Il est impératif d'appliquer les consignes suivantes avant de débuter le TP :

- Créer sur votre répertoire personnel un répertoire intitulé **TP2VHDL**.
- Créer un projet de nom **exercicesTP2** pour la cible désirée.

Pour chaque exercice :

- Ouvrir l'éditeur de texte.
- Sauver le nom du fichier du **même nom que l'entité** (.vhd).
- Choisir l'option **Set as Top-Level Entity** du menu **Project**.
- Compiler le programme.
- Réaliser un fichier de simulation du **même nom que l'entité** (.vwf).
- Simuler.

TP 2 : Les instructions concurrentes en VHDL

Les instructions concurrentes interviennent à l'intérieur même d'une architecture dans la description du fonctionnement d'un circuit.

Ces instructions ont pour but d'affecter des valeurs à des composants ou de réaliser des connexions électriques. **Elles n'ont donc pas d'ordre d'exécution** et sont effectuées en parallèle.

Étant décrites dans le corps de l'architecture, ces instructions **ne peuvent porter que sur des signaux**.

Les principales instructions concurrentes sont :

- Les affectations simples (<=).
- Les affectations conditionnelles (WHEN - ELSE).
- Les affectations sélectives (WITH-SELECT).
- Les processus (cf. instructions séquentielles).
- Les instanciations de composants : (qui consistent à utiliser un sous-ensemble décrit en VHDL comme composant dans un ensemble plus vaste).
- Les instructions « GENERATE » : (afin de créer des structures régulières).
- Les définitions de blocs (une architecture peut être divisée en blocs de façon à constituer une hiérarchie interne dans un composant complexe).

1 – Les affectations simples :

Les affectations simples traduisent une simple interconnexion entre deux équipotentielles. L'opérateur correspondant est noté <=.

```
Usage :   nom <= valeur;

Exemple : s <= A AND B;
```

2 – L'affectation conditionnelle WHEN…ELSE :

La structure « WHEN...ELSE » permet de spécifier une expression (ou une valeur) différente à chaque cas rencontré. Ces cas peuvent être un test ou une combinaison de tests (AND, OR, ...).

L'expression spécifiée doit être une expression logique de choix; **le dernier cas présenté recouvre tous les autres cas non décrits**.

```
Usage :   nom_var <=val_1 WHEN condition_1 ELSE
          val_2 WHEN condition_2 ELSE
          ...
          val_sinon; -- Dernier cas regroupant tous les cas non cités
```

```
Exemple : ARCHITECTURE archi OF mux IS
          BEGIN
          s <= e0  WHEN c="00" ELSE
               e1  WHEN c="01" ELSE
               e2  WHEN c="10" ELSE
               e1;
          END archi;
```

Exercice : Construire à l'aide de l'instruction WHEN…ELSE le multiplexeur 2 vers 1 traduisant la condition suivante (**description comportementale**) :

S=1 si e0=1 et c=0 ou e1=1 et c=1
S=0 sinon

C	e0	e1	S
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

3 – L'affectation sélective WITH…SELECT …WHEN :

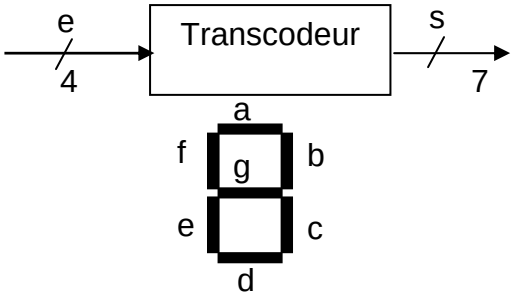
Cette instruction permet de choisir la valeur à affecter à un signal en fonction des différentes valeurs possibles d'une expression. **Les conditions peuvent porter sur une ou plusieurs valeurs et sont séparées par des virgules**.

L'usage type de cette instruction est la réalisation de décodeur ou de multiplexeur.

```
Usage :   WITH expression SELECT
          nom_var <= val_1 WHEN valeur1_1 ou valeur1_2, -- Une ou
          val_2 WHEN valeur2_1 ou valeur2_2,-- plusieurs valeurs
          ...
          val_sinon WHEN OTHERS; -- autres cas non listés
```

```
Exemple :  WITH e SELECT
           S <= A WHEN 0,
                B WHEN 1,
                D WHEN OTHERS;
```

Exercice : Construire à l'aide de l'instruction WITH…SELECT…WHEN le transcodeur DCB 7 segments suivant :



e3	e2	e1	e0	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1

4 – L'instruction "GENERATE" :

Les instructions « GENERATE » permettent de créer des structures régulières en **dupliquant un bloc d'instructions concurrentes** un certain nombre de fois ou en créant un tel bloc si une condition est réalisée. Il existe deux types d'instructions « GENERATE » : Les instructions à structure répétitive et à structure conditionnelle.

Ces instructions s'appliquent parfaitement dans des descriptions structurelles de circuits et pour des circuits comme les registres ou les multiplexeurs.

Structure répétitive :

```
Usage :    (étiquette:) FOR variable IN debut TO fin GENERATE
           instructions concurrentes;
           END GENERATE (étiquette);
```

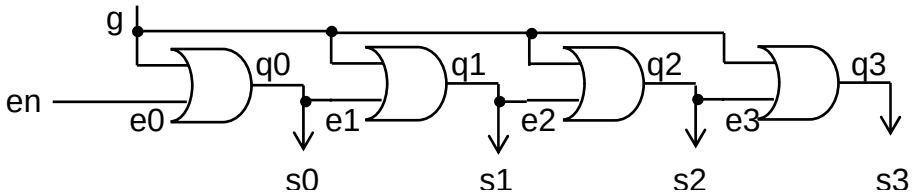
```
Exemple :  boucle1 : FOR i IN 0 TO 4 GENERATE
           q(i) <= e(i-1) AND A ;
           END GENERATE boucle1;
```

