Laboratoire 1

Switches, Lights, et Multiplexers

L'objectif de ce premier exercice est de prendre en main les notions d'entité et d'architecture en VHDL. Nous mettrons ces notions en pratique par l'interconnexion de périphériques simple d'entrées et de sorties (comme les SWitchs, les afficheurs 7 segments et les LED) du circuit FPGA.

Part I

La carte DE2 fournit

- 18 switchs reliés aux pins notées SW_{17-0},
- 18 LED rouges notées $LEDR_{17-0}$, utilisées pour afficher des valeurs binaires.
- 8 LED vertes notées $LEDG_{7-0}$.
- 4 boutons poussoirs KEY_{3-0}.
- 8 afficheurs 7 segments *HEX0 à HEX7*.

La figure 1 montre un exemple simple de code VHDL utilisant les Switchs et affichant leur état sur les LED. L'affectation des signaux est réalisée ici bit à bit même si l'affectation du tableau complet de SW sur LEDR peut également être réalisé.

```
LEDR(17) <= SW(17);
LEDR(16) <= SW(16);
...
LEDR(0) <= SW(0);
```

Les noms données aux pins d'entrées/sorties sont fixés dans un fichier spécifique à la carte DE2, appelé *DE2_pin_assignments.qsf* ou DE2_115_pin_assignments.qsf dans le cas des 2 cartes disponibles au département SI de l'UCP. C'est pourquoi vous devez utiliser exclusivement ces noms de périphériques dans votre code, à moins de réaliser votre propre fichier *.qsf*. Ce fichier précise par exemple que le bit SW_0 est connecté au pin *N25* du FPGA, $LEDR_0$ sur le pin *AE23* pour la carte DE2. Pour utiliser ces affectations de pins, vous devez donc au préalable les importer comme indiqué dans la figure .

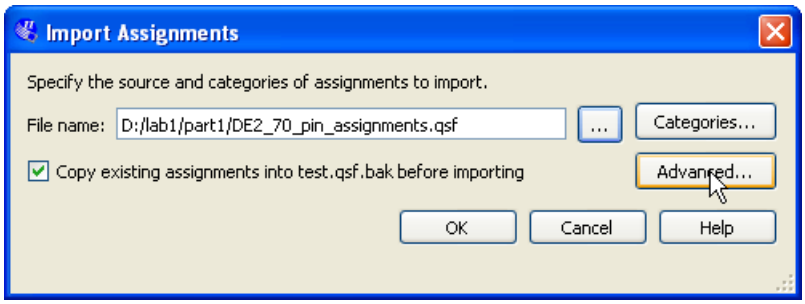


Figure 1. DE2-70 Import Assignments window.

(Noter que Quartus II utilise des crochets [ ] pour les types tableaux, alors que la syntaxe VHDL utilise des parenthèses ( )).

