A. 2006/07 1 VHDL 6



Breve storia del VHDL

VHDL: VHSIC Hardware Description Language (VHSIC = Very High Speed Integrated Circuit)

- progetto DoD 1983
- dal 1987 IEEE standard 1076
- standard 1164: package per logica a 9 valori
- versioni 1993 e successive integrazioni.

Usi del VHDL

- Design specification: progetto ad alto livello, descrizione di prestazioni e interfacce di componenti complesse
- Design entry: input dei dettagli di un progetto in un sistema CAD, descrizioni schematiche o testuali
- Design simulation: simulazione per verificare funzionamento e tempistica (test bench applicato al modello)
- Design documentation: documentazione di sistemi complessi

Struttura semplice di un file VHDL

Design entity:

- entity declaration
- architecture body

A diagram showing a large box labeled 'Design Entity' containing two smaller boxes: 'Entity Declaration' and 'Architecture Body'.

Esempio

Comparatore a 8 bit

```
entity compare is
  port (A, B: in bit_vector(0 to 7);
        EQ: out bit);
end compare;
```

```
architecture compare1 of compare is
  begin
    EQ <= '1' when (A = B) else 0;
  end compare1;
```

Entity

Definisce la visione dall'esterno di un componente

- nome
- ingressi/uscite (port) e loro tipi
- parametri (generic)

A.A. 2006/07 1 VHDL 13

```
entity compare is
  port (A, B: in bit_vector(0 to 7);
        EQ: out bit);
end compare;
```

A.A. 2006/07 1 VHDL 14

```
entity NAME_OF_ENTITY is [ generic generic_declarations;]
  port (signal_names: mode type;
        signal_names: mode type;
        :
        signal_names: mode type);
end [NAME_OF_ENTITY];
```

A.A. 2006/07 1 VHDL 15

Mode:

- in
- out
- buffer
- inout

Type:

- bit
- bit_vector
- std_logic
- boolean
- integer
- real
- character
- time

A.A. 2006/07 1 VHDL 16

Standard logic:

- standard IEEE 1164 (fine anni '80)
- valori per la simulazione: 'U' (uninitialized), 'X' (unknown), 'Z' (alta impedenza), 'W' (weak unknown), 'L' (weak 0), 'H' (weak 1), '-' (don't care), 0, 1
- package std_logic_1164: contiene I tipi enumerati std_logic e std_ulogic

A.A. 2006/07 1 VHDL 17

std_logic vs std_ulogic:

- u sta per unresolved signal: è illegale che due valori siano contemporaneamente forzati su un signal di tipo std_ulogic
- std_logic: esistono funzioni di risoluzione che definiscono cosa succede quando si forzano contemporaneamente due valori su un segnale

A.A. 2006/07 1 VHDL 18

Esempi

```
entity fulladder is
  port (X: in bit;
        Y : in bit;
        Cin: in bit;
        Cout: out bit;
        Sum: out bit);
end fulladder;
```

```
entity BUZZER is
  port (DOOR, IGNITION, SBELT: in std_logic;
        WARNING: out std_logic);
end BUZZER;
```

A.A. 2006/07 1 VHDL 19

Architecture

- Descrive il comportamento o la realizzazione interna di una entity
- Possono esistere più architecture per la stessa entity
- Ogni architecture deve essere collegata ad una entity

A.A. 2006/07 1 VHDL 20

Esempi

```
architecture FA1 of fulladder is
begin
  Sum <= X xor Y xor Cin;
  Cout <= (X and Y) or (X and Cin) or (Y and Cin);
end FA1;
```

```
architecture buzzer1 of BUZZER is
begin
  WARNING <= (not DOOR and IGNITION) or (not SBELT and IGNITION);
end buzzer1;
```

A.A. 2006/07 1 VHDL 21

```
architecture architecture_name of NAME_OF_ENTITY is
-- Declarations
-- components declarations
-- signal declarations
-- constant declarations
-- function declarations
-- procedure declarations
-- type declarations
:
begin
-- statements
:
end architecture_name;
```

A.A. 2006/07 1 VHDL 22

Tipi di dati

Tipo	Valori	Esempio
Bit	'1', '0'	Q <= '1';
Bit_vector	(array of bits)	DataOut <= '1010';
Boolean	True, False	EQ <= True;
Integer	-2, 0, 4	Count <= Count + 2;
Real	1.0, -1.0E5	V1 = V2 / 5.3
Time	1ua, 7 ns	Q <= '1' after 6 ns;
Character	'a'	CharData <= 'X';
String	(array of char)	Msg <= "MEM"&Addr

A.A. 2006/07 1 VHDL 23

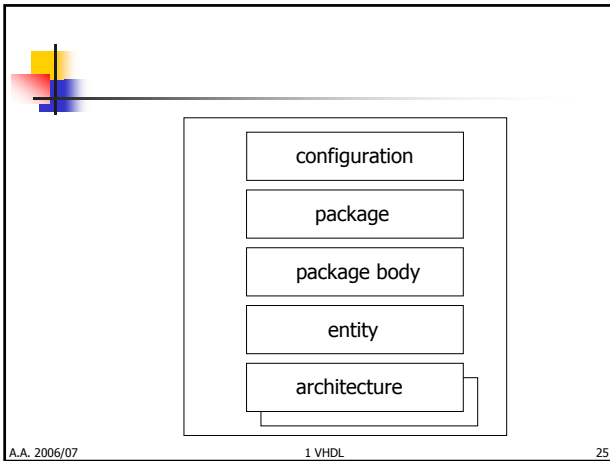
Design units

Segmenti di codice VHDL che possono essere compilati separatamente e memorizzati in una libreria.

Esistono 5 tipi di design units:

- entity } **obbligatorie**
- architecture }
- package }
- package body } **opzionali**
- configuration }

A.A. 2006/07 1 VHDL 24



A.A. 2006/07

1 VHDL

25

Package and Package body

Package: raggruppa dichiarazioni comuni usate da varie design unit.

Consta di:

- package declaration (con dichiarazioni di tipi, sottotipi, costanti, segnali globali, funzioni e procedure, attributi, file, componenti, etc.)
- package body: contiene le implementazioni delle funzioni, procedure, etc.

```
library ieee;
use ieee.std_logic_1164.all;
```

A.A. 2006/07

1 VHDL

26

Configuration

- Associa a una entity una delle architecture che la realizzano

```
entity ALU is
port ( opcode: ...
end entity ALU;

architecture FIRST of ALU is
component COMPARE is
port ( A, B: in bit; C: out bit);
end component COMPARE;
...
begin
IO: COMPARE port map (S, D, Q);
...
end architecture FIRST;
```

A.A. 2006/07

1 VHDL

27

```
configuration FAST_ALU of ALU is
for FIRST
for IO: COMPARE use entity WORK.COMPARE(ARCH_BEHAV);
...
end configuration FAST_ALU;
```

A.A. 2006/07

1 VHDL

28

Stili (Livelli di astrazione)

behavior	Performance Specifications Test Benches Sequential Descriptions State Machines
dataflow	Register Transfers Selected Assignments Arithmetic Operations
structure	Boolean Equations Hierarchy Physical Informations

↑
Livello di astrazione

A.A. 2006/07

1 VHDL

29

Stili (Livelli di astrazione)

- Descrizione *behavior*: comportamento nel tempo come algoritmo, diagramma degli stati, timing diagram
- contiene
 - istruzioni *process*, ciascuna delle quali contiene
 - istruzioni sequenziali, che includono istruzioni di assegnazione ai segnali
 - istruzioni *wait*.

A.A. 2006/07

1 VHDL

30

Esempi

Comparatore

```
architecture ARCH_BEHAV of COMPARE is
begin
  P1: process (A, B)
  begin
    if (A = B) then
      C <= '1' after 1 ns;
    else
      C <= '0' after 2 ns;
    end if;
  end process;
end ARCH_BEHAV;
```

A.A. 2006/07 1 VHDL 31

Esempi

Registro a 4 bit

```
architecture behavior of reg4 is
begin
  storage : process is
  variable stored_d0, stored_d1, stored_d2, stored_d3 : bit;
  begin
    if en = '1' and clk = '1' then
      stored_d0 := d0;
      stored_d1 := d1;
      stored_d2 := d2;
      stored_d3 := d3;
    end if;
    q0 <= stored_d0 after 5 ns;
    q1 <= stored_d1 after 5 ns;
    q2 <= stored_d2 after 5 ns;
    q3 <= stored_d3 after 5 ns;
    wait on d0, d1, d2, d3, en, clk;
  end process;
end architecture behavior;
```

A.A. 2006/07 1 VHDL 32

FSM come diagramma degli stati

```

graph TD
    START((START)) -- rdy_in --> RDY((RDY))
    RDY -- strobe --> START
    RDY -- rdy_in --> POLL((POLL))
    POLL -- rdy_in --> RDY
    POLL -- rdy_in --> SEND((SEND))
    SEND -- rdy_in --> POLL
    SEND -- strobe --> RDY
  
```

A.A. 2006/07 1 VHDL 33

```
...
FSM: process (rst, CLK)
begin
  if rst='1' then
    current_state <= START;
  elsif rising_edge(CLK) then
    case current_state is
      when START => current_state <= RDY;
      when RDY =>
        if strobe='1' then current_state <= SEND;
        end if;
      when SEND => current_state <= POLL;
      when POLL =>
        if rdy_in='1' then current_state <= RDY;
        end if;
    end case;
  end if;
end process;
...

```

A.A. 2006/07 1 VHDL 34

Stili (Livelli di astrazione)

- Descrizione *dataflow*: flusso dei dati attraverso funzioni logiche e aritmetiche combinatorie (sovente detto RT)
- Esempio: contatore caricabile a 20 bit

```

graph LR
    data --> combin[logica combin.]
    load --> combin
    clock --> combin
    combin --> reg[registro a 20 bit]
    clock --> reg
    reg --> Q[Q[19:0]]
  
```

A.A. 2006/07 1 VHDL 35

Esempi

```
entity AND2 is
  port (in1, in2: in std_logic;
        out1: out std_logic);
end AND2;

architecture behavioral_2 of AND2 is
begin
  out1 <= in1 and in2;
end behavioral_2;
```

A.A. 2006/07 1 VHDL 36

```

entity XNOR2 is
  port (A, B: in std_logic;
        Z: out std_logic);
end XNOR2;

architecture behavioral_xnor of XNOR2 is
  -- signal declaration (of internal signals X, Y)
  signal X, Y: std_logic;
begin
  X <= A and B;
  Y <= (not A) and (not B);
  Z <= X or Y;
end behavioral_xnor;
    
```

A.A. 2006/07 1 VHDL 37

Stili (Livelli di astrazione)

Descrizione *structure*: netlist di elementi di libreria istanziati e interconnessi tramite segnali.

