Politecnico di Milano

Reti Logiche A

Corso di Reti Logiche A

Introduzione al VHDL
- Una prima presentazione -

VHSIC-HDL

Very High Speed Integrated Circuit - Hardware Description Language

Marco D. Santambrogio:
marco.santambrogio@polimi.it

VHDL

- **V**HSIC **H**ardware **D**escription **L**anguage
  - **V**ery **H**igh **S**peed **I**ntegrated **C**ircuit
  - Linguaggio per la descrizione dei C.I.

- Esistono tre modi per descrivere un C.I.
  - Dataflow – Descrivere un circuito usando i gate
  - Structural – Descrivere un circuito usando delle distinte descrizioni strutturali
  - Behavioral – Attraverso del *codice* (ifs, loops, etc.)

Introduzione

Il Linguaggio VHDL viene utilizzato per: Documentare, Simulare, Sintetizzare circuiti e sistemi logici.
Esso è costituito da più parti alcune delle quali fanno parte integrale del linguaggio stesso, mentre altre vengono integrate da opportuni "packages" realizzati all'interno di "libraries"

User Library	User Package	blocchi e funzioni precedentemente sviluppati dall'utente
Vendor Library	Vendor Package	blocchi e funzioni sviluppati dal fornitore
Library IEEE	Package STD_LOGIC_1164 Package STD_LOGIC_ARITH	Definizione variabili logiche "fisiche" Definizione di funzioni aritmetico-logiche
Library STD	Package TEXTIO Package STANDARD	Operazioni su stringhe Unità di tempo, caratteri ASCII, ecc.
Library WORK	Local User Package	Variabili, blocchi e funzioni locali

Struttura generale di un programma

```
use libreria

entity circuito is
...
end circutio;

architecture archi of circuito is
-- istruzione concorrente 1
-- istruzione concorrente 2
begin
      pa: process
      begin
            -- istruzione sequenziale pa_1
            -- istruzione sequenziale pa_2
            -- ...
      end;

      pb: process
      begin
            -- istruzione sequenziale pa_2
            -- ...
      end;
end;
```

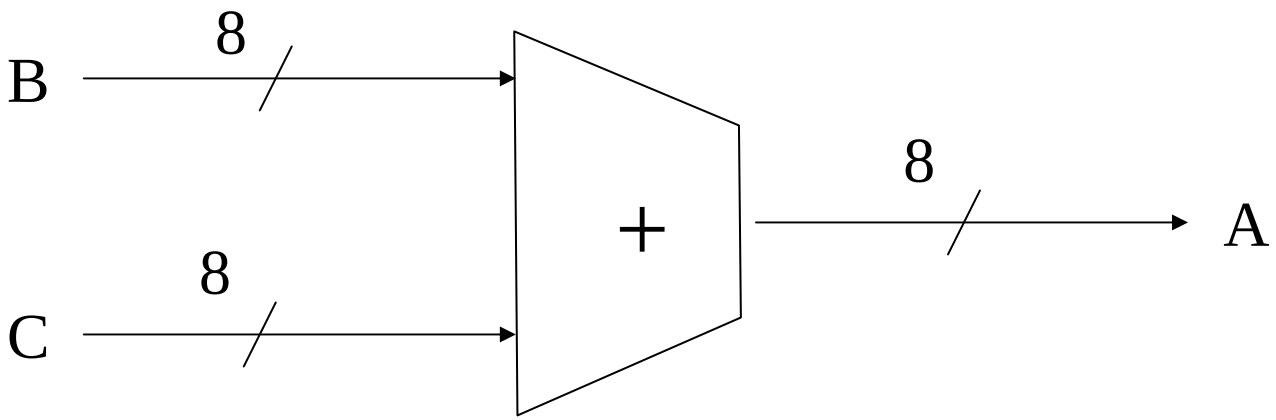
Area concorrente

Area sequenziale

Area sequenziale

Un primo esempio

```
A <= B + C after 5.0 ns
```



Aspetti introdotti:

- Descrizione / documentazione
- Simulazione
- Sintesi Logica

Descrizione / Documentazione

- Una delle funzioni del VHDL è quella di descrivere / documentare il funzionamento di un sistema in modo chiare ed inequivocabile
- Non è detto che questo sistema debba essere realizzato
- Alle volte è IMPOSSIBILE la realizzazione fisica del circuito
- Potrebbe essere la descrizione di un sistema già in funzione
- Potrebbe essere un modo per descrivere gli stimoli da impiegare per testare un circuito

Simulazione

- Un sistema descritto in VHDL viene solitamente SIMULATO per analizzarne in comportamento (simulazione comportamentale)
- Bisogna fornire degli stimoli (INPUT)
- Ed avere un sistema capace di osservare l'evoluzione del modello durante la simulazione, registrarne le variazioni per un'eventuale ispezione di funzionamento
- Il simulatore deve aver la possibilità di rappresentare una variabile come "unknown".

