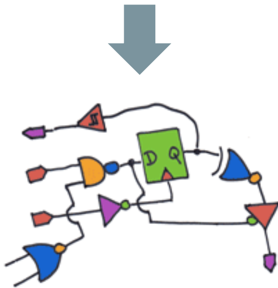


INITIATION AU LANGAGE VHDL

Ahmed AOUCHAR

L'École des INGÉNIEURS Scientifiques

1. INTRODUCTION

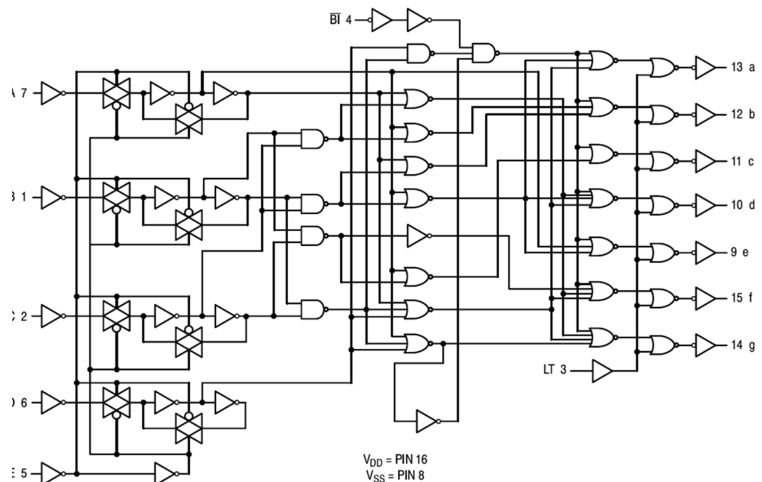
1.1. PRÉSENTATION

- VHDL sont les initiales de « VHSIC Hardware Description Language »
- Il a été développé en 1981 par le département de défense américaine pour décrire du « hardware »
- Il a été standardisé en 1987 par l'IEEE et mis à jour régulièrement
- VHDL a été d'abord adopté pour la simulation ensuite la synthèse logique
- C'est un langage typé comme le C, il faut déclarer le type de chaque signal manipulé
- La syntaxe ressemble à celle du C mais il ne faut pas tomber dans le piège, il faut toujours avoir à l'esprit que l'on décrit du matériel « hardware »

1.2. LA SYNTHÈSE LOGIQUE

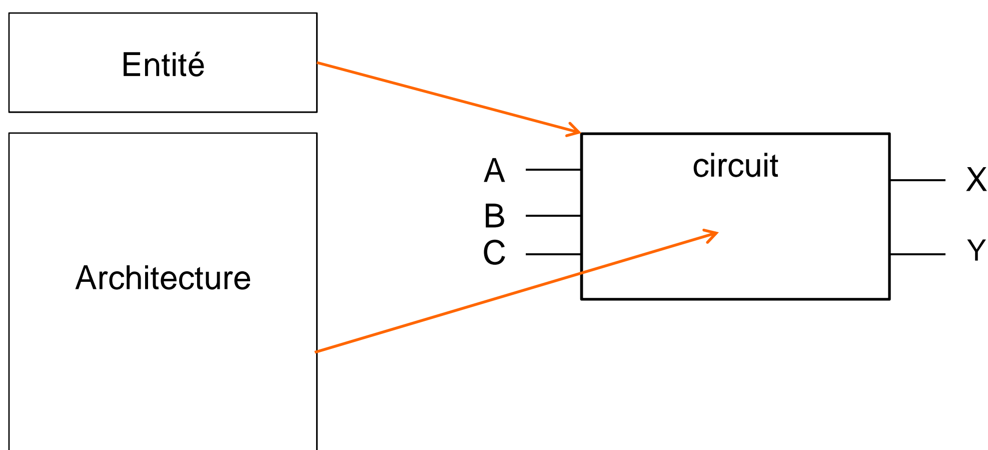
- L'art de passer du code à des portes logiques :

```
architecture Behavioral of hex_7seg is
begin
   with hex select
   seg <=
--------------------------
--     seg              hex
--------------------------
   "1000000" when "0000" ,
   "1111001" when "0001" ,
   "0100100" when "0010" ,
   "0110000" when "0011" ,
   "0011001" when "0100" ,
   "0010010" when "0101" ,
   "0000010" when "0110" ,
   "1111000" when "0111" ,
   "0000000" when "1000" ,
   "0010000" when "1001" ,
   "0001000" when "1010" ,
   "0000011" when "1011" ,
   "1000110" when "1100" ,
   "0100001" when "1101" ,
   "0000110" when "1110" ,
   "0001110" when others;
end Behavioral;
```



