

## Initiation au langage VHDL

Auteur : Hervé Le Provost  
[herve.le-provost@cea.fr](mailto:herve.le-provost@cea.fr)

**Résumé** Ce document décrit les éléments de base utiles à la compréhension du langage VHDL et à l'écriture de programmes. Il est agrémenté d'exemples pour illustrer les constructions VHDL les plus utilisées.

**Mots clés** VHDL, RTL

### Liste des Abréviations

ASIC	application specific integrated circuit
DoD	Department Of Defense
FPGA	Field Programmable Gate Array
IEEE	Institute of Electrical and Electronic Engineers
RTL	Register Transfer Level
SIWG	VHDL Synthesis Interoperability Working Group
VASG	VHDL Analysis and Standardization Group
VHSIC	Very High Speed Integrated Circuit
VHDL	VHSIC Hardware Description Language
VITAL	VHDL Initiative Towards ASIC Libraries

	<b>Université de Marne-La-Vallée</b>		
	Initiation au langage VHDL	Date: 13 juin 2008	Page : 2/57

## - Table des Matières -

<b>1.</b>	<b>INTRODUCTION.....</b>	<b>3</b>
1.1.	OBJET DU DOCUMENT.....	3
1.2.	POURQUOI LE LANGAGE VHDL.....	3
1.2.1.	<i>Naissance.....</i>	3
1.2.2.	<i>Evolution.....</i>	3
1.2.3.	<i>Succès.....</i>	4
<b>2.</b>	<b>CONCEVOIR UN SYSTEME A PARTIR D'UNE DESCRIPTION VHDL.....</b>	<b>4</b>
2.1.	LE DÉVELOPPEMENT EN VHDL.....	4
2.1.1.	<i>Banc de test VHDL.....</i>	4
2.1.2.	<i>Le flot de développement VHDL.....</i>	5
2.2.	PARTICULARITÉS DU LANGAGE.....	7
2.2.1.	<i>Signaux, process, instructions concurrentes et simulation.....</i>	7
2.2.2.	<i>Inférence d'éléments mémoires.....</i>	8
<b>3.</b>	<b>LES ÉLÉMENTS DE BASE.....</b>	<b>10</b>
3.1.	LES ENTITÉS DE CONCEPTION ET LA CONFIGURATION.....	10
3.1.1.	<i>La déclaration d'entité.....</i>	10
3.1.2.	<i>Les architectures.....</i>	12
3.1.3.	<i>La configuration.....</i>	14
3.2.	LES TYPES DE DONNÉES.....	21
3.2.1.	<i>Type scalaire.....</i>	21
3.2.2.	<i>Type composite.....</i>	23
3.2.3.	<i>Type fichier.....</i>	25
3.3.	LES CLASSES D'OBJET.....	26
3.3.1.	<i>Les signaux.....</i>	26
3.3.2.	<i>Les variables.....</i>	27
3.3.3.	<i>Les constantes.....</i>	27
3.3.4.	<i>Les fichiers.....</i>	27
3.4.	LES SOUS-PROGRAMMES.....	27
3.4.1.	<i>Les procédures.....</i>	27
3.4.2.	<i>Les fonctions.....</i>	29
3.5.	LES ÉLÉMENTS D'ARCHIVAGE.....	32
3.5.1.	<i>Les librairies.....</i>	32
3.5.2.	<i>Les paquetages.....</i>	32
3.6.	ÉLÉMENTS LEXICAUX.....	32
3.6.1.	<i>Éléments basiques.....</i>	32
3.6.2.	<i>Mots réservés.....</i>	34
<b>4.</b>	<b>LA PROGRAMMATION VHDL.....</b>	<b>35</b>
4.1.	LES INSTRUCTIONS.....	35
4.1.1.	<i>Instructions séquentielles.....</i>	35
4.1.2.	<i>Instructions concurrentes.....</i>	41
4.2.	LES OPÉRATEURS.....	42
4.3.	LES DESCRIPTIONS.....	42
4.3.1.	<i>Descriptions comportementales.....</i>	42
4.3.2.	<i>Descriptions "flots de données".....</i>	43
4.3.3.	<i>Descriptions structurelles.....</i>	43
4.3.4.	<i>Les machines d'état.....</i>	45

	<b>Université de Marne-La-Vallée</b>		
	Initiation au langage VHDL		
		Date: 13 juin 2008	Page : 3/57

## - Liste des Figures –

Figure 1 – <i>Banc de test VHDL</i> .....	4
Figure 2 – <i>Flot de développement VHDL</i> .....	7
Figure 3 – <i>Cycle de simulation</i> .....	8
Figure 4 – <i>inférence d'un élément mémoire</i> .....	9
Figure 5 – <i>Inférence d'une bascule D</i> .....	10
Figure 6 – Architecture structurelle de l'entité regDecal.....	17

## - Liste des Tableaux –

Tableau 1 – Mots réservés

34

## 1. INTRODUCTION

### 1.1. Objet du document

L'objet de ce document est de présenter les principes de base sur lesquels repose le développement d'un système à l'aide d'une description VHDL. Les éléments fondamentaux du langage VHDL ainsi que les différentes méthodes de programmation y sont décrits et illustrés par des exemples. Le contenu est non exhaustif, le document de référence sur lequel s'appuie ce document est la norme VHDL IEEE [1].

### 1.2. Pourquoi le langage VHDL

Le langage VHDL est utilisé pour décrire des systèmes logiques synchrones ou asynchrones.

#### 1.2.1. Naissance

En 1981, le Département de la Défense (DoD) des Etats-Unis d'Amérique a initié puis dirigé le projet "Very High Speed Integrated Circuit" (VHSIC). Ce projet avait pour but de formaliser la description des circuits intégrés développés pour le DoD dans un langage commun. L'intérêt premier était de définir, au travers du langage, une spécification complète et non ambiguë du circuit à développer indépendante de la technologie employée et des outils de CAO.

#### 1.2.2. Evolution

Le Développement du langage a été confié par le DoD aux sociétés IBM, Intermetrics et Texas Instruments. Ce langage est baptisé VHDL (VHSIC Hardware Description Language). En 1987, il fait l'objet d'une normalisation par l'IEEE (Institute of Electrical and Electronic Engineers). Le groupe [VASG](#) (VHDL Analysis and Standardization Group) est un groupe de travail IEEE responsable du Manuel de référence du langage VHDL. La norme VHDL IEEE 1076 comme toute norme IEEE doit être révisée et affinée au moins une fois tous les 5 ans. Ses révisions successives sont **P1076-87**, **P1076-93**, **P1076-2000**, **P1076-2002**, les derniers digits indiquent l'année de révision de la norme. La révision **P1076-2006** est en cours de normalisation. Des groupes de travail distincts du VASG, sont chargés de normaliser des extensions, des restrictions du langage VHDL pour des besoins spécifiques. Les groupes actifs sont :

- **P1076.1** Extension de la norme VHDL aux signaux Analogiques et mixtes ([VHDL-AMS](#))
- **P1076.4-2000** Norme VITAL de spécification de la modélisation d'ASIC (Application

	Université de Marne-La-Vallée		
	Initiation au langage VHDL		
		Date: 13 juin 2008	Page : 4/57

Specific Integrated Circuit) ([VITAL](#))

- **P1076.6-2004** Norme pour la synthèse VHDL RTL (Register Transfer Level) ([SIWG](#))

Les groupes dont l'activité a cessé sont :

- **P1076.2** Normalisation IEEE de packages mathématiques VHDL
- **P1076.3** Normalisation IEEE de packages de synthèse VHDL
- **P1076.5** Groupe de travail sur des bibliothèques d'utilitaires VHDL
- **P1164** Normalisation de la logique à plusieurs valeurs pour l'interopérabilité des modèles VHDL (Std\_logic\_1164)

### 1.2.3. Succès

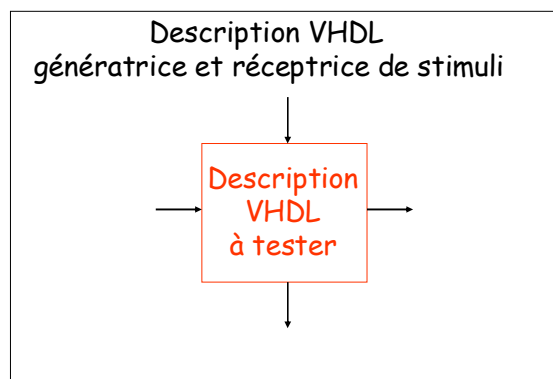
La vivacité de ces groupes de travail constitués de concepteurs en électronique et de développeurs de logiciels de CAO a assuré le succès de ce langage. Aujourd'hui, il est proposé par tous les vendeurs de composants logiques programmables pour décrire la logique. Il a remplacé la multitude de langages propriétaires HDL (citons MINC, ABEL, PALASM), et ainsi facilité la réutilisation et l'échange entre concepteurs de descriptions logiques éprouvées.

## 2. CONCEVOIR UN SYSTEME A PARTIR D'UNE DESCRIPTION VHDL

### 2.1. Le développement en VHDL

#### 2.1.1. Banc de test VHDL

Contrairement aux descripteurs HDL auxquels il a succédé, les capacités du langage VHDL ne se limitent pas à la description de la logique qui sera implémentée dans un circuit. La norme n'est pas limitative et intègre les instructions séquentielles, la déclaration typée de données, l'appel de fonctions et procédures, la gestion de bibliothèques, l'accès à des fichiers. Il est alors possible de modéliser un objet matériel et son banc de test dans le même langage VHDL (Figure 1). La fonction du banc de test est de se connecter sur les entrées, sorties et entrées/sorties de l'objet à tester, exciter l'objet par des stimuli et vérifier sa réponse.



**Figure 1 – Banc de test VHDL**

Des bibliothèques de modèles VHDL aident à la modélisation et aux tests de systèmes complexes tels que les systèmes processeurs. De nombreux modèles sont développés et mis à disposition par la communauté VHDL ou par les vendeurs de circuits intégrés.

	<b>Université de Marne-La-Vallée</b>		
	Initiation au langage VHDL		
		Date: 13 juin 2008	Page : 5/57

### 2.1.2. Le flot de développement VHDL

#### *Synthèse et placement/routage*

La synthèse logique est le processus par lequel une description logique HDL est transformée en une liste de signaux interconnectant les primitives (portes logiques, registres, blocs mémoire, multiplieurs...) de la bibliothèque du circuit ciblé. Le placeur/routeur place et connecte physiquement ces primitives. S'il existe des outils de synthèse indépendants des fabricants de circuits intégrés, le placeur/routeur est toujours la propriété du fabricant. La norme **P1076.6-2004** définit quelles descriptions VHDL sont synthétisables, elles sont nommées VHDL RTL. Cette norme vise à garantir la portabilité d'une conception VHDL d'un outil de synthèse à un autre. Cependant, des synthétiseurs ne supportent pas toutes les descriptions RTL et l'instanciation directe de primitives propriétaires du circuit ciblé limite la portabilité du VHDL à des circuits ciblés de la même famille.

