Université Mohammed 1 er
Ecole National des Sciences Appliquées
Oujda

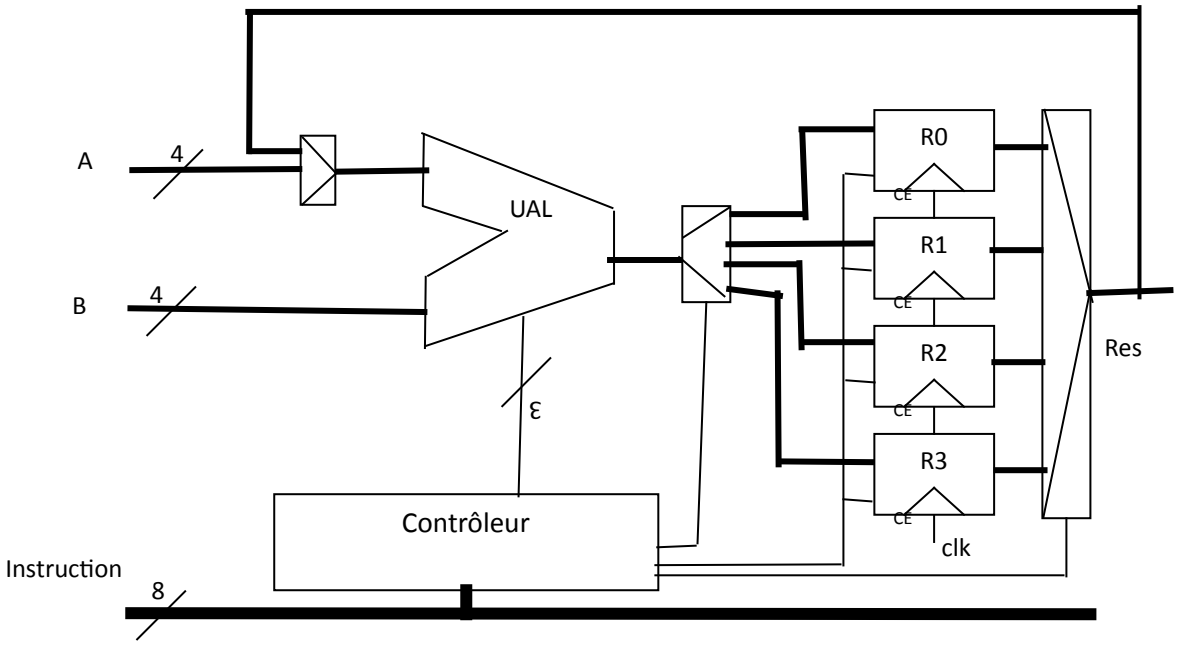
Mini-projet en VHDL

Travail réalisé par :
BOUSELHAM Loubna
MERZOUKI Alae
MOURHLI Med
RABYI Kaoutar
YAHYAOUI Fatima

Département : Génie Electrique
Année universitaire : 2008/2009

Sommaire

Conception d'un mini-calculateur



I. Conception de l'Unité arithmétique et logique :

1. Conception du comparateur 4 bits :

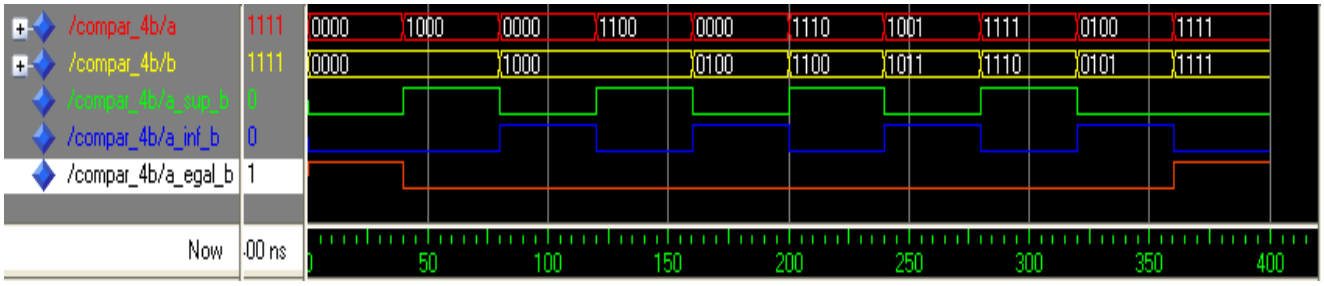
a. Table de vérité

Entrées				Sorties		
A_3, B_3	A_2, B_2	A_1, B_1	A_0, B_0	A_SUP_B	A_INF_B	A_EGAL_B
$A_3>B_3$	x	x	x	1	0	0
$A_3<B_3$	x	x	x	0	1	0
$A_3=B_3$	$A_2>B_2$	x	x	1	0	0
$A_3=B_3$	$A_2<B_2$	x	x	0	1	0
$A_3=B_3$	$A_2=B_2$	$A_1>B_1$	x	1	0	0
$A_3=B_3$	$A_2=B_2$	$A_1<B_1$	x	0	1	0
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0>B_0$	1	0	0
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0<B_0$	0	1	0
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	0	0	1

b. Fichier « compar_4b » :

```
 1    LIBRARY ieee;
 2    USE ieee.std_logic_1164.all;
 3    entity compar_4b is
 4    port (A,B :in std_logic_vector(3 downto 0);
 5    A_SUP_B,A_INF_B,A_EGAL_B :out std_logic);
 6    end compar_4b;
 7    architecture arch_compar_4b of compar_4b is
 8    begin
 9    A_SUP_B <= '1' WHEN A > B ELSE '0';
10    A_INF_B <= '1' WHEN A < B ELSE '0';
11    A_EGAL_B  <= '1' WHEN A = B ELSE '0';
12    end arch_compar_4b;
```

On tape les lignes de commande suivantes dans la fenêtre *transcript :*
force A 0000 0,1000 40,0000 80,1100 120,0000 160,1110 200,1001 240,1111 280,0100 320,1111 360
(On force le signal A à 0000 au temps 0, à 1000 au temps 40,à 0000 au temps 80,à 1100 au temps 120,à 0000 au temps 160,à 1110 au temps 200,à 1001 au temps 240,à 1111 au temps 280,à 0100 au temps 320 et à 1111 au temps 360)
force B 0000 0,0000 40,1000 80,1000 120,0100 160,1100 200,1011 240,1110 280,0101 320,1111 360
Run 400
La simulation donne les résultats suivants:

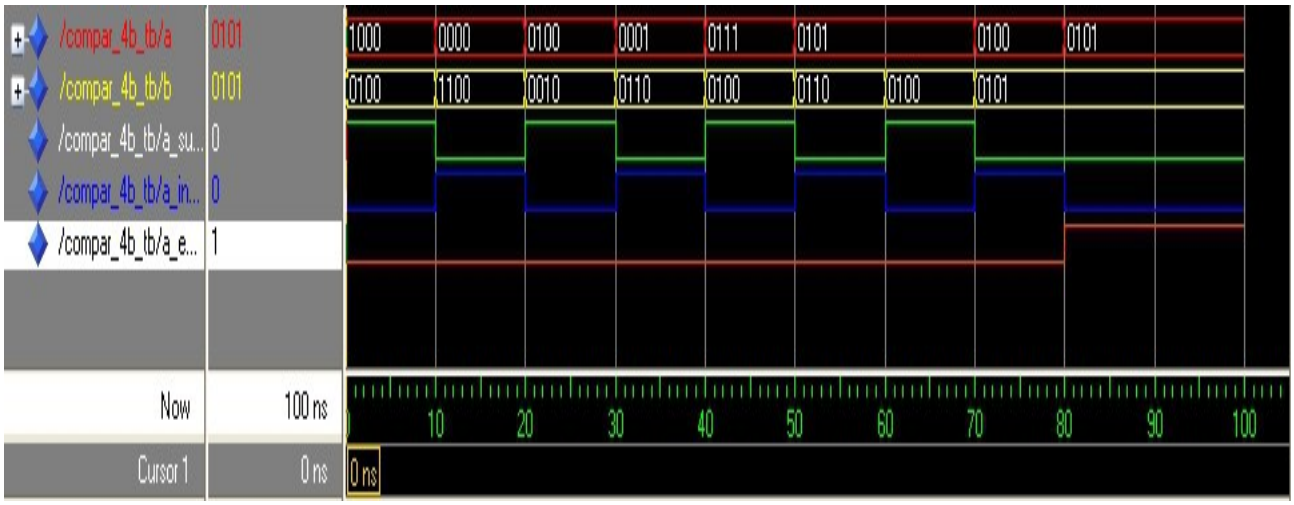


c. *Fichier « compar_4b_TB » :*

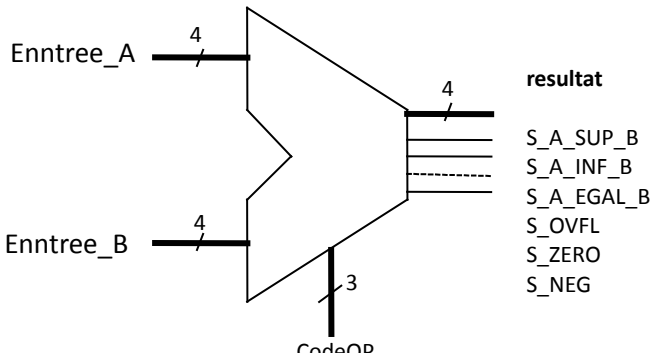
```
1   library ieee;
2   use ieee.std_logic_1164.all;
3       entity compar_4b_TB is
4       end compar_4b_TB;
5       architecture TB1 of compar_4b_TB is
6       component compar_4b
7           port (A,B : in std_logic_vector (3 downto 0);
8                 A_SUP_B,A_INF_B,A_EGAL_B :out std_logic);
9       end component;
10  signal A, B: std_logic_vector ( 3 downto 0);
11  signal A_SUP_B,A_INF_B,A_EGAL_B : std_logic;
12  begin
13      compar: compar_4b port map (A,B,A_SUP_B,A_INF_B,A_EGAL_B);
14      process
15          variable C:integer :=0;
16          begin
17              --case1:A3>B3
18              A <= "1000";
19              B <= "0100";
20              wait for 10ns;
21              assert (A_SUP_B ='1') report "SUP Error!" Severity error;
22              assert (A_INF_B='0') report "INF Error!" Severity error;
23              assert (A_EGAL_B='0') report "EGAL Error!" Severity error;
24              if(A_SUP_B/='1' or A_INF_B/='0' or A_EGAL_B/='0') then
25              C:=C+1;
26          end if;
27          -- case2: A3<B3
28              A <= "0000";
29              B <= "1100";
30              wait for 10ns;
31              assert (A_SUP_B='0') report "SUP Error!" Severity error;
32              assert (A_INF_B='1') report "INF Error!" Severity error;

33              assert (A_EGAL_B='0') report "EGAL Error!" Severity error;
34              if(A_SUP_B/='0' or A_INF_B/='1' or A_EGAL_B/='0') then
35              C:=C+1;
36          end if;

37          -- case3: A3=B3 A2>B2
38              A <= "0100";
39              B <= "0010";
40              wait for 10ns;
41              assert (A_SUP_B='1') report "SUP Error!" Severity error;
42              assert (A_INF_B='0') report "INF Error!" Severity error;
43              assert (A_EGAL_B='0') report "EGAL Error!" Severity error;
44              if(A_SUP_B/='1' or A_INF_B/='0' or A_EGAL_B/='0') then
45              C:=C+1;
46          end if;
```

```
47          -- case4:A3=B3 A2<B2
48              A <= "0001";
49              B <= "0110";
50              wait for 10ns;
51              assert (A_SUP_B='0') report "SUP Error!" Severity error;
52              assert (A_INF_B='1') report "INF Error!" Severity error;
53              assert (A_EGAL_B='0') report "EGAL Error!" Severity error;
54              if(A_SUP_B/='0' or A_INF_B/='1' or A_EGAL_B/='0') then
55              C:=C+1;
56          end if;
57          -- case5:A3=B3 A2=B2 A1>B1
58              A <= "0111";
59              B <= "0100";
60              wait for 10ns;
61              assert (A_SUP_B='1') report "SUP Error!" Severity error;
62              assert (A_INF_B='0') report "INF Error!" Severity error;
63              assert (A_EGAL_B='0') report "EGAL Error!" Severity error;
64              if(A_SUP_B/='1' or A_INF_B/='0' or A_EGAL_B/='0') then
65              C:=C+1;
66          end if;
67          -- case6:A3=B3 A2=B2 A1<B1
68              A <= "0101";
69              B <= "0110";
70              wait for 10ns;
71              assert (A_SUP_B='0') report "SUP Error!" Severity error;
72              assert (A_INF_B='1') report "INF Error!" Severity error;
73              assert (A_EGAL_B='0') report "EGAL Error!" Severity error;
74              if(A_SUP_B/='0' or A_INF_B/='1' or A_EGAL_B/='0') then
75              C:=C+1;
76          end if;

77          -- case7:A3=B3 A2=B2 A1=B1 A0>B0
78              A <= "0101";
79              B <= "0100";
80              wait for 10ns;
81              assert (A_SUP_B='1') report "SUP Error!" Severity error;
82              assert (A_INF_B='0') report "INF Error!" Severity error;
83              assert (A_EGAL_B='0') report "EGAL Error!" Severity error;
84              if(A_SUP_B/='1' or A_INF_B/='0' or A_EGAL_B/='0') then
85              C:=C+1;
86          end if;
87          -- case8:A3=B3 A2=B2 A1=B1 A0<B0
88              A <= "0100";
89              B <= "0101";
90              wait for 10ns;
91              assert (A_SUP_B='0') report "SUP Error!" Severity error;
92              assert (A_INF_B='1') report "INF Error!" Severity error;
93              assert (A_EGAL_B='0') report "EGAL Error!" Severity error;
94              if(A_SUP_B/='0' or A_INF_B/='1' or A_EGAL_B/='0') then
95              C:=C+1;
96          end if;
97          -- case9:A3=B3 A2=B2 A1=B1 A0=B0
98              A <= "0101";
99              B <= "0101";
100             wait for 10ns;
101             assert (A_SUP_B='0') report "SUP Error!" Severity error;
102             assert (A_INF_B='0') report "INF Error!" Severity error;
103             assert (A_EGAL_B='1') report "EGAL Error!" Severity error;
104             if(A_SUP_B/='0' or A_INF_B/='0' or A_EGAL_B/='1') then
105             C:=C+1;
106         end if;
```