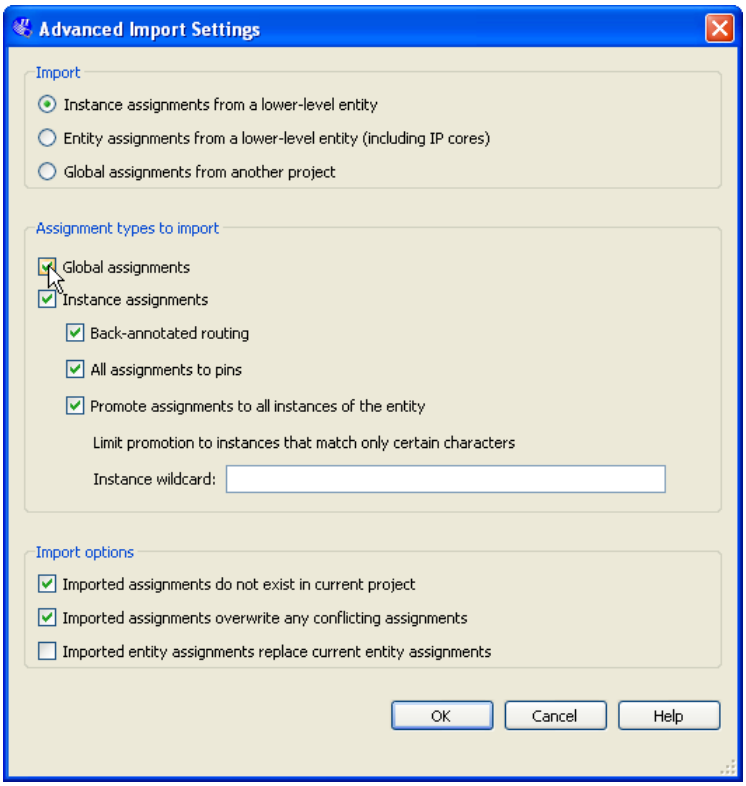


Figure 2. DE2-70 Advanced Import Settings window.

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

-- Simple module that connects the SW switches to the LEDR lights
ENTITY part1 IS
    PORT ( SW    : IN    STD_LOGIC_VECTOR(17 DOWNTO 0);
           LEDR  : OUT   STD_LOGIC_VECTOR(17 DOWNTO 0)); -- red LEDs
END part1;

