Synthèse en VHDL des systèmes embarqués

Dr. Ing. Chiheb Ameur ABID

Contact: chiheb.abid@gmail.com

Mars 2023

Plan

Technologies des systèmes logiques

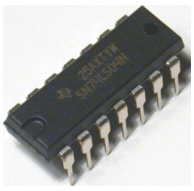
Technologies des systèmes logiques

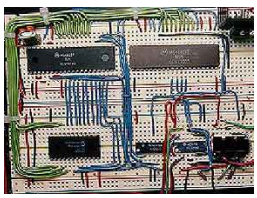
- Logique standard 74xx et 40xx
- ASIC (Application Specific Integrated Circuit) pré-caractérisé - full custom
- PLD (Programmable Logic Device, circuit logique programmable)
  - PAL (Programmable Array Logic) : contiennent une matrice de portes ET programmable et une matrice de portes OU fixe. Ils sont rapides, mais ont une capacité limitée et ne sont programmables qu'une seule fois.
  - GAL (Generic Array Logic) : améliorent les PAL en rendant la matrice de portes OU programmable et en utilisant une technologie CMOS ou EECMOS qui permet la reprogrammation
  - CPLD (Complex Programmable Logic Device) : regroupent plusieurs GAL reliés par un réseau d'interconnexion. Ils offrent une plus grande capacité et une plus grande flexibilité que les PAL et les GAL, mais sont plus lents
  - FPGA (Field Programmable Gate Array) : ce sont des circuits qui contiennent des blocs logiques élémentaires (LUT, bascules, multiplexeurs, etc.) reliés par un réseau d'interconnexion programmable.

Technologies des systèmes logiques

Logique standard

- La conception d'un système se repose sur l'usage d'une 'bibliothèque' de circuits intégrés

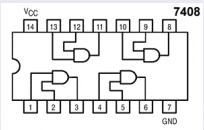




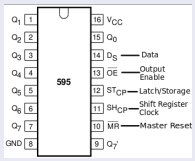
Logique standard

Technologies

- Technologie SSI (≈ 100 transistors / cm2)



- MSI, LSI ou VLSI (≈ 1million transistors / cm2) de la famille TTL (Transistors Transistors Logic) ou CMOS (transistors à effet de champ MOS)



Logique standard

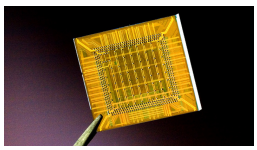
Avantages et inconvénients

- Avantages
  - ✔ Projets didactiques
  - ✔ Disponibilité
- Inconvénients
  - ✖ Temps de développement trop long
  - ✖ Coût de réalisation très important
  - ✖ Taille de la réalisation
  - ✖ Difficile de gérer des projets complexes

Les ASICs (Application Specific Integerated Circuit)

Les ASICs (Application Specific Integerated Circuit)

- Les concepteurs de systèmes s'adressent directement aux fondeurs de silicium



Les ASICs (Application Specific Integerated Circuit)

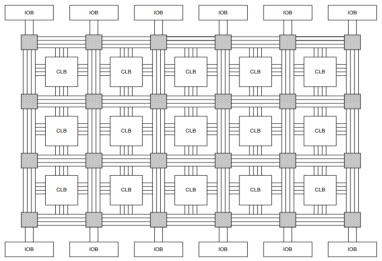
Création d'un processus

- Coût d'un fab dépasse le 1Milliard de $
- Fondeurs de silicium
  - ☞ Intel, Texas Instruments, STMicroelectronics, etc.
  - ☞ TSCM, UMC, GlobalFoundries, etc.
- Coût très important, mais 'rentable pour des systèmes à grande diffusion'
- Haute performance

Les FPGAs (Field Programmable Gatway Array)

