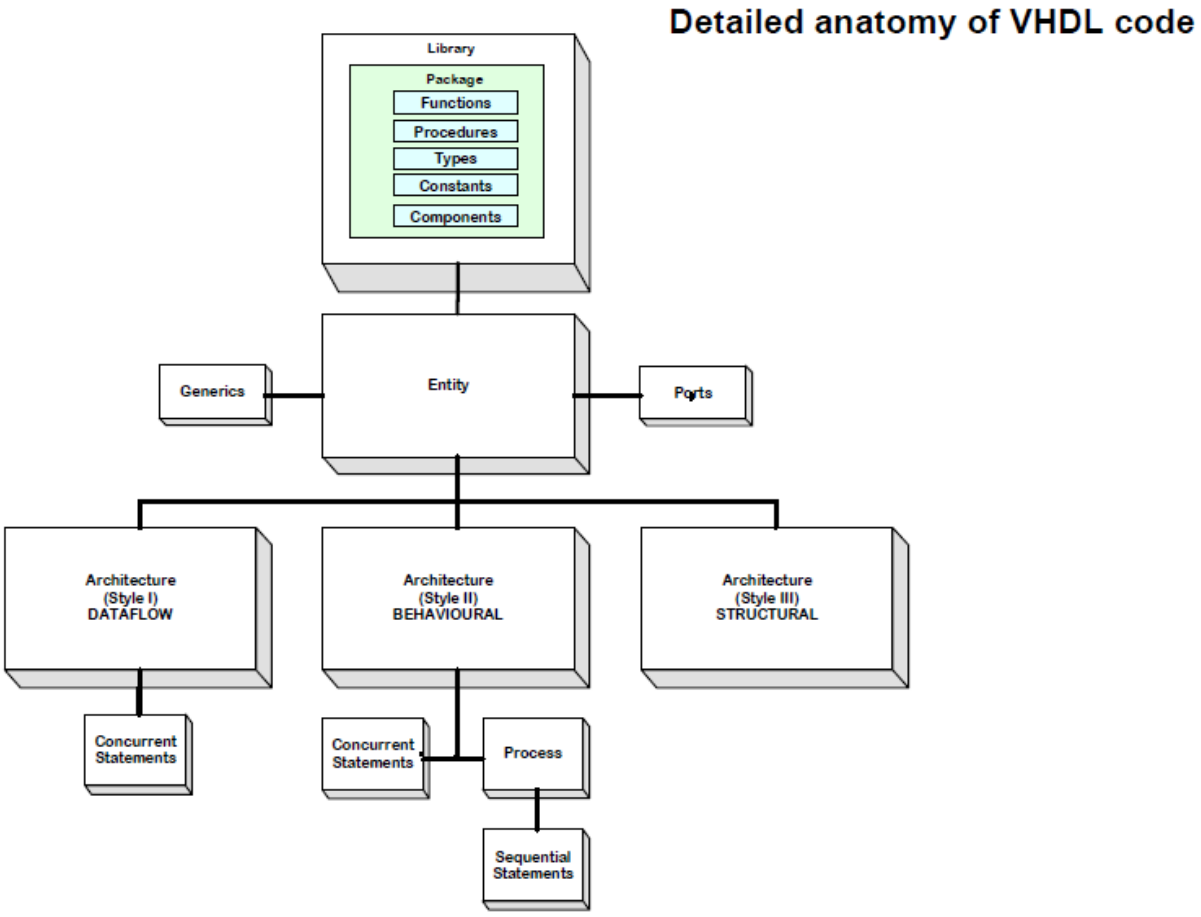
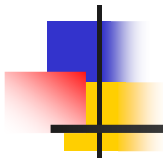
VHDL : Cours


Detailed anatomy of VHDL code
Library
Package
Functions
Procedures
Types
Constants
Components
Generics
Entity
Ports
Architecture (Style I) DATAFLOW
Architecture (Style II) BEHAVIOURAL
Architecture (Style III) STRUCTURAL
Concurrent Statements
Concurrent Statements
Process
Sequential Statements



C2 – Structure d'une description VHDL

Yann DOUZE

VHDL

Structure d'une description VHDL

- 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.
- Voir support de cours.

Déclaration des bibliothèques

- Toute description **VHDL** utilisée pour la synthèse a besoin de bibliothèques.
- L'**IEEE** (**I**nstitut of **E**lectrical and **E**lectronics **E**ngineers) les a normalisées et plus particulièrement la bibliothèque **IEEE 1164**.
- 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;
```


Déclaration de l'entité

- ***Syntaxe:***

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

- Exemple :

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

- **Remarque :** Après la dernière définition de signal de l'instruction ***port*** il ne faut jamais mettre de point virgule.

Déclaration des signaux E/S

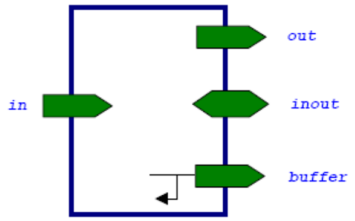
- L'instruction ***port*** :

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

Exemple: ***CLOCK:*** ***in*** ***std_logic;***

BUS : ***out*** ***std_logic_vector (7 downto 0);***

- 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 pouvant être lu (déconseillé).

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.
- ***'X' ou 'W'*** : pour un niveau inconnu.
- ***'U'*** : pour non initialisé.
- ***'Z'*** : pour état haute impédance.
- ***'-'*** : Quelconque, c'est à dire n'importe quelle valeur.

Description de l'architecture

- L'architecture est relative à une entité. Elle décrit le corps du design, son comportement, elle établit à travers les instructions les relations entre les entrées et les sorties.
- **Exemple :**

```
-- Opérateurs logiques de base
entity PORTES is
    port (A,B :in std_logic;
    Y1,Y2,Y3,Y4,Y5,Y6,Y7:out std_logic);
end entity;
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 architecture;
```


VHDL : langage concurrent ?

```
architecture DESCRIPTION of DECOD is
Begin
     -- instructions concurrentes
    D0 <= (not(IN1) and not(IN0));      -- première instruction
    D1 <= (not(IN1) and IN0);           -- deuxième instruction
end architecture;
```

- Entre le BEGIN et le END de l'architecture, on est dans un contexte d'instructions concurrentes.
- Instructions concurrentes :
  - L'ordre dans lequel sont écrites les instructions n'a aucune importance.
  - Toutes les instructions sont évaluées et affectent les signaux de sortie en même temps.
  - C'est la différence majeure avec un langage informatique.

L'architecture ci dessous est équivalente :

```
architecture DESCRIPTION of DECOD  is
begin
    D1 <= (not(IN1) and IN0);           -- deuxième instruction
    D0 <= (not(IN1) AND not(IN0));      -- première instruction
end architecture;
```

Exemple d'une description VHDL :

```
library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;
```

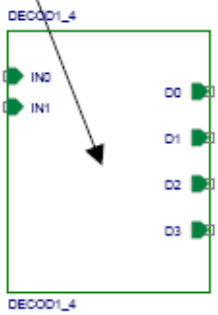
Déclaration des bibliothèques

Commentaires, en VHDL ils commencent par - -

```
-- décodeur
-- Un parmi quatre
```

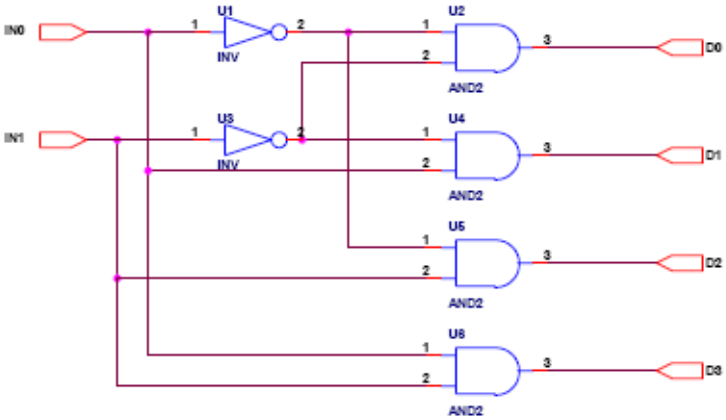
Déclaration de l'entité du décodeur
Correspondance schématique

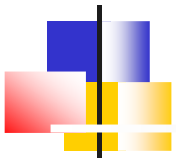
```
entity DECOD1_4 is
      port(IN0, IN1: in std_logic;
           D0, D1, D2, D3: out std_logic);
end DECOD1_4;
```



Déclaration de l'architecture du décodeur
Correspondance schématique

```
architecture DESCRIPTION of DECOD1_4 is
begin
      D0 <= (not(IN1) and not(IN0));
      D1 <= (not(IN1) and IN0);
      D2 <= (IN1 and not(IN0));
      D3 <= (IN1 and IN0);
end DESCRIPTION;
```

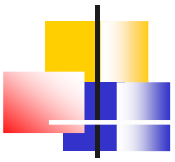




C3 – Les opérateurs de base

Yann DOUZE

VHDL



Affectation simple : <=

Exemples :

S <= E2 ;

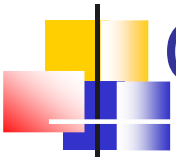
S <= '0' ;

S <= '1' ;

Pour les **signaux** de plusieurs bits on utilise les doubles cotes **" ... "** ,

BINAIRE, exemple : ***BUS <= "1001" ; -- BUS = 9 en décimal***

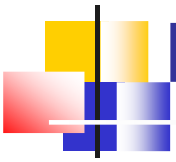
HEXA, exemple : ***BUS <= x"9" ; -- BUS = 9 en hexa***



Opérateurs logiques

NON	→	not
ET	→	and
NON ET	→	nand
OU	→	or
NON OU	→	nor
OU EXCLUSIF	→	xor

Exemple : ***S1 <= (E1 and E2) or (E3 nand E4);***



Exercices

Faire les exercices 1 et 2.

Opérateurs relationnels

- Ils permettent de modifier l'état d'un signal suivant le résultat d'un test ou d'une condition.

Egal	→ **=**
Non égal	→ **/=**
Inférieur	→ **<**
Inférieur ou égal	→ **<=**
Supérieur	→ **>**
Supérieur ou égal	→ **>=**

Affectation conditionnelle

- 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]*** permet de définir la valeur du ***SIGNAL*** par défaut.

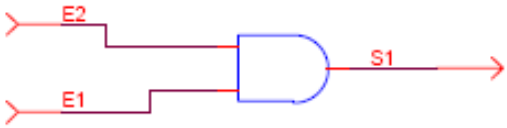
Affectation conditionnelle (2)

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 : ET logique



Exemple N°2 :

-- Description comportementale d'un multiplexeur 2 vers 1

```
Y <= A when (SEL='0' ) else
     B when (SEL='1' ) else '0';
```


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*** par défaut

Affectation sélective (2)

Exemple : ***Multiplexeur 2 vers 1***

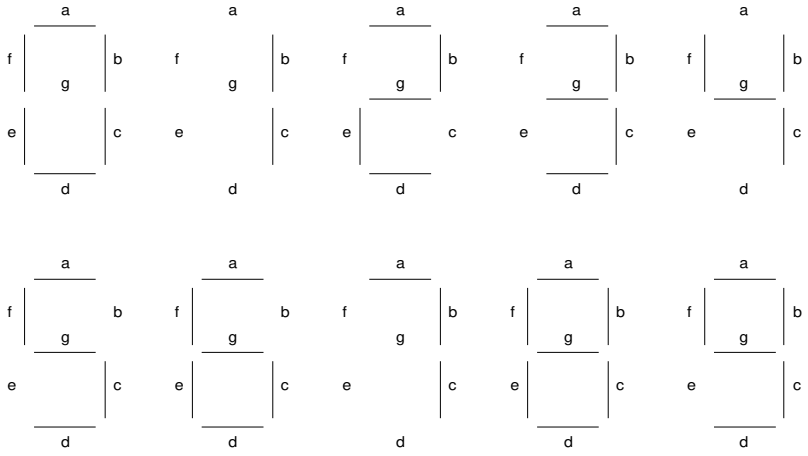
```
with SEL select
      Y <=  A when '0',
      B when '1',
      '0' when others;
```

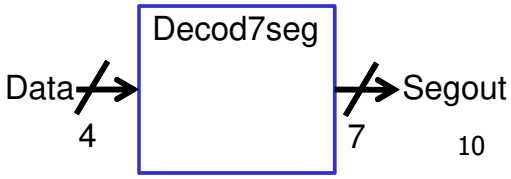
Remarque: Dans le cas du multiplexeur, ***when others*** est nécessaire car il faut toujours définir les autres cas du signal de sélection pour prendre en compte toutes les valeurs possibles de celui-ci.

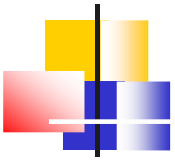
```
with SEL select
      Y <= A when '0',
       B when '1',
       '-' when others;
-- pour les autres cas de SEL, Y prendra une valeur quelconque
-- permet d'optimiser la synthèse
```


Exemple : décodeur 7 segments (1)

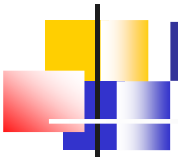
7-Segments Decoder - BCD (0..9) only







Exemple : décodeur 7 segments (2)



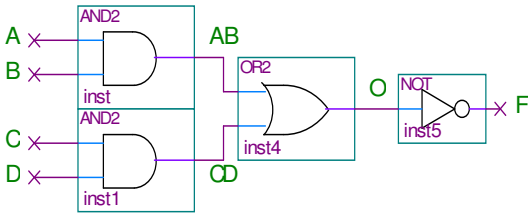
Exercice

- Faire l'exercice 3 en utilisant une affectation sélective ou conditionelle.

Signaux internes

- **Syntaxe :** **signal** ***NOM_DU_SIGNAL*** ***: type;***
- **Exemple :** **signal** ***I*** ***: std_logic;***
  signal ***BUS*** ***: std_logic_vector (7 downto 0);***

On peut décrire le schéma suivant de 2 manières différentes :



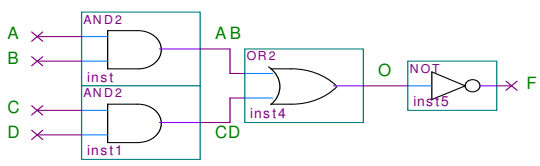
- Sans signaux internes

