Introduzione al linguaggio VHDL

Motivazioni

- Introdurre i costrutti base del VHDL
- Introdurre il ciclo di simulazione del VHDL e il suo modello timing
- Illustrare l'utilita' del VHDL come linguaggio di descrizione dell'hardware digitale (HDL – Hardware Description Language)

Sommario – I

- Introduzione
- Esempio di progetto in VHDL
- Componenti di un modello VHDL
  - Dichiarazione di *entity*
  - Descrizione di *architecture*
- Modello timing

Sommario – II

- Costrutti base del VHDL
  - Tipi di dato
  - Oggetti
  - Istruzioni sequenziali e concorrenti
  - *Packages* e librerie
  - Attributi
  - Operatori predefiniti
- Esempi
- Utilizzo di un simulatore freeware
- Riepilogo

Ragioni per l'utilizzo del VHDL

- Il VHDL e' uno standard internazionale dell'IEEE (IEEE 1076-1993 e poi 2008) per descrivere l'hardware digitale in ambito industriale
  - VHDL sta per VHSIC (Very High Speed Integrated Circuit) Hardware Description Language
- Il VHDL consente di modellare l' hardware dal livello gate a quello di sistema
- Il VHDL può supportare strumenti automatici per la sintesi e per la verifica
- Il VHDL mette a disposizione meccanismi per il progetto digitale e la sua documentazione

5

Storia del VHDL (I)

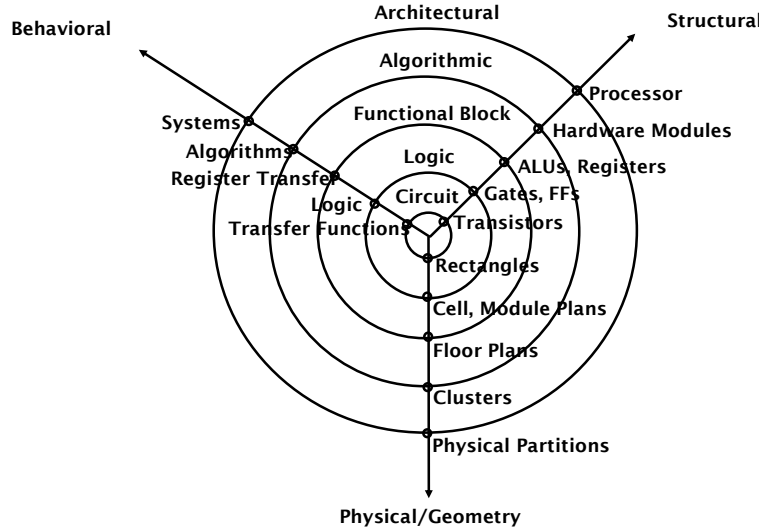
- Programma (DARPA) Very High Speed Integrated Circuit (VHSIC)
  - 1980
  - Sforzo significativo per avvanzare lo stato dell'arte nella tecnologia VLSI
  - Bisogno di un linguaggio di descrizione comune
  - $17M per lo sviluppo diretto del VHDL
  - $16M per la realizzazione di strumenti basati sul VHDL

6

Storia del VHDL (II)

- Nel 1983, Intermetrics, IBM and Texas Instruments furono incaricate di sviluppare il VHDL
- Nel 1985, rilascio della versione finale del linguaggio: VHDL Version 7.2
- Nel 1987, il VHDL divenne IEEE Standard 1076-1987 e nel 1988 ANSI standard
- Nel 1993, il VHDL fu ristandardizzato per semplificarne l'utilizzo e per potenziarlo
- Nuovi standard nel 2002 e 2008

Gajski e Kuhn's Y Chart



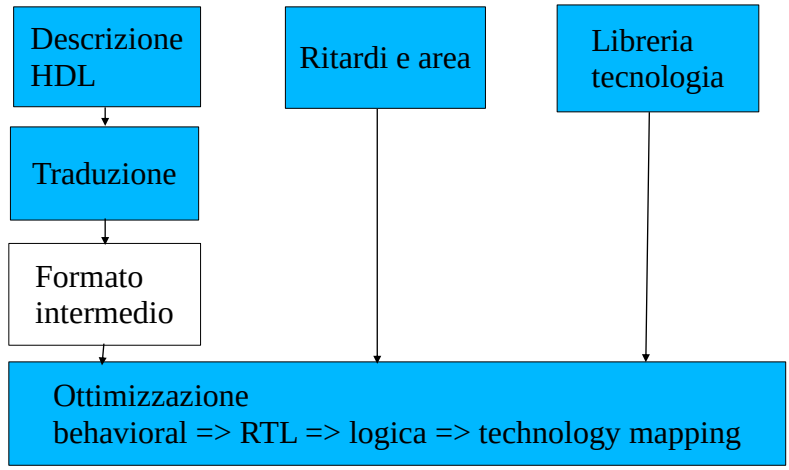
Vantaggi del VHDL

- E' indipendente dalla tecnologia
- Permette di descrivere una grande varieta' di hardware digitale a diversi livelli di astrazione
- Permette di usare diverse metodologie di progetto
- Facilita la comunicazione attraverso un linguaggio standard
- Permette una migliore gestione dei progetti
- E' particolarmente versatile
- Ha dato luogo a standard :
  - WAVES, VITAL, Analog VHDL

Linguaggi di descrizione dell'hardware – Linguaggi di descrizione del software

- Si descrive una rete logica come insieme di componenti interconnessi o mediante il suo comportamento
- Sintesi: realizzazione del circuito mediante una specifica tecnologia
- Simulazione: realizzazione di un modello utilizzabile per predire il comportamento del sistema per un dato insieme di stimoli
- Si descrive un algoritmo come sequenza di operazioni eseguite da una specifica architettura hardware (Von-Neumann – o sistemi distribuiti)
- Compilazione: traduzione del programma nel linguaggio macchina di una specifica CPU

Sintesi logica (I)



11

Sintesi logica (II)

- Insieme di operazioni da svolgere per passare dalle specifiche a una descrizione livello gate della rete
- Ciascun passo di sintesi è accompagnato da un processo di ottimizzazione, ovvero una ricerca di una soluzione soddisfacente in uno spazio le cui coordinate sono:
  - Costo (area)
  - Prestazioni (ritardo, throughput)
  - Affidabilita'
  - Consumo di Potenza
- Tali aspetti sono fra loro correlati
- La complessita' dei sistemi in esame richiede l'utilizzo di approcci automatici EDA (Electronic Design Automation)
- Questi algoritmi sono trattati in parte in questo corso
  - Problemi di tempo
  - Si puo' programmare (relativamente bene) in C senza conoscere i dettagli dei compilatori, in VHDL e' piu' difficile

