Introduzione al VHDL

Christian Pilato

pilato@elet.polimi.it

Sommario

- Introduzione
- Struttura di un modello
  - Interfaccia
  - Funzionalità
- Concetti base
  - Livelli di astrazione
  - Concorrenza
  - Sequenzialità
  - Gerarchia
  - Temporizzazioni

Introduzione

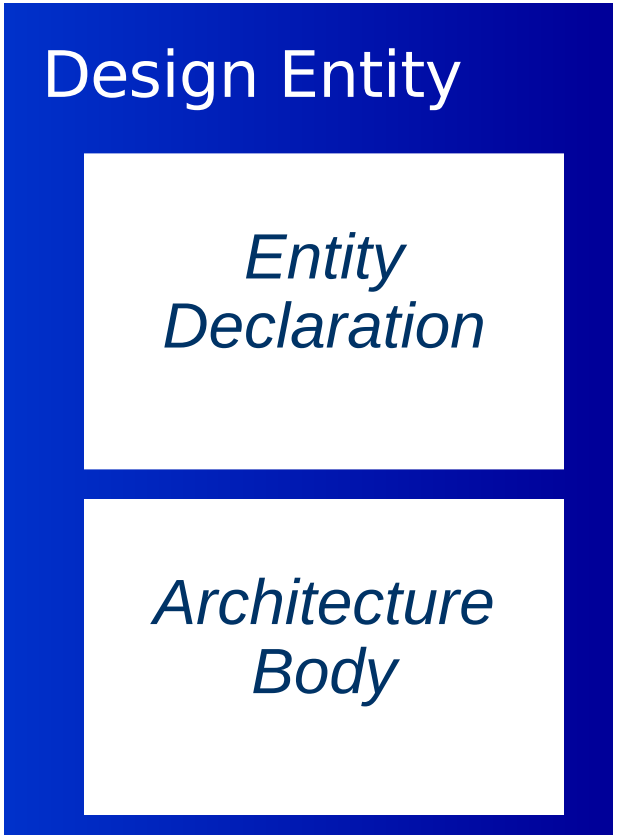
- La tecnologia microelettronica si è evoluta molto rapidamente negli ultimi decenni
  - La complessità dei circuiti è sempre maggiore
  - Le prestazioni devono essere sempre migliori
  - I costi devono essere continuamente ridotti
  - L'affidabilità non deve essere penalizzata
- La rapida evoluzione delle tecnologie favorisce l'obsolescenza dei circuiti
  - Il time-to-market deve essere ridotto al minimo
  - I tempi di sviluppo devono essere brevi
- L'uso di strumenti CAD aiuta a soddisfare questi vincoli

Obiettivi

- Assistere la progettazione di circuiti e sistemi digitali attraverso un linguaggio di descrizione dell'hardware
  - Al linguaggio viene poi affiancato uno strumento idoneo alla progettazione, alla simulazione e alla verifica del progetto
- HDL – Hardware Description Language
  - VHDL
  - VERILOG
- VHDL – VHSIC Hardware Description Language
  - VHSIC – Very High Speed Integrated Circuit
  - Definito negli anni '80 dal Dipartimento della Difesa USA
  - Ultima versione pubblica risale al 1993 (IEEE Std 1076-1993)
  - Usa *Multi Valued Logic*, tipo di dato a nove valori per la descrizione dei segnali (IEEE Std 1164)
  - NON è case-sensitive

La Modellizzazione

- Lo sviluppo di un modello si distingue in varie fasi
  - Analisi
  - Progettazione
  - Scrittura
  - Compilazione
  - Simulazione
  - Validazione
- La progettazione (o specifica concettuale) consiste nella descrizione di:
  - Interfaccia
  - Funzionalità



