Rappel

Entité

```
entity nom is
-- les ports d'entrées/sorties
port (in1 : in std_logic;
      in2 : in std_logic;
      -- port_nom: sens(in/out/inout) type;
      ...
      out1 : out std_logic;
      out2 : out std_logic;
      ...
      outn : out std_logic
      );
end;
-- end entity;
-- end nom;
```

Rappel

Architecture

```
architecture arch_nom of nom is
   -- partie déclarative de l'architecture
   signal sig1, sig2, ..., sign : std_logic;
   -- signal nom : type;

   -- utilisation de composants
   component comp_nom
   ...
   end component;

   begin
   ...
   end;
   -- end nom;
```

Rappel

Instructions

- concurrentes
  → en dehors d'un `process`
- séquentielles
  → à l'intérieur d'un `process`

```
architecture arch_nom of nom is
   signal sig1, sig2: std_logic;
   begin
   -- instruction concurrente
   sig1 <= a xor b;
   --instruction séquentielle
   process(a,b) --process(all)
   begin
      sig2 <= a xor b;
   end process;
   end;
```

Rappel

Instructions

concurrent

```
architecture arch_nom of nom is
begin
with sel select
  sortie <= in1 when "00",
            in2 when "01",
        in3 when "10",
        ...
        in1 when others;
end;
```

séquentiel

```
architecture arch_nom of nom is
begin
  process(all)
  begin
  case sel is
     when "00" => sortie <= in1;
     when "01" => sortie <= in2;
     when "10" => sortie <= in3;
     ...
     when others => sortie <= in1;
  end case;
  end process;
end;
```

Rappel

Instructions

concurrent

```
architecture arch_nom of nom is
begin
  sortie <= in1 when sig1 = '0'
     else in2;
end;
```

séquentiel

```
architecture arch_nom of nom is
begin
  process(all)
  begin
    if sig1='0' then
       sortie <= in1;
    else
       sortie <= in2;
    end if;
  end process;
end;
```

Paramètres génériques

```
entity nom is
-- paramètres génériques
generic (param1: integer:=10;
         param2: integer:=10;
 ...
         paramn: integer:=10
);
-- les ports d'entrées/sorties
port (in1 : in std_logic_vector(param1-1 downto 0);
      out1 : out std_logic_vector(param2-1 downto 0);
      ...
      outn : out std_logic
      );
end;
```

Le multiplexeur

Exemple de description générique

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

entity MuxGenerique is
generic ( TAILLE : integer := 8 );
port (
      in0, in1, in2, in3 : in
    std_logic_vector(TAILLE-1 downto 0);
      Selecteur : in std_logic_vector(1 downto 0);
      Sortie : out std_logic_vector(TAILLE-1 downto
    0));
end MuxGenerique;
```

Le multiplexeur

Exemple de description générique

```
architecture CaseProcess of MuxGenerique is
begin
process(all)
begin
case Selecteur is
  when "00" => Sortie <= in0;
  when "01" => Sortie <= in1;
  when "10" => Sortie <= in2;
  when others => Sortie <= in3;
end case;
end process;
end CaseProcess;
```

Le multiplexeur

Exemple de description générique

Création d'un paquetage pour déclarer une constante :

```
package Config is
  constant TAILLEDATA : integer;
end package Config;
package body Config is
  constant TAILLEDATA: integer := 16;
end package body Config;
```

Il est aussi possible de déclarer une constante dans la partie déclarative du fichier, dans l'entité ou dans un `process` selon la visibilité souhaitée.
Instanciation et paramétrage du multiplexeur générique :

Le multiplexeur

Exemple de description générique

```
library work;
-- appel du package Config
use work.Config.all;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Multiplexeur16bits is port (
     ina, inb, inc, ind : in std_logic_vector(TAILLEDATA-1 downto 0);
     Sel : in std_logic_vector(1 downto 0);
     S : out std_logic_vector(TAILLEDATA-1 downto 0));
end Multiplexeur16bits;