Structure conditionnelle :

```
Usage :    (étiquette:) IF condition GENERATE
           instructions concurrentes;
           END GENERATE (étiquette);
```

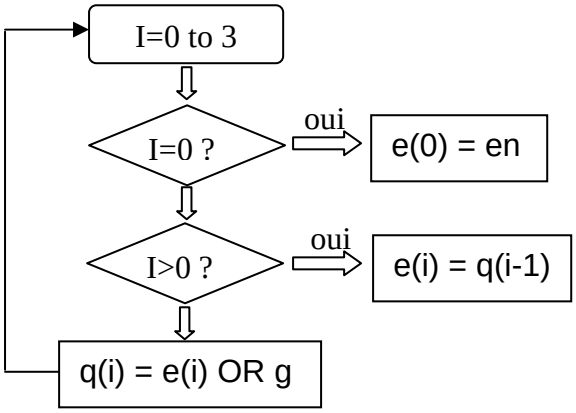
Dans les deux structures, la variable de boucle ne se déclare pas et l'étiquette est obligatoire.

Exercice : Construire, en complétant l'entité ci dessous, la structure répétitive suivante :




```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
ENTITY repet IS
PORT(
en, g : IN STD_LOGIC; -- Entrées binaires
s : OUT STD_LOGIC_VECTOR (3 DOWNTO 0) -- Bus 4 bits
);
END repet;
ARCHITECTURE archi OF repet IS
SIGNAL q : STD_LOGIC_VECTOR (3 DOWNTO 0);
SIGNAL e : STD_LOGIC_VECTOR (3 DOWNTO 0);
BEGIN
```

Quels sont les rôles des signaux q et e ? pourquoi sont ils définis en STD_LOGIC_VECTOR (et non IN ou OUT) ?
Compléter l'architecture à l'aide de l'organigramme ci-dessous :



5 – Les blocs et instanciations :

L'instanciation consiste à utiliser un sous-ensemble (entité et architecture décrits en VHDL) comme composant dans un ensemble plus vaste.

Trois opérations sont nécessaires pour une instanciation :

- Le couple entité/architecture du sous-ensemble doit être crée et annexé à une librairie de l'utilisateur.
- Ce sous-ensemble doit être déclaré comme composant dans l'ensemble qui l'utilise.

➢ Chaque exemplaire de ce sous-ensemble doit être connecté à des équipotentielles utilisés dans le schéma principal.

La déclaration du composant est à placer dans la partie déclarative de l'architecture du circuit utilisateur (ou dans un package visible à l'aide de l'instruction USE).

```
Syntaxe :  component nom_composant -- nom égal à celui de l'entité
           port (liste_ports) ; -- Liste identique à celle de l'entité
           END component ;
```

Après déclaration, ce sous-ensemble peut être utilisé comme un composant prédéfini (par exemple, une bascule D peut être soit définie puis utilisée par le principe d'instanciation, soit directement utilisée à partir de la bibliothèque ALTERA où elle est déjà prédéfinie).

L'utilisation du composant se fait de la façon suivante (**étiquette obligatoire**) :

```
Syntaxe :  (étiquette:) nom_composant PORT MAP (liste_associa.) ;
```

La liste d'association a pour but d'établir la correspondance entre les équipotentielles du schéma et les ports du sous-ensemble. Cette association se fait soit par position (les signaux à connecter apparaissent dans l'ordre des ports auxquels ils doivent correspondre) soit de façon explicite (à l'aide de l'opérateur =>).

```
Exemple :  ARCHITECTURE archi OF repet IS
           SIGNAL q : STD_LOGIC_VECTOR ( 3 DOWNTO 0);
           SIGNAL e : STD_LOGIC_VECTOR ( 3 DOWNTO 0);
                 COMPONENT ou – Déclaration du composant
                 PORT (a, b : IN STD_LOGIC ;
                 a_ou_b : OUT STD_LOGIC) ;
                 END COMPONENT;
           BEGIN
           s<=q;
           gen_circuit : FOR i IN 0 TO 3 GENERATE
           gen_test1 : IF i=0 GENERATE
           e(0) <= en;
           END GENERATE gen_test1;
           gen_test2 : IF i>0 GENERATE
           e(i) <= q(i-1);
           END GENERATE gen_test2;
           appel : ou PORT MAP (e(i), g, q(i)); -- Appel du composant
           END GENERATE gen_circuit; -- Instanciation par position
           END archi;
```

Un bloc est utilisé dans une architecture de façon à créer une hiérarchie dans la structure d'un circuit. L'intérêt de cette démarche est d'améliorer la lecture et la visibilité des objets et de leurs portées.
Ces blocs permettent de regrouper plusieurs instructions concurrentes et de leur faire partager certaines descriptions locales non visibles du reste de la description.

```
Syntaxe :  (étiquette:) BLOCK (expression_de_garde)
           déclaration des signaux, composants ... ;
           ...
           BEGIN
           ...
           instructions concurrentes;
           ...
           END BLOCK (étiquette);
```

L'étiquette reste obligatoire.

Il est impératif d'appliquer les consignes suivantes avant de débuter le TP :

- Créer sur votre répertoire personnel un répertoire intitulé **TP3VHDL**.
- Créer un projet de nom **exercicesTP3** pour la cible désirée.

Pour chaque exercice :

- Ouvrir l'éditeur de texte
- Sauver le nom du fichier du **même nom que l'entité** (.vhd).
- Choisir l'option **Set as Top-Level Entity** du menu **Project**.
- Compiler le programme.
- Réaliser un fichier de simulation du **même nom que l'entité** (.vwf).
- Simuler.

TP 3 : Les instructions séquentielles en VHDL

Les instructions séquentielles permettent d'appliquer à la description d'une partie d'un circuit une démarche algorithmique. Elles affectent **un ordre précis** aux instructions à réaliser. L'ordre d'écriture n'est pas indifférent !