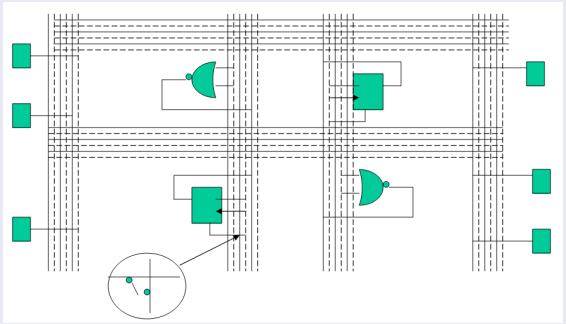
Présentation des circuits FPGA

- Un FPGA est composé de :
  - Un réseau de blocs de logique programmable (Configurable Logic Block CLB), chaque bloc pouvant réaliser des fonctions complexes de plusieurs variables, et comportant des éléments à mémoire ;
  - Un réseau d'interconnexions programmables entre les blocs
  - Des blocs spéciaux d'entrée et de sortie (Input/Output Block – IOB).



Les FPGAs (Field Programmable Gatway Array)

Principe de xonfiguration d'un circuit FPGA



Plan

Présentation de VHDL

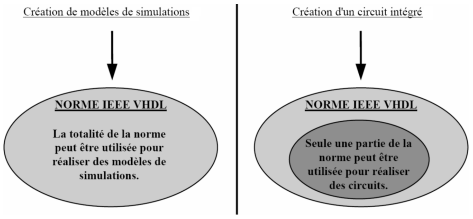
Historique

- Langage standard de description de circuits ou de systèmes numériques en vue de
  - Modélisation (simulation) des circuits ou systèmes
  - Synthèse (génération automatique) niveau RTL
  - Descriptions de programmes de test (stimuli)
  - Description de type hiérarchique (netlist)
- Inventé par le département de la défense Américaine (DoD) au début des années 80
- VHDL : VHSIC Hardware Description Language
  - Very High Speed Integrated Circuit
- Utilise une syntaxe similaire à ADA
- L'objectif est de standardiser les méthodes de modélisation de composantes électroniques.
- Standardisé par IEEE en 1987 (revue en 1993, 2002, 2008, 2019.

Présentation de VHDL

Modélisation ou Synthèse ?

- Le simulateur
  - Modélisation (simulation) des circuits ou systèmes
  - Comprend l'ensemble du langage VHDL
- Le synthétiseur
  - Traduit la description VHDL en une netlist logique
  - Comprend uniquement une partie du langage VHDL



Présentation de VHDL

Modélisation ou Synthèse ?

- Modélisation
  - Tout le langage : Logique + Temporel
  - Un modèle peut être comportemental, structurel ou de type data-flow.
  - Exemple : créer desprogrammes de test
- Synthèse
  - Langage simplifié (pas de retards). Le style d'écriture anticipe une primitive circuit.
  - La synthèse demande une bonne connaissance du circuit et de la technologie

Plan

Structure d'une description VHDL

Entité et architecture

- En VHDL, une structure logique est décrite à l'aide d'une entité et d'une architecture
  1. L'entité décrit les entrées/sorties
  2. La partie architecture contient les instructions VHDL permettant de réaliser le fonctionnement attendu.

ENTITE

ENTITY *Nom de l'entité* **IS**

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'entity* ;

ARCHITECTURE

ARCHITECTURE *Nom de l'architecture* **OF** *Nom de l'entité* **IS**

Zone de déclaration.

BEGIN

Description de la structure logique.

END *nom de l'architecture* ;

Structure d'une description VHDL

Entité

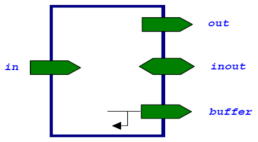
- Le bloc `entity` définit le nom d'un module ainsi que son interface avec le monde extérieur

```
1  ENTITY nom_de_l_entité IS
2  generic (
3          paramètre1 : type := valeur_par_défaut;
4          paramètre2 : type := valeur_par_défaut;
5          ...
6  );
7  port (
8          port1 : direction type;
9          port2 : direction type;
10         ...
11 );
12 end nom_de_l_entité;
```