#### *Simulation*

Avant l'avènement des environnements de développement VHDL, la simulation de systèmes numériques complexes était réalisable. Les grands acteurs du marché de la CAO proposaient des environnements propriétaires intégrant une saisie de schéma, un simulateur et des outils de visualisation de chronogrammes. Le concepteur assemblait des blocs à la schématique, les blocs possédant des descriptions HDL sous-jacentes étaient transformés en une liste de portes logiques et de bascules interconnectées entre elles. La simulation de ces blocs s'effectuait alors au niveau de la porte logique. En raison du nombre potentiel d'événements générés dans le simulateur par des milliers de portes, le développement de systèmes complexes était pénalisé par des temps de simulation importants. Il était possible de développer et intégrer au simulateur ses propres modèles de simulation, une librairie de fonctions était mise à disposition du concepteur pour gérer l'arrivée et la création d'événements sur les ports du modèle. Les modèles étaient écrits dans un langage séquentiel tel que le C. Dans l'environnement propriétaire CADENCE, ces modèles étaient appelés UCP (User Coded Primitives) et un simulateur devait être spécialement compilé pour intégrer ces modèles. Comparativement, un simulateur VHDL bénéficie des avantages inhérents au langage :

- Une saisie de schéma n'est pas indispensable. Le langage prévoit l'interconnexion de composants. Des descriptions de systèmes complexes peuvent être échangées entre concepteurs sous forme textuelle, les problèmes de portabilité entre saisies de schéma sont évités.
- Une gestion très avancée d'événements est intégrée au langage.
- La simulation peut être effectuée sur des descriptions de haut-niveau tels que des machines d'états et non plus au niveau de la porte logique. Les temps de simulation sont très réduits.

#### *Flot de développement VHDL*

Les phases de développements typiques (Figure 2) sont par ordre chronologique :

1. Descriptions du banc de test et du circuit électronique à réaliser en VHDL non RTL, validation du système par la simulation.
2. La description non synthétisable (RTL) est remplacée par une description équivalente synthétisable, validation du système par la simulation.
3. Synthèse du circuit VHDL RTL, paramétrée dans un format propre à l'outil de synthèse par les contraintes du concepteur. Le synthétiseur génère une description VHDL interconnectant des composants de la librairie du circuit ciblé dont les modèles de simulation sont

	<b>Université de Marne-La-Vallée</b>		
	Initiation au langage VHDL		
		Date: 13 juin 2008	Page : 6/57

conformes à la norme VITAL. Les modèles VITAL utilisés sont des modèles de simulation fonctionnels.

4. La description synthétisable est remplacée par la description synthétisée, validation du système par la simulation.
5. Placement/routage de la description VHDL synthétisée, paramétré dans un format propre à l'outil de placement/routage par les contraintes du concepteur. Le placeur/routeur génère une description VHDL interconnectant des composants de la librairie du circuit ciblé dont les modèles de simulation sont conformes à la norme VITAL. Les modèles VITAL utilisés sont des modèles de simulation rétro-annotés par les informations de temps de propagation et de délais extraites par le placeur/routeur.
6. La description synthétisée est remplacée par la description placée/routée, validation du système par la simulation.
7. Test in Situ

Les phases de simulation intensive avec correction de bogues fonctionnels sont effectuées sur les descriptions VHDL du concepteur. Les simulations post-synthèse et post-routage sont des simulations au niveau porte pénalisantes en temps de simulation. Elles sont utiles pour valider l'interprétation faite par le synthétiseur de la description VHDL et confirmer les fréquences de fonctionnement du système annoncées par les rapports d'analyse du placeur/routeur. Le concepteur juge de la pertinence de respecter tous les points de validation par la simulation. Il peut par exemple décider de ne pas effectuer les simulations au niveau porte et tester directement sa logique sur le matériel ciblé. En cas de dysfonctionnement constaté sur la cible, une simulation post-routage ou post-synthèse est utile au débogage. Le concepteur peut aussi décrire son système directement en VHDL RTL, la description VHDL non RTL est utile soit pour accélérer la modélisation de son système en s'affranchissant des limites imposées par la synthèse, soit dans les cas où la répartition des ressources entre le matériel et le logiciel n'est pas définie. Les descriptions VHDL générées par les outils de synthèse et de placement/routage ne sont utilisées que pour la simulation, les échanges entre ces deux outils se font dans un autre format. L'intérêt de ce flot de développement est que le même banc de test VHDL est utilisé à chaque étape de la création du circuit logique.

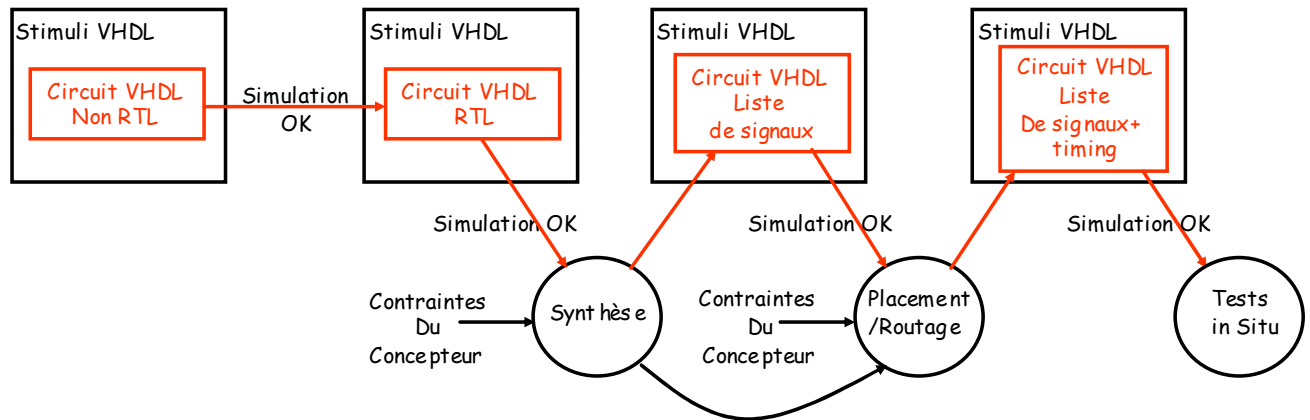


Figure 2 – Flot de développement VHDL

## 2.2. Particularités du langage

La différence majeure entre un langage classique de programmation, citons le "C" ou le "C++", et le langage VHDL est l'introduction de la notion de "process" et de "signaux". Cette notion change fondamentalement la façon de penser et concevoir un programme.

### 2.2.1. Signaux, process, instructions concurrentes et simulation

Une analogie directe est d'assimiler un signal à une équipotentielle d'un circuit électronique et comparer un process à un sous-programme d'interruption dont le signal d'interruption est un événement survenu sur l'un des signaux déclarés dans la liste de sensibilité du process. Il existe différentes façons de déclarer un process, la façon la plus proche de l'analogie avec le concept d'interruption est :

--Déclaration de process

Nom du process : **process** (liste de sensibilité)

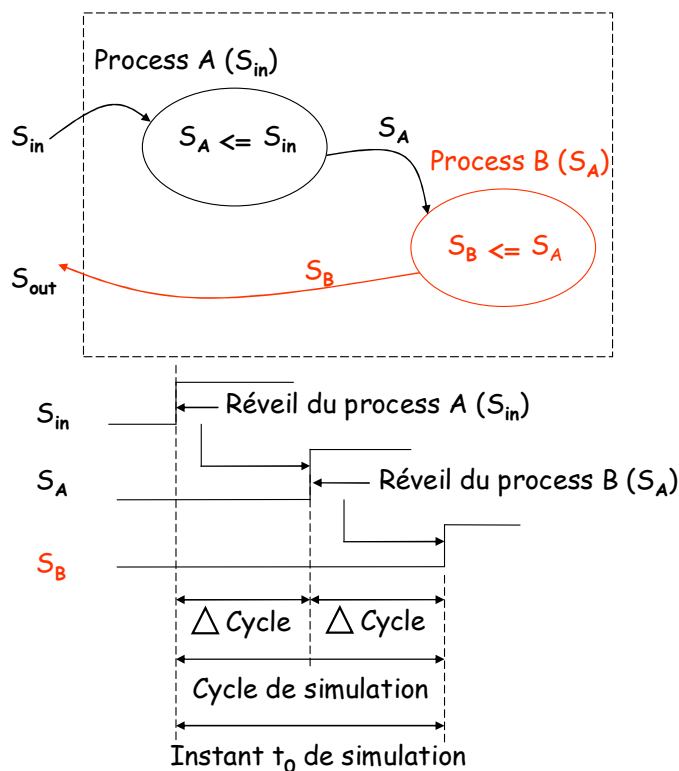
**Begin**

-- Corps du process

**End process** nom du process

La liste de sensibilité est constituée de un ou plusieurs signaux, elle peut être vide mais nous n'envisagerons pas ce cas dans cette introduction. Le corps du process peut contenir des affectations de signaux. Ces signaux, s'ils changent d'état, déclencheront à leur tour l'invocation des process dont ils font partie de la liste de sensibilité. Il est alors primordial d'intégrer la notion de causalité et de cycles de simulation pour comprendre le fonctionnement du mécanisme des process. Considérons une description VHDL décrivant plusieurs process communiquant entre eux par des signaux, l'état des signaux est stable jusqu'à l'instant de simulation  $t_0$  de stimulation du système par des signaux. A l'instant  $t_0$ , tous les process dont au moins un des signaux de la liste de sensibilité a

changé d'état sont classés actifs. Les instructions de tous les process actifs sont déroulées séquentiellement sachant que l'affectation d'un signal notée  $S_A \leq S_{in}$  (le signal  $S_A$  prend la valeur du signal  $S_{in}$ ) est effective après un cycle élémentaire de simulation nommé delta-cycle. Ce cycle élémentaire est un cycle interne de simulation, le temps de simulation n'a pas avancé et la valeur des signaux est mise à jour dans des structures mémoires nommées pilotes. Après un delta-cycle, les process sont à nouveau évalués pour être classés actif. Un nouveau cycle élémentaire de simulation est déclenché. Les delta-cycles se cascaden jusqu'à ce que plus aucun changement d'état ne soit constaté sur aucun des signaux intervenant dans la liste de sensibilité des process. Le système est alors stable, le temps de simulation peut avancer et les signaux prennent la dernière valeur mise à jour dans le pilote pour l'instant  $t_0$  (Figure 3).



**Figure 3** – Cycle de simulation

L'ordre dans lequel ont été évalués les process à chaque delta-cycle n'a pas d'importance. Une simulation VHDL repose sur le principe de concurrence entre des instructions dites concurrentes tels que les process, elles sont concurrentes car elles sont toutes évaluées à un même delta-cycle. L'évaluation d'une instruction concurrente à un delta-cycle ne peut pas influencer l'évaluation d'une autre instruction concurrente au même delta-cycle. Leur ordre d'apparition à l'intérieur d'une description VHDL n'a donc aucune importance. Par opposition, l'ordre d'évaluation des instructions séquentielles à l'intérieur d'un process doit respecter leur ordre d'apparition, comme pour un langage classique. A un même delta-cycle, une instruction séquentielle peut modifier des objets utilisés au même delta-cycle dans la suite de la description.

### 2.2.2. Inférence d'éléments mémoires

#### *Description d'un élément mémoire*

Un élément mémoire est identifiable lorsqu'un signal ne fait pas partie de la liste de sensibilité d'un



process mais participe à l'affectation d'un signal à l'intérieur du process. La Figure 4 illustre le cas d'affectations simples de signaux. Notez que le signal  $S_{out}$  prend la valeur de  $S_b$  à l'instant où un événement se produit sur  $S_{in}$ , sinon  $S_{out}$  garde sa valeur puisque le process n'est pas invoqué. Il y a donc mémorisation de la valeur de  $S_b$  sur les fronts montants et descendants de  $S_{in}$ .

