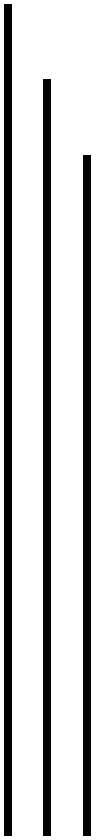
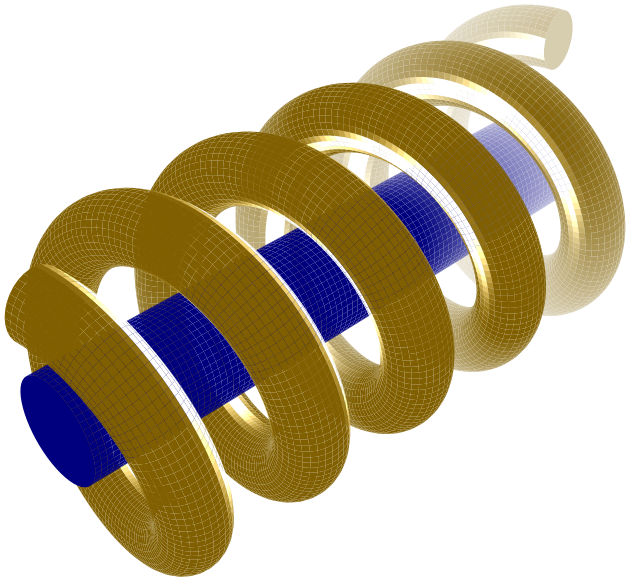


LANGAGE DE DESCRIPTION DU MATÉRIEL

Jorge Luiz MAYORAL



Cergy, 25 Septembre 2004

EISTI

TABLE DES MATIÈRES

2 †

1

MAX+PLUS II 9.23 Basilene

Richard FAGOT

1.1 Introduction

Devant la diversité des logiciels pour l'édition, la synthèse et l'implémentation de code VHDL tels *Xilinx*®, *Altera*® ou *Actel*®, il est apparu inévitable de faire un choix pour que les étudiants bénéficient d'un outil à la fois puissant et fonctionnel et ceci à moindre frais pour l'école. *Xilinx*® ayant déjà été évalué, c'est au tour du logiciel *Max+Plus*® *II d'Altera*® d'être testé. Ce logiciel présente l'intérêt supplémentaire d'être livré avec une carte supportant deux composants *FPGA* (Field Programmable Gate Array).

Les composants *FPGA* sont les prémisses des futurs *FPPA* (*Field Programmable Process Array* qui sont un ensemble de processeurs programmables permettant de configurer l'architecture des *FPPA* pour résoudre des systèmes complexes.

De plus, actuellement l'état de l'art dans le domaine des microprocesseurs sont les machines non synchrones qui permettront de réduire la consommation d'énergie, d'augmenter la vitesse et d'augmenter la quantité de blocs logiques par mm^2 grâce à l'élimination de l'horloge.

1.2 Carte d'expérimentation

Cette carte possède deux composants *FPGA* (*Field Programmable Gate Array*) et est ainsi divisée en deux parties organisées autour des deux composants *FPGA* :

① Le premier (EPM7128S) est entouré de deux afficheurs à 7 segments, deux blocs de 8 mini-interrupteurs, de deux boutons poussoirs et de quatre connecteurs externes permettant de brancher d'autres périphériques ;

② Le deuxième (*FLEX10K20*) est entouré de deux afficheurs à 7 segments, d'un bloc de 8 mini-interrupteurs, de deux boutons poussoirs et d'une sortie *Video Graphics Array - VGA*.

Elle possède également une entrée pour l'alimentation, un port parallèle pour la communication avec l'ordinateur, un port *PS2* permettant de connecter une souris ou un clavier et trois cavaliers permettant de choisir quel composant sera programmé.

1.2.1 EPM7128S

De la famille des *MAX7000* l'*EPM7128S* est construit sur des éléments *Electricaly Erasable Programable Read Only Memory - EEPROM*. Il possède 84 broches dont 34 (au moins) programmables. Ce composant a une capacité de 2500 portes logiques. Son architecture simple fait qu'il est parfaitement adapté à la découverte de la logique combinatoire.

1.2.2 FEL10K20

Ce composant appartient à la famille des *FLEX10K*. Il est construit sur des éléments reconfigurables de type (*Static Random Access Memory - SRAM*), possède 240 broches dont 31 programmables, est composé de 1152 éléments logiques et blocs de tableaux incorporés (EAB *Embeded Array Block*). Chaque EAB fournit une mémoire de 2048 bits. Le FLEX10K est idéal pour les programmes avancés tel qu'une architecture d'ordinateur.

1.3 Projet : Additionneur

Ce mini-projet a pour objectif de fournir une première approche du logiciel. De conception simple il permet d'utiliser les fonctionalités les plus importantes du logiciel.

1.3.1 Édition

Lancer le logiciel Max+Plus 2 à partir de :

① Start → programme → Max+ Plus II 9.23 Baseline → Max+ Plus II 9.23 Baseline.

La fenêtre Max+plus II Manager s'ouvre.

Créer un document de type texte en faisant :

① *File → New*.
- — *Text editor file*;
- — *OK*.

Une fenêtre intitulée *Untitled* s'ouvre. Nous sauvegardons le contenu par :

① *File → Save*.

- Mettre un nom dans *file name* (ce nom sera celui de votre entité) ;
- Sélectionner dans *automatic extention* l'extenstion *VHD* ;
- Vérifier que c'est bien l'extension dans *File name* ;
- *OK*.
- *File → Set Projet to Current File → Save & Check.*

A partir de là nous pouvons commencer à saisir le code VHDL. Sur cet exemple, le programme est fait pour la carte de test. Or, sur cette carte les boutons poussoirs sont à 0 quand nous appuyons dessus et les leds des afficheurs s'allument quand nous passons à l'état logique 0. C'est pourquoi les front montants et descendants et les sorties des afficheurs peuvent sembler inversés, c'est pour respecter les spécifications de la carte de test.

1.3.1.1 Appel des librairies

L'appel des librairies est illustré par le Script 1.3.1.

Script 1.3.1 ***Appel des libraries.***

```
1    LIBRARY IEEE;
2    USE IEEE.STD_LOGIC_1164.all;
3    USE IEEE.STD_LOGIC_ARITH.all;
4    USE IEEE.STD_LOGIC_UNSIGNED.all;
```

C'est la librairie élémentaire à charger avec ses extensions principales. Elle permet l'utilisation du type *std_logic*.

1.3.2 Définition de l'entité

Dans ce cas, l'entité a pour nom *additionneur*. C'est le nom de fichier qui a été introduit plus haut et le code est illustré par le Script 1.3.2.

Script 1.3.2 ***L'entité additionner.***

```
1    ENTITY additionneur3 IS
2    PORT(
3         clk           : IN  STD_LOGIC;
4         a,b,c,d,e,f,g : OUT STD_LOGIC
5         );
6    END additionneur3;
```

Notre additionner sera constitué d'un signal d'horloge en entrée et de 7 signaux de sortie destinés à l'afficheur (tous ces signaux sont du type *std_logic*).

1.3.3 Définition de l'architecture

Le code de l'architecture est illustré par le Script 1.3.3.

Script 1.3.3 ***L'entité additionner.***

```
1     ARCHITECTURE arch OF additionneur3 IS
2     SIGNAL S   : STD_LOGIC_VECTOR(2 DOWNTO 0);
3     SIGNAL val : STD_LOGIC_VECTOR(3 DOWNTO 0);
```

Nous avons besoin de deux signaux : val (signal qui est un vecteur de 4 bits de type *std_logic*), qui va s'incrémenter à chaque impulsion de l'horloge et qui contient les deux mots de 2 bits à additionner. *S* (signal qui est un vecteur de 3 bits de type *std_logic*), sur 3 bits qui contient le résultat de la somme retenue comprise. Le niveau d'abstraction choisi est le niveau comportemental, donc le code est le suivant :

1.3.3.1 Premier PROCESSUS

Le premier processus est illustré par le Script 1.3.4.

Script 1.3.4 ***Le process.***

```
1    BEGIN
2     PROCESS(clk)
3    BEGIN
4    IF (clk'EVENT AND clk='0') THEN
5      IF (val = "1111") THEN
6          val <= "0000";
7      ELSE
8          val <= val + "0001";
9      END IF;
10     S<=val(1 downto 0)+val(3 downto 2);
11   END IF;
12  END PROCESS;
```

Ce processus réalise trois opérations de manière synchrone : le premier *if* contient la condition de synchronisation. Suit le compteur qu'on réinitialise à 0 si il dépasse sa valeur maximale et qu'on incrémente de 1 sinon et *S* qui prend la nouvelle valeur de la somme.

1.3.3.2 Deuxième PROCESSUS

Le code du deuxième processus est illustré par le Script 1.3.5.

