6. Application à la Logique Combinatoire

6.1 Passage d'une fonction logique vers une table de vérité simple ou à variables introduites

Toute fonction logique combinatoire peut être transformée en une table de vérité simple ou à variables introduites par application du théorème d'expansion de Shannon.

Théorème d'expansion de Shannon
Toute fonction logique combinatoire peut être décomposée suivant une ou plusieurs variables :

F(x0, .. ,xn) = $\overline{x0}$. F(0, x1, .. ,xn) + x0 . F(1,x1, .. ,xn)
= .... *(cf. Théorème d'expansion de Shannon généralisé)*

✍ **Exemples :** On considère la fonction f(A, B, C) = $A\overline{B} + \overline{A}.C + AB\overline{C}$
Donner la table de vérité en considérant comme variables principales :

a) A, B et C

A	B	C	F
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

b) A et B

A	B	F
0	0	
0	1	
1	0	
1	1	

c) A (seule)

A	F
0	
1	

6.2 Modèle de référence : table de vérité

Le modèle de référence de la logique combinatoire est la table de vérité. Nous pouvons structurer la table de vérité selon l'organisation des entrées et /ou les valeurs stockées dans la table.

Dans la table de vérité simple, on peut stocker les valeurs 0, 1 ou x (non spécifiée). Pour la valeur non spécifiée, la fonction est indifférente qu'elle vaille 0 ou 1. En général, on exploite cet état de fait pour la simplification.

Dans une table de vérité avec variables introduites, le 0, 1 et x peuvent être remplacés par des expressions logiques. L'intérêt de la table de vérité à variables introduites est de réduire le nombre de lignes de la table de vérité simple équivalente. Par ailleurs, cette table de vérité à variables introduites se prête bien aux instructions de langage de description matérielle.

Nous distinguerons 2 tables de vérité : - Table de vérité **simple avec ou sans priorité**.
- Table de vérité **à variables introduites avec ou sans priorité**.

Dans tous les cas, les valeurs non spécifiées peuvent être des valeurs possibles dans la table.

Une table de vérité sans priorité signifie que toutes les combinaisons des entrées principales de la table sont explicitement décrites dans celle-ci.

Exemple ci-contre
Les combinaisons de A et B sont toutes exprimées.

Exemple : table simple

A	B	f
0	0	0
0	1	1
1	0	X
1	1	1

Exemple table à variables introduites

A	B	f
0	0	C + D
0	1	1
1	0	X
1	1	E.F

Une table de vérité avec priorité signifie que pour certaines combinaisons des entrées principales, certaines entrées sont plus importantes que d'autres:
- Hiérarchisation des entrées.
- Combinaisons des entrées moins prioritaires notées 'x'
- Table incomplètement spécifiée ou non.

Exemple : table simple

A	B	f
0	x	0
1	0	1
1	1	0

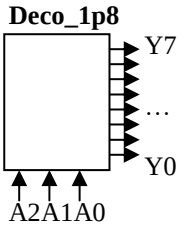
Exemple table à variables introduites

sraz	sload	en	Id[]
1	x	x	0
0	1	x	E[]
0	0	1	Iq[]+1
0	0	0	Iq[]

✍ Exemples 1 : Table de vérité sans priorité

- *Table de vérité simple*

Décodeur 1 parmi 8



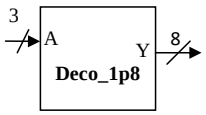


Table de vérité

Entrées			Sorties							
A2	A1	A0	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
0	0	0								
0	0	1								
0	1	0								
0	1	1								
1	0	0								
1	0	1								
1	1	0								
1	1	1								

Description en VHDL

Table de vérité sans priorité donc utilisation de l'instruction d'assignation sélective du mode concurrent **(with select)**

Deux modes de calcul :

Calcul sur tout le vecteur

```
with A select
    Y   <=  "00000001" when "000",
            "00000010" when "001",
            "00000100" when "010",
            "00001000" when "011",
            "00010000" when "100",
            "00100000" when "101",
            "01000000" when "110",
            "10000000" when others;
```