La simulation donne les résultats suivants:



2. Conception de l'UAL :



a. Fichier « Opérations.vhd » :

Le champ CodeOp sert à indiquer la nature de l'opération à réaliser :

CodeOP	Opération à réaliser
000	Resultat <= A ET B
001	Resultat <= A NON ET B
010	Resultat <= A OU B
011	Resultat <= A NON OU B
100	Resultat <= A XOR B
101	Resultat <= NOT A
110	Resultat <= A + B
111	Resultat <= A - B

b. Fichier « Indications.vhd » :

- S_OVFL : sortie permettant d'indiquer si l'opération effectue un débordement.
- S_ZERO : sortie permettant d'indiquer si le résultat est nul.

```
 1    LIBRARY ieee;
 2    USE ieee.std_logic_1164.all;
 3    USE ieee.std_logic_unsigned.all;
 4
 5    entity operations is
 6    port  ( A,B: in std_logic_vector(3 downto 0);
 7            Code: in std_logic_vector(2 downto 0);
 8            RES:out std_logic_vector(3 downto 0));
 9       end operations;
10       Architecture arch_operations of operations is
11            begin
12                 Op:process (Code)
13                      begin
14                          case Code is
15                          when "000" => RES <= A and B;
16                          when "001" => RES <= A nand B;
17                          when "010" => RES <= A or B;
18                          when "011" => RES <= A nor B;
19                          when "100" => RES <= A xor B;
20                          when "101" => RES <= not A;
21                          when "110" => RES <= A + B;
22                          when others => RES <= A - B;
23                     end case;
24                 end process Op;
25            end arch_operations;
```

– S_NEG : sortie permettant d'indiquer sur le résultat est négatif.

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

entity out_indications is
port (RES:in std_logic_vector(3 downto 0);
      OVFL,NEG,ZERO:out std_logic);
  end out_indications;
  architecture arch_out_indications of out_indications is
      begin
          indic:process(RES)
          begin
            if (RES) > "1111" then OVFL <= '1';
            else OVFL <= '0';
            end if;
            if (RES) < "0000" then NEG <= '1';
            else NEG <= '0';
            end if;
            if (RES) ="0000" then ZERO <= '1';
            else ZERO <= '0';
            end if;
        end process indic;
    end arch_out_indications;
```

c. Fichier « Unite.vhd » : description complète de l'UAL :

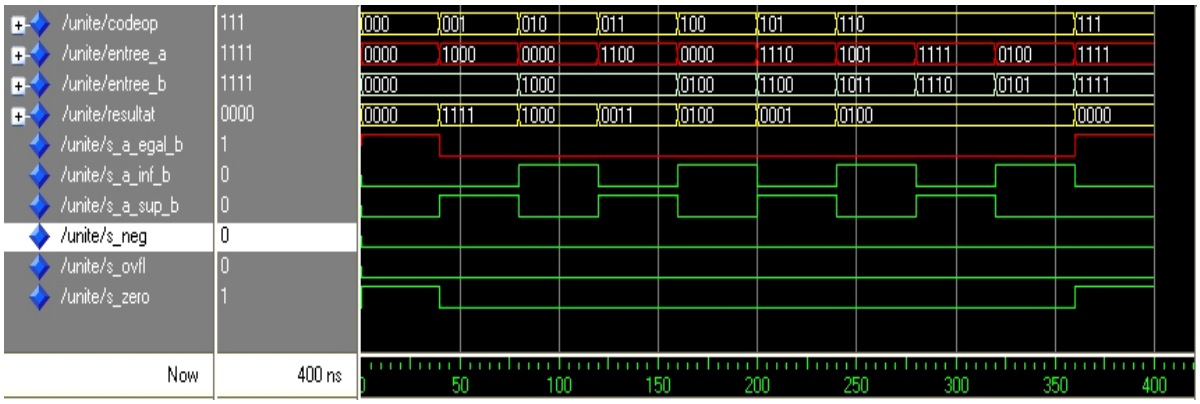
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
entity Unite is
port ( entree_A,entree_B: in std_logic_vector(3 downto 0);
       CodeOP:in std_logic_vector(2 downto 0);
       resultat:out std_logic_vector(3 downto 0);
       S_A_SUP_B,S_A_INF_B,S_A_EGAL_B,S_NEG,S_ZERO,S_OVFL:out std_logic);
   end Unite;
architecture arch_Unite of Unite is
    signal Y:std_logic_vector(3 downto 0);
component compar_4b
port (A,B :in std_logic_vector(3 downto 0);
      A_SUP_B :out std_logic;
      A_INF_B :out std_logic;
      A_EGAL_B :out std_logic);
  end component;
  component operations
  port ( A,B: in std_logic_vector(3 downto 0);
         Code: in std_logic_vector(2 downto 0);
         RES:out std_logic_vector(3 downto 0));
   end component;
   component out_indications
       port (RES:in std_logic_vector(3 downto 0);
             OVFL,NEG,ZERO:out std_logic);
         end component;
   begin
   Op:operations port map(entree_A,entree_B,CodeOp,Y);
   resultat <= Y;
   c:compar_4b port map(entree_A,entree_B,S_A_SUP_B,S_A_INF_B,S_A_EGAL_B);
   Indic:out_indications port map (Y,S_OVFL,S_NEG,S_ZERO);
   end arch_Unite;
```

On tape les lignes de commande suivantes dans la fenêtre *transcript* :

force entree_A 0000 0,1000 40,0000 80,1100 120,0000 160,1110 200,1001 240,1111 280,0100 320,1111 360

force entree_B 0000 0,0000 40,1000 80,1000 120,0100 160,1100 200,1011 240,1110 280,0101 320,1111 360

force CodeOp 000 0,001 40,010 80,011 120,100 160,101 200,110 240,111 360
run 400
La simulation donne les resultats suivants :



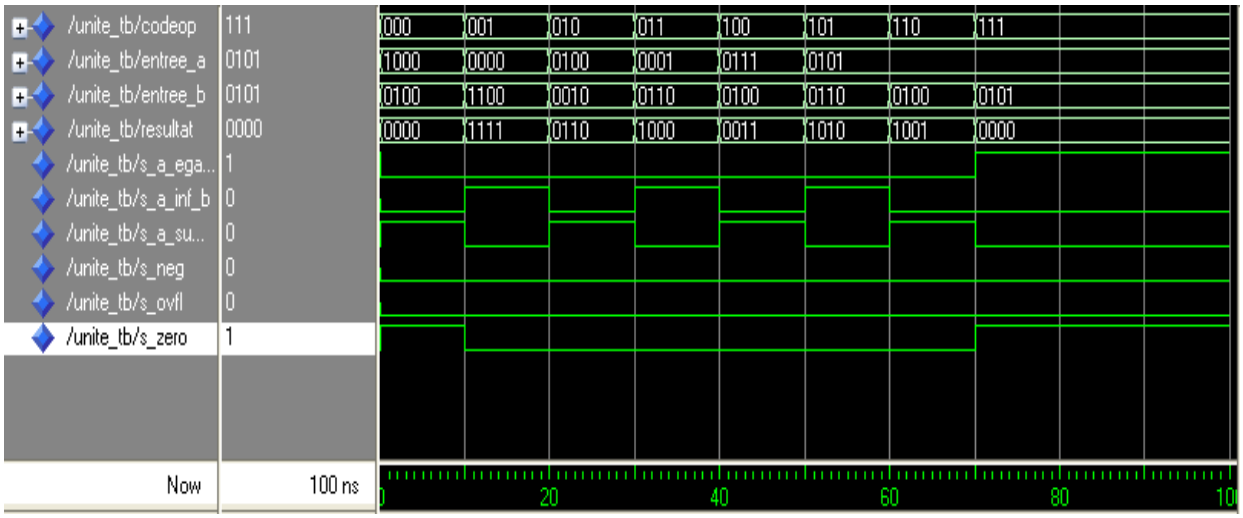
d. Fichier « Unite_TB.vhd » : TestBench de l'UAL :

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

    entity Unite_TB is
    end Unite_TB;
    architecture TB3 of Unite_TB is
    component Unite
    port (entree_A,entree_B: in std_logic_vector(3 downto 0);
          CodeOP:in std_logic_vector(2 downto 0);
          resultat:out std_logic_vector(3 downto 0);
          S_A_SUP_B,S_A_INF_B,S_A_EGAL_B,S_NEG,S_ZERO,S_OVFL:out std_logic);
   end component;
   signal entree_A,entree_B:std_logic_vector(3 downto 0);
   signal CodeOP:std_logic_vector(2 downto 0);
   signal resultat:std_logic_vector(3 downto 0);
   signal S_A_SUP_B,S_A_INF_B,S_A_EGAL_B,S_NEG,S_ZERO,S_OVFL:std_logic;
   begin
    U: Unite port map (entree_A,entree_B,CodeOP,resultat,S_A_SUP_B,S_A_INF_B,S_A_EGAL_B,S_NEG
    process
        variable C:integer :=0;
        begin
            --case1:
             CodeOp <= "000";
             entree_A <= "1000";
             entree_B <= "0100";
            wait for 10ns;
            assert (resultat ="0000")report "ET Error!" Severity error;
            assert (S_A_SUP_B='1') report "SUP Error!" Severity error;
            assert (S_A_INF_B='0') report "INF Error!" Severity error;
            assert (S_A_EGAL_B='0') report "EGAL Error!" Severity error;
            if (resultat /="1100"or S_A_SUP_B/='1' or S_A_INF_B/='0' or S_A_EGAL_B/='0')then
              C:=C+1;
        end if;
```