Script 1.3.5 ***Le deuxième processus.***

```
1    PROCESS (S,clk)
2    BEGIN
3    IF (clk'EVENT AND clk='1') THEN
4     CASE S IS
5      WHEN "000"  =>  a <= '0';
6                      b <= '0';
7                      c <= '0';
8                      d <= '0';
9                      e <= '0';
10                     f <= '0';
11                     g <= '0';
12     when "001"  =>  a <= '1';
13                     b <= '0';
14                     c <= '0';
15                     d <= '1';
16                     e <= '1';
17                     f <= '1';
18                     g <= '1';
19     when "010"  =>  a <= '0';
20                     b <= '0';
21                     c <= '1';
22                     d <= '0';
23                     e <= '0';
24                     f <= '1';
25                     g <= '0';
26     WHEN "011"  =>  a <= '0';
27                     b <= '0';
28                     c <= '0';
29                     d <= '0';
30                     e <= '1';
31                     f <= '1';
32                     g <= '0';
33     WHEN OTHERS =>  a <= '1';
34                     b <= '1';
35                     c <= '1';
36                     d <= '1';
37                     e <= '1';
38                     f <= '1';
39                     g <= '1';
40     END CASE;
41    END IF;
42   END PROCESS;
43  END arch;
```

Dans ce *processus* nous affectons les valeurs que doivent prendre les différentes *leds* de l'afficheur de manière à ce qu'il affiche la valeur correspondante au résultat de l'addition. Ne sont pris en compte dans l'affichage que les 2 bits de poids faibles (et pas le bit de poids fort qui correspond à la retenue). On a donc 4 instances de *when* correspondants aux quatre valeurs que peut prendre le résultat de l'addition (de 0 à 3) et une cinquième instance pour tous les autres cas.

1.3.4 Synthèse

Après avoir sauvegardé notre fichier en cliquant sur l'icône représentant une disquette nous pouvons passer à la synthèse. Pour cela, il faut choisir sur la base de quel composant nous désirons faire la simulation. Dans notre cas, le *FPGA EPF10K20RC240-4*.

1.3.4.1 Sélection du composant

On commence par sélectionner le composant qui contiendra par la suite le programme en faisant :

① *Assign* → *Device*.

— *Device Family* → *FLEX10K* ;

— *devices* → *EPF10K20RC240-4* (après avoir décoché *Show Only Fastest Speed Grades*) ;

— *OK*.

1.3.4.2 Réalisation de la synthèse

Pour faire la synthèse proprement dite nous faisons :

① *Max+plus II* → *Compiler* (La fenêtre compiler s'ouvre et est active.) ;

— *Processing* → *Design Doctor* (Pour avoir des informations supplémentaires sur la synthèse) ;

② *Start*.

1.3.5 Simulation

La synthèse étant terminée nous pouvons passer à la simulation.

1.3.5.1 Création d'un fichier SCF

Avant de pouvoir faire une simulation il faut d'abord créer une *waveform* qui représentera l'évolution des différents signaux que l'on aura choisi de visualiser. Pour cela il faut faire :

① *File* → *New*.

— *Waveform Editor File*;
— *OK*.

Une fenêtre nommée *Waveform Editor* s'ouvre. Nous la sauvegardons en faisant :

① *File* → *Save*.

— nous lui donnons un nom ;

— *OK.*

1.3.5.2 Ajout des signaux

Maintenant nous pouvons ajouter les différents signaux devant intervenir dans la simulation :

① *Node → Enter Nodes From SNF.*

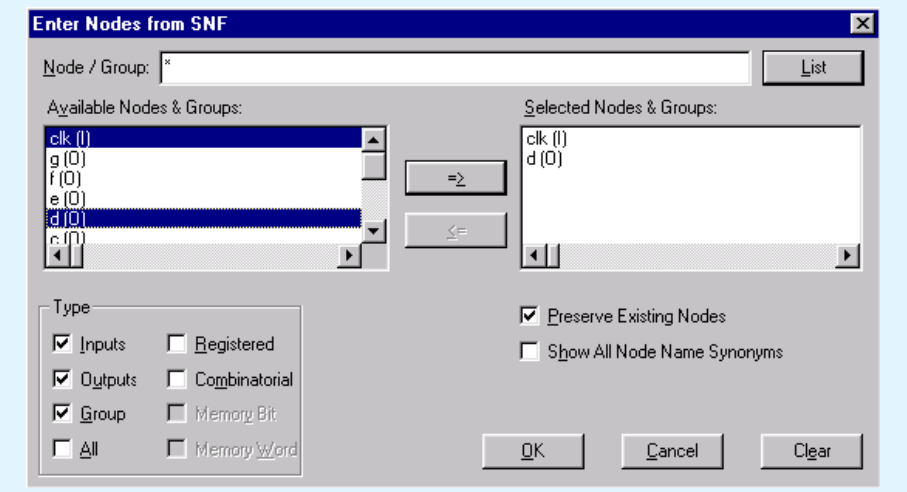
— *List* ;

— Dans la case *Available Nodes and Groups* sélectionner les signaux à visualiser en cliquant avec la souris sur chacun d'entre eux tout en maintenant la touche contrôle appuyée ;

— cliquer sur la petite flèche entre les deux cases ;

— OK.

Figure 1.1: *Configuration du composant [21].*



1.3.5.3 Initialisation

Nous fixons la durée de la simulation par :

① *File → End Time.*
— Rentrer une valeur avec son unité;
— *OK.*

Et la taille de la grille (valeur de temps élémentaire ou période de l'horloge) par :

① *Option → Grid Size*.

— Entrer une valeur avec son unité (plus petite que précédemment) ;

— *OK*.

1.3.5.4 Initialisation des entrées

Pour initialiser les entrées nous cliquons sur le signal considéré on sélectionne les parties du temps à modifier à l'aide de la souris puis nous utilisons les icônes de la barre d'outils à gauche. Pour une horloge par exemple, il suffit de la sélectionner et de cliquer sur l'icône *Clock WaveForm*.

Un clic droit de la souris sur un vecteur ouvre un menu contextuel qui propose, entre autre, de grouper (*enter group*) utile pour des vecteurs éclatés que l'on désire reconstituer ou pour un ensemble de signaux de commande d'un afficheur ou de dégrouper (*ungroup*). Nous sauvegardons (*file → save*).

1.3.5.5 Simulation

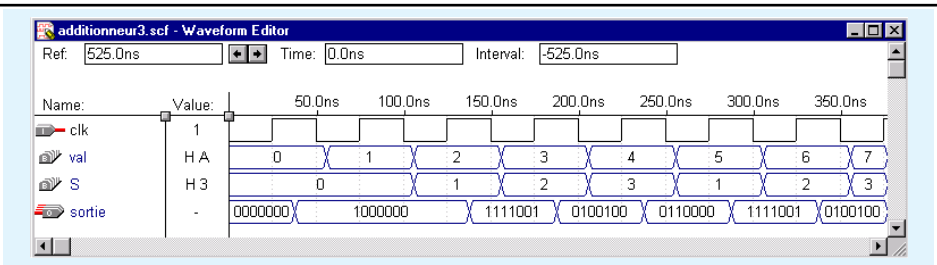
① Revenir sur la fenêtre *Compiler* ou *File → Projet → Save, Compile & Simulation*.

— Dans le premier cas, un double-cliquer sur l'icône *SNF*.

(a) *Start* ;

(b) *Open SCF*.

Figure 1.2: *La simulation [21].*



1.3.6 Implémentation

Après avoir vérifié le bon fonctionnement en simulation nous passons à l'implémentation sur le composant. Pour cela il faut d'abord configurer les broches du composant pour pouvoir après implémenter le programme.

1.3.6.1 Affectation des broches

① Revenir sur la fenêtre du programme en $\mathcal{V}$HDL ;

② Assign → *Pin/Location/Chip*.

Une fenêtre s'ouvre. Dans la première case (*Node Name*) nous mettons le nom de l'entrée/sortie à ajouter. En tapant un nom existant (dans l'exemple de l'additionner : *a, b, c, d, e, f , g ou clk*) la partie chip ressource change le bouton radio de *pin* s'activant et le contenu de *pin type* valant *input*, *output* ou *bidir* suivant le nom qu'on a mis. Nous choisissons le numéro de la broche dans *pin* et nous cliquons sur *add*.