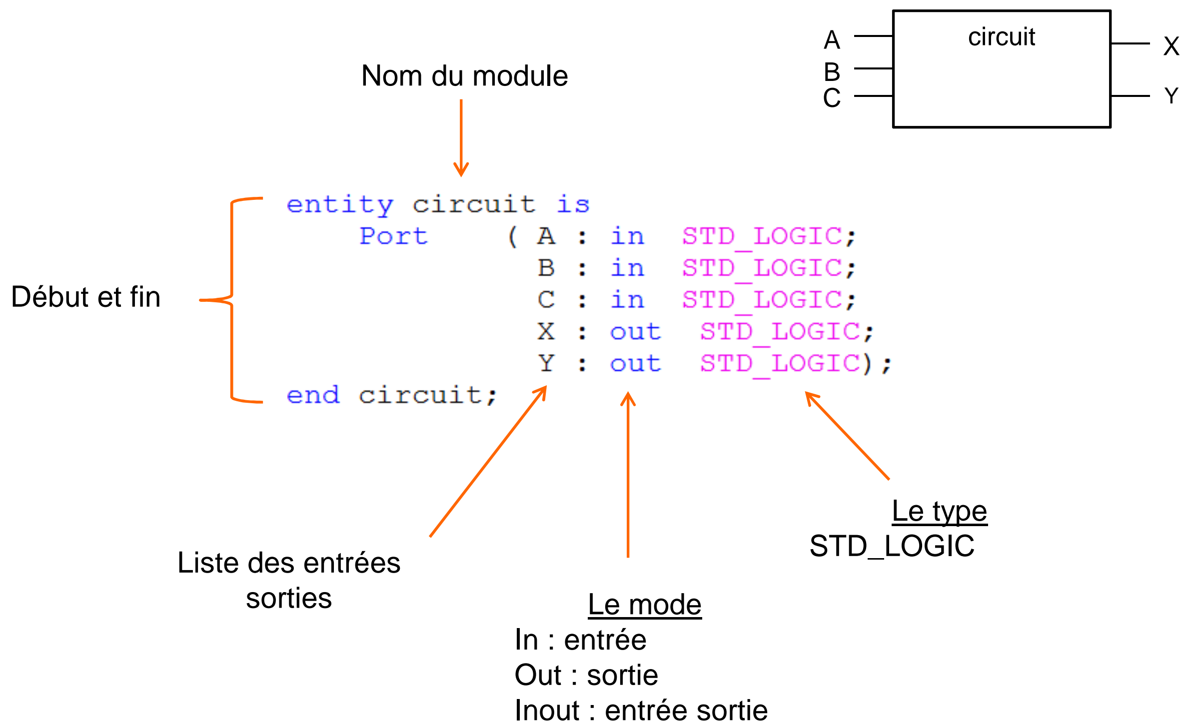
2. RÈGLES D'ÉCRITURE

2.1. STRUCTURE D'UN PROGRAMME VHDL



- L'entité décrit la façade du module « circuit » : une vue externe
- L'architecture détaille le fonctionnement interne : c'est la partie la plus délicate à coder