Design Entity

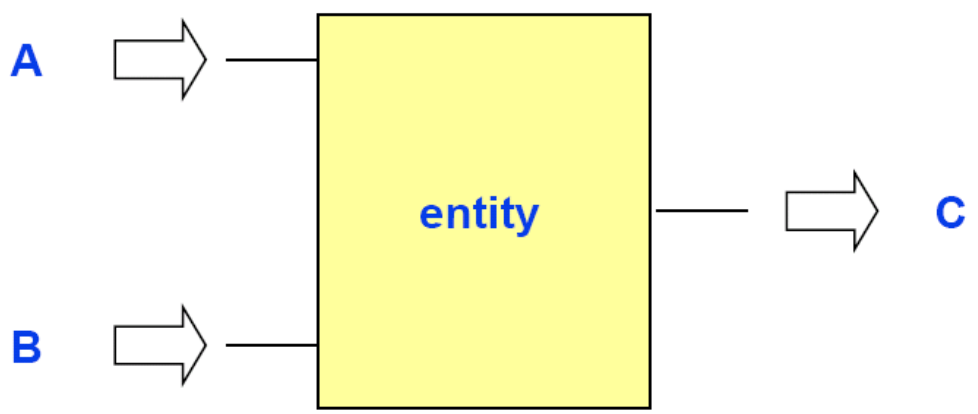
- E’ l’unità base della progettazione in VHDL
- Può rappresentare
  - Una singola porta logica elementare
  - Un circuito integrato
  - Una scheda PCB (Printed Circuit Board)
  - Un intero sistema
- Un modello VHDL può essere definito a diversi livelli di astrazione, partendo da un modello ad alto livello per poi raffinarlo

Livelli di Astrazione

- Diverse *Architecture Body* possono essere associate ad una stessa *Entity Declaration* all'interno della stessa entità
- Ogni architettura rappresenta un diverso aspetto o modalità di realizzazione delle funzionalità
- Descrizione BEHAVIORAL (COMPORTAMENTALE)
  - Supporta descrizioni algoritmiche
- Descrizione DATAFLOW (FLUSSO DATI)
  - Descrive il trasferimento dei dati da registro a registro
- Descrizione STRUCTURAL (FLUSSO DATI)
  - Strutture composte con diversi livelli gerarchici

Esempio (1/5) – Analisi e Specifica

- L'entità è dotata di due ingressi ed un'uscita
- I segnali sono monodimensionali (1 bit…)
- Se ambedue gli ingressi sono a livello logico basso (0), l'uscita è a livello logico alto (1)
- In tutti gli altri casi, l'uscita è bassa (0)



Ingressi		Uscita
A	B	C
0	0	1
0	1	0
1	0	0
1	1	0

Esempio (2/5) – Entity Declaration

```
entity ENTITY_NAME is
    port (PORT_NAME : DIRECTION TYPE;
                    …
          PORT_NAME : DIRECTION TYPE );
end ENTITY_NAME;
```

- La definizione dell'interfaccia inizia con la parola chiave **entity**
- ENTITY_NAME è il nome dell'interfaccia, da usare come riferimento
- PORT_NAME: Il nome dei segnali che permettono all'interfaccia di comunicare con il mondo esterno
- DIRECTION: in/out/inout, indica la direzione del segnale rispetto all'entità stessa
- TYPE: indica il tipo del segnale e con esso l'insieme dei valori ammissibili e le operazioni definite (es. interi, reali, valori logici, …)
- La definizione dell'interfaccia termina con la parola chiave **end** e il nome dell'interfaccia stessa

Esempio (3/5) –Entity Declaration

```
entity NOR_GATE is
   port ( A, B : in bit;
            C : out bit );
end NOR_GATE;
```

- L’entità ha nome NOR_GATE
- A e B le porte di ingresso, ciascuna ampia 1 bit
- C è la porta di uscita del risultato, anch’essa di 1 bit
- Le porte dello stesso tipo e direzione possono essere elencate consecutivamente, separate da virgole (,)
- Le porte di tipo o direzione differente vanno separate da ;
- L’ultima definizione di porta **NON** va terminata da ;

Esempio (4/5) – Architecture Body

architecture BODY_NAME **of** ENTITY_NAME **is**

istruzioni dichiarative

begin

istruzioni funzionali del modello

end BODY_NAME**;**

- La definizione dell'interfaccia inizia con la parola chiave **architecture**
- BODY_NAME è il nome di riferimento dell'architettura
- ENTITY_NAME è il nome dell'entità di cui stiamo descrivendo le funzionalità
  - Serve ad associare un'architettura all'entità corrispondente
- Le *istruzioni dichiarative* permettono di dichiarare variabili, segnali o sottocomponenti che verranno usati nell'architettura
- Le *istruzioni funzionali* permettono di definire il valore delle porte di uscita a diversi livelli di astrazione
  - Queste istruzioni sono **CONCORRENTI**

Esempio (5/5) – Architecture Body

```
architecture DATA_FLOW of NOR_GATE is
   begin
        C <= A nor B;
end DATA_FLOW;
```

