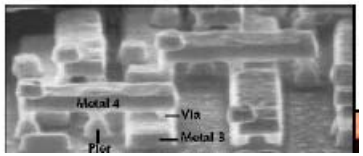
Département EEA – Master 1 EEA - Parc Valrose 06104 Nice

VHDL :

Notions avancées et Synthèse



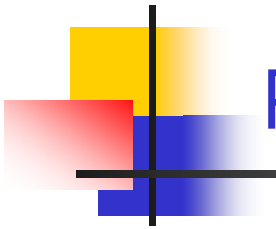
C. Belleudy

Compléments: C. André

Université Nice SOPHIA ANTIPOLIS

CNRS CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

Laboratoire I3S, UNSA-CNRS 2000, route des lucioles Les Algorithmes - bât. Euclide B, 06903 Sophia Antipolis -France



Plan

1 - Introduction

2 – Concepts de base, types, opérateurs

3 - Instructions concurrentes

4 - Instructions séquentielles

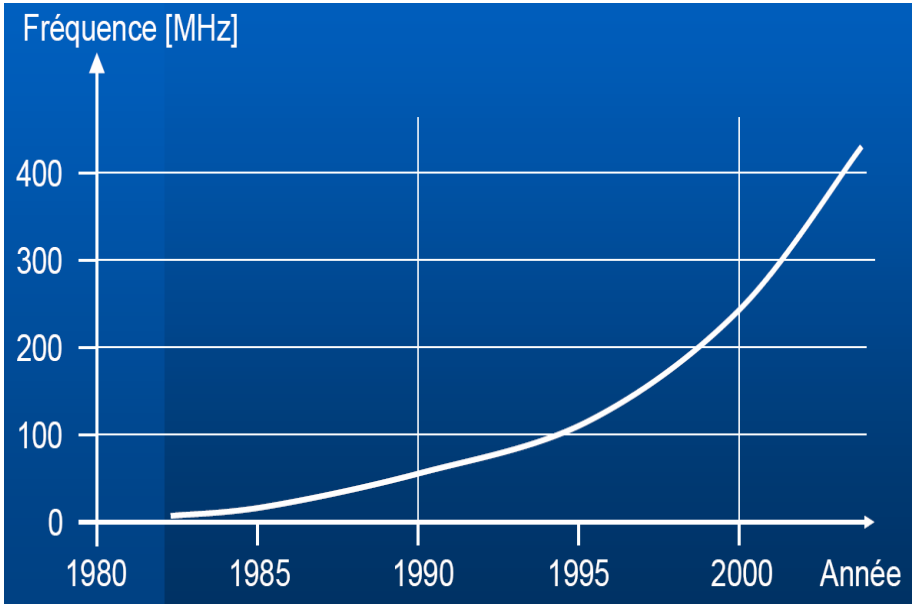
5 - Bibliothèques et sous programmes

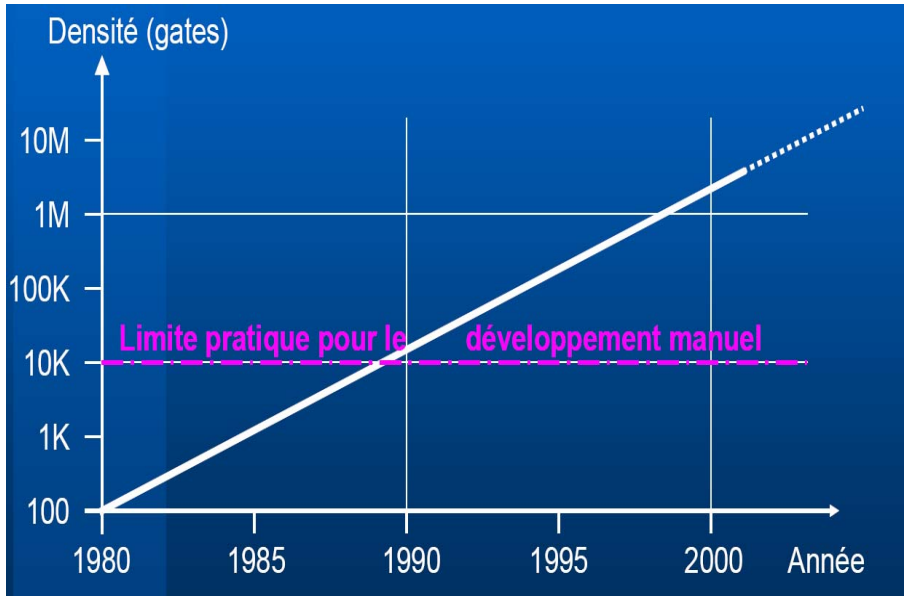
6 - Simulation d'un programme VHDL

7 - Synthèse

Concevoir des circuits

	2001	2003	2005	2008	2011	2014
Minimum **Feature Size**		.13µm	.10µm	70nm	50nm	35nm
Max Number of **Package Pins**	1,792	2,518	3,158	4,437	6,234	8,758
Memory						
DRAM Chip **Complexity** (bit)	2G	4G	8G	-	64G	-
Cost (10^{-6} USD cents/bit)	7.6	3.8	1.9	-	0.24	-
Logic						
High-Volume Processors **Density** (trans/cm²)	13M	24M	44M	109M	269M	664M
High-Volume Processors **Cost** (10^{-6} USD cents/trans)	525	262	131	-	16	-
High-Performance Processor Chip **Complexity** (trans/chip)	220M	441M	882M	2.5G	7G	20G





Pouquoi un langage de description de matériel ?

- Description performante, lisible, portable.
- Maîtriser le temps de développement.
- Pérennité des descriptions
- Réutilisation des modules déjà développés et testés
- Simulations à différents niveaux de la conception

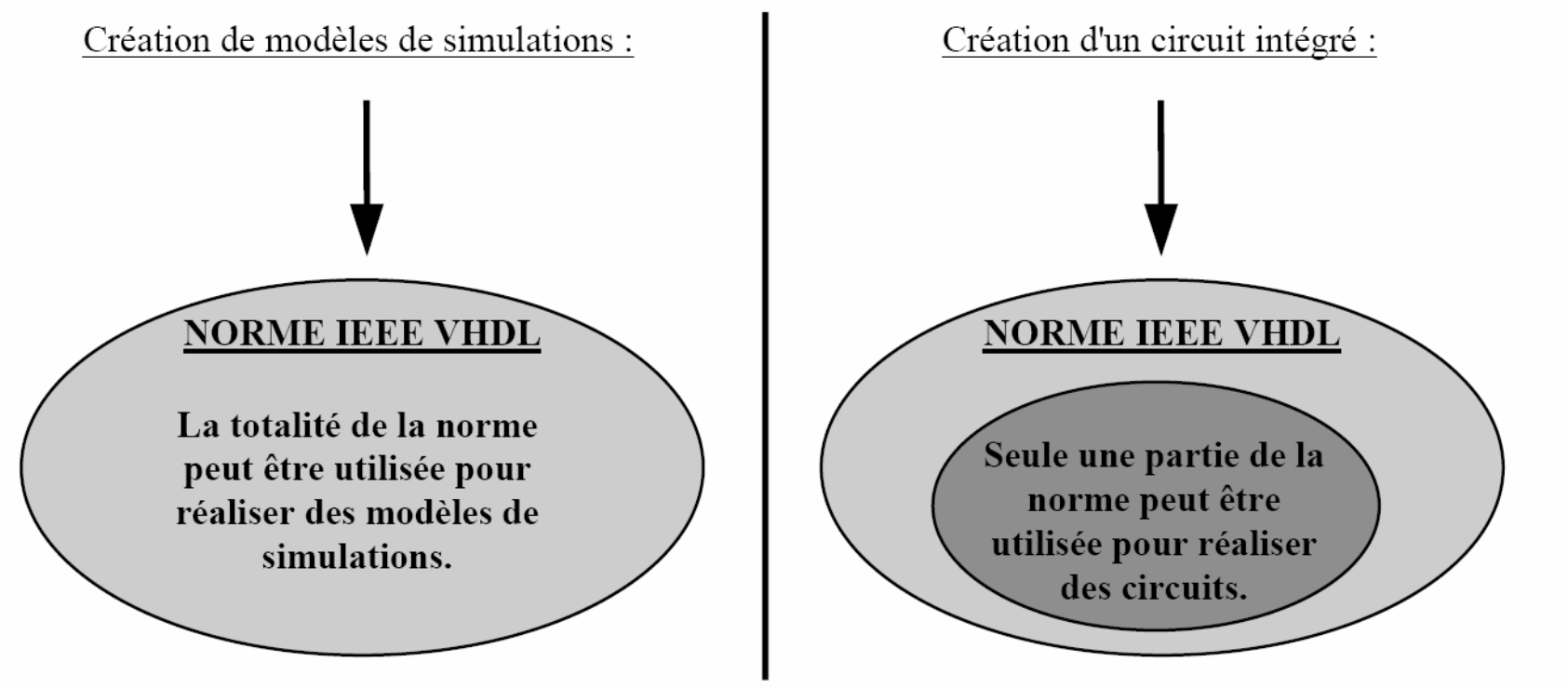
VHDL : VHSIC (Very High Speed Integrated Circuit) Hardware Description Langage

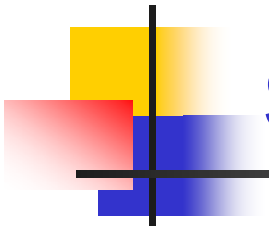
Développer par le DOD (département américain de la défense).

Langage VHDL : objectifs

- *Spécification (cahier des charges),*
- *Conception (comparaison de différents algorithmes ou solutions)*
- *Description pour la synthèse automatique*
- *Simulation fonctionnelle*
- *Vérification des contraintes temporelles*
- *Preuve formelle*
- *Langage unique pour la spécification, la description, la simulation (la synthèse).*
- *Indépendance vis-à-vis du circuit cible (ASIC, FPGA …)*
- *Langage assurant une portabilité des outils,*
- *Langage Hiérarchique Top-down*

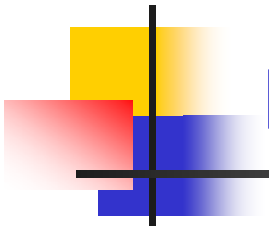
Simulation/Synthèse VHDL





Simulation/synthèse

- Le simulateur :
  - Interprète le langage VHDL
  - Comprend l'ensemble du langage VHDL
- Le synthétiseur :
  - Traduit la description VHDL en une *netlist* logique
  - Comprend uniquement un sous-ensemble du langage VHDL



Plan

1 - Introduction

2 - Concepts de base, types, opérateurs

3 - Instructions concurrentes

4 - Instructions séquentielles

5 - Bibliothèques et sous programmes

6 - Simulation d'un programme VHDL

7 - Synthèse

Syntaxe du langage

Utile pour lire la carte de référence du langage

- Notation BNF (Backus Naur Form)

symbole	signification
::=	est défini par
{ }	0 ou plusieurs répétitions de ce qui est entre les accolades
[ ]	0 ou une fois ce qui est entre les crochets (optionnel)
|	ou (exclusif)

TERMINAUX **( )** **::=**

non terminaux

Exemples:

identifier **::=** letter **{** **[**underline**]** alphanumeric **}**

A, **a**, **int**, **int_32** → legal
1A, **_alpha** → illegal

decimal literal **::=** integer **[**.integer**]** **[**E **[+|-]** integer**]**

12, **12.50**, **0.5E-3** → legal
.5, **0.5f** → illegal

***Remarque**: VHDL est indifférent à la casse (case insensitive):* **Azerty ≡ aZERTY**

Comment décrire un composant : la syntaxe

- Un composant = **entité (ENTITY)**.

Description externe :

- son nom,
- définition des paramètres génériques,
- ses entrées et ses sorties.


```
ENTITY nom_du_composant IS
        GENERIC ( paramètre1 : type_du_paramètre := valeur_par_défaut;
                  paramètre2 : type_du_paramètre := valeur_par_défaut;
                  ...);
        PORT (    nom_signal, ... : direction type := initialisation ;
                  ...
                  nom_signal, ... : direction type );
        END nom_du_composant ;
        ...
```

Comment décrire un composant

- Un composant = **entité (ENTITY)**.

Description externe :

- son nom,
- définition des paramètres génériques,
- ses entrées et ses sorties.


```
ENTITY ENTITYID IS
        [GENERIC ( { ID : TYPEID [:= expr]; });  ]
        [ PORT ( { ID : (in | out | inout) TYPEID [:= expr]; } );  ]
        [ { declaration } ]
[ begin
        { parallel_statement } ]
END [ENTITY ] ENTITYID ;
```

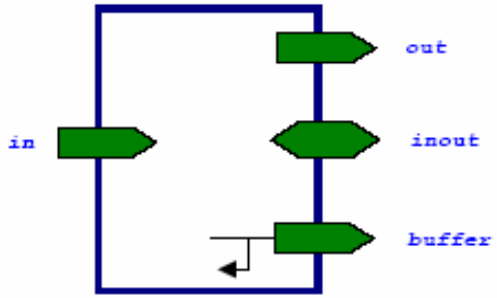
Notation BNF (pour l'exemple)

Comment décrire un composant : la syntaxe


```
ENTITY EX1MEEA IS
PORT (   E1 : IN  std_logic ;
         E2 : IN  std_logic ;
         Sel : IN  std_logic_vector(1 downto 0) ;
         S : OUT  std_logic) ;
END EX1MEEA; ...
```

Rq : après la dernière déclaration, pas de ;

```
ENTITY nom_du_composant IS
        PORT (   nom_signal, ... : direction type := initialisation ;
                 ...
                 nom_signal, ... : direction type );
        END nom_du_composant ;
        ...
```



Type :
bit (0,1), bit_vector,
std_logic (0,1, X, U, ...),
std_logic_vector ...

Comment décrire un composant : Description interne

...

ARCHITECTURE nom_architecture ***OF*** nom_du_composant ***IS***

<<Déclaration des composants internes>>

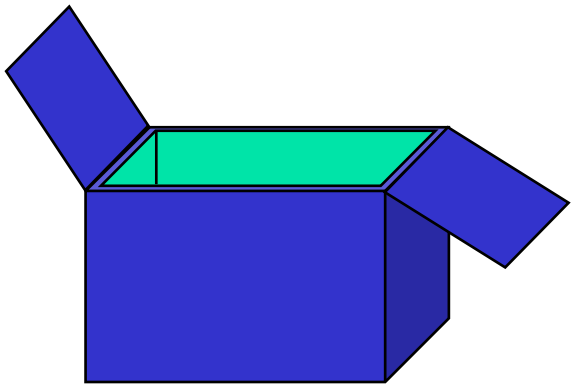
<<Déclaration des types>>

<<Déclaration des signaux>>

BEGIN

<< zone de description du composant>>

END nom_architecture;



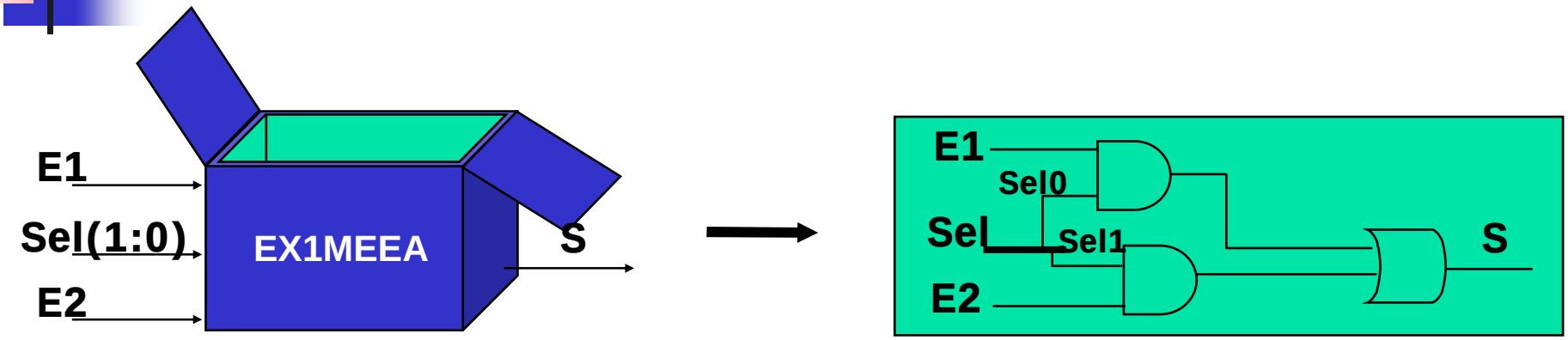
Description flot de données (data flow)

Description structurelle (structural)

Description comportementale (behavioral)

Combinaison de plusieurs types de descriptions

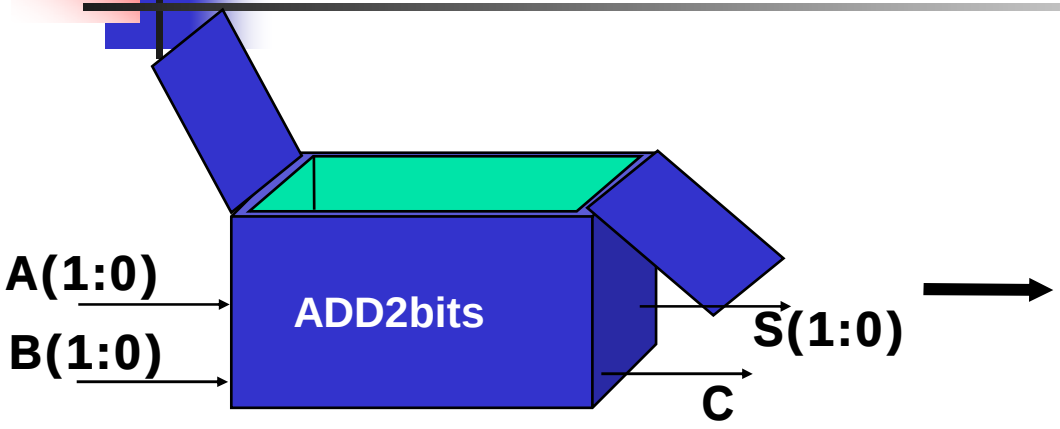
Description flot de donnée : exemple

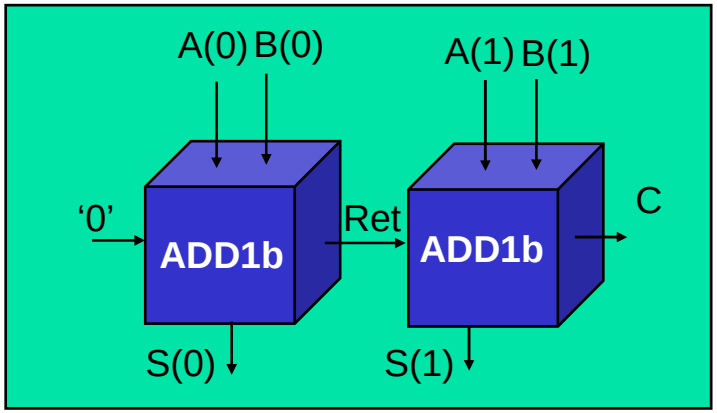


Description flot de donnée :
équivalent au niveau RTL (Register Transfert Langage), les signaux passent à travers des couches d'opérateurs logiques qui décrivent les étapes successives qui font passer des entrées d'un module à sa sortie.

```
ENTITY EX1MEEA IS
PORT (E1,E2 : IN std_logic ;
            Sel : IN std_logic_vector(1 downto 0) ;
            S : OUT std_logic) ;
END EX1MEEA;
ARCHITECTURE arch OF EX1MEEA IS
BEGIN
    S <= (E1 AND SEL(0) OR (E2 AND SEL(1));
END arch;
```

Description structurelle

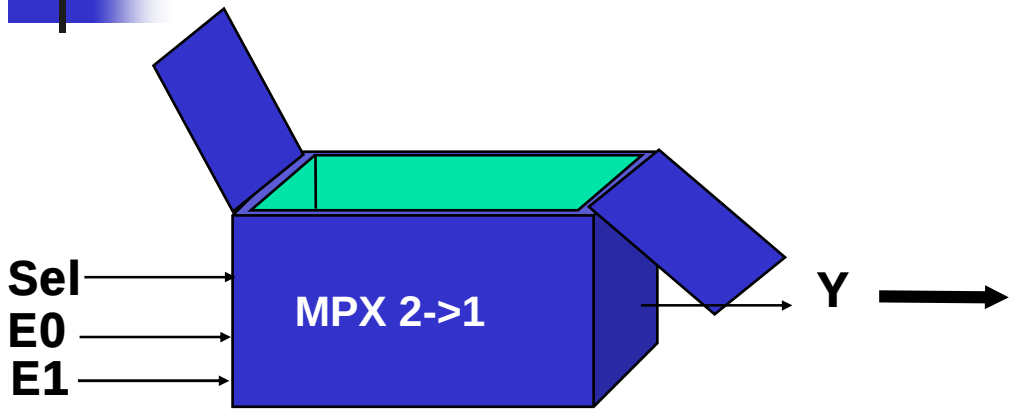




Description structurelle : utilise des composants existant dans une librairie de travail. Le programme instancie les composants et décrit leurs interconnexions.

```
ENTITY add2bits IS
PORT (      A, B : IN std_logic_vector (1 downto 0);
            S : OUT std_logic_vector (1 downto 0);
            C : OUT std_logic);
END add2bits;
ARCHITECTURE arch OF add2bits IS
COMPONENT add1b PORT ( Ai,Bi, Rim1 : IN std_logic;  Si, Ri : OUT std_logic);
END COMPONENT;
SIGNAL Masse, Ret : std_logic;
BEGIN
            Masse <='0';
            U1 : add1b PORT MAP (A(0), B(0), masse, S(0), Ret);
            U2 : add1b PORT MAP (A(1), B(1), Ret, S(1), C);
END arch;
```

Description comportementale



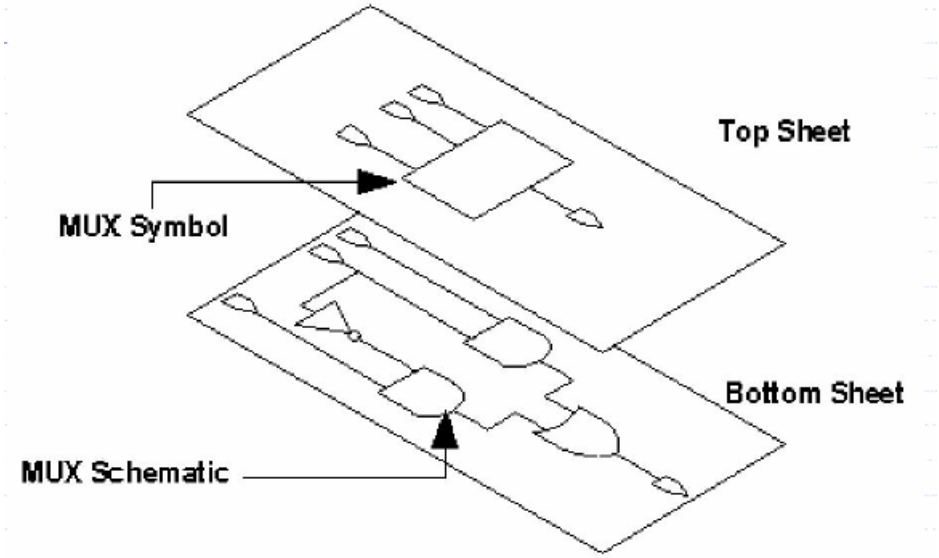
Si Sel= 1
Alors Y= E0;
Sinon Y= E1;

Description comportementale : comprend des blocs d'algorithmes séquentiels exécutés par des processus indépendants. Elle peut traduire un fonctionnement séquentiel ou combinatoire

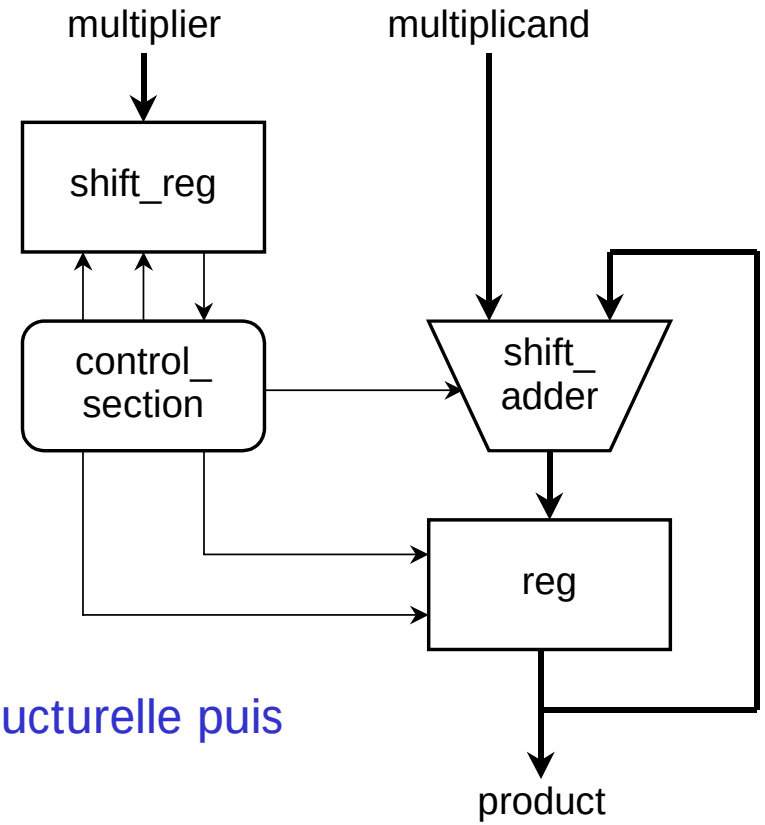
```
ENTITY MPX2 IS
PORT (      E0, E1, Sel : IN std_logic;
            Y : OUT std_logic);
END MPX2 ;
ARCHITECTURE arch OF MPX2 IS
BEGIN
            PROCESS (sel,E0,E1)
            BEGIN
                        IF sel='1'   THEN  Y <= E0;
                                     ELSE  Y <= E1;
                        END IF ;
            END PROCESS ;
END arch;
```

Exercices

Exo1 :



Exo2 :



Décrire ces éléments de façon flot de donnée, structurelle puis comportementale.
Exo1 pour 1 bit puis pour n bits.
Exo2 pour 4 bits puis pour n bits => Pensez circuit.

