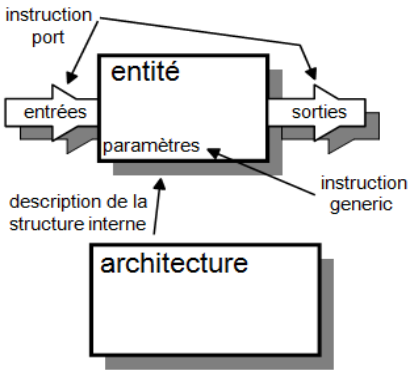
Version 1.1 ; Date : 15/02/2019

Synthèse en VHDL : aide mémoire

I- Structure d'une description VHDL

Une description VHDL est composée de deux parties: une entité et une architecture. L'entité décrit l'interface entre le design et le monde extérieur. L'architecture décrit la structure interne du composant.



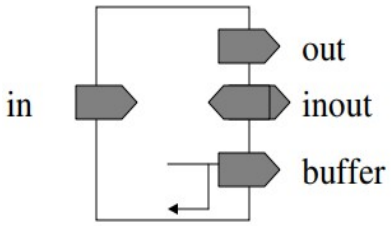
ENTITÉ	**ENTITY** *Nom de l'entité* **IS** **GENERIC (***Description des paramètres en explicitant pour chacun le nom, le type et éventuellement la valeur par défaut***) ;** **PORT (***Description des entrées et des sorties de la structure en explicitant pour chacune d'entre elles : le nom, la direction (**IN**, **OUT** et **INOUT**) et le type***) ;** **END** *Nom de l'entité* **;**
ARCHITECTURE	**ARCHITECTURE** *Nom de l'architecture* **OF** *Nom de l'entité* **IS** *Zone de déclaration.* **BEGIN** Description du fonctionnement **END** *nom de l'architecture* **;**

II- La partie entity

Facultative, l'instruction *GENERIC*, dans la partie *ENTITY*, permet de spécifier certains paramètres de l'entité, comme par exemple la largeur d'un bus. Le paramètre peut être initialisé directement dans l'entité à une certaine valeur, puis modifié ultérieurement lors de son utilisation. Un paramètre est considérée comme une constante dont sa valeur est spécifiée lors de l'utilisation du composant.

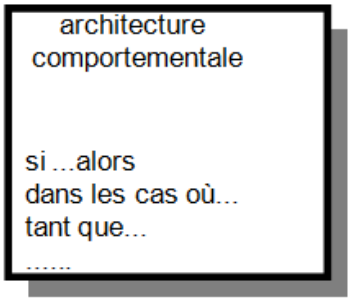
L'instruction port, dans la partie *ENTITY*, permet de définir les ports d'entrées et de sorties du design. Pour chacun, on définit son **mode** et son **type**.

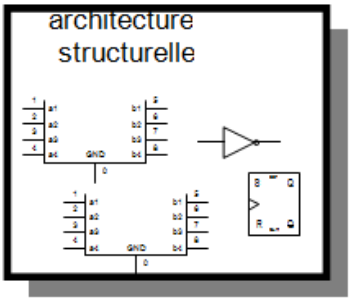
Le mode d'un port peut être :
- **IN**, un port d'entrée. Il n'est pas possible de modifier la valeur d'une entrée dans la partie architecture.
- **OUT**, port de sortie. Il n'est pas possible de lire la valeur d'une sortie directement dans la partie architecture.
- **INOUT**, un port bidirectionnel.
- **BUFFER**, un port de sortie mais sa valeur peut être lue en interne dans la partie architecture. Il est conseillé de ne pas utiliser ce mode.

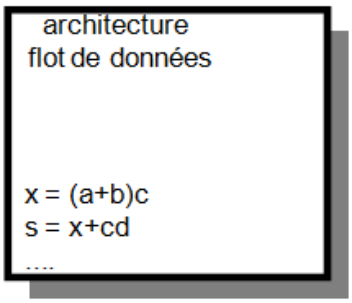


III- Partie architecture

1) Types de descriptions





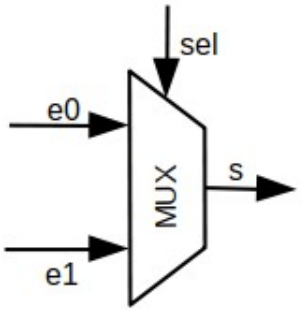


a) Description par flots de données

Une description par flots de données spécifie pour un circuit combinatoire comment une donnée est transférée d'un signal à un autre en utilisant des affectations concurrentes. Autrement dit, elle s'effectue à travers les équations logiques décrivant le circuit.

Exemple : description d'un multiplexeur 2 vers 1

```
library ieee ;
use ieee.std_logic_1164.all ;
entity MUX is port (
sel,e0,e1 : in std_logic ;
s : out std_logic) ;
end entity ;
architecture arch_mux of MUX is
begin
     s<=(e0 and not sel) or (e1 and sel) ;
end architecture ;
```



b) Description structurelle

La description du circuit s'effectue sous forme de boites noires (composants) interconnectés en réalisant l'assemblage des composants déjà décrits.