Sintesi Logica

- Passaggio tra descrizione comportamentale e descrizione a porte logiche
- La sintesi avviene tramite appositi programmi che si appoggiano a librerie dove sono descritte le porte logiche da impiegare (fornite dal venditore)
- La sintesi è un processo delicato che deve essere opportunamente “guidato ed ottimizzato”
- Solo un ristretto sottoinsieme del VHDL si presta ad essere Sintetizzato automaticamente.
  Non tutto ciò che è scritto in VHDL è sintetizzabile
- La restante parte è da impiegarsi per la descrizione e per la simulazione

Progetto in VHDL

- Può essere composto da più unità compilate e salvate in opportune librerie
- Queste unità sono:
  - Entity
  - Architecture
  - Package
- Entity ed Architecture descrivono i componenti come interfaccia e come struttura interna
- Package: contiene funzioni e/o grandezze di uso comune

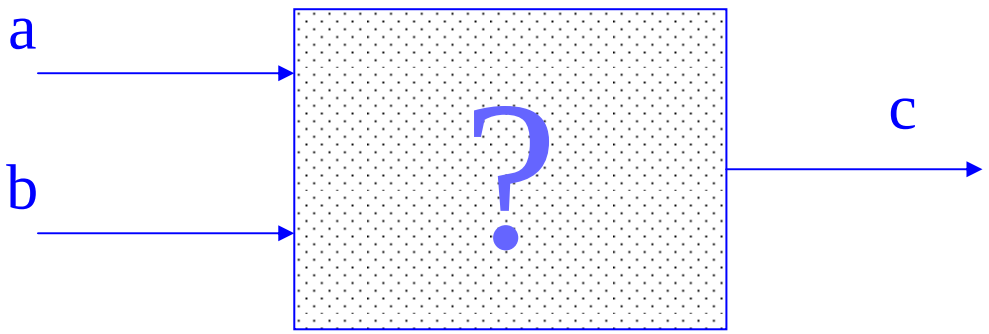
Esempio

```
package my_defs is                                  (Package)
       constant unit_delay: time := 1 ns  -- ritardo
end my_defs

entity COMPARE is                                   (Entity)
       port (a, b : in bit;
             c : out bit);
end COMPARE

architecture DATAFLOW of COMPARE is      (Architecture)
       begin
       c <= not (a xor b) after work.my_defs.unit_delay;
end DATAFLOW
```

Entity

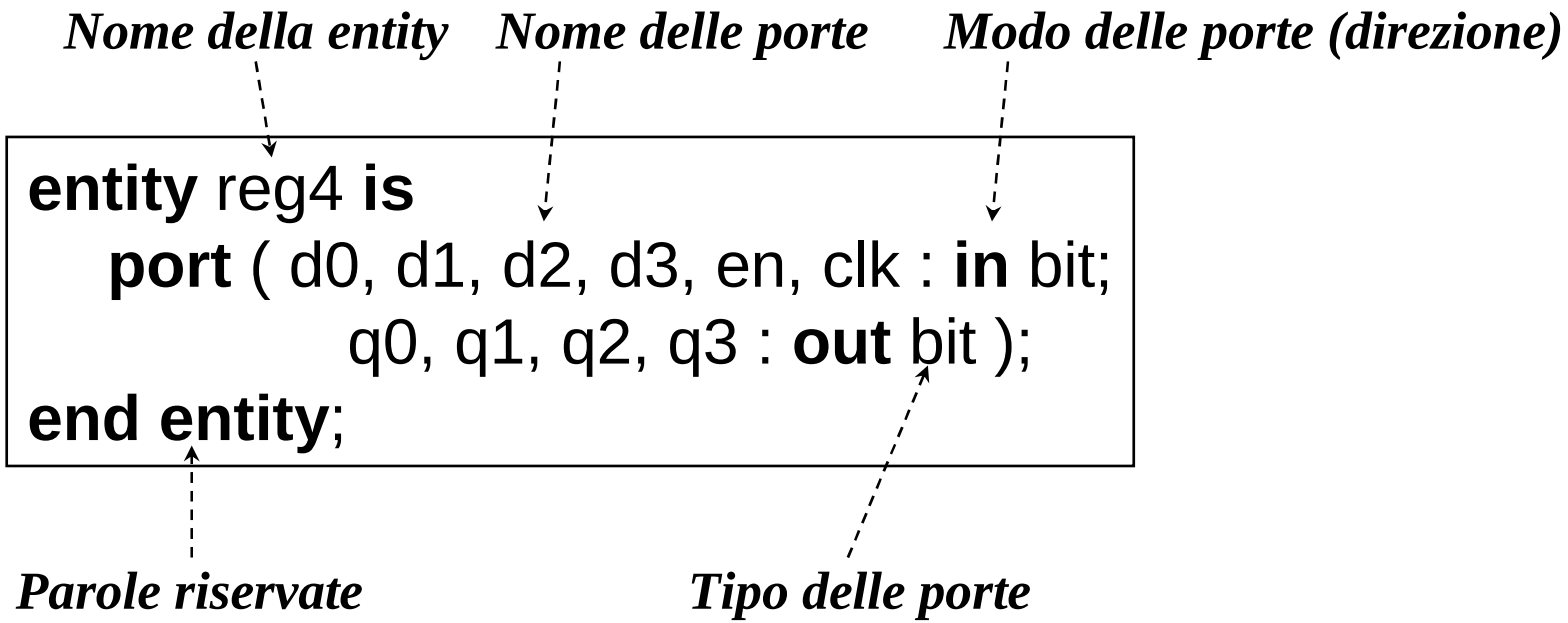
- Descrive un componente solo come Interfaccia da e verso l'esterno, ne fornisce una visione "ai morsetti"
- Non fornisce alcun dettaglio sul funzionamento o sull'architettura

```
entity COMPARE is
       port (a, b : in bit;
             c : out bit);
end COMPARE
```



