UQAC
Université du Québec
à Chicoutimi

Rapport de laboratoire No 1
Introduction au VHDL
Hiver 2023

Conception de systèmes digitaux
6GEI367

Département des Sciences Appliquées
Module d'ingénierie

Travail d'équipe

Maxime Simard
SIMM26050001

Samuel Gaudreault
GAUS09109500

Date de remise : 3 02 2023

Table des matières

Liste des figures

Liste des tableaux

Introduction

Dans ce laboratoire, nous apprendrons à concevoir un pilote de périphérique simple. Tout d'abord, nous utiliserons les interrupteurs et témoins lumineux, puis écrirons en VHDL un détecteur du bit le plus significatif. Ensuite, nous combinerons ce code avec un décodeur 7-segments et écrirons deux multiplexeurs. Finalement, après avoir écrit quatre lettres avec le décodeur 7-segments, nous combinerons cette partie avec le multiplexeur de la partie précédente pour faire un affichage dépendant des entrées (switch).

Question 1 : Utilisation des interrupteurs et témoins lumineux

Explication du programme

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3
4  PACKAGE lab1_1 IS
5     COMPONENT partie1 IS
6        PORT (
7           SW : IN STD_LOGIC_VECTOR(9 DOWNTO 0);
8           LEDR : OUT STD_LOGIC_VECTOR(9 DOWNTO 0)
9        );
10    END COMPONENT;
11 END PACKAGE;
12 -------------------------------------------------------
13 LIBRARY ieee;
14 USE ieee.std_logic_1164.ALL;
15
16 ENTITY partie1 IS
17    PORT (
18       SW : IN STD_LOGIC_VECTOR(9 DOWNTO 0);
19       LEDR : OUT STD_LOGIC_VECTOR(9 DOWNTO 0)
20    );
21 END partie1;
22
23 ARCHITECTURE Behavior OF partie1 IS
24 BEGIN
25    LEDR <= SW;
26 END Behavior;
```

Figure 1 - Code VHDL #1

Ce code VHDL assigne la valeur de l'entrée SW (10 bits) à la sortie LEDR (10 bits). À l'aide du fichier « DE_10.qsf », on peut comprendre que l'entrée SW représente les interrupteurs du FPGA et que la sortie LEDR représente les témoins lumineux. Cela a donc pour effet d'allumer la lumière correspondante à l'interrupteur lorsqu'on active ce-dernier.

Testbench

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.numeric_std.ALL;
USE work.lab1_1.ALL;

ENTITY partie1_tb IS
END partie1_tb;

ARCHITECTURE test OF partie1_tb IS
    SIGNAL SW : STD_LOGIC_VECTOR(9 DOWNTO 0);
    SIGNAL LEDR : STD_LOGIC_VECTOR(9 DOWNTO 0);
BEGIN
    DUT : partie1 PORT MAP(SW, LEDR);
    PROCESS BEGIN
        REPORT "Testbench starting...";
        FOR i IN 0 TO 1023 LOOP
            SW <= STD_LOGIC_VECTOR(to_unsigned(i, SW'length));
            WAIT FOR 10 ns;
            IF SW /= LEDR THEN
                REPORT "SW = " & to_string(SW) & "; LEDR = " & to_string(LEDR);
            END IF;
        END LOOP;
        WAIT;
    END PROCESS;
END ARCHITECTURE test;
```

Figure 2 - Code VHDL du testbench #1

Ce testbench vérifie pour toutes les valeurs possibles si l'entrée n'égale pas la sortie. Si c'est le cas, on écrit les valeurs qui sont problématiques, sinon on n'écrit rien.

```
# ** Note: Testbench starting...
#    Time: 0 ps  Iteration: 0  Instance: /partie1_tb
```

Figure 3 - Résultat du testbench #1

Comme on peut le voir dans la figure ci-dessus, puisqu'il n'y a aucune sortie, les valeurs à la sortie sont les valeurs souhaitées.