- L'Architecture Body definito ha nome DATA_FLOW ed è associato all'entità NOR_GATE definita nelle slide precedenti
- Non ci sono istruzioni dichiarative
- Il corpo funzionale è formato da una sola istruzione *NOR*
- L'operatore <= è l'operatore di **assegnamento** tra segnali
- Ogni istruzione è terminata da ;

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*
- Una volta completata la stesura del codice, questo va compilato
  - Correggete gli eventuali errori sintattici
  - Includete le librerie:

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
```

Il Codice Sorgente...

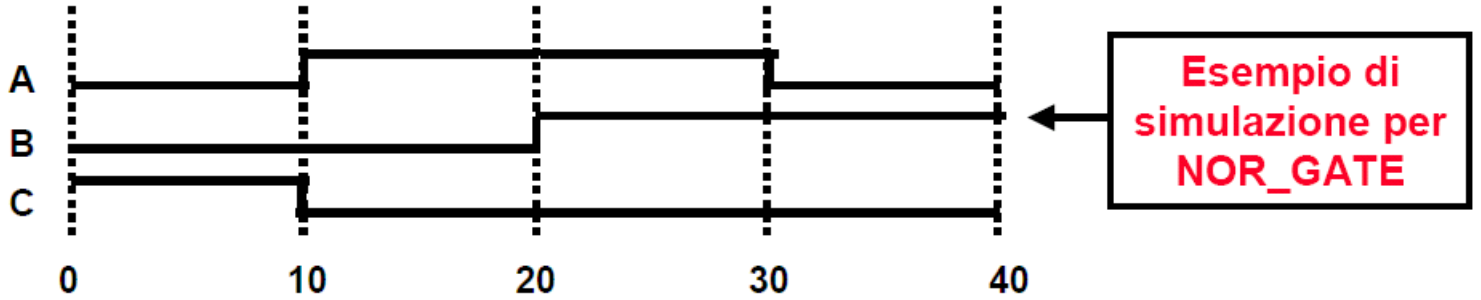
```
library IEEE;
use IEEE.std_logic_1164.ALL;

entity NOR_GATE is
    port ( A, B : in bit;
                C : out bit );
end NOR_GATE;

architecture DATA_FLOW of NOR_GATE is
    begin
            C <= A nor B;
end DATA_FLOW;
```

Verifica

- L’architettura va verificata con appositi strumenti
- Tramite un editor di forme d’onde, vengono definiti gli ingressi di test
- Con uno strumento di simulazione (ModelSim) possiamo analizzare l’andamento degli altri segnali (uscita, altri segnali ausiliari)
- Se il segnale d’uscita non corrisponde a quanto



Principi Base del VHDL (1/2)

- Diversi livelli di astrazione
  - Comportamentale
  - Flusso dati
  - Strutturale
  - Mista …
- Concorrenza
  - A livello dei modelli strutturali
  - A livello dei processi …
- Sequenzialità all'interno di un processo
  - Tutte le istruzioni definite nelle istruzioni funzionali sono concorrenti
  - E' possibile definire una sequenza di istruzioni esplicitamente sequenziali

Principi Base del VHDL (2/2)

- Gerarchia
  - Diversi livelli gerarchici di progettazione
  - Supporta progettazione top/down o bottom/up
  - I singoli componenti a livello inferiore possono essere descritti a livelli diversi di astrazione
    - Supporto alla progettazione incrementale
- Temporizzazione
  - Descrizione dell'andamento temporale dei segnali attraverso forme d'onda

Costanti e Segnali

- Una costante è un nome assegnato ad un valore fisso:
  constant name**:** type **:=** expression**;**
  constant name**:** array_type**(**index_costraint**) :=** expression**;**
- I segnali connettono le varie *entity* e comunicano i cambiamenti di valore tra i processi
  signal name**:** type**;**
  signal name**:** array_type**(**index_costraint**);**
- Esempi:

```
costant vdd : Real := 4.5;
costant FIVE_to : std_logic_vector (0 to 3) := “0101”;
costant FIVE_downto : std_logic_vector (3 downto 0):=
    “1010”;
signal count : integer range 1 to 10;
signal parity_bit : bit;
signal system_bus : bit_vector (15 downto 0);
```

Tipi – Scalari

- Real: reali da -1.0E-38 a +1.0E+38
- Integer: interi naturali positivi su 32 bit
- Boolean: false, true
- Character: ‘a’, ‘b’, …
- Bit: 0, 1
- Time: tempo in unità fisiche (fs, ps, ns, us, ms, sec, min, hr)
- Std_Logic

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
```

