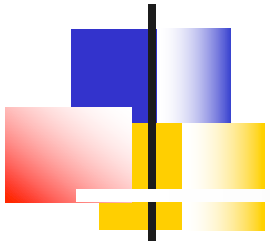
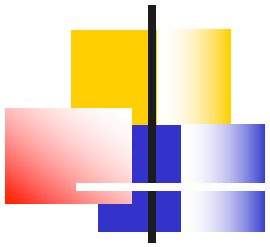


C3 – Les opérateurs de base

Yann DOUZE

VHDL



Affectation simple : <=

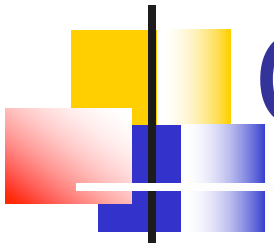
Exemples :

```
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`

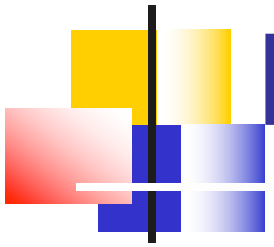
HEXA, exemple : `BUS <= x"9" ; -- BUS = 9 en hexa`



Opérateurs logiques

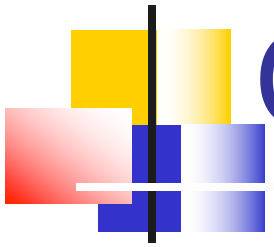
NON	→	***not***
ET	→	***and***
NON ET	→	***nand***
OU	→	***or***
NON OU	→	***nor***
OU EXCLUSIF	→	***xor***

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



Exercices

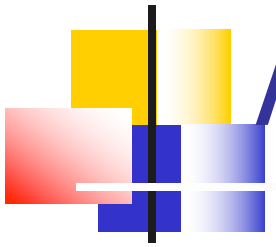
Faire les exercices 1 et 2.



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];
```

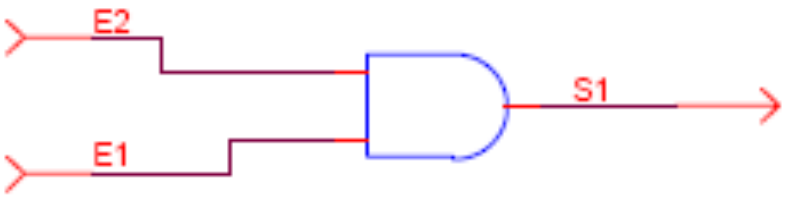
Remarque : l'instruction `[else expression]` permet de définir la valeur du `SIGNAL` par défaut.