architecture instance of Multiplexeur16bits is
-- partie déclarative de l'archi
component MuxGenerique is
generic ( TAILLE : integer := 8 );
port ( in0, in1, in2, in3 : in  std_logic_vector(TAILLE-1 downto 0);
       Selecteur          : in  std_logic_vector(1 downto 0);
       Sortie             : out std_logic_vector(TAILLE-1 downto 0));
```

Le multiplexeur

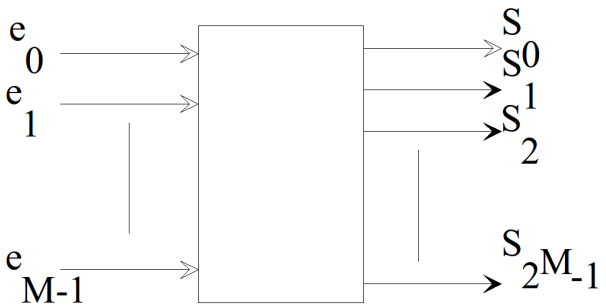
Exemple de description générique

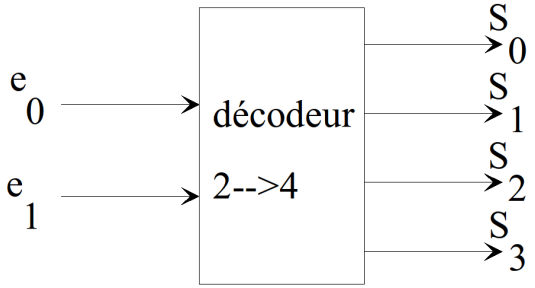
```
end component;

begin
-- instanciation du composant
MonMux: MuxGenerique
  generic map(TAILLE=>TAILLEDATA)
  port map(in0=>ina,in1=>inb,in2=>inc,in3=>ind,
      Selecteur=>Sel,Sortie=>S);
end instance;
```

Le décodeur

Fonction et spécifications

- Optimisation d'un cas particulier du multiplexeur
- Très utilisé pour l'adressage
- Sortie active = sortie dont le numéro correspond à la valeur d'entrée
- *entree* : entier d'entrée sur N bits
- $\{Sortie_0; Sortie_1; ...; Sortie_{2^N-1}\}$: sorties sur 1 bit
- Sortie active = $Sortie_{entree}$ ($\forall$ autres inactives)





$$S_0 = \bar{e}_0 \cdot \bar{e}_1$$
$$S_1 = e_0 \cdot \bar{e}_1$$
$$S_2 = \bar{e}_0 \cdot e_1$$
$$S_3 = e_0 \cdot e_1$$

Le décodeur

Description VHDL

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

entity Decodeur is
port(
     Entree:  in  std_logic_vector(1 downto 0);
     Sortie : out std_logic_vector(3 downto 0));
end Decodeur;

architecture ConcSelect of Decodeur is
begin
with Entree select
Sortie <= "0001" when "00",
     "0010" when "01",
```

Le décodeur
Description VHDL

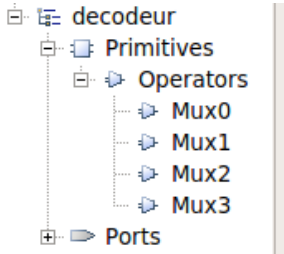
```
    "0100" when "10",
    "1000" when "11",
    "0000" when others;
end ConcSelect;
```

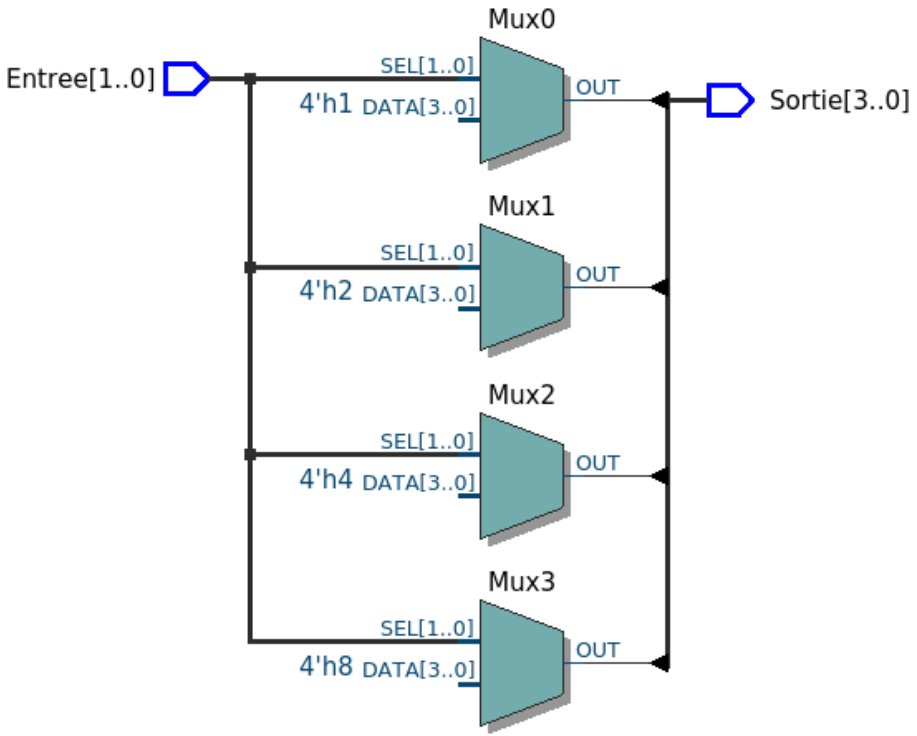
- le cas `Others` n'est pas nécessaire ici pour la synthèse