Descrizione delle entity

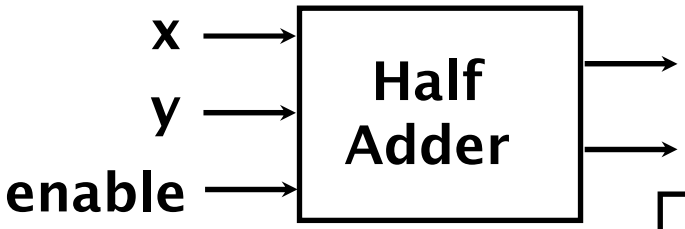
- *Entity*
  - Descrive gli ingressi e le uscite di un modulo



VHDL – Half Adder

- Problema: descrivere un singolo bit half adder con i segnali di carry ed enable
- Specifica
  - Ingressi e uscite sono tutti definiti da un solo bit
  - Quando enable è alto, result è x + y
  - Quando enable è alto, carry è uguale x and y
  - Le uscite sono uguali a 0 quando enable è basso

Entity Half Adder




```
ENTITY half_adder IS

    GENERIC(prop_delay : TIME := 10 ns);

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

END half_adder;
```

Architecture

- Descrive il funzionamento dell'"Entity" tramite:
  - descrizione astratta (comportamentale)
  - equazioni logico/ aritmetiche (dataflow)
  - Interconnessione tra moduli (strutturale)

- Lo stesso componente può essere descritto a più livelli di astrazione

Architecture (comportamentale)

- Descrizione comportamentale ad alto livello di astrazione (risulta molto simile ad un algoritmo espresso secondo il classici linguaggi sequenziali (C, Fortran, Pascal, ecc..)
- Utile per simulare parti di circuito senza dover scendere troppo nel dettaglio del funzionamento.
- Utilizzo di PROCESS (con lista dei segnali di attivazione)
- Più operazioni agenti in parallelo risiedono in diversi “Process”
- Diversi processi comunicano tra loro mediante “SEGNALI” ma al loro interno lavorano mediante “VARIABILI”
- Al loro interno i “Processes” sono sequenziali
- ATTENZIONE: non tutto ciò che viene descritto al livello comportamentale risulta sintetizzabile

Processi

- Un **process** è un'istruzione concorrente che contiene un'area sequenziale.
- Un processo viene eseguito parallelamente alle altre istruzioni concorrenti.
  - L'esecuzione del suo body può essere condizionata da una **sensitivity list**, una lista di segnali.
  - La sensitivity list non si usa quando il body contiene un'istruzione wait.
    - In questo caso l'esecuzione viene avviata e si sospende nel modo indicato da *wait*.
- Note:
  - In un processo le variabili locali conservano in proprio valore nel tempo tra un'esecuzione e l'altra.

Processi - Esempio

```
p1: process (B, C)
begin
    A <= B and C;
    C <= '0';
end;
```

```
p1_2: process
begin
    A <= B and C;
    C <= '0';
    wait on B, C;
end;
```

Architecture (comportamentale)
Half Adder

```
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 half_adder_a;
```