```
architecture V1 of AOI is
Begin
   F <= not ((A and B) or (C and D));
end architecture V1;
```


Signaux internes (2)

- Avec des signaux internes

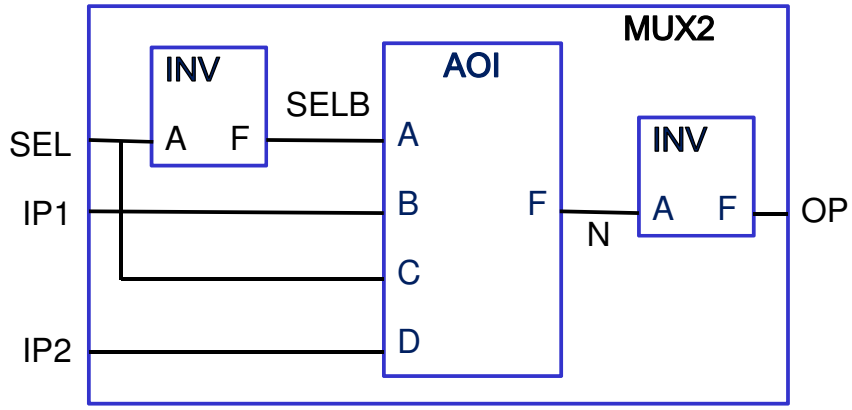
```
architecture V2 of AOI is
   --zone de declaration des signaux
   signal AB,CD,O: STD_LOGIC;
begin
   --instructions concurrentes
   AB <= A and B;
   CD <= C and D;
   O  <= AB or CD;
   F  <= not O;
end V2;
```



- Instructions concurrentes :
  - L'ordre dans lequel sont écrites les instructions n'a aucune importance.
  - Toutes les instructions sont évaluées et affectent les signaux de sortie en même temps.
  - Différence majeure avec un langage informatique.

Description structurelle

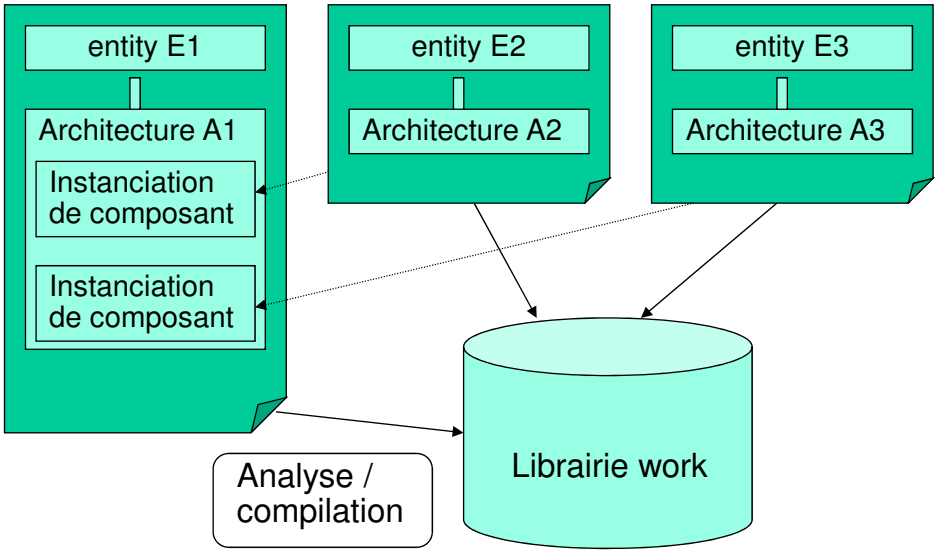
- C´est une description de type **hiérarchique** par liste de connexions (netlist).
- Une description est structurelle si elle comporte un ou plusieurs composants.
- Exemple :



Description structurelle

- La **marche à suivre** :
  - Dessiner le schéma des composants à instancier.
  - Déclarer les listes de signaux internes nécessaires pour le câblage: SIGNAL...
  - Instancier chaque composant en indiquant sa liste de connexions: PORT MAP...

Instanciation de composant



Déclaration des composants INV et AOI

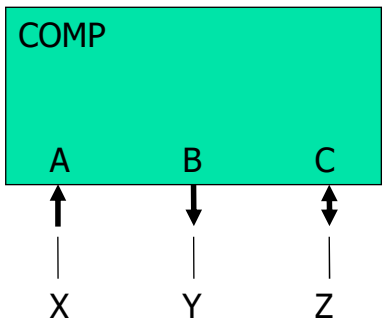
```
entity INV is
  port ( A   : in  STD_LOGIC;
         F   : out STD_LOGIC);
end entity;
architecture V1 of INV is
  ...
end architecture;

entity AOI is
  port ( A,B,C,D  : in  STD_LOGIC;
         F        : out STD_LOGIC);
end entity;
architecture V1 of AOI is
  ...
end architecture;
```

Instanciation Directe

```
entity MUX2 is
   port ( SEL,IP1,IP2   : in  STD_LOGIC;
          op            : out STD_LOGIC);
end entity;
architecture DIRECTE of MUX2 is
   signal SELB, N: STD_LOGIC;
begin
   G1: entity WORK.INV(V1) port map (A => SEL, F => SELB);
   G2: entity WORK.AOI(V1) port map (
       A => SELB, B => IP1,
       C => SEL, D => IP2,
       F => N);
   G3: entity WORK.INV(V1) port map (A => N, F => OP);
end architecture;
```


Association par nom ou par position

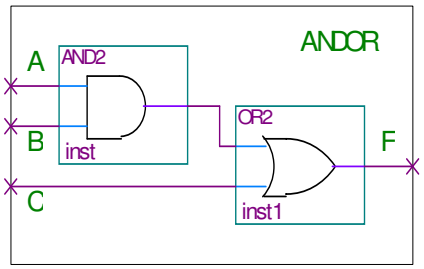
```
entity COMP
port(  A: in     STD_LOGIC;
       B: out    STD_LOGIC;
       C: inout  STD_LOGIC);
end entity;

Architecture V1 of COMP is
Signal X,Y,Z: STD_LOGIC;
begin
```




```
C1: entity work.COMP port map (A => X, B => Y, C => Z);
--association par nom
```

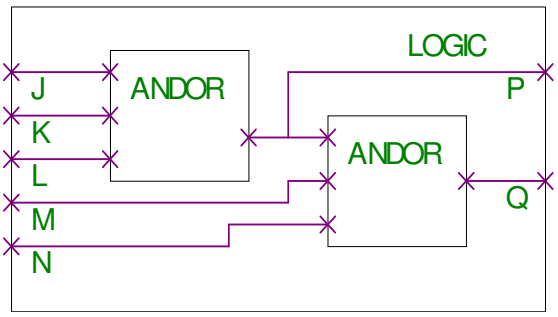
```
C1: entity work.COMP port map (X, Y, Z); --association par position
```

Exercice à compléter

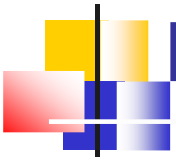



```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity ANDOR is
   port (
 A,B,C : in std_logic;
 F : out std_logic);
end entity;

Architecture dataflow of ANDOR is
Begin
   F <= (A and B) or C;
End architecture;
```


Exercice à compléter




```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity LOGIC is
   port (
   J,K,L,M,N : in std_logic;
   P,Q : out std_logic);
end entity LOGIC;
Architecture STRUCT of LOGIC is
```



Exercices

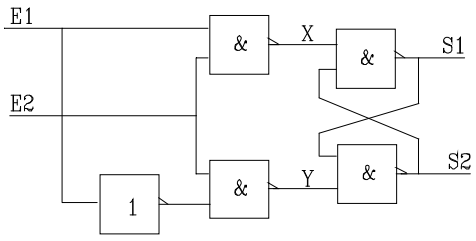
- Faire les exercices 4 et 5.

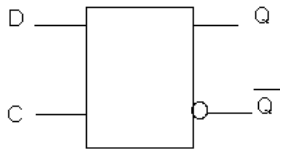
C4 – Les instructions séquentielles

Yann DOUZE

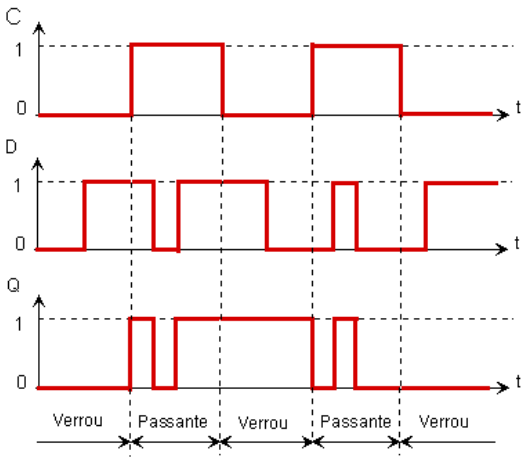
VHDL

Rappel : Bascule D Latch

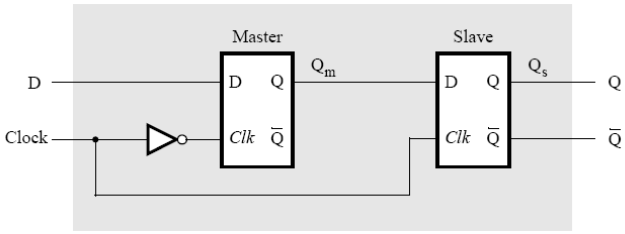


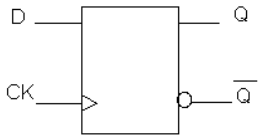


Entrées		Sorties	
C	D	Qn+1	$\overline{Qn+1}$
0	X	Qn	$\overline{Qn}$
1	0	0	1
1	1	1	0

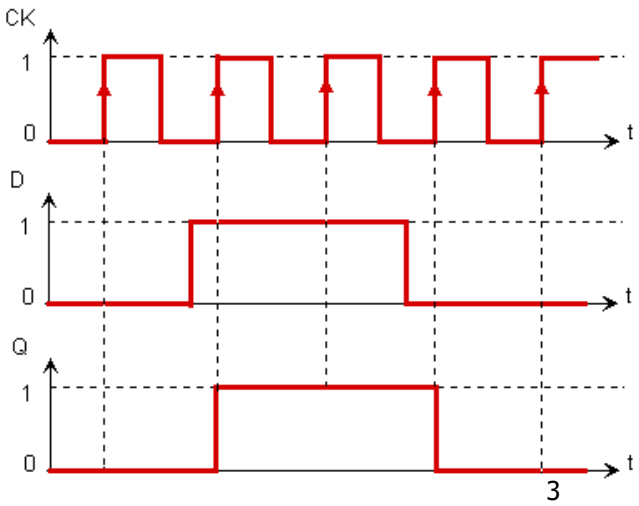


Rappel : Bascule D Flip-Flop





Entrées		Sorties	
CK	D	Qn+1	$\overline{Qn+1}$
0	X	Qn	$\overline{Qn}$
1	X	Qn	$\overline{Qn}$
↓	X	Qn	$\overline{Qn}$
↑	0	0	1
↑	1	1	0

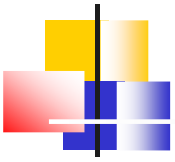


Définition du process

- Un **process** est une partie de la description d'un circuit dans laquelle les instructions sont exécutées séquentiellement : 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.

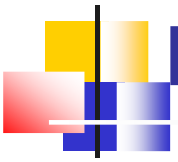
Syntaxe:

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



Règles de fonctionnement d'un process

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



Role des processus

- Les processus sont utilisé avec 3 approches différentes :

1. Pour décrire un circuit combinatoire avec des instructions évolué de la programmation structuré : if, case, etc…
2. Pour décrire un circuit synchrone qui comporte une ou plusieurs cellules de mémorisation (bascules Dff).
3. Pour décrire une fonction non synthétisable tel qu'un banc de test ou un modèle.

Instructions séquentielles

- Attention !
- Les instructions **if** et **case** n'existent et ne peuvent exister que dans un process.

Instruction if

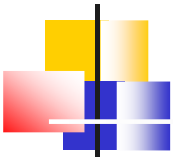
Syntaxe :

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

Exemple:

```
Process (A,B,E1,E2,E3)
Begin
    if  A='1'  then   SORTIE <= E1;
    Elsif B = '1' then SORTIE <= E2;
    Else  SORTIE <= E3;
    end if ;
end process ;
```

Remarque : l'instruction if ne peut être utilisée que dans un process

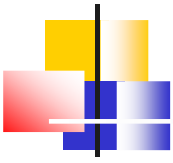


Instruction case

Syntaxe:

CASE expression IS
WHEN choix => instructions;
[WHEN choix => instructions;]
[WHEN OTHERS => instructions;]
END CASE;

Remarque : pareil que l'instruction if, l'instruction case ne peut être utlisée que dans un process.



Exemple

```
process (TEST,A,B)
begin
   case TEST is
    when "00" => F <= A and B;
    when "01" => F <= A or B;
    when "10" => F <= A xor B;
    when "11" => F <= A nand B;
    when others => F <= '0';
   end case;
end process;
```

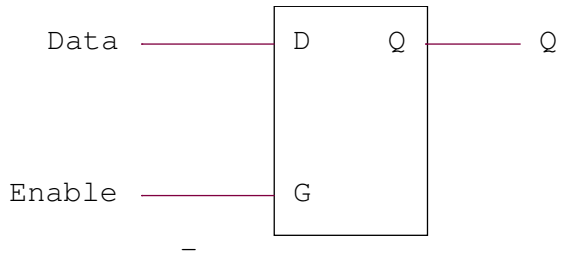
Process combinatoire

```
process (SEL,A,B,C)
begin
  if SEL ='1' then
    OP <= A and B;
  else
    OP <= C;
  end if;
end process;
```

- La liste de sensibilité doit être complète : tout les signaux lus dans le process doivent apparaître dans la liste de sensibilité.
- les sorties doivent être assignés dans tout les cas afin d'éviter les bascules D Latch.

Affectation incomplète

```
process (Enable,Data)
begin
  if Enable = '1' then
      Q <= Data;
  end if;
end process;
```



• Une affectation incomplète génère une bascule D Latch (verrou) non désiré (transparent latch)

• Certain FPGA n'ont pas de bascule D Latch (verrou), une boucle asynchrone sur du combinatoire est alors créé (interdit!)

Assignement par défaut

- Pour éviter les transparents Latch on préconise de faire un assignement par défaut.

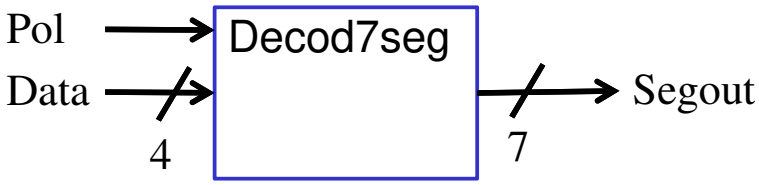
```
Process (SELA, SELB, A, B)
begin
   OP <= '0';
   if SELA = '1' then
      OP <= A;
   end if;
   if SELB = '1' then
      OP <= B;
   end if;
end process;
```

Assignement par défaut

Remplace l'événement par défaut

Remplace l'événement à nouveau

Exemple : décodeur 7 segments




```
Entity SEVEN_SEG is
port(
 Data : in  std_logic_vector(3 downto 0); --Expected within 0…9
 Pol  : in  std_logic;                    -- '0' if active LOW
 Segout: out std_logic_vector(1 to 7)); --Segments A,B,C,D,E,F,G
end entity;
```

Architecture combinatoire du décodeur 7 segments

```
architecture COMB of SEVEN_SEG is
     signal sevseg : std_logic_vector(1 to 7);
begin
Process(Data, Pol,sevseg)
Begin
     case(Data) is
               when x”0”=> sevseg <= “1111110”;
               when x"1" => sevseg <= "0110000";
               when x"2" => sevseg <= "1101101";
               when x"3" => sevseg <= "1111001";
               when x"4" => sevseg <= "0110011";
               when x"5" => sevseg <= "1011011";
               when x"6" => sevseg <= "1011111";
               when x"7" => sevseg <= "1110000";
               when x"8" => sevseg <= "1111111";
          when x"9" => sevseg <= "1111011";
               when others => sevseg <= (others => '-');
     end case;
     if (Pol=‘1’) then
          Segout <= sevseg;
     else
          Segout <= not(sevseg);
     end if;
End process;
End architecture;
```


Exercice

- Exercice 1 : Multiplexeur 8 vers 1

Process synchrone

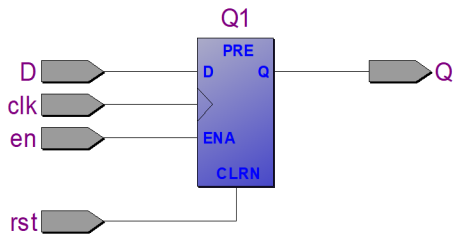
```
process(CLK)
begin
if RISING_EDGE(CLK) then
    Q1 <= D;
end if;
end process;
```