Eléments de syntaxe : Identificateurs et valeurs littérales

Les commentaires :

Les commentaires débutent par un double tiret.
-- déclaration de signaux interne

Les identificateurs :

Ils se composent d'une suite de lettres ou de chiffres (et éventuellement du caractère '_') débutant par une lettre : A, Horloge1, DATA_OUT.
VHDL ne fait pas de différence, pour les identificateurs et les mots-clés entre majuscules et minuscules.

signal CLK : **bit;** et **SIGNAL** Clk : **BIT**; -- sont identiques.

Les littéraux

Les littéraux sont utilisés pour la représentation de valeurs.
Les plus fréquents :

Entiers : 12 et 0.
Bits : '0' et '1';
Chaines de bits "0110_1110" (le caractère _ est sans action).

Quelques éléments de syntaxe : Identificateurs et valeurs littérales

Les littéraux (suite)

Binaire : Bus <= "1001" ;
Hexadécimale : Bus <= X'9' ; (dans certains outils de synthèse on trouve : 16#9#)
Octale : Bus <= O"11" ;

Chaînes de caractères : "cy122ilk".
Concaténation de chaînes : ’1’ & "10" & ’0’ équivaut à "1100".

Exo 1 : S<= A (7 downto 0) & B(4 downto 1) & "11 ". Déterminer la taille de S ?

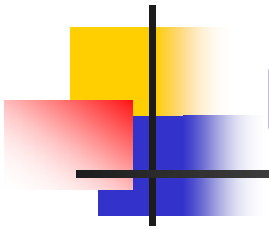
Exo 2 : Décrire un additionneur 4 bits

Entrées : R_{-1}, A, B, Sorties : S, Carry.

Exo 3 : Décalage (gauche ou droite) : concaténation ?

Remarque : la base décimale ne peut pas être utilisée lors de l’affectation des signaux. On peut seulement l’utiliser avec des opérateurs comme le + ou le -.

⇒ ~~A <= 4~~ ; A <= B+4;



Les Types

VHDL : langage fortement typé. Tout objet doit avoir un type défini avant sa création. Le type définit le format des données et l'ensemble des opérations légales sur ces données.

Quatre types :

- les types scalaires : numériques et énumérés (pas de structure interne),
- les types composés : tableau et enregistrement (record) qui possèdent des sous-éléments,
- les types access qui sont des pointeurs,
- le type file qui permet de gérer les fichiers.

A partir d'un type =>possibilité de définir un sous-type par adjonction d'une contrainte et/ou d'une fonction de résolution.

Contrainte : restriction des valeurs possibles à un domaine plus limité que celui qui définit le type de base.

```
SUBTYPE int1 IS INTEGER RANGE expr1 TO expr2;          --(expr2>expr1)
SUBTYPE int2 IS INTEGER RANGE expr3 DOWNTO expr4;      --(expr3>expr4)
```

Exemple : entier de 0 à 255 => *INTEGER RANGE* 0 *TO* 255

Un sous type hérite des opérations définies sur le type de base mais le contrôle du respect de la contrainte sur le résultat est à la charge du programmeur.

Exemple : 200 + 200 = ?

Les types prédéfinis scalaires

Nom du type	Définition de l'ensemble
BIT	Deux valeurs possibles '1 ' ou ' 0 '
BOOLEAN	Deux valeurs possibles True ou False
CHARACTER	Caractères a, b, c ..., 1, 2 ...
INTEGER	Entiers (signés) sur 32 bits (-2, ...*10^9 ... à + 2,...*10^9)
REAL	Réels : -1.0E38 à 1.0E38 A:=2.3 ou 1.4E5 ou 1.4E-5
TIME	Nombre réel de temps : fs, ps, ns, us, ms, sec, min, hr. Codage sur 64 bits. Attention : 45ns => erreur mais 45 ns

Type std_ulogic : (std_logic n'a pas U)

'U' Unresolved
'X' Forcing Unknown (synthesizable unknown)
'0' Forcing Low (synthesizable logic '0')
'1' Forcing High (synthesizable logic '1')
'Z' High impedance (synthesizable tri-state buffer)

'W' Weak unknown
'L' Weak low
'H' Weak high
'–' Don't care

Exercice : Corriger les expressions suivantes

Temps := 768sec; X<=''1112_0000 ''; Y<= 17#ABCD#; Z:= A + .3; T:=3,4;

Conversion de type

- On peut effectuer une conversion de type entre deux types de structures proches.
- Cette conversion n'est autorisée que dans trois cas :
  - Conversion de type à représentation entier ou flottant,
  - Conversion de tableaux.
- Syntaxe :

  Type_vers_lequel_on_veut_convertir(valeur_à_convertir)

- **Exemple :** A,B,C,D : REAL; I : INTEGER;

  A:=B*REAL(I); -- conversion de I en REAL puis *

  I:=INTEGER(C*D); -- entier le plus proche du réel C*D

Déclaration d'un type scalaire

Déclaration d'un type énumération :

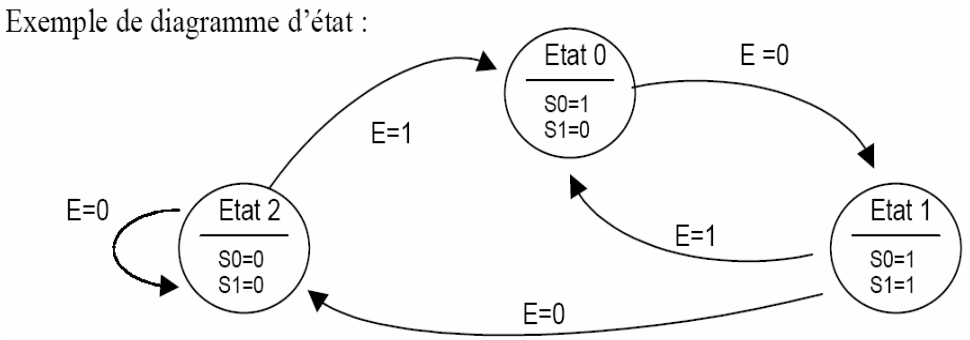
TYPE nom_du _type ***IS*** (val1, val2, ...);

Exemples :

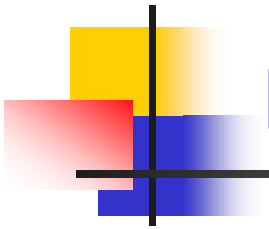
1)TYPE Signal_instruction ***IS*** (MOVE, ADD, SUB, RTS); ⟶ Définition du type
SIGNAL instr : Signal_instruction; ⟶ Création du signal de ce type

2)Machine d'états

TYPE etat ***IS*** (etat0, etat1, etat2);
SIGNAL etat_automate : etat;



Que signifie etat_automate <= etat_automate +1; ou +2 …



Les types scalaires

Valeur d'initialisation par défaut :
premier élément de l'énumération ou la plus petite valeur numérique.

Signal_instruction => MOVE,
INTEGER => le plus petit entier négatif.

- **Exercices :**
  - **Exo 1 :** Machine d'états => exemple du feux rouge.

    Déclarer un signal état correspondant à l'automate de deux feux qui sont sur des voies de circulation opposées.

    Quelle sera sa valeur d'initialisation pour le simulateur ?
  - **Exo 2 :** Donner deux déclarations pour un type *index* correspondant à l'indice d'un tableau de 1 à 15.

    Préciser sa valeur d'initialisation.

Les types structurés : les tableaux

TYPE nom_du_tableau ***IS ARRAY*** (**étendue du tableau**) ***OF***
type_des_éléments_du_tableau (:=initialisation);

Deux types de tableau : (étendue du tableau)

contraint => expr1 ***TO*** expr2 ou expr3 ***DOWNTO*** expr4
non contraint => ***NATURAL RANGE*** <> (paquetage standard)

Exemples :

Tableau à 1 dimension :

TYPE tableau_entier ***IS ARRAY*** (7 ***DOWNTO*** 0) ***OF INTEGER***;
SIGNAL tableau : tableau_entier;

Tableau à 2 dimensions :

TYPE Tableau_8bits ***IS ARRAY*** (7 ***DOWNTO*** 0) ***OF BIT***;
TYPE Tableau_4x8bits ***IS ARRAY*** (3 DOWNTO 0) ***OF*** Tableau_8bits;
Equivalent à :
TYPE Tableau_4x8bits ***IS ARRAY*** (3 ***DOWNTO*** 0, 7 ***DOWNTO*** 0) ***OF BIT***;

Déclaration d'une mémoire :

TYPE mémoire ***IS ARRAY*** (0 ***TO*** m-1, n-1 ***DOWNTO*** 0) ***OF STD_LOGIC***;
SIGNAL M1 : mémoire; … S<=M1(2,0); (exemple 1)

Les types structurés : les tableaux

Déclaration d'une mémoire (suite) :

ou
TYPE mémoire ***IS ARRAY*** (0 ***TO*** m-1) ***OF STD_LOGIC_VECTOR*** (n-1 ***DOWNTO*** 0);
SIGNAL M2 : mémoire; … S<=M1(2); (exemple 2)

Quelques types tableaux prédéfinis :

BIT_VECTOR :

TYPE bit_vector **IS ARRAY** (**NATURAL RANGE** <>) **OF BIT**;

STD_LOGIC_VECTOR :

TYPE std_logic_vector **IS ARRAY** (**NATURAL RANGE** <>) **OF STD_LOGIC**;

Exercices :
- Quel est le type de S dans l'exemple 1 et 2 ?
- Déclarer une table de vérité ayant quatre entrées et trois sorties, ainsi que son contenu (table_dec).
- Déclarer de deux façons différentes une mémoire ROM de dimension 1024*8; Ecrire pour les 1000 premières cases la valeur 0 puis 1.
- Idem pour une mémoire RAM.

Les types structurés : enregistrement (RECORD)

Type constitué d'un ensemble d'éléments qui peuvent être différents, équivalent à des structures en C

```
TYPE nom IS
        RECORD
        déclaration de l'élément1 ;
        déclaration de l'élément 2; …
END RECORD nom;
```

Exemples :

```
TYPE nom_du_mois IS (jan, fev, mar, avr, jun,
jul, aou, sep, oct, nov, dec);
TYPE date IS RECORD
        année : INTEGER;
        mois : nom_du_mois;
        jours : INTEGER ;
END RECORD date;
```

Exemples :

```
TYPE bus_micro IS RECORD
        nbr_bit_adresse : INTEGER;
        nbr_bit_donnee : INTEGER;
        cycle : TIME;
END RECORD bus_micro;
SIGNAL bus : bus_micro;
bus.nbr_bit_adresse <= 24;
bus.nbr_bit_donnee <=12;
bus.cycle <= 60 us;
```

Exercice : Considérons un plan mémoire de 2^P pages et 2^M mots de n bits.
Une case mémoire est caractérisée par un numéro de page, une adresse et un mot.
Donner la déclaration de ce nouveau type, de la mémoire associée. Ecrivez un accès à cette mémoire lecture puis écriture.

Les types access et file

Les types pointeur (access) et fichiers (file) ne correspondent à aucun objet synthétisable dans le circuit.

ACCESS

permet de créer dynamiquement des objets nouveaux.

TYPE nom ***IS ACCESS*** type_pointe;
Exemple :
TYPE pointeur ***IS ACCESS INTEGER***;
VARIABLE A,B : pointeur;
…
A:=***NEW INTEGER*** (18); -- A pointe sur l'entier 18
DEALLOCATE(A) libére la place occupée par l'objet A.

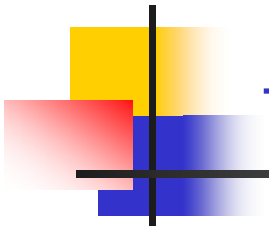
FILE

Donne accès aux procédures d'accès prédéfinies :

FILE_OPEN (); FILE_CLOSE(); READ(); WRITE(); …

Peut-être utile pour charger des valeurs d'initialisation d'une mémoire ou lors de la simulation pour prendre en compte par exemple des fichiers sons, images …
Pour utiliser ces fonctions, il faut introduire la bibiothèque suivante :

USE std.TEXTIO..all;



Trois classes d'objet : constantes, signaux, variables

Déclaration d'une constante :

Valeur non modifiable au cours de l'exécution d'un programme

Déclaration : partie d'une entité, d'une architecture, d'un processus, d'un bloc, d'un paquetage, d'un sous-programme

CONSTANT label **:** type **:=** valeur ;

CONSTANT largeur_du_bus **:** integer **:=** 8;

Déclaration d'un signal :

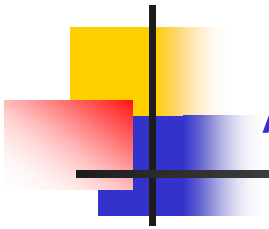
Correspond aux canaux de communication dans un circuit

Déclaration : partie déclarative de l'architecture (plus rarement, de l'entité, d'un bloc ou d'un paquetage)

SIGNAL nom_du_signal **:** type (**:=** valeur initiale) ;

SIGNAL codeop : std_logic_vector (2 downto 0) ou (0 to 2);

Affectation : codeop <= "10" ou codeop(0) <= '0'.



Autre classe d'objet : constantes, signaux, variables

Déclaration d'une variable
(attention pas de correspondance physique) :

Idem langage de programmation, elles servent de stockage de valeurs nécessaires au bon déroulement d'un algorithme.
Déclaration : processus ou sous-programmes (fonctions ou procédures)

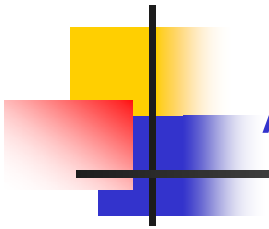
VARIABLE nom_de_la_variable **:** type **:=** valeur initiale ;
VARIABLE nbr_de_boucle : integer := 0 ;
affectation : nbr_de_boucle := 17;

NB : pour les boucles for, il n'est pas nécessaire de déclarer la variable qui sert d'indice de boucle.

Durée de vie des variables :

Fonction : nulle au sens du simulateur,
Processus : du début à la fin de la simulation
=> synthèse ? Problème de portabilité

(une affectation de variable est dynamique, une affectation de signaux est statique ou rémanent)



Autre classe d'objet : constantes, signaux, variables

Exo1 :

Expliquer la différence entre A:=B; et C<=D;

Quelle est la classe de A, B, C, D et leur durée de vie ?

Exo2 :

```
ENTITY bascule IS
PORT ( CLK : IN std_logic ;
       d : IN std_logic ;
       q : OUT std_logic);
END bascule;
```

Quelle est la classe de CLK, d, Q, valeur d'initialisation ?

Autre : les alias

Alias :

Permet de désigner par un autre nom un objet qui existe déjà. => synonyme.
La classe de l'objet n'est pas modifiée.

ALIAS newname : subtype ***IS*** nom_de_l_objet_renommé ;

Exemple :

```
SIGNAL instr : bit_vector(0 TO 15);
ALIAS opcode : bit_vector(0 TO 9) IS instr (0 TO 9) ;
ALIAS source : bit_vector(2 DOWNTO 0) IS instr (10 TO 12);
ALIAS destination : bit_vector(2 DOWNTO 0) IS instr (13 TO 15);
```

Affectation

- Affectation à délai nul : A **<=** B;
- Affectation à délai non nul:

  A **<=** B AFTER 10 ns; synthèse

  C **<=** '0', '1' AFTER 20 ns, '0' AFTER 40 ns; simulation

  Dessiner le chronogramme de A.

- Affectation et modèle de transmission

  Modèle inertiel: filtre les impulsions dont la durée est inférieure au temps précisé.

  A<= B ***AFTER*** 20 ns ;

  Modèle transport : transmission de toute impulsion.

  A<= ***TRANSPORT*** B ***AFTER*** 20 ns;

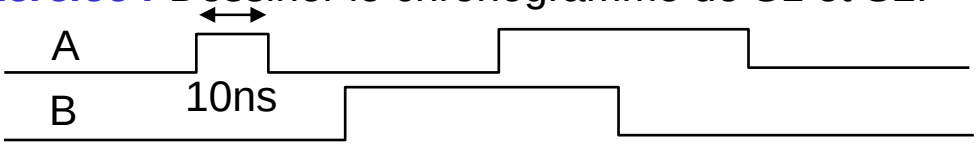
  Reject : rejet de signaux dont la durée est précisée.

  A <= ***REJECT*** 6ns ***INERTIAL*** B ***AFTER*** 20 ns;

Affectation de variables : A:=B;

Affectation de signaux : C<= D;

Exercice : Dessiner le chronogramme de S1 et S2.



S1 <= A ***OR*** B ***AFTER*** 20 ns ;

S2 <= ***TRANSPORT*** (A ***OR*** B) ***AFTER*** 20 ns;

Affectation

- ***Affectation d'un tableau ou d'une partie :***

```
TYPE tableau IS ARRAY 0 TO 99 OF INTEGER;
VARIABLE TAB1, TAB2 : tableau;
...        TAB1:=TAB2;      TAB1(3 TO 5) :=(1,2,3);
```

- ***Autres exemples :***

Q<= ''101111''; équivalent à Q<=('1','0','1','1','1','1'); ou
Q<=(5=>'1', 4=>'0',3=>'1',2=>'1',1=>'1',0=>'1'); ou
Q<= (5=>'1', 3 downto 0 =>'1', others =>'0');

Pour mettre tous les bits à un état : Q <=(others=>'Z');

Les opérateurs: de décalage et de rotation

SLL, SRL, SLA, SRA, ROL, ROR.

Ces six opérateurs travaillent sur des tableaux à une dimension de type booléen.

SXL : décalage gauche ou droite logique.
SXA : décalage gauche ou droite arithmétique,
ROX : rotation à gauche ou à droite.

Exemple :

S <= B **SLL** 3 ;

→ ”1000 1010” ***SLL*** 3 → ”0101 000”
→ ”1000 1010” ***SLL*** (-2) → ”0010 0010” → ”1000 1010” ***SRL*** 2 **??**
→ ”**0**000 1010” ***SLA*** 3 → ”0101 0**000**”
→ ”**1**000 1010” ***SLA*** 3 → ”0101 0**111**”

Exercices :

S0 <= B srl 3; -- Réécrire avec la concaténation sur n bits idem avec sra
S1 <= A sll 2 ; -- Réécrire avec la concaténation sur n bits idem avec sla
S2 <= B rol 3;
”0100 1011” ***SRA*** 3 → ? , ”1100 1011” ***SRA*** 3 →
Idem avec les instructions ROL, ROR.

Les opérateurs

Les opérateurs logiques

NOT, AND, OR, NAND, NOR, XOR, XNOR.
Ils sont prédéfinis pour des opérandes de type booléen.

Les opérateurs relationnels

=, /=, <, <=, >, >=.

Le résultat de ces opérateurs est de type booléen.

Attention : S<=(A=B) Format de S ?

Les opérateurs arithmétiques

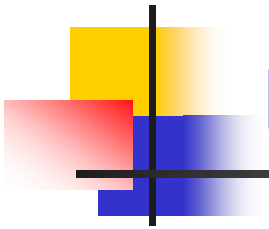
+, - , *, **(Exposant), /, MOD(modulo), REM(reste), ABS.

S<= A MOD B; → A modulo B avec le signe de B
S <= A Rem B; → A modulo B avec le signe de A

Seuls les opérateurs +,- sont supportés par les outils de synthèse.

Les opérateurs modulo et reste agissent sur des entiers.

MAXplusII : pour les autres fonctions → bibliothèque Lpm



Les opérateurs

Les opérateurs arithmétiques (suite)

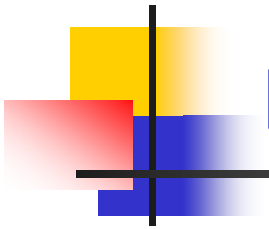
Lors des opérations : Le type d'entrées des opérandes d'entrée doit correspondre à celui de la sortie.
A+B → A et B de type entier → addition sur des entiers
A+B → A et B de type réel → addition sur des réels (gestion de l'exposant, mantisse …)
Ces opérateurs s'appliquent à des opérandes de booléen (+,-), entier ou flottant.

