VHDL - Logique programmable

Partie 3
Le style flot de données

Denis Giacona

ENSISA
École Nationale Supérieure d'Ingénieur Sud Alsace
12, rue des frères Lumière
68 093 MULHOUSE CEDEX
FRANCE

Tél. 33 (0)3 89 33 69 00

1.Présentation des instructions concurrentes d'assignation de signal

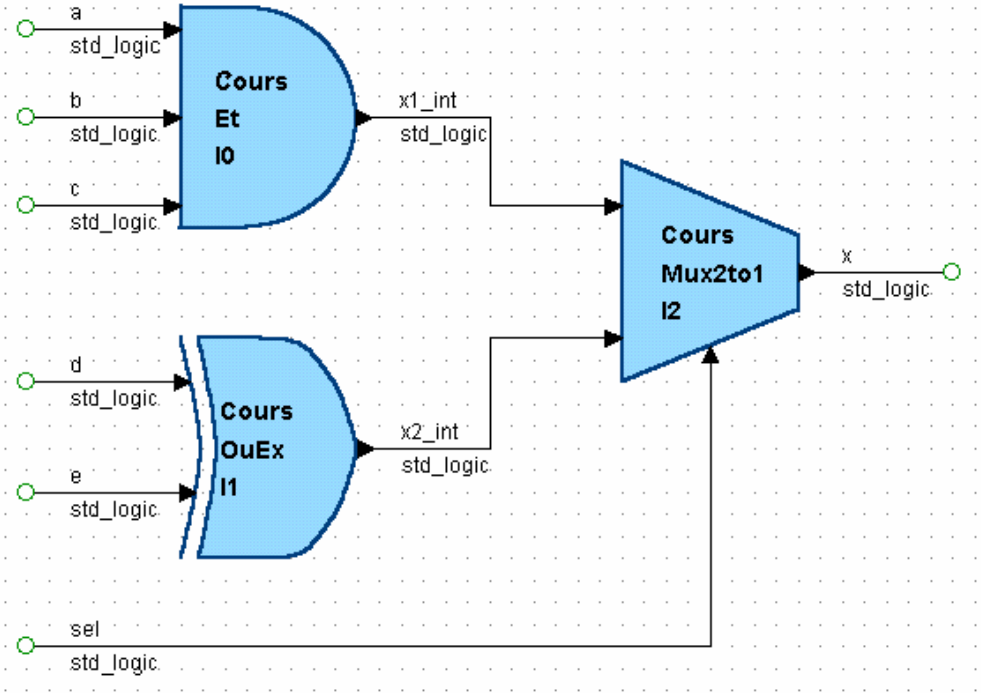
Instruction d'assignation inconditionnelle	`… <= …`
Instruction d'assignation conditionnelle	`… <= … when … else …`
Instruction d'assignation sélective	`with … select … <= … when …`

- **Les instructions sont en concurrence les unes par rapport aux autres ; elles dénotent des signaux qui évoluent de manière asynchrone les uns par rapport aux autres (parallélisme réel)**

- **☝ l'ordre d'écriture des instructions n'a pas d'importance**

Dans le schéma ci-contre :

- **la position des blocs logiques est quelconque**
- **les interconnexions sont déterminantes**
- **les signaux d'entrée et les signaux internes évoluent de façon parallèle et indépendante**



Plusieurs possibilités pour décrire ce système, par exemple :

<u>Solution 1</u>
Trois expressions correspondant respectivement aux trois blocs logiques (il faut alors utiliser les deux signaux internes)

<u>Solution 2</u>
Une seule expression booléenne, en faisant abstraction des signaux internes