```
process(CLK)
begin
if FALLING_EDGE(CLK) then
    Q2 <= D;
end if;
end process;
```

- **Un process synchrone est exécuté à chaque front montant de l'horloge**.
- Pour tester le front montant de l'horloge, on utilise la fonction rising_edge() qui est définit dans le package STD_LOGIC_1164.
- RISING_EDGE est VRAI lorsque l'horloge passe de l'état '0' à l'état '1'.
- Utile pour décrire une bascule D flip-flop (Dff).

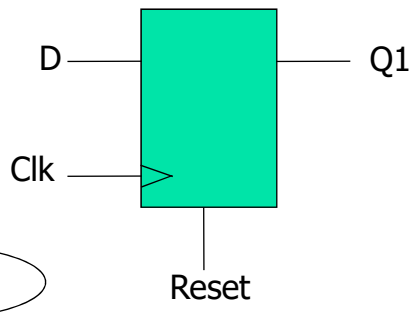
Comment rajouter un reset asynchrone?

Reset Asynchrone




```
process(Reset,Clk)
begin
if Reset = '1' then
    Q1 <= '0';
elsif RISING_EDGE(Clk) then
    Q1 <= D;
end if;
end process;
```

- Le reset asynchrone est testé avant le front montant de l'horloge.

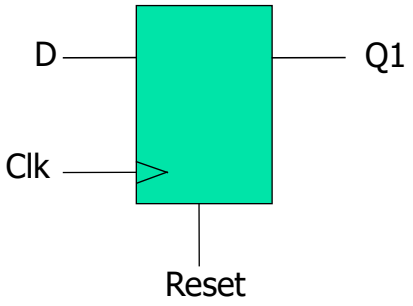


Comment faire un reset synchrone?

Reset Synchrone

```
process(Clk)
begin
if RISING_EDGE(Clk) then
    if Reset = '1' then
     Q1 <= '0';
    else
     Q1 <= D;
    end if;
end if;
end process;
```

- Le reset synchrone est testé après le front montant de l'horloge.



Transfert par registre (RTL)

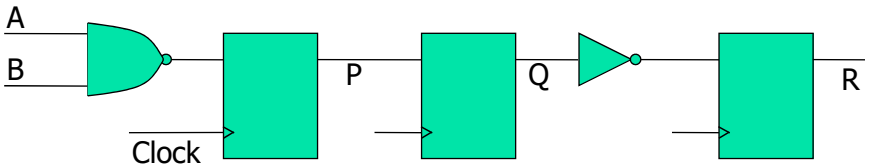
- Dans un process synchrone, à chaque affectation, une bascule est créée.

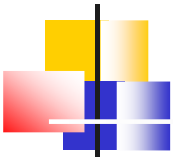
```
process (Clock)
begin
   if Rising_edge(Clock) then
      P <= A nand B;
      Q <= P;
      R <= not Q;
   end if;
end process;
```

Clock

Registres

Logique combinatoire

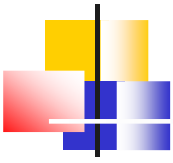




Règles des process synchrones

- Liste de sensibilité doit inclure le clock et le reset et rien d'autre.
- Tout les signaux assignés dans le process doivent être initialisés par un reset.
- Pour la description d'un process synchrone, respectez la structure suivante :

```
process(clk,rst)
begin
if rst = '1' then
 -- valeur initiale
elsif rising_edge(clk) then
 -- instructions
end if;
end process;
```

- Attention les signaux prennent leurs nouvelles valeurs qu'à la fin du process.
- Complétez le code et dessinez le chronogramme d'un registre à décalage à 4 étages ?

Registre à décalage

```
-- Registre à décalage
entity registre_decalage is
port( Qin, rst, clk : in std_logic;
       Qout          : out std_logic);
end entity;
architecture RTL of registre_decalage is
    signal Q1,Q2,Q3 : std_logic;
begin
process(Rst,Clk)
begin
if Rst = '1' then
    Qout <= '0'; Q1<='0';Q2<='0';Q3<='0';
Elsif rising_edge(clk) then
    Qout <= Q3;
    Q3 <= Q2;
    Q2 <= Q1;
    Q1 <= Qin;
End if;
End process;
End architecture;
```


Mauvais style de process (1) ?

```
process(Clock,Reset)
Begin
     if Rising_edge(Reset) then
          ...
     elsif Rising_edge(Clock) then
          ...
     end if;
end process;
```

```
process(Clock,Reset,Ena)
begin
     if Reset = '1' then
          -- Actions asynchrone
     elsif Rising_edge(Clock) and Ena = '1' then
          -- Actions synchrones
     end if;
end process;
```

```
process(Clock,Reset)
begin
     if Reset = '1' then
          -- Actions asynchrone
     elsif Rising_edge(Clock) then
          -- Actions synchrones
     end if;
end process;
```

```
process(All_Inputs)
begin
 -- Logique purement combinatoire
end process;
```

Mauvais style de process (2) ?

```
process(Clock,Reset)
begin
      if Reset = '1' then
            -- Actions asynchrone
      elsif Rising_edge(Clock) then
            -- Actions synchrones
      end if;
            -- d'autres actions
end process;
```

```
process(Clock)
begin
      if Rising_edge(Clock) then
            -- Actions synchrones
      end if;
end process;
```

```
process(Clock,Reset)
begin
      if Reset = '0' then
            if Rising_edge(Clock) then
                  -- Actions synchrones
            end if;
      else
            -- Actions asynchrones
      end if;
end process;
```


Exercice

Exercice 2 : serial OR ou LFSR

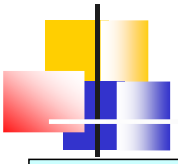
Process non-synthétisable

- Les process non synthétisable sont utilisé pour :
  - décrire un modèle
  - écrire des bancs de test (test bench)
- Les process non synthétisable **ne possèdent pas de liste de sensibilité** et ce sont des **instructions WAIT** qui synchronisent le process.
- Trois formes de WAIT peuvent être combinées
  - WAIT ON *événement*; Exemple : wait on A,B,C,D
    - Remplace la liste de sensibilité d'un process classique
  - WAIT FOR *durée*; Exemple : wait for 100 ns
    - Permet de créer un délais
  - WAIT UNTIL *condition*; Exemple : wait until rst = '1'
    - Condition bloquante
  - WAIT;
    - Boucle infini, comme un while(1) en C

Exemples de Process non-synthétisable

```
-- Génération d'un reset
STIMULUS: process
begin
   reset <= '1';
   wait for 50 NS;
   reset <= '0';
   wait;
end process STIMULUS;
```

```
-- Génération d'une horloge
ClockGenerator: process
begin
Clock <= '0';
wait for 5 NS;
Clock <= '1';
wait for 5 NS;
end process;
```



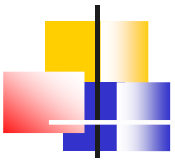
Instructions de boucle

```
for PARAMETER in LOOP_RANGE loop
   ...
end loop;
```

```
While CONDITION loop
   ...
end loop;
```

```
boucle1: -- étiquette optionnelle
FOR i IN 0 TO 10 LOOP
  -- calcul des puissances de 2
  b := 2**i;
  WAIT FOR 10 ns; --toutes les 10 ns
END LOOP;
```

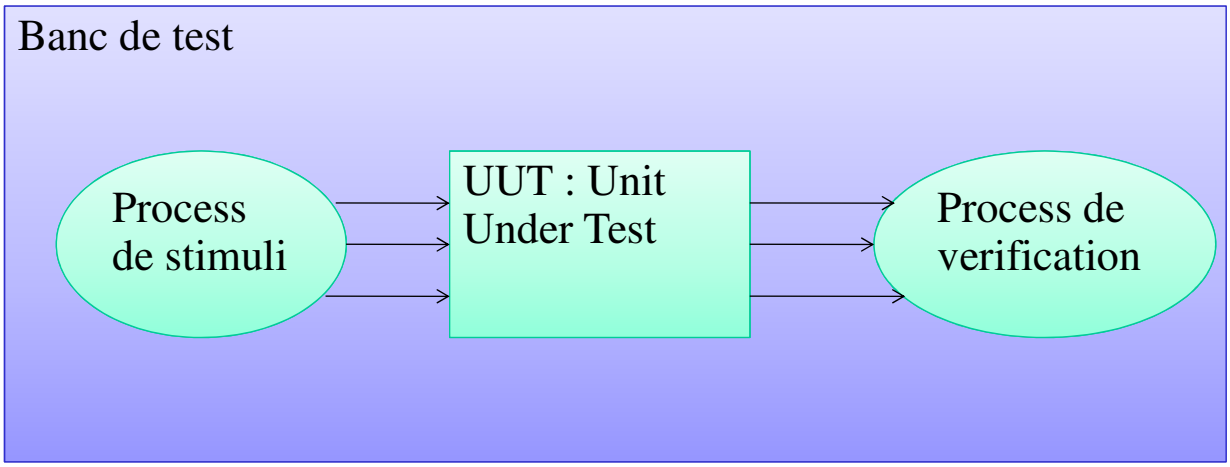
```
I := 0;
WHILE b < 1025 LOOP
   b := 2**i;
   i := i+1;
   WAIT FOR 10 ns;
END LOOP;
```

Exemple de boucle

```
ClockGenerator: process
begin
  while not Stop loop
    Clock <= '0';
    wait for 5 NS;
    Clock <= '1';
    wait for 5 NS;
  end loop;
  wait;
end process;
```

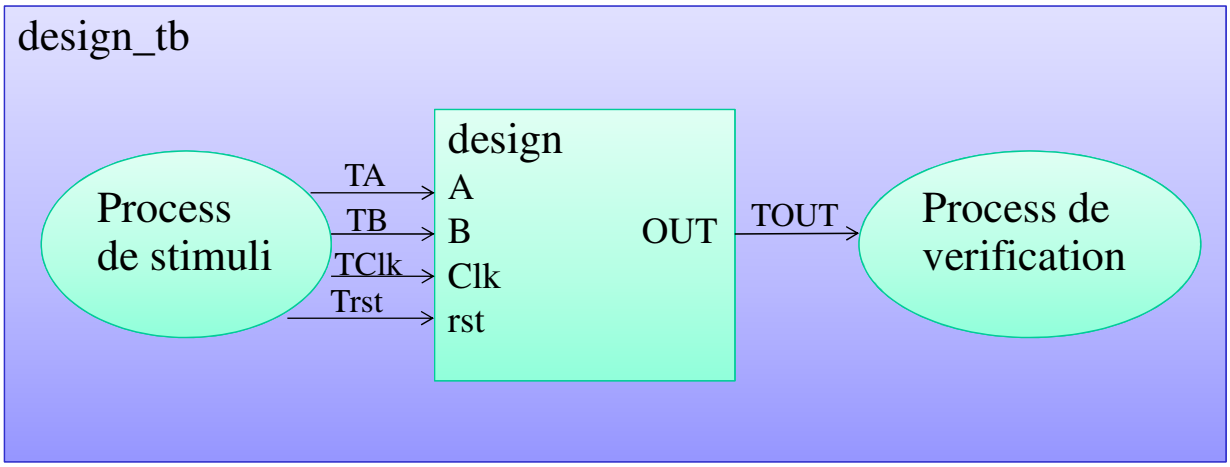
Banc de test (Test Bench)



Verification = Assertion

- **ASSERT** : permet d'écrire des messages à l'écran.
- Syntaxe :
  - `ASSERT condition REPORT message <SECURITY …>`
- Forcer un message:
  - `ASSERT FALSE REPORT "Toujours à l'écran " SEVERITY note;`
  - `REPORT "Toujours à l'écran " SEVERITY note;`
- Test en contexte séquentiel,
  - `ASSERT s = '1' REPORT "la sortie vaut '0' et ce n'est pas normal " SEVERITY warning; -- mentionne un warning`
  - `ASSERT s = '1' REPORT "la sortie vaut '0' et ce n'est pas normal " SEVERITY failure; -- Mentionne une erreur et arrête l'exécution`

Exemple



Banc de Test (1)

```
Entity design_tb is
End entity;
Architecture bench of design_tb is
    Signal Tclk : std_logic := '0';
    Signal Trst : std_logic;
    signal TA,TB,TOUT : std_logic_vector(3 downto 0);
    signal Done : boolean := False;
Begin
-- instanciation du composant à tester
UUT: entity work.design port map (
    A => TA, B => TB,
    OUT => TOUT,
     clk => Tclk, rst => Trst);
-- Génération d'une horloge
Tclk <= '0' when Done else not Tclk after 50 ns;
-- Génération d'un reset au début
Trst <= '1', '0' after 5 ns;
```

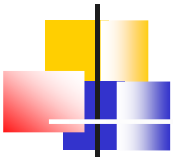
Banc de Test (2)

```
Stimuli: process
begin
    TA <= "0000";
    TB <= "0000";
    wait for 10 NS;
    TA <= "1111";
    wait for 10 NS;
    TB <= "1111";
    wait for 10 NS;
    TA <= "0101";
    TB <= "1010";
    wait;
end process;
```

```
Verification: process
begin
wait for 5 NS;
assert TOUT = "0000" report "erreur"
severity warning;
wait for 10 NS;
assert TOUT = "1111" report "erreur"
severity warning;
wait for 10 NS;
assert TOUT = "1111" report "erreur"
severity warning;
wait for 10 NS;
assert TOUT = "1010" report "erreur"
severity warning;
Done <= True;
wait;
end process;
End architecture;
```


Banc de Test : alternative