2.2. L'ENTITÉ



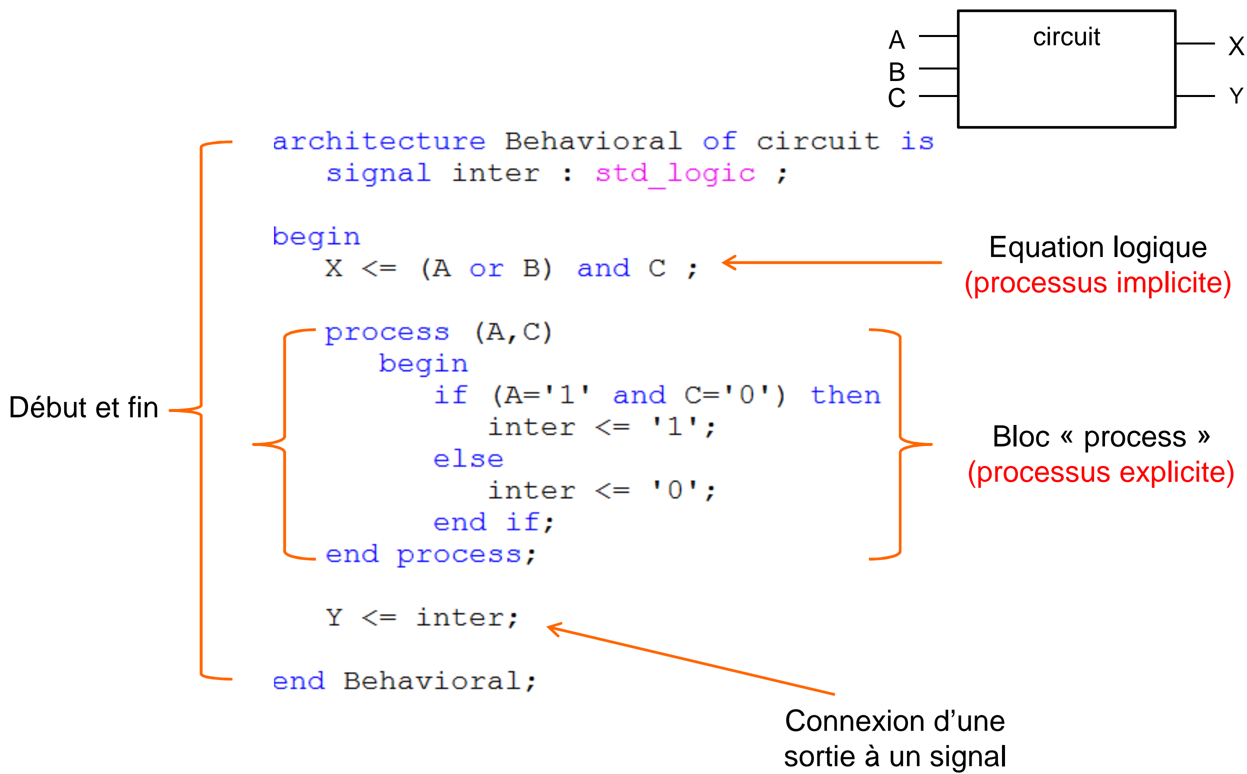
2.3. LE TYPE « STD_LOGIC »

- Désigne un signal logique qui peut prendre les valeurs suivantes :

'0'	0V	Y
'1'	+Vdd	Y
'X' : inconnu	?	Y
'Z' : haute impédance	0/Vdd	Y
'H' : rappel à Vdd	0/Vdd (Vdd)	Y
'L' : rappel à 0V	0/Vdd (0V)	Y
'-' : quelconque	0/Vdd	Y