Contiene:

- *dichiarazioni di segnali (signal)* per le interconnessioni interne
 - le porte delle *entity* sono anch'esse trattate come segnali
- *istanziazioni di componenti*
 - istanziazioni di coppie *entity/architecture* già dichiarate
- *mappe delle porte nelle istanziazioni di componenti*
 - connettono i segnali alle porte dei componenti
- istruzioni *wait*.

A.A. 2006/07 1 VHDL 38

Esempio: registro a 4 bit

A.A. 2006/07 1 VHDL 39

1- dichiarazione di *entity/architecture* dei D-latch e della porta AND

```

entity d_latch is
  port (d, clk : in bit; q : out bit );
end d_latch;

architecture basic of d_latch is
begin
  latch_behavior : process is
  begin
    if clk = '1' then
      q <= d after 2 ns;
    end if;
    wait on clk, d;
  end process latch_behavior;
end basic;
    
```

```

entity and2 is
  port ( a, b : in bit; y : out bit );
end and2;

architecture basic of and2 is
begin
  and2_behavior : process is
  begin
    y <= a and b after 2 ns;
    wait on a, b;
  end process and2_behavior;
end basic;
    
```

A.A. 2006/07 1 VHDL 40

2- Uso dei componenti definiti per implementare un registro

```

architecture struct of reg4 is
  signal int_clk : bit;
begin
  bit0 : entity work.d_latch(basic)
    port map ( d0, int_clk, q0 );
  bit1 : entity work.d_latch(basic)
    port map ( d1, int_clk, q1 );
  bit2 : entity work.d_latch(basic)
    port map ( d2, int_clk, q2 );
  bit3 : entity work.d_latch(basic)
    port map ( d3, int_clk, q3 );
  gate : entity work.and2(basic)
    port map ( en, clk, int_clk );
end architecture struct;
    
```

A.A. 2006/07 1 VHDL 41

Esempio: shift/compare

- Descrizione dei blocchi (stili diversi)
- Loro interconnessione (struttura)

A.A. 2006/07 1 VHDL 42

Comparatore

```

library ieee;
use ieee.std_logic_1164.all;

entity compare is
port (A, B: in std_logic_vector(0 to 7);
EQ: out std_logic);
end compare;

architecture compare1 of compare is
begin
EQ <= '1' when (A = B) else 0;
end compare1;
    
```

dataflow

signal assignment

A.A. 2006/07 1 VHDL 43

Signal Assignment

- conditional signal assignment

```

architecture mux1 of mux is
begin
Y <= A when (Sel = "00") else
B when (Sel = "01") else
C when (Sel = "10") else
D when (Sel = "11");
end mux1;
    
```

- selected signal assignment

```

architecture mux2 of mux is
begin
with Sel select
Y <= A when "00",
B when "01",
C when "10",
D when "11";
end mux2;
    
```

A.A. 2006/07 1 VHDL 44

Shifter

Specifiche di funzionamento:

- reset asincrono
- caricamento con Load alto
- con Load basso rotazione di una posizione ogni fronte di salita del clock

Descrizione:

- behavioral (**process**)
- dataflow (**concurrent statements**)
- structural (**interconnected components**)

A.A. 2006/07 1 VHDL 45

Descrizione behavioral

Process:

- unica operazione concorrente
- le istruzioni interne a un process sono sequenziali
- concorrenza: processi multipli interagenti
- i segnali presenti in un process sono tutti aggiornati alla fine dell'esecuzione del processo.

A.A. 2006/07 1 VHDL 46

A.A. 2006/07 1 VHDL 47

Sintassi

```

architecture arch_name of ent_name is
begin
process_name: process(sensitivity_list)
[ process_declarations]
begin
sequential statement;
....
sequential statement;
end process;
end arch_name;
    
```

A.A. 2006/07 1 VHDL 48

- Sensitivity list: lista di segnali cui il process è sensibile. Evento su uno dei segnali ⇒ attivazione
- Process declarations: variabili e costanti usate nel process
- Tra begin e end: istruzioni eseguite sequenzialmente
- Process senza sensitivity list: wait until o wait for

A.A. 2006/07 1 VHDL 49

```

library ieee;
use ieee.std_logic_1164.all;

entity rotate is
port (Clk, Rst, Load: in std_logic;
      Data: in std_logic_vector(0 to 7);
      Q: out std_logic_vector(0 to 7);
end rotate;
    
```

A.A. 2006/07 1 VHDL 50

```

architecture rotate1 of rotate is
signal Qreg: std_logic_vector(0 to 7);
begin
reg: process(Rst, Clk)
begin
if Rst = '1' then
Qreg <= '00000000';
elseif (Clk='1' and Clk'event) then
if (Load='1') then
Qreg <= Data;
else
Qreg <= Qreg(1 to 7) & Qreg(0);
end if;
end if;
end process;
Q <= Qreg;
end rotate1;
    
```

A.A. 2006/07 1 VHDL 51

Descrizione dataflow

- lo hardware è intrinsecamente concorrente e composto dall'interconnessione di componenti elementari
- le assegnazioni a segnali sono istruzioni concorrenti (*concurrent statements*):
 - eseguite quando avviene un evento sui segnali, eventualmente con ritardo
 - evento: variazione di valore
- ordine delle istruzioni di assegnazione irrilevante

A.A. 2006/07 1 VHDL 52

Macchina di Moore

- circuito sequenziale sincrono in cui si distinguono parte combinatoria e registro di stato
- le uscite dipendono solo dallo stato
- concurrent statements per descrivere rete combinatoria, registro di stato e uscita

A.A. 2006/07 1 VHDL 53

A.A. 2006/07 1 VHDL 54

Uso di procedure

```

architecture rotate2 of rotate is
  signal D, Qreg: std_logic_vector(0 to 7);
begin
  D <= Data when (Load='1') else
    Qreg(1 to 7) & Qreg(0);
  dff(Rst, Clk, D, Qreg);
  Q <= Qreg;
end rotate2;

procedure dff (signal Rst, Clk: in std_logic;
  signal D: in std_logic_vector(0 to 7);
  signal Q: out std_logic_vector(0 to 7)) is
begin
  if Rst = '1' then
    Q <= '00000000';
  elseif (Clk='1' and Clk'event) then
    Q <= D;
  end if;
end dff;
    
```

A.A. 2006/07 1 VHDL 55

Uso di component

```

architecture rotate3 of rotate is
  component dff
    port(Rst, Clk: std_logic;
      D: in std_logic_vector(0 to 7);
      Q: out std_logic_vector(0 to 7));
  end component;
  signal D, Qreg: std_logic_vector(0 to 7);
begin
  D <= Data when (Load='1') else
    Qreg(1 to 7) & Qreg(0);
  REG1: dff port map(Rst, Clk, D, Qreg);
  Q <= Qreg;
end rotate4;
    
```

A.A. 2006/07 1 VHDL 56

```

library ieee;
use ieee.std_logic_1164.all;

entity dff is
  port (Rst, Clk: in std_logic;
    D: in std_logic_vector(0 to 7);
    Q: out std_logic_vector(0 to 7);
  end dff;

architecture behavior of dff is
begin
  process(Rst, Clk);
  begin
    if Rst = '1' then
      Q <= '00000000';
    elseif (Clk='1' and Clk'event) then
      Q <= D;
    end if;
  end process;
end behavior;
    
```

A.A. 2006/07 1 VHDL 57

Descrizione strutturale

- Descrizione strutturale: netlist di elementi di libreria istanziati e interconnessi tramite segnali.

A.A. 2006/07 1 VHDL 58

```

entity shiftcompare is
  port (Clk, Rst, Load: in std_ulogic;
    Init: in std_ulogic_vector(0 to 7);
    Test: in std_ulogic_vector(0 to 7);
    Limit: out std_ulogic);
end shiftcompare;
    
```

A.A. 2006/07 1 VHDL 59

```

library ieee;
use ieee.std_logic_1164.all;

entity shiftcompare is
  port (Clk, Rst, Load: in std_ulogic;
    Init: in std_ulogic_vector(0 to 7);
    Test: in std_ulogic_vector(0 to 7);
    Limit: out std_ulogic);
end shiftcompare;
    
```

A.A. 2006/07 1 VHDL 60

```

architecture structure of shiftcompare is
  component compare
    port(A, B: in std_ulogic_vector(0 to 7); EQ: out std_ulogic);
  end component;
  component shift
    port(Clk, Rst, Load: in std_ulogic;
          Data: in std_ulogic_vector(0 to 7);
          Q: out std_ulogic_vector(0 to 7));
  end component;
  signal Q: out std_ulogic_vector(0 to 7);
begin
  COMP1: compare port map (A=>Q, B=>Test EQ=>Limit);
  SHIFT1: shift port map (Clk=>Clk, Rst=>Rst, Load=>Load, Data=>Init, Q=>Q);
end structure;
    
```

A.A. 2006/07 1 VHDL 61

Ulteriori esempi di descrizioni strutturali

```

architecture ARCH_STRUC of COMP is
  component XR2 is
    port (X,Y: in bit; Z: out bit);
  end component XR2;
  component INV is
    port (X: in bit; Z: out bit);
  end component INV;
  signal I: bit;
begin
  I0: XR2 port map (A,B,I);
  I1: INV port map (I,C);
end architecture ARCH_STRUC;
    
```

A.A. 2006/07 1 VHDL 62

```

architecture structural of BUZZER is
  component AND2
    port (in1, in2: in std_logic; out1: out std_logic);
  end component;
  component OR2
    port (in1, in2: in std_logic; out1: out std_logic);
  end component;
  component NOT1
    port (in1: in std_logic; out1: out std_logic);
  end component;

  signal DOOR_NOT, SBELT_NOT, B1, B2: std_logic;
begin
  U0: NOT1 port map (DOOR, DOOR_NOT);
  U1: NOT1 port map (SBELT, SBELT_NOT);
  U2: AND2 port map (IGNITION, DOOR_NOT, B1);
  U3: AND2 port map (IGNITION, SBELT_NOT, B2);
  U4: OR2 port map (B1, B2, WARNING);
end structural;
    
```

A.A. 2006/07 1 VHDL 63

Progetto gerarchico

- Componenti definiti riusati come blocchi, celle o macro in entity di livello superiore
- Componenti di livello gerarchico inferiore descritti a diversi livelli di astrazione
- Descrizioni strutturali: interconnessione di componenti di livello gerarchico inferiore
- Esempio: full-adder da 4 bit come interconnessione di 4 full-adder da 1 bit