```
Stimuli_&_verification: process
Begin
TA <= "0000";
TB <= "0000";
wait for 5 NS;
assert TOUT = "0000" report "erreur" severity warning;
wait for 5 NS;
TA <= "1111";
wait for 5 NS;
assert TOUT = "1111" report "erreur" severity warning;
wait for 5 NS;
TA <= "0101";
TB <= "1010";
wait for 5 NS;
assert TOUT = "1010" report "erreur" severity warning;
Done <= True;
wait;
end process;
End architecture;
```



Exercice

- Exercice 3 : circuit Min Max

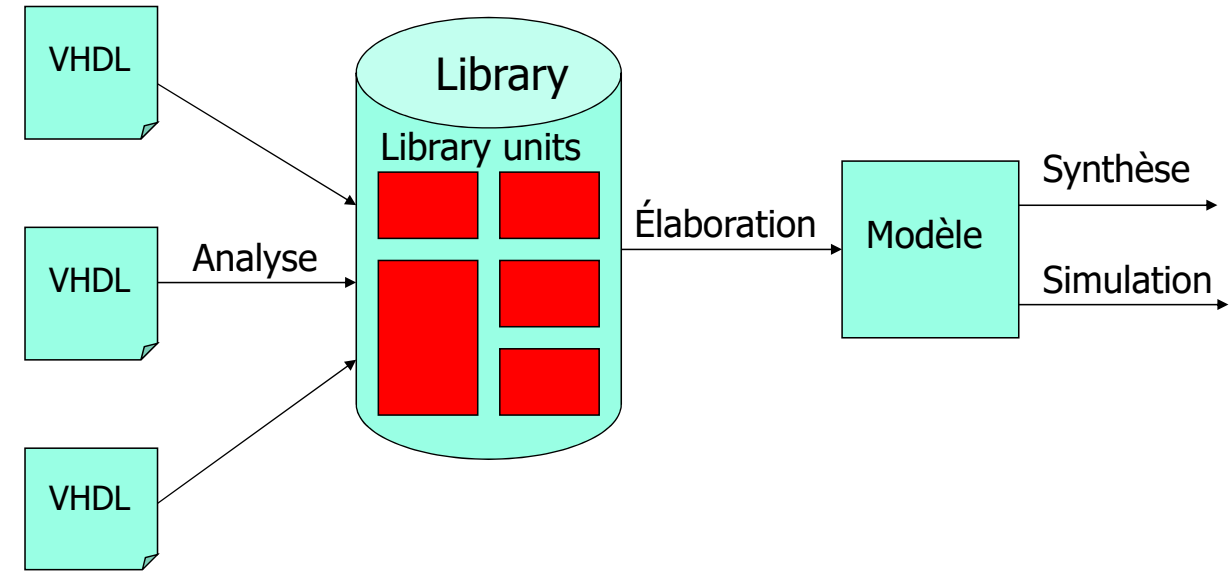
C5 – Fichiers et Librairies

Yann DOUZE

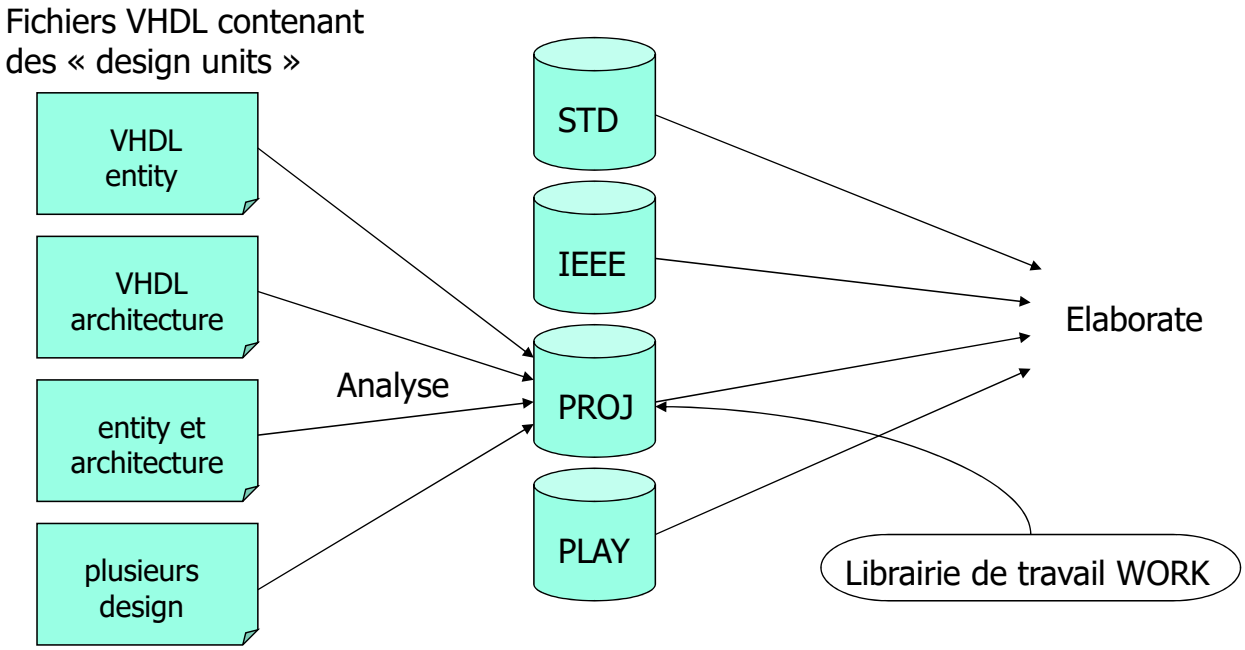
VHDL

Compilation

Fichiers VHDL contenant
des « design units »



Librairie de travail



La clause de contexte

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity TEST_BENCH1 is
end entity;

architecture TB of TEST_BENCH1 is
  ...
end architecture;

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity ENTITY2 is
  port( ... );
end entity;

architecture RTL of ENTITY2 is
  ...
end architecture;
```

Entité vide => la clause de contexte se reporte devant l'architecture

Chaque entité/ package/ configuration nécessite une clause de contexte

Clause de contexte implicite

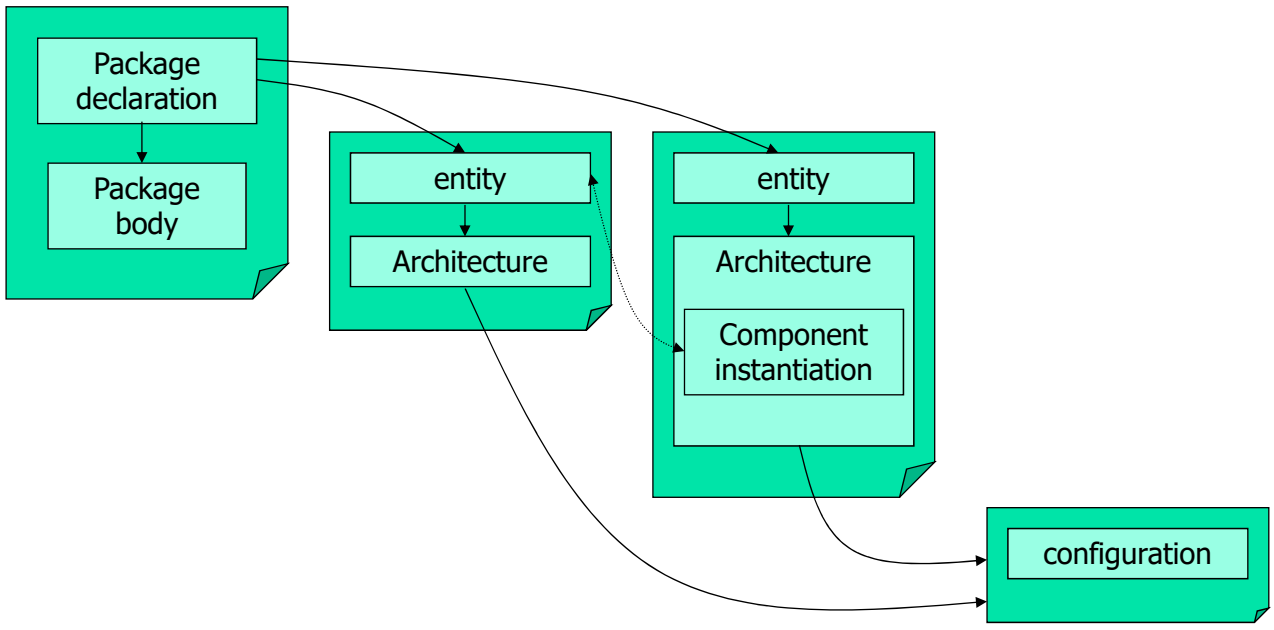
```
library STD, WORK;
use STD.STANDARD.all;
```

Inclut par défaut

```
INTEGER   0  1  99
BIT       '0'  '1'
BOOLEAN   FALSE  TRUE
STRING    "Hello World"
TIME      10 NS
```

Types du package STD.STANDARD

Ordre de la compilation



Style Recommandé

VHDL 93

Indentation

Instanciation directe

```
architecture BENCH of NOR2_TB is


  signal A, B, F: STD_LOGIC;

begin

  stimulus: process is
  begin
    A <= '0';
    B <= '0';
    wait for 10 NS;
    B <= '1';
    wait;
  end process stimulus;

  UUT: entity work.NOR2 port map (
        A => A,
        B => B,
        F => F);

end architecture BENCH;
```

Alignement

Des espaces

Des labels compréhensibles

Association par nom

Noms des labels

Identifiant = lettres, nombres et underscores

```
G4X6     Gate_45     \extended!"£$%^&*()\
TheState      The_State
```

Noms différents

La casse est ignorée pour les identifiants

```
INPUT    Input    input
```

mêmes noms

Beaucoup d'identifiant son des mots réservés

```
And   buffer   bus   function   register   select
```

Identifiants illegaux ...

```
4plus4   A$1   V-3   The__State    _State_
```

Double underscore

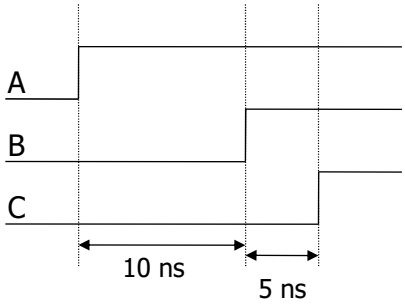
C6 – Délais, Delta Délais et variables

Yann DOUZE

Les délais

chronogramme de A ,B et C ?

```
process (A,B)
begin
   B <= A after 10 ns;
   C <= B after 5 ns;
end process;
```

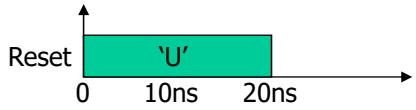


- Les délais sont utilisés pour :
  - Générer des stimuli pendant la simulation
  - Modéliser les retards dû à la technologie
- Les délais sont ignorés à la synthèse.

Génération d'une impulsion (1)

```
process
begin
   reset <= '0';
   reset <= '1' after 10 ns;
   reset <= '0' after 20 ns;
   wait;
end process;
```

Mauvaise syntaxe !

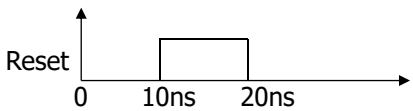


Génération d'une impulsion (2)

```
Reset <= '0', '1' after 10 ns, '0' after 20 ns;
```

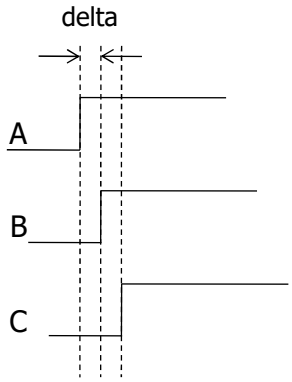
```
process
begin
   reset <= '0';
   wait for 10 ns;
   reset <= '1'
   wait for 10 ns;
   reset <= '0';
   wait;
end process;
```

Bonne syntaxe !



Delta Délais

```
process (A,B)
begin
   B <= A;
   C <= B;
end process;
```

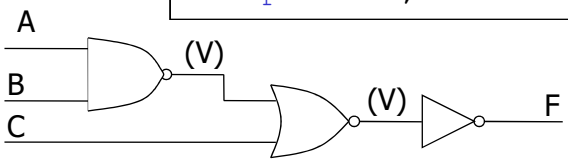


- En jargon VHDL, un délai delta = délai infinitésimal non nul
- Un signal prend sa nouvelle valeur après un délai delta.
- Une variable prend sa nouvelle valeur immédiatement.

Les Variables

- **Les variables ne peuvent être déclarées et n'existent que dans un process.**
- **L'affectation d'une variable est immédiate : la valeur affectée à V à la première ligne peut directement être réutilisée à la deuxième.**

```
process (A, B, C)
-- zone de déclaration d'une variable
   variable V: STD_LOGIC;
begin
   V := A nand B;
   V := V nor C;
   F <= not V;
end process;
```



Redessinez le schéma si on considère V comme un signal ?

Exemple : parité impaire

```
Entity parite is
PORT ( a   : IN std_logic_vector(0 TO 3) ;
       s   : OUT std_logic );
END enttity;
architecture behaviour of parite is
begin
   process(a)
      variable parite : std_logic ;
   begin
      parite := '1' ;
      FOR i in 0 to 3 LOOP
         if a(i) = '1' then
            parite := not parite;
         end if;
      END LOOP;
      s <= parite;
   end process;
END architecture;
```


Exercice

V est une variable qui prend sa valeur immédiatement.

Tandis que S et T prennent leur valeur à la fin du process après un delta délais.

Au démarrage, S= 0 et A passe de 0 à 1

A est dans la liste de sensibilité, donc le process s'exécute

```
process(A, S)
   variable V: STD_LOGIC;
begin
   V := A;
   S <= V;
   V := S;
   T <= V;
end process;
```

V=1

S=V=1 mais que à la fin du process

S vaut déjà 1

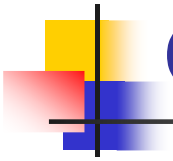
S vaut 0 donc V=0

V=1

T=0

T=V=1

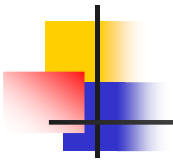
- Supposer que S vaut '0', et A change de l'état '0' à l'état '1'.

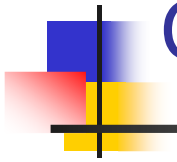


Questions

1. Qu'elle est la valeur de S à la fin du process, avant le delta délai ? S=0
2. Qu'elle est la valeur de V à la fin du process, avant le delta délai ? V=0
3. Après l'exécution du process et après le delta délai, S et T prennent leurs nouvelles valeurs, que valent S et T ? S=1 T=0
4. Après la première exécution du process et après le delta délai, que se passe-t-il ?

vue que S=1,S à changer d'état, le process se réexécute, T=1 après le delta délais.