Pour pouvoir utiliser ces opérateurs pour des opérandes de type std_logic ou std_logic_vector, il faut introduire dans le fichier les bibliothèques suivantes :

```
LIBRARY IEEE;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
USE ieee.std_logic_arith.all;
```

Les opérateurs de signe : +,-

- Opérateur unaire
- Rq : les opérateurs de multiplication sont plus prioritaires que les opérateurs de signe: a*-b → erreur; utiliser a*(-b)



Exercices

- *Exo1 : Préciser la fonction réalisée par les instructions suivantes :*
  - S <= speed >273;
  - S <= T_ext >= T_int;
  - S <= B=''00010110'';
  - S<=(E=A) ***OR*** (E=D);

Type de S ?

- *Exo2 : Comparaison attention*
  - Décrire un comparateur =, >, < pour des données signées et non signées.
- *Exo3 : UAL (ALU)*
  - Décrire une ALU opérant sur des std_logic_vector;.

Plan

Update 5 fev

1 - Introduction

2 - Concepts de base, types, opérateurs

3 - Instructions concurrentes

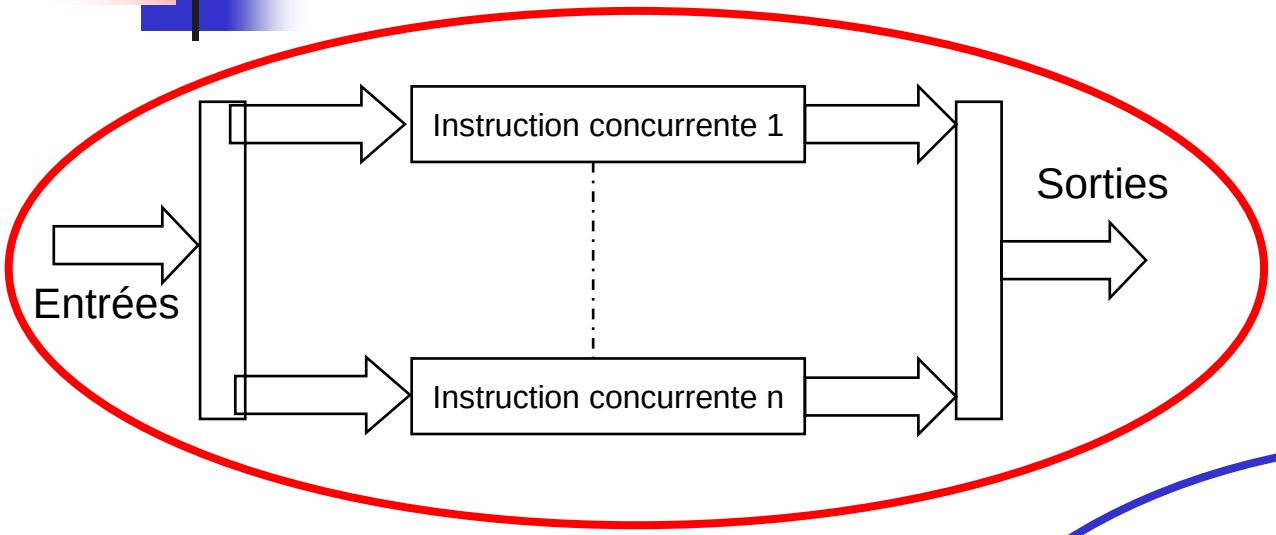
4 - Instructions séquentielles

5 - Bibliothèques et sous programmes

6 - Simulation d'un programme VHDL

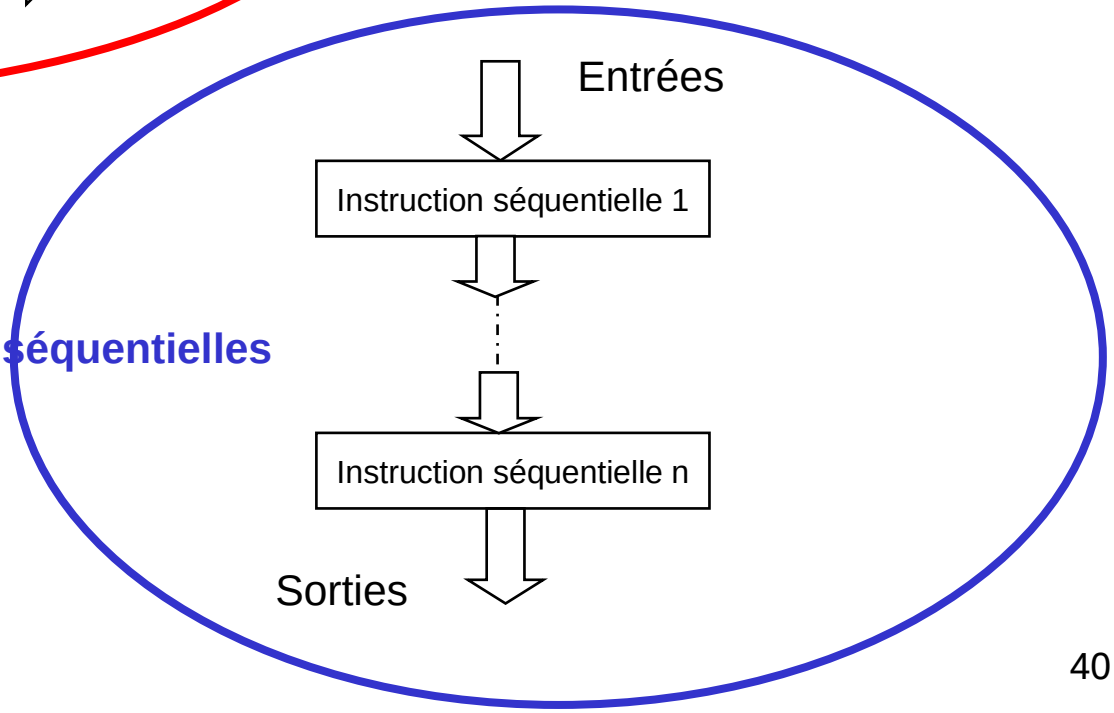
7 - Synthèse

Description concurrente ou processus



Instructions concurrentes

Instructions séquentielles



Description concurrente

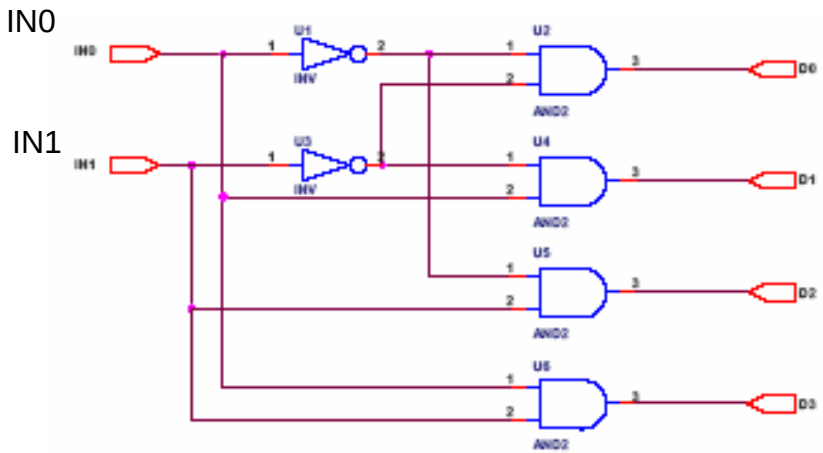
1 - Description concurrente :

Chaque ligne est évaluée indépendamment de l'ordre dans lequel elles sont écrites.
Les instructions sont concurrentes :

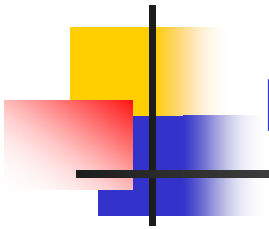
les opérations s'exécutent simultanément et en permanence.

U1 : add1b ***PORT MAP*** (a,b,s);
sel <= d ***AND*** c ***AND*** (***NOT*** s) ;
OUT <= x ***WHEN*** (sel='0') ***ELSE*** y ;
MULT : ***PROCESS*** ...
END PROCESS MULT;

⇒Exécution simultanée des instructions.



⇒ *Donner la description concurrente du schéma suivant.*



Les instructions concurrentes

1 - Affectation simple ou Equations booléennes

A<=B ; C<= B ***AND*** D; ...

2 - Formes conditionnelles

a – Affectation conditionnelle :

signal1 <= signal2 ***WHEN*** *condition* ***ELSE*** *valeur;*

Exemple : description d'un multiplexeur à 4 entrées

```
Y <=    i0 WHEN a='0' AND b='0' ELSE
        i1 WHEN a='1' AND b='0' ELSE
        i2 WHEN a='0' AND b='1' ELSE
        i3 WHEN a='1' AND b='1' ELSE
        'x'; -- indéterminé
```

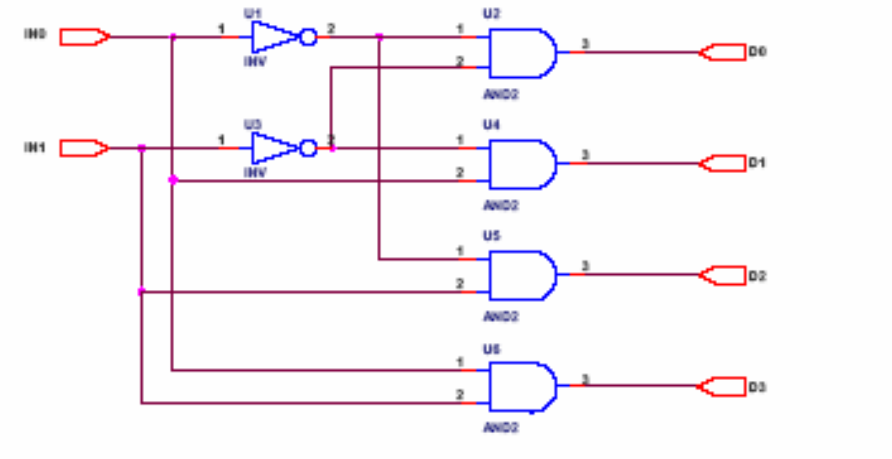
Les instructions concurrentes

b – Affectation sélective :

WITH expression ***SELECT*** signal1 <= signal2 ***WHEN*** condition1,
....
signaln ***WHEN*** condition n,

Si toutes les conditions ne sont pas spécifiées :
signaln ***WHEN*** others;

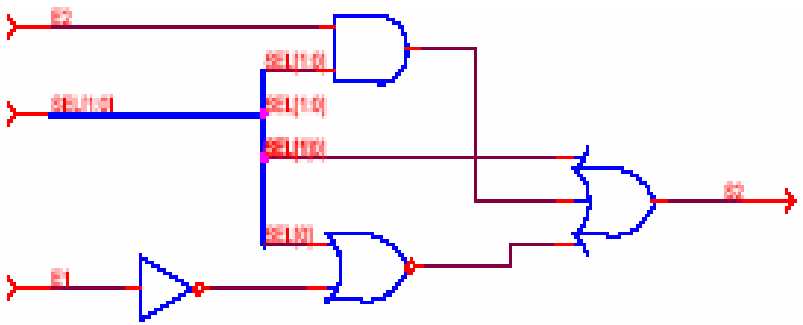
=> Décrire un MPX 4=>1 avec un bus de sélection Sel avec le *WHEN ELSE* et le *WITH SELECT*.



Décrire ce schéma en mode concurrent à l'aide d'instructions d'affectations sélectives ou conditionnelles ?

Les instructions concurrentes

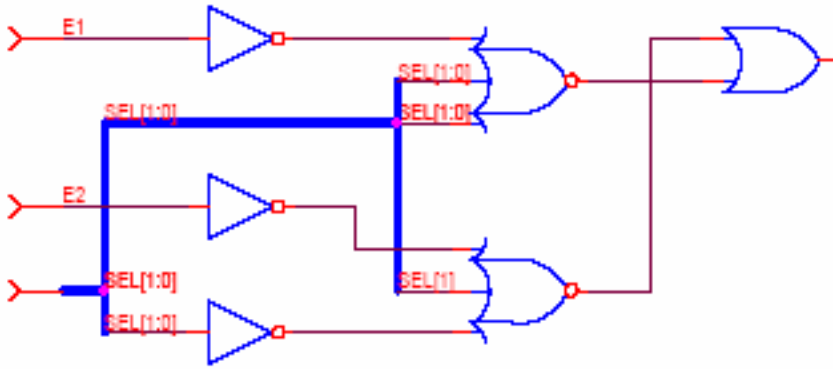
When others : Entrée de forçage à 1



```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;
Use ieee.std_logic_unsigned.all;
entity TABLE2 is
      port (
            E1,E2 : in std_logic;
            SEL   : in std_logic_vector(1 downto 0);
            S2    : out std_logic);
end TABLE2;
architecture DESCRIPTION of TABLE2 is
begin
      with SEL select
            S2 <= E1  when "00",
                  E2  when "01",
                  '1' when others;   -- Pour les autres cas de SEL S2
                                     -- prendra la valeur 1 logique
end DESCRIPTION;
```

Les instructions concurrentes

When others : Entrée de forcage - autre cas




```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;
Use ieee.std_logic_unsigned.all;
entity TABLE3 is
      port (
            E1,E2 : in std_logic;
            SEL   : in std_logic_vector(1 downto 0);
            S2    : out std_logic);
end TABLE3;
architecture DESCRIPTION of TABLE3 is
begin
      with SEL select
            S2 <= E1  when "00",
                  E2  when "01",
                  '-' when others;       -- Pour les autres cas de SEL S2
                                         -- prendra la valeur quelconque
end DESCRIPTION;
```

Cette description correspond-t-elle au schéma ? Dans le cas contraire, donner une nouvelle description.

Hiérarchie et Instanciation de composant

ARCHITECTURE nom_architecture ***OF*** nom_du_composant ***IS***

…

COMPONENT nom_du_composant_appelé

PORT (description externe du composant) => idem celle de l'entité

END COMPONENT;

…

BEGIN

…

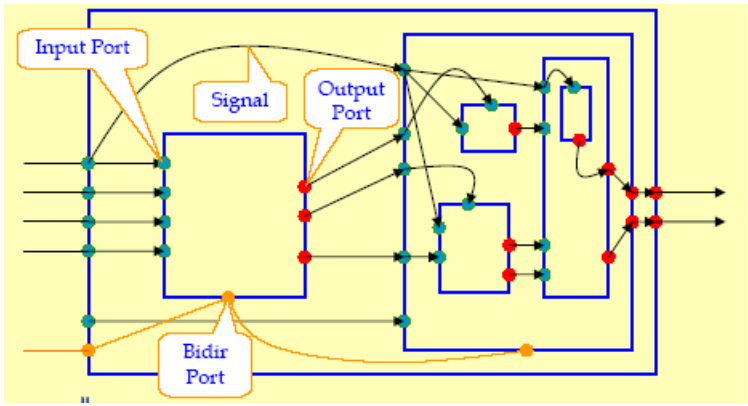
nom_instance : nom_du_composant_appelé

GENERIC MAP (correspondances des paramètres génériques)

PORT MAP (correspondances des ports) ;

…

END nom_architecture;



Exemple :

Réaliser un additionneur pipeline 8 bits en utilisant les composants suivants: add n bits, reg n bits.

Génération répétitive d'instructions

Deux formes : conditionnelle ou itérative

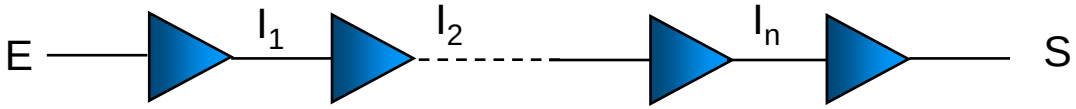
```
        label : IF condition_booléenne  (ou FOR index IN intervalle_discret) GENERATE
                    instructions concurrentes;
        END GENERATE label;
END arch;
```

Exemple :

```
ARCHITECTURE arch OF mpx16 IS
BEGIN
        mpx_gen : FOR i IN 0 TO 15 GENERATE
        mpxu : mpx21 PORT MAP (E0(i),E1(i),SEL,S(i));
        END GENERATE mpx_gen;
END arch;
```

Dessiner le schéma correspondant

On donne le schéma ci-dessous. Décrire ce composant en VHDL en utilisant l'instruction generate. Idem pour un registre à décalage.



Déclaration d'un processus

Il est possible de déclarer plusieurs ***PROCESS*** qui sont alors concurrents :

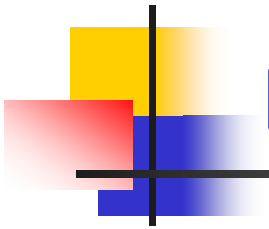
```
nom_process1 : PROCESS (liste sensible1)
        BEGIN .....
        END PROCESS nom_process1;
        nom_process2 : PROCESS (liste sensible2)
        BEGIN .....
        END PROCESS nom_process2; ....
```

Attention :

```
basc_p1 : PROCESS (clk) BEGIN
          IF (clk'EVENT  AND  clk='1') THEN q<=d;
          END IF;
         END PROCESS basc_p1;
basc_p2 : PROCESS (RAZ) BEGIN
          IF (RAZ='1') THEN  q<='0'; END IF;
         END PROCESS basc_p2;
```

Erreur !! Deux (ou plus) processus ne peuvent affecter le même signal.
L'état d'un signal est habituellement modifié par un et un seul process.

Néanmoins, les signaux qui disposent d'une fonction de résolution peuvent être modifié par plusieurs process.
La fonction de résolution calcule l'état des signaux en fonction des valeurs imposées par les process agissant sur le signal.



Plan

1 - Introduction

2 – Concepts de base, types, opérateurs

3 - Instructions concurrentes

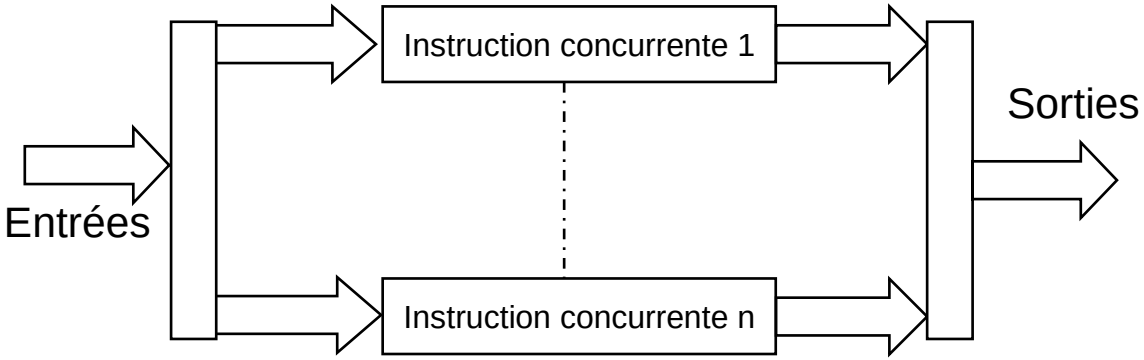
4 - Instructions séquentielles

5 - Bibliothèques et sous programmes

6 - Simulation d'un programme VHDL

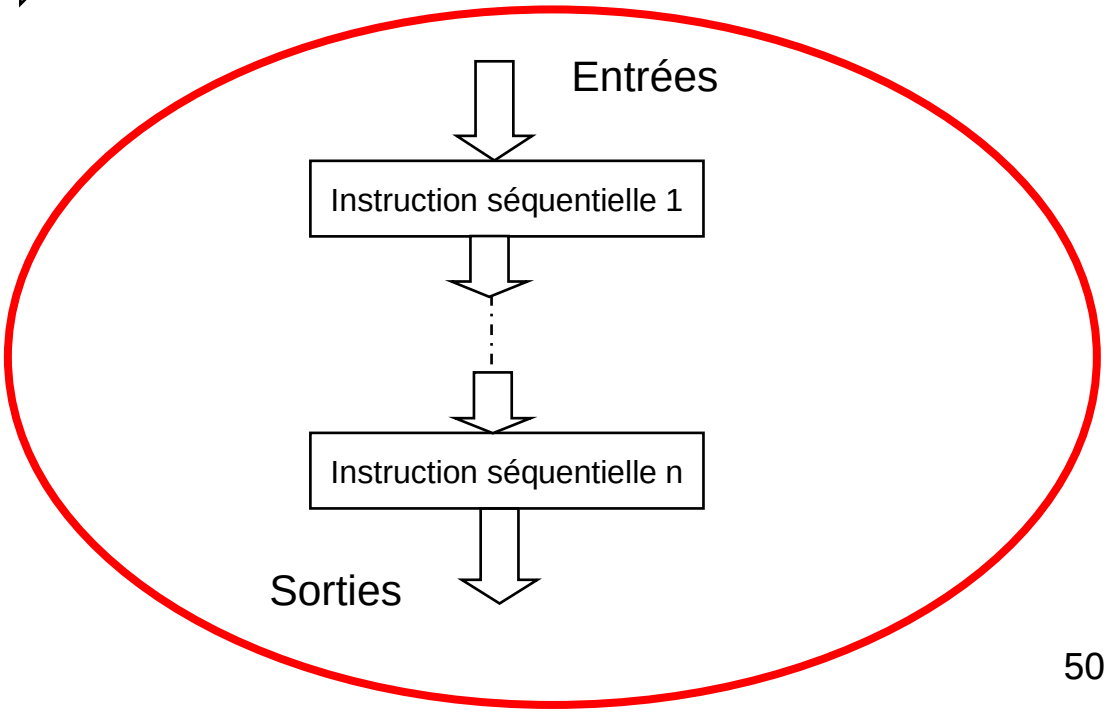
7 - Synthèse

Description concurrente ou processus



Instructions concurrentes

Instructions séquentielles (process ou sous-programmes)



Déclaration d'un processus

Les instructions placées dans une zone ***process*** sont exécutées selon l'ordre de présentation (séquentiellement).

Un ***process*** est déclenché à chaque changement d'au moins un des signaux définis dans la liste sensible, située juste aprés le mot clé ***process,*** et encadrée par des parenthèses.

[nom du process] : PROCESS (nom_des_signaux_de_la_liste_de_sensibilité)

BEGIN

suite d'instructions ...

END PROCESS [nom_du _process];

Etablir la liste sensible d'un MPX 4=>1, d'une bascule J,K simple, d'une bascule JK avec Preset et Clear (synchrone puis asynchrone).

Les instructions séquentielles : WAIT

Syntaxe : ***WAIT*** (***ON*** liste de signaux) (***UNTIL*** Condition_booléenne) (***FOR*** temps);

L'instruction ***WAIT*** peut être utilisée pour synchroniser le déclenchement d'un ***PROCESS*** (ou dans un sous-programme). L'exécution du processus peut-être suspendue en fonction d'une condition et pour un temps donné. L'exécution reprend si la condition spécifiée passe de la valeur ***FALSE*** à la valeur ***TRUE*** après le temps spécifié.