A.A. 2006/07 1 VHDL 64

```

entity FULLADDER is
  port (a, b, c: in std_logic;
        sum, carry: out std_logic);
end FULLADDER;

architecture fulladder_behav of FULLADDER is
begin
  sum <= (a xor b) xor c;
  carry <= (a and b) or (c and (a xor b));
end fulladder_behav;
    
```

A.A. 2006/07 1 VHDL 65

```

entity FOURBITADD is
  port (a, b: in std_logic_vector(3 downto 0);
        Cin : in std_logic;
        sum: out std_logic_vector (3 downto 0);
        Cout, V: out std_logic);
end FOURBITADD;
    
```

A.A. 2006/07 1 VHDL 66

```

architecture fouradder_structure of FOURBITADD is
  signal c: std_logic_vector (4 downto 0);
  component FULLADDER
    port(a, b, c: in std_logic;
         sum, carry: out std_logic);
  end component;
begin
  FA0: FULLADDER
    port map (a(0), b(0), C(0), sum(0), c(1));
  FA1: FULLADDER
    port map (a(1), b(1), C(1), sum(1), c(2));
  FA2: FULLADDER
    port map (a(2), b(2), C(2), sum(2), c(3));
  FA3: FULLADDER
    port map (a(3), b(3), C(3), sum(3), c(4));
  V <= c(3) xor c(4);
  Cout <= c(4);
end fouradder_structure;

```

A.A. 2006/07 1 VHDL 67

Modelli Misti Comportamento/Struttura

Un'architettura può contenere sia parti strutturali, sia parti comportamentali

- istruzioni **process** e istanziazioni di componenti
 - chiamate collettivamente *istruzioni concorrenti*
- i processi possono leggere e assegnare i segnali

Esempio: modello a livello register-transfer

- data path descritto come structure
- control unit descritta come behavior

A.A. 2006/07 1 VHDL 68

Esempio

Moltiplicatore a livello RT:

- descrizione strutturale del datapath
- descrizione comportamentale della control unit

A.A. 2006/07 1 VHDL 69

```

entity multiplier is
  port ( clk, reset : in bit;
        multiplicand, multiplier : in integer;
        product : out integer );
end multiplier;

```

A.A. 2006/07 1 VHDL 70

```

architecture mixed of multiplier is
  signal partial_product, full_product : integer;
  signal arith_control, result_en, mult_bit, mult_load : bit;

```

A.A. 2006/07 1 VHDL 71

```

begin
  arith_unit : entity work.shift_adder(behavior)
    port map ( addend => multiplicand, augend => full_product,
              sum => partial_product,
              add_control => arith_control );

```

A.A. 2006/07 1 VHDL 72

```

result : entity work.reg(behavior)
port map ( d => partial_product, q => full_product,
en => result_en, reset => reset );
    
```

A.A. 2006/07 1 VHDL 73

```

multiplier_sr : entity work.shift_reg(behavior)
port map ( d => multiplier, q => mult_bit,
load => mult_load, clk => clk );
product <= full_product;
    
```

A.A. 2006/07 1 VHDL 74

```

...
control_section : process is
-- variable declarations for control_section
-- ...
begin
-- sequential statements to assign values to control signals
-- ...
wait on clk, reset;
end process control_section;
end mixed;
    
```

A.A. 2006/07 1 VHDL 75

Validazione con simulazione

Uso di un modello *test bench*

- corpo di un'architettura che include un'istanziamento del progetto da validare (DUT *device under test*)
- applica sequenze di valori di collaudo agli ingressi
- osserva i valori sui segnali di uscita
 - o con un simulatore
 - o con un processo che verifica la correttezza del comportamento

A.A. 2006/07 1 VHDL 76

Esempi

```


entity test_bench is
end test_bench;
architecture test_reg4 of test_bench is
signal d0, d1, d2, d3, en, clk, q0, q1, q2, q3 : bit;
begin
dut : entity work.reg4(behav)
port map ( d0, d1, d2, d3, en, clk, q0, q1, q2, q3 );
stimulus : process is
begin
d0 <= '1'; d1 <= '1'; d2 <= '1'; d3 <= '1'; wait for 20 ns;
en <= '0'; clk <= '0'; wait for 20 ns;
en <= '1'; wait for 20 ns;
clk <= '1'; wait for 20 ns;
d0 <= '0'; d1 <= '0'; d2 <= '0'; d3 <= '0'; wait for 20 ns;
...
wait;
end process stimulus;
end test_reg4;
    
```

A.A. 2006/07 1 VHDL 77

```

entity test_bench is
end test_bench;
architecture behavior of test_bench is
component shiftcompare
port(Clk, Rst, Load: in std_logic;
Init: in std_logic_vector(0 to 7);
Test: out std_logic_vector(0 to 7);
Limit: out std_logic);
end component;
signal Clk, Rst, Load: std_logic;
signal Init: std_logic_vector(0 to 7);
signal Test: std_logic_vector(0 to 7);
signal Limit: std_logic;
    
```


A.A. 2006/07 1 VHDL 78



```

begin
  dut : shiftcompare  port map (Clk, Rst, Load, Init, Test, Limit);
  clock: process
    variable clktmp: std_logic := '0';
  begin
    clktmp := not clktmp;
    Clk <= clktmp;
    wait for 50 ns;
  end process;
  stimulus : process
  begin
    Rst <= '0'; Load <= '1'; Init <= '00001111'; Test <= '11110000';
    wait for 100 ns;
    Load <= '0';
    wait for 600 ns;
  end process;
end behavior;
    
```


A.A. 2006/07 1 VHDL 79



Design Processing

- Analisi
- Elaborazione
- Simulazione
- Sintesi

A.A. 2006/07 1 VHDL 80



Analisi


Controllo di errori sintattici e semantici

- sintassi: grammatica del linguaggio
- semantica: significato del modello

Analisi separata di ogni *design unit*

- entity declaration
- architecture body
- ...
- meglio se ogni design unit è in un file separato


A.A. 2006/07 1 VHDL 81



Le design units analizzate vengono memorizzate in una *libreria*

- in una forma interna dipendente dall'implementazione
- la libreria corrente è chiamata *work*

A.A. 2006/07 1 VHDL 82




Elaborazione

“Appiattimento” della gerarchia di progetto

- creazione delle porte
- creazione dei segnali e dei processi dentro un architecture body
- per ogni istanziazione di componente, copia di entity istanziata e architecture body

A.A. 2006/07 1 VHDL 83

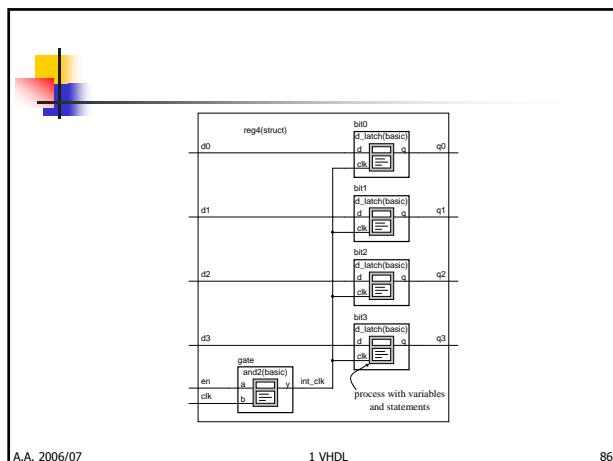
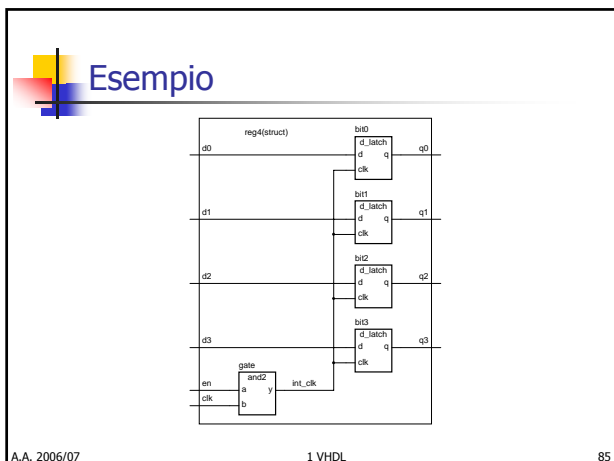


- ciclo ricorsivo
 - fino a architecture body puramente comportamentali

Risultato finale:

- insieme piatto di segnali e processi

A.A. 2006/07 1 VHDL 84



Simulazione

Esecuzione dei processi nel modello elaborato
 Simulazione a eventi discreti

- il tempo avanza con passi discreti
- quando un segnale cambia: *evento*

A.A. 2006/07 1 VHDL 87

Il processo è sensibile agli eventi sui suoi ingressi

- specificati in istruzioni **wait**
- riparte e programma nuovi valori sui segnali di uscita
 - programma *transazioni*
 - evento su un segnale se il nuovo valore è diverso dal vecchio

A.A. 2006/07 1 VHDL 88

Algoritmo di Simulazione

Fase di inizializzazione

- a ogni segnale è assegnato il suo valore iniziale
- il tempo di simulazione vale 0
- per ogni processo
 - attivazione
 - esecuzione fino a un'istruzione **wait**, poi sospensione
 - normalmente l'esecuzione programma transazioni su segnali in tempi successivi

A.A. 2006/07 1 VHDL 89

Ciclo di simulazione

- avanzare il tempo di simulazione al tempo della prossima transazione
- per ogni transazione a questo tempo
 - aggiorna il valore del segnale
 - genera un evento se il nuovo valore è diverso dal vecchio

A.A. 2006/07 1 VHDL 90

- Per ogni processo sensibile a qualunque di questi eventi o il cui tempo di "wait for ..." è terminato
 - ricomincia
 - esegui fino a un'istruzione **wait**, poi sospendi

La simulazione termina quando non ci sono più transazioni programmate.

A.A. 2006/07 1 VHDL 91

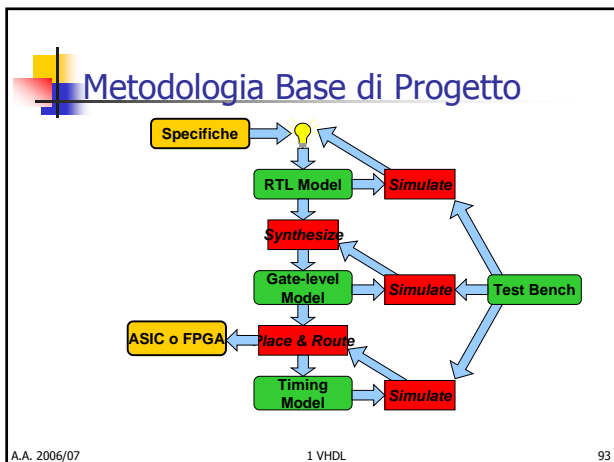
Sintesi

Traduce progetti a livello register-transfer (RTL) in netlist a livello gate

Restrizioni sullo stile di progetto per modelli RTL

Dipende dallo strumento.

