Fondements des systèmes embarqués
TD N°03 : VHDL

Exercice 1

1) Décrire en VHDL le circuit multiplexeur, appelé Mux.

```
library ieee;
use ieee.std_logic_1164.all;
entity mux is
port
( Sel : in std_logic;
D,Y : in std_logic_vector(3 downto 0);
Z: out std_logic_vector(3 downto 0) );
end mux;

architecture comport of mux is
begin
P1: process(Sel,D,Y)
begin
if(Sel = '1') then
Z <= Y;
else
Z <= D;
end if;
end process P1;
end comport;
```

2) On suppose que le transcodeur 7 segment est de type anode commune, c'est-à-dire un 0 Volt sur un segment permet son allumage et un 5 Volt (1 logique) permet de l'éteindre. On suppose aussi que les segments sont connectés comme suit :

- Segment a est connecté à CodeHex(0)
- Segment b est connecté à CodeHex(1)
- Segment c est connecté à CodeHex(2)
- Segment d est connecté à CodeHex(3)
- Segment e est connecté à CodeHex(4)
- Segment f est connecté à CodeHex(5)
- Segment g est connecté à CodeHex(6)

Donner la valeur de la sortie CodeHex du transcodeur pour chaque code des chiffres 0 à F.

```
-- Decimal Z(3 downto 0) ----- output CodeHex(6 downto 0)
-- 0        "0000" -----          "1000000"
-- 1        "0001" -----          "1111001"
-- 2        "0010" -----          "0100100"
-- 3        "0011" -----          "0110000"
-- 4        "0100" -----          "0011001"
-- 5        "0101" -----          "0010010"
-- 6        "0110" -----          "0000010"
-- 7        "0111" -----          "1111000"
-- 8        "1000" -----          "0000000"
```

```
-- 9        "1001" -----        "0010000"
-- A        "1010" -----        "0001000"
---- b       "1011" -----       "0000011"
---- C       "1100" -----       "1000110"
---- d       "1101" -----       "0100001"
---- E       "1110" -----       "0000110"
---- F       "1111" -----       "0001110"
```

3) Décrire en VHDL comportemental le transcodeur 7 segments, appelé Dec7seg en utilisant un processus et l'instruction case du VHDL.

```
library ieee;
use ieee.std_logic_1164.all;
entity Dec7seg is
port (Z: in std_logic_vector(3 downto 0);
CodeHex : out std_logic_vector(6 downto 0) );
end Dec7seg;
architecture transcodeur of Dec7seg is
begin
P2: process(Z)
begin
case Z is
when "0000" => CodeHex <= "1000000"; -- 0
when "0001" => CodeHex <= "1111001"; -- 1
when "0010" => CodeHex <= "0100100"; -- 2
when "0011" => CodeHex <= "0110000"; -- 3
when "0100" => CodeHex <= "0011001"; -- 4
when "0101" => CodeHex <= "0010010"; -- 5
when "0110" => CodeHex <= "0000010"; -- 6
when "0111" => CodeHex <= "1111000"; -- 7
when "1000" => CodeHex <= "0000000"; -- 8
when "1001" => CodeHex <= "0010000"; -- 9
when "1010" => CodeHex <= "0001000"; -- A
when "1011" => CodeHex <= "0000011"; -- b
when "1100" => CodeHex <= "1000110"; -- C
when "1101" => CodeHex <= "0100001"; -- d
when "1110" => CodeHex <= "0000110"; -- E
when "1111" => CodeHex <= "0001110"; -- E
when others => CodeHex <= "1111111";
end case;
end process P2;
end transcodeur;
```

4) Le circuit SHIFTER présenté sur la figure précédente permet 5 fonctions de décalage en fonction de l'entrée Fct codée sur trois bits. Ces 5 fonctions sont résumées dans le tableau ci-après.

Entrée Fct(2 downto 0)	Valeur de Sortie Y(3 downto 0) en fonction de l'entrée D(3 downto 0)				Fonction de décalage réalisée par le circuit SHIFTER
	Y(3)	**Y(2)**	**Y(1)**	**Y(0)**	
"000 "	0	D(3)	D(2)	D(1)	Décalage à droite
"001 "	D(2)	D(1)	D(0)	0	Décalage à gauche
"010 "	D(0)	D(3)	D(2)	D(1)	Rotation à droite
"011 "	D(2)	D(1)	D(0)	D(3)	Rotation à gauche
"1xx " (x=0 ou 1) peu n'importe	D(3)	D(2)	D(1)	D(0)	Pas de rotation Y=D

Donner le code VHDL comportemental de ce circuit en utilisant un processus et l'instruction case.

```
library ieee;
use ieee.std_logic_1164.all;
entity SHIFTER is
port(
Fct : in std_logic_vector(2 downto 0);
D : in std_logic_vector(3 downto 0);
Y : out std_logic_vector(3 downto 0) );
end SHIFTER;
architecture arch of SHIFTER is
begin
P3: process(Fct,D)
begin
case Fct is
  -- decalage droite
when "000" => Y <= '0' & D(3 downto 1);
  -- decalage gauche
when "001" => Y <= D(2 downto 0)& '0';
  -- rotation droite
when "010" => Y <= (D(0),D(3),D(2),D(1) );
  -- rotation gauche
when "011" => Y <= (D(2),D(1),D(0),D(3) );
  -- pas de rotation ni décalage
when others => Y <= D;
end case;
end process P3;
```

