Le langage VHDL

Introduction :

Au paravant, pour décrie le fonctionnement d'un circuit électronique, les techniciens et les ingénieurs utilisaient des langages de bas niveaux tels que ABEL, PALASM, etc, ou plus simplement un outil de saisie de schémas.

Actuellement, la densité de fonctions logiques (portes et bascules) intégrée dans les PLDs (Programmable Logic Device) est telle qu'il n'est plus possible d'utiliser les outils d'hier pour développer des circuits d'aujourd'hui.

Les sociétés de développement et les ingénieurs ont voulu s'affranchir des contraintes technologiques des circuits. Ils ont donc crée des langages dits de haut niveau à savoir VHDL, VERILOG... Ils permettent au code écrit d'être portable, c'est-à-dire qu'une description écrite pour un circuit peut être facilement utilisée pour un autre circuit.

1. Définition et historique de VHDL

VHDL signifie VHSIC (Very High Speed Integrated Circuit) Hardware Description Language. C'est le langage de description de matériel pour circuits à très haute vitesse d'intégration. Le VHDL est un langage de description de matériel qui fourni un moyen de spécifier un système digital à travers différents niveaux d'abstraction. Il supporte une spécification comportementale durant les premières phases de la conception du processus et une spécification structurelle durant les dernières phases d'implémentation.

Il a été conçu dans les années 80 pour répondre aux besoins du département de la défense américaine, mais il n'a été normalisé qu'en 1987 par l'IEEE.

En 1993, une nouvelle normalisation par l'IEEE du VHDL a permis d'étendre le domaine d'utilisation du VHDL vers :

- la synthèse automatique de circuit à partir des descriptions

- la vérification des contraintes temporelles

- la preuve formelle d'équivalence de circuits

Le VHDL peut être divisé en 4 phases :

1- Analyse des besoins et phase de spécification
2- Phase de conception
3- Implémentation et phase de tests
4- Phase de fabrication industrielle

2. Pourquoi utiliser le VHDL

Le VHDL est un langage normalisé, puissant, moderne, et qui permet une excellente lisibilité, une haute modularité et une meilleure productivité des descriptions.

VHDL répond à deux objectifs :

1- Il permet d'obtenir un modèle de simulation permettant de valider une solution avant de réaliser le composant
2- Il permet la synthèse et la mise en œuvre de circuits programmable de type FPGA, CPLD

3. Conception classique de circuits numériques

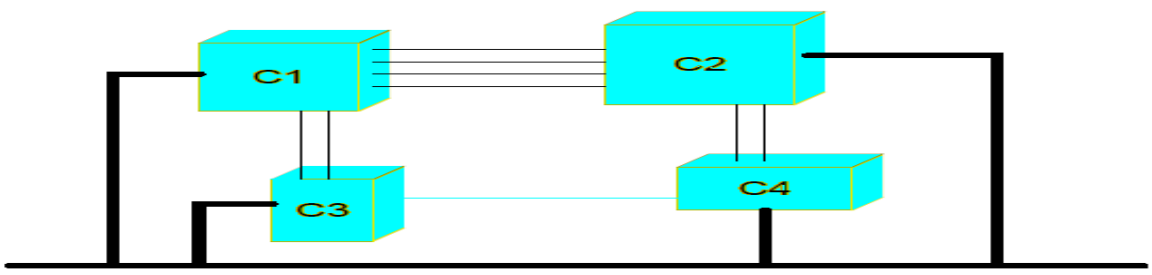
- Jusqu'à la fin des années 80, la conception de circuits était réalisée en utilisant des descriptions schématiques.
- Une description schématique est formée, en général, d'interconnexion de portes logiques dessinées à la main ou par un logiciel de CAO (ORCAD, Protel, PSPICE, VALID, Electronic Workbench, …etc).
- Le schéma décrit sous forme de portes logiques et d'autres composants électroniques peut être simulé sur ordinateur pour l'analyser avent de passer à la phase de production.
- A partir des années 90, le nombre de portes logiques dépasse les centaines de milliers et la conception manuelle devient quasiment impossible.
- Actuellement, la durée de vie d'un circuit (en terme de performance) devient très petite.
- La conception d'un circuit doit être donc très rapide, si on ne veut pas que ce circuit soit obsolète avant même sa sortie en production

4. Conception moderne de circuits numériques

Aujourd'hui, la conception de circuits complexes utilise les langages de description matérielle dits aussi HDLs (Hardware Description Languages).

Un HDL est un moyen de description d'un **système matériel.** Ce dernier est en général, un schéma mettant en œuvre :

– un certain nombre de composants et des connexions entre ces composants



Un processus de conception d'un système digital (numérique) peut généralement être divisé en quatre phases :

1. Analyse des besoins et phase de spécification,
2. Phase de conception,
3. Implémentation et phase de tests,
4. Phase de fabrication industrielle.

5. Principe d'utilisation du langage

Pour simuler ou effectuer **la synthèse logique** d'un modèle VHDL, il faut d'abord le compiler (on dit **analyser** pour le VHDL). Les résultats d'analyse sont rangés dans une bibliothèque.

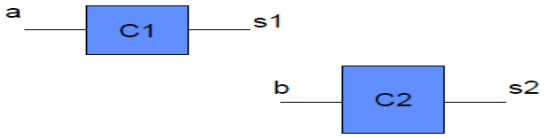
Avant analyse, il faut donc préciser au compilateur la bibliothèque dans laquelle sera rangé le résultat. Dans un programme VHDL, on peut ainsi faire référence à un objet préalablement analysé en prenant soin de préciser dans quelle bibliothèque se trouve l'objet. Une fois la compilation terminée, il faut effectuer l'édition de liens (on parle d'**élaboration** pour le VHDL). La simulation peut alors s'effectuer sur le résultat de l'élaboration (rangé dans la bibliothèque spécifiée avant élaboration).

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, …) de la bibliothèque du circuit ciblé.

6. VHDL introduit les notions suivantes

- **Fonctionnement concurrent des composants** :

Si a et b évoluent en même temps, alors les composants C1 et C2 ont un fonctionnement parallèle (concurrent).



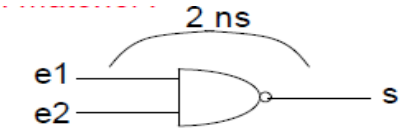
- **Notion de signal** :

Connexion des composants entre eux.

- **Notion de temps est gérée** :

Prise en compte des contraintes réel du matériel :

• temps de traversée, de calcul, …etc.



7. Les avantages de VHDL

- ✓ Indépendant du constructeur
- ✓ Indépendant de la technologie
- ✓ Indépendant de la démarche
- ✓ Indépendant du niveau de conception

} Portabilité

- ✓ **Standard IEEE**
  - Reconnu par les vendeurs d'outils CAO
  - Grand nombres de bibliothèques : d'opérateurs, de composants, de fonction…
- ✓ Développement parallèle = Travail d'équipe
  - Découpage en unités de conception
  - Développement et validation parallèle

  ➔Nécessite de bien identifier les interfaces entre les blocs fonctionnels
- ✓ Langage moderne, puissant, général :
  - Jeu d'instructions complet et très riche
  - Fort typage des données
  - Compilation séparée des entités

Inconvénients

- ➢ Description complexe
- ➢ Tout n'est pas synthétisable

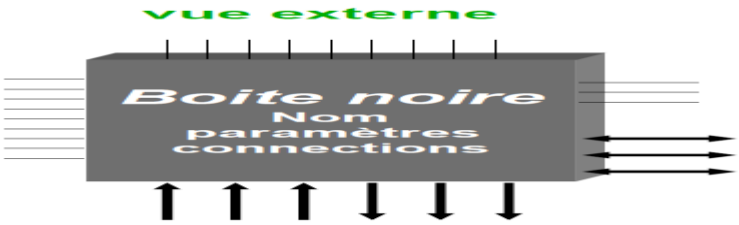
8. Unités de conception VHDL

VHDL dispose de cinq unités de conception :

1. Entité (vue externe de la boite)
2. Architecture (vue intérieur de la boite)
3. Configuration (correspondance composants entités)
4. Spécification de paquetage (décrits les types, et sous programmes)
5. Corps de paquetage (décrits le corps des sous programmes)