A.A. 2006/07 1 VHDL 92



Il lessico del VHDL

A.A. 2006/07 1 VHDL 94

Elementi lessicali

Identificatori base:

- caratteri alfanumerici e _
- primo carattere lettera e ultimo non _
- case-insensitive
- qualsiasi lunghezza

Identificatori riservati:


- in grassetto negli esempi

A.A. 2006/07 1 VHDL 95

Numeri:

- default base 10
 - Esempi:
 - Interi 12 10 256E3 12e+6
 - Reali 1.2 256.24 3.14E-2
- altre basi:
 - base 2: **2#10010#**
 - base 16: **16#12#**
 - base 8: **8#22#**


A.A. 2006/07 1 VHDL 96



Caratteri:

- carattere tra singoli apici: 'a', 'B', ''
- stringa tra doppi apici: "This is a string"
- stringhe di bit:
 - base 2 B"1100_1001", b"1001011"
 - base 16 X"C9", X"4b"
 - base 8 O"311", o"113"

A.A. 2006/07 1 VHDL 97



Letterali fisici:


- rappresentano quantità fisiche (tempo, tensione, corrente, distanza)
- includono valore numerico e unità di misura:
 - 300 ns
 - 900 ps
 - 40 ma

A.A. 2006/07 1 VHDL 98



Oggetti e Tipi in VHDL

A.A. 2006/07 1 VHDL 99



Tre tipi di oggetti in VHDL:

- segnali (**signal**)
- variabili (**variable**)
- costanti (**constants**)

Ogni oggetto è dichiarato ed ha un suo tipo


A.A. 2006/07 1 VHDL 100



I segnali

- Segnale:
 - oggetto che connette elementi concorrenti (components, processes, concurrent assignments)
 - simile a un filo
 - nome con valore modificabile per assegnazione dopo la valutazione dell'espressione a destra con ritardo

A.A. 2006/07 1 VHDL 101



I segnali

- Dichiarazione


```

signal SUM, CARRY: std_logic;
signal CLOCK: bit;
signal TRIGGER: integer :=0;
signal DATA_BUS: bit_vector (0 to 7);
signal VALUE: integer range 0 to 100;
                
```
- visibilità:
 - locale all'architecture in cui è dichiarato
 - globale se dichiarato in un package e se il package è usato

A.A. 2006/07 1 VHDL 102

- Inizializzazione: in fase di dichiarazione, utile per la simulazione, non per la sintesi

```
signal BusA: std_logic_vector (15 downto 0) :=(others => 'Z');
```

- assegnazione:

```
SUM <= (A xor B) after 2 ns;
```

- le assegnazioni ai segnali sono concorrenti

A.A. 2006/07 1 VHDL 103

- Uso:

 - trasporto di informazioni tra elementi funzionali
 - segnali usati in espressioni logiche e assegnati direttamente

A.A. 2006/07 1 VHDL 104

Esempio: shift compare

```
signal Q: out std_ulogic_vector(0 to 7);
```

A.A. 2006/07 1 VHDL 105

```
architecture structure of shiftcompare is
  component compare
    port(A, B: in std_ulogic_vector(0 to 7); EQ: out std_ulogic);
  end component;
  component shift
    port(Clk, Rst, Load: in std_ulogic;
          Data: in std_ulogic_vector(0 to 7);
          Q: out std_ulogic_vector(0 to 7));
  end component;
  signal Q: out std_ulogic_vector(0 to 7);
begin
  COMP1: compare port map (A=>Q, B=>Test EQ=>Limit);
  SHIFT1: shift port map (Clk=>Clk, Rst=>Rst, Load=>Load, Data=>Init, Q=>Q);
end structure;
```

A.A. 2006/07 1 VHDL 106

Esempio: sincronizzatore

- Process dff con:
 - tre segnali Q1, Q2 e Q3 rappresentano celle di un registro
 - il segnale D3 è un segnale intermedio che rappresenta una funzione combinatoria di Q1, Q2 e Q3 e alimenta Q3
 - l'uscita di Q3 è assegnata a Request
- Ricordare: si tratta di statement concorrenti.

A.A. 2006/07 1 VHDL 107

```
entity synch is
  port(Rst, Clk, Grant, nSelect: in std_logic;
        Request: out std_logic);
end synch;

architecture dataflow of synch is
  signal Q1, Q2, Q3: std_logic;
begin
  diff: process (Rst, Clk);
  begin
    if Rst='1' then
      Q1 <= '0'; Q2 <= '0'; Q3 <= '0';
    elsif Clk='1' and Clk'event then
      Q1 <= Grant; Q2 <= Select; Q3 <= D3;
    end if;
  end process;
  D3 <= Q1 and Q2 or Q3;
  Request <= Q3;
end dataflow;
```

A.A. 2006/07 1 VHDL 108

Le variabili

- Variabile:
 - oggetto che memorizza valori tra statement VHDL sequenziali
 - permesse solo all'interno di processi, procedure e funzioni e ivi visibili
 - nome con valore modificabile per assegnazione istantaneamente

A.A. 2006/07 1 VHDL 109

Dichiarazione

```
variable CNTR_BIT: bit :=0;
variable VAR1: boolean :=FALSE;
variable SUM: integer range 0 to 256 :=16;
variable STS_BIT: bit_vector (7 downto 0);
```

- assegnazione:
 - variable_name := expression;
- le assegnazioni alle variabili sono sequenziali

A.A. 2006/07 1 VHDL 110

Esempio

```
entity synch is
port(Rst, Clk, Grant, nSelect: in std_ulogic;
Request: out std_ulogic;
end synch;
```

```
architecture behavior of synch is
begin
process (Rst, Clk);
variable Q1, Q2, Q3: std_ulogic;
begin
if Rst = '1' then
Q1 := '0';
Q2 := '0';
Q3 := '0';
elseif Clk = '1' and Clk'event then
Q1 := Grant;
Q2 := Select;
Q3 := Q1 and Q3 or Q2;
end if;
Request <= Q3;
end process;
end behavior;
```

A.A. 2006/07 1 VHDL 111

Differenza segnali-variabili

```
architecture VAR of EXAMPLE is
signal TRIGGER, RESULT: integer := 0;
begin
process
variable variable1: integer :=1;
variable variable2: integer :=2;
variable variable3: integer :=3;
begin
wait on TRIGGER;
variable1 := variable2;
variable2 := variable1 + variable3;
variable3 := variable2;
RESULT <= variable1 + variable2 + variable3;
end process;
end VAR
```

A.A. 2006/07 1 VHDL 112

```
architecture VAR of EXAMPLE is
signal TRIGGER, RESULT: integer := 0;
begin
process
variable variable1: integer :=1;
variable variable2: integer :=2;
variable variable3: integer :=3;
begin
wait on TRIGGER;
variable1 := variable2;
variable2 := variable1 + variable3;
variable3 := variable2;
RESULT <= variable1 + variable2 + variable3;
end process;
end VAR
```

TRIGGER

variable1 = 2

A.A. 2006/07 1 VHDL 113

```
architecture VAR of EXAMPLE is
signal TRIGGER, RESULT: integer := 0;
begin
process
variable variable1: integer :=1;
variable variable2: integer :=2;
variable variable3: integer :=3;
begin
wait on TRIGGER;
variable1 := variable2;
variable2 := variable1 + variable3;
variable3 := variable2;
RESULT <= variable1 + variable2 + variable3;
end process;
end VAR
```

TRIGGER

variable2 = 5

A.A. 2006/07 1 VHDL 114

```

architecture VAR of EXAMPLE is
  signal TRIGGER, RESULT: integer := 0;
begin
  process
    variable variable1: integer :=1;
    variable variable2: integer :=2;
    variable variable3: integer :=3;
  begin
    wait on TRIGGER;
    variable1 := variable2;
    variable2 := variable1 + variable3;
    variable3 := variable2;
    RESULT <= variable1 + variable2 + variable3;
  end process;
end VAR
    
```

TRIGGER

variable3 = 5

A.A. 2006/07 1 VHDL 115

```

architecture VAR of EXAMPLE is
  signal TRIGGER, RESULT: integer := 0;
begin
  process
    variable variable1: integer :=1;
    variable variable2: integer :=2;
    variable variable3: integer :=3;
  begin
    wait on TRIGGER;
    variable1 := variable2;
    variable2 := variable1 + variable3;
    variable3 := variable2;
    RESULT <= variable1 + variable2 + variable3;
  end process;
end VAR
    
```

TRIGGER+Δ

RESULT = 12

A.A. 2006/07 1 VHDL 116

```

architecture SIGN of EXAMPLE is
  signal TRIGGER, RESULT: integer := 0;
  signal signal1: integer :=1;
  signal signal2: integer :=2;
  signal signal3: integer :=3;
begin
  process
  begin
    wait on TRIGGER;
    signal1 <= signal2;
    signal2 <= signal1 + signal3;
    signal3 <= signal2;
    RESULT <= signal1 + signal2 + signal3;
  end process;
end SIGN;
    
```

A.A. 2006/07 1 VHDL 117

```

architecture SIGN of EXAMPLE is
  signal TRIGGER, RESULT: integer := 0;
  signal signal1: integer :=1;
  signal signal2: integer :=2;
  signal signal3: integer :=3;
begin
  process
  begin
    wait on TRIGGER;
    signal1 <= signal2;
    signal2 <= signal1 + signal3;
    signal3 <= signal2;
    RESULT <= signal1 + signal2 + signal3;
  end process;
end SIGN;
    
```

TRIGGER+Δ

signal1 = 2

signal2 = 4

signal3 = 2

RESULT = 6

A.A. 2006/07 1 VHDL 118

Costanti

Oggetti dichiarati con valore e tipo:

- Costante: nome assegnato con valore fisso
- uso: maggior leggibilità e modificabilità

```

constant RISE_FALL_TIME: time := 2 ns;
constant DELAY1: time := 4 ns;
constant RISE_TIME, FALL_TIME: time:= 1 ns;
constant DATA_BUS: integer:= 16;
    
```

A.A. 2006/07 1 VHDL 119

I Tipi di Dato

VHDL: linguaggio fortemente tipizzato.
4 classi di tipi:

- scalari
- compositi
 - array (elementi dello stesso tipo)
 - record (elementi di tipi diversi)
- accesso (puntatori)
- file