ARCHITECTURE Behavior OF part1 IS
BEGIN
    LEDR <= SW;
END Behavior
```

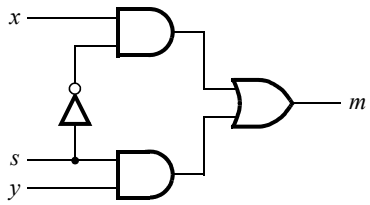
Figure 3. Code VHDL utilisant les périphériques des cartes DE2-series.

Réaliser les étapes suivants pour implémenter le code de la Figure 3.

1. Créer un nouveau projet Quartus II. Si vous utilisez la carte Altera DE2, sélectionner le circuit Cyclone II EP2C35F672C6, Cyclone II EP2C70F896C6 pour la DE2-70 board, ou Cyclone IV EP4CE115F29C7 pour la DE2-115 board.
2. Créer une entité VHDL pour le code de la Figure 3.
3. Inclure dans le projet le pin assignments pour la carte correspondant aux DE2-series et compilez le projet.
4. Télécharger le fichier de configuration du FPGA sur la carte avec Tools-Programmer. Vérifier le bon fonctionnement des périphériques.

Part II

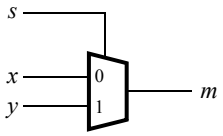
La Figure 4a montre une somme de produits qui implémente un circuit *multiplexer* 2 vers 1 avec une entrée de sélection s. Si $s = 0$ la sortie du mux m est égale à l'entrée x, et si $s = 1$ la sortie est y. La partie b de la figure donne la table de vérité et la partie c montre le symbole représentant le circuit.



a) Circuit

s	m
0	x
1	y

b) Truth table



c) Symbol

Figure 4. Un multiplexer 2 vers 1.

Le comportement du multiplexer peut être représenté par le code VHDL suivant:

m <= (NOT (s) AND x) OR (s AND y);

A vous d'écrire l'entité VHDL décrivant le circuit de la Figure 5a et qui utilise 8 affectations comme celle précédente. Le circuit utilise 2 entrées de 8 bits X et Y et produit la sortie de 8 bits M. Si $s = 0$ alors $M = X$, et si $s = 1$ alors $M = Y$. Nous appellerons ce circuit un multiplexeur 2 vers 1 de largeur 8. Son symbole est représenté en Figure 5b, dans lequel X, Y, et M apparaissent comme des bus de 8 bits.

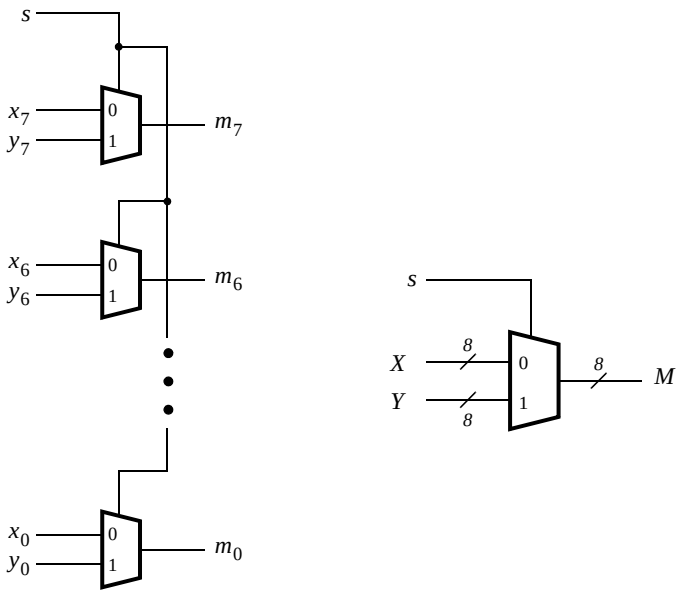


Figure 5. Multiplexer 2 vers 1 de 8 bits.

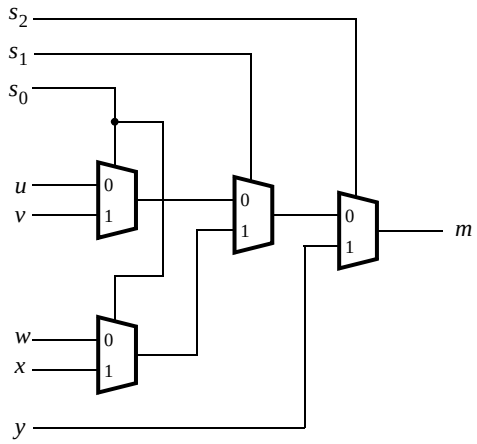
Vous devez donc

1. Créer un nouveau projet Quartus II.
2. Ajouter le fichier VHDL de description de ce circuit. Vous utiliserez les pins SW_{17} comme entrée s, les switches SW_{7-0} comme entrée X et SW_{15-8} comme entrée Y. Connecter enfin les entrées *SW* aux LED rouges *LEDR* et la sortie M aux LED vertes $LEDG_{7-0}$.
3. Importer les pins assignments, comme décrit précédemment.
4. Compiler le projet.
5. Télécharger et tester les fonctionnalités.

Part III

Après une sélection de un parmi 2, nous considérons maintenant un circuit de sélection d'une sortie *m* parmi 5 entrées u, v, w, x, et y. La partie a de la Figure 6 montre comme concevoir ce multiplexer 5 vers 1 en utilisant quatre mux 2-to-1. Le circuit utilise 3 bits de sélection $s_2s_1s_0$ et implémente la table de vérité de la Figure 6b.

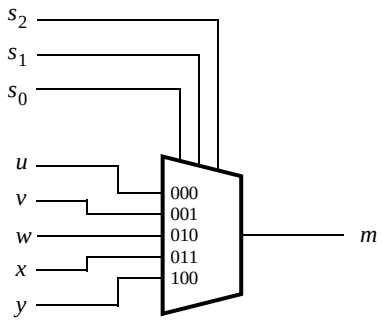