Ces instructions sont au cœur des descriptions comportementales des circuits et il est essentiel de connaître parfaitement l'architecture du circuit à décrire (signaux d'horloge, registres, blocs combinatoires).

Les instructions séquentielles sont **internes aux processus, aux procédures et aux fonctions**.

Les principales instructions séquentielles sont :

- L'affectation séquentielle (<=) : Identique à l'affectation concurrente (seule sa position en dehors ou dans un module séquentiel distingue ces deux types).
- L'affectation d'une variable (:=) : C'est toujours une instruction séquentielle.
- Les tests (IF et CASE).
- Les contrôles de boucle (LOOP, FOR et WHILE).

1 – Le processus (process) :

Un process permet de définir un comportement qui doit avoir lieu lorsque le processus est activé. Ce comportement est spécifié à l'aide d'instructions séquentielles exécutées dans le process. C'est donc un **liste d'instructions séquentielles**.

Lorsque toutes les instructions sont exécutées à l'intérieur d'un process, celui-ci recommence. L'affectation des sorties ne se fait qu'à la fin du process.

Un process est une liste d'instructions séquentielles mais le process en lui même est une instruction concurrente (plusieurs process peuvent s'exécuter en parallèle).

```
Usage :   (étiquette:) PROCESS (signal1, signal2,...) -- Etiq. Facult.
          déclaration éventuelle des variables;
          ....
          BEGIN
          ...
          instructions;
          ...
          END PROCESS (étiquette);
```

Toutes les entrées qui modifient les sorties du process doivent être déclarées dans ce que l'on appelle **la liste des sensibilités** (liste signal1, signal2,...).
La liste des sensibilités indique donc au processus les signaux dont les modifications sont à prendre en compte pour évaluer son contenu (exemple : Horloge pour un système synchrone). L'exécution du process **reprend lorsqu'une des variables de cette liste change d'état**.

2 – L'instruction IF....THEN....ELSE....END IF :

Cette instruction permet de **sélectionner une ou plusieurs instructions** à exécuter **en fonction de valeurs prises** par une ou des conditions.

Afin d'éviter des erreurs, la description doit tenir compte de tous les cas et l'instruction ELSE prend alors en charge tous les cas non traités (c'est une instruction classique identique à celle rencontrée en C).

```
Usage :   IF condition1 THEN
          instructions séquentielles;
          ELSIF condition2 THEN
          instructions séquentielles; --Plusieurs instructions possibles
          ELSIF condition3 THEN    -- entre THEN et ELSE ou ELSIF
          instructions séquentielles;
          ...
          ELSE
          instructions séquentielles;
          END IF; -- Un seul END IF même pour plusieurs ELSIF
```

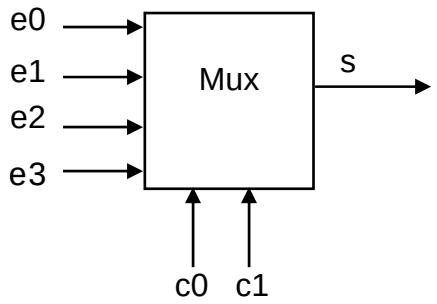
```
Exemple : IF A>B THEN
          S<='1';
          ELSIF A=B THEN
          S<='0';
          END IF;
```

Les conditions (condition1, condition2,…) sont des expressions logiques ou des combinaisons entre tests logiques (and, or, …).

De même, il est possible d'imbriquer plusieurs boucles IF les unes dans les autres :

```
Usage :   IF ... THEN
             IF ... THEN
             ELSIF ... THEN
             END IF;
          ELSE
          END IF;
```

Exercice : Construire le multiplexeur 4 vers 1 à l'aide de la structure IF…THEN…



Principe :

1 sortie s, 2entrées de sélection c
et 4 entrées d'informations e

Si c1c0=00 ⇨s=e0 Si c1c0=10 ⇨s=e2
Si c1c0=01 ⇨s=e1 Si c1c0=11 ⇨s=e3

3 – L'instruction CASE...WHEN.....END CASE :

Cette structure est utilisée lorsqu'une expression unique est à évaluer. Elle permet de **sélectionner une ou des instructions** à exécuter en fonction **des valeurs prises par une seule expression**.

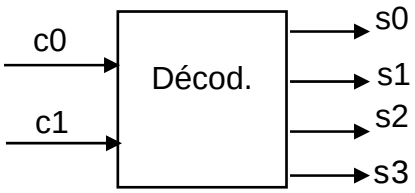
Comme pour l'instruction IF....THEN....ELSE l'option WHEN OTHERS permet de prendre en compte tous les cas non listés.

```
Usage :   CASE expression IS -- Une seule expression (ou port)
          WHEN choix1_1 ou choix1_2 ou …=> instructions séq.;
          WHEN choix2_1 ou choix2_2 ou …=> instructions séq.;
           ....
          WHEN OTHERS => instructions séq.;
          END CASE;
```

```
Exemple : CASE a IS
          WHEN "00" => s<=e1;
          WHEN "01" => s<=e2;
          WHEN OTHERS => s<='0';
          END CASE;
```

Cette structure est par exemple bien adaptée à la réalisation de décodeurs.

Exercice : Construire le décodeur 1 parmi 4 l'aide de la structure CASE ..IS…



Principe :

4 sorties s et 2 entrées de sélection c

Si c1c0 = 00 : s0=1 (les autres à 0)
Si c1c0 = 01 : s1=1 (les autres à 0)

4 – L'instruction WAIT.....UNTIL :

Cette instruction permet de bloquer l'exécution d'un traitement tant que l'expression spécifiée ne présente pas un front montant. Cette instruction est une alternative à l'usage de l'attribut EVENT.

