Cours VHDL - III

L3-S6 - Université de Cergy-Pontoise

Laurent Rodriguez – Benoît Miramond

Plan du cours

I – Historique de conception des circuits intégrés
- **HDL**
  - Modèles de conceptions
- **VHDL**
  - Les modèles de conceptions en VHDL
  - Les 5 briques de base

II – VHDL et FPGA
- **VHDL**
  - **Syntaxe et typage**
  - **Retour sur les briques de bases**
  - **Retour sur la conception structurelle et comportementale en VHDL**
    Port map, Equations logiques, Tables de vérités (With ... Select)
- **FPGA**
  - **Qu'est ce qu'un FPGA**
  - **Flot de conception**
  - **Carte de développement et environnement de TP**

Plan du cours

III – Modélisation

- **Compléments sur la description Structurelle**
- **Description comportementale approfondie**
  - Circuits combinatoires
  - Styles de description
    - Flots de données
    - Instructions concurrentes
    - Table de vérité
    - Fonctions
- **Circuits standards**
  - Comparateur
  - Multiplexeurs
  - Encodeurs
- **Process et gestion du temps**

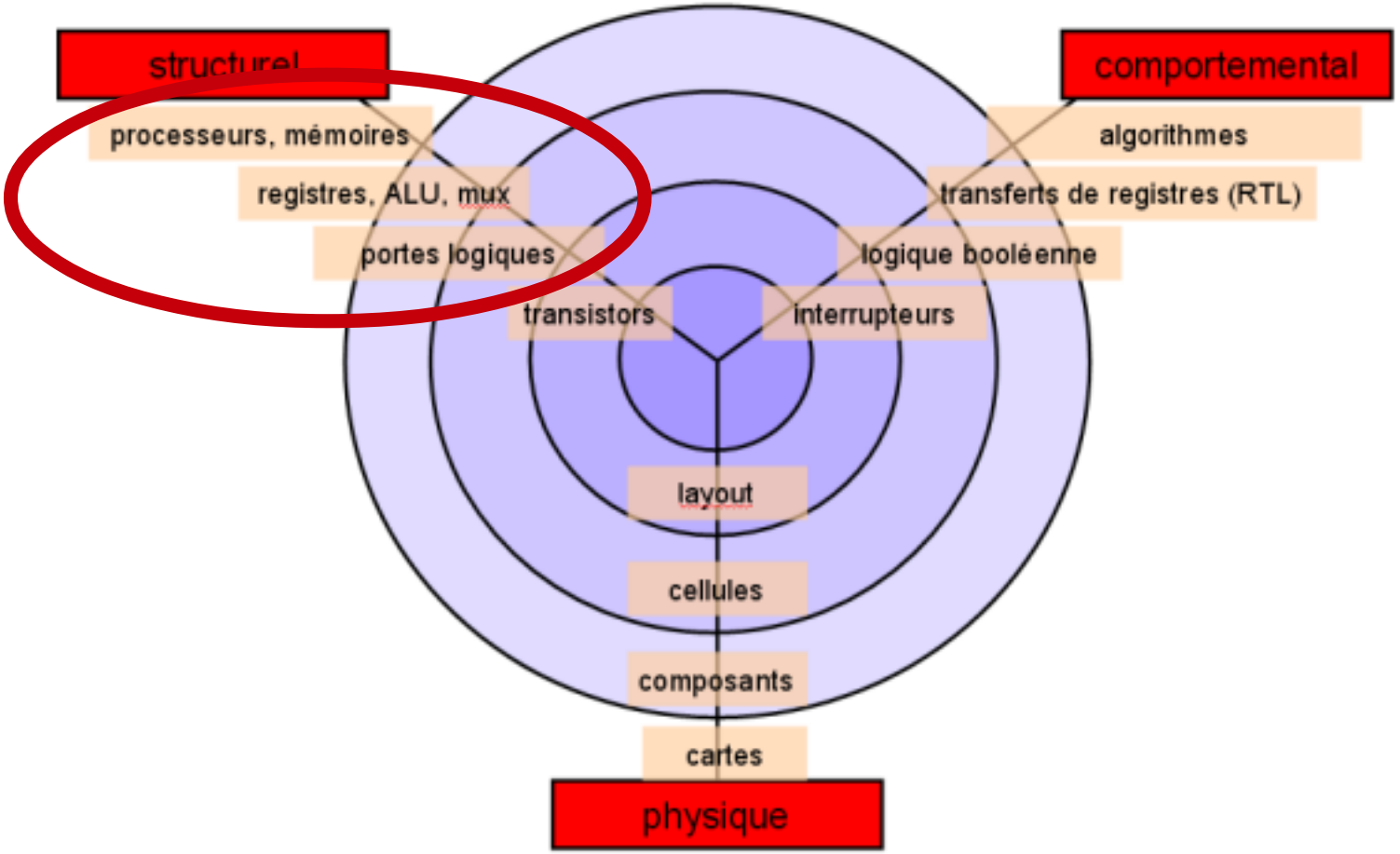
IV – Circuits Génériques et génération de circuits