12

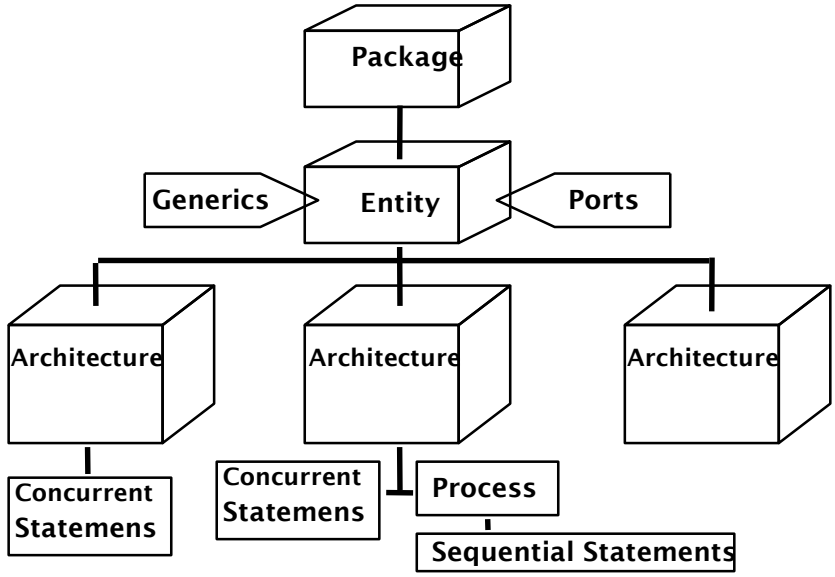
Verifica

- La verifica di progetto procede in senso inverso alla sintesi logica e puo' essere fatta in due modi diversi
- Si puo' voler verificare la funzionalita' di un circuito digitale, ma anche valutarne le prestazioni o l'affidabilita'
- Simulazione
  - In generale produce risultati parziali (computazionalmente esponenziale)
- Verifica formale
  - Quando e' computazionalmente fattibile, produce risultati completi
- Esempio: il teorema di Pitagora
  - provare diversi triangoli rettangoli e verificare la relazione (simulazione)
  - prova formale del teorema

Struttura di un progetto VHDL

- Un progetto complesso descritto in VHDL ha dei punti in comune con la descrizione di un programma in un linguaggio ad alto livello
- Ad esempio puo' essere importante utilizzare un approccio strutturato alla programmazione
- Diversamente dal software, la descrizione di una rete puo' passare attraverso diversi stadi nei quali alcune parti del progetto vengono successivamente specificate con maggiori dettagli
- Possono essere possibili differenti viste dello stesso oggetto

Visione di insieme



15

Esempio di progettoVHDL

- **Problema: progettare un half adder con un segnale di enable**
- **Specifiche**
  - **Ingressi e uscite sono di un bit**
  - **Quando l'enable e' alto, l'uscita result vale $(x + y)_{mod\ 2}$**
  - **Quando l'enable e' alto, l'uscita carry fornisce il riporto**
  - **Le uscite sono zero quando l'enable e' basso**

16

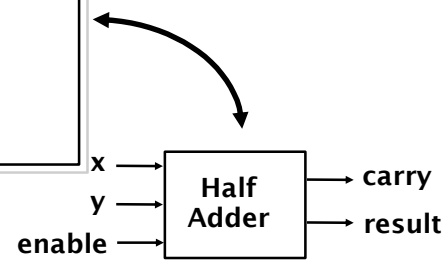
Esempio di progetto VHDL
Dichiarazione di entity

- **Primo passo, la *entity declaration* descrive l'interfaccia del componente**
  - **porte di input e output**
  - **notare che il VHDL e' case unsensitive: le maiuscole si utilizzano solo per evidenziare le parole riservate del linguaggio**

```
ENTITY half_adder IS

PORT( x, y, enable: IN BIT;
      carry, result: OUT BIT);

END ENTITY half_adder;
```



17

Esempio di progetto VHDL
Specifiche behavioral

- **Una descrizione ad alto livello dell' half-adder puo' essere fornita in questo modo:**

```
ARCHITECTURE half_adder_a OF half_adder IS
  BEGIN
    PROCESS (x, y, enable)
        BEGIN
        IF enable = '1' THEN
          result <= x XOR y;
          carry <= x AND y;
        ELSE
          carry <= '0';
          result <= '0';
        END IF;
    END PROCESS;
  END ARCHITECTURE half_adder_a;
```

- **Questa descrizione puo' essere simulata**

18

Esempio di progetto VHDL
Data Flow

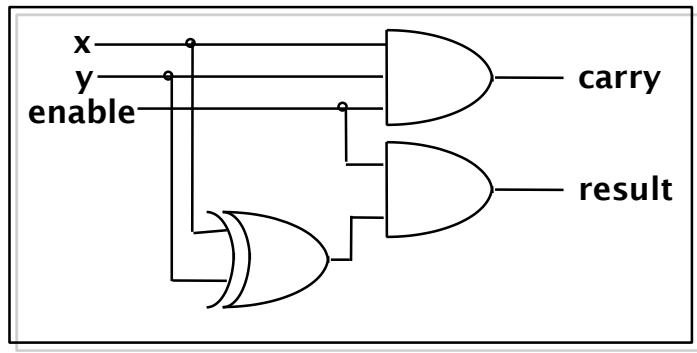
- Un secondo metodo consiste nell'utilizzo di espressioni logiche per sviluppare la descrizione del data flow

```
ARCHITECTURE half_adder_b OF half_adder IS
  BEGIN
    carry <= enable AND (x AND y);
    result <= enable AND (x XOR y);
END ARCHITECTURE half_adder_b;
```

- Il modello risulta anch'esso simulabile

Esempio di progetto VHDL
Strutturale

- Come ulteriore alternativa, si puo' utilizzare una descrizione strutturale



- Questi gate possono essere presi da una libreria di componenti

Esempio di progetto VHDL
Strutturale


```
ARCHITECTURE half_adder_c OF half_adder IS

COMPONENT and2
        PORT (in0, in1 : IN BIT;
              out0 : OUT BIT);
END COMPONENT;

COMPONENT and3
        PORT (in0, in1, in2 : IN BIT;
              out0 : OUT BIT);
END COMPONENT;

COMPONENT xor2
        PORT (in0, in1 : IN BIT;
              out0 : OUT BIT);
END COMPONENT;

FOR ALL : and2 USE ENTITY gate_lib.and2_nty(and2_a);
FOR ALL : and3 USE ENTITY gate_lib.and3_nty(and3_a);
FOR ALL : xor2 USE ENTITY gate_lib.xor2_nty(xor2_a);

-- description is continued on next slide
```

21

Esempio di progetto VHDL
Strutturale


```
-- continuing half_adder_c description

SIGNAL xor_res : BIT; -- internal signal
-- Note that other signals are already declared in entity

BEGIN

        A0 : and2 PORT MAP (enable, xor_res, result);
        A1 : and3 PORT MAP (x, y, enable, carry);
        X0 : xor2 PORT MAP (x, y, xor_res);

END ARCHITECTURE half_adder_c;
```