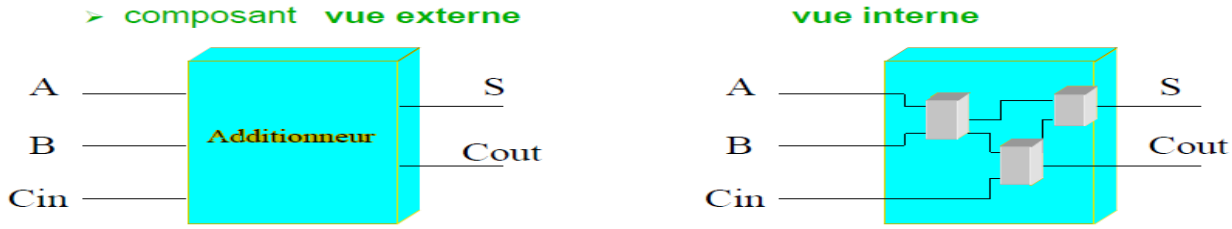
Une description **VHDL doit obligatoirement être composée de 2 parties indissociables à savoir :**

- ***L'entité (ENTITY),*** *elle définit les entrées et sorties* (vue externe de la boite)



- ***L'architecture (ARCHITECTURE),*** *elle contient les instructions VHDL permettant* de réaliser le fonctionnement attendu (vue intérieur de la boite).

Exemple d'un additionneur 1 bit



Les autres unités sont optionnelles mais quasiment indispensables pour concevoir de gros circuits nécessitant une méthode de conception efficace.

NB Une unité de conception doit être écrite dans un même fichier mais un même fichier peut contenir plusieurs unités de compilation.

9. Les éléments de base du langage VHDL

Un code VHDL est composé d'une suite d'éléments, il est insensible à la casse, écrire un mot en majuscule ou en minuscule n'a pas de différence. Une convention a été faite c'est d'écrire la première lettre d'un mot en majuscule et le reste du mot en minuscule.

9.1 Les commentaires

Les commentaires sont facultatifs lors de rédaction du code, cependant ils sont très utiles pour la bonne lisibilité du code, il est donc conseillé de les inclure.

Ils commencent par 2 tirets (- -)

```
--ceci est un commentaire
```

9.2 Syntaxe des instructions

Elles se terminent par un point virgule (;)

Y<= (not C and B) or (A and C and not D); -- est une instruction d'affectation

9.3 Les littéraux

Ce sont des valeurs explicites d'un type donné :

68 est un littérale pour le type entier

‘’Bonjour ‘’ est un littérale pour le type chaine de caractère

‘0’ est un littérale pour un bit

9.4 Les mots réservés (mot clé)

C'est des mots spécifiques à un langage. Ils sont affichés en gras ou en une autre couleur

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

9.5 Les identificateurs

Les identificateurs sont les noms qui sont choisis par le concepteur pour les différents objets du langage: noms d'entités, de signaux, de variables, de composant, de procédures …etc.

- **Identificateurs simples**

Les identificateurs simples (l'immense majorité des identificateurs utilisés) ne comprennent que les caractères alphabétiques (A à Z et a à z, sans distinguer majuscule et minuscule), les chiffres (0 à 9) et des blancs-soulignés (*underscore*, le '_').
Limitations:

- ✓ Pour de simples raisons de lisibilité, le blanc-souligné ne doit pas apparaître au début, ni à la fin, ni à côté d'un autre blanc-souligné. Deux identificateurs qui ne diffèrent que par lui sont différents.
- ✓ On ne peut pas commencer par un chiffre.
- ✓ On ne peut pas utiliser un mot-clé (mot réservé).

Exemples: TOTO, NabUCHodoNoSor, X123, A_B_C
Contre-exemples: **buffer (mot-clé)**, 2AB, A$, X__Y, Z_

- ➢ **Identificateurs étendus**

Il arrive que les limitations sur la syntaxe des identificateurs simples soient gênantes dans certains cas, en particulier quand le modèle que l'on écrit correspond à quelque objet déjà écrit dans un autre langage ou venant d'une bibliothèque qui n'a pas les mêmes conventions de nommage. Pour ces cas très particuliers (rares), il est possible de s'affranchir de toutes les limitations en encadrant l'identificateur de deux barres obliques inversées (*backslash* \). Si l'on veut mettre une barre oblique inversée dans l'identificateur, il suffit de la doubler. Dans ce cadre :

- ✓ Les majuscules et minuscules sont différenciées.
- ✓ Les blancs et blancs insécables font partie de l'identificateur.
- ✓ Tout caractère «graphique» au sens ASCII peut être mis dans l'identificateur (pas le retour à la ligne).
- ✓ On peut utiliser des mots réservés, par exemple \buffer\.
- ✓ Un identificateur étendu est toujours distinct d'un identificateur simple, même si le contenu est le même: ainsi VHDL, \VHDL\ et \vhdl\ sont 3 identificateurs distincts.

Exemples: \group\ , \123\, \a\\b\.

9.6 Les opérateurs

Les opérateurs prédéfinis en VHDL sont classiques. Ils ne portent que sur les types prédéfinis, BOOLEAN, BIT, INTEGER. Il faut donc définir une surcharge d'opérateur lorsqu'il s'agit d'effectuer des opérations sur un nouveau type.

Types d'opération	**Opérateurs**	**Notes**
Logiques	and, or, nand, nor, xor, not	
Relationnels	=,/=, <, <=, >, >=	
Arithmétiques	*, /, mod, rem	REM et MOD donne le reste de la division (A rem B) a le signe de A (A mod B) a le signe de B
Divers	**, abs, &t	** : exponentiation Abs : valeur absolue &t : concaténation

9.7. Les objets données de VHDL : Constantes, variables, signaux et ports

Avant d'introduire les types de données connus du VHDL, il importe de savoir quels sont les éléments auxquels seront assignés ces types. Le langage VHDL exploite quatre objets différents :

✓ Les constantes

Elles permettent d'apporter une plus grande lisibilité du code lors des opérations d'affectation ou de comparaison. Les constantes reçoivent leur valeur au moment de leur déclaration.

Exemple

constant max : std_logic_vector(9 downto 0) := "1111100111";

✓ Les ports

Les Ports sont les signaux d'entrées et de sorties du composant VHDL en cours de description.

✓ Les signaux

Un signal est l'image d'une entrée ou d'une sortie d'un composant logique. L'affectation de ce dernier établit un lien définitif entre un ou plusieurs autres signaux.

L'affectation d'un signal se fait à l'aide de l'opérateur **<=**

Nom_Signal <= expression ;

Exemple

```
Signal A, B, Y : Std_Logic
    Y<= A or B ;
```

L'exemple ci-dessus modélise l'affectation du signal de sortie d'une porte OU logique. La modification du signal à la prochaine itération de la simulation (retard delta).

✓ Les variables

Une variable est l'image d'un objet auquel on peut affecter, à tout moment, la valeur qu'on veut. L'affectation d'une valeur à une variable modifie immédiatement cette variable. Elle se fait à l'aide de l'opérateur **:=** comme suit :
Nom_Variable := expression ;

Exemple1 :

```
variable V : bit ;        --déclaration de la variable V de type bit
    V := '1';             --affectation de '1' à V
```

Exemple2

variable A : integer := 0 ;

9.8. Les types d'objets

Le VHDL est un langage déclaratif et fortement typé. Tout objet manipulé (variable, signal, constante, fonction, procédure, composant…) doit être déclaré et avoir un type avant sa première utilisation. Chaque objet appartient à une classe.

En résumé : le type définit le format de l'objet et la classe spécifie le comportement de celui-ci.

Le langage distingue quatre catégories de types :

1. Les types **scalaires** (entiers, réels, énumérés, physiques)
2. Les types **composés** (tableau, enregistrement)
3. Les types **pointeurs** (accès)
4. Les types **fichiers** (pour la gestion des fichiers)

- **Types entiers**

Le type entier « integer » prédéfini dans le paquetage standard STD permet de définir des nombres signés sur 32 bits compris entre -2^{31} et $2^{31}-1$.