A.A. 2006/07 1 VHDL 120

Altra classificazione:

- tipi standard
- tipi definiti dall'utente
- tipi enumerati

A.A. 2006/07 1 VHDL 121

Tipi scalari

- bit: `signal A: bit:= '1';`
- boolean: `variable TEST: boolean;`
- character: `variable VAL: character:= '$';`
- integer: `constant CONST1: integer := 129;`
- natural: `variable VAR1: natural := 2;`
- positive: `variable VAR2: positive := 2;`
- real: `variable VAR3: real := +64.2E12;`
- severity level: note, warning, error, failure
- time: `variable DELAY: time := 5ns;`

A.A. 2006/07 1 VHDL 122

Tipi composti

- bit_vector: `signal INBUS: bit_vector(7 downto 0);`
- string: `variable VAR4: string(1 to 12) := "@$#ABC*()_ %Z";`
- record: tipo definito dall'utente
- array types:
 - bounded: `type MyArray is array (15 downto 0) of std_ulogic;`
 - unbounded: `type UArr is array (natural range <>) of std_ulogic;`

A.A. 2006/07 1 VHDL 123

Array

Array: `type array_name is array (indexing scheme) of element_type;`

Esempi: `type MY_WORD is array (15 downto 0) of std_logic;`
`type MY_MATRIX3X2 is array (1 to 3, 1 to 2) of natural;`

Unconstrained Array: `type array_name is array (type range <>) of element_type;`

Esempi: `type MATRIX is array (integer range <>) of integer;`
`variable MATRIX8: MATRIX (2 downto -8) := (3, 5, 1, 4, 7, 9, 12, 14, 20, 18);`

A.A. 2006/07 1 VHDL 124

Record

Record:

```

type name is
record
    identifier : subtype_indication;
    :
    identifier : subtype_indication;
end record;
    
```

Esempio:

```

type MY_MODULE is
record
    RISE_TIME : time;
    FALL_TIME : time;
    SIZE : integer range 0 to 200;
    DATA : bit_vector (15 downto 0);
end record;
    
```

A.A. 2006/07 1 VHDL 125

Tipi definiti dall'utente

`type identifier is type_definition;`

Esempi:

```

type small_int is range 0 to 1024;
type probability is range 0.0 to 1.0;
type conductance is range 0 to 2E-9
units
    mho;
    mmho = 1E-3 mho;
    umho = 1E-6 mho;
    nmho = 1E-9 mho;
    pmho = 1E-12 mho;
end units conductance;
    
```

A.A. 2006/07 1 VHDL 126

Tipi enumerati

`type type_name is (identifier list or character literal);`

Esempi:

```
type my_3values is ('0', '1', 'Z');
type PC_OPER is (load, store, add, sub, div, mult, shift, shiftr);
type hex_digit is ('0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F');
type state_type is (S0, S1, S2, S3);
```

A.A. 2006/07 1 VHDL 127

Esempio

```
architecture FSM1 of FSM is
type state is (START, RDY, SEND, POLL);
signal current_state: state;
begin
P: process (rst, CLK)
begin
if rst='1' then
current_state <= START;
elsif rising_edge(CLK) then
case current_state is
when START => current_state <= RDY;
when RDY =>
if strobe='1' then current_state <= SEND;
end if;
when SEND => current_state <= POLL;
when POLL =>
if rdy_in='1' then current_state <= RDY;
end if;
end case;
end if;
end process;
end FSM1;
```

A.A. 2006/07 1 VHDL 128

Gli Operatori

Operano su segnali, variabili e costanti

- logici
- relazionali
- addizione
- moltiplicazione
- segno
- vari
- shift

A.A. 2006/07 1 VHDL 129

- logici:
 - and, or, nand, nor, xor, nxor
- relazionali:
 - =, /=, >, <=, >, >=
- addizione
 - +, -, &

A.A. 2006/07 1 VHDL 130

- moltiplicazione:
 - *, /, mod, rem
- segno:
 - +, -
- vari
 - **, abs, not
- shift:
 - sll, srl, sla, sra, rol, ror

A.A. 2006/07 1 VHDL 131

Attributi

- Restituiscono informazioni su segnali, variabili o tipi
- tipi di attributi sulla base di quello che ritornano:
 - valore
 - funzione
 - segnale
 - tipo
 - intervallo.

A.A. 2006/07 1 VHDL 132

Attributi di valore

type'left	type bit_array is array(1 to 5) of bit; variable L: integer:= bit_array'left;	L vale 1
type'right	type bit_array is array(1 to 5) of bit; variable R: integer:= bit_array'right;	R vale 5
type'low	type bit_array is array(15 downto 0) of bit; variable L: integer:= bit_array'low;	L vale 0
type'high	type bit_array is array(-15 to 15) of bit; variable H: integer:= bit_array'high;	H vale 15
type'length	type bit_array is array(0 to 31) of bit; variable LEN: integer:= bit_array'length;	LEN vale 32
type'ascending	type asc_array is array(0 to 31) of bit; type desc_array is array(36 downto 4) of bit; variable A1: boolean:= asc_array'ascending; variable A2: boolean:= asc_array'ascending;	A1 è True A2 è False

A.A. 2006/07 1 VHDL 133

Attributi di funzione

```

type state_type is (Init, Hold, Strobe, Read, Idle);
type'pos(value) variable P: integer:= state_type'pos(Read);
type'val(value) variable V: state_type:= state_type'val(2);
type'succ(value) variable V: state_type:= state_type'succ(Init);
type'pred(value) variable V: state_type:= state_type'pred(Hold);
type'leftof(value) variable V: state_type:= state_type'leftof(Idle);
type'rightof(Value) V: state_type:= state_type'rightof(Read);
    
```

P vale 3
V vale Strobe
V vale Hold
V vale Init
V vale Read
V vale Idle

A.A. 2006/07 1 VHDL 134

Attributi di segnale

```

signal_name'event
signal_name'active
signal_name'transaction
signal_name'last_event
signal_name'last_active
signal_name'last_value
signal_name'delayed(T)
signal_name'stable(T)
signal_name'quiet(T)
    
```

Esempio: fronte di salita di un clock

```

if (CLOCK'event and CLOCK='1') then ...
    
```

A.A. 2006/07 1 VHDL 135

Concurrent statements

A.A. 2006/07 1 VHDL 136

Concurrent statements

- Si trovano nella *concurrent area*, cioè tra begin e end di una architecture.
- Esempio:

```

architecture arch1 of my_circuit is
  signal Reset, DivClk: std_logic;
  constant MaxCount: std_logic_vector(15 downto 0):= '10001111';
  component count port(Clk, Rst: in std_logic;
    Q: out std_logic_vector(15 downto 0));
  begin
    Reset <= '1' when Qout = MaxCount else '0';
    Control: process(DivClk);
    begin
      ....
    end;
  end arch1;
    
```

A.A. 2006/07 1 VHDL 137

Concurrent statements

Tipi di concurrent statements:

- concurrent signal assignments
- procedure calls
- generate statements
- concurrent processes
- component instantiations

A.A. 2006/07 1 VHDL 138

Concurrent signal assignment

- Sintassi:
Target_signal <= expression;
- Uso:
 - descrizione di logica combinatoria
 - connessione di componenti a livello inferiore
- Esempi:


```
Sum <= (A xor B) xor Cin;
Carry <= (A and B);
Z <= (not X) or Y after 2 ns;
```

A.A. 2006/07 1 VHDL 139

Esempio: full-adder da 4 bit

```
entity ADD4 is
  port (A: in STD_LOGIC_VECTOR (3 downto 0);
        B: in STD_LOGIC_VECTOR (3 downto 0);
        CIN: in STD_LOGIC;
        SUM: out STD_LOGIC_VECTOR (3 downto 0);
        COUT: out STD_LOGIC );
end ADD4;

architecture ADD4_concurnt of ADD4 is
  signal SUMINT: STD_LOGIC_VECTOR(4 downto 0);
begin
  SUMINT <= ('0' & A) + ('0' & B) + ("0000" & CIN);
  COUT <= SUMINT(4);
  SUM <= SUMINT(3 downto 0);
end ADD4_concurnt;
```

A.A. 2006/07 1 VHDL 140

Conditional signal assignments

- Sintassi:
Target_signal <= expression when Boolean_condition else
expression when Boolean_condition else
:
expression;
- Esempio: multiplexer 4:1


```
entity MUX_4_1_Conc is
  port (S1, S0, A, B, C, D: in std_logic;
        Z: out std_logic);
end MUX_4_1_Conc;
architecture concurr_MUX41 of MUX_4_1_Conc is
begin
  Z <= A when S1='0' and S0='0' else
        B when S1='0' and S0='1' else
        C when S1='1' and S0='0' else
        D when others;
end concurr_MUX41;
```

A.A. 2006/07 1 VHDL 141

Specifica di ritardi

- Nelle assegnazioni di segnali possono essere specificati ritardi:
- Modelli di ritardo:
 - inerziale (filtra gli impulsi più corti di una soglia):
Y1 <= not(A and B) after 7ns;
 - trasporto (propaga tutti gli impulsi):
Y2 <= not(A and B) transport after 7ns;

A.A. 2006/07 1 VHDL 142

Esempio: multiplexer 4:1 con tabella di verità

```
entity MUX_4_1_funcTab is
  port (A, B, C, D: in std_logic;
        SEL: in std_logic_vector (1 downto 0);
        Z: out std_logic);
end MUX_4_1_funcTab;

architecture concurr_MUX41 of MUX_4_1_funcTab is
begin
  Z <= A when SEL = "00" else
        B when SEL = "01" else
        C when SEL = "10" else
        D when others;
end concurr_MUX41;
```

A.A. 2006/07 1 VHDL 143

Selected signal assignments

- Sintassi:
with choice_expression select
target_name <= expression when choices,
:
target_name <= expression when choices;
- Esempio: multiplexer 4:1


```
entity MUX_4_1_Conc2 is
  port (A, B, C, D: in std_logic; SEL: in std_logic_vector(1 downto 0);
        Z: out std_logic);
end MUX_4_1_Conc2;
architecture concurr_MUX41b of MUX_4_1_Conc2 is
begin
  with SEL select
    Z <= A when "00",
          B when "01",
          C when "10",
          D when "11";
end concurr_MUX41b;
```

A.A. 2006/07 1 VHDL 144

Esempio: full-adder da 4 bit

```
entity FullAdd_Conc is
  port (A, B, C: in std_logic;
        sum, cout: out std_logic);
end FullAdd_Conc;
```

A.A. 2006/07 1 VHDL 145

```
architecture FullAdd_Conc of FullAdd_Conc is
  signal INS: std_logic_vector(2 downto 0);
begin
  INS(2) <= A;
  INS(1) <= B;
  INS(0) <= C;
  with INS select
    (sum, cout) <= std_logic_vector("00") when "000",
                  std_logic_vector("10") when "001",
                  std_logic_vector("10") when "010",
                  std_logic_vector("01") when "011",
                  std_logic_vector("10") when "100",
                  std_logic_vector("01") when "101",
                  std_logic_vector("11") when "110",
                  std_logic_vector("11") when "111",
                  std_logic_vector("11") when others;
end FullAdd_Conc;
```

A.A. 2006/07 1 VHDL 146

Differenza tra conditional e selected signal assignments:

- nei conditional signal assignments c'è una priorità (implicata dall'ordine delle espressioni)
- nei selected signal assignments non c'è priorità.