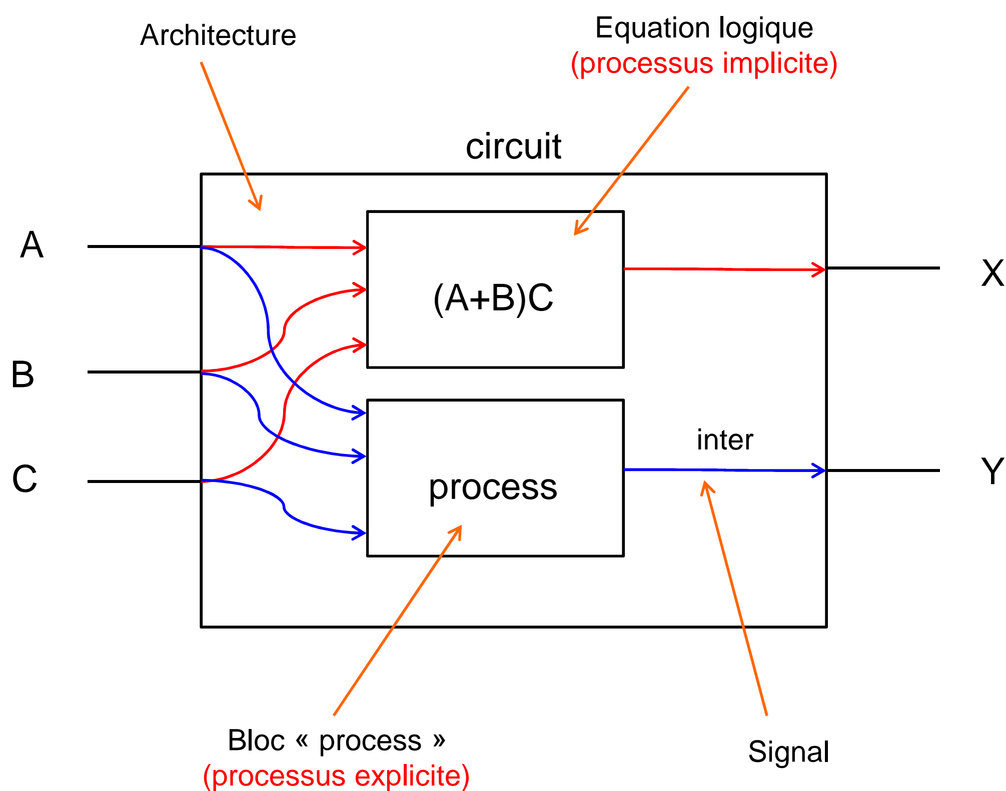
2.4. L'ARCHITECTURE

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS CENTRE DE RECHERCHE



L'ARCHITECTURE (2)

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS CENTRE DE RECHERCHE



2.5. SYNTAXE

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS CENTRE DE RECHERCHE

- Les commentaire commencent après un double tiret (--) et se terminent à la fin de la ligne

```
-- commentaire
```

- Le signe d'affectation d'un signal est la flèche vers la gauche (<=)

```
X <= (A or B) and C ;
```

- Le signe d'affectation d'une valeur initiale est (:=)

```
signal inter : std_logic := '0';
```

- Pas de distinction entre les majuscules et les minuscules

$$inter \equiv INter \equiv inTER$$

- Le séparateur d'instructions est point virgule (;)

2.6. TYPES DE DONNÉES

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS CENTRE DE RECHERCHE

- Le type le plus utilisé pour la synthèse logique est « std_logic » qui est défini dans la librairie « IEEE.STD_LOGIC_1164 »

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

- Un bit est équivalent à un caractère en C, sa valeur est donnée entre quottes

```
inter <= '0';
```

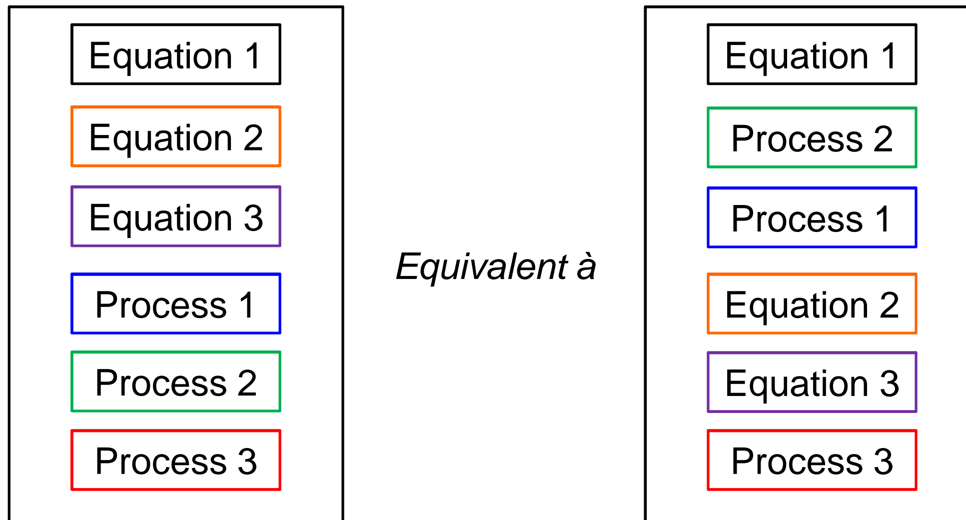
- Un bus (plusieurs bits) est équivalent à une chaine de caractères, sa valeur est donnée en guillemets

```
signal un_sig : std_logic_vector (3 downto 0);
un_sig <= "1001";
```

