# Électronique Numérique

## Chapitre 6 :

## *VHDL*

*Biblio : (docs web)*

*Mrs LECARDONNEL et LETENNEUR (intro au VHDL)*
*Mr RABASTE (iufm d'Aix-Marseille)*
*Mrs WEBER et MEAUDRE (un outil: VHDL) (iut de Cachan)*

perisse@cict.fr





## VHDL
## Introduction

L'abréviation **VHDL** signifie **V**HSIC **H**ardware **D**escription **L**anguage.
**VHSIC** : **V**ery **H**igh **S**peed **I**ntegrated **C**ircuit.

Ce langage a été écrit dans les années 70 pour réaliser la simulation de circuits électroniques. On l'a ensuite étendu en lui rajoutant des extensions pour permettre la conception (synthèse) de circuits logiques programmables.

**P.L.D.** : **P**rogrammable **L**ogic **D**evice.

Auparavant pour décrire le fonctionnement d'un circuit électronique programmable les techniciens et les ingénieurs utilisaient des langages de bas niveau (ABEL, PALASM, ORCAD/PLD,..) ou plus simplement un outil de saisie de schémas.

Actuellement la densité de fonctions logiques (portes et bascules) intégrée dans les **PLD** est telle (plusieurs milliers de portes voire millions de portes) qu'il n'est plus possible d'utiliser les outils d'hier pour développer les circuits d'aujourd'hui.


perisse@cict.fr

# VHDL
# Introduction

**Les sociétés de développement et les ingénieurs ont voulu s'affranchir des contraintes technologiques des circuits. Ils ont donc créé des langages dits de haut niveau à savoir VHDL et VERILOG. Ces deux langages font abstraction des contraintes technologies des circuits PLDs.**

**Ils permettent au code écrit d'être portable, c'est à dire qu'une description écrite pour un circuit peut être facilement utilisée pour un autre circuit.**
**Il faut avoir à l'esprit que ces langages dits de haut niveau permettent de matérialiser les structures électroniques d'un circuit.**
**En effet les instructions écrites dans ces langages se traduisent par une configuration logique de portes et de bascules qui est intégrée à l'intérieur des circuits PLDs. C'est pour cela que je préfère parler de description VHDL ou VERILOG que de langage.**

**Dans ce cours, je m'intéresserai seulement à VHDL et aux fonctionnalités de base de celui-ci lors des phases de conception ou synthèse (c'est à dire à la conception de PLD).**





# VHDL
# Structure d'une description VHDL simple

Une description **VHDL** est composée de 2 parties **indissociables** à savoir :

-***L'entité*** (***ENTITY***), elle définit les entrées et sorties.

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

***Exemple :*** Un démultiplexeur 2 vers 4.

```
-- démultiplexeur / décodeur                          Commentaires
-- Deux vers quatre

library ieee;                                         Déclaration des
Use ieee.std_logic_1164.all;                          bibliothèques

entity DEMUX2_4 is
port(IN0, IN1: in std_logic;                          Déclaration de
D0, D1, D2, D3: out std_logic);                       l'entité du
end DEMUX2_4;                                         démultiplexeur

architecture DESCRIPTION of DEMUX2_4 is
begin
D0 <= (not(IN1) and not(IN0));
D1 <= (not(IN1) and IN0);                             Déclaration de
D2 <= (IN1 and not(IN0));                             l'architecture du
D3 <= (IN1 and IN0);                                  démultiplexeur
end DESCRIPTION;
```

# VHDL
# Déclaration des bibliothèques

Toute description VHDL utilisée pour la synthèse a besoin de bibliothèques.

L'IEEE les a normalisées et plus particulièrement la bibliothèque IEEE1164.

IEEE : Institut of Electrical and Electronics Engineers.

Elles contiennent les définitions des types de signaux électroniques, des fonctions et sous programmes permettant de réaliser des opérations arithmétiques et logiques,...

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;
Use ieee.std_logic_unsigned.all;

-- cette dernière bibliothèque est souvent utilisée pour l'écriture de compteurs
```

La directive Use permet de sélectionner les bibliothèques à utiliser.





# VHDL
# Déclaration de l'entité

Elle permet de définir le *NOM* de la description VHDL ainsi que les entrées et sorties utilisées, l'instruction qui les définit c'est *port* :

*Syntaxe:*

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

*Exemple :*

```
entity SEQUENCEMENT is
        port (  CLOCK : in std_logic;
                RESET : in std_logic;
                Q : out std_logic_vector(1 downto 0) );
end SEQUENCEMENT;
```

**Remarques :**

- VHDL n'est pas sensible à la « casse », c'est à dire qu'il ne fait pas la distinction entre les majuscules et les minuscules.
- Après la dernière définition de signal de l'instruction *port* il ne faut jamais mettre de point virgule.

# VHDL
# Déclaration des entrées / sorties (I/O)

## L'instruction *port*

***Remarque :*** **On doit définir pour chaque signal : le *NOM_DU_SIGNAL*, le *sens* et le *type***

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

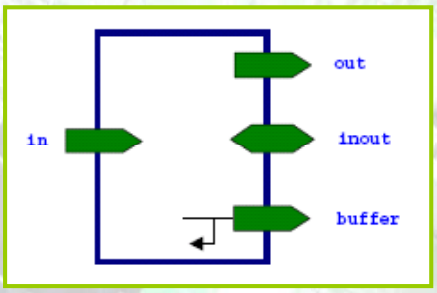
***Exemple:*** *CLOCK: in std_logic;*
*BUS : out std_logic_vector(7 downto 0);*

→ **Le *NOM_DU_SIGNAL*.**

**Il est composé de caractères, le premier caractère doit être une lettre, sa longueur est quelconque, mais elle ne doit pas dépasser une ligne de code.**

→ **Le *SENS* du signal.**

- ***in*** : **pour un signal en entrée.**
- ***out*** : **pour un signal en sortie.**
- ***inout*** : **pour un signal en entrée sortie**
- ***buffer*** : **pour un signal en sortie mais utilisé comme entrée dans la description.**






LINEL
Electronique Numérique

# VHDL
# Déclaration des entrées / sorties (I/O)

→ **Le *TYPE***

**Le *TYPE* utilisé pour les signaux d'entrées / sorties est :**

- **le *std_logic* pour un signal.**
- **le *std_logic_vector* pour un bus composé de plusieurs signaux.**

***Par exemple*** **un bus bidirectionnel de 5 bits s'écrira :**

*LATCH : inout std_logic_vector (4 downto 0) ;*
**Où** ***LATCH(4)*** **correspond au MSB**
**et** ***LATCH(0)*** **correspond au LSB.**

**Les valeurs que peuvent prendre un signal de type *std_logic* sont :**

- ***'0' ou 'L'*** **: pour un niveau bas.**
- ***'1' ou 'H'*** **: pour un niveau haut.**
- ***'Z'*** **: pour état haute impédance.**
- ***'-'*** **: Quelconque, c'est à dire n'importe quelle valeur.**

# VHDL
# Déclaration de l'architecture

L'architecture décrit le fonctionnement souhaité pour un circuit ou une partie du circuit.
En effet le fonctionnement d'un circuit est généralement décrit par plusieurs modules VHDL. Il faut comprendre par module le couple *ENTITE/ARCHITECTURE*. Dans le cas de simples PLDs on trouve souvent un seul module.
L'architecture établit à travers les instructions les relations entre les entrées et les sorties. On peut avoir un fonctionnement purement combinatoire, séquentiel voire les deux séquentiel et combinatoire.

*Exemples :*

```
-- Opérateurs logiques de base
entity PORTES is
port (      A,B :in std_logic;
            Y1,Y2,Y3,Y4,Y5,Y6,Y7:out std_logic);
end PORTES;
architecture DESCRIPTION of PORTES is
begin
            Y1 <= A and B;
            Y2 <= A or B;
            Y3 <= A xor B;
            Y4 <= not A;
            Y5 <= A nand B;
            Y6 <= A nor B;
            Y7 <= not(A xor B);
end DESCRIPTION;
```





# VHDL
# Les instructions concurrentes, logique combinatoire.

Qu'est ce que le mode *« concurrent »* ? Pour une description VHDL toutes les instructions sont évaluées et affectent les signaux de sortie en même temps. L'ordre dans lequel elles sont écrites n'a aucune importance. En effet la description génère des structures électroniques, c'est la grande différence entre une description VHDL et un langage informatique classique.
Dans un système à microprocesseur, les instructions sont exécutées les unes à la suite des autres.
Avec VHDL il faut essayer de penser à la structure qui va être générée par le synthétiseur pour écrire une bonne description, cela n'est pas toujours évident.

Reprenons l'exemple du décodeur 2 vers 4 :
→ l'ordre dans lequel seront écrites les instructions n'a aucune importance.

```
architecture DESCRIPTION_1 of DEMUX2_4 is
begin
D0 <= (not(IN1) and not(IN0)); -- première instruction
D1 <= (not(IN1) and IN0); -- deuxième instruction
D2 <= (IN1 and not(IN0)); -- troisième instruction
D3 <= (IN1 and IN0); -- quatrième instruction
end DESCRIPTION_1;
```

```
architecture DESCRIPTION_2 of DEMUX2_4 is
begin
D1 <= (not(IN1) and IN0); -- deuxième instruction
D2 <= (IN1 and not(IN0)); -- troisième instruction
D0 <= (not(IN1) AND not(IN0)); -- première instruction
D3 <= (IN1 AND IN0); -- quatrième instruction
end DESCRIPTION_2;
```

**Les deux architectures sont équivalentes**

# VHDL
# Les opérateurs

## L'affectation simple : <=

Dans une description VHDL, c'est certainement l'opérateur le plus utilisé. En effet il permet de modifier l'état d'un signal en fonction d'autres signaux et/ou d'autres opérateurs.
Exemple avec des portes logiques : *S1 <= E2 and E1 ;*

Les valeurs numériques que l'on peut affecter à un signal sont les suivantes :

- '1' ou 'H' pour un niveau haut avec un signal de 1 bit.
- '0' ou 'L' pour un niveau bas avec un signal de 1 bit.
- 'Z' pour un état haute impédance avec un signal de 1 bit.
- '-' pour un état quelconque, c'est à dire '0' ou '1'. Cette valeur est très utilisée avec les instructions : *when … else* et *with …. Select …. ;* Pour les signaux composés de plusieurs bits on utilise les guillemets " … " .

Les bases numériques utilisées pour les bus peuvent être :

BINAIRE, exemple : *BUS <= "1001" ;* *-- BUS = 9 en décimal*
HEXA, exemple : *BUS <= X"9" ;* *-- BUS = 9 en décimal*
OCTAL, exemple : *BUS <= O"11" ;* *-- BUS = 9 en décimal*

**Remarque :** La base décimale ne peut pas être utilisée lors de l'affectation de signaux. On peut seulement l'utiliser avec certains opérateurs, comme + et – pour réaliser des compteurs.





# VHDL
# Les opérateurs

## Exemple concernant l'affectation simple : <=