22

Decoder behavioral con enable


```
ENTITY decoder_2_to_4 IS
      PORT (E_n, A0, A1: IN BIT;
            O0, O1, O2, O3: OUT BIT);
END decoder_2_to_4;
ARCHITECTURE behavioral OF decoder_2_to_4 IS
BEGIN
     PROCESS (E_n, A0, A1)
     BEGIN
        IF E_n='1' THEN
          O0 <= (NOT A0) AND (NOT A1);
          O1 <= A0 AND (NOT A1);
          O2 <= (NOT A0) AND A1;
          O3 <= A0 AND A1;
        ELSE
          O0 <= '0';
          O1 <= '0';
          O2 <= '0';
          O3 <= '0';
        END IF;
     END PROCESS;
END ARCHITECTURE decoder_2_to_4;
```

23

Decoder dataflow


```
ARCHITECTURE dataflow OF decoder_2_to_4 IS
SIGNAL nA0,nA1: BIT;
BEGIN
    nA0 <= not A0;
    nA1 <= not A1;
    O0 <= nA0 AND nA1 AND E_n;
    O1 <= A0 AND nA1 AND E_n;
    O2 <= nA0 AND A1 AND E_n;
    O3 <= A0 AND A1 AND E_n;

END ARCHITECTURE dataflow;
```

24

Decoder strutturale

interfaccia

riferimento ai componenti

```
-- 2-to-4 Line Decoder: Structural VHDL Description

LIBRARY ieee, logic_lib;
USE ieee.std_logic_1164.all, logic_lib.gates.all;
ENTITY decoder_2_to_4 is
   PORT(E, A0, A1: IN std_logic;
        D0, D1, D2, D3: OUT std_logic);
END ENTITY decoder_2_to_4;


ARCHITECTURE structural_1 OF decoder_2_to_4 IS

COMPONENT not1
   port(in1: in std_logic;
        out1: out std_logic);
END COMPONENT;

COMPONENT and3
   port(in1, in2, in3: in std_logic;
        out1: out std_logic);
END COMPONENT;

FOR ALL: not1 USE ENTITY gate_lib.not1_nty(not1_a);
FOR ALL: and3 USE ENTITY gate_lib.and3_nty(and3_a);
```


Decoder strutturale

istanza dei componenti

```
SIGNAL not_A0, not_A1: std_logic;
BEGIN
   g0: NOT1 PORT MAP (in1 => A0, out1 => not_A0);
   g1: NOT1 PORT MAP (in1 => A1, out1 => not_A1);
   g2: AND3 PORT MAP (in1 => not_A0, in2 => not_A1,
                      in3 => E, out1 => D0);
   g3: AND3 PORT MAP (in1 => A0, in2 => not_A1,
                      in3 => E, out1 => D1);
   g4: AND3 PORT MAP (in1 => not_A0, in2 => A1,
                      in3 => E, out1 => D2);
   g5: AND3 PORT MAP (in1 => A0, in2 => A1,
                      in3 => E, out1 => D3);
END ARCHITECTURE structural_1;
```

MPX 4-to-1 Strutturale


```
-- 4-to-1 Line Multiplexer: Structural VHDL Description
--
library ieee, logic_lib;
use ieee.std_logic_1164.all, logic_lib.gates.all;
entity multiplexer_4_to_1_st is
   port(S: in std_logic_vector(0 to 1);
        D: in std_logic_vector(0 to 3);
        Y: out std_logic);
end multiplexer_4_to_1_st;

architecture structural_2 of multiplexer_4_to_1_st is

component NOT1
   port(in1: in std_logic;
        out1: out std_logic);
end component;

component AND3
   port(in1, in2, in3: in std_logic;
        out1: out std_logic);
end component;

component OR4
   port(in1, in2, in3, in4: in std_logic;
        out1: out std_logic);
end component;

-- binding (omitted)
signal not_S: std_logic_vector(0 to 1);
signal N: std_logic_vector(0 to 3);
```

27

MPX 4-to-1 Strutturale


```
begin
   g0: NOT1 port map (S(0), not_S(0));
   g1: NOT1 port map (S(1), not_S(1));
   g2: AND3 port map (not_S(1), not_S(0), D(0), N(0));
   g3: AND3 port map (not_S(1), S(0), D(1), N(1));
   g4: AND3 port map (S(1), not_S(0), D(2), N(2));
   g5: AND3 port map (S(1), S(0), D(3), N(3));
   g6: OR4 port map (N(0), N(1), N(2), N(3), Y);
end structural_2;
```

28

MPX 4-to-1 data flow (when-else)


```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY multiplexer_4_to_1_we IS
     PORT (S : IN std_logic_vector(1 downto 0);
           D : IN std_logic_vector(3 downto 0);
           Y : OUT std_logic);
END ENTITY multiplexer_4_to_1_we;

ARCHITECTURE function_table OF
multiplexer_4_to_1_we IS
BEGIN
Y <=   D(0) WHEN S = "00" ELSE
       D(1) WHEN S = "01" ELSE
       D(2) WHEN S = "10" ELSE
       D(3) WHEN S = "11" ELSE
       'X';
END ARCHITECTURE function_table;
```

29

MPX 4-to-1 data flow (with-select)


```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY multiplexer_4_to_1_ws IS
     PORT (S : IN std_logic_vector(1 downto 0);
           D : IN std_logic_vector(3 downto 0);
           Y : OUT std_logic);
END ENTITY multiplexer_4_to_1_ws;

ARCHITECTURE function_table_ws OF
multiplexer_4_to_1_ws IS
BEGIN
WITH S SELECT
Y <=   D(0) WHEN "00",
       D(1) WHEN "01",
       D(2) WHEN "10",
       D(3) WHEN "11",
       'X' WHEN OTHERS;
END ARCHITECTURE function_table_ws;
```

30

Modelli dei componenti VHDL

- Una descrizione completa di un componente VHDL richiede una VHDL *entity* e una VHDL *architecture*
  - La entity definisce l'interfaccia dei componenti
  - L'architettura definisce la funzione dei componenti
- Diverse possibili architetture possono essere utilizzate per la stessa entity
- Tre diversi tipi di descrizione di un componente in VHDL:
  - Strutturale
  - Comportamentale (Behavioral)
  - Timing e delay

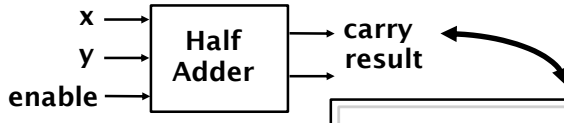
Modelli di componenti VHDL

- L'unita' fondamentale per la descrizone del behavior di un componente e' il *processo*
  - I processi possono essere definiti esplicitamente o implicitamente e sono contenuti nelle architetture
- Il meccanismo primario di comunicazione tra processi e' il *segnale*
  - L'esecuzione di un processo risulta in nuovi valori assegnati a segnali che sono poi accessibili ad altri processi
  - Un segnale puo' essere reso accessibile a processi in altre architetture connettendo il segnale a porte nelle entities associate alle due architetture
  - Esempio di istruzione di assegnamento di un segnale:

```
Output <= My_id + 10;
```