2 versions possibles :
le « when …others » peut être utilisé différemment

```
with A select
    Y   <=  "00000001" when "000",
            "00000010" when "001",
            "00000100" when "010",
            "00001000" when "011",
            "00010000" when "100",
            "00100000" when "101",
            "01000000" when "110",
            "10000000" when "111",
            "00000000" when others;
```

Calcul pour chaque bit du vecteur

```
Y(0) <= '1' when A = "000" else
        '0';
Y(1) <= '1' when A = "001" else
        '0';
Y(2) <= '1' when A = "010" else
        '0';
Y(3) <= '1' when A = "011" else
        '0';
Y(4) <= '1' when A = "100" else
        '0';
Y(5) <= '1' when A = "101" else
        '0';
Y(6) <= '1' when A = "110" else
        '0';
Y(7) <= '1' when A = "111" else
        '0';
```

- *Table de vérité à variables introduites*

Exemple table de vérité à variables introduites

A	B	S
0	0	C + D
0	1	1
1	0	X
1	1	E.F

```
architecture ar of test is
signal I : std_logic_vector (1 downto 0);
begin

    I <= A & B;

    with I select
        S <= C OR D  when "00",
             '1'     when "01",
             X       when "10",
             E AND F when others;


end architecture;
```

Remarque :
Lorsque les expressions par exemple C or D sont complexes alors on définit des signaux intermédiaires pour les évaluer à part.

✍ **Exemples 2 : Table de vérité avec priorité**

- *Table de vérité simple*

Encodeur de priorité

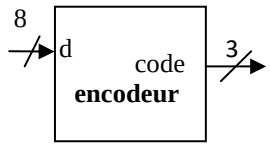


Table de vérité

Entrées								Sorties		
d0	d1	d2	d3	d4	d5	d6	d7	Code2	Code1	Code0
1	x	x	x	x	x	x	x	0	0	0
0	1	x	x	x	x	x	x	0	0	1
0	0	1	x	x	x	x	x	0	1	0
0	0	0	1	x	x	x	x	0	1	1
0	0	0	0	1	x	x	x	1	0	0
0	0	0	0	0	1	x	x	1	0	1
0	0	0	0	0	0	1	x	1	1	0
0	0	0	0	0	0	0	1	1	1	1

Description en VHDL
Table de vérité avec priorité donc utilisation de l'instruction d'assignation conditionnelle du mode concurrent (**when else**)

```
Code <= "000" when d(0) = '1' else
        "001" when d(1) = '1' else
        "010" when d(2) = '1' else
        "011" when d(3) = '1' else
        "100" when d(4) = '1' else
        "101" when d(5) = '1' else
        "110" when d(6) = '1' else
        "111" when d(7) = '1' else
        "000";
```

- *Table de vérité à variables introduites*

Vue externe du bloc combinatoire

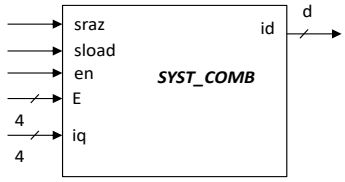


Table de vérité

sraz	sload	en	Id[]
1	x	x	0
0	1	x	E[]
0	0	1	Iq[]+1
0	0	0	Iq[]

```
Id <= (others =>'0') when sraz  = '1' else
      E              when sload = '1' else
      Iq + 1         when en    = '1' else
      Iq;
```

✍ **Exemples de synthèse : Donner la traduction en VHDL en utilisant les instructions adéquates**

a)

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Codage en VHDL

```
architecture ar of test is
signal I : std_logic_vector (2 downto 0);
begin

    I <= A & B & C;

    with I select
        F <= '0'  when "000",
             '1'  when "001",
             '0'  when "010",
             '1'  when "011",
             '1'  when "100",
             '1'  when "101",
             '1'  when "110",
             '0'  when others;

end architecture;
```