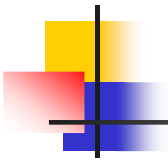




C7 – Les Types

Yann DOUZE

VHDL



Les Types énumérés

- Définition d'un nouveau type de donnée

```
type Opcode is (Add, Neg, Load, Store, Jmp, Halt);
signal S: Opcode;
```

```
S <= Add;
```

```
process(S)
begin
case S is
when Add =>
       ...
```

Synthèse des types énumérés

- Encodage du type énuméré pour la synthèse

	Binaire	One hot
Add	000	100000
Neg	001	010000
Load	010	001000
Store	011	000100
Jmp	100	000010
Halt	101	000001

ASIC FPGA

Les Types définit par défaut

```
library STD, WORK;
use STD.STANDARD.all;
```

Inclut par défaut

```
INTEGER     0  1  99
BIT         '0'  '1'
BIT_VECTOR  "101011"
BOOLEAN     FALSE  TRUE
STRING      "Hello World"
TIME        10 NS
REAL        1.345
```

Types du package STD.STANDARD

Les types logiques à valeurs multiple

Dans le package STD.STANDARD

```
type BOLEAN is (False, True);

type BIT is ('0', '1');
```

Dans le package IEEE.STD_LOGIC_1164

```
type STD_ULOGIC is (
'U',    -- non initialisé (par défaut)
'X',    -- état inconnu
'0',    -- 0 puissant
'1',    -- 1 puissant
'Z',    -- Haute impédance
'W',    -- État inconnu mais faible
'L',    -- 0 faible
'H',    -- 1 faible
'-');   -- indifférent (pour la synthèse)

subtype STD_LOGIC is RESOLVED STD_ULOGIC;
```

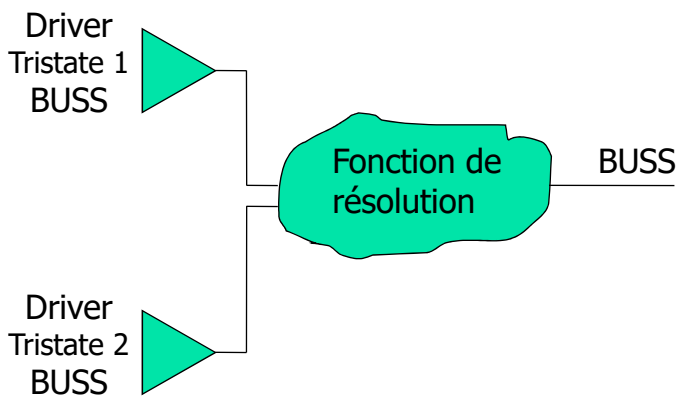
Résolution : définit la priorité

5

Drivers et Fonction de Résolution

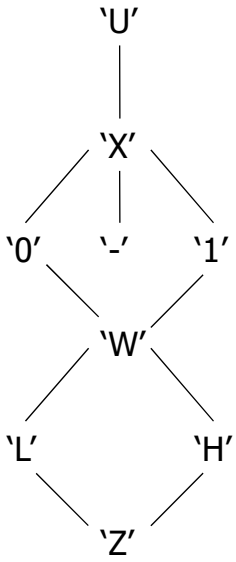
```
subtype STD_LOGIC is RESOLVED STD_ULOGIC;
```

```
Signal BUSS,ENB1,ENB2,D1,D2: STD_LOGIC;
...
TRISTATE1: process (ENB1,D1)
Begin
if ENB1 = '1' then
        BUSS <= D1;
else
        BUSS <= 'Z';
end if;
End process;

TRISTATE2: process (ENB2,D1)
Begin
   if ENB2 = '1' then
      BUSS <= D2;
   else
      BUSS <= 'Z';
   end if;
End process;
```



Résolution de STD_LOGIC

- Le type STD_LOGIC est définit dans le package STD_LOGIC_1164
- Les Valeurs les plus hautes dans le schéma sont prioritaires



Valeurs initiales

- Signaux et variables sont initialisés au début d'une simulation.
- La valeur par défaut est la valeur la plus à gauche du type.
- Attention : La synthèse ignore les valeurs initiales.

```
type Opcode is (Add, Neg, Load, Store, Jmp, Halt);
signal S: Opcode;
signal CLOCK, RESET: STD_LOGIC;
variable V1: STD_LOGIC_VECTOR(0 to 1);
variable V2: STD_LOGIC_VECTOR(0 to 1):="01";
signal N: Opcode:= Halt;
constant size: INTEGER := 16;
constant ZERO: STD_LOGIC_VECTOR := "0000";
```

Valeur initiale Add

Valeur initiale 'U'

Valeur initiale 'UU'

Relation implicite des opérateurs

- Pour le type STD_LOGIC

```
'U' < 'X' < '0' < '1' < 'Z' < 'w' < 'L' < 'H' < '-'
```

- Pour le type STD_LOGIC_VECTOR

```
'0' < "00" < "000" < "001" < "100" < "111" < "1111"
```


Opérations arithmétiques

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity ADDER is
  port ( A,B : in STD_LOGIC_VECTOR(7 downto 0);
         SUM : out STD_LOGIC_VECTOR(7 downto 0));
end entity;

architecture A1 of ADDER is
begin
  SUM <= A + B;
end architecture;
```

Erreur: "+" n'est pas définit pour le type STD_LOGIC_VECTOR

Utilisation de NUMERIC_STD

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

entity ADDER is
   port (     A,B : in UNSIGNED(7 downto 0);
              SUM : out UNSIGNED(7 downto 0);
end entity;

architecture A1 of ADDER is
begin
   SUM <= A + B;
end architecture;
```

IEEE Std 1076.3

Opérateur "+" est surchargé pour les types UNSIGNED et SIGNED

Problématique ?

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

entity ADDER is
   port ( A,B : in STD_LOGIC_VECTOR(7 downto 0);
          SUM : out STD_LOGIC_VECTOR(7 downto 0));
end entity;

architecture A1 of ADDER is
begin


end architecture;
```

Comment faire l'addition de A et B si ils sont de type std_logic_vector ?

Conversion de type

```
Signal U: UNSIGNED(7 downto 0);
Signal S: SIGNED (7 downto 0);
Signal V: STD_LOGIC_VECTOR(7 downto 0);
```

Conversion entre des types qui sont prochent

```
U <= UNSIGNED(S);
S <= SIGNED(U);
U <= UNSIGNED(V);
S <= SIGNED(V);
V <= STD_LOGIC_VECTOR(U);
V <= STD_LOGIC_VECTOR(S);
```

Le nom du type est utilisé pour la conversion de type

Solution

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

entity ADDER is
   port ( A,B : in STD_LOGIC_VECTOR(7 downto 0);
          SUM : out STD_LOGIC_VECTOR(7 downto 0));
end entity;

architecture A1 of ADDER is
begin
   SUM <= STD_LOGIC_VECTOR(UNSIGNED(A) + UNSIGNED(B));
   -- ou SUM <= STD_LOGIC_VECTOR(SIGNED(A) + SIGNED(B));
end architecture;
```

Fonctions de conversion

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;
```

```
Signal U: UNSIGNED(7 downto 0);
Signal S: SIGNED (7 downto 0);
Signal N: INTEGER;
```

Opérations de conversion

```
N <= TO_INTEGER(U);
N <= TO_INTEGER(S);
U <= TO_UNSIGNED(N,8);
S <= TO_SIGNED(N,8);
```

Taille du vecteur d'arrivée

Conversion entre un INTEGER et un STD_LOGIC_VECTOR

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;
```

```
Signal V: STD_LOGIC_VECTOR(7 downto 0);
Signal N: INTEGER;
```

Fonction de conversion

Conversion de type

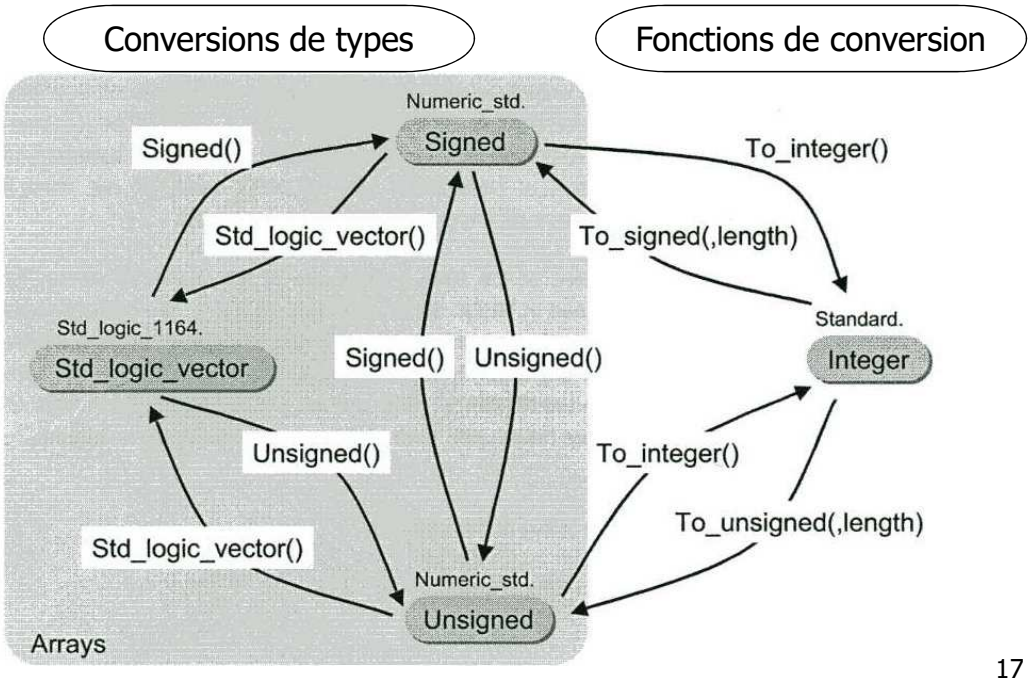
```
N <= TO_INTEGER(UNSIGNED(V));
V <= STD_LOGIC_VECTOR(TO_UNSIGNED(N,8));
```

Conversion de type

Fonction de conversion

```
V <= STD_LOGIC_VECTOR(SIGNED(V)+1);
```

Tableau Récapitulatif



Sommaire de NUMERIC_STD

```
+  -  *  /  rem  mod
<  <=  >  >=  = /=

UNSIGNED x UNSIGNED
UNSIGNED x NATURAL
NATURAL x UNSIGNED
SIGNED x SIGNED
SIGNED x INTEGER
INTEGER x SIGNED
```

```
sll  srl  rol  ror

UNSIGNED x UNSIGNED
SIGNED x INTEGER
```

```
not  and  or  nand nor xor
xnor

UNSIGNED x UNSIGNED
SIGNED x INTEGER
```

```
TO_INTEGER  [UNSIGNED             ] return INTEGER
TO_INTEGER  [SIGNED               ] return INTEGER
TO_UNSIGNED [NATURAL, NATURAL     ] return UNSIGNED
TO_SIGNED   [INTEGER, NATURAL     ] return SIGNED
RESIZE      [UNSIGNED,NATURAL     ] return UNSIGNED
RESIZE      [SIGNED,  NATURAL     ] return SIGNED
```

Signature (VHDL'93)

Exercice 1 (Addition de A,B et C)

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

Entity ADDER is
port( A      : in STD_LOGIC_VECTOR(7 downto 0);
      B      : in INTEGER;
      C      : in SIGNED(7 downto 0));
      SUM    : out STD_LOGIC_VECTOR(7 downto 0);
end entity;

Architecture BEHAVIOUR of ADDER is
Begin
SUM <=

End architecture;
```


Exercice 2 (Multiplexeur 8 vers 1)

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

entity Mux8to1 is
port( Address :   in STD_LOGIC_VECTOR(2 downto 0);
      IP      :   in STD_LOGIC_VECTOR(7 downto 0);
      OP      :   out STD_LOGIC);
end entity;
architecture BEHAVIOUR of Mux8to1 is
begin

OP <=IP(                              (Address)));

end architecture ;
```

Package STD_LOGIC_UNSIGNED/SIGNED

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
-- use IEEE.STD_LOGIC_SIGNED.all;
```

Ne pas utiliser en même temps

+ -	*	< <= > >= = /=
STD_LOGIC_VECTOR STD_ULOGIC INTEGER	STD_LOGIC_VECTOR	STD_LOGIC_VECTOR INTEGER

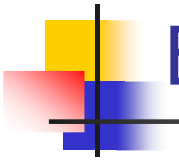
```
CONV_INTEGER[STD_LOGIC_VECTOR] return INTEGER
```


Package STD_LOGIC_ARITH

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
```

+ -	*	< <= > >= = /=
STD_ULOGIC UNSIGNED SIGNED INTEGER	UNSIGNED SIGNED	UNSIGNED SIGNED INTEGER

```
CONV_INTEGER[INTEGER/UNSIGNED/SIGNED/STD_ULOGIC] return INTEGER
CONV_UNSIGNED[INTEGER/UNSIGNED/SIGNED/STD_ULOGIC, INTEGER] return UNSIGNED
CONV_SIGNED[INTEGER/UNSIGNED/SIGNED/STD_ULOGIC, INTEGER] return SIGNED
CONV_STD_LOGIC_VECTOR[INTEGER/UNSIGNED/SIGNED/STD_ULOGIC, INTEGER] return
STD_LOGIC_VECTOR
EXT[STD_LOGIC_VECTOR, INTEGER] return STD_LOGIC_VECTOR
SXT[STD_LOGIC_VECTOR, INTEGER] return STD_LOGIC_VECTOR
```



Exercice

- Faire l'exercice du C7

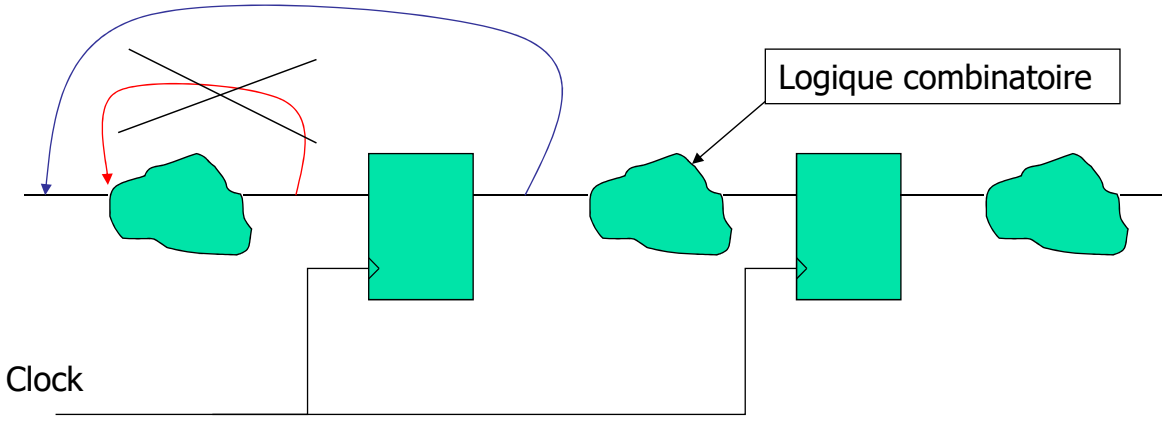
C8 – Les Process synchrones

Yann DOUZE

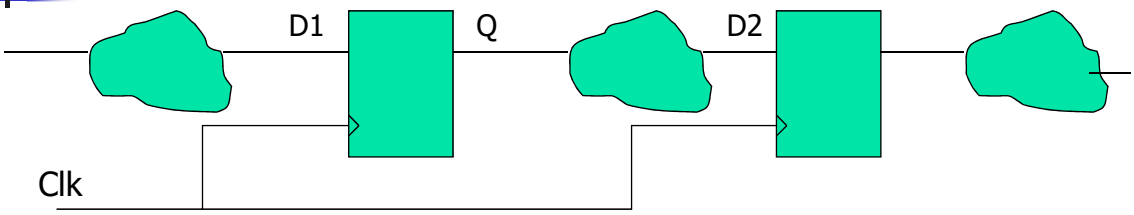
VHDL

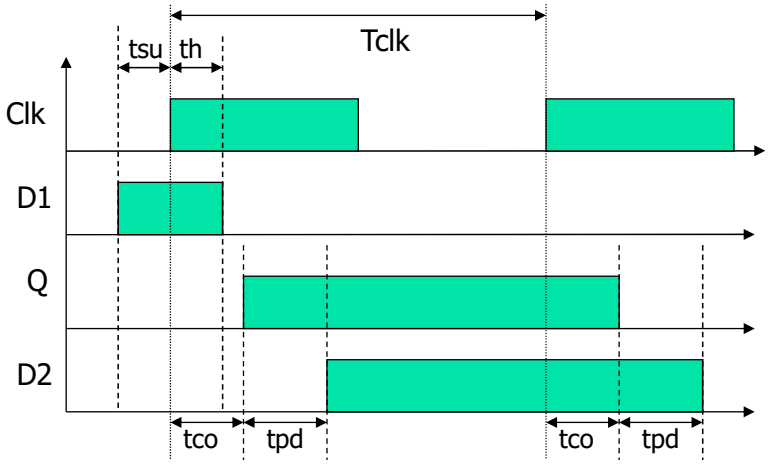
Design Synchrone

- Tous les registres se font dans des bascules D flip-flops avec une seule horloge externe.
- Pas de rebouclage sur le combinatoire.



