Chapitre 4.

Programmation VHDL

I. Introduction

VHDL est un langage de description de matériel, destiné à représenter le comportement ainsi que l'architecture d'un système numérique.

L' acronym VHDL signifies Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (HDL).

Ce langage a trouvé ses premières applications dans la modélisation et la simulation de circuits numériques, on l'a ensuite étendu en lui rajoutant des extensions pour permettre la conception et la synthèse des circuits logiques programmables (P.L.D. Programmable Logic Device) ou FPGA (Field Programmable Gate Array).

Donc Le VHDL ayant une double fonction (simulation et synthèse), une partie seulement du VHDL est synthétisable, l'autre existant uniquement pour faciliter la simulation (écriture de modèles comportementaux et de test **benches).**

a. Les avantages du langage VHDL

- La portabilité.
- La conception de haut niveau.
- La possibilité de décrire des systèmes très complexes en quelques lignes de code.
- Traduit en schéma de portes logiques.

b. Environnement du VHDL

- plateforme : PC/Station
  - Éditeurs de texte.
  - Compilateur.
  - Synthèse.
  - Simulateurs.
  - Interfaces graphiques pour le résulta.
- FPGA development board (ex Altera DE2 et DE1)

Remarque: **Beaucoup de logiciels prennent en charge toutes ces fonctionnalités : Xillinx ISE, Altera Quartus, Lattice ISP Lever, Altium Designer, etc...**

c. **Historique**

Dans les années 80, et dans le cadre de l'initiative de développement des circuits intégrés à très haute vitesse VHSIC, le département de la défense des Etats-Unis lance un appel d'offre pour un langage de description de matériel unique qui permettrait de décrire tous les systèmes électroniques utilisés.

C'est en 1983, que d'importante société telle qu'IBM et Texas instruments se sont investies dans ce projet. En 1985 le développement de la première version officielle du VHDL(version 7.2), en 1986 VHDL est donné à IEEE (**I**nstitut of **E**lectrical and **E**lectronics **E**ngineers) pour la standardisation et la norme VHDL 1076 a été approuvée le 10 décembre 1987 par l'IEEE sous la référence IEEE1076.

En 1993, IEEE définit le standard IEEE 1164 normalisant la représentation des signaux logiques multivaleurs. La dernière évolution de la norme est la norme IEEE 1076-2001.

II- Structure d'une description VHDL simple

Un opérateur élémentaire, un circuit intégré, une carte électronique ou un système complet est complètement défini par des signaux d'entrées et de sorties et par fonction réalisée de façon interne. Ce double aspect - signaux de communications et fonction- se retrouve à tous les niveaux de la hiérarchie d'une application. L'élément essentiel de toute description en VHDL, nommé ***design entity*** dans le langage, est formé par le couple ***entité-architecture***, qui décrit l'apparence externe d'une unité de conception et son fonctionnement interne.

- ***L'entité:*** décrit l'interface externe du circuit : signaux d'entrées et de sorties.
- ***L'architecture*** : décrit le fonctionnement interne du circuit.

Une même entité peut avoir plusieurs architecture.