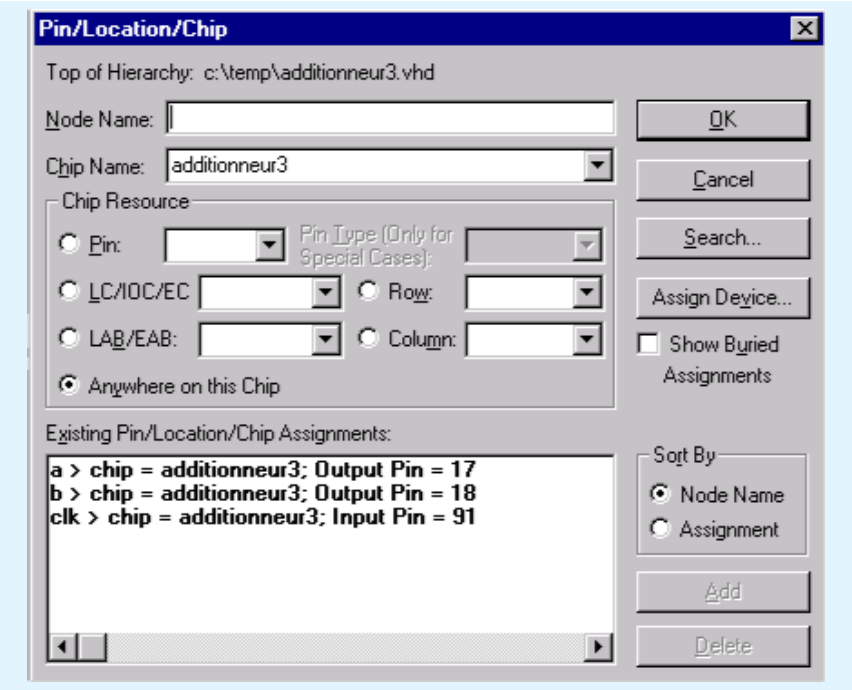
1.3.6.2 Exemple de l'additionneur

Tableau 1.3.1 *Affectation des broches.*

Nom	***Broche***
a	**17**
b	**18**
c	**19**
d	**20**
e	**21**
f	**23**
g	**24**
clk	**28**

Les broches associées à *a, b, c, d, e, f, et g* correspondent à l'afficheur 7 segments, comme l'illustre le Tableau 1.3.1.

Figure 1.3: *Les broches [21].*



Une fois toutes les broches configurées (Figure 1.3), il faut relancer la compilation. Ensuite :

① Double-cliquer sur *POF* ;

② *JTAG → Multi-Device JTAG Chain* ;

— *Device name → EPF10K20* ;

— *Select Programming File* ;

(a) Sélectionner additionneur.sof ;

(b) *OK*.

— *Add* ;

— *Detect JTAG Chain Info* ;

(a) *OK*.

— *OK*.

③ *Configure*.

1.3.7 Remarques

L'horloge étant remplacée par un bouton poussoir il se produit un phénomène de rebond qui rend le système instable. Un moyen de résoudre ce problème est de remplacer le bouton poussoir par l'horloge interne de la carte en insérant un retard par l'intermédiaire d'un compteur car sa fréquence est beaucoup trop grande pour pouvoir visualiser les changements d'états. En plus de cela un état indéterminé apparaît dont la source demeure inconnue. A part cette partie qui ne concerne que le programme implémenté tout marche parfaitement. Il faut souligner cependant la nécessité d'avoir du supprimer un signal qui servait à actualiser les sorties et qui rendait le code plus léger et plus lisible. Ceci est du au fait que sur le *10K20* cela ne marchait pas alors que sur l'*EPM7128S* tout ce passait correctement. Il reste encore, donc, trois problèmes qu'il est nécessaire de résoudre pour pouvoir utiliser le logiciel *ALTERA*© et de pouvoir développer le microprocesseur dans les meilleures conditions.

GLOSSAIRE

Accès (access type)

Voir Type accès.

Accessible en lecture (readable)

Un signal ou une variable sont dits *accessibles en lecture* s'ils peuvent apparaître à droite d'une affectation. Les paramètres de sortie de procédure (mode out) ne sont pas accessibles en lecture.

Affectation (assignment)

C'est l'instruction qui permet de modifier la valeur d'une variable, d'un signal ou d'initialiser des constantes. Pour les constantes et variables le symbole en est $:=$ tandis que pour les signaux, il faut utiliser $<=$.

Affectation gardée (guarded assignment)

C'est une affectation de signal conditionnée par la valeur d'un signal *GUARD* de type booléen. Elle s'utilise principalement dans un bloc gardé (auquel cas le signal *GUARD* est la condition de garde du bloc) et se note par l'utilisation du mot clé guarded.

Alias (alias)

Déclarer un alias sert à donner un nom différent à un objet. Le nom original continue à être utilisable. Le nouveau nom peut désigner une structure différente de celle désignée par le nom original: il est possible de renommer, par exemple, un bit particulier d'un vecteur de bits.

Allocateur (allocator)

Voir Allocation.

Allocation (allocation)

C'est l'opération qui permet d'allouer la place mémoire nécessaire pour stocker les objets pointés: l'instruction particulière new permet de réserver une nouvelle zone mémoire. Cette allocation peut aussi être utilisée pour initialiser la valeur de l'objet pointé. L'opération inverse de l'allocation est la libération.

Architecture (architecture)

C'est une des cinq unités de conception *VHDL*. Elle se compile séparément. Une architecture décrit la vue interne d'une entité. C'est là qu'est décrit soit le comportement, soit la structure du modèle. Une spécification d'entité peut avoir plusieurs architectures associées, chacune de ces architectures correspond généralement à un niveau de finesse de description différent.

Registre (record)

C'est une famille de types qui a la particularité de réunir des éléments de types éventuellement différents. Ces éléments sont désignés par un nom et sont appelés *champs* de l'article. Avec les tableaux, les articles font partie de la famille des types composites.

Assertion (assert statement)

C'est une instruction, existant sous forme d'instruction séquentielle ou concurrente, qui permet d'envoyer un message donné si une condition indiquée n'est pas réalisée.

Association par nom (named association)

Une association de paramètres de sous-programmes, de paramètres génériques ou de ports est dite *faite par nom* si le nom du paramètre ou port formel apparaît explicitement. L'association peut aussi se faire de manière positionnelle ou même mixte (positionnelle et par nom) si l'association positionnelle est en tête.

Association positionnelle

Une association de paramètres de sous-programmes, de paramètres génériques ou de ports est dite *positionnelle* si le nom du paramètre ou port formel n'apparaît pas. L'association est faite en suivant l'ordre d'écriture des paramètres ou des ports. L'association peut aussi se faire par nom ou même mixte (positionnelle et par nom) si l'association positionnelle est en tête.

Attribut (attribute)

C'est un ensemble de propriétés d'un type, d'un label, d'une variable ou d'une entité. Il existe deux sortes d'attributs : ceux qui sont predefines, et ceux définis par le concepteur. Leur référence se fait par le caractère: ' (nommé apostrophe, quote, tick ....) suivi du nom de l'attribut. La valeur d'un attribut défini par le concepteur est calculée à l'élaboration et constante ensuite.

Bibliothèque (library)

C'est l'endroit où le compilateur stocke les unités de conception qu'il a compilées et qui sont donc exactes du point de vue du langage. Il est possible de référencer (c'est-à-dire de *voir,* d'avoir accès) le contenu d'une bibliothèque en utilisant une clause de la bibliothèque.

Bibliothèque de ressources (resource library)

Des unités de conception appartenant aux bibliothèques ressources peuvent être référencées. Pour cela, il est nécessaire que la bibliothèque dans laquelle elles se trouvent sont, soit référencée par une clause library. Cette clause doit précéder l'unité de conception qui se sert de la ressource. Le nombre de ces bibliothèques n'est pas limité par la norme.

Bibliothèque de travail (working library)

A un instant donné, une seule bibliothèque est *bibliothèque de travail*, les autres sont des *bibliothèques de ressources.* Le choix de la bibliothèque de travail est fait hors langage par une commande liée à la plate-forme *VHDL* utilisée. Elle peut être référencée en *VHDL*

par le nom logique *WORK*. Concrètement, c'est dans cette bibliothèque que seront rangées les unités de conception compilées avec succès.

Bloc (block)

C'est le support de la hiérarchie en *VHDL*. C'est aussi un moyen d'encapsuler des déclarations et de garder des affectations de signal (bloc gardé). Un bloc peut être ouvert partout où l'on peut mettre une instruction concurrente (c'en est d'ailleurs une). Le bloc (une architecture est un bloc) et le composant sont les deux moyens de hiérarchiser une description structurelle en *VHDL*.

Bloc externe (external block)

C'est une entité, c'est-à-dire un couple spécification d'entité/architecture.

Bloc gardé (guarded block)

La condition de garde d'un bloc est une expression booléenne donnée en tête du bloc. Les affectations gardées intérieures à ce bloc ne seront exécutées que si la condition de garde est vraie.

Bloc générique (generic block)

Un bloc générique est un bloc paramètre (on parlera paramètres génériques). De l'intérieur du bloc, ces paramètres sont vus comme des constantes et peuvent être manipulés comme telles. La possibilité d'utiliser des blocs génériques (souvent des entités) est extrêmement utilisée en *VHDL*. Les bibliothèques de modèles sont très souvent génériques pour être générales.

Boîte (<>, box)

C'est une notation qui permet de déclarer un tableau non contraint, c'est-à-dire sans préciser l'intervalle de variation des indices et donc la taille du tableau. Celle-ci sera connue plus tard, lors de l'exécution.