```
Library ieee;
Use ieee.std_logic_1164.all;

entity AFFEC is
        port (  E1,E2 : in std_logic;
                BUS1,BUS2,BUS3 : out std_logic_vector(3 downto 0);
                S1,S2,S3,S4 : out std_logic);
end AFFEC;

architecture DESCRIPTION of AFFEC is
begin
        S1 <= '1';                              -- S1 = 1
        S2 <= '0';                              -- S2 = 0
        S3 <= E1;                               -- S3 = E1
        S4 <= '1' when (E1 ='1') else 'Z';      -- S4 = 1 si E1=1 sinon S4
                                                -- prend la valeur haute impédance
        BUS1 <= "1000";                         -- BUS1 = "1000"
        BUS2 <= E1 & E2 & "10";                 -- BUS2 = E1 & E2 & 10
        BUS3 <= x"A";                           -- valeur en HEXA -> BUS3 = 10(déc)
end DESCRIPTION;
```

# VHDL
# Les opérateurs

## Opérateur de concaténation : &.

**Cet opérateur permet de joindre des signaux entre eux .**

```
-- Soit A et B de type 3 bits et S1 de type 8 bits
-- A = "001" et B ="110"
S1 <= A & B & "01" ;
-- S1 prendra la valeur suivante après cette affectation
-- S1 = "001 110 01"
```



LINEL
Electronique Numérique

# VHDL
# Les opérateurs

## Opérateurs logiques.

| Opérateur VHDL | |
|---|---|
| ET | and |
| NON ET | nand |
| OU | or |
| NON OU | nor |
| OU EXCLUSIF | xor |
| NON OU EXCLUSIF | xnor |
| NON | not |
| DECALAGE A GAUCHE | sll |
| DECALAGE A DROITE | srl |
| ROTATION A GAUCHE | rol |
| ROTATION A DROITE | ror |

**Exemples :**

```
S1 <= A sll 2 ;      -- S1 = A décalé de 2 bits à gauche.
S2 <= A rol 3 ;      -- S2 = A avec une rotation de 3 bits à gauche
S3 <= not (R);       -- S3 = R
```

# VHDL
# Les opérateurs

**Remarque :** Pour réaliser des décalages logiques en synthèse logique, il est préférable d'utiliser les instructions suivantes :

**Décalage à droite :**

```
-- Si A est de type std_logic_vector(7 downto 0)

        S1 <= '0' & A(7 downto 1); -- décalage d'un bit à droite
        S1 <= "000" & A(7 downto 3); -- décalage de trois bits à droite
```

**Décalage à gauche :**

```
-- Si A est de type std_logic_vector(7 downto 0)

        S1 <= A(6 downto 0) & '0'; -- décalage d'un bit à gauche
        S1 <= A(4 downto 0) & "000"; -- décalage de trois bits à gauche
```





# VHDL
# Les opérateurs

## Opérateurs arithmétiques.

| Opérateur VHDL | | |
|---|---|---|
| | ADDITION | + |
| | SOUSTRACTION | - |
| | MULTIPLICATION | * |
| | DIVISION | / |

**Remarque N°1 :** Pour pouvoir utiliser les opérateurs ci-dessus il faut rajouter les bibliothèques suivantes au début du fichier VHDL:

```
Use ieee.numeric_std.all ;
Use ieee.std_logic_arith.all ;
```

**Exemples :**

```
S1 <= A – 3 ;        -- S1 = A – 3
S1 <= S1 + 1 ;       -- On incrémente de 1 le signal S1
```

**Remarque N°2 :** Attention l'utilisation de ces opérateurs avec des signaux comportant un nombre de bits important peut générer de grandes structures électroniques.

**Exemples :**

```
S1 <= A * B ;        -- S1 = A multiplié par B : A et B sont codés sur 4 bits
S2 <= A / B ;        -- S2 = A divisé par B : A et B sont codés sur 4 bits
```

# VHDL
# Les opérateurs

## Opérateurs relationnels.

**Ils permettent de modifier l'état d'un signal ou de signaux suivant le résultat d'un test ou d'une condition. En logique combinatoire ils sont souvent utilisés avec les instructions :**

- ***when … else …***
- ***with …. select ….***

**Opérateur VHDL**

| | |
|---|---|
| Egal | = |
| Non égal | /= |
| Inférieur | < |
| Inférieur ou égal | <= |
| Supérieur | > |
| Supérieur ou égal | >= |





# VHDL
# Les instructions du mode « concurrent »

## Affectation conditionnelle :

**Cette instruction modifie l'état d'un signal suivant le résultat d'une condition logique entre un ou des signaux, valeurs, constantes.**

***SIGNAL <= expression when condition***
***[else expression when condition]***
***[else expression];***

**Remarque :**

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

**Voir exemples ci-après**

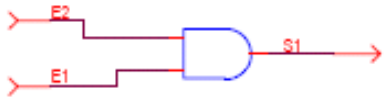
# VHDL
# Les instructions du mode « concurrent »

**Exemple N°1 :**

*-- S1 prend la valeur de E2 quand E1='1' sinon S1 prend la valeur '0'*

```
S1 <= E2 when ( E1= '1') else '0';
```

Schéma correspondant :

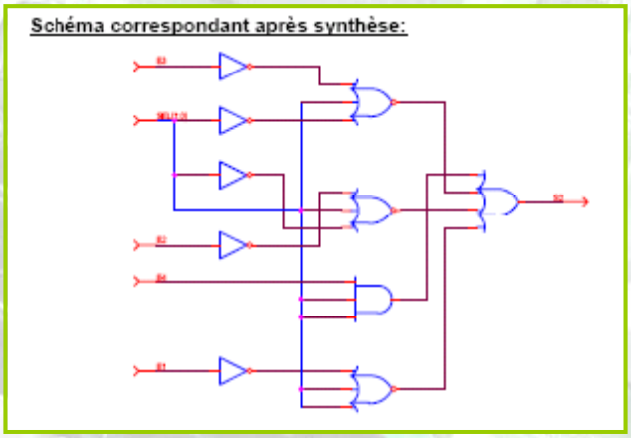



**Exemple N°2 :**

*-- Structure évoluée d'un multiplexeur 4 vers 1*

```
S2 <= E1 when (SEL="00" ) else
E2 when (SEL="01" ) else
E3 when (SEL="10" ) else
E4 when (SEL="11" )
else '0';
```

Schéma correspondant après synthèse:







# VHDL
# Les instructions du mode « concurrent »

## Affectation sélective :

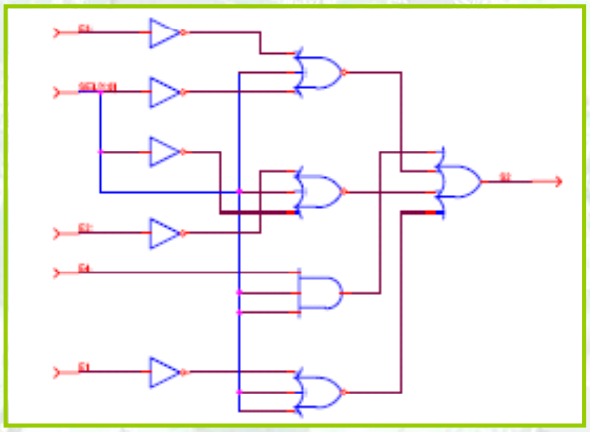
**Cette instruction permet d'affecter différentes valeurs à un signal, selon les valeurs prises par un signal dit de sélection.**

```
with SIGNAL_DE_SELECTION select
SIGNAL <= expression when valeur_de_selection,
          [expression when valeur_de_selection,]
          [expression when others];
```

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

# VHDL
# Les instructions du mode « concurrent »

**Exemple N°1 :** *-- Multiplexeur 4 vers 1*



```
        with SEL select
        S2 <=     E1 when "00",
                  E2 when "01",
                  E3 when "10",
                  E4 when "11",
'                 0' when others;
```

**Remarque:**
***when others*** **est nécessaire car il faut toujours définir les autres cas du signal de sélection pour prendre en compte toutes les valeurs possibles de celui-ci.**
**En conclusion, les descriptions précédentes donnent le même schéma, ce qui est rassurant. L'étude des deux instructions montre toute la puissance du langage VHDL pour décrire un circuit électronique, en effet si on avait été obligé d'écrire les équations avec des opérateurs de base pour chaque sortie, on aurait eu les instructions suivantes :**

```
S2 <= (E1 and not(SEL(1)) and not(SEL(0))) or (E2 and not SEL(1) and(SEL(0)) or (E3 and SEL(1)
and not(SEL(0))) or (E4 and SEL(1) and SEL(0));
```

**L'équation logique ci-dessus donne aussi le même schéma, mais elle est peu compréhensible, c'est pourquoi on préfère des descriptions de plus haut niveau en utilisant les instructions VHDL évoluées.**



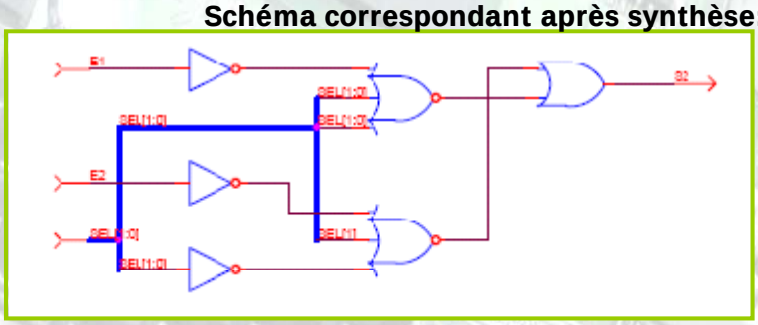


# VHDL
# Les instructions du mode « concurrent »

**Exemple N°2 :** **Affectation sélective avec les autres cas forcés à '0'.**

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;
Use ieee.std_logic_unsigned.all;

entity TABLE1 is
        port (    E1,E2 : in std_logic;
                  SEL : in std_logic_vector(1 downto 0);
                  S2 : out std_logic);
end TABLE1;

architecture DESCRIPTION of TABLE1 is
begin
        with SEL select
        S2 <=     E1 when "00",
                  E2 when "01",
                  '0' when others;
end DESCRIPTION;
```

**Schéma correspondant après synthèse:**



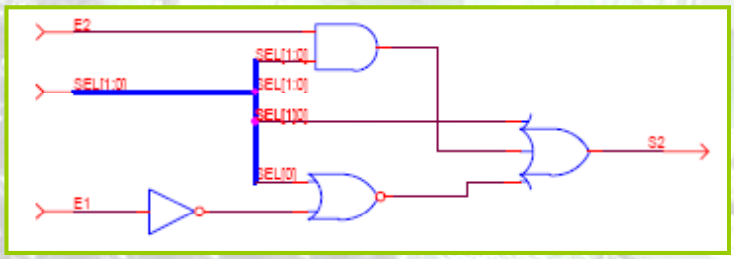
*-- Pour les autres cas de SEL S2*
*-- prendra la valeur 0 logique*

# VHDL
# Les instructions du mode « concurrent »

**Exemple N°3 : Affectation sélective avec les autres cas forcés à un '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;
end DESCRIPTION;
```

**Schéma correspondant après synthèse:**




```
-- Pour les autres cas de SEL S2
-- prendra la valeur 1 logique
```



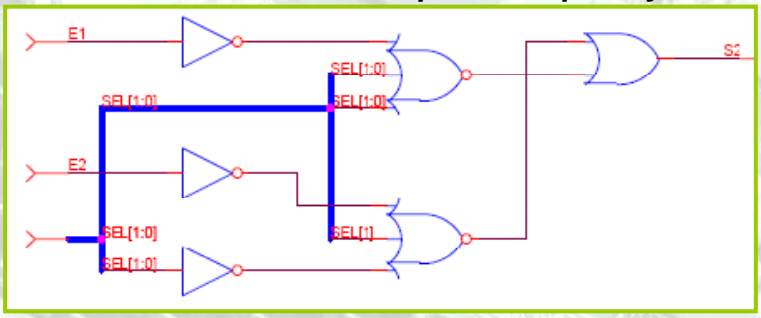


# VHDL
# Les instructions du mode « concurrent »

**Exemple N°4 : Affectation sélective avec les autres cas forcés à une valeur quelconque '-'.**

```
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;
end DESCRIPTION;
```

**Schéma correspondant après synthèse:**




```
-- Pour les autres cas de SEL S2
-- prendra la valeur quelconque
```

# VHDL
# Les instructions du mode séquentiel

## Définition d'un PROCESS.

Un process est une partie de la description d'un circuit dans laquelle les instructions sont exécutées séquentiellement c'est à dire les unes à la suite des autres.
Il permet d'effectuer des opérations sur les signaux en utilisant les instructions standard de la programmation structurée comme dans les systèmes à microprocesseurs.
L'exécution d'un process est déclenchée par un ou des changements d'états de signaux logiques.
Le nom de ces signaux est défini dans la liste de sensibilité lors de la déclaration du process.