Architecture (dataflow) - Half Adder

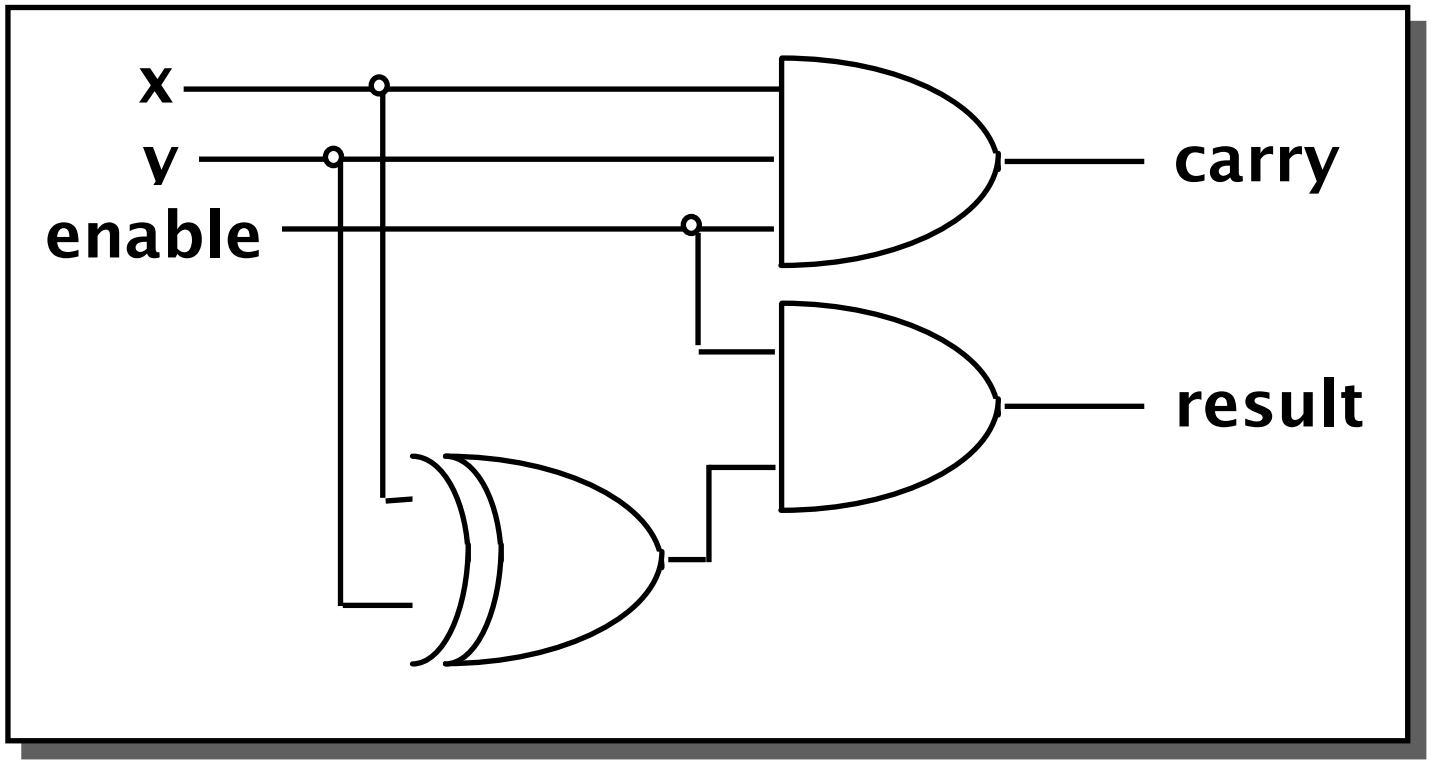
```
ARCHITECTURE half_adder_b OF half_adder IS
    BEGIN
      carry <= enable AND (x AND y);
      result <= enable AND (x XOR y);
    END half_adder_b;
```

Descrizione secondo equazioni logiche

Architecture (strutturale)

- Descrizione Strutturale ovvero mediante blocchi tra loro interconnessi
- I vari “components” devono essere già presenti in una libreria di riferimento
- La dichiarazione dei “components” e’ spesso raccolta in un “package”
- La “port map” indica il collegamento fisico
- Vengono solitamente impiegati segnali interni

Architecture (strutturale) - Half Adder



Architecture (strutturale) - Half Adder

```
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;

-- description is continued on next slide
```

Architecture (strutturale) - Half Adder

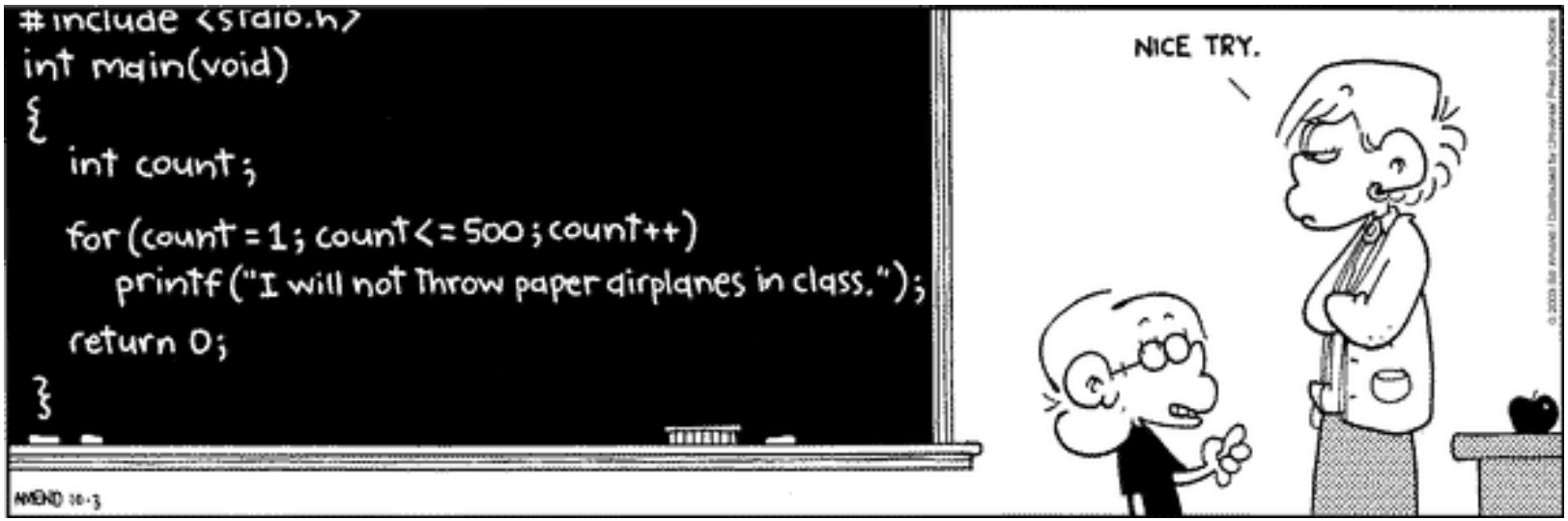
```
-- continuing half_adder_c description

    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 half_adder_c;
```