```
-- case4:
    CodeOp <= "011";
    entree_A <= "0001";
    entree_B <= "0110";
    wait for 10ns;
    assert (resultat ="1000")report "NON_OU Error!" Severity error;
    assert (S_A_SUP_B='0') report "SUP Error!" Severity error;
    assert (S_A_INF_B='1') report "INF Error!" Severity error;
    assert (S_A_EGAL_B='0') report "EGAL Error!" Severity error;
    if(resultat /="1000" or S_A_SUP_B/='0' or S_A_INF_B/='1' or S_A_EGAL_B/='0') then
    C:=C+1;
end if;
-- case5:
    CodeOp <= "100";
    entree_A <= "0111";
    entree_B <= "0100";
    wait for 10ns;
    assert (resultat ="0011")report " OU_Exclusive Error!" Severity error;
    assert (S_A_SUP_B='1') report "SUP Error!" Severity error;
    assert (S_A_INF_B='0') report "INF Error!" Severity error;
    assert (S_A_EGAL_B='0') report "EGAL Error!" Severity error;
    if(resultat /="0011" or S_A_SUP_B/='1' or S_A_INF_B/='0' or S_A_EGAL_B/='0') then
    C:=C+1;
end if;

-- case6:
    CodeOp <= "101";
    entree_A <= "0101";
    entree_B <= "0110";
    wait for 10ns;
    assert (resultat ="1010")report "NOT Error!" Severity error;
    assert (S_A_SUP_B='0') report "SUP Error!" Severity error;
    assert (S_A_INF_B='1') report "INF Error!" Severity error;
    assert (S_A_EGAL_B='0') report "EGAL Error!" Severity error;
    if(resultat /="1010" or S_A_SUP_B/='0' or S_A_INF_B/='1' or S_A_EGAL_B/='0') then
    C:=C+1;
end if;




-- case7:
    CodeOp <= "110";
    entree_A <= "0101";
    entree_B <= "0100";
    wait for 10ns;
    assert (resultat ="1001")report "NOT Error!" Severity error;
    assert (S_A_SUP_B='1') report "SUP Error!" Severity error;
    assert (S_A_INF_B='0') report "INF Error!" Severity error;
    assert (S_A_EGAL_B='0') report "EGAL Error!" Severity error;
    if(resultat /="1001"or S_A_SUP_B/='1' or S_A_INF_B/='0' or S_A_EGAL_B/='0') then
    C:=C+1;
end if;
 -- case8:
    CodeOp <="111";
    entree_A <= "0101";
    entree_B <= "0101";
    wait for 10ns;
    assert (resultat ="0000")report "Soustraction Error!" Severity error;
    assert (S_A_SUP_B='0') report "SUP Error!" Severity error;
    assert (S_A_INF_B='0') report "INF Error!" Severity error;
    assert (S_A_EGAL_B='1') report "EGAL Error!" Severity error;
    if(resultat /="0000" or S_A_SUP_B/='0' or S_A_INF_B/='0' or S_A_EGAL_B/='1') then
    C:=C+1;
end if;
```

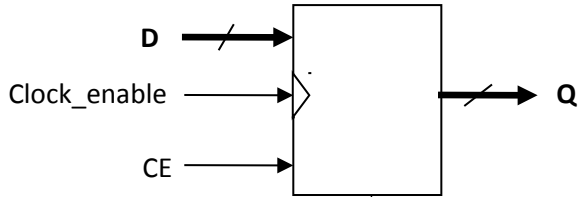
```
        -- summary of testbench
        if (C=0) then
            assert false
            report "Testbench of Unite completed successfully!"
            severity note;
            else
            assert true
            report " Something wrong,try again"
            severity error;
        end if;
        wait;
    end process;
end TB3;
```

La simulation donne les résultats suivants :



II. Conception des registres :

(Registre de mémorisation : Il s'agit en fait d'un ensemble de bascules D flip-flop formant un mot binaire de 4 bits)




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

entity REGISTRE is
port (clock_enable,CLR,CE :in std_logic;
      D :in std_logic_vector(3 downto 0);
      Q :out std_logic_vector(3 downto 0));
end REGISTRE;
architecture ARCH_REGISTRE of REGISTRE is
signal X :std_logic_vector(3 downto 0);
begin
process(clock_enable,CLR)
begin
if CLR='1' then X <= "0000";
elsif (clock_enable'event and clock_enable='1') then X <= D;
end if;
end process;
Q <= X when CE = '1' else "XXXX";
end ARCH_REGISTRE;
```

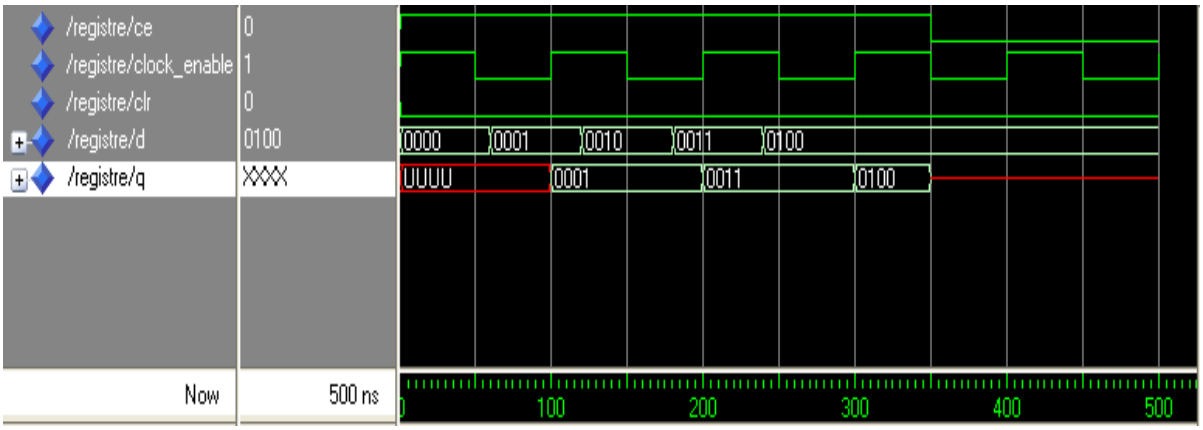
1. Fichier « Registre.vhd » :

Commentaire : Si CE=1,l'entrée de registre sera recopié dans la sortie ,si non la sortie est un état inconnu fort

- On tape les lignes de commande suivantes dans la fenêtre *transcript :*

force CE 1 0,0 350
force D 0000 0,0001 60,0010 120,0011 180,0100 240
force clr 0
run 500
pour clock_enable a une période de 100ns

- La simulation donne les résultats suivants :

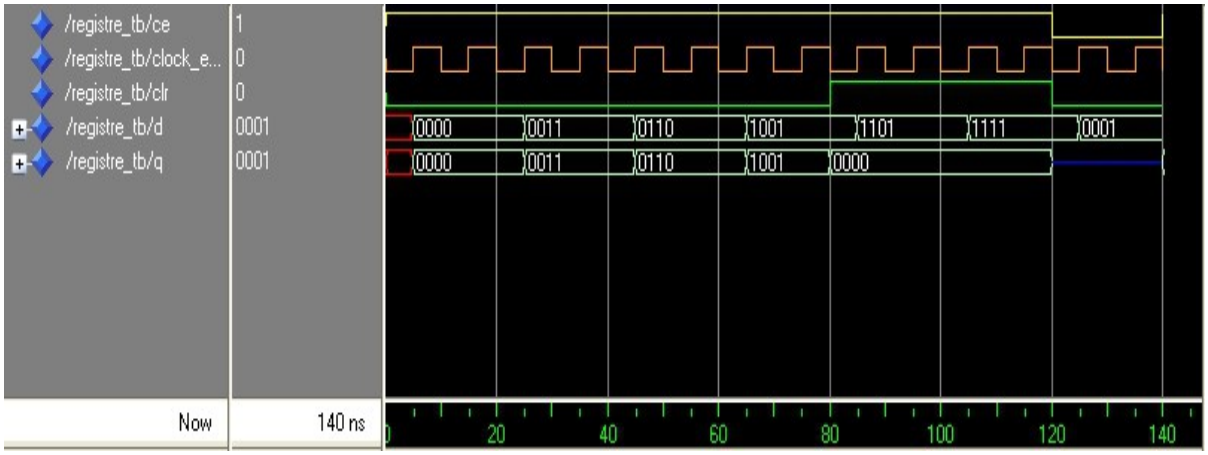



```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
entity REGISTRE_TB is
    end REGISTRE_TB;
    architecture TB4 of REGISTRE_TB is
    component REGISTRE
        port (clock_enable,CLR,CE :in std_logic;
              D :in std_logic_vector(3 downto 0);
              Q :out std_logic_vector(3 downto 0));
    end component;
    signal clock_enable,CLR,CE :std_logic;
    signal D :std_logic_vector(3 downto 0);
    signal Q :std_logic_vector(3 downto 0);
   begin
    R:REGISTRE port map (clock_enable,CLR,CE,D,Q);
    process
        begin
             clock_enable <= '0';
             wait for 5 ns;
             clock_enable <= '1';
             wait for 5 ns;
             end process;
    process
        begin
             CE <= '1';
             wait for 120 ns;
             CE <= '0';
             wait for 20 ns;
             end process;
```

2. Fichier « REGISTRE_TB » :

```
      process
          begin
               CLR <= '0';
               wait for 80 ns;
               CLR <= '1';
               wait for 40 ns;
           end process;
      process
           variable C:integer :=0;
          begin
```

```
    --case1
    wait for 5 ns;
    D <= "0000";
    wait for 15 ns;
    assert(Q = "0000")report "Error1!" severity Error;
    if (Q /= "0000")then
    C:=C+1;
end if;
--case2
wait for 5 ns;
    D <= "0011";
    wait for 15 ns;
    assert(Q = "0011")report "Error2!" severity Error;
    if (Q /= "0011")then
    C:=C+1;
end if;
--case3
    wait for 5 ns;
    D <= "0110";
    wait for 15 ns;
    assert(Q = "0110")report "Error3!" severity Error;
    if (Q /= "0110")then
    C:=C+1;
end if;
--case4
    wait for 5 ns;
    D <= "1001";
    wait for 15 ns;
    assert(Q = "1001")report "Error4!" severity Error;
    if (Q /= "1001")then
    C:=C+1;
end if;
```



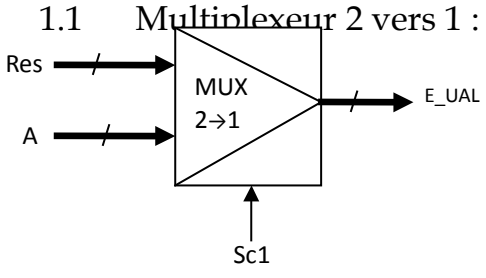

```
        --case5
             wait for 5 ns;
             D <= "1101";
             wait for 15 ns;
             assert(Q = "0000")report "Error5!" severity Error;
             if (Q /= "0000")then
             C:=C+1;
        end if;
        --case6
             wait for 5 ns;
             D <= "1111";
             wait for 15 ns;
             assert(Q = "0000")report "Error6!" severity Error;
             if (Q /= "0000")then
             C:=C+1;
        end if;
        --case7
             wait for 5 ns;
             D <= "0001";
             wait for 15 ns;
             assert(Q = "XXXX")report "Error5!" severity Error;
             if (Q /= "XXXX")then
             C:=C+1;
        end if;

            -- summary of testbench
           if (C=0) then
               assert false
               report "Testbench of REGISTRE_TB completed successfully!"
               severity note;
               else
               assert true
               report " Something wrong,try again"
               severity error;
           end if;
           wait;
      end process;
end TB4;
```

