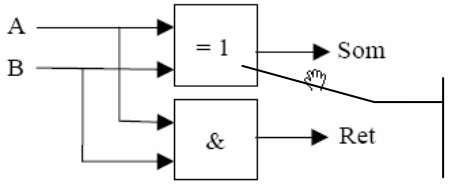
VHDL

Instructions concurrentes
Logique combinatoire

Présention

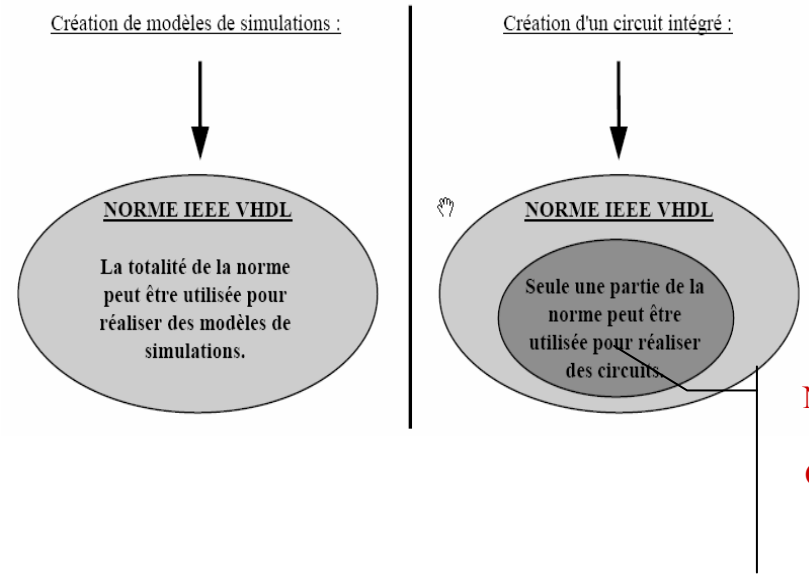
Programmation ou description?

- Les objectifs du langage VHDL
  - ✓ Conception de circuits intégrés reCONFIGURABLE ou non (ASIC, FPGA…) : SYNTHESE
  - ✓ Mise au point de modèle de simulations numériques (circuits virtuels) : MODELISATION
- Le langage est capable de DECRIRE
  - ✓ des comportements CONCURRENTS (//)
  - ✓ Des comportements séquentiels

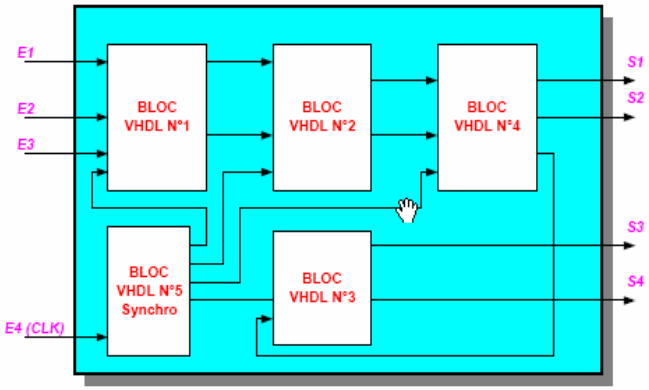


Les deux portes travaillent en //

Synthèse ou modélisation



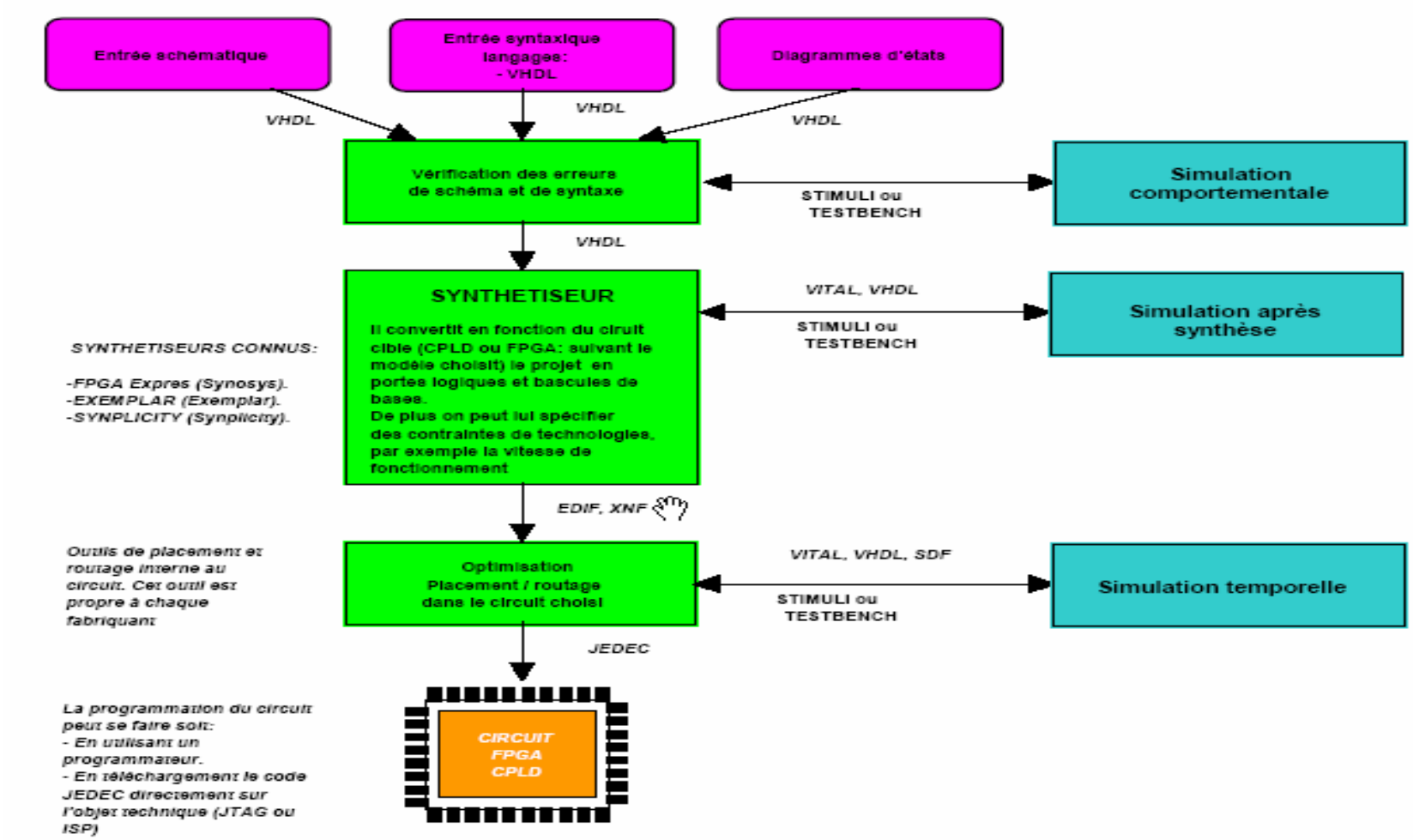
Nous nous focaliserons dans ce cours à la synthèse uniquement