Dichiarazione di Entity

- Lo scopo primario di una entity e' di dichiarare i segnali nell'interfaccia del componente
  - Questi segnali sono elencati nella dichiarazione PORT
    - La *entity* e' simile al simbolo schematico del componente




```
ENTITY half_adder IS
  GENERIC(prop_delay : TIME := 10 ns);
  PORT( x, y, enable : IN BIT;
carry, result : OUT BIT);
END half_adder;
```


Dichiarazione di Entity

Port

- PORT dichiara i segnali di interfaccia verso il mondo esterno
- Nella PORT sono presenti tre campi
  - Nome
  - Modo
  - Tipo di dato

```
PORT (signal_name : mode data_type);
```

- Esempio di clausola PORT:

```
PORT ( input : IN BIT_VECTOR(3 DOWNTO 0);
       ready, output : OUT BIT );
```

- Segnali di interfaccia (i.e. 'ports') dello stesso modo e tipo o sottotipo si possono dchiarare nella stessa linea

Dichirarzione di Entity
Port

- **Il modo descrive la direzione in cui il dato viaggia rispetto al *componente***
- **Ci sono cinque modi disponibili per una porta:**
  - **In – il dato entra nella porta e si puo' solo leggere**
  - **Out – il dato esce dalla porta**
  - **Buffer – il dato si puo' muovere in entrambe le direzioni, ma solamente un driver alla volta puo' essere attivo**
  - **Inout – il dato si puo' muovere in entrambe le direzioni, con un qualsiasi numero di drivers attivi; richiede una Bus Resolution Function**
  - **Linkage – la direzione del data flow e' sconosciuta**

35

Dichiarazioni di Entity
Generic

- **Le dichiarazioni di Generics si possono usare per leggibilita', manutenzione e configurazione**
- **Sintassi :**

```
GENERIC (generic_name : type [:= default_value]);
```

  - Il default_value e' opzionale, se manca nella dichiarazione, deve essere presente quando il componente e' usato (*instanziato*)
- **Esempio :**

```
GENERIC (My_ID : INTEGER := 37);
```

  - Il generic *My_ID,* ha un valore di default di 37, e puo' essere referenziato da qualsiasi architettura della entity con questa generic
  - Il valore di default puo' essere sovrascritto quando il componente viene instanziato

36

Corpo delle Architetture

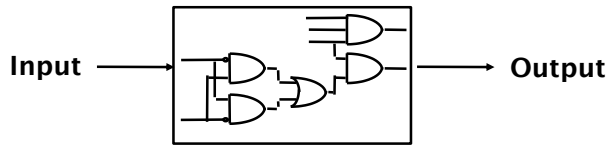
- **Descrive le operazioni del componente**
- **Consiste di due parti :**
  - **Parte dichiarativa – include le dichiarazioni necessarie**
    - e.g. dichiarazioni di tipo, di segnale, di componente, di sottoprogramma
  - **Parte di istruzioni -- include le istruzioni che descrivono l'organizzazione e/o la funzione del componente**
    - e.g. assegnamento concorrente di segnali, istruzioni di processo, instanze a componenti

```
ARCHITECTURE half_adder_d OF half_adder IS
  SIGNAL xor_res : BIT;     -- architecture declarative part
  BEGIN                     -- begins architecture statement part
    carry <= enable AND (x AND y);
    result <= enable AND xor_res;
    xor_res <= x XOR y;
  END ARCHITECTURE half_adder_d;
```

37

Descrizioni strutturali

- **I componenti VHDL predefiniti sono *instanziati e* connessi insieme**
- **Le descrizioni strutturali possono connettere semplici gate o componenti piu' complessi a loro volta descritti in maniera strutturale o comportamentale**
  - **Il VHDL supporta descrizioni gerarchiche**
  - **Il VHDL consente di descrivere facilmente strutture altamente ripetitive**



38

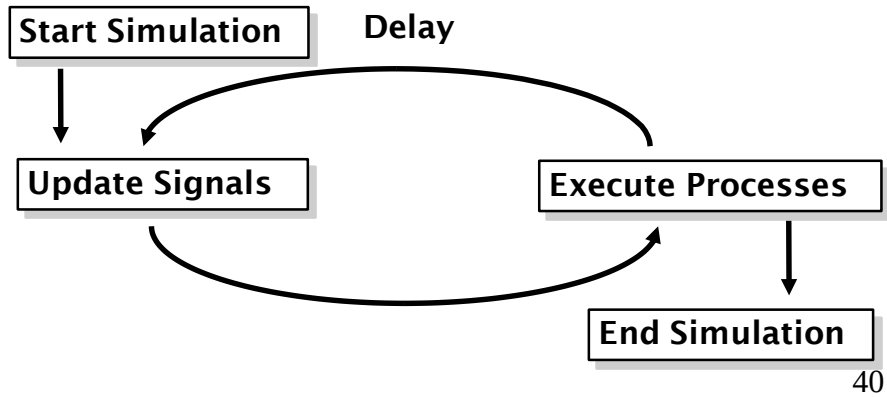
Descrizioni Comportamentali (Behavioral)

- Il VHDL mette a dispsizione due stili per descrivere il behavior di un componente
  - Data Flow: istruzioni concorrenti di assegnmento dei segnali
  - Behavioral: *processi* utilizzati per descrivere comportamenti complessi mediante costrutti di linguaggi ad alto livello
    - e.g. variabili, cicli, if-then-else
- Un modello behavioral model puo' essere largamente indipendente dall'implementazione del sistema

39

Modello Timing

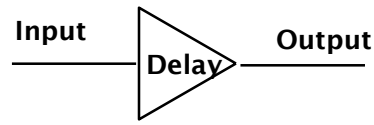
- Il VHDL utilizza il seguente ciclo per modellare gli stimoli e la risposta dell'hardware digitale



40

Tipi di ritardo

- Tutti gli assegnamenti di segnali del VHDL devono avere un ritardo non nullo prima che il segnale possa assumere il suo nuovo valore
- Questo ritardo puo' essere in una di queste tre forme:
  - Trasporto – definisce solo il ritardo di propagazione
  - Inerziale – definisce sia il ritardo di propagazione che la minima ampiezza di un impulso su ingresso che da luogo a una risposta sull'uscita
  - Delta -- valore di default se non si definisce alcun ritardo