Structure d'une description VHDL

Les modes des ports

- VHDL définit quatre modes qui déterminent le sens de transfert d'une donnée au travers du port
  1. `in` définit un signal d'entrée l'extérieur
  2. `out` fournit un signal de sortie, mais ne peut pas relire ce signal
  3. `inout` Le signal est bidirectionnel : en sortie, il est fourni par l'entité ; en entrée, il est fourni par l'extérieur
  4. `buffer` un signal utilisable en sortie, qui peut aussi être relu par l'entité comme un signal interne



Plan

Types d'objets

Types prédéfinis

- Types prédéfinis proposés par le langage VHDL initial

Nom du type	Définition de l'ensemble
BIT	Deux valeurs possibles '0' ou '1'
INTEGER	Entiers (nombre positif ou négatif sur 32 bits)
REAL	Réels
BOOLEAN	Deux valeurs possibles True ou False
CHARACTER	Caractères a, b, c ..., 1, 2 ...
TIME	Nombre réel de temps fs, ps ..., min, hr.

- Types prédéfinis dans la bibliothèque standard IEEE

Nom du type	Définition
`std_logic`	Défini dans le paquetage `ieee.std_logic_1164`
`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

Types d'objets

Le type std_logic

- Le type `std_logic` est un type énuméré qui permet de représenter les valeurs logiques d'un signal numérique
- Il est défini dans le package `std_logic_1164`, qui fait partie de la bibliothèque IEEE
- Il peut prendre neuf valeurs différentes, qui sont :
  - `'0'` : niveau logique bas
  - `'1'` : niveau logique haut
  - `'Z'` : état haute impédance
  - `'X'` : état inconnu ou indéterminé
  - `'U'` : état non initialisé
  - `'W'` : état faible inconnu
  - `'L'` : état faible bas
  - `'H'` : état faible haut
  - `'-'` : état indifférent ou non affecté

Types d'objets

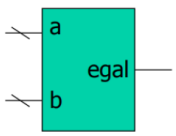
Le type std_logic_vector

- Le type `std_logic_vector` est défini dans le paquetage `ieee.std_logic_1164`
- Il est basé sur le type `std_logic`

Types d'objets

Exemple

```
ENTITY comparateur IS
        PORT(SIGNAL a : IN  std_logic_vector(7 DOWNTO 0);
        SIGNAL b : IN  std_logic_vector(7 DOWNTO 0);
        SIGNAL egal : OUT  std_logic);
END comparateur;
```



Opérateurs élémentaires

Opérateurs élémentaires

Classe	Opérateurs	Types d'opérandes	Résultat
Op. logiques	`and or nand nor xor`	bits ou booléens	bit ou booléen
Op. relationnels	= /= < <= > >=	tous types	booléen
Op. additifs	+ - &	numériques tableaux (concaténation)	numérique tableau
Signe	+ -	numériques	numérique
Opérateurs multiplicatifs	/ `mod rem`	numériques (restrictions) entiers (restrictions)	numérique entier
Op. divers	`not abs`	bit ou booléen numérique	bit ou booléen numérique

- Les opérateurs multiplicatifs et l'opérateur d'exponentiation (**) sont soumis à des restrictions, notamment en synthèse où seules les opérations qui se résument à des décalages sont généralement acceptées.
- Certaines librairies (`numeric_bit` de la librairie IEEE) surdéfinissent (au sens des langages objets) les opérateurs d'addition et de soustraction pour les étendre au type `bit_vector`, par exemple.
- Tous les opérateurs logiques ont la même priorité, il est donc plus que conseillé de parenthéser toutes les expressions qui contiennent des opérateurs différents de cette classe.