Après avoir été décrit par une entité et une architecture, le composant doit être d'abord déclaré dans la zone déclarative de la partie architecture dans laquelle il serait utilisé.

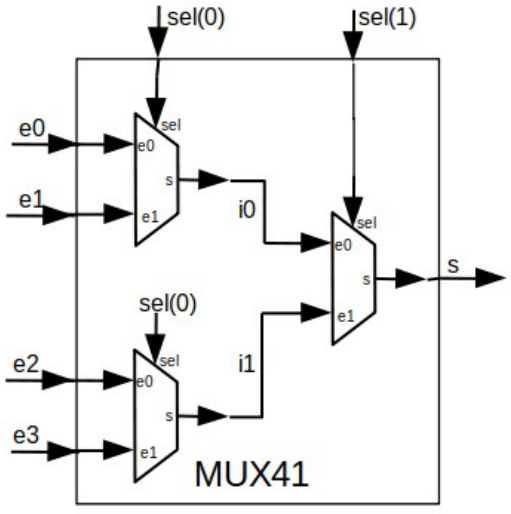
COMPONENT nom de l'entité décrivant le composant à utiliser
GENERIC (les paramètres du composant) ;
PORT(les entrées et les sorties du composant) ;
END COMPONENT ;

L'utilisation d'un composant s'effectue par la création d'une instance du composant en lui spécifiant un nom. On précise ensuite les liaisons existantes entre les ports du composant (ports formels) et les connexions de l'architecture (ports effectifs). L'instanciation doit être faite entre les mots begin et end de l'architecture :

nom de instance : nom de l'entité décrivant le composant
GENERIC MAP (nom du paramètre => valeur,...,nom du paramètre => valeur)
PORT MAP (nom du port => signal,.., nom du port => signal) ;

Exemple : description d'un multiplexeur 4 vers 1 à l'aide de trois multiplexeurs 2 vers 1

```
library ieee;
use ieee.std_logic_1164.all;
entity MUX4 is port (
e0,e1,e2,e3 : in std_logic ;
sel : in std_logic_vector(1 downto 0);
s : out std_logic);
end entity;
component MUX port (
sel,e0,e1 : in std_logic;
s : out std_logic);
end component;
architecture arch_mux4 of MUX4 is
signal i0,i1 : std_logic;
begin
  inst0 : MUX port
map(sel=>sel(0),e0=>e0,e1=>e1,s=>i0);
  inst1 : MUX port
map(sel=>sel(0),e0=>e2,e1=>e3,s=>i1);
  inst2 : MUX port
map(sel=>sel(1),e0=>i0,e1=>i1,s=>s);
end architecture ;
```



c) Description comportementale

Une description comportementale fournit un algorithme qui modélise le fonctionnement du circuit en utilisant des instructions séquentielles. Un algorithme est défini par un '*process*'.

```
[étiquette :] PROCESS (LISTE_DE_SENSIBILITE)
[NOM_DES_OBJETS_INTERNES : « type » ; ]
BEGIN
        INSTRUCTIONS_SEQUENTIELLES ;
END PROCESS [étiquette] ;
```

Dans les processus, il est possible de déclarer des variables. Les variables permettent de garder des résultats intermédiaires et ne correspondent à rien de physique.

Signaux VS Variables	
Utilisés avec tous les types de descriptions Déclarés dans la partie déclarative de l'architecture	Utilisées uniquement dans les process Déclarées dans la partie déclarative d'un process
Dans un process, la mise à jour est effectué à la fin du process	Mise à jour immédiate.
Un point interne	Aucun sens physique

2) Instructions séquentielles et instructions concurrentes

Instructions séquentielles (uniquement dans un PROCESS)	

Instruction IF	**IF** condition1 **THEN**
	instructions séquentielles;
	ELSIF condition2 **THEN**
	instructions séquentielles;
	ELSIF condition3 **THEN**
	instructions séquentielles;
	ELSE
	instructions séquentielles;
	END IF;

Instruction CASE	**CASE** signal **IS** **WHEN** valeur1 **=>** instructions séquentielles; **WHEN** valeur2 **=>** instructions séquentielles; **WHEN** valeur3 **=>** instructions séquentielles; **WHEN OTHERS =>** instructions séquentielles; **END CASE;**
Boucle FOR...LOOP	**FOR** indice **IN** valeur initiale **TO**/**DOWNTO** valeur finale **LOOP** instructions séquentielles;

	END LOOP;
Boucle WHILE...LOOP	**WHILE** condition **LOOP** instructions séquentielles ; **END LOOP ;**