Changement de nom (renaming)

Voir Alias et Sous-type.

Classe d'objet (class of object)

Il existe trois classes d'objets en *VHDL*: ce sont les constantes, les variables, les signaux.

Clause (clause)

Une clause est une forme de spécification. Il existe deux clauses en *VHDL*: use et library. Elles servent à indiquer que l'on veut pouvoir accéder à des objets contenus dans des unités de conception (clause use) ou bibliothèques (clause library).

Commentaires (commente)

Les commentaires sont des informations destinées à améliorer la lisibilité d'un programme. Ils sont ignorés par l'analyseur. En *VHDL*, les commentaires commencent par

deux tirets – et se finissent avec la ligne. Il est inutile de rappeler le grand intérêt des commentaires pour la maintenabilité d'une description : no comment.

Compatibilité

Voir types compatibles.

Composant (component)

C'est l'idée que l'on se fait d'une entité (spécification d'entité/ architecture) qui servira plus tard à *instancier* des composants réels mais qui peut très bien ne pas (encore) exister. Comme le bloc, c'est un élément de la structuration en *VHDL*.

Concurrence (concurrency)

Les concepts de concurrence et de parallélisme sont souvent confondus. Des instructions parallèles s'exécutent en fonction de priorités données par le système, sans autre point de synchronisation que ceux que l'on programme explicitement (rendez-vous, sémaphores ...). Le fond de ce mécanisme est non déterministe et la notion de temps vrai est sous-jacente. Pour des instructions concurrentes, l'ordre des exécutions est indifférent. C'est ce qui permet de mettre en oeuvre un parallélisme effectif, mais la notion de temps vrai a disparu.

Condition de garde d'un bloc (guard expression)

C'est une expression booléenne, qui, indiquée en tête d'un bloc (le bloc est alors dit bloc gardé), conditionne les affectations de signaux internes à ce bloc.

Configuration (configuration)

C'est une des cinq unités de conception *VHDL*. Elle sert à indiquer la correspondance entre une instance de composant et un modèle (un couple spécification d'entité/ architecture). Les correspondances entre les ports du composant et ceux du modèle, ainsi qu'entre paramètres génériques locaux et formels peuvent se faire dans cette unité de conception.

Constante (constant)

Les constantes constituent une des trois classes d'objets VHDL Initialisées à une valeur, elles ne peuvent plus être modifiées par la suite. La valeur d'initialisation doit être du même type que celui de la constante. Elle sera calculée à l'exécution et peut être une expression complexe, incluant éventuellement des appels de fonctions.

Constante à valeur différée (deferisico constant)

Une déclaration de constante sans préciser sa valeur se nommera *déclaration de constante à valeur différée*. En pratique, ce type de déclaration de constantes se trouve restreint à la partie spécification d'un paquetage. De l'extérieur du paquetage, la constante peut être utilisée, mais sans voir sa valeur qui est définie par une déclaration complète de constante à l'intérieur du corps du paquetage. C'est un exemple de masquage d'information en *VHDL*.

Conversion de types (type conversion)

Il est possible d'effectuer une conversion explicite entre deux types de structures proches. Cette conversion est très restrictive et n'est autorisée que pour les types entiers ou flottants et les tableaux de même dimension ayant des types d'éléments et des index convertibles. Un type peut toujours être converti en lui-même.

Corps de paquetage (package body)

C'est une des cinq unités de conception *VHDL*. Elle se compile séparément. C'est là où sont écrits les algorithmes des sous-programmes exportés dans la spécification du paquetage. On peut y trouver d'autres sous-programmes ou objets d'usage interne à ce paquetage et qui ne sont pas accessibles de l'extérieur.

Cycle de conception VHDL

Le cycle de conception d'une session *VHDL* est : compilation, élaboration, exécution et exploitation. La compilation s'occupe de tous les aspects statiques de la description, l'élaboration de tous ses aspects paramétrables. L'exécution, selon l'application, simule, synthétise, prouve ... et l'exploitation retrouve le concepteur comme interlocuteur.

Déclaration (declaration)

Une déclaration associe un nom ou identificateur à un objet du langage, qu'il soit une variable, un signal, une constante, un type, un sous-programme, un composant, etc... *VHDL* est un langage où, à de très rares exceptions près (les labels par exemple), on doit déclarer toute chose avant de l'utiliser.

Déclaration complète (full declaration)

Lors de leur déclaration, la plupart des objets sont déclarés complètement. Deux exceptions fréquentes sont les déclarations incomplètes de type (souvent pour les structures récursives) et les constantes à valeurs différées (pour le masquage d'information). Dans ces deux cas, les déclarations classiques de ces objets sont obligatoires et seront faites ultérieurement: elles portent le nom de *déclarations complètes*.

Déclaration de constante à valeur différée (deferisico constant declaration)

Voir constante à valeur différée

Déclaration de sous-programme (subprogram declaration)

Aussi appelée spécification de sous-programme. cette déclaration se trouve souvent dans la partie spécification de paquetage. Elle permet de définir toutes les informations utiles à l'appel du sous-programme: son profil.

Déclaration incomplète de type (incomplete type declaration)

En *VHDL*, tout objet utilisé doit être préalablement déclaré. Cette contrainte peut être critique dans le cas de la déclaration de structures de données récursives, elles s'utilisent

elles-mêmes avant d'être complètement définies. La déclaration incomplète de type permet de résoudre ce problème. Le nom du type est donné, mais pas sa structure. Il est alors possible d'utiliser ce type dans une structure de donnée et de ne le définir complètement que plus tard. Cette déclaration tardive mais complète sera exigée par le compilateur.

Délai delta (delta delay)

C'est le délai utilisé par le simulateur pour s'allouer à chaque pas de simulation un nombre variable de tranches infinitésimales de temps et ainsi gérer la succession des affectations de signaux. Il a deux caractéristiques essentielles: il n'est pas nul et le cumul de délais *delta* ne peut jamais atteindre ne serait ce qu'une femtoseconde. Les deux dimensions du temps en *VHDL* sont le temps *réel*, qui se compte en pas de simulation et en unités de type *TIME* (c'est ce temps que voit le concepteur) et le temps *delta*, qui est géré par le simulateur. Le temps *delta* est le reflet de la causalité.

Description flot de données (data-flow description)

C'est une description qui montre les flots de données sortants en fonction des entrants d'un modèle sans se préoccuper de sa structure. Une description *flot de données*, qui n'est qu'un raccourci d'écriture de description comportementale, utilise principalement l'affectation concurrente de signal et l'appel de fonctions pour modéliser.

Description comportementale (behavioural description)

La description de niveau comportemental s'attache à décrire le fonctionnement d'un modèle sans se soucier d'un éventuel découpage proche de la réalisation, donc de la structure. La description prend la forme d'un algorithme. En langage *VHDL*, une description purement comportementale sera un processus au sens général (l'appel d'une procédure concurrente ou la description de niveau flot de données ayant chacun leur traduction en terme de processus) dans lequel se trouvera une algorithmique du genre de celle que l'on utilise dans les langage de programmation classiques. Ce niveau de description, bien que présent dans les *feuilles* de la hiérarchie, peut aussi s'adresser aux concepteurs désirant modéliser un système avec un haut niveau d'abstraction. Le temps peut y intervenir.

Description fonctionnelle (functionnal description)

Voir description comportementale.

Description hybride

C'est une description mélangeant au moins deux des trois genres de descriptions: comportementale, flot de données et structurelle.

Description structurelle (structural description)

C'est le niveau de description le plus simple à appréhender. Il consiste à décrire le modèle par sa structuration, c'est-à-dire par un ensemble d'éléments interconnectés. En langage *VHDL*, une telle description est constituée de déclarations et d'instances de composants. Deux propriétés se déduisent immédiatement: ce niveau de description ne fait pas intervenir le temps et ne peut, en aucun cas, constituer une *feuille* (élément terminal dans

la hiérarchie de description) au moment de la simulation. Une *feuille* fera nécessairement l'objet d'une description comportementale ou *flot de données*.

Echéancier

C'est l'agenda du simulateur. Sur la base des événements de la date courante, il y inscrit des événements aux dates futures.

Effet de bord

Se dit d'une modification de l'environnement par un autre moyen que les paramètres de sortie de procédures, les signaux de mode out des entités ou les valeurs de retour des fonctions. Par exemple, l'affectation d'un signal global (non déclaré localement mais dans un paquetage) est un effet de bord.

Elaboration (elaboration)

L'élaboration est la phase d'initialisation de *VHDL* qui consiste à créer les objets qui seront peut-être modifiés par la suite. Voir Cycle de conception *VHDL*.

Elément d'onde (waveform element)

C'est un couple valeur/délai séparé par le mot clé after. Un élément d'onde apparaît dans la partie droite d'une affectation de signal. La valeur est la valeur prévue du signal à la date correspondant à l'instant présent plus le délai. Séparés par des virgules, les éléments d'onde constituent une forme d'onde.

