Politecnico di Milano

Reti Logiche

Introduzione al VHDL

Antonio Miele
antonio.miele@polimi.it
Marco Lattuada
marco.lattuada@polimi.it

Perché c'è bisogno di un hdl?

- Perché c'è bisogno di un linguaggio di descrizione dello hardware (hardware description language, hdl)?

- I linguaggi di programmazione non supportano pienamente la specifica di diverse caratteristiche fondamentali dello hardware:
  - Interfacce input/output
  - Tipi di dati e specifica dell'ampiezza dei dati
  - Temporizzazione
  - Concorrenza
  - Sincronizzazione

VHDL

- Il VHDL è un linguaggio di descrizione dello hardware
- Viene utilizzato per: documentare, simulare, sintetizzare circuiti e sistemi logici
- VHDL sta per VHSIC-HDL cioè Very High Speed Integrated Circuit – Hardware Description Language
- Il VHDL è stato definito negli anni ‘80 dal dipartimento della difesa USA
- L’ultima versione pubblica risale al 1993 (IEEE std 1076-1993)

Descrizione / Documentazione

- Una delle funzioni del VHDL è quella di descrivere/documentare il funzionamento di un sistema in modo chiaro 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 valori "unknown" o "non-initialized" o alta impedenza

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

Scrittura del codice sorgente

- Il codice sorgente di un modello VHDL è un file di semplice testo
- In genere si usa un nome uguale al nome dell'entità; l'estensione deve essere *.vhd
- Il vhdl è case insensitive
- "--" indica l'inizio di una riga di commento al codice

Tipi

- Il VHDL e' costituito da vari tipi (types) 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 tipizzato
  - Non è possibile fare il casting in modo implicito
  - Ci sono delle funzioni che permettono di "tradurre" da un tipo ad un altro

Tipi

Scalari	**Vettori**
Character	String
Bit	Bit_vector
Std_logic	Std_logic_vector
Boolean	
Real	
Integer	
Time	
File	

- È possibile definire nuovi tipi e sottotipi

Bit

- Il bit assume solo valori ‘0’ o ‘1’ e va dichiarato tra virgolette singole
  - Es: `'0'` `'1'`
- Si può utilizzare la dichiarazione esplicita
  - Es: `bit'('0')` `bit'('1')`

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')`

- E' il tipo più usato per la sintesi logica

- Per utilizzarlo va inclusa la libreria IEEE e specificato l'utilizzo del package std_logic_1164:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
```

std_logic

- Assume i valori:
  - U: unitialized
  - X: unknown
  - 0
  - 1
  - Z: alta impedenza
  - W: weak unknown
  - L: weak 0
  - H: weak 1
  - - : don’t care, indifferenza

Time

- È la sola grandezza fisica predefinita in VHDL
- È importante separare il valore dall'unità di grandezza
  - Es: `10 ns, 123 us, 6.3 sec`
- Unità di grandezza consentite:
  `fs ps ns us ms sec min hr`

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 package STD_LOGIC_ARITH
  - Es: **signed'("111001")** (ossia -7)
    unsigned'("111001") (ossia 57)
- bisogna includere i package:

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

Type e subtype

- In VHDL si possono “inventare” nuovi tipi

```
TYPE mese IS (gennaio, febbraio, giugno);

TYPE bit IS ('0', '1');

TYPE mystring IS ARRAY (0 to 4) OF bit;

TYPE norange IS ARRAY (natural range <>) OF bit;