La figure 7 montre le résultat attendu pour ce circuit en travaillant sur des mots de 3 bits. Vous utiliserez pour cela 3 instances du circuit de la figure 6a.



a) Circuit

s_2	s_1	s_0	m
0	0	0	u
0	0	1	v
0	1	0	w
0	1	1	x
1	0	0	y
1	0	1	y
1	1	0	y
1	1	1	y

b) Truth table



c) Symbol

Figure 6. A 5-to-1 multiplexer.

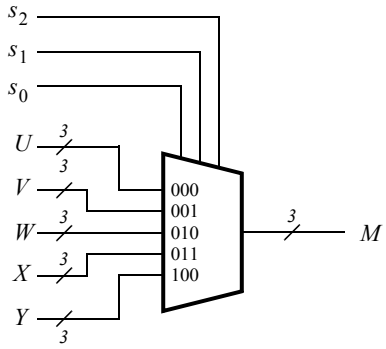


Figure 7. Multiplexer 5 vers 1 sur mots de 3 bits.

vous devez donc.

1. Créer un nouveau projet.

2. Créer une nouvelle entité VHDL. Les entrées de sélection seront reliées aux switches SW_{17-15}, les 15 switches restant SW_{14-0} seront reliées de manière à fournir au circuits les 5 entrées de 3 bits U à Y. Connecter tous les *SW* switches aux *LEDR* et la sortie M aux $LEDG_{2-0}$.

3. Inclure les pin assignments et compiler le projet.

4. Télécharger le bitstream. Assurer vous que chaque entrée de U à Y peut être sélectionné vers M.

Part IV

Nous allons nous resservir de ces circuits précédents pour décoder les afficheurs 7 segments de la Figure 8. Chaque décodeur dispose d'une entrée de 3 bits $c_2c_1c_0$. La Table 1 liste les caractères affichables en fonction de la valeur de $c_2c_1c_0$. Pour simplifier le décodeur seuls 4 caractères seront utilisés (plus le 'blank' de code 100 à 111).

Les 7 segments sont identifiés par un indice de 0 à 6 (cf. figure 8). Chaque segment est dit actif à l'état bas (la valeur du signal est à zéro). Vous devez donc écrire une entité VHDL qui active les 7 segments pour ce jeu de caractères.

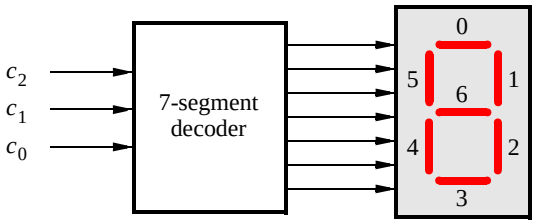


Figure 8. Un décodeur 7-segments.

$c_2c_1c_0$	Character
000	H
001	E
010	L
011	O
100	
101	
110	
111	

Table 1. Codes caractères.

Vous devrez donc :

1. Créer un nouveau projet Quartus II.

2. Implémenter le décodeur. Connecter $c_2c_1c_0$ aux switches SW_{2-0}, et les sorties du décodeur à l'afficheur *HEX0*. Chaque segment est noté $HEX0_0$, $HEX0_1$, ..., $HEX0_6$, comme dans la Figure 8 et dans le code suivant :

HEX0 : OUT STD_LOGIC_VECTOR(0 TO 6);

3. Importer les pin assignments, compiler le projet.

4. Télécharger et vérifier le comportement.

Part V

Considérez le circuit de la Figure 9.

Il utilise le multiplexeur 5 vers 1 précédent pour choisir quelle commande activera le code caractère de l'afficheur. Par exemple, chaque commande pourra commander un des 5 caractère H, E, L, O, et 'blank'. Les codes correspondant (Table 1) seront positionnées sur SW_{14-0}, et le choix du caractère à afficher se fera sur SW_{17-15}.

Une partie du code à réaliser est fournie en Figure 10. Ici, de manière à illustrer les commandes dinstanciation du VHDL les parties III et IV sont utilisées comme sous-circuits. Étendre ce code pour utiliser 5 afficheurs au lieu d'un seul.