Autre codage :
on note que les '1' et on utilise le when others pour tous les '0'

```
architecture ar of test is
signal I : std_logic_vector (2 downto 0);
begin

    I <= A & B & C;

    with I select
        F <= '1'  when "001",
             '1'  when "011",
             '1'  when "100",
             '1'  when "101",
             '1'  when "110",
             '0'  when others;

end architecture;
```

b)

A	B	F
0	0	C
0	1	C
1	0	1
1	1	/C

Codage en VHDL

```
architecture ar of test is
signal I : std_logic_vector (1 downto 0);
begin

    I <= A & B 1;

    with I select
        F <= C     when "00",
             C     when "01",
             '1'   when "10",
             NOT C when others;

end architecture;
```

c)

A	F
0	C
1	/B+B./C

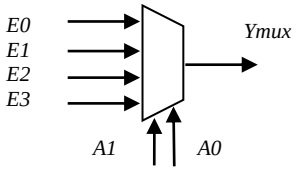
Codage en VHDL

```
with A select
    F <= C     when '0',
         NOT B OR (B AND NOT C) when others;
```

Fonction MULTIPLEXAGE

- **Application au mux 1 vers 4**

Vue fonctionnelle usuelle



Vue composant à construire

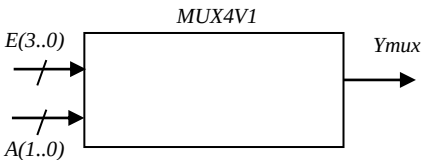


Table de vérité

A(1)	A(0)	Ymux
0	0	E0
0	1	E1
1	0	E1
1	1	E2

Instruction VHDL

table de vérité à Variables introduites sans priorité

→ with … select

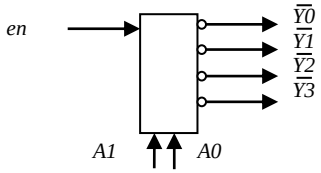
Codage en VHDL

```
library ieee;
use     ieee.std_logic_1164.all;
use     ieee.std_logic_unsigned.all;


entity Mux4v1 is
port(
            A     : in  std_logic_vector (1 downto 0);
            E     : in  std_logic_vector (3 downto 0);
            Ymux  : out std_logic
    );
end entity;


architecture ar of Mux4v1 is

begin
      with A select
            Ymux <= E(0) when "00",
                    E(1) when "01",
                    E(2) when "10",
                    E(3) when others;

end architecture;
```

Fonction DECODAGE

- **Application au décodeur 1p4 avec entrée de validation active niveau haut et sorties actives niveau bas**

Vue fonctionnelle usuelle



Vue composant à construire

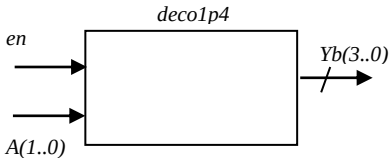


Table de vérité (simple)

en	A(1)	A(0)	Yb(3..0)
0	*	*	1 1 1 1
1	0	0	1 1 1 0
	0	1	1 1 0 1
	1	0	1 0 1 1
	1	1	0 1 1 1

Table de vérité (à variables introduites)

A(1)	A(0)	Yb(3..0)
0	0	1 1 1 /en
0	1	1 1 /en 1
1	0	1 /en 1 1
1	1	/en 1 1 1

Rem : /en = $\overline{en}$

Instruction VHDL

with select

ou

when else

Codage en VHDL

```
library ieee;
use     ieee.std_logic_1164.all;
use     ieee.std_logic_unsigned.all;


entity Deco1p4 is
port(
            A  : in std_logic_vector (1 downto 0);
            en : in std_logic;
            Yb :out std_logic_vector (3 downto 0)

    );
end entity;


architecture ar of Deco1p4 is
signal I : std_logic_vector (2 downto 0);
begin

      I <= en & A;

      with I select
            Yb    <=  "1110" when "100",
                      "1101" when "101",
                      "1011" when "110",
                      "0111" when "111",
                      "1111" when others;

end architecture;
```