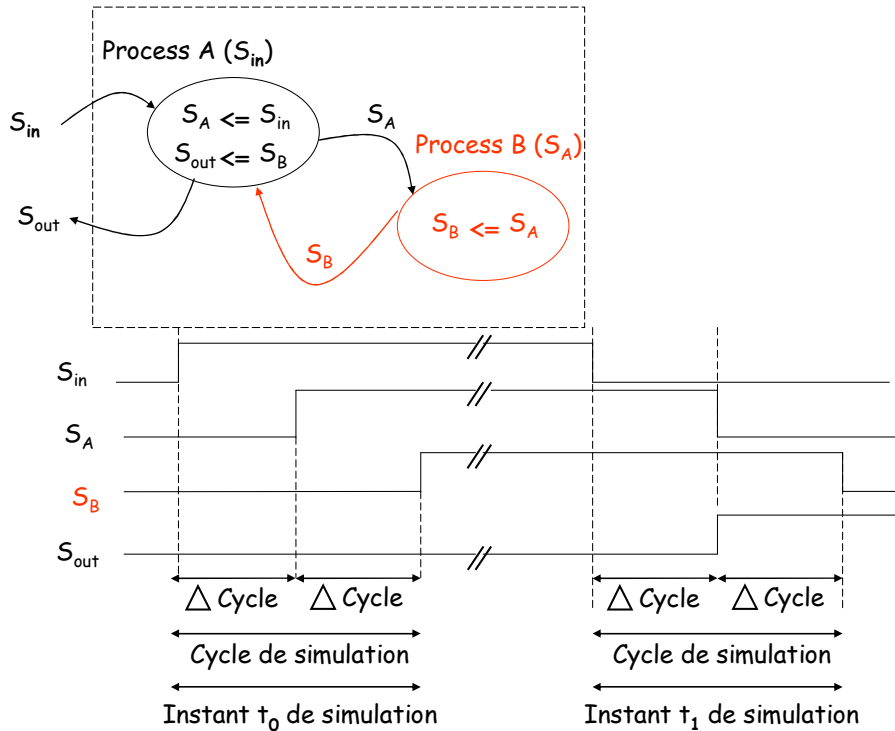


Figure 4 – inférence d'un élément mémoire

#### Synthèse d'un élément mémoire

Dans les langages HDL précurseurs du VHDL, les éléments mémoires ou combinatoires étaient déclarés explicitement. En VHDL, la déclaration d'un signal ne laisse pas supposer s'il sera connecté par exemple à la sortie d'une bascule D, d'une bascule T, d'un latch ou d'une porte logique ET. L'outil de synthèse doit analyser le contexte d'affectation d'un signal dans une description VHDL pour en déduire quelle logique lui associer. Prenons le cas d'une bascule D dont la table de vérité est représentée Figure 5. Le code VHDL suivant implémente cette table de vérité : le signal "q" prend la valeur du signal "d" lorsque un événement survient sur le signal "h" et que le signal vaut '1' (front montant), sinon "q" garde sa valeur.

#### Exemple code :

```
-- Le signal "d" prend la valeur du signal "q" sur front montant du signal "h"
Bascule : process (h)    -- Process réveillé par un événement survenu sur le signal "h"
Begin
If (h'event and h = '1') then    -- Transition du signal "h" vers le niveau '1'
    Q <= d;    -- Le signal "q" prend la valeur du signal "d"
End if
End process bascule
```

Bascule D

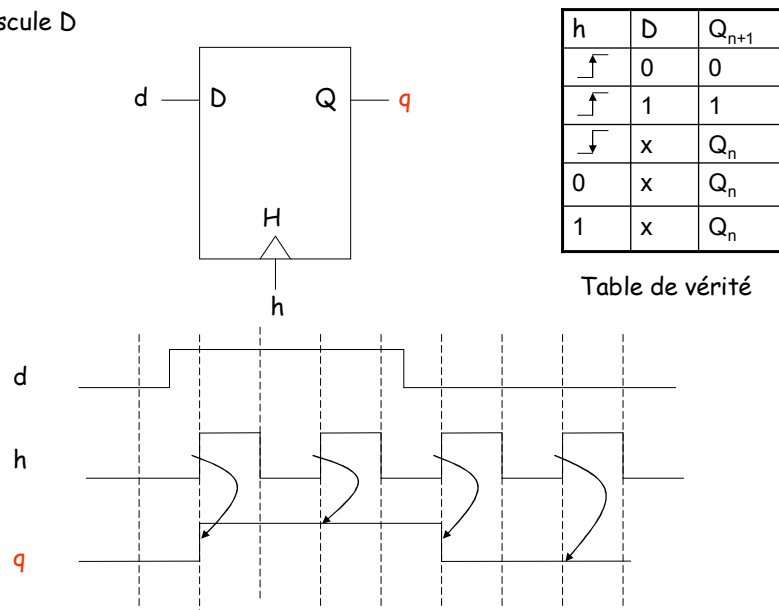


Figure 5 – Inférence d'une bascule D

### 3. LES ÉLÉMENTS DE BASE

Ce chapitre décrit les éléments de base du langage utiles à l'élaboration d'un programme VHDL. Une description syntaxique du langage incomplète est donnée, elle met en valeur les constructions les plus utiles et fréquemment rencontrées. Les descriptions syntaxiques entre crochets sont optionnelles. Pour une description complète de la grammaire du langage VHDL, il faut se référer au LRM (Language Reference Manual) [1].

#### 3.1. Les entités de conception et la configuration

Les entités de conception sont les briques d'une construction VHDL. Elles peuvent représenter un système complet, un sous-système, une carte, un circuit intégré, une porte logique. Elles présentent des ports d'entrée et de sortie bien définis et exécutent une fonction bien définie. Une configuration peut-être utilisée pour décrire comment ces entités sont connectées entre elles pour former un système complet. Une entité de conception est définie par une déclaration d'entité associée au corps d'une architecture.

##### 3.1.1. La déclaration d'entité

Une déclaration d'entité définit les ports d'interface entre une entité de conception et son environnement. Une déclaration d'entité peut être partagée par plusieurs entités de conception, chaque entité de conception possédant une architecture différente. Une déclaration d'entité peut être vue comme une classe d'entité de conception, chacune présentant les mêmes interfaces.

	<b>Université de Marne-La-Vallée</b>		
	Initiation au langage VHDL		
		Date: 13 juin 2008	Page : 11/57

### *Syntaxe*

Les paramètres génériques sont des constantes dont les valeurs peuvent être spécifiées par l'environnement, sinon une valeur par défaut doit leur être affectée.

Le champ mode des ports peut prendre les noms **in**, **out**, **inout**, **buffer** et **linkage**. Une valeur par défaut peut être affectée aux ports en mode **in**, cette valeur est prise en compte à la simulation et à la synthèse pour initialiser la valeur des signaux. Les modes des ports couramment utilisés sont : **in**, port d'entrée, **out**, port de sortie, **inout**, port d'entrée/sortie. Le mode **buffer** peut être utile, il permet de déclarer un port de sortie dont on peut relire la valeur à l'intérieur de l'unité de conception. Cependant, l'utilisation de ce mode est déconseillée car il est mal supporté par les outils de simulation et synthèse. La lecture est interdite sur un port de sortie de même que l'écriture sur un port d'entrée.

La partie déclarative des paramètres et ports déclare des objets qui sont communs à toutes les entités de conception dont les interfaces sont définies par la déclaration d'entité.

La partie instructions déclare des appels de procédures concurrentes ou des process. Ces déclarations sont utilisées pour contrôler les conditions d'opération de l'entité de conception. Les process ou procédures doivent être passifs, c'est-à-dire ne pas contenir d'affectation de signaux.

```

entity nomEntité is
  [generic (
    nomParamètre : type := valeurParDéfaut;
    ...
    nomParamètre : type := valeurParDéfaut);]
  [Port (
    nomPort : mode type := valeurParDéfaut;
    ...
    nomPort : mode type := valeurParDéfaut);]
  [déclarations sous-programmes, types, constantes...]
  [begin instructions process ou procédures passifs]
end [entity] [nomEntité];

```

### Exemple code 1 :

-- Déclaration d'entité pour un banc de test

**Entity** bancDeTest **is**

**End entity** bancDeTest;

	<b>Université de Marne-La-Vallée</b>		
	Initiation au langage VHDL		
		Date: 13 juin 2008	Page : 12/57

Exemple code 2:

-- Déclaration d'entité pour une bascule D

```
Entity basculeD is
  Port (
    d, h : in std_logic;
    q    : out std_logic
  );
End entity basculeD;
```

Exemple code 3:

-- Déclaration d'entité pour une bascule D avec contrôle temporel

```
Entity basculeD is
  Generic (
    tSetUp    : time := 5 ns ;
    tHold     : time := 4 ns ;
    tCkToQ   : time := 2 ns
  );
  Port (
    d      : in std_logic := '1';
    h      : in std_logic := '1';
    q      : out std_logic
  );
  Constant : pulseWidth : time := 10 ns ;
  Begin
  checkSetUpTime (tSetUp, d, h) ;
  checkHoldTime (tHold, d, h) ;
  checkPulseWidth (pulseWidth, h) ;
End entity basculeD;
```

### 3.1.2. Les architectures

Le corps d'une architecture définit le corps d'une entité de conception. Il spécifie les relations entre les entrées et les sorties d'une entité de conception. Cette spécification peut être exprimée sous forme comportementale, structurelle, flot de données, les trois formes peuvent coexister à l'intérieur d'un même corps d'architecture.

La partie déclarative déclare des objets qui seront visibles et utilisables par l'entité de conception. Toutes les instructions qui décrivent le corps de l'architecture sont des instructions concurrentes qui s'exécutent de façon asynchrone les unes par rapport aux autres.

	<b>Université de Marne-La-Vallée</b>		
	Initiation au langage VHDL		
		Date: 13 juin 2008	Page : 13/57

```

architecture nomArchitecture of nomEntité is
[déclarations sous-programmes, types, constantes,
signaux, alias, composants...]
begin
instructions concurrentes
end [architecture] [nomArchitecture];
```

Exemple code 1:

*-- Description d'une architecture pour une bascule D*

```

architecture rtl of basculeD is
begin
  pBascD : process (h)
  begin
    if (h'event and h = '1') then
      q    <= d;
    end if;
  end process pBascD;
end architecture rtl;
```

Exemple code 2:

*-- Description d'une architecture pour un registre à décalage*

```

architecture rtl of regDecal is
signal sig1, sig2 : std_logic;
begin
  pRegDecal : process (h)
  begin
    if (h'event and h = '1') then
      sig1 <= d;
      sig2 <= sig1;
      q    <= sig2;
    end if;
  end process pRegDecal;
end architecture rtl;
```

	<b>Université de Marne-La-Vallée</b>		
	Initiation au langage VHDL		
		Date: 13 juin 2008	Page : 14/57

### 3.1.3. La configuration

#### *Déclaration d'un composant*

La déclaration d'un composant déclare les interfaces à une entité de conception virtuelle. Une configuration de composant peut être utilisée pour associer l'instance d'un composant avec une unité de conception présente en librairie.

```

component nomComposant [is]
[generic (
  nomParamètre : type := valeurParDéfaut;
  ...
  nomParamètre : type := valeurParDéfaut) ;]
Port (
  nomPort : mode type;
  ...
  nomPort : mode type)
end component [nomComposant];

```

#### Exemple code :

```

-- Déclaration du composant basculeD
component basculeD is
  Port (
    d      : in std_logic;
    h      : in std_logic ;
    q      : out std_logic
  );
end component basculeD;

```

#### *Instanciation d'un composant*

L'instanciation d'un composant dans la description d'une unité de conception définit un sous-composant de cette unité. L'instanciation associe des signaux ou des valeurs aux ports et des valeurs aux paramètres génériques de ce sous-composant. Il existe trois façons d'instancier un composant :