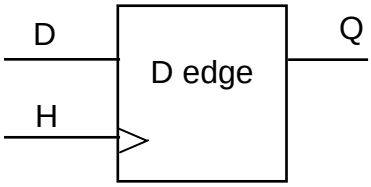
Pour utiliser une instruction WAIT, il ne doit **pas y avoir de liste de sensibilité** indiquée au processus et l'instruction doit être **placée en début de process**.

```
Usage :    WAIT UNTIL condition;
```

La condition WAIT UNTIL synchronise tous les traitements du process : Il est donc forcément synchrone.

D'autres instructions WAIT existent également : WAIT ON ou encore WAIT FOR pour un délai (par exemple WAIT FOR 10 ns).

Exercice : Construire une bascule D edge triggered à l'aide de WAIT … UNTIL :



Principe :

1 entrée d'information, 1 horloge et 1 sortie.

Q recopie D sur un front montant de H (mémoire sinon).

Exercice : Réaliser la même bascule D à l'aide de l'instruction IF…THEN… ELSE et de l'attribut EVENT.

5 – Les instructions de boucle :

Les boucles permettent de répéter un schéma de traitement (comme l'instruction concurrente GENERATE) soit de façon indexée à un indice (FOR) soit de façon continue jusqu'à une condition de sortie (EXIT). Trois types de boucles existent :

- Les boucles WHILE (condition de maintien).
- Les boucles FOR (itératives).
- Les boucles simples (sans condition de sortie).

L'instruction FOR...LOOP....END LOOP :

C'est une boucle pour laquelle **le nombre d'itérations est précisé** à l'aide d'une variable de boucle (que l'on ne déclare pas dans le programme).

```
Usage :   (étiquette:) FOR indice IN mini TO maxi LOOP
          liste d'instructions;
          END LOOP (étiquette);
```

L'étiquette est facultative et l'indice peut également être décrémenté à l'aide de l'instruction maxi DOWNTO mini.

```
Exemple : FOR i IN 0 TO 3 LOOP
          S(i) <= a(i) OR b(i);
          END LOOP;
```

L'instruction WHILE...LOOP....END LOOP :

C'est une boucle pour laquelle **une condition de maintien** dans cette boucle est précisée.

```
Usage :   (étiquette:) WHILE condition LOOP
          liste d'instructions;
          END LOOP (étiquette);
```

Là encore, l'étiquette est facultative est la condition de maintien peut être une combinaison de tests logiques.

L'instruction LOOP....END LOOP :

Cette instruction correspond à une boucle simple **sans condition de sortie**.

```
Usage :   (étiquette:) LOOP
          liste d'instructions;
          END LOOP (étiquette);
```

Les instruction EXIT et NEXT :

Pour ne pas rester indéfiniment dans une boucle simple, **une condition de sortie** est nécessaire (EXIT) :

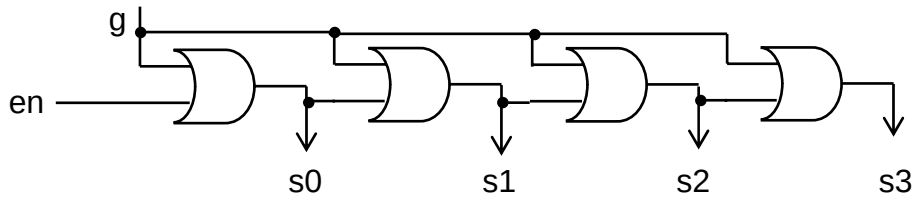
```
Usage :   EXIT (étiquette:) WHEN condition;
```

De la même façon, l'option NEXT permet de passer à l'itération suivante dans une boucle FOR :

```
Usage :   NEXT (étiquette:) WHEN condition;
```

Dans ces deux cas, l'étiquette est nécessaire afin de repérer la boucle concernée. De plus, la condition de sortie peut être une combinaison de conditions ou de tests.

Exercice : Réaliser à partir d'une boucle FOR la structure récursive en portes OR.



6 – Les procédures et les fonctions :

Pour être lisible et compréhensible, un programme doit être constitué de petits modules testés et mis au point séparément puis réutilisables pour d'autres programmes.
Les outils de base de cette **construction modulaire** sont les **sous programmes, les procédures ou les fonctions** (et également les packages des librairies).
Les fonctions et les procédures sont deux types de sous programmes qui ne diffèrent principalement que par leurs principes d'échange d'informations.

Les fonctions :

Une fonction a pour but de retourner au programme principal une valeur unique**.** Elle possède donc un type. Elle peut recevoir des arguments (uniquement des signaux ou des constantes) dont les valeurs sont transmises lors de **l'appel mais ne peut pas modifier les valeurs des données qu'elle reçoit**.
Toute variable locale déclarée au sein de la fonction est détruite lorsque la fonction se termine. Lors de son utilisation, le nom de la fonction peut apparaître partout où une valeur du même type peut être utilisée.
Le corps d'une fonction ne doit pas contenir d'instruction WAIT.

```
Déclaration :       FUNCTION nom (liste des paramètres) RETURN
                    nom_type; --Type (bit,…)

Corps de la fonction :  FUNCTION nom (liste des paramètres)
                        RETURN nom_type IS -- Nature des arguments
                        déclarations;
                        BEGIN
                        instructions séquentielles;
                        RETURN nom_valeur_calculée
                        END nom;

Utilisation :       nom (liste paramètres) -- Exemple : s<=nom(a) ;
```

Les procédures :

Une procédure peut recevoir des constantes, des variables ou des signaux pouvant être déclarés en entrée (mode in), en sortie (out) ou bidirectionnel (inout) (sauf les

constantes qui sont toujours en mode in). La nature des paramètres doit donc être précisée dans la liste d'appel.
Les variables locales déclarées au sein d'une procédure sont, comme pour la fonction, détruites en fin d'usage.
Une procédure peut être appelée **par une instruction concurrente ou par une instruction séquentielle**. Toutefois, si l'un des arguments est une variable, seule une instruction séquentielle peut appeler la procédure.