Exemple :

```
PROCESS  -- pas de liste sensible
BEGIN
        suite d'instructions;
    WAIT UNTIL condition booléenne;
        suite d'instructions;
END PROCESS ;
END arch ;
```

Asynchrone ? (reset, preset …)

Exemple :

```
ENTITY bascule IS
PORT (   CLK : IN std_logic ;
         d : IN std_logic ;
         q : OUT std_logic);
END bascule;
ARCHITECTURE arch OF bascule IS
BEGIN
PROCESS
BEGIN
    WAIT UNTIL clk='1' ;
        q <=d ;
END PROCESS ;
END arch ;
```

Le signal (d) est recopié en sortie (q) sur front montant de clk.

Les instructions séquentielles : IF THEN ELSE

IF (valeur booléenne) ***THEN*** **(**suite d'instructions)
ELSIF (valeur booléenne) ***THEN*** (suite d'instructions)
……
ELSE (suite d'instructions)
END IF ;

La condition implicite pour le ***ELSE*** final est en fait la condition complémentaire de l'ensemble des conditions déjà recensées.

La suite d'instruction est prise en compte que si la valeur booléenne est égale à ***TRUE***.

Exemple :

```
ENTITY dff IS
PORT ( CLK : IN std_logic ;
       d : IN std_logic ;
       q : OUT std_logic);
END dff ;
ARCHITECTURE arch OF dff IS
BEGIN
PROCESS (clk,d)
BEGIN
        IF (clk'EVENT AND clk='1')
                THEN q <=d ;
        END IF ;
END PROCESS ;
END arch ;
```

Falling_edge(CLK)
Ou
rising_edge(CLK)

Exo : Donner la description d'un registre (n bits) à chargement parallèle commandé avec Clear et Preset synchrones puis asynchrones avec front montant puis descendant.

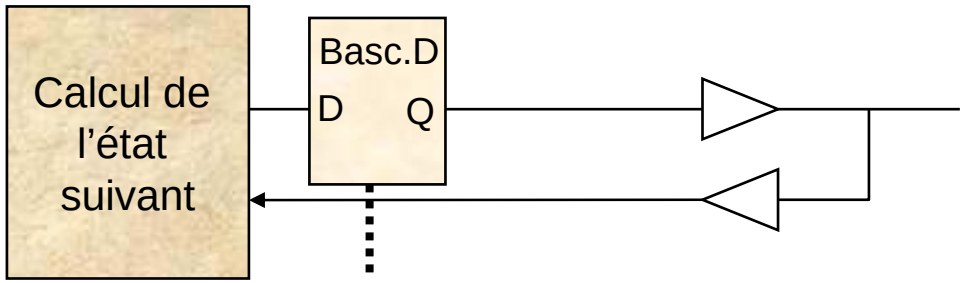
Description d'un Compteur

```
ENTITY compteur IS
PORT (     CLK : IN std_logic ;
           CQ : OUT ??  std_logic_vector(7 downto 0);

END compteur ;

ARCHITECTURE arch OF compteur IS
BEGIN
PROCESS (clk)
BEGIN
           IF (clk'EVENT  AND  clk='1')
           THEN     CQ <= CQ + 1;        => Erreur
           END IF ;
END PROCESS ;
END arch ;
```

INOUT ?

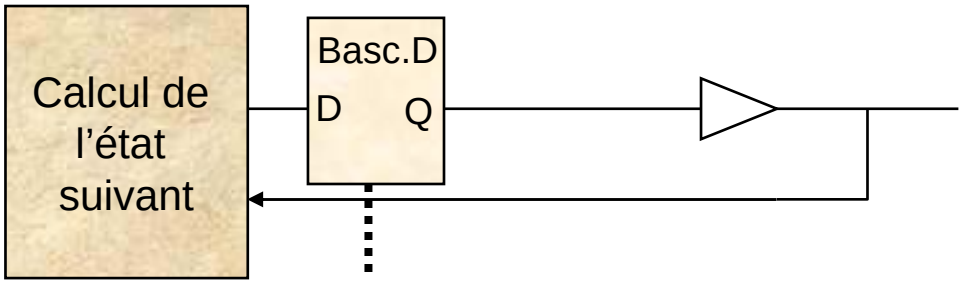
BUFFER ?

Q => INOUT ou BUFFER ?



INOUT

BUFFER




```
ENTITY compteur IS
PORT (      CLK : IN std_logic ;
            CQ : BUFFER  std_logic_vector(7 downto 0);
END compteur;
ARCHITECTURE arch OF compteur IS
BEGIN
PROCESS (clk)
BEGIN
      IF (clk'EVENT  AND  clk='1') THEN    CQ <= CQ + 1; END IF ;
END PROCESS ;  END arch;
```

CQ => Autre possibilité ?

Déclaration d'un signal intermédiaire qui sera copié dans le signal de sortie à la fin du ***PROCESS***.

```
ENTITY compteur IS
PORT (      CLK : IN std_logic ;
            CQ : OUT  std_logic_vector(7 downto 0);
END compteur ;
ARCHITECTURE arch OF compteur IS
SIGNAL CMP : std_logic_vector(7 downto 0);
BEGIN
PROCESS (clk)
 BEGIN
      IF (clk'EVENT  AND  clk='1')
            THEN    CMP <= CMP + 1;
      END IF ;
 END PROCESS ;
  CQ<= CMP;
END arch;
```

Quel est le cycle parcouru par ces compteurs ?

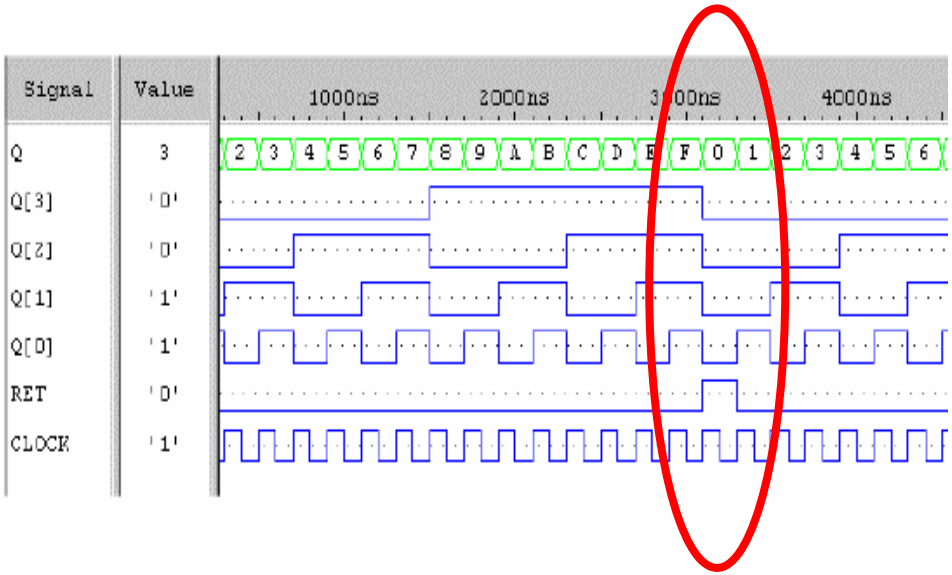
Description d'un Compteur

Exo : Reprendre la description de ce compteur et rajouter les commandes load, Clear et Preset.
Rmq : Q<="00000000" ; Q<= (others =>'0');
Rendre générique ce compteur.

Attention : dans un process, les sorties ne sont ré-actualisées qu'à la fin du process.

Exemple :

```
...PROCESS (clk)
BEGIN
        IF (clk'EVENT AND clk='1')
        THEN CQ <= CQ + 1;
             IF CQ ="1111"
             THEN retenue<='1';
             ELSE retenue<='0';
             END IF;
        END IF;
END PROCESS;
```



Quand le signal « retenue » passera-t-il à 1 ?

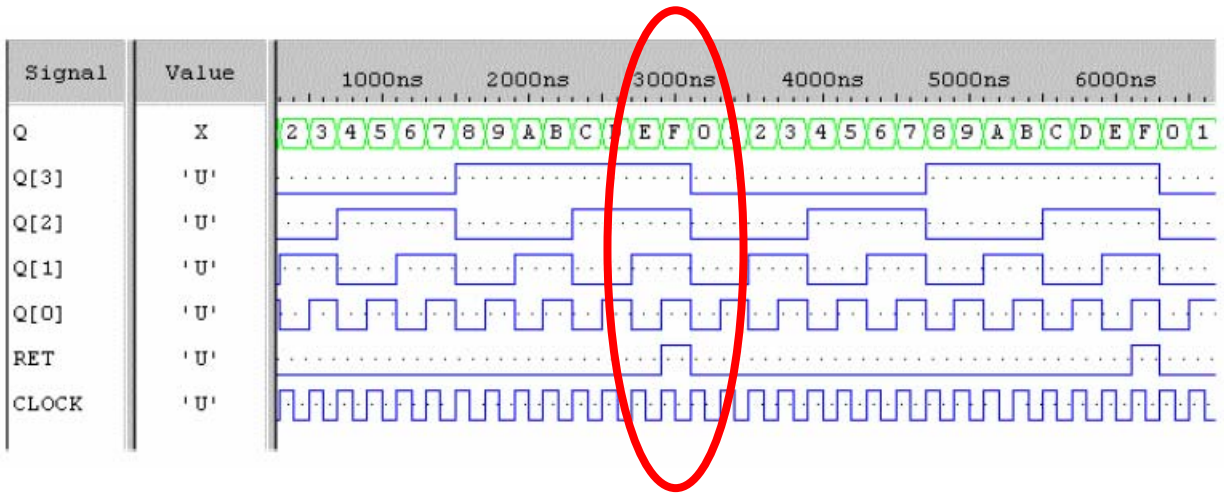
Remède :

1 - ***...PROCESS*** (clk)
BEGIN
IF (clk'***EVENT AND*** clk='1')
THEN CQ <= CQ + 1;
IF CQ ="1110"
THEN retenue<='1';
ELSE retenue<='0';
END IF ;
END IF ;
END PROCESS ;
=> Difficilement lisible, synchrone

2 - ***...PROCESS*** (clk)
BEGIN
IF (clk'***EVENT AND*** clk='1')
THEN CQ <= CQ + 1;
END IF ;

END PROCESS ;
retenue<='1' ***WHEN*** (CQ =''1111'') ***ELSE*** '0';

=> Plus facilement lisible, asynchrone



Les instructions séquentielles : CASE WHEN

Cette syntaxe permet de générer des affectations conditionnelles.

```
PROCESS (...)
BEGIN
CASE expression IS
WHEN choix 1 => suite d'instructions;
      WHEN choix 2 => suite d'instructions;
            ....;
      WHEN OTHERS => suite d'instructions;

      END CASE ;
```

Attention :

1- ne pas confondre => (implique) et <= (affecte).

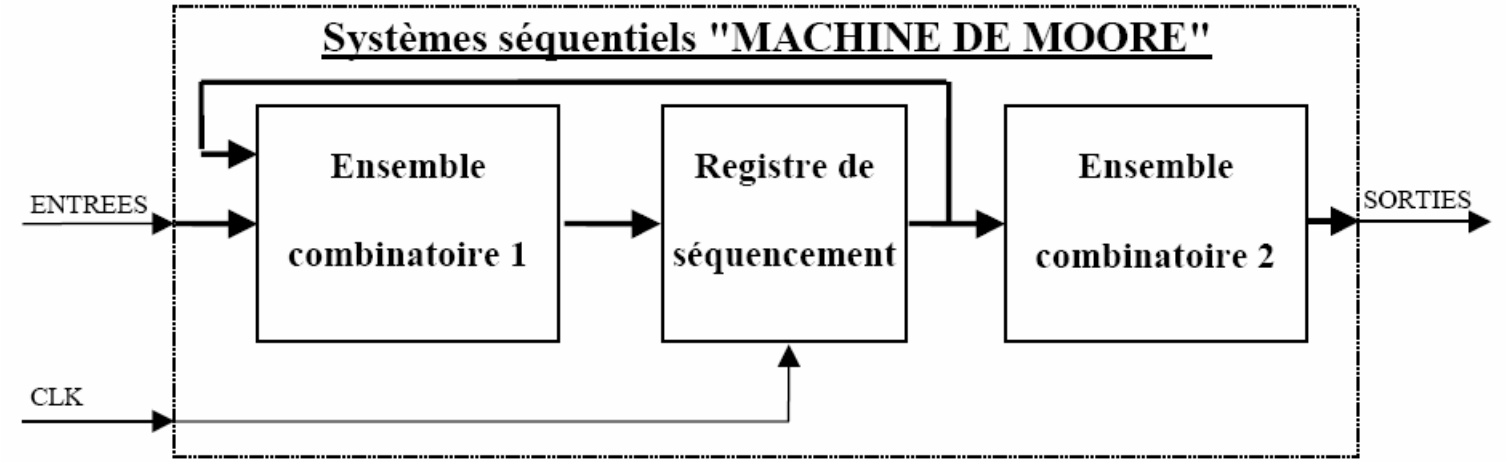
2 - Instruction *NULL* : l'exécution passe à la ligne suivante.

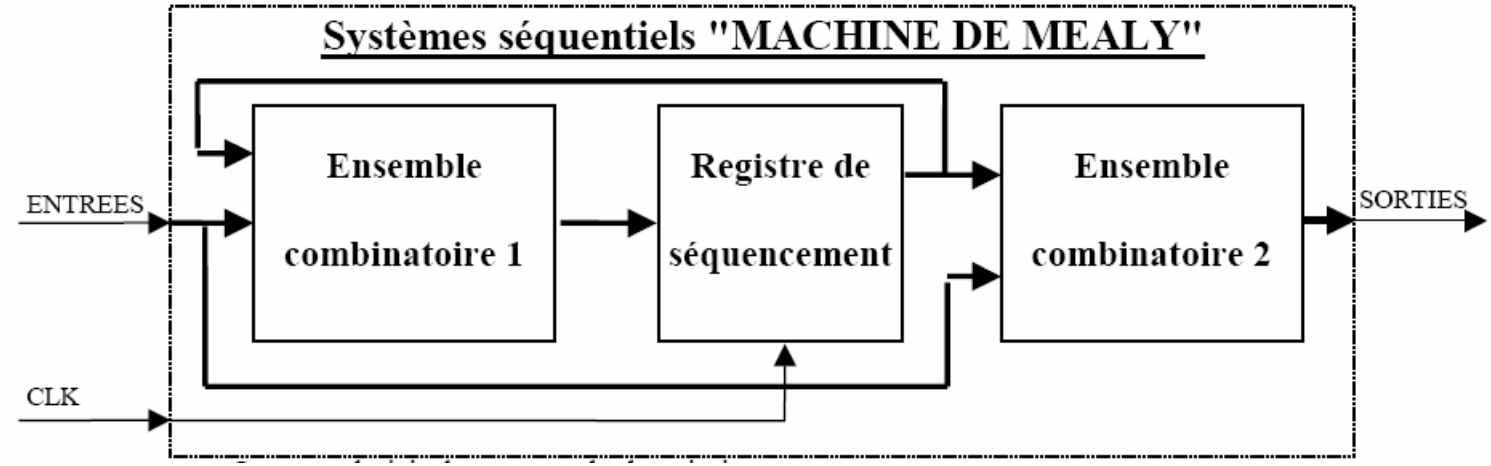
Exemple

```
ENTITY mux4 IS
PORT (    sel : IN std_logic_vector (1 DOWNTO 0);
          a, b, c, d : IN std_logic ;
          y : OUT std_logic);
END mux4 ;
ARCHITECTURE arch OF mux4 IS
BEGIN
PROCESS (a,b,c,d,sel)
BEGIN
      CASE sel IS
            WHEN ''00'' => y<= a;
            WHEN ''01'' => y<= b;
            WHEN ''10'' => y<= c;
            WHEN OTHERS => y<= d;
      END CASE ;
END PROCESS ;
END arch ;
```

Réécrire cette description en utilisant l'instruction NULL.

Rappels : Machines de Moore ou de Mealy

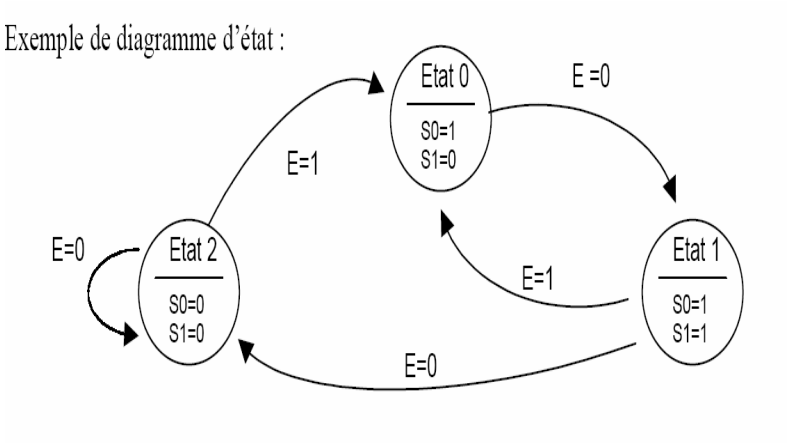




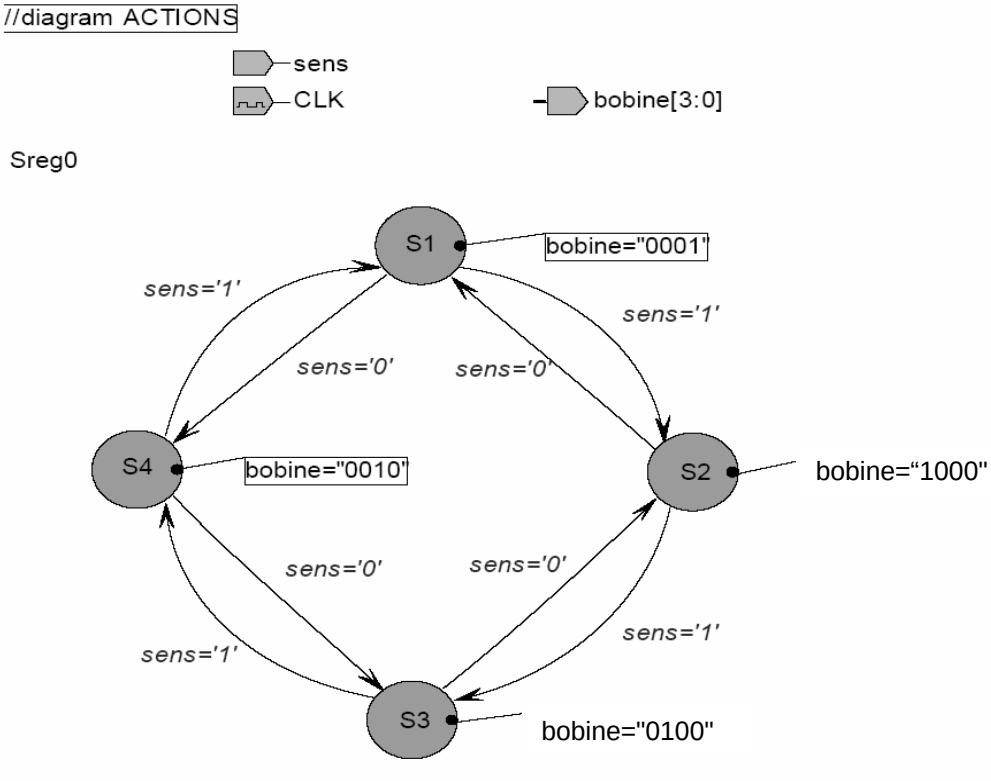
Les instructions séquentielles : machines d'états

Exo 1 : Décrire un registre à décalage sur n bits.

Exo 2 : Machine d'états



Proposer deux descriptions l'une contenant un processus, l'autre deux processus pour les deux graphes d'état ci-dessus.



Moteur pas à pas

Les instructions séquentielles : CASE WHEN

Exo 4 : Décrire une machine de Mealy

Exo 5 : Moore ou Mealy

```
    TYPE STATE_TYPE IS (__state_name, __state_name, __state_name);
    SIGNAL state: STATE_TYPE;
BEGIN
    PROCESS (clk)
    BEGIN
        IF reset = '1' THEN
            state <= _state_name;
        ELSIF clk'EVENT AND clk = '1' THEN
            CASE state IS
                WHEN __state_name =>
                    IF __condition THEN
                WHEN __state_name =>
                    IF __condition THEN
            END CASE;
        END IF;
    END PROCESS;
    WITH state SELECT
        __output_name    <=    __output_value    WHEN    __state_name,
                               __output_value    WHEN    __state_name,
                               __output_value    WHEN    __state_name;
```

Moore ?
ou
Mealy ?

Structure de boucles

Ces instructions permettent comme dans les langages traditionnels de répéter un grand nombre de fois des instructions.

Syntaxe :

nom : schéma_d_itération
LOOP
suite d'instructions;
END LOOP ;

OU

WHILE condition
LOOP
suite dinstructions
END LOOP;

Schéma d'itération : non connu avant l'exécution

```
VARIABLE N : INTEGER;
…
LOOP   -- boucle infini ???
          N:=N+1;
END LOOP;
```

```
VARIABLE R, M, N : INTEGER;
          …
WHILE   R/=0 LOOP
          R:=M mod N;
          M:=N;
          N:=R;
END LOOP;
```

Structures de boucles : FOR ... LOOP

Schéma d'itération connu à l'exécution :

```
FOR parameter_specification
IN discrete_range
LOOP
        suite d'instructions;
END LOOP ;
```

=> Exo : Calcul du bit de parité

Exemple :

```
ENTITY shiftreg IS
PORT(    clk : IN std_logic ;
         d : IN std_logic ;
         q : BUFFER std_logic_vector (7 DOWNTO
0) );
END shiftreg;
ARCHITECTURE arch OF shiftreg IS
BEGIN
PROCESS
BEGIN
      WAIT UNTIL clk='1' ;
      FOR i IN 7 DOWNTO 1 LOOP
            q(i)<= q(i-1) ;
      END LOOP;
            q(0) <= d;
END PROCESS;
END arch ;
```

Structures de boucles

Instructions associées aux boucles :
***NEXT, EXIT*.**

NEXT : arrêt de l'itération en cours.
Si condition => arrêt si la condition est vrai;

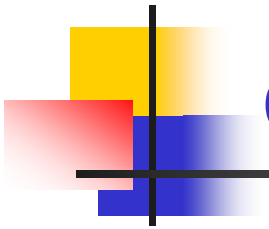
***NEXT* nom_boucle;**
Ou
***NEXT* nom_boucle *WHEN* condition;**

***EXIT* : permet de sortir de la boucle.**

***EXIT* nom_boucle;**
Ou
***EXIT* nom_boucle *WHEN* condition;**
Exemple :

```
LOOP
        R:= M mod N;
        M:=N; N:=R
        EXIT WHEN R=0 ;
END LOOP;
```

Synthèse ?



Concurrent/séquentiel (processus)

Attention :

En mode concurrent :

```
a <= b;
c <= b;
a <= 1;  Erreur !!!!!
```

En mode séquentiel :

```
PROCESS (a, b, c)
BEGIN
        a <= b;
        c <= b;  -- Cette description équivaut à a = 1 et c=b
        a <= 1;  -- (pas d'erreur générée).
END PROCESS;
```

Instruction assert

ASSERT condition ***REPORT*** message ***SEVERITY*** niveau_de_sévérité_de_l_erreur;

TYPE SEVERITY_LEVEL ***IS*** (NOTE, WARNING, ERROR, FAILURE);

Un niveau de sévérité = Failure arrête la simulation.