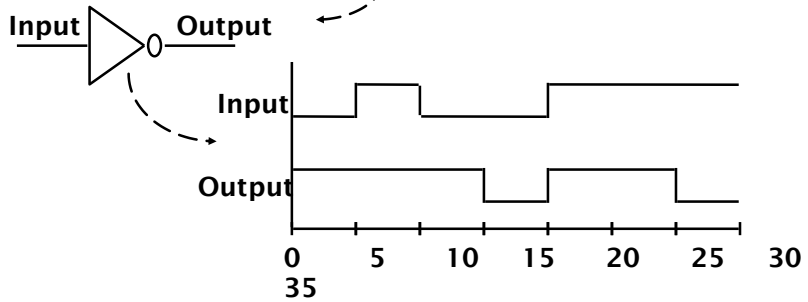


41

Ritardo di tipo trasporto

- Il ritardo di tipo trasporto deve essere specificato
  - I.e. keyword “TRANSPORT”
- Il segnale assume il suo nuovo valore dopo il ritardo specificato

```
-- TRANSPORT delay example
Output <= TRANSPORT NOT Input AFTER 10 ns;
```



42

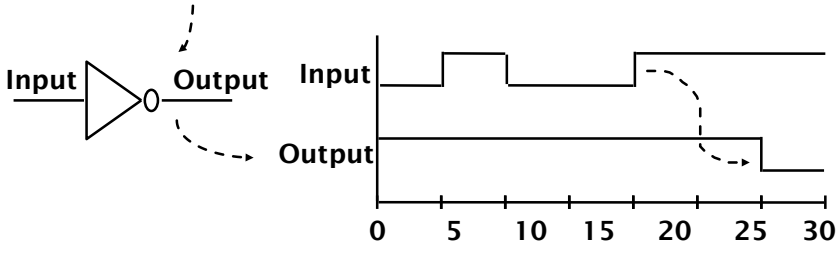
Ritardo inerziale

- Oltre a specificare il ritardo di propagazione, specifica un valore di soglia per la dimensione degli impulsi di ingresso al di sotto del quale non sono propagati:

```
Target <= [REJECT time_expression] INERTIAL waveform;
```

- Il ritardo inerziale e' di default e REJECT e' opzionale:

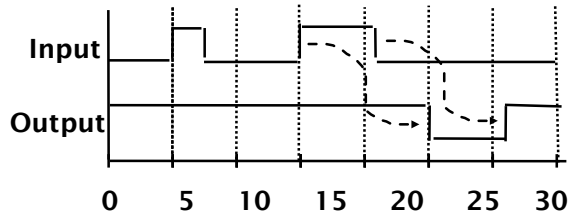
```
Output <= NOT Input AFTER 10 ns;
-- Propagation delay and minimum pulse width are 10ns
```



Ritardo inerziale (cont.)

- Esempio di gate con 'inerzia' piu' piccola del ritardo di propagazione
  - e.g. Invertitore con ritardo di propagazione di 10ns che sopprime impulsi < 5ns

```
Output <=  REJECT 5ns INERTIAL NOT Input AFTER 10ns;
```



Ritardo Delta

- Default se non si specifica niente
  - Nel VHDL l'assegnamento dei segnali non avviene immediatamente per evitare problemi di loop nel simulatore
  - Delta e' l'unita' minima di tempo del VHDL per i ritardi
  - E.g.

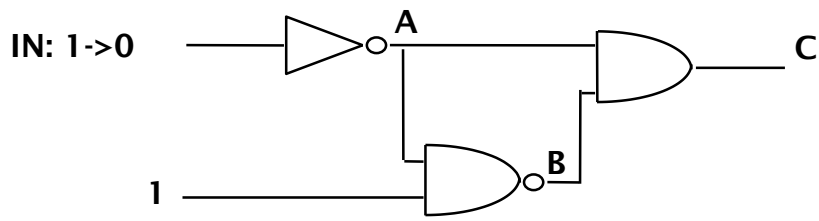
```
Output <= NOT Input;
-- Output assumes new value in one delta cycle
```

- Consente di supportare un modello concorrente di esecuzione dei processi VHDL
  - L'ordine in cui i processi sono eseguiti dal simulatore non cambia il risultato della simulazione

45

Ritardo Delta
Esempio senza ritardo Delta

- Qual'e' il comportamento di C?



NAND gate valutato prima:
IN: 1->0
A: 0->1
B: 1->0
C: 0->0

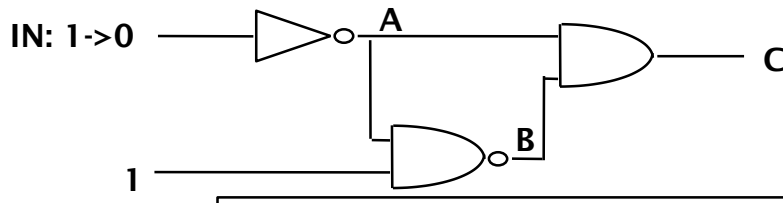
AND gate valutato prima:
IN: 1->0
A: 0->1
C: 0->1
B: 1->0
C: 1->0

46

Ritardo Delta

Esempio con Ritardo Delta

- Comportamento di C?



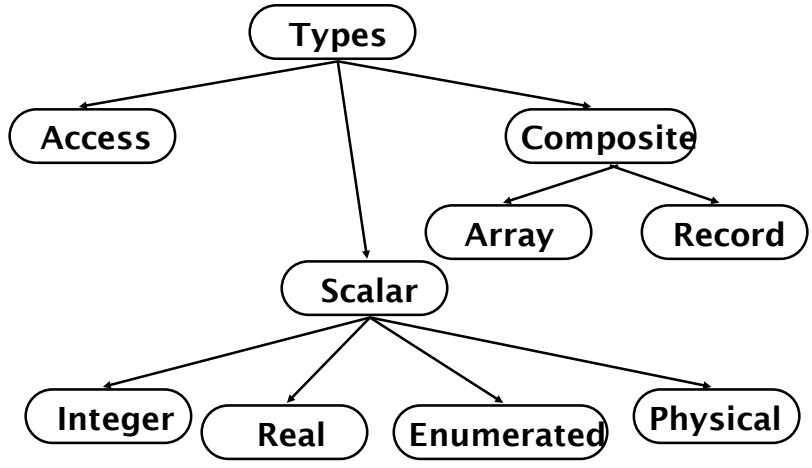
Utilizzo del ritardo delta

Time	Delta	Event
0 ns	1	IN: 1->0
		eval INVERTER
	2	A: 0->1
		eval NAND, AND
	3	B: 1->0, C: 0->1
		eval AND
	4	C: 1->0
1 ns		

47

Tipi di dato

- Nel VHDL tutte le dichiarazioni di porte, segnali, e variabili devono specificare il loro tipo o sottotipo



48

Tipi di dato VHDL
Tipi scalari

- **Interi**
  - Minimum range (standard): da - 2,147,483,647 a 2,147,483,647
  - Esempio di assegnamento a variabili :

```
ARCHITECTURE test_int OF test IS
BEGIN
PROCESS (X)
       VARIABLE a: INTEGER;
BEGIN
       a := 1;   -- OK
       a := -1;   -- OK
       a := 1.0;   -- illegal
END PROCESS;
END test_int;
```

49

Tipi di dato VHDL
Tipi Scalari (Cont.)