Types personnalisés

Définir un nouveau type

- Types d'entiers

```
type type_name is range range_specification;
```

Exemple

```
type temperature is range e to 273;
type smalljnteger is range -32 to 31;
```

- Types énumératifs

```
type type_name is (liste de valeurs);
```

Exemple

```
type boolean is (false, true);
type bit is ( ' a ', ' 1' );
type std_ulogic is ('U','X','a','1','Z','W','L','H','-');
```


Styles de descriptions

Architecture

- Le corps d'architecture décrit le fonctionnement interne du bloc logique
- Syntaxe

```
architecture nom_architecture of nom_entité is
        déclaration_de_composant
        | déclaration_de_constante
        | déclaration_de_signal_interne
        | déclaration_de_type
        | déclaration_d_alias
begin
        instruction_concurrente_d_assignation_de_signal
        | instruction_concurrente_d_instanciation_de_composant
        | instruction_concurrente_de_processus
        | instruction_de_génération
end [architecture] [nom_architecture];
```


Styles de descriptions

Styles de descriptions

- VHDL offre trois styles de description d'un circuit
  1. Par flot de données : spécifier des assignations de signaux concurrents, choisies et conditionnelles
  2. Structurel : décrire la structure du circuit (instanciation de composants)
  3. Comportemental : décrire le comportement (par un algorithme)

Plan

Description par flot de données

Présentation

- Utiliser des instructions concurrentes d'affectation de signaux.
- Description de la manière dont les données circulent de signal en signal, ou d'une entrée vers une sortie
- L'ordre d'écriture des instructions d'affectation de signaux est quelconque
  - Instructions concurrentes
- Trois types d'instructions d'affectation

Description par flot de données

Affectation d'un signal

- Syntaxe de l'affectation d'un signal

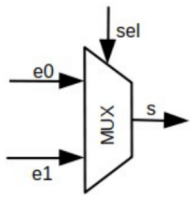
```
signal <= expression;
```

  - L'affectation représente un lien définitif entre le signal et le circuit générant l'expression (connexion)
  - L'affectation du signal ne modifie pas la valeur courante mais les valeurs futures
  - Affecter une expression à un signal correspond à connecter un signal sur la sortie d'une porte

Description par flot de données

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 ;
```



Description par flot de données

Affectation conditionnelle

- L'interconnexion est soumise à des conditions

```
SIGNAL <= expression when condition_1
        [else expression when condition_2]
        ...
        [else expression when condition_n]
        [else expression];
```

  - L'instruction `[else expression]` n'est pas obligatoire mais elle est fortement conseillée, elle permet de définir la valeur du `SIGNAL` dans le cas où la condition n'est pas remplie

Description par flot de données

Exemple d'affectation conditionnelle : description de la porte logique et

```
library ieee;
use ieee.std_logic_1164.all;
entity et_logique is port (
        E1,E2 : in std_logic ;
        S : out std_logic);
end entity;
architecture arch_et_logique of et_logique is
begin
        S <= E2 when ( E1= '1') else '0';
end architecture;
```


Description par flot de données

Affectation sélective

- Affecter différentes valeurs à un signal, selon les valeurs prises par un signal dit de sélection

```
with SIGNAL_DE_SELECTION select
        SIGNAL <= expression when valeur_de_selection_1,
                  [expression when valeur_de_selection_2,]
                  ...
                  [expression when valeur_de_selection_n,]
                  [expression when others];
```

  - L'instruction `[expression when others]` n'est pas obligatoire mais fortement conseillée, elle permet de définir la valeur du `SIGNAL` dans le cas où la condition n'est pas remplie.

Description par flot de données

Exemple d'affectation sélective : description d

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity mux4 is
port(d0,d1,d2,d3: in std_logic_vector(3 downto 0);
        s: in std_logic_vector(1 downto 0);
        y: out std_logic_vector(3 downto 0));
end;
architecture synth1 of mux4 is
begin
with s select y <=
        d0 when "00",
        d1 when "01",
        d2 when "10",
        d3 when others;
end;
end architecture;
```


Description par flot de données

Configuration d'une architecture

- Pour la même `entity`, on peut proposer plusieurs architectures
  - Proposer plusieurs versions pour choisir par la suite la meilleure
- Lors de la synthèse ou de la simulation, une seule architecture est utilisée

```
configuration nom_configuration of nom_entité is
        for nom_architecture
        end for;
end [configuration] [nom_configuration];
```