- **...**

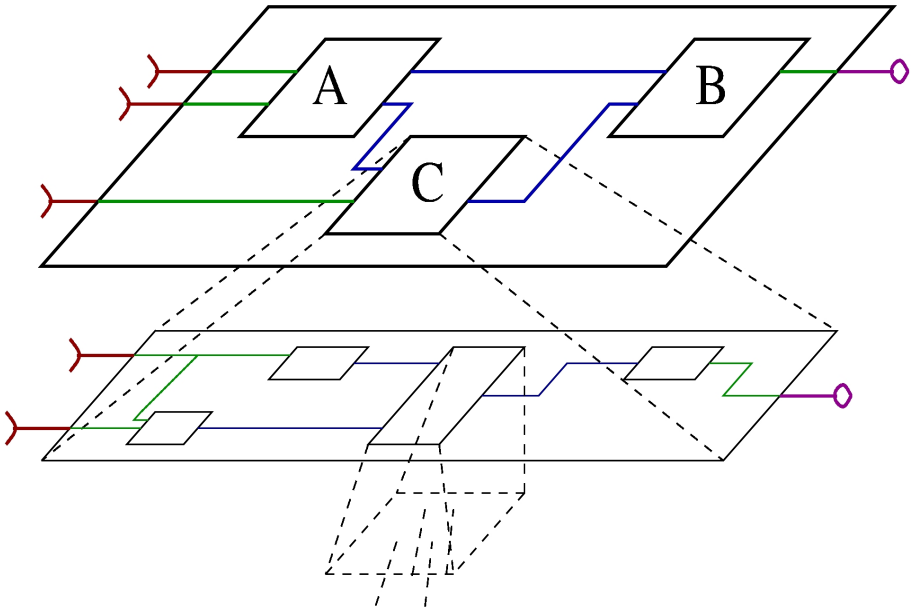
III - Modélisation

Compléments sur la description Structurelle

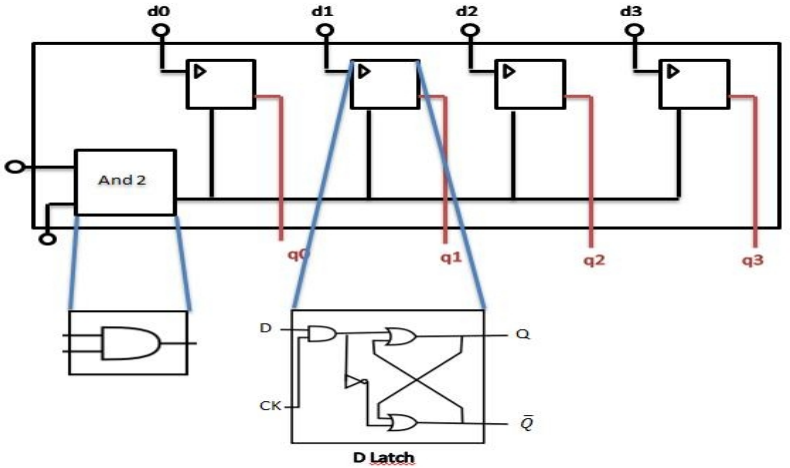
II - description comportementale - Plusieurs niveaux



I - Complément sur la description structurelle



- une description hiérarchique Compréhensible
- Une synthèse logique efficace



I - Complément sur la description structurelle - assemblage des composants

VHDL permet donc l'assemblage de « composants » => description structurelle

- Un composant peut être appelé plusieurs fois dans un même circuit
- Pour différencier ces même composant il faut donner un nom d'instance
- l'appel d'un composant se dit aussi « instanciation »

Pour instancier un composant il faut connaitre :

- Le prototype du composant (ports d' E/S). La directive COMPONENT le permet
- L'unité de configuration permet de choisir l'architecture utilisé pour chaque instance de composant

<NOM_INSTANCE>:<NOM_COMPOSANT> port map(LISTE DES CONNEXIONS);

A noter :

- La déclaration du « component » est redondante avec celle de l'entité associée
  - compilation indépendante entre l'entité du composant et le circuit l'utilisant
  - conception descendante (le composant peut être déclaré avant l'entité

- La configuration est optionnelle mais peut être utile (ex : pour l'accélération de simulations)