Exemple :

ASSERT NOW<1 ***REPORT*** ''fin de simulation'' ***SEVERITY*** failure;

Cette ligne arrêtera la simulation dès que la date sera égale à 1 minute.

PROCESS BEGIN

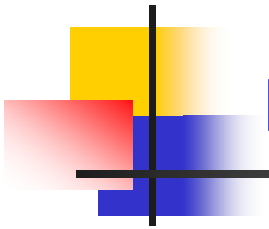
ASSERT (S1/='0' AND S2/='0') ***REPORT*** ''Message pour l'opérateur '' SEVERITY warning;

WAIT ON S1, S2; ***END PROCESS;***

Exercice :

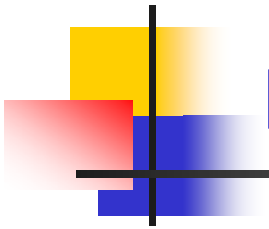
Ecrire un programme permettant de générer une erreur pour la division par 0.

Ecrire un programme permettant d'émettre un warning si le nombre N n'est pas du DCBN, ou un code ASCII.



Donner la modélisation des composants suivants

- Mémoires RAM, ROM, synchrone, asynchrones
- Piles FIFO, LIFO
- Processeur
- Séquenceur programmable,
- Description d’un système asynchrone.



Plan

1 - Introduction

2 - Concepts de base, types, opérateurs

3 - Instructions concurrentes

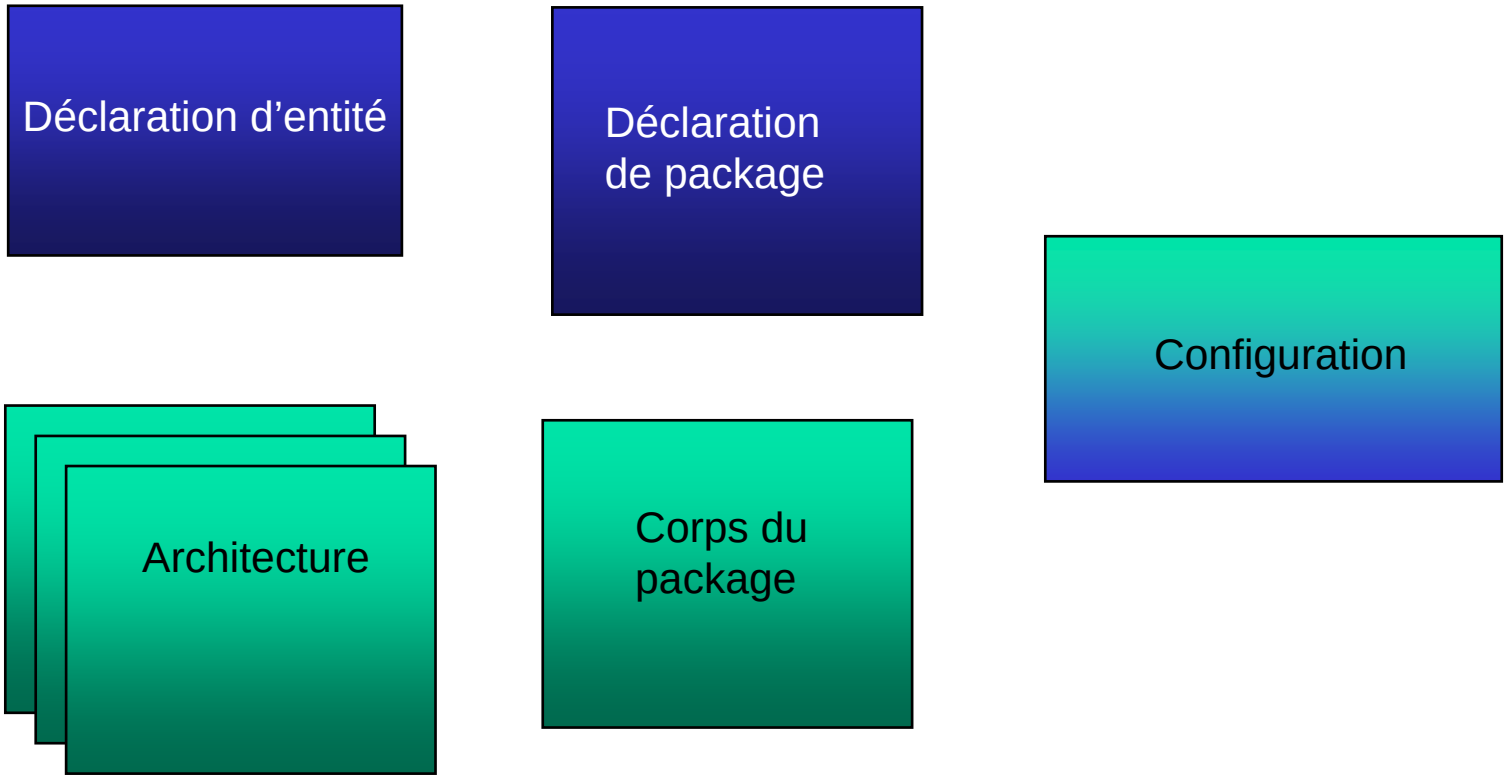
4 - Instructions séquentielles

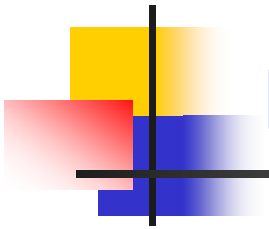
5 - Bibliothèques et sous programmes

6 - Simulation d'un programme VHDL

7 - Synthèse

Cinq catégories d'unités de conception en VHDL

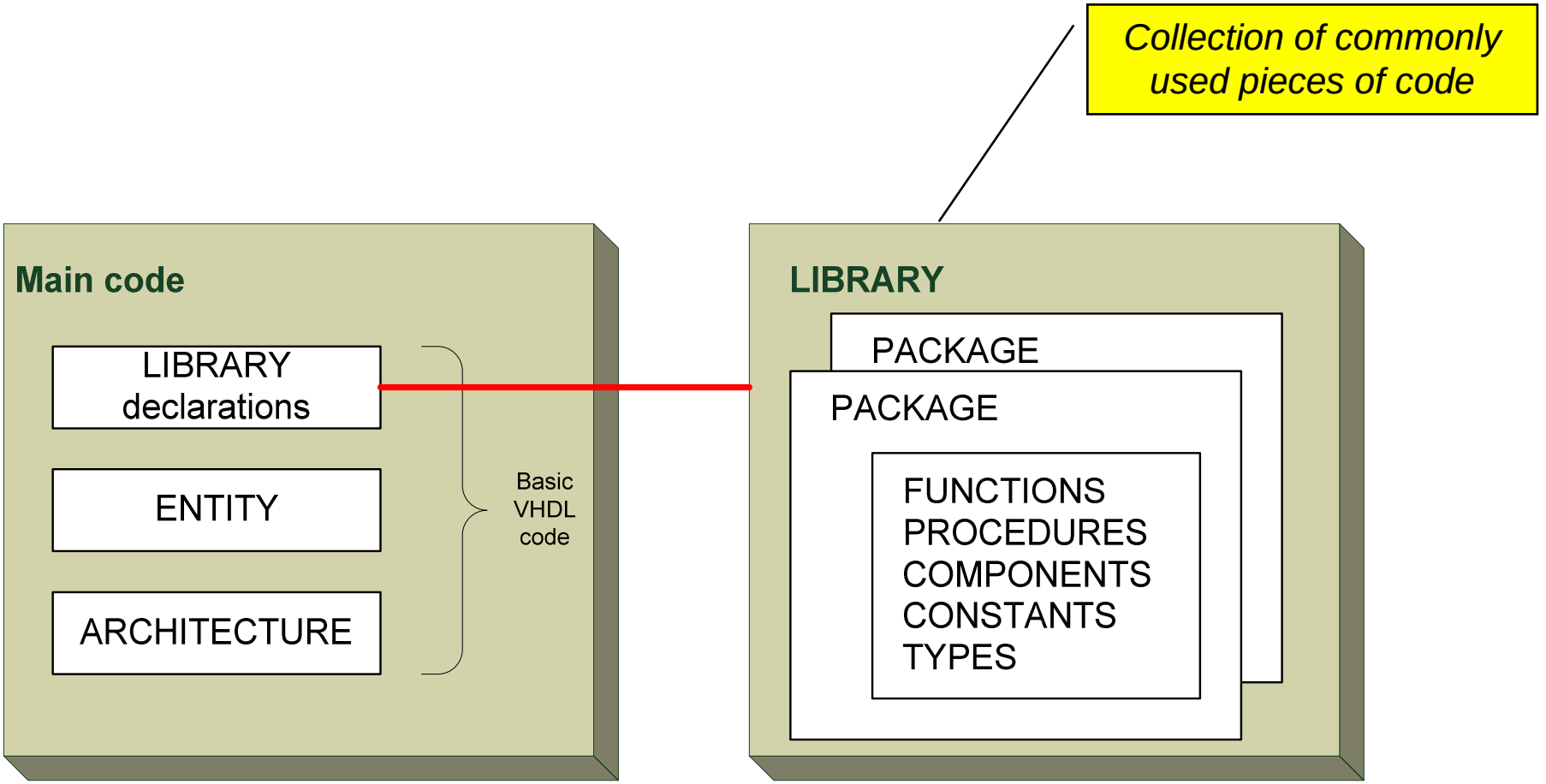




Notion de bibliothèque

- VHDL est un langage modulaire. On ne va pas y écrire de longues descriptions mais des unités plus petites et hiérarchisées.
- Complexité des systèmes => hiérarchie, sous-système => bibliothèque
- Les algorithmes utilisent des sous-programmes qui sont classés par paquet (package).
- Il est possible de posséder un grand nombre de bibliothèques mais à un instant donné une seule de ces bibliothèques est dite bibliothèque de travail.
- Le choix de la bibliothèque de travail se fait hors langage par une commande liée à la plateforme VHDL utilisée. Elle est généralement référencée en VHDL par le nom logique WORK.
- C'est dans cette bibliothèque que sont rangées les unités de conception compilées avec succès.
- Une bibliothèque comporte des packages, un ***Package*** étant une collection d'éléments regroupés dans un même fichier. Ces éléments sont soit des composants, soit des sous-programmes, soit des déclarations de type.

Code structure



Librairie et Package (suite)

En pratique, on trouve trois types de bibliothèques :

- ***WORK*** : bibliothèque de travail,
- ***STD*** : bibliothèque standard VHDL,
- ***IEEE*** : bibliothèque standard IEEE.

Les bibliothèques WORK et STD font l'objet d'une clause library implicite, c'est-à-dire que toute unité est comme précédée de la clause :

```
LIBRARY work, std;
USE std.standard.all;
```

Pour la librairie IEEE, il est nécessaire de la déclarer ainsi que les paquetages utilisés :

```
LIBRARY IEEE;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;  --opérations  non signées
ou USE ieee.std_logic_signed.all;       --opérations signées
ou USE ieee.std_logic_arith.all;        -- opérations ALU
```

Notion de paquetage

- Un package contient l'ensemble des fonctions, procédures, constantes, variables ... utilisées par plusieurs descriptions.
- 2 parties : spécification + description interne

Syntaxe :

PACKAGE nom_du_package ***IS***

<<Déclaration des composants>>

<<Déclaration des types>>

<<Déclaration des signaux>>

END nom_du_package;

Un package est toujours associé à une librairie dont la visibilité se fait par l'instruction ***LIBRARY*** et celle du package par ***USE***.

LIBRARY nom_de_la_librairie;

USE nom_de_la librairie.nom_du_package.***ALL***;

Notion de paquetage : exemple

Considérons une constante utilisée par deux architectures :

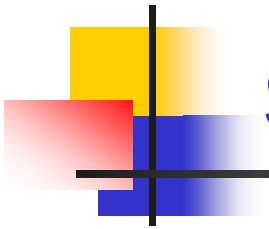
Tps : temps de propagation = 27 ns

Déclaration :

```
PACKAGE Pack1 IS
CONSTANT tps : TIME;
END Pack1;
...
PACKAGE BODY Pack1_Bod OF Pack1 IS
CONSTANT tps : TIME : = 27 ns;
END Pack1_Bod;
```

Utilisation :

```
USE WORK.Pack1.all;
ENTITY A IS ...
END A;
ARCHITECTURE arch_A OF A IS ...
        S <= E1 AND E2 AFTER tps;
        ...
END arch_A;
```



Sous programmes : fonction, procédure

Le rôle d'une procédure ou d'une fonction est de permettre au sein d'une description la création d'outils dédiés à certaines tâches pour un type déterminé de signaux.

Les Fonctions :

Une fonction reçoit des paramètres d'entrée et renvoie un paramètre de retour.

Les procédures :

Une procédure ne possède pas un ensemble de paramètres d'entrée et un paramètre de sortie mais un ensemble de paramètres d'entrée/sortie.

L'appel d'une procédure est une instruction alors que l'appel d'une fonction est une expression.

Les fonctions de résolution

Ces fonctions peuvent être appelées explicitement (dans une expression) ou implicitement (chaque fois qu'une source concernée doit être résolue).

Fonction

FUNCTION Nom_de_la_fonction (liste des paramètres : type)
RETURN type du paramètre retourné ***IS***
Zone de déclaration des variables***;***
BEGIN
    Instructions séquentielles;
    RETURN nom_de_la_variable_de_retour_ou_valeur_de_retour;
END ;

Une fonction reçoit des paramètres d'entrée et renvoie un paramètre de retour.

L'appel d'une fonction peut se faire soit dans les instructions concurrentes soit dans les instructions séquentielles. Dans le cas des instructions concurrentes, la fonction sera toujours vérifiée.

Exemple : fonction convertissant un boolean en un bit.

```
FUNCTION bool_to_bit (X: BOOLEAN) RETURN BIT IS
BEGIN
        IF (X=TRUE) THEN  RETURN '1';
        ELSE RETURN  '0';
        END IF;
END;
```

Fonction : exemple/exercices

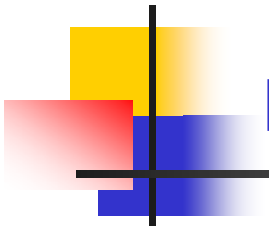
```
ENTITY  Montage1 IS
   PORT (  E1, E2, E3, E4 : IN std_logic ;
           S1,S2 : OUT std_logic);
END Montage1;
ARCHITECTURE arch OF Montage1 IS
   FUNCTION NON_ET (A,B,C : STD_LOGIC) RETURN  STD_LOGIC IS
      VARIABLE result : STD_LOGIC;
   BEGIN
           result:= NOT (A AND B AND C):
           RETURN result;
   END;
   BEGIN
           S1 <= NON_ET(E1,E2,E3);
           S2 <= NON_ET(E1,E2,E4);
END arch ;
```

Déclaration

Utilisation

Une fonction ne doit jamais arriver à son mot clé END final, cela provoquerait une erreur. Elle doit toujours rencontrer le mot clé return et rendre la main.

Exo1 : Définir une fonction capable de faire la conversion d'un entier en std_logic_vector.
Exo2 : Définir une fonction capable de faire l'addition d'un std_logic_vector.



Procédures

Une procédure, à la différence d'une fonction, accepte des paramètres d'entrée dont la direction peut-être IN, OUT, INOUT.

Exemple : la bascule SR

```
ENTITY  Test_SR IS
   PORT(   E1, E2 : IN std_logic ;
           S1,S2 : INOUT std_logic);
END Test_SR;
ARCHITECTURE arch OF Test_SR IS
   PROCEDURE SR (signal A,B : IN STD_LOGIC, signal Qb, Q : INOUT std_logic) IS
   BEGIN
           Q <= NOT (A AND Qb):
           Qb <= NOT( B AND Q);
   END;
   BEGIN
           SR(E1,E2,S1,S2);
END arch ;
```

Procédures

Si les paramètres de la procédure sont des variables l'affectation se fait par **:=** .
Reprenons l'exemple précédent avec des variables :

```
ENTITY  Test_SR IS
   PORT(   E1, E2 : IN std_logic ;
           S1,S2 : INOUT std_logic);
END Test_SR;
ARCHITECTURE arch OF Test_SR IS
   PROCEDURE SR (signal A,B : IN STD_LOGIC, variable Qb, Q : INOUT std_logic) IS
   BEGIN
           Q := NOT (A AND Qb):
           Qb := NOT( B AND Q);
 END;
BEGIN
           PROCESS (E1,E2)
           VARIABLE Q1,Q2 : std_logic;
           BEGIN
                   SR(E1,E2,Q1,Q2);
                   IF (Q1='1') THEN S1<='1';  ELSE Q1<='0'; END IF;
                   IF (Q2='1') THEN S2<='1';  ELSE Q2<='0'; END IF;
           END PROCESS;
END arch ;
```

Fonctions et procédures au sein des Packages

Afin de rendre accessible les fonctions et les procédures par plusieurs architectures, il est possible de créer un package.

```
PACKAGE Pack_NONET IS
        FUNCTION NONET (A,B,C : std_logic) return std_logic;
END Pack_NONET;

PACKAGE BODY Pack_NONET IS
   FUNCTION NONET (A,B,C : std_logic) return std_logic;
      VARIABLE result : STD_LOGIC;
   BEGIN
        result<= NOT (A AND B AND C):
        RETURN result;
   END;
END Pack_NONET;
```

Exercice : créer un package qui contient deux fonctions l'une permettant de calculer le MAX de deux entiers et l'autre le MIN en valeur absolue.

Pour rappeler la fonction,

```
LIBRARY USER;
USE USER.Pack_NONET.all;
ENTITY  Montage1 IS
   PORT (  E1, E2, E3, E4 : IN std_logic ;
           S1,S2 : OUT std_logic);
END Montage1;
ARCHITECTURE arch OF Montage1 IS
   BEGIN
        S1 <= NON_ET(E1,E2,E3);
        S2 <= NON_ET(E1,E2,E4);
END arch ;
```

Surcharge des sous programmes

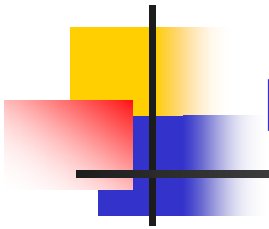
- Deux sous programmes sont dits surchargés si ils ont le même nom mais pas le même profil.
- Exemple :

FUNCTION MIN (A,B : ***INTEGER***) ***RETURN INTEGER*** ;
FUNCTION MIN (C,D : ***INTEGER***) ***RETURN INTEGER*** ; => erreur

FUNCTION MIN (A,B : ***INTEGER***) ***RETURN INTEGER*** ;
FUNCTION MIN (A,B : ***REAL***) ***RETURN REAL*** ;
FUNCTION MIN (A,B : ***BIT***) ***RETURN BIT*** ;

Lorsqu'on donne un opérateur comme nom de fonction, l'opérateur est lui-même surchargé.

FUNCTION "+" (A,B : ***BIT***) ***RETURN BIT*** ;
FUNCTION "+" (A,B : ***BIT_VECTOR***) ***RETURN BIT_VECTOR*** ;
FUNCTION "+" (A,B : ***STD_LOGIC_VECTOR***) ***RETURN STD_LOGIC_VECTOR*** ;



Les attributs = fonction

Un attribut est une caractéristique associée à un type ou un objet qui pourra être connue dynamiquement en cours de programme. Un attribut se note :

nom_type_ou_objet'nom_de_l_attribut

Attributs sur un type scalaire T

T'BASE : retourne le type de T.
T'HIGH : retourne la borne supérieure de T.
T'LOW : retourne la borne inférieure de T.
T'LEFT : retourne la borne de gauche de T.
T'HIGH : retourne la borne de droite de T.
T'POS(X) **:** retourne la position de l'élément X dans la liste.
T'VAL(X) **:** retourne la valeur du Xième élément.
T'SUCC(X) : retourne l'élément suivant de X.
T'PRED (X) : retourne l'élément précédent X.
T'RIGHTOF (X) **:** retourne l'élément à droite de X.
T'LEFTOF (X) : retourne l'élément à gauche de X.
T'IMAGE (expr.) : retourne une chaîne de caractères qui représente la valeur de l'expression.
T'VALUE (X) : prend une chaîne de caractère et retourne une valeur du type T qui lui correspond
T'ASCENDING : prend la valeur TRUE si T est ascendant)

Les attributs prédéfinis : exemple

Exemples et exercices :

1 – ***TYPE*** Feux_croisement ***IS*** {ROUGE, ORANGE, VERT};
SIGNAL feux : Feux_croisement;
Donner la valeur retournée par ces attributs

Feux'LEFT =
Feux'SUCC(ROUGE)=
FEUX'PRED(ORANGE) =
Feux'VAL(0)

Feux'RIGHT = ,
FEUX'SUCC(VERT) ?
Feux'PRED(ROUGE) ?
Feux'POS(VERT)

2 – **TYPE** niveau **IS** ('U','0','1','Z');
TYPE adresse **IS INTEGER RANGE** 7 **DOWNTO** 0;
TYPE code **IS** (NOP, ADD, SUB);

SIGNAL niv : niveau;
SIGNAL adr : adresse;
SIGNAL codeop : code;

Indiquer la valeur retournée par chacun des attributs pour les trois types ci-dessus.

Les attributs prédéfinis sur les tableaux

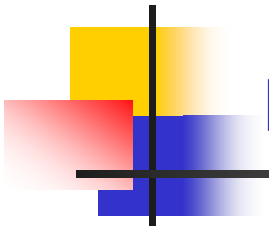
Attributs sur les tableaux

A'**ASCENDING**(N)	: rend une valeur booléenne qui est TRUE si le tableau A est ascendant.
A'**LENGTH**(N)	: nombre d'éléments dans le tableau
A'**HIGH**(N)	: élément haut du tableau
A'**LOW**(N)	: élément bas du tableau
A'**LEFT**(N)	: élément de gauche du tableau,
A'**RIGHT**(N)	: élément de droite du tableau,
A'**RANGE**(N)	: rang du tableau
A'**REVERSE_RANGE**(N)	: inverse du rang du tableau

N est la dimension (1..) considérée. Dans le cas d'un tableau à 1 dimension, (N) peut être omis. Par défaut N=1.

Exercices :

TYPE mémoire **IS ARRAY** (0 TO 255) OF STD_LOGIC_VECTOR(7 downto 0);
SIGNAL mem : mémoire;
TYPE ecran **IS ARRAY** (255 **DOWNTO** 0, 1 **TO** 1024) **OF BIT;**
SIGNAL visu : ecran;
Indiquer la valeur retournée par chacun des attributs pour les types ci-dessus.