Un sous type "subtype" permet de déclarer un type héritant des propriétés du type père. Il existe 2 sous types "subtype" associés à INTEGER : les entiers naturels et les entiers positifs. Leur déclaration dans le paquetage STD est comme suit :

subtype NATURAL is INTEGER range 0 to INTEGER 'HIGH;

subtype positive is INTEGER range 1 to INTEGER 'HIGH;

Notez que

1- range permet d'indiquer l'intervalle,

2- 'HIGH indique la plus grande valeur du type INTEGER, c'est un attribut de type.

Les types entiers servent à définir des indices de tableaux et de boucles. Pour cela, il est intéressant de les restreindre de façon à contrôler les débordements.

Par exemple, on peut déclarer :

subtype UN_A_DIX is natural range (1 to 10);

subtype DIX_A_UN is natural (10 dowto 1);

Il s'agit d'objets compatibles entre eux mais avec un ordre d'indexage différent. Il est possible d'effectuer des opérations entre ces objets. **subtype** est utilisé pour restreindre un type existant.

En revanche, si on avait déclaré :

type UN_A_DIX is natural range (1 to 10);

type DIX_A_UN is natural range (10 downto 1);

Les deux types étant indépendants, une opération entre les objets UN_A_DIX et DIX_A_UN aurait causé une erreur de compilation. **type** est utilisé pour déclarer un nouveau type.

- **Type real** : flottant compris entre -1.0E38 et 1.0E38

- **Types énumérés**

Un type énuméré est un type défini par une énumération exhaustive :

Syntaxe : type nom **is** (valeurs possibles) ;

Exemple

type COULEURS **is** (ROUGE, BLEU, VERT, ORANGE);

L'ordre de déclaration est important. Lors de l'initialisation d'un signal T de type énuméré, le signal prend la valeur T'LEFT.

Par exemple un signal de type COULEURS sera ROUGE en début de simulation.

Dans le paquetage STANDARD de la bibliothèque STD, plusieurs types énumérés sont définis :

type **BOOLEAN** is (FALSE, TRUE);

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

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

type **CHARACTER** is (NUL, SOH,STX, ETX, . . . , '0', '1', . . .);

Notez que la valeur d'un bit est équivalente à un caractère et est toujours entre quottes : '0' et '1' différents des entiers 0 et 1.

Dans le paquetage STD_LOGIC_1164 de la bibliothèque IEEE, le type STD_ULOGIC est défini par :

type STD_ULOGIC is ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-');

Il permet d'avoir 9 états significatifs de la logique. Par exemple, au démarrage les signaux sont dans un état inconnu 'U'.

- **Types physiques**

VHDL permet de définir des types physiques pour représenter une grandeur physique, comme le temps, la tension, … etc.

Un type physique est la combinaison d'un type entier et d'un système d'unités.

Le type **TIME** est le seul type physique prédéfini en VHDL dans STD :

```
type TIME is range $- to $+   -- l'intervalle dépend de la machine
units
        fs;
        ps = 1000 fs;
        ns = 1000 ps;
        µs = 1000 ns;
        Ms = 1000 µs;
        sec = 1000 Ms
        min = 60 sec;
        hr = 60 min;
end units;
```

Les simulateurs VHDL utilisent la fonction now, de la bibliothèque STD, qui retourne le temps de type TIME.

- **Types tableaux**

Les types TABLEAU ou array sont des collections d'objets de même type, indexées par des entiers ou des énumérés.

Exemples :

```
type BUS is array (0 to 31) of BIT;
type RAM is array (0 to 1024, 0 to 31) of BIT;
```

Un tableau peut avoir une taille inconnue donc non contraint, comme :

- Le type **BIT_VECTOR** de la bibliothèque STD :

  type BIT_VECTOR is array (NATURAL range <>) of BIT;

- Le type **STRING** de la bibliothèque STD. Ce type permet de définir les chaînes de caractères. Sa déclaration dans STD est comme suit :

```
type string is array (positive range <>) of character;
        -- "ceci est une chaîne de caractère"
```

La contrainte de taille est alors exprimée dans la déclaration de l'objet :

```
signal toto BIT_VECTOR (31 downto 0);
```

La valeur d'un vecteur peut être représentée dans une base différente : " 001100 " =O "14 "=X "0C " (en binaire, en Octal ou en heXadécimal) .

Il peut y avoir des tableaux de tableaux ou des tableaux à plusieurs dimensions; Les affectations diffèrent quelque peu comme illustrées dans l'exemple suivant :

```
type TAB1 is array(0 to 2) of BIT_VECTOR(7 downto 0);
        type TAB2 is array(0 to 3, 1 to 8) of BIT;
        signal A : TAB1;
        signal B : TAB2;
begin
-- tableau de tableau
        A(0)<= "01001111";
        A(2)(5) <= '1';
-- tableau à 2 dimensions
        B(3, 5) <='0';
end exemple;
```

- **Types fichiers**

Le type fichier FILE permet l'échange de données entre l'extérieur et le simulateur VHDL. Il est utilisé principalement pour créer des fichiers de tests ou TESTBENCH de modèles.

Le paquetage TEXTIO de la bibliothèque STD définit un type fichier texte TEXT et des procédures pour accéder aux lignes du fichier et aux chaînes dans la ligne. Pour l'utiliser, il est nécessaire de le déclarer au début du fichier :

```
use STD.TEXTIO.ALL;
```

Un fichier doit être en lecture ou en écriture mais pas les deux en même temps. L'exemple commenté suivant illustre 2 processus permettant respectivement de lire le fichier "entrees.dat" et d'écrire les résultats dans "sorties.dat":

```
LECTURE : process
        variable L : LINE;              --le type LINE est un pointeur
        file ENTREES : TEXT open READ_MODE is "entrees.dat"; --fichier spécifié
        variable A : BIT_VECTOR (7 downto 0);          -- variables à lire
        variable B : NATURAL range 0 to 11;
begin
        READLINE (ENTREES, L);   -- lecture d'une nouvelle ligne dans le fichier
        READ (L,A);              -- lecture dans la ligne du 1er symbole
        VA <= A;                 -- utilisation pour la simulation
        READ (L, B);             -- lecture dans la ligne du 2ème symbole
        VB <= B;                 -- utilisation pour la simulation
        wait for 20 ns;
end process LECTURE;

ECRITURE : process (S)
        variable L : LINE;
        file SORTIES : TEXT open WRITE-MODE is "sorties.dat";
begin
        WRITE (L, S);                    -- écriture de S dans la ligne
        WRITE (L, STRING'("  à t=  ")); -- écriture de texte dans la ligne
        WRITE (L, NOW);                  -- écriture du temps de simulation dans la ligne
        WRITELINE (SORTIES, L);          -- écriture de la ligne dans le fichier
end process ECRITURE;
```

- **Type enregistrement RECORD**

Ce type permet de définir un objet dont les composantes sont hétérogènes.

```
        type OPTYPE is (MOV, ADD, SUB, JMP, CALL)
        type INSTRUCTION is RECORD
        OPCODE : OPTYPE;
        ADR : BIT_VECTOR (7 downto 0);
        OP2 : BIT_VECTOR (7 downto 0);
end RECORD;
```

L'affectation d'un objet de type RECORD peut s'effectuer de différentes façons :

```
        signal INST1 : INSTRUCTION;
        signal INST2 : INSTRUCTION;
begin
INST1 <= (MOV, "00011100", X"FF");
INST2.ADR <= X"8A";
end;
```

Les records sont très utiles pour les blocs dont les ports ne sont pas figés. Les ports sont alors de type RECORD et ne changent pas. En cas de modification, il suffit de modifier le type RECORD sans avoir à modifier les entités.

- **Le type Std_Logic**

C'est le type utilisé dans la pratique dans l'industrie. Il comprend tous les états nécessaires pour modéliser le comportement des systèmes numériques.