A.A. 2006/07 1 VHDL 147

Procedure calls

- Chiamate concorrenti a procedure all'interno di una architecture
- Simili ai process, ma il corpo della procedura può trovarsi altrove (per esempio in un package).

```
architecture rotate2 of rotate is
  signal D, Qreg: std_logic_vector(0 to 7);
begin
  D <= Data when (Load='1') else
        Qreg(1 to 7) & Qreg(0);
  dff(Rst, Clk, D, Qreg);
  Q <= Qreg;
end rotate2;
```

A.A. 2006/07 1 VHDL 148

Generate statements

- Creazione di più istanze di concurrent statements
- Uso: istanziazione di più componenti identici
- Tipi:
 - for-generate
 - if-generate

A.A. 2006/07 1 VHDL 149

- Esempi:
 - RAM 16x4 a partire da RAM 16x1

```
architecture gen_ex of my_entity is
  component RAM16X1
    port(A0, A1, A2, A3, WE, D: in std_logic;
          O: out std_logic);
  end component;
begin
  RAMGEN: for i in 0 to 3 generate
    RAM: RAM16x1 portmap(...);
  end generate;
end gen_ex;
```

A.A. 2006/07 1 VHDL 150

■ 10-bit parity generator con EXOR in cascata

```
entity parity10 is
  port(D: in std_logic_vector(0 to 9);
        ODD: out std_logic);
  constant width: integer :=10;
end parity10;
```

```
architecture structure of parity10 is
  component xor2
    port(A, B: in std_logic;
          y: out std_logic);
  end component;
  signal p: std_logic_vector(0 to width-2);
begin
  G: for I in 0 to (width-2) generate
    G0: if I=0 generate
      X0: xor2 port map (A=>D(0), B=>D(1), Y=>p(0));
    end generate G0;
    G1: if I>0 and I<(width-2) generate
      X0: xor2 port map (A=>p(i-1), B=>D(i+1), Y=>p(i));
    end generate G1;
    G2: if I=(width-2) generate
      X0: xor2 port map (A=>p(i-1), B=>D(i+1), Y=>ODD);
    end generate G2;
  end generate G;
end structure;
```

Concurrent processes

- I processi sono tra di loro concorrenti e all'interno contengono sequential statements
- Modellano l'intrinseca concorrenza dell'hardware
- Esempio: due processi che descrivono un clock e una sequenza di stimoli di un test bench

```
architecture Stim1 of TEST_COUNT4EN is
  component COUNT4EN
    port(CLK, RESET, EN: in std_logic;
          COUNT: out std_logic_vector(3 downto 0));
  end component;
  constant CLK_CYCLE: Time :=20 ns;
```

```
signal CLK, INIT_RESET, EN: std_logic;
signal COUNT_OUT: out std_logic_vector(3 downto 0);
begin
  U0: COUNT4EN port map (CLK=>CLK, RESET=>INIT_RESET,
                        EN=>EN, COUNT=>COUNT_OUT);

  process begin
    CLK <= '1';
    wait for CLK_CYCLE/2;
    CLK <= '0';
    wait for CLK_CYCLE/2;
  end process;
  process begin
    INIT_RESET <= '0'; EN <= '1';
    wait for CLK_CYCLE/3;
    INIT_RESET <= '0';
    wait for CLK_CYCLE*10;
    EN <= '0';
    wait for CLK_CYCLE*3;
    EN <= '1';
    wait
  end process;
end Stim1;
```

Component instatiations

- Circuito come componenti predefinite interconnesse
 - dichiarazione componenti usati
 - dichiarazione dei segnali di interconnessione
 - istanziazione dei componenti

```
architecture architecture_name of NAME_OF_ENTITY is
  -- Declarations
  component declarations
  signal declarations
begin
  -- Statements
  component instantiation and connections
  :
end architecture_name;
```

Dichiarazione di componenti

```
component component_name [is]
  port (port_signal_names: mode type;
        port_signal_names: mode type;
        :
        port_signal_names: mode type);
end component [component_name];
```

Esempi:

```
component OR2
  port (in1, in2: in std_logic;
        out1: out std_logic);
end component;

component FULLADDER
  port(a, b, c: in std_logic;
        sum, carry: out std_logic);
end component;
```

Instanziazione di componenti e interconnessione

- Instanziazione di componenti:
 - già definiti
 - di libreria

instance_name : component name
port map (port1=>signal1, port2=> signal2,... port3=>signaln);

oppure
port map (signal1, signal2,...signaln);

A.A. 2006/07 1 VHDL 157

Esempio:

```

component NAND2
  port (in1, in2: in std_logic;
        out1: out std_logic);
end component;

signal int1, int2, int3: std_logic;
architecture struct of EXAMPLE is
  U1: NAND2 port map (A,B,int1);
  U2: NAND2 port map (in1=>C, in2=>D, out1=>int2);
  U3: NAND2 port map (in1=>int1, int2, Z);
  .....
end architecture;
    
```

A.A. 2006/07 1 VHDL 158

Esempio: progetto di ALU

ALU: Arithmetic Logic Unit: circuito che esegue operazioni logico-aritmetiche su due input

ALU seriale: input e output memorizzati in shift register ed elaborati 1 bit alla volta

Vantaggi/svantaggi:

- meno hardware
- più tempo.

A.A. 2006/07 1 VHDL 159

Specifiche

ALU seriale

- combinatoria
- 6 input:
 - 2 data input A, B da 1 bit
 - 1 carry input Cin da 1 bit
 - 3 input di selezione di funzione F1, F2, F3 (opcode)
- 2 output:
 - 1 data output S da 1 bit
 - 1 carry out Cout da 1 bit

A.A. 2006/07 1 VHDL 160

Funzioni

F1F2F3	Operazione	Nome
0 0 0	S <= 0	Clear
0 0 1	S <= A'	Complement
0 1 0	S <= A più Cin	Increment
0 1 1	S <= A' più Cin	Negate
1 0 0	S <= B	Transfer B
1 0 1	S <= A XOR B	Exor
1 1 0	S <= A più B più CI	Add
1 1 1	S <= A più B' più Cin	Subtract

A.A. 2006/07 1 VHDL 161

Entity

```

entity alu_1 is
  port(opcode : in bit_vector(2 downto 0);
        A : in bit;
        B : in bit;
        Cin : in bit;
        clock : in bit;
        S : out bit;
        Cout : out bit);
end alu_1;
    
```

A.A. 2006/07 1 VHDL 162

Behavioral Architecture

```

begin -- behave
process
begin
wait until clock'event and clock='1';
case opcode is
when "000" => -- clear, S <= 0
S <= '0';
when "001" => -- complement, S <= A'
S <= not(A);
when "010" => -- increment, S <= A+Cin
S <= A xor Cin;
Cout <= A and Cin;
when "011" => -- negate, S <= A' + Cin
A <= not(A xor Cin);
Cout <= (not(A)) and Cin;
when "100" => -- transfer B, S <= B
S <= B;

```

A.A. 2006/07 1 VHDL 163

```

when "101" => -- XOR, A XOR B
S <= A xor B;
when "110" => -- Add, A + B + Cin
S <= (A xor B) xor Cin;
Cout <= (A and Cin) or (B and (A xor Cin));
when "111" => -- Subtract, A + B' + Cin
S <= A xor (not(B)) xor Cin;
Cout <= (A and Cin) or ((not(B)) and (A xor Cin));
when OTHERS =>
end case; -- opcode
end process;
end behave_alu_1;

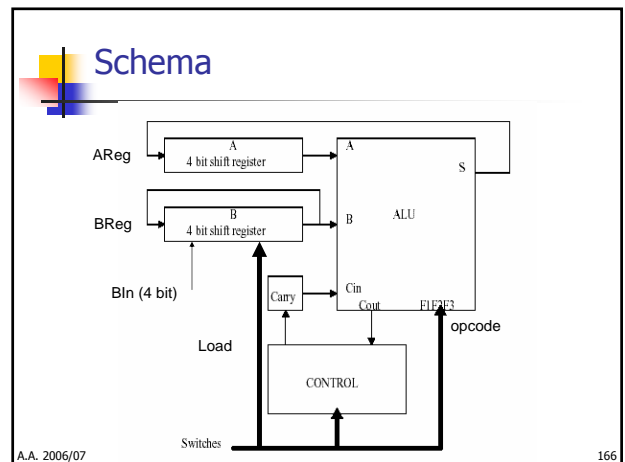
```

A.A. 2006/07 1 VHDL 164

ALU da 4 bit: specifiche

- 2 shift register AReg e BReg per gli operandi su 4 bit
- input:
 - 3 input di selezione di funzione F1, F2, F3 (opcode)
 - 4 segnali di dato BIn per il registro B
 - 1 segnale di Load
 - 1 segnale di Clock
- output:
 - 4 segnali AReg
 - 4 segnali Breg
 - 1 segnale Carry

A.A. 2006/07 1 VHDL 165



Entity

```

entity alu_2 is
port(opcode : in bit_vector(2 downto 0);
BIn : in bit_vector(3 downto 0);
Load : in bit;
Clock : in bit;
AReg : buffer bit_vector(3 downto 0);
BReg : buffer bit_vector(3 downto 0);
Carry : buffer bit);
end alu_2;

```

A.A. 2006/07 1 VHDL 167

Behavioral Architecture ALU 4 bit

```

architecture behave_alu_2 of alu_2 is
begin -- behave_alu_2
process
variable A : bit;
variable B : bit;
variable Cin : bit;
variable S : bit;
variable Cout : bit;
begin -- process
wait until clock'event and clock = '1';
if (Load = '1') then
BReg <= BIn;
AReg <= "0000";
else
A := AReg(0);
AReg(0) <= AReg(1);
AReg(1) <= AReg(2);
AReg(2) <= AReg(3);

```

A.A. 2006/07 1 VHDL 168

```

B := BReg(0);
BReg(0) <= BReg(1);
BReg(1) <= BReg(2);
BReg(2) <= BReg(3);
BReg(3) <= B;
Cin := Carry;
case opcode is
when "000" => -- clear, S <= 0
  S := '0';
when "001" => -- complement, S <= A'
  S := not(A);
when "010" => -- increment, S <= A+Cin
  S := A xor Cin;
  Cout := A and Cin;
when "011" => -- negate, S <= A' + Cin
  S := not(A xor Cin);
  Cout := (not(A)) and Cin;

```

```

when "100" => -- transfer B, S <= B
  S := B;
when "101" => -- XOR, A XOR B
  S := A xor B;
when "110" => -- Add, A + B + Cin
  S := (A xor B) xor Cin;
  Cout := (A and Cin) or (B and (A xor Cin));
when "111" => -- Subtract, A + B' + Cin
  S := A xor (not(B)) xor Cin;
  Cout := (A and Cin) or ((not(B)) and (A xor Cin));
when others =>
end case; -- opcode
AReg(3) <= S;
Carry <= Cout;
end if;
end process;
end behave_alu_2;

```

Sequential statements

Sequential statements

Servono a descrivere il comportamento di un sistema come sequenza di eventi correlati. Il comportamento sequenziale è un modello naturale per:

- macchine a stati finiti
- logica combinatoria con priorità di operazioni

Process: statement concorrente che contiene statements sequenziali ed è eseguito in base a certi eventi.