Description par flot de données

Configuration d'une architecture

```
library ieee;
use ieee.std_logic_1164.all;
entity multi_arch is
  port(a,b,c : in std_logic;
  s: out std_logic);
end entity;

architecture version1 of multi_arch is
begin
  s<=(a and b and c);
end architecture;
architecture version2 of multi_arch is
begin
  s<=c when a='1' and b='1' else '0';
end architecture;

configuration cfg of multi_arch is
  for version1 -- select the version
  end for;
end configuration;
```


Plan

Description structurelle

Présentation

- Description de type hiérarchique par liste de connexions
- Permet de construire une description à partir d'autres composants de manière hiérarchique
- Le nombre de niveaux de hiérarchie est quelconque
- Une description est structurelle au sens VHDL si elle comporte un ou plusieurs composants (component)

Description structurelle

Présentation

- Une description structurelle s'effectue en trois étapes

1. Déclarer
   - Un composant (COMPONENT) : Support pour le câblage
   - Une liste de signaux (SIGNAL) nécessaires au câblage
2. Instancier
   - Chaque composant en fixant les paramètres (GENERIC MAP) et le câblage (PORT MAP)
3. Configurer
   - Choisir pour chaque composant instancié le modèle correct (couple Entité-architecture). (USE)

Description structurelle

Déclaration d'un composant

- La déclaration d'un composant s'effectue dans la partie déclarative de l'architecture

```
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 ;
```

- Exemple

```
component MUX port (
  sel,e0,e1 : in std_logic;
  s : out std_logic);
end component;
```


Description structurelle

Instanciation d'un composant

- L'instanciation d'un composant se fait entre `begin` et `end` de l'architecture
- Il est nécessaire de prévoir des signaux pour les connecter aux entrées sorties du composant instancié

```
nom de instance : nom de l'entité décrivant le composant
GENERIC MAP (paramètre1 => valeur,...,paramètre_n => valeur)
PORT MAP (nom du port => signal,.., nom du port => signal) ;
```

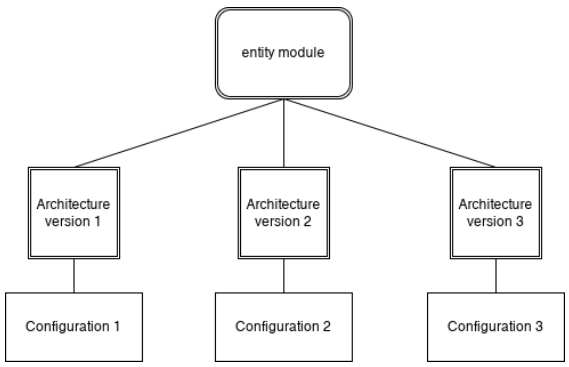
- Exemple

```
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);
  ...
end architecture;
```


Description structurelle

Spécification d'une configuration

- Choisir l'architecture à utiliser dans le cas où plusieurs architectures sont définies



Description structurelle

Spécification d'une configuration

- La spécification d'une configuration s'effectue dans la partie déclarative de l'architecture

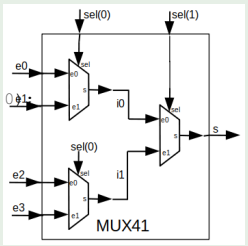
```
1 FOR liste_noms_composants:
2   USE ENTITY nom_entité(nom_architecture)
3     [GENERIC MAP(liste_associations_paramètres)]
4     [PORT MAP(liste_associations_ports)]
```

☞ `liste_noms_composants` représente les étiquettes des instances, ou `all`, ou `others`