Autre codage possible :

```
library ieee;
use     ieee.std_logic_1164.all;
use     ieee.std_logic_unsigned.all;

entity Deco1p4 is
port(
            A  : in std_logic_vector (1 downto 0);
            en : in std_logic;
            Yb :out std_logic_vector (3 downto 0)
    );
end entity;

architecture ar of Deco1p4 is

begin
      Yb(0) <= '0' when A = "00" and en='1' else
                  '1';
      Yb(1) <= '0' when A = "01" and en='1' else
                  '1';
      Yb(2) <= '0' when A = "10" and en='1' else
                  '1';
      Yb(3) <= '0' when A = "11" and en='1' else
                  '1';

end architecture;
```

Autre codage possible :

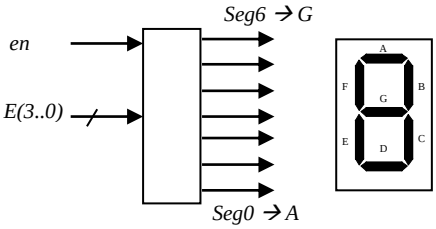
```
library ieee;
use     ieee.std_logic_1164.all;
use     ieee.std_logic_unsigned.all;

entity Deco1p4 is
port(
            A  : in std_logic_vector (1 downto 0);
            en : in std_logic;
            Yb :out std_logic_vector (3 downto 0)
    );
end entity;

architecture ar of Deco1p4 is

begin
      Yb(0) <= not en  when A = "00" else
               '1';
      Yb(1) <= not en  when A = "01" else
               '1';
      Yb(2) <= not en  when A = "10" else
               '1';
      Yb(3) <= not en  when A = "11" else
               '1';

end architecture;
```

Autre codage possible :

```
library ieee;
use     ieee.std_logic_1164.all;
use     ieee.std_logic_unsigned.all;

entity Deco1p4 is
port(
            A  : in std_logic_vector (1 downto 0);
            en : in std_logic;
            Yb :out std_logic_vector (3 downto 0)
    );
end entity;

architecture ar of Deco1p4 is

begin
      Yb <= "1111" when en = '0' else
            "1110" when A = "00" else
            "1101" when A = "01" else
            "1011" when A = "10" else
            "0111" ;

end architecture;
```

- **Application au décodeur 7 segments avec entrée de validation active niveau haut et sorties actives niveau haut**

Vue fonctionnelle usuelle



Vue composant à construire

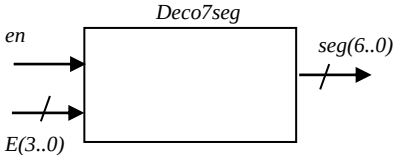


Table de vérité (simple)

en	E(3..0)	Seg(6..0)
0	*	0000000
1	0 0 0 0	0111111
	0 0 0 1	0000110
	0 0 1 0	1001111
	0 0 1 1	1100110
	…	…
	1 1 1 0	1111001
	1 1 1 1	1110001

Instruction VHDL

with select

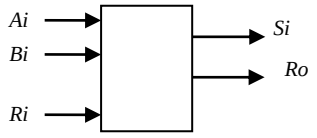
Codage en VHDL

```
library ieee;
use     ieee.std_logic_1164.all;
use     ieee.std_logic_unsigned.all;


entity Deco7seg is
port(
      en : in  std_logic;
      E  : in  std_logic_vector (3 downto 0);
      seg: out std_logic_vector (6 downto 0)

    );
end entity;


architecture ar of Deco7seg is
signal I : std_logic_vector (4 downto 0);
begin

  I <= en & E;

  with I select
        seg <= "0111111"      when "10000",
               "0000110"      when "10001",
               "1011011"      when "10010",
               "1001111"      when "10011",
               "1100110"      when "10100",
               "1101101"      when "10101",
               "1111101"      when "10110",
               "0000111"      when "10111",
               "1111111       when "11000",
               "1101111"      when "11001",
               "1110111"      when "11010",
               "1111100"      when "11011",
               "0111001"      when "11100",
               "1011110"      when "11101",
               "1111001"      when "11110",
               "1110001"      when "11111",
               "0000000"      when others;




end architecture;
```