La correspondance entre paramètres réels (pour l'appel) et paramètres formels (description interne) se fait soit par positions, soit par associations (=>).

Le corps d'une procédure peut contenir l'instruction WAIT.

```
Déclaration :      PROCEDURE nom (liste paramètres);

Corps de la procédure :PROCEDURE nom (liste paramètres) IS
                       déclarations éventuelles; -- Nature des signaux
                       BEGIN
                       instructions séquentielles;
                       END PROCEDURE nom;

 Utilisation :     nom (liste paramètres);
```

Pour le corps de la procédure (uniquement), la liste des paramètres doit contenir la nature des arguments.

```
Exemple :  PROCEDURE demo (signal a, b : IN BIT; -- 2 bits en entrée
                           signal s : OUT BIT); -- 1 bit en sortie
```

```
Exemple :  ARCHITECTURE archi OF somme IS
                PROCEDURE calc (signal e : IN BIT_VECTOR;
                signal resu : OUT BIT) IS -- Nature des variables
                variable somlog : bit :='0';
                BEGIN
                somlog :=  e(0) OR e(2) OR e(4) OR e(6);
                resu <= somlog
                END PROCEDURE calc;
           BEGIN
           PROCESS (a)
           BEGIN
           calc (a, s); -- équivalent à calc (e => a , resu => s)
           END PROCESS;
           END archi;
```

Il est impératif d'appliquer les consignes suivantes avant de débuter le TP :

- Créer sur votre répertoire personnel un répertoire intitulé **TP4VHDL**.
- Créer un projet de nom **exercicesTP4** pour la cible désirée.

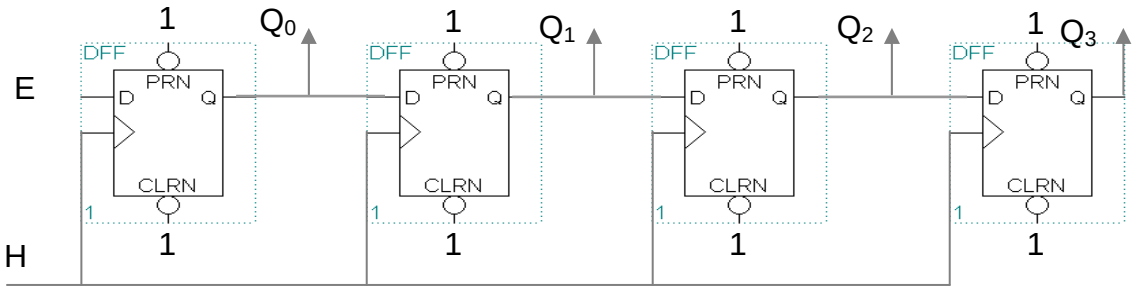
Pour chaque exercice :

- Ouvrir l'éditeur de texte.
- Sauver le nom du fichier du **même nom que l'entité** (.vhd).
- Choisir l'option **Set as Top-Level Entity** du menu **Project**.
- Compiler le programme.
- Réaliser un fichier de simulation du **même nom que l'entité** (.vwf).
- Simuler.

TP 4 : Systèmes séquentiels en VHDL

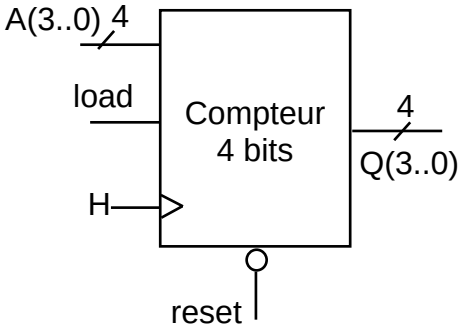
Exercice : Réaliser le registre à décalage à droite 4 bits ci-dessous et possédant les caractéristiques suivantes :

- Deux entrées E (entrée série) et H (horloge).
- Une sortie Q codée sur 4 bits (Q_0 à Q_3).
- Une remise à zéro du registre active à 0.



Exercice : Réaliser un compteur synchrone 4 bits uniquement à partir d'une description comportementale et possédant :

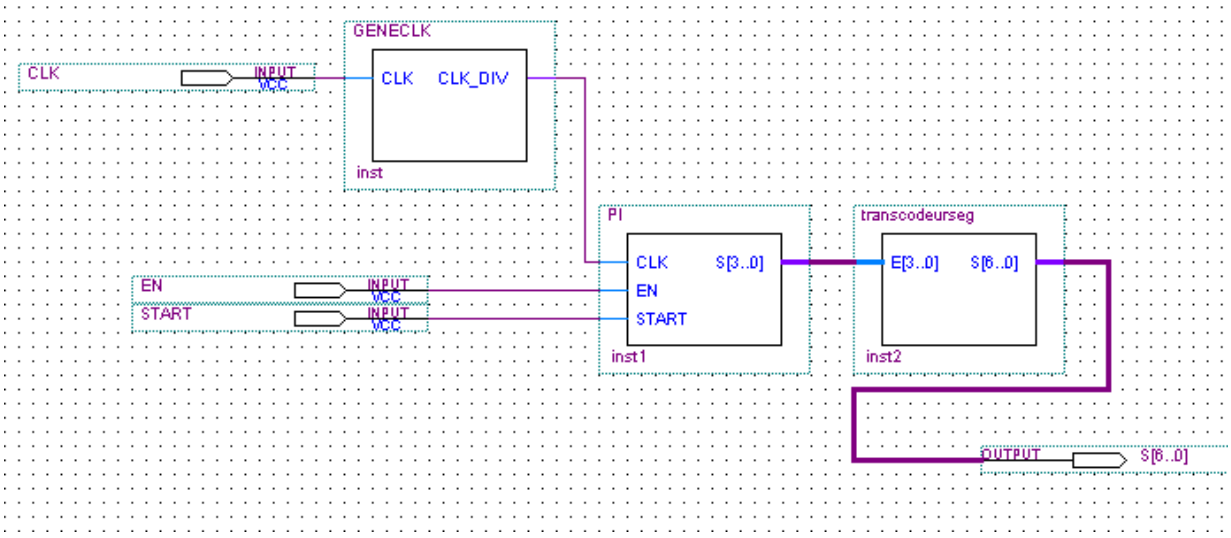
- Une entrée d'horloge H (front montant).
- Une entrée de chargement load synchrone (active à 1) permettant de charger la valeur A (4 bits) sur le front montant de h .
- Une entrée reset active à l'état bas.
- Une sortie Q codée sur 4 bits (Q_0 à Q_3).