TYPE rec IS RECORD
   campo1: bit; ...
END RECORD;
```

- E dei sottoinsiemi

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

Identificativi

- Ogni elemento della specifica ha un nome simbolico
- I nomi seguono le solite regole sintattiche dei linguaggi di programmazione
  - Il primo carattere deve essere una lettera
  - I seguenti possono essere alfanumerici
- Il VHDL e' un linguaggio "case insensitive"
  - (ossia abcd equivale a AbCd)
- Vi sono "nomi riservati" quali:

  in, out, signal, port, library, map, entity, ...

Identificativi

- Ogni identificativo ha un dominio di validità:
  - Libreria
  - Package
  - Architettura di un'entità
  - Processi
- I nomi "relativi" vengono indicati con un "." nella sintassi
  - Es: library_name.pakage_name.item.name
WORK.my_defs.unit_delay

Unità di progetto

- Può essere composto da più unità compilate e salvate in opportune librerie

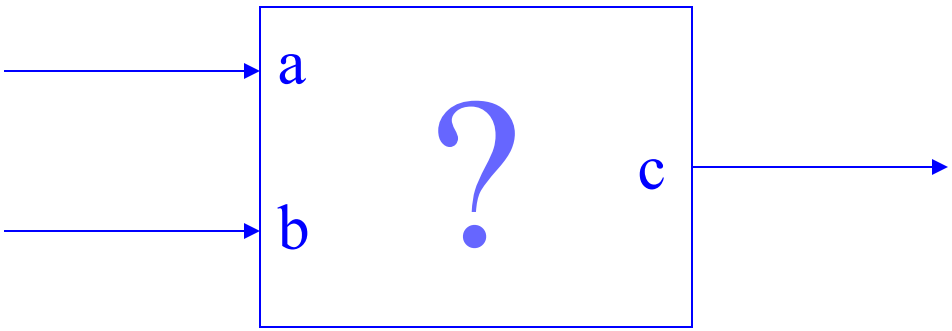
- Queste unità sono:
  - Dichiarazione di entità (entity)
  - Architettura di una entità (architecture)
  - Dichiarazione di Package (package)
  - Corpo del package (package body)
  - Configurazione (configuration)

Unità di progetto

- L’entità e l’architettura descrivono i componenti come interfaccia e come struttura interna
- Il package ed il suo corpo sono usate per contenere funzioni e/o costanti di uso comune
- Le configurazioni permettono di configurare i componenti specificati

Entity

- Descrive un componente solo come interfaccia da e verso l'esterno
- Fornisce una visione "ai morsetti"
- Non fornisce alcun dettaglio sul funzionamento o sull'architettura
- Può rappresentare
  - Una singola porta logica
  - Un componente
  - Un intero sistema



Entity

- Sintassi:

```
entity ENTITY_NAME is
        generic(
                GENERIC_NAME: TYPE := DEFAULT_VALUE
        );
        port( PORT_NAME: DIRECTION TYPE;
        ...
                PORT_NAME: DIRECTION TYPE;
        )
end [entity] [ENTITY_NAME];
```

- Specifica il nome dell'interfaccia
- Descrive la lista di porte
  - Per ogni porta è specificato il nome, il tipo e la direzione (IN, OUT, INOUT)

Entity

- La lista dei generics permette di parametrizzare la specifica
  - Il valore attuale del parametro è stabilito durante l'istanziazione del componente
- Esempio:

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

entity COMPARE2 is
      generic (WIDTH : integer := 4);
      port (a, b : in std_logic_vector(0 to WIDTH);
            c : out std_logic);
end COMPARE2;
```

Architecture

- L’architettura specifica il funzionamento di una entità

- Diverse architetture possono essere associate ad una stessa entità
  - Ogni architettura rappresenta una diversa realizzazione della funzionalità
  - La configurazione permette di stabilire quale implementazione usare in un progetto

Architecture

- Sintassi:

```
architecture ARCH_NAME of ENTITY_NAME is
        istruzioni dichiarative
begin
        istruzioni funzionali
end [architecture] [ARCH_NAME]
```

- Le istruzioni dichiarative permettono di dichiarare costanti, segnali e/o componenti che verranno utilizzate nel modello
- Le istruzioni funzionali specificano il funzionamento mediante istruzioni concorrenti

Costanti

- Aiutano a migliorare la leggibilità del codice
- Sintassi:

```
constant nome: tipo := espressione;
constant nome: tipo_array[(intervallo)] :=
  espressione;
```

- Esempio:

```
constant GROUND: bit :=0 ;
constant SYS_BUS : std_logic_vector (7 downto 0):=
    "00001111";
constant SYS_ADD : std_logic_vector (0 to 7):=
    "00001111";
```