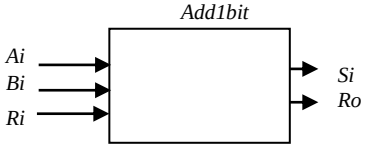
Fonction CALCUL

- **Additionneur 1 bit**

Vue fonctionnelle usuelle



Vue composant à construire



Principe :

Ri
Ai
+Bi
Ro Si

Exemples :

	0	1	1
	0	0	1
+	1	1	1
	0 1	1 0	1 1

Table de vérité :

opération	Ri	Bi	Ai	Ro	Si	résultat
0+0+0	0	0	0	0	0	0
0+0+1	0	0	1	0	1	1
0+1+0	0	1	0	0	1	1
0+1+1	0	1	1	1	0	2
1+0+0	1	0	0	0	1	1
1+0+1	1	0	1	1	0	2
1+1+0	1	1	0	1	0	2
1+1+1	1	1	1	1	1	3

Instruction VHDL

with … select

Ou

instruction de calcul arithmétique : +

Codage en VHDL

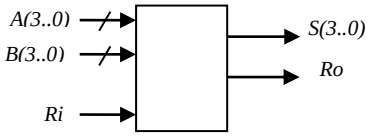
```
library ieee;
use     ieee.std_logic_1164.all;
use     ieee.std_logic_unsigned.all;

entity Add_1bit is
port(
     Ai, Bi, Ri : in  std_logic;
     Si, Ro     : out std_logic

    );
end entity;

architecture ar of Add_1bit is
signal I : std_logic_vector (2 downto 0);
begin
        I <= Ri & Bi & Ai;

        with I select
                Ro <=   '0' when "000",
                        '0' when "001",
                        '0' when "010",
                        '1' when "011",
                        '0' when "100",
                        '1' when "101",
                        '1' when "110",
                        '1' when others;

        with I select
                Si <=   '0' when "000",
                        '1' when "001",
                        '1' when "010",
                        '0' when "011",
                        '1' when "100",
                        '0' when "101",
                        '0' when "110",
                        '1' when others;
end architecture;
```

Autre codages possibles :

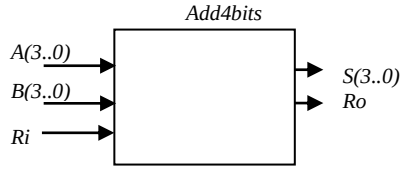
```
library ieee;
use     ieee.std_logic_1164.all;
use     ieee.std_logic_unsigned.all;


entity Add_1bit is
port(
     Ai, Bi, Ri : in  std_logic;
     Si, Ro     : out std_logic

    );
end entity;


architecture ar of Add_1bit is
signal opAi, opBi, OpRi: std_logic_vector (1 downto 0);
signal Res             : std_logic_vector (1 downto 0);

begin
        opAi <= '0' & Ai;
        opBi <= '0' & Bi;
        opRi <= '0' & Ri;

        Res <= opAi + opBi + opRi;

        Ro <= Res(1);
        Si <= Res(0);

end architecture;
```

- **Additionneur 4 bits**

Vue fonctionnelle usuelle



Vue composant à construire



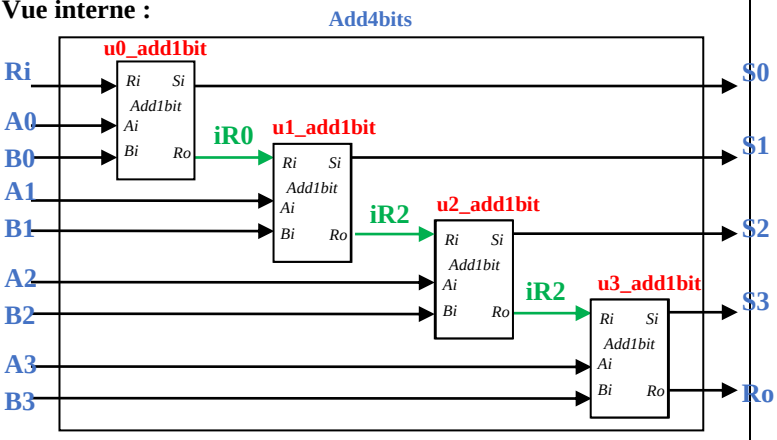
Principe :