Le décodeur

Vue du niveau transfert de registres (RTL)

- On constate qu'il y a bien utilisation de multiplexeurs
- On retrouve l'entrée du décodeur comme sélecteur
- L'entrée du multiplexeur devient une constante

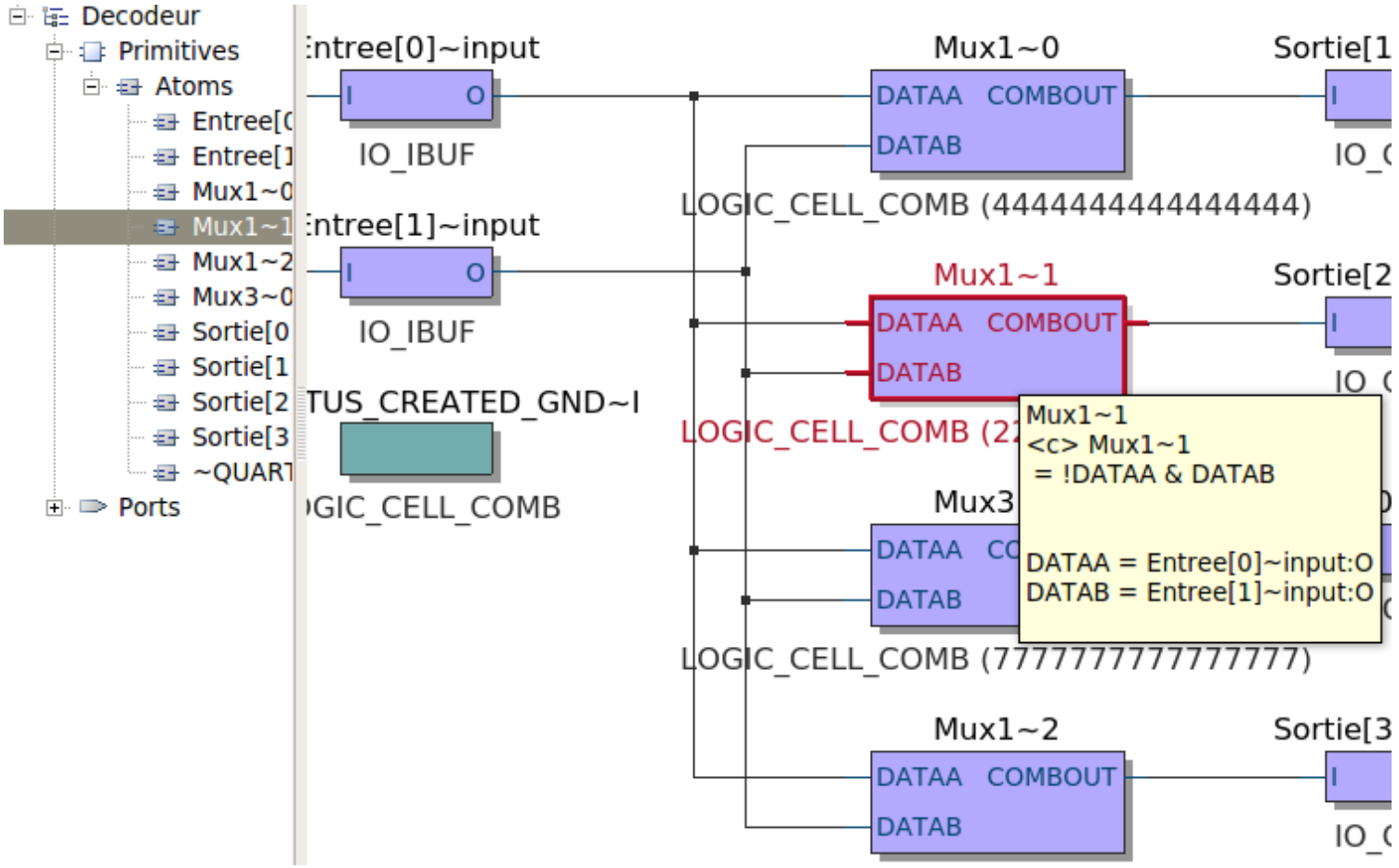




Le décodeur

Vue du niveau cellules FPGA

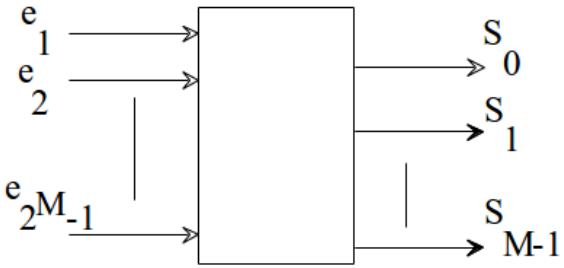
- On voit l'optimisation par rapport à la vue RTL

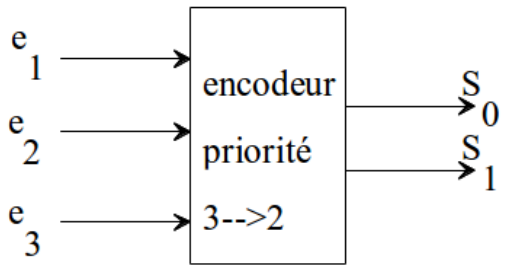


L'encodeur

Fonction et spécifications

- Donne l'indice le plus élevé parmi les entrées actives
- Presque fonction «réciproque» du décodeur
- Priorité d'évènements (interruptions)
- $\{E_1; E_2; ....; E_N\}$: entrées binaires
- Sortie : entier (nb bits $\geq Log_2(N)$)
- Sortie = N si E_N active sinon N-1 si E_{N-1} active sinon... 1 si E_1 active sinon 0