Il comprend 9 états :
'U' état non initialisé
'X' état inconnu fort
'0' état logique 0 fort
'1' état logique 1 fort
'Z' états hauts impédance
'W' état inconnu faible
'L' état logique 0 faible
'H' état logique 1 faible
'_' état indifférent

Le type Std_Logic peut décrire par ces différents états des portes logiques 3 états, des résistances de pull up ou des sorties de circuits à collecteur ouvert.
Le type Std_Logic est définit par le paquetage normalisé IEEE.Std_Logic_1164, il est donc indispensable de déclarer celui-ci au début de son code VHDL.

NB : Les types bit et bit_vector ne peuvent prendre que deux valeurs ('0' et '1'), pour cela la norme IEEE 1164 les introduit deux autres types énumérés (STD_LOGIC et STD_LOGIC_VECTOR).

10. Notation d'agrégat

La notation d'agrégat permet de spécifier la valeur d'objets de type tableau ou enregistrement. Dans un agrégat les éléments sont spécifiés par association **positionnelle, mixte ou nommée.**

Exemple

```
Type Optype is ( ADD, SUB, MUL, DIV, BRA);
Type Tab is array ( to 5) of Optype;
Signal A;
A<=(ADD, MUL, SUB, BRA, DIV);   -- Positionnelle
```

A<=(1=>ADD, 2=> SUB, 5=>MUL, 3=>BRA, 4=>DIV); -- Nommée
A<=(ADD, 2 to 3 =>MUL, **others**=>SUB); -- mixte

Le mot clé **others** signifie tous les autres éléments

11. Les attributs en VHDL

Il s'agit de caractéristiques de types ou d'objets qu'il est possible d'utiliser dans le modèle. Ils sont représentés de cette façon:

<OBJET>'<ATTRIBUT>

Il existe des attributs sur les types, sur les objets de type tableau et sur les signaux.

Il est possible de créer ses propres attributs. Certains outils de synthèse en tirent profit pour passer des arguments de synthèse.

- **Attributs sur les types**

L'exemple suivant illustre les principaux attributs de type :

type COULEUR is (BLEU, ROUGE, VERT);

COULEUR 'LEFT renvoie BLEU

COLEUR'RIGHT renvoie VERT

COULEUR'POS (BLEU) renvoie 0

COULEUR'VAL(0) renvoie BLEU

COULEUR'SUCC(BLEU) renvoie ROUGE

COULEUR 'PRED(ROUGE) renvoie BLEU

- **Attributs sur les objets de type tableau**

Exemples :

type MOT is BIT_VECTOR (7 downto 0);

type TAB is array (4 downto 0) of MOT;

signal NOM : MOT;

signal TABLEAU : TAB;

MOT'LEFT renvoie 7;

MOT'LENGHT renvoie 8;

TABLEAU'RIGHT renvoie 0;

TABLEAU'RANGE renvoie 4 downto 0;

Ces attributs sont très utiles pour créer des indices ou pour écrire des sous-programmes manipulant des tableaux de taille variable.

- **Attributs sur les signaux**

Ils servent à indiquer les caractéristiques d'évolution temporelle des signaux.

Exemples :

CLK'EVENT renvoie un BOOLEAN indiquant si le signal CLK a changé.

CLK'DELAYED (1 ns) est un signal identique à CLK décalé de 1 ns.

- **Attributs définis par le concepteur**

Ils permettent d'associer des caractéristiques propres aux objets. Certains outils de synthèse ont leurs propres attributs pour rentrer des contraintes de synthèse dans le code.

L'exemple suivant définit le brochage de certains signaux.

Exemple :

ATTRIBUTE NUMERO_BROCHE : POSITIVE;

ATTRIBUTE NUMERO_BROCHE of ENTREE1 is 12;

ATTRIBUTE NUMERO_BROCHE of ENTREE2 is 17;

12. Unités de conception

Une unité de conception est un ensemble d'éléments VHDL avec lesquels nous allons décrire un système numérique.

Elle est constituée d'une entité (définit l'interface), une ou plusieurs architectures (définit le fonctionnement), des bibliothèques et de la configuration.

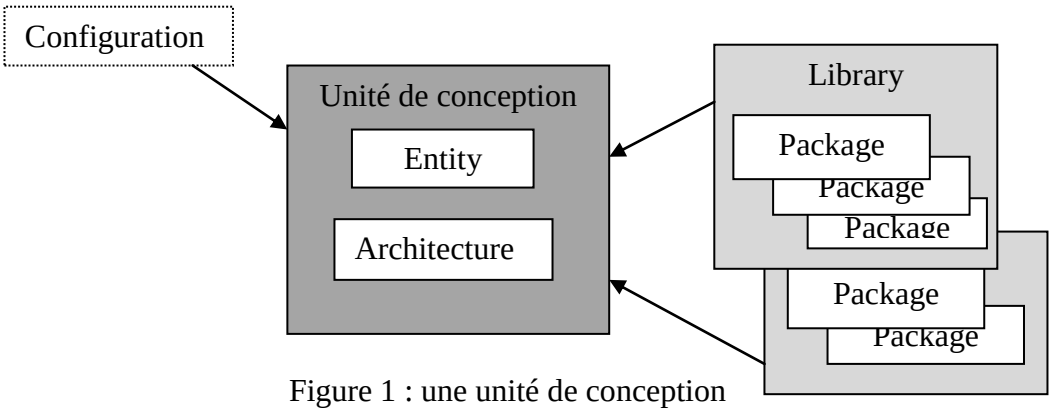


Figure 1 : une unité de conception

12.1. L'entité

L'entité définit la vue externe.

- Elle permet de définir le ***NOM de la description VHDL***
- Les ***entrées et les sorties*** utilisées. L'instruction ***port*** *spécifie :*
  - *Les N*oms de ces entrées/sorties

- Leurs Sens
- Les Types des données transportées

❖ Les paramètres éventuels pour les modèles génériques

Syntaxe de définition de l'entité

```
Entity NOM_DE_L_ENTITE is
    port (Description des signaux d'entrées /sorties ...);
end NOM_DE_L_ENTITE;
```

L'instruction port

Syntaxe ***NOM_DU_SIGNAL : sens type ;***

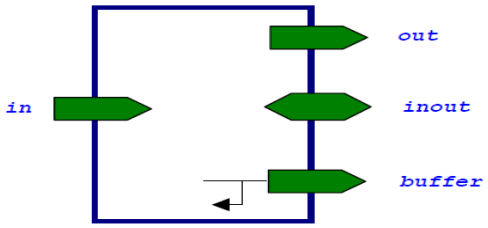
Pour chaque signal, on définit: Le ***NOM_DU_SIGNAL, le sens de ce signal et son type.***

Exemple

port (CLOCK: in std_logic;

BUS : out std_logic_vector (7 downto 0)

) ;

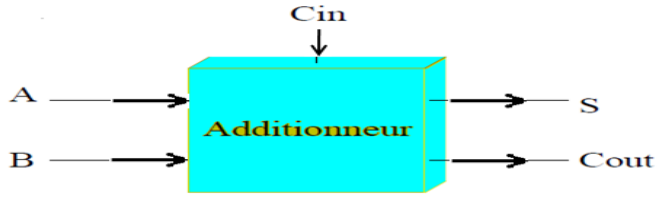


Le *SENS du signal :* le sens d'un signal peut être :

- ***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.***

Exemple : Donner l'entité correspondante à un additionneur 1bit

On sait qu'un additionneur complet à 1bit est schématisé comme suit



```
entity   Additionneur  is
    port (
        A : in bit;
        B : in bit;
        Cin : in bit;
        S : out bit;
        Cout : out bit);
end Additionneur ;
```

Ou

```
entity   Additionneur  is
    port ( A, B, Cin : in bit; S, Cout : out bit);
end Additionneur ;
```

Remarque : Après la dernière définition de signal de l'instruction **port** *il ne faut jamais* mettre de point virgule.