Notre cible en TP: FPGA Cyclone 2 sur la carte DE2

Flot de conception

- **Un outils de développement: Quartus II d'Altera**



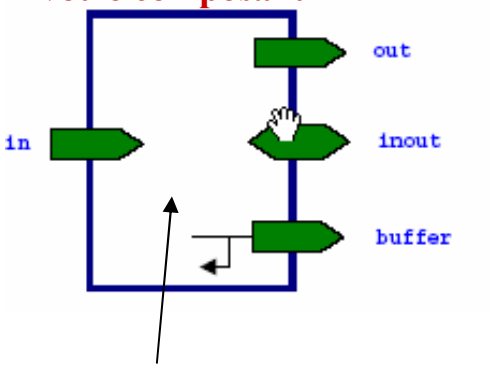
VHDL: concepts de base

Structure générale

Votre fichier texte de description: xxx**.vhd**

Le couple entity architecture définit votre COMPOSANT

***entity* définit l'accès à votre composant**



***architecture* définit le contenu (la fonction) de votre composant**

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

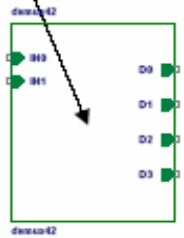
Déclaration des bibliothèques

Commentaires, en VHDL ils commencent par - -

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

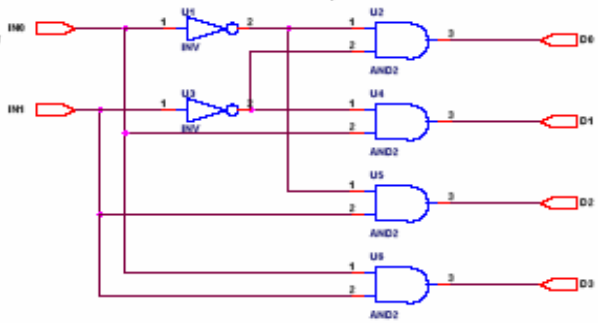
Déclaration de l'entité du démultiplexeur
Correspondance schématique

```
entity DEMUX2_4 is
      port(IN0, IN1: in std_logic;
           D0, D1, D2, D3: out std_logic);
end DEMUX2_4;
```



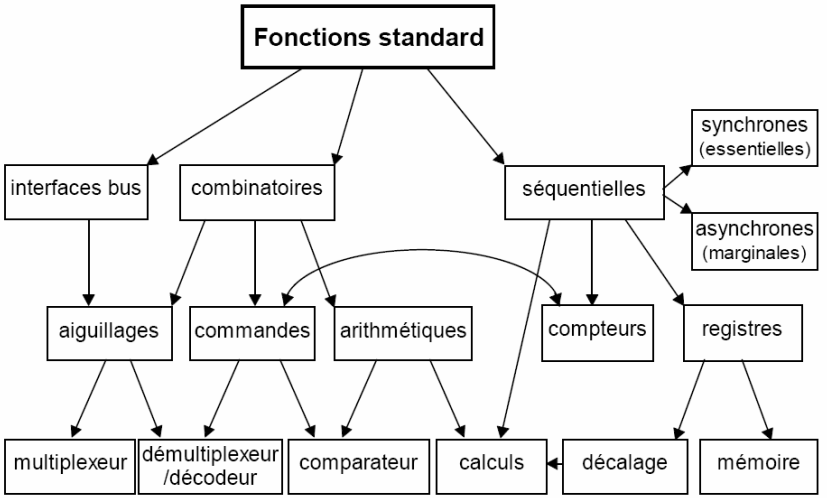
Déclaration de l'architecture du démultiplexeur
Correspondance schématique

```
architecture DESCRIPTION of DEMUX2_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;
```



VHDL: concepts de base

- ***M*éthodologie de conception**
  - Guide pratique du débutant
    - ✓ Décomposition du cahier des charges en fonctions élémentaires
    - ✓ Classification de la fonction
      - COMBINATOIRE:
        - instructions dites concurrentes
      - SEQUENTIELLE:
        - utilisation d'un PROCESS



Logique combinatoire

La sortie **ne** dépend **pas** de l'état passé

Un vecteur d'entrée = un vecteur de sortie unique

Des exemples:

•Multiplexeurs

•Additionneurs

•Décodeur 7 segements

•Encodeurs de priorité

Logique séquentielle

La sortie **dépend** de son état passé

Système nécessitant une horloge (systèmes dits synchrones)

Des exemples:

•Compteurs

•Registres à décalage

•Machine d'état (automate)

VHDL: concepts de base

- **Les objets manipulés**
  - Caractères: ’0’, ’x’,’a’,’%’
  - Chaînes: ”xx”,”bonjour”,”$@&”
  - Des bits: ‘0’, ‘1’
  - Chaînes de bits dans différentes base
    - ✓ ”00101101”, X”2D”, O ”055”
  - Décimaux:27, -5, 4e3, 76_562, 4.25

Ne pas confondre 1 bit

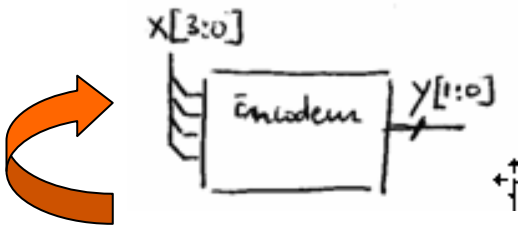
exemple ‘0’ ou ‘1’

Avec un vecteur de bits

” 11 ” ou ” 1101110 ”

- **Les opérateurs**
  - Logiques (boolean, bit, std_ulogic)
    - ✓ AND, OR, NAND, NOR, XOR, NOT
  - Relationnels (retournent un boolean)
    - ✓ = /= < <= > >=
  - Arithmétiques
    - ✓ + - * / ** MOD REM
  - Concaténations d’éléments de tableaux &
    - ✓ "bon" & "jour" => "bonjour"

Un bus (ou ensemble de fils électrique) est représenté sous forme d’un vecteur de bits



STD_LOGIC_VECTOR (3 DOWNTO 0);

VHDL: concepts de base

Les librairies

- Facilite la tâche du concepteur
- Rajout de fonctionnalités supplémentaires

La librairie IEEE

- A mettre au début de votre description
- Pour rajouter les types étendues **std_logic** et **std_logic_vector**
  - ✓ use IEEE.STD_LOGIC_1164.all;
- DORENAVANT nous remplacerons *SYSTEMATIQUEMENT*
  - ✓ BIT par STD_LOGIC
  - ✓ BIT_VECTOR par STD_LOGIC_VECTOR
- Pour utiliser des fonctions arithmétiques sur ces STD_LOGIC_VECTOR
  - ✓ USE IEEE.NUMERIC_STD.ALL;
  - ✓ Et aussi USE IEEE.LOGIC_UNSIGNED.ALL; ou USE IEEE.LOGIC_UNSIGNED.ALL;