1. En précisant le nom du composant instancié, ce nom doit être le nom d'un composant déjà déclaré. Des règles d'association par défaut sont appliquées dans le cas où le composant est instancié par un nom. En l'absence de configuration, l'association est faite avec la déclaration d'entité qui a le même nom que celui du composant instancié.
2. en précisant le nom de l'entité, ce nom doit être le nom d'une entité déjà déclarée. Si un nom d'architecture est présent, il doit être le nom du corps d'une architecture associée à l'entité. Le nom d'architecture est utilisé pendant la phase d'élaboration du système pour

	<b>Université de Marne-La-Vallée</b>		
	Initiation au langage VHDL	Date: 13 juin 2008	Page : 15/57

- sélectionner un corps d'architecture parmi éventuellement plusieurs corps d'architecture.
3. en précisant le nom d'une configuration, ce nom doit être le nom d'une configuration déjà déclarée.

```

labelInstance : [component] nomComposant
ou
labelInstance : entity nomEntité [(nomArchitecture)]
ou
labelInstance : configuration nomConfiguration
[generic map (
  nomParamètre => valeur;
  ...
  nomParamètre => valeur) ;]
[Port map (
  nomPort => nomSignal;
  ...
  nomPort => nomSignal)];

```

Exemple code 1:

```

-- instantiation d' un composant par son nom
architecture rtlStructural of regDecal is
signal sig1, sig2 : std_logic;
component basculeD is
  Port (
    h, d : in std_logic;
    q    : out std_logic
  );
end component basculeD;
begin
--instantiation bascules 1, 2, 3
basculeD1: basculeD
  port map (h => h, d => d, q => sig1);
basculeD2: basculeD
  port map (h => h, d => sig1, q => sig2);
basculeD3: basculeD
  port map (h => h, d => sig2, q => q);
end architecture rtlStructural;

```

	<b>Université de Marne-La-Vallée</b>		
	Initiation au langage VHDL		
		Date: 13 juin 2008	Page : 16/57

Exemple code 2:

```
-- instantiation d'un composant par le nom d'une entité
architecture rtlStructural of regDecal is
signal sig1, sig2 : std_logic;
begin
--instantiation bascules 1, 2, 3
basculeD1: entity basculeD (rtl)
    port map (h => h, d => d, q => sig1);
basculeD2: entity basculeD (rtl)
    port map (h => h, d => sig1, q => sig2);
basculeD3: entity basculeD (rtl)
    port map (h => h, d => sig2, q => q);
end architecture rtlStructural;
```

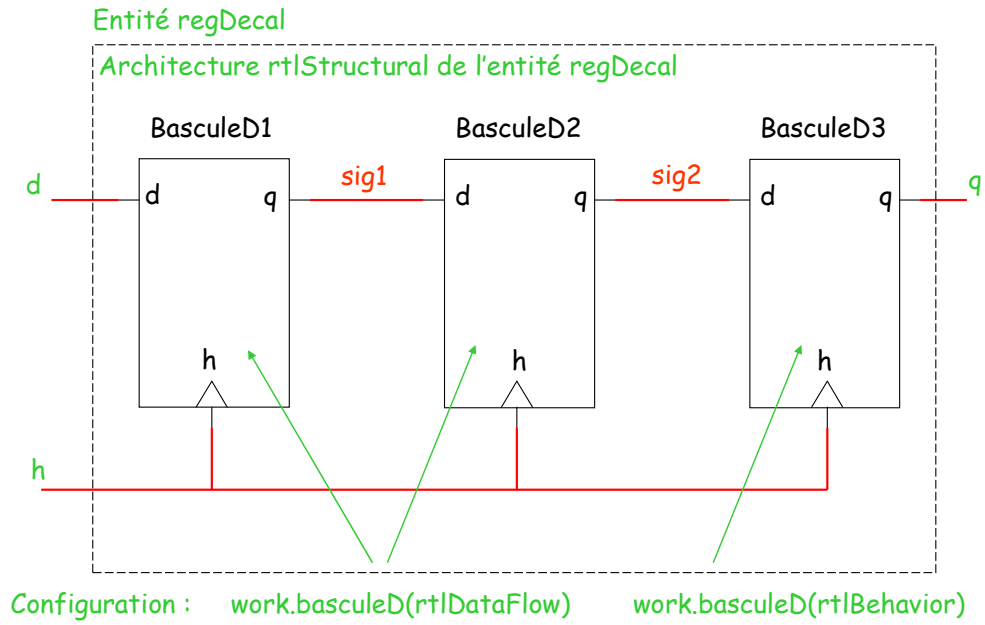
*Configuration de composants*

Les composants instanciés par un nom peuvent être configurés en précisant leur association avec une entité déjà déclarée et une architecture. La configuration peut être explicitée dans un module de configuration dédié ou bien précisée au sein des corps d'architecture où les composants sont instanciés.

```
Configuration nomConfiguration of nomEntité is
For nomArchitecture
    For labelInstance : nomComposant
        Use entity nomBibliothèque.nomEntité (nomArchitecture)
            [generic map (
                nomParametre => valeur;
                ...
                nomParametre => valeur) ;]
            [Port map (
                nomPort => nomSignal;
                ...
                nomPort => nomSignal)];
    End for
End for
End [configuration] [nomConfiguration]
```

L'exemple suivant illustre la configuration d'un registre à décalage (Figure 6) à l'aide des deux méthodes.





**Figure 6** – Architecture structurelle de l'entité regDecal

	<b>Université de Marne-La-Vallée</b>		
	Initiation au langage VHDL		
		Date: 13 juin 2008	Page : 18/57

Exemple Code :

*--commun aux 2 exemples de configuration*

*-- bascule D*

**library** ieee;

**use** ieee.std\_logic\_1164.all;

**entity** basculeD **is**

**port** (

    d, h : in std\_logic;

    q : out std\_logic

);

**end entity** basculeD ;

**architecture** rtlBehavior **of** basculeD **is**

**begin**

    pBascD : **process** (h)

**begin**

**if** (h'event and h = '1') **then**

            q <= d;

**end if**;

**end process** pBascD;

**end architecture** rtlBehavior;

**architecture** rtlDataFlow **of** basculeD **is**

**begin**

    q <= d **when** (h'event **and** h = '1');

**end architecture** rtlDataFlow;

*--registre à décalage*

**library** ieee;

**use** ieee.std\_logic\_1164.all;

**entity** regDecal **is**

**port** (

    h,d : in std\_logic;

    q : out std\_logic

);

**end regDecal**;

Dans le code suivant, la configuration est explicitée au sein du corps d'architecture où les composants sont instanciés.

	<b>Université de Marne-La-Vallée</b>		
	Initiation au langage VHDL		
		Date: 13 juin 2008	Page : 19/57

Sous-exemple code 1:

```

architecture rtlStructural of regDecal is
signal sig1, sig2 : std_logic;
component c_basculeD is
    Port (
        h, d   : in std_logic;
        q     : out std_logic
    );
end component c_basculeD;
for basculeD1 : c_basculeD use entity work.basculeD (rtlDataFlow);
for basculeD2 : c_basculeD use entity work.basculeD (rtlDataFlow);
for basculeD3 : c_basculeD use entity work.basculeD (rtlBehavior);
begin
--instantiation bascules 1, 2, 3
basculeD1: c_basculeD
    port map (h => h, d => d, q => sig1);
basculeD2: c_basculeD
    port map (h => h, d => sig1, q => sig2);
basculeD3: c_basculeD
    port map (h => h, d => sig2, q => q);
end architecture rtlStructural;

```

Dans le code suivant, la configuration est explicitée dans un module de configuration dédié.

	Université de Marne-La-Vallée		
	Initiation au langage VHDL		
		Date: 13 juin 2008	Page : 20/57

Sous-exemple code 2:

```

architecture rtlStructural of regDecal is
signal sig1, sig2 : std_logic;
component c_basculeD is
Port (
        h, d   : in std_logic;
        q     : out std_logic
);
end component c_basculeD;
begin
--instanciation bascules 1, 2, 3
basculeD1 : c_basculeD
    port map (h => h, d => d, q     => sig1);
basculeD2 : c_basculeD
    port map (h => h, d => sig1, q => sig2);
basculeD3 : c_basculeD
    port map (h => h, d => sig2, q => q);
end architecture rtlStructural;

--configuration du registre à décalage
configuration c1 of regDecal is
for rtlStructural
    for basculeD1 : c_basculeD
        use entity work.basculeD (rtlDataFlow);
    end for;
    for basculeD2 : c_basculeD
        use entity work.basculeD (rtlDataFlow);
    end for;
    for basculeD3 : c_basculeD
        use entity work.basculeD (rtlBehavior);
    end for;
end for;
end configuration c1;

-- banc de test
library ieee;
use ieee.std_logic_1164.all;

entity bancDeTest is
end entity bancDeTest;

architecture tb of bancDeTest is
signal h, d: std_logic := '0';
signal q : std_logic;
component c_regDecal is

```

	Université de Marne-La-Vallée		
	Initiation au langage VHDL		
		Date: 13 juin 2008	Page : 21/57

```

    port(
        h,d   : in std_logic;
        q     : out std_logic
    );
end component c_regDecal;
for regDecal1 : c_regDecal use configuration work.c1;
begin
h <= not h after 100 ns;
d <= not d after 350 ns;
regDecal1 : c_regDecal
    port map (h => h, d => d, q => q);
end architecture tb;

```

#### *Remarques sur la connexion de composants*

. Un port en mode **in** peut être associé à une expression statique. Tout port peut être associé à un signal, ou à un port d'une autre entité. Il est alors dit connecté, un port en mode **in** doit être connecté à moins qu'une valeur par défaut lui soit affectée. Un port est non connecté s'il est associé au mot réservé **open**. Les modes des ports couramment utilisés sont **in**, **out** et **inout**.

### 3.2. Les types de données

Le langage VHDL est un langage fortement typé, tout objet utilisé dans une description doit au préalable être déclaré et associé à un type. Cette contrainte autorise des vérifications à la compilation sur la compatibilité des objets mis en relation dans des expressions de test ou d'affectation. Les types couramment utilisés sont les types scalaires, composites et fichiers. L'utilisation des types accès et protégés est marginale. A partir de types de base, le langage prévoit de définir des sous-types

#### 3.2.1. Type scalaire

Les types scalaires sont classés en type énuméré, entier, physique, flottant.

##### *Type énuméré*

Les types prédéfinis dans le package standard sont les types boolean, bit, character et severity\_level.

Définition des types boolean, bit, character, severity\_level

- **type** BOOLEAN **is** (FALSE, TRUE);
- **type** BIT **is** ('0', '1');
- **type** CHARACTER **is** (NUL, SOH, STX,...) ; --256 characters
- **type** SEVERITY\_LEVEL **is** (NOTE, WARNING, ERROR, FAILURE);

Des types importants prédéfinis dans le package std\_logic\_1164 sont les types std\_ulogic, std\_logic et std\_logic\_vector.

Définition des types std\_ulogic, std\_logic et std\_logic\_vector

- **type** std\_ulogic **is** ('U', -- Uninitialized  
'X', -- Forcing Unknown

```
'0', -- Forcing 0
'1', -- Forcing 1
'Z', -- High Impedance
'W', -- Weak Unknown
'L', -- Weak 0
'H', -- Weak 1
'-' -- Don't care
);
```

- **subtype** std\_logic **is resolved** std\_ulogic;
- **type** std\_logic\_vector **is** array ( natural **range**  $\diamond$ ) **of** std\_logic;

A l'origine, seul le type BIT était prédéfini par la norme IEEE **P1076-87**, les développeurs ont manifesté le besoin de modéliser notamment les états "haute-impédance" et "inconnu" et la norme IEEE **P1076-93** a introduit le package std\_logic\_1164.