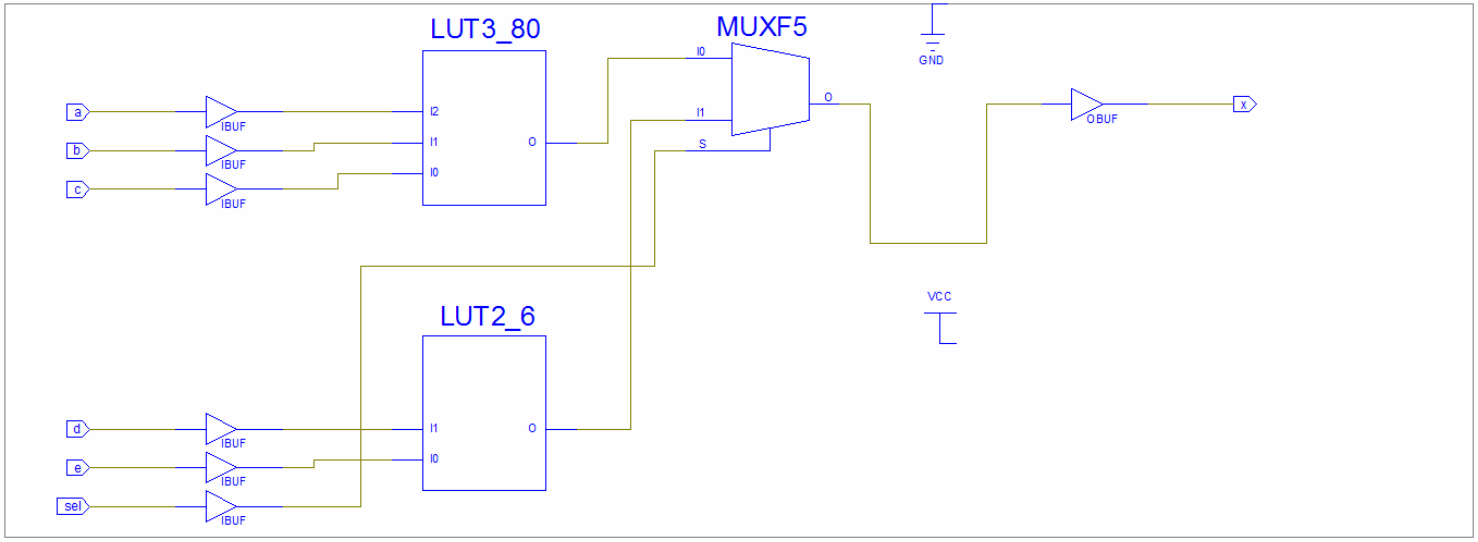
```
-- Solution 1 (l'ordre des trois instructions est quelconque)

architecture arch_Flot_donnees of Flot_donnees is
  signal x1_int, x2_int : std_logic;

begin

  et :          x1_int <= a and b and c;

  ou_ex :       x2_int <= d xor e;

  mux2to1 :     x <=    x1_int when sel = '0' else
                        x2_int;

end arch_Flot_donnees;
```

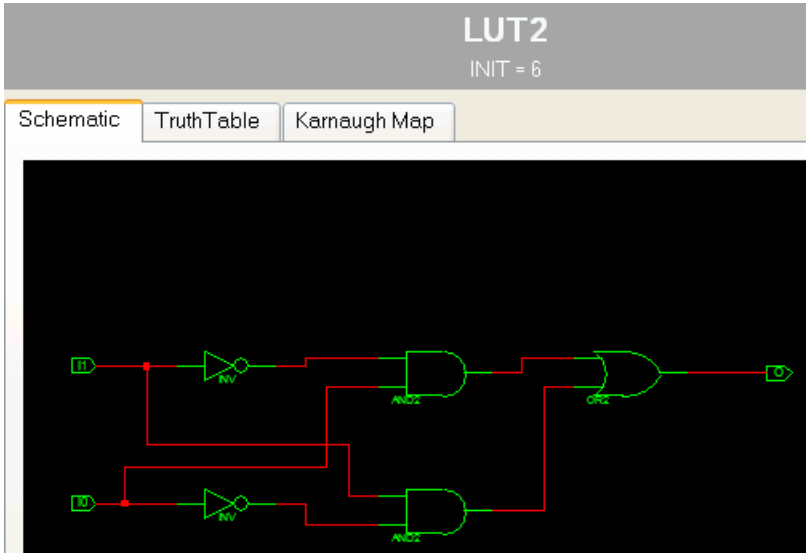
- **EQUATIONS pour un circuit CPLD**

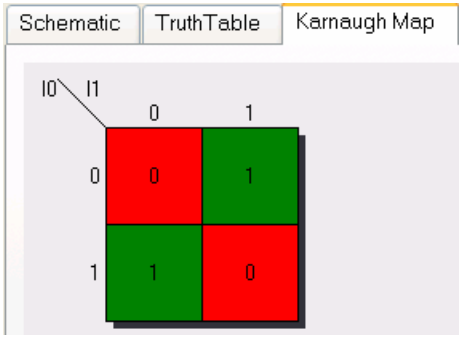
```
x =    a * b  * c  * /sel
     + /d * e  * sel
     +  d * /e * sel
```

Sauf directive de compilation particulière, le compilateur supprime automatiquement les signaux internes pour optimiser l'expression globale.