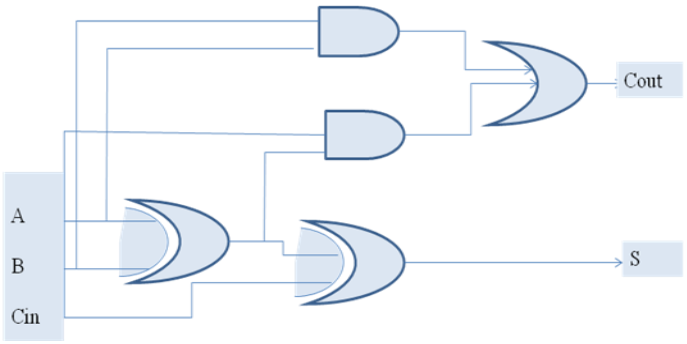
12.2 Spécification d'architecture

- L'architecture est la description interne du circuit
- Elle contient une **partie déclarative** et une **partie exécutive**
- C'est cette partie (architecture) qui contient les instructions permettant de décrire le modèle du circuit
- L'architecture est associée obligatoirement à une entité

  Entity Nom_entite is

  Architecture Nom_Architecture of Nom_entite is
- Le nom de l'architecture est quelconque mais doit être unique
- Plusieurs architectures peuvent correspondre à une même entité

L'exemple suivant montre un schéma d'un additionneur 1 bit



En VHDL l'architecture de cet additionneur sera donnée comme suit :

Architecture add_un_bit of Additionneur is -- Additionneur est le nom de l'entité
--correspondante à cette architecture

begin

S <= A xor B xor Cin;

Cout <= (A and B) or ((A xor B) and Cin);

end add_un_bit;

Donc en résumé la description VHDL de cet additionneur peut être donnée comme suit :

```
-- Déclaration des librairies (bibliothèques) à utiliser
entity   Additionneur  is
        port (
                A : in bit;
                B : in bit;
                Cin : in bit;
                S : out bit;
                Cout : out bit);
end Additionneur ;
Architecture add_un_bit of Additionneur  is
-- {Partie declarative }
begin
S <= A xor B xor Cin;
Cout <= (A and B) or ((A xor B) and Cin);
end add_un_bit;
```

VHDL est un langage très déclaratif qui permet au compilateur d'effectuer des vérifications poussées et de fiabiliser l'écriture d'un modèle.

Dans une architecture, il est nécessaire de déclarer les objets utilisés et leurs types avant la zone décrivant l'architecture (le corps de l'architecture situé entre **begin** et **end**).

Parmi les objets très utilisés, figurent les signaux (équipotentielles reliant les instructions et les composants) et les composants (correspondent effectivement aux composants dont on a besoin dans l'architecture pour une description structurelle).

Les ports déclarés dans l'entité sont des signaux utilisables dans l'architecture, il ne faut pas les déclarer une nouvelle fois.

13. Déclaration des bibliothèques

Il est possible d'utiliser d'autres bibliothèques qui sont alors des bibliothèques de **ressources.**

Donc, il est très pratique d'utiliser les paquetages des bibliothèques, ce qui permet d'utiliser des objets (constantes, fonctions, composants, …) qui peuvent être définis dans une bibliothèque différente de celle en cours. C'est le cas des bibliothèques standards comme IEEE qui définit des types et objets compris par les outils de synthèse.

Pour utiliser le contenu d'un paquetage, il faut déclarer la bibliothèque dans laquelle il se trouve ainsi que ce paquetage :

-- Déclaration de la bibliothèque avec le mot clé library

library Nom_de_la_librairie

-- Selectionner le paquetage à utiliser avec directive use

use Nom_de_la_librairie.Nom_paquetage.all;

-- Si on ne veut pas utiliser tout le paquetage mais seulement un seul objet de ce paquetage.

use Nom_de_la_librairie . Nom_paquetage. Objet;

Exemple

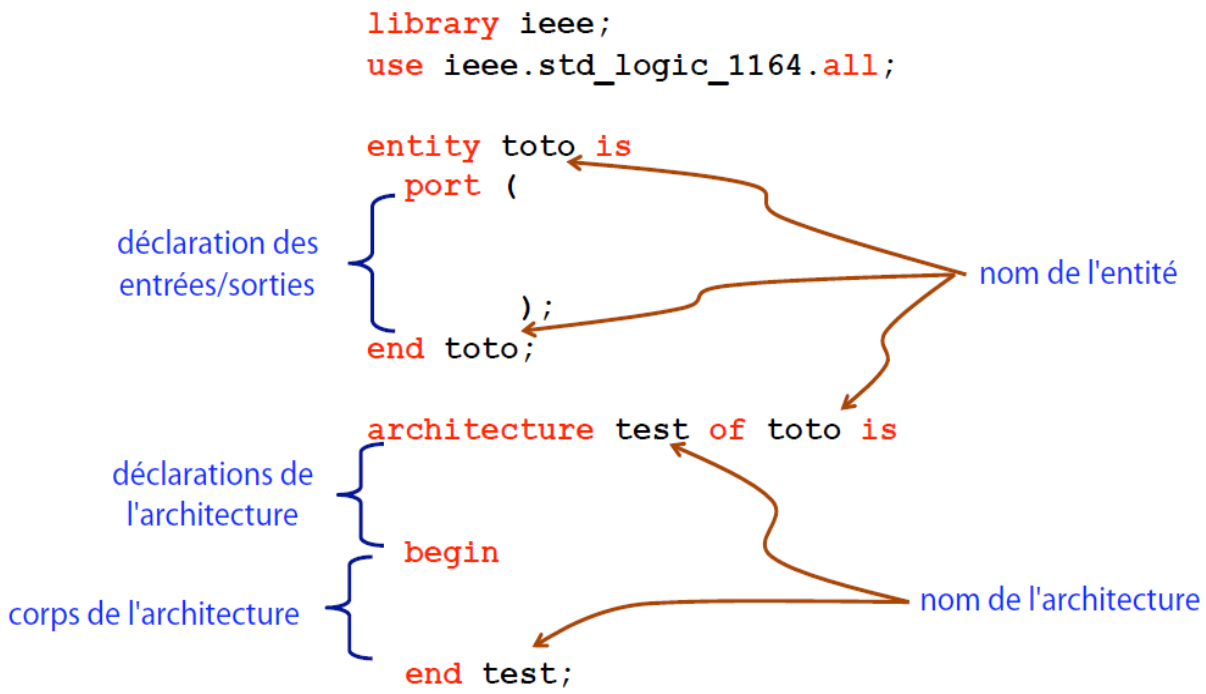
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
```

La bibliothèque par défaut est work. Work est aussi le nom symbolique de la bibliothèque dans laquelle sont rangés les résultats. La bibliothèque STD est une bibliothèque standard fournie avec le langage. Elle contient des définitions des types et des fonctions de base (integer, bit, boolean …) dans le paquetage STANDARD et des fonctions sur les caractères dans le paquetage TEXTIO.

Par défaut, les bibliothèques STD et WORK n'ont pas besoin d'être déclarées pour être utilisables. Tout se passe comme si un programme VHDL commençait toujours par :

```
Library STD;
Library WORK;
```

A ce niveau on peut donner la structure d'un programme VHDL simple



14. Les descriptions d'une architecture en VHDL

Il existe deux façons différentes pour la spécification d'une architecture en VHDL. L'une dite **Comportementale** et l'autre **Structurelle**.

14.1. Description comportementale

Décrit la fonction d'un système sans s'attacher à comment cette fonction sera implémentée en pratique.

- Sous forme d'algorithme à l'aide d'instructions séquentielles ou sous forme de flow de données.
- Fait appel à un **process**.
- Le temps peut intervenir

Exemple de description comportementale à l'aide d'instructions séquentielles

```
Architecture comportementale of demonstration is
--partie déclarative
Begin
C<='1' when (A=B) else '0';
End comportementale ;
```

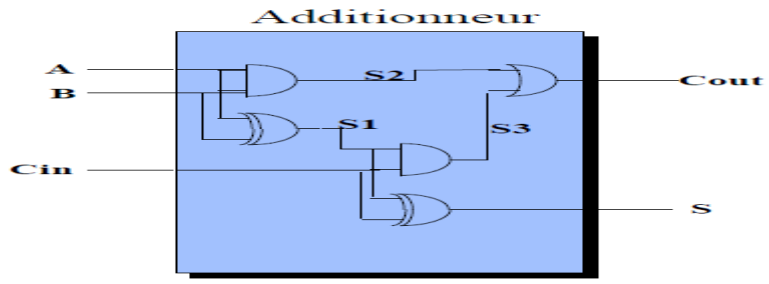
Exemple de description comportementale sous forme de flow de données

Dans ce cas, le flot de données sortant est exprimé par rapport au flot de données entrant

Exemple1