Question 2 : Détecteur du « 1 » le plus significatif

Explication du programme

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3
4  PACKAGE lab1_2 IS
5     COMPONENT partie2 IS
6        PORT (
7           SW : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
8           LEDR : OUT STD_LOGIC_VECTOR(2 DOWNTO 0)
9        );
10    END COMPONENT;
11 END PACKAGE;
12 ------------------------------------------------------
13 LIBRARY ieee;
14 USE ieee.std_logic_1164.ALL;
15
16 ENTITY partie2 IS
17    PORT (
18       SW : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
19       LEDR : OUT STD_LOGIC_VECTOR(2 DOWNTO 0)
20    );
21 END partie2;
22
23 ARCHITECTURE arch OF partie2 IS
24 BEGIN
25    PROCESS (ALL) BEGIN
26       CASE ? sw IS
27          WHEN "1-------" => LEDR <= 3d"7";
28          WHEN "01------" => LEDR <= 3d"6";
29          WHEN "001-----" => LEDR <= 3d"5";
30          WHEN "0001----" => LEDR <= 3d"4";
31          WHEN "00001---" => LEDR <= 3d"3";
32          WHEN "000001--" => LEDR <= 3d"2";
33          WHEN "0000001-" => LEDR <= 3d"1";
34          WHEN "0000000-" => LEDR <= 3d"0";
35          WHEN OTHERS => LEDR <= 3d"0";
36       END CASE?;
37    END PROCESS;
38 END arch;
```

Figure 4 - Code VHDL #2

On utilise encore les interrupteurs du FPGA en entrée pour représenter 8 bits, et les témoins lumineux pour représenter un chiffre de 3 bits (0 à 7). Le chiffre affiché à l'aide des témoins lumineux est la position du bit le plus significatif de l'entrée (interrupteurs). Pour ce faire, on utilise les « don't care », vérifiant donc si le bit le plus à gauche (plus significatif) est à 1. Si oui, la sortie sera la position 7. Si non, on vérifie le bit de la prochaine position, ainsi de suite.

Testbench

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.numeric_std.ALL;
USE work.lab1_2.ALL;

ENTITY partie2_tb IS
END partie2_tb;

ARCHITECTURE test OF partie2_tb IS
    SIGNAL SW : STD_LOGIC_VECTOR(7 DOWNTO 0);
    SIGNAL position : STD_LOGIC_VECTOR(2 DOWNTO 0);
BEGIN
    DUT : partie2 PORT MAP(SW, position);
    PROCESS BEGIN
        REPORT "Testbench starting...";

        SW <= "00000001";
        WAIT FOR 10 ns;
        REPORT "SW = " & to_string(SW) & "; position = " & to_string(position);

        SW <= "00000010";
        WAIT FOR 10 ns;
        REPORT "SW = " & to_string(SW) & "; position = " & to_string(position);

        SW <= "00000100";
        WAIT FOR 10 ns;
        REPORT "SW = " & to_string(SW) & "; position = " & to_string(position);

        SW <= "00001000";
        WAIT FOR 10 ns;
        REPORT "SW = " & to_string(SW) & "; position = " & to_string(position);

        SW <= "00010000";
        WAIT FOR 10 ns;
        REPORT "SW = " & to_string(SW) & "; position = " & to_string(position);

        SW <= "00100000";
        WAIT FOR 10 ns;
        REPORT "SW = " & to_string(SW) & "; position = " & to_string(position);

        SW <= "01000000";
        WAIT FOR 10 ns;
        REPORT "SW = " & to_string(SW) & "; position = " & to_string(position);

        SW <= "10000000";
        WAIT FOR 10 ns;
        REPORT "SW = " & to_string(SW) & "; position = " & to_string(position);

        WAIT;
    END PROCESS;
END ARCHITECTURE test;
```

Figure 5 - Code VHDL du testbench #2

Ce testbench vérifie si l'entité « partie2 » est bel est bien capable de détecter la position du bit le plus significatif en essayant huit cas possibles.

```
# ** Note: Testbench starting...
#    Time: 0 ps  Iteration: 0  Instance: /partie2_tb
# ** Note: SW = 00000001; position = 000
#    Time: 10 ns  Iteration: 0  Instance: /partie2_tb
# ** Note: SW = 00000010; position = 001
#    Time: 20 ns  Iteration: 0  Instance: /partie2_tb
# ** Note: SW = 00000100; position = 010
#    Time: 30 ns  Iteration: 0  Instance: /partie2_tb
# ** Note: SW = 00001000; position = 011
#    Time: 40 ns  Iteration: 0  Instance: /partie2_tb
# ** Note: SW = 00010000; position = 100
#    Time: 50 ns  Iteration: 0  Instance: /partie2_tb
# ** Note: SW = 00100000; position = 101
#    Time: 60 ns  Iteration: 0  Instance: /partie2_tb
# ** Note: SW = 01000000; position = 110
#    Time: 70 ns  Iteration: 0  Instance: /partie2_tb
# ** Note: SW = 10000000; position = 111
#    Time: 80 ns  Iteration: 0  Instance: /partie2_tb
```

Figure 6 - Résultat du testbench #2

Comme on peut le voir dans la figure ci-dessus, les valeurs correspondent bien à la position du bit le plus significatif.