Architecture

- Tutte queste tecniche costitutive per l'Architecture possono fondersi tra loro

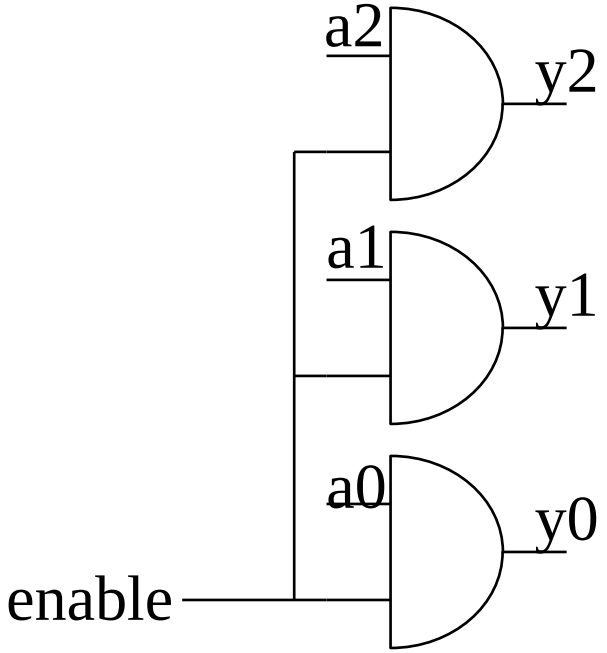
- Una buona descrizione architetturale è il primo passo per una buona Sintesi
  - Non tutto ciò che è scritto in VHDL può essere sintetizzato!!

Alcuni esempi



Controllo dei processi: IF-THEN-ELSE

```
SIGNAL a : STD_LOGIC_VECTOR(2 downto 0);
SIGNAL y : STD_LOGIC_VECTOR(2 downto 0);
SIGNAL enable : STD_LOGIC;

PROCESS( enable)
BEGIN
    IF (enable = '1') THEN
        y <= a;
    ELSE
        y <= "000";
    END IF;
END PROCESS;
```

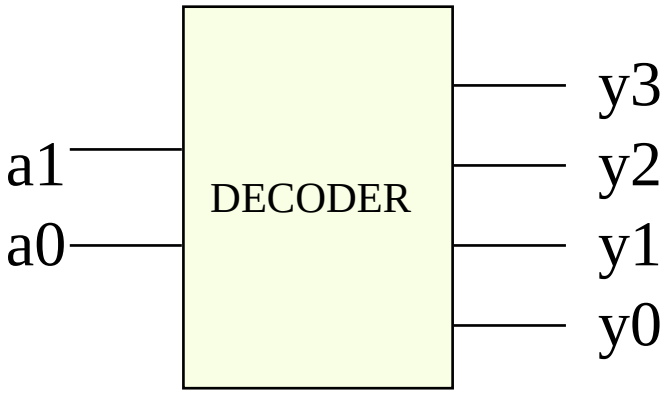


Controllo dei processi: CASE

```
SIGNAL a : STD_LOGIC;
SIGNAL y : STD_LOGIC;
SIGNAL enable : STD_LOGIC_VECTOR(1 downto
    0);

PROCESS ( enable )
BEGIN
    CASE enable IS
        WHEN “00” =>
            y <= a;
        WHEN “01” =>
            y <= ‘0’;
        WHEN “10” =>
            y <= ‘0’;
        WHEN OTHERS =>
            y <= ‘1’;
    END CASE;
END PROCESS;
```