- **Schéma technologique pour un circuit FPGA**







2.Instruction concurrente d'assignation inconditionnelle de signal <=

Se prononce : « prend la valeur de »
ou bien « reçoit »

```
[étiquette :]

identificateur_signal  <=     littéral
                            | nom_de_signal
                            | agrégat
                            | concaténation_de_bits
                            | expression_logique
                            | expression_arithmétique;
```

identificateur_de_signal ::= nom_de signal | agrégat

2.1. Assignation de littéraux

```
signal stop : std_logic;
signal data1, data2, data3 : std_logic_vector(3 downto 0);
…
begin
 stop     <= '1';          -- bit
 data3(3) <= '0';
 data3(2) <= '1';
 data3(1) <= '0';
 data3(0) <= '1';

 data1    <= "0101";       -- vecteur de bits

 data2    <= x"5";         -- valeur en hexadécimal
```

Les trois vecteurs `data1`, `data2` et `data3` prennent tous la même valeur (un littéral fixé à la compilation).

2.2. Assignation d'un signal vecteur (ou élément de vecteur)

```
entity bloc_logique is port(
 data : in std_logic_vector(7 downto 0);
…
x <= data;              -- vecteur complet
y <= data(0);           -- le bit de rang 0
z <= data(7 downto 4); -- 4 bits particuliers
```

Note

- **Lorsque les indices ne sont pas indiqués, les éléments sont assignés en fonction de leur position dans le vecteur.**

- **Prendre l'habitude de toujours utiliser la même direction pour les vecteurs (traditionnellement `downto` pour les applications destinées à une synthèse)**

2.3. Assignation d'agrégats

Un agrégat est un moyen d'indiquer la valeur de chaque élément d'une donnée d'un type composite.

(élément {,élément})

élément = nom_signal
| expression_logique
| index_sup downto index_inf => '0' | '1'
| index_inf to index_sup => '0' | '1'
| others => '0' | '1'

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

entity agregat is port(
  a, b, c, d : in std_logic;
  t, x, y, z : out std_logic_vector(7 downto 0);
  w, u       : out std_logic_vector(0 to 3));
end agregat;
```

Note

- **Les agrégats peuvent être utilisés de part et d'autre d'un opérateur d'assignation.**

```
architecture arch_agregat of agregat is
  signal temp    : std_logic_vector(3 downto 0);

begin
  temp <= (a, b, a or b, c and d);

  t(7 downto 4) <= temp;
  t(3 downto 0) <= (others => '1');

  (u(3),u(2))   <= ('1','0');
  (u(1),u(0))   <= "11";

  w(0 to 2)     <= o"6";
  w(3)          <= '1';

  x <= ('0','1','1','0',3 downto 0 => '0');

  y <= (7 => '0', 6 downto 5 => '1',others => '0');

  z(3 downto 0) <= temp(3 downto 0);
  z(7 downto 4) <= (d,c,b,a);


end arch_agregat;
```

```
DESIGN EQUATIONS

t(0) = VCC
t(1) = VCC
t(2) = VCC
t(3) = VCC
t(4) = c * d
t(5) = b + a
t(6) = b
t(7) = a

u(0) = VCC
u(1) = VCC
u(2) = GND
u(3) = VCC

w(0) = VCC
w(1) = VCC
w(2) = GND
w(3) = VCC

x(0) = GND
x(1) = GND
x(2) = GND
x(3) = GND
x(4) = GND
x(5) = VCC
x(6) = VCC
x(7) = GND

y(0) = GND
y(1) = GND
y(2) = GND
y(3) = GND
y(4) = GND
y(5) = VCC
y(6) = VCC
y(7) = GND

z(0) = t(4).CMB
z(1) = t(5).CMB
z(2) = b
z(3) = a
z(4) = a
z(5) = b
z(6) = c
z(7) = d
```

2.4. Assignation d'une concaténation de bits (opérateur &)