$$S_0 = e_3 + \bar{e}_3 \cdot \bar{e}_2 \cdot e_1$$
$$S_1 = e_3 + e_2$$

L'encodeur

Description VHDL

- Nous utilisons un `process` pour bénéficier des indéterminations dans les choix

```
entity Encodeur is
port(
     Entree : in std_logic_vector(2 downto 0);
     Sortie : out std_logic_vector(1 downto 0));
end Encodeur;

architecture ProCase of Encodeur is begin
process (Entree)
   begin
   case? Entree is
            when "1--"  => Sortie <= "11" ;
```

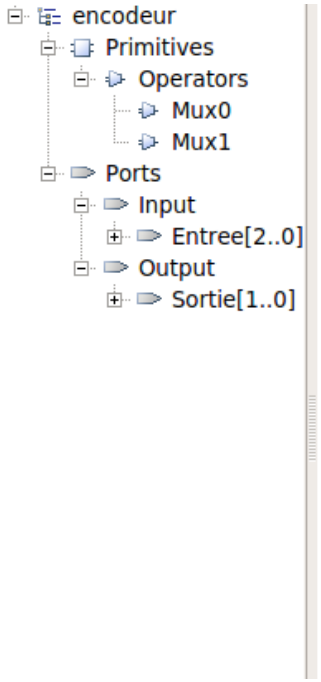
L'encodeur
Description VHDL

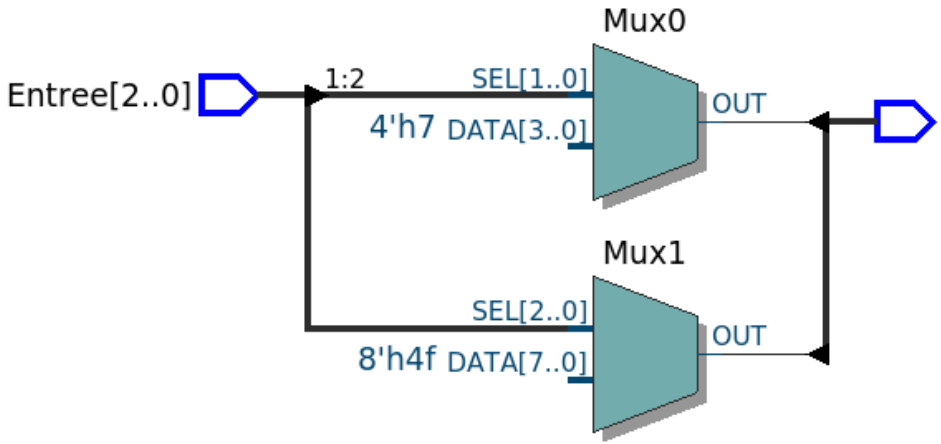
```
            when "01-"  => Sortie <= "10" ;
            when "001"  => Sortie <= "01" ;
            when others => Sortie <= "00" ;
   end case? ;
   end process;
end ProCase;
```