- La simulation donne les résultats suivants :

III. Conception des multiplexeurs et démultiplexeurs :

1. Conception des multiplexeurs :

1.1 Multiplexeur 2 vers 1 :



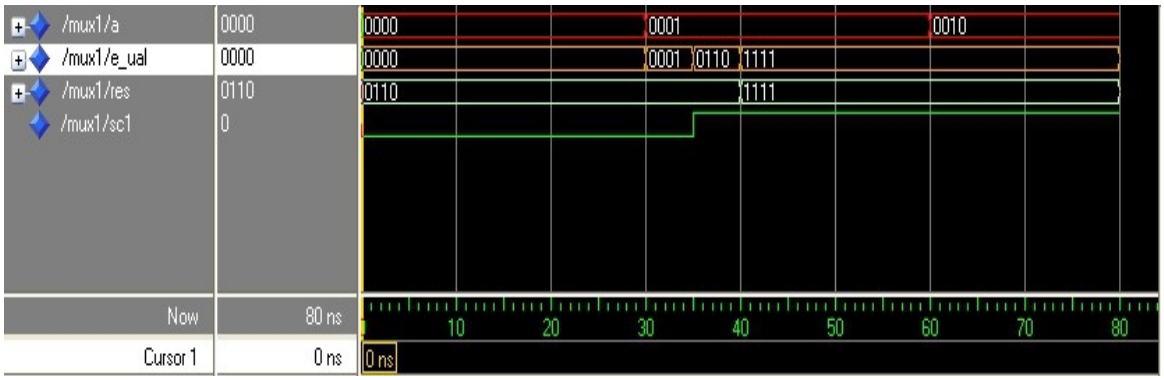
Sc1	E_UAL
O	Res
1	A

a. Fichier « MUX1.vhd » :

On tape les lignes de commande suivantes dans la fenêtre *transcript :*
force A 0000 0,0001 30,0010 60
force res 0110 0,1111 40,1010 80
force Sc1 0 0,1 35
run 80
La simulation donne les résultats suivants :

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
entity MUX1 is
port  ( A,Res:in std_logic_vector(3 downto 0);
        Sc1:in std_logic;
        E_UAL:out std_logic_vector(3 downto 0));
        end MUX1;
        architecture arch_MUX1 of MUX1 is
        begin
        E_UAL <= A when Sc1 ='0'else
                 Res;
                 end arch_MUX1;
```

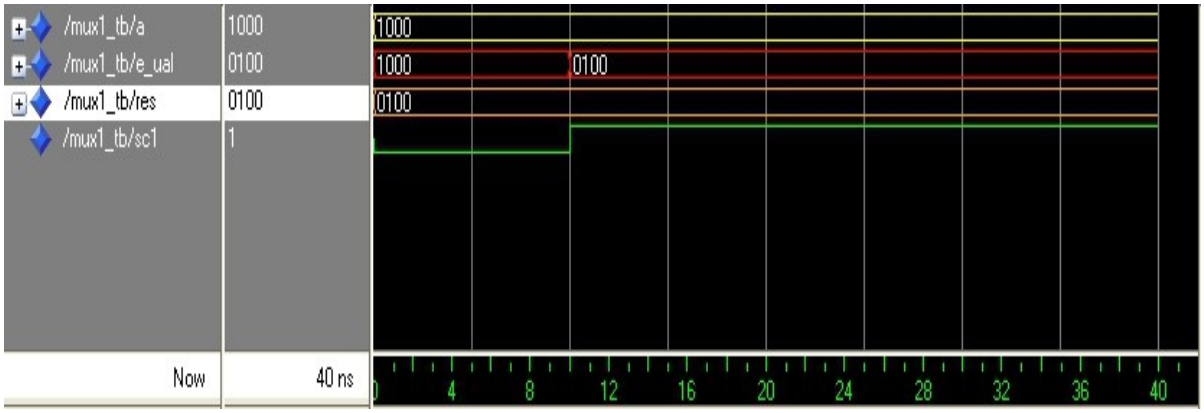
b. Fichier « MUX1_TB.vhd » :



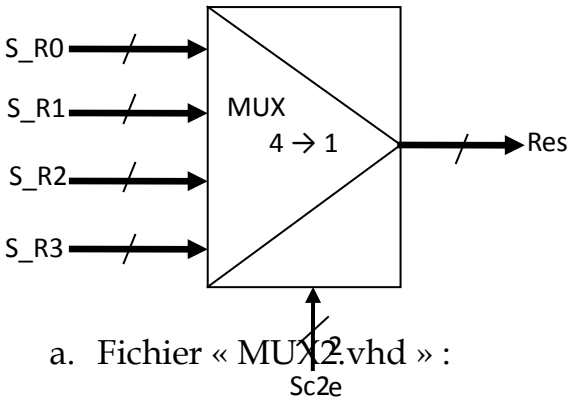

```
library ieee;
use ieee.std_logic_1164.all;

    entity MUX1_TB is
    end MUX1_TB;

    architecture TB5 of MUX1_TB is
    component MUX1
    port ( A,Res:in std_logic_vector(3 downto 0);
           Sc1:in std_logic;
           E_UAL:out std_logic_vector(3 downto 0));
       end component;
       signal A,Res:std_logic_vector(3 downto 0);
       signal Sc1:std_logic;
       signal E_UAL:std_logic_vector(3 downto 0);
       begin
        M1: MUX1 port map (A,Res,Sc1,E_UAL);
    process
    variable C:integer :=0;
        begin
            -- case1
            A <= "1000";
            Res <= "0100";
            Sc1 <= '0';
            wait for 10ns;
            assert (E_UAL = "1000") report "Error!" Severity error;
            if(E_UAL /= "1000") then
            C:=C+1;
        end if;
```

```
            -- case2
            A <= "1000";
            Res <= "0100";
            Sc1 <= '1';
            wait for 10ns;
            assert (E_UAL = "0100") report "Error!" Severity error;
            if(E_UAL /= "0100") then
            C:=C+1;
        end if;
        -- summary of testbench
        if (C=0) then
            assert false
            report "Testbench of MUX1 completed successfully!"
            severity note;
            else
            assert true
            report " Something wrong,try again"
            severity error;
        end if;
        wait;
    end process;
end TB5;
```

La simulation donne les résultats suivants :



1.2 Multiplexeur 4 vers 1



Sc2	Res
00	S_R0
01	S_R1
10	S_R2
11	S_R3

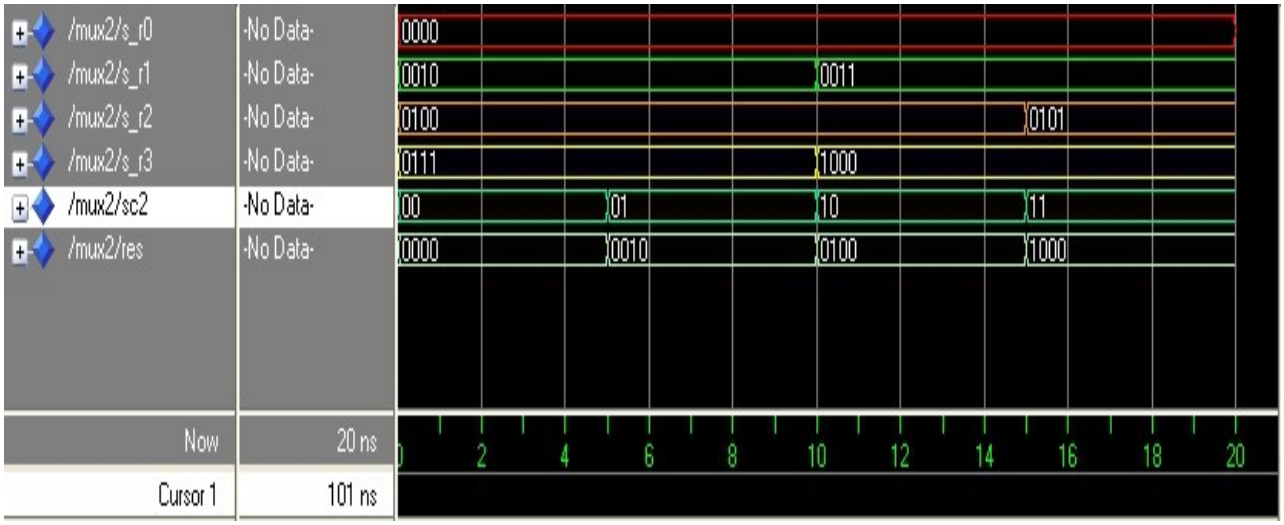
a. Fichier « MUX2.vhd » :

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
entity MUX2 is
port ( S_R0,S_R1,S_R2,S_R3:in std_logic_vector(3 downto 0);
       Sc2:in std_logic_vector(1 downto 0);
       Res:out std_logic_vector(3 downto 0));
       end MUX2;
 architecture arch_MUX2 of MUX2 is
       begin
       Res <= S_R0 when Sc2="00"else
              S_R1 when Sc2 ="01"else
              S_R2 when Sc2="10"else
              S_R3;
     end arch_MUX2;
```

- On tape les lignes de commande suivantes dans la fenêtre *transcript :*

force S_R0 0000 0,0001 20
force S_R1 0010 0,0011 10
force S_R2 0100 0,0101 15
force S_R3 0111 0,1000 10
force Sc2 00 0,01 5,10 10,11 15
run 20

- La simulation donne les résultats suivants :

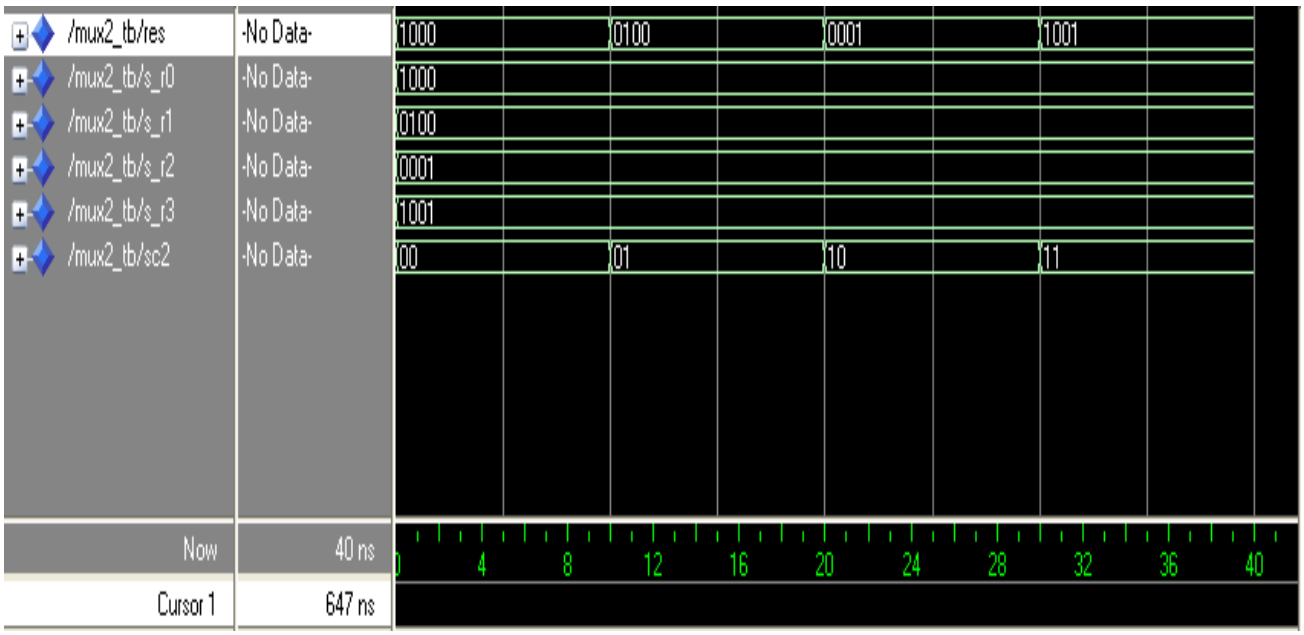


b. Fichier « MUX2_TB » :

