Structure d'un calculateur : **MI01**

nope

Automne 2015

Table des matières

Chapitre 1

Bases du VHDL

1.1 VHDL ????

Le *VHDL*, pour *VHSIC Hardware Description Language*, est un langage permettant de spécifier des architectures matérielles. Il permet, par le biais de synthétiseurs, de transformer des logiques/processus en matériel réel composé de portes logiques.

1.2 Structure d'un programme VHDL

La modélisation d'un composant en VHDL se fait par le biais de deux structures : *Entity* et *Architecture*. Ces deux structures fonctionnent ensemble afin de modéliser de façon complète un composant.

1.2.1 Entity

Cette structure permet de définir quelles seront les interfaces du composant. On ne modélise pas son contenu, ni son comportement, mais juste les entrées/sorties du composant.
Ces entrées/sorties d'un composant s'appellent **ports** en VHDL.
La structure *entity* se présente généralement de cette façon :

```
ENTITY nomEntite IS
PORT(entreeExemple : IN typeSignal, sortieExemple : OUT typeSignal);
END
```

Un exemple de structure *entity* pour le composant *inverseur* peut être le suivant :

```
ENTITY inverseur IS
PORT (entree : IN BIT, sortie : OUT BIT);
END inverseur;
```

Dans cet exemple, on a un composant *inverseur* qui a donc un fil d'entrée qui s'appelle *entree* et un fil de sortie s'appelant *sortie*.

1.2.2 Architecture

Cette structure permet d'implémenter le composant en lui-même ; elle est donc associée à une entité. On va ici définir comment se comporte notre composant. La structure *architecture* se présente sous la forme suivante :

```
ARCHITECTURE nomArchitecture OF entiteEnQuestion IS
-- déclarations
BEGIN
-- instructions
END nomArchitecture;
```

Pour reprendre l'exemple de l'entité *inverseur*, son *architecture* serait :

```
ARCHITECTURE architectureInverseur OF inverseur IS
BEGIN
sortie <= NOT e;
END architectureInverseur;
```

1.3 Types de données en VHDL

1.3.1 Conteneurs et types

Une donnée en VHDL se déclare de la façon suivante :

```
CONTENEUR nom : TYPE ;
```

Il existe 3 conteneurs en VHDL :
- Les constantes - *CONSTANT* -, qui prennent une valeur qui ne bougera pas ;
- Les variables - *VARIABLE* -, qui ont une valeur modifiable ;
- Les signaux - *SIGNAL* -, qui modélisent une entrée/sortie entre deux processus d'une fonction ;

Ces conteneurs peuvent contenir des données ayant un certain type parmi les suivants :
- Type entier : *INTEGER* ;
- Type bit (0/1) : *BIT* ;
- Type vecteur - aka tableau de taille n : type_*VECTOR* (0 *TO* n-1) ;
- Type énuméré : on créé un type qui peut prendre des valeurs qu'on définit [1] ;

Les types logiques standard

Après avoir lu les différents types énoncés, on peut se dire qu'il est logique d'utiliser le type *BIT* pour les signaux. En réalité, il existe des *standards* qui sont à privilégier sur le type *BIT* décrit plus en haut : ces standards sont les types ***std_logic*** et ***std_logic_vector***.
Par rapport au type *BIT*, ces deux types nécessitent l'utilisation de la bibliothèque standard IEEE et apportent deux valeurs en plus du 0 et 1 : Z et X, qui sont utilisés lorsque on n'est pas sur de la validité du signal.
Plus important encore, elles permettent l'utilisation des opérateurs logiques qui seront décrits plus loin.
Pour inclure la bibliothèque standard IEEE, il faut mettre au début du programme VHDL :