Process: sintassi

```

[process_label:] process [ (sensitivity_list) ] [is]
  [ process_declarations ]
begin
  lista di sequential statements come:
  signal assignments
  variable assignments
  case statement
  exit statement
  if statement
  loop statement
  next statement
  null statement
  procedure call
  wait statement
end process [process_label];

```

- Il process è normalmente dormiente (idle)
- Se c'è una sensitivity list (lista di segnali cui il process è sensibile), un evento su uno dei segnali ⇒ attivazione del processo
- Se non c'è una una sensitivity list, il process sarà sempre in esecuzione fino a quando raggiunge un wait statement
- I process devono avere in alternativa o una sensitivity list o uno o più wait statement (non entrambi)

- process declarations: variabili e costanti usate nel process
- tra begin e end: istruzioni eseguite sequenzialmente
- ricordare la differenza tra assegnazione a segnali e a variabili

A.A. 2006/07 1 VHDL 175

If

```

if condition then
    sequential statements
[elsif condition then
    sequential statements ]
[else
    sequential statements ]
end if;
    
```

Esempio: multiplexer 4:1

```

entity MUX_4_1a is
    port (S1, S0, A, B, C, D: in std_logic;
          Z: out std_logic);
end MUX_4_1a;
    
```

A.A. 2006/07 1 VHDL 176

```

architecture behav_MUX41a of MUX_4_1a is
begin
    P1: process (S1, S0, A, B, C, D)
    begin
        if S1='0' and S0='0' then
            Z <= A;
        elsif S1='0' and S0='1' then
            Z <= B;
        elsif S1='1' and S0='0' then
            Z <= C;
        elsif S1='1' and S0='1' then
            Z <= D;
        end if;
    end process P1;
end behav_MUX41a;
    
```

A.A. 2006/07 1 VHDL 177

Case

```

case expression is
    when choices =>
        sequential statements
    when choices =>
        sequential statements
    -- branches are allowed
    [when others => sequential statements ]
end case;
    
```

Esempio: multiplexer 4:1

```

entity MUX_4_1 is
    port ( SEL: in std_logic_vector(2 downto 1);
          A, B, C, D: in std_logic;
          Z: out std_logic);
end MUX_4_1;
    
```

A.A. 2006/07 1 VHDL 178

```

architecture behav_MUX41 of MUX_4_1 is
begin
    PR_MUX: process (SEL, A, B, C, D)
    begin
        case SEL is
            when "00" => Z <= A;
            when "01" => Z <= B;
            when "10" => Z <= C;
            when "11" => Z <= D;
            when others => Z <= 'X';
        end case;
    end process PR_MUX;
end behav_MUX41;
    
```

A.A. 2006/07 1 VHDL 179

Loop

```

[ loop_label :]iteration_scheme loop
    sequential statements
    [next [label] [when condition];
    [exit [label] [when condition];
end loop [loop_label];
    
```

- next termina l'iterazione corrente e l'esecuzione procede alla prossima iterazione
- exit termina completamente il loop
- 3 schemi di iterazione:
 - loop base
 - while loop
 - for loop

A.A. 2006/07 1 VHDL 180

Basic loop

```
[ loop_label :] loop
  sequential statements
  [next [label] [when condition];
  [exit [label] [when condition];
end loop [ loop_label];
```

- assenza di iteration_scheme
- esecuzione continua fino a exit 0 next
- deve avere almeno un wait

A.A. 2006/07 1 VHDL 181

Esempio: contatore modulo 32

```
entity COUNT31 is
  port (CLK: in std_logic;
        COUNT: out integer);
end COUNT31;
architecture behav_COUNT of COUNT31 is
begin
  P_COUNT: process
  variable intern_value: integer :=0;
  begin
    COUNT <= intern_value;
    loop
      wait until CLK='1';
      intern_value := (intern_value + 1) mod 32;
      COUNT <= intern_value;
    end loop;
  end process P_COUNT;
end behav_COUNT;
```

A.A. 2006/07 1 VHDL 182

While-loop

```
[ loop_label :] while condition loop
  sequential statements
  [next [label] [when condition];
  [exit [label] [when condition];
end loop [ loop_label ];
```

- test della condizione prima dell'iterazione

A.A. 2006/07 1 VHDL 183

For-loop

```
[ loop_label :] for identifier in range loop
  sequential statements
  [next [label] [when condition];
  [exit [label] [when condition];
end loop [ loop_label ];
```

- indice dichiarato implicitamente e disponibile solo all'interno del loop
- range
 - integer_expression to integer_expression
 - integer_expression downto integer_expression

A.A. 2006/07 1 VHDL 184

Esempio: controllore di parità

```
entity PAR_GEN is
  port (A : in std_logic_vector (3 downto 0);
        PAR: out std_logic);
end PAR_GEN;
architecture ARC of PAR_GEN is
begin
  P1: process (A)
  variable TMP: std_logic;
  begin
    TMP := '0';
    for I in A'low to A'high loop
      TMP := TMP xor A(I);
    end loop;
    PAR <= TMP;
  end process P1;
end ARC;
```

A.A. 2006/07 1 VHDL 185

Wait

- Wait ferma un processo fino ad un evento
- in un process wait e sensitivity list sono mutuamente esclusivi
- 4 forme:
 - wait until condition;
 - wait for time expression;
 - wait on signal;
 - wait;
- esempi:


```
wait until CLK='1';
wait until CLK='0';
wait until CLK'event and CLK='1';
wait until not CLK'stable and CLK='1';
```

A.A. 2006/07 1 VHDL 186

Null

- Non avviene alcuna azione
- Utile quando tutte le possibilità devono essere considerate, anche se alcune possono essere ignorate

A.A. 2006/07 1 VHDL 187

Process con sensitivity list

D-FF, positive-edge triggered, clear asincrono

```
entity DFF_CLEAR is
  port (CLK, CLEAR, D : in std_logic;
        Q : out std_logic);
end DFF_CLEAR;
architecture BEHAV_DFF of DFF_CLEAR is
begin
  DFF_PROCESS: process (CLK, CLEAR)
  begin
    if (CLEAR = '1') then
      Q <= '0';
    elsif (CLK'event and CLK = '1') then
      Q <= D;
    end if;
  end process;
end BEHAV_DFF;
```

A.A. 2006/07 1 VHDL 188

Process per reti combinatorie

Multiplexer 4-1

```
entity simple_mux is
  port (Sel: in bit_vector (0 to 1);
        A, B, C, D: in bit;
        Y: out bit);
end simple_mux;
architecture behavior of simple_mux is
begin
  process (Sel, A, B, C, D)
  begin
    if Sel = "00" then Y <= A;
    elsif Sel = "01" then Y <= B;
    elsif Sel = "10" then Y <= C;
    elsif Sel = "11" then Y <= D;
    end if;
  end process;
end simple_mux;
```

A.A. 2006/07 1 VHDL 189

Un process è sintetizzato come rete combinatoria se valgono le seguenti regole:

1. La sensitivity list include tutti i segnali usati come input al process
2. Gli assignment per gli output del process coprono tutti i casi possibili degli input del process
3. Tutte le variabili usate nel process hannoun valore prima di essere lette

A.A. 2006/07 1 VHDL 190

Esempio di process solo apparentemente combinatorio:

```
entity simple_mux is
  port (Sel: in bit_vector (0 to 1);
        A, B, C, D: in bit;
        Y: out bit);
end simple_mux;
architecture behavior of simple_mux is
begin
  process (Sel, A, B, C, D)
  begin
    if Sel = "00" then Y <= A;
    elsif Sel = "01" then Y <= B;
    elsif Sel = "10" then Y <= C;
    end if;
  end process;
end simple_mux;
```

Il caso Sel="11" non è specificato, Y mantiene il valore precedente (comportamento sequenziale)

A.A. 2006/07 1 VHDL 191

Process per reti sequenziali

1. Il process non include tutti i suoi input nella sensitivity list
2. Usare statement if-then-else che non enumerano tutti i casi (uno o più segnali mantengono il valore)
3. Usare variabili che mantengono il valore tra diverse iterazioni del process (per es., specificare una variabile come input di un'assegnazione prima di averle assegnato un valore).