La même chose en hexa : `un_sig <= x"9";`

2.7. LE PARALLÉLISME

- Une architecture peut comporter plusieurs opérations logiques et plusieurs processus : tout s'exécute en parallèle, sans priorité
- L'ordre d'apparition dans le code n'a pas d'incidence sur le résultat



2.8. LES OPÉRATEURS

Type	***Opérateur***	***Signification***
Logique	NOT, AND, OR, NAND, NOR, XOR, XNOR	
	NOT	Négation, inversion
Arithmétique	+, -, * , / , mod	Add, sous, mult , div , modulo
Rotation	rol , ror , srl , sll	Rotate Shift Logical
Comparaison relative	< , <= , > , >=	
Comparaison d'égalité	= , /=	Egal, différent

2.9. SYNTAXE À L'EXTÉRIEUR DU PROCESS

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS CENTRE DE RECHERCHE

- Affectation inconditionnelle

```
Y <= A or B ;
```

- Affectation inconditionnelle

```
Y <= '1' when (SEL = "101") else '0' ;
```

- Affectation sélective

```
with SEL select
   Y <= A when "00",
        B when "01",
        C when "10",
        D when others;
```


2.10. SYNTAXE DANS LE PROCESS

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS CENTRE DE RECHERCHE

- Affectation inconditionnelle

```
Y <= A or B ;
```

- Affectation inconditionnelle

```
if (SEL = "101") then
   Y <= '1';
else
   Y <= '0';
end if;
```

- Affectation sélective

```
case sel is
   when "00" =>    y <= '1';
   when "01" =>    y <= '0';
   when "10" =>    y <= '0';
   when others => y <= '1';
end case;
```

3. EXEMPLES COMBINATOIRES

3.1. PORTES LOGIQUES

$$Y = \overline{A}\,\overline{B}\,\overline{C} + A\overline{B}\,\overline{C} + A\overline{B}C$$

```
begin

   Y <=


end Behavioral;
```

C	A	B	
I2	I1	I0	O
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Table de vérité

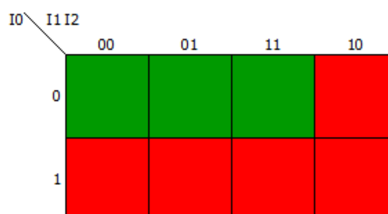
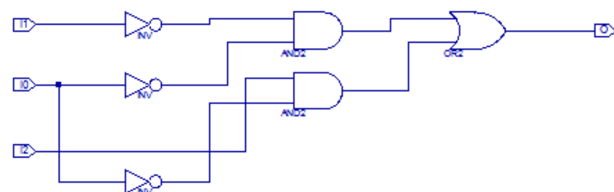
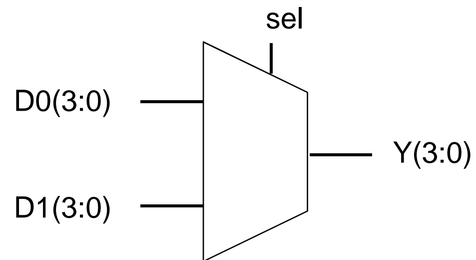