Description structurelle

Exemple : description d'un multiplexeur 4-1

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  entity MUX4 is port (
4  e0,e1,e2,e3 : in std_logic ;
5  sel : in std_logic_vector(1 downto 0);
6  s : out std_logic);
7  end entity;
8  component MUX port (
9  sel,e0,e1 : in std_logic;
10 s : out std_logic);
11 end component;
12 architecture arch_mux4 of MUX4 is
13 signal i0,i1 : std_logic;
14 begin
15 inst0 : MUX port map(sel=>sel(0),e0=>e0,e1=>e1,s=>i0);
16 inst1 : MUX port map(sel=>sel(0),e0=>e2,e1=>e3,s=>i1);
17 inst2 : MUX port map(sel=>sel(1),e0=>i0,e1=>i1,s=>s);
18 end architecture;
```



Description structurelle

Spécification d'une configuration

- La sélection d'une architecture peut s'effectuer lors de l'instanciation

```
label: entity work.entity_name (nom_architecture)
    generic map (...) port map (...) ;
```


Description structurelle

Instruction `for...generate`

- Permet de dupliquer un bloc d'instructions concurrentes un certain nombre de fois
- Syntaxe

```
label: FOR variable IN gammeDeValeurs GENERATE
  instructions concurrentes
END GENERATE [label];
```

- Exemple

```
signal a, b, x: std_ulogic_vector(7 downto 9);
...
gen: for i in 0 to 7 generate
  x(i) <= a(i) xor b(7-i);
end generate;
```


Description structurelle

Instruction `if...generate`

- Permet de créer un bloc si une condition est vérifiée
  - La condition doit être connue au préalable
- Syntaxe

```
label: if condition generate
  instructions concurrentes
[elsif condition generate
  instructions concurrentes]
[else generate
  instructions concurrentes]
end generate [label];
```

Description structurelle

Exemple : instruction `if...generate`

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  entity and_or is
4    generic(mode : integer range 0 to 1:=0);
5    port(a,b : in std_logic;
6    s : out std_logic);
7  end entity;
8
9  architecture if_and_or of and_or is
10 begin
11   mycase: if mode=0 generate
12     s<=a and b;
13   else generate
14     s<=a or b;
15   end generate;
16 end architecture;
```


Description structurelle

Instruction `case...generate`

- Permet de créer selon la valeur d'une expression
  - La valeur de l'expression doit être connue au préalable
  - Uniquement à partir de VHDL2008
- Syntaxe

```
1  label: case expression generate
2  when choice =>
3    instructions concurrentes
4  when choice =>
5    instructions concurrentes
6    ...
7  [when other=>
8    instructions concurrentes]
9  end generate [label];
```

- Exemple

Description structurelle

Exemple : instruction `case...generate`

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  entity and_or is
4    generic(mode : integer range 0 to 1:=0); -- OR ou AND
5    port(a,b : in std_logic;
6    s : out std_logic);
7  end entity;
8
9  architecture gen_and_or of and_or is
10 begin
11   mycase: case mode generate
12   when 0 =>
13   s<=a and b;
14   when others =>
15   s<=a or b;
16   end generate;
17 end architecture;
```


Plan

Description comportementale

Présentation

- On décrit le comportement du circuit de façon algorithmique
  - Définition des processus
  - Abstraction du contenu matériel du circuit
- Un processus est considéré comme une instruction concurrente
  - L'ordre d'écriture des instructions d'appel de processus est quelconque.
- Un processus contient un algorithme
  - Un processus contient des instructions séquentielles qui ne servent qu'à traduire simplement et efficacement le comportement d'un sous-ensemble matériel.

Description comportementale

Présentation

- Déclaration d'un appel d'un processus

```
[label:] Process(liste de sensibilité)
Déclarations
begin
        Instructions séquentielles
        ........
end Process [label];
```

  - La liste de sensibilité est la liste des signaux qui déclenchent, par le changement de valeur de l'un quelconque d'entre eux, l'activité du processus. Cette liste peut être remplacée par une instruction « wait » dans le corps du processus

```
wait [on liste_de_signaux ] [until condition ] ;
```


Description comportementale

Variables

- Les variables sont des objets qui servent à stocker un résultat intermédiaire pour faciliter la construction d'un processus
  - Une variable ne correspond à rien de physique
- Affectation d'une variable