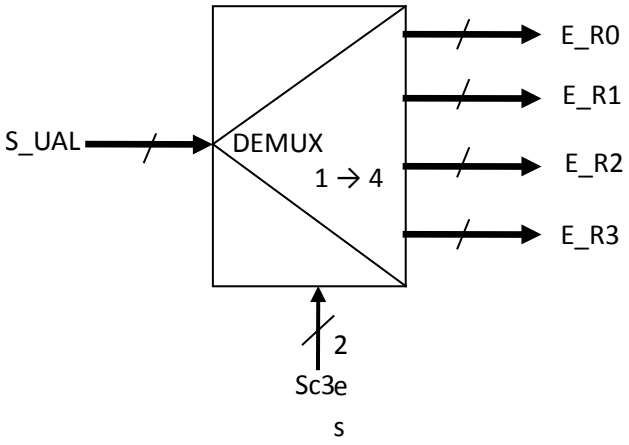
```
library ieee;
use ieee.std_logic_1164.all;

    entity MUX2_TB is
    end MUX2_TB;

    architecture TB6 of MUX2_TB is
    component MUX2
    port ( S_R0,S_R1,S_R2,S_R3:in std_logic_vector(3 downto 0);
           Sc2:in std_logic_vector(1 downto 0);
           Res:out std_logic_vector(3 downto 0));
       end component;
       Signal S_R0,S_R1,S_R2,S_R3:std_logic_vector(3 downto 0);
       Signal Sc2:std_logic_vector(1 downto 0);
       Signal Res:std_logic_vector(3 downto 0);
       begin
        M2: MUX2 port map (S_R0,S_R1,S_R2,S_R3,Sc2,Res);
    process
    variable C:integer :=0;
        begin
        -- case1
            S_R0 <= "1000";
            S_R1 <= "0100";
            S_R2 <= "0001";
            S_R3 <= "1001";
            Sc2 <= "00";
            wait for 10ns;
            assert (Res = "1000") report "Error!" Severity error;
            if(Res /= "1000") then
            C:=C+1;
        end if;


        -- case2
        S_R0 <= "1000";
        S_R1 <= "0100";
        S_R2 <= "0001";
        S_R3 <= "1001";
        Sc2 <= "01";
        wait for 10ns;
        assert (Res = "0100") report "Error!" Severity error;
        if(Res /= "0100") then
        C:=C+1;
    end if;
    -- case3
        S_R0 <= "1000";
        S_R1 <= "0100";
        S_R2 <= "0001";
        S_R3 <= "1001";
        Sc2 <= "10";
        wait for 10ns;
        assert (Res = "0001") report "Error!" Severity error;
        if(Res /= "0001") then
        C:=C+1;
    end if;
```

```
        --case4
            S_R0 <= "1000";
            S_R1 <= "0100";
            S_R2 <= "0001";
            S_R3 <= "1001";
            Sc2 <= "11";
            wait for 10ns;
            assert (Res = "1001") report "Error!" Severity error;
            if(Res /= "1001") then
            C:=C+1;
        end if;
        -- summary of testbench
        if (C=0) then
            assert false
            report "Testbench of MUX2 completed successfully!"
            severity note;
            else
            assert true
            report " Something wrong,try again"
            severity error;
        end if;
        wait;
    end process;
end TB6;
```

La simulation donne les résultats suivants :



2. Conception de démultiplexeur :



Sc3	E_R0	E_R1	E_R2	E_R3
00	1	0	0	0
01	0	1	0	0
10	0	0	1	0
11	0	0	0	1

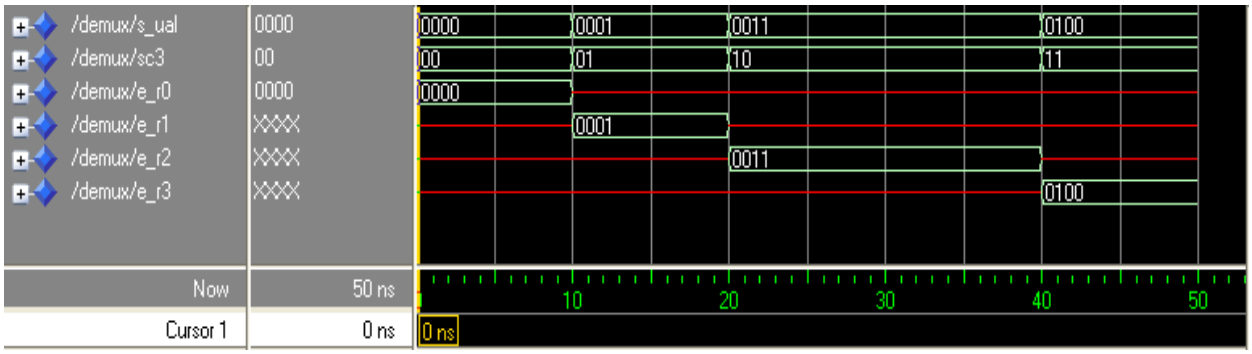
a. Fichier « DEMUX.vhd » :

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
entity DEMUX is
port ( S_UAL:in std_logic_vector(3 downto 0);
       Sc3:in std_logic_vector(1 downto 0);
       E_R0,E_R1,E_R2,E_R3:out std_logic_vector(3 downto 0));
       end DEMUX;
 architecture arch_DEMUX of DEMUX is
       begin
       E_R0 <= S_UAL when Sc3 ="00" else
               "XXXX";
       E_R1 <= S_UAL when Sc3 ="01" else
               "XXXX";
       E_R2 <= S_UAL when Sc3 ="10" else
               "XXXX";
       E_R3 <= S_UAL when Sc3 ="11" else
               "XXXX";
  end arch_DEMUX;
```

- On tape les lignes de commande suivantes dans le transcript :

force S_UAL 0000 0,0001 10,0011 20,0100 40
force Sc3 00 0,01 10,10 20,11 40
run 50

- La simulation donne les résultats suivants :

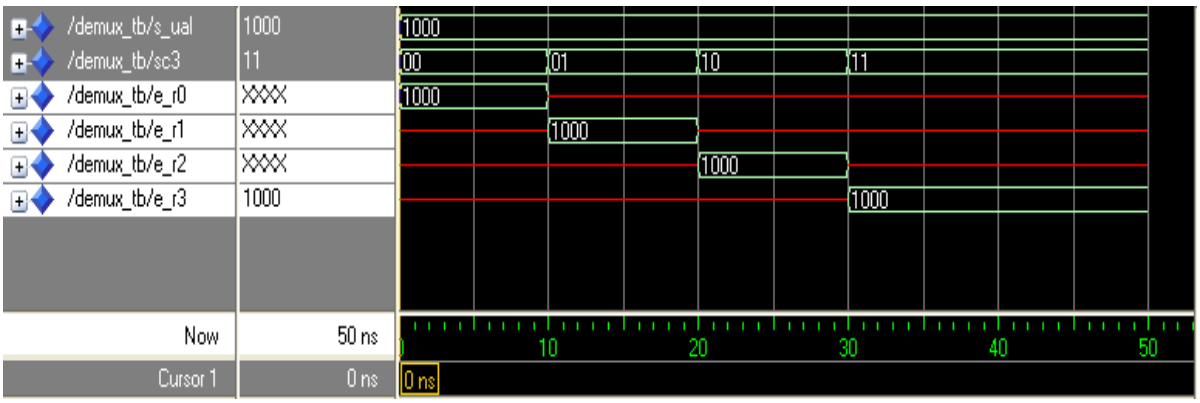


b. Fichier « DEMUX_TB » :

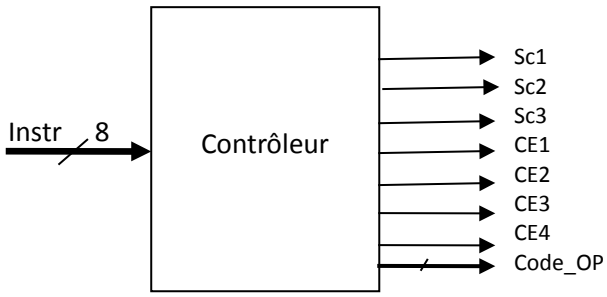
```
library ieee;
use ieee.std_logic_1164.all;

    entity DEMUX_TB is
    end DEMUX_TB;

    architecture TB7 of DEMUX_TB is
    component DEMUX
    port ( S_UAL:in std_logic_vector(3 downto 0);
           Sc3:in std_logic_vector(1 downto 0);
           E_R0,E_R1,E_R2,E_R3:out std_logic_vector(3 downto 0));
       end component;
       Signal S_UAL:std_logic_vector(3 downto 0);
       Signal Sc3:std_logic_vector(1 downto 0);
       Signal E_R0,E_R1,E_R2,E_R3:std_logic_vector(3 downto 0);
       begin
        DM: DEMUX port map (S_UAL,Sc3,E_R0,E_R1,E_R2,E_R3);
    process
    variable C:integer :=0;
        begin

        -- case1
        S_UAL <= "1000";
        Sc3 <= "00";
        wait for 10ns;
        assert (E_R0 = "1000") report "Error!" Severity error;
        assert (E_R1 = "XXXX") report "Error!" Severity error;
        assert (E_R2 = "XXXX") report "Error!" Severity error;
        assert (E_R3 = "XXXX") report "Error!" Severity error;
        if(E_R0 /= "1000" or E_R1 /= "XXXX" or E_R2 /= "XXXX" or E_R3 /= "XXXX") then
            C:=C+1;
        end if;
-- case2
S_UAL <= "1000";
Sc3 <= "01";
wait for 10ns;
assert (E_R0 = "XXXX") report "Error!" Severity error;
assert (E_R1 = "1000") report "Error!" Severity error;
assert (E_R2 = "XXXX") report "Error!" Severity error;
assert (E_R3 = "XXXX") report "Error!" Severity error;
if(E_R0 /= "XXXX" or E_R1 /= "1000" or E_R2 /= "XXXX" or E_R3 /= "XXXX") then
    C:=C+1;
end if;
-- case3
S_UAL <= "1000";
Sc3 <= "10";
wait for 10ns;
assert (E_R0 = "XXXX") report "Error!" Severity error;
assert (E_R1 = "XXXX") report "Error!" Severity error;
assert (E_R2 = "1000") report "Error!" Severity error;
assert (E_R3 = "XXXX") report "Error!" Severity error;
if(E_R0 /= "XXXX" or E_R1 /= "XXXX" or E_R2 /= "1000" or E_R3 /= "XXXX") then
    C:=C+1;
end if;
-- case4
S_UAL <= "1000";
Sc3 <= "11";
wait for 10ns;
assert (E_R0 = "XXXX") report "Error!" Severity error;
assert (E_R1 = "XXXX") report "Error!" Severity error;
assert (E_R2 = "XXXX") report "Error!" Severity error;
assert (E_R3 = "1000") report "Error!" Severity error;
if(E_R0 /= "XXXX" or E_R1 /= "XXXX" or E_R2 /= "XXXX" or E_R3 /= "1000") then
    C:=C+1;
end if;
```