Ri
A3 A2 A1 A0
+ B3 B2 B1 B0
Ro S3 S2 S1 S0

Exemple :

				1
	0	1	0	1
+	0	1	0	1
0	1	0	1	1

Vue interne :

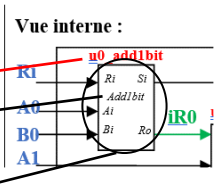


Instruction VHDL

- **instance de composant (instanciation)**
- **port ….map**

Codage en VHDL

```
library ieee;
use     ieee.std_logic_1164.all;
use     ieee.std_logic_unsigned.all;

library work;
use work.sin_pack.all;

entity Add_4bits is
port(
            A,B : in  std_logic_vector (3 downto 0);
            Ri  : in  std_logic;
            S   : out std_logic_vector (3 downto 0);
            Ro  : out std_logic
    );
end entity;

architecture ar of Add_4bits is
signal iR   : std_logic_vector (2 downto 0);
begin

      u0_add1bit : add1bit
                   port map (
                              Ai => A(0),
                              Bi => B(0),
                              Ri => Ri,
                              Ro => iR(0),
                              Si => S(0)
                   );
```



Nom d'utilisation.
Ce nom est libre, on note souvent :
- ***u****…. pour indiquer « utlisation »,*
- *puis un* ***indice*** *au cas où on utilise plusieurs fois le même composant,*
- *puis le* ***nom du composant*** *lors de sa création, c'est-à-dire le nom que l'on va trouver dans le package)*

Il faut regarder sur le schéma les noms des signaux connectés.

A lire : « … est connecté sur … »

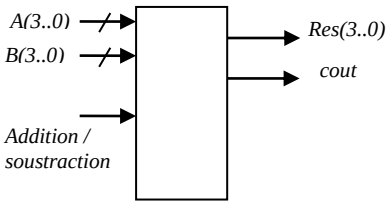
```
      u1_add1bit : add1bit
                        port map (
                                          Ai => A(1),
                                          Bi => B(1),
                                          Ri => iR(0),
                                          Ro => iR(1),
                                          Si => S(1)
                                    );

      u2_add1bit : add1bit
                        port map (
                                          Ai => A(2),
                                          Bi => B(2),
                                          Ri => iR(1),
                                          Ro => iR(2),
                                          Si => S(2)
                                    );

      u3_add1bit : add1bit
                        port map (
                                          Ai => A(3),
                                          Bi => B(3),
                                          Ri => iR(2),
                                          Ro => Ro,
                                          Si => S(3)
                                    );

end architecture;
```

Autre codage possible :

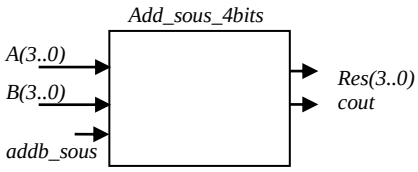
```
library ieee;
use     ieee.std_logic_1164.all;
use     ieee.std_logic_unsigned.all;

library work;
use work.sin_pack.all;

entity Add_4bits is
port(
            A,B : in  std_logic_vector (3 downto 0);
            Ri  : in  std_logic;
            S   : out std_logic_vector (3 downto 0);
            Ro  : out std_logic
    );
end entity;

architecture ar of Add_4bits is
signal opAi, opBi, OpRi: std_logic_vector (4 downto 0);
signal Res             : std_logic_vector (4 downto 0);

begin
      opAi <= '0' & A;
      opBi <= '0' & B;
      opRi <= "0000" & Ri;

      Res <= opAi + opBi + opRi;

      Ro <= Res(4);
      S  <= Res(3 downto 0);

end architecture;
```