Question 3a : Décodeur 7-segments

Explication du programme

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3
4  PACKAGE lab1_3a IS
5     COMPONENT partie3a IS
6        PORT (
7           SW : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
8           HEX0 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
9        );
10    END COMPONENT;
11
12    COMPONENT bit_significatif IS
13       PORT (
14          SW : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
15          LEDR : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
16       );
17    END COMPONENT;
18 END PACKAGE;
19 ----------------------------------------------------------
20 LIBRARY ieee;
21 USE ieee.std_logic_1164.ALL;
22 USE work.sseg_constants.ALL;
23 USE work.lab1_3a.ALL;
24
25 ENTITY partie3a IS
26    PORT (
27       SW : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
28       HEX0 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
29    );
30 END partie3a;
31
32 ARCHITECTURE arch OF partie3a IS
33    SIGNAL a : STD_LOGIC_VECTOR(3 DOWNTO 0);
34 BEGIN
35    bit_significatif1 : bit_significatif PORT MAP(SW, a);
36    sseg0 : sseg PORT MAP(a, HEX0);
37 END arch;
38 ----------------------------------------------------------
39 LIBRARY ieee;
40 USE ieee.std_logic_1164.ALL;
41
42 ENTITY bit_significatif IS
43    PORT (
44       SW : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
45       LEDR : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
46    );
47 END bit_significatif;
48
49 ARCHITECTURE arch OF bit_significatif IS
50 BEGIN
51    PROCESS (ALL) BEGIN
52       CASE ? sw IS
53          WHEN "1-------" => LEDR <= 4d"7";
54          WHEN "01------" => LEDR <= 4d"6";
55          WHEN "001-----" => LEDR <= 4d"5";
56          WHEN "0001----" => LEDR <= 4d"4";
57          WHEN "00001---" => LEDR <= 4d"3";
58          WHEN "000001--" => LEDR <= 4d"2";
59          WHEN "0000001-" => LEDR <= 4d"1";
60          WHEN "0000000-" => LEDR <= 4d"0";
61          WHEN OTHERS => LEDR <= 4d"0";
62       END CASE?;
63    END PROCESS;
64 END arch;
```

Figure 7 - Code VHDL #3 a)

Ce code prend en entrée 8 interrupteurs et a en sortie HEX0 d'une longueur de 7 bits. Cette sortie représente l'afficheur 7-segments le plus à droite du FPGA. De plus, nous avons réécris puis adapté le code de la partie 2 afin de pouvoir facilement détecter le bit le plus significatif en s'adaptant aux demandes de la partie 3 a).

On instancie donc le component de bit significatif et le component « sseg » (provenant du livre), permettant d'afficher sur un afficheur 7-segments le bit le plus significatif des 8 interrupteurs.

Testbench

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3  USE ieee.std_logic_unsigned.ALL;
4  USE ieee.numeric_std.ALL;
5  USE work.lab1_3a.ALL;
6
7  ENTITY partie3a_tb IS
8  END partie3a_tb;
9
10 ARCHITECTURE test OF partie3a_tb IS
11    SIGNAL SW : STD_LOGIC_VECTOR(7 DOWNTO 0);
12    SIGNAL HEX0 : STD_LOGIC_VECTOR(6 DOWNTO 0);
13 BEGIN
14    DUT : partie3a PORT MAP(SW, HEX0);
15    PROCESS BEGIN
16       REPORT "Testbench starting...";
17
18       SW <= "00000001";
19       WAIT FOR 10 ns;
20       REPORT "SW = " & to_string(SW) & "; HEX0 = " & to_string(HEX0);
21
22       SW <= "00000010";
23       WAIT FOR 10 ns;
24       REPORT "SW = " & to_string(SW) & "; HEX0 = " & to_string(HEX0);
25
26       SW <= "00000100";
27       WAIT FOR 10 ns;
28       REPORT "SW = " & to_string(SW) & "; HEX0 = " & to_string(HEX0);
29
30       SW <= "00001000";
31       WAIT FOR 10 ns;
32       REPORT "SW = " & to_string(SW) & "; HEX0 = " & to_string(HEX0);
33
34       SW <= "00010000";
35       WAIT FOR 10 ns;
36       REPORT "SW = " & to_string(SW) & "; HEX0 = " & to_string(HEX0);
37
38       SW <= "00100000";
39       WAIT FOR 10 ns;
40       REPORT "SW = " & to_string(SW) & "; HEX0 = " & to_string(HEX0);
41
42       SW <= "01000000";
43       WAIT FOR 10 ns;
44       REPORT "SW = " & to_string(SW) & "; HEX0 = " & to_string(HEX0);
45
46       SW <= "10000000";
47       WAIT FOR 10 ns;
48       REPORT "SW = " & to_string(SW) & "; HEX0 = " & to_string(HEX0);
49
50       WAIT;
51    END PROCESS;
52 END ARCHITECTURE test;
```

Figure 8 - Code VHDL du testbench #3 a)