```
        -- summary of testbench
        if (C=0) then
            assert false
            report "Testbench of DEMUX completed successfully!"
            severity note;
            else
            assert true
            report " Something wrong,try again"
            severity error;
        end if;
        wait;
    end process;
end TB7;
```

La simulation donne les résultats suivants :



IV. Conception du contrôleur :



- Le bit 3 de l'instruction permet de préciser si l'on souhaite effectuer une opération sur les entrées A et B ou sur B et une donnée provenant de l'un des registres => C'est le signal de sélection de multiplexeur 2 vers 1.
- Les bits 5 et 4 permettent de sélectionner le registre dans lequel une donnée servira à effectuer un calcul avec l'opérande B => C'est le signal de sélection de multiplexeur 4 vers 1.
- Les bits 7 et 6 permettent de sélectionner le registre dans le quel le résultat de l'UAL sera stocké => C'est le signal de sélection du démultiplexeur 1 vers 4.

Fichier « CONTROL1.vhd » :

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
entity control1 is
port( Instr:in std_logic_vector(7 downto 0);
      Sc1:out std_logic;
      Sc2,Sc3:out std_logic_vector(1 downto 0);
      Code_OP:out std_logic_vector(2 downto 0);
      CE0,CE1,CE2,CE3:out std_logic);
      end control1;
      architecture arch_control1 of control1 is
          signal S:std_logic_vector(1 downto 0);
      begin
          Code_OP <= Instr(2)& Instr(1)& Instr(0);
          Sc1 <= Instr(3);
          Sc2 <= Instr(5)& Instr(4);
          Sc3 <= Instr(7)& Instr(6);
          S <= Instr(7)& Instr(6);
          CE0 <= '1' when S="00" else
                 '0';
          CE1 <= '1' when S="01" else
                 '0';
          CE2 <= '1' when S="10" else
                 '0';
          CE3 <= '1' when S="11" else
                 '0';
    end arch_control1;
```

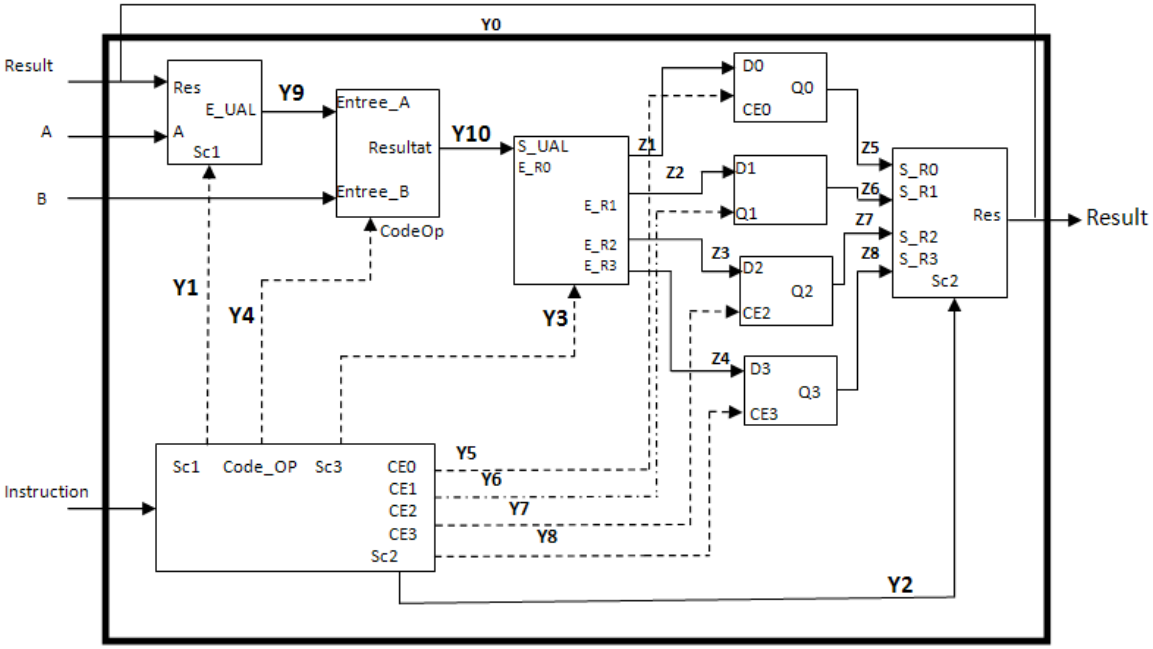
- On tape les lignes de commande dans le transcript :

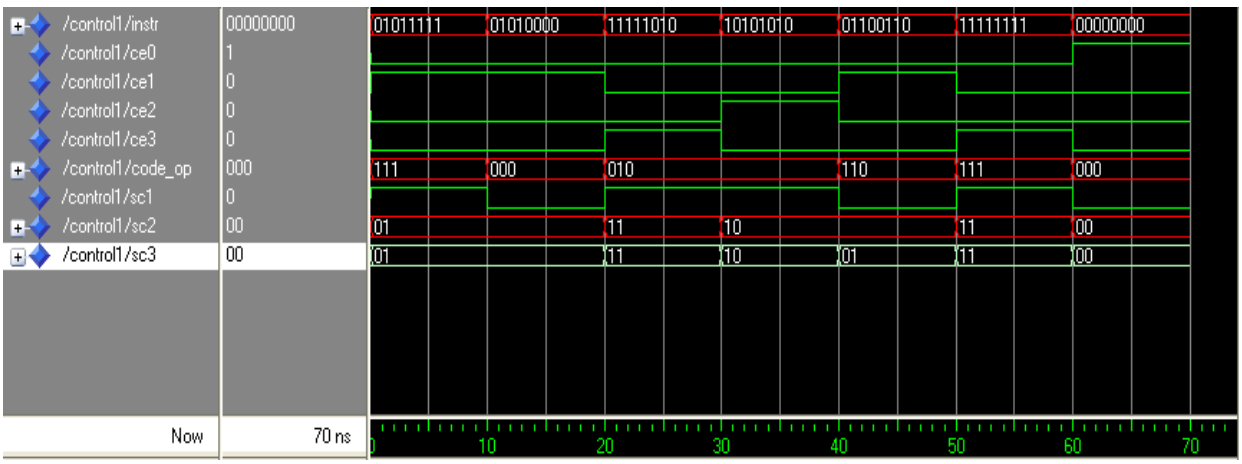
force Instr 01011111 0,01010000 10,11111010 20,10101010 30,01100110 40,11111111 50,00000000 60

run 70

- La simulation donne les résultats suivants :

V. Conception final du mini-calculateur :





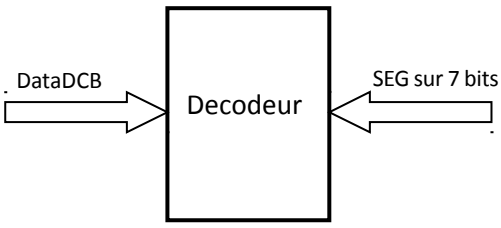
- Fichier « mini_calcul.vhd » :

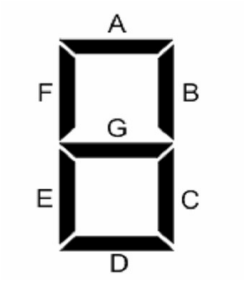
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
entity MINI_CALCUL is
port ( A,B:in std_logic_vector(3 downto 0);
       Instruction:in std_logic_vector(7 downto 0);
       Clock,CLR:in std_logic;
       Result:inout std_logic_vector(3 downto 0);
       A_SUP_B,A_INF_B,A_EGAL_B,NEG,ZERO,OVFL:out std_logic);
       end MINI_CALCUL;
architecture arch_mini_calcul of MINI_CALCUL is
 -- Multiplexeur 2 vers 1
 component MUX1
 port( A,Res:in std_logic_vector(3 downto 0);
       Sc1:in std_logic;
       E_UAL:out std_logic_vector(3 downto 0));
       end component;
 -- UAL
   component Unite
   port ( entree_A,entree_B: in std_logic_vector(3 downto 0);
          CodeOP:in std_logic_vector(2 downto 0);
          resultat:out std_logic_vector(3 downto 0);
          S_A_SUP_B,S_A_INF_B,S_A_EGAL_B,S_NEG,S_ZERO,S_OVFL:out std_logic);
   end component;
 -- Demultiplexeur
 component DEMUX
 port ( S_UAL:in std_logic_vector(3 downto 0);
        Sc3:in std_logic_vector(1 downto 0);
        E_R0,E_R1,E_R2,E_R3:out std_logic_vector(3 downto 0));
       end component;
```

```
 --Registre
 component REGISTRE
 port (clock_enable,CLR,CE :in std_logic;
       D :in std_logic_vector(3 downto 0);
       Q :out std_logic_vector(3 downto 0));
end component;
-- Multiplexeur 4 vers 1
component MUX2
port ( S_R0,S_R1,S_R2,S_R3:in std_logic_vector(3 downto 0);
       Sc2:in std_logic_vector(1 downto 0);
       Res:out std_logic_vector(3 downto 0));
       end component;
--Controleur
component control1
port( Instr:in std_logic_vector(7 downto 0);
      Sc1:out std_logic;
      Sc2,Sc3:out std_logic_vector(1 downto 0);
      Code_OP:out std_logic_vector(2 downto 0);
      CE0,CE1,CE2,CE3:out std_logic);
      end component;
      Signal Y1,Y5,Y6,Y7,Y8:std_logic;
      Signal Y2,Y3:Std_logic_vector(1 downto 0);
      Signal Y4:Std_logic_vector(2 downto 0);
      Signal Y0,Y9,Y10,Z1,Z2,Z3,Z4,Z5,Z6,Z7,Z8:Std_logic_vector(3 downto 0);


begin
    C1:control1 port map (Instruction,Y1,Y2,Y3,Y4,Y5,Y6,Y7,Y8);
    C2:MUX1 port map(A,Result,Y1,Y9);
    C3:Unite port map (Y9,B,Y4,Y10,A_SUP_B,A_INF_B,A_EGAL_B,NEG,ZERO,OVFL);
    C4:DEMUX port map (Y10,Y3,Z1,Z2,Z3,Z4);
    C5:REGISTRE port map(Clock,CLR,Y5,Z1,Z5);
    C6:REGISTRE port map(Clock,CLR,Y6,Z2,Z6);
    C7:REGISTRE port map(Clock,CLR,Y7,Z3,Z7);
    C8:REGISTRE port map(Clock,CLR,Y8,Z4,Z8);
    C9:MUX2 port map (Z5,Z6,Z7,Z8,Y2,Y0);
    Result <= Y0 ;
end arch_mini_calcul;
```

Exercices en VHDL

I. Modélisation d'un décodeur 7 segments :





1. Table de vérité :

E1	E2	E3	E4		SEG_A	SEG_B	SEG_C	SEG_D	SEG_E	SEG_F	SEG_G
0	0	0	0		1	1	1	1	1	1	0
0	0	0	1		0	1	1	0	0	0	0
0	0	1	0		1	1	0	1	1	0	1
0	0	1	1		1	1	1	1	0	0	1
0	1	0	0		0	1	1	0	0	1	1
0	1	0	1		1	0	1	1	0	1	1
0	1	1	0		1	0	1	1	1	1	1
0	1	1	1		1	1	1	0	0	0	0
1	0	0	0		1	1	1	1	1	1	1
1	0	0	1		1	1	1	1	0	1	1

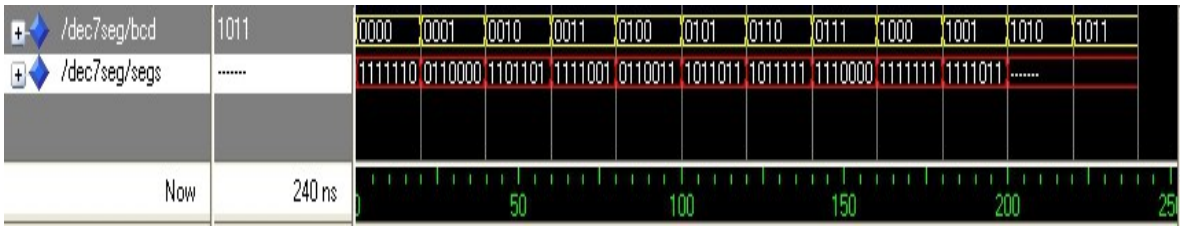
2. Fichier « dec7seg.vhd »

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
entity dec7seg is
port( BCD: in std_logic_vector (3 downto 0);
      SEGS: out std_logic_vector(6 downto 0));
      end dec7seg;
architecture bhv of dec7seg is
      begin
      with BCD select
           SEGS <= "1111110" when "0000",
                   "0110000" when "0001",
                   "1101101" when "0010",
                   "1111001" when "0011",
                   "0110011" when "0100",
                   "1011011" when "0101",
                   "1011111" when "0110",
                   "1110000" when "0111",
                   "1111111" when "1000",
                   "1111011" when "1001",
                   "-------" when others;
    end bhv;
```