I - Complément sur la description structurelle - assemblage des composants

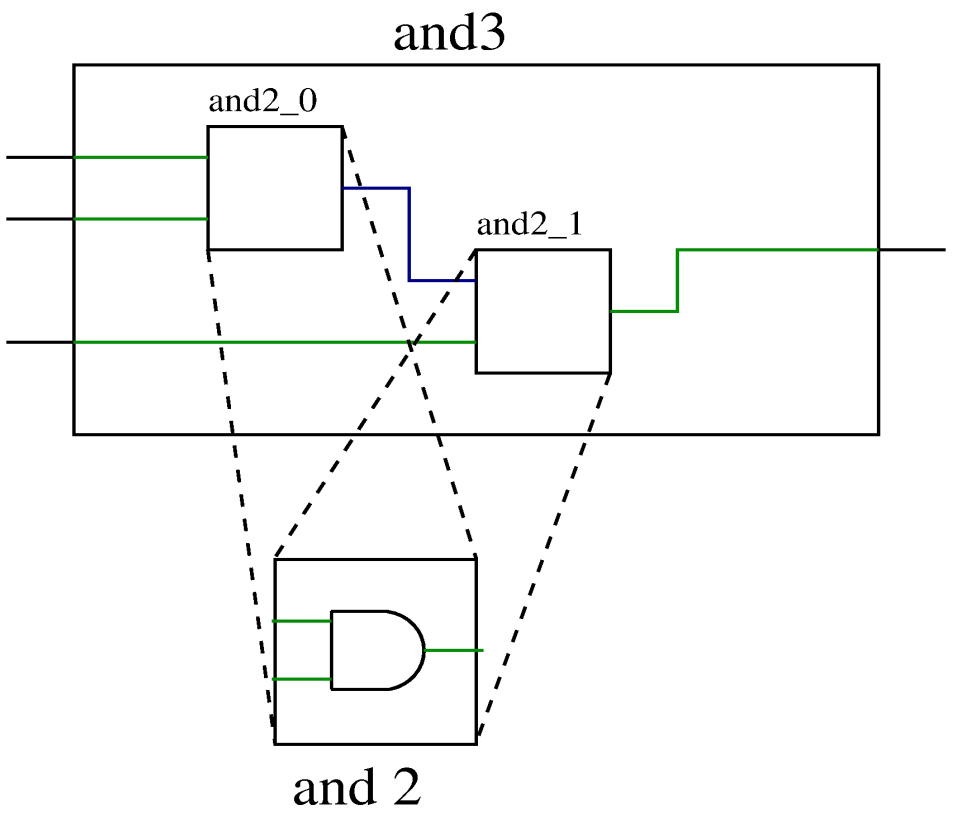
```
ENTITY and3 IS
 PORT(
 e1 : IN BIT;
 e2 : IN BIT;
 e3 : IN BIT;
 s : OUT BIT
 );
END ENTITY;


ARCHITECTURE arc_and3 OF and3 IS

SIGNAL z : BIT;

COMPONENT and2
        PORT (
        a : BIT;
        b : BIT;
        s : BIT);
END COMPONENT;

BEGIN
 and2_0 : and2 PORT MAP (a=>e1, b=>e2 , s=>z);
 and2_1 : and2 PORT MAP (Z, E3, S);
END ARC
```



III - Modélisation

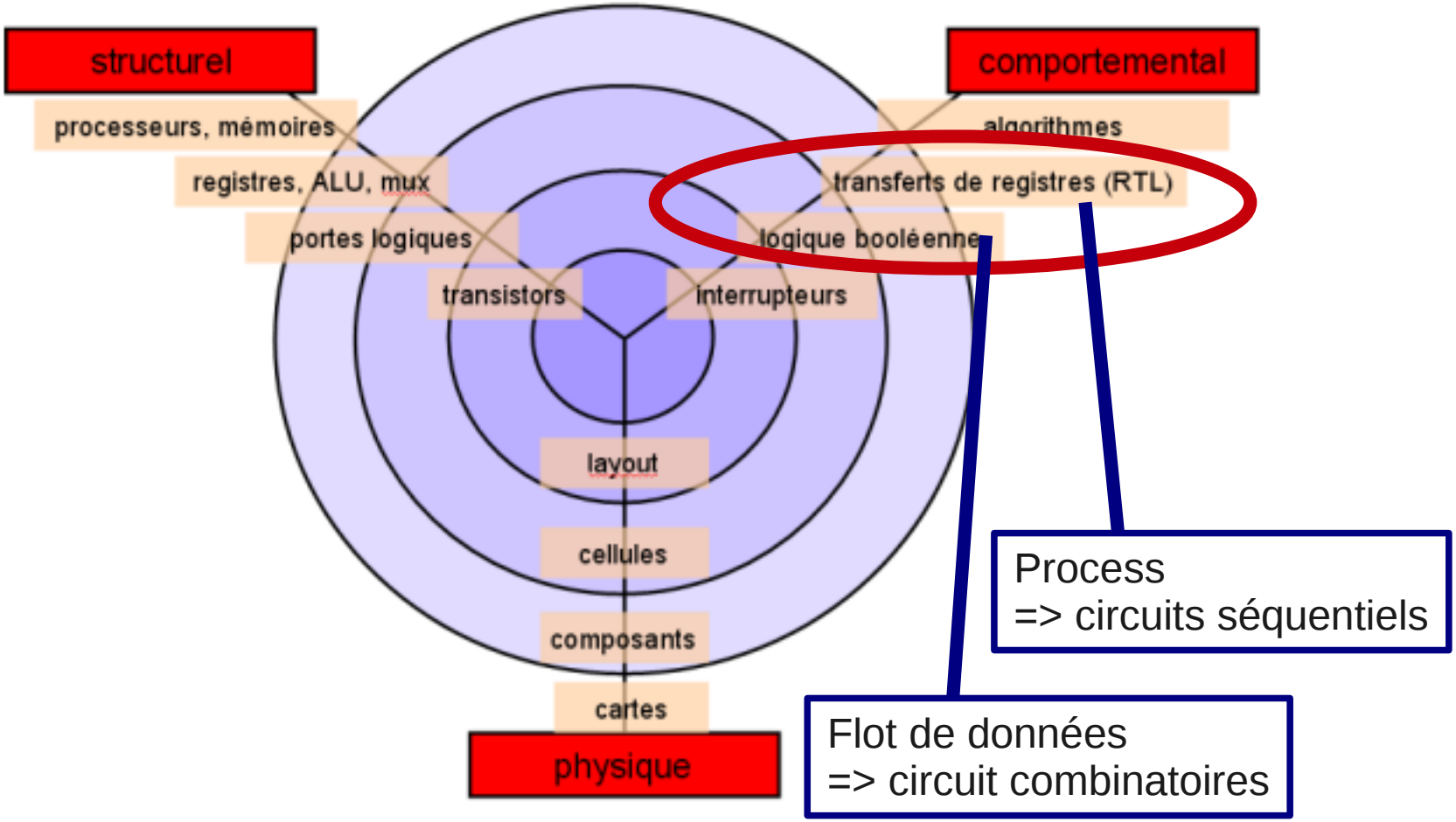
Description comportementale : circuits combinatoires