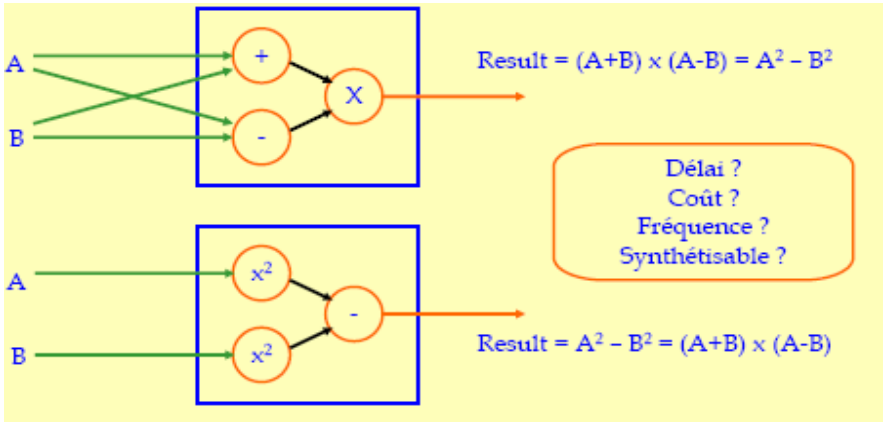
Les attributs prédéfinis sur les signaux

Attributs sur signaux

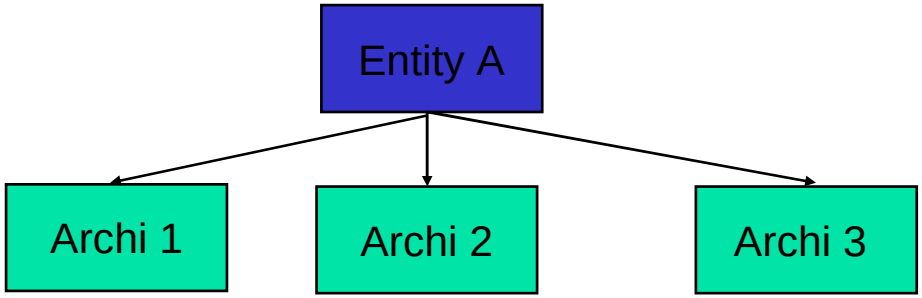
S'ACTIVE : vaut TRUE si S est actif.
S'DELAYED (T) : retourne un signal toujours égal à S retardé du temps T.
S'EVENT : vaut TRUE si S varie
S'LAST_VALUE : retourne la valeur précédente de S.
S'LAST_EVENT : type TIME, temps écoulé depuis le dernier événement sur S
S'LAST_ACTIVE : type TIME, temps écoulé depuis la dernière activité de S
S'STABLE(T) : signal égal à TRUE si S n'a pas été modifié pendant le temps T.
S'TRANSACTION : signal de type bit qui change d'état à chaque transition sur S.
S'QUIET(T): le signal égal à TRUE si S n'a pas été actif depuis T.
S'DRIVING: FALSE si la transaction dans le process en cours est NULL
S'DRIVING_VALUE: valeur de S pour la transaction dans le process en cours.

Exercice :
Ecrire un programme VHDL qui permette de vérifier si, pour une bascule, D les temps de set-up et de hold sont bien vérifiés.

Notion de configuration



- Il est possible de définir plusieurs architectures pour une même entité.



Synthèse, simulation → préciser quelle architecture → configuration.

Notion de configuration

CONFIGURATION nom_config ***OF*** nom_entité ***IS***

Partie déclaration (clause use et spécification d'attributs);

Partie configuration;

END nom_config;

Partie déclaration :

FOR nom_de_l_instance_du_composant : nom_du_composant ***USE ENTITY***

nom_de_l_entité(nom_de_l_architecture) ;

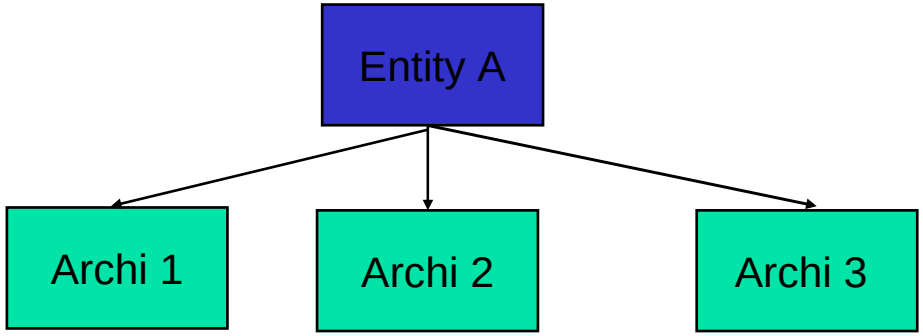
END FOR;

Partie configuration :

FOR nom_de_l_instance_du_composant : nom_du_composant ***USE CONFIGURATION*** (nom_de_la_configuration) ;

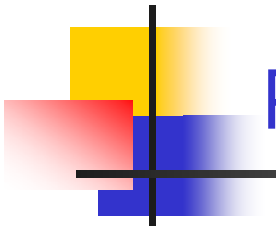
END FOR;

Notion de configuration : exemple




```
CONFIGURATION Conf_B OF B IS
        FOR arch_B
                FOR U1 : A USE ENTITY WORK.A(Archi2)
                END FOR;
        END FOR;
END Conf_B;
```

- Pour le composant U1 de l'architecture arch_B de l'entité B, utiliser l'architecture Arch2 pour l'entité A.
- Description portable : une seule configuration.



Plan

1 - Introduction

2 – Concepts de base, types, opérateurs

3 - Instructions concurrentes

4 - Instructions séquentielles

5 - Bibliothèques et sous programmes

6 - Simulation d'un programme VHDL

7 - Synthèse

Fichier de simulation

EN VHDL, il est possible de décrire des stimuli destinés à la simulation.

Exemple :

```
LIBRARY  ….
  ENTITY addsub IS
  PORT (    data1, data2 : IN std_logic_vector(3 downto 0);
            sel : IN std_logic ;
            data_out : OUT std_logic_vector(3 downto 0));
END addsub;
ARCHITECTURE arch OF addsub IS
  BEGIN
    WITH sel SELECT
    data_out <= data1 + data2 WHEN ’0’,
                data1 - data2 WHEN ’1’,
                NULL    WHEN others;
END arch;
```

1. *Dessiner les chronogrammes correspondant.*
2. *Décrire le fichier de simulation pour un compteur (inclure un process pour l’horloge)*

```
LIBRARY  ….
  ENTITY test_addsub IS
END test_addsub;

ARCHITECTURE sim_arch OF test_addsub IS
  COMPONENT addsub
    PORT ( data1 : IN std_logic_vector(3 downto 0);
           data2 : IN std_logic_vector(3 downto 0);
           sel : IN std_logic ;
           data_out : OUT std_logic_vector(3 downto 0));
  END COMPONENT;
  SIGNAL E1,E2, S : std_logic_vector(3 downto 0);
  SIGNAL Test_sel : std_logic;
  BEGIN
    U1 : addsub PORT MAP (E1,E2,Test_sel,S);
    Test_SEL <= ’1’ ,’0’ AFTER 200 ns;
    E1 <= "0000","1101" AFTER 100 ns,
        "0000" AFTER 200 ns,"1101" AFTER 300 ns;
    E2 <= "1100" ,"0011" AFTER 100 ns,
         "1100" AFTER 200 ns ,"0011" AFTER 300 ns;
END sim_arch;
```

Entrées clavier

- Il est possible de faire l'acquisition de données par un dialogue console. Les instructions à utiliser sont :

  -- Copie d'une chaine de caractère dans une variable L_ECR

  ```
  WRITE(L_ECR,STRING'("chaine de caractères"));
  ```

  -- Affichage à l'écran du contenu de L_ECR

  ```
  WRITELINE(OUTPUT,L_ECR); -- affichage à l'écran du contenu de L_ECR
  ```

  -- lecture d'une chaine de caractère tapée au clavier et mise en mémoire de cette chaine dans L_LEC

  ```
  READLINE(INPUT,L_LEC);
  ```

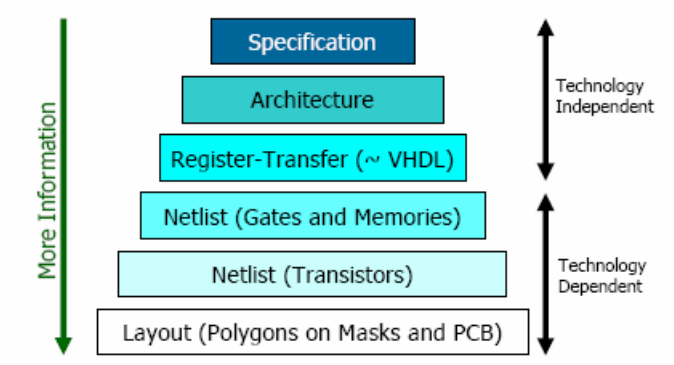
  -- transformation de la chaine de caractères mémorisée dans L_ECR en une variable de type TIME.

  ```
  READ(L_LEC,DATE,OK);
  ```

  L_ECR et L_LEC sont des variables de type LINE, DATE de type TIME et Ok est un booléen. OK indique si l'opération READ c'est bien passée.

- Attention pour utiliser ces instructions, il faut ajouter aux librairies le paquetage textio par l'appel suivant : USE std.textio.all;

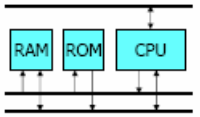
Les différents niveaux d'abstraction (ref)



Spécifications + cahier des charges (vitesse, énergie, prix, surface …)

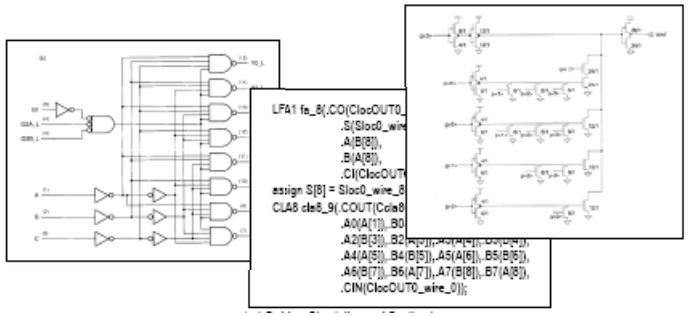
$$H(n) = \sum_{n=0}^{M} k_n x_n$$

Architecture : description de la structure du circuit

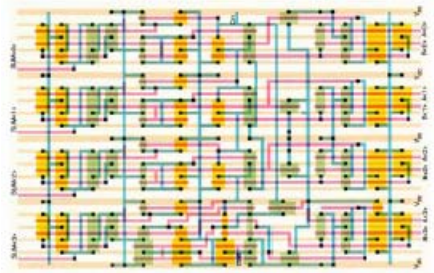


Register transfer level : Séparation de blocs logic et de blocs de mémorisation (registres).

Netlist : ensemble de portes (ou transistors) interconnectées

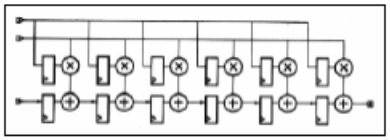


Physical Layout : mask

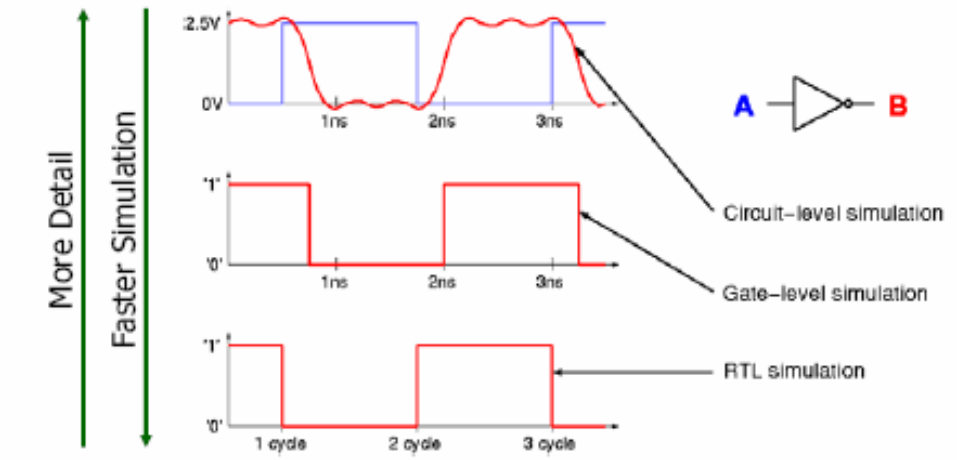


A: r8 ← r3 + 5

```
if signalA = 1 then register8 := register3 + 5;
```



Simulation et niveaux d'abstraction



Niveau circuit : temps et tension continus,

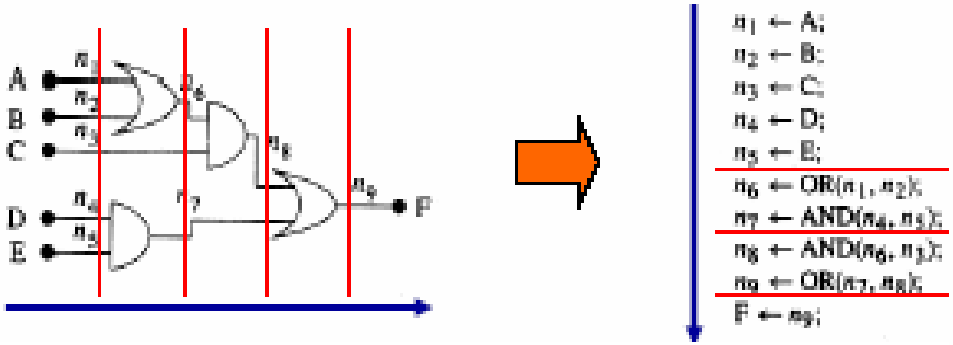
Niveau porte (gate-level) : temps continu, niveaux discrets

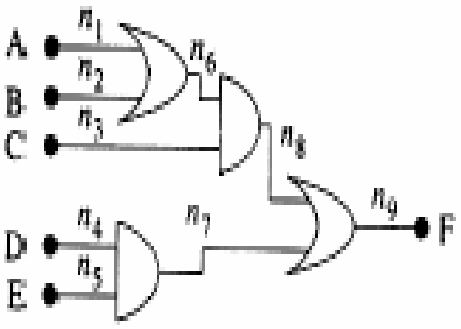
RTL simulation : temps plutôt en cycle machine, niveau logique

Différents types de simulation

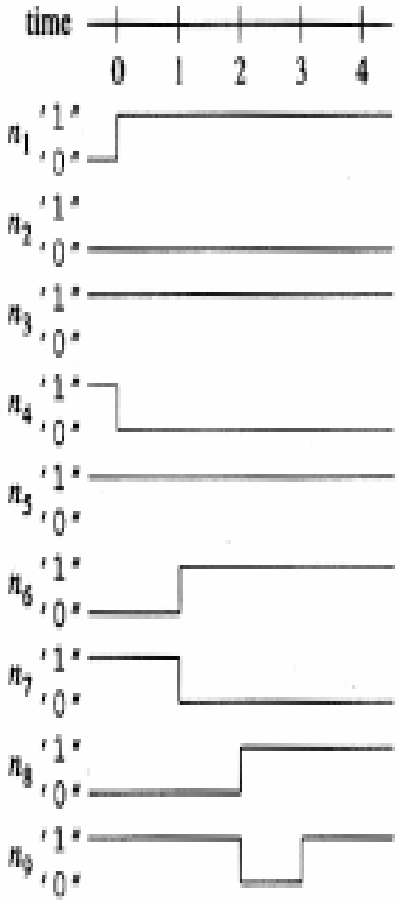
Simulation : compiler-driven

Principe : *transforme le circuit en un programme qui affecte un nouvel état aux nœuds du circuit quand de nouveaux stimuli sont appliqués.*






```
for (t ← t_start; t ≤ t_end; t ← t + 1) {
  new[1] ← A;
  new[2] ← B;
  new[3] ← C;
  new[4] ← D;
  new[5] ← E;
  new[6] ← OR(old[1], old[2]);
  new[7] ← AND(old[4], old[5]);
  new[8] ← AND(old[6], old[3]);
  new[9] ← OR(old[7], old[8]);
  F ← new[9];
  old ← new;
}
```



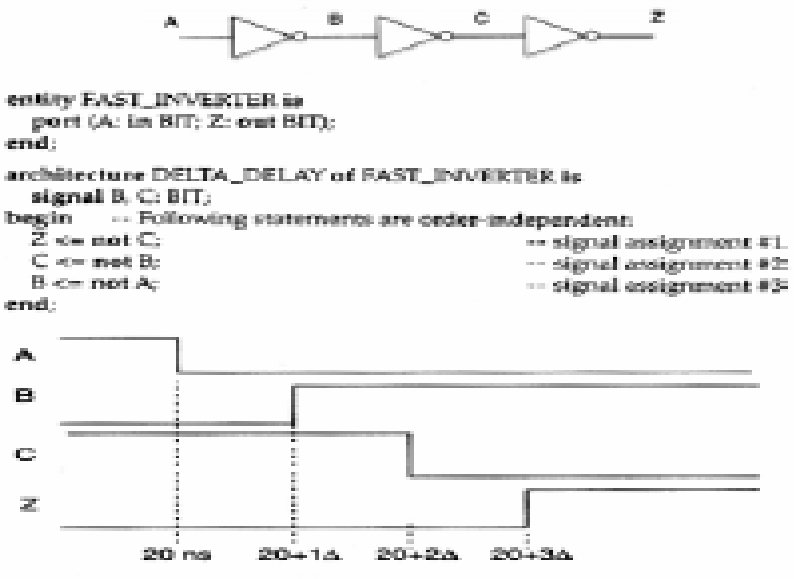
Modèle rapide ?, il calcule n'importe quoi même quand il n'y rien à calculer.

Pas de temps de réponse

Différents types de simulation

Simulation : event-driven

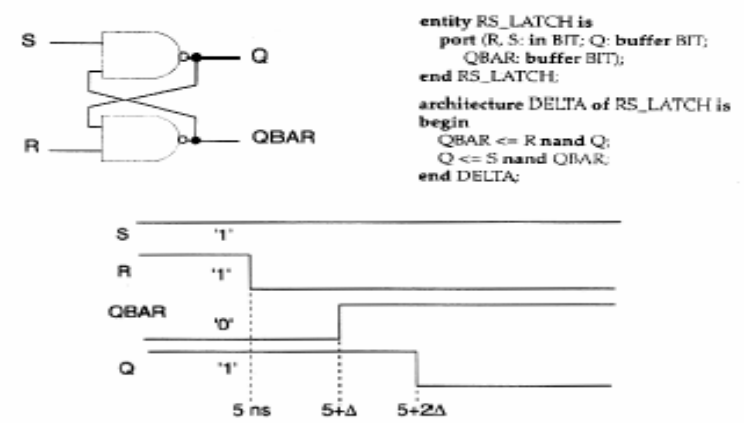
Principe : A chaque changement d'état d'un ou de plusieurs signaux d'entrée, le simulateur calcule l'état de signaux de sortie en tenant en compte des rebouclages et jusqu'à aboutir à une situation stable (δ-cycles).
C'est à ce moment seulement que les valeurs des signaux de sortie sont réactualisées.

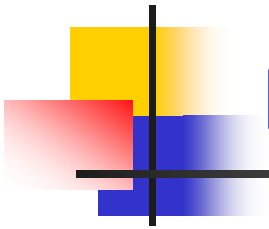


Temps de réponse des portes ?

Les événements sont mis dans une file avec leur date.

Delta représente un léger retard entre des événements successifs ou le temps de réponse du circuit.





Plan

1 - Introduction

2 - Syntaxe de base, types, opérateurs

3 - Instructions concurrentes

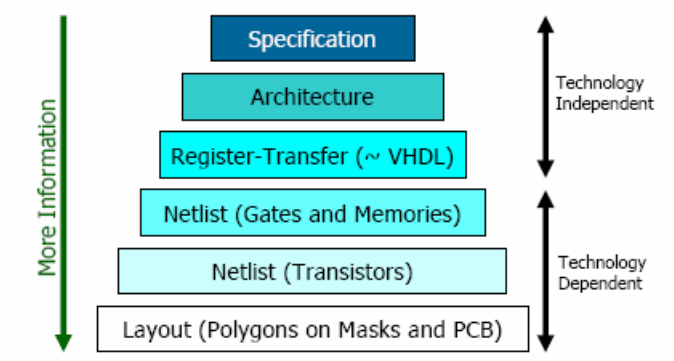
4 - Instructions séquentielles

5 - Bibliothèques et sous programmes

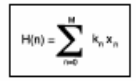
6 - Simulation d'un programme VHDL

7 - Synthèse

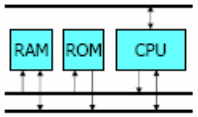
Les différents niveaux d'abstraction



Spécifications + cahier des charges (vitesse, énergie, prix, surface …)

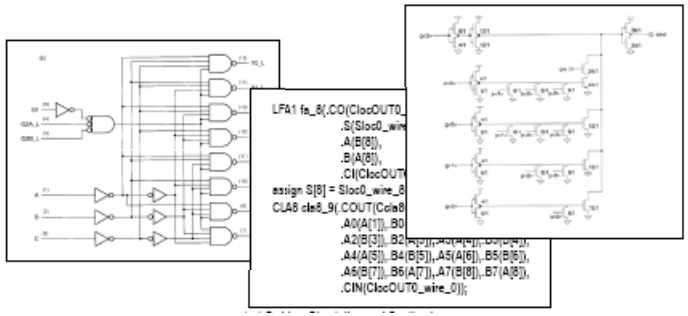


Architecture : description de la structure du circuit

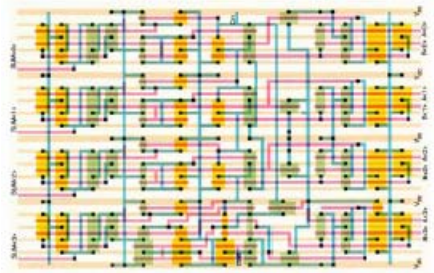


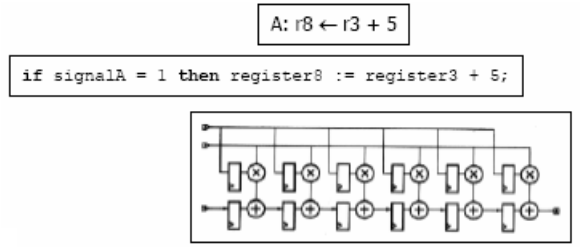
Register transfer level : Séparation d ebloc logic et de blocs de mémorisation (registres).