Contraintes de temps





Tclk: période de l'horloge

tsu: time setup

th: time hold

tco: clock to output

tpd: propagation delay

Tclk > tco + tpd + tsu

Fmax = 1 / (tco + tsu + tpd)

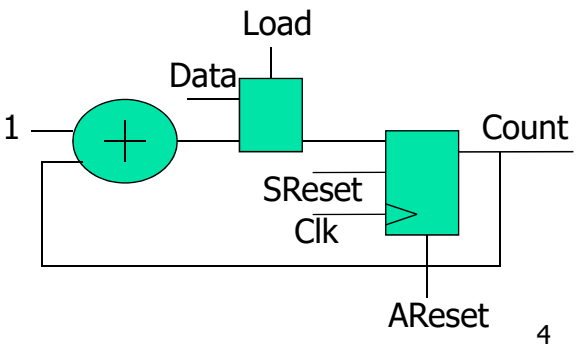
Actions Synchrones et Asynchrones

```
signal Count : unsigned(7 downto 0);
process (Clk, AReset)
begin
  if AReset = '1' then
    Count <= "00000000";
  elsif RISING_EDGE(Clk) then
    if  SReset = '1' then
      Count <= "00000000";
    elsif Load = '1' then
      Count <= UNSIGNED(Data);
    else
      Count <= Count + '1';
    end if;
  end if;
end process;
```

Reset Asynchrone

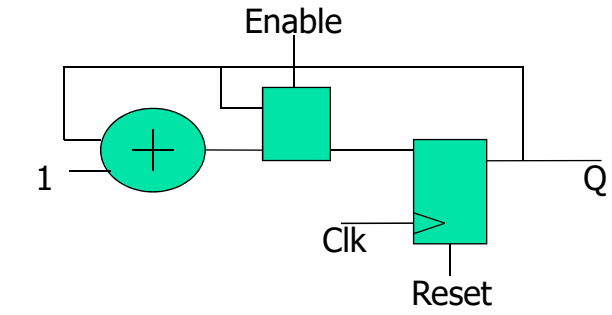
Reset Synchrone

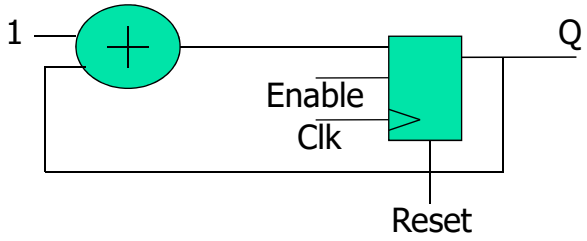
Load Synchrone



Clock Enables

```
process (Clk, Reset)
begin
  if Reset = '1' then
    Q <= "00000000";
  elsif RISING_EDGE(Clk) then
    if  Enable = '1' then
      Q <= Q + 1;
    end if;
  end if;
end process;
```



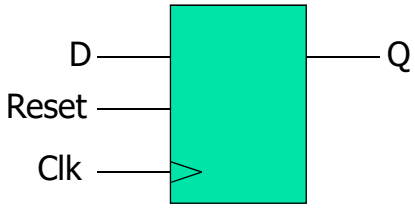


Style de code légal utilisant Wait Until

```
-- Process sans liste de sensibilité

Process
begin
  wait until Clock = '1';
  if Reset = '1' then
    Q <= '0';
  else
    Q <= D;
  end if;
end process;
```

Détection du front montant

Reset Synchrone



Style de code légal utilisant 'EVENT

- S'EVENT est vrai si est seulement si il y a un évènement sur S

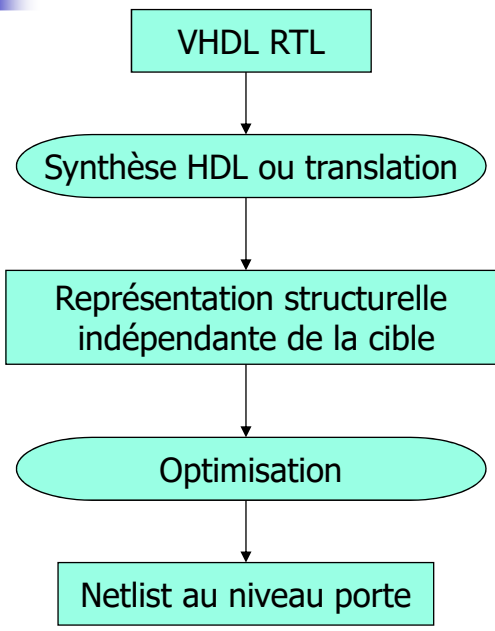
```
process(Clk)
begin
   if Clk'EVENT and Clk = '1' then
      Q <= D;
   end if;
end process;
```

```
process
begin
   wait until Clk'EVENT and Clk = '1';
      ...
end process;
```


Détection de fronts

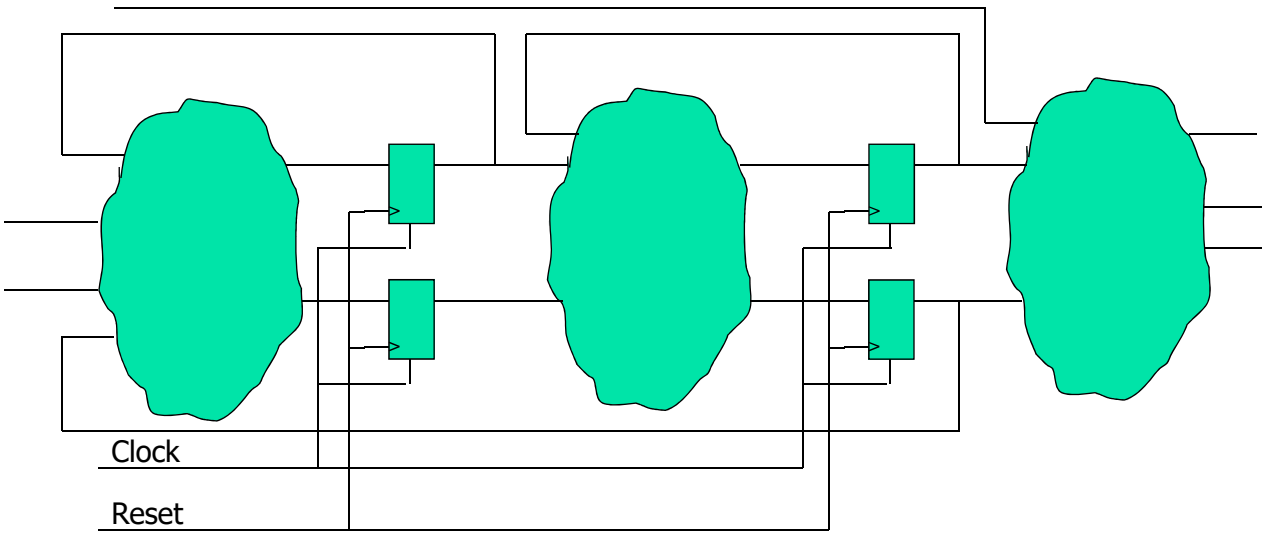
- Les opérateurs de détection de fronts (Rising_edge(clk) et Clk'event and clk='1') ne doivent être utilisé que pour tester le front d'une horloge (clk).
- Chaque fois que l'on fait un test de front, l'outil de synthèse comprend qu'il s'agit d'une horloge.
- Horloge = le signal le plus rapide du circuit.

Fonctionnement des outils de synthèse



- Pas d'optimisation
- Structure inféré par le code VHDL
- Optimisation de la logique combinatoire et des contraintes de temps entre registre
- Dépend de la cible

Synthèse RTL



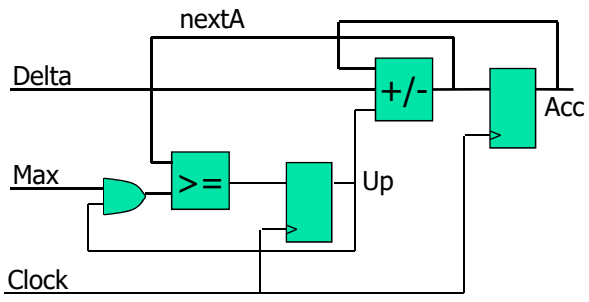
- La synthèse RTL n'ajoute, ne supprime ou ne bouge pas de registre.
- La synthèse RTL optimise uniquement la logique combinatoire.

Règles : Les registres à la synthèse

- Les outils de synthèse RTL infèrent les registres directement à partir du code VHDL suivant certaine règle élémentaire :
  1. Des **registres** sont **inférés** à la **synthèse** que dans les **process synchrone.**
  2. **Signaux:** Tout les signaux assignés dans un **process synchrone** sont synthétisés par des registres.
  3. **Variables** : les variables assignés dans un **process synchrone** peuvent être synthétisées soit par un fil, soit par un registre.
     - Les variables sur lesquelles sont assignées une nouvelle valeur avant d'être lues sont synthétisées par un fil. (**assigné avant d'être lu => fil)**
     - Les variables qui sont lues avant d'être assignées sont synthétisées par un registre. (**lu avant d'être assigné => registre**)

Les registres à la synthèse

```
signal Acc, Delta, Max: signed(11 downto 0);
process (Clock)
   variable Up : std_logic;
   variable nextA : signed(11 downto 0);
begin
   if RISING_EDGE(Clock) then
      if Up = '1' then
         nextA := Acc + Delta;
         if nextA >= Max then
            Up := '0';
         end if;
      else
         nextA := Acc - Delta;
         if nextA < 0 then
            Up := '1';
         end if;
      end if;
      Acc <= nextA;
   end if;
end process;
```



Exercice 1

```
Signal INPUT : std_logic_vector(7 downto 0)
Signal REG : std_logic;
AND-REG : process (CLOCK)
  variable V : STD_LOGIC;
begin
  if RISING_EDGE(CLOCK) then
    V := '1';
    for I in 0 to 7 loop
      V := V and INPUT(I);
    end loop;
    REG <= V;
  end if;
end process;
```

1 DFF

Combien de bascule D flip-flops ?

1 Flip-flops

Exercice 2

```
Signal OUTPUT : std_logic;
COUNTER : process (CLOCK)
  variable COUNT : UNSIGNED(7 downto 0);
begin
  if RISING_EDGE(CLOCK) then
    if RESET = '1' then
      COUNT := "00000000";
    else
      COUNT := COUNT + 1;
    end if;
    OUTPUT <= COUNT(7);
  end if;
end process;
```

先读再写 8 DFF

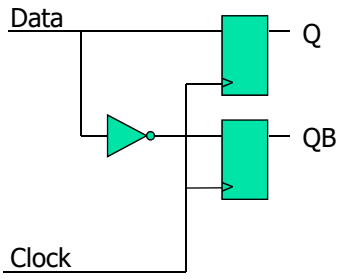
1 DFF

Combien de flip-flops ?

9 Flip-flops

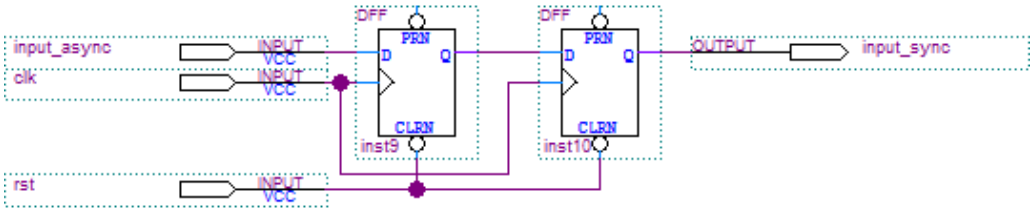
La synthèse n'optimise pas les registres !

```
process (CLOCK)
begin
  if RISING_EDGE(CLOCK) then
    Q <= Data;
    QB <= not Data;
  end if;
end process;
```



Comment faut il réécrire le code pour s'assurer qu'il n'y est qu'une seule bascule D ?

Exercice : Synchronisation des entrées



- Avantage : sûreté de fonctionnement
- Inconvénient : introduit du délai (pipeline)

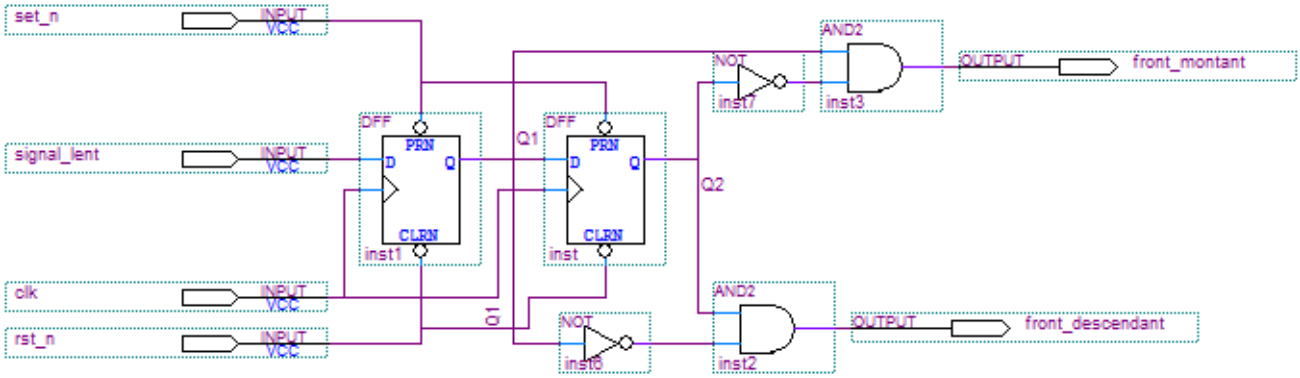
Code : synchronisation des entrées

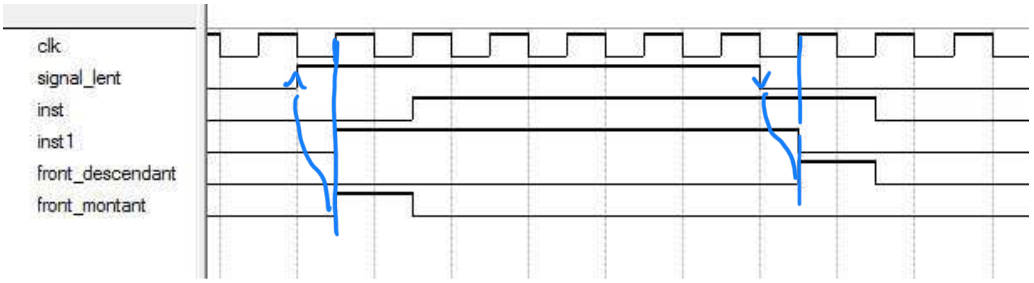
```
Entity resynchro is
port(
      clk, rst, input_async:        in       std_logic;
      input_sync:                   out      std_logic);
end entity;
architecture RTL of resynchro is
Signal Q : std_logic;
Begin
      Process (clk,rst)
      begin
            if (rst='1') then
                  Q <= '0'; input_sync <= '0';
            elsif rising_edge(clk) then
                  Q<= input_async;
                  input_sync <= Q;
            end if;
      end process;
```