Table de Karnaugh



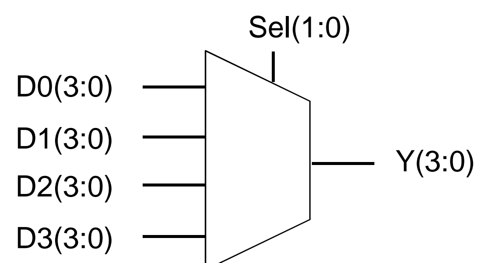
Résultat après synthèse

3.2. MULTIPLEXEUR À 2 ENTRÉES




```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity mux2 is
    Port ( d0 : in  STD_LOGIC_VECTOR (3 downto 0);
           d1 : in  STD_LOGIC_VECTOR (3 downto 0);
           sel : in  STD_LOGIC;
           y : out  STD_LOGIC_VECTOR (3 downto 0));
end mux2;

architecture Behavioral of mux2 is

begin


end Behavioral;
```

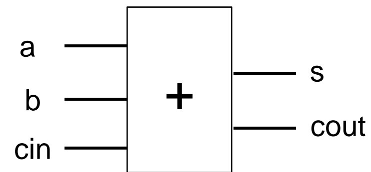

3.3. MULTIPLEXEUR À PLUSIEURS ENTRÉES




```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity mux2 is
    Port ( d0 : in  STD_LOGIC_VECTOR (3 downto 0);
           d1 : in  STD_LOGIC_VECTOR (3 downto 0);
           d2 : in  STD_LOGIC_VECTOR (3 downto 0);
           d3: in  STD_LOGIC_VECTOR (3 downto 0);
           sel : in  STD_LOGIC_VECTOR (1 downto 0);
           y : out  STD_LOGIC_VECTOR (3 downto 0));
end mux2;

architecture Behavioral of mux2 is

begin



end Behavioral;
```

3.4. ADDITIONNEUR COMPLET

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS CENTRE DE RECHERCHE

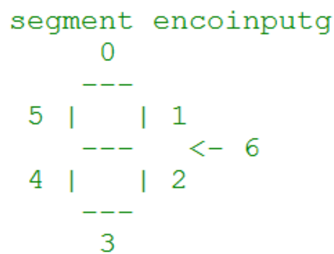



```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity add is
    Port ( a : in  STD_LOGIC;
           b : in  STD_LOGIC;
           cin : in  STD_LOGIC;
           s : out  STD_LOGIC;
           cout : out  STD_LOGIC);
end add;

architecture Behavioral of add is

   signal p, g : std_logic;

begin




end Behavioral;
```


3.5. DÉCODEUR HEXA VERS 7 SEGMENTS

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS CENTRE DE RECHERCHE

```
   LED:    out    STD_LOGIC_VECTOR (6 downto 0);
```

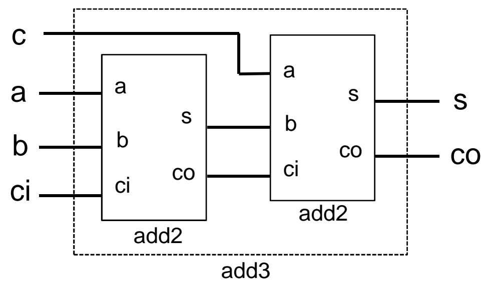



```
  with HEX SELect
LED<= "1111001" when "0001",   --1
      "0100100" when "0010",   --2
      "0110000" when "0011",   --3
      "0011001" when "0100",   --4
      "0010010" when "0101",   --5
      "0000010" when "0110",   --6
      "1111000" when "0111",   --7
      "0000000" when "1000",   --8
                when "1001",   --9
                when "1010",   --A
                when "1011",   --b
                when "1100",   --C
                when "1101",   --d
                when "1110",   --E
                when "1111",   --F
                when others;   --0
```