Fonction de résolution

```
-----
-- resolution function
-----
-
CONSTANT resolution_table : stdlogic_table := (
-- -----
-- | U X 0 1 Z W L H - | |
-- -----
('U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U'), -- | U |
('U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X'), -- | X |
('U', 'X', '0', 'X', '0', '0', '0', '0', 'X'), -- | 0 |
('U', 'X', 'X', '1', '1', '1', '1', '1', 'X'), -- | 1 |
('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', 'X'), -- | Z |
('U', 'X', '0', '1', 'W', 'W', 'W', 'W', 'X'), -- | W |
('U', 'X', '0', '1', 'L', 'W', 'L', 'W', 'X'), -- | L |
('U', 'X', '0', '1', 'H', 'W', 'W', 'H', 'X'), -- | H |
('U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X') -- | - |
);
```

### Type entier

Un type entier définit un entier dont l'ensemble des valeurs est inclus dans une plage spécifiée

#### Exemple

- **type** BYTE\_LENGTH\_INTEGER **is range** 0 to 255;
- **subtype** HIGH\_BIT\_LOW **is** BYTE\_LENGTH\_INTEGER **range** 0 to 127;

Le seul type prédéfini est le type **integer**, la plage de variation d'un **integer** dépend de la machine

	<b>Université de Marne-La-Vallée</b>		
	Initiation au langage VHDL	Date: 13 juin 2008	Page : 23/57

sur laquelle est compilée la description VHDL. Cette plage peut être déterminée par la valeur des attributs 'low et 'high du type **integer** (**integer**'low et **integer**'high). Quelle que soit la machine, elle inclue la plage -2147483647 à +2147483647.

### *Type physique*

Le seul type physique prédéfini est le type **time**. La plage de variation d'un **time** dépend de la machine sur laquelle est compilée la description VHDL. Quelle que soit la machine, elle inclue la plage -2147483647 à +2147483647. La déclaration du type **time** se trouve dans le package **standard**.

Définition du type **time**

- **type TIME is range implementation\_defined**  
**units**  
fs; -- femtosecond  
ps = 1000 fs; -- picosecond  
ns = 1000 ps; -- nanosecond  
us = 1000 ns; -- microsecond  
ms = 1000 us; -- millisecond  
sec = 1000 ms; -- second  
min = 60 sec; -- minute  
hr = 60 min; -- hour  
**end units;**

### *Type flottant*

Le seul type flottant prédéfini est le type **real**, la plage de variation d'un **real** est machine dépendante. Cette plage peut être déterminée par la valeur des attributs 'low et 'high du type **real** (**real**'low et **real**'high).

### 3.2.2. Type composite

Le type composite est utilisé pour définir des collections de valeurs. Le type est nommé "tableau" si les valeurs d'une collection sont du même type, le type est nommé "record" si les valeurs d'une collection sont de type hétérogène.

### *Type tableau*

Les types définissant des tableaux peuvent être contraints ou non contraints.

Exemple

- **type MY\_WORD is array (0 to 31) of BIT ;**
- **type MEMORY is array (INTEGER range <>) of MY\_WORD ;**

Les types prédéfinis dans le package standard sont les types **string** et **bit\_vector**.

Définition du type **string**

- **subtype POSITIVE is INTEGER range 1 to INTEGER'HIGH ;**
- **type STRING is array (POSITIVE range <>) of CHARACTER ;**

	<b>Université de Marne-La-Vallée</b>		
	Initiation au langage VHDL		
		Date: 13 juin 2008	Page : 24/57

Définition du type **bit\_vector**

- **subtype** NATURAL **is** INTEGER **range** 0 **to** INTEGER'HIGH ;
- **type** BIT\_VECTOR **is** array (NATURAL range  $\langle \rangle$ ) **of** BIT ;

*Type record*

Exemple

- **type** DATE **is**  
**record**  
DAY: INTEGER **range** 1 **to** 31;  
MONTH: MONTH\_NAME;  
YEAR: INTEGER **range** 0 **to** 4000;  
**end record**;



	<b>Université de Marne-La-Vallée</b>	
	Initiation au langage VHDL	
		Date: 13 juin 2008
		Page : 25/57

Exemple code :

```

-- utilisation record
library ieee;
use ieee.std_logic_1164.all;
use std.textio.all;

entity testRecord is
end entity testRecord ;

architecture eval of testRecord is
begin
test : process
variable l : line;
type DATE is
    record
        jour: INTEGER range 1 to 31;
        mois: INTEGER range 1 to 12;
        annee: INTEGER range 0 to 4000;
    end record;
variable dateNaissance : DATE;
begin

dateNaissance.jour      := 3;
dateNaissance.mois     := 2;
dateNaissance.annee    := 1984;

write (l, STRING("date de naissance : "));
write (l, dateNaissance.jour);
write (l, STRING("/ "));
write (l, dateNaissance.mois);
write (l, STRING("/ "));
write (l, dateNaissance.annee);
writeLine (output, l);

wait;

end process test;
end architecture eval;

```

### 3.2.3. Type fichier

Les types fichiers sont utilisés pour définir des objets représentant des fichiers stockés sur la machine de développement du code VHDL.

Le package standard TEXTIO permet de manipuler des fichiers de type ascii. L'accès en écriture et en lecture à des fichiers est très utile pour la description de bancs de test. Des fichiers d'entrée peuvent stocker les stimuli d'excitation du système à tester et des fichiers de sortie peuvent stocker les résultats. Ces fichiers sont archivables avec la description VHDL.

	<b>Université de Marne-La-Vallée</b>		
	Initiation au langage VHDL		
		Date: 13 juin 2008	Page : 26/57

Exemple Code:

```

architecture eval of testRange is
begin
test : process
variable r : real;
variable i : integer;
variable l : line;
file fichierRange : text open WRITE_MODE is "range.txt";
begin

write (l, STRING("range integer "));

write (l, STRING(" low :"));
i := integer'low;
write (l, i);

write (l, STRING(" high :"));
i := integer'high;
write (l, i);

writeLine (fichierRange, l);

wait;

end process test;
end architecture eval;

```

### 3.3. Les classes d'objet

#### 3.3.1. Les signaux

Les signaux représentent les équipotentielles d'un circuit électronique. Ils sont affectés par l'instruction "<="". Cette instruction est séquentielle si elle est rencontrée dans le corps d'un process et concurrente à l'extérieur.

#### *Les attributs*

Des attributs sont associés aux signaux. Les plus utilisés pour un signal nommé "s" sont :

- S'event représente une fonction qui renvoie la valeur booléenne "true" si un événement est survenu sur le signal dans le cycle de simulation courant.
- S'last\_event représente une fonction qui renvoie la valeur de type "time" égale au temps passé depuis le dernier événement.
- S'stable(T) représente un signal qui prend la valeur booléenne "true" quand aucun événement n'est survenu sur le signal "s" pendant la durée de type "time" T.

	<b>Université de Marne-La-Vallée</b>		
	Initiation au langage VHDL		
		Date: 13 juin 2008	Page : 27/57

### 3.3.2. Les variables

Les variables ne peuvent apparaître que dans le domaine de programmation séquentiel. Elles sont affectées par l'instruction ":", l'affectation est immédiate.

### 3.3.3. Les constantes

Sont assimilés à des constantes tous les objets dont la valeur peut être résolue durant la phase d'élaboration de la description VHDL. Les constantes sont déclarées par le mot clé **constant**, les paramètres génériques déclarés dans l'en-tête des entités doivent aussi être des constantes.

### 3.3.4. Les fichiers

Les fichiers sont des objets spéciaux utilisés exclusivement dans des descriptions non synthétisables.

## 3.4. Les sous-programmes

### 3.4.1. Les procédures

Tout comme l'instruction d'affectation d'un signal, une procédure appelée au même niveau que l'instruction "process" est dite concurrente alors qu'une procédure appelée à l'intérieur du corps d'un process est dite séquentielle.

#### *Déclaration*

```

Procedure nomProcedure ([classeObjet] nomParametre : [sens] type ;
                        ...
                        [classeObjet] nomParametre : [sens] type) ;
classeObjet = constant ou variable ou signal
Sens = in ou out ou inout
Défaut : classeObjet=variable et sens=in

```

#### *Définition*

```

Procedure nomProcedure ([classeObjet] nomParametre : [sens] type ;
                        ...
                        [classeObjet] nomParametre : [sens] type) is
Begin
    ListeInstructionsSéquentielles
End procedure [nomProcedure] ;

```

	<b>Université de Marne-La-Vallée</b>		
	Initiation au langage VHDL	Date: 13 juin 2008	Page : 28/57

Exemple code

-- exemple procedure séquentielle

**library** ieee;

**use** ieee.std\_logic\_1164.all;

**use** std.textio.all;

**entity** testProcedure is

**end entity** testProcedure ;

**architecture** eval of testProcedure is

**procedure** rotate ( **variable** aDecaler : **in** bit\_vector(7 downTo 0);  
**variable** decale : **out** bit\_vector(7 downTo 0);  
**constant** deCombien : **in** integer;  
**constant** direction : **in** string) is

**begin**

**if** (direction = "right") **then**  
    decale := aDecaler **ror** deCombien;

**elsif** (direction = "left") **then**  
    decale := aDecaler **rol** deCombien;

**else**  
    **null**;

**end if**;

**end procedure** rotate;

**begin**

**test** : **process**

**variable** l : line;

**variable** byteIn, byteOut : bit\_vector(7 downTo 0);

**begin**

    byteIn := "10110000";  
    write (l, STRING("avant rotation : "));  
    write (l, byteIn);  
    writeLine (output, l);  
    rotate (byteIn, byteOut, 3, "left");  
    write (l, STRING("apres rotation : "));  
    write (l, byteOut);  
    writeLine (output, l);  
    **wait**;

**end process** test;

**end architecture** eval;

Résultats sortie simulation :

# avant rotation : 10110000

# apres rotation : 1000101

Exemple Code :

	<b>Université de Marne-La-Vallée</b>		
	Initiation au langage VHDL		
		Date: 13 juin 2008	Page : 29/57

*-- exemple procedure concurrente, registre a decalage*

**library** ieee;

**use** ieee.std\_logic\_1164.all;

**entity** testProcedureConc **is**

**port** (

    rst, h, d       : **in** std\_logic;

    q             : **out** std\_logic

);

**end entity** testProcedureConc ;

**architecture** rtl **of** testProcedureConc **is**

**signal** sig1, sig2 : std\_logic;

**procedure** registre (**signal** raz             : **in** std\_logic;

**signal** horloge       : **in** std\_logic;

**signal** entree        : **in** std\_logic;

**signal** sortie        : **out** std\_logic) **is**

**begin**

**if** (raz = '1') **then**

        sortie <= '0';

**elsif** (horloge'event **and** horloge = '1') **then**

        sortie <= entree;

**end if**;

**end procedure** registre;

**begin**

    reg1:registre (rst, h, d, sig1);

    reg2:registre (raz=>rst, entree=>sig1, sortie=>sig2, horloge=>h);

    reg3:registre (rst, h, sig2, q);

**end architecture** rtl;

### 3.4.2. Les fonctions

#### *Déclaration*

**function** nomFonction (nomParametre : type ;

    ...

    nomParametre : type)

**return** type ;

	Université de Marne-La-Vallée		
	Initiation au langage VHDL		
		Date: 13 juin 2008	Page : 30/57