En-tête de sous-programme (subprogram header)

C'est par cela que commence un sous-programme. Ces informations, très proches de celles de la déclaration de sous–programme constituent son profit.

Encapsuler, Encapsulation

C'est le fait de découper en vue externe et vue interne une notion. En *VHDL*, les bibliothèques, les paquetages, les entités, les processus, les blocs et les sous-programmes sont autant de moyens d'encapsuler l'information. Cette encapsulation est une forme de masquage d'information.

Entité (design entity)

C'est un couple spécification d'entité/ architecture, un modèle. Il peut y avoir plusieurs architectures pour la même spécification d'entité. Lors de la simulation, ce sont des entités que l'on simule (une seule architecture par spécification d'entité).

Evénement sur un signal (event)

Un évènement se produit sur un signal si celui-ci change de valeur. La notion d'évènement est à distinguer de la notion de transaction.

Exécution (execution)

Voir Cycle de conception *VHDL*.

Exploitation

Voir Cycle de conception *VHDL*.

Exporter

Les sous-programmes, types et autres objets figurant dans une spécification de paquetage sont dits *exportés par ce paquetage*. Ils sont visibles et utilisables par le monde extérieur au paquetage.

Expression (expression)

Une expression permet d'indiquer la façon de calculer une valeur. Une constante ou un appel de fonction sont des exemples d'expression.

Expression qualifiée (qualifed expression)

Voir Qualification d'expression.

Fichier (file)

Le type fichier (mot clé file) permet d'avoir accès aux fichiers du système sur lequel est installée la plate-forme *VHDL*. Leur utilisation est courante : lecture du contenu d'une *ROM*, écriture de trace, lecture de stimuli, sortie de résultats...

Flot de données (data flow)

Ensemble des informations transitant sur les ports ou signaux internes d'un modèle.

Fonction (function)

Les fonctions sont un des deux genres de sous-programmes en *VHDL*. Elles rendent un résultat (et un seul) et n'agissent pas par effet de bord.

Fonction de résolution (resolution function)

Quand la valeur d'un signal peut provenir de plusieurs sources (collecteurs ouverts, bus trois-états ...), il faut savoir gérer les conflits. On demandera donc à un signal multi-sources de posséder une fonction de résolution. C'est un mécanisme qu'offre *VHDL* et qui permet de résoudre les conflits susceptibles de survenir et de calculer la valeur résultante (on dit valeur résolue) du signal. Un signal possédant une fonction de résolution est dit un signal résolu et cette fonction de résolution est appelée par le simulateur pour calculer sa valeur.

Format intermédiaire (interchange format)

Le résultat de la compilation est appelé *format intermédiair*. De nombreuses discussions sur l'intérêt de normaliser un format intermédiaire ont abouti à la création du groupe de normalisation *VIFASG*. Il est décidé de définir un jeu de sous-programmes permettant le stockage et la relecture de la sémantique d'un texte *VHDL*. Le format physique sous-jacent n'entre pas dans le cadre de cette normalisation.

Forme d'onde (waveform)

Dans une affectation de signal, les différents champs (les couples valeur/délai séparés par le mot clé after) sont appelés des éléments d'onde. Séparés par des virgules, ils constituent une forme d'onde, c'est-à-dire une suite de valeurs prévues du signal à des dates données.

Généricité (genericity)

Voir Bloc générique.

Glossaire (glossary)

C'est une sorte de dictionnaire qui peut être récursif, la preuve...

Groupe de normalisation (standardization groups)

Voir VASG,VDEG,VIFASG,WAVES.

Hold

Voir Temps de maintien

Identificateur (identifier)

C'est le nom d'un objet. L'association d'un nom à un objet se fait lors de la déclaration de cet objet. Un identificateur est une suite de lettres (jeu de caractères *ASCII*), de chiffres et de traits bas (_) commençant par une lettre et qui n'est pas un mot clé.

Inertiel (inertial delay)

Il s'agit d'un mode d'affectation des signaux en *VHDL*. Il filtre les impulsions dont la durée est inférieure au délai de l'affectation qui est indiqué par le mot clé after. C'est ce mode qui est pris par défaut à chaque affectation de signal. L'autre mode s'appelle transport. La figure 8.4 visualise les effets de ces deux modes.

Initialisation (initialization phase)

Cette phase est la première de l'exécution. Tous les processus sont lancés au moins une fois pendant cette phase.

Instance (instantiation)

Une instance de composant est une occurrence du modèle de ce composant. A partir d'un modèle d'inverseur, on peut créer (instancier) plusieurs inverseurs *réels* (*INV1, INV2 et INV3* par exemple) ayant chacun leurs propres ports mais les caractéristiques de leur modèle père. *INV1, INV2 et INV3* sont les instances (on dit aussi instanciations) du composant inverseur.

Instanciation (instantiation)

Voir Instance.

Instancier (to instantiate)

C'est l'action de créer une instance (ou instanciation).

Instruction concurrente (concurrent statement)

Un système matériel se décrit naturellement de façon concurrente. Certaines parties gèrent des accès disque, d'autres des accès mémoire et beaucoup des accès à des bus. Tous ces fonctionnements se passent *en même temps* pour l'esprit humain qui fait la description, c'est cela la notion de *concurrence*. *VHDL* possède tout un jeu d'instructions concurrentes, le chapitre 9 leur est consacré.

Instruction séquentielle (sequential staternent)

Dans une description de matériel, certaines choses se décrivent naturellement de manière séquentielle. Un protocole de lecture d'une mémoire s'exprime par exemple: positionner tel signal, puis tel autre, attendre tant de temps puis faire.... VHDL dispose de tout un jeu d'instructions séquentielles utilisables dans un processus ou dans le corps d'un sous-programme.

Interopérabilité (interoperability)

Peu français, ce terme désigne la faculté de pouvoir réunir au sein d'une même modélisation, des descriptions de composants fournies par des constructeurs différents.

Itération (iteration)

Une boucle (mot clé loop) sert à répéter la séquence d'instructions qui s'y trouve. Chaque déroulement s'appelle une itération.

Label (label)

Certaines instructions peuvent avoir un nom. Ce nom s'appelle un label. Un label est même obligatoire pour certaines instructions concurrentes. Libération (deallocation) Lorsqu'un objet pointé par un type accès n'est plus utile, il est possible de récupérer la zone mémoire qu'il occupe. Cette opération s'appelle une libération mémoire et se fait par appel à la procédure *DEALLOCATE*, déclarée automatiquement pour tout objet pointé. On parle de libérer un pointeur.

Limite de résolution secondaire (secondary unit as resolution limit)

Tout type physique possède une unité de base. Pourtant, si dans une unité de conception, l'unité physique utilisée est très grande (les minutes pour le type *TIME* par exemple), le codage peut poser un problème (surtout s'il est effectué sur 32 bits). La norme admet alors le codage à partir d'une limite de résolution secondaire: c'est l'unité physique la plus petite utilisée dans cette unité de conception qui sert alors d'unité de base. Cette unité secondaire de résolution peut être à l'origine d'incohérences entre plusieurs unités de conception se partageant (ou s'échangeant) un même type physique.

Liste de sensibilité d'un processus (sensitivity list)

C'est une liste de signaux. Tout événement sur un signal provoque l'activation des processus dans la liste de sensibilité desquels il apparaît.

Littéral (literal)

Les littéraux sont les valeurs *dures* du langage : par exemple 123 est un littéral. Les littéraux sont classés en numériques (123, 3.14), énumérés (*TRUE* en est un), chaînes de caractères (*en voici une*) ou de bits ("101"), et null qui est un littéral à lui tout seul, et qui représente la valeur de tout objet de type accès non initialisé.

LRM

C'est de l'anglais, cela signifie *Language Reference Manual* c'est-à-dire *manuel de référence du langage*. Aussi appelé *La norme*, ce document dûment normalisé par *IEEE* contient (théoriquement) toutes les informations relatives à la syntaxe et à la sémantique de VHDL. Destiné avant tout aux constructeurs de compilateurs, ce n'est pas un *manuel utilisateur* et sa lecture est très indigeste pour un néophyte. Cet ouvrage reste néanmoins une référence pour tous les problèmes pointus pouvant survenir lors de modélisations.

Manuel de référence (language reference manual)

Voir *LRM*.

Masquage d'information

C'est un vieux principe de truand qui veut que *moins on en sait, mieux on se porte*. Ce principe est érigé en dogme en *VHDL*. Il consiste à ne donner à voir au concepteur que ce dont il a besoin, en lui en cachant la réalisation. Ainsi, le concepteur n'est ni submergé de détails inutiles, ni soumis à la tentation de modifier ce qui a été fait (et testé) par quelqu'un d'autre, ni à celle d'utiliser les dites informations pour se livrer à du *reverse-engineering*.

Mode d'affectation des signaux

Il existe deux modes d'affectation de signaux différents, le mode inertiel (par défaut) et le mode transport.