Segnali

- Sono l'astrazione dei "collegamenti fisici"
- Connettono i vari elementi della specifica (l'entità, i componenti istanziati e i processi)
- Un segnale può essere inizializzato
  - ATTENZIONE: in fase di SINTESI l'inizializzazione potrebbe essere disattesa!
- Sintassi

```
signal nome: tipo := espressione;
signal nome: tipo_array[(intervallo)] :=
  espressione
```

- Esempio:

```
signal SYS_BUS : std_logic_vector (7 downto 0);
```

Indirizzamento negli array

- Per riferirsi ad una posizione di un array si usano le parentesi tonde
  - Esempio: `my_array(2)`
- Per riferirsi ad un intervallo si usano le parole chiave to e downto
  - Esempio: `my_array(2 to 4), my_array(5 downto 2)`

Attributi

- Esistono degli attributi predefiniti che sono associati ai segnali (e alle variabili)
- Si accede agli attributi tramite l'operatore ' :
  - Esempio: **`sig_array'lenght`**
- Per i segnali è definito l'attributo **`event`**
  - assume valore "vero" se in un determinato istante si è verificato un evento sul segnale

Tipi di descrizione

- L'architettura può essere descritta tramite tre diversi approcci:
  - Descrizione DATAFLOW
    - Basata su formule logiche
  - Descrizione STRUCTURAL
    - Composta da componenti istanziati e interconnessi tra di loro
  - Descrizione BEHAVIORAL (comportamentale)
    - Basata su descrizioni algoritmiche
- La descrizione può essere anche di tipo misto

Descrizione dataflow

- L'architettura è descritta in termini di una serie di espressioni
- Le espressioni sono eseguite in modo concorrente
  - La posizione relativa delle istruzioni che rappresentano le varie espressioni è ININFLUENTE
- Le espressioni possibili sono:
  - Assegnamento di segnali (o porte)
  - Assegnamento condizionale di segnali
  - Assegnamento selettivo di segnali

Descrizione dataflow

- Esempio:

```
architecture DATAFLOW of COMPARE is
begin
        c <= not (a xor b) after 1 ns;
end DATAFLOW
```

Operatori

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

- NOTA: il precedente elenco è ordinato in base alla priorità

Assegnamento a segnale

- Sintassi:

```
segnale <= valore1 [after ritardo];
```

- Esempi

```
sum <= a xor b after 5 ns;
carry <= a and b;
data_out(0) <= data_in(7);
data_out(7 downto 1) <= data_in(0 to 6);
```

Gli indici degli intervalli possono essere diversi

Assegnamento a segnale

- Assegnamento posizionale

```
vettore_di_bit <= (bit1, bit2, bit3, bit4);
```

- Assegnamento nominale

```
vettore_di_bit <= (2=>bit1, 1=>bit2,0=>bit3,
  3=>bit4);
```

- Altri assegnamenti

```
vettore_di_bit <= (0 to 1 => '0', others
  =>'1');
```

Esempio

2-TO-4 LINE DECODER

Assegnamento condizionale

- Sintassi

```
segnale <= valore1 when condizione1 else
           valore2 when condizione2 else
           ...
           valoren;
```

- Esempio:

```
Z <= d0 when sel0='0' and sel1='0' else
     d1 when sel0='1' and sel1='0' else
     d3;
```

Esempio

MULTIPLEXER (ELSE VERSION)

Assegnamento selettivo

- Sintassi

```
with espressione select
        segnale <= valore1 when caso1,
                   valore2 when caso2,
        ...
        [          valoren when others];
```

- Non si possono sovrapporre i casi
- Esempio:

```
with alu_func select
   res <= a+b when "000"|"001",
          a-b when "010"|"011",
          "0000" when others;
```

Esempio

MULTIPLEXER (SELECT VERSION)