L'encodeur

Vue du niveau transfert de registres (RTL)

- On constate à nouveau l'utilisation de multiplexeurs
- On retrouve l'entrée de l'encodeur comme sélecteur
- L'entrée du multiplexeur devient une constante

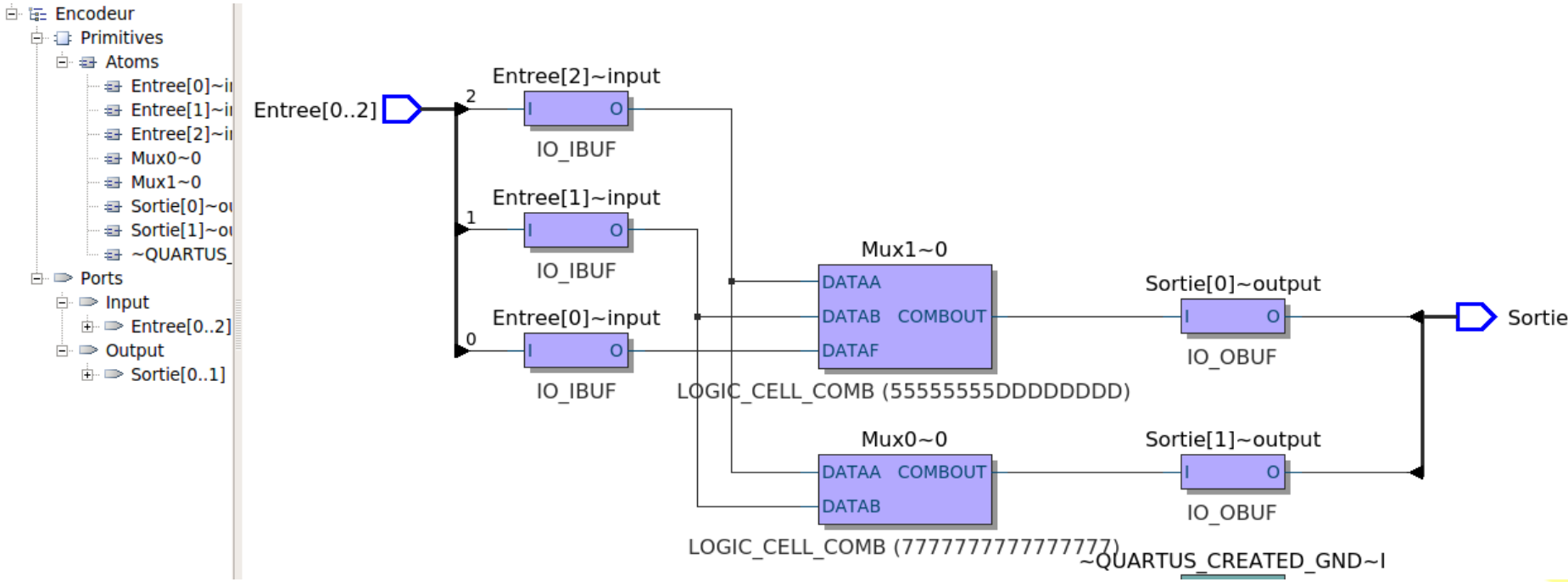




L'encodeur

Vue du niveau cellules FPGA

- On voit à nouveau l'optimisation par rapport à la vue RTL



Exercices

1. Reprendre le multiplexeur et le modifier pour qu'il soit de type $5 \rightarrow 1$. Utiliser un `process`
2. Que se passe-t-il si l'on n'utilise pas le cas `others` ?
3. Reprendre le décodeur et le décrire en utilisant un `process`
4. Reprendre l'encodeur et le décrire en utilisant uniquement des instructions `concurrentes`

Exercices

Solution question 1

```
entity Multiplexeur is
port (
     in0, in1, in2, in3, in4 : in std_logic_vector(7
    downto 0);
     Selecteur               : in std_logic_vector(2
    downto 0);
     Sortie                  : out std_logic_vector(7
    downto 0));
end Multiplexeur;

architecture CaseProcess of Multiplexeur is
begin
process(all)
```