Objectifs du chapitre

Les objectifs de ce chapitre sont de comprendre :

- Les différences entre instructions concurrentes et séquentielles.
- Les processus et les différences entre variables et signal.
- Les structures de contrôle utilisées dans les processus.
- Les affectations concurrentes des signaux, les raccourcis d'écriture.
- Les différents types de délais dans les affectations.
- Comment vérifier une propriété par les assertions.
- Les différents types de sous-programmes.

II - description comportementale - Plusieurs niveaux



II - description comportementale - Circuit combinatoires

- Ne possède pas d'élément de mémorisation et donc ne présente pas d'état interne.
- Sont les principaux responsables des délais apparaissant dans un circuit => pour la simulation les instructions concurrentes sont munies d'une échéance représentant le délai de propagation du signal.

Plusieurs manières de décrire un circuit combinatoire :

- Par instructions concurrentes (manipulant des signaux)
  - Les équations logiques (ou flot de données)
  - Par les tables de vérités
  - Sous-programmes : Procédures, fonctions
  - assertions
- Par processus (détaillé dans le prochain cours) – plus naturel pour l'humain – munis d'une liste de sensibilité devant contenir tous les signaux lus dans le corps du processus – (manipulant signaux mais aussi des variables internes)

II - description comportementale - Signaux

- Dans les instructions concurrentes, on manipule des signaux :
  - Les signaux sont affectés avec l'instruction <= qui se dit aussi "reçoit" plutôt que "égal" .
  - Il peuvent être vu comme des variables globales typées permettant de connecter différentes structures internes ou de décrire un comportement en assemblant des signaux de même type via des opérateurs.
  - Pour la simulation on spécifie les délais de propagation « AFTER ».

II - description comportementale - **Styles :** Flot de données

Porte AND en flot de données et équivalent en processus

```
entity and2 is
    generic (Tprop: TIME := 0 ns);
    port (A, B: in BIT; Q: out BIT);
end entity and2;
```

```
architecture dfl1 of and2 is
begin
    Q <= A and B after Tprop;
end dfl1;

architecture dfl2 of and2 is
begin
    Q <= '1' after Tprop when A = '1' and B = '1'
              else '0' after Tprop;
end dfl2;
```

```
architecture dfl2 of and2 is
begin
        A <= A and B after Tprop;
end dfl1;
```

Process

```
architecture proc of and2 is
begin
    P_AND2: process (A, B)
    begin
        if A = '1' and B = '1' then
            Q <= '1' after Tprop;
        else
            Q <= '0' after Tprop;
        end if;
    end process P_AND2;
end proc;
```