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*

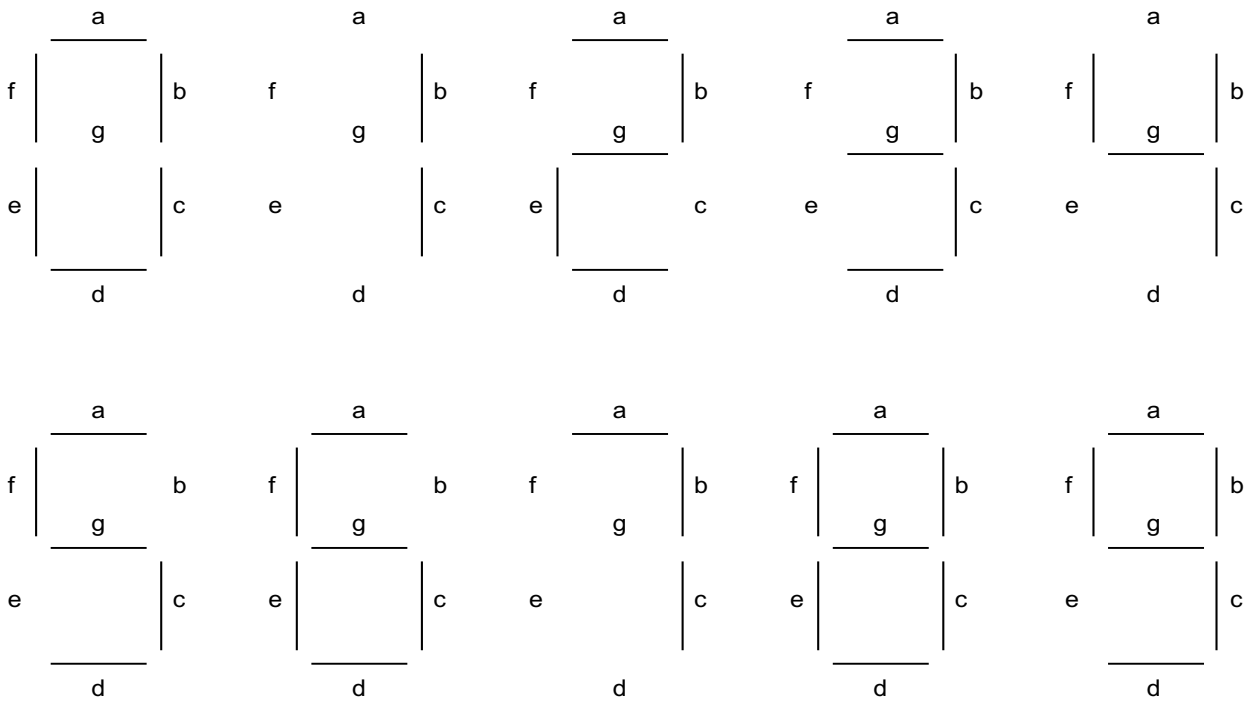
```
with SEL select
     Y <=  A when '0',
     B when '1',
     '0' when others;
```

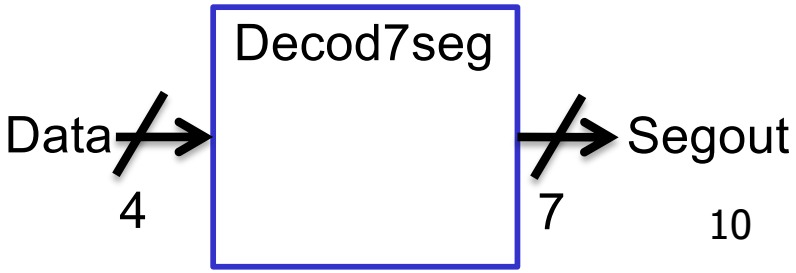
Remarque: Dans le cas du multiplexeur, *when others* est nécessaire car il faut toujours définir les autres cas du signal de sélection pour prendre en compte toutes les valeurs possibles de celui-ci.

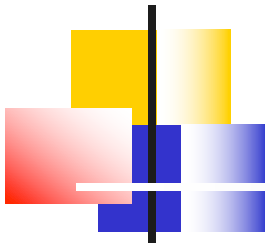
```
with SEL select
    Y <= A when '0',
     B when '1',
     '-' when others;
-- pour les autres cas de SEL, Y prendra une valeur quelconque
-- permet d'optimiser la synthèse
```

Exemple : décodeur 7 segments (1)

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

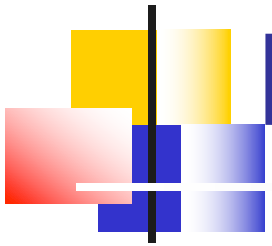






Exemple : décodeur 7 segments (2)

```
Entity decod7Seg is
port (
    Data : in std_logic_vector(3 downto 0); -- Expected within 0 .. 9
    Segout : out std_logic_vector(1 to 7) ); -- Segments A, B, C, D, E, F, G
end entity;

Architecture behavior of decod7seg is
begin
with Data select
    Segout <=   "1111110" when x"0",
                "0110000" when x"1",
                "1101101" when x"2",
                "1111001" when x"3",
                "0110011" when x"4",
                ….
                "1111011" when x"9",
                "----------" when others;
End architecture;
```