```
entity concatenation is port(
  data : in std_logic_vector (7 downto 0);
  x    : out std_logic);
end concatenation;

architecture arch_concatenation of concatenation is
  constant start : std_logic := '0';
  constant stop  : std_logic := '1';
  signal temp    : std_logic_vector(9 downto 0);

begin
  temp <= start & data & stop ;
```

Note

- L'opérateur de concaténation & n'est permis que sur la partie droite de l'opérateur d'assignation <=

2.5. Assignation d'une expression logique

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

entity mux4to1_4bits_v3 is port(
     a,b,c,d : in std_logic_vector(3 downto 0);
     s :       in std_logic_vector(1 downto 0);
     x :       out std_logic_vector(3 downto 0));
end mux4to1_4bits_v3;

architecture archmux4to1_4bits_v3 of mux4to1_4bits_v3 is
begin
  x(3) <= (not s(1) and not s(0) and a(3))
     or   (not s(1) and     s(0) and b(3))
     or   (    s(1) and not s(0) and c(3))
     or   (    s(1) and     s(0) and d(3));
  x(2) <= (not s(1) and not s(0) and a(2))
     or   (not s(1) and     s(0) and b(2))
     or   (    s(1) and not s(0) and c(2))
     or   (    s(1) and     s(0) and d(2));
  x(1) <= (not s(1) and not s(0) and a(1))
     or   (not s(1) and     s(0) and b(1))
     or   (    s(1) and not s(0) and c(1))
     or   (    s(1) and     s(0) and d(1));
  x(0) <= (not s(1) and not s(0) and a(0))
     or   (not s(1) and     s(0) and b(0))
     or   (    s(1) and not s(0) and c(0))
     or   (    s(1) and     s(0) and d(0));
end archmux4to1_4bits_v3;
```

```
DESIGN EQUATIONS (for CPLDs)

    x(0) =
          d(0) * s(0) * s(1)
        + c(0) * /s(0) * s(1)
        + b(0) * s(0) * /s(1)
        + a(0) * /s(0) * /s(1)

    x(1) =
          d(1) * s(0) * s(1)
        + c(1) * /s(0) * s(1)
        + b(1) * s(0) * /s(1)
        + a(1) * /s(0) * /s(1)

    x(2) =
          d(2) * s(0) * s(1)
        + c(2) * /s(0) * s(1)
        + b(2) * s(0) * /s(1)
        + a(2) * /s(0) * /s(1)

    x(3) =
          d(3) * s(0) * s(1)
        + c(3) * /s(0) * s(1)
        + b(3) * s(0) * /s(1)
        + a(3) * /s(0) * /s(1)
```

2.6. Assignation d'une expression comportant des vecteurs

```
entity assignation_vecteurs is port(
    a, b     :   in std_logic_vector(7 downto 0);
    x,y,z    :   out std_logic_vector(7 downto 0));
end assignation_vecteurs;


architecture arch_assignation_vecteurs of assignation_vecteurs is

begin

x <= a and b;
y <= a or b;
z <= a xor b;

end arch_assignation_vecteurs;
```

```
z(0) =
        /a(0) * b(0)
      + a(0) * /b(0)

   z(1) =
        /a(1) * b(1)
      + a(1) * /b(1)

   z(2) =
        /a(2) * b(2)
      + a(2) * /b(2)

   z(3) =
        /a(3) * b(3)
      + a(3) * /b(3)

   z(4) =
        /a(4) * b(4)
      + a(4) * /b(4)

   z(5) =
        /a(5) * b(5)
      + a(5) * /b(5)

   z(6) =
        /a(6) * b(6)
      + a(6) * /b(6)

   z(7) =
        /a(7) * b(7)
      + a(7) * /b(7)
```

2.7. Assignation d'une expression arithmétique

Les opérateurs + et – sont utilisés fréquemment pour décrire des incrémentations ou décrémentations de compteurs, des additions et des soustractions.