IEEE Std_Logic

- Disponibile nelle versioni std_logic e std_ulogic
  - Std_ulogic è completamente equivalente, ma non prevede la risoluzione dei segnali
- 9 possibili valori:
  - U : uninitialized
  - X : unknown, sconosciuto
  - 0
  - 1
  - Z : alta impedenza
  - W : weak unknown
  - L : weak 0
  - H : weak 1
  - - : don’t care, indifferenza

Vettori

- I tipi possono essere aggregati in vettori (*array*) predefiniti
  - String : array di caratteri
    - “Tipo string”
  - Bit_vector
    - “00011011”
  - Std_logic_vector
    - “11xxx00-”
- Può essere necessario dover esplicitare il tipo
  - String’(“1000”)
    - Potrebbe essere interpretato altrimenti come bit_vector o st_logic_vector

Identificativi

- I nomi seguono le solite regole sintattiche dei linguaggi di programmazione
  - Il primo carattere deve essere alfabetico
  - I seguenti possono essere alfanumerici
  - NON è case-sensitive
- Oggetti diversi devono avere nomi diversi
- Domini di validità dei nomi:
  - Entità di una libreria
  - Architetture di una entità
  - Processi di un’architettura

Operatori Logici

And	And logico
Or	Or Logico
Nand	And logico complementato
Nor	Or logico complementato
Xor	Or logico esclusivo

Operatori Relazionali

=	Uguale a
/=	Diverso da
<, >	Minore di, maggiore di
<=, >=	Minore o uguale a, maggiore o uguale a

Operatori Aritmetici (1/2)

&	Concatenazione
+	Addizione
-	Sottrazione
+	Più unario
-	Opposto (unario)

Operatori Aritmetici (2/2)

*	Moltiplicazione
/	Divisione
Mod	Modulo
Rem	Resto (conserva il segno)
**	Esponenziazione
abs	Valore assoluto
not	Complemento

Operatori VHDL93

sll, slr	Scorrimento logico a sinistra/destra
sla, sra	Scorrimento aritmetico a sinistra/destra
rol, ror	Rotazione a sinistra/destra
xnor	Or esclusivo complementato

Dichiarazioni

- Costanti (migliorano la leggibilità)
  - `constant nome: tipo := espressione;`
  - `constant nome: tipo_array[(intervallo)] := espressione`
- Segnali (connettono entità e processi)
  - `signal nome: tipo [ := espressione];`
  - `signal nome: tipo_array[(intervallo)] [:= espressione];`
- Entità
  - Rappresentano i modelli VHDL
  - Non è obbligatorio avere la definizione dell'architettura
  - E' possibile avere più architetture per ogni entità

Dichiarazioni – Architetture

- Per ogni entità specifica
  - Il comportamento
  - Le interconnessioni
  - Le relazioni tra ingressi e uscite
  - I componenti in termini di entità già definite
- Tre stili principali
  - Dataflow: Definisce implicitamente la struttura ed il comportamento
  - Structural: Definisce le connessioni tra componenti
  - Behavioral: Un processo descritto in modo sequenziale

Architettura Dataflow

- L’insieme di espressioni è eseguito in modo concorrente
  - La posizione relativa di istruzioni/espressioni è ininfluente
- Le categorie di espressioni possibili sono
  - Assegnamento di segnali
  - Assegnamento condizionato di segnali
  - Assegnamento selettivo di segnali
  - Istanze di componenti
  - Dichiarazioni di blocchi
  - Procedure, asserzioni, processi

Assegnamento di Segnali (1/2)

- Sintassi:
  - `nome_segnale <= valore [after ritardo];`
- Esempi
  - `sum <= a xor b after 5ns;`
  - `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 di Segnali (2/2)

- 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');`

Assegnamento Condizionato

- Sintassi
  - ```
    segnale <= valore1 when condizione1 else
               valore2 when condizione2 else
               … …
               Altro_valore;
    ```

  o