Esempio

```
...
begin

with sel select
  q<= i0 after 10 ns when 0;
  q<= i1 after 10 ns when 1;
  q<= i2 after 10 ns when 2;
  q<= i3 after 10 ns when 3;
  q<='U' after 10 ns when others;

sel <= 0 when a='0' and b='0' else
       1 when a='0' and b='1' else
       2 when a='1' and b='0' else
       3 when a='1' and b='1' else
       4;

end mux;
```

I due assegnamenti sono CONCORRENTI

Descrizione strutturale

- La descrizione strutturale è composta da istanze di componenti interconnessi tra loro
- La descrizione strutturale permette una specifica gerarchica del sistema
  - Supporta la progettazione incrementale sia top/down che bottom/up
- I vari componenti devono essere già presenti in una libreria di riferimento
- La dichiarazione dei componenti può essere raccolta in un “package”
- Se esistono più architetture per lo stesso componente si può usare una configurazione per specificare quale implementazione considerare

Descrizione strutturale

- Nella parte dichiarativa dell'architettura vanno specificati i componenti utilizzati
  - Si sostituisce la parola entity con la parola component

```
component COMPONENT_NAME is
        generic(
                GENERIC_NAME: TYPE := DEFAULT_VALUE
        );
        port( PORT_NAME: DIRECTION TYPE;
        ...
                PORT_NAME: DIRECTION TYPE;
        )
end component [COMPONENT_NAME];
```

Descrizione strutturale

- I componenti vanno istanziati dopo l'istruzione begin

```
nome_istanza : nome_componente
[generic map (assegnamento costanti, ...)]
port map (assegnamento segnali, ...);
```

- La "port map" indica il collegamento fisico
  - Assegnamento posizionale
  - Assegnamento nominale
- Eventuali valori predefiniti (generic) possono essere sovrascritti

Descrizione strutturale

- Nel port map per ciascuna porta si può specificare
  - Un segnale
  - Una costante
  - `Open` (non connesso)
  - Una porta della entità
- Vengono solitamente impiegati segnali per connettere le varie istanze dei componenti

Descrizione strutturale

- Esempio:

```
architecture STRUCTURAL of COMPARE is
signal I: bit;

component XR2 port (x,y: in BIT; z: out BIT);
end component;
component INV port (x: in BIT; y: out BIT);
end component;

begin
      U0: XR2 port map (A,B,I);
      U1: INV port map (y=>C,x=>I);
end STRUCTURAL;
```

Esempio

ADDER

Descrizione comportamentale

- Descrizione di un'architettura con una sintassi algoritmica mediante processi
  - risulta molto simile ad un algoritmo espresso secondo il classici linguaggi sequenziali (C, Fortran, Pascal, ecc..)
- Le istruzioni sono eseguite sequenzialmente all'interno di un processo
- Ogni processo è visto dall'esterno come una sola istruzione concorrente

ADDER Comportamentale

Descrizione comportamentale

- Utile per simulare parti di circuito senza dover scendere troppo nel dettaglio del funzionamento
- Diversi processi comunicano tra loro mediante segnali ma al loro interno lavorano mediante variabili
- ATTENZIONE: non tutto ciò che viene descritto al livello comportamentale risulta sintetizzabile

Aree concorrenti e sequenziali

```
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

Processi

- Sintassi:

```
[etichetta:] process [(lista di sensibilità)]
parte dichiarativa (variabili, ...)
begin
istruzioni sequenziali
...
end process [etichetta];
```

- La lista di sensibilità indica i segnali le cui variazioni causano l'attivazione del processo
- Prima dell'istruzione begin vengono dichiarati le variabili usate nel processo
- Dopo l'istruzione begin viene specificato il funzionamento del processo

Costrutto condizionale if

- Sintassi:

```
if condizione2 then
   istruzioni sequenziali;
{elsif condizione2 then
   altre istruzioni sequenziali;}
[else
   ultime istruzioni sequenziali;]
end if;
```

- Esempio:

```
if a > 0 then
   b <= '0';
else
   b <= '1';
end if;
```

Processi

- Esempio:

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

Esempio

POSITIVE EDGE-TRIGGERED D FLIP-FLOP

Esempio

CONTATORE BINARIO A 4 BIT

Processi – esecuzione

- Il processo è eseguito una prima volta durante l'inizializzazione
- Il processo viene risvegliato alla variazione dei segnali della lista di sensibilità
- Il processo è eseguito interamente o interrotto nel caso si incontra un'istruzione wait
- I segnali sono aggiornati solo al termine dell'esecuzione con l'ultimo valore assegnato
- L'esecuzione al suo interno è strettamente sequenziale

Variabili

- Sintassi:

```
variable var_name : tipo [:= valore];
```

- La visibilità di una variabile è limitata al processo in cui è dichiarata
- L'assegnamento del valore ad una variabile avviene istantaneamente
- In un processo le variabili locali conservano in proprio valore nel tempo tra un'esecuzione e l'altra

```
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;
```

Segnali e variabili

	Segnali	**Variabili**
Dichiarazione	Parte dichiarativa di un'architettura	Parte dichiarativa di un processo
Valore predefinito	Valore minimo del dominio di appartenenza	
Assegnamento	<=	:=
Inizializzazione	:=	
Natura dell'assegnamento	Concorrente	Sequenziale
Utilizzo	In architetture e processi	Solo in processi
Effetto dell'assegnamento	Non immediato (in base ai "tempi" della simulazione)	Immediato

Wait

- Sospende il processo in attesa di un evento

- **`wait;`**
  - Sospende il processo a tempo indefinito
- **`wait for time;`**
  - Sospende il processo per il tempo specificato
- **`wait on lista_segnali;`**
  - Sospende il processo fino alla variazione di uno dei segnali elencati
- **`wait until condizione;`**
  - Sospende il processo fino a quando una variazione dei segnali rende la condizione vera

Wait

- Le istruzioni di wait possono essere utilizzate al posto della lista di sensibilità
- **`Wait on lista_segnali;`** posto come ultima istruzione di un processo equivale alla specifica della lista di sensibilità
- Esempio:

```
process
        begin
        if (A = B) then
              C <= '1' after 1 ns;
        else
              C <= '0' after 1 ns;
        end if;
        wait on a, b;
end process;
```

Costrutto condizionale case

- Sintassi:

```
case espressione is
    when scelta1 => istruzioni sequenziali;
    when scelta2 => istruzioni sequenziali;
   [when others => ultime istruzioni
                   sequenziali;]
end case;
```

- Le scelte non possono sovrapporsi
- È possibile definire intervalli
  - e.g.: 1 to 4, 4 downto 1.

Costrutto condizionale case

- Esempio:

```
case muxval is
   when 0 => q<=i0 after 10 ns;
   when 1 => q<=i1 after 10 ns;
end case;
```

Esempio

RICONOSCITORE DI SEQUENZE

Librerie

- Le unità di progetto possono essere raggruppate in librerie
- In un dato istante c'è una sola libreria di lavoro e diverse librerie di risorse
- La libreria di lavoro è referenziata con il nome **`work`**
- Esempio di inclusione di una libreria e degli elementi contenuti

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
```

- Le librerie **`work`** e **`std`** e il package **`std.standard.all`** sono inclusi implicitamente

Testbench

- I moduli di test, chiamati testbench, sono specificati direttamente in VHDL
- Sono connessi al componente da testare o lo o istanziano all'interno
- Specificano gli stimoli in ingresso e collezionano i risultati
  - Queste due funzionalità possono essere eseguite direttamente dal simulatore
- Si possono definire dei puntatori a files (in processi) per poter caricare gli stimoli o salvare i risultati
- È utile usare istruzioni di asserzione
- Il codice del testbench può anche essere generato tramite un editor di forme d'onda

Esempio

TESTBENCH

Bibliografia

- Tutorial sul linguaggio VHDL
  - VHDL tutorial – Peter Ashenden (scaricabile dal sito del corso)
  - VHDL cookbook (disponibile su Internet)
- Strumenti di sintesi
  - Webpack (sintesi su FPGA) - www.xilinx.com
- Strumenti di simulazione
  - Modelsim - http://www.model.com/