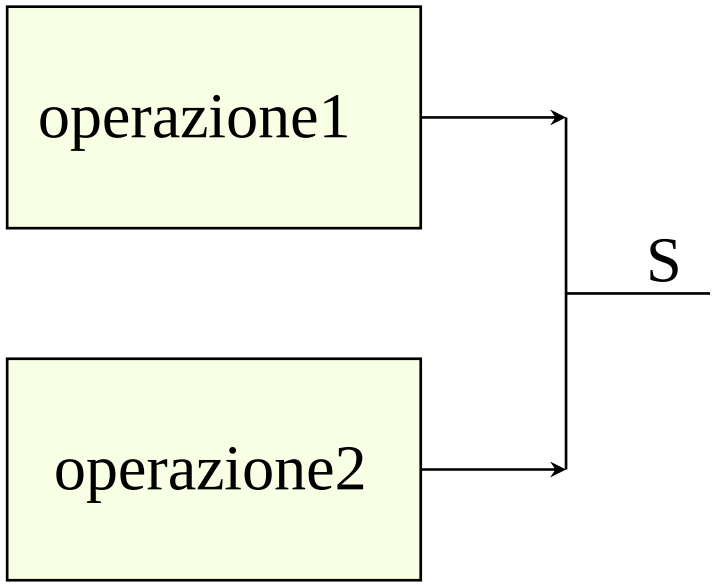
Controllo dei processi: CASE - DECODER

```
SIGNAL y : STD_LOGIC_VECTOR(3 downto 0);
SIGNAL a : STD_LOGIC_VECTOR(1 downto 0);

PROCESS (a)
BEGIN
    CASE a IS
        WHEN “00” =>
            y <= “0001”;
        WHEN “01” =>
            y <= “0010”;
        WHEN “10” =>
            y <= “0100”;
        WHEN “11” =>
            y <= “1000”;
        WHEN OTHERS => ;
    END CASE;
END PROCESS;
```

