VHDL

NOZIONI DI CARATTERE GENERALE

Il *VHDL* è un linguaggio di descrizione dell'*hardware* adatto a modellare *sistemi digitali* sia per scopi di sola documentazione o *simulazione*, che per scopi di *sintesi* o per *implementare dispositivi reali*. La sua principale caratteristica è l'*indipendenza tecnologica*, ovvero la capacità di descrivere qualsiasi sistema digitale senza vincolare la descrizione a dispositivi reali particolari.

RAPPRESENTAZIONE HARDWARE

Un sistema hardware si può sempre rappresentare in questi termini: *riceve* degli *input*, *esegue* delle *operazioni*, *produce* degli *output*. In generale sarà costituito da uno o più sottosistemi rappresentabili in questi termini. Allora per descrivere un sottosistema si devono definire:

- la sua *interfaccia esterna*, ovvero gli ingressi e le uscite che rappresentano le sue relazioni con gli altri sottosistemi o con l'esterno;
- il suo *funzionamento interno*, ovvero che cosa fa il sistema e/o come lo fa.

In VHDL il *modello di un sottosistema* è detto *design entity* ed è costituito da due parti: una *entity* (che descrive l'*interfaccia*) e una *architecture* (che descrive il *funzionamento*).

Nella descrizione hanno un ruolo fondamentale i *segnali* che "spostano" i dati tra parti diverse del sistema. Le *porte* dell'*interfaccia esterna* sono segnali particolari perché spostano i dati da e verso l'esterno.

ENTITY

L'interfaccia della *design entity* rappresenta il sottosistema *esternamente*, per mezzo dei canali che servono a interagire con altri sistemi. L'istruzione **`entity`** dichiara una nuova *design entity*, ne definisce il nome e, al suo interno, l'istruzione **`port`** definisce le *porte* di *ingresso* e *uscita* dell'*interfaccia*.

Le *porte* sono segnali disponibili all'interno, per l'*architettura*, e all'esterno, per il *collegamento* con altre *entità*. Per ogni porta si devono definire: il *nome* del *segnale*, il *modo* o *direzione* (*in*, *out*, *inout*) e il *tipo* di *dato* (*signed*, *unisgned*, *std_logic*, ecc...).

ECCO UN ESEMPIO PER CAPIRE LA SINTASSI:

```
entity nome_entità is
    port (  nome_porta : direzione tipo;
            nome_porta : direzione tipo
            );
end nome_entità;
```

ARCHITECTURE

Dopo aver definito l'interfaccia di una *design entity*, si passa a descrivere il suo effettivo *funzionamento interno* con un'*architecture*. Il *corpo* (*body*), che inizia con la parola chiave **begin**, è la parte dell'*architettura* dove si inseriscono le istruzioni descrittive ed è un'*area concorrente*.

In un sorgente VHDL le aree di codice disponibili a *descrizioni concorrenti* o *sequenziali* sono dette rispettivamente *aree concorrenti* e *aree sequenziali*. Nelle prime si usano istruzioni concorrenti, la cui esecuzione è indipendente dall'ordine in cui vengono scritte. Nelle seconde si usano istruzioni sequenziali che vengono eseguite nell'ordine in cui sono scritte.

Il corpo può essere preceduto da un'*area dichiarativa*. Nell'architettura si possono utilizzare come *segnali* le *porte* dichiarate nell'*entità*.

ECCO UN ESEMPIO PER CAPIRE LA SINTASSI:

```
architecture nome_architettura of nome_entità is

{eventuale_dichiarazione}

begin

    {istruzione_concorrente}

end nome_architettura;
```

COMPONENT

Un *componente* è una *design entity* che viene riusata più volte nel modello. Creare un'*istanza* di un *componente* significa usare una copia di una *design entity* che è già stata definita da qualche altra parte.

Un'*istanza* è un'*istruzione concorrente* che crea la copia del componente nel punto desiderato del modello e collega le sue *porte* ad altri *segnali* dell'*architettura*. In questo modo si ottiene una descrizione strutturale *gate-level* equivalente a quella che si disegnerebbe in uno schematico.

In un *progetto gerarchico* si utilizza un *istanza indiretta* per inserire un *componente* all'interno di una *architettura*.

Nell'istruzione di *istanza* si deve definire il *nome* da assegnare all'*istanza*, indicare il *nome* dell'*elemento istanziato* e *mappare* le *porte*. L'istruzione da utilizzare nei due casi è la stessa, salvo che il nome dell'elemento istanziato indica due cose diverse.

Nella **port map** (*mappatura* delle *porte*) si associano le *porte* (parametri formali) del *componente istanziato*, a *segnali* (parametri attuali) dell'*architettura corrente*. La mappatura si può definire in due modi:

- *associazione posizionale*: si inserisce la lista dei segnali attuali nello stesso ordine in cui sono definite le porte del componente istanziato;
- *associazione per nome*: si inserisce la lista delle coppie di oggetti formale e attuale (separati dal simbolo =>) in un ordine qualsiasi.

La dichiarazione del *component* va fatta nell'*area dichiarativa* di un'*architettura*. Il nome del *component* e la sua *dichiarazione* delle *porte* devono essere quelli dell'*entity* a cui si riferisce affinché un *component* faccia riferimento implicitamente a una precisa *entity*.

ECCO UN ESEMPIO PER CAPIRE LA SINTASSI (DICHIARAZIONE):

```
component nome_componente
    port (  nome_porta : direzione tipo;
            nome_porta : direzione tipo
            );
end component
```

ECCO UN ESEMPIO PER CAPIRE LA SINTASSI (ISTANZA):

```
nome_istanza : nome_componente port map (  formale => attuale,
                                           formale => attuale
                                           );
```

I caratteri e le parole scritte in *corsivo* e puntinate sono omissibili. Un'analoga formattazione sarà mantenuta anche successivamente. Nel caso contestuale la stringa "`formale =>`" può essere tralasciata, ricordando però che in tal modo l'ordine delle porte dovrà essere lo stesso utilizzato nella dichiarazione.

PROCESS

Un *process* è un'*istruzione concorrente* che contiene un'*area sequenziale*. Si usa nel *corpo* di un'*architettura*. Le istruzioni sequenziali che può contenere si inseriscono nel suo *body*, che inizia con la parola chiave **`begin`** e può essere preceduto da un'*area dichiarativa*.

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*. Il processo viene eseguito una volta e poi rimane sospeso. L'esecuzione riparte solo se si verificano eventi sui segnali della *sensitivity list*.

In un processo le *variabili locali* conservano in proprio valore nel tempo tra un'esecuzione e l'altra.

ECCO UN ESEMPIO PER CAPIRE LA SINTASSI:

```
process (sensitivity_list)
{eventuale_dichiarazione}
begin
    {istruzioni_sequenziali}
end process;
```