Notre but sera d'afficher n'importe quel mot de 5 caractères de la Table 1, et de permettre la rotation de ces mots lorsque les switches SW_{17-15} sont modifiés.

Par exemple, si le mot affiché est HELLO, alors votre circuit produira les patterns illustrés dans la Table 2.

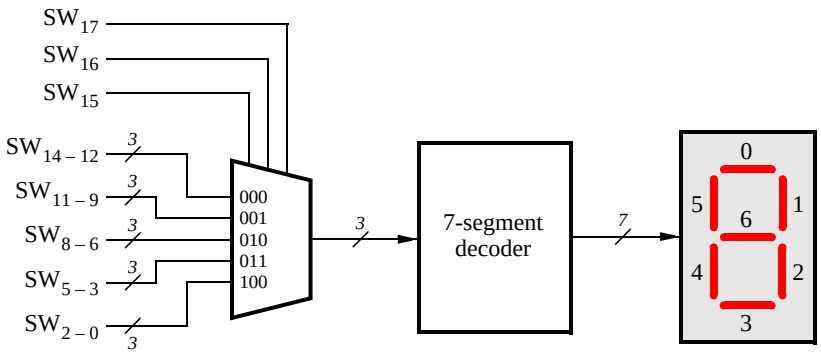


Figure 9. Un circuit qui permet de sélectionner et afficher un des 5 caractères.

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

ENTITY part5 IS
   PORT ( SW     : IN       STD_LOGIC_VECTOR(17 DOWNTO 0);
          HEX0  : OUT     STD_LOGIC_VECTOR(0 TO 6));
END part5;

ARCHITECTURE Behavior OF part5 IS
   COMPONENT mux_3bit_5to1
      PORT ( S, U, V, W, X, Y     : IN       STD_LOGIC_VECTOR(2 DOWNTO 0);
             M                    : OUT     STD_LOGIC_VECTOR(2 DOWNTO 0));
   END COMPONENT;
   COMPONENT char_7seg
      PORT ( C            : IN       STD_LOGIC_VECTOR(2 DOWNTO 0);
             Display     : OUT     STD_LOGIC_VECTOR(0 TO 6));
   END COMPONENT;
   SIGNAL M : STD_LOGIC_VECTOR(2 DOWNTO 0);
BEGIN
   M0: mux_3bit_5to1 PORT MAP (SW(17 DOWNTO 15), SW(14 DOWNTO 12), SW(11 DOWNTO 9),
      SW(8 DOWNTO 6), SW(5 DOWNTO 3), SW(2 DOWNTO 0), M);
   H0: char_7seg PORT MAP (M, HEX0);
END Behavior;

LIBRARY ieee;
USE ieee.std_logic_1164.all;

- - implements a 3-bit wide 5-to-1 multiplexer
ENTITY mux_3bit_5to1 IS
   PORT ( S, U, V, W, X, Y     : IN       STD_LOGIC_VECTOR(2 DOWNTO 0);
          M                    : OUT     STD_LOGIC_VECTOR(2 DOWNTO 0));
END mux_3bit_5to1;

ARCHITECTURE Behavior OF mux_3bit_5to1 IS

   . . . code not shown

END Behavior;

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY char_7seg IS
   PORT ( C            : IN       STD_LOGIC_VECTOR(2 DOWNTO 0);
          Display     : OUT     STD_LOGIC_VECTOR(0 TO 6));
END char_7seg;

ARCHITECTURE Behavior OF char_7seg IS

   . . . code not shown

END Behavior;
```

Figure 10. code VHDL de la Figure 9.

SW_{17} SW_{16} SW_{15}	Character pattern				
000	H	E	L	L	O
001	E	L	L	O	H
010	L	L	O	H	E
011	L	O	H	E	L
100	O	H	E	L	L

Table 2. Rotation du mot HELLO sur les 5 afficheurs.

Réaliser les étapes suivantes :

1. Créer un nouveau projet.
2. Coder votre entité VHDL de rotation. Connecter les switches SW_{17-15} aux entrées de sélection de chacune des 5 instances du multiplexeur 5 vers 1 de 3 bits. Connecter SW_{14-0} à chaque instance de multiplexeur pour réaliser le schéma d'affichage représenté dans la Table 2. Connecter les sorties des décodeurs aux afficheurs 7-segment *HEX4*, *HEX3*, *HEX2*, *HEX1*, et *HEX0*.
3. Compiler.
4. Télécharger et vérifier le fonctionnement en positionnant les codes caractères sur SW_{14-0} puis en jouant sur SW_{17-15} pour observer la rotation.

Si vous avez terminé, ajouter un compteur qui automatise la génération des commandes SW_17, SW_16, SW_15 pour la rotation des caractères.