Ce testbench vérifie exactement la même chose que le testbench de la partie #2, sauf que cette fois-ci on regarde si la valeur de HEX0 correspond bien au bon affichage 7-segments. Voici à quel vecteur correspond les chiffres de 0 à 9 :

```
constant SS_0 : sseg_type := 7b"1000000";
constant SS_1 : sseg_type := 7b"1111001";
constant SS_2 : sseg_type := 7b"0100100";
constant SS_3 : sseg_type := 7b"0110000";
constant SS_4 : sseg_type := 7b"0011001";
constant SS_5 : sseg_type := 7b"0010010";
constant SS_6 : sseg_type := 7b"0000010";
constant SS_7 : sseg_type := 7b"1111000";
constant SS_8 : sseg_type := 7b"0000000";
constant SS_9 : sseg_type := 7b"0010000";
constant SOFF : sseg_type := 7b"1111111";
```

Figure 9 - Valeur de chaque chiffre sur un 7-segments

```
# ** Note: Testbench starting...
#    Time: 0 ps  Iteration: 0  Instance: /partie3a_tb
# ** Note: SW = 00000001; HEX0 = 1000000
#    Time: 10 ns  Iteration: 0  Instance: /partie3a_tb
# ** Note: SW = 00000010; HEX0 = 1111001
#    Time: 20 ns  Iteration: 0  Instance: /partie3a_tb
# ** Note: SW = 00000100; HEX0 = 0100100
#    Time: 30 ns  Iteration: 0  Instance: /partie3a_tb
# ** Note: SW = 00001000; HEX0 = 0110000
#    Time: 40 ns  Iteration: 0  Instance: /partie3a_tb
# ** Note: SW = 00010000; HEX0 = 0011001
#    Time: 50 ns  Iteration: 0  Instance: /partie3a_tb
# ** Note: SW = 00100000; HEX0 = 0010010
#    Time: 60 ns  Iteration: 0  Instance: /partie3a_tb
# ** Note: SW = 01000000; HEX0 = 0000010
#    Time: 70 ns  Iteration: 0  Instance: /partie3a_tb
# ** Note: SW = 10000000; HEX0 = 1111000
#    Time: 80 ns  Iteration: 0  Instance: /partie3a_tb
```

Figure 10 - Résultat du testbench #3 a)

Comme on peut le voir dans la figure ci-dessus, les valeurs correspondent bien à la position du bit le plus significatif.

Question 3b : Réalisation d'un multiplexeur simple

Explication du programme

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3
4  PACKAGE lab1_3b IS
5     COMPONENT partie3b IS
6        PORT (
7           SW : IN STD_LOGIC_VECTOR(8 DOWNTO 0);
8           LEDR : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
9        );
10    END COMPONENT;
11 END PACKAGE;
12 -------------------------------------------------------
13 LIBRARY ieee;
14 USE ieee.std_logic_1164.ALL;
15
16 ENTITY partie3b IS
17    PORT (
18       SW : IN STD_LOGIC_VECTOR(8 DOWNTO 0);
19       LEDR : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
20    );
21 END partie3b;
22
23 ARCHITECTURE arch OF partie3b IS
24    SIGNAL s : STD_LOGIC;
25    SIGNAL x, y : STD_LOGIC_VECTOR(3 DOWNTO 0);
26 BEGIN
27    s <= SW(8);
28    x <= SW(7 DOWNTO 4);
29    y <= SW(3 DOWNTO 0);
30    WITH s SELECT
31       LEDR <= x WHEN '0', y WHEN OTHERS;
32 END arch;
```

Figure 11 - Code VHDL #3 b)

Ce code prend neuf interrupteurs en entrées et a trois témoins lumineux en sortie. Selon la valeur de l'interrupteur le plus à gauche (le 9ème), on décide si on envoie les quatre premiers interrupteurs ou les quatre derniers dans la sortie (c'est-à-dire les lumières). Ceci est donc un multiplexeur 2 à 1.

Testbench

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.numeric_std.ALL;
USE work.lab1_3b.ALL;

ENTITY partie3b_tb IS
END partie3b_tb;

ARCHITECTURE test OF partie3b_tb IS
   SIGNAL SW : STD_LOGIC_VECTOR (8 DOWNTO 0);
   SIGNAL LEDR : STD_LOGIC_VECTOR (3 DOWNTO 0);
BEGIN
   DUT : partie3b PORT MAP(SW, LEDR);
   PROCESS BEGIN
      REPORT "Testbench starting...";
      FOR i IN 0 TO 511 LOOP
         SW <= STD_LOGIC_VECTOR(to_unsigned(i, SW'length));
         WAIT FOR 10 ns;
         IF SW(8) = '0' THEN
            IF LEDR /= SW(7 DOWNTO 4) THEN
               REPORT "SW = " & to_string(SW) & "; LEDR = " & to_string(LEDR);
            END IF;
         ELSIF SW(8) = '1' THEN
            IF LEDR /= SW(3 DOWNTO 0) THEN
               REPORT "SW = " & to_string(SW) & "; LEDR = " & to_string(LEDR);
            END IF;
         END IF;
      END LOOP;
      WAIT;
   END PROCESS;
END ARCHITECTURE test;
```