Attention !
Afficheur à anodes communes

0 : segment allumé
1 : segment éteint

3.6. ASSOCIATION DE MODULES (SCHÉMA)

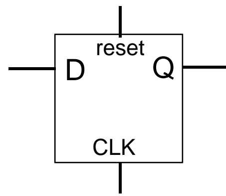


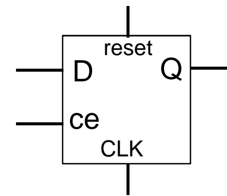

```
architecture Behavioral of add3 is

signal s_sig, co_sig : std_logic;
   COMPONENT add2
   PORT(
      a : IN std_logic;
      b : IN std_logic;
      cin : IN std_logic;
      s : OUT std_logic;
      co : OUT std_logic
      );
   END COMPONENT;
```

```
begin
      Inst_1: add2 PORT MAP(
      a => a,
      b => b,
      cin => c,
      s => s_sig,
      co => co_sig
   );

      Inst_2: add2 PORT MAP(
      a =>
      b =>
      cin =>
      s =>
      co =>
   );

end Behavioral;
```

4. EXEMPLES SÉQUENTIELS

4.1. BASCULE D AVEC RESET/VALIDATION

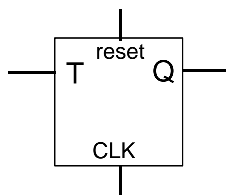


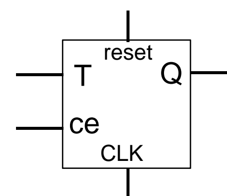



```
process (clk)
begin
   if rising_edge (clk) then




   end if;
end process;
```

```
process (clk)
begin
   if rising edge (clk) then




   end if;
end process;
```


4.2. BASCULE T (TOGGLE)

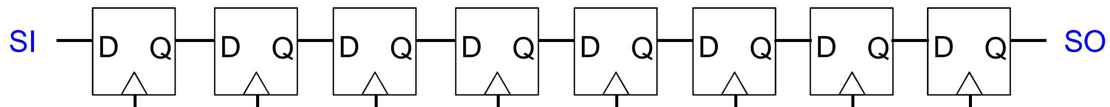





```
process (clk)
begin
   if rising_edge (clk) then




   end if;
end process;
```

```
process (clk)
begin
   if rising_edge (clk) then




   end if;
end process;
```

4.3. REGISTRE À DÉCALAGE




```
entity shift_registers_1 is
    port(CLK, SI : in std_logic;
         SO : out std_logic);
end shift_registers_1;

architecture archi of shift_registers_1 is
    signal tmp: std_logic_vector(7 downto 0);
begin

    process (CLK)
    begin
        if rising_edge (clk) then




        end if;
    end process;

    SO <= tmp(7);
end archi;
```