DFF

DFF

Exercice : Détection des fronts d'un signal lent





Détection de front lent : code

```
Entity detect_fronts is
port(
    clk, rst, signal_lent : in     std_logic;
    fm, fd: out std_logic);
end entity;
architecture RTL of detect_fronts is
Signal Q1,Q2 : std_logic;
begin
PROCESS ( clk , rst)
BEGIN
if rst ='1' then
            Q1 <= '0'; Q2 <= '0';
elsif rising_edge (clk) then
            Q1 <= signal_lent;
            Q2 <= Q1;
end if;
end process;
Fm <= Q1 and (not Q2);
Fd <= Q2 and (not Q1);
end architecture;
```


Exemple d'utilisation : compteur d'évenements

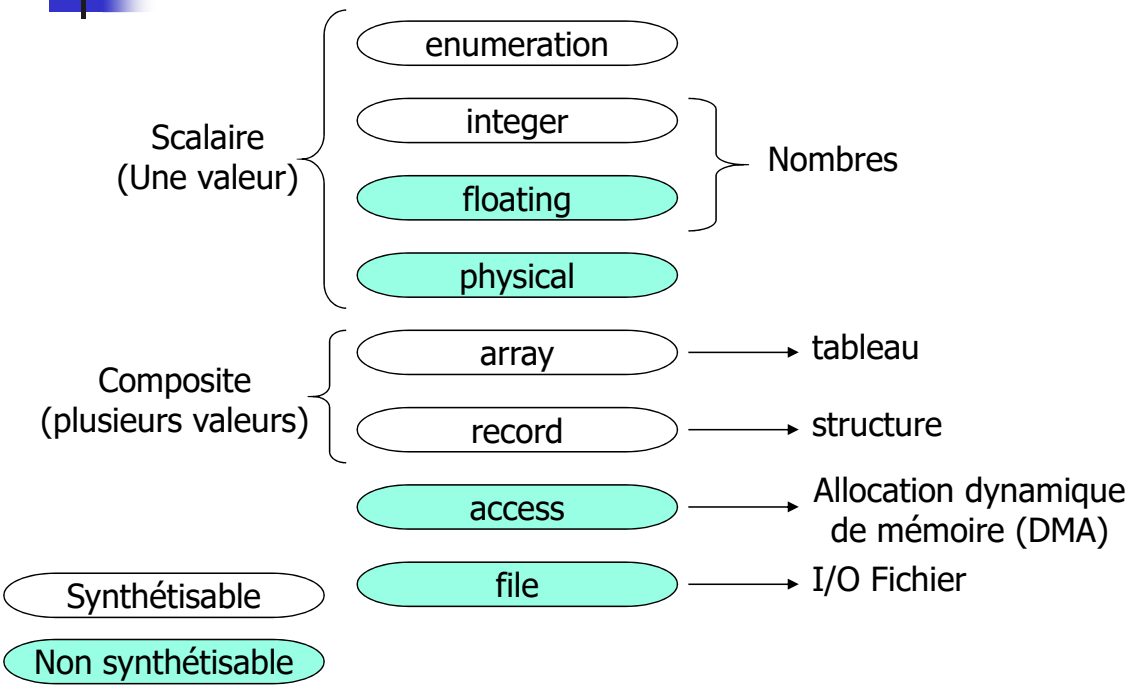
```
entity cnt_evt is
port(clk, rst, sig_lent : std_logic;
            cnt : std_logic_vector(7 downto 0));
end entity;
architecture RTL of cnt_evt is
    siignal Q : unsigned (7 downto 0);
    signal fm : std_logic;
Begin
U1 : entity work.detect_front port map (
    clk => clk, rst => rst, signal_lent=>sig_lent, fm => fm);
Process (clk,rst)
    begin
        if (rst='1') then
            Q <= (others => '0');
        elsif rising_edge(clk) then
            if (fm='1') then
                Q<= Q +'1';
            end if;
        end if;
    end process;
    cnt <= std_logic_vector(Q);
End architecture;
```

C9 – Sous-Types (subtype)

Yann DOUZE

VHDL

Les types de donnés en VHDL

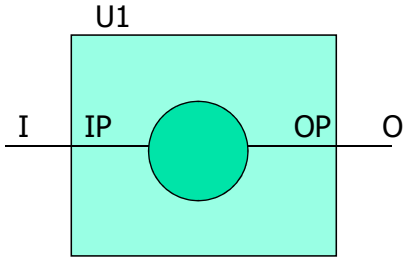


Types de données et Packages

```
package MY_TYPES is
   type CODE is (A, B, C, D, E);
end package MY_TYPES;
```

Dans un fichier séparé

```
use WORK.MY_TYPES.all;
entity FILTER is
   port(IP: in CODE;
        OP : out CODE);
end entity FILTER;
```




```
use WORK.MY_TYPES.all;
   ...
signal I, O : CODE;
   ...
U1: entity work.FILTER port map (IP => I, OP => O);
```


Sous-types du type entier

```
type INTEGER is range -2**31+1 to 2**31-1;
subtype NATURAL is INTEGER range 0 to 2**31-1;
subtype POSITIVE is INTEGER range 1 to 2**31-1;
```

Définit dans le package STD.STANDARD

```
subtype SHORT is INTEGER range -128 to +127;
subtype LONG is INTEGER range -2**15 to 2**15-1;
signal S: SHORT;
signal L: LONG;
```

Sous-types définit par l'utilisateur

```
variable I: INTEGER range 0 to 255;
```

Sous-type anonyme

```
I := -1;
S <= L;
```

Ces assignements sont ils erronés?

Synthèse des sous-types entier

```
subtype Byte is INTEGER range 0 to 255;
signal B: Byte;
```

8 bits, non signé

```
subtype Int8 is INTEGER range -128 to +127;
signal I: Int8;
```

8 bits, signé complément à 2

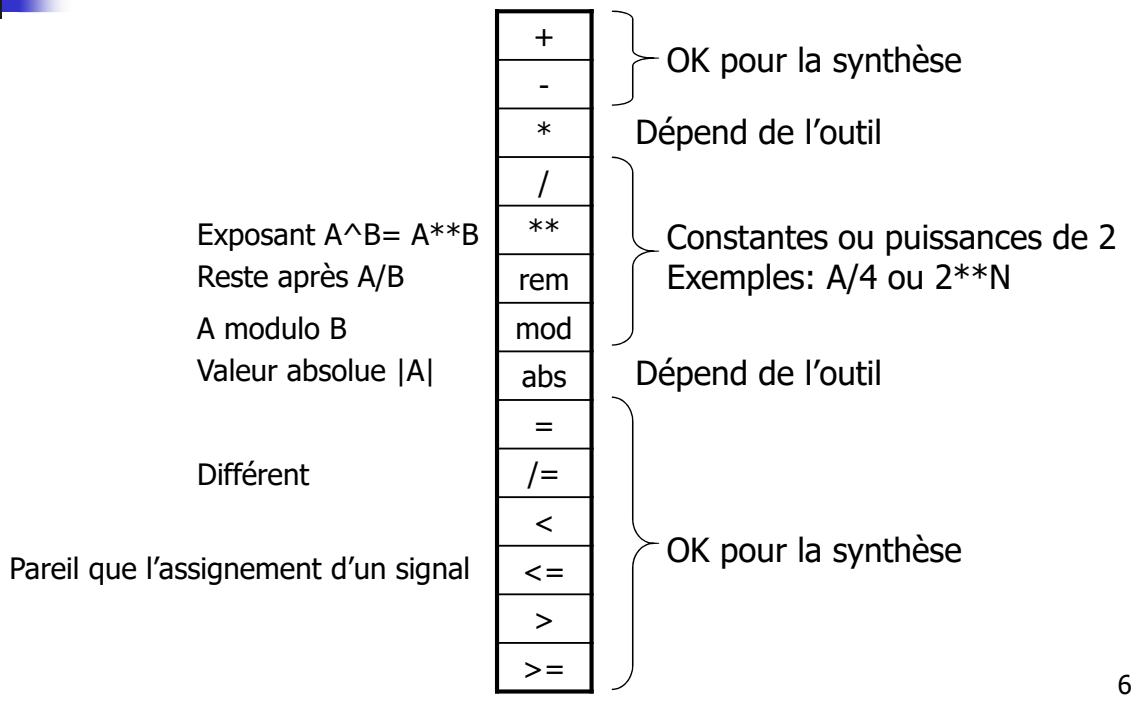
```
subtype Silly is INTEGER range 1000 to 1001;
signal S: Silly;
```

10 bits, non signé

```
signal J: INTEGER;
```

32 bits, complément à 2, attention!

Opérateurs Arithmétique



Représentation des valeurs entières

Nombres entiers

```
0     99    1e6   1E6   1000000    1_000_000
```

_ ignoré

Ecrire un STD_LOGIC_VECTOR en binaire, octal et hexadécimal

```
B"0000_1111"    O"017"    X"0F"
```

VHDL 1993

Sous-types d'un tableau (array)

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

package BusType is
   subtype DataBus is STD_LOGIC_VECTOR(7 downto 0);
end package Bustype;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
Use WORK.BusType.all;

Entity CNTL is
   port( CLK   : in     STD_LOGIC;
         D     : in     DataBus;
         Sin   : in     STD_LOGIC_VECTOR(3 downto 0);
         Q     : out    DataBus;
         Sout  : out    STD_LOGIC_VECTOR(1 downto 0));
End entity CNTL;
```

Les types tableaux (array)

Type tableau non contraint

```
type STD_LOGIC_VECTOR is array (NATURAL range <>) of STD_LOGIC;
```

Type de l'index

Type de l'élément

```
subtype Byte is STD_LOGIC_VECTOR(7 downto 0);
```

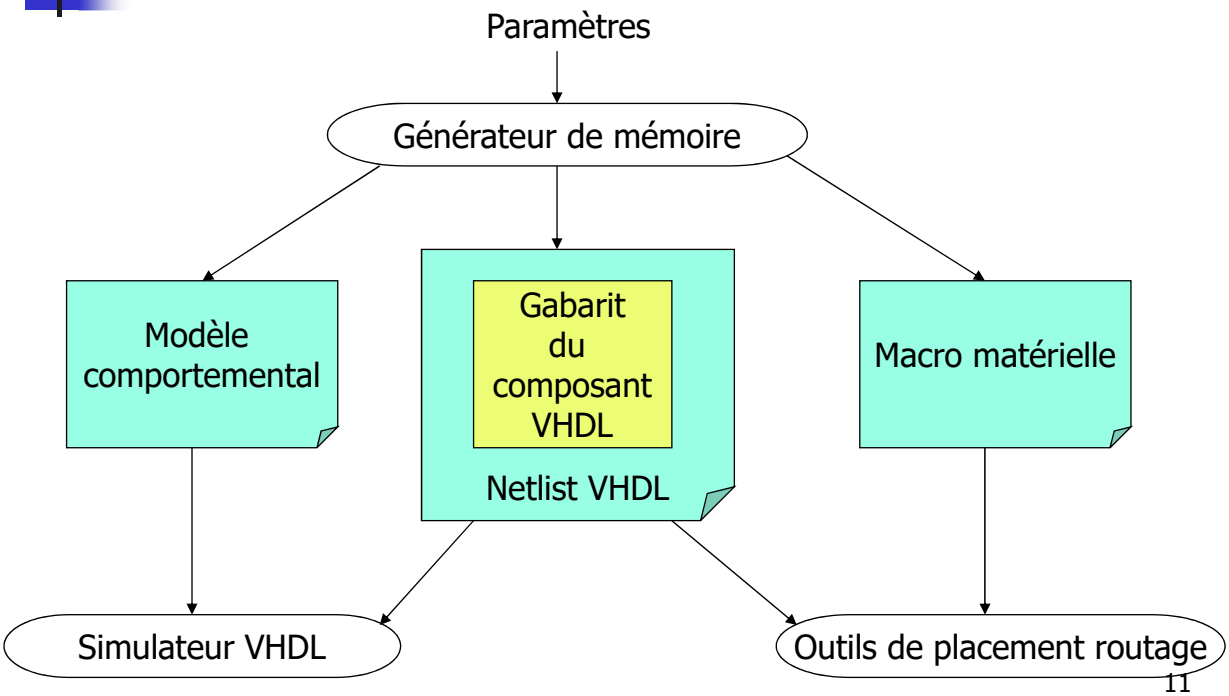
```
type RAM1Kx8 is array (0 to 1023) of Byte;
variable RAM: RAM1Kx8;
```

Type tableau contraint

Modélisation des mémoires

```
entity DualPortRam is
   port ( Clock, Wr, Rd   : in    Std_logic;
          AddrWr, AddrRd : in    Std_logic_vector(3 downto 0);
          DataWr         : in    Std_logic_vector(7 downto 0);
          DataRd         : out   Std_logic_vector(7 downto 0));
end entity;

architecture modele of DualPortRam is
    type RamType is   array (0 to 15) of Std_logic_vector(7 downto 0);
    signal RAM : RamType;
begin
   process (Clock)
   begin
      if RISING_EDGE(Clock) then
         if Wr = '1' then
            Ram(To_integer(Unsigned(AddrWr))) <= DataWr;
         end if;
         if Rd = '1' then
            DataRd <= Ram(To_integer(Unsigned(AddrRd)));
         end if;
      end if;
   end process;
end architecture;
```

Instanciation de mémoire



Attribut 'RANGE

```
Signal A: STD_LOGIC_VECTOR(...);
```

```
process(A)
   variable V: STD_LOGIC;
begin
   V := '0';

   for I in 0 to 7 loop
      V := V xor A(I);
   end loop;

   for I in A'RANGE loop
      V := V xor A(I);
   end loop;
   ...
end process
```

Pas générique

Bien

Attributs de type et tableau

```
signal A: STD_LOGIC_VECTOR(7 downto 0);
subtype SHORT is INTEGER range 0 to 15;
type MODE is (W, X, Y, Z);
```

- Attributs de tableau (à utiliser dès que possible)

```
A'LOW   = 0
A'HIGH  = 7
A'LEFT  = 7
A'RIGHT = 0
```

```
A'RANGE = 7 downto 0
A'REVERSE_RANGE = 0 to 7
A'LENGHT = 8
```

- Attributs de type (à éviter en synthèse)

```
SHORT'LOW   = 0
SHORT'HIGH  = 15
SHORT'LEFT  = 0
SHORT'RIGHT = 15
```

```
MODE'LOW   = W
MODE'HIGH  = Z
MODE'LEFT  = W
MODE'RIGHT = Z
```


Agrégats

```
Type BCD6 is array (5 downto 0) of STD_LOGIC_VECTOR(3 downto 0);

Variable V: BCD6 := ("1001","1000","0111","0110","0101","0100");
```

```
V := ("1001", "1000", others => "0000");

V := (3 => "0110", 1 => "1001", others => "0000");

V := (others => "0000");
```

```
Variable A: STD_LOGIC_VECTOR(3 downto 0);
```

```
A := (others => '1');
```

Types ambigus

```
port (A,B: in STD_LOGIC);
```

```
process(A, B)
begin
   case A & B is
   when "00" = > ...
```

Illégal!