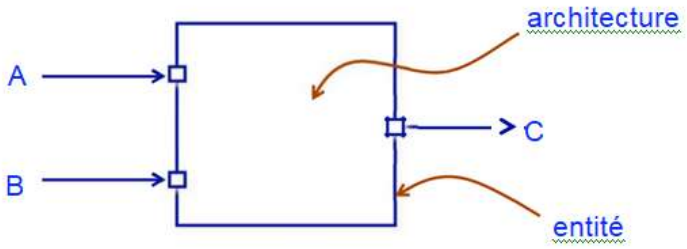


Fig. 1: Entité et architecture

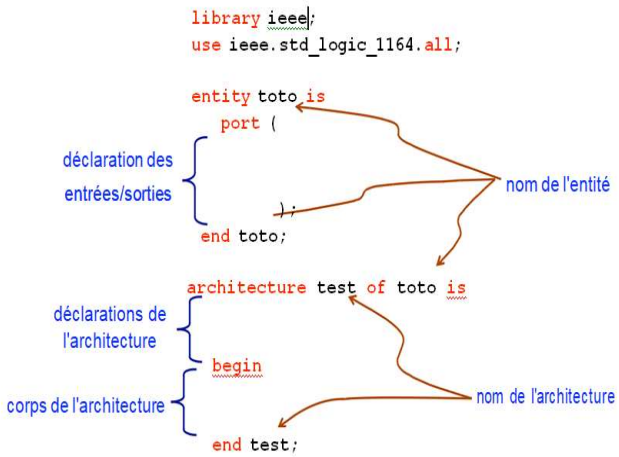


Fig. 2: Structure d'un programme VHDL

II.1- Déclaration des bibliothèques

Toute description VHDL utilisée pour la synthèse a besoin de bibliothèques. L'**IEEE** (**I**nstitut of **E**lectrical and **E**lectronics **E**ngineers) les a normalisées et plus particulièrement la bibliothèque **IEEE1164** . Elles contiennent les définitions des types de signaux électroniques, des fonctions et sous programmes permettant de réaliser des opérations arithmétiques et logiques:

- **Library ieee;** Déclaration standard.
- **Use ieee.std_logic_1164.all;** Pour définir les types std_logic ('X','U','L','H','0','1')
- **Use ieee.std_logic_arith.all;** Pour utiliser les opérateurs arithmétiques '+','-','*','/' pour les types std_logic
- **Use ieee.std_logic_signed.all;** Le type std_logic avec valeur signée.
- **Use ieee.std_logic_usigned.all;** Le type std_logic avec valeur non signée.
- **Use ieee.numeric_std.all;** Utilisation des valeurs décimales pour le type std_logic.
- **Use ieee.std_logic_textio.all;** Utilisation des valeurs ASCII pour le type std_logic.
- **Use ieee.numeric_bit.all;** Utilisation des valeurs décimales pour le type std_logic.
- **Use ieee.math_real.all;**
- **Use ieee.math_complex.all;**

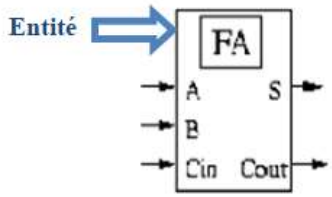
Note :

IEEE 1164 : référence de multivaleur système publiée par ieee dans l'année 1993

IEEE 1076 : référence de langage VHDL publiée par ieee dans l'année 1987

1. La directive ***Use*** permet de sélectionner les bibliothèques à utiliser
2. La directive ***All*** pour utiliser tu les applications
3. La différence entre **Signed et Unsigned (**Un vecteur représente alors un nombre signé ou non signé**).**