```
Architecture Arch_Comport of Entite_X
Begin
   Som <=A xor B;
  Cout <=A and B;
End Arch_Comport;
```

Exemple2




```
architecture flot of Additionneur is
signal S1, S2, S3 : bit ;
begin
      S1 <= A xor B after 10 ns ;
      S2 <= A and B after 5 ns ;
      S3 <= S1 and Cin after 5 ns ;
      S <= S1 xor Cin after 10 ns ;
      Cout <= S2 or S3 after 5 ns ;
end flot ;
```

14.2. Description *Structurelle*

Décrit la structure d'un système sous forme d'arbre (hiérarchique), en décrivant les interconnexions entre éléments qui sont préalablement décrits en vhdl (en utilisant **component)**.

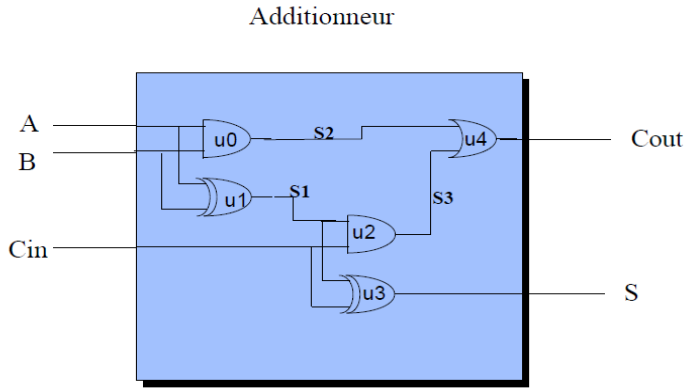
- Ne fait pas intervenir le temps
- Décrit la structure de la fonction réalisée
- Décrit un schéma, des connexions entre composants supposés exister dans une librairie de travail, sous forme d'unités de conception. Le programme se contente alors d'instancier les composants nécessaires et de décrire leurs interconnexions.

Exemple

```
Architecture structurelle of demonstration is
--partie déclarative
Begin
C<=not(A xor B);
End structurelle ;
```

Exercice

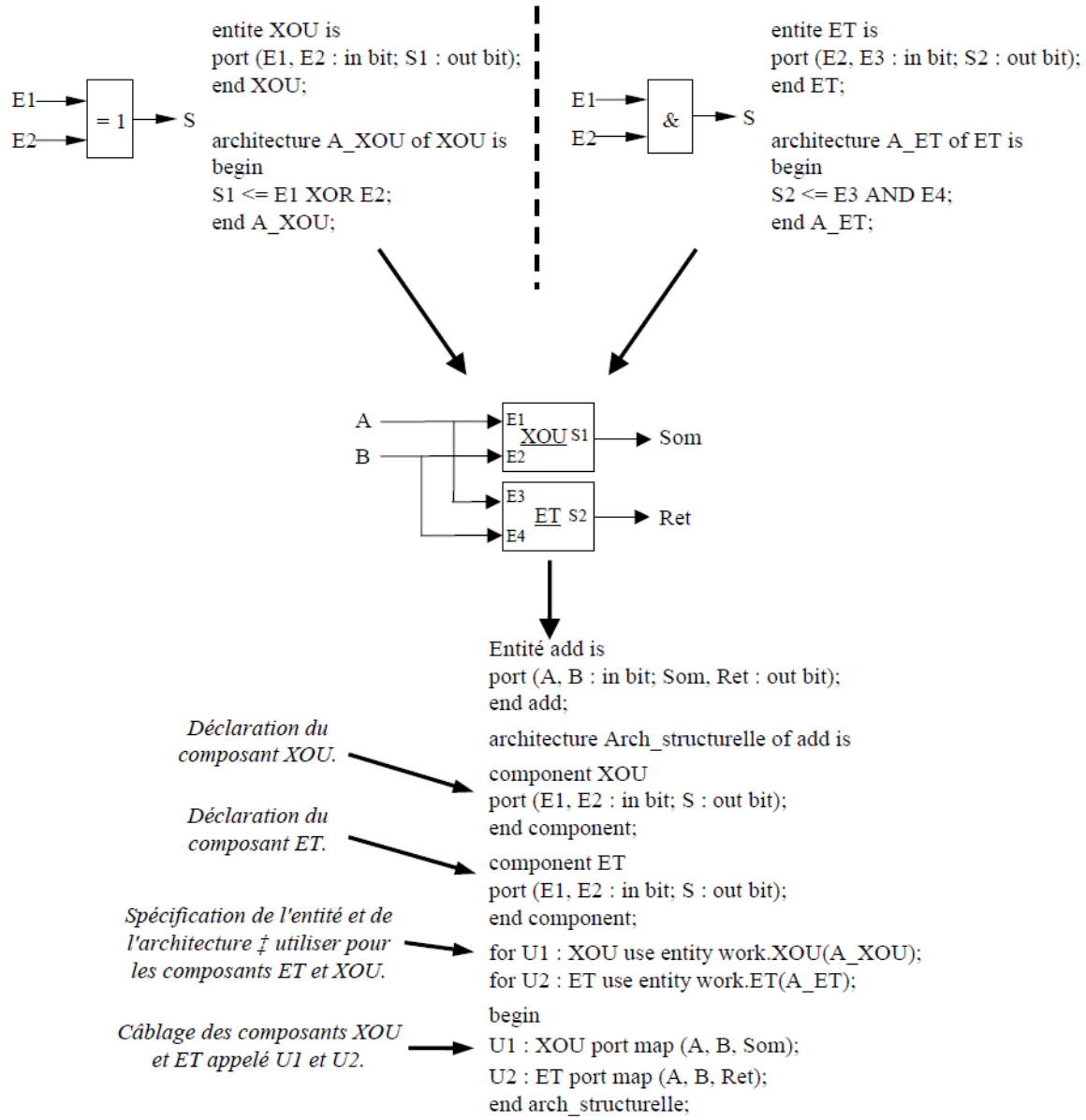
Donner la description Structurelle de VHDL de cet additionneur 1bit.



Corrigé

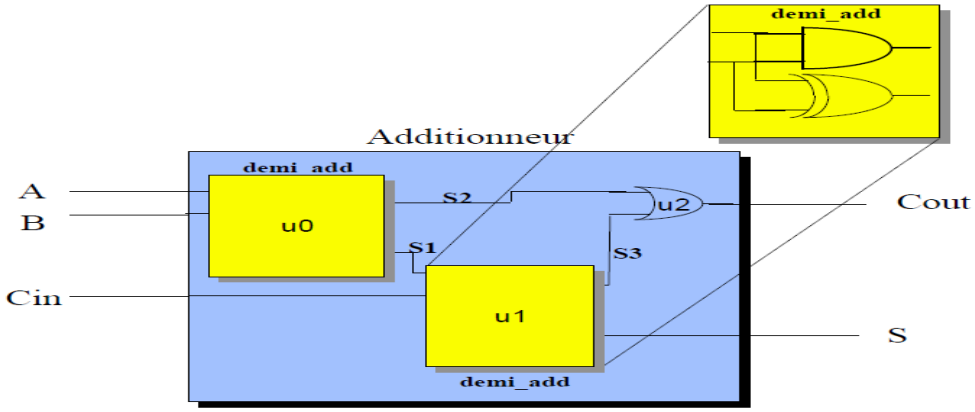
```
architecture structurelle1 of Additionneur is

component porteOU
      port ( e1 : in bit; e2 : in bit; s : out bit );
end component;

component porteET
port ( e1 : in bit; e2 : in bit; s : out bit );
end component;

component porteXOR
port ( e1 : in bit; e2 : in bit; s : out bit );
end component;

signal S1, S2, S3 : bit;

begin
      u0 : porteET
            port map (A, B, S2);
      u1 : porteXOR
           port map (A, B, S1);
      u2 : porteET
            port map ( S1, Cin, S3);
      u3 : porteXOR
            port map ( S1, Cin, S);
      u4 : porteOU
            port map ( S2, S1, Cout);
end structurelle1;
```

Exemple2



Modèle structurel : Hiérarchisation

Au lieu de considérer que des portes logiques de base on peut utiliser d'autres composants comme le demi-additionneur dans la réalisation d'un additionneur 1bit.



NB : Dans une description VHDL, on peut trouver ces différentes descriptions.

15. Les signaux et les variables en VHDL

15.1 Les variables

Une variable est l'image d'un objet auquel on peut affecter, à tout moment, la valeur qu'on veut. L'affectation d'une variable se fait à l'aide de l'opérateur :=

Nom_Variable := expression ;

Exemple : Begin