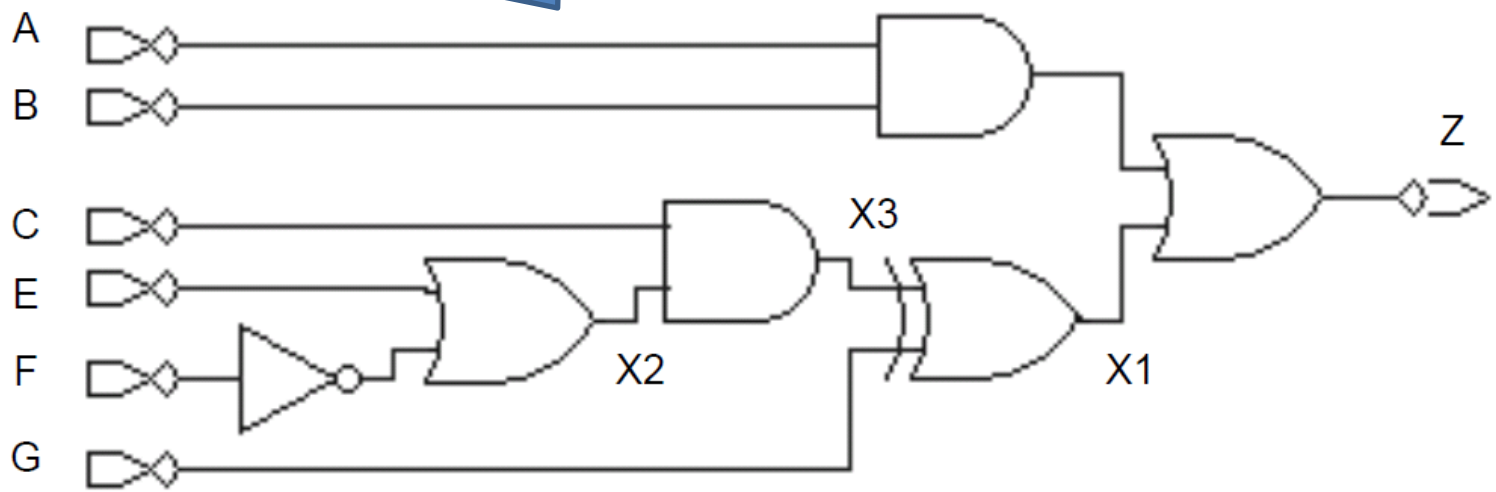
II - description comportementale - **Styles :** Flot de données – instructions concurentes

```
entity rlogic is
    port (A, B, C, E, F, G: in BIT; Z: out BIT);
end entity rlogic;

architecture comb of rlogic is
    signal X1, X2, X3: BIT;
begin
    Z  <= (A and B) or X1;
    X1 <= X3 xor G;
    X2 <= E or (not F);
    X3 <= C and X2;
end architecture comb;
```

```
entity rlogic is
      port(A, B, C, E, F, G : in BIT ; Z : out BIT) ;
end entity rlogic ;

architecture combFD of rlogic is
begin
      Z <= (A and B)
            or ((C and (E or (not F)))
            xor G );
End architecture combFD ;
```



II - description comportementale - Styles : Table de vérités

```
entity fadd1 is
    port (A, B, CI: in BIT; S, CO: out BIT);
end entity fadd1;

architecture tt of fadd1 is
    -- table de vérité:
    --  CI  A  B  S  CO
    --   0  0  0  0  0
    --   0  0  1  1  0
    --   0  1  0  1  0
    --   0  1  1  0  1
    --   1  0  0  1  0
    --   1  0  1  0  1
    --   1  1  0  0  1
    --   1  1  1  1  1

    subtype vec2 is BIT_VECTOR (1 to 2);
    subtype vec3 is BIT_VECTOR (1 to 3);

    signal wout: vec2;
    signal win: vec3;
```

```
begin
    win <= CI & A & B;
    with win select
        wout <= "00" when "000",
                "10" when "001",
                "10" when "010",
                "01" when "011",
                "10" when "100",
                "01" when "101",
                "01" when "110",
                "11" when "111";
    S  <=wout(1);
    CO <= wout(2);
end architecture tt;
```

```
architecture dfl of fadd1 is
begin
    S  <= A xor B xor CI;
    CO <= (A and B) or (A and CI) or (B and CI);
end architecture dfl;
```

Comparaison avec la forme Data-flow

II - description comportementale - Sous-programmes

2 types de sous-programmes:

- **Les fonctions**
- **Les procédures**

Les sous-programmes font partie du domaine séquentiel et ont la même utilité que pour les autres langages de programmation : regrouper les instructions qu'on utilise souvent.

Les procédures font peuvent être appelées aussi bien dans le domaine séquentiel que concurrent.

II - description comportementale - Sous-programmes

Différence entre fonctions et procédures :

	Fonction	Procédure
Paramètres d'appel	Non modifiables (entrées)	Les sorties sont modifiables
Valeur retournée	oui	non
syntaxe	**function** F(A: unsigned(3 downto 0)) **return** std_logic **is** ... **begin** ... **end** function F;	**procedure** P(A: in unsigned(3 downto 0)) S : out std_logic) **is** ... **begin** ... **end** procedure P;

Par rapport à la fonction, la procédure permet d'avoir plusieurs sorties, mais à condition de déclarer les entrées et les sorties comme dans une entité ou composant.