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity FA is
port (
a, b, cin : in std_logic;
s, cout : out std_logic
);
end FA;
```



II.2- Déclaration de l'entité

La déclaration d'entité décrit l'interface entre le monde extérieur et une unité de conception: signaux d'entrées et de sorties et, éventuellement, paramètres génériques (constantes dont la valeur est fixée par l'environnement de l'unité considérée). la déclaration d'entité peut également contenir des instructions concurrentes ***passives***, c'est à dire qui ne contiennent aucune affectation de signaux ; de telles instructions ne génèrent pas de circuits lors du processus de synthèse.

Une même entité peut être associer à plusieurs architectures différentes. Elle décrit alors une classe d'unités de conception qui présentent au monde extérieur le même aspect, avec des fonctionnements internes différents

L'entité précise :

* le nom du circuit.
* Les ports d'entrée-sortie :
  - Leurs noms.
  - Leurs directions (in, out, inout,...)
  - Leurs types (bit, bit_vector, integer, std_logic,...)
* Les paramètres éventuels pour les modèles génériques.

Fig .3. Exemple de syntaxe d'une déclaration d'entité

Ports: les signaux d'interface d'une entité constituent dans la terminologie VHDL un ***port***. chaque signal du port doit posséder un nom, un mode et un type.

a - Le nom du signal : le nom de chaque signal est choisi par l'utilisateur est connu à l'intérieur de toutes les architectures qui font référence à l'entité correspondante. il est

constitué par une chaine de caractères alphanumériques qui commence par une lettre et qui peut contenir le caractère souligné(_). VHDL ne distingue pas les majuscules des miniscules AmI et aMi représentent donc le même objet.

b- Le mode du signal

le mode précise le sens du signal :

- **In** : pour un signal en entrée.
- **Out** : pour un signal en sortie.
- **inout** : pour un signal en entrée sortie
- **buffer** : pour un signal en sortie mais utilisé comme entrée dans la Description

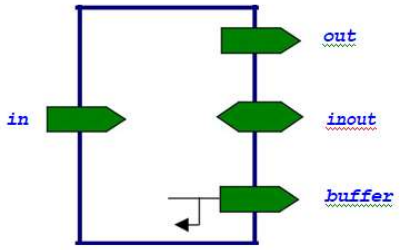


Fig. 4 les quatre modes du signal sur le langege VHDL

c- Le type du signl

Le type utilisé pour les signaux d'entrées / sorties est :

- le **std_logic** pour un signal.
- le **std_logic_vector** pour un bus composé de plusieurs signaux.

Les valeurs que peuvent prendre un signal de type ***std_logic*** et ***std_logic_vector*** sont :

- 0 : Niveau logique **bas à basse impédance** (mise à la masse via une faible impédance)
- 1 : Niveau logique **haut à basse impédance** (mise à Vcc via une faible impédance)
- Z : Niveau logique **flottant** (entrée déconnectée)
- L : Niveau logique **bas à haute impédance** (mise à la masse via une résistance de *pull-down*)
- H : Niveau logique **haut à haute impédance** (mise à Vcc via une résistance de *pull-up*)
- W : Niveau logique **inconnu à haute impédance** (pouvant être 'L', 'Z' ou 'H')
- X : Niveau logique **inconnu** (pouvant être '0', 'L', 'Z', 'H' ou '1')
- U : **Non défini**

- - : **N'importe quel niveau logique** (renvoie toujours *true* lors d'une comparaison avec les 8 autres niveaux logiques).

II.3- Déclaration de l'architecture

L'architecture décrit le fonctionnement souhaité pour un circuit ou une partie du circuit. En effet le fonctionnement d'un circuit est généralement décrit par plusieurs modules VHDL. Il faut comprendre par module le couple *ENTITE/ARCHITECTURE*. Dans le cas de simples **PLDs** on trouve souvent un seul module.

L'architecture établit à travers les instructions les relations entre les entrées et les sorties. On peut avoir un fonctionnement purement combinatoire, séquentiel voire les deux séquentiel et combinatoire.

L'architecture est divisée en deux parties: une zone déclarative et une zone d'instructions. ces instructions sont concurrentes, elles s'exécutent en parallèles. cela signifie que les instructions d'une architecture peuvent être écrites dans un ordre quelconque, le fonctionnement ne dépend pas de cet ordre.

Trois styles de description peuvent être utilisés en VHDL :

- le style "***structurel***" : interconnexion de composants, chacun d'eux étant une instance de couple entité/architecture.

- le style "***dataflow***" : correspond grosso modo à un **RTL (R**egister **T**ransfert **L**anguage) , ensemble d'instructions sur signaux qui décrivent les connexions entre portes logiques et les chargements de registres.

- le style "***comportemental***" : ensemble de processus qui expriment le comportement du système.

Une définition d'architecture a la forme suivante :

```
architecture nom_de_l_architecture of nom_de_l_entité is
        déclarations
begin
        instructions-concurrentes
end nom_de_l_architecture
```

III- Les opérateurs du langage

le tableau ci-dessous liste les opérateurs standard et le type des opérandes manipulés:

CATEGORIE	TYPE OPERANDE		SIGNIFICATION
Opérateurs logiques and nand or nor xor xnor not	boolean bit ou bit_vector__		Et Non-et Ou Non-ou Ou exclusif Non ou exclusif Non
Opérateurs relationnels = /= < <= > >=	entrée Tout type scalaire	résultat boolean	Egal Non égal Inférieur Inférieur ou égal Supérieur Supérieur ou égal
Opérateurs arithmétiques + - * / Abs ** Mod rem	Integer, real integer		+ unaire (signe +) ou addition - unaire (signe -) ou soustraction Multiplication Division Valeur absolue Exponentiation Modulo Reste
Décalages Sll Srl Rol Ror	bit_vector (amplitude : integer)		Logique gauche Logique droit Circulaire gauche Circulaire droit
Autres &	Bit, bit_vector	bit_vector	concaténation

III. Fonctionnements concurrents et séquentiels

III.1 Fonctionnement concurrent

Le comportement d'un circuit peut être décrit par un ensemble d'actions s'exécutant en parallèle. C'est pourquoi VHDL offre un jeu d'instructions dites concurrentes.

Une instruction concurrente est une instruction dont l'exécution est indépendante de son ordre d'apparition dans le code VHDL.

Par exemple, prenons le cas d'un simple latch, comme le montre la figure :

Les deux portes constituant ce latch fonctionnent en parallèle. Une description possible de ce circuit est donnée dans le code suivant (seule l'architecture est donnée).