```
[Nom_du_process :] process(Liste_de_sensibilité_nom_des_signaux)
Begin
        -- instructions du process
end process [Nom_du_process] ;
```

Remarque: Le nom du process entre crochet est facultatif, mais il peut être très utile pour repérer un process parmi d'autres lors de phases de mise au point ou de simulations.
Règles de fonctionnement d'un process :

1) L'exécution d'un process a lieu à chaque changement d'état d'un signal de la liste de sensibilité.
2) Les instructions du process s'exécutent séquentiellement.
3) Les changements d'état des signaux par les instructions du process sont pris en compte à la fin du process.





# VHDL
# Les instructions du mode séquentiel

## Les deux principales structures utilisées dans un process.

### L'assignation conditionnelle

```
if condition then instructions
        [elsif condition then instructions]
        [else instructions]
end if ;
```

*Exemple:*

```
if (RESET='1') then SORTIE <= "0000";
```

### L'assignation sélective

```
case signal_de_slection is
    when valeur_de_sélection => instructions
   [when others => instructions]
end case;
```

*Exemple:*

```
case SEL is
        when "000" => S1 <= E1;
        when "001" => S1 <= '0';
        when "010" | "011" => S1 <='1';
        -- La barre | permet de réaliser
        -- un ou logique entre les deux
        -- valeurs "010" et "011"
        when others => S1 <= '0';
end case;
```

Remarque: ne pas confondre => (implique) et <= (affecte). Ces deux instructions vont être utilisées dans la suite du polycopié.

# VHDL Les instructions du mode séquentiel

## Exemples n°1 de process : Bascule D synchrone

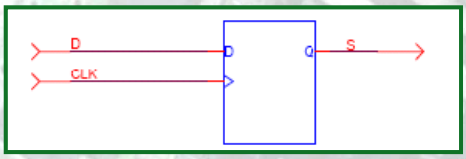
```
Library ieee;
Use ieee.std_logic_1164.all;

entity BASCULED is
port (      D,CLK :     in std_logic;
            S :         out std_logic);
end BASCULED;

architecture DESCRIPTION of BASCULED is
begin
PRO_BASCULED : process (CLK)
begin
            if (CLK'event and CLK ='1') then
                        S <= D;
            end if;
end process PRO_BASCULED;
end DESCRIPTION;
```

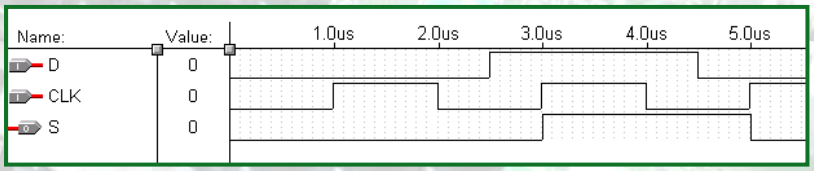
***Commentaires :***

-Seul le signal *CLK* fait partie de la liste de sensibilité. D'après les règles de fonctionnement énoncées précédemment, seul un changement d'état du signal *CLK* va déclencher le process et par conséquent évaluer les instructions de celui-ci.

- L'instruction *if (CLK'event and CLK='1') then* permet de détecter un front montant du signal *CLK*. La détection de front est réalisée par l'attribut *event* appliqué à l'horloge *CLK*. Si on veut un déclenchement sur un front descendant, il faut écrire l'instruction suivante : *if (CLK'event and CLK='0').*
-Les bibliothèques IEEE possèdent deux instructions permettant de détecter les fronts montants *rising_edge(CLK)* ou descendants *falling_edge(CLK)*.

- Si la condition est remplie alors le signal de sortie *S* sera affecté avec la valeur du signal d'entrée *D*.



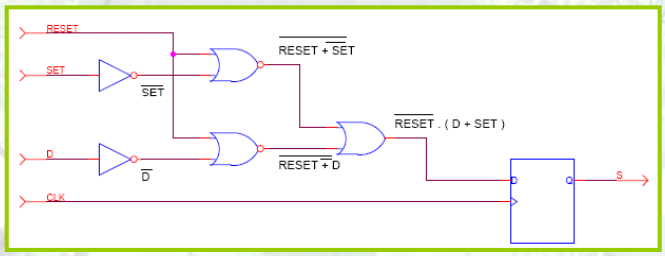


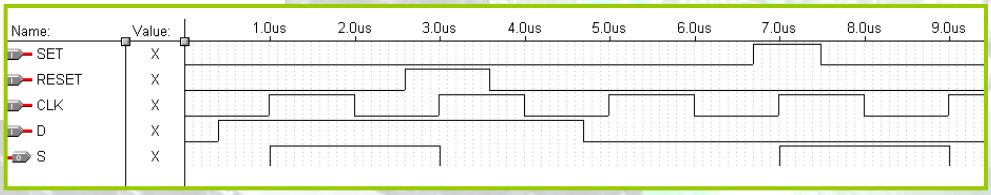

LINEL
Electronique Numérique

# VHDL Les instructions du mode séquentiel

## Exemples n°2 : Bascule D synchrone (Reset et Set synchrone)

```
--bascule D avec Reset et Set Synchrone
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY bascdsrsyn IS
    PORT (
         D, CLK, SET, RESET :   IN STD_LOGIC;
         S:                     OUT STD_LOGIC
         );
END bascdsrsyn ;

ARCHITECTURE archi_bascdsrsyn OF bascdsrsyn IS
BEGIN

    PROCESS (CLK)
    BEGIN
        IF  (CLK'EVENT AND CLK = '1') THEN
            IF      RESET = '1' THEN S <= '0';
            ELSIF   SET = '1'   THEN S <= '1';
            ELSE    S <= D;
            END IF;
        END IF;
    END PROCESS;
END archi_bascdsrsyn;
```



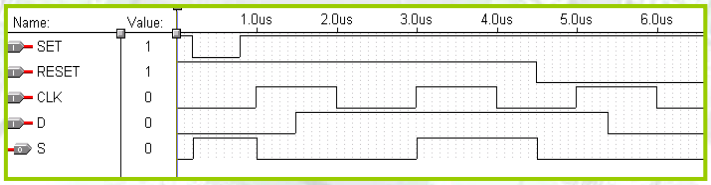
# VHDL Les instructions du mode séquentiel

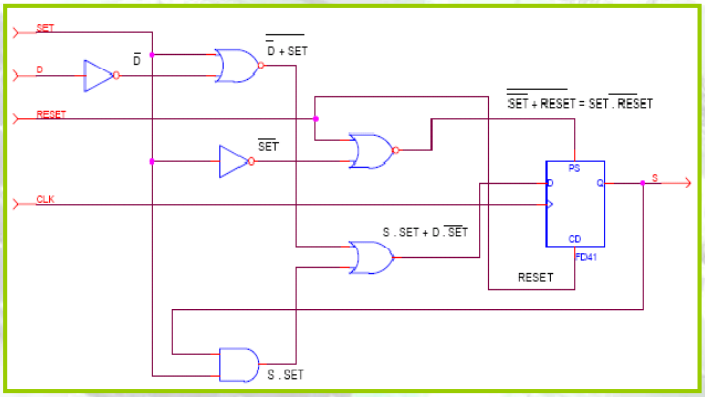
## Exemples n°3 : Bascule D synchrone (Reset et Set Asynchrone)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY bascdsra IS
    PORT (
         D, CLK, SET, RESET : IN STD_LOGIC;
         S:                   OUT STD_LOGIC
         );
END bascdsra ;

ARCHITECTURE archi_bascdsra OF bascdsra IS
SIGNAL     test: STD_LOGIC ;
BEGIN

    PROCESS ( RESET, SET, CLK)
    BEGIN
    IF RESET = '0' THEN test <= '0';
    ELSIF SET = '0' THEN test <= '1';
    ELSIF CLK'EVENT AND CLK = '1' THEN test <= D;
    ELSE test <= test;
    END IF;
    END PROCESS;
S <= test;
END archi_bascdsra;
```

***Commentaire***

**L'entrée *RESET* est prioritaire sur l'entrée *SET* qui est à son tour prioritaire sur le front montant de l'horloge *CLK*. Cette description est asynchrone car les signaux d'entrée *SET* et *RESET* sont mentionnés dans la liste de sensibilité du *process*.**








# VHDL Les instructions du mode séquentiel

## Exemples n°4 : Bascule T (notion de signal interne)

```
Library ieee;
Use ieee.std_logic_1164.all;

entity BASCULET is
port (      D,CLK :     in std_logic;
            S :         out std_logic);
end BASCULET;

architecture DESCRIPTION of BASCULET is
signal S_INTERNE : std_logic;
begin
PRO_BASCULET : process (CLK)
Begin
            if (CLK'event and CLK ='1') then
              if (D ='1') then
              S_INTERNE <= not (S_INTERNE);
              end if;
            end if;
end process PRO_BASCULET;
S <= S_INTERNE;
end DESCRIPTION;
```

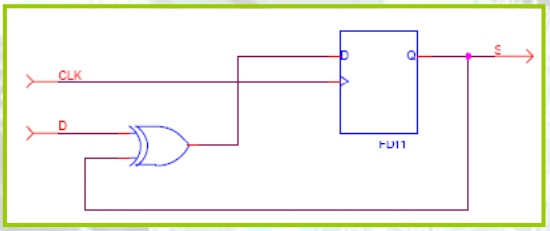
***Commentaires :*** La description fait appel à un signal interne appelé *S_INTERNE*;

Pourquoi faire appel à celui-ci ?

La réponse est que le signal *S* est déclaré comme une sortie dans l'entité, et par conséquent on ne peut pas utiliser une sortie en entrée.

Pour contourner cette difficulté on utilise un signal interne qui peut être à la fois une entrée ou une sortie.

Avec cette façon d'écrire, les signaux de sortie d'une description ne sont jamais utilisés comme des entrées. Cela permet une plus grande portabilité du code.




***Remarque :*** Si on ne déclare pas de signal interne, le synthétiseur renverra certainement une erreur du type :

***Error, cannot read output: s; [use mode buffer or inout].***

Le synthétiseur signale qu'il ne peut pas lire la sortie *S* et par conséquent, celle-ci doit être du type ***buffer*** ou ***inout***.