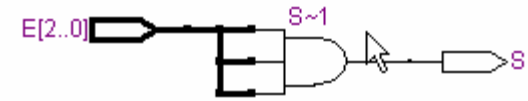
Exemples Applicatifs

- Q<=Q+1; -- Q étant par exemple un std_logic_vector et 1 est un entier!!
- A<B -- A et B des std_logic_vector
- oData<=CONV_STD_LOGIC_VECTOR(TEMP,8); avec TEMP integer range 0 to 255;

Logique combinatoire: Instructions concurrentes

Assignations simples

Exemple: PORTE ET

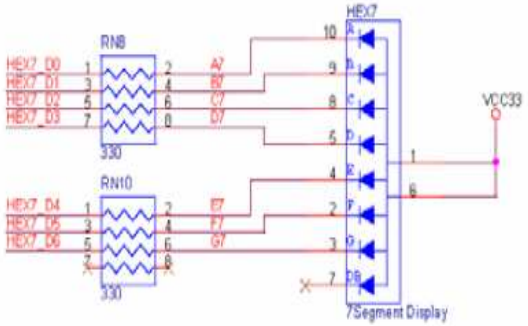



```
library IEEE;
use IEEE.std_logic_1164.all;
 --librairie pour inclure type std_logic

--portes ET
--3 entrées E2 E1 E0
-- 1 sortie S0
entity ET3 is
port(
                E:IN std_logic_vector(2 downto 0);
                S:OUT std_logic
                );
end ET3;

--definition de l'architecture
architecture arch_ET3 of ET3 is
begin
S<=E(2) and E(1) and E(0); -- E(2) accès au fil 2
end arch_ET3;
```

Bit Poid Fort

Bit Poid faible

Exemple: allumer des LEDS




```
library IEEE;
use IEEE.std_logic_1164.all;

entity example is
port(
                HEX4,HEX5,HEX6:OUT std_logic;
                HEX:OUT std_logic_vector(3 downto 0);
                        --3 fils S1[3:1]
                );
end example;

--definition de l'architecture
architecture arch_example of example is
begin
HEX4<='0';  --allumé
HEX5<='1'; -- éteint
HEX6<='1'; -- éteint

HEX<="1010";
end arch_example;
```

Ne pas confondre 1 bit

exemple ‘0’ ou ‘1’

Avec un vecteur de bits

” 11 ” ou ” 1101110 ”

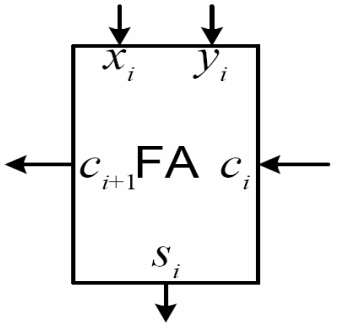
Logique combinatoire: Instructions concurrentes

Assignations simples

Exemple: Manipuler des bits

```
library IEEE;
use IEEE.std_logic_1164.all;
entity example is
port(
                E:IN std_logic_vector(2 downto 0);
                S1:OUT std_logic; --1 fil
                S2,S3:OUT std_logic_vector(3 downto 1);
                 --3 fils S1[3:1]
                S4:OUT std_logic_vector(2 downto 0)
                );
end example;

--definition de l'architecture
architecture arch_example of example is
begin
S1<='0';
S2<='1'& E(1 downto 0);
-- operateur COLLER (ou CONCATENE)  &
-- S2(3)          S2(1)          S2(0)
--  '1'            E(1)           E(0)
S3<="101";
S4<= "111" XOR E; --manip sur les bus directement
end arch_example;
```

Exemple: Additionneur complet 1 bit



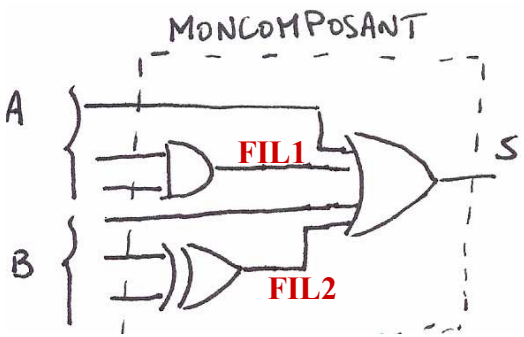
x_i	y_i	c_i	c_{i+1}	s_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY fa IS PORT (
Ci, Xi, Yi: IN STD_LOGIC;
Cout, Si: OUT STD_LOGIC);
--Ci+1 renommé avec Cout
END fa;
ARCHITECTURE Dataflow OF fa IS
BEGIN
Cout <= (Xi AND Yi) OR (Ci AND (Xi XOR Yi));
Si <= Xi XOR Yi XOR Ci;
END Dataflow;
```

Logique combinatoire: Instructions concurrentes

Le mot clé SIGNAL

- ✓ Permet d'établir des connexions (physiques et logiques) à l'intérieur d'un composant
- ✓ Équivaut à « tirer » un fil
- ✓ Méthode pour repérer le besoin en signaux
  - Dessiner votre boîte
  - Nommer les entrées sorties
  - Dessiner l'intérieur du composant: les fils « internes » correspondent à des signaux qu'il faut déclarer juste après votre architecture



L'ordre des lignes n'est pas important (CONCURRENCE!!)

La déclaration des signaux est entre architecture et begin

```
library IEEE;
use IEEE.std_logic_1164.all;


entity example is
port(
                A,B:IN std_logic_vector(3 downto 1);
                S:OUT std_logic
                );
end example;


architecture arch_ example of example is
signal FIL1,FIL2:std_logic;
begin
S<=FIL2 OR FIL2 OR A(1) OR B(1);
FIL2<=B(2) AND B(3);
FIL1<=A(3) AND A(2);
end arch_ example;
```

Logique combinatoire: Instructions concurrentes

Assignation sélective

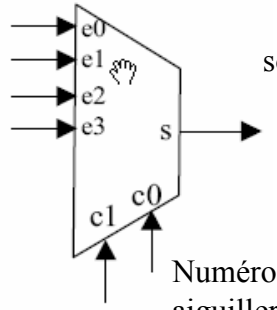
- Structure WITH/ SELECT

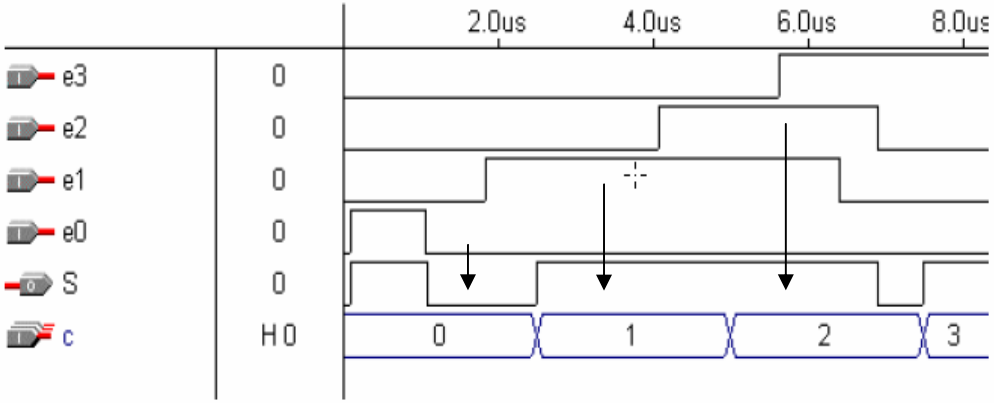
```
with  expression  select
signal1  <=  signal1 when   valeur 1,
             signal2 when   valeur2,
             ------
             signal par  défaut when others ;
```

Le multiplexeur: aiguille une entrée vers la sortie en fonction d'un numéro d'aiguillage

MUX

1 voie parmi 4