Exercice

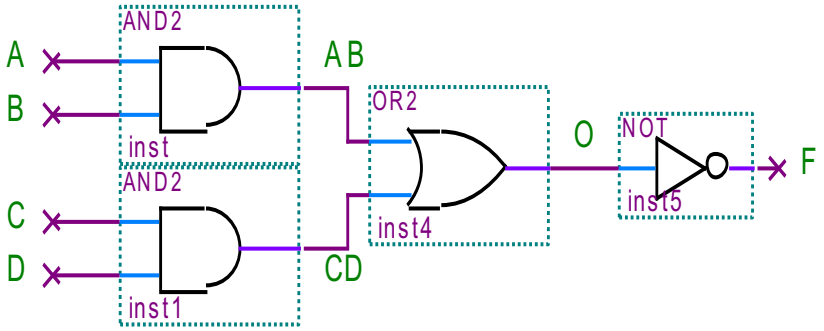
- Faire l'exercice 3 en utilisant une affectation sélective ou conditionelle.

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 :



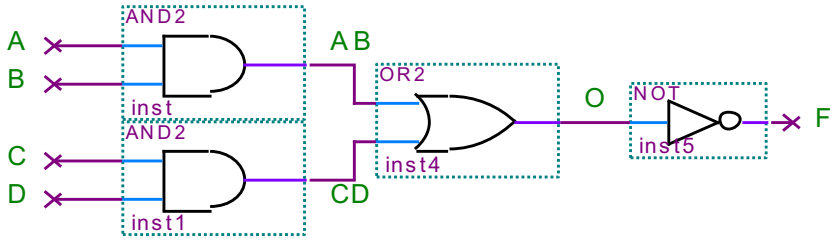
- Sans signaux internes

```
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

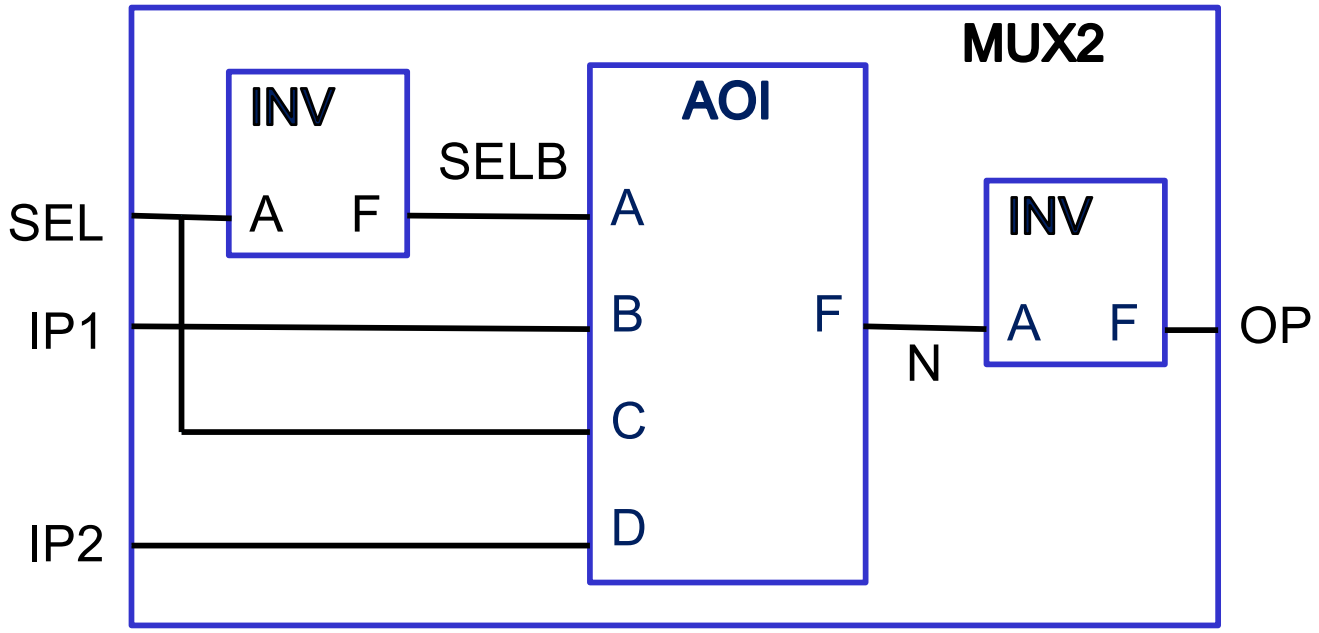
```
architecture V2 of AOI is