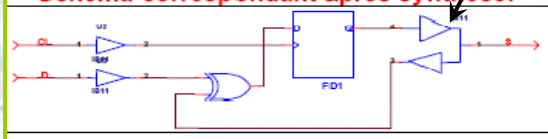
# VHDL Les instructions du mode séquentiel
## Exemples de process :

**Si on souhaite ne pas utiliser de variable interne on peut déclarer le signal S de type *inout* ou *buffer*.**

**Avec S de type *inout***

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;
Use ieee.std_logic_unsigned.all;

entity BASCULET is
        port (
                D,CLK   : in std_logic;
                S       : inout std_logic);
end BASCULET;

architecture DESCRIPTION of BASCULET is
begin
        PRO_BASCULET : process (CLK)
        Begin
                if (CLK'event and CLK='1') then
                        if (D='1') then
                                S <= not (S);
                        end if;
                end if;
        end process PRO_BASCULET;
end DESCRIPTION;
```

**Schéma correspondant après synthèse:**



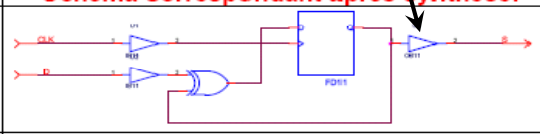
**Commentaires :**

On peut constater que S est bien du type inout.

**Avec S de type *buffer***

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;
Use ieee.std_logic_unsigned.all;

entity BASCULET is
        port (
                D,CLK   : in std_logic;
                S       : buffer std_logic);
end BASCULET;

architecture DESCRIPTION of BASCULET is
begin
        PRO_BASCULET : process (CLK)
        Begin
                if (CLK'event and CLK='1') then
                        if (D='1') then
                                S <= not (S);
                        end if;
                end if;
        end process PRO_BASCULET;
end DESCRIPTION;
```

**Schéma correspondant après synthèse:**



**Commentaires :**

On peut constater que S est bien du type buffer.

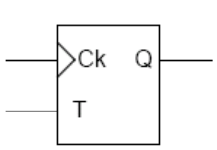


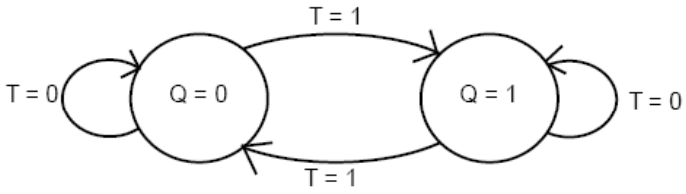


# VHDL Les instructions du mode séquentiel
## Exemples n°4 : Bascule T synchrone

**Rappel : Bascule T (T pour Toggle, c'est à dire bascule)**

**- Si T = "actif" changer d'état à la prochaine transition de l'horloge,**
**- Si T = « 0 » conserver l'état initial.**




**Donner le programme VHDL d'une bascule T :**

**- sous forme comportementale classique (assignation conditionnelle)**

```
if condition then instructions
[elsif condition then instructions]
[else instructions]
end if ;
```

**- en transcrivant le diagramme d'état (assignation sélective)**

```
case signal_de_slection is
        when valeur_de_sélection => instructions
        [when others => instructions]
end case;
```

# VHDL Les instructions du mode séquentiel

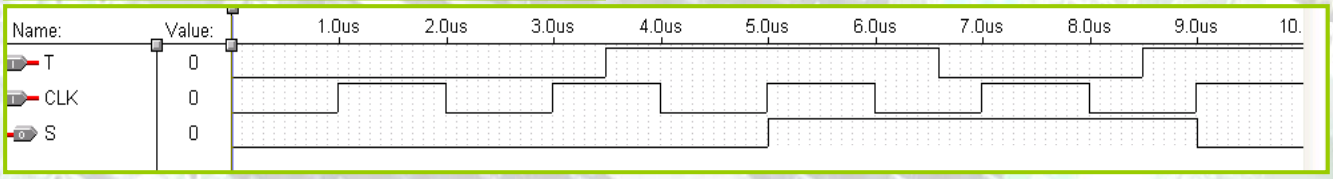
## Exemples n°4 : Bascule T synchrone

```
-- bascule T synchrone
Library ieee;
Use ieee.std_logic_1164.all;

entity bascT is
port (  T,CLK :     in std_logic;
        S :         out std_logic);
end bascT;

architecture arch_bascT of bascT is
signal S_int : std_logic;
begin
PRO_BASCULET : process (CLK)
begin
    if (CLK'event and CLK ='1') then
        if T='1' then   S_int <= not(S_int);
        else S_int<=S_int;
        end if;
    end if;
end process PRO_BASCULET;
S<=S_int;
end arch_bascT;
```

**bascule T :**

**(assignation conditionnelle)**

*if condition then instructions*
*[elsif condition then instructions]*
*[else instructions]*
*end if ;*








# VHDL Les instructions du mode séquentiel

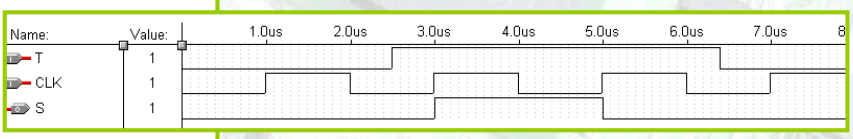
## Exemples n°4 : Bascule T synchrone

```
-- BASCULE T
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

ENTITY basctcase IS
    PORT (  T,CLK   :   IN std_logic;
            S       :   OUT STD_LOGIC);
END basctcase;

ARCHITECTURE archi_basctcase OF basctcase IS
SIGNAL     etat: STD_LOGIC;
BEGIN
    PROCESS
    BEGIN
    WAIT UNTIL CLK = '1';
        CASE etat IS
            WHEN '0' => IF T = '1'  THEN etat <= '1';
                                    ELSE etat <= '0';
                        END IF;
            WHEN '1' => IF T ='1'   THEN etat <= '0';
                                    ELSE etat <= '1';
                                    END IF;
            WHEN OTHERS =>  etat <= '0';
        END CASE;
    END PROCESS;
S <= '1' WHEN etat ='1' ELSE '0';-- Equation des sorties
END archi_basctcase;
```

**bascule T :**

**(assignation sélective)**

*case signal_de_slection is*
*when valeur_de_sélection => instructions*
*[when others => instructions]*
*end case;*



# VHDL Les instructions du mode séquentiel
# Les compteurs :

## Compteur simple :

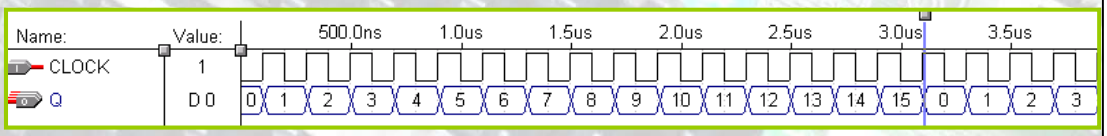
```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_unsigned.all;

entity CMP4BITS is
PORT (      CLOCK : in std_logic;
            Q : inout std_logic_vector(3 downto 0));
end CMP4BITS;

architecture DESCRIPTION of CMP4BITS is
begin
process (CLOCK)
            begin
            if (CLOCK ='1' and CLOCK'event) then
            Q <= Q + 1;
            end if;
end process;
end DESCRIPTION;
```

***Commentaires :***

**Le déclenchement du *process* se fera sur un changement d'état du signal *CLOCK*, l'incrémentation de la sortie *Q* se fera sur le front montant de l'horloge *CLOCK*.**

**- L'incrémentation du compteur est réalisée par l'opérateur + associé à la valeur 1. Cela est logique, mais elle l'est beaucoup moins pour les PLDs et les synthétiseurs, pourquoi ?**

**La réponse est que les entrées et sorties ainsi que les signaux sont déclarés de type *std_logic* ou *std_logic_vector*, et par conséquent on ne peut pas leur associer de valeur entière décimale.**






LINEL
Electronique Numérique

# VHDL Les instructions du mode séquentiel
# Les compteurs :

***Commentaires (suite):***

**- Un signal peut prendre comme valeur les états '1' ou '0' et un bus n'importe quelle valeur, du moment qu'elle est écrite entre deux guillemets "1010" ou X"A" ou o"12", mais pas une valeur comme par exemple 1,2,3,4. Ces valeurs décimales sont interprétées par le synthétiseur comme des valeurs entières (integer), on ne peut pas par défaut additionner un nombre entier 1 avec un bus de type électronique (*st_logic_vector*), c'est pour cela que l'on rajoute dans la partie déclaration des bibliothèques les lignes :**

```
Use ieee.numeric_std.all;
Use ieee.std_logic_unsigned.all;
```

**Ces deux bibliothèques ont des fonctions de conversions de types et elles permettent d'associer un entier avec des signaux électroniques. Elles permettent d'écrire facilement des compteurs, décompteurs, additionneurs, soustracteurs, .....**

**Remarque : Il ne faut pas oublier de les rajouter dans les descriptions.**

**– Le signal *Q* est déclaré dans l'entité de type *inout*, cela est logique car il est utilisé à la fois comme entrée et comme sortie pour permettre l'incrémentation du compteur. Ce type d'écriture est peu utilisé car elle ne permet pas au code d'être portable, on préféra utiliser un signal interne, celui-ci peut être à la fois une entrée et une sortie.**

# VHDL
# Exercice sur les Compteurs :

## *Quelle différence entre les deux descriptions ?*

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_unsigned.all;

entity CMP3BITS is
PORT (      CLOCK, RESET : in std_logic;
            Q : out std_logic_vector(2 downto 0));
end CMP3BITS;

architecture DESCRIPTION of CMP3BITS is
signal CMP: std_logic_vector (2 downto 0);
begin
 process (RESET,CLOCK)
 begin
    if RESET ='1' then      CMP <= "000";
    elsif (CLOCK ='1' and CLOCK'event) then
                            CMP <= CMP + 1;
    end if;
 end process;
Q <= CMP;
end DESCRIPTION;
```

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_unsigned.all;

entity CMP3BITS is
PORT (      CLOCK, RESET : in std_logic;
            Q :  out std_logic_vector (2 downto 0));
end CMP3BITS;

architecture DESCRIPTION of CMP3BITS is
signal CMP: std_logic_vector (2 downto 0);
begin
process (CLOCK)
begin
  if (CLOCK ='1' and CLOCK'event) then
    if RESET ='1'  then   CMP <= "000";
                  else   CMP <= CMP + 1;
    end if;
  end if;
end process;
Q <= CMP;
end DESCRIPTION;
```





# VHDL Les instructions du mode séquentiel
# Les Compteurs :

**Compteur 3 bits avec <u>remise à zéro asynchrone</u>**

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_unsigned.all;
entity CMP3BITS is
PORT (      CLOCK, RESET :  in std_logic;
            Q :  out std_logic_vector(2 downto 0));
end CMP3BI
architecture DESCRIPTION of CMP3BITS is
signal CMP: std_logic_vector (2 downto 0);
begin
process (RESET,CLOCK)
begin
   if RESET ='1' then       CMP <= "000";
  elsif (CLOCK ='1' and CLOCK'event) then
                           CMP <= CMP + 1;
  end if;
end process;
Q <= CMP;
end DESCRIPTION;
```

**Compteur 3 bits avec <u>remise à zéro synchrone</u>**

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_unsigned.all;
entity CMP3BITS is
PORT (      CLOCK, RESET : in std_logic;
            Q : out std_logic_vector (2 downto 0));
end CMP3BITS;
architecture DESCRIPTION of CMP3BITS is
signal CMP: std_logic_vector (2 downto 0);
begin
process (CLOCK)
begin
  if (CLOCK ='1' and CLOCK'event) then
    if RESET ='1'  then   CMP <= "000";
                  else   CMP <= CMP + 1;
    end if;
  end if;
end process;
Q <= CMP;
end DESCRIPTION;
```