Instructions concurrentes	
Instruction WHEN...ELSE	signal **<=** valeur1 **WHEN** condition1 **ELSE** valeur2 **WHEN** condition2 **ELSE** valeur3 **WHEN** condition3 **ELSE** valeur**;**
Instruction WITH...SELECT	**WITH** signal de sélection **SELECT** signal **<=** valeur1 **WHEN** valeur1 de sélection**,** valeur2 **WHEN** valeur2 de sélection**,** valeur **WHEN OTHERS ;**
Boucle FOR..GENERATE	étiquette : **FOR** indice **IN** valeur initiale **TO**/**DOWNTO** valeur maximale **GENERATE** instructions concurrentes ; **END GENERATE ;**
Instruction IF..GENERATE	étiquette : **IF** condition **GENERATE** instructions concurrentes ; **END GENERATE**

IV- Types, opérateurs et attributs

1) Types d'objets

Types prédéfinis dans la bibliothèque standard	
Nom du type	**Définition**
bit	Deux valeurs possibles '0' ou '1'
bit_vector	Un vecteur de bits
integer	Valeur entière positive ou négative codée sur 32 bits
boolean	Deux valeurs possibles TRUE ou FALSE
time	Nombre réel de temps fs, ps, us, ms, sec, min
Types prédéfinis dans la bibliothèque standard IEEE	
Nom du type	**Définition**
std_logic	Défini dans le paquetage ieee.std_logic_1164 Valeurs possibles : 'U' = Non initialisé, 'X ' = inconnu forçage fort, '0' = forçage fort, '1' = forçage fort, 'Z ' = haute impédance, 'W' = inconnu forçage faible, 'L' =

	forçage faible, 'H' = forçage faible, et '-' = quelconque.
std_logic_vector	Défini dans le paquetage ieee.std_logic_1164 Un vecteur de std_logic
unsigned	Défini dans le paquetage ieee.numeric_std Vecteur de bits représentant un nombre entier non signé
signed	Défini dans le paquetage ieee.numeric_std Vecteur de bits représentant un nombre entier signé

2) Opérateurs

Opérateur	VHDL
Affectation	<= pour les signaux := pour les variables (uniquement dans un process)
Concaténation	&
Opérateurs logiques ET NON ET OU NON OU OU EXCLUSIF NON OU EXCLUSIF NON	 AND NAND OR NOR XOR NXOR NOT
Opérateurs relationnels ÉGAL DIFFÉRENT INFÉRIEUR, INFÉRIEUR OU ÉGAL SUPÉRIEUR, SUPÉRIEUR OU ÉGAL	 = /= <, <= >, >=
Opérateurs arithmétiques ADDITION SOUSTRACTION MULTIPLICATION DIVISION EXPONENTIEL MOD	 + - * / ** (NB. 2ème opérande doit être une constante) Modulo

3) Attributs

Attribut	Définition - informations de retour
'high	Associé à un vecteur, il renvoie une valeur entière indiquant le rang le plus haut.
'low	Associé à un vecteur, il renvoie une valeur entière indiquant le rang le plus bas.
'left	Associé à un vecteur, il retourne la valeur entière spécifiant le rang le plus à gauche.
'right	Associé à un vecteur, il retourne la valeur entière spécifiant le rang le plus à droite.
'event	Associé à un signal, il retourne TRUE si le signal change d'état.
'stable(T)	Associé à un signal, il retourne TRUE si le signal n'est pas modifié pendant la durée T.
'length	Associé à un vecteur, il retourne X'high - X'low + 1 sous la forme d'un entier.

IV- Fonctions et procédures

VHDL offre l'utilisation des fonctions et des procédures. Elles sont utilisées pour améliorer la lecture des descriptions VHDL ainsi que pour favoriser la réutilisation du code.

1) Les procédures

Une procédure peut contenir des contrôles de synchronisation, et elle peut appeler d'autres procédures et fonctions.

```
PROCEDURE nom de la procédure (
      [classeObjet] nomParametre : [sens] type ;
      ...
      [classeObjet] nomParametre : [sens] type) IS
 [déclaration des variables]
BEGIN
Instructions séquentielles
END PROCEDURE [nom de la procédure];

classeObjet = constant ou variable ou signal
sens = in ou out ou inout
```

Exemple

```
procedure registre(signal raz: in std_logic; signal h: in std_logic;
signal e : in std_logic; signal s : out std_logic) is
begin
   if (raz = '1') then s <= '0';
   elsif (h'event and h = '1') then
      s <= e;
   end if;
end procedure registre;
```

2) Les fonctions

Les fonctions sont utilisées pour synthétiser des parties combinatoires uniquement. Elle ne contiennent pas des contrôles de synchronisation. Une fonction doit renvoyer toujours un seul résultat.

FUNCTION nom de la fonction **(**
nomParamètre **:** type ;
....
nomParamètre **:** type**) RETURN** type du paramètre retour **IS**
[déclaration des variables]
BEGIN
Instructions séquentielles
END FUNCTION;

Exemple

```
function NONET (A, B, C : bit) return bit is
variable result : bit;
begin
   result := not (A and B and C);
   return result;
end function;
```