A.A. 2006/07 1 VHDL 192

Esempio: shifter a 8 bit

```

entity rotate is
port (Clk, Rst, Load: in std_logic;
      Data: in std_logic_vector(0 to 7);
      Q: out std_logic_vector(0 to 7));
end rotate;

architecture rotate1 of rotate is
signal Qreg: std_logic_vector(0 to 7);
begin
reg: process(Rst, Clk)
begin
if Rst = '1' then Qreg <= '00000000';
elseif (Clk='1' and Clk'event) then
if (Load='1') then
Qreg <= Data;
else
Qreg <= Qreg(1 to 7) & Qreg(0);
end if;
end if;
end process;
Q <= Qreg;
end rotate1;
    
```

A.A. 2006/07 1 VHDL 193

Esempio: macchina di Mealy

- Riconoscitore della sequenza 1011
- Sliding window

```

entity myvhdl is
port (CLK, RST, X: in STD_LOGIC;
      Z: out STD_LOGIC);
end;
    
```

A.A. 2006/07 1 VHDL 194

Esempio: macchina di Mealy

- Riconoscitore della sequenza 1011
- Sliding window

```

entity myvhdl is
port (CLK, RST, X: in STD_LOGIC;
      Z: out STD_LOGIC);
end;
    
```

A.A. 2006/07 1 VHDL 195

```

architecture myvhdl_arch of myvhdl is
type Sreg0_type is (S1, S2, S3, S4);
signal Sreg0: Sreg0_type;
begin
Sreg0_machine: process (CLK)
begin
if CLK'event and CLK = '1' then
if RST='1' then
Sreg0 <= S1;
else
case Sreg0 is
when S1 =>
if X='0' then
Sreg0 <= S1;
elseif X='1' then
Sreg0 <= S2;
end if;
end if;
end process;
    
```

A.A. 2006/07 1 VHDL 196

```

when S2 =>
if X='1' then
Sreg0 <= S2;
elseif X='0' then
Sreg0 <= S3;
end if;
when S3 =>
if X='1' then
Sreg0 <= S4;
elseif X='0' then
Sreg0 <= S1;
end if;
when S4 =>
if X='0' then
Sreg0 <= S3;
elseif X='1' then
Sreg0 <= S2;
end if;
    
```

A.A. 2006/07 1 VHDL 197

```

when others =>
null;
end case;
end if;
end process;
Z_assignment:
Z <= '0' when (Sreg0 = S1 and X='0') else
'0' when (Sreg0 = S1 and X='1') else
'0' when (Sreg0 = S2 and X='1') else
'0' when (Sreg0 = S2 and X='0') else
'0' when (Sreg0 = S3 and X='1') else
'0' when (Sreg0 = S3 and X='0') else
'0' when (Sreg0 = S4 and X='0') else
'1' when (Sreg0 = S4 and X='1') else
'1';
end myvhdl_arch;
    
```

A.A. 2006/07 1 VHDL 198

Process per test stimuli

- Descrizione di un ambiente di test:
 - applicazione sequenziale di stimoli
 - eventualmente controllo delle uscite

```

process
constant PERIOD: time := 40ns;
begin
  Rst <= '1';
  A <= '00000000';
  B <= '00000000';
  wait for PERIOD;
  CheckState(Q, "00000000");
  Rst <= '0';
  A <= "10101010";
  B <= "01010101";
  wait for PERIOD*4;
  CheckState(Q, "11111111");
  wait;
end process;
    
```

A.A. 2006/07 1 VHDL 199

Tecniche di partizionamento

A.A. 2006/07 1 VHDL 200

Il partizionamento

Utile per gestire progetti complessi sviluppati in squadra.

Il VHDL supporta il partizionamento mediante:

- block
- package
- library
- component
- configuration.

A.A. 2006/07 1 VHDL 201

Block

Block: raggruppamento di statements correlati logicamente all'interno di un'architecture.

Possono contenere dichiarazioni di costanti e segnali locali.

Migliorano la leggibilità del progetto.

A.A. 2006/07 1 VHDL 202

Sintassi:

```

architecture my_arch of my_entity is
begin
  BLOCK1: block
  signal a, b: std_logic;
  begin
    statements
  end block BLOCK1;

  BLOCK2: block
  signal a, b: std_logic;
  begin
    statements
  end block BLOCK2;
end my_arch;
    
```

A.A. 2006/07 1 VHDL 203

Package

Package: contenitore di dichiarazioni usate comunemente:

- tipi e sottotipi
- costanti
- componenti
- attributi
- funzioni e procedure.

A.A. 2006/07 1 VHDL 204

Quando le dichiarazioni sono usate altrove, si ricorre a use per rendere visibile il package.

```

library ieee;
use ieee.std_logic_1164.all;
package my_types is
  subtype byte is std_logic(0 to 7);
  constant CLEAR: byte :=(others =>'0');
end my_types;

use work.my_types.all;
use ieee.std_logic_1164.all;
entity rotate is
  port (Clk, Rst, Load: in std_logic;
        Data: in byte;
        Q: out byte);
end rotate;

architecture rotate4 of rotate is
  signal Qreg: byte;
begin
  Qreg<= Data when (Load='1') else
    Qreg(1 to byte'LENGTH-1) & Qreg(0);
  dff(Rst, Clk, Qreg, Q);
end rotate4;
    
```

A.A. 2006/07 1 VHDL 205

Uso dei package

- Raggruppati in librerie per rendere comuni certe dichiarazioni.
- Package predisposti (esempio il package **std_logic_1164** contiene le dichiarazioni per i tipi **std_logic**, **std_ulogic**, **std_logic_vector**, **std_ulogic_vector**).
- Package venduti con il software di simulazione o di sintesi
- Package standard per le dichiarazioni dei tipi standard (**bit**, **bit_vector**, **integer**, etc.). Non richiede **use**.

A.A. 2006/07 1 VHDL 206

Package body

Se il package contiene funzioni o procedure, il package body contiene la loro descrizione.

Esempio: batteria di 8 flip-flop tipo D:

```

package my_reg8 is
  subtype byte8 is std_logic_vector(0 to 7);
  constant CLEAR8: byte8 :=(others =>'0');
  procedure dff8(signal Rst, Clk: in std_logic;
                signal D: in byte;
                signal Q: out byte8);
end my_reg8;

package body my_reg8 is
  procedure dff8(signal Rst, Clk: in std_logic;
                signal D: in byte8;
                signal Q: out byte8) is
  begin
    if Rst='1' then
      Q <= CLEAR8;
    elsif Clk='1' and Clk'event then
      Q <= D;
    end if;
    end dff;
  end my_reg8;
    
```

A.A. 2006/07 1 VHDL 207

Library

Raggruppamento di package e package body referenziabili da più file sorgente:

```

library mylib;
use mylib.my_package.all;
oppure
use mylib.my_package.dff;
    
```

Se non specificato, si assume che le design unit siano nella libreria work:

```

use work.my_package.all;
    
```

A.A. 2006/07 1 VHDL 208

Component

Usati per connettere più design unit tra di loro a formare un progetto gerarchico più grande.

```

entity shiftcompare is
  port (Clk, Rst, Load: in std_ulogic;
        Init: in std_ulogic_vector(0 to 7);
        Test: in std_ulogic_vector(0 to 7);
        Limit: out std_ulogic);
end shiftcompare;
    
```


A.A. 2006/07 1 VHDL 209

```

library ieee;
use ieee.std_logic_1164.all;

entity shiftcompare is
  port (Clk, Rst, Load: in std_ulogic;
        Init: in std_ulogic_vector(0 to 7);
        Test: in std_ulogic_vector(0 to 7);
        Limit: out std_ulogic);
end shiftcompare;
    
```

A.A. 2006/07 1 VHDL 210




```

architecture structure of shiftcompare is
  component compare
    port(A, B: in std_ulogic_vector(0 to 7); EQ: out std_ulogic);
  end component;
  component shift
    port(Clk, Rst, Load: in std_ulogic;
         Data: in std_ulogic_vector(0 to 7);
         Q: out std_ulogic_vector(0 to 7));
  end component;
  signal Q: out std_ulogic_vector(0 to 7);

begin
  COMP1: compare port map (A=>Q, B=>Test EQ=>Limit);
  SHIFT1: shift port map (Clk=>Clk, Rst=>Rst, Load=>Load, Data=>Init, Q=>Q);
end structure;
    
```

A.A. 2006/07 1 VHDL 211




Port mapping

Due tecniche:

- associazione posizionale
- U1: nand2 port map(a,b, y);
- associazione per nome
- U1: nand2 port map(a=>in1,b=>in2, y=>out1);

Vantaggio dell'associazione per nome: minori errori di scambio a parità di tipo.

A.A. 2006/07 1 VHDL 212




Generics

Generics: passaggio di informazioni specifiche per ciascuna istanziazione.
Esempio: batteria di flip-flop tipo D:

```

library ieee;
use ieee.std_logic_1164.all;
entity dffr is
  generic (wid: positive);
  port (Rst, Clk: in std_logic;
        signal D: in std_logic_vector(wid-1 downto 0);
        signal Q: out std_logic_vector(wid-1 downto 0));
end dffr;
    
```


A.A. 2006/07 1 VHDL 213



```

architecture behavior of dffr is
begin
  process(Rst, Clk)
    variable Qreg: std_logic_vector(wid-1 downto 0);
  begin
    if Rst='1' then
      Qreg := (others => '0');
    elsif Clk='1' and Clk'event then
      for i in Qreg'range loop
        Qreg(i) := D(i);
      end loop;
    end if;
    Q <= Qreg;
  end process;
end behavior;
    
```

A.A. 2006/07 1 VHDL 214




```

architecture sample of reg is
  component dffr
    generic(wid: positive);
    port (Rst, Clk: in std_logic;
          signal D: in std_logic_vector(wid-1 downto 0);
          signal Q: out std_logic_vector(wid-1 downto 0));
  end component;

  constant WID8: positive := 8;
  constant WID16: positive := 16;
  constant WID32: positive := 32;
  signal D8, Q8: std_logic_vector(7 downto 0);
  signal D16, Q16: std_logic_vector(15 downto 0);
  signal D32, Q32: std_logic_vector(31 downto 0);
    
```


A.A. 2006/07 1 VHDL 215



```

begin;
  FF8: dffr generic map(WID8) port map(Rst, Clk, D8, Q8);
  FF16: dffr generic map(WID16) port map(Rst, Clk, D16, Q16);
  FF32: dffr generic map(WID32) port map(Rst, Clk, D32, Q32);
end sample;
    
```

A.A. 2006/07 1 VHDL 216




Configuration

Caratteristica del VHDL per la gestione di progetti grandi e complessi.

Configuration declaration: design unit primitiva che dichiara il collegamento tra tutte (alcune) le istanziazioni di componenti e le entità di livello inferiore.

A.A. 2006/07 1 VHDL 217




Esempio

Entity: board
 Architecture: structure
 All'interno dell'architecture structure si trova un'istanziamento di un component chip

```
architecture structure of board is
  component chip port(...);
begin
  U1: chip port map(...);
end structure;
```

A.A. 2006/07 1 VHDL 218




L'entity chip è realizzata con due architecture alternative A1 e A2:

```
architecture A1 of chip is
  ...
end A1;

architecture A2 of chip is
  ...
end A2;
```

A.A. 2006/07 1 VHDL 219



Configuration permette di stabilire quale architecture realizza chip:

```
configuration this_build of board is
  for structure
    for U1: chip
      use entity work.chip(A1);
      port map (reset => gnd, cin => 1, ...);
    end for;
  end for;
end this_build;
```

A.A. 2006/07 1 VHDL 220