4.4. COMPTEUR BINAIRE AVEC RESET

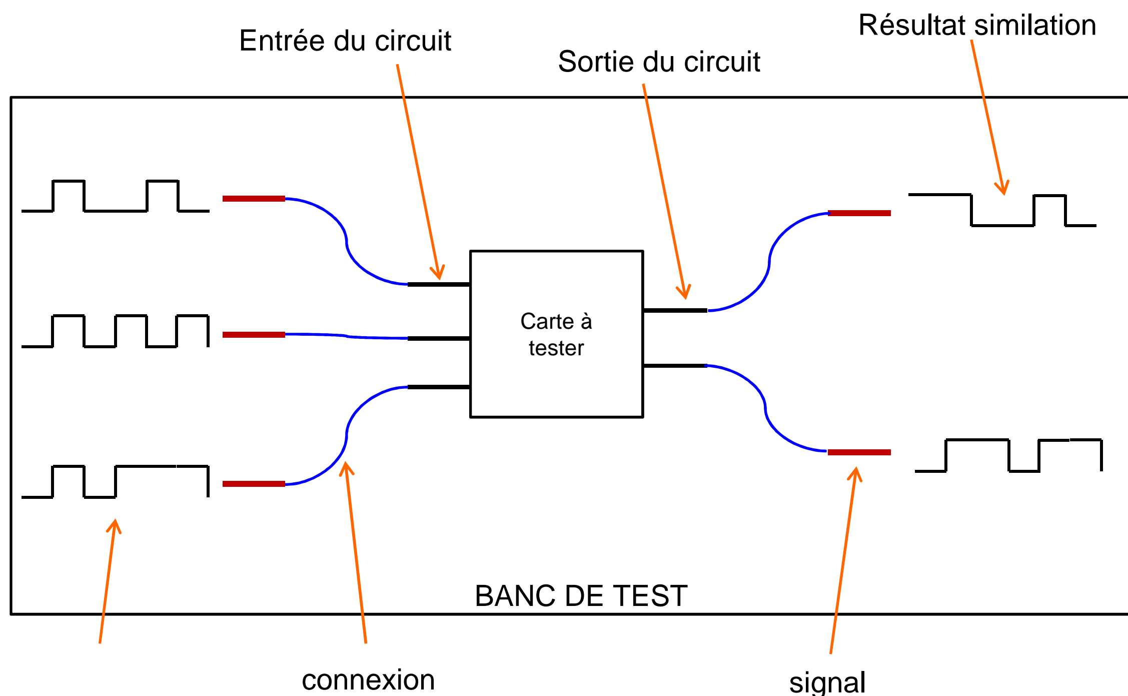
```
entity counters_1 is
    port(CLK, CLR : in std_logic;
         Q : out std_logic_vector(3 downto 0));
end counters_1;

architecture archi of counters_1 is
    signal tmp: std_logic_vector(3 downto 0);
begin
    process (CLK)
    begin






    end process;

    Q <= tmp;
end archi;
```

```
entity counters_1 is
    port(CLK, CLR : in std_logic;
         Q : out std_logic_vector(3 downto 0));
end counters_1;

architecture archi of counters_1 is
    signal tmp: std_logic_vector(3 downto 0);
begin
    process (CLK)
    begin
        if rising_edge (clk) then




        end if;
    end process;

    Q <= tmp;
end archi;
```


ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

5. SIMULATEUR (TESTBENCH)

5.1. PRINCIPE D'UN BANC DE TEST



5.2. CIRCUIT COMBINATOIRE À TESTER

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity additionneur1 is
    Port ( a : in std_logic;
           b : in std_logic;
           retenue_e : in std_logic;
           somme : out std_logic;
           retenue_s : out std_logic);
end additionneur1;

architecture Behavioral of additionneur1 is

begin
   somme <= a xor b xor retenue_e;
   retenue_s <= (a and b)or
     (a and retenue_e)or(b and retenue_e);

end Behavioral;
```

- Nous avons affaire à un circuit combinatoire à 3 entrées et 2 sorties, il s'agit d'un additionneur complet.

5.3. TESTBENCH : DÉCLARATIONS

```
ENTITY additionneur1_testeur_vhd_tb IS
END additionneur1_testeur_vhd_tb;

ARCHITECTURE behavior OF additionneur1_testeur_vhd_tb IS

   COMPONENT additionneur1
   PORT(
      a : IN std_logic;
      b : IN std_logic;
      retenue_e : IN std_logic;
      somme : OUT std_logic;
      retenue_s : OUT std_logic
      );
   END COMPONENT;

   SIGNAL a :  std_logic;
   SIGNAL b :  std_logic;
   SIGNAL retenue_e :  std_logic;
   SIGNAL somme :  std_logic;
   SIGNAL retenue_s :  std_logic;

BEGIN
```

- En début de programme, on déclare le composant à tester ainsi que les différents signaux qui vont servir à connecter le composant.

5.4. TESTBENCH : SIMULATION

```
BEGIN

   uut: additionneur1 PORT MAP(
      a => a,
      b => b,
      retenue_e => retenue_e,
      somme => somme,
      retenue_s => retenue_s
   );

   tb : PROCESS
   BEGIN
      a<='0'; b<='0'; retenue_e<='0';
      wait for 100 ns;
      a<='1';
      wait for 100 ns;
      b<='1';
      wait for 200 ns;
      retenue_e<='1';
      wait for 100 ns;
      a<='0'; b<='0';
   END PROCESS;

END;
```

- La fin du programme sert à connecter le composant et lui imposer des « stimuli » en entrée pour savoir comment il réagit.