***<u>différence entre les deux descriptions</u> :***

***Dans la deuxième le signal RESET n'est plus dans la liste de sensibilité, par conséquence le process ne sera déclenché que par le signal CLOCK. La remise à zéro ne se fera que si un front montant sur CLOCK a lieu.***

# VHDL séquentiel Compt/Décompt à entrée de préchargement :

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_unsigned.all;

entity CMP4BITSLUD is
PORT (      RESET, CLOCK, LOAD, UP:        in std_logic;
            DATA :                         in std_logic_vector (3 downto 0);
            Q :                            out std_logic_vector (3 downto 0));
end CMP4BITSLUD;

architecture DESCRIPTION of CMP4BITSLUD is
signal CMP: std_logic_vector (3 downto 0);
begin
process (RESET,CLOCK)
begin
  if RESET ='1' then CMP <= "0000"; -- Remise à zero asynchrone du compteur
  elsif (CLOCK ='1' and CLOCK'event) then
      if (LOAD ='1') then CMP <= DATA; -- Préchargement synchrone
      else
          if (UP ='1')      then CMP <= CMP + 1; -- Incrémentation synchrone
                            else CMP <= CMP – 1; -- Décrémentattion synchrone
          end if;
      end if;
  end if;
end process;
Q <= CMP;
end DESCRIPTION;
```

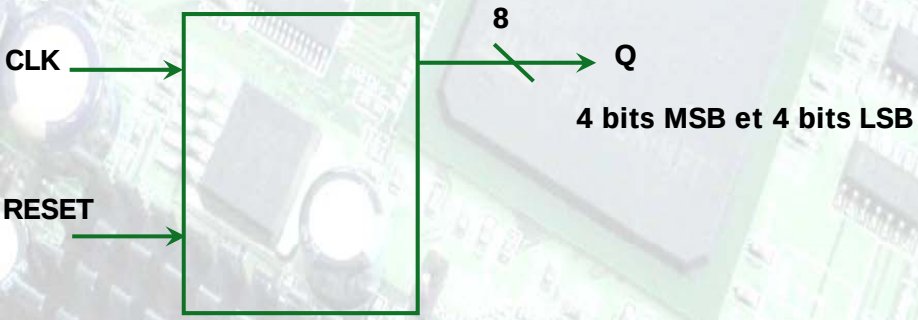
**Remarque :** La mise à zéro des sorties du compteur passe par l'instruction : `CMP <= "0000";` une autre façon d'écrire cette instruction est : `CMP <= (others => '0') ;`
Cette dernière est très utilisée dans les descriptions car elle permet de s'affranchir de la taille du bus. En effet `others=>'0'` correspond à mettre tous les bits du bus à zéro quelque soit le nombre de bits du bus. De la même façon on peut écrire `others=>'1'` pour mettre tous les bits à un.





# VHDL Les instructions du mode séquentiel EXERCICE

***Cahier des charges :***
***Réaliser en VHDL un compteur BCD sur 2 digits***

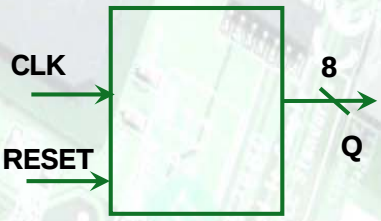


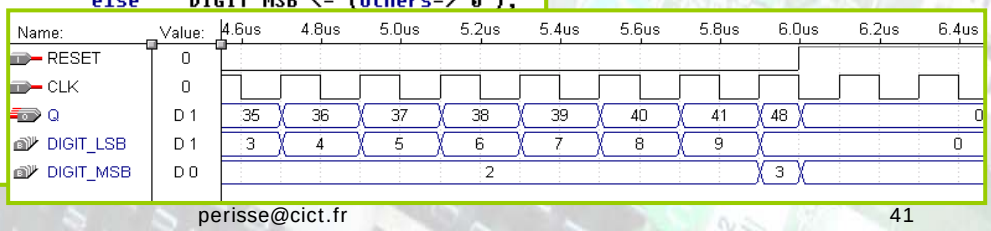

**4 bits MSB et 4 bits LSB**

# VHDL Les instructions du mode séquentiel EXERCICE

```
-- Compteur BCD 2 digits
LIBRARY ieee;
USE ieee.std_logic_1164.all;
Use ieee.std_logic_unsigned.all;

entity cmpbcd2digits is
PORT (  RESET, CLK  :   in std_logic;
        Q           :   out std_logic_vector (7 downto 0));
end cmpbcd2digits;

architecture arch_cmpbcd2digits of cmpbcd2digits is
signal DIGIT_MSB, DIGIT_LSB: std_logic_vector (3 downto 0);
begin
    process (RESET,CLK)
    begin
    if RESET ='1' then  DIGIT_LSB <= (others=>'0');
                        DIGIT_MSB <= (others=>'0');
    elsif (CLK ='1' and CLK'event) then
                if  DIGIT_LSB < 9   then    DIGIT_LSB <= DIGIT_LSB + 1;
                                    else    DIGIT_LSB <= (others=>'0');
                if  DIGIT_MSB < 9   then    DIGIT_MSB <= DIGIT_MSB + 1 ;
                                    else    DIGIT_MSB <= (others=>'0');
                end if;
                end if;
    end if;
    end process;
Q <= DIGIT_MSB & DIGIT_LSB;
end arch_cmpbcd2digits;
```

***Cahier des charges : Réaliser en VHDL un compteur BCD sur 2 digits***








# Syntaxe résumée du langage VHDL.

```
-- Nom des auteurs
-- Description du rôle du fichier VHDL
-- Circuit logique programmable utilisé
library ieee ;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;
-- librairie additionnelle pour les opérations arithmétiques
entity NOMENTITE is
-- Description des signaux d'entrée et de sortie
port (      ENTREE : in std_logic; -- signal simple
            SORTIE : out std_logic; -- signal simple
            ENTREESORTIE : inout std_logic; -- signal simple
            BUFFER : buffer std_logic; -- signal simple
            BUS : in std_logic_vector (3 downto 0); -- signaux regroupés en un bus
            A,B : in std_logic_vector (3 downto 0); -- 2 bus 4 bits en entrée
            C,D,E,F : out std_logic -- 4 signaux de même type PAS DE POINT VIRGULE !
            );
-- En plus des lignes précédentes, il est nécessaire de spécifier le brochage
-- retenu pour le circuit utilisé.
-- La syntaxe suivante est propre à OrCad Express pour un SPLD
-- La définition du brochage doit respecter les possibilités du circuit :
-- IN   CLOCK IO   OUT
end NOMENTITE;
```

***Commentaires relatifs au fichier VHDL***

***Définition des librairies utilisées***

***Description de l'entité : interface externe. NOMENTITE permet d'identifier l'entité***

***Début de la description des signaux***

***Fin de la description des signaux***

***Fin de l'entité : NOMENTITE correspond au nom donné dans la déclaration***

# Syntaxe résumée du langage VHDL.

```
architecture NOMARCHITECTURE of NOMENTITE is
signal NOMAUX : STD_LOGIC_VECTOR (3 downto 0);
Begin                          -- partie Combinatoire
NOMBUS <= NOMAUX ; -- affectation du signal auxiliaire au bus de sortie
SORTIE <= ENTREE0 when ( ENTREECOMMANDE = "00") else
ENTREE1 when( ENTREECOMMANDE = "01") else
ENTREE2 when( ENTREECOMMANDE = "10") else ENTREE3 ;
with ENTREECOMMANDE select -- signal de sélection
SORTIE <= ENTREE0 when "00", -- quand EntreeCommande = "00", Sortie vaut Entree0
ENTREE1 when "01", -- quand EntreeCommande = "01", Sortie vaut Entree1
ENTREE2 when "10", -- quand EntreeCommande = "10", Sortie vaut Entree2
ENTREE3 when others; -- Sortie vaut Entree3 dans les autres cas
                               -- partie Séquentielle
process (SIGNAL1, SIGNAL2,... )
begin
if SIGNAL1'event and SIGNAL1 = '1' -- détection d'un front montant sur Signal1
then if NOMAUX >= x"9" -- test d'inégalité
then NOMAUX <= x"0"; -- affectation simple
else NOMAUX <= NOMAUX + 1; -- incrémentation et affectation (autorisé
end if; -- seulement avec la librairie ieee.std_logic_unsigned.all)
end if;
case ENTREECOMMANDE is -- signal de sélection
when "00" => SORTIE <= ENTREE0; --quand EntreeCommande = "00", Sortie vaut Entree0
when "01" => SORTIE <= ENTREE1; --quand EntreeCommande = "01", Sortie vaut Entree1
whenN others => SORTIE <= ENTREE3; --dans les autres cas, Sortie vaut Entree3
end case;
end process ;
end NOMARCHITECTURE;
```

Déclaration facultative de signaux auxiliaires

Description combinatoire : mode « Concurrent »

Affectation simple

Affectation conditionnelle

Affectation sélective

Description séquentielle : PROCESS suivi de la liste des signaux pouvant déclencher le process

Assignation Conditionnelle : if ... then ... else...

Assignation Sélective : case ... is ... when...

Fin du Process

Fin de l'Architecture : NOMARCHITECTURE



LINEL
Electronique Numérique

# Syntaxe résumée du langage VHDL.