On tape les lignes de commande suivantes dans la fenêtre *transcript :*

- force bcd 0000 0,0001 20,0010 40,0011 60,0100 80,0101 100,0110 120,0111 140,1000 160,1001 180,1010 200,1011 220
- run 240

La simulation donne les résultats suivants :



3. Fichier « dec7seg _TB.vhd »

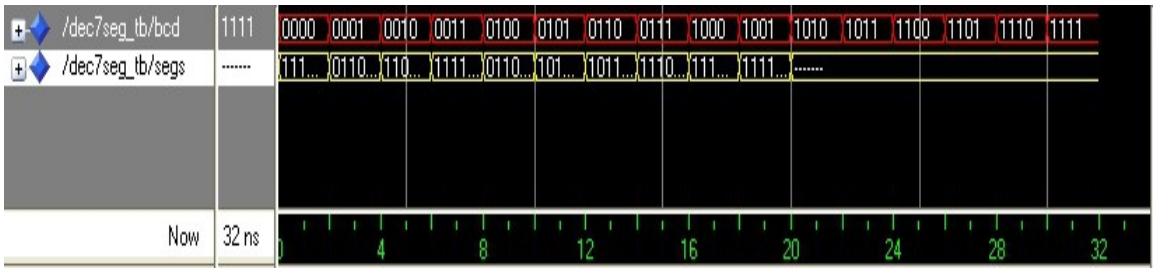
```
library ieee;
use ieee.std_logic_1164.all;

    entity dec7seg_TB is
    end dec7seg_TB;

    architecture TB2 of dec7seg_TB is
    component dec7seg
    port( BCD: in std_logic_vector (3 downto 0);
          SEGS: out std_logic_vector(6 downto 0));
      end component;
      signal BCD:std_logic_vector (3 downto 0);
      signal SEGS:std_logic_vector(6 downto 0);
      begin
        dec:dec7seg port map (BCD,SEGS);
          process
              variable C:integer:=0;
              begin
                  --case 1
                  BCD <= "0000";
                  wait for 2 ns;
                  assert(SEGS = "1111110")
                  report "SEGS error!" severity error;
                 if ( SEGS /= "1111110") then
                  C:= C +1;
                 end if;
                  --case 2
                  BCD <= "0001";
                  wait for 2 ns;
                  assert(SEGS = "0110000")
                  report "SEGS error!" severity error;
                 if ( SEGS /= "0110000") then
                  C:= C +1;
                 end if;
```

```
 --case 3
 BCD <= "0010";
 wait for 2 ns;
 assert(SEGS = "1101101")
 report "SEGS error!" severity error;
if ( SEGS /= "1101101") then
 C:= C +1;
end if;
 --case 4
 BCD <= "0011";
 wait for 2 ns;
 assert(SEGS = "1111001")
 report "SEGS error!" severity error;
if ( SEGS /= "1111001") then
     C:= C +1;
end if;
 --case 5
 BCD <= "0100";
 wait for 2 ns;
 assert(SEGS = "0110011")
 report "SEGS error!" severity error;
    if ( SEGS /= "0110011") then
     C:= C +1;
 end if;
 --case 6
 BCD <= "0101";
 wait for 2 ns;
 assert(SEGS = "1011011")
 report "SEGS error!" severity error;
if ( SEGS /= "1011011") then
     C:= C +1;
end if;
 --case 7
 BCD <= "0110";
 wait for 2 ns;
 assert(SEGS = "1011111")
 report "SEGS error!" severity error;
  if ( SEGS /= "1011111") then
     C:= C +1;
  end if;
 --case 8
 BCD <= "0111";
 wait for 2 ns;
 assert(SEGS = "1110000")
 report "SEGS error!" severity error;
    if ( SEGS /= "1110000") then
     C:= C +1;
    end if;
 --case 9
 BCD <= "1000";
 wait for 2 ns;
 assert(SEGS = "1111111")
 report "SEGS error!" severity error;
    if ( SEGS /= "1111111") then
```

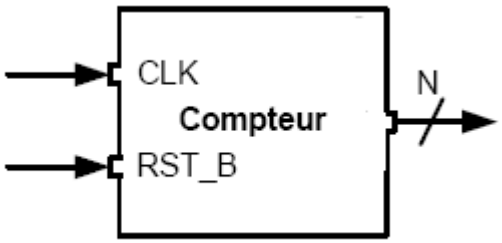
```
C:= C +1;
end if;
--case 10
BCD <= "1001";
wait for 2 ns;
assert(SEGS = "1111011")
report "SEGS error!" severity error;
   if ( SEGS /= "1111011") then
    C:= C +1;
   End if;
-- lorsque l'entree BCD dépasse 9
--case 11
BCD <= "1010";
wait for 2 ns;
assert(SEGS = "-------")
report "SEGS error!" severity error;
    if ( SEGS /= "-------") then
    C:= C +1;
    end if;
--case 12
BCD <= "1011";
wait for 2 ns;
assert(SEGS = "-------")
report "SEGS error!" severity error;
    if ( SEGS /= "-------") then
    C:= C +1;
    end if;
--case 13
BCD <= "1100";
wait for 2 ns;
assert(SEGS = "-------")
report "SEGS error!" severity error;
   if ( SEGS /= "-------") then
    C:= C +1;
    end if;
--case 14
BCD <= "1101";
wait for 2 ns;
assert(SEGS = "-------")
report "SEGS error!" severity error;
   if ( SEGS /= "-------") then
    C:= C +1;
   end if;
--case 15
BCD <= "1110";
wait for 2 ns;
assert(SEGS = "-------")
report "SEGS error!" severity error;
   if ( SEGS /= "-------") then
    C:= C +1;
end if;
--case 16
BCD <= "1111";
wait for 2 ns;
assert(SEGS = "-------")
report "SEGS error!" severity error;
```

```
                if ( SEGS /= "-------") then
                   C:= C +1;
                end if;
               -- summury of test bench
               if (C=0) then
               assert false
               report"Testbench of compara_4b completed successfully!"
               severity note;
               else
               assert true
               report " Something wrong,try again"
               severity error;
               end if;
          wait;
    end process;
end TB2;
```

La simulation du test bench du décodeur 7 segments donne les résultats suivants :



II. Modélisation d'un compteur synchrone :



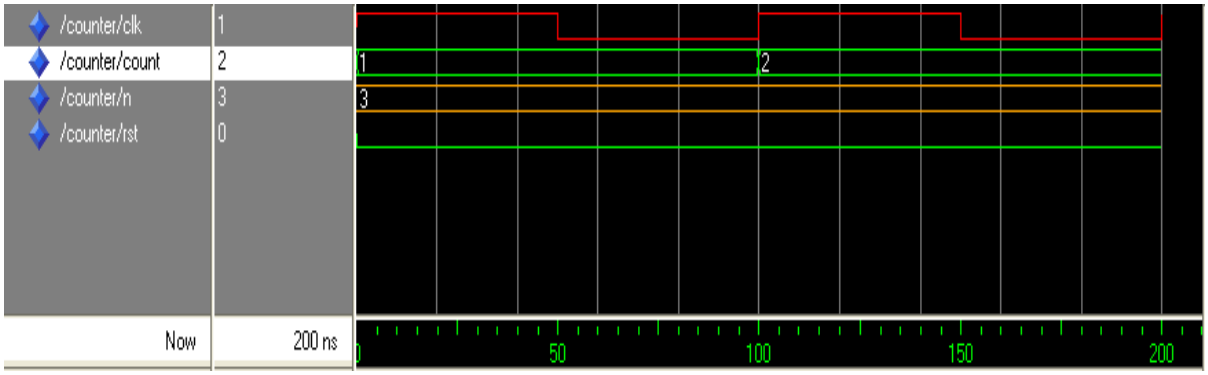
1. Fichier « compteur.vhd » :

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


entity counter is

generic(n: natural := 3);
port( clk: in std_logic;
      rst: in std_logic;
      count: out integer range 0 to n-1);
end counter;