5) Le circuit global aura comme entrées (Fct, D et sel) et comme sortie (CodeHex) comme présenté dans figure précédente. Donner la description VHDL structurelle du circuit global qui réutilise des instances des circuits décrits dans les questions précédentes.

```
library ieee;
use ieee.std_logic_1164.all;
entity GLOBAL is
port(
 Fct : in std_logic_vector(2 downto 0);
Sel: in std_logic;
D : in std_logic_vector(3 downto 0);
CodeHex : out std_logic_vector(6 downto 0));
end GLOBAL;
```

```
architecture archG of GLOBAL is
signal Y, Z : std_logic_vector(3 downto 0);

component SHIFTER
port(
Fct : in std_logic_vector(2 downto 0);
D : in std_logic_vector(3 downto 0);
Y : out std_logic_vector(3 downto 0));
end component;
component mux
port ( Sel : in std_logic;
D,Y : in std_logic_vector(3 downto 0);
Z: out std_logic_vector(3 downto 0) );
end component;

component Dec7seg
port(Z: in std_logic_vector(3 downto 0);
CodeHex : out std_logic_vector(6 downto 0) );
end component;

begin
SHIFTER1: SHIFTER  port map(Fct,D,Y);
mux1: mux port map(Sel,D ,Y,  Z);
Dec7seg1: Dec7seg  port map(Z,  CodeHex);
end archG;
```

6) Coder en VHDL un testbench qui pour la valeur de D égale à "0101", permet de stimuler toutes les fonctions de décalage et d'afficher le résultat Y sur l'afficheur 7 segments.

```
library ieee;
use ieee.std_logic_1164.all;
entity TB is
end TB;
architecture test of TB is
component GLOBAL
port(Fct : in std_logic_vector(2 downto 0);
Sel : in std_logic;
D : in std_logic_vector(3 downto 0);
CodeHex:out std_logic_vector(6 downto 0));
end component;
signal D : std_logic_vector(3 downto 0);
signal sel: std_logic;
signal Fct : std_logic_vector(2 downto 0);
signal CodeHex : std_logic_vector(6 downto 0);
begin
GLOBAL1: GLOBAL port map(Fct, Sel, D, CodeHex);
D <= "0101";
Sel <= '1';
stimuli: process
begin
Fct <= "000";
wait for 20 ns;
Fct <= "001";
```

```
wait for 20 ns;
Fct <= "010";
wait for 20 ns;
Fct <= "011";
wait for 20 ns;
Fct <= "100";
wait for 20 ns;
Fct <= "101";
wait for 20 ns;
Fct <= "110";
wait for 20 ns;
Fct <= "111";
wait;
end process stimuli;
end test ;
```

Exercice 2 : Circuit de complément à 2 générique

1) Donner le code VHDL comportemental de ce circuit en se basant sur des instructions concurrentes

```
library ieee;
use ieee.std_logic_1164.all;
entity OPP1 is
port (Ce, Xe : in std_logic;
Xs, Cs: out std_logic);
end OPP1;
architecture comport of OPP1 is
begin
Xs <= Ce or Xe;
Cs <= Ce xor Xe;
end comport;
```

2) Coder en VHDL le circuit OPP3 en utilisant l'instruction concurrente **for … generate**.

Indication : exprimer la sortie **S** en fonction de l'entrée **E** et du signal interne C (exprimer S(i) en fonction de E(i) et C(i) pour i allant de 0 à 2) (exprimer de même C(i+1) en fonction de E(i) et C(i) pour i allant de 0 à 2).

```
library ieee;
use ieee.std_logic_1164.all;
entity OPP3 is
port(
E : in std_logic_vector(2 downto 0);
S : out std_logic_vector(2 downto 0);
Cout: out std_logic);
end OPP3;

architecture comport3 of OPP3 is
signal C : std_logic_vector(3 downto 0);
begin
gen1: for I in 0 to 2 generate
C(I+1) <= C(I) xor E(I);
S(I) <= C(I) or E(I);
```

```
end generate gen1;
C(0) <= '0';
Cout <= C(3);
end comport3;
```

3) Donner le code VHDL du circuit générique OPPN correspondant (le cas général de la question 2).

```
library ieee;
use ieee.std_logic_1164.all;
entity OPPN is
generic(N : integer := 8);
port( E : in std_logic_vector(N-1 downto 0);
S : out std_logic_vector(N-1 downto 0) ;
Cout: out std_logic);
end OPPN;
architecture comportN of OPPN is
signal C : std_logic_vector(N downto 0);
begin
gen1: for I in 0 to N-1 generate
C(I+1) <= C(I) xor E(I);
S(I) <= C(I) or E(I);
end generate gen1;
C(0) <= '0';
Cout <= C(N);
end comportN;
```

4) Reprendre la même question en décrivant de façon structurelle le circuit générique de paramètre N qui réutilise N circuits OPP1.

```
library ieee;
use ieee.std_logic_1164.all;
entity OPPN is
generic(N : integer := 8);
port
( E : in std_logic_vector(N-1 downto 0);
S : out std_logic_vector(N-1 downto 0);
Cout: out std_logic);
end OPPN;
architecture Structurelle of OPPN is
signal C : std_logic_vector(N downto 0);
component OPP1 is
port
( Ce, Xe : in std_logic;
Xs, Cs: out std_logic
);
end component;
begin
gen2: for I in 0 to N-1 generate
OPPx: OPP1 port map(C(I), E(I),S(I),C(I+1));
```

```
-- OPPx: OPP1 port map(Xe => E(I), Ce => C(I),Xs => S(I),Cs => C(I+1));
end generate gen2;
C(0) <= '0';
Cout <= C(N);
end Structurelle;
```