Figure 12 - Code VHDL du testbench #3 b)

Dans ce testbench, on vérifie simplement si, lorsque le 9ème interrupteur à 0, la valeur de sortie est égale aux quatre premiers interrupteurs. Si ce n'est pas le cas, on écrit un message. On fait la même chose pour l'interrupteur à 1 avec les quatre derniers interrupteurs.

```
# ** Note: Testbench starting...
#    Time: 0 ps  Iteration: 0  Instance: /partie3b_tb
```

Figure 13 - Résultat du testbench #3 b)

Comme on peut le voir dans la figure ci-dessus, puisqu'il n'y a aucune sortie, les valeurs à la sortie sont les valeurs souhaitées.

Question 3c : Réalisation d'un multiplexeur complexe

Explication du programme

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3
4  PACKAGE lab1_3c IS
5     COMPONENT partie3c IS
6        PORT (
7           SW : IN STD_LOGIC_VECTOR(9 DOWNTO 0);
8           LEDR : OUT STD_LOGIC_VECTOR(1 DOWNTO 0)
9        );
10    END COMPONENT;
11 END PACKAGE;
12 ----------------------------------------------------------
13 LIBRARY ieee;
14 USE ieee.std_logic_1164.ALL;
15
16 ENTITY partie3c IS
17    PORT (
18       SW : IN STD_LOGIC_VECTOR(9 DOWNTO 0);
19       LEDR : OUT STD_LOGIC_VECTOR(1 DOWNTO 0)
20    );
21 END partie3c;
22
23 ARCHITECTURE arch OF partie3c IS
24    SIGNAL s, u, v, w, x : STD_LOGIC_VECTOR(1 DOWNTO 0);
25 BEGIN
26    s <= SW(9 DOWNTO 8);
27    u <= SW(7 DOWNTO 6);
28    v <= SW(5 DOWNTO 4);
29    w <= SW(3 DOWNTO 2);
30    x <= SW(1 DOWNTO 0);
31    WITH s SELECT
32       LEDR <= u WHEN 2d"0", v WHEN 2d"1", w WHEN 2d"2", x WHEN 2d"3", 2d"0" WHEN OTHERS;
33 END arch;
```

Figure 14 - Code VHDL #3 c)

Ce code fait pratiquement la même chose que la partie #3 b), mais cette fois-ci deux interrupteurs contrôlent quelle valeur sortira de ce multiplexeur (donc 2 bits). Les huit autres interrupteurs sont divisés en groupe de deux, ce qui résulte donc à quatre groupes de 2 bits. Ceci est donc un multiplexeur 4 à 2.

Testbench

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.numeric_std.ALL;
USE work.lab1_3c.ALL;

ENTITY partie3c_tb IS
END partie3c_tb;

ARCHITECTURE test OF partie3c_tb IS
   SIGNAL SW : STD_LOGIC_VECTOR (9 DOWNTO 0);
   SIGNAL LEDR : STD_LOGIC_VECTOR (1 DOWNTO 0);
BEGIN
   DUT : partie3c PORT MAP(SW, LEDR);
   PROCESS BEGIN
      REPORT "Testbench starting...";
      FOR i IN 0 TO 511 LOOP
         SW <= STD_LOGIC_VECTOR(to_unsigned(i, SW'length));
         WAIT FOR 10 ns;
         IF SW(9 DOWNTO 8) = "00" THEN
            IF LEDR /= SW(7 DOWNTO 6) THEN
               REPORT "SW = " & to_string(SW) & "; LEDR = " & to_string(LEDR);
            END IF;
         ELSIF SW(9 DOWNTO 8) = "01" THEN
            IF LEDR /= SW(5 DOWNTO 4) THEN
               REPORT "SW = " & to_string(SW) & "; LEDR = " & to_string(LEDR);
            END IF;
         ELSIF SW(9 DOWNTO 8) = "10" THEN
            IF LEDR /= SW(3 DOWNTO 1) THEN
               REPORT "SW = " & to_string(SW) & "; LEDR = " & to_string(LEDR);
            END IF;
         ELSIF SW(9 DOWNTO 8) = "11" THEN
            IF LEDR /= SW(1 DOWNTO 0) THEN
               REPORT "SW = " & to_string(SW) & "; LEDR = " & to_string(LEDR);
            END IF;
         END IF;
      END LOOP;
      WAIT;
   END PROCESS;
END ARCHITECTURE test;
```

Figure 15 - Code VHDL du testbench #3 c)

Ce testbench est très similaire à celui de la partie #3 b), mais cette fois-ci il vérifie pour les quatre situations possibles.

```
# ** Note: Testbench starting...
#    Time: 0 ps  Iteration: 0  Instance: /partie3c_tb
```

Figure 16 - Résultat du testbench #3 c)

Comme on peut le voir dans la figure ci-dessus, puisqu'il n'y a aucune sortie, les valeurs à la sortie sont les valeurs souhaitées.