Ils sont prédéfinis pour les types numériques.

On peut néanmoins les appliquer à des `std_logic_vector` à condition d'utiliser la surcharge d'opérateur (redéfinition d'un opérateur pour un nouveau type) définie dans le paquetage `std_logic_unsigned` de la bibliothèque `ieee` (outil Xilinx ISE).

```
Sum     <= a + b;

Count   <= count + 1;
```

2.8. Signaux multisources dans le cas d'une synthèse

Lorsqu'on assigne deux valeurs différentes à un même signal, cela revient à connecter plusieurs sources ensemble → le synthétiseur génère une erreur.

```
y       <= a;
y       <= b;
```

➔ **Can't handle multiple drivers for 'y' in selected device**

3.Instruction concurrente d'assignation conditionnelle de signal … <= … when … else …

C'est l'assignation d'une valeur (de valeurs) à un signal (ou à un agrégat de signaux) sur la base d'une condition.

```
[étiquette :]
identificateur_signal
    <=    {valeur_signal when condition else}
           valeur_signal;

        | {valeur_signal when condition else}
           valeur_signal when condition;
```

Identificateur_signal = nom_signal | agrégat
Valeur_signal = littéral | identificateur_signal | expression_logique
Condition = expression booléenne comportant
- des opérateurs relationnels: =, /=, <, <=, >, >=
- des opérateurs logiques: and, or, xor

Nota bene

- L'ordre de déclaration des conditions permet de définir des priorités pour les signaux en jeu ; à tout moment, le signal prend la valeur correspondant à la première condition évaluée à `true`
- Les conditions listées ne sont pas nécessairement mutuellement exclusives
- Tous les signaux en jeu ne doivent pas apparaître obligatoirement dans l'expression d'une condition
- Une liste explicite de tous les cas n'est pas obligatoire
  - Si la liste n'est pas exhaustive alors :
    - ➔ si le mot clé `else` suit le dernier `when`, celui-ci se rapporte aux conditions manquantes et détermine la valeur à assigner pour l'ensemble des cas manquants

```
-- Soient a, b, c, d les signaux en jeu
x <= '1' when d = '1' else
     '1' when a = b and c = '0' else
     '0';
```

x = a * b * /c
+ /a * /b * /c
+ d

    - ➔ sinon, le signal conserve sa valeur pour toutes les conditions manquantes et il y a mémorisation implicite (en logique séquentielle asynchrone)

```
x <= '1' when d = '1' else
     '1' when a = b and c = '0';
```

x = a * b * /c
+ /a * /b * /c
+ x.CMB
+ d

3.1. Condition = expression booléenne

```
signal a,b,c,d : std_logic_vector(3 downto 0);

begin

x <= '0' when (a > b) and c = d;

y <= '1' when etat = attente and data = x"1FAE";
```

3.2. Le signal est un vecteur, liste de conditions avec mot-clé else

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

entity mux4to1_4bits_v4 is port(
  a,b,c,d :  in std_logic_vector(3 downto 0);
  s :        in std_logic_vector(1 downto 0);
  x :        out std_logic_vector(3 downto 0));
end mux4to1_4bits_v4;

architecture archmux4to1_4bits_v4 of mux4to1_4bits_v4 is
begin

  x <= a when (s = "00") else
       b when (s = "01") else -- liste explicite non exhaustive
       c when (s = "10") else    -- (3 cas sur 81 !!)
       d;
end archmux4to1_4bits_v4;
```

```
DESIGN EQUATIONS

    x(0) = d(0) *  s(0) * s(1)
         + c(0) * /s(0) * s(1)
         + b(0) *  s(0) * /s(1)
         + a(0) * /s(0) * /s(1)

    x(1) = d(1) *  s(0) * s(1)
         + c(1) * /s(0) * s(1)
         + b(1) *  s(0) * /s(1)
         + a(1) * /s(0) * /s(1)

    x(2) = d(2) * s(0)  * s(1)
         + c(2) * /s(0) * s(1)
         + b(2) * s(0)  * /s(1)
         + a(2) * /s(0) * /s(1)

    x(3) = d(3) * s(0)  * s(1)
         + c(3) * /s(0) * s(1)
         + b(3) * s(0)  * /s(1)
         + a(3) * /s(0) * /s(1)
```

3.3. Conditions non mutuellement exclusives avec mot-clé else

L'ordre des conditions détermine la priorité