Exercice : Réalisation d'un générateur du nombre PI.

Ce sujet étant un projet à part entière destiné à être chargé dans une cible, il est nécessaire de définir un nouveau projet pour lequel l'entité de hiérarchie la plus élevée sera l'assemblage graphique des modules développés en VHDL (cf TP1).

On se propose de réaliser un compteur permettant d'afficher toutes les secondes sur un afficheur 7 segments la suite du nombre PI : 3,1415926. Le schéma synoptique est le suivant :



Il comporte :

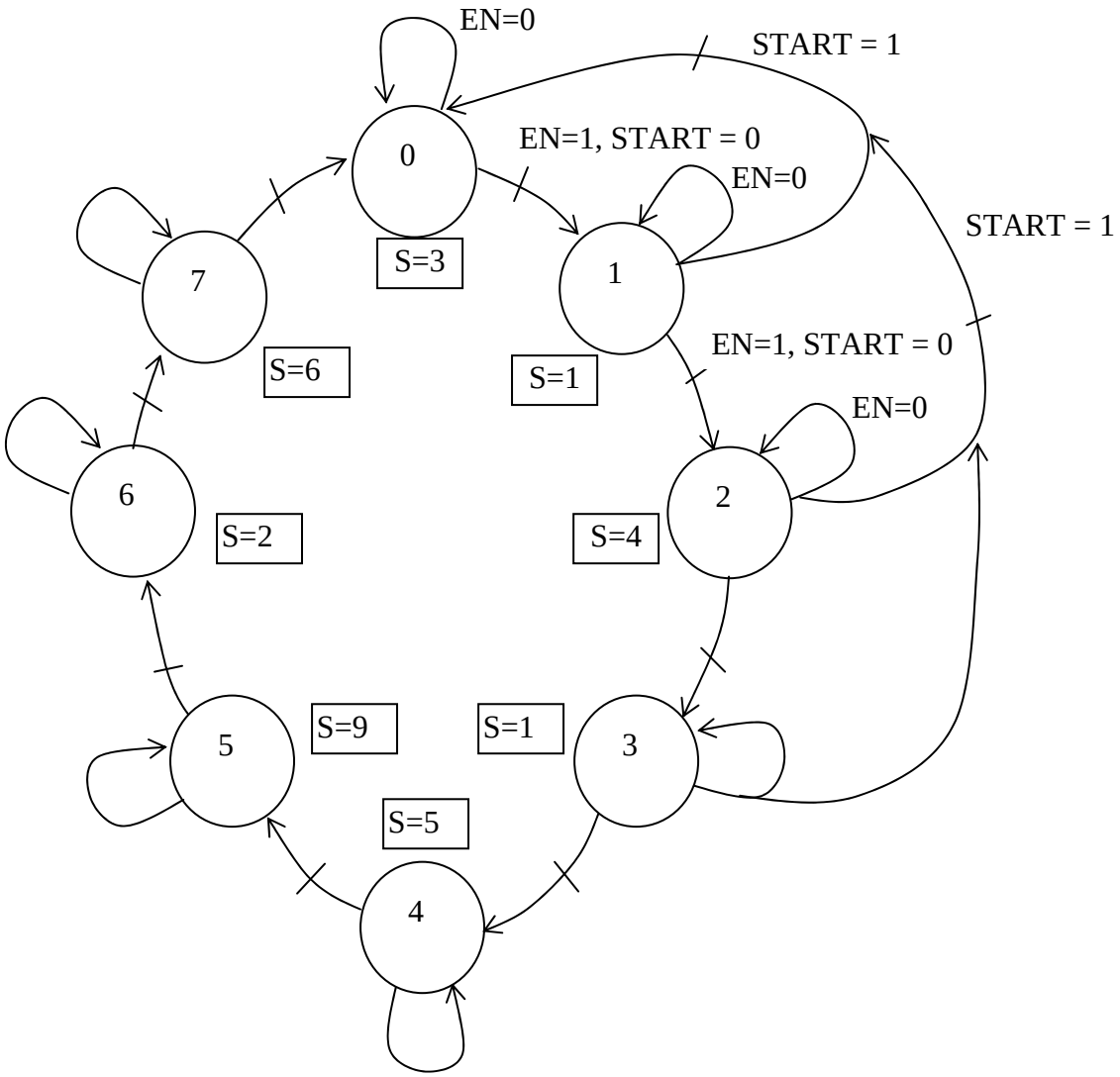
- Une base de temps de 1s (GENECLK) permettant, à partir de l'horloge de la carte de développement, d'obtenir un signal possédant un état haut et un état bas d'une durée de 0,5s chacun.
- Un transcodeur DCB 7 segments (TRANSCODEURSEG) permettant d'afficher sur un afficheur 7 segments la valeur binaire reçue et codée sur 4 bits.
- Un module de génération (PI) permettant à chaque front d'horloge de dénombrer les états voulus. Il comporte également :
  - Une entrée EN telle que : Si EN=1, le compteur passe à l'état suivant, sinon il reste bloqué sur son état.
  - Une entré START telle que : Si START = 1, le compteur reprend la séquence depuis le début.