Question 4 : Utilisation d'un décodeur 7-segments

Explication du programme

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

PACKAGE lab1_4 IS
    COMPONENT partie4 IS
        PORT (
            SW : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
            HEX0 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
        );
    END COMPONENT;

    SUBTYPE sseg_type IS STD_LOGIC_VECTOR(6 DOWNTO 0);

    CONSTANT SS_A : sseg_type := 7b"0001000";
    CONSTANT SS_B : sseg_type := 7b"0000011";
    CONSTANT SS_C : sseg_type := 7b"0100111";
    CONSTANT SS_D : sseg_type := 7b"0100001";
    CONSTANT SOFF : sseg_type := 7b"1111111";

    COMPONENT sseg_letter IS
        PORT (
            bin : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
            segs : OUT sseg_type
        );
    END COMPONENT;
END PACKAGE;
-------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE work.lab1_4.ALL;

ENTITY partie4 IS
    PORT (
        SW : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
        HEX0 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
    );
END partie4;

ARCHITECTURE arch OF partie4 IS
BEGIN
    sseg0 : sseg_letter PORT MAP(SW, HEX0);
END arch;
-------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE work.lab1_4.ALL;

ENTITY sseg_letter IS
    PORT (
        bin : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
        segs : OUT sseg_type
    );
END sseg_letter;
ARCHITECTURE impl OF sseg_letter IS
BEGIN
    PROCESS (ALL) BEGIN
        CASE bin IS
            WHEN 2d"0" => segs <= SS_A;
            WHEN 2d"1" => segs <= SS_B;
            WHEN 2d"2" => segs <= SS_C;
            WHEN 2d"3" => segs <= SS_D;
            WHEN OTHERS => segs <= SOFF;
        END CASE;
    END PROCESS;
END impl;
```

Figure 17 - Code VHDL #4

Puisque les 4 premières lettres du nom de famille de Maxime Simard (Sima) ne pouvaient pas tous être écrites sur un 7-segments, nous avons simplement utiliser les lettres « Abcd ». Tout d'abord, nous avons créé une entité représentant un décodeur 7-segments pour les lettres. Pour ce faire, nous nous sommes inspirés du code du livre en définissant un nouveau type (sseg_type) et en définissant une constante pour chaque lettre. Cette entité prend ensuite en entrée un vecteur de 2 bits, et chaque combinaison de ce vecteur correspond à une lettre. Voici le tableau de correspondance :

Tableau 1 - Affichage en fonction des interrupteurs

Valeur entrée (2 bits)	Affichage sortie (7 bits)
0 (00)	A (0001000)
1 (01)	b (0000011)
2 (10)	c (0100111)
3 (11)	d (0100001)

Le décodeur de lettre vers 7-segment prend donc en entrée 2 bits, et retourne une des quatre lettres à l'aide de 7 bits dépendant l'entrée reçue.

L'entité « partie4 », elle, instancie ensuite simplement l'entité de décodage vers 7-segments en lui donnant en entrée deux interrupteurs.

Testbench

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.numeric_std.ALL;
USE work.lab1_4.ALL;

ENTITY partie4_tb IS
END partie4_tb;

ARCHITECTURE test OF partie4_tb IS
    SIGNAL SW : STD_LOGIC_VECTOR(1 DOWNTO 0);
    SIGNAL HEX0 : STD_LOGIC_VECTOR(6 DOWNTO 0);
BEGIN
    DUT : partie4 PORT MAP(SW, HEX0);
    PROCESS BEGIN
        REPORT "Testbench starting...";

        SW <= "00";
        WAIT FOR 10 ns;
        REPORT "SW = " & to_string(SW) & "; HEX0 = " & to_string(HEX0);

        SW <= "01";
        WAIT FOR 10 ns;
        REPORT "SW = " & to_string(SW) & "; HEX0 = " & to_string(HEX0);

        SW <= "10";
        WAIT FOR 10 ns;
        REPORT "SW = " & to_string(SW) & "; HEX0 = " & to_string(HEX0);

        SW <= "11";
        WAIT FOR 10 ns;
        REPORT "SW = " & to_string(SW) & "; HEX0 = " & to_string(HEX0);

        WAIT;
    END PROCESS;
END ARCHITECTURE test;
```