```
LIBRARY ieee;
 USE ieee.std_logic_1164.all;
 ----------------------------------------
 ENTITY MUX IS
 PORT            (
    c: IN STD_LOGIC_VECTOR (1 DOWNTO 0);
    e: IN STD_LOGIC_VECTOR (3 DOWNTO 0);
    s: OUT STD_LOGIC
                 );
 END MUX;
 ----------------------------------------
 ARCHITECTURE arch_MUX OF MUX IS
 BEGIN
WITH c SELECT
 y <=            e(0) WHEN "00",
                 e(1) WHEN "01",
                 e(2) WHEN "10",
                 e(4) WHEN "11",
                 '0'  WHEN others; --par défaut
 END arch_MUX ;
 ----------------------------------------
```

Logique combinatoire: Instructions concurrentes

- Structure WITH/ SELECT : *COMPLEMENTS et exemples*

Compléments sur WITH/SELECT

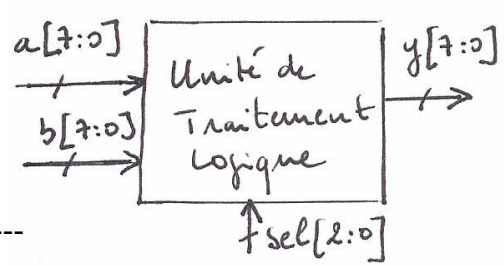
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY maj IS
PORT(a,b,c: IN std_logic; m: OUT std_logic);
END maj;
--Dataflow style architecture
ARCHITECTURE concurrent OF maj IS
BEGIN
 WITH a & b & c SELECT
 m <= '1' WHEN "110"|"101"|"011"|"111",
        '0' WHENOTHERS;
 END concurrent;
```

Je colle 3 bits : a **&** b **&** c pour former un vecteur

Si (a **&** b **&** c) vaut
"110 " OU "101" OU "011" OU "111" alors m <= '1'

Pour exprimer un OU booléen le symbole | est utilisé

Exemple: Unité Logique




```
LIBRARY ieee;
 USE ieee.std_logic_1164.all;
 USE ieee.std_logic_unsigned.all;
-------------------------------------------
 ENTITY LU IS
 PORT (
    a, b: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
    sel: IN STD_LOGIC_VECTOR (2 DOWNTO 0);
    y: OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
 END LU;

 ARCHITECTURE dataflow OF LU IS
 ----- Logic unit: -----------
 WITH sel(2 DOWNTO 0) SELECT
 logic <=        NOT a WHEN "000",
                 NOT b WHEN "001",
                  a AND b WHEN "010",
                 a OR b WHEN "011",
                 a NAND b WHEN "100",
                 a NOR b WHEN "101",
                 a XOR b WHEN "110",
                 NOT (a XOR b) WHEN OTHERS;
END LU;
```

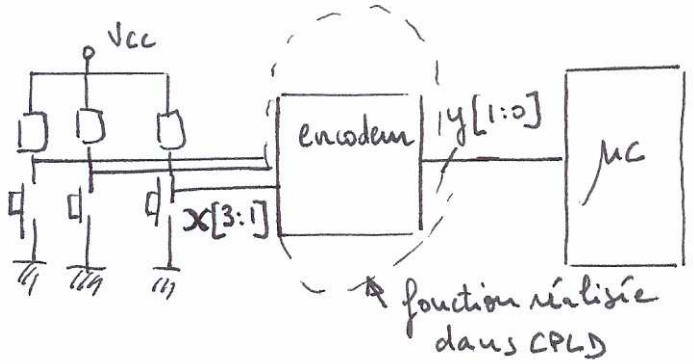
Logique combinatoire: Instructions concurrentes

- **Assignation conditionnelle**
  - Structure WHEN/ELSE

```
signal <=   signal1  when expresion_boolénne else
            .....
            signal1xx  when expresion_boolénne else

            signal par défaut;
```

Exemple: Encodeur

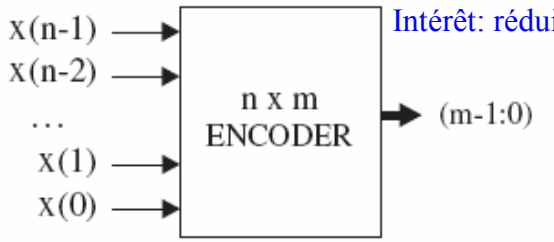
```
---- Solution 1: with WHEN/ELSE ----------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 ------------------------------------------
5 ENTITY encoder IS
6 PORT ( x: IN STD_LOGIC_VECTOR (3 DOWNTO 1);
7 y: OUT STD_LOGIC_VECTOR (1 DOWNTO 0));
8 END encoder;
9 ------------------------------------------
10 ARCHITECTURE encoder1 OF encoder IS
11 BEGIN
12 y <=  "01" WHEN x="110" ELSE – touche X1 appuyée
13         "10" WHEN x="101" ELSE
14         "10" WHEN x="011" ELSE
20         "00"; --par défaut
21 END encoder1;
22 ------------------------------------------
```



Exemple d'application:

encodeur clavier pour PIC

Intérêt: réduire le nombre d'entrée du PIC



Logique combinatoire: Instructions concurrentes

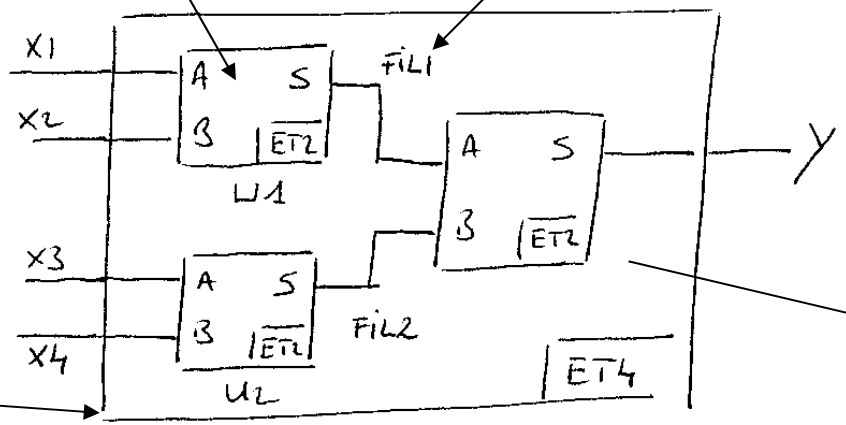
- **Instanciation (*placement*) de composants déjà crées**
  - Découpage de votre projet en fonctions: création de composants adéquats
  - Assemblage des composants créés pour structurer votre projet
  - MOT CLE: PORTMAP
    - ✓ **Ma Référence : *nom du composant* port map (liste ordonnée de signaux) ;**
      - **ou**
    - ✓ **Ma Référence : *nom du composant* port map (port=> signal , port => signal) ;**