Mode inertiel (inertial delay)

Voir Inertiel.

Mode transport (transport delay)

Voir transport.

Modèle (model)

Dans cet ouvrage, modèle désigne un couple spécification d'entité/architecture: une entité. La spécification d'entité est la vue externe du modèle (ce que l'on voit de l'extérieur) et l'architecture est la vue interne (comment marche-t-il ?).

Modèle inertiel (inertial delay)

Voir Inertiel.

Modèle transport (transport delay)

Voir transport.

Mot clé (key word)

Ce sont des mots réservés du langage. On ne peut les utiliser comme identificateurs. Il y en a 82 et il vaut mieux les connaître. Signe particulier: ils sont tous américains.

Niveau de sévérité (severity level)

Le niveau de sévérité de l'erreur d'une instruction d'assertion est une expression du type *SEVERITY-LEVEL*. Ce type est défini dans le paquetage *STANDARD* (détaillé au chapitre 12), et donc connu par défaut. C'est un type énuméré défini par: *type SEVERITY-LEVEL is (NOTE, WARNING, ERROR, FAURE)*;

Niveaux de Description

Voir descriptions structurelle, comportementale, ftot de données et hybride.

Norme VHDL 7.2 (VHDL version 7.2)

Voir *VHDL* 7.2.

Notation basée (based literal)

C'est une notation qui permet d'exprimer les littéraux dans une base autre que la base dix, prise par défaut. Les bases possibles vont de deux à seize.

Notation pointée (selected name)

S'il est visible, un objet se dénote soit directement par son nom, soit plus laborieusement sous forme d'une notation pointée qui traduit le *chemin* pour l'atteindre.

Objet (object)

D'une manière générale, tout ce qui a un nom en *VHDL* est un objet. Cela inclut tout ce qui explicitement se déclare, plus les labels. Il arrive pourtant de restreindre ce terme aux trois classes d'objets: constantes, variables, signaux.

Objet pointé (object designated by the access value)

C'est un objet qui est créé dynamiquement (à l'exécution) lors de l'allocation. Il disparaît par une opération de libération mémoire.

Opérateur (operator)

Il existe six classes d'opérateurs *VHDL* et un niveau de priorité unique est attribué à chaque classe. Par ordre de priorité croissante, on trouve: les opérateurs logiques, les opérateurs relationnels, les opérateurs d'addition, les opérateurs de signe, les opérateurs de multiplication et les autres. Comme les fonctions, les opérateurs peuvent faire l'objet de surcharges.

Paquetage (package)

C'est le moyen d'encapsuler des objets et des sous-programmes. Il est constitué d'un couple spécification de paquetage/corps de paquetage. Le corps de paquetage est optionnel mais s'il existe, il est unique.

Paramètre d'entrée de sous-programme (parameter of mode in)

C'est un objet transmis à un sous-programme qui ne doit qu'être consulté (lu) et ne peut être modifié (écrit). Le mode in permet de préciser cet état de fait. Le compilateur vérifiera les restrictions d'emploi qui en découlent.

Paramètre de sortie de procédure (parameter of mode out)

C'est une valeur rendue par une procédure. Le mot clé out permet de spécifier ce mode de passage qui n'est qu'un retour.

Paramètre effectif d'un sous-programme (actual parameter of a subprogram)

C'est le paramètre qui est associé au paramètre formel d'un sous-programme lors de l'appel. L'association se fait par position ou par nom du paramètre formel.

Paramètre formel de sous-programme (formal parameter of a subprogram)

C'est le paramètre tel qu'il est donné dans la spécification du sous-programme. C'est ce nom qui sera utilisé lors de l'appel du sous-programme en association par nom.

Paramètre générique effectif (actual generic parameter)

C'est le paramètre avec lequel sera instanciée l'unité de conception générique. L'association du paramètre générique effectif au paramètre générique formel se fait lors de l'instance.

Paramètre générique formel (format generic)

C'est avec ce paramètre que s'écrit l'unité de conception générique. Lors de l'instance de cette description, ce paramètre sera remplacé par la valeur du paramètre générique effectif.

Pas de simulation (step of the simulation cycle)

C'est un instant du temps vrai. Il correspond à une date unique mais peut être divisé en autant de *délais - delta* que nécessaire par le simulateur pour effectuer la succession des instructions concurrentes de cet instant. Phase d'initialisation (initialization phase) voir Initialisation.

Pilote d'un signal (driver)

C'est une caractéristique qui est associée à chaque signal et qui contient sa valeur courante et la liste des valeurs prévues et leurs dates. Un signal n'a qu'un pilote par processus où il apparaît, mais il peut apparaître dans plusieurs processus. Dans ce cas, ce doit

être un signal résolu.

Plan mémoire

Une *ROM* ou une *RAM* sont des systèmes comprenant des blocs de contrôle et un bloc où est mémorisé l'information. C'est ce bloc qui est appelé plan mémoire.

Plate-forme VHDL

C'est l'ensemble de l'environnement de développement et de test des modélisations VHDL. Cela comprend généralement un simulateur mais peut aussi s'étendre à un éditeur *langage* textuel, à des éditeurs graphiques (pour entrée de descriptions structurelles) et à tout autre outil du monde *VHDL*.

Pointeur (access type)

Voir 71jpe accès.

Polymorphisme

C'est le moyen (en informatique) de présenter la même information de différentes manières. Association de sous-éléments et conversion de paramètres sont deux façons de pratiquer le polymorphisme en VHDL . Ce ne sont pas les seules, les alias par exemple en constituent une autre forme.

Port (port)

Les ports sont les points d'entrées/ sorties des entités ou des composants. Un port est caractérisé par le sens et le type des données qui y transitent.

Port effectif (actual port)

C'est un port *réel* associé à un port formel.

Port formel (formal port)

C'est le nom du port tel qu'il apparaît dans une spécification d'entité. C'est ce nom qui sera utilisé dans le cas d'une association par nom lors de la configuration.

Procédure (procedure)

La procédure est un des deux genres de sous-programmes VHDL . Elle peut agir par effet de bord. Elle modifie aussi (éventuellement) la valeur des paramètres transmis à l'appel.

Processus (process)

C'est la base de la concurrence en VHDL . Toute instruction concurrente peut être traduite par un processus, c'est le processus équivalent. Un processus ne contient que des instructions séquentielles. Un processus vit toujours arrivé *à* sa fin (mots clés *end* processus), il s'exécute *à* nouveau depuis son mot clé de début begin. Il est possible, par la spécification d'une liste de sensibilité (des signaux) ou par une instruction wait on, de

l'endormir en attendant des événements sur les signaux indiqués.

Processus passif (passive process)

Sont appelés passifs, les processus où aucun signal n'apparaît à gauche d'une affectation. Ces processus sont dits passifs, non parce qu'ils ne s'exécutent pas, mais parce que leur exécution ne va pas entraîner l'exécution d'autres processus. L'instruction concurrente d'assertion et, sous certaines conditions (pas de paramètres de mode out ni inout), l'appel concurrent de procédures ont des processus équivalents passifs.

Profil (parameter and result type profile)

Le profil d'un sous-programme est un ensemble d'informations qui comprend le nombre, l'ordre et le type des paramètres formels ainsi que, pour une fonction, le type du résultat rendu. Le fait de pouvoir surcharger deux sous-programmes de profils différents est particulièrement pratique et accroît la facilité d'écriture tout en assurant une bonne lisibilité des programmes.

Processus passif (passive process)

Sont appelés passifs, les processus où aucun signal n'apparaît à gauche d'une affectation. Ces processus sont dits passifs, non parce qu'ils ne s'exécutent pas, mais parce que leur exécution ne va pas entraîner l'exécution d'autres processus. L'instruction concurrente d'assertion et, sous certaines conditions (pas de paramètres de mode out ni inout), l'appel concurrent de procédures ont des processus équivalents passifs.

Profil (parameter and result type profile)

Le profil d'un sous-programme est un ensemble d'informations qui comprend le nombre, l'ordre et le type des paramètres formels ainsi que, pour une fonction, le type du résultat rendu. Le fait de pouvoir surcharger deux sous-programmes de profils différents est particulièrement pratique et accroît la facilité d'écriture tout en assurant une bonne lisibilité des programmes.

Qualification d'expression (qualified expression)

La qualification d'expression permet de lever une ambiguïté sur le type d'une expression ou d'un agrégat en indiquant explicitement le nom de ce type.

Rationale

C'est un document édité par le *VASG*. Il explique en détail les raisons des choix qui ont conduit à la norme. Sa lecture (pour personne avertie uniquement) peut être très instructive.

Récursivité (recursivity)

La récursivité est la propriété qu'a un sous-programme de pouvoir s'appeler lui-même. Une structure de donnée s'appelant elle–même est aussi dite *récursive*. La récursivité croisée (le sous–programme A appelle le sous-programme B et réciproquement) est également autorisée en $\mathcal{V}$HDL .