```
architecture comportement of VERROU is
begin
  Q<= S nand NQ;
  NQ <= R nand Q;
end comportement;
```

Ces deux instructions s'exécutent en même temps. Elles sont concurrentes ; leur ordre d'écriture n'est pas significatif ; quel que soit l'ordre de ces instructions, la description reste inchangée.

III.2 Fonctionnement séquentiel

En VHDL, les instructions séquentielles ne s'utilisent qu'à l'intérieur des processus ***process.*** Un ***process*** est une partie de la description d'un circuit dans laquelle les instructions sont exécutées séquentiellement, c'est à dire les unes à la suite des autres (contrairement aux instructions concurrentes). Il permet d'effectuer des opérations sur les signaux en utilisant les instructions standard de la programmation structurée comme dans les systèmes à microprocesseurs.

Syntaxe :

```
[Nom_du_process :] process (Liste_de_sensibilité_nom_des_signaux)
                   begin
                      -- instructions du process
                   end process [Nom_du_process] ;
```

Règles de fonctionnement d'un process :

1) L'exécution d'un **process** est déclenchée par un ou des changements d'états de signaux logiques. Le nom de ces signaux est défini dans **la liste de sensibilité** lors de la déclaration du **process**.

2) Les instructions du ***process*** s'exécutent séquentiellement.

3) Les modifications apportées aux valeurs de signaux par les instructions prennent effet à la fin du processus.

L'exemple de la description suivante montre une architecture (seule) d'une bascule **D** contenant un processus qui est exécuté lors du changement d'état de l'horloge CLK.

```
architecture comportement of basc_D is
begin
   Process (CLK)
   Begin
        If ( CLK= '1') then
            Q <= D;
        End if ;
   End process ;
end comportemen
```

V. Instructions en mode concurrent

V.1 Assignation inconditionnelle

sa forme générale:

```
signal <= expression;
```

V.2. Assignation conditionnelle

l'assignation conditionnelle se présente comme suit:

```
signal <= exp1 when condition1 else
          exp2 when condition2 else
          .....
          expN-1 when conditionN-1 else expN;
```

V.3. Assignation sélective

a la syntaxe suivante:

```
with expression select
    signal <= exp1 when choix1,
              exp2 when choix2,
              ....
              expN when choixN;
```

V.4. Instanciation du composant

Consiste à utiliser un sous-ensemble décrit en VHDL comme composant dans un ensemble plus vaste

L'instanciation d'un composant se fait dans le corps de l'architecture de cette façon :

```
<nom_instance>:<nom_composant> port map (liste des connexions);
```

V.5. Instruction generate

-- structure répétitive :

```
etiquette : for variable in valeur_debut to valeur_fin generate
                instructions_concurrentes
            end generate [etiquette] ;
```

ou :

-- structure conditionnelle :

```
etiquette : if condition generate
                instructions_concurrentes
            end generate [etiquette] ;
```

VI. Instruction en mode séquentiel

VI.1. Assignation inconditionnelle de variable

forme générale

```
var := expression;
```

VI.2. Instruction *wait*

L' instruction ***wait*** suspend l'exécution d'un ***process*** jusqu'à ce qu'un événement, une condition ou une clause de temps écoulé (time out) soit vraie. Si aucune clause de réveil n'est stipulée, le processus s'arrête définitivement.

« ***wait*** » peut être utilisé des manières suivantes:

wait on [nomSignal1, nomSignal2…] ;

wait until [expressionBooléenne] ;

wait for [expressionTemps] ;