architecture behv of counter is



begin
    process(clk,rst)
        variable X :integer range 0 to n-1;
    begin
    if (rst='1')then X:=0;
    elsif(clk='1' and clk'event) then
      if X > n-1 then X := 0;
      else X := X + 1;
       end if;
       end if;
       count <= X;
    end process;
end behv;
```

- On tape les lignes de commande dans le transcript :

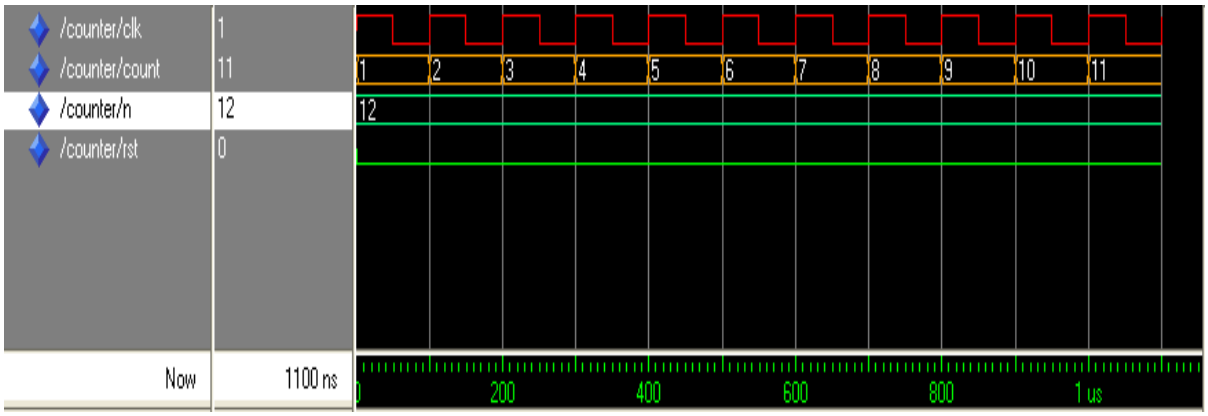
Force clk 1 0, 0 {50 ns} -r 100
Force rst 0 0

- La simulation donne les résultats suivants :



On change la valeur du N, et on lui donne cette fois la valeur 12 ,(*generic(n: natural := 12);*)

Le résultat de la simulation est le suivant :

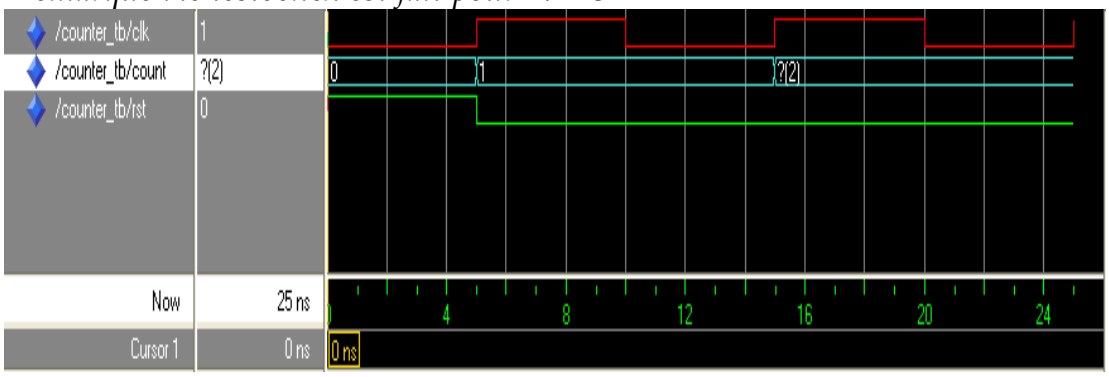


2. Fichier « compteur_TB.vhd » :

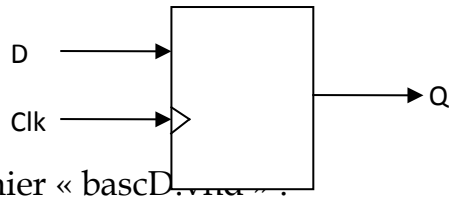
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity counter_TB is
end counter_TB;

architecture TB8 of counter_TB is

    component counter
    port(clk: in std_logic;
         rst: in std_logic;
       count: out integer range 0 to 1);
end component;

Signal clk:std_logic;
Signal rst:std_logic;
Signal count: integer range 0 to 1;
begin
```

```
    U_counter: counter port map (clk,rst,count);
    process
    begin
    clk <= '0';
     wait for 5 ns;
    clk <= '1';
     wait for 5 ns;
    end process;

    process

   variable C: integer :=0;

    begin
    rst <= '1';
    wait for 5 ns;
    rst <= '0';
    -- test case 1
    wait for 5 ns;
    assert (count =1) report "Error1" severity error;
    if (count/=1) then
       C:= C+1;
   end if;
   -- test case 2
    wait for 10 ns;
    assert (count = 2) report "Error2" severity error;
    if (count/=2) then
       C:= C+1;
   end if;
    -- summary of testbench
         if (C=0) then
             assert false
             report "Testbench of counter completed successfully!"
             severity note;
             else
             assert true
             report " Something wrong,try again"
             severity error;
         end if;
         wait;
     end process;
end TB8;
```

Remarque : le testbench est fait pour N = 3



III. Modélisation d'une bascule D flip flop:



1. Fichier « bascD.vhd » :

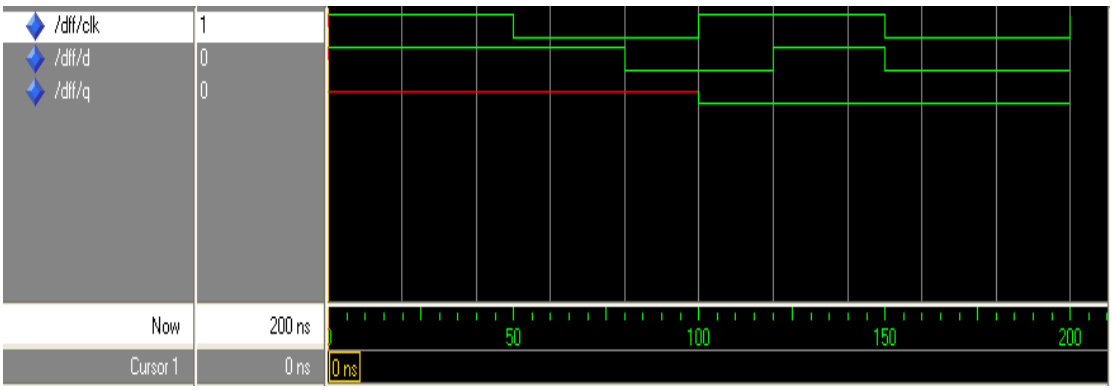
```
library ieee ;
use ieee.std_logic_1164.all;

entity dff is
port ( D: in std_logic;
       clk:  in std_logic;
       Q: out std_logic);
end dff;
architecture behv of dff is
begin

    process(D, clk)
    begin
   if (clk='1' and clk'event) then
       Q <= D;
   end if;
    end process;

end behv;
```

- On tape les lignes de commande suivante dans le transcript :

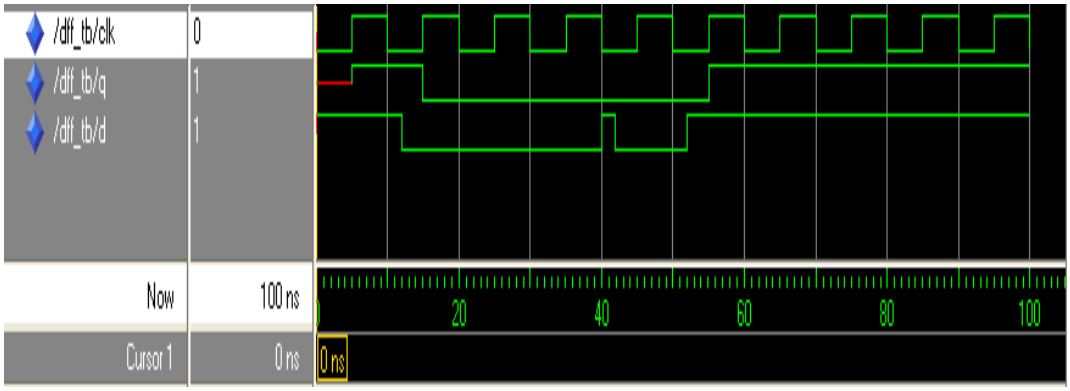
Force clk 1 0, 0 {50 ns} -r 100
force D 1 0, 0 80,1 120,0 150
run 200

- La simulation donne les résultats suivants:

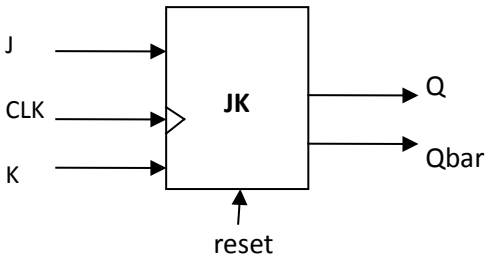
2. Fichier « bascD_TB.vhd » :



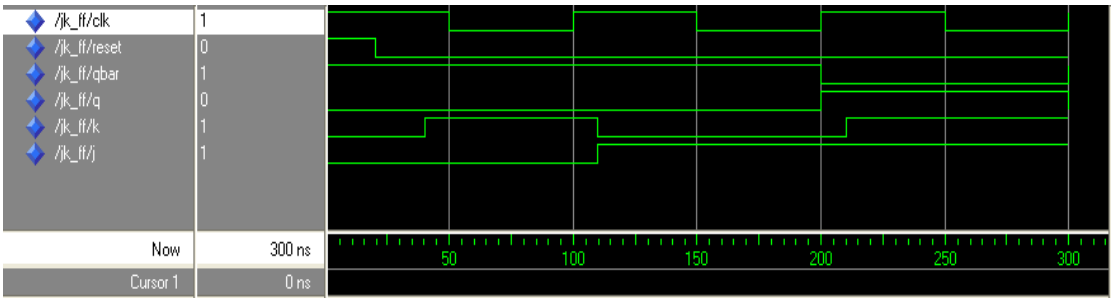
⇨ Simulation :



IV. Modélisation d'une bascule JK:



1. Fichier « JK.vhd » :




```
    end process;

    Q <= state;
    Qbar <= not state;

end behv;
```

- On tape les lignes de commande suivante dans le transcript :

Force clk 1 0, 0 {50 ns} -r 100
force reset 1 0,0 20
force j 0 0,1 110
force k 0 0,1 40,0 110,1 210
run 300

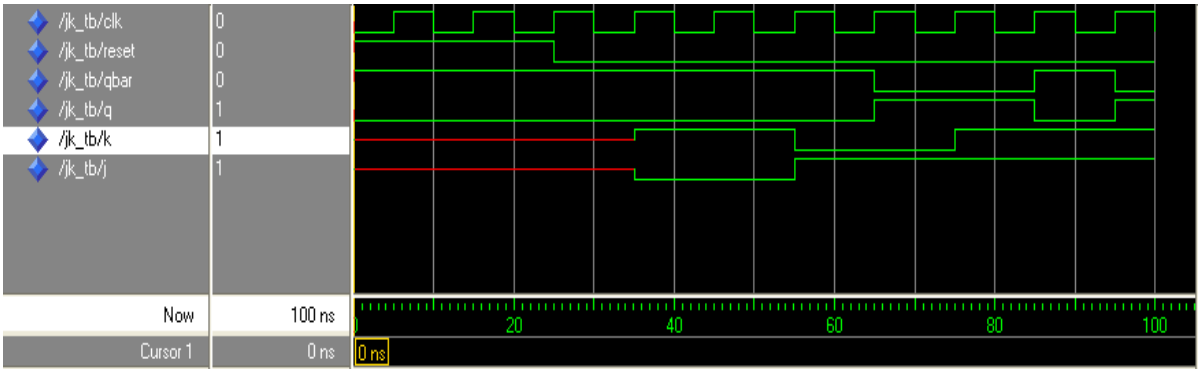
- La simulation donne les résultats suivants :

2. Fichier « JK_TB.vhd » :

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

entity jk_TB is
end jk_TB;


architecture TB10 of jk_TB is

    signal J, K:  std_logic;
    signal clk: std_logic;
    signal reset: std_logic;
    signal Q, Qbar: std_logic;

    component JK_FF is
    port ( clk:  in std_logic;
           J, K:  in std_logic;
         reset:  in std_logic;
       Q, Qbar: out std_logic);
    end component;

begin

    C: JK_FF port map (clk, J, K,reset,Q,Qbar);
    process
    begin
   clk <= '0';
   wait for 5 ns;
   clk <= '1';
   wait for 5 ns;
    end process;
```

```
    process
    variable cnt: integer := 0;
    begin

      reset <= '1';
    wait for 25 ns;

    reset <= '0';
    wait for 10 ns;

    -- case 1
    J <= '0';
    K <= '1';
    wait for 15 ns;
    assert (Q='0') report "Error1!" severity error;
    if (Q/='0') then
        cnt := cnt + 1;
    end if;

    -- case 2
    wait for 5 ns;
    J <= '1';
    K <= '0';
    wait for 15 ns;
    assert (Q='1') report "Error2!" severity error;
    if (Q/='0') then
        cnt := cnt + 1;
    end if;

   wait for 15 ns;
   assert (Q='1') report "Error2!" severity error;
   if (Q/='0') then
       cnt := cnt + 1;
   end if;

   -- case 3
   wait for 5 ns;
   J <= '1';
   K <= '1';
   wait for 15 ns;
   assert (Q='0') report "Error3!" severity error;
   if (Q/='0') then
       cnt := cnt + 1;
   end if;

   -- summary of all the tests
   if (cnt=0) then
       assert false
       report "Testbench of Bascule JK completed successfully!"
       severity note;
   else
       assert true
       report "Something wrong, try again"
       severity error;
   end if;

   wait;

    end process;

end TB10;
```

⇨ Simulation :


/jk_tb/clk
/jk_tb/reset
/jk_tb/qbar
/jk_tb/q
/jk_tb/k
/jk_tb/j
0
0
0
1
1
1
Now
100 ns
Cursor 1
0 ns
0 ns
20
40
60
80
100