Repository

C'est un ensemble de programmes et de descriptions $\mathcal{V}$HDL mis dans le domaine public. Gérée par le *VDEG*, cette banque d'informations permet de faciliter les échanges entre concepteurs $\mathcal{V}$HDL .

Reverse-engineering

Procédé d'espionnage industriel tellement laid qu'il n'a pas de traduction en français.

Schéma d'itération (iteration scheme)

Le nombre de fois où sera répétée la séquence d'instructions contenue dans une boucle (mot clé loop) est régi par le schéma d'itération. Il existe deux schémas d'itération distincts correspondant aux mots clés while et for.

Sémantique

C'est le sens, la signification d'une notion. La sémantique complète la syntaxe qui n'est, elle, que la forme, la notation.

Set-up

Voir Temps de pré-positionnement.

Signal (signal)

Les signaux sont spécifiques des langages de description de matériel. Ils modélisent les informations qui passent sur les fils, les bus, ou d'une manière générale qui transitent entre les composants d'une description de matériel. Chacun d'eux possède un pilote (par processus où il apparaît) et est sujet à des transactions et des événements. Les signaux constituent une des trois classes d'objets $\mathcal{V}$HDL .

Signal accessible en lecture (readable signal)

Voir Accessible en lecture.

Signal gardé (guarded signal)

C'est un signal qui, lors de sa déclaration, a été classé comme registre (mot clé *register*) ou bus (mot clé *bus*). Il ne peut être affecté que par une affectation gardée (mot clé *guarded*). Cette classification (*registre* ou *bus*) sert à décrire son comportement lors de sa déconnexion.

Signal interne (local signal)

C'est un signal déclaré à l'intérieur d'un bloc ou d'une architecture et qui n'est pas visible de l'extérieur de cette structure.

Signal multi-sources (multiple source signal)

C'est un signal qui est la cible de plusieurs affectations concurrentes de signaux. Un tel signal doit être un signal résolu.

Signal résolu (resolved signal)

Un signal possédant une fonction de résolution dans sa déclaration ou ayant un type en possédant une est dit résolu. Cela signifie qu'il peut avoir plusieurs sources, c'est-à-dire être la cible de plusieurs affectations concurrentes différentes. La fonction de résolution est là pour résoudre les conflits. Durant la simulation, cette fonction est appelée systématiquement pour calculer la valeur que va prendre le signal.

Signal statique (static signal name)

Un signal est statique si son nom est connu à la compilation de l'unité de conception dans laquelle il se trouve. Par exemple, le signal *TAB(1)*, où *TAB* est un tableau et 1 un paramètre de procédure n'est pas statique. Les signaux faisant partie de la liste de sensibilité d'un processus doivent être statiques.

Simulation (simulation)

Ce n'est qu'une des formes que peut prendre l'exécution, mais c'est la plus courante. Voir Cycle de conception $\mathcal{V}$HDL .

Source d'un signal (source of a signal)

C'est tous les signaux apparaissant dans une partie droite d'une affectation concurrente de signal.

Sous-programme (subprogram)

Les sous-programmes permettent d'écrire du code réutilisable. Les valeurs des paramètres peuvent varier à chaque appel et donc donner des exécutions différentes. En $\mathcal{V}$HDL , les sous-programmes sont principalement utilisés pour décrire une suite d'instructions séquentielles, un calcul, une conversion, des morceaux de processus ou des fonctions de résolution. Il existe deux genres de sous–programmes: les procédures (mot clé *procedure*) et les fonctions (mot clé *function*).

Sous-type (subtype)

C'est une restriction sur les valeurs d'un type ou d'un autre sous-type. Ces restrictions (ou contraintes) peuvent être dynamiques et donc calculées à l'exécution. Sans contrainte, le sous-typage peut servir à renommer un objet.

Sous-type dynamique

C'est un sous-type dont les contraintes sont calculées à l'exécution. C'est une des plus précieuses utilisations du sous-typage.

Spécification (specification)

Une spécification permet de donner des informations complémentaires sur un objet du langage.

Spécification d'entité (entity declaration)

C'est la vue externe d'une entité. On y trouve principalement la description des ports formels et des paramètres formels de généricité de ce modèle. Associée à une architecture, la spécification d'entité forme un modèle (aussi appelé entité). Il peut y avoir plusieurs architectures pour la même spécification d'entité, mais lors de l'exécution (la simulation), un seul couple spécification d'entité/architecture sera sélectionné pour chaque composant. La spécification d'entité est une des cinq unités de conception $\mathcal{V}$HDL , elle se compile séparément.

Spécification de paquetage (package declaration)

La spécification de paquetage est une des cinq unités de conception $\mathcal{V}$HDL , elle se compile séparément. C'est la vue externe d'un paquetage. On y déclare les objets et les sous-programmes que le paquetage exporte. L'algorithme réalisant ces sous-programmes est, lui, caché dans le corps du paquetage.

Spécification de sous-programme (subprogram declaration)

Voir Déclaration de sous-programme.

Stimulus, stimuli

C'est du latin, d'où le *i* au pluriel. En $\mathcal{V}$HDL , un stimulus désigne un couple (valeur, date). Un stimulus est très proche d'un élément d'onde.

Surcharge (overloading)

Deux sous-programmes sont dits surchargés s'ils ont le même nom et que leurs profils diffèrent. Le fait de pouvoir surcharger deux sous-programmes de profils différents est particulièrement pratique et augmente la facilité d'écriture et la lisibilité d'un programme. On parle aussi de surcharge de symboles de types énumérés pour indiquer que plusieurs types énumérés peuvent se partager le même symbole.

Systèmes matériels (hardware)

Ce sont indifféremment des cartes ou des circuits intégrés en électronique mais ce terme ne se restreint pas à ce domaine. Un carburateur de voiture ou un neurone, par exemple, sont des systèmes matériels et $\mathcal{V}$HDL permet de les décrire.

Tableau (array)

C'est une famille de types qui a la particularité de réunir des éléments de type identique. On accède à ces éléments par un (ou plusieurs) indice. Alliés aux articles, les tableaux constituent la famille des types composites. Un tableau à une seule dimension (un seul indice) est appelé vecteur.

Tableau contraint (constrained array)

C'est un tableau dont l'intervalle de variation des indices (et donc la taille) est connu dès l'initialisation (à l'élaboration). Les mots clés *to* et *downto* permettent de préciser le

sens de variation des indices.

Tableau non contraint (unconstrained array)

C'est un tableau où le symbole <> (que l'on prononcera box) en indice permet de repousser la définition d'un intervalle d'indiçage et d'une direction de variation à plus tard, lors de l'exécution. Deux exemples de tableaux non contraints sont les vecteurs de type *BIT* (type *BIT-VECTOR*) et les chaînes de caractères (type *STRING*).

Temps de hold

Voir Temps de maintien.

Temps de set-up

Voir Temps de pré-positionnement.

Temps de maintien d'un signal (hold)

Une exigence très courante en électronique est relative à la durée minimale de stabilité d'une donnée après un front. Ce temps minimal s'appelle temps de maintien.

Temps de pré-positionnement d'un signal (set-up)

C'est le délai de pré-positionnement d'un signal par rapport à un autre. Une vérification de pré-positionnement est tout à fait classique pour les éléments de mémorisation. Une donnée DIN devant être mémorisée sur le front montant d'un signal *CLK* se doit d'être stable pendant un certain délai de temps minimum.

Transaction (transaction)

Un transaction est créée à chaque fois que l'on calcule la valeur d'un signal, même si cette valeur calculée est la même que la valeur courante du signal (contrairement à l'événement).

Transport (transport)

Il s'agit d'un mode d'affectation des signaux en VHDL. Ce mode décrit un comportement à réponse fréquentielle infinie de l'affectation de signal. Ce mode transmet intégralement les impulsions, quelles que soient leurs durées. Il est indiqué par le mot clé transport après le symbole d'affectation. L'autre mode s'appelle inertiel et il est pris par défaut à chaque affectation de signal.

Type (type)

$\mathcal{V}$HDL est un langage typé: tout objet du langage doit posséder un type avant d'être utilisé. Un type définit implicitement ou non, (type énuméré) l'ensemble des valeurs que peut prendre un objet, et l'ensemble des opérations disponibles sur cet objet. Ce type permet à l'analyseur/compilateur $\mathcal{V}$HDL de faire un grand nombre de vérifications de cohérence.

Type accès (access type)

Un type accès (mot clé *access en VHDL*) est souvent aussi appelé pointeur. Un type accès pointe sur un objet de type précédemment défini. Il permet principalement de créer dynamiquement (c'est-à–dire, lors de l'exécution) des objets nouveaux. Des opérations d'allocation et de libération mémoire sont associées à tout type accès.

Type compatible

Deux types sont compatibles si tout objet appartenant à l'un d'eux peut être mis à la place d'un objet appartenant à l'autre sans erreur de compilation. Néanmoins, des vérifications dynamiques peuvent conduire à une erreur au moment de l'exécution s'il se trouve que la valeur effectivement prise par un objet n'est pas dans l'ensemble des valeurs autorisées pour le type attendu.