Figure 18 - Code VHDL du testbench #4

Ce testbench vérifie exactement la même chose que le testbench de la partie #4 a), mais cette fois-ci on regarde si la valeur de HEX0 correspond bien à la lettre souhaitée.

```
# ** Note: Testbench starting...
#    Time: 0 ps  Iteration: 0  Instance: /partie4_tb
# ** Note: SW = 00; HEX0 = 0001000
#    Time: 10 ns  Iteration: 0  Instance: /partie4_tb
# ** Note: SW = 01; HEX0 = 0000011
#    Time: 20 ns  Iteration: 0  Instance: /partie4_tb
# ** Note: SW = 10; HEX0 = 0100111
#    Time: 30 ns  Iteration: 0  Instance: /partie4_tb
# ** Note: SW = 11; HEX0 = 0100001
#    Time: 40 ns  Iteration: 0  Instance: /partie4_tb
```

Figure 19 - Résultat du testbench #4

Comme on peut le voir dans la figure ci-dessus, les valeurs correspondent bien à la lettre souhaitée.

Question 5 : Révision

Explication du programme

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3
4  PACKAGE lab1_5 IS
5     COMPONENT partie5 IS
6        PORT (
7           SW : IN STD_LOGIC_VECTOR(9 DOWNTO 0);
8           HEX0, HEX1, HEX2, HEX3 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
9        );
10    END COMPONENT;
11 END PACKAGE;
12 ------------------------------------------------------
13 LIBRARY ieee;
14 USE ieee.std_logic_1164.ALL;
15 USE work.lab1_4.ALL;
16 USE work.lab1_3c.ALL;
17
18 ENTITY partie5 IS
19    PORT (
20       SW : IN STD_LOGIC_VECTOR(9 DOWNTO 0);
21       HEX0, HEX1, HEX2, HEX3 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
22    );
23 END partie5;
24
25 ARCHITECTURE arch OF partie5 IS
26    SIGNAL choix : STD_LOGIC_VECTOR(1 DOWNTO 0);
27 BEGIN
28    inst_partie3c : partie3c PORT MAP(SW, choix);
29
30    sseg3 : sseg_letter PORT MAP(2d"0" XOR choix, HEX3);
31    sseg2 : sseg_letter PORT MAP(2d"1" XOR choix, HEX2);
32    sseg1 : sseg_letter PORT MAP(2d"2" XOR choix, HEX1);
33    sseg0 : sseg_letter PORT MAP(2d"3" XOR choix, HEX0);
34 END arch;
```

Figure 20 - Code VHDL #5

Ce code importe tout d'abord le multiplexeur 4 à 2 de la partie #3 c), ainsi que le décodeur 7-segments de lettre de la partie #4. L'entité prend en entrée 10 interrupteurs et a en sortie les quatre afficheurs 7-segments de droite. On instancie le multiplexeur en donnant en entrée l'ensemble des interrupteurs, et en sortie un signal correspondant au choix d'affichage. On instancie ensuite les quatre décodeurs 7-segments puis donnons en entrée la valeur normalement souhaitée (de 0 à 3) combiné avec un XOR du choix, permettant donc de mélanger l'ordre des lettres sur les quatre afficheurs. En sortie des décodeurs, nous avons finalement les quatre afficheurs.

Testbench

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.numeric_std.ALL;
USE work.lab1_5.ALL;

ENTITY partie5_tb IS
END partie5_tb;

ARCHITECTURE test OF partie5_tb IS
    SIGNAL SW : STD_LOGIC_VECTOR (9 DOWNTO 0);
    SIGNAL HEX0, HEX1, HEX2, HEX3 : STD_LOGIC_VECTOR (6 DOWNTO 0);
BEGIN
    DUT : partie5 PORT MAP(SW, HEX0, HEX1, HEX2, HEX3);
    PROCESS BEGIN
        REPORT "Testbench starting...";

        SW <= "0000000000";
        WAIT FOR 10 ns;
        REPORT "SW = " & to_string(SW) & "; HEX0 = " & to_string(HEX0) & "; HEX1 = " & to_string(HEX1) & "; HEX2 = " & to_string(HEX2) & "; HEX3 = " & to_string(HEX3);

        SW <= "0001000000";
        WAIT FOR 10 ns;
        REPORT "SW = " & to_string(SW) & "; HEX0 = " & to_string(HEX0) & "; HEX1 = " & to_string(HEX1) & "; HEX2 = " & to_string(HEX2) & "; HEX3 = " & to_string(HEX3);

        SW <= "0010000000";
        WAIT FOR 10 ns;
        REPORT "SW = " & to_string(SW) & "; HEX0 = " & to_string(HEX0) & "; HEX1 = " & to_string(HEX1) & "; HEX2 = " & to_string(HEX2) & "; HEX3 = " & to_string(HEX3);

        SW <= "0011000000";
        WAIT FOR 10 ns;
        REPORT "SW = " & to_string(SW) & "; HEX0 = " & to_string(HEX0) & "; HEX1 = " & to_string(HEX1) & "; HEX2 = " & to_string(HEX2) & "; HEX3 = " & to_string(HEX3);

        SW <= "0100000000";
        WAIT FOR 10 ns;
        REPORT "SW = " & to_string(SW) & "; HEX0 = " & to_string(HEX0) & "; HEX1 = " & to_string(HEX1) & "; HEX2 = " & to_string(HEX2) & "; HEX3 = " & to_string(HEX3);

        SW <= "0100010000";
        WAIT FOR 10 ns;
        REPORT "SW = " & to_string(SW) & "; HEX0 = " & to_string(HEX0) & "; HEX1 = " & to_string(HEX1) & "; HEX2 = " & to_string(HEX2) & "; HEX3 = " & to_string(HEX3);

        SW <= "0100100000";
        WAIT FOR 10 ns;
        REPORT "SW = " & to_string(SW) & "; HEX0 = " & to_string(HEX0) & "; HEX1 = " & to_string(HEX1) & "; HEX2 = " & to_string(HEX2) & "; HEX3 = " & to_string(HEX3);

        SW <= "0100110000";
        WAIT FOR 10 ns;
        REPORT "SW = " & to_string(SW) & "; HEX0 = " & to_string(HEX0) & "; HEX1 = " & to_string(HEX1) & "; HEX2 = " & to_string(HEX2) & "; HEX3 = " & to_string(HEX3);

        WAIT;
    END PROCESS;
END ARCHITECTURE test;
```