Exercices

Solution question 1

```
begin
    case Selecteur is
      when "000" => Sortie <= in0;
      when "001" => Sortie <= in1;
      when "010" => Sortie <= in2;
      when "011" => Sortie <= in3;
      when "100" => Sortie <= in4;
      when others => Sortie <= (others => '0');
    end case;
end process;
end CaseProcess;
```

Exercices

Solution question 3

```
entity Decodeur is
port(
     Entree : in std_logic_vector(1 downto 0);
     Sortie : out std_logic_vector(3 downto 0));
end Decodeur;
architecture ConcSelect of Decodeur is
begin
process(all)
begin
   case Entree is
      when "00"    => Sortie <= "0001";
      when "01"    => Sortie <= "0010";
      when "10"    => Sortie <= "0100";
      when "11"    => Sortie <= "1000";
```

Exercices

Solution question 3

```
      when others  => Sortie <= "0000";
end process;
end ConcSelect;
```

Exercices

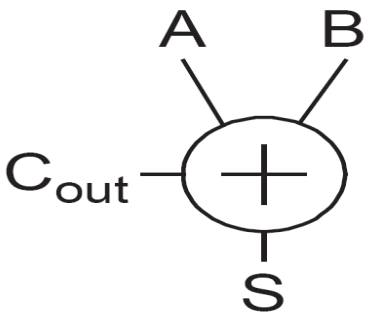
Solution question 4

```
entity Encodeur is
port(
     Entree : in  std_logic_vector(2 downto 0);
     Sortie : out std_logic_vector(1 downto 0));
end Encodeur;

architecture ProConc of Encodeur is begin
   Sortie(0) <= Entree(2) or (not Entree(2) and not
    Entree(1) and Entree(0));
   Sortie(1) <= Entree(2) or Entree(1);
end ProConc;
```

Additionneur

Fonction et spécifications

Half adder



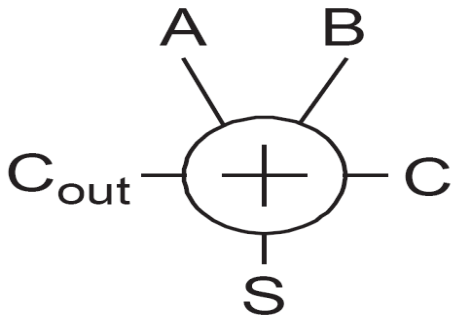
A	B	C_{OUT}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$G = A \cdot B$$

$$P = A \oplus B$$

$$K = \overline{A} \cdot \overline{B}$$

Full adder



A	B	C	G	P	K	C_{OUT}	S
0	0	0	0	0	1	0	0
		1				0	1
0	1	0	0	1	0	0	1
		1				1	0
1	0	0	0	1	0	0	1
		1				1	0
1	1	0	1	0	0	1	0
		1				1	1