```
entity comb1 is port(
  a, b, c, d     : in std_logic;
  s0, s1, s2, s3 : in std_logic;
  x              : out std_logic);
end comb1;

architecture archcomb1 of comb1 is
begin

  x <= a when s0 = '1' else  -- priorité accordée à s0
       b when s1 = '1' else  -- si s0='0', priorité à s1
       c when s2 = '1' else  -- si s0='0' et s1='0', priorité à s2
       d when s3 = '1' else
       a and b and c and d;  -- valeur signal = expression logique

end archcomb1;
```

```
DESIGN EQUATIONS

    x =
          a * s0
        + b * /s0 * s1
        + c * /s0 * /s1 * s2
        + d * /s0 * /s1 * /s2 * s3
        + a * b * c * d
```

3.4. Liste de conditions non exhaustive avec dernier mot-clé else

```
entity when_else_incomplet is port(
  a, b, c, d : in std_logic;
  s          : in std_logic_vector(1 downto 0);
  x          : out std_logic);
end when_else_incomplet;

architecture archwhen_else_incomplet of when_else_incomplet
is
begin
  x <= a when (s = "01") else
       b when (s = "10") else
       c;
end archwhen_else_incomplet;
```

```
DESIGN EQUATIONS

    x =
          c * s(0) * s(1)
        + b * /s(0) * s(1)
        + a * s(0) * /s(1)
        + c * /s(0) * /s(1)
```

3.5. Liste de conditions non exhaustive, dernière clause else manquante

❒ Mémorisation implicite

```
entity when_else_manquant is port(
    a, b:    in std_logic;
    s:       in std_logic_vector(1 downto 0);
    x:       out std_logic);
end when_else_manquant;

architecture archwhen_else_manquant of when_else_manquant is
begin
    x <= a when (s = "01") else
         b when (s = "10");

end archwhen_else_manquant;
```

Logique séquentielle asynchrone
x : valeur actuelle
x.CMB : valeur précédente
→ **traduit une rétroaction**

```
DESIGN EQUATIONS

    x =
          s(0) * s(1) * x.CMB      -- mémorisation implicite
        + b * /s(0) * s(1)
        + a * s(0) * /s(1)
        + /s(0) * /s(1) * x.CMB    -- mémorisation implicite
```

3.6. Identificateur_signal = agrégat

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

entity two_mux4to1_v2 is port(
   a,b,c,d : in std_logic;
   s : in std_logic_vector(1 downto 0);
   x,y : out std_logic);
end two_mux4to1_v2;

architecture archtwo_mux4to1_v2 of two_mux4to1_v2 is
begin
  (x,y) <= (a,d) when (s = "00") else
           (b,c) when (s = "01") else
           (c,b) when (s = "10") else
           (d,a);
end archtwo_mux4to1_v2;
```

```
DESIGN EQUATIONS
    x =
          d * s(0) * s(1)
        + c * /s(0) * s(1)
        + b * s(0) * /s(1)
        + a * /s(0) * /s(1)

    y =
          a * s(0) * s(1)
        + b * /s(0) * s(1)
        + c * s(0) * /s(1)
        + d * /s(0) * /s(1)
```

4.Instruction concurrente d'assignation sélective de signal with … select … <= … when …

Cette instruction permet d'assigner une valeur (des valeurs) à un signal (un agrégat de signaux) en fonction des valeurs d'un sélecteur.

```
[étiquette :]
with sélecteur select
  identificateur_signal
        <= { valeur_signal when valeur_sélecteur,}
             valeur_signal when valeur_sélecteur;
```

Identificateur_signal	= nom_signal | agrégat
Sélecteur	= identificateur_signal
Valeur_signal	= littéral | identificateur_signal | expression_logique
Valeur_sélecteur	= littéral | choix_multiple | concaténation | agrégat | `others`
Choix_multiple	= valeur { | valeur}

Note

- Toutes les valeurs du sélecteur doivent être mutuellement exclusives
- Si la liste des valeurs de sélecteur n'est pas exhaustive, le mot réservé `others` doit être utilisé pour rassembler toutes les valeurs de sélecteur non citées explicitement
- L'opérateur | sert à composer un choix multiple