Réaliser et tester séparément les modules GENECLK, PI et TRANSCODEURSEG. Construire en particulier le module PI à l'aide du diagramme d'état ci-dessous et de l'instruction CASE…WHEN.

Assembler sous l'éditeur graphique ces trois modules et affecter les broches concernées (CLK, afficheurs, signaux de commande).

Charger et tester le séquenceur.

Diagramme d'état :

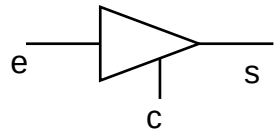


Pour en savoir plus sur la logique trois états

La logique trois états est une logique qui possède, en plus des états binaires 0 et 1, un état supplémentaire appelé **haute impédance**.

Cet état haute impédance est égal à un niveau de tension flottant et est donc équivalent à une sortie en l'air donc **déconnectée**.

Porte de base en logique trois états :

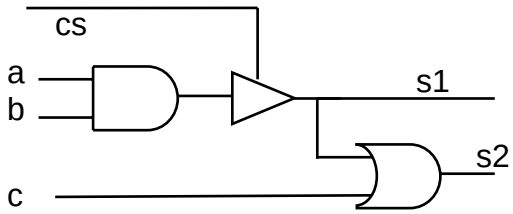


c=1 ⇨ basse impédance (normal)
c=0 ⇨ haute impédance (déconnecté)

e	c	s
0	1	0
1	1	1
X	0	HiZ

Lorsque plusieurs circuits sont connectés (soudés) à un bus, les sorties peuvent prendre des états différents et donc créer des conflits.
Cette logique est utilisée pour éviter les conflits de données sur les bus. En effet, lorsque un circuit est en état basse impédance, tous les autres sont en haute impédance et sont donc considérés comme déconnectés.

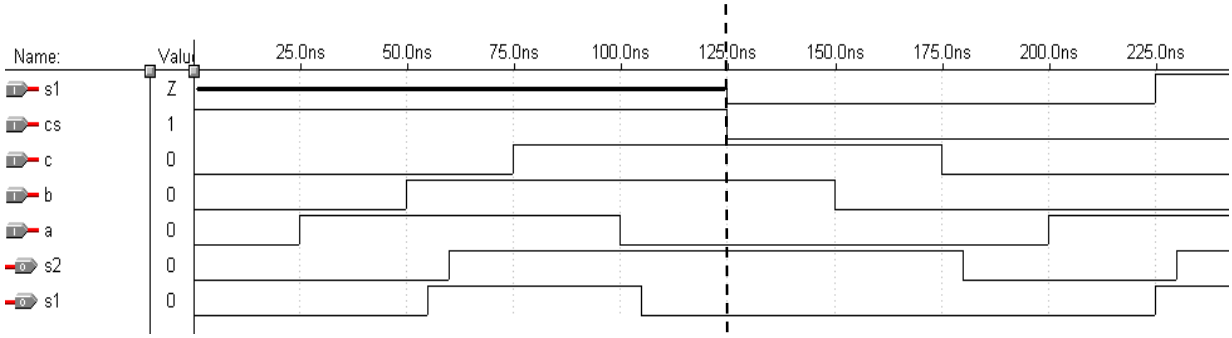
La logique trois états est gérée par le langage VHDL grâce à un composant prédéfini dans la bibliothèque ALTERA.



Grâce à la commande trois états (cs), s1 peut être utilisée soit en entrée, soit en sortie

cs=1⇨ basse impédance : s1 = a . b et s2 = a . b + c
cs=0 ⇨ haute impédance : s2 = s1 + c

Le composant gérant la logique trois états s'appelle TRI dans la bibliothèque ALTERA (c:\maxplus2\vhdl93\altera\maxplus2.vhd).