- **Additionneur / Soustracteur 4 bits**

Vue fonctionnelle usuelle

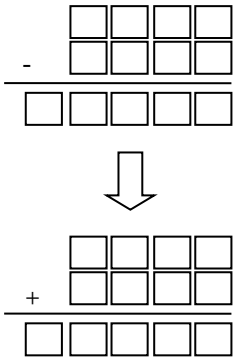


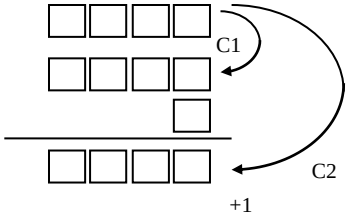
Vue composant à construire



Principe de la soustraction :

Exemple :





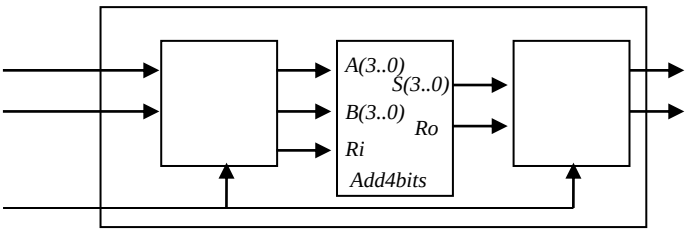
Bilan :

Si addition

Res =
Cout =

Si soustraction

Res =
Cout =



Instruction VHDL

- ……
- ……

Codage en VHDL

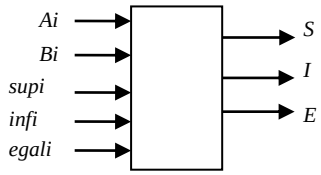
```
library     ieee;
use         ieee.std_logic_1164.all;
use         ieee.std_logic_unsigned.all;

library ……
use ……..


entity      add_sous_4bits    is
 port (     A, B                    : in    std_logic_vector (3 downto 0);
            addb_sous               : in    std_logic;
            S                       : out  std_logic_vector (3 downto 0);
            cout        : out  std_logic
      );
end add_ entity _4bits;
```

```
architecture ar of add_sous_4bits    is




begin

end architecture;
```

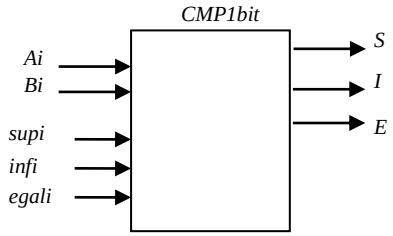
Fonction COMPARAISON

- **Comparateur 1 bit**

Vue fonctionnelle usuelle



Vue composant à construire



Principe :

Table de vérité (simple):

supi	infi	egali	Ai	Bi	S	I	E

Table de vérité (à variables introduites) :

supi	infi	egali	S	I	E

Instruction VHDL :

Codage en VHDL

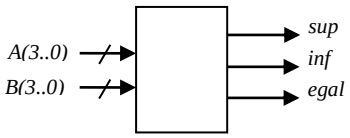
```
library     ieee;
use         ieee.std_logic_1164.all;
use         ieee.std_logic_unsigned.all;

entity      CMP1bit    is
 port (     Ai, Bi                  : in    std_logic;
            supi, infi, egali       : in std_logic ;
            S, I, E                 : out  std_logic
      );
end entity;

architecture ar of CMP1bit is

begin









end architecture;
```

Autre codages possibles :

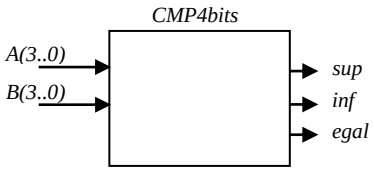
```
library     ieee;
use         ieee.std_logic_1164.all;
use         ieee.std_logic_unsigned.all;

entity      CMP1bit    is
port (      Ai, Bi                  : in    std_logic;
            supi, infi, egali       : in std_logic ;
            S, I, E                 : out  std_logic
      );
end entity;

architecture ar of CMP1bit is

begin









end architecture;
```