4.1. Liste de valeurs de sélecteur non exhaustive, clause others présente

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

entity mux4to1_v2 is port(
  a,b,c,d : in std_logic;
  s : in std_logic_vector(1 downto 0);
  x : out std_logic);
end mux4to1_v2;

architecture archmux4to1_v2 of mux4to1_v2 is
begin
  with s select
    x <= a when "00",
         b when "01",
         c when "10",
         d when others;
end archmux4to1_v2;
```

```
DESIGN EQUATIONS
    x =
          d * s(0) * s(1)
        + c * /s(0) * s(1)
        + b * s(0) * /s(1)
        + a * /s(0) * /s(1)
```

4.2. Liste de valeurs de sélecteur non exhaustive, clause others absente

```
with s select
  x <= a when "00",
       b when "01",
       c when "10",
       d when "11";  -- la clause others manque
```

Rappel : le type std_logic comporte 9 valeurs

```
Compilation failed :(E469) CASE statement
is missing 77 choices
```

4.3. Utilisation de la valeur métalogique '-'

```
with s select
  x <= a when "00",
       b when "01",
       c when "10",
       '-' when others;
```

Le synthétiseur peut choisir une valeur de signal qui simplifie l'expression logique résultante (dans ce cas simple, ce sera '0')

4.4. Valeur_signal = identificateur_signal | expression_logique

valeur_sélecteur = choix_multiple | agrégat | concaténation

```
entity comb2 is port(
  a, b, c, d : in std_logic;
  s          : in std_logic_vector(2 downto 0);
  x          : out std_logic);
end comb2;

architecture archcomb2 of comb2 is
begin
  with s select
    x <= a when ('0','0','0'),          -- agrégat
         b when '0' & '0' & '1',        -- concaténation
         c when "010",                  -- littéral
         d when "011",
         a and b when "100" | "101",    -- choix multiple
         c and d when "110" | "111",    --  sur expression logique
        '-' when others;
end archcomb2;
```

```
DESIGN EQUATIONS

    x =
          d * s(0) * s(1) * /s(2)
        + c * /s(0) * s(1) * /s(2)
        + b * s(0) * /s(1) * /s(2)
        + a * /s(0) * /s(1) * /s(2)
        + c * d * s(1)
        + a * b * /s(1)
```

4.5. Valeur_sélecteur = littéral en octal

```
entity comb3 is port(
   a, b, c, d, e, f   : in std_logic;
   s0, s1, s2         : in std_logic;
   x                  : out std_logic);
end comb3;

architecture archcomb3 of comb3 is

signal temp : std_logic_vector(2 downto 0);

begin

  temp <= (s2,s1,s0);      -- agrégat

    with temp select
        x <=a when o"0", -- littéral en octal
            b when o"1",
            c when o"2",
            d when o"3",
            e when o"4",
            f when o"5",
            '0' when others;
end archcomb3;
```

```
DESIGN EQUATIONS

    x =
          f * s0 * /s1 * s2
        + e * /s0 * /s1 * s2
        + d * s0 * s1 * /s2
        + c * /s0 * s1 * /s2
        + b * s0 * /s1 * /s2
        + a * /s0 * /s1 * /s2
```

4.6. Identificateur_signal = agrégat

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

entity two_mux4to1_v1 is port(
  a, b, c, d:  in std_logic;
  s:           in std_logic_vector(1 downto 0);
  x,y:         out std_logic);
end two_mux4to1_v1;

architecture archtwo_mux4to1_v1 of two_mux4to1_v1 is
begin
  with s select
    (x,y) <= (a,d) when "00",
             (b,c) when "01",
             (c,b) when "10",
             (d,a) when others;

end archtwo_mux4to1_v1;
```

```
DESIGN EQUATIONS
    x =
          d * s(0) * s(1)
        + c * /s(0) * s(1)
        + b * s(0) * /s(1)
        + a * /s(0) * /s(1)

    y =
          a * s(0) * s(1)
        + b * /s(0) * s(1)
        + c * s(0) * /s(1)
        + d * /s(0) * /s(1)
```