*Définition*

```
function nomFonction (nomParametre : type ;  
                    ...  
                    nomParametre : type) is  
Begin  
    ListeInstructionsSéquentielles  
End function [nomFonction] ;
```

	Université de Marne-La-Vallée		
	Initiation au langage VHDL		
		Date: 13 juin 2008	Page : 31/57

Exemple code

```

-- exemple fonction
library ieee;
use ieee.std_logic_1164.all;
use std.textio.all;

entity testFonction is
end entity testFonction ;

architecture eval of testFonction is
function rotate (aDecaler      : bit_vector(7 downTo 0);
                 deCombien     : integer;
                 direction      : string) return bit_vector is
variable decale : bit_vector(7 downTo 0);
begin
    decale := (others=>'0');
    if (direction = "right") then
        decale := aDecaler ror deCombien;
    elsif (direction = "left") then
        decale := aDecaler rol deCombien;
    else
        null;
    end if;
    return decale;
end function rotate;
begin
test : process
variable l : line;
variable byteIn, byteOut : bit_vector(7 downTo 0);
begin
    byteIn := "10110000";
    write (l, STRING("avant rotation : "));
    write (l, byteIn);
    writeLine (output, l);
    byteOut := rotate (byteIn, 3, "left");
    write (l, STRING("apres rotation : "));
    write (l, byteOut);
    writeLine (output, l);
    wait;

end process test;
end architecture eval;

```

Résultats sortie simulation :  
# avant rotation : 10110000  
# apres rotation : 10000101

	<b>Université de Marne-La-Vallée</b>		
	Initiation au langage VHDL		
		Date: 13 juin 2008	Page : 32/57

### 3.5. Les éléments d'archivage

#### 3.5.1. Les librairies

Une librairie est une bibliothèque où sont stockées des entités de conception précompilées. Elles sont déclarées en en-tête d'une description VHDL par l'instruction **library** et l'instruction **use** permet de sélectionner quels objets pourront être utilisés dans la suite de la description. Les descriptions sont compilées dans la librairie par défaut de travail **work** à moins qu'une autre librairie soit couramment activée dans l'outil de compilation. La déclaration des librairies **std**, **work** et la sélection des paquetages et corps de librairies **std.standard.all**, **work.all** n'est pas nécessaire.

#### 3.5.2. Les paquetages

##### *Spécification et corps de paquetage*

Les spécifications de paquetages définissent les prototypes de procédure et fonctions, déclarent des constantes, des composants, des types. Le corps du paquetage décrit le corps des fonctions et procédures.

```

Package nomPaquetage is
    listeInstructionsDéclarations
end package nomPaquetage ;

Package body nomPaquetage is
    listeInstructionsDéclarationsInternes
    corpsProcedures
    corpsFpnctions
End [package body] [nomPaquetage] ;
```

##### *Les paquetages normalisés*

Les paquetages développés et normalisés par l'IEEE sont les suivants :

- standard, textio (Librairie std)
- math\_complex, math\_real, numeric\_bit, numeric\_std, std\_logic\_1164 (librairie ieee)

Les paquetages développés par la société SYNOPSYS et utilisés dans beaucoup d'environnement VHDL sont les suivants :

- std\_logic\_arith, std\_logic\_misc, std\_logic\_signed, std\_logic\_textio, std\_logic\_unsigned (librairie ieee)

### 3.6. Eléments lexicaux

#### 3.6.1. Éléments basiques

##### *Commentaires et insensibilité aux types des lettres*

Le langage est totalement insensible au type minuscule ou majuscule des lettres. Les caractères



	Université de Marne-La-Vallée		
	Initiation au langage VHDL		
		Date: 13 juin 2008	Page : 33/57

d'une ligne suivant les signes "-" sont commentés. Il est possible de passer des paramètres aux outils de synthèse à l'aide de mots clés, ces mots clés sont dépendants de l'outil de synthèse.

*Représentation des nombres*

Exemple

-- Integer literals of value 255:

2#1111\_1111#

16#FF#

016#0FF#

-- Integer literals of value 224:

16#E#E1

2#1110\_0000#

-- Real literals of value 4095.0:

16#F.FF#E+2

2#1.1111\_1111\_111#E11

### 3.6.2. Mots réservés

<b>abs</b>	<b>generate</b>	<b>package</b>	<b>wait</b>
<b>access</b>	<b>generic</b>	<b>port</b>	<b>when</b>
<b>after</b>	<b>group</b>	<b>postponed</b>	<b>while</b>
<b>alias</b>	<b>guarded</b>	<b>procedural</b>	<b>with</b>
<b>all</b>		<b>procedure</b>	
<b>and</b>		<b>process</b>	
<b>architecture</b>		<b>protected</b>	
<b>array</b>		<b>pure</b>	
<b>assert</b>			
<b>attribute</b>			
<b>begin</b>	<b>if</b>	<b>range</b>	<b>xnor</b>
<b>block</b>	<b>impure</b>	<b>record</b>	<b>xor</b>
<b>body</b>	<b>in</b>	<b>reference</b>	
<b>buffer</b>	<b>inertial</b>	<b>register</b>	
<b>bus</b>	<b>inout</b>	<b>reject</b>	
	<b>is</b>	<b>rem</b>	
		<b>report</b>	
		<b>return</b>	
		<b>rol</b>	
		<b>ror</b>	
<b>case</b>	<b>label</b>	<b>select</b>	
<b>component</b>	<b>library</b>	<b>severity</b>	
<b>configuration</b>	<b>linkage</b>	<b>signal</b>	
<b>constant</b>	<b>literal</b>	<b>shared</b>	
	<b>loop</b>	<b>sla</b>	
		<b>sll</b>	
		<b>sra</b>	
		<b>srl</b>	
		<b>subtype</b>	
<b>disconnect</b>	<b>map</b>	<b>then</b>	
<b>downto</b>	<b>mod</b>	<b>to</b>	
		<b>transport</b>	
		<b>type</b>	
<b>else</b>	<b>nand</b>	<b>unaffected</b>	
<b>elsif</b>	<b>new</b>	<b>units</b>	
<b>end</b>	<b>next</b>	<b>until</b>	
<b>entity</b>	<b>nor</b>	<b>use</b>	
<b>exit</b>	<b>not</b>		
	<b>null</b>		
<b>file</b>	<b>of</b>	<b>variable</b>	
<b>for</b>	<b>on</b>		
<b>function</b>	<b>open</b>		
	<b>or</b>		
	<b>others</b>		
	<b>out</b>		

Tableau 1 – Mots réservés

	<b>Université de Marne-La-Vallée</b>		
	Initiation au langage VHDL		
		Date: 13 juin 2008	Page : 35/57

## 4. LA PROGRAMMATION VHDL

### 4.1. Les instructions

#### 4.1.1. Instructions séquentielles

Les instructions séquentielles sont utilisées pour définir l'algorithme de traitement exécuté par un sous-programme ou un process. Elles sont exécutées dans l'ordre où elles apparaissent.

##### *Instruction "wait"*

L'instruction "wait" provoque la suspension du process ou de la procédure dans lesquels elle apparaît.

**Wait** [**on** nomSignal1, nomSignal2...] [**until** expressionBooléenne] [**for** expressionTemps]

L'instruction wait suspend le sous-programme ou le process tant qu'aucun événement ne survient sur l'un des signaux de la liste de sensibilité (nomSignal1, nomSignal2...) et que l'expression booléenne n'est pas évaluée à TRUE. Tous les signaux de la liste de sensibilité doivent être accessibles en lecture. Si aucune attente sur une condition booléenne n'est spécifiée, elle est supposée toujours vraie. La clause temporelle exprime une condition de "dépassement de temps". Au delà de la durée spécifiée et même si les conditions d'attente sur les signaux et d'évaluation de l'expression booléenne ne sont pas satisfaites, le process ou le sous-programme continue. Si aucune clause temporelle n'est exprimée, la clause **for** (std.standard.time'high – std.standard.now) est implicite, std.standard.now représentant la valeur courante du temps et std.standard.time'high représentant la valeur maximale que peut prendre le temps. La condition de dépassement du temps n'est alors jamais vérifiée. Si la liste de sensibilité n'est pas spécifiée, elle est déduite de la présence éventuelle de nom de signaux dans l'expression booléenne.

	<b>Université de Marne-La-Vallée</b>		
	Initiation au langage VHDL		
		Date: 13 juin 2008	Page : 36/57

Exemple code 1

**wait** ; -- attendre à jamais

Exemple code 2

**wait for** 200 ns – attendre 200 ns

Exemple code 3

--expressions strictement équivalentes

**wait on h until** h = '0'; -- attendre un transition sur le signal "h" vers l'état '0'

**wait until** h = '0';

Exemple code 4

**wait on h until** nClear='1'; -- attendre tant que un événement sur le signal "h" ne coïncide pas avec une valeur sur le signal nClear égale à '1'

Un process dont la liste de sensibilité (signal1, signal2...) est non vide peut toujours être exprimé par un process équivalent dont la liste de sensibilité est vide mais se terminant par l'instruction **wait on** signal1, signal2....

Exemple code 5

-- le process suivant est équivalent au process de description comportemental d'une bascule D décrite au paragraphe "Configuration de composants".

pBascD : **process**

**begin**

**if** (h'event **and** h = '1') **then**

    q <= d;

**end if**;

**wait on** h;

**end process** pBascD;

Attention : les outils de synthèse ne supportent pas cette forme de description. La description RTL suivante est préconisée pour décrire une bascule D en utilisant l'instruction **wait** :

pBasc : **process**

**begin**

**wait until** (h'event **and** h = '1');

    q <= d;

**end process** pBasc;

*Instruction "assert"*

L'instruction "assert" vérifie que la condition spécifiée est vraie sinon elle reporte une erreur.

	Université de Marne-La-Vallée		
	Initiation au langage VHDL		
		Date: 13 juin 2008	Page : 37/57

**Assert** condition [**report** chaineDeCaractères] [**severity** niveauDeSévérité] ;

Le niveau de sévérité est défini dans le package standard par le type SEVERITY\_LEVEL :

- **type** SEVERITY\_LEVEL is (NOTE, WARNING, ERROR, FAILURE);

Le niveau FAILURE provoque l'arrêt du simulateur. Par défaut, si les champs **report** et **severity** ne sont pas spécifiés, la chaîne de caractères rapportée est "assertion violation", le niveau de sévérité est "error".

Exemple code

**assert** h='0' **report** "il y a un problème sévère" **severity** error;

*Instruction "report"*

L'instruction **report** affiche un message.

**report** chaineDeCaractères [**severity** niveauDeSévérité] ;

Le niveau de sévérité est de type SEVERITY\_LEVEL et par défaut si le champ **severity** n'est pas spécifié, il prend la valeur NOTE.

Exemple code

**report** "Setup or Hold violation; outputs driven to 'X'"  
**severity** WARNING;

*Instruction "affectation de signal"*

nomSignal <= valeur [after valeurTemps] ;

Exemple code

Sig <= '1' after 30 ns;

*Instruction "affectation de variable"*

L'instruction d'affectation d'une variable remplace la valeur courante de la variable par une nouvelle valeur spécifiée par une expression. La variable et la valeur de remplacement doivent être

	<b>Université de Marne-La-Vallée</b>		
	Initiation au langage VHDL		
		Date: 13 juin 2008	Page : 38/57

du même type.

nomVariable := expression ;

*Instruction "appel de procédure"*

Se référer au paragraphe 3.4.