II - description comportementale - Utilisation de fonctions

```
function parity (bv: BIT_VECTOR) return BIT is
    variable result: BIT;
begin
    result := '0';  -- l'initialisation dans la déclaration n'est pas supportée en synthèse
    for i in bv'RANGE loop
        result := result xor bv(i);
    end loop;
    return result;
end function parity;
```

```
use WORK.utils_pkg.all;
architecture A of E is
    signal data: BIT_VECTOR(7 downto 0);
    signal par  : BIT;
begin
    ...
    par <= parity(data);
    ...
end architecture A;
```

II - description comportementale - Utilisation de procédures

```
procedure MinMax( a,b : in unigned  (7 downto 0);
                  min : out unsigned(7 downto 0);
                  max : out unsigned(7 downto 0)) is
begin
 if (a < b) then
        min <= a;
        max <= b;
else
        min <= b;
        max <= a;

end if;
end procedure MinMax;


...


MinMax(x, y, z, t);
```

```
function min (a,b : unsigned(7 downto 0)
 return unsigned(7 downto 0) is
 varaiable min : unsigned(7 downto0);
 begin
     if (a < b) then
        min := a;
     else
        min := b;
     end if;
     return min;
endfunction

...

z := min(x, y);
```

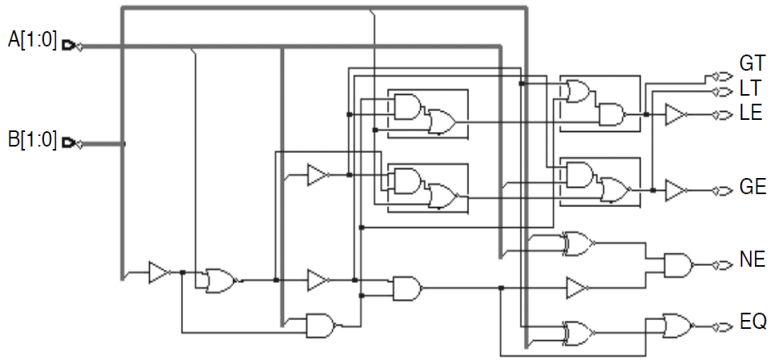
III - Modélisation

Description comportementale : circuits standards

III – Circuits standards - Comparateur

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
entity comparator is
    generic (NBits: POSITIVE := 2);
    port (A, B: in std_logic_vector(NBits-1 downto 0);
          EQ, GT, LT, NE, GE, LE: out std_logic);
end comparator;

architecture bhv of comparator is
begin
    process (A, B)
    begin
        if unsigned(A) > unsigned(B) then
            EQ <= '0'; LT <= '0'; LE <= '0'; GT <= '1'; NE <= '1'; GE <= '1';
        elsif unsigned(A) < unsigned(B) then
            EQ <= '0'; GT <= '0'; GE <= '0'; LT <= '1'; NE <= '1'; LE <= '1';
        else
            NE <= '0'; GT <= '0'; LT <= '0'; EQ <= '1'; GE <= '1'; LE <= '1';
        end if;
    end process;
end bhv;
```



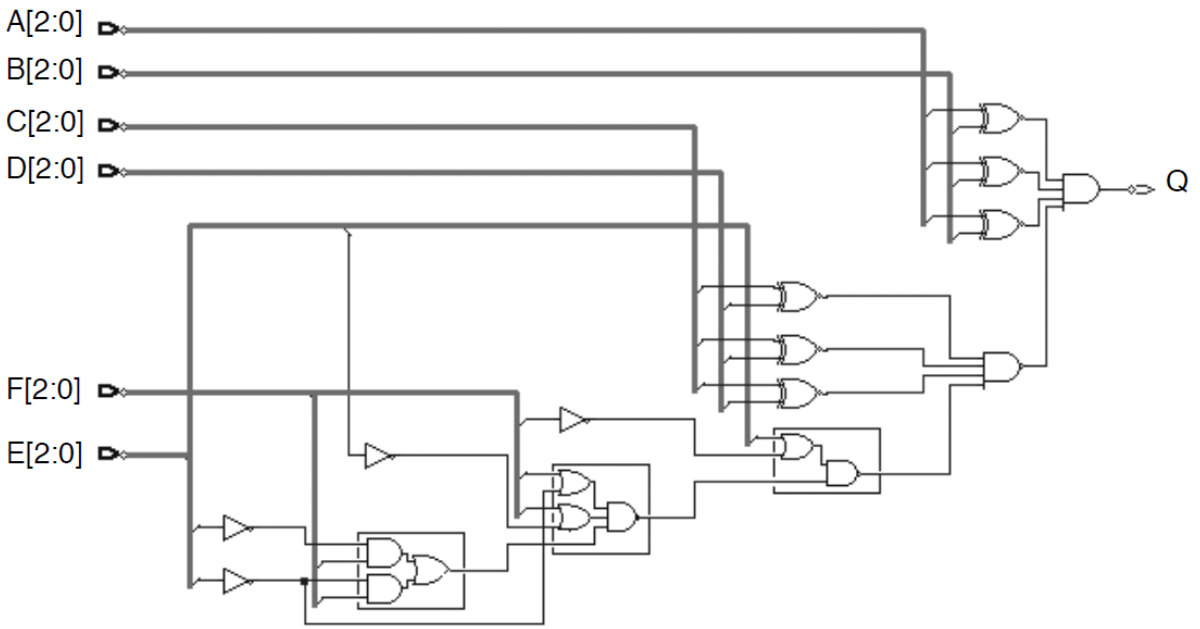
Circuit équivalent pour Nbits = 2

III – Circuits standards – Comparateur 2

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

entity comparator2 is
    port (A, B, C, D, E, F: in unsigned(2 downto 0);
          Q: out std_logic);
end comparator2;

architecture dfl of comparator2 is
begin
    Q <= '1' when (A = B and (C /= D or E >= F)) else '0';
end dfl;
```