Exemple: faire une porte ET 4 entrée avec des ET 2 entrées

Exemple
U1:ET2 port map (A=>X1,B=>X2,S=>FIL1);

Le symbole s'interprète comme « connecté à »

Il faut un SIGNAL



ET4 est un composant (entity+architecture)

Analogie avec ORCAD:
utilisation de composant disponible dans des librairies que vous placez et reliez sur votre feuille

On créera 1 composant ET2 (entity+architecture)

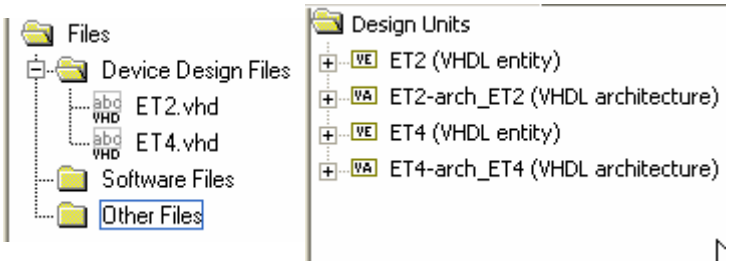
Utilisé 3 fois pour décrire ET4

Logique combinatoire: Instructions concurrentes

Notre ET4

1 Je commence par faire ma ET2

```
--librairie pour inclure type std_logic
library IEEE;
use IEEE.std_logic_1164.all;

ENTITY ET2 IS
                PORT
                (
                A,B: IN         STD_LOGIC;
                S: OUT          STD_LOGIC
                );
END ET2;

ARCHITECTURE arch_ET2 OF ET2 IS
BEGIN
S<= A and B;
END arch_ET2;
```



2 fichiers .vhd avec chacun 1 entity+1 architecture

2 **Je réutilise ET2 dans ma ET4**

```
library IEEE; --librairie pour inclure type std_logic
use IEEE.std_logic_1164.all;

ENTITY ET4 IS
                PORT
                (X1,X2,X3,X4    : IN
                STD_LOGIC;
                Y: OUT          STD_LOGIC
                );
END ET4;

ARCHITECTURE arch_ET4 OF ET4 IS
-- partie déclarative COMPOSANT
COMPONENT ET2 is
PORT
                (A,B: IN        STD_LOGIC;
                S: OUT          STD_LOGIC);
END COMPONENT ET2;
-- partie déclarative SIGNAL
--pas de IN ou OUT car signal INTERNE
SIGNAL FIL1,FIL2: STD_LOGIC ;

BEGIN
-----------------------
-- 1ere porte ET placée
U1:ET2 port map (A=>X1,B=>X2,S=>FIL1);
-- 2ème porte ET placée
U2:ET2 port map (A=>X3,B=>X4,S=>FIL2);
-- 3ème porte ET placée
U3:ET2 port map (A=>FIL1,B=>FIL2,S=>Y);
-- on pourrait faire à la place !!!!
-- Y<= X1 and X2 and X3 and X4
END arch_ET4;
```

Je respecte les noms choisis sur papier

On déclare ET2

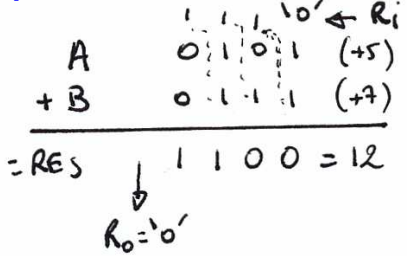
Les fils de connexions INTERNES

PORT MAP pour placement et connexion

Logique combinatoire: Instructions concurrentes

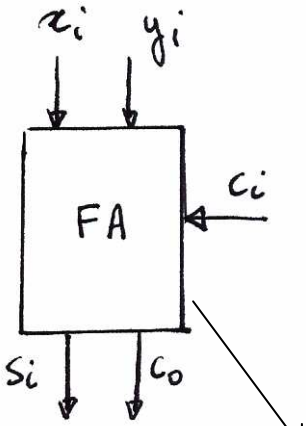
- **Instanciation (*placement*) de composants déjà crées**

exemple: applicatif: Comment faire un additionneur 4 bits?



On peut prévoir une retenue d'entrée initiale et une retenue finale

etape 1: je crée un composant ADDITIONNEUR 1 bits



x_i	y_i	c_i	co	s_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

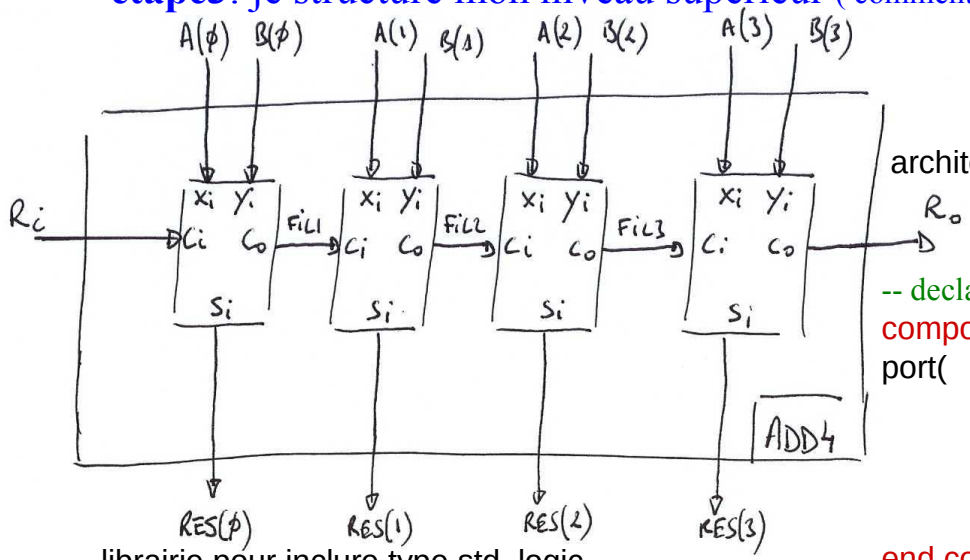
Full **A**dder

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY fa IS PORT (
Ci, Xi, Yi: IN STD_LOGIC;
Co, Si: OUT STD_LOGIC);
END fa;

ARCHITECTURE Dataflow OF fa IS
BEGIN
Co <= (Xi AND Yi) OR (Ci AND (Xi XOR Yi));
Si <= Xi XOR Yi XOR Ci;
END Dataflow;
```

Logique combinatoire: Instructions concurrentes

étape 2: je valide le composant (compilation /simulation)

étape3: je structure mon niveau supérieur (comment faire 4 bits avec 1 bit?)