*Instruction "if"*

```

if condition then
    listeInstructionsSéquentielles
[elsif condition then
    listeInstructionsSéquentielles]
....
[elsif condition then
    listeInstructionsSéquentielles]
[else
    listeInstructionsSéquentielles]
end if

```

*Instruction "case"*

```

case expression is
    when choix =>
        listeInstructionsSéquentielles
    ...
    when choix =>
        listeInstructionsSéquentielles
    when others =>
        listeInstructionsSéquentielles
end case;

```

	Université de Marne-La-Vallée		
	Initiation au langage VHDL		
		Date: 13 juin 2008	Page : 39/57

*Instruction "loop"*

```

While condition Loop
    listeInstructionsSéquentielles
end loop ;
ou
for identifiantVariable in gammeDiscrete loop
    listeInstructionsSéquentielles
end loop ;

```

*Instruction "next"*

```

Next [when condition];

```

	Université de Marne-La-Vallée		
	Initiation au langage VHDL		
		Date: 13 juin 2008	Page : 40/57

Exemple code

```

test : process
variable i, k : integer;
variable l : line;
begin
    for i in 2 downto 0 loop
        for k in 2 downto 0 loop
            next when (k=1);
            write (l, STRING'(" i = "));
            write (l, i);
            write (l, STRING'(" k = "));
            write (l, k);
            writeLine (output, l);
        end loop;
    end loop;
wait;
end process test;

```

Résultats sortie simulation :

```

# i = 2 k = 2
# i = 2 k = 0
# i = 1 k = 2
# i = 1 k = 0
# i = 0 k = 2
# i = 0 k = 0

```

*Instruction "exit"*

<b>exit</b> [when condition];
-------------------------------

*Programme précédent*

Ligne : **next when** (k=1); Remplacée par : **exit when** (k=1);

Résultats sortie simulation :

```

# i = 2 k = 2
# i = 1 k = 2
# i = 0 k = 2

```

*Instruction "return"*

<b>Return</b> [expression] ;
------------------------------

*Instruction "null"*



	<b>Université de Marne-La-Vallée</b>		
	Initiation au langage VHDL		
		Date: 13 juin 2008	Page : 41/57

**Null ;**

#### 4.1.2. Instructions concurrentes

##### *Instruction process*

```
[nomProcess] Process [listeDeSensibilite] [is]
    partieDéclarative
begin
    listeInstructionsSéquentielles
end process [nomProcess] ;
```

La partie déclarative du process peut notamment déclarer des variables, des sous-programmes, des types, des sous-types, des constantes, des fichiers. Elle peut aussi décrire le corps d'un sous-programme. Elle ne peut pas déclarer de signaux. Le corps du process ne contient que des instructions séquentielles.

##### *Instruction concurrente d'appel de procédure*

La définition est identique à celle décrite au paragraphe 3.4.1. On distingue un appel de procédure concurrent, d'un appel de procédure séquentielle par sa location dans la description, cet appel est situé au même niveau que l'instruction process. Une instruction concurrente d'appel de procédure a toujours un process équivalent dont la liste de sensibilité éventuelle est la liste des signaux déclarés en mode **in** des arguments de la procédure.

##### Exemple code

*CheckTiming (tPLH, tPHL, Clk, D, Q); -- A concurrent procedure call statement*

**Process** *-- The equivalent process.*

```
begin
    CheckTiming (tPLH, tPHL, Clk, D, Q);
    wait on Clk, D, Q;
end process;
```

##### *Instruction concurrente d'assignement de signaux*

On distingue une instruction concurrente d'assignement de signaux d'une instruction séquentielle par sa location dans la description, cette instruction est située au même niveau que l'instruction process. Une instruction concurrente d'assignement de signaux a toujours un process équivalent dont la liste de sensibilité est la liste des signaux apparaissant à droite de l'instruction d'affectation

	Université de Marne-La-Vallée		
	Initiation au langage VHDL		
		Date: 13 juin 2008	Page : 42/57

### Exemple code

```

architecture rtlNoProcess of multiplexeur is
begin
    with cmd select
        q    <=  bus0 when "00",
                bus1 when "01",
                bus2 when "10",
                bus3 when "11",
                bus0 when others;
end rtlNoProcess;

```

*Instruction d'instanciation de composants*

Se référer au Paragraphe 3.1.3.

## 4.2. Les opérateurs

*Opérateurs logiques*

**and** | **or** | **nand** | **nor** | **xor** | **xnor**

*Opérateurs relationnels*

= | /= | < | <= | > | >=

*Opérateurs de décalage*

**sll** | **srl** | **sla** | **sra** | **rol** | **ror**

*Opérateurs d'addition*

+ | - | &

*Signes*

+ | -

*Opérateurs de multiplication*

\* | / | **mod** | **rem**

*Opérateurs divers*

\*\* | **abs** | **not**

## 4.3. Les descriptions

Les trois types de description peuvent coexister pour modéliser le fonctionnement d'un système complet.

### 4.3.1. Descriptions comportementales

Une description comportementale décrit le traitement des données à l'aide de process dont le corps est constitué d'instructions séquentielles. Les algorithmes de traitement peuvent être complexes et si

	<b>Université de Marne-La-Vallée</b>		
	Initiation au langage VHDL	Date: 13 juin 2008	Page : 43/57

la description doit être synthétisable, une attention particulière doit être portée sur la compatibilité RTL du code.

#### 4.3.2. Descriptions "flots de données"

Une description "flot de données" est constituée d'instructions concurrentes d'assignement de signaux. La compatibilité RTL du code est facilement analysable. Par contre, la complexité du système décrit est limitée.

#### 4.3.3. Descriptions structurelles

Une description structurelle décrit un système sous forme d'une liste de primitives interconnectées entre elles. Si toutes les primitives sont synthétisables, alors le code est compatible RTL.

	Université de Marne-La-Vallée		
	Initiation au langage VHDL		
		Date: 13 juin 2008	Page : 44/57

Exemple code

```

library ieee;
use ieee.std_logic_1164.all;

entity or_2 is
  port (
    e          : in  std_logic_vector (1 downTo 0);
    s          : out std_logic
  );
end or_2;

-- Description flots de données
architecture rtlDataFlow of or_2 is
begin
  s <= e(1) or e(0);
end architecture rtlDataFlow;

library ieee;
use ieee.std_logic_1164.all;

entity or_8 is
  port (
    e          : in  std_logic_vector (7 downTo 0);
    s          : out std_logic
  );
end or_8;

-- Description comportementale
architecture rtlBehavior of or_8 is
begin
  pOr : process (e)
    variable i : integer;
    variable j : std_logic;
  begin
    j := '0';
    for i in 7 downTo 0 loop
      if (e(i) = '1') then
        j := '1';
      end if;
    end loop;
    s <= j;
  end process pOr;
end architecture rtlBehavior;

-- Description flots de données
architecture rtlDataFlow of or_8 is
begin

```

	Université de Marne-La-Vallée		
	Initiation au langage VHDL		
		Date: 13 juin 2008	Page : 45/57

```

s      <= e(7) or e(6) or e(5) or e(4) or e(3) or e(2) or e(1) or e(0);
end architecture rtlDataFlow;

```

-- Description structurelle

```

architecture rtlStruct of or_8 is
signal sLevel0      : std_logic_vector (3 downto 0);
signal sLevel1      : std_logic_vector (1 downto 0);
component c_or_2 is
port (
e          : in   std_logic_vector (1 downto 0);
s          : out  std_logic
);
end component c_or_2;
for all : c_or_2
use entity work.or_2 (rtlDataFlow);
begin
or_2_1 : c_or_2
port map (e=>e(1 downto 0), s=>sLevel0(0));
or_2_2 : c_or_2
port map (e=>e(3 downto 2), s=>sLevel0(1));
or_2_3 : c_or_2
port map (e=>e(5 downto 4), s=>sLevel0(2));
or_2_4 : c_or_2
port map (e=>e(7 downto 6), s=>sLevel0(3));
or_2_5 : c_or_2
port map (e=>sLevel0(1 downto 0), s=>sLevel1(0));
or_2_6 : c_or_2
port map (e=>sLevel0(3 downto 2), s=>sLevel1(1));
or_2_7 : c_or_2
port map (e=>sLevel1, s=>s);
end architecture rtlStruct;

```

#### 4.3.4. Les machines d'état

La description d'une machine d'états est couramment rencontrée dans les unités de conception VHDL et particulièrement pour les systèmes synchrones.

	<b>Université de Marne-La-Vallée</b>	
	Initiation au langage VHDL	
		Date: 13 juin 2008
		Page : 46/57

### Exemple code

```

-- exemple machine d'états
-- feux tricolore, par défaut au rouge, démarre sur ordre start
-- cycle sur les états "rouge", vert" et orange.
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity machineEtats is
  port(
    reset      : in  std_logic;
    clk        : in  std_logic;
    start      : in  std_logic;
    cmdVert    : out std_logic;
    cmdOrange  : out std_logic;
    cmdRouge   : out std_logic
  );
end machineEtats;

architecture rtl of machineEtats is

  -- State Machine : feux tricolore
  type feux_state_type is (
    feuxInit,
    feuxVert,
    feuxOrange,
    feuxRouge
  );
  signal feuxState      : feux_state_type;
  signal compteur      : std_logic_vector (3 downTo 0);
  -- temps d'attente dans chaque état exprimé en nombre de coups d'horloge
  constant dureeVert    : std_logic_vector (3 downTo 0) := "1100";
  constant dureeOrange  : std_logic_vector (3 downTo 0) := "0010";
  constant dureeRouge   : std_logic_vector (3 downTo 0) := "1110";

begin
  -- link arbiter
  controleFeux : process (clk, reset)
  begin

    if (reset = '1') then
      feuxState <= feuxInit;
      compteur  <= (others => '0');
      cmdVert   <= '0';
      cmdOrange <= '0';
      cmdRouge  <= '1';
    
```

**elsif** (*clk'event and clk='1'*) **then**

-- State Machine : feux tricolore

**case** *feuxState* **is**

**when** *feuxInit* =>

-- rouge allumé par défaut

*cmdVert* <= '0';

*cmdOrange* <= '0';

*cmdRouge* <= '1';

**if** (*start* = '1') **then**

*compteur* <= *dureeRouge*;

*feuxState* <= *feuxRouge*;

**end if**;

**when** *feuxRouge* =>

*cmdVert* <= '0';

*cmdOrange* <= '0';

*cmdRouge* <= '1';

**if** (*compteur* = "0000") **then**

*compteur* <= *dureeVert*;

*feuxState* <= *feuxVert*;

**else**

*compteur* <= *compteur* - '1';

*feuxState* <= *feuxRouge*;

**end if**;

**when** *feuxVert* =>

*cmdVert* <= '1';

*cmdOrange* <= '0';

*cmdRouge* <= '0';

**if** (*compteur* = "0000") **then**

*compteur* <= *dureeOrange*;

*feuxState* <= *feuxOrange*;

**else**

*compteur* <= *compteur* - '1';

*feuxState* <= *feuxVert*;

**end if**;

**when** *feuxOrange* =>

*cmdVert* <= '0';

*cmdOrange* <= '1';

*cmdRouge* <= '0';

**if** (*compteur* = "0000") **then**

*compteur* <= *dureeRouge*;

*feuxState* <= *feuxRouge*;