- **Real**
  - Minimum range (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;   -- OK
       a := -7.5;   -- OK
       a := 1;   -- illegal
       a := 1.7E13;   -- OK
       a := 5.3 ns;   -- illegal
END PROCESS;
END test_real;
```

50

Tipi di dato VHDL
Tipi scalari (Cont.)

- **Enumerated**
  - Lista di possibili valori forniti dall'utente
  - Esempio di dichiarazione :

```
TYPE binary IS ( ON, OFF );
... some statements ...
ARCHITECTURE test_enum OF test IS
BEGIN
PROCESS (X)
       VARIABLE a: binary;
BEGIN
       a := ON;  -- OK
       ... more statements ...
       a := OFF;  -- OK
       ... more statements ...
END PROCESS;
END test_enum;
```

51

Tipi di dato VHDL
Tipi scalari (Cont.)

- **Physical**
  - Richiede un unita' di misura associata
  - Si deve specificare un range
  - Esempio di dichiarazione :

```
TYPE resistance IS RANGE 0 TO 10000000

UNITS
ohm;  -- ohm
Kohm = 1000 ohm;  -- i.e. 1 KW
Mohm = 1000 kohm;  -- i.e. 1 MW
END UNITS;
```

  - Il tempo (Time) e' il solo tipo fisico predefinito VHDL standard

52

Tipi di dato VHDL
Tipi Compositi

- **Array**
  - Utilizzato per ragruppare elementi dello stesso tipo in un singolo oggetto VHDL
  - Il Range puo' essere unconstrained nella dischiarazione
    - Deve essere specificato quando si utilizza l'array
  - Esempio di dichiarazione per un array mono dimensionale (vettore) :

```
TYPE data_bus IS ARRAY(0 TO 31) OF BIT;
```

0 …element indices…31

0	…array values…	1

```
VARIABLE X : data_bus;
VARIABLE Y : BIT;

Y := X(12);  -- Y gets value of element at index 12
```


Tipi di dato VHDL Data
Tipi compositi (Cont.)

- **Esempio di array monodimensionale utilizzante DOWNTO :**

```
TYPE reg_type IS ARRAY(15 DOWNTO 0) OF BIT;
```

15…element indices…0

0	…array values…	1

```
VARIABLE X : reg_type;
VARIABLE Y : BIT;

Y := X(4);  -- Y gets value of element at index 4
```

- **La keyword DOWNTO deve essere utilizzata se l'indice di sinistra e' piu' grande di quello di destra**

Tipi di dato VHDL
Tipi compositi (Cont.)

- Records
  - Utilizzati per ragruppare elementi di tipi differenti in un singolo oggetto VHDL
  - Gli elementi sono referenziati mediante 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;  -- status of the switch
switch.IDnumber := 30;  -- e.g. number of the switch
```

55

Tipi di dato VHDL D
Tipo Accesso

- Access
  - Simile ai puntatori in altri linguaggi
  - Memorizzazione dinamica
  - Utile per realizzare a un livello astratto code, fifos, 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

56

Tipi di dato VHDL D
Tipo Accesso

- Esempio (FIFO)

```
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
```

57

Tipi di dato VHDL
Sottotipi

- Subtype
  - Permette di definire dei vincoli sui tipi di dato
    - e.g. un subtype si basa su tipi unconstrained
  - Possono includere l'intero range del tipo base
  - Assegnmenti fuori dal range del subtype sono illegali
    - Le violazioni di range sono rivelate run time e non alla compilazione dove si controllano solo i tipi
- Sintassi della dichiarazione :

```
SUBTYPE name IS base_type RANGE <user range>;
```

  - Esempio :

```
SUBTYPE first_ten IS INTEGER RANGE 0 TO 9;
```

58

Tipi di dato VHDL
Sommario

- **Tutte le dichiarazioni di porte, segnali e variabili VHDL devono includere il loro tipo e sottotipo**
- **I tipi di dato VHDL possono manifestarsi in 3 forme:**
  - **Access – puntatori per allocaziopne dinamica**
  - **Scalar -- Integer, Real, Enumerated, e Physical**
  - **Composite -- Array, e Record**
- **Il VHDL mette a disposizione un insieme di tipi dato predefiniti**
  - **L'utilizzatore ne puo' aggiungere**

VHDL Oggetti

- **Ci sono 4 tipi di oggetti nel VHDL**
  - **Costanti**
  - **Variabili**
  - **Segnali**
  - **File**
- **Lo** *scope* di un oggetto nel VHDL e' 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

Oggetti VHDL
Costanti

- Nome assegnato a un valore specifico di un tipo
- Aggiornamento e leggibilita'
- La dichiarazione di una costante puo' omettere il valore cosi' da deferire l'assegnamento
  - Riconfigurazione
- Sintassi della dichiarazione :

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

- Esempio :

```
CONSTANT PI : REAL := 3.14;
CONSTANT SPEED : INTEGER;
```

61

Oggetti VHDL
Variabili

- Meccanismo per la memorizzazione locale
  - E.g. contatori, valori intermedi
- Lo *scope e'* il processo in cui sono dichiarate
  - VHDL '93 mette a disposizione variabili globali, da discutere nei prossimi moduli
- Tutte 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";
VARIABLE freq : INTEGER;
```

62

Oggetti VHDL
Segnali

- Comunicazione fra componenti VHDL
- I segnali reali dei sistemi sono spesso mappati su segnali VHDL
- Tutti gli assegnamenti di segnali VHDL richiedono o un ritardo di tipo delta o specificato dall'utilizzatore prima che il nuovo valore sia assunto
- Sintassi della dichiarazione :

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

- Esempio :

```
SIGNAL brdy : BIT;
brdy <= '0' AFTER 5ns, '1' AFTER 10ns;
```


Segnali e variabili

- Differenza fra variabili e segnali

```
ARCHITECTURE test1 OF mux 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 test1;
```

```
ARCHITECTURE test2 OF mux 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 test2;
```

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

Oggetti VHDL
Segnali vs Variabili

- La differenza chiave fra variabili e segnali sta nel ritardo degli assegnamenti

```
ARCHITECTURE sig_ex OF test IS
PROCESS (a, b, c, out_1)
BEGIN
       out_1 <= a NAND b;
       out_2 <= out_1 XOR c;
END PROCESS;
END sig_ex;
```

Time	a	b	c	out_1	out_2
0	0	1	1	1	0
1	1	1	1	1	0
1+d	1	1	1	0	0
1+2d	1	1	1	0	1

65

Oggetti VHDL
Segnali vs Variabili (Cont.)