   --zone de declaration des signaux
   signal AB,CD,O: STD_LOGIC;
begin
   --instructions concurrentes
   AB <= A and B;
   CD <= C and D;
   O  <= AB or CD;
   F  <= not O;
end architecture;
```

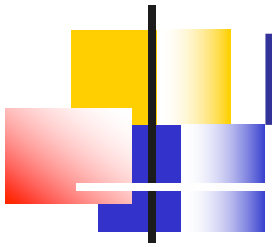


- Instructions concurrentes :
  - L'ordre dans lequel sont écrites les instructions n'a aucune importance.
  - Toutes les instructions sont évaluées et affectent les signaux de sortie en même temps.
  - Différence majeure avec un langage informatique.

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 :

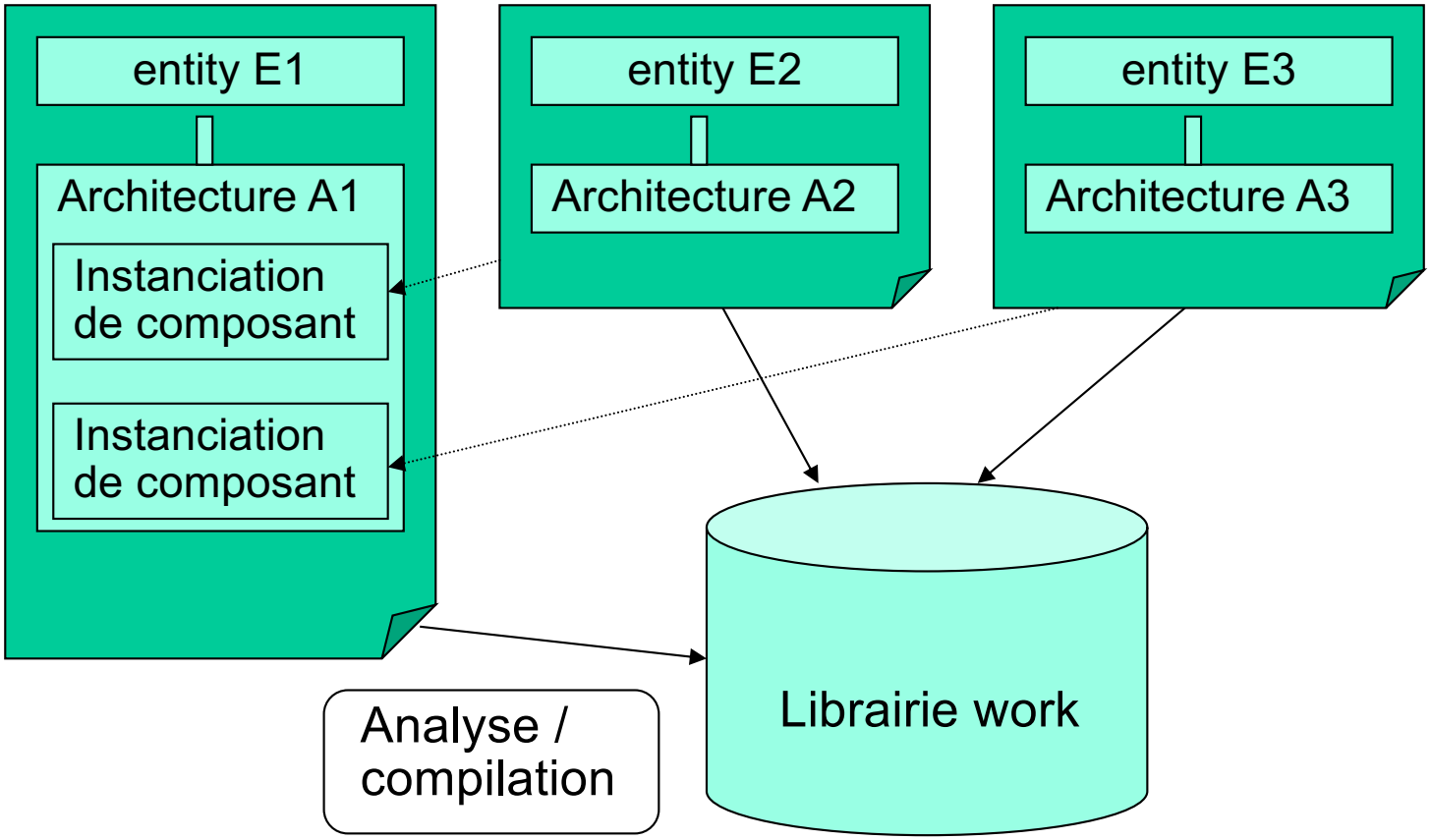


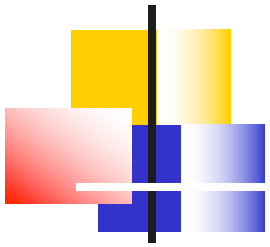


Description structurelle

- 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

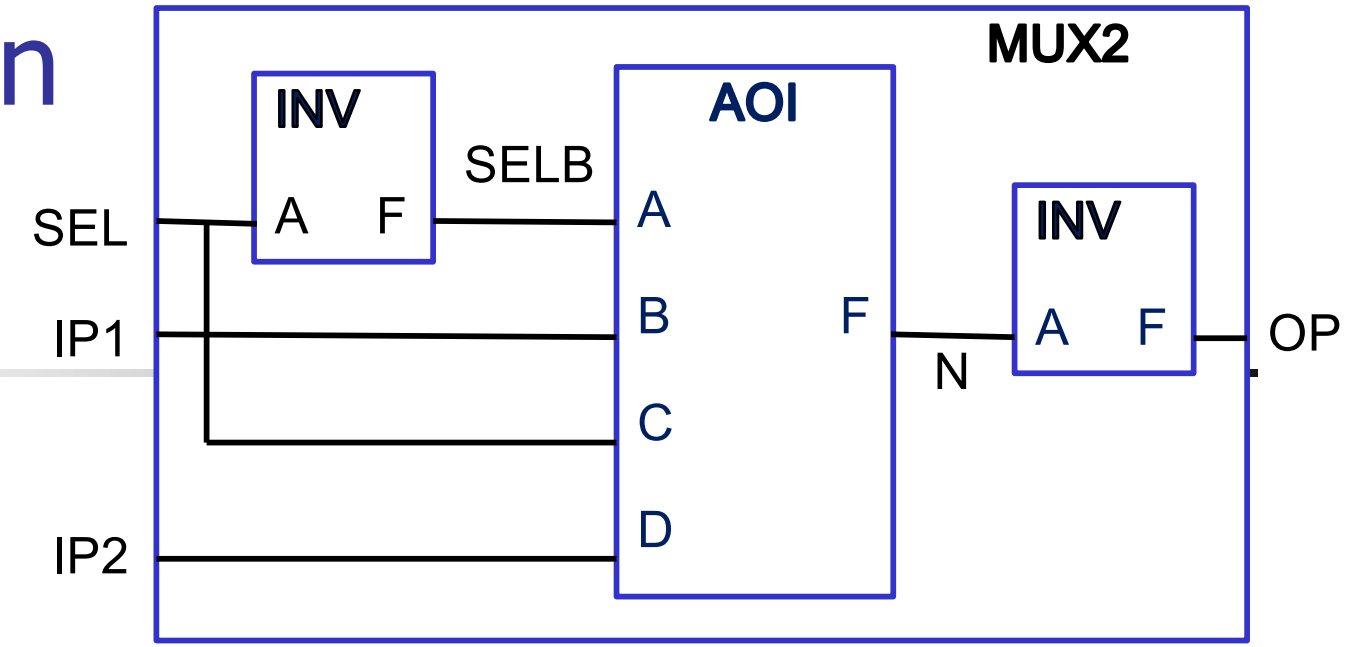




Déclaration des composants INV et AOI