```
library IEEE;
use IEEE.NUMERIC_STD.all
```

```
variable N: INTEGER;
```

Illégal!

```
N := TO_INTEGER("1111");
```


Expressions de qualification

```
process(A, B)
   subtype T is STD_LOGIC_VECTOR(0 to 1);
begin
   case T'(A & B) is
   when "00" = > ...
```

```
N := TO_INTEGER(UNSIGNED'("1111"));
```

N = 15

```
N := TO_INTEGER(SIGNED'("1111"));
```

N = -1

C10 – Generic, generate

Yann DOUZE

Port map

```
-- entité du composant COUNTER
entity COUNTER is
  port ( CLK, RST : in     Std_logic;
         UpDn     : in     Std_logic :='0';
         Q        : out    Std_logic_vector(2 downto 0));
end entity;
--Architecture STRUCT d'un composant BLOK instanciant COUNTER
architecture STRUCT of BLOK is
begin
  -- association par position
  G1 : entity work.COUNTER port map (Clk32MHz, RST, open, Count);
  -- association par nom
  G2 : entity work.COUNTER port map( RST  => RST,
                                     CLK  => Clk32MHz,
                                     Q(2) => Q1MHz,
                                     Q(1) => Q2MHz,
                                     Q(0) => Q4MHz);
end architecture;
```

Valeur par défaut

Non connecté

VHDL 93

Compteur 8 bits

```
entity COUNTER8BIT is
  port ( CLK, RST : in     Std_logic;
         Q        : out    Std_logic_vector( 7 downto 0));
end entity;
architecture RTL of COUNTER8BIT is
  signal CNT: unsigned( 7 downto 0);
begin
  process (CLK,RST)
  begin
    if RST = '1' then
      CNT <= "00000000";
    elsif rising_edge(CLK) then
      CNT <= CNT + '1';
    end if;
  end process;
  Q <= std_logic_vector(CNT);
end architecture;
```


Compteur génériques

```
entity COUNTER is
generic(N : integer:=8);
port ( CLK, RST : in     Std_logic;
         Q        : out    Std_logic_vector(N-1 downto 0));
end entity;
architecture RTL of COUNTER is
  signal CNT: unsigned(N-1 downto 0);
begin
  process (CLK,RST)
  begin
    if RST = '1' then
      CNT <= (others => '0');
    elsif rising_edge(CLK) then
      CNT <= CNT + '1';
    end if;
  end process;
  Q <= std_logic_vector(CNT);
end architecture;
```

Instanciation d'un composant générique

```
-- l'entité du composant COUNTER
entity COUNTER is
  generic(N : integer:=8);
  port (CLK, RST : in  Std_logic;
        Q        : out   Std_logic_vector(N-1 downto 0));
end entity;

-- Utilisé dans l'architecture STRUCT d'un composant BLOK
architecture STRUCT of BLOK is
  signal Count4: std_logic_vector(3 downto 0);
  signal Count6: std-logic_vector(5 downto 0);
begin
-- association par position
  U1: entity work.COUNTER generic map(4)
  port map(CLK , RST, Count4);
-- association par nom
  U2: entity work.COUNTER generic map (N => 6)
  port map (CLK => CLK, RST => RST, Q => Count6);
end architecture;
```


Les délais génériques

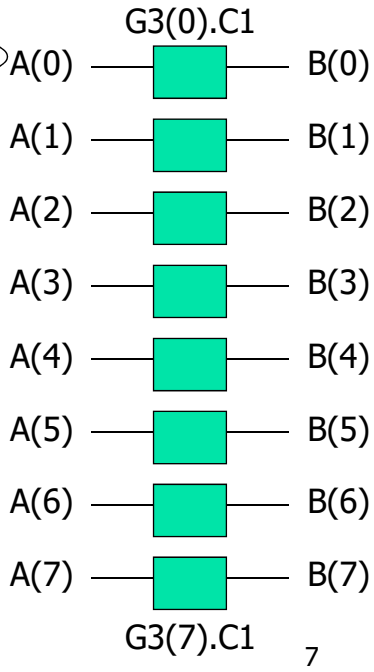
```
-- Entité du composant NAND2
entity NAND2 is
  generic (TPLH,TPHL: TIME := 0 NS);
  port ( A, B : in     Std_logic;
         F    : out    Std_logic);
end entity;

-- Architecture STRUCT du composant BLOK
architecture STRUCT of BLOK is
  signal N1,N2,N3,N4,N5,N6,N7,N8,N9 : Std_logic;
begin
  G1: entity work.NAND2 generic map (1.9 NS, 2.8 NS)
      port map (N1, N2, N3);
  G2: entity work.NAND2 generic map (TPLH => 2 NS, TPHL => 3 NS)
      port map (A => N4, B => N5, F => N6);
  G3: entity work.NAND2 port map (A => N7, B => N8, F => N9);
end architecture;
```

Instruction generate

```
architecture A1 of BLOK is
begin
 G1: for I in SOME_RANGE generate
     -- Instanciation de composant ou process
 end generate;

 G2: if CONDITION generate
     -- Instanciation de composant ou process
 end generate;

--Exemple :
 G3: for I in 0 to 7 generate
  C1: entity work.COMP port map (D=>A(I),Q=>B(I));
 end generate;

end architecture;
```

Structure régulière

Structure optionnelle



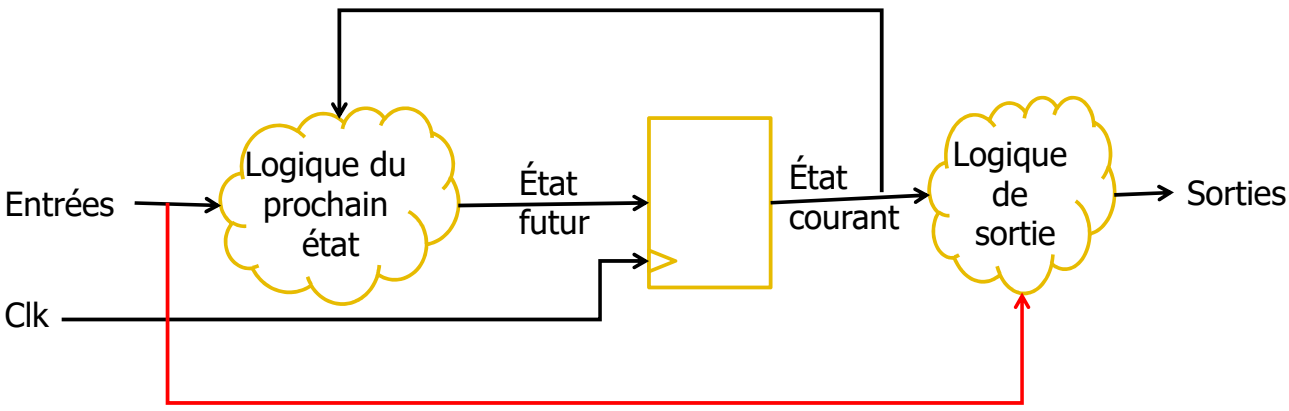
Additionneur structurel générique

```
entity ADDN is
  generic(N: positive :=4);
  port(  Cin : in std_logic;
         A,B: in std_logic_vector(N-1 downto 0);
         Cout : out std_logic;
         SUM: out std_logic_vector(N-1 downto 0));
end entity;
architecture STRUCT of ADDN is
signal C : std_logic_vector(N downto 0);
begin
  C(0) <= Cin;
  L1: for I in A'reverse_range generate
    U1 : entity work.ADDC1 port map(
    Cin => C(I), A => A(I), B => B(I),
    SUM => SUM(I), Cout => C(I+1));
  end generate;
  Cout <= C(N);
end architecture;
```

C11 MAE / FSM

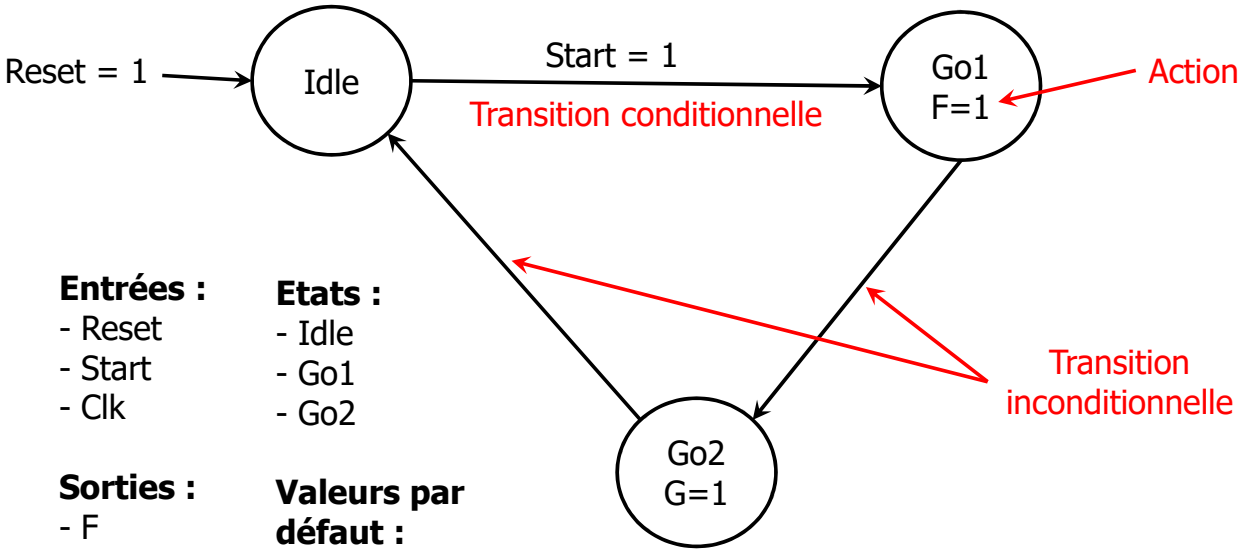
Yann DOUZE

Machine de Moore et Mealy



- Machine de Moore→ les sorties ne dépendent que de l'état courant.
- Machine de Mealy → les sorties dépendent de l'état courant et des entrées.

Diagramme d'état



Entrées :
- Reset
- Start
- Clk

Sorties :
- F
- G

Etats :
- Idle
- Go1
- Go2

Valeurs par défaut :
- F = 0
- G = 0

Description VHDL : Machine de Moore (à éviter, moins performante)

```
architecture FSM of EXEMPLE is
  type StateType is (Idle, Go1, Go2);
  Signal State : StateType;
begin
process(Clk, Reset)
  if Reset = '1' then
    State <= Idle;
  elsif Rising_edge(Clk) then
    case State is
    when Idle =>
      if Start = '1' then
        State <= Go1;
      end if;
    when Go1 =>
      State <= Go2;
    when Go2 =>
      State <= Idle;
    end case;
  end if;
end process;
```

```
output : process(State)
begin
  F <= '0';
  G <= '0';
  if State = Go1 then
    F <= '1';
  elsif State = Go2 then
    G <= '1';
  end if;
end process;
end architecture;
```

Description VHDL : Machine de Mealy

```
architecture FSM of EXEMPLE is
  type StateType is (Idle, Go1, Go2);
  Signal State : StateType;
begin
process(Clk, Reset)
begin
  if Reset = '1' then
    State <= Idle;
    G <= '0';
    F <= '0';
  elsif Rising_edge(Clk) then
    case State is
    when Idle =>
      if Start = '1' then
        State <= Go1;
        F <= '1';
      end if;
    when Go1 =>
      State <= Go2;
      G <= '1';
      F <= '0';
    when Go2 =>
      State <= Idle;
      G <= '0';
    end case;
  end if;
end process;
end architecture;
```


Codeur AMI (Alternate Mark Inversion)

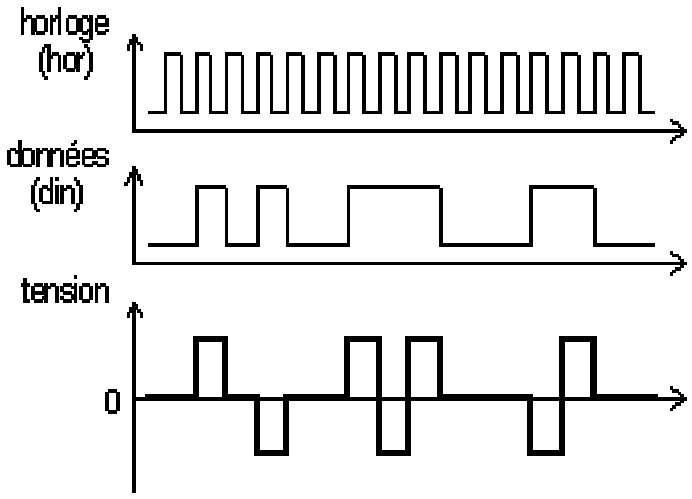
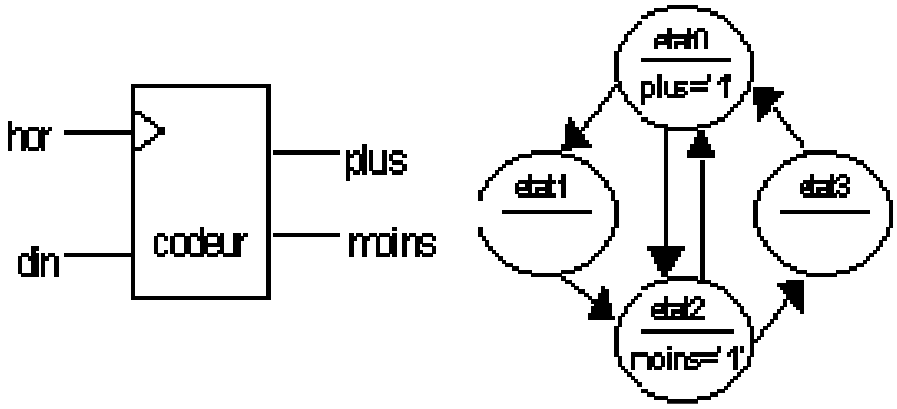


Diagramme d'état du codeur AMI



Code VHDL du codeur AMI

```
architecture MAE of AMI is
   type StateType is (E0, E1, E2, E3);
   Signal State : StateType;
begin
process(Clk, Reset)
begin
   if Reset = '1' then
      State <= E3;
      plus <= '0';
      moins <= '0';
   elsif Rising_edge(Clk) then
      case State is
      when E0 =>if Din = '0' then
                  State <= E1;
                  plus <= '0';
               elsif Din = '1' then
                  State <= E2;
                  moins <= '1';
                  plus <= '0';
               end if;
      when E1 =>if Din = '1' then
                  State <= E2;
                  moins <= '1';
                  plus <= '0';
               end if;
      when E2 =>if Din = '1' then
                  State <= E0;
                  plus <= '1';
                  moins <= '0';
               elsif Din = '0' then
                  State <= E3;
                  plus <= '0';
                  moins <= '0';
               end if;
      when E3 =>if Din = '1' then
                  State <= E0;
                  plus <= '1';
                  moins <= '0';
               end if;
      end case;
   end if;
end process;
end architecture;
```