```
ARCHITECTURE var_ex 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 var_ex;
```

Time	a	b	c	out_3	out_4
0	0	1	1	1	0
1	1	1	1	0	0
1+d	1	1	1	0	1

66

Oggetti VHDL
File

- I file danno modo a un progetto VHDL di comunicare con l'ambiente esterno
- Dichiarazioni di file
- I file possono essere aperti in lettura o in scrittura
  - Nel VHDL87, i files son aperti e chiusi quando gli oggetti associati entrano o escono dallo *scope corrente*
  - Nel VHDL93 sono state aggiunte procedure esplicite `FILE_OPEN()` and `FILE_CLOSE()`
- Il package STANDARD definisce le routine base di I/O per i tipi VHDL
- Il package TEXTIO definisce routine piu' potenti per la gestione I/O di file di testo

Ancora sul ciclo di simulazione
Istruzioni Sequenziali vs Concorrenti

- Il VHDL e' 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 flessibilita'
  - Supportano vari livelli di astrazione
  - Supportano la modellistica di eventi concorrenti e sequenziali come si osserva nei sistemi reali

Istruzioni Concorrenti

- La granularita' base della concorrenza e' il *process*
- Meccanismo per avere la concorrenza :
  - 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
    - I.e. 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

69

Istruzioni sequenziali

- Le istruzioni dentro un *process* vengono eseguite sequenzialmente

```
ARCHITECTURE sequential OF test_mux IS
BEGIN
select_proc : PROCESS (x,y)
BEGIN
IF (select_sig = '0') THEN
  z <= x;
ELSIF (select_sig = '1') THEN
     z <= y;
ELSE
          z <= "XXXX";
END IF;
END PROCESS select_proc;
END sequential;
```

70

Packages e Librerie

- I costrutti dichiarati dentro architetture e entities non sono visibili ad altri VHDL componenti
  - Lo *scope* di sottoprogrammi, tipi di dato definiti dall'utilizzatore, costanti, e segnali e' limitato ai componenti VHDL in cui sono dichiarati

- Packages e librerie danno la capacita' di riutilizzare costrutti in piu' entities e architetture
  - Gli oggetti dichiarati nei packages possono essere utilizzati (i.e. inclusi) in altri componenti VHDL

Packages

- Consistono di due parti
  - Dichiarazione -- contiene le diclarazioni di oggetti definiti nel package
  - Body -- contiene le definizioni necessarie per certi oggetti nella dichiarazione
    - e.g. Dichiarazioni di sottoprogrammi
  - Esempi di oggetti VHDL contenuti in packages :
  - Dichiarazioni base
    - Types, subtypes
    - Costanti
    - Sottoprogrammi
    - Use clause
  - Segnali
  - Attributi
  - Componenti

Packages
Dichiarazione

- Un esempio di dichiarazione di package :

```
PACKAGE prova IS
TYPE binary IS ( ON, OFF );
CONSTANT PI : REAL := 3.14;
CONSTANT My_ID : INTEGER;
PROCEDURE add_bits3(SIGNAL a, b, en : IN BIT;
                    SIGNAL temp_result, temp_carry : OUT BIT);
END prova;
```

- Alcuni oggetti richiedono dichiarazioni mentre altri abbisognano di ulteriori dettagli da specificare nel body
  - per la definizione di tipi e sottotipi, la dichiarazione e' sufficiente
  - I sottoprogrammi richiedono la descrizione

73

Packages
Package Body

- Il body include le descrizioni funzionali degli oggetti dichiarati
  - e.g. Descrizioni di sottoprogrammi, assegnamento a costanti

```
PACKAGE BODY prova IS
CONSTANT My_ID : INTEGER := 2;

PROCEDURE add_bits3(SIGNAL a, b, en : IN BIT;
                    SIGNAL temp_result, temp_carry : OUT BIT) IS
BEGIN    -- this function can return a carry
             temp_result <= (a XOR b) AND en;
             temp_carry <= a AND b AND en;
END add_bits3;
END prova;
```

74

Packages
Clausola di utilizzo

- I packages devono essere resi visibili prima di utilizzarne il contenuto
  - La clausola USE rende i packages visibili a entities, architetture, e ad altri packages

```
-- use only the binary and add_bits3 declarations
USE my_stuff.binary, prova.add_bits3;

... ENTITY declaration...
... ARCHITECTURE declaration ...
```

```
-- use all of the declarations in package my_stuff
USE prova.ALL;

... ENTITY declaration...
... ARCHITECTURE declaration ...
```

75

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
  - E.g. 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 diverse librerie sono fornite dai venditori di simulatori o di moduli IP
    - E.g. Librerie proprietarie e IEEE standard

76

Attributi

- Gli attributi mettono a disposizione informazioni su certi oggetti in VHDL
  - E.g. tipi, sottotipi, procedure, funzioni, segnali, variabili, costanti, entities, architetture, configurazioni, packages, componenti
- Forma generale :

```
name'attribute_identifier  -- read as "tick"
```

- Il VHDL ha diversi attributi predefiniti, ad esempio :
  - X'EVENT -- TRUE quando c'e' un evento sul segnale X
  - X'LAST_VALUE -- ritorna il valore precedente di X
  - Y'HIGH – ritorna il valorepiu' alto nel range di Y
  - X'STABLE(t) -- TRUE quando nessun evento e' avvenuto sul segnale X nel tempo 't'

77

Attributi
Esempio di registro

- L'esempio seguente mostra come gli attributi si possano utilizzare per fare un registro a 8-bit
- Specifiche :
  - Campionamento sul fronte di salita del clock
  - Memorizzazione solo sul segnale di enable alto
  - Data setup time = x_setup
  - Propagation delay = prop_delay

```
ENTITY 8_bit_reg IS
GENERIC (x_setup, prop_delay : TIME);
PORT(enable, clk : IN level;
     a : IN level_vector(7 DOWNTO 0);
     b : OUT level_vector(7 DOWNTO 0));
END 8_bit_reg;
```

- il tipo qsim_state e' utilizzato – valori 0, 1, X, e Z

78

Attributi
Registro (Cont.)

- Primo tentativo di definizione dell'architettura
- 'STABLE e' utilizzato per rivelare violazioni del tempo di setup

```
ARCHITECTURE first_attempt OF 8_bit_reg IS
BEGIN
PROCESS (clk)
BEGIN
IF (enable = '1') AND a'STABLE(x_setup) AND
   (clk = '1') THEN
b <= a AFTER prop_delay;
END IF;
END PROCESS;
END first_attempt;
```

- Cosa succede se *a* non soddisfa il tempo di setup?

79

Attributi
Esempio di registro (Cont.)

- L'esempio precedente non funziona correttamente
- L'utilizzo di 'LAST_VALUE assicura che il clock abbia una transizione dal valore '0'