```
--librairie pour inclure type std_logic
library IEEE;
use IEEE.std_logic_1164.all;
entity add4full is
port(
                Ri:IN std_logic;
                A:IN std_logic_vector(3 downto 0);
                B:IN std_logic_vector(3 downto 0);
                RES:OUT std_logic_vector(3 downto 0);
                Ro:OUT std_logic
                );
end add4full;
```

```
architecture arch_add4full of add4full is
-- declaration  du composant add1full
component add1full is
port(
                Ci:IN std_logic;
                Xi,Yi:IN std_logic;
                Si,Co:OUT std_logic
                );
end component add1full;
-- declaration des fils internes pour le report carry
signal Fil1,Fil2,Fil3:std_logic;
begin
-- placement des 4 additionneurs complets
U0: add1full port map (Ci => Ri,Xi=>A(0),Yi=>B(0),Si =>RES(0),Co=>Fil1);
U1: add1full port map (Ci=>Fil1,Xi=>A(1),Yi=>B(1),Si =>RES(1),Co=>Fil2);
U2: add1full port map (Ci =>Fil2,Xi=>A(2),Yi=>B(2),Si =>RES(2),Co=>Fil3);
U3: add1full port map (Ci =>Fil3,Xi=>A(3),Yi=>B(3),Si =>RES(3),Co=>Ro);
end arch_add4full;
```

Logique combinatoire: Instructions concurrentes

- **Application à la carte DE2**
  - On veut éviter le « pin assignement » à la main
  - On souhaite utiliser le fichier de brochage *DE2_pin_assignments.csv*
  - On souhaite pouvoir réutiliser notre travail pour une notre carte avec un minimum de travail

SOLUTION: créer une « coquille vide » dans un fichier avec une entity compatible avec *DE2_pin_assignments.csv* et une architecture vide!

```
library IEEE;
use IEEE.std_logic_1164.all;

-- TOP LEVEL CARTEDE2


entity CARTEDE2_TOP is
port(
    -- compatible fichier DE2_pin_assignments.csv
    SW:IN std_logic_vector(17 downto 0);
    HEX0,HEX1,HEX2:OUT  std_logic_vector(6 downto 0)
    );
end CARTEDE2_TOP;


architecture arch_test of CARTEDE2_TOP is

component xxxxxxx is
port(
    --à compléter
    );
end component xxxxx;

begin

--à compléter
u0: xxxxx port map (......);
u1: xxxxx port map (......);
end arch_test;
```

Vos composants que vous souhaitez tester:

Additionneur, decodeur 7 segments, leds etc…..

Logique combinatoire: exemples

Mémoire ROM

- Utilisation d'une LUT

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY rom8x8 IS

  PORT (
    adresse : IN  std_logic_vector(2 DOWNTO 0);
    donnees : OUT std_logic_vector(7 DOWNTO 0));
END rom8x8;


ARCHITECTURE arch_ROM OF rom8x8 IS
  TYPE tableau IS ARRAY (0 TO 7) OF std_logic_vector(7 DOWNTO 0);
  CONSTANT rom : tableau :=
    (
     X"42", X"6F", X"6E", X"6A", X"6F", X"75", X"72",X"00");
  -- B      o      n      j      o      u     r

  SIGNAL adr : integer range 0 to 7;

BEGIN
  -- convertion type std_logic vers integer
  adr <= to_integer(unsigned(adresse));-- mettre USE ieee.numeric_std.ALL;
  donnees <= rom(adr);
END arch_ROM;
```

L'indice d'un tableau doit être un entier

Il faut réaliser dans notre cas une double conversion:

std_logic -> unsigned : unsigned(….)

unsigned -> integer: To_integer(…..)

Pour cela: inclure USE ieee.numeric_std.ALL;

Création de nouveaux types: TYPE

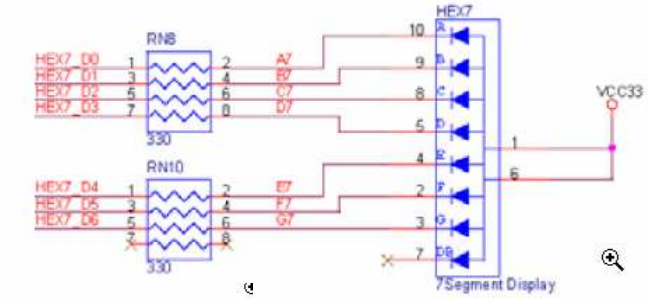
Tableau: ARRAY

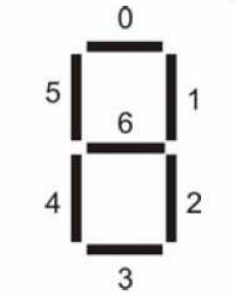
Logique combinatoire: exemples

Décodeurs 7 segments

- UTILISATION D'UN TABLEAU (*LUT*) POUR DECRIRE LE SYSTEME

```
entity decod7seg is
port(
  iDigit:IN std_logic_vector(3 downto 0);
  oSeg:OUT  std_logic_vector(6 downto 0)
    );
end decod7seg;
```





Carte DE2:

✓Anode commune

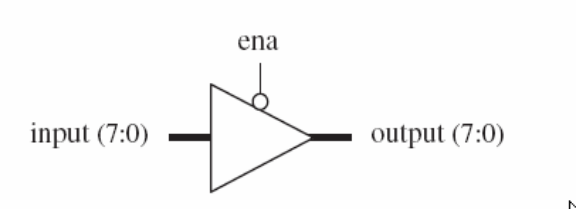
✓Segment actif à '0'

✓Brochage: voir p31 du manuel

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE. ieee.numeric_std.ALL;

architecture arch_dec_7seg_v1 of decod7seg is

-- definition  d'un nouveau type
-- tableau de 16 elements de 7 bits
type ROM is array(15 downto 0) of std_logic_vector(6 downto 0);

--initialisation du tableau
-- tableau vu comme une memoire(LUT)
 signal LUT:ROM:=("1000000","1111001","0100100","0110000",
"0011001","0010010","0000010","1111000","0000000","0011000","000
1000","0000011","1000110","0100001","0000110","0001110");

begin
-- pour indexer tableau il faut un entier
-- fonction de conversion conv_integer et unsigned dans
-- ieee.numeric_std.ALL;
oSeg<=LUT(conv_integer(unsigned(iDigit)));