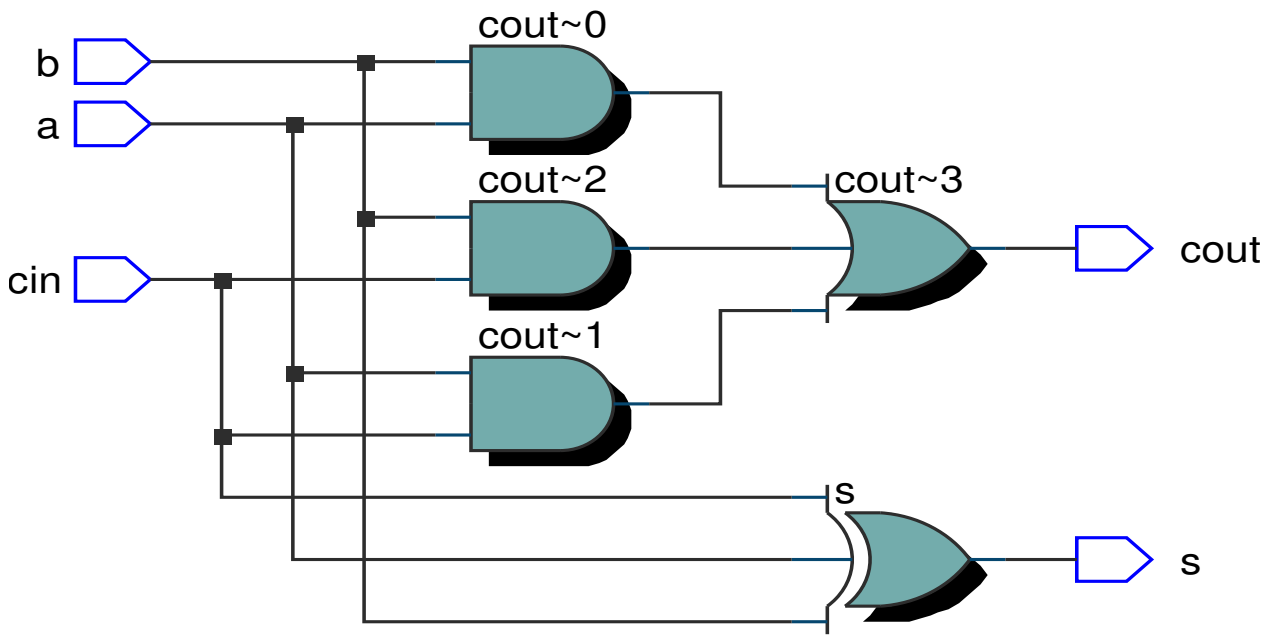
Additionneur

Description VHDL

```
library ieee;
use ieee.std_logic_1164.all;
entity additionneur is
port(
     a,b,cin  : in  std_logic;
     s,cout   : out std_logic);
end additionneur;
--end;
architecture archConc of additionneur is
begin
   s    <= a xor b xor cin;
   cout <= (a and b) or (a and cin) or (b and cin);
end archConc;
```

Additionneur

Vue du niveau transfert de registre (RTL)



Soustracteur

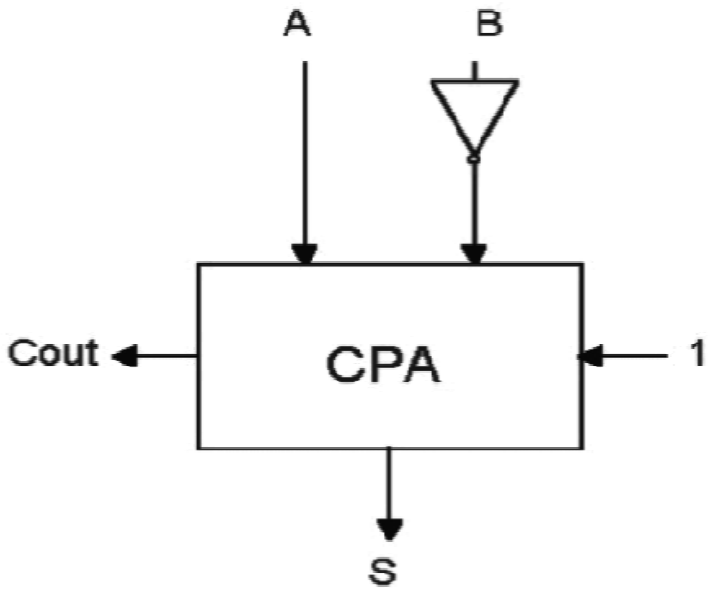
Principe :

- Complémenter à 2 l'opérande à soustraire et réaliser une addition du résultat avec l'autre opérande

Exemple :