```
ARCHITECTURE behavior OF 8_bit_reg IS
BEGIN
PROCESS (clk)
BEGIN
IF (enable = '1') AND (clk = '1')
    AND (clk'LAST_VALUE = '0') THEN
IF (a'stable(x_setup)) THEN
  b <= a AFTER prop_delay;
ELSE
  b <= 'X' AFTER prop_delay;
END IF;
END IF;
END PROCESS;
END behavior;
```

80

Registro

- Problema del tempo di hold
- Cosa succede se e' il segnale di enable a violare il tempo di setup?

Operatori

- Espressioni complesse :

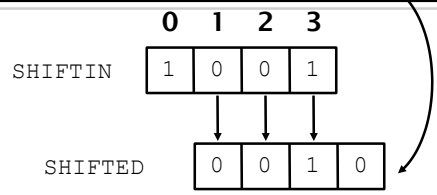
```
res <= a AND NOT(B) OR NOT(a) AND b;
```

  - Possono utilizzare parentesi
- Livelli di precedenza in ordine decrescente :
  - Miscellaneous operators -- **, abs, not
  - Multiplication operators -- *, /, mod, rem
  - Sign operator -- +, -
  - Addition operators -- +, -, &
  - Shift operators -- sll, srl, sla, sra, rol, ror
  - Relational operators -- =, /=, <, <=, >, >=
  - Logical operators -- AND, OR, NAND, NOR, XOR, XNOR

Operatori
Esempi

- L'operatore di concatenamento &

```
VARIABLE shifted, shiftin : BIT_VECTOR(0 TO 3);
...
shifted := shiftin(1 TO 3) & '0';
```



- L'operatore esponente **

```
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
```


Esempi

- Costruire una libreria di gate logici
  - AND, OR, NAND, NOR, INV, etc.
- Includere istruzioni sequenziali
  - DFF, Registri, etc.
- Includere dispositivi tri-state
- Utilizzare logica a 4-valori
  - 'X', '0', '1', 'Z'
  - Dichiarazioni globali in un package

Global Package


```
PACKAGE resources IS

   TYPE level IS ('X', '0', '1', 'Z'); -- enumerated type

   TYPE level_vector IS ARRAY (NATURAL RANGE <>) OF level;
   -- type for vectors (buses)

   SUBTYPE delay IS TIME; -- subtype for gate delays

   -- Function and procedure declarations go here

END resources;
```

85

AND gate a due ingressi


```
USE work.resources.all;

ENTITY and2 IS

  GENERIC(trise : delay := 10 ns;
          tfall : delay := 8 ns);

  PORT(a, b : IN level;
       c : OUT level);

END and2;
```

```
ARCHITECTURE behav OF and2 IS

  BEGIN

    one : PROCESS (a,b)

      BEGIN
        IF (a = '1' AND b = '1') THEN
          c <= '1' AFTER trise;
        ELSIF (a = '0' OR b = '0') THEN
          c <= '0' AFTER tfall;
        ELSE
          c<= 'X' AFTER (trise+tfall)/2;
        END IF;

    END PROCESS one;

END behav;
```

86

Tri-State Buffer


```
USE work.resources.all;

ENTITY tri_state IS

  GENERIC(trise : delay := 6 ns;
          tfall : delay := 5 ns;
          thiz  : delay := 8 ns);

  PORT(a : IN level;
       e : IN level;
       b : OUT level);

END tri_state;
```

```
ARCHITECTURE behav OF tri_state IS

  BEGIN

    one : PROCESS (a,e)

      BEGIN
        IF (e = '1' AND a = '1') THEN
                -- enabled and valid data
          b <= '1' AFTER trise;
        ELSIF (e = '1' AND a = '0') THEN
          b <= '0' AFTER tfall;
        ELSIF (e = '0') THEN -- disabled
          b <= 'Z' AFTER thiz;
        ELSE -- invalid data or enable
          b <= 'X' AFTER (trise+tfall)/2;
        END IF;

    END PROCESS one;

END behav;
```

87

D Flip Flop


```
USE work.resources.all;

ENTITY dff IS

  GENERIC(tprop : delay := 8 ns;
          tsu   : delay := 2 ns);

  PORT(d      : IN level;
       clk    : IN level;
       enable : IN level;
       q      : OUT level;
       qn     : OUT level);

END dff;
```

```
ARCHITECTURE behav  OF dff IS
  BEGIN
   one : PROCESS (clk)
     BEGIN
       -- check for rising clock edge
       IF ((clk = '1' AND clk'LAST_VALUE = '0')
            AND enable = '1') THEN -- ff enabled
         -- first, check setup time requirement
         IF (d'STABLE(tsu)) THEN
           -- check valid input data
           IF (d = '0') THEN
             q <= '0' AFTER tprop;
             qn <= '1' AFTER tprop;
           ELSIF (d = '1') THEN
             q <= '1' AFTER tprop;
             qn <= '0' AFTER tprop;
           ELSE -- else invalid data
             q <= 'X';
             qn <= 'X';
           END IF;
         ELSE -- else violated setup time requirement
           q <= 'X';
           qn <= 'X';
         END IF;
       END IF;
   END PROCESS one;
END behav;
```

88

Sommario

- Il VHDL e' uno standard internazionale per la descrizione e il modeling dell'hardware digitale
- Il VHDL da al progettista molti modi differenti per descrivere l'hardware
- Disponibilita' di tool di simulazione
- Modelli di esecuzione sequenziali e concorrenti modes of execution
- Package e librerie supportano la manutenzione e il riutilizzo del software

Riferimenti

[Bhasker95] Bhasker, J. A VHDL Primer, Prentice Hall, 1995.

[Calhoun95] Calhoun, J.S., Reese, B.,. *"Class Notes for EE-4993/6993: Special Topics in Electrical Engineering (VHDL)"*, Mississippi State University, http://www.erc.msstate.edu/mpl/vhdl-class/html, 1995.

[Coelho89] Coelho, D. R., The VHDL Handbook, Kluwer Academic Publishers, 1989.

[Gajski83] Gajski, Daniel D. and Kuhn, Robert H., "Guest Editors Introduction - New VLSI Tools", IEEE Computer, pp 11-14, IEEE, 1983

[Lipsett89] Lipsett, R., C. Schaefer, C. Ussery, VHDL: Hardware Description and Design, Kluwer Academic Publishers, , 1989.

[LRM94] IEEE Standard VHDL Language Reference Manual, IEEE Std 1076-1993, 1994.

[Navabi93] Navabi, Z., VHDL: Analysis and Modeling of Digital Systems, McGraw-Hill, 1993.

[Menchini94] Menchini, P., *"Class Notes for Top Down Design with VHDL"*, 1994.

[MG90] An Introduction to Modeling in VHDL, Mentor Graphics Corporation, 1990.

[MG93] Introduction to VHDL, Mentor Graphics Corporation, 1993.

[Perry94] Perry, D. L., VHDL, McGraw-Hill, 1994.

[Smith88] Smith, David, "What is Logic Synthesis", *VLSI Design and Test*, October, 1988.

[USE/DA94] USE/DA Standards Survey, 1994.

[VI93] VHDL International Survey, 1993.

[Walker85] Walker, Robert A. and Thomas, Donald E., "A Model of Design Representation and Syntheses", 22nd Design Automation Conference, pp. 453-459, IEEE, 1985

[Williams94] Williams, R. D., *"Class Notes for EE 435: Computer Organization and Design"*, University of Virginia, 1994.