Type composite (composite type)

C'est une famille de types qui réunit les types dont la structure est composée d'éléments: les articles et les tableaux.

Type énuméré (enumeration type)

C'est un type dont les valeurs sont des symboles. Par exemple, *BOOLEAN* est un type énuméré. La valeur d'un objet de ce type sera à prendre dans l'ensemble des symboles qui lui sont associés: *FALSE* ou *TRUE* en l'occurrence. Plusieurs types énumérés peuvent se partager les mêmes symboles.

Type fichier (file type)

Voir Fichier.

Type physique (physical type)

Il existe en VHDL la notion d'unité de quantité. On définit par exemple dans le paquetage *STANDARD*, le type *TIME* qui est un type physique. C'est ici la notion de tempsvrai que connaît le simulateur et qui sera utile dans bon nombre d'instructions. C'est le seul type physique prédéfini. Un type physique est caractérisé par son unité de base (c'est la femtoseconde pour le type *TIME*), l'intervalle de ses valeurs autorisées et une éventuelle collection de sous-unités ainsi que leur correspondance entre elles.

Type scalaire (scalar type)

Un type scalaire est un type qui ne possède pas de sous–éléments. Les entiers, les flottants, les types physiques et les types énumérés sont des types scalaires. Les tableaux ou les enregistrements ne font pas partie des types scalaires, ils sont appelés types composites.

Unité de base (base unit)

C'est la plus petite unité d'un type physique. Toutes les autres unités s'expriment comme multiples (entiers) de cette unité.

Unité de conception (design unit)

C'est l'élément de base des bibliothèques et l'unité de compilation VHDL . Il en existe de cinq sortes : les spécifications de paquetages, les corps de paquetages, les spécifications d'entités, les architectures et les configurations.

Unité de conception générique

Voir Bloc générique.

Valeur courante d'un signal (current value)

Contrairement à l'affectation de variables, l'affectation de signaux ne change pas la valeur courante du signal, mais modifie les valeurs prévues (contenues dans son pilote) que sera susceptible de prendre ce signal. Lorsqu'on affecte un signal à une variable, c'est la valeur courante de ce signal qui est prise par la variable.

Valeur résolue d'un signal (resolved value)

C'est la valeur d'un signal résolu. Ce signal multi-sources est doté d'une fonction de résolution qui est appelée par le simulateur pour calculer la valeur du signal résultant des valeurs des différentes sources. Le résultat de cette fonction de résolution est la valeur résolue du signal.

Valeurs prévues d'un signal (projected output waveforms)

Ce sont des couples valeur/date. Sauf remise en cause par d'autres instructions, elles représentent les valeurs du signal aux dates spécifiées. Cet *agenda du signal*, associé à sa valeur courante constitue le pilote du signal.

Variable (variable)

Les variables sont une des classes d'objets VHDL (les deux autres sont les constantes et les signaux). Leurs valeurs peuvent être modifiées tout au long de la simulation. Contrairement aux signaux, on peut modifier (par affectation par exemple), la valeur courante d'une variable. C'est un objet lié à l'algorithmique et à la séquentialité. Son domaine d'utilisation est restreint aux sous-programmes et aux processus.

Variable accessible en lecture

Voir accessible en lecture.

VASG (VHDL Analysis and Group)

C'est le plus important des groupes de normalisation VHDL . Il a créé la norme VHDL . Son travail de déverminage de cette norme est achevé et l'essentiel de ses activités se recentre sur la définition de la nouvelle norme des années 92.

VDEG (VHDL Design Exchange Group)

Ce groupe de normalisation définit des sous-ensembles de VHDL et des paquetages standard. Des discussions sont en cours pour définir les meilleures façons d'intégrer le

temps dans une description, ou pour définir des bibliothèques d'usage courant, voire des paquetages d'utilitaires. Le *repository* est maintenu par ce groupe.

Vecteur (one-dimensional array)

C'est un tableau dont l'indice a une seule dimension. Les chaînes de caractères (type *STRING*) et les vecteurs de bits (type *BIT-VECTOR*) sont parmi les vecteurs les plus utilisés.

VHDL 7.2 (VHDL version 7.2: baseline language)

C'est la norme génératrice du standard *VHDL 1076-1987 IEEE*. Son existence dura six mois et laisse encore quelques traces dans la définition de certaines constructions.

VIFASG (VHDL Intermediate Format and Analysis Standardization Group)

Ce groupe de normalisation VHDL travaille à la standardisation d'un format intermédiaire. Ce format se présentera sous la forme d'un modèle de données et d'un jeu de sous–programmes permettant le stockage et la relecture de la sémantique d'un texte VHDL . Brancher un outil *X* sur une plate-forme *Y*, sera alors possible par une simple édition de liens.

Visibilité (visibility)

C'est la portion de code VHDL où une déclaration est visible, c'est-à-dire dénotable par une notation pointé ou directement par son nom.

REFERENCES

[1] D. Green, "Moderm Logic," Addison-Wesley Publishing Company, Wokungham, England, ISBN :0-201-14541-3, 1986.

[2] F. J. Hill & G. R. Peterson, "Introduction to Switching Theory and Logic Design," Third Edition, Joh Wiley & Sons, New York, 1981.

[3] L. S. Levy, "Discrete Structure of Computer Science," John Wiley & Sons, ISBN : 0-471-03208-5, New York, 1980.

[4] M. Carvallo, "Monographie des Treillis et Algèbre de Boole," Gauthier-Villars, Paris, 1962.

[5] M. Carvallo, "Principes et Applications de l'Analyse Booléenne," Gauthier-Villars, Paris, 1965.

[6] R. E. Prather, "Discrete Mathematical Structures for Computer Science," Houghton Mifflin Company, ISBN :0-395-20622-7, Boston, 1976.

[7] S. C. Lee, "Digital Circuits and Logic Design," Prentice-Hall, ISBN: 0-13-212225-1, Englewood Cliffs, New Jersey, 1976.

[8] Z. Kohavi, "Switching and Automata Theory," Second Edition, Tata McGraw-Hill Publishing Co. LTD., New Delhi, 1978.

[9] W. I. Fletcher, "An Engineering Approach to Digital Design," Prentice-Hall, ISBN: 0-13-277699-5, Englewood Cliffs, New Jersey, 1980.

[10] ALTERA®, "Datasheet : FLEX 10*K* - Embedded Logic Device Family," [Online] Available at http://www.altera.com/literature/ds/dsf10k.pdf (Verified 16 Avril, 2003).

[11] ALTERA®, "Datasheet : MAX 7000 - Programmable Logic Device Family," [Online] Available at http://www.altera.com/literature/ds/m7000.pdf (Verified 16 Avril, 2003).

[12] CYPRESS®, "Applications Handbook," San Jose, CA 95134, Cypress Semiconductor, 1993.

[13] D .D. Gajski & R. Kuhn, "Guset Editor's Introduction : New VLSI Tools," IEEE Computer, Vol 16, n:12, pp : 11-14, Décembre 1983.

[14] D. E. Thomas , E. D. Lagnese, R. A. Walker, J. A. Nestor, J. V. Rajan & R. L. Blackburn, "Algorithmic and Register Transfer Level Synthesis : The System Architect's Workbench," Kluwer Academic Publishers, Boston 1990.

[15] D. P. Story, "The Acrotex Education Bundle." [Online] Available at http://www.math.uakron.edu/~dpstory/acrotex.html (Verified 30 Nov.2002).

[16] F. J. Hill; & G. R. Peterson, "Introduction to Switching Theory and Logical Design," John Wiley & Sons, New York, 1981.

[17] J. F. Wakerly, "Micro-computer Architecture & Programming," John Wiley & Sons, Singapore, 1981.

[18] K. Skahill, "VHDL for Programmable Logic," San Jose, CA 95134, Cypress Semiconductor, 1995.

[19] M. M. Mano & C. R. Kime , "Logic and Computer Design Fundamentals," ISBN 0-13-031486-2, Editeur Prentice Hall, Second Edition, 2001.

[20] R. Airiau, J.-M. Bergé, V. Olive & J. Rouillard, "VHDL du Langage A La Modélisation,"Presses Polytechniques et Universitaires Romandes, Lausanne, Suisse, ISBN : 2-88074-191-2, 1990.

[21] R. Fagot, "BenchMark du Logiciel $Max+Plus_{®}$ *II* d'$Altera_{®}$," Rapport Stage, EISTI, 2001.

[22] Z. Navabi, "VHDL - Analysis and Modeling of Digital Systems," McGraw-Hill, Singapore, ISBN : 0-07-112732-1, 1993.

[23] Z. Kohavi, "Switching and Finite Automata Theory," Tata McGraw Hill Publishing Co. LTD., 2^{nd} Édition, New Delhi, 1978.