```
process(N)
begin
    variable A,B : Integer ;
        A := N ;
        B := A ;
End process;
End ;
```

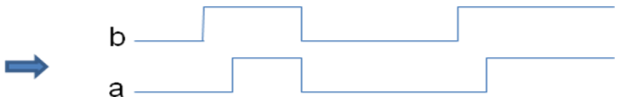
15.2 Les signaux

- ✓ Les composants électroniques communiquent entre eux à l'aide de signaux
- ✓ En VHDL, un signal est un objet modélisant une interconnexion
- ✓ Si le signal modélise une interconnexion entre l'entité dans laquelle ce signal est définit et un objet externe, le signal est dit port. Il doit avoir un type et une direction
- ✓ Si le signal modélise une interconnexion entre deux composants internes de la même entité, ce dernier est dit interne. Il n'a pas de direction mais doit avoir un type.
- ✓ Un signal externe (port) est déclaré dans l'entité. Un signal interne est déclaré dans l'architecture.
- ✓ Un signal peut être de n'importe quel type

15.2.1 Manipulation de signaux

Signal a, b : std_logic;
Signal c: std_logic_vector(7 downto 0);

a <= '1' after 10ns, when b='1' else '0';

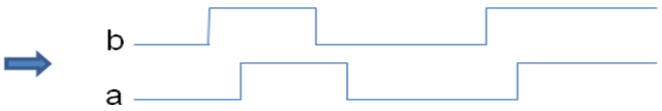


a <= '0', '1' after 10ns, '0' after 20ns,
'1' after 30ns;

c <= (7=>'1', others =>'0');

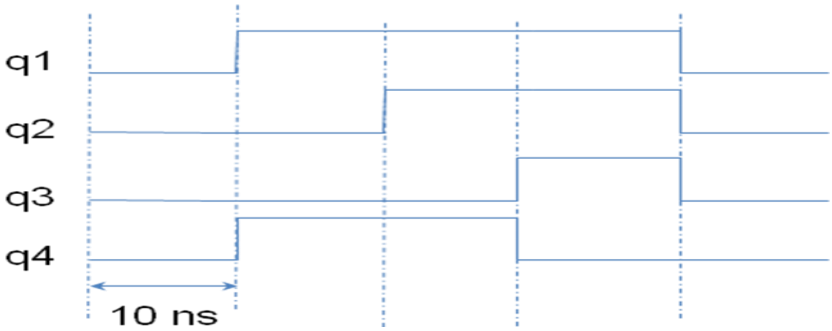
C ="10000000"

a <= b after 10ns;



Exercice

Un dispositif génère 4 signaux (q1, q2, q3, q4) définissant une séquence comme indiqué dans le chronogramme suivant :



Ecrire un programme VHDL modélisant un circuit générant cette séquence

15.3. Différence variables-signaux

➢ Portée

- Une variable à une portée locale, elle sert de moyen de retenir des informations temporaires à l'intérieur d'un processus.
- Un signal à une portée globale, il sert de moyen de communication (interconnexion) entre les éléments internes et externes d'une description

➢ Mise à jour

- Une variable est mise à jour immédiatement après son affectation
- Un signal n'est mis à jour qu'après l'arrêt du processus le manipulant

➢ Sens de l'affectation

- Affecter une variable à une autre revient à remplacer le contenu de la variable par le contenu de l'autre.
- Affecter un signal à un autre (par des process différents) revient à les interconnecter.

16. La concurrence en VHDL

VHDL est un langage concurrent. Pour une description VHDL toutes les instructions sont évaluées et affectent les signaux de sortie en même temps. L'ordre dans lequel elles sont écrites n'a aucune importance. En effet la description génère des structures électroniques, c'est la grande différence entre une description VHDL et un langage informatique classique.

Exemple : Pour le décodeur 1 parmi 4, l'ordre dans lequel seront écrites les instructions n'a aucune importance.

```
architecture DESCRIPTION of DECOD1_4 is
begin
D0 <= (not(IN1) and not(IN0));          -- première instruction
D1 <= (not(IN1) and IN0);               -- deuxième instruction
D2 <= (IN1 and not(IN0));               -- troisième instruction
D3 <= (IN1 and IN0);                    -- quatrième instruction
end DESCRIPTION;
```

L'architecture ci dessous est équivalente :

```
architecture DESCRIPTION of DECOD1_4 is
begin
D1 <= (not(IN1) and IN0);               -- deuxième instruction
D2 <= (IN1 and not(IN0));               -- troisième instruction
D0 <= (not(IN1) AND not(IN0));          -- première instruction
D3 <= (IN1 AND IN0);                    -- quatrième instruction
end DESCRIPTION;
```

Chaque instruction concurrente est prise en charge par un processus indépendant qui s'exécute de façon indéfinie. Après chaque exécution, le processus est mis en attente d'un événement sur les opérandes de l'instruction concurrente. Un programme VHDL est donc composé d'un ensemble de processus s'exécutant en parallèle de façon indéfinie et se mettant en attente d'événements sur les signaux déclencheurs.

16.1 Les instructions concurrentes

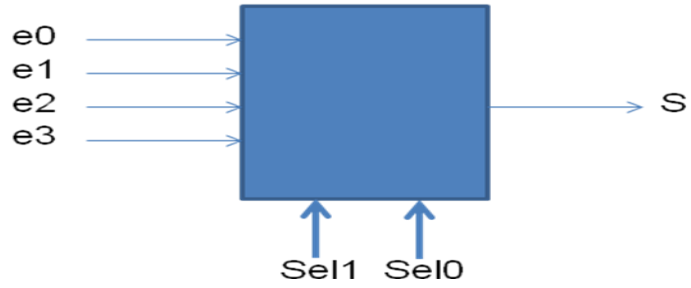
- S <= a **op** b **op** c … -- avec **op** est un opérateur quelconque
- S <= a when condition1 else
  b when condition 2 else c;

- With a select