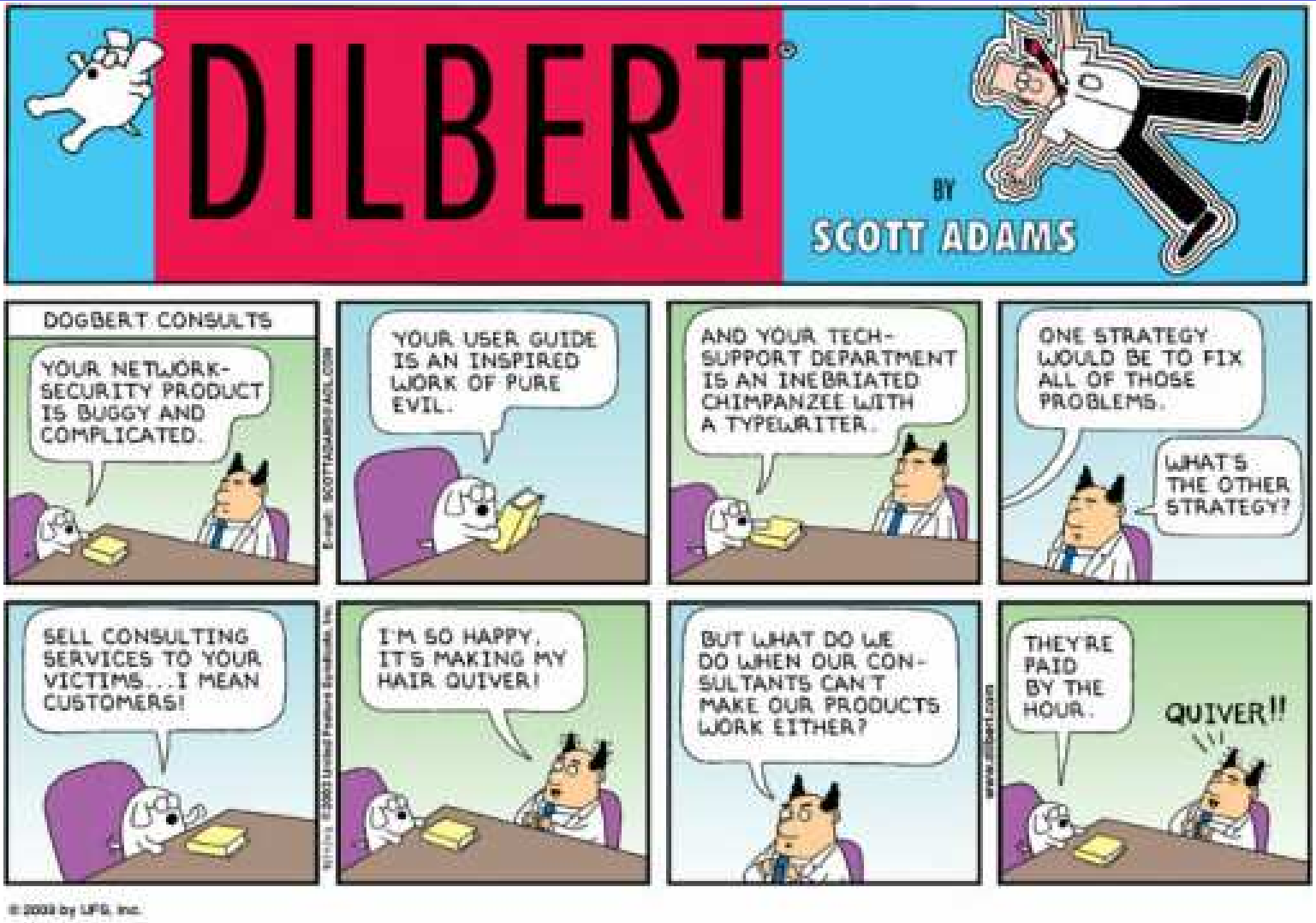


Double DRIVEN

```
ARCHITECTURE arch  OF circ IS
SIGNAL s : BIT;
BEGIN
      PROCESS
      BEGIN
            s <= operazione1;
      END PROCESS;

      PROCESS
      BEGIN
            s <= operazione2;
      END PROCESS;
END arch;
```



Domande?


DILBERT
BY
SCOTT ADAMS
DOGBERT CONSULTS
YOUR NETWORK-SECURITY PRODUCT IS BUGGY AND COMPLICATED.
YOUR USER GUIDE IS AN INSPIRED WORK OF PURE EVIL.
AND YOUR TECH-SUPPORT DEPARTMENT IS AN INEBRIATED CHIMPANZEE WITH A TYPEWRITER.
ONE STRATEGY WOULD BE TO FIX ALL OF THOSE PROBLEMS.
WHAT'S THE OTHER STRATEGY?
SELL CONSULTING SERVICES TO YOUR VICTIMS...I MEAN CUSTOMERS!
I'M SO HAPPY, IT'S MAKING MY HAIR QUIVER!
BUT WHAT DO WE DO WHEN OUR CONSULTANTS CAN'T MAKE OUR PRODUCTS WORK EITHER?
THEY'RE PAID BY THE HOUR.
QUIVER!!
© 2003 by UFS, Inc.

- Un po’ di sintassi -

Appunti non esaustivi.
Lo studio non può considerarsi completo con solo questa presentazione

Introduzione

- Il VHDL e' costituito da vari formati (types)ed operatori (operators) per consentire simulazione e sintesi a vari livelli
- Nel package STANDARD si trovano descritti quegli oggetti destinati alla descrizione COMPORTAMENTALE (non sempre sintetizzabile)
- Nel package IEEE1164 vi si trovano gli oggetti destinati alla sintesi ed alla simulazione logica
- Il VHDL e' un linguaggio fortemente basato sulla sintassi

$$1+1 \neq 2.0$$

Sintassi

- Le varie espressioni sintattiche scritte in VHDL si possono ricondurre ai seguenti oggetti:
  - Scalari e Vettori
  - Nomi
  - Oggetti:
    - Costanti
    - Segnali
    - Variabili
  - Espressioni

Sintassi - Scalari e Vettori

Scalari	**Vettori**
character	string
bit	bit_vector
std_logic	std_logic_vector
boolean	
real	
integer	
time	

Bit

- Il BIT assume solo valori ‘0’ o ‘1’ e va dichiarato tra virgolette singole
  - Es: ‘0’ ‘1’
- In caso di equivoco si usi la dichiarazione esplicita
  - Es: bit’(‘0’) bit’(‘1’)

Bit_vector

- Il Bit_vector e' un array di Bit che assumono solo valori '0' o '1' e va dichiarato tra virgolette doppie, e' comunque consentito adottare una notazione ottale o esagesimale. Il carattere '_' puo' essere adottato per comodita', ma non viene interpretato
  - Es: "0001_1001" x"00FF"
- In caso di equivoco si usi la dichiarazione esplicita
  - Es: bit_vector'("0110_0101_0011")

STD_logic

- E’ il “type” piu’ usato per la sintesi logica
- Assume i valori:

'U'	uninitialized		
'X'	unknown	'W'	weak unknown
'0'	0 logic	'L'	weak 0
'1'	1 logic	'H'	weak 1
'Z'	high impedence	'-'	don't care

STD_logic

- Viene dichiarato racchiuso tra virgolette singole
  - Es: ‘U’ ‘X’ ‘1’ ‘0’
- In caso di equivoco si usi la dichiarazione esplicita
  - Es: std_logic’(‘1’)

STD_logic_vector

- Viene dichiarato racchiuso tra virgolette doppie
  - Es: “001XX” “UUUU”
- In caso si voglia esprimere un particolare valore espresso secondo una notazione di tipo “unsigned” o “signed” (complemento a 2) si deve impiegare il pakage STD_LOGIC_ARITH
  - Es: signed’(“111001”) (ossia -7)
    unsigned(“111001”) (ossia 57)

```
Library IEEE;
Use IEEE.STD_LOGIC_1164.all;
Use IEEE.STD_LOGIC_ARITH.all;
```

Real

- Puo' essere utile per simulazioni ad alto livello
- **NON VIENE SINTETIZZATO**
- DEVE contenere il punto decimale ed eventualmente il segno
  - Es: 1.0 +2.23 - 4.56 -1.0E+38
- Per impiegare un array di numeri reali deve essere opportunamente dichiarato