Soustracteur

$A - B = A + (-B) = A + \overline{B} + 1$

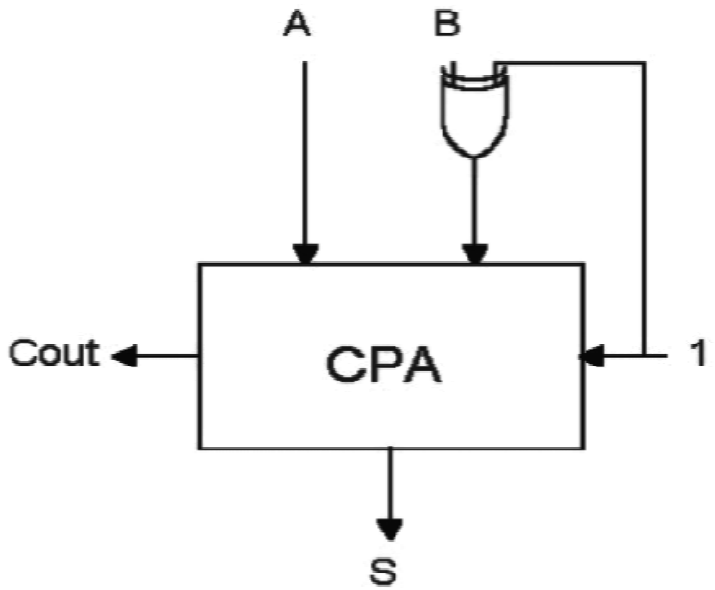


Additionneur/soustracteur

mode addition : $sub = 0$

mode soustraction : $sub = 1$

$A \pm B = A + (B \oplus sub) + sub$



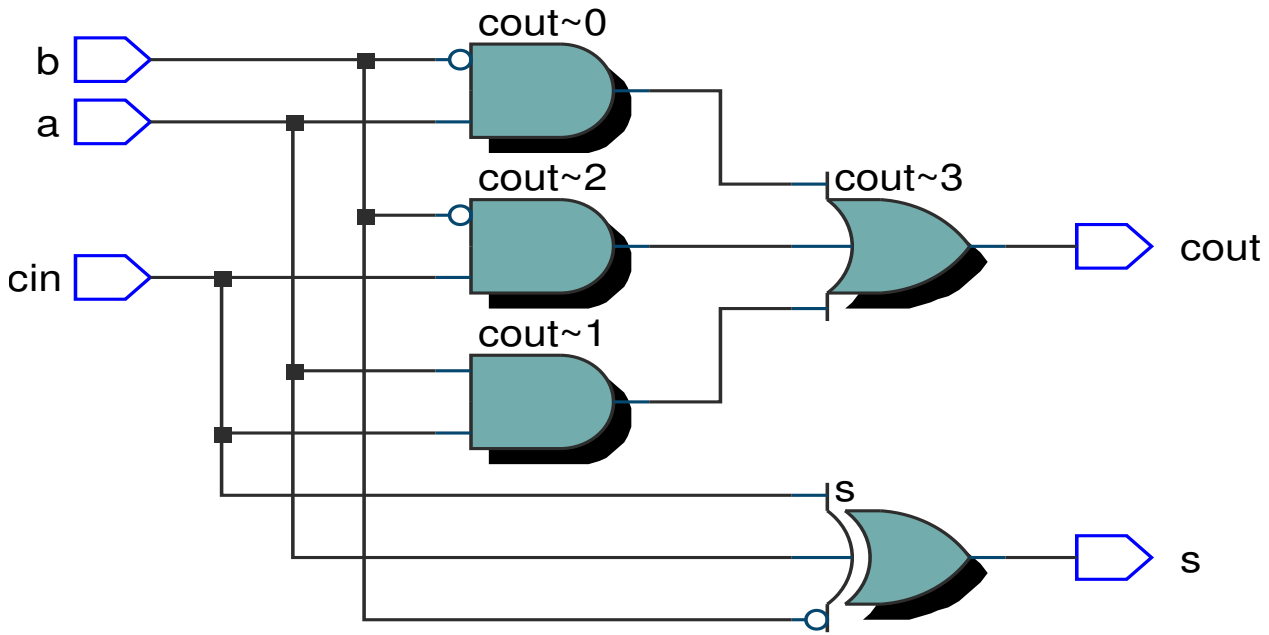
Soustracteur

Description VHDL

```
library ieee;
use ieee.std_logic_1164.all;
entity soustracteur is
port(
     a,b,cin  : in  std_logic;
     s,cout   : out std_logic);
end soustracteur;
--end;
architecture archConc of soustracteur is
   signal bn :std_logic;
begin
   bn   <= not b;
   s    <= a xor bn xor cin;
   cout <= (a and bn) or (a and cin) or (bn and cin);
end archConc;
```

Soustracteur

Vue du niveau de transfert de registres (RTL)



Comparateur

Fonctionalité et spécifications

Principe :

- Calculer si $A = B$ ou si $A \geq B$ permet de déterminer les autres résultats de comparaison

$$EQ = (A = B) \qquad NE = (A \neq B) = \overline{EQ} \qquad GT = (A > B) = \overline{EQ} \cdot GE$$
$$GE = (A \geq B) \qquad LT = (A < B) = \overline{GE} \qquad LE = (A \leq B) = \overline{GE} + EQ$$

- $EQ = (A = B) = (B - A = 0)$
- $EQ_{i+1} = (A_i = B_i) \cdot EQ_i = \overline{(A_i \oplus B_i)} \cdot EQ_i$
- $EQ_0 = 1; EQ_n = EQ$ ou utiliser un soustracteur optimisé (sans sortie S_i)
- $GE = (A \geq B) = (A - B \geq 0)$
- $GE_{i+1} = (A_i > B_i) + (A_i = B_i) \cdot GE_i = A_i \cdot \overline{B_i} + \overline{(A_i \oplus B_i)} \cdot GE_i$
- $GE_0 = 1; GE_n = GE$ ou utiliser un soustracteur