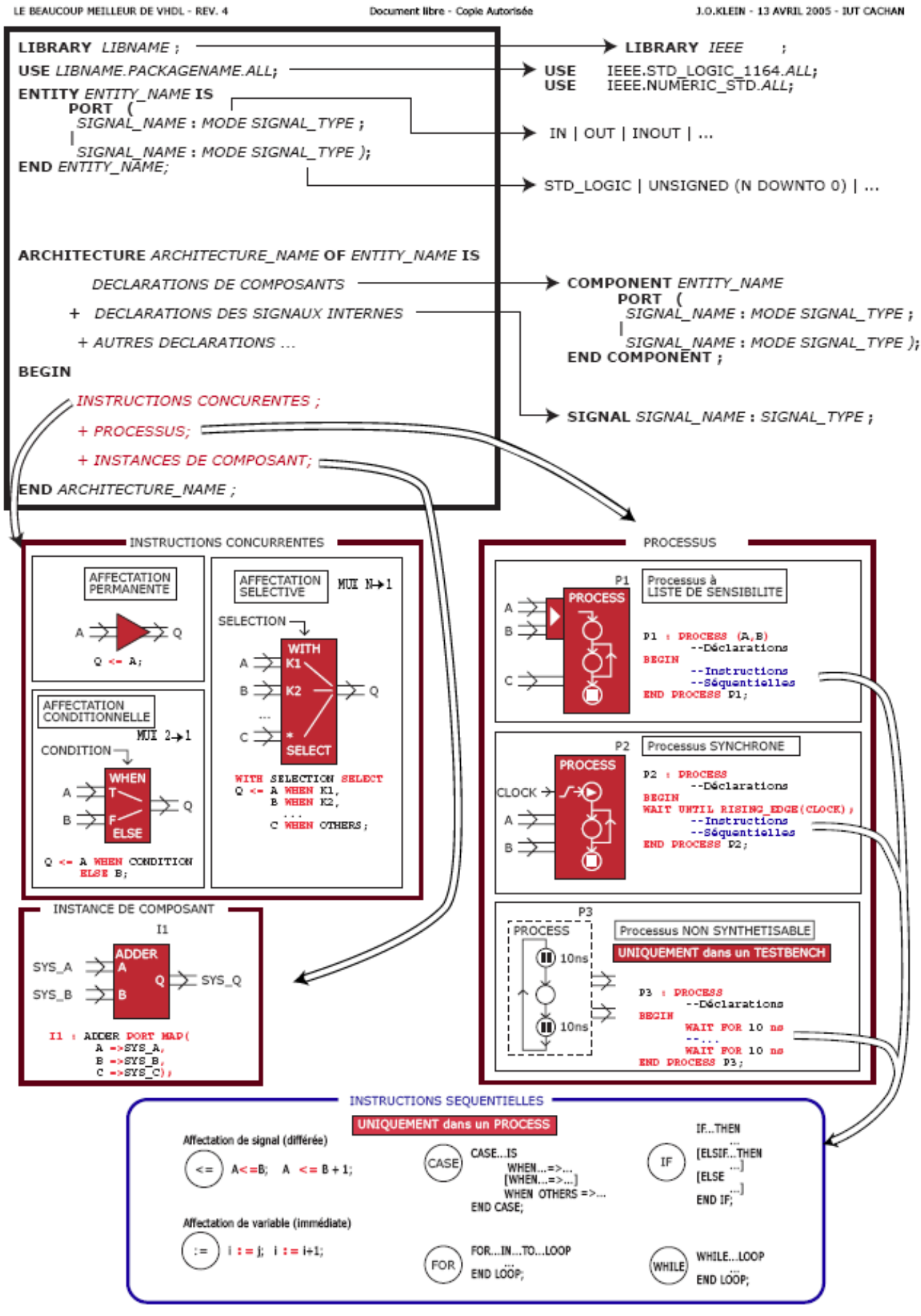


cs = 1 : basse impédance
s1 = a.b

cs = 0 : haute impédance
s2 = s1 + c

Exemple :

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
LIBRARY altera;
USE altera.altera_primitives_components.all;
ENTITY troisetats IS
PORT(
a, b, c, cs : IN STD_LOGIC;
s2 : OUT STD_LOGIC;
s1 : INOUT STD_LOGIC -- s1 entrée ou sortie INOUT
);
END troisetats;
ARCHITECTURE archi OF troisetats IS
SIGNAL x : STD_LOGIC; -- calcul intermédiaire a AND b
BEGIN
s2<=s1 OR c;
x<= a AND b;
TRI0 : TRI  PORT MAP (x, cs, s1); -- porte 3 états
END archi;
```


LE BEAUCOUP MEILLEUR DE VHDL - REV. 4
Document libre - Copie Autorisée
J.O.KLEIN - 13 AVRIL 2005 - IUT CACHAN
LIBRARY LIBNAME ;
USE LIBNAME.PACKAGENAME.ALL;
ENTITY ENTITY_NAME IS
PORT (
SIGNAL_NAME : MODE SIGNAL_TYPE ;
SIGNAL_NAME : MODE SIGNAL_TYPE);
END ENTITY_NAME;
ARCHITECTURE ARCHITECTURE_NAME OF ENTITY_NAME IS
DECLARATIONS DE COMPOSANTS
+ DECLARATIONS DES SIGNAUX INTERNES
+ AUTRES DECLARATIONS ...
BEGIN
INSTRUCTIONS CONCURENTES ;
+ PROCESSUS;
+ INSTANCES DE COMPOSANT;
END ARCHITECTURE_NAME ;
LIBRARY IEEE ;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
IN | OUT | INOUT | ...
STD_LOGIC | UNSIGNED (N DOWNTO 0) | ...
COMPONENT ENTITY_NAME
PORT (
SIGNAL_NAME : MODE SIGNAL_TYPE ;
SIGNAL_NAME : MODE SIGNAL_TYPE);
END COMPONENT ;
SIGNAL SIGNAL_NAME : SIGNAL_TYPE ;
INSTRUCTIONS CONCURRENTES
AFFECTATION PERMANENTE
Q <= A;
AFFECTATION CONDITIONNELLE
MUX 2→1
CONDITION
WHEN
ELSE
Q <= A WHEN CONDITION
ELSE B;
AFFECTATION SELECTIVE
MUX N→1
SELECTION
WITH
SELECT
WITH SELECTION SELECT
Q <= A WHEN K1,
B WHEN K2,
...
C WHEN OTHERS;
INSTANCE DE COMPOSANT
I1
ADDER
SYS_A
SYS_B
SYS_Q
I1 : ADDER PORT MAP(
A =>SYS_A,
B =>SYS_B,
C =>SYS_C);
PROCESSUS
P1
Processus à LISTE DE SENSIBILITE
PROCESS
P1 : PROCESS (A,B)
--Déclarations
BEGIN
--Instructions
--Séquentielles
END PROCESS P1;
P2
Processus SYNCHRONE
P2 : PROCESS
--Déclarations
BEGIN
WAIT UNTIL RISING_EDGE(CLOCK);
--Instructions
--Séquentielles
END PROCESS P2;
P3
Processus NON SYNTHETISABLE
UNIQUEMENT dans un TESTBENCH
10ns
P3 : PROCESS
--Déclarations
BEGIN
WAIT FOR 10 ns
--...
WAIT FOR 10 ns
END PROCESS P3;
INSTRUCTIONS SEQUENTIELLES
UNIQUEMENT dans un PROCESS
Affectation de signal (différée)
<= A<=B; A <= B + 1;
Affectation de variable (immédiate)
:= i := j; i := i+1;
CASE
CASE...IS
WHEN...=>...
[WHEN...=>...]
WHEN OTHERS =>...
END CASE;
IF
IF...THEN
[ELSIF...THEN
[ELSE
END IF;
FOR
FOR...IN...TO...LOOP
END LOOP;
WHILE
WHILE...LOOP
END LOOP;