```
-- REMARQUES

-- « Valeurs Numériques »
-- '1' : binaire pour un bit
-- "11" : binaire pour un bus
-- x"11" : heXadécimal pour un bus
-- 11 : décimal (ne peut pas s'appliquer à tous les types de signaux)
-- « Tests »
-- = égal /= différent < inférieur > supérieur
-- >= supérieur ou égal <= inférieur ou égal
```

# VHDL séquentiel : Les erreurs classiques avec l'utilisation de *process*

## Exemple : compteur avec retenue (fonctionnement incorrect).

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;
Use ieee.std_logic_unsigned.all;

entity CMP4BITSRET is
PORT (      RESET, CLOCK :        in std_logic;
            RET :                 out std_logic;
            Q :                   out std_logic_vector (3 downto 0));
end CMP4BITSRET;

architecture DESCRIPTION of CMP4BITSRET is
signal CMP: std_logic_vector (3 downto 0);
begin
process (RESET,CLOCK)
begin
  if RESET ='1' then        CMP <= "0000";
  elsif (CLOCK ='1' and CLOCK'event) then CMP <= CMP + 1;
        if (CMP = "1111") then RET <= '1';
       else RET <= '0';
       end if;
  end if;
end process;
Q <= CMP;
end DESCRIPTION;
```

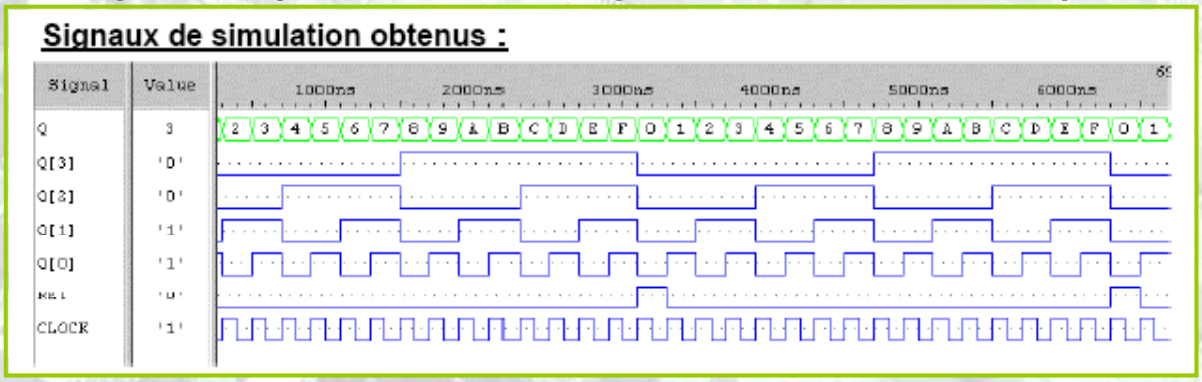


LINEL
Electronique Numérique

# VHDL séquentiel : Les erreurs classiques avec l'utilisation de *process*

## Exemple : compteur avec retenue (fonctionnement incorrect).

**Signaux de simulation obtenus :**




***Les résultats de simulation appellent quelques commentaires :***

On s'aperçoit que le signal de retenue ***RET*** passe au niveau logique haut quand la valeur du compteur vaut ***0***, pourquoi ? → Les valeurs de signaux à l'intérieur d'un ***process*** ne sont mis à jour qu'à la fin de celui-ci.

Dans notre cas, prenons l'état où ***CMP=14*** , au coup d'horloge suivant on incrémente le compteur ***CMP***, mais la nouvelle valeur ne sera affectée à ***CMP*** qu'à la fin du ***process***, donc quand le test pour valider le signal de retenue est effectué, la valeur de ***CMP*** est égale à ***14***, et celui-ci n'est pas valide.

Au coup d'horloge suivant ***CMP=15*** et ***CMP*** est incrémenté donc prendra la valeur ***0*** à la fin du ***process***., mais la condition ***CMP= "1111"*** sera valide et le signal de retenue ***RET*** passera au niveau logique un.

# VHDL séquentiel : Les erreurs classiques avec l'utilisation de *process*

**Comment faire pour pallier à ce problème ? deux solutions :**

**1) Il faut anticiper d'un coup d'horloge, on valide la retenue quand la valeur du compteur vaut *14*, c'est à dire *n-1*.**

```
Library ieee;
Use ...

entity CMP4BITSRET is
PORT (      RESET, CLOCK : in std_logic;
            RET : out std_logic;
            Q : out std_logic_vector (3 downto 0));
end CMP4BITSRET;

architecture DESCRIPTION of CMP4BITSRET is
signal CMP: std_logic_vector (3 downto 0);
begin
            process (RESET,CLOCK)
            begin
              if RESET ='1' then CMP <= "0000";
              elsif (CLOCK ='1' and CLOCK'event) then CMP <= CMP + 1;
                        if (CMP = "1110") then RET <= '1'; else RET <= '0';
-- La retenue passera à un quand CMP = 14 décimal
                        end if;
              end if;
            end process;
Q <= CMP;
end DESCRIPTION;
```

**Remarque :** Dans ce cas la validation de la retenue s'effectue de façon synchrone car elle est dans le *process*, mais la description est peu lisible.





# VHDL séquentiel : Les erreurs classiques avec l'utilisation de *process*

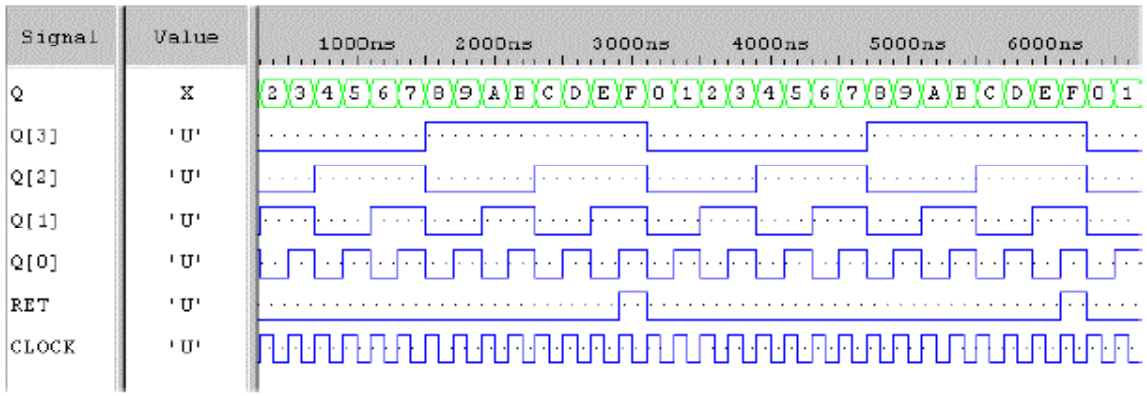
**2) Le test de validation de la retenue est effectuée en dehors du process.**

```
Library ieee;
Use ...

entity CMP4BITSRET is
PORT (      RESET, CLOCK :      in std_logic;
            RET :               out std_logic;
            Q :                 out std_logic_vector (3 downto 0));
end CMP4BITSRET;
architecture DESCRIPTION of CMP4BITSRET is
signal CMP: std_logic_vector (3 downto 0);
begin
            process (RESET,CLOCK)
            begin
            if RESET='1' then CMP <= "0000";
            elsif (CLOCK ='1' and CLOCK'event) then CMP <= CMP + 1;
            end if;
            end process;
Q <= CMP;
-- Validation de la retenue
RET <= '1' when (CMP = "1111") else '0';
end DESCRIPTION;
```

**Remarque :** Dans ce cas la validation de la retenue s'effectue de façon asynchrone car elle est en dehors du *process*, mais la description est lisible.

# VHDL séquentiel : Les erreurs classiques avec l'utilisation de *process*

**Signaux de simulation obtenus :**





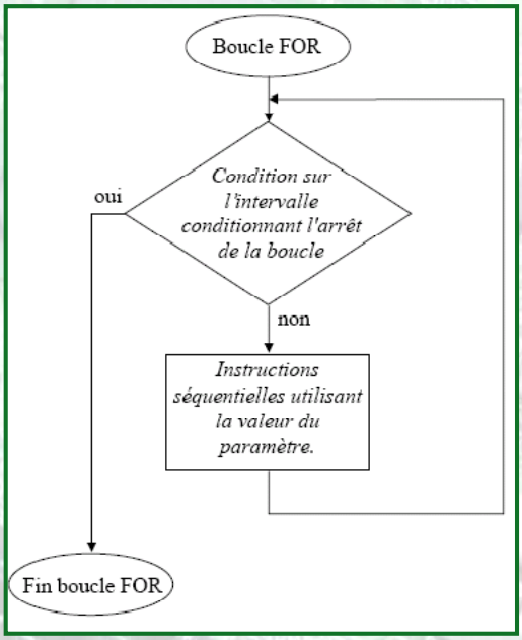



# VHDL
# Boucle FOR LOOP et variable de boucle

**BOUCLE FOR**

**FOR** *paramètre* **IN** *intervalle* **LOOP**
*instructions séquentielles*;
**END LOOP;**

**POUR** *le paramètre* **COMPRIS**
*dans l'intervalle* **EXECUTER**
*instructions séquentielles*;
**FIN DE LA BOUCLE;**




***Exemple de boucle FOR LOOP :***

```
FOR i IN 0 to 3 LOOP
        IF (A = i) THEN
        S <= B;
        END IF;
END LOOP;
```

Exécuter, pour i = 0, i = 1, i = 2 puis i = 3
les instructions suivantes :

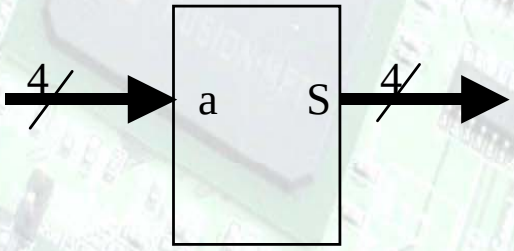
```
        SI (A = i) THEN
        S <= B;
        END IF;
FIN DE LA BOUCLE;
```

# VHDL
# Boucle FOR LOOP et variable de boucle

***Exemple 1: codage de données:***

***Donner une description VHDL afin de satisfaire le cahier des charges suivant :***

```
--Codage de données Si=f(ai)
-- s0 = a0
-- pour i>0     Si  =  /ai
```




***Remarque :***

***On peut effectuer une itération via une instruction de boucle et un compteur de boucle. La variable de boucle est purement virtuelle, elle ne correspond à rien de physique dans le futur circuit et ne doit pas être déclarée.***



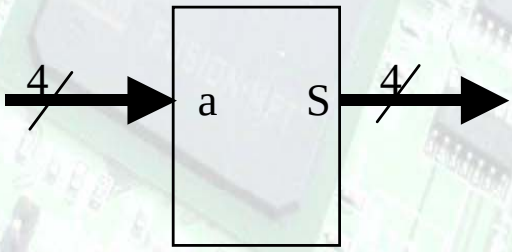


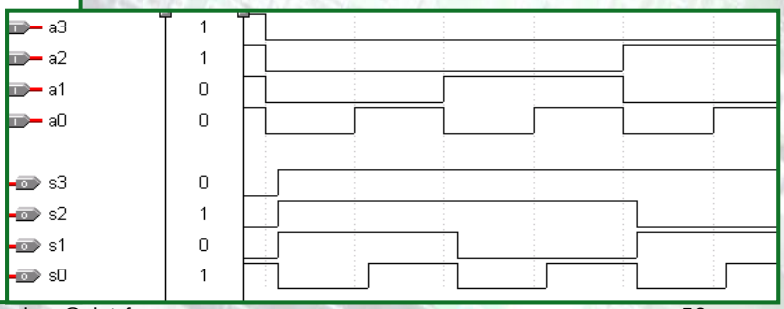
# VHDL
# Boucle FOR LOOP et variable de boucle

***Exemple 1 : Codage de données:***

***Le cahier des charges est le suivant :***

```
--Codage de données Si=f(ai)
-- s0 = a0
-- pour i>0     Si  =  /ai

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY codage IS
    PORT (  a : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
            s : OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END codage;

ARCHITECTURE archi_codage OF codage IS
BEGIN
    PROCESS
    BEGIN
        s(0) <= a(0);
        FOR i IN 1 TO 3 LOOP
        S(i)<=  NOT a(i) ;
        END LOOP;
    END PROCESS;
END archi_codage;
```



# VHDL
# Boucle FOR LOOP et variable de boucle

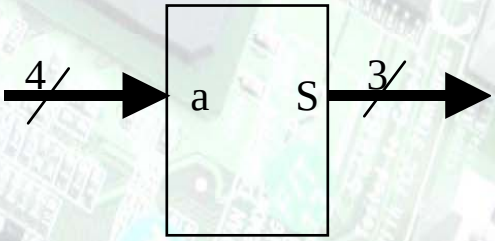
***Remarque :***
***On peut utiliser une variable (en générale entière), pour effectuer des calculs intermédiaires. Cette variable ne correspond à rien de physique. Elle est utilisée par le compilateur pour synthétiser la logique de l'application.***