```
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;
```

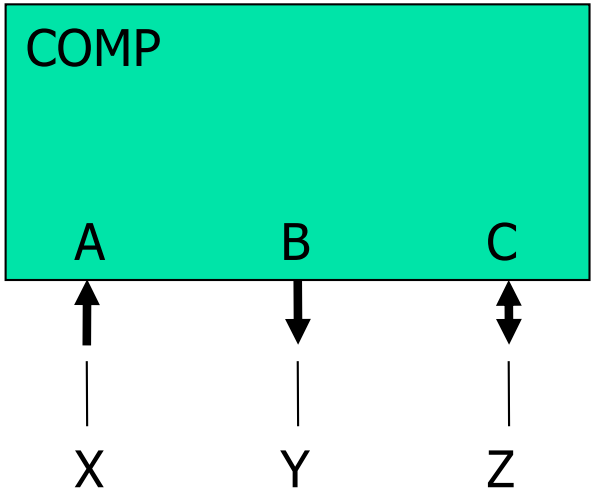
Instanciation Directe




```
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

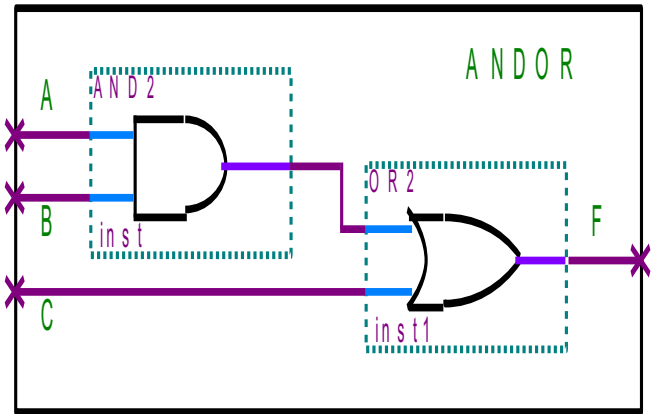
```
entity COMP
port(  A: in      STD_LOGIC;
       B: out     STD_LOGIC;
       C: inout   STD_LOGIC);
end entity;

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
```

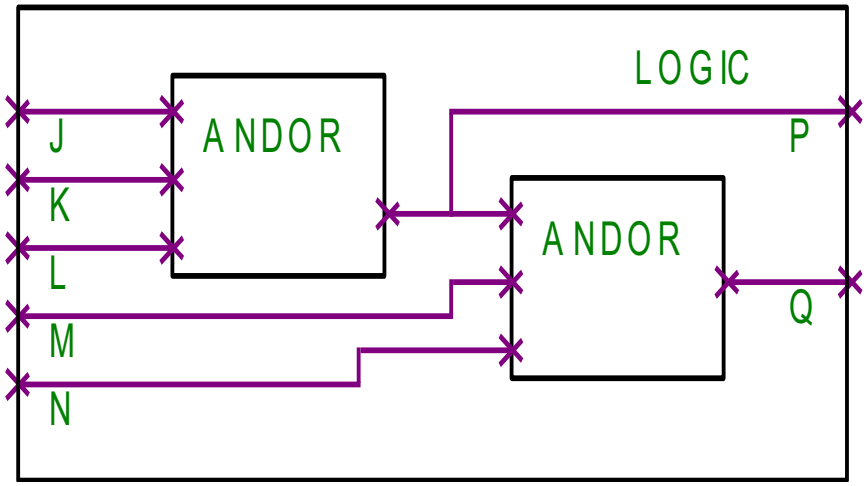
```
C1: entity work.COMP port map (X, Y, Z); --association par position
```

Exercice à compléter

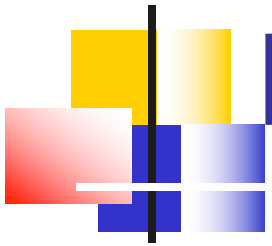



```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

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 ;
Architecture STRUCT of LOGIC is
```



Exercices

- Faire les exercices 4 et 5.