**else**

	<b>Université de Marne-La-Vallée</b>		
	Initiation au langage VHDL		
		Date: 13 juin 2008	Page : 48/57

```

                                compteur    <= compteur - '1';
                                feuxState    <= feuxOrange;
                                end if;

                                when others =>

                                    feuxState    <= feuxInit;

                                end case;
                                end if;
                                end process controleFeux;
end rtl;

--Banc de test
library ieee;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.ALL;

entity machineEtatsTb_vhd is
end machineEtatsTb_vhd;

architecture behavior of machineEtatsTb_vhd is

    -- Component Declaration for the Unit Under Test (UUT)
    component machineEtats
    port(
        reset          : in std_logic;
        clk            : in std_logic;
        start          : in std_logic;
        cmdVert        : out std_logic;
        cmdOrange      : out std_logic;
        cmdRouge       : out std_logic
    );
    end component;
    --Inputs
    signal reset      : std_logic := '0';
    signal clk        : std_logic := '0';
    signal start      : std_logic := '0';
    --Outputs
    signal cmdVert    : std_logic;
    signal cmdOrange  : std_logic;
    signal cmdRouge   : std_logic;
begin
    -- Instantiate the Unit Under Test (UUT)
    uut: machineEtats port map(
        reset => reset,

```



	Université de Marne-La-Vallée		
	Initiation au langage VHDL		
		Date: 13 juin 2008	Page : 49/57

```

    clk => clk,
    start => start,
    cmdVert => cmdVert,
    cmdOrange => cmdOrange,
    cmdRouge => cmdRouge
);
tb : process
begin
    -- Wait 100 ns for global reset to finish
    wait for 100 ns;

    -- generate reset
    reset <= '1', '0' after 200 ns;

    wait for 250 ns;

    -- generate start
    start <= '1', '0' after 200 ns;

    wait; -- will wait forever
end process;

horloge : process
begin
    clk <= not clk after 100 ns;
    wait on clk;
end process;
end;

```

### - References -

1. IEEE Computer Society, "1076 IEEE Standard VHDL Language Reference Manual", IEEE Std 1076-2002

	<b>Université de Marne-La-Vallée</b>		
	Initiation au langage VHDL		
		Date: 13 juin 2008	Page : 50/57

# Annexes

	Université de Marne-La-Vallée		
	Initiation au langage VHDL		
		Date: 13 juin 2008	Page : 51/57

Exemple 1

```
library ieee;
use ieee.std_logic_1164.all;
```

```
entity and_3 is
  port(
    i1      : in  std_logic;
    i2      : in  std_logic;
    i3      : in  std_logic;
    s       : out std_logic
  );
end and_3;
```

```
architecture rtlBehav of and_3 is
begin
  pAnd : process (i1, i2, i3)
  begin
    if (i1='1' and i2='1' and i3='1') then
      s    <= '1';
    else
      s    <= '0';
    end if;
  end process pAnd;
end architecture rtlBehav;
```

```
architecture rtlDataFlow of and_3 is
begin
  s    <= i1 and i2 and i3;
end architecture rtlDataFlow;
```

Exemple 2

```
-- Latch
library ieee;
use ieee.std_logic_1164.all;
```

```
entity latch is
  port (
    rst, d, ena : in std_logic;
    q           : out std_logic
  );
end entity latch ;
```

```
architecture rtlBehavior of latch is
begin
  pLatch : process (rst, ena, d)
  begin
    if (rst = '1') then
```

	Université de Marne-La-Vallée		
	Initiation au langage VHDL		
		Date: 13 juin 2008	Page : 52/57

```

        q    <= '0';
    elsif (ena = '1') then
        q    <= d;
    end if;
end process pLatch;
end architecture rtlBehavior;

```

### Exemple 3

-- bascule JK

**library** ieee;

**use** ieee.std\_logic\_1164.all;

**entity** basculeJK **is**

**port**(

nCl : **in** std\_logic;

clk : **in** std\_logic;

J : **in** std\_logic;

K : **in** std\_logic;

q : **out** std\_logic

);

**end** basculeJK;

**architecture** rtl **of** basculeJK **is**

**signal** i\_q : std\_logic;

**begin**

q <= i\_q;

pBascJK : **process** (clk, nCl)

**begin**

**if** (nCl = '0') **then**

i\_q <= '0';

**elsif** (clk'event and clk = '1') **then**

**if** (j='0' and k='1') **then**

i\_q <= '0';

**elsif** (j='1' and k='0') **then**

i\_q <= '1';

**elsif** (j='1' and k='1') **then**

i\_q <= not i\_q;

**end if;**

**end if;**

**end process** pBascJK;

**end** rtl;

### Exemple 4

--registre à décalage

**library** ieee;

**use** ieee.std\_logic\_1164.all;

	<b>Université de Marne-La-Vallée</b>		
	Initiation au langage VHDL		
		Date: 13 juin 2008	Page : 53/57

```

entity regDecalBeh is
  port (
    rst, h, d      : in std_logic;
    q              : out std_logic
  );
end regDecalBeh;

architecture rtlBehavior of regDecalBeh is
signal sig : std_logic_vector (3 downTo 0):="0010";
begin
  pRegDecal : process (rst, h)
    variable i : natural range 1 to 3;
    begin

    if (rst = '1') then
      sig  <= (others => '0');
      q    <= '0';
    elsif (h'event and h='1') then
      for i in 3 downTo 1 loop
        sig (i) <= sig (i-1);
      end loop;
      sig (0) <= d;
      q      <= sig (3);
    end if;
    end process pRegDecal;

end architecture rtlBehavior;

```

### Exemple 5

--registre à décalage paramétrable

**library** ieee;

**use** ieee.std\_logic\_1164.all;

```

entity regDecalParam is
  generic (n : integer range 1 to 32 := 8);
  port (
    rst, h, d      : in std_logic;
    q              : out std_logic
  );
end regDecalParam;

architecture rtlBehavior of regDecalParam is
signal sig : std_logic_vector (n-1 downTo 0);
begin
  pRegDecal : process (rst, h)
    variable i : integer range 1 to 32 ;
    begin

```

	Université de Marne-La-Vallée		
	Initiation au langage VHDL		
		Date: 13 juin 2008	Page : 54/57

```

if (rst = '1') then
    sig    <= (others => '0');
    q      <= '0';
elsif (h'event and h='1') then
    for i in n-2 downto 1 loop
        sig (i) <= sig (i-1);
    end loop;
    sig (0) <= d;
    q      <= sig (n-2);
end if;
end process pRegDecal;

```

**end architecture** rtlBehavior;

#### Exemple 6

```

library ieee;
use ieee.std_logic_1164.all;

```

**entity** multiplexeur **is**

```

port(
    bus0 : in    std_logic_vector (7 downto 0);
    bus1 : in    std_logic_vector (7 downto 0);
    bus2 : in    std_logic_vector (7 downto 0);
    bus3 : in    std_logic_vector (7 downto 0);
    cmd  : in    std_logic_vector (1 downto 0);
    q    : out   std_logic_vector (7 downto 0)

```

);

**end** multiplexeur;

**architecture** rtl **of** multiplexeur **is**

**begin**

```

    pMux : process (cmd, bus0, bus1, bus2, bus3)

```

**begin**

```

    case cmd is

```

```

        when "00" =>
            q    <= bus0;

```

```

        when "01" =>
            q    <= bus1;

```

```

        when "10" =>
            q    <= bus2;

```

```

        when "11" =>
            q    <= bus3;

```

```

        when others =>
            q    <= bus0;

```

```

    end case;

```

```

end process pMux;
end rtl;

```

```

--paquetage ieee standard
package STANDARD is

```

```

-- predefined enumeration types:

```

```

type BOOLEAN is (FALSE, TRUE);

```

```

type BIT is ('0', '1');

```

```

type CHARACTER is (
    NUL, SOH, STX, ETX, EOT, ENQ, ACK, BEL,
    BS, HT, LF, VT, FF, CR, SO, SI,
    DLE, DC1, DC2, DC3, DC4, NAK, SYN, ETB,
    CAN, EM, SUB, ESC, FSP, GSP, RSP, USP,

```

```

    ' ', '!', '"', '#', '$', '%', '&', "'",
    '(', ')', '*', '+', ',', '-', '.', '/',
    '0', '1', '2', '3', '4', '5', '6', '7',
    '8', '9', ':', ';', '<', '=', '>', '?',

```

```

    '@', 'A', 'B', 'C', 'D', 'E', 'F', 'G',
    'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O',
    'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W',
    'X', 'Y', 'Z', '[', '\', ']', '^', '_'

```

```

    '`', 'a', 'b', 'c', 'd', 'e', 'f', 'g',
    'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o',
    'p', 'q', 'r', 's', 't', 'u', 'v', 'w',
    'x', 'y', 'z', '{', '|', '}', '~', DEL,

```

```

    C128, C129, C130, C131, C132, C133, C134, C135,
    C136, C137, C138, C139, C140, C141, C142, C143,
    C144, C145, C146, C147, C148, C149, C150, C151,
    C152, C153, C154, C155, C156, C157, C158, C159,

```

```

    ' ', '¡', '¢', '£', '¤', '¥', '¦', '§',
    '¨', '©', 'ª', «, ¬, ®, ¯,
    °, ±, ², ³, ´, µ, ¶, ¸,
    ¹, º, »', ¼, ½, ¾, ¿,
    À, Á, Â, Ã, Ä, Å, Æ, Ç,
    È, É, Ê, Ë, Ì, Í, Î, Ï,
    Ð, Ñ, Ò, Ó, Ô, Õ, Ö, ×,
    Ø, Ù, Ú, Û, Ü, Ý, Þ, ß,
    à, á, â, ã, ä, å, æ, ç,

```

	Université de Marne-La-Vallée		
	Initiation au langage VHDL		
		Date: 13 juin 2008	Page : 56/57

```
'è', 'é', 'ê', 'ë', 'ì', 'í', 'î', 'ï',
'ð', 'ñ', 'ò', 'ó', 'ô', 'õ', '÷',
'ø', 'ù', 'ú', 'û', 'ü', 'ý', 'þ', 'ÿ' );
```

```
type SEVERITY_LEVEL is (NOTE, WARNING, ERROR, FAILURE);
```

```
type FILE_OPEN_KIND is (READ_MODE, WRITE_MODE, APPEND_MODE);
```

```
type FILE_OPEN_STATUS is (OPEN_OK, STATUS_ERROR, NAME_ERROR,
MODE_ERROR);
```

```
-- predefined numeric types:
```

```
type INTEGER is range -2147483647 to 2147483647;
```

```
type REAL is range -1.7014111e+308 to 1.7014111e+308;
```

```
-- predefined type TIME:
```

```
type TIME is range -2147483647 to 2147483647
```

```
-- this declaration is for the convenience of the parser. Internally
```

```
-- the parser treats it as if the range were:
```

```
-- range -9223372036854775807 to 9223372036854775807
```

```
units
```

```
fs; -- femtosecond
ps = 1000 fs; -- picosecond
ns = 1000 ps; -- nanosecond
us = 1000 ns; -- microsecond
ms = 1000 us; -- millisecond
sec = 1000 ms; -- second
min = 60 sec; -- minute
hr = 60 min; -- hour
```

```
end units;
```

```
subtype DELAY_LENGTH is TIME range 0 fs to TIME'HIGH;
```

```
-- function that returns the current simulation time:
```

```
function NOW return DELAY_LENGTH;
```

```
-- predefined numeric subtypes:
```

```
subtype NATURAL is INTEGER range 0 to INTEGER'HIGH;
```

```
subtype POSITIVE is INTEGER range 1 to INTEGER'HIGH;
```

```
-- predefined array types:
```



	Université de Marne-La-Vallée		
	Initiation au langage VHDL		
		Date: 13 juin 2008	Page : 57/57

**type** *STRING* **is** array (*POSITIVE range* <>) **of** *CHARACTER*;

**type** *BIT\_VECTOR* **is** array (*NATURAL range* <>) **of** *BIT*;

**attribute** *FOREIGN*: *STRING*;

**end** *STANDARD*;