```
s <= valeur1 when condition1,
     valeur2 when condition2,
     valeur3 when others;
```

Exercice

Donner une description VHDL d'un multiplexeur 4 vers 1.



Sel1	Sel0	S
0	0	e0
0	1	e1
1	0	e2
1	1	e3

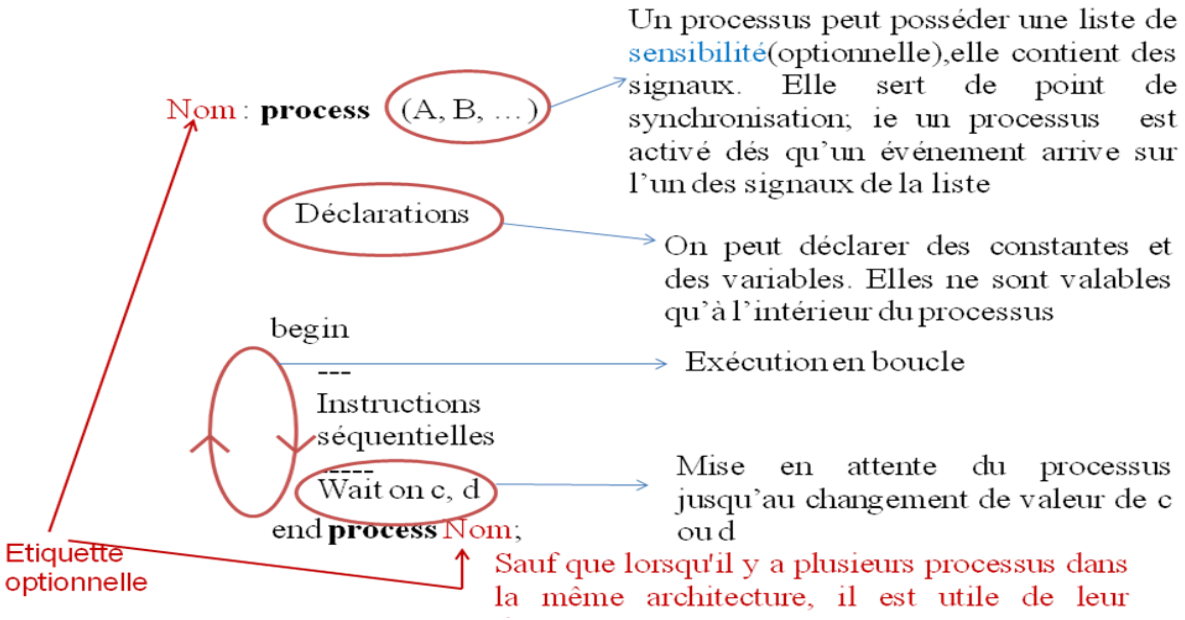
17. Les Processus

Un processus 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. Il permet d'effectuer des opérations sur les signaux en utilisant les instructions standards de la programmation structurée comme dans les systèmes à microprocesseurs.

Un Processus est **explicite** (défini avec le mot clé **process**) ou **implicite** (instruction concurrente). Donc en VHDL, tout est processus.
Le déclencheur d'un processus peut être :

- le changement de valeurs d'un ensemble de signaux ; les signaux de la liste de sensibilité par exemple
- la vérification d'une condition, ou
- l'épuisement d'un délai.

17.1 Anatomie d'un process



17.2 Mise en attente d'un process

Un processus s'exécute de façon indéfinie (cyclique) jusqu'à la fin de la simulation. Le programmeur peut imposer au processus d'attendre un événement sur des signaux particuliers ou un délai avant de poursuivre son exécution.

Il existe deux façons d'arrêter un processus :

- ✓ Ajouter une **liste de sensibilité** dans l'entête du processus (comme le montre la figure ci-desssus.

nom: process (**x, y, z**) -- le processus se déclenche dés qu'un

-- changement arrive sur les signaux x, y, ou z

- ✓ Ajouter une **instruction wait** dans le processus (wait permet la synchronisation des processus en les suspendant)

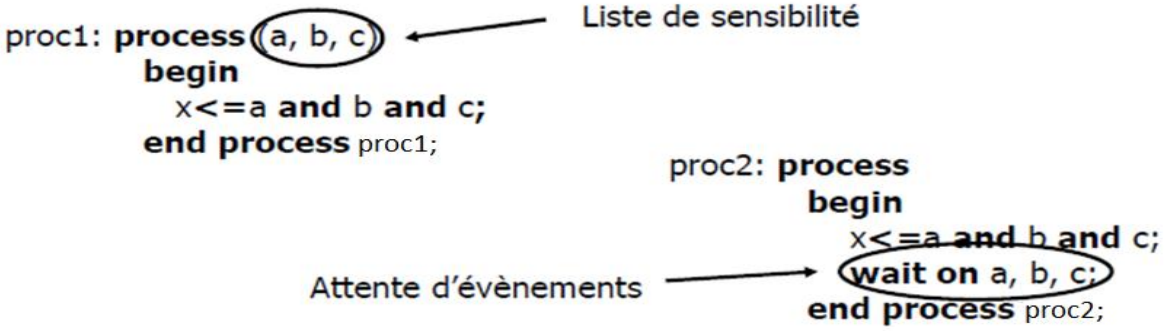
wait on x, y -- attente de changement sur x ou y

wait for delai -- attente d'un délais

wait until condition -- attente de vérification d'une condition

wait -- stoppe le processus indéfiniment

Il y a une équivalence entre l'**instruction wait** et la **liste de sensibilité,** comme le montrent les deux processus ci-dessous :



17.3 Les instructions séquentielles

Ce sont des instructions qui s'exécutent de façon séquentielles. Leurs **POSITION** à une importance sur le résultat final. Elles sont définies obligatoirement à l'intérieur d'un processus, d'une procédure ou d'une fonction. Une instruction séquentielle s'exécute de façon séquentielle à l'intérieur d'un processus mais elle est considérée comme concurrente vis avis des autres processus du programme.

On cite les instructions suivantes :

➢ Instruction **IF**

```
if condition then instructions1
elsif condition then instruction2
else instructions 3
end if;
```

➢ Instruction **case**

```
case identificateur is
    when valeur1 => Instruction 1;
    when valeur2 => Instruction 2;
    when others => Instruction 3;
end case;
```

➢ Instruction **For loop**

```
[Etiquette:] For identificateur in range loop
                instructions
end loop;
```

➢ Instruction **While loop**

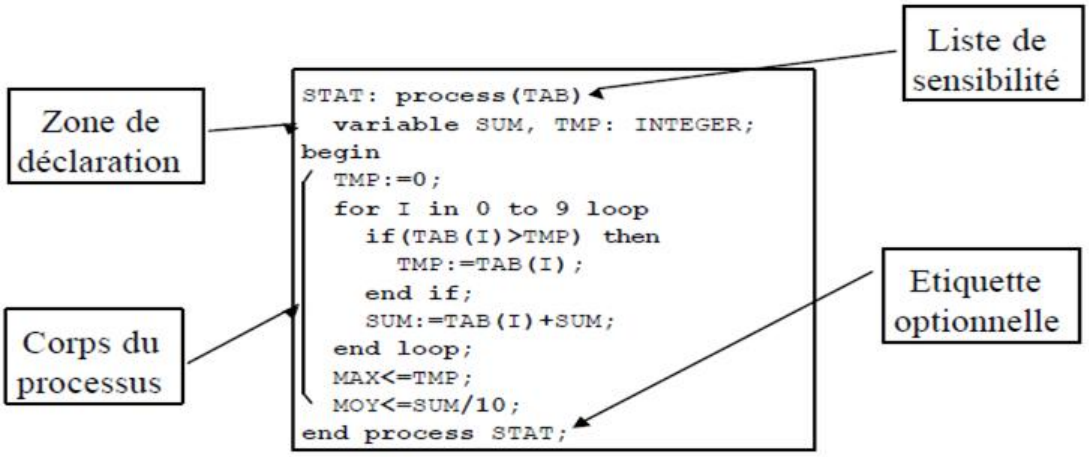
```
[Etiquette:] While condition loop
                instructions
end loop;
```

On peut utiliser les instructions suivantes pour mieux manipuler les boucles :

Exit [Etiquette] when condition : sortir de la boucle si condition est vérifiée

Next[Etiquette] when condition : sortir de l'itération si condition est vérifiée

Exemple



17.4 Équivalence entre processus implicites et explicites

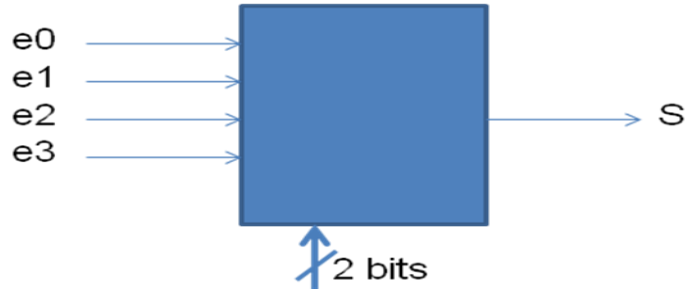
Processus implicite	**Processus explicite équivalent**

s <= a AND b;	equivalent: PROCESS
	BEGIN
	s <= a AND b;
	WAIT ON a,b;
	END PROCESS;

neuf <= '1'WHEN etat = 9 ELSE '0'	equivalent:PROCESS(etat)
	BEGIN
	IF etat = 9 THEN neuf <= '1';
	ELSE neuf <= '0';
	END IF;
	END PROCESS;

Exercice

Donner une description VHDL (comportementale) d'un multiplexeur 4 vers 1. La sortie est à haute impédance si aucune sélection n'est effectuée.



Sel		S
0	0	e0
0	1	e1
1	0	e2
1	1	e3