end arch_dec_7seg_v1;
```

Logique combinatoire: exemples

Buffers 3 états

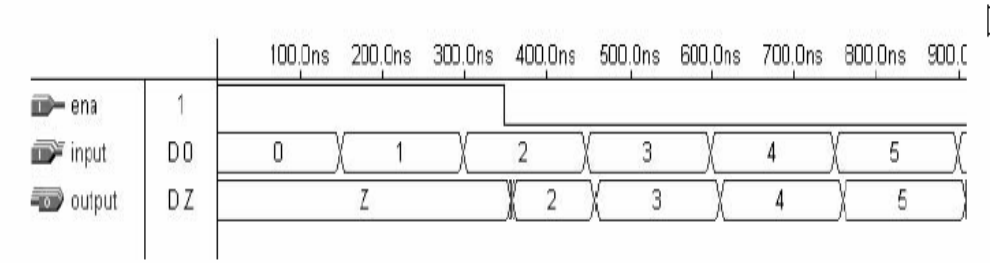


De manière générale il faut se poser la question:

Le composant cible dispose t'il des ressources nécessaires pour synthétiser ma fonction

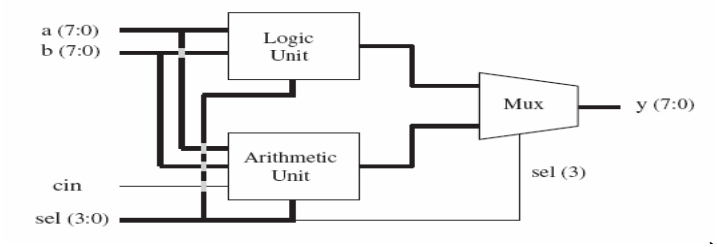
Pas de 3 états possibles si le composants n'en a pas!!

```
1 LIBRARY ieee;
2 USE ieee.std_logic_1164.all;
3 ---------------------------------------------
4 ENTITY tri_state IS
5 PORT (
 ena: IN STD_LOGIC;
6 input: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
7 output: OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
8 END tri_state;
9 ---------------------------------------------
10 ARCHITECTURE tri_state OF tri_state IS
11 BEGIN
12 output <= input WHEN (ena='0') ELSE
13                (OTHERS => 'Z');
14 END tri_state;
15 ---------------------------------------------
```



Logique combinatoire: exemples

Arithmetic Logic Unit (ALU)



sel	Operation	Function	Unit
0000	y <= a	Transfer a	Arithmetic
0001	y <= a+1	Increment a	
0010	y <= a-1	Decrement a	
0011	y <= b	Transfer b	
0100	y <= b+1	Increment b	
0101	y <= b-1	Decrement b	
0110	y <= a+b	Add a and b	
0111	y <= a+b+cin	Add a and b with carry	
1000	y <= NOT a	Complement a	Logic
1001	y <= NOT b	Complement b	
1010	y <= a AND b	AND	
1011	y <= a OR b	OR	
1100	y <= a NAND b	NAND	
1101	y <= a NOR b	NOR	
1110	y <= a XOR b	XOR	
1111	y <= a XNOR b	XNOR	

Sélection Op arithmétique/logique

Code Opératoire (mot de commande sur 3 bits)

```
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 USE ieee.std_logic_unsigned.all;
5 ---------------------------------------------
6 ENTITY ALU IS
7 PORT (a, b: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
8 sel: IN STD_LOGIC_VECTOR (3 DOWNTO 0);
9 cin: IN STD_LOGIC;
10 y: OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
11 END ALU;
12 ---------------------------------------------
13 ARCHITECTURE dataflow OF ALU IS
14 SIGNAL arith, logic: STD_LOGIC_VECTOR (7 DOWNTO 0);
15 BEGIN
16 ----- Arithmetic unit: ------
17 WITH sel(2 DOWNTO 0) SELECT
18 arith <=        a WHEN "000",
19                  a+1 WHEN "001",
20                  a-1 WHEN "010",
21                  b WHEN "011",
22                 b+1 WHEN "100",
                   b-1 WHEN "101",
24                  a+b WHEN "110",
25                 a+b+cin WHEN OTHERS;
```

```
26 ----- Logic unit: -----------
27 WITH sel(2 DOWNTO 0) SELECT
28 logic <=        NOT a WHEN "000",
29                NOT b WHEN "001",
30                 a AND b WHEN "010",
31                a OR b WHEN "011",
32                a NAND b WHEN "100",
33                a NOR b WHEN "101",
34                a XOR b WHEN "110",
35                NOT (a XOR b) WHEN OTHERS;
36 -------- Mux: ---------------
37 WITH sel(3) SELECT
38   y <=          arith WHEN '0',
39                 logic WHEN OTHERS;
40 END dataflow;
41 ---------------------------------------------
```

VHDL: concepts de base

Complément sur les opérations arithmétiques

- Le rajout de use IEEE.numeric_std.all; permet
  - ✓ De travailler avec des valeurs signées et non signées
    - signal A,B: signed(3 downto 0);
    - signal tempo: unsigned(3 downto 0);
  - ✓ De convertir un std_logic_vector en signed ou unsigned
    - A<= signed(SW(3 downto 0));
    - B<=unsigned(RES);
  - ✓ De convertir des signed ou unsigned en std_logic_vector
    - LEDG(3 downto 0)<=std_logic_vector(tempo);
  - ✓ De redimensionner des vecteurs
    - Permet d'étendre le bit de signe correctement!
    - signal A,B: signed(LARG downto 0);
      A<= resize(signed(SW(LARG downto 1)),LARG+1);
  - ✓ De travailler avec les opérateurs arithmétiques standart
    - >, >=, =<,<, + ,- etc….
- Le rajout de use IEEE.std_logic_unsigned.all; permet
  - ✓ De travailler avec les opérateurs arithmétiques standard
  - ✓ de mélanger des entiers avec des std_logic_vector: A<= A +1;

IEEE.std_logic_unsigned.all et IEEE.std_logic_arith.all sont d'anciennes bibliothèques

Ne pas mettre en même temps:

IEEE.numeric_std.all;

IEEE.std_logic_arith.all;

Préfèrez l'emploi de IEEE.numeric_std.all;

Alternative à resize

A<=resize(signed(SW(LARG downto 1)),LARG+1);

Recopie du bit bit de poids forts

A<= A(3)&A

Logique combinatoire: exemples

Additionneur « haut niveau »

- Emploi des librairies IEEE;
- L'opérateur + ne gère pas la retenue finale
  - ✓ Astuce: On augmente la taille de 1 si l'on souhaite conserver la retenue d'entrée
  - ✓ Exemple additionneur 8 bits: on travaille sur 9 bits(extension). Le 9ème sera la retenue

```
--librairie pour inclure type std_logic
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity adddirect is
generic (LARG:integer:=4);
-- parametre generique
-- taille aditionneur changer en 1 clic!

port(
Cin:IN std_logic;
A:IN unsigned (LARG-1 downto 0);
B:IN unsigned (A'range);
Res:OUT unsigned(A'range);
Cout:OUT std_logic
  );
end adddirect;
```