III – Circuits standards – Multiplexeurs

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

entity mux41 is
    port (SEL: in   std_logic_vector(1 downto 0);
          INP : in   std_logic_vector(3 downto 0);
          Q:    out std_logic);
end mux41;
```

```
architecture dfl1 of mux41 is
begin
    Q <= INP(3) when SEL = "00" else
         INP(2) when SEL = "01" else
         INP(1) when SEL = "10" else
         INP(0);  -- when SEL = "11"
end dfl1;
```

```
architecture proc1 of mux41 is
begin
    process (SEL, INP)
    begin
        if SEL = "00" then
            Q <= INP(3);
        elsif SEL = "01" then
            Q <= INP(2);
        elsif SEL = "10" then
            Q <= INP(1);
        else -- SEL = "00"
            Q <= INP(0);
        end if;
    end process;
end proc1;
```

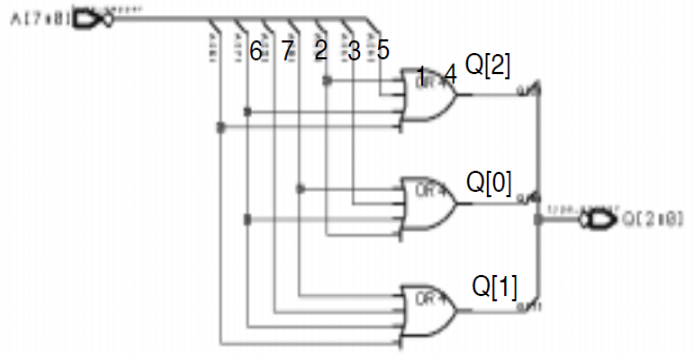
III – Circuits standards – Multiplexeurs

```
architecture dfl2 of mux41 is
begin
    with SEL select
        Q <= INP(3) when "00",
             INP(2) when "01",
             INP(1) when "10",
             INP(0) when "11",
             INP(3) when others;
end dfl2;
```

```
architecture proc2 of mux41 is
begin
    process (SEL, INP)
    begin
        case SEL is
            when "00"     => Q <= INP(3);
            when "01"     => Q <= INP(2);
            when "10"     => Q <= INP(1);
            when "00"     => Q <= INP(0);
            when others => Q <= INP(3);
        end case;
    end process;
end proc2;
```

Forme recommandée pour la synthèse

III – Circuits standards – Encodeurs



Entrées								Sorties		
A7	A6	A5	A4	A3	A2	A1	A0	Q2	Q1	Q0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

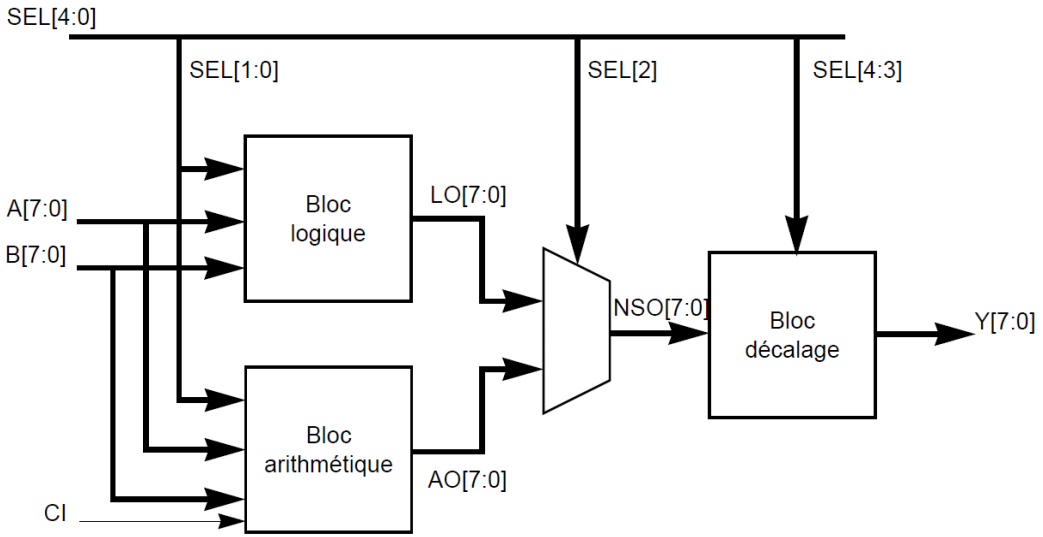
```
library IEEE;
use IEEE.std_logic_1164.all;