Netlist : ensemble de portes (ou transistors) interconnectées

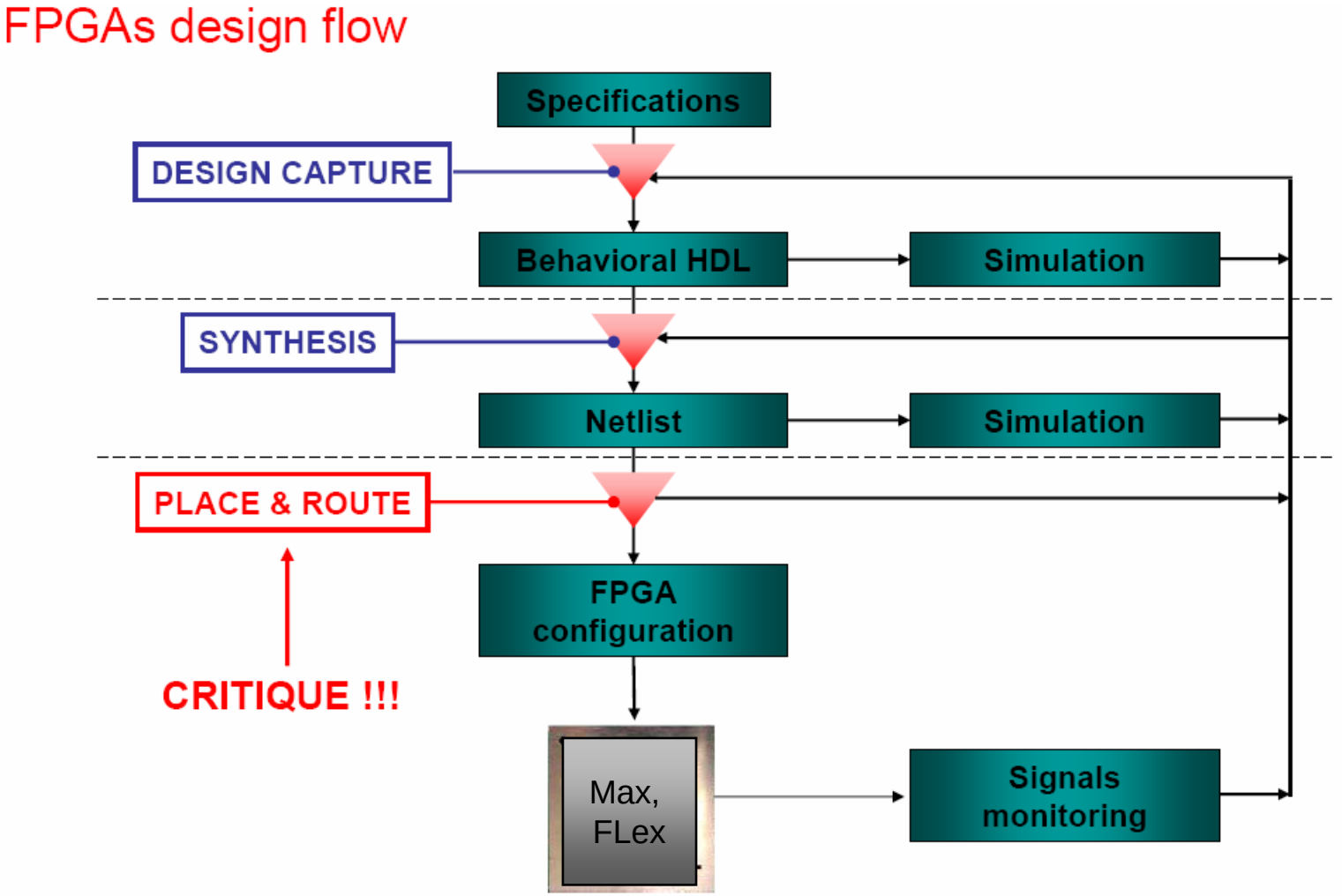


Physical Layout : mask





Synthèse : design flow utilisé en TP



Synthèse VHDL

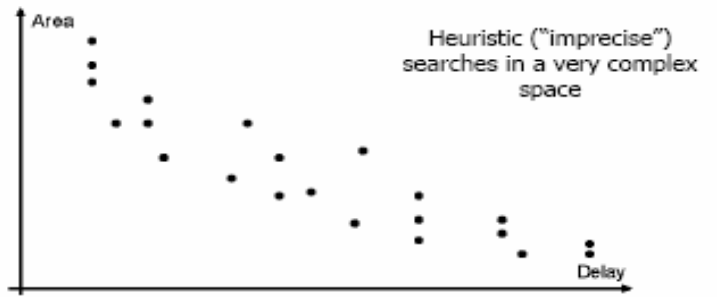
La synthèse est une phase importante car elle détermine les caractéristiques du composant final.

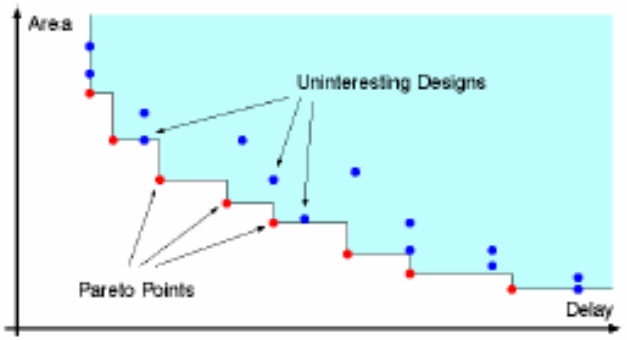
Spécification : contrainte temporelle, surface, coût, consommation ? → circuits différents
Solution différente selon la cible technologique (ASIC, FPGA …)

⇒ Optimisation indépendante de la technologie : simplification booléenne,

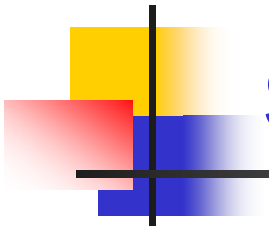
⇒ Optimisation dépendante de la technologie : FPGA → fonction booléenne + DFF dans une seule cellule

Espace de conception : diagramme de Pareto





Compromis temps/surface …



Synthétiseur logique

Entrées :

Description VHDL,

Optimisation choisie : temps, surface, puissance ?

Sorties :

Netlist (portes, DFF, latch …)

Report : Estimation du temps, surface, consommation

Conception avec le VHDL : quelques conseils

- *Il faut penser circuit.*
- *Une bonne conception commence par une décomposition du système (hiérarchie)*
- *Il faut imaginer l'architecture physique du système.*
- *VHDL n'est pas un outil de CONCEPTION mais de DESCRIPTION !*
- *Attention : Une description peut-être simulable mais pas synthétisable.*
- *Pour garantir une bonne qualité :*
    - *Une fonction par module,*
    - *Faire des descriptions simples et lisibles,*
    - *Respecter les syntaxes synthétisables (IEEE 1076.6-1999)*
    - *Expliciter les éléments mémoires,*
    - *Utiliser uniquement les bibliothèques IEEE*

        $\Rightarrow$ *lpm maxplus2, portabilité ?*

=> différentes cibles technologiques (ASIC, FPGA …)

La synthèse est une phase importante car elle détermine les caractéristiques du composant final.

Synthèse de système combinatoire

Fonction combinatoire : à chaque combinaison d'entrée correspond une combinaison de sortie (indépendantes des entrées et des sorties précédentes).

Description concurrente :

```
S <= A AND B OR C;
```

Description séquentielle :

```
PROCESS (A, B, C)
BEGIN
        IF (C='1') THEN S<='1' ;
        ELSIF (A='1' AND B='1') THEN S<='1' ;
        ELSE S<='0' ;
        END IF ;
END PROCESS ;
```

Les deux descriptions sont identiques d'un point de vue de la synthèse.

Synthèse de système combinatoire

Description de fonctions combinatoires à l'aide d'instructions concurrentes :

Avantages : lisible, simple, reflète la réalité

Inconvénients : simplification réalisée, système de faible complexité.

Description de fonctions combinatoires à l'aide de process :

Avantages : pas nécessaire de faire de simplification (seront faites par le synthétiseur), Description comportementale pour des fonctions compliquées.
Inconvénients : écriture peu lisible, synthèse ? Attention aux Pièges

Quelques règles :

1 - Tout processus décrivant de la logique combinatoire doit être sensible à toute entrée intervenant dans le processus.

2 - Toute sortie affectée à l'intérieur d'un processus doit avoir une valeur spécifiée pour tous les cas possibles.

- CASE : ne pas oublier WHEN OTHERS.
- IF : ne pas oublier le ELSE.

Synthèse de système combinatoire : les pièges

```
IF A='1' THEN
   Z <= B; END IF ;
-- LATCH

IF A='1' THEN
   Z <= B;
ELSE Z<= C; END IF ;
-- MUX

Z<=C;
IF A='1' THEN
  Z <= B; END IF ;
-- MUX

X <= NOT X;
Z <=X ;
-- boucle combinatoire
-- erreur
```

Identification of Sequential Elements

- Memory element if a signal or a variable
  - Is "alive on entry" of a process (i.e., someone might use its value), and
  - Is conditionally assigned therein
- Typical analysis techniques of compilers

```
if A = '1' then
   Z <= B;
end if;
```

Z is alive on entry
and assigned in an *if*
→ LATCH

```
if A = '1' then
   Z <= B;
else
   Z <= C;
end if;
```

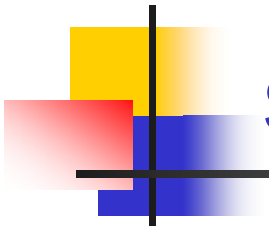
=

```
Z <= C;
if A = '1' then
   Z <= B;
end if;
```

Z is assigned in an *if*
but not alive on entry
(= always assigned something new)
→ MUX

```
X <= not X;
Z <= X;
```

X is alive on entry
and assigned unconditionally
→ combinatorial loop
probably an error...



Synthèse : système séquentiel

Logique séquentielle :

- un processus ne peut contenir qu'une horloge.
- un signal dont la valeur n'est pas indiquée dans une configuration se verra affecté un verrou ou une bascule D.
- tout système logique doit être initialisé par un RAZ asynchrone.

Synthèse VHDL

- Les boucles LOOP et GENERATE : sont déroulées dans l'outil de synthèse pour dupliquer le matériel

  → toute boucle doit être de dimension connue à la compilation.

  Boucle de dimension variable :

  → déterminer la dimension maximale qui fournira la taille effective du matériel
  → Ajouter une conditionnelle permettant de réduire au moment de l'exécution la dimension effectivement utilisée par la boucle.

  Exemple :

  i:=0 WHILE (i<N) LOOP ... → FOR i IN 0 TO 99 LOOP IF (i<N) THEN

- Les affectations de signaux ou de variables dans les PROCESSes déclenchés sur une horloge avec une condition sur l'un des fronts → registre sur front.

  Les bascules sur niveau sont considérées comme de la logique combinatoire. La définition d'une bascule sur niveau correspond au cas où une affectation de signal ou de variable n'est pas complètement définie.

```
PPROCESS (a,b,c)
BEGIN
        IF (e=OK) THEN s<=a; END IF; k<=a+b;
END PROCESS;
```

Synthèse VHDL

Pour k → combinatoire, S non précisé pour e différent de OK, S est mémorisé. Attention certains synthétiseurs ne reconnaissent pas les bascules à niveau (4 à 6 portes).

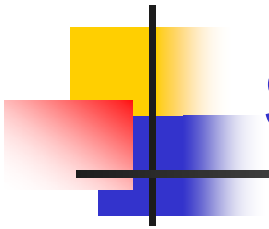
On peut aussi dans certains cas, préciser une valeur par défaut (évite la bascule).

```
PROCESS (a,b,c)
BEGIN
        s<=d; IF (e=OK) THEN s<=a; END IF; k<=a+b;
END PROCESS;
```

Les opérateurs relationnels <> sont synthétisés sous la forme de comparateurs.

Les opérateurs +, -,*, and, or sont synthétisés sous la forme d'opérateurs arithmétiques ou logiques correspondants.

Les opérateurs de concaténation & et de sélection de champ, dans le cas des tableaux et les opérateurs * et / sur des puissances de 2 pour des valeurs numériques entières sont synthétisés sous la forme de décalage.



Synthèse VHDL

Les instructions concurrentes WHEN (affectation sélectives) et WITH (affectation conditionnelle) et les instructions équivalentes IF et CASE sont synthétisées sous la forme de multiplexeurs. Dans le cas de PROCESS synchrone, ces multiplexeurs permettent d'aiguiller à chaque top d'horloge vers une partie traitement qui sera effectuée pour ce top.

Les bus trois états → pull up ou pull down.

PROCESS synchrone/ PROCESS asynchrone : on peut décrire un PROCESS qui ferait les deux.

PROCESS : exécution séquentielle donc les sorties ne sont activées qu'en fin de réaction du PROCESS.

VARIABLE : interne au PROCESS = connexion interne

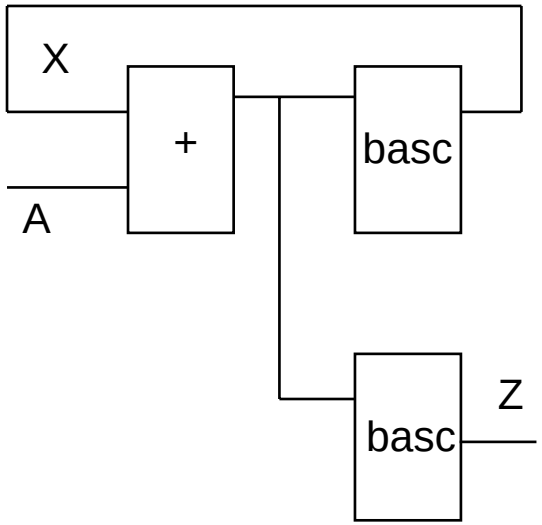
SIGNAL : visible dans et hors du PROCESS

Synthèse VHDL

Dans le cas d'un PROCESS combinatoire, une variable ne peut être réutilisée d'une itération à l'autre.

Les variables d'un PROCESS synchrone sur front correspondent à des bascules sur front si elles sont utilisées d'une itération à une autre sans avoir été à nouveau affectées.

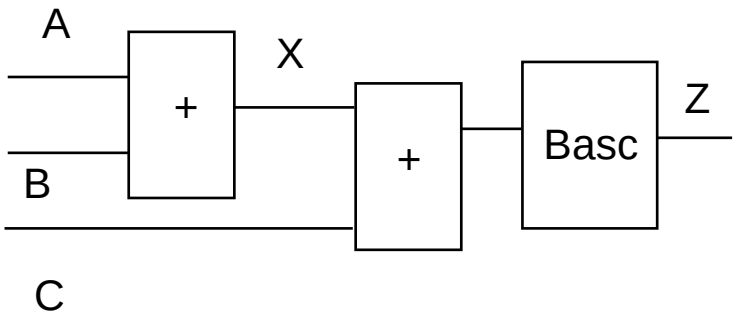
```
PROCESS (clk)
VARIABLE X : INTEGER ;
BEGIN
        IF (clk='1' AND clk'EVENT) THEN
                x := x+ A;
                z <= x;
        END IF ;
END PROCESS ;
```



Synthèse vhdl

Si une variable n'est pas réutilisée d'une itération à une autre, elle correspond alors à une simple connexion entre les étages de logique.

```
PROCESS (clk)
VARIABLE X : INTEGER;
BEGIN
        IF (clk='1' AND clk'EVENT) THEN
                X:=A+ B;
                Z<=X + C;
        END IF ;
END PROCESS ;
```



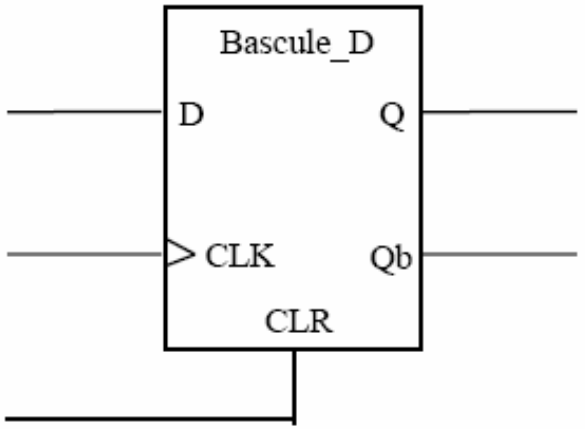
Synthèse de systèmes séquentiels

La synthèse d'une fonction logique séquentielle est beaucoup plus complexe que celle d'une fonction combinatoire. Les possibilités de description offertes par le langage VHDL sont vastes et bien souvent irréalisables dans l'état actuel des technologies. Prenons l'exemple des attributs, ils sont puissants, efficaces, mais posent, pour certains, de gros problèmes aux outils de synthèse qui ne savent pas comment réaliser des structures capables de représenter le fonctionnement de ces attributs. L'attribut 'quiet(T), qui retourne une information de type booléen qui est vraie lorsque le signal auquel il est associé n'a pas été affecté depuis le temps T, ne représente aucune structure électronique simple. Les synthétiseurs actuels refusent donc cet attribut faute de savoir quoi en faire.

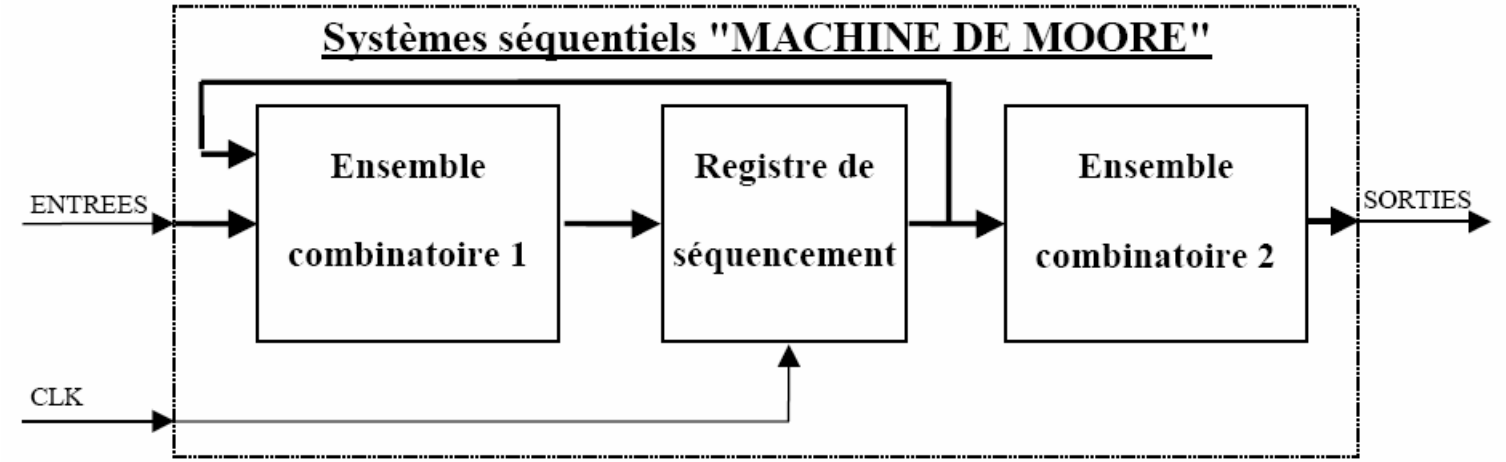
Par contre, il existe des attributs qui sont très utilisés et qui représentent même un véritable standard. La description d'un front est très simple à réaliser si l'on utilise l'attribut 'event associé à une condition de niveau pour la détection d'un front montant ou descendant. L'exemple ci-dessous qui correspond à la description d'une bascule D ne posera donc aucun problème de synthèse.

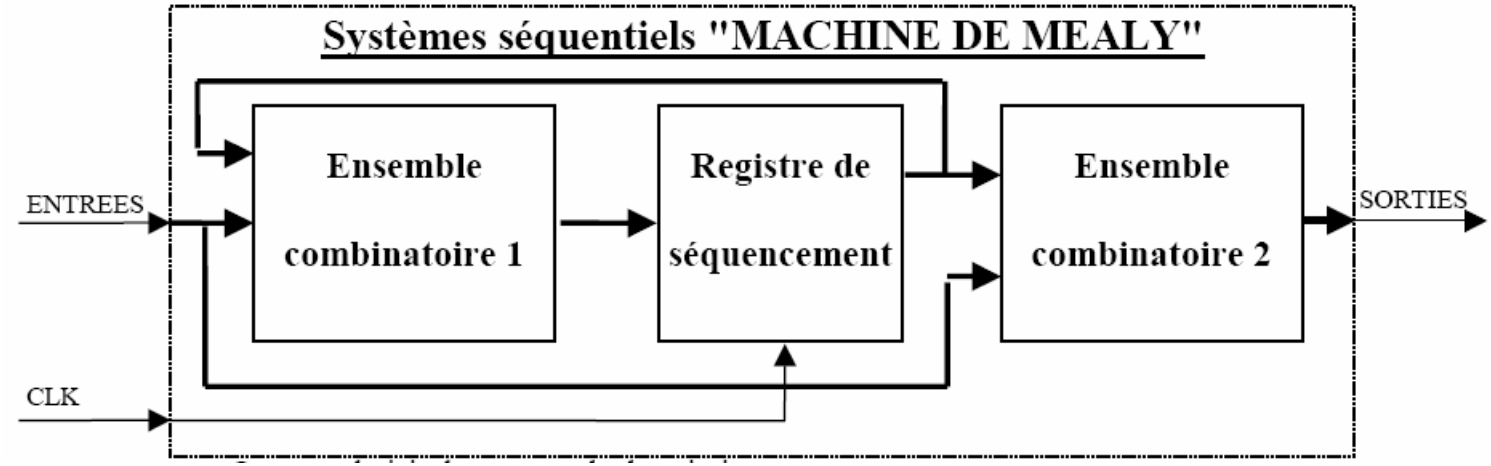
Synthèse de systèmes séquentiels

```
Library ieee;
use ieee.std_logic_1164.all;
entity bascule_D is
port (D, clk, clr : in std_logic; Q, Qb : out std_logic);
end bascule_D;

architecture bascule_D of bascule_D is
BEGIN
        process (clk, clr)
        begin
        if (clr = '1') then                    Entrée CLR active à '1'
           Q <= '0';
           Qb <= '1';
        elsif (clk'event and clk = '1') then
           Q <= D;
           Qb <=  not D;                       Détection d'un front montant
        end if;
        end process;
end bascule_D;
```



Machine de Moore/Mealy





Machine d'états

On peut choisir deux types de description :

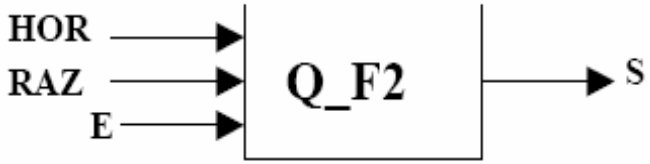
❶ comportementale, se limitant à décrire le comportement de la structure à l'aide d'instructions IF, THEN, ELSE, CASE, WHEN...,

❷ structurelle ou semi-structurelle, décrivant chacune des parties constituant les machines de MOORE ou de MEALY.