Integer

- Puo' essere utile per simulazioni ad alto livello
- **NON SEMPRE VIENE SINTETIZZATO**
- NON DEVE contenere il punto decimale ma puo' eventualmente contenere il segno
  - Es: 10 +223 - 456
- Un intero puo' eventualmente essere espresso in una latra base
  - Es: 16#00F0F#
- Nel pakage STANDARD sono descritti due subset degli Integer: positive e natural

Time

- E’ la sola grandezza fisica predefinita in VHDL.
- E’ definita nel Pakage STANDARD
- E’ importante separare il valore dall’unita’ di grandezza
  - Es: 10 ns 123 us 6.3 sec
- Unita’ di grandezza consentite:

  fs ps ns us ms sec min hr

type, subtype

- In VHDL si possono “inventare” delle variabili “su misura

```
TYPE mese IS (gennaio, febbraio, giugno);
TYPE bit IS (‘0’, ‘1’);
```

- E dei sottoinsiemi di queste”

```
SUBTYPE mesefreddo IS
mese range gennaio to febbraio;
```

Sintassi - Nomi

- Ogni oggetto (entity, architerctures, segnali, …) ha un nome simbolico
- Il VHDL e' un linguaggio "Case insentistive" (ossia abcd e' analogo a AbCd)
- Vi sono "nomi riservati" quali:
  in, out, signal, port, library, map, entity, ….
- I nomi "relativi" vengono indicati con un "." nella sintassi
  - Es: libray_name.pakage_name.item.neme
    WORK.my_defs.unit_delay

Sintassi - Dichiarazioni di oggetti

- In VHDL vi sono grandezze che mantengono il loro valore immutabile ed altre che possono cambiare valore
  - constants: gandezze fisse
  - signals: rappresentano collegamenti fisici (sono grandezze concorrenti)
  - variables: rappresentano variabili all'interno di un processo (sono grandezze sequenziali)
- Inoltre si possono definire dei puntatori a files (ovviamente per blocchi puramente comportamentali)
- Ogni grandezza impiegata deve essere definita a priori

```
variable x: integer;
signal aBc: bit;
constant Vdd: real := 12.3;
```

Sintassi - Signals

- Sono l'astrazione dei "collegamenti fisici"
- Fanno comunicare tra loro varie entity
- Un segnale può essere inizializzato (ATTENZIONE in fase di SINTESI l'inizializzazione potrebbe essere disattesa!)
- In una entity un segnale viene dichiarato tramite la `port`

```
signal count: integer range 1 to 10;
signal GROUND: bit :=0 ;
signal SYS_BUS : std_logic_vector (7 downto 0);
port (A, B : in std_logic);
```

Sintassi - Variables

- Una variabile viene impiegata nei `process`
- l'assegnamento del valore ad una variabile avviene istantaneamente in simulazione (all'opposto di un segnale per cui lassegnamento avviene in base al "tempo di simulazione")
- deve essere dichiarata prima di essere usata

```
variable INDEX : integer range 1 to 50;
variable CYCLE : time range 10 ns to 50 ns := 10ns;
varaible MEMORY : bit_vector (0 to 7);
variable x,y,z : integer;
```

Sintassi - Operandi

- Logici:
  `and, or, nand, nor, xor`
- Relazionali
  `=, /=, <, <=, >, >=`
- Concatenazione e aritnetici
  `&, +, -, *, /, mod, rem, **, abs`
- Logici:
  `not`

- NOTA: il precedente elenco e' ordinato in base alla priotita'

Pakages STD_logic_arith

- Il pakage STANDARD non consente comparazioni o operazioni aritmetiche tra “`bit_vector`”
- Alcuni venditori provvedono un pakage per definire le operazioni tra std_logic_vector
- Servono per definire se le grandezze impiegate sono di tipo “signed” o “unsigned”
  - ovvero ad esempio come interpretare la grandezza “1011” ossia 11 oppure -5 ?)

```
use IEEE.std_logic_signed.all;
use IEEE.std_logic_unsigned.all;
```

WebPACK

- ISE *WebPACK* è un tool gratuito e scaricabile dal web che supporta la scrittura, la sintesi e la verifica di specifiche in HDL sia per CPLD che FPGA.
- E' possibile scaricarlo gratuitamente da:
  - www.xilinx.com

Bibliografia

- The VHDL Cookbook, P. J.Ashenden
- The Hamburg VHDL archive:
  - http://tech-www.informatik.uni-hamburg.de/vhdl/
- Il Linguaggio VHDL. Introduzione al linguaggio VHDL per la descrizione di sistemi digitali, Xilinx Inc.
- WEB-LINKS:
  - http://www.ashenden.com.au/designers-guide/DG.html
  - http://www.ashenden.com.au/designers-guide/DG-intro-lectures.html
  - http://www.ashenden.com.au/students-guide/SG.html