***Exemple 2 : un autre codage de données:***

***Donner une description VHDL afin de satisfaire le cahier des charges suivant :***

***Le nombre S en sortie indique, en binaire, le nombre de signaux d'entrée à 1.***

***Si a = 1001 alors S = 010***
***Si a = 1100 alors S = 010***
***Si a = 1111 alors S = 100***





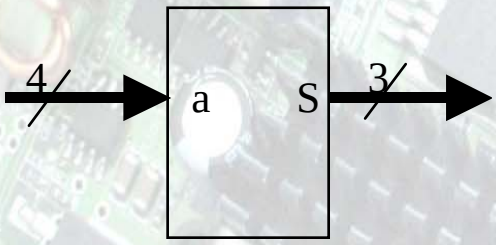


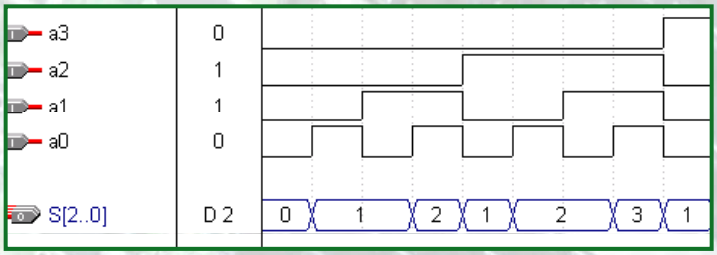

# VHDL
# Boucle FOR LOOP et variable de boucle

***Exemple 2 : un autre codage de données :***

***Le cahier des charges est le suivant : le nombre S en sortie indique, en binaire, le nombre de signaux d'entrée à 1.***

***Si a = 1001 alors S = 010***
***Si a = 1100 alors S = 010***
***Si a = 1111 alors S = 100***




```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY codagebis IS
    PORT (  a : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
            S : OUT INTEGER RANGE 0 TO 4);
END codagebis;

ARCHITECTURE archi_codagebis OF codagebis IS
BEGIN
PROCESS (a)
VARIABLE resultat : INTEGER;
BEGIN
resultat := 0;
    FOR i IN 0 TO 3 LOOP
            IF (a (i) = '1') THEN
            resultat := resultat + 1;
            END IF;
    END LOOP;
s <= resultat;
END PROCESS;
END archi_codagebis;
```

# VHDL
# Boucle FOR LOOP et variable de boucle

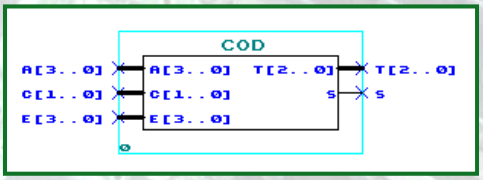
***Exemple 3 :***

***-Cet exemple pourrait aussi être décrit via une table de vérité grâce à l'instruction CASE.***

***-On peut ici préciser la notion de PROCESS: le compilateur prend en compte la totalité du PROCESS, et génère la logique correspondante.***

***-***

***On peut faire cohabiter plusieurs PROCESS dans un même programme VHDL. Ils deviennent « concurrents »: les logiques correspondantes sont générées indépendamment l'une de l'autre***



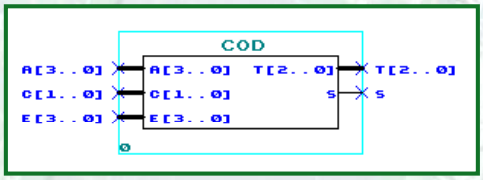

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY cod IS
    PORT (  a : IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
            t : OUT INTEGER RANGE 0 TO 4;
            c : IN  INTEGER RANGE 0 TO 3;
            e : IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
            s : OUT STD_LOGIC);
END cod;
ARCHITECTURE arch_cod OF cod IS
BEGIN
    PROCESS (a)
    VARIABLE resultat : INTEGER;
    BEGIN
        resultat := 0;
        FOR i IN 0 TO 3 LOOP
            IF (a(i) ='1') THEN resultat:=resultat + 1;
            END IF;
        END LOOP;
        t <= resultat;
    END PROCESS;
    PROCESS(c)
    BEGIN
        CASE c IS   WHEN 0 =>   s<= e(0);
                    WHEN 1 ->   s<= e(1);
                    WHEN 2 =>   s<= e(2);
                    WHEN 3 =>   s<= e(3);
        END CASE;
    END PROCESS;
END arch_cod;
```

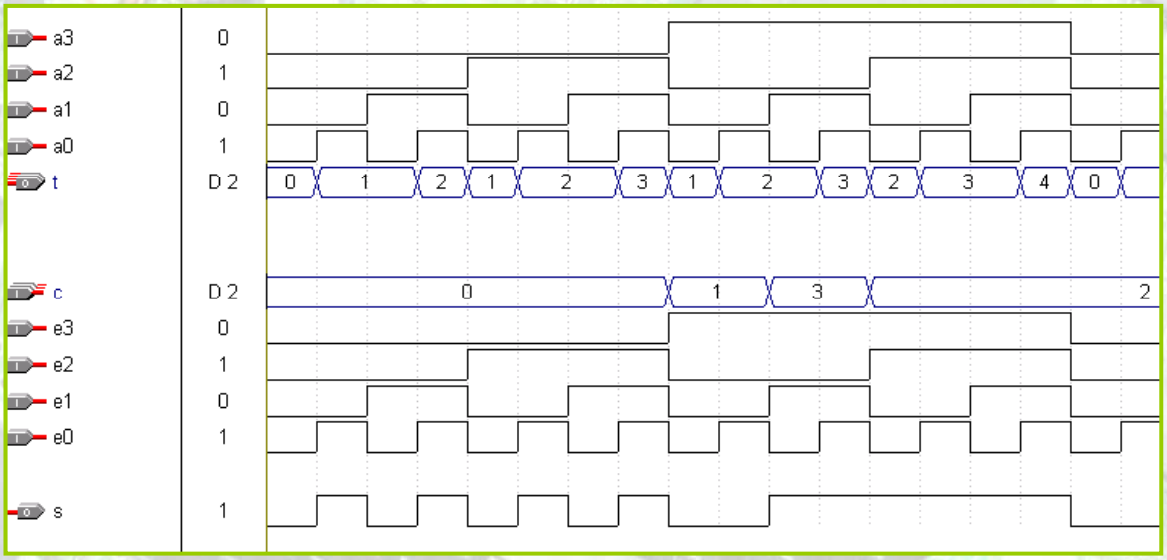




# VHDL
# Boucle FOR LOOP et variable de boucle

***Exemple (suite)***



# VHDL
## Exercice : un diviseur de fréquence

***Le nombre d'états d'un système n'est pas lié au nombre d'états possibles des sorties.***

***Considérons par exemple un diviseur de fréquence fournissant une impulsion S en sortie toutes les quatre impulsions du signal d'horloge H.***

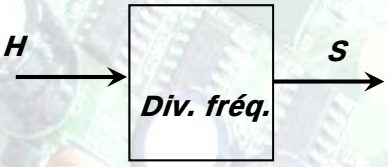
***Pour synthétiser un tel diviseur, il ne suffit évidemment pas de prendre en compte les deux états possibles de la sortie, mais tous les états intermédiaires du système.***


***Donner le diagramme d'état de ce diviseur.***
***En déduire le code VHDL.***

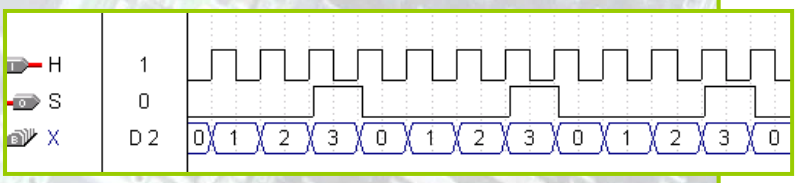


LINEL
Electronique Numérique

# VHDL
## Exercice : un diviseur de fréquence




***Donner le diagramme d'état de ce diviseur.***
***En déduire le code VHDL.***



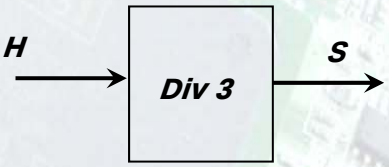

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY div4 IS
    PORT (
         H :    IN  STD_LOGIC;
         S:     OUT STD_LOGIC
         );
END div4;

ARCHITECTURE arch_div4 OF div4 IS
    type TYPE_ETAT is (E1,E2,E3,E4);
    signal X : TYPE_ETAT ;
BEGIN
    PROCESS (H)
        BEGIN
            IF H'EVENT AND H='1' THEN
                case X is
                        when E1 =>      X<=E2;
                        when E2 =>      X<=E3;
                        when E3 =>      X<=E4;
                        when OTHERS =>  X<=E1;
                END case;
            END IF;
    END PROCESS;
S<='1' when X=E4 else '0';
END arch_div4;
```

# VHDL
## Exercice : un diviseur de fréquence par 3

***Sachant que l'on ne peut pas compter les fronts montants et les fronts descendants sur un même process . (maxplus2 d'ALTERA)***

***Trouver à l'aide de plusieurs process une méthode permettant de fabriquer un signal S qui à une période 3 fois plus grande que la période d'horloge.***

***Ts=3.Th***

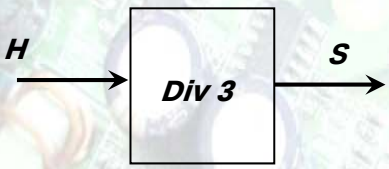


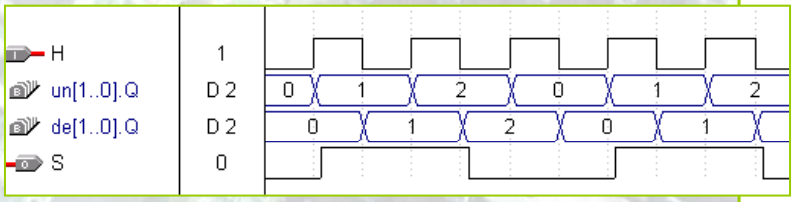

***Donner le code VHDL de ce diviseur.***





# VHDL
## Exercice : un diviseur de fréquence par 3




```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY div3 IS
    PORT (  H :     IN  STD_LOGIC;
            S:      OUT STD_LOGIC);
END div3;

ARCHITECTURE arch_div3 OF div3 IS
    signal un,de : integer range 0 to 2;
BEGIN
    PROCESS (H)
        BEGIN
            IF H'EVENT AND H='1' THEN
                un<=un+1;
                if un=2 then un<=0;
                end if;
            END IF;
    END PROCESS;
    PROCESS (H)
        BEGIN
            IF H'EVENT AND H='0' THEN
                de<=de+1;
                if de=2 then de<=0;
                end if;
            END IF;
    END PROCESS;
    PROCESS (un,de)
        BEGIN
            if (un=1 or de=1) then S<='1';
            else S<='0';
            end if;
    END PROCESS;
END arch_div3;
```