```
architecture arch1_add4full of adddirect is
--creation de TEMP pour resultat: extension de 1 bit
signal TEMP: unsigned(LARG downto 0);
begin
TEMP<=('0'&A)+('0'&B)+Cin; --A et B etendu de 1 bit.
Res<=TEMP(TEMP'HIGH-1 downto 0);
Cout<=TEMP(TEMP'HIGH);
--TEMP'HIGH renvoi indice poids fort
end arch1_add4full;
```

Les attributs des signaux -Exemple S[5:0]

S'HIGH renvoie 5 et S'LOW renvoie 0

S'RANGE renvoie 5 downto 0

S'event renvoie TRUE si changement d'état de S

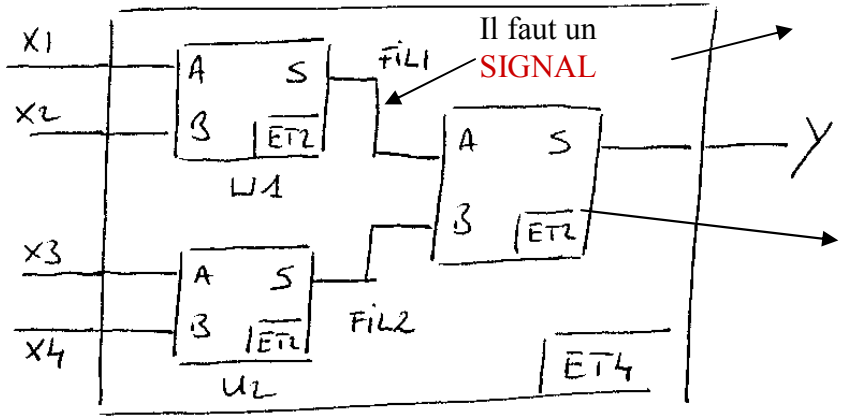
Utilisation de GENERIC lors du PORT MAP

```
U1: generic(10)
    adddirect PORT MAP(xxxxxx);
```

BILAN

- **Les questions à se poser**
  - On identifie les fonctions et on les dessine *sur papier*
  - On repère et nomme les entrées de chaque blocs (on évite d'utiliser les mêmes noms)
  - On répertorie les signaux INTERNES (mot clé SIGNAL)
  - Le bloc est-il combinatoire ou séquentiel?
    - ✓ Si séquentiel alors description avec le mot clé PROCESS + instructions autorisées
  - Le bloc est-il utilisé plusieurs fois
    - ✓ Si oui il vaut mieux créer un composant (entity+ architecture)
    - ✓ Sinon le bloc est synthétiser par les lignes de codes directement

Exemple: faire une porte ET 4entrée avec des ET 2 entrées



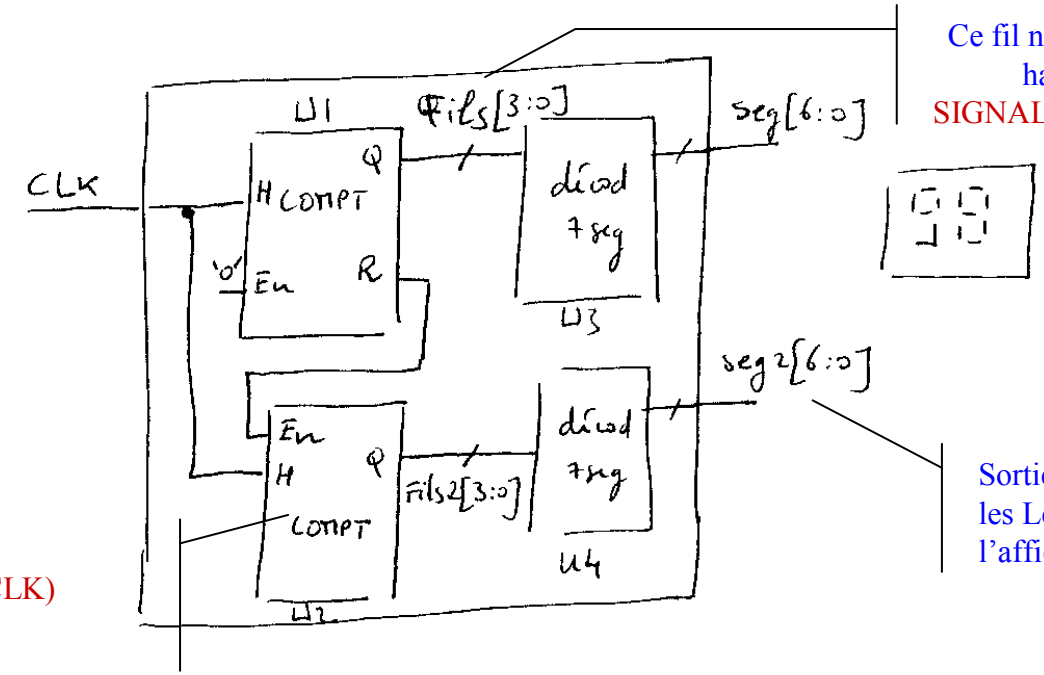
ET4 est un composant (entity+architecture)

On créera 1 composant ET2 (entity+architecture)

Utilisé 3 fois pour décrire ET4

BILAN

- **Un exemple : Horloge BCD 2 digits**
  - Blocs décodeurs 7 segments - *combinatoire*
  - Blocs compteurs – *séquentiel*
  - Les Blocs compteurs sont cascadé pour la propagation de la retenue
    - ✓ Après 9 j'ai 0 avec un de retenue! Affichage 10



Ce fil ne sort pas du composant de plus haut niveau: on le déclarera
SIGNAL FILS: bit_vector(3 downto 0);

Sorties pilotant les Leds de l'afficheur 1

Séquentiel donc process

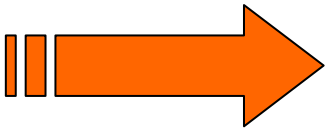
```
COMPT:    PROCESS(CLK)
          Begin
          ….
          END PROCESS;
```

BILAN

Les conseils

- Pour décrire des systèmes combinatoires les instructions types « concurrentes » seront préférées
- L'ordre des instructions est SANS IMPORTANCE (car en parallèle)
- Il est souhaite de scinder les projets en composants simples
  - ✓ APPROCHE METHODOLOGIQUE TOP-DOWN
- Utilisation des bibliothèques IEEE

Squelette de description VHDL



```
--les libraries
library IEEE;
use IEEE.std_logic_1164.all;
…….

ENTITY LENIVEAUTOP (
……..)
End ENTITY

ARCHITECTURE …..

COMPONENT Truc
…
END COMPONENT
COMPONENT Machin
…
END COMPONENT

SIGNAL: …….
SIGNAL:  ……..

XX<=“1110”;
YY<= A AND B;
U1: Truc PORT MAP( …….);
S<= “10” when (A=B) else
       “00”;
U2: Machin PORT MAP( …….);

With (Toto) select
 G<= ……

END ARCHITECTURE
```

Déclaration de composants créés

Utilisation des ressources disponibles

Bibliographie

- Certaines illustrations et exemples proviennent de cours ou d'ouvrages présents ci-dessous

  - ✓ Introduction à la Synthèse logique **Philippe LECARDONNEL & Philippe LETENNEUR**
  - ✓ Le langage de description VHDL **T. BLOTIN**
  - ✓ VHDl **J.maillefert**
  - ✓ Circuit Design with VHDL **Volnei A. Pedroni**

- **Un lien bien utile pour l'étudiant GEII**
  - ✓ **http://perso.orange.fr/xcotton/electron/coursetdocs.htm**