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all
```

1.4 Opérateurs principaux en VHDL

1.4.1 Opérateurs d'affectation

Il y a deux opérateurs d'affectation qui dépendent du conteneur :
- Pour un *signal*, on utilise *<signal>* **<=** *<valeur>* ;
- Pour une variable/constante, on utilise *<variable>* **:=** *<valeur>* ;

1. par exemple *TYPE feu IS (vert,orange,rouge)* définit un nouveau type qui ne peut prendre que les valeurs 'vert', 'orange', ou 'rouge'.

1.4.2 Opérations logiques

Sur le type *std_logic*, il est possible d'utiliser les opérations *AND, OR, NOT, XOR*. Ces opérations fonctionnent avec les valeurs *true* ou *false*.

1.4.3 Opérateurs arithmétiques et de comparaison

Il existe bien sur les opérateurs classiques de comparaison : = /= < <= > = ainsi que les opérateurs mathématiques + - * /.

1.5 Structures de contrôle

1.5.1 Affectation conditionnelle

L'affectation conditionnelle prend la forme des mots clés *WHEN, ELSE* en VHDL. C'est l'équivalent du *if, else* dans un langage de programmation *traditionnel*. Une affectation conditionnelle générale prend la forme suivante :

```
signal_random <= 'valeur' WHEN 'condition' [ELSE]
 ['valeur2' WHEN 'condition2' ELSE]
 ...;
```

Chapitre 2

Modélisation matérielle

2.1 Les types de modélisation

Il est possible de modéliser un même composant sous deux façons différentes : de façon comportementale ou structurelle.

2.1.1 Modélisation comportementale

La modélisation comportementale consiste, comme son nom l'indique, à écrire comment le circuit doit se comporter, son fonctionnement effectif. Cette modélisation se fait par le biais des opérateurs décrits plus en haut.
Dans cette modélisation, c'est le synthétiseur qui va choisir les circuits et composants du nouveau composant en fonction des instructions logiques écrites.
Un exemple de modélisation comportementale d'un "non et" serait :

```
s <= NOT(x AND y);
```

2.1.2 Modélisation structurelle

Dans cette modélisation, on modélise le comportement à travers des composants. En fait, on spécifie exactement quels seront les composants utilisés pour arriver au comportement souhaité, au lieu de spécifier le comportement de façon abstraite.
En VHDL, l'utilisation d'un composant se fait par le mot-clé *PORTMAP*.
L'utilisation générale se fait de la manière suivante :

```
nomComposant : composantUtilisé PORTMAP (<signaux d'entrée du composant>);
```

Dans l'exemple suivant, on souhaite utiliser le composant *additionneur1bit*, qui prend en entrée *a,b* qui sont les bits à additionner, et *Cin* qui est un bit de retenue.

```
c1 : additionneur1bit PORTMAP (a,b,Cin);
```

Le composant c1 est donc un composant de type *additionneur1bit* qui prendra en entrée les signaux *a,b,Cin*.

2.2 Logique séquentielle / Machines à états

2.2.1 Concurrentiel vs Séquentiel

Il y a deux logiques utilisables lors de la modélisation d'un composant : on peut penser de façon *concurrentielle* ou de façon *séquentielle*.

Comme leurs noms l'indiquent, ces deux logiques différent sur l'ordre d'exécution des commandes. En logique *concurrentielle*, il n'y a pas d'ordre d'exécution pour chaque instruction. Les instructions peuvent donc s'effectuer dans n'importe quel ordre, ce qui est peu utilisable pour résoudre la majorité des problèmes.
La logique *séquentielle*, quant à elle, possède un ordre d'exécution, comme les programmes *classsiques*. On parle alors de *processus*, ou *process*.

2.2.2 Machines à états

Les machines à états sont des graphes permettant de modéliser une séquence à travers ses états. Les machines à états sont très utiles à la modélisation séquentielle puisque ils permettent de voir exactement l'état du système à travers le temps, les entrées et ses sorties.
Comme énoncé dans le cours, la machine a états permet de voir quelle séquence sera exécutée en fonction des entrées.

Problème du chariot

todo

2.2.3 Le séquentiel en VHDL

Le séquentiel prend la forme de *PROCESS* en VHDL.
Un *process* est déclenché par des signaux d'entrées. Si des process contiennent le même signal de déclenchement [1], ils sont déclenchés en même temps.
Les process permettent l'accès à des instructions/constructions bien plus élaboreés que celles disponibles lors d'une modélisation concurentielle. On a donc accès à des boucles - *LOOP* -, des tant-que - *WHILE* -, ...

1. Ces signaux sont regroupés dans une *liste de sensibilité* par process