  - ```
    if condizione1 then
               segnale <= valore1
    else if condizione2 then
               segnale <= valore2
               … …
    else
               segnale <= Altro_valore;
    ```

condizione è **boolean**

else deve essere presente

La prima espressione vera viene assegnata

Possono essere usati i ritardi

Assegnamento Selettivo

- Sintassi
  - ```
    with espressione select
            segnale <= valore1 when caso1,
            segnale <= valore2 when caso2,
            … …
            espressioneN when casoN
            [,Altra_espressione when others];
    ```
- **others** può essere esclusa se tutti i casi sono considerati
- Non si possono sovrapporre i casi
    ```

Blocchi

- Sintassi
  - [etichetta:] **block** [(condizione_guard)]
    dichiarazioni
    begin
    istruzioni concorrenti
    end block [etichetta];
- Nel blocco possono essere dichiarati costanti, segnali, procedure
- La condizione_guard agisce da interruttore per le istruzioni definite all'interno del blocco
  - Le istruzioni sono identificate dalla parola chiave **guarded**

Architettura Strutturale (1/2)

- Permette di definire, dichiarare e usare dei componenti per costruire un'architettura
- Dichiarazione
  - `component nome_locale_componente`
    `port(elenco_porte);`
    `end component;`
- Configurazione: scelta dell'architettura
  - Se non viene effettuata, viene usata l'architettura predefinita (in genere l'ultima descritta)
  - `for nomi_oggetti : nome_locale_componente`
    `use entity nome_entità(nome_architettura);`

Architettura Strutturale (2/2)

- Utilizzo: i componenti sono istanziati
  - `NOME: nome_locale_componente`
    `port map(assegnamento posizionale o nominale);`
- Nella dichiarazione vengono definiti i parametri *formali*
- Nella istanza vengono usati i nomi *locali* (parametri *attuali*)
- Eventuali valori predefiniti (generic) possono essere sovrascritti

Architettura Behavioral

- Descrive un'architettura con una sintassi algoritmica
- L'insieme di istruzioni è eseguito **sequenzialmente** all'interno di un ***processo***
- Il processo, nella sua interezza, è una sola istruzione *concorrente*
- Permette l'uso di variabili
  - La visibilità delle variabili è limitata al processo
  - Vengono dichiarate come

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

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 di un assegnamento	Non immediato	Immediato

Processi

- Sintassi

```
[etichetta:] process [(lista di sensibilità)]
dichiarazioni (variabili…)
begin
istruzioni sequenziali…
end process [etichetta];
```

- La lista di sensibilità indica i **segnali** le cui variazioni causano l'attivazione del processo
- La lista può essere sostituita dall'istruzione

```
wait on lista_segnali;
```

- La combinazione di diverse istruzioni **wait** permette di definire comportamenti complessi
  - La lista di sensibilità è globale

Processi – Esecuzione

- Il processo è eseguito una prima volta durante l'inizializzazione
  - Completamente se definito tramite lista di sensibilità
  - Fino alla prima **wait** in caso contrario
- Alla variazione dei segnali della lista di sensibilità, il processo viene riavviato
- Alla variazione dei segnali nella lista di attesa, il processo riprende l'esecuzione
- I segnali sono aggiornati solo alla terminazione del processo
- L'esecuzione al suo interno è strettamente sequenziale

Wait

- **wait;**
  - Sospende il processo a tempo indefinito
  - Utile in fase di test
- **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

Costrutti Condizionali – if

- Sintassi:

```
if condizione1 then
  istruzioni sequenziali;
{elsif condizione2 then
  altre istruzioni sequenziali;}
[else
  ultime istruzioni sequenziali;]
end if;
```

Costrutti Condizionali – case

- Sintassi:

```
case espressione is
  when scelta1 => istruzioni sequenziali;
  … … …
  when sceltaN => altre istr. sequenziali;
  [when others => ultime istr. sequenziali;]
end case;
```

- Equivale a **with** … **select**
- Tutte le scelte devono essere incluse
  - E’ possibile utilizzare **others**
- Le scelte non possono sovrapporsi
- E’ possibile definire degli intervalli (1 to 4, …)

Cicli

- `[etichetta:] for indice in intervallo loop`
  `istruzioni sequenziali;`
  `end loop [etichetta];`
  - indice è intero e non può essere modificato all'interno del ciclo
  - intervallo è di tipo enumerativo
- `[etichetta:] while condizione loop`
  `istruzioni sequenziali;`
  `end loop [etichetta];`
  - La condizione è valutata prima di ogni iterazione
- `Exit [etichetta:] [when condizione];`
  - Termina l'esecuzione del ciclo specificato (anche se non il più interno)
- `Next [etichetta:] [when condizione];`
  - Passa alla prossima iterazione