entity encoder83 is
    port (A: in  std_logic_vector(7 downto 0);
          Q: out std_logic_vector(2 downto 0));
end encoder83;

architecture csel of encoder83 is
begin
    with A select
        Q <= "000" when "00000001",
             "001" when "00000010",
             "010" when "00000100",
             "011" when "00001000",
             "100" when "00010000",
             "101" when "00100000",
             "110" when "01000000",
             "111" when "10000000",
             "---"  when others;
end csel:
```

III – Circuits standards – ALU

S4	S3	S2	S1	S0	CI	Opération	Fonction	Unité
0	0	0	0	0	0	Y <= A	Transfert de A	arithmétique
0	0	0	0	0	1	Y <= A + 1	Incrémentation de A	
0	0	0	0	1	0	Y <= A + B	Addition	
0	0	0	0	1	1	Y <= A + $\overline{B}$ + 1	Addition avec retenue	
0	0	0	1	0	0	Y <= A + $\overline{B}$	A + compl. à 1 de B	
0	0	0	1	0	1	Y <= A + B + 1	Soustraction	
0	0	0	1	1	0	Y <= A - 1	Décrémentation de A	
0	0	0	1	1	1	Y <= B	Transfert de B	
0	0	1	0	0	0	Y <= A and B	AND	logique
0	0	1	0	1	0	Y <= A or B	OR	
0	0	1	1	0	0	Y <= $\overline{A}$ xor B	XOR	
0	0	1	1	1	0	Y <= $\overline{A}$	Complément	
0	0	0	0	0	0	Y <= A	Transfert de A	décalage
0	1	0	0	0	0	Y <= shl A	Décalage à gauche de A	
1	0	0	0	0	0	Y <= shr A	Décalage à droite de A	
1	1	0	0	0	0	Y <= 0	Transfert de zéros	



III – Circuits standards – ALU

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

package ALU_PKG is
	type alu_op is (ADD,SUB,INC1,DEC1,LNOT,LAND,LOR,LSHR);
end package PAQ;

use work.ALU_PKG.all;

entity ALU is
	generic(NBits: POSITIVE);
	port(A,B: in unsigned(NBits-1 downto 0);
		Q: out unsigned(NBits-1 downto 0);
		OP: in alu_op);
end ALU;

architecture dfl of ALU is
begin
		--oups
end ALU;
```

Implémentation délicate à optimiser par le synthétiseur à cause de la forme when ...

III – Circuits standards – ALU 2

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

entity ALU is
    port (
        SEL: in  unsigned(4 downto 0);   -- sélecteur de l'opération
        CI  : in  std_logic;             -- retenue d'entrée
        A, B: in  unsigned(7 downto 0);  -- opérandes
        Y   : out unsigned(7 downto 0)   -- résultat
    );
end entity ALU;
```

```
architecture str of ALU is
    component ulogic
        port (SEL: in  unsigned(1 downto 0);
              A, B : in  unsigned(7 downto 0);
              LO   : out unsigned(7 downto 0));
    end;
    component uarith
        port (SEL: in  unsigned(1 downto 0);
              CI   : in  std_logic;
              A, B : in  unsigned(7 downto 0);
              AO   : out unsigned(7 downto 0));
    end;
    component ushift
        port (SEL: in  unsigned(1 downto 0);
              A    : in  unsigned(7 downto 0);
              SO   : out unsigned(7 downto 0));
    end;
    signal LO, AO, NSO: unsigned(7 downto 0);
begin
    logic_unit: ulogic port map (SEL(1:0), A, B, LO);
    arith_unit: uarith port map (SEL(1:0), CI, A, B, AO);
    mux: NSO <= LO when SEL(2) = '1' else AO;
    shift_unit : ushift port map (SEL(4:3), NSO, Y);
end architecture str;
```

Biblio

Documents de cours

- Ce cours (bientot) en ligne à :

Http://perso-etis.ensea.fr/rodriguez/

- Un très bon support de cours

http://hdl.telecom-paristech.fr/index.html

- Le cours de Licence 2 (en particulier pour ceux qui ne l'ont pas suivi):

http://perso-etis.ensea.fr/miramond/Enseignement/L2/circuits_numeriques.html

- Le cours de Licence 3 de 2013 :

http://perso-etis.ensea.fr/miramond/Enseignement/L3/Cours_VHDL_2011.html

Documents de développement

- Quartus

http://quartushelp.altera.com/current/

- Documentation Altera sur les Cyclones II et IV (entre autre...)

http://www.altera.com/literature/lit-index.html