# VHDL
## Exercice : un diviseur de fréquence par 3 ou 4

***Cahier des charges du diviseur :***

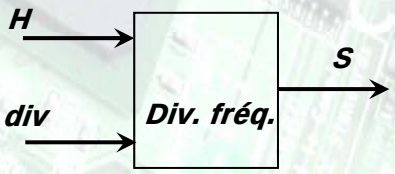
*-si* div = '0' ***la sortie S est un signal périodique, de fréquence égale au tiers de la fréquence d'horloge,*** ***→ (Ts=3.Th)***

→ *Le rapport cyclique du signal S est de 1/3.*

***- si*** div = '1' ***la fréquence du signal S est égale au quart de la fréquence d'horloge.***

***→ (Ts=4.Th)***

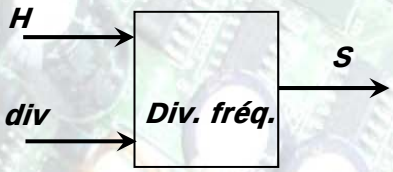
→ *Le rapport cyclique du signal S est de 1/4.*

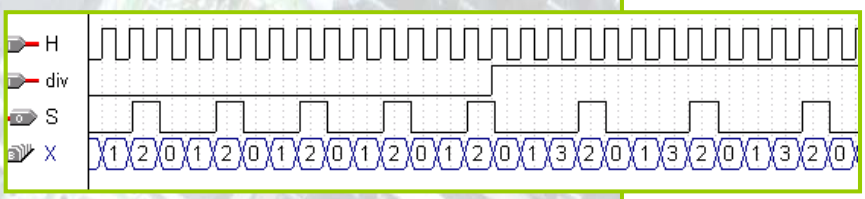





LINEL
Electronique Numérique

VHDL
Exercice : un diviseur de fréquence par 3 ou 4



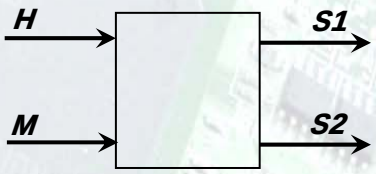




```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY div3_4 IS
    PORT (
         H,div :    IN  STD_LOGIC;
         S:         OUT STD_LOGIC
         );
END div3_4;
ARCHITECTURE arch_div3_4 OF div3_4 IS
    type TYPE_ETAT is (E1,E2,E3,E4);
    signal X : TYPE_ETAT ;
BEGIN
    PROCESS (H)
        BEGIN
            IF H'EVENT AND H='1' THEN
                case X is
                    when E1     =>  X<=E2;
                    when E2     =>  if div='0'  then X<=E3;
                                                 else X<=E4;
                                    end if;
                    when E4     =>  if div='1'  then X<=E3;
                                                 else X<=E1;
                                    end if;
                    when OTHERS =>  X<=E1;
                END case;
            END IF;
    END PROCESS;
S<='1' when X=E3 else '0';
END arch_div3_4;
```

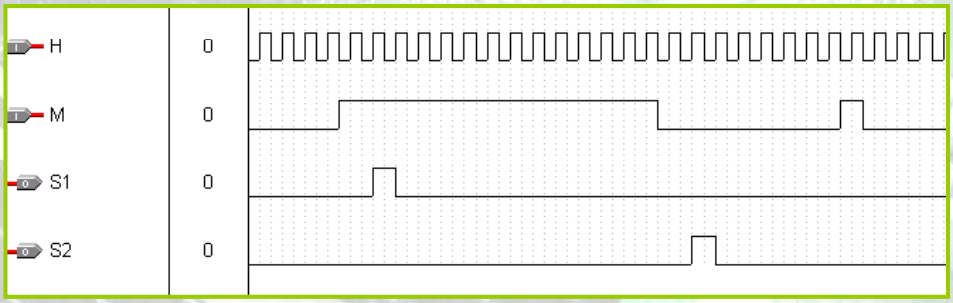
# VHDL
## Exercice : Le monoimpulseur

***Rôle d'un monoimpulseur :***
*Ce type de commande permet de filtrer les bruits parasites sur la commande de marche et d'attendre qu'elle soit bien établie.*

***Fonctionnement :***
*Le système comporte deux entrées : une horloge H et un ordre de marche M ainsi que deux sorties synchrones S1 et S2.*
*La première sortie génère une impulsion synchrone à l'horloge après la détection d'un front montant sur l'entrée marche. Pour que ce front soit validé, la commande de marche doit avoir été activée au moins pendant deux coups d'horloge consécutifs.*
*La sortie S2 présente une impulsion synchrone à l'horloge après que la commande de marche soit revenue à l'état logique bas pendant au moins deux coups d'horloge.*




***Chronogramme espéré du monoimpulseur***






LINEL
Electronique Numérique

# VHDL
## Exercice : Le monoimpulseur

***Donner le diagramme d'état.***
***le nombre d'états de ce diagramme implique le nombre de bits internes et par conséquent le nombre de bascules nécessaires pour coder la machine.***

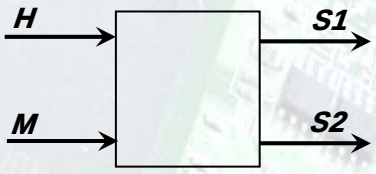
***Donner la table de transition.***
***état initial → état final.***

***On veut réaliser le monoimpulseur à l'aide de bascules D;***
***donner les tables de vérité pour chacune des bascules,***
***donner les équations d'entrées de ces bascules à l'aide des tableaux de Karnaugh.***

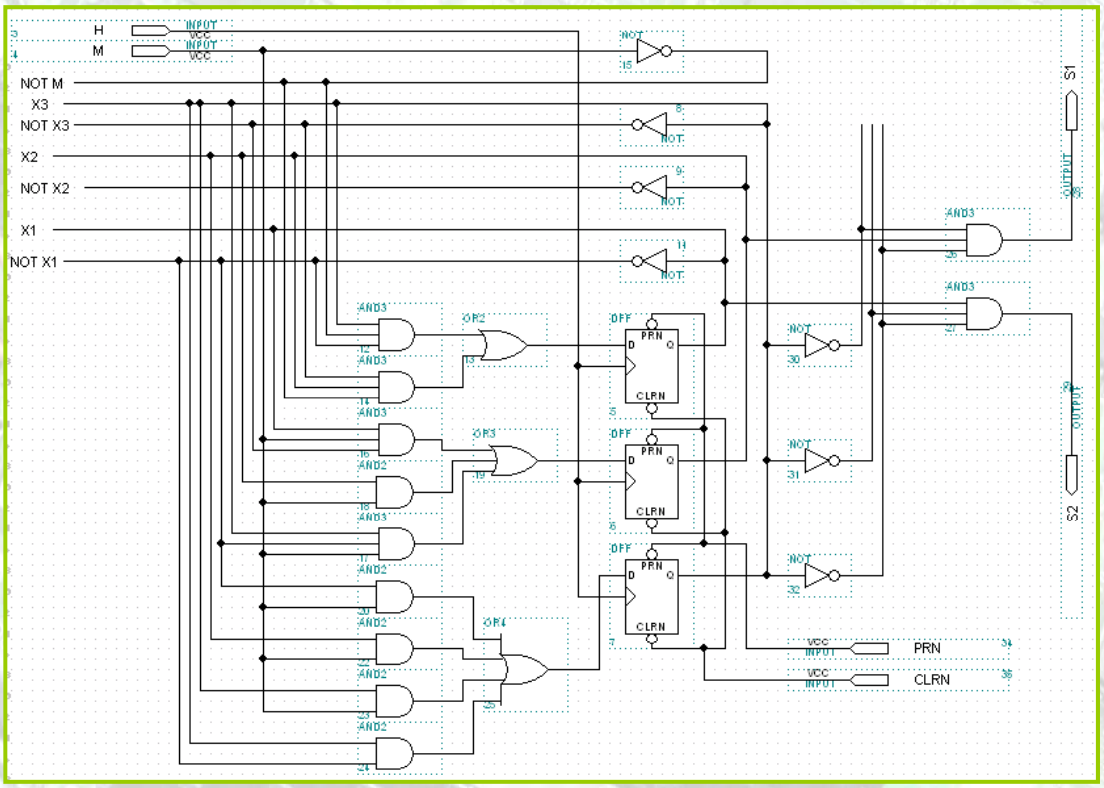
***Donner les équations pour les sortie S1 et S2.***

***Donner un schéma structurel du monoimpulseur complet.***

***Donner à partir du diagramme d'état le code VHDL.***



# VHDL
## *Exercice : Le monoimpulseur*

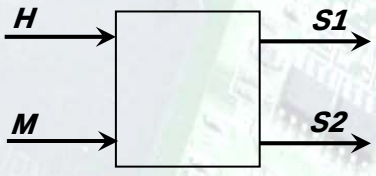


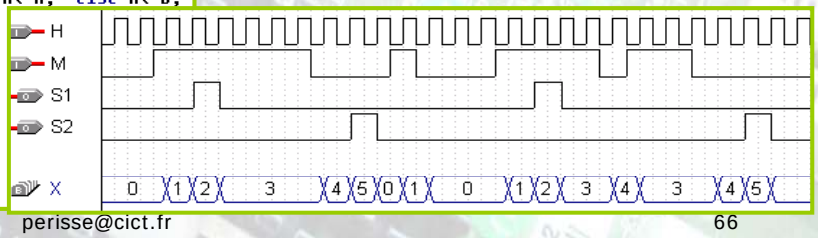





# VHDL
## *Exercice : Le monoimpulseur*

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY monopulse IS
    PORT (  H,M :       IN  STD_LOGIC;
            S1,S2:      OUT STD_LOGIC);
END monopulse;
ARCHITECTURE arch_monopulse OF monopulse IS
    type TYPE_ETAT is (A,B,C,D,E,F);
    signal X : TYPE_ETAT ;
BEGIN
    PROCESS (H)
        BEGIN
            IF H'EVENT AND H='1' THEN
                case X is
                when A  =>      if M='0'    then X<=A;  else X<=B;
                                end if;
                when B  =>      if M='0'    then X<=A;  else X<=C;
                                end if;
                when C  =>      if M='1'    then X<=D;  else X<=E;
                                end if;
                when D  =>      if M='0'    then X<=E;  else X<=D;
                                end if;
                when E  =>      if M='0'    then X<=F;  else X<=D;
                                end if;
                when F  =>      if M='0'    then X<=A;  else X<=B;
                                end if;
                when OTHERS =>  X<=A;
                END case;
            END IF;
    END PROCESS;
S1<='1' when X=C else '0';
S2<='1' when X=F else '0';
END arch_monopulse;
```



# VHDL
## *Exercice : Registre à décalage*

```
--registre à décalage à chargement parallèle synchrone
LIBRARY ieee;
USE ieee.std_logic_1164.all;
--USE work.std_arith.all;

entity DECAL_P is
    port (  H,R,IN_SERIE,LOAD :in std_logic;
            D :in std_logic_vector(3 downto 0);
            OUT_SERIE :out std_logic);
end DECAL_P;

architecture ARCH_DECAL_P of DECAL_P is
signal Q :std_logic_vector(3 downto 0);
begin
    process(H,R)
    begin
        if R='1' then Q <= "0000";
        elsif (H'event and H='1') then
        if LOAD = '1' then Q <= D;
        else Q <= Q(2 downto 0) & IN_SERIE;
        end if;
        end if;
    end process;
OUT_SERIE <= Q(3);
end ARCH_DECAL_P;
```