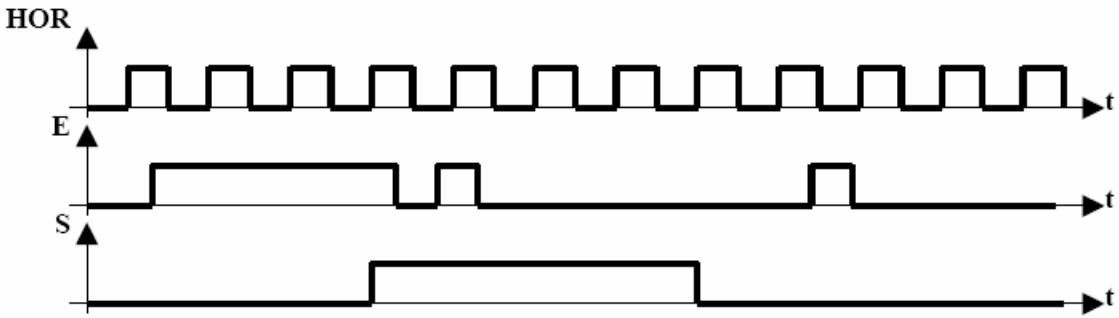
➔ Dans le premier cas, la description est indépendante du choix de la structure finale. En d'autres termes, la description n'impose pas une structure cible, le synthétiseur est donc libre de la choisir

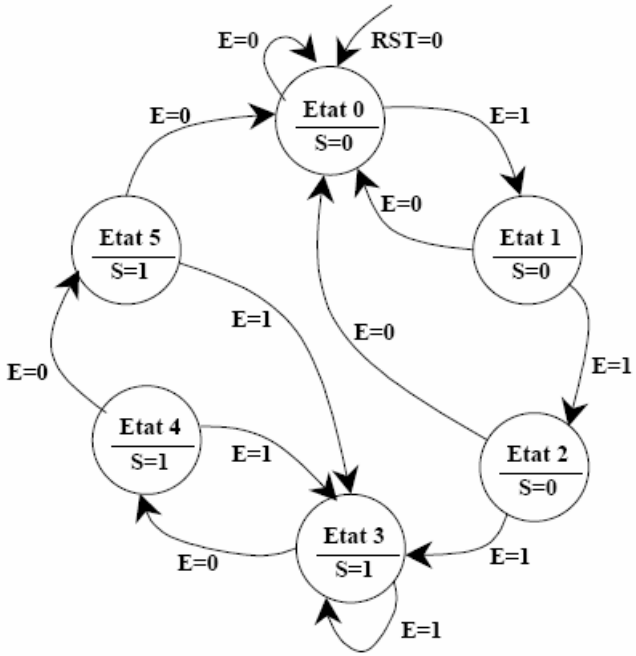
➔ Dans le deuxième cas, le choix de la structure cible est implicite ou partiel. Tout ce passe comme si on imposait une structure cible au synthétiseur. Dans ces conditions, le résultat de la synthèse est conforme à la description, et de ce fait optimal si le choix de la structure cible l'est. Notons qu'il est souvent intéressant d'utiliser ce genre de description lorsque l'on souhaite imposer des critères de synthèse du type : utiliser un minimum de bascules, privilégier la rapidité du montage, garantir certaines contraintes de temps, etc.

Exemple :



On désire réaliser une fonction dont la sortie S recopie l'état logique présent sur son entrée E si celle-ci est restée stable après 2 coups d'horloge successifs.





Exemple :

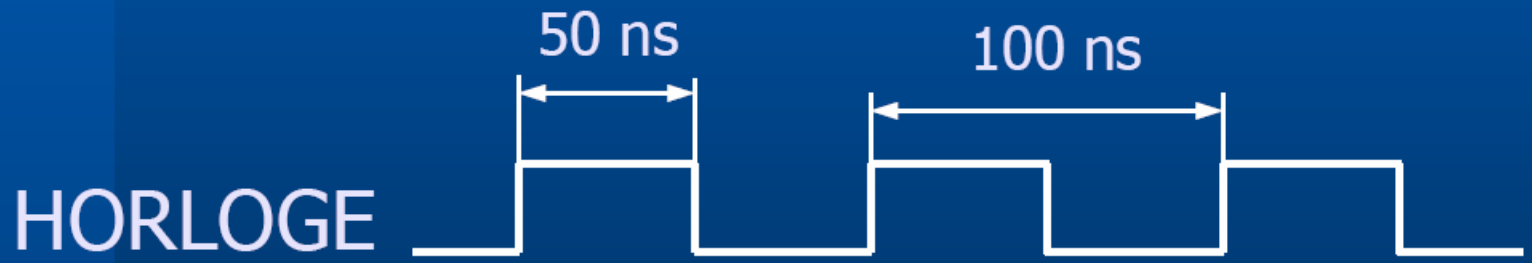
```
library ieee ;
use  ieee.std_logic_1164.ALL;

entity MAE is              --MAE = Machine A Etat
  port ( E,RST,HOR         : in   STD_LOGIC ;
         S                 : out STD_LOGIC  );
end MAE;
architecture COMPORTEMENT of MAE is
  signal REG_ETAT                : STD_LOGIC_VECTOR(2 downto 0);
  process (HOR,RST)
  begin
  if RST='0' then REG_ETAT <= "000";
                elsif (HOR'event and HOR='1') then
                        case REG_ETAT is
                        when "000" => S = '0' ;
                                      if E = '1' then REG_ETAT <= "001";
                                                 else REG_ETAT <= "000";
                                      end if ;
                        when "001" => S = '0' ;
                                      if E = '1' then REG_ETAT <= "010";
                                                 else REG_ETAT <= "000";
                                      end if ;
                        when "010" => S = '0' ;
                                      if E = '1' then REG_ETAT <= "011";
                                                 else REG_ETAT <= "000";
                                      end if ;
                        when "011" => S = '1' ;
                                      if E = '1' then REG_ETAT <= "011";
                                                 else REG_ETAT <= "100";
                                      end if ;
                        when "100" => S = '1' ;
                                      if E = '1' then REG_ETAT <= "011";
                                                 else REG_ETAT <= "101";
                                      end if ;
                        when "101" => S = '0' ;
                                      if E = '1' then REG_ETAT <= "011";
                                                 else REG_ETAT <= "000";
                                      end if ;
                        when others => REG_ETAT <= "00";
                        end case ;
  end if ;
  end process ;
end COMPORTEMENT ;
```

Les pièges VHDL

Description en VHDL d'une horloge :

```
Horloge <= not Horloge after 50 ns;
```

Le chronogramme est le suivant :



C'est une horloge à 10 Mhz

Les pièges VHDL

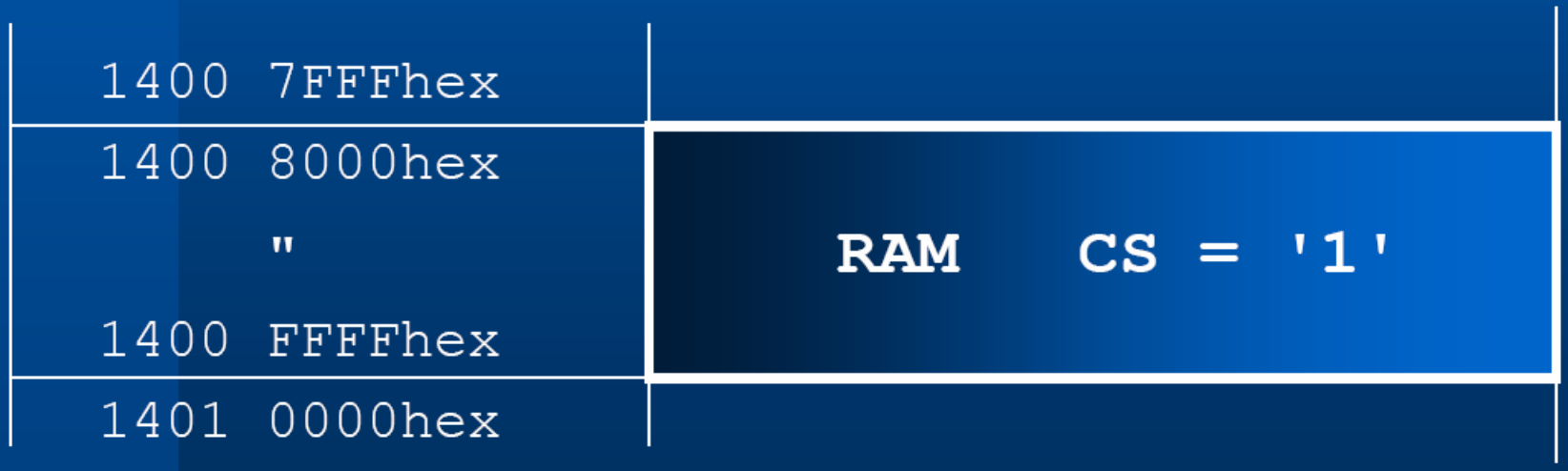
- Description correcte au sens du langage
- Description parfaitement simulable
- aucun circuit correspondant à cette fonctionnalité

La description est non synthétisable

Exemple : décodeur d'adresse

Nous allons décrire le décodeur d'adresse pour une RAM 32K

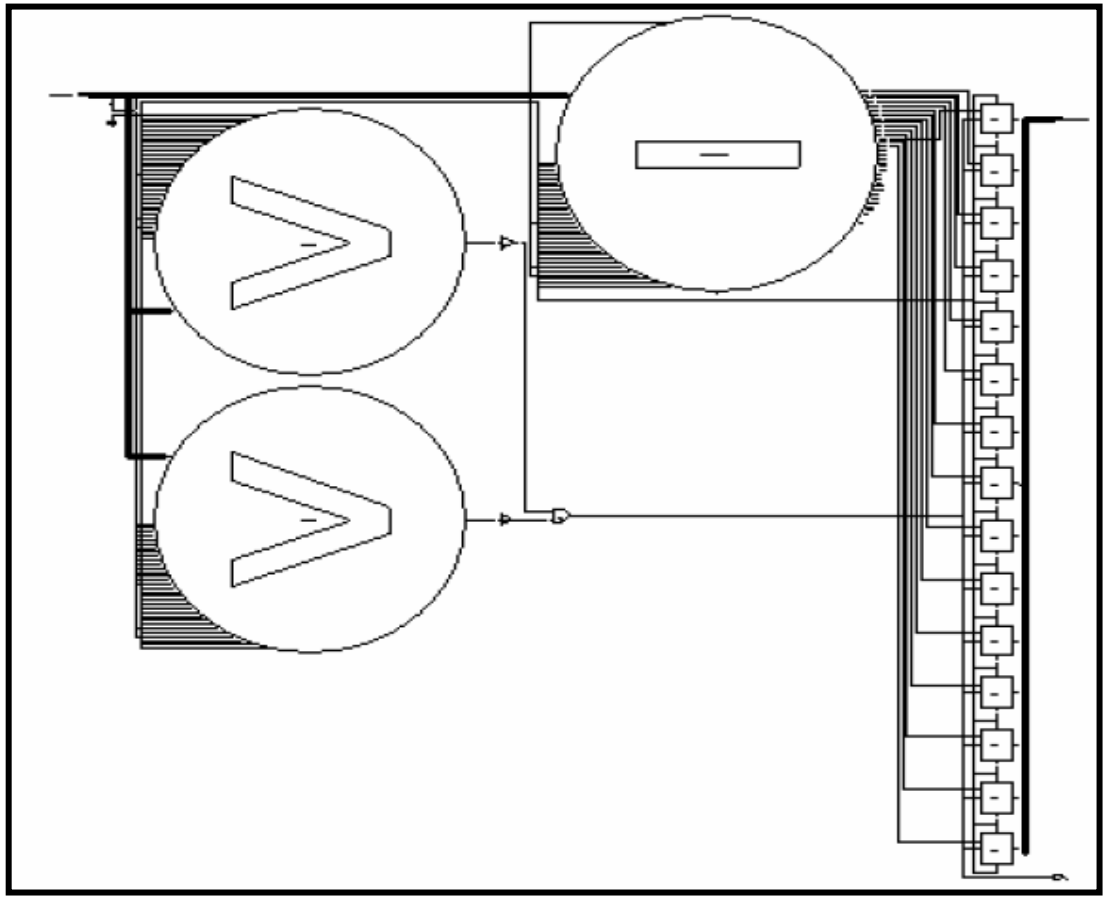
Plan d'adressage de la RAM :



Description VHDL

```
architecture Specific of Decode is
begin
  process(Adr_32)
  begin
    if (Adr_32 >= x"14008000")
                  and (Adr_32 <= x"1400FFFF") then
       CS <= '1';
       Adr_RAM <= Adr_32 - x"14008000";
    else
       CS <= '0';
    end if;
  end process;
end Specific;
```

Synthèse VHDL



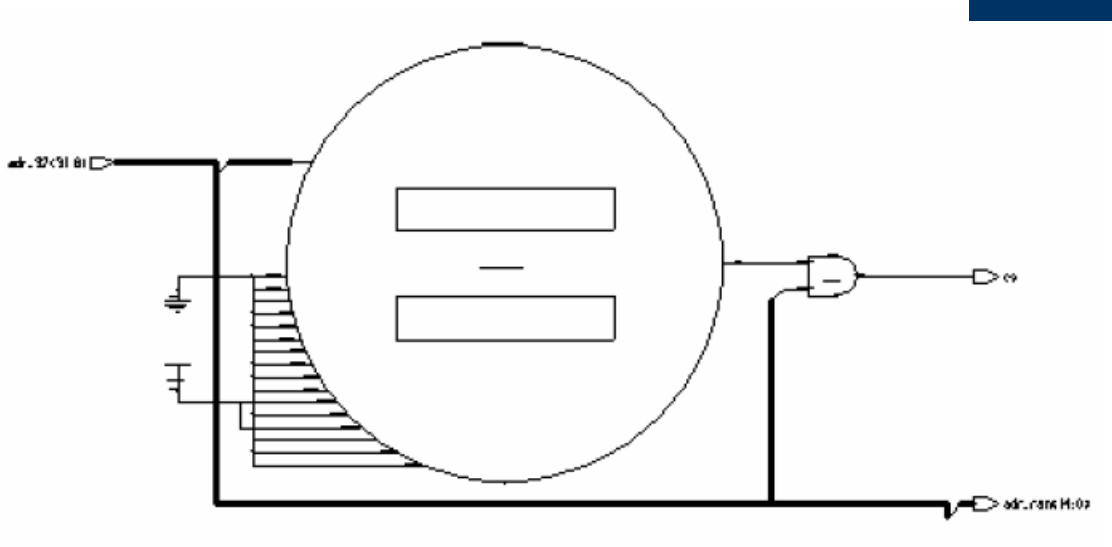
Technologie :
ALTERA
Flex10K

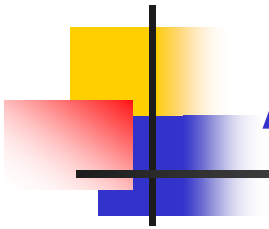
Description VHDL pensée

```
architecture Flot_Don of Decode is
begin

  CS <= '1' when (Adr_32(31 downto 16) = x"1400")
                                  and (Adr_32(15) = '1')
        else '0';

  Adr_RAM <=Adr_32(14 downto 0);

end Flot_Don;
```

Synthèse VHDL

Technologie :
ALTERA, Flex10K

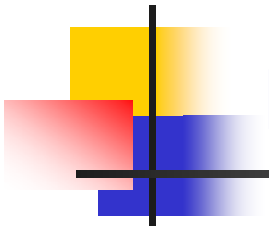




Autre problème

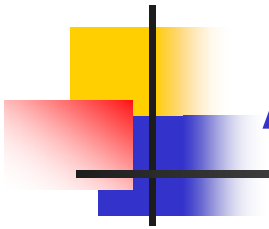
Nous allons démontrer par un 2ème exemple une autre problématique d'un langage de haut niveau tel le VHDL.

La description d'éléments mémoires



Description d’éléments mémoires

- En VHDL :
  - Pas de déclaration d'élément mémoire
- La description indique au synthétiseur :
  - Signal correspond à un élément mémoire
- Nécessaire de diriger le synthétiseur :
  - Obtenir l'élément mémoire désiré



Avertissement

Avec le langage VHDL :

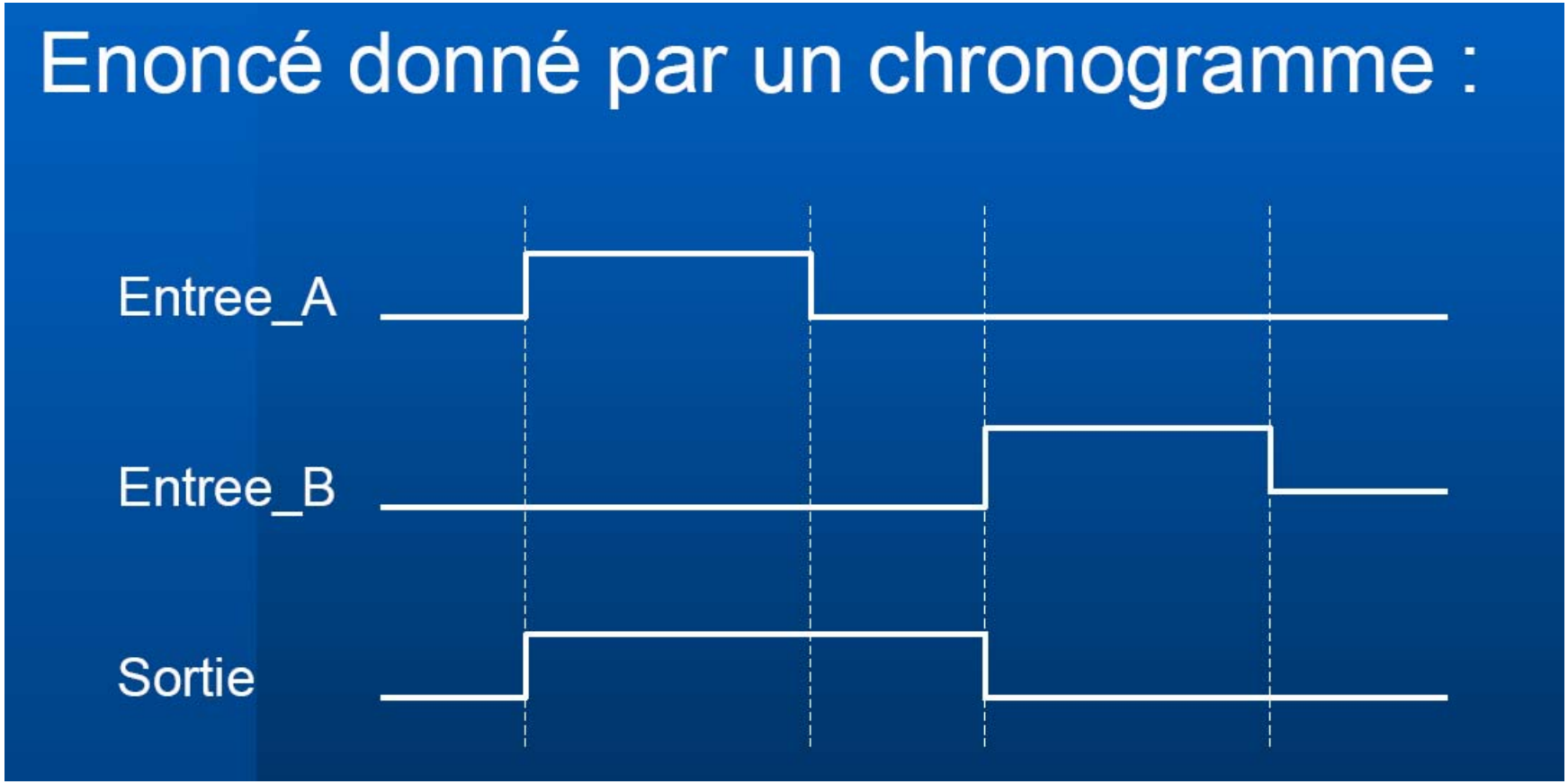
- Il est possible d'obtenir un élément mémoire lorsque vous ne le voulez pas !

INVERSEMENT

- Il est possible de ne pas avoir d'élément mémoire lorsque vous le voulez !

Problème

Enoncé donné par un chronogramme :

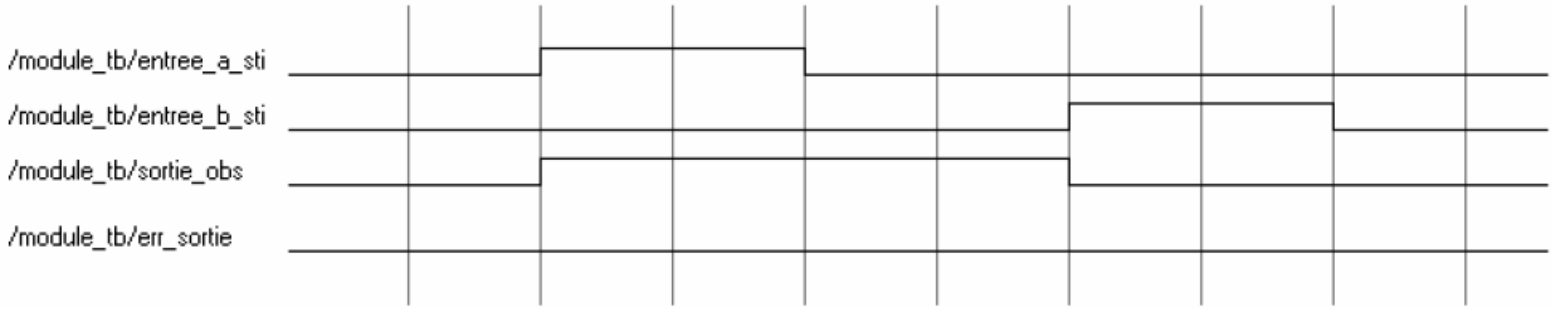


Description étudiant

```
architecture Comport of Module is
begin
  process(Entree_A, Entree_B)
  begin
     if Rising_Edge(Entree_A) then
          Sortie <= '1';
     elsif Rising_Edge(Entree_B) then
          Sortie <= '0';
     end if;
  end process;
end Comport;
```

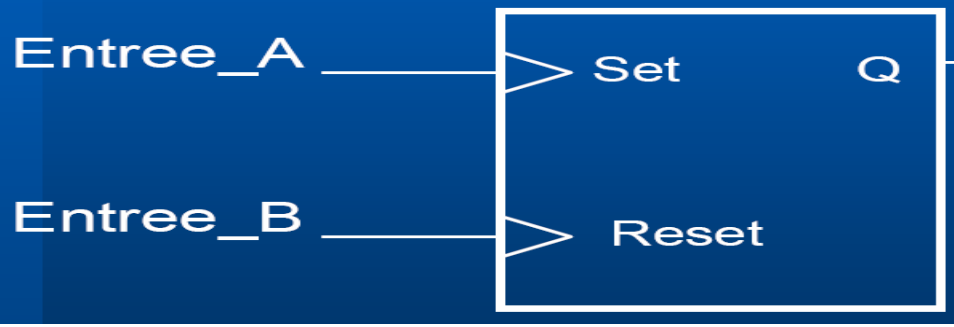
Simulation

- La description se simule correctement :



Matériel décrit/synthèse

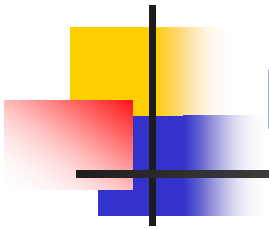
L'élément mémoire décrit est le suivant :



- Message de Leonardo :

```
-- Compiling root entity module(comport)
"F:/En_cours/Exe/Module.vhd",line 30:
   Error, sortie is clocked by multiple clocks.
```

Description non synthétisable



Bonne méthode

- Conception :
  - Pensez circuit => le fonctionnement correspond à une bascule RS
- Réalisation :
  - Décrire la bascule RS (sensible à des niveaux)
  - Simulation sera correcte
  - Synthèse sera correcte

Description d'une RS/synthèse

```
architecture Comport of Module is
begin
  process(Entree_A, Entree_B) --Bascule SR
  begin
     if Entree_A = '1' then
          Sortie <= '1'; --Set
     elsif Entree_B = '1' then
          Sortie <= '0'; --Reset
     end if;
  end process;
end Comport;
```