```
variable := expression;
```

  - L'affectation de la variable est instantanée, ensuite il n'existe plus aucun lien entre la variable et l'expression

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 effectuée à la fin du process	Mise à jour immédiate.
Un point interne	Aucun sens physique

Description comportementale

Instructions séquentielles : if...then

- Les variables sont des objets qui servent à stocker un résultat intermédiaire pour faciliter la construction d'un processus
- Syntaxe

```
IF condition1 THEN
        instructions séquentielles;
[ELSIF condition2 THEN
        instructions séquentielles;]
......
[ELSIF condition3 THEN
        instructions séquentielles;]
[ELSE
        instructions;]
END IF;
```

  - L'affectation de la variable est instantanée, ensuite il n'existe plus aucun lien entre la variable et l'expression

Description structurelle

Exemple : instruction séquentielle `if...then`

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  entity mux21 is
4    port(e0,e1,sel : in std_logic;
5    s : out std_logic);
6  end entity;
7
8  architecture be_mux21 of mux21 is
9  begin
10   process(e0,e1,sel)
11   begin
12     if (sel='1') then
13       s<=e1;
14     else
15       s<=e0;
16     end if;
17   end process;
18 end architecture;
```


Description comportementale

Instructions séquentielles : `case`

- Les variables sont des objets qui servent à stocker un résultat intermédiaire pour faciliter la construction d'un processus
- Syntaxe

```
CASE signal IS
  WHEN valeur_1 => instructions séquentielles;
  [WHEN valeur_2 => instructions séquentielles;]
  ...
  [WHEN valeur_n => instructions séquentielles;]
  [WHEN OTHERS => instructions séquentielles;]
END CASE;
```

  - La description avec l'instruction `CASE` doit obligatoirement avoir le choix `OTHERS` à la fin, si toutes les valeurs de l'expression ne sont pas énumérées.

Description structurelle

Exemple : instruction séquentielle `case`

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  entity mux41 is
4    port(sel:in std_logic_vector(1 downto 0);
5    e: in std_logic_vector(3 downto 0);
6    s : out std_logic);
7  end entity;
8  architecture be_mux41 of mux41 is
9  begin
10   process(sel,e)
11   begin
12     case sel is
13       when "00" => s<=e(0);
14       when "01" => s<= e(1);
15       when "10" => s<=e(2);
16       when "11" => s<=e(3);
17       when others => s<='X';
18     end case;
19   end process;
20 end architecture;
```


Description comportementale

Instructions séquentielles : `for...loop`

- Répéter un bloc d'instructions un nombre déterminé de fois, en faisant varier une variable d'itération selon un intervalle donné
- Syntaxe

```
FOR compteur IN gammeDeValeurs LOOP
        instructions séquentielles;
END LOOP;
```

  - La déclaration du compteur et son type sont implicitement déclarés par la gamme de valeurs de la boucle

Description structurelle

Exemple : instruction séquentielle `for...loop`

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  entity ex_for is
4    generic(N: natural range 1 to 32:=16);
5    port(e: in std_logic_vector(N-1 downto 0);
6    s : out std_logic);
7  end entity;
8  architecture be_ex_for of ex_for is
9  begin
10   process(e)
11   begin
12     s<='1';
13     for i in 0 to N-1 loop
14       if e(i)='0' then
15         s<='0';
16       end if;
17     end loop;
18   end process;
19 end architecture;
```


Description comportementale

Instructions séquentielles : `WHILE...LOOP`

- Répéter un bloc d'instructions tant qu'une condition est vérifiée
- Syntaxe

```
WHILE ( condition de répétition ) LOOP
        instructions séquentielles;
END LOOP;
```

  - La déclaration du compteur doit être effectuée

Avertissement

`WHILE...LOOP` n'est pas synthétisable

MERCI POUR VOTRE ATTENTION

Questions ?