Figure 21 - Code VHDL du testbench #5

Dans ce testbench, on vérifie simplement que les bits 8 et 9 contrôlent bel et bien quelle paire d'interrupteur indiquera l'ordre des lettres. On vérifie aussi que ces interrupteurs changent bien l'ordre d'affichage.

```
# ** Note: Testbench starting...
#    Time: 0 ps  Iteration: 0  Instance: /partie5_tb
# ** Note: SW = 0000000000; HEX0 = 0100001; HEX1 = 0100111; HEX2 = 0000011; HEX3 = 0001000
#    Time: 10 ns  Iteration: 0  Instance: /partie5_tb
# ** Note: SW = 0001000000; HEX0 = 0100111; HEX1 = 0100001; HEX2 = 0001000; HEX3 = 0000011
#    Time: 20 ns  Iteration: 0  Instance: /partie5_tb
# ** Note: SW = 0010000000; HEX0 = 0000011; HEX1 = 0001000; HEX2 = 0100001; HEX3 = 0100111
#    Time: 30 ns  Iteration: 0  Instance: /partie5_tb
# ** Note: SW = 0011000000; HEX0 = 0001000; HEX1 = 0000011; HEX2 = 0100111; HEX3 = 0100001
#    Time: 40 ns  Iteration: 0  Instance: /partie5_tb
# ** Note: SW = 0100000000; HEX0 = 0100001; HEX1 = 0100111; HEX2 = 0000011; HEX3 = 0001000
#    Time: 50 ns  Iteration: 0  Instance: /partie5_tb
# ** Note: SW = 0100010000; HEX0 = 0100111; HEX1 = 0100001; HEX2 = 0001000; HEX3 = 0000011
#    Time: 60 ns  Iteration: 0  Instance: /partie5_tb
# ** Note: SW = 0100100000; HEX0 = 0000011; HEX1 = 0001000; HEX2 = 0100001; HEX3 = 0100111
#    Time: 70 ns  Iteration: 0  Instance: /partie5_tb
# ** Note: SW = 0100110000; HEX0 = 0001000; HEX1 = 0000011; HEX2 = 0100111; HEX3 = 0100001
#    Time: 80 ns  Iteration: 0  Instance: /partie5_tb
```

Figure 22 - Résultat du testbench #5

Comme on peut le voir, dans les quatre premiers cas c'est la première paire qui contrôle la sortie, puis dans les quatre derniers cas c'est la deuxième paire. On peut aussi bien voir que les lettres changent d'ordre selon l'entrée.

Conclusion

En conclusion, ce laboratoire nous a permis de mieux comprendre les aspects de base du VHDL, notamment la création d'un projet, la mise en place de « testbench », la création de package, l'importation de fichier, l'importance du fichier « DE10_Lite.qsf », le fonctionnement du « workspace » et l'utilité du « port map ». Il nous a aussi permis de mettre en place nous-même certaines composantes comme un décodeur et un multiplexeur, puis de combiner ces deux composantes dans une entité.

Références

Aucune référence.