- **Comparateur 4 bits**

Vue fonctionnelle usuelle

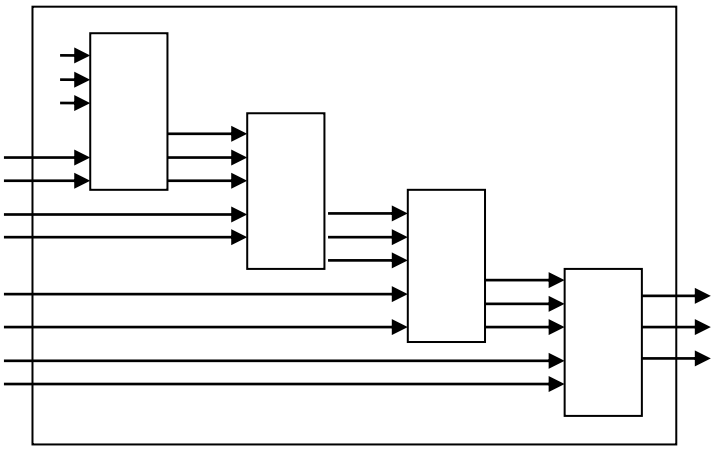


Vue composant à construire



Principe : **Exemple :**

Vue interne :



Instruction VHDL

- **……**
- **……**

Codage en VHDL

```
library     ieee;
use         ieee.std_logic_1164.all;
use         ieee.std_logic_unsigned.all;

library ……
use ……..


entity      CMP4bits    is
 port (     A, B                    : in    std_logic_vector (3 downto 0);
            sup,inf,egal            : out   std_logic
      );
end entity;

architecture ar of CMP4bits is

            ……..……..……..……..

begin

            ……
                ……..……
                            ……
                            ……
                            ……
                            ……
```

```
        ……….
            ……..……
                        ……..……
                        ……..……
                        ……..……
                        ……..……


        ……….
            ……..……
                        ……..……
                        ……..……
                        ……..……
                        ……..……


        ……….
            ……..……
                        ……..……
                        ……..……
                        ……..……
                        ……..……

end architecture;
```

Autre codage possible :

```
library     ieee;
use         ieee.std_logic_1164.all;
use         ieee.std_logic_unsigned.all;

entity      CMP4bits    is
 port (     A, B                : in    std_logic_vector (3 downto 0);
            sup,inf,egal        : out  std_logic
      );
end entity;

architecture ar of CMP4bits is

begin


        ……..……..……..

        ……..……..……..

        ……..……..……..

        ……..……..……..

end architecture;
```

PACKAGE

```
library             ieee;
use                 ieee.std_logic_1164.all;
use                 ieee.std_logic_unsigned.all;


PACKAGE SIN_PACK is
        -- ------------------------------------------------ Mulitplexeur 4 vers 1 ----------------------------
        Component   mux4v1          is
                    port (      E           : in  std_logic_vector(3 downto 0);
                    A           : in  std_logic_vector(1 downto 0);
                    Ymux        : out std_logic
              );
        end Component   ;

        -- ------------------------------------------------ Décodeur 1 parmi 4 ----------------------------
        Component   deco1p4         is
         port (     en          : in  std_logic;
        A           : in  std_logic_vector(1 downto 0);
        Yb          : out std_logic_vector(3 downto 0)
             );
        end Component   ;

        -- ------------------------------------------------ Décodeur binaire – 7 segments --------------
        Component   deco7Seg    is
         port (     en          : in  std_logic;
        E           : in  std_logic_vector(3 downto 0);
        seg         : out std_logic_vector(6 downto 0)
             );
        end Component   ;

        -- ------------------------------------------------ Additionneur complet 1 bit ---------------------
        Component   add1bit     is
         port (     Ai, Bi      : in    std_logic;
        Si, Ro      : out  std_logic
             );
        end Component   ;
```

end PACKAGE;